IBM SPSS Modeler 18.1 Python スクリプトとオートメーション・ガイド

注記

本書および本書で紹介する製品をご使用になる前に、 383 ページの『特記事項』に記載されている情報をお読みください。

本書は、IBM SPSS Modeler バージョン 18 リリース 0 モディフィケーション 0 および新しい版で明記されない限り、以降のすべてのリリースおよびモディフィケーションに適用されます。

お客様の環境によっては、資料中の円記号がバックスラッシュと表示されたり、バックスラッシュが円記号と表示されたりする場合があります。

原典： IBM SPSS Modeler 18.1 Python Scripting and Automation Guide

発行： 日本アイ・ビー・エム株式会社

担当： トランスレーション・サービス・センター

目次

第 1 章 スクリプトとスクリプト言語

スクリプトの概要

IBM® SPSS® Modeler のスクリプトは、ユーザー・インターフェースのプロセスを自動化する強力なツールです。スクリプトで、マウスやキーボードを使用した場合と同じ種類のアクションを実行できます。また、頻繁に繰り返したり手動で実行するのに時間がかかるタスクを自動化するために使用できます。

次の処理にスクリプトを使用できます。

- ストリームでノードを実行する特定の順序を指定する。
- CLEM (Control Language for Expression Manipulation) のサブセットを使用して、ノードにプロパティーを設定したり、フィールドを作成したりする。
- 通常はユーザーとの対話によって実行される一連の操作 (例えば、モデルを作成してテストするなど) を自動化する。
- ユーザーとの対話を高度に必要とする複雑な処理 (例えば、モデルの生成とテストを繰り返す交差検証手順など) を設定する。
- ストリームを操作する処理 (例えば、モデル学習ストリームの取得や実行、対応するモデル・テスト・ストリームの自動生成など) を設定する。

この章では、ストリームレベルのスクリプト、スタンドアロン スクリプト、および IBM SPSS Modeler インターフェースのスーパーノード内のスクリプトに関する概要の説明と例を記述しています。スクリプト言語、構文、およびコマンドは、以後の章で説明します。

注:

IBM SPSS Modeler 内の IBM SPSS Statistics で作成されたスクリプトはインポートおよび実行できません。

スクリプトの種類

IBM SPSS Modeler では、次の 3 種類のスクリプトが使用されます。

- ストリーム・スクリプト は、ストリーム・プロパティーとして格納されるため、特定のストリームと一緒に保存およびロードされます。例えば、モデル・ナゲットの学習と適用のプロセスを自動化するストリーム・スクリプトを書くことができます。また、特定のストリームが実行されたときは常に、そのストリームのキャンバスの内容ではなく、スクリプトが実行されるように指定することもできます。
- スタンドアロン スクリプトは、どのストリームとも関連付けがなく、外部のテキスト ファイルに保存されます。スタンドアロン スクリプトは、例えば、複数のストリームを一緒に操作する場合に使用できます。
- スーパーノード スクリプトは、スーパーノード ストリーム・プロパティーとして格納されます。スーパーノード・スクリプトは、ターミナル・スーパーノードでのみ使用可能です。スーパーノード スクリプトは、スーパーノードの内容の実行順序を制御するのに使用できます。ターミナル以外の (ソースまたはプロセス) スーパーノードの場合、ストリーム・スクリプト内で直接、スーパーノードまたはスーパーノード内のノードにプロパティーを定義できます。

ストリーム・スクリプト

スクリプトを使用して特定のストリーム内の操作をカスタマイズできます。また、スクリプトをそのストリームとともに保存することができます。ストリーム・スクリプトは、ストリーム内のターミナル・ノードの、特定の実行順序を指示するために使用されます。ストリーム・スクリプト ダイアログ・ボックスを使用して、現在のストリームとともに保存されているスクリプトを編集します。

「ストリームのプロパティー」ダイアログ・ボックスの「ストリーム・スクリプト」タブにアクセスするには

1. 「ツール」メニューから次の各項目を選択します。

   「ストリームのプロパティー」 > 「実行」
2. 「実行」タブをクリックして、現在のストリームのスクリプトの処理を行います。

ストリーム・スクリプトのダイアログ・ボックスの一番上にあるツールバー・アイコンを使用すると、次のような操作を実行できます。

- ウィンドウに既存のスタンドアロン スクリプトの内容をインポートする。
- スクリプトをテキスト・ファイルとして保存する。
- スクリプトを印刷する。
- デフォルト スクリプトを追加する。
- スクリプトを編集する (元に戻す、切り取り、コピー、貼り付けなど、標準的な編集機能を使用)。
- 現在のスクリプト全体を実行する。
- スクリプトから選択した行を実行する。
- 実行時にスクリプトを停止する (このアイコンは、スクリプトの実行時のみ使用可能になります)。
- スクリプトのシンタックスをチェックして、エラーが見つかれば、ダイアログ・ボックスの下部ペインにそれを表示する。

注**:** バージョン 16.0 以降、SPSS Modeler は Python スクリプト言語を使用します。16.0 よりも前のすべてのバージョンでは、SPSS Modeler に固有のスクリプト言語 (現在では「従来のスクリプト」と呼ばれる) を使用していました。処理しているスクリプトのタイプに応じて、「実行」タブで「デフォルト **(**オプション スクリプト**)**」実行モードを選択してから、「**Python**」または「従来のもの」のいずれかを選択します。

ストリームの実行時にスクリプトを実行するかどうかを指定できます。ストリームの実行時に常に、スクリプトに指定された実行順序でこのスクリプトを実行するには、「このスクリプトを実行」を選択します。この設定により、ストリーム・レベルでの自動化を実現でき、素早いモデル構築が可能になります。ただし、デフォルトでは、ストリーム実行時にこのスクリプトは無視されます。「このスクリプトを無視」 オプションを選択した場合でも、常にこのダイアログ・ボックスで直接スクリプトを実行できます。

スクリプト・エディターには、スクリプト・オーサリングを支援する以下の機能が用意されています。

- シンタックスの強調表示。キーワード、リテラル値 (文字列や数値など)、コメントが強調表示されます。
- 行番号付け。
- ブロックの一致。カーソルがプログラム・ブロックの開始位置に置かれると、対応する終了ブロックも強調表示されます。
- 自動入力の候補表示。

シンタックスの強調表示で使用される色とテキストのスタイルは、IBM SPSS Modeler の表示設定を使用してカスタマイズすることができます。この表示設定にアクセスするには、「ツール」 > 「オプション」 > 「ユーザー オプション」を選択して、「シンタックス」タブを選択します。

候補として表示されるシンタックス入力のリストにアクセスするには、コンテキスト・メニューから「自動候補提示機能」を選択するか、Ctrl + スペースを押します。カーソル・キーを使用してリストを上下に移動し、Enter キーを押して、選択したテキストを挿入します。既存のテキストを変更せずに自動候補提示機能モードを終了するには、Esc キーを押します。

「デバッグ」タブには、デバッグ・メッセージが表示されます。このタブを使用すると、スクリプトの実行中にスクリプトの状態を評価することができます。「デバッグ」タブは、読み取り専用テキスト領域と 1 行の入力テキスト・フィールドから構成されています。テキスト領域には、スクリプトによって (例えばエラー・メッセージ・テキストを通じて) 標準出力または標準エラーのいずれかに送信されたテキストが表示されます。入力テキスト・フィールドは、ユーザーから入力を受け取ります。次に、この入力は、ダイアログ内で最後に実行されたスクリプトのコンテキスト (スクリプト・コンテキスト) 内で評価されます。テキスト領域には、ユーザーがコマンドのトレースを確認できるように、コマンドと結果の出力が表示されます。テキスト入力フィールドには、コマンド・プロンプト (従来のスクリプトの -->) が常に表示されます。

