IBM SPSS Modeler 18.4 Python Handbuch für Scripterstellung und Automatisierung

Hinweis

Vor Verwendung dieser Informationen und des darin beschriebenen Produkts sollten die Informationen unter „Bemerkungen” auf Seite 487 gelesen werden.

Produktinformation

Diese Ausgabe bezieht sich auf Version 18, Release 4, Modifikation 0 von IBM® SPSS Modeler und auf alle nachfolgenden Releases und Modifikationen, bis dieser Hinweis in einer Neuausgabe geändert wird.

© Copyright International Business Machines Corporation .

Inhaltsverzeichnis

Kapitel 1. Scripts und die Scriptsprache 1
Scripting - Übersicht 1
Scripttypen 1
Stream-Scripts 1
Beispiel für Stream-Script: Trainieren eines neuronalen Netzes 3
Größenbegrenzungen für Jython-Code 4
Standalone-Scripts 4
Beispiel für Standalone-Script: Speichern und Laden von Modellen 4
Beispiel für Standalone-Script: Generieren eines Merkmalauswahlmodells 5
Superknotenscripts 5
Beispiel für Superknotenscript 6
Verwendung von Schleifen und bedingte Ausführung in Streams 6
Verwendung von Schleifen in Streams 7
Bedingte Ausführung in Streams 11
Ausführen und Unterbrechen von Scripts 12
Suchen und Ersetzen 13

Kapitel 2. Scriptsprache 17
Scriptsprache - Übersicht 17
Python und Jython 17
Python-Scripting 18
Vorgänge 18
Listen 18
Zeichenfolgen 19
Anmerkungen 21
Anweisungssyntax 21
IDs 21
Codeblöcke 22
Übergeben von Argumenten an ein Script 22
Beispiele 23
Mathematische Methoden 23
Verwendung von Nicht-ASCII-Zeichen 25
Objektorientierte Programmierung 26
Definieren einer Klasse 27
Erstellen einer Klasseninstanz 27
Hinzufügen von Attributen zu einer Klasseninstanz 27
Definieren von Klassenattributen und Methoden 27
Ausgeblendete Variablen 28
Vererbung 28

Kapitel 3. Scripting in IBM SPSS Modeler 31
Scripttypen 31
Streams, Superknotenstreams und Diagramme 31
Streams 31
Superknotenstreams 31
Diagramme 31
Ausführen eines Streams 31
Scripting-Kontext 32
Referenzieren vorhandener Knoten 33
Suchen von Knoten 33

Festlegen von Eigenschaften ... 34
Erstellen von Knoten und Ändern von Streams ... 35
Erstellen von Knoten ... 36
Aktivieren und Aufheben von Links für Knoten ... 36
Importieren, Ersetzen und Löschen von Knoten ... 38
Traversieren durch Knoten in einem Stream ... 38
Entfernen von Elementen ... 39
Abrufen von Informationen zu Knoten ... 40

Kapitel 4. Scripting-API ... 43
Einführung in die Scripting-API ... 43
Beispiel 1: Suchen nach Knoten mit einem benutzerdefinierten Filter ... 43
Beispiel 2: Benutzern ermöglichen, Verzeichnis- oder Dateiinformationen basierend auf ihren Berechtigungen abzurufen ... 43
Metadaten: Informationen zu Daten ... 44
Zugriff auf generierte Objekte ... 47
Fehler bearbeiten ... 48
Stream-, Sitzungs- und Superknotenparameter ... 49
Globale Werte ... 53
Arbeiten mit mehreren Streams: Standalone-Scripts ... 54

Kapitel 5. Tipps zum Scripting ... 55
Ändern der Streamausführung ... 55
Verwendung von Schleifen bei Knoten ... 55
Zugriff auf Objekte in IBM SPSS Collaboration and Deployment Services Repository ... 56
Erstellen eines verschlüsselten Kennworts ... 58
Scriptprüfung ... 58
Scripting über die Befehlszeile ... 59
Kompatibilität mit früheren Releases ... 59
Zugriff auf Streamausführungsergebnisse ... 59
Tabelleninhaltsmodell ... 60
XML-Inhaltsmodell ... 62
JSON-Inhaltsmodell ... 63
Inhaltsmodell für Spaltenstatistiken und Inhaltsmodell für paarweise Statistikdaten ... 65

Kapitel 6. Befehlszeilenargumente ... 69
Aufrufen der Software ... 69
Verwenden von Befehlszeilenargumenten ... 69
Systemargumente ... 70
Parameterargumente ... 71
Argumente zum Herstellen einer Serververbindung ... 72
IBM SPSS Collaboration and Deployment Services Repository-Verbindungsargumente ... 74
Argumente zum Herstellen einer Verbindung zu IBM SPSS Analytic Server ... 74
Kombinieren mehrerer Argumente ... 75

Kapitel 7. Eigenschaftsreferenz ... 77
Eigenschaftsreferenz - Übersicht ... 77
Syntax für Eigenschaften ... 77
Beispiele für Knoten- und Streameigenschaften ... 79
Knoteneigenschaften - Übersicht ... 79
Allgemeine Knoteneigenschaften ... 80

Kapitel 8. Streameigenschaften ... 81

Kapitel 9. Eigenschaften von Quellenknoten ... 85
Allgemeine Eigenschaften von Quellenknoten ... 85

Eigenschaften von "asimport" ... 92
Eigenschaften des Knotens "cognosimport" ... 93
Eigenschaften von "databasenode" ... 97
Eigenschaften von "datacollectionimportnode" ... 99
Eigenschaften von "excelimportnode" ... 103
extensionimportnode, Eigenschafen ... 104
Eigenschaften von "fixedfilenode" ... 107
Eigenschaften des Knotens "gsdata_import" ... 111
Eigenschaften von "jsonimportnode" ... 112
Eigenschaften von "sasimportnode" ... 112
simgennode, Eigenschaften ... 113
Eigenschaften von "statisticsimportnode" ... 115
Eigenschaften des Knotens "tm1odataimport" ... 115
Eigenschaften des Knotens "tm1import" (nicht mehr unterstützt) ... 116
twcimport, Knoteneigenschaften ... 117
userinputnode, Eigenschaften ... 118
Eigenschaften von "variablefilenode" ... 119
Eigenschaften von "xmlimportnode" ... 125

Kapitel 10. Eigenschaften von Datensatzoperationsknoten ... 127
appendnode, Eigenschaften ... 127
aggregatenode, Eigenschaften ... 127
balancenode, Eigenschaften ... 129
cplexoptnode, Eigenschaften ... 129
Eigenschaften von "derive_stbnode" ... 133
distinctnode, Eigenschaften ... 135
extensionprocessnode, Eigenschaften ... 137
mergenode, Eigenschaften ... 138
rfmaggregatenode, Eigenschaften ... 140
samplenode, Eigenschaften ... 143
selectnode, Eigenschaften ... 145
sortnode, Eigenschaften ... 146
spacetimeboxes, Eigenschaften ... 146
Eigenschaften von "streamingtimeseries" ... 148

Kapitel 11. Eigenschaften von Feldoperationsknoten ... 159
anonymizenode, Eigenschaften ... 159
Eigenschaften von "autodataprepnode" ... 160
astimeintervalsnode, Eigenschaften ... 164
binningnode, Eigenschaften ... 165
derivenode, Eigenschaften ... 168
ensemblenode, Eigenschaften ... 172
fillernode, Eigenschaften ... 173
Eigenschaften von "filternode" ... 174
historynode, Eigenschaften ... 175
partitionnode, Eigenschaften ... 176
reclassifynode, Eigenschaften ... 177
reordernode, Eigenschaften ... 178
reprojectnode, Eigenschaften ... 179
restructurenode, Eigenschaften ... 179
rfmanalysisnode, Eigenschaften ... 180
settoflagnode, Eigenschaften ... 182
Eigenschaften von "statisticstransformnode" ... 183
Eigenschaften von "timeintervalsnode" (nicht mehr unterstützt) ... 183
transposenode, Eigenschaften ... 189
Eigenschaften von "typenode" ... 191

Kapitel 12. Eigenschaften von Diagrammknoten..... 199
Allgemeine Eigenschaften von Diagrammknoten.....199
Eigenschaften von "collectionnode"..... 200
Eigenschaften von "distributionnode"..... 201
Eigenschaften von "evaluationnode"..... 202
Eigenschaften von "graphboardnode"..... 204
Eigenschaften von "histogramnode".....209
mapvisualization, Eigenschaften.....210
Eigenschaften von "multiplotnode"..... 215
Eigenschaften von "plotnode".....216
Eigenschaften von "timeplotnode"..... 219
Eigenschaften von "eplotnode".....220
Eigenschaften von "tsnenode"..... 221
Eigenschaften von "webnode"..... 223

Kapitel 13. Eigenschaften von Modellierungsknoten..... 225
Allgemeine Eigenschaften von Modellierungsknoten..... 225
anomalydetectionnode, Eigenschaften..... 226
apriorinode, Eigenschaften..... 227
associationrulesnode, Eigenschaften..... 229
autoclassifiernode, Eigenschaften..... 232
Festlegen der Algorithmuseigenschaften.....235
autoclusternode, Eigenschaften..... 235
autonumericnode, Eigenschaften..... 237
Eigenschaften von "bayesnetnode"..... 239
c50node, Eigenschaften.....242
carmanode, Eigenschaften.....243
cartnode, Eigenschaften.....244
chaidnode, Eigenschaften..... 247
coxregnode, Eigenschaften..... 250
Eigenschaften von "decisionlistnode".....252
discriminantnode, Eigenschaften.....254
extensionmodelnode, Eigenschaften.....256
factornode, Eigenschaften..... 259
featureselectionnode, Eigenschaften..... 261
genlinnode, Eigenschaften..... 263
Eigenschaften von "glmmnode".....269
Eigenschaften von "gle".....275
kmeansnode, Eigenschaften..... 282
kmeansasnode, Eigenschaften..... 283
knnnode, Eigenschaften..... 284
kohonennode, Eigenschaften.....286
Eigenschaften von "linearnode".....287
Eigenschaften von "linearasnode"..... 289
logregnode, Eigenschaften.....291
lsvmnode, Eigenschaften..... 297
neuralnetnode, Eigenschaften..... 298
neuralnetworknode, Eigenschaften..... 301
questnode, Eigenschaften.....303
randomtrees, Eigenschaften..... 306
regressionnode, Eigenschaften.....308
sequencenode, Eigenschaften..... 311
slrmnode, Eigenschaften.....312
Eigenschaften von "statisticsmodelnode".....313
stpnode, Eigenschaften.....314
svmnode, Eigenschaften..... 320

Eigenschaften von "tcmnode"....322
ts, Eigenschaften....327
treeas, Eigenschaften....338
Eigenschaften von "twostepnode"....341
Eigenschaften von "twostepAS"....342

Kapitel 14. Eigenschaften von Modellnuggetknoten....345
Eigenschaften von "applyanomalydetectionnode"....345
Eigenschaften von "applyapriorinode"....345
Eigenschaften von "applyassociationrulesnode"....346
Eigenschaften von "applyautoclassifiernode"....347
Eigenschaften von "applyautoclusternode"....349
Eigenschaften von "applyautonumericnode"....349
Eigenschaften von "applybayesnetnode"....349
Eigenschaften von "applyc50node"....350
Eigenschaften von "applycarmanode"....350
Eigenschaften von "applycartnode"....350
Eigenschaften von "applychaidnode"....351
Eigenschaften von "applycoxregnode"....351
Eigenschaften von "applydecisionlistnode"....352
Eigenschaften von "applydiscriminantnode"....352
applyextension, Eigenschaften....353
Eigenschaften von "applyfactornode"....354
Eigenschaften von "applyfeatureselectionnode"....354
Eigenschaften von "applygeneralizedlinearnode"....355
Eigenschaften von "applyglmmnode"....355
Eigenschaften von "applygle"....356
Eigenschaften von "applygmm"....356
Eigenschaften von "applykmeansnode"....357
Eigenschaften von "applyknnnode"....357
Eigenschaften von "applykohonennode"....357
Eigenschaften von "applylinearnode"....357
Eigenschaften von "applylinearasnode"....358
Eigenschaften von "applylogregnode"....358
Eigenschaften von "applylsvmnode"....358
Eigenschaften von "applyneuralnetnode"....359
applyneuralnetworknode, Eigenschaften....359
Eigenschaften von "applyocsvmnode"....360
Eigenschaften von "applyquestnode"....360
Eigenschaften von "applyrandomtrees"....361
Eigenschaften von "applyregressionnode"....362
Eigenschaften von "applyselflearningnode"....362
Eigenschaften von "applysequencenode"....362
Eigenschaften von "applysvmnode"....363
Eigenschaften von "applystpnode"....363
Eigenschaften von "applytcmnode"....363
Eigenschaften von "applyts"....363
Eigenschaften des Knotens "applytimeseriesnode" (nicht mehr unterstützt)....364
Eigenschaften von "applytreeas"....364
Eigenschaften von "applytwostepnode"....365
Eigenschaften von "applytwostepAS"....365
Eigenschaften von "applyxgboosttreenode"....365
applyxgboostlinearnode, Eigenschaften....366
Eigenschaften von "hdbscannugget"....366
Eigenschaften von "kdeapply"....366

Kapitel 15. Eigenschaften von Datenbankmodellierungsknoten....367

Knoteneigenschaften für Microsoft-Modellierung....367
Eigenschaften von Microsoft-Modellierungsknoten....367
Eigenschaften von Microsoft-Modellnuggets....369
Knoteneigenschaften für Oracle-Modellierung....371
Eigenschaften von Oracle-Modellierungsknoten....372
Eigenschaften von Oracle-Modellnuggets....378
Knoteneigenschaften für IBM Netezza Analytics-Modellierung....379
Eigenschaften von Netezza-Modellierungsknoten....379
Eigenschaften von Netezza-Modellnuggets....396

Kapitel 16. Eigenschaften von Ausgabeknoten....399
analysisnode, Eigenschaften....399
dataauditnode, Eigenschaften....400
extensionoutputnode, Eigenschaften....402
kdeexport, Eigenschaften....404
matrixnode, Eigenschaften....405
meansnode, Eigenschaften....407
reportnode, Eigenschaften....409
setglobalsnode, Eigenschaften....411
simevalnode, Eigenschaften....412
simfitnode, Eigenschaften....413
statisticsnode, Eigenschaften....414
Eigenschaften von "statisticsoutputnode"....415
tablenode, Eigenschaften....415
transformnode, Eigenschaften....418

Kapitel 17. Eigenschaften von Exportknoten....421
Allgemeine Eigenschaften von Exportknoten....421
Eigenschaften von "asexport"....421
Eigenschaften von "cognosexportnode"....422
databaseexportnode, Eigenschaften....424
Eigenschaften von "datacollectionexportnode"....429
Eigenschaften von "excelexportnode"....430
Eigenschaften von "extensionexportnode"....431
Eigenschaften von "jsonexportnode"....432
Eigenschaften von "outputfilenode"....433
Eigenschaften von "sasexportnode"....434
Eigenschaften von "statisticsexportnode....434
Eigenschaften des Knotens "tm1odataexport"....435
Eigenschaften des Knotens "tm1export" (nicht mehr unterstützt)....437
Eigenschaften von "xmlexportnode"....439

Kapitel 18. Eigenschaften von IBM SPSS Statistics-Knoten....441
Eigenschaften von "statisticsimportnode"....441
Eigenschaften von "statisticstransformnode"....441
Eigenschaften von "statisticsmodelnode"....442
Eigenschaften von "statisticsoutputnode"....443
Eigenschaften von "statisticsexportnode....443

Kapitel 19. Eigenschaften von Python-Knoten....445
Eigenschaften von "gmm"....445
Eigenschaften von "hdbscannode"....446
Eigenschaften von "kdemodel"....448
kdeexport, Eigenschaften....449
Eigenschaften von "gmm"....450
Eigenschaften von "ocsvmnode"....451
rfnode, Eigenschaften....454

Eigenschaften von "smotenode"....456
Eigenschaften von "tsnenode"....457
Eigenschaften von "xgboostlinearnode"....459
Eigenschaften von "xgboosttreenode"....460

Kapitel 20. Eigenschaften des Spark-Knotens....463
Eigenschaften von "isotonicasnode"....463
kmeansasnode, Eigenschaften....463
Eigenschaften von "multilayerperceptronnode"....465
Eigenschaften von "xgboostasnode"....465

Kapitel 21. Superknoteneigenschaften....469

Anhang A. Knotennamenreferenz....471
Modellnuggetnamen....471
Vermeidung doppelter Modellnamen....473
Namen der Ausgabetypen....473

Anhang B. Migration von traditionellem Scripting zu Python-Scripting....475
Übersicht über die Migration traditioneller Scripts....475
Allgemeine Unterschiede....475
Scripting-Kontext....475
Befehle und Funktionen....475
Literale und Kommentare....476
Operatoren....476
Bedingte Befehle und Schleifenbefehle....477
Variablen....478
Knoten-, Ausgabe- und Modelltypen....478
Eigenschaftsnamen....479
Knotenreferenzen....479
Abrufen und Festlegen von Eigenschaften....479
Bearbeiten von Streams....480
Knotenoperationen....481
Verwendung von Schleifen....481
Streams ausführen....482
Zugriff auf Objekte über das Dateisystem und das Repository....483
Streamoperationen....483
Modelloperationen....484
Dokumentausgabeoperationen....484
Weitere Unterschiede zwischen traditionellem Scripting und Python-Scripting....485

Bemerkungen....487
Marken....488
Bedingungen für Produktdokumentation....489

Index....491

Kapitel 1. Scripts und die Scriptsprache

Scripting - Übersicht

Die Scripterstellung in IBM SPSS Modeler ist ein leistungsstarkes Tool, mit dem Prozesse in der Benutzerschnittstelle automatisiert werden. Scripts können dieselben Arten von Aktionen durchführen, die Sie mit einer Maus oder einer Tastatur durchführen. So können Sie Aufgaben automatisieren, die bei einer manuellen Durchführung sehr viele Wiederholungen verlangen oder sehr viel Zeit beanspruchen.

Scripts können zu folgenden Zwecken verwendet werden:

- Eine bestimmte Reihenfolge für die Knotenausführung in einem Stream erzwingen.
- Die Eigenschaften von Knoten festlegen und Ableitungen durchführen, indem Sie ein Subset von CLEM (Control Language for Expression Manipulation) verwenden.
- Eine automatische Abfolge von Aktionen festlegen, für die normalerweise Benutzeraktivitäten erforderlich sind. So können Sie beispielsweise ein Modell erstellen und dieses anschließend testen.
- Komplexe Prozesse einrichten, für die häufige Interventionen des Benutzers notwendig sind, wie dies beispielsweise bei Kreuzvalidierungen der Fall ist, bei denen ein Modell wiederholt generiert und getestet werden muss.
- Prozesse einrichten, mit denen Streams bearbeitet werden. Sie können zum Beispiel einen Modelltrainings-Stream ausführen und automatisch den entsprechenden Modelltest-Stream erstellen.

In diesem Kapitel finden Sie allgemeine Beschreibungen und Beispiele für Scripts auf der Streamebene, Standalone-Scripts und Scripts innerhalb von Superknoten auf der IBM SPSS Modeler-Benutzerschnittstelle. Weitere Informationen zu Scriptsprache, Syntax und Befehlen finden Sie in den nachfolgenden Kapiteln.

Anmerkung:

Sie können keine Scripts importieren und ausführen, die in IBM SPSS Statistics innerhalb von IBM SPSS Modeler erstellt wurden.

Scripttypen

IBM SPSS Modeler verwendet drei Scripttypen:

- **Stream-Scripts** werden als Streameigenschaft gespeichert und daher zusammen mit einem bestimmten Stream gespeichert und geladen. Beispielsweise können Sie ein Stream-Script schreiben, das das Trainieren und Anwenden eines Modellnuggets automatisiert. Außerdem können Sie angeben, dass bei jeder Ausführung eines bestimmten Streams statt des Inhalts des Streamerstellungsbereichs das Script ausgeführt werden soll.
- **Standalone-Scripts** sind mit keinem bestimmten Stream verknüpft und werden in externen Textdateien gespeichert. Mit einem Standalone-Script können beispielsweise mehrere Streams gemeinsam bearbeitet werden.
- **Superknotenscripts** werden als Streameigenschaft von Superknoten gespeichert. Superknotenscripts stehen nur in Endsuperknoten zur Verfügung. Mit Superknotenscripts kann die Ausführungssequenz der Superknoteninhalte gesteuert werden. Bei Superknoten, bei denen es sich nicht um Endknoten handelt (also Quellen- oder Prozessknoten), können Sie Eigenschaften für den Superknoten bzw. die Knoten, die er enthält, direkt im Stream-Script definieren.

Stream-Scripts

Mit Scripts können in einem bestimmten Stream enthaltene Operationen angepasst und zusammen mit dem Stream gespeichert werden. Stream-Scripts können verwendet werden, um eine bestimmte Ausfüh-

rungsreihenfolge der in einem Stream enthaltenen Endknoten vorzugeben. Die Bearbeitung des mit dem aktuellen Stream gespeicherten Scripts erfolgt im Dialogfeld "Script" des Streams.

So können Sie im Dialogfeld "Streameigenschaften" auf die Registerkarte für das Stream-Script zugreifen:

1. Wählen Sie im Menü **Tools** Folgendes aus:

   Streameigenschaften > **Ausführung**
2. Klicken Sie auf die Registerkarte **Ausführung**, um mit den Scripts des aktuellen Streams zu arbeiten.

Verwenden Sie die Symbolleistensymbole oben im Dialogfeld für das Stream-Script für die folgenden Operationen:

- Inhalte eines bereits vorhandenen Standalone-Scripts in das Fenster importieren.
- Script als Textdatei speichern.
- Script drucken.
- Standardscript anhängen.
- Script bearbeiten (Rückgängig machen, Ausschneiden, Kopieren, Einfügen und andere gängige Editierfunktionen).
- Gesamtes aktuelles Script ausführen.
- Ausgewählte Zeilen eines Scripts ausführen.
- Script während der Ausführung stoppen. (Dieses Symbol ist nur während einer Scriptausführung aktiviert.)
- Syntax des Scripts überprüfen und etwaige Fehler zur Untersuchung im unteren Bereich des Dialogfelds anzeigen.

Anmerkung: Ab Version 16.0 verwendet SPSS Modeler die Scripting-Sprache Python. Alle Versionen vor Version 16.0 verwendeten eine spezielle Scripting-Sprache von SPSS Modeler, die jetzt als traditionelles Scripting bezeichnet wird. Wählen Sie je nach Typ des verwendeten Scripts auf der Registerkarte **Ausführung** den Ausführungsmodus **Standard (optionales Script)** und dann entweder **Python** oder **Traditionell** aus.

Sie können angeben, ob ein Script während der Ausführung eines Streams ausgeführt wird oder nicht. Wählen Sie **Dieses Script ausführen** aus, um das Script bei jeder Ausführung des Streams unter Beachtung der Ausführungsreihenfolge des Scripts auszuführen. Diese Einstellung führt zu einer Automatisierung auf Streamebene und sorgt für eine schnellere Modellbildung. In der Standardeinstellung wird das Script allerdings während der Streamausführung ignoriert. Auch wenn Sie die Option **Dieses Script ignorieren** auswählen, können Sie das Script stets direkt über dieses Dialogfeld ausführen.

Der Scripteditor umfasst die folgenden Funktionen zur Unterstützung von Script-Authoring:

- Syntaxhervorhebung; Schlüsselwörter, Literalwerte (wie Zeichenfolgen und Zahlen) und Kommentare werden hervorgehoben.
- Zeilennummerierung.
- Blockabgleich; wenn der Cursor an den Anfang eines Programmblocks gesetzt wird, wird der entsprechende Endblock ebenfalls hervorgehoben.
- Vorschlag für Auto-Vervollständigen.

Die von der Syntaxhervorhebung verwendeten Farben und Textstile können mit den Anzeigevorgaben von IBM SPSS Modeler angepasst werden. Sie können auf die Anzeigevorgaben zugreifen, indem Sie **Tools** > **Optionen** > **Benutzeroptionen** und anschließend die Registerkarte **Syntax** auswählen.

Eine Liste vorgeschlagener Syntaxvervollständigungen kann aufgerufen werden, indem Sie im Kontextmenü **Automatisch vorschlagen** auswählen oder Strg+Leertaste drücken. Mit den Cursortasten können Sie sich in der Liste nach oben und unten bewegen. Zum Einfügen des ausgewählten Texts drücken Sie dann die Eingabetaste. Drücken Sie Esc, um die automatischen Vorschläge zu verlassen, ohne den vorhandenen Text zu ändern.

Die Registerkarte **Debug** zeigt Debugnachrichten an und kann verwendet werden, um den Scriptstatus auszuwerten, sobald das Script ausgeführt wird. Die Registerkarte **Debug** besteht aus einem schreibge-

schützten Textbereich und einem einzeiligen Eingabetextfeld. Der Textbereich zeigt Text an, der von den Scripts an die Standardausgabe oder an die Standard-Fehlerausgabe gesendet wird, z. B. über Fehlernachrichtentext. Das Eingabetextfeld übernimmt die Eingabe des Benutzers. Diese Eingabe wird dann im Kontext des Scripts ausgewertet, das zuletzt im Dialog ausgeführt wurde (so genannter *Scripting-Kontext*). Der Textbereich enthält den Befehl und die resultierende Ausgabe, sodass der Benutzer einen Trace der Befehle sehen kann. Das Eingabetextfeld enthält immer die Eingabeaufforderung (- -> für traditionelles Scripting).

In den folgenden Fällen wird ein neuer Scripting-Kontext erstellt:

- Ein Script wird ausgeführt, indem Sie **Dieses Script ausführen** oder **Nur ausgewählte Zeilen ausführen** auswählen.
- Die Scriptsprache wird geändert.

Wenn ein neuer Scripting-Kontext erstellt wird, wird der Inhalt des Textbereichs gelöscht.

Anmerkung: Durch die Ausführung eines Streams außerhalb des Scriptbereichs wird der Scriptkontext des Scriptbereichs nicht geändert. Die Werte von Variablen, die als Teil dieser Ausführung erzeugt werden, sind im Scriptdialogfeld nicht sichtbar.

Beispiel für Stream-Script: Trainieren eines neuronalen Netzes

Ein Stream kann verwendet werden, um bei der Ausführung ein neuronales Netzmodell zu trainieren. Um das Modell zu testen, müssen Sie den Modellknoten ausführen, um das Modell in den Stream einzufügen, die entsprechenden Verbindungen herzustellen und einen Analyseknoten auszuführen.

Mit einem IBM SPSS Modeler-Script können Sie das Testen des Modellnuggets nach seiner Erstellung automatisieren. Das folgende Stream-Script zum Testen des Demo-Streams `druglearn.str` (im Ordner `/Demos/streams/` Ihrer IBM SPSS Modeler-Installation verfügbar) könnte beispielsweise über das Dialogfeld "Streameigenschaften" (**Tools > Streameigenschaften > Script**) ausgeführt werden:

```
stream = modeler.script.stream()
neuralnetnode = stream.findByType("neuralnetwork", None)
results = []
neuralnetnode.run(results)
appliernode = stream.createModelApplierAt(results[0], "Drug", 594, 187)
analysisnode = stream.createAt("analysis", "Drug", 688, 187)
typenode = stream.findByType("type", None)
stream.linkBetween(appliernode, typenode, analysisnode)
analysisnode.run([])
```

In der folgenden Auflistung werden die einzelnen Zeilen dieses Scriptbeispiels beschrieben.

- Die erste Zeile definiert eine Variable, die auf den aktuellen Stream verweist.
- In Zeile 2 sucht das Script den neuronalen Netzbuilderknoten.
- In Zeile 3 erstellt das Script eine Liste, in der die Ausführungsergebnisse gespeichert werden können.
- In Zeile 4 wird das neuronale Netzmodellnugget erstellt. Es wird in der Liste gespeichert, die in Zeile 3 definiert wird.
- In Zeile 5 wird ein Modellanwendungsknoten für das Modellnugget erstellt und im Streamerstellungsbereich angeordnet.
- In Zeile 6 wird der Analyseknoten `Drug` erstellt.
- In Zeile 7 sucht das Script den Typknoten.
- In Zeile 8 stellt das Script eine Verbindung zum in Zeile 5 erstellten Modellanwendungsknoten zwischen dem Typknoten und Analyseknoten her.
- Schließlich wird der Analyseknoten ausgeführt, um den Analysebericht zu erstellen.

Sie können mithilfe eines Scripts einen völlig neuen Stream, ausgehend von einem leeren Erstellungsbereich, erstellen und ausführen.

Größenbegrenzungen für Jython-Code

Jython kompiliert jedes Script in Java-Bytecode, der dann von Java Virtual Machine (JVM) ausgeführt wird. In Java gilt allerdings einen Grenzwert für die Größe einer einzelnen Bytecodedatei. Wenn Jython versucht, den Bytecode zu laden, kann dies einen Absturz der JVM verursachen. IBM SPSS Modeler kann dies nicht verhindern.

Stellen Sie sicher, dass Sie Ihre Jython-Scripts mit gängigen Programmiermethoden schreiben (z. B. Minimieren von doppeltem Code mithilfe von Variablen oder Funktionen zum Berechnen von gemeinsamen Zwischenwerten). Gegebenenfalls müssen Sie Ihren Code auf mehrere Quellendateien aufteilen oder ihn mithilfe von Modulen codieren, weil diese in separate Bytecodedateien kompiliert werden.

Standalone-Scripts

Im Dialogfeld "Standalone-Script" wird ein Script erstellt oder bearbeitet, das als Textdatei gespeichert wird. Es zeigt den Namen der Datei und bietet Optionen zum Laden, Speichern, Importieren und Ausführen von Scripts.

So öffnen Sie das Dialogfeld für Standalone-Scripts:

Wählen Sie im Hauptmenü Folgendes aus:

Tools > **Standalone-Script**

Für Standalone-Scripts stehen dieselbe Symbolleiste und dieselben Optionen zur Überprüfung der Script-Syntax zur Verfügung wie für Stream-Scripts. Weitere Informationen finden Sie im Thema „Stream-Scripts” auf Seite 1.

Beispiel für Standalone-Script: Speichern und Laden von Modellen

Standalone-Scripts sind bei der Streambearbeitung hilfreich. Nehmen wir an, Sie besitzen zwei Streams - einen, der ein Modell erstellt, und einen anderen, der das aus dem ersten Stream mit vorhandenen Datenfeldern generierte Regelset anhand von Diagrammen untersucht. Ein Standalone-Script für dieses Szenario könnte wie folgt aussehen:

```
taskrunner = modeler.script.session().getTaskRunner()

# Modify this to the correct Modeler installation Demos folder.
# Note use of forward slash and trailing slash.
installation = "C:/Program Files/IBM/SPSS/Modeler/19/Demos/"

# First load the model builder stream from file and build a model
druglearn_stream = taskrunner.openStreamFromFile(installation + "streams/druglearn.str", True)
results = []
druglearn_stream.findByType("c50", None).run(results)

# Save the model to file
taskrunner.saveModelToFile(results[0], "rule.gm")

# Now load the plot stream, read the model from file and insert it into the stream
drugplot_stream = taskrunner.openStreamFromFile(installation + "streams/drugplot.str", True)
model = taskrunner.openModelFromFile("rule.gm", True)
modelapplier = drugplot_stream.createModelApplier(model, "Drug")

# Now find the plot node, disconnect it and connect the
# model applier node between the derive node and the plot node
derivenode = drugplot_stream.findByType("derive", None)
plotnode = drugplot_stream.findByType("plot", None)
drugplot_stream.disconnect(plotnode)
modelapplier.setPositionBetween(derivenode, plotnode)
drugplot_stream.linkBetween(modelapplier, derivenode, plotnode)
plotnode.setPropertyValue("color_field", "$C-Drug")
plotnode.run([])
```

Anmerkung: Weitere Informationen zur Scriptsprache im Allgemeinen finden Sie in „Scriptsprache - Übersicht” auf Seite 17.

Beispiel für Standalone-Script: Generieren eines Merkmalauswahlmodells

In diesem Beispiel wird ausgehend von einem leeren Erstellungsbereich ein Stream erstellt, der ein Merkmalauswahlmodell generiert, das Modell anwendet und eine Tabelle erstellt, in der die 15 wichtigsten Ziele in Bezug auf das angegebene Ziel aufgeführt werden.

```
stream = modeler.script.session().createProcessorStream("featureselection",
True)

statisticsimportnode = stream.createAt("statisticsimport", "Statistics
File", 150, 97)
statisticsimportnode.setPropertyValue("full_filename", "$CLEO_DEMOS/custo
mer_dbase.sav")

typenode = stream.createAt("type", "Type", 258, 97)
typenode.setKeyedPropertyValue("direction", "response_01", "Target")

featureselectionnode = stream.createAt("featureselection", "Feature Selecti
on", 366, 97)
featureselectionnode.setPropertyValue("top_n", 15)
featureselectionnode.setPropertyValue("max_missing_values", 80.0)
featureselectionnode.setPropertyValue("selection_mode", "TopN")
featureselectionnode.setPropertyValue("important_label", "Check Me Out!")
featureselectionnode.setPropertyValue("criteria", "Likelihood")

stream.link(statisticsimportnode, typenode)
stream.link(typenode, featureselectionnode)
models = []
featureselectionnode.run(models)

# Assumes the stream automatically places model apply nodes in the stream
applynode = stream.findByType("applyfeatureselection", None)
tablenode = stream.createAt("table", "Table", applynode.getXPosition() + 96,
applynode.getYPosition())
stream.link(applynode, tablenode)
tablenode.run([])
```

Das Script erstellt einen Quellenknoten zum Einlesen der Daten, verwendet einen Typknoten, um die Rolle (Verwendung) des Felds `response_01` auf `Target` zu setzen und erstellt anschließend einen Merkmalauswahlknoten und führt diesen aus. Außerdem verbindet das Script die Knoten und positioniert sie im Streamerstellungsbereich, um ein lesbares Layout zu erstellen. Anschließend wird das resultierende Modellnugget mit einem Tabellenknoten verbunden, in dem die 15 wichtigsten Felder aufgeführt sind, die durch die Eigenschaften `selection_mode` und `top_n` bestimmt wurden. Weitere Informationen finden Sie im Thema „featureselectionnode, Eigenschaften” auf Seite 261.

Superknotenscripts

Mithilfe der Scriptsprache von IBM SPSS Modeler können Sie Scripts innerhalb jedes beliebigen Endsuperknotens erstellen und speichern. Diese Scripts stehen ausschließlich für Endsuperknoten zur Verfügung und werden häufig beim Erstellen von Vorlagenstreams oder zum Erzwingen einer bestimmten Ausführungsreihenfolge für die Superknoteninhalte verwendet. Mit Superknotenscripts können Sie außerdem mehrere Scripts in einem Stream ausführen.

Beispiel: Angenommen, Sie müssen die Ausführungsreihenfolge für einen komplexen Stream angeben und Ihr Superknoten enthält mehrere Knoten, darunter einen Globalwerteknoten, der ausgeführt werden muss, bevor ein neues Feld, das in einem Plotknoten verwendet wird, abgeleitet wird. In diesem Fall können Sie ein Superknotenscript erstellen, das zuerst den Globalwerteknoten ausführt. Die durch diesen Knoten berechneten Werte, wie Durchschnitt oder Standardabweichung, können anschließend bei der Ausführung des Plotknotens verwendet werden.

Innerhalb eines Superknotenscripts können Sie Knoteneigenschaften auf dieselbe Weise angeben wie bei anderen Scripts. Alternativ können Sie die Eigenschaften für einen beliebigen Superknoten oder den

darin gekapselten Knoten direkt über ein Stream-Script ändern und definieren. Weitere Informationen finden Sie im Thema Kapitel 21, „Superknoteneigenschaften”, auf Seite 469. Diese Methode funktioniert für Quellen- und Prozesssuperknoten ebenso wie für Endsuperknoten.

Anmerkung: Da nur Endsuperknoten ihre eigenen Scripts ausführen können, steht die Registerkarte "Scripts" des Dialogfelds "Superknoten" nur für Endsuperknoten zur Verfügung.

So öffnen Sie das Dialogfeld "Superknotenscript" im Haupterstellungsbereich:

Wählen Sie einen Endsuperknoten im Streamerstellungsbereich aus und wählen Sie im Menü "Superknoten" Folgendes aus:

Superknotenscript...

So öffnen Sie das Dialogfeld "Superknotenscript" im vergrößerten Superknoten-Erstellungsbereich:

Klicken Sie mit der rechten Maustaste auf den Superknoten-Erstellungsbereich und wählen Sie im Kontextmenü Folgendes aus:

Superknotenscript...

Beispiel für Superknotenscript

Das folgende Superknotenscript gibt die Reihenfolge an, in der die Endknoten innerhalb des Superknotens ausgeführt werden sollen. Mit dieser Reihenfolge wird sichergestellt, dass der Globalwerteknoten zuerst ausgeführt wird und somit die von diesem Knoten berechneten Werte bei der Ausführung eines weiteren Knotens verwendet werden können.

```
execute 'Set Globals'
execute 'gains'
execute 'profit'
execute 'age v. $CC-pep'
execute 'Table'
```

Sperren und Entsperren von Superknoten

Das folgende Beispiel stellt dar, wie Sie einen Superknoten sperren und entsperren:

```
stream = modeler.script.stream()
superNode=stream.findByID('id854RNTSD5MB')
# unlock one super node
print 'unlock the super node with password abcd'
if superNode.unlock('abcd'):
    print 'unlocked.'
else:
    print 'invalid password.'
# lock one super node
print 'lock the super node with password abcd'
superNode.lock('abcd')
```

Verwendung von Schleifen und bedingte Ausführung in Streams

Ab Version 16.0 ermöglicht SPSS Modeler es Ihnen, einige grundlegende Scripts in einem Stream zu erstellen, indem Sie Werte in verschiedenen Dialogfeldern auswählen, statt die Anweisungen direkt in der Scriptsprache schreiben zu müssen. Die beiden Haupttypen von Scripts, die Sie auf diese Weise erstellen können, sind einfache Schleifen sowie eine Möglichkeit zum Ausführen von Knoten, wenn eine Bedingung erfüllt ist.

Sie können Regeln zur Verwendung von Schleifen und für die bedingte Ausführung in einem Stream kombinieren. Sie haben z. B. Daten über den Kfz-Verkauf von Herstellern weltweit Sie könnten eine Schleife definieren, um die Daten in einem Stream zu verarbeiten, Details nach dem Herstellerland

ermitteln und die Daten in unterschiedlichen Diagrammen ausgeben mit Details wie Umsatzvolumen nach Modell, Emissionsstufen nach Hersteller und Motorgröße usw. Wenn Sie nur Daten europäischer Hersteller analysieren wollen, könnten Sie auch Bedingungen zur Schleife hinzufügen, die verhindern, dass Diagramme für amerikanische und asiatische Hersteller erstellt werden.

Anmerkung: Da sowohl die Verwendung von Schleifen als auch die bedingte Ausführung auf Hintergrundscripts basieren, werden sie nur auf einen gesamten Stream bei dessen Ausführung angewendet.

- **Verwendung von Schleifen**. Sie können mit Schleifen sich wiederholende Tasks automatisieren. Dabei könnte z. B. eine bestimmte Anzahl von Knoten zu einem Stream hinzugefügt und jeweils ein Knotenparameter einzeln geändert werden. Alternativ könnten Sie die wiederholte Ausführung eines Streams oder einer Verzweigung für eine bestimmte Anzahl von Malen steuern (siehe folgende Beispiele):
  - Führen Sie den Stream eine bestimmte Anzahl Male aus und ändern Sie die Quelle bei jeder Ausführung.
  - Führen Sie den Stream eine bestimmte Anzahl Male aus und ändern Sie bei jeder Ausführung den Wert einer Variablen.
  - Führen Sie den Stream eine bestimmte Anzahl Male aus und geben Sie bei jeder Ausführung ein Extrafeld ein.
  - Erstellen Sie ein Modell eine bestimmte Anzahl Male und ändern Sie bei jeder Erstellung eine Modelleinstellung.
- **Bedingte Ausführung**. Damit können Sie steuern, wie Endknoten in Abhängigkeit von vordefinierten Bedingungen ausgeführt werden (siehe folgende Beispiele):
  - Steuern Sie, ob ein Knoten ausgeführt wird, in Abhängigkeit davon, ob ein bestimmter Wert "true" oder "false" ist.
  - Definieren Sie, ob Schleifen parallel oder sequenziell ausgeführt werden.

Sowohl die Verwendung von Schleifen als auch die bedingte Ausführung werden auf der Registerkarte Ausführung im Dialogfeld Streameigenschaften definiert. Alle Knoten, die in bedingten oder Schleifenanforderungen verwendet werden, werden mit einem zusätzlichen Symbol im Streamerstellungsbereich angezeigt, um darauf hinzuweisen, dass sie Teil einer Schleifen- oder einer bedingten Ausführung sind.

Sie können auf drei Arten auf die Registerkarte Ausführung zugreifen:

- Über die Menüs oben im Hauptdialogfeld:
  1. Wählen Sie im Menü "Extras" Folgendes aus:

     Streameigenschaften > **Ausführung**
  2. Klicken Sie auf die Registerkarte <uicontrol>Ausführung</uicontrol>, um mit den Scripts des aktuellen Streams zu arbeiten.
- Aus einem Stream:
  1. Klicken Sie mit der rechten Maustaste auf einen Knoten und wählen Sie **Verwendung von Schleifen/bedingte Ausführung** aus.
  2. Wählen Sie die entsprechende Option aus dem Untermenü aus.
- Klicken Sie in der Grafiksymbolleiste oben im Hauptdialogfeld auf das Symbol für die Streameigenschaften.

Wenn Sie zum ersten Mal Details zur Schleifenausführung oder zur bedingten Ausführung eingerichtet haben, wählen Sie auf der Registerkarte "Ausführung" den Ausführungsmodus **Schleifen/bedingte Ausführung** und anschließend die Unterregisterkarte **Bedingt** oder **Schleife** aus.

Verwendung von Schleifen in Streams

Mit Schleifen können Sie sich wiederholende Tasks in Streams automatisieren (siehe folgende Beispiele):

- Führen Sie den Stream eine bestimmte Anzahl Male aus und ändern Sie die Quelle bei jeder Ausführung.

- Führen Sie den Stream eine bestimmte Anzahl Male aus und ändern Sie bei jeder Ausführung den Wert einer Variablen.
- Führen Sie den Stream eine bestimmte Anzahl Male aus und geben Sie bei jeder Ausführung ein Extrafeld ein.
- Erstellen Sie ein Modell eine bestimmte Anzahl Male und ändern Sie bei jeder Erstellung eine Modelleinstellung.

Sie definieren die zu erfüllenden Bedingungen auf der Unterregisterkarte **Verwendung von Schleifen** der Registerkarte "Streamausführung". Um die Unterregisterkarte anzuzeigen, wählen Sie den Ausführungsmodus **Verwendung von Schleifen/Bedingte Ausführung** aus.

Alle definierten Anforderungen für die Verwendung von Schleifen werden bei der Ausführung des Streams wirksam, wenn der Ausführungsmodus **Verwendung von Schleifen/Bedingte Ausführung** aktiviert wurde. Optional können Sie den Script-Code für Ihre Schleifenanforderungen generieren und in den Scripteditor einfügen, indem Sie unten rechts auf der Unterregisterkarte "Verwendung von Schleifen" auf **Einfügen ...** klicken. Die Anzeige der Hauptregisterkarte "Ausführung" ändert sich und der Ausführungsmodus **Standard (optionales Script)** wird mit dem Script im oberen Teil der Registerkarte angezeigt. Dies bedeutet, dass Sie eine Schleifenstruktur mithilfe der verschiedenen Optionen im Dialogfeld Verwendung von Schleifen definieren können, bevor Sie ein Script generieren, das Sie im Scripteditor weiter anpassen können. Wenn Sie auf **Einfügen ...** klicken, werden alle Bedingungen für die bedingte Ausführung, die von Ihnen definiert wurden, auch in dem generierten Script angezeigt.

Wichtig: Die Variablen für die Verwendung von Schleifen, die Sie in einem SPSS Modeler-Stream festlegen, können überschrieben werden, wenn Sie den Stream in einem Job von IBM SPSS Collaboration and Deployment Services ausführen. Der Grund hierfür ist, dass der Jobeditoreintrag von IBM SPSS Collaboration and Deployment Services den SPSS Modeler-Eintrag überschreibt. Wenn Sie z. B. eine Variable für die Verwendung von Schleifen im Stream so festlegen, dass für jede Schleife ein anderer Ausgabedateiname erstellt wird, werden die Dateien in SPSS Modeler ordnungsgemäß benannt, jedoch von dem festen Eintrag überschrieben, der auf der Registerkarte "Ergebnis" von IBM SPSS Collaboration and Deployment Services Deployment Manager eingegeben wird.

So richten Sie eine Schleife ein:

1. Erstellen Sie einen Iterationsschlüssel, um die Hauptschleifenstruktur zu definieren, die in einem Stream ausgeführt werden soll. Weitere Informationen finden Sie in Erstellen eines Iterationsschlüssels zur Verwendung von Schleifen in Streams.
2. Definieren Sie bei Bedarf eine oder mehrere Iterationsvariablen. Weitere Informationen finden Sie in Erstellen einer Iterationsvariablen zur Verwendung von Schleifen in Stream.
3. Die Iterationen und die Variablen, die Sie erstellt haben, werden im Hauptteil der Unterregisterkarte angezeigt. Standardmäßig werden Iterationen in der aufgeführten Reihenfolge ausgeführt. Um eine Iteration in der Liste nach oben oder unten zu verschieben, wählen Sie sie durch Anklicken aus und ändern Sie die Reihenfolge mit dem Auf- oder Abwärtspfeil in der rechten Spalte der Unterregisterkarte.

Erstellen eines Iterationsschlüssels zur Verwendung von Schleifen in Streams

Sie verwenden einen Iterationsschlüssel, um die Hauptschleifenstuktur zu definieren, die in einem Stream ausgeführt werden soll. Bei der Analyse von Kfz-Verkäufen könnten Sie z. B. einen Streamparameter namens *Herstellerland* erstellen und diesen als Iterationsschlüssel verwenden. Beim Ausführen des Streams wird dieser Schlüssel während jeder Iteration auf die einzelnen Länderwerte in Ihren Daten gesetzt. Verwenden Sie das Dialogfeld Iterationsschlüssel definieren, um den Schlüssel zu definieren.

Wählen Sie zum Öffnen des Dialogfensters entweder die Schaltfläche **Iterationsschlüssel...** in der linken unteren Ecke der Unterregisterkarte "Schleifen" oder klicken Sie mit der rechten Maustaste auf einen beliebigen Knoten im Stream und wählen Sie **Schleifen/bedingte Ausführung** > **Iterationsschlüssel definieren (Felder)** oder **Schleifen/bedingte Ausführung** > **Iterationsschlüssel definieren (Werte)** aus.

Wenn Sie das Dialogfeld vom Stream aus öffnen, sind einige der Felder wie der Knotenname möglicherweise bereits ausgefüllt.

Füllen Sie die folgenden Felder aus, um einen Iterationsschlüssel zu definieren:

Iteration nach. Sie können eine der folgenden Optionen auswählen:

- **Streamparameter - Felder**. Verwenden Sie diese Option, um eine Schleife zu erstellen, die den Wert eines vorhandenen Streamparameters nacheinander auf jedes angegebene Feld setzt.
- **Streamparameter - Werte**. Verwenden Sie diese Option, um eine Schleife zu erstellen, die den Wert eines vorhandenen Streamparameters nacheinander auf jeden angegebenen Wert setzt.
- **Knoteneigenschaft - Felder**. Verwenden Sie diese Option, um eine Schleife zu erstellen, die den Wert einer Knoteneigenschaft nacheinander auf jedes angegebene Feld setzt.
- **Knoteneigenschaft - Werte**. Verwenden Sie diese Option, um eine Schleife zu erstellen, die den Wert einer Knoteneigenschaft nacheinander auf jeden angegebenen Wert setzt.

Festzulegen. Wählen Sie das Element aus, dessen Wert bei jeder Schleifenausführung festgelegt werden soll. Sie können eine der folgenden Optionen auswählen:

- **Parameter**. Nur verfügbar, wenn Sie **Streamparameter - Felder** oder **Streamparameter - Werte** auswählen. Wählen Sie den erforderlichen Parameter aus der Liste der verfügbaren Parameter aus.
- **Knoten**. Nur verfügbar, wenn Sie **Knoteneigenschaft - Felder** oder **Knoteneigenschaft - Werte** auswählen. Wählen Sie den Knoten aus, für die Sie eine Schleife definieren wollen. Klicken Sie auf die Suchschaltfläche, um das Dialogfeld für die Knotenauswahl zu öffnen, und wählen Sie den gewünschten Knoten aus. Werden zu viele Knoten angezeigt, können Sie die Anzeige so filtern, dass nur bestimmte Knotentypen angezeigt werden. Wählen Sie dazu eine der folgenden Kategorien aus: Quellen-, Prozess-, Diagramm-, Modellierungs, Ausgabe-, Export- oder Modellanwendungsknoten.
- **Eigenschaft**. Nur verfügbar, wenn Sie **Knoteneigenschaft - Felder** oder **Knoteneigenschaft - Werte** auswählen. Wählen Sie die Eigenschaft des Knotens aus der Liste aus.

Zu verwendende Felder. Nur verfügbar, wenn Sie **Streamparameter - Felder** oder **Knoteneigenschaft - Felder** auswählen. Wählen Sie die Felder in einem Knoten aus, um die Iterationswerte anzugeben. Sie können eine der folgenden Optionen auswählen:

- **Knoten**. Nur verfügbar, wenn Sie **Streamparameter - Felder** auswählen. Wählen Sie den Knoten mit den Details aus, für die Sie eine Schleife konfigurieren möchten. Klicken Sie auf die Suchschaltfläche, um das Dialogfeld für die Knotenauswahl zu öffnen, und wählen Sie den gewünschten Knoten aus. Werden zu viele Knoten angezeigt, können Sie die Anzeige so filtern, dass nur bestimmte Knotentypen angezeigt werden. Wählen Sie dazu eine der folgenden Kategorien aus: Quellen-, Prozess-, Diagramm-, Modellierungs, Ausgabe-, Export- oder Modellanwendungsknoten.
- **Feldliste**. Klicken Sie auf die Listenschaltfläche in der rechten Spalte, um das Dialogfeld Felder auswählen anzuzeigen, in dem Sie die Felder im Knoten zur Angabe der Iterationsdaten auswählen können. Weitere Informationen finden Sie im Thema „Auswählen von Feldern für Iterationen” auf Seite 10.

Zu verwendende Werte. Nur verfügbar, wenn Sie **Streamparameter - Werte** oder **Knoteneigenschaft - Werte** auswählen. Wählen Sie die Werte im ausgewählten Feld aus, die als Iterationswerte verwendet werden sollen. Sie können eine der folgenden Optionen auswählen:

- **Knoten**. Nur verfügbar, wenn Sie **Streamparameter - Werte** auswählen. Wählen Sie den Knoten mit den Details aus, für die Sie eine Schleife konfigurieren möchten. Klicken Sie auf die Suchschaltfläche, um das Dialogfeld für die Knotenauswahl zu öffnen, und wählen Sie den gewünschten Knoten aus. Werden zu viele Knoten angezeigt, können Sie die Anzeige so filtern, dass nur bestimmte Knotentypen angezeigt werden. Wählen Sie dazu eine der folgenden Kategorien aus: Quellen-, Prozess-, Diagramm-, Modellierungs, Ausgabe-, Export- oder Modellanwendungsknoten.
- **Feldliste**. Wählen Sie das Feld im Knoten aus, das die Iterationsdaten angeben soll.
- **Werteliste**. Klicken Sie auf die Listenschaltfläche in der rechten Spalte, um das Dialogfeld Werte auswählen anzuzeigen, in dem Sie die Werte im Feld zur Angabe der Iterationsdaten auswählen können.

Erstellen einer Iterationsvariablen zur Verwendung von Schleifen in Streams

Sie können Iterationsvariablen verwenden, um die Werte von Streamparametern oder Eigenschaften ausgewählter Knoten in einem Stream bei jeder Schleifenausführung zu ändern. Wenn Ihre Streamschleife z. B. Kfz-Verkaufsdaten analysiert und *Herstellerland* als Iterationsschlüssel verwendet, können Sie eine Diagrammausgabe erhalten, die den Umsatz nach Modell anzeigt, und eine andere Diagrammausgabegabe, die Abgasemissionen anzeigt. In diesen Fällen könnten Sie Iterationsvariablen erstellen, die neue Titel für die entsprechenden Diagramme erstellen, z. B. *Schwedische Fahrzeugemissionen* und *Japanische Kfz-Verkäufe nach Modell*. Verwenden Sie das Dialogfeld Iterationsvariable definieren, um die erforderlichen Variablen zu definieren.

Wählen Sie zum Öffnen des Dialogfeldes die Option **Variable hinzufügen...** aus, die sich in der linken unteren Ecke der untergeordneten Registerkarte "Schleifen" befindet. Klicken Sie alternativ mit der rechten Maustaste auf einen beliebigen Knoten im Stream und wählen Sie die Optionen **Schleifen/Bedingte Ausführung** > **Iterationsvariable definieren** aus.

Füllen Sie die folgenden Felder aus, um eine Iterationsvariable zu definieren:

Ändern. Wählen Sie den Typ von Attribut aus, den Sie ändern wollen. Sie können zwischen **Streamparameter** oder **Knoteneigenschaft** wählen.

- Wenn Sie **Streamparameter** auswählen, wählen Sie den erforderlichen Parameter aus und definieren Sie dann mit einer der folgenden Optionen (sofern für Ihren Stream verfügbar), auf welchen Wert dieser Parameter bei jeder Schleifeniteration gesetzt werden soll:
  - **Globale Variable**. Wählen Sie die globale Variable aus, auf die der Streamparameter gesetzt werden soll.
  - **Tabellenausgabezelle**. Um einen Streamparameter als Wert in einer Tabellenausgabezelle festzulegen, wählen Sie die Tabelle in der Liste aus und geben Sie die **Zeile** und **Spalte** ein, die verwendet werden sollen.
  - **Manuell eingeben**. Wählen Sie diese Option aus, wenn Sie manuell einen Wert eingeben wollen, den dieser Parameter in den einzelnen Iterationen annehmen soll. Wenn Sie zur Unterregisterkarte Verwendung von Schleifen zurückkehren, wird eine neue Spalte erstellt, in die Sie den erforderlichen Text eingeben können.
- Wenn Sie **Knoteneigenschaft** auswählen, wählen Sie den erforderlichen Knoten und eine seiner Eigenschaften aus und legen Sie dann den Wert fest, der für diese Eigenschaft verwendet werden soll. Definieren Sie den neuen Eigenschaftswert mit einer der folgenden Optionen:
  - **Allein**. Der Eigenschaftswert verwendet den Iterationsschlüsselwert. Weitere Informationen finden Sie im Thema „Erstellen eines Iterationsschlüssels zur Verwendung von Schleifen in Streams” auf Seite 8.
  - **Als Präfix für Stamm**. Verwendet den Iterationsschlüsselwert als Präfix für den Wert, den Sie in das Feld **Stamm** eingeben.
  - **Als Suffix für Stamm**. Verwendet den Iterationsschlüsselwert als Suffix für den Wert, den Sie in das Feld **Stamm** eingeben.

  Wenn Sie die Präfix- oder die Suffix-Option ausgewählt haben, werden Sie aufgefordert, den zusätzlichen Text in das Feld **Stamm** einzugeben. Wenn Ihr Iterationsschlüssel z. B. *Herstellerland* lautet und Sie die Option **Als Präfix für Stamm** auswählen, könnten Sie in dieses Feld - *Verkauf nach Modell* eingeben.

Auswählen von Feldern für Iterationen

Beim Erstellen von Iterationen können Sie über das Dialogfeld Felder auswählen ein oder mehrere Felder auswählen.

Sortieren nach: Sie können verfügbare Felder für die Anzeige sortieren, indem Sie eine der folgenden Optionen auswählen:

- **Natürlich**: Zeigt die Felder in der Reihenfolge an, in der im aktuellen Datenstream an den aktuellen Knoten übergeben wurden.

- **Name**: Verwendet eine alphabetische Reihenfolge zum Sortieren der Felder für die Anzeige.
- **Typ**: Zeigt Felder sortiert nach ihrem Messniveau an. Diese Option ist hilfreich bei der Auswahl von Feldern mit einem bestimmten Messniveau.

Wählen Sie Felder einzeln aus der Liste aus oder wählen Sie bei gedrückter Umschalttaste oder bei gedrückter Steuertaste mehrere Felder aus. Sie können auch die Schaltflächen unter der Liste verwenden, um Gruppen von Feldern basierend auf deren Messniveau auszuwählen oder um alle Felder in der Tabelle aus- oder abzuwählen.

Die zur Auswahl verfügbaren Felder werden gefiltert, sodass nur die Felder angezeigt werden, die für den verwendeten Streamparameter oder die verwendete Knoteneigenschaft geeignet sind. Wenn Sie z. B. einen Streamparameter mit dem Speichertyp "Zeichenfolge" verwenden, werden nur Felder mit diesem Speichertyp angezeigt.

Bedingte Ausführung in Streams

Mit bedingter Ausführung können Sie steuern, wie Endknoten ausgeführt werden, und zwar in Abhängigkeit davon, ob die Streaminhalte den definierten Bedingungen entsprechen. Beispiele:

- Steuern Sie, ob ein Knoten ausgeführt wird, in Abhängigkeit davon, ob ein bestimmter Wert "true" oder "false" ist.
- Definieren Sie, ob Schleifen parallel oder sequenziell ausgeführt werden.

Sie definieren die zu erfüllenden Bedingungen auf der Unterregisterkarte **Bedingt** der Registerkarte "Streamausführung". Um die Unterregisterkarte anzuzeigen, wählen Sie den Ausführungsmodus **Verwendung von Schleifen/Bedingte Ausführung** aus.

Alle definierten Anforderungen für die bedingte Ausführung werden bei der Ausführung des Streams wirksam, wenn der Ausführungsmodus **Verwendung von Schleifen/Bedingte Ausführung** aktiviert wurde. Optional können Sie den Script-Code für Ihre bedingten Ausführungsanforderungen generieren und in den Scripteditor einfügen, indem Sie auf **Einfügen...** klicken. In der rechten unteren Ecke der Unterregisterkarte "Bedingt" wird die Anzeige der Hauptregisterkarte "Ausführung" so geändert, dass der Ausführungsmodus **Standard (optionales Script)** mit dem Script im oberen Teil der Registerkarte angezeigt wird. Dies bedeutet, dass Sie Bedingungen mithilfe der verschiedenen Optionen im Dialogfeld Verwendung von Schleifen definieren können, bevor Sie ein Script generieren, das Sie im Scripteditor weiter anpassen können. Beachten Sie beim Klicken auf **Einfügen...**, dass alle Schleifenanforderungen, die Sie definiert haben, auch im generierten Script angezeigt werden.

So definieren Sie eine Bedingung:

1. Klicken Sie in der rechten Spalte der Unterregisterkarte "Bedingt" auf die Schaltfläche "Neue Bedingung hinzufügen" , um das Dialogfeld **Bedingte Ausführungsanweisung hinzufügen** zu öffnen. In diesem Dialogfeld geben Sie die Bedingung an, die erfüllt sein muss, damit der Knoten ausgeführt wird.
2. Geben Sie im Dialogfenster **Bedingte Ausführungsanweisung hinzufügen** Folgendes an:
   a. **Knoten**. Wählen Sie den Knoten aus, für die Sie eine bedingte Ausführung konfigurieren wollen. Klicken Sie auf die Suchschaltfläche, um das Dialogfeld für die Knotenauswahl zu öffnen, und wählen Sie den gewünschten Knoten aus. Werden zu viele Knoten angezeigt, können Sie die Anzeige so filtern, dass nur Knoten einer der folgenden Kategorien angezeigt werden: Export-, Diagramm-, Modellierungs- oder Ausgabeknoten.
   b. **Bedingung basierend auf**. Geben Sie die Bedingung an, die erfüllt sein muss, damit der Knoten ausgeführt wird. Sie können unter vier Optionen wählen: **Streamparameter**, **Globale Variable**, **Tabellenausgabezelle** oder **Immer wahr**. Welche Details Sie in der unteren Hälfte des Dialogfelds angeben können, wird von der ausgewählten Bedingung gesteuert.
      - **Streamparameter**. Wählen Sie den Parameter aus der verfügbaren Liste aus und wählen Sie dann den **Operator** für diesen Parameter aus. Der Operator kann beispielsweise 'Größer als', 'Gleich', 'Kleiner als', 'Zwischen' usw. sein. Anschließend geben Sie je nach Operator den **Wert** oder den Mindest- und Maximalwert ein.

- **Globale Variable**. Wählen Sie die Variable aus der Liste der verfügbaren Variablen aus. Beispielsweise kann die Variable Mittelwert, Summe, Minimum, Maximumoder Standardabweichung sein. Anschließend wählen Sie **Operator** und die erforderlichen Werte aus.
- **Tabellenausgabezelle**. Wählen Sie den Tabellenknoten aus der verfügbaren Liste aus und wählen Sie anschließend **Zeile** und **Spalte** in der Tabelle aus. Anschließend wählen Sie **Operator** und die erforderlichen Werte aus.
- **Immer wahr**. Wählen Sie diese Option aus, wenn der Knoten immer ausgeführt werden muss. Wenn Sie diese Option auswählen, müssen keine weiteren Parameter mehr ausgewählt werden.

3. Wiederholen Sie Schritt 1 und 2 so oft, bis Sie alle erforderlichen Bedingungen definiert haben. Der ausgewählte Knoten und die Bedingung, die erfüllt werden muss, damit der Knoten ausgeführt wird, werden im Hauptteil der Unterregisterkarte in den Spalten **Ausführungsknoten** bzw. **Wenn diese Bedingung wahr ist** angezeigt.
4. Standardmäßig werden Knoten und Bedingungen in der aufgeführten Reihenfolge ausgeführt. Um einen Knoten und eine Bedingung in der Liste nach oben oder unten zu verschieben, wählen Sie diese durch Anklicken aus und ändern Sie die Reihenfolge mit dem Auf- oder Abwärtspfeil in der rechten Spalte der Unterregisterkarte.

Darüber hinaus können Sie die folgenden Optionen unten in der Unterregisterkarte Bedingt festlegen:

- **Alle der Reihenfolge nach auswerten**. Wählen Sie diese Option aus, um jede Bedingung in der auf der Unterregisterkarte aufgeführten Reihenfolge auszuwerten. Die Knoten, für die die Bedingungen erfüllt sind, werden ausgeführt, sobald alle Bedingungen ausgewertet wurden.
- **Jeweils nur einen ausführen**. Nur verfügbar, wenn **Alle der Reihenfolge nach auswerten** ausgewählt wurde. Wenn Sie diese Option auswählen und eine Bedingung als "True" ausgewertet wird, wird der Knoten ausgeführt, dem diese Bedingung zugeordnet ist, bevor die nächste Bedingung ausgewertet wird.
- **Auswerten bis zum ersten Treffer**. Wenn Sie diese Option auswählen, wird nur der erste Knoten ausgeführt, für den die angegebenen Bedingungen als "True" ausgewertet werden.

Ausführen und Unterbrechen von Scripts

Es gibt mehrere Möglichkeiten zur Ausführung von Scripts. Beispielsweise führt die Schaltfläche "Dieses Script ausführen" im Dialogfeld für das Stream-Script oder Standalone-Script das vollständige Script aus:

Abbildung 1. Schaltfläche "Dieses Script ausführen"

Die Schaltfläche "Ausgewählte Zeilen ausführen" führt eine einzelne Zeile oder einen Block benachbarter Zeilen aus, die Sie im Script ausgewählt haben:

Abbildung 2. Schaltfläche "Ausgewählte Zeilen ausführen"

Zum Ausführen von Scripts stehen folgende Methoden zur Auswahl:

- Klicken Sie im Dialogfeld für ein Stream-Script oder ein Standalone-Script auf die Schaltfläche "Dieses Script ausführen" oder "Ausgewählte Zeilen ausführen".
- Ausführen eines Streams mit **Dieses Script ausführen** als Standardausführungsmethode.
- Verwenden Sie das Flag `-execute` beim Start im interaktiven Modus. Weitere Informationen finden Sie im Thema „Verwenden von Befehlszeilenargumenten” auf Seite 69.

Anmerkung: Ein Superknotenscript wird bei der Ausführung des Superknotens ausgeführt, sofern Sie **Dieses Script ausführen** im Superknotenscript-Dialogfeld ausgewählt haben.

Unterbrechen der Scriptausführung

Während der Scriptausführung ist im Dialogfeld "Stream-Script" die rote Stoppschaltfläche aktiviert. Mit dieser Schaltfläche können Sie die Ausführung des Scripts und aller aktuellen Streams abbrechen.

Suchen und Ersetzen

Das Dialogfeld "Suchen/Ersetzen" ist in Situationen verfügbar, in denen Sie Script- oder Ausdruckstext bearbeiten, u. a. im Script-Editor, im CLEM Expression Builder und bei der Definition von Vorlagen im Berichtsknoten. Während der Bearbeitung von Text in einem dieser Bereiche können Sie durch Drücken von Strg+F das Dialogfeld aufrufen. Achten Sie dabei darauf, dass sich der Cursor über einem Textbereich befindet. So können Sie beispielsweise bei der Arbeit in einem Füllerknoten das Dialogfeld aus einem der Textbereiche auf der Registerkarte "Einstellungen" aufrufen oder über das Textfeld im Expression Builder.

1. Achten Sie darauf, dass sich der Cursor in einem Textfeld befindet, und drücken Sie Strg+F, um das Dialogfeld "Suchen/Ersetzen" aufzurufen.
2. Geben Sie den Text ein, nach dem Sie suchen möchten, oder treffen Sie eine Auswahl aus der Drop-down-Liste der kürzlich durchsuchten Elemente.
3. Geben Sie, falls erforderlich, den Ersatztext ein.
4. Klicken Sie auf **Weitersuchen**, um die Suche zu starten.
5. Klicken Sie auf **Ersetzen**, um die aktuelle Auswahl zu ersetzen, oder **Alle ersetzen**, um alle bzw. die ausgewählten Instanzen zu ersetzen.
6. Das Dialogfeld wird nach jedem Vorgang geschlossen. Drücken Sie in einem Textbereich die Taste F3, um den letzten Suchvorgang zu wiederholen, bzw. drücken Sie Strg+F, um erneut auf das Dialogfeld zuzugreifen.

Suchoptionen

Groß-/Kleinschreibung beachten. Gibt an, ob beim Suchvorgang die Groß- und Kleinschreibung berücksichtigt wird; beispielsweise, ob *myvar* als identisch mit *myVar* betrachtet wird. Ersetzungstext wird unabhängig von dieser Einstellung stets genau so eingefügt, wie er eingegeben wurde.

Nur ganze Wörter. Gibt an, ob beim Suchvorgang auch Wortteile gefunden werden. Wenn diese Option ausgewählt ist, werden bei einer Suche nach *Tag* Terme wie *Tagesordnung* oder *Tag-und-Nacht-Gleiche* nicht als Treffer ausgegeben.

Reguläre Ausdrücke. Gibt an, ob die Syntax für reguläre Ausdrücke verwendet werden soll (siehe nächsten Abschnitt). Bei Auswahl dieser Option wird die Option **Nur ganze Wörter** inaktiviert und ihr Wert ignoriert.

Nur ausgewählter Text. Legt den Suchumfang bei Verwendung der Option **Alle ersetzen** fest.

Syntax für reguläre Ausdrücke

Mithilfe von regulären Ausdrücken können Sie nach Sonderzeichen (z. B. Tabulatoren oder Zeilenumbrüche), Zeichenklassen bzw. -bereichen, wie *a* bis *d*, beliebige Ziffer oder Nichtziffer, und nach Begrenzungen, wie beispielsweise Zeilenanfang bzw. Zeilenende, suchen. Folgende Arten von Ausdrücken werden unterstützt:

Tabelle 1. Zeichenübereinstimmungen

Zeichen	Übereinstimmung/Bedeutung
x	Das Zeichen x
\\	Umgekehrter Schrägstrich
\0n	Das Zeichen mit dem Oktalwert 0n (0 <= n <= 7)
\0nn	Das Zeichen mit dem Oktalwert 0nn (0 <= n <= 7)
\0mnn	Das Zeichen mit dem Oktalwert 0mnn (0 <= m <= 3; 0 <= n <= 7)

Tabelle 1. Zeichenübereinstimmungen (Forts.)	
Zeichen	**Übereinstimmung/Bedeutung**
\xhh	Das Zeichen mit dem Hexadezimalwert 0xhh
\uhhhh	Das Zeichen mit dem Hexadezimalwert 0xhhhh
\t	Das Tabulatorzeichen ('\u0009')
\n	Das Zeilenvorschubzeichen ('\u000A')
\r	Das Rücklaufzeichen ('\u000D')
\f	Das Formularvorschubzeichen ('\u000C')
\a	Das Alarmzeichen ('\u0007')
\e	Das Escapezeichen ('\u001B')
\cx	Das Steuerzeichen, das x entspricht

Tabelle 2. Übereinstimmende Zeichenklassen	
Zeichenklassen	**Übereinstimmung/Bedeutung**
[abc]	a, b oder c (einfache Klasse)
[^abc]	Beliebiges Zeichen mit Ausnahme von a, b oder c (Differenzmenge)
[a-zA-Z]	a bis einschließlich z bzw. A bis einschließlich Z (Bereich)
[a-d[m-p]]	a bis d bzw. m bis P (Vereinigungsmenge) Alternative Angabemöglichkeit: [a-dm-p]
[a-z&&[def]]	a bis z und d, e bzw. f (Schnittmenge)
[a-z&&[^bc]]	a bis z mit Ausnahme von b und c (Differenzmenge) Alternative Angabemöglichkeit: [ad-z]
[a-z&&[^m-p]]	a bis z mit Ausnahme von m bis p (Differenzmenge) Alternative Angabemöglichkeit: [a-lq-z]

Tabelle 3. Vordefinierte Zeichenklassen	
Vordefinierte Zeichenklassen	**Übereinstimmung/Bedeutung**
.	Beliebiges Zeichen (evtl. Übereinstimmung mit Zeilenabschlusszeichen)
\d	Beliebige Ziffer: [0-9]
\D	Nichtziffer: [^0-9]
\s	Ein Leerzeichen: [\t\n\x0B\f\r]
\S	Ein Nichtleerzeichen: [^\s]
\w	Ein Zeichen: [a-zA-Z_0-9]
\W	Ein Nichtzeichen: [^\w]

Tabelle 4. Übereinstimmungen mit Begrenzungszeichen	
Zeichen(folgen) für Begrenzungszeichen	**Übereinstimmung/Bedeutung**
^	Zeilenanfang

Tabelle 4. Übereinstimmungen mit Begrenzungszeichen (Forts.)

Zeichen(folgen) für Begrenzungszeichen	**Übereinstimmung/Bedeutung**
$	Zeilenende
\b	Wortgrenze
\B	Nichtwortgrenze
\A	Beginn der Eingabe
\Z	Ende der Eingabe mit Ausnahme des letzten Abschlusszeichens, sofern vorhanden
\z	Ende der Eingabe

Kapitel 2. Scriptsprache

Scriptsprache - Übersicht

Mit den Scriptfunktionen für IBM SPSS Modeler können Sie Scripts erstellen, die mit der SPSS Modeler-Benutzerschnittstelle arbeiten, Ausgabeobjekte manipulieren und Befehlssyntax ausführen. Sie können Scripts direkt aus SPSS Modeler ausführen.

Scripts in IBM SPSS Modeler werden in der Scriptsprache Python geschrieben. Die von IBM SPSS Modeler verwendete Java-basierte Implementierung von Python wird als Jython bezeichnet. Die Scriptsprache besteht aus folgenden Elementen:

- Format für die Referenzierung von Knoten, Streams, Projekten, Ausgaben und anderen IBM SPSS Modeler-Objekten
- Set mit Scriptanweisungen bzw. Befehlen, die zur Bearbeitung dieser Objekte verwendet werden können
- Script-Ausdruckssprache für die Festlegung der Werte von Variablen, Parametern und anderen Objekten
- Unterstützung für Kommentare, Fortsetzungen und Blöcke mit Literaltext

In den folgenden Abschnitten werden die Python-Scriptsprache, die Jython-Implementierung von Python und die grundlegende Syntax für das erste Arbeiten mit Scripting in IBM SPSS Modeler beschrieben. Informationen zu speziellen Eigenschaften und Befehlen finden Sie in den nachfolgenden Abschnitten.

Python und Jython

Jython ist eine Implementierung der Python-Scriptsprache, die in Java geschrieben ist und in die Java-Plattform integriert ist. Python ist eine leistungsfähige objektorientierte Scriptsprache. Jython ist nützlich, da es die Produktivitätsfunktionen einer ausgereiften Scriptsprache bereitstellt und anders als Python in einer beliebigen Umgebung ausgeführt werden kann, die Java Virtual Machine (JVM) unterstützt. Dies bedeutet, dass die Java-Bibliotheken in JVM beim Schreiben von Programmen zur Verfügung stehen. Mit Jython können Sie sich diesen Unterschied zunutze machen und die Syntax und die meisten Funktionen der Python-Sprache verwenden.

Als Scriptsprache ist Python (und deren Jython-Implementierung) einfach zu erlernen und effizient zu codieren und sie hat die minimal erforderliche Struktur zum Erstellen eines lauffähigen Programms. Code kann zeilenweise interaktiv eingegeben werden. Python ist eine interpretierte Scriptsprache; es gibt keinen Vorkompilierungsschritt wie in Java. Python-Programme sind einfache Textdateien, die bei ihrer Eingabe (nach der Analyse auf Syntaxfehler) interpretiert werden. Einfache Ausdrücke wie z. B. definierte Werte und auch komplexere Aktionen wie Funktionsdefinitionen werden unverzüglich ausgeführt und sind sofort einsatzbereit. Änderungen am Code können schnell getestet werden. Die Scriptinterpretation hat jedoch einige Nachteile. So ist beispielsweise die Verwendung einer nicht definierten Variablen kein Compilerfehler und wird deshalb erst bei der Ausführung der Anweisung erkannt, in der die Variable verwendet wird. In diesem Fall kann das Programm bearbeitet und ausgeführt werden, um den Fehler zu beheben.

Python betrachtet alles, d. h. alle Daten und den gesamten Code, als Objekt. Sie können diese Objekte daher mit Codezeilen manipulieren. Einige Typen wie Zahlen und Zeichenfolgen werden praktischerweise als Werte und nicht als Objekte betrachtet; dies wird von Python unterstützt. Es gibt einen unterstützten Nullwert. Dieser Nullwert hat den reservierten Namen None.

Eine umfassendere Einführung in Python- und Jython-Scripting sowie einige Beispielscripts finden Sie in http://www.ibm.com/developerworks/java/tutorials/j-jython1/j-jython1.html und http://www.ibm.com/developerworks/java/tutorials/j-jython2/j-jython2.html.

Python-Scripting

Dieses Handbuch zur Python-Scriptsprache bietet eine Einführung in die Komponenten, die am häufigsten beim Scripting in IBM SPSS Modeler verwendet werden, einschließlich der Konzepte und Programmiergrundlagen. Es liefert Ihnen ausreichend Kenntnisse, um mit der Entwicklung Ihrer eigenen Python-Scripts zu beginnen, die in IBM SPSS Modeler verwendet werden sollen.

Vorgänge

Die Zuordnung erfolgt mit einem Gleichheitszeichen (=). Um beispielsweise den Wert "3" einer Variablen mit dem Namen "x" zuzuordnen, würden Sie die folgende Anweisung verwenden:

```
x = 3
```

Das Gleichheitszeichen wird auch für die Zuordnung von Zeichenfolgedaten zu einer Variablen verwendet. Wenn Sie z. B. den Wert "ein Zeichenfolgewert" einer Variablen namens "y" zuweisen wollen, würden Sie die folgende Anweisung verwenden:

```
y = "a string value"
```

In der folgenden Tabelle werden einige der gängigen Vergleichs- und numerischen Operationen sowie deren Beschreibungen aufgelistet.

Tabelle 5. Allgemeine Vergleichsoperationen und numerische Operationen

Operation	**Beschreibung**
`x < y`	Ist x kleiner als y?
`x > y`	Ist x größer als y?
`x <= y`	Ist x kleiner-gleich y?
`x >= y`	Ist x größer-gleich y?
`x == y`	Ist x gleich y?
`x != y`	Ist x ungleich y?
`x <> y`	Ist x ungleich y?
`x + y`	y zu x addieren
`x - y`	y von x subtrahieren
`x * y`	x mit y multiplizieren
`x / y`	x durch y dividieren
`x ** y`	x hoch y

Listen

Listen sind Folgen von Elementen. Eine Liste kann eine beliebige Anzahl von Elementen enthalten und die Elemente der Liste können einen beliebigen Objekttyp haben. Listen kann man sich auch als Arrays vorstellen. Die Anzahl der Elemente in einer Liste kann sich durch Hinzufügen, Entfernen oder Ersetzen von Elementen verringern oder erhöhen.

Beispiele

`[]`	Eine beliebige Liste ohne Inhalt.
`[1]`	Eine Liste mit einem einzigen Element: einer ganzen Zahl.

`["Mike", 10, "Don", 20]`	Eine Liste mit vier Elementen: zwei Zeichenfolgeelemente und zwei ganzzahlige Elemente.
`[[],[7],[8,9]]`	Eine aus Listen bestehende Liste. Jede Unterliste ist entweder eine Liste ohne Inhalt oder eine Liste mit ganzzahligen Elementen.
`x = 7; y = 2; z = 3;` `[1, x, y, x + y]`	Eine Liste mit ganzen Zahlen. Dieses Beispiel veranschaulicht die Verwendung von Variablen und Ausdrücken.

Sie können eine Liste einer Variablen zuordnen. Beispiel:

```
mylist1 = ["one", "two", "three"]
```

Sie können dann auf bestimmte Elemente der Liste zugreifen. Beispiel:

```
mylist[0]
```

Dies resultiert in der folgenden Ausgabe:

```
one
```

Die Zahl in eckigen Klammern (`[]`) wird als *Index* bezeichnet und verweist auf ein bestimmtes Element der Liste. Die Elemente einer Liste werden mit 0 beginnend indexiert.

Sie können auch einen Bereich von Elementen einer Liste auswählen. Dies wird als *Slicing* bezeichnet. Beispielsweise wird mit `x[1:3]` das zweite und dritte Element von `x` ausgewählt. Der Index für das Bereichsende ist um 1 größer als der Index des letzten auszuwählenden Elements.

Zeichenfolgen

Eine *Zeichenfolge* ist eine unveränderliche Folge von Zeichen, die als Wert behandelt wird. Zeichenfolgen unterstützen alle Funktionen und Operationen mit unveränderlichen Folgen, die in einer neuen Zeichenfolge resultieren. Beispiel: `"abcdef"[1:4]` wird als `"bcd"` ausgegeben.

In Python werden Zeichen als Zeichenfolgen der Länge 1 dargestellt.

Zeichenfolgeliterale werden durch die Verwendung von ein- oder dreifachen Anführungszeichen definiert. Zeichenfolgen, die mit einfachen Anführungszeichen definiert sind, können nicht mehrere Zeilen umfassen, Zeichenfolgen in dreifachen Anführungszeichen dagegen schon. Eine Zeichenfolge kann in einfache Anführungszeichen (') oder doppelte Anführungszeichen (") eingeschlossen werden. Ein Anführungszeichen kann das andere Anführungszeichen ohne Escapezeichen oder das Anführungszeichen mit Escapezeichen einschließen, dem ein Backslash (\) vorangestellt ist.

Beispiele

```
"This is a string"
'This is also a string'
"It's a string"
'This book is called "Python Scripting and Automation Guide".'
"This is an escape quote (\") in a quoted string"
```

Mehrere durch Leerzeichen voneinander getrennte Zeichenfolgen werden vom Python-Parser automatisch verkettet. Dies vereinfacht die Eingabe langer Zeichenfolgen und das Mischen unterschiedlicher Anführungszeichen in einer einzigen Zeichenfolge. Beispiel:

```
"This string uses ' and " 'that string uses ".'
```

Dies resultiert in der folgenden Ausgabe:

```
This string uses ' and that string uses ".
```

Zeichenfolgen unterstützen mehrere nützliche Methoden. Einige dieser Methoden werden in der folgenden Tabelle genannt.

Tabelle 6. Zeichenfolgemethoden

Methode	**Verwendung**
`s.capitalize()`	Anfangsbuchstabe von `s` wird großgeschrieben.
`s.count(ss {,start {,end}})`	Zählt die Vorkommen von `ss` in `s[start:end]`
`s.startswith(str {, start {, end}})` `s.endswith(str {, start {, end}})`	Testet, ob `s` mit `str` beginnt. Testet, ob `s` mit `str` endet.
`s.expandtabs({size})`	Ersetzt Tabstopps durch Leerzeichen. Standardgröße (`size`) ist 8.
`s.find(str {, start {, end}})` `s.rfind(str {, start {, end}})`	Sucht den ersten Index von `str` in `s`; wird er nicht gefunden, lautet das Ergebnis `-1`. `rfind` sucht von rechts nach links.
`s.index(str {, start {, end}})` `s.rindex(str {, start {, end}})`	Sucht den ersten Index von `str` in `s`; wenn er nicht gefunden wird, wird `ValueError` ausgelöst. `rindex` sucht von rechts nach links.
`s.isalnum`	Testet, ob die Zeichenfolge alphanumerisch ist.
`s.isalpha`	Testet, ob die Zeichenfolge alphabetisch ist.
`s.isnum`	Testet, ob die Zeichenfolge numerisch ist.
`s.isupper`	Testet, ob die Zeichenfolge ganz in Großbuchstaben geschrieben ist.
`s.islower`	Testet, ob die Zeichenfolge ganz in Kleinbuchstaben geschrieben ist.
`s.isspace`	Testet, ob die Zeichenfolge nur aus Leerzeichen besteht.
`s.istitle`	Testet, ob die Zeichenfolge eine Folge von alphanumerischen Zeichen mit Großschreibung des ersten Buchstabens ist.

`s.lower()`
`s.upper()`
`s.swapcase()`
`s.title()`
	Konvertiert alles in Kleinbuchstaben.
	Konvertiert alles in Großbuchstaben.
	Invertiert die Groß-/Kleinschreibung.
	Konvertiert alles in Schreibung mit großem Anfangsbuchstaben.

`s.join(seq)`	Verknüpft die Zeichenfolgen in `seq` mit `s` als Trennzeichen.
`s.splitlines({keep})`	Teilt `s` in Zeilen auf. Wenn 'keep' `true` ist, werden die neuen Zeilen beibehalten.
`s.split({sep {, max}})`	Teilt `s` mithilfe von `sep` (Standardeinstellung von `sep` ist ein Leerzeichen) bis zu `max` Male in "Wörter" auf.

Tabelle 6. Zeichenfolgemethoden (Forts.)

Methode	Verwendung

`s.ljust(width)`
`s.rjust(width)`
`s.center(width)`
`s.zfill(width)`
	Richtet die Zeichenfolge in einem Feld der Breite `width` linksbündig aus.
	Richtet die Zeichenfolge in einem Feld der Breite `width` rechtsbündig aus.
	Zentriert die Zeichenfolge in einem Feld der Breite `width`.
	Füllt mit 0 auf.

`s.lstrip()`
`s.rstrip()`
`s.strip()`
	Entfernt führende Leerzeichen.
	Entfernt folgende Leerzeichen.
	Entfernt führende und folgende Leerzeichen.

`s.translate(str {,delc})`	Übersetzen Sie `s` mithilfe der Tabelle, nachdem Sie alle Zeichen in `delc` entfernt haben. `str` sollte eine Zeichenfolge mit der Länge == 256 sein.
`s.replace(old, new {, max})`	Ersetzt alle oder maximal `max` Vorkommen der Zeichenfolge `old` durch die Zeichenfolge `new`.

Anmerkungen

Anmerkungen sind Kommentare, die durch das Nummernzeichen (#) eingeführt werden. Der gesamte Text, der auf das Nummernzeichen in derselben Zeile folgt, wird als Teil der Anmerkung betrachtet und ignoriert. Eine Anmerkung kann in einer beliebigen Spalte beginnen. Das folgende Beispiel veranschaulicht die Verwendung von Anmerkungen:

```
#The HelloWorld application is one of the most simple
print 'Hello World' # print the Hello World line
```

Anweisungssyntax

Die Anweisungssyntax für Python ist sehr einfach. Im Allgemeinen ist jede Quellcodezeile eine einzelne Anweisung. Außer bei `expression`- und `assignment`-Anweisungen wird jede Anweisung durch ein Schlüsselwort wie `if` oder `for` eingeleitet. Leerzeilen oder Anmerkungszeilen können an beliebiger Stelle zwischen Anweisungen im Code eingefügt werden. Wenn mehrere Anweisungen in einer Zeile stehen, müssen sie durch ein Semikolon (`;`) voneinander getrennt werden.

Sehr lange Anweisungen können in weiteren Zeilen fortgesetzt werden. In diesem Fall muss die Anweisung, die in der nächsten Zeile fortgesetzt werden soll, mit einem umgekehrten Schrägstrich (\) enden. Beispiel:

```
x = "A looooooooooooooooooong string" + \
     "another looooooooooooooooooong string"
```

Wenn eine Struktur in runde Klammern (`()`), eckige Klammern (`[]`) oder geschweifte Klammern (`{}`) eingeschlossen ist, kann die Anweisung ohne umgekehrten Schrägstrich nach einem Komma in der nächsten Zeile fortgesetzt werden. Beispiel:

```
x = (1, 2, 3, "hello",
      "goodbye", 4, 5, 6)
```

IDs

IDs werden verwendet, um Variablen, Funktionen, Klassen und Schlüsselwörter zu bezeichnen. IDs können eine beliebige Länge haben. Sie müssen jedoch entweder mit einem alphabetischen Zeichen

in Groß- oder Kleinschreibung oder dem Unterstreichungszeichen (_) beginnen. Namen, die mit einem Unterstreichungszeichen beginnen, sind im Allgemeinen für interne oder private Namen reserviert. Nach dem ersten Zeichen kann die ID eine beliebige Anzahl von Buchstaben, der Ziffern 0-9 und Unterstrichen in beliebiger Kombination enthalten.

Es gibt in Jython einige reservierte Wörter, die nicht zur Benennung von Variablen, Funktionen oder Klassen verwendet werden können. Sie sind in die folgenden Kategorien aufgeteilt:

- **Einführungszeichen für Anweisungen:** `assert`, `break`, `class`, `continue`, `def`, `del`, `elif`, `else`, `except`, `exec`, `finally`, `for`, `from`, `global`, `if`, `import`, `pass`, `print`, `raise`, `return`, `try` und `while`
- **Einführungszeichen für Parameter:** `as`, `import` und `in`
- **Operatoren:** and, `in`, `is`, `lambda`, `not` und `or`

Eine falsche Verwendung von Schlüsselwörtern hat in der Regel einen `Syntaxfehler` zur Folge.

Codeblöcke

Codeblöcke sind Gruppen von Anweisungen, die an Stellen verwendet werden, an denen einzelne Anweisungen erwartet werden. Codeblöcke können auf jede der folgenden Anweisungen folgen: `if`, `elif`, `else`, `for`, `while`, `try`, `except`, `def` und `class`. Diese Anweisungen führen den Codeblock mit dem Doppelpunkt (`:`) ein. Beispiel:

```
if x == 1:
    y = 2
    z = 3
elif:
    y = 4
    z = 5
```

Zur Begrenzung von Codeblöcken wird Einrückung verwendet (anstelle der geschweiften Klammern, die in Java verwendet werden). Alle Zeilen in einem Block müssen an dieselbe Position eingerückt werden. Eine Änderung der Einrückung bedeutet das Ende eines Codeblocks. Gewöhnlich wird pro Ebene um vier Leerschritte eingerückt. Zur Einrückung der Zeilen empfiehlt es sich, Leerzeichen anstelle von Tabstopps zu verwenden. Leerzeichen und Tabstopps dürfen nicht gemischt werden. Die Zeilen im äußersten Block eines Moduls müssen in Spalte 1 beginnen, andernfalls wird ein Fehler des Typs `SyntaxError` ausgegeben.

Die Anweisungen, aus denen ein Codeblock besteht (und die auf einen Doppelpunkt folgen), können auch in einer einzigen Zeile durch Semikolons getrennt angeordnet werden. Beispiel:

```
if x == 1: y = 2; z = 3;
```

Übergeben von Argumenten an ein Script

Die Übergabe von Argumenten an ein Script ist nützlich, da dadurch ein Script wiederholt ohne Änderung verwendet werden kann. Die Argumente, die in der Befehlszeile übergeben werden, werden als Werte in der Liste `sys.argv` übergeben. Die Anzahl der übergebenen Werte kann über den Befehl `len(sys.argv)` abgerufen werden. Beispiel:

```
import sys
print "test1"
print sys.argv[0]
print sys.argv[1]
print len(sys.argv)
```

In diesem Beispiel importiert der Befehl `import` die gesamte Klasse `sys`, sodass die für diese Klasse vorhandenen Methoden wie `argv` verwendet werden können.

Das Script in diesem Beispiel kann über die folgende Befehlszeile aufgerufen werden:

```
/u/mjloos/test1 mike don
```

Daraus resultiert die folgende Ausgabe:

```
/u/mjloos/test1 mike don
test1
mike
don
3
```

Beispiele

Das Schlüsselwort `print` gibt die ihm direkt nachfolgenden Argumente aus. Wenn auf die Anweisung ein Komma folgt, enthält die Ausgabe keinen Zeilenumbruch. Beispiel:

```
print "This demonstrates the use of a",
print " comma at the end of a print statement."
```

Dies resultiert in der folgenden Ausgabe:

```
This demonstrates the use of a comma at the end of a print statement.
```

Die Anweisung `for` wird zum Durchlaufen eines Codeblocks verwendet. Beispiel:

```
mylist1 = ["one", "two", "three"]
for lv in mylist1:
    print lv
    continue
```

In diesem Beispiel werden drei Zeichenfolgen der Liste `mylist1` zugeordnet. Die Elemente der Liste werden dann ausgegeben, wobei jeweils ein Element in jeder Zeile steht. Dies resultiert in der folgenden Ausgabe:

```
one
two
three
```

In diesem Beispiel nimmt der Iterator `lv` nacheinander den Wert jedes Elements in der Liste `mylist1` an, während die For-Schleife den Codeblock für jedes Element implementiert. Ein Iterator kann eine gültige Kennung beliebiger Länge sein.

Die Anweisung `if` ist eine bedingte Anweisung. Sie wertet die Bedingung aus und gibt je nach Ergebnis der Bewertung entweder "true" oder "false" zurück. Beispiel:

```
mylist1 = ["one", "two", "three"]
for lv in mylist1:
    if lv == "two"
        print "The value of lv is ", lv
    else
        print "The value of lv is not two, but ", lv
    continue
```

In diesem Beispiel wird der Wert des Iterators `lv` ausgewertet. Wenn der Wert von`lv` `two`ist, wird eine andere Zeichenfolge an die Zeichenfolge zurückgegeben, die zurückgegeben wird, wenn der Wert von`lv` nicht `two`ist. Dies resultiert in der folgenden Ausgabe:

```
The value of lv is not two, but one
The value of lv is two
The value of lv is not two, but three
```

Mathematische Methoden

Aus dem Modul `math` können Sie auf nützliche mathematische Methoden zugreifen. Einige dieser Methoden werden in der folgenden Tabelle genannt. Sofern nichts anderes angegeben ist, werden alle Werte als Gleitkommazahlen zurückgegeben.

Tabelle 7. Mathematische Methoden

Methode	**Verwendung**
`math.ceil(x)`	Gibt die obere Grenze (ceiling) von x als Gleitkommazahl zurück. Dies ist die kleinste Ganzzahl, die größer-gleich x ist.
`math.copysign(x, y)`	Gibt x mit dem Vorzeichen von y zurück. `copysign(1, -0.0)` gibt Folgendes zurück: `-1`
`math.fabs(x)`	Gibt den absoluten Wert von x zurück.
`math.factorial(x)`	Gibt x-Fakultät zurück. Wenn x eine negative Zahl oder keine Ganzzahl ist, tritt ein `ValueError` auf.
`math.floor(x)`	Gibt die Untergrenze (floor) von x als Gleitkommazahl zurück. Dies ist die größte Ganzzahl, die kleiner-gleich x ist.
`math.frexp(x)`	Gibt die Mantisse (`m`) und den Exponenten (`e`) von x als Paar `(m, e)` zurück. `m` ist eine Gleitkommazahl und e ist eine Ganzzahl, sodass sich genau `x == m * 2**e` ergibt. Wenn x null ist, wird `(0.0, 0)` zurückgegeben; andernfalls wird `0.5 <= abs(m) < 1` zurückgegeben.
`math.fsum(iterable)`	Gibt eine genaue Gleitkommasumme von Werten in `iterable` zurück.
`math.isinf(x)`	Prüft, ob die Gleitkommazahl x positiv oder negativ unendlich ist.
`math.isnan(x)`	Prüft, ob die Gleitkommazahl x eine Nichtzahl (`NaN` -not a number) ist.
`math.ldexp(x, i)`	Gibt `x * (2**i)` zurück. Dies ist im Wesentlichen die Umkehrfunktion von `frexp`.
`math.modf(x)`	Gibt die ganzzahligen und die Bruchteile von x zurück. Beide Ergebnisse übernehmen das Vorzeichen von x und sind Gleitkommazahlen.
`math.trunc(x)`	Gibt den `Real` Wert x zurück, der auf eine `Integral` abgeschnitten wurde.
`math.exp(x)`	Gibt `e**x` zurück.
`math.log(x[, base])`	Gibt den Logarithmus von x zur Basis `base` zurück. Ohne Angabe von `base` wird der natürliche Logarithmus von x zurückgegeben.
`math.log1p(x)`	Gibt den natürlichen Logarithmus von `1+x (base e)` zurück.
`math.log10(x)`	Gibt den Logarithmus von x zur Basis 10 zurück.
`math.pow(x, y)`	Gibt x hoch y zurück. `pow(1.0, x)` und `pow(x, 0.0)` geben immer 1 zurück, selbst wenn x null oder eine Nichtzahl (NaN) ist.
`math.sqrt(x)`	Gibt die Quadratwurzel von x zurück.

Zusätzlich zu den mathematischen Funktionen gibt es einige nützliche trigonometrische Methoden. Diese Methoden werden in der folgenden Tabelle dargestellt.

Tabelle 8. Trigonometrische Methoden	
Methode	**Verwendung**
`math.acos(x)`	Gibt den Arkuskosinus von x in Radianten zurück.
`math.asin(x)`	Gibt den Arkussinus von x in Radianten zurück.
`math.atan(x)`	Gibt den Arkustangens von x in Radianten zurück.
`math.atan2(y, x)`	Gibt `atan(y / x)` in Radianten zurück.
`math.cos(x)`	Gibt den Kosinus von x in Radianten zurück.
`math.hypot(x, y)`	Gibt die euklidische Norm `sqrt(x*x + y*y)` zurück. Dies ist die Länge des Vektors vom Ursprung zum Punkt `(x, y)`.
`math.sin(x)`	Gibt den Sinus von x in Radianten zurück.
`math.tan(x)`	Gibt den Tangens von x in Radianten zurück.
`math.degrees(x)`	Konvertiert den Winkel x von Radianten in Grad.
`math.radians(x)`	Konvertiert den Winkel x von Grad in Radianten.
`math.acosh(x)`	Umkehrfunktion des Hyperbelkosinus von x zurückgeben
`math.asinh(x)`	Umkehrfunktion des Hyperbelsinus von x zurückgeben
`math.atanh(x)`	Umkehrfunktion des Hyperbeltangens von x zurückgeben
`math.cosh(x)`	Hyperbelsinus von x zurückgeben
`math.sinh(x)`	Hyperbelsinus von x zurückgeben
`math.tanh(x)`	Hyperbeltangens von x zurückgeben

Es gibt auch zwei mathematische Konstanten. Der Wert von `math.pi` ist die mathematische Konstante Pi. Der Wert von `math.e` ist die mathematische Konstante e.

Verwendung von Nicht-ASCII-Zeichen

Damit Nicht-ASCII-Zeichen verwendet werden können, ist bei Python eine explizite Codierung und Decodierung von Zeichenfolgen in Unicode erforderlich. In IBM SPSS Modeler wird davon ausgegangen, dass Python-Scripts in UTF-8 codiert sind, einer Standard-Unicode-Codierung, die Nicht-ASCII-Zeichen unterstützt. Das folgende Script wird kompiliert, da der Python-Compiler von SPSS Modeler auf UTF-8 gesetzt wurde.

```
stream = modeler.script.stream()
filenode = stream.createAt("variablefile", "テストノード", 96, 64)
```

Der resultierende Knoten hat jedoch eine falsche Beschriftung.

Abbildung 3. Falsch dargestellte Knotenbeschriftung mit Nicht-ASCII

Die Beschriftung ist falsch, da das Zeichenfolgeliteral von Python in eine ASCII-Zeichenfolge konvertiert wurde.

Python erlaubt die Angabe von Unicode-Zeichenfolgeliteralen, indem das Zeichenpräfix u vor dem Zeichenfolgeliteral hinzugefügt wird:

```
stream = modeler.script.stream()
filenode = stream.createAt("variablefile", u"テストノード", 96, 64)
```

Hiermit wird eine Unicode-Zeichenfolge erstellt und die Beschriftung wird korrekt dargestellt.

Abbildung 4. Korrekt dargestellte Knotenbeschriftung mit Nicht-ASCII

Die Verwendung von Python und Unicode ist ein umfassendes Thema, das über den Rahmen dieses Dokuments hinausgeht. Es gibt viele Handbücher und Onlineressourcen, die sich ausführlich mit diesem Thema befassen.

Objektorientierte Programmierung

Objektorientierte Programmierung basiert auf der Erstellung eines Modells des Zielproblems in Ihren Programmen. Objektorientierte Programmierung verringert Programmierfehler und fördert die Wiederverwendung von Code. Python ist eine objektorientierte Sprache. In Python definierte Objekte haben die folgenden Funktionen:

- **Identität.** Jedes Objekt muss eindeutig sein und diese Bedingung muss testbar sein. Zu diesem Zweck gibt es die Tests `is` und `is not`.
- **Status.** Jedes Objekt muss den Status speichern können. Zu diesem Zweck gibt es Attribute wie Felder und Instanzvariablen.
- **Verhalten.** Jedes Objekt muss seinen Status manipulieren können. Zu diesem Zweck gibt es Methoden.

Python umfasst die folgenden Funktionen zur Unterstützung objektorientierter Programmierung:

- **Klassenbasierte Objekterstellung.** Klassen sind Vorlagen für die Erstellung von Objekten. Objekte sind Datenstrukturen mit zugehörigem Verhalten.
- **Vererbung mit Polymorphie.** Python unterstützt sowohl Einfach- als auch Mehrfachvererbung. Alle Python-Instanzdefinitionsmethoden sind polymorph und können von Unterklassen außer Kraft gesetzt werden.
- **Kapselung mit Ausblenden von Daten.** Python erlaubt das Ausblenden von Attributen. Wenn Attribute ausgeblendet sind, kann von außerhalb der Klasse nur über Methoden der Klasse auf sie zugegriffen werden. Klassen implementieren Methoden, um die Daten zu ändern.

Definieren einer Klasse

In einer Python-Klasse können sowohl Variablen als auch Methoden definiert werden. Anders als in Java können Sie in Python eine beliebige Anzahl öffentlicher Klassen pro Quellendatei (oder *Modul*) definieren. Ein Modul in Python kann daher als Entsprechung eines Pakets in Java betrachtet werden.

In Python werden Klassen mit der Anweisung `class` definiert. Die Anweisung `class` hat das folgende Format:

```
class name (superclasses): statement
```

oder

```
class name (superclasses):
    assignment
    .
    .
    function
    .
    .
```

Beim Definieren einer Klasse haben Sie die Möglichkeit, null oder mehr *Zuordnungsanweisungen* anzugeben. Diese erstellen Klassenattribute, die von allen Instanzen der Klasse gemeinsam genutzt werden. Sie können auch null oder mehr *Funktionsdefinitionen* angeben. Diese Funktionsdefinitionen erstellen Methoden. Die Superklassenliste ist optional.

Der Klassenname sollte in seinem Bereich, d. h. in einem Modul, einer Funktion oder einer Klasse, eindeutig sein. Sie können mehrere Variablen zum Verweis auf dieselbe Klasse definieren.

Erstellen einer Klasseninstanz

Klassen werden zur Aufnahme von (gemeinsam genutzten) Klassenattributen oder zum Erstellen von Klasseninstanzen verwendet. Wenn Sie eine Instanz einer Klasse erstellen wollen, rufen Sie die Klasse so auf, als wäre sie eine Funktion. Beispiel einer Klasse:

```
class MyClass:
    pass
```

Hier wird die Anweisung `pass` verwendet, da eine Anweisung zum Abschließen der Klasse erforderlich ist. Vom Programm aus ist jedoch keine Aktion erforderlich.

Die folgende Anweisung erstellt eine Instanz der Klasse `MyClass`:

```
x = MyClass()
```

Hinzufügen von Attributen zu einer Klasseninstanz

Anders als in Java können Clients in Python Attribute einer Instanz einer Klasse hinzufügen. Nur diese eine Instanz wird geändert. Wenn Sie z. B. Attribute einer Instanz x hinzufügen wollen, legen Sie neue Werte für diese Instanz fest:

```
x.attr1 = 1
x.attr2 = 2
    .
    .
x.attrN = n
```

Definieren von Klassenattributen und Methoden

Jede Variable, die in einer Klasse gebunden ist, ist ein *Klassenattribut*. Jede in einer Klasse definierte Funktion ist eine *Methode*. Methoden erhalten als erstes Argument eine Instanz der Klasse, die normaler-

weise `self` genannt wird. Sie könnten z. B. den folgenden Code eingeben, um einige Klassenattribute und Methoden zu definieren:

```
class MyClass
    attr1 = 10        #class attributes
    attr2 = "hello"

    def method1(self):
        print MyClass.attr1   #reference the class attribute

    def method2(self):
        print MyClass.attr2   #reference the class attribute

    def method3(self, text):
        self.text = text        #instance attribute
        print text, self.text   #print my argument and my attribute

    method4 = method3   #make an alias for method3
```

In einer Klasse sollten Sie alle Verweise auf Klassenattribute mit dem Klassennamen qualifizieren, z. B. `MyClass.attr1`. Alle Verweise auf Instanzattribute sollten mit der Variablen `self` qualifiziert werden, z. B. `self.text`. Außerhalb der Klasse sollte Sie alle Verweise auf Klassenattribute mit dem Klassennamen (z. B. `MyClass.attr1`) oder mit einer Instanz der Klasse qualifizieren (z. B. `x.attr1`, wobei x eine Instanz der Klasse ist). Außerhalb der Klasse sollten alle Verweise auf Instanzvariablen mit einer Instanz der Klasse qualifiziert werden, z. B. `x.text`.

Ausgeblendete Variablen

Daten können durch das Erstellen *nicht öffentlicher* Variablen ausgeblendet werden. Auf nicht öffentliche Variablen kann nur von der Klasse selbst zugegriffen werden. Wenn Sie Namen der Form `__xxx` oder `__xxx_yyy` deklarieren, d. h. mit zwei vorausgehenden Unterstrichen, fügt der Python-Parser dem deklarierten Namen automatisch den Klassennamen hinzu und erstellt so ausgeblendete Variablen. Beispiel:

```
class MyClass:
    __attr = 10   #private class attribute

    def method1(self):
        pass

    def method2(self, p1, p2):
        pass

    def __privateMethod(self, text):
        self.__text = text    #private attribute
```

Anders als in Java müssen in Python alle Verweise auf Instanzvariablen mit `self` qualifiziert werden; es gibt keine implizierte Verwendung von `this`.

Vererbung

Die Fähigkeit zur Vererbung von Klassen ist elementar für die objektorientierte Programmierung. Python unterstützt sowohl Einfach- als auch Mehrfachvererbung. *Einfachvererbung* bedeutet, dass es nur eine Superklasse geben kann. *Mehrfachvererbung* bedeutet, dass es mehrere Superklasse geben kann.

Die Vererbung wird durch Unterklassenbildung anderer Klassen implementiert. Eine beliebige Anzahl von Python-Klassen können Superklassen sein. In der Jython-Implementierung von Python kann die Vererbung direkt oder indirekt nur von einer Java-Klasse erfolgen. Es muss keine Superklasse angegeben werden.

Jedes Attribut oder jede Methode in einer Superklasse ist auch in jeder Unterklasse enthalten und kann von der Klasse selbst oder von jedem beliebigen Client verwendet werden, sofern das Attribut oder die Methode nicht ausgeblendet ist. Jede Instanz einer Unterklasse kann überall verwendet werden, wo auch eine Instanz einer Superklasse verwendet werden kann; dies ist ein Beispiel für *Polymorphie*. Durch diese Funktionen wird die Wiederverwendung ermöglicht und die Erweiterung erleichtert.

Beispiel

```
class Class1: pass    #no inheritance
class Class2: pass
class Class3(Class1): pass     #single inheritance
class Class4(Class3, Class2): pass     #multiple inheritance
```

Kapitel 3. Scripting in IBM SPSS Modeler

Scripttypen

In IBM SPSS Modeler gibt es drei Typen von Scripts:

- *Stream-Scripts* werden verwendet, um die Ausführung eines einzelnen Streams zu steuern. Sie werden im Stream gespeichert.
- *Superknotenscripts* werden verwendet, um das Verhalten von Superknoten zu steuern.
- *Standalone- oder Sitzungsscripts* können verwendet werden, um die Ausführung über eine Reihe unterschiedlicher Streams zu koordinieren.

Es stehen verschiedene Methoden zur Verfügung, die in Scripts in IBM SPSS Modeler verwendet werden können. Mit diesen können Sie auf eine Vielzahl von SPSS Modeler-Funktionen zugreifen. Diese Methoden werden auch in Kapitel 4, „Scripting-API", auf Seite 43 zur Erstellung erweiterter Funktionen verwendet.

Streams, Superknotenstreams und Diagramme

Meistens bedeutet der Begriff *Stream* dasselbe, unabhängig davon, ob es sich um einen Stream handelt, der aus einer Datei geladen oder der in einem Superknoten verwendet wird. Im Allgemeinen ist damit eine Sammlung von Knoten gemeint, die miteinander verbunden sind und ausgeführt werden können. Beim Scripting werden jedoch nicht alle Operationen an allen Stellen unterstützt. Ein Scriptautor sollte sich deshalb bewusst sein, welche Streamvariante er verwendet.

Streams

Ein Stream ist der Hauptdokumenttyp von IBM SPSS Modeler. Er kann gespeichert, geladen, bearbeitet und ausgeführt werden. Streams können auch Parameter, globale Werte, ein Script und weitere zugehörige Informationen haben.

Superknotenstreams

Ein *Superknotenstream* ist der Typ von Stream, der in einem Superknoten verwendet wird. Wie ein normaler Stream enthält er Knoten, die miteinander verbunden sind. Superknotenstreams unterscheiden sich durch eine Reihe von Punkten von einem normalen Stream:

- Parameter und Scripts sind dem Superknoten zugeordnet, dem der Superknotenstream gehört, und nicht dem Superknotenstream selbst.
- Superknotenstreams haben je nach Typ des Superknotens zusätzliche Ein- und Ausgabe-Verbindungsknoten. Diese Verbindungsknoten werden verwendet, um Informationen in den und aus dem Superknotenstream zu leiten. Sie werden automatisch bei der Erstellung des Superknotens erstellt.

Diagramme

Der Begriff *Diagramm* deckt die Funktionen ab, die sowohl von normalen Streams als auch von Superknotenstreams unterstützt werden, z. B. das Hinzufügen und Entfernen von Knoten und das Ändern von Verbindungen zwischen den Knoten.

Ausführen eines Streams

Das folgende Beispiel führt alle ausführbaren Knoten im Stream aus. Es ist der einfachste Typ von Stream-Script:

```
modeler.script.stream().runAll(None)
```

Das folgende Beispiel führt ebenfalls alle ausführbaren Knoten im Stream aus:

```
stream = modeler.script.stream()
stream.runAll(None)
```

In diesem Beispiel wird der Stream in einer Variablen namens `stream` gespeichert. Das Speichern des Streams in einer Variablen ist hilfreich, da typischerweise ein Script verwendet wird, um entweder den Stream oder die Knoten in einem Stream zu ändern. Durch die Erstellung einer Variablen, die die Streamergebnisse speichert, ergibt sich ein knapperes Script.

Scripting-Kontext

Das Modul `modeler.script` stellt den Kontext bereit, in dem ein Script ausgeführt wird. Das Modul wird zur Laufzeit automatisch in ein SPSS Modeler-Script importiert. Das Modul definiert vier Funktionen, die ein Script mit Zugriff auf seine Ausführungsumgebung bereitstellen:

- Die Funktion `session()` gibt die Sitzung für das Script zurück. Die Sitzung definiert Informationen wie die Ländereinstellung und das SPSS Modeler-Back-End (entweder ein lokaler Prozess oder eine vernetzte SPSS Modeler Server-Instanz) für die Ausführung von Streams.
- Die Funktion `stream()` kann in Verbindung mit Stream- und Superknotenscripts verwendet werden. Diese Funktion gibt den Stream zurück, dem das ausgeführte Streamscript oder Superknotenscript gehört.
- Die Funktion `diagram()` kann in Verbindung mit Superknotenscripts verwendet werden. Diese Funktion gibt das Diagramm im Superknoten zurück. Bei anderen Scripttypen hat diese Funktion dieselbe Rückgabe wie die Funktion `stream()`.
- Die Funktion `supernode()` kann in Verbindung mit Superknotenscripts verwendet werden. Diese Funktion gibt den Superknoten zurück, dem das ausgeführte Script gehört.

Die vier Funktionen und ihre Ausgaben sind in der folgenden Tabelle zusammengefasst.

Tabelle 9. Zusammenfassung der `modeler.script`-Funktionen

Scripttyp	**`session()`**	**`stream()`**	**`diagram()`**	**`supernode()`**
Standalone	Gibt eine Sitzung zurück	Gibt den aktuellen verwalteten Stream zum Zeitpunkt des Scriptaufrufs zurück (z. B. den Stream, der über die Stapelmodusoption `-stream` übergeben wird), bzw. None.	Wie bei `stream()`	Nicht zutreffend
Stream	Gibt eine Sitzung zurück	Gibt einen Stream zurück	Wie bei `stream()`	Nicht zutreffend
Superknoten	Gibt eine Sitzung zurück	Gibt einen Stream zurück	Gibt einen Superknotenstream zurück	Gibt einen Superknoten zurück

Das Modul `modeler.script` definiert auch eine Möglichkeit zum Beenden des Scripts mit einem Beendigungscode. Die Funktion `exit(`*`exit-code`*`)` stoppt die Ausführung des Scripts und gibt den angegebenen ganzzahligen Beendigungscode zurück.

Eine der für einen Stream definierten Methoden ist `runAll(List)`. Diese Methode führt alle ausführbaren Knoten aus. Alle Modelle oder Ausgaben, die durch die Ausführung der Knoten generiert werden, werden der angegebenen Liste hinzugefügt.

Üblicherweise generiert eine Streamausführung Ausgaben wie Modelle, Grafiken oder andere Ausgaben. Zur Erfassung dieser Ausgabe kann ein Script eine Variable angeben, die in eine Liste initialisiert wird. Beispiel:

```
stream = modeler.script.stream()
results = []
stream.runAll(results)
```

Nach Abschluss der Ausführung kann über die Liste `results` auf alle Objekte zugegriffen werden, die von der Ausführung generiert wurden.

Referenzieren vorhandener Knoten

Ein Stream ist oft mit einigen Parametern vordefiniert, die geändert werden müssen, bevor der Stream ausgeführt werden kann. Die Änderung dieser Parameter umfasst die folgenden Tasks:

1. Suchen der Knoten im entsprechenden Stream.
2. Ändern der Knoten- und/oder Streameinstellungen.

Suchen von Knoten

Streams bieten eine Reihe von Möglichkeiten zum Suchen eines vorhandenen Knotens. Diese Methoden werden in der folgenden Tabelle zusammengefasst.

Tabelle 10. Methoden zum Lokalisieren eines vorhandenen Knotens

Methode	Rückgabetyp	Beschreibung
`s.findAll(type, label)`	Datenerfassung	Gibt eine Liste aller Knoten mit dem angegebenen Typ und der angegebenen Beschriftung zurück. Entweder der Typ oder die Beschriftung kann None sein. In diesem Fall wird der jeweils andere Parameter verwendet.
`s.findAll(filter, recursive)`	Datenerfassung	Gibt eine Sammlung aller Knoten zurück, die vom angegebenen Filter akzeptiert werden. Wenn das rekursive Flag `True` lautet, werden auch Superknoten im angegebenen Stream gesucht.
`s.findByID(id)`	Knoten	Gibt den Knoten mit der angegebenen ID zurück bzw. None, wenn kein derartiger Knoten vorhanden ist. Die Suche ist auf den aktuellen Stream eingeschränkt.

Tabelle 10. Methoden zum Lokalisieren eines vorhandenen Knotens (Forts.)		
Methode	**Rückgabetyp**	**Beschreibung**
`s.findByType(type, label)`	Knoten	Gibt den Knoten mit dem angegebenen Typ und/oder der angegebenen Beschriftung zurück. Entweder der Typ oder der Name kann None sein. In diesem Fall wird der jeweils andere Parameter verwendet. Wenn sich für mehreren Knoten eine Übereinstimmung ergibt, wird ein beliebiger ausgewählt und zurückgegeben. Wenn sich für keinen Knoten eine Übereinstimmung ergibt, lautet der Rückgabewert None.
`s.findDownstream(fromNodes)`	Datenerfassung	Sucht in der angegebenen Liste von Knoten und gibt das Set von Knoten an, die den angegebenen Knoten nachgeordnet sind. Die zurückgegebene Liste enthält die ursprünglich bereitgestellten Knoten.
`s.findUpstream(fromNodes)`	Datenerfassung	Sucht in der angegebenen Liste von Knoten und gibt das Set von Knoten an, die den angegebenen Knoten vorgeordnet sind. Die zurückgegebene Liste enthält die ursprünglich bereitgestellten Knoten.
`s.findProcessorForID(String id, boolean recursive)`	Knoten	Gibt den Knoten mit der angegebenen ID zurück bzw. None, wenn kein derartiger Knoten vorhanden ist. Wenn das rekursive Flag `true` ist, werden alle zusammengesetzten Knoten in diesem Diagramm ebenfalls durchsucht.

Wenn z. B. ein Stream einen einzigen Filterknoten enthält, auf den das Script zugreifen muss, kann der Filterknoten mithilfe des folgenden Scripts gefunden werden:

```
stream = modeler.script.stream()
node = stream.findByType("filter", None)
...
```

Wenn die ID des Knotens (siehe Registerkarte "Anmerkungen" im Knotendialogfeld) bekannt ist, kann anhand der ID nach dem Knoten gesucht werden. Beispiel:

```
stream = modeler.script.stream()
node = stream.findByID("id32FJT71G2")  # the filter node ID
...
```

Festlegen von Eigenschaften

Knoten, Streams, Modelle und Ausgaben haben Eigenschaften, die abgerufen und in den meisten Fällen auch festgelegt werden können. Eigenschaften werden üblicherweise verwendet, um das Verhalten oder

Aussehen des Objekts zu ändern. Die verfügbaren Methoden für den Zugriff und das Festlegen von Objekteigenschaften sind in der folgenden Tabelle zusammengefasst.

Tabelle 11. Methoden zum Zugreifen auf Objekteigenschaften und zum Festlegen dieser Eigenschaften

Methode	Rückgabetyp	Beschreibung
`p.getPropertyValue(propertyName)`	Object	Gibt den Wert der angegebenen Eigenschaft zurück, bzw. None, wenn keine solche Eigenschaft vorhanden ist.
`p.setPropertyValue(propertyName, value)`	Nicht zutreffend	Definiert den Wert der angegebenen Eigenschaft.
`p.setPropertyValues(properties)`	Nicht zutreffend	Definiert die Werte der angegebenen Eigenschaften. Jeder Eintrag in der Eigenschaftszuordnung besteht aus einem Schlüssel, der den Eigenschaftsnamen und den Wert darstellt, der der entsprechenden Eigenschaft zugewiesen werden sollte.
`p.getKeyedPropertyValue(propertyName, keyName)`	Object	Gibt den Wert der angegebenen Eigenschaft und des zugehörigen Schlüssels zurück, bzw. None, wenn keine solche Eigenschaft oder kein solcher Schlüssel vorhanden ist.
`p.setKeyedPropertyValue(propertyName, keyName, value)`	Nicht zutreffend	Definiert den Wert der angegebenen Eigenschaft und des Schlüssels.

Wenn Sie z. B. den Wert eines Knotens Variable Datei am Anfang eines Streams festlegen wollen, können Sie das folgende Script verwenden:

```
stream = modeler.script.stream()
node = stream.findByType("variablefile", None)
node.setPropertyValue("full_filename", "$CLEO/DEMOS/DRUG1n")
...
```

Alternativ könnten Sie ein Feld aus einem Filterknoten filtern. In diesem Fall wird der Wert auch in den Feldnamen eingegeben. Beispiel:

```
stream = modeler.script.stream()
# Locate the filter node ...
node = stream.findByType("filter", None)
# ... and filter out the "Na" field
node.setKeyedPropertyValue("include", "Na", False)
```

Erstellen von Knoten und Ändern von Streams

In einigen Situationen wollen Sie möglicherweise vorhandenen Streams neue Knoten hinzufügen. Die Hinzufügung von Knoten zu vorhandenen Streams umfasst die folgenden Tasks:

1. Erstellen der Knoten.
2. Aktivieren von Links für die Knoten in der vorhandenen Streamfolge.

Erstellen von Knoten

Streams bieten eine Reihe von Möglichkeiten zum Erstellen von Knoten. Diese Methoden werden in der folgenden Tabelle zusammengefasst.

Tabelle 12. Methoden zum Erstellen von Knoten

Methode	Rückgabetyp	Beschreibung
`s.create(nodeType, name)`	Knoten	Erstellt einen Knoten des angegebenen Typs und fügt ihn dem angegebenen Stream hinzu.
`s.createAt(nodeType, name, x, y)`	Knoten	Erstellt einen Knoten des angegebenen Typs und fügt ihn dem angegebenen Stream an der angegebenen Position hinzu. Bei x < 0 oder y < 0 ist die Position nicht festgelegt.
`s.createModelApplier(modelOutput, name)`	Knoten	Erstellt einen Modellanwendungsknoten, der vom angegebenen Modellausgabeobjekt abgeleitet wird.

Sie können das folgende Script verwenden, um z. B. einen neuen Typknoten in einem Stream zu erstellen:

```
stream = modeler.script.stream()
# Create a new type node
node = stream.create("type", "My Type")
```

Aktivieren und Aufheben von Links für Knoten

Wenn in einem Stream ein neuer Knoten erstellt wird, muss er in der Folge von Knoten verbunden werden, bevor er verwendet werden kann. Streams bieten eine Reihe von Möglichkeiten zum Aktivieren und Aufheben von Links für Knoten. Diese Methoden werden in der folgenden Tabelle zusammengefasst.

Tabelle 13. Methoden zum Aktivieren und Aufheben von Links für Knoten

Methode	Rückgabetyp	Beschreibung
`s.link(source, target)`	Nicht zutreffend	Erstellt einen neuen Link zwischen dem Quellen- und dem Zielknoten.
`s.link(source, targets)`	Nicht zutreffend	Erstellt neue Links zwischen dem Quellenknoten und jedem Zielknoten in der angegebenen Liste.
`s.linkBetween(inserted, source, target)`	Nicht zutreffend	Verbindet einen Knoten zwischen zwei anderen Knoteninstanzen (dem Quellen- und dem Zielknoten) und legt die Position des eingefügten Knotens zwischen diesen beiden fest. Jeder direkte Link zwischen dem Quellen- und dem Zielknoten wird zuerst entfernt.

Tabelle 13. Methoden zum Aktivieren und Aufheben von Links für Knoten (Forts.)

Methode	Rückgabetyp	Beschreibung
`s.linkPath(path)`	Nicht zutreffend	Erstellt einen neuen Pfad zwischen Knoteninstanzen. Der erste Knoten wird mit dem zweiten verbunden, der zweite wird mit dem dritten verbunden usw.
`s.unlink(source, target)`	Nicht zutreffend	Entfernt jeden direkten Link zwischen dem Quellen- und dem Zielknoten.
`s.unlink(source, targets)`	Nicht zutreffend	Entfernt alle direkten Links zwischen dem Quellenknoten und jedem Objekt in der Zielliste.
`s.unlinkPath(path)`	Nicht zutreffend	Entfernt jeden Pfad zwischen Knoteninstanzen.
`s.disconnect(node)`	Nicht zutreffend	Entfernt alle Links zwischen dem angegebenen Knoten und allen anderen Knoten im angegebenen Stream.
`s.isValidLink(source, target)`	*Boolesch*	Gibt `True` zurück, wenn die Erstellung eines Links zwischen dem angegebenen Quellen- und Zielknoten zulässig wäre. Diese Methode prüft, dass beide Objekte zum angegebenen Stream gehören, dass der Quellenknoten einen Link bereitstellen kann, dass der Zielknoten einen Link empfangen kann und dass die Erstellung eines solchen Links keinen Zirkelbezug im Stream verursacht.

