Contents

Preface

Audience for This Guide .......................................................... xlv
The Purpose of This Guide ........................................................ xlv
Conventions .............................................................................. xlv
If You Need Help ..................................................................... xlv
Comments on the Documentation ............................................. xlvi

1 List of functions by category

Analytics - Association Rules .................................................. 47
Analytics - Classification ....................................................... 47
Analytics - Clustering ............................................................. 48
Analytics - Column Properties ............................................... 48
Analytics - Data Transformation ............................................ 48
Analytics - Diagnostic Measures ........................................... 48
Analytics - Discretization ...................................................... 49
Analytics - Model Management ............................................. 49
Analytics - Probability Distributions ..................................... 50
Analytics - Quantiles & Outliers .......................................... 53
Analytics - Regression .......................................................... 53
Analytics - Sampling ............................................................. 54
Analytics - Statistics .............................................................. 54
Analytics - Time Series ........................................................ 56
Utilities - Actions ................................................................... 56
Utilities - Checking ............................................................... 56
Utilities - Data Exploration .................................................. 57
Utilities - Preprocessing ....................................................... 57

2 Reference Documentation: Analytics

ACC - Classification accuracy .................................................... 59
ALTER_MODEL - Alter properties of an analytics model ............... 61
ANOVA_CRD_TEST - Analyze variance using Completely Randomized Design and One Way ANOVA ........................................ 64
ANOVA_RBD_TEST - Analyze variance using ANOVA Randomized Block Design .................................................. 70
APPLY_DISC - Discretize data using given discretization bin limits .............................................................. 76
ARULE - Build an Association Rules model .................................................................81
BITABLE - Bivariate Frequencies ..............................................................................85
BTBNET_GROW - Build a tree-like Bayesian Network model .....................................87
CANONICAL_CORR - Canonical Correlation ...............................................................90
CANONICAL_CORR_AGG - Canonical Correlation .....................................................94
CBFS_AGG - Correlation-based feature selection .....................................................96
CERROR - Classification Error ..................................................................................98
CHISQ_TEST - Pearson's Chi-square test of independence ......................................100
CHISQ_TEST_AGG - Pearson's Chi-square ...............................................................103
CHISQ_TEST_S_AGG - Pearson's Chi-square with degree of freedom ..................104
CLEANUP - Remove the model management infrastructure from the current database 106
CMATRIX_ACC - Classification accuracy from a Confusion Matrix ..........................107
CMATRIX_STATS - Print Classification quality factors from a Confusion Matrix ........109
CMATRIX_WACC - Weighted Classification accuracy from a Confusion Matrix ......112
COL2TRCV_MANOVA_ONE_WAY_TEST - data transformation from column representation to trcv for Multivariate Analyzsis of Variance in one way setting ........................114
COL2TRCV_MANOVA_TWO_WAY_TEST - data transformation from column representation to grcv for Multivariate Analysis of Variance in two way setting ........................122
COLUMN_PROPERTIES - Create a column properties table ......................................130
COLUMN_PROPERTIES_CHECK - Check a column properties table for correctness ......133
COND_ENTROPY - Conditional Entropy ....................................................................134
CONFUSION_MATRIX - Build a Confusion Matrix ....................................................136
COPY_MODEL - Duplicate an analytics model ............................................................139
CORR - Mutual Correlation ......................................................................................141
CORR_AGG - Pearson's Correlation .........................................................................143
CORR_AGG - Pearson's Correlation with weight ......................................................144
CORR_MATRIX_AGG - Correlation Matrix ................................................................145
CORRELATION1000MATRIX - Correlation matrix ....................................................147
CORRELATION500PAIRS - Correlation of pairs of variables ......................................151
COV - Covariance ....................................................................................................154
COV_AGG - Pearson's Covariance ..........................................................................156
COV_MATRIX_AGG - Covariance Matrix ..................................................................157
COVARIANCE1000MATRIX - Covariance matrix .....................................................159
COVARIANCE500PAIRS - Covariance of pairs of variables ........................................163
COVARIANCEMATRIX - Covariance matrix as string .............................................166
CROSS_VALIDATION - Build and cross validate a Classification model ..................169
CUMULATIVE - Cumulative Distributions ...............................................................171
DBERN - Density of the Bernoulli Distribution .........................................................177
DBETA - Probability density function for Beta distribution ......................................178
DBINOM - Density of the Binomial Distribution .......................................................179
DCAUCHY - Probability density function for Cauchy Distribution .................................................. 181
DCHISQ - Probability density function for Chi-square Distribution .............................................. 182
DECTREE - Build then prune a Decision Tree model ...................................................................... 183
DENSITY - Density of Distributions .................................................................................................. 188
DEXP - Probability density function for Exponential Distribution ................................................ 191
DF - Probability density function for Fisher Distribution ............................................................ 192
DFISK - Probability density function for Fisk (or log-logistic) Distribution .................................... 193
DGAMMA - Probability density function for Gamma distribution .................................................. 195
DGEOM - Density of the Geometric Distribution ............................................................................. 196
DHYPER - Density of the Hypergeometric Distribution ................................................................... 197
DIVCLUSTER - Build a Hierarchical Clustering model .................................................................... 199
DLNORM - Density of Galton (or LogNormal) Distribution ............................................................. 204
DLOGIS - Probability density function for Logistic Distribution .................................................... 206
DMWW - Density of the Mann-Whitney-Wilcoxon Distribution ...................................................... 207
DNBINOM - Density of the Negative Binomial Distribution ............................................................ 208
DNORM - Density of the Standard Normal Distribution .................................................................... 210
DNORM3P - Density of the Normal Distribution ............................................................................. 211
DPOIS - Density of the Poisson Distribution ..................................................................................... 213
DROP_ALL_MODELS - Drop selected or all analytics models .......................................................... 214
DROP_MODEL - Drop an analytics model ......................................................................................... 215
DROP_SUMMARY1000 - Drop tables created by SUMMARY1000 .................................................. 217
DT - Density of the T-student Distribution ...................................................................................... 218
DUNIF - Density of the Uniform Distribution .................................................................................... 219
DWALD - Density of the Wald distribution ....................................................................................... 220
DWEIBULL - Density of the Weibull (or Rosin-Rammer) Distribution ........................................... 221
DWILCOX - Density of the Wilcoxon Distribution .......................................................................... 222
EFDISC - Discretization bins of equal frequency ............................................................................. 223
EMDISC - Discretization bins of minimal entropy ............................................................................. 228
ENTROPY - Univariate Entropy .......................................................................................................... 231
EWDISC - Discretization bins of equal width ................................................................................. 233
EWDISC_NICE - Discretization bins of equal width with human-friendly limits ............................. 236
EXPORT_MODEL - Export one or more analytic models to a file set ................................................. 240
EXPORT_PMML - Export an analytics model as PMML document to a file .................................... 242
FMEASURE - F-Measure from a Confusion Matrix ........................................................................ 244
FPR - False Positive Rate from a Confusion Matrix ........................................................................ 246
GET_COLUMN_LIST - Retrieve a list of columns with a given role and type from a column properties table ..................................................................................................................................... 249
GLM - Build a Generalized Linear Model ......................................................................................... 250
GRANT_MODEL - Grant privileges on an analytics model ............................................................... 254
GROW_DECTREE - Build a Decision Tree model ............................................................................. 257
<p>| <strong>GROW_REGTREE</strong> - Grow a Regression Tree model | 260 |
| <strong>HIST</strong> - Histograms | 264 |
| <strong>IMPORT_MODEL</strong> - Import one or more analytics models from a file set | 270 |
| <strong>IMPUTE_DATA</strong> - Impute missing data | 273 |
| <strong>INITIALIZE</strong> - Initialize the model management infrastructure | 275 |
| <strong>IQR</strong> - InterQuartile Range | 276 |
| <strong>IS_INITIALIZED</strong> - Check the model management infrastructure | 277 |
| <strong>JOINT_ENTROPY</strong> - Bivariate Entropy | 278 |
| <strong>KMEANS</strong> - Build a K-means Clustering model | 281 |
| <strong>KNN</strong> - Build a K-Nearest Neighbors model | 284 |
| <strong>KURTOSIS_AGG</strong> - excess Kurtosis | 287 |
| <strong>LDF_MANOVA_ONE WAY TEST</strong> - LDF of ABC Multivariate Analysis of Variance result in one way setting | 288 |
| <strong>LDF_MANOVA_TWO WAY TEST</strong> - LDF of Multivariate Analysis of Variance result in TWO way setting | 289 |
| <strong>LINEAR_REGRESSION</strong> - Build a Linear Regression model | 291 |
| <strong>LIST_COLPROPS</strong> - List column properties for selected or all analytics models | 296 |
| <strong>LIST_COMPONENTS</strong> - List components of selected or all analytics models | 298 |
| <strong>LIST_MODELS</strong> - List selected or all analytics models | 301 |
| <strong>LIST_PARAMS</strong> - List parameters for selected or all analytics models | 303 |
| <strong>LIST_PRIVILEGES</strong> - List privileges on all analytics models | 306 |
| <strong>MAE</strong> - Mean Absolute Error | 307 |
| <strong>MANOVA_ONE WAY TEST</strong> - Multivariate Analysis of Variance in one way setting | 309 |
| <strong>MANOVA_Two WAY TEST</strong> - Multivariate Analysis of Variance in Two way setting | 326 |
| <strong>MEDIAN</strong> - Median value for a numeric attribute | 331 |
| <strong>MEDIAN_DISC</strong> - Median value for a discrete attribute | 332 |
| <strong>METADATA_ANALYZE</strong> - Check and Repair Metadata Repository | 333 |
| <strong>MIGRATE_MODEL</strong> - Migrate an analytics model | 336 |
| <strong>MODEL_EXISTS</strong> - Check if an analytics model exists | 336 |
| <strong>MOMENTS</strong> - Moments of a column | 338 |
| <strong>MSE</strong> - Mean Squared Error | 342 |
| <strong>MTBNET_DIFF</strong> - Show differences between multiple tree-shaped bayesian networks | 344 |
| <strong>MTBNET_GROW</strong> - Build a multi-tree bayesian network for correlations | 345 |
| <strong>MUTUALINFO</strong> - Mutual Information | 349 |
| <strong>MUTUALINFO_AGG</strong> - Mutual Information | 351 |
| <strong>MWW_TEST</strong> - Mann-Whitney-Wilcoxon test of independence | 352 |
| <strong>NAIVEBAYES</strong> - Build a Naive Bayes model | 356 |
| <strong>OUTLIERS</strong> - Detect outliers for a numeric attribute | 359 |
| <strong>PBERN</strong> - Cumulative Bernoulli Distribution | 361 |
| <strong>PBERN_H</strong> - Cumulative Bernoulli Distribution, high tail | 362 |</p>
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PBETA</td>
<td>Cumulative Beta Distribution</td>
<td>364</td>
</tr>
<tr>
<td>PBETA_H</td>
<td>Cumulative Beta Distribution, high tail</td>
<td>365</td>
</tr>
<tr>
<td>PBINOM</td>
<td>Cumulative Binomial Distribution</td>
<td>367</td>
</tr>
<tr>
<td>PBINOM_H</td>
<td>Cumulative Binomial Distribution, high tail</td>
<td>369</td>
</tr>
<tr>
<td>PCA</td>
<td>Build a Principal Component Analysis model</td>
<td>371</td>
</tr>
<tr>
<td>PCAUCHY</td>
<td>Cumulative Cauchy Distribution</td>
<td>374</td>
</tr>
<tr>
<td>PCAUCHY_H</td>
<td>Cumulative Cauchy Distribution, high tail</td>
<td>375</td>
</tr>
<tr>
<td>PCHISQ</td>
<td>Cumulative Chi-square Distribution</td>
<td>377</td>
</tr>
<tr>
<td>PCHISQ_H</td>
<td>Cumulative Chi-square Distribution, high tail</td>
<td>378</td>
</tr>
<tr>
<td>PCHISQ_S</td>
<td>Cumulative Chi-square Distribution with string argument</td>
<td>379</td>
</tr>
<tr>
<td>PERCENTAGE_SPLIT</td>
<td>Build and evaluate a Classification model on a randomly split table</td>
<td>382</td>
</tr>
<tr>
<td>PEXP</td>
<td>Cumulative Exponential Distribution</td>
<td>384</td>
</tr>
<tr>
<td>PEXP_H</td>
<td>Cumulative Exponential Distribution, high tail</td>
<td>385</td>
</tr>
<tr>
<td>PF</td>
<td>Cumulative Fisher Distribution</td>
<td>386</td>
</tr>
<tr>
<td>PF_H</td>
<td>Cumulative Fisher Distribution, high tail</td>
<td>387</td>
</tr>
<tr>
<td>PFISK</td>
<td>Cumulative Fisk (or log-logistic) Distribution</td>
<td>389</td>
</tr>
<tr>
<td>PFISK_H</td>
<td>Cumulative Fisk (or log-logistic) Distribution, high tail</td>
<td>390</td>
</tr>
<tr>
<td>PGAMMA</td>
<td>Cumulative Gamma Distribution</td>
<td>392</td>
</tr>
<tr>
<td>PGAMMA_H</td>
<td>Cumulative Gamma Distribution, high tail</td>
<td>393</td>
</tr>
<tr>
<td>PGEOM</td>
<td>Cumulative Geometric Distribution</td>
<td>394</td>
</tr>
<tr>
<td>PGEOM_H</td>
<td>Cumulative Geometric Distribution, high tail</td>
<td>396</td>
</tr>
<tr>
<td>PHYPER</td>
<td>Cumulative Hypergeometric Distribution</td>
<td>397</td>
</tr>
<tr>
<td>PHYPER_H</td>
<td>Cumulative Hypergeometric Distribution, high tail</td>
<td>399</td>
</tr>
<tr>
<td>PLNORM</td>
<td>Cumulative Galton (or LogNormal) Distribution</td>
<td>402</td>
</tr>
<tr>
<td>PLNORM_H</td>
<td>Cumulative Galton (or LogNormal) Distribution, high tail</td>
<td>403</td>
</tr>
<tr>
<td>PLOGIS</td>
<td>Cumulative Logistic Distribution</td>
<td>404</td>
</tr>
<tr>
<td>PLOGIS_H</td>
<td>Cumulative Logistic Distribution, high tail</td>
<td>406</td>
</tr>
<tr>
<td>PMML_DECTREE</td>
<td>Convert a Decision Tree model to PMML</td>
<td>407</td>
</tr>
<tr>
<td>PMML_MODEL</td>
<td>Generate a PMML representation of an analytics model</td>
<td>409</td>
</tr>
<tr>
<td>PMML_NAIVEBAYES</td>
<td>Convert a Naive Bayes model to PMML</td>
<td>411</td>
</tr>
<tr>
<td>PMWW</td>
<td>Cumulative Mann-Whitney-Wilcoxon Distribution</td>
<td>413</td>
</tr>
<tr>
<td>PMWW_H</td>
<td>Cumulative Mann-Whitney-Wilcoxon Distribution, high tail</td>
<td>414</td>
</tr>
<tr>
<td>PNBINOM</td>
<td>Cumulative Negative Binomial Distribution</td>
<td>416</td>
</tr>
<tr>
<td>PNBINOM_H</td>
<td>Negative Binomial Distribution, high tail</td>
<td>417</td>
</tr>
<tr>
<td>PNORM</td>
<td>Cumulative Standard Normal Distribution</td>
<td>419</td>
</tr>
<tr>
<td>PNORM3P</td>
<td>Cumulative Normal Distribution</td>
<td>420</td>
</tr>
<tr>
<td>PNORM_H</td>
<td>Cumulative Standard Normal Distribution, high tail</td>
<td>422</td>
</tr>
<tr>
<td>PPOINT</td>
<td>Percentage Point Distributions</td>
<td>423</td>
</tr>
<tr>
<td>PPOIS</td>
<td>Cumulative Poisson Distribution</td>
<td>426</td>
</tr>
<tr>
<td>PPOIS_H</td>
<td>Cumulative Poisson Distribution, high tail</td>
<td>428</td>
</tr>
</tbody>
</table>
PPV - Positive Predictive Value from a Confusion Matrix .......................................................... 429
PREDICT_ARULE - Predict an Association Rules model ............................................................... 431
PREDICT_DECETREE - Apply a Decision Tree model .............................................................. 437
PREDICT_DIVCLUSTER - Apply a Hierarchical Clustering model ........................................... 439
PREDICT_GLM - Apply a Generalized Linear Model .......................................................... 442
PREDICT_KMEANS - K-means Clustering .............................................................................. 443
PREDICT_KNN - Apply a K-Nearest Neighbors model ............................................................... 446
PREDICT_LINEAR_REGRESSION - Apply a Linear Regression Model ....................................... 449
PREDICT_NAIVEBAYES - Apply a Naive Bayes model ........................................................... 452
PREDICT_REGTREE - Apply a Regression Tree model ............................................................. 455
PREDICT_TWOSTEP - Apply a TwoStep Clustering model ...................................................... 457
PRINT_ARULE - Print an Association Rules model .................................................................. 458
PRINT_DECETREE - Print a Decision Tree model .................................................................... 464
PRINT_GLM - Print a Generalized Linear Model ...................................................................... 466
PRINT_KMEANS - Print a K-means model ............................................................................... 470
PRINT_MANOVA_ONE_WAY_TEST - Pretty print of Multivariate Analysis of Variance result in one way setting .......................................................................................... 473
PRINT_MANOVA_TWO_WAY_TEST - Pretty print of Multivariate Analysis of Variance result in two way setting ............................................................................................. 478
PRINT_MODEL - Print an analytics model .................................................................................. 479
PRINT_REGTREE - Print a Regression Tree model ................................................................. 481
PRINT_TIMESERIES - Print Time Series predictions .............................................................. 483
PRINT_TWOSTEP - Print a TwoStep clustering model ............................................................. 486
PROJECT_PCA - Apply a Principal Component Analysis model ............................................. 489
PRUNE_DECETREE - Prune a Decision Tree model ............................................................... 491
PRUNE_REGTREE - Prune a Regression Tree model .............................................................. 493
PT - Cumulative T-student Distribution ............................................................................... 495
PT - Cumulative T-student Distribution with string argument ............................................... 497
PT_H - Cumulative T-student Distribution, high tail ............................................................ 498
PUNIF - Cumulative Uniform Distribution ............................................................................ 499
PUNIF_H - Cumulative Uniform Distribution, high tail ......................................................... 500
PWALD - Cumulative Wald Distribution ............................................................................... 501
PWALD_H - Cumulative Wald Distribution, high tail ............................................................ 503
PWEIBULL - Cumulative Weibull (or Rosin-Rammer) Distribution ....................................... 504
PWEIBULL_H - Cumulative Weibull (or Rosin-Rammer) Distribution, high tail .................... 506
PWILCOX - Cumulative Wilcoxon Distribution ..................................................................... 507
PWILCOX_H - Cumulative Wilcoxon Distribution, high tail .................................................. 508
QBERN - Inverse Bernoulli Distribution ............................................................................... 510
QBERN_H - Inverse Bernoulli Distribution, high tail ............................................................. 511
QBETA - Inverse Beta Distribution ......................................................................................... 513
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>QBETA_H</td>
<td>Inverse Beta Distribution</td>
<td>514</td>
</tr>
<tr>
<td>QBINOM</td>
<td>Inverse Binomial Distribution</td>
<td>516</td>
</tr>
<tr>
<td>QBINOM_H</td>
<td>Inverse Binomial Distribution, high tail</td>
<td>519</td>
</tr>
<tr>
<td>QCAUCHY</td>
<td>Inverse Cauchy Distribution</td>
<td>522</td>
</tr>
<tr>
<td>QCAUCHY_H</td>
<td>Inverse Cauchy Distribution, high tail</td>
<td>524</td>
</tr>
<tr>
<td>QCHISQ</td>
<td>Inverse Chi-square Distribution</td>
<td>526</td>
</tr>
<tr>
<td>QCHISQ_H</td>
<td>Inverse Chi-square Distribution, high tail</td>
<td>527</td>
</tr>
<tr>
<td>QEXP</td>
<td>Inverse Exponential Distribution</td>
<td>528</td>
</tr>
<tr>
<td>QEXP_H</td>
<td>Inverse Exponential Distribution, high tail</td>
<td>530</td>
</tr>
<tr>
<td>QF</td>
<td>Inverse Fisher Distribution</td>
<td>531</td>
</tr>
<tr>
<td>QF_H</td>
<td>Inverse Fisher Distribution, high tail</td>
<td>532</td>
</tr>
<tr>
<td>QFISK</td>
<td>Inverse Fisk (or log-logistic) Distribution</td>
<td>533</td>
</tr>
<tr>
<td>QFISK_H</td>
<td>Inverse Fisk (or log-logistic) Distribution, high tail</td>
<td>535</td>
</tr>
<tr>
<td>QGAMMA</td>
<td>Inverse Gamma Distribution</td>
<td>536</td>
</tr>
<tr>
<td>QGAMMA_H</td>
<td>Inverse Gamma Distribution, high tail</td>
<td>538</td>
</tr>
<tr>
<td>QGEOM</td>
<td>Inverse Geometric Distribution</td>
<td>539</td>
</tr>
<tr>
<td>QGEOM_H</td>
<td>Inverse Geometric Distribution, high tail</td>
<td>541</td>
</tr>
<tr>
<td>QHYPER</td>
<td>Inverse Hypergeometric Distribution</td>
<td>542</td>
</tr>
<tr>
<td>QHYPER_H</td>
<td>Inverse Hypergeometric Distribution, high tail</td>
<td>544</td>
</tr>
<tr>
<td>QLNORM</td>
<td>Inverse Galton (or LogNormal) Distribution</td>
<td>546</td>
</tr>
<tr>
<td>QLNORM_H</td>
<td>Inverse Galton (or LogNormal) Distribution, high tail</td>
<td>547</td>
</tr>
<tr>
<td>QLOGIS</td>
<td>Inverse Logistic Distribution</td>
<td>549</td>
</tr>
<tr>
<td>QLOGIS_H</td>
<td>Inverse Logistic Distribution, high tail</td>
<td>551</td>
</tr>
<tr>
<td>QMWW</td>
<td>Inverse Mann-Whitney-Wilcoxon Distribution</td>
<td>552</td>
</tr>
<tr>
<td>QMWW_H</td>
<td>Inverse Mann-Whitney-Wilcoxon Distribution, high tail</td>
<td>553</td>
</tr>
<tr>
<td>QNBINOM</td>
<td>Inverse Negative Binomial Distribution</td>
<td>555</td>
</tr>
<tr>
<td>QNBINOM_H</td>
<td>Inverse Negative Binomial Distribution, high tail</td>
<td>557</td>
</tr>
<tr>
<td>QNORM</td>
<td>Inverse Standard Normal Distribution</td>
<td>559</td>
</tr>
<tr>
<td>QNORM3P</td>
<td>Inverse Normal Distribution</td>
<td>560</td>
</tr>
<tr>
<td>QNORM_H</td>
<td>Inverse Standard Normal Distribution, high tail</td>
<td>562</td>
</tr>
<tr>
<td>QPOIS</td>
<td>Inverse Poisson Distribution</td>
<td>563</td>
</tr>
<tr>
<td>QPOIS_H</td>
<td>Inverse Poisson Distribution, high tail</td>
<td>565</td>
</tr>
<tr>
<td>QT</td>
<td>Inverse t-Student Distribution</td>
<td>567</td>
</tr>
<tr>
<td>QT_H</td>
<td>Inverse t-Student Distribution, high tail</td>
<td>568</td>
</tr>
<tr>
<td>QUANTILE</td>
<td>Quantile for a numeric attribute</td>
<td>569</td>
</tr>
<tr>
<td>QUANTILE_DISC</td>
<td>Quantile for a discrete attribute</td>
<td>571</td>
</tr>
<tr>
<td>QUARTILE</td>
<td>Quartile for a numeric attribute</td>
<td>573</td>
</tr>
<tr>
<td>QUARTILE_DISC</td>
<td>Quartile for a discrete attribute</td>
<td>574</td>
</tr>
<tr>
<td>QUNIF</td>
<td>Inverse Uniform Distribution</td>
<td>576</td>
</tr>
<tr>
<td>QUNIF_H</td>
<td>Inverse Uniform Distribution, high tail</td>
<td>577</td>
</tr>
</tbody>
</table>
QWALD - Inverse Wald Distribution ................................................................. 579
QWALD_H - Inverse Wald Distribution, high tail ........................................ 581
QWEIBULL - Inverse Weibull (or Rosin-Rammer) Distribution ..................... 583
QWEIBULL_H - Inverse Weibull (or Rosin-Rammer) Distribution, high tail ...... 584
QWILCOX - Inverse Wilcoxon Distribution .................................................. 586
QWILCOX_H - Inverse Wilcoxon Distribution, high tail ............................... 587
RAE - Relative Absolute Error .................................................................... 588
RANDOM_SAMPLE - Random Sample .......................................................... 591
REGISTER_MODEL - Register a version 1.x analytics model to model management 593
REGTREE - Build then prune a Regression Tree model .................................. 600
REVOKE_MODEL - Revoke privileges on an analytics model ......................... 603
RSE - Relative Squared Error ...................................................................... 606
SET_CLUSTERNAME - set name and/or description of a K-means cluster .......... 608
SET_COLUMN_PROPERTIES - Set columns role, type or weight in a column properties table ... 610
SKEWNESS_AGG - Skewness ...................................................................... 613
SPEARMAN_CORR - Spearman Rank Correlation .......................................... 614
SPEARMAN_CORR_S - Spearman Rank Correlation as string ....................... 618
SPLIT_DATA - Split a table into training and testing sets .............................. 621
STD_NORM - Standardization and Normalization ......................................... 623
SUMMARY1000 - Summary of up to 1000 columns ..................................... 627
SUMMARY1000CHAR - Summary of up to 1000 character columns ............... 641
SUMMARY1000DATE - Summary of up to 1000 date columns ....................... 647
SUMMARY1000INTERVAL - Summary of up to 1000 interval columns .......... 649
SUMMARY1000NUM - Summary of up to 1000 numeric columns .................. 652
SUMMARY1000TIME - Summary of up to 1000 time columns ....................... 661
SUMMARY1000TIMESTAMP - Summary of up to 1000 timestamp columns ...... 664
T_LS_TEST - T-Student test for the linear relationship of two columns .......... 667
T_LS_TEST_S_AGG - T-Student test between linearly dependent samples as string 670
T_ME_TEST - T-Student test for the expected mean of a column ................. 671
T_ME_TEST_S_AGG - T-Student test for the expected mean of a variable as string 674
T_PMD_TEST - T-Student test for the expected mean difference between two paired columns 675
T_PMD_TEST_S_AGG - T-Student test for the expected mean difference between two paired variables as string ................................................................. 678
T_TEST_AGG - T-Student test of a variable split into two classes .................. 679
T_TEST_S_AGG - T-Student test of a variable split into two classes as string .... 680
T_UMD_TEST - T-Student test for the expected mean difference between class values of a column ................................................................. 682
TANET_APPLY - Apply a tree-augmented network model ............................. 684
TANET_CLASSAPPLY - Apply a tree-augmented network .............................. 687
TANET_GROW - Build a tree-augmented network model .............................. 689
3 Reference Documentation: Utilities

_sp_utl_aggregateExists - Check if a UDA aggregate function exists ............................................. 733
_sp_utl_columnContainsNulls - Check if a column contains at least one NULL value ....................... 734
_sp_utl_columnExists - Check if a column exists ........................................................................... 736
_sp_utl_columnIsId - Check if a column contains unique values ...................................................... 737
_sp_utl_columnIsNumeric - Check if a column is of a numeric data type ....................................... 739
_sp_utl_columnListExists - Check if a columns exists .................................................................... 741
_sp_utl_columnsEqualTypes - Check if two columns have the same data type ................................. 742
_sp_utl_dropAllAggregates - Drop all UDA aggregate functions with the given prefix .................. 743
_sp_utl_dropAllFunctions - Drop all UDF functions with the given prefix ........................................ 744
_sp_utl_dropAllLike - Drop all database objects with the given LIKE expression ............................. 745
_sp_utl_dropAllProcedures - Drop all stored procedures with the given prefix ............................... 746
_sp_utl_dropAllUDX - Drop all UDx functions with the given prefix ................................................. 747
_sp_utl_functionExists - Check if a UDF function exists ................................................................. 748
_sp_utl_getColumnType - Column data type .................................................................................. 749
_sp_utl_getTableSize - Table row count ......................................................................................... 751
_sp_utl_isTempTable - Check if a table is temporary ...................................................................... 751
_sp_utl_justExecute - Execute a SQL statement ............................................................................. 753
_sp_utl_procedureExists - Check if a stored procedure exists ......................................................... 754
_sp_utl_relationExists - Check if a table or a view exists ............................................................... 756
_sp_utl_sequenceExists - Check if a sequence exists ...................................................................... 757
_sp_utl_tableExists - Check if a table exists .................................................................................. 758
_sp_utl_viewExists - Check if a view exists ................................................................................... 759
drand64 - 64 bits pseudo-random number generator ..................................................................... 761
DROP_TABLE - Drop a table ............................................................................................................. 761
ISDATE_TINY - Check if a string has the compact date format YYYYMMDD .......................... 763
msghelp - Show information related to a message number in the correct language of the locale.
.................................................................................................................................................. 765

Notices and Trademarks

Notices........................................................................................................................................ 766
Trademarks .................................................................................................................................. 767
Regulatory and Compliance ......................................................................................................... 768
  Regulatory Notices..................................................................................................................... 768
  Homologation Statement........................................................................................................... 768
  FCC - Industry Canada Statement.............................................................................................. 768
  WEEE......................................................................................................................................... 768
  CE Statement (Europe)............................................................................................................... 768
  VCCI Statement.......................................................................................................................... 769

Index
Preface

This guide describes the functions available for use within the IBM Netezza In-Database Analytics Package.

Audience for This Guide

The IBM Netezza In-Database Analytics Package was designed for users interested in using the provided data mining algorithms. Because this package includes heavy use of mathematical and statistical processes, you should have a strong understanding of these concepts to use it. Further, you should be familiar with the basic operation and concepts of the IBM Netezza system and the NPS.

The Purpose of This Guide

This guide describes the use of each algorithm in the IBM Netezza In-Database Analytics Package.

Conventions

Note on Terminology: The terms User-Defined Analytic Process (UDAP) and Analytic Executable (AE) are synonymous.

The following conventions apply:

► *Italics* for emphasis on terms and user-defined values, such as user input.
► Upper case for SQL commands, for example, INSERT or DELETE.
► Bold for command line input, for example, *nzsystem stop*.
► Bold to denote parameter names, argument names, or other named references.
► Angle brackets ( < > ) to indicate a placeholder (variable) that should be replaced with actual text, for example, `nzmat <- nz.matrix("<matrix_name>")`.
► A single backslash ("\") at the end of a line of code to denote a line continuation. Omit the backslash when using the code at the command line, in a SQL command, or in a file.
► When referencing a sequence of menu and submenu selections, the ">" character denotes the different menu options, for example *Menu Name > Submenu Name > Selection*.

If You Need Help

If you are having trouble using the IBM Netezza appliance, IBM Netezza Analytics or any of its components:

1. Retry the action, carefully following the instructions in the documentation.
3. If you have an active service contract maintenance agreement with IBM, you can contact customer support teams via telephone. For individual countries, please visit the Technical Support section of the IBM Directory of worldwide contacts.
Comments on the Documentation

We welcome any questions, comments, or suggestions that you have for the IBM Netezza documentation. Please send us an e-mail message at netezza-doc@wwpdl.vnet.ibm.com and include the following information:

► The name and version of the manual that you are using
► Any comments that you have about the manual
► Your name, address, and phone number

We appreciate your comments.
CHAPTER 1
List of functions by category

Analytics - Association Rules

ARULE - Build an Association Rules model
PREDICT_ARULE - Predict an Association Rules model
PRINT_ARULE - Print an Association Rules model
VERIFY_ARULE - Verify if data is prepared to build an Association Rules model

Analytics - Classification

CROSS_VALIDATION - Build and cross validate a Classification model
DECTREE - Build and prune a Decision Tree model
GROW_DECTREE - Build a Decision Tree model
KNN - Build a K-Nearest Neighbors model
NAIVEBAYES - Build a Naive Bayes model
PERCENTAGE_SPLIT - Build and evaluate a Classification model on a randomly split table
PMML_DECTREE - Convert a Decision Tree model to PMML
PMML_NAIVEBAYES - Convert a Naive Bayes model to PMML
PREDICT_DECTREE - Apply a Decision Tree model
PREDICT_KNN - Apply a K-Nearest Neighbors model
PREDICT_NAIVEBAYES - Apply a Naive Bayes model
PRINT_DECTREE - Print a Decision Tree model
PRUNE_DECTREE - Prune a Decision Tree model
TRAIN_TEST - Build and evaluate a Classification model on given training and testing sets
Analytics - Clustering

DIVCLUSTER - Build a Hierarchical Clustering model
KMEANS - Build a K-means Clustering model
PREDICT_DIVCLUSTER - Apply a Hierarchical Clustering model
PREDICT_KMEANS - K-means Clustering
PREDICT_TWOSTEP - Apply a TwoStep Clustering model
PRINT_KMEANS - Print a K-means model
PRINT_TWOSTEP - Print a TwoStep clustering model
SET_CLUSTERNAME - set name and/or description of a K-means cluster
TWOSTEP - Build a TwoStep Clustering model

Analytics - Column Properties

COLUMN_PROPERTIES - Create a column properties table
COLUMN_PROPERTIES_CHECK - Check a column properties table for correctness
GET_COLUMN_LIST - Retrieve a list of columns with a given role and type from a column properties table
SET_COLUMN_PROPERTIES - Set columns role, type or weight in a column properties table

Analytics - Data Transformation

IMPUTE_DATA - Impute missing data
PCA - Build a Principal Component Analysis model
PROJECT_PCA - Apply a Principal Component Analysis model
SPLIT_DATA - Split a table into training and testing sets
STD_NORM - Standardization and Normalization

Analytics - Diagnostic Measures

ACC - Classification accuracy
CERROR - Classification Error
CMATRIX_ACC - Classification accuracy from a Confusion Matrix
CMATRIX_STATS - Print Classification quality factors from a Confusion Matrix
CMATRIX_WACC - Weighted Classification accuracy from a Confusion Matrix
CONFUSION_MATRIX - Build a Confusion Matrix
FMEASURE - F-Measure from a Confusion Matrix
FPR - False Positive Rate from a Confusion Matrix
MAE - Mean Absolute Error
MSE - Mean Squared Error
PPV - Positive Predictive Value from a Confusion Matrix
RAE - Relative Absolute Error
RSE - Relative Squared Error
TPR - True Positive Rate from a Confusion Matrix
WACC - Weighted Classification accuracy

Analytics - Discretization
APPLY_DISC - Discretize data using given discretization bin limits
EFDISC - Discretization bins of equal frequency
EMDISC - Discretization bins of minimal entropy
EWDISC - Discretization bins of equal width
EWDISC_NICE - Discretization bins of equal width with human-friendly limits

Analytics - Model Management
ALTER_MODEL - Alter properties of an analytics model
CLEANUP - Remove the model management infrastructure from the current database
COPY_MODEL - Duplicate an analytics model
DROP_ALL_MODELS - Drop selected or all analytics models
DROP_MODEL - Drop an analytics model
EXPORT_MODEL - Export one or more analytic models to a file set
EXPORT_PMML - Export an analytics model as PMML document to a file
GRANT_MODEL - Grant privileges on an analytics model
IMPORT_MODEL - Import one or more analytics models from a file set
INITIALIZE - Initialize the model management infrastructure
IS_INITIALIZED - Check the model management infrastructure
LIST_COLPROPS - List column properties for selected or all analytics models
LIST_COMPONENTS - List components of selected or all analytics models
LIST_MODELS - List selected or all analytics models
LIST_PARAMS - List parameters for selected or all analytics models
LIST_PRIVILEGES - List privileges on all analytics models
METADATA_ANALYZE - Check and Repair Metadata Repository
MIGRATE_MODEL - Migrate an analytics model
MODEL_EXISTS - Check if an analytics model exists
PMML_MODEL - Generate a PMML representation of an analytics model
PRINT_MODEL - Print an analytics model
REGISTER_MODEL - Register a version 1.x analytics model to model management
REVOKE_MODEL - Revoke privileges on an analytics model

Analytics - Probability Distributions

CUMULATIVE - Cumulative Distributions
DBERN - Density of the Bernoulli Distribution
DBETA - Probability density function for Beta distribution
DBINOM - Density of the Binomial Distribution
DCAUCHY - Probability density function for Cauchy Distribution
DCHISQ - Probability density function for Chi-square Distribution
DENSITY - Density of Distributions
DEXP - Probability density function for Exponential Distribution
DF - Probability density function for Fisher Distribution
DFISK - Probability density function for Fisk (or log-logistic) Distribution
DGAMMA - Probability density function for Gamma distribution
DGEOM - Density of the Geometric Distribution
DHYPER - Density of the Hypergeometric Distribution
DLNORM - Density of Galton (or LogNormal) Distribution
DLOGIS - Probability density function for Logistic Distribution
DMWW - Density of the Mann-Whitney-Wilcoxon Distribution
DNBINOM - Density of the Negative Binomial Distribution
DNORM - Density of the Standard Normal Distribution
DNORM3P - Density of the Normal Distribution
DPOIS - Density of the Poisson Distribution
DT - Density of the T-student Distribution
DUNIF - Density of the Uniform Distribution
DWALD - Density of the Wald distribution
DWEIBULL - Density of the Weibull (or Rosin-Rammer) Distribution
DWILCOX - Density of the Wilcoxon Distribution
PBERN - Cumulative Bernoulli Distribution
PBERN_H - Cumulative Bernoulli Distribution, high tail
PBETA - Cumulative Beta Distribution
PBETA_H - Cumulative Beta Distribution, high tail
### List of functions by category

- **PBINOM** - Cumulative Binomial Distribution
- **PBINOM_H** - Cumulative Binomial Distribution, high tail
- **PCAUCHY** - Cumulative Cauchy Distribution
- **PCAUCHY_H** - Cumulative Cauchy Distribution, high tail
- **PCHISQ** - Cumulative Chi-square Distribution
- **PCHISQ_H** - Cumulative Chi-square Distribution, high tail
- **PCHISQ_S** - Cumulative Chi-square Distribution with string argument
- **PEXP** - Cumulative Exponential Distribution
- **PEXP_H** - Cumulative Exponential Distribution, high tail
- **PF** - Cumulative Fisher Distribution
- **PF_H** - Cumulative Fisher Distribution, high tail
- **PFISK** - Cumulative Fisk (or log-logistic) Distribution
- **PFISK_H** - Cumulative Fisk (or log-logistic) Distribution, high tail
- **PGAMMA** - Cumulative Gamma Distribution
- **PGAMMA_H** - Cumulative Gamma Distribution, high tail
- **PGEOM** - Cumulative Geometric Distribution
- **PGEOM_H** - Cumulative Geometric Distribution, high tail
- **PHYPER** - Cumulative Hypergeometric Distribution
- **PHYPER_H** - Cumulative Hypergeometric Distribution, high tail
- **PLNORM** - Cumulative Galton (or LogNormal) Distribution
- **PLNORM_H** - Cumulative Galton (or LogNormal) Distribution, high tail
- **PLOGIS** - Cumulative Logistic Distribution
- **PLOGIS_H** - Cumulative Logistic Distribution, high tail
- **PMWW** - Cumulative Mann-Whitney-Wilcoxon Distribution
- **PMWW_H** - Cumulative Mann-Whitney-Wilcoxon Distribution, high tail
- **PNBINOM** - Cumulative Negative Binomial Distribution
- **PNBINOM_H** - Negative Binomial Distribution, high tail
- **PNORM** - Cumulative Standard Normal Distribution
- **PNORM3P** - Cumulative Normal Distribution
- **PNORM_H** - Cumulative Standard Normal Distribution, high tail
- **PPOINT** - Percentage Point Distributions
- **PPOIS** - Cumulative Poisson Distribution
- **PPOIS_H** - Cumulative Poisson Distribution, high tail
- **PT** - Cumulative T-student Distribution
- **PT** - Cumulative T-student Distribution with string argument
PT_H - Cumulative T-student Distribution, high tail
PUNIF - Cumulative Uniform Distribution
PUNIF_H - Cumulative Uniform Distribution, high tail
PWALD - Cumulative Wald Distribution
PWALD_H - Cumulative Wald Distribution, high tail
PWEIBULL - Cumulative Weibull (or Rosin-Rammer) Distribution
PWEIBULL_H - Cumulative Weibull (or Rosin-Rammer) Distribution, high tail
PWILCOX - Cumulative Wilcoxon Distribution
PWILCOX_H - Cumulative Wilcoxon Distribution, high tail
QBERN - Inverse Bernoulli Distribution
QBERN_H - Inverse Bernoulli Distribution, high tail
QBETA - Inverse Beta Distribution
QBETA_H - Inverse Beta Distribution, high tail
QBINOM - Inverse Binomial Distribution
QBINOM_H - Inverse Binomial Distribution, high tail
QCAUCHY - Inverse Cauchy Distribution
QCAUCHY_H - Inverse Cauchy Distribution, high tail
QCHISQ - Inverse Chi-square Distribution
QCHISQ_H - Inverse Chi-square Distribution, high tail
QEXP - Inverse Exponential Distribution
QEXP_H - Inverse Exponential Distribution, high tail
QF - Inverse Fisher Distribution
QF_H - Inverse Fisher Distribution, high tail
QFISK - Inverse Fisk (or log-logistic) Distribution
QFISK_H - Inverse Fisk (or log-logistic) Distribution, high tail
QGAMMA - Inverse Gamma Distribution
QGAMMA_H - Inverse Gamma Distribution, high tail
QGEOGAM - Inverse Geometric Distribution
QGEOGAM_H - Inverse Geometric Distribution, high tail
QHYPER - Inverse Hypergeometric Distribution
QHYPER_H - Inverse Hypergeometric Distribution, high tail
QLNORM - Inverse Galton (or LogNormal) Distribution
QLNORM_H - Inverse Galton (or LogNormal) Distribution, high tail
List of functions by category

QLOGIS - Inverse Logistic Distribution
QLOGIS_H - Inverse Logistic Distribution, high tail
QMWW - Inverse Mann-Whitney-Wilcoxon Distribution
QMWW_H - Inverse Mann-Whitney-Wilcoxon Distribution, high tail
QNBINOM - Inverse Negative Binomial Distribution
QNBINOM_H - Inverse Negative Binomial Distribution, high tail
QNORM - Inverse Standard Normal Distribution
QNORM3P - Inverse Normal Distribution
QNORM_H - Inverse Standard Normal Distribution, high tail
QPOIS - Inverse Poisson Distribution
QPOIS_H - Inverse Poisson Distribution, high tail
QT - Inverse t-Student Distribution
QT_H - Inverse t-Student Distribution, high tail
QUNIF - Inverse Uniform Distribution
QUNIF_H - Inverse Uniform Distribution, high tail
QWALD - Inverse Wald Distribution
QWALD_H - Inverse Wald Distribution, high tail
QWEIBULL - Inverse Weibull (or Rosin-Rammer) Distribution
QWEIBULL_H - Inverse Weibull (or Rosin-Rammer) Distribution, high tail
QWILCOX - Inverse Wilcoxon Distribution
QWILCOX_H - Inverse Wilcoxon Distribution, high tail

Analytics - Quantiles & Outliers

IQR - InterQuartile Range
MEDIAN - Median value for a numeric attribute
MEDIAN_DISC - Median value for a discrete attribute
OUTLIERS - Detect outliers for a numeric attribute
QUANTILE - Quantile for a numeric attribute
QUANTILE_DISC - Quantile for a discrete attribute
QUARTILE - Quartile for a numeric attribute
QUARTILE_DISC - Quartile for a discrete attribute

Analytics - Regression

BTBNET_GROW - Build a tree-like Bayesian Network model
GLM - Build a Generalized Linear Model
GROW_REGTREE - Grow a Regression Tree model
LINEAR_REGRESSION - Build a Linear Regression model
MTBNET_DIFF - Show differences between multiple tree-shaped bayesian networks
MTBNET_GROW - Build a multi-tree bayesian network for correlations
PREDICT_GLM - Apply a Generalized Linear Model
PREDICT_LINEAR_REGRESSION - Apply a Linear Regression Model
PREDICT_REGTREE - Apply a Regression Tree model
PRINT_GLM - Print a Generalized Linear Model
PRINT_REGTREE - Print a Regression Tree model
PRUNE_REGTREE - Prune a Regression Tree model
REGTREE - Build then prune a Regression Tree model
TANET_APPLY - Apply a tree-augmented network model
TANET_CLASSAPPLY - Apply a tree-augmented network
TANET_GROW - Build a tree-augmented network model
TBNET1G - Build a tree-like Bayesian Network model
TBNET1G2P - Build a tree-like Bayesian Network model
TBNET2G - Build a tree-like Bayesian Network model in bi-partite mode
TBNET_APPLY - Apply a tree-like Bayesian Network model
TBNET_GROW - Build a tree-like Bayesian Network model

Analytics - Sampling
RANDOM_SAMPLE - Random Sample

Analytics - Statistics
ANOVA_CRD_TEST - Analyze variance using Completely Randomized Design and One Way ANOVA
ANOVA_RBD_TEST - Analyze variance using ANOVA Randomized Block Design
BITABLE - Bivariate Frequencies
CANONICAL_CORR - Canonical Correlation
CANONICAL_CORR_AGG - Canonical Correlation
CBFS_AGG - Correlation-based feature selection
CHISQ_TEST - Pearson's Chi-square test of independence
CHISQ_TEST_AGG - Pearson's Chi-square
CHISQ_TEST_S_AGG - Pearson's Chi-square with degree of freedom
COL2TRCV_MANOVA_ONE_WAY_TEST - data transformation from column representation to trcv for Multivariate Analysis of Variance in one way setting
List of functions by category

COL2TRCV_MANOVA_TWO_WAY_TEST - data transformation from column representation to grcv for Multivariate Analysis of Variance in two way setting
COND_ENTROPY - Conditional Entropy
CORR - Mutual Correlation
CORR_AGG - Pearson's Correlation
CORR_AGG - Pearson's Correlation with weight
CORR_MATRIX_AGG - Correlation Matrix
CORRELATION1000MATRIX - Correlation matrix
CORRELATION500PAIRS - Correlation of pairs of variables
COV - Covariance
COV_AGG - Pearson's Covariance
COV_MATRIX_AGG - Covariance Matrix
COVARIANCE1000MATRIX - Covariance matrix
COVARIANCE500PAIRS - Covariance of pairs of variables
COVARIANCEMATRIX - Covariance matrix as string
DROP_SUMMARY1000 - Drop tables created by SUMMARY1000
ENTROPY - Univariate Entropy
HIST - Histograms
JOINT_ENTROPY - Bivariate Entropy
KURTOSIS_AGG - excess Kurtosis
LDF_MANOVA_ONE_WAY_TEST - LDF of ABC Multivariate Analysis of Variance result in one way setting
LDF_MANOVA_TWO_WAY_TEST - LDF of Multivariate Analysis of Variance result in TWO way setting
MANOVA_ONE_WAY_TEST - Multivariate Analysis of Variance in one way setting
MANOVA_TWO_WAY_TEST - Multivariate Analysis of Variance in Two way setting
MOMENTS - Moments of a column
MUTUALINFO - Mutual Information
MUTUALINFO_AGG - Mutual Information
MWW_TEST - Mann-Whitney-Wilcoxon test of independence
PRINT_MANOVA_ONE_WAY_TEST - Pretty print of Multivariate Analysis of Variance result in one way setting
PRINT_MANOVA_TWO_WAY_TEST - Pretty print of Multivariate Analysis of Variance result in two way setting
SKEWNESS_AGG - Skewness
SPEARMAN_CORR - Spearman Rank Correlation
SPEARMAN_CORR_S - Spearman Rank Correlation as string
SUMMARY1000 - Summary of up to 1000 columns
SUMMARY1000CHAR - Summary of up to 1000 character columns
SUMMARY1000DATE - Summary of up to 1000 date columns
SUMMARY1000INTERVAL - Summary of up to 1000 interval columns
SUMMARY1000NUM - Summary of up to 1000 numeric columns
SUMMARY1000TIME - Summary of up to 1000 time columns
SUMMARY1000TIMESTAMP - Summary of up to 1000 timestamp columns
T_LS_TEST - T-Student test for the linear relationship of two columns
T_LS_TEST_S_AGG - T-Student test between linearly dependent samples as string
T_ME_TEST - T-Student test for the expected mean of a column
T_ME_TEST_S_AGG - T-Student test for the expected mean of a variable as string
T_PMD_TEST - T-Student test for the expected mean difference between two paired columns
T_PMD_TEST_S_AGG - T-Student test for the expected mean difference between two paired variables as string
T_TEST_AGG - T-Student test of a variable split into two classes
T_TEST_S_AGG - T-Student test of a variable split into two classes as string
T_UMD_TEST - T-Student test for the expected mean difference between class values of a column
UNITABLE - Univariate Frequencies
WILCOXON_TEST - Wilcoxon paired sample difference test

Analytics - Time Series
PRINT_TIMESERIES - Print Time Series predictions
TIMESERIES - Predict future values for Time Series

Utilities - Actions
_sp_utl_dropAllAggregates - Drop all UDA aggregate functions with the given prefix
_sp_utl_dropAllFunctions - Drop all UDF functions with the given prefix
_sp_utl_dropAllLike - Drop all database objects with the given LIKE expression
_sp_utl_dropAllProcedures - Drop all stored procedures with the given prefix
_sp_utl_dropAllUDX - Drop all UDx functions with the given prefix
_sp_utl_justExecute - Execute a SQL statement
DROP_TABLE - Drop a table

Utilities - Checking
_sp_utl_aggregateExists - Check if a UDA aggregate function exists
_sp_utl_columnContainsNulls - Check if a column contains at least one NULL value
List of functions by category

_sp_utl_columnExists - Check if a column exists
_sp_utl_columnIsId - Check if a column contains unique values
_sp_utl_columnIsNumeric - Check if a column is of a numeric data type
_sp_utl_columnListExists - Check if a columns exists
_sp_utl_columnsEqualTypes - Check if two columns have the same data type
_sp_utl_functionExists - Check if a UDF function exists
_sp_utl_isTempTable - Check if a table is temporary
_sp_utl_procedureExists - Check if a stored procedure exists
_sp_utl_relationExists - Check if a table or a view exists
_sp_utl_sequenceExists - Check if a sequence exists
_sp_utl_tableExists - Check if a table exists
_sp_utl_viewExists - Check if a view exists
ISDATE_TINY - Check if a string has the compact date format YYYYMMDD
msghelp - Show information related to a message number in the correct language of the locale.

Utilities - Data Exploration
_sp_utl_getColumnType - Column data type
_sp_utl_getTableSize - Table row count

Utilities - Preprocessing
drand64 - 64 bits pseudo-random number generator
Reference Documentation: Analytics

ACC - Classification accuracy

This stored procedure calculates the Classification accuracy, i.e. the ratio of correctly classified predictions.

Usage

The ACC stored procedure has the following syntax:

\[
\text{ACC(NVARCHAR(ANY) paramString)}
\]

Parameters

- **paramString**
  - comma-separated list of \(<\text{parameter}>=<\text{value}>\) entries with parameters below
  - Type: NVARCHAR(ANY)

- **intable**
  - the input table containing real values
  - Type: NVARCHAR(256)

- **id**
  - the input table column in \(<\text{intable}>\) identifying a unique instance id
  - Type: NVARCHAR(128)

- **target**
  - the input table column in \(<\text{intable}>\) containing real values
  - Type: NVARCHAR(128)

- **resulttable**
  - the input table containing predicted values
  - Type: NVARCHAR(256)
The following stored procedure calculates the ratio of correctly classified predictions among the total number of predictions. This is done by comparing the predictions made when applying a Classification model onto data, and the real values for this data.

**Examples**

```sql
CALL nza..TRAIN_TEST('modelType=dectree, model=iris_c45, traintable=nza..iris_train, testtable=nza..iris_test, target=class, id=id, outtable=iris_pred, minsplit=2');
CALL nza..ACC('intable=nza..iris_test, resulttable=iris_pred, id=id, target=class');
CALL nza..DROP_MODEL('model=iris_c45');
CALL nza..DROP_TABLE('iris_pred');
```

```
TRAIN_TEST
############
  0.918919
(1 row)
ACC
-------
  0.918919
(1 row)
DROP_MODEL
---------
  t
(1 row)
```
DROP_TABLE
------------
t
(1 row)

Related Functions
► category Analytics - Diagnostic Measures
► CERROR
► WACC

ALTER_MODEL - Alter properties of an analytics model
This stored procedure alters the properties of the given model

Usage
The ALTER_MODEL stored procedure has the following syntax:

► ALTER_MODEL(NVARCHAR(ANY))
  ▲ Parameters
  ► paramString
    comma-separated list of <parameter>=<value> entries with parameters below
    Type: NVARCHAR(ANY)
  ► model
    the model to be altered
    Type: NVARCHAR(64)
  ► name
    the new name of the model. If not specified, the model is not renamed. Otherwise, the model
    name and the names of the managed tables are changed
    Type: NVARCHAR(64)
    Default: <none>
  ► owner
    the new owner of the model. If not specified, the owner of the model does not change. Other-
    wise, the new owner will get all model privileges.
    Type: NVARCHAR(128)
    Default: <none>
  ► description
    the description for the model. If not specified, the description of the model does not change. Other-
    wise, the given description will be associated to the model.
    Type: NVARCHAR(8192)
Returns: BOOLEAN always true (otherwise an exception is raised)

Details
This stored procedure alters the properties of the given model. The following model properties can be altered:
- the model name,
- the owner,
- the description,
- the copyright statement,
- the application name,
- the application version,
- the user category.
Examples

CALL nza..ARULE('intable=nza..retail, model=mbamodel, supporttype=percent, support=5, lvl=0, maxsetsize=5, confidence=0.5');

CALL nza..ALTER_MODEL('model=mbamodel, description=Market basket analysis');

SELECT modelname, description FROM v_nza_models WHERE upper(modelname)='MBAMODEL';

CALL nza..DROP_MODEL('model=mbamodel');

NOTICE:

RUNNING FPGrowth algorithm:
DATASET : "NZA"."RETAIL"
Transaction column : "TID"
Item column : "ITEM"
Group by : <none>
Minimum support : 5 %
Minimum confidence: 0.5
Max frequent itemset size : 5
Level of conditional dbs : 0
Result tables prefix : "NZA_META_MBAMODEL"

ARULE
------
   14
(1 row)
ALTER_MODEL
------------
   t
(1 row)

MODELNAME | DESCRIPTION
------------+------------------------
MBAMODEL | Market basket analysis

(1 row)

NOTICE: Dropped: MBAMODEL
DROP_MODEL
Related Functions
► category Analytics - Model Management
► LIST_MODELS

ANOVA_CRD_TEST - Analyze variance using Completely Randomized Design and One Way ANOVA

This stored procedure performs a completely randomized design ANOVA

Usage
The ANOVA_CRD_TEST stored procedure has the following syntax:

► ANOVA_CRD_TEST(NVARCHAR(ANY) paramStringstring)
  ▶ Parameters
  ► paramStringstring
    comma-separated list of <parameter>=<value> entries with parameters below
    Type: NVARCHAR(ANY)
  ► intable
    the input table
    Type: NVARCHAR(256)
  ► treatment
    the input table column identifying a unique treatment
    Type: NVARCHAR(128)
  ► incolumn
    the input table observation columns, separated by a semi-colon (;). If several observations are specified, an output table must be specified too.
    Type: NVARCHAR(ANY)
  ► variable
    the observation column. Deprecated, use parameter incolumn instead.
    Type: NVARCHAR(128)
    Default: <none>
  ► by
    the input table column which uniquely identifies a group on which to perform ANOVA.
If specified, an output table must be specified too.
Type: NVARCHAR(ANY)
Default: <none>

**outtable**

the output table. This parameter is required if parameter by is specified and/or multiple observation columns are provided. Otherwise the parameter is ignored.
Type: NVARCHAR(ANY)
Default: <none>

Returns

NVARCHAR(200) The test results or the number of rows in the outtable.

The test result can contain following results: SScbetween, DFBetween, SScwithin, DFWithin, F and P. See the output table column description for more information on them.

**Details**

This stored procedure analyzes the variance of one or several observations for different treatments. It assumes that the input table contains one or several columns with numerical (double) observation results of an experiment concerning treatments indicated by the treatment parameter.

The One-way ANOVA considers independent samples (the treatments) while the Completely Randomized Design considers equally sized "samples". The implementation covers both cases. For more information, see 'Completely Randomized Design'.

The stored procedure returns, among others, the p-value of the F test. If the p-value is larger than 0.95 then it can be said that the hypothesis is incorrect or in this case the treatment had no effect.

If several observation columns are given and/or the treatment are grouped by the column specified in parameter by, an output table is written with following columns: totno, totsu, totmean, totss, sscot, sscbetween, dfbetween, sscwithin, dfwithin, f, p. If parameter by is specified, a column <by> is added. If parameter incolumn is specified, a column incolumn is added containing the column name specified in parameter inColumn.

The output table columns have following meaning:
- totno - the total number of observations,
- totsum - the sum of the observations,
- totmean - the mean value of the observations,
- totss - the sum of squares of the observations,
- sscot - the variance of the observations,
- sscbetween - the sum of squares between the treatments (around the overall mean),
- dfbetween - the number of degrees of freedom between the treatments,
- sscwithin - the sum of squares within the treatments (around the mean of each treatment),
- dfwithin - the sum of the number of degrees of freedom within each treatment,
- f - the value of the F-statistics for the treatments,
- p - the probability that the true F statistics is lower or equal to the F-statistics above (computed from the
Note that the larger the value of \( \text{ssc}_{\text{between}} \) as compared to \( \text{ssc}_{\text{within}} \), the greater the impact of the treatment on the measured variable. \( \text{ssc}_{\text{between}} \) and \( \text{ssc}_{\text{within}} \) sum up to the overall centered sum of squares. F-statistics is the quotient of the variance due to treatment to the rest variance.

**Examples**

```sql
CREATE TABLE wheattest2(fieldId INTEGER, variety CHAR(1), yield DOUBLE, barrels INTEGER);

INSERT INTO wheattest2 VALUES(1,'A',80, 4);
INSERT INTO wheattest2 VALUES(2,'A',65, 3);
INSERT INTO wheattest2 VALUES(3,'A',50, 2);
INSERT INTO wheattest2 VALUES(4,'B',100, 5);
INSERT INTO wheattest2 VALUES(5,'B',85, 4);
INSERT INTO wheattest2 VALUES(6,'B',70, 3);
INSERT INTO wheattest2 VALUES(7,'C',60, 2);
INSERT INTO wheattest2 VALUES(8,'C',75, 3);
INSERT INTO wheattest2 VALUES(9,'C',90, 5);

CALL nza..ANOVA_CRD_TEST('intable=wheattest2, variable=yield, treatment=variety');
CALL nza..ANOVA_CRD_TEST('intable=wheattest2, incolumn=yield; barrels, treatment=variety, outtable=outtab');
SELECT * FROM outtab ORDER BY incolumn;
CALL nza..DROP_TABLE('outtab');
CALL nza..DROP_TABLE('wheattest2');
```

```
<table>
<thead>
<tr>
<th>SScbetween= 600 DFbetween= 2 SScwithin= 1350 DFwithin= 6 F= 1.3333333333333 p= 0.6681838871188</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 row</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>2</th>
</tr>
</thead>
</table>
```
### INCOLUMN | TOTNO | TOTSU | TOTMEAN | TOTSS | SSCTOT | SSCBETWEEN | DFBETWEEN | SSCWITHIN | F | P
---|---|---|---|---|---|---|---|---|---|---
BARRELS | 9 | 31 | 3.4444444444444 | 117 | 10.222222222222 | 1.5555555555555 | 2 | 8.6666666666667 | 6 | 0.538461538461538 | 0.39057491575573 | 1.3333333333333 | 0.6681838871188

| YIELD | 9 | 675 | 75 | 52575 | 1950 | 600 | 2 | 1350 | 6 | 1.3333333333333 | 0.6681838871188 |

### DROP_TABLE

---

`t`

(1 row)

### DROP_TABLE

---

`t`

(1 row)

```sql
CREATE TABLE wheatbytest(fieldId INTEGER, variety CHAR(1), yield DOUBLE, barrels INTEGER, experiment INT4);

INSERT INTO wheatbytest VALUES(1,'A',65,3,1);
INSERT INTO wheatbytest VALUES(2,'A',66,4,1);
INSERT INTO wheatbytest VALUES(3,'A',64,2,1);
INSERT INTO wheatbytest VALUES(4,'B',84,3,1);
INSERT INTO wheatbytest VALUES(5,'B',85,4,1);
INSERT INTO wheatbytest VALUES(6,'B',86,5,1);
INSERT INTO wheatbytest VALUES(7,'C',75,3,1);
```
INSERT INTO wheatbytest VALUES(8,'C',76,4,1);
INSERT INTO wheatbytest VALUES(9,'C',74,2,1);
INSERT INTO wheatbytest VALUES(1,'A',80,4,2);
INSERT INTO wheatbytest VALUES(2,'A',65,3,2);
INSERT INTO wheatbytest VALUES(3,'A',50,2,2);
INSERT INTO wheatbytest VALUES(4,'B',100,5,2);
INSERT INTO wheatbytest VALUES(5,'B',85,4,2);
INSERT INTO wheatbytest VALUES(6,'B',70,3,2);
INSERT INTO wheatbytest VALUES(7,'C',60,2,2);
INSERT INTO wheatbytest VALUES(8,'C',75,3,2);
INSERT INTO wheatbytest VALUES(9,'C',90,5,2);
CALL nza..ANOVA_CRD_TEST('intable=wheatbytest, incolumn=yield, treatment=variety, by=experiment, outtable=outbytab');
SELECT * FROM outbytab ORDER BY experiment;
CALL nza..DROP_TABLE('outbytab');
CALL nza..ANOVA_CRD_TEST('intable=wheatbytest, incolumn=yield;barrels, treatment=variety, by=experiment, outtable=outbytab');
SELECT * FROM outbytab ORDER BY incolumn, experiment;
CALL nza..DROP_TABLE('outbytab');
CALL nza..DROP_TABLE('wheatbytest');

ANOVA_CRD_TEST
----------------
 2
(1 row)

<table>
<thead>
<tr>
<th>EXPERIMENT</th>
<th>TOTNO</th>
<th>TOTSU</th>
<th>TOTMEAN</th>
<th>TOTSS</th>
<th>SSCTOT</th>
<th>SSCBETWEEN</th>
<th>DFBETWEEN</th>
<th>SSCWITHIN</th>
<th>DFWITHIN</th>
<th>F</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>+-----------------+------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>600</td>
<td>9</td>
<td>675</td>
<td>75</td>
<td>51231</td>
<td>606</td>
<td>300</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.99999902940985</td>
<td>2</td>
<td>9</td>
<td>675</td>
<td>75</td>
<td>52575</td>
<td>1950</td>
<td>600</td>
<td>2</td>
<td>1350</td>
<td>6</td>
<td>1.3333333333333</td>
</tr>
</tbody>
</table>

(2 rows)

DROP_TABLE
-----------
t
(1 row)

ANOVA_CRD_TEST
-------------

4
(1 row)

| INCOLUMN | EXPERIMENT | TOTNO | TOTSU | TOTMEAN | TOTSS | SSCTOT | SSCBETWEEN | DFBETWEEN |
|----------|------------+-------+-------+---------+-------+--------+------------+-----------|
| BARRELS  | 1          | 9     | 30    | 3.3333333333333 | 108   | 8      | 2          | 2         |
| BARRELS  | 2          | 9     | 31    | 3.4444444444444 | 117   | 10.222222222222 | 1.5555555555555 | 2         |
| YIELD    | 1          | 9     | 675   | 75      | 51231 | 606    | 2          | 2         |
| YIELD    | 2          | 9     | 675   | 75      | 52575 | 1950   | 600        | 2         |

(4 rows)

DROP_TABLE
Related Functions

► category Analytics - Statistics
► ANOVA_RBD_TEST

ANOVA_RBD_TEST - Analyze variance using ANOVA Randomized Block Design

This stored procedure performs a randomized block design ANOVA

Usage

The ANOVA_RBD_TEST stored procedure has the following syntax:

► ANOVA_RBD_TEST(NVARCHAR(ANY) paramStringstring)

▲ Parameters

► paramStringtring
comma-separated list of <parameter>=<value> entries with parameters below
Type: NVARCHAR(ANY)

► intable
the input table
Type: NVARCHAR(256)

► treatment
the input table column identifying a unique treatment
Type: NVARCHAR(128)

► block
the input table column identifying a unique block of treatments
Type: NVARCHAR(128)
► **incolumn**
the input table observation columns, separated by a semi-colon (;). If several observations are specified, an output table must be specified too.
Type: NVARCHAR(ANY)

► **variable**
the observation column. Deprecated, use parameter incolumn instead.
Type: NVARCHAR(128)
Default: <none>

► **by**
the input table column which uniquely identifies a group on which to perform ANOVA. If specified, an output table must be specified too.
Type: NVARCHAR(ANY)
Default: <none>

► **outtable**
the output table. This parameter is required if parameter by is specified and/or multiple observation columns are provided. Otherwise the parameter is ignored.
Type: NVARCHAR(ANY)
Default: <none>

▲ Returns
NVARCHAR(200) The test results or the number of rows in the outtable.

The test result can contain following results: blSSbetween, blDFbetween, SSwithin, DFwithin, Fbl, Pbl, grSSbetween, grDFbetween, Fgr and Pgr. See the output table column description for more information on them.

**Details**
This stored procedure analyzes the variance of one or several observations for different blocks of treatments. It assumes that the input table contains one or several columns with numerical (double) observation results of an experiment concerning treatments indicated by the treatment parameter. The treatments are performed repeatedly in various blocks, e.g. from different laboratories where experiments are carried out. This means there are independent samples (treatments) repeatedly drawn for each block. In particular, the number of observations for the same treatment in different blocks should be the same.

Generally, we would expect differences between blocks (we split into blocks just to reduce the variance resulting) and are curious about the differences concerning treatments.

The stored procedure returns, among others, the p-value of the F test, related to the block split and the group split of the data. If the p-value is larger than 0.95 then it can be said that the hypothesis is incorrect or in this case the treatment or block split had no effect.

If several observation columns are given and/or the treatment are grouped by the column specified in parameter by, an output table is written with following columns: b1SSbetween, b1DFbetween, SSwithin, DFwithin, F1, P1, g1SSbetween, g1DFbetween, Fgr and Pgr. If parameter by is specified, a column <by> is added. If parameter incolumn is specified, a column incolumn is added containing the column name specified in parameter incolumn.

The output table columns have following meaning:
- bllsscbetween - the sum of squares between the blocks of treatments (around the overall mean),
- blldfbetween - the number of degrees of freedom between the blocks of treatments,
- sllsscwithin - the sum of squares within the treatments (around the mean of each treatment),
- dfwithin - the sum of the number of degrees of freedom within each treatment,
- grllsscbetween - the sum of squares between the groups (around the overall mean),
- grdfbetween - the number of degrees of freedom between the groups,
- fbl - the value of the F-statistics for the blocks of treatments,
- pbl - the probability that the true F statistics is lower or equal to the F-statistics above (computed from the sample),
- fgr - the value of the F-statistics for the groups,
- pgr - the probability that the true F statistics is lower or equal to the F-statistics above (computed from the sample).

Examples

```
CREATE TABLE textiles(chemical NVARCHAR(4), bolt INT4, concentration DOUBLE, ph DOUBLE);
INSERT INTO textiles VALUES('A',1,65,10.1);
INSERT INTO textiles VALUES('A',2,50,12.2);
INSERT INTO textiles VALUES('A',3,45,11.9);
INSERT INTO textiles VALUES('B',1,42,11.4);
INSERT INTO textiles VALUES('B',2,30,12.9);
INSERT INTO textiles VALUES('B',3,22,12.7);
INSERT INTO textiles VALUES('C',1,44,9.9);
INSERT INTO textiles VALUES('C',2,80,12.3);
INSERT INTO textiles VALUES('C',3,76,11.4);
INSERT INTO textiles VALUES('D',1,79,12.1);
INSERT INTO textiles VALUES('D',2,99,13.4);
INSERT INTO textiles VALUES('D',3,86,12.9);
CALL nza..ANOVA_RBD_TEST('intable=textiles, incolumn=ph, treatment=chemical, block=bolt');
CALL nza..ANOVA_RBD_TEST('intable=textiles, incolumn=ph;concentration, treatment=chemical, block=bolt, outtable=outtab');
SELECT * FROM outtab ORDER BY incolumn;
CALL nza..DROP_TABLE('outtab');
```
CALL nza..DROP_TABLE('textiles');

ANOVA_RBD_TEST
---------------------------------------------------------------
---------------------------------------------------------------
---------------------------------------------------------------
--------------
blSScbetween= 7.1716666666662 blDFbetween= 2 SScwithin= 0.53500000000031 DFwithin= 6 Fbl= 40.214953271002 pbl= 0.999665449548 grSScbetween= 5.2 grDFbetween= 3 Fgr= 19.439252336438 pgr= 0.99828746900889

(1 row)

ANOVA_RBD_TEST
----------------
2

(1 row)

INCOLUMN | BLSSCBETWEEN | BLDFBETWEEN | SScWITHIN
| DFWITHIN | FBL | PBL |
GRSSCBETWEEN | GRDFBETWEEN | FGR | PGR
---------------+-----------------+-------------
+-----------------+-----------------+-------------
+------------------+-----------------+-------------
+-----------------+------------------+

CONCENTRATION | 145.16666666666 | 2 | 1258.8333333334 | 6 | 0.34595524956968 |
0.27921891773379 | 5083.6666666667 | 3 | 8.0767906792002 | 0.98422520011584

PH | 7.1716666666671 | 2 | 0.53499999999994 | 6 | 40.214953271075 |
0.999665449548 | 5.2 | 0.99828746900833 |
19.439252336471 | 0.99828746900833 |

(2 rows)

DROP_TABLE
----------

| t

(1 row)

DROP_TABLE
----------

| t
CREATE TABLE bytextiles(chemical NVARCHAR(4), bolt INT4, concentration DOUBLE, ph DOUBLE, experiment INT4);

INSERT INTO bytextiles VALUES('A',1,65,10.1,1);
INSERT INTO bytextiles VALUES('A',2,50,12.2,1);
INSERT INTO bytextiles VALUES('A',3,45,11.9,1);
INSERT INTO bytextiles VALUES('B',1,42,11.4,1);
INSERT INTO bytextiles VALUES('B',2,30,12.9,1);
INSERT INTO bytextiles VALUES('B',3,22,12.7,1);
INSERT INTO bytextiles VALUES('C',1,44,9.9,1);
INSERT INTO bytextiles VALUES('C',2,80,12.3,1);
INSERT INTO bytextiles VALUES('C',3,76,11.4,1);
INSERT INTO bytextiles VALUES('D',1,79,12.1,1);
INSERT INTO bytextiles VALUES('D',2,99,13.4,1);
INSERT INTO bytextiles VALUES('D',3,86,12.9,1);
INSERT INTO bytextiles VALUES('A',1,100-65,15-10.1,2);
INSERT INTO bytextiles VALUES('A',2,100-50,15-12.2,2);
INSERT INTO bytextiles VALUES('A',3,100-45,15-11.9,2);
INSERT INTO bytextiles VALUES('B',1,100-42,15-11.4,2);
INSERT INTO bytextiles VALUES('B',2,100-30,15-12.9,2);
INSERT INTO bytextiles VALUES('B',3,100-22,15-12.7,2);
INSERT INTO bytextiles VALUES('C',1,100-44,15-9.9,2);
INSERT INTO bytextiles VALUES('C',2,100-80,15-12.3,2);
INSERT INTO bytextiles VALUES('C',3,100-76,15-11.4,2);
INSERT INTO bytextiles VALUES('D',1,100-79,15-12.1,2);
INSERT INTO bytextiles VALUES('D',2,100-99,15-13.4,2);
INSERT INTO bytextiles VALUES('D',3,100-86,15-12.9,2);

CALL nza..ANOVA_RBD_TEST('intable=bytextiles, incolumn=ph, treatment=chemical, block=bolt, by=experiment, outtable=outbytab');

SELECT * FROM outbytab ORDER BY experiment;
CALL nza..DROP_TABLE('outbytab');
call nza..ANOVA_RBD_TEST('intable=bytextiles, incolumn=ph;concentration, treatment=chemical, block=bolt, by=experiment, outtable=outbytab');
SELECT * FROM outbytab ORDER BY incolumn, experiment;
CALL nza..DROP_TABLE('outbytab');
CALL nza..DROP_TABLE('bytextiles');

ANOVA_RBD_TEST
--------------

2
(1 row)

| EXPERIMENT | BLSSCBETWEEN | BLDFBETWEEN | SSCWITHIN |
|            | DFWITHIN | FBL | PBL | GRSSCBETWEEN |
|            | GRDFBETWEEN | FGR | PGR |

EXPERIMENT | BLSSCBETWEEN | BLDFBETWEEN | SSCWITHIN |
           | DFWITHIN | FBL | PBL | GRSSCBETWEEN |
|------------|----------|-----------------+-------------+------------------|
| 1          | 7.1716666666671 |           2 | 0.53499999999963 |
| 3          | 19.439252336462 | 0.99828746900889 |
| 2          | 7.1716666666667 | 2 | 0.53500000000004 |
| 3          | 19.439252336447 | 0.99828746900889 |

(2 rows)

DROP_TABLE
----------

t
(1 row)

ANOVA_RBD_TEST
--------------

4
(1 row)

| INCOLUMN | EXPERIMENT | BLSSCBETWEEN | BLDFBETWEEN | SSCWITHIN | DFWITHIN | FBL | PBL |
|          |           | GRSSCBETWEEN | GRDFBETWEEN | FGR | PGR |

INCOLUMN | EXPERIMENT | BLSSCBETWEEN | BLDFBETWEEN | SSCWITHIN | DFWITHIN | FBL | PBL | GRSSCBETWEEN | GRDFBETWEEN | FGR | PGR |
|----------|------------|-----------------+-------------+-----------------+----------|-----------------+-------------+-----------------+----------|-----------------+-----------------+-------------+-----------------|----------|-----------------+-------------+-----------------|----------|-----------------+-------------+-----------------|----------|-----------------+-------------+-----------------|----------|-----------------+-------------+-----------------|----------|-----------------+-------------+-----------------|----------|-----------------+-------------+-----------------|----------|-----------------+-------------+-----------------|----------|-----------------+-------------+-----------------|----------|-----------------+-------------|
+-----------------+------------------+
| CONCENTRATION | 1          | 145.16666666666 |
| 2 | 1258.8333333333 | 6 | 0.34595524956968 |
| 0.27921891773379 | 5083.6666666667 | 3 | 8.0767906792003 |
| 0.98422520011997 |

| CONCENTRATION | 2          | 145.16666666666 |
| 2 | 1258.8333333333 | 6 | 0.34595524956972 |
| 0.27921891773382 | 5083.6666666667 | 3 | 8.0767906792004 |
| 0.98422520011997 |

| PH            | 1          | 7.1716666666671 |
| 2 | 0.53499999999963 | 6 | 40.214953271058 |
| 0.999665449548 | 5.2 | 3 | 19.439252336462 |
| 0.99828746900889 |

| PH            | 2          | 7.1716666666666 |
| 2 | 0.53500000000005 | 6 | 40.214953271024 |
| 0.999665449548 | 5.2 | 3 | 19.439252336447 |
| 0.99828746900889 |

(4 rows)

DROP_TABLE
-----------
t
(1 row)

DROP_TABLE
-----------
t
(1 row)

Related Functions
► category Analytics - Statistics
► ANOVA_CRD_TEST
► ANOVA_CRD_TEST

APPLY_DISC - Discretize data using given discretization bin limits
This stored procedure discretize numeric input data columns according to the given limits for discretization bins
Usage

The APPLY_DISC stored procedure has the following syntax:

► APPLY_DISC(NVARCHAR(ANY) paramString)

▲ Parameters

► paramString
  comma-separated list of <parameter>=<value> entries with parameters below
  Type: NVARCHAR(ANY)

► intable
  the input table or view
  Type: NVARCHAR(256)

► outtable
  the output table or view to store the discretized data into
  Type: NVARCHAR(256)

► btable
  the input table containing the discretization bin limits for numeric columns of the input table to be discretized. This table contains following columns: colname, break.
  Type: NVARCHAR(256)

► view
  a flag indicating whether the output object should be stored as a view (true) or as a table (false)
  Type: BOOLEAN
  Default: false

► replace
  a flag indicating whether the discretized columns should replace the original columns (true) or should be added with another name (false). The name of the columns is then prefixed with 'disc_'.
  Type: BOOLEAN
  Default: true

▲ Returns

INTEGER the number of discretized columns

Details

This stored procedure discretized numeric input table columns using the given discretization bin limits for the column. The original column value is transformed to the index of the discretization bin to which this value belongs.

The discretization bin limits are available in <btable> in following columns: colname, break. The column colname references the name of the input table columns, and the column break contains a bin limit for this column.

Only input table columns that have bin limits defined in <btable> are discretized.

- If the flag replace=true, the output table contains the input table columns that are not discretized and the
discretized columns with the same name as in the input table.
- If the flag replace=false, the output table contains all input table columns with their original name, and the discretized column with their name prefixed with 'disc_'.

Examples

```sql
CALL nza..EWDISC('outtable=disc_bounds,
intable=nza..disc_iris,
incolumn=petal_length:5;sepal_width:4');
SELECT * FROM disc_bounds order by colname, break;
CALL nza..APPLY_DISC('outtable=disc_extended,
intable=nza..disc_iris, btable=disc_bounds, replace=F');
SELECT * FROM disc_extended order by 1,2,3,4 limit 10;
CALL nza..APPLY_DISC('outtable=disc_replaced,
intable=nza..disc_iris, btable=disc_bounds, replace=T');
SELECT species, petal_width, petal_length, sepal_width,
sepal_length FROM disc_replaced order by 1,2,3,4,5 limit 10;
CALL nza..DROP_TABLE('disc_bounds');
CALL nza..DROP_TABLE('disc_extended');
CALL nza..DROP_TABLE('disc_replaced');
```

EWDISC
-------

2
(1 row)

<table>
<thead>
<tr>
<th>COLNAME</th>
<th>BREAK</th>
</tr>
</thead>
<tbody>
<tr>
<td>PETAL_LENGTH</td>
<td>2.18</td>
</tr>
<tr>
<td>PETAL_LENGTH</td>
<td>3.36</td>
</tr>
<tr>
<td>PETAL_LENGTH</td>
<td>4.54</td>
</tr>
<tr>
<td>PETAL_LENGTH</td>
<td>5.72</td>
</tr>
<tr>
<td>SEPAL_WIDTH</td>
<td>2.6</td>
</tr>
<tr>
<td>SEPAL_WIDTH</td>
<td>3.2</td>
</tr>
<tr>
<td>SEPAL_WIDTH</td>
<td>3.8</td>
</tr>
</tbody>
</table>
(7 rows)

**APPLY_DISC**

------------

2

(1 row)

<table>
<thead>
<tr>
<th>SEPAL_LENGTH</th>
<th>SEPAL_WIDTH</th>
<th>PETAL_LENGTH</th>
<th>PETAL_WIDTH</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPECIES</td>
<td>DISC_PETAL_LENGTH</td>
<td>DISC_SEPAL_WIDTH</td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------</td>
<td>------------------</td>
<td></td>
</tr>
<tr>
<td>setosa</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4.3</td>
<td>3</td>
<td>1.1</td>
<td>0.1</td>
</tr>
<tr>
<td>setosa</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4.4</td>
<td>2.9</td>
<td>1.4</td>
<td>0.2</td>
</tr>
<tr>
<td>setosa</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4.4</td>
<td>3</td>
<td>1.3</td>
<td>0.2</td>
</tr>
<tr>
<td>setosa</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4.4</td>
<td>3.2</td>
<td>1.3</td>
<td>0.2</td>
</tr>
<tr>
<td>setosa</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4.5</td>
<td>2.3</td>
<td>1.3</td>
<td>0.3</td>
</tr>
<tr>
<td>setosa</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4.6</td>
<td>3.1</td>
<td>1.5</td>
<td>0.2</td>
</tr>
<tr>
<td>setosa</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4.6</td>
<td>3.2</td>
<td>1.4</td>
<td>0.2</td>
</tr>
<tr>
<td>setosa</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4.6</td>
<td>3.4</td>
<td>1.4</td>
<td>0.3</td>
</tr>
<tr>
<td>setosa</td>
<td>1</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4.6</td>
<td>3.6</td>
<td>1</td>
<td>0.2</td>
</tr>
<tr>
<td>setosa</td>
<td>1</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4.7</td>
<td>3.2</td>
<td>1.3</td>
<td>0.2</td>
</tr>
<tr>
<td>setosa</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

(10 rows)

**APPLY_DISC**

------------

2

(1 row)
<table>
<thead>
<tr>
<th>SPECIES</th>
<th>PETAL_WIDTH</th>
<th>PETAL_LENGTH</th>
<th>SEPAL_WIDTH</th>
<th>SEPAL_LENGTH</th>
</tr>
</thead>
<tbody>
<tr>
<td>setosa</td>
<td>0.1</td>
<td>1</td>
<td>2</td>
<td>4.3</td>
</tr>
<tr>
<td>setosa</td>
<td>0.1</td>
<td>1</td>
<td>2</td>
<td>4.8</td>
</tr>
<tr>
<td>setosa</td>
<td>0.1</td>
<td>1</td>
<td>2</td>
<td>4.9</td>
</tr>
<tr>
<td>setosa</td>
<td>0.1</td>
<td>1</td>
<td>2</td>
<td>4.9</td>
</tr>
<tr>
<td>setosa</td>
<td>0.1</td>
<td>1</td>
<td>2</td>
<td>4.9</td>
</tr>
<tr>
<td>setosa</td>
<td>0.1</td>
<td>1</td>
<td>2</td>
<td>5.2</td>
</tr>
<tr>
<td>setosa</td>
<td>0.2</td>
<td>1</td>
<td>2</td>
<td>4.4</td>
</tr>
<tr>
<td>setosa</td>
<td>0.2</td>
<td>1</td>
<td>2</td>
<td>4.4</td>
</tr>
<tr>
<td>setosa</td>
<td>0.2</td>
<td>1</td>
<td>2</td>
<td>4.4</td>
</tr>
<tr>
<td>setosa</td>
<td>0.2</td>
<td>1</td>
<td>2</td>
<td>4.6</td>
</tr>
</tbody>
</table>

(10 rows)

```
DROP_TABLE
---------
t
(1 row)
```

```
DROP_TABLE
---------
t
(1 row)
```
Related Functions

- category Analytics - Discretization
- EFDISC
- EMDISC
- EWDISC
- EWDISC_NICE

ARULE - Build an Association Rules model

This stored procedure builds an Association Rule model by using the FP-growth algorithm. The procedure extracts sets of items that are frequently used by transactions. It also implies rules by associating these items. The algorithm is concurrently executed on independent parts of the data, where the level of parallelism is controlled by the lvl parameter. Be careful when you specify the lvl parameter because it affects the consumption of temporary database space and the main memory in the nodes.

Usage

The ARULE stored procedure has the following syntax:

```
ARULE(NVARCHAR(ANY) paramString)
```

Parameters

- **paramString**
  comma-separated list of `<parameter>=<value>` entries with parameters below
  Type: NVARCHAR(ANY)

- **model**
  the name of the Association Rules model to build
  Type: NVARCHAR(ANY)

- **intable**
  the input table
  Type: NVARCHAR(256)

- **tid**
  the input table column identifying transactions
  Type: NVARCHAR(128)
Default: tid

- **item**
  the input table column identifying items in transactions
  Type: NVARCHAR(128)
  Default: item

- **by**
  the input table column identifying groups of transactions if any. Association Rules mining is done separately on each of these groups. Leave the parameter undefined if no groups are to be considered.
  Type: NVARCHAR(128)
  Default: <none>

- **lvl**
  the level of splitting data into independent parts - conditional data sets.
  ARULE first temporarily redistributes the data into overlapping parts in such a way, that each part can be processed in parallel without communication between the SPUs. Note that for this to work, there can be redundancy between the parts, such that the accumulative size of the temporary parts can be much higher than the one of the original data set. The parameter lvl controls how many parts are created. The higher lvl:
  - The more computation and temporary database space is required for the splitting
  - The smaller the amount of main memory that is required for each data slice
  Note: To fully use the benefits of parallel computing, do not specify the value of the lvl parameter too low. Additionally, the lower the value of the lvl parameter, the higher the memory consumption for each part. The higher memory consumption might cause an out-of-memory error on the SPUs. If an out-of-memory error on the SPUs occurs, increase the lvl parameter.
  
  If you specify the value 0, the algorithm is executed in a serial way for each data set group. However, only if the data set fits in one node, and only if the splitting increases the total number of rows dramatically, the stored procedure might be executed faster when you specify the value 0.
  Type: INTEGER
  Default: 1
  Min: 0
  Max: 64

- **maxsetsize**
  the maximum number of items occurring in mined association rules (identical with maximum number of items in frequent itemset to be extracted by FPGrowth algorithm).
  Type: INTEGER
Default: 6
Min: 1
Max: 64

► support
minimum support value satisfied by all association rules. According to supporttype, it defines the absolute number (#supporting transactions) or the percentage of transactions (#supporting transactions/#total transactions*100). Too low minimum support increases the number of generated rules and the computational expense.

Type: DOUBLE
Default: 5 if supporttype=percent, undefined if supporttype=absolute
Min: 0.0 (excluded)
Max: 100.0 if supporttype=percent, no limit otherwise

► supporttype
the type how the minimum support should be interpreted. The following values are allowed: absolute, percent.

NOTE: The support and supporttype values are common to all groups in the dataset. E.g. if 3 is the absolute minimum support, then an itemset will be considered frequent if at least 3 transactions contain its items, no matter what is the number of transactions in this group. Use supporttype=percent to indicate a minimum support depending on the size of the groups. Specifying supporttype=absolute takes effect only if a support is explicitly supplied.

Type: VARCHAR(64)
Default: percent

► confidence
the minimum confidence for an association rule to be extracted.

Type: DOUBLE
Default: 0.5
Min: 0.0
Max: 1.0

▲ Returns
INT8 number of association rules discovered (for all groups).

Details
This stored procedure builds an Association Rules model. The model is saved to the database in a set of tables and registered in the database model metadata. Use the stored procedure PRINT_ARULE to display the Association Rules of the model, or the Model Management functions to further manipulate the model or access the model tables.

Examples

CALL nza..ARULE('intable=nza..retail, model=assoc, supporttype=percent, support=5, lvl=0, maxsetsize=5, confidence=0.5');
CALL nza..PRINT_ARULE('model=assoc, minsize=2, 
minsupp=0.05, minconf=0.6, minlift=1.3, minconv=1.3');

CALL nza..PMML_MODEL('model=assoc, outtable=ARule_pmml');

CALL nza..DROP_MODEL('model=assoc');

CALL nza..DROP_TABLE('ARule_pmml');

ARULE
-------

14

(1 row)

PRINT_ARULE

----------------------------------
----------------------------------
----------------------------------
----------------------------------
----------------------------------
----------------------------------
----------------------------------
----------------------------------
----------------------------------
----------------------------------
----------------------------------
----------------------------------
----------------------------------

--

GRP=| {41} -> {39} [supp=0.12946620993172, 
conf=0.76373369019739, lift=1.328708230788, 
conv=1.7996889669349]

GRP=| {32,39} -> {48} [supp=0.061273564574306, 
conf=0.63891188645772, lift=1.3368399129912, 
conv=1.4458326995372]

GRP=| {38,48} -> {39} [supp=0.069213493341803, 
conf=0.7681268821752, lift=1.336513116731, 
conv=1.8337868834446]

GRP=| {41,48} -> {39} [supp=0.083550736144824, 
conf=0.81681082279885, lift=1.4210493489806, 
conv=2.3211298697154]

GRP=| {39,41} -> {48} [supp=0.083550736144824, 
conf=0.64534781846855, lift=1.3503062625329, 
conv=1.4720704655452]

(1 row)
Related Functions

- category Analytics - Association Rules
- Error: Reference source not found
- PRINT_ARULE
- LIST_MODELS

BITABLE - Bivariate Frequencies

This stored procedure creates a bivariate frequency table for two columns of the input table.

Usage

The BITABLE stored procedure has the following syntax:

- **BITABLE**(NVARCHAR(ANY) paramString)
  ▲ Parameters
  - **paramString**
    comma-separated list of <parameter>=<value> entries with parameters below
    Type: NVARCHAR(ANY)
  - **intable**
    the input table or view
    Type: NVARCHAR(256)
incolumn
the input table columns to consider, separated by a semi-colon (;). Only first two columns will be used to construct bivariate frequency table.
Type: NVARCHAR(ANY)

outtable
the output table where the bivariate frequencies are written to
Type: NVARCHAR(ANY)

freq
flag indicating whether frequencies should be attached to the output table
Type: BOOLEAN
Default: false

cum
flag indicating whether cumulative frequencies should be attached to the output table
(setting this flag automatically sets freq flag as frequencies have to be calculated prior to cumulative frequencies)
Type: BOOLEAN
Default: false

▲ Returns
INTEGER the number of bivariate frequencies calculated

Details
This stored procedure counts the frequency of all pairs of distinct values for two columns of the input table.
The bivariate frequencies are stored in the output table with following columns: <incolumn1>, <incolumn2>, count.
Two additional columns freq (frequency) and cum (cumulative frequency) can be available in the output table if the corresponding parameters are set to true.

Examples
CALL
nza..BITABLE('intable=nza..censusincome,incolumn=class_of_worker;major_occupation_code,outtable=gdm_jf1,freq=t,cum =t');

SELECT sum(count) = (SELECT count(*) FROM nza..censusincome) eq FROM gdm_jf1;

SELECT count(*) = (SELECT count(distinct class_of_worker)*count(distinct major_occupation_code) FROM nza..censusincome) eq FROM gdm_jf1;

CALL nza..DROP_TABLE('gdm_jf1');
Related Functions

- category Analytics - Statistics
- UNITABLE
- HIST

**BTBNET_GROW - Build a tree-like Bayesian Network model**

This stored procedure builds a tree-like Bayesian Network for continuous and discrete variables. A spanning tree is constructed joining all the variables or the value pairs on grounds of most strong correlations. This gives the user an overview of most significant interrelations governing the whole set of variables.

**Usage**

The BTBNET_GROW stored procedure has the following syntax:

```sql
BTBNET_GROW(NVARCHAR(ANY) paramString)
```
Parameters

► **paramString**
comma-separated list of `<parameter>=<value>` entries with parameters below
Type: NVARCHAR(ANY)

► **model**
the Bayesian Network model to build
Type: NVARCHAR(ANY)

► **intree**
the input table
Type: NVARCHAR(256)

► **incolumn**
the input table columns to consider, separated by a semi-colon (;). Each column name may be followed by a colon (:) and either:
- a value for a discrete column,
- a boolean expression for a numeric column.
  . The columns followed by a colon (:) are transformed into numeric input columns taking a value 0 or 1. The value 1 indicates that the column has the given discrete value or that the boolean expression is true.
Type: NVARCHAR(ANY)

► **baseidx**
the numeric id to be assigned to the first variable
Type: int4
Default: 777

► **samplesize**
the sample size to take if the number of records is too large
Type: int4
Default: 330000

► **talk**
if talk=yes then additional information on progress will be displayed
Type: NVARCHAR(ANY)
Default: <none>

► **edgelabsort**
if edgelabsort=yes then the left end of the edge will have a name lower in alphabetic order than the right one
Type: NVARCHAR(ANY)
Default: <none>

Returns
NVARCHAR(3000) a termination message

Details
This stored procedure builds a Bayesian Network over the given input table columns. The numeric columns are taken as is, the discrete columns specified with a colon (<column>:<value or boolexpr>) are transformed into numeric columns. Only the numeric columns, either from the input table or transformed, are considered for the model. The model is stored as a set of edges, storing the correlation between two columns and the mean and standard deviation of both columns.

A sampling approach is taken to overcome the computational complexity. If parameter talk=yes, the stored procedure displays notices during the computation. This can be useful for a larger number of columns.

The model can be applied using stored procedure TBNET_APPLY. To apply the model on similar data and to recreate the same transformation as during the creation of the model, get the expression <expr> stored in table nza_meta_<model>_<callcodes> (1 row, 1 column) and create a view on the apply input table as follows: select <expr> from <intable>. The stored procedure TBNET_APPLY should then be called on this view.

Examples

CALL nza..BTBNET_GROW('model=struc_iris, intable=nza..iris, incolumn=SEPALLENGTH;SEPALWIDTH:">3";PETALLENGTH;PETALWIDTH;class:"virginica";class:"setosa",coldefrole=ignore');

SELECT varxname, varyname, corr::numeric(6,4) AS corr FROM nza_meta_struc_iris_model order by 1,2;

CALL nza..DROP_MODEL('model=struc_iris');

BTBNET_GROW

Over node no6 in BTBNet_Grow process

(1 row)

<table>
<thead>
<tr>
<th>VARXNAME</th>
<th>VARYNAME</th>
<th>CORR</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASS_4</td>
<td>SEPALWIDTH_3</td>
<td>0.5594</td>
</tr>
<tr>
<td>PETALLENGTH</td>
<td>CLASS_4</td>
<td>-0.9227</td>
</tr>
<tr>
<td>PETALLENGTH</td>
<td>PETALWIDTH</td>
<td>0.9628</td>
</tr>
<tr>
<td>PETALLENGTH</td>
<td>SEPALLENGTH</td>
<td>0.8718</td>
</tr>
<tr>
<td>PETALWIDTH</td>
<td>CLASS_5</td>
<td>0.7691</td>
</tr>
</tbody>
</table>

(5 rows)
IBM Netezza In-Database Analytics Reference Guide

<table>
<thead>
<tr>
<th>VARNAME</th>
<th>ORGVARNAME</th>
<th>THRESHOLD</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASS_4</td>
<td>CLASS</td>
<td>setosa</td>
</tr>
<tr>
<td>CLASS_5</td>
<td>CLASS</td>
<td>virginica</td>
</tr>
<tr>
<td>SEPALWIDTH_3</td>
<td>SEPALWIDTH</td>
<td>&gt;3</td>
</tr>
</tbody>
</table>

(3 rows)

THECALL

PETALLENGTH,PETALWIDTH,SEPALLength,case when SEPALWIDTH>3 then 1 else 0 end as SEPALWIDTH_3,case when CLASS='setosa' then 1 else 0 end as CLASS_4,case when CLASS='virginica' then 1 else 0 end as CLASS_5

(1 row)

DROP_MODEL

(1 row)

Related Functions

- category Analytics - Regression
- TBNET1G
- TBNET2G
- TBNET_APPLY

CANONICAL_CORR - Canonical Correlation

This stored procedure calculates the canonical correlation between two sets of numeric input columns.
Usage
The CANONICAL_CORR stored procedure has the following syntax:

► **CANONICAL_CORR(NVARCHAR(ANY) paramString)***

▲ **Parameters**

► **paramString**
  comma-separated list of <parameter>=<value> entries with parameters below
  
  Type: NVARCHAR(ANY)

► **intable**
  the input table
  
  Type: NVARCHAR(256)

► **outtable**
  the output table to write the correlations into
  
  Type: NVARCHAR(256)

► **incolumn**
  the numeric input table columns to be correlated, separated by a semicolon (;). Each column is followed by :X or :Y to indicate it belongs to set X or set Y.
  
  Type: NVARCHAR(ANY)

► **by**
  the input table column which splits the data into groups for which the operation is to be performed
  
  Type: NVARCHAR(ANY)

  Default: <none>

▲ **Returns**

INT8 the number of correlations written into the output table

Details
This stored procedure calculates the canonical correlation of two sets of input columns, either in the whole input table or within the groups defined in the column specified by parameter <by>. Canonical correlation is a measure saying how easily it is to predict one column set from the other. It takes a value between -1 (inversely correlated) and 1 (correlated), 0 means that the two sets of columns are independent.

The output table is created with following column: canonical_correlation. If the parameter by is specified, an additional column <by> is added to indicate for which group the correlations have been calculated. The output column canonical_correlation is a string containing the correlation between the two sets of input columns is given as well as two vectors a and b containing the coefficients of the variables of sets X and Y respectively. Given two input column X1 and X2 in set X, and two input columns Y1 and Y2 in set Y, the given correlation between set X and Y is the maximum correlation found between a combination of X1 and X2 and a combination of Y1 and Y2. The maximum correlation is found between a[1] * X1 + a[2] * X2 and b[1] * Y1 + b[2] * Y2. You can check the correlation using following statement: SELECT nza..CORR_AGG((a[1] * X1 + a[2] * X2)::DOUBLE, (b[1] * Y1 + b[2] * Y2)::DOUBLE) FROM <intable>;

PLEASE note that if the variables (columns) are functionally dependent (that is one can be computed as a
combination of the others, especially a linear combination of them) then the output of the procedure can be (in fact will be) non-deterministic. Therefore the user is advised not to use functionally dependent variables.

Examples

```sql
CALL nza..CANONICAL_CORR('intable=nza..CensusIncome,
incolumn=age:X;wage_per_hour:Y, outtable=resultTable');
SELECT * FROM resultTable;

CALL nza..DROP_TABLE('resultTable');

CANONICAL_CORR
----------------
1

(CANONICAL CORRELATION
-----------------------

correlation= 0.0369383,
a= "                1      ",
b= "       0.00299795      

(1 row)

DROP_TABLE
------------
t

(CANONICAL CORRELATION
-----------------------

correlation= 0.7914334,
a= "                1      ",
b= " 0.00247572       

(1 row)

CALL nza..CANONICAL_CORR('intable=nza..iris,
incolumn=sepal_length:X;sepal_width:X;petal_length:Y;petal_width:Y, outtable=resultTable');
SELECT * FROM resultTable;
CALL nza..DROP_TABLE('resultTable');

CANONICAL_CORR
```
1

(1 row)

----------------

CANONICAL_CORRELATION

---------------------------------------------------------------
---------------------------------------------------------------
correlation= 0.940897,
  a=   "         0.780366              -0.625323       ",
  b=   "          0.58457              -0.481924       

(1 row)

DROP_TABLE

-------------

t

(1 row)

CALL nza..CANONICAL_CORR('intable=nza..CensusIncome, incolumn=age:X;wage_per_hour:Y, outtable=resultTable, by=sex');
SELECT * FROM resultTable ORDER BY sex;
CALL nza..DROP_TABLE('resultTable');

----------------

CANONICAL_CORRELATION

----------------------------------------------
----------------------------------------------
correlation= 0.0253748,
  a=   "                1      ",
  b=   "       0.00233639      

<table>
<thead>
<tr>
<th>SEX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Female</td>
</tr>
</tbody>
</table>

|  1 |

(1 row)
correlation= 0.0507014,
a= "              1      ",
b= "       0.00365224      
| Male
   (2 rows)

DROP_TABLE
-------------
t
   (1 row)

Related Functions
► category Analytics - Statistics
► CORR
► CORRELATION1000MATRIX
► CORRELATION500PAIRS

CANONICAL_CORR_AGG - Canonical Correlation

This function calculate the canonical correlation between two sets of numeric input variables. It takes as arguments the number of variables in the set X, the number of variables in the set Y, and the variables themselves, and computes the highest correlation between linear combinations within both groups.

Usage
The CANONICAL_CORR_AGG aggregate has the following syntax:

► CANONICAL_CORR_AGG(INT4 noX, INT4 noY, DOUBLE X1, DOUBLE X2, DOUBLE Y1, DOUBLE Y2, DOUBLE Y3)
  ▶ Parameters
    ▶ noX
      the number of variables in set X
      Type: INT4
      Min: 1
      Max: 61-noY
noY
the number of variables in set Y
Type: INT4
Min: 1
Max: 61

X1,X2,etc.
the first, second, etc... variable in set X
Type: DOUBLE

Y1,Y2,etc.
the first, second, etc... variable in set Y
Type: DOUBLE

Returns
NVARCHAR(10000) the string with the canonical correlation and the correlation coefficients for the variables of sets X and Y

Details
This function calculates the canonical correlation of two sets of numeric input variables. Canonical correlation is a measure saying how easily it is to predict one variable set from the other. It takes a value between -1 (inversely correlated) and 1 (correlated), 0 means that the two sets of variables are independent.

The major limitation is that the number of input variables cannot exceed 61.

This function returns a string containing the correlation between the two sets of input variables as well as two vectors a and b containing the coefficients of the variables of sets X and Y respectively. Given two input variables X1 and X2 in set X, and two input variables Y1 and Y2 in set Y, the given correlation between set X and Y is the maximum correlation found between a combination of X1 and X2 and a combination of Y1 and Y2. The maximum correlation is found between a[1] * X1 + a[2] * X2 and b[1] * Y1 + b[2] * Y2. You can check the correlation using following statement: SELECT nza..CORR_AGG((a[1] * X1 + a[2] * X2)::DOUBLE, (b[1] * Y1 + b[2] * Y2)::DOUBLE) FROM <intable>;

PLEASE note that if the variables (columns) are functionally dependent (that is one can be computed as a combination of the others, especially a linear combination of them ) then the output of the function can be (in fact will be) non-deterministic. Therefore the user is advised not to use functionally dependent variables.

Examples
CREATE TABLE my2table AS SELECT 0.4*petallength+2.5 as x1, 0.6*petallength+2.5 as x2,0.3*sepallength+2.5 as y1, 0.2*sepallength+2.5 as y2, 0.5*sepallength+2.5 as y3 FROM nza..iris;
SELECT nza..CANONICAL_CORR_AGG(2,3,X1,X2,Y1,Y2,Y3) FROM my2table;
SELECT nza..CORR_AGG(0.832525*x1-0.553988*x2, 0.00131594*y1+0.00210631*y2+0.000660639*y3) FROM my2table;
CALL nza..DROP_TABLE('my2table');
**CANONICAL_CORR_AGG**

```
correlation= 0.871754,

a= " 0.832525 -0.553988 ",
b= " 0.00131594 0.00210631 0.000660639 "
```

(1 row)

**CORR_AGG**

```
0.87175415726467
```

(1 row)

**DROP_TABLE**

```
t
```

(1 row)

---

**Related Functions**

- category Analytics - Statistics
- CANONICAL_CORR
- CORR_AGG

---

**CBFS_AGG - Correlation-based feature selection**

This function calculates the N best predictors for a numeric target variable out of several numeric input variables. It checks for each variable the contribution expressed by its partial correlation to the target variable and takes stepwise the most contributing variable. If the strongest contribution is too small, the process is stopped prematurely.
Usage

The CBFS_AGG aggregate has the following syntax:

```
CBFS_AGG(INT4 N, DOUBLE T, DOUBLE X1, DOUBLE X2, DOUBLE X3)
```

▲ Parameters

► **N**
  the number of best predictors to find out
  Type: INT4
  Max: number of variables Xi

► **T**
  the target variable
  Type: DOUBLE

► **X1, X2, etc.**
  the variables to choose from
  Type: DOUBLE

▲ Returns

NVARCHAR(10000) the string with the correlation matrix and the selected best predictors

Details

This function determines the N variables most strongly partially correlated with the target variable T, under
the normality assumption.

The major limitation is that the number of input variables cannot exceed 62.

This function returns a string containing the correlation matrix between variables T, X1, X2, X3, etc. The N best
predictors are then indicated with their correlation to the target variable T.

Examples

```
SELECT nza..CBFS_AGG(2, SEPALLENGTH, SEPALWIDTH, PETALLENGTH, PETALWIDTH) from nza..iris;
```

```
CBFS_AGG

-------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------

  correlation matrix

   0          1   -0.109369       0.871754

  0.817954
```
IBM Netezza In-Database Analytics Reference Guide

1      -0.109369               1
-0.420516       -0.356544

2       0.871754       -0.420516
1        0.962757

3       0.817954       -0.356544
0.962757               1

Feature selection 2
1) 2        0.871754    2) 1        0.578643
(1 row)

Related Functions
► category Analytics - Statistics
► CORR_AGG

CERROR - Classification Error

This stored procedure calculates the classification error, i.e. the ratio of wrongly classified predictions.

Usage
The CERROR stored procedure has the following syntax:

► CERROR(NVARCHAR(ANY) paramString)

Parameters
► paramString
   comma-separated list of <parameter>=<value> entries with parameters below
   Type: NVARCHAR(ANY)

► pred_table
   the input table or view containing predicted values
   Type: NVARCHAR(256)

► pred_column
   the input table column in <pred_table> containing predicted values
   Type: NVARCHAR(128)

► pred_id
   the input table column in <pred_table> identifying a unique instance id
   Type: NVARCHAR(128)
**true_table**
the input table or view containing real values
Type: NVARCHAR(256)

**true_column**
the input table column in <true_table> containing real values
Type: NVARCHAR(128)

**true_id**
the input table column in <true_table> identifying a unique instance id
Type: NVARCHAR(128)

**check**
flag indicating to check parameters or not. This may consume some time but prevents usage errors. Allowed values are: 'all' (check all), 'none' (check nothing), 'nulls' (check for nulls only).
Type: NVARCHAR(ANY)
Default: all

▲ Returns
DOUBLE the Classification error

**Details**
This stored procedure calculates the ratio of wrongly classified predictions among the total number of predictions. This is done by comparing the predictions made when applying a Classification model onto data, and the real values for this data.

**Examples**

```sql
CALL nza..GROW_DECTREE('model=adult_tree, intable=nza..adult_train, id=id, target=income, minsplit=1000');
CALL nza..PREDICT_DECTREE('model=adult_tree, intable=nza..adult_test, id=id, outtable=adult_pred');
CALL nza..CERROR('pred_table=adult_pred, true_table=nza..adult_test, pred_column=class, true_column=income, pred_id=id, true_id=id');
CALL nza..DROP_MODEL('model=adult_tree');
CALL nza..DROP_TABLE('adult_pred');
```

**GROW_DECTREE**
--------------

(1 row)

**PREDICT_DECTREE**
-------------------

(0 rows)
Related Functions

- category Analytics - Diagnostic Measures
- MAE
- MSE
- RAE
- RSE

CHISQ_TEST - Pearson's Chi-square test of independence

This stored procedure calculates the Chi-square value between two input columns and return the probability of their independence

**Usage**

The CHISQ_TEST stored procedure has the following syntax:

```
CHISQ_TEST(NVARCHAR(ANY) paramString)
```
Parameters

- **paramString**
  comma-separated list of <parameter>=<value> entries with parameters below
  Type: NVARCHAR(ANY)

- **intable**
  the input table
  Type: NVARCHAR(256)

- **outtable**
  the output table to write the Chi-square values into
  Type: NVARCHAR(256)

- **incolumn**
  the two input table columns, separated by a semicolon (;)
  Type: NVARCHAR(ANY)

- **by**
  the input table column which splits the data into groups for which the operation is to be performed
  Type: NVARCHAR(128)
  Default: <none>

Returns

DOUBLE the probability of the independence of the two input table columns, or the number of groups in the output table for which a Chi-square value has been calculated

Details

This stored procedure calculates the Chi-square value between two input columns, either in the whole input table or within the groups defined in the column specified by parameter <by>. Chi-square is a measure saying how easily it is to predict one column from the other. The stored procedure then returns a probability that the two columns are independent.

Rows with nulls in (any of) the columns of interest are omitted.

If parameter 'by' is specified, an output table must be specified. The output table is created with following columns: over_<by>, chi2statistic, df, percentage. The Chi-square value between the two input columns is given for each group of the <by> column. If the parameter by is not specified, the column is named over_no_group and contains a '-' value.

Additionally to the Chi-square statistics, this stored procedure returns the degree of freedom of the input variables and a percentage between 0 and 1.

- If the percentage ranges from 0 to 0.05, the columns are definitely independent,
- If the percentage ranges between 0.05 and 0.95, they are to be treated as independent, though one is not sure,
- If the percentage ranges between 0.95 and 1, then a predictive mutual dependency is definitely there.

Examples

```sql
CALL nza..CHISQ_TEST('intable=nza..censusincome, 
```
incolumn=citizenship;sex, outtable=Result7');
SELECT * FROM result7;
Drop TABLE result7;

chisq_test

-------------

0.999999

(1 row)

percentage | chi2statistic | df | over_no_group
------------------+-----------------+----+---------------
0.99999999847558 | 47.000679914049 |  4 | -

(1 row)

CALL nza..CHISQ_TEST('intable=nza..censusincome, incolumn=race;sex, outtable=Result7, by=class_of_worker');
SELECT Over_CLASS_OF_WORKER, percentage, chi2statistic, df FROM result7 ORDER BY Over_CLASS_OF_WORKER;

CALL nza..DROP_TABLE('result7');

CHISQ_TEST

-------------

9

(1 row)

| Over_CLASS_OF_WORKER      |    PERCENTAGE    | CHI2STATISTIC | DF
|---------------------------|------------------|---------------|
| Federal government        | 0.99999948679363 | 52.179603606458 |  4
| Local government          | 0.82417595627736 |  6.329968182374 |  4
| Never worked              | 0.86042942906178 |  6.9311728303627 |  4
| Not in universe           | 0.94196124426241 |  9.1254701438947 |  4
| Private                   | 0.99999886640305 |  70.000679914049 |  4

Related Functions

- category Analytics - Statistics
- CHISQ_TEST_AGG
- CHISQ_TEST_S_AGG
- PCHISQ

**CHISQ_TEST_AGG - Pearson's Chi-square**

This function calculates the Chi-square value between two integer input variables.

**Usage**

The `CHISQ_TEST_AGG` aggregate has the following syntax:

```
CHISQ_TEST_AGG(INT4 X, INT4 Y)
```

- **Parameters**
  - **X**
    - the first input variable
    - Type: INT4
    - Min: 0
  - **Y**
    - the second input variable
    - Type: INT4
    - Min: 0

- **Returns**
DOUBLE the Chi-square value, or NULL if there is too few data

**Details**

This function calculates the Chi-square value of two integer input variables. Chi-square is a measure saying how easily it is to predict one column from the other. It takes positive values, the lower the Chi-square value the more independent are the input variables.

Input values smaller than 0 are ignored. Use only for small data sets. For larger ones, the stored procedure CHISQ_TEST is recommended.

**Examples**

```
SELECT nza..CHISQ_TEST_AGG(
 case when sex='Female' then 0  when sex='Male' then 1
 else -1 end,
 case
 when race='Amer Indian Aleut or Eskimo' then 1
 when race='Black' then 2
 when race='White' then 3
 when race='Asian or Pacific Islander' then 4
 when race='Other' then 5
 else -1 end
) FROM nza..censusincome;
```

CHISQ_TEST_AGG
----------------
101.155617992

(1 row)

**Related Functions**

- CHISQ_TEST_AGG
- CHISQ_TEST_S_AGG
- PCHISQ

**CHISQ_TEST_S_AGG** - Pearson's Chi-square with degree of freedom

This function calculates the Chi-square value and the degree of freedom between two integer input...
variables

Usage
The CHISQ_TEST_S_AGG aggregate has the following syntax:

- **CHISQ_TEST_S_AGG**(INT4 X, INT4 Y)
  - **Parameters**
    - **X**
      - the first input variable
      - Type: INT4
      - Min: 0
    - **Y**
      - the second input variable
      - Type: INT4
      - Min: 0
  - **Returns**
    - VARCHAR(300) a string containing the Chi-square value and the degree of freedom, or NULL if there is too few data

Details
This function calculates the Chi-square value and the degree of freedom of two integer input variables. Chi-square is a measure saying how easily it is to predict one column from the other. It takes positive values, the lower the Chi-square value the more independent are the input variables.

Input values smaller than 0 are ignored. Use only for small data sets. For larger ones, the stored procedure CHISQ_TEST is recommended.

Examples

```
SELECT nza..CHISQ_TEST_S_AGG(
    case when sex='Female' then 0 when sex='Male' then 1 else -1 end,
    case
      when race='Amer Indian Aleut or Eskimo' then 1
      when race='Black' then 2
      when race='White' then 3
      when race='Asian or Pacific Islander' then 4
      when race='Other' then 5
    else -1 end
  ) FROM nza..censusincome;
```

CHISQ_TEST_S_AGG
chi2Stat = 101.156, df = 4
(1 row)

Related Functions
► category Analytics - Statistics
► CHISQ_TEST
► CHISQ_TEST_AGG
► PCHISQ
► PCHISQ_S

CLEANUP - Remove the model management infrastructure from the current database

This stored procedure removes the model management infrastructure from the current database, provided that there are no registered models.

Usage
The CLEANUP stored procedure has the following syntax:

>
CLEANUP()

▲ Returns
NVARCHAR(ANY) a text explaining that the cleanup was successful.

Details
This stored procedure removes the model management infrastructure from the current database, provided that there are no registered models.
- If the model management tables are empty, all objects (tables, views, sequences, procedures) are removed.
- If at least one model management table is not empty, an exception is raised, and nothing is done.

Examples
CALL nza..CLEANUP();
CALL nza..INITIALIZE();
The metadata objects have been removed successfully.
(1 row)

INITIALIZE

The metadata objects are successfully initialized.
(1 row)

Related Functions
► category Analytics - Model Management
► DROP_ALL_MODELS
► INITIALIZE

CMATRIX_ACC - Classification accuracy from a Confusion Matrix

This stored procedure calculates the Classification accuracy, i.e. the ratio of correctly classified predictions, from a confusion matrix.

Usage
The CMATRIX_ACC stored procedure has the following syntax:

► CMATRIX_ACC(NVARCHAR(ANY) paramString)
  ▲ Parameters
  ► paramString
    comma-separated list of <parameter>=<value> entries with parameters below
    Type: NVARCHAR(ANY)
  ► matrixTable
    the confusion matrix table
    Type: NVARCHAR(256)
  ▲ Returns
    DOUBLE the Classification accuracy

Details
This stored procedure calculates the ratio of correctly classified predictions among the total number of predictions from a confusion matrix. The confusion matrix table has the following columns: real, prediction, cnt. The column cnt contains the frequency of making a given prediction for the given real value.
Examples

CALL nza..TRAIN_TEST('modelType=dectree, model=iris_c45, traintable=nza..iris_train, testtable=nza..iris_test, target=class, id=id, outtable=iris_pred, minsplit=2');

CALL nza..CONFUSION_MATRIX('intable=nza..iris_test, resulttable=iris_pred, id=id, resultid=id, target=class, resulttarget=class, matrixTable=confMatrix');

CALL nza..CMATRIX_ACC('matrixTable=confMatrix');</p>

CALL nza..DROP_MODEL('model=iris_c45');</p>

CALL nza..DROP_TABLE('iris_pred');</p>

CALL nza..DROP_TABLE('confMatrix');</p>

TRAIN_TEST
-------------
0.918919
(1 row)

CONFUSION_MATRIX
------------------
5
(1 row)

CMATRIX_ACC
-----------
0.918919
(1 row)

DROP_MODEL
-----------
t
(1 row)

DROP_TABLE
-----------
DROP_TABLE

Related Functions
► category Analytics - Diagnostic Measures
► CONFUSION_MATRIX
► CMATRIX_WACC
► ACC

CMATRIX_STATS - Print Classification quality factors from a Confusion Matrix
This stored procedure calculates and displays different Classification quality factors from a confusion matrix.

Usage
The CMATRIX_STATS stored procedure has the following syntax:

► CMATRIX_STATS(NVARCHAR(ANY) paramString)
 ▲ Parameters
 ► paramString
  comma-separated list of <parameter>=<value> entries with parameters below
  Type: NVARCHAR(ANY)
 ► matrixTable
  the confusion matrix table
  Type: NVARCHAR(256)
 ▲ Returns
 TEXT pretty print of the quality factors of the confusion matrix

Details
This stored procedure calculates and displays different Classification quality factors from a confusion matrix. The confusion matrix table has the following columns: real, prediction, cnt. The column cnt contains the frequency of making a given prediction for the given real value.

Following quality factors are calculated:
- True Positive Rate for each predicted class,
- False Positive Rate for each predicted class,
- Positive Predictive Value for each predicted class,
- F-measure for each predicted class,
- correctly and incorrectly classified predictions,
- prediction accuracy and weighted prediction accuracy.

Examples

CALL nza..TRAIN_TEST('modelType=dectree, model=iris_c45, traintable=nza..iris_train, testtable=nza..iris_test, target=class, id=id, outtable=iris_pred, minsplit=2');

CALL nza..CONFUSION_MATRIX('intable=nza..iris_test, resulttable=iris_pred, id=id, resultid=id, target=class, resulttarget=class, matrixTable=confMatrix');

CALL nza..CMATRIX_STATS('matrixTable=confMatrix');

CALL nza..DROP_MODEL('model=iris_c45');

CALL nza..DROP_TABLE('iris_pred');

CALL nza..DROP_TABLE('confMatrix');
class -> setosa
  True Positive Rate (sensitivity/recall): 0.846154
  False Positive Rate: 0
  Positive Predictive Value (precision): 1
  F-Measure: 0.91666675694444

class -> versicolor
  True Positive Rate (sensitivity/recall): 0.933333
  False Positive Rate: 0.090909
  Positive Predictive Value (precision): 0.875
  F-Measure: 0.90322565036418

class -> virginica
  True Positive Rate (sensitivity/recall): 1
  False Positive Rate: 0.035714
  Positive Predictive Value (precision): 0.9
  F-Measure: 0.94736842105263

Correctly Classified Instances: 34
Incorrectly Classified Instances: 3
Accuracy: 91.8919 %
Weighted Accuracy: 92.649566666667 %
(1 row)

DROP_MODEL
----------
t
(1 row)

DROP_TABLE
----------
t
(1 row)
Related Functions

- category Analytics - Diagnostic Measures
- CONFUSION_MATRIX
- TPR
- FPR
- PPV
- FMEASURE
- CMATRIX_ACC
- CMATRIX_WACC

**CMATRIX_WACC - Weighted Classification accuracy from a Confusion Matrix**

This stored procedure calculates the weighted Classification accuracy, i.e. the weighted ratio of correctly classified predictions, from a confusion matrix.