以下の場合、新しいスクリプト・コンテキストが作成されます。

- スクリプトを実行するには、「このスクリプトを実行」または「選択した行のみ実行」を使用します。
- スクリプト言語を変更した場合。

新しいスクリプト・コンテキストが作成されると、テキスト領域がクリアされます。

注: スクリプト ペインの外部でストリームを実行しても、スクリプト ペインのスクリプト・コンテキストは変更されません。この実行の一部として作成された変数の値は、スクリプト・ダイアログ・ボックス内には表示されません。

ストリーム・スクリプトの例 :ニューラル・ネットワークの学習

ストリームは実行時に、ニューラル・ネットワーク・モデルの学習に使用できます。通常、モデルをテストするには、モデル作成ノードを実行してモデルをストリームに追加し、適切な接続を確立して、精度分析ノードを実行します。

IBM SPSS Modeler スクリプトを使用すると、モデル・ナゲット作成後のテスト プロセスを自動化できます。例えば、デモ・ストリーム druglearn.str (IBM SPSS Modeler インストールの下の /Demos/streams/ フォルダー内) をテストする次のストリーム・スクリプトは、「ストリーム・プロパティー」ダイアログ (「ツール」>「ストリームのプロパティー」>「スクリプト」) で実行できます。

```
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([])
```

このスクリプト例の各行について、次に説明します。

- 1 行目では、現在のストリームを指し示す変数を定義します。

- 2 行目では、ニューラル ネットワーク ビルダー ノードを検索します。
- 3 行目では、実行結果を格納できるリストを作成します。
- 4 行目では、ニューラル ネットワーク モデル ナゲットが作成されます。これは、3 行目で定義したリストに格納されます。
- 5 行目では、モデル ナゲットのモデル適用ノードが作成され、ストリーム領域に配置されます。
- 6 行目では、Drug という名前の分析ノードが作成されます。
- 7 行目では、データ型ノードを検索します。
- 8 行目では、5 行目で作成したモデル適用ノードを、データ型ノードと分析ノードとの間で接続します。
- 最後に、分析ノードが実行されて、分析レポートが生成されます。

空の領域から、ストリームを初めから作成して実行するスクリプトを使用することも可能です。スクリプト言語一般については、スクリプト言語の概要を参照してください。

Jython コードのサイズ制限

Jython は各スクリプトを Java バイトコードにコンパイルし、その Java バイトコードは Java 仮想マシン (JVM) によって実行されます。ただし、Java では単一のバイトコード・ファイルのサイズに制限があります。そのため、Jython がバイトコードをロードしようとした際に、JVM が異常終了することがあります。IBM SPSS Modeler は、その発生を防ぐことができません。

Jython スクリプトの記述には、適正なコーディング手法 (変数や関数を使用して共通の中間値を計算することにより重複コードを最小にするなど) を使用するようにしてください。必要に応じて、コードを複数のソース・ファイルに分割するか、モジュールを使用してコードを定義して、これらが別々のバイトコード・ファイルにコンパイルされるようにしなければならない場合があります。

スタンドアロン スクリプト

「スタンドアロン スクリプト」ダイアログ・ボックスでは、テキスト・ファイルとして保存されるスクリプトを作成したり編集したりします。このダイアログ・ボックスには、ファイル名が表示されます。スクリプトのロード、保存、インポート、および実行の機能が備わっています。

スタンドアロン スクリプトのダイアログ・ボックスにアクセスするには

メイン・メニューから次の各項目を選択します。

「ツール」 > 「スタンドアロン スクリプト」

スタンドアロン スクリプトでは、ストリーム・スクリプトと同じツールバーやスクリプト・シンタックス検査オプションを使用することができます。 詳しくは、トピック 2 ページの『ストリーム・スクリプト』を参照してください。

スタンドアロン スクリプトの例 :モデルの保存とロード

スタンドアロン スクリプトは、ストリームを操作するときに役立ちます。2 種類のストリームがある場合を想定します。1 つはモデルを作成するストリームであり、もう 1 つはグラフを使用して最初のストリームと既存のデータ・フィールドから生成されたルール・セットを探索するストリームです。この場合のスタンドアロン スクリプトは次のようになります。

```
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/18.1/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([])
```

注: スクリプト言語一般については、『スクリプト言語の概要』を参照してください。

スタンドアロン スクリプトの例 :変数選択モデルの生成

この例では、空の領域から変数選択モデルを生成するストリームを構築し、そのモデルを適用して、指定された対象に関連する最も重要な上位 15 のフィールドを表示するテーブルを作成します。

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

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

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

featureselectionnode = stream.createAt("featureselection", "Feature Selection", 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([])
```

このスクリプトで、データを読み込む入力ノードを作成し、response_01 フィールドの役割を Target に設定するデータ型ノードを使用し、その後フィールド選択ノードを作成して実行します。また、読みやすいレイアウトになるように、ストリーム領域で各ノードを接続し、配置します。その後、作成されるモデル・ナゲットがテーブル・ノードへ接続されます。テーブル・ノードでは、selection_mode プロパティーと top_n プロパティーに設定されたとおりに、もっとも重要な上位 15 フィールドが一覧表示されます。詳しくは、トピック 216 ページの『featureselectionnode プロパティー』を参照してください。

スーパーノード・スクリプト

IBM SPSS Modeler のスクリプト言語を使用して、スクリプトを作成し、任意のターミナル・スーパーノード内に保存できます。これらのスクリプトはテーミナル・スーパーノードにのみ使用でき、テンプレートストリームの作成時、およびスーパーノードの内容に特定の実行順序を指定する際に使用できます。スーパーノード・スクリプトを使用すると、ストリーム内で複数のスクリプトを実行することもできます。

例えば、複雑なストリームで実行の順序を指定する必要があり、スーパーノードには、散布図ノードで使用される新しいフィールドを作成する前に実行される必要のあるグローバル・ノードを含む、いくつかのノードがあるとします。この場合、まずグローバル・ノードを実行するスーパーノード・スクリプトを作成できます。このノードが計算する平均や標準偏差などの値は、散布図ノードを実行するときに使用します。

スーパーノード・スクリプト内では、ほかのスクリプトの場合と同様の方法で、ノード・プロパティーを指定できます。また、ストリーム・スクリプトから直接に、任意のスーパーノードまたはカプセル化されたノードのプロパティーを変更または定義することもできます。詳しくは、トピック 365 ページの『第 20 章 スーパーノードのプロパティー』を参照してください。この手法は、ソース スーパーノード、プロセス スーパーノード、およびターミナル・スーパーノードに適用できます。

注: 独自のスクリプトを実行することができるのはターミナル・スーパーノードの場合だけなので、「スーパーノード」ダイアログ・ボックスの「スクリプト」タブは、ターミナル・スーパーノードの場合にだけ利用可能です。

メイン キャンバスから「スーパーノード・スクリプト」ダイアログ・ボックスを開くには

ストリーム キャンバスでターミナル・スーパーノードを選択して、「スーパーノード」メニューから次の項目を選択します。

スーパーノード・スクリプト...

ズーム・インしたスーパーノード・キャンバスから「スーパーノード・スクリプト」ダイアログ・ボックスを開くには

スーパーノード領域を右クリックして表示されるコンテキスト・メニューから、次の項目を選択します。

スーパーノード・スクリプト...

スーパーノード・スクリプトの例

次のスーパーノード・スクリプトでは、スーパーノード内のターミナル・ノードが実行されるべき順序が宣言されます。この順序によって、まずグローバル・ノードが実行されて、別のノードを実行したときに、このノードによって算出される値が使用されるようになります。

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

スーパーノードのロックとロック解除

次の例は、スーパーノードのロックおよびロック解除の方法を説明しています。