Das folgende Beispielscript führt die folgenden fünf Tasks durch:

1. Es erstellt einen Eingabeknoten "Variable Datei", einen Filterknoten und einen Tabellenausgabeknoten.
2. Es verbindet die Knoten miteinander.
3. Es legt den Dateinamen im Eingabeknoten "Variable Datei" fest.
4. Es filtert das Feld "Drug" aus der Ergebnisausgabe.
5. Es führt den Tabellenknoten aus.

```
stream = modeler.script.stream()
filenode = stream.createAt("variablefile", "My File Input ", 96, 64)
filternode = stream.createAt("filter", "Filter", 192, 64)
tablenode = stream.createAt("table", "Table", 288, 64)
stream.link(filenode, filternode)
stream.link(filternode, tablenode)
filenode.setPropertyValue("full_filename", "$CLEO_DEMOS/DRUG1n")
filternode.setKeyedPropertyValue("include", "Drug", False)
results = []
tablenode.run(results)
```

Importieren, Ersetzen und Löschen von Knoten

Neben dem Erstellen und Verbinden von Knoten ist es oft auch erforderlich, Knoten zu ersetzen oder aus dem Stream zu löschen. Die verfügbaren Methoden zum Importieren, Ersetzen und Löschen von Knoten sind in der folgenden Tabelle zusammengefasst.

Tabelle 14. Methoden zum Importieren, Ersetzen oder Löschen von Knoten

Methode	Rückgabetyp	Beschreibung
`s.replace(originalNode, replacementNode, discardOriginal)`	Nicht zutreffend	Ersetzt den angegebenen Knoten im angegebenen Stream. Sowohl der Originalknoten als auch der Ersetzungsknoten müssen dem angegebenen Stream gehören.
`s.insert(source, nodes, newIDs)`	Liste	Fügt Kopien der Knoten in der angegebenen Liste ein. Es wird angenommen, dass alle Knoten in der angegebenen Liste im angegebenen Stream enthalten sind. Das Flag `newIDs` gibt an, ob für jeden Knoten eine neue ID generiert werden soll oder ob die vorhandene ID kopiert und verwendet werden soll. Es wird angenommen, dass alle Knoten in einem Stream eine eindeutige ID haben. Dieses Flag muss daher auf `True` gesetzt werden, wenn der Quellenstream dem angegebenen Stream entspricht. Die Methode gibt die Liste neu eingefügter Knoten zurück, wobei die Reihenfolge der Knoten nicht definiert ist (d. h. die Reihenfolge entspricht nicht unbedingt der Reihenfolge der Knoten in der Eingabeliste).
`s.delete(node)`	Nicht zutreffend	Löscht den angegebenen Knoten aus dem angegebenen Stream. Der Knoten muss dem angegebenen Stream gehören.
`s.deleteAll(nodes)`	Nicht zutreffend	Löscht alle angegebenen Knoten aus dem angegebenen Stream. Alle Knoten in der Sammlung müssen dem angegebenen Stream gehören.
`s.clear()`	Nicht zutreffend	Löscht alle Knoten aus dem angegebenen Stream.

Traversieren durch Knoten in einem Stream

Eine allgemeine Voraussetzung ist die Ermittlung von Knoten, die einem bestimmten Knoten vorgeordnet oder nachgeordnet sind. Der Stream bietet eine Reihe von Methoden, die verwendet werden können, um diese Knoten zu ermitteln. Diese Methoden werden in der folgenden Tabelle zusammengefasst.

Tabelle 15. Methoden zum Angeben vorgeordneter und nachgeordneter Knoten

Methode	Rückgabetyp	Beschreibung
`s.iterator()`	Iterator	Gibt einen Iterator über die Knotenobjekte zurück, die im angegebenen Stream enthalten sind. Wenn der Stream zwischen Aufrufen der Funktion `next()` geändert wird, kann das Verhalten des Iterators undefiniert sein.
`s.predecessorAt(node, index)`	Knoten	Gibt den angegebenen unmittelbaren Vorgänger des angegebenen Knotens zurück, bzw. None, wenn der Index außerhalb des gültigen Bereichs liegt.
`s.predecessorCount(node)`	*Ganzzahl*	Gibt die Anzahl der unmittelbaren Vorgänger des angegebenen Knotens zurück.
`s.predecessors(node)`	Liste	Gibt die unmittelbaren Vorgänger des angegebenen Knotens zurück.
`s.successorAt(node, index)`	Knoten	Gibt den angegebenen unmittelbaren Nachfolger des angegebenen Knotens zurück, bzw. None, wenn der Index außerhalb des gültigen Bereichs liegt.
`s.successorCount(node)`	*Ganzzahl*	Gibt die Anzahl der unmittelbaren Nachfolger des angegebenen Knotens zurück.
`s.successors(node)`	Liste	Gibt die unmittelbaren Nachfolger des angegebenen Knotens zurück.

Entfernen von Elementen

Traditionelles Scripting unterstützt verschiedene Verwendungen des Befehls `clear`, z. B.:

- `clear outputs` löscht alle Ausgabeelemente aus der Managerpalette.
- `clear generated palette` löscht alle Modellnuggets aus der Modellpalette.
- `clear stream` entfernt den Inhalt eines Streams.

Python-Scripting unterstützt ein ähnliches Set von Funktionen. Mit dem Befehl `removeAll()` werden die Stream-, Ausgabe- und Modellmanager gelöscht. Beispiele:

- So löschen Sie den Stream-Manager:

```
session = modeler.script.session()
session.getStreamManager.removeAll()
```

- So löschen Sie den Ausgabemanager:

```
session = modeler.script.session()
session.getDocumentOutputManager().removeAll()
```

- So löschen Sie den Modellmanager:

```
session = modeler.script.session()
session.getModelOutputManager().removeAll()
```

Abrufen von Informationen zu Knoten

Knoten fallen in eine Reihe unterschiedlicher Kategorien wie Datenimport- und -exportknoten, Modellerstellungsknoten und andere Typen von Knoten. Jeder Knoten stellt eine Reihe von Methoden bereit, mit denen Informationen zum Knoten abgerufen werden können.

Die Methoden, mit denen die ID, der Name und die Beschriftung eines Knotens abgerufen werden können, sind in der folgenden Tabelle zusammengefasst.

Tabelle 16. Methoden zum Abrufen von ID, Name und Beschriftung eines Knotens

Methode	Rückgabetyp	Beschreibung
`n.getLabel()`	*string*	Gibt die Anzeigebeschriftung des angegebenen Knotens zurück. Die Beschriftung entspricht nur dann dem Wert der Eigenschaft `custom_name`, wenn diese Eigenschaft eine nicht leere Zeichenfolge ist und die Eigenschaft `use_custom_name` nicht definiert ist; andernfalls entspricht die Beschriftung dem Wert von `getName()`.
`n.setLabel(label)`	Nicht zutreffend	Legt die Anzeigebeschriftung des angegebenen Knotens fest. Wenn die neue Beschriftung eine nicht leere Zeichenfolge ist, wird sie der Eigenschaft `custom_name` zugewiesen und die Eigenschaft `use_custom_name` wird auf `False` gesetzt, damit die angegebene Beschriftung Vorrang hat; andernfalls wird der Eigenschaft `custom_name` eine leere Zeichenfolge zugewiesen und die Eigenschaft `use_custom_name` wird auf `True` gesetzt.
`n.getName()`	*string*	Gibt den Namen des angegebenen Knotens zurück.

Tabelle 16. Methoden zum Abrufen von ID, Name und Beschriftung eines Knotens (Forts.)		
Methode	**Rückgabetyp**	**Beschreibung**
`n.getID()`	*string*	Gibt die ID des angegebenen Knotens zurück. Bei jeder Erstellung eines neuen Knotens wird eine neue ID erzeugt. Die ID wird im Knoten als persistent definiert, wenn dieser als Teil eines Streams gespeichert wird, damit beim Öffnen des Streams die Knoten-IDs beibehalten werden. Wenn jedoch ein gespeicherter Knoten in einen Stream eingefügt wird, gilt der eingefügte Knoten als neues Objekt und ihm wird eine neue ID zugewiesen.

Die Methoden, mit denen weitere Informationen zu einem Knoten abgerufen werden können, sind in der folgenden Tabelle zusammengefasst.

Tabelle 17. Methoden zum Abrufen von Informationen zu einem Knoten		
Methode	**Rückgabetyp**	**Beschreibung**
`n.getTypeName()`	*string*	Gibt den Scriptnamen dieses Knotens zurück. Dies ist derselbe Name, der zum Erstellen einer neuen Instanz dieses Knoten verwendet werden könnte.
`n.isInitial()`	*boolesch*	Gibt `True` zurück, wenn dies ein *Ursprungsknoten* ist, d. h. ein Knoten, der am Anfang eines Streams auftritt.
`n.isInline()`	*boolesch*	Gibt `True` zurück, wenn dies ein *Inline-Knoten* ist, d. h. ein Knoten, der in der Mitte eines Streams auftritt.
`n.isTerminal()`	*boolesch*	Gibt `True` zurück, wenn dies ein *Endknoten* ist, d. h. ein Knoten, der am Ende eines Streams auftritt.
`n.getXPosition()`	*Ganzzahl*	Gibt den Offset der X-Position des Knotens im Stream an.
`n.getYPosition()`	*Ganzzahl*	Gibt den Offset der Y-Position des Knotens im Stream an.
`n.setXYPosition(x, y)`	Nicht zutreffend	Legt die Position des Knotens im Stream fest.
`n.setPositionBetween(source, target)`	Nicht zutreffend	Legt die Position des Knotens im Stream so fest, dass er zwischen den angegebenen Knoten angeordnet ist.

Tabelle 17. Methoden zum Abrufen von Informationen zu einem Knoten (Forts.)

Methode	**Rückgabetyp**	**Beschreibung**
`n.isCacheEnabled()`	*boolesch*	Gibt `True` zurück, wenn der Cache aktiviert ist; andernfalls wird `False` zurückgegeben.
`n.setCacheEnabled(val)`	Nicht zutreffend	Aktiviert oder inaktiviert den Cache für dieses Objekt. Wenn der Cache voll ist und das Caching inaktiviert wird, wird der Cache geleert.
`n.isCacheFull()`	*boolesch*	Gibt `True` zurück, wenn der Cache voll ist; andernfalls wird `False` zurückgegeben.
`n.flushCache()`	Nicht zutreffend	Leert den Cache dieses Knotens. Hat keine Auswirkung, wenn der Cache nicht aktiviert oder nicht voll ist.

Kapitel 4. Scripting-API

Einführung in die Scripting-API

Die Scripting-API bietet Zugriff auf eine Vielzahl von SPSS Modeler-Funktionen. Alle bisher beschriebenen Methoden sind Teil der API und können ohne weitere Importe implizit im Script aufgerufen werden. Wenn Sie jedoch auf die API-Klassen verweisen wollen, müssen Sie die API mit der folgenden Anweisung explizit importieren:

```
import modeler.api
```

Diese Importanweisung ist für viele Beispiele der Scripting-API erforderlich.

Eine vollständige Beschreibung der Klassen, Methoden und Parameter, die über die Scripting-API verfügbar sind, finden Sie im Handbuch *IBM SPSS Modeler Python Scripting API Reference Guide*.

Beispiel 1: Suchen nach Knoten mit einem benutzerdefinierten Filter

Im Abschnitt „Suchen von Knoten” auf Seite 33 ist ein Beispiel für die Suche nach einem Knoten in einem Stream enthalten, wobei der Typname des Knotens als Suchkriterium verwendet wird. In einigen Situationen ist eine allgemeinere Suche erforderlich. Diese kann mit der Klasse `NodeFilter` und der Streammethode `findAll()` implementiert werden. Diese Art von Suche umfasst die folgenden beiden Schritte:

1. Erstellen einer neuen Klasse, die `NodeFilter` erweitert und eine benutzerdefinierte Version der Methode `accept()` implementiert.
2. Aufrufen der Streammethode `findAll()` mit einer Instanz dieser neuen Klasse. Dadurch werden alle Knoten zurückgegeben, die die in der Methode `accept()` definierten Kriterien erfüllen.

Im folgenden Beispiel wird gezeigt, wie in einem Stream nach Knoten mit aktiviertem Knotencache gesucht werden kann. Die Liste der zurückgegebenen Knoten könnte verwendet werden, um die Caches dieser Knoten zu leeren oder zu inaktivieren.

```
import modeler.api

class CacheFilter(modeler.api.NodeFilter):
    """A node filter for nodes with caching enabled"""
    def accept(this, node):
        return node.isCacheEnabled()

cachingnodes = modeler.script.stream().findAll(CacheFilter(), False)
```

Beispiel 2: Benutzern ermöglichen, Verzeichnis- oder Dateiinformationen basierend auf ihren Berechtigungen abzurufen

Damit die PSAPI nicht für Benutzer geöffnet werden muss, kann eine Methode `session.getServerFileSystem()` über den Aufruf der PSAPI-Funktion verwendet werden, um ein Dateisystemobjekt zu erstellen.

Das folgende Beispiel zeigt, wie ein Benutzer Verzeichnis- oder Dateiinformationen basierend auf den Berechtigungen des Benutzers abrufen kann, der eine Verbindung zu IBM SPSS Modeler Server herstellt.

```
import modeler.api
stream = modeler.script.stream()
sourceNode = stream.findByID('')
session = modeler.script.session()
fileSystem = session.getServerFileSystem()
```

```
parameter = stream.getParameterValue('VPATH')
serverDirectory = fileSystem.getServerFile(parameter)
files = fileSystem.getFiles(serverDirectory)
for f in files:
    if f.isDirectory():
        print 'Directory:'
    else:
        print 'File:'
        sourceNode.setPropertyValue('full_filename',f.getPath())
        break
    print f.getName(),f.getPath()
stream.execute()
```

Metadaten: Informationen zu Daten

Da Knoten in einem Stream miteinander verbunden sind, sind Informationen zu den in jedem Knoten verfügbaren Spalten oder Feldern verfügbar. In der Modeler-Benutzerschnittstelle können Sie so z. B. auswählen, nach welchen Felder sortiert oder kumuliert werden soll. Diese Informationen werden als Datenmodell bezeichnet.

Scripts können auch auf das Datenmodell zugreifen, indem sie die Felder betrachten, die in einen Knoten eintreten oder aus einem Knoten austreten. Bei einigen Knoten sind das Eingabe- und das Ausgabedatenmodell gleich. Beispielsweise ordnet ein Knoten "sort" die Datensätze einfach um, ändert jedoch nicht das Datenmodell. Einige, wie der Knoten "derive", können neue Felder hinzufügen. Andere, wie der Knoten "filter", können Felder umbenennen oder entfernen.

Im folgenden Beispiel verwendet das Script den IBM SPSS Modeler `druglearn.str`-Standardstream und erstellt für jedes Feld ein Modell, bei dem eines der Eingabefelder gelöscht ist. Die Vorgehensweise sieht dabei folgendermaßen aus:

1. Zugreifen auf das Ausgabedatenmodell aus dem Typknoten
2. Durchlaufen jedes Felds im Ausgabedatenmodell in einer Schleife
3. Ändern des Knotens "filter" für jedes Eingabefeld
4. Ändern des Namens des zu erstellenden Modells
5. Ausführen des Modellerstellungsknotens

Anmerkung: Denken Sie vor der Ausführung des Scripts im `druglean.str`-Stream daran, als Scriptsprache Python festzulegen (der Stream wurde in einer früheren Version von IBM SPSS Modeler erstellt, weshalb "Traditionell" als Scriptsprache für den Stream festgelegt ist).

```
import modeler.api

stream = modeler.script.stream()
filternode = stream.findByType("filter", None)
typenode = stream.findByType("type", None)
c50node = stream.findByType("c50", None)
# Always use a custom model name
c50node.setPropertyValue("use_model_name", True)

lastRemoved = None
fields = typenode.getOutputDataModel()
for field in fields:
    # If this is the target field then ignore it
    if field.getModelingRole() == modeler.api.ModelingRole.OUT:
        continue

    # Re-enable the field that was most recently removed
    if lastRemoved != None:
        filternode.setKeyedPropertyValue("include", lastRemoved, True)

    # Remove the field
    lastRemoved = field.getColumnName()
    filternode.setKeyedPropertyValue("include", lastRemoved, False)

    # Set the name of the new model then run the build
    c50node.setPropertyValue("model_name", "Exclude " + lastRemoved)
    c50node.run([])
```

Das Datenmodellobjekt bietet eine Reihe von Methoden für den Zugriff auf Informationen zu den Feldern oder Spalten im Datenmodell. Diese Methoden werden in der folgenden Tabelle zusammengefasst.

Tabelle 18. Methoden des Datenmodellobjekts für den Zugriff auf Informationen zu Feldern oder Spalten

Methode	**Rückgabetyp**	**Beschreibung**
`d.getColumnCount()`	*Ganzzahl*	Gibt die Anzahl der Spalten im Datenmodell zurück.
`d.columnIterator()`	Iterator	Gibt einen Iterator zurück, der seinerseits jede Spalte in der "natürlichen" Einfügereihenfolge zurückgibt. Der Iterator gibt Spalteninstanzen zurück.
`d.nameIterator()`	Iterator	Gibt einen Iterator zurück, der seinerseits den Namen jeder Spalte in der "natürlichen" Einfügereihenfolge zurückgibt.
`d.contains(name)`	*boolesch*	Gibt `True` zurück, wenn eine Spalte mit dem angegebenen Namen in diesem Datenmodell vorhanden ist; andernfalls wird `False` zurückgegeben.
`d.getColumn(name)`	Spalte	Gibt die Spalte mit dem angegebenen Namen zurück.
`d.getColumnGroup(name)`	Spaltengruppe	Gibt die angegebene Spaltengruppe zurück, bzw. None, wenn keine solche Spaltengruppe vorhanden ist.
`d.getColumnGroupCount()`	*Ganzzahl*	Gibt die Anzahl der Spaltengruppen in diesem Datenmodell zurück.
`d.columnGroupIterator()`	Iterator	Gibt einen Iterator zurück, der nacheinander jede Spaltengruppe zurückgibt.
`d.toArray()`	Spalte[]	Gibt das Datenmodell als Array von Spalten zurück. Die Spalten sind in ihrer "natürlichen" Einfügereihenfolge geordnet.

Jedes Feld (Spaltenobjekt) umfasst eine Reihe von Methoden für den Zugriff auf Informationen zu der Spalte. Die folgende Tabelle enthält eine Auswahl davon.

Tabelle 19. Spaltenobjektmethoden für den Zugriff auf Informationen zu der Spalte

Methode	**Rückgabetyp**	**Beschreibung**
`c.getColumnName()`	*string*	Gibt den Namen der Spalte zurück.
`c.getColumnLabel()`	*string*	Gibt die Beschriftung der Spalte zurück bzw. eine leere Zeichenfolge, wenn der Spalte keine Beschriftung zugeordnet ist.

Tabelle 19. Spaltenobjektmethoden für den Zugriff auf Informationen zu der Spalte (Forts.)

Methode	Rückgabetyp	Beschreibung
`c.getMeasureType()`	Maßtyp	Gibt den Maßtyp für die Spalte zurück.
`c.getStorageType()`	Speichertyp	Gibt den Speichertyp für die Spalte zurück.
`c.isMeasureDiscrete()`	*boolesch*	Gibt `True` zurück, wenn die Spalte diskret ist. Spalten, die entweder ein Set oder ein Flag sind, werden als diskret betrachtet.
`c.isModelOutputColumn()`	*boolesch*	Gibt `True` zurück, wenn die Spalte eine Modellausgabespalte ist.
`c.isStorageDatetime()`	*boolesch*	Gibt `True` zurück, wenn der Speicher der Spalte ein Uhrzeit-, ein Datum- oder ein Zeitmarkenwert ist.
`c.isStorageNumeric()`	*boolesch*	Gibt `True` zurück, wenn der Speicher der Spalte eine ganze oder eine reelle Zahl ist.
`c.isValidValue(value)`	*boolesch*	Gibt `True` zurück, wenn der angegebene Wert für diesen Speicher gültig ist, und gibt `valid` zurück, wenn die gültigen Spaltenwerte bekannt sind.
`c.getModelingRole()`	Modellierungsrolle	Gibt die Modellierungsrolle für die Spalte zurück.
`c.getSetValues()`	Object[]	Gibt ein Array gültiger Werte für die Spalte zurück, bzw. `None`, wenn die Werte nicht bekannt sind oder wenn die Spalte kein Set ist.
`c.getValueLabel(value)`	*string*	Gibt die Beschriftung für den Wert in der Spalte zurück bzw. eine leere Zeichenfolge, wenn dem Wert keine Beschriftung zugeordnet ist.
`c.getFalseFlag()`	Object	Gibt den Indikatorwert "false" für die Spalte zurück, bzw. `None`, wenn der Wert nicht bekannt ist oder wenn die Spalte kein Flag ist.
`c.getTrueFlag()`	Object	Gibt den Indikatorwert "true" für die Spalte zurück, bzw. `None`, wenn der Wert nicht bekannt ist oder wenn die Spalte kein Flag ist.

Tabelle 19. Spaltenobjektmethoden für den Zugriff auf Informationen zu der Spalte (Forts.)		
Methode	**Rückgabetyp**	**Beschreibung**
`c.getLowerBound()`	Object	Gibt den unteren Grenzwert für die Werte in der Spalte zurück, bzw. None, wenn der Wert nicht bekannt ist oder wenn die Spalte nicht fortlaufend ist.
`c.getUpperBound()`	Object	Gibt den oberen Grenzwert für die Werte in der Spalte zurück, bzw. None, wenn der Wert nicht bekannt ist oder wenn die Spalte nicht fortlaufend ist.

Für die meisten der Methoden, die auf Informationen zu einer Spalte zugreifen, sind entsprechende Methoden im Datenmodellobjekt selbst definiert. Die folgenden beiden Anweisungen sind beispielsweise funktional entsprechend:

```
dataModel.getColumn("someName").getModelingRole()
dataModel.getModelingRole("someName")
```

Zugriff auf generierte Objekte

Bei der Ausführung eines Streams werden in der Regel zusätzliche Ausgabeobjekte erzeugt. Diese zusätzlichen Objekte könnten ein neues Modell oder eine Ausgabe sein, die Informationen bereitstellt, die in nachfolgenden Ausführungen verwendet werden.

Im Beispiel unten wird der Stream `druglearn.str` erneut als Ausgangspunkt für den Stream verwendet. In diesem Beispiel werden alle Knoten im Stream ausgeführt und die Ergebnisse werden in einer Liste gespeichert. Das Script durchläuft dann die Ergebnisse in einer Schleife. Alle Modellausgaben, die das Ergebnis der Ausführung sind, werden als IBM SPSS Modeler-Modelldatei (`.gm`) gespeichert und das Modell wird im PMML-Format exportiert.

```
import modeler.api

stream = modeler.script.stream()

# Set this to an existing folder on your system.
# Include a trailing directory separator
modelFolder = "C:/temp/models/"

# Execute the stream
models = []
stream.runAll(models)

# Save any models that were created
taskrunner = modeler.script.session().getTaskRunner()
for model in models:
    # If the stream execution built other outputs then ignore them
    if not(isinstance(model, modeler.api.ModelOutput)):
        continue

    label = model.getLabel()
    algorithm = model.getModelDetail().getAlgorithmName()

    # save each model...
    modelFile = modelFolder + label + algorithm + ".gm"
    taskrunner.saveModelToFile(model, modelFile)

    # ...and export each model PMML...
    modelFile = modelFolder + label + algorithm + ".xml"
    taskrunner.exportModelToFile(model, modelFile, modeler.api.FileFormat.XML)
```

Die Klasse der ausführbaren Komponente bietet eine praktische Möglichkeit zum Ausführen verschiedener allgemeiner Tasks. Die in dieser Klasse verfügbaren Methoden werden in der folgenden Tabelle zusammengefasst.

Tabelle 20. Methode der Klasse der ausführbaren Komponente zum Ausführen allgemeiner Tasks

Methode	Rückgabetyp	Beschreibung
`t.createStream(name, autoConnect, autoManage)`	Stream	Erstellt einen neuen Stream und gibt ihn zurück. Für Code, der Streams nicht öffentlich erstellen muss, ohne sie dem Benutzer sichtbar zu machen, sollte das Flag `autoManage` auf `False` gesetzt werden.
`t.exportDocumentToFile(documentOutput, filename, fileFormat)`	Nicht zutreffend	Exportiert die Streambeschreibung unter Verwendung des angegebenen Dateiformats in eine Datei.
`t.exportModelToFile(modelOutput, filename, fileFormat)`	Nicht zutreffend	Exportiert das Modell unter Verwendung des angegebenen Dateiformats in eine Datei.
`t.exportStreamToFile(stream, filename, fileFormat)`	Nicht zutreffend	Exportiert den Stream unter Verwendung des angegebenen Dateiformats in eine Datei.
`t.insertNodeFromFile(filename, diagram)`	Knoten	Liest einen Knoten aus der angegebenen Datei, gibt ihn zurück und fügt ihn in das angegebene Diagramm ein. Damit können sowohl Knoten- als auch Superknotenobjekte gelesen werden.
`t.openDocumentFromFile(filename, autoManage)`	Dokumentausgabe	Liest ein Dokument aus der angegebenen Datei und gibt es zurück.
`t.openModelFromFile(filename, autoManage)`	Modellausgabe	Liest ein Modell aus der angegebenen Datei und gibt es zurück.
`t.openStreamFromFile(filename, autoManage)`	Stream	Liest einen Stream aus der angegebenen Datei und gibt ihn zurück.
`t.saveDocumentToFile(documentOutput, filename)`	Nicht zutreffend	Speichert das Dokument an der angegebenen Dateiposition.
`t.saveModelToFile(modelOutput, filename)`	Nicht zutreffend	Speichert das Modell an der angegebenen Dateiposition.
`t.saveStreamToFile(stream, filename)`	Nicht zutreffend	Speichert den Stream an der angegebenen Dateiposition.

Fehler bearbeiten

Die Python-Sprache bietet Fehlerbehandlung über den Codeblock `try...except`. Dieser kann in Scripts verwendet werden, um Ausnahmebedingungen abzufangen und um Probleme zu behandeln, die sonst zur Beendigung des Scripts führen würden.

Im Beispielscript unten wird versucht, ein Modell aus IBM SPSS Collaboration and Deployment Services Repository abzurufen. Diese Operation kann dazu führen, dass eine Ausnahmebedingung ausgelöst wird. Beispielsweise könnten die Berechtigungsnachweise für die Repository-Anmeldung nicht korrekt eingerichtet sein oder der Repository-Pfad ist falsch. Im Script kann dies dazu führen, dass `ModelerException` ausgelöst wird (alle von IBM SPSS Modeler generierten Ausnahmebedingungen werden von `modeler.api.ModelerException` abgeleitet).

```
import modeler.api

session = modeler.script.session()
try:
    repo = session.getRepository()
    m = repo.retrieveModel("/some-non-existent-path", None, None, True)
    # print goes to the Modeler UI script panel Debug tab
    print "Everything OK"
except modeler.api.ModelerException, e:
    print "An error occurred:", e.getMessage()
```

Anmerkung: Einige Scriptoperation können dazu führen, dass Java-Standardausnahmebedingungen ausgelöst werden. Diese sind nicht von `ModelerException` abgeleitet. Zum Abfangen aller Java-Ausnahmebedingungen kann ein zusätzlicher except-Block verwendet werden. Beispiel:

```
import modeler.api
import java.lang.Exception

session = modeler.script.session()
try:
    repo = session.getRepository()
    m = repo.retrieveModel("/some-non-existent-path", None, None, True)
    # print goes to the Modeler UI script panel Debug tab
    print "Everything OK"
except modeler.api.ModelerException, e:
    print "An error occurred:", e.getMessage()
except java.lang.Exception, e:
    print "A Java exception occurred:", e.getMessage()
```

Stream-, Sitzungs- und Superknotenparameter

Parameter sind eine praktische Möglichkeit, um Werte zur Laufzeit zu übergeben, statt sie direkt im Script fest zu codieren. Parameter und ihre Werte werden auf dieselbe Weise definiert wie für Streams, d. h. als Einträge in der Parametertabelle eines Streams oder Superknotens oder als Parameter in der Befehlszeile. Die Stream- und Superknotenklassen implementieren eine Gruppe von Funktionen, die vom ParameterProvider-Objekt definiert werden (siehe folgende Tabelle). Die Sitzung stellt einen Aufruf `getParameters()` bereit, der ein Objekt zurückgibt, das diese Funktionen definiert.

Tabelle 21. Vom ParameterProvider-Objekt definierte Funktionen

Methode	Rückgabetyp	Beschreibung
`p.parameterIterator()`	Iterator	Gibt einen Iterator von Parameternamen für dieses Objekt zurück.
`p.getParameterDefinition(parameterName)`	ParameterDefinition	Gibt die Parameterdefinition für den Parameter mit dem angegebenen Namen zurück bzw.None, wenn kein solcher Parameter in diesem Provider existiert. Das Ergebnis kann eine Momentaufnahme der Definition zum Zeitpunkt des Aufrufs der Methode sein und spiegelt nicht unbedingt nachfolgende Änderungen am Parameter durch den Provider wider.

Tabelle 21. Vom ParameterProvider-Objekt definierte Funktionen (Forts.)		
Methode	**Rückgabetyp**	**Beschreibung**
`p.getParameterLabel(parameterName)`	*string*	Gibt die Beschriftung des angegebenen Parameters zurück bzw. None, wenn kein solcher Parameter vorhanden ist.
`p.setParameterLabel(parameterName, label)`	Nicht zutreffend	Definiert die Beschriftung des angegebenen Parameters.
`p.getParameterStorage(parameterName)`	ParameterStorage	Gibt den Speicher des angegebenen Parameters zurück bzw. None, wenn kein solcher Parameter vorhanden ist.
`p.setParameterStorage(parameterName, storage)`	Nicht zutreffend	Definiert den Speicher des angegebenen Parameters.
`p.getParameterType(parameterName)`	ParameterType	Gibt den Typ des angegebenen Parameters zurück bzw. None, wenn kein solcher Parameter vorhanden ist.
`p.setParameterType(parameterName, type)`	Nicht zutreffend	Definiert den Typ des angegebenen Parameters.
`p.getParameterValue(parameterName)`	Object	Gibt den Wert des angegebenen Parameters zurück bzw. None, wenn kein solcher Parameter vorhanden ist.
`p.setParameterValue(parameterName, value)`	Nicht zutreffend	Definiert den Wert des angegebenen Parameters.

Im folgenden Beispiel aggregiert das Script einige Telekommunikationsdaten, um die Region mit den niedrigsten durchschnittlichen Einnahmen zu ermitteln. Dann wird ein Streamparameter mit dieser Region definiert. Dieser Streamparameter wird dann in einem Auswahlknoten verwendet, um diese Region aus den Daten auszuschließen, bevor mit den restlichen Daten ein Abwanderungsmodell erstellt wird.

Dieses Beispiel ist konstruiert, da das Script den Auswahlknoten selbst generiert und daher den korrekten Wert direkt in den Ausdruck für den Auswahlknoten generiert haben könnte. Streams sind jedoch in der Regel vordefiniert, weshalb die Festlegung von Parametern auf diese Weise ein nützliches Beispiel darstellt.

Der erste Teil des Beispielscripts erstellt den Streamparameter, der die Region mit den niedrigsten durchschnittlichen Einnahmen enthält. Das Script erstellt auch die Knoten in der Aggregationsverzweigung und in der Modellerstellungsverzweigung und verbindet diese miteinander.

```
import modeler.api

stream = modeler.script.stream()

# Initialize a stream parameter
stream.setParameterStorage("LowestRegion", modeler.api.ParameterStorage.INTEGER)

# First create the aggregation branch to compute the average income per region
statisticsimportnode = stream.createAt("statisticsimport", "SPSS File", 114, 142)
statisticsimportnode.setPropertyValue("full_filename", "$CLEO_DEMOS/telco.sav")
statisticsimportnode.setPropertyValue("use_field_format_for_storage", True)

aggregatenode = modeler.script.stream().createAt("aggregate", "Aggregate", 294, 142)
aggregatenode.setPropertyValue("keys", ["region"])
aggregatenode.setKeyedPropertyValue("aggregates", "income", ["Mean"])
```

```
tablenode = modeler.script.stream().createAt("table", "Table", 462, 142)

stream.link(statisticsimportnode, aggregatenode)
stream.link(aggregatenode, tablenode)

selectnode = stream.createAt("select", "Select", 210, 232)
selectnode.setPropertyValue("mode", "Discard")
# Reference the stream parameter in the selection
selectnode.setPropertyValue("condition", "'region' = '$P-LowestRegion'")

typenode = stream.createAt("type", "Type", 366, 232)
typenode.setKeyedPropertyValue("direction", "churn", "Target")

c50node = stream.createAt("c50", "C5.0", 534, 232)

stream.link(statisticsimportnode, selectnode)
stream.link(selectnode, typenode)
stream.link(typenode, c50node)
```

Das Beispielscript erzeugt den folgenden Stream.

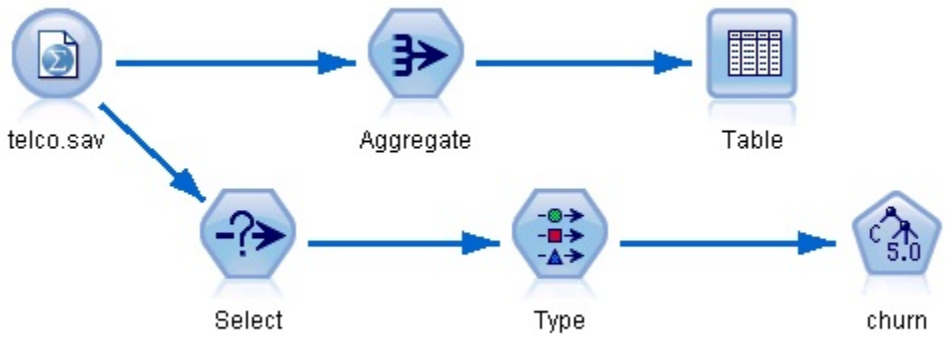

Abbildung 5. Stream, der aus dem Beispielscript resultiert

Der folgende Teil des Beispielscripts führt den Tabellenknoten am Ende der Aggregationsverzweigung aus.

```
# First execute the table node
results = []
tablenode.run(results)
```

Der folgende Teil des Beispielscripts greift auf die Tabellenausgabe zu, die durch die Ausführung des Tabellenknotens generiert wurde. Das Script durchläuft dann die Zeilen in der Tabelle und sucht nach den niedrigsten Durchschnittseinnahmen.

```
# Running the table node should produce a single table as output
table = results[0]

# table output contains a RowSet so we can access values as rows and columns
rowset = table.getRowSet()
min_income = 1000000.0
min_region = None

# From the way the aggregate node is defined, the first column
# contains the region and the second contains the average income
row = 0
rowcount = rowset.getRowCount()
while row < rowcount:
    if rowset.getValueAt(row, 1) < min_income:
        min_income = rowset.getValueAt(row, 1)
        min_region = rowset.getValueAt(row, 0)
    row += 1
```

Der folgende Teil des Scripts verwendet die Region mit den niedrigsten Durchschnittseinnahmen zur Definition des zuvor erstellten Streamparameters "LowestRegion". Das Script führt dann das Modellerstellungsprogramm aus, wobei die angegebene Region aus den Trainingsdaten ausgeschlossen wird.

```
# Check that a value was assigned
if min_region != None:
    stream.setParameterValue("LowestRegion", min_region)
else:
    stream.setParameterValue("LowestRegion", -1)

# Finally run the model builder with the selection criteria
c50node.run([])
```

Das vollständige Beispielscript ist im Folgenden aufgeführt.

```
import modeler.api

stream = modeler.script.stream()

# Create a stream parameter
stream.setParameterStorage("LowestRegion", modeler.api.ParameterStorage.INTEGER)

# First create the aggregation branch to compute the average income per region
statisticsimportnode = stream.createAt("statisticsimport", "SPSS File", 114, 142)
statisticsimportnode.setPropertyValue("full_filename", "$CLEO_DEMOS/telco.sav")
statisticsimportnode.setPropertyValue("use_field_format_for_storage", True)

aggregatenode = modeler.script.stream().createAt("aggregate", "Aggregate", 294, 142)
aggregatenode.setPropertyValue("keys", ["region"])
aggregatenode.setKeyedPropertyValue("aggregates", "income", ["Mean"])

tablenode = modeler.script.stream().createAt("table", "Table", 462, 142)

stream.link(statisticsimportnode, aggregatenode)
stream.link(aggregatenode, tablenode)

selectnode = stream.createAt("select", "Select", 210, 232)
selectnode.setPropertyValue("mode", "Discard")
# Reference the stream parameter in the selection
selectnode.setPropertyValue("condition", "'region' = '$P-LowestRegion'")

typenode = stream.createAt("type", "Type", 366, 232)
typenode.setKeyedPropertyValue("direction", "churn", "Target")

c50node = stream.createAt("c50", "C5.0", 534, 232)

stream.link(statisticsimportnode, selectnode)
stream.link(selectnode, typenode)
stream.link(typenode, c50node)


# First execute the table node
results = []
tablenode.run(results)

# Running the table node should produce a single table as output
table = results[0]

# table output contains a RowSet so we can access values as rows and columns
rowset = table.getRowSet()
min_income = 1000000.0
min_region = None

# From the way the aggregate node is defined, the first column
# contains the region and the second contains the average income
row = 0
rowcount = rowset.getRowCount()
while row < rowcount:
    if rowset.getValueAt(row, 1) < min_income:
        min_income = rowset.getValueAt(row, 1)
        min_region = rowset.getValueAt(row, 0)
    row += 1

# Check that a value was assigned
if min_region != None:
    stream.setParameterValue("LowestRegion", min_region)
else:
    stream.setParameterValue("LowestRegion", -1)
```

```
# Finally run the model builder with the selection criteria
c50node.run([])
```

Globale Werte

Globale Werte werden zum Berechnen verschiedener Auswertungsstatistiken für angegebene Felder verwendet. Diese Auswertungswerte sind überall im Stream zugänglich. Globale Werte sind den Streamparametern darin ähnlich, dass über den Stream anhand des Namens auf sie zugegriffen wird. Sie unterscheiden sich von Streamparametern darin, dass die zugehörigen Werte automatisch bei der Ausführung eines Globalwerteknotens aktualisiert werden und nicht vom Scripting oder von der Befehlszeile zugewiesen werden. Der Zugriff auf die globalen Werte für einen Stream erfolgt über den Aufruf der Methode `getGlobalValues()` des Streams.

Das GlobalValues-Objekt definiert die Funktionen, die in der folgenden Tabelle aufgelistet werden.

Tabelle 22. Vom GlobalValues-Objekt definierte Funktionen

Methode	Rückgabetyp	Beschreibung
`g.fieldNameIterator()`	Iterator	Gibt einen Iterator für jeden Feldnamen mit mindestens einem globalen Wert zurück.
`g.getValue(type, fieldName)`	Object	Gibt den globalen Wert für den angegebenen Typ und Feldnamen zurück bzw. None, wenn kein Wert gefunden werden kann. Als zurückgegebener Wert wird im Allgemeinen eine Zahl erwartet. Einige künftige Funktionen könnten jedoch auch andere Wertetypen zurückgeben.
`g.getValues(fieldName)`	Zuordnung	Gibt eine Zuordnung zurück, die die bekannten Einträge für den angegebenen Feldnamen enthält, bzw. None, wenn es keine vorhandenen Einträge für das Feld gibt.

`GlobalValues.Type` definiert den Typ der verfügbaren Auswertungsstatistiken. Die folgenden Auswertungsstatistikdaten sind verfügbar:

- `MAX`: Maximalwert des Felds.
- `MEAN`: Mittelwert des Felds.
- `MIN`: Minimalwert des Felds.
- `STDDEV`: Standardabweichung des Felds.
- `SUM`: Summe der Werte im Feld.

Das folgende Script z. B. greift auf den Mittelwert des Felds "income" zu, das von einem Globalwerteknoten berechnet wird:

```
import modeler.api

globals = modeler.script.stream().getGlobalValues()
mean_income = globals.getValue(modeler.api.GlobalValues.Type.MEAN, "income")
```

Arbeiten mit mehreren Streams: Standalone-Scripts

Zum Arbeiten mit mehreren Streams muss ein Standalone-Script verwendet werden. Das Standalone-Script kann in der Benutzerschnittstelle von IBM SPSS Modeler bearbeitet und ausgeführt oder als Befehlszeilenparameter im Stapelmodus übergeben werden.

Das folgende Standalone-Script öffnet zwei Streams. Einer dieser Streams erstellt ein Modell, der zweite plottet die Verteilung der vorhergesagten Werte.

```
# Change to the appropriate location for your system
demosDir = "C:/Program Files/IBM/SPSS/Modeler/18.4.0/DEMOS/streams/"

session = modeler.script.session()
tasks = session.getTaskRunner()

# Open the model build stream, locate the C5.0 node and run it
buildstream = tasks.openStreamFromFile(demosDir + "druglearn.str", True)
c50node = buildstream.findByType("c50", None)
results = []
c50node.run(results)

# Now open the plot stream, find the Na_to_K derive and the histogram
plotstream = tasks.openStreamFromFile(demosDir + "drugplot.str", True)
derivenode = plotstream.findByType("derive", None)
histogramnode = plotstream.findByType("histogram", None)

# Create a model applier node, insert it between the derive and histogram nodes
# then run the histgram
applyc50 = plotstream.createModelApplier(results[0], results[0].getName())
applyc50.setPositionBetween(derivenode, histogramnode)
plotstream.linkBetween(applyc50, derivenode, histogramnode)
histogramnode.setPropertyValue("color_field", "$C-Drug")
histogramnode.run([])

# Finally, tidy up the streams
buildstream.close()
plotstream.close()
```

Das folgende Beispiel zeigt, wie Sie auch geöffnete Streams (alle Streams, die auf der Registerkarte Streams geöffnet sind) iterieren können. Beachten Sie, dass dies nur in Standalone-Scripts unterstützt wird.

```
for stream in modeler.script.streams():
    print stream.getName()
```

Kapitel 5. Tipps zum Scripting

In diesem Abschnitt erhalten Sie eine Übersicht der Tipps und Verfahren für die Verwendung von Scripts, wie beispielsweise die Änderung der Streamausführung, die Verwendung von verschlüsselten Kennwörtern in Scripts und den Zugriff auf Objekte in IBM SPSS Collaboration and Deployment Services Repository.

Ändern der Streamausführung

Wenn ein Stream ausgeführt wird, werden die Terminal-Knoten in einer für die Standardsituation optimierten Reihenfolge ausgeführt. In bestimmten Fällen kann eine andere Ausführungsreihenfolge wünschenswert sein. Um die Ausführungsreihenfolge eines Streams zu ändern, führen Sie im Dialogfeld "Streameigenschaften" auf der Registerkarte "Ausführung" folgende Schritte aus:

1. Starten Sie mit einem leeren Script.
2. Klicken Sie in der Symbolleiste auf die Schaltfläche **Standardscript anhängen**, um ein Standard-Stream-Script hinzuzufügen.
3. Bringen Sie die im Standard-Stream-Script enthaltenen Anweisungen in die für die Ausführung gewünschte Reihenfolge.

Verwendung von Schleifen bei Knoten

Sie können eine `for`-Schleife verwenden, um alle Knoten in einem Stream in einer Schleife zu durchlaufen. Die beiden folgenden Scriptbeispiele durchlaufen alle Knoten in einer Schleife und ändern dabei die Feldnamen in allen Filterknoten in Großbuchstaben.

Diese Scripts können in jedem Stream verwendet werden, der einen Filterknoten enthält, selbst wenn tatsächlich gar keine Felder gefiltert werden. Fügen Sie einfach einen Filterknoten hinzu, der alle Felder weitergibt, um die Feldnamen durchgängig in Großbuchstaben zu ändern.

```
# Alternative 1: using the data model nameIterator() function
stream = modeler.script.stream()
for node in stream.iterator():
    if (node.getTypeName() == "filter"):
        # nameIterator() returns the field names
        for field in node.getInputDataModel().nameIterator():
            newname = field.upper()
            node.setKeyedPropertyValue("new_name", field, newname)
```

```
# Alternative 2: using the data model iterator() function
stream = modeler.script.stream()
for node in stream.iterator():
    if (node.getTypeName() == "filter"):
        # iterator() returns the field objects so we need
        # to call getColumnName() to get the name
        for field in node.getInputDataModel().iterator():
            newname = field.getColumnName().upper()
            node.setKeyedPropertyValue("new_name", field.getColumnName(), newname)
```

Das Script durchläuft alle Knoten im aktuellen Stream und prüft jeweils, ob es sich bei den einzelnen Knoten um einen Filter handelt. Wenn ja, durchläuft das Script die einzelnen Felder im Knoten und verwendet die Funktion `field.upper()` oder `field.getColumnName().upper()`, um den Namen in Großbuchstaben zu ändern.

Zugriff auf Objekte in IBM SPSS Collaboration and Deployment Services Repository

Wenn Sie über eine Lizenz für IBM SPSS Collaboration and Deployment Services Repository verfügen, können Sie mithilfe von Scriptbefehlen Objekte im Repository speichern und aus dem Repository abrufen. Mit dem Repository können Sie die Lebensdauer von Data-Mining-Modellen und zugehörigen Vorhersageobjekten im Zusammenhang mit Unternehmensanwendungen, Tools und Lösungen verwalten.

Verbinden mit IBM SPSS Collaboration and Deployment Services Repository

Wenn Sie auf das Repository zugreifen wollen, müssen Sie zunächst entweder über das Menü **Tools** der SPSS Modeler-Benutzerschnittstelle oder über die Befehlszeile eine gültige Verbindung einrichten. Weitere Informationen finden Sie unter „IBM SPSS Collaboration and Deployment Services Repository-Verbindungsargumente” auf Seite 74.

Zugreifen auf das Repository

Auf das Repository kann über die Sitzung zugegriffen werden. Beispiel:

```
repo = modeler.script.session().getRepository()
```

Abrufen von Objekten aus dem Repository

Innerhalb eines Scripts können Sie mit den Funktionen `retrieve*` auf verschiedene Objekte zugreifen, beispielsweise auf Streams, Modelle, Ausgaben und Knoten. Die folgende Tabelle zeigt eine Übersicht über die Abruffunktionen.

Tabelle 23. Abrufen von Scripting-Funktionen

Objekttyp	Repository-Funktion
Stream	repo.retrieveStream(String path, String version, String label, Boolean autoManage)
Model	repo.retrieveModel(String path, String version, String label, Boolean autoManage)
Ausgabe	repo.retrieveDocument(String path, String version, String label, Boolean autoManage)
Knoten	repo.retrieveProcessor(String path, String version, String label, ProcessorDiagram diagram)

Mit der folgenden Funktion können Sie beispielsweise einen Stream aus dem Repository abrufen:

```
stream = repo.retrieveStream("/projects/retention/risk_score.str", None, "production", True)
```

In diesem Beispiel wird der Stream `risk_score.str` aus dem angegebenen Ordner abgerufen. Die Beschriftung `production` gibt an, welche Version des Streams abgerufen werden soll, und der letzte Parameter gibt an, dass der Stream durch SPSS Modeler verwaltet werden soll (damit der Stream beispielsweise auf der Registerkarte **Streams** angezeigt wird, falls die SPSS Modeler-Benutzerschnittstelle sichtbar ist). Alternativer Befehl, wenn eine bestimmte unbeschriftete Version verwendet werden soll:

```
stream = repo.retrieveStream("/projects/retention/risk_score.str", "0:2015-10-12 14:15:41.281",
None, True)
```

Anmerkung: Wenn sowohl der Versions- als auch der Beschriftungsparameter None ist, wird die aktuellste Version zurückgegeben.

Speichern von Objekten im Repository

Verwenden Sie die Funktionen `store*`, um Objekte über ein Script im Repository zu speichern. Die folgende Tabelle zeigt eine Übersicht über die Speicherfunktionen.

Tabelle 24. Speichern von Scripting-Funktionen

Objekttyp	**Repository-Funktion**
Stream	repo.storeStream(ProcessorStream stream, String path, String label)
Model	repo.storeModel(ModelOutput modelOutput, String path, String label)
Ausgabe	repo.storeDocument(DocumentOutput documentOutput, String path, String label)
Knoten	repo.storeProcessor(Processor node, String path, String label)

Mit der folgenden Funktion können Sie beispielsweise eine neue Version des Streams `risk_score.str` speichern:

```
versionId = repo.storeStream(stream, "/projects/retention/risk_score.str", "test")
```

In diesem Beispiel wird eine neue Version des Streams gespeichert, ihr die Beschriftung `"test"` zugeordnet und die Versionsmarkierung für die neu erstellte Version zurückgegeben.

Anmerkung: Wenn Sie der neuen Version keine Beschriftung zuordnen wollen, übergeben Sie None als Beschriftung.

Verwalten von Repository-Ordnern

Durch die Verwendung von Ordnern im Repository können Sie Objekte in logische Gruppen organisieren, sodass zusammengehörige Objekte besser zu erkennen sind. Erstellen Sie mithilfe der Funktion `createFolder()` Ordner, wie im folgenden Beispiel dargestellt:

```
newpath = repo.createFolder("/projects", "cross-sell")
```

In diesem Beispiel wird ein neuer Ordner namens `"cross-sell"` im Ordner `"/projects"` erstellt. Die Funktion gibt den vollständigen Pfad zu dem neuen Ordner zurück.

Mit der Funktion `renameFolder()` können Sie einen Ordner umbenennen:

```
repo.renameFolder("/projects/cross-sell", "cross-sell-Q1")
```

Der erste Parameter ist der vollständige Pfad zum Ordner, der umbenannt werden soll, und der zweite Parameter ist der neue Name für diesen Ordner.

Mit der Funktion `deleteFolder()` können Sie einen leeren Ordner löschen:

```
repo.deleteFolder("/projects/cross-sell")
```

Sperren und Entsperren von Objekten

Mit einem Script können Sie ein Objekt sperren, um zu verhindern, dass andere Benutzer seine bestehenden Versionen aktualisieren oder neue Versionen erstellen. Außerdem können Sie ein Objekt entsperren, das Sie gesperrt haben.

Die Syntax zum Sperren und Entsperren eines Objekts:

```
repo.lockFile(REPOSITORY_PATH)
repo.lockFile(URI)

repo.unlockFile(REPOSITORY_PATH)
repo.unlockFile(URI)
```

Wie beim Speichern und Abrufen von Objekten gibt REPOSITORY_PATH die Position des Objekts im Repository an. Der Pfad muss in Anführungszeichen eingeschlossen sein und es müssen normale Schrägstriche als Trennzeichen verwendet werden. Die Groß- und Kleinschreibung wird nicht berücksichtigt.

```
repo.lockFile("/myfolder/Stream1.str")
repo.unlockFile("/myfolder/Stream1.str")
```

Alternativ können Sie statt eines Repository-Pfads einen URI (Uniform Resource Identifier) verwenden, um die Position des Objekts anzugeben. Der URI muss das Präfix `spsscr:` enthalten und muss vollständig in Anführungszeichen eingeschlossen sein. Nur normale Schrägstriche sind als Pfadtrennzeichen zulässig und Leerzeichen müssen codiert werden. Statt eines Leerzeichens muss im Pfad also `%20` verwendet werden. Die Groß- und Kleinschreibung wird beim URI nicht berücksichtigt. Beispiele:

```
repo.lockFile("spsscr:///myfolder/Stream1.str")
repo.unlockFile("spsscr:///myfolder/Stream1.str")
```

Beachten Sie, dass das Sperren von Objekten für alle Versionen eines Objekts gilt - Sie können keine einzelnen Versionen sperren oder entsperren.

Erstellen eines verschlüsselten Kennworts

In bestimmten Fällen müssen Sie möglicherweise ein Kennwort in ein Script aufnehmen, beispielsweise um auf eine kennwortgeschützte Datenquelle zuzugreifen. Verschlüsselte Kennwörter können in folgenden Elementen verwendet werden:

- Knoteneigenschaften für Datenbankquellenknoten und Ausgabeknoten
- Befehlszeilenargumente für die Anmeldung beim Server
- Die Datenbankverbindungseigenschaften, die in einer *.par*-Datei (die über die Registerkarte "Veröffentlichen" eines Exportknotens generierte Parameterdatei) gespeichert sind.

Über die Benutzerschnittstelle steht ein Tool zur Verfügung, mit dem Sie verschlüsselte Kennwörter auf der Grundlage des Blowfish-Algorithmus erstellen können. (Weitere Informationen finden Sie unter http://www.schneier.com/blowfish.html.) Nach der Verschlüsselung können Sie das Kennwort in Scriptdateien und Befehlszeilenargumente kopieren und dort speichern. Die Knoteneigenschaft `epassword`, die für `databasenode` und `databaseexportnode` verwendet wird, speichert das verschlüsselte Kennwort.

1. Um ein verschlüsseltes Kennwort zu erstellen, wählen Sie im Menü "Extras" Folgendes aus:

   Kennwort verschlüsseln...
2. Geben Sie ein Kennwort im Textfeld "Kennwort" ein.
3. Klicken Sie auf **Verschlüsseln**, um eine Zufallsverschlüsselung des Kennworts zu generieren.
4. Klicken Sie auf die Schaltfläche "Kopieren", um das verschlüsselte Kennwort in die Zwischenablage zu kopieren.
5. Fügen Sie das Kennwort in das gewünschte Script bzw. den gewünschten Parameter ein.

Scriptprüfung

Die Syntax aller Scripttypen können Sie sehr schnell prüfen, indem Sie in der Symbolleiste des Dialogfelds "Standalone-Script" auf die rote Prüfschaltfläche klicken.

Abbildung 6. Symbolleistenschaltflächen für Stream-Scripts

Die Scriptprüfung informiert Sie über alle in Ihrem Code enthaltenen Fehler und macht Verbesserungsvorschläge. Um die den Fehler enthaltende Zeile anzuzeigen, klicken Sie auf das in der unteren Hälfte des Dialogfelds angezeigte Feedback. Der Fehler wird dann rot hervorgehoben.

Scripting über die Befehlszeile

Mit Scripts können Sie Vorgänge ausführen, die normalerweise über die Benutzerschnittstelle durchgeführt werden. Geben Sie in der Befehlszeile beim Start von IBM SPSS Modeler einfach ein Standalone-Script an und führen Sie es aus. Beispiel:

```
client -script scores.txt -execute
```

Das Flag `-script` lädt das angegebene Script, während das Flag `-execute` alle im Script enthaltenen Befehle ausführt.

Kompatibilität mit früheren Releases

In früheren IBM SPSS Modeler-Versionen erstellte Scripts laufen in der aktuellen Version normalerweise unverändert. Allerdings können nun automatisch Modellnuggets in den Stream aufgenommen werden (das ist die Standardeinstellung) und ein vorhandenes Nugget dieses Typs im Stream ersetzen oder ergänzen. Ob dies tatsächlich geschieht, hängt von den Einstellungen der Optionen **Modell zum Stream hinzufügen** und **Vorheriges Modell ersetzen** (**Tools > Optionen > Benutzeroptionen > Benachrichtigungen**) ab. Es kann beispielsweise erforderlich sein, ein Script aus einer früheren Version zu modifizieren, bei der die Nugget-Ersetzung durch Löschen des vorhandenen Nuggets und Einsetzen des neuen erfolgt.

In der aktuellen Version erstellte Scripts funktionieren eventuell nicht in früheren Versionen.

Wenn ein in einer älteren Version erstelltes Script einen Befehl verwendet, der mittlerweile ersetzt wurde (oder nicht mehr verwendet wird), wird die alte Form weiterhin unterstützt, es wird jedoch eine Warnnachricht angezeigt. Beispielsweise wurde das alte Schlüsselwort `generated` durch `model` und `clear generated` wurde durch `clear generated palette` ersetzt. Scripts, die die alten Formen verwenden, werden weiterhin ausgeführt, es wird jedoch eine Warnnachricht angezeigt.

Zugriff auf Streamausführungsergebnisse

Viele IBM SPSS Modeler-Knoten erzeugen Ausgabeobjekte, wie z. B. Modelle, Diagramme und Tabellendaten. Viele dieser Ausgabedaten enthalten nützliche Werte, mit denen Scripts die nachfolgende Ausführung steuern können. Diese Werte werden in Containern mit Inhalt (einfach als "Container" bezeichnet) gruppiert, auf die über Tags oder IDs zugegriffen werden kann, die die einzelnen Container bezeichnen. Wie auf diese Werte zugegriffen wird, hängt vom Format oder "Inhaltsmodell" ab, das der jeweilige Container verwendet.

Viele Ausgaben von Vorhersagemodellen verwenden z. B. eine Variante von XML mit der Bezeichnung PMML zum Darstellen von Informationen zum Modell, wie z. B., welche Felder ein Entscheidungsbaum an jeder Aufteilung verwendet oder wie und mit welcher Stärke die Neuronen in einem neuronalen Netz verbunden sind. Modellausgaben, die PMML verwenden, stellen ein XML-Inhaltsmodell bereit, über das auf diese Informationen zugegriffen werden kann. Beispiel:

```
stream = modeler.script.stream()
# Assume the stream contains a single C5.0 model builder node
# and that the datasource, predictors and targets have already been
# set up
modelbuilder = stream.findByType("c50", None)
results = []
modelbuilder.run(results)
modeloutput = results[0]

# check how many contents are there and what are their names
tags = modeloutput.getContentModelTags()
print "Content Model Tags :" , tags

# Now that we have the C5.0 model output object, access the
# relevant content model
cm = modeloutput.getContentModel("PMML")
if (cm != None) :
```

```
    # The PMML content model is a generic XML-based content model that
    # uses XPath syntax. Use that to find the names of the data fields.
    # The call returns a list of strings match the XPath values
    dataFieldNames = cm.getStringValues("/PMML/DataDictionary/DataField",
"name")
    print "Data Field Names:" , dataFieldNames
```

IBM SPSS Modeler unterstützt die folgenden Inhaltsmodelle beim Scripting:

- Das **Tabelleninhaltsmodell** ermöglicht Zugriff auf die einfachen Tabellendaten, die als Zeilen und Spalten dargestellt werden.
- Das **XML-Inhaltsmodell** ermöglicht Zugriff auf Inhalt im XML-Format.
- Das **JSON-Inhaltsmodell** ermöglicht Zugriff auf Inhalt im JSON-Format.
- Das **Inhaltsmodell für Spaltenstatistikdaten** ermöglicht Zugriff auf Auswertungsstatistikdaten zu einem bestimmten Feld.
- Das **Inhaltsmodell für paarweise Spaltenstatistikdaten** ermöglicht Zugriff auf Auswertungsstatistikdaten zwischen zwei Feldern oder Werte zwischen zwei separaten Feldern.

Beachten Sie, dass die folgenden Knoten diese Inhaltsmodelle nicht enthalten:

- Zeitreihe
- Diskriminanz
- SLRM
- TCM
- Alle Python-Knoten
- Alle Spark-Knoten
- Alle Datenbankmodellierungsknoten
- Erweiterungsmodell
- STP

Tabelleninhaltsmodell

Das Tabelleninhaltsmodell stellt ein einfaches Modell für den Zugriff auf einfache Zeilen- und Spaltendaten bereit. Die Werte in einer bestimmten Spalte müssen alle denselben Speichertyp haben (z. B. Zeichenfolgen oder Ganzzahlen).

API

Tabelle 25. API

Zurück	Methode	Beschreibung
`int`	`getRowCount()`	Gibt die Anzahl der Zeilen in dieser Tabelle zurück.
`int`	`getColumnCount()`	Gibt die Anzahl der Spalten in dieser Tabelle zurück.
`String`	`getColumnName(int columnIndex)`	Gibt den Namen der Spalte am angegebenen Spaltenindex zurück. Der Spaltenindex beginnt bei 0.
`StorageType`	`getStorageType(int columnIndex)`	Gibt den Speichertyp der Spalte am angegebenen Index zurück. Der Spaltenindex beginnt bei 0.

Tabelle 25. API (Forts.)		
Zurück	**Methode**	**Beschreibung**
`Object`	`getValueAt(int rowIndex, int columnIndex)`	Gibt den Wert am angegebenen Zeilen- und Spaltenindex zurück. Der Zeilenindex und der Spaltenindex beginnen bei 0.
`void`	`reset()`	Führt eine Flushoperation für den internen Speicher aus, der diesem Inhaltsmodell zugeordnet ist.

Knoten und Ausgaben

In dieser Tabelle sind Knoten aufgelistet, die die Ausgaben erstellen, die diesen Typ von Inhaltsmodell enthalten.

Tabelle 26. Knoten und Ausgaben		
Knotenname	**Ausgabename**	**Container-ID**
`table`	`table`	`"table"`

Beispielscript

```
stream = modeler.script.stream()
from modeler.api import StorageType

# Set up the variable file import node
varfilenode = stream.createAt("variablefile", "DRUG Data", 96, 96)
varfilenode.setPropertyValue("full_filename", "$CLEO_DEMOS/DRUG1n")

# Next create the aggregate node and connect it to the variable file node
aggregatenode = stream.createAt("aggregate", "Aggregate", 192, 96)
stream.link(varfilenode, aggregatenode)

# Configure the aggregate node
aggregatenode.setPropertyValue("keys", ["Drug"])
aggregatenode.setKeyedPropertyValue("aggregates", "Age", ["Min", "Max"])
aggregatenode.setKeyedPropertyValue("aggregates", "Na", ["Mean", "SDev"])

# Then create the table output node and connect it to the aggregate node
tablenode = stream.createAt("table", "Table", 288, 96)
stream.link(aggregatenode, tablenode)

# Execute the table node and capture the resulting table output object
results = []
tablenode.run(results)
tableoutput = results[0]

# Access the table output's content model
tablecontent = tableoutput.getContentModel("table")

# For each column, print column name, type and the first row
# of values from the table content
col = 0
while col < tablecontent.getColumnCount():
     print tablecontent.getColumnName(col), \
     tablecontent.getStorageType(col), \
```

```
    tablecontent.getValueAt(0, col)
    col = col + 1
```

Die Ausgabe auf der Registerkarte für die Scripting-Fehlerbehebung sieht ungefähr wie folgt aus:

```
Age_Min Integer 15
Age_Max Integer 74
Na_Mean Real 0.730851098901
Na_SDev Real 0.116669731242
Drug String drugY
Record_Count Integer 91
```

XML-Inhaltsmodell

Das XML-Inhaltsmodell bietet Zugriff auf XML-basierten Inhalt.

Das XML-Inhaltsmodell unterstützt den Zugriff auf Komponenten auf der Basis von XPath-Ausdrücken. XPath-Ausdrücke sind Zeichenfolgen, die definieren, welche Elemente oder Attribute vom Aufrufenden benötigt werden. Das XML-Inhaltsmodell blendet die Details der Erstellung verschiedener Objekte und der Kompilierung von Ausdrücken, die normalerweise für die XPath-Unterstützung benötigt werden, aus. Dies erleichtert das Aufrufen aus Python-Scriptumgebungen.

Das XML-Inhaltsmodell enthält eine Funktion, die das XML-Dokument als Zeichenfolge zurückgibt. Dadurch können Python-Scriptbenutzer ihre bevorzugte Python-Bibliothek zum Analysieren des XML-Codes verwenden.

API

Tabelle 27. API

Zurück	Methode	Beschreibung
`String`	`getXMLAsString()`	Gibt die XML-Daten als Zeichenfolge zurück.
`number`	`getNumericValue(String xpath)`	Gibt das Ergebnis der Auswertung des Pfads mit dem numerischen Datentyp zurück (z. B. Zählen der Anzahl Elemente, die mit dem Pfadausdruck übereinstimmen).
`boolean`	`getBooleanValue(String xpath)`	Gibt das boolesche Ergebnis der Auswertung des angegebenen Pfadausdrucks zurück.
`String`	`getStringValue(String xpath, String attribute)`	Gibt entweder den Attributwert oder den XML-Knotenwert zurück, der mit dem angegebenen Pfad übereinstimmt.
`List of strings`	`getStringValues(String xpath, String attribute)`	Gibt eine Liste aller Attributwerte oder XML-Knotenwerte zurück, die mit dem angegebenen Pfad übereinstimmen.
`List of lists of strings`	`getValuesList(String xpath, <List of strings> attributes, boolean includeValue)`	Gibt eine Liste aller Attributwerte, die mit dem angegebenen Pfad übereinstimmen, zusammen mit dem XML-Knotenwert (falls erforderlich) zurück.

Tabelle 27. API (Forts.)		
Zurück	**Methode**	**Beschreibung**
`Hash table (key:string, value:list of string)`	`getValuesMap(String xpath, String keyAttribute, <List of strings> attributes, boolean includeValue)`	Gibt eine Hashtabelle zurück, die das Schlüsselattribut oder den XML-Knotenwert als Schlüssel und die Liste angegebener Attributwerte als Tabellenwerte verwendet.
`boolean`	`isNamespaceAware()`	Gibt zurück, ob die XML-Parser Namespaces erkennen sollen. Der Standardwert ist `False`.
`void`	`setNamespaceAware(boolean value)`	Legt fest, ob die XML-Parser Namespaces erkennen sollen. Hierdurch wird auch `reset()` aufgerufen, um sicherzustellen, dass Änderungen von nachfolgenden Aufrufen berücksichtigt werden.
`void`	`reset()`	Führt eine Flushoperation für den internen Speicher aus, der diesem Inhaltsmodell zugeordnet ist (beispielsweise im Cache gespeichertes DOM-Objekt).

Knoten und Ausgaben

In dieser Tabelle sind Knoten aufgelistet, die die Ausgaben erstellen, die diesen Typ von Inhaltsmodell enthalten.

Tabelle 28. Knoten und Ausgaben		
Knotenname	**Ausgabename**	**Container-ID**
`Most model builders`	`Most generated models`	`"PMML"`
`"autodataprep"`	`n/a`	`"PMML"`

Beispielscript

Der Python-Scriptcode für den Zugriff auf den Inhalt könnte wie folgt aussehen:

```
results = []
modelbuilder.run(results)
modeloutput = results[0]
cm = modeloutput.getContentModel("PMML")

dataFieldNames = cm.getStringValues("/PMML/DataDictionary/DataField", "name")
predictedNames = cm.getStringValues("//MiningSchema/MiningField[@usage
Type='predicted']", "name")
```

JSON-Inhaltsmodell

Das JSON-Inhaltsmodell bietet Unterstützung für Inhalt im JSON-Format. Es stellt eine Basis-API zur Verfügung, die Aufrufenden die Extraktion von Werten ermöglicht. Dabei wird angenommen, dass sie wissen, auf welche Werte zugegriffen werden soll.

API

Tabelle 29. API

Zurück	Methode	Beschreibung
`String`	`getJSONAsString()`	Gibt den JSON-Inhalt als Zeichenfolge zurück.
`Object`	`getObjectAt(<List of cbjecta> path, JSONArtifact artifact) throws Exception`	Gibt das Objekt im angegebenen Pfad zurück. Das angegebene Stammartefakt kann null sein. In diesem Fall wird der Stamm des Inhalts verwendet. Der zurückgegebene Wert kann eine Literalzeichenfolge, eine Ganzzahl, eine reelle Zahl oder ein boolescher Wert oder ein JSON-Artefakt (ein JSON-Objekt oder ein JSON-Array) sein.
`Hash table (key:object, value:object>`	`getChildValuesAt(<List of object> path, JSONArtifact artifact) throws Exception`	Gibt die untergeordneten Werte des angegebenen Pfads zurück, wenn der Pfad zu einem JSON-Objekt führt, oder andernfalls null. Die Schlüssel in der Tabelle sind Zeichenfolgen, während der zugeordnete Wert eine Literalzeichenfolge, eine Ganzzahl, eine reelle Zahl oder ein boolescher Wert oder ein JSON-Artefakt (ein JSON-Objekt oder ein JSON-Array) sein kann.
`List of objects`	`getChildrenAt(<List of object> path path, JSONArtifact artifact) throws Exception`	Gibt die Liste von Objekten im angegebenen Pfad zurück, wenn der Pfad zu einem JSON-Array führt, oder andernfalls null. Die zurückgegebenen Werte können eine Literalzeichenfolge, eine Ganzzahl, eine reelle Zahl oder ein boolescher Wert oder ein JSON-Artefakt (ein JSON-Objekt oder ein JSON-Array) sein.
`void`	`reset()`	Führt eine Flushoperation für den internen Speicher aus, der diesem Inhaltsmodell zugeordnet ist (beispielsweise im Cache gespeichertes DOM-Objekt).

Knoten und Ausgaben

In dieser Tabelle sind Knoten aufgelistet, die die Ausgaben erstellen, die diesen Typ von Inhaltsmodell enthalten.

Tabelle 30. Knoten und Ausgaben

Knotenname	Ausgabename	Container-ID
`"applykmeansas"`	`n/a`	`"JSON_MV"`

Tabelle 30. Knoten und Ausgaben (Forts.)		
Knotenname	**Ausgabename**	**Container-ID**
"applyxgboosttree"	n/a	"JSON_MV"

Beispielscripts

Die folgenden Scripts rufen JSON-Dateien ab:

```
applykmeansas = stream.findByType("applykmeansas",None)
mvjson = applykmeansas.getContentModel("JSON_MV")
print(mvjson.getJSONAsString())
```

```
applyxgboosttree = stream.findByType("applyxgboosttree",None)
mvjson = applyxgboosttree.getContentModel("JSON_MV")
print(mvjson.getJSONAsString())
```

Inhaltsmodell für Spaltenstatistiken und Inhaltsmodell für paarweise Statistikdaten

Das Inhaltsmodell für Spaltenstatistikdaten ermöglicht Zugriff auf Statistikdaten, die für jedes Feld berechnet werden können (univariate Statistik). Das Inhaltsmodell für paarweise Statistikdaten ermöglicht Zugriff auf Statistikdaten, die zwischen Paaren von Feldern oder Werten in einem Feld berechnet werden können.

Folgende Statistikdatenmaße sind möglich:

- Count
- UniqueCount
- ValidCount
- Mean
- Sum
- Min
- Max
- Range
- Variance
- StandardDeviation
- StandardErrorOfMean
- Skewness
- SkewnessStandardError
- Kurtosis
- KurtosisStandardError
- Median
- Mode
- Pearson
- Covariance
- TTest
- FTest

Einige Werte sind nur für Einzelspaltenstatistikdaten geeignet, andere nur für paarweise Statistikdaten.

Knoten, durch die sie erzeugt werden:

- Der **Statistikknoten** erzeugt Spaltenstatistikdaten und kann paarweise Statistikdaten erzeugen, wenn Korrelationsfelder angegeben werden.
- Der **Data Audit-Knoten** erzeugt Spaltenstatistikdaten und kann paarweise Statistikdaten erzeugen, wenn ein Überlagerungsfeld angegeben wird.
- Der **Mittelwertknoten** erzeugt paarweise Statistikdaten, wenn Feldpaare verglichen werden oder wenn die Werte eines Felds mit anderen Feldzusammenfassungen verglichen werden.

Welche Inhaltsmodelle und Statistikdaten verfügbar sind, hängt von den Fähigkeiten des jeweiligen Knotens und den Einstellungen im Knoten ab.

ColumnStatsContentModel-API

Tabelle 31. ColumnStatsContentModel-API

Zurück	Methode	Beschreibung
`List<StatisticType>`	`getAvailableStatistics()`	Gibt die verfügbaren Statistikdaten in diesem Modell zurück. Nicht alle Felder enthalten notwendigerweise Werte für alle Statistikdaten.
`List<String>`	`getAvailableColumns()`	Gibt die Namen der Spalten zurück, für die Statistikdaten berechnet wurden.
`Number`	`getStatistic(String column, StatisticType statistic)`	Gibt die statistischen Werte zurück, die der Spalte zugeordnet sind.
`void`	`reset()`	Führt eine Flushoperation für den internen Speicher aus, der diesem Inhaltsmodell zugeordnet ist.

PairwiseStatsContentModel-API

Tabelle 32. PairwiseStatsContentModel-API

Zurück	Methode	Beschreibung
`List<StatisticType>`	`getAvailableStatistics()`	Gibt die verfügbaren Statistikdaten in diesem Modell zurück. Nicht alle Felder enthalten notwendigerweise Werte für alle Statistikdaten.
`List<String>`	`getAvailablePrimaryColumns()`	Gibt die Namen der Primärspalten zurück, für die Statistikdaten berechnet wurden.
`List<Object>`	`getAvailablePrimaryValues()`	Gibt die Werte der Primärspalte zurück, für die Statistikdaten berechnet wurden.
`List<String>`	`getAvailableSecondaryColumns()`	Gibt die Namen der Sekundärspalten zurück, für die Statistikdaten berechnet wurden.

Tabelle 32. PairwiseStatsContentModel-API (Forts.)		
Zurück	**Methode**	**Beschreibung**
`Number`	`getStatistic(String primaryColumn, String secondaryColumn, StatisticType statistic)`	Gibt die statistischen Werte zurück, die den Spalten zugeordnet sind.
`Number`	`getStatistic(String primaryColumn, Object primaryValue, String secondaryColumn, StatisticType statistic)`	Gibt die statistischen Werte zurück, die dem Primärspaltenwert und der Sekundärspalte zugeordnet sind.
`void`	`reset()`	Führt eine Flushoperation für den internen Speicher aus, der diesem Inhaltsmodell zugeordnet ist.

Knoten und Ausgaben

In dieser Tabelle sind Knoten aufgelistet, die die Ausgaben erstellen, die diesen Typ von Inhaltsmodell enthalten.

Tabelle 33. Knoten und Ausgaben			
Knotenname	**Ausgabename**	**Container-ID**	**Hinweise**
`"means"` (Mittelwertknoten)	`"means"`	`"columnStatistics"`	
`"means"` (Mittelwertknoten)	`"means"`	`"pairwiseStatistics"`	
`"dataaudit"` (Data Audit-Knoten)	`"means"`	`"columnStatistics"`	
`"statistics"` (Statistikknoten)	`"statistics"`	`"columnStatistics"`	Wird nur generiert, wenn bestimmte Felder untersucht werden.
`"statistics"` (Statistikknoten)	`"statistics"`	`"pairwiseStatistics"`	Wird nur generiert, wenn Felder korreliert werden.

Beispielscript

```
from modeler.api import StatisticType
stream = modeler.script.stream()

# Set up the input data
varfile = stream.createAt("variablefile", "File", 96, 96)
varfile.setPropertyValue("full_filename", "$CLEO/DEMOS/DRUG1n")

# Now create the statistics node. This can produce both
# column statistics and pairwise statistics
```

```
statisticsnode = stream.createAt("statistics", "Stats", 192, 96)
statisticsnode.setPropertyValue("examine", ["Age", "Na", "K"])
statisticsnode.setPropertyValue("correlate", ["Age", "Na", "K"])
stream.link(varfile, statisticsnode)

results = []
statisticsnode.run(results)
statsoutput = results[0]
statscm = statsoutput.getContentModel("columnStatistics")
if (statscm != None):
    cols = statscm.getAvailableColumns()
    stats = statscm.getAvailableStatistics()
    print "Column stats:", cols[0], str(stats[0]), " = ", statscm.getStatis
tic(cols[0], stats[0])

statscm = statsoutput.getContentModel("pairwiseStatistics")
if (statscm != None):
    pcols = statscm.getAvailablePrimaryColumns()
    scols = statscm.getAvailableSecondaryColumns()
    stats = statscm.getAvailableStatistics()
    corr = statscm.getStatistic(pcols[0], scols[0], StatisticType.Pearson)
    print "Pairwise stats:", pcols[0], scols[0], " Pearson = ", corr
```

Kapitel 6. Befehlszeilenargumente

Aufrufen der Software

Sie können IBM SPSS Modeler wie folgt über die Befehlszeile Ihres Betriebssystems starten:

1. Öffnen Sie auf einem Computer, auf dem IBM SPSS Modeler installiert ist, ein DOS- oder Befehlszeilenfenster.
2. Geben Sie den Befehl `modelerclient` gefolgt von den erforderlichen Argumenten ein, um die IBM SPSS Modeler-Schnittstelle im interaktiven Modus zu starten. Beispiel:

```
modelerclient -stream report.str -execute
```

Mithilfe der verfügbaren Argumente (Flags) können Sie eine Verbindung zu einem Server herstellen, Streams laden, Scripts ausführen oder je nach Bedarf weitere Parameter angeben.

Verwenden von Befehlszeilenargumenten

Sie können Befehlszeilenargumente (auch als *Flags* bezeichnet) an den ursprünglichen Befehl `modelerclient` anhängen, um die Vorgehensweise beim Aufrufen von IBM SPSS Modeler zu ändern.

Mehrere Typen von Befehlszeilenargumenten sind verfügbar und werden weiter unten in diesem Abschnitt beschrieben.

Tabelle 34. Typen von Befehlszeilenargumenten

Argumenttyp	**Weitere Informationen**
Systemargumente	Weitere Informationen finden Sie im Thema „Systemargumente” auf Seite 70.
Parameterargumente	Weitere Informationen finden Sie im Thema „Parameterargumente” auf Seite 71.
Argumente zum Herstellen einer Serververbindung	Weitere Informationen finden Sie im Thema „Argumente zum Herstellen einer Serververbindung” auf Seite 72.
Argumente zum Herstellen einer Verbindung zu IBM SPSS Collaboration and Deployment Services Repository	Weitere Informationen finden Sie im Thema „IBM SPSS Collaboration and Deployment Services Repository-Verbindungsargumente” auf Seite 74.
Argumente zum Herstellen einer Verbindung zu IBM SPSS Analytic Server	Weitere Informationen finden Sie im Thema „Argumente zum Herstellen einer Verbindung zu IBM SPSS Analytic Server” auf Seite 74.

Beispielsweise können Sie mit den Flags `-server`, `-stream` und `-execute` wie folgt eine Verbindung zu einem Server herstellen und dann einen Stream laden und ausführen:

```
modelerclient -server -hostname myserver -port 80 -username dminer
-password 1234 -stream mystream.str -execute
```

Beachten Sie: Bei der Ausführung unter einer lokalen Client-Installation sind die Argumente für die Serververbindung nicht erforderlich.

Parameterwerte, die Leerzeichen enthalten, können in doppelte Anführungszeichen eingeschlossen werden. Beispiel:

```
modelerclient -stream mystream.str -Pusername="Joe User" -execute
```

Sie können auch IBM SPSS Modeler-Status und -Scripts auf diese Weise ausführen, indem Sie die Flags `-state` bzw. `-script` verwenden.

Anmerkung: Wenn Sie einen strukturierten Parameter in einem Befehl verwenden, müssen Sie vor Anführungszeichen einen umgekehrten Schrägstrich (\) angeben. Dadurch wird verhindert, dass die Anführungszeichen während der Interpretation der Zeichenfolge entfernt werden.

Befehlszeilenargumente für Debugging

Verwenden Sie zum Debuggen einer Befehlszeile den Befehl `modelerclient`, um IBM SPSS Modeler mit den gewünschten Argumenten zu starten. Damit können Sie prüfen, ob die Befehle erwartungsgemäß ausgeführt werden. Außerdem können Sie die Werte jedes Parameters bestätigen, der von der Befehlszeile in das Dialogfeld "Sitzungsparameter" (Menü "Extras", "Sitzungsparameter festlegen") übergeben wird.

Systemargumente

In der nachstehenden Tabelle werden die Systemargumente beschrieben, die für das Aufrufen der Benutzerschnittstelle über die Befehlszeile zur Verfügung stehen.

Tabelle 35. Systemargumente

Argument	Verhalten/Beschreibung
`@` <commandFile>	Das Symbol @, gefolgt von einem Dateinamen, bezeichnet eine Liste von Befehlen. Wenn der Befehl `modelerclient` auf ein Argument mit dem Symbol @ trifft, werden die Befehle in dieser Datei so abgearbeitet, als hätten Sie diese Befehle direkt in der Befehlszeile eingegeben. Weitere Informationen finden Sie im Thema „Kombinieren mehrerer Argumente" auf Seite 75.
`-directory` <dir>	Bestimmt das Standardarbeitsverzeichnis. Im lokalen Modus wird dieses Verzeichnis sowohl für Daten als auch für die Ausgabe herangezogen. Beispiel: `-directory c:/` oder `-directory c:\\`
`-server_directory` <dir>	Bestimmt das Serverstandardverzeichnis für Daten. Das Arbeitsverzeichnis, das mithilfe des Flags `-directory` angegeben wird, wird für die Ausgabe genutzt.
`-execute`	Nach dem Starten: Alle Streams, Statusangaben oder Scripts ausführen, die beim Starten geladen waren. Wird ein Script zusätzlich zu einem Stream oder einem Status geladen, wird nur das Script ausgeführt.
`-stream` <stream>	Beim Starten: Angegebenen Stream laden. Sie können mehrere Streams angeben; der zuletzt genannte Stream wird dabei als aktueller Stream festgelegt.
`-script` <script>	Beim Starten: Angegebenes Standalone-Script laden. Sie können dieses Script zusätzlich zu einem Stream oder einem Status angeben (siehe unten); beim Starten kann jedoch nur ein einziges Script geladen werden.
`-model` <model>	Beim Starten: Angegebenes generiertes Modell (Datei im Format `.gm`) laden.
`-state` <state>	Beim Starten: Angegebenen gespeicherten Status laden.
`-project` <project>	Angegebenes Projekt laden. Beim Starten kann nur ein einziges Projekt geladen werden.
`-output` <output>	Beim Starten: Gespeichertes Ausgabeobjekt (Datei im Format `.cou`) laden.

Tabelle 35. Systemargumente (Forts.)	
Argument	**Verhalten/Beschreibung**
`-help`	Liste der Befehlszeilenargumente abrufen. Wenn diese Option angegeben ist, werden alle anderen Argumente ignoriert und der Hilfebildschirm wird geöffnet.
`-P <name>=<value>`	Bestimmt einen Startparameter. Kann auch zum Festlegen von Knoteneigenschaften (Slotparameter) herangezogen werden.

Anmerkung: Standardverzeichnisse können auch über die Benutzerschnittstelle festgelegt werden. Wählen Sie hierzu im Menü "Datei" die Option **Arbeitsverzeichnis festlegen** bzw. **Serververzeichnis festlegen** aus.

Laden mehrerer Dateien

Über die Befehlszeile können Sie beim Start mehrere Streams, Status und Ausgaben laden, indem Sie für jedes geladene Objekt das relevante Argument wiederholen. Sollen beispielsweise zwei Streams mit den Bezeichnungen `report.str` und `train.str` geladen und ausgeführt werden, geben Sie den folgenden Befehl ein:

```
modelerclient -stream report.str -stream train.str -execute
```

Laden von Objekten aus IBM SPSS Collaboration and Deployment Services Repository

Da Sie bestimmte Objekte aus einer Datei oder aus IBM SPSS Collaboration and Deployment Services Repository (falls lizenziert) laden können, gibt das Dateinamenspräfix `spsscr:` und optional `file:` (für Objekte auf Platte) an, wo IBM SPSS Modeler nach dem Objekt suchen soll. Das Präfix funktioniert mit folgenden Flags:

- `-stream`
- `-script`
- `-output`
- `-model`
- `-project`

Das Präfix wird zur Erstellung eines URI verwendet, der die Position des Objekts angibt. Beispiel: `-stream "spsscr:///folder_1/scoring_stream.str"`. Das Vorhandensein des Präfix `spsscr:` erfordert, dass eine gültige Verbindung zu IBM SPSS Collaboration and Deployment Services Repository in demselben Befehl angegeben wurde. Der vollständige Befehl sieht also etwa wie folgt aus:

```
modelerclient -spsscr_hostname myhost -spsscr_port 8080
-spsscr_username myusername -spsscr_password mypassword
-stream "spsscr:///folder_1/scoring_stream.str" -execute
```

Beachten Sie: In der Befehlszeile *müssen* Sie einen URI verwenden. Das einfachere REPOSITORY_PATH wird nicht unterstützt. (Es funktioniert nur innerhalb von Scripts.) Weitere Details zu URIs für Objekte in IBM SPSS Collaboration and Deployment Services Repository finden Sie in „Zugriff auf Objekte in IBM SPSS Collaboration and Deployment Services Repository” auf Seite 56.

Parameterargumente

Bei der Ausführung von IBM SPSS Modeler über die Befehlszeile können Parameter als Flags herangezogen werden. In Befehlszeilenargumenten wird das Flag `-P` verwendet, um einen Parameter im Format `-P` <*Name*>=<*Wert*> anzugeben.

Die folgenden Parameter stehen zur Auswahl:

- **Einfache Parameter** (oder Parameter, die direkt in CLEM-Ausdrücken verwendet werden).
- **Slot-Parameter** (auch als Knoteneigenschaften bezeichnet). Mit diesen Parametern werden die Einstellungen für die Knoten im Stream bearbeitet. Weitere Informationen finden Sie im Thema „Knoteneigenschaften - Übersicht” auf Seite 79.
- **Befehlszeilenparameter** dienen zum Ändern der Vorgehensweise beim Aufrufen von IBM SPSS Modeler.

Geben Sie beispielsweise die Benutzernamen und Kennwörter für Datenquellen in Form von Befehlszeilenflags an:

```
modelerclient -stream response.str -P:databasenode.datasource="{\"ORA
10gR2\",user1,mypsw,false}"
```

Das Format stimmt mit dem Parameter `datasource` der Knoteneigenschaft `databasenode` überein. Weitere Informationen finden Sie in „Eigenschaften von "databasenode"” auf Seite 97.

Der letzte Parameter muss auf `true` gesetzt werden, wenn Sie ein verschlüsseltes Kennwort übergeben. Beachten Sie auch, dass vor dem Namen und dem Kennwort für den Datenbankbenutzer keine führende Leerzeichen verwendet sollten (sofern Ihr Benutzername oder Kennwort nicht tatsächlich mit einem führenden Leerzeichen beginnt).

Anmerkung: Wenn der Knoten benannt ist, müssen Sie seinen Namen in Anführungszeichen einschließen und den Anführungszeichen einen umgekehrten Schrägstrich (\) als Escapezeichen voranstellen. Wenn beispielsweise der Datenquellenknoten im vorherigen Beispiel den Namen `Source_ABC` hat, würde der Eintrag wie folgt lauten:

```
modelerclient -stream response.str -P:databasenode.\"Source_ABC\".datasour
ce="{\"ORA 10gR2\",
 user1,mypsw,true}"
```

Ein umgekehrter Schrägstrich ist auch vor Anführungszeichen erforderlich, die einen strukturierten Parameter angeben. Siehe das folgende Beispiel für eine TM1-Datenquelle:

```
clemb -server -hostname 9.115.21.169 -port 28053 -username administrator
    -execute -stream C:\Share\TM1_Script.str -P:tm1import.pm_host="http://9.115.21.163:9510/
pmhub/pm"
    -P:tm1import.tm1_connection={\"SData\",\"\",\"admin\",\"apple\"}
    -P:tm1import.selected_view={\"SalesPriorCube\",\"salesmargin%\"}
```

Anmerkung: Wenn der Datenbankname (in der Eigenschaft `datasource` mindestens ein Leerzeichen, mindestens einen Punkt oder mindestens einen Unterstrich enthält, können Sie ihn in Zeichenfolgen aus Backslash und doppeltem Anführungszeichen einschließen, damit er als Zeichenfolge behandelt wird. Beispiele: `"{\"db2v9.7.6_linux\"}"` oder `"{\"TDATA 131\"}"`. Schließen Sie außerdem `datasource`-Zeichenfolgewerte immer in Anführungszeichen und geschweifte Klammern ein, wie in dem folgenden Beispiel gezeigt: `"{\"SQL Server\",spssuser,abcd1234,false}"`.

Argumente zum Herstellen einer Serververbindung

Das Flag `-server` besagt, dass IBM SPSS Modeler eine Verbindung zu einem öffentlichen Server herstellen soll. Mit den Flags `-hostname`, `-use_ssl`, `-port`, `-username` `-password` und `-domain` legen Sie fest, auf welche Weise IBM SPSS Modeler die Verbindung zum öffentlichen Server herstellen soll. Wenn kein Argument vom Typ -server angegeben wurde, wird der Standardserver bzw. der lokale Server verwendet.

Beispiele

So stellen Sie eine Verbindung zu einem öffentlichen Server her:

```
modelerclient -server -hostname myserver -port 80 -username dminer
-password 1234 -stream mystream.str -execute
```

So stellen Sie eine Verbindung zu einem Server-Cluster her:

```
modelerclient -server -cluster "QA Machines" \
-spsscr_hostname pes_host -spsscr_port 8080 \
-spsscr_username asmith -spsscr_epassword xyz
```

Beachten Sie, dass zum Herstellen einer Verbindung zu einem Server-Cluster der Coordinator of Processes über IBM SPSS Collaboration and Deployment Services erforderlich ist. Daher muss das Argument `-cluster` in Kombination mit den Optionen für eine Repository-Verbindung (`spsscr_*`) verwendet werden. Weitere Informationen finden Sie in „IBM SPSS Collaboration and Deployment Services Repository-Verbindungsargumente" auf Seite 74.

Tabelle 36. Argumente zum Herstellen einer Serververbindung

Argument	**Verhalten/Beschreibung**
`-server`	Startet IBM SPSS Modeler im Servermodus. Hierzu wird eine Verbindung zu einem öffentlichen Server mit den Flags `-hostname`, `-port`, `-username`, `-password` und `-domain` hergestellt.
`-hostname <name>`	Hostname des Server-Computers. Nur im Servermodus verfügbar.
`-use_ssl`	Gibt an, dass die Verbindung SSL (Secure Socket Layer) verwenden sollte. Dieses Flag ist optional, bei der Standardeinstellung wird SSL *nicht* verwendet.
`-port <number>`	Portnummer des angegebenen Servers. Nur im Servermodus verfügbar.
`-cluster <name>`	Gibt eine Verbindung zu einem Server-Cluster und nicht zu einem benannten Server an; dieses Argument ist eine Alternative zu den Argumenten `hostname`, `port` und `use_ssl`. Bei dem Namen handelt es sich um den Clusternamen oder um einen eindeutigen URI, der den Cluster in der IBM SPSS Collaboration and Deployment Services Repository-Instanz identifiziert. Der Server-Cluster wird von Coordinator of Processes über IBM SPSS Collaboration and Deployment Services verwaltet. Weitere Informationen finden Sie im Thema „IBM SPSS Collaboration and Deployment Services Repository-Verbindungsargumente" auf Seite 74.
`-username <name>`	Benutzername, mit dem die Anmeldung beim Server erfolgt. Nur im Servermodus verfügbar.
`-password <password>`	Kennwort, mit dem die Anmeldung beim Server erfolgt. Nur im Servermodus verfügbar. **Anmerkung:** Falls das Argument `-password` nicht vorliegt, werden Sie zur Angabe eines Kennworts aufgefordert.
`-epassword <encoded-passwordstring>`	Codiertes Kennwort, mit dem die Anmeldung beim Server erfolgt. Nur im Servermodus verfügbar. **Anmerkung:** Ein verschlüsseltes Kennwort kann in IBM SPSS Modeler mit den Befehlen im Menü "Tools" generiert werden.
`-domain <name>`	Domäne, mit der die Anmeldung beim Server erfolgt. Nur im Servermodus verfügbar.
`-P <name>=<value>`	Bestimmt einen Startparameter. Kann auch zum Festlegen von Knoteneigenschaften (Slotparameter) herangezogen werden.

IBM SPSS Collaboration and Deployment Services Repository-Verbindungsargumente

Wenn Sie Objekte aus IBM SPSS Collaboration and Deployment Services mithilfe der Befehlszeile speichern oder abrufen möchten, müssen Sie eine gültige Verbindung zum IBM SPSS Collaboration and Deployment Services Repository angeben. Beispiel:

```
modelerclient -spsscr_hostname myhost -spsscr_port 8080
-spsscr_username myusername -spsscr_password mypassword
-stream "spsscr:///folder_1/scoring_stream.str" -execute
```

In der folgenden Tabelle werden die Argumente aufgeführt, die zum Einrichten der Verbindung verwendet werden können.

Tabelle 37. IBM SPSS Collaboration and Deployment Services Repository-Verbindungsargumente

Argument	Verhalten/Beschreibung
`-spsscr_hostname` <hostname or IP address>	Der Hostname bzw. die IP-Adresse des Servers, auf dem IBM SPSS Collaboration and Deployment Services Repository installiert ist.
`-spsscr_port` <number>	Die Nummer des Ports, an dem IBM SPSS Collaboration and Deployment Services Repository Verbindungen akzeptiert (üblicherweise standardmäßig 8080).
`-spsscr_use_ssl`	Gibt an, dass die Verbindung SSL (Secure Socket Layer) verwenden sollte. Dieses Flag ist optional, bei der Standardeinstellung wird SSL *nicht* verwendet.
`-spsscr_username` <name>	Benutzername, mit dem die Anmeldung beim IBM SPSS Collaboration and Deployment Services Repository erfolgt.
`-spsscr_password` <password>	Kennwort, mit dem die Anmeldung beim IBM SPSS Collaboration and Deployment Services Repository erfolgt.
`-spsscr_epassword` <encoded password>	Codiertes Kennwort, mit dem die Anmeldung beim IBM SPSS Collaboration and Deployment Services Repository erfolgt.
`-spsscr_providername` <name>	Authentifizierungsprovider, mit dem die Anmeldung bei IBM SPSS Collaboration and Deployment Services Repository erfolgt (Active Directory oder LDAP). Dies ist nicht erforderlich, wenn der native Provider (lokales Repository) verwendet wird.

Argumente zum Herstellen einer Verbindung zu IBM SPSS Analytic Server

Wenn Sie Objekte über die Befehlszeile in IBM SPSS Analytic Server speichern oder von dort abrufen wollen, müssen Sie eine gültige Verbindung zu IBM SPSS Analytic Server angeben.

Anmerkung: Die Standardposition von Analytic Server wird aus SPSS Modeler Server abgerufen. Benutzer können über **Tools** > **Analytic Server-Verbindungen** auch ihre eigenen Analytic Server-Verbindungen definieren.

In der folgenden Tabelle werden die Argumente aufgeführt, die zum Einrichten der Verbindung verwendet werden können.

Tabelle 38. Argumente zum Herstellen einer Verbindung zu IBM SPSS Analytic Server

Argument	Verhalten/Beschreibung
`-analytic_server_username`	Benutzername, mit dem die Anmeldung an IBM SPSS Analytic Server erfolgt.

Tabelle 38. Argumente zum Herstellen einer Verbindung zu IBM SPSS Analytic Server (Forts.)	
Argument	**Verhalten/Beschreibung**
`-analytic_server_password`	Kennwort, mit dem die Anmeldung an IBM SPSS Analytic Server erfolgt.
`-analytic_server_epassword`	Verschlüsseltes Kennwort, mit dem die Anmeldung an IBM SPSS Analytic Server erfolgt.
`-analytic_server_credential`	Berechtigungsnachweise, mit denen die Anmeldung an IBM SPSS Analytic Server erfolgt.

Kombinieren mehrerer Argumente

Sie können mehrere Argumente in einer einzigen Befehlsdatei kombinieren, die mit dem Symbol @, gefolgt vom Dateinamen, beim Aufrufen angegeben wird. Auf diese Weise können Sie das Aufrufen über die Befehlszeile verkürzen und die im Betriebssystem geltenden Einschränkungen für die Befehlslänge umgehen. Der folgende Startbefehl verwendet beispielsweise die Argumente, die in der von <command-FileName> referenzierten Datei angegeben sind.

```
modelerclient @<commandFileName>
```

Schließen Sie den Dateinamen und den Pfad in Anführungszeichen ein, falls Leerzeichen erforderlich sind, beispielsweise:

```
modelerclient @ "C:\Program Files\IBM\SPSS\Modeler\nn\scripts\my_com
mand_file.txt"
```

Die Befehlsdatei kann alle Argumente umfassen, die zuvor beim Starten einzeln angegeben wurden, und zwar mit jeweils einem Argument pro Zeile. Beispiel:

```
-stream report.str
-Porder.full_filename=APR_orders.dat
-Preport.filename=APR_report.txt
-execute
```

Beim Schreiben und Referenzieren von Befehlsdateien sind die folgenden Einschränkungen zu beachten:

- Geben Sie nur je einen Befehl pro Zeile ein.
- Geben Sie ein Argument `@CommandFile` nicht in einer Befehlsdatei an.

Kapitel 7. Eigenschaftsreferenz

Eigenschaftsreferenz - Übersicht

Für Knoten, Streams, Projekte und Superknoten können Sie eine Reihe von verschiedenen Eigenschaften festlegen. Einige Eigenschaften wie Name, Anmerkung und QuickInfo gelten für alle Knoten, während andere sich nur auf bestimmte Knotentypen beziehen. Wieder andere Eigenschaften beziehen sich auf Streamoperationen auf hoher Ebene wie Zwischenspeichern oder das Verhalten von Superknoten. Der Zugriff auf Eigenschaften erfolgt über die Standardbenutzerschnittstelle (z. B. beim Öffnen eines Dialogfelds zum Bearbeiten von Optionen für einen Knoten). Eigenschaften können auf vielfältige Weise verwendet werden.

- Eigenschaften lassen sich mit Scripts ändern, wie in diesem Abschnitt beschrieben. Weitere Informationen finden Sie unter „Syntax für Eigenschaften” auf Seite 77.
- Knoteneigenschaften können in Superknotenparametern verwendet werden.
- Knoteneigenschaften können auch als Teil einer Befehlszeilenoption (mit dem Flag `-P`) verwendet werden, wenn IBM SPSS Modeler gestartet wird.

Im Zusammenhang mit Scripts in IBM SPSS Modeler werden Knoten- und Streameigenschaften häufig als **Slotparameter** bezeichnet. In diesem Handbuch werden sie als Knoten- oder Streameigenschaften beschrieben.

Syntax für Eigenschaften

Eigenschaften können mit der folgenden Syntax festgelegt werden.

```
OBJECT.setPropertyValue(PROPERTY, VALUE)
```

oder:

```
OBJECT.setKeyedPropertyValue(PROPERTY, KEY, VALUE)
```

Der Wert von Eigenschaften kann mit der folgenden Syntax abgerufen werden:

```
VARIABLE = OBJECT.getPropertyValue(PROPERTY)
```

oder:

```
VARIABLE = OBJECT.getKeyedPropertyValue(PROPERTY, KEY)
```

Dabei ist `OBJECT` ein Knoten oder eine Ausgabe, `PROPERTY` ist der Name der Knoteneigenschaft, auf die Ihr Ausdruck verweist, und `KEY` ist der Schlüsselwert für verschlüsselte Eigenschaften. Beispielsweise wird die folgende Syntax verwendet, um den Filterknoten zu suchen und dann den Standard so festzulegen, dass alle Felder eingeschlossen werden und das Feld Age in nachfolgenden Daten gefiltert wird:

```
filternode = modeler.script.stream().findByType("filter", None)
filternode.setPropertyValue("default_include", True)
filternode.setKeyedPropertyValue("include", "Age", False)
```

Alle in IBM SPSS Modeler verwendeten Knoten können mithilfe der Streamfunktion `findByType(TYPE, LABEL)` gesucht werden. Mindestens eine Instanz von `TYPE` oder `LABEL` muss angegeben werden.

Strukturierte Eigenschaften

Es gibt zwei Möglichkeiten, wie Scripts strukturierte Eigenschaften verwenden können, um eine größere Klarheit bei der Analyse zu erreichen:

- Den Namen der Eigenschaften für komplexe Knoten, wie Typ-, Filter- und Balancierungsknoten, strukturieren.
- Ein Format zum Festlegen mehrerer Eigenschaften gleichzeitig angeben.

Strukturieren komplexer Benutzerschnittstellen

Die Scripts für Knoten mit Tabellen und anderen komplexen Benutzerschnittstellen (z. B. Typ-, Filter- und Balancierungsknoten) müssen eine bestimmte Struktur besitzen, damit die Analyse ordnungsgemäß erfolgt. Für diese Eigenschaften ist ein komplexerer Name als für einen einzelnen IDs erforderlich. Dieser Name wird als Schlüssel bezeichnet. Innerhalb eines Filterknotens wird beispielsweise jedes verfügbare Feld (auf der vorausgehenden Seite) ein- oder ausgeschaltet. Um auf diese Information zurückzugreifen, speichert der Filterknoten ein Informationselement pro Feld (ob das Feld jeweils wahr oder falsch ist). Diese Eigenschaft kann den Wert `True` oder `False` besitzen (bzw. zugewiesen bekommen). Angenommen, ein Filterknoten mit dem Namen `mynode` hat (auf der Upstream-Seite) ein Feld mit dem Namen `Age`. Um dieses auszuschalten, legen Sie für die Eigenschaft `include` mit dem Schlüssel `Age` wie folgt den Wert `False` fest:

```
mynode.setKeyedPropertyValue("include", "Age", False)
```

Strukturieren zum Festlegen mehrerer Eigenschaften

Für mehrere Knoten können Sie mehr als einen Knoten oder eine Streameigenschaft gleichzeitig zuweisen. Dies wird als **Multiset-Befehl** oder **Blockset** bezeichnet.

In manchen Fällen kann eine strukturierte Eigenschaft äußerst komplex sein. Ein Beispiel:

```
sortnode.setPropertyValue("keys", [["K", "Descending"], ["Age", "Ascen
ding"], ["Na", "Descending"]])
```

Ein weiterer Vorteil strukturierter Eigenschaften besteht darin, dass mehrere Eigenschaften an einem Knoten festgelegt werden können, bevor der Knoten stabil ist. Standardmäßig legt ein Multiset alle Eigenschaften in einem Block fest, bevor je nach individueller Eigenschafteneinstellung Vorgänge ausgeführt werden. Beispiel: Wenn die Feldeigenschaften beim Definieren eines Knotes "Datei (fest)" in zwei Schritten festgelegt werden, treten Fehler auf, da der Knoten erst konsistent ist, wenn beide Einstellungen gültig sind. Durch Definieren der Eigenschaften als Multiset wird dieses Problem umgangen, indem beide Eigenschaften festgelegt werden, bevor das Datenmodell aktualisiert wird.

Abkürzungen

Für die Syntax der Knoteneigenschaften werden Standardabkürzungen verwendet. Sich mit den Abkürzungen vertraut zu machen kann beim Erstellen von Scripts sehr hilfreich sein.

Tabelle 39. In der Syntax verwendete Standardabkürzungen

Abkürzung	**Bedeutung**
Abs.	Absoluter Wert
len	Länge
Min.	Minimal
Max.	Maximum
correl	Korrelation

Tabelle 39. In der Syntax verwendete Standardabkürzungen (Forts.)	
Abkürzung	**Bedeutung**
covar	Kovarianz
num	Zahl oder numerisch
pct	Prozent oder Prozentsatz
transp	Transparenz
xval	Kreuzvalidierung
Var	Varianz oder Variable (in Quellenknoten)

Beispiele für Knoten- und Streameigenschaften

Mit IBM SPSS Modeler können Knoten- und Streameigenschaften auf vielfältige Weise verwendet werden. Meistens werden sie in einem Script, entweder einem **Standalone-Script** zur Automatisierung mehrerer Streams oder Operationen oder einem **Stream-Script** zur Automatisierung von Prozessen innerhalb eines einzelnen Streams, verwendet. Superknotenparameter können ebenfalls innerhalb des Superknotens anhand der Knoteneigenschaften angegeben werden. Auf niedrigster Ebene können Eigenschaften auch als Befehlszeilenoption zum Starten von IBM SPSS Modeler verwendet werden. Mit dem Argument `-p` als Teil des Befehlszeilenaufrufs können Sie eine Streameigenschaft verwenden, um eine Einstellung im Stream zu ändern.

Tabelle 40. Beispiele für Knoten- und Streameigenschaften	
Eigenschaft	**Bedeutung**
`s.max_size`	Bezieht sich auf die Eigenschaft `max_size` des Knotens `s`.
`s:samplenode.max_size`	Bezieht sich auf die Eigenschaft `max_size` des Knotens `s`, bei dem es sich um einen Stichprobenknoten handeln muss.
`:samplenode.max_size`	Bezieht sich auf die Eigenschaft `max_size` des Stichprobenknotens im aktuellen Stream (es darf nur ein Stichprobenknoten vorhanden sein).
`s:sample.max_size`	Bezieht sich auf die Eigenschaft `max_size` des Knotens `s`, bei dem es sich um einen Stichprobenknoten handeln muss.
`t.direction.Age`	Bezieht sich auf die Rolle des Felds *Alter* im Typknoten `t`.
`:.max_size`	*** NICHT ZULÄSSIG *** Sie müssen entweder den Knotennamen oder den Knotentyp angeben.

Das Beispiel `s:sample.max_size` veranschaulicht, dass Knotentypen nicht vollständig ausgeschrieben werden müssen.

Im Beispiel `t.direction.`Age wird deutlich, dass einige Slotnamen selbst strukturiert werden können, und zwar wenn die Attribute eines Knotens komplexer sind als nur individuelle Slots mit individuellen Werten. Solche Slots werden als **strukturierte** oder **komplexe** Eigenschaften bezeichnet.

Knoteneigenschaften - Übersicht

Jeder Knotentyp besitzt eine eigene Gruppe zulässiger Eigenschaften und jede Eigenschaft besitzt einen Typ. Dabei kann es sich um einen allgemeinen Typ einer Zahl, eines Flags, oder einer Zeichenfolgen

handeln. In diesem Fall wird für die Einstellungen der Eigenschaft der richtige Typ erzwungen. Wenn sie nicht erzwungen werden können, wird ein Fehler ausgegeben. Alternativ kann die Eigenschaftsreferenz den Bereich zulässiger Werte, wie `Discard`, `PairAndDiscard` und `IncludeAsText`, angeben. In diesem Fall tritt bei Verwendung eines anderen Werts ein Fehler auf. Flag-Eigenschaften sollten anhand der Werte `true` und `false` gelesen bzw. festgelegt werden. (Abweichungen wie beispielsweise `Off`, `OFF`, `off`, `No`, `NO`, `no`, `n`, `N`, `f`, `F`, `false`, `False`, `FALSE` oder `0` werden beim Festlegen der Werte ebenfalls erkannt, können jedoch beim Lesen der Eigenschaftswerte in einigen Fällen zu Fehlern führen. Alle anderen Werte werden als wahr betrachtet. Durch die durchgängige Verwendung von `true` und `false` können Verwechslungen vermieden werden.) Die Referenztabellen in diesem Handbuch weisen strukturierte Eigenschaften als solche in der Spalte **Eigenschaftsbeschreibung** aus und geben ihr Verwendungsformat an.

Allgemeine Knoteneigenschaften

Zahlreiche Eigenschaften beziehen sich in IBM SPSS Modeler auf alle Knoten (einschließlich Superknoten).

Tabelle 41. Allgemeine Knoteneigenschaften

Eigenschaftsname	Datentyp	Eigenschaftsbeschreibung
`use_custom_name`	*Flag*	
`name`	*string*	Schreibgeschützte Eigenschaft zum Lesen des Namens (entweder automatisch oder benutzerdefiniert) für einen Knoten im Erstellungsbereich.
`custom_name`	*string*	Gibt einen benutzerdefinierten Namen für den Knoten an.
`tooltip`	*string*	
`annotation`	*string*	
`keywords`	*string*	Strukturierter Slot, der eine Liste der mit dem Objekt verknüpften Schlüsselwörter angibt (Beispiel: `["Keyword1" "Keyword2"]`).
`cache_enabled`	*Flag*	

`node_type`	`source_supernode`
	`process_supernode`
	`terminal_supernode`
	alle angegebenen Knotennamen für
	Scripting
		Schreibgeschützte Eigenschaft, die den Bezug zu einem Knoten nach Typ herstellt. Statt auf den Knoten nur mit dem Namen zu verweisen, wie `real_income`, können Sie beispielsweise auch den Typ, wie `userinputnode` oder `filternode`, angeben.

Superknotenspezifische Eigenschaften werden wie alle anderen Knoten separat erörtert. Weitere Informationen finden Sie im Thema Kapitel 21, „Superknoteneigenschaften”, auf Seite 469.

Kapitel 8. Streameigenschaften

Verschiedene Streameigenschaften können durch Scripts gesteuert werden. Um Streameigenschaften zu referenzieren, müssen Sie die Ausführungsmethode für die Verwendung von Scripts festlegen:

```
stream = modeler.script.stream()
stream.setPropertyValue("execute_method", "Script")
```

Beispiel

Die Knoteneigenschaft dient zur Referenzierung der Knoten im aktuellen Stream. Das folgende Stream-Script ist ein Beispiel:

```
stream = modeler.script.stream()
annotation = stream.getPropertyValue("annotation")

annotation = annotation + "\n\nThis stream is called \"" + stream.getLabel() + "\" and
 contains the following nodes:\n"

for node in stream.iterator():
    annotation = annotation + "\n" + node.getTypeName() + " node called \"" + node.getLabel()
 + "\""

stream.setPropertyValue("annotation", annotation)
```

Im oben genannten Beispiel wird anhand der Knoteneigenschaft eine Liste aller Knoten im Stream erstellt und diese Liste in die Streamanmerkungen geschrieben. Die erzeugte Anmerkung sieht wie folgt aus:

```
This stream is called "druglearn" and contains the following nodes:

type node called "Define Types"
derive node called "Na_to_K"
variablefile node called "DRUG1n"
neuralnetwork node called "Drug"
c50 node called "Drug"
filter node called "Discard Fields"
```

In der folgenden Tabelle werden die Streameigenschaften beschrieben.

Tabelle 42. Streameigenschaften

Eigenschaftsname	Datentyp	Eigenschaftsbeschreibung
execute_method	Normal Script	

Tabelle 42. Streameigenschaften (Forts.)

Eigenschaftsname	Datentyp	Eigenschaftsbeschreibung
date_format	"DDMMYY" "MMDDYY" "YYMMDD" "YYYYMMDD" "YYYYDDD" DAY MONTH "DD-MM-YY" "DD-MM-YYYY" "MM-DD-YY" "MM-DD-YYYY" "DD-MON-YY" "DD-MON-YYYY" "YYYY-MM-DD" "DD.MM.YY" "DD.MM.YYYY" "MM.DD.YYYY" "DD.MON.YY" "DD.MON.YYYY" "DD/MM/YY" "DD/MM/YYYY" "MM/DD/YY" "MM/DD/YYYY" "DD/MON/YY" "DD/MON/YYYY" MON YYYY q Q YYYY ww WK YYYY	
date_baseline	*number*	
date_2digit_baseline	*number*	
time_format	"HHMMSS" "HHMM" "MMSS" "HH:MM:SS" "HH:MM" "MM:SS" "(H)H:(M)M:(S)S" "(H)H:(M)M" "(M)M:(S)S" "HH.MM.SS" "HH.MM" "MM.SS" "(H)H.(M)M.(S)S" "(H)H.(M)M" "(M)M.(S)S"	
time_rollover	*Flag*	
import_datetime_as_string	*Flag*	
decimal_places	*number*	
decimal_symbol	Default Period Comma	
angles_in_radians	*Flag*	
use_max_set_size	*Flag*	
max_set_size	*number*	

Tabelle 42. Streameigenschaften (Forts.)

Eigenschaftsname	Datentyp	Eigenschaftsbeschreibung
ruleset_evaluation	Voting FirstHit	
refresh_source_nodes	*Flag*	Dient zur automatischen Aktualisierung von Quellenknoten nach Ausführung des Streams.
script	*string*	
annotation	*string*	
name	*string*	**Anmerkung:** Diese Eigenschaft ist schreibgeschützt. Wenn Sie den Namen eines Streams ändern möchten, speichern Sie ihn unter einem anderen Namen.
parameters		Diese Eigenschaft dient zur Aktualisierung von Streamparametern innerhalb eines Standalone-Scripts.
nodes		Details finden Sie weiter unten.
encoding	SystemDefault "UTF-8"	
stream_rewriting	*Boolesch*	
stream_rewriting_maximise_sql	*Boolesch*	
stream_rewriting_optimise_clem_execution	*Boolesch*	
stream_rewriting_optimise_syntax_execution	*Boolesch*	
enable_parallelism	*Boolesch*	
sql_generation	*Boolesch*	
database_caching	*Boolesch*	
sql_logging	*Boolesch*	
sql_generation_logging	*Boolesch*	
sql_log_native	*Boolesch*	
sql_log_prettyprint	*Boolesch*	
record_count_suppress_input	*Boolesch*	

Tabelle 42. Streameigenschaften (Forts.)

Eigenschaftsname	Datentyp	Eigenschaftsbeschreibung
record_count_feedback_interval	*Integer*	
use_stream_auto_create_node_settings	*Boolesch*	Bei "true" werden die für den Stream spezifischen Einstellungen verwendet. Andernfalls werden die Vorgaben verwendet.
create_model_applier_for_new_models	*Boolesch*	Bei "true" wird ein neuer Modellanwender hinzugefügt, wenn ein Modellerstellungsprogramm ein neues Modell erstellt und es keine aktiven Update-Links hat. **Anmerkung:** Wenn Sie IBM SPSS Modeler Batch Version 15 verwenden, müssen Sie den Modellanwender explizit in Ihrem Script hinzufügen.
create_model_applier_update_links	createEnabled createDisabled doNotCreate	Definiert den Typ von Link, wenn ein Modellanwendungsknoten automatisch hinzugefügt wird.
create_source_node_from_builders	*Boolesch*	Bei "true" wird ein neuer Quellenknoten hinzugefügt, wenn ein Quellenerstellungsprogramm eine neue Quellenausgabe erstellt und sie keine aktiven Update-Links hat.
create_source_node_update_links	createEnabled createDisabled doNotCreate	Definiert den Typ von Link, wenn ein Quellenknoten automatisch hinzugefügt wird.
has_coordinate_system	*Boolesch*	Bei "true" wird ein Koordinatensystem auf den gesamten Datenstrom angewendet.
coordinate_system	*string*	Name des ausgewählten projizierten Koordinatensystems.
deployment_area	ModelRefresh Scoring None	Wählen Sie aus, wie der Stream bereitgestellt werden soll. Wenn dieser Wert auf None gesetzt wird, werden keine anderen Bereitstellungseinträge verwendet.
scoring_terminal_node_id	*string*	Wählen Sie die Scoring-Verzweigung im Stream aus. Dies kann ein beliebiger Endknoten im Stream sein.
scoring_node_id	*string*	Wählen Sie das Nugget in der Scoring-Verzweigung aus.
model_build_node_id	*string*	Wählen Sie den Modellierungsknoten im Stream aus.

Kapitel 9. Eigenschaften von Quellenknoten

Allgemeine Eigenschaften von Quellenknoten

Eigenschaften, die alle Quellenknoten besitzen, sind unten aufgelistet. Die darauf folgenden Themen enthalten Informationen zu bestimmten Knoten.

Beispiel 1

```
varfilenode = modeler.script.stream().create("variablefile", "Var. File")
varfilenode.setPropertyValue("full_filename", "$CLEO_DEMOS/DRUG1n")
varfilenode.setKeyedPropertyValue("check", "Age", "None")
varfilenode.setKeyedPropertyValue("values", "Age", [1, 100])
varfilenode.setKeyedPropertyValue("type", "Age", "Range")
varfilenode.setKeyedPropertyValue("direction", "Age", "Input")
```

Beispiel 2

Dieses Script geht davon aus, dass die angegebene Datendatei ein Feld mit der Bezeichnung Region enthält, das eine mehrzeilige Zeichenfolge darstellt.

```
from modeler.api import StorageType
from modeler.api import MeasureType

# Create a Variable File node that reads the data set containing
# the "Region" field
varfilenode = modeler.script.stream().create("variablefile", "My Geo Data")
varfilenode.setPropertyValue("full_filename", "C:/mydata/mygeodata.csv")
varfilenode.setPropertyValue("treat_square_brackets_as_lists", True)

# Override the storage type to be a list...
varfilenode.setKeyedPropertyValue("custom_storage_type", "Region", Storage
Type.LIST)
# ...and specify the type if values in the list and the list depth
varfilenode.setKeyedPropertyValue("custom_list_storage_type", "Region", Sto
rageType.INTEGER)
varfilenode.setKeyedPropertyValue("custom_list_depth", "Region", 2)

# Now change the measurement to indentify the field as a geospatial value...
varfilenode.setKeyedPropertyValue("measure_type", "Region", MeasureType.GEO
SPATIAL)
# ...and finally specify the necessary information about the specific
# type of geospatial object
varfilenode.setKeyedPropertyValue("geo_type", "Region", "MultiLineString")
varfilenode.setKeyedPropertyValue("geo_coordinates", "Region", "2D")
varfilenode.setKeyedPropertyValue("has_coordinate_system", "Region", True)
varfilenode.setKeyedPropertyValue("coordinate_system", "Region",
 "ETRS_1989_EPSG_Arctic_zone_5-47")
```

Tabelle 43. Allgemeine Eigenschaften von Quellenknoten

Eigenschaftsname	Datentyp	Eigenschaftsbeschreibung
direction	Input Target Both None Partition Split Frequency RecordID	Verschlüsselte Eigenschaft für Feldrollen. Format: NODE.direction.FIELDNAME **Anmerkung:** Die Werte In und Out werden nicht mehr verwendet. In zukünftigen Versionen fällt möglicherweise die Unterstützung dafür weg.
type	Range Flag Set Typeless Discrete Ordered Set Default	Feldtyp. Wenn diese Eigenschaft auf *Default* (Standard) gesetzt wird, werden alle Eigenschafteneinstellungen vom Typvalues gelöscht, und wenn value_mode den Wert *Default* (Angeben) besitzt, wird er auf *Read* (Lesen) zurückgesetzt. Wenn value_mode bereits auf *Pass* (Übergeben) oder *Read* (Lesen) gesetzt ist, hat das Einstellen von type keinerlei Auswirkung. Format: NODE.type.FIELDNAME
storage	Unknown String Integer Real Time Date Timestamp	Schreibgeschützte verschlüsselte Eigenschaft für Feldspeichertyp. Format: NODE.storage.FIELDNAME

Tabelle 43. Allgemeine Eigenschaften von Quellenknoten (Forts.)

Eigenschaftsname	Datentyp	Eigenschaftsbeschreibung
check	None Nullify Coerce Discard Warn Abort	Verschlüsselte Eigenschaft für das Überprüfen von Feldtyp und Bereich. Format: NODE.check.FIELDNAME
values	[*Wert Wert*]	Bei einem stetigen Feld (Bereich) ist der erste Wert das Minimum und der letzte das Maximum. Geben Sie für nominale Felder (Setfelder) alle Werte an. Bei Flagfeldern steht der erste Wert für *false* (falsch) und der letzte für *true* (wahr). Bei automatischer Festlegung dieser Eigenschaft wird die Eigenschaft value_mode auf *Specify* festgelegt. Die Speicherung wird basierend auf dem ersten Wert in der Liste festgelegt. Wenn z. B. der erste Wert eine *Zeichenfolge* ist, wird die Speicherung auf "String" gesetzt. Format: NODE.values.FIELDNAME
value_mode	Read Pass Read+ Current Specify	Bestimmt, wie Werte beim nächsten Datendurchlauf für ein Feld festgelegt werden. Format: NODE.value_mode.FIELDNAME Beachten Sie, dass Sie diese Eigenschaft nicht direkt auf *Specify* festlegen können. Um bestimmte Werte zu verwenden, legen Sie die Eigenschaft values fest.
default_value_mode	Read Pass	Gibt die Standardmethode für das Festlegen von Werten für alle Felder an. Format: NODE.default_value_mode Diese Einstellung kann mithilfe der Eigenschaft value_mode für bestimmte Felder überschrieben werden.

Tabelle 43. Allgemeine Eigenschaften von Quellenknoten (Forts.)

Eigenschaftsname	Datentyp	Eigenschaftsbeschreibung
`extend_values`	*Flag*	Gilt, wenn `value_mode` auf *Read* gesetzt ist. Setzen Sie den Wert auf *T*, um neu gelesene Werte zu bereits für das Feld vorhandenen Werten hinzuzufügen. Setzen Sie den Wert auf *F*, um vorhandene Werte zu verwerfen und sie durch neu gelesene Werte zu ersetzen. Format: `NODE.extend_values.FIELDNAME`
`value_labels`	*string*	Wird zur Angabe einer Wertbeschriftung verwendet. Beachten Sie, dass Werte zuerst angegeben werden müssen.
`enable_missing`	*Flag*	Bei Festlegung von *T* wird die Verfolgung von fehlenden Werten für das Feld aktiviert. Format: `NODE.enable_missing.FIELDNAME`
`missing_values`	[*Wert Wert ...*]	Gibt Datenwerte an, die fehlende Daten kennzeichnen. Format: `NODE.missing_values.FIELDNAME`
`range_missing`	*Flag*	Wenn diese Eigenschaft auf *T* gesetzt ist, wird angegeben, ob ein Bereich fehlender Werte (leer) für ein Feld definiert ist. Format: `NODE.range_missing.FIELDNAME`
`missing_lower`	*string*	Wenn `range_missing` "true" ist, gibt diese Eigenschaft die Untergrenze des Bereichs fehlender Werte an. Format: `NODE.missing_lower.FIELDNAME`
`missing_upper`	*string*	Wenn `range_missing` "true" ist, gibt diese Eigenschaft die Obergrenze des Bereichs fehlender Werte an. Format: `NODE.missing_upper.FIELDNAME`

Tabelle 43. Allgemeine Eigenschaften von Quellenknoten (Forts.)