**Usage**

The CMATRIX_WACC stored procedure has the following syntax:

```
CMATRIX_WACC(NVARCHAR(ANY) paramString)
```

▲ Parameters

- **paramString**
  comma-separated list of `<parameter>=<value>` entries with parameters below
  Type: NVARCHAR(ANY)

- **matrixTable**
  the confusion matrix table
  Type: NVARCHAR(256)

▲ Returns

DOUBLE the weighted Classification accuracy
Details

This stored procedure calculates the weighted ratio of correctly classified predictions among the total number of predictions from a confusion matrix. The confusion matrix table has the following columns: real, prediction, cnt. The column cnt contains the frequency of making a given prediction for the given real value.

Each class is assigned the same weight in calculating the ratio, independently of the real frequency of the class in the input data. This is useful when classes are not equally frequent.

Examples

CALL nza..TRAIN_TEST('modelType=dectree, model=iris_c45, traintable=nza..iris_train, testtable=nza..iris_test, target=class, id=id, outtable=iris_pred, minsplit=2');

CALL nza..CONFUSION_MATRIX('intable=nza..iris_test, resulttable=iris_pred, id=id, resultid=id, target=class, resulttarget=class, matrixTable=confMatrix');

CALL nza..CMATRIX_WACC('matrixTable=confMatrix');

CALL nza..DROP_MODEL('model=iris_c45');

CALL nza..DROP_TABLE('iris_pred');

CALL nza..DROP_TABLE('confMatrix');

TRAIN_TEST

-------------

0.918919

(1 row)

CONFUSION_MATRIX

---------------------

5

(1 row)

CMATRIX_WACC

---------------------

0.92649566666667

(1 row)

DROP_MODEL

---------------------

t
(1 row)

DROP_TABLE
------------
t
(1 row)

DROP_TABLE
------------
t
(1 row)

Related Functions
► category Analytics - Diagnostic Measures
► CONFUSION_MATRIX
► CMATRIX_ACC
► WACC

COL2TRCV_MANOVA_ONE_WAY_TEST - data transformation from column representation to trcv for Multivariate Analysis of Variance in one way setting

This stored procedure transforms data between column format and trcv format

Usage
The COL2TRCV_MANOVA_ONE_WAY_TEST stored procedure has the following syntax:

► COL2TRCV_MANOVA_ONE_WAY_TEST(NVARCHAR(ANY) paramString)
  ▲ Parameters
    ► paramString
      comma-separated list of <parameter>=<value> entries with parameters below
      Type: NVARCHAR(ANY)
    ► intable
      the input table name
Type: NVARCHAR(ANY)

- **outtable**
  the output table name for trcv data
  Type: NVARCHAR(ANY)

- **taskname_outtable**
  the output table name for task names and ids, not required
  Type: NVARCHAR(ANY)

- **varname_outtable**
  the output table name for variable names and ids, not required
  Type: NVARCHAR(ANY)

- **factor1**
  the input table column identifying a first factor (so-called treatment in RBD/CRD nomenclature)
  Type: NVARCHAR(ANY)

- **incolumn**
  the input table observation columns (dependent variables), separated by a semi-colon (;).
  Type: NVARCHAR(ANY)

- **id**
  the input table column which uniquely identifies records
  Type: NVARCHAR(ANY)

- **by**
  the input table column which splits the input table into subtables, on each of them a separate MANOVA is run. If not specified, the whole input table is subject of a single MANOVA run.
  Type: NVARCHAR(ANY)
  Default: <none>

▲ Returns
   NVARCHAR(2000) A string confirming execution of the process or indicating a failure. The real output is contained in the output table

Details
This stored procedure performs one-way analysis of variance/covariance aiming to tell whether or not the groups of data identified by factor1 have the same mean value in all dependent variables or not.

Examples

```
CREATE TABLE wheattest2(fieldId INTEGER, variety CHAR(1), yield DOUBLE, barrels INTEGER);

INSERT INTO wheattest2 VALUES(1,'A',80, 4);
INSERT INTO wheattest2 VALUES(2,'A',65, 3);
INSERT INTO wheattest2 VALUES(3,'A',50, 2);
INSERT INTO wheattest2 VALUES(4,'B',100, 5);
```
INSERT INTO wheattest2 VALUES(5,'B',85, 4);
INSERT INTO wheattest2 VALUES(6,'B',70, 3);
INSERT INTO wheattest2 VALUES(7,'C',60, 2);
INSERT INTO wheattest2 VALUES(8,'C',75, 3);
INSERT INTO wheattest2 VALUES(9,'C',90, 5);

CALL nza..COL2TRCV_MANOVA_ONE_WAY_TEST('intable=wheattest2,id=fieldId, incolumn= yield; barrels, factor1=variety, outtable=outtab_trcv');

select * from outtab_trcv order by id_task, row,col;

CALL nza..MANOVA_ONE_WAY_TEST('intable=outtab_trcv, outtable=outtab');

drop table outtab;
drop table outtab_trcv;
drop table wheattest2;

<table>
<thead>
<tr>
<th>ID_TASK</th>
<th>ROW</th>
<th>COL</th>
<th>VAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>80</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>65</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>2</td>
<td>50</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>2</td>
<td>100</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>2</td>
<td>85</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
CREATE TABLE wheatbytest(fieldId INTEGER, variety CHAR(1), yield DOUBLE, barrels INTEGER, experiment INT4);

INSERT INTO wheatbytest VALUES(1,'A',65,3,1);
-- suspect: INSERT INTO  wheatbytest VALUES(2,'A',66,4,1);
INSERT INTO wheatbytest VALUES(2,'A',65,3,1);
INSERT INTO  wheatbytest VALUES(3,'A',64,2,1);
INSERT INTO  wheatbytest VALUES(4,'B',84,3,1);
INSERT INTO  wheatbytest VALUES(5,'B',85,4,1);
INSERT INTO  wheatbytest VALUES(6,'B',86,5,1);
INSERT INTO  wheatbytest VALUES(7,'C',75,3,1);
INSERT INTO  wheatbytest VALUES(8,'C',76,4,1);
INSERT INTO  wheatbytest VALUES(9,'C',74,2,1);
INSERT INTO  wheatbytest VALUES(11,'A',80,4,2);
INSERT INTO  wheatbytest VALUES(12,'A',65,3,2);
INSERT INTO  wheatbytest VALUES(13,'A',50,2,2);
INSERT INTO  wheatbytest VALUES(14,'B',100,5,2);
INSERT INTO  wheatbytest VALUES(15,'B',85,4,2);
INSERT INTO  wheatbytest VALUES(16,'B',70,3,2);
INSERT INTO  wheatbytest VALUES(17,'C',60,2,2);
INSERT INTO  wheatbytest VALUES(18,'C',75,3,2);
INSERT INTO wheatbytest VALUES(19,'C',90,5,2);

CALL nza..COL2TRCV_MANOVA_ONE_WAY_TEST('intable=wheatbytest, id=fieldid, incolumn=yield;barrels, factor1=variety, by=experiment, outtable=outbytab_trcv');

SELECT * FROM outbytab_trcv order by id_task,row,col;

CALL nza..MANOVA_ONE_WAY_TEST('intable=outbytab_trcv, outtable=outbytab');

CALL nza..DROP_TABLE('outbytab_trcv');

CALL nza..DROP_TABLE('outbytab');

CALL nza..DROP_TABLE('wheatbytest');

<table>
<thead>
<tr>
<th>ID_TASK</th>
<th>ROW</th>
<th>COL</th>
<th>VAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>65</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>65</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>2</td>
<td>64</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>2</td>
<td>84</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>2</td>
<td>85</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>2</td>
<td>86</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>2</td>
<td>75</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>2</td>
<td>76</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>9</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>9</td>
<td>2</td>
<td>74</td>
</tr>
<tr>
<td>1</td>
<td>9</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>80</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>65</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2</td>
<td>50</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>2</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>2</td>
<td>85</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>2</td>
<td>70</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>2</td>
<td>60</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>2</td>
<td>75</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>2</td>
<td>90</td>
</tr>
</tbody>
</table>
CREATE TABLE wheatbytest(fieldId INTEGER, variety CHAR(1), yield DOUBLE, barrels INTEGER, experiment varchar(100));

INSERT INTO wheatbytest VALUES(1,'A',65,3,'one');
-- suspect: INSERT INTO wheatbytest
VALUES(2,'A',66,4,'one');

INSERT INTO wheatbytest VALUES(2,'A',65,3,'one');

INSERT INTO wheatbytest VALUES(3,'A',64,2,'one');

INSERT INTO wheatbytest VALUES(4,'B',84,3,'one');

INSERT INTO wheatbytest VALUES(5,'B',85,4,'one');

INSERT INTO wheatbytest VALUES(6,'B',86,5,'one');

INSERT INTO wheatbytest VALUES(7,'C',75,3,'one');

INSERT INTO wheatbytest VALUES(8,'C',76,4,'one');

INSERT INTO wheatbytest VALUES(9,'C',74,2,'one');

INSERT INTO wheatbytest VALUES(11,'A',80,4,'two');

INSERT INTO wheatbytest VALUES(12,'A',65,3,'two');

INSERT INTO wheatbytest VALUES(13,'A',50,2,'two');

INSERT INTO wheatbytest VALUES(14,'B',100,5,'two');

INSERT INTO wheatbytest VALUES(15,'B',85,4,'two');

INSERT INTO wheatbytest VALUES(16,'B',70,3,'two');

INSERT INTO wheatbytest VALUES(17,'C',60,2,'two');

INSERT INTO wheatbytest VALUES(18,'C',75,3,'two');

INSERT INTO wheatbytest VALUES(19,'C',90,5,'two');

CALL nza..COL2TRCV_MANOVA_ONE_WAY_TEST('intable=wheatbytest,
id=fieldid, incolumn=yield;barrels, factor1=variety,
by=experiment, outtable=outbytab_trcv,
taskname_outtable=tasknames');

SELECT * FROM tasknames  order by id_task;

CALL nza..DROP_TABLE('outbytab_trcv');

CALL nza..DROP_TABLE('tasknames');
CALL nza..DROP_TABLE('wheatbytest');

------------------------------------
call now:

select x.id_task as id_task,y.id_matrix,y.row,y.col,y.val
from
    (select *,nzm..reorder_matrix(id_task) over
        (partition by id_task order by row desc, col desc) as id_task_ro
        from "OUTBYTAB_TRCV") x
    ,TABLE(nza..ctf_mat_manova(x.id_task_ro,x.row,x.col,x.val)) y;

(1 row)

<table>
<thead>
<tr>
<th>ID_TASK</th>
<th>TASK_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>one</td>
</tr>
<tr>
<td>2</td>
<td>two</td>
</tr>
</tbody>
</table>

(2 rows)

DROP_TABLE
-----------
t
(1 row)

DROP_TABLE
-----------
t
(1 row)

DROP_TABLE
-----------
t
(1 row)
Related Functions

- category Analytics - Statistics
- MANOVA_Two_WAY_TEST
- ANOVA_CRD_TEST

COL2TRCV_MANOVA_TWO_WAY_TEST - data transformation from column representation to grcv for Multivariate Analysis of Variance in two way setting

This stored procedure transforms data between column format and trcv format

Usage

The COL2TRCV_MANOVA_TWO_WAY_TEST stored procedure has the following syntax:

```
COL2TRCV_MANOVA_TWO_WAY_TEST(NVARCHAR(ANY) paramString)
```

- **paramString**
  - comma-separated list of <parameter>=<value> entries with parameters below
  - Type: NVARCHAR(ANY)

- **itable**
  - the input table name
  - Type: NVARCHAR(ANY)

- **outtable**
  - the output table name for trcv data
  - Type: NVARCHAR(ANY)

- **taskname_outtable**
  - the output table name for task names and ids, not required
  - Type: NVARCHAR(ANY)

- **varname_outtable**
  - the output table name for variable names and ids, not required
  - Type: NVARCHAR(ANY)

- **factor1**
  - the input table column identifying a first factor (so-called treatment in RBD/CRD nomenclature)
  - Type: NVARCHAR(ANY)

- **factor2**
  - the input table column identifying a second factor (so-called block in RBD nomenclature)
Type: NVARCHAR(ANY)

► incolumn
the input table observation columns (dependent variables), separated by a semi-colon (;).
Type: NVARCHAR(ANY)

► id
the input table column which uniquely identifies records
Type: NVARCHAR(ANY)

► by
the input table column which splits the input table into subtables, on each of them a separate MANOVA is run. If not specified, the whole input table is subject of a single MANOVA run.
Type: NVARCHAR(ANY)
Default: <none>

▲ Returns
NVARCHAR(2000) A string confirming execution of the process or indicating a failure. The real output is contained in the output table

Details
This stored procedure performs one-way analysis of variance/covariance aiming to tell whether or not the groups of data identified by factor1 have the same mean value in all dependent variables or not.

Examples

CREATE TABLE wheattest2(fieldId INTEGER, variety CHAR(1), yield DOUBLE, barrels INTEGER);

INSERT INTO wheattest2 VALUES(1,'A',80, 4);
INSERT INTO wheattest2 VALUES(2,'A',65, 3);
INSERT INTO wheattest2 VALUES(3,'A',50, 2);
INSERT INTO wheattest2 VALUES(4,'B',100, 5);
INSERT INTO wheattest2 VALUES(5,'B',85, 4);
INSERT INTO wheattest2 VALUES(6,'B',70, 3);
INSERT INTO wheattest2 VALUES(7,'C',60, 2);
INSERT INTO wheattest2 VALUES(8,'C',75, 3);
INSERT INTO wheattest2 VALUES(9,'C',90, 5);

CALL nza..COL2TRCV_MANOVA_TWO_WAY_TEST('intable=wheattest2,id=fieldId, incolumn=yield, factor1=variety, factor2= barrels, outtable=outtab_trcv');

select * from outtab_trcv order by id_task, row, col;

CALL nza..MANOVA_TWO_WAY_TEST('intable=outtab_trcv, outtable=outtab');
```
drop table outtab;
drop table outtab_trcv;
drop table wheattest2;

| ID_TASK | ROW | COL | VAL |
|---------+-----+-----+-----|
| 0       | 1   | 1   | 1   |
| 0       | 1   | 2   | 3   |
| 0       | 1   | 3   | 80  |
| 0       | 2   | 1   | 1   |
| 0       | 2   | 2   | 2   |
| 0       | 2   | 3   | 65  |
| 0       | 3   | 1   | 1   |
| 0       | 3   | 2   | 1   |
| 0       | 3   | 3   | 50  |
| 0       | 4   | 1   | 2   |
| 0       | 4   | 2   | 4   |
| 0       | 4   | 3   | 100 |
| 0       | 5   | 1   | 2   |
| 0       | 5   | 2   | 3   |
| 0       | 5   | 3   | 85  |
| 0       | 6   | 1   | 2   |
| 0       | 6   | 2   | 2   |
| 0       | 6   | 3   | 70  |
| 0       | 7   | 1   | 3   |
| 0       | 7   | 2   | 1   |
| 0       | 7   | 3   | 60  |
| 0       | 8   | 1   | 3   |
| 0       | 8   | 2   | 2   |
| 0       | 8   | 3   | 75  |
| 0       | 9   | 1   | 3   |
| 0       | 9   | 2   | 4   |
```
CREATE TABLE wheatbytest(fieldId INTEGER, variety CHAR(1), yield DOUBLE, barrels INTEGER, experiment INT4);

INSERT INTO wheatbytest VALUES(1,'A',65,3,1);
-- suspect: INSERT INTO  wheatbytest VALUES(2,'A',66,4,1);
INSERT INTO wheatbytest VALUES(2,'A',65,3,1);
INSERT INTO wheatbytest VALUES(3,'A',64,2,1);
INSERT INTO wheatbytest VALUES(4,'B',84,3,1);
INSERT INTO wheatbytest VALUES(5,'B',85,4,1);
INSERT INTO wheatbytest VALUES(6,'B',86,5,1);
INSERT INTO wheatbytest VALUES(7,'C',75,3,1);
INSERT INTO wheatbytest VALUES(8,'C',76,4,1);
INSERT INTO wheatbytest VALUES(9,'C',74,2,1);
INSERT INTO wheatbytest VALUES(11,'A',80,4,2);
INSERT INTO wheatbytest VALUES(12,'A',65,3,2);
INSERT INTO wheatbytest VALUES(13,'A',50,2,2);
INSERT INTO wheatbytest VALUES(14,'B',100,5,2);
INSERT INTO wheatbytest VALUES(15,'B',85,4,2);
INSERT INTO wheatbytest VALUES(16,'B',70,3,2);
INSERT INTO wheatbytest VALUES(17,'C',60,2,2);
INSERT INTO wheatbytest VALUES(18,'C',75,3,2);
INSERT INTO wheatbytest VALUES(19,'C',90,5,2);

CALL nza..COL2TRCV_MANOVA_TWO_WAY_TEST('intable=wheatbytest, id=fieldid, incolumn=yield, factor2=barrels, factor1=variety, by=experiment, outtable=outbytab_trcv');

SELECT * FROM outbytab_trcv  order by id_task,row,col;

CALL nza..MANOVA_TWO_WAY_TEST('intable=outbytab_trcv, outtable=outbytab');

CALL nza..DROP_TABLE('outbytab_trcv');

CALL nza..DROP_TABLE('outbytab');

CALL nza..DROP_TABLE('wheatbytest');

ID_TASK | ROW | COL | VAL
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
<td>65</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>65</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>3</td>
<td>64</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>3</td>
<td>84</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>3</td>
<td>85</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>3</td>
<td>86</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>3</td>
<td>75</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>3</td>
<td>76</td>
</tr>
<tr>
<td>1</td>
<td>9</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>9</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>9</td>
<td>3</td>
<td>74</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
<td>80</td>
</tr>
</tbody>
</table>
CREATE TABLE wheatbytest(fieldId INTEGER, variety CHAR(1), yield DOUBLE, barrels INTEGER, experiment varchar(100));

INSERT INTO wheatbytest VALUES(1,'A',65,3,'one');

-- suspect: INSERT INTO wheatbytest VALUES(2,'A',66,4,'one');
INSERT INTO wheatbytest VALUES(2,'A',65,3,'one');
INSERT INTO wheatbytest VALUES(3,'A',64,2,'one');
INSERT INTO wheatbytest VALUES(4,'B',84,3,'one');
INSERT INTO wheatbytest VALUES(5,'B',85,4,'one');
INSERT INTO wheatbytest VALUES(6,'B',86,5,'one');
INSERT INTO wheatbytest VALUES(7,'C',75,3,'one');
INSERT INTO wheatbytest VALUES(8,'C',76,4,'one');
INSERT INTO wheatbytest VALUES(9,'C',74,2,'one');
INSERT INTO wheatbytest VALUES(11,'A',80,4,'two');
INSERT INTO wheatbytest VALUES(12,'A',65,3,'two');
INSERT INTO wheatbytest VALUES(13,'A',50,2,'two');
INSERT INTO wheatbytest VALUES(14,'B',100,5,'two');
INSERT INTO wheatbytest VALUES(15,'B',85,4,'two');
INSERT INTO wheatbytest VALUES(16,'B',70,3,'two');
INSERT INTO wheatbytest VALUES(17,'C',60,2,'two');
INSERT INTO wheatbytest VALUES(18,'C',75,3,'two');
INSERT INTO wheatbytest VALUES(19,'C',90,5,'two');

CALL nza..COL2TRCV_MANOVA_TWO_WAY_TEST('intable=wheatbytest, id=fieldid, incolumn=yield, factor2=barrels, factor1=variety, by=experiment, outtable=outbytab_trcv, taskname_outtable=tasknames');

SELECT * FROM tasknames order by id_task;
CALL nza..DROP_TABLE('outbytab_trcv');
CALL nza..DROP_TABLE('tasknames');
CALL nza..DROP_TABLE('wheatbytest');

CALL nza..COL2TRCV_MANOVA_TWO_WAY_TEST

--
call now:

select x.id_task as id_task,y.id_matrix,y.row,y.col,y.val from (select * ,nzm..reorder_matrix(id_task)
over (partition by id_task order by row desc, col desc) as
id_task_ro
from "OUTBYTAB_TRCV") x
, TABLE(nza..ctf_mat_manova(x.id_task_ro,x.row,x .col,x.val)) y;
(1 row)
MIECIO(ADMIN) => SELECT * FROM tasknames order by id_task;

<table>
<thead>
<tr>
<th>ID_TASK</th>
<th>TASK_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>one</td>
</tr>
<tr>
<td>2</td>
<td>two</td>
</tr>
</tbody>
</table>

(2 rows)
DROP_TABLE
---------
t
(1 row)
DROP_TABLE
---------
t
(1 row)
DROP_TABLE
---------
t
(1 row)

create view irisX05 as
select *, mod(id,3) as factor from nza..iris;

CALL nza..COL2TRCV_MANOVA_TWO_WAY_TEST('intable=irisX05,id=Id, incolumn= sepalwidth;sepallength;petallength;petalwidth, factor1=factor, factor2=class, outtable=outtab05a');

CALL nza..DROP_TABLE('outtab05a');

CALL nza..DROP_TABLE('irisX05');

CREATE VIEW
Related Functions

- category Analytics - Statistics
- MANOVA_Two_WAY_TEST
- Error: Reference source not found

COLUMN_PROPERTIES - Create a column properties table

This stored procedure creates a column properties table based on the given input table.

Usage

The COLUMN_PROPERTIES stored procedure has the following syntax:

- COLUMN_PROPERTIES((NVARCHAR(ANY) paramString))
  ▲ Parameters
  ▶ paramString
    comma-separated list of <parameter>=<value> entries with parameters below
    Type: NVARCHAR(ANY)
  ▶ intable
    the input table
    Type: NVARCHAR(ANY)
► **outtable**

the output table (column properties table) that receives the column properties.

Type: NVARCHAR(ANY)

► **incolumn**

the input table columns with special properties, separated by a semi-colon (;).

Each column is followed by one or several of the following properties:

- its type: ':nom' (for nominal), ':cont' (for continuous). Per default, all numerical types are continuous, other types are nominal.

- its role: ':id', ':target', ':input', ':ignore', ':objweight', ':trials'.

- its weight: ':colweight(<wgt>)' where <wgt> is a numeric value of the weight.

If the parameter is undefined, all columns of the input table have default properties.

Type: NVARCHAR(ANY)

Default: <none>

► **coldeftype**

default type of the input table columns. Allowed values are 'nom' and 'cont'.

If the parameter is undefined, all numeric columns are considered continuous, other columns nominal.

Type: NVARCHAR(ANY)

Default: <none>

► **coldefrole**

default role of the input table columns. Allowed values are 'input' and 'ignore'.

If the parameter is undefined, all columns are considered 'input' columns.

Type: NVARCHAR(ANY)

Default: input

▲ Returns

INTEGER the number of records written into the column properties table

**Details**

This stored procedure creates a column properties table based on all columns of the input table. A column properties table can be used as input in different stored procedures to build an analytics model. The output table containing the column properties has following columns: idcol, colname, coldatatype, coltype, colrole, colweight. For each input column specified, one row is written into the output table. The output column colweight contains 1 per default. This table can be modified manually or using SET_COLUMN_PROPERTIES before using it to build an analytics model.

**Examples**

```
CALL nza..COLUMN_PROPERTIES('intable=nza..weather, outtable=weatherDefSchema, coldeftype=nom, incolumn=instance:cont:id;play:target;WINDY:ignore');

SELECT * FROM weatherDefSchema ORDER BY idcol;
```
CALL nza..DROP_TABLE('weatherDefSchema');

COLUMN_PROPERTIES

-------------------
6

(IDCOL | COLNAME | COLDATATYPE | COLTYPE | COLROLE | COLWEIGHT
-------+-------------+--------------------------------
1     | INSTANCE    | INTEGER                        | cont    | id      |         1
2     | OUTLOOK     | NATIONAL CHARACTER VARYING(10) | nom     | input   |         1
3     | TEMPERATURE | NATIONAL CHARACTER VARYING(10) | nom     | input   |         1
4     | HUMIDITY    | NATIONAL CHARACTER VARYING(10) | nom     | input   |         1
5     | WINDY       | NATIONAL CHARACTER VARYING(10) | nom     | ignore  |         1
6     | PLAY        | NATIONAL CHARACTER VARYING(3)  | nom     | target  |         1

(DROP_TABLE

-------------------
t

(1 row)

Related Functions

► category Analytics - Column Properties
► COLUMN_PROPERTIES_CHECK
► SET_COLUMN_PROPERTIES
► GET_COLUMN_LIST
COLUMN_PROPERTIES_CHECK - Check a column properties table for correctness

This stored procedure checks if the given table is a correct column properties table for the given input table.

**Usage**

The COLUMN_PROPERTIES_CHECK stored procedure has the following syntax:

```sql
COLUMN_PROPERTIES_CHECK((NVARCHAR(ANY) paramString)

▲ Parameters

- **paramString**
  - comma-separated list of <parameter>=<value> entries with parameters below
  - Type: NVARCHAR(ANY)

- **inTable**
  - the data input table
  - Type: NVARCHAR(ANY)

- **colPropertiesTable**
  - the column properties table to check for correctness
  - Type: NVARCHAR(ANY)

▲ Returns

  BOOLEAN true if the column properties table is correct regarding the given data input table

**Details**

This stored procedure checks that the given column properties table is correct regarding the given data input table. It raises an exception when:

- the column name is incorrect,
- the column role or type does not exist,
- the column type can not be casted from the column data type,
- more than one column with role=id or role=objweights or role=trials are present... This stored procedure do not check if target is defined.

**Examples**

```sql
CALL nza..COLUMN_PROPERTIES('intable=nza..weather, outtable=weatherDefSchema, coldeftype=nom, incolumn=temperature;WINDY:nom:ignore');
CALL nza..COLUMN_PROPERTIES_CHECK('intable=nza..weather, colPropertiesTable=weatherDefSchema');
CALL nza..DROP_TABLE('weatherDefSchema');
```
Related Functions

- category Analytics - Column Properties
- COLUMN_PROPERTIES

COND_ENTROPY - Conditional Entropy

This stored procedure calculates the conditional entropy of a column Y given a column X.

Usage

The COND_ENTROPY stored procedure has the following syntax:

```
COND_ENTROPY(NVARCHAR(ANY) paramString)
```

- **paramString**
  - comma-separated list of <parameter>=<value> entries with parameters below
  - Type: NVARCHAR(ANY)

- **intable**
  - the input table
  - Type: NVARCHAR(256)

- **incolumn**
the two input table columns separated by a semicolon (;). The conditional column is followed by :X, the column for which conditional entropy has to be calculated is followed by :Y.

Type: NVARCHAR(ANY)

► by
the input table column which splits the data into groups for which the operation is to be performed. If specified, an output table must be indicated.

Type: NVARCHAR(128)
Default: <none>

► outtable
the output table to write the entropy into. This parameter must be specified if parameter by is specified.

Type: NVARCHAR(256)
Default: <none>

▲ Returns
DOUBLE the conditional entropy of the input table column Y, or the number of groups in the output table for which conditional entropy has been calculated

Details
This stored procedure calculates the entropy of an input column Y given another input column X, either in the whole input table or within the groups defined in the column specified by parameter <by>. Conditional entropy gives an impression of the predictability of Y given X, when compared with the entropy of X. It takes a positive value, the lower the better predictability.

If parameter ‘by’ is specified, an output table must be specified. The output table is created with following columns: over_<by>, condentropy. The conditional entropy of the input column Y is given for each group of the <by> column.

Examples

CALL nza..COND_ENTROPY('intable=nza..CensusIncome, incolumn=age:X;wage_per_hour:Y');

```
COND_ENTROPY
-----------------
 0.61428780276346
```

CALL nza..COND_ENTROPY('intable=nza..CensusIncome, incolumn=age:X; wage_per_hour:Y, outtable=resultTable, by=sex');

SELECT * FROM resultTable ORDER BY over_sex;
CALL nza..DROP_TABLE('resultTable');

COND_ENTROPY

--------------

2

(1 row)

CONDENTROPY    | OVER_SEX
------------------+----------
0.58848694657749 | Female
0.58477089622504 | Male

(2 rows)

DROP_TABLE

----------

t

(1 row)

Related Functions

- category Analytics - Statistics
- ENTROPY
- JOINT_ENTROPY
- MUTUALINFO

CONFUSION_MATRIX - Build a Confusion Matrix

This stored procedure builds a confusion matrix for Classification predictions

Usage

The CONFUSION_MATRIX stored procedure has the following syntax:

CONFUSION_MATRIX(NVARCHAR(ANY) paramString)

- Parameters
  - paramString
    - comma-separated list of <parameter>=<value> entries with parameters below
**Type:** NVARCHAR(ANY)

- **intable**
  the input table containing real values
  Type: NVARCHAR(256)

- **id**
  the input table column in `<intable>` identifying a unique instance id
  Type: NVARCHAR(128)

- **target**
  the input table column in `<intable>` containing real values
  Type: NVARCHAR(128)

- **resulttable**
  the input table containing predicted values
  Type: NVARCHAR(256)

- **resultid**
  the input table column in `<resulttable>` identifying a unique instance id
  Type: NVARCHAR(128)
  Default: id

- **resulttarget**
  the input table column in `<resulttable>` containing predicted values
  Type: NVARCHAR(128)
  Default: class

- **matrixTable**
  the output table where the confusion matrix will be stored
  Type: NVARCHAR(ANY)

▲ **Returns**
INTEGR the number of rows written in the confusion matrix output table

**Details**
This stored procedure builds a confusion matrix for Classification predictions. This is done by comparing the predictions made when applying a Classification model onto data, and the real values for this data.
The output table that contains the confusion matrix has following columns: real, prediction, cnt. The column cnt contains the frequency of making a given prediction for the given real value.

**Examples**

```sql
CALL nza..TRAIN_TEST('modelType=dectree, model=iris_c45,
  traintable=nza..iris_train, testtable=nza..iris_test,
  target=class, id=id, outtable=iris_pred, minsplit=2');

CALL nza..CONFUSION_MATRIX('intable=nza..iris_test,
  resulttable=iris_pred, id=id, resultid=id, target=class,
  resulttarget=class, matrixTable=confMatrix');
```
CALL nza..DROP_MODEL('model=iris_c45');
CALL nza..DROP_TABLE('iris_pred');
CALL nza..DROP_TABLE('confMatrix');

TRAIN_TEST
------------

0.918919
(1 row)

CONFUSION_MATRIX
------------------

5
(1 row)

DROP_MODEL
------------

t
(1 row)

DROP_TABLE
------------

t
(1 row)

DROP_TABLE
------------

t
(1 row)

Related Functions
► category Analytics - Diagnostic Measures
COPY_MODEL - Duplicate an analytics model

This stored procedure duplicates the given analytics model to a new model with the given new name. The source model can be in another database.

Usage

The COPY_MODEL stored procedure has the following syntax:

```sql
COPY_MODEL(NVARCHAR(ANY))
```

- **paramString**: comma-separated list of <parameter>=<value> entries with parameters below
  - **model**: the model to be duplicated. It can be qualified by a database name (<database>..<model>) if not in the current database.
  - **copy**: The name of the copy of the model to be created in the current database

Returns

BOOLEAN always true (otherwise an exception is raised)

Details

This stored procedure duplicates the given analytics model.

The model can be in any database, but the model copy is always created in the current database. Only models in state "Complete" can be copied.

- To duplicate an analytics model, the user needs the SELECT privilege for the model.
- The stored procedure copies the model properties and the managed components of the model; referenced tables are not copied.
- The current user is the owner of the new model. Creator and creation time are the same as in the source model.

Examples

```sql
CALL nza..ARULE('intable=nza..retail, model=mbamodel, supporttype=percent, support=5, lvl=0, maxsetsize=5, confidence=0.5');
CALL nza..COPY_MODEL('model=mbamodel, copy=mbaorig');
CALL nza..LIST_MODELS('where=upper(modelname) like ''MBA%''');
CALL nza..DROP_MODEL('model=mbamodel');
CALL nza..DROP_MODEL('model=mbaorig');
```
NOTICE:

RUNNING FP-Growth algorithm:
DATASET : "NZA".."RETAIL"
Transaction column : "TID"
Item column : "ITEM"
Group by : <none>
Minimum support : 5 %
Minimum confidence: 0.5
Max frequent itemset size : 5
Level of conditional dbs : 0
Result tables prefix : "NZA_META_MBAMODEL"

ARULE
------
 14
(1 row)

COPY_MODEL
----------
  t
(1 row)

NOTICE:

MODELNAME | OWNER | CREATED | STATE | MININGFUNCTION | ALGORITHM | USERCATEGORY
-----------+-------+---------+-------+----------------+-----------+--------------
+------------------+-----------+--------------
MBAMODEL | JOE | 2012-01-01 00:00:00 | Complete | associationRules | FP-Growth |
MBAORIG | JOE | 2012-01-01 00:00:00 | Complete | associationRules | FP-Growth |
LIST_MODELS
----------
Related Functions

- category Analytics - Model Management
- LIST_MODELS

CORR - Mutual Correlation

This stored procedure calculates the correlation between two numeric input columns.

Usage

The CORR stored procedure has the following syntax:

```
CORR(NVARCHAR(ANY) paramString)
```

Parameters

- **paramString**
  
  A comma-separated list of `<parameter>=<value>` entries with parameters below.

  Type: NVARCHAR(ANY)

- **intable**
  
  The input table.

  Type: NVARCHAR(256)

- **incolumn**
  
  The two numeric input table columns, separated by a semicolon (;). Optionally, a third numeric column can be specified for weights followed by :objweight.
Type: NVARCHAR(ANY)

- **by**
  the input table column which splits the data into groups for which the operation is to be performed. If specified, an output table must be indicated.
  
  Type: NVARCHAR(128)
  Default: <none>

- **outtable**
  the output table to write the correlations into. This parameter must be specified if parameter by is specified.
  
  Type: NVARCHAR(256)
  Default: <none>

▲ Returns
  DOUBBLE the correlation between the two input table columns, or the number of groups in the output table for which a correlation has been calculated

**Details**

This stored procedure calculates the correlation between two numeric input columns, either in the whole input table or within the groups defined in the column specified by parameter <by>. Correlation is a measure saying how easily it is to predict one column from the other. It takes a value between -1 (inversely correlated) and 1 (correlated), 0 means that the two columns are independent.

If parameter 'by' is specified, an output table must be specified. The output table is created with following columns: <by>, correlation. The correlation between the two input columns is given for each group of the <by> column.

**Examples**

```sql
CALL nza..CORR('intable=nza..CensusIncome, incolumn=age;wage_per_hour');

CORR
-------------------
 0.036938290262767
(1 row)
```

```sql
CALL nza..CORR('intable=nza..CensusIncome, incolumn=age;capital_gains;wage_per_hour;objweight');

CORR
-------------------
```

CALL nza..CORR('intable=nza..CensusIncome,
ingcolumn=age;wage_per_hour, outtable=resultTable, by=sex');
SELECT * FROM resultTable ORDER BY sex;
CALL nza..DROP_TABLE('resultTable');

<table>
<thead>
<tr>
<th>CORRELATION</th>
<th>SEX</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.025374759165581</td>
<td>Female</td>
</tr>
<tr>
<td>0.050701396395573</td>
<td>Male</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DROP_TABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
</tr>
</tbody>
</table>

**Related Functions**

- category Analytics - Statistics
- CANONICAL_CORR
- CORRELATION1000MATRIX
- CORRELATION500PAIRS
- COV

**CORR_AGG - Pearson’s Correlation**

This function calculates the correlation between two numeric input variables.
Usage

The CORR_AGG aggregate has the following syntax:

► CORR_AGG(DOUBLE X, DOUBLE Y)

▲ Parameters

► X
the first variable to correlate
Type: DOUBLE

► Y
the second variable to correlate
Type: DOUBLE

▲ Returns
DOUBLE the Pearson's correlation, or NULL if there is too few data

Details

This function calculates the correlation of two numeric input variables. The Pearson's correlation coefficient measures the strength of relationship between linearly dependent variables that are normally distributed. It takes a value between -1 (inversely correlated) and 1 (correlated), 0 means that the two variables are independent.

Examples

```
SELECT nza..CORR_AGG(petallength,petalwidth) FROM
nza..iris;
```

```
CORR_AGG
------------------
0.96275709705097
(1 row)
```

Related Functions

► category Analytics - Statistics
► CORR
► COV_AGG

CORR_AGG - Pearson's Correlation with weight

This function calculate the correlation between two numeric input variables, each pair of values being weighted.
Usage
The CORR_AGG aggregate has the following syntax:

► CORR_AGG(DOUBLE X, DOUBLE Y, DOUBLE W)
  ▲ Parameters
    ► X
      the first variable to correlate
      Type: DOUBLE
    ► Y
      the second variable to correlate
      Type: DOUBLE
    ► W
      the weight of the (X,Y) pair
      Type: DOUBLE
  ▲ Returns
    DOUBLE the Pearson's correlation, or NULL if there is too few data

Details
This function calculates the correlation of two numeric input variables, modulated by a weight for each pair of values. The Pearson's correlation coefficient measures the strength of relationship between linearly dependent variables that are normally distributed. It takes a value between -1 (inversely correlated) and 1 (correlated), 0 means that the two variables are independent.

Examples

```
SELECT nza..CORR_AGG(age, capital_gain, hours_per_week) FROM nza..adult;

CORR_AGG
------------------
0.08016233217621
(1 row)
```

Related Functions
► category Analytics - Statistics
► CORR
► COV_AGG

CORR_MATRIX_AGG - Correlation Matrix
This function calculates the matrix of correlations between pairs of numeric input variables divided into two sets X and Y. It takes as arguments the number of variables in the set X, the number of variables in the set Y, and the variables themselves, and computes the correlations between all pairs containing a variable of set X and a variable of set Y.

Usage

The CORR_MATRIX_AGG aggregate has the following syntax:

\[ \text{CORR\_MATRIX\_AGG}(\text{INT4 noX}, \text{INT4 noY}, \text{DOUBLE X1, X2, etc.}, \text{DOUBLE Y1, Y2, etc.}) \]

- **Parameters**
  - **noX**
    - the number of variables in set X
    - Type: INT4
    - Min: 1
    - Max: 61-noY
  - **noY**
    - the number of variables in set Y
    - Type: INT4
    - Min: 1
    - Max: 61-noX
  - **X1, X2, etc.**
    - the first, second, etc... variable in set X
    - Type: DOUBLE
  - **Y1, Y2, etc.**
    - the first, second, etc... variable in set Y
    - Type: DOUBLE

- **Returns**
  - NVARCHAR(10000) the string with the matrix of correlations between the variables of sets X and Y

Details

This function calculates the matrix of correlations between pairs of numeric input variables taken from two sets X and Y. Correlation is a measure saying how easily it is to predict one variable from the other. It takes a value between -1 (inversely correlated) and 1 (correlated), 0 means that the two variables are independent.

The major limitation is that the number of input variables cannot exceed 61.

This function returns a string containing the correlation matrix between columns of sets X and Y. Columns of set X are the rows of the matrix, columns of set Y the columns of the matrix.
Examples

```sql
SELECT nza..CORR_MATRIX_AGG(2,3,petallength,petalwidth,sepallength,sepalwidth,petalwidth) FROM nza..iris;
```

```
CORR_MATRIX_AGG
------------------------------------------
------------------------------------------
               our result
 0     0.871754   -0.420516    0.962757
 1     0.817954   -0.356544    1
(1 row)
```

Related Functions

- category Analytics - Statistics
- CORRELATION1000MATRIX
- CORR_AGG
- COV_MATRIX_AGG

**CORRELATION1000MATRIX - Correlation matrix**

This stored procedure calculates the matrix of correlations between pairs of numeric columns divided in two sets

**Usage**

The CORRELATION1000MATRIX stored procedure has the following syntax:

- **CORRELATION1000MATRIX**(NVARCHAR(ANY) paramString)
  - **Parameters**
    - **paramString**
      - comma-separated list of <parameter>=<value> entries with parameters below
        - **Type**: NVARCHAR(ANY)
    - **intable**
      - the input table
        - **Type**: NVARCHAR(256)
    - **outtable**
the output table to write the correlations into

Type: NVARCHAR(256)

► incolumn
the numeric input table columns to be correlated, separated by a semicolon (;). Each
column is followed by :X or :Y to indicate it belongs to set X or set Y. If neither :X nor :Y
is specified for any of the input columns, the matrix contains the correlations between
all pairs of input columns.

Type: NVARCHAR(ANY)

► by
the input table column which splits the data into groups for which the operation is to
be performed

Type: NVARCHAR(128)

Default: <none>

▲ Returns
INT8 the number of correlations written into the output table

Details
This stored procedure calculates the matrix of correlations between pairs of numeric input
columns divided into two sets X and Y, either in the whole input table or within the groups defined
in the column specified by parameter <by>. Correlation is a measure saying how easily it is to pre-
dict one column from the other. It takes a value between -1 (inversely correlated) and 1 (correl-
ated), 0 means that the two columns are independent.

The output table is created with following column: varxname, varyname, correlation. If the para-
meter by is specified, an additional column grouped_on is added to indicate for which group the
correlations have been calculated.

Examples

CALL nza..CORRELATION1000MATRIX('intable=nza..iris,
in-column=sepallength;sepalwidth;petallength;petalwidth,
outtable=Result7');

SELECT * FROM result7 ORDER BY varxname, varyname;

CALL nza..DROP_TABLE('Result7');

CORRELATION1000MATRIX

-----------------------

16

(1 row)

<table>
<thead>
<tr>
<th>VARXNAME</th>
<th>VARYNAME</th>
<th>CORRELATION</th>
</tr>
</thead>
</table>
## Reference Documentation: Analytics

### Correlation Matrix

<table>
<thead>
<tr>
<th>Column 1</th>
<th>Column 2</th>
<th>Correlation</th>
</tr>
</thead>
<tbody>
<tr>
<td>PETALLENGTH</td>
<td>PETALLENGTH</td>
<td>1</td>
</tr>
<tr>
<td>PETALLENGTH</td>
<td>PETALWIDTH</td>
<td>0.96275709705097</td>
</tr>
<tr>
<td>PETALLENGTH</td>
<td>SEPALLENGTH</td>
<td>0.87175415730488</td>
</tr>
<tr>
<td>PETALLENGTH</td>
<td>SEPALWIDTH</td>
<td>-0.42051609640115</td>
</tr>
<tr>
<td>PETALWIDTH</td>
<td>PETALLENGTH</td>
<td>0.96275709705097</td>
</tr>
<tr>
<td>PETALWIDTH</td>
<td>PETALWIDTH</td>
<td>1</td>
</tr>
<tr>
<td>PETALWIDTH</td>
<td>SEPALLENGTH</td>
<td>0.81795363336916</td>
</tr>
<tr>
<td>PETALWIDTH</td>
<td>SEPALWIDTH</td>
<td>-0.3565440896138</td>
</tr>
<tr>
<td>SEPALLENGTH</td>
<td>PETALLENGTH</td>
<td>0.87175415730488</td>
</tr>
<tr>
<td>SEPALLENGTH</td>
<td>PETALWIDTH</td>
<td>0.81795363336916</td>
</tr>
<tr>
<td>SEPALLENGTH</td>
<td>SEPALLENGTH</td>
<td>1</td>
</tr>
<tr>
<td>SEPALLENGTH</td>
<td>SEPALWIDTH</td>
<td>-0.10936924995065</td>
</tr>
<tr>
<td>SEPALWIDTH</td>
<td>PETALLENGTH</td>
<td>-0.42051609640115</td>
</tr>
<tr>
<td>SEPALWIDTH</td>
<td>PETALWIDTH</td>
<td>-0.3565440896138</td>
</tr>
<tr>
<td>SEPALWIDTH</td>
<td>SEPALLENGTH</td>
<td>-0.10936924995063</td>
</tr>
<tr>
<td>SEPALWIDTH</td>
<td>SEPALWIDTH</td>
<td>1</td>
</tr>
</tbody>
</table>

(16 rows)

### Drop Table

```
DROP_TABLE

t
```

(1 row)

```sql
CALL nza..CORRELATION1000MATRIX('intable=nza..iris,
incolumn=sepalwidth:X;sepalwidth:X;petallength:X;petalwidth:X;petalwidth:Y,
outtable=Result7');
SELECT * FROM result7 ORDER BY varxname, varyname;
CALL nza..DROP_TABLE('Result7');
```

CORRELATION1000MATRIX

```
4
```

(1 row)
<table>
<thead>
<tr>
<th>VARXNAME</th>
<th>VARYNAME</th>
<th>CORRELATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEPALLENGTH</td>
<td>PETALLENGTH</td>
<td>0.87175415730488</td>
</tr>
<tr>
<td>SEPALLENGTH</td>
<td>PETALWIDTH</td>
<td>0.81795363336916</td>
</tr>
<tr>
<td>SEPALWIDTH</td>
<td>PETALLENGTH</td>
<td>-0.42051609640115</td>
</tr>
<tr>
<td>SEPALWIDTH</td>
<td>PETALWIDTH</td>
<td>-0.3565440896138</td>
</tr>
</tbody>
</table>

(4 rows)

DROP_TABLE

--------
t

(1 row)

CALL nza..CORRELATION1000MATRIX('intable=nza..iris,
by=class,
incolumn=sepallength:X;sepalwidth:X;petallength:Y;petalwidth:Y, outtable=Result7');

SELECT * FROM result7 ORDER BY grouped_on, varxname, varyname;

CALL nza..DROP_TABLE('Result7');

<table>
<thead>
<tr>
<th>GROUPED_ON</th>
<th>VARXNAME</th>
<th>VARYNAME</th>
<th>CORRELATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>setosa</td>
<td>SEPALLENGTH</td>
<td>PETALLENGTH</td>
<td>0.26387409291868</td>
</tr>
<tr>
<td>setosa</td>
<td>SEPALLENGTH</td>
<td>PETALWIDTH</td>
<td>0.2790915749996</td>
</tr>
</tbody>
</table>
### Related Functions

- category Analytics - Statistics
- CANONICAL_CORR
- CORR
- CORRELATION500PAIRS
- COVARIANCE1000MATRIX

### CORRELATION500PAIRS - Correlation of pairs of variables

This stored procedure calculates correlations between pairs of numeric columns.

#### Usage

The CORRELATION500PAIRS stored procedure has the following syntax:

```
CORRELATION500PAIRS(NVARCHAR(ANY) paramString)
```

- **Parameters**
  - `paramString` comma-separated list of `<parameter>=<value>` entries with parameters below
  - Type: NVARCHAR(ANY)
### Returns

**INT8** the number of correlations written into the output table

### Details

This stored procedure calculates the correlations between the given pairs of numeric input columns, either in the whole input table or within the groups defined in the column specified by parameter `<by>`. Correlation is a measure saying how easily it is to predict one column from the other. It takes a value between -1 (inversely correlated) and 1 (correlated), 0 means that the two columns are independent.

The output table is created with following column: grouped_on, varxname, varyname, correlation. If the parameter `by` is not specified, the column `grouped_on` contains -1. Otherwise, it contains the groups defined in the input column `<by>`.

### Examples

```sql
CALL nza..CORRELATION500PAIRS('intable=nza..iris, incolumn=sepalength:petallength;sepalwidth:petalwidth, outtable=Result7');
SELECT * FROM result7 ORDER BY grouped_on, varxname, varyname;
CALL nza..DROP_TABLE('Result7');
CORRELATION500PAIRS
---------------------
2
(1 row)
```
<table>
<thead>
<tr>
<th>GROUPED_ON</th>
<th>VARXNAME</th>
<th>VARYNAME</th>
<th>CORRELATION</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SEPALLENGTH</td>
<td>PETALLENGTH</td>
<td>-0.87175415730488</td>
</tr>
<tr>
<td></td>
<td>SEPALWIDTH</td>
<td>PETALWIDTH</td>
<td>-0.35654408961381</td>
</tr>
</tbody>
</table>

(2 rows)

DROP_TABLE

-----
t

(1 row)

CALL nza..CORRELATION500PAIRS('intable=nza..iris,
incolumn=sepallength:petallength;sepalwidth:petalwidth,
outtable=Result7, by=class');

SELECT * FROM result7 ORDER BY grouped_on, varxname, varyname;

CALL nza..DROP_TABLE('Result7');

<table>
<thead>
<tr>
<th>GROUPED_ON</th>
<th>VARXNAME</th>
<th>VARYNAME</th>
<th>CORRELATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>setosa</td>
<td>SEPALLENGTH</td>
<td>PETALLENGTH</td>
<td>0.26387409291868</td>
</tr>
<tr>
<td>setosa</td>
<td>SEPALWIDTH</td>
<td>PETALWIDTH</td>
<td>0.2799728885169</td>
</tr>
<tr>
<td>versicolor</td>
<td>SEPALLENGTH</td>
<td>PETALLENGTH</td>
<td>0.754048958592</td>
</tr>
<tr>
<td>versicolor</td>
<td>SEPALWIDTH</td>
<td>PETALWIDTH</td>
<td>0.66399872002411</td>
</tr>
<tr>
<td>virginica</td>
<td>SEPALLENGTH</td>
<td>PETALLENGTH</td>
<td>0.86422473293558</td>
</tr>
<tr>
<td>virginica</td>
<td>SEPALWIDTH</td>
<td>PETALWIDTH</td>
<td>0.53772802626619</td>
</tr>
</tbody>
</table>

(6 rows)

DROP_TABLE

---
Related Functions

- category Analytics - Statistics
- CANONICAL_CORR
- CORR
- CORRELATION1000MATRIX
- COVARIANCE500PAIRS

COV - Covariance

This stored procedure calculates the covariance between two numeric columns.

Usage

The COV stored procedure has the following syntax:

`COV(NVARCHAR(ANY) paramString)`

- **Parameters**
  - **paramString**
    - comma-separated list of `<parameter>=<value>` entries with parameters below
    - Type: NVARCHAR(ANY)
  - **intable**
    - the input table
    - Type: NVARCHAR(256)
  - **incolumn**
    - the two numeric input table column, separated by a semicolon (;)
    - Type: NVARCHAR(ANY)
  - **by**
    - the input table column which splits the data into groups for which the operation is to be performed. If specified, an output table must be indicated.
    - Type: NVARCHAR(128)
    - Default: <none>
  - **outtable**
    - the output table to write the covariances into. This parameter must be specified if parameter by is specified.
Returns
DOUBLE the covariance between the two input table columns, or the number of groups in the output table for which a covariance has been calculated.

Details
This stored procedure calculates the covariance of two numeric input columns, either in the whole input table or within the groups defined in the column specified by parameter <by>. Covariance is a measure saying how easily it is to predict one column from the other. It takes a negative value when both columns are inversely correlated, a positive value when they are correlated, 0 means that the two columns are independent.

If parameter 'by' is specified, an output table must be specified. The output table is created with following columns: <by>, covariance. The covariance between the two input columns is given for each group of the <by> column.

Examples

```
CALL nza..COV('intable=nza..CensusIncome, incolumn=age;wage_per_hour');

COV
------------------
226.54940380617
(1 row)

CALL nza..COV('intable=nza..CensusIncome, incolumn=age;wage_per_hour, outtable=resultTable, by=sex');
SELECT * FROM resulttable ORDER BY sex;
CALL nza..DROP_TABLE('resulttable');

COV
-----
2
(1 row)

COVARIANCE | SEX
-----------+--------
142.43203639029 | Female
333.74358352398 | Male
```
(2 rows)

DROP_TABLE
------------
t
(1 row)

Related Functions
► category Analytics - Statistics
► COVARIANCEMATRIX
► COVARIANCE1000MATRIX
► COVARIANCE500PAIRS
► CORR

COV_AGG - Pearson's Covariance
This function calculate the covariance between two numeric input variables

Usage
The COV_AGG aggregate has the following syntax:

► COV_AGG(DOUBLE X, DOUBLE Y)
  ▲ Parameters
  ► X
    the first variable
    Type: DOUBLE
  ► Y
    the second variable
    Type: DOUBLE
  ▲ Returns
    DOUBLE the Pearson's covariance, or NULL if there is too few data

Details
This function calculates the covariance of two numeric input variables. The Pearson’s covariance coefficient measures the strength of relationship between linearly dependent variables that are normally distributed. It takes a negative value when both variables are inversely correlated, a positive value when they are correlated, 0 means that the two variables are independent.
Examples

```sql
SELECT nza..COV_AGG(petal_length, petal_width) FROM nza..iris;
```

```
COV_AGG
-----------
1.2963874720358
(1 row)
```

Related Functions

- category Analytics - Statistics
- CORR_AGG
- COV

**COV_MATRIX_AGG - Covariance Matrix**

This function calculates the matrix of covariances between pairs of numeric input variables divided into two sets X and Y. It takes as arguments the number of variables in the set X, the number of variables in the set Y, and the variables themselves, and computes the covariances between all pairs containing a variable of set X and a variable of set Y.

**Usage**

The `COV_MATRIX_AGG` aggregate has the following syntax:

```sql
COV_MATRIX_AGG(INT4 noX, INT4 noY, DOUBLE X1, DOUBLE X2, DOUBLE Y1, DOUBLE Y2, DOUBLE Y3)
```

- **Parameters**
  - **noX**
    - the number of variables in set X
    - Type: INT4
    - Min: 1
    - Max: 61-noY
  - **noY**
    - the number of variables in set Y
    - Type: INT4
    - Min: 1
    - Max: 61-noX
  - **X1, X2, etc.**
    - the first, second, etc... variable in set X
Type: DOUBLE

► Y1,Y2,etc.
the first, second, etc... variable in set Y

Type: DOUBLE

▲ Returns
NVARCHAR(10000) the string with the matrix of covariances between the variables of sets X and Y

Details
This function calculates the matrix of covariances between pairs of numeric input variables taken from two sets X and Y. Covariance is a measure saying how easily it is to predict one variable from the other. It takes a negative value when both variables are inversely correlated, a positive value when they are correlated, 0 means that the two variables are independent.

The major limitation is that the number of input variables cannot exceed 61.

This function returns a string containing the covariance matrix between columns of sets X and Y. Columns of set X are the rows of the matrix, columns of set Y the columns of the matrix.

Examples

SELECT nza..COV_MATRIX_AGG(2,3,petallength,petalwidth,sepallength,sepalwidth,petalwidth) FROM nza..iris;

COV_MATRIX_AGG
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
our result
0    1.27368    -0.321713
1.29639
1    0.516904    -0.117981
0.582414
(1 row)

Related Functions
► category Analytics - Statistics
► CORR_MATRIX_AGG
COVARIANCE1000MATRIX - Covariance matrix

This stored procedure calculates the matrix of covariances between pairs of numeric columns divided in two sets.

Usage

The COVARIANCE1000MATRIX stored procedure has the following syntax:

- \textbf{COVARIANCE1000MATRIX(NVARCHAR(ANY) \ paramString)}

\begin{itemize}
  \item Parameters
    \begin{itemize}
      \item \textbf{paramString}
        comma-separated list of \texttt{<parameter>=<value>} entries with parameters below
        Type: NVARCHAR(ANY)
      \item \textbf{inTable}
        the input table
        Type: NVARCHAR(256)
      \item \textbf{outTable}
        the output table to write the covariances into
        Type: NVARCHAR(256)
      \item \textbf{inColumn}
        the numeric input table columns to calculate covariances for, separated by a semicolon (\,\,). Each column is followed by \,X or \,Y to indicate it belongs to set X or set Y. If neither \,X nor \,Y is specified for any of the input columns, the matrix contains the covariances between all pairs of input columns.
        Type: NVARCHAR(ANY)
      \item \textbf{by}
        the input table column which splits the data into groups for which the operation is to be performed
        Type: NVARCHAR(128)
        Default: \texttt{<none>} remark: If by is equal null, then the group is ignored.
    \end{itemize}
\end{itemize}

\begin{itemize}
  \item Returns
    INT8 the number of covariances written into the output table
\end{itemize}

Details

This stored procedure calculates the matrix of covariances between pairs of numeric input columns divided into two sets X and Y, either in the whole input table or within the groups defined in the column specified by parameter \texttt{<by>}. Covariance is a measure saying how easily it is to predict one column from the other. It takes a negative value when both columns are inversely correlated, a positive value when they are correlated, 0 means that the two columns are independent.
The output table is created with following column: varxname, varyname, covariance, cntx. If the parameter by is specified, an additional column grouped_on is added to indicate for which group the covariances have been calculated.

Examples

```sql
CALL nza..COVARIANCE1000MATRIX('intable=nza..iris, incolumn=sepallength;sepalwidth;petallength;petalwidth, outtable=Result7');
SELECT * FROM result7 ORDER BY varxname, varyname;
CALL nza..DROP_TABLE('Result7');
```

<table>
<thead>
<tr>
<th>VARXNAME</th>
<th>VARYNAME</th>
<th>COVARIANCE</th>
<th>CNTX</th>
</tr>
</thead>
<tbody>
<tr>
<td>PETALLENGTH</td>
<td>PETALLENGTH</td>
<td>3.1131794183445</td>
<td>150</td>
</tr>
<tr>
<td>PETALLENGTH</td>
<td>PETALWIDTH</td>
<td>1.2963874720358</td>
<td>150</td>
</tr>
<tr>
<td>PETALLENGTH</td>
<td>SEPALLENGTH</td>
<td>1.2736823266219</td>
<td>150</td>
</tr>
<tr>
<td>PETALLENGTH</td>
<td>SEPALWIDTH</td>
<td>-0.32171275167785</td>
<td>150</td>
</tr>
<tr>
<td>PETALWIDTH</td>
<td>PETALLENGTH</td>
<td>1.2963874720358</td>
<td>150</td>
</tr>
<tr>
<td>PETALWIDTH</td>
<td>PETALWIDTH</td>
<td>0.58241431767338</td>
<td>150</td>
</tr>
<tr>
<td>PETALWIDTH</td>
<td>SEPALLENGTH</td>
<td>0.51690380313199</td>
<td>150</td>
</tr>
<tr>
<td>PETALWIDTH</td>
<td>SEPALWIDTH</td>
<td>-0.11798120805369</td>
<td>150</td>
</tr>
<tr>
<td>SEPALLENGTH</td>
<td>PETALLENGTH</td>
<td>1.2736823266219</td>
<td>150</td>
</tr>
<tr>
<td>SEPALLENGTH</td>
<td>PETALWIDTH</td>
<td>0.51690380313199</td>
<td>150</td>
</tr>
<tr>
<td>SEPALLENGTH</td>
<td>SEPALLENGTH</td>
<td>0.68569351230425</td>
<td>150</td>
</tr>
<tr>
<td>SEPALLENGTH</td>
<td>SEPALWIDTH</td>
<td>-0.039268456375831</td>
<td>150</td>
</tr>
<tr>
<td>SEPALWIDTH</td>
<td>PETALLENGTH</td>
<td>-0.32171275167785</td>
<td>150</td>
</tr>
<tr>
<td>SEPALWIDTH</td>
<td>PETALWIDTH</td>
<td>-0.11798120805369</td>
<td>150</td>
</tr>
<tr>
<td>SEPALWIDTH</td>
<td>SEPALLENGTH</td>
<td>-0.039268456375831</td>
<td>150</td>
</tr>
<tr>
<td>SEPALWIDTH</td>
<td>SEPALWIDTH</td>
<td>0.18800402684564</td>
<td>150</td>
</tr>
</tbody>
</table>
CALL nza..COVARIANCE1000MATRIX('intable=nza..iris, incolumn=sepallength:X;sepalwidth:X;petallength:Y;petalwidth:Y, outtable=Result7');
SELECT * FROM result7 ORDER BY varxname, varyname;
CALL nza..DROP_TABLE('Result7');

CALL nza..COVARIANCE1000MATRIX('by=class, intable=nza..iris, incolumn=sepallength:X;sepalwidth:X;petallength:Y;petalwidth:Y, outtable=Result7');
SELECT * FROM result7 ORDER BY varxname, varyname;
```sql
outtable=Result7');

SELECT * FROM result7 ORDER BY grouped_on, varxname, varyname;

CALL nza..DROP_TABLE('Result7');

COVARIANCE1000MATRIX
----------------------
12
(1 row)

GROUPED_ON | VARXNAME   | VARYNAME   |    COVARIANCE
Cnty
------------+-------------+-------------+----------
setosa     | SEPALLENGTH | PETALLENGTH | 0.016138775510204 | 50
setosa     | SEPALLENGTH | PETALWIDTH  | 0.01054693877551 | 50
setosa     | SEPALWIDTH  | PETALLENGTH | 0.011681632653061 | 50
setosa     | SEPALWIDTH  | PETALWIDTH  | 0.011436734693878 | 50
versicolor | SEPALLENGTH | PETALLENGTH | 0.18289795918368 | 50
versicolor | SEPALLENGTH | PETALWIDTH  | 0.055779591836735 | 50
versicolor | SEPALWIDTH  | PETALLENGTH | 0.082653061224491 | 50
versicolor | SEPALWIDTH  | PETALWIDTH  | 0.041204081632653 | 50
virginica  | SEPALLENGTH | PETALLENGTH | 0.30328979591837 | 50
virginica  | SEPALLENGTH | PETALWIDTH  | 0.049093877551019 | 50
virginica  | SEPALWIDTH  | PETALLENGTH | 0.071379591836735 | 50
virginica  | SEPALWIDTH  | PETALWIDTH  |
Related Functions

- category Analytics - Statistics
- CORRELATION1000MATRIX
- COV
- COVARIANCEMATRIX
- COVARIANCE500PAIRS

COVARIANCE500PAIRS - Covariance of pairs of variables

This stored procedure calculates covariances between pairs of numeric columns.

Usage

The COVARIANCE500PAIRS stored procedure has the following syntax:

```sql
COVARIANCE500PAIRS(NVARCHAR(ANY) paramString)
```

- **Parameters**
  - **paramString**
    - comma-separated list of `<parameter>=<value>` entries with parameters below
    - Type: NVARCHAR(ANY)
  - **inTable**
    - the input table
    - Type: NVARCHAR(256)
  - **outTable**
    - the output table to write the covariances into
    - Type: NVARCHAR(256)
  - **incolumn**
    - the pairs of numeric input table columns. The pairs are separated by a semicolon (;), the columns of a pair are separated by a colon (:).
    - Type: NVARCHAR(ANY)
by
the input table column which splits the data into groups for which the operation is to be performed
Type: NVARCHAR(128)
Default: <none>

Returns
INT8 the number of covariances written into the output table

Details
This stored procedure calculates the covariances between the given pairs of numeric input columns, either in the whole input table or within the groups defined in the column specified by parameter <by>. Covariance is a measure saying how easily it is to predict one column from the other. It takes a negative value when both columns are inversely correlated, a positive value when they are correlated, 0 means that the two columns are independent.

The output table is created with following column: grouped_on, varxname, varyname, covariance. If the parameter by is not specified, the column grouped_on contains -1. Otherwise, it contains the groups defined in the input column <by>.

Examples

CALL nza..COVARIANCE500PAIRS('intable=nza..iris, incolumn=sepallength:petallength;sepalwidth:petalwidth, outtable=Result7');

SELECT * FROM result7 ORDER BY grouped_on, varxname, varyname;

CALL nza..DROP_TABLE('Result7');

COVARIANCE500PAIRS
---------------------
2
(1 row)

GROUPED_ON | VARXNAME   | VARYNAME   | COVARIANCE
-----------|-------------|-------------|------------
-1          | SEPALLENGTH | PETALLENGTH | 1.2736823266219
-1          | SEPALWIDTH  | PETALWIDTH  | -0.11798120805369
(2 rows)
CALL nza..COVARIANCE500PAIRS('intable=nza..iris,
incolumn=sepal length:petal length; sepal width: petal width,
outtable=Result7, by=class');
SELECT * FROM result7 ORDER BY grouped_on, var xname, varyname;
CALL nza..DROP_TABLE('Result7');

GROUPED_ON      |  VARXNAME   |  VARYNAME   |    COVARIANCE  
----------------+-------------+-------------+-------------------
setosa          |  SEPALLENGTH |  PETALLENGTH |  0.016138775510204
setosa          |  SEPALWIDTH  |  PETALWIDTH  |  0.011436734693878
versicolor      |  SEPALLENGTH |  PETALLENGTH |  0.18289795918368
versicolor      |  SEPALWIDTH  |  PETALWIDTH  |  0.041204081632653
virginica       |  SEPALLENGTH |  PETALLENGTH |  0.30328979591837
virginica       |  SEPALWIDTH  |  PETALWIDTH  |  0.047628571428573
(6 rows)
Related Functions

- category Analytics - Statistics
- CORRELATION500PAIRS
- COV
- COVARIANCEMATRIX
- COVARIANCE1000MATRIX

COVARIANCEMATRIX - Covariance matrix as string

This stored procedure calculates the matrix of covariances between pairs of numeric columns divided in two sets. It writes the matrix as a string into the output table.

Usage

The COVARIANCEMATRIX stored procedure has the following syntax:

- COVARIANCEMATRIX(NVARCHAR(ANY) paramString)

  ▲ Parameters

  - paramString
    comma-separated list of <parameter>=<value> entries with parameters below
    Type: NVARCHAR(ANY)
  - inable
    the input table
    Type: NVARCHAR(256)
  - outtable
    the output table to write the covariances into
    Type: NVARCHAR(256)
  - incolumn
    the numeric input table columns to calculate covariances for, separated by a semicolon (;). Each column is followed by :X or :Y to indicate it belongs to set X or set Y. If neither :X nor :Y is specified for any of the input columns, the matrix contains the covariances between all pairs of input columns.
    Type: NVARCHAR(ANY)
  - by
    the input table column which splits the data into groups for which the operation is to be performed
    Type: NVARCHAR(128)
    Default: <none>

  ▲ Returns

  INT8 the number of groups for which a covariance matrix has been calculated
Details
This stored procedure calculates the matrix of covariances between pairs of numeric input columns divided into two sets X and Y, either in the whole input table or within the groups defined in the column specified by parameter <by>. Covariance is a measure saying how easily it is to predict one column from the other. It takes a negative value when both columns are inversely correlated, a positive value when they are correlated, 0 means that the two columns are independent.

The output table is created with following string column: covariance. The covariance columns contains a string with the covariance matrix: columns of set X are the rows of the matrix, columns of set Y the columns of the matrix. If the parameter by is specified, an additional column grouped_on is added to indicate for which group the covariances have been calculated.

Examples
CALL nza..COVARIANCEMATRIX('intable=nza..CensusIncome, incolumn=age:X;wage_per_hour:Y, outtable=resultTable');
SELECT * FROM resulttable;
CALL nza..DROP_TABLE('resulttable');

COVARIANCEMATRIX
------------------
1
(1 row)

COVARIANCE
------------------------------------------------
our result

  0      226.549
(1 row)

DROP_TABLE
------------
t
(1 row)

CALL nza..COVARIANCEMATRIX('intable=nza..CensusIncome, incolumn=wage_per_hour:X;capital_gains:X;age:Y, outtable=resultTable, by=sex');
SELECT * FROM resulttable ORDER BY sex;
CALL nza..DROP_TABLE('resulttable');

COVARIANCE
------------------

2

(1 row)

COVARIANCE

| SEX 
|---------------------------------------------------------
| ----------------------------------------+--------
| our result

0 142.432
1 1921.78  | Female

our result

0 333.744
1 10270.4  | Male

(2 rows)

DROP_TABLE

-----------

(1 row)

Related Functions

► category Analytics - Statistics
► COV
► COV_MATRIX_AGG
► COVARIANCE1000MATRIX
► COVARIANCE500PAIRS
CROSS_VALIDATION - Build and cross validate a Classification model

This stored procedure builds a Classification model and cross validates it. Cross validation is a technique for estimating the quality and performance of a predictive model.

Usage

The CROSS_VALIDATION stored procedure has the following syntax:

```
CROSS_VALIDATION(NVARCHAR(ANY) paramString)
```

Parameters

- **paramString**
  - comma-separated list of `<parameter>=<value>` entries with parameters below
  - Type: NVARCHAR(ANY)
- **modelType**
  - the name of the procedure that builds a Classification model (e.g. naivebayes, dectree, knn)
  - Type: NVARCHAR(ANY)
- **model**
  - the name of the Classification model to build
  - Type: NVARCHAR(ANY)
- **intable**
  - the input table
  - Type: NVARCHAR(256)
- **id**
  - the input table column identifying a unique instance id
  - Type: NVARCHAR(128)
- **target**
  - the input table column representing the class
  - Type: NVARCHAR(128)
- **outtable**
  - the output table where the predictions will be stored
  - Type: NVARCHAR(256)
- **folds**
  - the number of cross validation subsets
  - Type: INTEGER
  - Default: 10
- **seed**
  - the seed of the random function
  - Type: FLOAT
  - Default: random()
Returns
 FLOAT the prediction accuracy and -1 if accuracy cannot be calculated

Details
This stored procedure builds a Classification model. The model is saved to the database in a set of tables and registered in the database model metadata. Use the Model Management functions to further manipulate the model or access the model tables.

All parameters specific to a the given Classification stored procedure can be defined too: they will be used when building and testing the model.

A table <outtable> is created with the following columns: id, class, fold. The id column matches the <id> column from intable, class is the predicted class label and fold is the fold number when the prediction was made. The size of this table is <folds> time the size of <intable>.

Examples

CALL nza..CROSS_VALIDATION('modelType=dectree,
intable=nza..iris, folds=3, model=iris_c45, target=class,
 id=id, outtable=iris_pred, seed=12345');

CALL nza..DROP_MODEL('model=iris_c45');

CALL nza..DROP_TABLE('iris_pred');

CROSS_VALIDATION

------------------

0.94

(1 row)

DROP_MODEL

--------------

(1 row)

DROP_TABLE

--------------

(1 row)

Related Functions

► category Analytics - Classification
► PERCENTAGE_SPLIT
► TRAIN_TEST
CUMULATIVE - Cumulative Distributions

This stored procedure computes the probability that a variable following a given distribution takes a value smaller or equal to x.

Usage
The CUMULATIVE stored procedure has the following syntax:

```sql
CUMULATIVE(NVARCHAR(ANY) paramString)
```

Parameters

- **paramString**
  - comma-separated list of <parameter>=<value> entries with parameters below
  - Type: NVARCHAR(ANY)

- **intable**
  - the input table
  - Type: NVARCHAR(256)

- **type**
  - the kind of distribution(s) to use:
    - n - normal (PNORM)
    - c - chi-square (PCHISQ)
    - t - t-student (PT)
    - u - uniform (PUNIF)
    - e - exponential (PEXP)
    - w - Weibull (PWEIBULL)
  - Note that several distributions can be calculated simultaneously by concatenating the distribution abbreviations in the parameter value.
  - Type: NVARCHAR(ANY)

- **incolumn**
  - the numeric input table column containing the x value(s)
  - Type: NVARCHAR(128)

- **id**
  - the input table column identifying a unique instance id
  - Type: NVARCHAR(128)

- **outtable**
  - the output table
Type: NVARCHAR(256)

► df
the number of degrees of freedom for a student or chi-square distribution
Type: INT8
Default: <none>

► min
the minimum for a uniform distribution
Type: DOUBLE
Default: <none>

► max
the maximum for a uniform distribution
Type: DOUBLE
Default: <none>

► scale
the scale for an exponential or weibull distribution
Type: DOUBLE
Default: <none>
Min: 0.00000000001

► shape
the shape for a weibull distribution
Type: DOUBLE
Default: <none>

► mean
the mean value for a normal distribution
Type: DOUBLE
Default: <none>

► variance
the variance for a normal distribution
Type: NVARCHAR(ANY)
Default: <none>
Min: 0.00000000001

▲ Returns
INT8 the number of input table values for which a probability is calculated

Details
You have a table in which you have computed some summary statistics (chi-square tests etc.) and
you want to know what is the probability that the statistics gets smaller or equal to this value given that this
statistics comes from a given distribution (you may compare alternative distributions if you are not sure).
The CUMULATIVE stored procedure calculates a probability saying how likely the value is under the given dis-
tribution.
- If the probability ranges between 0 and 0.05, the value is close to the left extreme of the distribution,
- If it ranges between 0.05 and 0.95, the value is not to be considered extreme,
- If it ranges between 0.95 and 1, the value is close to the right extreme of the distribution.
The parameters for the distributions can contain literal values or expressions.
The output table has following columns: id, <incolumn>. Additionally, the output table contains a column for
each distribution, named like the cumulative distribution function, and containing the output of this func-
tion. It also contains a column per specific parameter for this distribution, named as the parameter, and
containing the value calculated for it in each row.

Examples

```
CALL nza..CUMULATIVE('intable=nza..iris, id=id, type="n",
incolumn=petallength, outtable=iripetall, mean=3.75,
variance=3.11');
SELECT * FROM iripetall ORDER BY id LIMIT 10;
CALL nza..DROP_TABLE('iripetall');
```

```
CUMULATIVE
-----------

   ID |       PNORM       | PETALLENGTH | MEAN | VARIANCE
-----+-------------------+-------------+------+----------
1    | 0.091337668708794 |         1.4 | 3.75 |     3.11
2    | 0.091337668708794 |         1.4 | 3.75 |     3.11
3    | 0.082375704686719 |         1.3 | 3.75 |     3.11
4    | 0.10100286923082  |         1.5 | 3.75 |     3.11
5    | 0.091337668708794 |         1.4 | 3.75 |     3.11
6    | 0.12252669600944  |         1.7 | 3.75 |     3.11
7    | 0.091337668708794 |         1.4 | 3.75 |     3.11
8    | 0.10100286923082  |         1.5 | 3.75 |     3.11
9    | 0.091337668708794 |         1.4 | 3.75 |     3.11
10   | 0.10100286923082  |         1.5 | 3.75 |     3.11
```
DROP_TABLE
------------

(10 rows)

CREATE TABLE pd_mytable(id INT4, thePoint DOUBLE, df INT4);
INSERT INTO pd_mytable values(1,0,2);
INSERT INTO pd_mytable values(2,1,2);
INSERT INTO pd_mytable values(3,2,2);
INSERT INTO pd_mytable values(4,3,2);
INSERT INTO pd_mytable values(5,3.5,2);
INSERT INTO pd_mytable values(6,3.5,3);
INSERT INTO pd_mytable values(7,3.5,4);
CALL nza..CUMULATIVE('intable=pd_mytable, type="c",
icolumn=thePoint, id=id, outtable=CUMULATIVE_mp_5,
df=df');
SELECT * FROM CUMULATIVE_mp_5 ORDER BY ID;
CALL nza..DROP_TABLE('CUMULATIVE_mp_5');
CALL nza..DROP_TABLE('pd_mytable');

CUMULATIVE
------------

7
(1 row)

<table>
<thead>
<tr>
<th>ID</th>
<th>PCHISQ</th>
<th>THEPOINT</th>
<th>DF</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>0.39346934028786</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
3 | 0.63212055882856 | 2 | 2
4 | 0.77686983985157 | 3 | 2
5 | 0.82622605654955 | 3.5 | 2
6 | 0.67923787919436 | 3.5 | 3
7 | 0.52212165550395 | 3.5 | 4

(7 rows)

DROP_TABLE
------------
t
(1 row)

DROP_TABLE
------------
t
(1 row)

CREATE TABLE pd_mytable(id INT4, thePoint DOUBLE, df INT4);
INSERT INTO pd_mytable values(1,0,2);
INSERT INTO pd_mytable values(2,1,2);
INSERT INTO pd_mytable values(3,2,2);
INSERT INTO pd_mytable values(4,3,2);
INSERT INTO pd_mytable values(5,3.5,2);
INSERT INTO pd_mytable values(6,3.5,3);
INSERT INTO pd_mytable values(7,3.5,4);
CALL nza..CUMULATIVE('intable=pd_mytable, type=ce, incolumn=thePoint, id=id, outtable=CUMULATIVE_mp_5, df=df, scale=10');
SELECT * FROM CUMULATIVE_mp_5 ORDER BY ID;
CALL nza..DROP_TABLE('CUMULATIVE_mp_5');
CALL nza..DROP_TABLE('pd_mytable');

CUMULATIVE
------------
### IBM Netezza In-Database Analytics Reference Guide

#### Table 7

<table>
<thead>
<tr>
<th>ID</th>
<th>PCHISQ</th>
<th>PEXP</th>
<th>THEPOINT</th>
<th>DF</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SCALE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+----+-------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>0.39346934028786</td>
<td>0.09516258196404</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>0.63212055882856</td>
<td>0.18126924692202</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>0.77686983985157</td>
<td>0.25918177931828</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>0.82622605654955</td>
<td>0.29531191028129</td>
<td>3.5</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>0.67923787919436</td>
<td>0.29531191028129</td>
<td>3.5</td>
</tr>
<tr>
<td>7</td>
<td>10</td>
<td>0.52212165550395</td>
<td>0.29531191028129</td>
<td>3.5</td>
</tr>
</tbody>
</table>

(7 rows)

#### SQL Commands

```sql
DROP_TABLE
---------
t
(1 row)

DROP_TABLE
---------
t
(1 row)```
Related Functions

- category Analytics - Probability Distributions
- PCHISQ
- PEXP
- PNORM
- PT
- PUNIF
- PWEIBULL

DBERN - Density of the Bernoulli Distribution

Given the success probability p, this function returns the probability that we get x successes in a single Bernoulli trial.

Usage

The DBERN function has the following syntax:

`DBERN(int8 x, DOUBLE p)`

Parameters

- `x` the value at which to compute
  Type: INT8

- `p` the success probability
  Type: DOUBLE
  Min: 0.000000000001
  Max: 1-0.000000000001

Returns

DOUBLE the probability density at the point x

Details

The probability density is assigned to each discrete event that can happen, here 0 (tails) or 1 (heads).
- DBERN(0,p)=1-p,
- DBERN(1,p)=p.
If p is not between 0 and 1, DBERN(x,p) is null for all x values.

Examples

```
SELECT nza..DBERN(0, 0.4);
```

```
DBERN
-------
```

00X6332-00 Rev. 1
0.6
(1 row)

```
SELECT nza..DBERN(1, 0.4);
```

```
DBERN
-------
 0.4
(1 row)
```

Related Functions

- category Analytics - Probability Distributions
- PBERN
- PBERN_H
- QBERN
- QBERN_H

DBETA - Probability density function for Beta distribution

Given two shape parameters, this function returns the probability density that a variable following the Beta distribution takes a value equal to \( x \)

Usage

The DBETA function has the following syntax:

```
DBETA(DOUBLE x, DOUBLE shapeOne, DOUBLE shapeTwo)
```

- Parameters
  - \( x \)
    - the value at which to compute
    - Type: DOUBLE
  - shapeOne
    - the first shape of the distribution
    - Type: DOUBLE
    - Min: 0.00000000001
  - shapeTwo
the second shape of the distribution

Type: DOUBLE
Min: 0.00000000001

▲ Returns
DOUBLE the probability density at the point x

Details
DBETA(x,shapeOne,shapeTwo)=x^{shapeOne-1}*(1-x)^{shapeTwo-1} / BETA(shapeOne,shapeTwo) for x between 0 and 1.
DBETA(x,shapeOne,shapeTwo)=0 for x outside of this range.
If shapeOne or shapeTwo is 0 or less, DBETA(x,shapeOne,shapeTwo) is null for all x values.

Examples

SELECT nza..DBETA(0.0, 3, 0.4),nza..DBETA(0.1, 3, 0.4),nza..DBETA(0.2, 3, 0.4),nza..DBETA(0.3, 3, 0.4),nza..DBETA(0.4, 3, 0.4);

| DBETA | DBETA | DBETA | DBETA |
|-------+-------+-------+-------|
| 0.0071585286491726 | 0.030730899489516 | 0.074912270202529 | 0.14608260524138 |

(1 row)

Related Functions
► category Analytics - Probability Distributions
► PBETA
► PBETA_H
► QBETA
► QBETA_H

DBINOM - Density of the Binomial Distribution

Given the success probability p and the number of trials, this function returns the probability that we get x successes in a serie of Bernoulli trials.

Usage
The DBINOM function has the following syntax:
DBINOM(INT8 x, INT8 N, DOUBLE p)

Parameters

- **x**
  - the value at which to compute
  - Type: INT8

- **N**
  - the number of trials
  - Type: INT8
  - Min: 1

- **p**
  - the success probability
  - Type: DOUBLE
  - Min: 0.00000000001
  - Max: 1-0.00000000001

Returns

- DOUBLE the probability density at the point x

Details

The probability of having x successes in N trials with probability p of success in each is as follows:

\[ DBINOM(x,N,p) = \frac{N!}{x!(N-x)!} \cdot p^x \cdot (1-p)^{N-x} \]

for all positive x and N values where x <= N and for p between 0 and 1.

If N is 0 or less, DBINOM(x,N,p) is null for all x and p values.

If p is not between 0 and 1, DBINOM(x,p) is null for all x values.

Examples

```
SELECT nza..DBINOM(0,3,0.4), nza..DBINOM(1,3,0.4),
nza..DBINOM(2,3,0.4), nza..DBINOM(3,3,0.4),
nza..DBINOM(4,3,0.4);
```

```
DBINOM | DBINOM | DBINOM | DBINOM | DBINOM
-------+--------+--------+--------+--------
 0.216 |  0.432 |  0.288 |  0.064 |      0
```

(1 row)

```
SELECT nza..dbinom(3,5,0.6), nza..dbinom(30,50,0.6),
nza..dbinom(300,500,0.6), nza..dbinom(3000,5000,0.6),
nza..dbinom(30000,50000,0.6),
nza..dbinom(300000,500000,0.6),
```

\begin{align*}
nza.dbinom(3000000,5000000,0.6),
nza.dbinom(30000000,50000000,0.6);
\end{align*}

\begin{tabular}{|c|c|c|}
\hline
DBINOM & DBINOM & DBINOM \\
\hline
DBINOM & DBINOM & DBINOM \\
\hline
DBINOM & DBINOM \\
\hline
\end{tabular}

\begin{verbatim}
0.3456 0.11455853912702 0.036399043096838 0.011515940477347 0.0036419193276137 0.001151655236622 0.00036419700272489 0.00011498375120855
\end{verbatim}

(1 row)

Related Functions

- category Analytics - Probability Distributions
- PBINOM
- PBINOM_H
- QBINOM
- QBINOM_H

DCAUCHY - Probability density function for Cauchy Distribution

Given the peak location and the interquartile range, this function returns the probability density that a variable following the Cauchy distribution takes a value equal to \( x \)

Usage

The DCAUCHY function has the following syntax:

\begin{verbatim}
DCAUCHY(DOUBLE x, DOUBLE location, DOUBLE scale)
\end{verbatim}

Parameters

- **x**
  - the value at which to compute
  - Type: DOUBLE
- **location**
  - the location of the peak of the Cauchy distribution
  - Type: DOUBLE
- **scale**
  - a value corresponding to half of the interquartile range. Smaller values result in a narrower peak.
  - Type: DOUBLE
Min: 0.00000000001

▲ Returns
DOUBLE the probability density at point x

Details
DCAUCHY(x, location, scale) = 1/ (PI*scale*(1+((x-location)/scale)^2).

The maximum (mode) lies at location.

If scale=0 or less, DCAUCHY(x, location, scale) is null for all x and location values.

Examples

```
SELECT nza..DCAUCHY(0, 3, 0.4),nza..DCAUCHY(1, 3, 0.4),nza..DCAUCHY(2, 3, 0.4),nza..DCAUCHY(3, 3, 0.4),nza..DCAUCHY(4, 3, 0.4);
```

```
DCAUCHY | DCAUCHY | DCAUCHY
-----------------+-------------------+------------------
| DCAUCHY | DCAUCHY

|-----------------+-------------------+------------------|
| 0.01389999502986 | 0.030606719825364 | 0.10976202971855 |
| 0.79577471545948 | 0.10976202971855 |
```

(1 row)

Related Functions

► category Analytics - Probability Distributions
► PCAUCHY
► PCAUCHY_H
► QCAUCHY
► QCAUCHY_H

DCHISQ - Probability density function for Chi-square Distribution

Given a degree of freedom, this function returns the probability density that a variable following the Chi-square distribution takes a value equal to x

Usage

The DCHISQ function has the following syntax:

► DCHISQ(DOUBLE x, INT8 df)

▲ Parameters
x
the value at which to compute
Type: DOUBLE
Min: 0.00000000001

df
the number of degrees of freedom
Type: INT8
Min: 1

Returns
DOUBLE the probability density at point x

Details
DCHISQ(x,df)= x^{df/2-1}*e^{-x/2} / (2^{df/2}*GAMMA(df/2)) for all positive x values.
For x<=0 we assume that DCHISQ(x,df)=0, though there is no practical meaning of this.
If df is 0 or less, DCHISQ(x,df) is null for all x values.

Examples
SELECT nza..DCHISQ(4.6188, 9);

DCHISQ
-------------------
0.079912209787221
(1 row)

Related Functions
► category Analytics - Probability Distributions
► PCHISQ
► PCHISQ_H
► PCHISQ_S
► QCHISQ
► QCHISQ_H

DECTREE - Build then prune a Decision Tree model

This stored procedure builds a Decision Tree model by growing and pruning a tree.

Usage
The DECTREE stored procedure has the following syntax:
DECTREE(NVARCHAR(ANY) paramString)

Parameters

paramString
comma-separated list of <parameter>=<value> entries with parameters below
Type: NVARCHAR(ANY)

model
the name of the Decision Tree model to build
Type: NVARCHAR(ANY)

intable
the input table
Type: NVARCHAR(256)

id
the input table column identifying a unique instance id
Type: NVARCHAR(128)

target
the input table column representing the class
Type: NVARCHAR(128)

incolumn
the input table columns with special properties, separated by a semi-colon (;).
Each column is followed by one or several of the following properties:
- its type: ':nom' (for nominal), ':cont' (for continuous). Per default, all numerical types are continuous, other types are nominal.
- its role: ':id', ':target', ':input', ':ignore'.
(Remark: ':objweight' is unsupported, i.e. ':objweight' same as ':ignore').
(Remark: ':colweight(<wgt>)' is unsupported, i.e. ':colweight(<wgt>)' same as ':colweight(1)' same as ':input').
If the parameter is undefined, all columns of the input table have default properties.
Type: NVARCHAR(ANY)
Default: <none>

coldeftype
default type of the input table columns. Allowed values are 'nom' and 'cont'.
If the parameter is undefined, all numeric columns are considered continuous, other columns nominal.
Type: NVARCHAR(ANY)
Default: <none>

coldefrole
default role of the input table columns. Allowed values are 'input' and 'ignore'.
If the parameter is undefined, all columns are considered 'input' columns.

Type: NVARCHAR(ANY)
Default: <none>

► **colPropertiesTable**
the input table where column properties for the input table columns are stored. The format of this table is the output format of stored procedure nza..COLUMN_PROPERTIES().

If the parameter is undefined, the input table column properties will be detected automatically.

(Remark: colPropertiesTable with "COLROLE" column with value 'objweight' is unsupported, i.e. same as 'ignore')

(Remark: colPropertiesTable with "COLWEIGHT" column with value '"<wgt>" is unsupported, i.e. same as '1')

Type: NVARCHAR(256)
Default: <none>

► **weights**
the input table containing optional instance or class weights for the input table columns.

If the parameter is undefined, we assume that the weights are uniformly equal to 1.

The <weights> table contains following columns:
- weight: a numeric column containing the instance or class weight,
- id: a column to be joined with the <id> column of <intable>, defining instance weights,
- class: a column to be joined with the <target> column of <intable>, defining class weights.

The id or class column can be missing, at least one of them must be present. For instances or classes not occurring in this table, weights of 1 are assumed.

Type: NVARCHAR(256)
Default: <none>

► **eval**
the class impurity measure used for split evaluation. Allowed values are 'entropy' and 'gini'.

Type: NVARCHAR(ANY)
Default: entropy

► **minimprove**
the minimum improvement of the split evaluation measure required

Type: DOUBLE
Default: 0.01
Min: 0.0

► **minsplit**
the minimum number of instances per tree node that can be split

Type: INTEGER
Default: 50
Min: 2

- **maxdepth**
  the maximum number of tree levels (including leaves)
  Type: INTEGER
  Default: 10
  Min: 1
  Max: 62

- **valtable**
  the input table containing the validation dataset. If this parameter is undefined, no pruning will be performed.
  Type: NVARCHAR(256)
  Default: <none>

- **valweights**
  the input table containing optional instance or class weights for the validation dataset. It is similar to the <weights> table.
  Type: NVARCHAR(256)
  Default: <none>

- **qmeasure**
  the quality measure for pruning. Allowed values are Acc or wAcc.
  Type: NVARCHAR(ANY)
  Default: Acc

- **statistics**
  flags indicating which statistics to collect. Allowed values are: none, columns, values:n, all.
  If statistics=none, no statistics are collected.
  If statistics=columns, statistics on the input table columns like mean value are collected.
  If statistics=values:n with n a positive number, statistics about the columns and the column values are collected. Up to <n> column value statistics are collected:
  - If a nominal column contains more than <n> values, only the <n> most frequent column statistics are kept.
  - If a numeric column contains more than <n> values, the values will be discretized and the statistics will be collected on the discretized values.
  Indicating statistics=all is equal to statistics=values:100.
  Type: NVARCHAR(ANY)
  Default: 'none'

▲ Returns
INTEGER the number of Decision Tree nodes (including leaves)

Details
This stored procedure builds a Decision tree model by growing and (optionally) pruning the tree. The model is saved to the database in a set of tables and registered in the database model metadata. Use the Model Management functions to further manipulate the model or access the model tables.

A top-down tree growing algorithm is used with the following features:
- binary splits (equality-based for nominal attributes, inequality-based for continuous attributes),
- no missing value handling,
- entropy or Gini index for split evaluation,
- stop criteria satisfied at a node with a uniform class,
- stop criteria satisfied when further splits do not improve the class impurity by at least <minimprove>,
- stop criteria satisfied when the number of instances is less than <minsplit>,
- stop criteria satisfied when the tree depth reaches <maxdepth>.

A bottom-up reduced error pruning algorithm is used. It bases on the prediction accuracy of the model against the validation dataset. The pruning is activated when parameter <valtable> is specified.

Examples

```
CALL nza..DECTREE('model=adult_tree, intable=nza..adult_train, id=id, target=income, minsplit=1000, eval=entropy, valtable=nza..adult_prune, qmeasure=wAcc');

CALL nza..DROP_MODEL('model=adult_tree');
```

```
DECTREE
--------
13
(1 row)

DROP_MODEL
----------
t
(1 row)
```

Related Functions
- category Analytics - Classification
- GROW_DECTREE
- PRUNE_DECTREE
DENSITY - Density of Distributions

This stored procedure computes the probability that a variable following a given distribution takes a value equal to x.

**Usage**

The DENSITY stored procedure has the following syntax:

```sql
DENSITY(NVARCHAR(ANY) paramString)
```

**Parameters**

- `paramString` 
  comma-separated list of `<parameter>=<value>` entries with parameters below
  
  **Type:** NVARCHAR(ANY)

- `inTable` 
  the input table
  
  **Type:** NVARCHAR(256)

- `type` 
  the kind of distribution(s) to use:
  
  - n - normal (DNORM)
  - c - chi-square (DCHISQ)
  - t - t-student (DT)
  - u - uniform (DUNIF)
  - e - exponential (DEXP)
  - w - Weibull (DWEIBULL)
  
  Note that several distributions can be calculated simultaneously by concatenating the distribution abbreviations in the parameter value.
  
  **Type:** NVARCHAR(ANY)

- `incolumn` 
  the numeric input table column containing the x value(s)
  
  **Type:** NVARCHAR(128)

- `id` 
  the input table column identifying a unique instance id
  
  **Type:** NVARCHAR(128)

- `outTable`
the output table
Type: NVARCHAR(256)

- **df**
  the number of degrees of freedom for a student or chi-square distribution
  Type: INT8
  Default: <none>

- **min**
  the minimum for a uniform distribution
  Type: DOUBLE
  Default: <none>

- **max**
  the maximum for a uniform distribution
  Type: DOUBLE
  Default: <none>

- **scale**
  the scale for an exponential or weibull distribution
  Type: DOUBLE
  Default: <none>
  Min: 0.00000000001

- **shape**
  the shape for a weibull distribution
  Type: DOUBLE
  Default: <none>

- **mean**
  the mean value for a normal distribution
  Type: DOUBLE
  Default: <none>

- **variance**
  the variance for a normal distribution
  Type: NVARCHAR(ANY)
  Default: <none>
  Min: 0.00000000001

▲ Returns
INT8 the number of input table values for which a probability is calculated

**Details**
You have a table in which you have computed some summary statistics (chi-square tests etc.) and you want to know what is the probability that the statistics gets this value given that this statistics comes from a given
distribution (you may compare alternative distributions if you are not sure). Density is a kind of probability at the point \( x \) and its close neighborhood (the density should be multiplied with the width of the neighborhood to get the probability). Formally the density is the derivative of a cumulative distribution function of a probability distribution. It has to be positive.

The parameters for the distributions can contain literal values or expressions. The output table has following columns: id, \(<\text{incolumn}>\). Additionally, the output table contains a column for each distribution, named like the distribution density function, and containing the output of this function. It also contains a column per specific parameter for this distribution, named as the parameter, and containing the value calculated for it in each row.

**Examples**

```
CALL nza..DENSITY('intable=nza..iris, id=id, type="n",
  incolumn=petallength, outtable=iripetall, mean=3.75,
  variance=3.11');

SELECT * FROM iripetall ORDER BY id LIMIT 10;

CALL nza..DROP_TABLE('iripetall');
```

**DENSITY**

```
--------
150
(1 row)
```

```
   ID  |      DNORM      | PETALLENGTH | MEAN | VARIANCE
-----+----------------+-------------+------+----------
   1  | 0.16417862459158 |     1.4     | 3.75 |     3.11
   2  | 0.16417862459158 |     1.4     | 3.75 |     3.11
   3  | 0.1519854194291  |     1.3     | 3.75 |     3.11
   4  | 0.17678070212437 |     1.5     | 3.75 |     3.11
   5  | 0.16417862459158 |     1.4     | 3.75 |     3.11
   6  | 0.20299343912957 |     1.7     | 3.75 |     3.11
   7  | 0.16417862459158 |     1.4     | 3.75 |     3.11
   8  | 0.17678070212437 |     1.5     | 3.75 |     3.11
   9  | 0.16417862459158 |     1.4     | 3.75 |     3.11
  10  | 0.17678070212437 |     1.5     | 3.75 |     3.11
```

(10 rows)
DROP_TABLE
----------
t
(1 row)

Related Functions
► category Analytics - Probability Distributions
► DCHISQ
► DEXP
► DNORM
► DT
► DUNIF
► DWEIBULL

DEXP - Probability density function for Exponential Distribution

Given a degree of freedom, this function returns the probability density that a variable following the Exponential distribution takes a value equal to x

Usage
The DEXP function has the following syntax:

► DEXP(DOUBLE x, DOUBLE scale)

▲ Parameters
► x
  the value at which to compute
  Type: DOUBLE
  Min: 0.00000000001
► scale
  the exponential scale
  Type: DOUBLE
  Min: 0.00000000001

▲ Returns
  DOUBLE the probability density at point x

Details
DEXP(x, scale) = exp(-x/scale)/scale for all positive x values.
If x is negative, DEXP(x, scale) is null for all scale values.
If scale is 0 or less, DEXP(x, scale) is null for all x values.
Examples

```sql
SELECT nza.DEXP(2, 4);

DEXP
------------------
0.15163266492816
(1 row)
```

Related Functions

- category Analytics - Probability Distributions
- PEXP
- PEXP_H
- QEXP
- QEXP_H

DF - Probability density function for Fisher Distribution

Given the degrees of freedom of the nominator and the denominator, this function returns the probability density that a variable following the Fisher distribution takes a value equal to x.

Usage

The DF function has the following syntax:

```sql
DF(DOUBLE x, INT8 dfNomin, INT8 dfDenom)
```

▲ Parameters

- **x**
  the value at which to compute
  Type: DOUBLE
  Min: 0.00000000001

- **dfNomin**
  the number of degrees of freedom of the nominator
  Type: INT8
  Min: 1

- **dfDenom**
  the number of degrees of freedom of the denominator
  Type: INT8
DFISK - Probability density function for Fisk (or log-logistic) Distribution

Given the median and the shape, this function returns the probability density that a variable following the Fisk distribution takes a value equal to x

Usage

The DFISK function has the following syntax:

```
DFISK(DOUBLE x, DOUBLE median, DOUBLE shape)
```

▲ Parameters

▲ x
the value at which to compute
Type: DOUBLE
Min: 0.000000001

▲ median
the median value of the distribution (also called scale)
Type: DOUBLE
Min: 0.000000001

**shape**
the shape of the distribution
Type: DOUBLE
Min: 0.000000001

Returns
DOUBLE the probability density at point x

Details
DFISK(x,median,shape) = (shape/median)*(x/median)^(shape-1) / (1+(x/median)^shape)^2 for all positive x, median and shape values.
If x, median or shape is 0 or less, PFISK(x,median,shape) is null.

Examples

```
SELECT nza..DFISK(2.43,5,8);
   DFISK
         -------------------
         0.010182975576092
         (1 row)

SELECT nza..DFISK(2.43,2,3);
   DFISK
         -------------------
         0.28373389001712
         (1 row)

SELECT (nza..PFISK(0.000001+2.43,2,3) - nza..PFISK(2.43,2,3))/0.000001-nza..DFISK(2.43,2,3) diff;
   DIFF
         ----------------------
         -1.0810318662458e-07
```
DGAMMA - Probability density function for Gamma distribution

Given a shape and an inverted scale, this function returns the probability density that a variable following the Gamma distribution takes a value equal to x.

Usage

The DGAMMA function has the following syntax:

```
DGAMMA(DOUBLE x, DOUBLE shape, DOUBLE scaleInv)
```

- **Parameters**
  - **x**
    - the value at which to compute
    - Type: DOUBLE
  - **shape**
    - the shape of the distribution
    - Type: DOUBLE
    - Min: 0.00000000001
  - **scaleInv**
    - the inverted scale of the distribution
    - Type: DOUBLE
    - Min: 0.00000000001

- **Returns**
  - DOUBLE the probability density at the point x

Details

If shape or scaleInv is 0 or less, DGAMMA(x,shape, scaleInv) is null for all x values.

Examples

```
SELECT nza..DGAMMA(0, 3, 0.4), nza..DGAMMA(1, 3, 0.4), nza..DGAMMA(2, 3, 0.4), nza..DGAMMA(3, 3, 0.4), nza..DGAMMA(4, 3, 0.4);
```
Related Functions

- category Analytics - Probability Distributions
- PGAMMA
- PGAMMA_H
- QGAMMA
- QGAMMA_H

DGEOM - Density of the Geometric Distribution

Given the success probability \( p \), this function returns the probability that we get \( x \) failures before the first success in a series of Bernoulli trials.

Usage

The DGEOM function has the following syntax:

\[
\text{DGEOM(INT8} \ x, \ \text{DOUBLE} \ p)\]

- Parameters
  - \( x \)
    - the value at which to compute
    - Type: INT8
  - \( p \)
    - the success probability
    - Type: DOUBLE
    - Min: 0.0000000001
    - Max: 0.9999999999

- Returns
  - DOUBLE the probability density at the point \( x \)
Details
DGEOM(x,p) = p*(1-p)^x for all x values and for p between 0 and 1.
If p is not between 0 and 1, DGEOM(x,p) is null for all x values.

Examples

```sql
SELECT nza..DGEOM(0,0.4), nza..DGEOM(1,0.4), nza..DGEOM(2,0.4),
    nza..DGEOM(3,0.4);
```

| DGEOM | DGEOM | DGEOM | DGEOM |
|-------+-------+-------+--------|
| 0.4   | 0.24   | 0.144 | 0.0864 |

(1 row)

Related Functions
► category Analytics - Probability Distributions
► PGEOM
► PGEOM_H
► QGEOM
► QGEOM_H

DHYPHER - Density of the Hypergeometric Distribution

Given the number of white and black balls in the urn and the number of trials, this function returns the probability that we draw x white balls from N trials from an urn without replacement.

Usage

The DHYPHER function has the following syntax:

```
DHYPER(BIGINT x, BIGINT wu, BIGINT bu, BIGINT N)
```

Parameters

► x
the number of white balls drawn without replacement from an urn which contains both black and white balls
Type: BIGINT

► wu
the number of white balls in the urn
Type: BIGINT
Min: 0

► bu
the number of black balls in the urn
Type: BIGINT
Min: 0

► N
the number of balls drawn from the urn without replacement
Type: BIGINT
Min: 1

▲ Returns
DOUBLE the probability density at the point x

Details
In an urn containing $wu$ white balls and $bu$ black balls, we draw $N$ balls without replacement. Drawing a white ball is a success (1), a black ball is a failure (0). The Hypergeometric distribution calculates the probability that we get exactly $x$ successes in the $N$ trials.

If $wu$ or $bu$ is negative, DHYPER($x$, $wu$, $bu$, $N$) is null for all $x$ and $N$ values.
If $N$ is 0 or less, DHYPER($x$, $wu$, $bu$, $N$) is null for all $x$, $wu$ and $bu$ values.

Examples

```sql
SELECT nza..DHYPER(1,1,1,1);
```
```
DHYPER
-------
 0.5
(1 row)
```

```sql
SELECT nza..DHYPER(1,2,3,4);
```
```
DHYPER
-------
 0.4
(1 row)
```

```sql
SELECT nza..DHYPER(1,4,1,2);
```
```
DHYPER
-------
```

Related Functions

- category Analytics - Probability Distributions
- PHYPER
- PHYPER_H
- QHYPER
- QHYPER_H

DIVCLUSTER - Build a Hierarchical Clustering model

This stored procedure builds a Hierarchical Clustering model using a divisive method (top-down). The K-means algorithm is used recursively. The hierarchy of clusters is represented in a binary tree structure (each parent node has exactly 2 children node). The leaves of the cluster tree are identified by negative numbers. The divisive clustering algorithm may return different results for the same dataset and the same random generator seed when you use different input data distribution or a different number of dataslices. This is due to the behavior of the random number generator, which generates random sequences depending on the number of dataslices and data distribution. The algorithm returns the same model when the same machine, the same input data distribution, and the same random seed is used.

Usage

The DIVCLUSTER stored procedure has the following syntax:
DIVCLUSTER(NVARCHAR(ANY) paramString)

▲ Parameters

► paramString
comma-separated list of <parameter>=<value> entries with parameters below
Type: NVARCHAR(ANY)

► model
the name of the Hierarchical Clustering model to build
Type: NVARCHAR(ANY)

► intable
the input table
Type: NVARCHAR(256)

► outtable
the output table where clusters are assigned to each input table record
Type: NVARCHAR(ANY)

► id
the input table column identifying a unique instance id
Type: NVARCHAR(128)

► target
the input table column representing a class or a value to predict, this column is ignored by the Hierarchical Clustering algorithm.
Type: NVARCHAR(128)
Default: <none>

► distance
the distance function. Allowed values are: euclidean, manhatthan, canberra, maximum.
Type: NVARCHAR(ANY)
Default: euclidean

► maxiter
the maximum number of iterations to perform in the base K-means Clustering algorithm
Type: INTEGER
Default: 5
Min: 1
Max: 1000

► minsplit
the minimum number of instances per cluster that can be split
Type: INTEGER
Default: 5
Min: 2

► **maxdepth**
the maximum number of cluster levels (including leaves)
Type: INTEGER
Default: 3
Min: 1
Max: 60

► **randseed**
the random generator seed
Type: INTEGER
Default: 12345

► **incolmn**
the input table columns with special properties, separated by a semi-colon (;).
Each column is followed by one or several of the following properties:
- its type: ':nom' (for nominal), ':cont' (for continuous). Per default, all numerical types are continuous, other types are nominal.
- its role: ':id', ':target', ':input', ':ignore'.
  (Remark: ':objweight' is unsupported, i.e. ':objweight' same as ':ignore').
  (Remark: ':colweight(<wgt>)' is unsupported, i.e. ':colweight(<wgt>)' same as ':colweight(1)' same as ':input').
If the parameter is undefined, all columns of the input table have default properties.
Type: NVARCHAR(ANY)
Default: <none>

► **coldeftype**
default type of the input table columns. Allowed values are 'nom' and 'cont'.
If the parameter is undefined, all numeric columns are considered continuous, other columns nominal.
Type: NVARCHAR(ANY)
Default: <none>

► **coldefrole**
default role of the input table columns. Allowed values are 'input' and 'ignore'.
If the parameter is undefined, all columns are considered 'input' columns.
Type: NVARCHAR(ANY)
Default: <none>

► **colPropertiesTable**
the input table where column properties for the input table columns are stored. The format of this table is the output format of stored procedure nza..COLUMN_PROPERTIES().

If the parameter is undefined, the input table column properties will be detected automatically.

(Remark: colPropertiesTable with "COLROLE" column with value 'objweight' is unsupported, i.e. same as 'ignore')

(Remark: colPropertiesTable with "COLWEIGHT" column with value '<wgt>' is unsupported, i.e. same as '1')

Type: NVARCHAR(256)
Default: <none>

▲ Returns
BIGINT the number of created clusters

Details
This stored procedure builds a Hierarchical Clustering model. The model is saved to the database in a set of tables and registered in the database model metadata. Use the Model Management functions to further manipulate the model or access the model tables.

The hierarchy of clusters is build by recursively partitioning data into two separate clusters using the base K-means Clustering algorithm. The top-down Hierarchical Clustering algorithm is used with the following features:
- base K-means algorithm with k=2 (binary splits),
- no missing value handling,
- stop criteria satisfied when the number of instances is less than <minsplit>,
- stop criteria satisfied when the cluster tree depth reaches <maxdepth>.

The output table <outtable> is created with following columns: id, cluster_id, distance. The id column matches the <id> column of the input table. Each input table record is associated with a cluster, where the distance from the record to the cluster center is the smallest. The cluster ID and the distance to the cluster center are given in the columns cluster_id and distance.

Examples

CALL nza..DIVCLUSTER('model=iris_mdl, intable=nza..iris, outtable=iris_out, id=id, distance=euclidean, maxiter=5, minsplit=5, maxdepth=3, randseed=12345');

CALL nza..DROP_MODEL('model=iris_mdl');

CALL nza..DROP_TABLE('iris_out');

CALL nza..DIVCLUSTER('model=adult_mdl, intable=nza..adult, outtable=adult_out, id=id, distance=euclidean, maxiter=5, minsplit=5, maxdepth=3,
randseed=12345');
CALL nza..DROP_MODEL('model=adult_mdl');
CALL nza..DROP_TABLE('adult_out');
CALL nza..DIVCLUSTER('model=csi_mdl, intable=nza..censusincome, outtable=csi_out, id=id, distance=euclidean, maxiter=5, minsplit=5, maxdepth=3, randseed=12345');
CALL nza..DROP_MODEL('model=csi_mdl');
CALL nza..DROP_TABLE('csi_out');

DIVCLUSTER
------------
13
(1 row)

DROP_MODEL
------------
t
(1 row)

DROP_TABLE
------------
t
(1 row)

DIVCLUSTER
------------
15
(1 row)

DROP_MODEL
------------
t
(1 row)

DROP_TABLE
Related Functions

- category Analytics - Clustering
- PREDICT_DIVCLUSTER
- LIST_MODELS

DLNORM - Density of Galton (or LogNormal) Distribution

Given a logarithmic scale median and shape, this function returns the probability density that a variable following the LogNormal distribution takes a value equal to x

Usage

The DLNORM function has the following syntax:

- DLNORM(DOUBLE x, DOUBLE medlog, DOUBLE sdlog)

  ▲ Parameters
  - x
the value at which to compute
Type: DOUBLE
Min: 0.000000001
► medlog
the median value on the logarithmic scale
Type: DOUBLE
► sdlog
the shape on the logarithmic scale
Type: DOUBLE
Min: 0.000000001
▲ Returns
DOUBLE the probability density at point x

Details
If x or sdlog is 0 or less, DLNORM(x,medlog,sdlog) is null for all medlog values.

Examples

```sql
SELECT nza..DLNORM(2.43,0,1);
  DLNORM
  ---------------
  0.11069183133537
(1 row)
```

```sql
SELECT nza..DLNORM(2.43,2,3);
  DLNORM
  ---------------
  0.051090719919889
(1 row)
```

Related Functions
► category Analytics - Probability Distributions
► PLNORM
► PLNORM_H
► QLNORM
► QLNORM_H
DLOGIS - Probability density function for Logistic Distribution

Given the mean and the scale, this function returns the probability density that a variable following the Logistic distribution takes a value equal to x

Usage
The DLOGIS function has the following syntax:

```
DLOGIS(DOUBLE x, DOUBLE mean, DOUBLE scale)
```

Parameters
- **x**: the value at which to compute
  - Type: DOUBLE
- **mean**: the mean value of the distribution
  - Type: DOUBLE
- **scale**: the scale of the distribution
  - Type: DOUBLE
  - Min: 0.000000001

Returns
- **DOUBLE**: the probability density at point x

Details
If scale is 0 or less, DLOGIS(x,mean,scale) is null for all x and mean values.

Examples

```
SELECT nza..DLOGIS(2.43,0,1);
DLOGIS
-------------------
0.074366478038637
(1 row)

SELECT nza..DLOGIS(2.43,2,3);
DLOGIS
```
DMWW - Density of the Mann-Whitney-Wilcoxon Distribution

Given the number of items separated in two sets, the minimum number of items in set 1, this function returns the probability that a variable following the MWW distribution takes a value equal to MWWmin.

Usage

The DMWW function has the following syntax:

```plaintext
DMWW(INT8 MWWmin, INT8 COUNTmin, INT8 NoItems)
```

**Parameters**

- **MWWmin**
  - the value at which to compute
  - Type: INT8
  - Min: 0
  - Max: COUNTmin*(NoItems+NoItems-COUNTmin+1)/2;

- **COUNTmin**
  - the number of items in set 1
  - Type: INT8
  - Min: 1
  - Max: NoItems-1

- **NoItems**
  - the total number of items
  - Type: INT8
  - Min: 1

**Returns**

DOUBLE the probability density at point MWWmin.
Details
DMWW(MWWmin,COUNTmin,Noltems)=PMWW(MWWmin,COUNTmin,Noltems) - 
PMWW(MWWmin-1,COUNTmin,Noltems) for all MWWmin, COUNTmin and Noltems values in 
their respective validity interval.
If MWWmin, COUNTmin or Noltems are not in their validity interval, DMWW(MWWmin,COUNT-
min,Noltems) returns null.

Examples

SELECT nza..DMWW(12,4,18);
DMWW
---------------------
  0.00065359477124183
(1 row)

SELECT nza..DMWW(30,4,9);
DMWW
---------------------
  0.0079365079365079
(1 row)

Related Functions
► category Analytics - Probability Distributions
► PMWW
► PMWW_H
► QMWW
► QMWW_H

DNBINOM - Density of the Negative Binomial Distribution
Given the success probability p and the number of successes needed, this function returns the 
probability that we get x failures before getting s successes in a serie of Bernoulli trials.

Usage
The DNBINOM function has the following syntax:
DNBINOM(INT8 x, INT8 s, DOUBLE p)

Parameters

- **x**
  - the value at which to compute
  - Type: INT8

- **s**
  - the number of successes needed to stop the series of Bernoulli trials
  - Type: INT8
  - Min: 1

- **p**
  - the success probability
  - Type: DOUBLE
  - Min: 0.00000000001
  - Max: 1-0.00000000001

Returns

DOUBLE the probability density at the point x

Details

DNBINOM(x,s,p) returns the probability to get exactly x failures while getting s successes during a series of Bernoulli trials with success probability p.

DNBINOM(x,s,p) = PNBINOM(x,s,p) - PNBINOM(x-1,s,p) for all positive x and s values, and for p between 0 and 1.

If s is 0 or less, DNBINOM(x,s,p) is null for all x and p values.

If p is not between 0 and 1, DNBINOM(x,s,p) is null for all x and s values.

Examples

```sql
SELECT nza..DNBINOM(0,3,0.4), nza..DNBINOM(1,3,0.4),
   nza..DNBINOM(2,3,0.4), nza..DNBINOM(3,3,0.4),
   nza..DNBINOM(4,3,0.4);

DNBINOM | DNBINOM | DNBINOM | DNBINOM | DNBINOM
---------+---------+---------+---------+---------
 0.064   | 0.1152  | 0.13824 | 0.13824 | 0.124416
(1 row)
```

```sql
SELECT nza..DNBINOM(4,3,0.4), nza..DNBINOM(44,30,0.4),
   nza..DNBINOM(444,300,0.4), nza..DNBINOM(4444,3000,0.4),
   nza..DNBINOM(44444,30000,0.4);
```
### Related Functions

- category Analytics - Probability Distributions
- DBINOM
- PNBINOM
- PNBINOM_H
- QNBINOM
- QNBINOM_H

### DNORM - Density of the Standard Normal Distribution

This function returns the probability that a variable following the Standard Normal distribution takes a value equal to x.

**Usage**

The DNORM function has the following syntax:

```
DNORM(DOUBLE x)
```

**Parameters**

- **x**
  - the value at which to compute
  - Type: DOUBLE

**Returns**

- DOUBLE the probability density at point x

**Details**

The density function is mathematically a derivative of the cumulative distribution function. It is positive in the whole domain.

\[ DNORM(x) = \frac{\exp(-x^2/2)}{\sqrt{2\pi}} \]

for all x values.

---

<table>
<thead>
<tr>
<th>DNBINOM</th>
<th>DNBINOM</th>
<th>DNBINOM</th>
<th>DNBINOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.124416</td>
<td>0.037987</td>
<td>0.011826</td>
<td></td>
</tr>
<tr>
<td>0.003302</td>
<td>0.000301</td>
<td>0.000301</td>
<td>0.000301</td>
</tr>
</tbody>
</table>

(1 row)
Examples

```sql
SELECT nza..DNORM(2.43);
```

```
DNORM
-------------------
 0.020829426985092
-------------------
(1 row)
```

Related Functions

- category Analytics - Probability Distributions
- DNORM3P
- PNORM
- PNORM_H
- QNORM
- QNORM_H

**DNORM3P - Density of the Normal Distribution**

Given a mean value and a standard deviation, this function returns the probability that a variable following the Normal distribution takes a value equal to \( x \).

**Usage**

The `DNORM3P` function has the following syntax:

```
DNORM3P(DOUBLE x, DOUBLE mean, DOUBLE stdev)
```

▲ **Parameters**

- **x**
  - the value at which to compute
  - Type: DOUBLE

- **mean**
  - the mean value of the distribution
  - Type: DOUBLE

- **stdev**
  - the standard deviation of the distribution
  - Type: DOUBLE
  - Min: 0.000000000001

▲ **Returns**

- DOUBLE the probability density at point \( x \)
Details

The density function is mathematically a derivative of the cumulative distribution function. It is positive in the whole domain.

\[ \text{DNORM3P}(x, \text{mean}, \text{stdev}) = \exp\left(-\frac{(x - \text{mean})^2}{2\text{stdev}^2}\right) \sqrt{2 \pi \text{stdev}^2} \] for all \( x \) and mean values and for all strictly positive \( \text{stdev} \) values.

If \( \text{stdev} \) is 0 or less, \( \text{DNORM3P}(x, \text{mean}, \text{stdev}) \) is null for all \( x \) and mean values.

Examples

```sql
SELECT nza..DNORM3P(2.43, 0, 1);
DNORM3P
-------------------
0.020829426985092
(1 row)
```

```sql
SELECT nza..DNORM3P(2.43+5, 5, 1);
DNORM3P
-------------------
0.020829426985092
(1 row)
```

```sql
SELECT nza..DNORM3P(2.43*2, 0, 2);
DNORM3P
-------------------
0.010414713492546
(1 row)
```

```sql
SELECT nza..DNORM3P(2.43*2-7, -7, 2);
DNORM3P
-------------------
0.010414713492546
(1 row)
```
Related Functions

- category Analytics - Probability Distributions
- DNORM
- PNORM3P
- QNORM3P

DPOIS - Density of the Poisson Distribution

Given the mean number of success within a fixed time interval, this function returns the probability that a variable following the Poisson distribution takes a value equal to x. The value x is the number of successes expected within this time interval.

Usage

The DPOIS function has the following syntax:

DPOIS(INT8 x, DOUBLE lambda)

Parameters

- x
  the value at which to compute
  Type: INT8

- lambda
  the mean number of successes within a fixed time interval
  Type: DOUBLE
  Min: 0.00000000001

Returns

DOUBLE the probability density at the point x

Details

If lambda is 0 or less, PPOIS(x,lambda) is null for all x values.

Examples

```
SELECT nza..DPOIS(0,0.4), nza..DPOIS(1,0.4), nza..DPOIS(2,0.4), nza..DPOIS(3,0.4);
```

<table>
<thead>
<tr>
<th>DPOIS</th>
<th>DPOIS</th>
<th>DPOIS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.67032004603564</td>
<td>0.26812801841426</td>
<td>0.053625603682851</td>
</tr>
</tbody>
</table>
DROP_ALL_MODELS - Drop selected or all analytics models

This stored procedure drops all models for which the user has the DROP privilege. A WHERE clause can be used to drop only selected models.

Usage

The DROP_ALL_MODELS stored procedure has the following syntax:

- **DROP_ALL_MODELS(NVARCHAR(ANY))**

  ▲ Parameters
  - **paramString**
    - comma-separated list of \(<parameter>=<value>\) entries with parameters below
    - Type: NVARCHAR(ANY)
  - **where**
    - a WHERE clause to limit the set of models to be dropped. All columns from view V_NZA_MODELS can be used in the WHERE clause.
    - Type: NVARCHAR(ANY)

  ▲ Returns
  - BOOLEAN true if all models for which the user has the DROP privilege could be dropped successfully. False, if there were errors.

Examples