```
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')
```

ストリームでのループと条件付き実行

バージョン 16.0 以降、SPSS Modeler では、スクリプト言語で直接指示を作成するのではなく、さまざまなダイアログ・ボックスで値を選択することによって、ストリーム内でいくつかの基本的なスクリプトを作成することができます。この方法で作成できるスクリプトの 2 つの主なタイプは、単純ループと、条件が満たされた場合にノードを実行する方法です。

ストリーム内でループ規則と条件付き実行規則の両方を組み合わせることができます。例えば、世界中の製造業者の自動車の販売に関連したデータがあるとします。ストリーム内のデータを処理するループを設定して、製造業者の国別に詳細を識別し、モデル別の販売数、製造業者別およびエンジン・サイズ別の排気ガスレベルなどの詳細を示すさまざまなグラフにデータを出力することができます。ヨーロッパの情報のみに関心がある場合は、アメリカとアジアの製造業者用のグラフが作成されないようにする条件をループに追加することもできます。

注: ループと条件付き実行は、いずれもバックグラウンドのスクリプトに基づいているため、これらはストリームの実行時にストリーム全体にのみ適用されます。

- ループ ループを使用して、反復タスクを自動化できます。例えば、指定の数のノードをストリームに追加し、追加するたびに 1 つのノード・パラメーターを変更することができます。あるいは、以下の例のように、ストリームまたは枝を指定の数だけ繰り返し実行することを制御できます。
  - ストリームを指定の回数実行し、実行するたびにソースを変更する。
  - ストリームを指定の回数実行し、実行するたびに変数の値を変更する。
  - ストリームを指定の回数実行し、実行するたびに 1 つの追加フィールドを入力する。
  - モデルを指定の回数構築し、毎回モデル設定を変更する。
- 条件付き実行 これを使用すると、事前定義した条件に基づいて、どのようにターミナル・ノードを実行するかを制御できます。例えば、以下のようにします。
  - 指定の値が true か false かに基づいて、ノードを実行するかどうかを制御する。
  - ノードのループを並行して実行するのか、順次に実行するのかを定義する。

ループと条件付き実行は両方とも、「ストリームのプロパティー」ダイアログ・ボックス内の「実行」タブで設定します。条件またはループ要件に使用されているノードは、追加の記号が付加された状態でストリーム・キャンバスに表示され、ループおよび条件付き実行に関与していることが示されます。

「実行」タブには、以下の 3 つのいずれかの方法でアクセスできます。

- メイン・ダイアログ・ボックスの上部にあるメニューを使用する。
  1. 「ツール」メニューから次の各項目を選択します。

     「ストリームのプロパティー」 > 「実行」
  2. 「実行」タブをクリックして、現在のストリーム用のスクリプトを処理します。
- ストリーム内から。
  1. ノードを右クリックして「ループ/条件付き実行」を選択する。
  2. 該当するサブメニューのオプションを選択します。
- メイン・ダイアログ・ボックスの上部にあるグラフィック・ツールバーで、ストリーム・プロパティーのアイコンをクリックする。

ループまたは条件付き実行の詳細を初めて設定する場合は、「実行」タブで「ループ/条件付き実行」実行モードを選択してから、「条件」または「ループ」サブタブを選択します。

ストリームでのループ

ループを使用すると、ストリーム内の反復タスクを自動化できます。例えば、以下のようにできます。

- ストリームを指定の回数実行し、実行するたびにソースを変更する。
- ストリームを指定の回数実行し、実行するたびに変数の値を変更する。
- ストリームを指定の回数実行し、実行するたびに 1 つの追加フィールドを入力する。
- モデルを指定の回数構築し、毎回モデル設定を変更する。

満たすべき条件は、ストリームの「実行」タブの「ループ」サブタブで設定します。サブタブを表示するには、「ループ/条件付き実行」実行モードを選択します。

定義するループ要件は、「ループ/条件付き実行」実行モードが設定されている場合は、ストリーム実行時に有効になります。オプションで、ループ要件のスクリプト・コードを生成して、スクリプト・エディターに貼り付けることができます。これを行うには、「ループ」サブタブの右下隅にある「貼り付け...」をクリックします。メインの「実行」タブの表示が「デフォルト (オプション スクリプト)」実行モードを表示するように変わり、スクリプトがタブの上部に表示されます。つまり、スクリプト・エディターで詳細にカスタマイズ可能なスクリプトを生成する前に、ダイアログ・ボックスのさまざまなループ・オプションを使用してループ構造を定義できます。「貼り付け...」をクリックすると、生成されたスクリプトには、定義した条件付き実行要件も表示されることに注意してください。

重要: SPSS Modeler ストリームで設定したループ変数は、このストリームを IBM SPSS Collaboration and Deployment Services ジョブで実行する場合、上書きされる可能性があります。 IBM SPSS Collaboration and Deployment Services ジョブ エディタのエントリは、SPSS Modeler のエントリによって上書きされるからです。例えば、ループごとに異なる出力ファイルを作成するようにストリーム内のループ変数を設定した場合、これらのファイルは SPSS Modeler 内で正しく命名されますが、IBM SPSS Collaboration and Deployment Services Deployment Manager の「結果」タブに入力された固定エントリによって上書きされます。

ループをセットアップするには

1. ストリームで実行するメインのループ構造を定義する反復キーを作成します。詳しくは、反復キーの作成を参照してください。
2. 必要に応じて、1 つ以上の反復変数を定義します。詳しくは、反復変数の作成を参照してください。
3. 作成した反復および変数が、サブタブの本文に表示されます。デフォルトで、反復は表示されている順に実行されます。反復をリスト内で上または下に移動するには、反復をクリックして選択し、サブタブの右側の列にある上矢印または下矢印を使用して順序を変更します。

ストリームでのループのための反復キーの作成

反復キーを使用して、ストリームで実行するメインのループ構造を定義します。例えば、自動車販売を分析している場合は、ストリーム・パラメーター製造国 を作成し、これを反復キーとして使用できます。ストリームを実行すると、このキーは、反復のたびにデータ内でそれぞれ異なる国の値に設定されます。「反復キーの定義」ダイアログ・ボックスを使用して、キーを設定します。

このダイアログ・ボックスを開くには、「ループ」サブタブの左下隅にある「反復キー」ボタンを選択するか、ストリーム内の任意のノードを右クリックして「ループ/条件付き実行」 > 「反復キーの定義 **(**フィールド**)**」または「ループ/条件付き実行」 > 「反復キーの定義 **(**値**)**」を選択します。ストリームからダイアログ・ボックスを開くと、ノードの名前などの一部のフィールドが自動的に入力されます。

反復キーを設定するには、以下のフィールドに入力します。

反復対象。次のいずれかのオプションを選択できます。

- ストリーム パラメーター - フィールド。このオプションは、既存のストリーム・パラメーターの値を、指定した各フィールドに順に設定するループを作成する場合に使用します。
- ストリーム パラメーター - 値。このオプションは、既存のストリーム・パラメーターの値を、指定した各値に順に設定するループを作成する場合に使用します。
- ノード プロパティー - フィールド。このオプションは、ノード・プロパティーの値を、指定した各フィールドに順に設定するループを作成する場合に使用します。
- ノード プロパティー - 値。このオプションは、ノード・プロパティーの値を、指定した各値に順に設定するループを作成する場合に使用します。

設定内容。ループが実行されるたびに値が設定される項目を選択します。次のいずれかのオプションを選択できます。

- パラメーター。「ストリーム パラメーター - フィールド」または「ストリーム パラメーター - 値」を選択した場合にのみ使用可能です。使用可能なリストから、必要なパラメーターを選択します。
- ノード。「ノード プロパティー - フィールド」または「ノード プロパティー - 値」を選択した場合にのみ使用可能です。ループをセットアップするノードを選択します。参照ボタンをクリックして「ノード選択」ダイアログを開き、目的のノードを選択します。リストされているノードが多すぎる場合は、ソース・ノード、プロセス・ノード、グラフ・ノード、モデリング・ノード、出力ノード、エクスポート・ノード、またはモデル・ノードの適用のいずれかのカテゴリー別にノードを表示するように、表示をフィルタリングできます。
- プロパティー: 「ノード プロパティー - フィールド」または「ノード プロパティー - 値」を選択した場合にのみ使用可能です。使用可能なリストからノードのプロパティーを選択します。

使用するフィールド。「ストリーム パラメーター - フィールド」または「ノード プロパティー - フィールド」を選択した場合にのみ使用可能です。反復値を提供するために使用するノード内のフィールドを選択します。次のいずれかのオプションを選択できます。

- ノード。「ストリーム パラメーター - フィールド」を選択した場合にのみ使用可能です。ループを設定する詳細を含むノードを選択します。参照ボタンをクリックして「ノード選択」ダイアログを開き、目的のノードを選択します。リストされているノードが多すぎる場合は、ソース・ノード、プロセス・ノード、グラフ・ノード、モデリング・ノード、出力ノード、エクスポート・ノード、またはモデル・ノードの適用のいずれかのカテゴリー別にノードを表示するように、表示をフィルタリングできます。
- フィールド リスト。右側の列にあるリスト・ボタンをクリックして、「フィールドの選択」ダイアログ・ボックスを表示します。このダイアログ・ボックスで、反復データを提供するノード内のフィールドを選択します。詳しくは、 11 ページの『反復のためのフィールドの選択』を参照してください。