Eigenschaftsname	Datentyp	Eigenschaftsbeschreibung
null_missing	*Flag*	Wenn diese Eigenschaft auf *T* gesetzt ist, werden Nullen (undefinierte Werte, die in der Software als $null$ angezeigt werden) als fehlende Werte betrachtet. Format: NODE.null_missing.FIELDNAME
whitespace_missing	*Flag*	Wenn diese Eigenschaft auf *T* festgelegt ist, werden Werte, die nur leere Bereiche enthalten (Leerzeichen, Tabulatoren und Zeilenumbrüche), als fehlende Werte betrachtet. Format: NODE.whitespace_missing.FIELDNAME
description	*string*	Dient zur Angabe einer Feldbeschriftung oder Beschreibung.
default_include	*Flag*	Verschlüsselte Eigenschaft zur Angabe, ob das Standardverhalten übergeben wird oder Felder gefiltert werden: NODE.default_include Beispiel: set mynode:filternode.default_include = false
include	*Flag*	Verschlüsselte Eigenschaft, mit der bestimmt wird, ob einzelne Felder aufgenommen oder gefiltert werden: NODE.include.FIELDNAME.
new_name	*string*	

Tabelle 43. Allgemeine Eigenschaften von Quellenknoten (Forts.)

Eigenschaftsname	Datentyp	Eigenschaftsbeschreibung
measure_type	Range / MeasureType.RANGE Discrete / MeasureType.DISCRETE Flag / MeasureType.FLAG Set / MeasureType.SET OrderedSet / MeasureType.ORDERED_SET Typeless / MeasureType.TYPELESS Collection / MeasureType.COLLECTION Geospatial / MeasureType.GEOSPATIAL	Diese verschlüsselte Eigenschaft ähnelt type dahingehend, dass sie zum Definieren der dem Feld zugeordneten Messung verwendet werden kann. Das Python-Scripting unterscheidet sich dadurch, dass der Setter-Funktion auch einer der MeasureType-Werte übergeben werden kann, während die Getter-Funktion immer für MeasureType-Werte zurückgibt.
collection_measure	Range / MeasureType.RANGE Flag / MeasureType.FLAG Set / MeasureType.SET OrderedSet / MeasureType.ORDERED_SET Typeless / MeasureType.TYPELESS	Bei Sammlungsfeldern (Listen mit einer Tiefe von 0) definiert diese verschlüsselte Eigenschaft den Messtyp, der den zugrunde liegenden Werten zugeordnet ist.

Tabelle 43. Allgemeine Eigenschaften von Quellenknoten (Forts.)

Eigenschaftsname	Datentyp	Eigenschaftsbeschreibung
geo_type	Point MultiPoint LineString MultiLineString Polygon MultiPolygon	Bei georäumlichen Feldern definiert diese verschlüsselte Eigenschaft den Typ des durch dieses Feld dargestellten georäumlichen Objekts. Dies sollte konsistent mit der Listentiefe der Werte sein.
has_coordinate_system	*Boolesch*	Bei georäumlichen Feldern definiert diese Eigenschaft, ob dieses Feld ein Koordinatensystem hat
coordinate_system	*string*	Bei georäumlichen Feldern definiert diese verschlüsselte Eigenschaft das Koordinatensystem für dieses Feld.
custom_storage_type	Unknown / MeasureType.UNKNOWN String / MeasureType.STRING Integer / MeasureType.INTEGER Real / MeasureType.REAL Time / MeasureType.TIME Date / MeasureType.DATE Timestamp / MeasureType.TIMESTAMP List / MeasureType.LIST	Diese verschlüsselte Eigenschaft ähnelt custom_storage dahingehend, dass sie zum Definieren der Speicherüberschreibung für das Feld verwendet werden kann. Das Python-Scripting unterscheidet sich dadurch, dass der Setter-Funktion auch einer der StorageType-Werte übergeben werden kann, während die Getter-Funktion immer für StorageType-Werte zurückgibt.

Tabelle 43. Allgemeine Eigenschaften von Quellenknoten (Forts.)		
Eigenschaftsname	**Datentyp**	**Eigenschaftsbeschreibung**
custom_list_storage_type	String / MeasureType.STRING Integer / MeasureType.INTEGER Real / MeasureType.REAL Time / MeasureType.TIME Date / MeasureType.DATE Timestamp / MeasureType.TIMESTAMP	Bei Listenfeldern gibt diese verschlüsselte Eigenschaft den Speichertyp der zugrunde liegenden Werte an.
custom_list_depth	*Integer*	Bei Listenfeldern gibt diese verschlüsselte Eigenschaft die Tiefe des Felds an.
max_list_length	*Integer*	Steht nur bei Daten mit dem Messniveau *Georäumlich* oder *Sammlung* zur Verfügung. Legen Sie die maximale Länge der Liste durch Angabe der Anzahl der Elemente fest, die die Liste enthalten kann.
max_string_length	*Integer*	Nur für Daten *ohne Typ* verfügbar und Verwendung beim Generieren von SQL zur Erstellung einer Tabelle. Geben Sie den Wert der längsten Zeichenfolge in Ihren Daten ein. Dadurch wird in der Tabelle eine Spalte generiert, die groß genug für diese Zeichenfolge ist.

Eigenschaften von "asimport"

Die Analytic Server-Quelle ermöglicht Ihnen die Ausführung eines Streams unter HDFS (Hadoop Distributed File System).

Beispiel

```
node.setPropertyValue("use_default_as", False)
node.setPropertyValue("connection",
["false","9.119.141.141","9080","analyticserver","ibm","admin","admin","fal
se","","","","",""])
```

Tabelle 44. Eigenschaften von "asimport"		
Eigenschaften von asimport	**Datentyp**	**Eigenschaftsbeschreibung**
data_source	*string*	Der Name der Datenquelle.

Tabelle 44. Eigenschaften von "asimport" (Forts.)		
Eigenschaften von asimport	**Datentyp**	**Eigenschaftsbeschreibung**
use_default_as	*Boolesch*	Wenn True festgelegt ist, wird die Analytic Server-Standardverbindung verwendet, die in der Serverdatei options.cfg konfiguriert ist. Wenn die Option auf False gesetzt ist, wird die Verbindung dieses Knotens verwendet.
connection	["string","string","string", "string","string","string","string", "string" ,"string","string", "string" ,"string"]	Eine Listeneigenschaft mit den Verbindungsdetails für Analytic Server. Das Format lautet: ["is_secure_connect", "server_url", "server_port", "context_root", "consumer", "user_name", "password", "use-kerberos-auth", "kerberos-krb5-config-file-path", "kerberos-jaas-config-file-path", "kerberos-krb5-service-principal-name", "enable-kerberos-debug"]. Dabei gibt is_secure_connect: an, ob eine sichere Verbindung verwendet wird, und ist entweder true oder false. use-kerberos-auth: gibt an, ob die Kerberos-Authentifizierung verwendet wird, und lautet entweder true oder false. enable-kerberos-debug: gibt an, ob der Debugmodus der Kerberos-Authentifizierung verwendet wird, und ist entweder true oder false.

Eigenschaften des Knotens "cognosimport"

Der IBM Cognos-Quellenknoten importiert Daten aus Cognos Analytics-Datenbanken.

Beispiel

```
node = stream.create("cognosimport", "My node")
node.setPropertyValue("cognos_connection", ["http://mycogsrv1:9300/p2pd/serv
let/dispatch",
 True, "", "", ""])
node.setPropertyValue("cognos_package_name", "/Public Folders/GOSALES")
node.setPropertyValue("cognos_items", ["[GreatOutdoors].[BRANCH].
[BRANCH_CODE]", "[GreatOutdoors]
.[BRANCH].[COUNTRY_CODE]"])
```

Tabelle 45. Eigenschaften des Knotens "cognosimport"		
Eigenschaften des Knotens `cognosimport`	**Datentyp**	**Eigenschaftsbeschreibung**

`mode`	`Data`
	
	`Report`
		Gibt an, ob Cognos-Daten (Data - Standard) oder Berichte (Report) importiert werden sollen.

Tabelle 45. Eigenschaften des Knotens "cognosimport" (Forts.)		
Eigenschaften des Knotens `cognosimport`	**Datentyp**	**Eigenschaftsbeschreibung**
`cognos_connection`	*["Zeichenfolge",flag,"Zeichenfolge", "Zeichenfolge", "Zeichenfolge"]*	Eine Listeneigenschaft mit den Verbindungsdetails für den Cognos-Server. Format: `["Cognos_server_URL", login_mode, "namespace", "username", "password"]` Dabei gilt Folgendes: `Cognos_server_URL` ist die URL des Cognos-Servers, der die Quelle enthält. `login_mode` gibt an, ob die anonyme Anmeldung verwendet wird. Der Wert lautet entweder `true` oder `false`. Wenn der Wert auf `true` gesetzt ist, sollten die folgenden Felder auf `""`. gesetzt werden `namespace` gibt den Sicherheitsauthentifizierungsprovider an, der für die Anmeldung am Server verwendet wird. `username` und `password` werden für die Anmeldung am Cognos-Server verwendet. Anstelle von `login_mode` sind die folgenden Modi verfügbar: • `anonymousMode`. Beispiel: `['Cognos_server_url', 'anonymousMode', "namespace", "username", "password"]` • `credentialMode`. Beispiel: `['Cognos_server_url', 'credentialMode', "namespace", "username", "password"]` • `storedCredentialMode`. Beispiel: `['Cognos_server_url', 'storedCredentialMode', "stored_credential_name"]` Dabei ist `stored_credential_name` der Name eines Cognos-Berechtigungsnachweises im Repository.

Tabelle 45. Eigenschaften des Knotens "cognosimport" (Forts.)		
Eigenschaften des Knotens `cognosimport`	**Datentyp**	**Eigenschaftsbeschreibung**
`cognos_package_name`	*string*	Pfad und Name des Cognos-Pakets, von dem Sie Datenobjekte importieren, z. B.: `/Public Folders/GOSALES` **Anmerkung:** Nur normale Schrägstriche sind gültig.
`cognos_items`	*["Feld", "Feld", ..., "Feld"]*	Der Name eines oder mehrerer Datenobjekte, die importiert werden sollen. Das Format von *Feld* ist [*Namespace*].[*Abfragesubjekt*].[*Abfrageelement*]
`cognos_filters`	*Feld*	Der Name eines oder mehrerer Filter, die vor dem Datenimport angewendet werden sollen.
`cognos_data_parameters`	*Liste*	Werte für Eingabeaufforderungsparameter für Daten. Paare aus Name und Wert sind in eckige Klammern eingeschlossen und mehrere Paare werden durch Kommas getrennt. Die gesamte Zeichenfolge steht in eckigen Klammern. Format: [["*Parameter1*", "*Wert*"],...,["*ParameterN*", "*Wert*"]]
`cognos_report_directory`	*Feld*	Der Cognos-Pfad eines Ordners bzw. Pakets, aus dem Berichte importiert werden sollen. Beispiel: `/Public Folders/GOSALES` **Anmerkung:** Nur normale Schrägstriche sind gültig.
`cognos_report_name`	*Feld*	Der Pfad und Name innerhalb des Speicherorts eines zu importierenden Berichts.

Tabelle 45. Eigenschaften des Knotens "cognosimport" (Forts.)		
Eigenschaften des Knotens `cognosimport`	**Datentyp**	**Eigenschaftsbeschreibung**
`cognos_report_parameters`	*Liste*	Werte für Berichtsparameter. Paare aus Name und Wert sind in eckige Klammern eingeschlossen und mehrere Paare werden durch Kommas getrennt. Die gesamte Zeichenfolge steht in eckigen Klammern. Format: [["*Parameter1*", "*Wert*"],...,["*ParameterN*", "*Wert*"]]

Eigenschaften von "databasenode"

Mit dem Datenbankknoten lassen sich Daten aus einer Reihe von anderen Paketen importieren, die ODBC (Open Database Connectivity) verwenden; dazu gehören Microsoft SQL Server, Db2, Oracle und andere.

Beispiel

```
import modeler.api
stream = modeler.script.stream()
node = stream.create("database", "My node")
node.setPropertyValue("mode", "Table")
node.setPropertyValue("query", "SELECT * FROM drug1n")
node.setPropertyValue("datasource", "Drug1n_db")
node.setPropertyValue("username", "spss")
node.setPropertyValue("password", "spss")
node.setPropertyValue("tablename", ".Drug1n")
```

Tabelle 46. Eigenschaften von "databasenode"		
Eigenschaften von `databasenode`	**Datentyp**	**Eigenschaftsbeschreibung**

`mode`	`Table`
	
	`Query`
		Legen Sie *Table* (Tabelle) fest, um die Verbindung zu einer Datenbanktabelle über Dialogfeldsteuerelemente herzustellen, oder *Query* (Abfrage), um die ausgewählte Datenbank mit SQL abzufragen.

`datasource`	*string*	Datenbankname (siehe auch Hinweis unten).
`username`	*string*	Datenbankverbindungsdetails (siehe auch Hinweis unten).
`password`	*string*	

Tabelle 46. Eigenschaften von "databasenode" (Forts.)		
Eigenschaften von databasenode	**Datentyp**	**Eigenschaftsbeschreibung**
credential	*string*	Name des in IBM SPSS Collaboration and Deployment Services gespeicherten Berechtigungsnachweises. Er kann anstelle der Eigenschaften username und password verwendet werden. Der Benutzername und das Kennwort des Berechtigungsnachweises müssen mit dem Benutzernamen und Kennwort übereinstimmen, die zum Zugreifen auf die Datenbank erforderlich sind.
use_credential		Setzen Sie diese Eigenschaft auf True oder False.
epassword	*string*	Gibt ein verschlüsseltes Kennwort an, als Alternative zum festen Verschlüsseln eines Kennworts in einem Script. Weitere Informationen finden Sie im Thema „Erstellen eines verschlüsselten Kennworts" auf Seite 58. Diese Eigenschaft ist während der Ausführung schreibgeschützt.
tablename	*string*	Name der Tabelle, auf die Sie zugreifen wollen.
strip_spaces	None Left Right Both	Optionen zum Verwerfen von führenden und nachfolgenden Leerzeichen in Zeichenfolgen.
use_quotes	AsNeeded Always Never	Geben Sie an, ob die Tabellen- und Spaltennamen in Anführungszeichen eingeschlossen sind, wenn Abfragen an die Datenbank gesendet werden (wenn sie z. B. Leerzeichen oder Satzzeichen enthalten).
query	*string*	Gibt den SQL-Code für die Abfrage an, die Sie senden wollen.

Anmerkung: Wenn der Datenbankname (in der Eigenschaft datasource) Leerzeichen enthält, können Sie anstelle einzelner Eigenschaften für datasource, username und password auch eine einzige Datenquelleneigenschaft im folgenden Format verwenden:

Tabelle 47. Datenquellenspezifische Eigenschaften von "databasenode"		
Eigenschaften von databasenode	**Datentyp**	**Eigenschaftsbeschreibung**
datasource	*string*	Format: [database_name,username,password[,true | false]] Der letzte Parameter ist für die Verwendung bei verschlüsselten Kennwörtern gedacht. Wenn er auf true gesetzt ist, wird das Kennwort vor der Verwendung verschlüsselt.

Verwenden Sie ebenfalls dieses Format, wenn Sie die Datenquelle ändern; falls Sie jedoch nur den Benutzernamen oder das Kennwort wollen ändern, können Sie auch die Eigenschaften username oder password verwenden.

Eigenschaften von "datacollectionimportnode"

Der Data Collection-Datenimportknoten importiert Umfragedaten auf der Grundlage des von den Marktforschungsprodukten verwendeten Data Collection Data Model. Um diesen Knoten verwenden zu können, muss die Data Collection Data Library installiert sein.

Beispiel

```
node = stream.create("datacollectionimport", "My node")
node.setPropertyValue("metadata_name", "mrQvDsc")
node.setPropertyValue("metadata_file", "C:/Program Files/IBM/SPSS/DataCollec
tion/DDL/Data/
Quanvert/Museum/museum.pkd")
node.setPropertyValue("casedata_name", "mrQvDsc")
node.setPropertyValue("casedata_source_type", "File")
node.setPropertyValue("casedata_file", "C:/Program Files/IBM/SPSS/DataCollec
tion/DDL/Data/
Quanvert/Museum/museum.pkd")
node.setPropertyValue("import_system_variables", "Common")
node.setPropertyValue("import_multi_response", "MultipleFlags")
```

Tabelle 48. Eigenschaften von "datacollectionimportnode"

Eigenschaften von datacollectionimportnode	**Datentyp**	**Eigenschaftsbeschreibung**
metadata_name	*string*	Der Name des MDSC. Der spezielle Wert DimensionsMDD gibt an, dass das standardmäßige Data Collection-Metadatendokument verwendet werden soll. Weitere mögliche Werte: mrADODsc mrI2dDsc mrLogDsc mrQdiDrsDsc mrQvDsc mrSampleReportingMDSC mrSavDsc mrSCDsc mrScriptMDSC Der spezielle Wert none zeigt an, dass es keinen MDSC gibt.
metadata_file	*string*	Name der Datei, in der die Metadaten gespeichert werden.

Tabelle 48. Eigenschaften von "datacollectionimportnode" (Forts.)

Eigenschaften von datacollectionimportnode	**Datentyp**	**Eigenschaftsbeschreibung**
casedata_name	*string*	Der Name des CDSC. Mögliche Werte: mrADODsc mrI2dDsc mrLogDsc mrPunchDSC mrQdiDrsDsc mrQvDsc mrRdbDsc2 mrSavDsc mrScDSC mrXmlDsc Der spezielle Wert none zeigt an, dass es keinen CDSC gibt.
casedata_source_type	Unknown File Folder UDL DSN	Gibt den Quellentyp des CDSC an.
casedata_file	*string*	Wenn casedata_source_type den Wert *File* aufweist, wird hier die Datei angegeben, die die Falldaten enthält.
casedata_folder	*string*	Wenn casedata_source_type den Wert *Folder* aufweist, wird hier der Ordner angegeben, der die Falldaten enthält.
casedata_udl_string	*string*	Wenn casedata_source_type den Wert *UDL* aufweist, wird hier die OLE-DB-Verbindungszeichenfolge für die Datenquelle angegeben, die die Falldaten enthält.

Tabelle 48. Eigenschaften von "datacollectionimportnode" (Forts.)		
Eigenschaften von datacollectionimportnode	**Datentyp**	**Eigenschaftsbeschreibung**
casedata_dsn_string	*string*	Wenn casedata_source_type den Wert *DSN*, aufweist, wird hier die ODBC-Verbindungszeichenfolge für die Datenquelle angegeben.
casedata_project	*string*	Beim Lesen von Falldaten aus einer Data Collection-Datenbank, können Sie den Namen des Projekts eingeben. Bei allen anderen Falldatentypen sollte diese Einstellung leer bleiben.
version_import_mode	All Latest Specify	Gibt an, wie mit Versionen umgegangen werden soll.
specific_version	*string*	Wenn version_import_mode den Wert *Specify* aufweist, wird hier die Version der zu importierenden Falldaten festgelegt.
use_language	*string*	Gibt an, ob Beschriftungen einer bestimmten Sprache verwendet werden sollen.
language	*string*	Wenn use_language den Wert "true" aufweist, wird hier der beim Import zu verwendende Sprachcode festgelegt. Bei diesem Sprachcode sollte es sich um einen der in den Falldaten verfügbaren Sprachcodes handeln.
use_context	*string*	Gibt an, ob ein bestimmter Kontext importiert werden sollte. Kontexte dienen dazu, die den Antworten zugeordnete Beschreibung zu variieren.
context	*string*	Wenn use_context den Wert "true"aufweist, wird hier der zu importierende Kontext festgelegt. Bei diesem Kontext sollte es sich um einen der in den Falldaten verfügbaren Kontexte handeln.
use_label_type	*string*	Gibt an, ob ein bestimmter Beschriftungstyp importiert werden sollte.
label_type	*string*	Wenn use_label_type den Wert "true" aufweist, wird hier der zu importierende Beschriftungstyp festgelegt. Bei diesem Beschriftungstyp sollte es sich um einen der in den Falldaten verfügbaren Beschriftungstypen handeln.

Tabelle 48. Eigenschaften von "datacollectionimportnode" (Forts.)		
Eigenschaften von datacollectionimportnode	**Datentyp**	**Eigenschaftsbeschreibung**
user_id	*string*	Bei Datenbanken, bei denen eine explizite Anmeldung erforderlich ist, können Sie eine Benutzer-ID und ein Kennwort für den Zugriff auf die Datenquelle angeben.
password	*string*	
import_system_variables	Common None All	Gibt an, welche Systemvariablen importiert werden sollen.
import_codes_variables	*Flag*	
import_sourcefile_variables	*Flag*	
import_multi_response	MultipleFlags Single	

Eigenschaften von "excelimportnode"

Der Excel-Importknoten importiert Daten aus Microsoft Excel im XLSX-Dateiformat. Es ist keine ODBC-Datenquelle erforderlich.

Beispiele

```
#To use a named range:
node = stream.create("excelimport", "My node")
node.setPropertyValue("excel_file_type", "Excel2007")
node.setPropertyValue("full_filename", "C:/drug.xlsx")
node.setPropertyValue("use_named_range", True)
node.setPropertyValue("named_range", "DRUG")
node.setPropertyValue("read_field_names", True)

#To use an explicit range:
node = stream.create("excelimport", "My node")
node.setPropertyValue("excel_file_type", "Excel2007")
node.setPropertyValue("full_filename", "C:/drug.xlsx")
node.setPropertyValue("worksheet_mode", "Name")
node.setPropertyValue("worksheet_name", "Drug")
node.setPropertyValue("explicit_range_start", "A1")
node.setPropertyValue("explicit_range_end", "F300")
```

Tabelle 49. excelimportnode, Eigenschaften		
Eigenschaften von excelimportnode	**Datentyp**	**Eigenschaftsbeschreibung**
excel_file_type	Excel2007	
full_filename	*string*	Der vollständige Dateiname mit Pfad.

Tabelle 49. excelimportnode, Eigenschaften (Forts.)		
Eigenschaften von excelimportnode	**Datentyp**	**Eigenschaftsbeschreibung**
use_named_range	*boolesch*	Gibt an, ob ein benannter Bereich verwendet werden soll. Bei "true" wird die Eigenschaft named_range zur Angabe des zu lesenden Bereichs verwendet. Andere Einstellungen für Arbeitsblatt und Datenbereich werden ignoriert.
named_range	*string*	
worksheet_mode	Index Name	Gibt an, ob das Arbeitsblatt durch Index oder durch Namen definiert wird.
worksheet_index	*Integer*	Index des zu lesenden Arbeitsblattes, beginnend mit 0 für das erste Arbeitsblatt, 1 für das zweite usw.
worksheet_name	*string*	Name des zu lesenden Arbeitsblattes.
data_range_mode	FirstNonBlank ExplicitRange	Gibt an, wie der Bereich festgelegt werden sollte.
blank_rows	StopReading ReturnBlankRows	Wenn data_range_mode den Wert *FirstNonBlank* aufweist, wird hier angegeben, wie mit leeren Zeilen umgegangen werden soll.
explicit_range_start	*string*	Wenn data_range_mode den Wert *ExplicitRange* aufweist, wird hier der Startpunkt des zu lesenden Bereichs angegeben.
explicit_range_end	*string*	
read_field_names	*boolesch*	Gibt an, ob die erste Zeile im angegebenen Bereich als Feldnamen (Spaltennamen) verwendet werden soll.
scanLineCount	*Integer*	Gibt die Anzahl der Zeilen an, die für den Spalten und Speichertyp durchsucht werden sollen. Der Standardwert ist 200.

extensionimportnode, Eigenschafen

Mit dem Erweiterungsimportknoten können Sie Scripts in R oder Python for Spark ausführen, um Daten zu importieren.

Beispiel für Python for Spark

```
##### Script example for Python for Spark
import modeler.api
```

```
stream = modeler.script.stream()
node = stream.create("extension_importer", "extension_importer")
node.setPropertyValue("syntax_type", "Python")

python_script = """
import spss.pyspark
from pyspark.sql.types import *

cxt = spss.pyspark.runtime.getContext()

_schema = StructType([StructField('id', LongType(), nullable=False), \
StructField('age', LongType(), nullable=True), \
StructField('Sex', StringType(), nullable=True), \
StructField('BP', StringType(), nullable=True), \
StructField('Cholesterol', StringType(), nullable=True), \
StructField('K', DoubleType(), nullable=True), \
StructField('Na', DoubleType(), nullable=True), \
StructField('Drug', StringType(), nullable=True)])

if cxt.isComputeDataModelOnly():
    cxt.setSparkOutputSchema(_schema)
else:
    df = cxt.getSparkInputData()
    if df is None:
        drugList=[(1,23,'F','HIGH','HIGH',0.792535,0.031258,'drugY'), \
(2,47,'M','LOW','HIGH',0.739309,0.056468,'drugC'),\
              (3,47,'M','LOW','HIGH',0.697269,0.068944,'drugC'),\
              (4,28,'F','NORMAL','HIGH',0.563682,0.072289,'drugX'),\
              (5,61,'F','LOW','HIGH',0.559294,0.030998,'drugY'),\
              (6,22,'F','NORMAL','HIGH',0.676901,0.078647,'drugX'),\
              (7,49,'F','NORMAL','HIGH',0.789637,0.048518,'drugY'),\
              (8,41,'M','LOW','HIGH',0.766635,0.069461,'drugC'),\
              (9,60,'M','NORMAL','HIGH',0.777205,0.05123,'drugY'),\
              (10,43,'M','LOW','NORMAL',0.526102,0.027164,'drugY')]
        sqlcxt = cxt.getSparkSQLContext()
        rdd = cxt.getSparkContext().parallelize(drugList)
        print 'pyspark read data count = '+str(rdd.count())
        df = sqlcxt.createDataFrame(rdd, _schema)

    cxt.setSparkOutputData(df)
"""

node.setPropertyValue("python_syntax", python_script)
```

Beispiel für R

```
#### Script example for R
node.setPropertyValue("syntax_type", "R")

R_script = """# 'JSON Import' Node v1.0 for IBM SPSS Modeler
# 'RJSONIO' package created by Duncan Temple Lang - http://cran.r-project.org/web/packages/RJSO
NIO
# 'plyr' package created by Hadley Wickham http://cran.r-project.org/web/packages/plyr
# Node developer: Danil Savine - IBM Extreme Blue 2014
# Description: This node allows you to import into SPSS a table data from  a JSON.
# Install function for packages
packages <- function(x){
  x <- as.character(match.call()[[2]])
  if (!require(x,character.only=TRUE)){
    install.packages(pkgs=x,repos="http://cran.r-project.org")
    require(x,character.only=TRUE)
  }
}
#  packages
packages(RJSONIO)
packages(plyr)
### This function is used to generate automatically the dataModel
getMetaData <- function (data) {
  if (dim(data)[1]<=0) {

    print("Warning : modelerData has no line, all fieldStorage fields set to strings")
    getStorage <- function(x){return("string")}

  } else {

    getStorage <- function(x) {
      res <- NULL
      #if x is a factor, typeof will return an integer so we treat the case on the side
      if(is.factor(x)) {
```

```
        res <- "string"
      } else {
        res <- switch(typeof(unlist(x)),
                      integer = "integer",
                      double = "real",
                      character = "string",
                      "string")
      }
      return (res)
    }
  }

  col = vector("list", dim(data)[2])
  for (i in 1:dim(data)[2]) {
    col[[i]] <- c(fieldName=names(data[i]),
                  fieldLabel="",
                  fieldStorage=getStorage(data[i]),
                  fieldMeasure="",
                  fieldFormat="",
                  fieldRole="")
  }
  mdm<-do.call(cbind,col)
  mdm<-data.frame(mdm)
  return(mdm)
}
# From JSON to a list
txt <- readLines('C:/test.json')
formatedtxt <- paste(txt, collapse = '')
json.list <- fromJSON(formatedtxt)
  # Apply path to json.list
  if(strsplit(x='true', split='
  ' ,fixed=TRUE)[[1]][1]) {
    path.list <- unlist(strsplit(x='id_array', split=','))
    i = 1
    while(i<length(path.list)+1){
      if(is.null(getElement(json.list, path.list[i]))){
        json.list <- json.list[[1]]
      }else{
      json.list <- getElement(json.list, path.list[i])
      i <- i+1
      }
    }
  }
# From list to dataframe via unlisted json
i <-1
filled <- data.frame()
while(i < length(json.list)+ 1){
  unlisted.json <- unlist(json.list[[i]])
  to.fill <- data.frame(t(as.data.frame(unlisted.json, row.names = names(unlisted.json))),
stringsAsFactors=FALSE)
  filled <- rbind.fill(filled,to.fill)
  i <-  1 + i
}
# Export to SPSS Modeler Data
modelerData <- filled
print(modelerData)
modelerDataModel <- getMetaData(modelerData)
print(modelerDataModel)

"""

node.setPropertyValue("r_syntax", R_script)
```

Tabelle 50. extensionimportnode, Eigenschafen

Eigenschaften von `extensionimportnode`	**Datentyp**	**Eigenschaftsbeschreibung**
`syntax_type`	*R* *Python*	Gibt das Script an, das ausgeführt wird - R oder Python (R ist der Standardwert).
`r_syntax`	*string*	Die R-Scriptsyntax für die Ausführung.
`python_syntax`	*string*	Die Python-Scriptsyntax für die Ausführung.

Eigenschaften von "fixedfilenode"

Der Knoten "Datei (fest)" importiert Daten aus Textdateien mit festen Feldern, also aus Dateien, deren Felder nicht begrenzt sind, sondern an derselben Position beginnen und eine feste Länge haben. Maschinell generierte Daten oder Bestandsdaten werden häufig im Format mit festen Feldern gespeichert.

Beispiel

```
node = stream.create("fixedfile", "My node")
node.setPropertyValue("full_filename", "$CLEO_DEMOS/DRUG1n")
node.setPropertyValue("record_len", 32)
node.setPropertyValue("skip_header", 1)
node.setPropertyValue("fields", [["Age", 1, 3], ["Sex", 5, 7], ["BP", 9,
10], ["Cholesterol",
 12, 22], ["Na", 24, 25], ["K", 27, 27], ["Drug", 29, 32]])
node.setPropertyValue("decimal_symbol", "Period")
node.setPropertyValue("lines_to_scan", 30)
```

Tabelle 51. Eigenschaften von "fixedfilenode"

Eigenschaften von `fixedfile-node`	**Datentyp**	**Eigenschaftsbeschreibung**
`record_len`	*number*	Legt die Zahl der Zeichen in jedem Datensatz fest.
`line_oriented`	*Flag*	Überspringt am Ende jedes Datensatzes das Zeilenwechselzeichen.

`decimal_symbol`	`Default`
	`Comma`
	`Period`
		Der Typ des in Ihrer Datenquelle verwendeten Dezimaltrennzeichens.

`skip_header`	*number*	Gibt die Anzahl der ab dem Beginn des ersten Datensatzes zu ignorierenden Zeilen an. Dies ist nützlich, um Spaltenkopfzeilen zu ignorieren.
`auto_recognize_datetime`	*Flag*	Gibt an, ob Datums- oder Zeitangaben automatisch in den Quellendaten identifiziert werden.
`lines_to_scan`	*number*	
`fields`	*Liste*	Strukturierte Eigenschaft.
`full_filename`	*string*	Vollständiger Name der zu lesenden Datei, einschließlich Pfadangabe.

`strip_spaces`	`None`
	`Left`
	`Right`
	`Both`
		Verwirft beim Importieren führende und nachfolgende Leerzeichen in Zeichenfolgen.

Tabelle 51. Eigenschaften von "fixedfilenode" (Forts.)		
Eigenschaften von `fixedfile-node`	**Datentyp**	**Eigenschaftsbeschreibung**

`invalid_char_mode`	`Discard`
	
	`Replace`
		Entfernt ungültige Zeichen (null, 0 oder jedes nicht in der aktuellen Codierung enthaltene Zeichen) aus der Dateneingabe oder ersetzt ungültige Zeichen durch das festgelegte, aus einem Zeichen bestehende Symbol.

`invalid_char_replacement`	*string*	
`use_custom_values`	*Flag*	

`custom_storage`	`Unknown`
	
	`String`
	
	`Integer`
	
	`Real`
	
	`Time`
	
	`Date`
	
	`Timestamp`	

Tabelle 51. Eigenschaften von "fixedfilenode" (Forts.)

Eigenschaften von `fixedfile-node`	**Datentyp**	**Eigenschaftsbeschreibung**

`custom_date_format`	`"DDMMYY"`
	`"MMDDYY"`
	`"YYMMDD"`
	`"YYYYMMDD"`
	`"YYYYDDD"`
	`DAY`
	`MONTH`
	`"DD-MM-YY"`
	`"DD-MM-YYYY"`
	`"MM-DD-YY"`
	`"MM-DD-YYYY"`
	`"DD-MON-YY"`
	`"DD-MON-YYYY"`
	`"YYYY-MM-DD"`
	`"DD.MM.YY"`
	`"DD.MM.YYYY"`
	`"MM.DD.YY"`
	`"MM.DD.YYYY"`
	`"DD.MON.YY"`
	`"DD.MON.YYYY"`
		Diese Eigenschaft ist nur anwendbar, wenn ein benutzerdefinierter Speichertyp angegeben wurde.

Tabelle 51. Eigenschaften von "fixedfilenode" (Forts.)

Eigenschaften von `fixedfile-node`	Datentyp	Eigenschaftsbeschreibung
	"DD/MM/YY" "DD/MM/YYYY" "MM/DD/YY" "MM/DD/YYYY" "DD/MON/YY" "DD/MON/YYYY" MON YYYY q Q YYYY ww WK YYYY	

Tabelle 51. Eigenschaften von "fixedfilenode" (Forts.)		
Eigenschaften von `fixedfile-node`	**Datentyp**	**Eigenschaftsbeschreibung**

`custom_time_format`	`"HHMMSS"`
	`"HHMM"`
	`"MMSS"`
	`"HH:MM:SS"`
	`"HH:MM"`
	`"MM:SS"`
	`"(H)H:(M)M:(S)S"`
	`"(H)H:(M)M"`
	`"(M)M:(S)S"`
	`"HH.MM.SS"`
	`"HH.MM"`
	`"MM.SS"`
	`"(H)H.(M)M.(S)S"`
	`"(H)H.(M)M"`
	`"(M)M.(S)S"`
		Diese Eigenschaft ist nur anwendbar, wenn ein benutzerdefinierter Speichertyp angegeben wurde.

`custom_decimal_symbol`	*Feld*	Nur anwendbar, wenn ein benutzerdefinierter Speichertyp angegeben wurde.

`encoding`	`StreamDefault`
	`SystemDefault`
	`"UTF-8"`
		Legt die Textcodierungsmethode fest.

Eigenschaften des Knotens "gsdata_import"

Mit dem georäumlichen Quellenknoten können Sie Kartendaten oder georäumliche Daten in Ihre Datenmining-Sitzung einführen.

Tabelle 52. Eigenschaften des Knotens "gsdata_import"		
Eigenschaften des Knotens `gsdata_import`	**Datentyp**	**Eigenschaftsbeschreibung**
`full_filename`	*string*	Geben Sie den Dateipfad zu der .shp-Datei ein, die Sie laden wollen.
`map_service_URL`	*string*	Geben Sie die Kartenservice-URL ein, zu der Sie die Verbindung herstellen wollen.
`map_name`	*string*	Nur, wenn `map_service_URL` verwendet wird; enthält die Ordnerstruktur auf der höchsten Ebene des Kartenservice.

Eigenschaften von "jsonimportnode"

Der JSON-Quellenknoten importiert Daten aus einer JSON-Datei.

Tabelle 53. Eigenschaften von "jsonimportnode"		
Eigenschaften von `jsonimport-node`	**Datentyp**	**Eigenschaftsbeschreibung**
`full_filename`	*string*	Der vollständige Dateiname mit Pfad.
`string_format`	*Datensätze* *Werte*	Geben Sie das Format der JSON-Zeichenfolge an. Der Standardwert ist `records`.
`auto_label`		Hinzugefügt in Version 18.2.1.1.

Eigenschaften von "sasimportnode"

Der SAS-Importknoten importiert SAS-Daten in IBM SPSS Modeler.

Beispiel

```
node = stream.create("sasimport", "My node")
node.setPropertyValue("format", "Windows")
node.setPropertyValue("full_filename", "C:/data/retail.sas7bdat")
node.setPropertyValue("member_name", "Test")
node.setPropertyValue("read_formats", False)
node.setPropertyValue("full_format_filename", "Test")
node.setPropertyValue("import_names", True)
```

Tabelle 54. Eigenschaften von "sasimportnode"		
Eigenschaften von sasimport-node	**Datentyp**	**Eigenschaftsbeschreibung**
format	Windows UNIX Transport SAS7 SAS8 SAS9	Format der zu importierenden Datei.
full_filename	*string*	Der vollständige, von Ihnen eingegebene Dateiname, einschließlich der Pfadangabe.
member_name	*string*	Geben Sie ein Mitglied an, das aus der oben angegebenen SAS-Transportdatei importiert werden soll.
read_formats	*Flag*	Liest Datenformate (wie z. B. Variablenbeschriftungen) aus der angegebenen Formatdatei.
full_format_filename	*string*	
import_names	NamesAndLabels LabelsasNames	Gibt die Methode für die Zuordnung von Variablennamen und -beschriftungen beim Import an.

simgennode, Eigenschaften

Der Simulationsgenerierungsknoten bietet eine einfache Möglichkeit, simulierte Daten entweder völlig neu anhand von durch den Benutzer angegebenen statistischen Verteilungen oder automatisch anhand der Verteilungen aus der Ausführung eines Simulationsanpassungsknotens für vorhandene historische Daten zu generieren. Dies ist nützlich, wenn Sie das Ergebnis eines Vorhersagemodells bei Unsicherheiten in den Modelleingaben bewerten möchten.

Tabelle 55. simgennode, Eigenschaften		
Eigenschaften von simgennode	**Datentyp**	**Eigenschaftsbeschreibung**
fields	Strukturierte Eigenschaft	Siehe Beispiel
correlations	Strukturierte Eigenschaft	Siehe Beispiel
keep_min_max_setting	*Boolesch*	
refit_correlations	*Boolesch*	

Tabelle 55. simgennode, Eigenschaften (Forts.)		
Eigenschaften von simgennode	**Datentyp**	**Eigenschaftsbeschreibung**
max_cases	*Integer*	Minimalwert ist 1.000, Maximalwert ist 2.147.483.647
create_iteration_field	*Boolesch*	
iteration_field_name	*string*	
replicate_results	*Boolesch*	
random_seed	*Integer*	
parameter_xml	*string*	Gibt den Parameter Xml als Zeichenfolge zurück.

Beispiel für "fields"

Hierbei handelt es sich um einen strukturierten Slotparameter mit der folgenden Syntax:

```
simgennode.setPropertyValue("fields", [
    [field1, storage, locked, [distribution1], min, max],
    [field2, storage, locked, [distribution2], min, max],
    [field3, storage, locked, [distribution3], min, max]
])
```

distribution ist eine Deklaration des Verteilungsnamens gefolgt von einer Liste, die Paare von Attributnamen und Werten enthält. Jede Verteilung ist folgendermaßen definiert:

```
[distributionname, [[par1], [par2], [par3]]]

simgennode = modeler.script.stream().createAt("simgen", u"Sim Gen", 726, 322)
simgennode.setPropertyValue("fields", [["Age", "integer", False, ["Uniform",[["min","1"],
["max","2"]]], "", ""]])
```

Sie können z. B. das folgende Script verwenden, um einen Knoten zu erstellen, der ein einzelnes Feld mit einer Binomialverteilung generiert:

```
simgen_node1 = modeler.script.stream().createAt("simgen", u"Sim Gen", 200, 200)
simgen_node1.setPropertyValue("fields", [["Education", "Real", False, ["Binomial", [["n", 32],
 ["prob", 0.7]]], "", ""]])
```

Die Binomialverteilung verarbeitet zwei Parameter: n und prob. Da die Binomialverteilung Mindest- und Maximalwerte nicht unterstützt, werden sie als leere Zeichenfolge bereitgestellt.

Anmerkung: Sie können distribution nicht direkt festlegen; Sie verwenden diese Eigenschaft zusammen mit der Eigenschaft fields.

Im folgenden Beispiel werden alle möglichen Verteilungstypen gezeigt. Beachten Sie, dass der Schwellenwert als thresh in NegativeBinomialFailures und NegativeBinomialTrial eingegeben wird.

```
stream = modeler.script.stream()

simgennode = stream.createAt("simgen", u"Sim Gen", 200, 200)

beta_dist = ["Field1", "Real", False, ["Beta",[["shape1","1"],["shape2","2"]]], "", ""]
binomial_dist = ["Field2", "Real", False, ["Binomial",[["n" ,"1"],["prob","1"]]], "", ""]
categorical_dist = ["Field3", "String", False, ["Categorical", [["A",0.3],["B",0.5],["C",0.2]]], "", ""]
dice_dist = ["Field4", "Real", False, ["Dice", [["1" ,"0.5"],["2","0.5"]]], "", ""]
exponential_dist = ["Field5", "Real", False, ["Exponential", [["scale","1"]]], "", ""]
fixed_dist = ["Field6", "Real", False, ["Fixed", [["value","1" ]]], "", ""]
gamma_dist = ["Field7", "Real", False, ["Gamma", [["scale","1"],["shape"," 1"]]], "", ""]
lognormal_dist = ["Field8", "Real", False, ["Lognormal", [["a","1"],["b","1" ]]], "", ""]
negbinomialfailures_dist = ["Field9", "Real", False, ["NegativeBinomialFailures",[["prob","0.5"],["thresh","1"]]], "", ""]
negbinomialtrial_dist = ["Field10", "Real", False, ["NegativeBinomialTrials",[["prob","0.2"],["thresh","1"]]], "", ""]
normal_dist = ["Field11", "Real", False, ["Normal", [["mean","1"] ,["stddev","2"]]], "", ""]
poisson_dist = ["Field12", "Real", False, ["Poisson", [["mean","1"]]], "", ""]
range_dist = ["Field13", "Real", False, ["Range", [["BEGIN","[1,3]"] ,["END","[2,4]"],["PROB","[[0.5],[0.5]]"]]], "", ""]
triangular_dist = ["Field14", "Real", False, ["Triangular", [["min","0"],["max","1"],["mode","1"]]], "", ""]
uniform_dist = ["Field15", "Real", False, ["Uniform", [["min","1"],["max","2"]]], "", ""]
weibull_dist = ["Field16", "Real", False, ["Weibull", [["a","0"],["b","1 "],["c","1"]]], "", ""]
```

```
simgennode.setPropertyValue("fields", [\
beta_dist, \
binomial_dist, \
categorical_dist, \
dice_dist, \
exponential_dist, \
fixed_dist, \
gamma_dist, \
lognormal_dist, \
negbinomialfailures_dist, \
negbinomialtrial_dist, \
normal_dist, \
poisson_dist, \
range_dist, \
triangular_dist, \
uniform_dist, \
weibull_dist
])
```

Beispiel für "correlations"

Hierbei handelt es sich um einen strukturierten Slotparameter mit der folgenden Syntax:

```
simgennode.setPropertyValue("correlations", [
    [field1, field2, correlation],
    [field1, field3, correlation],
    [field2, field3, correlation]
])
```

Die Korrelation kann eine beliebige Zahl zwischen +1 und -1 sein. Sie können beliebig viele Korrelationen angeben. Jede nicht angegebene Korrelation wird auf Null gesetzt. Wenn Felder unbekannt sind, sollte der Korrelationswert in der Korrelationsmatrix (oder -tabelle) festgelegt werden. Er wird als roter Text angezeigt. Wenn Felder unbekannt sind, kann der Knoten nicht ausgeführt werden.

Eigenschaften von "statisticsimportnode"

Der IBM SPSS Statistics-Dateiknoten liest Daten aus dem Dateiformat *.sav* ein, das von IBM SPSS Statistics verwendet wird, sowie in IBM SPSS Modeler gespeicherte Cache-Dateien, die ebenfalls dasselbe Format verwenden.

Eine Beschreibung der Eigenschaften für diesen Knoten finden Sie in „Eigenschaften von "statisticsimportnode"" auf Seite 441.

Eigenschaften des Knotens "tm1odataimport"

Der IBM Cognos TM1-Quellenknoten importiert Daten aus Cognos TM1-Datenbanken.

Tabelle 56. tm1odataimport, Knoteneigenschaften

Eigenschaften des Knotens `tm1odataimport`	**Datentyp**	**Eigenschaftsbeschreibung**
`credential_type`	*inputCredential* oder *storedCredential*	Dient zur Angabe des Berechtigungsnachweistyps.
`input_credential`	*Liste*	Wenn `credential_type` auf `inputCredential` gesetzt ist; geben Sie die Domäne, den Benutzernamen und das Kennwort an.
`stored_credential_name`	*string*	Wenn `credential_type` `storedCredential` ist, geben Sie den Namen des Berechtigungsnachweises auf dem C&DS-Server an.

Tabelle 56. tm1odataimport, Knoteneigenschaften (Forts.)		
Eigenschaften des Knotens tm1odataimport	**Datentyp**	**Eigenschaftsbeschreibung**
selected_view	*["feld" "feld"]*	Eine Listeneigenschaft, die die Details der Eigenschaften des ausgewählten TM1-Cubes und den Namen der Cube-Ansicht enthält, aus der Daten nach SPSS importiert werden. Beispiel: `TM1_import.setPropertyValue("selected_view", ['plan_BudgetPlan', 'Goal Input'])`
is_private_view	*Flag*	Gibt an, ob `selected_view` eine private Ansicht ist. Der Standardwert ist `false`.
selected_columns	*["feld"]*	Geben Sie die ausgewählte Spalte an; nur ein Element kann angegeben werden. Beispiel: `setPropertyValue("selected_columns", ["Measures"])`
selected_rows	*["feld" "feld"]*	Geben Sie die ausgewählten Zeilen an. Beispiel: `setPropertyValue("selected_rows", ["Dimension_1_1", "Dimension_2_1", "Dimension_3_1", "Periods"])`
connection_type	`AdminServer` `TM1Server`	Gibt den Verbindungstyp an. Der Standardwert ist `AdminServer`.
admin_host	*string*	URL für den Hostnamen der REST-API. Erforderlich, wenn `connection_type` auf `AdminServer` gesetzt ist.
server_name	*string*	Name des TM1-Servers, der aus `admin_host` ausgewählt wird. Erforderlich, wenn `connection_type` auf `AdminServer` gesetzt ist.
server_url	*string*	Die URL für die REST-API von TM1 Server. Erforderlich, wenn `connection_type` auf `TM1Server` gesetzt ist.

Eigenschaften des Knotens "tm1import" (nicht mehr unterstützt)

Der IBM Cognos TM1-Quellenknoten importiert Daten aus Cognos TM1-Datenbanken.

Anmerkung: Dieser Knoten wird seit Modeler 18.0 nicht mehr unterstützt. Der Name des Ersatzknotenscripts ist *tm1odataimport*.

Tabelle 57. Eigenschaften des Knotens "tm1import"		
Eigenschaften des Knotens `tm1import`	**Datentyp**	**Eigenschaftsbeschreibung**
`pm_host`	*string*	**Anmerkung:** Nur für Version 16.0 und 17.0 Der Hostname. Zum Beispiel: `TM1_import.setPropertyValue("pm_host", 'http://9.191.86.82:9510/pmhub/pm')`
`tm1_connection`	*["feld","feld", ... ,"feld"]*	**Anmerkung:** Nur für Version 16.0 und 17.0 Eine Listeneigenschaft mit den Verbindungsdetails für den TM1-Server. Format: `[ "TM1_Server_Name","tm1_ username","tm1_ password"]` Zum Beispiel: `TM1_import.setPropertyValue("tm1_connection", ['Planning Sample', "admin", "apple"])`
`selected_view`	*["feld" "feld"]*	Eine Listeneigenschaft, die die Details der Eigenschaften des ausgewählten TM1-Cubes und den Namen der Cube-Ansicht enthält, aus der Daten nach SPSS importiert werden. Zum Beispiel: `TM1_import.setPropertyValue("selected_view", ['plan_BudgetPlan', 'Goal Input'])`
`selected_column`	*["feld"]*	Geben Sie die ausgewählte Spalte an; nur ein Element kann angegeben werden. Zum Beispiel: `setPropertyValue("selected_columns", ["Measures"])`
`selected_rows`	*["feld" "feld"]*	Geben Sie die ausgewählten Zeilen an. Zum Beispiel: `setPropertyValue("selected_rows", ["Dimension_1_1", "Dimension_2_1", "Dimension_3_1", "Periods"])`

twcimport, Knoteneigenschaften

Der TWC-Quellenknoten importiert Wetterdaten von The Weather Company, einem IBM Unternehmen. Hiermit können Sie Langzeitwetterdaten oder Wettervorhersagedaten für eine Position abrufen. Dies kann Sie dabei unterstützen, wetterbasierte Geschäftslösungen zu entwickeln, um anhand von äußerst genauen und präzisen Wetterdaten bessere Entscheidungen zu treffen.

Tabelle 58. twcimport, Knoteneigenschaften		
Eigenschaften des Knotens `twcimport`	**Datentyp**	**Eigenschaftsbeschreibung**
`TWCDataImport.latitude`	*reelle Zahl*	Gibt einen Breitengradwert im Format `[-90.0~90.0]` an.
`TWCDataImport.longitude`	*reelle Zahl*	Gibt einen Längengradwert im Format `[-180.0~180.0]` an.
`TWCDataImport.licenseKey`	*string*	Gibt den Lizenzschlüssel an, der von The Weather Company empfangen wurde.

`TWCDataImport.measurmentUnit`	`English`
	
	`Metric`
	
	`Hybrid`
		Gibt die Maßeinheit an. Mögliche Werte sind `English`, `Metric` oder `Hybrid`. `Metric` ist der Standardwert.

`TWCDataImport.dataType`	`Historical`
	
	`Forecast`
		Gibt den Typ der einzugebenden Wetterdaten an. Mögliche Werte sind `Historical` oder `Forecast`. `Historical` ist der Standardwert.

`TWCDataImport.startDate`	*integer*	Wenn `Historical` für `TWCDataImport.dataType` angegeben wurde, geben Sie ein Startdatum im Format `jjjjMMtt` an.
`TWCDataImport.endDate`	*integer*	Wenn `Historical` für `TWCDataImport.dataType` angegeben wurde, geben Sie ein Enddatum im Format `jjjjMMtt` an.

`TWCDataImport.forecastHour`	`6`
	
	`12`
	
	`24`
	
	`48`
		Wenn `Forecast` für `TWCDataImport.dataType` angegeben wurde, geben Sie 6, 12, 24 oder 48 für die Stunden an.

userinputnode, Eigenschaften

Der Benutzereingabeknoten bietet eine einfache Möglichkeit, künstliche Daten zu erstellen. Dazu können entweder neue Daten ohne Vorlage erstellt oder vorhandene Daten geändert werden. Diese Funktion ist nützlich, wenn Sie z. B. ein Testdataset für die Modellierung erstellen möchten.

Beispiel

```
node = stream.create("userinput", "My node")
node.setPropertyValue("names", ["test1", "test2"])
node.setKeyedPropertyValue("data", "test1", "2, 4, 8")
node.setKeyedPropertyValue("custom_storage", "test1", "Integer")
node.setPropertyValue("data_mode", "Ordered")
```

Tabelle 59. userinputnode, Eigenschaften		
Eigenschaften von `userinput-node`	**Datentyp**	**Eigenschaftsbeschreibung**
`data`		
`names`		Strukturierter Slot, der eine vom Knoten erstellte Liste der Feldnamen festlegt oder zurückgibt.

`custom_storage`	`Unknown`
	`String`
	`Integer`
	`Real`
	`Time`
	`Date`
	`Timestamp`
		Schlüsselslot, der den Speichertyp für ein Feld festlegt oder zurückgibt.

`data_mode`	`Combined` `Ordered`	Wenn `Combined` angegeben wird, werden Datensätze für jede Kombination aus Set-Werten und Min./Max.-Werten erstellt. Die Anzahl der erstellten Datensätze entspricht dem Produkt der Anzahl der Werte in jedem Feld. Wenn `Ordered` angegeben wird, wird zur Erstellung einer Datenzeile aus jeder Spalte für jeden Datensatz genau ein Wert abgerufen. Die Anzahl der erstellten Datensätze entspricht der höchsten Anzahl von Werten, die einem Feld zugeordnet sind. Alle Felder mit weniger Datenwerten werden mit Nullwerten aufgefüllt.
`values`		**Anmerkung:** Diese Eigenschaft wurde zugunsten von `userinputnode.data` verworfen und sollte nicht mehr verwendet werden.

Eigenschaften von "variablefilenode"

Der Knoten "Variable Datei" liest Daten aus Textdateien mit freien Feldern, also aus Dateien, deren Datensätze eine konstante Anzahl von Feldern, aber eine variable Anzahl von Zeichen enthalten. Dieser Knoten ist außerdem nützlich für Dateien mit fester Länge, Überschriftentext und bestimmten Anmerkungen.

Beispiel

```
node = stream.create("variablefile", "My node")
node.setPropertyValue("full_filename", "$CLEO_DEMOS/DRUG1n")
node.setPropertyValue("read_field_names", True)
```

```
node.setPropertyValue("delimit_other", True)
node.setPropertyValue("other", ",")
node.setPropertyValue("quotes_1", "Discard")
node.setPropertyValue("decimal_symbol", "Comma")
node.setPropertyValue("invalid_char_mode", "Replace")
node.setPropertyValue("invalid_char_replacement", "|")
node.setKeyedPropertyValue("use_custom_values", "Age", True)
node.setKeyedPropertyValue("direction", "Age", "Input")
node.setKeyedPropertyValue("type", "Age", "Range")
node.setKeyedPropertyValue("values", "Age", [1, 100])
```

Tabelle 60. Eigenschaften von "variablefilenode"

Eigenschaften von variablefilenode	**Datentyp**	**Eigenschaftsbeschreibung**
skip_header	*number*	Gibt die Anzahl der ab dem Beginn des ersten Datensatzes zu ignorierenden Zeichen an.
num_fields_auto	*Flag*	Stellt die Anzahl der Felder in jedem Datensatz automatisch fest. Datensätze müssen mit einem Zeilenwechselzeichen abgeschlossen werden.
num_fields	*number*	Legt die Anzahl der Felder in jedem Datensatz manuell fest.
delimit_space	*Flag*	Gibt an, welches Zeichen in der Datei für die Feldbegrenzungen verwendet wird.
delimit_tab	*Flag*	
delimit_new_line	*Flag*	
delimit_non_printing	*Flag*	
delimit_comma	*Flag*	Wenn das Komma sowohl als Feldtrennzeichen als auch als Dezimaltrennzeichen für Streams festgelegt ist, setzen Sie delimit_other auf *true* und legen Sie ein Komma als Trennzeichen fest, indem Sie die Eigenschaft other verwenden.
delimit_other	*Flag*	Hier können Sie ein benutzerdefiniertes Trennzeichen festlegen, indem Sie die Eigenschaft other verwenden.
other	*string*	Gibt an, welches Trennzeichen verwendet wird, wenn delimit_other auf *true* gesetzt ist.
decimal_symbol	Default Comma Period	Legt das in der Datenquelle verwendete Dezimaltrennzeichen fest.
multi_blank	*Flag*	Behandelt mehrere angrenzende leere Trennzeichen als ein einziges Trennzeichen.

Tabelle 60. Eigenschaften von "variablefilenode" (Forts.)

Eigenschaften von variablefilenode	**Datentyp**	**Eigenschaftsbeschreibung**
read_field_names	*Flag*	Behandelt die erste Zeile der Datendatei als Beschriftungen für die Spalten.
strip_spaces	None Left Right Both	Verwirft beim Importieren führende und nachfolgende Leerzeichen in Zeichenfolgen.
invalid_char_mode	Discard Replace	Entfernt ungültige Zeichen (null, 0 oder jedes nicht in der aktuellen Codierung enthaltene Zeichen) aus der Dateneingabe oder ersetzt ungültige Zeichen durch das festgelegte, aus einem Zeichen bestehende Symbol.
invalid_char_replacement	*string*	
break_case_by_newline	*Flag*	Gibt an, dass das Zeilenvorschubzeichen als Zeilenbegrenzer verwendet wird.
lines_to_scan	*number*	Gibt an, wie viele Zeilen nach angegebenen Datentypen durchsucht werden sollen.
auto_recognize_datetime	*Flag*	Gibt an, ob Datums- oder Zeitangaben automatisch in den Quellendaten identifiziert werden.
quotes_1	Discard PairAndDiscard IncludeAsText	Gibt an, wie einfache Anführungszeichen beim Import behandelt werden.
quotes_2	Discard PairAndDiscard IncludeAsText	Gibt an, wie doppelte Anführungszeichen beim Import behandelt werden.
full_filename	*string*	Vollständiger Name der zu lesenden Datei, einschließlich Pfadangabe.
use_custom_values	*Flag*	

Tabelle 60. Eigenschaften von "variablefilenode" (Forts.)

Eigenschaften von variablefilenode	**Datentyp**	**Eigenschaftsbeschreibung**
custom_storage	Unknown String Integer Real Time Date Timestamp	

Tabelle 60. Eigenschaften von "variablefilenode" (Forts.)		
Eigenschaften von variablefilenode	**Datentyp**	**Eigenschaftsbeschreibung**
custom_date_format	"DDMMYY" "MMDDYY" "YYMMDD" "YYYYMMDD" "YYYYDDD" DAY MONTH "DD-MM-YY" "DD-MM-YYYY" "MM-DD-YY" "MM-DD-YYYY" "DD-MON-YY" "DD-MON-YYYY" "YYYY-MM-DD" "DD.MM.YY" "DD.MM.YYYY" "MM.DD.YY" "MM.DD.YYYY" "DD.MON.YY" "DD.MON.YYYY"	Nur anwendbar, wenn ein benutzerdefinierter Speichertyp angegeben wurde.

Tabelle 60. Eigenschaften von "variablefilenode" (Forts.)

Eigenschaften von `variablefilenode`	Datentyp	Eigenschaftsbeschreibung
	"DD/MM/YY" "DD/MM/YYYY" "MM/DD/YY" "MM/DD/YYYY" "DD/MON/YY" "DD/MON/YYYY" MON YYYY q Q YYYY ww WK YYYY	

Tabelle 60. Eigenschaften von "variablefilenode" (Forts.)		
Eigenschaften von variablefilenode	**Datentyp**	**Eigenschaftsbeschreibung**
custom_time_format	"HHMMSS" "HHMM" "MMSS" "HH:MM:SS" "HH:MM" "MM:SS" "(H)H:(M)M:(S)S" "(H)H:(M)M" "(M)M:(S)S" "HH.MM.SS" "HH.MM" "MM.SS" "(H)H.(M)M.(S)S" "(H)H.(M)M" "(M)M.(S)S"	Nur anwendbar, wenn ein benutzerdefinierter Speichertyp angegeben wurde.
custom_decimal_symbol	*Feld*	Nur anwendbar, wenn ein benutzerdefinierter Speichertyp angegeben wurde.
encoding	StreamDefault SystemDefault "UTF-8"	Legt die Textcodierungsmethode fest.

Eigenschaften von "xmlimportnode"

Der XML-Quellenknoten importiert Daten im XML-Format in den Stream. Sie können eine einzelne Datei oder alle Dateien in einem Verzeichnis importieren. Optional können Sie eine Schemadatei angeben, aus der die XML-Struktur gelesen werden soll.

Beispiel

```
node = stream.create("xmlimport", "My node")
node.setPropertyValue("full_filename", "c:/import/ebooks.xml")
node.setPropertyValue("records", "/author/name")
```

Tabelle 61. Eigenschaften von "xmlimportnode"

Eigenschaften von xmlimport-node	**Datentyp**	**Eigenschaftsbeschreibung**
read	single directory	Liest eine einzige Datendatei (Standard) oder alle XML-Dateien in einem Verzeichnis.
recurse	*Flag*	Legt fest, ob zusätzlich XML-Dateien aus allen Unterverzeichnissen des angegebenen Verzeichnisses gelesen werden sollen.
full_filename	*string*	(erforderlich) Vollständiger Pfad und Dateiname der zu importierenden XML-Datei (bei read = single).
directory_name	*string*	(erforderlich) Vollständiger Pfad und Dateiname des Verzeichnisses, in dem sich die zu importierenden XML-Dateien befinden (falls read = directory).
full_schema_filename	*string*	Vollständiger Pfad und Dateiname der XSD- oder DTD-Datei, aus der die XML-Struktur gelesen werden soll. Wenn Sie diesen Parameter auslassen, wird die Struktur aus der XML-Quellendatei gelesen.
records	*string*	XPath-Ausdruck (z. B. /author/name), der die Datensatzgrenze definiert. Jedes Mal, wenn dieses Element in der Quellendatei gefunden wird, wird ein neuer Datensatz erstellt.
mode	read specify	Alle Daten lesen (Standard) oder festlegen, welche Objekte gelesen werden sollen.
fields		Liste der zu importierenden Objekte (Elemente und Attribute). Jedes Objekt in der Liste ist ein XPath-Ausdruck.

Kapitel 10. Eigenschaften von Datensatzoperationsknoten

appendnode, Eigenschaften

Der Anhangknoten verkettet Gruppen von Datensätzen miteinander. Er ist insbesondere nützlich für die Kombination von Datasets mit ähnlicher Struktur, aber unterschiedlichen Daten.

Beispiel

```
node = stream.create("append", "My node")
node.setPropertyValue("match_by", "Name")
node.setPropertyValue("match_case", True)
node.setPropertyValue("include_fields_from", "All")
node.setPropertyValue("create_tag_field", True)
node.setPropertyValue("tag_field_name", "Append_Flag")
```

Tabelle 62. appendnode, Eigenschaften

Eigenschaften von appendnode	Datentyp	Eigenschaftsbeschreibung
match_by	Position Name	Datasets können Sie auf der Grundlage der Position der Felder in der Hauptdatenquelle oder auf der Grundlage der Namen von Feldern der Eingabedatasets anhängen.
match_case	*Flag*	Aktiviert für die Übereinstimmung von Feldnamen die Unterscheidung zwischen Groß- und Kleinschreibung.
include_fields_from	Main All	
create_tag_field	*Flag*	
tag_field_name	*string*	

aggregatenode, Eigenschaften

Der Aggregatknoten ersetzt eine Sequenz von Eingabedatensätzen durch zusammengefasste, aggregierte Ausgabedatensätze.

Beispiel

```
node = stream.create("aggregate", "My node")
# dbnode is a configured database import node
stream.link(dbnode, node)
node.setPropertyValue("contiguous", True)
node.setPropertyValue("keys", ["Drug"])
node.setKeyedPropertyValue("aggregates", "Age", ["Sum", "Mean"])
```

```
node.setPropertyValue("inc_record_count", True)
node.setPropertyValue("count_field", "index")
node.setPropertyValue("extension", "Aggregated_")
node.setPropertyValue("add_as", "Prefix")
```

Tabelle 63. aggregatenode, Eigenschaften

Eigenschaften von aggregatenode	**Datentyp**	**Eigenschaftsbeschreibung**
keys	*Liste*	Listet Felder auf, die als Schlüssel für die Aggregation verwendet werden können. Bei den Schlüsselfeldern Sex und Region beispielsweise erhält jede eindeutige Kombination von M und F mit den Regionen N und S (vier eindeutige Kombinationen) einen aggregierten Datensatz.
contiguous	*Flag*	Wählen Sie diese Option aus, wenn Sie wissen, dass alle Datensätze mit denselben Schlüsselwerten in der Eingabe als zusammenhängende Gruppe vorliegen (z. B., wenn die Eingabe nach Schlüsselfeldern sortiert ist). Dadurch lässt sich eventuell die Leistungsfähigkeit verbessern.
aggregates		Strukturierte Eigenschaft, die die numerischen Felder auflistet, deren Werte aggregiert werden, sowie die ausgewählten Aggregationsmodi.
aggregate_exprs		Verschlüsselte Eigenschaft, die den abgeleiteten Feldnamen mit dem Aggregatausdruck verschlüsselt, der zur Berechnung verwendet wird. Beispiel: `aggregatenode.setKeyedPropertyValue ("aggregate_exprs", "Na_MAX", "MAX('Na')")`
extension	*string*	Geben Sie ein Präfix oder Suffix für doppelt aggregierte Felder an (Beispiel unten).
add_as	Suffix Prefix	
inc_record_count	*Flag*	Erstellt ein zusätzliches Feld, das angibt, wie viele Eingabedatensätze aus den einzelnen Aggregatdatensätzen aggregiert wurden.
count_field	*string*	Gibt den Namen des Felds für die Datensatzanzahl an.
allow_approximation	*boolesch*	Ermöglicht eine Approximation von Reihenfolgestatistiken, wenn eine Aggregation in Analytic Server ausgeführt wird
bin_count	*Integer*	Gibt die Anzahl bei der Approximation zu verwendender Klassen an

balancenode, Eigenschaften

Der Balancierungsknoten korrigiert Unausgewogenheiten in einem Dataset, sodass dieses eine bestimmte Bedingung erfüllt. Die Balancierungsanweisung passt den Anteil der Datensätze, bei denen eine Bedingung wahr ist, um den angegebenen Faktor an.

Beispiel

```
node = stream.create("balance", "My node")
node.setPropertyValue("training_data_only", True)
node.setPropertyValue("directives", [[1.3, "Age > 60"], [1.5, "Na > 0.5"]])
```

Tabelle 64. balancenode, Eigenschaften

Eigenschaften von balancenode	**Datentyp**	**Eigenschaftsbeschreibung**
directives		Strukturierte Eigenschaft, die für eine auf der angegebenen Zahl basierenden Gewichtung des Anteils von Feldwerten verwendet wird (siehe Beispiel unten).
training_data_only	*Flag*	Gibt an, dass nur Trainingsdaten balanciert werden sollen. Wenn im Stream kein Partitionsfeld vorhanden ist, wird diese Option ignoriert.

Diese Knoteneigenschaft besitzt folgendes Format:

[[*Zahl, Zeichenfolge*] \ [*Zahl, Zeichenfolge*] \ ... [*Zahl, Zeichenfolge*]].

Anmerkung: Wenn Zeichenfolgen (mithilfe von doppelten Anführungszeichen) in den Ausdruck eingebettet werden, muss ihnen das Escapezeichen " \ " vorangestellt werden. Das Zeichen " \ " dient außerdem als Fortsetzungszeichen, mit dem Sie die Argumente übersichtlich aufführen können.

cplexoptnode, Eigenschaften

Der Knoten "CPLEX-Optimierung" bietet die Möglichkeit zur Verwendung einer komplexen mathematisch basierten Optimierung (CPLEX) über eine OPL-Modelldatei (Optimization Programming Language). Diese Funktionalität war in IBM Analytical Decision Management verfügbar, das nicht mehr unterstützt wird. Sie können den CPLEX-Knoten jedoch auch in SPSS Modeler verwenden, ohne dass IBM Analytical Decision Management erforderlich ist.

Tabelle 65. cplexoptnode, Eigenschaften

Eigenschaften von cplexoptnode	**Datenart**	**Eigenschaftsbeschreibung**
opl_model_text	*string*	Das OPL-Scriptprogramm (Optimization Programming Language), das vom Knoten "CPLEX-Optimierung" ausgeführt wird und anschließend das Optimierungsergebnis generiert.

Tabelle 65. cplexoptnode, Eigenschaften (Forts.)		
Eigenschaften von cplexoptnode	**Datenart**	**Eigenschaftsbeschreibung**
opl_tuple_set_name	*string*	Der Name des Tupelsets im OPL-Modell, das den ankommenden Daten entspricht. Diese Angabe ist nicht erforderlich und wird in der Regel nicht über ein Script festgelegt. Sie sollte nur zum Bearbeiten von Feldzuordnungen einer ausgewählten Datenquelle verwendet werden.
data_input_map	*Liste mit strukturierten Eigenschaften*	Die Eingabefeldzuordnungen für eine Datenquelle. Diese Angabe ist nicht erforderlich und wird in der Regel nicht über ein Script festgelegt. Sie sollte nur zum Bearbeiten von Feldzuordnungen einer ausgewählten Datenquelle verwendet werden.

Tabelle 65. cplexoptnode, Eigenschaften (Forts.)

Eigenschaften von cplexoptnode	**Datenart**	**Eigenschaftsbeschreibung**
`md_data_input_map`	*Liste mit strukturierten Eigenschaften*	Die Feldzuordnungen für die einzelnen in OPL definierten Tupel, jeweils mit entsprechender Felddatenquelle (ankommende Daten). Benutzer können sie pro Datenquelle einzeln bearbeiten. Mit diesem Scriptt können Sie die Eigenschaft direkt so festlegen, dass alle Zuordnungen auf einmal festgelegt werden. Diese Einstellung wird in der Benutzerschnittstelle nicht angezeigt. Bei allen Entitäten in der Liste handelt es sich um strukturierte Daten: **Datenquellentag.** Der Tag der Datenquelle, der in der Dropdown-Liste für Datenquellen gefunden werden kann. Für `0_Products_Type` beispielsweise lautet der Tag `0`. **Datenquellenindex.** Die physische Folge (Index) der Datenquelle. Dies wird durch die Verbindungsreihenfolge bestimmt. **Quellenknoten.** Der Quellenknoten (Anmerkung) der Datenquelle. Dieser kann in der Dropdown-Liste für Datenquellen gefunden werden. Für `0_Products_Type` beispielsweise lautet der Quellenknoten `Products`. **Verbundener Knoten.** Der A-priori-Knoten (Anmerkung), der den aktuellen Knoten "CPLEX-Optimierung" verbindet. Dieser kann in der Dropdown-Liste für Datenquellen gefunden werden. Für `0_Products_Type` beispielsweise ist `Type` der verbundene Knoten. **Tupelmengenname.** Der Tupelmengenname der Datenquelle. Er muss mit der Definition in OPL übereinstimmen. **Tupelfeldname.** Der Name des Tupelmengenfelds der Datenquelle. Er muss mit der Definition in der OPL-Tupelmengendefinition übereinstimmen. **Speichertyp.** Der Feldspeichertyp. Mögliche Werte sind `int`, `float` oder `string`.

Tabelle 65. cplexoptnode, Eigenschaften (Forts.)

Eigenschaften von cplexoptnode	**Datenart**	**Eigenschaftsbeschreibung**
		Datenfeldname. Der Feldname der Datenquelle. **Beispiel:** `[[0,0,'Product','Type','Pro ducts','prod_id_tup','int','prod_id'] , [0,0,'Product','Type','Pro ducts','prod_name_tup','string', 'prod_name'],[1,1,'Components','Ty pe','Components', 'comp_id_tup','int','comp_id'], [1,1,'Components','Type', 'Compo nents','comp_name_tup','string','comp _name']]`
`opl_data_text`	*string*	Die Definition einiger Variablen oder Daten, die für OPL verwendet werden.
`output_value_mode`	*string*	Mögliche Werte sind `raw` oder `dvar`. Wenn `dvar`angegeben ist, muss der Benutzer auf der Registerkarte "Ausgabe" den Namen der Objektfunktionsvariable in OPL für die Ausgabe angeben. Wenn `raw` angegeben wird, wird die Zielfunktion direkt ausgegeben, unabhängig vom Namen.
`decision_variable_name`	*string*	Der Name der Zielfunktionsvariablen ist in OPL definiert. Dies ist nur aktiviert, wenn die Eigenschaft `output_value_mode` auf `dvar` gesetzt ist.
`objective_function_value_fieldname`	*string*	Der in der Ausgabe zu verwendende Feldname für den Zielfunktionswert. Der Standardwert ist `_OBJECTIVE`.

Tabelle 65. cplexoptnode, Eigenschaften (Forts.)		
Eigenschaften von cplexoptnode	**Datenart**	**Eigenschaftsbeschreibung**
`output_tuple_set_names`	*string*	Der Name der vordefinierten Tupel aus den ankommenden Daten. Dies wird als Index für die Entscheidungsvariable verwendet und es wird erwartet, dass dieser Wert mit den Variablenausgaben ausgegeben wird. Das Ausgabetupel muss mit der Entscheidungsvariablendefinition in der OPL konsistent sein. Wenn mehrere Indizes vorhanden sind, können die Tupelnamen über ein Komma verbunden werden (`,`). Ein Beispiel für ein einzelnes Tupel ist `Products`mit der entsprechenden OPL-Definition`dvar float+ Production[Products];` Ein Beispiel für mehrere Tupel ist`Products,Components`, wobei die entsprechende OPL-Definition `dvar float+ Production[Products][Components];`lautet
`decision_output_map`	*Liste mit strukturierten Eigenschaften*	Die Ausgabefelder sowie die in OPL definierten Feldzuordnung zwischen den Variablen, die ausgegeben wird. Bei allen Entitäten in der Liste handelt es sich um strukturierte Daten: **Der Variablenname.** Der Variablenamen in OPL, der ausgegeben werden soll. **Speichertyp.** Mögliche Werte sind `int`, `float` oder `string`. **Ausgabefeldname.** Der erwartete Feldname in den Ergebnissen (Ausgabe oder Export). **Beispiel:** `[['Production','int','res'],['Re mark','string','res_1']['Cost', 'float','res_2']]`

Eigenschaften von "derive_stbnode"

Der Knoten "Space-Time-Boxes" leitet Space-Time-Boxes aus den Feldern für den Breitengrad, den Längengrad und die Zeitmarke ab. Sie können auch mehrere Space-Time-Boxes als Aufenthaltsorte angeben.

Beispiel

```
node = modeler.script.stream().createAt("derive_stb", "My node", 96, 96)

# Individual Records mode
```

```
node.setPropertyValue("mode", "IndividualRecords")
node.setPropertyValue("latitude_field", "Latitude")
node.setPropertyValue("longitude_field", "Longitude")
node.setPropertyValue("timestamp_field", "OccurredAt")
node.setPropertyValue("densities", ["STB_GH7_1HOUR", "STB_GH7_30MINS"])
node.setPropertyValue("add_extension_as", "Prefix")
node.setPropertyValue("name_extension", "stb_")

# Hangouts mode
node.setPropertyValue("mode", "Hangouts")
node.setPropertyValue("hangout_density", "STB_GH7_30MINS")
node.setPropertyValue("id_field", "Event")
node.setPropertyValue("qualifying_duration", "30MINUTES")
node.setPropertyValue("min_events", 4)
node.setPropertyValue("qualifying_pct", 65)
```

Tabelle 66. Eigenschaften des Knotens "Space-Time-Boxes"

Eigenschaften von derive_stbnode	**Datentyp**	**Eigenschaftsbeschreibung**
mode	`IndividualRecords` `Hangouts`	
latitude_field	*Feld*	
longitude_field	*Feld*	
timestamp_field	*Feld*	
hangout_density	*Dichte*	Eine einfache Dichte. Gültige Dichtewerte siehe densities.

densities	[*Dichte*,*Dichte*,..., *Dichte*]	Jede Dichte ist eine Zeichenfolge, z. B. STB_GH8_1DAY.
		Anmerkung: Es gibt Einschränkungen dazu, welche Dichten gültig sind. Für den Geohash-Teil können Werte von GH1 bis GH15 verwendet werden. Für den Zeitteil können die folgenden Werte verwendet werden:
		`EVER`
		`1YEAR`
		`1MONTH`
		`1DAY`
		`12HOURS`
		`8HOURS`
		`6HOURS`
		`4HOURS`
		`3HOURS`
		`2HOURS`
		`1HOUR`
		`30MINS`
		`15MINS`
		`10MINS`
		`5MINS`
		`2MINS`
		`1MIN`
		`30SECS`
		`15SECS`
		`10SECS`
		`5SECS`
		`2SECS`
		`1SEC`

id_field	*Feld*	

Tabelle 66. Eigenschaften des Knotens "Space-Time-Boxes" (Forts.)		
Eigenschaften von derive_stbnode	**Datentyp**	**Eigenschaftsbeschreibung**
qualifying_duration	1DAY 12HOURS 8HOURS 6HOURS 4HOURS 3HOURS 2Hours 1HOUR 30MIN 15MIN 10MIN 5MIN 2MIN 1MIN 30SECS 15SECS 10SECS 5SECS 2SECS 1SECS	Muss eine Zeichenfolge sein.
min_events	*Integer*	Der gültige ganzzahlige Minimalwert ist 2.
qualifying_pct	*Integer*	Muss im Bereich von 1 bis 100 liegen.
add_extension_as	Prefix Suffix	
name_extension	*string*	

distinctnode, Eigenschaften

Der Duplikatknoten entfernt doppelte Datensätze, entweder indem jeweils der erste Datensatz an den Datenstream übergeben wird oder aber indem der erste Datensatz verworfen wird und stattdessen etwaige Duplikate an den Stream übergeben werden.

Beispiel

```
node = stream.create("distinct", "My node")
node.setPropertyValue("mode", "Include")
node.setPropertyValue("fields", ["Age" "Sex"])
node.setPropertyValue("keys_pre_sorted", True)
```

Tabelle 67. distinctnode, Eigenschaften		
Eigenschaften von distinctnode	**Datentyp**	**Eigenschaftsbeschreibung**
mode	Include Discard	Duplikatknoten entfernen doppelte Datensätze, entweder indem jeweils der erste Datensatz an den Datenstream übergeben wird oder aber indem der erste Datensatz verworfen wird und stattdessen etwaige Duplikate an den Stream übergeben werden.

Tabelle 67. distinctnode, Eigenschaften (Forts.)		
Eigenschaften von distinctnode	**Datentyp**	**Eigenschaftsbeschreibung**
grouping_fields	*Liste*	Listet die Felder auf, die verwendet werden, um zu bestimmen, ob die Datensätze identisch sind. **Anmerkung:** Diese Eigenschaft wird ab IBM SPSS Modeler 16 nicht mehr unterstützt.
composite_value	Strukturierter Slot	Siehe unten stehendes Beispiel.
composite_values	Strukturierter Slot	Siehe unten stehendes Beispiel.
inc_record_count	*Flag*	Erstellt ein zusätzliches Feld, das angibt, wie viele Eingabedatensätze aus den einzelnen Aggregatdatensätzen aggregiert wurden.
count_field	*string*	Gibt den Namen des Felds für die Datensatzanzahl an.
sort_keys	Strukturierte Eigenschaft	**Anmerkung:** Diese Eigenschaft wird ab IBM SPSS Modeler 16 nicht mehr unterstützt.
default_ascending	*Flag*	
low_distinct_key_count	*Flag*	Gibt an, dass Sie nur über eine kleine Anzahl an Datensätzen und/oder eine kleine Anzahl an eindeutigen Werten der Schlüsselfelder verfügen.
keys_pre_sorted	*Flag*	Gibt an, dass alle Datensätze mit denselben Schlüsselwerten in der Eingabe zusammengefasst werden.
disable_sql_generation	*Flag*	

Beispiel für die Eigenschaft composite_value

Die Eigenschaft composite_value hat das folgende allgemeine Format:

```
node.setKeyedPropertyValue("composite_value", FIELD, FILLOPTION)
```

FILLOPTION hat das Format [FillType, Option1, Option2, ...].

Beispiele:

```
node.setKeyedPropertyValue("composite_value", "Age", ["First"])
node.setKeyedPropertyValue("composite_value", "Age", ["last"])
node.setKeyedPropertyValue("composite_value", "Age", ["Total"])
node.setKeyedPropertyValue("composite_value", "Age", ["Average"])
node.setKeyedPropertyValue("composite_value", "Age", ["Min"])
node.setKeyedPropertyValue("composite_value", "Age", ["Max"])
node.setKeyedPropertyValue("composite_value", "Date", ["Earliest"])
node.setKeyedPropertyValue("composite_value", "Date", ["Latest"])
node.setKeyedPropertyValue("composite_value", "Code", ["FirstAlpha"])
node.setKeyedPropertyValue("composite_value", "Code", ["LastAlpha"])
```

Für die benutzerdefinierten Optionen sind mehrere Argumente erforderlich, die als Liste hinzugefügt werden. Beispiel:

```
node.setKeyedPropertyValue("composite_value", "Name", ["MostFrequent", "FirstRecord"])
node.setKeyedPropertyValue("composite_value", "Date", ["LeastFrequent", "LastRecord"])
node.setKeyedPropertyValue("composite_value", "Pending", ["IncludesValue", "T", "F"])
```

```
node.setKeyedPropertyValue("composite_value", "Marital", ["FirstMatch", "Married", "Divorced",
"Separated"])
node.setKeyedPropertyValue("composite_value", "Code", ["Concatenate"])
node.setKeyedPropertyValue("composite_value", "Code", ["Concatenate", "Space"])
node.setKeyedPropertyValue("composite_value", "Code", ["Concatenate", "Comma"])
node.setKeyedPropertyValue("composite_value", "Code", ["Concatenate", "UnderScore"])
```

Beispiel für die Eigenschaft composite_values

Die Eigenschaft composite_values hat das folgende allgemeine Format:

```
node.setPropertyValue("composite_values", [
                        [FIELD1, [FILLOPTION1]],
                        [FIELD2, [FILLOPTION2]],
.
.
])
```

Beispiel:

```
node.setPropertyValue("composite_values", [
    ["Age", ["First"]],
    ["Name", ["MostFrequent", "First"]],
    ["Pending", ["IncludesValue", "T"]],
    ["Marital", ["FirstMatch", "Married", "Divorced", "Separated"]],
    ["Code", ["Concatenate", "Comma"]]
])
```

extensionprocessnode, Eigenschaften

Mit dem Erweiterungstransformationsknoten können Sie Daten aus einem Datenstrom verwenden und mithilfe von Scripting in R oder Python for Spark Transformationen auf die Daten anwenden.

Beispiel für Python for Spark

```
#### script example for Python for Spark
import modeler.api
stream = modeler.script.stream()
node = stream.create("extension_process", "extension_process")
node.setPropertyValue("syntax_type", "Python")

process_script = """
import spss.pyspark.runtime
from pyspark.sql.types import *

cxt = spss.pyspark.runtime.getContext()

if cxt.isComputeDataModelOnly():
    _schema = StructType([StructField("Age", LongType(), nullable=True), \
                      StructField("Sex", StringType(), nullable=True), \
                      StructField("BP", StringType(), nullable=True), \
                      StructField("Na", DoubleType(), nullable=True), \
                      StructField("K", DoubleType(), nullable=True), \
                      StructField("Drug", StringType(), nullable=True)])
    cxt.setSparkOutputSchema(_schema)
else:
    df = cxt.getSparkInputData()
    print df.dtypes[:]
    _newDF = df.select("Age","Sex","BP","Na","K","Drug")
    print _newDF.dtypes[:]
    cxt.setSparkOutputData(_newDF)
"""

node.setPropertyValue("python_syntax", process_script)
```

Beispiel für R

```
#### script example for R
node.setPropertyValue("syntax_type", "R")
node.setPropertyValue("r_syntax", """day<-as.Date(modelerData$dob, format="%Y-%m-%d")
next_day<-day + 1
modelerData<-cbind(modelerData,next_day)
var1<-c(fieldName="Next day",fieldLabel="",fieldStorage="date",fieldMeasure="",fieldFormat="",
fieldRole="")
modelerDataModel<-data.frame(modelerDataModel,var1)""")
```

Tabelle 68. extensionprocessnode, Eigenschaften

Eigenschaften von extensionprocessnode	Datentyp	Eigenschaftsbeschreibung
syntax_type	*R* *Python*	Gibt das Script an, das ausgeführt wird - R oder Python (R ist der Standardwert).
r_syntax	*string*	Die R-Scriptsyntax für die Ausführung.
python_syntax	*string*	Die Python-Scriptsyntax für die Ausführung.
use_batch_size	*Flag*	Aktiviert die Verwendung der Stapelverarbeitung.
batch_size	*Integer*	Geben Sie die Anzahl der Datensätze an, die in die einzelnen Stapel eingeschlossen werden.
convert_flags	`StringsAndDoubles` `LogicalValues`	Option zum Konvertieren von Flagfeldern.
convert_missing	*Flag*	Option zum Konvertieren fehlender Werte in den R-Wert NA .
convert_datetime	*Flag*	Option zum Konvertieren von Variablen mit Datums- oder Datums-/Zeitformaten in R-Datums-/Zeitformate.
convert_datetime_class	`POSIXct` `POSIXlt`	Optionen, die angeben, in welches Format Variablen mit Datums- oder Datums-/Zeitformaten konvertiert werden.

mergenode, Eigenschaften

Der Zusammenführungsknoten erstellt aus mehreren Eingabedatensätzen einen einzelnen Ausgabedatensatz mit einigen oder allen der Eingabefelder. Er wird zum Zusammenführen von Daten aus verschiedenen Quellen verwendet, beispielsweise Daten über Auslandskunden und erworbene demografische Daten.

Beispiel

```
node = stream.create("merge", "My node")
# assume customerdata and salesdata are configured database import nodes
stream.link(customerdata, node)
stream.link(salesdata, node)
node.setPropertyValue("method", "Keys")
node.setPropertyValue("key_fields", ["id"])
node.setPropertyValue("common_keys", True)
node.setPropertyValue("join", "PartialOuter")
node.setKeyedPropertyValue("outer_join_tag", "2", True)
```

```
node.setKeyedPropertyValue("outer_join_tag", "4", True)
node.setPropertyValue("single_large_input", True)
node.setPropertyValue("single_large_input_tag", "2")
node.setPropertyValue("use_existing_sort_keys", True)
node.setPropertyValue("existing_sort_keys", [["id", "Ascending"]])
```

Tabelle 69. mergenode, Eigenschaften

Eigenschaften von `mergenode`	**Datentyp**	**Eigenschaftsbeschreibung**

`method`	`Order`
	
	`Keys`
	
	`Condition`
	
	`Rankedcondition`
		Gibt an, ob die Datensätze in der Reihenfolge zusammengeführt werden sollen, in der sie in den Datendateien aufgeführt sind, ob eines oder mehrere Felder verwendet werden sollen, um Datensätze mit demselben Wert in den Schlüsselfeldern zusammenzuführen, ob die Datensätze zusammengeführt werden, wenn eine bestimmte Bedingung erfüllt ist, oder ob jede Zeilenpaarung im primären und allen sekundären Datasets zusammengeführt werden soll. Der Rangfolgeausdruck wird verwendet, um mehrere Übereinstimmungen von niedrig nach hoch zu sortieren.

`condition`	*string*	Wenn `method` auf `Condition` gesetzt ist, wird hier die Bedingung für das Einschließen oder Verwerfen von Datensätzen angegeben.
`key_fields`	*Liste*	
`common_keys`	*Flag*	

`join`	`Inner`
	
	`FullOuter`
	
	`PartialOuter`
	
	`Anti`	

`outer_join_tag.n`	*Flag*	Bei dieser Eigenschaft ist *n* der im Dialogfeld für die Datasetauswahl angezeigte Tagname. Beachten Sie, dass mehrere Tagnamen angegeben werden können, da jede beliebige Zahl von Datasets unvollständige Datensätze beitragen könnte.
`single_large_input`	*Flag*	Gibt an, ob die Optimierung verwendet werden soll, wenn eine Eingabe vorhanden ist, die im Vergleich mit den anderen Eingaben relativ groß ist.

Tabelle 69. mergenode, Eigenschaften (Forts.)		
Eigenschaften von `mergenode`	**Datentyp**	**Eigenschaftsbeschreibung**
`single_large_input_tag`	*string*	Geben Sie den Tagnamen an, der im Dialogfeld "Großes Dataset auswählen" angezeigt wird. Beachten Sie, dass die Verwendung dieser Eigenschaft leicht von der Eigenschaft `outer_join_tag` abweicht (Flag gegenüber Zeichenfolge), da nur ein einziges Eingabedataset angegeben werden kann.
`use_existing_sort_keys`	*Flag*	Gibt an, ob die Eingaben bereits nach einem oder mehreren Schlüsselfeldern sortiert sind.
`existing_sort_keys`	[[*'Zeichenfolge',* `'Ascending'`] \ [*'Zeichenfolge',* `'Descending'`]]	Gibt die bereits sortieren Felder und ihre Sortierrichtung an.
`primary_dataset`	*string*	Wenn `method` auf `Rankedcondition` gesetzt ist, wählen Sie die Primärdatei in der Zusammenführung aus. Dies kann als die linke Seite einer Outer Join-Zusammenführung angesehen werden.
`rename_duplicate_fields`	*boolesch*	Wenn `method` auf `Rankedcondition` gesetzt und dafür Y festgelegt ist und wenn das resultierende zusammengeführte Dataset mehrere gleichnamige Felder aus unterschiedlichen Datenquellen enthält, werden die entsprechenden Tags aus den Datenquellen am Anfang der Feldspaltenheader hinzugefügt.
`merge_condition`	*string*	
`ranking_expression`	*string*	
`Num_matches`	*Integer*	Die Anzahl zurückzugebender Übereinstimmungen basierend auf `merge_condition` und `ranking_expression`. Minimum: 1, Maximum: 100.

rfmaggregatenode, Eigenschaften

Mit dem Knoten "RFM-Aggregat" (Recency, Frequency, Monetary Aggregate) können Sie Daten über die früheren Transaktionen von Kunden verwenden, alle nicht benötigten Daten entfernen und alle verbliebenen Transaktionsdaten zu einer einzigen Zeile zusammenfassen, die angibt, wann der betreffende Kunde zuletzt mit Ihnen in Geschäftskontakt stand (Recency - Aktualität), wie viele Transaktionen er vorgenommen hat (Frequency - Häufigkeit) und wie hoch der Gesamtwert dieser Transaktionen ist (Monetary - Geldwert).

Beispiel

```
node = stream.create("rfmaggregate", "My node")
node.setPropertyValue("relative_to", "Fixed")
node.setPropertyValue("reference_date", "2007-10-12")
```

```
node.setPropertyValue("id_field", "CardID")
node.setPropertyValue("date_field", "Date")
node.setPropertyValue("value_field", "Amount")
node.setPropertyValue("only_recent_transactions", True)
node.setPropertyValue("transaction_date_after", "2000-10-01")
```

Tabelle 70. rfmaggregatenode, Eigenschaften

Eigenschaften von `rfmaggregatenode`	**Datentyp**	**Eigenschaftsbeschreibung**

`relative_to`	`Fixed`
	
	`Today`
		Dient zur Angabe des Datums, ausgehend von dem die Aktualität der Transaktionen berechnet werden soll.

`reference_date`	*Datum*	Nur verfügbar, wenn unter `relative_to` die Option `Fixed` festgelegt wurde.
`contiguous`	*Flag*	Wenn die Daten vorsortiert sind, sodass alle Datensätze mit derselben ID zusammen im Datenstream erscheinen, können Sie mit dieser Option die Verarbeitung beschleunigen.
`id_field`	*Feld*	Dient zur Angabe des für die Identifizierung des Kunden und seiner Transaktionen zu verwendenden Felds.
`date_field`	*Feld*	Dient zur Angabe des Datumsfelds, das für die Berechnung der Aktualität verwendet werden soll.
`value_field`	*Feld*	Dient zur Angabe des Felds, das für die Berechnung der Geldwerts verwendet werden soll.
`extension`	*string*	Geben Sie ein Präfix oder Suffix für doppelt aggregierte Felder an.

`add_as`	`Suffix`
	
	`Prefix`
		Gibt an, ob die Erweiterung (`extension`) als Suffix oder als Präfix hinzugefügt werden soll.

`discard_low_value_records`	*Flag*	Ermöglicht die Verwendung der Einstellung `discard_records_below`.
`discard_records_below`	*number*	Dient zur Angabe eines Mindestwerts für die bei der Berechnung der RFM-Gesamtwerte verwendeten Transaktionsdetails. Die für den Wert geltenden Einheiten beziehen sich auf das ausgewählte Feld `value`.
`only_recent_transactions`	*Flag*	Dient zur Aktivierung der Einstellung `specify_transaction_date` bzw. `transaction_within_last`.
`specify_transaction_date`	*Flag*	

Tabelle 70. rfmaggregatenode, Eigenschaften (Forts.)		
Eigenschaften von rfmaggregatenode	**Datentyp**	**Eigenschaftsbeschreibung**
transaction_date_after	*Datum*	Nur verfügbar, wenn specify_transaction_date ausgewählt wurde. Dient zur Angabe des Transaktionsdatums, nach dem die Datensätze in die Analyse aufgenommen werden sollen.
transaction_within_last	*number*	Nur verfügbar, wenn transaction_within_last ausgewählt wurde. Hier können Sie anhand von Anzahl und Typ der Zeiträume (Tage, Wochen, Monate oder Jahre) angeben, wie weit die in die Analyse aufzunehmenden Datensätze ausgehend von "Aktualität (Recency) berechnen relativ zu" zurückliegen dürfen.
transaction_scale	Days Weeks Months Years	Nur verfügbar, wenn transaction_within_last ausgewählt wurde. Hier können Sie anhand von Anzahl und Typ der Zeiträume (Tage, Wochen, Monate oder Jahre) angeben, wie weit die in die Analyse aufzunehmenden Datensätze ausgehend von "Aktualität (Recency) berechnen relativ zu" zurückliegen dürfen.
save_r2	*Flag*	Zeigt für jeden Kunden das Datum der zweitaktuellsten Transaktion an.
save_r3	*Flag*	Nur verfügbar, wenn save_r2 ausgewählt wurde. Zeigt für jeden Kunden das Datum der drittaktuellsten Transaktion an.

Eigenschaften von "Rprocessnode"

Mit dem Knoten "R-Transformation" können Sie Daten aus einem IBM(r) SPSS(r) Modeler-Stream beziehen und diese mit ihrem eigenen benutzerdefinierten R-Script ändern. Nachdem die Daten geändert wurden, werden sie an den Stream zurückgegeben.

Beispiel

```
node = stream.create("rprocess", "My node")
node.setPropertyValue("custom_name", "my_node")
node.setPropertyValue("syntax", """day<-as.Date(modelerData$dob, format="%Y-
%m-%d")
next_day<-day + 1
modelerData<-cbind(modelerData,next_day)
var1<-c(fieldName="Next day",fieldLabel="",fieldStorage="date",fieldMeasu□
re="",fieldFormat="",
fieldRole="")
modelerDataModel<-data.frame(modelerDataModel,var1)""")
node.setPropertyValue("convert_datetime", "POSIXct")
```

Tabelle 71. Eigenschaften von "Rprocessnode"		
Eigenschaften von Rprocessnode	**Datentyp**	**Eigenschaftsbeschreibung**
syntax	*string*	
convert_flags	`StringsAndDoubles` `LogicalValues`	
convert_datetime	*Flag*	
convert_datetime_class	`POSIXct` `POSIXlt`	
convert_missing	*Flag*	
use_batch_size	*Flag*	Aktiviert die Verwendung der Stapelverarbeitung.
batch_size	*Integer*	Geben Sie die Anzahl der Datensätze an, die in die einzelnen Stapel eingeschlossen werden sollen.

samplenode, Eigenschaften

Der Stichprobenknoten wählte ein Subset der Datensätze aus. Es wird eine Vielzahl von Stichprobentypen unterstützt, darunter geschichtete, gruppierte (Clusterstichproben) und nichtzufällige (strukturierte) Stichproben. Eine Stichprobenziehung kann nützlich zur Verbesserung der Leistungsfähigkeit und zur Auswahl von verwandten Datensätzen bzw. Transaktionen für die Analyse sein.

Beispiel

```
/* Create two Sample nodes to extract
   different samples from the same data */

node = stream.create("sample", "My node")
node.setPropertyValue("method", "Simple")
node.setPropertyValue("mode", "Include")
node.setPropertyValue("sample_type", "First")
node.setPropertyValue("first_n", 500)

node = stream.create("sample", "My node")
node.setPropertyValue("method", "Complex")
node.setPropertyValue("stratify_by", ["Sex", "Cholesterol"])
node.setPropertyValue("sample_units", "Proportions")
node.setPropertyValue("sample_size_proportions", "Custom")
node.setPropertyValue("sizes_proportions", [["M", "High", "Default"], ["M",
"Normal", "Default"],
 ["F", "High", 0.3], ["F", "Normal", 0.3]])
```

Tabelle 72. samplenode, Eigenschaften		
Eigenschaften von samplenode	**Datenart**	**Eigenschaftsbeschreibung**
method	Simple Komplex	

Tabelle 72. samplenode, Eigenschaften (Forts.)

Eigenschaften von samplenode	**Datenart**	**Eigenschaftsbeschreibung**
mode	Include Discard	Einschließen oder Verwerfen von Datensätzen, die die angegebene Bedingung erfüllen.
sample_type	First OneInN RandomPct	Gibt die Methode der Stichprobenziehung an.
first_n	*Integer*	Datensätze bis zum angegebenen Abbruchpunkt werden eingeschlossen oder verworfen.
one_in_n	*number*	Jeden *n*-ten Datensatz einschließen oder verwerfen.
rand_pct	*number*	Geben Sie den Prozentsatz der einzuschließenden oder zu verwerfenden Datensätze an.
use_max_size	*Flag*	Aktivieren Sie die Verwendung der Einstellung maximum_size.
maximum_size	*Integer*	Geben Sie die größte Stichprobe an, die in den Datenstream eingeschlossen oder verworfen werden soll. Diese Option ist redundant und wird daher inaktiviert, wenn First und Include angegeben werden.
set_random_seed	*Flag*	Aktiviert die Verwendung der Einstellung für den Zufallsstartwert.
random_seed	*Integer*	Geben Sie den Wert an, der als Startwert für den Zufallsgenerator verwendet wird.
complex_sample_type	Random Systematic	
sample_units	Proportions Counts	
sample_size_proportions	Fixed Custom Variable	
sample_size_counts	Fixed Custom Variable	

Tabelle 72. samplenode, Eigenschaften (Forts.)		
Eigenschaften von samplenode	**Datenart**	**Eigenschaftsbeschreibung**
fixed_proportions	*number*	
fixed_counts	*Integer*	
variable_proportions	*Feld*	
variable_counts	*Feld*	
use_min_stratum_size	*Flag*	
minimum_stratum_size	*Integer*	Diese Option gilt nur, wenn eine komplexe Stichprobe mit Sample units=Proportions entnommen wird.
use_max_stratum_size	*Flag*	
maximum_stratum_size	*Integer*	Diese Option gilt nur, wenn eine komplexe Stichprobe mit Sample units=Proportions entnommen wird.
clusters	*Feld*	
stratify_by	*[field1 ... Feldn]*	
specify_input_weight	*Flag*	
input_weight	*Feld*	
new_output_weight	*string*	
sizes_proportions	[[string *Zeichenfolgewert*][string *Zeichenfolgewert*]...]	Wenn sample_units=proportions und sample_size_proportions=Custom festgelegt wurden, wird hiermit ein Wert für jede mögliche Kombination der Werte von Schichtungsfeldern angegeben.
default_proportion	*number*	
sizes_counts	[[string *Zeichenfolgewert*][string *Zeichenfolgewert*]...]	Dient zur Angabe eines Werts für jede mögliche Kombination der Werte von Schichtungsfeldern. Die Verwendung ist ähnlich wie bei sizes_proportions, es wird jedoch statt eines Anteils eine Ganzzahl angegeben.
default_count	*number*	

selectnode, Eigenschaften

Der Auswahlknoten wählt auf der Grundlage einer bestimmten Bedingung ein Subset von Datensätzen aus einem Datenstream aus oder verwirft sie. Sie können beispielsweise die Datensätze auswählen, die zu einer bestimmten Verkaufsregion gehören.

Beispiel

```
node = stream.create("select", "My node")
node.setPropertyValue("mode", "Include")
node.setPropertyValue("condition", "Age < 18")
```

Tabelle 73. selectnode, Eigenschaften		
Eigenschaften von `selectnode`	**Datentyp**	**Eigenschaftsbeschreibung**

`mode`	`Include`
	
	`Discard`
		Definiert, ob die ausgewählten Datensätze eingeschlossen oder verworfen werden sollen.

`condition`	*string*	Bedingung für das Einschließen oder Verwerfen von Datensätzen.

sortnode, Eigenschaften

Der Sortierknoten sortiert Datensätze anhand der Werte eines oder mehrerer Felder in aufsteigender oder absteigender Reihenfolge.

Beispiel

```
node = stream.create("sort", "My node")
node.setPropertyValue("keys", [["Age", "Ascending"], ["Sex", "Descending"]])
node.setPropertyValue("default_ascending", False)
node.setPropertyValue("use_existing_keys", True)
node.setPropertyValue("existing_keys", [["Age", "Ascending"]])
```

Tabelle 74. sortnode, Eigenschaften		
Eigenschaften von `sortnode`	**Datentyp**	**Eigenschaftsbeschreibung**
`keys`	*Liste*	Gibt die Felder an, nach denen sortiert werden soll. Wenn keine Richtung angegeben ist, wird der Standard verwendet.
`default_ascending`	*Flag*	Gibt die Standardsortierreihenfolge an.
`use_existing_keys`	*Flag*	Gibt an, ob die Sortierung durch Verwendung der vorherigen Sortierreihenfolge für bereits sortierte Felder optimiert werden soll.
`existing_keys`		Gibt die bereits sortieren Felder und ihre Sortierrichtung an. Verwendet dasselbe Format wie die Eigenschaft `keys`.

spacetimeboxes, Eigenschaften

Space-Time-Boxes (STB) sind eine Erweiterung der räumlichen Positionen in einer Geohashtabelle. Genauer ist eine STB eine alphanumerische Zeichenfolge, die einen regelmäßig geformten Bereich von Raum und Zeit darstellt.

Tabelle 75. spacetimeboxes, Eigenschaften		
Eigenschaften von spacetimeboxes	**Datentyp**	**Eigenschaftsbeschreibung**
mode	*IndividualRecords* *Hangouts*	
latitude_field	*Feld*	
longitude_field	*Feld*	
timestamp_field	*Feld*	

densities	*[Dichte, Dichte, Dichte...]*	Jede Dichte ist eine Zeichenfolge. Beispiel: STB_GH8_1DAY
		Hinweis: Es gibt Einschränkungen dazu, welche Dichten gültig sind.
		Für Geohash-Werte kann "GH1" bis "GH15" verwendet werden.
		Für den Zeitteil können die folgenden Werte verwendet werden:
		`EVER`
		`1YEAR`
		`1MONTH`
		`1DAY`
		`12HOURS`
		`8HOURS`
		`6HOURS`
		`4HOURS`
		`3HOURS`
		`2HOURS`
		`1HOUR`
		`30MINS`
		`15MINS`
		`10MINS`
		`5MINS`
		`2 MINS`
		`1 MIN`
		`30SECS`
		`15SECS`
		`10SECS`
		`5 SECS`
		`2 SECS`
		`1SEC`

field_name_extension	*string*	
add_extension_as	*Präfix* *Suffix*	
hangout_density	*Dichte*	Einfache Dichte (siehe oben)
id_field	*Feld*	

Tabelle 75. spacetimeboxes, Eigenschaften (Forts.)		
Eigenschaften von spacetimeboxes	**Datentyp**	**Eigenschaftsbeschreibung**
qualifying_duration	1DAY 12HOURS 8HOURS 6HOURS 4HOURS 2HOURS 1HOUR 30MIN 15MIN 10MIN 5MIN 2MIN 1MIN 30SECS 15SECS 10SECS 5SECS 2SECS 1SECS	Dieser Wert muss eine Zeichenfolge sein.
min_events	*Integer*	Minimalwert ist 2
qualifying_pct	*Integer*	Muss im Bereich von 1 bis 100 liegen.

Eigenschaften von "streamingtimeseries"

Der Streaming-Zeitreihenknoten erstellt und scort Zeitreihenmodelle in einem Schritt.

Anmerkung: Dieser Streaming-Zeitreihenknoten ersetzt den ursprünglichen Streaming-ZR-Knoten, der in SPSS Modeler Version 18 nicht mehr unterstützt wird.

Tabelle 76. Eigenschaften von "streamingtimeseries"		
Eigenschaften von streamingtimeseries	**Werte**	**Eigenschaftsbeschreibung**
targets	*Feld*	Der Streaming-Zeitreihenknoten sagt mindestens ein Ziel voraus; optional können dabei ein oder mehrere Eingabefelder als Prädiktoren verwendet werden. Häufigkeits- und Gewichtungsfelder werden nicht verwendet. Weitere Informationen finden Sie im Thema „Allgemeine Eigenschaften von Modellierungsknoten" auf Seite 225.
candidate_inputs	[*field1 ... FeldN]*	Im Modell verwendete Eingabe- bzw. Prädiktorfelder.
use_period	*Flag*	

Tabelle 76. Eigenschaften von "streamingtimeseries" (Forts.)		
Eigenschaften von `streamingtimeseries`	**Werte**	**Eigenschaftsbeschreibung**
`date_time_field`	*Feld*	

`input_interval`	`None`
	`Unknown`
	`Year`
	`Quarter`
	`Month`
	`Week`
	`Day`
	`Hour`
	`Hour_nonperiod`
	`Minute`
	`Minute_nonperiod`
	`Second`
	`Second_nonperiod`	

`period_field`	*Feld*	
`period_start_value`	*Integer*	
`num_days_per_week`	*Integer*	

`start_day_of_week`	`Sunday`
	`Monday`
	`Tuesday`
	`Wednesday`
	`Thursday`
	`Friday`
	`Saturday`	

`num_hours_per_day`	*Integer*	

Tabelle 76. Eigenschaften von "streamingtimeseries" (Forts.)		
Eigenschaften von `streamingtimeseries`	**Werte**	**Eigenschaftsbeschreibung**
`start_hour_of_day`	*Integer*	
`timestamp_increments`	*Integer*	
`cyclic_increments`	*Integer*	
`cyclic_periods`	*Liste*	

`output_interval`	`None`
	`Year`
	`Quarter`
	`Month`
	`Week`
	`Day`
	`Hour`
	`Minute`
	`Second`	

`is_same_interval`	*Flag*	
`cross_hour`	*Flag*	
`aggregate_and_distribute`	*Liste*	

`aggregate_default`	`Mean`
	`Sum`
	`Mode`
	`Min`
	`Max`	

`distribute_default`	`Mean` `Sum`	

Tabelle 76. Eigenschaften von "streamingtimeseries" (Forts.)

Eigenschaften von `streamingtimeseries`	**Werte**	**Eigenschaftsbeschreibung**

`group_default`	`Mean`
	`Sum`
	`Mode`
	`Min`
	`Max`	

`missing_imput`	`Linear_interp`
	`Series_mean`
	`K_mean`
	`K_median`
	`Linear_trend`	

`k_span_points`	*Integer*	
`use_estimation_period`	*Flag*	
`estimation_period`	`Observations` `Times`	
`date_estimation`	*Liste*	Nur verfügbar, wenn Sie `date_time_field` verwenden
`period_estimation`	*Liste*	Nur verfügbar, wenn Sie `use_period` verwenden
`observations_type`	`Latest` `Earliest`	
`observations_num`	*Integer*	
`observations_exclude`	*Integer*	

`method`	`ExpertModeler`
	`Exsmooth`
	`Arima`	

Tabelle 76. Eigenschaften von "streamingtimeseries" (Forts.)

Eigenschaften von `streamingtimeseries`	**Werte**	**Eigenschaftsbeschreibung**

`expert_modeler_method`	`ExpertModeler`
	
	`Exsmooth`
	
	`Arima`	

`consider_seasonal`	*Flag*	
`detect_outliers`	*Flag*	
`expert_outlier_additive`	*Flag*	
`expert_outlier_level_shift`	*Flag*	
`expert_outlier_innovational`	*Flag*	
`expert_outlier_level_shift`	*Flag*	
`expert_outlier_transient`	*Flag*	
`expert_outlier_seasonal_additive`	*Flag*	
`expert_outlier_local_trend`	*Flag*	
`expert_outlier_additive_patch`	*Flag*	
`consider_newesmodels`	*Flag*	

Tabelle 76. Eigenschaften von "streamingtimeseries" (Forts.)		
Eigenschaften von `streamingtimeseries`	**Werte**	**Eigenschaftsbeschreibung**
exsmooth_model_type	Simple HoltsLinearTrend BrownsLinearTrend DampedTrend SimpleSeasonal WintersAdditive WintersMultiplicative DampedTrendAdditive DampedTrendMultiplicative MultiplicativeTrendAdditive MultiplicativeSeasonal MultiplicativeTrendMultiplicative MultiplicativeTrend	
futureValue_type_method	Compute specify	
exsmooth_transformation_type	None SquareRoot NaturalLog	
arima.p	*Integer*	
arima.d	*Integer*	
arima.q	*Integer*	
arima.sp	*Integer*	
arima.sd	*Integer*	

Tabelle 76. Eigenschaften von "streamingtimeseries" (Forts.)

Eigenschaften von `streamingtimeseries`	**Werte**	**Eigenschaftsbeschreibung**
`arima.sq`	*Integer*	

`arima_transformation_type`	`None`
	`SquareRoot`
	`NaturalLog`	

`arima_include_constant`	*Flag*	
`tf_arima.p.` *Feldname*	*Integer*	Für Transferfunktionen.
`tf_arima.d.` *Feldname*	*Integer*	Für Transferfunktionen.
`tf_arima.q.` *Feldname*	*Integer*	Für Transferfunktionen.
`tf_arima.sp.` *Feldname*	*Integer*	Für Transferfunktionen.
`tf_arima.sd.` *Feldname*	*Integer*	Für Transferfunktionen.
`tf_arima.sq.` *Feldname*	*Integer*	Für Transferfunktionen.
`tf_arima.delay.` *Feldname*	*Integer*	Für Transferfunktionen.

`tf_arima.transformation_type.` *Feldname*	`None`
	`SquareRoot`
	`NaturalLog`
		Für Transferfunktionen.

`arima_detect_outliers`	*Flag*	
`arima_outlier_additive`	*Flag*	
`arima_outlier_level_shift`	*Flag*	
`arima_outlier_innovational`	*Flag*	
`arima_outlier_transient`	*Flag*	
`arima_outlier_seasonal_additive`	*Flag*	
`arima_outlier_local_trend`	*Flag*	
`arima_outlier_additive_patch`	*Flag*	
`conf_limit_pct`	*Reelle Zahl*	
`events`	*Felder*	
`forecastperiods`	*Integer*	
`extend_records_into_future`	*Flag*	
`conf_limits`	*Flag*	
`noise_res`	*Flag*	

Eigenschaften von "streamingts" (nicht mehr unterstützt)

Anmerkung: Dieser ursprüngliche Streaming-Zeitreihenknoten wird in SPSS Modeler Version 18 nicht mehr unterstützt und durch den neuen Streaming-Zeitreihenknoten ersetzt, der für die Nutzung der Leistungsstärke von IBM SPSS Analytic Server und für die Verarbeitung großer Datenmengen konzipiert ist.

Der Streaming-ZR-Knoten erstellt und scort Zeitreihenmodelle in einem Schritt, ohne dass ein Zeitintervallknoten benötigt wird.

Beispiel

```
node = stream.create("streamingts", "My node")
node.setPropertyValue("deployment_force_rebuild", True)
node.setPropertyValue("deployment_rebuild_mode", "Count")
node.setPropertyValue("deployment_rebuild_count", 3)
node.setPropertyValue("deployment_rebuild_pct", 11)
node.setPropertyValue("deployment_rebuild_field", "Year")
```

Tabelle 77. Eigenschaften von "streamingts"

Eigenschaften von streamingts	**Datentyp**	**Eigenschaftsbeschreibung**
custom_fields	*Flag*	Bei custom_fields=false werden die Einstellungen aus einem vorgeordneten Typknoten verwendet. Bei custom_fields=true müssen targets und inputs angegeben werden.
targets	[*field1...fieldN*]	
inputs	[*field1...fieldN*]	
method	ExpertModeler Exsmooth Arima	
calculate_conf	*Flag*	
conf_limit_pct	*Reelle Zahl*	
use_time_intervals_node	*Flag*	Bei use_time_intervals_node=true werden die Einstellungen aus einem vorgeordneten Zeitintervallknoten verwendet. Bei use_time_intervals_node=false müssen interval_offset_position, interval_offset, und interval_type angegeben werden.
interval_offset_position	LastObservation LastRecord	LastObservation bezieht sich auf **Letzte gültige Beobachtung**. LastRecord bezieht sich auf **Vom letzten Datensatz rückwärts zählen**.
interval_offset	*number*	

Tabelle 77. Eigenschaften von "streamingts" (Forts.)

Eigenschaften von streamingts	**Datentyp**	**Eigenschaftsbeschreibung**

interval_type	`Periods`
	`Years`
	`Quarters`
	`Months`
	`WeeksNonPeriodic`
	`DaysNonPeriodic`
	`HoursNonPeriodic`
	`MinutesNonPeriodic`
	`SecondsNonPeriodic`	

events	*Felder*	

expert_modeler_method	`AllModels`
	`Exsmooth`
	`Arima`	

consider_seasonal	*Flag*	
detect_outliers	*Flag*	
expert_outlier_additive	*Flag*	
expert_outlier_level_shift	*Flag*	
expert_outlier_innovational	*Flag*	
expert_outlier_transient	*Flag*	
expert_outlier_seasonal_additive	*Flag*	
expert_outlier_local_trend	*Flag*	
expert_outlier_additive_patch	*Flag*	

exsmooth_model_type	`Simple`
	`HoltsLinearTrend`
	`BrownsLinearTrend`
	`DampedTrend`
	`SimpleSeasonal`
	`WintersAdditive`
	`WintersMultiplicative`	

exsmooth_transformation_type	`None`
	`SquareRoot`
	`NaturalLog`	

arima_p	*Integer*	Dieselbe Eigenschaft wie für den Zeitreihenmodellierungsknoten
arima_d	*Integer*	Dieselbe Eigenschaft wie für den Zeitreihenmodellierungsknoten
arima_q	*Integer*	Dieselbe Eigenschaft wie für den Zeitreihenmodellierungsknoten

Tabelle 77. Eigenschaften von "streamingts" (Forts.)

Eigenschaften von streamingts	**Datentyp**	**Eigenschaftsbeschreibung**
arima_sp	*Integer*	Dieselbe Eigenschaft wie für den Zeitreihenmodellierungsknoten
arima_sd	*Integer*	Dieselbe Eigenschaft wie für den Zeitreihenmodellierungsknoten
arima_sq	*Integer*	Dieselbe Eigenschaft wie für den Zeitreihenmodellierungsknoten
arima_transformation_type	None SquareRoot NaturalLog	Dieselbe Eigenschaft wie für den Zeitreihenmodellierungsknoten
arima_include_constant	*Flag*	Dieselbe Eigenschaft wie für den Zeitreihenmodellierungsknoten
tf_arima_p.*fieldname*	*Integer*	Dieselbe Eigenschaft wie für den Zeitreihenmodellierungsknoten. Für Transferfunktionen.
tf_arima_d.*fieldname*	*Integer*	Dieselbe Eigenschaft wie für den Zeitreihenmodellierungsknoten. Für Transferfunktionen.
tf_arima_q.*fieldname*	*Integer*	Dieselbe Eigenschaft wie für den Zeitreihenmodellierungsknoten. Für Transferfunktionen.
tf_arima_sp.*fieldname*	*Integer*	Dieselbe Eigenschaft wie für den Zeitreihenmodellierungsknoten. Für Transferfunktionen.
tf_arima_sd.*fieldname*	*Integer*	Dieselbe Eigenschaft wie für den Zeitreihenmodellierungsknoten. Für Transferfunktionen.
tf_arima_sq.*fieldname*	*Integer*	Dieselbe Eigenschaft wie für den Zeitreihenmodellierungsknoten. Für Transferfunktionen.
tf_arima_delay.*fieldname*	*Integer*	Dieselbe Eigenschaft wie für den Zeitreihenmodellierungsknoten. Für Transferfunktionen.
tf_arima_transformation_type. *fieldname*	None SquareRoot NaturalLog	
arima_detect_outlier_mode	None Automatic	
arima_outlier_additive	*Flag*	
arima_outlier_level_shift	*Flag*	
arima_outlier_innovational	*Flag*	

Tabelle 77. Eigenschaften von "streamingts" (Forts.)		
Eigenschaften von `streamingts`	**Datentyp**	**Eigenschaftsbeschreibung**
`arima_outlier_transient`	*Flag*	
`arima_outlier_seasonal_additive`	*Flag*	
`arima_outlier_local_trend`	*Flag*	
`arima_outlier_additive_patch`	*Flag*	
`deployment_force_rebuild`	*Flag*	
`deployment_rebuild_mode`	`Count` `Percent`	
`deployment_rebuild_count`	*number*	
`deployment_rebuild_pct`	*number*	
`deployment_rebuild_field`	*<field>*	

Kapitel 11. Eigenschaften von Feldoperationsknoten

anonymizenode, Eigenschaften

Der Anonymisierungsknoten ändert die Art und Weise, wie Feldnamen und -werte weiter unten im Stream dargestellt werden, und verschleiert damit die ursprünglichen Daten. Dies kann sinnvoll sein, wenn andere Benutzer in die Lage versetzt werden sollen, Modelle unter Verwendung vertraulicher Daten wie beispielsweise Kundennamen zu erstellen.

Beispiel

```
stream = modeler.script.stream()
varfilenode = stream.createAt("variablefile", "File", 96, 96)
varfilenode.setPropertyValue("full_filename", "$CLEO/DEMOS/DRUG1n")
node = stream.createAt("anonymize", "My node", 192, 96)
# Anonymize node requires the input fields while setting the values
stream.link(varfilenode, node)
node.setKeyedPropertyValue("enable_anonymize", "Age", True)
node.setKeyedPropertyValue("transformation", "Age", "Random")
node.setKeyedPropertyValue("set_random_seed", "Age", True)
node.setKeyedPropertyValue("random_seed", "Age", 123)
node.setKeyedPropertyValue("enable_anonymize", "Drug", True)
node.setKeyedPropertyValue("use_prefix", "Drug", True)
node.setKeyedPropertyValue("prefix", "Drug", "myprefix")
```

Tabelle 78. anonymizenode, Eigenschaften

Eigenschaften von anonymizenode	**Datentyp**	**Eigenschaftsbeschreibung**
enable_anonymize	*Flag*	Bei Festlegung auf True wird die Anonymisierung von Feldwerten aktiviert (entspricht der Auswahl von **Ja** für das betreffende Feld in der Spalte "Werte anonymisieren").
use_prefix	*Flag*	Bei Festlegung auf True wird ein benutzerdefiniertes Präfix verwendet, sofern eines angegeben wurde. Gilt für Felder, die mit der Hash-Methode anonymisiert werden, und entspricht der Auswahl des Optionsfelds **Benutzerdefiniert** im Dialogfeld "Werte ersetzen" für das betreffende Feld.
prefix	*string*	Entspricht der Eingabe eines Präfixes in das Textfeld im Dialogfeld "Werte ersetzen". Das Standardpräfix ist der Standardwert, wenn keine anderen Angaben gemacht wurden.
transformation	Random Fixed	Bestimmt, ob die Transformationsparameter für ein durch die Transformationsmethode anonymisiertes Feld zufällig oder fest sein sollen.
set_random_seed	*Flag*	Bei Festlegung auf True wird der angegebene Startwert für den Zufallsgenerator verwendet (sofern außerdem transformation auf Random gesetzt ist).

Tabelle 78. anonymizenode, Eigenschaften (Forts.)		
Eigenschaften von anonymizenode	**Datentyp**	**Eigenschaftsbeschreibung**
random_seed	*Integer*	Wenn set_random_seed auf True gesetzt ist, wird dieser Wert als Startwert für den Zufallsgenerator verwendet.
scale	*number*	Wenn transformation auf Fixed gesetzt ist, wird dieser Wert als Wert für "Skalieren um" verwendet. Der Höchstwert für die Skalierung ist normalerweise 10; er kann jedoch gesenkt werden, um einen Überlauf zu vermeiden.
translate	*number*	Wenn transformation auf Fixed gesetzt ist, wird dieser Wert als Wert für die Verschiebung ("translate") verwendet. Der Höchstwert für die Verschiebung ist normalerweise 1000; er kann jedoch gesenkt werden, um einen Überlauf zu vermeiden.

Eigenschaften von "autodataprepnode"

Der Knoten "Automated Data Preparation" (ADP) kann Ihre Daten analysieren und Korrekturen identifizieren, problematische oder vermutlich überflüssige Felder ausschließen, wie erforderlich neue Attribute ableiten und die Leistung durch intelligente Prüf- und Stichprobenverfahren verbessern. Sie können den Knoten vollständig automatisiert nutzen, damit er Korrekturen wählen und anwenden kann. Sie können die Änderungen aber auch prüfen, bevor sie durchgeführt werden, und wie gewünscht akzeptieren, ablehnen oder ändern.

Beispiel

```
node = stream.create("autodataprep", "My node")
node.setPropertyValue("objective", "Balanced")
node.setPropertyValue("excluded_fields", "Filter")
node.setPropertyValue("prepare_dates_and_times", True)
node.setPropertyValue("compute_time_until_date", True)
node.setPropertyValue("reference_date", "Today")
node.setPropertyValue("units_for_date_durations", "Automatic")
```

Tabelle 79. Eigenschaften von "autodataprepnode"		
Eigenschaften von autodataprepnode	**Datentyp**	**Eigenschaftsbeschreibung**
objective	Balanced Speed Accuracy Custom	
custom_fields	*Flag*	Bei "true" können Sie Ziel-, Eingabe- und andere Felder für den aktuellen Knoten angeben. Bei "false" werden die aktuellen Einstellungen aus einem vorgeordneten Typknoten verwendet.

Tabelle 79. Eigenschaften von "autodataprepnode" (Forts.)