```sql
CALL nza..ARULE('intable=nza..retail, model=mbamodel, supporttype=percent, support=5, lvl=0, maxsetsize=5, confidence=0.5');

CALL nza..DROP_ALL_MODELS('where=upper(MODELNAME) LIKE ''MBA%'');

NOTICE:

RUNNING FPGrowth algorithm:
```
DATASET : "NZA"."RETAIL"
Transaction column : "TID"
Item column : "ITEM"
Group by : <none>
Minimum support : 5 %
Minimum confidence: 0.5
Max frequent itemset size : 5
Level of conditional dbs : 0
Result tables prefix : "NZA_META_MBAMODEL"

ARULE
-------
14
(1 row)

NOTICE: Dropped: MBAMODEL

DROP_ALL_MODELS
-----------------

(1 row)

Related Functions
► category Analytics - Model Management
► LIST_MODELS

DROP_MODEL - Drop an analytics model
This stored procedure drops the given model. All managed tables of this model are also dropped.

Usage
The DROP_MODEL stored procedure has the following syntax:

► DROP_MODEL(NVARCHAR(ANY))
  ▲ Parameters
paramString
comma-separated list of <parameter>=<value> entries with parameters below
Type: NVARCHAR(ANY)

model
the model to be dropped
Type: NVARCHAR(64)

Returns
BOOLEAN true if the model was dropped successfully. False, if there were errors.

Examples

CALL nza..ARULE('intable=nza..retail, model=mbamodel, supporttype=percent, support=5, lvl=0, maxsetsize=5, confidence=0.5');

CALL nza..DROP_MODEL('model=mbamodel');

NOTICE:
RUNNING FPGrowth algorithm:
DATASET : "NZA".."RETAIL"
Transaction column : "TID"
Item column : "ITEM"
Group by : <none>
Minimum support : 5 %
Minimum confidence: 0.5
Max frequent itemset size : 5
Level of conditional dbs : 0
Result tables prefix : "NZA_META_MBAMODEL"

ARULE
-------
14
(1 row)

NOTICE: Dropped: MBAMODEL

DROP_MODEL
------------
Related Functions

- category Analytics - Model Management
- LIST_MODELS

DROP_SUMMARY1000 - Drop tables created by SUMMARY1000

This stored procedure drops all the tables created in the stored procedure SUMMARY1000.

Usage

The DROP_SUMMARY1000 stored procedure has the following syntax:

```
DROP_SUMMARY1000(NVARCHAR(ANY) paramString)
```

Parameters

- `paramString`
  
  comma-separated list of `<parameter>=<value>` entries with parameters below
  
  Type: NVARCHAR(ANY)

- `intable`
  
  the table name specified in parameter outtable of SUMMARY1000
  
  Type: NVARCHAR(ANY)

Returns

NVARCHAR(ANY) an informative string

Details

This stored procedure drops all output tables created by the stored procedure SUMMARY1000. The stored procedure SUMMARY1000 is called with a parameter outtable=<tbname>. It creates the table <tbname> and, depending on the type of its input table columns, one or several of the following tables: <tbname>_char, <tbname>_num, <tbname>_date, <tbname>_interval, <tbname>_time and <tbname>_timestamp. Calling DROP_SUMMARY('intable=<tbname>') drops all the existing tables having these names.

Examples

```
CALL nza..SUMMARY1000('intable=nza..iris, outtable=result7');
CALL nza..DROP_SUMMARY1000('intable=result7');
SUMMARY1000
-------------
Done
```
Related Functions

- category Analytics - Statistics
- SUMMARY1000

DT - Density of the T-student Distribution

Given a degree of freedom, this function returns the probability density that a variable following the T-student distribution takes a value equal to x.

Usage

The DT function has the following syntax:

\[
\text{DT}(\text{DOUBLE } x, \text{INT8 } df)
\]

Parameters

- **x**
  - the value at which to compute
  - Type: DOUBLE

- **df**
  - the number of degrees of freedom
  - Type: INT8
  - Min: 1

Returns

DOUBLE the probability density at point x

Details

DT(x,df) is a derivative of PT(x,df) on x. It is positive on the whole domain. If df is 0 or less, DT(x,df) is null for all x values.

Examples

\[
\text{SELECT nza.DT(-2.821, 9) ;}
\]
DUNIF - Density of the Uniform Distribution

Given a minimum and a maximum for the uniform interval, this function returns the probability that a variable following the Uniform distribution takes a value equal to x

Usage
The DUNIF function has the following syntax:

DUNIF(DOUBLE x, DOUBLE minimum, DOUBLE maximum)

Parameters
- x
  the value at which to compute
  Type: DOUBLE
  Min: 0.00000000001
- minimum
  left border of support, x must be above this
  Type: DOUBLE
- maximum
  right border of support, x must be below this
  Type: DOUBLE

Returns
DOUBLE the probability density at point x

Details
DUNIF(x, minimum, maximum)=1/(maximum-minimum) for all x, minimum and maximum values where x is between minimum and maximum.

If minimum is not smaller than maximum, or if x is not between minimum and maximum, DUNIF(x, minimum, maximum) is null.
Examples

```
SELECT nza..DUNIF(2,-3.0,11.3);
```

```
DUNIF
0.06993006993007
```

Related Functions

- category Analytics - Probability Distributions
- PUNIF
- PUNIF_H
- QUNIF
- QUNIF_H

DWALD - Density of the Wald distribution

Given a location and a shape, this function returns the probability that a variable following the Wald distribution takes a value equal to \( x \)

**Usage**

The `DWALD` function has the following syntax:

```
DWALD(DOUBLE x, DOUBLE location, DOUBLE shape)
```

**Parameters**

- \( x \)
  - the value at which to compute
  - Type: DOUBLE

- location
  - the mean of the distribution
  - Type: DOUBLE
  - Min: 0.0000000001

- shape
  - the shape of the distribution
  - Type: DOUBLE
  - Min: 0.0000000001
Returns
DOUBLE the probability density at point x

Details
If location or shape is 0 or less, DWALD(x,location,shape) is null for all x values.

Examples

```sql
SELECT nza..DWALD(1,1,1), nza..DWALD(2,1,1), nza..DWALD(2,2,1), nza..DWALD(2,2,2);
```

<table>
<thead>
<tr>
<th>DWALD</th>
<th>DWALD</th>
<th>DWALD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.39894228040143</td>
<td>0.10984782236693</td>
<td>0.14104739588694</td>
</tr>
<tr>
<td>0.19947114020072</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(1 row)

Related Functions
- category Analytics - Probability Distributions
  - PWALD
  - PWALD_H
  - QWALD
  - QWALD_H

DWEIBULL - Density of the Weibull (or Rosin-Rammer) Distribution

Given a scale and a shape, this function returns the probability that a variable following the Weibull distribution takes a value equal to x

Usage
The DWEIBULL function has the following syntax:

```sql
DWEIBULL(DOUBLE x, DOUBLE scale, DOUBLE kShape)
```

Parameters
- `x`
  - the value at which to compute
  - Type: DOUBLE
  - Min: 0
- `scale`
  - the scale of the distribution
Type: DOUBLE
Min: 0.00000000001

▲ kShape
the shape of the distribution
Type: DOUBLE
Min: 0.00000000001

▲ Returns
DOUBLE the probability density at point x

Details
If x is less than 0, or if scale or shape are 0 or less, DWEIBULL(x, scale, kShape) is null.

Examples
SELECT nza..DWEIBULL(2, 4, 1);

DWEIBULL
------------------
0.15163266492816
(1 row)

Related Functions
► category Analytics - Probability Distributions
► PWEIBULL
► PWEIBULL_H
► QWEIBULL
► QWEIBULL_H

DWILCOX - Density of the Wilcoxon Distribution
Given the number of items, this function returns the probability that a variable following the Wilcoxon distribution takes a value equal to Wmin

Usage
The DWILCOX function has the following syntax:

▲ DWILCOX(INT8 Wmin, INT8 NoItems)
▲ Parameters
► Wmin
the value at which to compute
Type: INT8
Min: 0
Max: NoItems*(NoItems+1)/2

► **NoItems**
the number of items
Type: INT8
Min: 1
Max: 15

▲ Returns
DOUBLE the probability density at point Wmin

**Details**

\[ DWILCOX(W_{\text{min}}, \text{NoItems}) = PWILCOX(W_{\text{min}}, \text{NoItems}) - PWILCOX(W_{\text{min}}-1, \text{NoItems}) \]
for all \( W_{\text{min}} \) and \( \text{NoItems} \) values in their respective validity interval.
If \( W_{\text{min}} \) or \( \text{NoItems} \) are not in their validity interval, \( DWILCOX(W_{\text{min}}, \text{NoItems}) \) returns null.

**Examples**

```sql
SELECT nza..DWILCOX(42,15);
```

```
DWILCOX
------------------
 0.01373291015625
(1 row)
```

**Related Functions**

- category Analytics - Probability Distributions
- PWILCOX
- PWILCOX_H
- QWILCOX
- QWILCOX_H

**EFDISC - Discretization bins of equal frequency**

This stored procedure calculates the limits for discretization bins of equal frequency on numeric columns.
Each bin contains approximatively an equal number of data records.

**Usage**

The EFDISC stored procedure has the following syntax:
EFDISC(NVARCHAR(ANY) paramString)
▲ Parameters
► paramString
comma-separated list of <parameter>=<value> entries with parameters below
Type: NVARCHAR(ANY)
► intable
the input table or view
Type: NVARCHAR(ANY)
► outtable
the output table to store the discretization bin limits into
Type: NVARCHAR(ANY)
► incolumn
the input table columns to consider, separated by a semi-colon (;).
Each column name may be followed by a colon (:) and the number of discretization bins to be calculated for this column.
Type: NVARCHAR(ANY)
► bins
the default number of discretization bins to be calculated
Type: INTEGER
Default: 10
► binprec
the precision allowed for considering an even distribution of data records in the calculated discretization bins. The number of data records in each bin must be within \[iw-<\text{binprec})*iw,iw+<\text{binprec})*iw\] where iw is the size of the input table divided by the number of requested discretization bin limits.
Type: DOUBLE
Default: 0.1
Min: 0
Max: 1
▲ Returns
INTEGER the number of columns for which discretization bin limits are calculated

Details
This stored procedure calculates the limits for discretization bins of equal frequency on numeric columns. Each bin contains approximately an equal number of data records.

The discretization bin limits are written into the output table with following columns: colname, break. The column colname references the name of the input table columns, and the column
break contains a bin limit for this column.

When it is impossible to discretize with the specified number of bins, the stored procedure tries to calculate a fewer number of bins. If the data is skewed, it is even possible that all data records fall into a single bin: in this case, a single row is written for this column into the output table with a NULL value in the column break.

**Examples**

```sql
CALL nza..EFDISC('outtable=disc_bounds1, intable=nza..iris, incolumn=petallength:petalwidth:4, bins=3');
SELECT * FROM disc_bounds1 ORDER BY colname, break;
CALL nza..APPLY_DISC('outtable=disc_iris_discrete, intable=nza..iris, btable=disc_bounds1, replace=T');
SELECT petallength, count(*) FROM disc_iris_discrete GROUP BY petallength ORDER BY petallength;
SELECT petalwidth, count(*) FROM disc_iris_discrete GROUP BY petalwidth ORDER BY petalwidth;
CALL nza..DROP_TABLE('disc_bounds1');
CALL nza..DROP_TABLE('disc_iris_discrete');
```

**EFDISC**

-------

2

(1 row)

<table>
<thead>
<tr>
<th>COLNAME</th>
<th>BREAK</th>
</tr>
</thead>
<tbody>
<tr>
<td>PETALLENGTH</td>
<td>1.9</td>
</tr>
<tr>
<td>PETALLENGTH</td>
<td>4.9</td>
</tr>
<tr>
<td>PETALWIDTH</td>
<td>0.3</td>
</tr>
<tr>
<td>PETALWIDTH</td>
<td>1.3</td>
</tr>
<tr>
<td>PETALWIDTH</td>
<td>1.8</td>
</tr>
</tbody>
</table>

(5 rows)

**APPLY_DISC**

--------

2

(1 row)
<table>
<thead>
<tr>
<th>PETALLENGTH</th>
<th>COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>50</td>
</tr>
<tr>
<td>2</td>
<td>54</td>
</tr>
<tr>
<td>3</td>
<td>46</td>
</tr>
</tbody>
</table>

(3 rows)

<table>
<thead>
<tr>
<th>PETALWIDTH</th>
<th>COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>41</td>
</tr>
<tr>
<td>2</td>
<td>37</td>
</tr>
<tr>
<td>3</td>
<td>38</td>
</tr>
<tr>
<td>4</td>
<td>34</td>
</tr>
</tbody>
</table>

(4 rows)

DROP_TABLE
---------
t
(1 row)

DROP_TABLE
---------
t
(1 row)

CALL nza..EFDISC('outtable=disc_bounds2, intable=nza..iris, incolumn= petallength:6, binprec=0.001');

SELECT * FROM disc_bounds2 ORDER BY colname, break;

CALL nza..APPLY_DISC('outtable=disc_iris_discrete2,
intable=nza..iris, btable=disc_bounds2, replace=T');

SELECT petallength, count(*) FROM disc_iris_discrete2 GROUP BY petallength ORDER BY petallength;

CALL nza..DROP_TABLE('disc_bounds2');

CALL nza..DROP_TABLE('disc_iris_discrete2');

<table>
<thead>
<tr>
<th>COLNAME</th>
<th>BREAK</th>
</tr>
</thead>
<tbody>
<tr>
<td>PETALLENGTH</td>
<td>4.3</td>
</tr>
</tbody>
</table>

(A 1 row)

<table>
<thead>
<tr>
<th>PETALLENGTH</th>
<th>COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>75</td>
</tr>
<tr>
<td>2</td>
<td>75</td>
</tr>
</tbody>
</table>

(Two rows)

1

(A 1 row)

<table>
<thead>
<tr>
<th>DROP_TABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
</tr>
</tbody>
</table>

(A 1 row)

<table>
<thead>
<tr>
<th>DROP_TABLE</th>
</tr>
</thead>
</table>


t
(1 row)

Related Functions
► category Analytics - Discretization
► EMDISC
► EWDISC
► APPLY_DISC

EMDISC - Discretization bins of minimal entropy
This stored procedure calculates the limits for discretization bins of minimal entropy on numeric columns. Each bin contains approximatively an equal number of data records.

Usage
The EMDISC stored procedure has the following syntax:

► EMDISC(NVARCHAR(ANY) paramString)

▲ Parameters
► paramString
  comma-separated list of <parameter>=<value> entries with parameters below
  Type: NVARCHAR(ANY)
► intable
  the input table or view
  Type: NVARCHAR(ANY)
► outtable
  the output table to store the discretization bin limits into
  Type: NVARCHAR(ANY)
► icolumn
  the input table columns to consider, separated by a semi-colon (;)
  Type: NVARCHAR(ANY)
► target
  the input table column containing a class label
  Type: NVARCHAR(ANY)

▲ Returns
INTEGER the number of columns for which discretization bin limits are calculated
Details

This stored procedure calculates the limits for discretization bins on numeric columns that minimize the entropy of the input table column <target>.

The discretization bin limits are written into the output table with following columns: colname, break. The column colname references the name of the input table columns, and the column break contains a bin limit for this column.

Entropy Minimization Discretization was originally presented in [U. M. Fayyad, K. B. Irani Multi-interval discretization of continuous-valued attributes for classification learning]. In this method a set of sorted values is recursively partitioned selecting splits that minimize class information entropy. Following original notation, let T be the value of attribute A that partition set S of examples into two disjoint subsets S1 and S2. Let’s also assume that there are k classes C1, ..., Ck and let P(Ci,S) be the proportion of examples in S that are labeled with class Ci.

- The class entropy of a subset S is defined as: \( E(S) = -P(C1,S) \log(P(C1,S)) - P(C2,S) \log(P(C2,S)) - \ldots - P(Ck,S) \log(P(Ck,S)) \)

- The class information entropy of the partition induced by split T on attribute A is given with the following equation: \( E(A,T;S) = (|S1|/|S|)E(S1)+(|S2|/|S|)E(S2) \) Amongst all candidate split points TA is selected for which \( E(A,T;S) \) is minimal.

The stopping criterion problem is solved using so called Minimum Description Length Principle Criterion.

Examples

```sql
CALL nza..EMDISC('outtable=disc_bounds1, intable=nza..disc_iris, incolumn=petal_length;petal_width, target=species');
SELECT * FROM disc_bounds1 ORDER BY colname, break;
CALL nza..APPLY_DISC('outtable=disc_iris_discrete, intable=nza..disc_iris, btable=disc_bounds1, replace=T');
SELECT petal_length, count(*) FROM disc_iris_discrete GROUP BY petal_length ORDER BY petal_length;
SELECT petal_width, count(*) FROM disc_iris_discrete GROUP BY petal_width ORDER BY petal_width;
CALL nza..DROP_TABLE('disc_bounds1');
CALL nza..DROP_TABLE('disc_iris_discrete');
```

EMDISC
--------

2

(1 row)

<table>
<thead>
<tr>
<th>COLNAME</th>
<th>BREAK</th>
</tr>
</thead>
<tbody>
<tr>
<td>PETAL_LENGTH</td>
<td>1.9</td>
</tr>
</tbody>
</table>
**IBM Netezza In-Database Analytics Reference Guide**

<table>
<thead>
<tr>
<th>PETAL_LENGTH</th>
<th>4.7</th>
</tr>
</thead>
<tbody>
<tr>
<td>PETAL_WIDTH</td>
<td>0.6</td>
</tr>
<tr>
<td>PETAL_WIDTH</td>
<td>1.7</td>
</tr>
</tbody>
</table>

(4 rows)

**APPLY_DISC**

```
-------------------
     2
(1 row)
```

**PETAL_LENGTH | COUNT**

```
-------------+-------
   1       |   50  
   2       |   45  
   3       |   55  
(3 rows)
```

**PETAL_WIDTH | COUNT**

```
-------------+-------
   1       |   50  
   2       |   54  
   3       |   46  
(3 rows)
```

**DROP_TABLE**

```
---------
t
(1 row)
```

**DROP_TABLE**

```
---------
```
Related Functions

- category Analytics - Discretization
- EFDISC
- EWDISC
- APPLY_DISC

ENTROPY - Univariate Entropy

This stored procedure calculates the entropy of a column

Usage

The ENTROPY stored procedure has the following syntax:

```
ENTROPY(NVARCHAR(ANY) paramString)
```

▲ Parameters

- **paramString**
  
  comma-separated list of \( \text{parameter} = \text{value} \) entries with parameters below
  
  Type: NVARCHAR(ANY)

- **intable**
  
  the input table
  
  Type: NVARCHAR(256)

- **incolumn**
  
  the input table column
  
  Type: NVARCHAR(128)

- **by**
  
  the input table column which splits the data into groups for which the operation is to be performed. If specified, an output table must be indicated.
  
  Type: NVARCHAR(128)
  
  Default: <none>

- **outtable**
  
  the output table to write the entropy into. This parameter must be specified if parameter by is specified.
  
  Type: NVARCHAR(256)
  
  Default: <none>

▲ Returns
DOUBLE the entropy of the input table column, or the number of groups in the output table for which entropy has been calculated

Details

This stored procedure calculates the entropy of an input column, either in the whole input table or within the groups defined in the column specified by parameter <by>. Univariate Entropy is a measure saying how evenly the cases are distributed for the values of the input column. It takes a positive value, 0 means that the column contains only one value.

If parameter 'by' is specified, an output table must be specified. The output table is created with following columns: over_<by>, entropy. The entropy of the input column is given for each group of the <by> column.

Examples

CALL nza..ENTROPY('intable=nza..adult, incolumn=WORKCLASS');

ENTROPY
-----------------
1.358645281597
(1 row)

CALL nza..ENTROPY('intable=nza..adult, incolumn=WORKCLASS, by=EDUCATION, outtable=entropy_adult_4');
SELECT * FROM entropy_adult_4 order by over_education;
CALL nza..DROP_TABLE('entropy_adult_4');

ENTROPY
-------
16
(1 row)

ENTROPY | OVER_EDUCATION
---------+------------------
0.89894981922821 | 10th
0.74279033472579 | 11th
0.87727051185427 | 12th
0.69526495582091 | 1st-4th
0.70599054726772 | 5th-6th
1.1611784201972 | 7th-8th
0.8852814808038 | 9th
1.4635115503386 | Assoc-acdm
1.2741250142814 | Assoc-voc
1.5749857842579 | Bachelors
2.096963873495 | Doctorate
1.1911841663211 | HS-grad
1.927161656386 | Masters
0.53352632278282 | Preschool
2.0542073033733 | Prof-school
1.3084798585826 | Some-college

(16 rows)

DROP_TABLE
------------
t
(1 row)

**Related Functions**

- category Analytics - Statistics
- COND_ENTROPY
- JOINT_ENTROPY
- MUTUALINFO

**EWDISC - Discretization bins of equal width**

This stored procedure calculates the limits for discretization bins of equal width on numeric columns

**Usage**

The EWDISC stored procedure has the following syntax:

- EWDISC(NVARCHAR(ANY) paramString)

  ▲ Parameters
paramString
comma-separated list of <parameter>=<value> entries with parameters below
Type: NVARCHAR(ANY)

intable
the input table or view
Type: NVARCHAR(256)

outtable
the output table to store the discretization bin limits into
Type: NVARCHAR(256)

incolumn
the input table columns to consider, separated by a semi-colon (;).
Each column name may be followed by a colon (:) and the number of discretization bins to be calculated for this column.
Type: NVARCHAR(ANY)

bins
the default number of discretization bins to be calculated
Type: INTEGER
Default: 10

Returns
INTEGER the number of columns for which discretization bin limits are calculated

Details
This stored procedure calculates the limits for discretization bins of equal width on numeric columns. The width of the bins for a given column is calculated as (vmax - vmin)/k where k is the number of discretization bins requested for the column and vmin and vmax are the minimum and maximum values of the column. The discretization bin limits are placed at vmin + i * w, for i = 1, ..., k-1.

The discretization bin limits are written into the output table with following columns: colname, break. The column colname references the name of the input table columns, and the column break contains a bin limit for this column.

Examples
CALL nza..EWDISC('outtable=disc_bounds1, intable=nza..iris, incolumn=petallength:5;sepalwidth, bins=4');
SELECT * FROM disc_bounds1 ORDER BY colname, break;
CALL nza..APPLY_DISC('outtable=disc_iris_discrete, intable=nza..iris, btable=disc_bounds1, replace=T');
SELECT petallength, count(*) FROM disc_iris_discrete
GROUP BY petallength ORDER BY petallength;

SELECT sepalwidth, count(*) FROM disc_iris_discrete GROUP BY sepalwidth ORDER BY sepalwidth;

CALL nza..DROP_TABLE('disc_bounds1');

CALL nza..DROP_TABLE('disc_iris_discrete');

EW DISC
--------

2

(1 row)

<table>
<thead>
<tr>
<th>COLNAME</th>
<th>BREAK</th>
</tr>
</thead>
<tbody>
<tr>
<td>PETALLENGTH</td>
<td>2.18</td>
</tr>
<tr>
<td>PETALLENGTH</td>
<td>3.36</td>
</tr>
<tr>
<td>PETALLENGTH</td>
<td>4.54</td>
</tr>
<tr>
<td>PETALLENGTH</td>
<td>5.72</td>
</tr>
<tr>
<td>SEPALWIDTH</td>
<td>2.6</td>
</tr>
<tr>
<td>SEPALWIDTH</td>
<td>3.2</td>
</tr>
<tr>
<td>SEPALWIDTH</td>
<td>3.8</td>
</tr>
</tbody>
</table>

(7 rows)

APPLY DISC
---------

2

(1 row)

<table>
<thead>
<tr>
<th>PETALLENGTH</th>
<th>COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>50</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>34</td>
</tr>
<tr>
<td>4</td>
<td>47</td>
</tr>
<tr>
<td>5</td>
<td>16</td>
</tr>
</tbody>
</table>
(5 rows)

| SEPALWIDTH | COUNT |
|------------+-------|
| 1          | 24    |
| 2          | 84    |
| 3          | 36    |
| 4          | 6     |

(4 rows)

DROP_TABLE

---------
t

(1 row)

DROP_TABLE

---------
t

(1 row)

Related Functions

► category Analytics - Discretization
► EFDISC
► EMDISC
► EWDISC_NICE
► APPLY_DISC

EWDISC_NICE - Discretization bins of equal width with human-friendly limits

This stored procedure calculates the limits for discretization bins of equal width on numeric columns. The discretization bin limits are placed on human-friendly numeric values.
Usage

The EWDISC_NICE stored procedure has the following syntax:

► EWDISC_NICE(NVARCHAR(ANY) paramString)

▲ Parameters

► paramString
  comma-separated list of <parameter>=<value> entries with parameters below
  Type: NVARCHAR(ANY)

► intable
  the input table or view
  Type: NVARCHAR(ANY)

► outtable
  the output table to store the discretization bin limits into
  Type: NVARCHAR(ANY)

► incolumn
  the input table columns to consider, separated by a semi-colon (;).
  Each column name may be followed by a colon (:) and the number of discretization bins to be calculated for this column.
  Type: NVARCHAR(ANY)

► bins
  the default number of discretization bins to be calculated
  Type: INTEGER
  Default: 10

▲ Returns
  INTEGER the number of columns for which discretization bin limits are calculated

Details

This stored procedure calculates the limits for discretization bins of equal width on numeric columns. The width of the bins for a given column is calculated as $6 \times \text{stddev} / k$ where $k$ is the number of discretization bins requested for the column and stddev is the standard deviation of the column values. The width is then made "nice" by choosing the nearest scale 1, 2, 2.5 or 5 of a power of ten. The bin limits are then set around the column mean value so that the bin limits are multiples of the nice width.

NOTE: the resulting number of bins may slightly differ from the requested number of bins.

The discretization bin limits are written into the output table with following columns: colname, break. The column colname references the name of the input table columns, and the column break contains a bin limit for this column.

Examples

CALL nza..EWDISC_NICE('outtable=disc_bounds1, intable=nza..iris, incolumn=petallength:5;sepalwidth, bins=4');
SELECT * FROM disc_bounds1 ORDER BY colname, break;

CALL nza..APPLY_DISC('outtable=disc_iris_discrete, intable=nza..iris, btable=disc_bounds1, replace=T');

SELECT petallength, count(*) FROM disc_iris_discrete GROUP BY petallength ORDER BY petallength;

CALL nza..DROP_TABLE('disc_iris_discrete');

EWDISC_NICE
-------------
2
(1 row)

<table>
<thead>
<tr>
<th>COLNAME</th>
<th>BREAK</th>
</tr>
</thead>
<tbody>
<tr>
<td>PETALLENGTH</td>
<td>2</td>
</tr>
<tr>
<td>PETALLENGTH</td>
<td>3</td>
</tr>
<tr>
<td>PETALLENGTH</td>
<td>4</td>
</tr>
<tr>
<td>PETALLENGTH</td>
<td>5</td>
</tr>
<tr>
<td>PETALLENGTH</td>
<td>6</td>
</tr>
<tr>
<td>SEPALWIDTH</td>
<td>2.5</td>
</tr>
<tr>
<td>SEPALWIDTH</td>
<td>3</td>
</tr>
<tr>
<td>SEPALWIDTH</td>
<td>3.5</td>
</tr>
<tr>
<td>SEPALWIDTH</td>
<td>4</td>
</tr>
</tbody>
</table>

(9 rows)

APPLY_DISC
-------------
2
(1 row)
### PETALLENGTH | COUNT

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>50</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>15</td>
</tr>
<tr>
<td>4</td>
<td>42</td>
</tr>
<tr>
<td>5</td>
<td>33</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
</tr>
</tbody>
</table>

(6 rows)

### SEPALWIDTH | COUNT

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>19</td>
</tr>
<tr>
<td>2</td>
<td>64</td>
</tr>
<tr>
<td>3</td>
<td>49</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>

(5 rows)

### DROP_TABLE

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
</tr>
</tbody>
</table>

(1 row)

### DROP_TABLE

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
</tr>
</tbody>
</table>

(1 row)

### Related Functions

- category Analytics - Discretization
- EFDISC
EMDISC
EWDISC
APPLY_DISC

IBM Netezza In-Database Analytics Reference Guide

EXPORT_MODEL - Export one or more analytic models to a file set

This stored procedure exports the specified analytic models in a proprietary format to a set of files. All files are stored in the specified directory and have a common prefix. Model contents, model metadata, and model privileges can be exported.

Usage

The EXPORT_MODEL stored procedure has the following syntax:

```
EXPORT_MODEL(NVARCHAR(ANY))
```

Parameters

- **paramString**
  a comma-separated list of `<parameter>=<value>` entries with parameters below
  Type: NVARCHAR(ANY)

- **model**
  a semicolon-separated list of names of models to be exported. If all models must be exported, specify model=all. If parameter where is not specified, parameter model must be specified.
  Type: NVARCHAR(ANY)

- **where**
  a WHERE clause to filter the set of models to be exported. All columns from view V_NZA_MODELS can be used in the WHERE clause. If parameter model is not specified, parameter where must be specified.
  Type: NVARCHAR(ANY)

- **directory**
  the name of the directory where all result files are written to. The directory must exist. An absolute path name must be specified.
  Type: NVARCHAR(ANY)

- **name**
  the name of the result file set. This name is used as prefix for each file of the file set. If a file set with this name already exists, use the parameter overwrite to delete it first.
  Type: NVARCHAR(ANY)

- **acl**
  a flag ("access control list") controlling export of privileges. Only the administrator can use this parameter. If set to true, model privileges are exported also.
  Type: BOOLEAN
► **overwrite**
  a flag used to overwrite existing files. If a file set with the specified name already exists in the target directory, export is possible only if overwrite is set to true. In this case all existing files of the file set are deleted.
  
  Type: BOOLEAN

► **verbose**
  a flag controlling if additional output about the progress of the export process is printed.
  
  Type: BOOLEAN

▲ **Returns**

  INTEGER the number of successfully exported models

**Details**

The parameters model and where are used to specify the models to be exported. If both parameters are specified, the two model sets are merged. To export all models, pass model=all (the where parameter is ignored in this case). You can use any existing directory as target directory that can be written by the database process. A directory can contain export file sets of different export operations, but all of them must have different names. The overwrite parameter can be used to automatically delete file sets with the same name before export. If a model cannot be exported (for example, the user does not have the SELECT privilege for this model), the export of this model is skipped and the system continuous the export with the next model. Only on severe errors the export process is aborted completely. Only the database administrator can export the model privileges (which user and which group has which access right on a model), using the acl parameter.

**Examples**

```
CALL nza..TIMESERIES('model=tsmodel,intable=nza..iris,time=ID,target=PETALLENGTH,by=CLASS,algorithm=esmoothing');

CALL nza..EXPORT_MODEL('model=tsmodel,directory=/tmp/export,name=tsexport,acl=true');

CALL nza..DROP_MODEL('model=tsmodel');

TIMESERIES
----------
   3
(1 row)

EXPORT_MODEL
------------
   1
(1 row)

NOTICE: Dropped: TSMODEL
```
EXPORT_PMML - Export an analytics model as PMML document to a file

This stored procedure exports the given analytics model as PMML document to a file, or it exports a model from a PMML table to a file. If no PMML table exists containing the PMML document for this model, one can be created automatically when requested. Optionally, instead of writing to a file, the result can be returned by the procedure.

Usage

The EXPORT_PMML stored procedure has the following syntax:

- **EXPORT_PMML(NVARCHAR(ANY))**
  - **Parameters**
    - **paramString**
      - comma-separated list of <parameter>=<value> entries with parameters below
      - Type: NVARCHAR(ANY)
    - **model**
      - the model to be exported as PMML document to a file
      - Type: NVARCHAR(64)
    - **type**
      - the type of PMML document: "standard" (default), or "spss"
      - Type: VARCHAR(16)
    - **intable**
      - the table containing the PMML model to be exported
      - Type: NVARCHAR(256)
    - **file**
      - the file (absolute path name) to write the PMML document into. If file is not specified, the PMML document is returned by the procedure.
      - Type: NVARCHAR(256)
outtable

the output table to write the PMML document into. If it is not specified, the PMML document
will be written to the file only. This parameter is ignored if a PMML table exists already for the
given model. This parameter is not allowed if intable was specified.

Type: NVARCHAR(128)

Returns

TEXT if a file is specified, always 't' (otherwise an exception is raised). If no file is specified, the
PMML document.

Details

This stored procedure can be called in two ways:

1. EXPORT_PMML('model=..., type=..., file=..., outtable=...'), where model is mandatory

2. EXPORT_PMML('intable=..., file=..., model=...'), where intable is mandatory

In case 1, the stored procedure exports the given analytics model as PMML document to a file (or returns
the PMML document).

- If there is a registered table for the model containing the PMML document of the given type, the stored
procedure reads the table, concatenates the rows and writes them to the given file. In this case, the out-
table parameter is ignored.

- If there is no registered table for the model containing the PMML document of the given type, and the
<outtable> parameter is specified, the procedure internally calls PMML_MODEL to create the table first. It
then copies the PMML document from the table to the given file as above.

- Otherwise, the PMML document is created directly (without an intermediate PMML table) and written to
the file.

In case 2, the stored procedure exports an analytics model from a PMML table to a file. The PMML table is
specified by the intable parameter. If the PMML table contains more than one model, the model parameter
is needed to identify the model, else it is ignored.

NOTE: If possible, an existing file is overwritten without notice.

Examples

CALL nza..ARULE('intable=nza..retail, model=mbamodel,
supporttype=percent, support=5, lvl=0, maxsetsize=5,
confidence=0.5');

CALL nza..EXPORT_PMML('model=mbamodel,
file=/tmp/mbamodel.pmml');

CALL nza..DROP_MODEL('model=mbamodel');

NOTICE:

RUNNING FP Growth algorithm:

DATASET : "NZA"."RETAIL"
Transaction column : "TID"
Item column : "ITEM"
Group by : <none>
Minimum support : 5 %
Minimum confidence: 0.5
Max frequent itemset size : 5
Level of conditional dbs : 0
Result tables prefix : "NZA_META_MBAMODEL"

ARULE
-------
14
(1 row)
EXPORT_PMML

---------------
t
(1 row)

NOTICE: Dropped: MBAMODEL

DROP_MODEL
---------------
t
(1 row)

Related Functions
► category Analytics - Model Management
► LIST_MODELS
► PMML_MODEL

FMEASURE - F-Measure from a Confusion Matrix

This stored procedure calculates the F-measure of a class from a confusion matrix

Usage

The FMEASURE stored procedure has the following syntax:

► FMEASURE(NVARCHAR(ANY) paramString)
  ▲ Parameters
paramString
coma-separated list of <parameter>=<value> entries with parameters below
Type: NVARCHAR(ANY)

matrixTable
t the confusion matrix table
Type: NVARCHAR(256)

class
t the class in the confusion matrix table to calculate the F-measure on
Type: NVARCHAR(ANY)

Returns
DOUBLE the F-Measure

Details
This stored procedure calculates the F-measure of a class from a confusion matrix. The F-measure is calculated as 2 * TPR * PPV / (TPR + PPV), where TPR and PPV are the result of the corresponding stored procedures called on the same confusion matrix.

The confusion matrix table has the following columns: real, prediction, cnt. The column cnt contains the frequency of making a given prediction for the given real value.

Examples

CALL nza..TRAIN_TEST('modelType=dectree, model=iris_c45, traintable=nza..iris_train, testtable=nza..iris_test, target=class, id=id, outtable=iris_pred, minsplit=2');

CALL nza..CONFUSION_MATRIX('intable=nza..iris_test, resulttable=iris_pred, id=id, resultid=id, target=class, resulttarget=class, matrixTable=confMatrix');

CALL nza..FMEASURE('matrixTable=confMatrix, class=virginica');

CALL nza..DROP_MODEL('model=iris_c45');

CALL nza..DROP_TABLE('iris_pred');

CALL nza..DROP_TABLE('confMatrix');

TRAIN_TEST
----------
0.918919
(1 row)

CONFUSION_MATRIX
-----------------
5
Related Functions

- category Analytics - Diagnostic Measures
- CONFUSION_MATRIX
- TPR
- PPV

FPR - False Positive Rate from a Confusion Matrix

This stored procedure calculates the False Positive Rate of a class from a confusion matrix
Usage
The FPR stored procedure has the following syntax:

► **FPR(NVARCHAR(ANY) paramString)**

▲ Parameters

► **paramString**

   comma-separated list of <parameter>=<value> entries with parameters below
   
   Type: NVARCHAR(ANY)

► **matrixTable**

   the confusion matrix table
   
   Type: NVARCHAR(256)

► **class**

   the class in the confusion matrix table to calculate the False Positive Rate on
   
   Type: NVARCHAR(ANY)

▲ Returns

   DOUBLE the False Positive Rate

Details
This stored procedure calculates the False Positive Rate of a class from a confusion matrix. The False Positive Rate is the ratio of the number of wrongly classified records into the given class divided by the number of real values that are not the given class.

The confusion matrix table has the following columns: real, prediction, cnt. The column cnt contains the frequency of making a given prediction for the given real value.

Examples

CALL nza..TRAIN_TEST('modelType=dectree, model=iris_c45, traintable=nza..iris_train, testtable=nza..iris_test, target=class, id=id, outtable=iris_pred, minsplit=2');

CALL nza..CONFUSION_MATRIX('intable=nza..iris_test, resulttable=iris_pred, id=id, resultid=id, target=class, resulttarget=class, matrixTable=confMatrix');

CALL nza..FPR('matrixTable=confMatrix, class=virginica');

CALL nza..DROP_MODEL('model=iris_c45');

CALL nza..DROP_TABLE('iris_pred');

CALL nza..DROP_TABLE('confMatrix');

TRAIN_TEST
-----------

0.918919

(1 row)
CONFUSION_MATRIX
---------------
  5
(1 row)

FPR
-----
0.035714
(1 row)

DROP_MODEL
-----------
t
(1 row)

DROP_TABLE
-----------
t
(1 row)

Related Functions
► category Analytics - Diagnostic Measures
► CONFUSION_MATRIX
GET_COLUMN_LIST - Retrieve a list of columns with a given role and type from a column properties table

This stored procedure looks into a column properties table and returns those columns that have the desired roles and types.

Usage

The GET_COLUMN_LIST stored procedure has the following syntax:

```
GET_COLUMN_LIST((NVARCHAR(ANY) paramString)
▲ Parameters
  ▶ paramString
    Comma-separated list of <parameter>=<value> entries with parameters below
    Type: NVARCHAR(ANY)
  ▶ colPropertiesTable
    Full valid name of the column properties table in the form database.schema.objectname. Database and schema are optional (default: current database/default database, current schema).
    Type: NVARCHAR(ANY)
  ▶ role
    The expected column roles of the result columns, separated by a semicolon or comma. If it is NULL or empty, all columns are matched. Allowed values are: 'id', 'target', 'input', 'ignore', 'obj-weight', 'trials'.
    Type: NVARCHAR(ANY)
    Default: <none>
  ▶ type
    The expected column types of the result columns, separated by a semicolon or comma. If it is NULL or empty, all columns are matched. Allowed values are: 'nom', 'cont'.
    Type: NVARCHAR(ANY)
    Default: <none>
  ▶ separator
    A string used to separate the result columns (typically a single character)
    Type: NVARCHAR(ANY)
▲ Returns
  NVARCHAR(ANY) The list of columns, filtered by their column role and column type.
```

Examples

```
CALL nza..COLUMN_PROPERTIES('intable=nza..weatherr, outtable=weatherDefSchema, coldeftype=nom ,incolumn=temperature| GRADE:cont;WINDY:ignore');

CALL nza..GET_COLUMN_LIST('colPropertiesTable=weatherDefSchema, role=input;id,target;objweight, type=nom;cont, separator=";"');
```
CALL nza..DROP_TABLE('weatherDefSchema');
COLUMN_PROPERTIES
-------------------
6
(1 row)

GET_COLUMN_LIST
------------------------------------------------------
"INSTANCE";"OUTLOOK";"TEMPERATURE";"HUMIDITY";"GRADE"
(1 row)

DROP_TABLE
------------

t
(1 row)

Related Functions
▶ category Analytics - Column Properties
▶ COLUMN_PROPERTIES

GLM - Build a Generalized Linear Model
This stored procedure builds a Generalized Linear Regression model

Usage
The GLM stored procedure has the following syntax:

▶ GLM(NVARCHAR(ANY) paramString)
  ▲ Parameters
  ▶ paramString
    comma-separated list of <parameter>=<value> entries with parameters below
    Type: NVARCHAR(ANY)
  ▶ model
    the name of the GLM model
Reference Documentation: Analytics

Type: NVARCHAR(ANY)

▶ `intable`
the input table
Type: NVARCHAR(256)

▶ `id`
the input table column identifying a unique instance id
Type: NVARCHAR(128)

▶ `target`
the input table column to predict a value for. Only numeric type of target column is accepted.
Type: NVARCHAR(128)

▶ `incolumn`
the input table columns with special properties, separated by a semi-colon (;).
Each column is followed by one or several of the following properties:
- its type: ':nom' (for nominal), ':cont' (for continuous). Per default, all numerical types are continuous, other types are nominal.
- its role: ':id', ':target', ':input', ':ignore', ':objweight'.
(Remark: ':colweight(<wgt>)' is unsupported, i.e. ':colweight(<wgt>)' same as ':colweight(1)' same as ':input').
If the parameter is undefined, all columns of the input table have default properties.
Type: NVARCHAR(ANY)
Default: <none>

▶ `coldeftype`
default type of the input table columns. Allowed values are 'nom' and 'cont'.
If the parameter is undefined, all numeric columns are considered continuous, other columns nominal.
Type: NVARCHAR(ANY)
Default: <none>

▶ `coldefrole`
default role of the input table columns. Allowed values are 'input' and 'ignore'.
If the parameter is undefined, all columns are considered 'input' columns.
Type: NVARCHAR(ANY)
Default: <none>

▶ `colPropertiesTable`
the input table where column properties for the input table columns are stored. The format of this table is the output format of stored procedure nza..COLUMN_PROPERTIES().
If the parameter is undefined, the input table column properties will be detected automatically.
(Remark: colPropertiesTable with "COLWEIGHT" column with value '<wgt>' is unsupported, i.e. same as '1')
Type: NVARCHAR(256)
Default: <none>

► intercept
flag indicating whether the model is built with or without an intercept value
Type: BOOLEAN
Default: true

► interaction
the definition of the allowed interactions between input columns. The interaction is a
list of factors separated by a semicolon (;). A factor is a list of variables separated by a
star (*). A variable is a column name of the input table. Continuous variables can be
followed by a caret (^) and a numeric value, in this case the given power of values of
this column is meant. Nominal variables can be followed by a sign equal (=) and a
value, so that only the given value of the variable is allowed to interact with the other
variables of this factor. If no value is indicated after a nominal variable, all distinct val-
ues interact independantly with the other variables of the factor. By default, all input
columns are considered independent and do not interact with each other.
Type: NVARCHAR(ANY)
Default: '

► family
the type of the distribution. Allowed values are: bernoulli, gaussian, poisson, binomial,
negativebinomial, wald (inversegaussian), gamma.
Type: NVARCHAR(ANY)
Default: bernoulli

► family_param
additional parameter used for some distributions. IF family_param='quasi' than quasi-
likelihood in case of Poisson and Binomial distributions is optimized. IF
family_param=-1 (or is omitted than mentioned distribution parameter is estimated
from data. IF family_param is given explicit than should by > 0.
Type: DOUBLE
Default: -1

► link
the type of the link function. Allowed values are: canbinom, cangeom, cannegbinom,
cauchit, clog, cloglog, gausit, identity, inverse, invnegative, invsquare, log, logit, log-
log, oddspower, power, probit, sqrt.
Type: NVARCHAR(ANY)
Default: logit

► link_param
an additional parameter used for some links like: cannegbinom, oddspower, power.
The range of value depends on the used link function.
Type: DOUBLE
Default: 1

► **maxit**
the maximum number of interactions
Type: INTEGER
Default: 20

► **eps**
the maximum (relative) error used as stopping criteria
Type: DOUBLE
Default: 1e-3

► **tol**
the tolerance for the linear equation solver when to consider a value to be equal to zero
Type: DOUBLE
Default: 1e-7

► **method**
the method used to calculate a GLM model. Allowed values are: irls, psgd.
Type: NVARCHAR(ANY)
Default: irls

► **trials**
- the input table column containing the number of trials for the binomial distribution. This parameter must be specified when family=binomial. This parameter is ignored for other distributions.
Type: NVARCHAR(ANY)
Default: ''

► **debug**
flag indicating to display debug information
Type: BOOLEAN
Default: false

▲ Returns
BIGINT the number of variables (column or column=value) used in the GLM model

**Details**
This stored procedure builds a Generalized Linear model. The model is saved to the database in a set of tables and registered in the database model metadata. Use the Model Management functions to further manipulate the model or access the model tables.

**Examples**
```
CALL nza..GLM('model=adult_glm, intable=nza..adult_train, id=id, target=age');
```
CALL nza..DROP_MODEL('model=adult_glm');

GLM
-----
58
(1 row)

DROP_MODEL
------------
t
(1 row)

Related Functions
► category Analytics - Regression
► PREDICT_GLM
► PRINT_GLM
► LIST_MODELS

GRANT_MODEL - Grant privileges on an analytics model
This stored procedure grants one or more privileges on an analytics model to users and/or groups

Usage
The GRANT_MODEL stored procedure has the following syntax:

► GRANT_MODEL(NVARCHAR(ANY))
  ▲ Parameters
    ► paramString
      comma-separated list of <parameter>=<value> entries with parameters below
      Type: NVARCHAR(ANY)
    ► model
      the model to be granted privileges on
      Type: NVARCHAR(64)
    ► privilege
      a semicolon-separated list of privileges. Allowed privileges are: list, select, alter, update, drop.
Type: VARCHAR(256)

► **user**
a semicolon-separated list of users to grant privileges to. If not specified, no user is granted the privileges, and the parameter group has to be specified.
Type: NVARCHAR(256)
Default: <none>

► **group**
a semicolon-separated list of groups to grant privileges to. If not specified, no group is granted the privileges, and the parameter user has to be specified.
Type: NVARCHAR(256)
Default: <none>

► **grantoption**
flag indicating if the users or groups can further grant the given privileges to other users or groups
Type: BOOLEAN
Default: false

▲ Returns
BOOLEAN always true (otherwise an exception is raised)

**Details**
This stored procedure grants privileges to users and groups on the given model. The possible privileges are:

- **LIST**: Allows to list a model and its properties (using the views and the list procedures),
- **SELECT**: Allows to read the model contents (to SELECT the model tables, print the model, convert the model to PMML),
- **ALTER**: Allows to change the model properties (name, owner, ...),
- **UPDATE**: Allows to change the model contents using specific procedures,
- **DROP**: Allows to drop a model.

To grant a privilege to a user or group (including group "PUBLIC"), you must have the LIST privilege on this user or group. By default, the owner of a model, the ADMIN user, and the database owner have all privileges on a model. You cannot grant privileges to yourself.

**Examples**

```
CALL nza..ARULE('intable=nza..retail, model=mbamodel,
supporttype=percent, support=5, lvl=0, maxsetsize=5,
confidence=0.5');

CALL nza..GRANT_MODEL('model=mbamodel, privilege=LIST;SELECT,
user=inzauser');

CALL nza..DROP_MODEL('model=mbamodel');
```

**NOTICE:**
RUNNING FPGrowth algorithm:
DATASET : "NZA".."RETAIL"
Transaction column : "TID"
Item column : "ITEM"
Group by : <none>
Minimum support : 5 %
Minimum confidence: 0.5
Max frequent itemset size : 5
Level of conditional dbs : 0
Result tables prefix : "NZA_META_MBAMODEL"

ARULE
-------
14
(1 row)

GRANT_MODEL
-------------
t
(1 row)

NOTICE: Dropped: MBAMODEL

DROP_MODEL
-----------
t
(1 row)

Related Functions
► category Analytics - Model Management
► LIST_MODELS
► REVOKE_MODEL
GROW_DECTREE - Build a Decision Tree model

This stored procedure builds a Decision Tree model

Usage

The GROW_DECTREE stored procedure has the following syntax:

```
GROW_DECTREE(NVARCHAR(ANY) paramString)
```

Parameters

- **paramString**
  The comma-separated list of `<parameter>=<value>` entries with parameters below
  Type: NVARCHAR(ANY)

- **model**
  The name of the Decision Tree model to build.
  Type: NVARCHAR(ANY)

- **intable**
  The name of the input table.
  Type: NVARCHAR(256)

- **id**
  The column of the input table that identifies a unique instance ID.
  Type: NVARCHAR(128)

- **target**
  The input table column representing the class.
  Type: NVARCHAR(128)

- **incolumn**
  The input table columns with special properties, separated by a semi-colon (;).
  Each column is followed by one or several of the following properties:
  - type: ':nom' (for nominal), ':cont' (for continuous). By default, all numerical types are continuous, other types are nominal
  - role: ':id', ':target', ':input', ':ignore'.
  (Remark: ':objweight' is unsupported, i.e. ':objweight' same as ':ignore').
  (Remark: ':colweight(<wgt>)' is unsupported, i.e. ':colweight(<wgt>)' same as ':colweight(1)' same as ':input').
  If the parameter is undefined, all columns of the input table have default properties.
  Type: NVARCHAR(ANY)
  Default: <none>

- **coldeftype**
  The default type of the input table columns. Valid values are 'nom' and 'cont'.
If the parameter is undefined, all numeric columns are considered continuous, other columns nominal.
Type: NVARCHAR(ANY)
Default: <none>

► **coldefrole**
The default role of the input table columns. Allowed values are 'input' and 'ignore'.
If the parameter is undefined, all columns are considered 'input' columns.
Type: NVARCHAR(ANY)
Default: <none>

► **colPropertiesTable**
The input table where column properties for the input table columns are stored. The format of this table is the output format of the nza..COLUMN.PROPERTIES() stored procedure.
If the parameter is undefined, the input table column properties is detected automatically.

(Remark: colPropertiesTable with "COLROLE" column with value 'objweight' is unsupported, i.e. same as 'ignore')
(Remark: colPropertiesTable with "COLWEIGHT" column with value '<wgt>' is unsupported, i.e. same as '1')
Type: NVARCHAR(256)
Default: <none>

► **weights**
The input table containing optional instance or class weights for the input table columns.
If the parameter is undefined, it is assumed that the weights are uniformly equal to 1.
The <weights> table contains following columns:
- weight: a numeric column containing the instance or class weight
- id: a column to be joined with the <id> column of <intable>, defining instance weights
- class: a column to be joined with the <target> column of <intable>, defining class weights
Either the id or class column can be missing, at least one of them must be present. For instances or classes not occurring in this table, weights of 1 are assumed.
Type: NVARCHAR(256)
Default: <none>

► **eval**
The class impurity measure used for split evaluation. Valid values are 'entropy' and
'gini'.
Type: NVARCHAR(ANY)
Default: entropy

► **minimprove**
The minimum improvement of the split evaluation measure that is required.
Type: DOUBLE
Default: 0.01
Min: 0.0

► **minsplit**
The minimum number of instances per tree node that can be split.
Type: INTEGER
Default: 50
Min: 2

► **maxdepth**
The maximum number of tree levels, including leaves.
Type: INTEGER
Default: 10
Min: 1
Max: 62

► **statistics**
flags indicating which statistics to collect. Allowed values are: none, columns, values:n, all.
If statistics=none, no statistics are collected.
If statistics=columns, statistics on the input table columns like mean value are collected.
If statistics=values:n with n a positive number, statistics about the columns and the column values are collected. Up to <n> column value statistics are collected:
- If a nominal column contains more than <n> values, only the <n> most frequent column statistics are kept.
- If a numeric column contains more than <n> values, the values will be discretized and the statistics will be collected on the discretized values.
Indicating statistics=all is equal to statistics=values:100.
Type: NVARCHAR(ANY)
Default: 'none'

▲ Returns
INTEGER The number of Decision Tree nodes, including leaves.

**Details**
This stored procedure builds a Decision tree model. The model is saved to the database in a set of tables and registered in the database model metadata. Use the Model Management functions to further manipu-
late the model or access the model tables.
A top-down tree growing algorithm is used with the following features:
- binary splits (equality-based for nominal attributes, inequality-based for continuous attributes)
- no missing value handling
- entropy or Gini index for split evaluation
- stop criteria satisfied at a node with a uniform class
- stop criteria satisfied when further splits do not improve the class impurity by at least <minimprove>
- stop criteria satisfied when the number of instances is less than <minsplit>
- stop criteria satisfied when the tree depth reaches <maxdepth>

Examples

```
CALL nza..GROW_DECTREE('model=adult_tree,
intable=nza..adult_train, id=id, target=income,
eval=gini, minsplit=1000, maxdepth=62');
CALL nza..DROP_MODEL('model=adult_tree');
```

```
GROW_DECTREE
-------------------
  11
(1 row)

DROP_MODEL
------------
  t
(1 row)
```

Related Functions

- category Analytics - Classification
- DECTREE
- PRUNE_DECTREE
- PRINT_DECTREE
- PREDICT_DECTREE
- LIST_MODELS

GROW_REGTREE - Grow a Regression Tree model

This stored procedure builds a Regression Tree model
Usage

The GROW_REGTREE stored procedure has the following syntax:

**GROW_REGTREE(NVARCHAR(ANY) paramString)**

### Parameters

- **paramString**
  
  comma-separated list of `<parameter>=<value>` entries with parameters below
  
  Type: NVARCHAR(ANY)

- **model**
  
  the name of the Regression Tree model to build
  
  Type: NVARCHAR(ANY)

- **intable**
  
  the input table
  
  Type: NVARCHAR(256)

- **id**
  
  the input table column identifying a unique instance id
  
  Type: NVARCHAR(128)

- **target**
  
  the input table column representing the prediction target
  
  Type: NVARCHAR(128)

- **incolumn**
  
  the input table columns with special properties, separated by a semi-colon (;).
  
  Each column is followed by one or several of the following properties:
  
  - its type: `:nom` (for nominal), `:cont` (for continuous). Per default, all numerical types are continuous, other types are nominal.
  

  (Remark: `:objweight` is unsupported, i.e. `:objweight` same as `:ignore`).

  (Remark: `:colweight(<wgt>)` is unsupported, i.e. `:colweight(<wgt>)` same as `:colweight(1)` same as `:input`).

  If the parameter is undefined, all columns of the input table have default properties.

  Type: NVARCHAR(ANY)

  Default: <none>

- **coldeftype**
  
  default type of the input table columns. Allowed values are `:nom` and `:cont`.

  If the parameter is undefined, all numeric columns are considered continuous, other columns nominal.

  Type: NVARCHAR(ANY)

  Default: <none>
► **coldefrole**
  default role of the input table columns. Allowed values are 'input' and 'ignore'.
  If the parameter is undefined, all columns are considered 'input' columns.
  Type: NVARCHAR(ANY)
  Default: <none>

► **colPropertiesTable**
  the input table where column properties for the input table columns are stored. The format of this table is the output format of stored procedure nza..COLUMN_PROPERTIES().
  If the parameter is undefined, the input table column properties will be detected automatically.
  (Remark: colPropertiesTable with "COLROLE" column with value 'objweight' is unsupported, i.e. same as 'ignore')
  (Remark: colPropertiesTable with "COLWEIGHT" column with value '<wgt>' is unsupported, i.e. same as '1')
  Type: NVARCHAR(256)
  Default: <none>

► **eval**
  the split evaluation measure. Allowed values are: variance.
  Type: NVARCHAR(ANY)
  Default: variance

► **minimprove**
  the minimum improvement of the split evaluation measure required
  Type: DOUBLE
  Default: 0.1
  Min: 0.00001

► **minsplit**
  the minimum number of instances per tree node that can be split
  Type: INTEGER
  Default: 50
  Min: 2

► **maxdepth**
  the maximum number of tree levels (including leaves)
  Type: INTEGER
  Default: 10
  Min: 1
Max: 62

► statistics
flags indicating which statistics to collect. Allowed values are: none, columns, values:n, all.
If statistics=none, no statistics are collected.
If statistics=columns, statistics on the input table columns like mean value are collected.
If statistics=values:n with n a positive number, statistics about the columns and the column values are collected. Up to <n> column value statistics are collected:
- If a nominal column contains more than <n> values, only the <n> most frequent column statistics are kept.
- If a numeric column contains more than <n> values, the values will be discretized and the statistics will be collected on the discretized values.
Indicating statistics=all is equal to statistics=values:100.
Type: NVARCHAR(ANY)
Default: 'none'

▲ Returns
INTEGER the number of Regression Tree nodes (including leaves)

Details
This stored procedure builds a Regression tree model. The model is saved to the database in a set of tables and registered in the database model metadata. Use the Model Management functions to further manipulate the model or access the model tables.
A top-down tree growing algorithm is used with the following features:
- binary splits (equality-based for nominal attributes, inequality-based for continuous attributes),
- no missing value handling,
- variance for split evaluation,
- stop criteria satisfied when further splits do not improve the variance by at least <minimprove>,
- stop criteria satisfied when the number of instances is less than <minsplit>,
- stop criteria satisfied when the tree depth reaches <maxdepth>.

Examples

```sql
CALL nza..GROW_REGTREE('model=wrt, intable=nza..weatherr, id=instance, target=grade, minsplit=2, maxdepth=3');
```

```sql
CALL nza..DROP_MODEL('model=wrt');
```

```
GROW_REGTREE
-------------
  7
(1 row)
DROP_MODEL
```
HIST - Histograms

This stored procedure creates histograms. The number of bins and the bins themselves can be specified or are automatically calculated.

**Usage**

The HIST stored procedure has the following syntax:

```
HIST(NVARCHAR(ANY) paramString)
```

**Parameters**

- **paramString**
  - comma-separated list of <parameter>=<value> entries with parameters below
  - Type: NVARCHAR(ANY)

- **intable**
  - the input table or view
  - Type: NVARCHAR(ANY)

- **incolumn**
  - the input table column to build the histogram onto
  - Type: NVARCHAR(ANY)

- **outtable**
  - the output table to store the histogram into
  - Type: NVARCHAR(ANY)

- **nbreaks**
  - the number of bins for the histogram. If not specified, the number of bins is calculated automatically.
  - Type: INT4
Default: <none>
Min: 1

► **right**
the flag indicating whether the histogram bins should be right-closed (true) or right-open (false)
Type: BOOLEAN
Default: true

► **btable**
the input table with breaks for the histogram. If not specified, the bins are calculated automatically, using the parameter <nbreaks> if specified.
Type: NVARCHAR(ANY)
Default: <none>

► **bcolumn**
the <btable> column containing the breaks for the histogram. This column must be specified if the parameter <btable> is specified.
Type: NVARCHAR(ANY)
Default: <none>

► **density**
flag indicating whether densities should be attached to the output table
Type: BOOLEAN
Default: false

► **midpoints**
flag indicating whether the midpoints of the bins should be attached to the output table
Type: BOOLEAN
Default: false

► **freq**
flag indicating whether frequencies should be attached to the output table
Type: BOOLEAN
Default: false

► **cum**
flag indicating whether cumulative frequencies should be attached to the output table (setting this flag automatically sets freq flag as frequencies have to be calculated prior to cumulative frequencies)
Type: BOOLEAN
Default: false

▲ Returns
INTEGER the number of created bins

**Details**
This stored procedure calculates an histogram for the input table column. Three modes are available:
- When neither parameter `<nbreaks>` nor parameter `<btable>` are specified, the histogram contains equally-sized bins that are automatically calculated.

- When parameter `<btable>` is specified, the histogram contains the bins whose boundaries are defined in column `<bcolumn>` of `<btable>`.

- When parameter `<nbreaks>` is specified, the histogram contains `<nbreaks>` equally-sized bins that are automatically calculated.

If parameters `<nbreaks>` and `<btable>` are specified simultaneously, parameter `<nbreaks>` is ignored.

The histogram is stored in an output table with following columns: `idx`, `bleft`, `bright`, `count`. The histogram bins limits are defined in columns `bleft` and `bright`, and can be sorted in ascending order using the column `idx`. The frequency of the input column values per bin is indicated in column `count`.

Four additional columns density, midpoint, `freq` (frequency) and `cum` (cumulative frequency) can be available in the output table if the corresponding parameters are set to true.

**Examples**

```sql
CALL nza..HIST('intable=nza..iris,incolumn=petallength,outtable=hist1,right=F');
SELECT * FROM hist1 ORDER BY idx;
CALL nza..DROP_TABLE('hist1');
```

```
HIST
------
  12
(1 row)

<table>
<thead>
<tr>
<th>IDX</th>
<th>BLEFT</th>
<th>BRIGHT</th>
<th>COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1.5</td>
<td>23</td>
</tr>
<tr>
<td>1</td>
<td>1.5</td>
<td>2</td>
<td>27</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2.5</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>2.5</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>3.5</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>3.5</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>4.5</td>
<td>18</td>
</tr>
<tr>
<td>7</td>
<td>4.5</td>
<td>5</td>
<td>25</td>
</tr>
</tbody>
</table>
```
8 | 5 | 5.5 | 18
9 | 5.5 | 6 | 17
10 | 6 | 6.5 | 7
11 | 6.5 | 7 | 4
(12 rows)

DROP_TABLE
------------
t
(1 row)

CALL
nza..HIST('intable=nza..iris,incolumn=petallength,outtable=hist2,nbreaks=10,right=T');
SELECT * FROM hist2 ORDER BY idx;
CALL nza..DROP_TABLE('hist2');

HIST
-----
10
(1 row)

IDX | BLEFT | BRIGHT | COUNT
-----+-------+--------+-------
0 | 1 | 1.59 | 37
1 | 1.59 | 2.18 | 13
2 | 2.18 | 2.77 | 0
3 | 2.77 | 3.36 | 3
4 | 3.36 | 3.95 | 8
5 | 3.95 | 4.54 | 26
6 | 4.54 | 5.13 | 29
7 | 5.13 | 5.72 | 18
8 | 5.72 | 6.31 | 11
9 | 6.31 | 6.9 | 5
(10 rows)
**DROP_TABLE**

```
DROP_TABLE
---------
t
(1 row)
```

CREATE TABLE gdm_hbr (br DOUBLE);

```
CREATE TABLE gdm_hbr (br DOUBLE);
```

```
INSERT INTO gdm_hbr VALUES(0);
```

```
INSERT INTO gdm_hbr VALUES(2);
```

```
INSERT INTO gdm_hbr VALUES(4);
```

```
INSERT INTO gdm_hbr VALUES(6);
```

```
INSERT INTO gdm_hbr VALUES(8);
```

```
CALL nza..HIST('intable=nza..iris,incolumn=petallength,outtable=hist3,btable=gdm_hbr,bcolumn=br');
```

```
SELECT * FROM hist3 ORDER BY idx;
```

```
CALL nza..DROP_TABLE('hist3');
```

```
CALL nza..DROP_TABLE('gdm_hbr');
```

**HIST**

```
-------
4
(1 row)
```

<table>
<thead>
<tr>
<th>IDX</th>
<th>BLEFT</th>
<th>BRIGHT</th>
<th>COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>50</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>6</td>
<td>75</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

(4 rows)
DROP_TABLE
------------
t
(1 row)

DROP_TABLE
------------
t
(1 row)

CALL nza..HIST('in_table=nza..iris,in_column=sepal_length,out_table=hist4,right=F,density=T,midpoints=T,freq=T,cum=T');
SELECT * FROM hist4 ORDER BY idx;
CALL nza..DROP_TABLE('hist4');

HIST
------
8
(1 row)

IDX | BLEFT | BRIGHT | COUNT  | DENSITY | MIDPOINT | FREQ  | CUM
----+-------+--------+--------+---------+---------+-------+-----
 0 | 4     | 4.5    | 4      | 0.053333333333333 | 4.25    | 0.026666666666667 | 0.03
 1 | 4.5   | 5      | 18     | 0.24    | 4.75    | 0.12  | 0.15
 2 | 5     | 5.5    | 30     | 0.4     | 5.25    | 0.2   | 0.35
 3 | 5.5   | 6      | 31     | 0.413333333333333 | 5.75    | 0.206666666666667 | 0.55
 4 | 6     | 6.5    | 32     | 0.426666666666667 | 6.25    | 0.213333333333333 | 0.77
 5 | 6.5   | 7      | 22     | 0.293333333333333 | 6.75    | 0.146666666666667 | 0.91
DROP_TABLE

------------

t

(1 row)

Related Functions

► category Analytics - Statistics
► UNITABLE
► BITABLE

IMPORT_MODEL - Import one or more analytics models from a file set

This stored procedure imports the specified analytic models from a set of files into an INZA-enabled database. Model contents, model metadata, and model privileges can be imported. The file set must have been created with the EXPORT_MODEL procedure before.

Usage

The IMPORT_MODEL stored procedure has the following syntax:

► IMPORT_MODEL(NVARCHAR(ANY))
  ▲ Parameters
  ▶ paramString
     a comma-separated list of <parameter>=<value> entries with parameters below
     Type: NVARCHAR(ANY)
  ▶ model
     a semicolon-separated list of names of models to be imported from a file set. If all models must be imported, specify model=all. If parameter where is not specified, parameter model must be specified.
     Type: NVARCHAR(ANY)
  ▶ where
a WHERE clause to filter the set of models to be imported. All columns from view V_NZA_MODELS can be used in the WHERE clause. If parameter model is not specified, parameter where must be specified.

Type: NVARCHAR(ANY)

- **directory**
  the name of the directory where all files of the file set are stored. An absolute path name must be specified.

Type: NVARCHAR(ANY)

- **name**
  the name of the file set. This name is used as prefix for each file of the file set. If the directory contains only one file set, this parameter does not need to be specified.

Type: NVARCHAR(ANY)

- **acl**
  a flag ("access control list") controlling import of privileges. Only the administrator can use this parameter. If set to true, model privileges are imported also (if they exist in the file set).

Type: BOOLEAN

- **owner**
  the name of the owner of all imported models. If no name is specified, the current user is the new owner of all models. This parameter can be specified by the database administrator only.

Type: NVARCHAR(ANY)

- **overwrite**
  a flag used to overwrite existing models. If a model with the same name as an imported model already exists in the database, import of this model is possible only if overwrite is set to true. In this case, existing models are deleted first if name conflicts appear.

Type: BOOLEAN

- **verbose**
  a flag controlling if additional output about the progress of the import process is printed.

Type: BOOLEAN

▲ Returns
INTEGER the number of successfully imported models

**Details**

The parameters model and where are used to specify the models to be imported. If both parameters are specified, the two model sets are merged. To import all models, pass model=all (the where parameter is ignored in this case). The specified directory must contain an export file set generated by the procedure EXPORT_MODEL before. The database process must be able to read the files in this directory. If the directory contains more than one export file set, the parameter name must be used to select the desired file set for import. If a model to be imported has the same name as an existing model, the model is not imported. However, if the overwrite parameter is set to true and the user has the privilege to delete the existing model, the imported model overwrites the existing model. If a model cannot be imported (for example, an existing model cannot be overwritten), the import of this model is skipped and the system continuous the import with the next model. Only on severe errors the import process is aborted completely. Only the data-
base administrator can import the model privileges (which user and which group has which access
right on a model), using the acl parameter. Of course model privileges must have been exported
before, else this parameter has no effect. By default, the current user is the new owner of all im-
ported models and gets the corresponding privileges on the models. Only the administrator can
specify another (existing) owner for all imported models.

Examples

\c EXDB

CALL
nza..TIMESERIES('model=tmodel,intable=nza..iris,time=ID,
target=PETALLENGTH,by=CLASS,algorithm=esmoothing');

CALL
nza..EXPORT_MODEL('model=tmodel,directory=/tmp/export,na-
me=tsexport,acl=true');

CALL nza..DROP_MODEL('model=tmodel');
\c IMDB

CALL
nza..IMPORT_MODEL('model=tmodel,directory=/tmp/export,ac-
cl=true');

You are now connected to database EXDB.

TIMESERIES

------------

3

(1 row)

EXPORT_MODEL

------------

1

(1 row)

NOTICE:  Dropped: TSMODEL

DROP_MODEL

------------

1

(1 row)

You are now connected to database IMDB.

IMPORT_MODEL
Related Functions

- category Analytics - Model Management
- LIST_MODELS
- EXPORT_MODEL

**IMPUTE_DATA - Impute missing data**

This stored procedure replaces missing values in the input data.

**Usage**

The IMPUTE_DATA stored procedure has the following syntax:

```
IMPUTE_DATA(NVARCHAR(ANY) paramString)
```

**Parameters**

- **paramString**
  
  comma-separated list of <parameter>=<value> entries with parameters below
  
  Type: NVARCHAR(ANY)

- **intable**

  the input table
  
  Type: NVARCHAR(256)

- **method**

  the data imputation method. Allowed values are: mean, median, freq (most frequent value), replace. If not specified, the method is median for the numeric columns and freq for the nominal columns. The methods mean and median cannot be used with nominal columns.
  
  Type: NVARCHAR(ANY)

  Default: <none>

- **outable**

  the output table with the replaced missing values. If not specified, the missing values are replaced in the input table directly.
  
  Type: NVARCHAR(256)

  Default: <none>

- **inColumn**

  the input table column where missing values have to be replaced. If not specified, all input data columns are considered.
  
  Type: NVARCHAR(128)
Default: <none>

- **numericValue**
  the numeric replacement value when method=replace
  
  Type: DOUBLE
  Default: -1

- **nominalValue**
  the nominal replacement value when method=replace
  
  Type: NVARCHAR(ANY)
  Default: missing

▲ Returns
INTEGER the number of attributes where data was updated

Details
This stored procedure replaces missing values in the specified column or, if no column is specified, in all columns of the input table. If no replacement method is specified, missing values in a numeric column are replaced by the median value of this column, while missing values in a nominal column are replaced by the most frequent value. If two values have the same frequency, the alphabetically first value is used.

The input data can be modified in place or copied then modified into the output table. Imputation of missing data is helpful for most of the data mining procedures.

Examples

```sql
CREATE TABLE miss(id int, col float, attr float, data datetime, class NVARCHAR(10));

INSERT INTO miss VALUES (1,1.4,null,'2008-01-01','white');
INSERT INTO miss VALUES (2,2.5,random(),null,null);
INSERT INTO miss VALUES (3,null,random(),'2008-01-01','white');
INSERT INTO miss VALUES (4,1.4,random(),null,'black');
CALL nza..IMPUTE_DATA('intable=miss');
CALL nza..DROP_TABLE('miss');
```

```
IMPUTE_DATA
-------------
4

(1 row)
DROP_TABLE
```
Related Functions

- category Analytics - Data Transformation
- KNN
- NAIVEBAYES

INITIALIZE - Initialize the model management infrastructure

This stored procedure creates tables, views, a sequence and a stored procedure needed for the analytics model management in the current database. If the tables, views and sequence already exist, nothing is done.

Usage

The INITIALIZE stored procedure has the following syntax:

```sql
INITIALIZE(NVARCHAR(ANY) paramString)
```

Parameters

- **paramString**
  The strings "help", "noschema", or "force". "Noschema" instructs the procedure to move back the metadata from the INZA schema to the default schema (as preparation for disablement of full schema mode). "Force" instructs the procedure to always recreate the metadata views (needed after database backup was restored to a different database).
  Type: NVARCHAR(ANY)

Returns

NVARCHAR(ANY) a text explaining whether the infrastructure already exists, or has been created successfully

Details

This stored procedure creates the model management infrastructure into the current database if not already done. If full schema mode is enabled, the schema INZA is used, if it is disabled, we have no choice of a schema. The following database objects are created:

- model management tables: NZA_META_MODELS, NZA_META_COMPONENTS, NZA_META_PARAMS, NZA_META_COLPROPS
- model management views: V_NZA_MODELS, V_NZA_COMPONENTS, V_NZA_PARAMS, V_NZA_COLPROPS
- model management sequence: NZA_META_IDSEQUENCE
- model management procedure: METADATA_VERSION

The tables, the sequence, and the procedure are not public, the views are a public interface. The views return only records that belong to models where the user has the LIST privilege.
Examples

CALL nza..CLEANUP();
CALL nza..INITIALIZE();
CALL nza..INITIALIZE();

CLEANUP

------------------------------------------------------
The metadata objects have been removed successfully.
(1 row)

INITIALIZE

------------------------------------------------------
The metadata objects are successfully initialized.
(1 row)

INITIALIZE

--------------------------------------------------------
----------
Nothing to be done. The metadata objects are already initialized.
(1 row)

Related Functions

► category Analytics - Model Management
► IS_INITIALIZED

IQR - InterQuartile Range

This stored procedure calculates the interquartile range of a numeric column

Usage

The IQR stored procedure has the following syntax:

► IQR(NVARCHAR(ANY) paramString)
   ^ Parameters
   ► paramString
      comma-separated list of <parameter>=<value> entries with parameters below
      Type: NVARCHAR(ANY)
**intable**
the input table or view
Type: NVARCHAR(256)

**incolumn**
the input table column
Type: NVARCHAR(128)

Returns
REAL the interquartile range value

**Details**
This stored procedure calculates the width of the interquartile range, i.e. Q3-Q1. Q1 is the first quartile containing 25% of the population, and Q3 is the third quartile containing 75% of the population.

**Examples**
```sql
CALL nza..IQR('intable=nza..quant_iris, incolumn=sepal_length');
```
```
IQR
-----
  1.3
(1 row)
```

**Related Functions**
- category Analytics - Quantiles & Outliers
- QUARTILE
- QUARTILE_DISC

---

**IS_INITIALIZED - Check the model management infrastructure**

This stored procedure checks if the model management infrastructure is initialized, and if it needs to be migrated. If no database is given, the current database is assumed.

**Usage**
The IS_INITIALIZED stored procedure has the following syntax:

```sql
IS_INITIALIZED(NVARCHAR(ANY))
```

**Parameters**

- **paramString**
  comma-separated list of <parameter>=<value> entries with parameters below
  Type: NVARCHAR(ANY)
The **database** parameter specifies the database where to check the model management infrastructure. If not specified, the current database is used.

- **Type:** NVARCHAR(128)
- **Default:** <none>

**Returns**

NVARCHAR(ANY) a text explaining that the metadata tables are initialized (otherwise an exception is raised)

**Details**

This stored procedure checks if the model management infrastructure is initialized with the expected version.

- If the model management tables, views and sequence does not exist, the model management infrastructure has to be initialized.
- If they exist but their version is older than expected, the model management infrastructure has to be migrated to a newer version.
- Otherwise, nothing has to be done.

This stored procedure can check the model management infrastructure in a remote database using the `database` parameter. To check the current database, the stored procedure can be called with a NULL parameter, an empty parameter or without parameter.

**Examples**

```sql
CALL nza..IS_INITIALIZED();

IS_INITIALIZED
-------------------------------
The metadata tables are initialized.
(1 row)
```

**Related Functions**

- category Analytics - Model Management
- INITIALIZE
- REGISTER_MODEL

**JOINT_ENTROPY - Bivariate Entropy**

This stored procedure calculates the joint entropy of two columns.
Usage
The JOINT_ENTROPY stored procedure has the following syntax:

► JOINT_ENTROPY(NVARCHAR(ANY) paramString)

▲ Parameters

► paramString
comma-separated list of <parameter>=<value> entries with parameters below
Type: NVARCHAR(ANY)

► intable
the input table
Type: NVARCHAR(256)

► incolumn
the two input table columns separated by a semicolon (;)
Type: NVARCHAR(ANY)

► by
the input table column which splits the data into groups for which the operation is to be performed. If specified, an output table must be indicated.
Type: NVARCHAR(128)
Default: <none>

► outtable
the output table to write the entropy into. This parameter must be specified if parameter by is specified.
Type: NVARCHAR(256)
Default: <none>

▲ Returns
DOUBLE the joint entropy of both input table columns, or the number of groups in the output table for which entropy has been calculated

Details
This stored procedure calculates the entropy of two input columns, either in the whole input table or within the groups defined in the column specified by parameter <by>. Bivariate Entropy is a measure saying how evenly the cases are distributed for the values of both input columns. It takes a positive value, 0 means that the column contains only one value.

If parameter 'by' is specified, an output table must be specified. The output table is created with following columns: over_<by>, jointentropy. The joint entropy of both input columns is given for each group of the <by> column.

Examples

CALL nza..JOINT_ENTROPY('intable=nza..CensusIncome, incolumn=age:wage_per_hour');

JOINT_ENTROPY
CALL nza..JOINT_ENTROPY('intable=nza..CensusIncome,
incolumn=age;wage_per_hour, outtable=resultTable,
by=sex');
SELECT * FROM resultTable order by over_sex;
CALL nza..DROP_TABLE('resultTable');

JOIN_ENTROPY
------------
2
(1 row)

<table>
<thead>
<tr>
<th>JOINTENTROPY</th>
<th>OVER_SEX</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.9578241327179</td>
<td>Female</td>
</tr>
<tr>
<td>6.8873765557205</td>
<td>Male</td>
</tr>
</tbody>
</table>
(2 rows)

DROP_TABLE
----------
 t
(1 row)

Related Functions
► category Analytics - Statistics
► COND_ENTROPY
► ENTROPY
► MUTUALINFO
KMEANS - Build a K-means Clustering model

This stored procedure builds a Clustering model that clusters the input data into k centers. The centers are calculated as the mean value of the nearest input data records.

Usage

The KMEANS stored procedure has the following syntax:

```sql
KMEANS(NVARCHAR(ANY) paramString)
```

Parameters

- **paramString**
  comma-separated list of <parameter>=<value> entries with parameters below
  Type: NVARCHAR(ANY)

- **model**
  the name of the Hierarchical Clustering model to build
  Type: NVARCHAR(ANY)

- **intable**
  the input table
  Type: NVARCHAR(256)

- **outtable**
  the output table where clusters are assigned to each input table record
  Type: NVARCHAR(ANY)

- **id**
  the input table column identifying a unique instance id
  Type: NVARCHAR(128)

- **target**
  the input table column representing a class or a value to predict, this column is ignored by the Hierarchical Clustering algorithm.
  Type: NVARCHAR(128)
  Default: <none>

- **distance**
  the distance function. Allowed values are: euclidean, norm_euclidean, manhattan, canberra, maximum, mahalanobis.
  Type: NVARCHAR(ANY)
  Default: norm_euclidean

- **k**
  number of centers
  Type: INTEGER
  Default: 3

- **maxiter**
the maximum number of iterations to perform
Type: INTEGER
Default: 5
Min: 1
Max: 1000

► randseed
the random generator seed
Type: INTEGER
Default: 12345

► idbased
the specification that random generator seed is based on id column value
Type: BOOL
Default: false

► incolumn
the input table columns with special properties, separated by a semi-colon (;).
Each column is followed by one or several of the following properties:
- its type: ':nom' (for nominal), ':cont' (for continuous). Per default, all numerical types
  are continuous, other types are nominal.
- its role: ':id', ':target', ':input', ':ignore'.
  (Remark: ':objweight' is unsupported, i.e. ':objweight' same as ':ignore').
  (Remark: ':colweight(<wgt>)' is unsupported, i.e. ':colweight(<wgt>)' same as ':colweight(1)' same as ':input').
If the parameter is undefined, all columns of the input table have default properties.
Type: NVARCHAR(ANY)
Default: <none>

► coldeftype
default type of the input table columns. Allowed values are 'nom' and 'cont'.
If the parameter is undefined, all numeric columns are considered continuous, other
columns nominal.
Type: NVARCHAR(ANY)
Default: <none>

► coldefrole
default role of the input table columns. Allowed values are 'input' and 'ignore'.
If the parameter is undefined, all columns are considered 'input' columns.
Type: NVARCHAR(ANY)
Reference Documentation: Analytics

Default: <none>

► **colPropertiesTable**
the input table where column properties for the input table columns are stored. The format of this table is the output format of stored procedure nza..COLUMN_PROPERTIES().
If the parameter is undefined, the input table column properties will be detected automatically.
(Remark: colPropertiesTable with "COLROLE" column with value 'objweight' is unsupported, i.e. same as 'ignore')
(Remark: colPropertiesTable with "COLWEIGHT" column with value '<wgt>' is unsupported, i.e. same as '1')
Type: NVARCHAR(256)
Default: <none>

► **statistics**
flags indicating which statistics to collect. Allowed values are: none, columns, values:n, all.
If statistics=none, no statistics are collected.
If statistics=columns, statistics on the input table columns like mean value are collected.
If statistics=values:n with n a positive number, statistics about the columns and the column values are collected. Up to <n> column value statistics are collected:
- If a nominal column contains more than <n> values, only the <n> most frequent column statistics are kept.
- If a numeric column contains more than <n> values, the values will be discretized and the statistics will be collected on the discretized values.
Indicating statistics=all is equal to statistics=values:100.
Type: NVARCHAR(ANY)
Default: 'none'

► **transform**
flag indicating if the input table columns have to be transformed. Allowed values are: L (for leave as is), N (for normalization) or S (for standardization). If it is not specified, no transformation will be performed.
Type: NVARCHAR(ANY)
Default: L

▲ Returns
BIGINT the number of generated clusters

Details
This stored procedure builds a K-means Clustering model. The model is saved to the database in a set of tables and registered in the database model metadata. Use the Model Management functions to further manipulate the model or access the model tables.
The clusters are build iteratively by partitioning data into k (or less) separate clusters according to their distance to the cluster center, then re-calculating the cluster centers. The K-means Clustering algorithm stops after <maxiter> iterations.
The output table <outtable> is created with following columns: id, cluster_id, distance. The id column matches the <id> column of the input table. Each input table record is associated with a cluster, where the distance from the record to the cluster center is the smallest. The cluster ID and the distance to the cluster center are given in the columns cluster_id and distance.

Examples

```sql
CALL nza..KMEANS('model=adult_mdl, intable=nza..adult, outtable=adult_out, id=id, target=income, transform=S, distance=euclidean, k=3, maxiter=5, randseed=12345, idbased=false');
CALL nza..DROP_MODEL('model=adult_mdl');
CALL nza..DROP_TABLE('adult_out');
```

KMEANS
-------
3
(1 row)

DROP_MODEL
------------
1
(1 row)

DROP_TABLE
------------
1
(1 row)

Related Functions
► category Analytics - Clustering
► PREDICT_KMEANS
► LIST_MODELS

KNN - Build a K-Nearest Neighbors model

This stored procedure builds a K-Nearest Neighbors Classification or Regression model

Usage

The KNN stored procedure has the following syntax:
KNN(NVARCHAR(ANY) paramString)

Parameters

- **paramString**
  comma-separated list of <parameter>=<value> entries with parameters below
  Type: NVARCHAR(ANY)

- **model**
  the name of the KNN model to build
  Type: NVARCHAR(ANY)

- **intable**
  the input table
  Type: NVARCHAR(256)

- **id**
  the input table column identifying a unique instance id
  Type: NVARCHAR(128)

- **target**
  the input table column representing the class
  Type: NVARCHAR(128)

- **incolumn**
  the input table columns with special properties, separated by a semi-colon (;).
  Each column is followed by one or several of the following properties:
  - its type: 'nom' (for nominal), 'cont' (for continuous). Per default, all numerical types are continuous, other types are nominal.
  - its role: ':id', ':target', ':input', ':ignore'.
    (Remark: ':objweight' is unsupported, i.e. ':objweight' same as ':ignore').
    (Remark: ':colweight(<wgt>)' is unsupported, i.e. ':colweight(<wgt>)' same as ':colweight(1)' same as ':input').
  If the parameter is undefined, all columns of the input table have default properties.
  Type: NVARCHAR(ANY)
  Default: <none>

- **coldeftype**
  default type of the input table columns. Allowed values are 'nom' and 'cont'.
  If the parameter is undefined, all numeric columns are considered continuous, other columns nominal.
  Type: NVARCHAR(ANY)
  Default: <none>

- **coldefrole**
  default role of the input table columns. Allowed values are 'input' and 'ignore'.
If the parameter is undefined, all columns are considered 'input' columns.

Type: NVARCHAR(ANY)
Default: <none>

► colPropertiesTable
the input table where column properties for the input table columns are stored. The format of this table is the output format of stored procedure nza..COLUMN_PROPERTIES().

If the parameter is undefined, the input table column properties will be detected automatically.

(Remark: colPropertiesTable with "COLROLE" column with value 'objweight' is unsupported, i.e. same as 'ignore')

(Remark: colPropertiesTable with "COLWEIGHT" column with value '<wgt>' is unsupported, i.e. same as '1')

Type: NVARCHAR(256)
Default: <none>

▲ Returns
BIGINT the number of sample data records in the model

Details
This stored procedure builds a K-Nearest Neighbors Classification or Regression model. The model simply contains a sample among the input table records that will be used later for prediction.

Examples
CALL nza..KNN('model=iris_mdl, intable=nza..iris, id=id, target=class');

CALL nza..DROP_MODEL('model=iris_mdl');

KNN
-----
150
(1 row)

DROP_MODEL
-------------
t
(1 row)
KURTOSIS_AGG - excess Kurtosis

This function calculates the excess Kurtosis value of a single numeric variable, as the fourth central moment divided by the square of the variance of the probability distribution minus 3.

Usage

The KURTOSIS_AGG aggregate has the following syntax:

KURTOSIS_AGG(DOUBLE X)

Parameters

- X
  the input variable
  Type: DOUBLE

Returns

DOUBLE the excess Kurtosis value of the input variable

Details

This function calculate the excess Kurtosis value of a single numeric variable. The kurtosis is a measure of the thickness of the tails of the variable distribution. If the kurtosis is negative, the tails are thinner or lighter than in a normal distribution. If the kurtosis is positive, the tails are thicker or heavier than in a normal distribution.

The excess Kurtosis is calculated as the fourth moment of the variable around its mean divided by the square of the variance of the probability distribution minus 3. Note that the normal distribution, due to the subtraction of 3, has a kurtosis of zero. A k-th order central moment is the sum of the k-th powers of differences between the mean and the actual value divided by the number of cases. Because of the fact that we want to know the moments for a population but we are computing them from the given sample values, diverse corrections are taken into account. This leads to many formulas for assessing the Kurtosis. In this implementation we followed a "midway" policy dividing the estimated central moments by the number of cases minus 1. This policy is a generally established practice for variance but not for Kurtosis. Therefore the implementations match each other for a large number of cases only. Our decision to use our approach was based on the fact that an unbiased estimate is not available anyway.

If a value is missing, the whole row is ignored.

Examples

SELECT nza..KURTOSIS_AGG(petallength) FROM nza..iris;
Related Functions

- category Analytics - Statistics
- MOMENTS
- SKEWNESS_AGG
- SUMMARY1000

LDF_MANOVA_ONE_WAY_TEST - LDF of ABC Multivariate Analysis of Variance result in one way setting

This stored procedure prints the one way MANOVA linear discriminat function for the largest root,

Usage

The LDF_MANOVA_ONE_WAY_TEST stored procedure has the following syntax:

- LDF_MANOVA_ONE_WAY_TEST(NVARCHAR(ANY) paramString)

  ▲ Parameters

  ▶ paramString
  comma-separated list of <parameter>=<value> entries with parameters below
  Type: NVARCHAR(ANY)

  ▶ intable
  the input table name
  Type: NVARCHAR(ANY)

  ▶ id_task
  the id_task that will be presented, id_task must be positive integer >= 1.
  Type: integer

  ▲ Returns
  NVARCHAR(2000) A string containing the SQL form of LDF

Details

This stored procedure outputs in a SQL form the ldf of one-way analysis of variance/covariance aiming to tell whether or not the groups of data identified by factor1 have the same mean value in all dependent variables or not. Note that the one way analysis could have been performed for a multitude of tasks. Therefore id_task is needed.
Examples

```sql
create view irisX05 as
select *, mod(id,3) as factor from nza..iris;
CALL nza..MANOVA_ONE_WAY_TEST('intable=irisX05,id=Id, incolumn=
petallength;petalwidth;sepalwidth, factor1=class,
outtable=outtab05a, type=columns');
CALL nza..LDF_MANOVA_ONE_WAY_TEST('intable=outtab05a,id_task=1');
CALL nza..DROP_TABLE('outtab05a');
CALL nza..DROP_TABLE('irisX05');

DF_MANOVA_ONE_WAY_TEST
---------------------------------------------------------------
---------------------------------------------------------------
----------------------------
select <row id>
, + 0.63828142599355 * "PETALLENGTH"
+ 14.376609151486 * "PETALWIDTH"
+ 10.766016415897 * "SEPALWIDTH"
as factor1
from <table name>
(1 row)
```

Related Functions

- category Analytics - Statistics
- MANOVA_ONE_WAY_TEST
- PRINT_MANOVA_ONE_WAY_TEST

**LDF_MANOVA_TWO_WAY_TEST** - LDF of Multivariate Analysis of Variance result in TWO way setting

This stored procedure prints the TWO way MANOVA linear discriminat function for the largest root,

**Usage**

The LDF_MANOVA_TWO_WAY_TEST stored procedure has the following syntax:

```sql
LDF_MANOVA_TWO_WAY_TEST(NVARCHAR(ANY) paramString)
```

▲ Parameters
paramString
comma-separated list of <parameter>=<value> entries with parameters below
Type: NVARCHAR(ANY)

intable
the input table name
Type: NVARCHAR(ANY)

id_task
the id_task that will be presented, id_task must be positive integer >= 1.
Type: integer

Returns
NVARCHAR(2000) A string containing the SQL form of LDF

Details
This stored procedure outputs in a SQL form the LDF of TWO-way analysis of variance/covariance aiming to tell whether or not the groups of data identified by factor1 have the same mean value in all dependent variables or not. Note that the TWO way analysis could have been performed for a multitude of tasks. Therefore id_task is needed

Examples
create view irisX05 as
select *, mod(id,3) as factor from nza..iris;

CALL nza..MANOVA_TWO_WAY_TEST('intable=irisX05,id=Id,
incolumn=sepalwidth;petallength;sepallength,
factor1=factor,factor2=class, outtable=outtab05a,
type=columns');

CALL nza..LDF_MANOVA_TWO_WAY_TEST('intable=outtab05a,id_task=1');

CALL nza..DROP_TABLE('outtab05a');

CALL nza..DROP_TABLE('irisX05');

LDF_MANOVA_TWO_WAY_TEST
LDF_MANOVA_TWO_WAY_TEST
LDF_MANOVA_TWO_WAY_TEST
LDF_MANOVA_TWO_WAY_TEST
LDF_MANOVA_TWO_WAY_TEST
LDF_MANOVA_TWO_WAY_TEST
select <row id>
, + 0.98933945482201 * "SEPALWIDTH"
+ 0.0074007470715444 * "PETALLENGTH"
+ 0.0088047416599598 * "SEPALLENGTH"
  as factor1
, + -0.071356687984851 * "SEPALWIDTH"
+ 9.0908512260774 * "PETALLENGTH"
+ 3.7847858074782 * "SEPALLENGTH"
  as factor2
, + -0.81232580404822 * "SEPALWIDTH"
+ 0.010025802114667 * "PETALLENGTH"
+ 0.070947392958058 * "SEPALLENGTH"
  as factorInteract
from <table name>
(1 row)

Related Functions

► category Analytics - Statistics
► MANOVA_Two WAY_TEST
► PRINT_MANOVA_Two WAY_TEST

LINEAR_REGRESSION - Build a Linear Regression model

This stored procedure builds a Linear Regression model

Usage

The LINEAR_REGRESSION stored procedure has the following syntax:

► LINEAR_REGRESSION(NVARCHAR(ANY) paramString)
  ▲ Parameters
  ► paramString
    comma-separated list of <parameter>=<value> entries with parameters below
      Type: NVARCHAR(ANY)
  ► model
    the Linear Regression model to build
IBM Netezza In-Database Analytics Reference Guide

- **intable**
  the input table
  Type: NVARCHAR(256)

- **id**
  the input table column identifying a unique instance id
  Type: NVARCHAR(128)

- **target**
  the input table column representing the prediction target, definition of multitects can be processed by 'incolumn' parameter and column properties.
  Type: NVARCHAR(128)

- **nominalCols**
  the input table nominal columns, if any, separated by a semi-colon (;). Parameter 'nominalCols' is deprecated please use 'incolumn' instead.
  Type: NVARCHAR(ANY) (default) <none>

- **incolumn**
  the input table columns with special properties, separated by a semi-colon (;).
  Each column is followed by one or several of the following properties:
  - its type: ':nom' (for nominal), ':cont' (for continuous). Per default, all numerical types are continuous, other types are nominal.
  - its role: ':id', ':target', ':input', ':ignore'.
  (Remark: ':objweight' is unsupported, i.e. ':objweight' same as ':ignore').
  (Remark: ':colweight(<wgt>)' is unsupported, i.e. ':colweight(<wgt>)' same as ':colweight(1)' same as ':input').
  If the parameter is undefined, all columns of the input table have default properties.
  Type: NVARCHAR(ANY)
  Default: <none>

- **coldeftype**
  default type of the input table columns. Allowed values are 'nom' and 'cont'.
  If the parameter is undefined, all numeric columns are considered continuous, other columns nominal.
  Type: NVARCHAR(ANY)
  Default: <none>

- **coldefrole**
  default role of the input table columns. Allowed values are 'input' and 'ignore'.
  If the parameter is undefined, all columns are considered 'input' columns.
  Type: NVARCHAR(ANY)
Default: <none>

- **colPropertiesTable**
  the input table where column properties for the input table columns are stored. The format of this table is the output format of stored procedure nza..COLUMN_PROPERTIES().
  If the parameter is undefined, the input table column properties will be detected automatically.
  (Remark: colPropertiesTable with "COLROLE" column with value 'objweight' is unsupported, i.e. same as 'ignore')
  (Remark: colPropertiesTable with "COLWEIGHT" column with value '<wgt>' is unsupported, i.e. same as '1')
  Type: NVARCHAR(256)
  Default: <none>

- **useSVDSolver**
  a flag indicating whether Singular Value Decomposition and matrix multiplication should be used for solving the matrix equation
  Type: BOOLEAN
  Default: false

- **intercept**
  flag indicating whether the model is built with or without an intercept value. Parameter 'includeIntercept' is deprecated.
  Type: BOOLEAN
  Default: false

- **calculateDiagnostics**
  a flag indicating whether diagnostics information should be displayed
  Type: BOOLEAN
  Default: false

▲ Returns
  BOOLEAN always true

**Details**

This stored procedure builds the linear regression model using the QR solver of a non-singular model matrix, or the Moore-Penrose pseudoinversion in the case of a near-singular or exactly singular model matrix. Input data should be provided as a table with observations provided in rows, and predictors in columns. The list of predicted values may consist of more than 1 attribute, or a nominal attribute, in that case a model with multiple output variables will be created.

For each of the nominal attributes specified, the stored procedure will create the corresponding dictionary, encoding the mapping between the provided values and the model coefficients (each value of a given nominal attribute introduces a new coefficient into the model).

If requested, diagnostic information is displayed, as well as the set of matrices created for the model:
- `<model>_linearmodel_R2` - row vector containing $R^2$ (being a fraction of variance explained by the model) of models created for each output variable (when calculateDiagnostics is TRUE),
- `<model>_linearmodel_RSS` - row vector containing Residual Sum of Squares of models created for each output variable (when calculateDiagnostics is TRUE),
- `<model>_linearmodel_SDEV` - the matrix of standard deviations of model coefficients (when calculateDiagnostics is TRUE, diagnostics is possible and model is overdetemined),
- `<model>_linearmodel_TVAL` - the matrix of the test statistics for the models' coefficients (when calculateDiagnostics is TRUE, diagnostics is possible and model is overdetemined),
- `<model>_linearmodel_PVAL` - the matrix of the two--sided p-values for the models' coefficients (when calculateDiagnostics is TRUE, diagnostics is possible and model is overdetemined),
- `<model>_linearmodel_Y_VAR_EST` - the row vector containing the estimators of a variance of error term for each predicted variable (when calculateDiagnostics is TRUE, diagnostics is possible and model is overdetemined),
- `<model>_linearmodel` - the matrix (as table) containing model coefficients (when model can be constructed).

The model table contains following columns:
- `VAR_ID` - contains the identifier of the predictive attribute,
- `VAR_NAME` - contains the name of the predictive attribute,
- `LEVEL_ID` - contains the code of given nominal attribute level,
- `LEVEL_NAME` - contains the value of given nominal attribute level,
- `PREDICTED_ID` - contains the identifier of predicted attribute (useful in case of models with multiple predicted values),
- `PREDICTED_NAME` - contains the name of predicted attribute (useful in case of models with multiple predicted values),
- `PREDICTED_LEVEL_ID` - contains the code of given nominal attribute level (for given predicted value),
- `PREDICTED_LEVEL_NAME` - contains the value of given nominal attribute level (for given predicted value),
- `VALUE` - contains the value of the model coefficient unically identified by the values of columns described above,
- `ST_DEV` - contains the standard deviation of the model coefficient (when calculateDiagnostics is TRUE, diagnostics is possible and model is overdetemined, -1 otherwise),
- `TVAL` - contains the value of the the statistics for given model coefficient (when calculateDiagnostics is TRUE, diagnostics is possible and model is overdetemined, -1 otherwise),
- `PVAL` - contains the p-value of the two--sided test for given model coefficient (when calculateDiagnostics is TRUE, diagnostics is possible and model is overdetemined, -1 otherwise).

Additionally, for each output variable encoded using the convention described above, the additional rows will be present in the table, containing the following information:
- Intercept of the model (when includeIntercept=TRUE) (row with VAR_NAME='(Intercept)', VAR_ID=-1),
- $R^2$ of the model (row with VAR_NAME='[R^2]', VAR_ID=-2, ST_DEV = TVAL = PVAL = -1),
- RSS of the model (Residual Sum of Squares) (row with VAR_NAME='[RSS]', VAR_ID=-3, ST_DEV = TVAL = PVAL = -1),
- estimators of a variance of error term (row with VAR_NAME='[Y_VAR_EST]', VAR_ID=-4, ST_DEV = TVAL = PVAL = -1).

The constructed model can be applied to the data using the PREDICT_LINEAR_REGRESSION procedure. Note that using the Singular Value Decomposition and matrix multiplication (useSVDSolver=true) could be slower than the standard calculation, but is more stable in the case of an ill-posed, that is, near colinear, regression model.

**Examples**

```sql
CREATE TABLE adultT1 AS SELECT id, age AS v1, fnlwgt AS fnlwgt, education_num AS v3 FROM nza..adult;

CALL nza..LINEAR_REGRESSION('model=modelA, intable=adultT1, id=id, target=fnlwgt, calculateDiagnostics=true');

CALL nza..DROP_MODEL('model=modelA');

CALL nza..DROP_TABLE('adultT1');

LINEAR_REGRESSION

-------------------

 t
(1 row)

DROP_MODEL

-------------

 t
(1 row)

DROP_TABLE

--------------

 t
(1 row)

CREATE TABLE adultT2 AS SELECT id, age, education_num, income, sex FROM nza..adult;

CALL nza..LINEAR_REGRESSION('model=modelC, intable=adultT2, id=id, target=age, nominalCols=income;sex, calculateDiagnostics=true');

CALL nza..DROP_MODEL('model=modelC');

CALL nza..DROP_TABLE('adultT2');
```
**Related Functions**

► category Analytics - Regression
► PREDICT_LINEAR_REGRESSION

**LIST_COLPROPS - List column properties for selected or all analytics models**

This stored procedure lists the column properties for all models for which the user has the LIST privilege. A WHERE clause can be used to list only the column properties of specific models.

**Usage**

The LIST_COLPROPS stored procedure has the following syntax:

► **LIST_COLPROPS(NVARCHAR(ANY))**
  ▲ Parameters
  ► paramString
    comma-separated list of <parameter>=<value> entries with parameters below
    Type: NVARCHAR(ANY)
  ► where
    a WHERE clause to limit the set of models whose column properties are listed. All columns from views V_NZA_MODELS and V_NZA_COLPROPS can be used in the WHERE clause.
    Type: NVARCHAR(ANY)
Returns

INTEGER the number of column properties listed

Examples

CALL nza..ARULE('intable=nza..retail, model=mbamodel, supporttype=percent, support=5, lvl=0, maxsetsize=5, confidence=0.5');
CALL nza..LIST_COLPROPS('where=upper(MODELNAME) LIKE ''MBA%''');
CALL nza..DROP_MODEL('model=mbamodel');

NOTICE:

RUNNING FPGrowth algorithm:
DATASET : "NZA".."RETAIL"
Transaction column : "TID"
Item column : "ITEM"
Group by : <none>
Minimum support : 5 %
Minimum confidence: 0.5
Max frequent itemset size : 5
Level of conditional dbs : 0
Result tables prefix : "NZA_META_MBAMODEL"

ARULE
-------
14
(1 row)

NOTICE:

MODELNAME | COLUMNNAME | PROPERTYNAME | PROPERTYTYPE
| PROPERTYVALUE
+---------------------------------+---------------
MBAMODEL | ITEM | COLDATATYPE | NATIONAL CHARACTER VARYING(128) | INTEGER
MBAMODEL | ITEM | COLROLE | NATIONAL CHARACTER VARYING(128) | input
MBAMODEL | ITEM | COLTYPE | NATIONAL CHARACTER
VARYING(128) | cont

<table>
<thead>
<tr>
<th>MBAMODEL</th>
<th>ITEM</th>
<th>COLWEIGHT</th>
<th>DOUBLE PRECISION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MBAMODEL</th>
<th>ITEM</th>
<th>IDCOL</th>
<th>INTEGER</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MBAMODEL</th>
<th>TID</th>
<th>COLDATATYPE</th>
<th>NATIONAL CHARACTER VARYING(128)</th>
<th>INTEGER</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>MBAMODEL</th>
<th>TID</th>
<th>COLROLE</th>
<th>NATIONAL CHARACTER VARYING(128)</th>
<th>input</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>MBAMODEL</th>
<th>TID</th>
<th>COLTYPE</th>
<th>NATIONAL CHARACTER VARYING(128)</th>
<th>cont</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>MBAMODEL</th>
<th>TID</th>
<th>COLWEIGHT</th>
<th>DOUBLE PRECISION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MBAMODEL</th>
<th>TID</th>
<th>IDCOL</th>
<th>INTEGER</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

LIST_COLPROPS
---------------
10

(1 row)

NOTICE: Dropped: MBAMODEL
DROP_MODEL
---------------
t
(1 row)

Related Functions
- category Analytics - Model Management
- LIST_MODELS

LIST_COMPONENTS - List components of selected or all analytics models
This stored procedure lists the components belonging to all models for which the user has the LIST
privilege. A WHERE clause can be used to list only the components of specific models.

Usage

The LIST_COMPONENTS stored procedure has the following syntax:

- **LIST_COMPONENTS**(NVARCHAR(ANY))
  - Parameters
    - **paramString**
      comma-separated list of <parameter>=<value> entries with parameters below
      Type: NVARCHAR(ANY)
    - **where**
      a WHERE clause to limit the set of models whose components are listed. All columns from views V_NZA_MODELS and V_NZA_COMPONENTS can be used in the WHERE clause.
      Type: NVARCHAR(ANY)
  - Returns
    INTEGER the number of components listed

Details

This stored procedure lists the managed or referenced components that belong to an analytics model for which the user has the LIST privilege:
- tables,
- views,
- synonyms,
- matrices.

Examples

```sql
CALL nza..ARULE('intable=nza..retail, model=mbamod, supporttype=percent, support=5, lvl=0, maxsetsize=5, confidence=0.5');
CALL nza..LIST_COMPONENTS('where=upper(MODELNAME) LIKE ''MBA''');
CALL nza..DROP.MODEL('model=mbamod');
```

NOTICE:

- **RUNNING FPGrowth algorithm:**
- **DATASET**: "NZA"."RETAIL"
- **Transaction column**: "TID"
- **Item column**: "ITEM"
- **Group by**: <none>
- **Minimum support**: 5 %
Minimum confidence: 0.5
Max frequent itemset size: 5
Level of conditional dbs: 0
Result tables prefix: "NZA_META_MBAMODEL"

ARULE
-------
14
(1 row)

NOTICE:

<table>
<thead>
<tr>
<th>MODELNAME</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATABASE</td>
<td>TYPE</td>
</tr>
<tr>
<td>SEQID</td>
<td>-------+------------+-------------------</td>
</tr>
<tr>
<td>+------------+-------+------------+-------------------</td>
<td></td>
</tr>
<tr>
<td>+-------</td>
<td>-------</td>
</tr>
<tr>
<td>MBAMODEL</td>
<td>NZA_META_MBAMODEL_COLUMN_PROPERTIES</td>
</tr>
<tr>
<td>TEST_NZDOC</td>
<td>View</td>
</tr>
<tr>
<td>MBAMODEL</td>
<td>NZA_META_MBAMODEL_GROUP</td>
</tr>
<tr>
<td>TEST_NZDOC</td>
<td>Table</td>
</tr>
<tr>
<td>MBAMODEL</td>
<td>NZA_META_MBAMODEL_ITEM</td>
</tr>
<tr>
<td>TEST_NZDOC</td>
<td>Table</td>
</tr>
<tr>
<td>MBAMODEL</td>
<td>NZA_META_MBAMODEL_ITEMSET</td>
</tr>
<tr>
<td>TEST_NZDOC</td>
<td>Table</td>
</tr>
<tr>
<td>MBAMODEL</td>
<td>NZA_META_MBAMODEL_RULE</td>
</tr>
<tr>
<td>TEST_NZDOC</td>
<td>Table</td>
</tr>
<tr>
<td>MBAMODEL</td>
<td>RETAIL</td>
</tr>
</tbody>
</table>

LIST_COMPONENTS
-----------------
6
(1 row)
Related Functions

- category Analytics - Model Management
- LIST_MODELS

**LIST_MODELS - List selected or all analytics models**

This stored procedure lists all models for which the user has the LIST privilege. This stored procedure can also list all models that have been exported via EXPORT_MODEL. A WHERE clause can be used to list only specific models.

**Usage**

The LIST_MODELS stored procedure has the following syntax:

- **LIST_MODELS(NVARCHAR(ANY))**
  - Parameters
    - **paramString**
      comma-separated list of `<parameter>=<value>` entries with parameters below
      Type: NVARCHAR(ANY)
    - **directory**
      the name of a directory that contains the result file set of the EXPORT_MODEL command. If this parameter is specified, the exported models are listed. If this parameter is not specified, the models from the metadata repository are listed.
      Type: NVARCHAR(ANY)
    - **name**
      the name of the export result file set, as specified in EXPORT_MODEL. This parameter can only be specified if parameter directory is specified also. If the directory contains only one result file set, this parameter does not need to be specified.
      Type: NVARCHAR(ANY)
    - **where**
      a WHERE clause to limit the set of models that are listed. All columns from view V_NZA_MODELS can be used in the WHERE clause.
      Type: NVARCHAR(ANY)
Returns

INTEGER the number of models listed

Examples

CALL nza..ARULE('intable=nza..retail, model=mbamodel, supporttype=percent, support=5, lvl=0, maxsetsize=5, confidence=0.5');

CALL nza..LIST_MODELS('where=upper(MODELNAME) LIKE ''MBA %''');

CALL nza..EXPORT_MODEL('model=mbamodel, directory=/tmp/mbaexport, name=mba');

CALL nza..DROP_MODEL('model=mbamodel');

CALL nza..LIST_MODELS('directory=/tmp/mbaexport');

NOTICE:

RUNNING FPgrowth algorithm:

DATASET : "NZA"."RETAIL"

Transaction column : "TID"

Item column : "ITEM"

Group by : <none>

Minimum support : 5 %

Minimum confidence: 0.5

Max frequent itemset size : 5

Level of conditional dbs : 0

Result tables prefix : "NZA_META_MBA MODEL"

ARULE

-------

14

(1 row)

NOTICE:

MODELNAME | OWNER | CREATED | STATE | MININGFUNCTION | ALGORITHM | USERCATEGORY

+------------------+-----------+--------------+---------+-----------------+------------+--------------+

MBAMODEL | JOE | 2012-01-01 00:00:00 | Complete | associationRules | FPgrowth |
LIST_MODELS
-------------
  1
(1 row)
EXPORT_MODEL
-------------
  1
(1 row)
NOTICE: Dropped: MBAMODEL
DROP_MODEL
-------------
  t
(1 row)
NOTICE:

MODELNAME | OWNER | CREATED       | MININGFUNCTION | ALGORITHM | USERCATEGORY
-----------+-------+---------------------+------------------
+-----------+--------------
  MBAMODEL  | JOE   | 2012-01-01 00:00:00 | associationRules | FPGrowth  |

Related Functions
► category Analytics - Model Management

LIST_PARAMS - List parameters for selected or all analytics models

This stored procedure lists the parameters used to build the models for which the user has the LIST privilege. A WHERE clause can be used to list only the parameters of specific models.

Usage
The LIST_PARAMS stored procedure has the following syntax:
LIST_PARAMS(NVARCHAR(ANY))

Parameters

paramString

- comma-separated list of <parameter>=<value> entries with parameters below
  
  Type: NVARCHAR(ANY)

where

- a WHERE clause to limit the set of models whose parameters are listed. All columns from views V_NZA_MODELS and V_NZA_PARAMS can be used in the WHERE clause.
  
  Type: NVARCHAR(ANY)

Returns

- INTEGER the number of parameters listed

Examples

CALL nza..ARULE('intable=nza..retail, model=mbamodel, supporttype=percent, support=5, lvl=0, maxsetsize=5, confidence=0.5');

CALL nza..LIST_PARAMS('where=upper(MODELNAME) LIKE ''MBA%'');

CALL nza..DROP_MODEL('model=mbamodel');

NOTICE:

RUNNING FPGrowth algorithm:

DATASET : "NZA"..'RETAIL"

Transaction column : "TID"

Item column : "ITEM"

Group by : <none>

Minimum support : 5 %

Minimum confidence: 0.5

Max frequent itemset size : 5

Level of conditional dbs : 0

Result tables prefix : "NZA_META_MBAMODEL"

ARULE

-------

14

(1 row)
<table>
<thead>
<tr>
<th>MODELNAME</th>
<th>TASKSEQ</th>
<th>PARAMETERNAME</th>
<th>PARAMETERTYPE</th>
<th>PARAMETERVALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MBAMODEL</td>
<td>1</td>
<td>confidence</td>
<td>DOUBLE PRECISION</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>intable</td>
<td>NATIONAL CHARACTER VARYING(256)</td>
<td>nza..retail</td>
</tr>
<tr>
<td></td>
<td></td>
<td>item</td>
<td>NATIONAL CHARACTER VARYING(128)</td>
<td>item</td>
</tr>
<tr>
<td></td>
<td></td>
<td>lvl</td>
<td>INTEGER</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>maxSetSize</td>
<td>INTEGER</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>procedure</td>
<td>CHARACTER VARYING(64)</td>
<td>ARULE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>support</td>
<td>DOUBLE PRECISION</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>supportType</td>
<td>CHARACTER VARYING(64)</td>
<td>percent</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tid</td>
<td>NATIONAL CHARACTER VARYING(128)</td>
<td>tid</td>
</tr>
</tbody>
</table>

LIST_PARAMS
-------------
9

(1 row)

NOTICE: Dropped: MBAMODEL

DROP_MODEL
-------------
t
(1 row)
LIST_PRIVILEGES - List privileges on all analytics models

This stored procedure lists the effective privileges of a selected or all users on all models for which you have the LIST privilege.

Usage

The LIST_PRIVILEGES stored procedure has the following syntax:

```
LIST_PRIVILEGES(NVARCHAR(ANY))
```

Parameters

- **paramString**
  - comma-separated list of <parameter>=<value> entries with parameters below
  - Type: NVARCHAR(ANY)

- **user**
  - the user whose privileges are listed. If not specified, the privileges of all users are listed.
  - Type: NVARCHAR(ANY)

- **grant**
  - if true, the grant permissions are displayed. If false (default), the regular object permissions are displayed.
  - Type: NVARCHAR(ANY)BOOLEAN

Returns

INTEGER the number of privileges listed

Details

This stored procedure lists the effective privileges of all users on all models.
- You need SELECT privilege on the system view _V_SYS_PRIV to read the model privileges. Typically, only the administrator can grant this privilege to you.
- You need the LIST privilege on the model to read the model privileges. The model owner, the database owner or the administrator can grant the LIST privilege to you using the GRANT_MODEL stored procedure.

To get privileges for all users, the stored procedure can be called with a NULL parameter, an empty parameter or without parameter.

Examples

```
CALL nza..LIST_PRIVILEGES();
```
NOTICE:

<table>
<thead>
<tr>
<th>User Name</th>
<th>Model Name</th>
<th>L S A U D</th>
</tr>
</thead>
<tbody>
<tr>
<td>-----------+------------+-----------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Model Privileges</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(L)ist (S)elect (A)lter (U)pdate (D)rop</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LIST_PRIVILEGES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(l row)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Related Functions

- category Analytics - Model Management
- LIST_MODELS
- GRANT_MODEL

MAE - Mean Absolute Error

This stored procedure calculates the mean absolute error of Regression predictions

Usage

The MAE stored procedure has the following syntax:

MAE(NVARCHAR(ANY) paramString)

Parameters

- paramString
  - comma-separated list of <parameter>=<value> entries with parameters below
  - Type: NVARCHAR(ANY)

- pred_table
  - the input table or view containing predicted values
  - Type: NVARCHAR(256)

- pred_column
  - the input table column in <pred_table> containing predicted values
  - Type: NVARCHAR(128)

- pred_id
  - the input table column in <pred_table> identifying a unique instance id
  - Type: NVARCHAR(128)

- true_table
the input table or view containing real values
Type: NVARCHAR(256)

► true_column
the input table column in <true_table> containing real values
Type: NVARCHAR(128)

► true_id
the input table column in <true_table> identifying a unique instance id
Type: NVARCHAR(128)

► check
flag indicating to check parameters or not. This may consume some time but prevents
usage errors. Allowed values are: 'all' (check all), 'none' (check nothing), 'nulls' (check
for nulls only).
Type: NVARCHAR(ANY)
Default: all

▲ Returns
DOUBLE the Mean Absolute Error

Details
This stored procedure calculate the mean absolute error of Regression predictions. This is done by
comparing the predictions made when applying a Regression model onto data, and the real values
for this data.
The MAE value is calculated as sum(abs(true_i-pred_i))/count(true_i) where true_i is the real value
and pred_i is the predicted value.

Examples

    CALL nza..GROW_REGTREE('model=wrt, intable=nza..weatherr,
id=instance, target=grade, maxdepth=4, minsplit=2');
    CALL nza..PREDICT_REGTREE('model=wrt, intable=nza..weatherr, id=instance, outtable=wpr, var=TRUE');
    CALL nza..MAE('pred_table=wpr, true_table=nza..weatherr, pred_column=class, true_column=grade, pred_id=id, true_id=instance');
    CALL nza..DROP_MODEL('model=wrt');
    CALL nza..DROP_TABLE('wpr');

GROW_REGTREE
--------------
13
(1 row)

PREDICT_REGTREE
-----------------

22
(1 row)

MAE
-------------------

0.090909090909091
(1 row)

DROP_MODEL
------------

\[ t \]
(1 row)

DROP_TABLE
------------

\[ t \]
(1 row)

Related Functions

- category Analytics - Diagnostic Measures
- CERROR
- MSE
- RAE
- RSE

MANOVA_ONE WAY TEST - Multivariate Analyzsis of Variance in one way setting

This stored procedure performs a completely randomized design MANOVA
Usage
The MANOVA_ONE_WAY_TEST stored procedure has the following syntax:

- **MANOVA_ONE_WAY_TEST**(NVARCHAR(ANY) paramString)
  - **paramString**
    - comma-separated list of <parameter>=<value> entries with parameters below
    - Type: NVARCHAR(ANY)
  - **intable**
    - the input table name
    - Type: NVARCHAR(ANY)
  - **outtable**
    - the output table name
    - Type: NVARCHAR(ANY)
  - **factor1**
    - the input table column identifying a first factor (so-called treatment in RBD/CRD nomenclature) used for type='column' only for type='trcv' these are the values for column equal 1
    - Type: NVARCHAR(ANY)
  - **incolumn**
    - the input table observation columns (dependent variables), separated by a semi-colon (;). used for type='column' only for type='trcv' these are the values for column greater equal 2
    - Type: NVARCHAR(ANY)
  - **type**
    - the input table form: either 'columns' or 'trcv' trcv stands for "id_task, row, column, value" id_task must be positive integer >= 1. columns means the traditional table representation
    - Type: NVARCHAR(ANY)
    - Default: 'trcv'
  - **id**
    - the input table column which uniquely identifies records used for type='column' only for type='trcv' not needed due to the structure of the table
    - Type: NVARCHAR(ANY)
  - **by**
    - the input table column which splits the input table into subtables, on each of them a separate MANOVA is run If not specified, the whole input table is subject of a single MANOVA run. used for type='column' only for type='trcv' it is by default the column named id_task
Type: NVARCHAR(ANY)
Default: <none>

► _timecheck
the output will be enriched with execution time of critical sections if set to yes If not specified, the whole input table is subject of a single MANOVA run. used for type='column' only for type='trcv' it is by default the column named id_task

Type: NVARCHAR(ANY)
Default: <none>

▲ Returns
NVARCHAR(2000) A string confirming execution of the process or indicating a failure. The real output is contained in the output table

Details
This stored procedure performs one-way analysis of variance/covariance aiming to tell whether or not the groups of data identified by factor1 have the same mean value in all dependent variables or not. As an output 4 matrices (for each task) are produced that are stored in the output table the first matrix describes the ground means of all dependent variables (row vector) the second matrix describes the covariance table statistics and their p-values (row vector) the third matrix lists the eigenvalues of the covariance table statistics (column vector) the forth matrix lists the eigenvectors of the covariance table statistics (row of column vectors)

Examples