使用する値。「ストリーム パラメーター - 値」または「ノード プロパティー - 値」を選択した場合にのみ使用可能です。選択したフィールド内で、反復値として使用する値 (複数可) を選択します。次のいずれかのオプションを選択できます。

- ノード。「ストリーム パラメーター - 値」を選択した場合にのみ使用可能です。ループを設定する詳細を含むノードを選択します。参照ボタンをクリックして「ノード選択」ダイアログを開き、目的のノードを選択します。リストされているノードが多すぎる場合は、ソース・ノード、プロセス・ノード、グラフ・ノード、モデリング・ノード、出力ノード、エクスポート・ノード、またはモデル・ノードの適用のいずれかのカテゴリー別にノードを表示するように、表示をフィルタリングできます。
- フィールド リスト。反復データを提供するためのノード内のフィールドを選択します。
- 値リスト。右側の列にあるリスト・ボタンをクリックして、「値の選択」ダイアログ・ボックスを表示します。このダイアログ・ボックスで、反復データを提供するフィールド内の値を選択します。

ストリームでのループのための反復変数の作成

反復変数を使用して、ループが実行されるたびに、ストリーム内の選択したノードのストリーム・パラメーターまたはプロパティーの値を変更できます。例えば、ストリーム・ループが自動車販売データを分析していて、製造国 を反復キーとして使用している場合に、モデル別の販売を示すグラフ出力と、排気ガス情報を示すグラフ出力があるとします。このような場合に、結果グラフごとに新しいタイトル (スウェーデンの自動車排気ガス や日本のモデル別自動車販売 など) を作成する反復変数を作成できます。「反復変数の定義」ダイアログ・ボックスを使用して、必要な変数を設定します。

このダイアログ ボックスを開くには、「ループ」サブタブの左下隅にある「変数の追加」ボタンを選択するか、ストリーム内の任意のノードを右クリックして「ループ/条件付き実行」 > 「反復変数の定義」を選択します。

反復変数を設定するには、以下のフィールドに入力します。

変更。修正する属性の種類を選択します。「ストリーム パラメーター」または「ノード プロパティー」を選択します。

- 「ストリーム パラメーター」を選択した場合、必要なパラメーターを選択してから、以下のいずれかのオプションを使用して (ストリームで使用可能な場合)、ループを反復するたびにそのパラメーターに設定する値を定義します。
  - グローバル変数。ストリーム・パラメーターを設定するグローバル変数を選択します。
  - テーブル出力セル。ストリーム・パラメーターをテーブル出力セルの値に設定するには、リストからテーブルを選択して、使用する「行」と「列」を入力します。
  - 手動で入力。このオプションは、このパラメーターが反復のたびに取る値を手動で入力する場合に選択します。「ループ」サブタブに戻ると、必要なテキストを入力する新しい列が作成されます。

- 「ノード プロパティー」を選択した場合は、必要なノードといずれかのプロパティーを選択してから、そのプロパティーに使用する値を設定します。以下のオプションの 1 つを使用して、新しいプロパティー値を設定します。
  - 単独。プロパティー値は、反復キー値を使用します。詳しくは、 9 ページの『ストリームでのループのための反復キーの作成』を参照してください。
  - 語幹の接頭辞として。「語幹」フィールドに入力する内容の接頭辞として反復キー値を使用します。
  - 語幹の接尾辞として。「語幹」フィールドに入力する内容の接尾辞として反復キー値を使用します。

  接頭辞または接尾辞のオプションを選択した場合は、「語幹」フィールドに追加テキストを追加するように求められます。例えば、反復キー値が製造国 で、「語幹の接頭辞として」を選択した場合は、このフィールドに - モデル別の販売 と入力できます。

反復のためのフィールドの選択

反復を作成する場合は、「フィールドの選択」ダイアログ・ボックスを使用して、1 つ以上のフィールドを選択できます。

ソート基準: 以下のいずれかのオプションを選択することにより、使用可能なフィールドを表示用にソートすることができます。

- ファイル順: データ ストリームから現在のノードに渡された順に各フィールドを表示します。
- 名前: 各フィールドをアルファベット順にソートして表示します。
- タイプ: 各フィールドを測定の尺度順にソートして表示します。特定の尺度のフィールドを選択する場合に役立ちます。

リストからフィールドを 1 回に 1 つずつ選択するか、または Shift キーまたは Ctrl キーを押しながら複数のフィールドを選択します。また、リストの下のボタンを使用して、尺度に基づいて複数のフィールドを選択したり、テーブル中のすべてのフィールドを選択または選択解除することができます。

選択可能なフィールドは、使用しているストリーム・パラメーターまたはノード・プロパティーに適切なフィールドのみが表示されるようにフィルタリングされていることに注意してください。例えば、ストレージ・タイプが文字列のストリーム・パラメーターを使用している場合は、ストレージ・タイプが文字列のフィールドのみが表示されます。

ストリームでの条件付き実行

条件付き実行では、定義するストリーム内容の一致条件に基づいて、ターミナル・ノードを実行する方法を制御できます。例えば、以下のようにできます。

- 指定の値が true か false かに基づいて、ノードを実行するかどうかを制御する。
- ノードのループを並行して実行するのか、順次に実行するのかを定義する。

満たすべき条件は、ストリームの「実行」タブの「条件」サブタブで設定します。サブタブを表示するには、「ループ/条件付き実行」実行モードを選択します。

定義する条件付き実行要件は、「ループ/条件付き実行」実行モードが設定されている場合は、ストリーム実行時に有効になります。オプションで、条件付き実行要件のスクリプト・コードを生成して、スクリプト・エディターに貼り付けることができます。これを行うには、「条件」サブタブの右下隅にある「貼り付け...」をクリックします。メインの「実行」タブの表示が「デフォルト (オプション スクリプト)」実行モードを表示するように変わり、スクリプトがタブの上部に表示されます。つまり、スクリプト・エディターで詳細にカスタマイズ可能なスクリプトを生成する前に、ダイアログ・ボックスのさまざまなループ・オプ

ションを使用して条件を定義できます。「貼り付け...」をクリックすると、生成されたスクリプトには、定義したループ要件も表示されることに注意してください。

条件をセットアップするには以下を行います。

1. 「条件」サブタブの右側の列で「新規条件の追加」ボタン  をクリックして、「条件実行式の追加」ダイアログ ボックスを開きます。このダイアログで、ノードを実行するために満たす必要がある条件を指定します。
2. 「条件実行式の追加」ダイアログ ボックスで、以下のオプションを指定します。
   a. ノード。条件付き実行を設定するノードを選択します。参照ボタンをクリックして「ノード選択」ダイアログを開き、目的のノードを選択します。リストされているノードが多すぎる場合は、エクスポート・ノード、グラフ・ノード、モデリング・ノード、または出力ノードのいずれかのカテゴリー別にノードを表示するように、表示をフィルタリングできます。
   b. 条件の基準。ノードを実行するために満たす必要がある条件を指定します。「ストリーム　パラメーター」、「グローバル変数」、「テーブル出力セル」、または「常に **True**」の 4 つのオプションのいずれか 1 つを選択できます。ダイアログ・ボックスの下半分に入力する詳細は、選択する条件によって異なります。
      - ストリーム　パラメーター。使用可能なリストからパラメーターを選択してから、そのパラメーターの「演算子」を選択します。例えば、演算子は「より大きい」、「等しい」、「より小さい」、「間」などです。次に、演算子に応じて「値」か、最小値および最大値を入力します。
      - グローバル変数。使用可能なリストから変数を選択します。例えば、「平均」、「合計」、「最小値」、「最大値」、または「標準偏差」がリストに含まれている可能性があります。次に、「演算子」と必要な値を選択します。
      - テーブル出力セル。使用可能なリストからテーブル・ノードを選択して、テーブルの「行」と「列」を選択します。次に、「演算子」と必要な値を選択します。
      - 常に **True**。ノードを常に実行する必要がある場合は、このオプションを選択します。このオプションを選択する場合は、さらに選択するパラメーターはありません。
3. 必要なすべての条件を設定するまで、ステップ 1 と 2 を必要なだけ繰り返します。選択したノードと、ノードが実行される前に満たすべき条件が、サブタブの本体部分の「実行ノード」列と、「値の設定条件 **(**真の場合に値を設定**)**」列に表示されます。
4. デフォルトで、ノードと条件は表示されている順に実行されます。ノードと条件をリスト内で上または下に移動するには、ノードと条件をクリックして選択し、サブタブの右側の列にある上矢印または下矢印を使用して順序を変更します。

さらに、「条件」サブタブの下部にある以下のオプションを設定できます。

- すべてを順番に評価。このオプションは、各条件をサブタブに表示されている順序で評価する場合に選択します。条件が「True」のすべてのノードは、すべての条件が評価されてから 1 回だけ実行されます。
- 一度に **1** つずつ実行。「すべてを順番に評価」が選択されている場合にのみ使用できます。このオプションを選択すると、条件が「True」と評価される場合、その条件に関連付けられているノードは、次の条件が評価される前に実行されます。
- 最初のヒットまで評価。このオプションを選択すると、指定した条件から「True」の評価が返される最初のノードのみが実行されます。

スクリプトの実行と中断

その他多くの方法でスクリプトを実行できます。例えば、ストリーム・スクリプトまたはスタンドアロンのスクリプトのダイアログで、「このスクリプトを実行」ボタンをクリックすると、完全なスクリプトを実行します。

図 *1.* 「このスクリプトを実行」ボタン

「選択した行」ボタンをクリックすると、スクリプト内で選択した 1 行または隣接する行のブロックを実行します。

図 *2.* 「選択した行を実行」ボタン