Eigenschaften von autodata-prepnode	**Datentyp**	**Eigenschaftsbeschreibung**
target	*Feld*	Gibt ein einzelnes Zielfeld an.
inputs	[*field1 ... FeldN]*	Im Modell verwendete Eingabe- bzw. Prädiktorfelder.
use_frequency	*Flag*	
frequency_field	*Feld*	
use_weight	*Flag*	
weight_field	*Feld*	
excluded_fields	Filter None	
if_fields_do_not_match	StopExecution ClearAnalysis	
prepare_dates_and_times	*Flag*	Zugriff auf alle Datums- und Zeitfelder kontrollieren
compute_time_until_date	*Flag*	
reference_date	Today Fixed	
fixed_date	*Datum*	
units_for_date_durations	Automatic Fixed	
fixed_date_units	Years Months Days	
compute_time_until_time	*Flag*	
reference_time	CurrentTime Fixed	
fixed_time	*Zeit*	
units_for_time_durations	Automatic Fixed	

Tabelle 79. Eigenschaften von "autodataprepnode" (Forts.)		
Eigenschaften von autodata-prepnode	**Datentyp**	**Eigenschaftsbeschreibung**
fixed_date_units	Hours Minutes Seconds	
extract_year_from_date	*Flag*	
extract_month_from_date	*Flag*	
extract_day_from_date	*Flag*	
extract_hour_from_time	*Flag*	
extract_minute_from_time	*Flag*	
extract_second_from_time	*Flag*	
exclude_low_quality_in-puts	*Flag*	
exclude_too_many_missing	*Flag*	
maximum_percentage_mis-sing	*number*	
exclude_too_many_catego-ries	*Flag*	
maximum_number_categories	*number*	
exclude_if_large_category	*Flag*	
maximum_percentage_cate-gory	*number*	
prepare_inputs_and_target	*Flag*	
adjust_type_inputs	*Flag*	
adjust_type_target	*Flag*	
reorder_nominal_inputs	*Flag*	
reorder_nominal_target	*Flag*	
replace_outliers_inputs	*Flag*	
replace_outliers_target	*Flag*	
replace_missing_conti-nuous_inputs	*Flag*	
replace_missing_conti-nuous_target	*Flag*	
replace_missing_nomi-nal_inputs	*Flag*	
replace_missing_nomi-nal_target	*Flag*	

Tabelle 79. Eigenschaften von "autodataprepnode" (Forts.)		
Eigenschaften von autodata-prepnode	**Datentyp**	**Eigenschaftsbeschreibung**
replace_missing_ordi-nal_inputs	*Flag*	
replace_missing_ordi-nal_target	*Flag*	
maximum_values_for_ordi-nal	*number*	
minimum_values_for_conti-nuous	*number*	
outlier_cutoff_value	*number*	
outlier_method	Replace Delete	
rescale_continuous_inputs	*Flag*	
rescaling_method	MinMax ZScore	
min_max_minimum	*number*	
min_max_maximum	*number*	
z_score_final_mean	*number*	
z_score_final_sd	*number*	
rescale_continuous_target	*Flag*	
target_final_mean	*number*	
target_final_sd	*number*	
transform_select_in-put_fields	*Flag*	
maximize_associati-on_with_target	*Flag*	
p_value_for_merging	*number*	
merge_ordinal_features	*Flag*	
merge_nominal_features	*Flag*	
minimum_cases_in_category	*number*	
bin_continuous_fields	*Flag*	
p_value_for_binning	*number*	
perform_feature_selection	*Flag*	
p_value_for_selection	*number*	
perform_feature_construc-tion	*Flag*	

Tabelle 79. Eigenschaften von "autodataprepnode" (Forts.)		
Eigenschaften von autodata-prepnode	**Datentyp**	**Eigenschaftsbeschreibung**
transformed_tar-get_name_extension	*string*	
transformed_in-puts_name_extension	*string*	
constructed_featu-res_root_name	*string*	
years_duration_ name_ex-tension	*string*	
months_duration_ name_ex-tension	*string*	
days_duration_ name_ex-tension	*string*	
hours_duration_ name_ex-tension	*string*	
minutes_duration_ name_extension	*string*	
seconds_duration_ name_extension	*string*	
year_cyclical_name_exten-sion	*string*	
month_cyclical_name_ex-tension	*string*	
day_cyclical_name_exten-sion	*string*	
hour_cyclical_name_exten-sion	*string*	
minute_cyclical_name_ex-tension	*string*	
second_cyclical_name_ex-tension	*string*	

astimeintervalsnode, Eigenschaften

Verwenden Sie den Zeitintervallknoten, um Intervalle anzugeben und ein neues Zeitfeld für Schätzung oder Vorhersage abzuleiten. Die unterstützten Zeitintervalle reichen dabei von Sekunden bis hin zu Jahren.

Tabelle 80. astimeintervalsnode, Eigenschaften		
Eigenschaften von `astimeinter-valsnode`	**Datentyp**	**Eigenschaftsbeschreibung**
`time_field`	*Feld*	Kann nur ein einzelnes stetiges Feld akzeptieren. Dieses Feld wird vom Knoten als Aggregationsschlüssel für das Umwandeln des Intervalls verwendet. Wird hier ein Feld für ganze Zahlen verwendet, wird es als Zeitindex interpretiert.
`dimensions`	*[Feld1 Feld2 ... Feldn]*	Diese Felder werden zum Erstellen einzelner Zeitreihen basierend auf den Feldwerten verwendet.
`fields_to_aggregate`	*[Feld1 Feld2 ... Feldn]*	Diese Felder werden als Teil der Änderung des Zeitraums für das Zeitfeld aggregiert. Alle nicht in diese Auswahlfunktion eingeschlossenen Felder werden aus den Daten herausgefiltert, die den Knoten verlassen.

binningnode, Eigenschaften

Der Klassierknoten erstellt automatisch neue nominale Felder (Setfelder) auf der Grundlage der Werte eines oder mehrerer bestehender stetiger Felder (numerischer Bereich). Sie können beispielsweise ein stetiges Einkommensfeld in ein neues kategoriales Feld transformieren, das Einkommensgruppen als Abweichungen vom Mittelwert enthält. Nach der Erstellung von Klassen für das neue Feld können Sie einen Ableitungsknoten anhand der Trennwerte generieren.

Beispiel

```
node = stream.create("binning", "My node")
node.setPropertyValue("fields", ["Na", "K"])
node.setPropertyValue("method", "Rank")
node.setPropertyValue("fixed_width_name_extension", "_binned")
node.setPropertyValue("fixed_width_add_as", "Suffix")
node.setPropertyValue("fixed_bin_method", "Count")
node.setPropertyValue("fixed_bin_count", 10)
node.setPropertyValue("fixed_bin_width", 3.5)
node.setPropertyValue("tile10", True)
```

Tabelle 81. binningnode, Eigenschaften		
Eigenschaften von `binningnode`	**Datentyp**	**Eigenschaftsbeschreibung**
`fields`	*[field1 field2 ... Feldn]*	Stetige Felder (numerischer Bereich) mit ausstehender Transformation. Sie können mehrere Felder gleichzeitig klassieren.

Tabelle 81. binningnode, Eigenschaften (Forts.)		
Eigenschaften von binningnode	**Datentyp**	**Eigenschaftsbeschreibung**
method	FixedWidth EqualCount Rank SDev Optimal	Methode, die zur Ermittlung der Trennwerte für neue Feld-Bins (Kategorien) verwendet wird.
rcalculate_bins	Always IfNecessary	Gibt an, ob bei jeder Ausführung des Knotens die Klassen neu berechnet und die Daten in die relevante Klasse eingeordnet werden sollen oder ob Daten nur zu bestehenden Klassen und etwaig hinzugefügten neuen Klassen hinzugefügt werden sollen.
fixed_width_name_extension	*string*	Die Standarderweiterung lautet *_BIN*.
fixed_width_add_as	Suffix Prefix	Gibt an, ob die Erweiterung am Ende (Suffix) oder am Anfang (Präfix) des Feldnamens eingefügt werden soll. Die Standarderweiterung lautet *income_BIN*.
fixed_bin_method	Width Count	
fixed_bin_count	*Integer*	Gibt eine Ganzzahl an, die zur Bestimmung der Anzahl der Klassen (Kategorien) mit fester Breite für die neuen Felder verwendet wird.
fixed_bin_width	*Reelle Zahl*	Wert (ganzzahlig oder reell), der zu Berechnung der Breite der Klasse verwendet wird.
equal_count_name_ extension	*string*	Die Standarderweiterung lautet *_TILE*.
equal_count_add_as	Suffix Prefix	Gibt eine Erweiterung (Suffix oder Präfix) an, die für die mithilfe von Standard-N-Perzentilen generierten Felder verwendet wird. Die Standarderweiterung ist *_TILE* plus *N*; dabei steht *N* für die Nummer des Perzentils.
tile4	*Flag*	Generiert vier Quantilklassen, die jeweils 25 % der Fälle enthalten.
tile5	*Flag*	Generiert fünf Qintilklassen.
tile10	*Flag*	Generiert 10 Dezilklassen.
tile20	*Flag*	Generiert 20 Vingtilklassen.

Tabelle 81. binningnode, Eigenschaften (Forts.)

Eigenschaften von binningnode	Datentyp	Eigenschaftsbeschreibung
tile100	*Flag*	Generiert 100 Perzentilklassen.
use_custom_tile	*Flag*	
custom_tile_name_extension	*string*	Die Standarderweiterung lautet *_TILEN*.
custom_tile_add_as	Suffix Prefix	
custom_tile	*Integer*	
equal_count_method	RecordCount ValueSum	Die Methode RecordCount versucht, jeder Klasse eine gleich große Anzahl von Datensätzen zuzuweisen, während ValueSum Datensätze so zuweist, dass die Summe der Werte in jeder Klasse gleich groß ist.
tied_values_method	Next Current Random	Gibt an, in welche Klasse Daten mit gebundenen Werten (Werten mit Gleichstand) eingeordnet werden sollen.
rank_order	Ascending Descending	Diese Eigenschaft beinhaltet Ascending (der niedrigste Wert wird mit "1" gekennzeichnet) oder Descending (der höchste Wert wird mit "1" gekennzeichnet).
rank_add_as	Suffix Prefix	Mit dieser Option werden Rang, relativer Rang und Prozentsatzrang angewendet.
rank	*Flag*	
rank_name_extension	*string*	Die Standarderweiterung lautet *_RANK*.
rank_fractional	*Flag*	Weist Fällen Ränge zu, wobei der Wert des neuen Felds gleich dem Rang dividiert durch die Summe der Gewichtungen der nicht fehlenden Fälle ist. Relative Ränge fallen in den Bereich 0-1.
rank_fractional_name_ extension	*string*	Die Standarderweiterung lautet *_F_RANK*.
rank_pct	*Flag*	Die einzelnen Ränge werden durch die Anzahl der Datensätze mit gültigen Werten dividiert und mit 100 multipliziert. Als Prozentsatz angegebene Bruchzahlränge fallen in den Bereich 1-100.
rank_pct_name_extension	*string*	Die Standarderweiterung lautet *_P_RANK*.
sdev_name_extension	*string*	

Tabelle 81. binningnode, Eigenschaften (Forts.)		
Eigenschaften von binningnode	**Datentyp**	**Eigenschaftsbeschreibung**
sdev_add_as	Suffix Prefix	
sdev_count	One Two Three	
optimal_name_extension	*string*	Die Standarderweiterung lautet *_OPTIMAL*.
optimal_add_as	Suffix Prefix	
optimal_supervisor_field	*Feld*	Als Supervisorfeld ausgewähltes Feld, mit dem die für die Klassierung ausgewählten Felder in Bezug stehen.
optimal_merge_bins	*Flag*	Gibt an, dass alle Klassen mit kleinen Fallzahlen zu einer größeren, benachbarten Klasse hinzugefügt werden.
optimal_small_bin_threshold	*Integer*	
optimal_pre_bin	*Flag*	Gibt an, dass eine Vorklassierung des Datasets durchgeführt werden soll.
optimal_max_bins	*Integer*	Gibt eine Obergrenze an, um das Erstellen einer übermäßig großen Anzahl von Feldern zu verhindern.
optimal_lower_end_point	Inclusive Exclusive	
optimal_first_bin	Unbounded Bounded	
optimal_last_bin	Unbounded Bounded	

derivenode, Eigenschaften

Der Ableitungsknoten ändert Datenwerte oder erstellt neue Felder aus einem oder mehreren bestehenden Feldern. Er erstellt Felder des Typs "Formel", "Flag", "Nominal", "Status", "Anzahl" und "Bedingt".

Beispiel 1

```
# Create and configure a Flag Derive field node
node = stream.create("derive", "My node")
node.setPropertyValue("new_name", "DrugX_Flag")
node.setPropertyValue("result_type", "Flag")
node.setPropertyValue("flag_true", "1")
node.setPropertyValue("flag_false", "0")
node.setPropertyValue("flag_expr", "'Drug' == \"drugX\"")

# Create and configure a Conditional Derive field node
node = stream.create("derive", "My node")
node.setPropertyValue("result_type", "Conditional")
node.setPropertyValue("cond_if_cond", "@OFFSET(\"Age\", 1) = \"Age\"")
node.setPropertyValue("cond_then_expr", "(@OFFSET(\"Age\", 1) = \"Age\" ><
@INDEX")
node.setPropertyValue("cond_else_expr", "\"Age\"")
```

Beispiel 2

Dieses Script nimmt an, dass zwei numerische Spalten mit den Namen XPos und YPos vorhanden sind, die die X- und Y-Koordinaten eines Punkts (z. B. dem Ort eines Ereignisses) darstellen. Das Script erstellt einen Ableitungsknoten, der eine georäumliche Spalte aus den X- und Y-Koordinaten berechnet, die diesen Punkt in einem bestimmten Koordinatensystem darstellen:

```
stream = modeler.script.stream()
# Other stream configuration code
node = stream.createAt("derive", "Location", 192, 96)
node.setPropertyValue("new_name", "Location")
node.setPropertyValue("formula_expr", "['XPos', 'YPos']")
node.setPropertyValue("formula_type", "Geospatial")
# Now we have set the general measurement type, define the
# specifics of the geospatial object
node.setPropertyValue("geo_type", "Point")
node.setPropertyValue("has_coordinate_system", True)
node.setPropertyValue("coordinate_system", "ETRS_1989_EPSG_Arctic_zone_5-47")
```

Tabelle 82. derivenode, Eigenschaften

Eigenschaften von derivenode	**Datentyp**	**Eigenschaftsbeschreibung**
new_name	*string*	Name des neuen Felds.
mode	Single Multiple	Gibt eines oder mehrere Felder an.
fields	*Liste*	Wird nur im Modus "Multiple" (Mehrere) zur Auswahl mehrerer Felder verwendet.
name_extension	*string*	Gibt die Erweiterung für die neuen Feldnamen an.
add_as	Suffix Prefix	Fügt die Erweiterung als Präfix (am Anfang) oder als Suffix (am Ende) des Feldnamens ein.

Tabelle 82. derivenode, Eigenschaften (Forts.)

Eigenschaften von derivenode	**Datentyp**	**Eigenschaftsbeschreibung**
result_type	Formula Flag Set State Count Conditional	Die sechs Typen neuer Felder, die Sie erstellen können.
formula_expr	*string*	Ausdruck zum Berechnen eines neuen Feldwerts in einem Ableitungsknoten.
flag_expr	*string*	
flag_true	*string*	
flag_false	*string*	
set_default	*string*	
set_value_cond	*string*	Wird zur Bereitstellung der Bedingung, die einem bestimmten Wert zugeordnet ist, strukturiert.
state_on_val	*string*	Dient zur Angabe des Werts für das neue Feld, wenn die Bedingung für "On" (Ein) erfüllt ist.
state_off_val	*string*	Dient zur Angabe des Werts für das neue Feld, wenn die Bedingung für "Off" (Aus) erfüllt ist.
state_on_expression	*string*	
state_off_expression	*string*	
state_initial	On Off	Weist jedem Datensatz des neuen Felds einen Anfangswert On (Ein) oder Off (Aus) zu. Dieser Wert kann sich ändern, wenn die einzelnen Bedingungen erfüllt werden.
count_initial_val	*string*	
count_inc_condition	*string*	
count_inc_expression	*string*	
count_reset_condition	*string*	
cond_if_cond	*string*	
cond_then_expr	*string*	

Tabelle 82. derivenode, Eigenschaften (Forts.)

Eigenschaften von derivenode	**Datentyp**	**Eigenschaftsbeschreibung**
cond_else_expr	*string*	
formula_measure_type	Range / MeasureType.RANGE Discrete / MeasureType.DISCRETE Flag / MeasureType.FLAG Set / MeasureType.SET OrderedSet / MeasureType.ORDERED_SET Typeless / MeasureType.TYPELESS Collection / MeasureType.COLLECTION Geospatial / MeasureType.GEOSPATIAL	Diese Eigenschaft kann zum Definieren der dem abgeleiteten Feld zugeordneten Messung verwendet werden kann. An die Setter-Funktion kann entweder eine Zeichenfolge oder einer der MeasureType-Werte übergeben werden. Die Getter-Funktion gibt immer für die MeasureType-Werte Daten zurück.
collection_measure	Range / MeasureType.RANGE Flag / MeasureType.FLAG Set / MeasureType.SET OrderedSet / MeasureType.ORDERED_SET Typeless / MeasureType.TYPELESS	Bei Sammlungsfeldern (Listen mit einer Tiefe von 0) definiert diese Eigenschaft den Messtyp, der den zugrunde liegenden Werten zugeordnet ist.
geo_type	Point MultiPoint LineString MultiLineString Polygon MultiPolygon	Bei georäumlichen Feldern definiert diese Eigenschaft den Typ des durch dieses Feld dargestellten georäumlichen Objekts. Dies sollte konsistent mit der Listentiefe der Werte sein.

Tabelle 82. derivenode, Eigenschaften (Forts.)		
Eigenschaften von derivenode	**Datentyp**	**Eigenschaftsbeschreibung**
has_coordinate_system	*Boolesch*	Bei georäumlichen Feldern definiert diese Eigenschaft, ob dieses Feld ein Koordinatensystem hat
coordinate_system	*string*	Bei georäumlichen Feldern definiert diese Eigenschaft das Koordinatensystem für dieses Feld.

ensemblenode, Eigenschaften

Der Ensembleknoten kombiniert zwei oder mehr Modellnuggets, um genauere Vorhersagen zu erhalten, als aus einem Modell gewonnen werden können.

Beispiel

```
# Create and configure an Ensemble node
# Use this node with the models in demos\streams\pm_binaryclassifier.str
node = stream.create("ensemble", "My node")
node.setPropertyValue("ensemble_target_field", "response")
node.setPropertyValue("filter_individual_model_output", False)
node.setPropertyValue("flag_ensemble_method", "ConfidenceWeightedVoting")
node.setPropertyValue("flag_voting_tie_selection", "HighestConfidence")
```

Tabelle 83. ensemblenode, Eigenschaften		
Eigenschaften von ensemblenode	**Datentyp**	**Eigenschaftsbeschreibung**
ensemble_target_field	*Feld*	Gibt das Zielfeld für alle im Ensemble verwendeten Modelle an.
filter_individual_model_output	*Flag*	Gibt an, ob Scoring-Ergebnisse aus einzelnen Modellen unterdrückt werden sollen.
flag_ensemble_method	Voting ConfidenceWeightedVoting RawPropensityWeightedVoting AdjustedPropensityWeightedVoting HighestConfidence AverageRawPropensity AverageAdjustedPropensity	Gibt an, welche Methode für die Bestimmung des Ensemble-Score verwendet werden soll. Diese Einstellung gilt nur, wenn das ausgewählte Ziel ein Flagfeld ist.

Tabelle 83. ensemblenode, Eigenschaften (Forts.)		
Eigenschaften von ensemble-node	**Datentyp**	**Eigenschaftsbeschreibung**

`set_ensemble_method`	`Voting`
	
	`ConfidenceWeightedVoting`
	
	`HighestConfidence`
		Gibt an, welche Methode für die Bestimmung des Ensemble-Score verwendet werden soll. Diese Einstellung gilt nur, wenn das ausgewählte Ziel ein nominales Feld ist.

`flag_voting_tie_selection`	`Random`
	
	`HighestConfidence`
	
	`RawPropensity`
	
	`AdjustedPropensity`
		Wenn eine Voting-Methode ausgewählt ist, gibt diese Einstellung an, wie Gleichstände aufgelöst werden sollen. Diese Einstellung gilt nur, wenn das ausgewählte Ziel ein Flagfeld ist.

`set_voting_tie_selection`	`Random`
	
	`HighestConfidence`
		Wenn eine Voting-Methode ausgewählt ist, gibt diese Einstellung an, wie Gleichstände aufgelöst werden sollen. Diese Einstellung gilt nur, wenn das ausgewählte Ziel ein nominales Feld ist.

`calculate_standard_error`	*Flag*	Wenn das Zielfeld stetig ist, wird standardmäßig eine Standardfehlerberechnung durchgeführt, um den Unterschied zwischen den gemessenen oder geschätzten Werten und den wahren Werten zu berechnen sowie um zu zeigen, wie hoch die Übereinstimmung dieser Schätzungen war.

fillernode, Eigenschaften

Der Füllerknoten ersetzt Feldwerte und ändert den Speichertyp. Sie können auswählen, dass die Werte auf der Grundlage einer CLEM-Bedingung wie beispielsweise `@BLANK(@FIELD)` ersetzt werden sollen. Alternativ können Sie auswählen, dass alle Leerstellen oder Nullwerte mit einem bestimmten Wert ersetzt werden sollen. Ein Füllerknoten wird häufig zusammen mit einem Typknoten verwendet, um fehlende Werte zu ersetzen.

Beispiel

```
node = stream.create("filler", "My node")
node.setPropertyValue("fields", ["Age"])
node.setPropertyValue("replace_mode", "Always")
node.setPropertyValue("condition", "(\"Age\" > 60) and (\"Sex\" = \"M\"")
node.setPropertyValue("replace_with", "\"old man\"")
```

Tabelle 84. fillernode, Eigenschaften		
Eigenschaften von `fillernode`	**Datentyp**	**Eigenschaftsbeschreibung**
`fields`	*Liste*	Felder aus dem Dataset, deren Werte untersucht und ersetzt werden.

`replace_mode`	`Always`
	
	`Conditional`
	
	`Blank`
	
	`Null`
	
	`BlankAndNull`
		Sie können alle Werte, leere Werte, Nullwerte oder Werte ersetzen, die einer bestimmten Bedingung entsprechen.

`condition`	*string*	
`replace_with`	*string*	

Eigenschaften von "filternode"

Der Filterknoten filtert (verwirft) Felder, benennt Felder um und ordnet Felder von einem Quellenknoten einem anderen zu.

Beispiel:

```
node = stream.create("filter", "My node")
node.setPropertyValue("default_include", True)
node.setKeyedPropertyValue("new_name", "Drug", "Chemical")
node.setKeyedPropertyValue("include", "Drug", False)
```

Verwenden der Eigenschaft default_include. Beachten Sie, dass die Festlegung des Werts der Eigenschaft `default_include` nicht automatisch zum Ein- oder Ausschluss aller Felder führt; es wird lediglich die Standardvorgehensweise für die ausgewählten Felder festgelegt. Diese Eigenschaft entspricht in ihrer Funktion dem Klicken auf die Schaltfläche **Felder standardmäßig einschließen** im Dialogfeld des Filterknotens. Hier ein Beispiel: Angenommen, Sie führen folgendes Script aus:

```
node = modeler.script.stream().create("filter", "Filter")
node.setPropertyValue("default_include", False)
# Include these two fields in the list
for f in ["Age", "Sex"]:
    node.setKeyedPropertyValue("include", f, True)
```

Dies führt dazu, dass der Knoten die Felder Age und Sex weitergibt und alle anderen verwirft. Nachdem Sie das vorherige Script ausgeführt haben, müssen Sie nun die folgenden Zeilen zum Script hinzufügen, um zwei weitere Felder zu benennen:

```
node.setPropertyValue("default_include", False)
# Include these two fields in the list
for f in ["BP", "Na"]:
    node.setKeyedPropertyValue("include", f, True)
```

Dadurch werden dem Filter zwei weitere Felder hinzugefügt, sodass insgesamt vier Felder übergeben werden (Age, Sex, BP, Na). Anders ausgedrückt, wenn der Wert von `default_include` auf `False` (Falsch) gesetzt wird, bedeutet dies nicht, dass automatisch alle Felder zurückgesetzt werden.

Wenn sie stattdessen nun `default_include` auf `True` (Wahr) ändern (entweder mithilfe eines Scripts oder im Dialogfeld des Filterknotens, wird das Verhalten umgekehrt, sodass die vier oben aufgeführten Felder nicht aufgenommen, sondern stattdessen verworfen werden. Wenn Sie sich unsicher sind, sollten Sie ein wenig mit den Steuerelementen im Dialogfeld des Filterknotens herumexperimentieren. Dies kann Ihnen beim Verständnis dieser Interaktion helfen.

Tabelle 85. Eigenschaften von "filternode"

Eigenschaften von `filternode`	**Datentyp**	**Eigenschaftsbeschreibung**
`default_include`	*Flag*	Verschlüsselte Eigenschaft zur Angabe, ob das Standardverhalten übergeben wird oder Felder gefiltert werden: Beachten Sie, dass die Festlegung dieser Eigenschaft nicht automatisch zum Ein- oder Ausschluss aller Felder führt; es wird lediglich festgelegt, ob die ausgewählten Felder standardmäßig ein- oder ausgeschlossen werden sollen. Weitere Kommentare finden Sie im unten stehenden Beispiel:
`include`	*Flag*	Verschlüsselte Eigenschaft zum Einbeziehen und Entfernen von Feldern.
`new_name`	*string*	

historynode, Eigenschaften

Der Verlaufsknoten erstellt neue Felder mit Daten aus Feldern in vorangegangenen Datensätzen. Verlaufsknoten werden am häufigsten für sequenzielle Daten, beispielsweise Zeitreihendaten, verwendet. Vor der Verwendung eines Verlaufsknotens können Sie die Daten mithilfe eines Sortierknotens sortieren.

Beispiel

```
node = stream.create("history", "My node")
node.setPropertyValue("fields", ["Drug"])
node.setPropertyValue("offset", 1)
node.setPropertyValue("span", 3)
node.setPropertyValue("unavailable", "Discard")
node.setPropertyValue("fill_with", "undef")
```

Tabelle 86. historynode, Eigenschaften

Eigenschaften von `historynode`	**Datenart**	**Eigenschaftsbeschreibung**
`fields`	*Liste*	Felder, für die Sie einen Verlauf wollen.
`offset`	*number*	Dient zur Angabe des jüngsten Datensatzes (vor dem aktuellen Datensatz), aus dem Verlaufsfeldwerte extrahiert werden sollen.
`span`	*number*	Gibt an, aus wie vielen früheren Datensätzen Werte extrahiert werden sollen.

Tabelle 86. historynode, Eigenschaften (Forts.)		
Eigenschaften von historynode	**Datenart**	**Eigenschaftsbeschreibung**
unavailable	Discard Leave Fill	Für die Behandlung von Datensätzen, die keine Verlaufswerte besitzen, bezieht sich dies normalerweise auf die ersten Datensätze oben im Dataset, für die es keine vorangegangenen Datensätze gibt, die als Verlauf dienen könnten.
fill_with	String Number	Gibt einen Wert oder eine Zeichenfolge an, die für Datensätze verwendet werden soll, wenn kein Verlaufswert verfügbar ist.

partitionnode, Eigenschaften

Der Partitionsknoten erstellt ein Partitionsfeld, das Daten in getrennte Subsets für die Trainings-, Test- und Validierungsphase der Modellerstellung aufteilt.

Beispiel

```
node = stream.create("partition", "My node")
node.setPropertyValue("create_validation", True)
node.setPropertyValue("training_size", 33)
node.setPropertyValue("testing_size", 33)
node.setPropertyValue("validation_size", 33)
node.setPropertyValue("set_random_seed", True)
node.setPropertyValue("random_seed", 123)
node.setPropertyValue("value_mode", "System")
```

Tabelle 87. partitionnode, Eigenschaften		
Eigenschaften von partitionnode	**Datentyp**	**Eigenschaftsbeschreibung**
new_name	*string*	Der vom Knoten erstellte Name des Partitionsfelds.
create_validation	*Flag*	Gibt an, ob eine Validierungspartition erstellt werden soll.
training_size	*Integer*	Prozentsatz der Datensätze (0-100), die der Trainingspartition zugeordnet werden sollen.
testing_size	*Integer*	Prozentsatz der Datensätze (0-100), die der Testpartition zugeordnet werden sollen.
validation_size	*Integer*	Prozentsatz der Datensätze (0-100), die der Validierungspartition zugeordnet werden sollen. Wird ignoriert, wenn keine Validierungspartition erstellt wird.
training_label	*string*	Beschriftung der Trainingspartition.
testing_label	*string*	Beschriftung der Testpartition.

Tabelle 87. partitionnode, Eigenschaften (Forts.)		
Eigenschaften von partitionnode	**Datentyp**	**Eigenschaftsbeschreibung**
validation_label	*string*	Beschriftung der Validierungspartition. Wird ignoriert, wenn keine Validierungspartition erstellt wird.
value_mode	System SystemAndLabel Label	Gibt die Werte an, die für die einzelnen Partitionen in den Daten verwendet werden. Beispiel: Die Trainingsstichprobe kann durch die Systemganzzahl 1, die Beschriftung Training bzw. eine Kombination aus beiden durch 1_Training repräsentiert werden.
set_random_seed	*boolesch*	Gibt an, ob ein benutzerdefinierter Startwert für den Zufallsgenerator verwendet werden soll.
random_seed	*Integer*	Ein benutzerdefinierter Startwert für den Zufallsgenerator festlegen. Damit dieser Wert verwendet wird, muss set_random_seed auf True gesetzt sein.
enable_sql_generation	*boolesch*	Gibt an, ob SQL-Pushback für die Zuweisung von Datensätzen zu Partitionen verwendet werden soll.
unique_field		Gibt das Eingabefeld an, mit dessen Hilfe sichergestellt werden soll, dass Datensätze auf zufällige, aber wiederholbare Weise zu Partitionen zugeordnet werden. Damit dieser Wert verwendet wird, muss enable_sql_generation auf True gesetzt sein.

reclassifynode, Eigenschaften

Der Umcodierungsknoten transformiert ein Set kategorialer Werte in ein anderes. Die Umcodierung dient zur Reduzierung von Kategorien bzw. Neugruppierung von Daten für die Analyse.

Beispiel

```
node = stream.create("reclassify", "My node")
node.setPropertyValue("mode", "Multiple")
node.setPropertyValue("replace_field", True)
node.setPropertyValue("field", "Drug")
node.setPropertyValue("new_name", "Chemical")
node.setPropertyValue("fields", ["Drug", "BP"])
node.setPropertyValue("name_extension", "reclassified")
node.setPropertyValue("add_as", "Prefix")
node.setKeyedPropertyValue("reclassify", "drugA", True)
node.setPropertyValue("use_default", True)
node.setPropertyValue("default", "BrandX")
node.setPropertyValue("pick_list", ["BrandX", "Placebo", "Generic"])
```

Tabelle 88. reclassifynode, Eigenschaften		
Eigenschaften von `reclassifynode`	**Datentyp**	**Eigenschaftsbeschreibung**

`mode`	`Single`
	
	`Multiple`
		`Single` codiert die Kategorien eines einzelnen Felds um. `Multiple` aktiviert Optionen, die die Transformation von mehreren Feldern gleichzeitig erlauben.

`replace_field`	*Flag*	
`field`	*string*	Wird nur im Modus "Single" verwendet.
`new_name`	*string*	Wird nur im Modus "Single" verwendet.
`fields`	*[field1 field2 ... Feldn]*	Wird nur im Modus "Multiple" verwendet.
`name_extension`	*string*	Wird nur im Modus "Multiple" verwendet.

`add_as`	`Suffix`
	
	`Prefix`
		Wird nur im Modus "Multiple" verwendet.

`reclassify`	*string*	Strukturierte Eigenschaft für Feldwerte.
`use_default`	*Flag*	Standardwert verwenden.
`default`	*string*	Standardwert angeben.
`pick_list`	*[zeichenfolge zeichenfolge ... zeichenfolge]*	Ermöglicht einem Benutzer den Import einer Liste bekannter neuer Werte, um die Dropdown-Liste in der Tabelle zu füllen.

reordernode, Eigenschaften

Der Knoten "Felder ordnen" definiert die natürliche Reihenfolge, die bei der Anzeige der nachfolgenden Felder verwendet wird. Diese Reihenfolge betrifft die Anzeige von Feldern an unterschiedlichen Stellen, beispielsweise in Tabellen, Listen und in der Feldauswahl. Diese Operation ist nützlich, um bei der Arbeit mit umfangreichen Datasets die relevanten Felder deutlicher hervorzuheben.

Beispiel

```
node = stream.create("reorder", "My node")
node.setPropertyValue("mode", "Custom")
node.setPropertyValue("sort_by", "Storage")
node.setPropertyValue("ascending", False)
node.setPropertyValue("start_fields", ["Age", "Cholesterol"])
node.setPropertyValue("end_fields", ["Drug"])
```

Tabelle 89. reordernode, Eigenschaften		
Eigenschaften von `reordernode`	**Datentyp**	**Eigenschaftsbeschreibung**

`mode`	`Custom`
	
	`Auto`
		Sie können Werte automatisch sortieren oder eine benutzerdefinierte Reihenfolge angeben.

Tabelle 89. reordernode, Eigenschaften (Forts.)		
Eigenschaften von reordernode	**Datentyp**	**Eigenschaftsbeschreibung**
sort_by	Name Type Storage	
ascending	*Flag*	
start_fields	*[Feld1 Feld2 ... Feldn]*	Nach diesen Feldern werden neue Felder eingefügt.
end_fields	*[Feld1 Feld2 ... Feldn]*	Vor diesen Feldern werden neue Felder eingefügt.

reprojectnode, Eigenschaften

In SPSS Modeler verwenden Elemente wie die georäumlichen Funktionen von Expression Builder, der STP-Knoten (Spatio-Temporal Prediction - georäumliche temporale Vorhersage) und der Kartenvisualisierungsknoten das projizierte Koordinatensystem. Verwenden Sie den Reprojizierungsknoten, um das Koordinatensystem von importierten Daten zu ändern, die ein geografisches Koordinatensystem verwenden.

Tabelle 90. reprojectnode, Eigenschaften		
Eigenschaften von reprojectnode	**Datentyp**	**Eigenschaftsbeschreibung**
reproject_fields	*[Feld1 Feld2 ... Feldn]*	Listet alle Felder auf, die reprojiziert werden sollen.
reproject_type	Streamdefault Specify	Wählen Sie aus, wie die Felder reprojiziert werden sollen.
coordinate_system	*string*	Name des Koordinatensystems, das auf die Felder angewendet werden soll. Beispiel: set reprojectnode.coordinate_system = “WGS_1984_World_Mercator”

restructurenode, Eigenschaften

Der Knoten "Umstrukturieren" konvertiert ein nominales Feld oder ein Flagfeld in eine Gruppe von Feldern, die mit den Werten aus einem weiteren Feld ausgefüllt werden können. Beispiel: Aus einem Feld mit dem Namen *Zahlungsart* mit den Werten *Kreditkarte*, *Bar* und *EC-Karte* werden drei neue Felder erstellt (*Kreditkarte*, *Bar*, *EC-Karte*), die jeweils den Wert der tatsächlichen Zahlung enthalten können.

Beispiel

```
node = stream.create("restructure", "My node")
node.setKeyedPropertyValue("fields_from", "Drug", ["drugA", "drugX"])
```

```
node.setPropertyValue("include_field_name", True)
node.setPropertyValue("value_mode", "OtherFields")
node.setPropertyValue("value_fields", ["Age", "BP"])
```

Tabelle 91. restructurenode, Eigenschaften

Eigenschaften von `restructureno-de`	**Datentyp**	**Eigenschaftsbeschreibung**
`fields_from`	[*Kategorie Kategorie Kategorie*] `all`	
`include_field_name`	*Flag*	Gibt an, ob der Feldname im umstrukturierten Feldnamen verwendet werden soll.

`value_mode`	`OtherFields`
	
	`Flags`
		Legt den Modus für die Angabe der Werte für die umstrukturierten Felder fest. Bei `OtherFields` müssen Sie angeben, welche Felder verwendet werden sollen (siehe unten). Bei `Flags` sind die Werte numerische Flags.

`value_fields`	*Liste*	Erforderlich, wenn `value_mode` auf `OtherFields` gesetzt ist. Gibt an, welche Felder als Wertfelder verwendet werden sollen.

rfmanalysisnode, Eigenschaften

Mit dem Knoten "RFM-Analyse" (Recency-, Frequency-, Monetary-Analyse) können Sie quantitativ ermitteln, welche Kunden wahrscheinlich die besten sind, indem Sie untersuchen, wann sie zuletzt etwas von Ihnen erworben haben (Recency - Aktualität), wie häufig sie eingekauft haben (Frequency - Häufigkeit) und wie viel sie für alle Transaktionen zusammengenommen ausgegeben haben (Monetary - Geldwert).

Beispiel

```
node = stream.create("rfmanalysis", "My node")
node.setPropertyValue("recency", "Recency")
node.setPropertyValue("frequency", "Frequency")
node.setPropertyValue("monetary", "Monetary")
node.setPropertyValue("tied_values_method", "Next")
node.setPropertyValue("recalculate_bins", "IfNecessary")
node.setPropertyValue("recency_thresholds", [1, 500, 800, 1500, 2000, 2500])
```

Tabelle 92. rfmanalysisnode, Eigenschaften

Eigenschaften von `rfma-nalysisnode`	**Datentyp**	**Eigenschaftsbeschreibung**
`recency`	*Feld*	Gibt das Feld für "Recency" (Aktualität) an. Dabei kann es sich um ein Datum, eine Zeitmarke oder eine einfache Zahl handeln.
`frequency`	*Feld*	Gibt das Feld für "Frequency" (Häufigkeit) an.

Tabelle 92. rfmanalysisnode, Eigenschaften (Forts.)

Eigenschaften von `rfmanalysisnode`	**Datentyp**	**Eigenschaftsbeschreibung**
`monetary`	*Feld*	Gibt das Feld für "Monetary" (Geldwert) an.
`recency_bins`	*Integer*	Dient zur Angabe der Anzahl der zu generierenden Aktualitätsklassen.
`recency_weight`	*number*	Dient zur Angabe der Gewichtung für die Aktualitätsdaten. Der Standardwert ist 100.
`frequency_bins`	*Integer*	Dient zur Angabe der Anzahl der zu generierenden Häufigkeitsklassen.
`frequency_weight`	*number*	Dient zur Angabe der Gewichtung für die Häufigkeitsdaten. Der Standardwert ist 10.
`monetary_bins`	*Integer*	Dient zur Angabe der Anzahl der zu generierenden Klassen für den Geldwert.
`monetary_weight`	*number*	Dient zur Angabe der Gewichtung für die Geldwertdaten. Der Standardwert ist 1.

`tied_values_method`	`Next`
	
	`Current`
		Gibt an, in welche Klasse Daten mit gebundenen Werten (Werten mit Gleichstand) eingeordnet werden sollen.

`recalculate_bins`	`Always`
	
	`IfNecessary`	

`add_outliers`	*Flag*	Nur verfügbar, wenn `recalculate_bins` auf `IfNecessary` gesetzt ist. Wenn diese Einstellung festgelegt wurde, werden Datensätze, die unterhalb der untersten Klasse liegen, zur untersten Klasse hinzugefügt und Datensätze oberhalb der höchsten Klasse werden in die höchste Klasse aufgenommen.

`binned_field`	`Recency`
	
	`Frequency`
	
	`Monetary`	

`recency_thresholds`	*wert wert*	Nur verfügbar, wenn `recalculate_bins` auf `Always` gesetzt ist. Dient zur Angabe der oberen und unteren Schwellenwerte für die Aktualitätsklassen. Der obere Schwellenwert einer Klasse wird als unterer Schwellenwert der nächsten Klasse verwendet. So werden beispielsweise mit [10 30 60] zwei Klassen definiert, wobei für die erste Klasse die Schwellenwerte 10 und 30 gelten und für die zweite Klasse die Schwellenwerte 30 und 60.

Tabelle 92. rfmanalysisnode, Eigenschaften (Forts.)		
Eigenschaften von `rfmanalysisnode`	**Datentyp**	**Eigenschaftsbeschreibung**
`frequency_thresholds`	*wert wert*	Nur verfügbar, wenn `recalculate_bins` auf `Always` gesetzt ist.
`monetary_thresholds`	*wert wert*	Nur verfügbar, wenn `recalculate_bins` auf `Always` gesetzt ist.

settoflagnode, Eigenschaften

Der Dichotomknoten leitet mehrere Flagfelder auf der Grundlage der kategorialen Werte ab, die für ein oder mehrere nominale Felder definiert sind.

Beispiel

```
node = stream.create("settoflag", "My node")
node.setKeyedPropertyValue("fields_from", "Drug", ["drugA", "drugX"])
node.setPropertyValue("true_value", "1")
node.setPropertyValue("false_value", "0")
node.setPropertyValue("use_extension", True)
node.setPropertyValue("extension", "Drug_Flag")
node.setPropertyValue("add_as", "Suffix")
node.setPropertyValue("aggregate", True)
node.setPropertyValue("keys", ["Cholesterol"])
```

Tabelle 93. settoflagnode, Eigenschaften		
Eigenschaften von `settoflagnode`	**Datentyp**	**Eigenschaftsbeschreibung**
`fields_from`	[*Kategorie Kategorie Kategorie*] `all`	
`true_value`	*string*	Gibt den Wert "Wahr" an, den der Knoten zum Festlegen eines Flags verwendet. Der Standardwert ist T.
`false_value`	*string*	Gibt den Falsch-Wert an, den der Knoten zum Festlegen eines Flags verwendet. Der Standardwert ist F.
`use_extension`	*Flag*	Verwenden Sie eine Erweiterung als Suffix oder Präfix für das neue Flagfeld.
`extension`	*string*	

`add_as`	`Suffix`
	
	`Prefix`
		Gibt an, ob die Erweiterung als Suffix oder als Präfix hinzugefügt wird.

`aggregate`	*Flag*	Fasst Datensätze anhand von Schlüsselfeldern zu Gruppen zusammen. Alle in einer Gruppe vorhandenen Flagfelder werden aktiviert, wenn einer der Datensätze auf "true" gesetzt wird.

Tabelle 93. settoflagnode, Eigenschaften (Forts.)		
Eigenschaften von `settoflagnode`	**Datentyp**	**Eigenschaftsbeschreibung**
keys	*Liste*	Schlüsselfelder.

Eigenschaften von "statisticstransformnode"

Der Statistics-Transformationsknoten führt eine Auswahl von IBM SPSS Statistics-Syntaxbefehlen für Datenquellen in IBM SPSS Modeler aus. Dieser Knoten erfordert eine lizenzierte Kopie vonIBM SPSS Statistics.

Eine Beschreibung der Eigenschaften für diesen Knoten finden Sie in „Eigenschaften von "statisticstransformnode"" auf Seite 441.

Eigenschaften von "timeintervalsnode" (nicht mehr unterstützt)

Anmerkung: Dieser Knoten wird in SPSS Modeler Version 18 nicht mehr unterstützt und durch den neuen Zeitreihenknoten ersetzt.

Der Zeitintervallknoten gibt Intervalle an und erstellt (bei Bedarf) Beschriftungen für die Modellierung von Zeitreihendaten. Wenn die Werte nicht gleichmäßig verteilt sind, kann der Knoten nach Bedarf Werte auffüllen oder aggregieren, um ein gleichmäßiges Intervall zwischen den Datensätzen zu generieren.

Beispiel

```
node = stream.create("timeintervals", "My node")
node.setPropertyValue("interval_type", "SecondsPerDay")
node.setPropertyValue("days_per_week", 4)
node.setPropertyValue("week_begins_on", "Tuesday")
node.setPropertyValue("hours_per_day", 10)
node.setPropertyValue("day_begins_hour", 7)
node.setPropertyValue("day_begins_minute", 5)
node.setPropertyValue("day_begins_second", 17)
node.setPropertyValue("mode", "Label")
node.setPropertyValue("year_start", 2005)
node.setPropertyValue("month_start", "January")
node.setPropertyValue("day_start", 4)
node.setKeyedPropertyValue("pad", "AGE", "MeanOfRecentPoints")
node.setPropertyValue("agg_mode", "Specify")
node.setPropertyValue("agg_set_default", "Last")
```

Tabelle 94. Eigenschaften von "timeintervalsnode"

Eigenschaften von `timeintervalsnode`	**Datentyp**	**Eigenschaftsbeschreibung**

`interval_type`	`None`
	`Periods`
	`CyclicPeriods`
	`Years`
	`Quarters`
	`Months`
	`DaysPerWeek`
	`DaysNonPeriodic`
	`HoursPerDay`
	`HoursNonPeriodic`
	`MinutesPerDay`
	`MinutesNonPeriodic`
	`SecondsPerDay`
	`SecondsNonPeriodic`	

`mode`	`Label` `Create`	Gibt an, ob Sie die Datensätze nacheinander beschriftet werden sollen oder ob die Zeitreihe auf der Grundlage eines angegebenen Datums-, Zeitmarken- oder Zeitfelds erstellt werden soll.
`field`	*Feld*	Gibt beim Erstellen der Serie aus den Daten das Feld an, das das Datum bzw. die Uhrzeit für jeden Datensatz anzeigt.
`period_start`	*Integer*	Gibt das Startintervall für Perioden bzw. zyklische Perioden an.
`cycle_start`	*Integer*	Startzyklus für zyklische Perioden.
`year_start`	*Integer*	Bei entsprechenden Intervalltypen das Jahr, in das das erste Intervall fällt.
`quarter_start`	*Integer*	Bei entsprechenden Intervalltypen das Quartal, in das das erste Intervall fällt.

Tabelle 94. Eigenschaften von "timeintervalsnode" (Forts.)		
Eigenschaften von `timeinter-valsnode`	**Datentyp**	**Eigenschaftsbeschreibung**

`month_start`	`January`
	`February`
	`March`
	`April`
	`May`
	`June`
	`July`
	`August`
	`September`
	`October`
	`November`
	`December`	

`day_start`	*Integer*	
`hour_start`	*Integer*	
`minute_start`	*Integer*	
`second_start`	*Integer*	
`periods_per_cycle`	*Integer*	Bei zyklischen Perioden, die Anzahl innerhalb jedes Zyklus.

`fiscal_year_begins`	`January`
	`February`
	`March`
	`April`
	`May`
	`June`
	`July`
	`August`
	`September`
	`October`
	`November`
	`December`
		Gibt bei vierteljährlichen Intervallen den Monat an, in dem das Geschäftsjahr beginnt.

`week_begins_on`	`Sunday`
	`Monday`
	`Tuesday`
	`Wednesday`
	`Thursday`
	`Friday`
	`Saturday`
	`Sunday`
		Gibt bei periodischen Intervallen (Tage pro Woche, Stunden pro Tag, Minuten pro Tag und Sekunden pro Tag) den Tag an, an dem die Woche beginnt.

Tabelle 94. Eigenschaften von "timeintervalsnode" (Forts.)

Eigenschaften von `timeintervalsnode`	**Datentyp**	**Eigenschaftsbeschreibung**
`day_begins_hour`	*Integer*	Gibt bei periodischen Intervallen (Stunden pro Tag, Minuten pro Tag und Sekunden pro Tag) die Stunde an, zu der der Tag beginnt. Kann in Verbindung mit `day_begins_minute` und `day_begins_second` verwendet werden, um einen genauen Zeitpunkt anzugeben wie beispielsweise *8:05:01*. Siehe unten stehendes Anwendungsbeispiel.
`day_begins_minute`	*Integer*	Gibt bei periodischen Intervallen (Stunden pro Tag, Minuten pro Tag und Sekunden pro Tag) die Minute an, in der der Tag beginnt (z. B. die *5* in *8:05*).
`day_begins_second`	*Integer*	Gibt bei periodischen Intervallen (Stunden pro Tag, Minuten pro Tag und Sekunden pro Tag) die Sekunde an, in der der Tag beginnt (z. B. die *17* in *8:05:17*).
`days_per_week`	*Integer*	Gibt bei periodischen Intervallen (Tage pro Woche, Stunden pro Tag, Minuten pro Tag und Sekunden pro Tag) die Anzahl der Tage pro Woche an.
`hours_per_day`	*Integer*	Gibt bei periodischen Intervallen (Stunden pro Tag, Minuten pro Tag und Sekunden pro Tag) die Anzahl der Stunden pro Tag an.

`interval_increment`	`1`
	`2`
	`3`
	`4`
	`5`
	`6`
	`10`
	`15`
	`20`
	`30`
		Gibt bei Minuten pro Tag und Sekunden pro Tag die Anzahl der Minuten bzw. Sekunden an, um die der Wert für jeden Datensatz erhöht werden soll.

`field_name_extension`	*string*	

Tabelle 94. Eigenschaften von "timeintervalsnode" (Forts.)		
Eigenschaften von `timeinter-valsnode`	**Datentyp**	**Eigenschaftsbeschreibung**
`field_name_extensi-on_as_prefix`	*Flag*	

`date_format`	`"DDMMYY"`
	`"MMDDYY"`
	`"YYMMDD"`
	`"YYYYMMDD"`
	`"YYYYDDD"`
	`DAY`
	`MONTH`
	`"DD-MM-YY"`
	`"DD-MM-YYYY"`
	`"MM-DD-YY"`
	`"MM-DD-YYYY"`
	`"DD-MON-YY"`
	`"DD-MON-YYYY"`
	`"YYYY-MM-DD"`
	`"DD.MM.YY"`
	`"DD.MM.YYYY"`
	`"MM.DD.YYYY"`
	`"DD.MON.YY"`
	`"DD.MON.YYYY"`
	`"DD/MM/YY"`
	`"DD/MM/YYYY"`
	`"MM/DD/YY"`
	`"MM/DD/YYYY"`
	`"DD/MON/YY"`
	`"DD/MON/YYYY"`
	`MON YYYY`
	`q Q YYYY`
	`ww WK YYYY`	

`time_format`	`"HHMMSS"`
	`"HHMM"`
	`"MMSS"`
	`"HH:MM:SS"`
	`"HH:MM"`
	`"MM:SS"`
	`"(H)H:(M)M:(S)S"`
	`"(H)H:(M)M"`
	`"(M)M:(S)S"`
	`"HH.MM.SS"`
	`"HH.MM"`
	`"MM.SS"`
	`"(H)H.(M)M.(S)S"`
	`"(H)H.(M)M"`
	`"(M)M.(S)S"`	

Tabelle 94. Eigenschaften von "timeintervalsnode" (Forts.)

Eigenschaften von `timeintervalsnode`	**Datentyp**	**Eigenschaftsbeschreibung**

`aggregate`	`Mean`
	`Sum`
	`Mode`
	`Min`
	`Max`
	`First`
	`Last`
	`TrueIfAnyTrue`
		Gibt die Aggregationsmethode für ein Feld an.

`pad`	`Blank`
	`MeanOfRecentPoints`
	`True`
	`False`
		Gibt die Auffüllmethode für ein Feld an.

`agg_mode`	`All` `Specify`	Gibt an, ob alle Felder mit Standardfunktionen nach Bedarf aggregiert bzw. aufgefüllt werden sollen oder ob die zu verwendenden Felder und Funktionen angegeben werden sollen.

`agg_range_default`	`Mean`
	`Sum`
	`Mode`
	`Min`
	`Max`
		Gibt die beim Aggregieren von stetigen Feldern zu verwendende Standardfunktion an.

`agg_set_default`	`Mode`
	`First`
	`Last`
		Gibt die beim Aggregieren von nominalen Feldern zu verwendende Standardfunktion an.

Tabelle 94. Eigenschaften von "timeintervalsnode" (Forts.)		
Eigenschaften von timeinter-valsnode	**Datentyp**	**Eigenschaftsbeschreibung**
agg_flag_default	TrueIfAnyTrue Mode First Last	
pad_range_default	Blank MeanOfRecentPoints	Gibt die beim Auffüllen von stetigen Feldern zu verwendende Standardfunktion an.
pad_set_default	Blank MostRecentValue	
pad_flag_default	Blank True False	
max_records_to_create	*Integer*	Gibt die maximale Anzahl der beim Auffüllen der Reihe zu erstellenden Datensätze an.
estimation_from_beginning	*Flag*	
estimation_to_end	*Flag*	
estimation_start_offset	*Integer*	
estimation_num_holdouts	*Integer*	
create_future_records	*Flag*	
num_future_records	*Integer*	
create_future_field	*Flag*	
future_field_name	*string*	

transposenode, Eigenschaften

Der Transponierknoten tauscht die Daten in Zeilen und Spalten aus, sodass Datensätze zu Feldern und Felder zu Datensätzen werden.

Beispiel

```
node = stream.create("transpose", "My node")
node.setPropertyValue("transposed_names", "Read")
node.setPropertyValue("read_from_field", "TimeLabel")
```

```
node.setPropertyValue("max_num_fields", "1000")
node.setPropertyValue("id_field_name", "ID")
```

Tabelle 95. transposenode, Eigenschaften

Eigenschaften von transposenode	**Datentyp**	**Eigenschaftsbeschreibung**
transpose_method	*enum*	Gibt die Transponiermethode an: Normal (normal), CASE-VAR (casetovar) oder VAR-CASE (vartocase).
transposed_names	Prefix Read	Eigenschaft für die Transponiermethode "Normal". Neue Felder können automatisch auf der Grundlage eines angegebenen Präfixes generiert oder aus einem bestehenden Feld in den Daten eingelesen werden.
prefix	*string*	Eigenschaft für die Transponiermethode "Normal".
num_new_fields	*Integer*	Eigenschaft für die Transponiermethode "Normal". Bei Verwendung eines Präfixes wird die maximale Anzahl der zu erstellenden Felder angegeben.
read_from_field	*Feld*	Eigenschaft für die Transponiermethode "Normal". Felder, aus denen Namen gelesen werden. Es muss sich um ein instanziiertes Feld handeln. Andernfalls tritt bei der Ausführung des Knotens ein Fehler auf.
max_num_fields	*Integer*	Eigenschaft für die Transponiermethode "Normal". Beim Einlesen von Namen aus einem Feld wird eine Obergrenze angegeben, um das Erstellen einer übermäßig großen Anzahl von Feldern zu verhindern.
transpose_type	Numeric String Custom	Eigenschaft für die Transponiermethode "Normal". Standardmäßig werden nur stetige Felder (numerischer Bereich) transponiert, Sie können jedoch stattdessen auch ein benutzerdefiniertes Subset numerischer Felder auswählen oder alle Zeichenfolgefelder transponieren.
transpose_fields	*Liste*	Eigenschaft für die Transponiermethode "Normal". Gibt die bei Verwendung der Option Custom (Benutzerdefiniert) zu transponierenden Felder an.
id_field_name	*Feld*	Eigenschaft für die Transponiermethode "Normal".
transpose_casetovar_idfields	*Feld*	Eigenschaft für die CASE-VAR-Transponiermethode (casetovar). Akzeptiert mehrere Felder für die Verwendung als Indexfelder. field1 ... fieldN

Tabelle 95. transposenode, Eigenschaften (Forts.)		
Eigenschaften von `transposenode`	**Datentyp**	**Eigenschaftsbeschreibung**
`transpose_casetovar_columnfields`	*Feld*	Eigenschaft für die CASE-VAR-Transponiermethode (`casetovar`). Akzeptiert mehrere Felder für die Verwendung als Spaltenfelder. `field1 ... fieldN`
`transpose_casetovar_valuefields`	*Feld*	Eigenschaft für die CASE-VAR-Transponiermethode (`casetovar`). Akzeptiert mehrere Felder für die Verwendung als Wertfelder. `field1 ... fieldN`
`transpose_vartocase_idfields`	*Feld*	Eigenschaft für die VAR-CASE-Transponiermethode (`vartocase`). Akzeptiert mehrere Felder für die Verwendung als ID-Variablenfelder. `field1 ... fieldN`
`transpose_vartocase_valfields`	*Feld*	Eigenschaft für die VAR-CASE-Transponiermethode (`vartocase`). Akzeptiert mehrere Felder für die Verwendung als Wertvariablenfelder. `field1 ... fieldN`

Eigenschaften von "typenode"

Der Typknoten gibt Feldmetadaten und Eigenschaften an. Sie können beispielsweise ein Messniveau (stetig, nominal, ordinal oder Flag) für die einzelnen Felder angeben, Optionen für den Umgang mit fehlenden Werten und systemdefinierten Nullwerten festlegen, die Rolle eines Felds zu Modellierungszwecken festlegen, Feld- und Wertbeschriftungen angeben oder die Werte für ein Feld angeben.

Beispiel

```
node = stream.createAt("type", "My node", 50, 50)
node.setKeyedPropertyValue("check", "Cholesterol", "Coerce")
node.setKeyedPropertyValue("direction", "Drug", "Input")
node.setKeyedPropertyValue("type", "K", "Range")
node.setKeyedPropertyValue("values", "Drug", ["drugA", "drugB", "drugC",
"drugD", "drugX",
 "drugY", "drugZ"])
node.setKeyedPropertyValue("null_missing", "BP", False)
node.setKeyedPropertyValue("whitespace_missing", "BP", False)
node.setKeyedPropertyValue("description", "BP", "Blood Pressure")
node.setKeyedPropertyValue("value_labels", "BP", [["HIGH", "High Blood Press
ure"],
 ["NORMAL", "normal blood pressure"]])
```

Beachten Sie: In einigen Fällen müssen Sie den Knoten "type" möglicherweise vollständig instanziieren, damit die anderen Knoten ordnungsgemäß arbeiten, beispielsweise die Eigenschaft `fields from` (Fel-

der aus) des Dichotomknotens. Sie können einfach einen Tabellenknoten anschließen und ausführen, um die Felder zu instanziieren:

```
tablenode = stream.createAt("table", "Table node", 150, 50)
stream.link(node, tablenode)
tablenode.run(None)
stream.delete(tablenode)
```

Tabelle 96. Eigenschaften von "typenode"

Eigenschaften von typenode	**Datentyp**	**Eigenschaftsbeschreibung**
direction	Input Target Both None Partition Split Frequency RecordID	Verschlüsselte Eigenschaft für Feldrollen. **Anmerkung:** Die Werte In und Out werden nicht mehr verwendet. In zukünftigen Versionen fällt möglicherweise die Unterstützung dafür weg.
type	Range Flag Set Typeless Discrete OrderedSet Default	Messniveau des Feldes (zuvor als "Typ" eines Feldes bezeichnet). Durch Festlegen von type auf Default werden alle Parametereinstellungen für values gelöscht, und wenn value_mode den Wert Specify aufweist, wird dieser auf Read zurückgesetzt. Wenn value_mode auf Pass oder Read festgelegt ist, hat das Festlegen von type keine Auswirkungen auf value_mode. **Anmerkung:** Die intern verwendeten Datentypen unterscheiden sich von den im Typknoten sichtbaren Datentypen. Es ergibt sich die folgende Korrespondenz: Range -> Continuous Set - > Nominal OrderedSet -> Ordinal Discrete- > Categorical.

Tabelle 96. Eigenschaften von "typenode" (Forts.)

Eigenschaften von typenode	Datentyp	Eigenschaftsbeschreibung
storage	Unknown String Integer Real Time Date Timestamp	Schreibgeschützte verschlüsselte Eigenschaft für Feldspeichertyp.
check	None Nullify Coerce Discard Warn Abort	Verschlüsselte Eigenschaft für das Überprüfen von Feldtyp und Bereich.
values	[*Wert Wert*]	Bei einem stetigen Feld ist der erste Wert das Minimum und der letzte das Maximum. Geben Sie für nominale Felder alle Werte an. Bei Flagfeldern steht der erste Wert für *false* (falsch) und der letzte für *true* (wahr). Bei automatischer Festlegung dieser Eigenschaft wird die Eigenschaft value_mode auf Specify festgelegt.
value_mode	Read Pass Read+ Current Specify	Bestimmt, wie Werte festgelegt werden. Beachten Sie, dass Sie diese Eigenschaft nicht direkt auf Specify festlegen können. Um bestimmte Werte zu verwenden, legen Sie die Eigenschaft values fest.

Tabelle 96. Eigenschaften von "typenode" (Forts.)		
Eigenschaften von typenode	**Datentyp**	**Eigenschaftsbeschreibung**
extend_values	*Flag*	Gilt, wenn value_mode auf Read gesetzt ist. Setzen Sie den Wert auf T, um neu gelesene Werte zu bereits für das Feld vorhandenen Werten hinzuzufügen. Setzen Sie den Wert auf F, um vorhandene Werte zu verwerfen und sie durch neu gelesene Werte zu ersetzen.
enable_missing	*Flag*	Bei Festlegung von T wird die Verfolgung von fehlenden Werten für das Feld aktiviert.
missing_values	[*Wert Wert ...*]	Gibt Datenwerte an, die fehlende Daten kennzeichnen.
range_missing	*Flag*	Gibt an, ob ein Bereich fehlender Werte (leer) für ein Feld definiert ist.
missing_lower	*string*	Wenn range_missing "true" ist, gibt diese Eigenschaft die Untergrenze des Bereichs fehlender Werte an.
missing_upper	*string*	Wenn range_missing "true" ist, gibt diese Eigenschaft die Obergrenze des Bereichs fehlender Werte an.
null_missing	*Flag*	Bei Festlegung von T werden *Nullen* (undefinierte Werte, die in der Software als $null$ angezeigt werden) als fehlende Werte betrachtet.
whitespace_missing	*Flag*	Bei Festlegung von T werden Werte, die nur leere Bereiche enthalten (Leerzeichen, Tabulatoren und Zeilenumbrüche), als fehlende Werte betrachtet.
description	*string*	Gibt die Beschreibung für ein Feld an.
value_labels	*[[Wert Beschriftungszeichenfolge] [Wert Beschriftungszeichenfolge] ...]*	Gibt Beschriftungen für Wertpaare an.
display_places	*Integer*	Legt die Dezimalstellen für das Feld bei der Anzeige fest (gilt nur für Felder mit dem Speichertyp REAL). Der Wert –1 verwendet den Streamstandardwert.
export_places	*Integer*	Legt die Dezimalstellen für das Feld beim Exportieren fest (gilt nur für Felder mit dem Speichertyp REAL). Der Wert –1 verwendet den Streamstandardwert.
decimal_separator	DEFAULT PERIOD COMMA	Legt das Dezimaltrennzeichen für das Feld fest (gilt nur für Felder mit dem Speichertyp REAL).

Tabelle 96. Eigenschaften von "typenode" (Forts.)

Eigenschaften von typenode	Datentyp	Eigenschaftsbeschreibung
date_format	"DDMMYY" "MMDDYY" "YYMMDD" "YYYYMMDD" "YYYYDDD" DAY MONTH "DD-MM-YY" "DD-MM-YYYY" "MM-DD-YY" "MM-DD-YYYY" "DD-MON-YY" "DD-MON-YYYY" "YYYY-MM-DD" "DD.MM.YY" "DD.MM.YYYY" "MM.DD.YYYY" "DD.MON.YY" "DD.MON.YYYY" "DD/MM/YY" "DD/MM/YYYY" "MM/DD/YY" "MM/DD/YYYY" "DD/MON/YY" "DD/MON/YYYY" MON YYYY q Q YYYY ww WK YYYY	Legt das Datumsformat für das Feld fest (gilt nur für Felder mit dem Speichertyp DATE oder TIMESTAMP).
time_format	"HHMMSS" "HHMM" "MMSS" "HH:MM:SS" "HH:MM" "MM:SS" "(H)H:(M)M:(S)S" "(H)H:(M)M" "(M)M:(S)S" "HH.MM.SS" "HH.MM" "MM.SS" "(H)H.(M)M.(S)S" "(H)H.(M)M" "(M)M.(S)S"	Legt das Zeitformat für das Feld fest (gilt nur für Felder mit dem Speichertyp TIME oder TIMESTAMP).
number_format	DEFAULT STANDARD SCIENTIFIC CURRENCY	Legt das Zahlenanzeigeformat für das Feld fest.
standard_places	*Integer*	Legt die Dezimalstellen für das Feld für die Anzeige im Standardformat fest. Der Wert –1 verwendet den Streamstandardwert. Der vorhandene Slot display_places ändert dies zwar auch, wird aber nicht mehr verwendet.

Tabelle 96. Eigenschaften von "typenode" (Forts.)		
Eigenschaften von typenode	**Datentyp**	**Eigenschaftsbeschreibung**
scienti-fic_places	*Integer*	Legt die Dezimalstellen für das Feld für die Anzeige im wissenschaftlichen Format fest. Der Wert –1 verwendet den Streamstandardwert.
curren-cy_places	*Integer*	Legt die Dezimalstellen für das Feld für die Anzeige im Währungsformat fest. Der Wert –1 verwendet den Streamstandardwert.
grou-ping_symbol	DEFAULT NONE LOCALE PERIOD COMMA SPACE	Legt das Symbol für die Zifferngruppierung für das Feld fest.
co-lumn_width	*Integer*	Legt die Spaltenbreite für das Feld fest. Der Wert –1 legt die Spaltenbreite auf Auto fest.
justify	AUTO CENTER LEFT RIGHT	Legt die Spaltenausrichtung für das Feld fest.

Tabelle 96. Eigenschaften von "typenode" (Forts.)

Eigenschaften von typenode	Datentyp	Eigenschaftsbeschreibung
measure_type	Range / MeasureType.RANGE Discrete / MeasureType.DISCRETE Flag / MeasureType.FLAG Set / MeasureType.SET OrderedSet / MeasureType.ORDERED_SET Typeless / MeasureType.TYPELESS Collection / MeasureType.COLLECTION Geospatial / MeasureType.GEOSPATIAL	Diese verschlüsselte Eigenschaft ähnelt type dahingehend, dass sie zum Definieren der dem Feld zugeordneten Messung verwendet werden kann. Das Python-Scripting unterscheidet sich dadurch, dass der Setter-Funktion auch einer der MeasureType-Werte übergeben werden kann, während die Getter-Funktion immer für MeasureType-Werte zurückgibt.
collection_measure	Range / MeasureType.RANGE Flag / MeasureType.FLAG Set / MeasureType.SET OrderedSet / MeasureType.ORDERED_SET Typeless / MeasureType.TYPELESS	Bei Sammlungsfeldern (Listen mit einer Tiefe von 0) definiert diese verschlüsselte Eigenschaft den Messtyp, der den zugrunde liegenden Werten zugeordnet ist.
geo_type	Point MultiPoint LineString MultiLineString Polygon MultiPolygon	Bei georäumlichen Feldern definiert diese verschlüsselte Eigenschaft den Typ des durch dieses Feld dargestellten georäumlichen Objekts. Dies sollte konsistent mit der Listentiefe der Werte sein.
has_coordinate_ system	*Boolesch*	Bei georäumlichen Feldern definiert diese Eigenschaft, ob dieses Feld ein Koordinatensystem hat
coordinate_system	*string*	Bei georäumlichen Feldern definiert diese verschlüsselte Eigenschaft das Koordinatensystem für dieses Feld.

Tabelle 96. Eigenschaften von "typenode" (Forts.)

Eigenschaften von typenode	Datentyp	Eigenschaftsbeschreibung
custom_storage_ type	Unknown / MeasureType.UNKNOWN String / MeasureType.STRING Integer / MeasureType.INTEGER Real / MeasureType.REAL Time / MeasureType.TIME Date / MeasureType.DATE Timestamp / MeasureType.TIMESTAMP List / MeasureType.LIST	Diese verschlüsselte Eigenschaft ähnelt custom_storage dahingehend, dass sie zum Definieren der Speicherüberschreibung für das Feld verwendet werden kann. Das Python-Scripting unterscheidet sich dadurch, dass der Setter-Funktion auch einer der StorageType-Werte übergeben werden kann, während die Getter-Funktion immer für StorageType-Werte zurückgibt.
custom_list_storage_type	String / MeasureType.STRING Integer / MeasureType.INTEGER Real / MeasureType.REAL Time / MeasureType.TIME Date / MeasureType.DATE Timestamp / MeasureType.TIMESTAMP	Bei Listenfeldern gibt diese verschlüsselte Eigenschaft den Speichertyp der zugrunde liegenden Werte an.
custom_list_depth	*Integer*	Bei Listenfeldern gibt diese verschlüsselte Eigenschaft die Tiefe des Felds an.
max_list_length	*Integer*	Steht nur bei Daten mit dem Messniveau *Georäumlich* oder *Sammlung* zur Verfügung. Legen Sie die maximale Länge der Liste durch Angabe der Anzahl der Elemente fest, die die Liste enthalten kann.
max_string_length	*Integer*	Nur für Daten *ohne Typ* verfügbar und Verwendung beim Generieren von SQL zur Erstellung einer Tabelle. Geben Sie den Wert der längsten Zeichenfolge in Ihren Daten ein. Dadurch wird in der Tabelle eine Spalte generiert, die groß genug für diese Zeichenfolge ist.

Kapitel 12. Eigenschaften von Diagrammknoten

Allgemeine Eigenschaften von Diagrammknoten

In diesem Abschnitt werden die für Diagrammknoten verfügbaren Eigenschaften, einschließlich allgemeiner Eigenschaften sowie knotenspezifischer Eigenschaften, beschrieben.

Tabelle 97. Allgemeine Eigenschaften von Diagrammknoten

Allgemeine Eigenschaften von Diagrammknoten	**Datentyp**	**Eigenschaftsbeschreibung**
`title`	*string*	Gibt den Titel an. Beispiel: "Dies ist ein Titel."
`caption`	*string*	Gibt die Titelzeile an. Beispiel: "Dies ist eine Titelzeile."

`output_mode`	`Screen`
	
	`File`
		Gibt an, ob die Ausgabe des Diagrammknotens angezeigt oder in eine Datei geschrieben werden soll.

`output_format`	`BMP`
	
	`JPEG`
	
	`PNG`
	
	`HTML`
	
	`output` (*.cou*)
		Gibt den Ausgabetyp an. Der zulässige Ausgabetyp ist für jeden Knoten unterschiedlich.

`full_filename`	*string*	Gibt den Zielpfad und den Dateinamen für die vom Diagrammknoten generierten Ausgabe an.
`use_graph_size`	*Flag*	Gibt an, ob die Größe des Diagramms mithilfe der unten angegebenen Eigenschaften für Breite und Höhe explizit festgelegt wird. Dies betrifft nur Diagramme, die auf dem Bildschirm ausgegeben werden. Nicht für den Verteilungsknoten verfügbar.
`graph_width`	*number*	Wenn `use_graph_size` den Wert `True` aufweist, wird hier die Diagrammhöhe in Pixeln festgelegt.
`graph_height`	*number*	Wenn `use_graph_size` den Wert `True` aufweist, wird hier die Diagrammhöhe in Pixeln festgelegt.

Inaktivieren optionaler Felder

Optionale Felder, wie Überlagerungsfelder für Plots, können inaktiviert werden, indem der Eigenschaftswert auf `" "` (leere Zeichenfolge) gesetzt wird, wie im folgenden Beispiel gezeigt:

```
plotnode.setPropertyValue("color_field", "")
```

Angeben von Farben

Die Farben für Titel, Titelzeilen, Hintergründe und Beschriftungen können mit hexadezimalen Zeichenfolgen, die mit einem Hashzeichen (#) beginnen, festgelegt werden. Beispiel: Um den Diagrammhintergrund auf Blau festzulegen, verwenden Sie folgende Anweisung:

```
mygraphnode.setPropertyValue("graph_background", "#87CEEB")
```

Die ersten beiden Stellen, 87, geben den roten Inhalt an, die mittleren Stellen, CE, legen den grünen Inhalt fest und die beiden letzten Stellen, EB, definieren den blauen Inhalt. Jede Stelle kann einen Wert im Bereich von 0-9 oder A-F annehmen. Zusammen können diese Werte eine Farbe des RGB-Farbraums (Rot, Grün, Blau) definieren.

Anmerkung: Beim Angeben von Farben in Rot, Grün und Blau können Sie den richtigen Farbcode mit der Feldauswahl in der Benutzerschnittstelle festlegen. Bewegen Sie die Maus über die Farbe, um eine QuickInfo mit den gewünschten Informationen anzuzeigen.

Eigenschaften von "collectionnode"

Der Sammlungsknoten zeigt die Verteilung der Werte für ein numerisches Feld im Verhältnis zu den Werten eines anderen an. (Er erstellt histogrammähnliche Diagramme.) Er eignet sich besonders für die Darstellung einer Variablen oder eines Felds, dessen Werte sich mit der Zeit verändern. Mithilfe eines 3-D-Diagramms können Sie außerdem eine symbolische Achse anlegen, auf der die Verteilungen nach Kategorie angezeigt werden.

Beispiel

```
node = stream.create("collection", "My node")
# "Plot" tab
node.setPropertyValue("three_D", True)
node.setPropertyValue("collect_field", "Drug")
node.setPropertyValue("over_field", "Age")
node.setPropertyValue("by_field", "BP")
node.setPropertyValue("operation", "Sum")
# "Overlay" section
node.setPropertyValue("color_field", "Drug")
node.setPropertyValue("panel_field", "Sex")
node.setPropertyValue("animation_field", "")
# "Options" tab
node.setPropertyValue("range_mode", "Automatic")
node.setPropertyValue("range_min", 1)
node.setPropertyValue("range_max", 100)
node.setPropertyValue("bins", "ByNumber")
node.setPropertyValue("num_bins", 10)
node.setPropertyValue("bin_width", 5)
```

Tabelle 98. collectionnode, Eigenschaften

Eigenschaften von collectionnode	Datentyp	Eigenschaftsbeschreibung
over_field	*Feld*	
over_label_auto	*Flag*	
over_label	*string*	
collect_field	*Feld*	
collect_label_auto	*Flag*	

Tabelle 98. collectionnode, Eigenschaften (Forts.)		
Eigenschaften von collectionnode	**Datentyp**	**Eigenschaftsbeschreibung**
collect_label	*string*	
three_D	*Flag*	
by_field	*Feld*	
by_label_auto	*Flag*	
by_label	*string*	
operation	Sum Mean Min Max SDev	
color_field	*string*	
panel_field	*string*	
animation_field	*string*	
range_mode	Automatic UserDefined	
range_min	*number*	
range_max	*number*	
bins	ByNumber ByWidth	
num_bins	*number*	
bin_width	*number*	
use_grid	*Flag*	
graph_background	*color*	Die Standardfarben für Diagramme werden am Anfang dieses Abschnitts beschrieben.
page_background	*color*	Die Standardfarben für Diagramme werden am Anfang dieses Abschnitts beschrieben.

Eigenschaften von "distributionnode"

Der Verteilungsknoten zeigt das Auftreten symbolischer (kategorialer) Werte wie beispielsweise Hypothekenart oder Geschlecht. Verteilungsknoten eignen sich insbesondere zum Aufzeigen von Unausgewogenheiten in den Daten, die mithilfe eines Balancierungsknotens vor dem Erstellen eines Modells ausgeglichen werden können.

Beispiel

```
node = stream.create("distribution", "My node")
# "Plot" tab
node.setPropertyValue("plot", "Flags")
node.setPropertyValue("x_field", "Age")
node.setPropertyValue("color_field", "Drug")
node.setPropertyValue("normalize", True)
node.setPropertyValue("sort_mode", "ByOccurence")
node.setPropertyValue("use_proportional_scale", True)
```

Tabelle 99. Eigenschaften von "distributionnode"

Eigenschaften von distributionnode	**Datenart**	**Eigenschaftsbeschreibung**
plot	SelectedFields Flags	
x_field	*Feld*	
color_field	*Feld*	Überlagerungsfeld
normalize	*Flag*	
sort_mode	ByOccurence Alphabetic	
use_proportional_scale	*Flag*	

Eigenschaften von "evaluationnode"

Der Evaluierungsknoten erleichtert die Evaluation und den Vergleich von Vorhersagemodellen. Das Evaluierungsdiagramm zeigt, wie gut Modelle bestimmte Ergebnisse vorhersagen. Die Datensätze werden auf der Grundlage des vorhergesagten Werts und des Konfidenzwerts für die Vorhersage sortiert. Die Datensätze werden in gleich große Gruppen (**Quantile**) aufgeteilt. Anschließend wird der Wert des Geschäftskriteriums für jedes Quantil geplottet, vom höchsten Wert bis zum niedrigsten Wert. Mehrere Modelle werden als separate Linien im Plot dargestellt.

Beispiel

```
node = stream.create("evaluation", "My node")
# "Plot" tab
node.setPropertyValue("chart_type", "Gains")
node.setPropertyValue("cumulative", False)
node.setPropertyValue("field_detection_method", "Name")
node.setPropertyValue("inc_baseline", True)
node.setPropertyValue("n_tile", "Deciles")
node.setPropertyValue("style", "Point")
node.setPropertyValue("point_type", "Dot")
node.setPropertyValue("use_fixed_cost", True)
node.setPropertyValue("cost_value", 5.0)
node.setPropertyValue("cost_field", "Na")
node.setPropertyValue("use_fixed_revenue", True)
node.setPropertyValue("revenue_value", 30.0)
node.setPropertyValue("revenue_field", "Age")
node.setPropertyValue("use_fixed_weight", True)
```

```
node.setPropertyValue("weight_value", 2.0)
node.setPropertyValue("weight_field", "K")
```

Tabelle 100. Eigenschaften von "evaluationnode"

Eigenschaften von evaluationnode	**Datentyp**	**Eigenschaftsbeschreibung**
chart_type	Gains Response Lift Profit ROI ROC	
inc_baseline	*Flag*	
field_detection_method	Metadata Name	
use_fixed_cost	*Flag*	
cost_value	*number*	
cost_field	*string*	
use_fixed_revenue	*Flag*	
revenue_value	*number*	
revenue_field	*string*	
use_fixed_weight	*Flag*	
weight_value	*number*	
weight_field	*Feld*	
n_tile	Quartiles Quintles Deciles Vingtiles Percentiles 1000-tiles	
cumulative	*Flag*	
style	Line Point	

Tabelle 100. Eigenschaften von "evaluationnode" (Forts.)		
Eigenschaften von evaluationnode	**Datentyp**	**Eigenschaftsbeschreibung**
point_type	Rectangle Dot Triangle Hexagon Plus Pentagon Star BowTie HorizontalDash VerticalDash IronCross Factory House Cathedral OnionDome ConcaveTriangle OblateGlobe CatEye FourSidedPillow RoundRectangle Fan	
export_data	*Flag*	
data_filename	*string*	
delimiter	*string*	
new_line	*Flag*	
inc_field_names	*Flag*	
inc_best_line	*Flag*	
inc_business_rule	*Flag*	
business_rule_condition	*string*	
plot_score_fields	*Flag*	
score_fields	*[field1 ... Feldn]*	
target_field	*Feld*	
use_hit_condition	*Flag*	
hit_condition	*string*	
use_score_expression	*Flag*	
score_expression	*string*	
caption_auto	*Flag*	

Eigenschaften von "graphboardnode"

Der Diagrammtafelknoten bietet viele verschiedene Diagrammtypen in einem einzigen Knoten. Bei Verwendung dieses Knotens können Sie die Datenfelder auswählen, die Sie untersuchen möchten, und anschließend eines der für die ausgewählten Daten verfügbaren Diagramme auswählen. Der Knoten filtert automatisch alle Diagrammtypen heraus, die nicht für die Feldauswahl geeignet sind.

Anmerkung: Wenn Sie eine Eigenschaft festlegen, die für den Diagrammtyp ungültig ist (z. B. ein y_field für ein Histogramm), wird diese Eigenschaft ignoriert.

Anmerkung: In der Benutzerschnittstelle gibt es auf der Registerkarte "Detailliert" der verschiedenen Grafiktypen ein Feld **Zusammenfassung**. Dieses Feld wird derzeit nicht von Scripting unterstützt.

Beispiel

```
node = stream.create("graphboard", "My node")
node.setPropertyValue("graph_type", "Line")
node.setPropertyValue("x_field", "K")
node.setPropertyValue("y_field", "Na")
```

Tabelle 101. Eigenschaften von "graphboardnode"

Eigenschaften von graphboard	**Datentyp**	**Eigenschaftsbeschreibung**
graph_type	2DDotplot 3DArea 3DBar 3DDensity 3DHistogram 3DPie 3DScatterplot Area ArrowMap Bar BarCounts BarCountsMap BarMap BinnedScatter Boxplot Bubble ChoroplethMeans ChoroplethMedians ChoroplethSums ChoroplethValues	Identifiziert den Diagrammtyp.

Tabelle 101. Eigenschaften von "graphboardnode" (Forts.)

Eigenschaften von graphboard	Datentyp	Eigenschaftsbeschreibung
	ChoroplethCounts CoordinateMap CoordinateChoroplethMeans CoordinateChoroplethMedians CoordinateChoroplethSums CoordinateChoroplethValues CoordinateChoroplethCounts Dotplot Heatmap HexBinScatter Histogram Line LineChartMap LineOverlayMap Parallel Path Pie PieCountMap PieCounts PieMap	

Tabelle 101. Eigenschaften von "graphboardnode" (Forts.)		
Eigenschaften von graphboard	**Datentyp**	**Eigenschaftsbeschreibung**
	PointOverlayMap PolygonOverlayMap Ribbon Scatterplot SPLOM Surface	
x_field	*Feld*	Legt eine benutzerdefinierte Beschriftung für die *x*-Achse fest. Nur für Beschriftungen verfügbar.
y_field	*Feld*	Legt eine benutzerdefinierte Beschriftung für die *y*-Achse fest. Nur für Beschriftungen verfügbar.
z_field	*Feld*	In einigen 3-D-Diagrammen verwendet.
color_field	*Feld*	In Heat-Maps verwendet.
size_field	*Feld*	In Blasendiagrammen verwendet.
categories_field	*Feld*	
values_field	*Feld*	
rows_field	*Feld*	
columns_field	*Feld*	
fields	*Feld*	
start_longitude_field	*Feld*	Bei Pfeilen auf einer Referenzkarte verwendet.
end_longitude_field	*Feld*	
start_latitude_field	*Feld*	
end_latitude_field	*Feld*	
data_key_field	*Feld*	In verschiedenen Karten verwendet.
panelrow_field	*string*	
panelcol_field	*string*	
animation_field	*string*	

Tabelle 101. Eigenschaften von "graphboardnode" (Forts.)		
Eigenschaften von graphboard	**Datentyp**	**Eigenschaftsbeschreibung**
longitude_field	*Feld*	Bei Koordinaten in Karten verwendet.
latitude_field	*Feld*	
map_color_field	*Feld*	

Eigenschaften von "histogramnode"

Der Histogrammknoten zeigt das Auftreten bestimmter Werte in numerischen Feldern. Damit werden häufig die Daten vor der weiteren Bearbeitung und der Modellerstellung untersucht. Ähnlich wie der Verteilungsknoten kann der Histogrammknoten oft Unausgewogenheiten in den Daten aufdecken.

Beispiel

```
node = stream.create("histogram", "My node")
# "Plot" tab
node.setPropertyValue("field", "Drug")
node.setPropertyValue("color_field", "Drug")
node.setPropertyValue("panel_field", "Sex")
node.setPropertyValue("animation_field", "")
# "Options" tab
node.setPropertyValue("range_mode", "Automatic")
node.setPropertyValue("range_min", 1.0)
node.setPropertyValue("range_max", 100.0)
node.setPropertyValue("num_bins", 10)
node.setPropertyValue("bin_width", 10)
node.setPropertyValue("normalize", True)
node.setPropertyValue("separate_bands", False)
```

Tabelle 102. Eigenschaften von "histogramnode"		
Eigenschaften von histogramnode	**Datentyp**	**Eigenschaftsbeschreibung**
field	*Feld*	
color_field	*Feld*	
panel_field	*Feld*	
animation_field	*Feld*	
range_mode	Automatic UserDefined	
range_min	*number*	
range_max	*number*	
bins	ByNumber ByWidth	
num_bins	*number*	
bin_width	*number*	

Tabelle 102. Eigenschaften von "histogramnode" (Forts.)		
Eigenschaften von histogramnode	**Datentyp**	**Eigenschaftsbeschreibung**
normalize	*Flag*	
separate_bands	*Flag*	
x_label_auto	*Flag*	
x_label	*string*	
y_label_auto	*Flag*	
y_label	*string*	
use_grid	*Flag*	
graph_background	*color*	Die Standardfarben für Diagramme werden am Anfang dieses Abschnitts beschrieben.
page_background	*color*	Die Standardfarben für Diagramme werden am Anfang dieses Abschnitts beschrieben.
normal_curve	*Flag*	Gibt an, ob die Normalverteilungskurve in der Ausgabe angezeigt werden soll.

mapvisualization, Eigenschaften

Der Kartenvisualisierungsknoten kann mehrere Eingabeverbindungen akzeptieren und Geodaten in einer Karte als Reihe von Schichten anzeigen. Jede Schicht ist ein einziges georäumliches Feld. Die Basisschicht kann beispielsweise eine Karte eines Landes sein. Darüber kann sich eine Schicht für Straßen, eine Schicht für Flüsse und eine Schicht für Städte befinden.

Tabelle 103. mapvisualization, Eigenschaften		
Eigenschaften von mapvisualization	**Datentyp**	**Eigenschaftsbeschreibung**
tag	*string*	Legt den Namen des Tags für die Eingabe fest. Der Standardtag ist eine Zahl auf der Grundlage der Reihenfolge, in der Eingaben mit dem Knoten verbunden wurden (der erste Verbindungstag ist 1, der zweite Verbindungstag ist 2 usw.

Tabelle 103. mapvisualization, Eigenschaften (Forts.)		
Eigenschaften von `mapvisualization`	**Datentyp**	**Eigenschaftsbeschreibung**
`layer_field`	*Feld*	Wählt aus, welches Geofeld als Schicht auf der Karte angezeigt wird. Die Standardauswahl basiert auf der folgenden Sortierreihenfolge: • Zuerst - Point • LineString • Polygon • Multipoint • MultiLinestring • Zuletzt - MultiPolygon Wenn zwei Felder mit demselben Messtyp vorhanden sind, wird standardmäßig das erste Feld (alphabetisch nach Namen) ausgewählt.
`color_type`	*Boolesch*	Gibt an, ob eine Standardfarbe auf alle Strukturen des Geofelds angewendet wird, oder ob ein Überlagerungsfeld angewendet wird, wenn die Strukturen eine Farbe auf der Basis der Werte eines anderen Felds im Dataset erhalten sollen. Mögliche Werte sind `standard` oder `overlay`. Der Standardwert ist `standard`.
`color`	*string*	Wenn `standard` für `color_type` ausgewählt ist, enthält die Dropdown-Liste dieselbe Farbpalette wie "Reihenfolge für Diagrammkategorienfarbe" auf der Registerkarte "Anzeige" der Benutzeroptionen. Der Standardwert ist die Diagrammkategorienfarbe 1.
`color_field`	*Feld*	Wenn `overlay` für `color_type` ausgewählt ist, enthält die Dropdown-Liste alle Felder aus demselben Dataset wie das Geofeld, das als Schicht ausgewählt ist.
`symbol_type`	*Boolesch*	Gibt an, ob ein Standardsymbol auf alle Strukturen des Geofelds angewendet wird, oder ob ein Überlagerungsfeld angewendet wird, das das Symbol für die Punkte auf der Basis der Werte eines anderen Felds im Dataset ändert. Mögliche Werte sind `standard` oder `overlay`. Der Standardwert ist `standard`.

Tabelle 103. mapvisualization, Eigenschaften (Forts.)		
Eigenschaften von `mapvisualization`	**Datentyp**	**Eigenschaftsbeschreibung**
`symbol`	*string*	Wenn `standard` für `symbol_type` ausgewählt ist, enthält die Dropdown-Liste eine Auswahl der Symbole, die zum Anzeigen von Punkten auf der Karte verwendet werden können.
`symbol_field`	*Feld*	Wenn `overlay` für `symbol_type` ausgewählt ist, enthält die Dropdown-Liste alle nominalen, ordinalen oder kategorialen Felder aus demselben Dataset wie das Geofeld, das als Schicht ausgewählt ist.
`size_type`	*Boolesch*	Gibt an, ob eine Standardgröße auf alle Datensätze des Geofelds angewendet wird, oder ob ein Überlagerungsfeld angewendet wird, das die Größe eines Symbols oder die Linienstärke auf der Basis der Werte eines anderen Felds im Dataset ändert. Mögliche Werte sind `standard` oder `overlay`. Der Standardwert ist `standard`.
`size`	*string*	Wenn `standard` für `size_type` ausgewählt ist, enthält die Dropdown-Liste für `point` oder `multipoint` eine Auswahl von Größen für das ausgewählte Symbol. Für `linestring` oder `multilinestring` enthält die Dropdown-Liste eine Auswahl von Linienstärken.
`size_field`	*Feld*	Wenn `overlay` für `size_type` ausgewählt ist, enthält die Dropdown-Liste alle Felder aus demselben Dataset wie das Geofeld, das als Schicht ausgewählt ist.
`transp_type`	*Boolesch*	Gibt an, ob eine Standardtransparenz auf alle Datensätze des Geofelds angewendet wird, oder ob eine Überlagerungstransparenz angewendet wird, die die Transparenzstufe für das Symbol, die Linie oder das Polygon auf der Basis der Werte eines anderen Felds im Dataset ändert. Mögliche Werte sind `standard` oder `overlay`. Der Standardwert ist `standard`.

Tabelle 103. mapvisualization, Eigenschaften (Forts.)

Eigenschaften von mapvisualization	**Datentyp**	**Eigenschaftsbeschreibung**
transp	*Integer*	Falls standard für transp_type ausgewählt ist, enthält die Dropdown-Liste eine Auswahl von Transparenzstufen, die bei 0% (nicht transparent) beginnt und in Schritten von jeweils 10% bis 100% (transparent) erhöht wird. Legt die Transparenz von Punkten, Linien oder Polygonen auf der Karte fest. Wenn overlay für size_type ausgewählt ist, enthält die Dropdown-Liste alle Felder aus demselben Dataset wie das Geofeld, das als Schicht ausgewählt ist. Für points, multipoints, linestrings und multilinestrings, polygons und multipolygons (die untere Schicht) ist der Standardwert 0%. Für polygons und multipolygons, die nicht die untere Schicht darstellen, ist der Standardwert 50% (dies vermeidet, dass Schichten unterhalb dieser Polygone nicht zu erkennen sind).
transp_field	*Feld*	Wenn overlay für transp_type ausgewählt ist, enthält die Dropdown-Liste alle Felder aus demselben Dataset wie das Geofeld, das als Schicht ausgewählt ist.
data_label_field	*Feld*	Gibt das Feld an, das als Datenbeschriftung auf der Karte verwendet werden soll. Wird diese Einstellung z. B. auf eine Polygonschicht angewendet, kann die Datenbeschriftung das Feld name sein, das den Namen jedes Polygons enthält. Die Auswahl des Felds name würde in diesem Fall dazu führen, dass diese Namen auf der Karte angezeigt werden.
use_hex_binning	*Boolesch*	Aktiviert die Hexadezimalklassierung und aktiviert alle Dropdown-Listen für Aggregation. Diese Einstellung ist standardmäßig inaktiviert.

Tabelle 103. mapvisualization, Eigenschaften (Forts.)

Eigenschaften von `mapvisualization`	**Datentyp**	**Eigenschaftsbeschreibung**
`color_aggregation` und `transp_aggregation`	*string*	Wenn Sie ein Überlagerungsfeld für eine Punktschicht auswählen, für die die Hexadezimalklassierung verwendet wird, müssen alle Werte für dieses Feld für alle Punkte im Sechseck aggregiert werden. Deshalb müssen Sie eine Aggregationsfunktion für alle Überlagerungsfelder angeben, die Sie auf die Karte anwenden wollen. Die folgenden Aggregationsfunktionen sind verfügbar: Stetig (Speicher des Typs "Reelle Zahl " oder Ganzzahl"): • Summe • Mittelwert • Min. • Max • Median • 1. Quartil • 3. Quartil Stetig (Speicher des Typs "Zeit", "Datum" oder "Zeitmarke"): • Mittelwert • Min. • Max Nominal/Kategorial: • Modus • Min. • Max Flag: • Wahr, wenn beliebige wahr • Falsch, wenn beliebige falsch
`custom_storage`	*string*	Legt den Speichertyp für das Feld insgesamt fest. Der Standardwert ist `List`. Wenn `List` angegeben ist, sind die folgenden Steuerelemente `custom_value_storage` und `list_depth` inaktiviert.
`custom_value_storage`	*string*	Legt die Speichertypen der Elemente in der Liste statt für das Feld insgesamt fest. Der Standardwert ist `Real`.

Tabelle 103. mapvisualization, Eigenschaften (Forts.)		
Eigenschaften von `mapvisualization`	**Datentyp**	**Eigenschaftsbeschreibung**
`list_depth`	*Integer*	Legt die Tiefe des Listenfelds fest. Die erforderliche Tiefe hängt vom Typ des Geofelds ab, dabei gelten die folgenden Kriterien: • Point - 0 • LineString - 1 • Polygon - 2 • Multipoint - 1 • MultiLineString - 2 • Multipolygon - 3 Sie müssen den Typ des georäumlichen Felds, das Sie zurück in eine Liste umwandeln, und die erforderliche Tiefe für diese Art von Feld kennen. Bei falschen Einstellungen kann das Feld nicht verwendet werden. Der Standardwert ist `0`, der Mindestwert ist `0` und der Maximalwert ist `10`.

Eigenschaften von "multiplotnode"

Ein Multiplot erstellt ein Plot, bei dem mehrere *Y*-Felder über einem einzelnen *X*-Feld dargestellt werden. Die *Y*-Felder werden als farbige Linien geplottet, die jeweils einem Plotknoten mit dem Stil **Linie** und dem X-Modus **Sortieren** entsprechen. Multiplots sind hilfreich, wenn die Fluktuation mehrerer Variablen im Laufe der Zeit untersucht werden soll.

Beispiel

```
node = stream.create("multiplot", "My node")
# "Plot" tab
node.setPropertyValue("x_field", "Age")
node.setPropertyValue("y_fields", ["Drug", "BP"])
node.setPropertyValue("panel_field", "Sex")
# "Overlay" section
node.setPropertyValue("animation_field", "")
node.setPropertyValue("tooltip", "test")
node.setPropertyValue("normalize", True)
node.setPropertyValue("use_overlay_expr", False)
node.setPropertyValue("overlay_expression", "test")
node.setPropertyValue("records_limit", 500)
node.setPropertyValue("if_over_limit", "PlotSample")
```

Tabelle 104. Eigenschaften von "multiplotnode"		
Eigenschaften von `multiplotnode`	**Datentyp**	**Eigenschaftsbeschreibung**
`x_field`	*Feld*	
`y_fields`	*Liste*	

Tabelle 104. Eigenschaften von "multiplotnode" (Forts.)		
Eigenschaften von multi-plotnode	**Datentyp**	**Eigenschaftsbeschreibung**
panel_field	*Feld*	
animation_field	*Feld*	
normalize	*Flag*	
use_overlay_expr	*Flag*	
overlay_expression	*string*	
records_limit	*number*	
if_over_limit	PlotBins PlotSample PlotAll	
x_label_auto	*Flag*	
x_label	*string*	
y_label_auto	*Flag*	
y_label	*string*	
use_grid	*Flag*	
graph_background	*color*	Die Standardfarben für Diagramme werden am Anfang dieses Abschnitts beschrieben.
page_background	*color*	Die Standardfarben für Diagramme werden am Anfang dieses Abschnitts beschrieben.

Eigenschaften von "plotnode"

Der Plotknoten zeigt die Beziehung zwischen numerischen Feldern an. Sie können einen Plot mithilfe von Punkten (Streudiagramm) oder mit Linien erstellen.

Beispiel

```
node = stream.create("plot", "My node")
# "Plot" tab
node.setPropertyValue("three_D", True)
node.setPropertyValue("x_field", "BP")
node.setPropertyValue("y_field", "Cholesterol")
node.setPropertyValue("z_field", "Drug")
# "Overlay" section
node.setPropertyValue("color_field", "Drug")
node.setPropertyValue("size_field", "Age")
node.setPropertyValue("shape_field", "")
node.setPropertyValue("panel_field", "Sex")
node.setPropertyValue("animation_field", "BP")
node.setPropertyValue("transp_field", "")
node.setPropertyValue("style", "Point")
# "Output" tab
node.setPropertyValue("output_mode", "File")
```

```
node.setPropertyValue("output_format", "JPEG")
node.setPropertyValue("full_filename", "C:/temp/graph_output/plot_out
put.jpeg")
```

Tabelle 105. Eigenschaften von "plotnode"

Eigenschaften von plotnode	Datentyp	Eigenschaftsbeschreibung
x_field	*Feld*	Legt eine benutzerdefinierte Beschriftung für die *x*-Achse fest. Nur für Beschriftungen verfügbar.
y_field	*Feld*	Legt eine benutzerdefinierte Beschriftung für die *y*-Achse fest. Nur für Beschriftungen verfügbar.
three_D	*Flag*	Legt eine benutzerdefinierte Beschriftung für die *y*-Achse fest. Nur für Beschriftungen in 3-D-Diagrammen verfügbar.
z_field	*Feld*	
color_field	*Feld*	Überlagerungsfeld
size_field	*Feld*	
shape_field	*Feld*	
panel_field	*Feld*	Gibt ein nominales oder Flagfeld an, mit dem je ein separates Diagramm für jede Kategorie angelegt werden soll. Die Diagramme werden gemeinsam in einem Ausgabefenster dargestellt.
animation_field	*Feld*	Gibt ein nominales oder Flagfeld an, mit dem die Kategorien für die Datenwerte in Form einer Reihe von Diagrammen dargestellt werden, die nacheinander als Animation angezeigt werden.
transp_field	*Feld*	Gibt ein Feld an, mit dem die Kategorien für die Datenwerte in Form von verschiedenen Transparenzstufen für die einzelnen Kategorien dargestellt werden. Für Liniendiagramme nicht verfügbar.
overlay_type	None Smoother Function	Gibt an, ob eine Überlagerungsfunktion oder ein LOESS-Smoother angezeigt werden soll.
overlay_expression	*string*	Gibt den Ausdruck an, der verwendet werden soll, wenn overlay_type auf Function gesetzt ist.
style	Point Line	

Tabelle 105. Eigenschaften von "plotnode" (Forts.)

Eigenschaften von plotnode	Datentyp	Eigenschaftsbeschreibung
point_type	Rectangle Dot Triangle Hexagon Plus Pentagon Star BowTie HorizontalDash VerticalDash IronCross Factory House Cathedral OnionDome ConcaveTriangle OblateGlobe CatEye FourSidedPillow RoundRectangle Fan	
x_mode	Sort Overlay AsRead	
x_range_mode	Automatic UserDefined	
x_range_min	*number*	
x_range_max	*number*	
y_range_mode	Automatic UserDefined	
y_range_min	*number*	
y_range_max	*number*	
z_range_mode	Automatic UserDefined	
z_range_min	*number*	
z_range_max	*number*	
jitter	*Flag*	
records_limit	*number*	
if_over_limit	PlotBins PlotSample PlotAll	

Tabelle 105. Eigenschaften von "plotnode" (Forts.)		
Eigenschaften von plotnode	**Datentyp**	**Eigenschaftsbeschreibung**
x_label_auto	*Flag*	
x_label	*string*	
y_label_auto	*Flag*	
y_label	*string*	
z_label_auto	*Flag*	
z_label	*string*	
use_grid	*Flag*	
graph_background	*color*	Die Standardfarben für Diagramme werden am Anfang dieses Abschnitts beschrieben.
page_background	*color*	Die Standardfarben für Diagramme werden am Anfang dieses Abschnitts beschrieben.
use_overlay_expr	*Flag*	Wird zugunsten von overlay_type nicht mehr verwendet.

Eigenschaften von "timeplotnode"

Der Zeitdiagrammknoten zeigt ein oder mehrere Sets mit Zeitreihendaten an. Normalerweise wird zuerst mithilfe eines Zeitintervallknotens ein *TimeLabel*-Feld erstellt, das dann zur Beschriftung der *X*-Achse verwendet wird.

Beispiel

```
node = stream.create("timeplot", "My node")
node.setPropertyValue("y_fields", ["sales", "men", "women"])
node.setPropertyValue("panel", True)
node.setPropertyValue("normalize", True)
node.setPropertyValue("line", True)
node.setPropertyValue("smoother", True)
node.setPropertyValue("use_records_limit", True)
node.setPropertyValue("records_limit", 2000)
# Appearance settings
node.setPropertyValue("symbol_size", 2.0)
```

Tabelle 106. Eigenschaften von "timeplotnode"		
Eigenschaften von timeplot-node	**Datentyp**	**Eigenschaftsbeschreibung**
plot_series	Series Models	
use_custom_x_field	*Flag*	
x_field	*Feld*	
y_fields	*Liste*	
panel	*Flag*	
normalize	*Flag*	

Tabelle 106. Eigenschaften von "timeplotnode" (Forts.)		
Eigenschaften von `timeplot-node`	**Datentyp**	**Eigenschaftsbeschreibung**
`line`	*Flag*	
`points`	*Flag*	

`point_type`	`Rectangle`
	`Dot`
	`Triangle`
	`Hexagon`
	`Plus`
	`Pentagon`
	`Star`
	`BowTie`
	`HorizontalDash`
	`VerticalDash`
	`IronCross`
	`Factory`
	`House`
	`Cathedral`
	`OnionDome`
	`ConcaveTriangle`
	`OblateGlobe`
	`CatEye`
	`FourSidedPillow`
	`RoundRectangle`
	`Fan`	

`smoother`	*Flag*	Sie können nur dann Glättungselemente zum Plot hinzufügen, wenn Sie `panel` auf `True` setzen.
`use_records_limit`	*Flag*	
`records_limit`	*Integer*	
`symbol_size`	*number*	Gibt eine Symbolgröße an.

`panel_layout`	`Horizontal`
	
	`Vertical`	

Eigenschaften von "eplotnode"

Der Knoten "E-Plot (Beta)" zeigt die Beziehung zwischen numerischen Feldern an. Er ähnelt dem Plotknoten, hat aber andere Optionen und seine Ausgabe verwendet eine neue grafische Benutzerschnittstelle, die für diesen Knoten spezifisch ist. Verwenden Sie die Betaversion des Knotens, um sich mit den neuen Grafikfunktionen vertraut zu machen.

Tabelle 107. Eigenschaften von "eplotnode"		
Eigenschaften von `eplotnode`	**Datentyp**	**Eigenschaftsbeschreibung**
`x_field`	*string*	Geben Sie das Feld an, das auf der horizontalen X-Achse angezeigt werden soll.
`y_field`	*string*	Geben Sie das Feld an, das auf der vertikalen Y-Achse angezeigt werden soll.
`color_field`	*string*	Geben Sie das Feld an, das ggf. für die Farbzuordnungsüberlagerung in der Ausgabe verwendet werden soll.

Tabelle 107. Eigenschaften von "eplotnode" (Forts.)		
Eigenschaften von eplotnode	**Datentyp**	**Eigenschaftsbeschreibung**
size_field	*string*	Geben Sie das Feld an, das ggf. für die Größenzuordnungsüberlagerung in der Ausgabe verwendet werden soll.
shape_field	*string*	Geben Sie das Feld an, das ggf. für die Formzuordnungsüberlagerung in der Ausgabe verwendet werden soll.
interested_fields	*string*	Geben Sie die Felder an, die die Ausgabe eingeschlossen werden sollen.
records_limit	*Integer*	Geben Sie die maximale Anzahl Datensätze für die Darstellung in der Ausgabe an. 2000 ist der Standardwert.
if_over_limit	*boolesch*	Geben Sie an, ob die Option Sample oder die Option Use all data verwendet werden soll, wenn der in records_limit angegebene Grenzwert überschritten wird. Sample ist der Standardwert. Bei Angabe dieses Werts werden Zufallsstichproben der Daten erstellt, bis der in records_limit angegebene Grenzwert erreicht ist. Wenn Sie Use all data angeben, um records_limit zu ignorieren und alle Datenpunkte darzustellen, kann dies die Leistung erheblich beeinträchtigen.

Eigenschaften von "tsnenode"

t-SNE (t-Distributed Stochastic Neighbor Embedding) ist ein Tool zum Visualisieren von hochdimensionalen Daten. Es wandelt Affinitäten von Datenpunkten in Wahrscheinlichkeiten um. Der t-SNE-Knoten in SPSS Modeler ist in Python implementiert und erfordert die Python-Bibliothek scikit-learn©.

Tabelle 108. Eigenschaften von "tsnenode"		
Eigenschaften von tsnenode	**Datentyp**	**Eigenschaftsbeschreibung**
mode_type	*string*	Geben Sie den Modus simple oder expert an.
n_components	*string*	Dimension des eingebetteten Raums (2-D oder 3-D). Geben Sie 2 oder 3 an. Der Standardwert ist 2.
method	*string*	Geben Sie barnes_hut oder exact an. Der Standardwert ist barnes_hut.
init	*string*	Initialisierung der Einbettung. Geben Sie random oder pca an. Der Standardwert ist random.

Tabelle 108. Eigenschaften von "tsnenode" (Forts.)

Eigenschaften von tsnenode	Datentyp	Eigenschaftsbeschreibung
target_field In target umbenannt ab Version 18.2.1.1.	*string*	Zielfeldname. Dies wird eine Farbzuordnungstabelle im Ausgabediagramm sein. Das Diagramm wird eine Farbe verwenden, wenn kein Zielfeld angegeben wird.
perplexity	*Float*	Die Perplexität bezieht sich auf die Anzahl der nächsten Nachbarn, die in vielen anderen Lernalgorithmen für Mannigfaltigkeiten verwendet wird. Größere Datasets erfordern in der Regel eine höhere Perplexität. Ziehen Sie einen Wert zwischen 5 und 50 in Erwägung. Der Standardwert ist 30.
early_exaggeration	*Float*	Steuert, wie eng die natürlichen Cluster des Originalraums im eingebetteten Raum sind und wie viel Platz zwischen ihnen ist. Der Standardwert ist 12.0.
learning_rate	*Float*	Der Standardwert ist 200.
n_iter	*Integer*	Maximale Anzahl Iterationen für die Optimierung. Geben Sie mindestens 250 an. Der Standardwert ist 1000.
angle	*Float*	Die Winkelgröße des fernen Knotens, gemessen von einem Punkt. Geben Sie einen Wert im Bereich 0-1 an. Der Standardwert ist 0.5.
enable_random_seed	*boolesch*	Setzen Sie diese Eigenschaft auf true, um den Parameter random_seed zu aktivieren. Der Standardwert ist false.
random_seed	*Integer*	Der zu verwendende Startwert für Zufallszahlen. Der Standardwert ist None.
n_iter_without_progress	*Integer*	Maximale Anzahl der Iterationen ohne Fortschritt. Der Standardwert ist 300.
min_grad_norm	*string*	Wenn die Gradientennorm unter diesem Schwellenwert liegt, wird die Optimierung gestoppt. Der Standardwert ist 1.0E-7. Gültige Werte: • 1.0E-1 • 1.0E-2 • 1.0E-3 • 1.0E-4 • 1.0E-5 • 1.0E-6 • 1.0E-7 • 1.0E-8

Tabelle 108. Eigenschaften von "tsnenode" (Forts.)		
Eigenschaften von tsnenode	**Datentyp**	**Eigenschaftsbeschreibung**
isGridSearch	*boolesch*	Setzen Sie diese Eigenschaft auf true, um t-SNE mit mehreren unterschiedlichen Perplexitäten auszuführen. Der Standardwert ist false.
output_Rename	*boolesch*	Geben Sie true an, wenn Sie einen benutzerdefinierten Namen bereitstellen wollen, oder geben Sie false an, um die Ausgabe automatisch zu benennen. Der Standardwert ist false.
output_to	*string*	Geben Sie Screen oder Output an. Der Standardwert ist Screen.
full_filename	*string*	Geben Sie den Namen der Ausgabedatei an.
output_file_type	*string*	Ausgabedateiformat. Geben Sie HTML oder Output object an. Der Standardwert ist HTML.

Eigenschaften von "webnode"

Der Netzdiagrammknoten zeigt die Stärke der Beziehung zwischen den Werten aus mindestens zwei symbolischen (kategorialen) Feldern. Im Diagramm wird die Verbindungsstärke durch unterschiedlich breite Linien angezeigt. Sie können beispielsweise einen Webknoten verwenden, um die Beziehung zwischen dem Kauf einer Gruppe von Artikeln auf einer E-Commerce-Site zu untersuchen.

Beispiel

```
node = stream.create("web", "My node")
# "Plot" tab
node.setPropertyValue("use_directed_web", True)
node.setPropertyValue("to_field", "Drug")
node.setPropertyValue("fields", ["BP", "Cholesterol", "Sex", "Drug"])
node.setPropertyValue("from_fields", ["BP", "Cholesterol", "Sex"])
node.setPropertyValue("true_flags_only", False)
node.setPropertyValue("line_values", "Absolute")
node.setPropertyValue("strong_links_heavier", True)
# "Options" tab
node.setPropertyValue("max_num_links", 300)
node.setPropertyValue("links_above", 10)
node.setPropertyValue("num_links", "ShowAll")
node.setPropertyValue("discard_links_min", True)
node.setPropertyValue("links_min_records", 5)
node.setPropertyValue("discard_links_max", True)
node.setPropertyValue("weak_below", 10)
node.setPropertyValue("strong_above", 19)
node.setPropertyValue("link_size_continuous", True)
node.setPropertyValue("web_display", "Circular")
```

Tabelle 109. Eigenschaften von "webnode"		
Eigenschaften von webnode	**Datentyp**	**Eigenschaftsbeschreibung**
use_directed_web	*Flag*	

Tabelle 109. Eigenschaften von "webnode" (Forts.)		
Eigenschaften von webnode	**Datentyp**	**Eigenschaftsbeschreibung**
fields	*Liste*	
to_field	*Feld*	
from_fields	*Liste*	
true_flags_only	*Flag*	
line_values	Absolute OverallPct PctLarger PctSmaller	
strong_links_heavier	*Flag*	
num_links	ShowMaximum ShowLinksAbove ShowAll	
max_num_links	*number*	
links_above	*number*	
discard_links_min	*Flag*	
links_min_records	*number*	
discard_links_max	*Flag*	
links_max_records	*number*	
weak_below	*number*	
strong_above	*number*	
link_size_continuous	*Flag*	
web_display	Circular Network Directed Grid	
graph_background	*color*	Die Standardfarben für Diagramme werden am Anfang dieses Abschnitts beschrieben.
symbol_size	*number*	Gibt eine Symbolgröße an.

Kapitel 13. Eigenschaften von Modellierungsknoten

Allgemeine Eigenschaften von Modellierungsknoten

Folgende Eigenschaften haben einige oder alle Modellierungsknoten gemeinsam. Etwaige Ausnahmen sind in der Dokumentation für die einzelnen Modellierungsknoten angegeben.

Tabelle 110. Allgemeine Eigenschaften von Modellierungsknoten

Eigenschaft	Werte	Eigenschaftsbeschreibung
`custom_fields`	*Flag*	Bei "true" können Sie Ziel-, Eingabe- und andere Felder für den aktuellen Knoten angeben. Bei "false" werden die aktuellen Einstellungen aus einem vorgeordneten Typknoten verwendet.

`target`
 	
oder
 	
`targets`
	Feld
	
	oder
	
	[*field1 ... FeldN]*
		Gibt je nach Modelltyp ein einzelnes Zielfeld oder mehrere Zielfelder an.

`inputs`	[*field1 ... FeldN]*	Im Modell verwendete Eingabe- bzw. Prädiktorfelder.
`partition`	*Feld*	
`use_partitioned_data`	*Flag*	Wenn ein Partitionsfeld definiert ist, gewährleistet diese Option, dass nur Daten aus der Trainingspartition für die Modellerstellung verwendet werden.
`use_split_data`	*Flag*	
`splits`	*[field1 ... Feldn]*	Gibt das Feld bzw. die Felder für die Aufteilungsmodellierung an. Nur wirksam, wenn `use_split_data` auf `True` gesetzt ist.
`use_frequency`	*Flag*	Gewichtungs- und Häufigkeitsfelder werden von bestimmten Modellen je nach Angabe für die einzelnen Modelltypen verwendet.
`frequency_field`	*Feld*	
`use_weight`	*Flag*	
`weight_field`	*Feld*	
`use_model_name`	*Flag*	
`model_name`	*string*	Benutzerdefinierter Name für neues Modell.

`mode`	`Simple`
	
	`Expert`	

anomalydetectionnode, Eigenschaften

Der Anomalieerkennungsknoten ermittelt ungewöhnliche Fälle bzw. "Ausreißer", die nicht den Mustern der "normalen" Daten entsprechen. Mit diesem Knoten ist es möglich, Ausreißer zu identifizieren, auch wenn sie nicht zu bekannten Mustern passen, und auch wenn Sie nicht genau wissen, wonach Sie suchen.

Beispiel

```
node = stream.create("anomalydetection", "My node")
node.setPropertyValue("anomaly_method", "PerRecords")
node.setPropertyValue("percent_records", 95)
node.setPropertyValue("mode", "Expert")
node.setPropertyValue("peer_group_num_auto", True)
node.setPropertyValue("min_num_peer_groups", 3)
node.setPropertyValue("max_num_peer_groups", 10)
```

Tabelle 111. anomalydetectionnode, Eigenschaften

anomalydetectionnode -Eigenschaften	**Werte**	**Eigenschaftsbeschreibung**
inputs	*[field1 ... Feldn]*	Anomalieerkennungsmodelle führen ein Screening von Datensätzen auf der Grundlage der angegebenen Eingabefelder durch. Sie verwenden kein Zielfeld. Gewichtungs- und Häufigkeitsfelder werden ebenfalls nicht verwendet. Weitere Informationen finden Sie im Thema „Allgemeine Eigenschaften von Modellierungsknoten” auf Seite 225.
mode	Expert Simple	
anomaly_method	IndexLevel PerRecords NumRecords	Gibt die Methode für die Bestimmung des Trennwerts zur Kennzeichnung von Datensätzen als anomal an.
index_level	*number*	Gibt den minimalen Trennwert für die Kennzeichnung von Anomalien an.
percent_records	*number*	Legt den Schwellenwert für die Kennzeichnung von Datensätzen auf der Grundlage des Prozentsatzes der Datensätze in den Trainingsdaten fest.
num_records	*number*	Legt den Schwellenwert für die Kennzeichnung von Datensätzen auf der Grundlage der Anzahl der Datensätze in den Trainingsdaten fest.

Tabelle 111. anomalydetectionnode, Eigenschaften (Forts.)		
anomalydetectionnode -Eigenschaften	**Werte**	**Eigenschaftsbeschreibung**
num_fields	*Integer*	Die Anzahl der für die einzelnen anomalen Datensätze zu meldenden Felder.
impute_missing_values	*Flag*	
adjustment_coeff	*number*	Wert, der zum Balancieren der relativen Gewichtung verwendet wird, das den stetigen und den kategorialen Feldern bei der Berechnung der Distanz zugewiesen wird.
peer_group_num_auto	*Flag*	Berechnet automatisch die Anzahl der Peergruppen.
min_num_peer_groups	*Integer*	Gibt an, wie viele Peergruppen mindestens verwendet werden, wenn peer_group_num_auto auf True gesetzt ist.
max_num_per_groups	*Integer*	Gibt die maximale Anzahl an Peergruppen an.
num_peer_groups	*Integer*	Gibt an, wie viele Peergruppen verwendet werden, wenn peer_group_num_auto auf False gesetzt ist.
noise_level	*number*	Bestimmt, wie Ausreißer bei der Clusterbildung behandelt werden. Geben Sie einen Wert zwischen 0 und 0,5 an.
noise_ratio	*number*	Gibt an, welcher Anteil des der Komponente zugeordneten Arbeitsspeichers für die Rauschpufferung verwendet werden soll. Geben Sie einen Wert zwischen 0 und 0,5 an.

apriorinode, Eigenschaften

Der Apriori-Knoten extrahiert ein Regelset aus den Daten und daraus die Regeln mit dem höchsten Informationsgehalt. Apriori bietet fünf verschiedene Methoden zur Auswahl von Regeln und verwendet ein ausgereiftes Indizierungsschema zur effizienten Verarbeitung großer Datasets. Bei großen Problemen ist Apriori in der Regel schneller zu trainieren, es gibt keine willkürliche Begrenzung für die Anzahl der Regeln, die beibehalten werden können, und es können Regeln mit bis zu 32 Vorbedingungen verarbeitet werden. Bei Apriori müssen alle Ein- und Ausgabefelder kategorial sein; dafür bietet es jedoch eine bessere Leistung, da es für diesen Datentyp optimiert ist.

Beispiel

```
node = stream.create("apriori", "My node")
# "Fields" tab
node.setPropertyValue("custom_fields", True)
```

```
node.setPropertyValue("partition", "Test")
# For non-transactional
node.setPropertyValue("use_transactional_data", False)
node.setPropertyValue("consequents", ["Age"])
node.setPropertyValue("antecedents", ["BP", "Cholesterol", "Drug"])
# For transactional
node.setPropertyValue("use_transactional_data", True)
node.setPropertyValue("id_field", "Age")
node.setPropertyValue("contiguous", True)
node.setPropertyValue("content_field", "Drug")
# "Model" tab
node.setPropertyValue("use_model_name", False)
node.setPropertyValue("model_name", "Apriori_bp_choles_drug")
node.setPropertyValue("min_supp", 7.0)
node.setPropertyValue("min_conf", 30.0)
node.setPropertyValue("max_antecedents", 7)
node.setPropertyValue("true_flags", False)
node.setPropertyValue("optimize", "Memory")
# "Expert" tab
node.setPropertyValue("mode", "Expert")
node.setPropertyValue("evaluation", "ConfidenceRatio")
node.setPropertyValue("lower_bound", 7)
```

Tabelle 112. apriorinode, Eigenschaften

Eigenschaften von apriori-node	**Werte**	**Eigenschaftsbeschreibung**
consequents	*Feld*	Apriori-Modelle verwenden anstelle von standardmäßigen Ziel- und Eingabefeldern Sukzedenzien bzw. Antezedenzien. Gewichtungs- und Häufigkeitsfelder werden nicht verwendet. Weitere Informationen finden Sie im Thema „Allgemeine Eigenschaften von Modellierungsknoten” auf Seite 225.
antecedents	[*field1 ... FeldN]*	
min_supp	*number*	
min_conf	*number*	
max_antecedents	*number*	
true_flags	*Flag*	
optimize	Speed Memory	
use_transactional_data	*Flag*	Wenn der Wert true lautet, ist der Score für jede Transaktions-ID unabhängig von anderen Transaktions-IDs. Wenn die Datenmenge, für die das Scoring durchgeführt werden soll, zu groß ist, um eine akzeptable Leistung zu erzielen, ist es ratsam, die Daten aufzuteilen.
contiguous	*Flag*	
id_field	*string*	
content_field	*string*	

Tabelle 112. apriorinode, Eigenschaften (Forts.)		
Eigenschaften von apriori-node	**Werte**	**Eigenschaftsbeschreibung**
mode	Simple Expert	
evaluation	RuleConfidence DifferenceToPrior ConfidenceRatio InformationDiffe-rence NormalizedChiSquare	
lower_bound	*number*	
optimize	Speed Memory	Dient zur Angabe, ob die Modellerstellung in Bezug auf die Geschwindigkeit oder den Speicher optimiert werden soll.

associationrulesnode, Eigenschaften

Der Assoziationsregelknoten ähnelt dem Apriori-Knoten. Im Gegensatz zu diesem kann er jedoch Listendaten verarbeiten. Darüber hinaus kann der Assoziationsregelknoten in Verbindung mit IBM SPSS Analytic Server verwendet werden, um große Datenmengen zu verarbeiten und die schnellere parallele Verarbeitung zu nutzen.

Tabelle 113. associationrulesnode, Eigenschaften		
Eigenschaften von associ-ationrulesnode	**Datentyp**	**Eigenschaftsbeschreibung**
predictions	*Feld*	Felder in dieser Liste können nur als Prädiktor einer Regel angezeigt werden.
conditions	*[field1...fieldN]*	Felder in dieser Liste können nur als Bedingung einer Regel angezeigt werden.
max_rule_conditions	*Integer*	Die maximale Anzahl Bedingungen, die in eine einzelne Regel eingeschlossen werden können. Minimum: 1, Maximum: 9.
max_rule_predictions	*Integer*	Die maximale Anzahl Vorhersagen, die in eine einzelne Regel eingeschlossen werden können. Minimum: 1, Maximum: 5.
max_num_rules	*Integer*	Die maximale Anzahl Regeln, die als Teil der Regelerstellung berücksichtigt werden können. Minimum: 1, Maximum: 10.000.

Tabelle 113. associationrulesnode, Eigenschaften (Forts.)

Eigenschaften von associationrulesnode	**Datentyp**	**Eigenschaftsbeschreibung**
rule_criterion_top_n	Confidence Rulesupport Lift Conditionsupport Deployability	Das Regelkriterium, das den Wert bestimmt, nach dem die ersten "N" Regeln im Modell ausgewählt werden.
true_flags	*boolesch*	Die Einstellung *Y* legt fest, dass nur die wahren Werte für Flagfelder während der Regelerstellung berücksichtigt werden.
rule_criterion	*boolesch*	Die Einstellung *Y* legt fest, dass die Regelkriteriumswerte für das Ausschließen von Regeln während der Modellerstellung verwendet werden.
min_confidence	*number*	0.1 bis 100; der Prozentwert für das erforderliche Mindestkonfidenzniveau für eine Regel, die vom Modell erzeugt wird. Wenn das Modell eine Regel mit einem Konfidenzniveau erzeugt, das den hier angegebenen Wert unterschreitet, wird die Regel verworfen.
min_rule_support	*number*	0.1 bis 100; der Prozentwert für die erforderliche Mindestregelunterstützung für eine Regel, die vom Modell erzeugt wird. Wenn das Modell eine Regel mit einem Regelunterstützungsniveau erzeugt, das den hier angegebenen Wert unterschreitet, wird die Regel verworfen.
min_condition_support	*number*	0.1 bis 100; der Prozentwert für die erforderliche Mindestbedingungsunterstützung für eine Regel, die vom Modell erzeugt wird. Wenn das Modell eine Regel mit einem Bedingungsunterstützungsniveau erzeugt, das den hier angegebenen Wert unterschreitet, wird die Regel verworfen.
min_lift	*Integer*	1 bis 10; stellt den erforderlichen Mindestlift für eine Regel dar, die vom Modell erzeugt wird. Wenn das Modell eine Regel mit einem Liftniveau erzeugt, das den hier angegebenen Wert unterschreitet, wird die Regel verworfen.