```
CREATE TABLE wheattest2trcv(
  id_task    INTEGER,
  row        INTEGER,
  col        INTEGER,
  val        DOUBLE);
-- INSERT INTO wheattest2 VALUES(1,'A',80, 4);
INSERT INTO wheattest2trcv VALUES(1,9,1, 0);
INSERT INTO wheattest2trcv VALUES(1,9,2, 80);
INSERT INTO wheattest2trcv VALUES(1,9,3, 4);
-- INSERT INTO wheattest2 VALUES(2,'A',65, 3);
INSERT INTO wheattest2trcv VALUES(1,1,1, 0);
INSERT INTO wheattest2trcv VALUES(1,1,2, 65);
INSERT INTO wheattest2trcv VALUES(1,1,3, 3);
-- INSERT INTO wheattest2 VALUES(3,'A',50, 2);
INSERT INTO wheattest2trcv VALUES(1,2,1, 0);
INSERT INTO wheattest2trcv VALUES(1,2,2, 50);
```
INSERT INTO wheattest2trcv VALUES(1,2,3, 2);
-- INSERT INTO wheattest2 VALUES(4,'B',100, 5);
INSERT INTO wheattest2trcv VALUES(1,3,1, 1);
INSERT INTO wheattest2trcv VALUES(1,3,2, 100);
INSERT INTO wheattest2trcv VALUES(1,3,3, 5);
-- INSERT INTO wheattest2 VALUES(5,'B',85, 4);
INSERT INTO wheattest2trcv VALUES(1,4,1, 1);
INSERT INTO wheattest2trcv VALUES(1,4,2, 85);
INSERT INTO wheattest2trcv VALUES(1,4,3, 4);
-- INSERT INTO wheattest2 VALUES(6,'B',70, 3);
INSERT INTO wheattest2trcv VALUES(1,5,1, 1);
INSERT INTO wheattest2trcv VALUES(1,5,2, 70);
INSERT INTO wheattest2trcv VALUES(1,5,3, 3);
-- INSERT INTO wheattest2 VALUES(7,'C',60, 2);
INSERT INTO wheattest2trcv VALUES(1,6,1, 2);
INSERT INTO wheattest2trcv VALUES(1,6,2, 60);
INSERT INTO wheattest2trcv VALUES(1,6,3, 2);
-- INSERT INTO wheattest2 VALUES(8,'C',75, 3);
INSERT INTO wheattest2trcv VALUES(1,7,1, 2);
INSERT INTO wheattest2trcv VALUES(1,7,2, 75);
INSERT INTO wheattest2trcv VALUES(1,7,3, 3);
-- INSERT INTO wheattest2 VALUES(9,'C',90, 5);
INSERT INTO wheattest2trcv VALUES(1,8,1, 2);
INSERT INTO wheattest2trcv VALUES(1,8,2, 90);
INSERT INTO wheattest2trcv VALUES(1,8,3, 5);
CALL nza..MANOVA_ONE_WAY_TEST('intable=wheattest2trcv, outtable=outtab');
select * from outtab order by id_matrix , row , col ;
CALL nza..DROP_TABLE('outtab');
CALL nza..DROP_TABLE('wheattest2trcv');
<table>
<thead>
<tr>
<th>BONFERRONI_CORR</th>
</tr>
</thead>
<tbody>
<tr>
<td>+----------------</td>
</tr>
<tr>
<td>+--------------------------------------------------------------</td>
</tr>
<tr>
<td>------+------------------</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>0.034119110061522</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>0.034119110061522</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1.44736237142</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>0.4444444444444444</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>-0.64150029909958</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>30</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1.037037037037</td>
</tr>
</tbody>
</table>

```
CREATE TABLE wheattest2(fieldId INTEGER, variety CHAR(1), yield DOUBLE, barrels INTEGER);
```

```
INSERT INTO wheattest2 VALUES(1,'A',80, 4);
INSERT INTO wheattest2 VALUES(2,'A',65, 3);
INSERT INTO wheattest2 VALUES(3,'A',50, 2);
INSERT INTO wheattest2 VALUES(4,'B',100, 5);
INSERT INTO wheattest2 VALUES(5,'B',85, 4);
INSERT INTO wheattest2 VALUES(6,'B',70, 3);
INSERT INTO wheattest2 VALUES(7,'C',60, 2);
INSERT INTO wheattest2 VALUES(8,'C',75, 3);
INSERT INTO wheattest2 VALUES(9,'C',90, 5);
```

```
CALL nza..MANOVA_ONE_WAY_TEST('intable=wheattest2,id=fieldId,
```
incolumn= yield; barrels, factor1=variety, outtable=outtab, type=columns');

SELECT * FROM outtab ORDER BY id_task, id_matrix, row , col;

CALL nza..DROP_TABLE('outtab');

CALL nza..DROP_TABLE('wheattest2');

<table>
<thead>
<tr>
<th>ID_TASK</th>
<th>ID_MATRIX</th>
<th>ROW</th>
<th>COL</th>
<th>VAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>75</td>
</tr>
<tr>
<td>EXPLANATION</td>
<td></td>
<td></td>
<td></td>
<td>vector of Grand Means</td>
</tr>
<tr>
<td>TASK_NAME</td>
<td>VARIABLE_NAME</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>---------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;YIELD&quot;</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID_TASK</th>
<th>ID_MATRIX</th>
<th>ROW</th>
<th>COL</th>
<th>VAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3.4444444444444</td>
</tr>
<tr>
<td>EXPLANATION</td>
<td></td>
<td></td>
<td></td>
<td>vector of Grand Means</td>
</tr>
<tr>
<td>TASK_NAME</td>
<td>VARIABLE_NAME</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>---------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;BARRELS&quot;</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID_TASK</th>
<th>ID_MATRIX</th>
<th>ROW</th>
<th>COL</th>
<th>VAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>EXPLANATION</td>
<td></td>
<td></td>
<td></td>
<td>Factor_1_present - &quot;VARIETY&quot;</td>
</tr>
<tr>
<td>TASK_NAME</td>
<td>VARIABLE_NAME</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>---------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID_TASK</th>
<th>ID_MATRIX</th>
<th>ROW</th>
<th>COL</th>
<th>VAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>EXPLANATION</td>
<td></td>
<td></td>
<td></td>
<td>Factor_2_not_present</td>
</tr>
<tr>
<td>TASK_NAME</td>
<td>VARIABLE_NAME</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>---------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID_TASK</th>
<th>ID_MATRIX</th>
<th>ROW</th>
<th>COL</th>
<th>VAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>0.39512195121951</td>
</tr>
<tr>
<td>EXPLANATION</td>
<td></td>
<td></td>
<td></td>
<td>WilksLambda</td>
</tr>
<tr>
<td>TASK_NAME</td>
<td>VARIABLE_NAME</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>---------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID_TASK</th>
<th>ID_MATRIX</th>
<th>ROW</th>
<th>COL</th>
<th>VAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>0.28038050472772</td>
</tr>
<tr>
<td>EXPLANATION</td>
<td></td>
<td></td>
<td></td>
<td>WilksLambdaPval</td>
</tr>
<tr>
<td>TASK_NAME</td>
<td>VARIABLE_NAME</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>---------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID_TASK</th>
<th>ID_MATRIX</th>
<th>ROW</th>
<th>COL</th>
<th>VAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>5</td>
<td>1.44736237142</td>
</tr>
<tr>
<td>EXPLANATION</td>
<td></td>
<td></td>
<td></td>
<td>RoysLargestRoot</td>
</tr>
<tr>
<td>TASK_NAME</td>
<td>VARIABLE_NAME</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>---------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID_TASK</th>
<th>ID_MATRIX</th>
<th>ROW</th>
<th>COL</th>
<th>VAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>6</td>
<td>0.26118761661441</td>
</tr>
<tr>
<td>EXPLANATION</td>
<td></td>
<td></td>
<td></td>
<td>RoysLargestRootPval</td>
</tr>
<tr>
<td>TASK_NAME</td>
<td>VARIABLE_NAME</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>---------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID_TASK</th>
<th>ID_MATRIX</th>
<th>ROW</th>
<th>COL</th>
<th>VAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>7</td>
<td>1.4814814814815</td>
</tr>
<tr>
<td>EXPLANATION</td>
<td></td>
<td></td>
<td></td>
<td>HotellingsTrace</td>
</tr>
<tr>
<td>TASK_NAME</td>
<td>VARIABLE_NAME</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>---------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID_TASK</th>
<th>ID_MATRIX</th>
<th>ROW</th>
<th>COL</th>
<th>VAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>8</td>
<td>0.26289561423837</td>
</tr>
<tr>
<td>EXPLANATION</td>
<td></td>
<td></td>
<td></td>
<td>HotellingsTracePval</td>
</tr>
<tr>
<td>TASK_NAME</td>
<td>VARIABLE_NAME</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>---------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID_TASK</th>
<th>ID_MATRIX</th>
<th>ROW</th>
<th>COL</th>
<th>VAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>9</td>
<td>0.62439024390244</td>
</tr>
<tr>
<td>EXPLANATION</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TASK_NAME</td>
<td>VARIABLE_NAME</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>---------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
PillaisTrace
|                  |           |
| 1 |         2 |   1 |  10 |  0.30419561076293 |

PillaisTracePval
| 0.30419561076293 |
|                  |
| 1 |         2 |   1 |  11 |                     | df
|                  |
| 1 |         2 |   1 |  12 | 0.034119110061522 |

Min Eigenvalue
|                  |
| 1 |         2 |   1 |  13 |                     | 3 | number of groups
|                  |
| 1 |         3 |   1 |  1 | 0.034119110061522 |

vector of characteristic roots for factor1
|                  |
| 1 |         3 |   2 |  1 | 1.44736237142 |

vector of characteristic roots for factor1
|                  |
| 1 |         4 |   1 |  1 | 0.44444444444444 |

matrix of characteristic vectors (provided in columns) for factor1 "YIELD"
|                  |
| 1 |         4 |   1 |  2 | -0.64150029909958 |

matrix of characteristic vectors (provided in columns) for factor1 "YIELD"
|                  |
| 1 |         4 |   2 |  1 | 30 |

matrix of characteristic vectors (provided in columns) for factor1 "BARRELS"
|                  |
| 1 |         4 |   2 |  2 | 1.037037037037 |

matrix of characteristic vectors (provided in columns) for factor1 "BARRELS"

CREATE TABLE wheatbytest(fieldId INTEGER, variety CHAR(1), yield DOUBLE, barrels INTEGER, experiment INT4);

INSERT INTO wheatbytest VALUES(1,'A',65,3,1);

-- suspect: INSERT INTO wheatbytest VALUES(2,'A',66,4,1);
### INSERT INTO wheatbytest VALUES

<table>
<thead>
<tr>
<th>Task ID</th>
<th>Variety</th>
<th>Yield</th>
<th>Barrels</th>
<th>Experiment</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>A</td>
<td>65</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>A</td>
<td>64</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>B</td>
<td>84</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>B</td>
<td>85</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>B</td>
<td>86</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>C</td>
<td>75</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>C</td>
<td>76</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>C</td>
<td>74</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>A</td>
<td>80</td>
<td>1000</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>A</td>
<td>65</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>13</td>
<td>A</td>
<td>50</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>14</td>
<td>B</td>
<td>100</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>15</td>
<td>B</td>
<td>85</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>16</td>
<td>B</td>
<td>70</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>17</td>
<td>C</td>
<td>60</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>18</td>
<td>C</td>
<td>75</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>19</td>
<td>C</td>
<td>90</td>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>

### CALL nza..MANOVA_ONE_WAY_TEST

```
CALL nza..MANOVA_ONE_WAY_TEST('intable=wheatbytest, id=fieldid, incolumn=yield;barrels, factor1=variety, by=experiment, outtable=outbytab,type=columns');
```

### SELECT * FROM outbytab ORDER BY id_task, id_matrix, row, col

```sql
select * from outbytab order by id_task, id_matrix, row, col;
```

### CALL nza..DROP_TABLE

```
CALL nza..DROP_TABLE('outbytab');
```

### CALL nza..DROP_TABLE

```
CALL nza..DROP_TABLE('wheatbytest');
```

<table>
<thead>
<tr>
<th>ID_TASK</th>
<th>ID_MATRIX</th>
<th>ROW</th>
<th>COL</th>
<th>VAL</th>
<th>EXPLANATION</th>
<th>BONFERRONI_CORR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>74.888888888889</td>
<td>vector of Grand Means</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>14</td>
<td>vector of Grand Means</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Factor_1_present - &quot;VARIETY&quot;</td>
</tr>
</tbody>
</table>
IBM Netezza In-Database Analytics Reference Guide

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>Factor_2_not_present</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0.0064612213792179</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>1</th>
<th>4</th>
<th>WilksLambdaPval 3.7570973977608e-05</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>5</td>
<td>142.0904611388</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>1</th>
<th>6</th>
<th>RoysLargestRootPval 0.0011681190006551</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>7</td>
<td>142.17208110438</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>1</th>
<th>8</th>
<th>HotellingsTracePval 2.9717940537966e-05</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>9</td>
<td>1.068472267282</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>1</th>
<th>10</th>
<th>PillaisTracePval 0.084025135820219</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>11</td>
<td>2</td>
<td>df 2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>1</th>
<th>12</th>
<th>Min Eigenvalue</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>13</td>
<td>number of groups</td>
<td>3</td>
</tr>
</tbody>
</table>

| 1 | 3 | 1 | 1 | 0.081619965579003 | 1 |

<table>
<thead>
<tr>
<th>1</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>vector of characteristic roots for factor1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>142.0904611388</td>
<td>1</td>
</tr>
</tbody>
</table>
vector of characteristic roots for factor1
<table>
<thead>
<tr>
<th></th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>
| matrix of characteristic vectors (provided in columns) for factor1 | 1 | "YIELD"

1 | 4 | 1 | 2 | -34.929549570148 |
| matrix of characteristic vectors (provided in columns) for factor1 | 1 | "YIELD"

1 | 4 | 2 | 1 | -951 |
| matrix of characteristic vectors (provided in columns) for factor1 | 1 | "BARRELS"

1 | 4 | 2 | 2 | 9.2673191996181 |
| matrix of characteristic vectors (provided in columns) for factor1 | 1 | "BARRELS"

2 | 1 | 1 | 1 | 75 |
| vector of Grand Means | 2 | "YIELD"

2 | 1 | 1 | 2 | 114.11111111111 |
| vector of Grand Means | 2 | "BARRELS"

2 | 2 | 1 | 1 | 1 |
| Factor_1_present - "VARIETY" | 2 |

2 | 2 | 1 | 2 | 0 |
| Factor_2_not_present | 2 |

2 | 2 | 1 | 3 | 0.39466805085924 |
| WilksLambda | 2 |

2 | 2 | 1 | 4 | 0.27975267472799 |
| WilksLambdaPval | 0.48124379043852 | 2 |

2 | 2 | 1 | 5 | 1.4501142776685 |
| RoysLargestRoot | 2 |

2 | 2 | 1 | 6 | 0.26074770123253 |
| RoysLargestRootPval | 0.45350603876701 | 2 |

2 | 2 | 1 | 7 | 1.4842598766347 |
| HotellingsTrace | 2 |

2 | 2 | 1 | 8 | 0.26228606026505 |
| HotellingsTracePval | 0.45577814312074 | 2 |
2 | 2 | 1 | 9 | 0.62487394580152 | PillaisTrace
| | | 2 |

2 | 2 | 1 | 10 | 0.30370770512967 | PillaisTracePval
| 0.51517704010421 | 2 |

df
| | 2 |

2 | 2 | 1 | 11 | 2 | Min Eigenvalue
| | 2 |

2 | 2 | 1 | 12 | 0.034145598966242 | number of groups
| | 2 |

2 | 3 | 1 | 1 | 0.034145598966242 | vector of characteristic roots for factor1
| | 2 |

2 | 3 | 2 | 1 | 1.4501142776685 | vector of characteristic roots for factor1
| | 2 |

2 | 4 | 1 | 1 | 0.44444444444444 | matrix of characteristic vectors (provided in columns)
for factor1
| | 2 | "YIELD"

2 | 4 | 1 | 2 | -0.64235906742576 | matrix of characteristic vectors (provided in columns)
for factor1
| | 2 | "YIELD"

2 | 4 | 2 | 1 | -9930 | matrix of characteristic vectors (provided in columns)
for factor1
| | 2 | "BARRELS"

2 | 4 | 2 | 2 | 1.0398154321903 | matrix of characteristic vectors (provided in columns)
for factor1
| | 2 | "BARRELS"

(42 rows)

-- CREATE TABLE wheatbytest(fieldId INTEGER, variety CHAR(1), yield DOUBLE, barrels INTEGER, experiment INT4);

CREATE TABLE wheatbytesttrcv(
   id_task INTEGER
);
, col        INTEGER
, val        DOUBLE);
-- INSERT INTO wheatbytest VALUES(1,'A',65,3,1);
INSERT INTO wheatbytesttrcv VALUES(1,1,1, 0);
INSERT INTO wheatbytesttrcv VALUES(1,1,2, 1000);
INSERT INTO wheatbytesttrcv VALUES(1,1,3, 3);
-- INSERT INTO wheatbytest VALUES(2,'A',66,4,1);
INSERT INTO wheatbytesttrcv VALUES(1,2,1, 0);
--suspectINSERT INTO wheatbytesttrcv VALUES(1,2,2, 66);
--suspectINSERT INTO wheatbytesttrcv VALUES(1,2,3, 4);
INSERT INTO wheatbytesttrcv VALUES(1,2,2, 65);
INSERT INTO wheatbytesttrcv VALUES(1,2,3, 3);
-- INSERT INTO wheatbytest VALUES(3,'A',64,2,1);
INSERT INTO wheatbytesttrcv VALUES(1,3,1, 0);
INSERT INTO wheatbytesttrcv VALUES(1,3,2, 64);
INSERT INTO wheatbytesttrcv VALUES(1,3,3, 2);
-- INSERT INTO wheatbytest VALUES(4,'B',84,3,1);
INSERT INTO wheatbytesttrcv VALUES(1,4,1, 1);
INSERT INTO wheatbytesttrcv VALUES(1,4,2, 84);
INSERT INTO wheatbytesttrcv VALUES(1,4,3, 3);
-- INSERT INTO wheatbytest VALUES(5,'B',85,4,1);
INSERT INTO wheatbytesttrcv VALUES(1,5,1, 1);
INSERT INTO wheatbytesttrcv VALUES(1,5,2, 85);
INSERT INTO wheatbytesttrcv VALUES(1,5,3, 4);
-- INSERT INTO wheatbytest VALUES(6,'B',86,5,1);
INSERT INTO wheatbytesttrcv VALUES(1,6,1, 1);
INSERT INTO wheatbytesttrcv VALUES(1,6,2, 86);
INSERT INTO wheatbytesttrcv VALUES(1,6,3, 5);
-- INSERT INTO wheatbytest VALUES(7,'C',75,3,1);
INSERT INTO wheatbytesttrcv VALUES(1,7,1, 2);
INSERT INTO wheatbytesttrcv VALUES(1,7,2, 75);
INSERT INTO wheatbytesttrcv VALUES(1,7,3, 3);
-- INSERT INTO wheatbytest VALUES(8,'C',76,4,1);
INSERT INTO wheatbytesttrcv VALUES(1,8,1, 2);
INSERT INTO wheatbytesttrcv VALUES(1,8,2, 76);
INSERT INTO wheatbytesttrcv VALUES(1,8,3, 4);
-- INSERT INTO wheatbytest VALUES(9,'C',74,2,1);
INSERT INTO wheatbytesttrcv VALUES(1,9,1, 2);
INSERT INTO wheatbytesttrcv VALUES(1,9,2, 74);
INSERT INTO wheatbytesttrcv VALUES(1,9,3, 2);
-- INSERT INTO wheatbytest VALUES(11,'A',80, 4,2);
INSERT INTO wheatbytesttrcv VALUES(2,9,1, 0);
INSERT INTO wheatbytesttrcv VALUES(2,9,2, 80);
INSERT INTO wheatbytesttrcv VALUES(2,9,3, 4);
-- INSERT INTO wheatbytest VALUES(12,'A',65, 3,2);
INSERT INTO wheatbytesttrcv VALUES(2,1,1, 0);
INSERT INTO wheatbytesttrcv VALUES(2,1,2, 65);
INSERT INTO wheatbytesttrcv VALUES(2,1,3, 3);
--INSERT INTO wheatbytest VALUES(13,'A',50,2,2);
INSERT INTO wheatbytesttrcv VALUES(2,2,1, 1);
INSERT INTO wheatbytesttrcv VALUES(2,2,2, 100);
INSERT INTO wheatbytesttrcv VALUES(2,2,3, 5);
--INSERT INTO wheatbytest VALUES(14,'B',100,5,2);
INSERT INTO wheatbytesttrcv VALUES(2,3,1, 1);
INSERT INTO wheatbytesttrcv VALUES(2,3,2, 100);
INSERT INTO wheatbytesttrcv VALUES(2,3,3, 5);
--INSERT INTO wheatbytest VALUES(15,'B',85,4,2);
INSERT INTO wheatbytesttrcv VALUES(2,4,1, 1);
INSERT INTO wheatbytesttrcv VALUES(2,4,2, 85);
INSERT INTO wheatbytesttrcv VALUES(2,4,3, 4);
--INSERT INTO wheatbytest VALUES(16,'B',70,3,2);
INSERT INTO wheatbytesttrcv VALUES(2,5,1, 1);
INSERT INTO wheatbytesttrcv VALUES(2,5,2, 70);
INSERT INTO wheatbytesttrcv VALUES(2,5,3, 3);
--INSERT INTO wheatbytest VALUES(17,'C',60,2,2);
INSERT INTO wheatbytesttrcv VALUES(2,6,1, 2);
INSERT INTO wheatbytesttrcv VALUES(2,6,2, 60);
INSERT INTO wheatbytesttrcv VALUES(2,6,3, 2);
--INSERT INTO wheatbytest VALUES(18,'C',75,3,2);
INSERT INTO wheatbytesttrcv VALUES(2,7,1, 2);
INSERT INTO wheatbytesttrcv VALUES(2,7,2, 75);
INSERT INTO wheatbytesttrcv VALUES(2,7,3, 3);
-- INSERT INTO wheatbytest VALUES(19,'C',90,5,2);
INSERT INTO wheatbytesttrcv VALUES(2,8,1, 2);
INSERT INTO wheatbytesttrcv VALUES(2,8,2, 90);
INSERT INTO wheatbytesttrcv VALUES(2,8,3, 5);
CALL nza..MANOVA_ONE_WAY_TEST('intable=wheatbytesttrcv,
outtable=outbytab_trcv,type=trcv');
select * from outbytab_trcv order by id_task , id_matrix , row ,
col ;
CALL nza..DROP_TABLE('outbytab_trcv');
CALL nza..DROP_TABLE('wheatbytesttrcv');

<table>
<thead>
<tr>
<th>ID_TASK</th>
<th>TASK_NAME</th>
<th>ID_MATRIX</th>
<th>ROW</th>
<th>COL</th>
<th>VAL</th>
<th>EXPLANATION</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BONFERRONI_CORR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
+-------------------+
| 1 | 1 | 1 | 1 | 1 | 178.77777777778 | vector of Grand Means |
| 1 | 1 | 1 | 1 | 2 | 3.2222222222222 | vector of Grand Means |
| 1 | 2 | 1 | 1 | 1 | Factor_1_present |
| 1 | 2 | 1 | 1 | 2 | Factor_2_not_present |
| 1 | 2 | 1 | 1 | 3 | 0.45931933970544 | WilksLambda |

---
|     |   1 |   1 |   2 |   1 |   4 | 0.3733800101217 | WilksLambdaPval | 0.60734738828492 |
|     |   1 |   1 |   2 |   1 |   5 | 0.96040503530125 | RoysLargestRoot |
|     |   1 |   1 |   2 |   1 |   6 | 0.36431855104074 | RoysLargestRootPval | 0.59590909544905 |
|     |   1 |   1 |   2 |   1 |   7 | 1.0709584679764 | HotellingsTrace |
|     |   1 |   1 |   2 |   1 |   8 | 0.3784638014586 | HotellingsTracePval | 0.61369275390271 |
|     |   1 |   1 |   2 |   1 |   9 | 0.58944938422624 | PillaisTrace |
|     |   1 |   1 |   2 |   1 |  10 | 0.34070050877455 | PillaisTracePval | 0.56532418086986 |
| 2 | df |
|     |   1 |   1 |   2 |   1 |   11 | 2 | df |
|     |   1 |   1 |   2 |   1 |   12 | 0.11055343267515 | Min Eigenvalue |
|     |  3 | number of groups |
|     |   1 |   1 |   3 |   1 |   1 | 0.11055343267515 | vector of characteristic roots for factor1 |
|     |   1 |   1 |   3 |   2 |   1 | 0.96040503530125 | vector of characteristic roots for factor1 |
|     |   1 |   1 |   4 |   1 |   1 | 0.30127430916995 | matrix of characteristic vectors (provided in columns) for factor1 |
Reference Documentation: Analytics

-0.3545604608075 | matrix of characteristic vectors (provided in columns) for factor1 |
    1 |         1 |         4 |   2 |   1 |

-478.88888888889 | matrix of characteristic vectors (provided in columns) for factor1 |
    1 |         1 |         4 |   2 |   2 |

0.76968415880645 | matrix of characteristic vectors (provided in columns) for factor1 |

180.55555555556 | vector of Grand Means |

3.4444444444444 | vector of Grand Means |

1 | Factor_1_present |

0 | Factor_2_not_present |

0.68746202700252 | WilksLambda |

0.72662382122884 | WilksLambdaPval |

0.33309697063248 | RoysLargestRoot |

0.64969180345486 | RoysLargestRootPval |

0.4242597521801 | HotellingsTrace |

0.72653876921139 | HotellingsTracePval |

0.33341347678564 | PillaisTrace |

0.66968322586599 | PillaisTracePval
| 2 |  2 |  2 |   1 |  11 |
| 2 | df |
| 2 |  2 |  2 |   1 |  12 |
| 0.091162781547621 | Min Eigenvalue |
| 2 |  2 |  2 |   1 |  13 |
| number of groups |
| 2 |  2 |  3 |   1 |  1 |
| 0.091162781547621 | vector of characteristic roots for factor1 |
| 2 |  2 |  3 |   2 |   1 |
| 0.33309697063248 | vector of characteristic roots for factor1 |
| 2 |  2 |  4 |   1 |   1 |
| 0.31705878371191 | matrix of characteristic vectors (provided in columns) for factor1 |
| 2 |  2 |  4 |   1 |   2 |
| -0.060191048398572 | matrix of characteristic vectors (provided in columns) for factor1 |
| 2 |  2 |  4 |   2 |   1 |
| -392.22222222222 | matrix of characteristic vectors (provided in columns) for factor1 |
| 2 |  2 |  4 |   2 |   2 |
| 0.10720096846819 | matrix of characteristic vectors (provided in columns) for factor1 |

(42 rows)

Related Functions

► category Analytics - Statistics
► MANOVA_Two WAY_TEST
► ANOVA_CRD_TEST

MANOVA_Two WAY_TEST - Multivariate Analysis of Variance in Two way setting

This stored procedure performs a completely randomized design MANOVA
Usage

The MANOVA_Two_WAY_TEST stored procedure has the following syntax:

\[
\text{MANOVA\_Two\_WAY\_TEST(NVARCHAR(ANY) paramString)}
\]

- **paramString**
  - comma-separated list of \(<\text{parameter}>=\text{<value>}\) entries with parameters below
  - Type: NVARCHAR(ANY)

- **intable**
  - the input table name
  - Type: NVARCHAR(ANY)

- **outtable**
  - the output table name
  - Type: NVARCHAR(ANY)

- **factor1**
  - the input table column identifying a first factor (so-called treatment in RBD/CRD nomenclature)
  - used for type='column' only for type='trcv' these are the values for col-column equal 1
  - Type: NVARCHAR(ANY)

- **factor2**
  - the input table column identifying a second factor (so-called block in RBD nomenclature)
  - Type: NVARCHAR(ANY) used for type='column' only for type='trcv' these are the values for col-column equal 2

- **incolumn**
  - the input table observation columns (dependent variables), separated by a semi-colon (;).
  - used for type='column' only for type='trcv' these are the values for col-column greater equal 3
  - Type: NVARCHAR(ANY)

- **type**
  - the input table form: either 'columns' or 'trcv'
  - trcv stands for "id\_task, row, column, value"
  - id\_task must be positive integer >= 1.
  - columns means the traditional table representation
  - Type: NVARCHAR(ANY)
  - Default: 'trcv'

- **id**
  - the input table column which uniquely identifies records
  - used for type='column' only for type='trcv' not needed due to the structure of the table
  - Type: NVARCHAR(ANY)

- **by**
  - the input table column which splits the input table into subtables, on each of them a separate
  - MANOVA is run
  - If not specified, the whole input table is subject of a single MANOVA run
  - used for type='column' only for type='trcv' it is by default the column named id\_task
  - Type: NVARCHAR(ANY)
IBM Netezza In-Database Analytics Reference Guide

Default: <none>

- **_timecheck**
  the output will be enriched with execution time of critical sections if set to yes If not specified, the whole input table is subject of a single MANOVA run. used for type='column' only for type='trcv' it is by default the column named id_task

  Type: NVARCHAR(ANY)

  Default: <none>

▲ Returns

  NVARCHAR(2000) A string confirming execution of the process or indicating a failure. The real output is contained in the output table

**Details**

This stored procedure performs Two-way analysis of variance/covariance aiming to tell whether or not the groups of data identified by factor1 have the same mean value in all dependent variables or not.

**Examples**

```sql
CREATE TABLE wheattest2(fieldId INTEGER, variety CHAR(1), yield DOUBLE, barrels INTEGER);
INSERT INTO wheattest2 VALUES(1,'A',80, 4);
INSERT INTO wheattest2 VALUES(2,'A',65, 3);
INSERT INTO wheattest2 VALUES(3,'A',50, 2);
INSERT INTO wheattest2 VALUES(4,'B',100, 5);
INSERT INTO wheattest2 VALUES(5,'B',85, 4);
INSERT INTO wheattest2 VALUES(6,'B',70, 3);
INSERT INTO wheattest2 VALUES(7,'C',60, 2);
INSERT INTO wheattest2 VALUES(8,'C',75, 3);
INSERT INTO wheattest2 VALUES(9,'C',90, 5);
CALL nza..MANOVA_Two_WAY_TEST('intable=wheattest2,id=fieldId, incolumn= yield, factor2=barrels, factor1=variety, outtable=outtab, type=columns');
SELECT * FROM outtab  order by id_task,col;
CALL nza..DROP_TABLE('outtab');
CALL nza..DROP_TABLE('wheattest2');
CREATE TABLE
INSERT 0 1
```
CREATE TABLE wheatbytest(fieldId INTEGER, variety CHAR(1), yield DOUBLE, barrels INTEGER, experiment INT4);

INSERT INTO wheatbytest VALUES(1,'A',65,3,1);
-- suspect: INSERT INTO wheatbytest VALUES(2,'A',66,4,1);
INSERT INTO wheatbytest VALUES(2,'A',65,3,1);
INSERT INTO wheatbytest VALUES(3,'A',64,2,1);
INSERT INTO wheatbytest VALUES(4,'B',84,3,1);
INSERT INTO wheatbytest VALUES(5,'B',85,4,1);
INSERT INTO wheatbytest VALUES(6,'B',86,5,1);
INSERT INTO wheatbytest VALUES(7,'C',75,3,1);
INSERT INTO wheatbytest VALUES(8,'C',76,4,1);
INSERT INTO wheatbytest VALUES(9,'C',74,2,1);
INSERT INTO wheatbytest VALUES(11,'A',80,4,2);
INSERT INTO wheatbytest VALUES(12,'A',65,3,2);
INSERT INTO wheatbytest VALUES(13,'A',50,2,2);
INSERT INTO wheatbytest VALUES(14,'B',100,5,2);
INSERT INTO wheatbytest VALUES(15,'B',85,4,2);
INSERT INTO wheatbytest VALUES(16,'B',70,3,2);
INSERT INTO wheatbytest VALUES(17,'C',60,2,2);
INSERT INTO wheatbytest VALUES(18,'C',75,3,2);
INSERT INTO wheatbytest VALUES(19,'C',90,5,2);
CALL nza..MANOVA_Two_WAY_TEST('intable=wheatbytest, id=fieldid, incolumn=yield, factor2=barrels, factor1=variety, by=experiment, outtable=outbytab,type=columns');
SELECT * FROM outbytab order by id_task,col,task_name;
CALL nza..DROP_TABLE('outbytab');
CALL nza..DROP_TABLE('wheatbytest');
CREATE TABLE
Related Functions

- category Analytics - Statistics
- MANOVA_ONE_WAY_TEST
- PRINT_MANOVA_Two_WAY_TEST
- ANOVA_CRD_TEST

MEDIAN - Median value for a numeric attribute

This stored procedure calculates the median value of a numeric column

Usage

The MEDIAN stored procedure has the following syntax:

- MEDIAN(NVARCHAR(ANY) paramString)
  ▲ Parameters
paramString
comma-separated list of <parameter>=<value> entries with parameters below
Type: NVARCHAR(ANY)

intable
the input table or view
Type: NVARCHAR(256)

incolumn
the input table column
Type: NVARCHAR(128)

Returns
REAL the median value

Details
This stored procedure calculates the median value of a numeric column using Torben Mogensen's algorithm. The median value is the value separating the input values in two sets (lower and higher values) of equal size.

Examples
CALL nza..MEDIAN('intable=nza..quant_iris,
incolumn=sepal_length');

MEDIAN
--------
  5.8

(1 row)

Related Functions
► category Analytics - Quantiles & Outliers
► MEDIAN_DISC

MEDIAN_DISC - Median value for a discrete attribute
This stored procedure calculates the median value of a discrete column (when applicable)

Usage
The MEDIAN_DISC stored procedure has the following syntax:

► MEDIAN_DISC(NVARCHAR(ANY) paramString)
  ▲ Parameters
**paramString**
comma-separated list of `<parameter>=<value>` entries with parameters below
Type: NVARCHAR(ANY)

**intable**
the input table or view
Type: NVARCHAR(256)

**incolumn**
the input table column
Type: NVARCHAR(128)

Returns
NVARCHAR the median value

**Details**
This stored procedure calculates the median value of a discrete column. The median value is the value separating the input values in two sets (lower and higher values) of equal size.

**Examples**

```sql
CALL nza..MEDIAN_DISC('intable=nza..censusincome, incolumn=marital_status');
```

```
  MEDIAN_DISC
  ------------------
  Never married
  (1 row)
```

**Related Functions**
- category Analytics - Quantiles & Outliers
- MEDIAN

**METADATA_ANALYZE - Check and Repair Metadata Repository**

This procedure can check the consistency of the metadata repository and repair some of the detected inconsistencies. It should also be used to ensure consistency of the metadata tables before a database backup and after a database restore.

**Usage**

The METADATA_ANALYZE stored procedure has the following syntax:

```sql
METADATA_ANALYZE(NVARCHAR(ANY))
```

▲ Parameters
paramString
a comma-separated list of <parameter>=<value> entries with parameters below
Type: NVARCHAR(ANY)

mode
The operation mode of the procedure. Supported modes are: check, repair, delete, undo, backup, restore.
Type: VARCHAR(ANY)

model
The name of a model to be checked/repaired. The procedure checks/repairs only the metadata of this one model. If the parameter is not specified, all models are checked/repaired. This parameter is allowed only if the mode is check or repair.
Type: NVARCHAR(ANY)

verbose
If this is true (the default), a detailed result report is printed. If it is set to false, only a summary is printed.
Type: BOOLEAN

Returns
INTEGER The number of unresolved problems (the number of problems found minus the number of problems that could be fixed).

Details
Check mode: The structure of the metadata tables and views are checked, and the contents of the tables is analyzed to find any illegal or inconsistent values. All problems are reported. If a problem can be repaired, this is printed. If a model is too corrupt and should be deleted, this is printed also.

Repair mode: The same checks are done like in "check mode", but any problems that can be repaired are repaired immediately.

Delete mode: The models are deleted that are reported as too corrupt in "check mode".

Undo mode: All changes done in "repair mode" are undone. This works only if no other changes on the metadata tables have been performed since the last "repair mode" operation. Models that have been deleted in "delete mode" cannot be recovered!

Backup mode and Restore mode: After database backup and restore of an INZA-enabled database some of the metadata can be wrong. This procedure can fix the wrong values when you call it in "restore mode" after the database restore operation. However, to be able to always fix all wrong values, the procedure should also be called before a database backup in "backup mode". In this mode only those inconsistencies are checked and repaired that are later needed in "repair mode" to repair the metadata after database restore.

Examples

call nza..metadata_analyze();
NOTICE:
***************************************************************
*************
NOTICE: Metadata Repository Diagnostics and Repair
NOTICE: INZA-Version: 3.0.0.27267
NOTICE: Metadata-Version: 1.0
NOTICE: System Case: Uppercase
NOTICE: Operation Mode: check
NOTICE: Execution Time: 2013-03-07 08:43:43
NOTICE: Session User: ADMIN
NOTICE:
***************************************************************
*************
NOTICE:
NOTICE: Checking metadata database objects
NOTICE: Checking analytic models
NOTICE:
NOTICE:
***************************************************************
*************
NOTICE: Summary
NOTICE: Metadata Repository Structure: OK
NOTICE: Number of checked models: 3
NOTICE: Number of errors: 0
NOTICE: Number of errors that were fixed: 0
NOTICE:
***************************************************************
*************
NOTICE:
METADATA ANALYZE
------------------
0
(1 row)

Related Functions
► category Analytics - Model Management
MIGRATE_MODEL - Migrate an analytics model

This stored procedure migrates an older version of an analytics model to the latest version.

Usage

The MIGRATE_MODEL stored procedure has the following syntax:

- MIGRATE_MODEL(NVARCHAR(ANY))

  ▲ Parameters
  ▶ paramString
    comma-separated list of <parameter>=<value> entries with parameters below
    Type: NVARCHAR(ANY)
  ▶ model
    the model to be migrated
    Type: NVARCHAR(64)

  ▲ Returns
    TEXT A string indicating the model version before and after the migration separated by string ' => '

Details

This stored procedure migrates a given analytics model from an older version to the latest version supported. The version is available in column COMPONENTFORMAT of the view V_NZA_MODELS. The call is forwarded to a model-type specific migrate procedure of the form MIGRATE_<procedure-name>, where <procedure-name> is the name of the model building procedure. If there is no migrate procedure for the model type, an exception is raised. If the model has already the latest format, an exception is raised.

Related Functions

- category Analytics - Model Management
- LIST_MODELS

MODEL_EXISTS - Check if an analytics model exists

This stored procedure checks if the given model exists. The model can be searched in the current or in another given database.

Usage

The MODEL_EXISTS stored procedure has the following syntax:

- MODEL_EXISTS(NVARCHAR(ANY))
Parameters

- **paramString**
  - comma-separated list of <parameter>=<value> entries with parameters below
  - Type: NVARCHAR(ANY)

- **model**
  - the model to check. It can be qualified by a database name (<database>..<model>) if not in the current database.
  - Type: NVARCHAR(194)

- **verbose**
  - flag indicating to raise an exception if the model does not exist (inverse=false) or exists (inverse=true)
  - Type: BOOLEAN
  - Default: false

- **inverse**
  - flag indicating to invert the exception logic in the parameter verbose.
  - Type: BOOLEAN
  - Default: false

Returns

- BOOLEAN true if the model exists, false if the model does not exists and verbose=false (otherwise an exception is raised)

Details

This stored procedure checks if the given model exists. No privileges are required. Even if the procedure returns true for a model, the user needs the LIST privilege to further list properties of this model.

Examples

```sql
CALL nza..MODEL_EXISTS('model=nomodel, verbose=false, inverse=false');
CALL nza..MODEL_EXISTS('model=nomodel, verbose=true, inverse=false');
CALL nza..MODEL_EXISTS('model=nomodel, verbose=true, inverse=true');
```

```
MODEL_EXISTS
-------------
    f
(1 row)
```

ERROR: A model with name "NOMODEL" cannot be found.
Related Functions

- category Analytics - Model Management
- LIST_MODELS

MOMENTS - Moments of a column

This stored procedure calculates the moments of a numeric input column: mean, variance, standard deviation, skewness and (excess) kurtosis as well as the count of cases, the minimum and the maximum.

Usage

The MOMENTS stored procedure has the following syntax:

```
MOMENTS(NVARCHAR(ANY) paramString)
```

Parameters

- `paramString`
  - comma-separated list of `<parameter>=<value>` entries with parameters below
  - Type: NVARCHAR(ANY)

- `intable`
  - the input table
  - Type: NVARCHAR(256)

- `outtable`
  - the output table to write the moments into
  - Type: NVARCHAR(256)

- `incolumn`
  - the numeric input table column
  - Type: NVARCHAR(128)

- `by`
  - the input table column which splits the data into groups for which the operation is to be performed
Returns

INT8 the number of rows in the output table

Details

This stored procedure calculates the moments of a single numeric input column, i.e. its descriptive statistics, either in the whole input table or within the groups defined in the column specified by parameter <by>. A k-th order central moment is the sum of the k-th powers of differences between the mean and the actual value divided by the number of cases.

- the mean (or average) is computed as the sum of values in the column divided by the number of cases,
- the variance is called the second central moment and is obtained as the sum of squares of the difference between the estimated mean and the actual value, divided by the number of cases reduced by 1 (to estimate population),
- the standard deviation is the square root of variance,
- the skewness is the third central moment divided by the cube of standard deviation,
- the (excess) Kurtosis is the forth central moment divided by the forth power of standard deviation minus 3 (so that kurtosis of normal distribution is zero).

Note that computing some of the moments may not make sense (in that case NULL is returned), e.g. mean for less than 1 cases, variance for less than 2 cases etc. If a value is missing, the complete row is ignored.

The output table is created with following columns: columnname, countt, average, variance, stddev, skewness, kurtosis, minimum, maximum. If the parameter by is specified, an additional column <by> is added to indicate for which group the moments have been calculated.

Examples

```sql
CALL nza..MOMENTS('intable=nza..iris, incolumn=petallength, outtable=iris_mo_15');

SELECT * FROM iris_mo_15;

CALL nza..DROP_TABLE('iris_mo_15');

MOMENTS
---------

<table>
<thead>
<tr>
<th>PETALLENGTH</th>
<th>COUNTT</th>
<th>AVERAGE</th>
<th>VARIANCE</th>
<th>STDDEV</th>
<th>SKEWNESS</th>
<th>KURTOSIS</th>
<th>MINIMUM</th>
<th>MAXIMUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>150</td>
<td>3.7586666666667</td>
<td>3.1131794183445</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CALL nza..MOMENTS('intable=nza..adult,
incolumn=hours_per_week, by=education,
outtable=adult_mo_13');
SELECT *FROM adult_mo_13 ORDER BY EDUCATION;
CALL nza..DROP_TABLE('adult_mo_13');

MOMENTS
--------

16

<table>
<thead>
<tr>
<th>COLUMNNAME</th>
<th>COUNTT</th>
<th>AVERAGE</th>
<th>VARIANCE</th>
<th>STDDEV</th>
<th>SKEWNESS</th>
<th>KURTOSIS</th>
<th>MINIMUM</th>
<th>MAXIMUM</th>
<th>EDUCATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOURS_PER_WEEK</td>
<td>933</td>
<td>37.052519</td>
<td>190.11204568768</td>
<td>13.788112477337</td>
<td>0.41947853887823</td>
<td>2.8194664494733</td>
<td>1</td>
<td>99</td>
<td>10th</td>
</tr>
<tr>
<td>HOURS_PER_WEEK</td>
<td>1175</td>
<td>33.925957</td>
<td>195.03284352459</td>
<td>13.96541598108</td>
<td>0.13243191005983</td>
<td>0.56796404718583</td>
<td>2</td>
<td>99</td>
<td>11th</td>
</tr>
<tr>
<td>HOURS_PER_WEEK</td>
<td>433</td>
<td>35.780600</td>
<td>159.42628945343</td>
<td>12.626412734599</td>
<td>0.4030517521683</td>
<td>3.0001006846437</td>
<td>6</td>
<td>99</td>
<td>12th</td>
</tr>
<tr>
<td>HOURS_PER_WEEK</td>
<td>168</td>
<td>38.255952</td>
<td>165.08978471628</td>
<td>12.848726968703</td>
<td>0.70491994474217</td>
<td>3.5673978626614</td>
<td>4</td>
<td>96</td>
<td>1st-4th</td>
</tr>
<tr>
<td>HOURS_PER_WEEK</td>
<td>333</td>
<td>38.897898</td>
<td>111.33894135099</td>
<td>10.5517269326904</td>
<td>-0.049526193734453</td>
<td>3.3875831061533</td>
<td>3</td>
<td>84</td>
<td>5th-6th</td>
</tr>
</tbody>
</table>
HOURS_PER_WEEK | 646 | 39.366873 | 201.69310245518 |
14.201869681672 | 0.73753513008848 | 3.8823103687329 |
2 | 99 | 7th-8th

HOURS_PER_WEEK | 514 | 38.044747 | 122.42099574487 |
11.064402186511 | -0.13878017384913 | 3.5898830831323 |
1 | 99 | 9th

HOURS_PER_WEEK | 1067 | 40.504217 | 148.75865949489 |
12.19666917163 | 0.088267958549645 | 3.461544439783 |
1 | 99 | Assoc-acdm

HOURS_PER_WEEK | 1382 | 41.610709 | 116.49714808477 |
10.793384459231 | 0.56242110278045 | 5.486236627901 |
1 | 99 | Assoc-voc

HOURS_PER_WEEK | 5355 | 42.614006 | 131.01515681357 |
11.446185251583 | 0.0025610250718387 | 3.0060669086706 |
2 | 99 | Bachelors

HOURS_PER_WEEK | 413 | 46.973366 | 227.54055102377 |
15.0844732245 | 0.34961574790415 | 2.2253595730787 |
1 | 99 | Doctorate

HOURS_PER_WEEK | 10501 | 40.575374 | 128.45405634837 |
11.337557379985 | 0.4752525126981542 | 4.2257126981542 |
1 | 99 | HS-grad

HOURS_PER_WEEK | 1723 | 43.836332 | 150.74439350645 |
12.277800841619 | 0.19239697850573 | 3.138568576645 |
1 | 99 | Masters

HOURS_PER_WEEK | 51 | 36.647059 | 157.63294117647 |
12.555195784076 | 0.4024381312025 | 1.7918249561792 |
10 | 75 | Preschool

HOURS_PER_WEEK | 576 | 47.425347 | 219.21876509662 |
14.806038129649 | 0.236967958519008 | 1.72895218026 |
2 | 99 | Prof-school

HOURS_PER_WEEK | 7291 | 38.852284 | 162.86610555246 |
12.761900546253 | 0.1312456482153 | 1.9985175214256 |
1 | 99 | Some-college

(16 rows)

DROP_TABLE
-----------
t
(1 row)

Related Functions
► category Analytics - Statistics
MSE - Mean Squared Error

This stored procedure calculates the mean squared error of Regression predictions.

**Usage**

The MSE stored procedure has the following syntax:

```
MSE(NVARCHAR(ANY) paramString)
```

**Parameters**

- **paramString**
  comma-separated list of <parameter>=<value> entries with parameters below
  Type: NVARCHAR(ANY)

- **pred_table**
  the input table or view containing predicted values
  Type: NVARCHAR(256)

- **pred_column**
  the input table column in <pred_table> containing predicted values
  Type: NVARCHAR(128)

- **pred_id**
  the input table column in <pred_table> identifying a unique instance id
  Type: NVARCHAR(128)

- **true_table**
  the input table or view containing real values
  Type: NVARCHAR(256)

- **true_column**
  the input table column in <true_table> containing real values
  Type: NVARCHAR(128)

- **true_id**
  the input table column in <true_table> identifying a unique instance id
  Type: NVARCHAR(128)

- **check**
  flag indicating to check parameters or not. This may consume some time but prevents usage errors. Allowed values are: 'all' (check all), 'none' (check nothing), 'nulls' (check for nulls only).
Type: NVARCHAR(ANY)
Default: all

▲ Returns
   DOUBLE the Mean Squared Error

Details
This stored procedure calculates the mean squared error of Regression predictions. This is done by comparing the predictions made when applying a Regression model onto data, and the real values for this data. The MSE value is calculated as \[ \text{sum}((\text{true}_i - \text{pred}_i)^2) / \text{count}(\text{true}_i) \] where \( \text{true}_i \) is the real value and \( \text{pred}_i \) is the predicted value.

Examples

```sql
CALL nza..GROW_REGTREE('model=wrt, intable=nza..weatherr, id=instance, target=grade, maxdepth=4, minsplit=2');
CALL nza..PREDICT_REGTREE('model=wrt, intable=nza..weatherr, id=instance, outtable=wpr, var=TRUE');
CALL nza..MSE('pred_table=wpr,true_table=nza..weatherr,pred_column=class,true_column=grade,pred_id=id,true_id=instance');
CALL nza..DROP_MODEL('model=wrt');
CALL nza..DROP_TABLE('wpr');
```

```
GROW_REGTREE
---------------
      13
(1 row)

PREDICT_REGTREE
---------------
      22
(1 row)

MSE
---------------
0.090909090909091
(1 row)

DROP_MODEL
```
**Related Functions**

- category Analytics - Diagnostic Measures
- CERROR
- MAE
- RAE
- RSE

**MTBNET_DIFF - Show differences between multiple tree-shaped bayesian networks**

The procedure shows the differences between multiple tree-shaped bayesian networks for distinct values of the class. In particular the columns V1,v2 feature the edge that is present in the group indicated in the column PRESENT_ON, but is absent in the group MISSING_ON.

**Usage**

The MTBNET_DIFF stored procedure has the following syntax:

```
MTBNET_DIFF(nvarchar(ANY) paramString)
```

**Parameters**

- **paramString**
  A comma-separated list of `<parameter>=<value>` entries using the parameters below.
  Type: NVARCHAR(ANY)

- **model**
  The name of the input model containing the model data.
  Type: nvarchar(ANY)

- **modelfdiff**
  The output table name that is to contain the model differences between classes.
**Type:** NVARCHAR(ANY)

▲ **Returns**

INT - 0

**Examples**

```sql
call nza..drop_model('model=struc_iris');
call nza..MTBNet_grow('intable=nza..iris,class=class,incolumn=SEPALLENGTH; SEPALWIDTH; PETALLENGTH; PETALWIDTH,model=struc_iris');
drop table struc_diff_iris;
call nza..MTBNet_diff('model=struc_iris,modeldiff=struc_diff_iris');
select * from struc_diff_iris order by MISSING_ON , PRESENT_ON,v1,v2;
```

<table>
<thead>
<tr>
<th>MISSING_ON</th>
<th>PRESENT_ON</th>
<th>V1</th>
<th>V2</th>
</tr>
</thead>
<tbody>
<tr>
<td>setosa</td>
<td>versicolor</td>
<td>PETALLENGTH</td>
<td>SEPALLENGTH</td>
</tr>
<tr>
<td>setosa</td>
<td>virginica</td>
<td>PETALLENGTH</td>
<td>SEPALLENGTH</td>
</tr>
<tr>
<td>versicolor</td>
<td>setosa</td>
<td>SEPALLENGTH</td>
<td>SEPALWIDTH</td>
</tr>
<tr>
<td>versicolor</td>
<td>virginica</td>
<td>SEPALLENGTH</td>
<td>SEPALWIDTH</td>
</tr>
<tr>
<td>virginica</td>
<td>setosa</td>
<td>PETALLENGTH</td>
<td>PETALWIDTH</td>
</tr>
<tr>
<td>virginica</td>
<td>versicolor</td>
<td>PETALLENGTH</td>
<td>PETALWIDTH</td>
</tr>
</tbody>
</table>

(6 rows)

**Related Functions**

► category Analytics - Regression
► MTBNET_GROW

---

**MTBNET_GROW - Build a multi-tree bayesian network for correlations**

The procedure creates multiple tree-shaped bayesian networks for distinct values of the class

**Usage**

The MTBNET_GROW stored procedure has the following syntax:
MTBNET_GROW(nvarchar(ANY) paramString)

Parameters

- **paramString**
  - input parameters specification
  - Type: NVARCHAR(ANY)

- **intable**
  - table name - the table with data to grow the network
  - Type: nvarchar(ANY)

- **incolomn**
  - The input table columns with special properties, separated by a semi-colon (;).
  - Each column is followed by one or several of the following properties:
  - type: ':nom' (for nominal), ':cont' (for continuous). By default, all numerical types are continuous, other types are nominal
  - role: ':id', ':target', ':input', ':ignore'.
  - (Remark: ':objweight' is unsupported, i.e. ':objweight' same as ':ignore').
  - (Remark: ':colweight(<wgt>)' is unsupported, i.e. ':colweight(<wgt>)' same as ':colweight(1)' same as ':input').
  - If the parameter is undefined, all columns of the input table have default properties. Note that this procedure only accepts continuous columns with role 'input'.
  - Type: NVARCHAR(ANY)
  - Default: <none>

- **coldeftype**
  - The default type of the input table columns. Valid values are 'nom' and 'cont'.
  - If the parameter is undefined, all numeric columns are considered continuous, other columns nominal.
  - Type: NVARCHAR(ANY)
  - Default: <none>

- **coldefrole**
  - The default role of the input table columns. Allowed values are 'input' and 'ignore'.
  - If the parameter is undefined, all columns are considered 'input' columns.
  - Type: NVARCHAR(ANY)
  - Default: <none>

- **colPropertiesTable**
  - The input table where column properties for the input table columns are stored. The format of this table is the output format of the nza..COLUMN_PROPERTIES() stored procedure.
  - If the parameter is undefined, the input table column properties is detected automat-
(Remark: colPropertiesTable with "COLROLE" column with value 'objweight' is unsupported, i.e. same as 'ignore')

(Remark: colPropertiesTable with "COLWEIGHT" column with value '<wgt>' is unsupported, i.e. same as '1')

Type: NVARCHAR(256)
Default: <none>

► baseidx
the numeric id to be assigned to the first variable, for easier internal management - for future use
Type: int4

► samplesize
the size of the sample to take if the number of records is too large (for future use).
Type: int4

► talk
if yes then additional information on progress will be displayed. - for future use
Type: NVARCHAR(ANY)

► class
- the target class ; this should be a nominal variable
Type: NVARCHAR(ANY)

► edgelabsort
if edgelabsort=yes then the left end of the edge will have a name lower in alphabetic order than the right one
Type: NVARCHAR(ANY)
Default: <none>

► model
result table name with the model stored
Type: nvarchar(ANY)

▲ Returns
INT - currently 0, in future the count of records in the resulting table

Details

the multiple tree-shaped bayesian network is a set of tree-like bayesian networks such that the tree-structure is developed for each subset of data, identified by the class variable The difference to TAN_Grow is that the structure here can differ for each constituent network, whereas for TAN_grow the structure of the network is everywhere the same and only the correlation levels differ. With MTBNet_diff you can see which edges of the trees differ between various groups (class). The resultant model can be consumed by the TAN_apply procedure just like any TAN model.
Examples

call nza..drop_model('model=struc_iris');
call
nza..MTBNet_Grow('intable=nza..iris,class=class,incolumn=
SEPALLENGTH; SEPALWIDTH; PETALLENGTH;
PETALWIDTH,model=struc_iris,coldefrole=ignore');
select * from NZA_META_STRUC_IRIS_model order by
grouped_on,varxname,varyname;

<table>
<thead>
<tr>
<th>GROUPED_ON</th>
<th>BNID</th>
<th>VARXID</th>
<th>VARXNAME</th>
<th>VARXMEAN</th>
<th>VARXSTDDEV</th>
<th>VARYID</th>
<th>VARYNAME</th>
<th>VARYMEAN</th>
<th>VARYSTDDEV</th>
<th>CORR</th>
</tr>
</thead>
<tbody>
<tr>
<td>+------------------</td>
<td>--------</td>
<td>+-------------</td>
<td>+----------</td>
<td>+----------</td>
<td>+------------</td>
<td>+--------</td>
<td>+-----------</td>
<td>+----------</td>
<td>+------------</td>
<td>+------</td>
</tr>
</tbody>
</table>
Related Functions
► category Analytics - Regression
► TANET_APPLY
► MTBNET_DIFF
► TANET_GROW
► TBNET_GROW
► CORRELATION1000MATRIX

MUTUALINFO - Mutual Information
This stored procedure calculates the mutual information of two columns

Usage
The MUTUALINFO stored procedure has the following syntax:

► MUTUALINFO(NVARCHAR(ANY) paramString)

▲ Parameters
► paramString
comma-separated list of <parameter>=<value> entries with parameters below
Type: NVARCHAR(ANY)

► intable
the input table
Type: NVARCHAR(256)

► incolumn
the two input table columns separated by a semicolon (;)
Type: NVARCHAR(ANY)

► by
the input table column which splits the data into groups for which the operation is to be performed. If specified, an output table must be indicated.
Type: NVARCHAR(128)
Default: <none>

► outtable
the output table to write the mutual information into. This parameter must be specified if parameter by is specified.
Type: NVARCHAR(256)

Default: <none>

▲ Returns
DOUBLE the mutual information of the input table columns, or the number of groups in
the output table for which mutual information has been calculated

Details
This stored procedure calculates the mutual information of two input columns, either in the whole
input table or within the groups defined in the column specified by parameter <by>. Mutual In-
formation is a measure saying how easily it is to predict the value of one column from the value of
the other column. It takes a positive value, the lower the better predictability.

If parameter 'by' is specified, an output table must be specified. The output table is created with
following columns: over_<by>, mutualinfo. The mutual information of the input columns is given
for each group of the <by> column.

Examples

CALL nza..MUTUALINFO('intable=nza..CensusIncome,
incolumn=age;wage_per_hour');


MUTUALINFO

----------------
0.11860701863848
(1 row)

CALL nza..MUTUALINFO('intable=nza..CensusIncome,
incolumn=age;wage_per_hour, outtable=resultTable,
by=sex');

SELECT * FROM resultTable ORDER BY over_sex;

CALL nza..DROP_TABLE('resultTable');

MUTUALINFO

-------------
2
(1 row)

MUTUALINFO | OVER SEX
Related Functions
► category Analytics - Statistics
► COND_ENTROPY
► ENTROPY
► JOINT_ENTROPY
► MUTUALINFO_AGG

MUTUALINFO_AGG - Mutual Information
This function calculates the mutual information of two input variables

Usage
The MUTUALINFO_AGG stored procedure has the following syntax:

► MUTUALINFO_AGG(INT4 X, INT4 Y)
  ▲ Parameters
  ▶ X
    the first variable
    Type: INT4
  ▶ Y
    the second variable
    Type: INT4
  ▲ Returns
    DOUBLE the mutual information, unless too few data (then null returned)

Details
This function calculates the mutual information of two input variables. Mutual Information is a measure saying how easily it is to predict the value of one column from the value of the other column. It takes a positive
value, the lower the better predictability.

Examples

```sql
SELECT nza..MUTUALINFO_AGG(
    nza.._cf_utl_discrete(petallength,
    (SELECT MIN(petallength) FROM nza..iris),
    (SELECT MAX(petallength) FROM nza..iris),
    10),
    nza.._cf_utl_discrete(sepallength,
    (SELECT MIN(sepallength) FROM nza..iris),
    (SELECT MAX(sepallength) FROM nza..iris),
    10)
) FROM nza..iris;
```

```sql
SELECT -log(sqrt(1-c0*c0))/log(2.)
FROM (SELECT nza..CORR_AGG(sepallength,petallength) AS c0
FROM nza..iris) beta;
```

```
MUTUALINFO_AGG
-----------------
1.2966720934458
(1 row)
```

```
?COLUMN?
-----------------
1.0293125384505
(1 row)
```

Related Functions

- category Analytics - Statistics
- MUTUALINFO

MWW_TEST - Mann-Whitney-Wilcoxon test of independence
This procedure executes the Mann-Whitney-Wilcoxon test on an input column, whose values are split into two classes, to evaluate the significance of the difference of the class mean values.

**Usage**

The MWW_TEST stored procedure has the following syntax:

```
MWW_TEST(NVARCHAR(ANY) paramString)
```

**Parameters**

- **paramString**: comma-separated list of `<parameter>=<value>` entries with parameters below.
  
  Type: NVARCHAR(ANY)

- **intable**: the input table.
  
  Type: NVARCHAR(256)

- **inColumn**: the input table column. It does not need to be numerical, but must be ordered in its domain.
  
  Type: NVARCHAR(128)

- **class**: the input table column which splits data into two classes. The class column name is followed by two class values preceded by a colon (:).
  
  Type: NVARCHAR(ANY)

- **by**: the input table column which splits the data into groups for which the operation is to be performed. If specified, an output table must be specified too.
  
  Type: NVARCHAR(128)
  
  Default: <none>

- **outTable**: the output table to write the MWW statistics into. This parameter is required if parameter by is specified. Otherwise the parameter is ignored.
  
  Type: NVARCHAR(ANY)
  
  Default: <none>

**Returns**

NVARCHAR(200) The test results or the number of rows in the output table. The test result can contain following results: uStat, u2Stat, mu_U, sigma_U, zStat, pp, lower. See the output table column description for more information on them.

**Details**

This stored procedure executes the Mann-Whitney-Wilcoxon test on a column whose values are split into two classes, either in the whole input table or within the groups defined in the column specified by parameter <by>. The Mann-Whitney-Wilcoxon test is a statistical hypothesis test that determines whether one of two samples of independent observations tends to have larger values than the other. This is indicated by
the component of the output called pp.

The Mann-Whitney-Wilcoxon test is sometimes also referred to as: (1) The Mann-Whitney U test, (2) MWW, (3) The Wilcoxon rank-sum test.

The output table is created with following columns: n, n1, n2, ustat0, u2stat0, alphasum, norm1, ustat1, norm2, u2stat1, mu_u1, sigma_u, zstat, pp, ustat, u2stat, mu_u, nogroups, message, lower. If the parameter by is specified, an additional column <by> is added to indicate for which group the MWW test has been calculated. If pp < 0.05, then one of the two classes tends to have larger values than the other. The class which tends to have lower-ranked values is indicated.

Examples

```sql
CALL nza..MWW_TEST('intable=nza..adult, incolumn=HOURS_PER_WEEK, class=sex');

MWW_TEST
---------------------------------------------------------
---------------------------------------------------------
-------------------
nStat= 71896628.204303 n2Stat= 162803461.7957 mu_U= 117350045
sigma_U= 798032.67275233 zStat= -56.95683691613
pp= 0 (lower: Female)
(1 row)
```

```sql
CALL nza..MWW_TEST('intable=nza..adult, incolumn=HOURS_PER_WEEK, class=sex, by=education, outtable=resmww_05');

SELECT education, pp, lower, message FROM resmww_05 ORDER BY pp;

CALL nza..DROP_TABLE('resmww_05');

MWW_TEST
----------
5
(1 row)

EDUCATION | PP | LOWER | MESSAGE
-----------+-----+--------+---------
10th | 1.1653437612569e-19 | Female | O.K.
```
11th | 8.1862589474572e-18 | Female | O.K.
12th | 8.4349399121461e-08 | Female | O.K.
1st-4th | 8.5380864800925e-06 | Female | O.K.
5th-6th | 0.00011748909011573 | Female | O.K.
(5 rows)

DROP_TABLE
----------
t
(1 row)

CREATE VIEW addult AS SELECT hours_per_week::nvarchar(30) AS chours_per_week, education, sex from nza..adult;
CALL nza..MWW_TEST('intable=addult, incolumn=chours_per_week, class=sex');
CALL nza..DROP_TABLE('addult');

MWW_TEST
----------

uStat= 74212294.083753 u2Stat= 160487795.91625 mu_U= 117350045 sigma_U= 798032.67275233 zStat= -54.055118780375 pp= 0 (lower: Female)
(1 row)

DROP_TABLE
----------
t
(1 row)

Related Functions
► category Analytics - Statistics
► PMWW
NAIVEBAYES - Build a Naive Bayes model

This stored procedure builds a Naive Bayes model

Usage
The NAIVEBAYES stored procedure has the following syntax:

```sql
NAIVEBAYES(NVARCHAR(ANY) paramString)
```

Parameters

- **paramString**
  comma-separated list of <parameter>=<value> entries with parameters below
  Type: NVARCHAR(ANY)

- **model**
  the Naive Bayes model to build
  Type: NVARCHAR(ANY)

- **intable**
  the input table
  Type: NVARCHAR(256)

- **id**
  the input table column identifying a unique instance id
  Type: NVARCHAR(128)

- **target**
  the input table column representing the class
  Type: NVARCHAR(128)

- **incolumn**
  the input table columns with special properties, separated by a semi-colon (;).

Each column is followed by one or several of the following properties:

- its type: ':nom' (for nominal), ':cont' (for continuous). Per default, all numerical types are continuous, other types are nominal.
- its role: ':id', ':target', ':input', ':ignore'.

(Remark: ':objweight' is unsupported, i.e. ':objweight' same as ':ignore').
(Remark: ':colweight(<wgt>)' is unsupported, i.e. ':colweight(<wgt>)' same as ':colweight(1)' same as ':input').

If the parameter is undefined, all columns of the input table have default properties.
Type: NVARCHAR(ANY)
Default: <none>

► **coldeftype**
default type of the input table columns. Allowed values are 'nom' and 'cont'.
If the parameter is undefined, all numeric columns are considered continuous, other columns nominal.
Type: NVARCHAR(ANY)
Default: <none>

► **coldefrole**
default role of the input table columns. Allowed values are 'input' and 'ignore'.
If the parameter is undefined, all columns are considered 'input' columns.
Type: NVARCHAR(ANY)
Default: <none>

► **colPropertiesTable**
the input table where column properties for the input table columns are stored. The format of
this table is the output format of stored procedure nza..COLUMN_PROPERTIES().
If the parameter is undefined, the input table column properties will be detected automatically.
(Remark: colPropertiesTable with "COLROLE" column with value 'objweight' is unsupported, i.e.
same as 'ignore')
(Remark: colPropertiesTable with "COLWEIGHT" column with value '<wgt>' is unsupported, i.e.
same as '1')
Type: NVARCHAR(256)
Default: <none>

► **disc**
discretization type for numeric columns [ew, ef, em]
Type: NVARCHAR(ANY)
Default: 'ew'

► **bins**
default number of bins for numeric columns
Type: INTEGER
Default: '10'

▲ Returns
INTEGER the number of (column, value, class) statistics in the model

Details
This stored procedure builds a Naive Bayes model. The model is saved to the database in a set of tables and
registered in the database model metadata. Use the Model Management functions to further manipulate
the model or access the model tables.

Building a Naive Bayes model works only on nominal <target> column. Only VARCHAR and NVARCHAR
columns are treated as nominal attributes. Discretization is applied on numeric attributes according to the
The output of the building process is a contingency table that gathers statistics about the class labels per attribute and attribute value. The contingency table has following columns: attribute name, val, class, classvalcount, classcount, totalcount.

This algorithm also supports missing data.

Examples

```
CALL nza..NAIVEBAYES('model=NB_soybean, intable=nza..soybean_train, id=instance, target=class');

CALL nza..DROP_MODEL('model=NB_soybean');

---

NAIVEBAYES

1881
(1 row)

DROP_MODEL

---

t
(1 row)

CALL nza..NAIVEBAYES('model=NB_iris, intable=nza..iris, coldeftype=cont, incolumn=PETALWIDTH:nom;PETALLENGTH:ignore;class:nom:targ et;id:id');

CALL nza..DROP_MODEL('model=NB_iris');

---

NAIVEBAYES

126
(1 row)

DROP_MODEL

---

t
(1 row)
```
OUTLIERS - Detect outliers for a numeric attribute

This stored procedure detects outliers of a numeric column.

Usage

The OUTLIERS stored procedure has the following syntax:

```
OUTLIERS(NVARCHAR(ANY) paramString)
```

Parameters

- `paramString`  
  comma-separated list of `<parameter>=<value>` entries with parameters below  
  Type: NVARCHAR(ANY)
- `intable`  
  the input table or view  
  Type: NVARCHAR(256)
- `incolumn`  
  the input table column  
  Type: NVARCHAR(128)
- `outtable`  
  the output table to write outliers to  
  Type: NVARCHAR(ANY)
- `multiplier`  
  the value of the IQR multiplier  
  Type: DOUBLE  
  Default: 1.5

Returns

- INTEGER the number of outliers detected

Details

This stored procedure detects outliers as the values that fall out of the interval \([Q1-m\times(Q3-Q1),Q3+m\times(Q3-Q1)]\). \(Q_1, Q_3\) are the first and third quartiles of the input column.

The output table contains only one column `incolumn` containing the outliers.

Examples

```
CALL nza..OUTLIERS('intable=nza..quant_iris, 
incolumn=sepal_length, multiplier=0.5, 
```
```sql
outtable=quant_outliers');
SELECT * FROM quant_outliers ORDER BY sepal_length;
CALL nza..DROP_TABLE('quant_outliers');

OUTLIERS
---------
  9
(1 row)

SEPAL_LENGTH
-------------
  4.3
  4.4
  7.1
  7.2
  7.3
  7.4
  7.6
  7.7
  7.9
(9 rows)

DROP_TABLE
---------
  t
(1 row)
```

**Related Functions**

- **category Analytics - Quantiles & Outliers**
- **IQR**
PBERN - Cumulative Bernoulli Distribution

Given the success probability p, this function returns the probability that a variable following the Bernoulli distribution takes a value smaller or equal to x. The value x is the number of successes in a single Bernoulli trial.

Usage
The PBERN function has the following syntax:

```plaintext
PBERN(DOUBLE x, DOUBLE p)
```

**Parameters**

- **x**
  the value at which to compute
  Type: DOUBLE

- **p**
  the success probability
  Type: DOUBLE
  Min: 0.00000000001
  Max: 1-0.00000000001

**Returns**

DOUBLE the cumulative probability distribution from minus infinity to point x

Details
A Bernoulli trial models tossing once a manipulated coin that in general does not have the same chance of heads (or success, 1) and tails (or failure, 0). The parameter p indicates the probability of getting heads (1). The Bernoulli distribution is a distribution of a discrete variable that takes values 0 and 1 according to a single Bernoulli trial.

The cumulative probability distribution returns the probability to get an outcome of the trial equal or less than x.
- For x<0, PBERN(x,p) = 0,
- For x>=0 and x<1, PBERN(x,p) = (1-p),
- For x>=1, PBERN(x,p) = 1.

If p is not between 0 and 1, PBERN(x,p) is null for all x values.

Examples

```sql
SELECT nza..PBERN(0.2, 0.3);
```

```
PBERN
-------
0.7
(1 row)
```
SELECT nza..PBERN(0,0.3);

PBERN
-------
  0.7
(1 row)

SELECT nza..PBERN(-1,0.3);

PBERN
-------
  0
(1 row)

SELECT nza..PBERN(1,0.3);

PBERN
-------
  1
(1 row)

**Related Functions**

- category Analytics - Probability Distributions
- DBERN
- PBERN_H
- QBERN

**PBERN_H - Cumulative Bernoulli Distribution, high tail**

Given the success probability $p$, this function returns the probability that a variable following the Bernoulli distribution takes a value greater than $x$. The value $x$ is the number of successes in a single Bernoulli trial.
Usage

The PBERN_H function has the following syntax:

► PBERN_H(DOUBLE x, DOUBLE p)
  ▲ Parameters
    ► x
      the value at which to compute
      Type: DOUBLE
    ► p
      the success probability
      Type: DOUBLE
      Min: 0.00000000001
      Max: 1-0.00000000001
  ▲ Returns
    DOUBLE the cumulative probability distribution from x to plus infinity

Details

PBERN_H(x,p) = 1-PBERN(x,p) for all x values and for p between 0 and 1.
If p is not between 0 and 1, PBERN_H(x,p) is null for all x values.

Examples

SELECT nza..PBERN_H(0.2, 0.3), nza..PBERN(0.2, 0.3);

<table>
<thead>
<tr>
<th>PBERN_H</th>
<th>PBERN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3</td>
<td>0.7</td>
</tr>
</tbody>
</table>
(1 row)

SELECT nza..PBERN_H(0, 0.3), nza..PBERN(0, 0.3);

<table>
<thead>
<tr>
<th>PBERN_H</th>
<th>PBERN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3</td>
<td>0.7</td>
</tr>
</tbody>
</table>
(1 row)

SELECT nza..PBERN_H(-1, 0.3), nza..PBERN(-1, 0.3);
PBETA - Cumulative Beta Distribution

Given two shape parameters, this function returns the probability that a variable following the Beta distribution takes a value smaller or equal to x.

Usage

The PBETA function has the following syntax:

```
PETA(DOUBLE x,DOUBLE shapeOne, DOUBLE shapeTwo)
```

Parameters

- **x**
  - the value at which to compute
  - Type: DOUBLE

- **shapeOne**
  - the first shape of the distribution
  - Type: DOUBLE
  - Min: 0.00000000001

- **shapeTwo**
  - the second shape of the distribution
Type: DOUBLE
Min: 0.00000000001

▲ Returns
DOUBLE the cumulative probability distribution from minus infinity to point x

Details
The Beta distribution is a generic distribution, closely related to other distributions like the Fisher distribution via some closed-form mathematical relations. As the beta distribution has a bounded range with positive density (from 0 to 1), processes with natural lower and upper limits are modelled using it.

if a or b is 0 or less, PBETA_H(x,a,b) is null for all x values.

For density of this distribution see the documentation of DBETA.

Note that PBETA(x,a,b)=1-PBETA(1-x,b,a) for all x and for all positive a and b values.

Note also that PBETA(x,1,1) is a uniform distribution over the range [0,1], PBETA(x,1,1)=PUNIF(x,0,1)

Examples

```
SELECT nza..PBETA(0.0, 3, 0.4),nza..PBETA(0.1, 3, 0.4),nza..PBETA(0.2, 3, 0.4),nza..PBETA(0.3, 3, 0.4),nza..PBETA(0.4, 3, 0.4);
```

| PBETA | PBETA | PBETA | PBETA |
|-------+-------+-------+-------|
| 0.00023477562161745 | 0.0019774546733614 | 0.0070656037462229 | 0.0178553415291 |

(1 row)

Related Functions

- category Analytics - Probability Distributions
- DBETA
- PBETA_H
- PF
- QBETA

PBETA_H - Cumulative Beta Distribution, high tail

Given two shape parameters, this function returns the probability that a variable following the Beta distribution takes a value greater than x

Usage
The PBETA_H function has the following syntax:
PBETA_H(DOUBLE x, DOUBLE shapeOne, DOUBLE shapeTwo)
▲ Parameters
► x
the value at which to compute
  Type: DOUBLE
► shapeOne
the first shape of the distribution
  Type: DOUBLE
  Min: 0.00000000001
► shapeTwo
the second shape of the distribution
  Type: DOUBLE
  Min: 0.00000000001
▲ Returns
DOUBLE the cumulative probability distribution from x to plus infinity (effectively integral from x to 1 if x is below 1, and the integral is 0 if x is above 1)

Details
PBETA_H(x,a,b)=1-PBETA(x,a,b), for all x and for all positive a and b values.
if a or b is 0 or less, PBETA_H(x,a,b) is null for all x values.

Examples
SELECT nza..PBETA_H(0.0, 3, 0.4), nza..PBETA_H(0.1, 3, 0.4), nza..PBETA_H(0.2, 3, 0.4), nza..PBETA_H(0.3, 3, 0.4), nza..PBETA_H(0.4, 3, 0.4);

<table>
<thead>
<tr>
<th>PBETA_H</th>
<th>PBETA_H</th>
<th>PBETA_H</th>
</tr>
</thead>
<tbody>
<tr>
<td>--------</td>
<td>---------</td>
<td>---------</td>
</tr>
<tr>
<td>1</td>
<td>0.99976522437838</td>
<td>0.99802254532664</td>
</tr>
<tr>
<td>0.99293439625378</td>
<td>0.9821446584709</td>
<td></td>
</tr>
</tbody>
</table>
(1 row)

Related Functions
► category Analytics - Probability Distributions
► DBETA
PBINOM - Cumulative Binomial Distribution

Given the success probability \( p \) and the number of trials, this function returns the probability that a variable following the Binomial distribution takes a value smaller or equal to \( x \). The value \( x \) is the number of successes in a serie of Bernoulli trials.

**Usage**

The PBINOM function has the following syntax:

```
PBINOM(DOUBLE x, INT8 N, DOUBLE p)
```

**Parameters**

- **x**
  the value at which to compute
  Type: DOUBLE

- **N**
  the number of trials
  Type: INT8
  Min: 1

- **p**
  the success probability
  Type: DOUBLE
  Min: 0.0000000001
  Max: 1-0.0000000001

**Returns**

DOUBLE the cumulative probability distribution from minus infinity to point \( x \)

**Details**

A Bernoulli trial models tossing once a manipulated coin that in general does not have the same chance of heads (or success, 1) and tails (or failure, 0). The parameter \( p \) indicates the probability of getting heads (1). The cumulative Binomial distribution returns the probability that in a serie of \( N \) Bernoulli trials you will have \( x \) or less successes.

If \( x \) is 0 or less, PBINOM(\( x,N,p \)) = 0 for all positive \( N \) values and \( p \) between 0 and 1. If \( x \) is greater or equal to \( N \), PBINOM(\( x,N,p \)) = 1 for all positive \( N \) values and \( p \) between 0 and 1.

The Bernoulli distribution is a special case of Binomial distribution: PBERN(\( x,p \))=PBINOM(\( x,1,p \)) for all \( x \) values and \( p \) between 0 and 1.

If \( N \) is 0 or less, PBINOM(\( x,N,p \)) is null for all \( x \) and \( p \) values.

If \( p \) is not between 0 and 1, PBINOM(\( x,N,p \)) is null for all \( x \) and \( N \) values.
Examples

```
SELECT nza..PBINOM(0,3,0.4), nza..PBINOM(1,0,3,0.4),
nza..PBINOM(2,0,3,0.4), nza..PBINOM(3,0,3,0.4),
nza..PBINOM(4,3,0,4);

PBINOM | PBINOM | PBINOM | PBINOM | PBINOM
--------+--------+--------+--------+--------
  0.216 |  0.648 |  0.936 |      1 |      1
(1 row)
```

```
SELECT nza..PBINOM(0,3,0.6), nza..PBINOM(1,0,3,0.6),
nza..PBINOM(2,0,3,0.6), nza..PBINOM(3,0,3,0.6),
nza..PBINOM(4,3,0,6);

PBINOM | PBINOM | PBINOM | PBINOM | PBINOM
--------+--------+--------+--------+--------
  0.064 |  0.352 |  0.784 |      1 |      1
(1 row)
```

```
SELECT nza..PBINOM(3,5,0.6), nza..PBINOM(30,50,0.6),
nza..PBINOM(300,500,0.6), nza..PBINOM(3000,5000,0.6),
nza..PBINOM(30000,50000,0.6),
nza..PBINOM(300000,500000,0.6),
nza..PBINOM(3000000,5000000,0.6);

PBINOM | PBINOM | PBINOM | PBINOM | PBINOM
--------+--------+--------+--------+--------
  0.66304 | 0.55352362509135 | 0.51698820499616 | 0.5053741706129 | 0.50169957078773 | 0.50053750675188 | 0.50005361256692
(1 row)
```
Related Functions

- category Analytics - Probability Distributions
- DBINOM
- PBERN
- PBINOM_H
- QBINOM

PBINOM_H - Cumulative Binomial Distribution, high tail

Given the success probability p and the number of trials, this function returns the probability that a variable following the Binomial distribution takes a value greater than x. The value x is the number of successes in a serie of Bernoulli trials.

Usage

The PBINOM_H function has the following syntax:

PBINOM_H(DOUBLE x, INT8 N, DOUBLE p)

Parameters

- x
  the value at which to compute
  Type: DOUBLE
- N
  the number of trials
  Type: INT8
  Min: 1
- p
  the success probability
  Type: DOUBLE
  Min: 0.00000000001
  Max: 1-0.00000000001

Returns

DOUBLE the cumulative probability distribution from x to plus infinity

Details

The function PBINOM_H returns the probability that in a serie of N Bernoulli trials you will have more than x successes.

PBINOM_H(x,N,p) = 1-PBINOM(x,N,p) for all x and all positive N values and for p between 0 and 1.

If N is 0 or less, PBINOM_H(x,N,p) is null for all x and p values.

If p is not between 0 and 1, PBINOM_H(x,N,p) is null for all x and N values.
Examples

SELECT nza..PBINOM_H(0,3,0.4), nza..PBINOM_H(1,3,0.4),
       nza..PBINOM_H(2,3,0.4), nza..PBINOM_H(3,3,0.4),
       nza..PBINOM_H(4,3,0.4);

| PBINOM_H | PBINOM_H | PBINOM_H | PBINOM_H | PBINOM_H |
|----------+----------+----------+----------+----------|
| 0.784    | 0.352    | 0.064    | 0        | 0        |

(1 row)

SELECT nza..PBINOM_H(0,3,0.6), nza..PBINOM_H(1,3,0.6),
       nza..PBINOM_H(2,3,0.6), nza..PBINOM_H(3,3,0.6),
       nza..PBINOM_H(4,3,0.6);

| PBINOM_H | PBINOM_H | PBINOM_H | PBINOM_H | PBINOM_H |
|----------+----------+----------+----------+----------|
| 0.936    | 0.648    | 0.216    | 0        | 0        |

(1 row)

SELECT nza..PBINOM_H(3,5,0.6), nza..PBINOM_H(30,50,0.6),
       nza..PBINOM_H(300,500,0.6), nza..PBINOM_H(3000,5000,0.6),
       nza..PBINOM_H(30000,50000,0.6),

| PBINOM_H | PBINOM_H | PBINOM_H | PBINOM_H | PBINOM_H |
|----------+----------+----------+----------+----------|
| 0.33696  | 0.44647637921055 | 0.48301175754114 | 0.49462586651671 | 0.49830048721356 | 0.49946256442591 | 0.4998300410389 | 0.49994631333962 |

(1 row)
Related Functions

- category Analytics - Probability Distributions
- DBINOM
- PBINOM

PCA - Build a Principal Component Analysis model

This stored procedure builds a Principal Component Analysis model

Usage

The PCA stored procedure has the following syntax:

\[
\text{PCA(NVARCHAR(ANY)} \text{ paramString)}
\]

Parameters

- **paramString**
  
  comma-separated list of \(<\text{parameter}>=\text{value}\> \text{ entries with parameters below}
  
  Type: NVARCHAR(ANY)

- **model**
  
  the name of the Principal Component Analysis model to build
  
  Type: NVARCHAR(ANY)

- **intable**
  
  the input table
  
  Type: NVARCHAR(256)

- **id**
  
  the input table column identifying a unique instance id
  
  Type: NVARCHAR(128)

- **target**
  
  the input table column representing a class or a value to predict, this column is ignored by the PCA.
  
  Type: NVARCHAR(128)
  
  Default: <none>

- **inColumn**
  
  the input table columns with special properties, separated by a semi-colon (;).
  
  Each column is followed by one or several of the following properties:
  
  - its type: ':nom' (for nominal), ':cont' (for continuous). Per default, all numerical types are continuous, other types are nominal.
  - its role: ':id', ':target', ':input', ':ignore'.

  (Remark: ':objweight' is unsupported, i.e. ':objweight' same as ':ignore').
  
  (Remark: ':colweight(<wgt>)' is unsupported, i.e. ':colweight(<wgt>)' same as ':colweight(1)' same as ':input').
If the parameter is undefined, all columns of the input table have default properties. The PCA stored procedure is executed only on columns of role ':input' and of type ':cont'. Nominal columns and columns of role ':target' are ignored.

Type: NVARCHAR(ANY)
Default: <none>

► **coldeftype**

default type of the input table columns. Allowed values are 'nom' and 'cont'.

If the parameter is undefined, all numeric columns are considered continuous, other columns nominal.

Type: NVARCHAR(ANY)
Default: <none>

► **coldefrole**

default role of the input table columns. Allowed values are 'input' and 'ignore'.

If the parameter is undefined, all columns are considered 'input' columns.

Type: NVARCHAR(ANY)
Default: <none>

► **colPropertiesTable**

the input table where column properties for the input table columns are stored. The format of this table is the output format of stored procedure nza..COLUMN_PROPERTIES().

If the parameter is undefined, the input table column properties will be detected automatically.

(Remark: colPropertiesTable with "COLROLE" column with value 'objweight' is unsupported, i.e. same as 'ignore')

(Remark: colPropertiesTable with "COLWEIGHT" column with value '<wgt>' is unsupported, i.e. same as '1')

Type: NVARCHAR(256)
Default: <none>

► **forceEigensolve**

flag indicating to force using a less accurate but faster method

Type: BOOLEAN
Default: false

► **centerData**

flag indicating to first center the data before building the model. Having data that is not centered violates the Principal Component Analysis assumptions.

Type: BOOLEAN
Default: true
- **scaleData**
  flag indicating to first scale the data before building the model
  
  Type: BOOLEAN
  
  Default: false

- **saveScores**
  flag indicating to save the PCA scores of individual observations
  
  Type: BOOLEAN
  
  Default: false

▲ Returns

BOOLEAN always TRUE

**Details**

This stored procedure builds a Principal Component Analysis model of the data and provides a corresponding transformation into principal components, which can then be applied using the stored procedure nz..PROJECT_PCA.

Input data should be provided as datatable, with observations provided in rows, and attributes in columns. All attributes are assumed to be quantitative. The PCA model can be built using two strategies: SVD decomposition, which is more accurate, but at the expense of speed and memory, or by finding the eigenvectors of the unbiased covariance matrix estimator. If the parameter forceEigensolve is not TRUE, the best strategy, that is, the one providing the most accurate solution based on data size and memory availability is used.

Based on the parameters <centerData and scaleData, the input data can be centered and scaled. In that case the corresponding parameters, as well as the mean and variance estimators, are calculated and become part of the model. Centering and scaling is also performed during the application step.

Data centering (assuring that mean of each attribute is equal to 0) is an important assumption of PCA method - failing to meet it usually causes serious model degradation. Data scaling (assuring that the variance of each attribute is equal to 1) usually provides better approximation of the data in case of the presence of attributes that differ in orders of magnitude. It is equivalent to perform the PCA using the correlation instead of covariance matrix.

In order to express the model being created, the procedure creates a set of matrices, using the model name to construct the matrix name. The set consists of following matrices:

- **NZA_META_{modelName}_PCA_ATTMEAN** - row vector containing mean values of the attributes (when centerData is TRUE),

- **NZA_META_{modelName}_PCA_ATTSD** - row vector containing standard deviations of the attributes (when scaleData is TRUE),

- **NZA_META_{modelName}_PCA_ATTSD_DIV** - row vector containing reciprocals of non-zero standard deviations of the attributes or value 1 (when scaleData is TRUE),

- **NZA_META_{modelName}_PCA_SDEV** - row vector containing standard deviations of the principal components,

- **NZA_META_{modelName}_PCA** - the matrix of loadings (a matrix whose columns contain the eigenvectors of the covariance matrix),

- **NZA_META_{modelName}_PCA_SCORES** - the matrix of scores containing projections of individual observations to principal components (when saveScores is TRUE).
Examples

```
CALL nza..PCA('model=wq_pca, intable=nza..WineQuality, id=id, scaleData=TRUE, centerData=TRUE, forceEigensolve=FALSE');

CALL nza..DROP_MODEL('model=wq_pca');
```

```
PCA
-----
t
(1 row)
```

```
DROP_MODEL
------------
 t
(1 row)
```

Related Functions

- category Analytics - Data Transformation
- PROJECT_PCA
- LIST_MODELS

PCAUCHY - Cumulative Cauchy Distribution

Given the peak location and the interquartile range, this function returns the probability that a variable following the Cauchy distribution takes a value smaller or equal to x

Usage

The PCAUCHY function has the following syntax:

```
PCAUCHY(DOUBLE x, DOUBLE location, DOUBLE scale)
```

Parameters

- **x**
  - the value at which to compute
  - Type: DOUBLE

- **location**
the location of the peak of the Cauchy distribution
Type: DOUBLE

- scale
  a value corresponding to half of the interquartile range. Smaller values result in a narrower peak.
  Type: DOUBLE
  Min: 0.00000000001

Returns
DOUBLE the cumulative probability distribution from minus infinity to point x

Details
The Cauchy distribution describes the cross-section of resonant nuclear scattering, derived from the probability of a resonant state with a known lifetime. The cumulative distribution has the closed form:
\[ \text{PCAUCHY}(x, \text{location}, \text{scale}) = \frac{1}{\pi} \arctan\left(\frac{x - \text{location}}{\text{scale}}\right) + 0.5. \]

The special property of this distribution is that neither a mean nor a variance can be computed. It may be of interest because, like a normal distribution, it is a "stable distribution", such that a linear combination of two independent variables following it has the same distribution, up to the location and scale parameters.

The Cauchy distribution is sometimes also referred to as the Breit-Wigner distribution or the Lorentz distribution.

If scale=0 or less, PCAUCHY(x, location, scale) is null for all x and location values.

Examples
```
SELECT nza..PCAUCHY(0, 3, 0.4), nza..PCAUCHY(1.0, 3, 0.4), nza..PCAUCHY(2.0, 3, 0.4), nza..PCAUCHY(3, 3, 0.4), nza..PCAUCHY(4, 3, 0.4);
```

<table>
<thead>
<tr>
<th>PCAUCHY</th>
<th>PCAUCHY</th>
<th>PCAUCHY</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.042192463158841</td>
<td>0.062832958189001</td>
<td>0.12111894159084</td>
</tr>
<tr>
<td>0.5</td>
<td>0.87888105840916</td>
<td></td>
</tr>
</tbody>
</table>

(1 row)

Related Functions
- category Analytics - Probability Distributions
- DCAUCHY
- PCAUCHY_H
- QCAUCHY

PCAUCHY_H - Cumulative Cauchy Distribution, high tail
Given the peak location and the interquartile range, this function returns the probability that a variable following the Cauchy distribution takes a value greater than \( x \)

**Usage**

The \( \text{PCAUCHY\_H} \) function has the following syntax:

\[
\text{PCAUCHY\_H}(\text{DOUBLE} \ x, \ \text{DOUBLE} \ \text{location}, \ \text{DOUBLE} \ \text{scale})
\]

▲ **Parameters**

► \( x \)

  the value at which to compute

  Type: DOUBLE

► \( \text{location} \)

  the location of the peak of the Cauchy distribution

  Type: DOUBLE

► \( \text{scale} \)

  a value corresponding to half of the interquartile range. Smaller values result in a narrower peak.

  Type: DOUBLE

  Min: 0.00000000001

▲ **Returns**

DOUBLE the cumulative probability distribution from \( x \) to plus infinity

**Details**

\( \text{PCAUCHY\_H}(x,\text{location},\text{scale})=1-\text{PCAUCHY}(x,\text{location},\text{scale}) \) for all \( x \) and location values and for scale between 0 and 1.

If scale is not between 0 and 1, \( \text{PCAUCHY\_H}(x,\text{location},\text{scale}) \) is null for all \( x \) and location values.

**Examples**

```
SELECT nza..PCAUCHY_H(0, 3, 0.4), nza..PCAUCHY_H(1.0, 3, 0.4), nza..PCAUCHY_H(2.0, 3, 0.4), nza..PCAUCHY_H(3, 3, 0.4), nza..PCAUCHY_H(4, 3, 0.4);
```

<table>
<thead>
<tr>
<th>PCAUCHY_H</th>
<th>PCAUCHY_H</th>
<th>PCAUCHY_H</th>
<th>PCAUCHY_H</th>
<th>PCAUCHY_H</th>
</tr>
</thead>
<tbody>
<tr>
<td>+-----------+-----------+-----------+-----------+-----------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.95780753684116</td>
<td>0.937167041811</td>
<td>0.87888105840916</td>
<td>0.5</td>
<td>0.12111894159084</td>
</tr>
</tbody>
</table>

(1 row)
PCHISQ - Cumulative Chi-square Distribution

Given a degree of freedom, this function returns the probability that a variable following the Chi-square distribution takes a value smaller or equal to x.

Usage

The PCHISQ function has the following syntax:

```
PCHISQ(DOUBLE x, INT8 df)
```

### Parameters

- **x**
  - the value at which to compute
  - Type: DOUBLE
  - Min: 0.00000000001

- **df**
  - the number of degrees of freedom
  - Type: INT8
  - Min: 1

### Returns

DOUBLE the cumulative probability distribution from minus infinity to point x

### Details

If you have variables X1,...,Xn each of them following standard normal distribution, the variable X=X1^2+X2^2+...+Xn^2 follows the chi-square distribution with n degrees of freedom.

The Chi-square cumulative probability distribution has the form PCHISQ(x,df)=PGAMMA(x/2,df/2).

If df is 0 or less, PCHISQ(x,df) is null for all x values.

### Examples

```
SELECT nza..PCHISQ(4.6188, 9);
PCHISQ
------------------
0.13380702913458  (1 row)
```
Related Functions
► category Analytics - Probability Distributions
► DCHISQ
► PCHISQ_H
► PCHISQ_S
► PGAMMA
► QCHISQ

PCHISQ_H - Cumulative Chi-square Distribution, high tail
Given a degree of freedom, this function returns the probability that a variable following the Chi-square distribution takes a value greater than x

Usage
The PCHISQ_H function has the following syntax:

► PCHISQ_H(DOUBLE x, INT8 df)
▲ Parameters
► x
  the value at which to compute
  Type: DOUBLE
  Min: 0.00000000001
► df
  the number of degrees of freedom
  Type: INT8
  Min: 1
▲ Returns
  DOUBLE the cumulative probability distribution from x to plus infinity

Details
PCHISQ_H(x,df)=1-PCHISQ(x,df) for all x and df values.
If df is 0 or less, PCHISQ_H(x,df) is null for all x values.

Examples
SELECT nza..PCHISQ_H(4.6188, 9);
  PCHISQ_H
PCHISQ_S - Cumulative Chi-square Distribution with string argument

Given a degree of freedom, this function returns the probability that a variable following the Chi-square distribution takes a value greater than x. It differs from PCHISQ in that it takes both parameters as a single string.

Usage

The PCHISQ_S function has the following syntax:

- `PCHISQ_S(VARCHAR(200) paramString)
  
  Parameters
  - `paramString` comma-separated list of <parameter>=<value> entries with parameters below
    Type: VARCHAR(200)
  - `chi2Stat` the value at which to compute
    Type: DOUBLE
    Min: 0.000000000001
  - `df` the number of degrees of freedom
    Type: INT8
    Min: 1
  
  Returns
  DOUBLE the cumulative probability distribution from minus infinity to point x

Details

PCHISQ_S('chi2stat=x,df=df')=PCHISQ(x,df) for all <x> and <df> values.
Examples

```sql
SELECT nza..PCHISQ_S('chi2Stat=10.83, df=1');

  PCHISQ_S
------------------
  0.99900131362082
(1 row)

SELECT nza..PCHISQ_S(
  nza..CHISQ_TEST_S_AGG(
    case when sex='Female' then 0 when sex='Male' then 1 else -1 end,
    case
      when race='Amer Indian Aleut or Eskimo' then 1
      when race='Black' then 2
      when race='White' then 3
      when race='Asian or Pacific Islander' then 4
      when race='Other' then 5
      else -1 end
  )) FROM nza..censusincome;

  PCHISQ_S
------------------
  0.99999885498064
(1 row)
```

```sql
SELECT nza..PCHISQ_S(
  nza..CHISQ_TEST_S_AGG(
    case when sex='Female' then 0 when sex='Male' then 1 else -1 end,
    case
      when race='Amer Indian Aleut or Eskimo' then 1
  )) FROM nza..censusincome;

  PCHISQ_S
------------------
  0.99999885498064
(1 row)
```
when race='Black' then 2
when race='White' then 3
when race='Asian or Pacific Islander' then 4
when race='Other' then 5
else -1 end
)
,nza..CHISQ_TEST_S_AGG(
case when sex='Female' then 0 when sex='Male' then 1 else -1 end,
case
when race='Amer Indian Aleut or Eskimo' then 1
when race='Black' then 2
when race='White' then 3
when race='Asian or Pacific Islander' then 4
when race='Other' then 5
else -1 end
)
FROM nza..censusincome;

<table>
<thead>
<tr>
<th>PCHISQ_S</th>
<th>CHISQ_TEST_S_AGG</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.99999885498064</td>
<td>chi2Stat= 101.156, df= 4</td>
</tr>
</tbody>
</table>

(1 row)

SELECT nza..PCHISQ_S('chi2Stat=4.6188, df= 9');

<table>
<thead>
<tr>
<th>PCHISQ_S</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.13380701958885</td>
</tr>
</tbody>
</table>

(1 row)

Related Functions

category Analytics - Probability Distributions
PERCENTAGE_SPLIT - Build and evaluate a Classification model on a randomly split table

This stored procedure builds a Classification model on a part of the input data and evaluates its prediction quality on the rest of the data.

Usage

The PERCENTAGE_SPLIT stored procedure has the following syntax:

PERCENTAGE_SPLIT(NVARCHAR(ANY) paramString)

Parameters

- paramString
  comma-separated list of <parameter>=<value> entries with parameters below
  Type: NVARCHAR(ANY)

- modelType
  the name of the procedure that builds a Classification model (e.g. naivebayes, dectree, knn)
  Type: NVARCHAR(ANY)

- model
  the name of the Classification model to build
  Type: NVARCHAR(ANY)

- intable
  the input table
  Type: NVARCHAR(256)

- id
  the input table column identifying a unique instance id
  Type: NVARCHAR(128)

- target
  the input table column representing the class
  Type: NVARCHAR(128)

- outtable
  the output table where the predictions will be stored
  Type: NVARCHAR(256)
fraction
the fraction of the data to build the model on
Type: FLOAT
Default: 0.5
Min: 0.0
Max: 1.0

seed
the seed of the random function
Type: FLOAT
Default: random()

Returns
FLOAT prediction accuracy and -1 if accuracy cannot be calculated

Details
This stored procedure builds a Classification model. The procedure randomly selects the given fraction of
the input records to build the model, the rest of the records is used to evaluate the prediction quality of the
model. The model is saved to the database in a set of tables and registered in the database model
metadata. Use the Model Management functions to further manipulate the model or access the model
tables.

All parameters specific to a the given Classification stored procedure can be defined too: they will be used
when building and testing the model.

A table <outtable> is created with the following columns: id, class. The column id matches the <id> column
from intable and class is the predicted class label. The size of this table is the same as the size of <intable>.

Examples

CALL nza..PERCENTAGE_SPLIT('modelType=dectree,
intable=nza..iris, fraction=0.6, model=iris_c45, target=class,
id=id, outtable=iris_pred, seed=12345');

CALL nza..DROP_MODEL('model=iris_c45');

CALL nza..DROP_TABLE('iris_pred');

PERCENTAGE_SPLIT
------------------
0.9333333
(1 row)

DROP_MODEL
------------
t
(1 row)
DROP_TABLE

---

t

(1 row)

Related Functions

- category Analytics - Classification
- CROSS_VALIDATION
- TRAIN_TEST
- DECTREE
- NAIVEBAYES
- KNN
- LIST_MODELS

PEXP - Cumulative Exponential Distribution

Given an exponential scale, this function returns the probability that a variable following the Exponential distribution takes a value smaller or equal to x

Usage

The PEXP function has the following syntax:

- PEXP(DOUBLE x, DOUBLE scale)

  ▲ Parameters
  
  - x
    
    the value at which to compute
    
    Type: DOUBLE
    
    Min: 0.00000000001
  
  - scale
    
    the exponential scale
    
    Type: DOUBLE
    
    Min: 0.00000000001

  ▲ Returns
  
  DOUBLE the cumulative probability distribution from minus infinity to point x

Details

The cumulative exponential distribution function is given in closed form: PEXP(x, scale) = 1 - exp(- (x/scale)) for x greater than 0.
The exponential distribution may be encountered in behavior of technical systems for example in case of time between failures, provided that the probability of failure is very low and does not change over time. If x is negative, PEXP(x, scale) is null for all scale values. If scale is 0 or less, PEXP(x, scale) is null for all x values.

Examples

```
SELECT nza..PEXP (2, 4);

PEXP
------------------
  0.39346934028737
(1 row)
```

```
SELECT nza..PEXP (1e-20, 1);

PEXP
-------
  1e-20
(1 row)
```

Related Functions

► category Analytics - Probability Distributions
► DEXP
► PEXP_H
► QEXP

PEXP_H - Cumulative Exponential Distribution, high tail

Given an exponential scale, this function returns the probability that a variable following the Exponential distribution takes a value greater than x

Usage

The PEXP_H function has the following syntax:

```
PEXP_H (DOUBLE x, DOUBLE scale)
```

▲ Parameters

► x
  the value at which to compute
Type: DOUBLE
Min: 0.00000000001

scale
the exponential scale
Type: DOUBLE
Min: 0.00000000001

Returns
DOUBLE the cumulative probability distribution from x to plus infinity

Details
PEXP_H(x,scale)=1-PEXP(x,scale) for all x and df values.
If x is negative, PEXP_H(x,scale) is null for all scale values.
If scale is 0 or less, PEXP_H(x,scale) is null for all x values.

Examples

```
SELECT nza..PEXP_H(2,4),nza..PEXP(2,4);
```

<table>
<thead>
<tr>
<th>PEXP_H</th>
<th>PEXP</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.60653065971263</td>
<td>0.39346934028737</td>
</tr>
</tbody>
</table>

(1 row)

Related Functions

- category Analytics - Probability Distributions
- DEXP
- PEXP

PF - Cumulative Fisher Distribution

Given the degrees of freedom of the nominator and of the denominator, this function returns the probability that a variable following the Fisher distribution takes a value smaller or equal to x

Usage

The PF function has the following syntax:

```
PF(DOUBLE x, INT8 dfNomin, INT8 dfDenom)
```

Parameters
x
the value at which to compute
Type: DOUBLE
Min: 0.00000000001

► dfNomin
the number of degrees of freedom of the nominator
Type: INT8
Min: 1

► dfDenom
the number of degrees of freedom of the denominator
Type: INT8
Min: 1

▲ Returns
DOUBLE the cumulative probability distribution from minus infinity to point x

Details
The Fisher distribution, called also Fisher-Snedecor distribution or Fisher F-Distribution, describes the distribution of a quotient of two variables following a Chi-square distribution, both scaled by their number of degrees of freedom (dfNomin, dfDenom respectively).

If x, dfNomin or dfDenom is 0 or less, PF(x,dfNomin,dfDenom) is null.

Examples

```
SELECT nza..PF(9.55, 2, 3);
```

```
PF
-----------------
0.9499857832907
(1 row)
```

Related Functions
► category Analytics - Probability Distributions
► DF
► PF_H
► QF

PF_H - Cumulative Fisher Distribution, high tail
Given the degrees of freedom of the nominator and of the denominator, this function returns the probability that a variable following the Fisher distribution takes a value greater than x
Usage

The PF_H function has the following syntax:

- **PF_H**(DOUBLE $x$, INT8 $dfNomin$, INT8 $dfDenom$)

  ▲ Parameters
  - **$x$**
    - the value at which to compute
    - Type: DOUBLE
    - Min: 0.00000000001
  - **$dfNomin$**
    - the number of degrees of freedom of the nominator
    - Type: INT8
    - Min: 1
  - **$dfDenom$**
    - the number of degrees of freedom of the denominator
    - Type: INT8
    - Min: 1

  ▲ Returns
  - DOUBLE the cumulative probability distribution from $x$ to plus infinity

Details

$PF_H(x,dfNomin,dfDenom)= 1-PF(x,dfNomin,dfDenom)$ for all positive $x$, $dfNomin$ and $dfDenom$ values.

If $x$, $dfNomin$ or $dfDenom$ is 0 or less, $PF(x,dfNomin,dfDenom)$ is null.

Examples

```sql
SELECT nza..PF_H(9.55, 2, 3);
```

```
PF_H
-------------------
0.050014216709295
(1 row)
```

Related Functions

- category Analytics - Probability Distributions
- DF
PFISK - Cumulative Fisk (or log-logistic) Distribution

Given the median and the shape, this function returns the probability that a variable following the Fisk distribution takes a value smaller or equal to x

Usage

The PFISK function has the following syntax:

► PFISK\( (\text{DOUBLE } x, \text{DOUBLE median, DOUBLE shape}) \)

▲ Parameters

► x
the value at which to compute
Type: DOUBLE
Min: 0.000000001

► median
the median value of the distribution (also called scale)
Type: DOUBLE
Min: 0.000000001

► shape
the shape of the distribution
Type: DOUBLE
Min: 0.000000001

▲ Returns
DOUBLE the cumulative probability distribution from minus infinity to point \( x \)

Details

The Fisk distribution may be used in hydrology in models of precipitation or stream flow rates.

\( \text{PFISK}(x, \text{median, shape}) = x^{\text{shape}}/(\text{median}^{\text{shape}}+x^{\text{shape}}) \) for all positive \( x, \text{median} \) and \( \text{shape} \) values.

If \( x, \text{median} \) or \( \text{shape} \) is 0 or less, \( \text{PFISK}(x, \text{median, shape}) \) is null.

Examples

\[
\text{SELECT nza..PFISK(2.43,5,8)};
\]

\[
\begin{array}{l}
\text{PFISK} \\
0.0031027056133611 \\
\end{array}
\]

(1 row)
SELECT nza..PFISK(5,5,8);

PFISK
--------
0.5
(1 row)

SELECT nza..PFISK(2.43,2,3);

PFISK
----------------
0.642040659975
(1 row)

Related Functions
► category Analytics - Probability Distributions
► DFISK
► PFISK_H
► QFISK
► QFISK_H

PFISK_H - Cumulative Fisk (or log-logistic) Distribution, high tail
Given the median and the shape, this function returns the probability that a variable following the Fisk distribution takes a value greater than x

Usage
The PFISK_H function has the following syntax:

► PFISK_H(DOUBLE x, DOUBLE median, DOUBLE shape)
▲ Parameters
► x
  the value at which to compute
  Type: DOUBLE
  Min: 0.000000001
► median
the median value of the distribution (also called scale)
Type: DOUBLE
Min: 0.000000001

► shape
the shape of the distribution
Type: DOUBLE
Min: 0.000000001

▲ Returns
DOUBLE the cumulative probability distribution from x to plus infinity

Details
PFISK_H(x,median,shape)= 1-PFISK(x,median,shape) for all positive x, median and shape values.
If x, median or shape is 0 or less, PFISK_H(x,median,shape) is null.

Examples

```sql
SELECT nza..PFISK_H(2.43,5,8), nza..PFISK(2.43,5,8);
```

<table>
<thead>
<tr>
<th>PFISK_H</th>
<th>PFISK</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.9968972943</td>
<td>0.0031027056</td>
</tr>
<tr>
<td>(1 row)</td>
<td></td>
</tr>
</tbody>
</table>

```sql
SELECT nza..PFISK_H(5,5,8), nza..PFISK(5,5,8);
```

<table>
<thead>
<tr>
<th>PFISK_H</th>
<th>PFISK</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>(1 row)</td>
<td></td>
</tr>
</tbody>
</table>

```sql
SELECT nza..PFISK_H(2.43,2,3), nza..PFISK(2.43,2,3);
```

<table>
<thead>
<tr>
<th>PFISK_H</th>
<th>PFISK</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.35795934</td>
<td>0.6420406599</td>
</tr>
<tr>
<td>(1 row)</td>
<td></td>
</tr>
</tbody>
</table>
Related Functions
► category Analytics - Probability Distributions
► DFISK
► PFISK

PGAMMA - Cumulative Gamma Distribution

Given a shape and an inverted scale, this function returns the probability that a variable following the Gamma distribution takes a value smaller or equal to \( x \)

Usage

The PGAMMA function has the following syntax:

► PGAMMA(DOUBLE \( x \), DOUBLE shape, DOUBLE scaleInv)

▲ Parameters
► \( x \)
  the value at which to compute
  Type: DOUBLE
► shape
  the shape of the distribution
  Type: DOUBLE
  Min: 0.0000000001
► scaleInv
  the inverted scale of the distribution
  Type: DOUBLE
  Min: 0.0000000001

▲ Returns
  DOUBLE the cumulative probability distribution from minus infinity to point \( x \)

Details

The Gamma distribution is the distribution of a sum of random variables which are exponentially distributed. This can be observed for example for the time intervals between successes in the Poisson experiment, or with inter-spike intervals.

\[ \text{PGAMMA}(x,1,\text{scaleInv}) = \text{PEXP}(x,1/\text{scaleInv}) \] for all \( x \) and all positive scaleInv values.

If shape or scaleInv is 0 or less, PGAMMA\((x,\text{shape},\text{scaleInv})\) is null for all \( x \) values.

Examples

```
SELECT nza..PGAMMA(0,3,0.4), nza..PGAMMA(1.0,3,0.4),
```
nza..PGAMMA(2.0,3,0.4), nza..PGAMMA(3,3,0.4),
nza..PGamma(4,3,0.4);

<table>
<thead>
<tr>
<th>PGAMMA</th>
<th>PGAMMA</th>
<th>PGAMMA</th>
<th>PGAMMA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0079263318672562</td>
<td>0.047422596071624</td>
<td>0.12051290121633</td>
<td>0.2166415101611</td>
</tr>
</tbody>
</table>
(1 row)

SELECT nza..PGAMMA(3,1,2), nza..PEXP(3,1./2);

<table>
<thead>
<tr>
<th>PGAMMA</th>
<th>PEXP</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.99752124782333</td>
<td>0.99752124782333</td>
</tr>
</tbody>
</table>
(1 row)

Related Functions
► category Analytics - Probability Distributions
► DGAMMA
► PEXP
► PGAMMA_H
► QGAMMA

PGAMMA_H - Cumulative Gamma Distribution, high tail

Given a shape and an inverted scale, this function returns the probability that a variable following the Gamma distribution takes a value greater than x

Usage
The PGAMMA_H function has the following syntax:

► PGAMMA_H(DOUBLE x, DOUBLE shape, DOUBLE scaleInv)
  ▲ Parameters
    ► x
      the value at which to compute
      Type: DOUBLE
    ► shape
the shape of the distribution
Type: DOUBLE
Min: 0.00000000001

scaleInv
the inverted scale of the distribution
Type: DOUBLE
Min: 0.00000000001

Returns
DOUBLE the cumulative probability distribution from minus infinity to point x

Details
PGAMMA_H(x,shape,scaleInv)=1-PGAMMA(x,shape,scaleInv) for all x and all positive shape and scaleInv values.
if shape or scaleInv is 0 or less, PGAMMA_H(x,shape,scaleInv) is null for all x values.

Examples

```
SELECT nza..PGAMMA_H(0,3,0.4), nza..PGAMMA_H(1.0,3,0.4), nza..PGAMMA_H(2.0,3,0.4), nza..PGAMMA_H(3,3,0.4), nza..PGAMMA_H(4,3,0.4);
```

```
PGAMMA_H |     PGAMMA_H     |     PGAMMA_H     |
----------+------------------+------------------
+------------------+-----------------+------------------+------------------
  1   | 0.99207366813274 | 0.95257740392838 |
  0.87948709878747 | 0.7833584898389  |
(1 row)
```

Related Functions
- category Analytics - Probability Distributions
- DGAMMA
- PGAMMA

PGEOM - Cumulative Geometric Distribution

Given the success probability p, this function returns the probability that a variable following the Geometric distribution takes a value smaller or equal to x. The value x is the number of failure before the first success in a serie of Bernoulli trials.
Usage
The PGEOM function has the following syntax:

▶ PGEOM(DOUBLE x, DOUBLE p)
▲ Parameters
▶ x
the value at which to compute
Type: DOUBLE
▶ p
the success probability
Type: DOUBLE
Min: 0.00000000001
Max: 0.99999999999
▲ Returns
DOUBLE the cumulative probability distribution from minus infinity to point x

Details
A Bernoulli trial models tossing once a manipulated coin that in general does not have the same chance of heads (or success, 1) and tails (or failure, 0). The parameter p indicates the probability of getting heads (1). The cumulative Geometric distribution returns the probability that in a serie of Bernoulli trials you will have x or less failures before the first success.

- For x<0, PGEOM(x,p) = 0 for p between 0 and 1,
- For x>=0, PGEOM(x,p) = 1-p*(1-p)^{x+1} for p between 0 and 1.

Examples

```
SELECT nza..PGEOM(0,0.4), nza..PGEOM(1,0.4), nza..PGEOM(2,0.4),
nza..PGEOM(3,0.4);
```

<table>
<thead>
<tr>
<th>PGEOM</th>
<th>PGEOM</th>
<th>PGEOM</th>
<th>PGEOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.4</td>
<td>0.64</td>
<td>0.784</td>
<td>0.8704</td>
</tr>
</tbody>
</table>

(1 row)

```
SELECT nza..PGEOM(6,0.4);
```

```
PGEOM
-------
0.9720064
```
Related Functions

- category Analytics - Probability Distributions
- DGEOM
- PBERN
- PGEOM_H
- QGEOM

PGEOM_H - Cumulative Geometric Distribution, high tail

Given the success probability p, this function returns the probability that a variable following the Geometric distribution takes a value greater than x. The value x is the number of failure before the first success in a serie of Bernoulli trials.

Usage

The PGEOM_H function has the following syntax:

- PGEOM_H(DOUBLE x, DOUBLE p)

  ▲ Parameters
  - x
    the value at which to compute
    Type: DOUBLE
  - p
    the success probability
    Type: DOUBLE
    Min: 0.00000000001
    Max: 0.99999999999

  ▲ Returns
  DOUBLE the cumulative probability distribution from x to plus infinity

Details

The function PGEOM_H returns the probability that in a serie of Bernoulli trials you will have more than x failures before the first success. PGEOM_H(x,p) = 1-PGEOm(x,p) for all x values and for p between 0 and 1.

If p is not between 0 and 1, PGEOM_H(x,p) is null for all x values.
Examples

```
SELECT nza..PGEOM_H(0, 0.4), nza..PGEOM_H(1, 0.4),
nza..PGEOM_H(2, 0.4), nza..PGEOM_H(3, 0.4);
```

```
PGEOM_H | PGEOM_H | PGEOM_H | PGEOM_H
---------+---------+---------+---------
0.6      |   0.36  |   0.216 |  0.1296
```

(1 row)

```
SELECT nza..PGEOM_H(6, 0.4), nza..PGEOM(6, 0.4);
```

```
PGEOM_H  |   PGEOM  
-----------+-----------
0.0279936 | 0.9720064
```

(1 row)

Related Functions

- category Analytics - Probability Distributions
- DGEOM
- PGEOM

**PHYPER - Cumulative Hypergeometric Distribution**

Given the number of white and black balls in the urn and the number of trials, this function returns the probability that a variable following the Geometric distribution takes a value smaller or equal to x. The value x is the number of white balls drawn from an urn in a series of trials without replacement.

**Usage**

The PHYPER function has the following syntax:

- **PHYPER(BIGINT x, BIGINT wu, BIGINT bu, BIGINT N)**

  ▶ Parameters

  - **x**
    the number of white balls drawn without replacement from an urn which contains both black and white balls
    Type: BIGINT

  - **wu**
    the number of white balls in the urn
Type: BIGINT
Min: 0

► bu
  the number of black balls in the urn
Type: BIGINT
Min: 0

► N
  the number of balls drawn from the urn without replacement
Type: BIGINT
Min: 1

Returns
DOUBLE the cumulative probability distribution from minus infinity to point x

Details
In an urn containing wu white balls and bu black balls, we draw N balls without replacement. Drawing a white ball is a success (1), a black ball is a failure (0). The cumulative Hypergeometric distribution calculates the probability that we get x or less successes in the N trials.

If the number of balls in the urn increases to "infinity" and the quotient bu/wu remains fixed, the Hypergeometric distribution can be approximated with the binomial distribution having p=wu/(wu+bu).

If wu or bu is negative, PHYPER(x,wu,bu,N) is null for all x and N values.

If N is 0 or less, PHYPER(x,wu,bu,N) is null for all x, wu and bu values.

Examples

```
SELECT nza..PHYPER(1,1,1,1);

  PHYPER
    1
(1 row)

SELECT nza..PHYPER(1,2,3,4);

  PHYPER
    0.4
(1 row)
```
SELECT nza..PHYPER(10,20,30,40);

PHYPER
---------------------
1.7985883651357e-05
(1 row)

SELECT nza..PHYPER(10,40,20+30-40,20);

PHYPER
---------------------
1.7985883651357e-05
(1 row)

SELECT nza..PHYPER(0,3,11,4), nza..PHYPER(1.0,3,11,4), nza..PHYPER(2.0,3,11,4), nza..PHYPER(3,3,11,4), nza..PHYPER(4,3,11,4);

<table>
<thead>
<tr>
<th>PHYPER</th>
<th>PHYPER</th>
<th>PHYPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.32967032967033</td>
<td>0.82417582417582</td>
<td>0.98901098901099</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
(1 row)

Related Functions

► category Analytics - Probability Distributions
► DHYPER
► PHYPER_H
► QHYPER

PHYPER_H - Cumulative Hypergeometric Distribution, high tail

Given the number of white and black balls in the urn and the number of trials, this function returns the probability that a variable following the Geometric distribution takes a value greater than x. The value x is
the number of white balls drawn from an urn in a series of trials without replacement.