スクリプトの実行は、次のいずれかの方法で行います。

- ストリーム・スクリプトまたはスタンドアロン スクリプトのダイアログ・ボックスの「このスクリプトを実行」または「選択した行を実行」をクリックします。
- デフォルトの実行方法として 「このスクリプトを実行」が設定されているストリームを実行する。
- 起動時にインタラクティブ・モードで `-execute` フラグを使用します。詳しくは、トピック 65 ページの『コマンド・ライン引数の使用』を参照してください。

注: 「スーパーノード」ダイアログ・ボックスで 「このスクリプトを実行」を選択しているかぎり、スーパーノード・スクリプトは、スーパーノードの実行時に実行されます。

スクリプト実行の中断

「ストリーム・スクリプト」ダイアログ・ボックスのツールバーにある赤い中止ボタンは、スクリプト実行時に有効になります。このボタンを使用すると、スクリプトおよび現在のストリームの実行を中止することができます。

検索と置換

「検索/置換」ダイアログ・ボックスは、スクリプト・エディター、CLEM 式ビルダーなど、スクリプトまたは式のテキストを編集する場合、またはレポート・ノードでテンプレートを定義する場合に使用できます。これらの領域のいずれかでテキストを編集する場合、`Ctrl + F キー` を押してダイアログ・ボックスにアクセスし、カーソルがテキスト領域にフォーカスしていることを確認します。「フィルター」ノードを使用している場合、例えば、「設定」タブのテキスト領域から、または CLEM 式ビルダーのテキスト・フィールドからダイアログ・ボックスにアクセスできます。

1. テキスト領域内にカーソルを置いて、Ctrl + F キーを押して「検索/置換」ダイアログ・ボックスにアクセスします。
2. 検索するテキストを入力するか、最近検索した項目のドロップダウン・リストから選択します。
3. 置換テキストがある場合は、入力します。
4. 「次を検索」 をクリックして、検索を開始します。

5. 「置換」 をクリックして現在の選択内容を置換するか、「すべてを置換」 をクリックしてすべてまたは選択したインスタンスを更新します。
6. 各操作が終了すると、ダイアログ・ボックスが閉じます。テキスト領域で F3 を押すと最後の検索操作が繰り返され、または Ctrl + F キーを押すとダイアログに再度アクセスします。

検索オプション

大文字と小文字を区別 **:** 検索操作で、例えば *myvar* が *myVar* と位置するかどうかなど、大文字と小文字を区別するかどうかを指定します。この設定に関係なく、置換テキストは常に入力したとおりに挿入されます。

語全体のみ **:** 検索操作が語内に埋め込まれたテキストに一致するかどうかを指定します。このオプションを選択すると、*spider* に関する検索は、*spiderman* または *spider-man* に一致しません。

正規表現 **:** 正規表現のシンタックスを使用するかどうかを指定します (次項参照)。このオプションを選択すると、「語全体のみ」 オプションは無効化され、その値は無視されます。

選択されたテキストのみ**:** 「すべてを置換」オプションを使用する場合、検索の範囲を制御します。

正規表現シンタックス

正規表現を使用すると、タブまたは改行文字などの特殊文字、*a* から *d* までなど文字のクラスまたは範囲、行の開始または終了などの境界について検索することができます。次の種類の表現がサポートされています。

表 *1.* 文字の一致 :

Characters	一致
x	文字 x
\\	円記号
¥0n	8 進法の値を持つ文字 0n (0 <= n <= 7)
¥0nn	8 進法の値を持つ文字 0nn (0 <= n <= 7)
¥0mnn	8 進法の値を持つ文字 0mnn (0 <= m <= 3, 0 <= n <= 7)
¥xhh	16 進法の値を持つ文字 0xhh
¥uhhhh	16 進法の値を持つ文字 0xhhhh
¥t	タブ文字 ('¥u0009')
¥n	改行文字 ('¥u000A')
¥r	復帰文字 ('¥u000D')
¥f	改ページ文字 ('¥u000C')
¥a	アラート (ベル) 文字 ('¥u0007')
¥e	エスケープ文字 ('¥u001B')
¥cx	xに対応する制御文字

表 *2.* 文字クラスの一致 :

文字クラス	一致
「abc]	a、b、または c (単純クラス)
「^abc]	a、b、または c 以外の文字 (減法)
「a-zA-Z]	a から z または A から Z の各文字 (範囲)

表 2. 文字クラスの一致 (続き) :

文字クラス	一致
「a-d[m-p]]	a から d、または m から p (和集合)。または、「a-dm-p」と指定することもできます
「a-z&&[def]]	a から z、および d、e、または f (交差)
「a-z&&[^bc]]	a から z のうち、b と c を除いたもの (差集合)。または、「ad-z」と指定することもできます
「a-z&&[^m-p]]	a から z のうち、m から p までを除いたもの (差集合)。または、「a-lq-z」と指定することもできます

表 3. 事前設定された文字クラス :

事前設定された文字クラス	一致
.	任意の文字 (行末に一致する場合または一致しない場合があります)
¥d	任意の数字: [0-9]
¥D	数字以外: [^0-9]
¥s	空白文字: [¥t¥n¥x0B¥f¥r]
¥S	空白文字以外: [^¥s]
¥w	ワード文字: [a-zA-Z_0-9]
¥W	非ワード文字: [^¥w]

表 4. 境界の一致 :

境界の一致	一致
^	行頭
$	行末
¥b	語の境界
¥B	語以外の境界
¥A	入力の開始
¥Z	最後の行末以外の入力の終了
¥z	入力の終了

第 2 章 スクリプト言語

スクリプト言語の概要

IBM SPSS Modeler のスクリプト機能を使用すると、SPSS Modeler ユーザー・インターフェースで動作し、出力オブジェクトを操作し、コマンド・シンタックスを実行するスクリプトを作成できます。SPSS Modeler 内から直接スクリプトを実行できます。

IBM SPSS Modeler のスクリプトは、スクリプト言語 Python で作成されています。IBM SPSS Modeler で使用される Python の Java ベースの実装を Jython と呼びます。このスクリプト言語は、以下の機能で構成されています。

- ノード、ストリーム、プロジェクト、出力、およびその他の IBM SPSS Modeler オブジェクトを参照する形式
- 上記オブジェクトを操作するのに使用されるスクリプト ステートメントまたはコマンドのセット
- 変数、パラメーター、およびその他のオブジェクトに値を設定するためのスクリプト式の言語
- コメント、行の継続、およびリテラル テキストのブロックのサポート

以下のセクションでは、Python スクリプト言語、Python の Jython 実装、および IBM SPSS Modeler 内でスクリプトを使い始めるための基本シンタックスについて説明します。特定のプロパティーとコマンドについての情報は、以後のセクションにあります。

Python と Jython

Jython は、Python スクリプト言語の実装の 1 つであり、Java 言語で記述され、Java プラットフォームと統合されています。Python は強力なオブジェクト指向スクリプト言語です。Jython は、成熟したスクリプト言語の生産性向上機能を備え、Python とは異なり、Java 仮想マシン (JVM) をサポートするすべての環境で動作します。そのため、プログラムの作成時に JVM の Java ライブラリーを使用することができます。Jython を使用すると、この違いを利用できると同時に、Python 言語の構文とほとんどの機能を使用できます。

スクリプト言語であるため、Python (およびその Jython 実装) は習得が容易で効率的にコーディングできるほか、動作するプログラムの作成に最小限の構造しか必要としません。コードは対話式で (一度に 1 行) 入力することができます。Python はインタープリター式のスクリプト言語であり、Java にあるプリコンパイルの段階がありません。Python プログラムは単なるテキスト・ファイルであり、(構文エラーがないかどうか構文解析された後に) 入力として解釈されます。単純な式 (定義済みの値など) のほか、複雑な操作 (関数定義など) もただちに実行され、使用可能になります。コードに対して行った変更を迅速にテストすることができます。しかし、スクリプトの解釈には不利な点もあります。例えば、未定義の変数を使用してもコンパイラー・エラーにならないため、その変数を使用するステートメントが実行される場合に限り、その実行のときに検出されます。この場合は、プログラムを編集して実行し、エラーをデバッグすることができます。

Python では、データやコードも含め、あらゆるものをオブジェクトとして扱います。したがって、それらのオブジェクトを一連のコードで操作することができます。一部の型 (数値や文字列など) はオブジェクトではなく値と見なすと便利ですが、この扱いは Python でもサポートされています。サポートされているヌル値が 1 つあります。このヌル値には予約名 `None` が割り当てられています。

Python スクリプトおよび Jython スクリプトの概要やスクリプト例については、 http://www.ibm.com/developerworks/java/tutorials/j-jython1/j-jython1.htmlおよびhttp://www.ibm.com/developerworks/java/tutorials/j-jython2/j-jython2.html を参照してください。

Python スクリプト

Python スクリプト言語の以下のガイドでは、IBM SPSS Modeler でスクリプトを作成する場合に使用される可能性が高いコンポーネントの概要と、概念やプログラミングの基礎について取り上げます。これにより、IBM SPSS Modeler 内で使用する Python スクリプトの開発を始めるのに十分な知識を得ることができます。

操作

代入は等号 (=) を使用して行います。例えば、値「3」を「x」という変数に代入するには、以下のステートメントを使用します。