Tabelle 113. associationrulesnode, Eigenschaften (Forts.)

Eigenschaften von associationrulesnode	**Datentyp**	**Eigenschaftsbeschreibung**
exclude_rules	*boolesch*	Wird verwendet, um eine Liste zusammengehöriger Felder auszuwählen, aus denen das Modell keine Regeln erstellen soll. Beispiel: set :gsarsnode.exclude_rules = [[[field1,field2, field3]],[[field4, field5]]], wobei jede durch [] getrennte Liste von Feldern eine Zeile in der Tabelle ist.
num_bins	*Integer*	Legen Sie die Anzahl automatischer Klassen fest, in die stetige Felder eingeteilt werden. Minimum: 2, Maximum: 10.
max_list_length	*Integer*	Wird auf alle Listenfelder angewendet, für die die maximale Länge nicht bekannt ist. Elemente in der Liste bis zur hier angegebenen Zahl werden in die Modellerstellung eingeschlossen; weitere Elemente werden verworfen. Minimum: 1, Maximum: 100.
output_confidence	*boolesch*	
output_rule_support	*boolesch*	
output_lift	*boolesch*	
output_condition_support	*boolesch*	
output_deployability	*boolesch*	
rules_to_display	upto all	Die maximale Anzahl Regeln, die in den Ausgabetabellen angezeigt werden sollen.
display_upto	*Integer*	Wenn upto in rules_to_display festgelegt wird, legen Sie die Anzahl Regeln fest, die in den Ausgabetabellen angezeigt werden sollen. Minimum: 1.
field_transformations	*boolesch*	
records_summary	*boolesch*	
rule_statistics	*boolesch*	
most_frequent_values	*boolesch*	
most_frequent_fields	*boolesch*	
word_cloud	*boolesch*	

Tabelle 113. associationrulesnode, Eigenschaften (Forts.)		
Eigenschaften von associationrulesnode	**Datentyp**	**Eigenschaftsbeschreibung**
word_cloud_sort	Confidence Rulesupport Lift Conditionsupport Deployability	
word_cloud_display	*Integer*	Minimum: 1, Maximum: 20.
max_predictions	*Integer*	Die maximale Anzahl Regeln, die auf jede Eingabe mit dem Score angewendet werden können.
criterion	Confidence Rulesupport Lift Conditionsupport Deployability	Wählen Sie das Maß aus, das zum Festlegen der Stärke von Regeln verwendet wird.
allow_repeats	*boolesch*	Legt fest, ob Regeln mit derselben Vorhersage in den Score eingeschlossen werden.
check_input	NoPredictions Predictions NoCheck	

autoclassifiernode, Eigenschaften

Mit dem Knoten "Autom. Klassifikationsmerkmal" können Sie eine Reihe verschiedener Modelle für binäre Ergebnisse ("Ja" oder "Nein", "Abwanderung" oder "Keine Abwanderung" usw.) erstellen und vergleichen, um den besten Ansatz für die jeweilige Analyse auszuwählen. Es wird eine Reihe von Modellierungsalgorithmen unterstützt, sodass Sie die gewünschten Methoden, die spezifischen Optionen für die jeweilige Methode und die Kriterien zum Vergleich der Ergebnisse auswählen können. Der Knoten generiert auf der Basis der angegebenen Optionen eine Gruppe von Modellen und erstellt anhand der von Ihnen angegebenen Kriterien eine Rangordnung der besten Kandidaten.

Beispiel

```
node = stream.create("autoclassifier", "My node")
node.setPropertyValue("ranking_measure", "Accuracy")
```

```
node.setPropertyValue("ranking_dataset", "Training")
node.setPropertyValue("enable_accuracy_limit", True)
node.setPropertyValue("accuracy_limit", 0.9)
node.setPropertyValue("calculate_variable_importance", True)
node.setPropertyValue("use_costs", True)
node.setPropertyValue("svm", False)
```

Tabelle 114. autoclassifiernode, Eigenschaften

Eigenschaften von autoclassifier-node	**Werte**	**Eigenschaftsbeschreibung**
target	*Feld*	Für Flagziele verlangt der Knoten "Automatisches Klassifikationsmerkmal" ein einzelnes Ziel und eines oder mehrere Eingabefelder. Außerdem können Gewichtungs- und Häufigkeitsfelder angegeben werden. Weitere Informationen finden Sie im Thema „Allgemeine Eigenschaften von Modellierungsknoten” auf Seite 225.
ranking_measure	Accuracy Area_under_curve Profit Lift Num_variables	
ranking_dataset	Training Test	
number_of_models	*Integer*	Anzahl der Modelle, die in das Modellnugget aufgenommen werden sollen. Geben Sie eine Ganzzahl zwischen 1 und 100 an.
calculate_variable_importance	*Flag*	
enable_accuracy_limit	*Flag*	
accuracy_limit	*Integer*	Ganzzahl zwischen 0 und 100.
enable_ area_under_curve _limit	*Flag*	
area_under_curve_limit	*number*	Reelle Zahl zwischen 0,0 und 1,0.
enable_profit_limit	*Flag*	
profit_limit	*number*	Ganzzahl größer als 0.
enable_lift_limit	*Flag*	
lift_limit	*number*	Reelle Zahl größer als 1,0.

Tabelle 114. autoclassifiernode, Eigenschaften (Forts.)

Eigenschaften von autoclassifier-node	**Werte**	**Eigenschaftsbeschreibung**
enable_number_of_variables_limit	*Flag*	
number_of_variables_limit	*number*	Ganzzahl größer als 0.
use_fixed_cost	*Flag*	
fixed_cost	*number*	Reelle Zahl größer 0.0.
variable_cost	*Feld*	
use_fixed_revenue	*Flag*	
fixed_revenue	*number*	Reelle Zahl größer 0.0.
variable_revenue	*Feld*	
use_fixed_weight	*Flag*	
fixed_weight	*number*	Reelle Zahl größer als 0,0
variable_weight	*Feld*	
lift_percentile	*number*	Ganzzahl zwischen 0 und 100.
enable_model_build_time_limit	*Flag*	
model_build_time_limit	*number*	Ganzzahl für die Anzahl der Minuten, die maximal für die Erstellung jedes einzelnen Modells aufgewendet werden.
enable_stop_after_time_limit	*Flag*	
stop_after_time_limit	*number*	Reelle Zahl für die Anzahl der Stunden, die als Obergrenze für die insgesamt verstrichene Zeit für den Durchlauf eines automatischen Klassifikationsmerkmals verwendet wird.
enable_stop_after_valid_model_produced	*Flag*	
use_costs	*Flag*	
<algorithm>	*Flag*	Aktiviert oder inaktiviert die Verwendung eines bestimmten Algorithmus.
<algorithm>.<property>	*string*	Legt einen Eigenschaftswert für einen bestimmten Algorithmus fest. Weitere Informationen finden Sie im Thema „Festlegen der Algorithmuseigenschaften” auf Seite 235.

Festlegen der Algorithmuseigenschaften

Für die Knoten vom Typ "Automatisches Klassifikationsmerkmal", "Autonumerisch" und "Autom. Cluster" können Eigenschaften für bestimmte vom Knoten verwendete Algorithmen mithilfe des folgenden allgemeinen Formats festgelegt werden:

```
autonode.setKeyedPropertyValue(<algorithm>, <property>, <value>)
```

Beispiel:

```
node.setKeyedPropertyValue("neuralnetwork", "method", "MultilayerPerceptron")
```

Die Algorithmusnamen für den Knoten "Automatisches Klassifikationsmerkmal" lauten `cart`, `chaid`, `quest`, `c50`, `logreg`, `decisionlist`, `bayesnet`, `discriminant`, `svm` und `knn`.

Die Algorithmusnamen für den Knoten "Autonumerisch" lauten `cart`, `chaid`, `neuralnetwork`, `genlin`, `svm`, `regression`, `linear` und `knn`.

Algorithmusnamen für den Knoten "Autom. Cluster" sind `twostep`, `k-means` und `kohonen`.

Für die Eigenschaftsnamen wird der jeweils für den Algorithmusknoten dokumentierte Standard verwendet.

Algorithmuseigenschaften, die Punkte oder andere Satzzeichen enthalten, müssen in einzelne Anführungsstriche eingebettet sein, z. B.:

```
node.setKeyedPropertyValue("logreg", "tolerance", "1.0E-5")
```

Als Eigenschaft können auch mehrere Werte zugewiesen werden, z. B.:

```
node.setKeyedPropertyValue("decisionlist", "search_direction", ["Up",
"Down"])
```

So können Sie die Verwendung eines bestimmten Algorithmus aktivieren bzw. inaktivieren:

```
node.setPropertyValue("chaid", True)
```

Anmerkung: In Fällen, in denen bestimmte Algorithmusoptionen nicht im Knoten "Automatisches Klassifikationsmerkmal" verfügbar sind oder in denen nur ein einzelner Wert und kein Wertebereich angegeben werden kann, gelten dieselben Einschränkungen bei der Scripterstellung wie beim standardmäßigen Zugriff auf den Knoten.

autoclusternode, Eigenschaften

Mit dem Knoten "Autom. Cluster" können Sie Clustering-Modelle, die Gruppen und Datensätze mit ähnlichen Merkmalen identifizieren, schätzen und vergleichen. Die Funktionsweise des Knotens gleicht der von anderen Knoten für automatisierte Modellierung, und Sie können in einem einzigen Modellierungsdurchgang mit mehreren Optionskombinationen experimentieren. Die Modelle können anhand grundlegender Maße (Kennzahlen) verglichen werden, mit denen versucht wird, die Nützlichkeit der Clustermodelle zu filtern und zu einzustufen und ein Maß für die Bedeutung bestimmter Felder bereitzustellen.

Beispiel

```
node = stream.create("autocluster", "My node")
node.setPropertyValue("ranking_measure", "Silhouette")
node.setPropertyValue("ranking_dataset", "Training")
node.setPropertyValue("enable_silhouette_limit", True)
node.setPropertyValue("silhouette_limit", 5)
```

Tabelle 115. autoclusterernode, Eigenschaften		
Eigenschaften von autoclusternode	**Werte**	**Eigenschaftsbeschreibung**
evaluation	*Feld*	**Anmerkung:** Nur Knoten "Autom. Cluster". Kennzeichnet das Feld, für das ein Wichtigkeitswert berechnet wird. Kann auch verwendet werden, um festzustellen, wie gut das Cluster den Wert dieses Felds differenziert, also wie gut das Modell den Wert für dieses Feld vorhersagen kann.
ranking_measure	Silhouette Num_clusters Size_smallest_cluster Size_largest_cluster Smallest_to_largest Importance	
ranking_dataset	Training Test	
summary_limit	*Integer*	Anzahl der im Bericht aufzuführenden Modelle. Geben Sie eine Ganzzahl zwischen 1 und 100 an.
enable_silhouette_limit	*Flag*	
silhouette_limit	*Integer*	Ganzzahl zwischen 0 und 100.
enable_number_less_limit	*Flag*	
number_less_limit	*number*	Reelle Zahl zwischen 0,0 und 1,0.
enable_number_greater_limit	*Flag*	
number_greater_limit	*number*	Ganzzahl größer als 0.
enable_smallest_cluster_limit	*Flag*	
smallest_cluster_units	Percentage Counts	
smallest_cluster_limit_percentage	*number*	
smallest_cluster_limit_count	*Integer*	Ganzzahl größer als 0.

Tabelle 115. autoclusterernode, Eigenschaften (Forts.)		
Eigenschaften von autoclusternode	**Werte**	**Eigenschaftsbeschreibung**
enable_largest_cluster_limit	*Flag*	
largest_cluster_units	Percentage Counts	
largest_cluster_limit_percentage	*number*	
largest_cluster_limit_count	*Integer*	
enable_smallest_largest_limit	*Flag*	
smallest_largest_limit	*number*	
enable_importance_limit	*Flag*	
importance_limit_condition	Greater_than Less_than	
importance_limit_greater_than	*number*	Ganzzahl zwischen 0 und 100.
importance_limit_less_than	*number*	Ganzzahl zwischen 0 und 100.
<algorithm>	*Flag*	Aktiviert oder inaktiviert die Verwendung eines bestimmten Algorithmus.
<algorithm>.<property>	*string*	Legt einen Eigenschaftswert für einen bestimmten Algorithmus fest. Weitere Informationen finden Sie im Thema „Festlegen der Algorithmuseigenschaften" auf Seite 235.

autonumericnode, Eigenschaften

Der Knoten "Auto-Numerisch" schätzt und vergleicht mit einer Reihe verschiedener Methoden Modelle für die Ergebnisse stetiger numerischer Bereiche. Der Knoten arbeitet auf dieselbe Weise wie der Knoten "Automatisches Klassifikationsmerkmal": Sie können die zu verwendenden Algorithmen auswählen und in einem Modellierungsdurchlauf mit mehreren Optionskombinationen experimentieren. Folgende Algorithmen werden unterstützt: C&RT-Baum, CHAID, lineare Regression, verallgemeinerte lineare Regression und Support Vector Machines (SVM). Modelle können basierend auf Korrelation, relativem Fehler, oder Anzahl der verwendeten Variablen verglichen werden.

Beispiel

```
node = stream.create("autonumeric", "My node")
node.setPropertyValue("ranking_measure", "Correlation")
node.setPropertyValue("ranking_dataset", "Training")
node.setPropertyValue("enable_correlation_limit", True)
```

```
node.setPropertyValue("correlation_limit", 0.8)
node.setPropertyValue("calculate_variable_importance", True)
node.setPropertyValue("neuralnetwork", True)
node.setPropertyValue("chaid", False)
```

Tabelle 116. autonumericnode, Eigenschaften

Eigenschaften von autonumericnode	**Werte**	**Eigenschaftsbeschreibung**
custom_fields	*Flag*	Bei "True" werden anstelle der Typknoteneinstellungen Einstellungen aus benutzerdefinierten Feldern verwendet.
target	*Feld*	Für den Knoten "Autonumerisch" sind ein einzelnes Ziel und eines oder mehrere Eingabefelder erforderlich. Außerdem können Gewichtungs- und Häufigkeitsfelder angegeben werden. Weitere Informationen finden Sie im Thema „Allgemeine Eigenschaften von Modellierungsknoten” auf Seite 225.
inputs	*[field1 ... field2]*	
partition	*Feld*	
use_frequency	*Flag*	
frequency_field	*Feld*	
use_weight	*Flag*	
weight_field	*Feld*	
use_partitioned_data	*Flag*	Wenn ein Partitionsfeld definiert ist, werden nur die Trainingsdaten für die Modellerstellung verwendet.
ranking_measure	Correlation NumberOfFields	
ranking_dataset	Test Training	
number_of_models	*Integer*	Anzahl der Modelle, die in das Modellnugget aufgenommen werden sollen. Geben Sie eine Ganzzahl zwischen 1 und 100 an.
calculate_variable_importance	*Flag*	
enable_correlation_limit	*Flag*	
correlation_limit	*Integer*	
enable_number_of_fields_limit	*Flag*	
number_of_fields_limit	*Integer*	

Tabelle 116. autonumericnode, Eigenschaften (Forts.)		
Eigenschaften von autonumericnode	**Werte**	**Eigenschaftsbeschreibung**
enable_relative_error_limit	*Flag*	
relative_error_limit	*Integer*	
enable_model_build_time_limit	*Flag*	
model_build_time_limit	*Integer*	
enable_stop_after_time_limit	*Flag*	
stop_after_time_limit	*Integer*	
stop_if_valid_model	*Flag*	
<algorithm>	*Flag*	Aktiviert oder inaktiviert die Verwendung eines bestimmten Algorithmus.
<algorithm>.<property>	*string*	Legt einen Eigenschaftswert für einen bestimmten Algorithmus fest. Weitere Informationen finden Sie im Thema „Festlegen der Algorithmuseigenschaften” auf Seite 235.

Eigenschaften von "bayesnetnode"

Mithilfe des Bayes-Netzknotens können Sie ein Wahrscheinlichkeitsmodell erstellen, indem Sie beobachtete und aufgezeichnete Hinweise mit Weltwissen kombinieren, um die Wahrscheinlichkeit ihres Vorkommens zu ermitteln. Der Knoten konzentriert sich auf Netze vom Typ Tree Augmented Naïve Bayes (TAN) und Markov Blanket, die hauptsächlich für die Klassifizierung verwendet werden.

Beispiel

```
node = stream.create("bayesnet", "My node")
node.setPropertyValue("continue_training_existing_model", True)
node.setPropertyValue("structure_type", "MarkovBlanket")
node.setPropertyValue("use_feature_selection", True)
# Expert tab
node.setPropertyValue("mode", "Expert")
node.setPropertyValue("all_probabilities", True)
node.setPropertyValue("independence", "Pearson")
```

Tabelle 117. Eigenschaften von "bayesnetnode"		
bayesnetnode -Eigenschaften	**Werte**	**Eigenschaftsbeschreibung**
inputs	*[field1 ... Feldn]*	Bayes-Netzmodelle verwenden ein einzelnes Zielfeld und eines oder mehrere Eingabefelder. Stetige Felder werden automatisch klassiert. Weitere Informationen finden Sie im Thema „Allgemeine Eigenschaften von Modellierungsknoten” auf Seite 225.

Tabelle 117. Eigenschaften von "bayesnetnode" (Forts.)		
bayesnetnode -Eigenschaften	**Werte**	**Eigenschaftsbeschreibung**
continue_training_existing_model	*Flag*	
structure_type	TAN MarkovBlanket	Dient zur Auswahl der beim Erstellen des Bayes-Netzes zu verwendenden Struktur.
use_feature_selection	*Flag*	
parameter_learning_method	Likelihood Bayes	Gibt die Methode an, die zur Schätzung der Tabellen zur bedingten Wahrscheinlichkeit zwischen Knoten verwendet wird, wenn die Werte der übergeordneten Elemente bekannt sind.
mode	Expert Simple	
missing_values	*Flag*	
all_probabilities	*Flag*	
independence	Likelihood Pearson	Gibt die Methode an, die zur Einschätzung verwendet wird, ob paarige Beobachtungen bei zwei Variablen voneinander unabhängig sind.
significance_level	*number*	Gibt den Trennwert für die Bestimmung der Unabhängigkeit an.
maximal_conditioning_set	*number*	Legt die Maximalzahl der für die Unabhängigkeitstests zu verwendenden Konditionierungsvariablen fest.
inputs_always_selected	*[field1 ... Feldn]*	Gibt an, welche Felder aus dem Dataset immer beim Erstellen des Bayes-Netzes verwendet werden. **Anmerkung:** Das Zielfeld ist immer ausgewählt.
maximum_number_inputs	*number*	Gibt die maximale Anzahl an Eingabefeldern an, die beim Erstellen des Bayes-Netzes verwendet werden sollen.
calculate_variable_importance	*Flag*	
calculate_raw_propensities	*Flag*	
calculate_adjusted_propensities	*Flag*	

Tabelle 117. Eigenschaften von "bayesnetnode" (Forts.)		
bayesnetnode -Eigenschaften	**Werte**	**Eigenschaftsbeschreibung**
adjusted_propensity_partition	Test Validation	

buildr, Eigenschaften

Der R-Erstellungsknoten ermöglicht es Ihnen, ein benutzerdefiniertes R-Script einzugeben, um die Modellerstellung und das Modellscoring, die in IBM SPSS Modeler implementiert sind, auszuführen.

Beispiel

```
node = stream.create("buildr", "My node")
node.setPropertyValue("score_syntax", """
result<-predict(modelerModel,newdata=modelerData)
modelerData<-cbind(modelerData,result)
var1<-c(fieldName="NaPrediction",fieldLabel="",fieldStorage="real",fieldMea
sure="",
fieldFormat="",fieldRole="")
modelerDataModel<-data.frame(modelerDataModel,var1)""")
```

Tabelle 118. buildr, Eigenschaften		
Eigenschaften von buildr	**Werte**	**Eigenschaftsbeschreibung**
build_syntax	*string*	R-Scriptsyntax für die Modellerstellung.
score_syntax	*string*	R-Scriptsyntax für das Modellscoring.
convert_flags	StringsAndDoubles LogicalValues	Option zum Konvertieren von Flagfeldern.
convert_datetime	*Flag*	Option zum Konvertieren von Variablen mit Datums- oder Datums-/Zeitformaten in R-Datums-/Zeitformate.
convert_datetime_class	POSIXct POSIXlt	Optionen, die angeben, in welches Format Variablen mit Datums- oder Datums-/Zeitformaten konvertiert werden.
convert_missing	*Flag*	Option zum Konvertieren fehlender Werte in den R NA-Wert.
output_html	*Flag*	Option für die Anzeige von Diagrammen im R-Modellnugget.
output_text	*Flag*	Option zum Schreiben von R-Konsolentext auf eine Registerkarte des R-Modellnuggets.

c50node, Eigenschaften

Der C5.0-Knoten erstellt entweder einen Entscheidungsbaum oder ein Regelset. Das Modell teilt die Stichprobe auf der Basis des Felds auf, das auf der jeweiligen Ebene den maximalen Informationsgewinn liefert. Das Zielfeld muss kategorial sein. Es sind mehrere Aufteilungen in mehr als zwei Untergruppen zulässig.

Beispiel

```
node = stream.create("c50", "My node")
# "Model" tab
node.setPropertyValue("use_model_name", False)
node.setPropertyValue("model_name", "C5_Drug")
node.setPropertyValue("use_partitioned_data", True)
node.setPropertyValue("output_type", "DecisionTree")
node.setPropertyValue("use_xval", True)
node.setPropertyValue("xval_num_folds", 3)
node.setPropertyValue("mode", "Expert")
node.setPropertyValue("favor", "Generality")
node.setPropertyValue("min_child_records", 3)
# "Costs" tab
node.setPropertyValue("use_costs", True)
node.setPropertyValue("costs", [["drugA", "drugX", 2]])
```

Tabelle 119. c50node, Eigenschaften

Eigenschaften von c50node	Werte	Eigenschaftsbeschreibung
target	*Feld*	C50-Modelle verwenden ein einzelnes Zielfeld und eines oder mehrere Eingabefelder. Außerdem kann ein Gewichtungsfeld angegeben werden. Weitere Informationen finden Sie im Thema „Allgemeine Eigenschaften von Modellierungsknoten" auf Seite 225.
output_type	DecisionTree RuleSet	
group_symbolics	*Flag*	
use_boost	*Flag*	
boost_num_trials	*number*	
use_xval	*Flag*	
xval_num_folds	*number*	
mode	Simple Expert	
favor	Accuracy Generality	Genauigkeit oder Allgemeingültigkeit werden vorselektiert.
expected_noise	*number*	
min_child_records	*number*	
pruning_severity	*number*	

Tabelle 119. c50node, Eigenschaften (Forts.)		
Eigenschaften von c50node	**Werte**	**Eigenschaftsbeschreibung**
use_costs	*Flag*	
costs	*strukturiert*	Hierbei handelt es sich um eine strukturierte Eigenschaft.
use_winnowing	*Flag*	
use_global_pruning	*Flag*	Standardmäßig auf True gesetzt.
calculate_variable_importance	*Flag*	
calculate_raw_propensities	*Flag*	
calculate_adjusted_propensities	*Flag*	
adjusted_propensity_partition	Test Validation	

carmanode, Eigenschaften

Beim CARMA-Modell wird ein Regelset aus den Daten extrahiert, ohne dass Sie Eingabe- oder Zielfelder angeben müssen. Im Gegensatz zu Apriori bietet der CARMA-Knoten Erstellungseinstellungen für die Regelunterstützung (Unterstützung für Antezedens und Sukzedens) und nicht nur für die Unterstützung von Antezedens. Die erstellten Rgeln können somit für eine größere Palette an Anwendungen verwendet werden, beispielsweise um eine Liste mit Produkten und Dienstleistungen (Antezedenzien) zu finden, deren Nachfolger (Sukzedens) das Element darstellt, das Sie in der Ferienzeit desselben Jahres bewerben möchten.

Beispiel

```
node = stream.create("carma", "My node")
# "Fields" tab
node.setPropertyValue("custom_fields", True)
node.setPropertyValue("use_transactional_data", True)
node.setPropertyValue("inputs", ["BP", "Cholesterol", "Drug"])
node.setPropertyValue("partition", "Test")
# "Model" tab
node.setPropertyValue("use_model_name", False)
node.setPropertyValue("model_name", "age_bp_drug")
node.setPropertyValue("use_partitioned_data", False)
node.setPropertyValue("min_supp", 10.0)
node.setPropertyValue("min_conf", 30.0)
node.setPropertyValue("max_size", 5)
# Expert Options
node.setPropertyValue("mode", "Expert")
node.setPropertyValue("use_pruning", True)
node.setPropertyValue("pruning_value", 300)
node.setPropertyValue("vary_support", True)
node.setPropertyValue("estimated_transactions", 30)
node.setPropertyValue("rules_without_antecedents", True)
```

Tabelle 120. carmanode, Eigenschaften		
Eigenschaften von carmanode	**Werte**	**Eigenschaftsbeschreibung**
inputs	*[field1 ... Feldn]*	CARMA-Modelle verwenden eine Liste mit Eingabefeldern, jedoch kein Ziel. Gewichtungs- und Häufigkeitsfelder werden nicht verwendet. Weitere Informationen finden Sie im Thema „Allgemeine Eigenschaften von Modellierungsknoten" auf Seite 225.
id_field	*Feld*	Das Feld wird als ID-Feld für die Modellerstellung verwendet.
contiguous	*Flag*	Dient zur Angabe, ob IDs im ID-Feld zusammenhängend sind.
use_transactional_data	*Flag*	
content_field	*Feld*	
min_supp	*Zahl(Prozent)*	Bezieht sich auf die Regelunterstützung und nicht auf die Antezedens-Unterstützung. Der Standardwert ist 20 %.
min_conf	*Zahl(Prozent)*	Der Standardwert ist 20 %.
max_size	*number*	Der Standardwert ist 10.
mode	Simple Expert	Der Standardwert ist Simple.
exclude_multiple	*Flag*	Schließt Regeln mit mehreren Sukzedenzien aus. Der Standardwert ist False.
use_pruning	*Flag*	Der Standardwert ist False.
pruning_value	*number*	Der Standardwert ist 500.
vary_support	*Flag*	
estimated_transactions	*Integer*	
rules_without_antecedents	*Flag*	

cartnode, Eigenschaften

Der Knoten für Klassifizierungs- und Regressions-Bäume (C&RT-Bäume) erstellt einen Entscheidungsbaum, mit dem Sie zukünftige Beobachtungen vorhersagen oder klassifizieren können. Bei dieser Methode wird eine rekursive Partitionierung verwendet, um die Trainingsdatensätze in Segmente aufzuteilen. Dabei wird bei jedem Schritt die Unreinheit verringert und ein Knoten im Baum wird als "rein" betrachtet, wenn 100 % der Fälle in eine bestimmte Kategorie des Zielfelds fallen. Ziel- und Eingabefelder können numerische Bereiche oder kategorial (nominal, ordinal, oder Flags) sein; alle Aufteilungen sind binär (nur zwei Untergruppen).

Beispiel

```
node = stream.createAt("cart", "My node", 200, 100)
# "Fields" tab
node.setPropertyValue("custom_fields", True)
node.setPropertyValue("target", "Drug")
node.setPropertyValue("inputs", ["Age", "BP", "Cholesterol"])
# "Build Options" tab, "Objective" panel
node.setPropertyValue("model_output_type", "InteractiveBuilder")
node.setPropertyValue("use_tree_directives", True)
node.setPropertyValue("tree_directives", """Grow Node Index 0 Children 1 2
Grow Node Index 2 Children 3 4""")
# "Build Options" tab, "Basics" panel
node.setPropertyValue("prune_tree", False)
node.setPropertyValue("use_std_err_rule", True)
node.setPropertyValue("std_err_multiplier", 3.0)
node.setPropertyValue("max_surrogates", 7)
# "Build Options" tab, "Stopping Rules" panel
node.setPropertyValue("use_percentage", True)
node.setPropertyValue("min_parent_records_pc", 5)
node.setPropertyValue("min_child_records_pc", 3)
# "Build Options" tab, "Advanced" panel
node.setPropertyValue("min_impurity", 0.0003)
node.setPropertyValue("impurity_measure", "Twoing")
# "Model Options" tab
node.setPropertyValue("use_model_name", True)
node.setPropertyValue("model_name", "Cart_Drug")
```

Tabelle 121. cartnode, Eigenschaften

Eigenschaften von `cartnode`	**Werte**	**Eigenschaftsbeschreibung**
`target`	*Feld*	Modelle vom Typ "C&R-Baum" verwenden ein einzelnes Zielfeld und eines oder mehrere Eingabefelder. Außerdem kann ein Häufigkeitsfeld angegeben werden. Weitere Informationen finden Sie im Thema „Allgemeine Eigenschaften von Modellierungsknoten" auf Seite 225.
`continue_training_existing_model`	*Flag*	

`objective`	`Standard`
	
	`Boosting`
	
	`Bagging`
	
	`psm`
		`psm` wird für sehr große Datasets verwendet und macht eine Serververbindung erforderlich.

`model_output_type`	`Single`
	
	`InteractiveBuilder`	

`use_tree_directives`	*Flag*	

Tabelle 121. cartnode, Eigenschaften (Forts.)

Eigenschaften von cartnode	**Werte**	**Eigenschaftsbeschreibung**
tree_directives	*string*	Geben Sie Aufbauregeln für die Erweiterung des Baums an. Aufbauregeln können in dreifache Anführungszeichen gesetzt werden, um auf Escapezeichen für neue Zeilen oder Anführungszeichen verzichten zu können. Beachten Sie, dass die Anweisungen auf kleinste Änderungen in den Daten- oder Modellierungsoptionen reagieren und nicht für andere Datasets verallgemeinert werden können.
use_max_depth	Default Custom	
max_depth	*Integer*	Maximale Baumtiefe von 0 bis 1000. Wird nur bei use_max_depth = Custom verwendet.
prune_tree	*Flag*	Baum reduzieren, um zu große Anpassung zu vermeiden.
use_std_err	*Flag*	Maximale Risikendifferenz verwenden (in Standardfehler).
std_err_multiplier	*number*	Maximale Differenz.
max_surrogates	*number*	Maximale Anzahl Ersatztrenner.
use_percentage	*Flag*	
min_parent_records_pc	*number*	
min_child_records_pc	*number*	
min_parent_records_abs	*number*	
min_child_records_abs	*number*	
use_costs	*Flag*	
costs	*strukturiert*	Strukturierte Eigenschaft.
priors	Data Equal Custom	
custom_priors	*strukturiert*	Strukturierte Eigenschaft.
adjust_priors	*Flag*	
trails	*number*	Anzahl der Komponentenmodelle für Boosting oder Bagging.

Tabelle 121. cartnode, Eigenschaften (Forts.)		
Eigenschaften von cartnode	**Werte**	**Eigenschaftsbeschreibung**
set_ensemble_method	Voting HighestProbability HighestMeanProbability	Standardkombinationsregel für kategoriale Ziele.
range_ensemble_method	Mean Median	Standardkombinationsregel für stetige Ziele.
large_boost	*Flag*	Boosting auf sehr große Datasets anwenden.
min_impurity	*number*	
impurity_measure	Gini Twoing Ordered	
train_pct	*number*	Set zur Verhinderung übermäßiger Anpassung.
set_random_seed	*Flag*	Option "Ergebnisse replizieren".
seed	*number*	
calculate_variable_importance	*Flag*	
calculate_raw_propensities	*Flag*	
calculate_adjusted_propensities	*Flag*	
adjusted_propensity_partition	Test Validation	

chaidnode, Eigenschaften

Der CHAID-Knoten erzeugt Entscheidungsbäume unter Verwendung von Chi-Quadrat-Statistiken zur Ermittlung optimaler Aufteilungen. Im Gegensatz zu den Knoten vom Typ "C&RT-Baum" und "QUEST" kann CHAID nicht binäre Bäume generieren, d. h. Bäume mit Aufteilungen mit mehr als zwei Verzweigungen. Ziel- und Eingabefelder können in einem numerischen Bereich (stetig) oder kategorial sein. Exhaustive CHAID ist eine Änderung von CHAID, die noch gründlicher vorgeht, indem sie alle möglichen Aufteilungen untersucht. Allerdings wird hierbei mehr Rechenzeit beansprucht.

Beispiel

```
filenode = stream.createAt("variablefile", "My node", 100, 100)
filenode.setPropertyValue("full_filename", "$CLEO_DEMOS/DRUG1n")
node = stream.createAt("chaid", "My node", 200, 100)
stream.link(filenode, node)

node.setPropertyValue("custom_fields", True)
node.setPropertyValue("target", "Drug")
node.setPropertyValue("inputs", ["Age", "Na", "K", "Cholesterol", "BP"])
node.setPropertyValue("use_model_name", True)
node.setPropertyValue("model_name", "CHAID")
node.setPropertyValue("method", "Chaid")
node.setPropertyValue("model_output_type", "InteractiveBuilder")
node.setPropertyValue("use_tree_directives", True)
node.setPropertyValue("tree_directives", "Test")
node.setPropertyValue("split_alpha", 0.03)
node.setPropertyValue("merge_alpha", 0.04)
node.setPropertyValue("chi_square", "Pearson")
node.setPropertyValue("use_percentage", False)
node.setPropertyValue("min_parent_records_abs", 40)
node.setPropertyValue("min_child_records_abs", 30)
node.setPropertyValue("epsilon", 0.003)
node.setPropertyValue("max_iterations", 75)
node.setPropertyValue("split_merged_categories", True)
node.setPropertyValue("bonferroni_adjustment", True)
```

Tabelle 122. chaidnode, Eigenschaften

Eigenschaften von chaidnode	**Werte**	**Eigenschaftsbeschreibung**
target	*Feld*	CHAID-Modelle erfordern ein einzelnes Ziel und eines oder mehrere Eingabefelder. Außerdem kann ein Häufigkeitsfeld angegeben werden. Weitere Informationen finden Sie im Thema „Allgemeine Eigenschaften von Modellierungsknoten” auf Seite 225.
continue_training_existing_model	*Flag*	
objective	Standard Boosting Bagging psm	psm wird für sehr große Datasets verwendet und macht eine Serververbindung erforderlich.
model_output_type	Single InteractiveBuilder	
use_tree_directives	*Flag*	
tree_directives	*string*	
method	Chaid ExhaustiveChaid	

Tabelle 122. chaidnode, Eigenschaften (Forts.)

Eigenschaften von chaidnode	Werte	Eigenschaftsbeschreibung
use_max_depth	Default Custom	
max_depth	*Integer*	Maximale Baumtiefe von 0 bis 1000. Wird nur bei use_max_depth = Custom verwendet.
use_percentage	*Flag*	
min_parent_records_pc	*number*	
min_child_records_pc	*number*	
min_parent_records_abs	*number*	
min_child_records_abs	*number*	
use_costs	*Flag*	
costs	*strukturiert*	Strukturierte Eigenschaft.
trails	*number*	Anzahl der Komponentenmodelle für Boosting oder Bagging.
set_ensemble_method	Voting HighestProbability HighestMeanProbability	Standardkombinationsregel für kategoriale Ziele.
range_ensemble_method	Mean Median	Standardkombinationsregel für stetige Ziele.
large_boost	*Flag*	Boosting auf sehr große Datasets anwenden.
split_alpha	*number*	Signifikanzschwelle für Aufteilung.
merge_alpha	*number*	Signifikanzschwelle für Zusammenführung.
bonferroni_adjustment	*Flag*	Signifikanzwerte mit der Bonferroni-Methode anpassen.
split_merged_categories	*Flag*	Erneutes Aufteilen zusammengeführter Kategorien zulassen.
chi_square	Pearson LR	Verwendetes Verfahren für die Berechnung der Chi-Quadrat-Statistik: Pearson oder Likelihood-Quotient
epsilon	*number*	Minimale Änderung in der erwarteten Zellhäufigkeit...
max_iterations	*number*	Maximale Anzahl der Iterationen für Konvergenz.

Tabelle 122. chaidnode, Eigenschaften (Forts.)		
Eigenschaften von chaidnode	**Werte**	**Eigenschaftsbeschreibung**
set_random_seed	*Integer*	
seed	*number*	
calculate_variable_importance	*Flag*	
calculate_raw_propensities	*Flag*	
calculate_adjusted_propensities	*Flag*	
adjusted_propensity_partition	Test Validation	
maximum_number_of_models	*Integer*	

coxregnode, Eigenschaften

Der Knoten vom Typ "Cox-Regression" ermöglicht Ihnen auch bei zensierten Datensätzen die Erstellung eines Überlebensmodells für Daten über die Zeit bis zum Eintreten des Ereignisses. Das Modell erzeugt eine Überlebensfunktion, welche die Wahrscheinlichkeit vorhersagt, dass das Ereignis von Interesse zu einem vorgegebenen Zeitpunkt (*t*) für die angegebenen Werte der Eingabevariablen eingetreten ist.

Beispiel

```
node = stream.create("coxreg", "My node")
node.setPropertyValue("survival_time", "tenure")
node.setPropertyValue("method", "BackwardsStepwise")
# Expert tab
node.setPropertyValue("mode", "Expert")
node.setPropertyValue("removal_criterion", "Conditional")
node.setPropertyValue("survival", True)
```

Tabelle 123. coxregnode, Eigenschaften		
coxregnode -Eigenschaften	**Werte**	**Eigenschaftsbeschreibung**
survival_time	*Feld*	Cox-Regressionsmodelle erfordern ein einzelnes Feld, das die Überlebenszeiten enthält.
target	*Feld*	Cox-Regressionsmodelle erfordern ein einzelnes Zielfeld und eines oder mehrere Eingabefelder. Weitere Informationen finden Sie im Thema „Allgemeine Eigenschaften von Modellierungsknoten” auf Seite 225.
method	Enter Stepwise BackwardsStepwise	

Tabelle 123. coxregnode, Eigenschaften (Forts.)		
coxregnode -Eigenschaften	**Werte**	**Eigenschaftsbeschreibung**
groups	*Feld*	
model_type	MainEffects Custom	
custom_terms	[*"BP*Sex" "BP*Age"*]	
mode	Expert Simple	
max_iterations	*number*	
p_converge	1.0E-4 1.0E-5 1.0E-6 1.0E-7 1.0E-8 0	
p_converge	1.0E-4 1.0E-5 1.0E-6 1.0E-7 1.0E-8 0	
l_converge	1.0E-1 1.0E-2 1.0E-3 1.0E-4 1.0E-5 0	

Tabelle 123. coxregnode, Eigenschaften (Forts.)		
coxregnode -Eigenschaften	**Werte**	**Eigenschaftsbeschreibung**
removal_criterion	LR Wald Conditional	
probability_entry	*number*	
probability_removal	*number*	
output_display	EachStep LastStep	
ci_enable	*Flag*	
ci_value	90 95 99	
correlation	*Flag*	
display_baseline	*Flag*	
survival	*Flag*	
hazard	*Flag*	
log_minus_log	*Flag*	
one_minus_survival	*Flag*	
separate_line	*Feld*	
value	*Zahl* oder *Zeichenfolge*	Wenn für ein Feld kein Wert angegeben ist, wird die Standardoption "Mittelwert" für das betreffende Feld verwendet.

Eigenschaften von "decisionlistnode"

Der Knoten "Entscheidungsliste" kennzeichnet Untergruppen bzw. Segmente, die eine höhere oder geringere Wahrscheinlichkeit für ein bestimmtes binäres Ergebnis aufweisen als die Gesamtpopulation. Sie könnten beispielsweise nach Kunden suchen, deren Abwanderung unwahrscheinlich ist oder die mit großer Wahrscheinlichkeit positiv auf eine Kampagne reagieren. Sie können Ihr Fachwissen in das Modell integrieren, indem Sie eigene, benutzerdefinierte Segmente hinzufügen und eine Vorschau anzeigen, in der alternative Modelle nebeneinander angezeigt werden, um die Ergebnisse zu vergleichen. Entscheidungslistenmodelle bestehen aus einer Liste von Regeln, bei denen jede Regel eine Bedingung und ein Ergebnis aufweist. Regeln werden in der vorgegebenen Reihenfolge angewendet und die erste Regel, die zutrifft, bestimmt das Ergebnis.

Beispiel

```
node = stream.create("decisionlist", "My node")
node.setPropertyValue("search_direction", "Down")
node.setPropertyValue("target_value", 1)
node.setPropertyValue("max_rules", 4)
node.setPropertyValue("min_group_size_pct", 15)
```

Tabelle 124. Eigenschaften von "decisionlistnode"

Eigenschaften von decision-listnode	**Werte**	**Eigenschaftsbeschreibung**
target	*Feld*	Entscheidungslistenmodelle verwenden ein einzelnes Ziel und eines oder mehrere Eingabefelder. Außerdem kann ein Häufigkeitsfeld angegeben werden. Weitere Informationen finden Sie im Thema „Allgemeine Eigenschaften von Modellierungsknoten" auf Seite 225.
model_output_type	Model InteractiveBuilder	
search_direction	Up Down	Bezieht sich auf das Finden von Segmenten; dabei entspricht "Up" einer hohen Wahrscheinlichkeit und "Down" einer geringen Wahrscheinlichkeit.
target_value	*string*	Wenn dieser Wert nicht angegeben wird, nimmt er für Flags den Wert "True" (Wahr) an.
max_rules	*Integer*	Die maximale Anzahl der Segmente ausschließlich des Rests.
min_group_size	*Integer*	Mindestsegmentgröße.
min_group_size_pct	*number*	Mindestsegmentgröße als Prozentsatz.
confidence_level	*number*	Mindestschwellenwert, den ein Eingabefeld aufweist, um die Wahrscheinlichkeit eines Treffers zu verbessern (Lift), damit es zu einer Segmentdefinition hinzugefügt werden kann.
max_segments_per_rule	*Integer*	
mode	Simple Expert	
bin_method	EqualWidth EqualCount	
bin_count	*number*	
max_models_per_cycle	*Integer*	Suchbreite für Listen.
max_rules_per_cycle	*Integer*	Suchbreite für Segmentregeln.

Tabelle 124. Eigenschaften von "decisionlistnode" (Forts.)		
Eigenschaften von decision-listnode	**Werte**	**Eigenschaftsbeschreibung**
segment_growth	*number*	
include_missing	*Flag*	
final_results_only	*Flag*	
reuse_fields	*Flag*	Ermöglicht die Wiederverwendung von Attributen (Eingabefelder, die in Regeln vorkommen).
max_alternatives	*Integer*	
calculate_raw_propensi-ties	*Flag*	
calculate_adjusted_pro-pensities	*Flag*	
adjusted_propensity_par-tition	Test Validation	

discriminantnode, Eigenschaften

Die Diskriminanzanalyse macht strengere Annahmen als die logistische Regression, kann aber eine wertvolle Alternative oder Ergänzung zu einer logistischen Regressionsanalyse sein, wenn diese Annahmen erfüllt sind.

Beispiel

```
node = stream.create("discriminant", "My node")
node.setPropertyValue("target", "custcat")
node.setPropertyValue("use_partitioned_data", False)
node.setPropertyValue("method", "Stepwise")
```

Tabelle 125. discriminantnode, Eigenschaften		
Eigenschaften von discrimi-nantnode	**Werte**	**Eigenschaftsbeschreibung**
target	*Feld*	Diskriminanzmodelle erfordern ein einzelnes Zielfeld und eines oder mehrere Eingabefelder. Gewichtungs- und Häufigkeitsfelder werden nicht verwendet. Weitere Informationen finden Sie im Thema „Allgemeine Eigenschaften von Modellierungsknoten" auf Seite 225.
method	Enter Stepwise	
mode	Simple Expert	

Tabelle 125. discriminantnode, Eigenschaften (Forts.)		
Eigenschaften von discriminantnode	**Werte**	**Eigenschaftsbeschreibung**
prior_probabilities	AllEqual ComputeFromSizes	
covariance_matrix	WithinGroups SeparateGroups	
means	*Flag*	Statistikoptionen im Dialogfeld "Erweiterte Ausgabe".
univariate_anovas	*Flag*	
box_m	*Flag*	
within_group_covariance	*Flag*	
within_groups_correlation	*Flag*	
separate_groups_covariance	*Flag*	
total_covariance	*Flag*	
fishers	*Flag*	
unstandardized	*Flag*	
casewise_results	*Flag*	Klassifizierungsoptionen im Dialogfeld "Erweiterte Ausgabe".
limit_to_first	*number*	Der Standardwert ist 10.
summary_table	*Flag*	
leave_one_classification	*Flag*	
combined_groups	*Flag*	
separate_groups_covariance	*Flag*	Matrizenoption **Gruppenspezifische Kovarianzmatrix**
territorial_map	*Flag*	
combined_groups	*Flag*	Plotoption **Kombinierte Gruppen**.
separate_groups	*Flag*	Plotoption **Gruppenspezifisch**.
summary_of_steps	*Flag*	
F_pairwise	*Flag*	

Tabelle 125. discriminantnode, Eigenschaften (Forts.)		
Eigenschaften von discriminantnode	**Werte**	**Eigenschaftsbeschreibung**
stepwise_method	WilksLambda UnexplainedVariance MahalanobisDistance SmallestF RaosV	
V_to_enter	*number*	
criteria	UseValue UseProbability	
F_value_entry	*number*	Der Standardwert ist 3,84.
F_value_removal	*number*	Der Standardwert ist 2,71.
probability_entry	*number*	Der Standardwert ist 0,05.
probability_removal	*number*	Der Standardwert ist 0,10.
calculate_variable_importance	*Flag*	
calculate_raw_propensities	*Flag*	
calculate_adjusted_propensities	*Flag*	
adjusted_propensity_partition	Test Validation	

extensionmodelnode, Eigenschaften

Mit dem Erweiterungsmodellknoten können Sie Scripts in R oder Python for Spark ausführen, um Ergebnisse zu erstellen und ein Scoring durchzuführen.

Beispiel für Python for Spark

```
#### script example for Python for Spark
import modeler.api
stream = modeler.script.stream()
node = stream.create("extension_build", "extension_build")
node.setPropertyValue("syntax_type", "Python")

build_script = """
import json
import spss.pyspark.runtime
```

```
from pyspark.mllib.regression import LabeledPoint
from pyspark.mllib.linalg import DenseVector
from pyspark.mllib.tree import DecisionTree

cxt = spss.pyspark.runtime.getContext()
df = cxt.getSparkInputData()
schema = df.dtypes[:]

target = "Drug"
predictors = ["Age","BP","Sex","Cholesterol","Na","K"]

def metaMap(row,schema):
    col = 0
    meta = []
    for (cname, ctype) in schema:
        if ctype == 'string':
            meta.append(set([row[col]]))
        else:
            meta.append((row[col],row[col]))
        col += 1
    return meta

def metaReduce(meta1,meta2,schema):
    col = 0
    meta = []
    for (cname, ctype) in schema:
        if ctype == 'string':
            meta.append(meta1[col].union(meta2[col]))
        else:
            meta.append((min(meta1[col][0],meta2[col][0]),max(meta1[col][1],meta2[col][1])))
        col += 1
    return meta

metadata = df.rdd.map(lambda row: metaMap(row,schema)).reduce(lambda x,y:metaReduce(x,y,schema))

def setToList(v):
    if isinstance(v,set):
        return list(v)
    return v

metadata = map(lambda x: setToList(x), metadata)
print metadata

lookup = {}
for i in range(0,len(schema)):
    lookup[schema[i][0]] = i

def row2LabeledPoint(dm,lookup,target,predictors,row):
    target_index = lookup[target]
    tval = dm[target_index].index(row[target_index])
    pvals = []
    for predictor in predictors:
        predictor_index = lookup[predictor]
        if isinstance(dm[predictor_index],list):
            pval = dm[predictor_index].index(row[predictor_index])
        else:
            pval = row[predictor_index]
        pvals.append(pval)
    return LabeledPoint(tval,DenseVector(pvals))

# count number of target classes
predictorClassCount = len(metadata[lookup[target]])

# define function to extract categorical predictor information from datamodel
def getCategoricalFeatureInfo(dm,lookup,predictors):
    info = {}
    for i in range(0,len(predictors)):
        predictor = predictors[i]
        predictor_index = lookup[predictor]
        if isinstance(dm[predictor_index],list):
            info[i] = len(dm[predictor_index])
    return info

# convert dataframe to an RDD containing LabeledPoint
lps = df.rdd.map(lambda row: row2LabeledPoint(metadata,lookup,target,predictors,row))

treeModel = DecisionTree.trainClassifier(
    lps,
    numClasses=predictorClassCount,
    categoricalFeaturesInfo=getCategoricalFeatureInfo(metadata, lookup, predictors),
    impurity='gini',
    maxDepth=5,
```

```
    maxBins=100)

_outputPath = cxt.createTemporaryFolder()
treeModel.save(cxt.getSparkContext(), _outputPath)
cxt.setModelContentFromPath("TreeModel", _outputPath)
cxt.setModelContentFromString("model.dm",json.dumps(metadata), mimeType="application/json")\
   .setModelContentFromString("model.structure",treeModel.toDebugString())

"""

node.setPropertyValue("python_build_syntax", build_script)
```

Beispiel für R

```
#### script example for R
node.setPropertyValue("syntax_type", "R")
node.setPropertyValue("r_build_syntax", """modelerModel <- lm(modelerData$Na~modelerData$K,mo
delerData)
modelerDataModel
modelerModel
""")
```

Tabelle 126. extensionmodelnode, Eigenschaften

Eigenschaften von extensionmodelnode	Werte	Eigenschaftsbeschreibung
syntax_type	*R* *Python*	Gibt das Script an, das ausgeführt wird - R oder Python (R ist der Standardwert).
r_build_syntax	*string*	R-Scriptsyntax für die Modellerstellung.
r_score_syntax	*string*	R-Scriptsyntax für das Modellscoring.
python_build_syntax	*string*	Python-Scriptsyntax für die Modellerstellung.
python_score_syntax	*string*	Python-Scriptsyntax für das Modellscoring.
convert_flags	`StringsAndDoubles` `LogicalValues`	Option zum Konvertieren von Flagfeldern.
convert_missing	*Flag*	Option zum Konvertieren fehlender Werte in RNA-Werte.
convert_datetime	*Flag*	Option zum Konvertieren von Variablen mit Datums- oder Datums-/Zeitformaten in R-Datums-/Zeitformate.
convert_datetime_class	POSIXct POSIXlt	Optionen, die angeben, in welches Format Variablen mit Datums- oder Datums-/Zeitformaten konvertiert werden.
output_html	*Flag*	Option für die Anzeige von Diagrammen im R-Modellnugget.
output_text	*Flag*	Option zum Schreiben von R-Konsolentext auf eine Registerkarte des R-Modellnuggets.

factornode, Eigenschaften

Der Faktor/PCA-Knoten bietet leistungsstarke Datenreduktionsverfahren zur Verringerung der Komplexität der Daten. Die Hauptkomponentenanalyse (PCA) findet lineare Kombinationen der Eingabefelder, die die Varianz im gesamten Set der Felder am besten erfassen, wenn die Komponenten orthogonal (senkrecht) zueinander sind. Mit der Faktorenanalyse wird versucht, die zugrunde liegenden Faktoren zu bestimmen, die die Korrelationsmuster innerhalb eines Sets beobachteter Felder erklären. Bei beiden Ansätzen besteht das Ziel darin, eine kleinere Zahl abgeleiteter Felder zu finden, mit denen die Informationen in der ursprünglichen Menge der Felder effektiv zusammengefasst werden können.

Beispiel

```
node = stream.create("factor", "My node")
# "Fields" tab
node.setPropertyValue("custom_fields", True)
node.setPropertyValue("inputs", ["BP", "Na", "K"])
node.setPropertyValue("partition", "Test")
# "Model" tab
node.setPropertyValue("use_model_name", True)
node.setPropertyValue("model_name", "Factor_Age")
node.setPropertyValue("use_partitioned_data", False)
node.setPropertyValue("method", "GLS")
# Expert options
node.setPropertyValue("mode", "Expert")
node.setPropertyValue("complete_records", True)
node.setPropertyValue("matrix", "Covariance")
node.setPropertyValue("max_iterations", 30)
node.setPropertyValue("extract_factors", "ByFactors")
node.setPropertyValue("min_eigenvalue", 3.0)
node.setPropertyValue("max_factor", 7)
node.setPropertyValue("sort_values", True)
node.setPropertyValue("hide_values", True)
node.setPropertyValue("hide_below", 0.7)
# "Rotation" section
node.setPropertyValue("rotation", "DirectOblimin")
node.setPropertyValue("delta", 0.3)
node.setPropertyValue("kappa", 7.0)
```

Tabelle 127. factornode, Eigenschaften

Eigenschaften von `factornode`	**Werte**	**Eigenschaftsbeschreibung**
`inputs`	*[field1 ... FeldN]*	PCA-/Faktormodelle verwenden eine Liste mit Eingabefeldern, jedoch kein Ziel. Gewichtungs- und Häufigkeitsfelder werden nicht verwendet. Weitere Informationen finden Sie im Thema „Allgemeine Eigenschaften von Modellierungsknoten" auf Seite 225.

Tabelle 127. factornode, Eigenschaften (Forts.)		
Eigenschaften von `factornode`	**Werte**	**Eigenschaftsbeschreibung**
method	PC ULS GLS ML PAF Alpha Image	
mode	Simple Expert	
max_iterations	*number*	
complete_records	*Flag*	
matrix	Correlation Covariance	
extract_factors	ByEigenvalues ByFactors	
min_eigenvalue	*number*	
max_factor	*number*	
rotation	None Varimax DirectOblimin Equamax Quartimax Promax	

Tabelle 127. factornode, Eigenschaften (Forts.)		
Eigenschaften von `factornode`	**Werte**	**Eigenschaftsbeschreibung**
`delta`	*number*	Wenn Sie `DirectOblimin` als Rotationsdatentyp auswählen, können Sie einen Wert für `delta` festlegen. Wenn Sie keinen Wert festlegen, wird für `delta` der Standardwert verwendet.
`kappa`	*number*	Wenn Sie `Promax` als Rotationsdatentyp auswählen, können Sie einen Wert für `kappa` festlegen. Wenn Sie keinen Wert festlegen, wird für `kappa` der Standardwert verwendet.
`sort_values`	*Flag*	
`hide_values`	*Flag*	
`hide_below`	*number*	

featureselectionnode, Eigenschaften

Der Merkmalauswahlknoten sichtet die Eingabefelder, um auf der Grundlage einer Reihe von Kriterien (z. B. dem Prozentsatz der fehlenden Werte) zu entscheiden, ob diese entfernt werden sollen. Anschließend erstellt er eine Wichtigkeitsrangfolge der verbleibenden Eingaben in Bezug auf ein angegebenes Ziel. Beispiel: Angenommen, Sie haben ein Dataset mit Hunderten potenzieller Eingaben. Welche davon sind voraussichtlich für die Modellierung von medizinischen Behandlungsergebnissen von Bedeutung?

Beispiel

```
node = stream.create("featureselection", "My node")
node.setPropertyValue("screen_single_category", True)
node.setPropertyValue("max_single_category", 95)
node.setPropertyValue("screen_missing_values", True)
node.setPropertyValue("max_missing_values", 80)
node.setPropertyValue("criteria", "Likelihood")
node.setPropertyValue("unimportant_below", 0.8)
node.setPropertyValue("important_above", 0.9)
node.setPropertyValue("important_label", "Check Me Out!")
node.setPropertyValue("selection_mode", "TopN")
node.setPropertyValue("top_n", 15)
```

Ein detaillierteres Beispiel, mit dem ein Merkmalauswahlmodell erstellt und angewendet wird, finden Sie in „Beispiel für Standalone-Script: Generieren eines Merkmalauswahlmodells” auf Seite 5.

Tabelle 128. featureselectionnode, Eigenschaften

featureselectionnode -Eigenschaften	**Werte**	**Eigenschaftsbeschreibung**
target	*Feld*	Merkmalauswahlmodelle teilen Prädiktoren relativ zum angegebenen Ziel in Ränge ein. Gewichtungs- und Häufigkeitsfelder werden nicht verwendet. Weitere Informationen finden Sie im Thema „Allgemeine Eigenschaften von Modellierungsknoten” auf Seite 225.
screen_single_category	*Flag*	Bei True wird ein Screening der Felder durchgeführt, bei denen zu viele Datensätze (im Verhältnis zur Gesamtzahl der Datensätze) in dieselbe Kategorie fallen.
max_single_category	*number*	Gibt den Schwellenwert an, der verwendet wird, wenn screen_single_category aufTrue gesetzt ist.
screen_missing_values	*Flag*	Bei True wird ein Screening der Felder durchgeführt, die zu viele fehlende Werte (ausgedrückt als Prozentsatz der Gesamtzahl an Datensätzen) aufweisen.
max_missing_values	*number*	
screen_num_categories	*Flag*	Bei True wird ein Screening der Felder durchgeführt, die zu viele Kategorien im Verhältnis zur Gesamtzahl der Datensätze aufweisen.
max_num_categories	*number*	
screen_std_dev	*Flag*	Bei True wird ein Screening der Felder durchgeführt, deren Standardabweichung kleiner-gleich dem angegebenen Mindestwert ist.
min_std_dev	*number*	
screen_coeff_of_var	*Flag*	Bei True wird ein Screening der Felder durchgeführt, deren Varianzkoeffizient kleiner-gleich dem angegebenen Mindestwert ist.
min_coeff_of_var	*number*	
criteria	Pearson Likelihood CramersV Lambda	Wenn kategoriale Prädiktoren hinsichtlich eines kategorialen Ziels nach Rängen geordnet werden, wird hier das Maß angegeben, auf dem der Wert für die Wichtigkeit beruht.

Tabelle 128. featureselectionnode, Eigenschaften (Forts.)		
featureselectionnode -Eigenschaften	**Werte**	**Eigenschaftsbeschreibung**
unimportant_below	*number*	Gibt die *p*-Schwellenwerte an, die verwendet werden, um Variablen als "bedeutsam", "marginal" bzw. "unbedeutend" eingestuft werden. Zulässig sind Werte von 0,0 bis 1,0.
important_above	*number*	Zulässig sind Werte von 0,0 bis 1,0.
unimportant_label	*string*	Gibt die Beschriftung für die Rangstufe "unbedeutsam" an.
marginal_label	*string*	
important_label	*string*	
selection_mode	ImportanceLevel ImportanceValue TopN	
select_important	*Flag*	Wenn selection_mode auf ImportanceLevel gesetzt ist, wird hier angegeben, ob bedeutsame Felder ausgewählt werden sollen.
select_marginal	*Flag*	Wenn selection_mode auf ImportanceLevel gesetzt ist, wird hier angegeben, ob marginale Felder ausgewählt werden sollen.
select_unimportant	*Flag*	Wenn selection_mode auf ImportanceLevel gesetzt ist, wird hier angegeben, ob unbedeutende Felder ausgewählt werden sollen.
importance_value	*number*	Wenn selection_mode auf ImportanceValue gesetzt ist, wird hier der zu verwendende Trennwert angegeben. Zulässig sind Werte von 0 bis 100.
top_n	*Integer*	Wenn selection_mode auf TopN gesetzt ist, wird hier der zu verwendende Trennwert angegeben. Zulässig sind Werte von 0 bis 1000.

genlinnode, Eigenschaften

Das verallgemeinerte lineare Modell erweitert das allgemeine lineare Modell so, dass die abhängige Variable über eine angegebene Verknüpfungsfunktion in linearem Zusammenhang zu den Faktoren und Kovariaten steht. Außerdem ist es mit diesem Modell möglich, dass die abhängige Variable eine von der Normalverteilung abweichende Verteilung aufweist. Es deckt die Funktionalität vieler Statistikmodelle ab, einschließlich linearer Regression, logistischer Regression, loglinearer Modelle für Häufigkeitsdaten und Überlebensmodelle mit Intervallzensierung.

Beispiel

```
node = stream.create("genlin", "My node")
node.setPropertyValue("model_type", "MainAndAllTwoWayEffects")
node.setPropertyValue("offset_type", "Variable")
node.setPropertyValue("offset_field", "Claimant")
```

Tabelle 129. genlinnode, Eigenschaften

Eigenschaften von genlinnode	**Werte**	**Eigenschaftsbeschreibung**
target	*Feld*	Verallgemeinerte lineare Modelle erfordern ein einzelnes Zielfeld, bei dem es sich um ein nominales oder ein Flagfeld handeln muss, und eines oder mehrere Eingabefelder. Außerdem kann ein Gewichtungsfeld angegeben werden. Weitere Informationen finden Sie im Thema „Allgemeine Eigenschaften von Modellierungsknoten” auf Seite 225.
use_weight	*Flag*	
weight_field	*Feld*	Der Feldtyp ist nur stetig.
target_represents_trials	*Flag*	
trials_type	Variable FixedValue	
trials_field	*Feld*	Der Feldttyp ist stetig, Flag oder ordinal.
trials_number	*number*	Der Standardwert ist 10.
model_type	MainEffects MainAndAllTwoWayEffects	
offset_type	Variable FixedValue	
offset_field	*Feld*	Der Feldtyp ist nur stetig.
offset_value	*number*	Muss eine reelle Zahl sein.
base_category	Last First	
include_intercept	*Flag*	
mode	Simple Expert	

Tabelle 129. genlinnode, Eigenschaften (Forts.)

Eigenschaften von genlinnode	**Werte**	**Eigenschaftsbeschreibung**
distribution	BINOMIAL GAMMA IGAUSS NEGBIN NORMAL POISSON TWEEDIE MULTINOMIAL	IGAUSS: Invers normal. NEGBIN: Negativ binomial.
negbin_para_type	Specify Estimate	
negbin_parameter	*number*	Der Standardwert ist 1. Muss eine nicht-negative reelle Zahl enthalten.
tweedie_parameter	*number*	

Tabelle 129. genlinnode, Eigenschaften (Forts.)

Eigenschaften von genlinnode	**Werte**	**Eigenschaftsbeschreibung**
link_function	IDENTITY CLOGLOG LOG LOGC LOGIT NEGBIN NLOGLOG ODDSPOWER PROBIT POWER CUMCAUCHIT CUMCLOGLOG CUMLOGIT CUMNLOGLOG CUMPROBIT	CLOGLOG: Log-Log komplementär. LOGC: Log-Komplement. NEGBIN: Negativ binomial. NLOGLOG: Log-Log negativ. CUMCAUCHIT: Cauchit (kumulativ). CUMCLOGLOG: Log-Log komplementär (kumulativ). CUMLOGIT: Logit (kumulativ). CUMNLOGLOG: Log-Log negativ (kumulativ). CUMPROBIT: Probit (kumulativ).
power	*number*	Der Wert muss eine reelle Zahl ungleich null sein.
method	Hybrid Fisher NewtonRaphson	
max_fisher_iterations	*number*	Der Standardwert ist 1; nur positive Ganzzahlen zulässig.

Tabelle 129. genlinnode, Eigenschaften (Forts.)

Eigenschaften von genlinnode	**Werte**	**Eigenschaftsbeschreibung**
scale_method	MaxLikelihoodEstimate Deviance PearsonChiSquare FixedValue	
scale_value	*number*	Der Standardwert ist 1; muss größer als 0 sein.
covariance_matrix	ModelEstimator RobustEstimator	
max_iterations	*number*	Der Standardwert ist 100; nur nicht negative Ganzzahlen.
max_step_halving	*number*	Der Standardwert ist 5; nur positive Ganzzahlen.
check_separation	*Flag*	
start_iteration	*number*	Der Standardwert ist 20; nur positive Ganzzahlen zulässig.
estimates_change	*Flag*	
estimates_change_min	*number*	Der Standardwert ist 1E-006; nur positive Zahlen zulässig.
estimates_change_type	Absolute Relative	
loglikelihood_change	*Flag*	
loglikelihood_change_min	*number*	Nur positive Zahlen zulässig.
loglikelihood_change_type	Absolute Relative	
hessian_convergence	*Flag*	
hessian_convergence_min	*number*	Nur positive Zahlen zulässig.
hessian_convergence_type	Absolute Relative	
case_summary	*Flag*	
contrast_matrices	*Flag*	
descriptive_statistics	*Flag*	
estimable_functions	*Flag*	

Tabelle 129. genlinnode, Eigenschaften (Forts.)		
Eigenschaften von genlinnode	**Werte**	**Eigenschaftsbeschreibung**
model_info	*Flag*	
iteration_history	*Flag*	
goodness_of_fit	*Flag*	
print_interval	*number*	Der Standardwert ist 1; muss eine positive Ganzzahl sein.
model_summary	*Flag*	
lagrange_multiplier	*Flag*	
parameter_estimates	*Flag*	
include_exponential	*Flag*	
covariance_estimates	*Flag*	
correlation_estimates	*Flag*	
analysis_type	TypeI TypeIII TypeIAndTypeIII	
statistics	Wald LR	
citype	Wald Profile	
tolerancelevel	*number*	Der Standardwert ist 0,0001.
confidence_interval	*number*	Der Standardwert ist 95.
loglikelihood_function	Full Kernel	
singularity_tolerance	1E-007 1E-008 1E-009 1E-010 1E-011 1E-012	

Tabelle 129. genlinnode, Eigenschaften (Forts.)		
Eigenschaften von genlinnode	**Werte**	**Eigenschaftsbeschreibung**
value_order	Ascending Descending DataOrder	
calculate_variable_importance	*Flag*	
calculate_raw_propensities	*Flag*	
calculate_adjusted_propensities	*Flag*	
adjusted_propensity_partition	Test Validation	

Eigenschaften von "glmmnode"

Verallgemeinerte lineare gemischte Modelle (GLMM - Generalized Linear Mixed Models) erweitern lineare Modelle so, dass das Ziel nicht normalverteilt zu sein braucht und über eine angegebene Verknüpfungsfunktion in einer linearen Beziehung zu den Faktoren und Kovariaten steht und die Beobachtungen korreliert werden können. Verallgemeinerte lineare gemischte Modelle decken eine breite Palette verschiedener Modelle ab, von einfacher linearer Regression bis hin zu komplexen Mehrebenenmodellen für nicht normalverteilte Longitudinaldaten.

Tabelle 130. Eigenschaften von "glmmnode"		
Eigenschaften von glmmnode	**Werte**	**Eigenschaftsbeschreibung**
residual_subject_spec	*strukturiert*	Die Wertekombination der angegebenen kategorialen Felder, die Subjekte innerhalb des Datasets eindeutig definieren.
repeated_measures	*strukturiert*	Felder zur Ermittlung von wiederholten Beobachtungen.
residual_group_spec	[*field1 ... FeldN]*	Felder, die unabhängige Sätze von Kovarianzparametern für wiederholte Effekte definieren.

Tabelle 130. Eigenschaften von "glmmnode" (Forts.)

Eigenschaften von glmmnode	Werte	Eigenschaftsbeschreibung
residual_covariance_type	Diagonal AR1 ARMA11 COMPOUND_SYMMETRY IDENTITY TOEPLITZ UNSTRUCTURED VARIANCE_COMPONENTS	Definiert die Kovarianzstruktur für Residuen.
custom_target	*Flag*	Gibt an, ob das im vorgeordneten Knoten definierte Ziel (false) oder das im Feld target_field festgelegte benutzerdefinierte Ziel (true) verwendet werden soll.
target_field	*Feld*	Als Ziel zu verwendendes Feld, wenn custom_target auf true gesetzt ist.
use_trials	*Flag*	Gibt an, ob zusätzliche Felder oder Werte zur Angabe der Anzahl an Tests verwendet werden sollen, wenn es sich bei der Zielantwort um eine Reihe von Ereignissen handelt, die während Tests auftreten. Der Standardwert ist false.
use_field_or_value	Field Value	Gibt an, ob die Anzahl an Test in einem Feld (Standard) oder als Wert angegeben werden soll.
trials_field	*Feld*	Feld zur Angabe der Anzahl an Tests.
trials_value	*Integer*	Wert zur Angabe der Anzahl an Tests. Wenn angegeben, ist der Minimalwert 1.
use_custom_target_reference	*Flag*	Gibt an, ob eine benutzerdefinierte Referenzkategorie für ein kategoriales Ziel verwendet werden soll. Der Standardwert ist false.
target_reference_value	*string*	Zu verwendende Referenzkategorie, wenn use_custom_target_reference auf true gesetzt ist.

Tabelle 130. Eigenschaften von "glmmnode" (Forts.)

Eigenschaften von glmmnode	Werte	Eigenschaftsbeschreibung
dist_link_combination	Nominal Logit GammaLog BinomialLogit PoissonLog BinomialProbit NegbinLog BinomialLogC Custom	Allgemeine Modelle für die Verteilung von Werten für das Ziel. Wählen Sie Custom aus, um einen Verteilungstyp aus der von target_distribution bereitgestellten Liste festzulegen.
target_distribution	Normal Binomial Multinomial Gamma Inverse NegativeBinomial Poisson	Verteilung von Werten für das Ziel, wenn dist_link_combination auf Custom gesetzt ist.

Tabelle 130. Eigenschaften von "glmmnode" (Forts.)

Eigenschaften von glmmnode	Werte	Eigenschaftsbeschreibung
link_function_type	Identity LogC Log CLOGLOG Logit NLOGLOG PROBIT POWER CAUCHIT	Verknüpfungsfunktion zum Verknüpfen von Zielwerten mit Prädiktoren. Wenn target_distribution auf einen Binomial festgelegt ist, können Sie eine beliebige der aufgeführten Verknüpfungsfunktionen verwenden. Wenn target_distribution auf einen Multinomial festgelegt ist, können Sie CLOGLOG, CAUCHIT, LOGIT, NLOGLOG oder PROBIT verwenden. Wenn target_distribution auf einen anderen Wert als Binomial oder Multinomial festgelegt ist, können Sie IDENTITY, LOG oder POWER verwenden.
link_function_param	*number*	Für die Verknüpfungsfunktion zu verwendender Parameterwert. Wird nur verwendet, wenn normal_link_function oder link_function_type auf POWER gesetzt ist.
use_predefined_inputs	*Flag*	Gibt an, ob Felder mit festen Effekten als vorgeordnete Eingabefelder (true) oder als Felder aus fixed_effects_list (false) definiert sein sollen. Der Standardwert ist false.
fixed_effects_list	*strukturiert*	Wenn use_predefined_inputs auf false gesetzt ist, werden die Eingabefelder als Felder für feste Effekte verwendet.
use_intercept	*Flag*	Wenn true gesetzt ist (Standardeinstellung), wird der konstante Term in das Modell einbezogen.
random_effects_list	*strukturiert*	Liste von Feldern, die als zufällige Effekte festgelegt werden.
regression_weight_field	*Feld*	Zur Analysegewichtung zu verwendendes Feld.

Tabelle 130. Eigenschaften von "glmmnode" (Forts.)

Eigenschaften von glmmnode	Werte	Eigenschaftsbeschreibung
use_offset	None offset_value offset_field	Gibt an, wie der Offset festgelegt wird. Lautet der Wert None, wird kein Offset verwendet.
offset_value	*number*	Für den Offset zu verwendender Wert, wenn use_offset auf offset_value gesetzt ist.
offset_field	*Feld*	Für den Offsetwert zu verwendendes Feld, wenn use_offset auf offset_field gesetzt ist.
target_category_order	Ascending Descending Data	Sortierreihenfolge für kategoriale Ziele. Der Wert Data gibt an, dass die Sortierreihenfolge der Daten verwendet wird. Die Standardeinstellung ist Ascending.
inputs_category_order	Ascending Descending Data	Sortierreihenfolge für kategoriale Prädiktoren. Der Wert Data gibt an, dass die Sortierreihenfolge der Daten verwendet wird. Die Standardeinstellung ist Ascending.
max_iterations	*Integer*	Die maximale Anzahl der Iterationen, die im Algorithmus vorgenommen werden. Eine nicht negative Ganzzahl. Der Standardwert ist 100.
confidence_level	*Integer*	Konfidenzniveau für die Berechnung von Intervallschätzungen der Modellkoeffizienten. Eine nicht negative Ganzzahl. Der Maximalwert ist 100 und der Standardwert ist 95.
degrees_of_freedom_method	Fixed Varied	Gibt an, wie Freiheitsgrade für Signifikanztests berechnet werden.
test_fixed_effects_coefficients	Model Robust	Methode zur Berechnung der Kovarianzmatrix für Parameterschätzungen.
use_p_converge	*Flag*	Option für Parameterkonvergenz.
p_converge	*number*	Leerzeichen oder ein beliebiger positiver Wert.
p_converge_type	Absolute Relative	
use_l_converge	*Flag*	Option für Log-Likelihood-Konvergenz.

Tabelle 130. Eigenschaften von "glmmnode" (Forts.)		
Eigenschaften von glmmnode	**Werte**	**Eigenschaftsbeschreibung**
l_converge	*number*	Leerzeichen oder ein beliebiger positiver Wert.
l_converge_type	Absolute Relative	
use_h_converge	*Flag*	Option für Konvergenz der Hesse-Matrix.
h_converge	*number*	Leerzeichen oder ein beliebiger positiver Wert.
h_converge_type	Absolute Relative	
max_fisher_steps	*Integer*	
singularity_tolerance	*number*	
use_model_name	*Flag*	Gibt an, ob für das Modell ein angepasster Name (true) oder der vom System generierte Name (false) angegeben werden soll. Der Standardwert ist false.
model_name	*string*	Gibt den zu verwendenden Modellnamen an, wenn use_model_name auf true gesetzt ist.
confidence	onProbability onIncrease	Grundlage für die Berechnung des Konfidenzwerts für das Scoring: höchste vorhergesagte Wahrscheinlichkeit oder Differenz zwischen der höchsten und zweithöchsten vorhergesagten Wahrscheinlichkeit.
score_category_probabilities	*Flag*	Auf true gesetzt, werden vorhergesagte Wahrscheinlichkeiten für kategoriale Ziele generiert. Die Standardeinstellung ist false.
max_categories	*Integer*	Wenn score_category_probabilities auf true gesetzt ist, wird hier die maximale Anzahl der zu speichernden Kategorien festgelegt.
score_propensity	*Flag*	Auf true gesetzt, werden Propensity-Scores für Flagzielfelder generiert, die die Wahrscheinlichkeit angeben, mit der das Feld den Wert "true" haben wird.
emeans	*strukturiert*	Für jedes kategoriale Feld aus der Liste mit festen Effekten wird hier angegeben, ob geschätzte Randmittel generiert werden sollen.

Tabelle 130. Eigenschaften von "glmmnode" (Forts.)		
Eigenschaften von glmmnode	**Werte**	**Eigenschaftsbeschreibung**
covariance_list	*strukturiert*	Für jedes stetige Feld aus der Liste mit festen Effekten wird hier angegeben, ob der Mittelwert oder ein benutzerdefinierter Wert für die Berechnung der geschätzten Randmittel verwendet werden soll.
mean_scale	Original Transformed	Gibt an, ob geschätzte Randmittel anhand der ursprünglichen Skala des Ziels (Standard) oder anhand der Transformation der Verknüpfungsfunktion berechnet werden sollen.
comparison_adjustment_method	LSD SEQBONFERRONI SEQSIDAK	Zu verwendende Anpassungsmethode bei Hypothesentests mit mehreren Kontrasten.

Eigenschaften von "gle"

Ein GLE-Modell erweitert lineare Modelle so, dass das Ziel nicht normalverteilt zu sein braucht und über eine angegebene Verknüpfungsfunktion in einer linearen Beziehung zu den Faktoren und Kovariaten steht und die Beobachtungen korreliert werden können. Verallgemeinerte lineare gemischte Modelle decken eine breite Palette verschiedener Modelle ab, von einfacher linearer Regression bis hin zu komplexen Mehrebenenmodellen für nicht normalverteilte Longitudinaldaten.

Tabelle 131. Eigenschaften von "gle"		
Eigenschaften von gle	**Werte**	**Eigenschaftsbeschreibung**
custom_target	*Flag*	Gibt an, ob das im vorgeordneten Knoten definierte Ziel (false) oder das im Feld target_field festgelegte benutzerdefinierte Ziel (true) verwendet werden soll.
target_field	*Feld*	Als Ziel zu verwendendes Feld, wenn custom_target auf true gesetzt ist.
use_trials	*Flag*	Gibt an, ob zusätzliche Felder oder Werte zur Angabe der Anzahl an Tests verwendet werden sollen, wenn es sich bei der Zielantwort um eine Reihe von Ereignissen handelt, die während Tests auftreten. Der Standardwert ist false.
use_trials_field_or_value	Field Value	Gibt an, ob die Anzahl an Test in einem Feld (Standard) oder als Wert angegeben werden soll.
trials_field	*Feld*	Feld zur Angabe der Anzahl an Tests.
trials_value	*Integer*	Wert zur Angabe der Anzahl an Tests. Wenn angegeben, ist der Minimalwert 1.

Tabelle 131. Eigenschaften von "gle" (Forts.)

Eigenschaften von gle	Werte	Eigenschaftsbeschreibung
use_custom_target_reference	*Flag*	Gibt an, ob eine benutzerdefinierte Referenzkategorie für ein kategoriales Ziel verwendet werden soll. Der Standardwert ist false.
target_reference_value	*string*	Zu verwendende Referenzkategorie, wenn use_custom_target_reference auf true gesetzt ist.
dist_link_combination	NormalIdentity GammaLog PoissonLog NegbinLog TweedieIdentity NominalLogit BinomialLogit BinomialProbit BinomialLogC CUSTOM	Allgemeine Modelle für die Verteilung von Werten für das Ziel. Wählen Sie CUSTOM aus, um einen Verteilungstyp aus der von target_distribution bereitgestellten Liste festzulegen.
target_distribution	Normal Binomial Multinomial Gamma INVERSE_GAUSS NEG_BINOMIAL Poisson TWEEDIE UNKNOWN	Verteilung von Werten für das Ziel, wenn dist_link_combination auf Custom gesetzt ist.

Tabelle 131. Eigenschaften von "gle" (Forts.)

Eigenschaften von gle	Werte	Eigenschaftsbeschreibung
link_function_type	UNKNOWN IDENTITY LOG LOGIT PROBIT COMPL_LOG_LOG POWER LOG_COMPL NEG_LOG_LOG ODDS_POWER NEG_BINOMIAL GEN_LOGIT CUMUL_LOGIT CUMUL_PROBIT CUMUL_COMPL_LOG_LOG CUMUL_NEG_LOG_LOG CUMUL_CAUCHIT	Verknüpfungsfunktion zum Herstellen von Beziehungen zwischen Zielwerten und Prädiktoren. Wenn target_distribution auf Binomial gesetzt ist, können Sie Folgendes verwenden. UNKNOWN IDENTITY LOG LOGIT PROBIT COMPL_LOG_LOG POWER LOG_COMPL NEG_LOG_LOG ODDS_POWER Wenn target_distribution auf NEG_BINOMIAL gesetzt ist, können Sie Folgendes verwenden. NEG_BINOMIAL. Wenn target_distribution auf UNKNOWN gesetzt ist, können Sie Folgendes verwenden. GEN_LOGIT CUMUL_LOGIT CUMUL_PROBIT CUMUL_COMPL_LOG_LOG CUMUL_NEG_LOG_LOG CUMUL_CAUCHIT

Tabelle 131. Eigenschaften von "gle" (Forts.)

Eigenschaften von gle	Werte	Eigenschaftsbeschreibung
link_function_param	*number*	Zu verwendender Wert für Tweedie-Parameter. Wird nur verwendet, wenn normal_link_function oder link_function_type auf POWER gesetzt ist.
tweedie_param	*number*	Für die Verknüpfungsfunktion zu verwendender Parameterwert. Wird nur verwendet, wenn dist_link_combination auf TweedieIdentity oder link_function_type auf TWEEDIE gesetzt ist.
use_predefined_inputs	*Flag*	Gibt an, ob die Felder, die in einer vorgeordneten Ebene als Eingabefelder definiert wurden (true), oder die Felder in fixed_effects_list (false) als Felder für Modelleffekte verwendet werden sollen.
model_effects_list	*strukturiert*	Wenn use_predefined_inputs auf false gesetzt ist, werden die Eingabefelder als Felder für Modelleffekte verwendet.
use_intercept	*Flag*	Wenn true gesetzt ist (Standardeinstellung), wird der konstante Term in das Modell einbezogen.
regression_weight_field	*Feld*	Zur Analysegewichtung zu verwendendes Feld.
use_offset	None Value Variable	Gibt an, wie der Offset festgelegt wird. Lautet der Wert None, wird kein Offset verwendet.
offset_value	*number*	Für den Offset zu verwendender Wert, wenn use_offset auf offset_value gesetzt ist.
offset_field	*Feld*	Für den Offsetwert zu verwendendes Feld, wenn use_offset auf offset_field gesetzt ist.
target_category_order	Ascending Descending	Sortierreihenfolge für kategoriale Ziele. Der Standardwert ist Ascending.
inputs_category_order	Ascending Descending	Sortierreihenfolge für kategoriale Prädiktoren. Der Standardwert ist Ascending.
max_iterations	*Integer*	Die maximale Anzahl der Iterationen, die im Algorithmus vorgenommen werden. Eine nicht negative Ganzzahl. Der Standardwert ist 100.

Tabelle 131. Eigenschaften von "gle" (Forts.)

Eigenschaften von gle	Werte	Eigenschaftsbeschreibung
confidence_level	*number*	Konfidenzniveau für die Berechnung von Intervallschätzungen der Modellkoeffizienten. Eine nicht negative Ganzzahl. Der Maximalwert ist 100 und der Standardwert ist 95.
test_fixed_effects_coeffecients	Model Robust	Methode zur Berechnung der Kovarianzmatrix für Parameterschätzungen.
detect_outliers	*Flag*	Wenn dieser Wert wahr ist, sucht der Algorithmus für alle Verteilungen mit Ausnahme von multinomialen Verteilungen einflussreiche Ausreißer.
conduct_trend_analysis	*Flag*	Wenn dieser Wert wahr ist, führt der Algorithmus Trendanalysen für das Streudiagramm durch.
estimation_method	FISHER_SCORING NEWTON_RAPHSON HYBRID	Dient zur Angabe des Algorithmus für Maximum-Likelihood-Schätzungen.
max_fisher_iterations	*Integer*	Wenn für estimation_method der Wert FISHER_SCORING verwendet wird, die maximale Anzahl Iterationen. Minimum: 0, Maximum: 20.
scale_parameter_method	MLE FIXED DEVIANCE PEARSON_CHISQUARE	Dient zur Angabe der für die Schätzung des Skalenparameters verwendeten Methode.
scale_value	*number*	Nur verfügbar, wenn scale_parameter_method auf Fixed gesetzt ist.
negative_binomial_method	MLE FIXED	Dient zur Angabe der für die Schätzung des Hilfsparameters für negative Binomialverteilung verwendeten Methode.
negative_binomial_value	*number*	Nur verfügbar, wenn negative_binomial_method auf Fixed gesetzt ist.
non_neg_least_squares	*Flag*	Gibt an, ob die nicht negative Methode der kleinsten Quadrate ausgeführt werden soll. Der Standardwert ist false.
use_p_converge	*Flag*	Option für Parameterkonvergenz.
p_converge	*number*	Leerzeichen oder ein beliebiger positiver Wert.
p_converge_type	*Flag*	True = absolut, False = relativ
use_l_converge	*Flag*	Option für Log-Likelihood-Konvergenz.
l_converge	*number*	Leerzeichen oder ein beliebiger positiver Wert.

Tabelle 131. Eigenschaften von "gle" (Forts.)		
Eigenschaften von gle	**Werte**	**Eigenschaftsbeschreibung**
l_converge_type	*Flag*	True = absolut, False = relativ
use_h_converge	*Flag*	Option für Konvergenz der Hesse-Matrix.
h_converge	*number*	Leerzeichen oder ein beliebiger positiver Wert.
h_converge_type	*Flag*	True = absolut, False = relativ
max_iterations	*Integer*	Die maximale Anzahl der Iterationen, die im Algorithmus vorgenommen werden. Eine nicht negative Ganzzahl. Der Standardwert ist 100.
sing_tolerance	*Integer*	
use_model_selection	*Flag*	Aktiviert die Steuerelemente für den Parameterschwellenwert und die Modellauswahlmethode.

method	`LASSO`
	`ELASTIC_NET`
	`FORWARD_STEPWISE`
	`RIDGE`
		Legt die verwendete Modellauswahlmethode bzw. bei Verwendung von Ridge die verwendete Regularisierungsmethode fest.

detect_two_way_interactions	*Flag*	Bei True ermittelt das Modell automatisch Zweiwegeinteraktionen zwischen Eingabefeldern. Dieses Steuerelement sollte nur aktiviert werden, wenn es sich bei dem Modell nur um Haupteffekte handelt (d. h., wenn der Benutzer keine Effekte höherer Ordnung erstellt hat) und wenn method als 'Schrittweise vorwärts', 'Lasso' oder 'Elastic Net' ausgewählt wurde.
automatic_penalty_params	*Flag*	Nur verfügbar, wenn die Modellauswahl method Lasso oder Elastic Net ist. Mithilfe dieser Funktion können Sie Penalisierungsparameter eingeben, die zur Methode für die Variablenauswahl "Lasso" oder "Elastic Net" gehören. Bei True werden die Standardwerte verwendet. Bei False werden die Penalisierungsparameter aktiviert und es können benutzerdefinierte Werte eingegeben werden.
lasso_penalty_param	*number*	Nur verfügbar, wenn die Modellauswahl method Lasso oder Elastic Net ist und automatic_penalty_params False ist. Dient zur Angabe des Penalisierungsparameterwerts für Lasso.

Tabelle 131. Eigenschaften von "gle" (Forts.)

Eigenschaften von gle	Werte	Eigenschaftsbeschreibung
elastic_net_penalty_param1	*number*	Nur verfügbar, wenn die Modellauswahl method Lasso oder Elastic Net ist und automatic_penalty_params False ist. Dient zur Angabe des Penalisierungsparameterwerts für Elastic Net-Parameter 1.
elastic_net_penalty_param2	*number*	Nur verfügbar, wenn die Modellauswahl method Lasso oder Elastic Net ist und automatic_penalty_params False ist. Dient zur Angabe des Penalisierungsparameterwerts für Elastic Net-Parameter 2.
probability_entry	*number*	Nur verfügbar, wenn method auf "Schrittweise vorwärts" gesetzt ist. Dient zur Angabe des Signifikanzniveaus des Kriteriums "F-Statistik" für das Einschließen von Effekten.
probability_removal	*number*	Nur verfügbar, wenn method auf "Schrittweise vorwärts" gesetzt ist. Dient zur Angabe des Signifikanzniveaus des Kriteriums "F-Statistik" für das Ausschließen von Effekten.
use_max_effects	*Flag*	Nur verfügbar, wenn method auf "Schrittweise vorwärts" gesetzt ist. Aktiviert das Steuerelement max_effects. Bei False sollte die Standardanzahl eingeschlossener Effekte gleich der Gesamtzahl der für das Modell angegebenen Effekte minus konstantem Term sein.
max_effects	*Integer*	Geben Sie die maximale Anzahl der Effekte an, wenn Sie die Erstellungsmethode "Schrittweise vorwärts" verwenden.
use_max_steps	*Flag*	Aktiviert das Steuerelement max_steps. Bei False sollte die Standardanzahl der Schritte dreimal so hoch wie die Anzahl der für das Modell angegebenen Effekte ohne konstanten Term sein.
max_steps	*Integer*	Geben Sie die maximale Anzahl der Schritte an, die ausgeführt werden sollen, wenn die Erstellung method Schrittweise vorwärts verwendet wird.
use_model_name	*Flag*	Gibt an, ob ein angepasster Name für das Modell (true) oder der vom System generierte Name (false) angegeben werden soll. Der Standardwert ist false.
model_name	*string*	Gibt den zu verwendenden Modellnamen an, wenn use_model_name auf true gesetzt ist.

Tabelle 131. Eigenschaften von "gle" (Forts.)		
Eigenschaften von gle	**Werte**	**Eigenschaftsbeschreibung**
usePI	*Flag*	Bei true wird der Prädiktoreinfluss berechnet.

kmeansnode, Eigenschaften

Der K-Means-Knoten teilt das Dataset in unterschiedliche Gruppen (oder Cluster) auf. Bei dieser Methode wird eine festgelegte Anzahl von Clustern definiert, den Clustern werden iterativ Datensätze zugewiesen und die Clusterzentren werden angepasst, bis eine weitere Verfeinerung keine wesentliche Verbesserung des Modells mehr darstellen würde. Statt zu versuchen, ein Ergebnis vorherzusagen, versucht *K*-Means mithilfe eines als "nicht überwachtes Lernen" bezeichneten Verfahrens Muster im Set der Eingabefelder zu entdecken.

Beispiel

```
node = stream.create("kmeans", "My node")
# "Fields" tab
node.setPropertyValue("custom_fields", True)
node.setPropertyValue("inputs", ["Cholesterol", "BP", "Drug", "Na", "K",
"Age"])
# "Model" tab
node.setPropertyValue("use_model_name", True)
node.setPropertyValue("model_name", "Kmeans_allinputs")
node.setPropertyValue("num_clusters", 9)
node.setPropertyValue("gen_distance", True)
node.setPropertyValue("cluster_label", "Number")
node.setPropertyValue("label_prefix", "Kmeans_")
node.setPropertyValue("optimize", "Speed")
# "Expert" tab
node.setPropertyValue("mode", "Expert")
node.setPropertyValue("stop_on", "Custom")
node.setPropertyValue("max_iterations", 10)
node.setPropertyValue("tolerance", 3.0)
node.setPropertyValue("encoding_value", 0.3)
```

Tabelle 132. kmeansnode, Eigenschaften		
Eigenschaften von kmeansnode	**Werte**	**Eigenschaftsbeschreibung**
inputs	[*field1 ... FeldN]*	K-Means-Modelle führen eine Clusteranalyse an einer Menge von Eingabefeldern durch, verwenden jedoch kein Zielfeld. Gewichtungs- und Häufigkeitsfelder werden nicht verwendet. Weitere Informationen finden Sie im Thema „Allgemeine Eigenschaften von Modellierungsknoten” auf Seite 225.
num_clusters	*number*	
gen_distance	*Flag*	
cluster_label	String Number	

Tabelle 132. kmeansnode, Eigenschaften (Forts.)		
Eigenschaften von kmeansnode	**Werte**	**Eigenschaftsbeschreibung**
label_prefix	*string*	
mode	Simple Expert	
stop_on	Default Custom	
max_iterations	*number*	
tolerance	*number*	
encoding_value	*number*	
optimize	Speed Memory	Dient zur Angabe, ob die Modellerstellung in Bezug auf die Geschwindigkeit oder den Speicher optimiert werden soll.

kmeansasnode, Eigenschaften

K-Means ist einer der am häufigsten verwendeten Clusteralgorithmen. Er teilt Datenpunkte in eine vordefinierte Anzahl Cluster auf. Der K-Means-AS-Knoten in SPSS Modeler ist in Spark implementiert. Details zum K-Means-Algorithums finden Sie in https://spark.apache.org/docs/2.2.0/ml-clustering.html. Beachten Sie, dass der K-Means-AS-Knoten automatisch eine 1-aus-n-Codierung für kategoriale Variablen durchführt.

Tabelle 133. kmeansasnode, Eigenschaften		
Eigenschaften von kmeansasnode	**Werte**	**Eigenschaftsbeschreibung**
roleUse	*string*	Geben Sie predefined an, um vordefinierte Rollen zu verwenden, oder custom, um benutzerdefinierte Feldzuweisungen zu verwenden. Der Standardwert ist predefined.
autoModel	*boolesch*	Geben Sie true an, wenn Sie den Standardnamen ($S-prediction) für das neue generierte Scoring-Feld verwenden möchten, oder false, um einen angepassten Namen zu verwenden. Der Standardwert ist true.
features	*Feld*	Liste der Feldnamen für die Eingabe, wenn die Eigenschaft roleUse auf custom gesetzt ist.
name	*string*	Der Name des neuen generierten Scoring-Felds, wenn die Eigenschaft autoModel auf false gesetzt ist.

Tabelle 133. kmeansasnode, Eigenschaften (Forts.)		
Eigenschaften von kmeansas-node	**Werte**	**Eigenschaftsbeschreibung**
clustersNum	*Integer*	Die Anzahl der zu erstellenden Cluster. Der Standardwert ist 5.
initMode	*string*	Der Initialisierungsalgorithmus. Mögliche Werte sind k-means|| oder random. Der Standardwert ist k-means||.
initSteps	*Integer*	Die Anzahl der Initialisierungsschritte, wenn initMode auf k-means|| gesetzt ist. Der Standardwert ist 2.
advancedSettings	*boolesch*	Geben Sie true an, um die folgenden vier Eigenschaften verfügbar zu machen. Der Standardwert ist false.
maxIteration	*Integer*	Maximale Anzahl Iterationen für das Clustering. Der Standardwert ist 20.
tolerance	*string*	Die Toleranz zum Stoppen der Iterationen. Mögliche Einstellungen sind 1.0E-1, 1.0E-2, ..., 1.0E-6. Der Standardwert ist 1.0E-4.
setSeed	*boolesch*	Geben Sie true an, um einen benutzerdefinierten Startwert für Zufallszahlen zu verwenden. Der Standardwert ist false.
randomSeed	*Integer*	Der benutzerdefinierte Startwert für Zufallszahlen, wenn die Eigenschaft setSeed auf true gesetzt ist.

knnnode, Eigenschaften

Der Knoten "*k*-Nächste Nachbarn" (KNN) verknüpft einen neuen Fall mit der Kategorie oder dem Wert der *k* Objekte, die ihm im Prädiktorraum am nächsten liegen, wobei *k* eine ganze Zahl ist. Ähnliche Fälle liegen nah beieinander und Fälle mit geringer Ähnlichkeit sind weit voneinander entfernt.

Beispiel

```
node = stream.create("knn", "My node")
# Objectives tab
node.setPropertyValue("objective", "Custom")
# Settings tab - Neighbors panel
node.setPropertyValue("automatic_k_selection", False)
node.setPropertyValue("fixed_k", 2)
node.setPropertyValue("weight_by_importance", True)
# Settings tab - Analyze panel
node.setPropertyValue("save_distances", True)
```

Tabelle 134. knnnode, Eigenschaften		
Eigenschaften von knnnode	**Werte**	**Eigenschaftsbeschreibung**
analysis	PredictTarget IdentifyNeighbors	
objective	Balance Speed Accuracy Custom	
normalize_ranges	*Flag*	
use_case_labels	*Flag*	Kontrollkästchen markieren, um nächste Option zu aktivieren.
case_labels_field	*Feld*	
identify_focal_cases	*Flag*	Kontrollkästchen markieren, um nächste Option zu aktivieren.
focal_cases_field	*Feld*	
automatic_k_selection	*Flag*	
fixed_k	*Integer*	Nur aktiviert, wenn automatic_k_selectio auf False gesetzt ist.
minimum_k	*Integer*	Nur aktiviert, wenn automatic_k_selectio auf True gesetzt ist.
maximum_k	*Integer*	
distance_computation	Euclidean CityBlock	
weight_by_importance	*Flag*	
range_predictions	Mean Median	
perform_feature_selection	*Flag*	
forced_entry_inputs	[*field1 ... FeldN]*	
stop_on_error_ratio	*Flag*	
number_to_select	*Integer*	
minimum_change	*number*	
validation_fold_assign_by_field	*Flag*	

Tabelle 134. knnnode, Eigenschaften (Forts.)		
Eigenschaften von knnnode	**Werte**	**Eigenschaftsbeschreibung**
number_of_folds	*Integer*	Nur aktiviert, wenn validation_fold_assign_by_field auf False eingestellt ist.
set_random_seed	*Flag*	
random_seed	*number*	
folds_field	*Feld*	Nur aktiviert, wenn validation_fold_assign_by_field auf True eingestellt ist.
all_probabilities	*Flag*	
save_distances	*Flag*	
calculate_raw_propensities	*Flag*	
calculate_adjusted_propensities	*Flag*	
adjusted_propensity_partition	Test Validation	

kohonennode, Eigenschaften

Der Kohonen-Knoten erstellt eine Art von neuronalem Netz, das verwendet werden kann, um ein Clustering des Datasets in einzelne Gruppen vorzunehmen. Wenn das Netz voll trainiert ist, sollten ähnliche Datensätze auf der Ausgabekarte eng nebeneinander stehen, während Datensätze, die sich unterscheiden, weit voneinander entfernt sein sollten. Die Zahl der von jeder Einheit im Modellnugget erfassten Beobachtungen gibt Aufschluss über die starken Einheiten. Dadurch wird ein Eindruck von der ungefähren Zahl der Cluster vermittelt.

Beispiel

```
node = stream.create("kohonen", "My node")
# "Model" tab
node.setPropertyValue("use_model_name", False)
node.setPropertyValue("model_name", "Symbolic Cluster")
node.setPropertyValue("stop_on", "Time")
node.setPropertyValue("time", 1)
node.setPropertyValue("set_random_seed", True)
node.setPropertyValue("random_seed", 12345)
node.setPropertyValue("optimize", "Speed")
# "Expert" tab
node.setPropertyValue("mode", "Expert")
node.setPropertyValue("width", 3)
node.setPropertyValue("length", 3)
node.setPropertyValue("decay_style", "Exponential")
node.setPropertyValue("phase1_neighborhood", 3)
node.setPropertyValue("phase1_eta", 0.5)
node.setPropertyValue("phase1_cycles", 10)
node.setPropertyValue("phase2_neighborhood", 1)
node.setPropertyValue("phase2_eta", 0.2)
node.setPropertyValue("phase2_cycles", 75)
```

Tabelle 135. kohonennode, Eigenschaften		
Eigenschaften von kohonennode	**Werte**	**Eigenschaftsbeschreibung**
inputs	*[field1 ... FeldN]*	Kohonen-Modelle verwenden eine Liste mit Eingabefeldern, jedoch kein Ziel. Häufigkeits- und Gewichtungsfelder werden nicht verwendet. Weitere Informationen finden Sie im Thema „Allgemeine Eigenschaften von Modellierungsknoten" auf Seite 225.
continue	*Flag*	
show_feedback	*Flag*	
stop_on	Default Time	
time	*number*	
optimize	Speed Memory	Dient zur Angabe, ob die Modellerstellung in Bezug auf die Geschwindigkeit oder den Speicher optimiert werden soll.
cluster_label	*Flag*	
mode	Simple Expert	
width	*number*	
length	*number*	
decay_style	Linear Exponential	
phase1_neighborhood	*number*	
phase1_eta	*number*	
phase1_cycles	*number*	
phase2_neighborhood	*number*	
phase2_eta	*number*	
phase2_cycles	*number*	

Eigenschaften von "linearnode"

Lineare Regressionsmodelle sagen ein stetiges Ziel basierend auf linearen Beziehungen zwischen dem Ziel und einem oder mehreren Prädiktoren voraus.

Beispiel

```
node = stream.create("linear", "My node")
# Build Options tab - Objectives panel
node.setPropertyValue("objective", "Standard")
# Build Options tab - Model Selection panel
node.setPropertyValue("model_selection", "BestSubsets")
node.setPropertyValue("criteria_best_subsets", "ASE")
# Build Options tab - Ensembles panel
node.setPropertyValue("combining_rule_categorical", "HighestMeanProbability")
```

Tabelle 136. Eigenschaften von "linearnode"

Eigenschaften von linearnode	**Werte**	**Eigenschaftsbeschreibung**
target	*Feld*	Gibt ein einzelnes Zielfeld an.
inputs	[*field1 ... FeldN]*	Im Modell verwendete Prädiktorfelder.
continue_training_existing_model	*Flag*	
objective	Standard Bagging Boosting psm	psm wird für sehr große Datasets verwendet und macht eine Serververbindung erforderlich.
use_auto_data_preparation	*Flag*	
confidence_level	*number*	
model_selection	ForwardStepwise BestSubsets None	
criteria_forward_stepwise	AICC Fstatistics AdjustedRSquare ASE	
probability_entry	*number*	
probability_removal	*number*	
use_max_effects	*Flag*	
max_effects	*number*	
use_max_steps	*Flag*	

Tabelle 136. Eigenschaften von "linearnode" (Forts.)		
Eigenschaften von linearnode	**Werte**	**Eigenschaftsbeschreibung**
max_steps	*number*	
criteria_best_subsets	AICC AdjustedRSquare ASE	
combining_rule_continuous	Mean Median	
component_models_n	*number*	
use_random_seed	*Flag*	
random_seed	*number*	
use_custom_model_name	*Flag*	
custom_model_name	*string*	
use_custom_name	*Flag*	
custom_name	*string*	
tooltip	*string*	
keywords	*string*	
annotation	*string*	

Eigenschaften von "linearasnode"

Lineare Regressionsmodelle sagen ein stetiges Ziel basierend auf linearen Beziehungen zwischen dem Ziel und einem oder mehreren Prädiktoren voraus.

Tabelle 137. Eigenschaften von "linearasnode"		
Eigenschaften von linearasnode	**Werte**	**Eigenschaftsbeschreibung**
target	*Feld*	Gibt ein einzelnes Zielfeld an.
inputs	[*field1 ... FeldN*]	Im Modell verwendete Prädiktorfelder.
weight_field	*Feld*	Im Modell verwendetes Analysefeld.
custom_fields	*Flag*	Der Standardwert ist TRUE.
intercept	*Flag*	Der Standardwert ist TRUE.
detect_2way_interaction	*Flag*	Gibt an, ob Zweiwegeinteraktion berücksichtigt werden soll. Der Standardwert ist TRUE.

Tabelle 137. Eigenschaften von "linearasnode" (Forts.)

Eigenschaften von linearasnode	**Werte**	**Eigenschaftsbeschreibung**
cin	*number*	Das Konfidenzintervall, das zur Berechnung von Modellkoeffizienten verwendet wird. Geben Sie einen Wert größer als 0 und kleiner als 100 ein. Der Standardwert ist 95.
factor_order	ascending descending	Sortierreihenfolge für kategoriale Prädiktoren. Der Standardwert ist ascending.
var_select_method	ForwardStepwise BestSubsets none	Die zu verwendende Modellauswahlmethode. Der Standardwert ist ForwardStepwise.
criteria_for_forward_stepwise	AICC Fstatistics AdjustedRSquare ASE	Die Statistik, die zur Bestimmung verwendet wird, ob ein Effekt in das Modell aufgenommen oder aus diesem ausgeschlossen werden soll. Der Standardwert ist AdjustedRSquare.
pin	*number*	Der Effekt mit dem kleinsten p-Wert unterhalb dieses angegebenen pin-Schwellenwerts wird dem Modell hinzugefügt. Der Standardwert ist 0.05.
pout	*number*	Alle Effekte im Modell mit einem p-Wert größer als dieser angegebene pout-Schwellenwert werden entfernt. Der Standardwert ist 0.10.
use_custom_max_effects	*Flag*	Gibt an, ob die maximale Anzahl von Effekten im endgültigen Modell verwendet werden soll. Der Standardwert ist FALSE.
max_effects	*number*	Maximale Anzahl der Effekte, die im endgültigen Modell verwendet werden sollen. Der Standardwert ist 1.
use_custom_max_steps	*Flag*	Gibt an, ob die maximale Anzahl von Schritten verwendet werden soll. Der Standardwert ist FALSE.
max_steps	*number*	Die maximale Anzahl von Schritten vor dem Stoppen des schrittweisen Algorithmus. Der Standardwert ist 1.

Tabelle 137. Eigenschaften von "linearasnode" (Forts.)		
Eigenschaften von linearas-node	**Werte**	**Eigenschaftsbeschreibung**
criteria_for_best_sub-sets	AICC AdjustedRSquare ASE	Der zu verwendende Kriterienmodus. Der Standardwert ist AdjustedRS-quare.

logregnode, Eigenschaften

Die logistische Regression ist ein statistisches Verfahren zur Klassifizierung von Datensätzen auf der Grundlage der Werte von Eingabefeldern. Sie ist analog zur linearen Regression, verwendet aber ein kategoriales Zielfeld anstelle eines numerischen Bereichs.

Beispiel für ein multinomiales Modell

```
node = stream.create("logreg", "My node")
# "Fields" tab
node.setPropertyValue("custom_fields", True)
node.setPropertyValue("target", "Drug")
node.setPropertyValue("inputs", ["BP", "Cholesterol", "Age"])
node.setPropertyValue("partition", "Test")
# "Model" tab
node.setPropertyValue("use_model_name", True)
node.setPropertyValue("model_name", "Log_reg Drug")
node.setPropertyValue("use_partitioned_data", True)
node.setPropertyValue("method", "Stepwise")
node.setPropertyValue("logistic_procedure", "Multinomial")
node.setPropertyValue("multinomial_base_category", "BP")
node.setPropertyValue("model_type", "FullFactorial")
node.setPropertyValue("custom_terms", [["BP", "Sex"], ["Age"], ["Na", "K"]])
node.setPropertyValue("include_constant", False)
# "Expert" tab
node.setPropertyValue("mode", "Expert")
node.setPropertyValue("scale", "Pearson")
node.setPropertyValue("scale_value", 3.0)
node.setPropertyValue("all_probabilities", True)
node.setPropertyValue("tolerance", "1.0E-7")
# "Convergence..." section
node.setPropertyValue("max_iterations", 50)
node.setPropertyValue("max_steps", 3)
node.setPropertyValue("l_converge", "1.0E-3")
node.setPropertyValue("p_converge", "1.0E-7")
node.setPropertyValue("delta", 0.03)
# "Output..." section
node.setPropertyValue("summary", True)
node.setPropertyValue("likelihood_ratio", True)
node.setPropertyValue("asymptotic_correlation", True)
node.setPropertyValue("goodness_fit", True)
node.setPropertyValue("iteration_history", True)
node.setPropertyValue("history_steps", 3)
node.setPropertyValue("parameters", True)
node.setPropertyValue("confidence_interval", 90)
node.setPropertyValue("asymptotic_covariance", True)
node.setPropertyValue("classification_table", True)
# "Stepping" options
node.setPropertyValue("min_terms", 7)
node.setPropertyValue("use_max_terms", True)
```

```
node.setPropertyValue("max_terms", 10)
node.setPropertyValue("probability_entry", 3)
node.setPropertyValue("probability_removal", 5)
node.setPropertyValue("requirements", "Containment")
```

Beispiel für ein binomiales Modell

```
node = stream.create("logreg", "My node")
# "Fields" tab
node.setPropertyValue("custom_fields", True)
node.setPropertyValue("target", "Cholesterol")
node.setPropertyValue("inputs", ["BP", "Drug", "Age"])
node.setPropertyValue("partition", "Test")
# "Model" tab
node.setPropertyValue("use_model_name", False)
node.setPropertyValue("model_name", "Log_reg Cholesterol")
node.setPropertyValue("multinomial_base_category", "BP")
node.setPropertyValue("use_partitioned_data", True)
node.setPropertyValue("binomial_method", "Forwards")
node.setPropertyValue("logistic_procedure", "Binomial")
node.setPropertyValue("binomial_categorical_input", "Sex")
node.setKeyedPropertyValue("binomial_input_contrast", "Sex", "Simple")
node.setKeyedPropertyValue("binomial_input_category", "Sex", "Last")
node.setPropertyValue("include_constant", False)
# "Expert" tab
node.setPropertyValue("mode", "Expert")
node.setPropertyValue("scale", "Pearson")
node.setPropertyValue("scale_value", 3.0)
node.setPropertyValue("all_probabilities", True)
node.setPropertyValue("tolerance", "1.0E-7")
# "Convergence..." section
node.setPropertyValue("max_iterations", 50)
node.setPropertyValue("l_converge", "1.0E-3")
node.setPropertyValue("p_converge", "1.0E-7")
# "Output..." section
node.setPropertyValue("binomial_output_display", "at_each_step")
node.setPropertyValue("binomial_goodness_of_fit", True)
node.setPropertyValue("binomial_iteration_history", True)
node.setPropertyValue("binomial_parameters", True)
node.setPropertyValue("binomial_ci_enable", True)
node.setPropertyValue("binomial_ci", 85)
# "Stepping" options
node.setPropertyValue("binomial_removal_criterion", "LR")
node.setPropertyValue("binomial_probability_removal", 0.2)
```

Tabelle 138. logregnode, Eigenschaften

Eigenschaften von logregnode	**Werte**	**Eigenschaftsbeschreibung**
target	*Feld*	Logistische Regressionsmodelle erfordern ein einzelnes Zielfeld und eines oder mehrere Eingabefelder. Häufigkeits- und Gewichtungsfelder werden nicht verwendet. Weitere Informationen finden Sie im Thema „Allgemeine Eigenschaften von Modellierungsknoten” auf Seite 225.
logistic_procedure	Binomial Multinomial	

Tabelle 138. logregnode, Eigenschaften (Forts.)

Eigenschaften von logregnode	**Werte**	**Eigenschaftsbeschreibung**
include_constant	*Flag*	
mode	Simple Expert	
method	Enter Stepwise Forwards Backwards BackwardsStepwise	
binomial_method	Enter Forwards Backwards	
model_type	MainEffects FullFactorial Custom	Wenn als Modelltyp FullFactorial festgelegt ist, werden keine Schrittmethoden ausgeführt, auch wenn diese ebenfalls angegeben wurden. Stattdessen wird die Methode Enter verwendet. Wenn der Modelltyp auf Custom gesetzt ist, jedoch keine benutzerdefinierten Felder angegeben wurden, wird ein Haupteffektmodell erstellt.
custom_terms	[*[BP Sex][BP][Age]*]	
multinomial_base_category	*string*	Gibt an, wie die Referenzkategorie bestimmt wird.
binomial_categorical_input	*string*	

Tabelle 138. logregnode, Eigenschaften (Forts.)

Eigenschaften von logregnode	**Werte**	**Eigenschaftsbeschreibung**
binomial_input_contrast	Indicator Simple Difference Helmert Repeated Polynomial Deviation	Verschlüsselte Eigenschaft für kategoriale Eingaben, die angibt, wie der Kontrast bestimmt wird.
binomial_input_category	First Last	Verschlüsselte Eigenschaft für kategoriale Eingaben, die angibt, wie die Referenzkategorie bestimmt wird.
scale	None UserDefined Pearson Deviance	
scale_value	*number*	
all_probabilities	*Flag*	
tolerance	1.0E-5 1.0E-6 1.0E-7 1.0E-8 1.0E-9 1.0E-10	
min_terms	*number*	
use_max_terms	*Flag*	
max_terms	*number*	

Tabelle 138. logregnode, Eigenschaften (Forts.)

Eigenschaften von logregnode	**Werte**	**Eigenschaftsbeschreibung**
entry_criterion	Score LR	
removal_criterion	LR Wald	
probability_entry	*number*	
probability_removal	*number*	
binomial_probability_entry	*number*	
binomial_probability_removal	*number*	
requirements	HierarchyDiscrete HierarchyAll Containment None	
max_iterations	*number*	
max_steps	*number*	
p_converge	1.0E-4 1.0E-5 1.0E-6 1.0E-7 1.0E-8 0	
l_converge	1.0E-1 1.0E-2 1.0E-3 1.0E-4 1.0E-5 0	

Tabelle 138. logregnode, Eigenschaften (Forts.)		
Eigenschaften von logregnode	**Werte**	**Eigenschaftsbeschreibung**
delta	*number*	
iteration_history	*Flag*	
history_steps	*number*	
summary	*Flag*	
likelihood_ratio	*Flag*	
asymptotic_correlation	*Flag*	
goodness_fit	*Flag*	
parameters	*Flag*	
confidence_interval	*number*	
asymptotic_covariance	*Flag*	
classification_table	*Flag*	
stepwise_summary	*Flag*	
info_criteria	*Flag*	
monotonicity_measures	*Flag*	
binomial_output_display	at_each_step at_last_step	
binomial_goodness_of_fit	*Flag*	
binomial_parameters	*Flag*	
binomial_iterati-on_history	*Flag*	
binomial_classificati-on_plots	*Flag*	
binomial_ci_enable	*Flag*	
binomial_ci	*number*	
binomial_residual	outliers all	
binomial_residual_enable	*Flag*	
binomial_outlier_thres-hold	*number*	
binomial_classificati-on_cutoff	*number*	

Tabelle 138. logregnode, Eigenschaften (Forts.)		
Eigenschaften von logregnode	**Werte**	**Eigenschaftsbeschreibung**
binomial_removal_criterion	LR Wald Conditional	
calculate_variable_importance	*Flag*	
calculate_raw_propensities	*Flag*	

lsvmnode, Eigenschaften

Der Knoten "Linear Support Vector Machine" (LSVM) ermöglicht die Klassifizierung von Daten in eine von zwei Gruppen ohne Überanpassung. LSVM ist linear und eignet sich gut für umfangreiche Datasets, beispielsweise solche mit einer großen Anzahl von Datensätzen.

Tabelle 139. lsvmnode, Eigenschaften		
Eigenschaften von lsvmnode	**Werte**	**Eigenschaftsbeschreibung**
intercept	*Flag*	Schließt den konstanten Term in das Modell ein. Der Standardwert ist True.
target_order	Ascending Descending	Gibt die Sortierreihenfolge für das kategoriale Ziel an. Wird für stetige Ziele ignoriert. Die Standardeinstellung ist Ascending.
precision	*number*	Nur verwendet, wenn es sich beim Messniveau des Zielfelds um Continuous (Stetig) handelt. Gibt den Parameter für die Sensivität gegenüber dem Verlust für Regression an. Minimum ist 0, es gibt kein Maximum. Der Standardwert ist0.1.
exclude_missing_values	*Flag*	Bei True wird ein Datensatz ausgeschlossen, wenn ein einzelner Wert fehlt. Der Standardwert ist False.
penalty_function	L1 L2	Gibt den Typ der verwendeten Penalisierungsfunktion an. Der Standardwert ist L2.
lambda	*number*	Penalisierungsparameter (Regularisierung).

Tabelle 139. lsvmnode, Eigenschaften (Forts.)		
Eigenschaften von `lsvmnode`	**Werte**	**Eigenschaftsbeschreibung**
`calculate_variable_importance`	*Flag*	Bei Modellen, die zu einem angemessenen Maß an Wichtigkeit führen, zeigt diese Option ein Diagramm an, in dem der relative Einfluss der einzelnen Prädiktoren bei der Modellschätzung angegeben wird. Beachten Sie, dass die Berechnung des Variableneinflusses bei einigen Modellen längere Zeit in Anspruch nehmen kann, insbesondere bei der Arbeit mit großen Datasets, und daher bei einigen Modellen standardmäßig inaktiviert ist. Der Variableneinfluss ist für Entscheidungslistenmodelle nicht verfügbar.

neuralnetnode, Eigenschaften

Wichtig: In dieser Version ist eine neuere Fassung des neuronalen Netzmodellierungsknotens mit erweiterten Funktionen verfügbar, die im nächsten Abschnitt beschrieben wird (*neuralnetwork*). Sie können zwar auch weiterhin Modelle mit der Vorgängerversion erstellen und scoren, doch empfehlen wir die Aktualisierung Ihrer Scripts zur Verwendung der neuen Version. Details der vorherigen Version werden hier aus Referenzgründen aufbewahrt.

Beispiel

```
node = stream.create("neuralnet", "My node")
# "Fields" tab
node.setPropertyValue("custom_fields", True)
node.setPropertyValue("targets", ["Drug"])
node.setPropertyValue("inputs", ["Age", "Na", "K", "Cholesterol", "BP"])
# "Model" tab
node.setPropertyValue("use_partitioned_data", True)
node.setPropertyValue("method", "Dynamic")
node.setPropertyValue("train_pct", 30)
node.setPropertyValue("set_random_seed", True)
node.setPropertyValue("random_seed", 12345)
node.setPropertyValue("stop_on", "Time")
node.setPropertyValue("accuracy", 95)
node.setPropertyValue("cycles", 200)
node.setPropertyValue("time", 3)
node.setPropertyValue("optimize", "Speed")
# "Multiple Method Expert Options" section
node.setPropertyValue("m_topologies", "5 30 5; 2 20 3, 1 10 1")
node.setPropertyValue("m_non_pyramids", False)
node.setPropertyValue("m_persistence", 100)
```

Tabelle 140. neuralnetnode, Eigenschaften		
Eigenschaften von neuralnetnode	**Werte**	**Eigenschaftsbeschreibung**
targets	[*field1 ... FeldN]*	Der Netzknoten erwartet mindestens ein Zielfeld und mindestens ein Eingabefeld. Häufigkeits- und Gewichtungsfelder werden ignoriert. Weitere Informationen finden Sie im Thema „Allgemeine Eigenschaften von Modellierungsknoten" auf Seite 225.
method	Quick Dynamic Multiple Prune ExhaustivePrune RBFN	
prevent_overtrain	*Flag*	
train_pct	*number*	
set_random_seed	*Flag*	
random_seed	*number*	
mode	Simple Expert	
stop_on	Default Accuracy Cycles Time	Stopping mode.
accuracy	*number*	Stoppgenauigkeit.
cycles	*number*	Zu trainierende Zyklen.
time	*number*	Dauer der Trainingsphase (Minuten)
continue	*Flag*	
show_feedback	*Flag*	
binary_encode	*Flag*	
use_last_model	*Flag*	
gen_logfile	*Flag*	

Tabelle 140. neuralnetnode, Eigenschaften (Forts.)		
Eigenschaften von neuralnetnode	**Werte**	**Eigenschaftsbeschreibung**
logfile_name	*string*	
alpha	*number*	
initial_eta	*number*	
high_eta	*number*	
low_eta	*number*	
eta_decay_cycles	*number*	
hid_layers	One Two Three	
hl_units_one	*number*	
hl_units_two	*number*	
hl_units_three	*number*	
persistence	*number*	
m_topologies	*string*	
m_non_pyramids	*Flag*	
m_persistence	*number*	
p_hid_layers	One Two Three	
p_hl_units_one	*number*	
p_hl_units_two	*number*	
p_hl_units_three	*number*	
p_persistence	*number*	
p_hid_rate	*number*	
p_hid_pers	*number*	
p_inp_rate	*number*	
p_inp_pers	*number*	
p_overall_pers	*number*	
r_persistence	*number*	
r_num_clusters	*number*	
r_eta_auto	*Flag*	
r_alpha	*number*	

Tabelle 140. neuralnetnode, Eigenschaften (Forts.)		
Eigenschaften von neuralnetnode	**Werte**	**Eigenschaftsbeschreibung**
r_eta	*number*	
optimize	Speed Memory	Dient zur Angabe, ob die Modellerstellung in Bezug auf die Geschwindigkeit oder den Speicher optimiert werden soll.
calculate_variable_importance	*Flag*	Hinweis: Die in früheren Versionen verwendete Eigenschaft sensitivity_analysis wird zugunsten dieser Eigenschaft nicht mehr verwendet. Die alte Eigenschaft wird weiterhin unterstützt, es wird jedoch calculate_variable_importance empfohlen.
calculate_raw_propensities	*Flag*	
calculate_adjusted_propensities	*Flag*	
adjusted_propensity_partition	Test Validation	

neuralnetworknode, Eigenschaften

Der Netzknoten verwendet ein vereinfachtes Modell der Art und Weise, wie ein menschliches Gehirn Informationen verarbeitet. Es funktioniert, indem eine große Anzahl miteinander verbundener einfacher Verarbeitungseinheiten simuliert wird, die abstrakten Versionen von Neuronen ähnlich sind. Neuronale Netze sind leistungsstarke Mehrzweckschätzer, für deren Training und Anwendung nur sehr geringe statistische oder mathematische Kenntnisse erforderlich sind.

Beispiel

```
node = stream.create("neuralnetwork", "My node")
# Build Options tab - Objectives panel
node.setPropertyValue("objective", "Standard")
# Build Options tab - Ensembles panel
node.setPropertyValue("combining_rule_categorical", "HighestMeanProbability")
```

Tabelle 141. neuralnetworknode, Eigenschaften		
Eigenschaften von neuralnetworknode	**Werte**	**Eigenschaftsbeschreibung**
targets	[*field1 ... FeldN]*	Gibt die Zielfelder an.
inputs	[*field1 ... FeldN]*	Im Modell verwendete Prädiktorfelder.
splits	*[field1 ... FeldN*	Gibt das Feld bzw. die Felder für die Aufteilungsmodellierung an.