**Usage**

The `PHYPER_H` function has the following syntax:

- **`PHYPER_H(BIGINT x, BIGINT wu, BIGINT bu, BIGINT N)`**

  ▲ Parameters

  - **x**
    - the number of white balls drawn without replacement from an urn which contains both black and white balls
    - Type: `BIGINT`
  
  - **wu**
    - the number of white balls in the urn
    - Type: `BIGINT`
    - Min: 0
  
  - **bu**
    - the number of black balls in the urn
    - Type: `BIGINT`
    - Min: 0
  
  - **N**
    - the number of balls drawn from the urn without replacement
    - Type: `BIGINT`
    - Min: 1

  ▲ Returns

  - `DOUBLE` the cumulative probability distribution from `x` to plus infinity

**Details**

- `PHYPER_H(x,wu,bu,N) = 1-PHYPER(x,wu,bu,N)` for all `x` and positive `wu`, `bu` and `N` values.
- If `wu` or `bu` is negative, `PHYPER_H(x,wu,bu,N)` is null for all `x` and `N` values.
- If `N` is 0 or less, `PHYPER_H(x,wu,bu,N)` is null for all `x`, `wu` and `bu` values.

**Examples**

```sql
SELECT nza..PHYPER_H(1,1,1,1);

PHYPER_H
----------
  0
(1 row)
```
SELECT nza..PHYPER_H(1,2,3,4);

    PHYPER_H
    ---------
         0.6
(1 row)

SELECT nza..PHYPER_H(10,20,30,40);

    PHYPER_H
    ------------------
          0.99998201411635
(1 row)

SELECT nza..PHYPER_H(10,40,20+30-40,20);

    PHYPER_H
    ------------------
          0.99998201411635
(1 row)

SELECT nza..PHYPER_H(0,3,11,4), nza..PHYPER_H(1.0,3,11,4), nza..PHYPER_H(2.0,3,11,4), nza..PHYPER_H(3.0,3,11,4), nza..PHYPER_H(4.0,3,11,4);

    PHYPER_H     |     PHYPER_H     |     PHYPER_H      |
    PHYPER_H     | PHYPER_H
  +---------------------+----------
      0.67032967032967 | 0.17582417582418 | 0.010989010989011 |
0.4408920985006e-16 |        0
(1 row)
PLNORM - Cumulative Galton (or LogNormal) Distribution

Given a logarithmic scale median and shape, this function returns the probability that a variable following the LogNormal distribution takes a value smaller or equal to x.

Usage

The PLNORM function has the following syntax:

- PLNORM(DOUBLE x, DOUBLE medlog, DOUBLE sdlog)

Parameters

- **x**
  - the value at which to compute
  - Type: DOUBLE
  - Min: 0.000000001
- **medlog**
  - the median value on the logarithmic scale
  - Type: DOUBLE
- **sdlog**
  - the shape on the logarithmic scale
  - Type: DOUBLE
  - Min: 0.000000001

Returns

DOUBLE the cumulative probability distribution from minus infinity to point x.

Details

The LogNormal distribution is defined as PLNORM(x,medlog,sdlog)=PNORM((ln(x)-medlog)/sdlog) for all medlog and for all positive x and sdlog values.

The properties of living tissues, like weight or skin area, tend to follow the Galton distribution.

If x or sdlog is 0 or less, PLNORM(x,medlog,sdlog) is null for all medlog values.

Examples

```sql
SELECT nza..PLNORM(2.43,0,1);
```

```
PLNORM
```
Related Functions

- category Analytics - Probability Distributions
- DLNORM
- PLNORM_H
- PNORM
- QLNORM

PLNORM_H - Cumulative Galton (or LogNormal) Distribution, high tail

Given a logarithmic scale median and shape, this function returns the probability that a variable following the LogNormal distribution takes a value greater than x.

Usage

The PLNORM_H function has the following syntax:

```
PLNORM_H(DOUBLE x, DOUBLE medlog, DOUBLE sdlog)
```

- **x**
  - the value at which to compute
  - Type: DOUBLE
  - Min: 0.000000001

- **medlog**
  - the median value on the logarithmic scale
  - Type: DOUBLE

- **sdlog**
  - the shape on the logarithmic scale
IBM Netezza In-Database Analytics Reference Guide

Type: DOUBLE
Min: 0.000000001

▲ Returns
DOUBLE the cumulative probability distribution from x to plus infinity

Details
PLNORM_H(x,medlog,sdlog)=1-PLNORM(x,medlog,sdlog) for all medlog and for all positive x and sdlog values.
If x or sdlog is 0 or less, PLNORM_H(x,medlog,sdlog) is null for all medlog values.

Examples
SELECT nza..PLNORM_H(2.43,0,1);

```
PLNORM_H
------------------
 0.18729962396611
(1 row)
```

SELECT nza..PLNORM_H(2.43,2,3);

```
PLNORM_H
------------------
 0.64457059023842
(1 row)
```

Related Functions
► category Analytics - Probability Distributions
► DLNORM
► PLNORM

PLOGIS - Cumulative Logistic Distribution
Given the mean and the scale, this function returns the probability that a variable following the Logistic distribution takes a value smaller or equal to x
Usage
The PLOGIS function has the following syntax:

- PLOGIS(DOUBLE x, DOUBLE mean, DOUBLE scale)
  ▲ Parameters
  - x
    the value at which to compute
    Type: DOUBLE
  - mean
    the mean value of the distribution
    Type: DOUBLE
  - scale
    the scale of the distribution
    Type: DOUBLE
    Min: 0.000000001
  ▲ Returns
    DOUBLE the cumulative probability distribution from minus infinity to point x

Details
The logistic distribution is used in describing the spread of epidemics.
PLOGIS(x,mean,scale)=1./(1+exp(-(x-mean)/scale)) for all x and mean values and for all positive scale values.
If scale is 0 or less, PLOGIS(x,mean,scale) is null for all x and mean values.

Examples

```sql
SELECT nza..PLOGIS(2.43,0,1);
PLOGIS
------------------
0.91908653278453
(1 row)

SELECT nza..PLOGIS(2.43,2,3);
PLOGIS
------------------
0.53577211111408
(1 row)
```
Related Functions

- category Analytics - Probability Distributions
- DLOGIS
- PLOGIS_H
- QLOGIS

PLOGIS_H - Cumulative Logistic Distribution, high tail

Given the mean and the scale, this function returns the probability that a variable following the Logistic distribution takes a value greater than x.

Usage

The PLOGIS_H function has the following syntax:

```
PLOGIS_H(DOUBLE x, DOUBLE mean, DOUBLE scale)
```

Parameters

- **x**
  - the value at which to compute
  - Type: DOUBLE
- **mean**
  - the mean value of the distribution
  - Type: DOUBLE
- **scale**
  - the scale of the distribution
  - Type: DOUBLE
  - Min: 0.000000001

Returns

DOUBLE the cumulative probability distribution from x to plus infinity.

Details

PLOGIS_H(x,mean,scale)=1-PLOGIS(x,mean,scale) for all x and mean and for all positive scale values.

If scale is 0 or less, PLOGIS_H(x,mean,scale) is null for all x and mean values.

Examples

```
SELECT nza..PLOGIS_H(2.43,0,1);
```

`PLOGIS_H`
Related Functions

- category Analytics - Probability Distributions
- DLOGIS
- PLOGIS

PMML_DECTREE - Convert a Decision Tree model to PMML

This stored procedure converts a Decision Tree model to the PMML format.

Usage

The PMML_DECTREE stored procedure has the following syntax:

- `PMML_DECTREE(NVARCHAR(ANY) paramString)`
  - Parameters
    - `paramString` comma-separated list of `<parameter>=<value>` entries with parameters below
      - *Type:* NVARCHAR(ANY)
    - `model` the name of the Decision Tree model to convert
      - *Type:* NVARCHAR(ANY)
    - `outtable` the output table where the Decision Tree model is to be stored in PMML format. If the table does not exist it will be created. The PMML model is inserted into this table in one or several rows.
      - the output table should have following columns: seqid, modelname, model. The column modelname contains the name of the model, the PMML model is stored in one or several rows in the
column model, the rows are sorted by the sequence ID in column seqid.

Type: NVARCHAR(ANY)

▲ Returns
INTEGRAR the number of rows written into the output table

Details
This stored procedure converts a Decision Tree model to the PMML format and stores the converted model into the given output table.

Examples

```
CALL nza..DECTREE('model=adult_tree,
    intable=nza..adult_train, id=id, target=income,
    minsplit=1000, eval=entropy, valtable=nza..adult_prune,
    qmeasure=wAcc');

CALL nza..PMML_DECTREE('model=adult_tree,
    outtable=mining_models');

CALL nza..DROP_MODEL('model=adult_tree');

CALL nza..DROP_TABLE('mining_models');

DECTREE
---------

   13
(1 row)

PMML_DECTREE
--------------

    1
(1 row)

DROP_MODEL
------------

    t
(1 row)

DROP_TABLE
-----------
Related Functions

- category Analytics - Classification
- DECTREE
- GROW_DECTREE
- PRUNE_DECTREE
- PRINT_DECTREE
- PMML_MODEL

PMML_MODEL - Generate a PMML representation of an analytics model

This stored procedure stores the given analytics model as PMML document to a table.

Usage

The PMML_MODEL stored procedure has the following syntax:

```
PMML_MODEL(NVARCHAR(ANY))
```

**Parameters**

- **paramString**
  
  comma-separated list of `<parameter>=<value>` entries with parameters below
  
  Type: NVARCHAR(ANY)

- **model**
  
  the model to be stored as PMML document
  
  Type: NVARCHAR(64)

- **type**
  
  the type of PMML document to be created: "standard" (default), or "spss"
  
  Type: VARCHAR(16)

- **outtable**
  
  the output table to write the PMML document into
  
  Type: NVARCHAR(ANY)

**Returns**

INTEGER the number of records written into the PMML table

Details

This stored procedure export the given analytics model as PMML document to a table. Since there is no LOB data type in the Netezza database, the PMML document is written into several rows of the output table referenced by a sequence id. If the output table does not exist, it will be created with the 3 columns below. If the output table exists, it must have two or three columns:
- modelname containing the name of the model (optional, the type is NVARCHAR(128) or smaller),
- seqid containing the model sequence id (mandatory, the type is one of INTEGER/SMALLINT/BIGINT),
- pmml containing the PMML document (mandatory, the type is NVARCHAR(1024) or greater).
If the table exists and has two columns, it must be empty. If it has three columns, the model name must not yet exist (so when you create two PMML documents of different type (standard,spss) for the same model, they cannot be stored in the same table).
If no PMML can be built for the model type, an exception is raised.

Examples

CALL nza..ARULE('intable=nza..retail, model=mbamodel, supporttype=percent, support=5, lvl=0, maxsetsize=5, confidence=0.5');

CALL nza..PMML_MODEL('model=mbamodel, outtable=pmmlmodels');

CALL nza..DROP_MODEL('model=mbamodel');

CALL nza..DROP_TABLE('pmmlmodels');

NOTICE:

RUNNING FPGrowth algorithm:
DATASET : "NZA".."RETAIL"
Transaction column : "TID"
Item column : "ITEM"
Group by : <none>
Minimum support : 5 %
Minimum confidence: 0.5
Max frequent itemset size : 5
Level of conditional dbs : 0
Result tables prefix : "NZA_META_MBAMODEL"

ARULE
-------

14
(1 row)

PMML_MODEL
-----------
PMML_NAIVEBAYES - Convert a Naive Bayes model to PMML

This stored procedure converts a Naive Bayes model to the PMML format

Usage

The PMML_NAIVEBAYES stored procedure has the following syntax:

```sql
PMML_NAIVEBAYES(NVARCHAR(ANY) paramString)
```

Parameters

- **paramString**
  - comma-separated list of <parameter>=<value> entries with parameters below
  - Type: NVARCHAR(ANY)

- **model**
  - the name of the Naive Bayes model to convert
  - Type: NVARCHAR(ANY)

- **outtable**
  - the output table where the Naive Bayes model is to be stored in PMML format. If the table does not exist it will be created. The PMML model is inserted into this table in one or several rows.
  - the output table should have following columns: seqid, modelname, model. The column modelname contains the name of the model, the PMML model is stored in one or several rows in the column model, the rows are sorted by the sequence ID in column seqid.
Type: NVARCHAR(ANY)

▲ Returns
INTEGER the number of rows written into the output table

Details
This stored procedure converts a Naive Bayes model to the PMML format and stores the converted model into the given output table.

Examples
CALL nza..NAIVEBAYES('model=NB_soybean,
intable=nza..soybean_train, id=instance, target=class');
CALL nza..PMML_NAIVEBAYES('model=NB_soybean,
outtable=MINING_MODELS');
CALL nza..DROP_MODEL('model=NB_soybean');
CALL nza..DROP_TABLE('mining_models');

---

<table>
<thead>
<tr>
<th>NAIVEBAYES</th>
</tr>
</thead>
<tbody>
<tr>
<td>1881</td>
</tr>
<tr>
<td>(1 row)</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>PMML_NAIVEBAYES</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
</tr>
<tr>
<td>(1 row)</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>DROP_MODEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
</tr>
<tr>
<td>(1 row)</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>DROP_TABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
</tr>
<tr>
<td>(1 row)</td>
</tr>
</tbody>
</table>
Related Functions

- category Analytics - Classification
- NAIVEBAYES
- PREDICT_NAIVEBAYES
- PMML_MODEL

PMWW - Cumulative Mann-Whitney-Wilcoxon Distribution

Given the number of items separated in two sets, the minimum number of items in set 1, this function returns the probability that a variable following the MWW distribution takes a value smaller or equal to MWWmin.

Usage

The PMWW function has the following syntax:

- **PMWW** (BIGINT MWWmin, BIGINT COUNTmin, BIGINT NoItems)

▲ Parameters

- **MWWmin**
  - the value at which to compute
  - Type: BIGINT
  - Min: 0
  - Max: COUNTmin*(NoItems+NoItems-COUNTmin+1)/2;

- **COUNTmin**
  - the number of items in set 1
  - Type: BIGINT
  - Min: 1
  - Max: NoItems-1

- **NoItems**
  - the total number of items
  - Type: BIGINT
  - Min: 1

▲ Returns

DOUBLE the cumulative probability distribution from minus infinity to point MWWmin

Details

We have a set of observations of variable x split into two sets. We assume that the values of the variable x are distributed in both sets in the same way, and are independent. We assume that x is ordinal, so that we can rank all values of x. The statistics uStat is calculated as the sum of ranks of objects belonging to the first
set or to the second set, whichever is lower.

The function PMWW returns the probability that uStat is smaller or equal to MWWmin. If MWWmin, COUNTmin or NoItems are not in their validity interval, PMWW(MWWmin, Countmin, NoItems) returns null.

Examples

```
SELECT nza..PMWW(13,4,9);
```

```
PMWW
-------------------
0.055555555555556
(1 row)
```

```
SELECT nza..PMWW(30,4,9);
```

```
PMWW
------
1
(1 row)
```

Related Functions

► category Analytics - Probability Distributions
► DMWW
► MWW_TEST
► PMWW_H
► QMWW

PMWW_H - Cumulative Mann-Whitney-Wilcoxon Distribution, high tail

Given the number of items separated in two sets, the minimum number of items in set 1, this function returns the probability that a variable following the MWW distribution takes a value greater than MWWmin

Usage

The PMWW_H function has the following syntax:
**PMWW_H**<br>
BIGINT `MWWmin`, BIGINT `COUNTmin`, BIGINT `NoItems`<br>

▲ Parameters

- **MWWmin**
  - the value at which to compute
  - Type: BIGINT
  - Min: 0
  - Max: `COUNTmin`*(`NoItems`+`NoItems`-`COUNTmin`+1)/2;

- **COUNTmin**
  - the number of items in set 1
  - Type: BIGINT
  - Min: 1
  - Max: `NoItems`-1

- **NoItems**
  - the total number of items
  - Type: BIGINT
  - Min: 1

▲ Returns

- DOUBLE the cumulative probability distribution from `MWWmin` to plus infinity

**Details**

PMWW_H(`MWWmin`,`COUNTmin`,`NoItems`)=1-PMWW(`MWWmin`,`COUNTmin`,`NoItems`) for all `MWWmin`, `COUNTmin` and `NoItems` values in their respective validity interval.

If `MWWmin`, `COUNTmin` or `NoItems` are not in their validity interval, PMWW_H(`MWWmin`,`COUNTmin`,`NoItems`) returns null.

**Examples**

```
SELECT nza..PMWW_H(13,4,9);
```

```
PMWW_H
------------------
0.94444444444444
(1 row)
```

```
SELECT nza..PMWW_H(30,4,9);
```

```
PMWW_H
--------
0
(1 row)
```
Related Functions
► category Analytics - Probability Distributions
► DMWW
► PMWW

PNBINOM - Cumulative Negative Binomial Distribution

Given the success probability \( p \) and the number of successes needed, this function returns the probability that a variable following the Negative Binomial distribution takes a value smaller or equal to \( x \). The value \( x \) is the number of failures in a series of Bernoulli trials before the number of successes requested is reached.

Usage

The PNBINOM function has the following syntax:

► PNBINOM\( (\text{DOUBLE } x, \text{INT8 } s, \text{DOUBLE } p) \)

▲ Parameters
► \( x \)
the value at which to compute
Type: DOUBLE

► \( s \)
the number of successes needed to stop the series of Bernoulli trials
Type: INT8
Min: 1

► \( p \)
the success probability
Type: DOUBLE
Min: 0.00000000001
Max: 1-0.00000000001

▲ Returns
DOUBLE the cumulative probability distribution from minus infinity to point \( x \)

Details

A Bernoulli trial models tossing once a manipulated coin that in general does not have the same chance of heads (or success, 1) and tails (or failure, 0). The parameter \( p \) indicates the probability of getting heads (1). The cumulative Negative Binomial distribution returns the probability that in a series of Bernoulli trials you get \( x \) or less failures before reaching \( s \) successes.
Note that the difference with a Binomial distribution lies in the difference in the stopping condition of the Bernoulli trial serie. While the Negative Binomial distribution limits the number of successes, the Binomial distribution limits the overall number of trials. Hence the domain of the Binomial distribution is finite, while that of the Negative Binomial distribution is infinite.

If \( s \) is 0 or less, \( \text{PNBINOM}(x,s,p) \) is null for all \( x \) and \( p \) values.

If \( p \) is not between 0 and 1, \( \text{PNBINOM}(x,s,p) \) is null for all \( x \) and \( s \) values.

**Examples**

```sql
SELECT nza..PNBINOM(0,3,0.4), nza..PNBINOM(1,3,0.4), nza..PNBINOM(2,3,0.4), nza..PNBINOM(3,3,0.4), nza..PNBINOM(4,3,0.4);
```

<table>
<thead>
<tr>
<th>PNBINOM</th>
<th>PNBINOM</th>
<th>PNBINOM</th>
<th>PNBINOM</th>
<th>PNBINOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.064</td>
<td>0.1792</td>
<td>0.31744</td>
<td>0.45568</td>
<td>0.580096</td>
</tr>
</tbody>
</table>

(1 row)

```sql
SELECT nza..PNBINOM(3,3,0.4), nza..PNBINOM(30,30,0.4), nza..PNBINOM(300,300,0.4), nza..PNBINOM(3000,3000,0.4), nza..PNBINOM(30000,30000,0.4);
```

<table>
<thead>
<tr>
<th>PNBINOM</th>
<th>PNBINOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.45568</td>
<td>0.074623694097046</td>
</tr>
<tr>
<td>2.005553283824e-55</td>
<td>0.549861735637e-07</td>
</tr>
</tbody>
</table>

(1 row)

**Related Functions**

- category Analytics - Probability Distributions
- DNBINOM
- PBINOM
- PNBINOM_H
- QNBINOM

**PNBINOM_H - Negative Binomial Distribution, high tail**

Given the success probability \( p \) and the number of successes needed, this function returns the probability
that a variable following the Negative Binomial distribution takes a value greater than $x$. The value $x$ is the number of failures in a series of Bernoulli trials before the number of successes requested is reached.

**Usage**

The `PNBINOM_H` function has the following syntax:

```plaintext
PNBINOM_H(DOUBLE x, INT8 s, DOUBLE p)
```

**Parameters**

- $x$
  - the value at which to compute
  - Type: DOUBLE

- $s$
  - the number of successes needed to stop the series of Bernoulli trials
  - Type: INT8
  - Min: 1

- $p$
  - the success probability
  - Type: DOUBLE
  - Min: 0.00000000001
  - Max: 1.00000000001

**Returns**

- DOUBLE the cumulative probability distribution from $x$ to plus infinity

**Details**

The function `PNBINOM_H` returns the probability that in a series of $N$ Bernoulli trials you will have more than $x$ failures while getting $s$ successes.

$\text{PNBINOM}_H(x,s,p) = 1 - \text{PNBINOM}(x,s,p)$ for all $x$ and all positive $s$ values and for $p$ between 0 and 1.

If $s$ is 0 or less, $\text{PNBINOM}_H(x,s,p)$ is null for all $x$ and $p$ values.

If $p$ is not between 0 and 1, $\text{PNBINOM}_H(x,s,p)$ is null for all $x$ and $s$ values.

**Examples**

```sql
SELECT nza..PNBINOM_H(0,3,0.4), nza..PNBINOM_H(1.0,3,0.4), nza..PNBINOM_H(2.0,3,0.4), nza..PNBINOM_H(3,3,0.4), nza..PNBINOM_H(4,3,0.4);
```

```
+----------------+----------------+----------------+----------------+----------------+----------------+
| PNBINOM_H       | PNBINOM_H       | PNBINOM_H       | PNBINOM_H       | PNBINOM_H       | PNBINOM_H       |
|-----------------+-----------------+-----------------+-----------------+-----------------+-----------------|
| 0.00000000001   | 0.00000000001   | 0.00000000001   | 0.00000000001   | 0.00000000001   | 0.00000000001   |
|-----------------+-----------------+-----------------+-----------------+-----------------+-----------------|
|-----------------+-----------------+-----------------+-----------------+-----------------+-----------------|
|                 |                 |                 |                 |                 |                 |
|                 |                 |                 |                 |                 |                 |
|                 |                 |                 |                 |                 |                 |
|                 |                 |                 |                 |                 |                 |
|                 |                 |                 |                 |                 |                 |
|                 |                 |                 |                 |                 |                 |
|                 |                 |                 |                 |                 |                 |
|                 |                 |                 |                 |                 |                 |
|                 |                 |                 |                 |                 |                 |
|                 |                 |                 |                 |                 |                 |
|                 |                 |                 |                 |                 |                 |
|                 |                 |                 |                 |                 |                 |
```

---
### Reference Documentation: Analytics

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.936</td>
<td>0.8208</td>
<td>0.68256</td>
<td>0.54432</td>
</tr>
<tr>
<td>(1 row)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
SELECT nza..PNBINOM_H(3,3,0.4), nza..PNBINOM_H(30,30,0.4),
nza..PNBINOM_H(300,300,0.4), nza..PNBINOM_H(3000,3000,0.4),
nza..PNBINOM_H(30000,30000,0.4);
```

<table>
<thead>
<tr>
<th>PNBINOM_H</th>
<th>PNBINOM_H</th>
<th>PNBINOM_H</th>
<th>PNBINOM_H</th>
</tr>
</thead>
<tbody>
<tr>
<td>-----------</td>
<td>---------------</td>
<td>---------------</td>
<td>---------------</td>
</tr>
<tr>
<td>0.54432</td>
<td>0.92537630564857</td>
<td>0.99999954501382</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1 row)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Related Functions

- category Analytics - Probability Distributions
- DNBINOM
- PBINOM_H
- PNBINOM

### PNORM - Cumulative Standard Normal Distribution

This function returns the probability that a variable following the Standard Normal distribution takes a value smaller or equal to \( x \)

#### Usage

The PNORM function has the following syntax:

```
PNORM(DOUBLE x)
```

- **Parameters**
  - \( x \)
    - the value at which to compute

- **Returns**
  - DOUBLE the cumulative probability distribution from minus infinity to point \( x \)
Details

The Gaussian distribution, called also Normal distribution, is claimed to occur in many natural processes. In particular, a multitude of independent processes generating continuous numbers of whatever distribution tend to follow in average a normal distribution.

The Standard Normal Distribution has a mean equal to 0 and a variance equal to 1. If a variable \( X \) follows any normal distribution with mean \( \mu \) and standard deviation \( \sigma \), then \( \frac{(X-\mu)}{\sigma} \) follows the Standard Normal Distribution.

\[ \text{PNORM}(x) = \text{PNORM3P}(x, 0, 1) \] for all \( x \) values.

Examples

```sql
SELECT nza..PNORM(2.43);

PNORM
------------------
0.99245058858369
(1 row)
```

Related Functions

- category Analytics - Probability Distributions
- DNORM
- PNORM3P
- PNORM_H
- QNORM

PNORM3P - Cumulative Normal Distribution

Given a mean value and a standard deviation, this function returns the probability that a variable following the Normal distribution takes a value smaller or equal to \( x \)

Usage

The PNORM3P function has the following syntax:

```sql
PNORM3P(DOUBLE x, DOUBLE mean, DOUBLE stdev)
```

- Parameters
  - \( x \)
    - the value at which to compute
    - Type: DOUBLE
  - mean
the mean value of the distribution
Type: DOUBLE

- **stdev**
  the standard deviation of the distribution
Type: DOUBLE
  Min: 0.00000000001

▲ Returns
DOUBLE the cumulative probability distribution from minus infinity to point x

**Details**
The Gaussian distribution, called also Normal distribution, is claimed to occur in many natural processes. In particular, a multitude of independent processes generating continuous numbers of whatever distribution tend to follow in average a normal distribution.

The Normal Distribution with a mean equal to 0 and a variance equal to 1 is called Standard Normal Distribution. PNOTM3P(x,0,1)=PNOTM(x) for all x values.

If stdev is 0 or less, PNOTM3P(x,mean,stdev) is null for all x and mean values.

**Examples**
```
SELECT nza..PNORM3P(2.43,0,1);

PNORM3P
------------------
0.99245058858369
(1 row)

SELECT nza..PNORM3P(2.43+6,6,1);

PNORM3P
------------------
0.99245058858369
(1 row)

SELECT nza..PNORM3P(2.43*3+6,6,3);

PNORM3P
------------------
0.99245058858369
(1 row)
```
SELECT nza..PNORM3P(2.43*3-6,6,3);

PNORM3P
-------------------
0.058207555638553
(1 row)

SELECT nza..PNORM3P(18-2.43*3,6,3);

PNORM3P
-----------------
0.94179244436145
(1 row)

Related Functions
► category Analytics - Probability Distributions
► DNORM3P
► PNORM
► QNORM3P

PNORM_H - Cumulative Standard Normal Distribution, high tail

This function returns the probability that a variable following the Standard Normal distribution takes a value greater than to x

Usage
The PNORM_H function has the following syntax:

► PNORM_H(DOUBLE x)
  ▲ Parameters
    ► x
      the value at which to compute
      Type: DOUBLE
Returns
DOUBLE the cumulative probability distribution from point x to plus infinity

Details
PNORM_H(x)=1-PNORM(x) for all x values.

Examples
SELECT nza..PNORM_H(2.43);

<table>
<thead>
<tr>
<th>PNORM_H</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0075494114163092</td>
</tr>
</tbody>
</table>

(1 row)

Related Functions
► category Analytics - Probability Distributions
► DNORM
► PNORM

PPOINT - Percentage Point Distributions
This stored procedure computes the value x of a variable following a given distribution for which the probability of being smaller or equal to x is equal to the given percentage

Usage
The PPOINT stored procedure has the following syntax:

► PPOINT(NVARCHAR(ANY) paramString)

Parameters
► paramString
comma-separated list of <parameter>=<value> entries with parameters below
Type: NVARCHAR(ANY)

► intable
the input table
Type: NVARCHAR(256)

► type
the kind of distribution(s) to use:
n - normal (QNORM)
c - chi-square (QCHISQ)
t - t-student (QT)
u - uniform (QUNIF)
e - exponential (QEXP)
w - Weibull (QWEIBULL)

Note that several distributions can be calculated simultaneously by concatenating the distribution abbreviations in the parameter value.

Type: NVARCHAR(ANY)

► incolumn
the numeric input table column containing the percentage. The percentage must be between 0 an 1.
Type: NVARCHAR(128)

► id
the input table column identifying a unique instance id
Type: NVARCHAR(128)

► outtable
the output table
Type: NVARCHAR(256)

► df
the number of degrees of freedom for a student or chi-square distribution
Type: INT8
Default: <none>

► min
the minimum for a uniform distribution
Type: DOUBLE
Default: <none>

► max
the maximum for a uniform distribution
Type: DOUBLE
Default: <none>

► scale
the scale for an exponential or weibull distribution
Type: DOUBLE
Default: <none>
Min: 0.000000000001

► shape
the shape for a weibull distribution
Type: DOUBLE
Default: <none>

► **mean**
the mean value for a normal distribution
Type: DOUBLE
Default: <none>

► **variance**
the variance for a normal distribution
Type: NVARCHAR(ANY)
Default: <none>
Min: 0.00000000001

▲ Returns
INT8 the number of input table values for which a percentage point is calculated

**Details**

Formally the percentage point is the value for which a cumulative distribution function has a probability equal to the given percentage. The cumulative distribution function must be invertable. Hence, the percentage point may be undefined if there exist several x values for which the cumulative distribution function has the same value.

The parameters for the distributions can contain literal values or expressions.

The output table has following columns: id, <incolumn>. Additionally, the output table contains a column for each distribution, named like the distribution density function, and containing the output of this function. It also contains a column per specific parameter for this distribution, named as the parameter, and containing the value calculated for it in each row.

**Examples**

```sql
CALL nza..CUMULATIVE('intable=nza..iris, id=id, type="n", incolumn=petallength, outtable=iripetall, mean=3.75, variance=3.11');

CALL nza..PPOINT('intable=iripetall, id=id, type="n", incolumn=pnorm, outtable=iripetallpp, mean=mean, variance=variance');

SELECT AVG(a.petallength-b.qnorm) FROM nza..iris a INNER JOIN iripetallpp b ON a.id=b.id;

CALL nza..DROP_TABLE('iripetall');

CALL nza..DROP_TABLE('iripetallpp');
```

**CUMULATIVE**

```
-------------
| 150         |
-------------
(1 row)
```
PPOINT
-------
  150
(1 row)

AVG
---------------
-2.2204460492503e-17
(1 row)

DROP_TABLE
-----------
t
(1 row)

DROP_TABLE
-----------
t
(1 row)

Related Functions
► category Analytics - Probability Distributions
► QCHISQ
► QEXP
► QNORM
► QT
► QUNIF
► QWEIBULL

PPOIS - Cumulative Poisson Distribution
Given the mean number of success within a fixed time interval, this function returns the probabil-
ity that a variable following the Poisson distribution takes a value smaller or equal to x. The value x is the number of successes expected within this time interval.

Usage
The PPOIS function has the following syntax:

- **PPOIS**(*x*, *lambda*)

  ▶ **Parameters**
  - **x**
    the value at which to compute
    Type: DOUBLE
  - **lambda**
    the mean number of successes within a fixed time interval
    Type: DOUBLE
    Min: 0.000000000001

- **Returns**
  DOUBLE the cumulative probability distribution from minus infinity to point x

Details
In a Poisson process, the average rate of success over time is known. The probability of a single success within a time interval is proportional to its length and is independent of the probability outside of the interval. If the interval is getting shorter, the probability of getting more than 1 success goes down to zero. The random variable of a Poisson distribution takes on non-negative integer values, indicating the number of successes in the corresponding time interval.

Poisson distribution is also thought sometimes as an extension of the binomial distribution. PBINOM(*x*,*N*,*p*) is approximately PPOIS(*x*,*lambda*) if *p* *N* = *lambda* and *N* grows to infinity.

If *x* is 0 or less, PPOIS(*x*,*lambda*) = 0 for all positive *lambda* values.

If *lambda* is 0 or less, PPOIS(*x*,*lambda*) is null for all *x* values.

Examples

```sql
SELECT nza..PPOIS(0.0, 0.4), nza..PPOIS(1.0, 0.4), nza..PPOIS(2.0, 0.4), nza..PPOIS(3.0, 0.4);
```

<table>
<thead>
<tr>
<th>PPOIS</th>
<th>PPOIS</th>
<th>PPOIS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.67032004603564</td>
<td>0.9384480644499</td>
<td>0.99207366813275</td>
</tr>
<tr>
<td>0.99922374862379</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(1 row)


```sql
SELECT nza..PPOIS(6, 0.4);

PPOIS
------------------
0.99999977069309
(1 row)
```

Related Functions

- category Analytics - Probability Distributions
- DPOIS
- PBINOM
- PPOIS_H
- QPOIS

**PPOIS_H - Cumulative Poisson Distribution, high tail**

Given the mean number of success within a fixed time interval, this function returns the probability that a variable following the Poisson distribution takes a value greater than x. The value x is the number of successes expected within this time interval.

**Usage**

The PPOIS_H function has the following syntax:

```sql
PPOIS_H(DOUBLE x, DOUBLE lambda)
```

▲ Parameters

- **x**
  the value at which to compute
  Type: DOUBLE

- **lambda**
  the mean number of successes within a fixed time interval
  Type: DOUBLE
  Min: 0.00000000001

▲ Returns

DOUBLE the cumulative probability distribution from x to plus infinity

**Details**

\[ PPOIS_H(x,\lambda) = 1 - PPOIS(x,\lambda) \] for all x and all positive lambda values.
If x is 0 or less, PPOIS(x,\lambda) = 0 for all positive lambda values.
If lambda is 0 or less, PPOIS(x, lambda) is null for all x values.

Examples

```sql
SELECT nza..PPOIS_H(0, 0.4), nza..PPOIS_H(1, 0.4), nza..PPOIS_H(2, 0.4), nza..PPOIS_H(3, 0.4);
```

<table>
<thead>
<tr>
<th>PPOIS_H</th>
<th>PPOIS_H</th>
<th>PPOIS_H</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.32967995396334</td>
<td>0.061551935550118</td>
<td>0.0079263318672562</td>
</tr>
<tr>
<td>0.0007762513762073</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(1 row)

```sql
SELECT nza..PPOIS_H(6, 0.4);
```

<table>
<thead>
<tr>
<th>PPOIS_H</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2930691206771e-07</td>
</tr>
</tbody>
</table>

(1 row)

Related Functions

► category Analytics - Probability Distributions
► DPOIS
► PPOIS

PPV - Positive Predictive Value from a Confusion Matrix

This stored procedure calculates the Positive Predictive Value of a class from a confusion matrix.

Usage

The PPV stored procedure has the following syntax:

► PPV(NVARCHAR(ANY) paramString)

▲ Parameters

► paramString

comma-separated list of <parameter>=<value> entries with parameters below

Type: NVARCHAR(ANY)
This stored procedure calculates the Positive Predictive Value of a class from a confusion matrix. The Positive Predictive Value is the ratio of the number or correctly classified records into the given class divided by the total number of predictions into this class.

The confusion matrix table has the following columns: real, prediction, cnt. The column cnt contains the frequency of making a given prediction for the given real value.

**Examples**

```sql
CALL nza..TRAIN_TEST('modelType=dectree, model=iris_c45, traintable=nza..iris_train, testtable=nza..iris_test, target=class, id=id, outtable=iris_pred, mnsplît=2');
CALL nza..CONFUSION_MATRIX('intable=nza..iris_test, resulttable=iris_pred, id=id, resultid=id, target=class, resulttarget=class, matrixTable=confMatrix');
CALL nza..PPV('matrixTable=confMatrix, class=virginica');
CALL nza..DROP_MODEL('model=iris_c45');
CALL nza..DROP_TABLE('iris_pred');
CALL nza..DROP_TABLE('confMatrix');
```

**TRAIN_TEST**

```
0.918919
```

(1 row)

**CONFUSION_MATRIX**

```
5
```

(1 row)
PPV
-----
0.9
(1 row)

DROP_MODEL
-----------
t
(1 row)

DROP_TABLE
----------
t
(1 row)

DROP_TABLE
----------
t
(1 row)

Related Functions
► category Analytics - Diagnostic Measures
► CONFUSION_MATRIX

**PREDICT_ARULE - Predict an Association Rules model**

This stored procedure applies an Association Rules model to new data. All rules or only some of the rules corresponding to given criteria can be applied.

**Usage**

The PREDICT_ARULE stored procedure has the following syntax:

► PREDICT_ARULE(NVARCHAR(ANY) paramString)
Parameters

- **paramString**
  A comma-separated list of `<parameter>=<value>` entries using the parameters below.
  Type: NVARCHAR(ANY)

- **intable**
  the input table
  Type: NVARCHAR(ANY)

- **outtable**
  the output table
  Type: NVARCHAR(ANY)

- **model**
  The name of the Association Rules model to score.
  Type: NVARCHAR(ANY)

- **tid**
  the input table column identifying transactions
  Type: NVARCHAR(ANY)
  Default: tid

- **item**
  the input table column identifying items in transactions
  Type: NVARCHAR(ANY)
  Default: item

- **by**
  the input table column identifying groups of transactions if any. Association Rules mining is done separately on each of these groups. Leave the parameter undefined if no groups are to be considered.
  Type: NVARCHAR(ANY)
  Default: <none>

- **type**
  The type how the scoring algorithm should be applied to the input data. The following values are allowed: recommend, exclusiveRecommend.
  recommend: A rule is returned if its left hand side itemset is a subset of the transaction.
  exclusiveRecommend: A rule is returned if its left hand side itemset is a subset of the input itemset, and its right hand side itemset is not a subset of the transaction.
  Type: VARCHAR(64)
  Default: exclusiveRecommend
- **namemap**
  table which provides names of items and their associated mapped values in LHS_ITEMS, RHS_ITEMS columns of outtable
  Type: NVARCHAR(ANY)
  Default: <null>

- **itemname**
  the column name of namemap table where the item identifiers are stored
  Type: NVARCHAR(128)
  Default: item

- **itemnamemapped**
  the column name of namemap table where the item names are stored which should be used instead of the item identifier.
  Type: NVARCHAR(128)
  Default: item_name

- **minsize**
  The minimum number of items per association rule to be applied.
  Type: INTEGER
  Default: 1
  Min: 1
  Max: 64

- **maxsize**
  The maximum number of items per association rule to be applied.
  Type: INTEGER
  Default: 64
  Min: 1
  Max: 64

- **minsupp**
  The minimum support of an association rule to be applied.
  Type: DOUBLE
  Default: 0.0
  Min: 0.0
  Max: 1.0

- **maxsupp**
  The maximum support of an association rule to be applied.
  Type: DOUBLE
  Default: 1.0
  Min: 0.0
Max: 1.0

► **minconf**
The minimum confidence of an association rule to be applied.
Type: DOUBLE
Default: 0.0
Min: 0.0
Max: 1.0

► **maxconf**
The maximum confidence of an association rule to be applied.
Type: DOUBLE
Default: 1.0
Min: 0.0
Max: 1.0

► **minlift**
The minimum lift of an association rule to be applied.
Type: DOUBLE
Default: 0.0
Min: 0.0

► **maxlift**
The maximum lift of an association rule to be applied.
Type: DOUBLE
Default: null (unlimited)
Min: 0.0

► **minconv**
The minimum conviction of an association rule to be applied.
Type: DOUBLE
Default: 0.0
Min: 0.0

► **maxconv**
The maximum conviction of an association rule to be applied.
Type: DOUBLE
Default: null (unlimited)
Min: 0.0

► **minaffi**
The minimum affinity of an association rule to be applied.
Type: DOUBLE
Default: 0.0
Min: 0.0
Max: 1.0

- **maxaffi**
The maximum affinity of an association rule to be applied.
Type: DOUBLE
Default: 1.0
Min: 0.0
Max: 1.0

- **minleve**
The minimum leverage of an association rule to be applied.
Type: DOUBLE
Default: -0.25
Min: -0.25
Max: 1.0

- **maxleve**
The maximum leverage of an association rule to be applied.
Type: DOUBLE
Default: 1.0
Min: -0.25
Max: 1.0

▲ Returns
BITINT The number of transactions processed.

**Details**
This procedure predicts association rules of an Association Rules model. To select interesting rules from the set of all detected rules, the minimum/maximum constraints on the following measures of significance and interest can be used: support, confidence, lift and conviction.

- The support supp(X) of an itemset X is defined as the proportion of transactions in the dataset which contain all items of the itemset. For a rule, all items of the rule are considered.

- The confidence of a rule X->Y is defined as: conf(X->Y) = supp({X,Y})/supp(X). The confidence can be interpreted as an estimation of the probability P(Y|X), the probability of finding the items of Y in transactions that also contain the items of X.

- The lift of a rule X->Y is defined as: lift(X->Y) = supp({X,Y})/(supp(Y)*supp(X)). The lift is the ratio of the observed support for {X,Y} to the support expected if X and Y were independent.

- The conviction of a rule X->Y is defined as: conv(X->Y) = (1 - supp(Y))/(1 - conf(X->Y)). The conviction can be interpreted as can be interpreted as the ratio of the expected frequency that X occurs without Y, which is the frequency that the rule makes an incorrect prediction, if X and Y were independent to the observed fre-
Examples

CALL nza..ARULE('intable=nza..retail, model=assoc, supporttype=percent, support=5, lvl=0, maxsetsize=5, confidence=0.5');

CALL nza..PREDICT_ARULE('model=assoc, intable=nza..retail, outtable=retail_score, minsupp=0.08, minconf=0.6, minlift=1.33, minconv=1.3');

select * from retail_score;
drop table retail_score;
CALL nza..DROP_MODEL('model=assoc');

<table>
<thead>
<tr>
<th>GID</th>
<th>TID</th>
<th>LHS_SID</th>
<th>RHS_SID</th>
<th>LHS_ITEMS</th>
<th>RHS_ITEMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>12</td>
<td>2</td>
<td>39,41</td>
<td></td>
<td>48</td>
</tr>
<tr>
<td>0.083550736144824</td>
<td>0.64534781846855</td>
<td>1.3503062625329</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.4720704655452</td>
<td>0.15949591841154</td>
<td>0.021675339086308</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(1 row)
DROP TABLE

DROP_MODEL

-------------
t

(1 row)

Related Functions

► category Analytics - Association Rules
► ARULE

PREDICT_DECTREE - Apply a Decision Tree model

This stored procedure applies a Decision Tree model to generate classification predictions

Usage

The PREDICT_DECTREE stored procedure has the following syntax:

► PREDICT_DECTREE(NVARCHAR(ANY) paramString)
  ▲ Parameters
  ▶ paramString
    comma-separated list of <parameter>=<value> entries with parameters below
    Type: NVARCHAR(ANY)
  ▶ model
    the Decision Tree model
    Type: NVARCHAR(ANY)
  ▶ intable
    the input table
    Type: NVARCHAR(256)
outtable
the output table where the predictions will be stored
Type: NVARCHAR(256)

id
the input table column identifying a unique instance id
Type: NVARCHAR(128)
Default: <id> column used to build the model

target
the input table column representing the class.
Type: NVARCHAR(128)
Default: <target> column used to build the model

prob
the flag indicating whether the probability of the predicted class should be included into the output table or not
Type: BOOLEAN
Default: false

outtableprob
if specified, the probability output table where class probability predictions will be stored
Type: NVARCHAR(256)
Default: NULL

Returns
INTEGER the number of input table records for which predictions were generated

Details
This stored procedure applies a Decision tree model to predict a class value for records of the input table.
The predictions are stored in an output table with following columns: id, class, prob. The id column matches the <id> column of the input table and the class column contains the predicted class label for the corresponding input table record. The prob column is included only if the prob argument is true, it contains the probability that the predicted value is correct.

Examples
CALL nza..DECTREE('model=adult_tree,
intable=nza..adult_train, id=id, target=income,
minsplit=1000, eval=entropy, valtable=nza..adult_prune,
qmeasure=wAcc');

CALL nza..PREDICT_DECTREE('model=adult_tree,
intable=nza..adult_test, id=id, outtable=adult_pred,
prob=TRUE');
CALL nza..DROP_MODEL('model=adult_tree');
CALL nza..DROP_TABLE('adult_pred');

DECTREE
---------
13
(1 row)

PREDICT_DECTREE
-----------------
8140
(1 row)

DROP_MODEL
------------
t
(1 row)

DROP_TABLE
------------
t
(1 row)

---

Related Functions
► category Analytics - Classification
► DECTREE
► GROW_DECTREE
► PRUNE_DECTREE
► LIST_MODELS

---

PREDICT_DIVCLUSTER - Apply a Hierarchical Clustering model

This stored procedure applies a Hierarchical Clustering model to cluster records of a dataset.

Usage

The PREDICT_DIVCLUSTER stored procedure has the following syntax:

► PREDICT_DIVCLUSTER(NVARCHAR(ANY) paramString)
   ▲ Parameters
paramString
comma-separated list of <parameter>=<value> entries with parameters below
Type: NVARCHAR(ANY)

model
the name of the Hierarchical Clustering model
Type: NVARCHAR(ANY)

intable
the input table
Type: NVARCHAR(256)

outtable
the output table where the assigned clusters will be stored
Type: NVARCHAR(256)

id
the input table column identifying a unique instance id
Type: NVARCHAR(128)
Default: <id> column used to build the model

level
the level of the cluster hierarchy which should be applied to the data. For level=-1, only the leaves of the clustering tree are considered.
Type: INTEGER
Default: -1
Min: -1
Max: 60

Returns
BIGINT the number of input table records for which a cluster is assigned

Details
This stored procedure applies a Hierarchical Clustering model to cluster records of the input table. The clusters assigned to the input table records are stored in an output table with following columns: id, cluster_id, distance. The id column matches the <id> column of the input table. Each input table record is associated with a cluster, where the distance from the record to the cluster center is the smallest. The cluster ID and the distance to the cluster center are given in the columns cluster_id and distance. The negative number of cluster_id identifies leafs of the cluster tree.

Examples
CALL nza..DIVCLUSTER('model=iris_mdl, intable=nza..iris, outtable=iris_out, id=id, target=class, distance=euclidean, maxiter=5, msplit=5, maxdepth=3,
randseed=12345');

CALL nza..DROP_TABLE('iris_out');

CALL nza..PREDICT_DIVCLUSTER('model=iris_mdl, intable=nza..iris, outtable=iris_out, id=id, level=-1');

CALL nza..DROP_MODEL('model=iris_mdl');

CALL nza..DROP_TABLE('iris_out');

DIVCLUSTER
----------
  15
(1 row)

DROP_TABLE
----------
 t
(1 row)

PREDICT_DIVCLUSTER
-------------------
  150
(1 row)

DROP_MODEL
----------
 t
(1 row)

DROP_TABLE
----------
 t
(1 row)

Related Functions
► category Analytics - Clustering
► DIVCLUSTER
► LIST_MODELS
PREDICT_GLM - Apply a Generalized Linear Model

This stored procedure applies a Generalized Linear Model to generate regression predictions

Usage

The PREDICT_GLM stored procedure has the following syntax:

```sql
PREDICT_GLM(NVARCHAR(ANY) paramString)
```

Parameters

- **paramString**
  - comma-separated list of `<parameter>=<value>` entries with parameters below
  - Type: NVARCHAR(ANY)

- **model**
  - the Generalized Linear model
  - Type: NVARCHAR(ANY)

- **intable**
  - the input table
  - Type: NVARCHAR(256)

- **outtable**
  - the output table where the predictions will be stored
  - Type: NVARCHAR(256)

- **id**
  - the input table column identifying a unique instance id
  - Type: NVARCHAR(128)
  - Default: <id> column used to build the model

- **debug**
  - flag indicating to display debug information
  - Type: BOOLEAN
  - Default: false

Returns

- BIGINT the number of rows processed

Details

This stored procedure applies a Generalized Linear model to predict a numeric value for records of the input table.

The predictions are stored in an output table with following columns: id, pred.
Examples

CALL nza..GLM('model=adult_glm, intable=nza..adult_train, id=id, target=age');
CALL nza..PREDICT_GLM('model=adult_glm, intable=nza..adult_test, id=id, outtable=adult_pred');
CALL nza..DROP_MODEL('model=adult_glm');
CALL nza..DROP_TABLE('adult_pred');

GLM
-----
58
(1 row)

PREDICT_GLM
-------------
8140
(1 row)

DROP_MODEL
------------
t
(1 row)

DROP_TABLE
-----------
t
(1 row)

Related Functions
► category Analytics - Regression
► GLM
► LIST_MODELS

PREDICT_KMEANS - K-means Clustering
This stored procedure applies a K-means Clustering model to cluster records of a dataset.

Usage

The PREDICT_KMEANS stored procedure has the following syntax:

```sql
PREDICT_KMEANS(NVARCHAR(ANY) paramString)
```

Parameters

- `paramString` comma-separated list of `<parameter>=<value>` entries with parameters below
  - `model` the name of the K-means Clustering model
    - Type: NVARCHAR(ANY)
  - `intable` the input table
    - Type: NVARCHAR(256)
  - `outtable` the output table where the assigned clusters will be stored
    - Type: NVARCHAR(256)
  - `id` the input table column identifying a unique instance id
    - Type: NVARCHAR(128)
      - Default: `<id>` column used to build the model

Returns

- `BIGINT` the number of input table records for which a cluster is assigned

Details

This stored procedure applies a K-means Clustering model to cluster records of the input table. The clusters assigned to the input table records are stored in an output table with following columns: `id`, `cluster_id`, `distance`. The `id` column matches the `<id>` column of the input table. Each input table record is associated with a cluster, where the distance from the record to the cluster center is the smallest. The cluster ID and the distance to the cluster center are given in the columns `cluster_id` and `distance`.

Examples

```sql
CALL nza..KMEANS('model=adult_mdl, intable=nza..adult, outtable=adult_out, id=id, target=income, transform=L, distance=euclidean, k=3, maxiter=5, randseed=12345');
CALL nza..PREDICT_KMEANS('model=adult_mdl,'
INTABLE=nza..ADULT, OUTTABLE=ADULT_PRED, ID=id');

CALL nza..DROP_MODEL('model=adult_mdl');
CALL nza..DROP_TABLE('adult_out');
CALL nza..DROP_TABLE('adult_pred');

KMEANS
-------

3
(1 row)

PREDICT_KMEANS
---------------

32561
(1 row)

DROP_MODEL
----------

t
(1 row)

DROP_TABLE
----------

t
(1 row)

Related Functions

- category Analytics - Clustering
- KMEANS
- LIST_MODELS
PREDICT_KNN - Apply a K-Nearest Neighbors model

This stored procedure applies a K-Nearest Neighbors model to generate classification or regression predictions for a dataset.

Usage

The PREDICT_KNN stored procedure has the following syntax:

```
PREDICT_KNN(NVARCHAR(ANY) paramString)
```

Parameters

- **paramString**
  - comma-separated list of <parameter>=<value> entries with parameters below
  - Type: NVARCHAR(ANY)
- **model**
  - the name of the Decision Tree model
  - Type: NVARCHAR(ANY)
- **inable**
  - the input table
  - Type: NVARCHAR(256)
- **outtable**
  - the output table where the predictions will be stored
  - Type: NVARCHAR(ANY)
- **id**
  - the input table column identifying a unique instance id
  - Type: NVARCHAR(128)
  - Default: <id> column used to build the model
- **target**
  - the input table column representing the class.
  - Type: NVARCHAR(128)
  - Default: <target> column used to build the model
- **distance**
  - the distance function. Allowed values are: euclidean, manhatthan, canberra, maximum.
  - Type: NVARCHAR(ANY)
  - Default: euclidean
- **k**
  - number of nearest neighbors to consider
  - Type: INTEGER
Default: 3

► **stand**
flag indicating whether the measurements in the input table are standardized before calculating
the distance.
Type: BOOLEAN
Default: true

► **fast**
flag indicating that the algorithm used coresets based method.
Type: BOOLEAN
Default: true

► **weights**
the input table containing optional class weights for the input table <target> column.
The <weights> table is used only when the <target> column is not numeric. If the parameter is
undefined, we assume that the weights are uniformly equal to 1.
The <weights> table contains following columns:
  - weight: a numeric column containing the class weight,
  - class: a column to be joined with the <target> column of <intable>, defining class weights.
For classes not occurring in this table, weights of 1 are assumed.
Type: NVARCHAR(256)
Default: <none>

▲ Returns
BIGINT the number of input table records for which predictions were generated

**Details**
This stored procedure applies a K-Nearest Neighbors model to predict a class value or a numeric value for
records of the input table.
The predictions are stored in an output table with following columns: id, class. The id column matches the
<i>id</i> column of the input table and the class column contains the predicted class label or the predicted value
for the corresponding input table record.

**Examples**

```sql
CALL nza..KNN('model=iris_mdl_c, intable=nza..iris, id=id, target=class');

CALL nza..PREDICT_KNN('model=iris_mdl_c, intable=nza..iris, outtable=iris_cla, id=id, target=class, distance=euclidean, k=3, stand=false, fast=false');

CALL nza..KNN('model=iris_mdl_r, intable=nza..iris, id=id, target=petalwidth');

CALL nza..PREDICT_KNN('model=iris_mdl_r, intable=nza..iris, outtable=iris_reg, id=id, target=petalwidth, distance=euclidean,
```
k=1, stand=false, fast=false');
CALL nza..DROP_MODEL('model=iris_mdl_c');
CALL nza..DROP_MODEL('model=iris_mdl_r');
CALL nza..DROP_TABLE('iris_cla');
CALL nza..DROP_TABLE('iris_reg');

KNN
-----
150
(1 row)

PREDICT_KNN
------------
150
(1 row)

KNN
-----
150
(1 row)

PREDICT_KNN
------------
150
(1 row)

DROP_MODEL
----------
t
(1 row)

DROP_MODEL
----------
t
(1 row)

DROP_TABLE
-------------
t
(1 row)

DROP_TABLE
-------------
t
(1 row)

Related Functions
► category Analytics - Classification
► KNN
► LIST_MODELS

PREDICT_LINEAR_REGRESSION - Apply a Linear Regression Model

This stored procedure applies a Linear Regression model to generate regression predictions for a dataset

Usage

The PREDICT_LINEAR_REGRESSION stored procedure has the following syntax:

► PREDICT_LINEAR_REGRESSION(NVARCHAR(ANY) paramString)
  ▲ Parameters
    ► paramString
      comma-separated list of <parameter>=<value> entries with parameters below
      Type: NVARCHAR(ANY)
    ► model
      the Linear Regression model
      Type: NVARCHAR(ANY)
    ► intable
      the input table
      Type: NVARCHAR(256)
    ► outtable
      the output table where the predictions will be stored
      Type: NVARCHAR(256)
    ► id
the input table column identifying a unique instance id

Type: NVARCHAR(128)
Default: <id> column used to build the model

▲ Returns
BOOLEAN always true

Details
This stored procedure applies a Linear Regression model to predict one or several numeric values for records of the input table.

The predictions are stored in an output table with the <id> column of the input table and all predicted columns of the Linear Regression model containing their predicted value for the corresponding input table record.

Examples
CREATE TABLE adultT1 AS SELECT id, age AS v1, fnlwgt AS fnlwgt, education_num AS v3 FROM nza..adult;

CALL nza..LINEAR_REGRESSION('model=modelA, intable=adultT1, id=id, target=fnlwgt');

CALL nza..PREDICT_LINEAR_REGRESSION('model=modelA, intable=adultT1, outtable=adult_fnlwgt_pred, id=id');

CALL nza..DROP_MODEL('model=modelA');

CALL nza..DROP_TABLE('adult_fnlwgt_pred');

CALL nza..DROP_TABLE('adultT1');

LINEAR_REGRESSION
-------------------
t
(1 row)

PREDICT_LINEAR_REGRESSION
---------------------------
t
(1 row)

DROP_MODEL
-----------
CREATE TABLE adultT2 AS SELECT id, age, education_num, income, sex FROM nza..adult;

CALL nza..LINEAR_REGRESSION('model=modelC, intable=adultT2, id=id, target=age, nominalCols=income;sex, calculateDiagnostics=TRUE');

CALL nza..PREDICT_LINEAR_REGRESSION('model=modelC, intable=adultT2, outtable=adult_age_pred, id=id');

CALL nza..DROP_MODEL('model=modelC');

CALL nza..DROP_TABLE('adult_age_pred');

CALL nza..DROP_TABLE('adultT2');
DROP_MODEL
------------
t
(1 row)

DROP_TABLE
------------
t
(1 row)

DROP_TABLE
------------
t
(1 row)

Related Functions
► category Analytics - Regression
► LINEAR_REGRESSION

PREDICT_NAIVEBAYES - Apply a Naive Bayes model

This stored procedure applies a Naive Bayes model to generate classification predictions for a dataset.

Usage
The PREDICT_NAIVEBAYES stored procedure has the following syntax:

► PREDICT_NAIVEBAYES(NVARCHAR(ANY) paramString)
  ▲ Parameters
    ► paramString
      comma-separated list of <parameter>=<value> entries with parameters below
      Type: NVARCHAR(ANY)
    ► model
      the name of the Naive Bayes model
Reference Documentation: Analytics

Type: NVARCHAR(ANY)

► **intable**
  the input table
  Type: NVARCHAR(256)

► **outtable**
  the output table where the predictions will be stored
  Type: NVARCHAR(ANY)

► **id**
  the input table column identifying a unique instance id
  Type: NVARCHAR(128)
  Default: <id> column used to build the model

► **target**
  the input table column representing the class.
  Type: NVARCHAR(128)
  Default: <target> column used to build the model

► **outtableProb**
  the output table where the probabilities for each of the classes will be stored. If not specified, the table will not be created.
  Type: NVARCHAR(ANY)
  Default: <none>

► **mestimation**
  flag indicating to use m-estimation for probabilities. This kind of estimation of probabilities may be slower but can give better results for small or heavy unbalanced datasets.
  Type: NVARCHAR(ANY)
  Default: false

▲ Returns
  INTEGER the number of input table records for which predictions were generated

**Details**

This stored procedure applies a Naive Bayes model to predict a class value for records of the input table. The predictions are stored in an output table with following columns: id, class. The id column matches the <id> column of the input table and the class column contains the predicted class label for the corresponding input table record.

If the parameter <outtableProb> is specified, probabilities for predicting the class labels are stored in another output table with following columns: id, class, prob, logprob. This table contains probabilities for each record of the input table and for each class label.

**Examples**

```
CALL nza..NAIVEBAYES('model=NB_soybean,
```
```
intable=nza..soybean_train, id=instance, target=class');
CALL nza..PREDICT_NAIVEBAYES('model=NB_soybean,
intable=nza..soybean_test, id=instance,
outtable=soybean_pred');
CALL nza..DROP_MODEL('model=NB_soybean');
CALL nza..DROP_TABLE('soybean_pred');

NAIVEBAYES
------------
1881
(1 row)

PREDICT_NAIVEBAYES
---------------------
170
(1 row)

DROP_MODEL
-----------
t
(1 row)

DROP_TABLE
----------
t
(1 row)

Related Functions
► category Analytics - Classification
► NAIVEBAYES
► LIST_MODELS
```
PREDICT_REGTREE - Apply a Regression Tree model

This stored procedure applies a Regression Tree model to generate regression predictions for a dataset

Usage

The PREDICT_REGTREE stored procedure has the following syntax:

```
PREDICT_REGTREE(NVARCHAR(ANY) paramString)
```

- **paramString**
  - comma-separated list of <parameter>=<value> entries with parameters below
  - Type: NVARCHAR(ANY)

- **model**
  - the Regression Tree model
  - Type: NVARCHAR(ANY)

- **intable**
  - the input table
  - Type: NVARCHAR(256)

- **outtable**
  - the output table where the predictions will be stored
  - Type: NVARCHAR(256)

- **id**
  - the input table column identifying a unique instance id
  - Type: NVARCHAR(128)
  - Default: <id> column used to build the model

- **target**
  - the input table column representing the prediction target. The specified target column will not be used for prediction and is permitted to contain NULL values.
  - Type: NVARCHAR(128)
  - Default: <target> column used to build the model

- **var**
  - a flag indicating whether the variance of the predictions should be included into the output table
  - Type: BOOLEAN
  - Default: false

Returns

INTEGER the number of input table records for which predictions were generated

Details

This stored procedure applies a Regression tree model to predict a numeric value for records of the input
The predictions are stored in an output table with following columns: id, class, var. The id column matches the <id> column of the input table and the class column contains the predicted value for the corresponding input table record. The var column is included only if the var argument is true, it contains the prediction variance.

Examples

CALL nza..REGTREE('model=wrt, intable=nza..weatherr, id=instance, target=grade, minsplt=2, maxdepth=4');

CALL nza..PREDICT_REGTREE('model=wrt, intable=nza..weatherr, id=instance, outtable=wpr, var=TRUE');

CALL nza..DROP_MODEL('model=wrt');

CALL nza..DROP_TABLE('wpr');

```
REGTREE
-------
13
(1 row)

PREDICT_REGTREE
----------------
22
(1 row)

DROP_MODEL
-----------
t
(1 row)

DROP_TABLE
----------
t
(1 row)
```

Related Functions

► category Analytics - Regression
► GROW_REGTREE
► LIST_MODELS
PREDICT_TWOSTEP - Apply a TwoStep Clustering model

This stored procedure applies a TwoStep Clustering model to score records of a dataset.

Usage

The PREDICT_TWOSTEP stored procedure has the following syntax:

```
PREDICT_TWOSTEP(NVARCHAR(ANY) paramString)
```

Parameters

- **paramString**
  - comma-separated list of <parameter>=<value> entries with parameters below
  - Type: NVARCHAR(ANY)

- **model**
  - the name of the TwoStep Clustering model to be used
  - Type: NVARCHAR(ANY)

- **intable**
  - the input table
  - Type: NVARCHAR(256)

- **outtable**
  - the output table where the clusters assigned will be stored
  - Type: NVARCHAR(ANY)

- **id**
  - the input table column identifying a unique instance id
  - Type: NVARCHAR(128)
  - Default: the id column used to build the TwoStep clustering model

Returns

- BIGINT the number of records scored assigned to a cluster

Details

This stored procedure applies a TWOSTEP Clustering model to cluster records of the input table. The clusters assigned to the input table records are stored in an output table with following columns: id, cluster_id, distance. The id column matches the <id> column of the input table. Each input table record is associated with a cluster, where the distance from the record to the cluster is the smallest. The cluster ID and the distance to the cluster are given in the columns cluster_id and distance.

Examples

```
CALL nza..TWOSTEP('model=adult_mdl, intable=nza..adult, distance=euclidean, id=id');
```
CALL nza..PREDICT_TWOSTEP('model=adult_mdl, intable=nza..adult, outtable=adult_pred, id=id');
CALL nza..DROP_MODEL('model=adult_mdl');
CALL nza..DROP_TABLE('adult_pred');

TWOSTEP
--------
  2
(1 row)

PREDICT_TWOSTEP
----------------
    32561
(1 row)

DROP_MODEL
-----------
  t
(1 row)

DROP_TABLE
----------
  t
(1 row)

Related Functions
► category Analytics - Clustering
► TWOSTEP
► LIST_MODELS

PRINT_ARULE - Print an Association Rules model

This stored procedure displays the association rules of an Association Rules model. All rules or only some of the rules corresponding to given criteria can be displayed.
Reference Documentation: Analytics

Usage

The PRINT_ARULE stored procedure has the following syntax:

► PRINT_ARULE(NVARCHAR(ANY) paramString)

▲ Parameters

► paramString
A comma-separated list of <parameter>=<value> entries using the parameters below.
Type: NVARCHAR(ANY)

► model
The name of the Association Rules model to be printed.
Type: NVARCHAR(ANY)

► outtable
A table with Association Rules in a human readable format to be generated. Columns GRP, DRIVERS, DRIVER1, DRIVER2, .. DRIVERN, TARGET, NUM_DRIVERS, NUM_TARGET, SUPPORT_PCT, CONFIDENCE_PCT, LIFT, CONVICTION, ABSOLUTE_SUPPORT (Where DRIVERS are the concatenated ITEM resp. ITEM_DESCRIPTIONS of all DRIVERS) if 'null' no outtable is generated
Type: NVARCHAR(ANY)
Default: null

► namemap
the namemap table identifying items (column item) and there associated names (item_name). If namemap parameter is specified all items in textual output an in outtable, if specified, are replaced by there corresponding values from item names. Values from namemap column item are mapped to values from namemap column item_name. If a value is missing from namemap column item, its original value is displayed. If 'null', no name mapping is used.
Type: NVARCHAR(ANY)
Default: null

► item
the namemap table column identifying items. The type of column item must be of the same kind (numerical/ nominal) as column item of intable specified when the model was created.
Type: NVARCHAR(ANY)
Default: item

► item_name
the namemap table column identifying the item name associated with item
Type: NVARCHAR(ANY)
Default: item_name

► minsize
The minimum number of items per association rule to be displayed.
Type: INTEGER
Default: 1
Min: 1  
Max: 64

- **maxsize**
  The maximum number of items per association rule to be displayed.
  Type: INTEGER
  Default: 64
  Min: 1  
  Max: 64

- **minsupp**
  The minimum support of an association rule to be displayed.
  Type: DOUBLE
  Default: 0.0
  Min: 0.0  
  Max: 1.0

- **maxsupp**
  The maximum support of an association rule to be displayed.
  Type: DOUBLE
  Default: 1.0
  Min: 0.0  
  Max: 1.0

- **minconf**
  The minimum confidence of an association rule to be displayed.
  Type: DOUBLE
  Default: 0.0
  Min: 0.0  
  Max: 1.0

- **maxconf**
  The maximum confidence of an association rule to be displayed.
  Type: DOUBLE
  Default: 1.0
  Min: 0.0  
  Max: 1.0

- **minlift**
  The minimum lift of an association rule to be displayed.
  Type: DOUBLE
Default: 0.0
Min: 0.0

► **maxlift**
The maximum lift of an association rule to be displayed.
Type: DOUBLE
Default: 1e12
Min: 0.0

► **minconv**
The minimum conviction of an association rule to be displayed.
Type: DOUBLE
Default: 0.0
Min: 0.0

► **maxconv**
The maximum conviction of an association rule to be displayed.
Type: DOUBLE
Default: 1e12
Min: 0

▲ Returns
TEXT A formatted output of the association rules.

Details
This procedure displays association rules of an Association Rules model. To select interesting rules from the set of all detected rules, the minimum/maximum constraints on the following measures of significance and interest can be used: support, confidence, lift and conviction.

- The support \( \text{supp}(X) \) of an itemset \( X \) is defined as the proportion of transactions in the dataset which contain all items of the itemset. For a rule, all items of the rule are considered.

- The confidence of a rule \( X \rightarrow Y \) is defined as: \( \text{conf}(X \rightarrow Y) = \text{supp}(\{X,Y\})/\text{supp}(X) \). The confidence can be interpreted as an estimation of the probability \( P(Y|X) \), the probability of finding the items of \( Y \) in transactions that also contain the items of \( X \).

- The lift of a rule \( X \rightarrow Y \) is defined as: \( \text{lift}(X \rightarrow Y) = \text{supp}(\{X,Y\})/(\text{supp}(Y) \times \text{supp}(X)) \). The lift is the ratio of the observed support for \( \{X,Y\} \) to the support expected if \( X \) and \( Y \) were independent.

- The conviction of a rule \( X \rightarrow Y \) is defined as: \( \text{conv}(X \rightarrow Y) = (1 - \text{supp}(Y))/(1 - \text{conf}(X \rightarrow Y)) \). The conviction can be interpreted as can be interpreted as the ratio of the expected frequency that \( X \) occurs without \( Y \), which is the frequency that the rule makes an incorrect prediction, if \( X \) and \( Y \) were independent to the observed frequency of incorrect predictions.

Examples
```sql
CREATE TABLE retailnamemap AS SELECT DISTINCT(item), 'item'|| item::VARCHAR(5) AS item_name from nza..retail;
DELETE FROM retailnamemap WHERE item=39;
```
CALL nza..ARULE('intable=nza..retail, model=assoc, supporttype=percent, support=5, lvl=0, maxsetsize=5, confidence=0.5');

CALL nza..PRINT_ARULE('model=assoc, minsize=3, minsupp=0.05, minconf=0.6, minlift=1.3, minconv=1.3');

CALL nza..PRINT_ARULE('model=assoc, namemap=retailnamemap, item=item, item_name=item_name, outtable=assoc_out, minsize=3, minsupp=0.05, minconf=0.6, minlift=1.3, minconv=1.3');

SELECT * FROM assoc_out ORDER BY support_pct desc, confidence_pct DESC;

CALL nza..DROP_MODEL('model=assoc');

DROP TABLE assoc_out;

DROP TABLE retailnamemap;

INSERT 0 16470

DELETE 1

ARULE

-------

  14

(1 row)

PRINT_ARULE

---------------------------------------------------------------------
---------------------------------------------------------------------

| GRP= | {41,48} -> {39} [supp=0.083550736144824, conf=0.81681082279885, lift=1.4210493489806, conv=2.3211298697154, abs_supp=7366] |
|------|
| GRP= | {39,41} -> {48} [supp=0.083550736144824, conf=0.64534781846855, lift=1.3503062625329, conv=1.47207046555452, abs_supp=7366] |
|------|
| GRP= | {38,48} -> {39} [supp=0.069213493341803, conf=0.7681298688821752, lift=1.3363513116731, conv=1.8337868834446, abs_supp=6102] |
|------|
| GRP= | {32,39} -> {48} [supp=0.061273564574306, conf=0.63891188645772, lift=1.3368399129912, conv=1.4458326995372, abs_supp=5402] |

(1 row)
PRINT_ARULE

---------------------------------------------------------------

GRP | {item41,item48} -> {39}
[ supp=0.083550736144824, conf=0.81681082279885,
  lift=1.4210493489806, conv=2.3211298697154, abs_supp=7366]

GRP | {39,item41} -> {item48}
[ supp=0.083550736144824, conf=0.64534781846855,
  lift=1.3503062625329, conv=1.4720704655452, abs_supp=7366]

GRP | {item38,item48} -> {39}
[ supp=0.069213493341803, conf=0.76812688821752,
  lift=1.3363513116731, conv=1.8337868834446, abs_supp=6102]

GRP | {39,item32} -> {item48}
[ supp=0.061273564574306, conf=0.63891188645772,
  lift=1.3368399129912, conv=1.4458326995372, abs_supp=5402]

(1 row)

| item41,item48 | item41 | item48 | 39 | 2 |
| 1 | 8.3551 | 81.6811 | 1.42 | 2.32 |
| 7366 |
| 39,item41 | 39 | item41 | item48 | 2 |
| 1 | 8.3551 | 64.5348 | 1.35 | 1.47 |
| 7366 |
| item38,item48 | item38 | item48 | 39 | 2 |
| 1 | 6.9213 | 76.8127 | 1.34 | 1.83 |
| 6102 |
| item32,39 | item32 | 39 | item48 | 2 |
| 1 | 6.1274 | 63.8912 | 1.34 | 1.45 |
| 5402 |

(4 rows)

DROP_MODEL

------------------
t

(1 row)
Related Functions

- category Analytics - Association Rules
- ARULE
- PRINT_MODEL

PRINT_DECTREE - Print a Decision Tree model

This stored procedure displays the tree of a Decision Tree model.

Usage

The PRINT_DECTREE stored procedure has the following syntax:

```
PRINT_DECTREE(NVARCHAR(ANY) paramString)
```

Parameters

- `paramString`
  comma-separated list of `<parameter>=<value>` entries with parameters below
  Type: NVARCHAR(ANY)

- `model`
  the name of the Decision Tree model to print
  Type: NVARCHAR(ANY)

Returns

TEXT pretty print of the decision tree

Details

This stored procedure displays a tree representation of a Decision Tree. The size of the presented tree is limited to the 12th depth level.

Examples

```
CALL nza..DECTREE('model=adult_tree,
intable=nza..adult_train, id=id, target=income,
minsplit=1000, eval=entropy, valtable=nza..adult_prune,
qmeasure=wAcc');

CALL nza..PRINT_DECTREE('model=adult_tree');

CALL nza..DROP_MODEL('model=adult_tree');
```
DECTREE
--------
13
(1 row)

PRINT_DECTREE
---------------------------------------------------------------
---------------------------------------------------------------
---------------------------------------------------------------
---------------------------------------------------------------
---------------------------------------------------------------
---------------------------------------------------------------
---------------------------------------------------------------
---------------------------------------------------------------
-------
-- decision tree model: "ADULT_TREE" --
MARITAL_STATUS = Married-civ-spouse
| EDUCATION_NUM <= 12
| | CAPITAL_GAIN <= 5013
| | | EDUCATION_NUM <= 8
| | | | if true then class -> small
| | | | CAPITAL_LOSS <= 1844
| | | | | if true then class -> small
| | | | | if false then class -> large
| | | | if false then class -> large
| | | if false then class -> large
| | if false then class -> large
| if false then class -> large
| CAPITAL_GAIN <= 6849
| | if true then class -> small
| | if false then class -> large

(1 row)

DROP_MODEL
-----------
t
(1 row)
Related Functions
► category Analytics - Classification
► DECTREE
► GROW_DECTREE
► PRUNE_DECTREE
► PRINT_MODEL

PRINT_GLM - Print a Generalized Linear Model

This stored procedure displays the Generalized Linear Model

Usage
The PRINT_GLM stored procedure has the following syntax:

► PRINT_GLM(NVARCHAR(ANY) paramString)
  ▲ Parameters
  ► paramString
    comma-separated list of <parameter>=<value> entries with parameters below
    Type: NVARCHAR(ANY)
  ► model
    the name of the Generalized Linear Model to print
    Type: NVARCHAR(ANY)
  ▲ Returns
    TEXT pretty print of the GLM model

Details
This stored procedure displays the variables (column or column=value) used in the Generalized Linear model.

Examples
CALL nza..GLM('model=adult_glm, intable=nza..adult_train, id=id, target=age');
CALL nza..PRINT_GLM('model=adult_glm');
CALL nza..DROP_MODEL('model=adult_glm');

GLM

-----
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Beta</th>
<th>Std Error</th>
<th>Test</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAPITAL_GAIN</td>
<td>1236476372.4326</td>
<td>1.261565</td>
<td>980113039.86098</td>
<td>0</td>
</tr>
<tr>
<td>CAPITAL_LOSS</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>[EDUCATION=10th]</td>
<td>70694.25256</td>
<td>9.9e-05</td>
<td>712294983.15725</td>
<td>0</td>
</tr>
<tr>
<td>[EDUCATION=11th]</td>
<td>9457026.152379</td>
<td>0.011683</td>
<td>809445741.10391</td>
<td>0</td>
</tr>
<tr>
<td>[EDUCATION=12th]</td>
<td>1054976.097704</td>
<td>0.00131</td>
<td>805531163.51935</td>
<td>0</td>
</tr>
<tr>
<td>[EDUCATION=1st-4th]</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>[EDUCATION=5th-6th]</td>
<td>5953.979105</td>
<td>7e-06</td>
<td>806110584.50755</td>
<td>0</td>
</tr>
<tr>
<td>[EDUCATION=7th-8th]</td>
<td>1699415.276319</td>
<td>0.002126</td>
<td>799475249.68433</td>
<td>0</td>
</tr>
<tr>
<td>[EDUCATION=9th]</td>
<td>517064.283118</td>
<td>0.000669</td>
<td>773444626.15156</td>
<td>0</td>
</tr>
<tr>
<td>[EDUCATION=Assoc-acdm]</td>
<td>-6487425.351825</td>
<td>0.007926</td>
<td>-818542637.29585</td>
<td>0</td>
</tr>
<tr>
<td>[EDUCATION=Assoc-voc]</td>
<td>4297865.495047</td>
<td>0.005344</td>
<td>804207685.01062</td>
<td>0</td>
</tr>
<tr>
<td>[EDUCATION=Bachelors]</td>
<td>2790986.17114</td>
<td>0.003377</td>
<td>826572317.51956</td>
<td>0</td>
</tr>
<tr>
<td>[EDUCATION=Doctorate]</td>
<td>10081.224693</td>
<td>1.2e-05</td>
<td>812254638.51987</td>
<td>0</td>
</tr>
<tr>
<td>[EDUCATION=HS-grad]</td>
<td>23996067.744298</td>
<td>0.029649</td>
<td>809341980.55466</td>
<td>0</td>
</tr>
<tr>
<td>[EDUCATION=Masters]</td>
<td>-178807.154514</td>
<td>0.000209</td>
<td>-855890549.04825</td>
<td>0</td>
</tr>
<tr>
<td>[EDUCATION=Preschool]</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>[EDUCATION=Prof-school]</td>
<td>-66497403.583758</td>
<td>0.082528</td>
<td>-805752687.99542</td>
<td>0</td>
</tr>
<tr>
<td>[EDUCATION=Some-college]</td>
<td>-4792983.442041</td>
<td>0.005801</td>
<td>-826291353.47292</td>
<td>0</td>
</tr>
<tr>
<td>EDUCATION_NUM</td>
<td>-741594040.77619</td>
<td>0.919503</td>
<td>-806516120.1987</td>
<td>0</td>
</tr>
<tr>
<td>FNLWGT</td>
<td>7048682.96056</td>
<td>0.008357</td>
<td>843448003.01712</td>
<td>0</td>
</tr>
<tr>
<td>HOURS_PER_WEEK</td>
<td>-2925425212.5927</td>
<td>3.599251</td>
<td>-81278155.18794</td>
<td>0</td>
</tr>
<tr>
<td>[INCOME=large]</td>
<td>-78590200.483261</td>
<td>0.097219</td>
<td>-808382324.63756</td>
<td>0</td>
</tr>
<tr>
<td>[MARITAL_STATUS=Divorced]</td>
<td>10916381.920467</td>
<td>0.013598</td>
<td>80281369.22133</td>
<td>0</td>
</tr>
<tr>
<td>[MARITAL_STATUS=Married-AF-spouse]</td>
<td>32516.293256</td>
<td>4e-05</td>
<td>807324374.62716</td>
<td>0</td>
</tr>
<tr>
<td>[MARITAL_STATUS=Married-civ-spouse]</td>
<td>-90699696.605852</td>
<td>0.112281</td>
<td>-807793726.07924</td>
<td>0</td>
</tr>
<tr>
<td>[MARITAL_STATUS=Married-spouse-absent]</td>
<td>3092290.106745</td>
<td>0.003832</td>
<td>806968757.90802</td>
<td>0</td>
</tr>
<tr>
<td>[MARITAL_STATUS=Never-married]</td>
<td>33610782.85697</td>
<td>0.041477</td>
<td>810351610.16738</td>
<td>0</td>
</tr>
<tr>
<td>[MARITAL_STATUS=Separated]</td>
<td>1413861.75504</td>
<td>0.001778</td>
<td>795934019.08737</td>
<td>0</td>
</tr>
<tr>
<td>[OCCUPATION=Adm-clerical]</td>
<td>6831925.450798</td>
<td>0.008485</td>
<td>805160127.85801</td>
<td>0</td>
</tr>
<tr>
<td>[OCCUPATION=Armed-Forces]</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>[OCCUPATION=Craft-repair]</td>
<td>7800946.988712</td>
<td>0.009685</td>
<td>805454423.57801</td>
<td>0</td>
</tr>
<tr>
<td>[OCCUPATION=Exec-managerial]</td>
<td>7292348.517329</td>
<td>0.009091</td>
<td>802136233.81722</td>
<td>0</td>
</tr>
<tr>
<td>[OCCUPATION=Farming-fishing]</td>
<td>52271.588155</td>
<td>0.002129</td>
<td>406690834.27643</td>
<td>0</td>
</tr>
<tr>
<td>[OCCUPATION=Handlers-cleaners]</td>
<td>4848799.041996</td>
<td>0.006019</td>
<td>805454423.57801</td>
<td>0</td>
</tr>
<tr>
<td>[OCCUPATION=Machine-op-inspct]</td>
<td>-28622648.118112</td>
<td>0.035411</td>
<td>-808303537.86755</td>
<td>0</td>
</tr>
</tbody>
</table>
| [OCCUPATION=Other-service] | 22945284.976713 | 0.
<table>
<thead>
<tr>
<th></th>
<th>0.02844</th>
<th>806807222.15172</th>
<th>0</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>[OCCUPATION=Priv-house-serv]</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>[OCCUPATION=Prof-specialty]</td>
<td>-56259524.59247</td>
<td>0.06996</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[OCCUPATION=Protective-serv]</td>
<td>-7135928.83446</td>
<td>0.008702</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[OCCUPATION=Sales]</td>
<td>3866031.731685</td>
<td>0.004608</td>
<td>838909066.86626</td>
<td>0</td>
</tr>
<tr>
<td>[OCCUPATION=Tech-support]</td>
<td>4926746.289357</td>
<td>0.006112</td>
<td>806062900.07193</td>
<td>0</td>
</tr>
<tr>
<td>[RACE=Amer-Indian-Eskimo]</td>
<td>-13090239.556102</td>
<td>0.016326</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[RACE=Asian-Pac-Islander]</td>
<td>-57148707.07129</td>
<td>0.070897</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[RACE=Black]</td>
<td>7073138.392589</td>
<td>0.008752</td>
<td>808156798.35235</td>
<td>0</td>
</tr>
<tr>
<td>[RACE=Other]</td>
<td>56757.170433</td>
<td>7e-05</td>
<td>809453070.20306</td>
<td>0</td>
</tr>
<tr>
<td>[RELATIONSHIP=Husband]</td>
<td>-90025569.639478</td>
<td>0.111572</td>
<td>-806885688.31952</td>
<td>0</td>
</tr>
<tr>
<td>[RELATIONSHIP=Not-in-family]</td>
<td>21585833.265741</td>
<td>0.026898</td>
<td>802503255.78684</td>
<td>0</td>
</tr>
<tr>
<td>[RELATIONSHIP=Other-relative]</td>
<td>8426225.551671</td>
<td>0.010369</td>
<td>812598532.63938</td>
<td>0</td>
</tr>
<tr>
<td>[RELATIONSHIP=Own-child]</td>
<td>13638284.064545</td>
<td>0.016768</td>
<td>813331234.32789</td>
<td>0</td>
</tr>
<tr>
<td>[RELATIONSHIP=Unmarried]</td>
<td>12400492.327892</td>
<td>0.015344</td>
<td>808171903.33442</td>
<td>0</td>
</tr>
<tr>
<td>[SEX=Female]</td>
<td>47443608.699712</td>
<td>0.058642</td>
<td>809037545.9424</td>
<td>0</td>
</tr>
<tr>
<td>[WORKCLASS=Federal-gov]</td>
<td>2275316.264219</td>
<td>0.002843</td>
<td>800447560.49006</td>
<td>0</td>
</tr>
<tr>
<td>[WORKCLASS=Local-gov]</td>
<td>-73500307.465652</td>
<td>0.091017</td>
<td>807541076.89325</td>
<td>0</td>
</tr>
<tr>
<td>[WORKCLASS=Never-worked]</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>[WORKCLASS=Private]</td>
<td>22128409.07369</td>
<td>0.027512</td>
<td>804319208.69186</td>
<td>0</td>
</tr>
<tr>
<td>[WORKCLASS=Self-emp-inc]</td>
<td>954332.337204</td>
<td>0.001083</td>
<td>881332186.93148</td>
<td>0</td>
</tr>
</tbody>
</table>
Residuals Summary:
<table>
<thead>
<tr>
<th>Residual Type</th>
<th>RSS</th>
<th>df</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pearson</td>
<td>0</td>
<td>16222</td>
<td>0</td>
</tr>
<tr>
<td>Deviance</td>
<td>0</td>
<td>16222</td>
<td>0</td>
</tr>
</tbody>
</table>

(1 row)

DROP_MODEL

-------------
t

(1 row)

Related Functions
► category Analytics - Regression
► GLM
► PRINT_MODEL

PRINT_KMEANS - Print a K-means model
Print the clusters, the centers or the column statistics of a K-means model.

Usage
The PRINT_KMEANS stored procedure has the following syntax:

► PRINT_KMEANS(NVARCHAR(ANY) paramString)
  ▲ Parameters
  ► paramString
  comma-separated list of <parameter>=<value> entries with parameters below
  Type: NVARCHAR(ANY)
Reference Documentation: Analytics

► model
  the K-means model to print
  Type: NVARCHAR(ANY)

► clusters
  the list of cluster IDs of the model to print. The cluster IDs are separated by a semicolon (;). If not specified, information for all clusters are printed.
  Type: NVARCHAR(ANY)

► columns
  the list of input column names of the model to print. The input column names are separated by a semicolon (;). If not specified, information for all input columns are printed.
  Type: NVARCHAR(ANY)

► mode
  the display mode determining which information of the model to print. Allowed values are: clusters, centers, statistics.
  Type: NVARCHAR(ANY)
  Default: clusters

▲ Returns
TEXT pretty print of the K-means model

Details
The information about the K-means model is printed out like a table for all clusters and all input columns that are indicated.
- If mode=clusters, a table with one row per cluster is returned with following columns: clusterid, name, description, size, withinss.
- If mode=centers, a table with one row per input column per cluster is returned with following columns: clusterid, columnname, cardinality, mode, minimum, maximum, mean, variance, count, importance. If the model was built with statistics=none, only columns clusterid, columnname, mode (for nominal columns) and mean (for numeric columns contain values.
- If mode=statistics, a table with one row per value of the input columns and per cluster is returned with following columns: clusterid, columnname, value, count, relfrequency, deviation, mean and variance. The table is empty if the model was built with statistics=none or statistics=columns.

The rows printed out are sorted by clusterid, columnname and value (as applicable). The number of rows printed out is limited to about 1 Million characters. If more rows have to be printed out, an exception is raised.

A generic stored procedure PRINT_MODEL is available for all types of models that have a print stored procedure.

Examples

call nza..kmeans('model=iris_km, intable=nza..iris, distance=euclidean, id=id, k=5, maxiter=3, outtable=km_out, statistics=values:25');

CALL nza..PRINT_KMEANS('model=iris_km, mode=centers,
clusters=1;2, columns=class;petallength');
CALL nza..DROP_MODEL('model=iris_km');
CALL nza..DROP_TABLE('km_out');

KMEANS

--------

5

(1 row)

PRINT_KMEANS

<table>
<thead>
<tr>
<th>CLUSTERID</th>
<th>COLUMNNAME</th>
<th>CARDINALITY</th>
<th>MODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MINIMUM</td>
<td>MAXIMUM</td>
<td>MEAN</td>
<td>VARIANCE</td>
</tr>
<tr>
<td>COUNT</td>
<td>IMPORTANCE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+-----------+-------------+-------------+------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>CLASS</td>
<td>2</td>
<td>virginica</td>
</tr>
<tr>
<td></td>
<td></td>
<td>41</td>
<td>1.6868633187861</td>
</tr>
<tr>
<td>1</td>
<td>PETALLENGTH</td>
<td>17</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>5.1</td>
<td>4.1604651162791</td>
<td>0.30906097560976</td>
</tr>
<tr>
<td>2</td>
<td>CLASS</td>
<td>2</td>
<td>versicolor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>37</td>
<td></td>
</tr>
</tbody>
</table>
1.0593264025352 |
| 2         | PETALLENGTH | 14          |            | 4.3 |
| 5.6     | 5.2209302325581 | 0.16519519519519 | 37    | 1.2790199297733 |
+-----------------+
| (1 row) |

DROP_MODEL
----------
t
(1 row)

DROP_TABLE
----------
t
(1 row)

Related Functions
► category Analytics - Clustering
► KMEANS
► PRINT_MODEL

PRINT_MANOVA_ONE WAY TEST - Pretty print of Multivariate Analysis of Variance result in one way setting
This stored procedure prints the result of one way MANOVA output table

Usage
The PRINT_MANOVA_ONE WAY TEST stored procedure has the following syntax:

► PRINT_MANOVA_ONE WAY TEST(NVARCHAR(ANY) paramString)
▲ Parameters
► paramString
  comma-separated list of <parameter>=<value> entries with parameters below
select * from wheattest2trcv;
CREATE TABLE wheattest2trcv(
  id_task INTEGER,
  row INTEGER
);
, col INTEGER
, val DOUBLE);

-- INSERT INTO wheattest2 VALUES(1, 'A', 80, 4);
INSERT INTO wheattest2trcv VALUES(0,9,1, 0);
INSERT INTO wheattest2trcv VALUES(0,9,2, 80);
INSERT INTO wheattest2trcv VALUES(0,9,3, 4);

-- INSERT INTO wheattest2 VALUES(2, 'A', 65, 3);
INSERT INTO wheattest2trcv VALUES(0,1,1, 0);
INSERT INTO wheattest2trcv VALUES(0,1,2, 65);
INSERT INTO wheattest2trcv VALUES(0,1,3, 3);

-- INSERT INTO wheattest2 VALUES(3, 'A', 50, 2);
INSERT INTO wheattest2trcv VALUES(0,2,1, 0);
INSERT INTO wheattest2trcv VALUES(0,2,2, 50);
INSERT INTO wheattest2trcv VALUES(0,2,3, 2);

-- INSERT INTO wheattest2 VALUES(4, 'B', 100, 5);
INSERT INTO wheattest2trcv VALUES(0,3,1, 1);
INSERT INTO wheattest2trcv VALUES(0,3,2, 100);
INSERT INTO wheattest2trcv VALUES(0,3,3, 5);

-- INSERT INTO wheattest2 VALUES(5, 'B', 85, 4);
INSERT INTO wheattest2trcv VALUES(0,4,1, 1);
INSERT INTO wheattest2trcv VALUES(0,4,2, 85);
INSERT INTO wheattest2trcv VALUES(0,4,3, 4);

-- INSERT INTO wheattest2 VALUES(6, 'B', 70, 3);
INSERT INTO wheattest2trcv VALUES(0,5,1, 1);
INSERT INTO wheattest2trcv VALUES(0,5,2, 70);
INSERT INTO wheattest2trcv VALUES(0,5,3, 3);

-- INSERT INTO wheattest2 VALUES(7, 'C', 60, 2);
INSERT INTO wheattest2trcv VALUES(0,6,1, 2);
INSERT INTO wheattest2trcv VALUES(0,6,2, 60);
INSERT INTO wheattest2trcv VALUES(0,6,3, 2);

-- INSERT INTO wheattest2 VALUES(8, 'C', 75, 3);
INSERT INTO wheattest2trcv VALUES(0,7,1, 2);
INSERT INTO wheattest2trcv VALUES(0,7,2, 75);
INSERT INTO wheattest2trcv VALUES(0, 7, 3, 3);
-- INSERT INTO wheattest2 VALUES(9, 'C', 90, 5);
INSERT INTO wheattest2trcv VALUES(0, 8, 1, 2);
INSERT INTO wheattest2trcv VALUES(0, 8, 2, 90);
INSERT INTO wheattest2trcv VALUES(0, 8, 3, 5);
CALL nza..MANOVA_ONE_WAY_TEST('intable=wheattest2trcv, outtable=outtab');
CALL nza..PRINT_MANOVA_ONE_WAY_TEST('intable=outtab');
CALL nza..DROP_TABLE('outtab');
CALL nza..DROP_TABLE('wheattest2trcv');

MANOVA statistics and significance for table : "OUTTAB"
task: 0 one way MANOVA

<table>
<thead>
<tr>
<th>statistics</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wilk's Lambda</td>
<td>0.39512195121951</td>
</tr>
<tr>
<td>0.28038050472772</td>
<td></td>
</tr>
<tr>
<td>Roy's Largest Root</td>
<td>1.44736237142</td>
</tr>
<tr>
<td>0.26118761661441</td>
<td></td>
</tr>
<tr>
<td>Hotelling's Trace</td>
<td>1.4814814814815</td>
</tr>
<tr>
<td>0.26289561423837</td>
<td></td>
</tr>
<tr>
<td>Pillais' Trace</td>
<td>0.62439024390244</td>
</tr>
<tr>
<td>0.30419561076293</td>
<td></td>
</tr>
</tbody>
</table>

degrees of freedom: 2

CREATE TABLE wheatbytest(fieldId INTEGER, variety CHAR(1), yield DOUBLE, barrels INTEGER, experiment INT4);

INSERT INTO wheatbytest VALUES(1, 'A', 65, 3, 1);
-- suspect: INSERT INTO wheatbytest
VALUES(2, 'A', 66, 4, 1);
INSERT INTO wheatbytest VALUES(2, 'A', 65, 3, 1);
INSERT INTO wheatbytest VALUES(3, 'A', 64, 2, 1);
INSERT INTO wheatbytest VALUES(4, 'B', 84, 3, 1);
INSERT INTO wheatbytest VALUES(5, 'B', 85, 4, 1);
INSERT INTO wheatbytest VALUES(6, 'B', 86, 5, 1);
INSERT INTO wheatbytest VALUES(7, 'C', 75, 3, 1);
INSERT INTO  wheatbytest VALUES(8,'C',76,4,1);  
INSERT INTO  wheatbytest VALUES(9,'C',74,2,1);  
INSERT INTO  wheatbytest VALUES(11,'A',80,4,2);  
INSERT INTO  wheatbytest VALUES(12,'A',65,3,2);  
INSERT INTO  wheatbytest VALUES(13,'A',50,2,2);  
INSERT INTO  wheatbytest VALUES(14,'B',100,5,2);  
INSERT INTO  wheatbytest VALUES(15,'B',85,4,2);  
INSERT INTO  wheatbytest VALUES(16,'B',70,3,2);  
INSERT INTO  wheatbytest VALUES(17,'C',60,2,2);  
INSERT INTO  wheatbytest VALUES(18,'C',75,3,2);  
INSERT INTO  wheatbytest VALUES(19,'C',90,5,2);  
CALL nza..MANOVA_ONE_WAY_TEST('intable=wheatbytest, id=fieldid, 
incolumn=yield;barrels, factor1=variety, by=experiment, 
outtable=outbytab,type=columns');  
CALL nza..PRINT_MANOVA_ONE_WAY_TEST('intable=outbytab');  
CALL nza..DROP_TABLE('outbytab');  
CALL nza..DROP_TABLE('wheatbytest');  
MANOVA statistics and significance for table : "OUTBYTAB"  
task: 1 - 1 one way MANOVA  
| statistics         | p-value       |  
| Bonferroni correction  |  |  
Wilk`s Lambda | 1.5486982034876e-18 | 0  | 0  
Roy`s Largest Root | 6.102377495087e+17 | 0  | 0  
Hotelling`s Trace | 6.102377495087e+17 | 0  | 0  
Pillais` Trace | 1.0549258936356 | 0.046366790504921 |
| 0.090583701748115  
degrees of freedom: 2  
task: 2 - 2 one way MANOVA  
| statistics         | p-value       |  
| Bonferroni correction  |  |  
Wilk`s Lambda | 0.39512195121951 | 0.28038050472772 |
| 0.48214778202407  
Roy`s Largest Root | 1.44736237142 | 0.26118761661441 |
| 0.4541562621561  
Hotelling`s Trace | 1.4814814814815 | 0.26289561423837 |
| 0.45667712449096  

Pillais’ Trace | 0.62439024390244 | 
0.30419561076293 | 0.51585625191842

degrees of freedom: 2

Related Functions
► category Analytics - Statistics
► MANOVA_ONE WAY_TEST

PRINT_MANOVA_Two WAY_TEST - Pretty print of Multivariate Analysis of Variance result in two way setting

This stored procedure prints the result of Two way MANOVA output table

Usage
The PRINT_MANOVA_Two WAY_TEST stored procedure has the following syntax:

► PRINT_MANOVA_Two WAY_TEST(NVARCHAR(ANY) paramString)

▲ Parameters
► paramString
comma-separated list of <parameter>=<value> entries with parameters below
Type: NVARCHAR(ANY)
► intable
the input table name
Type: NVARCHAR(ANY)
► maxtasks
the maximum number of tasks that will be presented
Type: integer
Default: 20
► chooseindicator
- not required one of the values df PillaisTrace HotellingsTrace RoysLargestRoot WilksLambda PillaisTracePval HotellingsTracePval RoysLargestRootPval WilksLambdaPval
Type: NVARCHAR(ANY)
Default: <none>
► choosecomparator
> or < - not required , but required if previous parameter is there
Type: NVARCHAR(ANY)
Default: <none>
choosevalue
- not required, but required if previous parameter is there
  Type: DOUBLE
  Default: <none>

^ Returns
  NVARCHAR(2000) A string containing the pretty print of the tasks

Details
This stored procedure outputs in a pretty form the results of Two-way analysis of variance/covariance aiming to tell whether or not the groups of data identified by factor1 have the same mean value in all dependent variables or not. Note that the Two way analysis could have been performed for a multitude of tasks. In that case for all of the tasks output is deliverd unless limitation is encountered in terms of maxtasks parameter. Only up to maxtasks tasks will be displayed. You may also specify a condition for displaying tasks. For this purpose specify chooseindicator, choosecomparator, choosevalue

Related Functions
► category Analytics - Statistics
► PRINT_MANOVA_ONE_WAY_TEST
► MANOVA_Two_WAY_TEST

PRINT_MODEL - Print an analytics model
This stored procedure displays the content of a given analytics model

Usage
The PRINT_MODEL stored procedure has the following syntax:

^ PRINT_MODEL(NVARCHAR(ANY))
  ▲ Parameters
  ▶ paramString
    comma-separated list of <parameter>=<value> entries with parameters below
    Type: NVARCHAR(ANY)
  ▶ model
    the model to be displayed
    Type: NVARCHAR(64)
  ▲ Returns
    TEXT The printout of the model

Details
This stored procedure displays the content of a given analytics model. The call is forwarded to a model-type specific print procedure of the form PRINT_<procedure-name>, where <procedure-name> is the name of the model building procedure. If there is no print procedure for the model type, an exception is raised.
Additional model-type specific parameters can be specified in the parameter string, they are forwarded to
the specific print procedure.

Examples

CALL nza..ARULE('intable=nza..retail, model=mbamodel, supporttype=percent, support=5, lvl=0, maxsetsize=5, confidence=0.5');

CALL nza..PRINT_MODEL('model=mbamodel, minsize=3, minsupp=0.05, minconf=0.6, minlift=1.3, minconv=1.3');

CALL nza..DROP_MODEL('model=mbamodel');

NOTICE:

RUNNING FP-Growth algorithm:
DATASET : "NZA".."RETAIL"
Transaction column : "TID"
Item column : "ITEM"
Group by : <none>
Minimum support : 5 %
Minimum confidence: 0.5
Max frequent itemset size : 5
Level of conditional dbs : 0
Result tables prefix : "NZA_META_MBAMODEL"

---

ARULE

14
(1 row)

---

PRINT_MODEL

---------

---------
GRP=| {32,39} -> {48} [supp=0.061273564574306, conf=0.63891188645772, lift=1.336399129912, conv=1.4458326995372]

GRP=| {38,48} -> {39} [supp=0.069213493341803, conf=0.76812688821752, lift=1.3363513116731, conv=1.8337868834446]

GRP=| {41,48} -> {39} [supp=0.083550736144824, conf=0.81681082279885, lift=1.4210493489806, conv=2.3211298697154]

GRP=| {39,41} -> {48} [supp=0.083550736144824, conf=0.64534781846855, lift=1.3503062625329, conv=1.4720704655452]

(1 row)

NOTICE: Dropped: MBAMODEL
DROP_MODEL
----------
t
(1 row)

Related Functions
► category Analytics - Model Management
► LIST_MODELS

PRINT_REGTREE - Print a Regression Tree model

This stored procedure displays the tree of a Regression Tree model

Usage
The PRINT_REGTREE stored procedure has the following syntax:

► PRINT_REGTREE(NVARCHAR(ANY) paramString)
▲ Parameters
► paramString
  comma-separated list of <parameter>=<value> entries with parameters below
  Type: NVARCHAR(ANY)
► model
the name of the Decision Tree model to print
Type: NVARCHAR(ANY)

▲ Returns
TEXT pretty print of the regression tree

Details
This stored procedure displays a tree representation of a Regression Tree. The size of the presented tree is limited to the 12th depth level.

Examples

CALL nza..REGTREE('model=wrt, intable=nza..weatherr, id=instance, target=grade, minsplit=2, maxdepth=4');
CALL nza..PRINT_REGTREE('model=wrt');
CALL nza..DROP_MODEL('model=wrt');

REGTREE
--------
13
(1 row)

PRINT_REGTREE
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
-----------------------------
--
-- regression tree: "WRT" --
OUTLOOK = sun
 | TEMPERATURE <= 72
 | | TEMPERATURE <= 52
 | | | if true then class value -> 4
 | | | if false then class value -> 5
 | | if false then class value -> 2
 | OUTLOOK = cloudy
TEMPERATURE <= 12
  | if true then class value -> 2
  | if false then class value -> 3
HUMIDITY = low
  | if true then class value -> 2
  | if false then class value -> 1

DROP_MODEL

**Related Functions**

- category Analytics - Regression
- GROW_REGTREE
- PRINT_MODEL
- PRUNE_REGTREE
- REGTREE
- REGTREE
- GROW_REGTREE
- PRUNE_REGTREE
- PREDICT_REGTREE

**PRINT_TIMESERIES - Print Time Series predictions**

Print the history, interpolation and forecast values for the given time series

**Usage**

The PRINT_TIMESERIES stored procedure has the following syntax:

- PRINT_TIMESERIES(NVARCHAR(ANY) paramString)

  **Parameters**

  - paramString
    comma-separated list of <parameter>=<value> entries with parameters below

    Type: NVARCHAR(ANY)
► **model**
the Time Series model to print
Type: NVARCHAR(ANY)

► **series**
the list of IDs for the time series to print. The IDs are separated by a semicolon (;). If not specified, all time series are printed.
Type: NVARCHAR(ANY)

► **history**
a flag indicating whether the history data of the time series must be printed too
Type: BOOLEAN
Default: false

► **interpolation**
a flag indicating whether the interpolated data of the time series must be printed too
Type: BOOLEAN
Default: false

► **plot**
a flag indicating whether the time series must be printed as a plotter graph or as a table
Type: BOOLEAN
Default: false

▲ Returns
TEXT pretty print of the time series

**Details**
The time series forecasts are printed out like a table with following columns: tsid, time, forecast. If parameter history is true, a column history is added. If parameter interpolation is true, a column interpolated is added. The rows printed out are sorted by tsid and time.

The number of rows printed out is limited to about 1 Million characters. If more rows have to be printed out, an exception is raised.

A generic stored procedure PRINT_MODEL is available for all types of models that have a print stored procedure.

**Examples**

```
CALL nza..TIMESERIES('model=curves_ts,
intable=nza..curves, time=x, target=y, by=curve,
algorithm=exponentialsmoothing');

CALL nza..PRINT_TIMESERIES('model=curves_ts,
series="sinus", plot=true');

CALL nza..DROP_MODEL('model=curves_ts');
```
TIMESERIES
--------------

6
(1 row)

PRINT_TIMESERIES

sinus

<table>
<thead>
<tr>
<th>0</th>
<th>40.10^-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>+---------+---------+---------+---------+</td>
<td>---</td>
</tr>
<tr>
<td>51</td>
<td>x</td>
</tr>
<tr>
<td>52</td>
<td>x</td>
</tr>
<tr>
<td>53</td>
<td>x</td>
</tr>
<tr>
<td>54</td>
<td>x</td>
</tr>
<tr>
<td>55</td>
<td>x</td>
</tr>
<tr>
<td>56</td>
<td>x</td>
</tr>
<tr>
<td>57</td>
<td>x</td>
</tr>
<tr>
<td>58</td>
<td>x</td>
</tr>
<tr>
<td>59</td>
<td>x</td>
</tr>
<tr>
<td>60</td>
<td>x</td>
</tr>
<tr>
<td>61</td>
<td>x</td>
</tr>
<tr>
<td>62</td>
<td>x</td>
</tr>
</tbody>
</table>

V time
(+) history data       (i) interpolated value       (x)
forecast value

(1 row)

DROP_MODEL
-------------
t
(1 row)

**Related Functions**
► category Analytics - Time Series
► TIMESERIES
► PRINT_MODEL

**PRINT_TWOSTEP - Print a TwoStep clustering model**

Print the clusters, the centers or the column statistics of a TwoStep clustering model.

**Usage**

The PRINT_TWOSTEP stored procedure has the following syntax:

```
PRINT_TWOSTEP(NVARCHAR(ANY) paramString)
```

▲ Parameters

► **paramString**
   comma-separated list of <parameter>=<value> entries with parameters below
   
   Type: NVARCHAR(ANY)

► **model**
   the TwoStep clustering model to print
   
   Type: NVARCHAR(ANY)

► **clusters**
   the list of cluster IDs of the model to print. The cluster IDs are separated by a semi-colon (;). If not specified, information for all clusters are printed.
Type: NVARCHAR(ANY)

columns
the list of input column names of the model to print. The input column names are separated by a semicolon (;). If not specified, information for all input columns are printed.

Type: NVARCHAR(ANY)

mode
the display mode determining which information of the model to print. Allowed values are: clusters, centers, statistics.

Type: NVARCHAR(ANY)
Default: clusters

Returns
TEXT pretty print of the TwoStep clustering model

Details
The information about the TwoStep clustering model is printed out like a table for all clusters and all input columns that are indicated.

- If mode=clusters, a table with one row per cluster is returned with following columns: clusterid, name, description, size, withinss.

- If mode=centers, a table with one row per input column per cluster is returned with following columns: clusterid, columnname, cardinality, mode, minimum, maximum, mean, variance, count, importance. If the model was built with statistics=none or statistics=columns, output column importance is empty.

- If mode=statistics, a table with one row per value of the input columns and per cluster is returned with following columns: clusterid, columnname, value, count, relfrequency, deviation, mean and variance.

The rows printed out are sorted by clusterid, columnname and value (as applicable). The number of rows printed out is limited to about 1 Million characters. If more rows have to be printed out, an exception is raised.

A generic stored procedure PRINT_MODEL is available for all types of models that have a print stored procedure.

Examples

call nza..TWOSTEP('model=irisb, intable=nza..iris, distance=norm_euclidean, id=id, statistics=values:25');

CALL nza..PRINT_TWOSTEP('model=irisb, mode=centers, clusters=1;2, columns=class;petallength');

CALL nza..DROP_MODEL('model=irisb');

TWOSTEP
---------

2

(1 row)
```
PRINT_TWOSTEP

<table>
<thead>
<tr>
<th>CLUSTERID</th>
<th>COLUMNNAME</th>
<th>CARDINALITY</th>
<th>MODE</th>
<th>MINIMUM</th>
<th>MAXIMUM</th>
<th>MEAN</th>
<th>VARIANCE</th>
<th>COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CLASS</td>
<td>3</td>
<td>versicolor</td>
<td>50</td>
<td>1.9</td>
<td>1.464</td>
<td>0.028364102564103</td>
<td>50</td>
</tr>
<tr>
<td>2</td>
<td>CLASS</td>
<td>3</td>
<td>setosa</td>
<td>100</td>
<td>6.9</td>
<td>4.906</td>
<td>0.671688</td>
<td>100</td>
</tr>
</tbody>
</table>

(1 row)
```
### Related Functions

- category Analytics - Clustering
- TWOSTEP
- PRINT_MODEL

## PROJECT_PCA - Apply a Principal Component Analysis model

This stored procedure applies a Principal Component Analysis model to project the input data along the principal components of the model.

### Usage

The PROJECT_PCA stored procedure has the following syntax:

```sql
PROJECT_PCA(NVARCHAR(ANY) paramString)
```

**Parameters**

- **paramString**
  - comma-separated list of `<parameter>=<value>` entries with parameters below
  - Type: NVARCHAR(ANY)

- **model**
  - the name of the Principal Component Analysis model
  - Type: NVARCHAR(ANY)

- **intable**
  - the input table
  - Type: NVARCHAR(256)

- **outtable**
  - the output table where the projected data will be stored
  - Type: NVARCHAR(256)

- **id**
  - the input table column identifying a unique instance id
  - Type: NVARCHAR(128)
  - Default: `<id> column used to build the model`

- **pcNumber**
  - number of Principal Components used in the projection
IBM Netezza In-Database Analytics Reference Guide

Type: NVARCHAR(ANY)

▲ Returns
BOOLEAN Always TRUE

Details
This stored procedure applies a Principal Component Analysis model to project the input data along the principal components of the model.

The order of columns in the table of projected values is not important, the column names must match the column names of the input table on which the PCA model has been built.

Examples
CALL nza..PCA('model=wq_pca, intable=nza..WineQuality, id=id, scaleData=TRUE, centerData=TRUE, forceEigensolve=FALSE');

CALL nza..PROJECT_PCA('intable=nza..WineQuality, model=wq_pca, id=id, outtable=wq_proj4, pcNumber=4');

CALL nza..DROP_MODEL('model=wq_pca');

SELECT * FROM wq_proj4 ORDER BY id LIMIT 5;

CALL nza..DROP_TABLE('wq_proj4');

PCA
-----
t
(1 row)

PROJECT_PCA
-------------
t
(1 row)

DROP_MODEL
-------------
t
(1 row)

ID | PC1 | PC2 | PC3

458
|        PC4
+------------------+-------------------+-------------------
1 | -3.5429563417511 | 0.3550510560675   | -0.32579963947009 |
   | -1.7352335210206 |
2 | 0.61273723315898 | -0.28938152332064 | 0.81635108143025  |
   | 0.84864723812014 |
3 | -0.1423792719421 | 1.1679019509915   | -0.15305247901359 |
   | 0.19090003408588 |
4 | -1.3793842424156 | -0.19956694987349 | -0.32446338081778 |
   | -0.40878181730268 |
5 | -1.3793842424156 | -0.19956694987349 | -0.32446338081778 |
   | -0.40878181730268 |
(5 rows)

DROP_TABLE
----------
t
(1 row)

Related Functions
► category Analytics - Data Transformation
► PCA
► LIST_MODELS

PRUNE_DECTREE - Prune a Decision Tree model

This stored procedure prunes a previously built Decision Tree model

Usage
The PRUNE_DECTREE stored procedure has the following syntax:

► PRUNE_DECTREE(NVARCHAR(ANY) paramString)
  ▲ Parameters
  ► paramString
    comma-separated list of <parameter>=<value> entries with parameters below
    Type: NVARCHAR(ANY)
model  
the name of the Decision Tree model to prune  
Type: NVARCHAR(ANY)

valtable  
the input table  
Type: NVARCHAR(256)

d  
the input table column identifying a unique instance id  
Type: NVARCHAR(128)

target  
the input table column representing the class  
Type: NVARCHAR(128)

qmeasure  
the quality measure for pruning. Allowed values are Acc or wAcc.  
Type: NVARCHAR(ANY)  
Default: Acc

weights  
the input table containing optional instance or class weights for the input table columns.  
If the parameter is undefined, we assume that the weights are uniformly equal to 1.  
The <weights> table contains following columns:  
- weight: a numeric column containing the instance or class weight,  
- id: a column to be joined with the <id> column of <intable>, defining instance weights,  
- class: a column to be joined with the <target> column of <intable>, defining class weights.  
The id or class column can be missing, at least one of them must be present. For instances or classes not occurring in this table, weights of 1 are assumed.  
Type: NVARCHAR(256)  
Default: <none>

Returns  
INTEGER the number of nodes of the pruned tree (including leaves)

Details  
This stored procedure prunes a Decision tree model. The existing model is replaced by the pruned Decision Tree model. The model is saved to the database in a set of tables and registered in the database model metadata. Use the Model Management functions to further manipulate the model or access the model tables.
A bottom-up reduced error pruning algorithm is used. It bases on the prediction accuracy of the model against the input table.

Examples

```sql
CALL nza..GROW_DECTREE('model=adult_tree,
intable=nza..adult_train, id=id, target=income, eval=entropy,
minsplit=1000');

CALL nza..PRUNE_DECTREE('model=adult_tree,
valtable=nza..adult_prune, id=id, target=income,
qmeasure=wAcc');

CALL nza..DROP_MODEL('model=adult_tree');
```

```
GROW_DECTREE
--------------
  25
  (1 row)

PRUNE_DECTREE
--------------
  13
  (1 row)

DROP_MODEL
-------------
  t
  (1 row)
```

Related Functions

- category Analytics - Classification
- DECTREE
- GROW_DECTREE
- PRINT_DECTREE
- PREDICT_DECTREE
- LIST_MODELS

PRUNE_REGTREE - Prune a Regression Tree model

This stored procedure prunes a previously built Regression Tree model
Usage

The PRUNE_REGTREE stored procedure has the following syntax:

PRUNE_REGTREE(NVARCHAR(ANY) paramString)

Parameters

- paramString
  comma-separated list of <parameter>=<value> entries with parameters below
  Type: NVARCHAR(ANY)

- model
  the name of the Regression Tree model to prune
  Type: NVARCHAR(ANY)

- valtable
  the input table
  Type: NVARCHAR(256)

- id
  the input table column identifying a unique instance id
  Type: NVARCHAR(128)

- target
  the input table column representing the prediction target
  Type: NVARCHAR(128)

- qmeasure
  the quality measure for pruning the tree. Allowed values are: mse, r2, pearson, spearman.
  Type: NVARCHAR(ANY)
  Default: mse

Returns

INTEGER the number of nodes of the pruned tree (including leaves)

Details

This stored procedure prunes a Regression tree model. The existing model is replaced by the pruned Regression Tree model. The model is saved to the database in a set of tables and registered in the database model metadata. Use the Model Management functions to further manipulate the model or access the model tables.

A bottom-up reduced error pruning algorithm is used. It bases on the prediction accuracy of the model against the input table.

Examples

CALL nza..GROW_REGTREE('model=wrt, intable=nza..weather, id=instance, target=grade, minsplit=2, maxdepth=10');
CALL nza..PRUNE_REGTREE('model=wrt, valtable=nza..weather, id=instance, target=grade, qmeasure=mse');
CALL nza..DROP_MODEL('model=wrt');

GROW_REGTREE
-------------
15
(1 row)

PRUNE_REGTREE
-------------
15
(1 row)

DROP_MODEL
------------
t
(1 row)

Related Functions
► category Analytics - Regression
► GROW_REGTREE
► LIST_MODELS
► PREDICT_REGTREE
► PRINT_REGTREE
► REGTREE

PT - Cumulative T-student Distribution

Given a degree of freedom, this function returns the probability that a variable following the T-student distribution takes a value smaller or equal to x

Usage

The PT function has the following syntax:

► PT(DOUBLE x, INT8 df)
  ▲ Parameters
  ▶ x
  the value at which to compute
Type: DOUBLE

- df
  the number of degrees of freedom
  Type: INT8
  Min: 1

Returns
DOUBLE the cumulative probability distribution from minus infinity to point x

Details
Considering n independent random variables Z₁,...,Zₙ distributed according to the normal distribution with mean mu and a fixed variance, we create a new variable Z as their average. If s is the standard deviation from the sample, then the variable X = (Z - mu)/(s/sqrt(n)) follows the t-Student distribution with n-1 degrees of freedom. If df is 0 or less, PT(x,df) is null for all x values.

Examples

```sql
SELECT nza..PT(0.683,30);
```

<table>
<thead>
<tr>
<th>PT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.75007611634962</td>
</tr>
</tbody>
</table>

(1 row)

```sql
SELECT nza..PT(-2.821,9);
```

<table>
<thead>
<tr>
<th>PT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.010007152295943</td>
</tr>
</tbody>
</table>

(1 row)

Related Functions
- category Analytics - Probability Distributions
- DT
- PT_H
- QT
- T_LS_TEST
- T_ME_TEST
PT - Cumulative T-student Distribution with string argument

Given a degree of freedom, this function returns the probability that a variable following the T-student distribution takes a value greater than x. It differs from PT in that it takes both parameters as a single string.

Usage

The PT function has the following syntax:

```
PT(VARCHAR(ANY) paramString)
```

- **Parameters**
  - **paramString**
    - comma-separated list of <parameter>=<value> entries with parameters below
    - Type: VARCHAR(200)
  - **tStat**
    - the value at which to compute
    - Type: DOUBLE
    - Min: 0.00000000001
  - **df**
    - the number of degrees of freedom
    - Type: INT4
    - Min: 1

- **Returns**
  - DOUBLE the cumulative probability distribution from minus infinity to point x

Details

PT('tStat=<x>,df=<df>')=PT(<x>,<df>) for all <x> and <df> values.

Examples

```sql
SELECT nza..PT( nza..T_ME_TEST_S_AGG(petallength,3.7) ) FROM nza..iris;
```

```
PT
------------------
0.65778704505736
(1 row)
```
```
SELECT nza..PT('tStat= 0.407226,        df= 149');

PT
------------------
0.65778704505736
(1 row)
```

### Related Functions
- **category Analytics - Probability Distributions**
- **DT**
- **T_LS_TEST**
- **T_ME_TEST**
- **T_PMD_TEST**
- **T_UMD_TEST**

### PT_H - Cumulative T-student Distribution, high tail

Given a degree of freedom, this function returns the probability that a variable following the T-student distribution takes a value greater than $x$.

#### Usage

The PT_H function has the following syntax:

- **PT_H(DOUBLE x, INT8 df)**
  - **Parameters**
    - $x$ the value at which to compute
      Type: DOUBLE
    - $df$ the number of degrees of freedom
      Type: INT8
      Min: 1
  - **Returns**
    DOUBLE the cumulative probability distribution from $x$ to plus infinity

#### Details

$PT_H(x,df)=1-PT(x,df)$ for all $x$ and all strictly positive $df$ values.

If $df$ is 0 or less, $PT_H(x,df)$ is null for all $x$ values.
Examples

```sql
SELECT nza..PT_H(0.683, 30);
  PT_H
------------------
  0.24992388365038
(1 row)

SELECT nza..PT_H(-2.821, 9);
  PT_H
------------------
  0.98999284770406
(1 row)
```

Related Functions

- category Analytics - Probability Distributions
- DT
- PT

**PUNIF - Cumulative Uniform Distribution**

Given a minimum and a maximum for the uniform interval, this function returns probability that a variable following the Uniform distribution takes a value smaller or equal to \( x \)

**Usage**

The PUNIF function has the following syntax:

```sql
PUNIF(DOUBLE x, DOUBLE minimum, DOUBLE maximum)
```

- **Parameters**
  - **x**
    - the value at which to compute
    - Type: DOUBLE
    - Min: <minimum>
    - Max: <maximum>
  - **minimum**
    - the left border of the Uniform interval
Type: DOUBLE
Max: <maximum>

► maximum
the right border of the Uniform interval
Type: DOUBLE
Min: <minimum>

▲ Returns
DOUBLE the cumulative probability distribution from minus infinity to point x

Details
The Uniform distribution is the simplest form of a continuous probability distribution. Its probability density function is constant in the interval between minimum and maximum.
If minimum is not smaller than maximum or if x does not belong to the interval between minimum and maximum, PUNIF(x,minimum,maximum) is null.

Examples
SELECT nza..PUNIF(2,-3.0,11.3);

PUNIF
------------------
0.34965034965035 (1 row)

Related Functions
► category Analytics - Probability Distributions
► DUNIF
► PUNIF_H
► QUNIF

PUNIF_H - Cumulative Uniform Distribution, high tail
Given a minimum and a maximum for the uniform interval, this function returns the probability that a variable following the Uniform distribution takes a value greater than to x

Usage
The PUNIF_H function has the following syntax:
PUNIF_H(DOUBLE x, DOUBLE minimum, DOUBLE maximum)

Parameters

- **x**
  the value at which to compute
  Type: DOUBLE
  Min: <minimum>
  Max: <maximum>

- **minimum**
  the left border of the Uniform interval
  Type: DOUBLE
  Max: <maximum>

- **maximum**
  the right border of the Uniform interval
  Type: DOUBLE
  Min: <minimum>

Returns

DOUBLE the cumulative probability distribution from point x to plus infinity

Details

PUNIF_H(x,minimum,maximum)=1-PUNIF(x,minimum,maximum) for all x, minimum and maximum values where x is between minimum and maximum.

If minimum is not smaller than maximum or if x does not belong to the interval between minimum and maximum, PUNIF_H(x,minimum,maximum) is null.

Examples

```sql
SELECT nza..PUNIF_H(2,-3.0,11.3);

PUNIF_H
------------------
0.650349650349655
(1 row)
```

Related Functions

- **category** Analytics - Probability Distributions
- **DUNIF**
- **PUNIF**

PWALD - Cumulative Wald Distribution
Given a location and a shape, this function returns the probability that a variable following the Wald distribution takes a value smaller or equal to x

**Usage**

The PWALD function has the following syntax:

```
PWALD(DOUBLE x, DOUBLE location, DOUBLE shape)
```

▲ **Parameters**

- **x**
  - the value at which to compute
  - Type: DOUBLE

- **location**
  - the mean of the distribution
  - Type: DOUBLE
  - Min: 0.00000000001

- **shape**
  - the shape of the distribution
  - Type: DOUBLE
  - Min: 0.00000000001

▲ **Returns**

DOUBLE the cumulative probability distribution from minus infinity to point x

**Details**

The Wald distribution, called also Inverse Gaussian distribution, represents the first passage time for a Brownian motion. It can be used to represent market or fluid fluctuations.

If location or shape is 0 or less, PWALD(x,location,shape) is null for all x values.

**Examples**

```
SELECT nza..PWALD(2,2,2);

    PWALD
       ---------
       0.66810200122317
(1 row)

SELECT nza..PWALD(2,2,2), nza..PWALD(1,1,1), nza..PWALD(2,1,1), nza..PWALD(2,3,1), nza..PWALD(2,3,4);
```
Related Functions

- category Analytics - Probability Distributions
- DWALD
- PWALD_H
- QWALD

**PWALD_H - Cumulative Wald Distribution, high tail**

Given a location and a shape, this function returns the probability that a variable following the Wald distribution takes a value greater than \( x \)

**Usage**

The `PWALD_H` function has the following syntax:

\[
\text{PWALD_H(DOUBLE } x, \text{ DOUBLE location, DOUBLE shape)}
\]

**Parameters**

- **x**
  - the value at which to compute
  - Type: DOUBLE

- **location**
  - the mean of the distribution
  - Type: DOUBLE
  - Min: 0.00000000001

- **shape**
  - the shape of the distribution
  - Type: DOUBLE
  - Min: 0.00000000001

**Returns**

DOUBLE the cumulative probability distribution from \( x \) to plus infinity
Details

PWALD_H(x,location,shape) = 1-PWALD(x,location,shape) for all x and all positive location and shape values.

If location or shape is 0 or less, PWALD_H(x,location,shape) is null for all x values.

Examples

SELECT nza..PWALD_H(2,2,2), nza..PWALD_H(2,2,2) + nza..PWALD(2,2,2);

<table>
<thead>
<tr>
<th>PWALD_H</th>
<th>?COLUMN?</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.33189799877683</td>
<td>1</td>
</tr>
</tbody>
</table>

(1 row)

SELECT nza..PWALD_H(1,1,1), nza..PWALD_H(2,1,1), nza..PWALD_H(2,3,1), nza..PWALD_H(2,3,4);

<table>
<thead>
<tr>
<th>PWALD_H</th>
<th>PWALD_H</th>
<th>PWALD_H</th>
<th>PWALD_H</th>
</tr>
</thead>
<tbody>
<tr>
<td>+------------------</td>
<td>+------------------</td>
<td>+------------------</td>
<td>+------------------</td>
</tr>
<tr>
<td>0.33189799877683</td>
<td>0.11452457401399</td>
<td>0.36081045431102</td>
<td>0.54875921396817</td>
</tr>
</tbody>
</table>

(1 row)

Related Functions

► category Analytics - Probability Distributions
► DWALD
► PWALD

PWEIBULL - Cumulative Weibull (or Rosin-Rammer) Distribution

Given a scale and a shape, this function returns the probability that a variable following the Weibull distribution takes a value smaller or equal to x

Usage

The PWEIBULL function has the following syntax:
**PWEIBULL** (DOUBLE x, DOUBLE scale, DOUBLE kShape)

- **Parameters**
  - **x**
    - the value at which to compute
    - Type: DOUBLE
    - Min: 0
  - **scale**
    - the scale of the distribution
    - Type: DOUBLE
    - Min: 0.00000000001
  - **kShape**
    - the shape of the distribution
    - Type: DOUBLE
    - Min: 0.00000000001

- **Returns**
  - DOUBLE the cumulative probability distribution from minus infinity to point x

**Details**

The Weibull distribution gives a distribution of time between failures x, for which the failure rate is proportional to a power of time. The shape parameter kshape may be understood as follows:

- If it is lower than 1, then the failure rate decreases over time,
- If it is equal to 1, then the failure rate is constant over time,
- If it is greater than 1, then the failure rate increases over time ("aging process").

The cumulative Weibull distribution has the closed form: PWEIBULL(x, scale, kShape) = 1 - \exp(- (x/scale)^kShape) for all positive x, scale and kShape values.

The Exponential distribution is a special case of the Weibull distribution: PWEIBULL(x, scale, 1) = PEXP(x, scale). If x is less than 0, or if scale or shape are 0 or less, PWEIBULL(x, scale, kShape) is null.

**Examples**

```sql
SELECT nza..PWEIBULL(2, 4, 1);
```

```
PWEIBULL
------------------
0.39346934028737
(1 row)
```
Related Functions

- category Analytics - Probability Distributions
- DWEIBULL
- PEXP
- PWEIBULL_H
- QWEIBULL

PWEIBULL_H - Cumulative Weibull (or Rosin-Rammer) Distribution, high tail

Given a scale and a shape, this function returns the probability that a variable following the Weibull distribution takes a value greater than to x.

Usage

The PWEIBULL_H function has the following syntax:

- PWEIBULL_H(DOUBLE x, DOUBLE scale, DOUBLE kShape)

  ▲ Parameters
  - x
    the value at which to compute
    Type: DOUBLE
    Min: 0
  - scale
    the scale of the distribution
    Type: DOUBLE
    Min: 0.0000000001
  - kShape
    the shape of the distribution
    Type: DOUBLE
    Min: 0.0000000001

  ▲ Returns
  DOUBLE the cumulative probability distribution from point x to plus infinity

Details

PWEIBULL_H(x, scale, kShape) = 1 - PWEIBULL(x, scale, kShape) for all positive x, scale and kShape values.

If x is less than 0, or if scale or shape are 0 or less, PWEIBULL_H(x, scale, kShape) is null.
Examples

```
SELECT nza..PWEIBULL_H(2,4,1);
```

```
PWEIBULL_H
------------------
0.60653065971263
```

(1 row)

Related Functions

- category Analytics - Probability Distributions
- DWEIBULL
- PWEIBULL

**PWILCOX - Cumulative Wilcoxon Distribution**

Given the number of items, this function returns the probability that a variable following the Wilcoxon distribution takes a value smaller or equal to Wmin

**Usage**

The PWILCOX function has the following syntax:

```
PWILCOX(INT8 Wmin, INT8 NoItems)
```

- **Parameters**
  - **Wmin**
    - the value at which to compute
    - Type: INT8
    - Min: 0
    - Max: NoItems*(NoItems+1)/2
  - **NoItems**
    - the number of items
    - Type: INT8
    - Min: 1
    - Max: 15

- **Returns**
  - DOUBLE the cumulative probability distribution from minus infinity to point Wmin

**Details**

Given two variables x and y measured for the same objects, we split the objects into two sets: the first set
contains objects where \( x>y \) and the second set objects where \( x \leq y \). for each object we compute the rank of \( |x-y| \). The statistics \( sStat \) is calculated as the sum of ranks of objects belonging to the first set or to the second set, whichever is lower.

The function \( PWILCOX \) returns the probability that \( sStat \) is smaller or equal to \( Wmin \). This function has a limitation: the number of ranked items must be lower than or equal 15. For a higher number of items, approximations via normal distribution exist.

If \( Wmin \) or \( NoItems \) are not in their validity interval, \( PWILCOX(Wmin, NoItems) \) returns null.

**Examples**

```sql
SELECT nza..PWILCOX(42,15);

PWILCOX
------------------
0.16513061523438
(1 row)
```

**Related Functions**

- category Analytics - Probability Distributions
- DWILCOX
- PWILCOX_H
- QWILCOX
- WILCOXON_TEST

**PWILCOX_H - Cumulative Wilcoxon Distribution, high tail**

Given the number of items, this function returns the probability that a variable following the Wilcoxon distribution takes a value smaller or equal to \( Wmin \)

**Usage**

The \( PWILCOX_H \) function has the following syntax:

```sql
PWILCOX_H(INT8 Wmin, INT8 NoItems)
```

**Parameters**

- **Wmin**
  
  the value at which to compute
  
  Type: INT8
  
  Min: 0
  
  Max: Noitems*(Noitems+1)/2
**Noltems**
the number of items
Type: INT8
Min: 1
Max: 15

▲ Returns
DOUBLE the cumulative probability distribution from Wmin+1 to plus infinity

**Details**
PWILCOX_H(Wmin,Noltems)=1-PWILCOX(Wmin,Noltems) for all Wmin and Noltems values in their respective validity interval.
If Wmin or Noltems are not in their validity interval, PWILCOX_H(Wmin, Noltems) returns null.

**Examples**

```sql
SELECT nza..PWILCOX(42,15), nza..PWILCOX_H(42,15), nza..PWILCOX(42,15)+nza..PWILCOX_H(42,15);

<table>
<thead>
<tr>
<th>PWILCOX</th>
<th>PWILCOX_H</th>
<th>?COLUMN?</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.16513061523438</td>
<td>0.83486938476562</td>
<td></td>
</tr>
</tbody>
</table>

(1 row)
```

```sql
SELECT nza..PWILCOX(282,35), nza..PWILCOX_H(282,35), nza..PWILCOX(282,35)+nza..PWILCOX_H(282,35);

<table>
<thead>
<tr>
<th>PWILCOX</th>
<th>PWILCOX_H</th>
<th>?COLUMN?</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.29160523049112</td>
<td>0.70839476950888</td>
<td></td>
</tr>
</tbody>
</table>

(1 row)
```

```sql
SELECT nza..PWILCOX(420,150), nza..PWILCOX_H(420,150), nza..PWILCOX(420,150)+nza..PWILCOX_H(420,150);

<table>
<thead>
<tr>
<th>PWILCOX</th>
<th>PWILCOX_H</th>
<th>?COLUMN?</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

(1 row)
```
QBERN - Inverse Bernoulli Distribution

Given the success probability p, this function returns the value x of a variable following the Bernoulli distribution for which the probability of being smaller or equal to x is equal to the given percentage. The value x is the number of successes in a single Bernoulli trial.

Usage

The QBERN function has the following syntax:

QBERN(DOUBLE percentage, DOUBLE p)

Parameters

- **percentage**
  the probability limit requested
  Type: DOUBLE
  Min: 0.00000000001
  Max: 0.99999999999

- **p**
  the success probability
  Type: DOUBLE
  Min: 0.00000000001
  Max: 1-0.00000000001

Returns

INT8 the point x where PBERN(x,p)=percentage

Details

If PBERN(x,p)=perc, then QBERN(perc,p)=x for all positive x values and for p between 0 and 1.
If percentage is not between 0 and 1, QBERN(percentage,p) is null for all p values.
If p is not between 0 and 1, QBERN(percentage,p) is null for all percentage values.

Examples

```
SELECT nza..QBERN(0.95,0.330);
```

QBERN
Reference Documentation: Analytics

---

1
(1 row)

```sql
SELECT nza..QBERN(1.95,0.330);
```

QBERN
---

(1 row)

```sql
SELECT nza..QBERN(0.5,0.330);
```

QBERN
---

0
(1 row)

Related Functions

► category Analytics - Probability Distributions
► DBERN
► PBERN
► QBERN_H

QBERN_H - Inverse Bernoulli Distribution, high tail

Given the success probability p, this function returns the value x of a variable following the Bernoulli distribution for which the probability of being greater than x is equal to the given percentage. The value x is the number of successes in a single Bernoulli trial.

Usage

The QBERN_H function has the following syntax:

► QBERN_H(DOUBLE percentage, DOUBLE p)
  ▲ Parameters
    ► percentage
the probability limit requested  
Type: DOUBLE  
Min: 0.00000000001  
Max: 0.99999999999

► p  
the success probability  
Type: DOUBLE  
Min: 0.00000000001  
Max: 1-0.00000000001

▲ Returns  
INT8 the point x where PBERN_H(x,p)=percentage

Details  
If PBERN_H(x,p)=perc, then QBERN_H(perc,p)=x for all positive x values and for p between 0 and 1.  
QBERN_H(perc,p) = QBERN(1-perc,p) for all perc and p between 0 and 1.  
If percentage is not between 0 and 1, QBERN_H(percentage,p) is null for all p values.  
If p is not between 0 and 1, QBERN_H(percentage,p) is null for all percentage values.

Examples

```
SELECT nza..QBERN_H(0.95, 0.330), nza..QBERN(1-0.95, 0.330);

QBERN_H | QBERN
--------|-------
   0    |   0   
(1 row)

SELECT nza..QBERN_H(0.05, 0.330), nza..QBERN(1-0.05, 0.330);

QBERN_H | QBERN
--------|-------
   1    |   1   
(1 row)
```
SELECT nza..QBERN_H(0.6, 0.330), nza..QBERN(1-0.6, 0.330);

<table>
<thead>
<tr>
<th>QBERN_H</th>
<th>QBERN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

(1 row)

Related Functions

► category Analytics - Probability Distributions
► DBERN
► PBERN_H
► QBERN

QBETA - Inverse Beta Distribution

Given two shape parameters, this function returns the value x of a variable following the Beta distribution for which the probability of being smaller or equal to x is equal to the given percentage.

Usage

The QBETA function has the following syntax:

► QBETA(DOUBLE percentage, DOUBLE shape, DOUBLE shapeTwo)

▲ Parameters

► percentage
the probability limit requested
Type: DOUBLE
Min: 0.00000000001
Max: 0.99999999999

► shapeOne
the first shape of the distribution
Type: DOUBLE
Min: 0.00000000001

► shapeTwo
the second shape of the distribution
Type: DOUBLE
Min: 0.00000000001

▲ Returns
DOUBLE the point x where PBETA(x,shapeOne,shapeTwo)=percentage
Details

If \( PBETA(x,\text{shapeOne},\text{shapeTwo})=\text{perc} \), then \( QBETA(\text{perc},\text{shapeOne},\text{shapeTwo})=x \) for \( x \) between 0 and 1 and for all positive \text{shapeOne} and \text{shapeTwo} values. In this sense we speak about inverted function. Note that a function to be inverted must be strictly monotone in its domain. \( PBETA \) is so only for \( x \) between 0 an 1.

If percentage is not between 0 and 1, \( QBETA(\text{percentage},\text{shapeOne},\text{shapeTwo}) \) is null for all \text{shapeOne} and \text{shapeTwo} values.

If \text{shapeOne} or \text{shapeTwo} is 0 or less, \( QBETA(\text{percentage},\text{shapeOne},\text{shapeTwo}) \) is null for all percentage values.

Examples

```sql
SELECT nza..QBETA(0.0, 3, 0.4), nza..QBETA(0.0023477562161745, 3, 0.4), nza..QBETA(0.0019774546733614, 3, 0.4), nza..QBETA(0.0070656037462229, 3, 0.4), nza..QBETA(0.0178553415291, 3, 0.4);
```

<table>
<thead>
<tr>
<th>( QBETA )</th>
<th>( QBETA )</th>
<th>( QBETA )</th>
<th>( QBETA )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 0 )</td>
<td>0.10000000149012</td>
<td>0.20000000298023</td>
<td>( 0.29999998211861 )</td>
</tr>
<tr>
<td>(1 row)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Related Functions

► category Analytics - Probability Distributions
► DBETA
► PBETA
► QBETA_H

**QBETA_H - Inverse Beta Distribution, high tail**

Given two shape parameters, this function returns the value \( x \) of a variable following the Beta distribution for which the probability of being greater than \( x \) is equal to the given percentage

Usage

The QBETA_H function has the following syntax:
QBETA_H(DOUBLE percentage, DOUBLE shapeOne, DOUBLE shapeTwo)

Parameters

- **percentage**
  - the percentage for which x is to be identified
  - Type: DOUBLE
  - Min: 0.00000000001
  - Max: 0.99999999999

- **shapeOne**
  - the first shape of the distribution
  - Type: DOUBLE
  - Min: 0.00000000001

- **shapeTwo**
  - the second shape of the distribution
  - Type: DOUBLE
  - Min: 0.00000000001

Returns

DOUBLE the point x where PBETA_H(x,shapeOne,shapeTwo)=percentage

Details

If PBETA_H(x,shapeOne,shapeTwo)=perc, then QBETA_H(perc,shapeOne,shapeTwo)=x for x between 0 and 1 and for all positive shapeOne and shapeTwo values.

QBETA_H(perc,shapeOne,shapeTwo)=QBETA(1-perc,shapeOne,shapeTwo) for all positive shapeOne and shapeTwo values, and for all perc between 0 and 1.

If shapeOne or shapeTwo is 0 or less, QBETA_H(percentage,shapeOne,shapeTwo) is null for all percentage values.

Examples

```
SELECT nza..QBETA_H(1-0.0, 3, 0.4), nza..QBETA_H(1-0.00023477562161745, 3, 0.4), nza..QBETA_H(1-0.0019774546733614, 3, 0.4), nza..QBETA_H(1-0.0070656037462229, 3, 0.4), nza..QBETA_H(1-0.0178553415291, 3, 0.4);

<table>
<thead>
<tr>
<th>QBETA_H</th>
<th>QBETA_H</th>
<th>QBETA_H</th>
<th>QBETA_H</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.10000000149012</td>
<td>0.20000000298023</td>
<td>0.29999998211861</td>
<td>0.40000000596046</td>
</tr>
</tbody>
</table>
```

(1 row)
SELECT nza..QBETA_H(1-0.0, 3, 0.4), nza..QBETA_H(0.00023477562161745, 3, 0.4), nza..QBETA_H(0.0019774546733614, 3, 0.4), nza..QBETA_H(0.0070656037462229, 3, 0.4), nza..QBETA_H(0.0178553415291, 3, 0.4);

<table>
<thead>
<tr>
<th>QBETA_H</th>
<th>QBETA_H</th>
<th>QBETA_H</th>
<th>QBETA_H</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.99999994039536</td>
<td>0.99999994039536</td>
<td>0.99999886751175</td>
<td>0.99999837709427</td>
</tr>
</tbody>
</table>

(Q1 row)

Related Functions
- category Analytics - Probability Distributions
- DBETA
- PBETA_H
- QBETA

**QBINOM - Inverse Binomial Distribution**

Given the success probability p and the number of trials, this function returns the value x of a variable following the Binomial distribution for which the probability of being smaller or equal to x is equal to the given percentage. The value x is the number of successes in a serie of Bernoulli trials.

**Usage**

The QBINOM function has the following syntax:

- QBINOM(DOUBLE percentage, INT8 N, DOUBLE p)

**Parameters**

- **percentage**
  - the probability limit requested
  - Type: DOUBLE
  - Min: 0.00000000001
  - Max: 0.99999999999

- **N**
  - the number of trials
  - Type: INT8
Min: 1

p
the success probability
Type: DOUBLE
Min: 0.00000000001
Max: 1-0.00000000001

Returns
INT8 the point x where PBINOM(x,N,p)=percentage

Details
If PBINOM(x,N,p)=perc, then QBINOM(perc,N,p)=x for all positive x and N values and for p between 0 and 1.
If percentage is not between 0 and 1, QBINOM(percentage,N,p) is null for all p values.
If N is 0 or less, QBINOM(perc,N,p) is null for all perc and p values.
If p is not between 0 and 1, QBINOM(percentage,N,p) is null for all percentage and N values.

Examples

```
SELECT nza..QBINOM(0.95,1,0.330);
QBINOM
--------
    1
(1 row)
```

```
SELECT nza..QBINOM(0.215,3,0.4), nza..QBINOM(0.216,3,0.4),
nza..QBINOM(0.648,3,0.4), nza..QBINOM(0.936,3,0.4),
nza..QBINOM(1.0,3,0.4), nza..QBINOM(1.3,3,0.4);
QBINOM | QBINOM | QBINOM | QBINOM | QBINOM | QBINOM
--------+--------+--------+--------+--------+--------
      0 |      0 |      1 |      2 |   NULL |   NULL
(1 row)
```

```
SELECT nza..QBINOM(0.6,3,0.6), nza..QBINOM(0.6,30,0.6),
nza..QBINOM(0.6,300,0.6), nza..QBINOM(0.6,3000,0.6),
nza..QBINOM(0.6,30000,0.6), nza..QBINOM(0.6,300000,0.6),
nza..QBINOM(0.6,3000000,0.6);
QBINOM | QBINOM | QBINOM | QBINOM | QBINOM | QBINOM | QBINOM | QBINOM
--------+--------+--------+--------+--------+--------+--------+--------
      0 |      0 |      1 |      2 |      3 |      4 |      5 |      6
(1 row)
```
SELECT nza..QBINOM(0.2151,3,0.4),
nza..QBINOM(0.2161,3,0.4), nza..QBINOM(0.6481,3,0.4),
nza..QBINOM(0.9361,3,0.4), nza..QBINOM(1.001,3,0.4),
nza..QBINOM(1.31,3,0.4);

QBINOM | QBINOM | QBINOM | QBINOM | QBINOM | QBINOM
--------+--------+--------+--------+--------+--------
        |        |        |        |        | |
(1 row)

SELECT nza..QBINOM(0.215-0.0001,3,0.4),
nza..QBINOM(0.216-0.0001,3,0.4), nza..QBINOM(0.648-0.0001,3,0.4),
nza..QBINOM(0.936-0.0001,3,0.4), nza..QBINOM(1.-0.000001,3,0.4),
nza..QBINOM(1.3-0.0001,3,0.4);

QBINOM | QBINOM | QBINOM | QBINOM | QBINOM | QBINOM
--------+--------+--------+--------+--------+--------
        |        |        |        |        | |
(1 row)

SELECT nza..qbinom(0.785,3,0.6),
nza..QBINOM(0.784,3,0.6), nza..QBINOM(0.352,3, 0.6),
nza..QBINOM(0.064,3,0.6), nza..QBINOM(0,3,0.6);

QBINOM | QBINOM | QBINOM | QBINOM | QBINOM
--------+--------+--------+--------+--------
        |        |        |        |        |
(1 row)
SELECT nza..qbinom(0.785-0.0001,3,0.6), nza..QBINOM(0.784-0.0001,3,0.6), nza..QBINOM(0.352-0.001,3,0.6), nza..QBINOM(0.064-0.001,3,0.6), nza..QBINOM(0-0.001,3,0.6);

<table>
<thead>
<tr>
<th>QBINOM</th>
<th>QBINOM</th>
<th>QBINOM</th>
<th>QBINOM</th>
<th>QBINOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

(1 row)

SELECT nza..QBINOM(0.6,3000000000,0.6);

```
QBINOM
------------
1800006798
```

(1 row)

Related Functions

► category Analytics - Probability Distributions
► DBINOM
► PBINOM
► QBINOM_H

**QBINOM_H - Inverse Binomial Distribution, high tail**

Given the success probability \( p \) and the number of trials, this function returns the value \( x \) of a variable following the Binomial distribution for which the probability of being greater than \( x \) is equal to the given percentage. The value \( x \) is the number of successes in a series of Bernoulli trials.

**Usage**

The QBINOM_H function has the following syntax:

- **QBINOM_H**(DOUBLE percentage, INT8 N, DOUBLE p)
  - **Parameters**
    - **percentage**
      the probability limit requested
    - **N**
      Type: DOUBLE
      Min: 0.00000000001
      Max: 0.99999999999
N
the number of trials
Type: INT8
Min: 1

p
the success probability
Type: DOUBLE
Min: 0.00000000001
Max: 1-0.00000000001

Returns
INT8 the point x where PBINOM_H(x,N,p)=percentage

Details
If PBINOM_H(x,N,p)=perc, then QBINOM_H(perc,N,p)=x for all positive x and N values and for p between 0 and 1.
QBINOM_H(perc,N,p) = QBINOM(1-perc,N,p) for all positive x and N values and for p between 0 and 1.
If percentage is not between 0 and 1, QBINOM_H(percentage,N,p) is null for all p values.
If N is 0 or less, QBINOM_H(perc,N,p) is null for all perc and p values.
If p is not between 0 and 1, QBINOM_H(percentage,N,p) is null for all percentage and N values.

Examples
SELECT nza..QBINOM_H(0.95,1,0.330);

QBINOM_H
----------
  0
(1 row)

SELECT nza..QBINOM_H(1-0.215,3,0.4), nza..QBINOM_H(1-0.216,3,0.4), nza..QBINOM_H(1-0.648,3,0.4),
nza..QBINOM_H(1-0.936,3,0.4), nza..QBINOM_H(1-1.0,3,0.4), nza..QBINOM_H(1-1.3,3,0.4);

QBINOM_H | QBINOM_H | QBINOM_H | QBINOM_H | QBINOM_H | QBINOM_H
----------+----------+----------+----------+----------+----------
    +----------+
<table>
<thead>
<tr>
<th>nza..QBINOM_H(1-0.6, 3, 0.6), nza..QBINOM_H(1-0.6,30,0.6), nza..QBINOM_H(1-0.6,300,0.6), nza..QBINOM_H(1-0.6,3000,0.6), nza..QBINOM_H(1-0.6,30000,0.6), nza..QBINOM_H(1-0.6,300000,0.6), nza..QBINOM_H(1-0.6,3000000,0.6);</th>
</tr>
</thead>
<tbody>
<tr>
<td>QBINOM_H</td>
</tr>
<tr>
<td>----------+----------+----------+----------+----------+----------+----------</td>
</tr>
<tr>
<td>0 0 1 2</td>
</tr>
<tr>
<td>(1 row)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>nza..QBINOM_H(1-0.2151,3,0.4), nza..QBINOM_H(1-0.2161,3,0.4), nza..QBINOM_H(1-0.6481,3,0.4), nza..QBINOM_H(1-0.9361,3,0.4), nza..QBINOM_H(1-1.001,3,0.4), nza..QBINOM_H(1-1.31,3,0.4);</th>
</tr>
</thead>
<tbody>
<tr>
<td>QBINOM_H</td>
</tr>
<tr>
<td>----------+----------+----------+----------+----------+----------+----------</td>
</tr>
<tr>
<td>0 1 2 3</td>
</tr>
<tr>
<td>(1 row)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>nza..QBINOM_H(1-0.215-0.0001,3,0.4), nza..QBINOM_H(1-0.216-0.0001,3,0.4), nza..QBINOM_H(1-0.648-0.0001,3,0.4), nza..QBINOM_H(1-0.936-0.0001,3,0.4), nza..QBINOM_H(1-1-0.000001,3,0.4), nza..QBINOM_H(1-1.3-0.0001,3,0.4);</th>
</tr>
</thead>
<tbody>
<tr>
<td>QBINOM_H</td>
</tr>
<tr>
<td>----------+----------+----------+----------+----------+----------</td>
</tr>
<tr>
<td>0 1 2 3</td>
</tr>
<tr>
<td>(1 row)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>nza..QBINOM_H(1-0.215-0.0001,3,0.4), nza..QBINOM_H(1-0.216-0.0001,3,0.4), nza..QBINOM_H(1-0.648-0.0001,3,0.4), nza..QBINOM_H(1-0.936-0.0001,3,0.4), nza..QBINOM_H(1-1-0.000001,3,0.4), nza..QBINOM_H(1-1.3-0.0001,3,0.4);</th>
</tr>
</thead>
<tbody>
<tr>
<td>QBINOM_H</td>
</tr>
<tr>
<td>----------+----------+----------+----------+----------+----------</td>
</tr>
<tr>
<td>0 1 2 3</td>
</tr>
<tr>
<td>(1 row)</td>
</tr>
</tbody>
</table>
SELECT nza..QBINOM_H(1-0.785,3,0.6), nza..QBINOM_H(1-0.784,3,0.6), nza..QBINOM_H(1-0.352,3,0.6), nza..QBINOM_H(1-0.064,3,0.6), nza..QBINOM_H(1-0,3,0.6);

<table>
<thead>
<tr>
<th>QBINOM_H</th>
<th>QBINOM_H</th>
<th>QBINOM_H</th>
<th>QBINOM_H</th>
<th>QBINOM_H</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

(1 row)

SELECT nza..QBINOM_H(1-0.785-0.0001,3,0.6), nza..QBINOM_H(1-0.784-0.0001,3,0.6), nza..QBINOM_H(1-0.352-0.001,3,0.6), nza..QBINOM_H(1-0.064-0.001,3,0.6), nza..QBINOM_H(1-0-0.001,3,0.6);

<table>
<thead>
<tr>
<th>QBINOM_H</th>
<th>QBINOM_H</th>
<th>QBINOM_H</th>
<th>QBINOM_H</th>
<th>QBINOM_H</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

(1 row)

SELECT nza..QBINOM_H(1-0.6, 3000000000, 0.6);

<table>
<thead>
<tr>
<th>QBINOM_H</th>
</tr>
</thead>
<tbody>
<tr>
<td>1800006798</td>
</tr>
</tbody>
</table>

(1 row)

Related Functions
- category Analytics - Probability Distributions
- DBINOM
- PBINOM_H
- QBINOM

QCAUCHY - Inverse Cauchy Distribution
Given the peak location and the interquartile range, this function returns the value x of a variable
following the Cauchy distribution for which the probability of being smaller or equal to x is equal to the given percentage.

**Usage**

The QCAUCHY function has the following syntax:

- **QCAUCHY** (DOUBLE percentage, double location, double scale)

  - **Parameters**
    - **percentage**
      - the probability limit requested
      - Type: DOUBLE
      - Min: 0.000000000001
      - Max: 0.99999999999
    - **location**
      - the location of the peak of the Cauchy distribution
      - Type: DOUBLE
    - **scale**
      - a value corresponding to half of the interquartile range. Smaller values result in a narrower peak.
      - Type: DOUBLE
      - Min: 0.000000000001

  - **Returns**
    - DOUBLE the point x where PCAUCHY(x,location,scale)=percentage

**Details**

If PCAUCHY(x,location,scale)=perc, then QCAUCHY(perc,location,scale)=x for all x and location values and for all positive scale values. In this sense we speak about inverted function. Note that a function to be inverted must be strictly monotone in its domain. PCAUCHY is so because its derivative DCAUCHY is positive everywhere.

If scale=0 or less, QCAUCHY(perc,location,scale) is null for all perc and location values.

If percentage is not between 0 and 1, QCAUCHY(percentage,location,scale) is null for all location and scale values.

**Examples**

```
SELECT nza..QCAUCHY(0.95, 1, 0.330);
```

```
QCAUCHY
-----------------
3.0835379998428
(1 row)
```
SELECT nza..QCAUCHY(0.215, 3, 0.4), nza..QCAUCHY(0.216, 3, 0.4), nza..QCAUCHY(0.648, 3, 0.4), nza..QCAUCHY(0.936, 3, 0.4), nza..QCAUCHY(1.0, 3, 0.4), nza..QCAUCHY(1.3, 3, 0.4);

<table>
<thead>
<tr>
<th>QCAUCHY</th>
<th>QCAUCHY</th>
<th>QCAUCHY</th>
</tr>
</thead>
<tbody>
<tr>
<td>QCAUCHY</td>
<td>QCAUCHY</td>
<td>QCAUCHY</td>
</tr>
<tr>
<td>QCAUCHY</td>
<td>QCAUCHY</td>
<td>QCAUCHY</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>2.5007183854588</td>
<td>2.5039203343718</td>
<td>3.2006544803026</td>
</tr>
<tr>
<td>4.9625560021795</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(1 row)

SELECT nza..QCAUCHY(0.215, 3, 0.4), nza..QCAUCHY(0.215, 3, 1), nza..QCAUCHY(0.215, 3, 4);

<table>
<thead>
<tr>
<th>QCAUCHY</th>
<th>QCAUCHY</th>
<th>QCAUCHY</th>
</tr>
</thead>
<tbody>
<tr>
<td>QCAUCHY</td>
<td>QCAUCHY</td>
<td>QCAUCHY</td>
</tr>
<tr>
<td>QCAUCHY</td>
<td>QCAUCHY</td>
<td>QCAUCHY</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>2.5007183854588</td>
<td>1.751795963647</td>
<td>-1.9928161454122</td>
</tr>
</tbody>
</table>

(1 row)

**Related Functions**

- category Analytics - Probability Distributions
- DCAUCHY
- PCAUCHY
- QCAUCHY_H

**QCAUCHY_H - Inverse Cauchy Distribution, high tail**

Given the peak location and the interquartile range, this function returns the value x of a variable following the Cauchy distribution for which the probability of being greater than x is equal to the given percentage.

**Usage**

The QCAUCHY_H function has the following syntax:
QCAUCHY_H(DOUBLE percentage, double location, double scale)

Parameters

- **percentage**
  - the probability limit requested
  - Type: DOUBLE
  - Min: 0.00000000001
  - Max: 0.99999999999

- **location**
  - the location of the peak of the Cauchy distribution
  - Type: DOUBLE

- **scale**
  - a value corresponding to half of the interquartile range. Smaller values result in a narrower peak.
  - Type: DOUBLE
  - Min: 0.00000000001

Returns

DOUBLE the point x where PCAUCHY_H(x,location,scale)=percentage

Details

If PCAUCHY_H(x,location,scale)=perc, then QCAUCHY_H(perc,location,scale)=x.

QCAUCHY_H(perc,location,scale)=QCAUCHY(1-perc,location,scale) for all location values, for scale greater than 0 and for perc between 0 and 1.

If scale=0 or less, QCAUCHY_H(perc,location,scale) is null for all perc and location values.

If percentage is not between 0 and 1, QCAUCHY_H(percentage,location,scale) is null for all location and scale values.

Examples

```sql
SELECT nza..QCAUCHY_H(0.95,1, 0.330);
QCAUCHY_H
------------------
-1.0835379998428
(1 row)
```

```sql
SELECT nza..QCAUCHY_H(0.215, 3, 0.4), nza..QCAUCHY_H(0.216, 3, 0.4),nza..QCAUCHY_H(0.648, 3, 0.4),nza..QCAUCHY_H(0.936, 3, 0.4),nza..QCAUCHY_H(1.0, 3, 0.4),nza..QCAUCHY_H(1.3, 3, 0.4);
```

```sql
+ QCAUCHY_H | QCAUCHY_H | QCAUCHY_H |
+ QCAUCHY_H | QCAUCHY_H | QCAUCHY_H |
| -1.0835379998428 | -1.0835379998428 | -1.0835379998428 |
| 1.0835379998428 | 1.0835379998428 | 1.0835379998428 |
```

00X6332-00 Rev. 1 493
SELECT nza..QCAUCHY_H(0.215, 3, 0.4), nza..QCAUCHY_H(0.215, 3, 1), nza..QCAUCHY_H(0.215, 3, 4);

QCAUCHY_H | QCAUCHY_H | QCAUCHY_H
-----------------+----------------+-----------------
3.4992816145412 | 4.248204036353 | 7.9928161454122
(1 row)

Related Functions
► category Analytics - Probability Distributions
► DCAUCHY
► PCAUCHY_H
► QCAUCHY

QCHISQ - Inverse Chi-square Distribution

Given a degree of freedom, this function returns the value x of a variable following the Chi-square distribution for which the probability of being smaller or equal to x is equal to the given percentage.

Usage
The QCHISQ function has the following syntax:

► QCHISQ(DOUBLE percentage, INT8 df)
▲ Parameters
► percentage
the probability limit requested
Type: DOUBLE
Min: 0.000000000001
Max: 0.99999999999

- df
  the number of degrees of freedom
  Type: INT8
  Min: 1

▲ Returns
  DOUBLE the point x where PCHISQ(x,df)=percentage

Details
If PCHISQ(x,df)=perc, then QCHISQ(perc,df)=x. In this sense we speak about inverted function. Note that a function to be inverted must be strictly monotone in its domain. PCHISQ is so only for positive arguments because its derivative DCHISQ is positive only for positive values.

If percentage is not between 0 and 1, QCHISQ(percentage,df) is null for all df values.
If df is 0 or less, QCHISQ(percentage,df) is null for all percentage values.

Examples

```
SELECT nza..QCHISQ(0.95, 30);

QCHISQ
-------
 43.772971802033
(1 row)
```

Related Functions
- category Analytics - Probability Distributions
- DCHISQ
- PCHISQ
- PCHISQ_S
- QCHISQ_H

QCHISQ_H - Inverse Chi-square Distribution, high tail
Given a degree of freedom, this function returns the value x of a variable following the Chi-square distribution for which the probability of being greater than x is equal to the given percentage.

Usage
The QCHISQ_H function has the following syntax:

- QCHISQ_H(DOUBLE percentage, INT8 df)
Parameters

- **percentage**
  the probability limit requested
  
  Type: DOUBLE
  Min: 0.00000000001
  Max: 0.99999999999

- **df**
  the number of degrees of freedom
  
  Type: INT8
  Min: 1

Returns

DOUBLE the point x where PCHISQ_H(x,df)=percentage

Details

If PCHISQ_H(x,df)=perc, then QCHISQ_H(perc,df)=perc.

QCHISQ_H(perc,df)=QCHISQ(1-perc,df) for df greater than 0 and for perc between 0 and 1.

If percentage is not between 0 and 1, QCHISQ_H(percentage,df) is null for all df values.

If df is 0 or less, QCHISQ_H(percentage,df) is null for all percentage values.

Examples

```sql
SELECT nza..QCHISQ_H(1-0.95, 30);
```

```
QCHISQ_H
-----------------
43.772971802033
(1 row)
```

Related Functions

- **category** Analytics - Probability Distributions
- **DCHISQ**
- **PCHISQ_H**
- **QCHISQ**

QEXP - Inverse Exponential Distribution

Given an exponential scale, this function returns the value x of a variable following the Exponential distribution for which the probability of being smaller or equal to x is equal to the given percent-
Usage
The QEXP function has the following syntax:

- **QEXP**(DOUBLE percentage, DOUBLE scale)
  - Parameters
    - percentage
      - the probability limit requested
    - scale
      - the exponential scale
  - Returns
    - DOUBLE the point x where PEXP(x,scale)=percentage

Details
If PEXP(x,scale)=perc, then QEXP(perc,scale)=x.
If percentage is not between 0 and 1, QEXP(percentage,scale) is null for all df values.
If scale is 0 or less, QEXP(percentage,scale) is null for all percentage values.

Examples
```sql
SELECT nza..QEXP(0.2, 4);
```
```
QEXP
------------------
0.89257420525684
```
(1 row)

Related Functions
- category Analytics - Probability Distributions
  - DEXP
  - PEXP
  - QEXP_H
QEXP_H - Inverse Exponential Distribution, high tail

Given an exponential scale, this function returns the value x of a variable following the Exponential distribution for which the probability of being greater than x is equal to the given percentage.

Usage

The QEXP_H function has the following syntax:

```
QEXP_H(DOUBLE percentage, DOUBLE scale)
```

▲ Parameters

► **percentage**
  
  the probability limit requested
  
  Type: DOUBLE
  
  Min: 0.00000000001
  
  Max: 0.99999999999

► **scale**
  
  the exponential scale
  
  Type: DOUBLE
  
  Min: 0.00000000001

▲ Returns

DOUBLE the point x where PEXP_H(x,scale)=percentage

Details

If PEXP_H(x,scale)=perc, then QEXP_H(perc,scale)=x.

QEXP_H(perc,scale)=QEXP(1-perc,scale) for scale greater than 0 and for perc between 0 and 1.

If percentage is not between 0 and 1, QEXP_H(percentage,scale) is null for all scale values.

If scale is 0 or less, QEXP_H(percentage,scale) is null for all percentage values.

Examples

```
SELECT nza..QEXP_H(0.2, 4), nza..QEXP(1-0.2, 4);
```

```
QEXP_H      |      QEXP
-----------------+-----------------
6.4377516497364 | 6.4377516497364
```

(1 row)
Related Functions

- category Analytics - Probability Distributions
- DEXP
- PEXP_H
- QEXP

QF - Inverse Fisher Distribution

Given the degrees of freedom of the nominator and of the denominator, this function returns the value x of a variable following the Fisher distribution for which the probability of being smaller or equal to x is equal to the given percentage.

Usage

The QF function has the following syntax:

\[ \text{QF(DOUBLE percentage, INT8 dfNomin, INT8 dfDenom)} \]

▲ Parameters

- \textbf{percentage}
  the probability limit requested
  Type: DOUBLE
  Min: 0.00000000001
  Max: 0.99999999999

- \textbf{dfNomin}
  the number of degrees of freedom of the nominator
  Type: INT8
  Min: 1

- \textbf{dfDenom}
  the number of degrees of freedom of the denominator
  Type: INT8
  Min: 1

▲ Returns

DOUBLE the point x where PF(x,dfNomin,dfDenom)=percentage

Details

If PF(x,dfNomin,dfDenom)=perc, then QF(perc,dfNomin,dfDenom)=x.

If percentage is not between 0 and 1, QF(percentage,dfNomin,dfDenom) is null for all dfNomin and dfDenom values.

If dfNomin or dfDenom is 0 or less, QF(percentage,dfNomin,dfDenom) is null for all percentage values.

Examples

\[ \text{SELECT nza..QF(0.2, 2, 3)}; \]
Related Functions

► category Analytics - Probability Distributions
► DF
► PF
► QF_H

QF_H - Inverse Fisher Distribution, high tail

Given an exponential scale, this function returns the value x of a variable following the Fisher distribution for which the probability of being greater than x is equal to the given percentage.

Usage

The QF_H function has the following syntax:

► QF_H(DOUBLE percentage, INT8 dfNomin, INT8 dfDenom)

▲ Parameters

► percentage
the probability limit requested
Type: DOUBLE
Min: 0.00000000001
Max: 0.99999999999

► dfNomin
the number of degrees of freedom of the nominator
Type: INT8
Min: 1

► dfDenom
the number of degrees of freedom of the denominator
Type: INT8
Min: 1

▲ Returns
DOUBLE the point x where PF_H(x,dfNomin,dfDenom)=percentage
Details
If PF_H(x,dfNomin,dfDenom)=perc, then QF_H(perc,dfNomin,dfDenom)=x.
If percentage is not between 0 and 1, QF_H(percentage,dfNomin,dfDenom) is null for all dfNomin and df-Denom values.
If dfNomin or dfDenom is 0 or less, QF_H(percentage,dfNomin,dfDenom) is null for all percentage values.

Examples

```sql
SELECT nza..QF_H(0.2, 2, 3), nza..QF(1-0.2, 2, 3);
```

```
| QF_H      |       QF |
|----------------+-----------------|
| 2.886026349282 | 2.8860267314914 |
```

```
(1 row)
```

```sql
SELECT nza..QF_H(nza..PF_H(8,2,3), 2,3);  
```

```
<table>
<thead>
<tr>
<th>QF_H</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.9999994809429</td>
</tr>
</tbody>
</table>
```

```
(1 row)
```

Related Functions
► category Analytics - Probability Distributions
► DF
► PF_H
► QF

QFISK - Inverse Fisk (or log-logistic) Distribution

Given the median and the shape, this function returns the value x of a variable following the Fisk distribution for which the probability of being smaller or equal to x is equal to the given percentage.

Usage
The QFISK function has the following syntax:

```
QFISK(DOUBLE percentage, DOUBLE median, DOUBLE shape)
```

► Parameters
percentage
the probability limit requested
Type: DOUBLE
Min: 0.00000000001
Max: 0.99999999999

median
the median value of the distribution (also called scale)
Type: DOUBLE
Min: 0.000000001

shape
the shape of the distribution
Type: DOUBLE
Min: 0.000000001

Returns
DOUBLE the point x where PFISK(x,median,shape)=percentage

Details
The Inverse Fisk distribution has the closed form:
QFISK(p,median,shape) = median * (p/(1-p))^{1/shape}
If percentage is not between 0 and 1, QFISK(percentage,median,shape) is null for all positive median and shape values.
If median or shape is 0 or less, QFISK(percentage,median,shape) is null for all percentage values.

Examples

```
SELECT nza..QFISK(0.0031027056133611,5,8);
QFISK
-----
 2.43
(1 row)
```

```
SELECT nza..QFISK(nza..PFISK(0.7,5,4),5,4);
QFISK
-----
 0.7
```
QFISK_H - Inverse Fisk (or log-logistic) Distribution, high tail

Given the median and the shape, this function returns the value x of a variable following the Fisk distribution for which the probability of being greater than x is equal to the given percentage.

Usage

The QFISK_H function has the following syntax:

```
QFISK_H(DOUBLE percentage, DOUBLE median, DOUBLE shape)
```

Parameters

- **percentage**
  the probability limit requested
  Type: DOUBLE
  Min: 0.000000000001
  Max: 0.99999999999

- **median**
  the median value of the distribution (also called scale)
  Type: DOUBLE
  Min: 0.000000001

- **shape**
  the shape of the distribution
  Type: DOUBLE
  Min: 0.000000001

Returns

DOUBLE the point x where PFISK_H(x,median,shape)=percentage

Details

If PFISK_H(x,median,shape)=perc, then QFISK_H(perc,median,shape)=x for all positive x, median and shape values.

QFISK_H(perc,median,shape)=QFISK(1-perc,median,shape) for perc between 0 and 1 and for all positive median and shape values.
If percentage is not between 0 and 1, QFISK_H(percentage,median,shape) is null for all positive median and shape values.
If median or shape is 0 or less, QFISK_H(percentage,median,shape) is null for all percentage values.

**Examples**

```sql
SELECT nza..QFISK_H(0.0031027056133611,5,8),
       nza..QFISK(1-0.0031027056133611,5,8);
```

<table>
<thead>
<tr>
<th>QFISK_H</th>
<th>QFISK</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.288065843621</td>
<td>10.288065843621</td>
</tr>
</tbody>
</table>

(1 row)

```sql
SELECT nza..QFISK_H(nza..PFISK_H(0.7,5,4),5,4);
```

<table>
<thead>
<tr>
<th>QFISK_H</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.70000000000005</td>
</tr>
</tbody>
</table>

(1 row)

**Related Functions**

- category Analytics - Probability Distributions
- DFISK
- PFISK_H
- QFISK

**QGAMMA - Inverse Gamma Distribution**

Given a shape and an inverted scale, this function returns the value x of a variable following the Gamma distribution for which the probability of being smaller or equal to x is equal to the given percentage.

**Usage**

The QGAMMA function has the following syntax:
**QGAMMA**

**DOUBLE percentage, DOUBLE shape, DOUBLE scaleInv**

▲ Parameters

► **percentage**
the probability limit requested
Type: DOUBLE
Min: 0.00000000001
Max: 0.99999999999

► **shape**
the shape of the distribution
Type: DOUBLE
Min: 0.00000000001

► **scaleInv**
the inverted scale of the distribution
Type: DOUBLE
Min: 0.00000000001

▲ Returns
DOUBLE the point x where PGAMMA(x,shape,scaleInv)=percentage

**Details**

If PGAMMA(x,shape,scaleInv)=perc, then QGAMMA(perc,shape,scaleInv)=x for all x and for all positive shape and scaleInv values.

QGAMMA(percentage,shape,scaleInv)=QGAMMA_H(1-percentage,shape,scaleInv) for percentage between 0 and 1 and for all positive shape and scaleInv values.

If percentage is not between 0 and 1, QGAMMA(percentage,shape,scaleInv) is null for all shape and scaleInv values.

If shape or scaleInv is 0 or less, QGAMMA(percentage,shape,scaleInv) for all percentage values.

**Examples**

```
SELECT nza..QGAMMA(0.215,3,0.4), nza..QGAMMA(0.216,3,0.4), nza..QGAMMA(0.648,3,0.4), nza..QGAMMA(0.936,3,0.4), nza..QGAMMA(1.0,3,0.4), nza..QGAMMA(1.3,3,0.4);
```

<table>
<thead>
<tr>
<th>QGAMMA</th>
<th>QGAMMA</th>
<th>QGAMMA</th>
<th>QGAMMA</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.9841075165197</td>
<td>3.9937921699306</td>
<td>8.3431316468005</td>
<td>14.887879308886</td>
</tr>
</tbody>
</table>

(1 row)
Related Functions

- category Analytics - Probability Distributions
- DGAMMA
- PGAMMA
- QGAMMA_H

QGAMMA_H - Inverse Gamma Distribution, high tail

Given a shape and an inverted scale, this function returns the value $x$ of a variable following the Gamma distribution for which the probability of being greater than $x$ is equal to the given percentage.

Usage

The QGAMMA_H function has the following syntax:

```
QGAMMA_H(DOUBLE percentage, DOUBLE shape, DOUBLE scaleInv)
```

Parameters

- **percentage**
  - the probability limit requested
  - Type: DOUBLE
  - Min: 0.00000000001
  - Max: 0.99999999999

- **shape**
  - the shape of the distribution
  - Type: DOUBLE
  - Min: 0.00000000001

- **scaleInv**
  - the inverted scale of the distribution
  - Type: DOUBLE
  - Min: 0.00000000001

Returns

DOUBLE the point $x$ where $\text{PGAMMA}_H(x, \text{shape}, \text{scaleInv}) = \text{percentage}$

Details

If $\text{PGAMMA}_H(x, \text{shape}, \text{scaleInv}) = \text{perc}$, then $\text{QGAMMA}_H(\text{perc}, \text{shape}, \text{scaleInv}) = x$ for all $x$ and for all positive shape and scaleInv values.

$\text{QGAMMA}_H(\text{perc}, \text{shape}, \text{scaleInv}) = \text{QGAMMA}(1-\text{perc}, \text{shape}, \text{scaleInv})$ for all positive shapeOne and shapeTwo values, and for all perc between 0 and 1.

If percentage is not between 0 and 1, $\text{QGAMMA}_H(\text{percentage}, \text{shape}, \text{scaleInv})$ is null for all shape...
and scaleInv values.

If shapeOne or scaleInv is 0 or less, QGAMMA_H(percentage,shape,scaleInv) is null for all percentage values.

Examples

```sql
SELECT nza..QGAMMA_H(0.215,3,0.4), nza..QGAMMA_H(0.216,3,0.4),
      nza..QGAMMA_H(0.648,3,0.4), nza..QGAMMA_H(0.936,3,0.4),
      nza..QGAMMA_H(1.0,3,0.4), nza..QGAMMA_H(1.3,3,0.4);
```

```
QGAMMA_H     | QGAMMA_H     | QGAMMA_H     | QGAMMA_H     |
QGAMMA_H     | QGAMMA_H     | QGAMMA_H     |
-----------------+-----------------+-----------------+-----------------+----------+----------+-----------+-----------+-----------+-----------+-----------+-----------+-----------+-----------+-----------+-----------+-----------+-----------+-----------+-----------+
|          | 10.410824461052 | 10.392306644756 | 5.265106393617 |          | 2.2672819877043 |          |
(1 row)
```

Related Functions

- category Analytics - Probability Distributions
- DGAMMA
- PGAMMA_H
- QGAMMA

**QGEOM - Inverse Geometric Distribution**

Given the success probability p, this function returns the value x of a variable following the Geometric distribution for which the probability of being smaller or equal to x is equal to the given percentage. The value x is the number of failure before the first success in a serie of Bernoulli trials.

*Usage*

The QGEOM function has the following syntax:

```
QGEOM(DOUBLE percentage, DOUBLE p)
```

▲ **Parameters**

- **percentage**
  
  the probability limit requested

  Type: DOUBLE

  Min: 0.000000000001

  Max: 0.99999999999

- **p**
  
  the success probability
Type: DOUBLE
Min: 0.00000000001
Max: 0.99999999999

▲ Returns
INT8 the point x where PGEOM(x,p)=percentage

Details
If PGEOM(x,p)=perc, then QGEOM(perc,p)=x for all positive x values and for p between 0 and 1.
If percentage is not between 0 and 1, QGEOM(percentage,p) is null for all p values.
If p is not between 0 and 1, QGEOM(percentage,p) is null for all percentage values.

Examples

```sql
SELECT nza..QGEOM(0.670320,0.4), nza..QGEOM(0.93844807,0.4), nza..QGEOM(0.992073669,0.4), nza..QGEOM(0.999224,0.4);
```

```
+---+---+---+---+
| QGEOM | QGEOM | QGEOM | QGEOM |
| +---+---+---+---+
| 2  | 5  | 9  | 14  |
```

(1 row)

```sql
SELECT nza..QGEOM(nza..PGEOM(3,0.3)*0.4+nza..PGEOM(2,0.3)*0.6,0.3);
```

```
+---+
| QGEOM |
| +---+
| 3   |
```

(1 row)

```sql
SELECT nza..QGEOM(0.2,0.330);
```

```
+---+
| QGEOM |
| +---+
| 0    |
```

(1 row)
QGEOM_H - Inverse Geometric Distribution, high tail

Given the success probability p, this function returns the value x of a variable following the Geometric distribution for which the probability of being greater than x is equal to the given percentage. The value x is the number of failure before the first success in a series of Bernoulli trials.

Usage

The QGEOM_H function has the following syntax:

```sql
QGEOM_H(DOUBLE percentage, DOUBLE p)
```

**Parameters**

- `percentage`
  - the probability limit requested
  - Type: DOUBLE
  - Min: 0.00000000001
  - Max: 0.99999999999

- `p`
  - the success probability
  - Type: DOUBLE
  - Min: 0.00000000001
  - Max: 0.99999999999

**Returns**

INT8 the point x where PGEOM_H(x,p)=percentage

**Details**

If PGEOM_H(x,p)=perc, then QGEOM_H(perc,p)=x for all positive x values and for p between 0 and 1.
QGEOM_H(perc,p) = QGEOM(1-perc,p) for all perc and p between 0 and 1.
If percentage is not between 0 and 1, QGEOM_H(percentage,p) is null for all p values.
If p is not between 0 and 1, QGEOM_H(percentage,p) is null for all percentage values.

**Examples**

```sql
SELECT nza..QGEOM_H(1-0.670320,0.4), nza..QGEOM_H(1-0.93844807,0.4), nza..QGEOM_H(1-0.992073669,0.4), nza..QGEOM_H(1-0.999224,0.4);
```
QGEOM_H | QGEOM_H | QGEOM_H | QGEOM_H
---------+---------+---------+---------
   2 |     5 |     9 |    14
(1 row)

SELECT nza..QGEOM_H(nza..PGEOM_H(3,0.3)*0.1+nza..PGEOM_H(2,0.3)*0.9,0.3);
QGEOM_H
---------
   3
(1 row)

SELECT nza..QGEOM_H(0.2,0.330);
QGEOM_H
---------
   4
(1 row)

Related Functions
► category Analytics - Probability Distributions
► DGEOM
► PGEOM_H
► QGEOM

QHYPER - Inverse Hypergeometric Distribution

Given the number of white and black balls in the urn and the number of trials, this function returns the value x of a variable following the Hypergeometric distribution for which the probability of being smaller or equal to x is equal to the given percentage. The value x is the number of white balls drawn from an urn in a serie of trials without replacement.
Reference Documentation: Analytics

Usage
The QHYPER function has the following syntax:

► QHYPER(DOUBLE percentage, BIGINT wu, BIGINT bu, BIGINT N)

▲ Parameters

► percentage
the probability limit requested
Type: DOUBLE
Min: 0.00000000001
Max: 0.99999999999

► wu
the number of white balls in the urn
Type: BIGINT
Min: 0

► bu
the number of black balls in the urn
Type: BIGINT
Min: 0

► N
the number of balls drawn from the urn without replacement
Type: BIGINT
Min: 1

▲ Returns
BIGINT the point x where PHYPER(x,wu,bu,N)=percentage

Details
If PHYPER(x,wu,bu,N)=percentage, then QHYPER(percentage,wu,bu,N)=x for all x and for all positive wu, bu and N values.
If percentage is not between 0 and 1, QHYPER(percentage,wu,bu,N) is null for all N, wu and bu values.
If wu or bu is negative, QHYPER(percentage,wu,bu,N) is null for all percentage and N values.
If N is 0 or less, QHYPER(percentage,wu,bu,N) is null for all percentage, wu and bu values.

Examples

```
SELECT nza..QHYPER(0.95,4,5,6), nza..QHYPER(0.5,4,5,6);
QHYPER | QHYPER
--------+--------
     4 |      3
(1 row)
```
SELECT nza..QHYPER(0.32967032967033-0.0001,3,11,4),
nza..QHYPER( 0.82417582417582-0.0001,3,11,4),
nza..QHYPER(0.98901098901099-0.0001,3,11,4), nza..QHYPER(0.32967032967033+0.0001,3,11,4),
nza..QHYPER( 0.82417582417582+0.0001,3,11,4),
nza..QHYPER(0.98901098901099+0.0001,3,11,4),
nza..QHYPER(nza..PHYPER(2,3,11,4)*0.2+nza..PHYPER(1,3,11,4)*0.8,3,11,4);

QHYPER | QHYPER | QHYPER | QHYPER | QHYPER | QHYPER | QHYPER
---------+--------+--------+--------+--------+--------+--------
 0 | 1 | 2 | 1 | 2 | 3 |
 2
(1 row)

Related Functions
► category Analytics - Probability Distributions
► DHYPER
► PHYPER
► QHYPER_H

QHYPER_H - Inverse Hypergeometric Distribution, high tail

Given the number of white and black balls in the urn and the number of trials, this function returns the value x of a variable following the Hypergeometric distribution for which the probability of being greater than x is equal to the given percentage. The value x is the number of white balls drawn from an urn in a serie of trials without replacement.

Usage
The QHYPER_H function has the following syntax:

► QHYPER_H(DOUBLE percentage, BIGINT wu, BIGINT bu, BIGINT N)

▲ Parameters
► percentage
the probability limit requested

Type: DOUBLE
Min: 0.00000000001
Max: 0.99999999999

► **wu**
the number of white balls in the urn
Type: BIGINT
Min: 0

► **bu**
the number of black balls in the urn
Type: BIGINT
Min: 0

► **N**
the number of balls drawn from the urn without replacement
Type: BIGINT
Min: 1

▲ Returns
BIGINT the point x where PHYPER_H(x,wu,bu,N)=percentage

**Details**
If PHYPER_H(x,wu,bu,N)=percentage, then QHYPER_H(percentage,wu,bu,N)=x for all x and for all positive wu, bu and N values.

QHYPER_H(percentage,wu,bu,N) = QHYPER(1-percentage,wu,bu,N) for percentage between 0 and 1 and for all positive wu, bu and N values.

If percentage is not between 0 and 1, QHYPER_H(percentage,wu,bu,N) is null for all N, wu and bu values.

If wu or bu is negative, QHYPER_H(percentage,wu,bu,N) is null for all percentage, wu and bu values.

If N is 0 or less, QHYPER_H(percentage,wu,bu,N) is null for all percentage, wu and bu values.

**Examples**

```
SELECT nza..QHYPER_H(1-0.95,4,5,6), nza..QHYPER_H(1-0.5,4,5,6);
QHYPER_H   |   QHYPER_H
-----------+-----------
    4       |   3

(1 row)
```

```
SELECT nza..QHYPER_H(0.32967032967033-0.0001,3,11,4),
nza..QHYPER_H( 0.82417582417582-0.0001,3,11,4),
nza..QHYPER_H(0.98901098901099-0.0001,3,11,4),
nza..QHYPER_H( 0.32967032967033+0.0001,3,11,4),
nza..QHYPER_H( 0.82417582417582+0.0001,3,11,4),
```
QLNORM - Inverse Galton (or LogNormal) Distribution

Given a logarithmic scale median and shape, this function returns the value x of a variable following the LogNormal distribution for which the probability of being smaller or equal to x is equal to the given percentage.

Usage

The QLNORM function has the following syntax:

QLNORM(DOUBLE percentage, DOUBLE medlog, DOUBLE sdlog)

Parameters

- **percentage**
  the probability limit requested
  Type: DOUBLE
  Min: 0.00000000001
  Max: 0.99999999999

- **medlog**
  the median value on the logarithmic scale
  Type: DOUBLE

- **sdlog**
the shape on the logarithmic scale
Type: DOUBLE
Min: 0.000000001

▲ Returns
DOUBLE the point x where PLNORM(x,median,shape)=percentage

Details
If PLNORM(x,medlog,sdlog)=perc, then QLNORM(perc,medlog,sdlog)=x for all medlog and for all positive x and sdlog values.
If percentage is not between 0 and 1, QLNORM(percentage,medlog,sdlog) is null for all medlog and all positive sdlog values.
If sdlog is 0 or less, QLNORM(percentage,medlog,sdlog) is null for all percentage and medlog values.

Examples
SELECT nza..QLNORM(0.35542940976158,2,3);

QLNORM
-----------------
2.4299999999999

(1 row)

SELECT nza..QLNORM(nza..PLNORM(0.7,5,4),5,4);

QLNORM
--------
0.7

(1 row)

Related Functions
► category Analytics - Probability Distributions
► DLNORM
► PLNORM
► QLNORM_H

QLNORM_H - Inverse Galton (or LogNormal) Distribution, high tail
Given a logarithmic scale median and shape, this function returns the value x of a variable following the LogNormal distribution for which the probability of being greater than x is equal to the given percentage.
Usage

The QLNORM_H function has the following syntax:

► QLNORM_H(DOUBLE percentage, DOUBLE medlog, DOUBLE sdlog)

▲ Parameters

► percentage
  the probability limit requested
  Type: DOUBLE
  Min: 0.00000000001
  Max: 0.99999999999

► medlog
  the median value on the logarithmic scale
  Type: DOUBLE

► sdlog
  the shape on the logarithmic scale
  Type: DOUBLE
  Min: 0.000000001

▲ Returns
  DOUBLE the point x where PLNORM_H(x,medlog,sdlog)=percentage

Details

If PLNORM_H(x,medlog,sdlog)=perc, then QLNORM_H(perc,medlog,sdlog)=x for all medlog and for all positive x and sdlog values.

QLNORM_H(perc,medlog,sdlog)=QLNORM(1-p,medlog,sdlog) for perc between 0 and 1, for all medlog and for all positive sdlog values.

If percentage is not between 0 and 1, QLNORM_H(percentage,medlog,sdlog) is null for all medlog and for all positive sdlog values.

If sdlog is 0 or less, QLNORM_H(percentage,medlog,sdlog) is null for all percentage and medlog values.

Examples

SELECT nza..QLNORM_H(0.35542940976158,2,3),nza..QLNORM(1-0.35542940976158,2,3) ;

QLNORM_H | QLNORM
----------+----------
22.468374485955 | 22.468374485955
SELECT nza..QLNORM_H(1-0.35542940976158,2,3),nza..QLNORM(0.35542940976158,2,3);

QLNORM_H | QLNORM
-----------------+-----------------
2.4300000014364 | 2.4300000014364
(1 row)

SELECT nza..QLNORM_H(nza..PLNORM_H(0.7,5,4),5,4);

QLNORM_H
----------
0.7
(1 row)

Related Functions
► category Analytics - Probability Distributions
► DLNORM
► PLNORM_H
► QLNORM

QLOGIS - Inverse Logistic Distribution
Given the mean and the scale, this function returns the value x of a variable following the Logistic distribution for which the probability of being smaller or equal to x is equal to the given percentage.

Usage
The QLOGIS function has the following syntax:

► QLOGIS(DOUBLE percentage, DOUBLE mean, DOUBLE scale)
  ▲ Parameters
  ► percentage
    the probability limit requested
  Type: DOUBLE
Min: 0.00000000001
Max: 0.99999999999

► mean
the mean value of the distribution
Type: DOUBLE

► scale
the scale of the distribution
Type: DOUBLE
Min: 0.000000001

▲ Returns
DOUBLE the point x where PLOGIS(x,mean,scale)=percentage

Details
If PLOGIS(x,mean,scale)=perc then QLOGIS(perc,mean,scale)=x for all x and mean and for all positive scale values.
If percentage is not between 0 and 1, QLOGIS(percentage,mean,scale) is null for all mean and for all positive scale values.
If scale is 0 or less, QLOGIS(percentage,mean,scale) is null for all percentage and mean values.

Examples
SELECT nza..QLOGIS(0.53577211111408,2,3);

QLOGIS
---------
  2.43
(1 row)

SELECT nza..QLOGIS(nza..PLOGIS(0.7,5,4),5,4);

QLOGIS
---------
  0.7
(1 row)
Related Functions

- category Analytics - Probability Distributions
- DLOGIS
- PLOGIS
- QLOGIS_H

QLOGIS_H - Inverse Logistic Distribution, high tail

Given the mean and the scale, this function returns the value x of a variable following the Logistic distribution for which the probability of being greater than x is equal to the given percentage.

Usage

The QLOGIS_H function has the following syntax:

- QLOGIS_H(DOUBLE percentage, DOUBLE mean, DOUBLE scale)

Parameters

- **percentage**
  - the probability limit requested
  - Type: DOUBLE
  - Min: 0.00000000001
  - Max: 0.99999999999

- **mean**
  - the mean value of the distribution
  - Type: DOUBLE

- **scale**
  - the scale of the distribution
  - Type: DOUBLE
  - Min: 0.000000001

Returns

- DOUBLE the point x where PLOGIS_H(x,mean,scale)=percentage

Details

If PLOGIS_H(x,mean, scale)=perc, then QLOGIS_H(perc,mean, scale)=x for all x and mean and for all positive scale values.

QLOGIS_H(perc,mean, scale)=QLOGIS(1-perc,mean, scale) for perc between 0 and 1, for all mean and for all positive scale values.

If percentage is not between 0 and 1, QLOGIS_H(percentage,mean, scale) is null for all mean for all positive scale values.

If scale is 0 or less, QLOGIS_H(percentage,mean, scale) is null for all percentage and mean values.
Examples

```
SELECT nza..QLOGIS_H(nza..PLOGIS_H(0.7,5,4),5,4);
```

```
QLOGIS_H
----------
0.7
```

(1 row)

Related Functions

- category Analytics - Probability Distributions
- DLOGIS
- PLOGIS_H
- QLOGIS

**QMWW - Inverse Mann-Whitney-Wilcoxon Distribution**

Given the number of items separated in two sets, the minimum number of items in set 1, this function returns the value \( x \) of a variable following the MWW distribution for which the probability of being smaller or equal to \( x \) is equal to the given percentage.

**Usage**

The QMWW function has the following syntax:

```
QMWW(DOUBLE percentage, BIGINT COUNTmin, BIGINT NoItems)
```

▲ Parameters

- **percentage**
  the probability limit requested
  Type: DOUBLE
  Min: 0.0000001
  Max: 0.9999999

- **COUNTmin**
  the number of items in set 1
  Type: BIGINT
  Min: 1
  Max: NoItems-1

- **NoItems**
the total number of items
Type: BIGINT
Min: 1

▲ Returns
BIGINT the point x where PMWW(x,COUNTmin,NoItems)=percentage

Details
If PMWW(x,COUNTmin,NoItems)=perc, then QMWW(perc,COUNTmin,NoItems)=x for all x, COUNTmin and
NoItems values in their respective validity interval.
If percentage is not between 0 and 1, QMWW(percentage,COUNTmin,NoItems) is null for all COUNTmin and
NoItems values.
If COUNTmin or NoItems are not in their validity interval, QMWW(percentage,COUNTmin,NoItems) returns
null for all percentage values.

Examples
SELECT nza..QMWW(0.055,4,9);

QMWW
------
13
(1 row)

SELECT nza..QMWW(0.99,4,9);

QMWW
------
28
(1 row)

Related Functions
► category Analytics - Probability Distributions
► DMWW
► PMWW
► QMWW_H

QMWW_H - Inverse Mann-Whitney-Wilcoxon Distribution, high tail
Given the number of items separated in two sets, the minimum number of items in set 1, This function re-
turns the value \( x \) of a variable following the MWW distribution for which the probability of being greater than \( x \) is equal to the given percentage.

**Usage**

The QMWW\_H function has the following syntax:

\[
\text{QMWW\_H(DOUBLE percentage, BIGINT COUNTmin, BIGINT NoItems)}
\]

- **percentage**
  - the probability limit requested
  - Type: DOUBLE
  - Min: 0.0000001
  - Max: 0.9999999

- **COUNTmin**
  - the number of items in set 1
  - Type: BIGINT
  - Min: 1
  - Max: NoItems-1

- **NoItems**
  - the total number of items
  - Type: BIGINT
  - Min: 1

**Returns**

BIGINT the \( x \)

**Details**

If PMWW\_H\( (x,\text{COUNTmin},\text{NoItems})=\text{perc} \), then QMWW\_H\( (\text{perc},\text{COUNTmin},\text{NoItems})=x \) for all \( x \), \text{COUNTmin} and \text{NoItems} in their respective validity interval.

QMWW\_H\( (\text{perc},\text{COUNTmin},\text{NoItems})=\text{QMWW}(1-\text{perc},\text{COUNTmin},\text{NoItems}) \) for \text{perc} between 0 and 1, for all \text{COUNTmin} and \text{NoItems} in their respective validity interval.

If percentage is not between 0 and 1, QMWW\_H\( (\text{percentage},\text{COUNTmin},\text{NoItems}) \) is null for all \text{COUNTmin} and \text{NoItems}.

If \text{COUNTmin} or \text{NoItems} are not in their validity interval,
QMWW\_H\( (\text{percentage},\text{COUNTmin},\text{NoItems}) \) returns null for all percentage values.

**Examples**

\[
\text{SELECT nza..QMWW\_H(1-0.055,4,9)};
\]

\text{QMWW\_H}
Related Functions

- category Analytics - Probability Distributions
- DMWW
- PMWW_H
- QMW

QNBINOM - Inverse Negative Binomial Distribution

Given the success probability p and the number of successes needed, this function returns the value x of a variable following the Negative Binomial distribution for which the probability of being smaller or equal to x is equal to the given percentage. The value x is the number of failures in a serie of Bernoulli trials before the number of successes requested is reached.

Usage

The QNBINOM function has the following syntax:

- **QNBINOM(DOUBLE percentage, INT8 s, DOUBLE p)**

  ▲ Parameters

  ▶ **percentage**
  
  the probability limit requested

  Type: DOUBLE

  Min: 0.00000000001
  Max: 0.99999999999

  ▶ **s**
  
  the number of successes needed to stop the serie of Bernoulli trials

  Type: INT8
Min: 1

 ► p

 the success probability

 Type: DOUBLE

 Min: 0.00000000001

 Max: 1-0.00000000001

 Returns

 INT8 the point x where PNBINOM(x,s,p)=percentage

 Details

 If PNBINOM(x,s,p)=perc, then QNBINOM(perc,s,p)=x for all positive x and s values and for p between 0 and 1.

 If percentage is not between 0 and 1, QNBINOM(percentage,s,p) is null for all p values.

 If s is 0 or less, QNBINOM(perc,s,p) is null for all perc and p values.

 If p is not between 0 and 1, QNBINOM(percentage,s,p) is null for all percentage and s values.

 Examples

 SELECT nza..QNBINOM(0.95,1,0.330);

 QNBINOM

 -------

  7

 (1 row)

 SELECT nza..QNBINOM(0.215,3,0.4),
 nza..QNBINOM(0.1792,3,0.4), nza..QNBINOM(0.31744,3,0.4),
 nza..QNBINOM(0.45568,3,0.4),
 nza..QNBINOM(0.580096,3,0.4), nza..QNBINOM(1.3,3,0.4);

 QNBINOM | QNBINOM | QNBINOM | QNBINOM | QNBINOM | QNBINOM |
 QNBINOM

 ---------------+-------------------+-------------------+-------------------+-------------------+-------------------+

 +--------------+--------------+--------------+--------------+--------------+--------------|

  2 |  1 |  2 |  3 |  4 |

 (1 row)
SELECT nza..QNBINOM(0.215, 30, 0.4), nza..QNBINOM(0.1792, 30, 0.4),
nza..QNBINOM(0.31744, 30, 0.4), nza..QNBINOM(0.45568, 30, 0.4),
nza..QNBINOM(0.580096, 30, 0.4), nza..QNBINOM(1.3, 30, 0.4);

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>36</td>
<td>35</td>
<td>39</td>
<td>43</td>
<td>46</td>
<td></td>
</tr>
</tbody>
</table>

(1 row)

### Related Functions

- category Analytics - Probability Distributions
- DNBINOM
- PNBINOM
- QBINOM
- QNBINOM_H

### QNBINOM_H - Inverse Negative Binomial Distribution, high tail

Given the success probability p and the number of successes needed, this function returns the value x of a variable following the Negative Binomial distribution for which the probability of being greater than x is equal to the given percentage. The value x is the number of failures in a series of Bernoulli trials before the number of successes requested is reached.

#### Usage

The QNBINOM_H function has the following syntax:

- **QNBINOM_H(DOUBLE percentage, INT8 s, DOUBLE p)**
  - **percentage**
    - the probability limit requested
    - Type: DOUBLE
    - Min: 0.00000000001
    - Max: 0.99999999999
  - **s**
    - the number of successes needed to stop the series of Bernoulli trials
    - Type: INT8
    - Min: 1
  - **p**
    - the success probability
    - Type: DOUBLE
Returns
INT8 the point x where PNBINOM_H(x,s,p)=percentage

Details
If PNBINOM_H(x,s,p)=perc, then QNBINOM_H(perc,s,p)=x for all positive x and s values and for p between 0 and 1.

QNBINOM_H(perc,s,p) = QNBINOM(1-perc,s,p) for all positive s values and for percentage and p between 0 and 1.

If percentage is not between 0 and 1, QNBINOM_H(percentage,s,p) is null for all p values.
If s is 0 or less, QNBINOM_H(perc,s,p) is null for all perc and p values.
If p is not between 0 and 1, QNBINOM_H(percentage,s,p) is null for all percentage and s values.

Examples

```
SELECT nza..QNBINOM_H(1-0.95,1,0.330);

QNBINOM_H
-----------
    7
(1 row)
```

```
SELECT nza..QNBINOM_H(0.215,3,0.4),
nza..QNBINOM_H(0.1792,3,0.4),
nza..QNBINOM_H(0.31744,3,0.4),
nza..QNBINOM_H(0.45568,3,0.4),
nza..QNBINOM_H(0.580096,3,0.4),
nza..QNBINOM_H(1.3,3,0.4);

QNBINOM_H | QNBINOM_H | QNBINOM_H | QNBINOM_H |
QNBINOM_H | QNBINOM_H
-----------+-----------+-----------+-----------
    7 |    7 |      5 |      4 |
(1 row)
```
SELECT nza..QNBINOM_H(0.215,30,0.4), nza..QNBINOM_H(0.1792,30,0.4), nza..QNBINOM_H(0.31744,30,0.4), nza..QNBINOM_H(0.45568,30,0.4), nza..QNBINOM_H(0.580096,30,0.4), nza..QNBINOM_H(1.3,30,0.4);

<table>
<thead>
<tr>
<th>QNBINOM_H</th>
<th>QNBINOM_H</th>
<th>QNBINOM_H</th>
<th>QNBINOM_H</th>
<th>QNBINOM_H</th>
<th>QNBINOM_H</th>
</tr>
</thead>
<tbody>
<tr>
<td>53</td>
<td>55</td>
<td>49</td>
<td>46</td>
<td>42</td>
<td>42</td>
</tr>
</tbody>
</table>

(1 row)

Related Functions
- category Analytics - Probability Distributions
- DNBINOM
- PNBINOM_H
- QBINOM_H
- QNORM

QNORM - Inverse Standard Normal Distribution

This function returns the value x of a variable following the Standard Normal distribution for which the probability of being smaller or equal to x is equal to the given percentage.

Usage
The QNORM function has the following syntax:

QNORM(DOUBLE percentage)

- Parameters
  - percentage
    - the probability limit requested
    - Type: DOUBLE
    - Min: 0.00000000001
    - Max: 0.99999999999

- Returns
  - DOUBLE the point x where PNorm(x)=percentage

Details
If PNorm(x)=perc, then QNORM(perc)=x for all x values.
If percentage is not between 0 and 1, QNORM(percentage) is null.
Examples

```
SELECT nza.QNORM(0.7);
QNORM
--------------
0.52440051270804
(1 row)

SELECT nza.QNORM(nza.PNORM(0.7));
QNORM
-------
0.7
(1 row)
```

Related Functions

- category Analytics - Probability Distributions
- DNORM
- PNORM
- QNORM_H

QNORM3P - Inverse Normal Distribution

Given a mean value and a standard deviation, this function returns the value x of a variable following the Normal distribution for which the probability of being smaller or equal to x is equal to the given percentage.

Usage

The QNORM3P function has the following syntax:

```sql
QNORM3P(DOUBLE percentage, DOUBLE mean, DOUBLE stdev)
```

- Parameters
  - percentage
    - the probability limit requested
    - Type: DOUBLE
Min: 0.00000000001
Max: 0.99999999999

► mean
the mean value of the distribution
Type: DOUBLE

► stdev
the standard deviation of the distribution
Type: DOUBLE
Min: 0.00000000001

▲ Returns
DOUBLE the point x where PNORM3P(x,mean,stdev)=percentage

Details
If PNORM3P(x,mean,stdev)=perc, then QNORM3P(perc,mean,stdev)=x for all x and mean values and for all strictly positive stdev values.
If percentage is not between 0 and 1, QNORM3P(percentage,mean,stdev) is null for all mean and stdev values.
If stdev is 0 or less, QNORM3P(percentage,mean,stdev) is null for all percentage and mean values.

Examples

```
SELECT nza..QNORM3P(0.99245058858369,0,1);
QNORM3P
-----------------
2.4299999999999
(1 row)

SELECT nza..QNORM3P(0.99245058858369,6,1);
QNORM3P
-----------------
8.4299999999999
(1 row)

SELECT nza..QNORM3P(0.99245058858369,6,3);
QNORM3P
```
**QNORM_H - Inverse Standard Normal Distribution, high tail**

This function returns the value $x$ of a variable following the Standard Normal distribution for which the probability of being greater than $x$ is equal to the given percentage.

**Usage**

The QNORM_H function has the following syntax:

- **QNORM_H(DOUBLE percentage)**
  - Parameters
    - **percentage**
      - the probability limit requested
      - Type: DOUBLE
      - Min: 0.000000000001
      - Max: 0.99999999999
  - Returns
    - DOUBLE the point $x$ where PNORM_H($x$)=percentage

```sql
SELECT nza.QNORM3P(1-0.99245058858369,6,3);

QNORM3P
-1.2899999999998
(1 row)
```
Details
If PNORM_H(x)=perc, then QNORM_H(perc)=x for all x values.
QNORM_H(percentage)=QNORM(1-percentage) for percentage between 0 and 1.
If percentage is not between 0 and 1, QNORM_H(percentage) is null.

Examples

```sql
SELECT nza..QNORM_H(0.7);
QNORM_H
-------------------
-0.52440051270804
(1 row)
```

```sql
SELECT nza..QNORM_H(nza..PNORM_H(0.7));
QNORM_H
---------
0.7
(1 row)
```

Related Functions
► category Analytics - Probability Distributions
► DNORM
► PNORM_H
► QNORM

QPOIS - Inverse Poisson Distribution
Given the mean number of success within a fixed time interval, this function returns the value x of a variable following the Poisson distribution for which the probability of being smaller or equal to x is equal to the given percentage. The value x is the number of successes expected within this time interval.

Usage
The QPOIS function has the following syntax:

```
QPOIS(DOUBLE percentage, DOUBLE lambda)
```
▲ Parameters
percentage
the probability limit requested
Type: DOUBLE
Min: 0.00000000001
Max: 0.99999999999

lambda
the mean number of successes within a fixed time interval
Type: double
Min: 0.00000000001

Returns
INT8 the point x where PPOIS(x,lambda)=percentage

Details
If PPOIS(x,lambda)=perc, then QPOIS(perc,lambda)=x for all positive x and lambda values.
If percentage is not between 0 and 1, QPOIS(percentage,lambda) is null for all lambda values.
If lambda is 0 or less, PPOIS(x,lambda) is null for all x values.
Otherwise the probability of at most x successes is greater or equal to perc but the probability of
at most x-1 successes is smaller than perc Therefore, if one wants to draw a sample from the Pois-
son distribution with given lambda e.g. =3, then one shall call select nza..QPOIS(random(),3) ....

Examples

```
SELECT nza..QPOIS(0.670320, 0.4),
nza..QPOIS(0.93844807, 0.4), nza..QPOIS(0.992073669, 0.4),
nza..QPOIS(0.999224, 0.4);

QPOIS | QPOIS | QPOIS | QPOIS
-----+-------+-------+-------
  0   |     1 |     2 |     3
(1 row)
```

```
SELECT nza..QPOIS(nza..PPOIS(3, 0.3)*0.7+nza..PPOIS(2, 0.3)*0.3, 0.3);

QPOIS
-----
  3
```
SELECT nza..QPOIS(0.2, 0.330);

QPOIS
---
0

(1 row)

Related Functions
- category Analytics - Probability Distributions
- DPOIS
- PPOIS
- QPOIS_H

QPOIS_H - Inverse Poisson Distribution, high tail
Given the mean number of success within a fixed time interval, this function returns the value \( x \) of a variable following the Poisson distribution for which the probability of being greater than \( x \) is equal to the given percentage. The value \( x \) is the number of successes expected within this time interval.

Usage
The QPOIS_H function has the following syntax:

- **QPOIS_H**(DOUBLE percentage, DOUBLE lambda)
  - Parameters
    - **percentage**
      - the probability limit requested
      - Type: DOUBLE
      - Min: 0.000000000001
      - Max: 0.99999999999
    - **lambda**
      - the mean number of successes within a fixed time interval
      - Type: double
      - Min: 0.000000000001
  - Returns
    - INT8 the point \( x \) where PPOIS_H(x, lambda) = percentage
Details

If \( PPOIS_\text{H}(x,\lambda) = \text{perc} \), then \( QPOIS_\text{H}(\text{perc},\lambda) = x \) for all positive \( x \) and \( \lambda \) values.

\( QPOIS_\text{H}(\text{perc},\lambda) = QPOIS(1-\text{perc},\lambda) \) for all positive \( \lambda \) values and for \( \text{perc} \) between 0 and 1.

If percentage is not between 0 and 1, \( QPOIS_\text{H}(\text{percentage},\lambda) \) is null for all \( \lambda \) values.

If \( \lambda \) is 0 or less, \( PPOIS_\text{H}(x,\lambda) \) is null for all \( x \) values.

Otherwise the probability of at most \( x \) successes is greater or equal to 1-\( \text{perc} \) but the probability of at most \( x-1 \) successes is smaller than 1-\( \text{perc} \). Therefore, if one wants to draw a sample from the Poisson distribution with given \( \lambda \) e.g. \( =3 \), then one shall call select \( \text{nza..QPOIS(random(),3)} \) ....

Examples

SELECT \text{nza..QPOIS\_H(1-0.670320,0.4)}, \text{nza..QPOIS\_H(1-0.93844807,0.4)}, \text{nza..QPOIS\_H(1-0.992073669,0.4)}, \text{nza..QPOIS\_H(1-0.999224,0.4)};

\[
\begin{array}{|c|c|c|c|}
\hline
\text{QPOIS\_H} & \text{QPOIS\_H} & \text{QPOIS\_H} & \text{QPOIS\_H} \\ 
\hline
0 & 1 & 2 & 3 \\ 
\hline
\end{array}
\]

(1 row)

SELECT \text{nza..QPOIS\_H(nza..PPOIS\_H(3,0.3)*0.5+nza..PPOIS\_H(2,0.3)*0.5,0.3)};

\[
\begin{array}{|c|}
\hline
\text{QPOIS\_H} \\ 
\hline
3 \\ 
\hline
\end{array}
\]

(1 row)

SELECT \text{nza..QPOIS\_H(1-0.2,0.330)};

\[
\begin{array}{|c|}
\hline
\text{QPOIS\_H} \\ 
\hline
0 \\ 
\hline
\end{array}
\]

(1 row)
Related Functions
- category Analytics - Probability Distributions
- DPOIS
- PPOIS_H
- QPOIS

QT - Inverse t-Student Distribution
Given a degree of freedom, this function returns the value x of a variable following the T-student distribution for which the probability of being smaller or equal to x is equal to the given percentage.

Usage
The QT function has the following syntax:

► QT(DOUBLE percentage, INT8 df)
▲ Parameters
► percentage
  the probability limit requested
  Type: DOUBLE
  Min: 0.00000000001
  Max: 0.99999999999
► df
  the number of degrees of freedom
  Type: INT8
  Min: 1
▲ Returns
  DOUBLE the point x where PT(x,df)=percentage

Details
If PT(x,df)=perc, then QT(perc,df)=x for all x and all strictly positive df values.
If percentage is not between 0 and 1, QT(percentage,df) is null for all df values.
If df is 0 or less, QT(percentage,df) is null for all percentage values.

Examples

```
SELECT nza..QT(0.95,30);

QT
-----------------
1.6972612018151

(1 row)
```
Related Functions
► category Analytics - Probability Distributions
► DT
► PT
► QT_H

QT_H - Inverse t-Student Distribution, high tail
Given a degree of freedom, this function returns the value x of a variable following the T-student distribution for which the probability of being greater than x is equal to the given percentage.

Usage
The QT_H function has the following syntax:

► QT_H(DOUBLE percentage, INT8 df)
  ▲ Parameters
  ► percentage
    the probability limit requested
    Type: DOUBLE
    Min: 0.00000000001
    Max: 0.99999999999
  ▲ df
    the number of degrees of freedom
    Type: INT8
    Min: 1

  ▲ Returns
    DOUBLE the point x where PT_H(x,df)=percentage

Details
If PT_H(x,df)=perc, then QT_H(perc,df)=x for all x and for all strictly positive df values.
QT_H(perc,df)=QT(1-perc,df) for perc between 0 and 1, for all strictly positive df values.

If percentage is not between 0 and 1, QT_H(percentage,df) is null for all strictly positive df values.
If df is 0 or less, QT_H(percentage,df) is null for all percentage values.
Examples

```sql
SELECT nza..QT_H(0.95,30), nza..QT(1-0.95,30);
```

<table>
<thead>
<tr>
<th>QT_H</th>
<th>QT</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1.6972612018151</td>
<td>-1.6972612018151</td>
</tr>
</tbody>
</table>

(1 row)

Related Functions

► category Analytics - Probability Distributions
► DT
► PT_H
► QT

QUANTILE - Quantile for a numeric attribute

This stored procedure calculates quantile limit(s) for a numeric column

Usage

The QUANTILE stored procedure has the following syntax:

► QUANTILE(NVARCHAR(ANY) paramString)

▲ Parameters

► paramString
  comma-separated list of <parameter>=<value> entries with parameters below
  Type: NVARCHAR(ANY)

► intable
  the input table or view
  Type: NVARCHAR(256)

► incolumn
  the input table column
  Type: NVARCHAR(128)

► quantiles
  the list of quantiles to be calculated, separated by semicolon (;). Quantiles are values between 0 and 1 indicating the percentage of sorted values to be considered in each quantile.
  Type: NVARCHAR(ANY)

► outtable
the output table to write outliers to. This parameter must be specified if more than
one quantile have to be calculated. It is ignored and can be omitted if only one
quantile has to be calculated
Type: NVARCHAR(ANY)
Default: <none>

▲ Returns
REAL If only one quantile is requested, the value of the quantile; otherwise, the number of
calculated quantiles

Details
This stored procedure calculate the limits of the specified quantiles for a numeric column. Each
quantile is specified by the percentage (between 0 and 1) of input values that are below its limit.
If more than one quantiles are calculated, the output table contains following columns: p, value.
The column p contains the quantile percentage as indicated in parameter <quantiles> and the
column value contains the quantile limit.

Examples

CALL nza..QUANTILE('intable=nza..quant_iris,
incolumn=sepal_length, quantiles=0.3;0.7,
outtable=quant_qt');
SELECT * FROM quant_qt ORDER BY p;
CALL nza..DROP_TABLE('quant_qt');

QUANTILE
----------
2
(1 row)

| P  | VALUE |
|----+-------|
| 0.3 | 5.27  |
| 0.7 | 6.3   |
(2 rows)

DROP_TABLE
----------

t
CALL nza..QUANTILE('intable=nza..quant_iris, incolumn=sepal_length, quantiles=0.21');

<table>
<thead>
<tr>
<th>QUANTILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.029</td>
</tr>
</tbody>
</table>

(1 row)

Related Functions
► category Analytics - Quantiles & Outliers
► QUANTILE_DISC

**QUANTILE_DISC - Quantile for a discrete attribute**

This stored procedure calculates quantile limit(s) for a discrete column (when applicable)

**Usage**

The QUANTILE_DISC stored procedure has the following syntax:

► QUANTILE_DISC(NVARCHAR(ANY) paramString)

▲ Parameters
► paramString
coma-separated list of <parameter>=<value> entries with parameters below
Type: NVARCHAR(ANY)
► intable
t the input table or view
Type: NVARCHAR(256)
► incolumn
t the input table column
Type: NVARCHAR(128)
► quantiles
t the list of quantiles to be calculated, separated by semicolon (;). Quantiles are values between 0 and 1 indicating the percentage of sorted values to be considered in each quantile.
Type: NVARCHAR(ANY)
► outtable

the output table to write outliers to. This parameter must be specified if more than one quantile have to be calculated. It is ignored and can be omitted if only one quantile has to be calculated.

Type: NVARCHAR(ANY)

Default: <none>

▲ Returns

NVARCHAR If only one quantile is requested, the value of the quantile; otherwise, the number of calculated quantiles

Details

This stored procedure calculate the limits of the specified quantiles for a discrete column. Each quantile is specified by the percentage (between 0 and 1) of input values that are below its limit. If more than one quantiles are calculated, the output table contains following columns: p, value. The column p contains the quantile percentage as indicated in parameter <quantiles> and the column value contains the quantile limit.

Examples

CALL nza..QUANTILE_DISC('intable=nza..censusincome,
incolumn=marital_status, quantiles=0.3;0.7,
outtable=quant_qt');

SELECT * FROM quant_qt ORDER BY p;

CALL nza..DROP_TABLE('quant_qt');

QUANTILE_DISC
---------------

2

(1 row)

| P  | VALUE                        |
|-----+------------------------------|
| 0.3 | Married-civilian spouse present |
| 0.7 | Never married                |

(2 rows)

DROP_TABLE
------------
t
CALL nza..QUANTILE_DISC('intable=nza..censusincome, incolumn=marital_status, quantiles=0.21');

QUANTILE_DISC
---------------------------------
Married-civilian spouse present
(1 row)

Related Functions
► category Analytics - Quantiles & Outliers
► QUANTILE

QUARTILE - Quartile for a numeric attribute

This stored procedure calculates a quartile limit for a numeric column

Usage
The QUARTILE stored procedure has the following syntax:

► QUARTILE(NVARCHAR(ANY) paramString)
  ▲ Parameters
  ► paramString
    comma-separated list of <parameter>=<value> entries with parameters below
    Type: NVARCHAR(ANY)
  ► intable
    the input table or view
    Type: NVARCHAR(256)
  ► incolumn
    the input column
    Type: NVARCHAR(128)
  ► quartile
    the quartile to be calculated
    Type: INT4
    Default: 1
    Min: 1
Max: 3

▲ Returns
REAL the quartile limit

Details
This stored procedure calculates the quartile limit for a numeric column. The quartile parameter indicates which quartile limit has to be calculated: quartile=1 (0.25 quantile), quartile=2 (median value) and quartile=3 (0.75 quantile).

Examples

CALL nza..QUARTILE('intable=nza..quant_iris, incolumn=sepal_length, quartile=3');

<table>
<thead>
<tr>
<th>QUARTILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.4</td>
</tr>
</tbody>
</table>

(1 row)

CALL nza..QUARTILE('intable=nza..quant_iris, incolumn=sepal_length');

<table>
<thead>
<tr>
<th>QUARTILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
</tr>
</tbody>
</table>

(1 row)

Related Functions
► category Analytics - Quantiles & Outliers
► QUANTILE
► QUARTILE_DISC

QUARTILE_DISC - Quartile for a discrete attribute
This stored procedure calculates a quartile limit for a discrete column (when applicable)

Usage
The QUARTILE_DISC stored procedure has the following syntax:
QUARTILE_DISC(NVARCHAR(ANY) paramString)

Parameters

paramString
comma-separated list of parameter=value entries with parameters below
Type: NVARCHAR(ANY)

intable
the input table or view
Type: NVARCHAR(256)

incolumn
the input table column
Type: NVARCHAR(128)

quartile
the quartile to be calculated
Type: INT4
Default: 1
Min: 1
Max: 3

Returns
NVARCHAR(ANY) the quartile limit

Details
This stored procedure calculates the quartile limit for a discrete column. The quartile parameter indicates which quartile limit has to be calculated: quartile=1 (0.25 quantile), quartile=2 (median value) and quartile=3 (0.75 quantile).

Examples

CALL nza..QUARTILE_DISC('intable=nza..censusincome, incolumn=marital_status, quartile=3');

QUARTILE_DISC
---------------------
Never married
(1 row)

CALL nza..QUARTILE_DISC('intable=nza..censusincome, incolumn=marital_status');

QUARTILE_DISC
---------------------
Married-civilian spouse present
(1 row)

Related Functions
► category Analytics - Quantiles & Outliers
► QUANTILE_DISC
► QUARTILE

QUNIF - Inverse Uniform Distribution

Given a minimum and a maximum for the uniform interval, this function returns the value x of a variable following the Uniform distribution for which the probability of being smaller or equal to x is equal to the given percentage.

Usage
The QUNIF function has the following syntax:

► QUNIF(DOUBLE percentage, DOUBLE minimum, DOUBLE maximum)

▲ Parameters

► percentage
the probability limit requested
Type: DOUBLE
Min: 0.00000000001
Max: 0.99999999999

► minimum
the left border of the Uniform interval
Type: DOUBLE
Max: <maximum>

► maximum
the right border of the Uniform interval
Type: DOUBLE
Min: <minimum>

▲ Returns
DOUBLE the point x where PUNIF(x,minimum,maximum)=percentage

Details
If PUNIF(x,minimum,maximum)=perc, then QUNIF(perc,minimum,maximum)=x for all x, minimum
and maximum values where \( x \) is between minimum and maximum.

If percentage is not between 0 and 1, \( \text{QUNIF(percentage,minimum,maximum)} \) is null for all minimum and maximum values.

If minimum is not smaller than maximum, \( \text{QUNIF(percentage,minimum,maximum)} \) is null for all percentage values.

**Examples**

```sql
SELECT nza..QUNIF(nza..PUNIF(2,-3.0,11.3),-3.0,11.3);
```

```
QUNIF
-------
2
(1 row)
```

**Related Functions**

- category Analytics - Probability Distributions
- QUNIF
- PUNIF
- QUNIF_H

**QUNIF_H - Inverse Uniform Distribution, high tail**

Given a minimum and a maximum for the uniform interval, this function returns the value \( x \) of a variable following the Uniform distribution for which the probability of being greater than \( x \) is equal to the given percentage.

**Usage**

The QUNIF_H function has the following syntax:

```
QUNIF_H(DOUBLE percentage, DOUBLE minimum, DOUBLE maximum)
```

**Parameters**

- `percentage` - the probability limit requested
  
  Type: DOUBLE
  
  Min: 0.00000000001
  
  Max: 0.99999999999

- `minimum` - the left border of the Uniform interval
  
  Type: DOUBLE
Max: <maximum>

- **maximum**
  the right border of the Uniform interval

  Type: DOUBLE

  Min: <minimum>

▲ Returns

DOUBLE the point x where PUNIF_H(x,minimum,maximum)=percentage

**Details**

If PUNIF_H(x,minimum,maximum)=perc, then QUNIF_H(perc,minimum,maximum)=x for all x, minimum and maximum values where x is between minimum and maximum.

QUNIF_H(percentage,minimum,maximum)=QUNIF(1-percentage,minimum,maximum) for percentage between 0 and 1 and for all minimum and maximum values where minimum < maximum.

If percentage is not between 0 and 1, QUNIF_H(percentage,minimum,maximum) is null for all minimum and maximum values.

If minimum is not smaller than maximum, QUNIF_H(percentage,minimum,maximum) is null for all percentage values.

**Examples**

```sql
SELECT petallength, nza..PUNIF_H(petallength,1,6.9) as pp_H, nza..QUNIF_H(pp_H,1,6.9) AS inverse FROM nza..iris ORDER BY pp_H LIMIT 10;
```

<table>
<thead>
<tr>
<th>PETALLENGTH</th>
<th>PP_H</th>
<th>INVERSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.9</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>6.7</td>
<td>0.033898305084746</td>
<td>6.7</td>
</tr>
<tr>
<td>6.7</td>
<td>0.033898305084746</td>
<td>6.7</td>
</tr>
<tr>
<td>6.6</td>
<td>0.050847457627119</td>
<td>6.6</td>
</tr>
<tr>
<td>6.4</td>
<td>0.084745762711864</td>
<td>6.4</td>
</tr>
<tr>
<td>6.3</td>
<td>0.10169491525424</td>
<td>6.3</td>
</tr>
<tr>
<td>6.1</td>
<td>0.13559322033898</td>
<td>6.1</td>
</tr>
<tr>
<td>6.1</td>
<td>0.13559322033898</td>
<td>6.1</td>
</tr>
<tr>
<td>6.1</td>
<td>0.13559322033898</td>
<td>6.1</td>
</tr>
<tr>
<td>6</td>
<td>0.15254237288136</td>
<td>6</td>
</tr>
</tbody>
</table>

(10 rows)
SELECT nza..QUNIF_H(nza..PUNIF_H(2,-3.0,11.3),-3.0,11.3);

QUNIF_H
---------
     2
(1 row)

Related Functions
- category Analytics - Probability Distributions
- QUNIF
- PUNIF_H
- QUNIF

QWALD - Inverse Wald Distribution
Given a location and a shape, this function returns the value x of a variable following the Wald distribution for which the probability of being smaller or equal to x is equal to the given percentage.

Usage
The QWALD function has the following syntax:

QWALD(DOUBLE percentage, DOUBLE location, DOUBLE shape)

▲ Parameters
  ▶ percentage
  the probability limit requested
  Type: DOUBLE
  Min: 0.000000000001
  Max: 0.99999999999
  ▶ location
  the mean of the distribution
  Type: DOUBLE
  Min: 0.000000000001
  ▶ shape
  the shape of the distribution
  Type: DOUBLE
  Min: 0.000000000001

▲ Returns
DOUBLE the point \( x \) where \( \text{PWALD}(x, \text{location}, \text{shape}) = \text{percentage} \)

**Details**

If \( \text{PWALD}(x, \text{location}, \text{shape}) = \text{perc} \), then \( \text{QWALD}(\text{perc}, \text{location}, \text{shape}) = x \) for all \( x \) and all positive location and shape values.

If percentage is not between 0 and 1, \( \text{QWALD}(\text{percentage}, \text{location}, \text{shape}) \) is null for all location and shape values.

If location or shape is 0 or less, \( \text{QWALD}(\text{percentage}, \text{location}, \text{shape}) \) is null for all percentage values.

**Examples**

```sql
SELECT nza..QWALD(0.95,1,0.330);

QWALD
-----------------
3.9202609534676
(1 row)
```

```sql
SELECT nza..QWALD(0.215,3,0.4), nza..QWALD(0.216,3,0.4),
nza..QWALD(0.648,3,0.4), nza..QWALD(0.936,3,0.4),
nza..QWALD(1.0,3,0.4), nza..QWALD(1.3,3,0.4);

<table>
<thead>
<tr>
<th>QWALD</th>
<th>QWALD</th>
<th>QWALD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.23157796263695</td>
<td>0.23250064253807</td>
<td>1.2571559704842</td>
</tr>
<tr>
<td>10.634243061658</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
(1 row)
```

```sql
SELECT nza..QWALD(0.215,3,0.4), nza..QWALD(0.215,1), nza..QWALD(0.215,4);

<table>
<thead>
<tr>
<th>QWALD</th>
<th>QWALD</th>
<th>QWALD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.23157796263695</td>
<td>0.49881085753441</td>
<td>1.2064697228954</td>
</tr>
</tbody>
</table>
(1 row)
```
SELECT nza..PWALD(nza..QWALD(0.215,3,0.4),3,0.4),
nza..PWALD(nza..QWALD(0.215,3,1),3,1),
nza..PWALD(nza..QWALD(0.215,3,4),3,4);

<table>
<thead>
<tr>
<th>PWALD</th>
<th>PWALD</th>
<th>PWALD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.21500002412485</td>
<td>0.21499999261113</td>
<td>0.21499998376111</td>
</tr>
</tbody>
</table>

(1 row)

Related Functions
▶ category Analytics - Probability Distributions
▶ DWALD
▶ PWALD
▶ QWALD_H

QWALD_H - Inverse Wald Distribution, high tail

Given a location and a shape, this function returns the value x of a variable following the Wald distribution for which the probability of being greater than x is equal to the given percentage.

Usage
The QWALD_H function has the following syntax:

▶ QWALD_H(DOUBLE percentage, DOUBLE location, DOUBLE shape)

▲ Parameters
▶ percentage
  the probability limit requested
  Type: DOUBLE
  Min: 0.000000000001
  Max: 0.99999999999
▶ location
  the mean of the distribution
  Type: DOUBLE
  Min: 0.00000000001
▶ shape
  the shape of the distribution
  Type: DOUBLE
  Min: 0.00000000001
Returns

DOUBLE the point x where PWALD_H(x,location,shape)=percentage

Details

If PWALD_H(x,location,shape)=perc, then QWALD_H(perc,location,shape)=x for all x and all positive location and shape values.

QWALD_H(perc,location,shape)=QWALD(1-perc,location,shape) for all x and all positive location and shape values.

If percentage is not between 0 and 1, QWALD_H(percentage,location,shape) is null for all location and shape values.

If location or shape is 0 or less, QWALD_H(percentage,location,shape) is null for all percentage values.

Examples

```sql
SELECT nza..QWALD_H(0.95,1,0.330), nza..QWALD(1-0.95,1,0.330);

QWALD_H  |     QWALD
---------+---------
0.075281053781509  | 0.075281053781509
(1 row)
```

```sql
SELECT nza..QWALD_H(0.215,3,0.4), nza..QWALD_H(0.216,3,0.4), nza..QWALD_H(0.648,3,0.4), nza..QWALD_H(0.936,3,0.4), nza..QWALD_H(1.0,3,0.4), nza..QWALD_H(1.3,3,0.4);

QWALD_H  |     QWALD_H  |     QWALD_H      |
---------+---------------+------------------+
2.6490991284655  | 2.6320744413665  | 0.38724514320892 |
0.10962650179863  |               |                  |
(1 row)
```

```sql
SELECT nza..QWALD_H(0.215,3,0.4), nza..QWALD_H(0.215,3,1), nza..QWALD_H(0.215,3,4);
```
QWEIBULL - Inverse Weibull (or Rosin-Rammer) Distribution

Given a scale and a shape, this function returns the value x of a variable following the Weibull distribution for which the probability of being smaller or equal to x is equal to the given percentage.

Usage

The QWEIBULL function has the following syntax:

- QWEIBULL(DOUBLE percentage, DOUBLE scale, DOUBLE kShape)

  ▲ Parameters

  - percentage
    the probability limit requested
    Type: DOUBLE
    Min: 0.00000000001
    Max: 0.99999999999

  - scale
    the scale of the distribution
Type: DOUBLE
Min: 0.00000000001

- **kShape**
  the shape of the distribution

  Type: DOUBLE
  Min: 0.00000000001

- **Returns**
  DOUBLE the point x where PWEIBULL(x, scale, kShape) = percentage

**Details**
If PWEIBULL(x, scale, kShape) = perc then QWEIBULL(perc, scale, kShape) = x for all positive x, scale and kShape values.
If percentage is not between 0 and 1, QWEIBULL(percentage, scale, kShape) is null for all scale and kShape values.
If scale or kShape is 0 or less, QWEIBULL(percentage, scale, kShape) is null for all percentage values.

**Examples**
```
SELECT nza..QWEIBULL(nza..PWEIBULL(2,4,1),4,1);
QWEIBULL
---------
 2
(1 row)
```

**Related Functions**
- **category** Analytics - Probability Distributions
- DWEIBULL
- PWEIBULL
- QWEIBULL_H

---

**QWEIBULL_H - Inverse Weibull (or Rosin-Rammer) Distribution, high tail**

Given a scale and a shape, this function returns the value x of a variable following the Weibull distribution for which the probability of being greater than x is equal to the given percentage.

**Usage**
The QWEIBULL_H function has the following syntax:
**QWEIBULL_H(DOUBLE percentage, DOUBLE scale, DOUBLE kShape)**

**Parameters**

- **percentage**
  - the probability limit requested
  - Type: DOUBLE
  - Min: 0.00000000001
  - Max: 0.99999999999

- **scale**
  - the scale of the distribution
  - Type: DOUBLE
  - Min: 0.00000000001

- **kShape**
  - the shape of the distribution
  - Type: DOUBLE
  - Min: 0.00000000001

**Returns**

DOUBLE the point x where PWEIBULL_H(x,scale,kShape)=percentage

**Details**

If PWEIBULL_H(x,scale,kShape)=perc then QWEIBULL_H(perc,scale,kShape)=x for all positive x, scale and kShape values.

QWEIBULL_H(perc,scale,kShape)=QWEIBULL(1-perc,scale,kShape) for percentage between 0 and 1 and for all positive scale and kShape values.

If percentage is not between 0 and 1, QWEIBULL_H(percentage,scale,kShape) is null for all scale and kShape values.

If scale or kShape is 0 or less, QWEIBULL_H(percentage,scale,kShape) is null for all percentage values.

**Examples**

```sql
SELECT nza..QWEIBULL_H(nza..PWEIBULL_H(2,4,1),4,1);

QWEIBULL_H
-----------
 2
(1 row)
```

**Related Functions**

- category Analytics - Probability Distributions
QWILCOX - Inverse Wilcoxon Distribution

Given the number of items, this function returns the value x of a variable following the Wilcoxon distribution for which the probability of being smaller or equal to x is equal to the given percentage.

Usage

The QWILCOX function has the following syntax:

- **QWILCOX**(DOUBLE percentage, INT8 NoItems)
  - **Parameters**
    - **percentage**
      - the probability limit requested
      - Type: DOUBLE
      - Min: 0.0000001
      - Max: 0.9999999
    - **NoItems**
      - the number of items
      - Type: INT8
      - Min: 1
      - Max: 15
  - **Returns**
    - INT8 the point x where PWILCOX(x, NoItems) = percentage

Details

If PWILCOX(Wmin, NoItems) = perc then QWILCOX(perc, NoItems) = Wmin for all Wmin and NoItems values in their respective validity interval.

If percentage is not between 0 and 1, QWILCOX(percentage, NoItems) is null for all NoItems values.

If NoItems is not between 1 and 15, QWILCOX(percentage, NoItems) returns null for all percentage values.

Examples

```
SELECT nza..QWILCOX(0.16513565477462, 15);
QWILCOX
---------
554.0043332-00 Rev. 1
```
SELECT nza..QWILCOX(nza..PWILCOX(30,15)*0.7+nza..PWILCOX(29,15)*0.3,15) ;

QWILCOX
---------
30

(1 row)

Related Functions
► category Analytics - Probability Distributions
► DWILCOX
► PWILCOX
► QWILCOX_H

QWILCOX_H - Inverse Wilcoxon Distribution, high tail
Given the number of items, This function returns the value x of a variable following the Wilcoxon distribution for which the probability of being greater than x is equal to the given percentage

Usage
The QWILCOX_H function has the following syntax:

► QWILCOX_H(double percentage, int8 NoItems)
  ▲ Parameters
   ► percentage
     the probability limit requested
     Type: DOUBLE
     Min: 0.0000001
     Max: 0.9999999
   ► NoItems
     the number of items
     Type: INT8
     Min: 1
Max: 15

▲ Returns

INT8 the point x where PWILCOX_H(x,NoItems)=percentage

Details

If PWILCOX_H(Wmin,NoItems)=perc then QWILCOX_H(perc,NoItems)=Wmin for all Wmin and NoItems values in their respective validity interval.

QWILCOX_H(perc,NoItems)=QWILCOX(1-perc,NoItems) for all perc and NoItems values in their respective validity interval.

If percentage is not between 0 and 1, QWILCOX_H(percentage,NoItems) is null for all NoItems values.

If NoItems is not between 1 and 15, QWILCOX_H(percentage,NoItems) returns null for all percentage values.

Examples

SELECT nza..QWILCOX(0.16513,15), nza..QWILCOX_H(1-0.16513,15);

QWILCOX | QWILCOX_H
---------+-----------
  42     |   42

(1 row)

QWILCOX_H
---------
  30

(1 row)

Related Functions

► category Analytics - Probability Distributions
► DWILCOX
► PWILCOX_H
► QWILCOX

RAE - Relative Absolute Error
This stored procedure calculates the relative absolute error of Regression predictions.

**Usage**

The RAE stored procedure has the following syntax:

```sql
RAE(NVARCHAR(ANY) paramString)
```

**Parameters**

- `paramString` comma-separated list of `<parameter>=<value>` entries with parameters below
  - Type: NVARCHAR(ANY)
- `pred_table` the input table or view containing predicted values
  - Type: NVARCHAR(256)
- `pred_column` the input table column in `<pred_table>` containing predicted values
  - Type: NVARCHAR(128)
- `pred_id` the input table column in `<pred_table>` identifying a unique instance id
  - Type: NVARCHAR(128)
- `true_table` the input table or view containing real values
  - Type: NVARCHAR(256)
- `true_column` the input table column in `<true_table>` containing real values
  - Type: NVARCHAR(128)
- `true_id` the input table column in `<true_table>` identifying a unique instance id
  - Type: NVARCHAR(128)
- `check` flag indicating to check parameters or not. This may consume some time but prevents usage errors. Allowed values are: 'all' (check all), 'none' (check nothing), 'nulls' (check for nulls only).
  - Type: NVARCHAR(ANY)
  - Default: all

**Returns**

DOUBLE the Relative Absolute Error

**Details**

This stored procedure calculate the relative absolute error of Regression predictions. This is done by comparing the predictions made when applying a Regression model onto data, and the real values for this data. The RAE value is calculated as \( \frac{\sum|\text{true}_i-\text{pred}_i|}{\sum|\text{true}_i-\text{true}_m|} \) where \( \text{true}_i \) is the real
value, pred_i is the predicted value and true_m is the mean value of true_i.

Examples

CALL nza..GROW_REGTREE('model=wrt, intable=nza..weatherr, id=instance, target=grade, maxdepth=4, minsplit=2');

CALL nza..PREDICT_REGTREE('model=wrt, intable=nza..weatherr, id=instance, outtable=wpr, var=TRUE');

CALL nza..RAE('pred_table=wpr, true_table=nza..weatherr, pred_column=class, true_column=grade, pred_id=id, true_id=instance');

CALL nza..DROP_MODEL('model=wrt');

CALL nza..DROP_TABLE('wpr');

GROW_REGTREE
--------------

13
(1 row)

PREDICT_REGTREE
----------------

22
(1 row)

RAE
--------------

0.074829931972789
(1 row)

DROP_MODEL
----------

0
(1 row)
Related Functions
- category Analytics - Diagnostic Measures
- CERROR
- MAE
- MSE
- RSE

RANDOM_SAMPLE - Random Sample
This stored procedure creates a random sample of a table with a fixed size or a fixed probability

Usage
The RANDOM_SAMPLE stored procedure has the following syntax:

```
RANDOM_SAMPLE(NVARCHAR(ANY) paramString)
```

▲ Parameters
- **paramString**
  comma-separated list of <parameter>=<value> entries with parameters below
  Type: NVARCHAR(ANY)
- **intable**
  the input table
  Type: NVARCHAR(ANY)
- **outtable**
  the output table to write the sample into
  Type: NVARCHAR(ANY)
- **num**
  the number of rows in the sample (alias of size). If specified, the parameter <fraction> must not be specified. Only one of both parameters <num> and <size> must be specified.
  Type: BIGINT
- **size**
  the number of rows in the sample (alias of num). If specified, the parameter <fraction> must not be specified. Only one of both parameters <num> and <size> must be specified.
  Type: BIGINT
fraction
the probability of each row to be in the sample. If specified, the parameters <num> and <size> must not be specified. Otherwise, one of both parameters <num> or <size> must be specified.
Type: DOUBLE
Default: <none>
Min: 0.0
Max: 1.0

outsignature
the input table columns to keep in the sample, separated by a semi-colon (;). If not specified, all columns are kept in the output table.
Type: NVARCHAR(ANY)
Default: <none>

outclear
flag indicating whether the output table should be overwritten or not
Type: BOOLEAN
Default: false

replace
flag indicating if sampling should be done with replacement
Type: BOOLEAN
Default: false

randseed
the seed of the random function
Type: INT4
Default: random()*1000000.0

Returns
BIGINT the number of record in the sample output table

Details
This stored procedure creates a random sample of the rows of a table. It can ensure an exact number of rows in the sample or a fixed probability for each row.

Examples

CALL nza..RANDOM_SAMPLE('intable=nza..adult, size=1000, outtable=adult_size, outsignature=id;income, outclear=true, replace=true, randseed=11213');

CALL nza..DROP_TABLE('adult_size');
DROP_TABLE
---------
t
(1 row)

CALL nza..RANDOM_SAMPLE('intable=nza..adult, fraction=0.2, outtable=adult_fraction, randseed=11213');
CALL nza..DROP_TABLE('adult_fraction');

Related Functions
► category Analytics - Sampling

REGISTER_MODEL - Register a version 1.x analytics model to model management

This stored procedure adds a model created in version 1.x to the model management infrastructure of version 2. If the value of a parameter is not known anymore a question mark (?) could be specified if the default should not be used. But this could result in a model which could not be used or deliver wrong results in subsequent calls. Note that the calling user needs to have select privileges on the original model tables.
**Usage**

The REGISTER_MODEL stored procedure has the following syntax:

```
REGISTER_MODEL(NVARCHAR(ANY))
```

- **paramString**
  - comma-separated list of `<parameter>=<value>` entries with parameters below
  - Type: NVARCHAR(ANY)

- **model**
  - the model to be registered
  - Type: NVARCHAR(64)

- **procedure**
  - the stored procedure used to create the model. Allowed procedure names are: dectree, divcluster, kmeans, knn, linear_regression, naivebayes, pca, regtree, tanet_grow, tbnet_grow, tbnet1g, tbnet1g2p, tbnet2g
  - Type: NVARCHAR(ANY)

- **intable**
  - the input table used to build the model. Except for procedure=kmeans where it is not mandatory, the parameter intable needs to be specified. If not available anymore a table with the same column layout as used for scoring the model would also be fine.
  - Type: NVARCHAR(ANY)
  - Default: <none>

- **copy**
  - flag indicating that the original model table should be kept. It is not recommended to use false
  - Type: NVARCHAR(ANY)
  - Default: true

- **outtable**
  - the name where the scores should be stored. Allowed for procedure value: kmeans, divcluster, tbnet1g, tbnet1g2p and tbnet2g
  - Type: NVARCHAR(ANY)
  - Default: <none>

- **id**
  - the column name which uniquely identifies each row. Allowed for procedure value: kmeans, divcluster, dectree, regtree knn, naivebayes and linear_regression
  - Type: NVARCHAR(ANY)
  - Default: <none>

- **target**
name of the target column. Allowed for procedure value: kmeans, divcluster, dectree, regtree knn, naivebayes and linear_regression
Type: NVARCHAR(ANY)
Default: <none>

- **distance**
  the function to calculate the distance: euclidean, manhattan, maximum or canberra. Allowed for procedure value: divcluster and kmeans
  Type: NVARCHAR(ANY)
  Default: euclidean

- **k**
  the number of centers. Allowed for procedure value: kmeans
  Type: INTEGER
  Default: 3
  Min: 2

- **maxiter**
  the number of iterations to perform. Allowed for procedure value: kmeans and divcluster
  Type: INTEGER
  Default: 5
  Min: 1
  Max: 1000

- **randseed**
  the seed for the random number generation. Allowed for procedure value: kmeans and divcluster
  Type: INTEGER
  Default: 12345

- **minsplit**
  the minimum number of instances to split. Allowed for procedure value: divcluster, dectree and regtree
  Type: INTEGER
  Default: 5 (divcluster) 2 (dectree, regtree)
  Min: 2

- **maxdepth**
  the maximum depth of the hierarchy. Allowed for procedure value: divcluster, dectree and regtree
  Type: INTEGER
  Default: 3 (divcluster) 62 (dectree) 10 (regtree)
  Min: 1

- **weights**
table contain the weighs information. Allowed for procedure value: dectree
Type: NVARCHAR(ANY)
Default: <none>

► **eval**
the function used for split evaluation. Allowed for procedure value: dectree and regtree
dectree values: entropy, gini
regtree values: variance
Type: NVARCHAR(ANY)
Default: entropy (dectree) variance (regtree)

► **minimprove**
the minimum improvement required to split. Allowed for procedure value: dectree and regtree
Type: INTEGER
Default: 0.01 (dectree) 0.1 (regtree)
Min: 0.0

► **valtable**
the tablename containing the pruning dataset. Allowed for procedure value: dectree
Type: NVARCHAR(ANY)
Default: <none>

► **qmeasure**
the function of the quality measure for pruning. Allowed for procedure value: dectree and regtree
regtree values: mse, r2, pearson or spearman
dectree values: acc or wacc
Type: NVARCHAR(ANY)
Default: acc (dectree) mse (regtree)

► **nominalCols**
the list of nominal columns. Allowed for procedure value: linear_regression
Type: NVARCHAR(ANY)
Default: <none>

► **useSVDSolver**
the way to solve the matrix equation. Allowed for procedure value: linear_regression
Type: BOOLEAN
Default: false

► **includeIntercept**
the way to handle the intercept term. Allowed for procedure value: linear_regression
Type: BOOLEAN
Default: true

► **calculateDiagnostics**
define if diagnostics information should be displayed. Allowed for procedure value: linear_regression
Type: BOOLEAN
Default: false

► **forceEigensolve**
force use of less accurate but faster method of finding PCA. Allowed for procedure value: pca
Type: BOOLEAN
Default: false

► **centerData**
should procedure center the data before finding PCA (lack of data centering violates PCA assumptions). Allowed for procedure value: pca
Type: BOOLEAN
Default: true

► **scaleData**
should procedure scale the data before finding PCA. Allowed for procedure value: pca
Type: BOOLEAN
Default: false

► **saveScores**
should the PCA scores of individual observations be saved. Allowed for procedure value: pca
Type: BOOLEAN
Default: false

► **varlist**
the semicolon-separated list of columns in intable to compute BN, real. Allowed for procedure value: tbnet1g and tbnet_grow
Type: NVARCHAR(ANY)
Default: <none>

► **varlist1**
the semicolon-separated list of columns in intable to compute BN, real. Allowed for procedure value: tbnet2g and tbnet1g2p
Type: NVARCHAR(ANY)
Default: <none>

► **varlist2**
the semicolon-separated list of columns in intable to compute BN, real. Allowed for procedure value: tbnet2g and tbnet1g2p
**Type**: NVARCHAR(ANY)
**Default**: <none>

**baseidx**
the numeric id to be assigned to the first variable, for easier internal management. Allowed for procedure value: tbnet1g, tbnet2g, tbnet1g2p and tbnet_grow

**Type**: INTEGER
**Default**: 777

**samplesize**
the size of the sample to take if the number of records is too large. Allowed for procedure value: tbnet1g, tbnet_grow

**Type**: INTEGER
**Default**: 330000

**tanmodel**
result table name with the tan model stored. Allowed for procedure value: tanet_grow

**Type**: NVARCHAR(ANY)
**Default**: <none>

**class**
the target class. This should be a nominal variable. Allowed for procedure value: tanet_grow

**Type**: NVARCHAR(ANY)
**Default**: <none>

▲ Returns

INTEGER The model ID when successfully registered the model

**Details**

This stored procedure registers in the model management infrastructure an analytics model that has been created with version 1.x. The user needs to have SELECT privilege on the 1.x model to register the model.

The model will be migrated to the new table layout and the new parameter style.

Note: FPGROWTH model will no longer be supported. Therefore it is also not supported in the register function.

**Examples**

```
CREATE TABLE kmeans_iris (cluster_id INTEGER, class VARCHAR(16), petallength DOUBLE, petalwidth DOUBLE, sepallength DOUBLE, sepalwidth DOUBLE);
```

```
INSERT INTO kmeans_iris SELECT (MIN(id)-1)/50, COUNT(*), 0, class, AVG(petallength), AVG(petalwidth),
```
AVG(sepal_length), AVG(sepal_width) FROM nza..iris GROUP BY class;
CALL nza..register_model('procedure=kmeans, model=kmeans_iris, id=id, intable=nza..iris, outtable=kmeans_iris_out');
CALL nza..drop_model('model=kmeans_iris');
CALL nza..drop_table('kmeans_iris');
CALL nza..drop_table('kmeans_iris_out');

CREATE TABLE
INSERT 0 3
  REGISTER_MODEL
-----------------
   40
(1 row)

NOTICE: Dropped: KMEANS_IRIS
DROP_MODEL
------------
   t
(1 row)

DROP_TABLE
------------
   t
(1 row)

DROP_TABLE
------------
   t
(1 row)

Related Functions
► category Analytics - Model Management
► INITIALIZE
► IS_INITIALIZED
REGTREE - Build then prune a Regression Tree model

This stored procedure builds a Regression Tree model by growing and pruning a tree

Usage

The REGTREE stored procedure has the following syntax:

```sql
REGTREE(NVARCHAR(ANY) paramString)
```

Parameters

- **paramString**
  comma-separated list of `<parameter>=<value>` entries with parameters below
  Type: NVARCHAR(ANY)

- **model**
  the Regression Tree model to build
  Type: NVARCHAR(ANY)

- **intable**
  the input table
  Type: NVARCHAR(256)

- **id**
  the input table column identifying a unique instance id
  Type: NVARCHAR(128)

- **target**
  the input table column representing the prediction target
  Type: NVARCHAR(128)

- **incolumn**
  the input table columns with special properties, separated by a semi-colon (;).
  Each column is followed by one or several of the following properties:
  - its type: ':nom' (for nominal), ':cont' (for continuous). Per default, all numerical types are continuous, other types are nominal.
  - its role: ':id', ':target', ':input', ':ignore'.
    (Remark: ':objweight' is unsupported, i.e. ':objweight' same as ':ignore').
    (Remark: ':colweight(<wgt>)' is unsupported, i.e. ':colweight(<wgt>)' same as ':colweight(1)' same as ':input').
  If the parameter is undefined, all columns of the input table have default properties.
  Type: NVARCHAR(ANY)
  Default: <none>
► coldeftype
default type of the input table columns. Allowed values are 'nom' and 'cont'.
If the parameter is undefined, all numeric columns are considered continuous, other columns nominal.
Type: NVARCHAR(ANY)
Default: <none>

► coldefrole
default role of the input table columns. Allowed values are 'input' and 'ignore'.
If the parameter is undefined, all columns are considered 'input' columns.
Type: NVARCHAR(ANY)
Default: <none>

► colPropertiesTable
the input table where column properties for the input table columns are stored. The format of this table is the output format of stored procedure nza..COLUMN_PROPERTIES().
If the parameter is undefined, the input table column properties will be detected automatically.
(Remark: colPropertiesTable with "COLROLE" column with value 'objweight' is unsupported, i.e. same as 'ignore')
(Remark: colPropertiesTable with "COLWEIGHT" column with value '<wgt>' is unsupported, i.e. same as '1')
Type: NVARCHAR(256)
Default: <none>

► eval
the split evaluation measure. Allowed values are: variance.
Type: NVARCHAR(ANY)
Default: variance

► minimprove
the minimum improvement of the split evaluation measure required
Type: DOUBLE
Default: 0.1
Min: 0.00001

► minsplit
the minimum number of instances per tree node that can be split
Type: INTEGER
Default: 50
Min: 2

► maxdepth
the maximum number of tree levels (including leaves)
Type: INTEGER
Default: 10  
Min: 1  
Max: 62  
► valtable  
the input table containing the validation dataset. If this parameter is undefined, no pruning will be performed.  
Type: NVARCHAR(256)  
Default: <none>  
► qmeasure  
the quality measure for pruning the tree. Allowed values are: mse, r2.  
Type: NVARCHAR(ANY)  
Default: mse  
► statistics  
flags indicating which statistics to collect. Allowed values are: none, columns, values:n, all.  
If statistics=none, no statistics are collected.  
If statistics=columns, statistics on the input table columns like mean value are collected.  
If statistics=values:n with n a positive number, statistics about the columns and the column values are collected. Up to <n> column value statistics are collected:  
- If a nominal column contains more than <n> values, only the <n> most frequent column statistics are kept.  
- If a numeric column contains more than <n> values, the values will be discretized and the statistics will be collected on the discretized values.  
Indicating statistics=all is equal to statistics=values:100.  
Type: NVARCHAR(ANY)  
Default: 'none'  
▲ Returns  
INTEGER the number of Regression Tree nodes (including leaves)  

Details  
This stored procedure builds a Regression tree model by growing and (optionally) pruning the tree.  
The model is saved to the database in a set of tables and registered in the database model metadata. Use the Model Management functions to further manipulate the model or access the model tables.  
A top-down tree growing algorithm is used with the following features:  
- binary splits (equality-based for nominal attributes, inequality-based for continuous attributes),
- no missing value handling,
- variance for split evaluation,
- stop criteria satisfied when further splits do not improve the variance by at least \(<\text{minimprove}\>\),
- stop criteria satisfied when the number of instances is less than \(<\text{minsplit}\>\),
- stop criteria satisfied when the tree depth reaches \(<\text{maxdepth}\>\).

A bottom-up reduced error pruning algorithm is used. It bases on the prediction accuracy of the model against the validation dataset. The pruning is activated when parameter \(<\text{valtable}\>\) is specified.

**Examples**

```sql
CALL nza..REGTREE('intable=nza..weather, id=instance, target=grade, model=wrt, minsplit=2, maxdepth=4');
CALL nza..DROP_MODEL('model=wrt');
REGTREE
-------
  13
(1 row)

DROP_MODEL
---------
  t
(1 row)
```

**Related Functions**

- category Analytics - Regression
- GROW_REGTREE
- LIST_MODELS
- PREDICT_REGTREE
- PRINT_REGTREE
- PRUNE_REGTREE
- GROW_REGTREE

**REVOKE_MODEL - Revoke privileges on an analytics model**

This stored procedure revokes one or more privileges on an analytics model from users and/or groups.

**Usage**

The REVOKE_MODEL stored procedure has the following syntax:
REVOKE_MODEL(NVARCHAR(ANY))

Parameters

paramString
coma-separated list of <parameter>=<value> entries with parameters below
Type: NVARCHAR(ANY)

model
the model to revoke privileges from
Type: NVARCHAR(64)

privilege
a semicolon-separated list of privileges. Allowed privileges are: list, select, alter, update, drop.
Type: VARCHAR(256)

user
a semicolon-separated list of users to revoke privileges from. If not specified, no user
privileges are revoked, and the parameter group has to be specified.
Type: NVARCHAR(256)
Default: <none>

group
a semicolon-separated list of groups to revoke privileges from. If not specified, no
group privileges are revoked, and the parameter user has to be specified.
Type: NVARCHAR(256)
Default: <none>

Returns
BOOLEAN always true (otherwise an exception is raised)

Details
This stored procedure revokes privileges from users and groups on the given model. The possible
privileges are:
- LIST: Allows to list a model and its properties (using the views and the list procedures),
- SELECT: Allows to read the model contents (to SELECT the model tables, print the model, convert
  the model to PMML),
- ALTER: Allows to change the model properties (name, owner, ...),
- UPDATE: Allows to change the model contents using specific procedures,
- DROP: Allows to drop a model.

To revoke a privilege from a user or group (including group "PUBLIC"), you must have the LIST priv-
ilege on this user or group. By default, the owner of a model, the ADMIN user, and the database
owner have all privileges on a model.

Examples

```
CALL nza..ARULE('intable=nza..retail, model=mbamodel,
supporttype=percent, support=5, lvl=0, maxsetsize=5,
confidence=0.5');

CALL nza..GRANT_MODEL('model=mbamodel, privilege=LIST;SELECT,
user=inzauser');

CALL nza..REVOKE_MODEL('model=mbamodel, privilege=LIST;SELECT,
user=inzauser');

CALL nza..DROP_MODEL('model=mbamodel');
```

NOTICE:

RUNNING FPGrowth algorithm:

DATASET : "NZA".."RETAIL"
Transaction column : "TID"
Item column : "ITEM"
Group by : <none>
Minimum support : 5 %
Minimum confidence: 0.5
Max frequent itemset size : 5
Level of conditional dbs : 0
Result tables prefix : "NZA_META_MBAMODEL"

```
ARULE
-------
  14
(1 row)

GRANT_MODEL
------------
   t
(1 row)

REVOKE_MODEL
------------
```
Related Functions

- category Analytics - Model Management
- LIST_MODELS
- GRANT_MODEL

RSE - Relative Squared Error

This stored procedure calculates the relative squared error of Regression predictions.

Usage

The RSE stored procedure has the following syntax:

RSE(NVARCHAR(ANY) paramString)

Parameters

- paramString
  comma-separated list of <parameter>=<value> entries with parameters below
  Type: NVARCHAR(ANY)

- pred_table
  the input table or view containing predicted values
  Type: NVARCHAR(256)

- pred_column
  the input table column in <pred_table> containing predicted values
  Type: NVARCHAR(128)

- pred_id
  the input table column in <pred_table> identifying a unique instance id
  Type: NVARCHAR(128)
true_table
the input table or view containing real values
Type: NVARCHAR(256)

true_column
the input table column in <true_table> containing real values
Type: NVARCHAR(128)

true_id
the input table column in <true_table> identifying a unique instance id
Type: NVARCHAR(128)

check
flag indicating to check parameters or not. This may consume some time but prevents usage errors. Allowed values are: 'all' (check all), 'none' (check nothing), 'nulls' (check for nulls only).
Type: NVARCHAR(ANY)
Default: all

Returns
DOUBLE the Relative Squared Error

Details
This stored procedure calculate the relative squared error of Regression predictions. This is done by comparing the predictions made when applying a Regression model onto data, and the real values for this data.
The RSE value is calculated as sum((true_i-pred_i)^2)/sum((true_i-true_m)^2) where true_i is the real value, pred_i is the predicted value and true_m is the mean value of true_i.

Examples
CALL nza..GROW_REGTREE('model=wrt, intable=nza..weatherr, id=instance, target=grade, maxdepth=4, msplit=2');
CALL nza..PREDICT_REGTREE('model=wrt, intable=nza..weatherr, id=instance, outtable=wpr, var=TRUE');
CALL nza..RSE('pred_table=wpr, true_table=nza..weatherr, pred_column=class, true_column=grade, pred_id=id, true_id=instance');
CALL nza..DROP_MODEL('model=wrt');
CALL nza..DROP_TABLE('wpr');

GROW_REGTREE
--------------
13
(1 row)

PREDICT_REGTREE
Related Functions

- category Analytics - Diagnostic Measures
- CERROR
- MAE
- RAE
- MSE

SET_CLUSTERNAME - set name and/or description of a K-means cluster

This stored procedure allows modification of the name and/or description of a cluster in a Clustering model.

Usage

The SET_CLUSTERNAME stored procedure has the following syntax:
**SET_CLUSTERNAME**(NVARCHAR(ANY) paramString)

- **Parameters**
  - **paramString**
    - comma-separated list of <parameter>=<value> entries with parameters below
    - **model**
      - the name of clustering model
      - Type: NVARCHAR(ANY)
    - **clusterid**
      - the number of the cluster
      - Type: INTEGER
    - **name**
      - the name of the cluster
      - Type: NVARCHAR(100)
      - Default: clusterid
    - **description**
      - textual cluster description
      - Type: NVARCHAR(10000)
      - Default: null
  - **Returns**
    - BOOLEAN true if update was successful, false otherwise

**Details**

This stored procedure allows updates of <name> and/or <description> for a cluster with <clusterid> of K-means clustering model <model>.

**Examples**

```sql
CALL nza..KMEANS('model=iris_mdl, intable=nza..iris, outtable=iris_out, id=id, distance=euclidean, k=3, maxiter=5, randseed=12345');

CALL nza..SET_CLUSTERNAME('model=iris_mdl, clusterid=2, name=versicolor, description= Cluster with mostly versicolor');

CALL nza..DROP_MODEL('model=iris_mdl');
CALL nza..DROP_TABLE('iris_out');
```

```
KMEANS
-------
3
(1 row)
```
SET_CLUSTERNAMESPACE

---------------
t
(1 row)

DROP_MODEL

---------------
t
(1 row)

DROP_TABLE

---------------
t
(1 row)

Related Functions
► category Analytics - Clustering
► KMEANS

SET_COLUMN_PROPERTIES - Set columns role, type or weight in a column properties table

This stored procedure modifies the roles, types or weights of the columns in a column properties table

Usage

The SET_COLUMN_PROPERTIES stored procedure has the following syntax:

► SET_COLUMN_PROPERTIES((NVARCHAR(ANY) paramString)
  ▲ Parameters
  ► paramString
    comma-separated list of <parameter>=<value> entries with parameters below
    Type: NVARCHAR(ANY)
  ► intable
    the input table
    Type: NVARCHAR(256)
  ► colPropertiesTable
the name of the column properties table to modify
Type: NVARCHAR(ANY)

► **incolumn**
the input table columns with special properties, separated by a semi-colon (;).
Each column is followed by one or several of the following properties:
- its type: ':nom' (for nominal), ':cont' (for continuous). Per default, all numerical types are continuous, other types are nominal.
- its role: ':id', ':target', ':input', ':ignore', ':objweight', ':trials'.
- its weight: ':colweight(<wgt>)' where <wgt> is a numeric value of the weight.
If the parameter is undefined, all columns of the input table have default properties.
Type: NVARCHAR(ANY)
Default: <none>

► **coldeftype**
default type of the columns of the column properties table. Allowed values are 'nom' and 'cont'.
If the parameter is undefined, the column type is not modified per default.
Type: NVARCHAR(ANY)
Default: <none>

► **coldefrole**
default role of the columns of the column properties table. Allowed values are 'input' and 'ignore'.
If the parameter is undefined, all columns are considered 'input' columns per default.
Type: NVARCHAR(ANY)
Default: input

▲ **Returns**
BOOLEAN always true (otherwise an exception is raised)

**Details**
This stored procedure modifies the role, type or weight of the columns defined in the given column properties table. The column properties table is modified in place, no other output table is created. All columns referenced in parameter incolumn are modified accordingly. All other columns are modified to the type and role defined in parameters coldeftype or coldefrole, their weight is not modified.

**Examples**

```
CALL nza..COLUMN_PROPERTIES('intable=nza..weather, outtable=weatherDefSchema, coldeftype=nom ,incolumn=temperature;WINDY:nom:ignore');

CALL nza..SET_COLUMN_PROPERTIES('intable=nza..weather, colPropertiesTable=weatherDefSchema, coldeftype=nom ,incolumn=instance:cont:input:colweight(0.1); WINDY:nom:ignore');
```
SELECT * FROM weatherDefSchema ORDER BY idcol;

CALL nza..DROP_TABLE('weatherDefSchema');

COLUMN_PROPERTIES
-------------------
6
(1 row)

SET_COLUMN_PROPERTIES
-----------------------

(1 row)

<table>
<thead>
<tr>
<th>IDCOL</th>
<th>COLNAME</th>
<th>COLDATATYPE</th>
<th>COLTYPE</th>
<th>COLROLE</th>
<th>COLWEIGHT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>INSTANCE</td>
<td>INTEGER</td>
<td>cont</td>
<td>input</td>
<td>0.1</td>
</tr>
<tr>
<td>2</td>
<td>OUTLOOK</td>
<td>NATIONAL CHARACTER VARYING(10)</td>
<td>nom</td>
<td>input</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>TEMPERATURE</td>
<td>NATIONAL CHARACTER VARYING(10)</td>
<td>nom</td>
<td>input</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>HUMIDITY</td>
<td>NATIONAL CHARACTER VARYING(10)</td>
<td>nom</td>
<td>input</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>WINDY</td>
<td>NATIONAL CHARACTER VARYING(10)</td>
<td>nom</td>
<td>ignore</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>PLAY</td>
<td>NATIONAL CHARACTER VARYING(3)</td>
<td>nom</td>
<td>input</td>
<td>1</td>
</tr>
</tbody>
</table>

(6 rows)

DROP_TABLE

--------------------

(1 row)
**Related Functions**

- category Analytics - Column Properties
- COLUMN_PROPERTIES

---

**SKEWNESS_AGG - Skewness**

This function calculates the Skewness value of a single numeric variable, as the third central moment divided by the cube of the standard deviation.

**Usage**

The SKEWNESS_AGG aggregate has the following syntax:

```
SKEWNESS_AGG(DOUBLE X)
```

**Parameters**

- **X**
  - the input variable
  - Type: DOUBLE

**Returns**

DOUBLE the Skewness value of the input variable

**Details**

This function calculate the Skewness value of a single numeric variable. The skewness is a measure of the shift of the distribution around the mean value. If the skewness is negative, then the tail on the left side of the probability density function may be longer than that on the right side. the majority of the values would probably lie to the left of the mean value. If the skewness is positive, then the opposite will be true.

The Skewness is calculated as the third central moment divided by the cube of the standard deviation. Note that the normal distribution has a Skewness of 0, as its density function is symmetric around the mean. A k-th order central moment is the sum of the k-th powers of differences between the mean and the actual value divided by the number of cases. Because of the fact that we want to know the moments for a population but we are computing them from the given sample values, diverse corrections are taken into account. This leads to many formulas for assessing the Skewness. In this implementation we followed a "midway" policy dividing the estimated central moments by the number of cases minus 1. This policy is a generally established practice for variance but not for Skewness. Therefore the implementations match each other for large number of cases only.

If a value is missing, the whole row is ignored.

**Examples**

```
SELECT nza..SKEWNESS_AGG(petallength) FROM nza..iris;
```
SPEARMAN_CORR - Spearman Rank Correlation

This procedure calculates the Spearman rank correlation on two ordered input columns to evaluate the dependence between these columns.

Usage

The SPEARMAN_CORR stored procedure has the following syntax:

- **SPEARMAN_CORR**(NVARCHAR(ANY) paramString)

  ▲ Parameters
  
  ▶ **paramString**
  
  comma-separated list of <parameter>=<value> entries with parameters below
  
  Type: NVARCHAR(ANY)
  
  ▶ **intable**
  
  the input table
  
  Type: NVARCHAR(256)
  
  ▶ **inColumn**
  
  the two ordinal input table columns, separated by a semicolon (;). Optionally, a third numeric column can be specified for weights followed by :obj:weight.
  
  Type: NVARCHAR(ANY)
  
  ▶ **by**
  
  the input table column which splits the data into groups for which the operation is to be performed. If specified, an output table must be specified too.
  
  Type: NVARCHAR(128)
  
  Default: <none>
  
  ▶ **outTable**
  
  the output table to write the Spearman rank correlation into. This parameter is re-
quired if parameter by is specified. Otherwise the parameter is ignored.

Type: NVARCHAR(ANY)
Default: <none>

▲ Returns
DOUBLE The Spearman rank correlation or the number of rows in the output table

Details
This stored procedure calculates the Spearman rank correlation between two columns whose values are ordered in their respective domain, either in the whole input table or within the groups defined in the column specified by parameter <by>. If both columns are of type double, or int, or date, or time, the order is obviously guaranteed. In case of character columns, it is assumed that the order of their values is lexicographic. The Spearman rank correlation is a non-parametric measure of dependence between two variables. It takes a value between -1 (inversely correlated) and 1 (correlated), 0 means that the two columns are independent.

The output table is created with following columns: rho (correlation), n (number of records). If the parameter by is specified, an additional column <by> is added to indicate for which group the MWW test has been calculated. Note that only records where both columns are not null are considered.

Examples

```
CALL nza..SPEARMAN_CORR('intable=nza..adult, incolumn=workclass;education');

SPEARMAN_CORR
----------------------
0.0059993907642534
(1 row)

create view xadult as select * from nza..adult;
CALL nza..SPEARMAN_CORR('intable=xadult, incolumn=workclass;education');
DROP View xadult;

SPEARMAN_CORR
----------------------
0.0059993907642534
(1 row)

CALL nza..SPEARMAN_CORR('intable=nza..adult, incolumn=workclass;education;hours_per_week:objweight');
```
create view xadult as select * from nza..adult;
CALL nza..SPEARMAN_CORR('intable=xadult, incolumn=workclass;education;hours_per_week;objweight');
drop view  xadult;

CALL nza..SPEARMAN_CORR('intable=nza..WineQuality, incolumn=RESIDUALSUGAR;CHLORIDES');

CALL nza..SPEARMAN_CORR('intable=nza..WineQuality, incolumn=RESIDUALSUGAR;CHLORIDES, by=quality, outtable=spc_08');
SELECT * FROM spc_08 ORDER BY quality;
CALL nza..DROP_TABLE('spc_08');
<table>
<thead>
<tr>
<th>QUALITY</th>
<th>RHO</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0.3393673843897</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>0.30520232940759</td>
<td>163</td>
</tr>
<tr>
<td>5</td>
<td>0.16059812368123</td>
<td>1457</td>
</tr>
<tr>
<td>6</td>
<td>0.19412664982365</td>
<td>2198</td>
</tr>
<tr>
<td>7</td>
<td>0.17084455078142</td>
<td>880</td>
</tr>
<tr>
<td>8</td>
<td>0.20282460473778</td>
<td>175</td>
</tr>
<tr>
<td>9</td>
<td>0.4</td>
<td>5</td>
</tr>
</tbody>
</table>

(7 rows)

```
DROP_TABLE

-------------
t
(1 row)
```

create view xadult as select * from nza..WineQuality;
CALL nza..SPEARMAN_CORR('intable=xadult,
incolumn=RESIDUALSUGAR;CHLORIDES, by=quality, outtable=spc_08');
SELECT * FROM spc_08 ORDER BY quality;
CALL nza..DROP_TABLE('spc_08');
drop view  xadult;

```
SPEARMAN_CORR

-------------

7

(1 row)
```

<table>
<thead>
<tr>
<th>QUALITY</th>
<th>RHO</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0.3393673843897</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>0.30520232940759</td>
<td>163</td>
</tr>
<tr>
<td>5</td>
<td>0.16059812368123</td>
<td>1457</td>
</tr>
<tr>
<td>6</td>
<td>0.19412664982365</td>
<td>2198</td>
</tr>
</tbody>
</table>
Related Functions

- category Analytics - Statistics
- CORR
- SPEARMAN_CORR_S

**SPEARMAN_CORR_S - Spearman Rank Correlation as string**

This procedure calculates the Spearman rank correlation on two ordered input columns to evaluate the dependence between these columns.

**Usage**

The SPEARMAN_CORR_S stored procedure has the following syntax:

```plaintext
SPEARMAN_CORR_S(NVARCHAR(ANY) paramString)
```

- **paramString**
  comma-separated list of `<parameter>=<value>` entries with parameters below.
  Type: NVARCHAR(ANY)

- **itable**
  the input table
  Type: NVARCHAR(256)

- **incolumn**
  the two ordinal input table columns, separated by a semicolon (;). Optionally, a third numeric column can be specified for weights followed by :objweight.
  Type: NVARCHAR(ANY)
by
the input table column which splits the data into groups for which the operation is to be performed. If specified, an output table must be specified too.
Type: NVARCHAR(128)
Default: <none>

outtable
the output table to write the Spearman rank correlation into. This parameter is required if parameter by is specified. Otherwise the parameter is ignored.
Type: NVARCHAR(ANY)
Default: <none>

Returns
NVARCHAR(300) The Spearman rank variables or the number of rows in the output table The Spearman rank variables can contain following results: rho, n, df, tstat, pp. See the output table column description for more information on them.

Details
This stored procedure calculates the Spearman rank correlation between two columns whose values are ordered in their respective domain, either in the whole input table or within the groups defined in the column specified by parameter <by>. If both columns are of type double, or int, or date, or time, the order is obviously guaranteed. In case of character columns, it is assumed that the order of their values is lexicographic. The Spearman rank correlation is a non-parametric measure of dependence between two variables. It takes a value between -1 (inversely correlated) and 1 (correlated), 0 means that the two columns are independent.

The output table is created with following columns: rho (correlation), n (number of records), df (degree of freedom, tstat (t-Student statistic), pp. If the parameter by is specified, an additional column <by> is added to indicate for which group the MWW test has been calculated. Note that only records where both columns are not null are considered.

Examples
CALL nza.SPEARMAN_CORR_S('intable=nza.WineQuality, incolumn=RESIDUALSUGAR;CHLORIDES');
SPEARMAN_CORR_S
---------------------------------------------------------------
-- pp= 1, tStat= 16.373215814186, df= 4896, rho= 0.22784390383091
(1 row)

CALL nza.SPEARMAN_CORR_S('intable=nza.adult, incolumn=age;capital_gain;hours_per_week:objweight');
CALL nza..SPEARMAN_CORR_S('intable=nza..WineQuality,
incolumn=RESIDUALSUGAR;CHLORIDES,
by=quality,outtable=spc_08');

SELECT * FROM spc_08 ORDER BY quality;

CALL nza..DROP_TABLE('spc_08');
SPLIT_DATA - Split a table into training and testing sets

This stored procedure randomly splits the input data into two separated subsets.

Usage

The SPLIT_DATA stored procedure has the following syntax:

```sql
SPLIT_DATA(NVARCHAR(ANY) paramString)
```

- **paramString**
  comma-separated list of `<parameter>=<value>` entries with parameters below
  
  Type: NVARCHAR(ANY)

- **intable**
  the input table
  
  Type: NVARCHAR(256)

- **traintable**
  the output table that will contain the given fraction of the input records
  
  Type: NVARCHAR(256)

- **testtable**
  the output table that will contain the rest (1-fraction) of the input records
  
  Type: NVARCHAR(256)

- **id**
  the input table column identifying a unique instance id
  
  Type: NVARCHAR(128)

- **fraction**
the fraction of the data to split
Type: FLOAT
Default: 0.5
Min: 0.0
Max: 1.0

► seed
the seed of the random function
Type: FLOAT
Default: random()

▲ Returns
FLOAT number of records in traintable

Details
Two tables <traintable> and <testtable> are created. They have the same structure as <intable>. All records of <intable> are copied into the one or the other table with following ratio: <fraction> into <traintable>, 1-<fraction> into <testtable>.

Those two tables can be used to build and to test the prediction quality of a Classification or a Regression model.

Examples

```sql
CALL nza..SPLIT_DATA('intable=nza..iris, id=id, traintable=split_iris_train, testtable=split_iris_test, fraction=0.6, seed=1');

CALL nza..DROP_TABLE('split_iris_train');
CALL nza..DROP_TABLE('split_iris_test');
```

```
SPLIT_DATA
-------------
  90
(1 row)
DROP_TABLE
-------------
  t
(1 row)
DROP_TABLE
-------------
  t
```
STD_NORM - Standardization and Normalization

This stored procedure standardizes or normalizes columns of the input data

Usage

The STD_NORM stored procedure has the following syntax:

STD_NORM(NVARCHAR(ANY) paramString)

Parameters

- paramString
  comma-separated list of <parameter>=<value> entries with parameters below
  Type: NVARCHAR(ANY)

- intable
  the input table
  Type: NVARCHAR(256)

- incolumn
  the input table columns to consider, separated by a semi-colon (;).
  Each column name may be followed by :L to leave it unchanged, by :S to standardize its values,
  by :N to normalize its values or by :U to make it of unit length. Additionally, two columns may
  be indicated, separated by a slash (/), followed by :C to make the columns be a row unit vector
  or by :V to divide the column values by the length of the longest row vector.
  Type: NVARCHAR(ANY)

- id
  the input table column identifying a unique instance id
  Type: NVARCHAR(128)

- outtable
  the output table with the modified data
  Type: NVARCHAR(256)

- by
  the input table column which splits the data into groups for which the operation is to be per-
  formed
  Type: NVARCHAR(128)
  Default: <none>
Returns
NVARCHAR(1000) the number of transformed columns in the output. Columns specified with suffix :L are not counted.

Details
This stored procedure standardizes or normalizes columns of the input table, and writes the transformed columns into the given output table. The parameter incolumn specifies the columns to be considered and which transformation should be applied:
- <column>:L means that the <column> is leaved as is. The output table contains the same column with the same values.
- <column>:S means that the <column> is standardized. The output table contains a column STD_<column> that contains the standardized input values. A standardized value S is equal to (x-mean)/stddev, where x is the input value, mean the mean value of the column and stddev the standard deviation of the column. The standardized values range from minus infinity to plus infinity.
- <column>:N means that the <column> is normalized. The output table contains a column NRM_<column> that contains the normalized input values. A normalized value N is equal to x/absmax, where x is the input value and absmax the maximum absolute value of the column. The normalized values range from -1 to 1.
- <column>:U means that the <column> is normalized to a unit vector. The output table contains a column NRU_<column> that contains the euclidean-normalized input values. An euclidean normalized value U is equal to x/sqrt(ssq), where x is the input value and ssq the sum of square values of the column. The euclidean normalized values range from -1 to 1.
- <col1>/<col2>:C means that [<col1> <col2>] are normalized to be a unit vector in the space. The output table contains a column NRC_<rank>_<col1> and a column NRC_<rank>_<col2>, with <rank> the rank of this transformation in the list of :C or :V transformations, beginning with 1. Both columns contain their own normalized input values. The normalized value U1 for <col1> is equal to x/sqrt(x^2+y^2), where x is the input value of <col1> and y the input value of <col2>. The normalized value U2 for <col2> is equal to y/sqrt(x^2+y^2), where x is the input value of <col1> and y the input value of <col2>. The normalized values range from -1 to 1.
- <col1>/<col2>:V means that [<col1> <col2>] are normalized according to the longest row vector in the data. The output table contains a column NRM_<rank>_<col1> and a column NRM_<rank>_<col2>, with <rank> the rank of this transformation in the list of :C or :V transformations, beginning with 1. Both columns contain their own normalized input values. The normalized value V1 for <col1> is equal to x/sqrt(maxssq), where x is the input value of <col1> and maxssq the maximum of the square sum of the values of <col1> and <col2>. The normalized value V2 for <col2> is equal to y/sqrt(maxssq), where y is the input value of <col2> and maxssq the maximum of the square sum of the values of <col1> and <col2>. The normalized values range from -1 to 1.

The output table always contains the <id> column, plus the output column implied by the definition of parameter incolumn. If the parameter by is specified, an additional column <by> is added to indicate for which group the transformation has been calculated. The naming convention of the output columns imposes the following limitations: One input column should not appear twice in parameter incolumn with the same suffix :L, :S, :N or :U.
Examples

CALL nza..STD_NORM('intable=nza..iris, incolumn=petallength:S;petallength:L;petallength:N;petallength:U;sepallength/petallength:C, id=id, outtable=result7');
SELECT * FROM result7 WHERE id <= 10 ORDER BY id;
CALL nza..DROP_TABLE('result7');

STD_NORM

---------
5

(1 row)

ID | STD_PETALLENGTH | PETALLENGTH | NRM_PETALLENGTH | NRU_PETALLENGTH | NRC_1_SEPALLENGTH | NRC_1_PETALLENGTH
---+------------------+-------------+------------------|-------------------+-------------------+-------------------
1 | -1.3367940202882 | 1.4 | 0.20289855072464 | 0.027546462351378 | 0.96432614791712 | 0.26471698178117
2 | -1.3367940202882 | 1.4 | 0.20289855072464 | 0.027546462351378 | 0.96152394764082 | 0.27472112789738
3 | -1.3934698549528 | 1.3 | 0.18840579710145 | 0.025578857897708 | 0.9638111202542 | 0.2665860545384
4 | -1.2801181856237 | 1.5 | 0.21739130434783 | 0.029514066805048 | 0.95072983948148 | 0.31002059983092
5 | -1.3367940202882 | 1.4 | 0.20289855072464 | 0.027546462351378 | 0.96296401971418 | 0.26962992551997
6 | -1.1667665162945 | 1.7 | 0.2463768115942 | 0.033449275712387 | 0.9538492507392 | 0.30028587523271
7 | -1.3367940202882 | 1.4 | 0.20289855072464 | 0.027546462351378 | 0.95667388042886 | 0.2911616157827
8 | -1.2801181856237 | 1.5 | 0.21739130434783 | 0.029514066805048 | 0.95782628522115 | 0.28734788556635
9 | -1.3367940202882 | 1.4 | 0.20289855072464 | 0.027546462351378 | 0.95292578001326 | 0.30320365727695
10 | -1.2801181856237 | 1.5 | 0.21739130434783 | 0.029514066805048 | 0.95619995550743 | 0.29271427209411

(10 rows)

DROP_TABLE
CALL nza..STD_NORM('intable=nza..iris, incolumn=petallength:S;petallength:L;petallength:N;petallength:U;sepallength/petallength:C, id=id,outtable=result7, by=class');

SELECT * FROM result7 WHERE id % 15 = 0 ORDER BY id, class;

CALL nza..DROP_TABLE('result7');

STD_NORM

------------
5
(1 row)

+--------------------+-------------------+-------------------+-------------------+-------------------+--------------------+
<table>
<thead>
<tr>
<th>CLASS</th>
<th>ID</th>
<th>STD_PETALLENGTH</th>
<th>PETALLENGTH</th>
<th>NRM_PETALLENGTH</th>
<th>NRU_PETALLENGTH</th>
</tr>
</thead>
<tbody>
<tr>
<td>setosa</td>
<td>15</td>
<td>-1.5215159696786</td>
<td>1.2</td>
<td>0.6315789473684</td>
<td>0.115129433468</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.9792604195071</td>
<td>0.115129433468</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.2026056040359</td>
<td>0.2026056040359</td>
</tr>
<tr>
<td>setosa</td>
<td>30</td>
<td>0.78381125710714</td>
<td>1.6</td>
<td>0.84210526315789</td>
<td>0.15350591129067</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.94664990605014</td>
<td>0.94664990605014</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.3222637978043</td>
<td>0.3222637978043</td>
</tr>
<tr>
<td>setosa</td>
<td>45</td>
<td>2.51280666771964</td>
<td>1.9</td>
<td>0.18228826965767</td>
<td>0.93708211483265</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.34910902317295</td>
<td>0.34910902317295</td>
</tr>
<tr>
<td>versicolor</td>
<td>60</td>
<td>-0.7661025543912</td>
<td>3.9</td>
<td>0.76470588235294</td>
<td>0.12870516118045</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.8</td>
<td>0.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.6</td>
<td>0.6</td>
</tr>
<tr>
<td>versicolor</td>
<td>75</td>
<td>0.085122506043467</td>
<td>4.3</td>
<td>0.8304959978259</td>
<td>0.8304959978259</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.5576895748539</td>
<td>0.5576895748539</td>
</tr>
</tbody>
</table>

594 00X6332-00 Rev. 1
versicolor | 90  | -0.55329628928254 | 4 | 0.7843137254902 | 0.13200529351841 | 0.80873608430319 | 0.58817169767505
virginica  | 105 | 0.4493610863602 | 5.8 | 0.84057971014493 | 0.14702827760322 | 0.74614130128354 | 0.66578762268377
virginica  | 120 | -1.0001908051243 | 5 | 0.72463768115942 | 0.12674851517519 | 0.76822127959738 | 0.64018439966448
virginica  | 135 | 0.08697311348907 | 5.6 | 0.81159420289855 | 0.14195833699621 | 0.7365265810312 | 0.67627129268483
virginica  | 150 | -0.81899681868875 | 5.1 | 0.73913043478261 | 0.1292834854787 | 0.7565361485851 | 0.653953565013193

(10 rows)

DROP_TABLE

-------------
t

(1 row)

Related Functions
► category Analytics - Data Transformation

SUMMARY1000 - Summary of up to 1000 columns

This stored procedure calculates summary statistics of up to 1000 input columns. For continuous columns, following statistics are gathered: missing value count, non missing value count, average, variance, standard deviation, skewness, (excess) kurtosis, minimum and maximum. For discrete columns, the number of missing and non missing values, the number of distinct values and the frequency of the most frequent value are gathered.

Usage

The SUMMARY1000 stored procedure has the following syntax:

► SUMMARY1000(NVARCHAR(ANY) paramString)
▲ Parameters
   ► paramString
comma-separated list of <parameter>=<value> entries with parameters below
Type: NVARCHAR(ANY)

► intable
the input table
Type: NVARCHAR(256)

► outtable
the output table to write the summary statistics into. Additionally, other tables will be created depending on the type of the columns to be analyzed:
- <outtable>_num - for numeric columns (boolean, int4, int8, float, double, numeric)
- <outtable>_char - for character columns (char, nchar, varchar, nvarchar)
- <outtable>_date - for date columns
- <outtable>_time - for time columns (time, timetz)
- <outtable>_timestamp - for timestamp columns
- <outtable>_interval - for interval columns
The <outtable> itself contains all the data from those tables put together.
Type: NVARCHAR(256)

► incolumn
the input table columns separated by a semicolon (;). If this parameter is not specified, all input table columns are analyzed.
Type: NVARCHAR(ANY)
Default: <none>

► by
the input table column which splits the data into groups for which the operation is to be performed
Type: NVARCHAR(128)
Default: <none>

► talk
flag indicating whether additional information on the progress of the analysis must be displayed
Type: NVARCHAR(ANY)
Default: no

▲ Returns
NVARCHAR(ANY) an informative string

Details
This stored procedure gathers statistics about the input table columns. No more than 1000 columns are accepted. Null values are not part of the statistics computation (per column), but are
More than one output table is created. For each type of column encountered, one table is created: `<outtable>_num` (numeric), `<outtable>_char` (character), `<outtable>_date`, `<outtable>_time`, `<outtable>_timestamp`, `<outtable>_interval`. See the corresponding stored procedures `SUMMARY100.<type>` for a detailed description of these tables.

An output table `<outtable>` is created too, concatenating all the data of the type-specific output tables. The output table contains following columns: `columnname`, `distinctvalues`, `mostfrequentvalue`, `mostfrequentcases`, `average`, `variance`, `stddev`, `skewness`, `kurtosis`, `minimum`, `maximum`, `nonmissingcases`, `missing`. If the parameter `by` is specified, an additional column `grouped_on` is added to indicate for which group the statistics have been gathered. A detailed description of the statistics computed here can be found in the documentation of the stored procedure `MOMENTS`. When applicable, statistics are gathered for each input columns. Some statistics are type-specific, some statistics depends on the number of values (e.g. at least 2 values for calculating the average). If statistics cannot be gathered, the output table contains a NULL value.

Since more than one output table is actually generated, the stored procedure `DROP_SUMMARY1000` can be used to drop all of them.

**Examples**

```sql
CALL nza..SUMMARY1000('intable=nza..winequality,
outtable=wq_mo_05, talk=yes');

SELECT * FROM wq_mo_05 ORDER BY columnname;

CALL nza..DROP_SUMMARY1000('intable=wq_mo_05');

SUMMARY1000
-------------
Done
(1 row)

| COLUMNNAME | DISTINCTVALUES | MOSTFREQUENTVALUE | MOSTFREQUENTCASES | AVERAGE | VARIANCE | STDDEV | SKEWNESS | KURTOSIS | MINIMUM | MAXIMUM | NONMISSINGCASES | MISSING |
|------------|----------------|-------------------|-------------------+---------|---------|--------|---------|----------|----------|---------|----------|-----------------|---------|
| ALCOHOL    |                |                   |                   |         |         |        | -0.6994 | 8        |          | 14.2     | 4898     | 0      |
| CHLORIDES  |                |                   |                   |         |         |        | 0.346   |          | 0.009    | 0.346    | 0       | 0      |
| CITRIC_ACID|                |                   |                   |         |         |        |         | 5.021279 | 37.516765 | 0.009    | 0.346    | 0       | 0      |
```

00X6332-00 Rev. 1
| DENSITY              | 0.9940273764802 |
| FIXED_ACIDITY        | 6.8547876684361 |
| FREE_SULFUR_DIOXIDE  | 35.308084932626 |
| ID                   | 2449.5         |
| PH                   | 3.1882666394447 |
| QUALITY              | 5.8779093507554 |
| RESIDUALSUGAR        | 6.3914148632095 |
| SULPHATES            | 0.48984687627603 |
### TOTAL SULFUR DIOXIDE

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>138.36065741119</td>
<td>1806.0854908481</td>
<td>42.498064554143</td>
<td>0.39055030313114</td>
<td>0.56931602037303</td>
<td>9</td>
<td>440</td>
<td>4898</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

### VOLATILE ACIDITY

| 0.27824111882401 | 0.010159540992173 | 0.10079454842487 | 1.5763355750639 | 5.0835541889412 | 0.08 | 1.1 | 4898 | 0 |

(13 rows)

```sql
DROP_SUMMARY1000
```

---

```sql
CALL nza..SUMMARY1000('intable=nza..winequality, incolumn=FIXED_ACIDITY;VOLATILE_ACIDITY;CITRIC_ACID;RESIDUALSUGAR;CHLORIDES;FREE_SULFUR_DIOXIDE;TOTAL_SULFUR_DIOXIDE;DENSITY;PH;SULPHATES;ALCOHOL;QUALITY, outtable=wq_mo_05');
SELECT * FROM wq_mo_05 ORDER BY columnname;
CALL nza..DROP_SUMMARY1000('intable=wq_mo_05');
```

### SUMMARY1000

---

```sql
SUMMARY1000
```

---

```sql
COLUMNNAME | DISTINCTVALUES | MOSTFREQUENTVALUE | MOSTFREQUENTCASES |      AVERAGE      |      VARIANCE       | STDDEV       |     SKEWNESS     |     KURTOSIS     | MINIMUM |
---------+---------+-----------------+-------------------+-------------------+------------------+--------------+-----------------+-----------------+--------
ALCOHOL   |        |                 |                   | 10.514267047775   | 1.5144269817875 | 1.2306205677574 | 0.48714299689486 | -0.6994070978158 | 8 | 14.2 |
4898 | 0 |
CHLORIDES |        |                 |                   | 0.045772356063699 | 0.00047733370982461 | 0.021847968093729 | 5.0212795064905 | 37.516765256893 | 0.009 | 0.346 |
4898 | 0 |
CITRIC_ACID |        |                 |                   | 0.00047733370982461 | 0.021847968093729 | 5.0212795064905 | 37.516765256893 | 0.009 | 0.346 |
4898 | 0 |
| DENSITY              |                |
|                     | 0.9940273764802 |
|                     | 0.002990906916669 |
|                     | 0.9779453864398 |
|                     | 0.98711 |
|                     | 1.03898 |
|                     | 4898 |

| FIXED_ACIDITY        |                |
|                     | 6.8547876684361 |
|                     | 0.84386822768751 |
|                     | 0.64748697830986 |
|                     | 2.1676816699878 |
|                     | 4898 |

| FREE_SULFUR_DIOXIDE  |                |
|                     | 35.308084932626 |
|                     | 17.007137325233 |
|                     | 1.4061705044746 |
|                     | 11.450465023891 |
|                     | 4898 |

| PH                   |                |
|                     | 3.1882666394447 |
|                     | 0.022801181084074 |
|                     | 0.52828803485671 |
|                     | 2.72 |
|                     | 3.82 |
|                     | 4898 |

| QUALITY              |                |
|                     | 5.8779093507554 |
|                     | 0.78435568547105 |
|                     | 0.88563857496783 |
|                     | 0.15573278138988 |
|                     | 3 |
|                     | 9 |
|                     | 4898 |

| RESIDUALSUGAR        |                |
|                     | 6.3914148632095 |
|                     | 5.0720577840149 |
|                     | 3.4637343590186 |
|                     | 0.6 |
|                     | 65.8 |
|                     | 4898 |

| SULPHATES            |                |
|                     | 0.48984687627603 |
|                     | 0.11412583394884 |
|                     | 0.97679466587694 |
|                     | 1.5871445688284 |
|                     | 0.22 |
|                     | 1.08 |
|                     | 4898 |

| TOTAL_SULFUR_DIOXIDE |                |
|                     | 138.36065741119 |
|                     | 1806.0854908481 |
|                     | 42.498064554143 |
|                     | 0.39055030313114 |
|                     | 0.56931602037303 |
|                     | 9 |
|                     | 440 |
|                     | 4898 |

| VOLATILE_ACIDITY     |                |
|                     | 600 |

| 0.33419150673744 | 0.014645793009328 |
| 0.12101980420298 | 1.2813969516998 |
| 0 | 1.66 |
| 0 | 4898 |

| 8.9455241841782e-06 | 0.002990906916669 |
| 0.97737385720246 | 9.7799453864398 |
| 0.98711 | 1.03898 |
| 4898 | 0 |

| 0.71211358570046 | 0.84386822768751 |
| 0.64748697830986 | 2.1676816699878 |
| 3.8 | 14.2 |
| 4898 | 0 |

| 0.78435568547105 | 0.88563857496783 |
| 0.15573278138988 | 0.21442370898296 |
| 3 | 9 |
| 4898 | 0 |

| 25.725770164386 | 5.0720577840149 |
| 1.0766539468053 | 3.4637343590186 |
| 0.6 | 65.8 |
| 4898 | 0 |

| 0.013024705974517 | 0.11412583394884 |
| 0.97679466587694 | 1.5871445688284 |
| 0.22 | 1.08 |
| 4898 | 0 |

| 1806.0854908481 | 42.498064554143 |
| 0.39055030313114 | 0.56931602037303 |
| 9 | 440 |
| 4898 | 0 |
CALL nza..SUMMARY1000('intable=nza..iris, incolumn=sepalwidth;sepallength;petalwidth;petallength, outtable=Result72');

SELECT * FROM Result72 ORDER BY columnname;

CALL nza..DROP_SUMMARY1000('intable=Result72');

<table>
<thead>
<tr>
<th>COLUMNNAME</th>
<th>DISTINCTVALUES</th>
<th>MOSTFREQUENTVALUE</th>
<th>MOSTFREQUENTCASES</th>
<th>AVERAGE</th>
<th>VARIANCE</th>
<th>STDDEV</th>
<th>SKEWNESS</th>
<th>KURTOSIS</th>
<th>MINIMUM</th>
<th>MAXIMUM</th>
<th>NONMISSINGCASES</th>
<th>MISSING</th>
</tr>
</thead>
<tbody>
<tr>
<td>PETALLENGTH</td>
<td>3.7586666666667</td>
<td>3.1131794183445</td>
<td>1.7644204199523</td>
<td>-0.27080472910747</td>
<td>-1.4060569067921</td>
<td>1</td>
<td>6.9</td>
<td>-1.3463440065037</td>
<td>0.1</td>
<td>2.5</td>
<td>150</td>
<td>0</td>
</tr>
<tr>
<td>PETALWIDTH</td>
<td>1.1986666666667</td>
<td>0.58241431767338</td>
<td>0.76316074170084</td>
<td>-0.10359660798221</td>
<td>-1.3463440065037</td>
<td>0.1</td>
<td>2.5</td>
<td>0</td>
<td>150</td>
<td>0</td>
<td>0</td>
<td>150</td>
</tr>
<tr>
<td>SEPALLENGTH</td>
<td>5.843333333333333</td>
<td>0.68569351230425</td>
<td>0.82806612797787</td>
<td>0.31071214388165</td>
<td>-0.58974416259619</td>
<td>4.3</td>
<td>7.9</td>
<td>0</td>
<td>150</td>
<td>0</td>
<td>0</td>
<td>150</td>
</tr>
<tr>
<td>SEPALWIDTH</td>
<td>3.054</td>
<td>0.18800402684564</td>
<td>0.43359431136217</td>
<td>0.32959862667676</td>
<td>0.21983367738595</td>
<td>2</td>
<td>4.4</td>
<td>0</td>
<td>150</td>
<td>0</td>
<td>0</td>
<td>150</td>
</tr>
</tbody>
</table>
CALL nza..SUMMARY1000('intable=nza..iris, outtable=Result72');

SELECT * FROM Result72 ORDER BY columnname;

CALL nza..DROP_SUMMARY1000('intable=result72');

SUMMARY1000

-------------

Done

(1 row)

<table>
<thead>
<tr>
<th>COLUMNNAME</th>
<th>DISTINCTVALUES</th>
<th>MOSTFREQUENTVALUE</th>
<th>MOSTFREQUENTCASES</th>
<th>AVERAGE</th>
<th>VARIANCE</th>
<th>STDDEV</th>
<th>SKEWNESS</th>
<th>KURTOSIS</th>
<th>MINIMUM</th>
<th>MAXIMUM</th>
<th>NONMISSINGCASES</th>
<th>MISSING</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASS</td>
<td>3</td>
<td>virginica</td>
<td>150</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>75.5</td>
<td>1887.5</td>
<td>43.445367992457</td>
<td>-1.4196448216781e-15</td>
<td>-1.2121059602649</td>
<td>1</td>
<td>150</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PETALLENGTH</td>
<td>3.7586666666667</td>
<td>3.1131794183445</td>
<td>1.7644204199523</td>
<td>-0.27080472910747</td>
<td>-1.4060569067921</td>
<td>1</td>
<td>6.9</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PETALWIDTH</td>
<td>1.1986666666667</td>
<td>0.58241431767338</td>
<td>0.76316074170084</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CALL nza..SUMMARY1000('intable=nza..iris, by=class, incolumn=sepalwidth;sepallength;petalwidth;petallength, outtable=Result72');
SELECT * FROM Result72 ORDER BY columnname, grouped_on;
CALL nza..DROP_SUMMARY1000('intable=Result72');
<table>
<thead>
<tr>
<th>SETOSA</th>
<th>PETALWIDTH</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>0.6</td>
<td>50</td>
</tr>
<tr>
<td>1</td>
<td>1.8</td>
<td>50</td>
</tr>
<tr>
<td>0.244</td>
<td>0.01149387755102</td>
<td></td>
</tr>
<tr>
<td>0.10720950308168</td>
<td>1.149353251038</td>
<td>1.2105459872159</td>
</tr>
<tr>
<td>0.55189469566398</td>
<td>0.52746680546659</td>
<td>-0.31134291510743</td>
</tr>
<tr>
<td>4.5</td>
<td>6.9</td>
<td>50</td>
</tr>
<tr>
<td>0.30458775510204</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VERSICOLOR</th>
<th>PETALWIDTH</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.326</td>
<td>0.039106122448979</td>
<td></td>
</tr>
<tr>
<td>0.19775268000454</td>
<td>-0.029932414130041</td>
<td>-0.53807593528485</td>
</tr>
<tr>
<td>1</td>
<td>1.8</td>
<td>50</td>
</tr>
<tr>
<td>2.026</td>
<td>0.075432653061223</td>
<td></td>
</tr>
<tr>
<td>0.27465005563667</td>
<td>-0.12429785366125</td>
<td>-0.70812104544654</td>
</tr>
<tr>
<td>1.4</td>
<td>2.5</td>
<td>50</td>
</tr>
<tr>
<td>0.01149387755102</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VIRGINICA</th>
<th>PETALWIDTH</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2.026</td>
<td>0.075432653061223</td>
<td></td>
</tr>
<tr>
<td>0.27465005563667</td>
<td>-0.12429785366125</td>
<td>-0.70812104544654</td>
</tr>
<tr>
<td>1.4</td>
<td>2.5</td>
<td>50</td>
</tr>
<tr>
<td>2.026</td>
<td>0.075432653061223</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VERSICOLOR</th>
<th>SEPALWIDTH</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2.77</td>
<td>0.098469387755103</td>
<td></td>
</tr>
<tr>
<td>0.31379832337841</td>
<td>-0.34833105003606</td>
<td>-0.49930646192858</td>
</tr>
<tr>
<td>2</td>
<td>3.4</td>
<td>50</td>
</tr>
<tr>
<td>2.77</td>
<td>0.098469387755103</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VERSICOLOR</th>
<th>SEPALLENGTH</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5.936</td>
<td>0.26643265306123</td>
<td></td>
</tr>
<tr>
<td>0.51617114706387</td>
<td>0.10116251260099</td>
<td>-0.64685079391119</td>
</tr>
<tr>
<td>4.9</td>
<td>7</td>
<td>50</td>
</tr>
<tr>
<td>5.006</td>
<td>0.12424897959184</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VERSICOLOR</th>
<th>SEPALLENGTH</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5.936</td>
<td>0.26643265306123</td>
<td></td>
</tr>
<tr>
<td>0.51617114706387</td>
<td>0.10116251260099</td>
<td>-0.64685079391119</td>
</tr>
<tr>
<td>4.9</td>
<td>7</td>
<td>50</td>
</tr>
<tr>
<td>5.936</td>
<td>0.26643265306123</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VERSICOLOR</th>
<th>SEPALLENGTH</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3.418</td>
<td>0.14517959183674</td>
<td></td>
</tr>
<tr>
<td>0.3810243979547</td>
<td>0.10277069798361</td>
<td>0.61143137974919</td>
</tr>
<tr>
<td>2.3</td>
<td>4.4</td>
<td>50</td>
</tr>
<tr>
<td>3.418</td>
<td>0.14517959183674</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VERSICOLOR</th>
<th>SEPALLENGTH</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2.77</td>
<td>0.098469387755103</td>
<td></td>
</tr>
<tr>
<td>0.31379832337841</td>
<td>-0.34833105003606</td>
<td>-0.49930646192858</td>
</tr>
<tr>
<td>2</td>
<td>3.4</td>
<td>50</td>
</tr>
<tr>
<td>2.77</td>
<td>0.098469387755103</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VERSICOLOR</th>
<th>SEPALLENGTH</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3.418</td>
<td>0.14517959183674</td>
<td></td>
</tr>
<tr>
<td>0.3810243979547</td>
<td>0.10277069798361</td>
<td>0.61143137974919</td>
</tr>
<tr>
<td>2.3</td>
<td>4.4</td>
<td>50</td>
</tr>
<tr>
<td>2.77</td>
<td>0.098469387755103</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VERSICOLOR</th>
<th>SEPALLENGTH</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2.974</td>
<td>0.1040048163266</td>
<td></td>
</tr>
<tr>
<td>0.32249663817264</td>
<td>0.35131111148685</td>
<td>0.44937061679896</td>
</tr>
<tr>
<td>2.2</td>
<td>3.8</td>
<td>50</td>
</tr>
<tr>
<td>2.974</td>
<td>0.1040048163266</td>
<td></td>
</tr>
</tbody>
</table>

(12 rows)

DROP_SUMMARY1000
CREATE TABLE irisnull AS SELECT * FROM nza..iris;

INSERT INTO irisnull(id,petallength) SELECT id+300, petallength FROM nza..iris;

INSERT INTO irisnull(id,sepallength,class) SELECT id+600, sepallength, class FROM nza..iris;

CALL nza..SUMMARY1000('intable=irisnull, incolumn=SEPALLENGTH;SEPALWIDTH;PETALLENGTH;SEPALWIDTH, outtable=Result7');

SELECT * FROM result7 ORDER BY columnname;

CALL nza..DROP_SUMMARY1000('intable=Result7');

CALL nza..DROP_TABLE('irisnull');

SUMMARY1000

Done

(1 row)

<table>
<thead>
<tr>
<th>COLUMNNAME</th>
<th>DISTINCTVALUES</th>
<th>MOSTFREQUENTVALUE</th>
<th>MOSTFREQUENTCASES</th>
<th>AVERAGE</th>
<th>VARIANCE</th>
<th>STDDEV</th>
<th>SKEWNESS</th>
<th>KURTOSIS</th>
<th>MINIMUM</th>
<th>MAXIMUM</th>
<th>NONMISSINGCASES</th>
<th>MISSING</th>
</tr>
</thead>
<tbody>
<tr>
<td>PETALLENGTH</td>
<td>3.7586666666667</td>
<td>3.1027674470457</td>
<td>1.7614674129957</td>
<td>-0.27125871891336</td>
<td>-0.58165605575527</td>
<td>300</td>
<td>150</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEPALLENGTH</td>
<td>5.8433333333333</td>
<td>0.68340022296544</td>
<td>0.8268024227354</td>
<td>0.31123303635777</td>
<td>-0.58165605575527</td>
<td>300</td>
<td>150</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEPALWIDTH</td>
<td>3.054</td>
<td>0.18800402684564</td>
<td>0.43359431136217</td>
<td>0.32959862667676</td>
<td>0.21983367738595</td>
<td>150</td>
<td>300</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
DROP_SUMMARY

(4 rows)

DROP_TABLE

CREATE TABLE withothers AS SELECT
        'true'::boolean a, 'true'::boolean b
        , '10:11:12'::time c, '11:09:12'::time d
        , '2010-11-12'::date e, '2011-09-12'::date f
        , '2010-11-12 11:10:15'::timestamp g, '2011-09-12 11:10:15'::timestamp h
        , '10:11:12+01'::timetz i, '11:09:12-03'::timetz j
        , '1 day 10:11:12'::interval k, '11:09:12'::interval l;

INSERT INTO withothers VALUES(
        'true'::boolean, 'true'::boolean
        , '10:11:12'::time, '11:09:12'::time
        , '2010-11-12'::date, '2011-09-12'::date
        , '2010-11-12 11:10:15'::timestamp, '2011-09-12 11:10:15'::timestamp
        , '10:11:12+01'::timetz, '11:09:12-03'::timetz
        , '1 day 10:11:12'::interval, '11:09:12'::interval);

INSERT INTO withothers VALUES(
        'true'::boolean, 'true'::boolean
        , '10:11:12'::time, '11:09:12'::time
        , '2010-11-12'::date, '2011-09-12'::date

,'2010-11-12 11:10:15'::timestamp , '2011-09-12
11:10:15'::timestamp
,'10:11:12+01'::timetz , '11:09:12-03'::timetz
,'1 day 10:11:12'::interval , '11:09:12'::interval);

INSERT INTO withothers VALUES(
'false'::boolean , 'false'::boolean
,'20:11:12'::time , '01:09:12'::time
,'2012-11-12'::date , '2011-02-12'::date
,'2013-11-12 11:10:15'::timestamp , '2011-09-12
12:10:15'::timestamp
,'10:11:12+03'::timetz , '12:09:12-03'::timetz
,'2 days 10:11:12'::interval , '5 days 11:09:12'::interval);

CALL nza..SUMMARY1000('intable=withothers, outtable=wd_05');

SELECT * FROM wd_05 ORDER BY columnname;

CALL nza..DROP_SUMMARY1000('intable=wd_05');

CALL nza..DROP_TABLE('withothers');

SUMMARY1000
------------

Done
(1 row)

<table>
<thead>
<tr>
<th>COLUMNNAME</th>
<th>DISTINCTVALUES</th>
<th>MOSTFREQUENTVALUE</th>
<th>MOSTFREQUENTCASES</th>
<th>AVERAGE</th>
<th>VARIANCE</th>
<th>STDDEV</th>
<th>SKEWNESS</th>
<th>KURTOSIS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>0.75</td>
<td>0.25</td>
<td>0.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>-1.25</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>0.75</td>
<td>0.25</td>
<td>0.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>-1.25</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>12:41:12</td>
<td>324000000</td>
<td>05:00:00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---------</td>
<td>--------------</td>
<td>--------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>-1.25</td>
<td>10:11:12</td>
<td>20:11:12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>08:39:12</td>
<td>324000000</td>
<td>05:00:00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-0.99999999999999</td>
<td>-1.25</td>
<td>01:09:12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>11:09:12</td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2011-05-14</td>
<td>133590.25</td>
<td>366 days</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.999999999999937</td>
<td>-1.2500000000061</td>
<td>2010-11-12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2012-11-12</td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2011-07-21</td>
<td>11236</td>
<td>106 days</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-1.0000000000256</td>
<td>-1.25</td>
<td>2011-02-12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2011-09-12</td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2011-08-13 11:10:15</td>
<td>2.24175734784e+15</td>
<td>548 days</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.00000000000001</td>
<td>-1.2500000000038</td>
<td>2010-11-12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>11:10:15</td>
<td>2013-11-12 11:10:15</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2011-09-12 11:25:15</td>
<td>3239988.0881722</td>
<td>00:30:00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>-3186575.5274879</td>
<td>2011-09-12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>11:10:15</td>
<td>2011-09-13 12:10:15</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>08:41:12</td>
<td>12960000</td>
<td>01:00:00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-1.00000000000006</td>
<td>-1.25000000000112</td>
<td>07:11:12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>09:11:12</td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>J</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>14:24:12</td>
<td>3239999.9999997</td>
<td>00:30:00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.00000000000103</td>
<td>-1.2499999990159</td>
<td>14:09:12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>15:09:12</td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 DAY 16:11:12</td>
<td>1866240000</td>
<td>12:00:00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>-1.2500000000003</td>
<td>1 DAYS 10:11:12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 DAYS 10:11:12</td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 DAY 17:09:12</td>
<td>46656000000</td>
<td>2 DAYS 12:00:00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>-1.2500000000003</td>
<td>1 DAYS 10:11:12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>11:09:12</td>
<td></td>
<td>5 DAYS 11:09:12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(12 rows)
SUMMARY1000CHAR - Summary of up to 1000 character columns

This stored procedure calculates summary statistics of up to 1000 character input columns. The number of missing and non-missing values, the number of distinct values and the frequency of the most frequent value are gathered.

Usage

The SUMMARY1000CHAR stored procedure has the following syntax:

```
SUMMARY1000CHAR(NVARCHAR(ANY) paramString)
```

- **paramString**
  - comma-separated list of `<parameter>=<value>` entries with parameters below
  - Type: NVARCHAR(ANY)

- **intable**
  - the input table
  - Type: NVARCHAR(256)

- **outtable**
  - the output table to write the summary statistics into
Type: NVARCHAR(256)

► incolumn
the input table columns of type char, nchar, varchar or nvarchar of any length sepa-
rated by a semicolon (;). If this parameter is not specified, all input table columns of
character type are analyzed.
Type: NVARCHAR(ANY)
Default: <none>

► by
the input table column which splits the data into groups for which the operation is to
be performed
Type: NVARCHAR(128)
Default: <none>

► talk
flag indicating whether additional information on the progress of the analysis must be
displayed
Type: NVARCHAR(ANY)
Default: no

▲ Returns
NVARCHAR(ANY) an informative string

Details
This stored procedure gathers statistics about the input table columns of type char, nchar, varchar
or nvarchar of any length. No more than 1000 columns are accepted. Null values are not part of
the statistics computation (per column), but are counted.

An output table <outtable> is created with following columns: colname, distinctvalues, mostfre-
quenvalue, mostfrequentcases, nonmissingcases, missing. If the parameter by is specified, an ad-
ditional column grouped_on is added to indicate for which group the statistics have been
gathered. A detailed description of the statistics computed here can be found in the documenta-
tion of the stored procedure MOMENTS. If statistics cannot be gathered, the output table contains
a NULL value.

Examples

CALL nza..SUMMARY1000CHAR('intable=nza..iris,
outtable=Result72');
SELECT * FROM Result72;
CALL nza..DROP_TABLE('Result72');
SUMMARY1000CHAR

Done
(1 row)

| COLNAME | DISTINCTVALUES | MOSTFREQUENTVALUE | MOSTFREQUENTCASES | NONMISSINGCASES | MISSING |
|---------|----------------+-------------------+-------------------+-----------------+---------|
| CLASS   | 3 | virginica | 150 | 0 |

(1 row)

DROP_TABLE
---------

CALL nza..SUMMARY1000CHAR('intable=nza..adult,
outtable=Result71');

SELECT * FROM Result71 ORDER BY colname;

CALL nza..DROP_TABLE('Result71');

SUMMARY1000CHAR
-----------------  

Done

(1 row)

| COLNAME             | DISTINCTVALUES | MOSTFREQUENTVALUE | MOSTFREQUENTCASES | NONMISSINGCASES | MISSING |
|---------------------+----------------+-------------------+-------------------+-----------------+---------|
| EDUCATION           | 16 | HS-grad | 32561 | 0 |
| INCOME              | 2 | small | 32561 | 0 |
| MARITAL_STATUS      | 7 | Married-civ-spouse | 32561 | 0 |
| OCCUPATION          | 14 | Prof-specialty | 32561 | 0 |
| RACE                | 5 | White | 32561 | 0 |
| RELATIONSHIP        | 6 | Husband | 32561 | 0 |
SEX |              2 | Male               |
21790 |           32561 |       0

WORKCLASS |              8 | Private            |
24532 |           32561 |       0

(8 rows)

DROP_TABLE

----------
t

CALL nza..SUMMARY1000CHAR('intable=nza..adult, incolumn=education;race, outtable=Result71');
SELECT * FROM Result71 ORDER BY colname;
CALL nza..DROP_TABLE('Result71');

SUMMARY1000CHAR

----------

Done

(1 row)

COLNAME | DISTINCTVALUES | MOSTFREQUENTVALUE | MOSTFREQUENTCASES | NONMISSINGCASES | MISSING
----------------+-----------------+-------------------+-------------------+-----------------+---------
EDUCATION |             16 | HS-grad           | 10501 | 32561 | 0
RACE      |              5 | White             | 27816 | 32561 | 0

(2 rows)

DROP_TABLE

----------
t

(1 row)

CALL nza..SUMMARY1000CHAR('intable=nza..adult, outtable=Result73, by=sex');
SELECT * FROM Result73 ORDER BY colname, grouped_on;
CALL nza..DROP_TABLE('Result73');

SUMMARY

-----------------
Done
(1 row)

<table>
<thead>
<tr>
<th>COLNAME</th>
<th>GROUPED_ON</th>
<th>DISTINCTVALUES</th>
<th>MOSTFREQUENTVALUE</th>
<th>MOSTFREQUENTCASES</th>
<th>NONMISSINGCASES</th>
<th>MISSING</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDUCATION</td>
<td>Female</td>
<td>16</td>
<td>HS-grad</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3390</td>
<td>10771</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EDUCATION</td>
<td>Male</td>
<td>16</td>
<td>HS-grad</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>7111</td>
<td>21790</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INCOME</td>
<td>Female</td>
<td>2</td>
<td>small</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>9592</td>
<td>10771</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INCOME</td>
<td>Male</td>
<td>2</td>
<td>small</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>15128</td>
<td>21790</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MARITAL_STATUS</td>
<td>Female</td>
<td>7</td>
<td>Never-married</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4767</td>
<td>10771</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MARITAL_STATUS</td>
<td>Male</td>
<td>7</td>
<td>Married-civ-spouse</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>13319</td>
<td>21790</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OCCUPATION</td>
<td>Female</td>
<td>13</td>
<td>Adm-clerical</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2537</td>
<td>10771</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OCCUPATION</td>
<td>Male</td>
<td>14</td>
<td>Craft-repair</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3877</td>
<td>21790</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RACE</td>
<td>Female</td>
<td>5</td>
<td>White</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>8642</td>
<td>10771</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RACE</td>
<td>Male</td>
<td>5</td>
<td>White</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>19174</td>
<td>21790</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RELATIONSHIP</td>
<td>Female</td>
<td>6</td>
<td>Not-in-family</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3875</td>
<td>10771</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RELATIONSHIP</td>
<td>Male</td>
<td>6</td>
<td>Husband</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>13192</td>
<td>21790</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEX</td>
<td>Female</td>
<td>1</td>
<td>Female</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>10771</td>
<td>10771</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEX</td>
<td>Male</td>
<td>1</td>
<td>Male</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
WORKCLASS        | Female     | 8 | Private  
|                | 8591       | 10771 |  
WORKCLASS        | Male       | 8 | Private  
|                | 15941      | 21790 |  

(16 rows)

DROP_TABLE

-------------

t
(1 row)

CALL nza..SUMMARY1000CHAR('intable=nza..adult, incolumn=occupation;relationship, outtable=Result73, by=sex');

SELECT * FROM Result73 ORDER BY colname, grouped_on;

CALL nza..DROP_TABLE('Result73');

SUMMARY1000CHAR

-------------

Done

(1 row)

<table>
<thead>
<tr>
<th>COLNAME</th>
<th>GROUPED_ON</th>
<th>DISTINCTVALUES</th>
<th>MOSTFREQUENTVALUE</th>
<th>MOSTFREQUENTCASES</th>
<th>NONMISSINGCASES</th>
<th>MISSING</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-------------</td>
<td>----------------</td>
<td>--------------------</td>
<td>--------------------</td>
<td>-----------------</td>
<td>---------</td>
</tr>
<tr>
<td>OCCUPATION</td>
<td>Female</td>
<td>13</td>
<td>Adm-clerical</td>
<td>2537</td>
<td>10771</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Male</td>
<td>14</td>
<td>Craft-repair</td>
<td>3877</td>
<td>21790</td>
<td></td>
</tr>
<tr>
<td>RELATIONSHIP</td>
<td>Female</td>
<td>6</td>
<td>Not-in-family</td>
<td>3875</td>
<td>10771</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Male</td>
<td>6</td>
<td>Husband</td>
<td>13192</td>
<td>21790</td>
<td></td>
</tr>
</tbody>
</table>

(4 rows)

DROP_TABLE
SUMMARY1000DATE - Summary of up to 1000 date columns

This stored procedure calculates summary statistics of up to 1000 input columns of type date. The missing value count, non missing value count, average, variance, standard deviation, skewness, (excess) kurtosis, minimum and maximum of the input columns is gathered.

Usage

The SUMMARY1000DATE stored procedure has the following syntax:

```
SUMMARY1000DATE(NVARCHAR(ANY) paramString)
```

- Parameters
  - `paramString` comma-separated list of `<parameter>=<value>` entries with parameters below
    - `paramString` Type: NVARCHAR(ANY)
  - `inTable` the input table
    - `inTable` Type: NVARCHAR(256)
  - `outTable` the output table to write the summary statistics into
    - `outTable` Type: NVARCHAR(256)
  - `inColumn` the input table columns of type date separated by a semicolon (;). If this parameter is not specified, all input table columns of type date are analyzed.
    - `inColumn` Type: NVARCHAR(ANY)
      - Default: <none>
  - `by` the input table column which splits the data into groups for which the operation is to be performed
    - `by` Type: NVARCHAR(128)
      - Default: <none>
**talk**
flag indicating whether additional information on the progress of the analysis must be displayed
Type: NVARCHAR(ANY)
Default: no

**Returns**
NVARCHAR(ANY) an informative string

**Details**
This stored procedure gathers statistics about the input table columns of type date. No more than 1000 columns are accepted. Null values are not part of the statistics computation (per column), but are counted.

An output table <outtable> is created with following columns: columnname, columnid, countt, average, variance, stddev, skewness, kurtosis. minimum, maximum, missing. If the parameter by is specified, an additional column grouped_on is added to indicate for which group the statistics have been gathered. A detailed description of the statistics computed here can be found in the documentation of the stored procedure MOMENTS. If statistics cannot be gathered, for example when there are less than 2 values for average, the output table contains a NULL value.

Note that dates are stored internally as integers representing the number of days since January 1st, 2000. The average, minimum and maximum is displayed as a date. The standard deviation is displayed as an interval (usually in days). The variance is expressed in "squared days".

**Examples**
```sql
CREATE TABLE withDates(a date, b date);
INSERT INTO withDates VALUES(date('2012-01-02'),
 date('2012-01-05'));
INSERT INTO withDates VALUES(date('2012-01-12'),
 date('2012-01-25'));
INSERT INTO withDates VALUES(date('2012-01-22'),
 date('2012-02-13'));
INSERT INTO withDates VALUES(date('2012-02-01'),
 date('2012-03-04'));
CALL nza..SUMMARY1000DATE('intable=withDates,
incolumn=a;b, outtable=wd_05');
SELECT * FROM wd_05 ORDER BY columnname;
CALL nza..DROP_TABLE('wd_05');
CALL nza..DROP_TABLE('withDates');
SUMMARY1000DATE
```

---

IBM Netezza In-Database Analytics Reference Guide

►

616

00X6332-00 Rev. 1
<table>
<thead>
<tr>
<th>COLUMNNAME</th>
<th>COLUMNID</th>
<th>COUNTT</th>
<th>AVERAGE</th>
<th>VARIANCE</th>
<th>STDDEV</th>
<th>SKEWNESS</th>
<th>KURTOSIS</th>
<th>MINIMUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>4</td>
<td>2012-01-17</td>
<td>166.66666666667</td>
<td>13 days</td>
<td>-1.4183288621365e-08</td>
<td>-1.769997</td>
<td>2012-01-02</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>4</td>
<td>2012-02-03</td>
<td>640.33333333333</td>
<td>25 days</td>
<td>1.8833927739782e-09</td>
<td>-1.7553965310125</td>
<td>2012-01-05</td>
</tr>
</tbody>
</table>

(2 rows)

DROP_TABLE

-------------
t

(1 row)

DROP_TABLE

-------------
t

(1 row)

Related Functions

► category Analytics - Statistics
► MOMENTS
► SUMMARY1000

**SUMMARY1000INTERVAL - Summary of up to 1000 interval columns**

This stored procedure calculates summary statistics of up to 1000 input columns of type interval. The missing value count, non missing value count, average, variance, standard deviation, skewness, (excess) kurtosis, minimum and maximum of the input columns is gathered.

**Usage**

The SUMMARY1000INTERVAL stored procedure has the following syntax:

► SUMMARY1000INTERVAL(NVARCHAR(ANY) paramString)
Parameters

- **paramString**
  comma-separated list of <parameter>=<value> entries with parameters below
  Type: NVARCHAR(ANY)

- **intable**
  the input table
  Type: NVARCHAR(256)

- **outtable**
  the output table to write the summary statistics into
  Type: NVARCHAR(256)

- **incolumn**
  the input table columns of type interval separated by a semicolon (;). If this parameter is not specified, all input table columns of type interval are analyzed.
  Type: NVARCHAR(ANY)
  Default: <none>

- **by**
  the input table column which splits the data into groups for which the operation is to be performed
  Type: NVARCHAR(128)
  Default: <none>

- **talk**
  flag indicating whether additional information on the progress of the analysis must be displayed
  Type: NVARCHAR(ANY)
  Default: no

Returns

NVARCHAR(ANY) an informative string

Details

This stored procedure gathers statistics about the input table columns of type interval. No more than 1000 columns are accepted. Null values are not part of the statistics computation (per column), but are counted.

An output table <outtable> is created with following columns: columnname, columnid, countt, average, variance, stddev, skewness, kurtosis, minimum, maximum, missing. If the parameter by is specified, an additional column grouped_on is added to indicate for which group the statistics have been gathered. A detailed description of the statistics computed here can be found in the documentation of the stored procedure MOMENTS. If statistics cannot be gathered, for example when there are less than 2 values for average, the output table contains a NULL value.

Note that intervals are stored internally as a structure comprising an integer representing the num-
Reference Documentation: Analytics

The structure is flattened to an integer by assigning a month an average number of days. The average value, standard deviation, minimum and maximum are displayed as an interval. The variance is expressed in "squared seconds".

Examples

```sql
CREATE TABLE withintervals(a interval, b interval);
INSERT INTO withintervals VALUES('1 day'::interval, '3 days'::interval);
INSERT INTO withintervals VALUES('11 days'::interval, '23 days'::interval);
INSERT INTO withintervals VALUES('21 days'::interval, '43 days'::interval);
INSERT INTO withintervals VALUES('31 days'::interval, '63 days'::interval);
CALL nza..SUMMARY1000INTERVAL('intable=withintervals, incolumn=a:b, outtable=wd_05');
SELECT * FROM wd_05 ORDER BY columnname;
CALL nza..DROP_TABLE('wd_05');
CALL nza..DROP_TABLE('withintervals');
```

```
SUMMARY1000INTERVAL

Done

(1 row)

<table>
<thead>
<tr>
<th>COLUMNNAME</th>
<th>COLUMNID</th>
<th>COUNT</th>
<th>AVERAGE</th>
<th>VARIANCE</th>
<th>STDDEV</th>
<th>SKEWNESS</th>
<th>KURTOSIS</th>
<th>MINIMUM</th>
<th>MAXIMUM</th>
<th>MISSING</th>
</tr>
</thead>
<tbody>
<tr>
<td>-----------</td>
<td>----------</td>
<td>-------</td>
<td>---------</td>
<td>---------------</td>
<td>--------</td>
<td>----------</td>
<td>----------</td>
<td>---------</td>
<td>---------</td>
<td>---------</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>4</td>
<td>16 days</td>
<td>1244160000000</td>
<td>12 days 21:50:19.203707</td>
<td>0</td>
<td>-1.77</td>
<td>1 days 31 days</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>4</td>
<td>33 days</td>
<td>4976640000000</td>
<td>25 days 19:40:38.407415</td>
<td>8.5071745671901e-16</td>
<td>-1.77</td>
<td>3 days 63 days</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
```

(2 rows)

```sql
DROP_TABLE
```

---
SUMMARY1000NUM - Summary of up to 1000 numeric columns

This stored procedure calculates summary statistics of up to 1000 input columns of type boolean, int1, int2, int4, int8, float, double or numeric. The missing value count, non missing value count, average, variance, standard deviation, skewness, (excess) kurtosis, minimum and maximum of the input columns is gathered.

Usage

The SUMMARY1000NUM stored procedure has the following syntax:

```sql
SUMMARY1000NUM(NVARCHAR(ANY) paramString)
```

- **paramString**
  - comma-separated list of <parameter>=<value> entries with parameters below
  - Type: NVARCHAR(ANY)

- **itable**
  - the input table
  - Type: NVARCHAR(256)

- **outtable**
  - the output table to write the summary statistics into
  - Type: NVARCHAR(256)

- **incolumn**
  - the input table columns of type boolean, int1, int2, int4, int8, float, double or numeric separated by a semicolon (;). If this parameter is not specified, all input table columns of those types are analyzed.
  - Type: NVARCHAR(ANY)
  - Default: <none>
by
the input table column which splits the data into groups for which the operation is to be performed
Type: NVARCHAR(128)
Default: <none>

► talk
flag indicating whether additional information on the progress of the analysis must be displayed
Type: NVARCHAR(ANY)
Default: no

▲ Returns
NVARCHAR(ANY) an informative string

Details
This stored procedure gathers statistics about the input table columns of type boolean, int1, int2, int4, int8, float, double or numeric. No more than 1000 columns are accepted. Null values are not part of the statistics computation (per column), but are counted.

An output table <outtable> is created with following columns: columnname, columnid, countt, average, variance, stddev, skewness, kurtosis. minimum, maximum, missing. If the parameter by is specified, an additional column grouped_on is added to indicate for which group the statistics have been gathered. A detailed description of the statistics computed here can be found in the documentation of the stored procedure MOMENTS. If statistics cannot be gathered, for example when there are less than 2 values for average, the output table contains a NULL value.

All of these numeric column types are handled as double, where boolean values are mapped to 0 (false) and 1 (true).

Examples

CALL nza..SUMMARY1000NUM('intable=nza..winequality, incolumn=FIXED_ACIDITY;VOLATILE_ACIDITY;CITRIC_ACID;RESIDUALSUGAR;CHLORIDES;FREE_SULFUR_DIOXIDE;TOTAL_SULFUR_DIOXIDE;DENSITY;PH;SULPHATES;ALCOHOL;QUALITY, outtable=wq_mo_05, talk=yes');
SELECT * FROM wq_mo_05 ORDER BY columnname;

CALL nza..DROP_TABLE('wq_mo_05');

SUMMARY1000NUM
----------------
Done
(1 row)

<table>
<thead>
<tr>
<th>COLUMNNAME</th>
<th>COLUMNID</th>
<th>COUNTT</th>
<th>AVERAGE</th>
<th>VARIANCE</th>
<th>STDEV</th>
<th>SKEWNESS</th>
<th>KURTOSIS</th>
<th>MINIMUM</th>
<th>MAXIMUM</th>
<th>MISSING</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------------------</td>
<td>----</td>
<td>-------</td>
<td>-------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALCOHOL</td>
<td>10</td>
<td>4898</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10.514267047775</td>
<td>1.5144269817875</td>
<td>1.2306205677574</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.48714299689436</td>
<td>-0.69940709781905</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14.2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHLORIDES</td>
<td>4</td>
<td>4898</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.045772356063699</td>
<td>0.00047733370982461</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.021847968093729</td>
<td>5.0212795064904</td>
<td>37.516765256893</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.009</td>
<td>0.346</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CITRIC_ACID</td>
<td>2</td>
<td>4898</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.33419150673744</td>
<td>0.014645793009328</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.12101980420298</td>
<td>6.1655025701792</td>
<td>8.9455241841782e-06</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.009</td>
<td>0.346</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DENSITY</td>
<td>7</td>
<td>4898</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.9940273764802</td>
<td>8.9455241841782e-06</td>
<td>0.002990906916669</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.977337385720246</td>
<td>9.779453864398</td>
<td>0.98711</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.03898</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIXED_ACIDITY</td>
<td>0</td>
<td>4898</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.8547876684361</td>
<td>8.9455241841782e-06</td>
<td>0.002990906916669</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.64748697830986</td>
<td>2.1676816698978</td>
<td>3.8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14.2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FREE_SULFUR_DIOXIDE</td>
<td>5</td>
<td>4898</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35.308084932626</td>
<td>289.24271999932</td>
<td>17.007137325233</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.4061705044746</td>
<td>11.450465023891</td>
<td>11.450465023891</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>289</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PH</td>
<td>8</td>
<td>4898</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.1882666394447</td>
<td>0.022801181084077</td>
<td>0.15100059961496</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.45759561971141</td>
<td>2.72</td>
<td>3.82</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QUALITY</td>
<td>11</td>
<td>4898</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.8779093507554</td>
<td>0.15573278138988</td>
<td>0.21442370898296</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.15573278138988</td>
<td>0.21442370898296</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RESIDUALSUGAR</td>
<td>3</td>
<td>4898</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.3914148632095</td>
<td>25.725770164386</td>
<td>5.0720577840149</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.0766539468053</td>
<td>3.4637343590186</td>
<td>0.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>65.8</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SULPHATES</td>
<td>9</td>
<td>4898</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.48984687627603</td>
<td>0.013024705974517</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.11412583394884</td>
<td>1.5871445688284</td>
<td>1.5871445688284</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.22</td>
<td>1.08</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOTAL_SULFUR_DIOXIDE</td>
<td>6</td>
<td>4898</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>138.36065741119</td>
<td>1806.0854908481</td>
<td>42.498064554143</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
DROP_TABLE
--------------
t
(1 row)

CALL nza..SUMMARY1000NUM('intable=nza..iris,
incolumn=sepalwidth;sepallength;petalwidth;petallength,
outtable=Result72');

SELECT * FROM Result72 ORDER BY columnname;

CALL nza..DROP_TABLE('Result72');

SUMMARY1000NUM
--------------

Done
(1 row)

COLUMNNAME    | COLUMNID | COUNTT |     AVERAGE     | VARIANCE       |      STDDEV      |     SKEWNESS     | KURTOSIS       | MINIMUM | MAXIMUM | MISSING
-------------+----------|--------+-----------------+----------------+-----------------+-----------------+--------------+--------+--------+---------
PETALLENGTH  |        3 |    150 | 3.7586666666667 | 3.1131794183445 | 1.7644204199523 | 1.7644204199523 | -0.27080472910747 |       1 |     6.9 |       0
PETALWIDTH   |        2 |    150 | 1.1986666666667 | 0.58241431767338 | 0.76316074170084 | 0.76316074170084 | -0.10359660798221 |     0.1 |     2.5 |       0
SEPALLENGTH  |        1 |    150 | 5.8433333333333 | 0.68569351230425 | 0.82806612797787 | 0.82806612797787 | 0.31071214388165 |       4.3 |     7.9 |       0
SEPALWIDTH   |        0 |    150 | 3.054           | 0.1880402684564 | 0.43359431136217 | 0.43359431136217 | 0.32959862667676 |       2 |     4.4 |       0

(4 rows)
DROP_TABLE
--------------
t
(1 row)

CALL nza..SUMMARY1000NUM('intable=nza..iris,
outtable=Result72, talk=yes');
SELECT * FROM Result72 ORDER BY columnname;
CALL nza..DROP_TABLE('Result72');
SUMMARY1000NUM
--------------
Done
(1 row)

COLUMNNAME | COLUMNID | COUNTT | AVERAGE | VARIANCE | STDDEV | SKEWNESS | KURTOSIS | MINIMUM | MAXIMUM | MISSING
-------------+----------+--------+---------+----------+--------+----------+----------+---------+---------+---------
ID          |        4 |    150 |    75.5 |   1.212106 |  43.4455 |  -1.4061 |       0 |      1 |     150 |     0
PETALLENGTH |        1 |    150 |   3.1132 |   0.584241 |  0.5824 |     0.1 |       0 |      0 |     150 |     0
PETALWIDTH  |        0 |    150 |   1.1986 |   0.582414 |  0.5856 |     2.5 |       0 |      0 |     150 |     0
SEPALLENGTH |        3 |    150 |  5.8433 |   0.685693 |  0.6856 |     4.3 |       0 |      0 |     150 |     0
SEPALWIDTH  |        2 |    150 |  3.054  |  0.310712 |  0.3107 |     3.0 |       0 |      0 |     150 |     0
CALL nza..SUMMARY1000NUM('intable=nza..iris, incolumn=sepalwidth;sepallength;petalwidth;petallength,outtable=Result72, by=class');

SELECT * FROM Result72 ORDER BY columnname, grouped_on;

CALL nza..DROP_TABLE('Result72');

SUMMARY1000NUM

---------

 Done

---------

COLUMNNAME | GROUPED_ON | COLUMNID | COUNTT | AVERAGE | VARIANCE | SKEWNESS | KURTOSIS | MINIMUM | MAXIMUM | MISSING

---------+------------+----------+--------+---------+----------+----------+----------+---------+---------+---------

PETALLENGTH | setosa     | 3        | 50     | 1.464   | 0.03010612244898 | 0.17351115943645 | 0.068972290185229 | 0.73739158718861 | 1       | 1.9     | 0

PETALLENGTH | versicolor | 3        | 50     | 4.26    | 0.22081632653061 | 0.46991097723996 | -0.58224738191143 | -0.13291378666874 | 3       | 5.1     | 0

PETALLENGTH | virginica  | 3        | 50     | 5.552   | 0.30458775510205 | 0.55189469566399 | 0.52746680546555 | -0.31134291509779 | 4.5     | 6.9     | 0

PETALWIDTH  | setosa     | 2        | 50     | 0.244   | 0.0114938775510205 | 0.10720950308168 | 1.149353251038 | 1.2105459872159 | 0.1      | 0.6     | 0

PETALWIDTH  | versicolor | 2        | 50     | 1.326   | 0.039106122448979 | 0.19775268000454 | -0.029932414130041 | -0.53807593528488 | 1       | 1.8     | 0

PETALWIDTH  | virginica  | 2        | 50     | 2.026   | 0.075432653061225 | 0.27465005563667 | -0.12429785366124 | -0.7081210454404 | 1.4      | 2.5     | 0
DROP_TABLE
------------
t
(1 row)

CREATE TABLE irisnull AS SELECT * FROM nza..iris;
INSERT INTO irisnull(id,petallength) SELECT id+300, petallength FROM nza..iris;
INSERT INTO irisnull(id,sepallength,class) SELECT id+600, sepallength, class FROM nza..iris;
CALL nza..SUMMARY1000NUM('intable=irisnull, incolumn=SEPALLENGTH;SEPALWIDTH;PETALLENGTH;SEPALWIDTH, outtable=Result7');
SELECT * FROM result7 ORDER BY columnname, columnid;
CALL nza..DROP_TABLE('Result7');
CALL nza..DROP_TABLE('irisnull');
CREATE TABLE withbooleans as select 'true'::boolean a, 'true'::boolean b;

INSERT INTO withbooleans VALUES('true'::boolean, 'true'::boolean);

INSERT INTO withbooleans VALUES('true'::boolean, 'false'::boolean);

INSERT INTO withbooleans VALUES('false'::boolean, 'false'::boolean);

CALL nza..SUMMARY1000NUM('intable=withbooleans, incolumn=a;b,
outtable=wd_05');
SELECT * FROM wd_05 ORDER BY columnname;
CALL nza..DROP_TABLE('wd_05');
CALL nza..DROP_TABLE('withbooleans');

SUMMARY1000NUM

--------------
Done
(1 row)

<table>
<thead>
<tr>
<th>COLUMNNAME</th>
<th>COLUMNID</th>
<th>COUNTT</th>
<th>AVERAGE</th>
<th>VARIANCE</th>
<th>STDDEV</th>
<th>SKEWNESS</th>
<th>KURTOSIS</th>
<th>MINIMUM</th>
<th>MAXIMUM</th>
<th>MISSING</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>4</td>
<td>0.75</td>
<td></td>
<td>0.25</td>
<td>0.5</td>
<td>-1</td>
<td>-1.25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>4</td>
<td>0.5</td>
<td></td>
<td>0.33333333333333</td>
<td>0.57735026918963</td>
<td>-2.884440295753e-16</td>
<td>-2.25</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

(2 rows)

DROP_TABLE

--------------
t
(1 row)

DROP_TABLE

--------------
t
(1 row)

Related Functions
► category Analytics - Statistics
► MOMENTS
► SUMMARY1000
SUMMARY1000TIME - Summary of up to 1000 time columns

This stored procedure calculates summary statistics of up to 1000 input columns of type time or timetz. The missing value count, non missing value count, average, variance, standard deviation, skewness, (excess) kurtosis, minimum and maximum of the input columns is gathered.

Usage

The SUMMARY1000TIME stored procedure has the following syntax:

► SUMMARY1000TIME(NVARCHAR(ANY) paramString)

▲ Parameters

► paramString

cOMMA-separated list of <parameter>=<value> entries with parameters below

Type: NVARCHAR(ANY)

► intable

the input table

Type: NVARCHAR(256)

► outtable

the output table to write the summary statistics into

Type: NVARCHAR(256)

► incolumn

the input table columns of type time or timetz separated by a semicolon (;). If this parameter is not specified, all input table columns of type time or timetz are analyzed.

Type: NVARCHAR(ANY)

Default: <none>

► by

the input table column which splits the data into groups for which the operation is to be performed

Type: NVARCHAR(128)

Default: <none>

► talk

flag indicating whether additional information on the progress of the analysis must be displayed

Type: NVARCHAR(ANY)

Default: no

▲ Returns

NVARCHAR(ANY) an informative string

Details

This stored procedure gathers statistics about the input table columns of type time or timetz. No more than
1000 columns are accepted. Null values are not part of the statistics computation (per column), but are counted.

An output table `<outtable>` is created with following columns: `columnname`, `columnid`, `countt`, average, variance, stddev, skewness, kurtosis, minimum, maximum, missing. If the parameter `by` is specified, an additional column `grouped_on` is added to indicate for which group the statistics have been gathered. A detailed description of the statistics computed here can be found in the documentation of the stored procedure `MOMENTS`. If statistics cannot be gathered, for example when there are less than 2 values for average, the output table contains a NULL value.

Note that times are stored internally as bigint representing the number of microseconds passed since the begin of the day, plus, for timetz times, an integer representing the number of hours the time zone is shifted from the Greenwich line (plus means towards East). For purposes of analysis the timetz structure is flattened to a bigint indicating time in microseconds at Greenwich. The average, standard deviation, minimum and maximum is displayed as an interval. The interval is needed because the timetz may represent the time on the same day as is at Greenwich, the next day (more than 24 hours away) or the previous day ("negative time"). The variance is expressed in "squared seconds".

**Examples**

```sql
CREATE TABLE withtimes AS SELECT '00:00:00 '::time a,
    '00:00:00 '::time b;

INSERT INTO withtimes VALUES('00:00:00 '::time,
    '00:00:00 '::time);

INSERT INTO withtimes VALUES('00:30:00 '::time,
    '00:50:00 '::time);

INSERT INTO withtimes VALUES('01:00:00 '::time,
    '00:10:00 '::time);

CALL nza..SUMMARY1000TIME('intable=withtimes,
    outtable=wd_05');

SELECT * FROM wd_05 ORDER BY columnname;

CALL nza..DROP_TABLE('wd_05');

CALL nza..DROP_TABLE('withtimes');

SUMMARY1000TIME
-----------------

Done

(1 row)

<table>
<thead>
<tr>
<th>COLUMNNAME</th>
<th>COLUMNID</th>
<th>COUNTT</th>
<th>AVERAGE</th>
<th>VARIANCE</th>
<th>STDDEV</th>
<th>SKEWNESS</th>
<th>KURTOSIS</th>
<th>MINIMUM</th>
<th>MAXIMUM</th>
<th>MISSING</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

------------------------------------------

630 00X6332-00 Rev. 1
CREATE TABLE withtimetz AS SELECT '06:00:00+01'::timetz a, '07:00:00-03'::timetz b;

INSERT INTO withtimetz VALUES('00:00:00-01'::timetz, '00:00:00'::timetz);

INSERT INTO withtimetz VALUES('00:30:00-02'::timetz, '00:50:00'::timetz);

INSERT INTO withtimetz VALUES('01:00:00-03'::timetz, '00:10:00'::timetz);

INSERT INTO withtimetz VALUES('01:00:00+03'::timetz, '23:10:00-03'::timetz);

CALL nza..SUMMARY1000TIME('intable=withtimetzs, outtable=wd_05');

SELECT * FROM wd_05 ORDER BY columnname;

CALL nza..DROP_TABLE('wd_05');

CALL nza..DROP_TABLE('withtimetzs');

SUMMARY1000TIME

Done

(1 row)
<table>
<thead>
<tr>
<th>COLUMNNAME</th>
<th>COLUMNID</th>
<th>COUNTT</th>
<th>AVERAGE</th>
<th>VARIANCE</th>
<th>STDDEV</th>
<th>SKEWNESS</th>
<th>KURTOSIS</th>
<th>MINIMUM</th>
<th>MAXIMUM</th>
<th>MISSING</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>+-------+----------+----------+------------+--------------</td>
<td>---------------</td>
<td>---------------</td>
<td>--------------</td>
<td>-------------</td>
<td>---------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>5</td>
<td>02:06:00</td>
<td>97848000</td>
<td>-0.46926215253497</td>
<td>-0.46926215253497</td>
<td>-1.386798824613</td>
<td>-1 days 22:00:00</td>
<td>05:00:00</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>5</td>
<td>07:26:00</td>
<td>1649628000</td>
<td>0.95908656819197</td>
<td>0.95908656819197</td>
<td>-0.97954655695623</td>
<td>00:00:00</td>
<td>1 days 02:10:00</td>
<td>0</td>
</tr>
</tbody>
</table>

(2 rows)

DROP_TABLE
--------------
t
(1 row)

Related Functions
► category Analytics - Statistics
► MOMENTS
► SUMMARY1000

SUMMARY1000TIMESTAMP - Summary of up to 1000 timestamp columns

This stored procedure calculates summary statistics of up to 1000 input columns of type timestamp. The missing value count, non missing value count, average, variance, standard deviation, skewness, (excess) kurtosis, minimum and maximum of the input columns is gathered.

Usage
The SUMMARY1000TIMESTAMP stored procedure has the following syntax:
SUMMARY 1000 TIMESTAMP (NVARCHAR(ANY) paramString)

Parameters

- paramString
  comma-separated list of <parameter>=<value> entries with parameters below
  Type: NVARCHAR(ANY)

- intable
  the input table
  Type: NVARCHAR(256)

- outtable
  the output table to write the summary statistics into
  Type: NVARCHAR(256)

- incolumn
  the input table columns of type timestamp separated by a semicolon (;). If this parameter is not specified, all input table columns of type timestamp are analyzed.
  Type: NVARCHAR(ANY)
  Default: <none>

- by
  the input table column which splits the data into groups for which the operation is to be performed
  Type: NVARCHAR(128)
  Default: <none>

- talk
  flag indicating whether additional information on the progress of the analysis must be displayed
  Type: NVARCHAR(ANY)
  Default: no

Returns

NVARCHAR(ANY) an informative string

Details

This stored procedure gathers statistics about the input table columns of type timestamp. No more than 1000 columns are accepted. Null values are not part of the statistics computation (per column), but are counted.

An output table <outtable> is created with following columns: columnname, columnid, countt, average, variance, stddev, skewness, kurtosis, minimum, maximum, missing. If the parameter by is specified, an additional column grouped_on is added to indicate for which group the statistics have been gathered. A detailed description of the statistics computed here can be found in the documentation of the stored procedure MOMENTS. If statistics cannot be gathered, for example when there are less than 2 values for average, the output table contains a NULL value.

Note that timestamps are stored internally as big integers representing the number of microseconds since January 1st, 2000, 00:00:00. The average, minimum and maximum is displayed as a timestamp. The standard deviation is displayed as an interval. The variance is expressed in "squared seconds".
Examples

CREATE TABLE withtimestamps(a timestamp, b timestamp);

INSERT INTO withtimestamps VALUES('2012-01-02'::timestamp, '2012-01-05'::timestamp);
INSERT INTO withtimestamps VALUES('2012-01-12'::timestamp, '2012-01-25'::timestamp);
INSERT INTO withtimestamps VALUES('2012-01-22'::timestamp, '2012-02-13'::timestamp);
INSERT INTO withtimestamps VALUES('2012-02-01'::timestamp, '2012-03-03'::timestamp);

CALL nza..SUMMARY1000TIMESTAMP('intable=withtimestamps, incolumn=a;b, outtable=wd_05');

SELECT * FROM wd_05 ORDER BY columnname;

CALL nza..DROP_TABLE('wd_05');
CALL nza..DROP_TABLE('withtimestamps');

SUMMARY1000TIMESTAMP

--------------

Done

(1 row)

<table>
<thead>
<tr>
<th>COLUMNN</th>
<th>COLUMNID</th>
<th>COUNTT</th>
<th>AVERAGE</th>
<th>VARIANCE</th>
<th>STDDEV</th>
<th>SKEWNESS</th>
<th>KURTOSIS</th>
<th>MINIMUM</th>
<th>MAXIMUM</th>
<th>_MISSING</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td>0</td>
<td>4</td>
<td>2012-01-17 00:00:00</td>
<td>1244159999990.7</td>
<td>22 days 22:02:12</td>
<td>-1.6366285512535</td>
<td>2012-01-02 00:00:00</td>
<td>2012-02-01 00:00:00</td>
<td>0</td>
</tr>
</tbody>
</table>

(2 rows)

DROP_TABLE
Related Functions

- category Analytics - Statistics
- MOMENTS
- SUMMARY1000

T_LS_TEST - T-Student test for the linear relationship of two columns

This stored procedure calculates the t-Student statistics of two numeric input columns to evaluate the significance of the linear relationship between them.

Usage

The T_LS_TEST stored procedure has the following syntax:

**T_LS_TEST(NVARCHAR(ANY) paramString)**

▲ Parameters

- **paramString**
  comma-separated list of <parameter>=<value> entries with parameters below
  Type: NVARCHAR(ANY)

- **intable**
  the input table
  Type: NVARCHAR(256)

- **incolumn**
  the two numeric input table columns, separated by a semicolon (;). One column must be followed by :X, the other column by :Y.
  Type: NVARCHAR(ANY)

- **slope**
  the expected direction of dependence \( Y = X \times \text{slope} + \text{constant} \)
  Type: DOUBLE

- **by**
  the input table column which splits the data into groups for which the operation is to be performed
Type: NVARCHAR(128)
Default: <none>

► **outtable**
the output table to write the t-Student statistics into
Type: NVARCHAR(256)

▲ **Returns**
DOUBLE the probability of the dependence of the two input table columns, or the number of groups in the output table for which t-Student statistics have been calculated

**Details**
This stored procedure calculates the t-Student statistics between two input columns, either in the whole input table or within the groups defined in the column specified by parameter <by>. This t-Student test is a measure saying whether or not the two columns are linearly related with a given slope.

The output table is created with following columns: percentage, t_ls_test. If the parameter by is specified, an additional column <by> is added to indicate for which group the t-Student statistics have been calculated. The column t_ls_test contains a string with the t-Student value, the degree of freedom, the variance and the linear coefficients relating both columns. The column percentage indicates the probability of the dependence of the two columns:

- If the percentage ranges from 0 to 0.05, the slope is too high,
- If the percentage ranges between 0.05 and 0.95, the indicated slope is acceptable, the lower the percentage the more independent the columns are,
- If the percentage ranges between 0.95 and 1, the slope is too low.

**Examples**

```sql
CALL nza..T_LS_TEST('intable=nza..iris,
inctolumn=petallength:X;sepallength:Y, slope=1.5,
outtable=Result7');
SELECT * FROM result7;
CALL nza..DROP_TABLE('result7');
```

```
T_LS_TEST
----------
0.27454143618784
```
(1 row)

<table>
<thead>
<tr>
<th>PERCENTAGE</th>
<th>T_LS_TEST</th>
<th>------------------</th>
</tr>
</thead>
<tbody>
<tr>
<td>636</td>
<td>00X6332-00 Rev. 1</td>
<td></td>
</tr>
</tbody>
</table>
0.27454143618784 | tStat= -0.600513, df= 148, equation= y=4.30557+x*0.409126 , varE= 0.164597
(1 row)

DROP_TABLE
-------------
t
(1 row)

CALL nza..T_LS_TEST('intable=nza..iris, incolumn=petallength:X;sepallength:Y, slope=1.5, outtable=Result7, by=class');
SELECT * FROM result7 ORDER BY class;
CALL nza..DROP_TABLE('result7');

T_LS_TEST
-------------
3
(1 row)

<table>
<thead>
<tr>
<th>PERCENTAGE</th>
<th>T_LS_TEST</th>
<th>CLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0079567225658825</td>
<td>tStat= -2.4994, df= 48, equation= y=4.2212+x*0.536063 , varE= 0.115598</td>
<td>setosa</td>
</tr>
<tr>
<td>0.12846187162989</td>
<td>tStat= -1.14735, df= 48, equation= y=2.40752+x*0.828281 , varE= 0.114942</td>
<td>versicolor</td>
</tr>
<tr>
<td>0.21891760885789</td>
<td>tStat= -0.782378, df= 48, equation= y=1.05966+x*0.995739 , varE= 0.102345</td>
<td>virginica</td>
</tr>
</tbody>
</table>
(3 rows)

DROP_TABLE
-------------
Related Functions

- category Analytics - Statistics
- T_ME_TEST
- T_PMD_TEST
- T_UMD_TEST

**T_LS_TEST_S_AGG - T-Student test between linearly dependent samples as string**

This function calculates the t-Student statistics of two numeric variables to evaluate the significance of the linear relationship between them.

**Usage**

The T_LS_TEST_S_AGG aggregate has the following syntax:

```
T_LS_TEST_S_AGG(DOUBLE X, DOUBLE Y, DOUBLE Slope)
```

**Parameters**

- **X**
  the first variable
  Type: DOUBLE

- **Y**
  the second variable
  Type: DOUBLE

- **Slope**
  the constant value indicating the expected direction of dependence $Y=X\times Slope+constant$
  Type: DOUBLE

**Returns**

NVARCHAR(200) a string containing the t-Student statistics, the degree of freedom, the linear equation relating X and Y and the variance of the equation.

**Details**

This function calculates the t-Student statistics between two input variables. This t-Student test is a measure saying whether or not the two columns are linearly related with a given slope.
The tStatistics is Student-t distributed with countX-2 degrees of freedom. The SQL procedure to calculate the same statistics on a given table would look like:

```sql
SELECT (avgBeta-betaZero) * sqrt(countX-2)/sqrt(varE*(countX-1) + varX*(countX-1)) AS tStatistics FROM (SELECT COUNT(x-y) AS countX, VAR(y-avgBeta*x-avgAlpha) AS varE, VAR(x) AS varX FROM myTable, (SELECT SUM((x-avgix)*(y-avgiy))/SUM((x-avgix)*(x-avgix)) AS avgBeta, avgix*SUM((x-avgix)*(y-avgiy))/SUM((x-avgix)*(x-avgix)) AS avgAlpha FROM myTable, (SELECT AVG(x) AS avgix, AVG(y) AS avgiy FROM myTable) AS One ) as Two)
```

**Examples**

```sql
SELECT nza..T_LS_TEST_S_AGG(petallength,sepallength,1.5), nza..PT(nza..T_LS_TEST_S_AGG(petallength,sepallength,1.5)) FROM nza..iris;
```

```
<table>
<thead>
<tr>
<th>t_LS_TEST_S_AGG</th>
<th>PT</th>
</tr>
</thead>
<tbody>
<tr>
<td>----------------</td>
<td>----</td>
</tr>
<tr>
<td>tStat= -0.600513, df= 148, equation= y=4.30557+x*0.409126 , varE= 0.164597</td>
<td>0.27454143618784</td>
</tr>
</tbody>
</table>
```

(1 row)

**Related Functions**

- **category Analytics - Statistics**
- **PT**
- **T_LS_TEST**

**T_ME_TEST - T-Student test for the expected mean of a column**

This stored procedure calculates the t-Student statistics of a numeric input column to evaluate the significance of the difference between the sample mean and an expected mean value.

**Usage**

The T_ME_TEST stored procedure has the following syntax:

```sql
T_ME_TEST(NVARCHAR(ANY) paramString)
```

**Parameters**

- **paramString**

  comma-separated list of <parameter>=<value> entries with parameters below

  - **Type:** NVARCHAR(ANY)

- **intable**

  the input table

  - **Type:** NVARCHAR(256)
► **incolumn**
  the numeric input table column
  Type: NVARCHAR(128)

► **mean**
  the expected mean value of the input column
  Type: DOUBLE

► **by**
  the input table column which splits the data into groups for which the operation is to be performed
  Type: NVARCHAR(128)
  Default: <none>

► **outtable**
  the output table to write the t-Student statistics into
  Type: NVARCHAR(256)

▲ Returns
  DOUBLE the probability of the input table column to have the expected mean, or the number of groups in the output table for which t-Student statistics have been calculated

**Details**

This stored procedure calculates the t-Student statistics of an input column with the expected mean, either in the whole input table or within the groups defined in the column specified by parameter `<by>`. This t-Student test is a measure saying whether or not the column has the given mean.

The output table is created with following columns: percentage, t_me_test. If the parameter by is specified, an additional column `<by>` is added to indicate for which group the t-Student statistics have been calculated. The column t_me_test contains a string with the t-Student value and the degree of freedom. The column percentage indicates the probability of the column to have the expected mean value:
- If the percentage ranges from 0 to 0.05, the expected mean value is significantly too high,
- If the percentage ranges between 0.05 and 0.95, the expected mean value matches the mean value of the column,
- If the percentage ranges between 0.95 and 1, the expected mean value is significantly too low.

**Examples**

```sql
CALL nza..T_ME_TEST('intable=nza..iris, incolumn=petallength, mean=3.76, outtable=Result7');
SELECT * FROM result7;
CALL nza..DROP_TABLE('result7');
```

---

T_ME_TEST
0.4963139801387
(1 row)

PERCENTAGE | T_ME_TEST
-----------------+--------------------------------
0.4963139801387 | tStat= -0.00925513, df= 149
(1 row)

DROP_TABLE
------------

CALL nza..T_ME_TEST('intable=nza..iris,incolumn=petallength, mean=3.76, outtable=Result7, by=class');
SELECT * FROM result7 ORDER BY class;
CALL nza..DROP_TABLE('result7');

T_ME_TEST
-----------

3
(1 row)

PERCENTAGE | T_ME_TEST | CLASS
-----------------+----------+-------
0 | tStat= -93.5685, df= 49 | setosa
0.99999999948581 | tStat= 7.52384, df= 49 | versicolor
1 | tStat= 22.9597, df= 49 | virginica
(3 rows)

DROP_TABLE
------------
**Related Functions**

- category Analytics - Statistics
- T_LS_TEST
- T_PMD_TEST
- T/umd_TEST

**T_ME_TEST_S_AGG - T-Student test for the expected mean of a variable as string**

This function calculates the t-Student statistics of a numeric variable to evaluate the significance of the difference between the sample mean and an expected mean value.

**Usage**

The T_ME_TEST_S_AGG aggregate has the following syntax:

\[ T\_ME\_TEST\_S\_AGG(DOUBLE \, X, DOUBLE \, mean) \]

- **Parameters**
  - **X**
    - the input variable
    - Type: DOUBLE
  - **mean**
    - a constant indicating the expected mean value of the input variable
    - Type: DOUBLE

- **Returns**
  - NVARCHAR(200) a string containing the t-Student statistics and the degree of freedom of the input variable

**Details**

This function calculates the t-Student statistics and the degree of freedom of an input column with the expected mean. This t-Student test is a measure saying whether or not the column has the given mean.

**Examples**

```sql
SELECT nza..T_ME_TEST_S_AGG(petallength, 3.76) FROM nza..iris;
```
**T_ME_TEST_S_AGG**

---

tStat= -0.00925513,  df= 149

(1 row)

---

**Related Functions**

- category Analytics - Statistics
- T_ME_TEST

---

**T_PMD_TEST - T-Student test for the expected mean difference between two paired columns**

This stored procedure calculates the t-Student statistics of two paired numeric input columns to evaluate the significance of the difference of their mean values compared with an expected value.

**Usage**

The T_PMD_TEST stored procedure has the following syntax:

```
T_PMD_TEST(NVARCHAR(ANY) paramString)
```

- **paramString**
  
  comma-separated list of <parameter>=<value> entries with parameters below

  - **intable**
    
    the input table

    Type: NVARCHAR(256)

  - **outtable**
    
    the output table to write the t-Student statistics into

    Type: NVARCHAR(256)

  - **incolumn**
    
    the two numeric input table columns, separated by a semicolon (;). One column must be followed by :X, the other column by :Y.

    Type: NVARCHAR(ANY)

  - **expdiff**
    
    the expected difference between the mean values of the input columns (Y - X)

    Type: DOUBLE

    Default: 0
by
the input table column which splits the data into groups for which the operation is to be performed
Type: NVARCHAR(128)
Default: <none>

Returns
DOUBLE the probability of the two input table columns to take different values by expdiff in average, or the number of groups in the output table for which t-Student statistics have been calculated

Details
This stored procedure calculates the t-Student statistics of two paired columns, either in the whole input table or within the groups defined in the column specified by parameter <by>. This t-Student test is a measure saying whether or not the two columns take different values, on average distinct by <expdiff>.

The output table is created with following columns: percentage, tstudpairedmeandiff. If the parameter by is specified, an additional column <by> is added to indicate for which group the t-Student statistics have been calculated. The column tstudpairedmeandiff contains a string with the t-Student value and the degree of freedom. The column percentage indicates the probability of the column to have the expected mean value:
- If the percentage ranges from 0 to 0.05, the column Y has significantly bigger values than the column X,
- If the percentage ranges between 0.05 and 0.95, the expected mean difference value matches the difference between the mean values of the two columns,
- If the percentage ranges between 0.95 and 1, the column Y has significantly smaller values than the column X.

Examples
CALL nza..T_PMD_TEST('intable=nza..iris,
incolumn=petallength:X;sepallength:Y, outtable=Result7,
by=class');

SELECT * FROM result7 ORDER BY class;
CALL nza..DROP_TABLE('result7');

T_PMD_TEST
---------
3
(1 row)

PERCENTAGE | TSTUDPAIREDMEANDIFF | CLASS
<table>
<thead>
<tr>
<th>N</th>
<th>tStat</th>
<th>df</th>
<th>Species</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-71.6829</td>
<td>49</td>
<td>setosa</td>
</tr>
<tr>
<td>0</td>
<td>-34.0059</td>
<td>49</td>
<td>versicolor</td>
</tr>
<tr>
<td>0</td>
<td>-22.8981</td>
<td>49</td>
<td>virginica</td>
</tr>
</tbody>
</table>

(3 rows)

DROP_TABLE
-----------
t
(1 row)

CALL nza..T_PMD_TEST('intable=nza..iris, incolumn=petallength:X;sepallength:Y, outtable=Result7');
SELECT * FROM result7;
CALL nza..DROP_TABLE('result7');

T_PMD_TEST
-----------

0
(1 row)

PERCENTAGE | TSTUDPAIREDMEANDIFF
-------------+-------------------------

<table>
<thead>
<tr>
<th>N</th>
<th>tStat</th>
<th>df</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-22.8226</td>
<td>149</td>
</tr>
</tbody>
</table>

(1 row)

DROP_TABLE
-----------
t
(1 row)

CALL nza..T_PMD_TEST('intable=nza..iris, incolumn=petallength:X;sepallength:Y, outtable=Result7,
expdiff=-2');
SELECT * FROM result7;
CALL nza..DROP_TABLE('result7');

T_PMD_TEST

-------------
0.17773404645354
(1 row)

<table>
<thead>
<tr>
<th>PERCENTAGE</th>
<th>TSTUDPAIREDMEANDIFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.17773404645354</td>
<td>tStat= -0.926918, df= 149</td>
</tr>
</tbody>
</table>

(1 row)

DROP_TABLE

------------
t
(1 row)

Related Functions
► category Analytics - Statistics
► T_LS_TEST
► T_ME_TEST
► T_UMD_TEST
► WILCOXON_TEST

T_PMD_TEST_S_AGG - T-Student test for the expected mean difference between two paired variables as string

This function calculates the t-Student statistics of two paired numeric input columns to evaluate the significance of the difference of their mean values compared with an expected value

Usage
The T_PMD_TEST_S_AGG aggregate has the following syntax:
T_PMD_TEST_S_AGG(DOUBLE X, DOUBLE Y, DOUBLE expdiff)

Parameters

- **X**
  
  the first input variable
  
  Type: DOUBLE

- **Y**
  
  the second input variable
  
  Type: DOUBLE

- **expdiff**
  
  a constant indicating the expected difference between the mean values of X and Y
  
  Type: DOUBLE

Returns

NVARCHAR(200) a string containing the t-Student statistics and the degree of freedom of the input variables

Details

This function calculates the t-Student statistics and the degree of freedom of two paired columns. This t-Student test is a measure saying whether or not the two columns take different values, on average distinct by </expdiff>.

Examples