```
x = 3
```

等号は、文字列型のデータを変数に代入する場合にも使用されます。例えば、値「a string value」を「y」という変数に代入するには、以下のステートメントを使用します。

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

次の表に、よく使用される比較演算子および数値演算子と、その説明を示します。

表 5. 一般的な比較演算子および数値演算子

演算	説明
x < y	x が y より小さいかどうか
x > y	x が y より大きいかどうか
x <= y	x が y 以下かどうか
x >= y	x が y 以上かどうか
x == y	x が y と等しいかどうか
x != y	x が y と等しくないかどうか
x <> y	x が y と等しくないかどうか
x + y	y を x に加算する
x - y	y を x から減算する
x * y	x に y を乗算する
x / y	x を y で除算する
x ** y	x を y 乗する

リスト

リストは、一連の要素です。リストには任意の数の要素を入れることができ、リストの要素には任意のタイプのオブジェクトを使用できます。リストは配列と考えることもできます。リスト内の要素の数は、要素を追加、削除、または置換する際に増加または減少します。

例

[]	空のリスト。
[1]	単一の要素 (整数) を含むリスト。

`["Mike", 10, "Don", 20]`	4 つの要素 (2 つの文字列要素と、2 つの整数要素) を含むリスト。
`[[],[7],[8,9]]`	リストを含むリスト。各サブリストは、空のリスト、または整数要素のリスト。
`x = 7; y = 2; z = 3;` `[1, x, y, x + y]`	整数のリスト。この例は、変数と式の使い方を示しています。

リストを変数に割り当てることができます。例えば、以下のようにします。

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

その後、このリストの特定の要素にアクセスできます。例えば、以下のようにします。