Tabelle 141. neuralnetworknode, Eigenschaften (Forts.)		
Eigenschaften von neuralnetworknode	**Werte**	**Eigenschaftsbeschreibung**
use_partition	*Flag*	Wenn ein Partitionsfeld definiert ist, gewährleistet diese Option, dass nur Daten aus der Trainingspartition für die Modellerstellung verwendet werden.
continue	*Flag*	Training des bestehenden Modells fortsetzen.
objective	Standard Bagging Boosting psm	psm wird für sehr große Datasets verwendet und erfordert eine Serververbindung.
method	MultilayerPerceptron RadialBasisFunction	
use_custom_layers	*Flag*	
first_layer_units	*number*	
second_layer_units	*number*	
use_max_time	*Flag*	
max_time	*number*	
use_max_cycles	*Flag*	
max_cycles	*number*	
use_min_accuracy	*Flag*	
min_accuracy	*number*	
combining_rule_categorical	Voting HighestProbability HighestMeanProbability	
combining_rule_continuous	Mean Median	
component_models_n	*number*	
overfit_prevention_pct	*number*	
use_random_seed	*Flag*	
random_seed	*number*	

Tabelle 141. neuralnetworknode, Eigenschaften (Forts.)		
Eigenschaften von neuralnetworknode	**Werte**	**Eigenschaftsbeschreibung**
missing_values	listwiseDeletion missingValueImputation	
use_model_name	*Boolesch*	
model_name	*string*	
confidence	onProbability onIncrease	
score_category_probabilities	*Flag*	
max_categories	*number*	
score_propensity	*Flag*	
use_custom_name	*Flag*	
custom_name	*string*	
tooltip	*string*	
keywords	*string*	
annotation	*string*	

questnode, Eigenschaften

Der QUEST-Knoten bietet eine binäre Klassifizierungsmethode zum Erstellen von Entscheidungsbäumen, die dazu dient, die für große C&R-Baumanalysen erforderliche Verarbeitungszeit zu verkürzen. Gleichzeitig soll die in den Klassifizierungsbaummodellen festgestellte Tendenz verringert werden, die darin besteht, dass Eingaben bevorzugt werden, die mehr Aufteilungen erlauben. Eingabefelder können stetig (numerische Bereiche) sein, das Zielfeld muss aber kategorial sein. Alle Aufteilungen sind binär.

Beispiel

```
node = stream.create("quest", "My node")
node.setPropertyValue("custom_fields", True)
node.setPropertyValue("target", "Drug")
node.setPropertyValue("inputs", ["Age", "Na", "K", "Cholesterol", "BP"])
node.setPropertyValue("model_output_type", "InteractiveBuilder")
node.setPropertyValue("use_tree_directives", True)
node.setPropertyValue("max_surrogates", 5)
node.setPropertyValue("split_alpha", 0.03)
node.setPropertyValue("use_percentage", False)
node.setPropertyValue("min_parent_records_abs", 40)
node.setPropertyValue("min_child_records_abs", 30)
node.setPropertyValue("prune_tree", True)
node.setPropertyValue("use_std_err", True)
node.setPropertyValue("std_err_multiplier", 3)
```

Tabelle 142. questnode, Eigenschaften

Eigenschaften von questnode	Werte	Eigenschaftsbeschreibung
target	*Feld*	QUEST-Modelle erfordern ein einzelnes Ziel und eines oder mehrere Eingabefelder. Außerdem kann ein Häufigkeitsfeld angegeben werden. Weitere Informationen finden Sie im Thema „Allgemeine Eigenschaften von Modellierungsknoten" auf Seite 225.
continue_training_existing_model	*Flag*	
objective	Standard Boosting Bagging psm	psm wird für sehr große Datasets verwendet und erfordert eine Serververbindung.
model_output_type	Single InteractiveBuilder	
use_tree_directives	*Flag*	
tree_directives	*string*	
use_max_depth	Default Custom	
max_depth	*Integer*	Maximale Baumtiefe von 0 bis 1000. Wird nur bei use_max_depth = Custom verwendet.
prune_tree	*Flag*	Baum reduzieren, um zu große Anpassung zu vermeiden.
use_std_err	*Flag*	Maximale Risikendifferenz verwenden (in Standardfehler).
std_err_multiplier	*number*	Maximale Differenz.
max_surrogates	*number*	Maximale Anzahl Ersatztrenner.
use_percentage	*Flag*	
min_parent_records_pc	*number*	
min_child_records_pc	*number*	
min_parent_records_abs	*number*	
min_child_records_abs	*number*	
use_costs	*Flag*	
costs	*strukturiert*	Strukturierte Eigenschaft.

Tabelle 142. questnode, Eigenschaften (Forts.)

Eigenschaften von questnode	**Werte**	**Eigenschaftsbeschreibung**
priors	Data Equal Custom	
custom_priors	*strukturiert*	Strukturierte Eigenschaft.
adjust_priors	*Flag*	
trails	*number*	Anzahl der Komponentenmodelle für Boosting oder Bagging.
set_ensemble_method	Voting HighestProbability HighestMeanProbability	Standardkombinationsregel für kategoriale Ziele.
range_ensemble_method	Mean Median	Standardkombinationsregel für stetige Ziele.
large_boost	*Flag*	Boosting auf sehr große Datasets anwenden.
split_alpha	*number*	Signifikanzschwelle für Aufteilung.
train_pct	*number*	Set zur Verhinderung übermäßiger Anpassung.
set_random_seed	*Flag*	Option "Ergebnisse replizieren".
seed	*number*	
calculate_variable_importance	*Flag*	
calculate_raw_propensities	*Flag*	
calculate_adjusted_propensities	*Flag*	
adjusted_propensity_partition	Test Validation	

randomtrees, Eigenschaften

Der Random Trees-Knoten ähnelt dem vorhandenen C&RT-Knoten. Der Random Trees-Knoten ist jedoch so konzipiert, dass er durch die Verarbeitung großer Datenmengen eine einzige Baumstruktur erstellt und das resultierende Modell im Ausgabeviewer anzeigt, der in SPSS Modeler Version 17 hinzugefügt wurde. Der Random Trees-Knoten generiert einen Entscheidungsbaum, mit dem Sie zukünftige Beobachtungen vorhersagen oder klassifizieren können. Bei dieser Methode wird eine rekursive Partitionierung verwendet, um die Trainingsdatensätze in Segmente aufzuteilen. Dabei wird bei jedem Schritt die Unreinheit verringert. Ein Knoten im Baum wird als *rein* betrachtet, wenn 100 % der Fälle im Knoten in eine bestimmte Kategorie des Zielfelds fallen. Ziel- und Eingabefelder können numerische Bereiche oder kategorial (nominal, ordinal oder Flags) sein. Alle Aufteilungen sind binär (nur zwei Untergruppen).

Tabelle 143. randomtrees, Eigenschaften

Eigenschaften von `randomtrees`	**Werte**	**Eigenschaftsbeschreibung**
`target`	*Feld*	Im Random Trees-Knoten erfordern Modelle ein einzelnes Ziel und ein oder mehrere Eingabefelder. Außerdem kann ein Häufigkeitsfeld angegeben werden. Weitere Informationen finden Sie im Thema „Allgemeine Eigenschaften von Modellierungsknoten” auf Seite 225.
`number_of_models`	*Integer*	Legt die Anzahl der Modelle fest, die im Rahmen der Ensemblemodellierung erstellt werden sollen.
`use_number_of_predictors`	*Flag*	Legt fest, ob `number_of_predictors` verwendet wird.
`number_of_predictors`	*Integer*	Gibt die Anzahl der Prädiktoren an, die beim Erstellen von aufgeteilten Modellen verwendet werden soll.
`use_stop_rule_for_accuracy`	*Flag*	Legt fest, ob die Modellerstellung gestoppt wird, wenn die Genauigkeit nicht verbessert werden kann.
`sample_size`	*number*	Verkleinern Sie diesen Wert, um die Leistung bei der Verarbeitung sehr umfangreicher Datasets zu verbessern.

Tabelle 143. randomtrees, Eigenschaften (Forts.)

Eigenschaften von randomtrees	**Werte**	**Eigenschaftsbeschreibung**
handle_imbalanced_data	*Flag*	Wenn das Modellziel ein bestimmtes Flagergebnis ist und wenn das Verhältnis vom gewünschten Ergebnis zum unerwünschten Ergebnis sehr klein ist, sind die Daten unausgewogen und die vom Modell durchgeführte Bootstrap-Stichprobenziehung kann sich auf die Modellgenauigkeit auswirken. Aktivieren Sie die Verarbeitung unausgewogener Daten, damit das Modell einen größeren Anteil des gewünschten Ergebnisses erfasst und ein besseres Modell generiert.
use_weighted_sampling	*Flag*	Bei *False* werden Variablen für jeden Knoten zufällig mit derselben Wahrscheinlichkeit ausgewählt. Bei *True* werden Variablen gewichtet und entsprechend ausgewählt.
max_node_number	*Integer*	Maximale Anzahl der Knoten, die in einzelnen Bäumen zulässig sind. Wenn die Zahl bei der nächsten Aufteilung überschritten würde, wird der Baumaufbau gestoppt.
max_depth	*Integer*	Maximale Baumtiefe, bevor der Aufbau gestoppt wird.
min_child_node_size	*Integer*	Legt die minimale Anzahl der Datensätze fest, die nach der Aufteilung des übergeordneten Knotens in einem untergeordneten Knoten enthalten sein dürfen. Wenn ein untergeordneter Knoten weniger Datensätze als angegeben enthalten würde, wird der übergeordnete Knoten nicht aufgeteilt.
use_costs	*Flag*	
costs	*strukturiert*	Strukturierte Eigenschaft. Das Format ist eine Liste mit 3 Werten: der tatsächliche Wert, der vorhergesagte Wert und die Kosten, falls die Vorhersage falsch ist. Beispiel: tree.setPropertyValue("costs", [["drugA", "drugB", 3.0], ["drugX", "drugY", 4.0]])

Tabelle 143. randomtrees, Eigenschaften (Forts.)		
Eigenschaften von randomtrees	**Werte**	**Eigenschaftsbeschreibung**
default_cost_increase	none linear square custom	**Anmerkung:** Nur für ordinale Ziele aktiviert. Standardwerte in der Kostenmatrix festlegen.
max_pct_missing	*Integer*	Wenn der Prozentsatz der fehlenden Werte in einer Eingabe größer als der hier angegebene Wert ist, wird die Eingabe ausgeschlossen. Minimum: 0, Maximum: 100.
exclude_single_cat_pct	*Integer*	Wenn ein Kategorienwert einen höheren Prozentsatz der Datensätze als hier angegeben darstellt, wird das gesamte Feld aus der Modellerstellung ausgeschlossen. Minimum: 1, Maximum: 99.
max_category_number	*Integer*	Wenn die Anzahl der Kategorien in einem Feld diesen Wert überschreitet, wird das Feld aus der Modellerstellung ausgeschlossen. Minimum: 2.
min_field_variation	*number*	Wenn der Variationskoeffizient eines stetigen Felds kleiner ist als dieser Wert, wird das Feld aus der Modellerstellung ausgeschlossen.
num_bins	*Integer*	Wird nur verwendet, wenn die Daten aus stetigen Eingaben bestehen. Legen Sie die Anzahl der Klassen mit gleicher Häufigkeit fest, die für die Eingaben verwendet werden sollen. Optionen sind 2, 4, 5, 10, 20, 25, 50 oder 100.
topN	*Integer*	Gibt die Anzahl der aufzulistenden Regeln an. Der Standardwert ist 50, das Minimum ist 1 und das Maximum ist 1000.

regressionnode, Eigenschaften

Die lineare Regression ist ein gängiges statistisches Verfahren für die Auswertung von Daten und die Erstellung von Vorhersagen durch Anpassung einer geraden Linie oder Fläche, die die Diskrepanzen zwischen vorhergesagten und tatsächlichen Ausgabewerten minimiert.

Anmerkung: Der Regressionsknoten wird in einer zukünftigen Version durch den Linearknoten ersetzt. Es wird empfohlen, dass Sie von nun an lineare Modelle für lineare Regression verwenden.

Beispiel

```
node = stream.create("regression", "My node")
# "Fields" tab
node.setPropertyValue("custom_fields", True)
node.setPropertyValue("target", "Age")
node.setPropertyValue("inputs", ["Na", "K"])
node.setPropertyValue("partition", "Test")
node.setPropertyValue("use_weight", True)
node.setPropertyValue("weight_field", "Drug")
# "Model" tab
node.setPropertyValue("use_model_name", True)
node.setPropertyValue("model_name", "Regression Age")
node.setPropertyValue("use_partitioned_data", True)
node.setPropertyValue("method", "Stepwise")
node.setPropertyValue("include_constant", False)
# "Expert" tab
node.setPropertyValue("mode", "Expert")
node.setPropertyValue("complete_records", False)
node.setPropertyValue("tolerance", "1.0E-3")
# "Stepping..." section
node.setPropertyValue("stepping_method", "Probability")
node.setPropertyValue("probability_entry", 0.77)
node.setPropertyValue("probability_removal", 0.88)
node.setPropertyValue("F_value_entry", 7.0)
node.setPropertyValue("F_value_removal", 8.0)
# "Output..." section
node.setPropertyValue("model_fit", True)
node.setPropertyValue("r_squared_change", True)
node.setPropertyValue("selection_criteria", True)
node.setPropertyValue("descriptives", True)
node.setPropertyValue("p_correlations", True)
node.setPropertyValue("collinearity_diagnostics", True)
node.setPropertyValue("confidence_interval", True)
node.setPropertyValue("covariance_matrix", True)
node.setPropertyValue("durbin_watson", True)
```

Tabelle 144. regressionnode, Eigenschaften

Eigenschaften von `regressionnode`	**Werte**	**Eigenschaftsbeschreibung**
`target`	*Feld*	Regressionsmodelle erfordern ein einzelnes Zielfeld und eines oder mehrere Eingabefelder. Außerdem kann ein Gewichtungsfeld angegeben werden. Weitere Informationen finden Sie im Thema „Allgemeine Eigenschaften von Modellierungsknoten” auf Seite 225.

`method`	`Enter`
	`Stepwise`
	`Backwards`
	`Forwards`	

`include_constant`	*Flag*	
`use_weight`	*Flag*	

Tabelle 144. regressionnode, Eigenschaften (Forts.)		
Eigenschaften von regressionnode	**Werte**	**Eigenschaftsbeschreibung**
weight_field	*Feld*	
mode	Simple Expert	
complete_records	*Flag*	
tolerance	1.0E-1 1.0E-2 1.0E-3 1.0E-4 1.0E-5 1.0E-6 1.0E-7 1.0E-8 1.0E-9 1.0E-10 1.0E-11 1.0E-12	Verwenden Sie für Argumente doppelte Anführungszeichen.
stepping_method	useP useF	useP : Wahrscheinlichkeit von F verwenden useF: F-Wert verwenden
probability_entry	*number*	
probability_removal	*number*	
F_value_entry	*number*	
F_value_removal	*number*	
selection_criteria	*Flag*	
confidence_interval	*Flag*	
covariance_matrix	*Flag*	
collinearity_diagnostics	*Flag*	
regression_coefficients	*Flag*	

Tabelle 144. regressionnode, Eigenschaften (Forts.)		
Eigenschaften von regressionnode	**Werte**	**Eigenschaftsbeschreibung**
exclude_fields	*Flag*	
durbin_watson	*Flag*	
model_fit	*Flag*	
r_squared_change	*Flag*	
p_correlations	*Flag*	
descriptives	*Flag*	
calculate_variable_importance	*Flag*	

sequencenode, Eigenschaften

Der Sequenzknoten erkennt Assoziationsregeln in sequenziellen oder zeitorientierten Daten. Eine Sequenz ist eine Liste mit Elementsets, die in einer vorhersagbaren Reihenfolge auftreten. Beispiel: Ein Kunde, der einen Rasierer und After-Shave-Lotion kauft, kauft möglicherweise beim nächsten Einkauf Rasiercreme. Der Sequenzknoten basiert auf dem CARMA-Assoziationsregelalgorithmus, der eine effiziente bidirektionale Methode zum Suchen von Sequenzen verwendet.

Beispiel

```
node = stream.create("sequence", "My node")
# "Fields" tab
node.setPropertyValue("id_field", "Age")
node.setPropertyValue("contiguous", True)
node.setPropertyValue("use_time_field", True)
node.setPropertyValue("time_field", "Date1")
node.setPropertyValue("content_fields", ["Drug", "BP"])
node.setPropertyValue("partition", "Test")
# "Model" tab
node.setPropertyValue("use_model_name", True)
node.setPropertyValue("model_name", "Sequence_test")
node.setPropertyValue("use_partitioned_data", False)
node.setPropertyValue("min_supp", 15.0)
node.setPropertyValue("min_conf", 14.0)
node.setPropertyValue("max_size", 7)
node.setPropertyValue("max_predictions", 5)
# "Expert" tab
node.setPropertyValue("mode", "Expert")
node.setPropertyValue("use_max_duration", True)
node.setPropertyValue("max_duration", 3.0)
node.setPropertyValue("use_pruning", True)
node.setPropertyValue("pruning_value", 4.0)
node.setPropertyValue("set_mem_sequences", True)
node.setPropertyValue("mem_sequences", 5.0)
node.setPropertyValue("use_gaps", True)
node.setPropertyValue("min_item_gap", 20.0)
node.setPropertyValue("max_item_gap", 30.0)
```

Tabelle 145. sequencenode, Eigenschaften		
Eigenschaften von sequencenode	**Werte**	**Eigenschaftsbeschreibung**
id_field	*Feld*	Um ein Sequenzmodell zu erstellen, müssen Sie ein ID-Feld, ein optionales Zeitfeld und mindestens ein Inhaltsfeld angeben. Gewichtungs- und Häufigkeitsfelder werden nicht verwendet. Weitere Informationen finden Sie im Thema „Allgemeine Eigenschaften von Modellierungsknoten" auf Seite 225.
time_field	*Feld*	
use_time_field	*Flag*	
content_fields	[*field1 ... FeldN*]	
contiguous	*Flag*	
min_supp	*number*	
min_conf	*number*	
max_size	*number*	
max_predictions	*number*	
mode	Simple Expert	
use_max_duration	*Flag*	
max_duration	*number*	
use_gaps	*Flag*	
min_item_gap	*number*	
max_item_gap	*number*	
use_pruning	*Flag*	
pruning_value	*number*	
set_mem_sequences	*Flag*	
mem_sequences	*Integer*	

slrmnode, Eigenschaften

Mithilfe des Knotens für das lernfähige Antwortmodell (Self-Learning Response Model, SLRM) können Sie ein Modell erstellen, in dem das Modell anhand eines einzelnen neuen Falls oder einer kleinen Anzahl neuer Fälle neu eingeschätzt werden kann, ohne dass das Modell mit allen Daten neu trainiert werden muss.

Beispiel

```
node = stream.create("slrm", "My node")
node.setPropertyValue("target", "Offer")
```

```
node.setPropertyValue("target_response", "Response")
node.setPropertyValue("inputs", ["Cust_ID", "Age", "Ave_Bal"])
```

Tabelle 146. slrmnode, Eigenschaften

Eigenschaften von slrmnode	Werte	Eigenschaftsbeschreibung
target	*Feld*	Beim Zielfeld muss es sich um ein nominales oder ein Flagfeld handeln. Außerdem kann ein Häufigkeitsfeld angegeben werden. Weitere Informationen finden Sie im Thema „Allgemeine Eigenschaften von Modellierungsknoten" auf Seite 225.
target_response	*Feld*	Der Typ muss "Flag" sein.
continue_training_existing_model	*Flag*	
target_field_values	*Flag*	Alle verwenden: Alle Werte aus der Quelle verwenden. Angeben: Erforderliche Werte auswählen.
target_field_values_specify	*[field1 ... Feldn]*	
include_model_assessment	*Flag*	
model_assessment_random_seed	*number*	Muss eine reelle Zahl sein.
model_assessment_sample_size	*number*	Muss eine reelle Zahl sein.
model_assessment_iterations	*number*	Anzahl der Iterationen.
display_model_evaluation	*Flag*	
max_predictions	*number*	
randomization	*number*	
scoring_random_seed	*number*	
sort	Ascending Descending	Gibt an, ob die Angebote mit den höchsten oder die mit den niedrigsten Scores zuerst angezeigt werden.
model_reliability	*Flag*	
calculate_variable_importance	*Flag*	

Eigenschaften von "statisticsmodelnode"

Mithilfe des Statistics-Modellknotens können Sie Ihre Daten analysieren und bearbeiten, indem Sie IBM SPSS Statistics-Prozeduren ausführen, die PMML erzeugen. Dieser Knoten erfordert eine lizenzierte Kopie vonIBM SPSS Statistics.

Eine Beschreibung der Eigenschaften für diesen Knoten finden Sie in „Eigenschaften von "statisticsmodelnode"" auf Seite 442.

stpnode, Eigenschaften

Der STP-Knoten (Spatio-Temporal Prediction - räumliche temporale Vorhersage) verwendet Daten, die Positionsdaten, Eingabefelder für Vorhersagen (Prädiktoren), ein Zeitfeld und ein Zielfeld enthalten. Jede Position hat mehrere Zeilen in den Daten, die die Werte der einzelnen Prädiktoren zum Zeitpunkt der einzelnen Messungen darstellen. Nach der Analyse der Daten können diese zum Vorhersagen von Zielwerten an einer beliebigen Position innerhalb der in der Analyse verwendeten Shapedaten eingesetzt werden.

Tabelle 147. stpnode, Eigenschaften

Eigenschaften von stpnode	**Datentyp**	**Eigenschaftsbeschreibung**
Registerkarte **Felder**		
`target`	*Feld*	Das Zielfeld.
`location`	*Feld*	Das Feld für die Position des Modells. Nur georäumliche Felder sind zulässig.
`location_label`	*Feld*	Das in der Ausgabe zu verwendende kategoriale Feld, um die in `location` ausgewählten Positionen zu beschriften.
`time_field`	*Feld*	Das Zeitfeld für das Modell. Nur Felder mit fortlaufender Messung sind zulässig und der Speichertyp muss 'Zeit', 'Datum', 'Zeitmarke' oder 'Ganzzahl' sein.
`inputs`	*[field1 ... Feldn]*	Eine Liste von Eingabefeldern.
Registerkarte **Zeitintervalle**		

`interval_type_timestamp`	`Years`
	`Quarters`
	`Months`
	`Weeks`
	`Days`
	`Hours`
	`Minutes`
	`Seconds`	

Tabelle 147. stpnode, Eigenschaften (Forts.)

Eigenschaften von stpnode	**Datentyp**	**Eigenschaftsbeschreibung**
interval_type_date	Years Quarters Months Weeks Days	
interval_type_time	Hours Minutes Seconds	Begrenzt die Anzahl Tage pro Woche, die berücksichtigt werden, wenn der Zeitindex erstellt wird, den STP für die Berechnung verwendet.
interval_type_integer	Periods (nur Zeitindexfelder, Ganzzahlspeicherung)	Das Intervall, in das das Dataset umgewandelt wird. Die verfügbare Auswahl hängt vom Speichertyp des Felds ab, das als time_field für das Modell ausgewählt wird.
period_start	*Integer*	
start_month	January February March April May June July August September October November December	Der Monat, ab dem das Modell die Indexierung startet. Wenn z. B. March angegeben ist, der erste Datensatz im Dataset jedoch January ist, überspringt das Modell die ersten beiden Datensätze und startet die Indexierung im März.

Tabelle 147. stpnode, Eigenschaften (Forts.)

Eigenschaften von stpnode	**Datentyp**	**Eigenschaftsbeschreibung**
week_begins_on	Sunday Monday Tuesday Wednesday Thursday Friday Saturday	Der Startpunkt für den Zeitindex, der von STP aus den Daten erstellt wird.
days_per_week	*Integer*	Minimum: 1, Maximum: 7, in Inkrementen von 1.
hours_per_day	*Integer*	Die Anzahl Stunden, die das Modell in einem Tag berücksichtigt. Bei der Angabe 10 startet das Modell die Indexierung zur durch day_begins_at angegebenen Uhrzeit und setzt die Indexierung 10 Stunden fort, springt dann zum nächsten Wert, der mit dem Wert von day_begins_at übereinstimmt, usw.
day_begins_at	00:00 01:00 02:00 03:00 ... 23:00	Legt den Stundenwert fest, ab dem das Modell die Indexierung startet.

Tabelle 147. stpnode, Eigenschaften (Forts.)

Eigenschaften von stpnode	Datentyp	Eigenschaftsbeschreibung
interval_increment	1 2 3 4 5 6 10 12 15 20 30	Diese Inkrementeinstellung gilt für Minuten oder Sekunden. Sie legt fest, wo das Modell aus den Daten Indizes erstellt. Bei einem Inkrement von 30 und beim Intervalltyp seconds erstellt das Modell alle 30 Sekunden aus den Daten einen Index.
data_matches_interval	*boolesch*	Wird diese Eigenschaft auf N gesetzt, tritt die Konvertierung der Daten in den regulären Intervalltyp (interval_type) auf, bevor das Modell erstellt wird. Wenn Ihre Daten bereits im richtigen Format vorliegen und wenn interval_type und zugehörige Einstellungen mit Ihren Daten übereinstimmen, setzen Sie diese Einstellung auf Y, um die Konvertierung oder Aggregation Ihrer Daten zu verhindern. Durch das Setzen dieser Einstellung auf Y werden alle Aggregationssteuerelemente inaktiviert.

Tabelle 147. stpnode, Eigenschaften (Forts.)

Eigenschaften von stpnode	**Datentyp**	**Eigenschaftsbeschreibung**
agg_range_default	Sum Mean Min Max Median 1stQuartile 3rdQuartile	Legt die Standardaggregationsmethode fest, die für stetige Felder verwendet wird. Alle stetigen Felder, die nicht spezifisch in die angepasste Aggregation eingeschlossen werden, werden mit der hier angegebenen Methode aggregiert.
custom_agg	[[field, aggregation method],[]..] Demo: [['x5' 'FirstQuartile']['x4' 'Sum']]	Strukturierte Eigenschaft: Scriptparameter: custom_agg Beispiel: set :stpnode.custom_agg = [[field1 function] [field2 function]] Dabei ist function die Aggregationsfunktion, die mit diesem Feld verwendet wird.
Registerkarte **Basis**		
include_intercept	*Flag*	
max_autoregressive_lag	*Integer*	Minimum 1, Maximum 5 in Schritten von 1. Dies ist die Anzahl der vorherigen Datensätze, die für eine Vorhersage erforderlich sind. Wird z. B. 5 festgelegt, wird aus den vorherigen 5 Datensätzen eine neue Vorhersage erstellt. Die Anzahl Datensätze, die hier aus den Erstellungsdaten angegeben werden, werden in das Modell aufgenommen und daher braucht der Benutzer die Daten nicht erneut anzugeben, wenn für das Modell Scoring durchgeführt wird.

Tabelle 147. stpnode, Eigenschaften (Forts.)		
Eigenschaften von stpnode	**Datentyp**	**Eigenschaftsbeschreibung**
estimation_method	Parametric Nonparametric	Die Methode für das Modellieren der räumlichen Kovarianzmatrix.
parametric_model	Gaussian Exponential PoweredExponential	Reihenfolgeparameter für das räumliche Kovarianzmodell Parametric.
exponential_power	*number*	Potenzstufe für das Modell PoweredExponential. Minimum: 1, Maximum: 2.
Registerkarte **Erweitert**		
max_missing_values	*Integer*	Der maximale Prozentsatz von Datensätzen mit fehlenden Werten, die im Modell zulässig sind.
significance	*number*	Das Signifikanzniveau für Hypothesetests in der Modellerstellung. Gibt den Signifikanzwert für alle Tests in der STP-Modellschätzung an, einschließlich zwei Anpassungsgütetests, F-Tests für Effekte und t-Tests für Koeffizienten.
Registerkarte **Ausgabe**		
model_specifications	*Flag*	
temporal_summary	*Flag*	
location_summary	*Flag*	Legt fest, ob die Positionszusammenfassungstabelle in die Modellausgabe aufgenommen wird.
model_quality	*Flag*	
test_mean_structure	*Flag*	
mean_structure_coefficients	*Flag*	
autoregressive_coefficients	*Flag*	
test_decay_space	*Flag*	
parametric_spatial_covariance	*Flag*	
correlations_heat_map	*Flag*	
correlations_map	*Flag*	
location_clusters	*Flag*	

Tabelle 147. stpnode, Eigenschaften (Forts.)		
Eigenschaften von stpnode	**Datentyp**	**Eigenschaftsbeschreibung**
similarity_threshold	*number*	Der Schwellenwert, an dem Ausgabecluster als ähnlich genug betrachtet werden, um in einen einzelnen Cluster zusammengeführt werden zu können.
max_number_clusters	*Integer*	Die Obergrenze für die Anzahl Cluster, die in die Modellausgabe eingeschlossen werden können.
Registerkarte **Modelloptionen**		
use_model_name	*Flag*	
model_name	*string*	
uncertainty_factor	*number*	Minimum: 0, Maximum: 100. Legt die Unsicherheitszunahme (Fehlerzunahme) fest, die zukünftig auf Vorhersagen angewendet wird. Es handelt sich um die Ober- und Untergrenze für die Vorhersagen.

svmnode, Eigenschaften

Der Knoten "Support Vector Machine" (SVM) ermöglicht die Klassifizierung von Daten in eine von zwei Gruppen ohne Überanpassung. SVM eignet sich gut für umfangreiche Datasets, beispielsweise solche mit einer großen Anzahl an Eingabefeldern.

Beispiel

```
node = stream.create("svm", "My node")
# Expert tab
node.setPropertyValue("mode", "Expert")
node.setPropertyValue("all_probabilities", True)
node.setPropertyValue("kernel", "Polynomial")
node.setPropertyValue("gamma", 1.5)
```

Tabelle 148. svmnode, Eigenschaften		
Eigenschaften von svmnode	**Werte**	**Eigenschaftsbeschreibung**
all_probabilities	*Flag*	

Tabelle 148. svmnode, Eigenschaften (Forts.)

Eigenschaften von svmnode	Werte	Eigenschaftsbeschreibung
stopping_criteria	1.0E-1 1.0E-2 1.0E-3 (Standard) 1.0E-4 1.0E-5 1.0E-6	Bestimmt, wann der Optimierungsalgorithmus gestoppt werden soll.
regularization	*number*	Auch als C-Parameter bekannt.
precision	*number*	Nur verwendet, wenn es sich beim Messniveau des Zielfelds um Continuous (Stetig) handelt.
kernel	RBF(Standard) Polynomial Sigmoid Linear	Typ der für die Transformation verwendeten Kernfunktion.
rbf_gamma	*number*	Nur verwendet, wenn für kernel der Typ RBF gilt.
gamma	*number*	Nur verwendet, wenn für kernel der TypPolynomial oder Sigmoid gilt.
bias	*number*	
degree	*number*	Nur verwendet, wenn für kernel der Typ Polynomial gilt.
calculate_variable_importance	*Flag*	
calculate_raw_propensities	*Flag*	
calculate_adjusted_propensities	*Flag*	
adjusted_propensity_partition	Test Validation	

Eigenschaften von "tcmnode"

Mithilfe der temporalen kausalen Modellierung lassen sich kausale Schlüsselbeziehungen in Zeitreihendaten erkennen. Bei der temporalen kausalen Modellierung geben Sie ein Set von Zielzeitreihen und ein Set von möglichen Eingaben für diese Ziele an. Die Prozedur erstellt dann ein autoregressives Zeitreihenmodell für jedes Ziel und schließt nur jene Eingaben ein, die die signifikantesten kausalen Beziehungen mit dem Ziel haben.

Tabelle 149. Eigenschaften von "tcmnode"

Eigenschaften von tcmnode	**Werte**	**Eigenschaftsbeschreibung**
custom_fields	*boolesch*	
dimensionlist	[*dimension1 ... dimensionN*]	
data_struct	Multiple Single	
metric_fields	*Felder*	
both_target_and_input	[*f1 ... FN*]	
targets	[*f1 ... FN*]	
candidate_inputs	[*f1 ... FN*]	
forced_inputs	[*f1 ... FN*]	
use_timestamp	Timestamp Period	

Tabelle 149. Eigenschaften von "tcmnode" (Forts.)		
Eigenschaften von tcmnode	**Werte**	**Eigenschaftsbeschreibung**
input_interval	None Unknown Year Quarter Month Week Day Hour Hour_nonperiod Minute Minute_nonperiod Second Second_nonperiod	
period_field	*string*	
period_start_value	*Integer*	
num_days_per_week	*Integer*	
start_day_of_week	Sunday Monday Tuesday Wednesday Thursday Friday Saturday	
num_hours_per_day	*Integer*	
start_hour_of_day	*Integer*	
timestamp_increments	*Integer*	

Tabelle 149. Eigenschaften von "tcmnode" (Forts.)		
Eigenschaften von tcmnode	**Werte**	**Eigenschaftsbeschreibung**
cyclic_increments	*Integer*	
cyclic_periods	*Liste*	
output_interval	None Year Quarter Month Week Day Hour Minute Second	
is_same_interval	Same Notsame	
cross_hour	*boolesch*	
aggregate_and_distribute	*Liste*	
aggregate_default	Mean Sum Mode Min Max	
distribute_default	Mean Sum	

Tabelle 149. Eigenschaften von "tcmnode" (Forts.)		
Eigenschaften von tcmnode	**Werte**	**Eigenschaftsbeschreibung**
group_default	Mean Sum Mode Min Max	
missing_imput	Linear_interp Series_mean K_mean K_meridian Linear_trend None	
k_mean_param	*Integer*	
k_median_param	*Integer*	
missing_value_threshold	*Integer*	
conf_level	*Integer*	
max_num_predictor	*Integer*	
max_lag	*Integer*	
epsilon	*number*	
threshold	*Integer*	
is_re_est	*boolesch*	
num_targets	*Integer*	
percent_targets	*Integer*	
fields_display	*Liste*	
series_display	*Liste*	
network_graph_for_target	*boolesch*	
sign_level_for_target	*number*	
fit_and_outlier_for_target	*boolesch*	
sum_and_para_for_target	*boolesch*	
impact_diag_for_target	*boolesch*	

Tabelle 149. Eigenschaften von "tcmnode" (Forts.)		
Eigenschaften von tcmnode	**Werte**	**Eigenschaftsbeschreibung**
impact_diag_type_for_target	Effect Cause Both	
impact_diag_level_for_target	*Integer*	
series_plot_for_target	*boolesch*	
res_plot_for_target	*boolesch*	
top_input_for_target	*boolesch*	
forecast_table_for_target	*boolesch*	
same_as_for_target	*boolesch*	
network_graph_for_series	*boolesch*	
sign_level_for_series	*number*	
fit_and_outlier_for_series	*boolesch*	
sum_and_para_for_series	*boolesch*	
impact_diagram_for_series	*boolesch*	
impact_diagram_type_for_series	Effect Cause Both	
impact_diagram_level_for_series	*Integer*	
series_plot_for_series	*boolesch*	
residual_plot_for_series	*boolesch*	
forecast_table_for_series	*boolesch*	
outlier_root_cause_analysis	*boolesch*	
causal_levels	*Integer*	
outlier_table	Interactive Pivot Both	
rmsp_error	*boolesch*	

Tabelle 149. Eigenschaften von "tcmnode" (Forts.)		
Eigenschaften von tcmnode	**Werte**	**Eigenschaftsbeschreibung**
bic	*boolesch*	
r_square	*boolesch*	
outliers_over_time	*boolesch*	
series_transormation	*boolesch*	
use_estimation_period	*boolesch*	
estimation_period	Times Observation	
observations	*Liste*	
observations_type	Latest Earliest	
observations_num	*Integer*	
observations_exclude	*Integer*	
extend_records_into_future	*boolesch*	
forecastperiods	*Integer*	
max_num_distinct_values	*Integer*	
display_targets	FIXEDNUMBER PERCENTAGE	
goodness_fit_measure	ROOTMEAN BIC RSQUARE	
top_input_for_series	*boolesch*	
aic	*boolesch*	
rmse	*boolesch*	

ts, Eigenschaften

Der Zeitreihenknoten berechnet Schätzungen für die exponentielle Glättung sowie univariate und multivariate ARIMA-Modelle (ARIMA steht für Autoregressive Integrated Moving Average (autoregressiver, integrierter gleitender Durchschnitt)) für Zeitreihendaten und erstellt Vorhersagen über die zukünftige Leistung. Dieser Zeitreihenknoten ähnelt dem bisherigen Zeitreihenknoten, der in SPSS Modeler Version 18 nicht mehr unterstützt wird. Allerdings ist dieser neuere Zeitreihenknoten für die Nutzung der Leistungsstärke von IBM SPSS Analytic Server und für die Verarbeitung großer Datenmengen konzipiert. Er zeigt das resultierende Modell im Ausgabeviewer an, der in SPSS Modeler Version 17 hinzugefügt wurde.

Tabelle 150. ts, Eigenschaften		
Eigenschaften von ts	**Werte**	**Eigenschaftsbeschreibung**
targets	*Feld*	Der Zeitreihenknoten sagt mindestens ein Ziel voraus; optional können dabei ein oder mehrere Eingabefelder als Prädiktoren verwendet werden. Häufigkeits- und Gewichtungsfelder werden nicht verwendet. Weitere Informationen finden Sie im Thema „Allgemeine Eigenschaften von Modellierungsknoten" auf Seite 225.
candidate_inputs	[*field1 ... FeldN]*	Im Modell verwendete Eingabe- bzw. Prädiktorfelder.
use_period	*Flag*	
date_time_field	*Feld*	
input_interval	None Unknown Year Quarter Month Week Day Hour Hour_nonperiod Minute Minute_nonperiod Second Second_nonperiod	
period_field	*Feld*	

Tabelle 150. ts, Eigenschaften (Forts.)		
Eigenschaften von ts	**Werte**	**Eigenschaftsbeschreibung**
period_start_value	*Integer*	
num_days_per_week	*Integer*	
start_day_of_week	Sunday Monday Tuesday Wednesday Thursday Friday Saturday	
num_hours_per_day	*Integer*	
start_hour_of_day	*Integer*	
timestamp_increments	*Integer*	
cyclic_increments	*Integer*	
cyclic_periods	*Liste*	
output_interval	None Year Quarter Month Week Day Hour Minute Second	
is_same_interval	*Flag*	
cross_hour	*Flag*	
aggregate_and_distribute	*Liste*	

Tabelle 150. ts, Eigenschaften (Forts.)

Eigenschaften von ts	**Werte**	**Eigenschaftsbeschreibung**
aggregate_default	Mean Sum Mode Min Max	
distribute_default	Mean Sum	
group_default	Mean Sum Mode Min Max	
missing_imput	Linear_interp Series_mean K_mean K_median Linear_trend	
k_span_points	*Integer*	
use_estimation_period	*Flag*	
estimation_period	Observations Times	
date_estimation	*Liste*	Nur verfügbar, wenn Sie date_time_field verwenden
period_estimation	*Liste*	Nur verfügbar, wenn Sie use_period verwenden
observations_type	Latest Earliest	

Tabelle 150. ts, Eigenschaften (Forts.)		
Eigenschaften von ts	**Werte**	**Eigenschaftsbeschreibung**
observations_num	*Integer*	
observations_exclude	*Integer*	
method	ExpertModeler Exsmooth Arima	
expert_modeler_method	ExpertModeler Exsmooth Arima	
consider_seasonal	*Flag*	
detect_outliers	*Flag*	
expert_outlier_additive	*Flag*	
expert_outlier_level_shift	*Flag*	
expert_outlier_innovational	*Flag*	
expert_outlier_level_shift	*Flag*	
expert_outlier_transient	*Flag*	
expert_outlier_seasonal_additive	*Flag*	
expert_outlier_local_trend	*Flag*	
expert_outlier_additive_patch	*Flag*	
consider_newesmodels	*Flag*	

Tabelle 150. ts, Eigenschaften (Forts.)

Eigenschaften von ts	**Werte**	**Eigenschaftsbeschreibung**
exsmooth_model_type	Simple HoltsLinearTrend BrownsLinearTrend DampedTrend SimpleSeasonal WintersAdditive WintersMultiplicative DampedTrendAdditive DampedTrendMultiplicative MultiplicativeTrendAdditive MultiplicativeSeasonal MultiplicativeTrendMultiplicative MultiplicativeTrend	Gibt die Methode für exponentielle Glättung an. Der Standardwert ist Simple.

Tabelle 150. ts, Eigenschaften (Forts.)

Eigenschaften von ts	Werte	Eigenschaftsbeschreibung
futureValue_type_method	Compute specify	Wenn Compute verwendet wird, berechnet das System die zukünftigen Werte für die Vorhersageperiode der einzelnen Prädiktoren. Für jeden Prädiktor können Sie aus einer Liste von Funktionen auswählen (Leer, Mittelwert der zuletzt verwendeten Punkte, Zuletzt verwendeter Wert) oder Sie können specify verwenden, um Werte manuell einzugeben. Wenn Sie einzelne Felder und Eigenschaften angeben wollen, verwenden Sie die Eigenschaft extend_metric_values. Beispiel: `set :ts.futureVa□ lue_type_method="spe□ cify" set :ts.extend_met□ ric_values=[{'Mar□ ket_1','USER_SPECI□ FY', [1,2,3]}, {'Market_2','MOST_RE□ CENT_VALUE', ''},{'Market_3','RE□ CENT_POINTS_MEAN', ''}]`
exsmooth_transformation_type	None SquareRoot NaturalLog	
arima.p	*Integer*	
arima.d	*Integer*	
arima.q	*Integer*	
arima.sp	*Integer*	
arima.sd	*Integer*	
arima.sq	*Integer*	

Tabelle 150. ts, Eigenschaften (Forts.)

Eigenschaften von ts	**Werte**	**Eigenschaftsbeschreibung**
arima_transformation_type	None SquareRoot NaturalLog	
arima_include_constant	*Flag*	
tf_arima.p. *feldname*	*Integer*	Für Transferfunktionen.
tf_arima.d. *feldname*	*Integer*	Für Transferfunktionen.
tf_arima.q. *feldname*	*Integer*	Für Transferfunktionen.
tf_arima.sp. *feldname*	*Integer*	Für Transferfunktionen.
tf_arima.sd. *feldname*	*Integer*	Für Transferfunktionen.
tf_arima.sq. *feldname*	*Integer*	Für Transferfunktionen.
tf_arima.delay. *feldname*	*Integer*	Für Transferfunktionen.
tf_arima.transformation_type. *feldname*	None SquareRoot NaturalLog	Für Transferfunktionen.
arima_detect_outliers	*Flag*	
arima_outlier_additive	*Flag*	
arima_outlier_level_shift	*Flag*	
arima_outlier_innovational	*Flag*	
arima_outlier_transient	*Flag*	
arima_outlier_seasonal_additive	*Flag*	
arima_outlier_local_trend	*Flag*	
arima_outlier_additive_patch	*Flag*	
max_lags	*Integer*	
cal_PI	*Flag*	
conf_limit_pct	*Reelle Zahl*	
events	*Felder*	
continue	*Flag*	
scoring_model_only	*Flag*	Für Modelle mit sehr großen Zahlen (Zehntausende) von Zeitreihen.
forecastperiods	*Integer*	
extend_records_into_future	*Flag*	

Tabelle 150. ts, Eigenschaften (Forts.)		
Eigenschaften von ts	**Werte**	**Eigenschaftsbeschreibung**
extend_metric_values	*Felder*	Hiermit können Sie zukünftige Werte für Prädiktoren angeben.
conf_limits	*Flag*	
noise_res	*Flag*	
max_models_output	*Integer*	Steuert, wie viele Modelle in der Ausgabe angezeigt werden. Der Standardwert ist 10. Modelle werden nicht in der Ausgabe angezeigt, wenn die Gesamtzahl der erstellten Modelle diesen Wert überschreitet. Modelle sind weiterhin für das Scoring verfügbar.

Eigenschaften von "timeseriesnode" (nicht mehr unterstützt)

Anmerkung: Dieser ursprüngliche Zeitreihenknoten wird in SPSS Modeler Version 18 nicht mehr unterstützt und durch den neuen Zeitreihenknoten ersetzt, der für die Nutzung der Leistungsstärke von IBM SPSS Analytic Server und für die Verarbeitung großer Datenmengen konzipiert ist.

Der Zeitreihenknoten berechnet Schätzungen für die exponentielle Glättung sowie univariate und multivariate ARIMA-Modelle (ARIMA steht für Autoregressive Integrated Moving Average (autoregressiver, integrierter gleitender Durchschnitt)) für Zeitreihendaten und erstellt Vorhersagen über die zukünftige Leistung. Einem Zeitreihenknoten muss stets ein Zeitintervallknoten vorangehen.

Beispiel

```
node = stream.create("timeseries", "My node")
node.setPropertyValue("method", "Exsmooth")
node.setPropertyValue("exsmooth_model_type", "HoltsLinearTrend")
node.setPropertyValue("exsmooth_transformation_type", "None")
```

Tabelle 151. Eigenschaften von "timeseriesnode"		
Eigenschaften von `timeseriesnode`	**Werte**	**Eigenschaftsbeschreibung**
`targets`	*Feld*	Der Zeitreihenknoten sagt mindestens ein Ziel voraus; optional können dabei ein oder mehrere Eingabefelder als Prädiktoren verwendet werden. Häufigkeits- und Gewichtungsfelder werden nicht verwendet. Weitere Informationen finden Sie im Thema „Allgemeine Eigenschaften von Modellierungsknoten" auf Seite 225.
`continue`	*Flag*	

`method`	`ExpertModeler`
	
	`Exsmooth`
	
	`Arima`
	
	`Reuse`	

`expert_modeler_method`	*Flag*	
`consider_seasonal`	*Flag*	
`detect_outliers`	*Flag*	
`expert_outlier_additive`	*Flag*	
`expert_outlier_level_shift`	*Flag*	
`expert_outlier_innovational`	*Flag*	
`expert_outlier_level_shift`	*Flag*	
`expert_outlier_transient`	*Flag*	
`expert_outlier_seasonal_additive`	*Flag*	
`expert_outlier_local_trend`	*Flag*	
`expert_outlier_additive_patch`	*Flag*	

Tabelle 151. Eigenschaften von "timeseriesnode" (Forts.)		
Eigenschaften von timeseriesnode	**Werte**	**Eigenschaftsbeschreibung**
exsmooth_model_type	Simple HoltsLinearTrend BrownsLinearTrend DampedTrend SimpleSeasonal WintersAdditive WintersMultiplicative	
exsmooth_transformation_type	None SquareRoot NaturalLog	
arima_p	*Integer*	
arima_d	*Integer*	
arima_q	*Integer*	
arima_sp	*Integer*	
arima_sd	*Integer*	
arima_sq	*Integer*	
arima_transformation_type	None SquareRoot NaturalLog	
arima_include_constant	*Flag*	
tf_arima_p. *Feldname*	*Integer*	Für Transferfunktionen.
tf_arima_d. *Feldname*	*Integer*	Für Transferfunktionen.
tf_arima_q. *Feldname*	*Integer*	Für Transferfunktionen.
tf_arima_sp. *Feldname*	*Integer*	Für Transferfunktionen.
tf_arima_sd. *Feldname*	*Integer*	Für Transferfunktionen.
tf_arima_sq. *Feldname*	*Integer*	Für Transferfunktionen.
tf_arima_delay. *Feldname*	*Integer*	Für Transferfunktionen.

Tabelle 151. Eigenschaften von "timeseriesnode" (Forts.)		
Eigenschaften von `timeseriesnode`	**Werte**	**Eigenschaftsbeschreibung**

`tf_arima_transformation_type.` *Feldname*	`None`
	
	`SquareRoot`
	
	`NaturalLog`
		Für Transferfunktionen.

`arima_detect_outlier_mode`	`None`
	
	`Automatic`	

`arima_outlier_additive`	*Flag*	
`arima_outlier_level_shift`	*Flag*	
`arima_outlier_innovational`	*Flag*	
`arima_outlier_transient`	*Flag*	
`arima_outlier_seasonal_additive`	*Flag*	
`arima_outlier_local_trend`	*Flag*	
`arima_outlier_additive_patch`	*Flag*	
`conf_limit_pct`	*Reelle Zahl*	
`max_lags`	*Integer*	
`events`	*Felder*	
`scoring_model_only`	*Flag*	Für Modelle mit sehr großen Zahlen (Zehntausende) von Zeitreihen.

treeas, Eigenschaften

Der Tree-AS-Knoten ähnelt dem vorhandenen CHAID-Knoten, allerdings ist der Tree-AS-Knoten für die Verarbeitung großer Datenmengen konzipiert. Er erstellt daraus einen einzelnen Baum und zeigt das resultierende Modell im Ausgabeviewer an, der in SPSS Modeler Version 17 hinzugefügt wurde. Der Knoten generiert einen Entscheidungsbaum unter Verwendung von Chi-Quadrat-Statistiken (CHAID) zum Identifizieren optimaler Aufteilungen. Durch diese Verwendung von CHAID können nicht binäre Bäume generiert werden, d. h., einige Aufteilungen können mehr als zwei Verzweigungen enthalten. Ziel- und Eingabefelder können in einem numerischen Bereich (stetig) oder kategorial sein. Exhaustive CHAID ist eine Änderung von CHAID, die noch gründlicher vorgeht, indem sie alle möglichen Aufteilungen untersucht, allerdings mehr Rechenzeit beansprucht.

Tabelle 152. treeas, Eigenschaften

Eigenschaften von `treeas`	**Werte**	**Eigenschaftsbeschreibung**
`target`	*Feld*	Im Tree-AS-Knoten erfordern CHAID-Modelle ein einzelnes Ziel und eines oder mehrere Eingabefelder. Außerdem kann ein Häufigkeitsfeld angegeben werden. Weitere Informationen finden Sie im Thema „Allgemeine Eigenschaften von Modellierungsknoten" auf Seite 225.

`method`	`chaid`
	
	`exhaustive_chaid`	

`max_depth`	*Integer*	Maximale Baumtiefe, von 0 bis 20. Der Standardwert ist 5.
`num_bins`	*Integer*	Wird nur verwendet, wenn die Daten aus stetigen Eingaben bestehen. Legen Sie die Anzahl der Klassen mit gleicher Häufigkeit fest, die für die Eingaben verwendet werden sollen. Optionen sind 2, 4, 5, 10, 20, 25, 50 oder 100.
`record_threshold`	*Integer*	Die Anzahl der Datensätze, bei der das Modell beim Erstellen des Baums von der Verwendung von p-Werten zu Effektgrößen wechselt. Der Standardwert ist 1.000.000; erhöhen oder verringern Sie diesen Wert in Inkrementen von 10.000.
`split_alpha`	*number*	Signifikanzschwelle für Aufteilung. Der Wert muss zwischen 0,01 und 0,99 liegen.
`merge_alpha`	*number*	Signifikanzschwelle für Zusammenführung. Der Wert muss zwischen 0,01 und 0,99 liegen.
`bonferroni_adjustment`	*Flag*	Signifikanzwerte mit der Bonferroni-Methode anpassen.
`effect_size_threshold_cont`	*number*	Schwellenwert für die Effektgröße festlegen, wenn bei Verwendung eines stetigen Ziels Knoten aufgeteilt und Kategorien zusammengeführt werden. Der Wert muss zwischen 0,01 und 0,99 liegen.
`effect_size_threshold_cat`	*number*	Schwellenwert für die Effektgröße festlegen, wenn bei Verwendung eines kategorialen Ziels Knoten aufgeteilt und Kategorien zusammengeführt werden. Der Wert muss zwischen 0,01 und 0,99 liegen.

Tabelle 152. treeas, Eigenschaften (Forts.)

Eigenschaften von `treeas`	Werte	Eigenschaftsbeschreibung
`split_merged_categories`	*Flag*	Erneutes Aufteilen zusammengeführter Kategorien zulassen.
`grouping_sig_level`	*number*	Wird verwendet, um zu bestimmen, wie Knotengruppen gebildet werden oder wie ungewöhnliche Knoten identifiziert werden.

`chi_square`	`pearson`
	
	`likelihood_ratio`
		Verwendetes Verfahren für die Berechnung der Chi-Quadrat-Statistik: Pearson oder Likelihood-Quotient

`minimum_record_use`	`use_percentage`
	
	`use_absolute`	

`min_parent_records_pc`	*number*	Der Standardwert ist "2". Minimum: 1, Maximum: 100, in Inkrementen von 1. Der Wert der übergeordneten Verzweigung muss höher als der Wert der untergeordneten Verzweigung sein.
`min_child_records_pc`	*number*	Der Standardwert ist 1. Minimum: 1, Maximum: 100, in Inkrementen von 1.
`min_parent_records_abs`	*number*	Der Standardwert ist 100. Minimum: 1, Maximum: 100, in Inkrementen von 1. Der Wert der übergeordneten Verzweigung muss höher als der Wert der untergeordneten Verzweigung sein.
`min_child_records_abs`	*number*	Der Standardwert ist 50. Minimum: 1, Maximum: 100, in Inkrementen von 1.
`epsilon`	*number*	Minimale Änderung in der erwarteten Zellhäufigkeit...
`max_iterations`	*number*	Maximale Anzahl der Iterationen für Konvergenz.
`use_costs`	*Flag*	
`costs`	*strukturiert*	Strukturierte Eigenschaft. Das Format ist eine Liste mit 3 Werten: der tatsächliche Wert, der vorhergesagte Wert und die Kosten, falls die Vorhersage falsch ist. Beispiel: tree.setPropertyValue("costs", [["drugA", "drugB", 3.0], ["drugX", "drugY", 4.0]])

Tabelle 152. treeas, Eigenschaften (Forts.)		
Eigenschaften von treeas	**Werte**	**Eigenschaftsbeschreibung**
default_cost_increase	none linear square custom	**Anmerkung:** Nur für ordinale Ziele aktiviert. Standardwerte in der Kostenmatrix festlegen.
calculate_conf	*Flag*	
display_rule_id	*Flag*	Fügt ein Feld zur Scoring-Ausgabe hinzu, das die ID des Endknotens angibt, dem der jeweilige Datensatz zugewiesen ist.

Eigenschaften von "twostepnode"

Der TwoStep-Knoten verwendet eine aus zwei Schritten bestehende Clustering-Methode. Im ersten Schritt wird ein einzelner Durchlauf durch die Daten vorgenommen, bei dem die Eingangsrohdaten zu einem verwaltbaren Set von Subclustern komprimiert werden. Im zweiten Schritt werden die Subcluster mithilfe einer hierarchischen Clustering-Methode nach und nach in immer größere Cluster zusammengeführt. TwoStep hat den Vorteil, dass die optimale Anzahl von Clustern für die Trainingsdaten automatisch geschätzt wird. Mit dem Verfahren können gemischte Feldtypen und große Datasets effizient verarbeitet werden.

Beispiel

```
node = stream.create("twostep", "My node")
node.setPropertyValue("custom_fields", True)
node.setPropertyValue("inputs", ["Age", "K", "Na", "BP"])
node.setPropertyValue("partition", "Test")
node.setPropertyValue("use_model_name", False)
node.setPropertyValue("model_name", "TwoStep_Drug")
node.setPropertyValue("use_partitioned_data", True)
node.setPropertyValue("exclude_outliers", True)
node.setPropertyValue("cluster_label", "String")
node.setPropertyValue("label_prefix", "TwoStep_")
node.setPropertyValue("cluster_num_auto", False)
node.setPropertyValue("max_num_clusters", 9)
node.setPropertyValue("min_num_clusters", 3)
node.setPropertyValue("num_clusters", 7)
```

Tabelle 153. Eigenschaften von "twostepnode"		
Eigenschaften von twostep-node	**Werte**	**Eigenschaftsbeschreibung**
inputs	[*field1 ... FeldN]*	TwoStep-Modelle verwenden eine Liste mit Eingabefeldern, jedoch kein Ziel. Gewichtungs- und Häufigkeitsfelder werden nicht erkannt. Weitere Informationen finden Sie im Thema „Allgemeine Eigenschaften von Modellierungsknoten" auf Seite 225.

Tabelle 153. Eigenschaften von "twostepnode" (Forts.)		
Eigenschaften von twostep-node	**Werte**	**Eigenschaftsbeschreibung**
standardize	*Flag*	
exclude_outliers	*Flag*	
percentage	*number*	
cluster_num_auto	*Flag*	
min_num_clusters	*number*	
max_num_clusters	*number*	
num_clusters	*number*	
cluster_label	String Number	
label_prefix	*string*	
distance_measure	Euclidean Loglikelihood	
clustering_criterion	AIC BIC	

Eigenschaften von "twostepAS"

TwoStep-Cluster ist ein exploratives Tool, das zur Ermittlung natürlicher Gruppierungen (Cluster) innerhalb eines Datasets dient, die andernfalls nicht erkennbar wären. Der von dieser Prozedur verwendete Algorithmus hat mehrere wünschenswerte Funktionen, die ihn von konventionellen Clustering-Verfahren wie Handhabung von kategorialen und stetigen Variablen, automatische Auswahl von Clustern und Skalierbarkeit unterscheiden.

Tabelle 154. Eigenschaften von "twostepAS"		
Eigenschaften von twostepAS	**Werte**	**Eigenschaftsbeschreibung**
inputs	[*f1 ... FN*]	TwoStepAS-Modelle verwenden eine Liste mit Eingabefeldern, jedoch kein Ziel. Gewichtungs- und Häufigkeitsfelder werden nicht erkannt.
use_predefined_roles	Boolesch	Standard=True
use_custom_field_assignments	Boolesch	Standard=False
cluster_num_auto	Boolesch	Standard=True
min_num_clusters	Integer	Default=2
max_num_clusters	Integer	Default=15

Tabelle 154. Eigenschaften von "twostepAS" (Forts.)

Eigenschaften von twostepAS	**Werte**	**Eigenschaftsbeschreibung**
num_clusters	Integer	Default=5
clustering_criterion	AIC BIC	
automatic_clustering_method	use_clustering_criterion_setting Distance_jump Minimum Maximum	
feature_importance_method	use_clustering_criterion_setting effect_size	
use_random_seed	Boolesch	
random_seed	Integer	
distance_measure	Euclidean Loglikelihood	
include_outlier_clusters	Boolesch	Standard=True
num_cases_in_feature_tree_leaf_is_less_than	Integer	Default=10
top_perc_outliers	Integer	Default=5
initial_dist_change_threshold	Integer	Default=0
leaf_node_maximum_branches	Integer	Default=8
non_leaf_node_maximum_branches	Integer	Default=8
max_tree_depth	Integer	Default=3
adjustment_weight_on_measurement_level	Integer	Default=6
memory_allocation_mb	Zahl	Default=512
delayed_split	Boolesch	Standard=True
fields_to_standardize	[*f1 ... FN*]	
adaptive_feature_selection	Boolesch	Standard=True
featureMisPercent	Integer	Default=70
coefRange	Zahl	Default=0.05
percCasesSingleCategory	Integer	Default=95

Tabelle 154. Eigenschaften von "twostepAS" (Forts.)		
Eigenschaften von twostepAS	**Werte**	**Eigenschaftsbeschreibung**
numCases	Integer	Default=24
include_model_specifications	Boolesch	Standard=True
include_record_summary	Boolesch	Standard=True
include_field_transformations	Boolesch	Standard=True
excluded_inputs	Boolesch	Standard=True
evaluate_model_quality	Boolesch	Standard=True
show_feature_importance bar chart	Boolesch	Standard=True
show_feature_importance_ word_cloud	Boolesch	Standard=True
show_outlier_clusters interacti-ve_table_and_chart	Boolesch	Standard=True
show_outlier_clusters_pi-vot_table	Boolesch	Standard=True
across_cluster_feature_impor-tance	Boolesch	Standard=True
across_cluster_profiles_pi-vot_table	Boolesch	Standard=True
withinprofiles	Boolesch	Standard=True
cluster_distances	Boolesch	Standard=True
cluster_label	String Number	
label_prefix	String	

Kapitel 14. Eigenschaften von Modellnuggetknoten

Modellnuggetknoten besitzen dieselben allgemeinen Eigenschaften wie andere Knoten. Weitere Informationen finden Sie im Thema „Allgemeine Knoteneigenschaften” auf Seite 80.

Eigenschaften von "applyanomalydetectionnode"

Mithilfe von Modellierungsknoten vom Typ "Anomalieerkennung" kann ein Modellnugget vom Typ "Anomalieerkennung" generiert werden. Der Scriptname dieses Modellnuggets lautet *applyanomalydetectionnode*. Weitere Informationen zum Scripting für den Modellierungsknoten selbst finden Sie unter „anomalydetectionnode, Eigenschaften” auf Seite 226.

Tabelle 155. Eigenschaften von "applyanomalydetectionnode"

Eigenschaften von applyanomalydetectionnode	**Werte**	**Eigenschaftsbeschreibung**

`anomaly_score_method`	`FlagAndScore`
	`FlagOnly`
	`ScoreOnly`
		Bestimmt, welche Ausgaben für das Scoring erstellt werden.

`num_fields`	*Integer*	Zu meldende Felder.
`discard_records`	*Flag*	Gibt an, ob Datensätze aus der Ausgabe verworfen werden sollen oder nicht.
`discard_anomalous_records`	*Flag*	Gibt an, ob die anomalen oder die *nicht* anomalen Datensätze verworfen werden sollen. Der Standardwert ist `off`, was bedeutet, dass die *nicht* anomalen Datensätze verworfen werden. Bei on werden die anomalen Datensätze verworfen. Diese Eigenschaft ist nur aktiviert, wenn die Eigenschaft `discard_records` aktiviert ist.

Eigenschaften von "applyapriorinode"

Mithilfe von Apriori-Modellierungsknoten kann ein Modellnugget vom Typ "Apriori" generiert werden. Der Scriptname dieses Modellnuggets lautet *applyapriorinode*. Weitere Informationen zum Scripting für den Modellierungsknoten selbst finden Sie in „apriorinode, Eigenschaften” auf Seite 227

Tabelle 156. Eigenschaften von "applyapriorinode"

Eigenschaften von applyapriorinode	**Werte**	**Eigenschaftsbeschreibung**
`max_predictions`	*Anzahl (Ganzzahl)*	
`ignore_unmatached`	*Flag*	
`allow_repeats`	*Flag*	

Tabelle 156. Eigenschaften von "applyapriorinode" (Forts.)		
Eigenschaften von applyapriorinode	**Werte**	**Eigenschaftsbeschreibung**
check_basket	NoPredictions Predictions NoCheck	
criterion	Confidence Support RuleSupport Lift Deployability	

Eigenschaften von "applyassociationrulesnode"

Der Assoziationsregelmodellierungsknoten kann zum Generieren eines Assoziationsregelmodellnuggets verwendet werden. Der Scriptname dieses Modellnuggets lautet *applyassociationrulesnode*. Weitere Informationen zum Scripting für den Modellierungsknoten selbst finden Sie in „associationrulesnode, Eigenschaften” auf Seite 229.

Tabelle 157. applyassociationrulesnode, Eigenschaften		
Eigenschaften von applyassociationrulesnode	**Datentyp**	**Eigenschaftsbeschreibung**
max_predictions	*Integer*	Die maximale Anzahl Regeln, die auf jede Eingabe mit dem Score angewendet werden können.
criterion	Confidence Rulesupport Lift Conditionsupport Deployability	Wählen Sie das Maß aus, das zum Festlegen der Stärke von Regeln verwendet wird.
allow_repeats	*boolesch*	Legt fest, ob Regeln mit derselben Vorhersage in den Score eingeschlossen werden.

Tabelle 157. applyassociationrulesnode, Eigenschaften (Forts.)		
Eigenschaften von applyassociationrulesnode	**Datentyp**	**Eigenschaftsbeschreibung**
check_input	NoPredictions Predictions NoCheck	

Eigenschaften von "applyautoclassifiernode"

Mithilfe von Modellierungsknoten des Typs "Automatisches Klassifikationsmerkmal" kann ein Modellnugget vom Typ "Automatisches Klassifikationsmerkmal" generiert werden. Der Scriptname dieses Modellnuggets lautet *applyautoclassifiernode*. Weitere Informationen zu Scripts für den Modellierungsknoten selbst finden Sie in „autoclassifiernode, Eigenschaften” auf Seite 232.

Tabelle 158. Eigenschaften von "applyautoclassifiernode"		
Eigenschaften von applyautoclassifiernode	**Werte**	**Eigenschaftsbeschreibung**
flag_ensemble_method	Voting EvaluationWeightedVoting ConfidenceWeightedVoting RawPropensityWeightedVoting HighestConfidence AverageRawPropensity	Gibt an, welche Methode für die Bestimmung des Ensemble-Score verwendet werden soll. Diese Einstellung gilt nur, wenn das ausgewählte Ziel ein Flagfeld ist.
flag_evaluation_selection	Accuracy AUC_ROC	Diese Option gilt nur für Flagziele, um zu entscheiden, welches Evaluierungsmaß für das nach Bewertung gewichtete Voting ausgewählt wird.
filter_individual_model_output	*Flag*	Gibt an, ob Scoring-Ergebnisse aus einzelnen Modellen unterdrückt werden sollen.

Tabelle 158. Eigenschaften von "applyautoclassifiernode" (Forts.)		
Eigenschaften von applyautoclassifiernode	**Werte**	**Eigenschaftsbeschreibung**
is_ensemble_update	*Flag*	Aktiviert den Modus "Kontinuierliches automatisches maschinelles Lernen", bei dem neue Komponentenmodelle zu einer Gruppe vorhandener Auto-Modelle hinzufügt werden, anstatt das vorhandene Auto-Modell zu ersetzen, und die Maße vorhandener Komponentenmodelle mit Hilfe der neu verfügbaren Daten neu bewertet werden.
is_auto_ensemble_weights_reevaluation	*Flag*	Aktiviert die Neubewertung der automatischen Modellgewichtungen.
use_accumulated_factor	*Flag*	Der kumulierte Faktor wird zur Berechnung kumulierter Maße verwendet.
accumulated_factor	*Zahl (Double)*	Der Maximalwert ist 0.99 und der Minimalwert ist 0.85.
use_accumulated_reducing	*Flag*	Führt eine Modellreduktion basierend auf dem kumulierten Grenzwert bei der Modellaktualisierung durch.
accumulated_reducing_limit	*Zahl (Double)*	Der Maximalwert ist 0.7 und der Minimalwert ist 0.1.
use_accumulated_weighted_evaluation	*Flag*	Wenn das nach Bewertung gewichtete Voting als Ensemble-Methode ausgewählt wurde, wird das kumulierte Evaluierungsmaß für das Voting verwendet.
flag_voting_tie_selection	Random HighestConfidence RawPropensity	Wenn eine Voting-Methode ausgewählt ist, gibt diese Einstellung an, wie Gleichstände aufgelöst werden sollen. Diese Einstellung gilt nur, wenn das ausgewählte Ziel ein Flagfeld ist.
set_ensemble_method	Voting EvaluationWeightedVoting ConfidenceWeightedVoting HighestConfidence	Gibt an, welche Methode für die Bestimmung des Ensemble-Score verwendet werden soll. Diese Einstellung gilt nur, wenn das ausgewählte Ziel ein Setfeld ist.

Tabelle 158. Eigenschaften von "applyautoclassifiernode" (Forts.)		
Eigenschaften von applyautoclassifiernode	**Werte**	**Eigenschaftsbeschreibung**
set_voting_tie_selection	Random HighestConfidence	Wenn eine Voting-Methode ausgewählt ist, gibt diese Einstellung an, wie Gleichstände aufgelöst werden sollen. Diese Einstellung gilt nur, wenn das ausgewählte Ziel ein nominales Feld ist.

Eigenschaften von "applyautoclusternode"

Mithilfe von Modellierungsknoten des Typs "Autom. Cluster" kann ein Modellnugget vom Typ "Autom. Cluster" generiert werden. Der Scriptname dieses Modellnuggets lautet *applyautoclusternode*. Für dieses Modellnugget gelten keine weiteren Eigenschaften. Weitere Informationen zum Scripting für den Modellierungsknoten selbst finden Sie unter „autoclusterernode, Eigenschaften” auf Seite 235.

Eigenschaften von "applyautonumericnode"

Mithilfe von Modellierungsknoten des Typs "Autonumerisch" kann ein Modellnugget vom Typ "Autonumerisch" generiert werden. Der Scriptname dieses Modellnuggets lautet *applyautonumericnode*. Weitere Informationen zu Scripts für den Modellierungsknoten selbst finden Sie in „autonumericnode, Eigenschaften” auf Seite 237.

Tabelle 159. applyautonumericnode, Eigenschaften		
Eigenschaften von applyautonumericnode	**Werte**	**Eigenschaftsbeschreibung**
calculate_standard_error	*Flag*	

Eigenschaften von "applybayesnetnode"

Mithilfe von Bayes-Netzmodellierungsknoten kann ein Modellnugget vom Typ "Bayes-Netz" generiert werden. Der Scriptname dieses Modellnuggets lautet *applybayesnetnode*. Weitere Informationen zu Scripts für den Modellierungsknoten selbst finden Sie in „Eigenschaften von "bayesnetnode"” auf Seite 239.

Tabelle 160. Eigenschaften von "applybayesnetnode"		
Eigenschaften von applybayesnetnode	**Werte**	**Eigenschaftsbeschreibung**
all_probabilities	*Flag*	
raw_propensity	*Flag*	
adjusted_propensity	*Flag*	
calculate_raw_propensities	*Flag*	
calculate_adjusted_propensities	*Flag*	

Eigenschaften von "applyc50node"

Mithilfe von C5.0-Modellierungsknoten kann ein C5.0-Modellnugget generiert werden. Der Scriptname dieses Modellnuggets lautet *applyc50node*. Weitere Informationen zu Scripts für den Modellierungsknoten selbst finden Sie in „c50node, Eigenschaften" auf Seite 242.

Tabelle 161. Eigenschaften von "applyc50node"

Eigenschaften von applyc50node	Werte	Eigenschaftsbeschreibung
sql_generate	udf Never NoMissingValues	Dient zur Festlegung von SQL-Erzeugungsoptionen beim Ausführen des Regelsets. Der Standardwert ist udf.
calculate_conf	*Flag*	Diese Eigenschaft ist verfügbar, wenn die SQL-Erzeugung aktiviert ist; sie enthält Konfidenzberechnungen im erzeugten Baum.
calculate_raw_propensities	*Flag*	
calculate_adjusted_propensities	*Flag*	

Eigenschaften von "applycarmanode"

Mithilfe von CARMA-Modellierungsknoten kann ein CARMA-Modellnugget generiert werden. Der Scriptname dieses Modellnuggets lautet *applycarmanode*. Für dieses Modellnugget gelten keine weiteren Eigenschaften. Weitere Informationen zu Scripts für den Modellierungsknoten selbst finden Sie in „carmanode, Eigenschaften" auf Seite 243.

Eigenschaften von "applycartnode"

Mithilfe von Modellierungsknoten vom Typ "C&R-Baum" kann ein Modellnugget vom Typ "C&R-Baum" generiert werden. Der Scriptname dieses Modellnuggets lautet *applycartnode*. Weitere Informationen zu Scripts für den Modellierungsknoten selbst finden Sie in „cartnode, Eigenschaften" auf Seite 244.

Tabelle 162. Eigenschaften von "applycartnode"

Eigenschaften von applycartnode	Werte	Eigenschaftsbeschreibung
enable_sql_generation	Never MissingValues NoMissingValues	Dient zur Festlegung von SQL-Erzeugungsoptionen beim Ausführen des Regelsets.
calculate_conf	*Flag*	Diese Eigenschaft ist verfügbar, wenn die SQL-Erzeugung aktiviert ist; sie enthält Konfidenzberechnungen im erzeugten Baum.

Tabelle 162. Eigenschaften von "applycartnode" (Forts.)		
Eigenschaften von applycartnode	**Werte**	**Eigenschaftsbeschreibung**
display_rule_id	*Flag*	Fügt ein Feld zur Scoring-Ausgabe hinzu, das die ID des Endknotens angibt, dem der jeweilige Datensatz zugewiesen ist.
calculate_raw_propensities	*Flag*	
calculate_adjusted_propensities	*Flag*	

Eigenschaften von "applychaidnode"

Mithilfe von CHAID-Modellierungsknoten kann ein CHAID-Modellnugget generiert werden. Der Scriptname dieses Modellnuggets lautet *applychaidnode*. Weitere Informationen zu Scripts für den Modellierungsknoten selbst finden Sie in „chaidnode, Eigenschaften” auf Seite 247.

Tabelle 163. Eigenschaften von "applychaidnode"		
Eigenschaften von applychaidnode	**Werte**	**Eigenschaftsbeschreibung**
enable_sql_generation	Never MissingValues	Dient zur Festlegung von SQL-Erzeugungsoptionen beim Ausführen des Regelsets.
calculate_conf	*Flag*	
display_rule_id	*Flag*	Fügt ein Feld zur Scoring-Ausgabe hinzu, das die ID des Endknotens angibt, dem der jeweilige Datensatz zugewiesen ist.
calculate_raw_propensities	*Flag*	
calculate_adjusted_propensities	*Flag*	

Eigenschaften von "applycoxregnode"

Mithilfe von Cox-Modellierungsknoten kann ein Cox-Modellnugget generiert werden. Der Scriptname dieses Modellnuggets lautet *applycoxregnode*. Weitere Informationen zu Scripts für den Modellierungsknoten selbst finden Sie in „coxregnode, Eigenschaften” auf Seite 250.

Tabelle 164. Eigenschaften von "applycoxregnode"		
Eigenschaften von applycoxregnode	**Werte**	**Eigenschaftsbeschreibung**
future_time_as	Intervals Fields	
time_interval	*number*	
num_future_times	*Integer*	

Tabelle 164. Eigenschaften von "applycoxregnode" (Forts.)		
Eigenschaften von applycoxregnode	**Werte**	**Eigenschaftsbeschreibung**
time_field	*Feld*	
past_survival_time	*Feld*	
all_probabilities	*Flag*	
cumulative_hazard	*Flag*	

Eigenschaften von "applydecisionlistnode"

Mithilfe von Entscheidungslistenmodellierungsknoten kann ein Modellnugget vom Typ "Entscheidungsliste" generiert werden. Der Scriptname dieses Modellnuggets lautet *applydecisionlistnode*. Weitere Informationen zu Scripts für den Modellierungsknoten selbst finden Sie in „Eigenschaften von "decisionlistnode"" auf Seite 252.

Tabelle 165. Eigenschaften von "applydecisionlistnode"		
Eigenschaften von applydecisionlistnode	**Werte**	**Eigenschaftsbeschreibung**
enable_sql_generation	*Flag*	Wenn dieser Wert wahr ist, versucht IBM SPSS Modeler, das Entscheidungslistenmodell über einen Push-back-Vorgang an SQL zurückzuführen.
calculate_raw_propensities	*Flag*	
calculate_adjusted_propensities	*Flag*	

Eigenschaften von "applydiscriminantnode"

Mithilfe von Diskriminanzmodellierungsknoten kann ein Diskriminanzmodellnugget generiert werden. Der Scriptname dieses Modellnuggets lautet *applydiscriminantnode*. Weitere Informationen zu Scripts für den Modellierungsknoten selbst finden Sie in „discriminantnode, Eigenschaften" auf Seite 254.

Tabelle 166. Eigenschaften von "applydiscriminantnode"		
Eigenschaften von applydiscriminantnode	**Werte**	**Eigenschaftsbeschreibung**
calculate_raw_propensities	*Flag*	
calculate_adjusted_propensities	*Flag*	

applyextension, Eigenschaften

Erweiterungsmodellknoten können verwendet werden, um ein Erweiterungsmodellnugget zu generieren. Der Scriptname dieses Modellnuggets lautet *applyextension*. Weitere Informationen zum Scripting für den Modellierungsknoten selbst finden Sie in „extensionmodelnode, Eigenschaften” auf Seite 256.

Beispiel für Python for Spark

```
#### script example for Python for Spark
applyModel = stream.findByType("extension_apply", None)

score_script = """
import json
import spss.pyspark.runtime
from pyspark.mllib.regression import LabeledPoint
from pyspark.mllib.linalg import DenseVector
from pyspark.mllib.tree import DecisionTreeModel
from pyspark.sql.types import StringType, StructField

cxt = spss.pyspark.runtime.getContext()

if cxt.isComputeDataModelOnly():
    _schema = cxt.getSparkInputSchema()
    _schema.fields.append(StructField("Prediction", StringType(), nullable=True))
    cxt.setSparkOutputSchema(_schema)
else:
    df = cxt.getSparkInputData()

    _modelPath = cxt.getModelContentToPath("TreeModel")
    metadata = json.loads(cxt.getModelContentToString("model.dm"))

    schema = df.dtypes[:]
    target = "Drug"
    predictors = ["Age","BP","Sex","Cholesterol","Na","K"]

    lookup = {}
    for i in range(0,len(schema)):
        lookup[schema[i][0]] = i

    def row2LabeledPoint(dm,lookup,target,predictors,row):
        target_index = lookup[target]
        tval = dm[target_index].index(row[target_index])
        pvals = []
        for predictor in predictors:
            predictor_index = lookup[predictor]
            if isinstance(dm[predictor_index],list):
                pval = row[predictor_index] in dm[predictor_index] and dm[predictor_index].in
dex(row[predictor_index]) or -1
            else:
                pval = row[predictor_index]
            pvals.append(pval)
        return LabeledPoint(tval, DenseVector(pvals))

    # convert dataframe to an RDD containing LabeledPoint
    lps = df.rdd.map(lambda row: row2LabeledPoint(metadata,lookup,target,predictors,row))
    treeModel = DecisionTreeModel.load(cxt.getSparkContext(), _modelPath);
    # score the model, produces an RDD containing just double values
    predictions = treeModel.predict(lps.map(lambda lp: lp.features))

    def addPrediction(x,dm,lookup,target):
        result = []
        for _idx in range(0, len(x[0])):
            result.append(x[0][_idx])
        result.append(dm[lookup[target]][int(x[1])])
        return result

    _schema = cxt.getSparkInputSchema()
    _schema.fields.append(StructField("Prediction", StringType(), nullable=True))
    rdd2 = df.rdd.zip(predictions).map(lambda x:addPrediction(x, metadata, lookup, target))
    outDF = cxt.getSparkSQLContext().createDataFrame(rdd2, _schema)
```

```
    cxt.setSparkOutputData(outDF)
"""
applyModel.setPropertyValue("python_syntax", score_script)
```

Beispiel für R

```
#### script example for R
applyModel.setPropertyValue("r_syntax", """
result<-predict(modelerModel,newdata=modelerData)
modelerData<-cbind(modelerData,result)
var1<-c(fieldName="NaPrediction",fieldLabel="",fieldStorage="real",fieldMeasure="",
fieldFormat="",fieldRole="")
modelerDataModel<-data.frame(modelerDataModel,var1)""")
```

Tabelle 167. applyextension, Eigenschaften

Eigenschaften von applyextension	**Werte**	**Eigenschaftsbeschreibung**
r_syntax	*string*	R-Scriptsyntax für das Modellscoring.
python_syntax	*string*	Python-Scriptsyntax für das Modellscoring.
use_batch_size	*Flag*	Aktiviert die Verwendung der Stapelverarbeitung.
batch_size	*Integer*	Geben Sie die Anzahl der Datensätze an, die in die einzelnen Stapel eingeschlossen werden sollen.
convert_flags	`StringsAndDoubles` `LogicalValues`	Option zum Konvertieren von Flagfeldern.
convert_missing	*Flag*	Option zum Konvertieren fehlender Werte in den R-Wert NA .
convert_datetime	*Flag*	Option zum Konvertieren von Variablen mit Datums- oder Datums-/Zeitformaten in R-Datums-/Zeitformate.
convert_datetime_class	`POSIXct` `POSIXlt`	Optionen, die angeben, in welches Format Variablen mit Datums- oder Datums-/Zeitformaten konvertiert werden.

Eigenschaften von "applyfactornode"

Mithilfe von Faktormodellierungsknoten kann ein Faktormodellnugget generiert werden. Der Scriptname dieses Modellnuggets lautet *applyfactornode*. Für dieses Modellnugget gelten keine weiteren Eigenschaften. Weitere Informationen zu Scripts für den Modellierungsknoten selbst finden Sie in „factornode, Eigenschaften” auf Seite 259.

Eigenschaften von "applyfeatureselectionnode"

Mithilfe von Modellierungsknoten vom Typ "Merkmalauswahl" kann ein Modellnugget vom Typ "Merkmalauswahl" generiert werden. Der Scriptname dieses Modellnuggets lautet *applyfeatureselectionnode*. Weitere Informationen zu Scripts für den Modellierungsknoten selbst finden Sie in „featureselectionnode, Eigenschaften” auf Seite 261.

Tabelle 168. applyfeatureselectionnode, Eigenschaften		
Eigenschaften von applyfeatureselectionnode	**Werte**	**Eigenschaftsbeschreibung**
selected_ranked_fields		Gibt an, welche Felder mit Rangzahlen im Modell-Browser markiert werden.
selected_screened_fields		Gibt an, welche im Screening untersuchten Felder im Modell-Browser markiert werden.

Eigenschaften von "applygeneralizedlinearnode"

Mithilfe von Modellierungsknoten vom Typ "Verallgemeinert linear (genlin)" kann ein Modellnugget vom Typ "Verallgemeinert linear" generiert werden. Der Scriptname dieses Modellnuggets lautet *applygeneralizedlinearnode*. Weitere Informationen zu Scripts für den Modellierungsknoten selbst finden Sie in „genlinnode, Eigenschaften” auf Seite 263.

Tabelle 169. applygeneralizedlinearnode, Eigenschaften		
Eigenschaften von applygeneralizedlinearnode	**Werte**	**Eigenschaftsbeschreibung**
calculate_raw_propensities	*Flag*	
calculate_adjusted_propensities	*Flag*	

Eigenschaften von "applyglmmnode"

Mithilfe von GLMM-Modellierungsknoten kann ein GLMM-Modellnugget generiert werden. Der Scriptname dieses Modellnuggets lautet *applyglmmnode*. Weitere Informationen zu Scripts für den Modellierungsknoten selbst finden Sie in „Eigenschaften von "glmmnode"” auf Seite 269.

Tabelle 170. Eigenschaften von "applyglmmnode"		
Eigenschaften von applyglmmnode	**Werte**	**Eigenschaftsbeschreibung**
confidence	onProbability onIncrease	Grundlage für die Berechnung des Konfidenzwerts für das Scoring: höchste vorhergesagte Wahrscheinlichkeit oder Differenz zwischen der höchsten und zweithöchsten vorhergesagten Wahrscheinlichkeit.
score_category_probabilities	*Flag*	Auf True gesetzt, werden die vorhergesagten Wahrscheinlichkeiten für kategoriale Ziele generiert. Für jede Kategorie wird ein Feld erstellt. Der Standardwert ist False.
max_categories	*Integer*	Maximal zu erstellende Anzahl an Kategorien, für die Wahrscheinlichkeiten vorhergesagt werden sollen. Wird nur verwendet, wenn score_category_probabilities auf True gesetzt ist.

Tabelle 170. Eigenschaften von "applyglmmnode" (Forts.)		
Eigenschaften von applyglmmnode	**Werte**	**Eigenschaftsbeschreibung**
score_propensity	*Flag*	Auf True gesetzt, werden Raw-Propensity-Scores (die die Wahrscheinlichkeit für "True" angeben) für Modelle mit Flagzielen erzeugt. Bei aktiven Partitionen werden außerdem Adjusted-Propensitys-Scores anhand der Testpartition erzeugt. Der Standardwert ist False.
enable_sql_generation	udf native	Dient zur Festlegung von SQL-Erzeugungsoptionen beim Ausführen des Datenstroms. Entweder kann ein Pushback an die Datenbank ausgeführt und mit einem SPSS® Modeler Server-Scoring-Adapter gescort werden (falls eine Verbindung zu einer Datenbank mit einem installierten Scoring-Adapter besteht) oder es kann SPSS Modeler gescort werden. Der Standardwert ist udf.

Eigenschaften von "applygle"

Mithilfe des GLE-Modellierungsknotens kann ein GLE-Modellnugget generiert werden. Der Scriptname dieses Modellnuggets lautet *applygle*. Weitere Informationen zum Scripting für den Modellierungsknoten selbst finden Sie in „Eigenschaften von "gle"" auf Seite 275.

Tabelle 171. Eigenschaften von "applygle"		
Eigenschaften von applygle	**Werte**	**Eigenschaftsbeschreibung**
enable_sql_generation	udf native	Dient zur Festlegung von SQL-Erzeugungsoptionen beim Ausführen des Datenstroms. Entweder kann ein Pushback an die Datenbank ausgeführt und mit einem SPSS Modeler Server-Scoring-Adapter gescort werden (falls eine Verbindung zu einer Datenbank mit installiertem Scoring-Adapter besteht), oder es kann in SPSS Modeler gescort werden.

Eigenschaften von "applygmm"

Mithilfe des Knotens "Gaußsche Mischverteilung" kann ein Modellnugget für gaußsche Mischverteilung generiert werden. Der Scriptname dieses Modellnuggets lautet *applygmm*. Die Eigenschaften in der folgenden Tabelle sind in Version 18.2.1.1 und höher verfügbar. Weitere Informationen zum Scripting für den Modellierungsknoten selbst finden Sie in „Eigenschaften von "gmm"" auf Seite 445.

Tabelle 172. Eigenschaften von "applygmm"		
Eigenschaften von applygmm	**Datentyp**	**Eigenschaftsbeschreibung**
centers		
item_count		
total		
dimension		
components		
partition		

Eigenschaften von "applykmeansnode"

Mithilfe von K-Means-Modellierungsknoten kann ein K-Means-Modellnugget generiert werden. Der Scriptname dieses Modellnuggets lautet *applykmeansnode*. Für dieses Modellnugget gelten keine weiteren Eigenschaften. Weitere Informationen zu Scripts für den Modellierungsknoten selbst finden Sie in „kmeansnode, Eigenschaften” auf Seite 282.

Eigenschaften von "applyknnnode"

Mithilfe von KNN-Modellierungsknoten kann ein KNN-Modellnugget generiert werden. Der Scriptname dieses Modellnuggets lautet *applyknnnode*. Weitere Informationen zu Scripts für den Modellierungsknoten selbst finden Sie in „knnnode, Eigenschaften” auf Seite 284.

Tabelle 173. Eigenschaften von "applyknnnode"		
Eigenschaften von applyknnnode	**Werte**	**Eigenschaftsbeschreibung**
all_probabilities	*Flag*	
save_distances	*Flag*	

Eigenschaften von "applykohonennode"

Mithilfe von Kohonen-Modellierungsknoten kann ein Kohonen-Modellnugget generiert werden. Der Scriptname dieses Modellnuggets lautet *applykohonennode*. Für dieses Modellnugget gelten keine weiteren Eigenschaften. Weitere Informationen zu Scripts für den Modellierungsknoten selbst finden Sie in „c50node, Eigenschaften” auf Seite 242.

Eigenschaften von "applylinearnode"

Mithilfe von Cox-Modellierungsknoten kann ein Nugget für ein lineares Modell generiert werden. Der Scriptname dieses Modellnuggets lautet *applylinearnode*. Weitere Informationen zu Scripts für den Modellierungsknoten selbst finden Sie in „Eigenschaften von "linearnode"” auf Seite 287.

Tabelle 174. Eigenschaften von "applylinearnode"		
Eigenschaften von linear	**Werte**	**Eigenschaftsbeschreibung**
use_custom_name	*Flag*	
custom_name	*string*	

Tabelle 174. Eigenschaften von "applylinearnode" (Forts.)		
Eigenschaften von `linear`	**Werte**	**Eigenschaftsbeschreibung**

`enable_sql_generation`	`udf`
	
	`native`
	
	`puresql`
		Dient zur Festlegung von SQL-Erzeugungsoptionen beim Ausführen des Datenstroms. Entweder kann ein Pushback an die Datenbank ausgeführt und mit einem SPSS® Modeler Server-Scoring-Adapter gescort werden (falls eine Verbindung zu einer Datenbank mit einem installierten Scoring-Adapter besteht) oder es kann SPSS Modeler gescort werden oder es kann ein Pushback an die Datenbank ausgeführt und mit SQL gescort werden.
		
		Der Standardwert ist `udf`.

Eigenschaften von "applylinearasnode"

Mithilfe von Linear-AS-Modellierungsknoten kann ein Linear-AS-Modellnugget generiert werden. Der Scriptname dieses Modellnuggets lautet *applylinearasnode*. Weitere Informationen zu Scripts für den Modellierungsknoten selbst finden Sie in „Eigenschaften von "linearasnode"" auf Seite 289.

Tabelle 175. Eigenschaften von "applylinearasnode"		
Eigenschaft `applylinearasnode`	**Werte**	**Eigenschaftsbeschreibung**

`enable_sql_generation`	`udf`
	
	`native`
		Der Standardwert ist `udf`.

Eigenschaften von "applylogregnode"

Mithilfe von Modellierungsknoten vom Typ "Logistische Regression" kann ein Modellnugget vom Typ "Logistische Regression" generiert werden. Der Scriptname dieses Modellnuggets lautet *applylogregnode*. Weitere Informationen zu Scripts für den Modellierungsknoten selbst finden Sie in „logregnode, Eigenschaften" auf Seite 291.

Tabelle 176. Eigenschaften von "applylogregnode"		
Eigenschaften von `applylogregnode`	**Werte**	**Eigenschaftsbeschreibung**
`calculate_raw_propensities`	*Flag*	
`calculate_conf`	*Flag*	
`enable_sql_generation`	*Flag*	

Eigenschaften von "applylsvmnode"

Mithilfe von LSVM-Modellierungsknoten kann ein LSVM-Modellnugget generiert werden. Der Scriptname dieses Modellnuggets lautet *applylsvmnode*. Weitere Informationen zum Scripting für den Modellierungsknoten selbst finden Sie in „lsvmnode, Eigenschaften" auf Seite 297.

Tabelle 177. Eigenschaften von "applylsvmnode"		
Eigenschaften von applylsvmnode	**Werte**	**Eigenschaftsbeschreibung**
calculate_raw_propensities	*Flag*	Gibt an, ob Raw-Propensity-Scores berechnet werden sollen.
enable_sql_generation	udf native	Gibt an, das Scoring mithilfe des Scoring-Adapters (falls installiert) oder bei der Verarbeitung durchgeführt werden soll oder ob das Scoring außerhalb der Datenbank durchgeführt werden soll.

Eigenschaften von "applyneuralnetnode"

Mithilfe von Netzmodellierungsknoten kann ein Netzmodellnugget generiert werden. Der Scriptname dieses Modellnuggets lautet *applyneuralnetnode*. Weitere Informationen zu Scripts für den Modellierungsknoten selbst finden Sie in „neuralnetnode, Eigenschaften” auf Seite 298.

Vorsicht: In dieser Version ist eine neuere Fassung des Netznuggets mit erweiterten Funktionen verfügbar, die im nächsten Abschnitt beschrieben wird (*applyneuralnetwork*). Die Vorgängerversion ist zwar weiterhin verfügbar, wir empfehlen jedoch die Aktualisierung Ihrer Scripts zur Verwendung der neuen Version. Zur Referenz sind hier Details zur Vorgängerversion enthalten, doch wird die Unterstützung dafür in zukünftigen Versionen wegfallen.

Tabelle 178. Eigenschaften von "applyneuralnetnode"		
Eigenschaften von applyneuralnetnode	**Werte**	**Eigenschaftsbeschreibung**
calculate_conf	*Flag*	Diese Eigenschaft ist verfügbar, wenn die SQL-Erzeugung aktiviert ist; sie enthält Konfidenzberechnungen im erzeugten Baum.
enable_sql_generation	*Flag*	
nn_score_method	Difference SoftMax	
calculate_raw_propensities	*Flag*	
calculate_adjusted_propensities	*Flag*	

applyneuralnetworknode, Eigenschaften

Mithilfe von Netzmodellierungsknoten kann ein Netzmodellnugget generiert werden. Der Scriptname dieses Modellnuggets lautet *applyneuralnetworknode*. Weitere Informationen zum Scripting für den Modellierungsknoten selbst finden Sie in Eigenschaften von "neuralnetworknode".

Tabelle 179. applyneuralnetworknode, Eigenschaften		
Eigeschaften von applyneuralnetworknode	**Werte**	**Eigenschaftsbeschreibung**
use_custom_name	*Flag*	
custom_name	*string*	

Tabelle 179. applyneuralnetworknode, Eigenschaften (Forts.)		
Eigeschaften von applyneuralnetworknode	**Werte**	**Eigenschaftsbeschreibung**
confidence	onProbability onIncrease	
score_category_probabilities	*Flag*	
max_categories	*number*	
score_propensity	*Flag*	
enable_sql_generation	udf native puresql	Dient zur Festlegung von SQL-Erzeugungsoptionen beim Ausführen des Datenstroms. Entweder kann ein Pushback an die Datenbank ausgeführt und mit einem SPSS® Modeler Server-Scoring-Adapter gescort werden (falls eine Verbindung zu einer Datenbank mit einem installierten Scoring-Adapter besteht) oder es kann SPSS Modeler gescort werden oder es kann ein Pushback an die Datenbank ausgeführt und mit SQL gescort werden. Der Standardwert ist udf.

Eigenschaften von "applyocsvmnode"

Knoten "One-Class SVM" können verwendet werden, um ein Modellnugget "One-Class SVM" zu generieren. Der Scriptname dieses Modellnuggets lautet *applyocsvmnode*. Für dieses Modellnugget gelten keine weiteren Eigenschaften. Weitere Informationen zum Scripting für den Modellierungsknoten selbst finden Sie in „Eigenschaften von "ocsvmnode"” auf Seite 451.

Eigenschaften von "applyquestnode"

Mithilfe von QUEST-Modellierungsknoten kann ein QUEST-Modellnugget generiert werden. Der Scriptname dieses Modellnuggets lautet *applyquestnode*. Weitere Informationen zu Scripts für den Modellierungsknoten selbst finden Sie in „questnode, Eigenschaften” auf Seite 303.

Tabelle 180. Eigenschaften von "applyquestnode"		
Eigenschaften von applyquestnode	**Werte**	**Eigenschaftsbeschreibung**
enable_sql_generation	Never MissingValues NoMissingValues	Dient zur Festlegung von SQL-Erzeugungsoptionen beim Ausführen des Regelsets.
calculate_conf	*Flag*	

Tabelle 180. Eigenschaften von "applyquestnode" (Forts.)		
Eigenschaften von applyquestnode	**Werte**	**Eigenschaftsbeschreibung**
display_rule_id	*Flag*	Fügt ein Feld zur Scoring-Ausgabe hinzu, das die ID des Endknotens angibt, dem der jeweilige Datensatz zugewiesen ist.
calculate_raw_propensities	*Flag*	
calculate_adjusted_propensities	*Flag*	

Eigenschaften von "applyr"

Mithilfe von R-Erstellungsknoten kann ein R-Modellnugget generiert werden. Der Scriptname dieses Modellnuggets lautet *applyr*. Weitere Informationen zu Scripts für den Modellierungsknoten selbst finden Sie in „buildr, Eigenschaften” auf Seite 241.

Tabelle 181. Eigenschaften von "applyr"		
Eigenschaften von applyr	**Werte**	**Eigenschaftsbeschreibung**
score_syntax	*string*	R-Scriptsyntax für das Modellscoring.
convert_flags	StringsAndDoubles LogicalValues	Option zum Konvertieren von Flagfeldern.
convert_datetime	*Flag*	Option zum Konvertieren von Variablen mit Datums- oder Datums-/Zeitformaten in R-Datums-/Zeitformate.
convert_datetime_class	POSIXct POSIXlt	Optionen, die angeben, in welches Format Variablen mit Datums- oder Datums-/Zeitformaten konvertiert werden.
convert_missing	*Flag*	Option zum Konvertieren fehlender Werte in RNA-Werte.
use_batch_size	*Flag*	Aktiviert die Verwendung der Stapelverarbeitung.
batch_size	*Integer*	Geben Sie die Anzahl der Datensätze an, die in die einzelnen Stapel eingeschlossen werden sollen.

Eigenschaften von "applyrandomtrees"

Mithilfe des Random Trees-Modellierungsknotens kann ein Random Trees-Modellnugget generiert werden. Der Scriptname dieses Modellnuggets lautet *applyrandomtrees*. Weitere Informationen zum Scripting für den Modellierungsknoten selbst finden Sie in „randomtrees, Eigenschaften” auf Seite 306.

Tabelle 182. applyrandomtrees, Eigenschaften		
Eigenschaften von applyrandomtrees	**Werte**	**Eigenschaftsbeschreibung**
calculate_conf	*Flag*	Diese Eigenschaft enthält Konfidenzberechnungen im generierten Baum.
enable_sql_generation	udf native	Dient zur Festlegung von SQL-Erzeugungsoptionen beim Ausführen des Datenstroms. Entweder kann ein Pushback an die Datenbank ausgeführt und mit einem SPSS Modeler Server-Scoring-Adapter gescort werden (falls eine Verbindung zu einer Datenbank mit installiertem Scoring-Adapter besteht), oder es kann in SPSS Modeler gescort werden.

Eigenschaften von "applyregressionnode"

Mithilfe von Modellierungsknoten vom Typ "Lineare Regression" kann ein Modellnugget vom Typ "Lineare Regression" generiert werden. Der Scriptname dieses Modellnuggets lautet *applyregressionnode*. Für dieses Modellnugget gelten keine weiteren Eigenschaften. Weitere Informationen zu Scripts für den Modellierungsknoten selbst finden Sie in „regressionnode, Eigenschaften” auf Seite 308.

Eigenschaften von "applyselflearningnode"

Mithilfe von Modellierungsknoten vom Typ (Self-Learning Response Model (SLRM) kann ein SLRM-Modellnugget generiert werden. Der Scriptname dieses Modellnuggets lautet *applyselflearningnode*. Weitere Informationen zu Scripts für den Modellierungsknoten selbst finden Sie in „slrmnode, Eigenschaften” auf Seite 312.

Tabelle 183. Eigenschaften von "applyselflearningnode"		
Eigenschaften von applyselflearningnode	**Werte**	**Eigenschaftsbeschreibung**
max_predictions	*number*	
randomization	*number*	
scoring_random_seed	*number*	
sort	ascending descending	Gibt an, ob die Angebote mit den höchsten oder die mit den niedrigsten Scores zuerst angezeigt werden.
model_reliability	*Flag*	Berücksichtigt die Option für die Reliabilität auf der Registerkarte "Einstellungen".

Eigenschaften von "applysequencenode"

Mithilfe von Sequenzmodellierungsknoten kann ein Sequenzmodellnugget generiert werden. Der Scriptname dieses Modellnuggets lautet *applysequencenode*. Für dieses Modellnugget gelten keine weiteren Eigenschaften. Weitere Informationen zu Scripts für den Modellierungsknoten selbst finden Sie in „sequencenode, Eigenschaften” auf Seite 311.

Eigenschaften von "applysvmnode"

Mithilfe von SVM-Modellierungsknoten kann ein SVM-Modellnugget generiert werden. Der Scriptname dieses Modellnuggets lautet *applysvmnode*. Weitere Informationen zu Scripts für den Modellierungsknoten selbst finden Sie in „svmnode, Eigenschaften" auf Seite 320.

Tabelle 184. applysvmnode, Eigenschaften

Eigenschaften von applysvmnode	Werte	Eigenschaftsbeschreibung
all_probabilities	*Flag*	
calculate_raw_propensities	*Flag*	
calculate_adjusted_propensities	*Flag*	

Eigenschaften von "applystpnode"

Der STP-Modellierungsknoten kann zum Generieren eines zugehörigen Modellnuggets verwendet werden, das die Modellausgabe im Ausgabeviewer anzeigt. Der Scriptname dieses Modellnuggets lautet *applystpnode*. Weitere Informationen zum Scripting für den Modellierungsknoten selbst finden Sie in „stpnode, Eigenschaften" auf Seite 314.

Tabelle 185. Eigenschaften von "applystpnode"

Eigenschaften von applystpnode	Datentyp	Eigenschaftsbeschreibung
uncertainty_factor	*boolesch*	Minimum: 0, Maximum: 100.

Eigenschaften von "applytcmnode"

Mithilfe von TCM-Modellierungsknoten (Temporal Causal Modeling) kann ein TCM-Modellnugget generiert werden. Der Scriptname dieses Modellnuggets lautet *applytcmnode*. Weitere Informationen zum Scripting für den Modellierungsknoten selbst finden Sie in „Eigenschaften von "tcmnode"" auf Seite 322.

Tabelle 186. Eigenschaften von "applytcmnode"

Eigenschaften von applytcmnode	Werte	Eigenschaftsbeschreibung
ext_future	*Boolesch*	
ext_future_num	*Integer*	
noise_res	*Boolesch*	
conf_limits	*Boolesch*	
target_fields	*Liste*	
target_series	*Liste*	

Eigenschaften von "applyts"

Mithilfe des Zeitreihenmodellierungsknotens kann ein Zeitreihenmodellnugget generiert werden. Der Scriptname dieses Modellnuggets lautet *applyts*. Weitere Informationen zum Scripting für den Modellierungsknoten selbst finden Sie in „ts, Eigenschaften" auf Seite 327.

Tabelle 187. Eigenschaften von applyts		
Eigenschaften von applyts	**Werte**	**Eigenschaftsbeschreibung**
extend_records_into_future	*boolesch*	
ext_future_num	*Integer*	
compute_future_values_input	*boolesch*	
forecastperiods	*Integer*	
noise_res	*Boolesch*	
conf_limits	*Boolesch*	
target_fields	*Liste*	
target_series	*Liste*	
includeTargets	*Feld*	

Eigenschaften des Knotens "applytimeseriesnode" (nicht mehr unterstützt)

Mithilfe des Zeitreihenmodellierungsknotens kann ein Zeitreihenmodellnugget generiert werden. Der Scriptname dieses Modellnuggets lautet *applytimeseriesnode*. Weitere Informationen zu Scripts für den Modellierungsknoten selbst finden Sie in „Eigenschaften von "timeseriesnode" (nicht mehr unterstützt)" auf Seite 335.

Tabelle 188. Eigenschaften von "applytimeseriesnode"		
Eigenschaften von applytimeseriesnode	**Werte**	**Eigenschaftsbeschreibung**
calculate_conf	*Flag*	
calculate_residuals	*Flag*	

Eigenschaften von "applytreeas"

Mithilfe von Tree-AS-Modellierungsknoten kann ein Tree-AS-Modellnugget generiert werden. Der Scriptname dieses Modellnuggets lautet *applytreenas*. Weitere Informationen zum Scripting für den Modellierungsknoten selbst finden Sie in „treeas, Eigenschaften" auf Seite 338.

Tabelle 189. applytreeas, Eigenschaften		
Eigenschaften von applytreeas	**Werte**	**Eigenschaftsbeschreibung**
calculate_conf	*Flag*	Diese Eigenschaft enthält Konfidenzberechnungen im generierten Baum.
display_rule_id	*Flag*	Fügt ein Feld zur Scoring-Ausgabe hinzu, das die ID des Endknotens angibt, dem der jeweilige Datensatz zugewiesen ist.

Tabelle 189. applytreeas, Eigenschaften (Forts.)		
Eigenschaften von applytreeas	**Werte**	**Eigenschaftsbeschreibung**
enable_sql_generation	udf native	Dient zur Festlegung von SQL-Erzeugungsoptionen beim Ausführen des Datenstroms. Entweder kann ein Pushback an die Datenbank ausgeführt und mit einem SPSS Modeler Server-Scoring-Adapter gescort werden (falls eine Verbindung zu einer Datenbank mit installiertem Scoring-Adapter besteht), oder es kann in SPSS Modeler gescort werden.

Eigenschaften von "applytwostepnode"

Mithilfe von TwoStep-Modellierungsknoten kann ein TwoStep-Modellnugget generiert werden. Der Scriptname dieses Modellnuggets lautet *applytwostepnode*. Für dieses Modellnugget gelten keine weiteren Eigenschaften. Weitere Informationen zu Scripts für den Modellierungsknoten selbst finden Sie in „Eigenschaften von "twostepnode"" auf Seite 341.

Eigenschaften von "applytwostepAS"

Mithilfe von TwoStep AS-Modellierungsknoten kann ein TwoStep AS-Modellnugget generiert werden. Der Scriptname dieses Modellnuggets lautet *applytwostepAS*. Weitere Informationen zu Scripts für den Modellierungsknoten selbst finden Sie in „Eigenschaften von "twostepAS"" auf Seite 342.

Tabelle 190. Eigenschaften von "applytwostepAS"		
Eigenschaften von applytwostepAS	**Werte**	**Eigenschaftsbeschreibung**
enable_sql_generation	udf native	Dient zur Festlegung von SQL-Erzeugungsoptionen beim Ausführen des Datenstroms. Entweder kann ein Pushback an die Datenbank ausgeführt und mit einem SPSS® Modeler Server-Scoring-Adapter gescort werden (falls eine Verbindung zu einer Datenbank mit einem installierten Scoring-Adapter besteht) oder es kann SPSS Modeler gescort werden. Der Standardwert ist udf.

Eigenschaften von "applyxgboosttreenode"

Der Knoten "XGBoost Tree" kann verwendet werden, um ein Modellnugget "XGBoost Tree" zu generieren. Der Scriptname dieses Modellnuggets lautet *applyxgboosttreenode*. Die Eigenschaften in der folgenden Tabelle wurden in Version 18.2.1.1 hinzugefügt. Weitere Informationen zum Scripting für den Modellierungsknoten selbst finden Sie in „Eigenschaften von "xgboosttreenode"" auf Seite 460.

Tabelle 191. Eigenschaften von "applyxgboosttreenode"		
Eigenschaften von applyxgboosttreenode	**Datentyp**	**Eigenschaftsbeschreibung**
use_model_name		
model_name		

applyxgboostlinearnode, Eigenschaften

Knoten "XGBoost Linear" können verwendet werden, um ein Modellnugget "XGBoost Linear" zu generieren. Der Scriptname dieses Modellnuggets lautet *applyxgboostlinearnode*. Für dieses Modellnugget gelten keine weiteren Eigenschaften. Weitere Informationen zum Scripting für den Modellierungsknoten selbst finden Sie in „Eigenschaften von "xgboostlinearnode"" auf Seite 459.

Eigenschaften von "hdbscannugget"

Der HDBSCAN-Knoten kann verwendet werden, um ein HDBSCAN-Modellnugget zu generieren. Der Scriptname dieses Modellnuggets lautet hdbscannugget. Für dieses Modellnugget gelten keine weiteren Eigenschaften. Weitere Informationen zum Scripting für den Modellierungsknoten selbst finden Sie in „Eigenschaften von "hdbscannode"" auf Seite 446.

Eigenschaften von "kdeapply"

Mithilfe des KDE-Modellierungsknotens kann ein KDE-Modellnugget generiert werden. Der Scriptname dieses Modellnuggets lautet kdeapply. Informationen zum Scripting für den Modellierungsknoten selbst finden Sie in „Eigenschaften von "kdemodel"" auf Seite 448.

Tabelle 192. Eigenschaften von "kdeapply"		
Eigenschaften von kdeapply	**Datentyp**	**Eigenschaftsbeschreibung**
outLogDensity In out_log_density umbenannt ab Version 18.2.1.1.	*Boolesch*	Geben Sie True bzw. False an, um den Logarithmusdichtewert in die Ausgabe einzuschließen bzw. daraus auszuschließen. Der Standardwert ist False.

Kapitel 15. Eigenschaften von Datenbankmodellierungsknoten

IBM SPSS Modeler unterstützt die Integration mit Data-Mining- und Modellierungstools von Datenbankanbietern, einschließlich Microsoft SQL Server Analysis Services, Oracle Data Mining, und IBM Netezza Analytics. Sie können mithilfe von datenbankeigenen Algorithmen Modelle erstellen und scoren, ohne dazu die IBM SPSS Modeler-Anwendung verlassen zu müssen. Datenbankmodelle können außerdem mithilfe von Scripterstellung unter Verwendung der in diesem Abschnitt beschriebenen Eigenschaften erstellt und bearbeitet werden.

Der folgende Script-Auszug veranschaulicht z. B. die Erstellung eines Microsoft Decision Trees-Modells über die IBM SPSS Modeler-Scriptschnittstelle:

```
stream = modeler.script.stream()
msbuilder = stream.createAt("mstreenode", "MSBuilder", 200, 200)

msbuilder.setPropertyValue("analysis_server_name", 'localhost')
msbuilder.setPropertyValue("analysis_database_name", 'TESTDB')
msbuilder.setPropertyValue("mode", 'Expert')
msbuilder.setPropertyValue("datasource", 'LocalServer')
msbuilder.setPropertyValue("target", 'Drug')
msbuilder.setPropertyValue("inputs", ['Age', 'Sex'])
msbuilder.setPropertyValue("unique_field", 'IDX')
msbuilder.setPropertyValue("custom_fields", True)
msbuilder.setPropertyValue("model_name", 'MSDRUG')

typenode = stream.findByType("type", None)
stream.link(typenode, msbuilder)
results = []
msbuilder.run(results)
msapplier = stream.createModelApplierAt(results[0], "Drug", 200, 300)
tablenode = stream.createAt("table", "Results", 300, 300)
stream.linkBetween(msapplier, typenode, tablenode)
msapplier.setPropertyValue("sql_generate", True)
tablenode.run([])
```

Knoteneigenschaften für Microsoft-Modellierung

Eigenschaften von Microsoft-Modellierungsknoten

Allgemeine Eigenschaften

Folgende Eigenschaften haben alle Microsoft-Datenbankmodellierungsknoten gemeinsam.

Tabelle 193. Allgemeine Eigenschaften von Microsoft-Knoten

Allgemeine Eigenschaften von Microsoft-Knoten	**Werte**	**Eigenschaftsbeschreibung**
`analysis_database_name`	*string*	Name der Analysis Services-Datenbank.
`analysis_server_name`	*string*	Name des Analysis Services-Hosts.
`use_transactional_data`	*Flag*	Gibt an, ob die Eingabedaten in Tabellen- oder Transaktionsformat vorliegen.
`inputs`	*Liste*	Eingabefelder für Tabellendaten.

Tabelle 193. Allgemeine Eigenschaften von Microsoft-Knoten (Forts.)		
Allgemeine Eigenschaften von Microsoft-Knoten	**Werte**	**Eigenschaftsbeschreibung**
target	*Feld*	Vorhergesagtes Feld (gilt nicht für MS-Clustering- oder Sequenz-Clustering-Knoten).
unique_field	*Feld*	Schlüsselfeld.
msas_parameters	*strukturiert*	Algorithmusparameter. Weitere Informationen finden Sie im Thema „Algorithmusparameter" auf Seite 369.
with_drillthrough	*Flag*	Mit Drillthrough-Option.

MS-Entscheidungsbaum

Für Knoten vom Typ mstreenode sind keine speziellen Eigenschaften definiert. Informationen finden Sie unter den allgemeinen Microsoft-Eigenschaften am Anfang dieses Abschnitts.

MS-Clustering

Für Knoten vom Typ msclusternode sind keine speziellen Eigenschaften definiert. Informationen finden Sie unter den allgemeinen Microsoft-Eigenschaften am Anfang dieses Abschnitts.

MS-Assoziationsregeln

Für Knoten vom Typ msassocnode sind die folgenden speziellen Eigenschaften verfügbar.

Tabelle 194. Eigenschaften von "msassocnode"		
Eigenschaften von msassocnode	**Werte**	**Eigenschaftsbeschreibung**
id_field	*Feld*	Identifiziert jede Transaktion in den Daten.
trans_inputs	*Liste*	Eingabefelder für Transaktionsdaten.
transactional_target	*Feld*	Prädiktorfeld (Transaktionsdaten).

MS Naive Bayes

Für Knoten vom Typ msbayesnode sind keine speziellen Eigenschaften definiert. Informationen finden Sie unter den allgemeinen Microsoft-Eigenschaften am Anfang dieses Abschnitts.

MS - Lineare Regression

Für Knoten vom Typ msregressionnode sind keine speziellen Eigenschaften definiert. Informationen finden Sie unter den allgemeinen Microsoft-Eigenschaften am Anfang dieses Abschnitts.

MS - Neuronales Netz

Für Knoten vom Typ msneuralnetworknode sind keine speziellen Eigenschaften definiert. Informationen finden Sie unter den allgemeinen Microsoft-Eigenschaften am Anfang dieses Abschnitts.

MS - Logistische Regression

Für Knoten vom Typ mslogisticnode sind keine speziellen Eigenschaften definiert. Informationen finden Sie unter den allgemeinen Microsoft-Eigenschaften am Anfang dieses Abschnitts.

MS Time Series

Für Knoten vom Typ `mstimeseriesnode` sind keine speziellen Eigenschaften definiert. Informationen finden Sie unter den allgemeinen Microsoft-Eigenschaften am Anfang dieses Abschnitts.

MS Sequenz-Clustering

Für Knoten vom Typ `mssequenceclusternode` sind die folgenden speziellen Eigenschaften verfügbar.

Tabelle 195. Eigenschaften von "mssequenceclusternode"

Eigenschaften von `mssequenceclusternode`	**Werte**	**Eigenschaftsbeschreibung**
`id_field`	*Feld*	Identifiziert jede Transaktion in den Daten.
`input_fields`	*Liste*	Eingabefelder für Transaktionsdaten.
`sequence_field`	*Feld*	Sequenz-ID.
`target_field`	*Feld*	Prädiktorfeld (Tabellendaten).

Algorithmusparameter

Jeder Microsoft-Datenbankmodelltyp verfügt über bestimmte Parameter, die mit der Eigenschaft `msas_parameters` festgelegt werden können, z. B.:

```
stream = modeler.script.stream()
msregressionnode = stream.findByType("msregression", None)
msregressionnode.setPropertyValue("msas_parameters", [["MAXIMUM_INPUT_ATTRI
BUTES", 255],
["MAXIMUM_OUTPUT_ATTRIBUTES", 255]])
```

Diese Parameter werden vom SQL-Server abgeleitet. Gehen Sie wie folgt vor, um die relevanten Parameter für die einzelnen Knoten anzuzeigen:

1. Platzieren Sie einen Datenbankquellenknoten im Erstellungsbereich.
2. Öffnen Sie den Datenbankquellenknoten.
3. Wählen Sie eine gültige Quelle in der Dropdown-Liste **Datenquelle** aus.
4. Wählen Sie eine gültige Tabelle in der Liste **Tabellenname** aus.
5. Klicken Sie auf **OK**, um den Datenbankquellenknoten zu schließen.
6. Fügen Sie den Microsoft-Datenbankmodellierungsknoten ein, dessen Eigenschaften aufgelistet werden sollen.
7. Öffnen Sie den Datenbankmodellierungsknoten.
8. Wählen Sie die Registerkarte **Experten** aus.

Die verfügbaren `msas_parameters`-Eigenschaften für diesen Knoten werden angezeigt.

Eigenschaften von Microsoft-Modellnuggets

Folgende Eigenschaften gelten für die Modellnuggets, die mithilfe der Microsoft-Datenbankmodellierungsknoten erstellt wurden.

MS-Entscheidungsbaum

Tabelle 196. Eigenschaften des MS-Entscheidungsbaums

Eigenschaften von applymstreenode	**Werte**	**Beschreibung**
analysis_database_name	*string*	Dieser Knoten kann direkt in einem Stream gesc-ort werden. Anhand dieser Eigenschaft wird der Name der Analysis Services-Datenbank identifiziert.
analysis_server_name	*string*	Name des Analyseserver-Hosts.
datasource	*string*	Name der SQL Server ODBC-Datenquelle (DSN).
sql_generate	*Flag* udf	Aktiviert die SQL-Erzeugung.

MS - Lineare Regression

Tabelle 197. MS Lineare Regression - Eigenschaften

Eigenschaften von applymsregressionnode	**Werte**	**Beschreibung**
analysis_database_name	*string*	Dieser Knoten kann direkt in einem Stream gesc-ort werden. Anhand dieser Eigenschaft wird der Name der Analysis Services-Datenbank identifiziert.
analysis_server_name	*string*	Name des Analyseserver-Hosts.

MS - Neuronales Netz

Tabelle 198. MS Neuronales Netz - Eigenschaften

Eigenschaften von applymsneuralnetworknode	**Werte**	**Beschreibung**
analysis_database_name	*string*	Dieser Knoten kann direkt in einem Stream gesc-ort werden. Anhand dieser Eigenschaft wird der Name der Analysis Services-Datenbank identifiziert.
analysis_server_name	*string*	Name des Analyseserver-Hosts.

MS - Logistische Regression

Tabelle 199. MS Logistische Regression - Eigenschaften		
Eigenschaften von applymslogisticnode	**Werte**	**Beschreibung**
analysis_database_name	*string*	Dieser Knoten kann direkt in einem Stream gesc-ort werden. Anhand dieser Eigenschaft wird der Name der Analysis Services-Datenbank identifiziert.
analysis_server_name	*string*	Name des Analyseserver-Hosts.

MS Time Series

Tabelle 200. MS Time Series - Eigenschaften		
Eigenschaften von applymstimeseriesnode	**Werte**	**Beschreibung**
analysis_database_name	*string*	Dieser Knoten kann direkt in einem Stream gesc-ort werden. Anhand dieser Eigenschaft wird der Name der Analysis Services-Datenbank identifiziert.
analysis_server_name	*string*	Name des Analyseserver-Hosts.
start_from	new_predicti-on `historical_ prediction`	Gibt an, ob Zukunfts- oder historische Vorhersa-gen getroffen werden.
new_step	*number*	Definiert die Startzeit für Zukunftsvorhersagen.
historical_step	*number*	Definiert die Startzeit für historische Vorhersa-gen.
end_step	*number*	Definiert die Endzeit für Vorhersagen.

MS Sequenz-Clustering

Tabelle 201. Eigenschaften von "MS Sequence Clustering"		
Eigenschaften von applymssequenceclusternode	**Werte**	**Beschreibung**
analysis_database_name	*string*	Dieser Knoten kann direkt in einem Stream gesc-ort werden. Anhand dieser Eigenschaft wird der Name der Analysis Services-Datenbank identifiziert.
analysis_server_name	*string*	Name des Analyseserver-Hosts.

Knoteneigenschaften für Oracle-Modellierung

Eigenschaften von Oracle-Modellierungsknoten

Folgende Eigenschaften haben alle Oracle-Datenbankmodellierungsknoten gemeinsam.

Tabelle 202. Allgemeine Eigenschaften von Oracle-Knoten

Allgemeine Eigenschaften von Oracle-Knoten	**Werte**	**Eigenschaftsbeschreibung**
target	*Feld*	
inputs	*Liste mit Feldern*	
partition	*Feld*	Feld wird verwendet, um die Daten in getrennte Stichproben für die Trainings-, Test- und Validierungsphase der Modellerstellung aufzuteilen.
datasource		
username		
password		
epassword		
use_model_name	*Flag*	
model_name	*string*	Benutzerdefinierter Name für neues Modell.
use_partitioned_data	*Flag*	Wenn ein Partitionsfeld definiert ist, gewährleistet diese Option, dass nur Daten aus der Trainingspartition für die Modellerstellung verwendet werden.
unique_field	*Feld*	
auto_data_prep	*Flag*	Aktiviert oder inaktiviert die automatische Datenvorbereitungsfunktion von Oracle (nur 11g-Datenbanken).
costs	*strukturiert*	Strukturierte Eigenschaft im Format: `[[drugA drugB 1.5] [drugA drugC 2.1]]`, wobei die Argumente in [] tatsächlich vorausgesagte Kosten sind.

mode	`Simple`
	
	`Expert`
		Wenn diese Einstellung auf `Simple` (Einfach) gesetzt ist, werden bestimmte Eigenschaften ignoriert (vgl. die Eigenschaften der einzelnen Knoten).

use_prediction_probability	*Flag*	
prediction_probability	*string*	
use_prediction_set	*Flag*	

Oracle Naive Bayes

Für Knoten vom Typ `oranbnode` sind folgende Eigenschaften verfügbar.

Tabelle 203. Eigenschaften von "oranbnode"		
Eigenschaften von oranbnode	**Werte**	**Eigenschaftsbeschreibung**
singleton_threshold	*number*	0,0-1,0.*
pairwise_threshold	*number*	0,0-1,0.*
priors	Data Equal Custom	
custom_priors	*strukturiert*	Strukturierte Eigenschaft im Format: set :oranbnode.custom_priors = [[drugA 1][drugB 2][drugC 3][drugX 4][drugY 5]]

* Eigenschaft wird ignoriert, wenn mode auf Simple gesetzt ist.

Oracle Adaptive Bayes

Für Knoten vom Typ oraabnnode sind folgende Eigenschaften verfügbar.

Tabelle 204. Eigenschaften von "oraabnnode"		
Eigenschaften von oraabnnode	**Werte**	**Eigenschaftsbeschreibung**
model_type	SingleFeature MultiFeature NaiveBayes	
use_execution_time_limit	*Flag*	*
execution_time_limit	*Integer*	Der Wert muss größer als 0 sein.*
max_naive_bayes_predictors	*Integer*	Der Wert muss größer als 0 sein.*
max_predictors	*Integer*	Der Wert muss größer als 0 sein.*
priors	Data Equal Custom	
custom_priors	*strukturiert*	Strukturierte Eigenschaft im Format: set :oraabnnode.custom_priors = [[drugA 1][drugB 2][drugC 3][drugX 4][drugY 5]]

* Eigenschaft wird ignoriert, wenn mode auf Simple gesetzt ist.

Oracle Support Vector Machines

Für Knoten vom Typ orasvmnode sind folgende Eigenschaften verfügbar.

Tabelle 205. Eigenschaften von "orasvmnode"

Eigenschaften von orasvmnode	Werte	Eigenschaftsbeschreibung
active_learning	Enable Disable	
kernel_function	Linear Gaussian System	
normalization_method	zscore minmax none	
kernel_cache_size	*Integer*	Nur gaußscher Kern. Der Wert muss größer als 0 sein.*
convergence_tolerance	*number*	Der Wert muss größer als 0 sein.*
use_standard_deviation	*Flag*	Nur gaußscher Kern.*
standard_deviation	*number*	Der Wert muss größer als 0 sein.*
use_epsilon	*Flag*	Nur Regressionsmodelle.*
epsilon	*number*	Der Wert muss größer als 0 sein.*
use_complexity_factor	*Flag*	*
complexity_factor	*number*	*
use_outlier_rate	*Flag*	Nur Ein-Klassen-Variante.*
outlier_rate	*number*	Nur Ein-Klassen-Variante. 0,0-1,0.*
weights	Data Equal Custom	
custom_weights	*strukturiert*	Strukturierte Eigenschaft im Format: set :orasvmnode.custom_weights = [[drugA 1] [drugB 2][drugC 3][drugX 4][drugY 5]]

* Eigenschaft wird ignoriert, wenn mode auf Simple gesetzt ist.