```
SELECT nza..T_PMD_TEST_S_AGG(petallength, sepallength, -2) FROM nza..iris;
```

```
T_PMD_TEST_S_AGG

-----------------------------
tStat= -0.926918,   df= 149

(1 row)
```

Related Functions

- category Analytics - Statistics
- T_PMD_TEST

T_TEST_AGG - T-Student test of a variable split into two classes

This function calculates the t-Student statistics of a numeric variable split into two classes

Usage

The T_TEST_AGG aggregate has the following syntax:
T_TEST_AGG(INT4 class, DOUBLE X)

Parameters

- **class**
  the class (1 or 2) of the input variable value. Rows with a class other than 1 and 2 are ignored
  Type: INT4
  Min: 1
  Max: 2

- **X**
  the input variable
  Type: DOUBLE

Returns

DOUBLE the t-Student statistics for the two classes of the input variable

Details

This function calculates the t-Student statistics for an input variable whose values are split into two classes 1 and 2. The classes can be of unequal size and the values in it have different variances.

Examples

```
SELECT nza..T_TEST_AGG(case when class='setosa' then 1 when class='virginica' then 2 when class='versicolor' then 0 else 0 end,petallength) FROM nza..iris;
```

```
T_TEST_AGG
------------------
-49.965703359356
(1 row)
```

Related Functions

- category Analytics - Statistics
- T_TEST_S_AGG

T_TEST_S_AGG - T-Student test of a variable split into two classes as string

This function calculates the t-Student statistics of a numeric variable split into two classes.
Usage

The T_TEST_S_AGG aggregate has the following syntax:

- **T_TEST_S_AGG** (INT4 class, DOUBLE X)
  - **Parameters**
    - **class**
      the class (1 or 2) of the input variable value. Rows with a class other than 1 and 2 are ignored
      Type: INT4
      Min: 1
      Max: 2
    - **X**
      the input variable
      Type: DOUBLE
  - **Returns**
    NVARCHAR(200) a string containing the t-Student statistics and the degree of freedom for the two classes of the input variable

Details

This function calculates the t-Student statistics and the degree of freedom for an input variable whose values are split into two classes 1 and 2. The classes can be of unequal size and the values in it have different variances.

Examples

```sql
SELECT nza..PT(nza..T_TEST_S_AGG(case when class='setosa' then 0 when class='virginica' then 1 when class='versicolor' then 2 else 0 end, petallength)), nza..T_TEST_S_AGG(case when class='setosa' then 0 when class='virginica' then 1 when class='versicolor' then 2 else 0 end, petallength) FROM nza..iris;
```

<table>
<thead>
<tr>
<th>PT</th>
<th>T_TEST_S_AGG</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>tStat= 12.6038, df= 95</td>
</tr>
</tbody>
</table>

(1 row)

Related Functions

- category Analytics - Statistics
- PT
- T_TEST_AGG
T_UMD_TEST - T-Student test for the expected mean difference between class values of a column

This stored procedure calculates the t-Student statistics of a numeric input column, whose values are split into two classes, to evaluate the significance of the difference of the class mean values.

Usage

The T_UMD_TEST stored procedure has the following syntax:

```
T_UMD_TEST(NVARCHAR(ANY) paramString)
```

Parameters

- `paramString`:
  comma-separated list of `<parameter>=<value>` entries with parameters below
  Type: NVARCHAR(ANY)
- `intable`:
  the input table
  Type: NVARCHAR(256)
- `outtable`:
  the output table to write the t-Student statistics into
  Type: NVARCHAR(256)
- `incolumn`:
  the numeric input table column
  Type: NVARCHAR(128)
- `class`:
  the input table column which splits data into two classes. The class column name is followed by two class values preceded by a colon (:).
  Type: NVARCHAR(ANY)
- `by`:
  the input table column which splits the data into groups for which the operation is to be performed
  Type: NVARCHAR(128)
  Default: <none>

Returns

DOUBLE the probability of the input table column to have similar mean values in both classes, or the number of groups in the output table for which t-Student statistics have been calculated.

Details

This stored procedure calculates the t-Student statistics of a column whose values are split into...
two classes, either in the whole input table or within the groups defined in the column specified by parameter `<by>`. This t-Student test is a measure saying whether or not the two classes have the same mean value.

The output table is created with following columns: percentage, t_umd_test. If the parameter by is specified, an additional column `<by>` is added to indicate for which group the t-Student statistics have been calculated. The column t_umd_test contains a string with the t-Student value and the degree of freedom. The column percentage indicates the probability of the column to have the expected mean value:
- If the percentage ranges from 0 to 0.05, the second class has a significantly bigger mean value than the first class,
- If the percentage ranges between 0.05 and 0.95, the two classes have the same mean value,
- If the percentage ranges between 0.95 and 1, the second class has a significantly smaller mean value than the first class.

**Examples**

```sql
CALL nza..T_UMD_TEST('intable=nza..iris, incolumn=petallength, class=class:"virginica":"setosa", outtable=Result7');
SELECT * FROM result7;
CALL nza..DROP_TABLE('result7');
```

```
T_UMD_TEST
----------
1          
(1 row)

PERCENTAGE | T_UMD_TEST
-----------+---------------------------
1 | tStat=49.9657, df=58
(1 row)

DROP_TABLE
----------

CALL nza..T_UMD_TEST('intable=nza..iris, incolumn=sepalwidth, class=class:"virginica":"versicolor", outtable=Result7');
SELECT * FROM result7;
```

```sql
CALL nza..T_UMD_TEST('intable=nza..iris, incolumn=sepalwidth, class=class:"virginica":"versicolor", outtable=Result7');
SELECT * FROM result7;
```

```sql
T_UMD_TEST
----------
1          
(1 row)

PERCENTAGE | T_UMD_TEST
-----------+---------------------------
1 | tStat=49.9657, df=58
(1 row)

DROP_TABLE
----------

CALL nza..T_UMD_TEST('intable=nza..iris, incolumn=sepalwidth, class=class:"virginica":"versicolor", outtable=Result7');
SELECT * FROM result7;
```
CALL nza..DROP_TABLE('result7');

T_UMD_TEST

------------------
0.99908780317504
(1 row)

<table>
<thead>
<tr>
<th>PERCENTAGE</th>
<th>T_UMD_TEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.99908780317504</td>
<td>tStat= 3.20576, df= 97</td>
</tr>
</tbody>
</table>
(1 row)

DROP_TABLE

-----------
t
(1 row)

Related Functions
► category Analytics - Statistics
► PT
► T_LS_TEST
► T_ME_TEST
► T_PMD_TEST
► MWW_TEST

TANET_APPLY - Apply a tree-augmented network model
This stored procedure applies a tree-augmented network model to generate regression predictions for a dataset

Usage
The TANET_APPLY stored procedure has the following syntax:

► TANET_APPLY(NVARCHAR(ANY) paramString)
  ▲ Parameters
    ► paramString
A comma-separated list of `<parameter>=<value>` entries using the parameters below.

Type: NVARCHAR(ANY)

- **model**
  The Bayesian Network model to apply.
  Type: NVARCHAR(ANY)

- **intable**
  The name of the table containing input.
  Type: NVARCHAR(256)

- **id**
  The name of the column in the input table that identifies a unique instance ID.
  Type: NVARCHAR(128)

- **target**
  The model variable to be predicted.
  Type: NVARCHAR(128)

- **outtable**
  The name of the output table where the predictions are to be stored.
  Type: NVARCHAR(256)

- **type**
  The type of prediction to be made. Valid values are: best (most correlated neighbor).
  Type: NVARCHAR(ANY)
  Default: best

▲ Returns
NVARCHAR(3000) A termination message

**Details**

This procedure predict values of a continuous variable using a tree-augmented network model generated by TANET_GROW. Only one approach may be exploited:

1. type=best: most strongly correlated neighbor. In the tree-augmented network, the target variable's direct neighbors are identified. The neighbor is selected that has the strongest correlation to the target variable, according to its class. The input table column value corresponding to this neighbor is then used for prediction.

The output table contains following columns: `<id>`, `<target>_pred`.

**Examples**

```
CALL nza..TBNET_GROW('model=struc_iris, intable=nza..iris, incolumn=SEPALLENGTH;SEPALWIDTH;PETALLENGTH;PETALWIDTH');
CALL nza..TANET_GROW('model=plstruc_iris, intable=nza..iris, class=class, inmodel=struc_iris');
CALL nza..TANET_APPLY('model=plstruc_iris, intable=nza..iris, id=id, target=PETALLENGTH, outtable=iris_pred');
```
SELECT a.*, b.petallength, b.class FROM iris_pred
a, nza..iris b WHERE a.id=b.id AND mod(a.id,50)<3 ORDER BY id;

CALL nza..DROP_MODEL('model=struc_iris');
CALL nza..DROP_MODEL('model=plstruc_iris');
CALL nza..DROP_TABLE('iris_pred');

TBNET_GROW

------------------------------------------------------

Over node no4 in TBNet_Grow process
(1 row)

TANET_GROW

-------------

9

(1 row)

TANET_APPLY

-------------

Done

(1 row)

<table>
<thead>
<tr>
<th>ID</th>
<th>PETALLENGTH_PRED</th>
<th>PETALLENGTH</th>
<th>CLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.4695964749536</td>
<td>1.4</td>
<td>setosa</td>
</tr>
<tr>
<td>2</td>
<td>1.4695964749536</td>
<td>1.4</td>
<td>setosa</td>
</tr>
<tr>
<td>50</td>
<td>1.4695964749536</td>
<td>1.4</td>
<td>setosa</td>
</tr>
<tr>
<td>51</td>
<td>4.4346825882007</td>
<td>4.7</td>
<td>versicolor</td>
</tr>
<tr>
<td>52</td>
<td>4.6216150587153</td>
<td>4.5</td>
<td>versicolor</td>
</tr>
<tr>
<td>100</td>
<td>4.2477501176862</td>
<td>4.1</td>
<td>versicolor</td>
</tr>
<tr>
<td>101</td>
<td>5.3484352725181</td>
<td>6</td>
<td>virginica</td>
</tr>
<tr>
<td>102</td>
<td>4.9733948945806</td>
<td>5.1</td>
<td>virginica</td>
</tr>
<tr>
<td>150</td>
<td>5.0484029701681</td>
<td>5.1</td>
<td>virginica</td>
</tr>
</tbody>
</table>
DROP_MODEL
-------------
t
(1 row)

DROP_MODEL
-------------
t
(1 row)

DROP_TABLE
-------------
t
(1 row)

Related Functions

- category Analytics - Regression
- TANET_GROW
- TBNET_APPLY
- TBNET_GROW

TANET_CLASSAPPLY - Apply a tree-augmented network

The prediction is done based on the closest neighbor.

Usage

The TANET_CLASSAPPLY stored procedure has the following syntax:

- **TANET_CLASSAPPLY**
  - **paramString**
    - input parameters specification
    - Type: NVARCHAR(ANY)
  - **intable**
table name
  Type: NVARCHAR(ANY)
► model
  the name of table with the model
  Type: NVARCHAR(ANY)
► id
  the name of the variable that identifies the cases in the intable
  Type: NVARCHAR(ANY)
► alpha
  - degree of mixture of TAN and naive bayes
  Type: NVARCHAR(ANY)
► locality
  = L or G for local or global estimate
  Type: NVARCHAR(ANY)
► outtable
  result table name;
  Type: NVARCHAR(ANY)

▲ Returns
  NVARCHAR(ANY) - temporarily a termination message

Details
The Bayesian network, which is described by a model that is generated by TANet_Grow, is used for value prediction in a continuous table. The prediction is based on the formula $E(Y|X) = EY + \alpha \cdot \frac{\sigma_y}{\sigma_x} \cdot (X - EX)/\sigma_x$.

Prediction is done by the most strongly correlated neighbor in the Bayesian network (type=best). In the Bayesian network, the direct neighbors of the target variable are identified. Their correlations to the target variable are compared through the absolute value, and the variable that has the strongest correlation to the target variable is selected. The selected variable is then used for prediction.

Examples
```
drop table struc_iris;
call nza..TBNet_Grow('intable=nza..iris,incolumn=SEPALLENGTH; SEPALWIDTH; PETALLENGTH; PETALWIDTH,model=struc_iris');
drop table plstruc_iris;
call
```
nza..TANet_Grow('intable=nza..iris,class=class,inmodel=struc_iris,model=plstruc_iris');

drop table xxxx;

call nza..TANet_classapply('intable=nza..iris,id=id,outtable=xxxx,model=plstruc_iris');

select xxxx.*,iris.class from xxxx,nza..iris where xxxx.id=iris.id and mod(xxxx.id,50)<3;

<table>
<thead>
<tr>
<th>ID</th>
<th>setosa</th>
<th>versicolor</th>
<th>virginica</th>
</tr>
</thead>
<tbody>
<tr>
<td>52</td>
<td>5832.1062454441</td>
<td>0.32584847709342</td>
<td>setosa</td>
</tr>
<tr>
<td>3.5956098612882</td>
<td>versicolor</td>
<td>versicolor</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>2205.8997420564</td>
<td>0.16809775183382</td>
<td>setosa</td>
</tr>
<tr>
<td>53.604305455093</td>
<td>versicolor</td>
<td>versicolor</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.079752356340823</td>
<td>1400.5304766943</td>
<td>setosa</td>
</tr>
<tr>
<td>1325.0715820345</td>
<td>versicolor</td>
<td>setosa</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>0.027765922281455</td>
<td>1098.717459793</td>
<td>setosa</td>
</tr>
<tr>
<td>1054.1646366836</td>
<td>versicolor</td>
<td>setosa</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>10243.200694846</td>
<td>6.3633234501984</td>
<td>virginica</td>
</tr>
<tr>
<td>3.6539363034276</td>
<td>virginica</td>
<td>virginica</td>
<td></td>
</tr>
<tr>
<td>150</td>
<td>9943.6666708225</td>
<td>8.8681276552651</td>
<td>virginica</td>
</tr>
<tr>
<td>2.2810345426083</td>
<td>virginica</td>
<td>virginica</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.0057680745330313</td>
<td>899.42153810855</td>
<td>setosa</td>
</tr>
<tr>
<td>869.97595458162</td>
<td>setosa</td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>35319.288016761</td>
<td>411.01353877088</td>
<td>virginica</td>
</tr>
<tr>
<td>0.025474879139092</td>
<td>virginica</td>
<td></td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>7135.0623567413</td>
<td>2.8102743005517</td>
<td>versicolor</td>
</tr>
<tr>
<td>0.12221517332065</td>
<td>versicolor</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(9 rows)

Related Functions

► category Analytics - Regression
► TANET_GROW

TANET_GROW - Build a tree-augmented network model

This stored procedure builds a tree-augmented network model from a tree-like Bayesian Network model.
The Bayesian Network model is augmented according to the classes of a nominal target column.

**Usage**

The TANET_GROW stored procedure has the following syntax:

```plaintext
TANET_GROW(NVARCHAR(ANY) paramString)
```

- **paramString**
  A comma-separated list of <parameter>=<value> entries using the parameters below.
  Type: NVARCHAR(ANY)

- **model**
  The tree-augmented Network model to build.
  Type: NVARCHAR(ANY)

- **inmodel**
  The name of the input Bayesian Network model.
  Type: NVARCHAR(ANY)

- **intable**
  The name of the table containing input. This should be the same input table as used in TBNET_GROW to build <inmodel>, otherwise it must contain the same columns.
  Type: NVARCHAR(256)

- **class**
  The nominal column of the input table representing the class.
  Type: NVARCHAR(128)

- **edgelabsort**
  if edgelabsort=yes then the left end of the edge will have a name lower in alphabetic order than the right one
  Type: NVARCHAR(ANY)
  Default: <none>

- **Returns**
  INT8 The number of correlations in the model.

**Details**

This procedure builds a tree-augmented network model, which is a set of tree-like Bayesian Networks such that the tree structure is developed for the whole dataset but the actual correlations are derived for a specific subset of the data, identified by the class variable. The model is stored as a set of edges, storing for each class the correlation between two columns and the mean and standard deviation of both columns.

The model can be applied using stored procedure TANET_APPLY.
Examples

CALL nza..TBNET_GROW('model=struc_iris, intable=nza..iris, incolumn=SEPALLength;SEPALWidth;PETALLength;PETALWidth,coldefrole=ignore');

CALL nza..TANET_GROW('model=plstruc_iris, intable=nza..iris, class=class, inmodel=struc_iris');

SELECT * FROM nza_meta_plstruc_iris_model ORDER BY grouped_on, varxname, varyname;

CALL nza..DROP_MODEL('model=struc_iris');

CALL nza..DROP_MODEL('model=plstruc_iris');

TBNET_GROW

-------------------------------------
Over node no4in TBNet_Grow process
(1 row)

TANET_GROW

-------------
9
(1 row)

GROUPED_ON | VARXNAME | VARXMEAN | VARXSTDDEV |
           | VARYNAME | VARYMEAN | VARYSTDDEV |
           | CORR     +-------------------+-------------------+-------------------+
-------------------+-------------------+-------------------+-------------------+
setosa     | PETALLength | 1.4938775510204 | 0.17351115943645 |
           | PETALWidth  | 0.248979959183673 | 0.10720950308168 |
           |            | 0.30630821115803 |
setosa     | PETALLength | 1.4938775510204 | 0.17351115943645 |
           | SEPALLength | 5.1081632653061 | 0.35248968721346 |
           |            | 0.26387409291868 |
setosa     | SEPALWidth  | 3.4877551020408 | 0.38102439795469 |
           | PETALLength | 1.4938775510204 | 0.17351115943645 |
           |            | 0.1766946286968 |
versicolor | PETALLength | 4.3469387755102 | 0.46991097723996 |
           | PETALWidth  | 1.3530612244898 | 0.1977526800454 |
           |            | 0.7866680885228 |
| SEPALLENGTH | 6.0571428571429 | 0.51617114706387 |
| versicolor | SEPALWIDTH | 2.8265306122449 | 0.31379832337841 | PETALLENGTH | 4.3469387755102 | 0.46991097723996 | 0.56052209169299 |
| virginica | SEPALWIDTH | 3.034693877551 | 0.32249663817264 | PETALLENGTH | 5.665306122449 | 0.665306122449 |

(9 rows)

DROP_MODEL
---------

(1 row)

DROP_MODEL
---------

(1 row)

**Related Functions**
- category Analytics - Regression
- CORRELATION1000MATRIX
- TANET_APPLY
- TBNET_GROW

**TBNET1G - Build a tree-like Bayesian Network model**

This stored procedure builds a tree-like Bayesian Network for continuous variables. A spanning tree is constructed joining all the variables on grounds of most strong correlations. This gives the
user an overview of most significant interrelations governing the whole set of variables.

Usage
The `TBNET1G` stored procedure has the following syntax:

```sql
TBNET1G(NVARCHAR(ANY) paramString)
```

- **paramString**
  - comma-separated list of `<parameter>=<value>` entries with parameters below
  - Type: NVARCHAR(ANY)

- **model**
  - the Bayesian Network model to build
  - Type: NVARCHAR(ANY)

- **intable**
  - the input table
  - Type: NVARCHAR(256)

- **includelist**
  - The input table columns with special properties, separated by a semi-colon (;).
  - Each column is followed by one or several of the following properties:
    - type: ':nom' (for nominal), ':cont' (for continuous). By default, all numerical types are continuous, other types are nominal
    - role: ':id', ':target', ':input', ':ignore'.
    - Type: NVARCHAR(ANY)
    - Default: <none>

- **coldeftype**
  - The default type of the input table columns. Valid values are 'nom' and 'cont'.
  - If the parameter is undefined, all numeric columns are considered continuous, other columns nominal.
  - Type: NVARCHAR(ANY)
  - Default: <none>

- **coldefrole**
  - The default role of the input table columns. Allowed values are 'input' and 'ignore'.
  - If the parameter is undefined, all columns are considered 'input' columns.
  - Type: NVARCHAR(ANY)
Default: <none>

▶ colPropertiesTable
The input table where column properties for the input table columns are stored. The format of this table is the output format of the nza..COLUMN_PROPERTIES() stored procedure.

If the parameter is undefined, the input table column properties is detected automatically.

(Remark: colPropertiesTable with "COLROLE" column with value 'objweight' is unsupported, i.e. same as 'ignore')

(Remark: colPropertiesTable with "COLWEIGHT" column with value '<wgt>' is unsupported, i.e. same as '1')

Type: NVARCHAR(256)
Default: <none>

▶ baseidx
the numeric id to be assigned to the first variable

Type: int4
Default: 777

▶ samplesize
the sample size to take if the number of records is too large

Type: int4
Default: 330000

▶ talk
if talk=yes then additional information on progress will be displayed

Type: NVARCHAR(ANY)
Default: <none>

▶ nocheck
if nocheck=yes then no exception is thrown when a column in <incolumn> does not exist

Type: NVARCHAR(ANY)
Default: <none>

▶ edgelabsort
if edgelabsort=yes then the left end of the edge will have a name lower in alphabetic order than the right one

Type: NVARCHAR(ANY)
Default: <none>

▲ Returns
NVARCHAR(3000) a termination message
Details

This stored procedure builds a Bayesian Network over the given input table columns. The model is stored as a set of edges, storing the correlation between two columns. A sampling approach is taken to overcome the computational complexity.

If parameter talk=yes, the stored procedure displays notices during the computation. This can be useful for a larger number of columns.

The model cannot be applied using stored procedure as a model built by TBNET_GROW.

Examples

CALL nza..TBNET1G('model=struc_iris, intable=nza..iris, incolumn=SEPALLENGTH;SEPALWIDTH;PETALLENGTH;PETALWIDTH, talk=yes,coldefrole=ignore');

SELECT varxname, varyname, corr::numeric(6,4) AS corr FROM nza_meta_struc_iris_model ORDER BY varxname, varyname;

CALL nza..DROP_MODEL('model=struc_iris');

TBNET1G

----------------------------------
Over node no4in TBNet1g process
(1 row)

| VARXNAME   |  VARYNAME   |  CORR  |
|-------------+-------------+--------|
| PETALLENGTH | PETALWIDTH  |  0.9628|
| PETALLENGTH | SEPALLENGTH |  0.8718|
| PETALLENGTH | SEPALWIDTH  | -0.4205|

(3 rows)

DROP_MODEL

---------

t
(1 row)

Related Functions

► category Analytics - Regression
► TBNET1G2P
TBNET1G2P - Build a tree-like Bayesian Network model

This stored procedure builds a tree-like Bayesian Network for continuous variables. A spanning tree is constructed joining all the variables on grounds of most strong correlations. This gives the user an overview of most significant interrelations governing the whole set of variables.

The stored procedure constructs the tree in an incremental manner. It calculates correlations on one set of variables, then on the other set of variables, then between variables of the 2 sets. The final model is obtained by joining the three sub-models.

Usage

The TBNET1G2P stored procedure has the following syntax:

```
TBNET1G2P NVARCHAR(ANY) paramString
```

Parameters

- **paramString**
  comma-separated list of <parameter>=<value> entries with parameters below
  Type: NVARCHAR(ANY)

- **model**
  the Bayesian Network model to build
  Type: NVARCHAR(ANY)

- **intable**
  the input table
  Type: NVARCHAR(256)

- **incolumn**
  The input table columns with special properties, separated by a semi-colon (;).
  Each column is followed by one or several of the following properties:
  - type: `:nom` (for nominal), `:cont` (for continuous). By default, all numerical types are continuous, other types are nominal
  (Remark: `:objweight` is unsupported, i.e. `:objweight` same as `:ignore`).
  (Remark: `:colweight(<wgt>)` is unsupported, i.e. `:colweight(<wgt>)` same as `:colweight(1)` same as `:input`).
  If the parameter is undefined, all columns of the input table have default properties. Note that this procedure only accepts continuous columns with role `:input`. Additionally, each column is followed by a colon (:) and either X or Y to distinguish the two sets of variables.
Type: NVARCHAR(ANY)
Default: <none>

- **coldeftype**
The default type of the input table columns. Valid values are 'nom' and 'cont'.
If the parameter is undefined, all numeric columns are considered continuous, other columns nominal.
Type: NVARCHAR(ANY)
Default: <none>

- **coldefrole**
The default role of the input table columns. Allowed values are 'input' and 'ignore'.
If the parameter is undefined, all columns are considered 'input' columns.
Type: NVARCHAR(ANY)
Default: <none>

- **colPropertiesTable**
The input table where column properties for the input table columns are stored. The format of this table is the output format of the nza..COLUMN_PROPERTIES() stored procedure.
If the parameter is undefined, the input table column properties is detected automatically.
(Remark: colPropertiesTable with "COLROLE" column with value 'objweight' is unsupported, i.e. same as 'ignore')
(Remark: colPropertiesTable with "COLWEIGHT" column with value '<wgt>' is unsupported, i.e. same as '1')
Type: NVARCHAR(256)
Default: <none>

- **baseidx**
the numeric id to be assigned to the first variable
Type: int4
Default: 777

- **talk**
if talk=yes then additional information on progress will be displayed
Type: NVARCHAR(ANY)
Default: <none>

- **nocheck**
if nocheck=yes then no exception is thrown when a column in <incolumn> does not exist
Type: NVARCHAR(ANY)
Default: <none>

- **edgelabsort**
if edgelabsort=yes then the left end of the edge will have a name lower in alphabetic order than the right one
Type: NVARCHAR(ANY)
Default: <none>

▲ Returns
NVARCHAR(3000) a termination message

Details
This stored procedure builds a Bayesian Network over the given input table columns of set X, then of set Y, and then between variables of X and Y. All three sub-models are merged into one Bayesian Network model. The model is stored as a set of edges, storing the correlation between two columns.

If parameter talk=yes, the stored procedure displays notices during the computation. This can be useful for a larger number of columns.

The model cannot be applied using stored procedure as a model built by TBNET_GROW.

Examples

CALL nza..TBNET1G2P('model=struc_iris, intable=nza..iris, incolumn=SEPALLENGTH:X;SEPALWIDTH:X;PETALLENGTH:Y;PETALWIDTH:Y,coldefrole=ignore');
SELECT varxname, varyname, corr::numeric(6,4) AS corr FROM nza_meta_struc_iris_model ORDER BY varxname, varyname;
CALL nza..DROP_MODEL('model=struc_iris');

TBNET1G2P
-------------
Over node no2 plus 2
(1 row)

<table>
<thead>
<tr>
<th>VARXNAME</th>
<th>VARYNAME</th>
<th>CORR</th>
</tr>
</thead>
<tbody>
<tr>
<td>PETALLENGTH</td>
<td>PETALWIDTH</td>
<td>0.9628</td>
</tr>
<tr>
<td>SEPALLENGTH</td>
<td>PETALLENGTH</td>
<td>0.8718</td>
</tr>
<tr>
<td>SEPALWIDTH</td>
<td>PETALLENGTH</td>
<td>-0.4205</td>
</tr>
</tbody>
</table>

(3 rows)

DROP_MODEL
----------
Related Functions

- category Analytics - Regression
- TBNET1G
- TBNET2G
- TBNET_GROW

TBNET2G - Build a tree-like Bayesian Network model in bi-partite mode

This stored procedure builds a tree-like Bayesian Network for continuous variables. A spanning tree is constructed joining all the variables on grounds of most strong correlations. This gives the user an overview of most significant interrelations governing the whole set of variables.

The stored procedure operates with two sets of variables and the resulting tree will be bi-partite. The correlations between variables within each set will not be calculated. This feature is useful when the two sets characterize distinct objects and only links between the objects are of interest.

Usage

The TBNET2G stored procedure has the following syntax:

```
TBNET2G(NVARCHAR(ANY) paramString)
```

Parameters

- **paramString**
  
  comma-separated list of <parameter>=<value> entries with parameters below

  Type: NVARCHAR(ANY)

- **model**
  
  the Bayesian Network model to build

  Type: NVARCHAR(ANY)

- **intable**
  
  the input table

  Type: NVARCHAR(256)

- **in columnName**
  
  The input table columns with special properties, separated by a semi-colon (;).

  Each column is followed by one or several of the following properties:

  - type: ':nom' (for nominal), ':cont' (for continuous). By default, all numerical types are continuous, other types are nominal
  
  - role: ':id', ':target', ':input', ':ignore'.

  (Remark: ':objweight' is unsupported, i.e. ':objweight' same as ':ignore').
(Remark: ':colweight(<wgt>)' is unsupported, i.e. ':colweight(<wgt>)' same as ':colweight(1)' same as ':input').

If the parameter is undefined, all columns of the input table have default properties. Note that this procedure only accepts continuous columns with role 'input'. Additionally, each column is followed by a colon (:) and either X or Y to distinguish the two sets of variables.

Type: NVARCHAR(ANY)
Default: <none>

► **coldeftype**
The default type of the input table columns. Valid values are 'nom' and 'cont'.

If the parameter is undefined, all numeric columns are considered continuous, other columns nominal.

Type: NVARCHAR(ANY)
Default: <none>

► **coldefrole**
The default role of the input table columns. Allowed values are 'input' and 'ignore'.

If the parameter is undefined, all columns are considered 'input' columns.

Type: NVARCHAR(ANY)
Default: <none>

► **colPropertiesTable**
The input table where column properties for the input table columns are stored. The format of this table is the output format of the nza..COLUMN_PROPERTIES() stored procedure.

If the parameter is undefined, the input table column properties is detected automatically.

(Remark: colPropertiesTable with "COLROLE" column with value 'objweight' is unsupported, i.e. same as 'ignore')

(Remark: colPropertiesTable with "COLWEIGHT" column with value '<wgt>' is unsupported, i.e. same as '1')

Type: NVARCHAR(256)
Default: <none>

► **baseidx**
The numeric id to be assigned to the first variable

Type: int4
Default: 777

► **talk**
If talk=yes then additional information on progress will be displayed
**Details**

This stored procedure builds a Bayesian Network over the given input table columns between set X and Y. The model is stored as a set of edges, storing the correlation between a column of set X and a column of set Y. A sampling approach is taken to overcome the computational complexity.

If parameter talk=yes, the stored procedure displays notices during the computation. This can be useful for a larger number of columns.

The model cannot be applied using stored procedure as a model built by TBNET_GROW.

**Examples**

```sql
CALL nza..TBNET2G('model=struc_iris, intable=nza..iris, incolumn=SEPALLENGTH:X;SEPALWIDTH:X;PETALLENGTH:Y;PETALWIDTH:Y, coldefrole=ignore');

SELECT varxname, varyname, corr::numeric(6,4) AS corr FROM nza_meta_struc_iris_model ORDER BY varxname, varyname;

CALL nza..DROP_MODEL('model=struc_iris');
```

```
TBNET2G

Over node no2 plus  2
(1 row)

<table>
<thead>
<tr>
<th>VARXNAME</th>
<th>VARYNAME</th>
<th>CORR</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEPALLENGTH</td>
<td>PETALLENGTH</td>
<td>0.8718</td>
</tr>
<tr>
<td>SEPALLENGTH</td>
<td>PETALWIDTH</td>
<td>0.8180</td>
</tr>
</tbody>
</table>
```
SEPALWIDTH | PETALLENGTH | -0.4205

(3 rows)

DROP_MODEL

------------
t

(1 row)

CALL nza..TBNET2G('model=struc_iris, intable=nza..iris, incolumn=SEPALLENGTH|SEPALWIDTH:X;PETALLENGTH|PETALWIDTH:Y,coldefrole=ignore');

SELECT varxname, varyname, corr::numeric(6,4) AS corr
FROM nza_meta_struc_iris_model ORDER BY varxname, varyname;

CALL nza..DROP_MODEL('model=struc_iris');

TBNET2G

-----------------------
Over node no2 plus 2
(1 row)

<table>
<thead>
<tr>
<th>VARXNAME</th>
<th>VARYNAME</th>
<th>CORR</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEPALLENGTH</td>
<td>PETALLENGTH</td>
<td>0.8718</td>
</tr>
<tr>
<td>SEPALLENGTH</td>
<td>PETALWIDTH</td>
<td>0.8180</td>
</tr>
<tr>
<td>SEPALWIDTH</td>
<td>PETALLENGTH</td>
<td>-0.4205</td>
</tr>
</tbody>
</table>

(3 rows)

DROP_MODEL

------------
t

(1 row)
Related Functions

- category Analytics - Regression
- TBNET1G
- TBNET1G2P
- TBNET_GROW

**TBNET_APPLY - Apply a tree-like Bayesian Network model**

This procedure applies a tree-like Bayesian model to generate regression predictions for a dataset.

**Usage**

The TBNET_APPLY stored procedure has the following syntax:

```sql
TBNET_APPLY(NVARCHAR(ANY) paramString)
```

- **paramString**
  A comma-separated list of `<parameter>=<value>` entries using the parameters below.
  Type: NVARCHAR(ANY)

- **model**
  The Bayesian Network model to apply.
  Type: NVARCHAR(ANY)

- **intable**
  The name of the table containing input.
  Type: NVARCHAR(256)

- **id**
  The column of the input table that identifies a unique instance ID.
  Type: NVARCHAR(128)

- **target**
  The model variable to be predicted.
  Type: NVARCHAR(128)

- **outtable**
  The name of the output table where the predictions are to be stored.
  Type: NVARCHAR(256)

- **type**
  The type of prediction to be made. Valid values are best (most correlated neighbor), neighbors (weighted prediction of neighbors), and nn-neighbors (non null neighbors).
  Type: NVARCHAR(ANY)
Default: best

▲ Returns
NVARCHAR(3000) A termination message.

Details
This procedure predicts values of a continuous variable using a Bayesian Network model generated by TBNET_GROW. Three different approaches may be exploited:

(1) type=best: most strongly correlated neighbor. In the Bayesian network, the target variable's direct neighbors are identified. The neighbor is selected that has the strongest correlation to the target variable. The input table column value corresponding to this neighbor is then used for prediction.

(2) type=neighbors: all direct neighbors. In the Bayesian network, the target variable's direct neighbors are identified. The input table column values corresponding to these neighbors is then used for prediction. Their contribution is weighted by their squared correlation in the model.

(3) type=nn-neighbors: all non-null direct neighbors. The prediction is made as in (2), but taking into account non-null column values of the direct neighbors only. Note at this point that no prediction at all is made in case (2) when there are any missing values among the neighbors of the target variable.

The output table contains following columns: <id>, <target>_pred.

Examples

CALL nza..TBNET_GROW('model=struc_iris,
intable=nza..iris,
incolumn=SEPALLENGTH;SEPALWIDTH;PETALLENGTH;PETALWIDTH');

CALL nza..TBNET_APPLY('model=struc_iris,
intable=nza..iris, id=id, target=PETALLENGTH,
outtable=iris_pred');

SELECT a.*, b.petallength, b.class FROM iris_pred a,
nza..iris b WHERE a.id=b.id AND mod(a.id,50)<3 ORDER BY id;

CALL nza..DROP_MODEL('model=struc_iris');

CALL nza..DROP_TABLE('iris_pred');

TBNET_GROW
-------------------------------------
Over node no4in TBNet_Grow process
(1 row)

TBNET_APPLY
Done
(1 row)

<table>
<thead>
<tr>
<th>ID</th>
<th>PETALLENGTH_PRED</th>
<th>PETALLENGTH</th>
<th>CLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.5357492071881</td>
<td>1.4</td>
<td>setosa</td>
</tr>
<tr>
<td>2</td>
<td>1.5357492071881</td>
<td>1.4</td>
<td>setosa</td>
</tr>
<tr>
<td>50</td>
<td>1.5357492071881</td>
<td>1.4</td>
<td>setosa</td>
</tr>
<tr>
<td>51</td>
<td>4.2068115750528</td>
<td>4.7</td>
<td>versicolor</td>
</tr>
<tr>
<td>52</td>
<td>4.4294001057082</td>
<td>4.5</td>
<td>versicolor</td>
</tr>
<tr>
<td>100</td>
<td>3.9842230443974</td>
<td>4.1</td>
<td>versicolor</td>
</tr>
<tr>
<td>101</td>
<td>6.6552854122622</td>
<td>6</td>
<td>virginica</td>
</tr>
<tr>
<td>102</td>
<td>5.3197542283298</td>
<td>5.1</td>
<td>virginica</td>
</tr>
<tr>
<td>150</td>
<td>5.0971656976744</td>
<td>5.1</td>
<td>virginica</td>
</tr>
</tbody>
</table>

(9 rows)

DROP_MODEL

------------
t
(1 row)

DROP_TABLE

------------
t
(1 row)

Related Functions
► category Analytics - Regression
► TBNET_GROW

TBNET_GROW - Build a tree-like Bayesian Network model
This stored procedure builds a tree-like Bayesian Network for continuous variables. A spanning tree is constructed joining all the variables on grounds of most strong correlations. This gives the user an overview of most significant interrelations governing the whole set of variables.

**Usage**

The TBNET_GROW stored procedure has the following syntax:

```
TBNET_GROW(NVARCHAR(ANY) paramString)
```

**Parameters**

- **paramString**
  comma-separated list of `<parameter>=<value>` entries with parameters below
  
  Type: NVARCHAR(ANY)

- **model**
  the Bayesian Network model to build
  
  Type: NVARCHAR(ANY)

- **itable**
  the input table
  
  Type: NVARCHAR(256)

- **incolumn**
  The input table columns with special properties, separated by a semi-colon (;).
  Each column is followed by one or several of the following properties:
  - type: `:nom` (for nominal), `:cont` (for continuous). By default, all numerical types are continuous, other types are nominal

  (Remark: `:objweight` is unsupported, i.e. `:objweight` same as `:ignore`).
  
  (Remark: `:colweight(<wgt>)` is unsupported, i.e. `:colweight(<wgt>)` same as `:colweight(1)` same as `:input`).

  If the parameter is undefined, all columns of the input table have default properties.
  Note that this procedure only accepts continuous columns with role `input`.
  
  Type: NVARCHAR(ANY)
  Default: <none>

- **coldeftype**
  The default type of the input table columns. Valid values are 'nom' and 'cont'.
  If the parameter is undefined, all numeric columns are considered continuous, other columns nominal.
  
  Type: NVARCHAR(ANY)
  Default: <none>
► coldefrole
The default role of the input table columns. Allowed values are 'input' and 'ignore'.
If the parameter is undefined, all columns are considered 'input' columns.
Type: NVARCHAR(ANY)
Default: <none>

► colPropertiesTable
The input table where column properties for the input table columns are stored. The format of this table is the output format of the nza..COLUMN_PROPERTIES() stored procedure.
If the parameter is undefined, the input table column properties is detected automatically.
(Remark: colPropertiesTable with "COLROLE" column with value 'objweight' is unsupported, i.e. same as 'ignore')
(Remark: colPropertiesTable with "COLWEIGHT" column with value '<wgt>' is unsupported, i.e. same as '1')
Type: NVARCHAR(256)
Default: <none>

► baseidx
the numeric id to be assigned to the first variable
Type: int4
Default: 777

► samplesize
the sample size to take if the number of records is too large
Type: int4
Default: 330000

► talk
if talk=yes then additional information on progress will be displayed
Type: NVARCHAR(ANY)
Default: <none>

► sizewarn
if sizewarn=yes then no exception is thrown when there are less records than 3 times the number of columns. Instead, a notice is displayed and the stored procedure returns 'sizewarn'.
Type: NVARCHAR(ANY)
Default: <none>

► edgelabsort
if edgelabsort=yes then the left end of the edge will have a name lower in alphabetic order than the right one
Type: NVARCHAR(ANY)
Default: <none>

▲ Returns
NVARCHAR(3000) a termination message

Details
This stored procedure builds a Bayesian Network over the given input table columns. The model is stored as a set of edges, storing the correlation between two columns and the mean and standard deviation of both columns.

A sampling approach is taken to overcome the computational complexity. If parameter talk=yes, the stored procedure displays notices during the computation. This can be useful for a larger number of columns.

The model can be applied using stored procedure TBNET_APPLY.

Examples

CALL nza..TBNET_GROW('model=struc_iris,
intable=nza..iris,
incolumn=SEPALLENGTH;SEPALWIDTH;PETALLENGTH;PETALWIDTH,coldefrole=ignore');

SELECT varxname, varyname, corr::numeric(6,4) AS corr
FROM nza_meta_struc_iris_model ORDER BY varxname, varyname;

CALL nza..DROP_MODEL('model=struc_iris');

<table>
<thead>
<tr>
<th>VARXNAME</th>
<th>VARYNAME</th>
<th>CORR</th>
</tr>
</thead>
<tbody>
<tr>
<td>PETALLENGTH</td>
<td>PETALWIDTH</td>
<td>0.9628</td>
</tr>
<tr>
<td>PETALLENGTH</td>
<td>SEPALWIDTH</td>
<td>-0.4205</td>
</tr>
<tr>
<td>SEPALLENGTH</td>
<td>PETALLENGTH</td>
<td>0.8718</td>
</tr>
</tbody>
</table>

(3 rows)

DROP_MODEL

(1 row)
CALL nza..TBNET_GROW('model=struc_iris, intable=nza..iris, incolumn=SEPALLENGTH;SEPALWIDTH;PETALLENGTH;PETALWIDTH,coldefrole=ignore,edgelabsort=yes');

SELECT varxname, varyname, corr::numeric(6,4) AS corr FROM nza_meta_struc_iris_model ORDER BY varxname, varyname;

CALL nza..DROP_MODEL('model=struc_iris');

-----------------------------
Over node no4in TBNet_Grow process
(1 row)

<table>
<thead>
<tr>
<th>VARXNAME</th>
<th>VARYNAME</th>
<th>CORR</th>
</tr>
</thead>
<tbody>
<tr>
<td>PETALLENGTH</td>
<td>PETALWIDTH</td>
<td>0.9628</td>
</tr>
<tr>
<td>PETALLENGTH</td>
<td>SEPALWIDTH</td>
<td>-0.4205</td>
</tr>
<tr>
<td>PETALLENGTH</td>
<td>SEPALLENGTH</td>
<td>0.8718</td>
</tr>
</tbody>
</table>

(3 rows)

DROP_MODEL
-------------
t
(1 row)

Related Functions
- category Analytics - Regression
- TBNET1G
- TBNET2G
- TBNET APPLY

TIMESERIES - Predict future values for Time Series

This stored procedure predicts future values of series of timed numeric values
Usage

The TIMESERIES stored procedure has the following syntax:

TIMESERIES(NVARCHAR(ANY) paramString)

Parameters

- paramString
  comma-separated list of <parameter>=<value> entries with parameters below
  Type: NVARCHAR(ANY)

- model
  the name of the Time Series model to build
  Type: NVARCHAR(ANY)

- intable
  the input table
  Type: NVARCHAR(256)

- time
  the input table column which define an order on the numeric values
  Type: NVARCHAR(128)

- target
  the input table column which contains the numeric values
  Type: NVARCHAR(128)

- by
  the input table column which uniquely identifies a serie of values. If not specified, all numeric values belong to only one time series.
  Type: NVARCHAR(128)
  Default: <none>

- desctable
  the optional input table containing the name and descriptions of the time series. The table must contain following columns: <by>, 'NAME' NVARCHAR(ANY), 'DESCRIPTION' NVARCHAR(ANY). If not specified, the series do not have a name or a description.
  Type: NVARCHAR(ANY)
  Default: <none>

- algorithm
  the time series algorithm to use. Allowed values are: ExponentialSmoothing, ARIMA, SeasonalTrendDecomposition, SpectralAnalysis.
  Type: VARCHAR(32)
  Default: ExponentialSmoothing

- interpolationmethod

the interpolation method. Allowed values are: linear, cubicspline, exponentialspline.
Type: NVARCHAR(32)
Default: linear

► from
the value of column time to start the analysis from. If not specified, the analysis starts from the
first value of the time series in the input table.
Type: same as <time>
Default: <none>

► to
the value of column time to stop the analysis at. If not specified, the analysis stops at the last
value of the time series in the input table.
Type: same as <time>
Default: <none>

► forecasthorizon
the value of column time until which to predict. This parameter is not allowed for
algorithm=SpectralAnalysis. If not specified, the algorithm determines itself until which time it
predicts values.
Type: NVARCHAR(ANY)
Default: <none>

► forecasttimes
list of semi-column separated values of column time to predict at. This parameter is not allowed
for algorithm=SpectralAnalysis. If not specified, the times to predict values at is determined by
the algorithm.
Type: NVARCHAR(ANY)
Default: <none>

► trend
the trend type for algorithm=ExponentialSmoothing. Allowed values are: N (none), A (addictive), DA (damped additive), M (multiplicative), DM (damped multiplicative). If not specified, the trend type is determined by the algorithm.
Type: CHAR(2)
Default: <none>

► seasonality
the seasonality type for algorithm=ExponentialSmoothing. Allowed values are: N (none), A (additive), M (multiplicative). If not specified, the seasonality type is determined by the al-
gorithm.
Type: CHAR(1)
Default: <none>

► period
the seasonality period. This parameter is not allowed for algorithm=SpectralAnalysis. If not spe-
cified, the seasonality period is determined by the algorithm. If set to 0, no seasonality period
will be considered by the algorithm.
Type: DOUBLE
Default: <none>
Min: 0

- **unit**
  the seasonality period unit. This parameter is not allowed for algorithm=SpectralAnalysis. This parameter must be specified if the parameter period is specified and the <time> column is of type date, time or timestamp. Otherwise, it must not be specified. Allowed values are: ms, s, min, h, d, wk, qtr, q, a, y.
Type: NVARCHAR(ANY)
Default: <none>

- **p**
or p< the parameter p for algorithm=ARIMA, either equal to or below specified value. If not specified, the algorithm will determine its best value automatically.
Type: SMALLINT
Default: <none>

- **d**
or d< the parameter d for algorithm=ARIMA, either equal to or below specified value. If not specified, the algorithm will determine its best value automatically.
Type: SMALLINT
Default: <none>

- **q**
or q< the parameter q for algorithm=ARIMA, either equal to or below specified value. If not specified, the algorithm will determine its best value automatically.
Type: SMALLINT
Default: <none>

- **SP**
or SP< the seasonal parameter SP for algorithm=ARIMA, either equal to or below specified value. If not specified, the algorithm will determine its best value automatically.
Type: SMALLINT
Default: <none>

- **SD**
or SD< the seasonal parameter SD for algorithm=ARIMA, either equal to or below specified value. If not specified, the algorithm will determine its best value automatically.
Type: SMALLINT
Default: <none>

- **SQ**
or SQ< the seasonal parameter SQ for algorithm=ARIMA, either equal to or below specified value. If not specified, the algorithm will determine its best value automatically.

Type: SMALLINT
Default: <none>

► **outtable**

the output table containing predicted future values. This parameter is not allowed for algorithm=SpectralAnalysis. If not specified, no output table is written out.

Type: NVARCHAR(256)
Default: <none>

► **seasadjtable**

the output table containing seasonally adjusted values. This parameter is not allowed for algorithm=SpectralAnalysis or algorithm=ARIMA. If not specified, no output table is written out.

Type: NVARCHAR(256)
Default: <none>

▲ Returns

BIGINT the number of time series processed

**Details**

A time series model is built by analyzing series of timed numeric values, and is applied immediately for predicting future values. The model itself is stored but not really needed any more (except for understanding the predicted values).

If specified, a table <outtable> is additionally created with the following columns: <by>, <time>, forecast, standarderror. The table contains the forecast values for future time points of the time series identified by <by>. For each prediction, the standarderror value indicates a confidence interval around the forecast value.

If specified, a table <seasadjtable> is additionally created with the following columns: <by>, <time>, adjusted. The values in column <target> of the input table are seasonally adjusted and then copied into this table, with the values of columns <by> and <time>.

**Examples**

```sql
CALL nza..TIMESERIES('model=curves_ts, intable=nza..curves, outtable=curves_out, time=x, target=y, by=curve, algorithm=exponentialsmoothing');
SELECT * FROM curves_out WHERE curve='quadratic' ORDER BY x;
CALL nza..DROP_MODEL('model=curves_ts');
CALL nza..DROP_TABLE('curves_out');
TIMESERIES
----------
6
(1 row)
```
CURVE | X | Y | STANDARDERROR
-------------------+----+-----------------+------------------
quadratic | 51 | 2547.9999996322 | 2.0889322555742
quadratic | 52 | 2645.9999995728 | 6.2792178687503
quadratic | 53 | 2743.9999995133 | 12.558435352633
quadratic | 54 | 2841.9999994539 | 20.976177536159
(4 rows)

DROP_MODEL
----------
t
(1 row)

DROP_TABLE
----------
t
(1 row)

Related Functions
- category Analytics - Time Series
- PRINT_MODEL

TPR - True Positive Rate from a Confusion Matrix

This stored procedure calculates the True Positive Rate of a class from a confusion matrix

Usage

The TPR stored procedure has the following syntax:

TPR(NVARCHAR(ANY) paramString)

- Parameters
  - paramString
comma-separated list of <parameter>=<value> entries with parameters below

Type: NVARCHAR(ANY)

► **matrixTable**
the confusion matrix table
Type: NVARCHAR(256)

► **class**
the class in the confusion matrix table to calculate the True Positive Rate on
Type: NVARCHAR(ANY)

▲ Returns
DOUBLE the True Positive Rate

**Details**

This stored procedure calculates the True Positive Rate of a class from a confusion matrix. The True positive rate is the ratio of the number of correctly classified records into the given class divided by the number of real values that are in this class.

The confusion matrix table has the following columns: real, prediction, cnt. The column cnt contains the frequency of making a given prediction for the given real value.

**Examples**

```sql
CALL nza..TRAIN_TEST('modelType=dectree, model=iris_c45, traintable=nza..iris_train, testtable=nza..iris_test, target=class, id=id, outtable=iris_pred, minsplit=2');
CALL nza..CONFUSION_MATRIX('intable=nza..iris_test, resulttable=iris_pred, id=id, resultid=id, target=class, resulttarget=class, matrixTable=confMatrix');
CALL nza..TPR('matrixTable=confMatrix, class=virginica');
CALL nza..DROP_MODEL('model=iris_c45');
CALL nza..DROP_TABLE('iris_pred');
CALL nza..DROP_TABLE('confMatrix');
```

**TRAIN_TEST**

```
0.918919
(1 row)
```

**CONFUSION_MATRIX**

```
5
(1 row)
```
**TRAIN_TEST - Build and evaluate a Classification model on given training and testing sets**

This is the simple technique to estimate quality and performance of a predictive model

**Usage**

The TRAIN_TEST stored procedure has the following syntax:
TRAIN_TEST(NVARCHAR(ANY) paramString)

Parameters

- paramString
  comma-separated list of <parameter>=<value> entries with parameters below
  Type: NVARCHAR(ANY)

- modelType
  the name of the procedure that builds a Classification model (e.g. naivebayes, dectree, knn)
  Type: NVARCHAR(ANY)

- model
  the name of the Classification model to build
  Type: NVARCHAR(ANY)

- traintable
  the input table to build the model onto
  Type: NVARCHAR(256)

- testtable
  the input table to evaluate the model quality onto
  Type: NVARCHAR(256)

- id
  the input table column identifying a unique instance id
  Type: NVARCHAR(128)

- target
  the input table column representing the class
  Type: NVARCHAR(128)

- outtable
  the output table where the predictions will be stored
  Type: NVARCHAR(256)

Returns

FLOAT prediction accuracy and -1 if accuracy cannot be calculated

Details

This stored procedure builds a Classification model on <traintable>. The model is saved to the database in a set of tables and registered in the database model metadata. Use the Model Management functions to further manipulate the model or access the model tables.

The prediction quality of the model is then evaluated on <testtable>.

All parameters specific to a the given Classification stored procedure can be defined too: they will be used when building and testing the model.

A table <outtable> is created with the following columns: id, class. The column id matches the <id> column from intable and class is the predicted class label. The size of this table is the same as the size of <intable>. 
Examples

CALL nza..TRAIN_TEST('modelType=dectree,
   traintable=nza..iris_train, testtable=nza..iris_test,
   model=iris_c45, target=class, id=id,
   outtable=iris_pred');

CALL nza..DROP_MODEL('model=iris_c45');

CALL nza..DROP_TABLE('iris_pred');

TRAIN_TEST
----------------------
0.918919
(1 row)

DROP_MODEL
----------------------
(1 row)

DROP_TABLE
----------------------
(1 row)

Related Functions

► category Analytics - Classification
► SPLIT_DATA
► CROSS_VALIDATION
► PERCENTAGE_SPLIT
► DECTREE
► NAIVEBAYES
► KNN

TWOSTEP - Build a TwoStep Clustering model

This stored procedure builds a TwoStep Clustering model that first distributes the input data into a
hierarchical tree structure according to the distance between the data records, then reduces the
tree into k clusters. A second pass over the data associates the input data records to the next
cluster.
Usage

The TWOSTEP stored procedure has the following syntax:

► **TWOSTEP**(NVARCHAR(ANY) paramString)

▲ **Parameters**

► **paramString**

comma-separated list of <parameter>=<value> entries with parameters below

Type: NVARCHAR(ANY)

► **model**

the name of the TwoStep Clustering model to build

Type: NVARCHAR(ANY)

► **intable**

the input table

Type: NVARCHAR(392)

► **outtable**

the output table where clusters are assigned to each input table record

Type: NVARCHAR(261)

Default: <none>

► **id**

the input table column identifying a unique instance id

Type: NVARCHAR(128)

► **target**

the input table column representing a class or a value to predict, this column is ignored by the TwoStep Clustering algorithm.

Type: NVARCHAR(128)

Default: <none>

► **distance**

the distance function. Allowed values are: euclidean, norm_euclidean, loglikelihood.

Type: NVARCHAR(ANY)

Default: loglikelihood

► **k**

the number of clusters. If k is 0 or less, the procedure determines the optimal number of clusters.

Type: INTEGER

Default: 0

► **maxk**

the maximum number of clusters that can be determined automatically. If k is bigger than 0, this parameter is ignored.

Type: INTEGER
Default: 20
Min: 2

► incol
the input table columns with special properties, separated by a semi-colon (;).
Each column is followed by one or several of the following properties:
- its type: ':nom' (for nominal), ':cont' (for continuous). Per default, all numerical types are continuous, other types are nominal.
- its role: ':id', ':target', ':input', ':ignore'.
(Remark: ':objweight' is unsupported i.e. ':objweight' same as ':ignore').
- its weight: ':colweight(<wgt>)' where <wgt> is a numeric value of the weight.
If the parameter is undefined, all columns of the input table have default properties.
Type: NVARCHAR(ANY)
Default: <none>

► coldeftype
default type of the input table columns. Allowed values are 'nom' and 'cont'.
If the parameter is undefined, all numeric columns are considered continuous, other columns nominal.
Type: NVARCHAR(ANY)
Default: <none>

► coldefrole
default role of the input table columns. Allowed values are 'input' and 'ignore'.
If the parameter is undefined, all columns are considered 'input' columns.
Type: NVARCHAR(ANY)
Default: <none>

► colPropertiesTable
the input table where column properties for the input table columns are stored. The format of this table is the output format of stored procedure nza..COLUMN_PROPERTIES().
If the parameter is undefined, the input table column properties will be detected automatically.
(Remark: colPropertiesTable with "COLROLE" column with value 'objweight' is unsupported, i.e. same as 'ignore')
Type: NVARCHAR(256)
Default: <none>

► statistics
flags indicating which statistics to collect. Allowed values are: none, columns, values:n, all.
Regardless of the value of the parameter statistics, all statistics are gathered since they are needed to call PREDICT_TWOSTEP on this model. If statistics=none or statistics=columns, the importance of the attributes is not calculated. If statistics=none, statistics=columns or statistics=all, up to 100 discrete values are gathered.

If statistics=values:n with n a positive number, up to <n> column value statistics are collected:
- If a nominal column contains more than <n> values, only the <n> most frequent column statistics are kept.
- If a numeric column contains more than <n> values, the values will be discretized and the statistics will be collected on the discretized values.

Indicating statistics=all is equal to statistics=values:100.

Type: NVARCHAR(ANY)
Default: 'none'

► **bins**
the average number of bins for numerical statistics with more than <n> values.
Type: INTEGER
Default: 10
Min: 2

► **randseed**
the random generator seed
Type: INTEGER
Default: 12345

► **distancethreshold**
the threshold under which 2 data records can be merged into one cluster during the first pass. If not set, the distance threshold is calculated automatically.
Type: DOUBLE
Default: <none>
Min: 0.0

► **distancethresholdfactor**
the factor used to calculate the distance threshold automatically. The distance threshold is then the median distance value minus distancethresholdfactor times the interquartile distance (or the minimum distance if this value is below it). If distancethreshold is set, this parameter is ignored.
Type: DOUBLE
Default: 2.0

► **epsilon**
the value to be used as global variance of all continuous fields for the loglikelihood distance. If the value is 0.0 or less, the global variance is calculated for each continuous field. If distance is not loglikelihood, this parameter is ignored.
Type: DOUBLE
Default: 0.0

- **nodecapacity**
  the branching factor of the internal tree used in pass 1. Each node can have up to no-decacity subnodes.
  
  Type: INTEGER
  Default: 6
  Min: 2

- **leafcapacity**
  the number of clusters per leaf node in the internal tree used in pass 1.
  
  Type: INTEGER
  Default: 8
  Min: 2

- **maxleaves**
  the maximum number of leaf nodes in the internal tree used in pass 1. When the tree contains maxleaves leaf nodes, the following data records are aggregated into the existing clusters.
  
  Type: INTEGER
  Default: 1000
  Min: max(10,k)

- **outlierfraction**
  the fraction of the records to be considered as outlier in the internal tree used in pass 1. Clusters containing less than outlierfraction times the mean number of data records per cluster are removed.
  
  Type: DOUBLE
  Default: 0.0
  Min: 0.0
  Max: 1.0

▲ Returns
BIGINT the number of generated clusters

**Details**

This stored procedure builds a TwoStep Clustering model. The model is saved to the database in a set of tables and registered in the database model metadata. Use the Model Management functions to further manipulate the model or access the model tables.

The Clustering model is built in two steps:

- The input data records are distributed into a balanced tree according to their distance to the data records already in the tree node. The tree size is limited: outliers are removed and similar tree nodes are merged. Then k clusters are determined out of the tree.
- The input data records are assigned again to the nearest of the k clusters.

The output table `<outtable>` is created with following columns: id, cluster_id, distance. The id column matches the `<id>` column of the input table. Each input table record is associated with a cluster, where the distance from the record to the cluster is the smallest. The cluster ID and the distance to the cluster are given in the columns cluster_id and distance.

**Examples**

```sql
CALL nza..TWOSTEP('model=adult_mdl, intable=nza..adult, id=id, target=income');

CALL nza..DROP_MODEL('model=adult_mdl');
```

**TWOSTEP**

```
6
(1 row)
```

**DROP_MODEL**

```
t
(1 row)
```

**Related Functions**

- category Analytics - Clustering
- LIST_MODELS

**UNITABLE - Univariate Frequencies**

This stored procedure creates a univariate frequency table for one column of the input table.

**Usage**

The UNITABLE stored procedure has the following syntax:

```sql
UNITABLE(NVARCHAR(ANY) paramString)
```

**Parameters**

- `<paramString>`
  - comma-separated list of `<parameter>=<value>` entries with parameters below
  - Type: NVARCHAR(ANY)

- `<intable>`
the input table or view
Type: NVARCHAR(256)

► **incolumn**
the input table column
Type: NVARCHAR(128)

► **outtable**
the output table where the univariate frequencies are written to
Type: NVARCHAR(ANY)

▲ Returns
INTEGER the number of univariate frequencies calculated

**Details**
This stored procedure counts the frequency of all distinct values for one column of the input table.
The univariate frequencies are stored in the output table with following columns: `<incolumn>`,
count, freq, cum. The column count counts the number of input table records havinf the given
value of `<incolumn>`, the column freq give the relative frequency (percentage) of these records,
and the column cum cumulates the frequencies.

**Examples**

```
CALL nza..UNITABLE('intable=nza..winequality,incolumn=quality,
          outtable=gdm_ft');
SELECT * FROM gdm_ft ORDER BY quality;
CALL nza..DROP_TABLE('gdm_ft');
UNITABLE
----------
    7
(1 row)

QUALITY | COUNT | FREQ  |  CUM
---------+-------+-------+--------
     3 |    20 |  0.41 |   0.41
     4 |   163 |  3.33 |   3.74
     5 |  1457 | 29.75 |  33.48
     6 |  2198 | 44.88 |  78.36
     7 |   880 | 17.97 |  96.33
```
Related Functions

- category Analytics - Statistics
- BITABLE
- HIST

**VERIFY_ARULE - Verify if data is prepared to build an Association Rules model**

This stored procedure is used to verify whether the given dataset is properly prepared for the ARULE algorithm to run on it.

**Usage**

The VERIFY_ARULE stored procedure has the following syntax:

```
VERIFY_ARULE(NVARCHAR(ANY) paramString)
```

**Parameters**

- `paramString`
  A comma-separated list of `<parameter>=<value>` entries using the parameters below.
  Type: NVARCHAR(ANY)

- `intable`
  The name of the table containing the input.
  Type: NVARCHAR(256)

- `tid`
  The column of the input table that identifies transactions.
  Type: NVARCHAR(128)
  Default: tid

- `item`
  The column of the input table that identifies items in transactions.
  Type: NVARCHAR(128)
Default: item

▲ Returns
   BOOLEAN TRUE if the dataset is properly prepared; otherwise FALSE.

Details
This procedure checks the data as follows:
- no NULL transaction or item values
- no duplicate items in a transaction
- no negative item values or item values bigger than 100.000.000

Examples
   CALL nza..VERIFY_ARULE('intable=nza..retail');
   VERIFY_ARULE
       ---------------
       t
       (1 row)

Related Functions
▶ category Analytics - Association Rules
▶ Error: Reference source not found
▶ ARULE

WACC - Weighted Classification accuracy
This stored procedure calculates the weighted Classification accuracy, i.e. the weighted ratio of correctly classified predictions

Usage
The WACC stored procedure has the following syntax:

▶ WACC(NVARCHAR(ANY) paramString)
   ▲ Parameters
   ◀ paramString
      comma-separated list of <parameter>=<value> entries with parameters below
      Type: NVARCHAR(ANY)
   ▶ intable
the input table containing real values
Type: NVARCHAR(256)
► id
the input table column in <intable> identifying a unique instance id
Type: NVARCHAR(128)
► target
the input table column in <intable> containing real values
Type: NVARCHAR(128)
► resulttable
the input table containing predicted values
Type: NVARCHAR(256)
► resultid
the input table column in <resulttable> identifying a unique instance id
Type: NVARCHAR(128)
  Default: id
► resulttarget
the input table column in <resulttable> containing predicted values
Type: NVARCHAR(128)
  Default: class
▲ Returns
  DOUBLE the weighted Classification accuracy

Details
This stored procedure calculates the weighted ratio of correctly classified predictions among the total number of predictions. This is done by comparing the predictions made when applying a Classification model onto data, and the real values for this data.

Each class is assigned the same weight in calculating the ratio, independently of the real frequency of the class in the input data. This is useful when classes are not equally frequent.

Examples

```
CALL nza..TRAIN_TEST('modelType=dectree, model=iris_c45, traintable=nza..iris_train, testtable=nza..iris_test, target=class, id=id, outtable=iris_pred, mnsplit=2');
CALL nza..WACC('intable=nza..iris_test, resulttable=iris_pred, id=id, target=class');
CALL nza..DROP_MODEL('model=iris_c45');
CALL nza..DROP_TABLE('iris_pred');
```
IBM Netezza In-Database Analytics Reference Guide

Related Functions
► category Analytics - Diagnostic Measures
► CERROR
► ACC

WILCOXON_TEST - Wilcoxon paired sample difference test

This procedure executes the Wilcoxon test on paired numeric input columns to evaluate the significance of the difference of their mean values

Usage
The WILCOXON_TEST stored procedure has the following syntax:

► WILCOXON_TEST(NVARCHAR(ANY) paramString)
  ▲ Parameters
  ► paramString
    comma-separated list of <parameter>=<value> entries with parameters below
    Type: NVARCHAR(ANY)
  ► intable
    the input table
Type: NVARCHAR(256)

► incolumn
the two numeric input table columns, separated by a semicolon (;)
Type: NVARCHAR(ANY)

► by
the input table column which splits the data into groups for which the operation is to be performed. If specified, an output table must be specified too.
Type: NVARCHAR(128)
Default: <none>

► outtable
the output table to write the Wilcoxon statistics into. This parameter is required if parameter by is specified. Otherwise the parameter is ignored.
Type: NVARCHAR(ANY)
Default: <none>

▲ Returns
NVARCHAR(200) The test results or the number of rows in the output table. The test result can contain following results: sStat, wStat, zStat, noItem, pp, lower. See the output table column description for more information on them.

Details
This stored procedure executes the Wilcoxon test on two numeric columns, either in the whole input table or within the groups defined in the column specified by parameter <by>. The Wilcoxon test compares in non-parametric manner two variables to state if they are different in their mean value. This is indicated by the component of the output called pp.

The output table is created with following columns: n, sstat, wstat, zstat, pp, lower. If the parameter by is specified, an additional column <by> is added to indicate for which group the MWW test has been calculated. If pp < 0.05, then one of the two columns tends to have larger values than the other. The class which tends to have lower-ranked values is indicated.

Examples

CALL nza..WILCOXON_TEST('intable=nza..winequality, incolumn=VOLATILE_ACIDITY;CITRIC_ACID');

WILCOXON_TEST

--------------------------------------
sStat= 3285041 wStat= 8524620 zStat= 25.520776487317 noItem= 4898 pp= 0 (lower: "VOLATILE_ACIDITY")
(1 row)
CALL nza..WILCOXON_TEST('intable=nza..iris,
incolumn=sepalwidth;petalwidth');

WILCOXON_TEST

---------------------------------------------------------
-----------------
sStat= 11085 wStat= -1 zStat= -1 noItem= 150 pp= 0
(lower: "SEPALWIDTH")
(1 row)

CALL nza..WILCOXON_TEST('intable=nza..winequality,
incolumn=VOLATILE_ACIDITY;CITRIC_ACID, by=quality,
outtable=Result7');
SELECT quality, pp, lower FROM Result7 ORDER BY quality;
CALL nza..DROP_TABLE('Result7');

WILCOXON_TEST

---------------

7
(1 row)

| QUALITY | PP    | LOWER           |
|---------+-------+-----------------|
| 3       | 0.46282003479431 | "VOLATILE_ACIDITY" |
| 4       | 0.0055011068340939 | "CITRIC_ACID"     |
| 5       | 1.1102230246252e-16 | "VOLATILE_ACIDITY" |
| 6       | 0     | "VOLATILE_ACIDITY" |
| 7       | 0     | "VOLATILE_ACIDITY" |
| 8       | 7.4214593837052e-07 | "VOLATILE_ACIDITY" |
| 9       | 0.15625 | "VOLATILE_ACIDITY" |

(7 rows)

DROP_TABLE

-------------
Related Functions

- category Analytics - Statistics
- MWW_TEST
- PWILCOX
- T_PMD_TEST
CHAPTER 3

Reference Documentation: Utilities

_sp_utl_aggregateExists - Check if a UDA aggregate function exists

The stored procedure _sp_utl_aggregateExists is deprecated. This stored procedure checks if a UDA aggregate function exists. The aggregate function name must not contain a schema.

Usage

The _sp_utl_aggregateExists stored procedure has the following syntax:

► _sp_utl_aggregateExists(NVARCHAR(ANY) aggregateName, BOOLEAN verbose, BOOLEAN inverse)

▲ Parameters

► aggregateName
  the aggregate UDA function to look for. It can be qualified by a database name (<database>..<aggregateName>) if not in the current database. Do not specify arguments, this stored procedure does not check the full signature but only the name of the UDA aggregate function.
  Type: NVARCHAR(ANY)

► verbose
  flag indicating whether the stored procedure should raise exceptions or not
  Type: BOOLEAN

► inverse
  flag indicating whether the stored procedure raises an exception when the UDA aggregate function does not exists (inverse=false) or does exist (inverse=true). This flag is used only when verbose=true. It does not have any influence on the return value.
  Type: BOOLEAN

▲ Returns

BOOLEAN true if the specified UDA aggregate function exists
Examples

CALL
nza.._sp_utl_aggregateExists('nza.._ca_dm_decisiontree_entropy', true, false);

_SP_UTL_AGGREGATEEXISTS
-------------------------
t
(1 row)

CALL
nza.._sp_utl_aggregateExists('nza.._ca_dm_decisiontree_entropy', true, true);

ERROR: Aggregate "NZA"."_CA_DM_DECISIONTREE_ENTROPY" already exists.

CALL nza.._sp_utl_aggregateExists('does_not_exist', true, true);

_SP_UTL_AGGREGATEEXISTS
-------------------------
f
(1 row)

Related Functions
► category Utilities - Checking
► _sp_utl_functionExists
► _sp_utl_procedureExists

_sp_utl_columnContainsNulls - Check if a column contains at least one NULL value

The stored procedure _sp_utl_columnContainsNulls is deprecated. This stored procedure checks if a column of a table or view contains at least one NULL value. The table or view name must not
Usage

The _sp_utl_columnContainsNulls stored procedure has the following syntax:

**_sp_utl_columnContainsNulls**(NVARCHAR(ANY) tableName, NVARCHAR(ANY) columnName, BOOLEAN verbose, BOOLEAN inverse)

▲ Parameters

► **tableName**
  the table or view to search the column into. It can be qualified by a database name (<database>..<tableName>) if not in the current database.
  Type: NVARCHAR(ANY)

► **columnName**
  the column to check
  Type: NVARCHAR(ANY)

► **verbose**
  flag indicating whether the stored procedure should raise an exception or not when the existence of NULL values is not as expected. The stored procedure always raise an exception if the table or view, or the column does not exist.
  Type: BOOLEAN

► **inverse**
  flag indicating whether the stored procedure raises an exception when the given column does not contain NULL values (inverse=false) or does contain NULL values (inverse=true). This flag is used only when verbose=true. It does not have any influence on the return value.
  Type: BOOLEAN

▲ Returns

BOOLEAN true if the specified column contains at least one NULL value

Examples

```
CALL nza.._sp_utl_columnContainsNulls('nza..adult_test', 'age', true, true);

_SP_UTL_COLUMNCONTAINSNULLS
-----------------------------
  f
(1 row)
```

```
CALL nza.._sp_utl_columnContainsNulls('nza..adult_test', 'age', true, false);

ERROR:  Column "AGE" in relation "NZA"."ADULT_TEST" does not
```
contain null values.

Related Functions

- category Utilities - Checking
- _sp_utl_columnExists

_sp_utl_columnExists - Check if a column exists

The stored procedure _sp_utl_columnExists is deprecated. This stored procedure checks if a column exists in a table or a view. The table or view name must not contain a schema.

Usage

The _sp_utl_columnExists stored procedure has the following syntax:

```sql
_sp_utl_columnExists(NVARCHAR(ANY) tableName, NVARCHAR(ANY) columnName, BOOLEAN verbose, BOOLEAN inverse)
```

Parameters

- **tableName**
  - the table or view to search a column into. It can be qualified by a database name (<database>..<tableName>) if not in the current database.
  - Type: NVARCHAR(ANY)

- **columnName**
  - the column to look for
  - Type: NVARCHAR(ANY)

- **verbose**
  - flag indicating whether the stored procedure should raise exceptions or not
  - Type: BOOLEAN

- **inverse**
  - flag indicating whether the stored procedure raises an exception when the table, view or column does not exists (inverse=false) or does exist (inverse=true). This flag is used only when verbose=true. It does not have any influence on the return value.
  - Type: BOOLEAN

Returns

BOOLEAN true if the specified column exists

Examples

```sql
CALL nza.._sp_utl_columnExists('nza..weather', 'play', true, false);
```
CALL nza.._sp_utl_columnExists('NZA..weather', 'PLAY', true, true);
ERROR: Column "PLAY" in relation "NZA".."WEATHER" already exists.

CALL nza.._sp_utl_columnExists('NZA..weather', 'wrong_column', true, false);
ERROR: Column "WRONG_COLUMN" in relation "NZA".."WEATHER" does not exist.

CALL nza.._sp_utl_columnExists('"NZA".."weather"', '"PLAY"', false, false);

Related Functions
► category Utilities - Checking
► _sp_utl_relationExists

_sp_utl_columnIsId - Check if a column contains unique values

The stored procedure _sp_utl_columnIsId is deprecated. This stored procedure checks if a column of a table or view contains only unique values. The table or view name must not contains a schema.

Usage
The _sp_utl_columnIsId stored procedure has the following syntax:
__sp_utl_columnIsId__

### Parameters

- **tableName**
  - the table or view to search the column into. It can be qualified by a database name (`<database>..<tableName>`) if not in the current database.
  - Type: NVARCHAR(ANY)

- **columnName**
  - the column to check.
  - Type: NVARCHAR(ANY)

- **verbose**
  - flag indicating whether the stored procedure should raise an exception or not when the existence of unique values is not as expected. The stored procedure always raise an exception if the table or view, or the column does not exist.
  - Type: BOOLEAN

- **inverse**
  - flag indicating whether the stored procedure raises an exception when the given column does not contain unique values (inverse=false) or does contain unique values (inverse=true). This flag is used only when verbose=true. It does not have any influence on the return value.
  - Type: BOOLEAN

### Returns

 BOOLEAN true if the specified column contains only unique values

### Examples

```sql
CALL nza.._sp_utl_columnIsId('nza..iris', 'wrong_column', false, false);
ERROR: Column "WRONG_COLUMN" in relation "NZA".."IRIS" does not exist.

CALL nza.._sp_utl_columnIsId('nza..iris', 'id', true, true);
ERROR: Column "ID" in relation "NZA".."IRIS" contains unique values.

CALL nza.._sp_utl_columnIsId('nza..iris', 'id', true,
false);

_SP_UTL_COLUMNISID
-------------------
t
(1 row)

Related Functions
► category Utilities - Checking
► _sp_utl_columnExists

_sp_utl_columnIsNumeric - Check if a column is of a numeric data type
The stored procedure _sp_utl_columnIsNumeric is deprecated. This stored procedure checks if a column of
a table or view is of a numeric data type: int1, int2, int4, int8, float, double or numeric. Boolean columns, as
well as date, interval, time, timetz or timestamp columns are not considered numeric although they can be
casted to numeric values. The table or view name must not contains a schema.

Usage
The _sp_utl_columnIsNumeric stored procedure has the following syntax:

► _sp_utl_columnIsNumeric(NVARCHAR(ANY) tableName, NVARCHAR(ANY) columnName, BOOLEAN verbose, BOOLEAN inverse)
▲ Parameters
► tableName
the table or view to search the column into. It can be qualified by a database name
(<database>..<tableName>) if not in the current database.
Type: NVARCHAR(ANY)
► columnName
the column to check
Type: NVARCHAR(ANY)
► verbose
flag indicating whether the stored procedure should raise an exception or not when the data
type of the column is not as expected. The stored procedure always raise an exception if the
table or view, or the column does not exist.
Type: BOOLEAN
► inverse
flag indicating whether the stored procedure raises an exception when the given column is not
numeric (inverse=false) or is numeric (inverse=true). This flag is used only when verbose=true.
It does not have any influence on the return value.
Type: BOOLEAN

▲ Returns
BOOLEAN true if the specified column is of a numeric data type

Examples

CALL nza.._sp_utl_columnIsNumeric('nza..adult_test',
'age', true, false);

_SP_UTL_COLUMNISNUMERIC

-------------------------
t
(1 row)

CALL nza.._sp_utl_columnIsNumeric('nza..adult_testd',
'age', true, false);
ERROR: Relation "NZA".."ADULT_TESTD" does not exist or cannot be accessed.

CALL nza.._sp_utl_columnIsNumeric('nza..adult_test',
'age2', true, false);
ERROR: Column "AGE2" in relation "NZA".."ADULT_TEST" does not exist.

CALL
nza.._sp_utl_columnIsNumeric('nza..adult_test','age',
true, true);
ERROR: Column "AGE" in relation "NZA".."ADULT_TEST" contains continuous values.

Related Functions

► category Utilities - Checking
► _sp_utl_columnExists
_sp_utl_columnListExists - Check if a column exists

The stored procedure _sp_utl_columnListExists is deprecated. This stored procedure checks if a column exists in a table or a view. Columns are passed as list separated by semicolon. The table or view name must not contain a schema.

Usage

The _sp_utl_columnListExists stored procedure has the following syntax:

```
▶_sp_utl_columnListExists(NVARCHAR(ANY) tableName, NVARCHAR(ANY) columnName, BOOLEAN verbose, BOOLEAN inverse)
```

Parameters

► tableName
the table or view to search a column into. It can be qualified by a database name (<database>..<tableName>) if not in the current database.

Type: NVARCHAR(ANY)

► columnName
the columns to look for, list is separated by semicolons

Type: NVARCHAR(ANY)

► verbose
flag indicating whether the stored procedure should raise exceptions or not

Type: BOOLEAN

► inverse
flag indicating whether the stored procedure raises an exception when the table, view or column does not exist (inverse=false) or does exist (inverse=true). This flag is used only when verbose=true. It does not have any influence on the return value.

Type: BOOLEAN

Returns

BOOLEAN true if the specified columns exist

Examples

```
CALL nza.._sp_utl_columnListExists('nza..weather', 'play;temperature', true, false);

_SP_UTL_COLUMNLISTEXISTS
----------------------
t
(1 row)
```
IBM Netezza In-Database Analytics Reference Guide

Related Functions

► category Utilities - Checking
► _sp_utl_columnExists

_sp_utl_columnsEqualTypes - Check if two columns have the same data type

This stored procedure checks if two columns of tables or views have the same data type. The length or precision of the data types must match too.

Usage

The _sp_utl_columnsEqualTypes stored procedure has the following syntax:

► _sp_utl_columnsEqualTypes(NVARCHAR(ANY) tableName1, NVARCHAR(ANY) columnName1, NVARCHAR(ANY) tableName2, NVARCHAR(ANY) columnName2, BOOLEAN verbose, BOOLEAN inverse)

▲ Parameters

► tableName1
the table or view to search the first column into. It can be qualified by a database name (<database>..<tableName>) if not in the current database.
Type: NVARCHAR(ANY)

► columnName1
the first column to check
Type: NVARCHAR(ANY)

► tableName2
the table or view to search the second column into. It can be qualified by a database name (<database>..<tableName>) if not in the current database.
Type: NVARCHAR(ANY)

► columnName2
the second column to check
Type: NVARCHAR(ANY)

► verbose
flag indicating whether the stored procedure should raise an exception or not when the column data types are not as expected. The stored procedure always raise an exception if a table or view, or a column does not exist.
Type: BOOLEAN

► inverse
flag indicating whether the stored procedure raises an exception when the given columns have different data types (inverse=false) or have the same data type
(inverse=true). This flag is used only when verbose=true. It does not have any influence on the return value.

Type: BOOLEAN

▲ Returns

BOOLEAN true if the specified columns have the same data type

Examples

CALL nza.._sp_utl_columnsEqualTypes('nza..adult_test', 'age',
'nza..adult_test', 'age', true, false);

_SP_UTL_COLUMNSEQUALTYPES

--------------------------------

 t

(1 row)

CALL nza.._sp_utl_columnsEqualTypes('nza..adult_test', 'age',
'nza..adult_test', 'education', true, false);

ERROR:  Columns: "AGE", "EDUCATION" in relations
"NZA".."ADULT_TEST", "NZA".."ADULT_TEST" are not equal type.
type1:integer type2:character varying(12).

CALL nza.._sp_utl_columnsEqualTypes('nza..adult_test', 'age',
'nza..adult_test', 'capital_gain', true, true);

ERROR:  Columns: "AGE", "CAPITAL_GAIN" in relations
"NZA".."ADULT_TEST", "NZA".."ADULT_TEST" are equal type.
type1:integer type2:integer.

Related Functions

► category Utilities - Checking
► _sp_utl_columnExists

_sp_utl_dropAllAggregates - Drop all UDA aggregate functions with the given prefix

This stored procedure drops all UDA aggregate functions whose name matches a given prefix
Usage

The _sp_utl_dropAllAggregates stored procedure has the following syntax:

► _sp_utl_dropAllAggregates(NVARCHAR(ANY) inputName)

▲ Parameters

► inputName
  the prefix of the UDA aggregate functions to drop. Only UDA aggregate functions in
  the current database can be dropped.
  Type: NVARCHAR(ANY)

▲ Returns
  INT4 the number of UDA aggregate functions that have been dropped

Examples

CALL
nza.._sp_utl_dropAllAggregates('uda_does_not_exist');

_SP_UTL_DROPALLAGGREGATES
---------------------------
  0
(1 row)

CALL
nza.._sp_utl_dropAllAggregates('nza..uda_does_not_exist')
;
ERROR:  Cross database access not allowed:
'nza..uda_does_not_exist'.

Related Functions

► category Utilities - Actions
► _sp_utl_dropAllFunctions
► _sp_utl_dropAllProcedures
► _sp_utl_dropAllUDX

_sp_utl_dropAllFunctions - Drop all UDF functions with the given prefix

This stored procedure drops all UDF functions whose name matches a given prefix
Usage

The _sp_utl_dropAllFunctions stored procedure has the following syntax:

► _sp_utl_dropAllFunctions(NVARCHAR(ANY) inputName)

▲ Parameters

► inputName

the prefix of the UDF functions to drop. Only UDF functions in the current database can be dropped.

Type: NVARCHAR(ANY)

▲ Returns

INT4 the number of UDF functions that have been dropped

Examples

CALL nza.._sp_utl_dropAllFunctions('udf_does_not_exist');

_SP_UTL_DROPALLFUNCTIONS

--------------------------

0

(1 row)

CALL nza.._sp_utl_dropAllFunctions('nza..udf_does_not_exist');

ERROR: Cross database access not allowed: 'nza..udf_does_not_exist'.

Related Functions

► category Utilities - Actions

► _sp_utl_dropAllAggregates

► _sp_utl_dropAllProcedures

► _sp_utl_dropAllUDX

_sp_utl_dropAllLike - Drop all database objects with the given LIKE expression

This stored procedure drops all tables, temporary tables and views whose name matches a given LIKE expression

Usage

The _sp_utl_dropAllLike stored procedure has the following syntax:
_sp_utl_dropAllLike(NVARCHAR(ANY) tableExpr)

- **Parameters**
  - **tableExpr**
    - the LIKE expression of the database objects to drop
      - Type: NVARCHAR(ANY)

- **Returns**
  - INT4 number of tables, temporary tables and views that have been dropped

**Examples**

```
CALL nza.._sp_utl_dropAllLike('nza..object_doest_not %_exist');

_SP_UTL_DROPALLLIKE
---------------------
0
```

(1 row)

**Related Functions**

- **category** Utilities - Actions
- **DROP_TABLE**

_sp_utl_dropAllProcedures - Drop all stored procedures with the given prefix

This stored procedure drops all stored procedures whose name matches a given prefix

**Usage**

The _sp_utl_dropAllProcedures stored procedure has the following syntax:

```
▶ _sp_utl_dropAllProcedures(NVARCHAR(ANY) inputName)

- **Parameters**
  - **inputName**
    - the prefix of the stored procedures to drop. Only stored procedures in the current database can be dropped.
      - Type: NVARCHAR(ANY)

- **Returns**
  - INT4 the number of stored procedures that have been dropped
```
Examples

CALL nza.._sp_utl_dropAllProcedures('sp_does_not_exist');

_SP_UTL_DROPALLPROCEDURES
-----------------------------
0
(1 row)

CALL nza.._sp_utl_dropAllProcedures('nza..sp_does_not_exist');

ERROR:  Cross database access not allowed: 'nza..sp_does_not_exist'.

Related Functions
► category Utilities - Actions
► _sp_utl_dropAllAggregates
► _sp_utl_dropAllFunctions
► _sp_utl_dropAllUDX

_sp_utl_dropAllUDX - Drop all UDx functions with the given prefix

This stored procedure drops all UDA aggregate functions or UDF functions whose name matches a given prefix.

Usage

The _sp_utl_dropAllUDX stored procedure has the following syntax:

► _sp_utl_dropAllUDX(NVARCHAR(any) inputName);

▲ Parameters
► inputName
the prefix of the UDA aggregate functions or UDF functions to drop. Only UDx functions in the current database can be dropped.
Type: NVARCHAR(ANY)

▲ Returns
INT4 the number of UDA aggregate functions and UDF functions that have been dropped

Examples

CALL nza.._sp_utl_dropAllUDX('udx_does_not_exist');
CALL nza.._sp_utl_dropAllUDX('nza..udx_does_not_exist');
ERROR: Cross Database Access is not supported.

Related Functions

- category Utilities - Actions
- _sp_utl_dropAllAggregates
- _sp_utl_dropAllFunctions
- _sp_utl_dropAllProcedures

_sp_utl_functionExists - Check if a UDF function exists

The stored procedure _sp_utl_functionExists is deprecated. This stored procedure checks if an UDF function exists. The UDF function name must not contain a schema.

Usage

The _sp_utl_functionExists stored procedure has the following syntax:

```plaintext
_sp_utl_functionExists(NVARCHAR(ANY) functionName, BOOLEAN verbose, BOOLEAN inverse)
```

▲ Parameters

- **functionName**
  the UDF function to look for. It can be qualified by a database name (<database>..<functionName>) if not in the current database. Do not specify arguments, this stored procedure does not check the full signature but only the name of the UDF function.
  Type: NVARCHAR(ANY)

- **verbose**
  flag indicating whether the stored procedure should raise exceptions or not
  Type: BOOLEAN

- **inverse**
flag indicating whether the stored procedure raises an exception when the UDF function does not exist (inverse=false) or does exist (inverse=true). This flag is used only when verbose=true. It does not have any influence on the return value.

Type: BOOLEAN

▲ Returns

BOOLEAN true if the specified UDF function exists

Examples

CALL nza.._sp_utl_functionExists('nza..abs', true, false);

_SP_UTL_FUNCTIONEXISTS
------------------------
t
(1 row)

CALL nza.._sp_utl_functionExists('nza..abs', true, true);
ERROR: Function "NZA"..'ABS' already exists.

Related Functions

► category Utilities - Checking
► _sp_utl_aggregateExists
► _sp_utl_procedureExists

_sp_utl_getColumnType - Column data type

This stored procedure returns the SQL data type of a column of a relation

Usage

The _sp_utl_getColumnType stored procedure has the following syntax:

► _sp_utl_getColumnType(NVARCHAR(ANY) threePartName, NVARCHAR(ANY) columnName, BOOLEAN ignoreSize)

▲ Parameters

► threePartName
Full string name of a database object in the form database.schema.objectname.
Database and schema are optional (default: current database/default database, current schema).
Type: NVARCHAR(ANY)
columnName
The string name of a column whose SQL data type is returned. The column must exist.
Type: NVARCHAR(ANY)

ignoreSize
If true, the size of character data types and the scale and precision of the numeric
data type are removed before the SQL type is returned.
Type: BOOLEAN

Returns
NVARCHAR(ANY) The SQL data type of the column in lower case. If relation or column do
not exist, NULL is returned.

Examples
CALL nza.._sp_utl_getColumnType('nza..weather',
'INSTANCE', false);
_SP_UTL_GETCOLUMNTYPE
-----------------------
integer
(1 row)

CALL nza.._sp_utl_getColumnType('nza..weather',
'TEMPERATURE', true);
_SP_UTL_GETCOLUMNTYPE
-----------------------
national character varying
(1 row)

CALL
nza.._sp_utl_getColumnType('nza..table_does_not_exist',
'id', false);
_SP_UTL_GETCOLUMNTYPE
-----------------------
(1 row)

CALL nza.._sp_utl_getColumnType('nza..weather',
'column_does_not_exist', false);
**_SP_UTL_GETCOLUMNTYPE**

(1 row)

**Related Functions**

- **category**: Utilities - Data Exploration

---

**_sp_utl_getTableSize - Table row count**

This stored procedure returns the number of rows of the given table.

**Usage**

The _sp_utl_getTableSize stored procedure has the following syntax:

```
_sp_utl_getTableSize((NVARCHAR(ANY) tableName)
```

**Parameters**

- **tableName**
  - the table to get the size of
  - Type: NVARCHAR(ANY)

**Returns**

- INT8 the number of rows in the table

**Examples**

```
CALL nza.._sp_utl_getTableSize('nza..weather');
```

```
_SP_UTL_GETTABLESIZE
----------------------
22
(1 row)
```

**Related Functions**

- **category**: Utilities - Data Exploration

---

**_sp_utl_isTempTable - Check if a table is temporary**

The stored procedure _sp_utl_isTempTable is deprecated. This stored procedure checks if a table is temporary. The table name must not contain a schema.
Usage
The _sp_utl_isTempTable stored procedure has the following syntax:

► _sp_utl_isTempTable(NVARCHAR(ANY) tableName)
  ▲ Parameters
  ► tableName
    the table to check. It can be qualified by a database name (<database>.<tableName>)
    if not in the current database.
    Type: NVARCHAR(ANY)
  ▲ Returns
    BOOLEAN true if the specified table is temporary

Examples
CALL nza.._sp_utl_isTempTable('table_does_not_exist');
  _SP_UTL_ISTEMPTABLE
  ---------------------
  f
  (1 row)

CREATE TEMPORARY TABLE ttt (id INTEGER);
CALL nza.._sp_utl_isTempTable('ttt');
CALL nza..DROP_TABLE('ttt');

  _SP_UTL_ISTEMPTABLE
  ---------------------
  t
  (1 row)

  DROP_TABLE
  ---------
  t
  (1 row)
Related Functions

- category Utilities - Checking
- _sp_utl_tableExists

_sp_utl_justExecute - Execute a SQL statement

This stored procedure executes a SQL statement

Usage

The _sp_utl_justExecute stored procedure has the following syntax:

- **_sp_utl_justExecute**(NVARCHAR(ANY) commandString)

▲ Parameters

- **commandString**
  the SQL command to execute
  Type: NVARCHAR(ANY)

▲ Returns

BOOLEAN always true. An exception is raised if the command cannot be executed.

Examples

```
CALL nza.._sp_utl_justExecute('create table mytest(id INTEGER)');
CALL nza.._sp_utl_justExecute('insert into mytest values(1)');
CALL nza.._sp_utl_justExecute('select id from mytest');
CALL nza.._sp_utl_justExecute('drop table mytest');
CALL nza.._sp_utl_justExecute('drop table mytest');
```
IBM Netezza In-Database Analytics Reference Guide

```sql
_SP_UTL_JUSTEXECUTE
---------------------
t
(1 row)
```

```sql
_SP_UTL_JUSTEXECUTE
---------------------
t
(1 row)
```

ERROR: Table 'MYTEST' does not exist

Related Functions
► category Utilities - Actions

_sp_utl_procedureExists - Check if a stored procedure exists

The stored procedure _sp_utl_procedureExists is deprecated. This stored procedure checks if a stored procedure exists. The stored procedure name must not contain a schema.

Usage

The _sp_utl_procedureExists stored procedure has the following syntax:

► _sp_utl_procedureExists(NVARCHAR(ANY) procedureName, BOOLEAN verbose, BOOLEAN inverse)

▲ Parameters

► procedureName
the stored procedure to look for. It can be qualified by a database name (<database>..<procedureName>) if not in the current database. Do not specify arguments, this stored procedure does not check the full signature but only the name of the stored procedure.

Type: NVARCHAR(ANY)

► verbose
flag indicating whether the stored procedure should raise exceptions or not

Type: BOOLEAN
inverse
flag indicating whether the stored procedure raises an exception when the stored procedure
does not exists (inverse=false) or does exist (inverse=true). This flag is used only when
verbose=true. It does not have any influence on the return value.

Type: BOOLEAN

Returns
BOOLEAN true if the specified stored procedure exists

Examples

CALL nza.._sp_utl_procedureExists('nza..dectreee', true, false);
ERROR: Procedure "NZA"."DECTREEE" does not exist.

CALL nza.._sp_utl_procedureExists('nza..dectree', true, false);
_SP_UTL_PROCEDUREEXISTS
-------------------------
t
(1 row)

CALL nza.._sp_utl_procedureExists('nza..dectree', true, true);
ERROR: Procedure "NZA"."DECTREE" already exists.

CALL nza.._sp_utl_procedureExists('nza..dectreee', true, true);
_SP_UTL_PROCEDUREEXISTS
-------------------------
f
(1 row)

Related Functions
► category Utilities - Checking
► _sp_utl_aggregateExists
► _sp_utl_functionExists
_sp_utl_relationExists - Check if a table or a view exists

The stored procedure _sp_utl_relationExists is deprecated. This stored procedure checks if a table or a view exists. The table or view name must not contain a schema.

Usage
The _sp_utl_relationExists stored procedure has the following syntax:

► _sp_utl_relationExists(NVARCHAR(any) objectName, BOOLEAN verbose, BOOLEAN inverse)

▲ Parameters
  ► objectName
    the table or view to look for. It can be qualified by a database name (<database>..<objectName>) if not in the current database.
    Type: NVARCHAR(any)
  ► verbose
    flag indicating whether the stored procedure should raise exceptions or not
    Type: BOOLEAN
  ► inverse
    flag indicating whether the stored procedure raises an exception when the table or view does not exists (inverse=false) or does exist (inverse=true). This flag is used only when verbose=true. It does not have any influence on the return value.
    Type: BOOLEAN

▲ Returns
BOOLEAN true if the specified table or view exists

Examples

CALL nza.._sp_utl_relationExists('nza..weather', true, false);

_SP_UTL_RELATIONEXISTS
------------------------
t
(1 row)

CALL nza.._sp_utl_relationExists('nza..weather', true, true);

ERROR: Relation "NZA"."WEATHER" already exists.
CALL nza.._sp_utl_relationExists('table_does_not_exist', true, false);

ERROR: Relation "TABLE_DOES_NOT_EXIST" does not exist or cannot be accessed.

Related Functions

- category Utilities - Checking
- _sp_utl_tableExists
- _sp_utl_viewExists

_sp_utl_sequenceExists - Check if a sequence exists

The stored procedure _sp_utl_sequenceExists is deprecated. This stored procedure checks if a sequence exists. The sequence name must not contain a schema.

Usage

The _sp_utl_sequenceExists stored procedure has the following syntax:

```sql
_sp_utl_sequenceExists(NVARCHAR(ANY) sequenceName, BOOLEAN verbose, BOOLEAN inverse)
```

▲ Parameters

- **sequenceName**
  the sequence to look for. It can be qualified by a database name (<database>..<sequenceName>) if not in the current database.
  Type: NVARCHAR(ANY)

- **verbose**
  flag indicating whether the stored procedure should raise exceptions or not
  Type: BOOLEAN

- **inverse**
  flag indicating whether the stored procedure raises an exception when the sequence does not exists (inverse=false) or does exist (inverse=true). This flag is used only when verbose=true. It does not have any influence on the return value.
  Type: BOOLEAN

▲ Returns

BOOLEAN true if the specified sequence exists

Examples

CREATE SEQUENCE testnzaseq;
CALL nza.._sp_utl_sequenceExists('testnzaseq', true, false);
CALL nza.._sp_utl_sequenceExists('testnzaseq', true, true);
DROP SEQUENCE testnzaseq;
_SP_UTL_SEQUENCEEXISTS
------------------------
t
(1 row)

ERROR: Sequence "TESTNZASEQ" already exists.

Related Functions
► category Utilities - Checking

_sp_utl_tableExists - Check if a table exists

The stored procedure _sp_utl_tableExists is deprecated. This stored procedure checks if a table exists. The table name must not contain a schema.

Usage

The _sp_utl_tableExists stored procedure has the following syntax:

► _sp_utl_tableExists(NVARCHAR(ANY) tableName, BOOLEAN verbose, BOOLEAN inverse)
▲ Parameters
► tableName
   the table to look for. It can be qualified by a database name (<database>..<tableName>) if not in the current database.
   Type: NVARCHAR(ANY)
► verbose
   flag indicating whether the stored procedure should raise exceptions or not
   Type: BOOLEAN
► inverse
   flag indicating whether the stored procedure raises an exception when the table does not exists (inverse=false) or does exist (inverse=true). This flag is used only when verbose=true. It does not have any influence on the return value.
Type: BOOLEAN

▲ Returns

BOOLEAN true if the specified table exists

Examples

CALL nza.._sp_utl_tableExists('nza..weather', true, false);

_SP_UTL_TABLEEXISTS

---------------------
t
(1 row)

CALL nza.._sp_utl_tableExists('nza..weather', true, true);
ERROR: Table "NZA"."WEATHER" already exists.

CALL nza.._sp_utl_tableExists('nza..weatherrr', true, false);
ERROR: Table "NZA"."WEATHERRR" does not exist.

Related Functions

► category Utilities - Checking
► _sp_utl_isTempTable
► _sp_utl_relationExists
► _sp_utl_viewExists

_sp_utl_viewExists - Check if a view exists

The stored procedure _sp_utl_viewExists is deprecated. This stored procedure checks if a view exists. The view name must not contain a schema.

Usage

The _sp_utl_viewExists stored procedure has the following syntax:

► _sp_utl_viewExists(NVARCHAR(ANY) viewName, BOOLEAN verbose, BOOLEAN inverse)

▲ Parameters

► viewName
  the view to look for. It can be qualified by a database name (<database>..<viewName>) if not in
the current database.
Type: NVARCHAR(ANY)

► **verbose**
flag indicating whether the stored procedure should raise exceptions or not
Type: BOOLEAN

► **inverse**
flag indicating whether the stored procedure raises an exception when the view does
not exist (inverse=false) or does exist (inverse=true). This flag is used only when ver-
bose=true. It does not have any influence on the return value.
Type: BOOLEAN

▲ Returns
BOOLEAN true if the specified view exists

**Examples**

```sql
CALL nza.._sp_utl_viewExists('V_NZA_MODELS', true, false);

_SP_UTL_VIEWEXISTS
---------------
 t
(1 row)
```

```sql
CALL nza.._sp_utl_viewExists('V_NZA_MODELS', true, true);
ERROR: View "V_NZA_MODELS" already exists.
```

```sql
CALL nza.._sp_utl_viewExists('view_does_not_exist', true, false);
ERROR: View "VIEW DOES NOT_EXIST" does not exist.
```

**Related Functions**

► category Utilities - Checking
► _sp_utl_relationExists
► _sp_utl_tableExists
drand64 - 64 bits pseudo-random number generator

This function returns a random number using a data-slice independent 64 bits pseudo-random number generator.

Usage

The drand64 function has the following syntax:

```
> drand64(INT8 randomSeed)
▲ Parameters
  ▲ randomSeed
random seed
  Type: INT8
▲ Returns
DOUBLE random number between 0.0 and 1.0
```

Details

This function delivers a random double value between 0.0 and 1.0. The 64 bits pseudo-random number generator is based on different prime numbers per data slice, ensuring the independence of the sequences of numbers produced on any pair of data slices.

Examples

```
SELECT nza..drand64(123);

DRAND64
------------------
0.35831899940968
(1 row)
```

Related Functions

- category Utilities - Preprocessing

DROP_TABLE - Drop a table

This stored procedure drops a table if it exists

Usage

The DROP_TABLE stored procedure has the following syntax:

```
> DROP_TABLE(NVARCHAR(ANY) tableName)
```
Parameters

- **tableName**
  the table to drop. The table name can be indicated directly or prefixed with 'intable='.
  Type: NVARCHAR(ANY)

Returns

BOOLEAN true if the table was dropped, false if the table does not exist or cannot be dropped

Examples

```sql
CREATE TABLE mytest (id INTEGER);
CALL nza..DROP_TABLE('mytest');

DROP_TABLE
------------
t
(1 row)

CREATE TABLE mytest (id INTEGER);
CALL nza..DROP_TABLE('intable=mytest');

DROP_TABLE
------------
t
(1 row)

CALL nza..DROP_TABLE('nza..table_does_not_exist');

DROP_TABLE
------------
f
(1 row)
```
**Related Functions**

- category Utilities - Actions
- _sp_utl_dropAllLike

---

**ISDATE_TINY - Check if a string has the compact date format YYYYMMDD**

This function checks that the input string represents a valid date with format YYYYMMDD

**Usage**

The ISDATE_TINY function has the following syntax:

```
ISDATE_TINY(VARCHAR(100) date)
```

**Parameters**

- `date`
  - the date string to check
  - Type: VARCHAR(100)

**Returns**

- INT4 1 if the string is a valid date with format YYYYMMDD, NULL if the string is NULL, or 0 otherwise

**Details**

This function checks that the input string represents a valid date with format YYYYMMDD. A valid date consist of exactly 8 characters, each character must be between '0' and '9'. The month part must be between 01 and 12, the days between 01 and the number of days in the month. Leap years are taken into account.

Note that it is possible to pass date as INT. However, when INT is automatically converted to VARCHAR, its leading zeros are truncated: e.g. 00010522 gets '10522' and is finally not recognized as a valid date.

**Examples**

```sql
CREATE TABLE DateAsString(date VARCHAR(50));
INSERT INTO DateAsString VALUES ('2009010');
INSERT INTO DateAsString VALUES ('200901222');
INSERT INTO DateAsString VALUES ('200');
INSERT INTO DateAsString VALUES ('-20091101');
INSERT INTO DateAsString VALUES ('20090001');
INSERT INTO DateAsString VALUES ('20091301');
INSERT INTO DateAsString VALUES ('20090400');
INSERT INTO DateAsString VALUES ('20090431');
INSERT INTO DateAsString VALUES ('20080229');
INSERT INTO DateAsString VALUES ('20090229');
INSERT INTO DateAsString VALUES ('21000229');
```
INSERT INTO DateAsString VALUES ('19991919');
INSERT INTO DateAsString VALUES ('20040429');
INSERT INTO DateAsString VALUES ('yyyymmdd');
INSERT INTO DateAsString VALUES ('abcd0101');
INSERT INTO DateAsString VALUES ('Date');
INSERT INTO DateAsString VALUES (NULL);
SELECT nza.ISDATE_TINY(date), date FROM DateAsString ORDER BY isdate_tiny, date;
CALL nza.DROP_TABLE('DateAsString');

| ISDATE_TINY | DATE          |
|-------------+--------------|
|            |              |
| 0 | -20091101    |
| 0 | 19991919     |
| 0 | 200          |
| 0 | 20090001     |
| 0 | 2009010      |
| 0 | 200901222    |
| 0 | 20090229     |
| 0 | 20090400     |
| 0 | 20090431     |
| 0 | 20091301     |
| 0 | 21000229     |
| 0 | Date         |
| 0 | abcd0101     |
| 0 | yyyymmdd     |
| 1 | 20040429     |
| 1 | 20080229     |

(17 rows)
Related Functions

- category Utilities - Checking

**msghelp - Show information related to a message number in the correct language of the locale.**

This stored procedure provides detailed information on a specific message (by number) in the language of the locale.

**Usage**

The msghelp stored procedure has the following syntax:

```
msghelp(varchar(11) message_num);
```

**Parameters**

- **message_number**
  
  message number with 3-character product prefix + 2-character component prefix (e.g. 'NZAMT-0045')
  
  Type: VARCHAR(11)

**Returns**

boolean Always returns TRUE

**Examples**

```sql
    call nza..msghelp('NZAMT-0045');
    call nza..msghelp('NZAMT-0045E');
```

Appears on the screen as a notice (TODO: An actual example of output needs to go here when we've settled on a message format)

**Related Functions**

- category Utilities - Checking
Notices and Trademarks

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user’s responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785 U.S.A.

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
26 Forest Street
Marlborough, MA 01752 U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement
or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only. This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. (enter the year or years). All rights reserved.

Trademarks

IBM, the IBM logo, ibm.com and Netezza are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at ibm.com/legal/copytrade.shtml.

The following terms are trademarks or registered trademarks of other companies:

Adobe is a registered trademark of Adobe Systems Incorporated in the United States, and/or other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

NEC is a registered trademark of NEC Corporation.

UNIX is a registered trademark of The Open Group in the United States and other countries.
Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Red Hat is a trademark or registered trademark of Red Hat, Inc. in the United States and/or other countries.

D-CC, D-C++, Diab+, FastJ, pSOS+, SingleStep, Tornado, VxWorks, Wind River, and the Wind River logo are trademarks, registered trademarks, or service marks of Wind River Systems, Inc. Tornado patent pending.

APC and the APC logo are trademarks or registered trademarks of American Power Conversion Corporation.

Other company, product or service names may be trademarks or service marks of others.

Regulatory and Compliance

Regulatory Notices
Install the NPS system in a restricted-access location. Ensure that only those trained to operate or service the equipment have physical access to it. Install each AC power outlet near the NPS rack that plugs into it, and keep it freely accessible. Provide approved 30A circuit breakers on all power sources.

Product may be powered by redundant power sources. Disconnect ALL power sources before servicing. High leakage current. Earth connection essential before connecting supply. Courant de fuite élevé. Raccordement à la terre indispensable avant le raccordement au réseau.

Homologation Statement
Attention: This product is not intended to be connected directly or indirectly by any means whatsoever to interfaces of public telecommunications networks, neither to be used in a Public Services Network.

FCC - Industry Canada Statement
This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to part 15 of the FCC rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses, and can radiate radio-frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference, in which case users will be required to correct the interference at their own expense.

This Class A digital apparatus meets all requirements of the Canadian Interference-Causing Equipment Regulations.

Cet appareil numérique de la classe A respecte toutes les exigences du Règlement sur le matériel brouilleur du Canada.

WEEE
Netezza Corporation is committed to meeting the requirements of the European Union (EU) Waste Electrical and Electronic Equipment (WEEE) Directive. This Directive requires producers of electrical and electronic equipment to finance the takeback, for reuse or recycling, of their products placed on the EU market after August 13, 2005.

CE Statement (Europe)

Warning: This is a class A product. In a domestic environment this product may cause radio interference in which case the user may be required to take adequate measures.
VCCI Statement

この装置は、情報処理装置等電波障害自主規制協議会（VCCI）の基準に基づくクラスA情報技術装置です。この装置を家庭環境で使用すると電波障害を引き起こすことがあります。この場合には使用者が適切な対策を講ずるよう要求されることがあります。
Index

Symbols

_sp_utl_aggregateExists, 701
_sp_utl_columnContainsNulls, 703
_sp_utl_columnExists, 704
_sp_utl_columnIsId, 706
_sp_utl_columnIsNumeric, 707
_sp_utl_columnListExists, 709
_sp_utl_columnsEqualTypes, 710
_sp_utl_dropAllAggregates, 712
_sp_utl_dropAllFunctions, 713
_sp_utl_dropAllLike, 714
_sp_utl_dropAllProcedures, 714
_sp_utl_dropAllUDX, 715
_sp_utl_functionExists, 716
_sp_utl_getColumnType, 717
_sp_utl_getTableSize, 719
_sp_utl_isTempTable, 720
_sp_utl_justExecute, 721
_sp_utl_procedureExists, 722
_sp_utl_relationExists, 724
_sp_utl_sequenceExists, 725
_sp_utl_tableExists, 726
_sp_utl_viewExists, 727

A
ACC, 27
ALTER_MODEL, 29
ANOVA_CRD_TEST, 32
ANOVA_RBD_TEST, 38
APPLY_DISC, 45
ARULE, 49

B
BITABLE, 53
BTBNET_GROW, 55

C
CANONICAL_CORR, 59
CANONICAL_CORR_AGG, 62
CBFS_AGG, 65
CERROR, 66
CHISQ_TEST, 68
CHISQ_TEST_AGG, 71
CHISQ_TEST_S_AGG, 73
CLEANUP, 74
CMATRIX_ACC, 75
CMATRIX_STATS, 77
CMATRIX_WACC, 80
COL2TRCV_MANOVA_ONE WAY_TEST, 82
COL2TRCV_MANOVA_TWO WAY_TEST, 90
COLUMN_PROPERTIES, 98
COLUMN_PROPERTIES_CHECK, 101
COND_ENTROPY, 102
CONFUSION_MATRIX, 104
COPY_MODEL, 107
CORR, 109
CORR_AGG, 112
CORR_MATRIX_AGG, 114
CORRELATION1000MATRIX, 115
CORRELATION500PAIRS, 119
COV, 122
COV_AGG, 124
COV_MATRIX_AGG, 125
COVARIANCE1000MATRIX, 127
COVARIANCE500PAIRS, 131
COVARIANCEMATRIX, 134
CROSS_VALIDATION, 137
CUMULATIVE, 139

D
DBERN, 145
DBETA, 146
DBINOM, 148
DCAUCHY, 149
DCHISQ, 150
DECTREE, 152
DENSITY, 156
DEXP, 159
DF, 160
DFISK, 161
DGAMMA, 163
DGEOM, 164
DHYPER, 165
Index

DIVCLUSTER, 168
DLNORM, 172
DLOGIS, 174
DMWW, 175
DNBINOM, 177
DNORM, 178
DNORM3P, 179
DPOIS, 181
drand64, 729
DROP_ALL_MODELS, 182
DROP_MODEL, 183
DROP_SUMMARY1000, 185
DROP_TABLE, 729
DT, 186
DUNIF, 187
DWALD, 188
DWEIBULL, 189
DWILCOX, 190

E
EFDISC, 192
EMDISC, 196
ENTROPY, 199
EWDISC, 201
EWDISC_NICE, 205
EXPORT_MODEL, 208
EXPORT_PMML, 210

F
FMEASURE, 212
FPR, 215

G
GET_COLUMN_LIST, 217
GLM, 218
GRANT_MODEL, 222
GROW_DECTREE, 225
GROW_REGTREE, 229

H
HIST, 232

I
IMPORT_MODEL, 238
IMPUTE_DATA, 241
INITIALIZE, 243
IQR, 244
IS_INITIALIZED, 245
ISDATE_TINY, 731

J
JOINT_ENTROPY, 247

K
KMEANS, 249
KNN, 253
KURTOSIS_AGG, 255

L
LDF_MANOVA_ONE_WAY_TEST, 256
LDF_MANOVA_TWO_WAY_TEST, 257
LINEAR_REGRESSION, 259
LIST_COLPROPS, 264
LIST_COMPONENTS, 267
LIST_MODELS, 269
LIST_PARAMS, 272
LIST_PRIVILEGES, 274

M
MAE, 275
MANOVA_ONE_WAY_TEST, 278
MANOVA_Two_WAY_TEST, 295
MEDIAN, 299
MEDIAN_DISC, 300
METADATA_ANALYZE, 301
MIGRATE_MODEL, 304
MODEL_EXISTS, 304
MOMENTS, 306
MSE, 310
msghelp, 733
MTBNET_DIFF, 312
MTBNET_GROW, 314
MUTUALINFO, 317
<table>
<thead>
<tr>
<th>Index</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>MUTUALINFO_AGG</td>
<td>319</td>
</tr>
<tr>
<td>MWW_TEST</td>
<td>321</td>
</tr>
<tr>
<td>N</td>
<td></td>
</tr>
<tr>
<td>NAIVEBAYES</td>
<td>324</td>
</tr>
<tr>
<td>O</td>
<td></td>
</tr>
<tr>
<td>OUTLIERS</td>
<td>327</td>
</tr>
<tr>
<td>P</td>
<td></td>
</tr>
<tr>
<td>PBERN</td>
<td>329</td>
</tr>
<tr>
<td>PBERN_H</td>
<td>331</td>
</tr>
<tr>
<td>PBETA</td>
<td>332</td>
</tr>
<tr>
<td>PBETA_H</td>
<td>334</td>
</tr>
<tr>
<td>PBINOM</td>
<td>335</td>
</tr>
<tr>
<td>PBINOM_H</td>
<td>337</td>
</tr>
<tr>
<td>PCA</td>
<td>339</td>
</tr>
<tr>
<td>PCAUCHY</td>
<td>342</td>
</tr>
<tr>
<td>PCAUCHY_H</td>
<td>344</td>
</tr>
<tr>
<td>PCHISQ</td>
<td>345</td>
</tr>
<tr>
<td>PCHISQ_H</td>
<td>346</td>
</tr>
<tr>
<td>PCHISQ_S</td>
<td>347</td>
</tr>
<tr>
<td>PERCENTAGE_SPLIT</td>
<td>350</td>
</tr>
<tr>
<td>PEXP</td>
<td>352</td>
</tr>
<tr>
<td>PEXP_H</td>
<td>353</td>
</tr>
<tr>
<td>PF</td>
<td>354</td>
</tr>
<tr>
<td>PF_H</td>
<td>356</td>
</tr>
<tr>
<td>PFISK</td>
<td>357</td>
</tr>
<tr>
<td>PFISK_H</td>
<td>358</td>
</tr>
<tr>
<td>PGAMMA</td>
<td>360</td>
</tr>
<tr>
<td>PGAMMA_H</td>
<td>361</td>
</tr>
<tr>
<td>PGEOM</td>
<td>363</td>
</tr>
<tr>
<td>PGEOM_H</td>
<td>364</td>
</tr>
<tr>
<td>PHYPER</td>
<td>365</td>
</tr>
<tr>
<td>PHYPER_H</td>
<td>368</td>
</tr>
<tr>
<td>PLNORM</td>
<td>370</td>
</tr>
<tr>
<td>PLNORM_H</td>
<td>371</td>
</tr>
<tr>
<td>PLOGIS</td>
<td>373</td>
</tr>
<tr>
<td>PLOGIS_H</td>
<td>374</td>
</tr>
<tr>
<td>PMML_DECTREE</td>
<td>375</td>
</tr>
<tr>
<td>PMML_MODEL</td>
<td>377</td>
</tr>
<tr>
<td>PMML_NAIVEBAYES</td>
<td>379</td>
</tr>
<tr>
<td>PMWW</td>
<td>381</td>
</tr>
<tr>
<td>PMWW_H</td>
<td>383</td>
</tr>
<tr>
<td>PNBINOM</td>
<td>384</td>
</tr>
<tr>
<td>PNBINOM_H</td>
<td>386</td>
</tr>
<tr>
<td>PNORM</td>
<td>387</td>
</tr>
<tr>
<td>PNORM3P</td>
<td>388</td>
</tr>
<tr>
<td>PNORM_H</td>
<td>390</td>
</tr>
<tr>
<td>PPOINT</td>
<td>391</td>
</tr>
<tr>
<td>PPOIS</td>
<td>395</td>
</tr>
<tr>
<td>PPOIS_H</td>
<td>396</td>
</tr>
<tr>
<td>PPV</td>
<td>397</td>
</tr>
<tr>
<td>PREDICT_ARULE</td>
<td>399</td>
</tr>
<tr>
<td>PREDICT_DECTREE</td>
<td>405</td>
</tr>
<tr>
<td>PREDICT_DIVCLUSTER</td>
<td>407</td>
</tr>
<tr>
<td>PREDICT_GLM</td>
<td>410</td>
</tr>
<tr>
<td>PREDICT_KMEANS</td>
<td>412</td>
</tr>
<tr>
<td>PREDICT_KNN</td>
<td>414</td>
</tr>
<tr>
<td>PREDICT_LINEAR_REGRESSION</td>
<td>417</td>
</tr>
<tr>
<td>PREDICT_NAIVEBAYES</td>
<td>420</td>
</tr>
<tr>
<td>PREDICT_REGTREE</td>
<td>423</td>
</tr>
<tr>
<td>PREDICT_TWOSTEP</td>
<td>425</td>
</tr>
<tr>
<td>PRINT_ARULE</td>
<td>427</td>
</tr>
<tr>
<td>PRINT_DECTREE</td>
<td>432</td>
</tr>
<tr>
<td>PRINT_GLM</td>
<td>434</td>
</tr>
<tr>
<td>PRINT_KMEANS</td>
<td>438</td>
</tr>
<tr>
<td>PRINT_MANOVA_ONE_WAY_TEST</td>
<td>441</td>
</tr>
<tr>
<td>PRINT_MANOVA_Two_WAY_TEST</td>
<td>446</td>
</tr>
<tr>
<td>PRINT_MODEL</td>
<td>447</td>
</tr>
<tr>
<td>PRINT_REGTREE</td>
<td>449</td>
</tr>
<tr>
<td>PRINT_TIMESERIES</td>
<td>451</td>
</tr>
<tr>
<td>PRINT_TWOSTEP</td>
<td>454</td>
</tr>
<tr>
<td>PROJECT_PCA</td>
<td>457</td>
</tr>
<tr>
<td>PRUNE_DECTREE</td>
<td>459</td>
</tr>
<tr>
<td>PRUNE_REGTREE</td>
<td>462</td>
</tr>
<tr>
<td>PT</td>
<td>463</td>
</tr>
<tr>
<td>PT_H</td>
<td>466</td>
</tr>
<tr>
<td>PUNIF</td>
<td>467</td>
</tr>
<tr>
<td>PUNIF_H</td>
<td>469</td>
</tr>
<tr>
<td>PWALD</td>
<td>470</td>
</tr>
<tr>
<td>PWALD_H</td>
<td>471</td>
</tr>
<tr>
<td>PWEIBULL</td>
<td>473</td>
</tr>
<tr>
<td>PWEIBULL_H</td>
<td>474</td>
</tr>
<tr>
<td>PWILCOX</td>
<td>475</td>
</tr>
<tr>
<td>PWILCOX_H</td>
<td>476</td>
</tr>
</tbody>
</table>
TANET_CLASSAPPLY, 655
TANET_GROW, 658
TBNET1G, 661
TBNET1G2P, 664
TBNET2G, 667
TBNET_APPLY, 671
TBNET_GROW, 674
TIMESERIES, 678
TPR, 682
TRAIN_TEST, 685
TWOSTEP, 687

U
UNITABLE, 691

V
VERIFY_ARULE, 693

W
WACC, 694
WILCOXON_TEST, 696