```
mylist[0]
```

これは以下のような出力になります。

```
one
```

大括弧 ([]) 内の数値は、インデックス と呼ばれ、リストの特定の要素を参照します。リストの各要素には、0 から始まるインデックスが付けられます。

1 つのリストから複数の要素の範囲を選択することもできます。これはスライス と呼ばれます。例えば、`x[1:3]` は、x の 2 番目の要素と 3 番目の要素を選択します。末尾のインデックスは、選択範囲の 1 つあとのインデックスです。

文字列

文字列 は、値として扱われる一連の不変の文字です。文字列は、新しい文字列になるすべての不変のシーケンス関数および演算子をサポートします。例えば、`"abcdef"[1:4]` は、`"bcd"` という出力になります。

Python では、文字は長さが 1 の文字列として表されます。

文字列リテラルは、単一引用符または三重引用符によって定義されます。単一引用符を使用して定義される文字列は行をまたぐことはできませんが、三重引用符を使用して定義される文字列は行をまたぐことができます。文字列は単一引用符 (') または二重引用符 (") で囲むことができます。引用符の内側には、エスケープされていない他の引用符、または円記号 (¥) が先行するエスケープされた引用符を入れることができます。

例

```
"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"
```

空白文字で区切られた複数の文字列は、Python パーサーによって自動的に連結されます。これにより、長い文字列を入力したり、単一文字列で異なる種類の引用符を混在させたりすることができます。

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

これにより、次のように出力されます。

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

文字列は、いくつかの有用なメソッドをサポートしています。次の表に、これらのメソッドの一部を示します。

表 *6.* 文字列メソッド

メソッド	使用法
s.capitalize()	s の頭文字を大文字にします。
s.count(ss {,start {,end}})	s[start:end] 内の ss の出現回数をカウントします。
s.startswith(str {, start {, end}}) s.endswith(str {, start {, end}})	s が str で始まっているかどうかをテストします。 s が str で終わっているかどうかをテストします。
s.expandtabs({size})	タブをスペース (デフォルトの size は 8) で置換します。
s.find(str {, start {, end}}) s.rfind(str {, start {, end}})	s の中で str の最初のインデックスを検索します。見つからない場合、結果は -1 になります。rfind は、右から左に検索します。
s.index(str {, start {, end}}) s.rindex(str {, start {, end}})	s の中で str の最初のインデックスを検索します。見つからない場合、ValueError が発生します。rindex は、右から左に検索します。
s.isalnum	文字列が英数字かどうかを確認するためのテスト。
s.isalpha	文字列が英字かどうかを確認するためのテスト。
s.isnum	文字列が数値かどうかを確認するためのテスト。
s.isupper	文字列がすべて大文字かどうかを確認するためのテスト。
s.islower	文字列がすべて小文字かどうかを確認するためのテスト。
s.isspace	文字列がすべて空白文字かどうかを確認するためのテスト。
s.istitle	文字列が、大文字の頭文字を持つ英数字文字列のシーケンスであるかどうかを確認するためのテスト。
s.lower() s.upper() s.swapcase() s.title()	すべて小文字に変換します。 すべて大文字に変換します。 大文字/小文字をすべて逆に変換します。 すべてタイトル・ケースに変換します。
s.join(seq)	seq 内の文字列を s を区切り文字として結合します。
s.splitlines({keep})	s を複数行に分割します。keep が true の場合、改行を保持します。
s.split({sep {, max}})	s を sep (デフォルトの sep は空白文字です) を使用して max 回まで「単語」に分割します。
s.ljust(width) s.rjust(width) s.center(width) s.zfill(width)	幅が width のフィールド内で文字列を左揃えします。 幅が width のフィールド内で文字列を右揃えします。 幅が width のフィールド内で文字列を中央揃えします。 0 で埋めます。
s.lstrip() s.rstrip() s.strip()	先頭の空白文字を削除します。 末尾の空白文字を削除します。 先頭と末尾の空白文字を削除します。
s.translate(str {,delc})	delc の文字を削除した後で、テーブルを使用して s を変換します。str は、長さが == 256 の文字列である必要があります。
s.replace(old, new {, max})	文字列 old をすべて、または max 個の出現箇所を文字列 new で置き換えます。

注釈

注釈は、ポンド (ハッシュ) 記号 (#) で始まるコメントです。ポンド記号に続く同じ行のすべてのテキストは、注釈の一部と見なされて無視されます。注釈は、任意の桁から開始できます。以下の例で、注釈の使用法を示します。

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

ステートメントの構文

Python のステートメントのシンタックスは非常に単純です。一般に、各ソース行は単一ステートメントです。 expression および assignment ステートメントを除いて、各ステートメントはキーワード名 (if や for など) で始まります。空白行または注釈行は、コード内の任意のステートメントの間のどこにでも挿入できます。 1 行に 2 つ以上のステートメントがある場合、各ステートメントをセミコロン (;) で区切る必要があります。

長いステートメントは、複数の行に続けることができます。この場合、次の行に続けるステートメントの末尾に円記号 (¥) を使用する必要があります。例えば、以下のようにします。

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

ある構造が括弧 (())、大括弧 ([])、または中括弧 ({}) で囲まれている場合は、円記号を挿入することなく、ステートメントをカンマの後ろで新しい行に続けることができます。例えば、以下のようにします。

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

識別子

識別子は、変数、関数、クラス、およびキーワードに名前を付けるために使用します。識別子の長さは任意ですが、先頭の文字は英字 (大文字または小文字) または下線 (_) でなければなりません。下線で始まる名前は、一般に内部名またはプライベート名のために予約されています。識別子の先頭文字の後ろに、英字、0 から 9 の数字、および下線文字をいくつでも自由に組み合わせて使用できます。

Jython には、変数、関数、またはクラスの名前に使用できない予約語がいくつかあります。これらの予約語は、以下のカテゴリーに分かれています。

- ステートメント接頭部: assert、break、class、continue、def、del、elif、else、except、exec、finally、for、from、global、if、import、pass、print、raise、return、try、および while
- パラメーター接頭部: as、import、および in
- 演算子: and、in、is、lambda、not、および or

不適切なキーワードを使用すると、通常 SyntaxError が発生します。

コードのブロック

コードのブロックは、単一ステートメントが期待される場所に使用されるステートメントのグループです。コードのブロックは、if、elif、else、for、while、try、except、def、および class のいずれのステートメントの後ろにも置くことができます。これらのステートメントの後ろにコロン (:) を使用して、コードのブロックを続けます。例えば、以下のようにします。

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

コード・ブロックを区切るためにインデントが使用されます (Java では中括弧が使用される)。 1 つのブロック内のすべての行を同じ位置にインデントする必要があります。これは、インデントの変更が、コード・ブロックの終了を示すためです。通常は、レベルごとに 4 つのスペースでインデントします。行のインデントには、タブではなくスペースを使用することが推奨されています。スペースとタブを混在させることはできません。モジュールの最外部のブロックの行は、1 桁目から開始する必要があります。そうでないと、SyntaxError が発生します。

1 つのコード・ブロックを構成する複数のステートメント (コロンに続ける) は、セミコロンで区切って 1 行にすることもできます。例えば、以下のようにします。

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

スクリプトへの引数の引き渡し

スクリプトに引数を渡すことは、変更せずにスクリプトを繰り返し使用できるため便利です。コマンド・ライン行で渡される引数は、リスト sys.argv 内の値として渡されます。渡される値の数は、コマンド len(sys.argv) を使用して取得できます。以下に例を示します。

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

この例では、import コマンドは、sys クラス全体をインポートして、このクラスに存在しているメソッド (argv など) を使用できるようにします。

この例のスクリプトは、以下の行を使用して起動できます。

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

結果は以下の出力になります。

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

例

print キーワードは、このキーワードの直後の引数を表示します。ステートメントの後ろにコンマを続けると、改行は出力に含まれません。以下に例を示します。

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

これは以下のような出力になります。

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

for ステートメントは、コードのブロックを反復するために使用します。以下に例を示します。

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

この例では、3 つの文字列がリスト mylist1 に割り当てられます。リストの各要素が 1 行に 1 つずつ出力されます。これは以下のような出力になります。

```
one
two
three
```

この例では、for ループが要素ごとのコード・ブロックを実装するたびに、イテレーター lv がリスト mylist1 の各要素の値を順にとります。イテレーターは、任意の長さの有効な ID にすることができます。

if ステートメントは、条件ステートメントです。条件を評価し、評価の結果に基づいて true または false を返します。以下に例を示します。