Verallgemeinerte lineare Modelle von Oracle

Für Knoten vom Typ oraglmnode sind folgende Eigenschaften verfügbar.

Tabelle 206. Eigenschaften von "oraglmnode"

Eigenschaften von oraglmnode	Werte	Eigenschaftsbeschreibung
normalization_method	zscore minmax none	
missing_value_handling	ReplaceWithMean UseCompleteRecords	
use_row_weights	*Flag*	*
row_weights_field	*Feld*	*
save_row_diagnostics	*Flag*	*
row_diagnostics_table	*string*	*
coefficient_confidence	*number*	*
use_reference_category	*Flag*	*
reference_category	*string*	*
ridge_regression	Auto Off On	*
parameter_value	*number*	*
vif_for_ridge	*Flag*	*

* Eigenschaft wird ignoriert, wenn mode auf Simple gesetzt ist.

Oracle Decision Tree

Für Knoten vom Typ oradecisiontreenode sind folgende Eigenschaften verfügbar.

Tabelle 207. Eigenschaften von "oradecisiontreenode"

Eigenschaften von oradecisiontreenode	Werte	Eigenschaftsbeschreibung
use_costs	*Flag*	
impurity_metric	Entropy Gini	
term_max_depth	*Integer*	2-20.*
term_minpct_node	*number*	0,0-10,0.*
term_minpct_split	*number*	0,0-20,0.*

Tabelle 207. Eigenschaften von "oradecisiontreenode" (Forts.)		
Eigenschaften von oradecisiontreenode	**Werte**	**Eigenschaftsbeschreibung**
term_minrec_node	*Integer*	Der Wert muss größer als 0 sein.*
term_minrec_split	*Integer*	Der Wert muss größer als 0 sein.*
display_rule_ids	*Flag*	*

* Eigenschaft wird ignoriert, wenn mode auf Simple gesetzt ist.

Oracle O-Cluster

Für Knoten vom Typ oraoclusternode sind folgende Eigenschaften verfügbar.

Tabelle 208. Eigenschaften von "oraoclusternode"		
Eigenschaften von oraoclusternode	**Werte**	**Eigenschaftsbeschreibung**
max_num_clusters	*Integer*	Der Wert muss größer als 0 sein.
max_buffer	*Integer*	Der Wert muss größer als 0 sein.*
sensitivity	*number*	0,0-1,0.*

* Eigenschaft wird ignoriert, wenn mode auf Simple gesetzt ist.

Oracle KMeans

Für Knoten vom Typ orakmeansnode sind folgende Eigenschaften verfügbar.

Tabelle 209. Eigenschaften von "orakmeansnode"		
Eigenschaften von orakmeansnode	**Werte**	**Eigenschaftsbeschreibung**
num_clusters	*Integer*	Der Wert muss größer als 0 sein.
normalization_method	zscore minmax none	
distance_function	Euclidean Cosine	
iterations	*Integer*	0-20.*
conv_tolerance	*number*	0,0-0,5.*
split_criterion	Variance Size	Die Standardeinstellung ist "Variance".
num_bins	*Integer*	Der Wert muss größer als 0 sein.*
block_growth	*Integer*	1-5.*

Tabelle 209. Eigenschaften von "orakmeansnode" (Forts.)		
Eigenschaften von orakmeans-node	**Werte**	**Eigenschaftsbeschreibung**
min_pct_attr_support	*number*	0,0-1,0.*

* Eigenschaft wird ignoriert, wenn mode auf Simple gesetzt ist.

Oracle NMF

Für Knoten vom Typ oranmfnode sind folgende Eigenschaften verfügbar.

Tabelle 210. Eigenschaften von "oranmfnode"		
Eigenschaften von oranmfnode	**Werte**	**Eigenschaftsbeschreibung**
normalization_method	minmax none	
use_num_features	*Flag*	*
num_features	*Integer*	0–1. Der Standardwert wird vom Algorithmus durch Schätzung aus den Daten ermittelt.*
random_seed	*number*	*
num_iterations	*Integer*	0-500.*
conv_tolerance	*number*	0,0-0,5.*
display_all_features	*Flag*	*

* Eigenschaft wird ignoriert, wenn mode auf Simple gesetzt ist.

Oracle Apriori

Für Knoten vom Typ oraapriorinode sind folgende Eigenschaften verfügbar.

Tabelle 211. Eigenschaften von "oraapriorinode"		
Eigenschaften von oraapriori-node	**Werte**	**Eigenschaftsbeschreibung**
content_field	*Feld*	
id_field	*Feld*	
max_rule_length	*Integer*	2-20.
min_confidence	*number*	0,0-1,0.
min_support	*number*	0,0-1,0.
use_transactional_data	*Flag*	

Oracle Minimum Description Length (MDL)

Für Knoten vom Typ oramdlnode sind keine speziellen Eigenschaften definiert. Informationen finden Sie unter den allgemeinen Oracle-Eigenschaften am Anfang dieses Abschnitts.

Oracle Attribute Importance (AI)

Für Knoten vom Typ oraainode sind folgende Eigenschaften verfügbar.

Tabelle 212. oraainode, Eigenschaften

Eigenschaften von oraainode	Werte	Eigenschaftsbeschreibung
custom_fields	*Flag*	Bei "true" können Sie Ziel-, Eingabe- und andere Felder für den aktuellen Knoten angeben. Bei "false" werden die aktuellen Einstellungen aus einem vorgeordneten Typknoten verwendet.
selection_mode	ImportanceLevel ImportanceValue TopN	
select_important	*Flag*	Wenn selection_mode auf ImportanceLevel gesetzt ist, wird hier angegeben, ob bedeutsame Felder ausgewählt werden sollen.
important_label	*string*	Gibt die Beschriftung für die Rangstufe "bedeutsam" an.
select_marginal	*Flag*	Wenn selection_mode auf ImportanceLevel gesetzt ist, wird hier angegeben, ob marginale Felder ausgewählt werden sollen.
marginal_label	*string*	Gibt die Beschriftung für die Rangstufe "marginal" an.
important_above	*number*	0,0-1,0.
select_unimportant	*Flag*	Wenn selection_mode auf ImportanceLevel gesetzt ist, wird hier angegeben, ob unbedeutende Felder ausgewählt werden sollen.
unimportant_label	*string*	Gibt die Beschriftung für die Rangstufe "unbedeutsam" an.
unimportant_below	*number*	0,0-1,0.
importance_value	*number*	Wenn selection_mode auf ImportanceValue gesetzt ist, wird hier der zu verwendende Trennwert angegeben. Zulässig sind Werte von 0 bis 100.
top_n	*number*	Wenn selection_mode auf TopN gesetzt ist, wird hier der zu verwendende Trennwert angegeben. Zulässig sind Werte von 0 bis 1000.

Eigenschaften von Oracle-Modellnuggets

Folgende Eigenschaften gelten für die Modellnuggets, die mithilfe der Oracle-Modelle erstellt wurden.

Oracle Naive Bayes

Für Knoten vom Typ applyoranbnode sind keine speziellen Eigenschaften definiert.

Oracle Adaptive Bayes

Für Knoten vom Typ `applyoraabnnode` sind keine speziellen Eigenschaften definiert.

Oracle Support Vector Machines

Für Knoten vom Typ `applyorasvmnode` sind keine speziellen Eigenschaften definiert.

Oracle Decision Tree

Die folgenden Eigenschaften sind für Knoten des Typs `applyoradecisiontreenode` verfügbar.

Tabelle 213. Eigenschaften von "applyoradecisiontreenode"

Eigenschaften von `applyoradecisiontreenode`	**Werte**	**Eigenschaftsbeschreibung**
`use_costs`	*Flag*	
`display_rule_ids`	*Flag*	

Oracle O-Cluster

Für Knoten vom Typ `applyoraoclusternode` sind keine speziellen Eigenschaften definiert.

Oracle KMeans

Für Knoten vom Typ `applyorakmeansnode` sind keine speziellen Eigenschaften definiert.

Oracle NMF

Für Knoten vom Typ `applyoranmfnode` ist die folgende Eigenschaft verfügbar.

Tabelle 214. Eigenschaften von "applyoranmfnode"

Eigenschaften von `applyoranmfnode`	**Werte**	**Eigenschaftsbeschreibung**
`display_all_features`	*Flag*	

Oracle Apriori

Dieses Modellnugget kann nicht in Scripts verwendet werden.

Oracle MDL

Dieses Modellnugget kann nicht in Scripts verwendet werden.

Knoteneigenschaften für IBM Netezza Analytics-Modellierung

Eigenschaften von Netezza-Modellierungsknoten

Folgende Eigenschaften haben alle IBM Netezza-Datenbankmodellierungsknoten gemeinsam.

Tabelle 215. Allgemeine Eigenschaften von Netezza-Knoten

Allgemeine Eigenschaften von Netezza-Knoten	**Werte**	**Eigenschaftsbeschreibung**
custom_fields	*Flag*	Bei "true" können Sie Ziel-, Eingabe- und andere Felder für den aktuellen Knoten angeben. Bei "false" werden die aktuellen Einstellungen aus einem vorgeordneten Typknoten verwendet.
inputs	*[field1 ... Feldn]*	Im Modell verwendete Eingabe- bzw. Prädiktorfelder.
target	*Feld*	Zielfeld (stetig oder kategorial).
record_id	*Feld*	Das als eindeutige ID für einen Datensatz zu verwendende Feld.
use_upstream_connection	*Flag*	Falls "True" (Standard), die in einem vorausgehenden Knoten angegebenen Verbindungsdetails. Wird bei Angabe von move_data_to_connection nicht verwendet.
move_data_connection	*Flag*	Falls "True", wird der Wert in die durch connection angegebene Datenbank verschoben. Wird bei Angabe von use_upstream_connection nicht verwendet.
connection	*strukturiert*	Die Verbindungszeichenfolge für die Netezza-Datenbank, in der das Modell gespeichert ist. Strukturierte Eigenschaft im Format: ['odbc' '<dsn>' '<username>' '<psw>' '<catname>' '<conn_attribs>' [true|false]] Dabei gilt Folgendes: <dsn> ist der Datenquellenname <username> und <psw> sind der Benutzername und das Kennwort für die Datenbank <catname> ist der Katalogname <conn_attribs> sind die Verbindungsattribute true | false gibt an, ob das Kennwort erforderlich ist.
table_name	*string*	Der Name der Datenbanktabelle, in der das Modell gespeichert werden soll.
use_model_name	*Flag*	Bei "True" wird der von model_name angegebene Name als Name des Modells verwendet. Andernfalls wird der Modellname vom System erstellt.
model_name	*string*	Benutzerdefinierter Name für neues Modell.

Tabelle 215. Allgemeine Eigenschaften von Netezza-Knoten (Forts.)		
Allgemeine Eigenschaften von Netezza-Knoten	**Werte**	**Eigenschaftsbeschreibung**
include_input_fields	*Flag*	Bei "True" werden alle Eingabefelder nach unten weitergegeben. Andernfalls werden nur record_id und vom Modell erstellte Felder weitergegeben.

Netezza-Entscheidungsbaum

Die folgenden Eigenschaften sind für Knoten des Typs netezzadectreenode verfügbar.

Tabelle 216. Eigenschaften von "netezzadectreenode"		
Eigenschaften von netezzadectreenode	**Werte**	**Eigenschaftsbeschreibung**
impurity_measure	Entropy Gini	Das Maß der Unreinheit, das verwendet wird, um die beste Position für eine Baumteilung zu ermitteln.
max_tree_depth	*Integer*	Maximale Anzahl der Ebenen, auf die der Baum erweitert werden kann. Der Standardwert ist 62 (größter zulässiger Wert).
min_improvement_splits	*number*	Mindestverbesserung in Unreinheit, damit eine Aufteilung stattfinden kann. Der Standardwert ist 0,01.
min_instances_split	*Integer*	Mindestanzahl der nicht aufgeteilten Datensätze, die verbleiben müssen, bevor eine Aufteilung stattfinden kann. Der Standardwert ist 2 (kleinster zulässiger Wert).
weights	*strukturiert*	Relative Gewichtungen für Klassen. Strukturierte Eigenschaft im Format: set :netezza_dectree.weights = [[drugA 0.3][drugB 0.6]] Standargewichtung ist für alle Klassen 1.
pruning_measure	Acc wAcc	Die Standardeinstellung ist Acc (Genauigkeit). Bei der alternativen Einstellung wAcc (gewichtete Genauigkeit) werden Klassengewichtungen in die Reduzierung/Beschneidung mit einbezogen.

Tabelle 216. Eigenschaften von "netezzadectreenode" (Forts.)		
Eigenschaften von netezza-dectreenode	**Werte**	**Eigenschaftsbeschreibung**
prune_tree_options	allTrainingData partitionTrainingData useOtherTable	In der Standardeinstellung wird allTrainingData zur Schätzung der Modellgenauigkeit verwendet. Verwenden Sie partitionTrainingData, um den Prozentsatz der zu verwendenden Trainingsdaten festzulegen, oder useOtherTable, um ein Trainingsdataset aus einer angegebenen Datenbanktabelle zu verwenden.
perc_training_data	*number*	Wenn prune_tree_options auf partitionTrainingData gesetzt ist, wird der für das Training zu verwendende Prozentsatz angegeben.
prune_seed	*Integer*	Für die Reproduktion der Analyseergebnisse zu verwendender Zufallsstartwert, wenn prune_tree_options auf partitionTrainingData gesetzt ist. Der Standardwert ist 1.
pruning_table	*string*	Tabellenname eines separaten Datasets für die Reduzierung, anhand dessen die Modellgenauigkeit geschätzt wird.
compute_probabilities	*Flag*	Wenn "True", wird zusätzlich zum Vorhersagefeld auch ein Feld für das Konfidenzniveau (Wahrscheinlichkeit) erstellt.

Netezza-K-Means

Die folgenden Eigenschaften sind für Knoten des Typs netezzakmeansnode verfügbar.

Tabelle 217. Eigenschaften von "netezzakmeansnode"		
Eigenschaften von netezzak-meansnode	**Werte**	**Eigenschaftsbeschreibung**
distance_measure	Euclidean Manhattan Canberra maximum	Methode zur Messung des Abstands zwischen Datenpunkten.
num_clusters	*Integer*	Anzahl der zu erstellenden Cluster; Standardwert ist 3.

Tabelle 217. Eigenschaften von "netezzakmeansnode" (Forts.)		
Eigenschaften von netezzak-meansnode	**Werte**	**Eigenschaftsbeschreibung**
max_iterations	*Integer*	Anzahl der Algorithmusiterationen, nach der das Modelltrainig beendet werden soll; Standardwert ist 5.
rand_seed	*Integer*	Für die Reproduktion der Analyseergebnisse zu verwendender Zufallsstartwert; Standardwert ist 12345.

Netezza-Bayes-Netz

Die folgenden Eigenschaften sind für Knoten des Typs netezzabayesnode verfügbar.

Tabelle 218. Eigenschaften von "netezzabayesnode"		
Eigenschaften von netezza-bayesnode	**Werte**	**Eigenschaftsbeschreibung**
base_index	*Integer*	Die numerische Kennung, die zur internen Verwaltung dem ersten Eingabefeld zugewiesen wird; Standardwert ist 777.
sample_size	*Integer*	Umfang der zu ziehenden Stichprobe, wenn die Anzahl der Attribute sehr groß ist; Standardwert ist 10.000.
display_additional_in-formation	*Flag*	Wenn "True", werden weitere Fortschrittsinformationen in einem Nachrichtendialogfeld angezeigt.
type_of_prediction	best neighbors nn-neighbors	Typ des zu verwendenden Vorhersagealgorithmus: best (Nachbar mit höchster Korrelation), neighbors (gewichtete Vorhersage von Nachbarn) oder nn-neighbors (Nicht-NULL-Nachbarn).

Netezza - Naive Bayes

Die folgenden Eigenschaften sind für Knoten des Typs netezzanaivebayesnode verfügbar.

Tabelle 219. Eigenschaften von "netezzanaivebayesnode"		
Eigenschaften von netezza-naivebayesnode	**Werte**	**Eigenschaftsbeschreibung**
compute_probabilities	*Flag*	Wenn "True", wird zusätzlich zum Vorhersagefeld auch ein Feld für das Konfidenzniveau (Wahrscheinlichkeit) erstellt.
use_m_estimation	*Flag*	Wenn "True", wird das m-Schätzverfahren zur Vermeidung der Wahrscheinlichkeit null während der Schätzung verwendet.

Netezza-KNN

Die folgenden Eigenschaften sind für Knoten des Typs netezzaknnnode verfügbar.

Tabelle 220. Eigenschaften von "netezzaknnnode"		
Eigenschaften von netez-zaknnnode	**Werte**	**Eigenschaftsbeschreibung**
weights	*strukturiert*	Strukturierte Eigenschaft, die zur Zuweisung von Gewichtungen für die einzelnen Klassen verwendet wird. Beispiel: `set :netezzaknnnode.weights = [[drugA 0.3][drugB 0.6]]`
distance_measure	Euclidean Manhattan Canberra Maximum	Methode zur Messung des Abstands zwischen Datenpunkten.
num_nearest_neighbors	*Integer*	Anzahl der nächsten Nachbarn für einen bestimmten Fall; Standardwert ist 3
standardize_measurements	*Flag*	Wenn "True", werden vor der Berechnung der Abstandwerte die Messungen für stetige Eingabefelder standardisiert.
use_coresets	*Flag*	Wenn "True", wird Stichprobennahme mit Core-Sets verwendet, um die Berechnung bei großen Datasets zu beschleunigen.

Netezza - Divisives Clustering

Die folgenden Eigenschaften sind für Knoten des Typs netezzadivclusternode verfügbar.

Tabelle 221. Eigenschaften von "netezzadivclusternode"		
Eigenschaften von netezza-divclusternode	**Werte**	**Eigenschaftsbeschreibung**
distance_measure	Euclidean Manhattan Canberra Maximum	Methode zur Messung des Abstands zwischen Datenpunkten.
max_iterations	*Integer*	Maximale Anzahl an Algorithmusiterationen, die durchgeführt werden sollen, bevor das Modelltraining beendet wird; Standardwert ist 5.
max_tree_depth	*Integer*	Maximale Anzahl an Ebenen, in die das Dataset unterteilt werden kann; Standardwert ist 3.
rand_seed	*Integer*	Für die Reproduktion von Analysen verwendeter Zufallsstartwert; Standardwert ist 12345.
min_instances_split	*Integer*	Mindestanzahl von Datensätzen, die aufgeteilt werden können; Standardwert ist 5.

Tabelle 221. Eigenschaften von "netezzadivclusternode" (Forts.)		
Eigenschaften von netezza-divclusternode	**Werte**	**Eigenschaftsbeschreibung**
level	*Integer*	Hierarchieebene, auf der die Datensätze gescort werden; Standardwert ist -1.

Netezza-PCA

Die folgenden Eigenschaften sind für Knoten des Typs netezzapcanode verfügbar.

Tabelle 222. Eigenschaften von "netezzapcanode"		
Eigenschaften von netezzap-canode	**Werte**	**Eigenschaftsbeschreibung**
center_data	*Flag*	Wenn "True" (Standard), wird vor der Analyse Datenzentrierung (auch als Mittelwertsubtraktion bezeichnet) durchgeführt.
perform_data_scaling	*Flag*	Wenn "True", wird vor der Analyse eine Datenskalierung durchgeführt. Auf diese Weise wird die Analyse eventuell weniger arbiträr, wenn verschiedene Variablen in verschiedenen Einheiten gemessen werden.
force_eigensolve	*Flag*	Wenn "True", wird eine weniger genaue, jedoch schnellere Methode zur Ermittlung der Hauptkomponenten verwendet.
pc_number	*Integer*	Anzahl an Hauptkomponenten, auf die das Dataset reduziert werden soll; Standardwert ist 1.

Netezza-Regressionsbaum

Die folgenden Eigenschaften sind für Knoten des Typs netezzaregtreenode verfügbar.

Tabelle 223. Eigenschaften von "netezzaregtreenod"		
Eigenschaften von netezza-regtreenode	**Werte**	**Eigenschaftsbeschreibung**
max_tree_depth	*Integer*	Maximale Anzahl an Ebenen, auf die ein Baum unterhalb des Stammknotens erweitert werden kann; Standardwert ist 10.
split_evaluation_measure	Variance	Unreinheitsmaß für die Klasse, das verwendet wird, um die beste Position für eine Baumteilung zu ermitteln; Standardwert (und einzige derzeit mögliche Option) ist Variance.
min_improvement_splits	*number*	Mindestwert der Unreinheitsreduzierung, bevor eine neue Aufteilung des Baums erfolgt.
min_instances_split	*Integer*	Mindestanzahl an Datensätzen, die aufgeteilt werden kann.

Tabelle 223. Eigenschaften von "netezzaregtreenod" (Forts.)		
Eigenschaften von netezza-regtreenode	**Werte**	**Eigenschaftsbeschreibung**
pruning_measure	mse r2 pearson spearman	Für die Reduzierung zu verwendende Methode.
prune_tree_options	allTrainingData partitionTrainingData useOtherTable	In der Standardeinstellung wird allTrainingData zur Schätzung der Modellgenauigkeit verwendet. Verwenden Sie partitionTrainingData, um den Prozentsatz der zu verwendenden Trainingsdaten festzulegen, oder useOtherTable, um ein Trainingsdataset aus einer angegebenen Datenbanktabelle zu verwenden.
perc_training_data	*number*	Wenn prune_tree_options auf PercTrainingData gesetzt ist, wird der für das Training zu verwendende Prozentsatz angegeben.
prune_seed	*Integer*	Für die Reproduktion der Analyseergebnisse zu verwendender Zufallsstartwert, wenn prune_tree_options auf PercTrainingData gesetzt ist. Der Standardwert ist 1.
pruning_table	*string*	Tabellenname eines separaten Datasets für die Reduzierung, anhand dessen die Modellgenauigkeit geschätzt wird.
compute_probabilities	*Flag*	Wenn "True", wird angegeben, ob die Varianz zugewiesener Klassen in die Ausgabe aufgenommen werden soll.

Netezza - Lineare Regression

Die folgenden Eigenschaften sind für Knoten des Typs netezzalineregressionnode verfügbar.

Tabelle 224. Eigenschaften von "netezzalineregressionnode"		
Eigenschaften von netezza-lineregressionnode	**Werte**	**Eigenschaftsbeschreibung**
use_svd	*Flag*	Wenn "True", wird anstelle der ursprünglichen Matrix die Matrix zur Einzelwertzerlegung verwendet, um eine höhere Geschwindigkeit und numerische Genauigkeit zu erreichen.
include_intercept	*Flag*	Wenn "True" (Standard), wird die Gesamtgenauigkeit der Lösung erhöht.
calculate_model_diagnostics	*Flag*	Wenn "True", werden Diagnosedaten für das Modell berechnet.

Netezza-Zeitreihe

Die folgenden Eigenschaften sind für Knoten des Typs netezzatimeseriesnode verfügbar.

Tabelle 225. Eigenschaften von "netezzatimeseriesnode"		
Eigenschaften von netezzati-meseriesnode	**Werte**	**Eigenschaftsbeschreibung**
time_points	*Feld*	Das Eingabefeld, das die Datums- bzw. Zeitwerte für die Zeitreihe enthält.
time_series_ids	*Feld*	Eingabefeld mit Zeitreihen-IDs. Verwenden Sie das Feld, wenn die Eingabe mehrere Zeitreihen enthält.
model_table	*Feld*	Der Name der Datenbanktabelle, in der das Netezza-Zeitreihenmodell gespeichert werden soll.
description_table	*Feld*	Name der Eingabetabelle mit Zeitreihennamen und Beschreibungen.
seasonal_adjustment_table	*Feld*	Name der Ausgabetabelle, in der saisonal angepasste Werte gespeichert werden, die durch exponentielles Glätten oder Algorithmen zur saisonalen Zerlegung in Trends berechnet werden.

Tabelle 225. Eigenschaften von "netezzatimeseriesnode" (Forts.)

Eigenschaften von netezzatimeseriesnode	**Werte**	**Eigenschaftsbeschreibung**
algorithm_name	SpectralAnalysis oder spectral ExponentialSmoothing oder esmoothing ARIMA SeasonalTrendDecomposition oder std	Für die Modellierung von Zeitreihen zu verwendender Algorithmus.
trend_name	N A DA M DM	Trendtyp für exponentielles Glätten: N - keiner A - additiv DA - gedämpft additiv M - multiplikativ DM - gedämpft multiplikativ
seasonality_type	N A M	Saisonalitätstyp für exponentielles Glätten: N - keiner A - additiv M - multiplikativ
interpolation_method	linear cubicspline exponentialspline	Zu verwendende Interpolationsmethode.

Tabelle 225. Eigenschaften von "netezzatimeseriesnode" (Forts.)		
Eigenschaften von `netezzatimeseriesnode`	**Werte**	**Eigenschaftsbeschreibung**
`timerange_setting`	SD SP	Einstellung für den zu verwendenden Zeitbereich: SD - systembestimmt (verwendet den vollständigen Bereich der Zeitreihendaten) SP - benutzerdefiniert über `earliest_time` und `latest_time`

`earliest_time`
 	
`latest_time`
	Integer
	
	Datum
	
	Zeit
	
	Zeitmarke
		Start- und Endwerte, wenn `timerange_setting` auf SP gesetzt ist.
		
		Format wie beim `time_points`-Wert.
		
		Wenn das Feld `time_points` beispielsweise ein Datum enthält, sollte hier auch ein Datum enthalten sein.
		
		Beispiel:
		
		`set NZ_DT1.timerange_setting = 'SP'`
		
		`set NZ_DT1.earliest_time = '1921-01-01'`
		
		`set NZ_DT1.latest_time = '2121-01-01'`

Tabelle 225. Eigenschaften von "netezzatimeseriesnode" (Forts.)

Eigenschaften von netezzatimeseriesnode	**Werte**	**Eigenschaftsbeschreibung**

arima_setting	SD
	
	SP
		Einstellung für den ARIMA-Algorithmus (nur verwendet, wenn algorithm_name auf ARIMA gesetzt ist):
		
		SD - systembestimmt
		
		SP - benutzerdefiniert
		
		Verwenden Sie bei arima_setting = SP die folgenden Parameter, um die saisonalen und nicht saisonalen Werte festzulegen. Beispiel (nur nicht saisonal):
		
		`set NZ_DT1.algorithm_name = 'arima'`
		
		`set NZ_DT1.arima_setting = 'SP'`
		
		`set NZ_DT1.p_symbol = 'lesseq'`
		
		`set NZ_DT1.p = '4'`
		
		`set NZ_DT1.d_symbol = 'lesseq'`
		
		`set NZ_DT1.d = '2'`
		
		`set NZ_DT1.q_symbol = 'lesseq'`
		
		`set NZ_DT1.q = '4'`

p_symbol d_symbol q_symbol sp_symbol sd_symbol sq_symbol	less eq lesseq	ARIMA - Operator für die Parameter p, d, q, sp, sd und sq: less - kleiner als eq - gleich lesseq - kleiner-gleich
p	*Integer*	ARIMA - nicht saisonale Autokorrelationsmaße.

Tabelle 225. Eigenschaften von "netezzatimeseriesnode" (Forts.)		
Eigenschaften von `netezzatimeseriesnode`	**Werte**	**Eigenschaftsbeschreibung**
q	*Integer*	ARIMA - nicht saisonaler Ableitungswert.
d	*Integer*	ARIMA - nicht saisonale Anzahl von Ordnungen des gleitenden Durchschnitts im Modell.
sp	*Integer*	ARIMA - saisonale Autokorrelationsmaße.
sq	*Integer*	ARIMA - saisonaler Ableitungswert.
sd	*Integer*	ARIMA - saisonale Anzahl von Ordnungen des gleitenden Durchschnitts im Modell.

`advanced_setting`	SD
	
	SP
		Legt fest, wie erweiterte Einstellungen behandelt werden:
		
		SD - systembestimmt
		
		SP - benutzerdefiniert über `period`, `units_period` und `forecast_setting`.
		
		Beispiel:
		
		`set NZ_DT1.advanced_setting = 'SP'`
		
		`set NZ_DT1.period = 5`
		
		`set NZ_DT1.units_period = 'd'`

`period`	*Integer*	Länge des saisonalen Zyklus, die in Verbindung mit `units_period` angegeben wird. Wird nicht für Spektralanalyse verwendet.

Tabelle 225. Eigenschaften von "netezzatimeseriesnode" (Forts.)		
Eigenschaften von netezzatimeseriesnode	**Werte**	**Eigenschaftsbeschreibung**
units_period	ms s min h d wk q y	Einheiten für period: ms - Millisekunden s - Sekunden min - Minuten h - Stunden d - Tage wk - Wochen q - Quartale y - Jahre Beispiel: Verwenden Sie für einen wöchentliche Zeitreihe 1 für period und wk für units_period.
forecast_setting	forecasthorizon forecasttimes	Gibt an, wie Vorhersagen gemacht werden.
forecast_horizon	*Integer* *Datum* *Zeit* *Zeitmarke*	Wenn forecast_setting = forecasthorizon angegeben ist, wird ein Endpunktwert für die Vorhersage angegeben. Format wie beim time_points-Wert. Wenn das Feld time_points beispielsweise ein Datum enthält, sollte hier auch ein Datum enthalten sein.

Tabelle 225. Eigenschaften von "netezzatimeseriesnode" (Forts.)		
Eigenschaften von netezzatimeseriesnode	**Werte**	**Eigenschaftsbeschreibung**
forecast_times	*Integer* *Datum* *Zeit* *Zeitmarke*	Wenn forecast_setting = forecasttimes angegeben ist, werden die für die Vorhersagen zu verwendenden Werte angegeben. Format wie beim time_points-Wert. Wenn das Feld time_points beispielsweise ein Datum enthält, sollte hier auch ein Datum enthalten sein.
include_history	*Flag*	Gibt an, ob historische Werte bei der Ausgabe berücksichtigt werden sollen.
include_interpolated_values	*Flag*	Gibt an, ob interpolierte Werte bei der Ausgabe berücksichtigt werden sollen. Wird nicht verwendet, wenn include_history auf false gesetzt ist.

Verallgemeinertes lineares Netezza-Modell

Die folgenden Eigenschaften sind für Knoten des Typs netezzaglmnode verfügbar.

Tabelle 226. Eigenschaften von "netezzaglmnode"		
Eigenschaften von netezzaglmnode	**Werte**	**Eigenschaftsbeschreibung**
dist_family	bernoulli gaussian poisson negativebinomial wald gamma	Verteilungstyp. Die Standardeinstellung ist bernoulli.
dist_params	*number*	Zu verwendender Wert für Verteilungsparameter. Wird nur verwendet, wenn distribution auf Negativebinomial gesetzt ist.

Tabelle 226. Eigenschaften von "netezzaglmnode" (Forts.)

Eigenschaften von netezzaglmnode	**Werte**	**Eigenschaftsbeschreibung**
trials	*Integer*	Wird nur verwendet, wenn distribution auf Binomial gesetzt ist. Wenn es sich bei der Zielantwort um eine Reihe von Ereignissen handelt, die während Tests auftreten, enthält das Feld target die Anzahl der Ereignisse und das Feld trials die Anzahl der Tests.
model_table	*Feld*	Der Name der Datenbanktabelle, in der das verallgemeinerte lineare Netezza-Modell gespeichert werden soll.
maxit	*Integer*	Die maximale Anzahl der Iterationen, die im Algorithmus vorgenommen werden sollen. Der Standardwert ist 20.
eps	*number*	Der maximale Fehlerwert (in wissenschaftlicher Notation), bei dem der Algorithmus die Suche nach dem am besten passenden Modell beenden soll. Der Standardwert ist -3, d. h. 1E-3 bzw. 0,001.
tol	*number*	Der Wert (in wissenschaftlicher Notation), unterhalb dessen Fehler so behandelt werden, als hätten sie den Wert 0. Der Standardwert ist -7, es werden also Fehlerwerte unter 1E-7 (bzw. 0,0000001) als nicht signifikant gewertet.

Tabelle 226. Eigenschaften von "netezzaglmnode" (Forts.)

Eigenschaften von netez-zaglmnode	**Werte**	**Eigenschaftsbeschreibung**
link_func	identity inverse invnegative invsquare sqrt power oddspower log clog loglog cloglog logit probit gaussit cauchit canbinom cangeom cannegbinom	Zu verwendende Verknüpfungs-funktion. Die Standardeinstellung ist logit.
link_params	*number*	Für die Verknüpfungsfunktion zu verwendender Parameterwert. Wird nur verwendet, wenn link_function auf power oder oddspower gesetzt ist.

Tabelle 226. Eigenschaften von "netezzaglmnode" (Forts.)		
Eigenschaften von netezzaglmnode	**Werte**	**Eigenschaftsbeschreibung**
interaction	[[[*colnames1*],[*levels1*]], [[*colnames2*],[*levels2*]], ...,[[*colnamesN*],[*levelsN*]],]	Gibt die Interaktionen zwischen Feldern an. *Spaltennamen* ist eine Liste von Eingabefeldern und *Niveau* ist für jedes Feld immer 0. Beispiel: `[[["K","BP","Sex","K"], [0,0,0,0]], [["Age","Na"],[0,0]]]`
intercept	*Flag*	Wenn true gesetzt ist, wird konstanter Term in das Modell einbezogen.

Eigenschaften von Netezza-Modellnuggets

Folgende Eigenschaften haben alle Modellnuggets von Netezza-Datenbanken gemeinsam.

Tabelle 227. Allgemeine Eigenschaften von Netezza-Modellnuggets		
Allgemeine Eigenschaften von Netezza-Modellnuggets	**Werte**	**Eigenschaftsbeschreibung**
connection	*string*	Die Verbindungszeichenfolge für die Netezza-Datenbank, in der das Modell gespeichert ist.
table_name	*string*	Der Name der Datenbanktabelle, in der das Modell gespeichert werden soll.

Die anderen Eigenschaften des Modellnuggets stimmen mit denen für den zugehörigen Modellierungsknoten überein.

Die Scriptnamen des Modellnuggets lauten wie folgt.

Tabelle 228. Scriptnamen von Netezza-Modellnuggets	
Modellnugget	**Scriptname**
Entscheidungsbaum	applynetezzadectreenode
K-Means	applynetezzakmeansnode
Bayes-Netz	applynetezzabayesnode
Naive Bayes	applynetezzanaivebayesnode
KNN	applynetezzaknnnode
Divisives Clustering	applynetezzadivclusternode
PCA	applynetezzapcanode
Regressionsbaum	applynetezzaregtreenode
lineare Regression	applynetezzalineregressionnode
Zeitreihe	applynetezzatimeseriesnode

Tabelle 228. Scriptnamen von Netezza-Modellnuggets (Forts.)

Modellnugget	**Scriptname**
Verallgemeinert linear	`applynetezzaglmnode`

Kapitel 16. Eigenschaften von Ausgabeknoten

Die Eigenschaften von Ausgabeknoten unterscheiden sich von denen anderer Knotentypen. Statt auf eine bestimmte Knotenoption zu verweisen, speichern Ausgabeknoteneigenschaften eine Referenz zum Ausgabeobjekt. Dies ist nützlich, wenn ein Wert aus einer Tabelle als Streamparameter festgelegt wird.

In diesem Abschnitt werden die für Ausgabeknoten verfügbaren Scripteigenschaften beschrieben.

analysisnode, Eigenschaften

Der Analyseknoten evaluiert die Fähigkeit von Vorhersagemodellen, genaue Vorhersagen zu generieren. Mit Analyseknoten werden verschiedene Vergleiche zwischen den vorhergesagten Werten und den tatsächlichen Werten für ein oder mehrere Modellnuggets angestellt. Sie können auch Vorhersagemodelle miteinander vergleichen.

Beispiel

```
node = stream.create("analysis", "My node")
# "Analysis" tab
node.setPropertyValue("coincidence", True)
node.setPropertyValue("performance", True)
node.setPropertyValue("confidence", True)
node.setPropertyValue("threshold", 75)
node.setPropertyValue("improve_accuracy", 3)
node.setPropertyValue("inc_user_measure", True)
# "Define User Measure..."
node.setPropertyValue("user_if", "@TARGET = @PREDICTED")
node.setPropertyValue("user_then", "101")
node.setPropertyValue("user_else", "1")
node.setPropertyValue("user_compute", ["Mean", "Sum"])
node.setPropertyValue("by_fields", ["Drug"])
# "Output" tab
node.setPropertyValue("output_format", "HTML")
node.setPropertyValue("full_filename", "C:/output/analysis_out.html")
```

Tabelle 229. analysisnode, Eigenschaften

Eigenschaften von analysisnode	**Datentyp**	**Eigenschaftsbeschreibung**
output_mode	Screen File	Dient zur Angabe des Zielorts für die vom Ausgabeknoten erstellte Ausgabe.
use_output_name	*Flag*	Gibt an, ob ein benutzerdefinierter Ausgabename verwendet wird.
output_name	*string*	Wenn use_output_name "true" ist, gibt diese Eigenschaft den zu verwendenden Namen an.

Tabelle 229. analysisnode, Eigenschaften (Forts.)		
Eigenschaften von analysisnode	**Datentyp**	**Eigenschaftsbeschreibung**
output_format	Text (*.txt*) HTML (*.html*) Output (*.cou*)	Dient zur Angabe des Ausgabetyps.
by_fields	*Liste*	
full_filename	*string*	Bei Datenträger-, Daten- oder HTML-Ausgabe gibt diese Eigenschaft den Namen der Ausgabedatei an.
coincidence	*Flag*	
performance	*Flag*	
evaluation_binary	*Flag*	
confidence	*Flag*	
threshold	*number*	
improve_accuracy	*number*	
field_detection_method	Metadata Name	Bestimmt, wie vorhergesagte Felder dem ursprünglichen Zielfeld zugeordnet werden sollen. Geben Sie Metadata oder Name an.
inc_user_measure	*Flag*	
user_if	*Ausdruck*	
user_then	*Ausdruck*	
user_else	*Ausdruck*	
user_compute	[Mean Sum Min Max SDev]	

dataauditnode, Eigenschaften

Der Data Audit-Knoten bietet einen umfassenden ersten Einblick in die Daten mit statistischen Funktionen, Histogrammen und der Verteilung für die einzelnen Felder sowie Informationen zu Ausreißern, fehlenden Werten und Extremwerten. Ergebnisse werden in einer leicht lesbaren Matrix angezeigt, die sortiert werden kann und mit deren Hilfe Grafik- und Datenaufbereitungsknoten in vollständigem Umfang generiert werden können.

Beispiel

```
filenode = stream.createAt("variablefile", "File", 100, 100)
filenode.setPropertyValue("full_filename", "$CLEO_DEMOS/DRUG1n")
node = stream.createAt("dataaudit", "My node", 196, 100)
stream.link(filenode, node)
node.setPropertyValue("custom_fields", True)
node.setPropertyValue("fields", ["Age", "Na", "K"])
```

```
node.setPropertyValue("display_graphs", True)
node.setPropertyValue("basic_stats", True)
node.setPropertyValue("advanced_stats", True)
node.setPropertyValue("median_stats", False)
node.setPropertyValue("calculate", ["Count", "Breakdown"])
node.setPropertyValue("outlier_detection_method", "std")
node.setPropertyValue("outlier_detection_std_outlier", 1.0)
node.setPropertyValue("outlier_detection_std_extreme", 3.0)
node.setPropertyValue("output_mode", "Screen")
```

Tabelle 230. dataauditnode, Eigenschaften

Eigenschaften von dataauditnode	**Datentyp**	**Eigenschaftsbeschreibung**
custom_fields	*Flag*	
fields	*[Feld1 ... FeldN]*	
overlay	*Feld*	
display_graphs	*Flag*	Dient zur Aktivierung bzw. Inaktivierung der Anzeige von Diagrammen in der Ausgabematrix.
basic_stats	*Flag*	
advanced_stats	*Flag*	
median_stats	*Flag*	
calculate	Count Breakdown	Dient zur Berechnung fehlender Werte. Sie können eine der beiden Berechnungsmethoden, beide Methoden oder auch keine der Methoden auswählen.
outlier_detection_method	std iqr	Dient zur Angabe der Erkennungsmethode für Ausreißer und Extremwerte.
outlier_detection_std_outlier	*number*	Wenn für outlier_detection_method die Option std verwendet wird, wird die Zahl angegeben, die für die Definition der Ausreißer verwendet werden soll.
outlier_detection_std_extreme	*number*	Wenn für outlier_detection_method die Option std verwendet wird, wird die Zahl angegeben, die für die Definition der Extremwerte verwendet werden soll.
outlier_detection_iqr_outlier	*number*	Wenn für outlier_detection_method die Option iqr verwendet wird, wird die Zahl angegeben, die für die Definition der Ausreißer verwendet werden soll.

Tabelle 230. dataauditnode, Eigenschaften (Forts.)		
Eigenschaften von dataauditnode	**Datentyp**	**Eigenschaftsbeschreibung**
outlier_detection_iqr_extreme	*number*	Wenn für outlier_detection_method die Option iqr verwendet wird, wird die Zahl angegeben, die für die Definition der Extremwerte verwendet werden soll.
use_output_name	*Flag*	Gibt an, ob ein benutzerdefinierter Ausgabename verwendet wird.
output_name	*string*	Wenn use_output_name "true" ist, gibt diese Eigenschaft den zu verwendenden Namen an.
output_mode	Screen File	Dient zur Angabe des Zielorts für die vom Ausgabeknoten erstellte Ausgabe.
output_format	Formatted (*.tab*) Delimited (*.csv*) HTML (*.html*) Output (*.cou*)	Dient zur Angabe des Ausgabetyps.
paginate_output	*Flag*	Wenn output_format auf HTML gesetzt ist, wird hiermit die Ausgabe in Seiten unterteilt.
lines_per_page	*number*	Bei Verwendung mit paginate_output wird die Anzahl der Zeilen pro Ausgabeseite angegeben.
full_filename	*string*	

extensionoutputnode, Eigenschaften

Mit dem Erweiterungsausgabeknoten können Sie Daten und die Ergebnisse des Modellscorings mithilfe Ihres eigenen benutzerdefinierten Scripts in R oder Python for Spark analysieren. Die Ausgabe von der Analyse kann Text oder grafisch sein. Die Ausgabe wird der Registerkarte **Ausgabe** des Managerbereichs hinzugefügt. Alternativ kann die Ausgabe in eine Datei umgeleitet werden.

Beispiel für Python for Spark

```
#### script example for Python for Spark
import modeler.api
```

```
stream = modeler.script.stream()
node = stream.create("extension_output", "extension_output")
node.setPropertyValue("syntax_type", "Python")

python_script = """
import json
import spss.pyspark.runtime

cxt = spss.pyspark.runtime.getContext()
df = cxt.getSparkInputData()
schema = df.dtypes[:]
print df
"""

node.setPropertyValue("python_syntax", python_script)
```

Beispiel für R

```
#### script example for R
node.setPropertyValue("syntax_type", "R")
node.setPropertyValue("r_syntax", "print(modelerData$Age)")
```

Tabelle 231. extensionoutputnode, Eigenschaften

Eigenschaften von extensionoutputnode	Datentyp	Eigenschaftsbeschreibung
syntax_type	*R* *Python*	Gibt das Script an, das ausgeführt wird - R oder Python (R ist der Standardwert).
r_syntax	*string*	R-Scriptsyntax für das Modellscoring.
python_syntax	*string*	Python-Scriptsyntax für das Modellscoring.
convert_flags	`StringsAndDoubles` `LogicalValues`	Option zum Konvertieren von Flagfeldern.
convert_missing	*Flag*	Option zum Konvertieren fehlender Werte in den R-Wert NA .
convert_datetime	*Flag*	Option zum Konvertieren von Variablen mit Datums- oder Datums-/Zeitformaten in R-Datums-/Zeitformate.
convert_datetime_class	`POSIXct` `POSIXlt`	Optionen, die angeben, in welches Format Variablen mit Datums- oder Datums-/Zeitformaten konvertiert werden.
output_to	`Screen` `File`	Gibt den Ausgabetyp (`Screen` oder `File`) an.
output_type	`Graph` `Text`	Gibt an, ob eine grafische Ausgabe oder eine Textausgabe generiert wird.
full_filename	*string*	Dateiname, der für die generierte Ausgabe verwendet wird.
graph_file_type	`HTML` `COU`	Dateityp für die Ausgabedatei (`.html` oder `.cou`).

Tabelle 231. extensionoutputnode, Eigenschaften (Forts.)		
Eigenschaften von extensionoutputnode	**Datentyp**	**Eigenschaftsbeschreibung**
text_file_type	HTML TEXT COU	Gibt den Dateityp für die Textausgabe an (.html, .txt oder .cou).

kdeexport, Eigenschaften

Kernel Density Estimation (KDE) verwendet die Kugelbaum- oder KD-Baumalgorithmen für effiziente Abfragen und kombiniert Konzepte von unbeaufsichtigtem Lernen, Funktionsentwicklung und Datenmodellierung. Auf Nachbarn basierte Ansätze wie KDE sind einige der gängigsten und nützlichsten Dichteschätzungsverfahren. Die KDE-Modellierungs- und KDE-Simulationsknoten in SPSS Modeler stellen die zentralen Funktionen und häufig verwendeten KDE-Bibliotheksparametern bereit. Die Knoten sind in Python implementiert.

Tabelle 232. kdeexport, Eigenschaften		
Eigenschaften von kdeexport	**Datentyp**	**Eigenschaftsbeschreibung**
bandwidth	*DOUBLE*	Der Standardwert ist 1.
kernel	*string*	Der zu verwendende Kern: gaussian oder tophat. Der Standardwert ist gaussian.
algorithm	*string*	Der zu verwendende Baumalgorithmus: kd_tree, ball_tree oder auto. Der Standardwert ist auto.
metric	*string*	Die beim Berechnen des Abstands zu verwendende Metrik. Beim Algorithmus kd_tree können Sie aus folgenden Metriken wählen: Euclidean, Chebyshev, Cityblock, Minkowski, Manhattan, Infinity, P, L2 oder L1. Beim Algorithmus ball_tree können Sie aus folgenden Metriken wählen: Euclidian, Braycurtis, Chebyshev, Canberra, Cityblock, Dice, Hamming, Infinity, Jaccard, L1, L2, Minkowski, Matching, Manhattan, P, Rogersanimoto, Russellrao, Sokalmichener, Sokalsneath oder Kulsinski. Der Standardwert ist Euclidean.
atol	*Float*	Die gewünschte absolute Toleranz des Ergebnisses. Eine größere Toleranz führt in der Regel zu schnellerer Ausführung. Der Standardwert ist 0.0.
rtol	*Float*	Die gewünschte relative Toleranz des Ergebnisses. Eine größere Toleranz führt in der Regel zu schnellerer Ausführung. Der Standardwert ist 1E-8.

Tabelle 232. kdeexport, Eigenschaften (Forts.)		
Eigenschaften von kdeexport	**Datentyp**	**Eigenschaftsbeschreibung**
breadthFirst	*Boolesch*	Setzen Sie diese Eigenschaft auf True, um den Ansatz "Breite zuerst" zu verwenden. Setzen Sie diese Eigenschaft auf False, um den Ansatz "Tiefe zuerst" zu verwenden. Der Standardwert ist True.
LeafSize	*Integer*	Die Blattgröße der zugrunde liegenden Baums. Der Standardwert ist 40. Eine Änderung dieses Werts kann sich erheblich auf die Leistung auswirken.
pValue	*DOUBLE*	Geben Sie den P-Wert an, der verwendet werden soll, wenn Sie Minkowski als Metrik verwenden. Der Standardwert ist 1.5.

matrixnode, Eigenschaften

Der Matrixknoten erstellt eine Tabelle, die die Beziehungen zwischen den Feldern aufzeigt. Am häufigsten wird hiermit die Beziehung zwischen zwei symbolischen Feldern dargestellt, möglich ist aber auch die Darstellung von Beziehungen zwischen Flagfeldern oder numerischen Feldern.

Beispiel

```
node = stream.create("matrix", "My node")
# "Settings" tab
node.setPropertyValue("fields", "Numerics")
node.setPropertyValue("row", "K")
node.setPropertyValue("column", "Na")
node.setPropertyValue("cell_contents", "Function")
node.setPropertyValue("function_field", "Age")
node.setPropertyValue("function", "Sum")
# "Appearance" tab
node.setPropertyValue("sort_mode", "Ascending")
node.setPropertyValue("highlight_top", 1)
node.setPropertyValue("highlight_bottom", 5)
node.setPropertyValue("display", ["Counts", "Expected", "Residuals"])
node.setPropertyValue("include_totals", True)
# "Output" tab
node.setPropertyValue("full_filename", "C:/output/matrix_output.html")
node.setPropertyValue("output_format", "HTML")
node.setPropertyValue("paginate_output", True)
node.setPropertyValue("lines_per_page", 50)
```

Tabelle 233. matrixnode, Eigenschaften		
Eigenschaften von matrixnode	**Datentyp**	**Eigenschaftsbeschreibung**
fields	Selected Flags Numerics	
row	*Feld*	

Tabelle 233. matrixnode, Eigenschaften (Forts.)		
Eigenschaften von `matrixnode`	**Datentyp**	**Eigenschaftsbeschreibung**
`column`	*Feld*	
`include_missing_values`	*Flag*	Gibt an, ob benutzerdefiniert fehlende Werte (leer) und systemdefiniert fehlende Werte (null) in die Zeilen- und Spaltenausgabe eingeschlossen werden sollen.

`cell_contents`	`CrossTabs`
	
	`Function`	

`function_field`	*string*	

`function`	`Sum`
	
	`Mean`
	
	`Min`
	
	`Max`
	
	`SDev`	

`sort_mode`	`Unsorted`
	
	`Ascending`
	
	`Descending`	

`highlight_top`	*number*	Wenn ungleich 0, dann ist die Eigenschaft wahr.
`highlight_bottom`	*number*	Wenn ungleich 0, dann ist die Eigenschaft wahr.

`display`	`[Counts`
	
	`Expected`
	
	`Residuals`
	
	`RowPct`
	
	`ColumnPct`
	
	`TotalPct]`	

`include_totals`	*Flag*	
`use_output_name`	*Flag*	Gibt an, ob ein benutzerdefinierter Ausgabename verwendet wird.

Tabelle 233. matrixnode, Eigenschaften (Forts.)		
Eigenschaften von `matrixnode`	**Datentyp**	**Eigenschaftsbeschreibung**
`output_name`	*string*	Wenn `use_output_name` "true" ist, gibt diese Eigenschaft den zu verwendenden Namen an.

`output_mode`	`Screen`
	
	`File`
		Dient zur Angabe des Zielorts für die vom Ausgabeknoten erstellte Ausgabe.

`output_format`	`Formatted` (*.tab*) `Delimited` (*.csv*) `HTML` (*.html*) `Output` (*.cou*)	Dient zur Angabe des Ausgabetyps. Sowohl für das Format `Formatted` als auch für das Format `Delimited` kann der Modifikator `transposed` verwendet werden, der die Zeilen und Spalten in der Tabelle transponiert.
`paginate_output`	*Flag*	Wenn `output_format` auf `HTML` gesetzt ist, wird hiermit die Ausgabe in Seiten unterteilt.
`lines_per_page`	*number*	Bei Verwendung mit `paginate_output` wird die Anzahl der Zeilen pro Ausgabeseite angegeben.
`full_filename`	*string*	

meansnode, Eigenschaften

Der Mittelwertknoten vergleicht die Mittelwerte zwischen unabhängigen Gruppen oder zwischen Paaren von in Bezug stehenden Feldern, um zu testen, ob ein signifikanter Unterschied vorliegt. So können Sie beispielsweise die Einnahmen vor und nach der Durchführung einer Werbeaktion vergleichen oder die Einnahmen, die von Kunden stammen, die keine Werbezettel erhielten, mit den Einnahmen von Kunden vergleichen, die von der Werbeaktion erreicht wurden.

Beispiel

```
node = stream.create("means", "My node")
node.setPropertyValue("means_mode", "BetweenFields")
node.setPropertyValue("paired_fields", [["OPEN_BAL", "CURR_BAL"]])
node.setPropertyValue("label_correlations", True)
node.setPropertyValue("output_view", "Advanced")
node.setPropertyValue("output_mode", "File")
node.setPropertyValue("output_format", "HTML")
node.setPropertyValue("full_filename", "C:/output/means_output.html")
```

Tabelle 234. meansnode, Eigenschaften

Eigenschaften von meansnode	**Datentyp**	**Eigenschaftsbeschreibung**
means_mode	BetweenGroups BetweenFields	Gibt den Typ der Mittelwertstatistik an, die für die Daten ausgeführt werden soll.
test_fields	[field1 ... fieldn]	Gibt das Testfeld an, wenn means_mode auf BetweenGroups gesetzt ist.
grouping_field	*Feld*	Gibt das Gruppierungsfeld an.
paired_fields	[[field1 field2] [field3 field4] ...]	Gibt die zu verwendenden Feldpaare an, wenn means_mode auf BetweenFields gesetzt ist.
label_correlations	*Flag*	Gibt an, ob Korrelationsbeschriftungen in der Ausgabe angezeigt werden sollen. Diese Einstellung gilt nur, wenn means_mode auf BetweenFields gesetzt ist.
correlation_mode	Probability Absolute	Gibt an, ob die Korrelationen nach Wahrscheinlichkeit oder anhand des absoluten Werts beschriftet werden sollen.
weak_label	*string*	
medium_label	*string*	
strong_label	*string*	
weak_below_probability	*number*	Wenn correlation_mode auf Probability gesetzt ist, wird hier der Trennwert für schwache Korrelationen angegeben. Hierbei muss es sich um einen Wert zwischen 0 und 1 handeln, beispielsweise 0,90.
strong_above_probability	*number*	Trennwert für starke Korrelationen.
weak_below_absolute	*number*	Wenn correlation_mode auf Absolute gesetzt ist, wird hier der Trennwert für schwache Korrelationen angegeben. Hierbei muss es sich um einen Wert zwischen 0 und 1 handeln, beispielsweise 0,90.
strong_above_absolute	*number*	Trennwert für starke Korrelationen.
unimportant_label	*string*	
marginal_label	*string*	

Tabelle 234. meansnode, Eigenschaften (Forts.)		
Eigenschaften von meansnode	**Datentyp**	**Eigenschaftsbeschreibung**
important_label	*string*	
unimportant_below	*number*	Trennwert für niedrige Feldwichtigkeit. Hierbei muss es sich um einen Wert zwischen 0 und 1 handeln, beispielsweise 0,90.
important_above	*number*	
use_output_name	*Flag*	Gibt an, ob ein benutzerdefinierter Ausgabename verwendet wird.
output_name	*string*	Zu verwendender Name.
output_mode	Screen File	Gibt den Zielort für die vom Ausgabeknoten erstellte Ausgabe an.
output_format	Formatted (*.tab*) Delimited (*.csv*) HTML (*.html*) Output (*.cou*)	Gibt den Ausgabetyp an.
full_filename	*string*	
output_view	Simple Advanced	Gibt an, ob die einfache oder die erweiterte Ansicht in der Ausgabe angezeigt werden soll.

reportnode, Eigenschaften

Der Berichtknoten erstellt formatierte Berichte, die sowohl festen Text als auch Daten und andere aus den Daten abgeleitete Ausdrücke enthalten. Das Format des Berichts wird mithilfe von Textvorlagen festgelegt, mit denen der feste Text und die Datenausgabekonstruktionen definiert werden. Sie können eine benutzerdefinierte Textformatierung angeben; hierzu stehen HTML-Tags in der Vorlage sowie Optionen auf der Registerkarte "Ausgabe" zur Verfügung. Sie können Datenwerte und andere bedingte Ausgaben mithilfe von CLEM-Ausdrücken in der Vorlage einschließen.

Beispiel

```
node = stream.create("report", "My node")
node.setPropertyValue("output_format", "HTML")
node.setPropertyValue("full_filename", "C:/report_output.html")
node.setPropertyValue("lines_per_page", 50)
node.setPropertyValue("title", "Report node created by a script")
node.setPropertyValue("highlights", False)
```

Tabelle 235. reportnode, Eigenschaften		
Eigenschaften von reportnode	**Datentyp**	**Eigenschaftsbeschreibung**
output_mode	Screen File	Dient zur Angabe des Zielorts für die vom Ausgabeknoten erstellte Ausgabe.
output_format	HTML (*.html*) Text (*.txt*) Output (*.cou*)	Dient zur Angabe des Typs der Dateiausgabe.
format	Auto Custom	Dient zur Angabe, ob die Ausgabe automatisch oder mithilfe des in der Vorlage enthalten HTML-Codes formatiert wird. Geben Sie Custom an, um die HTML-Formatierung der Vorlage zu verwenden.
use_output_name	*Flag*	Gibt an, ob ein benutzerdefinierter Ausgabename verwendet wird.
output_name	*string*	Wenn use_output_name "true" ist, gibt diese Eigenschaft den zu verwendenden Namen an.
text	*string*	
full_filename	*string*	
highlights	*Flag*	
title	*string*	
lines_per_page	*number*	

routputnode, Eigenschaften

Mit dem Knoten "Routput" können Sie Daten und die Ergebnisse des Modellscorings mithilfe Ihres eigenen benutzerdefinierten R-Scripts analysieren. Die Ausgabe von der Analyse kann Text oder grafisch sein. Die Ausgabe wird der Registerkarte **Ausgabe** des Managerbereichs hinzugefügt. Alternativ kann die Ausgabe in eine Datei umgeleitet werden.

Tabelle 236. routputnode, Eigenschaften		
Eigenschaften von routputnode	**Datentyp**	**Eigenschaftsbeschreibung**
syntax	*string*	
convert_flags	StringsAndDoubles LogicalValues	

Tabelle 236. routputnode, Eigenschaften (Forts.)		
Eigenschaften von routputnode	**Datentyp**	**Eigenschaftsbeschreibung**
convert_datetime	*Flag*	
convert_datetime_class	POSIXct POSIXlt	
convert_missing	*Flag*	
output_name	Auto Custom	
custom_name	*string*	
output_to	Screen File	
output_type	Graph Text	
full_filename	*string*	
graph_file_type	HTML COU	
text_file_type	HTML TEXT COU	

setglobalsnode, Eigenschaften

Mit dem Globalwerteknoten werden die Daten gescannt und Übersichtswerte berechnet, die in CLEM-Ausdrücken herangezogen werden können. Sie können diesen Knoten beispielsweise verwenden, um Statistiken für ein Feld mit dem Namen *Alter* zu berechnen und dann den Gesamtmittelwert von *Alter* in CLEM-Ausdrücken zu verwenden, indem die Funktion `@GLOBAL_MEAN(age)` eingefügt wird.

Beispiel

```
node = stream.create("setglobals", "My node")
node.setKeyedPropertyValue("globals", "Na", ["Max", "Sum", "Mean"])
node.setKeyedPropertyValue("globals", "K", ["Max", "Sum", "Mean"])
node.setKeyedPropertyValue("globals", "Age", ["Max", "Sum", "Mean", "SDev"])
node.setPropertyValue("clear_first", False)
node.setPropertyValue("show_preview", True)
```

Tabelle 237. setglobalsnode, Eigenschaften		
Eigenschaften von `setglobalsnode`	**Datentyp**	**Eigenschaftsbeschreibung**
`globals`	`[Sum Mean Min Max SDev]`	Strukturierte Eigenschaft, bei der festzulegende Felder mit folgender Syntax referenziert werden müssen: `node.setKeyedProperty-Value( "globals", "Age", ["Max", "Sum", "Mean", "SDev"])`
`clear_first`	*Flag*	
`show_preview`	*Flag*	

simevalnode, Eigenschaften

Der Simulationsevaluierungsknoten wertet ein angegebenes vorhergesagtes Zielfeld aus und stellt Verteilungs- und Korrelationsinformationen zum Zielfeld dar.

Tabelle 238. simevalnode, Eigenschaften		
Eigenschaften von `simevalnode`	**Datentyp**	**Eigenschaftsbeschreibung**
`target`	*Feld*	
`iteration`	*Feld*	
`presorted_by_iteration`	*Boolesch*	
`max_iterations`	*number*	
`tornado_fields`	*[field1...fieldN]*	
`plot_pdf`	*Boolesch*	
`plot_cdf`	*Boolesch*	
`show_ref_mean`	*Boolesch*	
`show_ref_median`	*Boolesch*	
`show_ref_sigma`	*Boolesch*	
`num_ref_sigma`	*number*	
`show_ref_pct`	*Boolesch*	
`ref_pct_bottom`	*number*	
`ref_pct_top`	*number*	
`show_ref_custom`	*Boolesch*	
`ref_custom_values`	*[number1...numberN]*	

Tabelle 238. simevalnode, Eigenschaften (Forts.)		
Eigenschaften von simevalnode	**Datentyp**	**Eigenschaftsbeschreibung**
category_values	Category Probabilities Both	
category_groups	Categories Iterations	
create_pct_table	*Boolesch*	
pct_table	Quartiles Intervals Custom	
pct_intervals_num	*number*	
pct_custom_values	*[number1...numberN]*	

simfitnode, Eigenschaften

Der Simulationsanpassungsknoten prüft die statistische Verteilung der Daten in jedem Feld und generiert (oder aktualisiert) einen Simulationsgenerierungsknoten, wobei jedem Feld die am besten angepasste Verteilung zugewiesen wird. Der Simulationsgenerierungsknoten kann dann zum Generieren simulierter Daten verwendet werden.

Tabelle 239. simfitnode, Eigenschaften		
Eigenschaften von simfitnode	**Datentyp**	**Eigenschaftsbeschreibung**
build	Node XMLExport Both	
use_source_node_name	*Boolesch*	
source_node_name	*string*	Der benutzerdefinierte Name des Quellenknotens, der generiert oder aktualisiert wird.
use_cases	All LimitFirstN	
use_case_limit	*Integer*	
fit_criterion	AndersonDarling KolmogorovSmirnov	
num_bins	*Integer*	
parameter_xml_filename	*string*	
generate_parameter_import	*Boolesch*	

statisticsnode, Eigenschaften

Der Statistikknoten liefert grundlegende Übersichtsdaten zu numerischen Feldern. Er berechnet Übersichtsstatistiken für einzelne Felder und für die Korrelationen zwischen den Feldern.

Beispiel

```
node = stream.create("statistics", "My node")
# "Settings" tab
node.setPropertyValue("examine", ["Age", "BP", "Drug"])
node.setPropertyValue("statistics", ["mean", "sum", "sdev"])
node.setPropertyValue("correlate", ["BP", "Drug"])
# "Correlation Labels..." section
node.setPropertyValue("label_correlations", True)
node.setPropertyValue("weak_below_absolute", 0.25)
node.setPropertyValue("weak_label", "lower quartile")
node.setPropertyValue("strong_above_absolute", 0.75)
node.setPropertyValue("medium_label", "middle quartiles")
node.setPropertyValue("strong_label", "upper quartile")
# "Output" tab
node.setPropertyValue("full_filename", "c:/output/statistics_output.html")
node.setPropertyValue("output_format", "HTML")
```

Tabelle 240. statisticsnode, Eigenschaften

Eigenschaften von `statisticsnode`	**Datenart**	**Eigenschaftsbeschreibung**
`use_output_name`	*Flag*	Gibt an, ob ein benutzerdefinierter Ausgabename verwendet wird.
`output_name`	*string*	Wenn `use_output_name` "true" ist, gibt diese Eigenschaft den zu verwendenden Namen an.

`output_mode`	`Screen`
	
	`File`
		Dient zur Angabe des Zielorts für die vom Ausgabeknoten erstellte Ausgabe.

`output_format`	`Text` (*.txt*) `HTML` (*.html*) `Output` (*.cou*)	Dient zur Angabe des Ausgabetyps.
`full_filename`	*string*	
`examine`	*Liste*	
`correlate`	*Liste*	
`statistics`	`[count mean sum min max range variance sdev semean median mode]`	

Tabelle 240. statisticsnode, Eigenschaften (Forts.)		
Eigenschaften von `statisticsnode`	**Datenart**	**Eigenschaftsbeschreibung**

`correlation_mode`	`Probability`
	
	`Absolute`
		Gibt an, ob die Korrelationen nach Wahrscheinlichkeit oder anhand des absoluten Werts beschriftet werden sollen.

`label_correlations`	*Flag*	
`weak_label`	*string*	
`medium_label`	*string*	
`strong_label`	*string*	
`weak_below_probability`	*number*	Wenn `correlation_mode` auf `Probability` gesetzt ist, wird hier der Trennwert für schwache Korrelationen angegeben. Hierbei muss es sich um einen Wert zwischen 0 und 1 handeln, beispielsweise 0,90.
`strong_above_probability`	*number*	Trennwert für starke Korrelationen.
`weak_below_absolute`	*number*	Wenn `correlation_mode` auf `Absolute` gesetzt ist, wird hier der Trennwert für schwache Korrelationen angegeben. Hierbei muss es sich um einen Wert zwischen 0 und 1 handeln, beispielsweise 0,90.
`strong_above_absolute`	*number*	Trennwert für starke Korrelationen.

Eigenschaften von "statisticsoutputnode"

Mit dem Statistics-Ausgabeknoten können Sie eine IBM SPSS Statistics-Prozedur aufrufen, um Ihre IBM SPSS Modeler-Daten zu analysieren. Es stehen zahlreiche IBM SPSS Statistics-Analyseprozeduren zur Verfügung. Für diesen Knoten ist eine lizenzierte Kopie von IBM SPSS Statistics erforderlich.

Eine Beschreibung der Eigenschaften für diesen Knoten finden Sie in „Eigenschaften von "statisticsoutputnode"” auf Seite 443.

tablenode, Eigenschaften

Der Tabellenknoten zeigt die Daten in Tabellenform an, die auch in eine Datei geschrieben werden kann. Diese Vorgehensweise empfiehlt sich immer dann, wenn die Datenwerte überprüft oder in leicht lesbarer Form exportiert werden sollen.

Beispiel

```
node = stream.create("table", "My node")
node.setPropertyValue("highlight_expr", "Age > 30")
node.setPropertyValue("output_format", "HTML")
```

```
node.setPropertyValue("transpose_data", True)
node.setPropertyValue("full_filename", "C:/output/table_output.htm")
node.setPropertyValue("paginate_output", True)
node.setPropertyValue("lines_per_page", 50)
```

Tabelle 241. tablenode, Eigenschaften

Eigenschaften von tablenode	**Datentyp**	**Eigenschaftsbeschreibung**
full_filename	*string*	Bei Datenträger-, Daten- oder HTML-Ausgabe gibt diese Eigenschaft den Namen der Ausgabedatei an.
use_output_name	*Flag*	Gibt an, ob ein benutzerdefinierter Ausgabename verwendet wird.
output_name	*string*	Wenn use_output_name "true" ist, gibt diese Eigenschaft den zu verwendenden Namen an.
output_mode	Screen File	Dient zur Angabe des Zielorts für die vom Ausgabeknoten erstellte Ausgabe.
output_format	Formatted (*.tab*) Delimited (*.csv*) HTML (*.html*) Output (*.cou*)	Dient zur Angabe des Ausgabetyps.
transpose_data	*Flag*	Transponiert die Daten vor dem Export, sodass die Zeilen Felder und die Spalten Datensätze darstellen.
paginate_output	*Flag*	Wenn output_format auf HTML gesetzt ist, wird hiermit die Ausgabe in Seiten unterteilt.
lines_per_page	*number*	Bei Verwendung mit paginate_output wird die Anzahl der Zeilen pro Ausgabeseite angegeben.
highlight_expr	*string*	
output	*string*	Eine schreibgeschützte Eigenschaft, die eine Referenz zur letzten vom Knoten erstellten Tabelle enthält.
value_labels	*[[Wert Beschriftungszeichenfolge]* *[Wert Beschriftungszeichenfolge] ...]*	Gibt Beschriftungen für Wertpaare an.
display_places	*Integer*	Legt die Dezimalstellen für das Feld bei der Anzeige fest (gilt nur für Felder mit dem Speichertyp REELLE ZAHL). Mit dem Wert -1 wird der Streamstandard verwendet.

Tabelle 241. tablenode, Eigenschaften (Forts.)

Eigenschaften von tablenode	Datentyp	Eigenschaftsbeschreibung
export_places	*Integer*	Legt die Dezimalstellen für das Feld beim Exportieren fest (gilt nur für Felder mit dem Speichertyp REELLE ZAHL). Mit dem Wert -1 wird der Streamstandard verwendet.
decimal_separator	DEFAULT PERIOD COMMA	Legt das Dezimaltrennzeichen für das Feld fest (gilt nur für Felder mit dem Speichertyp REELLE ZAHL).
date_format	"DDMMYY" "MMDDYY" "YYMMDD" "YYYYMMDD" "YYYYDDD" DAY MONTH "DD-MM-YY" "DD-MM-YYYY" "MM-DD-YY" "MM-DD-YYYY" "DD-MON-YY" "DD-MON-YYYY" "YYYY-MM-DD" "DD.MM.YY" "DD.MM.YYYY" "MM.DD.YYYY" "DD.MON.YY" "DD.MON.YYYY" "DD/MM/YY" "DD/MM/YYYY" "MM/DD/YY" "MM/DD/YYYY" "DD/MON/YY" "DD/MON/YYYY" MON YYYY q Q YYYY ww WK YYYY	Legt das Datumsformat für das Feld fest (gilt nur für Felder mit dem Speichertyp DATE oder TIMESTAMP).

Tabelle 241. tablenode, Eigenschaften (Forts.)		
Eigenschaften von tablenode	**Datentyp**	**Eigenschaftsbeschreibung**
time_format	"HHMMSS" "HHMM" "MMSS" "HH:MM:SS" "HH:MM" "MM:SS" "(H)H:(M)M:(S)S" "(H)H:(M)M" "(M)M:(S)S" "HH.MM.SS" "HH.MM" "MM.SS" "(H)H.(M)M.(S)S" "(H)H.(M)M" "(M)M.(S)S"	Legt das Zeitformat für das Feld fest (gilt nur für Felder mit dem Speichertyp TIME oder TIMESTAMP).
column_width	*Integer*	Legt die Spaltenbreite für das Feld fest. Mit dem Wert -1 wird die Spaltenbreite auf Auto (Automatisch) gesetzt.
justify	AUTO CENTER LEFT RIGHT	Legt die Spaltenausrichtung für das Feld fest.

transformnode, Eigenschaften

Mit dem Transformationsknoten können Sie die Ergebnisse von Transformationen auswählen und visuell voranzeigen, bevor Sie sie auf ausgewählte Felder anwenden.

Beispiel

```
node = stream.create("transform", "My node")
node.setPropertyValue("fields", ["AGE", "INCOME"])
node.setPropertyValue("formula", "Select")
node.setPropertyValue("formula_log_n", True)
node.setPropertyValue("formula_log_n_offset", 1)
```

Tabelle 242. transformnode, Eigenschaften

Eigenschaften von `transformnode`	**Datentyp**	**Eigenschaftsbeschreibung**
`fields`	[*feld1... feldn*]	Die bei der Transformation zu verwendenden Felder.

`formula`	`All`
	
	`Select`
		Gibt an, ob alle oder nur die ausgewählten Transformationen berechnet werden sollen.

`formula_inverse`	*Flag*	Gibt an, ob die inversen Transformation verwendet werden soll.
`formula_inverse_offset`	*number*	Gibt eine relative Datenadresse an, die für die Formel verwendet werden soll. Standardmäßig auf 0 gesetzt, sofern vom Benutzer nicht anders angegeben.
`formula_log_n`	*Flag*	Gibt an, ob die $\log_n$-Transformation verwendet werden soll.
`formula_log_n_offset`	*number*	
`formula_log_10`	*Flag*	Gibt an, ob die $\log_{10}$-Transformation verwendet werden soll.
`formula_log_10_offset`	*number*	
`formula_exponential`	*Flag*	Gibt an, ob die exponentielle Transformation (e^x) verwendet werden soll.
`formula_square_root`	*Flag*	Gibt an, ob die Quadratwurzeltransformation verwendet werden soll.
`use_output_name`	*Flag*	Gibt an, ob ein benutzerdefinierter Ausgabename verwendet wird.
`output_name`	*string*	Wenn use_output_name true ist, gibt diese Eigenschaft den zu verwendenden Namen an.

`output_mode`	`Screen`
	
	`File`
		Dient zur Angabe des Zielorts für die vom Ausgabeknoten erstellte Ausgabe.

Tabelle 242. transformnode, Eigenschaften (Forts.)

Eigenschaften von `transformnode`	**Datentyp**	**Eigenschaftsbeschreibung**
`output_format`	`HTML` (*.html*) `Output` (*.cou*)	Dient zur Angabe des Ausgabetyps.
`paginate_output`	*Flag*	Wenn `output_format` auf `HTML` gesetzt ist, wird hiermit die Ausgabe in Seiten unterteilt.
`lines_per_page`	*number*	Bei Verwendung mit `paginate_output` wird die Anzahl der Zeilen pro Ausgabeseite angegeben.
`full_filename`	*string*	Gibt den für die Dateiausgabe zu verwendenden Dateinamen an.

Kapitel 17. Eigenschaften von Exportknoten

Allgemeine Eigenschaften von Exportknoten

Folgende Eigenschaften haben alle Exportknoten gemeinsam:

Tabelle 243. Allgemeine Eigenschaften von Exportknoten

Eigenschaft	Werte	Eigenschaftsbeschreibung
publish_path	*string*	Geben Sie den Stammnamen für die veröffentlichten Image- und Parameterdateien an.
publish_metadata	*Flag*	Gibt an, ob eine Metadatendatei erzeugt wird, welche die Ein- und Ausgaben des Bilds und der zugehörigen Datenmodelle beschreibt.
publish_use_parameters	*Flag*	Gibt an, ob Streamparameter in der *.par-Datei enthalten sind.
publish_parameters	*Zeichenfolgeliste*	Geben Sie die Parameter an, die eingeschlossen werden sollen.
execute_mode	export_data publish	Gibt an, ob der Knoten ohne Veröffentlichen des Streams ausgeführt wird oder ob der Stream automatisch beim Ausführen des Knotens veröffentlicht wird.

Eigenschaften von "asexport"

Der Analytic Server-Export ermöglicht Ihnen die Ausführung eines Streams unter HDFS (Hadoop Distributed File System).

Beispiel

```
node.setPropertyValue("use_default_as", False)
node.setPropertyValue("connection",
["false","9.119.141.141","9080","analyticserver","ibm","admin","admin","fal
se","","","","",""])
```

Tabelle 244. Eigenschaften von "asexport"

Eigenschaften von asexport	Datentyp	Eigenschaftsbeschreibung
data_source	*string*	Der Name der Datenquelle.
export_mode	*string*	Gibt an, ob die exportierten Daten an die vorhandene Datenquelle angehängt (**append**) werden oder die vorhandene Datenquelle überschreiben (**overwrite**).

Tabelle 244. Eigenschaften von "asexport" (Forts.)		
Eigenschaften von asexport	**Datentyp**	**Eigenschaftsbeschreibung**
use_default_as	*Boolesch*	Wenn True festgelegt ist, wird die Analytic Server-Standardverbindung verwendet, die in der Serverdatei options.cfg konfiguriert ist. Wenn die Option auf False gesetzt ist, wird die Verbindung dieses Knotens verwendet.
connection	["string","string","string", "string","string","string","string", "string" ,"string","string", "string" ,"string"]	Eine Listeneigenschaft mit den Verbindungsdetails für Analytic Server. Das Format lautet: ["is_secure_connect", "server_url", "server_port", "context_root", "consumer", "user_name", "password", "use-kerberos-auth", "kerberos-krb5-config-file-path", "kerberos-jaas-config-file-path", "kerberos-krb5-service-principal-name", "enable-kerberos-debug"]. Dabei gilt Folgendes: is_secure_connect: gibt an, ob eine sichere Verbindung verwendet wird, und ist entweder true oder false. use-kerberos-auth: gibt an, ob die Kerberos-Authentifizierung verwendet wird, und ist entweder true oder false. enable-kerberos-debug: gibt an, ob der Debugmodus der Kerberos-Authentifizierung verwendet wird, und ist entweder true oder false.

Eigenschaften von "cognosexportnode"

Der IBM Cognos-Exportknoten exportiert Daten in einem Format, das von Cognos-Datenbanken gelesen werden kann.

Für diesen Knoten müssen Sie eine Cognos-Verbindung und eine ODBC-Verbindung definieren.

Cognos-Verbindung

Im Folgenden finden Sie die Eigenschaften für die Cognos-Verbindung.

Tabelle 245. Eigenschaften von "cognosexportnode"

Eigenschaften von cognosexportnode	**Datentyp**	**Eigenschaftsbeschreibung**
`cognos_connection`	*["Zeichenfolge","Flag","Zeichenfolge","Zeichenfolge","Zeichenfolge"]*	Eine Listeneigenschaft mit den Verbindungsdetails für den Cognos-Server. Format: `["Cognos_server_URL", login_mode, "namespace", "username", "password"]` Dabei gilt Folgendes: `Cognos_server_URL` ist die URL des Cognos-Servers, der die Quelle enthält. `login_mode` gibt an, ob die anonyme Anmeldung verwendet wird, und ist entweder `true` oder `false`; bei Angabe von `true` sollten die folgenden Felder auf `""` gesetzt werden. `namespace` gibt den Sicherheitsauthentifizierungsprovider an, der für die Anmeldung beim Server verwendet wird. `username` und `password` werden für die Anmeldung beim Cognos-Server verwendet. Anstelle von `login_mode` sind die folgenden Modi verfügbar: • `anonymousMode`. Beispiel: `['Cognos_server_url', 'anonymousMode', "namespace", "username", "password"]` • `credentialMode`. Beispiel: `['Cognos_server_url', 'credentialMode', "namespace", "username", "password"]`

Tabelle 245. Eigenschaften von "cognosexportnode" (Forts.)		
Eigenschaften von cognosexportnode	**Datentyp**	**Eigenschaftsbeschreibung**
		• storedCredentialMode. Beispiel: ['Cognos_server_url', 'storedCredentialMode', "stored_credential_name"] Dabei ist stored_credential_name der Name eines Cognos-Berechtigungsnachweises im Repository.
cognos_package_name	*string*	Pfad und Name des Cognos-Pakets, an die Sie Daten exportieren, z. B.: /Public Folders/MyPackage
cognos_datasource	*string*	
cognos_export_mode	Publish ExportFile	
cognos_filename	*string*	

ODBC-Verbindung

Die Eigenschaften für die ODBC-Verbindung sind mit den Eigenschaften identisch, die für databaseexportnode im nächsten Abschnitt aufgelistet sind, mit der Ausnahme, dass die Eigenschaft datasource nicht gültig ist.

databaseexportnode, Eigenschaften

Der Datenbankexportknoten schreibt Daten in eine ODBC-kompatible relationale Datenquelle. Um Daten in eine ODBC-Datenquelle schreiben zu können, muss die Datenquelle vorhanden sein und Sie müssen über Schreibberechtigung für sie verfügen.

Beispiel

```
'''
Assumes a datasource named "MyDatasource" has been configured
'''
stream = modeler.script.stream()
db_exportnode = stream.createAt("databaseexport", "DB Export", 200, 200)
applynn = stream.findByType("applyneuralnetwork", None)
stream.link(applynn, db_exportnode)

# Export tab
db_exportnode.setPropertyValue("username", "user")
db_exportnode.setPropertyValue("datasource", "MyDatasource")
db_exportnode.setPropertyValue("password", "password")
db_exportnode.setPropertyValue("table_name", "predictions")
db_exportnode.setPropertyValue("write_mode", "Create")
db_exportnode.setPropertyValue("generate_import", True)
db_exportnode.setPropertyValue("drop_existing_table", True)
```

```
db_exportnode.setPropertyValue("delete_existing_rows", True)
db_exportnode.setPropertyValue("default_string_size", 32)

# Schema dialog
db_exportnode.setKeyedPropertyValue("type", "region", "VARCHAR(10)")
db_exportnode.setKeyedPropertyValue("export_db_primarykey", "id", True)
db_exportnode.setPropertyValue("use_custom_create_table_command", True)
db_exportnode.setPropertyValue("custom_create_table_command", "My SQL Code")

# Indexes dialog
db_exportnode.setPropertyValue("use_custom_create_index_command", True)
db_exportnode.setPropertyValue("custom_create_index_command", "CREATE BITMAP
INDEX <index-name>
 ON <table-name> <(index-columns)>")
db_exportnode.setKeyedPropertyValue("indexes", "MYINDEX", ["fields", ["id",
"region"]])
```

Tabelle 246. databaseexportnode, Eigenschaften

Eigenschaften von databaseexportnode	**Datentyp**	**Eigenschaftsbeschreibung**
datasource	*string*	
username	*string*	
password	*string*	
epassword	*string*	Dieser Slot ist während der Ausführung schreibgeschützt. Um ein verschlüsseltes Kennwort zu erstellen, verwenden Sie das Kennworttool im Menü "Tools". Weitere Informationen finden Sie im Thema „Erstellen eines verschlüsselten Kennworts” auf Seite 58.
table_name	*string*	
write_mode	Create Append Merge	
map	*string*	Ordnet einen Streamfeldnamen zu einer Datenbankspalte zu (nur gültig, wenn write_mode auf Merge eingestellt ist). Für eine Zusammenführung müssen alle Felder zugeordnet sein, damit sie exportiert werden. Feldnamen, die in der Datenbank nicht vorhanden sind, werden als neue Spalten hinzugefügt.

Tabelle 246. databaseexportnode, Eigenschaften (Forts.)

Eigenschaften von databaseexportnode	**Datentyp**	**Eigenschaftsbeschreibung**
key_fields	*Liste*	Gibt an, dass das Streamfeld für "key" verwendet wird. Die map-Eigenschaft zeigt, welche Entsprechung in der Datenbank vorhanden ist.
join	Database Add	
drop_existing_table	*Flag*	
delete_existing_rows	*Flag*	
default_string_size	*Integer*	
type		Strukturierte Eigenschaft, mit der das Schema festgelegt wird.
generate_import	*Flag*	
use_custom_create_table_command	*Flag*	Mit dem Slot *custom_create_table* ändern Sie den SQL-Standardbefehl CREATE TABLE.
custom_create_table_command	*string*	Gibt eine Befehlszeichenfolge an, die statt des SQL-Standardbefehls CREATE TABLE verwendet werden soll.
use_batch	*Flag*	Bei den folgenden Eigenschaften handelt es sich um erweiterte Optionen für das Massenladen von Datenbanken. Mit dem Wert "True" für Use_batch werden zeilenweise Commits zur Datenbank inaktiviert.
batch_size	*number*	Hier können Sie die Anzahl der Datensätze angeben, die an die Datenbank gesendet werden sollen, bevor die Übertragung in den Speicher erfolgt.
bulk_loading	Off ODBC External	Gibt den Typ für das Massenladen an. Zusätzliche Optionen für ODBC und External sind unten aufgeführt.
not_logged	*Flag*	
odbc_binding	Row Column	Geben Sie eine zeilen- oder spaltenweise Bindung für das Massenladen über ODBC an.

Tabelle 246. databaseexportnode, Eigenschaften (Forts.)

Eigenschaften von databaseexportnode	**Datentyp**	**Eigenschaftsbeschreibung**
loader_delimit_mode	Tab Space Other	Geben Sie für das Massenladen über ein externes Programm das Trennzeichen an. Wählen Sie Other in Verbindung mit der Eigenschaft loader_other_delimiter zur Angabe von Trennzeichen, wie z. B. Komma (,).
loader_other_delimiter	*string*	
specify_data_file	*Flag*	Mit dem Flag "True" wird die unten genannte Eigenschaft data_file aktiviert, mit der Sie den Dateinamen und den Pfad für das Speichern beim Massenladen in die Datenbank angeben können.
data_file	*string*	
specify_loader_program	*Flag*	Mit dem Flag "True" wird die unten genannte Eigenschaft loader_program aktiviert, mit der Sie den Namen und den Speicherort eines externen Ladescripts oder Ladeprogramms angeben können.
loader_program	*string*	
gen_logfile	*Flag*	Mit dem Flag "True" wird die unten genannte Eigenschaft logfile_name aktiviert, mit der Sie den Namen einer Datei zur Erzeugung eines Fehlerprotokolls auf dem Server angeben können.
logfile_name	*string*	
check_table_size	*Flag*	Mit dem Flag "True" wird die Tabellenprüfung ermöglicht, um sicherzustellen, dass die Zunahme der Größe der Datenbanktabelle der Anzahl der aus IBM SPSS Modeler exportierten Zeilen entspricht.
loader_options	*string*	Legen Sie für das Ladeprogramm zusätzliche Argumente fest, z. B. -comment und -specialdir.

Tabelle 246. databaseexportnode, Eigenschaften (Forts.)		
Eigenschaften von databaseexportnode	**Datentyp**	**Eigenschaftsbeschreibung**
export_db_primarykey	*Flag*	Gibt an, ob es sich bei einem bestimmten Feld um einen Primärschlüssel handelt.
use_custom_create_index_command	*Flag*	Bei true wird hiermit benutzerdefinierte SQL für alle Indizes aktiviert.
custom_create_index_command	*string*	Gibt den SQL-Befehl an, der zum Erstellen von Indizes verwendet wird, wenn benutzerdefiniertes SQL aktiviert ist. (Dieser Wert kann für bestimmte Indizes außer Kraft gesetzt werden (siehe unten).)
indexes.INDEXNAME.fields		Erstellt bei Bedarf den angegebenen Index und listet die in diesen Index aufzunehmenden Feldnamen auf.
INDEXNAME "use_custom_create_index_command"	*Flag*	Wird zum Aktivieren bzw. Inaktivieren der benutzerdefinierten SQL für einen bestimmten Index verwendet. Siehe Beispiele nach der folgenden Tabelle.
INDEXNAME "custom_create_index_command"	*string*	Gibt die für den angegebenen Index verwendete benutzerdefinierte SQL an. Siehe Beispiele nach der folgenden Tabelle.
indexes.INDEXNAME.remove	*Flag*	Bei True wird hiermit der angegebene Index aus der Menge der Indizes entfernt.
table_space	*string*	Gibt den zu erstellenden Tabellenbereich an.
use_partition	*Flag*	Gibt an, dass das Verteilungs-Hashfeld verwendet werden soll.
partition_field	*string*	Gibt den Inhalt des Verteilungs-Hashfelds an.

Anmerkung: Bei einigen Datenbanken können Sie angeben, dass Datenbanktabellen für den Export mit Komprimierung erstellt werden (z. B. Äquivalent von CREATE TABLE MYTABLE (...) COMPRESS YES; in SQL). Die Eigenschaften use_compression und compression_mode werden zur Unterstützung dieser Funktion wie folgt bereitgestellt.

Tabelle 247. Eigenschaften von "databaseexportnode" mit Komprimierungsfunktionen		
Eigenschaften von databaseexportnode	**Datentyp**	**Eigenschaftsbeschreibung**
use_compression	*boolesch*	Wenn auf True gesetzt, werden Tabellen für den Export mit Komprimierung erstellt.

Tabelle 247. Eigenschaften von "databaseexportnode" mit Komprimierungsfunktionen (Forts.)		
Eigenschaften von databaseexportnode	**Datentyp**	**Eigenschaftsbeschreibung**
compression_mode	Row Page	Legt das Komprimierungsniveau für SQL Server-Datenbanken fest.
	Default Direct_Load_Operations All_Operations Basic OLTP Query_High Query_Low Archive_High Archive_Low	Legt das Komprimierungsniveau für Oracle-Datenbanken fest. Beachten Sie, dass für die Werte OLTP, Query_High, Query_Low, Archive_High und Archive_Low mindestens Oracle 11gR2 erforderlich ist.

Beispiel für das Ändern des Befehls CREATE INDEX für einen bestimmten Index:

```
db_exportnode.setKeyedPropertyValue("indexes", "MYINDEX", ["use_custom_crea
te_index_command",
 True])db_exportnode.setKeyedPropertyValue("indexes", "MYINDEX", ["cus
tom_create_index_command",
 "CREATE BITMAP INDEX <index-name> ON <table-name> <(index-columns)>"])
```

Diese Änderung ist auch über eine Hashtabelle möglich:

```
db_exportnode.setKeyedPropertyValue("indexes", "MYINDEX", ["fields":["id",
"region"],
 "use_custom_create_index_command":True, "custom_create_index_command":"CREA
TE INDEX <index-name> ON
 <table-name> <(index-columns)>"])
```

Eigenschaften von "datacollectionexportnode"

Der Data Collection-Exportknoten gibt Daten in dem von der Marktforschungssoftware Data Collection verwendeten Format aus. Zur Verwendung dieses Knotens muss eine Data Collection-Datenbibliothek installiert sein.

Beispiel

```
stream = modeler.script.stream()
datacollectionexportnode = stream.createAt("datacollectionexport", "Data Col
lection", 200, 200)
```

```
datacollectionexportnode.setPropertyValue("metadata_file", "c:\\museums.mdd")
datacollectionexportnode.setPropertyValue("merge_metadata", "Overwrite")
datacollectionexportnode.setPropertyValue("casedata_file", "c:\\museumda
ta.sav")
datacollectionexportnode.setPropertyValue("generate_import", True)
datacollectionexportnode.setPropertyValue("enable_system_variables", True)
```

Tabelle 248. Eigenschaften von "datacollectionexportnode"

Eigenschaften von datacollectionexport-node	**Datentyp**	**Eigenschaftsbeschreibung**
metadata_file	*string*	Name der zu exportierenden Metadatendatei.
merge_metadata	Overwrite MergeCurrent	
enable_system_variables	*Flag*	Gibt an, ob die exportier-te *.mdd*-Datei Data Collection-Systemvariablen enthalten soll.
casedata_file	*string*	Der Name der *.sav*-Datei, in die die Falldaten exportiert wer-den.
generate_import	*Flag*	

Eigenschaften von "excelexportnode"

Der Excel-Exportknoten gibt Daten in Microsoft Excel aus.xlsx-Dateiformat. Optio-nal können Sie auswählen, dass bei der Ausführung des Knotens Excel automatisch gestartet und die exportierte Datei geöffnet werden soll.

Beispiel

```
stream = modeler.script.stream()
excelexportnode = stream.createAt("excelexport", "Excel", 200, 200)
excelexportnode.setPropertyValue("full_filename", "C:/output/myexport.xlsx")
excelexportnode.setPropertyValue("excel_file_type", "Excel2007")
excelexportnode.setPropertyValue("inc_field_names", True)
excelexportnode.setPropertyValue("inc_labels_as_cell_notes", False)
excelexportnode.setPropertyValue("launch_application", True)
excelexportnode.setPropertyValue("generate_import", True)
```

Tabelle 249. Eigenschaften von "excelexportnode"

Eigenschaften von excelexportnode	**Datentyp**	**Eigenschaftsbeschreibung**
full_filename	*string*	
excel_file_type	Excel2007	
export_mode	Create Append	
inc_field_names	*Flag*	Gibt an, ob Feldnamen in die erste Zeile des Arbeitsblattes eingefügt werden sollen.

Tabelle 249. Eigenschaften von "excelexportnode" (Forts.)		
Eigenschaften von excelexportnode	**Datentyp**	**Eigenschaftsbeschreibung**
start_cell	*string*	Gibt die Startzelle für den Export an.
worksheet_name	*string*	Name des zu schreibenden Arbeitsblattes.
launch_application	*Flag*	Gibt an, ob Excel für die resultierende Datei aufgerufen werden soll. Beachten Sie, dass der Pfad für den Start von Excel im Dialogfeld "Hilfsanwendungen" (Menü "Tools", "Hilfsanwendungen") angegeben werden muss.
generate_import	*Flag*	Gibt an, ob ein Excel-Importknoten generiert werden soll, der die exportierte Datendatei liest.

Eigenschaften von "extensionexportnode"

Mit dem Erweiterungsexportknoten können Sie Scripts in R oder Python for Spark ausführen, um Daten zu exportieren.

Beispiel für Python for Spark

```
#### script example for Python for Spark
import modeler.api
stream = modeler.script.stream()
node = stream.create("extension_export", "extension_export")
node.setPropertyValue("syntax_type", "Python")

python_script = """import spss.pyspark.runtime
from pyspark.sql import SQLContext
from pyspark.sql.types import *

cxt = spss.pyspark.runtime.getContext()
df = cxt.getSparkInputData()
print df.dtypes[:]
_newDF = df.select("Age","Drug")
print _newDF.dtypes[:]

df.select("Age", "Drug").write.save("c:/data/ageAndDrug.json", format="json")
"""

node.setPropertyValue("python_syntax", python_script)
```

Beispiel für R

```
#### script example for R
node.setPropertyValue("syntax_type", "R")
node.setPropertyValue("r_syntax", """write.csv(modelerData, "C:/export.csv")""")
```

Tabelle 250. Eigenschaften von "extensionexportnode"		
Eigenschaften von `extensionexportnode`	**Datentyp**	**Eigenschaftsbeschreibung**
`syntax_type`	*R* *Python*	Gibt das Script an, das ausgeführt wird - R oder Python (R ist der Standardwert).
`r_syntax`	*string*	Die R-Scriptsyntax für die Ausführung.
`python_syntax`	*string*	Die Python-Scriptsyntax für die Ausführung.
`convert_flags`	`StringsAndDoubles` `LogicalValues`	Option zum Konvertieren von Flagfeldern.
`convert_missing`	*Flag*	Option zum Konvertieren fehlender Werte in den R-Wert NA .
`convert_datetime`	*Flag*	Option zum Konvertieren von Variablen mit Datums- oder Datums-/Zeitformaten in R-Datums-/Zeitformate.
`convert_datetime_class`	`POSIXct` `POSIXlt`	Optionen, die angeben, in welches Format Variablen mit Datums- oder Datums-/Zeitformaten konvertiert werden.

Eigenschaften von "jsonexportnode"

Der JSON-Exportknoten gibt Daten im JSON-Format aus.

Tabelle 251. Eigenschaften von "jsonexportnode"		
Eigenschaften von `jsonexportnode`	**Datenart**	**Eigenschaftsbeschreibung**
`full_filename`	*string*	Der vollständige Dateiname mit Pfad.
`string_format`	*Datensätze* *Werte*	Geben Sie das Format der JSON-Zeichenfolge an. Der Standardwert ist `records`.
`generate_import`	*Flag*	Gibt an, ob ein JSON-Importknoten generiert werden soll, der die exportierte Datendatei liest. Der Standardwert ist `False`.

Eigenschaften von "outputfilenode"

Der Flatfile-Export gibt Daten in einer Textdatei mit Trennzeichen aus. Diese Vorgehensweise eignet sich für das Exportieren von Daten, die von anderen Analyse- oder Tabellenkalkulationsprogrammen gelesen werden können.

Beispiel

```
stream = modeler.script.stream()
outputfile = stream.createAt("outputfile", "File Output", 200, 200)
outputfile.setPropertyValue("full_filename", "c:/output/flatfile_output.txt")
outputfile.setPropertyValue("write_mode", "Append")
outputfile.setPropertyValue("inc_field_names", False)
outputfile.setPropertyValue("use_newline_after_records", False)
outputfile.setPropertyValue("delimit_mode", "Tab")
outputfile.setPropertyValue("other_delimiter", ",")
outputfile.setPropertyValue("quote_mode", "Double")
outputfile.setPropertyValue("other_quote", "*")
outputfile.setPropertyValue("decimal_symbol", "Period")
outputfile.setPropertyValue("generate_import", True)
```

Tabelle 252. Eigenschaften von "outputfilenode"

Eigenschaften von outputfilenode	Datentyp	Eigenschaftsbeschreibung
full_filename	*string*	Name der Ausgabedatei.
write_mode	Overwrite Append	
inc_field_names	*Flag*	
use_newline_after_records	*Flag*	
delimit_mode	Comma Tab Space Other	
other_delimiter	*Zeichen*	
quote_mode	None Single Double Other	
other_quote	*Flag*	
generate_import	*Flag*	

Tabelle 252. Eigenschaften von "outputfilenode" (Forts.)		
Eigenschaften von outputfilenode	**Datentyp**	**Eigenschaftsbeschreibung**
encoding	StreamDefault SystemDefault "UTF-8"	

Eigenschaften von "sasexportnode"

Mit dem SAS-Exportknoten werden Daten in das SAS-Format ausgegeben, die dann in SAS oder in SAS-kompatible Softwarepakete eingelesen werden können. Drei SAS-Dateiformate sind verfügbar: SAS für Windows/OS2, SAS für UNIX oder SAS Version 7/8.

Beispiel

```
stream = modeler.script.stream()
sasexportnode = stream.createAt("sasexport", "SAS Export", 200, 200)
sasexportnode.setPropertyValue("full_filename", "c:/output/SAS_out
put.sas7bdat")
sasexportnode.setPropertyValue("format", "SAS8")
sasexportnode.setPropertyValue("export_names", "NamesAndLabels")
sasexportnode.setPropertyValue("generate_import", True)
```

Tabelle 253. Eigenschaften von "sasexportnode"		
Eigenschaften von sasexportnode	**Datentyp**	**Eigenschaftsbeschreibung**
format	Windows UNIX SAS7 SAS8	Beschriftungsfelder für die Eigenschaft "Variante".
full_filename	*string*	
export_names	NamesAndLabels NamesAsLabels	Dient der Zuordnung von Feldnamen von IBM SPSS Modeler zu IBM SPSS Statistics- oder SAS-Variablennamen nach dem Export.
generate_import	*Flag*	

Eigenschaften von "statisticsexportnode

Der Statistikexportknoten gibt Daten im IBM SPSS Statistics-Format *.sav* oder *.zsav* aus. Die Dateien `.sav` oder `.zsav` können von IBM SPSS Statistics Base und anderen Produkten gelesen werden. Dieses Format wird auch für Cachedateien in IBM SPSS Modeler verwendet.

Eine Beschreibung der Eigenschaften für diesen Knoten finden Sie in „Eigenschaften von "statisticsexportnode" auf Seite 443.

Eigenschaften des Knotens "tm1odataexport"

Der IBM Cognos TM1-Exportknoten exportiert Daten in einem Format, das von Cognos TM1-Datenbanken gelesen werden kann.

Tabelle 254. Eigenschaften des Knotens "tm1odataexport"

Eigenschaften des Knotens `tm1odataexport`	**Datentyp**	**Eigenschaftsbeschreibung**
`credential_type`	inputCredential oder storedCredential	Dient zur Angabe des Berechtigungsnachweistyps.
`input_credential`	*Liste*	Wenn `credential_type` auf `inputCredential` gesetzt ist; geben Sie die Domäne, den Benutzernamen und das Kennwort an.
`stored_credential_name`	*string*	Wenn `credential_type` `storedCredential` ist, geben Sie den Namen des Berechtigungsnachweises auf dem C&DS-Server an.
`selected_cube`	*Feld*	Der Name des Cubes, in den Sie Daten exportieren. Beispiel: `TM1_export.setPropertyValue("selected_cube", "plan_BudgetPlan")`

Tabelle 254. Eigenschaften des Knotens "tm1odataexport" (Forts.)

Eigenschaften des Knotens `tm1odataexport`	**Datentyp**	**Eigenschaftsbeschreibung**
`spss_field_to_tm1_element_mapping`	*Liste*	Das zuzuordnende TM1-Element muss Teil der Spaltendimension für die ausgewählte Cube-Ansicht sein. Das Format lautet wie folgt: `[[[Field_1, Dimension_1, False], [Element_1, Dimension_2, True], ...], [[Field_2, ExistMeasureElement, False], [Field_3, NewMeasureElement, True], ...]]` Es gibt zwei Listen für die Beschreibung der Zuordnungsinformationen. Die Zuordnung eines Blattelements zu einer Dimension entspricht dem nachfolgenden Beispiel 2. Beispiel 1: Die erste Liste (`[[Field_1, Dimension_1, False], [Element_1, Dimension_2, True], ...]`) wird für die TM1-Dimensionszuordnungsinformationen verwendet. Jede Liste mit drei Werten gibt Informationen zur Dimensionszuordnung an. Der dritte Wert ist boolesch und gibt an, ob ein Element einer Dimension ausgewählt wird. Beispiel: `"[Field_1, Dimension_1, False]"` bedeutet, dass `Field_1` zu `Dimension_1` zugeordnet ist; `"[Element_1, Dimension_2, True]"` bedeutet, dass `Element_1` für `Dimension_2` ausgewählt ist. Beispiel 2: Die zweite Liste (`[[Field_2, ExistMeasureElement, False], [Field_3, NewMeasureElement, True], ...]`) wird für die Zuordnungsinformationen des TM1-Maßdimensionselements verwendet. Jede Liste mit 3 Werten gibt Informationen zur Maßelementzuordnung an. Der dritte Wert ist boolesch und gibt an, ob ein neues Element erstellt werden muss. `"[Field_2, ExistMeasureElement, False]"` bedeutet, dass `Field_2` zu `ExistMeasureElement` zugeordnet ist; `"[Field_3, NewMeasureElement, True]"` bedeutet, dass `NewMeasureElement` der in `selected_measure` ausgewählten Maßdimension entsprechen muss und dass `Field_3` zugeordnet ist.
`selected_measure`	*string*	Geben Sie die Maßdimension an. Beispiel: `setPropertyValue("selected_measure", "Measures")`
`connection_type`	`AdminServer` `TM1Server`	Gibt den Verbindungstyp an. Der Standardwert ist `AdminServer`.
`admin_host`	*string*	URL für den Hostnamen der REST-API. Erforderlich, wenn `connection_type` auf `AdminServer` gesetzt ist.

Tabelle 254. Eigenschaften des Knotens "tm1odataexport" (Forts.)		
Eigenschaften des Knotens `tm1odataexport`	**Datentyp**	**Eigenschaftsbeschreibung**
`server_name`	*string*	Name des TM1-Servers, der aus `admin_host` ausgewählt wird. Erforderlich, wenn `connection_type` auf `AdminServer` gesetzt ist.
`server_url`	*string*	Die URL für die REST-API von TM1 Server. Erforderlich, wenn `connection_type` auf `TM1Server` gesetzt ist.

Eigenschaften des Knotens "tm1export" (nicht mehr unterstützt)

Der IBM Cognos TM1-Exportknoten exportiert Daten in einem Format, das von Cognos TM1-Datenbanken gelesen werden kann.

Anmerkung: Dieser Knoten wird seit Modeler 18.0 nicht mehr unterstützt. Der Name des Ersatzknotenscripts ist *tm1odataexport*.

Tabelle 255. Eigenschaften des Knotens "tm1export"		
Eigenschaften des Knotens `tm1export`	**Datentyp**	**Eigenschaftsbeschreibung**
`pm_host`	*string*	**Anmerkung:** Nur für Version 16.0 und 17.0 Der Hostname. Zum Beispiel: `TM1_export.setPropertyValue("pm_host", 'http://9.191.86.82:9510/pmhub/pm')`
`tm1_connection`	*["feld","feld", ... ,"feld"]*	**Anmerkung:** Nur für Version 16.0 und 17.0 Eine Listeneigenschaft mit den Verbindungsdetails für den TM1-Server. Format: `[ "TM1_Server_Name", "tm1_ username", "tm1_password"]` Zum Beispiel: `TM1_export.setPropertyValue("tm1_connection", ['Planning Sample', "admin" "apple"])`
`selected_cube`	*Feld*	Der Name des Cubes, in den Sie Daten exportieren. Zum Beispiel: `TM1_export.setPropertyValue("selected_cube", "plan_BudgetPlan")`

Tabelle 255. Eigenschaften des Knotens "tm1export" (Forts.)

Eigenschaften des Knotens `tm1export`	**Datentyp**	**Eigenschaftsbeschreibung**
`spssfield_tm1element_mapping`	*Liste*	Das zuzuordnende TM1-Element muss Teil der Spaltendimension für die ausgewählte Cube-Ansicht sein. Format: `[[[Field_1, Dimension_1, False], [Element_1, Dimension_2, True], ...], [[Field_2, ExistMeasureElement, False], [Field_3, NewMeasureElement, True], ...]]` Es gibt 2 Listen zum Beschreiben der Zuordnungsinformationen. Die Zuordnung eines Blattelements zu einer Dimension entspricht Beispiel 2 unten: Beispiel 1: Die erste Liste (`[[Field_1, Dimension_1, False], [Element_1, Dimension_2, True], ...]`) wird für die Informationen zur TM1-Dimensionszuordnung verwendet. Jede Liste mit drei Werten gibt Informationen zur Dimensionszuordnung an. Der dritte Wert ist boolesch und gibt an, ob ein Element einer Dimension ausgewählt wird. Beispiel: "`[Field_1, Dimension_1, False]`" bedeutet, dass `Field_1` der Dimension `Dimension_1` zugeordnet wird; "`[Element_1, Dimension_2, True]`" bedeutet, dass `Element_1` für `Dimension_2` ausgewählt wird. Beispiel 2: Die zweite Liste (`[[Field_2, ExistMeasureElement, False], [Field_3, NewMeasureElement, True], ...]`) wird für die Zuordnungsinformationen des TM1-Maßdimensionselements verwendet. Jede Liste mit 3 Werten gibt Informationen zur Maßelementzuordnung an. Der dritte Wert ist boolesch und gibt an, ob ein neues Element erstellt werden muss. "`[Field_2, ExistMeasureElement, False]`" bedeutet, dass `Field_2` zu `ExistMeasureElement` zugeordnet ist; "`[Field_3, NewMeasureElement, True]`" bedeutet, dass `NewMeasureElement` der in `selected_measure` ausgewählten Maßdimension entsprechen muss und dass `Field_3` zugeordnet ist.

Tabelle 255. Eigenschaften des Knotens "tm1export" (Forts.)		
Eigenschaften des Knotens `tm1export`	**Datentyp**	**Eigenschaftsbeschreibung**
`selected_measure`	*string*	Geben Sie die Maßdimension an. Beispiel: `setPropertyValue("selected_measure", "Measures")`

Eigenschaften von "xmlexportnode"

Der XML-Exportknoten gibt Daten an eine Datei im XML-Format aus. Optional können Sie einen XML-Quellenknoten erstellen, um die exportierten Daten wieder in der Stream einzulesen.

Beispiel

```
stream = modeler.script.stream()
xmlexportnode = stream.createAt("xmlexport", "XML Export", 200, 200)
xmlexportnode.setPropertyValue("full_filename", "c:/export/data.xml")
xmlexportnode.setPropertyValue("map", [["/catalog/book/genre", "genre"], ["/
catalog/book/title", "title"]])
```

Tabelle 256. Eigenschaften von "xmlexportnode"		
Eigenschaften von `xmlexportnode`	**Datenart**	**Eigenschaftsbeschreibung**
`full_filename`	*string*	(erforderlich) Vollständiger Pfad und Dateiname der XML-Exportdatei.
`use_xml_schema`	*Flag*	Legt fest, ob ein XML-Schema (XSD- oder DTD-Datei) für die Steuerung der Struktur der exportierten Daten verwendet wird.
`full_schema_filename`	*string*	Vollständiger Pfad und Dateiname der zu verwendenden XSD- oder DTD-Datei. Erforderlich, wenn `use_xml_schema` auf "wahr" gesetzt ist.
`generate_import`	*Flag*	Generiert einen XML-Quellenknoten, der die exportierten Daten wieder in den Stream einliest.
`records`	*string*	XPath-Ausdruck, der die Datensatzgrenze angibt.
`map`	*string*	Ordnet den Feldnamen der XML-Struktur zu.

Kapitel 18. Eigenschaften von IBM SPSS Statistics-Knoten

Eigenschaften von "statisticsimportnode"

Der Statistikdateiknoten liest Daten aus dem Dateiformat *.sav* oder *.zsav* ein, das von IBM SPSS Statistics verwendet wird, sowie in IBM SPSS Modeler gespeicherte Cachedateien, die ebenfalls dasselbe Format verwenden.

Beispiel

```
stream = modeler.script.stream()
statisticsimportnode = stream.createAt("statisticsimport", "SAV Import",
200, 200)
statisticsimportnode.setPropertyValue("full_filename", "C:/data/drug1n.sav")
statisticsimportnode.setPropertyValue("import_names", True)
statisticsimportnode.setPropertyValue("import_data", True)
```

Tabelle 257. Eigenschaften von "statisticsimportnode"

Eigenschaften von `statisticsimportnode`	**Datenart**	**Eigenschaftsbeschreibung**
`full_filename`	*string*	Der vollständige Dateiname mit Pfad.
`password`	*string*	Das Kennwort. Der Parameter `password` muss vor dem Parameter `file_encrypted` festgelegt werden.
`file_encrypted`	*Flag*	Ob die Datei kennwortgeschützt ist.

`import_names`	`NamesAndLabels`
	
	`LabelsAsNames`
		Methode für die Behandlung von Variablennamen und -beschriftungen.

`import_data`	`DataAndLabels`
	
	`LabelsAsData`
		Methode für die Behandlung von Werten und Beschriftungen.

`use_field_format_for_storage`	*boolesch*	Gibt an, ob IBM SPSS Statistics-Feldformatinformationen beim Import verwendet werden.

Eigenschaften von "statisticstransformnode"

Der Statistics-Transformationsknoten führt eine Auswahl von IBM SPSS Statistics-Syntaxbefehlen für Datenquellen in IBM SPSS Modeler aus. Dieser Knoten erfordert eine lizenzierte Kopie vonIBM SPSS Statistics.

Beispiel

```
stream = modeler.script.stream()
statisticstransformnode = stream.createAt("statisticstransform", "Trans
```

```
form", 200, 200)
statisticstransformnode.setPropertyValue("syntax", "COMPUTE NewVar = Na +
K.")
statisticstransformnode.setKeyedPropertyValue("new_name", "NewVar", "Mixed
Drugs")
statisticstransformnode.setPropertyValue("check_before_saving", True)
```

Tabelle 258. Eigenschaften von "statisticstransformnode"

Eigenschaften von `statisticstransform-node`	Datentyp	Eigenschaftsbeschreibung
`syntax`	*string*	
`check_before_saving`	*Flag*	Überprüft die eingegebene Syntax vor dem Speichern der Einträge. Zeigt eine Fehlernachricht an, wenn die Syntax ungültig ist.
`default_include`	*Flag*	Weitere Informationen finden Sie im Thema „Eigenschaften von "filternode"" auf Seite 174.
`include`	*Flag*	Weitere Informationen finden Sie im Thema „Eigenschaften von "filternode"" auf Seite 174.
`new_name`	*string*	Weitere Informationen finden Sie im Thema „Eigenschaften von "filternode"" auf Seite 174.

Eigenschaften von "statisticsmodelnode"

Mithilfe des Statistics-Modellknotens können Sie Ihre Daten analysieren und bearbeiten, indem Sie IBM SPSS Statistics-Prozeduren ausführen, die PMML erzeugen. Dieser Knoten erfordert eine lizenzierte Kopie vonIBM SPSS Statistics.

Beispiel

```
stream = modeler.script.stream()
statisticsmodelnode = stream.createAt("statisticsmodel", "Model", 200, 200)
statisticsmodelnode.setPropertyValue("syntax", "COMPUTE NewVar = Na + K.")
statisticsmodelnode.setKeyedPropertyValue("new_name", "NewVar", "Mixed
Drugs")
```

Eigenschaften von `statisticsmodelnode`	Datentyp	Eigenschaftsbeschreibung
`syntax`	*string*	
`default_include`	*Flag*	Weitere Informationen finden Sie im Thema „Eigenschaften von "filternode"" auf Seite 174.
`include`	*Flag*	Weitere Informationen finden Sie im Thema „Eigenschaften von "filternode"" auf Seite 174.

Eigenschaften von statisticsmodelnode	Datentyp	Eigenschaftsbeschreibung
new_name	*string*	Weitere Informationen finden Sie im Thema „Eigenschaften von "filternode"" auf Seite 174.

Eigenschaften von "statisticsoutputnode"

Mit dem Statistics-Ausgabeknoten können Sie eine IBM SPSS Statistics-Prozedur aufrufen, um Ihre IBM SPSS Modeler-Daten zu analysieren. Es stehen zahlreiche IBM SPSS Statistics-Analyseprozeduren zur Verfügung. Für diesen Knoten ist eine lizenzierte Kopie von IBM SPSS Statistics erforderlich.

Beispiel

```
stream = modeler.script.stream()
statisticsoutputnode = stream.createAt("statisticsoutput", "Output", 200,
200)
statisticsoutputnode.setPropertyValue("syntax", "SORT CASES BY Age(A) Sex(A)
BP(A) Cholesterol(A)")
statisticsoutputnode.setPropertyValue("use_output_name", False)
statisticsoutputnode.setPropertyValue("output_mode", "File")
statisticsoutputnode.setPropertyValue("full_filename", "Cases by Age, Sex
and Medical History")
statisticsoutputnode.setPropertyValue("file_type", "HTML")
```

Tabelle 259. Eigenschaften von "statisticsoutputnode"

Eigenschaften von statisticsoutputnode	Datenart	Eigenschaftsbeschreibung
mode	Dialog Syntax	Wählt die Option "IBM SPSS Statistics-Dialogfeld" oder Syntaxeditor aus
syntax	*string*	
use_output_name	*Flag*	
output_name	*string*	
output_mode	Screen File	
full_filename	*string*	
file_type	HTML SPV SPW	

Eigenschaften von "statisticsexportnode

Der Statistikexportknoten gibt Daten im IBM SPSS Statistics-Format *.sav* oder *.zsav* aus. Die Dateien `.sav` oder `.zsav` können von IBM SPSS Statistics Base und anderen Produkten gelesen werden. Dieses Format wird auch für Cachedateien in IBM SPSS Modeler verwendet.

Beispiel

```
stream = modeler.script.stream()
statisticsexportnode = stream.createAt("statisticsexport", "Export", 200,
200)
statisticsexportnode.setPropertyValue("full_filename", "c:/output/SPSS_Sta
tistics_out.sav")
statisticsexportnode.setPropertyValue("field_names", "Names")
statisticsexportnode.setPropertyValue("launch_application", True)
statisticsexportnode.setPropertyValue("generate_import", True)
```

Tabelle 260. Eigenschaften von "statisticsexportnode"

Eigenschaften von `statisticsexportnode`	**Datentyp**	**Eigenschaftsbeschreibung**
`full_filename`	*string*	

`file_type`	`sav`
	
	`zsav`
		Datei im *sav*- oder *zsav*-Format speichern. Beispiel:
		
		`statisticsexportnode.setPropertyValue("file_type","sav")`

`encrypt_file`	*Flag*	Ob die Datei kennwortgeschützt ist.
`password`	*string*	Das Kennwort.
`launch_application`	*Flag*	

`export_names`	`NamesAndLabels`
	
	`NamesAsLabels`
		Dient der Zuordnung von Feldnamen von IBM SPSS Modeler zu IBM SPSS Statistics- oder SAS-Variablennamen nach dem Export.

`generate_import`	*Flag*	

Kapitel 19. Eigenschaften von Python-Knoten

Eigenschaften von "gmm"

Ein gaußsches Mischverteilungsmodell ist ein probabilistisches Modell, das voraussetzt, dass alle Datenpunkte aus einer Mischung einer endlichen Anzahl von gaußschen Verteilungen mit unbekannten Parametern generiert werden. Mischverteilungsmodelle kann man sich als das Verallgemeinern von K-Means-Clustering zum Aufnehmen von Informationen zur Kovarianzstruktur der Daten sowie der Mittelpunkte der latenten gaußschen Verteilungen vorstellen. Der Knoten des gaußschen Mischverteilungsmodells in SPSS Modeler stellt die zentralen Funktionen und häufig verwendeten Parameter der gaußschen Mischverteilungsbibliothek bereit. Der Knoten ist in Python implementiert.

Tabelle 261. Eigenschaften von "gmm"

Eigenschaften von gmm	Datentyp	Eigenschaftsbeschreibung
use_partition	*Boolesch*	Setzen Sie diese Eigenschaft auf True oder False, um anzugeben, ob partitionierte Daten verwendet werden sollen. Der Standardwert ist False.
covariance_type	*string*	Geben Sie Full, Tied, Diag oder Spherical als Kovarianztyp an.
number_component	*Integer*	Geben Sie eine Ganzzahl für die Anzahl der Mischverteilungskomponenten an. Der Minimalwert ist 1. Der Standardwert ist 2.
component_lable	*Boolesch*	Geben Sie True an, um die Clusterbeschriftung auf eine Zeichenfolge zu setzen. Geben Sie False an, um die Clusterbeschriftung auf eine Zahl zu setzen. Der Standardwert ist False.
label_prefix	*string*	Wenn Sie eine Zeichenfolge als Clusterbeschriftung verwenden, können Sie ein Präfix angeben.
enable_random_seed	*Boolesch*	Geben Sie True an, wenn Sie einen Startwert für Zufallszahlen verwenden wollen. Der Standardwert ist False.
random_seed	*Integer*	Wenn Sie einen Startwert für Zufallszahlen verwenden, geben Sie eine Ganzzahl an, die für das Generieren von Zufallsstichproben verwendet werden soll.
tol	*double*	Geben Sie den Konvergenzschwellenwert an. Der Standardwert ist 0.000.1.
max_iter	*Integer*	Geben Sie die maximale Anzahl auszuführender Iterationen an. Der Standardwert ist 100.

Tabelle 261. Eigenschaften von "gmm" (Forts.)		
Eigenschaften von gmm	**Datentyp**	**Eigenschaftsbeschreibung**
init_params	*string*	Legen Sie den zu verwendenden Initialisierungsparameter fest. Die Optionen sind Kmeans oder Random.
warm_start	*Boolesch*	Geben Sie True an, um die Lösung der letzten Anpassung als Initialisierung für den nächsten Aufruf der Anpassung zu verwenden. Der Standardwert ist False.

Eigenschaften von "hdbscannode"

HDBSCAN© (Hierarchical Density-Based Spatial Clustering) verwendet nicht überwachtes Lernen zum Suchen von Clustern (oder dicht besetzten Bereichen) eines Datasets. Der HDBSCAN-Knoten in SPSS Modeler stellt die zentralen Funktionen und häufig verwendeten Parameter der HDBSCAN-Bibliothek bereit. Der Knoten wird in Python implementiert und Sie können ihn verwenden, um Ihr Dataset in verschiedene Gruppen aufzuteilen, wenn Sie anfangs noch nicht wissen, was diese Gruppen enthalten.

Tabelle 262. Eigenschaften von "hdbscannode"		
Eigenschaften von hdbscannode	**Datentyp**	**Eigenschaftsbeschreibung**
inputs	*Feld*	Eingabefelder für Clustering.
useHPO	*Boolesch*	Geben Sie true oder false an, um Hyper-Parameter Optimization (HPO) auf der Basis von Rbfopt zu aktivieren oder zu inaktivieren, das automatisch die optimale Kombination von Parametern erkennt, damit das Modell die gewünschte Fehlerrate (oder eine niedrigere Rate) für die Stichproben erzielt. Der Standardwert ist false.
min_cluster_size	*Integer*	Die Mindestgröße von Clustern. Geben Sie eine Ganzzahl an. Der Standardwert ist 5.
min_samples	*Integer*	Die Anzahl der Stichproben in einer Nachbarschaft für einen Punkt, der als zentraler Punkt betrachtet werden soll. Geben Sie eine Ganzzahl an. Wenn die Option auf 0 gesetzt ist, wird min_cluster_size verwendet. Der Standardwert ist 0.
algorithm	*string*	Geben Sie an, welcher Algorithmus verwendet werden soll: best, generic, prims_kdtree, prims_balltree, boruvka_kdtree oder boruvka_balltree. Der Standardwert ist best.

Tabelle 262. Eigenschaften von "hdbscannode" (Forts.)

Eigenschaften von hdbscannode	Datentyp	Eigenschaftsbeschreibung
metric	*string*	Geben Sie an, welche Metrik beim Berechnen des Abstands zwischen Instanzen in einem Funktionsarray verwendet werden soll: euclidean, cityblock, L1, L2, manhattan, braycurtis, canberra, chebyshev, correlation, minkowski oder sqeuclidean. Der Standardwert ist euclidean.
useStringLabel	*Boolesch*	Geben Sie true an, um eine Zeichenfolge als Clusterbeschriftung zu verwenden, oder false, um eine Zahl als Clusterbeschriftung zu verwenden. Der Standardwert ist false.
stringLabelPrefix	*string*	Wenn der Parameter useStringLabel auf true gesetzt ist, geben Sie einen Wert für das Zeichenfolgenbeschriftungspräfix an. Das Standardpräfix ist cluster.
approx_min_span_tree	*Boolesch*	Geben Sie true an, um eine näherungsweise berechneten Spanning Tree zu akzeptieren, oder false, wenn Sie bereit sind, für die Fehlerfreiheit auf Geschwindigkeit zu verzichten. Der Standardwert ist true.
cluster_selection_method	*string*	Geben Sie die Methode an, die verwendet werden soll, um Cluster im komprimierten Baum auszuwählen: eom oder leaf. Der Standardwert ist eom (Excess of Mass-Algorithmus).
allow_single_cluster	*Boolesch*	Geben Sie true an, wenn Sie Ergebnisse mit einzelnen Clustern zulassen wollen. Der Standardwert ist false.
p_value	*DOUBLE*	Geben Sie den p value an, der verwendet werden soll, wenn Sie minkowski als Metrik verwenden. Der Standardwert ist 1.5.
leaf_size	*Integer*	Wenn Sie einen Bereichsbaumalgorithmus (boruvka_kdtree oder boruvka_balltree) verwenden, geben Sie die Anzahl der Punkte in einem Blattknoten des Baums an. Der Standardwert ist 40.
outputValidity	*Boolesch*	Geben Sie true oder false an, um zu steuern, ob das Gültigkeitsindexdiagramm in die Modellausgabe eingeschlossen wird.
outputCondensed	*Boolesch*	Geben Sie true oder false an, um zu steuern, ob das komprimierte Baumstrukturdiagramm in die Modellausgabe eingeschlossen wird.

Tabelle 262. Eigenschaften von "hdbscannode" (Forts.)		
Eigenschaften von hdbscannode	**Datentyp**	**Eigenschaftsbeschreibung**
outputSingleLinkage	*Boolesch*	Geben Sie true oder false an, um zu steuern, ob das Einzelverknüpfungsbaum-diagramm in die Modellausgabe eingeschlossen wird.
outputMinSpan	*Boolesch*	Geben Sie true oder false an, um zu steuern, ob das Min. Spanning Tree-Diagramm in die Modellausgabe eingeschlossen wird.
is_split		Hinzugefügt in Version 18.2.1.1.

Eigenschaften von "kdemodel"

Kernel Density Estimation (KDE) verwendet die Kugelbaum- oder KD-Baumalgorithmen für effiziente Abfragen und kombiniert Konzepte von unbeaufsichtigtem Lernen, Funktionsentwicklung und Datenmodellierung. Auf Nachbarn basierte Ansätze wie KDE sind einige der gängigsten und nützlichsten Dichteschätzungsverfahren. Die KDE-Modellierungs- und KDE-Simulationsknoten in SPSS Modeler stellen die zentralen Funktionen und häufig verwendeten KDE-Bibliotheksparametern bereit. Die Knoten sind in Python implementiert.

Tabelle 263. Eigenschaften von "kdemodel"		
Eigenschaften von kdemodel	**Datentyp**	**Eigenschaftsbeschreibung**
bandwidth	*DOUBLE*	Der Standardwert ist 1.
kernel	*string*	Der zu verwendende Kern: gaussian, tophat, epanechnikov, exponential, linear oder cosine. Der Standardwert ist gaussian.
algorithm	*string*	Der zu verwendende Baumalgorithmus: kd_tree, ball_tree oder auto. Der Standardwert ist auto.
metric	*string*	Die beim Berechnen des Abstands zu verwendende Metrik. Beim Algorithmus kd_tree können Sie aus folgenden Metriken wählen: Euclidean, Chebyshev, Cityblock, Minkowski, Manhattan, Infinity, P, L2 oder L1. Beim Algorithmus ball_tree können Sie aus folgenden Metriken wählen: Euclidian, Braycurtis, Chebyshev, Canberra, Cityblock, Dice, Hamming, Infinity, Jaccard, L1, L2, Minkowski, Matching, Manhattan, P, Rogersanimoto, Russellrao, Sokalmichener, Sokalsneath oder Kulsinski. Der Standardwert ist Euclidean.

Tabelle 263. Eigenschaften von "kdemodel" (Forts.)		
Eigenschaften von kdemodel	**Datentyp**	**Eigenschaftsbeschreibung**
atol	*Float*	Die gewünschte absolute Toleranz des Ergebnisses. Eine größere Toleranz führt in der Regel zu schnellerer Ausführung. Der Standardwert ist 0.0.
rtol	*Float*	Die gewünschte relative Toleranz des Ergebnisses. Eine größere Toleranz führt in der Regel zu schnellerer Ausführung. Der Standardwert ist 1E-8.
breadthFirst In breadth_first umbenannt ab Version 18.2.1.1.	*Boolesch*	Setzen Sie diese Eigenschaft auf True, um den Ansatz "Breite zuerst" zu verwenden. Setzen Sie diese Eigenschaft auf False, um den Ansatz "Tiefe zuerst" zu verwenden. Der Standardwert ist True.
LeafSize In leaf_size umbenannt ab Version 18.2.1.1.	*Integer*	Die Blattgröße der zugrunde liegenden Baums. Der Standardwert ist 40. Eine Änderung dieses Werts kann sich erheblich auf die Leistung auswirken.
pValue	*DOUBLE*	Geben Sie den P-Wert an, der verwendet werden soll, wenn Sie Minkowski als Metrik verwenden. Der Standardwert ist 1.5.
custom_name		
default_node_name		
use_HPO		

kdeexport, Eigenschaften

Kernel Density Estimation (KDE) verwendet die Kugelbaum- oder KD-Baumalgorithmen für effiziente Abfragen und kombiniert Konzepte von unbeaufsichtigtem Lernen, Funktionsentwicklung und Datenmodellierung. Auf Nachbarn basierte Ansätze wie KDE sind einige der gängigsten und nützlichsten Dichteschätzungsverfahren. Die KDE-Modellierungs- und KDE-Simulationsknoten in SPSS Modeler stellen die zentralen Funktionen und häufig verwendeten KDE-Bibliotheksparametern bereit. Die Knoten sind in Python implementiert.

Tabelle 264. kdeexport, Eigenschaften		
Eigenschaften von kdeexport	**Datentyp**	**Eigenschaftsbeschreibung**
bandwidth	*DOUBLE*	Der Standardwert ist 1.
kernel	*string*	Der zu verwendende Kern: gaussian oder tophat. Der Standardwert ist gaussian.
algorithm	*string*	Der zu verwendende Baumalgorithmus: kd_tree, ball_tree oder auto. Der Standardwert ist auto.

Tabelle 264. kdeexport, Eigenschaften (Forts.)		
Eigenschaften von kdeexport	**Datentyp**	**Eigenschaftsbeschreibung**
metric	*string*	Die beim Berechnen des Abstands zu verwendende Metrik. Beim Algorithmus kd_tree können Sie aus folgenden Metriken wählen: Euclidean, Chebyshev, Cityblock, Minkowski, Manhattan, Infinity, P, L2 oder L1. Beim Algorithmus ball_tree können Sie aus folgenden Metriken wählen: Euclidian, Braycurtis, Chebyshev, Canberra, Cityblock, Dice, Hamming, Infinity, Jaccard, L1, L2, Minkowski, Matching, Manhattan, P, Rogersanimoto, Russellrao, Sokalmichener, Sokalsneath oder Kulsinski. Der Standardwert ist Euclidean.
atol	*Float*	Die gewünschte absolute Toleranz des Ergebnisses. Eine größere Toleranz führt in der Regel zu schnellerer Ausführung. Der Standardwert ist 0.0.
rtol	*Float*	Die gewünschte relative Toleranz des Ergebnisses. Eine größere Toleranz führt in der Regel zu schnellerer Ausführung. Der Standardwert ist 1E-8.
breadthFirst	*Boolesch*	Setzen Sie diese Eigenschaft auf True, um den Ansatz "Breite zuerst" zu verwenden. Setzen Sie diese Eigenschaft auf False, um den Ansatz "Tiefe zuerst" zu verwenden. Der Standardwert ist True.
LeafSize	*Integer*	Die Blattgröße der zugrunde liegenden Baums. Der Standardwert ist 40. Eine Änderung dieses Werts kann sich erheblich auf die Leistung auswirken.
pValue	*DOUBLE*	Geben Sie den P-Wert an, der verwendet werden soll, wenn Sie Minkowski als Metrik verwenden. Der Standardwert ist 1.5.

Eigenschaften von "gmm"

Ein gaußsches Mischverteilungsmodell ist ein probabilistisches Modell, das voraussetzt, dass alle Datenpunkte aus einer Mischung einer endlichen Anzahl von gaußschen Verteilungen mit unbekannten Parametern generiert werden. Mischverteilungsmodelle kann man sich als das Verallgemeinern von K-Means-Clustering zum Aufnehmen von Informationen zur Kovarianzstruktur der Daten sowie der Mittelpunkte der latenten gaußschen Verteilungen vorstellen. Der Knoten des gaußschen Mischverteilungsmodells in SPSS Modeler stellt die zentralen Funktionen und häufig verwendeten Parameter der gaußschen Mischverteilungsbibliothek bereit. Der Knoten ist in Python implementiert.

Tabelle 265. Eigenschaften von "gmm"		
Eigenschaften von gmm	**Datentyp**	**Eigenschaftsbeschreibung**
use_partition	*Boolesch*	Setzen Sie diese Eigenschaft auf True oder False, um anzugeben, ob partitionierte Daten verwendet werden sollen. Der Standardwert ist False.
covariance_type	*string*	Geben Sie Full, Tied, Diag oder Spherical als Kovarianztyp an.
number_component	*Integer*	Geben Sie eine Ganzzahl für die Anzahl der Mischverteilungskomponenten an. Der Minimalwert ist 1. Der Standardwert ist 2.
component_lable	*Boolesch*	Geben Sie True an, um die Clusterbeschriftung auf eine Zeichenfolge zu setzen. Geben Sie False an, um die Clusterbeschriftung auf eine Zahl zu setzen. Der Standardwert ist False.
label_prefix	*string*	Wenn Sie eine Zeichenfolge als Clusterbeschriftung verwenden, können Sie ein Präfix angeben.
enable_random_seed	*Boolesch*	Geben Sie True an, wenn Sie einen Startwert für Zufallszahlen verwenden wollen. Der Standardwert ist False.
random_seed	*Integer*	Wenn Sie einen Startwert für Zufallszahlen verwenden, geben Sie eine Ganzzahl an, die für das Generieren von Zufallsstichproben verwendet werden soll.
tol	*double*	Geben Sie den Konvergenzschwellenwert an. Der Standardwert ist 0.000.1.
max_iter	*Integer*	Geben Sie die maximale Anzahl auszuführender Iterationen an. Der Standardwert ist 100.
init_params	*string*	Legen Sie den zu verwendenden Initialisierungsparameter fest. Die Optionen sind Kmeans oder Random.
warm_start	*Boolesch*	Geben Sie True an, um die Lösung der letzten Anpassung als Initialisierung für den nächsten Aufruf der Anpassung zu verwenden. Der Standardwert ist False.

Eigenschaften von "ocsvmnode"

Der Knoten "One-Class SVM" verwendet einen nicht überwachten Lernalgorithmus. Der Knoten kann für die Erkennung von Neuheiten verwendet werden. Er erkennt die flexible Grenze eines angegebenen Stichprobensets und klassifiziert neue Punkte danach, ob sie zu diesem Set gehören. Dieser One-Class SVM-Modellierungsknoten in SPSS Modeler ist in Python implementiert und erfordert die Bibliothek scikit-learn© Python.

Tabelle 266. Eigenschaften von "ocsvmnode"

Eigenschaften von "ocsvmnode"	Datentyp	Eigenschaftsbeschreibung
role_use In custom_fields umbenannt ab Version 18.2.1.1.	*string*	Geben Sie predefined an, um vordefinierte Rollen zu verwenden, oder custom, um benutzerdefinierte Feldzuweisungen zu verwenden. Der Standardwert ist "predefined".
splits	*Feld*	Liste der Feldnamen für die Aufteilung.
use_partition	*boolesch*	Geben Sie true oder false an. Der Standardwert ist true. Wenn die Option auf true gesetzt ist, werden beim Erstellen des Modells nur Trainingsdaten verwendet.
mode_type	*string*	Der Modus. Mögliche Werte sind simple oder expert. Wenn simple angegeben ist, werden alle Parameter auf der Registerkarte "Experten" inaktiviert.
stopping_criteria	*string*	Eine Zeichenfolge in wissenschaftlicher Notation. Mögliche Werte sind 1.0E-1, 1.0E-2, 1.0E-3, 1.0E-4, 1.0E-5 oder 1.0E-6. Der Standardwert ist 1.0E-3.
precision	*Float*	Die Regressionsgenauigkeit (Nu). An die Bruchzahl aus Trainingsfehlern und Unterstützungsvektoren gebunden. Geben Sie eine Zahl größer als 0 und kleiner-gleich 1.0 an. Der Standardwert ist 0.1.
kernel	*string*	Der Kerntyp, der im Algorithmus verwendet werden soll. Mögliche Werte sind linear, poly, rbf, sigmoid oder precomputed. Der Standardwert ist rbf.
enable_gamma	*boolesch*	Aktiviert den Parameter gamma. Geben Sie true oder false an. Der Standardwert ist true.
gamma	*Float*	Dieser Parameter ist nur für die Kerne rbf, poly und sigmoid aktiviert. Wenn der Parameter enable_gamma auf false gesetzt ist, wird dieser Parameter auf auto gesetzt. Wenn true festgelegt ist, lautet der Standardwert 0.1.
coef0	*Float*	Unabhängiger Term in der Kernfunktion. Dieser Parameter ist nur für die Kerne poly und sigmoid aktiviert. Der Standardwert ist0.0.
degree	*Integer*	Grad der polynomialen Kernfunktion. Dieser Parameter ist nur für den Kern poly aktiviert. Geben Sie eine beliebige Ganzzahl an. Der Standardwert ist 3.

Tabelle 266. Eigenschaften von "ocsvmnode" (Forts.)

Eigenschaften von "ocsvmnode"	Datentyp	Eigenschaftsbeschreibung
shrinking	*boolesch*	Gibt an, ob die Shrinking-Heuristik verwendet werden soll. Geben Sie true oder false an. Der Standardwert ist false.
enable_cache_size	*boolesch*	Aktiviert den Parameter cache_size. Geben Sie true oder false an. Der Standardwert ist false.
cache_size	*Float*	Die Größe des Kernel-Cache in MB. Der Standardwert ist 200.
pc_type	*string*	Der Typ der Parallelkoordinatengrafik. Mögliche Optionen sind independent oder general.
lines_amount	*Integer*	Maximale Anzahl der Zeilen, die in die Grafik eingeschlossen werden. Geben Sie eine Ganzzahl zwischen 1 und 1000an.
lines_fields_custom	*boolesch*	Aktiviert den Parameter lines_fields, mit dem Sie benutzerdefinierte Felder angeben können, die in der Grafikausgabe angezeigt werden sollen. Wenn die Option auf false gesetzt ist, werden alle Felder angezeigt. Wenn true festgelegt ist, werden nur die mit dem Parameter "lines_fields" angegebenen Felder angezeigt. Aufgrund von Leistungsaspekten werden maximal 20 Felder angezeigt.
lines_fields	*Feld*	Liste der Feldnamen, die als vertikale Achsen in die Grafik eingeschlossen werden sollen.
enable_graphic	*boolesch*	Geben Sie true oder false an. Aktiviert die Grafikausgabe (inaktivieren Sie diese Option, wenn Sie Zeit sparen und die Größe der Datenstromdatei reduzieren wollen).
enable_hpo	*boolesch*	Geben Sie true oder false an, um die HPO-Optionen zu aktivieren oder zu inaktivieren. Bei Angabe von true wird Rbfopt angewendet, um das beste One-Class-SVM-Modell automatisch zu bestimmen, das den vom Benutzer über den Parameter target_objval angegebenen Zielwert erreicht.
target_objval	*Float*	Der Zielfunktionswert (Fehlerrate des Modells für die Stichproben), der erreicht werden soll (z. B. der Wert des unbekannten Optimums). Setzen Sie diesen Parameter auf den entsprechenden Wert, wenn das Optimum unbekannt ist (z. B. 0.01).

Tabelle 266. Eigenschaften von "ocsvmnode" (Forts.)		
Eigenschaften von "ocsvmnode"	**Datentyp**	**Eigenschaftsbeschreibung**
max_iterations	*Integer*	Maximale Anzahl Iterationen zum Testen des Modells. Der Standardwert ist 1000.
max_evaluations	*Integer*	Maximale Anzahl Funktionsauswertungen zum Testen des Modells, wobei der Fokus weniger auf der Geschwindigkeit, sondern eher auf der Genauigkeit liegt. Der Standardwert ist 300.

rfnode, Eigenschaften

Der Random Forest-Knoten verwendet eine erweiterte Implementierung eines Bagging-Algorithmus mit einem Baummodell als Basismodell. Dieser Random Forest-Modellierungsknoten in SPSS Modeler ist in Python implementiert und macht die Python-Bibliothek scikit-learn© erforderlich.

Tabelle 267. rfnode, Eigenschaften		
Eigenschaften von rfnode	**Datentyp**	**Eigenschaftsbeschreibung**
role_use	*string*	Geben Sie predefined an, um vordefinierte Rollen zu verwenden, oder custom, um benutzerdefinierte Feldzuweisungen zu verwenden. Der Standardwert ist "predefined".
inputs	*Feld*	Liste der Feldnamen für die Eingabe.
splits	*Feld*	Liste der Feldnamen für die Aufteilung.
n_estimators	*Integer*	Zu erstellende Anzahl Bäume. Der Standardwert ist 10.
specify_max_depth	*boolesch*	Geben Sie die benutzerdefinierte maximale Tiefe an. Bei Angabe von false werden Knoten erweitert, bis alle Blätter rein sind oder weniger als min_samples_split Stichproben haben. Der Standardwert ist false.
max_depth	*Integer*	Die maximale Tiefe des Baums. Der Standardwert ist 10.
min_samples_leaf	*Integer*	Mindestgröße der Blattknoten. Der Standardwert ist 1.

Tabelle 267. rfnode, Eigenschaften (Forts.)

Eigenschaften von rfnode	Datentyp	Eigenschaftsbeschreibung
max_features	*string*	Die Anzahl der Merkmale, die bei der Suche nach der besten Aufteilung berücksichtigt werden sollen: • Bei Angabe von auto gilt max_features=sqrt(n_features) für Klassifikationsmerkmale und max_features=sqrt(n_features) für Regression. • Bei Angabe von sqrt gilt max_features=sqrt(n_features). • Bei Angabe von log2 gilt max_features=log2 (n_features). Der Standardwert ist auto.
bootstrap	*boolesch*	Bootstrap-Stichproben beim Erstellen von Bäumen verwenden. Der Standardwert ist true.
oob_score	*boolesch*	OOB-Stichproben (Out-of-Bag) zum Schätzen der Generalisierungsgenauigkeit verwenden. Der Standardwert ist false.
extreme	*boolesch*	Extrem randomisierte Bäume verwenden. Der Standardwert ist false.
use_random_seed	*boolesch*	Geben Sie dies an, um replizierte Ergebnisse zu erhalten. Der Standardwert ist false.
random_seed	*Integer*	Der beim Erstellen von Bäumen zu verwendende Startwert für Zufallszahlen. Geben Sie eine beliebige Ganzzahl an.
cache_size	*Float*	Die Größe des Kernel-Cache in MB. Der Standardwert ist 200.
enable_random_seed	*boolesch*	Aktiviert den Parameter random_seed. Geben Sie <codeph>true</codeph> oder <codeph>false</codeph> an. Der Standardwert ist false.
enable_hpo	*boolesch*	Geben Sie true oder false an, um die HPO-Optionen zu aktivieren oder zu inaktivieren. Bei Angabe von true wird Rbfopt angewendet, um das beste Random Forest-Modell automatisch zu bestimmen, das den vom Benutzer über den Parameter target_objval angegebenen Zielwert erreicht.
target_objval	*Float*	Der Zielfunktionswert (Fehlerrate des Modells für die Stichproben), der erreicht werden soll (z. B. der Wert des unbekannten Optimums). Setzen Sie diesen Parameter auf den entsprechenden Wert, wenn das Optimum unbekannt ist (z. B. 0.01).

Tabelle 267. rfnode, Eigenschaften (Forts.)		
Eigenschaften von rfnode	**Datentyp**	**Eigenschaftsbeschreibung**
max_iterations	*Integer*	Maximale Anzahl Iterationen zum Testen des Modells. Der Standardwert ist 1000.
max_evaluations	*Integer*	Maximale Anzahl Funktionsauswertungen zum Testen des Modells, wobei der Fokus weniger auf der Geschwindigkeit, sondern eher auf der Genauigkeit liegt. Der Standardwert ist 300.

Eigenschaften von "smotenode"

Der Knoten "Synthetic Minority Over-sampling Technique" (SMOTE) stellt einen Oversampling-Algorithmus bereit, um unausgewogene Datasets zu verarbeiten. Er stellt eine erweiterte Methode zur Balancierung von Daten bereit. Der SMOTE-Prozessknoten in SPSS Modeler ist in Python implementiert und erfordert die Python-Bibliothek imbalanced-learn©.

Tabelle 268. Eigenschaften von "smotenode"		
Eigenschaften von "smotenode"	**Datentyp**	**Eigenschaftsbeschreibung**
target_field In target umbenannt ab Version 18.2.1.1.	*Feld*	Das Zielfeld.
sample_ratio	*string*	Aktiviert einen benutzerdefinierten Verhältniswert. Die beiden Optionen sind "Auto" (sample_ratio_auto) oder "Set ratio" (sample_ratio_manual).
sample_ratio_value	*Float*	Das Verhältnis ist die Anzahl der Stichproben in der Minderheitsklasse über der Anzahl der Stichproben in der Mehrheitsklasse. Der Wert muss größer als 0 und kleiner-gleich 1 sein. Der Standardwert ist auto.
enable_random_seed	*boolesch*	Wenn die Option auf true gesetzt ist, wird die Eigenschaft random_seed aktiviert.
random_seed	*Integer*	Der Startwert, der vom Zufallszahlengenerator verwendet wird.
k_neighbours	*Integer*	Die Anzahl der nächsten Nachbarn, die zum Erstellen von künstlichen Stichproben verwendet werden. Der Standardwert ist 5.
m_neighbours	*Integer*	Die Anzahl der nächsten Nachbarn, die verwendet werden sollen, um zu ermitteln, ob eine Minderheitsstichprobe gefährdet ist. Diese Option ist nur für die SMOTE-Algorithmustypen borderline1 und borderline2 aktiviert. Der Standardwert ist 10.

Tabelle 268. Eigenschaften von "smotenode" (Forts.)		
Eigenschaften von "smotenode"	**Datentyp**	**Eigenschaftsbeschreibung**
algorithm_kind In algorithm umbenannt ab Version 18.2.1.1.	*string*	Der Typ des SMOTE-Algorithmus: regular, borderline1 oder borderline2.
usepartition In use_partition umbenannt ab Version 18.2.1.1.	*boolesch*	Wenn die Option auf true gesetzt ist, werden nur Trainingsdaten zur Modellerstellung verwendet. Der Standardwert ist true.

Eigenschaften von "tsnenode"

t-SNE (t-Distributed Stochastic Neighbor Embedding) ist ein Tool zum Visualisieren von hochdimensionalen Daten. Es wandelt Affinitäten von Datenpunkten in Wahrscheinlichkeiten um. Der t-SNE-Knoten in SPSS Modeler ist in Python implementiert und erfordert die Python-Bibliothek scikit-learn©.

Tabelle 269. Eigenschaften von "tsnenode"		
Eigenschaften von tsnenode	**Datentyp**	**Eigenschaftsbeschreibung**
mode_type	*string*	Geben Sie den Modus simple oder expert an.
n_components	*string*	Dimension des eingebetteten Raums (2-D oder 3-D). Geben Sie 2 oder 3 an. Der Standardwert ist 2.
method	*string*	Geben Sie barnes_hut oder exact an. Der Standardwert ist barnes_hut.
init	*string*	Initialisierung der Einbettung. Geben Sie random oder pca an. Der Standardwert ist random.
target_field In target umbenannt ab Version 18.2.1.1.	*string*	Zielfeldname. Dies wird eine Farbzuordnungstabelle im Ausgabediagramm sein. Das Diagramm wird eine Farbe verwenden, wenn kein Zielfeld angegeben wird.
perplexity	*Float*	Die Perplexität bezieht sich auf die Anzahl der nächsten Nachbarn, die in vielen anderen Lernalgorithmen für Mannigfaltigkeiten verwendet wird. Größere Datasets erfordern in der Regel eine höhere Perplexität. Ziehen Sie einen Wert zwischen 5 und 50 in Erwägung. Der Standardwert ist 30.
early_exaggeration	*Float*	Steuert, wie eng die natürlichen Cluster des Originalraums im eingebetteten Raum sind und wie viel Platz zwischen ihnen ist. Der Standardwert ist 12.0.
learning_rate	*Float*	Der Standardwert ist 200.

Tabelle 269. Eigenschaften von "tsnenode" (Forts.)

Eigenschaften von tsnenode	Datentyp	Eigenschaftsbeschreibung
n_iter	*Integer*	Maximale Anzahl Iterationen für die Optimierung. Geben Sie mindestens 250 an. Der Standardwert ist 1000.
angle	*Float*	Die Winkelgröße des fernen Knotens, gemessen von einem Punkt. Geben Sie einen Wert im Bereich 0-1 an. Der Standardwert ist 0.5.
enable_random_seed	*boolesch*	Setzen Sie diese Eigenschaft auf true, um den Parameter random_seed zu aktivieren. Der Standardwert ist false.
random_seed	*Integer*	Der zu verwendende Startwert für Zufallszahlen. Der Standardwert ist None.
n_iter_without_progress	*Integer*	Maximale Anzahl der Iterationen ohne Fortschritt. Der Standardwert ist 300.
min_grad_norm	*string*	Wenn die Gradientennorm unter diesem Schwellenwert liegt, wird die Optimierung gestoppt. Der Standardwert ist 1.0E-7. Gültige Werte: • 1.0E-1 • 1.0E-2 • 1.0E-3 • 1.0E-4 • 1.0E-5 • 1.0E-6 • 1.0E-7 • 1.0E-8
isGridSearch	*boolesch*	Setzen Sie diese Eigenschaft auf true, um t-SNE mit mehreren unterschiedlichen Perplexitäten auszuführen. Der Standardwert ist false.
output_Rename	*boolesch*	Geben Sie true an, wenn Sie einen benutzerdefinierten Namen bereitstellen wollen, oder geben Sie false an, um die Ausgabe automatisch zu benennen. Der Standardwert ist false.
output_to	*string*	Geben Sie Screen oder Output an. Der Standardwert ist Screen.
full_filename	*string*	Geben Sie den Namen der Ausgabedatei an.
output_file_type	*string*	Ausgabedateiformat. Geben Sie HTML oder Output object an. Der Standardwert ist HTML.

Eigenschaften von "xgboostlinearnode"

XGBoost Linear© ist eine erweiterte Implementierung eines Gradienten-Boosting-Algorithmus mit einem linearen Modell als Basismodell. Boosting-Algorithmen lernen iterativ schwache Klassifikationsmerkmale und fügen Sie einem endgültigen starken Klassifikationsmerkmal hinzu. Der Knoten "XGBoost Linear" in SPSS Modeler ist in Python implementiert.

Tabelle 270. Eigenschaften von "xgboostlinearnode"

Eigenschaften von "xgboostlinearnode"	**Datentyp**	**Eigenschaftsbeschreibung**
`TargetField` In `target` umbenannt ab Version 18.2.1.1.	*Feld*	
`InputFields` In `inputs` umbenannt ab Version 18.2.1.1.	*Feld*	
`alpha`	*double*	Der lineare Boosting-Parameter "alpha". Geben Sie eine beliebige Zahl (`0` oder höher) an. Der Standardwert ist `0`.
`lambda`	*double*	Der lineare Boosting-Parameter "lambda". Geben Sie eine beliebige Zahl (`0` oder höher) an. Der Standardwert ist `1`.
`lambdaBias`	*double*	Der lineare Boosting-Parameter "lambdaBias". Geben Sie eine beliebige Zahl an. Der Standardwert ist `0`.
`numBoostRound` In `num_boost_round` umbenannt ab Version 18.2.1.1.	*Integer*	Der Wert für die Anzahl Boosting-Runden zur Modellerstellung. Geben Sie einen Wert zwischen `1` und `1000` an. Der Standardwert ist `10`.
`objectiveType`	*string*	Der Lernzieltyp für die Aufgabe. Mögliche Werte sind `reg:linear`, `reg:logistic`, `reg:gamma`, `reg:tweedie`, `count:poisson`, `rank:pairwise`, `binary:logistic` oder `multi`. Hinweis: Für Flagziele können nur `binary:logistic` oder `multi` verwendet werden. Wenn `multi` verwendet wird, werden im Scoreergebnis die XGBoost-Lernzieltypen `multi:softmax` und `multi:softprob` angezeigt.
`random_seed`	*Integer*	Der Startwert für Zufallszahlen. Eine beliebige Zahl zwischen `0` und `9999999`. Der Standardwert ist `0`.

Tabelle 270. Eigenschaften von "xgboostlinearnode" (Forts.)		
Eigenschaften von "xgboostlinearnode"	**Datentyp**	**Eigenschaftsbeschreibung**
useHPO	*boolesch*	Geben Sie `true` oder `false` an, um die HPO-Optionen zu aktivieren oder zu inaktivieren. Bei Angabe von `true` wird Rbfopt angewendet, um das beste One-Class-SVM-Modell automatisch zu bestimmen, das den vom Benutzer über den Parameter `target_objval` angegebenen Zielwert erreicht.

Eigenschaften von "xgboosttreenode"

XGBoost Tree© ist eine erweiterte Implementierung eines Gradienten-Boosting-Algorithmus mit einem Baummodell als Basismodell. Boosting-Algorithmen lernen iterativ schwache Klassifikationsmerkmale und fügen Sie einem endgültigen starken Klassifikationsmerkmal hinzu. XGBoost Tree ist äußerst flexibel und stellt viele Parameter bereit, die die meisten Benutzer überfordern könnten. Der Knoten "XGBoost Tree" in SPSS Modeler stellt daher nur die zentralen Funktionen und gängigen Parameter dar. Der Knoten ist in Python implementiert.

Tabelle 271. Eigenschaften von "xgboosttreenode"		
Eigenschaften von "xgboosttreenode"	**Datentyp**	**Eigenschaftsbeschreibung**
`TargetField` In `target` umbenannt ab Version 18.2.1.1.	*Feld*	Die Zielfelder.
`InputFields` In `inputs` umbenannt ab Version 18.2.1.1.	*Feld*	Die Eingabefelder.
`treeMethod` In `tree_method` umbenannt ab Version 18.2.1.1.	*string*	Die Baummethode für die Modellerstellung. Mögliche Werte sind `auto`, `exact` oder `approx`. Der Standardwert ist `auto`.
`numBoostRound` In `num_boost_round` umbenannt ab Version 18.2.1.1.	*Integer*	Der Wert für die Anzahl Boosting-Runden zur Modellerstellung. Geben Sie einen Wert zwischen `1` und `1000` an. Der Standardwert ist `10`.
`maxDepth` In `max_depth` umbenannt ab Version 18.2.1.1.	*Integer*	Die maximale Tiefe für den Baumaufbau. Geben Sie `1` oder höher an. Der Standardwert ist `6`.
`minChildWeight` In `min_child_weight` umbenannt ab Version 18.2.1.1.	*double*	Die minimale Gewichtung untergeordneter Elemente für den Baumaufbau. Geben Sie `0` oder höher an. Der Standardwert ist `1`.

Tabelle 271. Eigenschaften von "xgboosttreenode" (Forts.)

Eigenschaften von "xgboosttreenode"	Datentyp	Eigenschaftsbeschreibung
maxDeltaStep In max_delta_step umbenannt ab Version 18.2.1.1.	*double*	Die maximale Delta-Schritt für den Baumaufbau. Geben Sie 0 oder höher an. Der Standardwert ist 0.
objectiveType In objective_type umbenannt ab Version 18.2.1.1.	*string*	Der Lernzieltyp für die Aufgabe. Mögliche Werte sind reg:linear, reg:logistic, reg:gamma, reg:tweedie, count:poisson, rank:pairwise, binary:logistic oder multi. Hinweis: Für Flagziele können nur binary:logistic oder multi verwendet werden. Wenn multi verwendet wird, werden im Scoreergebnis die XGBoost-Lernzieltypen multi:softmax und multi:softprob angezeigt.
earlyStopping In early_stopping umbenannt ab Version 18.2.1.1.	*boolesch*	Gibt an, ob die Early-Stopping-Funktion verwendet wird. Der Standardwert ist False.
earlyStoppingRounds In early_stopping_rounds umbenannt ab Version 18.2.1.1.	*Integer*	Die Anzahl der Validierungsfehler muss nach jeweils einer festgelegten Anzahl Early-Stopping-Runden zurückgehen, damit das Training fortgesetzt werden kann. Der Standardwert ist 10.
evaluationDataRatio In evaluation_data_ratio umbenannt ab Version 18.2.1.1.	*double*	Faktor der für Validierungsfehler verwendeten Eingabedaten. Der Standardwert ist 0.3.
random_seed	*Integer*	Der Startwert für Zufallszahlen. Eine beliebige Zahl zwischen 0 und 9999999. Der Standardwert ist 0.
sampleSize In sample_size umbenannt ab Version 18.2.1.1.	*double*	Die Teilstichprobe zur Steuerung der Überanpassung. Geben Sie einen Wert zwischen 0.1 und 1.0 an. Der Standardwert ist 0.1.
eta	*double*	Der Eta-Wert zur Steuerung der Überanpassung. Geben Sie einen Wert zwischen 0 und 1 an. Der Standardwert ist 0.3.
gamma	*double*	Der Gamma-Wert zur Steuerung der Überanpassung. Geben Sie eine beliebige Zahl (0 oder höher) an. Der Standardwert ist 6.
colsSampleRatio In col_sample_ratio umbenannt ab Version 18.2.1.1.	*double*	Die Spaltenstichprobe nach Baum zur Steuerung der Überanpassung. Geben Sie einen Wert zwischen 0.01 und 1 an. Der Standardwert ist 1.

Tabelle 271. Eigenschaften von "xgbosttreenode" (Forts.)

Eigenschaften von "xgboosttreenode"	**Datentyp**	**Eigenschaftsbeschreibung**
colsSampleLevel In col_sample_level umbenannt ab Version 18.2.1.1.	*double*	Die Spaltenstichprobe nach Ebene zur Steuerung der Überanpassung. Geben Sie einen Wert zwischen 0.01 und 1 an. Der Standardwert ist 1.
lambda	*double*	Der Lambda-Wert zur Steuerung der Überanpassung. Geben Sie eine beliebige Zahl (0 oder höher) an. Der Standardwert ist 1.
alpha	*double*	Der Alpha-Wert zur Steuerung der Überanpassung. Geben Sie eine beliebige Zahl (0 oder höher) an. Der Standardwert ist 0.
scalePosWeight In scale_pos_weight umbenannt ab Version 18.2.1.1.	*double*	Die Skalenpositionsgewichtung zur Behandlung von unausgewogenen Datasets. Der Standardwert ist 1.
use_HPO In Version 18.2.1.1 hinzugefügt.		

Kapitel 20. Eigenschaften des Spark-Knotens

Eigenschaften von "isotonicasnode"

Isotonische Regression gehört zur Familie der Regressionsalgorithmen. Der Knoten "Isonotisch-AS" in SPSS Modeler ist in Spark implementiert. Details zum Algorithmus für isotonische Regression finden Sie in https://spark.apache.org/docs/2.2.0/mllib-isotonic-regression.html.

Tabelle 272. Eigenschaften von "isotonicasnode"

Eigenschaften von `isotonicasnode`	**Datentyp**	**Eigenschaftsbeschreibung**
`label`	*string*	Diese Eigenschaft ist eine abhängige Variable, für die isotonische Regression berechnet wird.
`features`	*string*	Diese Eigenschaft ist eine unabhängige Variable.
`weightCol`	*string*	Die Gewichtung stellt eine Anzahl Maß dar. Der Standardwert ist 1.
`isotonic`	*boolesch*	Diese Eigenschaft gibt an, ob der Typ `isotonic` oder `antitonic` ist.
`featureIndex`	*Integer*	Diese Eigenschaft ist für den Index der Funktion, wenn `featuresCol` eine Vektorspalte ist. Der Standardwert ist `0`.

kmeansasnode, Eigenschaften

K-Means ist einer der am häufigsten verwendeten Clusteralgorithmen. Er teilt Datenpunkte in eine vordefinierte Anzahl Cluster auf. Der K-Means-AS-Knoten in SPSS Modeler ist in Spark implementiert. Details zum K-Means-Algorithums finden Sie in https://spark.apache.org/docs/2.2.0/ml-clustering.html. Beachten Sie, dass der K-Means-AS-Knoten automatisch eine 1-aus-n-Codierung für kategoriale Variablen durchführt.

Tabelle 273. kmeansasnode, Eigenschaften

Eigenschaften von `kmeansasnode`	**Werte**	**Eigenschaftsbeschreibung**
`roleUse`	*string*	Geben Sie `predefined` an, um vordefinierte Rollen zu verwenden, oder `custom`, um benutzerdefinierte Feldzuweisungen zu verwenden. Der Standardwert ist `predefined`.

Tabelle 273. kmeansasnode, Eigenschaften (Forts.)

Eigenschaften von kmeansasnode	**Werte**	**Eigenschaftsbeschreibung**
autoModel	*boolesch*	Geben Sie true an, wenn Sie den Standardnamen ($S-prediction) für das neue generierte Scoring-Feld verwenden möchten, oder false, um einen angepassten Namen zu verwenden. Der Standardwert ist true.
features	*Feld*	Liste der Feldnamen für die Eingabe, wenn die Eigenschaft roleUse auf custom gesetzt ist.
name	*string*	Der Name des neuen generierten Scoring-Felds, wenn die Eigenschaft autoModel auf false gesetzt ist.
clustersNum	*Integer*	Die Anzahl der zu erstellenden Cluster. Der Standardwert ist 5.
initMode	*string*	Der Initialisierungsalgorithmus. Mögliche Werte sind k-means|| oder random. Der Standardwert ist k-means||.
initSteps	*Integer*	Die Anzahl der Initialisierungsschritte, wenn initMode auf k-means|| gesetzt ist. Der Standardwert ist 2.
advancedSettings	*boolesch*	Geben Sie true an, um die folgenden vier Eigenschaften verfügbar zu machen. Der Standardwert ist false.
maxIteration	*Integer*	Maximale Anzahl Iterationen für das Clustering. Der Standardwert ist 20.
tolerance	*string*	Die Toleranz zum Stoppen der Iterationen. Mögliche Einstellungen sind 1.0E-1, 1.0E-2, ..., 1.0E-6. Der Standardwert ist 1.0E-4.
setSeed	*boolesch*	Geben Sie true an, um einen benutzerdefinierten Startwert für Zufallszahlen zu verwenden. Der Standardwert ist false.
randomSeed	*Integer*	Der benutzerdefinierte Startwert für Zufallszahlen, wenn die Eigenschaft setSeed auf true gesetzt ist.

Eigenschaften von "multilayerperceptronnode"

Mehrschicht-Perzeptron ist ein Klassifikationsmerkmal basierend auf dem künstlichen, neuronalen Feedforward-Netz und besteht aus mehreren Schichten. Jede Schicht ist vollständig mit der nächsten Schicht im Netz verbunden. Der Knoten "MultiLayerPerceptron-AS" in SPSS Modeler ist in Spark implementiert. Weitere Informationen zu MLPC (Multilayer Perceptron Classifier, Mehrschicht-Perzeptron-Klassifikationsmerkmal) finden Sie in https://spark.apache.org/docs/latest/ml-classification-regression.html#multilayer-perceptron-classifier.

Tabelle 274. Eigenschaften von "multilayerperceptronnode"

Eigenschaften von `multilayerperceptronnode`	**Datentyp**	**Eigenschaftsbeschreibung**
`features`	*Feld*	Mindestens ein Feld als Eingabe für die Vorhersage.
`label`	*Feld*	Das als Ziel für die Vorhersage zu verwendende Feld.
`layers[0]`	*Integer*	Die Anzahl einzuschließender Perzeptronschichten. Der Standardwert ist 1.
`layers[1…<latest-1>]`	*Integer*	Die Anzahl verborgener Schichten. Der Standardwert ist 1.
`layers[<latest>]`	*Integer*	Die Anzahl der Ausgabeschichten. Der Standardwert ist 1.
`seed`	*Integer*	Der benutzerdefinierte Startwert für Zufallszahlen.
`maxiter`	*Integer*	Die maximale Anzahl der auszuführenden Iterationen. Der Standardwert ist 10.

Eigenschaften von "xgboostasnode"

XGBoost ist eine erweiterte Implementierung eines Gradienten-Boosting-Algorithmus. Boosting-Algorithmen lernen iterativ schwache Klassifikationsmerkmale und fügen Sie einem endgültigen starken Klassifikationsmerkmal hinzu. XGBoost ist äußerst flexibel und stellt viele Parameter bereit, die die meisten Benutzer überfordern könnten. Der Knoten "XGBoost-AS" in SPSS Modeler stellt daher nur die zentralen Funktionen und gängigen Parameter dar. Der Knoten "XGBoost-AS" ist in Spark implementiert.

Tabelle 275. Eigenschaften von "xgboostasnode"

Eigenschaften von `xgboostasnode`	**Datentyp**	**Eigenschaftsbeschreibung**
`target_field`	*Feld*	Liste der Feldnamen für das Ziel.
`input_fields`	*Feld*	Liste der Feldnamen für Eingaben.
`nWorkers`	*Integer*	Die Anzahl der zum Trainieren des XGBoost-Modells verwendeten Worker. Der Standardwert ist 1.
`numThreadPerTask`	*Integer*	Die Anzahl der pro Worker verwendeten Threads. Der Standardwert ist 1.

Tabelle 275. Eigenschaften von "xgboostasnode" (Forts.)

Eigenschaften von xgboostasnode	Datentyp	Eigenschaftsbeschreibung
useExternalMemory	*boolesch*	Gibt an, ob externer Speicher als Cache verwendet wird. Der Standardwert ist **False**.
boosterType	*string*	Der zu verwendende Boostertyp. Verfügbare Optionen sind gbtree, gblinear oder dart. Der Standardwert ist gbtree.
numBoostRound	*Integer*	Die Anzahl der Runden für das Boosting. Geben Sie 0 oder höher an. Der Standardwert ist 10.
scalePosWeight	*double*	Steuert den Ausgleich zwischen positiven und negativen Gewichtungen. Der Standardwert ist 1.
randomseed	*Integer*	Der Startwert, der vom Zufallszahlengenerator verwendet wird. Der Standardwert ist **0**.
objectiveType	*string*	Das Lernziel. Mögliche Werte sind reg:linear, reg:logistic, reg:gamma, reg:tweedie, rank:pairwise, binary:logistic oder multi. Hinweis: Für Flagziele können nur binary:logistic oder multi verwendet werden. Wenn multi verwendet wird, werden im Scoreergebnis die XGBoost-Lernzieltypen multi:softmax und multi:softprob angezeigt. Der Standardwert ist reg:linear.
evalMetric	*string*	Auswertungsmetriken zum Validieren von Daten. Eine Standardmetrik wird dem Lernziel entsprechend zugeordnet. Mögliche Werte sind rmse, mae, logloss, error, merror, mlogloss, auc, ndcg, map oder gamma-deviance. Der Standardwert ist rmse.
lambda	*double*	L2-Regularisierungsterm bei Gewichtungen. Wenn Sie diesen Wert erhöhen, führt dies zu einem konservativeren Modell. Geben Sie eine beliebige Zahl (0 oder höher) an. Der Standardwert ist 1.
alpha	*double*	L1-Regularisierungsterm bei Gewichtungen. Wenn Sie diesen Wert erhöhen, führt dies zu einem konservativeren Modell. Geben Sie eine beliebige Zahl (0 oder höher) an. Der Standardwert ist 0.

Tabelle 275. Eigenschaften von "xgboostasnode" (Forts.)

Eigenschaften von xgboostasnode	Datentyp	Eigenschaftsbeschreibung
lambdaBias	*double*	L2-Regularisierungsterm bei Verzerrung. Wenn der Boostertyp gblinear verwendet wird, ist dieser lineare Boosterparameter für Lambda-Verzerrung verfügbar. Geben Sie eine beliebige Zahl (0 oder höher) an. Der Standardwert ist 0.
treeMethod	*string*	Wenn der Boostertyp gbtree oder dart verwendet wird, ist dieser Baummethodenparameter für den Baumaufbau (und die anderen Baumparameter, die folgen) verfügbar. Er gibt den zu verwendenden XGBoost-Baumerstellungsalgorithmus an. Verfügbare Optionen sind auto, exact und approx. Der Standardwert ist auto.
maxDepth	*Integer*	Die maximale Baumtiefe. Geben Sie 2 oder höher an. Der Standardwert ist 6.
minChildWeight	*double*	Die für ein untergeordnetes Element erforderliche Mindestsumme für die Instanzgewichtung (Hesse). Geben Sie 0 oder höher an. Der Standardwert ist 1.
maxDeltaStep	*double*	Der maximale Deltaschritt, der für die Gewichtungsschätzung der einzelnen Bäume zulässig ist. Geben Sie 0 oder höher an. Der Standardwert ist 0.
sampleSize	*double*	Die Teilstichprobe ist für das Verhältnis der Trainingsinstanz. Geben Sie einen Wert zwischen 0.1 und 1.0 an. Der Standardwert ist 1.0.
eta	*double*	Der Schwund der Schrittgröße während des Aktualisierungsschritts, um eine Überanpassung zu verhindern. Geben Sie einen Wert zwischen 0 und 1 an. Der Standardwert ist 0.3.
gamma	*double*	Die mindestens erforderliche Verlustverkleinerung, um eine weitere Partition für einen Blattknoten des Baums zu erstellen. Geben Sie eine beliebige Zahl (0 oder höher) an. Der Standardwert ist 6.
colsSampleRatio	*double*	Das Teilstichprobenverhältnis der Spalten beim Erstellen der einzelnen Bäume. Geben Sie einen Wert zwischen 0.01 und 1 an. Der Standardwert ist 1.
colsSampleLevel	*double*	Das Teilstichprobenverhältnis der Spalten für jede Aufteilung auf jeder Ebene. Geben Sie einen Wert zwischen 0.01 und 1 an. Der Standardwert ist 1.

Tabelle 275. Eigenschaften von "xgboostasnode" (Forts.)

Eigenschaften von xgboostas-node	**Datentyp**	**Eigenschaftsbeschreibung**
normalizeType	*string*	Wenn der Boostertyp "dart" verwendet wird, sind dieser Dartparameter und die folgenden drei Dartparameter verfügbar. Dieser Parameter legt den Normalisierungsalgorithmus fest. Geben Sie `tree` oder `forest` an. Der Standardwert ist `tree`.
sampleType	*string*	Der Typ des Stichprobenziehungsalgorithmus. Geben Sie `uniform` oder `weighted` an. Der Standardwert ist `uniform`.
rateDrop	*double*	Der Dart-Booster-Parameter für die Abbruchquote. Geben Sie einen Wert zwischen `0.0` und `1.0` an. Der Standardwert ist `0.0`.
skipDrop	*double*	Der Dart-Booster-Parameter für die Wahrscheinlichkeit des Überspringabbruchs. Geben Sie einen Wert zwischen `0.0` und `1.0` an. Der Standardwert ist `0.0`.

Kapitel 21. Superknoteneigenschaften

In den folgenden Tabellen werden die für Superknoten spezifischen Eigenschaften beschrieben. Beachten Sie, dass allgemeine Knoteneigenschaften auch für Superknoten gelten.

Tabelle 276. Eigenschaften von Endsuperknoten

Eigenschaftsname	Eigenschaftstyp/Liste der Werte	Eigenschaftsbeschreibung

`execute_method`	`Script`
	
	`Normal`	

`script`	*string*	

Superknotenparameter

Mithilfe von Scripts können Sie SuperNode-Parameter mit allgemeinem Format erstellen bzw. festlegen:

```
mySuperNode.setParameterValue("minvalue", 30)
```

Sie können den Parameterwert wie folgt abrufen:

```
value mySuperNode.getParameterValue("minvalue")
```

Suchen von vorhandenen Superknoten

Sie können Superknoten in Streams mit der Funktion findByType() suchen:

```
source_supernode = modeler.script.stream().findByType("source_super", None)
process_supernode = modeler.script.stream().findByType("process_super", None)
terminal_supernode = modeler.script.stream().findByType("terminal_super", No
ne)
```

Festlegen von Eigenschaften für gekapselte Knoten

Sie können Eigenschaften für einzelne, in einem Superknoten gekapselte Knoten festlegen, indem Sie auf das untergeordnete Diagramm im Superknoten zugreifen. Nehmen Sie beispielsweise an, dass Sie einen Quellensuperknoten mit einem gekapselten Knoten "Variable Datei" zum Einlesen der Daten haben. Sie können den Namen der zu lesenden Datei (mithilfe der Eigenschaft `full_filename` angegeben) durch Zugreifen auf das untergeordnete Diagramm und Suchen des relevanten Knotens wie folgt weitergeben:

```
childDiagram = source_supernode.getChildDiagram()
varfilenode = childDiagram.findByType("variablefile", None)
varfilenode.setPropertyValue("full_filename", "c:/mydata.txt")
```

Erstellen von Superknoten

Wenn Sie einen Superknoten und seinen Inhalt völlig neu erstellen wollen, können Sie dies auf ähnliche Weise tun, indem Sie den Superknoten erstellen, auf das untergeordnete Diagramm zugreifen und die gewünschten Knoten erstellen. Sie müssen außerdem sicherstellen, dass die Knoten im Superknotendi-

agramm auch mit den Eingabe- und/oder Ausgabeconnectorknoten verbunden sind. Beispiel für das Erstellen eines Prozesssuperknotens:

```
process_supernode = modeler.script.stream().createAt("process_super", "My Su
perNode", 200, 200)
childDiagram = process_supernode.getChildDiagram()
filternode = childDiagram.createAt("filter", "My Filter", 100, 100)
childDiagram.linkFromInputConnector(filternode)
childDiagram.linkToOutputConnector(filternode)
```

Anhang A. Knotennamenreferenz

In diesem Abschnitt erhalten Sie eine Referenz für die Scriptnamen der Knoten in IBM SPSS Modeler.

Modellnuggetnamen

Modellnuggets (auch als "generierte Modelle" bezeichnet) können ebenso wie Knoten- und Ausgabeobjekte nach Typ referenziert werden. In der folgenden Tabelle werden die Referenznamen für Modellobjekte aufgeführt.

Beachten Sie, dass diese Namen speziell zur Referenzierung von Modellnuggets in der Modellpalette (in der rechten oberen Ecke des IBM SPSS Modeler-Fensters) verwendet werden. Zur Referenzierung von Modellknoten, die zu Scoring-Zwecken zu einem Stream hinzugefügt wurden, wird ein anderes Set von Namen mit dem Präfix `apply...` verwendet.

Hinweis: Unter normalen Umständen wird die Referenzierung von Modellen sowohl anhand des Namens *als auch* anhand des Typs empfohlen, um Verwirrungen zu vermeiden.

Tabelle 277. Namen von Modellnuggets (Modellierungspalette)

Modellname	**Modell**
anomalydetection	Anomalie
Apriori-Modelle	Apriori
autoclassifier	Automatisches Klassifikationsmerkmal
autocluster	Autom. Cluster
autonumeric	Autonumerisch
bayesnet	Bayes-Netz
c50	C5.0
carma	Carma
cart	C&R-Baum
chaid	CHAID
coxreg	Cox-Regression
decisionlist	Entscheidungsliste
discriminant	Diskriminanz
Faktor	Faktor/PCA
featureselection	Merkmalauswahl
genlin	Verallgemeinerte lineare Regression
glmm	GLMM
kmeans	K-Means
knn	*k*-Nächste-Nachbarn
kohonen	Kohonen
linear	Linear
logreg	Logistische Regression

Tabelle 277. Namen von Modellnuggets (Modellierungspalette) (Forts.)	
Modellname	**Modell**
neuralnetwork	Neuronales Netz
quest	QUEST
Regression	Lineare Regression
Sequenz	Sequenz
slrm	Lernfähiges Antwortmodell
statisticsmodel	IBM SPSS Statistics-Modell
svm	Support Vector Machine
timeseries	Zeitreihe
TwoStep	Two Step

Tabelle 278. Namen von Modellnuggets (Datenbankmodellierungspalette)	
Modellname	**Model**
db2imcluster	IBM ISW-Clustering
db2imlog	IBM ISW Logistische Regression
db2imnb	IBM ISW Naive Bayes
db2imreg	IBM ISW-Regression
db2imtree	IBM ISW-Entscheidungsbaum
msassoc	MS-Assoziationsregeln
msbayes	MS Naive Bayes
mscluster	MS-Clustering
mslogistic	MS - Logistische Regression
msneuralnetwork	MS - Neuronales Netz
msregression	MS - Lineare Regression
mssequencecluster	MS Sequenz-Clustering
mstimeseries	MS Time Series
mstree	MS-Entscheidungsbaum
netezzabayes	Netezza-Bayes-Netz
netezzadectree	Netezza-Entscheidungsbaum
netezzadivcluster	Netezza - Divisives Clustering
netezzaglm	Verallgemeinertes lineares Netezza-Modell
netezzakmeans	Netezza-K-Means
netezzaknn	Netezza-KNN
netezzalineregression	Netezza - Lineare Regression
netezzanaivebayes	Netezza - Naive Bayes
netezzapca	Netezza-PCA

Tabelle 278. Namen von Modellnuggets (Datenbankmodellierungspalette) (Forts.)	
Modellname	**Model**
netezzaregtree	Netezza-Regressionsbaum
netezzatimeseries	Netezza-Zeitreihe
oraabn	Oracle Adaptive Bayes
oraai	Oracle AI
oradecisiontree	Oracle Decision Tree
oraglm	Oracle GLM
orakmeans	Oracle *k*-Means
oranb	Oracle Naive Bayes
oranmf	Oracle NMF
oraocluster	Oracle O-Cluster
orasvm	Oracle SVM

Vermeidung doppelter Modellnamen

Bei der Verwendung von Scripts zur Bearbeitung generierter Modelle sollten Sie sich bewusst sein, dass das Zulassen doppelter Modellnamen zu mehrdeutigen Referenzen führen kann. Um dies zu vermeiden, sollten bei der Scripterstellung eindeutige Namen für die generierten Modelle erforderlich sein.

So legen Sie Optionen für doppelte Modellnamen fest:

1. Wählen Sie in den Menüs Folgendes aus:

   Extras > **Benutzeroptionen**
2. Klicken Sie auf die Registerkarte **Benachrichtigungen**.
3. Wählen Sie die Option **Bisheriges Modell ersetzen**, um die Vergabe doppelter Namen für generierte Modelle zu beschränken.

Das Verhalten der Scriptausführung kann zwischen SPSS Modeler und IBM SPSS Collaboration and Deployment Services variieren, wenn mehrdeutige Modellverweise vorliegen. Der SPSS Modeler-Client beinhaltet die Option "Bisheriges Modell ersetzen", bei dem automatisch Modelle mit demselben Namen ersetzt werden (z. B. wenn ein Script in mehreren Iterationen eine Schleife durchläuft und jedes Mal ein anderes Modell erstellt). Diese Option steht jedoch nicht zur Verfügung, wenn dasselbe Script in IBM SPSS Collaboration and Deployment Services ausgeführt wird. Sie können diese Situation vermeiden, indem Sie entweder das in den einzelnen Iterationen generierte Modell umbenennen, um mehrdeutige Verweise auf Modelle zu vermeiden, oder indem Sie das aktuelle Modell vor dem Ende der Schleife löschen (z. B. durch Hinzufügen der Anweisung `clear generated palette`).

Namen der Ausgabetypen

In der folgenden Tabelle werden alle Ausgabeobjekttypen und die Knoten, von denen Sie erstellt werden, aufgelistet.

Tabelle 279. Ausgabeobjekttypen und die Knoten, von denen sie erstellt werden	
Ausgabeobjekttyp	**Knoten**
analysisoutput	Analyse
collectionoutput	Datenerfassung

Tabelle 279. Ausgabeobjekttypen und die Knoten, von denen sie erstellt werden (Forts.)

Ausgabeobjekttyp	**Knoten**
dataauditoutput	Data Audit
distributionoutput	Verteilung
evaluationoutput	Evaluierung
histogramoutput	Histogramm
matrixoutput	Matrix
meansoutput	Mittelwert
multiplotoutput	Multiplot
plotoutput	Diagramm
qualityoutput	Qualität
reportdocumentoutput	Dieser Objekttyp stammt nicht aus einem Knoten; es handelt sich um die von einem Projektbericht erstellte Ausgabe.
reportoutput	Bericht
statisticsprocedureoutput	Statistics-Ausgabe
statisticsoutput	Statistik
tableoutput	Tabelle
timeplotoutput	Zeitdiagramm
weboutput	Internet

Anhang B. Migration von traditionellem Scripting zu Python-Scripting

Übersicht über die Migration traditioneller Scripts

In diesem Abschnitt erhalten Sie eine Zusammenfassung der Unterschiede zwischen Python-Scripting und traditionellem Scripting in IBM SPSS Modeler sowie Informationen zur Migration von traditionellen Scripts in Python-Scripts. In diesem Abschnitt finden Sie eine Liste der traditionellen SPSS Modeler-Standardbefehle und der entsprechenden Python-Befehle.

Allgemeine Unterschiede

Traditionelles Scripting beruht stark auf Betriebssystembefehlsscripts. Traditionelles Scripting ist zeilenorientiert. Obwohl es einige Blockstrukturen wie z. B. `if...then...else...endif` und `for...endfor` gibt, hat eine Einrückung in der Regel keine Bedeutung.

Beim Python-Scripting ist die Einrückung von Bedeutung. Zeilen, die zum selben logischen Block gehören, müssen gleich weit eingerückt sein.

Anmerkung: Darauf müssen Sie beim Kopieren und Einfügen von Python-Code achten. Eine Zeile, die mit Tabstopps eingerückt ist, könnte im Editor genauso aussehen wie eine Zeile, die mit Leerzeichen eingerückt ist. Das Python-Script generiert jedoch einen Fehler, da diese Zeilen nicht als gleich weit eingerückt gelten.

Scripting-Kontext

Der Scripting-Kontext definiert die Umgebung, in der das Script ausgeführt wird, z. B. den Stream oder den Superknoten, der das Script ausführt. Beim traditionellem Scripting ist der Kontext implizit, d. h. es wird angenommen, dass alle Knotenreferenzen in einem Stream-Script in dem Stream liegen, der das Script ausführt.

Beim Python-Scripting wird der Scripting-Kontext explizit über das Modul `modeler.script` angegeben. Beispiel: Ein Python-Stream-Script kann mit dem folgenden Code auf den Stream zugreifen, der das Script ausführt:

```
s = modeler.script.stream()
```

Funktionen im Zusammenhang mit Streams können dann über das zurückgegebene Objekt aufgerufen werden.

Befehle und Funktionen

Traditionelles Scripting ist befehlsorientiert. Das heißt, dass jede Zeile des Scripts typischerweise mit dem auszuführenden Befehl beginnt und dann die Parameter folgen. Beispiel:

```
connect 'Type':typenode to :filternode
rename :derivenode as "Compute Total"
```

Python verwendet Funktionen, die üblicherweise über ein Objekt (Modul, Klasse oder Objekt) aufgerufen werden, das die Funktion definiert. Beispiel:

```
stream = modeler.script.stream()
typenode = stream.findByType("type", "Type)
filternode = stream.findByType("filter", None)
stream.link(typenode, filternode)
derive.setLabel("Compute Total")
```

Literale und Kommentare

Einige Literal- und Kommentarbefehle, die üblicherweise in IBM SPSS Modeler verwendet werden, haben funktional entsprechende Befehle in Python-Scripts. Dies könnte Ihnen dabei helfen, Ihre vorhandenen traditionellen SPSS Modeler-Scripts für die Verwendung in IBM SPSS Modeler 17 in Python-Scripts zu konvertieren.

Tabelle 280. Zuordnung von traditionellem Scripting zu Python-Scripting für Literale und Kommentare

Traditionelles Scripting	**Python-Scripting**
Ganzzahl, z. B. 4	Gleich
Gleitkommazahl, z. B. `0.003`	Gleich
Zeichenfolgen in einfachen Anführungszeichen, z. B. `'Hello'`	Gleich **Anmerkung:** Zeichenfolgeliteralen mit Nicht-ASCII-Zeichen muss ein u vorangestellt werden, damit sie als Unicode dargestellt werden.
Zeichenfolgen in doppelten Anführungszeichen, z. B. `"Hello again"`	Gleich **Anmerkung:** Zeichenfolgeliteralen mit Nicht-ASCII-Zeichen muss ein u vorangestellt werden, damit sie als Unicode dargestellt werden.
Lange Zeichenfolgen, z. B. `"""This is a string that spans multiple lines"""`	Gleich
Listen, z. B. `[1 2 3]`	`[1, 2, 3]`
Variablenreferenz, z. B. `set x = 3`	`x = 3`
Zeilenfortsetzung (\), z. B. `set x = [1 2 \ 3 4]`	`x = [ 1, 2,\ 3, 4]`
Blockkommentar, z. B. `/* This is a long comment over a line. */`	`""" This is a long comment over a line. """`
Zeilenkommentar, z. B. `set x = 3 # make x 3`	`x = 3 # make x 3`
`undef`	`None`
`true`	`True`
`false`	`False`

Operatoren

Einige Operatorbefehle, die üblicherweise in IBM SPSS Modeler verwendet werden, haben funktional entsprechende Befehle in Python-Scripts. Dies könnte Ihnen dabei helfen, Ihre vorhandenen traditionellen SPSS Modeler-Scripts für die Verwendung in IBM SPSS Modeler 17 in Python-Scripts zu konvertieren.

Tabelle 281. Zuordnung von traditionellem Scripting zu Python-Scripting für Operatoren

Traditionelles Scripting	Python-Scripting

`NUM1 + NUM2`
`LIST + ITEM`
`LIST1 + LIST2`
	`NUM1 + NUM2`
	`LIST.append(ITEM)`
	`LIST1.extend(LIST2)`

`NUM1 - NUM2` `LIST - ITEM`	`NUM1 - NUM2` `LIST.remove(ITEM)`
`NUM1 * NUM2`	`NUM1 * NUM2`
`NUM1 / NUM2`	`NUM1 / NUM2`
`=` `==`	`==`
`/=` `/==`	`!=`
`X ** Y`	`X ** Y`

`X < Y`
`X <= Y`
`X > Y`
`X >= Y`
	`X < Y`
	`X <= Y`
	`X > Y`
	`X >= Y`

`X div Y`
`X rem Y`
`X mod Y`
	`X // Y`
	`X % Y`
	`X % Y`

`and`
`or`
`not(EXPR)`
	`and`
	`or`
	`not EXPR`

Bedingte Befehle und Schleifenbefehle

Für einige bedingte Befehle und Schleifenbefehle, die üblicherweise in IBM SPSS Modeler verwendet werden, gibt es entsprechende Befehle beim Python-Scripting. Dies könnte Ihnen dabei helfen, Ihre vorhandenen traditionellen SPSS Modeler-Scripts für die Verwendung in IBM SPSS Modeler 17 in Python-Scripts zu konvertieren.

Tabelle 282. Zuordnung von traditionellem Scripting zu Python-Scripting für bedingte Befehle und Schleifenbefehle

Traditionelles Scripting	Python-Scripting

`for VAR from INT1 to INT2`
`  …`
`endfor`
	`for VAR in range(INT1, INT2):`
	`  …`
	oder
	`VAR = INT1`
	`while VAR <= INT2:`
	`  ...`
	`  VAR += 1`

Tabelle 282. Zuordnung von traditionellem Scripting zu Python-Scripting für bedingte Befehle und Schleifenbefehle (Forts.)

Traditionelles Scripting	Python-Scripting

`for VAR in LIST`
`  …`
`endfor`
	`for VAR in LIST:`
	`  …`

`for VAR in_fields_to NODE`
`  …`
`endfor`
	`for VAR in NODE.getInputDataModel():`
	`  ...`

`for VAR in_fields_at NODE`
`  …`
`endfor`
	`for VAR in NODE.getOutputDataModel():`
	`  ...`

`if…then`
`  …`
`elseif…then`
`  …`
`else`
`  …`
`endif`
	`if …:`
	`  …`
	`elif …:`
	`  …`
	`else:`
	`  …`

`with TYPE OBJECT`
`  …`
`endwith`
	Keine Entsprechung

`var VAR1`	Variablendeklaration nicht erforderlich

Variablen

Beim traditionellen Scripting werden Variablen deklariert, bevor sie referenziert werden. Beispiel:

```
var mynode
set mynode = create typenode at 96 96
```

Beim Python-Scripting werden Variablen bei ihrer ersten Referenzierung erstellt. Beispiel:

```
mynode = stream.createAt("type", "Type", 96, 96)
```

Beim traditionellen Scripting müssen Referenzen auf Variablen explizit mit dem Operator ^ entfernt werden. Beispiel:

```
var mynode
set mynode = create typenode at 96 96
set ^mynode.direction."Age" = Input
```

Wie bei den meisten Scriptsprachen ist dies beim Python-Scripting nicht erforderlich. Beispiel:

```
mynode = stream.createAt("type", "Type", 96, 96)
mynode.setKeyedPropertyValue("direction","Age","Input")
```

Knoten-, Ausgabe- und Modelltypen

Beim traditionellen Scripting wird für die unterschiedlichen Objekttypen (Knoten, Ausgabe und Modell) in der Regel der Typ an den Objekttyp angehängt. Beispiel: der Knoten "derive" hat den Typ `derivenode`:

```
set feature_name_node = create derivenode at 96 96
```

Die IBM SPSS Modeler-API in Python schließt das Suffix node nicht ein, sodass der Knoten "derive" den Typ `derive` hat. Beispiel:

```
feature_name_node = stream.createAt("derive", "Feature", 96, 96)
```

Typnamen beim traditionellen Scripting und beim Python-Scripting unterscheiden sich nur darin, dass beim Python-Scripting das Typsuffix fehlt.

Eigenschaftsnamen

Eigenschaftsnamen sind beim traditionellen Scripting und beim Python-Scripting gleich. Beispiel: Im Variablendateiknoten lautet die Eigenschaft, die die Dateiposition definiert, in beiden Scripting-Umgebungen `full_filename`.

Knotenreferenzen

Viele traditionelle Scripts verwenden eine implizite Suche, um den zu ändernden Knoten zu suchen und darauf zuzugreifen. Beispiel: die folgenden Befehle durchsuchen den aktuellen Stream nach einem Typknoten mit der Beschriftung "Type" und legen dann für die Richtung (Modellierungsrolle) das Feld "Age" als Eingabe und das Feld "Drug" als Ziel (vorherzusagender Wert) fest:

```
set 'Type':typenode.direction."Age" = Input
set 'Type':typenode.direction."Drug" = Target
```

Beim Python-Scripting müssen Knotenobjekte explizit gesucht werden, bevor die Funktion zum Festlegen des Eigenschaftswert aufgerufen wird. Beispiel:

```
typenode = stream.findByType("type", "Type")
typenode.setKeyedPropertyValue("direction", "Age", "Input")
typenode.setKeyedPropertyValue("direction", "Drug", "Target")
```

Anmerkung: In diesem Fall muss "Target" in Anführungszeichen für Zeichenfolgen stehen.

Python-Scripts können alternativ die Aufzählung `ModelingRole` im Paket `modeler.api` verwenden.

Obwohl Scripts beim Python-Scripting länger sein können, führt dies zu einer besseren Laufzeitleistung, da die Suche nach dem Knoten in der Regel nur einmal erfolgt. Beim Beispiel mit traditionellem Scripting wird die Suche nach dem Knoten bei jedem Befehl ausgeführt.

Die Suche nach Knoten anhand der ID wird ebenfalls unterstützt. (Die Knoten-ID ist auf der Registerkarte "Anmerkungen" des Knotendialogs zu sehen.) Beispiel beim traditionellen Scripting:

```
# id65EMPB9VL87 is the ID of a Type node
set @id65EMPB9VL87.direction."Age" = Input
```

Das folgende Script zeigt dasselbe Beispiel beim Python-Scripting:

```
typenode = stream.findByID("id65EMPB9VL87")
typenode.setKeyedPropertyValue("direction", "Age", "Input")
```

Abrufen und Festlegen von Eigenschaften

Traditionelles Scripting verwendet den Befehl `set` zum Festlegen eines Wertes. Der Ausdruck nach dem Befehl `set` kann eine Eigenschaftsdefinition sein. Das folgende Script zeigt zwei mögliche Scriptformate zum Festlegen einer Eigenschaft:

```
set <node reference>.<property> = <value>
set <node reference>.<keyed-property>.<key> = <value>
```

Beim Python-Scripting wird dasselbe Ergebnis mit den Funktionen `setPropertyValue()` und `setKeyedPropertyValue()` erreicht. Beispiel:

```
object.setPropertyValue(property, value)
object.setKeyedPropertyValue(keyed-property, key, value)
```

Beim traditionellen Scripting kann mit dem Befehl `get` auf Eigenschaftswerte zugegriffen werden. Beispiel:

```
var n v
set n = get node :filternode
set v = ^n.name
```

Beim Python-Scripting wird dasselbe Ergebnis erzielt, wenn die Funktion `getPropertyValue()` verwendet wird. Beispiel:

```
n = stream.findByType("filter", None)
v = n.getPropertyValue("name")
```

Bearbeiten von Streams

Beim traditionellen Scripting wird der Befehl `create` zum Erstellen eines neuen Knotens verwendet. Beispiel:

```
var agg select
set agg = create aggregatenode at 96 96
set select = create selectnode at 164 96
```

Beim Python-Scripting gibt es in Streams verschiedene Methoden zur Erstellung von Knoten. Beispiel:

```
stream = modeler.script.stream()
agg = stream.createAt("aggregate", "Aggregate", 96, 96)
select = stream.createAt("select", "Select", 164, 96)
```

Beim traditionellen Scripting wird der Befehl `connect` zum Herstellen von Verbindungen zwischen Knoten verwendet. Beispiel:

```
connect ^agg to ^select
```

Beim Python-Scripting wird die Methode `link` zum Herstellen von Verbindungen zwischen Knoten verwendet. Beispiel:

```
stream.link(agg, select)
```

Beim traditionellen Scripting wird der Befehl `disconnect` zum Entfernen von Verbindungen zwischen Knoten verwendet. Beispiel:

```
disconnect ^agg from ^select
```

Beim Python-Scripting wird die Methode `unlink` zum Entfernen von Verbindungen zwischen Knoten verwendet. Beispiel:

```
stream.unlink(agg, select)
```

Beim traditionellen Scripting wird der Befehl `position` zur Positionierung von Knoten im Streamerstellungsbereich oder zwischen anderen Knoten verwendet. Beispiel:

```
position ^agg at 256 256
position ^agg between ^myselect and ^mydistinct
```

Beim Python-Scripting wird dasselbe Ergebnis mit zwei separaten Methoden erreicht: `setXYPosition` und `setPositionBetween`. Beispiel:

```
agg.setXYPosition(256, 256)
agg.setPositionBetween(myselect, mydistinct)
```

Knotenoperationen

Einige Befehle für Knotenoperationen, die üblicherweise in IBM SPSS Modeler verwendet werden, haben funktional entsprechende Befehle in Python-Scripts. Dies könnte Ihnen dabei helfen, Ihre vorhandenen traditionellen SPSS Modeler-Scripts für die Verwendung in IBM SPSS Modeler 17 in Python-Scripts zu konvertieren.

Tabelle 283. Zuordnung von traditionellem Scripting zu Python-Scripting für Knotenoperationen

Traditionelles Scripting	Python-Scripting

`create nodespec at x y`	`stream.create(type, name)`
	`stream.createAt(type, name, x, y)`
	`stream.createBetween(type, name, preNode, postNode)`
	`stream.createModelApplier(model, name)`

`connect fromNode to toNode`	`stream.link(fromNode, toNode)`
`delete node`	`stream.delete(node)`
`disable node`	`stream.setEnabled(node, False)`
`enable node`	`stream.setEnabled(node, True)`
`disconnect fromNode from toNode`	`stream.unlink(fromNode, toNode)` `stream.disconnect(node)`
`duplicate node`	`node.duplicate()`
`execute node`	`stream.runSelected(nodes, results)` `stream.runAll(results)`
`flush node`	`node.flushCache()`
`position node at x y`	`node.setXYPosition(x, y)`
`position node between node1 and node2`	`node.setPositionBetween(node1, node2)`
`rename node as name`	`node.setLabel(name)`

Verwendung von Schleifen

Beim traditionellen Scripting werden zwei Hauptschleifenoptionen unterstützt:

- *Gezählte Schleifen*, bei denen eine Indexvariable sich zwischen zwei ganzzahligen Grenzen bewegt.
- *Sequenzschleifen*, die eine Folge von Werten in einer Schleife durchlaufen und den aktuellen Wert an die Schleifenvariable binden.

Das folgende Script ist ein Beispiel für eine gezählte Schleife beim traditionellen Scripting:

```
for i from 1 to 10
    println ^i
endfor
```

Das folgende Script ist ein Beispiel für eine Sequenzschleife beim traditionellen Scripting:

```
var items
set items = [a b c d]

for i in items
    println ^i
endfor
```

Es können auch andere Typen von Schleifen verwendet werden:

- Durchlaufen der Modelle in der Modellpalette oder der Ausgaben in der Ausgabepalette.
- Durchlaufen der Felder, die in einen Knoten eintreten oder aus einem Knoten austreten.

Python-Scripting unterstützt auch unterschiedliche Schleifentypen. Das folgende Script ist ein Beispiel für eine gezählte Schleife beim Python-Scripting:

```
i = 1
while i <= 10:
    print i
    i += 1
```

Das folgende Script ist ein Beispiel für eine Sequenzschleife beim Python-Scripting:

```
items = ["a", "b", "c", "d"]
for i in items:
    print i
```

Die Sequenzschleife ist sehr flexibel. Wenn sie mit API-Methoden von IBM SPSS Modeler kombiniert wird, kann sie die Mehrzahl der Anwendungsfälle beim traditionellen Scripting unterstützen. Das folgende Beispiel zeigt, wie mit einer Sequenzschleife beim Python-Scripting die Felder durchlaufen werden können, die aus einem Knoten austreten:

```
node = modeler.script.stream().findByType("filter", None)
for column in node.getOutputDataModel().columnIterator():
    print column.getColumnName()
```

Streams ausführen

Während der Streamausführung werden generierte Modelle oder Ausgabeobjekte zu einem der Objektmanager hinzugefügt. Beim traditionellen Scripting muss das Script entweder die erstellten Objekte im Objektmanager finden oder auf die zuletzt generierte Ausgabe aus dem Knoten zugreifen, der die Ausgabe erstellt hat.

Die Streamausführung in Python unterscheidet sich darin, dass alle von der Ausführung generierten Modell- oder Ausgabeobjekte in einer Liste zurückgegeben werden, die an die Ausführungsfunktion übergeben wird. Dadurch kann leichter auf die Ergebnisse der Streamausführung zugegriffen werden.

Traditionelles Scripting unterstützt drei Befehle zur Streamausführung:

- `execute_all` - führt alle ausführbaren Endknoten im Stream aus.
- `execute_script` - führt das Stream-Script unabhängig von der Einstellung der Scriptausführung aus.
- `execute` *node*führt den angegebenen Knoten aus.

Python-Scripting unterstützt eine ähnliche Gruppe von Funktionen:

- *stream*`.runAll(`*results-list*`)` - führt alle ausführbaren Endknoten im Stream aus.
- *stream*`.runScript(`*results-list*`)` - führt das Stream-Script unabhängig von der Einstellung der Scriptausführung aus.
- *stream*`.runSelected(`*node-array*`, `*results-list*`)`führt die angegebene Gruppe von Knoten in der angegebenen Reihenfolge aus.
- *node*`.run(`*results-list*`)`führt den angegebenen Knoten aus.

Beim traditionellen Scripting kann eine Streamausführung mit dem Befehl `exit` und einem optionalen ganzzahligen Code beendet werden. Beispiel:

```
exit 1
```

Beim Python-Scripting wird dasselbe Ergebnis mit dem folgenden Script erreicht:

```
modeler.script.exit(1)
```

Zugriff auf Objekte über das Dateisystem und das Repository

Beim traditionellen Scripting können Sie einen vorhandenen Stream, ein vorhandenes Modell oder ein vorhandenes Ausgabeobjekt mit dem Befehl open öffnen: Beispiel:

```
var s
set s = open stream "c:/my streams/modeling.str"
```

Beim Python-Scripting gibt es die Klasse `TaskRunner`, auf die von der Sitzung zugegriffen werden kann und mit der ähnliche Tasks ausgeführt werden können. Beispiel:

```
taskrunner = modeler.script.session().getTaskRunner()
s = taskrunner.openStreamFromFile("c:/my streams/modeling.str", True)
```

Wenn Sie beim traditionellen Scripting ein Objekt speichern wollen, können Sie den Befehl `save` verwenden. Beispiel:

```
save stream s as "c:/my streams/new_modeling.str"
```

Die entsprechende Vorgehensweise bei einem Python-Script ist die Verwendung der Klasse `TaskRunner`. Beispiel:

```
taskrunner.saveStreamToFile(s, "c:/my streams/new_modeling.str")
```

Auf IBM SPSS Collaboration and Deployment Services Repository basierende Operationen werden beim traditionellen Scripting über die Befehle `retrieve` und `store` unterstützt. Beispiel:

```
var s
set s = retrieve stream "/my repository folder/my_stream.str"
store stream ^s as "/my repository folder/my_stream_copy.str"
```

Beim Python-Scripting wird die entsprechende Funktionalität über das der Sitzung zugeordnete Repository-Objekt abgerufen. Beispiel:

```
session = modeler.script.session()
repo = session.getRepository()
s = repo.retrieveStream("/my repository folder/my_stream.str", None, None, True)
repo.storeStream(s, "/my repository folder/my_stream_copy.str", None)
```

Anmerkung: Für den Repository-Zugriff muss die Sitzung mit einer gültigen Repository-Verbindung konfiguriert worden sein.

Streamoperationen

Für einige Befehle für Streamoperationen, die üblicherweise in IBM SPSS Modeler verwendet werden, gibt es funktional entsprechende Befehle in Python-Scripts. Dies könnte Ihnen dabei helfen, Ihre vorhandenen traditionellen SPSS Modeler-Scripts für die Verwendung in IBM SPSS Modeler 17 in Python-Scripts zu konvertieren.

Tabelle 284. Zuordnung von traditionellem Scripting zu Python-Scripting für Streamoperationen	
Traditionelles Scripting	**Python-Scripting**
`create stream DEFAULT_FILENAME`	*taskrunner*`.createStream(`*name*`, `*autoConnect*`, `*autoManage*`)`
`close stream`	*stream*`.close()`
`clear stream`	*stream*`.clear()`
`get stream stream`	Keine Entsprechung
`load stream path`	Keine Entsprechung
`open stream path`	*taskrunner*`.openStreamFromFile(`*path*`, `*autoManage*`)`
`save stream as path`	*taskrunner*`.saveStreamToFile(`*stream*`, `*path*`)`
`retreive stream path`	*repository*`.retreiveStream(`*path*`, `*version*`, `*label*`, `*autoManage*`)`
`store stream as path`	*repository*`.storeStream(`*stream*`, `*path*`, `*label*`)`

Modelloperationen

Für einige Befehle für Modelloperationen, die in IBM SPSS Modeler häufig verwendet werden, gibt es entsprechende Befehle beim Python-Scripting. Dies könnte Ihnen dabei helfen, Ihre vorhandenen traditionellen SPSS Modeler-Scripts für die Verwendung in IBM SPSS Modeler 17 in Python-Scripts zu konvertieren.

Tabelle 285. Zuordnung von traditionellem Scripting zu Python-Scripting für Modelloperationen	
Traditionelles Scripting	**Python-Scripting**
`open model path`	*taskrunner*`.openModelFromFile(`*path*`, `*autoManage*`)`
`save model as path`	*taskrunner*`.saveModelToFile(`*model*`, `*path*`)`
`retrieve model path`	*repository*`.retrieveModel(`*path*`, `*version*`, `*label*`, `*autoManage*`)`
`store model as path`	*repository*`.storeModel(`*model*`, `*path*`, `*label*`)`

Dokumentausgabeoperationen

Für einige Befehle für Dokumentausgabeoperationen, die in IBM SPSS Modeler häufig verwendet werden, gibt es entsprechende Befehle beim Python-Scripting. Dies könnte Ihnen dabei helfen, Ihre vorhandenen traditionellen SPSS Modeler-Scripts für die Verwendung in IBM SPSS Modeler 17 in Python-Scripts zu konvertieren.

Tabelle 286. Zuordnung von traditionellem Scripting zu Python-Scripting für Dokumentausgabeoperationen	
Traditionelles Scripting	**Python-Scripting**
`open output path`	*taskrunner*`.openDocumentFromFile(`*path*`, `*autoManage*`)`

Tabelle 286. Zuordnung von traditionellem Scripting zu Python-Scripting für Dokumentausgabeoperationen (Forts.)	
Traditionelles Scripting	**Python-Scripting**
`save output as path`	`taskrunner.saveDocumentToFile(output, path)`
`retrieve output path`	`repository.retrieveDocument(path, version, label, autoManage)`
`store output as path`	`repository.storeDocument(output, path, label)`

Weitere Unterschiede zwischen traditionellem Scripting und Python-Scripting

Traditionelle Scripts bieten Unterstützung zur Manipulation von IBM SPSS Modeler-Projekten. Python-Scripting unterstützt diese Funktion derzeit nicht.

Traditionelles Scripting bietet eine gewisse Unterstützung beim Laden von *Statusobjekten* (Kombinationen von Streams und Modellen). Statusobjekte werden seit IBM SPSS Modeler 8.0 nicht weiter unterstützt. Python-Scripting unterstützt keine Statusobjekte.

Python-Scripting bietet die folgenden zusätzlichen Funktionen, die beim traditionellen Scripting nicht verfügbar sind:

- Klassen- und Funktionsdefinitionen
- Fehlerbehandlung
- Fortgeschrittenere Ein-/Ausgabe-Unterstützung
- Externe und Fremdanbieter-Module

Bemerkungen

Die vorliegenden Informationen wurden für Produkte und Services entwickelt, die auf dem deutschen Markt angeboten werden. IBM stellt dieses Material möglicherweise auch in anderen Sprachen zur Verfügung. Für den Zugriff auf das Material in einer anderen Sprache kann eine Kopie des Produkts oder der Produktversion in der jeweiligen Sprache erforderlich sein.

Möglicherweise bietet IBM die in dieser Dokumentation beschriebenen Produkte, Services oder Funktionen in anderen Ländern nicht an. Informationen über die gegenwärtig im jeweiligen Land verfügbaren Produkte und Services sind beim zuständigen IBM Ansprechpartner erhältlich. Hinweise auf IBM Lizenzprogramme oder andere IBM Produkte bedeuten nicht, dass nur Programme, Produkte oder Services von IBM verwendet werden können. Anstelle der IBM Produkte, Programme oder Services können auch andere, ihnen äquivalente Produkte, Programme oder Services verwendet werden, solange diese keine gewerblichen oder anderen Schutzrechte von IBM verletzen. Die Verantwortung für den Betrieb von Produkten, Programmen und Services anderer Anbieter liegt beim Kunden.

Für in diesem Handbuch beschriebene Erzeugnisse und Verfahren kann es IBM Patente oder Patentanmeldungen geben. Mit der Auslieferung dieser Dokumentation ist keine Lizenzierung dieser Patente verbunden. Lizenzanforderungen sind schriftlich an folgende Adresse zu richten (Anfragen an diese Adresse müssen auf Englisch formuliert werden):

IBM Director of Licensing
IBM Europe, Middle East & Africa
Tour Descartes
2, avenue Gambetta
92066 Paris La Defense
USA

For license inquiries regarding double-byte (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.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

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 jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

Trotz sorgfältiger Bearbeitung können technische Ungenauigkeiten oder Druckfehler in dieser Veröffentlichung nicht ausgeschlossen werden. Die hier enthaltenen Informationen werden in regelmäßigen Zeitabständen aktualisiert und als Neuausgabe veröffentlicht. IBM kann ohne weitere Mitteilung jederzeit Verbesserungen und/oder Änderungen an den in dieser Veröffentlichung beschriebenen Produkten und/oder Programmen vornehmen.

Verweise in diesen Informationen auf Websites anderer Anbieter als IBM werden lediglich als Service für den Kunden bereitgestellt und stellen keinerlei Billigung des Inhalts dieser Websites dar. Das über diese Websites verfügbare Material ist nicht Bestandteil des Materials für dieses IBM Produkt. Die Verwendung dieser Websites geschieht auf eigene Verantwortung.

Werden an IBM Informationen eingesandt, können diese beliebig verwendet werden, ohne dass eine Verpflichtung gegenüber dem Einsender entsteht.

Lizenznehmer des Programms, die Informationen zu diesem Produkt wünschen mit der Zielsetzung: (i) den Austausch von Informationen zwischen unabhängig voneinander erstellten Programmen und anderen

Programmen (einschließlich des vorliegenden Programms) sowie (ii) die gemeinsame Nutzung der ausgetauschten Informationen zu ermöglichen, wenden sich an folgende Adresse:

IBM Director of Licensing
IBM Europe, Middle East & Africa
Tour Descartes
2, avenue Gambetta
92066 Paris La Defense
USA

Die Bereitstellung dieser Informationen kann unter Umständen von bestimmten Bedingungen - in einigen Fällen auch von der Zahlung einer Gebühr - abhängig sein.

Die Lieferung des in diesem Dokument beschriebenen Lizenzprogramms sowie des zugehörigen Lizenzmaterials erfolgt auf der Basis der IBM Rahmenvereinbarung bzw. der Allgemeinen Geschäftsbedingungen von IBM, der IBM Internationalen Nutzungsbedingungen für Programmpakete oder einer äquivalenten Vereinbarung.

Die angeführten Leistungsdaten und Kundenbeispiele dienen nur zur Illustration. Die tatsächlichen Ergebnisse beim Leistungsverhalten sind abhängig von der jeweiligen Konfiguration und den Betriebsbedingungen.

Alle Informationen zu Produkten anderer Anbieter stammen vonIBM den Anbietern der aufgeführten Produkte, deren veröffentlichten Ankündigungen oder anderen allgemein verfügbaren Quellen. IBM hat diese Produkte nicht getestet und kann die Genauigkeit der Leistung, der Kompatibilität oder anderer Ansprüche im Zusammenhang mit Nicht-IBM-Produkten nicht bestätigen. Fragen zu den Leistungsmerkmalen von Produkten anderer Anbieter als IBM sind an den jeweiligen Anbieter zu richten.

Aussagen über Pläne und Absichten von IBM unterliegen Änderungen oder können zurückgenommen werden und repräsentieren nur die Ziele von IBM.

Diese Veröffentlichung enthält Beispiele für Daten und Berichte des alltäglichen Geschäftsablaufs. Sie sollen nur die Funktionen des Lizenzprogramms illustrieren und können Namen von Personen, Firmen, Marken oder Produkten enthalten. Alle diese Namen sind frei erfunden; Ähnlichkeiten mit tatsächlichen Namen und Adressen sind rein zufällig.

Marken

IBM, das IBM Logo und ibm.com sind Marken oder eingetragene Marken der IBM Corporation in vielen Ländern weltweit registriert. Weitere Produkt- und Servicenamen können Marken von IBM oder anderen Unternehmen sein. Eine aktuelle Liste der IBM Marken finden Sie auf der Webseite "Copyright and trademark information" unter www.ibm.com/legal/copytrade.shtml.

Adobe, das Adobe Logo, PostScript und das PostScript Logo sind entweder eingetragene Marken oder Marken von Adobe Systems Incorporated in den USA und/oder anderen Ländern.

Intel, das Intel-Logo, Intel Inside, das Intel Inside-Logo, Intel Centrino, das Intel Centrino-Logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium und Pentium sind Marken oder eingetragene Marken der Intel Corporation oder ihrer Tochtergesellschaften in den USA oder anderen Ländern.

Linux ist eine eingetragene Marke von Linus Torvalds in den USA und/oder anderen Ländern.

Microsoft, Windows, Windows NT und das Windows-Logo sind Marken der Microsoft Corporation in den USA und/oder anderen Ländern.

UNIX ist eine eingetragene Marke von The Open Group in den USA und anderen Ländern.

Java und alle auf Java basierenden Marken und Logos sind Marken oder eingetragene Marken der Oracle Corporation und/oder ihrer verbundenen Unternehmen.

Bedingungen für Produktdokumentation

Die Berechtigungen zur Nutzung dieser Veröffentlichungen werden Ihnen auf der Basis der folgenden Bedingungen gewährt.

Anwendbarkeit

Diese Bedingungen sind eine Ergänzung der Nutzungsbedingungen auf der IBM Website.

Persönlicher Gebrauch

Sie dürfen diese Veröffentlichungen für Ihre persönliche, nicht kommerzielle Nutzung unter der Voraussetzung vervielfältigen, dass alle Eigentumsvermerke erhalten bleiben. Sie dürfen diese Veröffentlichungen oder Teile der Veröffentlichungen ohne ausdrückliche Genehmigung von IBM weder weitergeben oder anzeigen noch abgeleitete Werke davon erstellen.

Kommerzielle Nutzung

Sie dürfen diese Veröffentlichungen nur innerhalb Ihres Unternehmens und unter der Voraussetzung, dass alle Eigentumsvermerke erhalten bleiben, vervielfältigen, weitergeben und anzeigen. Sie dürfen diese Veröffentlichungen oder Teile der Veröffentlichungen ohne ausdrückliche Genehmigung von IBMaußerhalb Ihres Unternehmens weder vervielfältigen, weitergeben oder anzeigen noch abgeleitete Werke davon erstellen.

Rechte

Abgesehen von den hier gewährten Berechtigungen werden keine weiteren Berechtigungen, Lizenzen oder Rechte (veröffentlicht oder stillschweigend) in Bezug auf die Veröffentlichungen oder darin enthaltene Informationen, Daten, Software oder geistiges Eigentum gewährt.

IBM behält sich das Recht vor, die hierin gewährten Berechtigungen nach eigenem Ermessen zurückzuziehen, wenn sich die Nutzung der Veröffentlichungen für IBM als nachteilig erweist oder wenn die obigen Nutzungsbestimmungen nicht genau befolgt werden.

Sie dürfen diese Informationen nur in Übereinstimmung mit allen anwendbaren Gesetzen und Vorschriften, einschließlich aller US-amerikanischen Exportgesetze und Verordnungen, herunterladen und exportieren.

IBM übernimmt keine Gewährleistung für den Inhalt dieser Veröffentlichungen. Diese Veröffentlichungen werden auf der Grundlage des gegenwärtigen Zustands (auf "as-is"-Basis) und ohne eine ausdrückliche oder stillschweigende Gewährleistung für die Handelsüblichkeit, die Verwendungsfähigkeit für einen bestimmten Zweck oder die Freiheit von Rechten Dritter zur Verfügung gestellt.

Index

A

Ableitungsknoten
  Eigenschaften 168
aggregatenode, Eigenschaften 127
Aggregatknoten
  Eigenschaften 127
Analyseknoten
  Eigenschaften 399
analysisnode, Eigenschaften 399
Analytic Server-Quellenknoten
  Eigenschaften 92
Anhangknoten
  Eigenschaften 127
Anmerkungen 21
Anomalieerkennungsmodelle
  Knoten, Scripteigenschaften 226, 345
anomalydetectionnode, Eigenschaften 226
Anonymisierungsknoten
  Eigenschaften 159
anonymizenode, Eigenschaften 159
Anweisungen 21
appendnode, Eigenschaften 127
applyassociationrulesnode, Eigenschaften 346
applyautoclusternode, Eigenschaften 349
applyautonumericnode, Eigenschaften 349
applycarmanode, Eigenschaften 350
applyextension, Eigenschaften 353
applyfactornode, Eigenschaften 354
applyfeatureselectionnode, Eigenschaften 354
applygeneralizedlinearnode, Eigenschaften 355
applykmeansnode, Eigenschaften 357
applykohonennode, Eigenschaften 357
applylinearasnode, Eigenschaften 358
applylinearnode, Eigenschaften 357
applymslogisticnode, Eigenschaften 369
applymsneuralnetworknode, Eigenschaften 369
applymsregressionnode, Eigenschaften 369
applymssequenceclusternode, Eigenschaften 369
applymstimeseriesnode, Eigenschaften 369
applymstreenode, Eigenschaften 369
applynetezzabayesnode, Eigenschaften 396
applynetezzadectreenode, Eigenschaften 396
applynetezzadivclusternode, Eigenschaften 396
applynetezzakmeansnode, Eigenschaft 396
applynetezzaknnnode, Eigenschaften 396
applynetezzalineregressionnode, Eigenschaften 396
applynetezzanaivebayesnode, Eigenschaften 396
applynetezzapcanode, Eigenschaften 396
applynetezzaregtreenode, Eigenschaften 396
applyneuralnetworknode, Eigenschaften 359
applyocsvm, Eigenschaften 360
applyoraabnnode, Eigenschaften 378
applyorakmeansnode, Eigenschaften 378
applyoranbnode, Eigenschaften 378
applyoraoclusternode, Eigenschaften 378
applyorasvmnode, Eigenschaften 378
applyrandomtrees, Eigenschaften 361
applyregressionnode, Eigenschaften 362
applysequencenode, Eigenschaften 362
applysvmnode, Eigenschaften 363
applytreeas, Eigenschaften 364
applytwostepAS, Eigenschaften 365
applytwostepnode, Eigenschaften 365
applyxgboostlinearnode, Eigenschaften 366
Apriori-Modelle
  Knoten, Scripteigenschaften 227, 345
apriorinode, Eigenschaften 227
Argumente
  Befehlsdatei 75
  Serververbindung 72
  System 70
  Verbindung zu IBM SPSS Analytic Server Repository 74
  Verbindung zu IBM SPSS Collaboration and Deployment Services Repository 74
Argumente übergeben 22
AS-Zeitintervallknoten
  Eigenschaften 164
associationrulesnode, Eigenschaften 229
Assoziationsregelknoten
  Eigenschaften 229
Assoziationsregelknotennugget
  Eigenschaften 346
astimeintervalsnode, Eigenschaften 164
Attribute definieren 27
Attribute hinzufügen 27
Aufschläger
  Befehlszeilenargumente 72
Ausführungsreihenfolge
  mit Scripts ändern 55
Ausgabeknoten
  Scripteigenschaften 399
Ausgabeobjekte
  Scriptnamen 473
Ausgeblendete Variablen 28
Auswahlknoten
  Eigenschaften 145
autoclassifiernode, Eigenschaften 232
autoclusternode, Eigenschaften 235
Autom. Cluster, Modelle
  Knoten, Scripteigenschaften 349
Automatische Datenaufbereitung
  Eigenschaften 160
Automatisches Klassifikationsmerkmal, Modelle
  Knoten, Scripteigenschaften 347
autonumericnode, Eigenschaften 237
Autonumerisch, Modelle
  Knoten, Scripteigenschaften 349
Autonumerische Modelle
  Knoten, Scripteigenschaften 237

B

balancenode, Eigenschaften 129

Balancierungsknoten
Eigenschaften 129
Bayes-Netzmodelle
Knoten, Scripteigenschaften 239, 349
bayesnet, Eigenschaften 239
Bedingte Ausführung von Streams 6, 11
Befehlszeile
IBM SPSS Modeler ausführen 69
Liste der Argumente 70, 72, 74
mehrere Argumente 75
Parameter 71
Scripting 59
Beispiele 23
Benutzereingabeknoten
Eigenschaften 118
Berichtknoten
Eigenschaften 409
binningnode, Eigenschaften 165
buildr, Eigenschaften 241

C

C&R-Baummodelle
Knoten, Scripteigenschaften 244, 350
C5.0-Modelle
Knoten, Scripteigenschaften 242, 350
c50node, Eigenschaften 242
CARMA-Modelle
Knoten, Scripteigenschaften 243, 350
carmanode, Eigenschaften 243
cartnode, Eigenschaften 244
CHAID-Modelle
Knoten, Scripteigenschaften 247, 351
chaidnode, Eigenschaften 247
clear generated palette, Befehl 59
CLEM
Scripting 1
Codeblöcke 22
Codierte Kennwörter
zu Scripts hinzufügen 58
collectionnode, Eigenschaften 200
Cox-Regressionsmodelle
Knoten, Scripteigenschaften 250, 351
coxregnode, Eigenschaften 250
CPLEX-Optimierung (Knoten)
Eigenschaften 129
cplexoptnode, Eigenschaften 129

D

Data Audit-Knoten
Eigenschaften 400
Data Collection-Exportknoten
Eigenschaften 429
Data Collection-Quellenknoten
Eigenschaften 99
dataauditnode, Eigenschaften 400
databaseexportnode, Eigenschaften 424
Datei (fest), Knoten
Eigenschaften 107
Datenbankexportknoten
Eigenschaften 424
Datenbankknoten
Datenbankknoten *(Forts.)*
Eigenschaften 97
Datenbankmodellierung 367
decisionlist, Eigenschaften 252
derive_stbnode
Eigenschaften 133
derivenode, Eigenschaften 168
Diagramme 31
Diagrammknoten
Scripteigenschaften 199
Diagrammtafelknoten
Eigenschaften 204
Dichotomknoten
Eigenschaften 182
directedwebnode, Eigenschaften 223
discriminantnode, Eigenschaften 254
Diskriminanzmodelle
Knoten, Scripteigenschaften 254, 352
distinctnode, Eigenschaften 135
Duplikatknoten
Eigenschaften 135

E

E-Plot-Knoten
Eigenschaften 220
Eigenschaften
allgemeine Scripts 80
Datenbankmodellierungsknoten 367
Filterknoten 78
Scripting 77–79, 225, 345, 421
Stream 81
Superknoten 469
Eigenschaften des Knotens "cognosimport" 93
Eigenschaften des Knotens "gsdata_import" 111
Eigenschaften des Knotens "Space-Time-Boxes" 133
Eigenschaften des Knotens "tm1import" 116
Eigenschaften festlegen 34
Eigenschaften von "applyanomalydetectionnode" 345
Eigenschaften von "applyapriorinode" 345
Eigenschaften von "applyautoclassifiernode" 347
Eigenschaften von "applybayesnetnode" 349
Eigenschaften von "applyc50node" 350
Eigenschaften von "applycartnode" 350
Eigenschaften von "applychaidnode" 351
Eigenschaften von "applycoxregnode" 351
Eigenschaften von "applydecisionlistnode" 352
Eigenschaften von "applydiscriminantnode" 352
Eigenschaften von "applygle" 356
Eigenschaften von "applyglmmnode" 355
Eigenschaften von "applygmm" 356
Eigenschaften von "applyknnnode" 357
Eigenschaften von "applylogregnode" 358
Eigenschaften von "applylsvmnode" 358
Eigenschaften von "applyneuralnetnode" 359
Eigenschaften von "applyoradecisiontreenode" 378
Eigenschaften von "applyoranmfnode" 378
Eigenschaften von "applyquestnode" 360
Eigenschaften von "applyr" 361
Eigenschaften von "applyselflearningnode" 362
Eigenschaften von "applystpnode" 363
Eigenschaften von "applytcmnode" 363
Eigenschaften von "applytimeseriesnode" 364
Eigenschaften von "applyxgboosttreenode" 365

Eigenschaften von "asexport" 421
Eigenschaften von "asimport" 92
Eigenschaften von "autodataprepnode" 160
Eigenschaften von "databasenode" 97
Eigenschaften von "datacollectionexportnode" 429
Eigenschaften von "datacollectionimportnode" 99
Eigenschaften von "distributionnode" 201
Eigenschaften von "eplotnode" 220
Eigenschaften von "evaluationnode" 202
Eigenschaften von "excelexportnode" 430, 432
Eigenschaften von "extensionexportnode" 431
Eigenschaften von "filternode" 174
Eigenschaften von "fixedfilenode" 107
Eigenschaften von "gle" 275
Eigenschaften von "glmmnode" 269
Eigenschaften von "gmm" 445, 450
Eigenschaften von "graphboardnode" 204
Eigenschaften von "hdbscannode" 446
Eigenschaften von "hdbscannugget" 366
Eigenschaften von "histogramnode" 209
Eigenschaften von "isotonicasnode" 463
Eigenschaften von "jsonimportnode" 112
Eigenschaften von "kdeapply" 366
Eigenschaften von "kdemodel" 448
Eigenschaften von "msassocnode" 367
Eigenschaften von "mssequenceclusternode" 367
Eigenschaften von "multilayerperceptronnode" 465
Eigenschaften von "multiplotnode" 215
Eigenschaften von "netezzabayesnode" 379
Eigenschaften von "netezzadectreenode" 379
Eigenschaften von "netezzadivclusternode" 379
Eigenschaften von "netezzaglmnode" 379
Eigenschaften von "netezzakmeansnode" 379
Eigenschaften von "netezzaknnnode" 379
Eigenschaften von "netezzalineregressionnode" 379
Eigenschaften von "netezzanaivebayesnode" 379
Eigenschaften von "netezzapcanode" 379
Eigenschaften von "netezzaregtreenod" 379
Eigenschaften von "netezzatimeseriesnode" 379
Eigenschaften von "ocsvmnode" 451
Eigenschaften von "oraabnnode" 372
Eigenschaften von "oraapriorinode" 372
Eigenschaften von "oradecisiontreenode" 372
Eigenschaften von "oraglmnode" 372
Eigenschaften von "orakmeansnode" 372
Eigenschaften von "oranbnode" 372
Eigenschaften von "oranmfnode" 372
Eigenschaften von "oraoclusternode" 372
Eigenschaften von "orasvmnode" 372
Eigenschaften von "outputfilenode" 433
Eigenschaften von "plotnode" 216
Eigenschaften von "Rprocessnode" 142
Eigenschaften von "sasexportnode" 434
Eigenschaften von "sasimportnode" 112
Eigenschaften von "smotenode" 456
Eigenschaften von "statisticsexportnode" 443
Eigenschaften von "statisticsimportnode" 5, 441
Eigenschaften von "statisticsmodelnode" 442
Eigenschaften von "statisticsoutputnode" 443
Eigenschaften von "statisticstransformnode" 441
Eigenschaften von "streamingtimeseries" 148
Eigenschaften von "streamingts" 155
Eigenschaften von "tcmnode" 322
Eigenschaften von "timeintervalsnode" 183
Eigenschaften von "timeplotnode" 219
Eigenschaften von "timeseriesnode" 335
Eigenschaften von "tsnenode" 221, 457
Eigenschaften von "twostepAS" 342
Eigenschaften von "twostepnode" 341
Eigenschaften von "typenode" 5, 191
Eigenschaften von "variablefilenode" 119
Eigenschaften von "webnode" 223
Eigenschaften von "xgboostasnode" 465
Eigenschaften von "xgboostlinearnode" 459
Eigenschaften von "xgboosttreenode" 460
Eigenschaften von "xmlexportnode" 439
Eigenschaften von "xmlimportnode" 125
Eigenschaften von applyts 363
Ensembleknoten
    Eigenschaften 172
ensemblenode, Eigenschaften 172
Entscheidungslistenmodelle
    Knoten, Scripteigenschaften 252, 352
Erweiterungsausgabeknoten
    Eigenschaften 402
Erweiterungsexportknoten
    Eigenschaften 431
Erweiterungsimportknoten
    Eigenschaften 104
Erweiterungsmodellknoten
    Knoten, Scripteigenschaften 256
Erweiterungstransformationsknoten
    Eigenschaften 137
Evaluierungsknoten
    Eigenschaften 202
Excel-Exportknoten
    Eigenschaften 430, 432
Excel-Quellenknoten
    Eigenschaften 103
excelimportnode, Eigenschaften 103
Exportknoten
    Knoten, Scripteigenschaften 421
exportModelToFile 47
extensionimportnode, Eigenschafen 104
extensionmodelnode, Eigenschaften 256
extensionoutputnode, Eigenschaften 402
extensionprocessnode, Eigenschaften 137

F

factornode, Eigenschaften 259
featureselectionnode, Eigenschaften 5, 261
Fehlerprüfung
    Scripting 58
Felder
    in Scripts inaktivieren 199
Felder ordnen (Knoten)
    Eigenschaften 178
Felder umordnen, Knoten
    Eigenschaften 178
Feldnamen
    Groß- und Kleinschreibung ändern 55
fillernode, Eigenschaften 173
Filterknoten
    Eigenschaften 174
Flags
    Befehlszeilenargumente 69
    mehrere Flags kombinieren 75

Flatfile-Knoten
Eigenschaften 433
flatfilenode, Eigenschaften 433
for, Befehl 55
Füllerknoten
Eigenschaften 173
Funktionen
bedingte Befehle 477
Dokumentausgabeoperationen 484
Knotenoperationen 481
Kommentare 476
Literale 476
Modelloperationen 484
Objektreferenzen 476
Operatoren 476
Schleifenbefehle 477
Streamoperationen 483

G

generated, Schlüsselwort 59
Generierte Modelle
Scriptnamen 471, 473
genlinnode, Eigenschaften 263
Georäumlicher Quellenknoten
Eigenschaften 111
Gerichteter Netzdiagrammknoten
Eigenschaften 223
GLE-Modelle
Knoten, Scripteigenschaften 275, 356
GLMM-Modelle
Knoten, Scripteigenschaften 269, 355
Globalwerteknoten
Eigenschaften 411

H

HDBSCAN, Knoten
Eigenschaften 446
Histogrammknoten
Eigenschaften 209
historynode, Eigenschaften 175

I

IBM Cognos TM1-Quellenknoten
Eigenschaften 115, 116
IBM Cognos-Quellenknoten
Eigenschaften 93
IBM SPSS Analytic Server Repository
Befehlszeilenargumente 74
IBM SPSS Collaboration and Deployment Services Repository
Befehlszeilenargumente 74
Scripting 56
IBM SPSS Modeler
über Befehlszeile ausführen 69
IBM SPSS Statistics-Ausgabeknoten
Eigenschaften 443
IBM SPSS Statistics-Exportknoten
Eigenschaften 443
IBM SPSS Statistics-Modelle
Knoten, Scripteigenschaften 442
IBM SPSS Statistics-Quellenknoten
Eigenschaften 441
IBM SPSS Statistics-Transformationsknoten
Eigenschaften 441
IDs 21
Iterationsschlüssel
Schleifen in Scripts 8
Iterationsvariable
Schleifen in Scripts 10

J

JSON-Inhaltsmodell 63
JSON-Quellenknoten
Eigenschaften 112
Jython 17

K

K-Means-AS-Modelle
Knoten, Scripteigenschaften 283, 463
K-Means-Modelle
Knoten, Scripteigenschaften 282, 357
Kartenvisualisierungsknoten
Eigenschaften 210
KDE-Modelle
Knoten, Scripteigenschaften 366
KDE-Modellierungsknoten
Eigenschaften 448
kdeexport, Eigenschaften 404, 449
Kennwörter
codiert 72
zu Scripts hinzufügen 58
Klasse definieren 27
Klasse erstellen 27
Klassierknoten
Eigenschaften 165
kmeansasnode, Eigenschaften 283, 463
kmeansnode, Eigenschaften 282
KNN-Modelle
Knoten, Scripteigenschaften 357
knnnode, Eigenschaften 284
Knoten
Ersetzung 38
importieren 38
Information 40
Links für Knoten aktivieren 36
Links für Knoten aufheben 36
löschen 38
Namensreferenz 471
Schleifen in Scripts 55
Knoten "Autom. Cluster"
Knoten, Scripteigenschaften 235
Knoten "Autom. Klassifikationsmerkmal"
Knoten, Scripteigenschaften 232
Knoten "Gaußsche Mischverteilung"
Eigenschaften 445, 450
Knoten "Isotonisch-AS"
Eigenschaften 463
Knoten "KDE-Simulation"

Knoten "KDE-Simulation" *(Forts.)*
Eigenschaften 404, 449
Knoten "One-Class SVM"
Eigenschaften 451
Knoten "XGBoost Linear"
Eigenschaften 459
Knoten "XGBoost Tree"
Eigenschaften 460
Knoten "XGBoost-AS"
Eigenschaften 465
Knoten erstellen 35, 36, 38
Knoten referenzieren
Eigenschaften festlegen 34
Knoten suchen 33
Knoten suchen 33
Knoten, Scripteigenschaften
Exportknoten 421
Modellierungsknoten 225
Modellnuggets 345
Kohonen-Modelle
Knoten, Scripteigenschaften 286, 357
kohonennode, Eigenschaften 286
Koordinatensystemreprojizierung
Eigenschaften 179

L

Lernfähige Antwortmodelle
Knoten, Scripteigenschaften 312, 362
Linear-AS-Modelle
Knoten, Scripteigenschaften 289, 358
Linear-AS, Eigenschaften 289
linear, Eigenschaften 287
lineare Modelle
Knoten, Scripteigenschaften 287, 357
Lineare Regressionsmodelle
Knoten, Scripteigenschaften 308, 361, 362
Listen 18
logistische Regressionsmodelle
Knoten, Scripteigenschaften 291, 358
logregnode, Eigenschaften 291
lowertoupper, Funktion 55
LSVM-Modelle
Knoten, Scripteigenschaften 297, 358
lsvmnode, Eigenschaften 297

M

mapvisualization, Eigenschaften 210
Mathematische Methoden 23
Matrixknoten
Eigenschaften 405
matrixnode, Eigenschaften 405
meansnode, Eigenschaften 407
mergenode, Eigenschaften 138
Merkmalauswahlmodelle
Knoten, Scripteigenschaften 261, 354
Scripting 5
zuweisen 5
Methoden definieren 27
Microsoft-Modelle
Knoten, Scripteigenschaften 367, 369
Migration
Migration *(Forts.)*
-Repository 483
allgemeine Unterschiede 475
auf Objekte zugreifen 483
Ausgabetypen 478
Befehle 475
Dateisystem 483
Eigenschaften abrufen 479
Eigenschaften festlegen 479
Eigenschaftsnamen 479
Funktionen 475
Knotenreferenzen 479
Knotentypen 478
Modelltypen 478
Schleifenbefehle 481
Scripting-Kontext 475
Sonstiges 485
Streams ausführen 482
Streams bearbeiten 480
Streams, Ausgabe und Modellmanager entfernen 39
Übersicht 475
Variablen 478
Mittelwertknoten
Eigenschaften 407
Modelle
Scriptnamen 471, 473
Modellierungsknoten
Knoten, Scripteigenschaften 225
Modellnuggets
Knoten, Scripteigenschaften 345
Scriptnamen 471, 473
Modellobjekte
Scriptnamen 471, 473
MS - Lineare Regression
Knoten, Scripteigenschaften 367, 369
MS - Logistische Regression
Knoten, Scripteigenschaften 367, 369
MS - Neuronales Netz
Knoten, Scripteigenschaften 367, 369
MS Sequenz-Clustering
Knoten, Scripteigenschaften 369
MS Time Series
Knoten, Scripteigenschaften 369
MS-Entscheidungsbaum
Knoten, Scripteigenschaften 367, 369
msbayesnode, Eigenschaften 367
msclusternode, Eigenschaften 367
mslogisticnode, Eigenschaften 367
msneuralnetworknode, Eigenschaften 367
msregressionnode, Eigenschaften 367
mstimeseriesnode, Eigenschaften 367
mstreenode, Eigenschaften 367
MultiLayerPerceptron-AS-Knoten
Eigenschaften 465
Multiplotknoten
Eigenschaften 215
Multiset-Befehl 78

N

Nächste-Nachbarn-Modelle
Knoten, Scripteigenschaften 284
Netezza - Divisives Clustering, Modelle

Netezza - Divisives Clustering, Modelle *(Forts.)*
Knoten, Scripteigenschaften 379, 396
Netezza - Lineare Regression, Modelle
Knoten, Scripteigenschaften 379, 396
Netezza - Naive Bayes, Modelle
Knoten, Scripteigenschaften 379, 396
Netezza-Bayes-Netzmodelle
Knoten, Scripteigenschaften 379, 396
Netezza-Entscheidungsbaummodelle
Knoten, Scripteigenschaften 379, 396
Netezza-K-Means-Modelle
Knoten, Scripteigenschaften 379, 396
Netezza-KNN-Modelle
Knoten, Scripteigenschaften 379, 396
Netezza-Modelle
Knoten, Scripteigenschaften 379
Netezza-PCA-Modelle
Knoten, Scripteigenschaften 379, 396
Netezza-Regressionsbaum, Modelle
Knoten, Scripteigenschaften 379, 396
Netezza-Zeitreihenmodelle
Knoten, Scripteigenschaften 379
Netzdiagrammknoten
Eigenschaften 223
neuralnetnode, Eigenschaften 298
neuralnetworknode, Eigenschaften 301
Neuronale Netze
Knoten, Scripteigenschaften 301, 359
neuronale Netzmodelle
Knoten, Scripteigenschaften 298, 359
Nicht-ASCII-Zeichen 25
Nuggets
Knoten, Scripteigenschaften 345
numericpredictornode, Eigenschaften 237

O

Objektorientierung 26
Operationen 18
oraainode, Eigenschaften 372
Oracle Adaptive Bayes-Modelle
Knoten, Scripteigenschaften 372, 378
Oracle AI-Modelle
Knoten, Scripteigenschaften 372
Oracle Apriori-Modelle
Knoten, Scripteigenschaften 372, 378
Oracle Decision Tree-Modelle
Knoten, Scripteigenschaften 372, 378
Oracle KMeans-Modelle
Knoten, Scripteigenschaften 372, 378
Oracle Naive Bayes-Modelle
Knoten, Scripteigenschaften 372, 378
Oracle NMF-Modelle
Knoten, Scripteigenschaften 372, 378
Oracle O-Cluster
Knoten, Scripteigenschaften 372, 378
Oracle-MDL-Modelle
Knoten, Scripteigenschaften 372, 378
Oracle-Modelle
Knoten, Scripteigenschaften 372
Oracle-SVM-Modelle
Knoten, Scripteigenschaften 372, 378
oramdlnode, Eigenschaften 372

P

Parameter
Scripting 18
Superknoten 469
partitionnode, Eigenschaften 176
Partitionsknoten
Eigenschaften 176
PCA-/Faktormodelle
Knoten, Scripteigenschaften 259, 354
PCA-Modelle
Knoten, Scripteigenschaften 259, 354
Plotknoten
Eigenschaften 216
Python
Scripting 18
Python-Modelle
Knoten, Scripteigenschaften 360, 365, 366
Scripting-Eigenschaften für Knoten "Gaußsche Misch-verteilung" 356

Q

Quellenknoten
Eigenschaften 85
QUEST-Modelle
Knoten, Scripteigenschaften 303, 360
questnode, Eigenschaften 303

R

R-Erstellungsknoten
Knoten, Scripteigenschaften 241
R-Transformationsknoten
Eigenschaften 142
Random Forest-Knoten
Eigenschaften 454
Random Trees-Modelle
Knoten, Scripteigenschaften 306, 361
randomtrees, Eigenschaften 306
Räumliche temporale Vorhersage, Knoten
Eigenschaften 314
reclassifynode, Eigenschaften 177
regressionnode, Eigenschaften 308
reordernode, Eigenschaften 178
reportnode, Eigenschaften 409
reprojectnode, Eigenschaften 179
Reprojizierungsknoten
Eigenschaften 179
restructurenode, Eigenschaften 179
retrieve, Befehl 56
RFM-Aggregat (Knoten)
Eigenschaften 140
RFM-Analyse (Knoten)
Eigenschaften 180
rfmaggregatenode, Eigenschaften 140
rfmanalysisnode, Eigenschaften 180
rfnode, Eigenschaften 454
Routput, Knoten
Eigenschaften 410
routputnode, Eigenschaften 410

S

Sammlungsknoten
  Eigenschaften 200
samplenode, Eigenschaften 143
SAS-Exportknoten
  Eigenschaften 434
SAS-Quellenknoten
  Eigenschaften 112
Schleifen
  Verwendung in Scripts 55
Schleifen in Streams verwenden 6, 7
Scripting
  allgemeine Eigenschaften 80
  Ausführung 12
  Ausgabeknoten 399
  bedingte Ausführung 6, 11
  Benutzerschnittstelle 1, 4, 5
  Diagramme 31
  Diagrammknoten 199
  Fehlerprüfung 58
  Felder auswählen 10
  in der Befehlszeile 59
  in Superknoten 5
  Iterationsschlüssel 8
  Iterationsvariable 10
  Kompatibilität mit früheren Versionen 59
  Kontext 32
  Merkmalauswahlmodelle 5
  Python-Scripting 476, 477, 481, 483, 484
  Standalone-Scripts 1, 31
  Streamausführungsreihenfolge 55
  Streams 1, 31
  Superknotenscripts 1, 31
  Superknotenstreams 31
  Syntax 18, 19, 21–23, 25–28
  traditionelles Scripting 476, 477, 481, 483, 484
  Übersicht 1, 17
  Unterbrechung 12
  verwendete Abkürzungen 78
  Verwendung von Schleifen 7
  Visuelle Schleife 6
Scripting-API
  auf generierte Objekte zugreifen 47
  Beispiel 43
  Einführung 43
  Fehlerbehandlung 48
  Globalwerte 53
  mehrere Streams 54
  Metadaten 44
  Sitzungsparameter 49
  Standalone-Scripts 54
  Streamparameter 49
  Suche 43
  Superknotenparameter 49
  Verzeichnis abrufen 43
Scripts
  bedingte Ausführung 6, 11
  Felder auswählen 10
  Iterationsschlüssel 8
  Iterationsvariable 10
  Schleifenbefehle 6, 7
  speichern 1
  Textdateien importieren 1
Scripts ausführen 12
Scripts unterbrechen 12
selectnode, Eigenschaften 145
sequencenode, Eigenschaften 311
Sequenzmodelle
  Knoten, Scripteigenschaften 311, 362
setglobalsnode, Eigenschaften 411
settoflagnode, Eigenschaften 182
Sicherheit
  Codierte Kennwörter 58, 72
simevalnode, Eigenschaften 412
simfitnode, Eigenschaften 413
simgen-Knoten
  Eigenschaften 113
simgennode, Eigenschaften 113
Simulationsanpassungsknoten
  Eigenschaften 413
Simulationsevaluierungsknoten
  Eigenschaften 412
Simulationsgenerierungsknoten
  Eigenschaften 113
Slotparameter 5, 77, 79
SLRM-Modelle
  Knoten, Scripteigenschaften 312, 362
slrmnode, Eigenschaften 312
SMOTE-Knoten
  Eigenschaften 456
Sortierknoten
  Eigenschaften 146
sortnode, Eigenschaften 146
spacetimeboxes, Eigenschaften 146
Standalone-Scripts 1, 4, 31
statisticsnode, Eigenschaften 414
Statistikknoten
  Eigenschaften 414
STB-Knoten
  Eigenschaften 133, 146
Stichprobenknoten
  Eigenschaften 143
store, Befehl 56
STP-Knoten
  Eigenschaften 314
STP-Knotennugget
  Eigenschaften 363
stpnode, Eigenschaften 314
stream.nodes, Eigenschaft 55
Streamausführungsreihenfolge
  mit Scripts ändern 55
Streaming-Zeitreihenknoten
  Eigenschaften 155
Streaming-Zeitreihenmodelle
  Knoten, Scripteigenschaften 148
Streams
  ändern 35
  Ausführung 31
  bedingte Ausführung 6, 11
  Eigenschaften 81
  Multiset-Befehl 77
  Schleifenbefehle 6, 7
  Scripting 1, 31
Streams ändern 35, 38
Streams ausführen 31
Strukturierte Eigenschaften 78

Superknoten
Eigenschaften 469
Eigenschaften festlegen 469
Parameter 469
Scripting 469
Scripts 1, 5, 6, 31
Stream 31
Streams 31
SVM-Modelle
Knoten, Scripteigenschaften 320, 363
svmnode, Eigenschaften 320
System
Befehlszeilenargumente 70

T

t-SNE-Knoten
Eigenschaften 221, 457
Tabelleninhaltsmodell 60
Tabellenknoten
Eigenschaften 415
tablenode, Eigenschaften 415
TCM-Modelle
Knoten, Scripteigenschaften 363
Temporale kausale Modelle
Knoten, Scripteigenschaften 322
tm1odataimport, Knoteneigenschaften 115
Transformationsknoten
Eigenschaften 418
transformnode, Eigenschaften 418
Transponierknoten
Eigenschaften 189
transposenode, Eigenschaften 189
Traversieren durch Knoten 38
Tree-AS-Modelle
Knoten, Scripteigenschaften 338, 364
treeas, Eigenschaften 338
ts, Eigenschaften 327
TWC-Importquellenknoten
Eigenschaften 117
twcimport, Knoteneigenschaften 117
TwoStep AS-Modelle
Knoten, Scripteigenschaften 342, 365
TwoStep-Modelle
Knoten, Scripteigenschaften 341, 365
Typknoten
Eigenschaften 191

U

Umcodierungsknoten
Eigenschaften 177
Umstrukturierungsknoten
Eigenschaften 179
userinputnode, Eigenschaften 118

V

Variable Datei (Knoten)
Eigenschaften 119
Variablen
Scripting 18
verallgemeinerte lineare Modelle
Knoten, Scripteigenschaften 263, 355
Verallgemeinerte lineare Netezza-Modelle
Knoten, Scripteigenschaften 379
Verallgemeinerte lineare Oracle-Modelle
Knoten, Scripteigenschaften 372
Vererbung 28
Verlaufsknoten
Eigenschaften 175
Verteilungsknoten
Eigenschaften 201

X

XML-Exportknoten
Eigenschaften 439
XML-Inhaltsmodell 62
XML-Quellenknoten
Eigenschaften 125

Z

Zeichenfolgefunktionen 55
Zeichenfolgen
Groß- und Kleinschreibung ändern 55
Zeitdiagrammknoten
Eigenschaften 219
Zeitintervallknoten
Eigenschaften 183
Zeitreihenmodelle
Knoten, Scripteigenschaften 327, 335, 363, 364
Zugriff auf Ergebnisse der Streamausführung
JSON-Inhaltsmodell 63
Tabelleninhaltsmodell 60
XML-Inhaltsmodell 62
Zugriff auf Streamausführungsergebnisse
JSON-Inhaltsmodell 63
Tabelleninhaltsmodell 60
XML-Inhaltsmodell 62
Zusammenführungsknoten
Eigenschaften 138