```
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
```

この例では、イテレーター lv の値が評価されます。 lv の値が two の場合は、lv が two ではない場合に返されるストリングとは異なるストリングが返されます。これにより、次のように出力されます。

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

数学メソッド

math モジュールから、有用な数学メソッドにアクセスできます。次の表に、これらのメソッドの一部を示します。特に指定のない限り、すべての値は浮動小数点として返されます。

表 7. 数学メソッド

メソッド	使用法
math.ceil(x)	x の天井値を浮動小数点として返します。これは、x 以上の最小の整数です。
math.copysign(x, y)	x を y の符号で返します。copysign(1, -0.0) は、-1 を返します。
math.fabs(x)	x の絶対値を返します。
math.factorial(x)	x 階乗を返します。x が負の場合、または整数でない場合、ValueError が発生します。
math.floor(x)	x の床値を浮動小数点として返します。これは、x 以下の最大の整数です。
math.frexp(x)	x の仮数 (m) と指数 (e) を (m, e) の組みとして返します。m は浮動小数点、e は整数で、x == m * 2**e となります。x がゼロの場合は (0.0, 0) を返し、それ以外の場合は 0.5 <= abs(m) < 1 を返します。
math.fsum(iterable)	iterable の中の値の正確な浮動小数点の和を返します。

表 7. 数学メソッド (続き)

メソッド	使用法
math.isinf(x)	浮動小数点 x が正または負の無限大かどうかをチェックします。
math.isnan(x)	浮動小数点 x が NaN (非数値) かどうかをチェックします。
math.ldexp(x, i)	x * (2**i) を返します。これは、本質的に関数 frexp の逆です。
math.modf(x)	x の小数部と整数部を返します。結果は両方とも x の符号を引き継ぎ、浮動小数点です。
math.trunc(x)	Integral に切り捨てられた Real 値 x を返します。
math.exp(x)	e**x を返します。
math.log(x[, base])	指定した値 base に対する x の対数を返します。 base を指定しない場合は、x の自然対数が返されます。
math.log1p(x)	1+x (base e) の自然対数を返します。
math.log10(x)	x の 10 を底とする対数を返します。
math.pow(x, y)	x を y 乗して返します。pow(1.0, x) および pow(x, 0.0) は、x がゼロまたは NaN であるとしても、常に 1 を返します。
math.sqrt(x)	x の平方根を返します。

数学関数に加えて、有用な三角関数メソッドもあります。次の表に、これらのメソッドを示します。

表 8. 三角関数メソッド

メソッド	使用法
math.acos(x)	x の逆余弦をラジアンで返します。
math.asin(x)	x の逆正弦をラジアンで返します。
math.atan(x)	x の逆正接をラジアンで返します。
math.atan2(y, x)	atan(y / x) をラジアンで返します。
math.cos(x)	x の余弦をラジアンで返します。
math.hypot(x, y)	ユークリッドノルム sqrt(x*x + y*y) を返します。これは原点から点 (x, y) へのベクトルの長さです。
math.sin(x)	x の正弦をラジアンで返します。
math.tan(x)	x の正接をラジアンで返します。
math.degrees(x)	角 x をラジアンから度に変換します。
math.radians(x)	角 x を度からラジアンに変換します。
math.acosh(x)	x の逆双曲線余弦を返します。
math.asinh(x)	x の逆双曲線正弦を返します。
math.atanh(x)	x の逆双曲線正接を返します。
math.cosh(x)	x の双曲線余弦を返します。
math.sinh(x)	x の双曲線余弦を返します。
math.tanh(x)	x の双曲線正接を返します。

2 つの数学定数もあります。math.pi の値は、数学定数 pi です。math.e の値は、数学定数 e です。

非 ASCII 文字の使用

非 ASCII 文字を使用するには、Python では、文字列を Unicode に明示的にエンコードまたはデコードする必要があります。IBM SPSS Modeler では、Python スクリプトは UTF-8 (非 ASCII 文字をサポートする標準 Unicode) でエンコードされていると想定されます。以下のスクリプトは、Python コンパイラーが SPSS Modeler によって UTF-8 に設定されているため、コンパイルされます。

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

しかし、結果ノードのラベルは正しくありません。

図 3. 非 *ASCII* 文字を含むノード・ラベル (正しく表示されていない)

ストリング・リテラル自体が Python によって ASCII 文字列に変換されているため、このラベルは正しくありません。

Python では、文字列リテラルの前に u 文字を追加することによって、Unicode 文字列リテラルを指定できます。

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

これにより、Unicode 文字列が作成され、ラベルが正しく表示されます。

図 4. 非 *ASCII* 文字を含むノード・ラベル (正しく表示されている)

Python と Unicode の使用は、本書の範囲を超えた大きなトピックです。このトピックを詳細に扱った書籍やオンライン情報源が数多くあります。

オブジェクト指向プログラミング

オブジェクト指向プログラミングは、対象問題のモデルをプログラム内で作成するという概念に基づいています。オブジェクト指向プログラミングにより、プログラミング・エラーが減り、コードの再使用が促進されます。 Python は、オブジェクト指向言語です。Python で定義されるオブジェクトには、以下の特徴があります。

- 同一**:** 各オブジェクトは個別であり、これはテスト可能でなければなりません。`is` テストと `is not` テストは、この目的のために存在しています。
- 状態。各オブジェクトは、状態を格納できる必要があります。フィールドやインスタンス変数などの属性は、この目的のために存在しています。
- 振る舞い。各オブジェクトは、状態を操作できる必要があります。メソッドは、この目的のために存在します。

Python には、オブジェクト指向プログラミングをサポートするための以下の特徴があります。

- クラス・ベースのオブジェクト作成。クラスは、オブジェクトを作成するためのテンプレートです。オブジェクトは、振る舞いが関連づけられているデータ構造です。
- ポリモアフィズムによる継承。Python は、単一継承と多重継承をサポートしています。 Python のすべてのインスタンス・メソッドは、ポリモアフィックであり、サブクラスによるオーバーライドが可能です。
- データ隠蔽によるカプセル化。Python では、属性を隠すことができます。隠すと、クラスの外側からは、そのクラスのメソッドによってのみ属性にアクセスできるようになります。クラスには、データを変更するためのメソッドを実装します。

クラスの定義

Python クラスの中では、変数とメソッドの両方を定義できます。Java と異なり、Python では、1 つのソース・ファイル (モジュール) で任意の数の公開クラスを定義できます。したがって、Python のモジュールは Java のパッケージに似ていると考えることができます。

Python では `class` ステートメントを使用してクラスを定義します。`class` ステートメントは、次の形式になっています。

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

or

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

クラスを定義するときには、任意の数の代入 ステートメントを記述することができます (記述しなくても構いません)。これにより、クラスのすべてのインスタンスで共有されるクラス属性が作成されます。また、任意の数の関数 定義を記述することもできます (記述しなくても構いません)。これらの関数定義により、メソッドが作成されます。スーパークラスのリストはオプションです。

クラス名はスコープの中 (モジュール、関数、またはクラスの中) で固有でなければなりません。複数の変数を定義して同じクラスを参照することができます。

クラス・インスタンスの作成

クラスは、クラス (共有) 属性の保持やクラス・インスタンスの作成に使用します。クラスのインスタンスを作成するには、そのクラスが関数であるかのように呼び出します。たとえば、次のクラスを考慮してください。

```
class MyClass:
    pass
```

クラスを完結させるためにはステートメントが必要ですがプログラムとしては動作が不要であるため、ここでは pass ステートメントを使用しています。

以下のステートメントは、クラス MyClass のインスタンスを作成します。

```
x = MyClass()
```

クラス・インスタンスへの属性の追加

Java と異なり、Python ではクライアントがクラスのインスタンスに属性を追加することができます。変更されるインスタンスは 1 つだけです。例えば、インスタンス x に複数の属性を追加するには、以下のようにしてそのインスタンスに新しい値を設定します。

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

クラス属性およびメソッドの定義

クラスにバインドされた変数はすべてクラス属性 です。クラス内で定義された関数はすべてメソッド です。メソッドは、クラスのインスタンス (慣習として self と呼びます) を第 1 引数として受け取ります。例えば、クラス属性およびメソッドを定義するには、以下のコードを入力します。

```
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
```

クラスの内側では、クラス属性に対するすべての参照をクラス名で修飾する必要があります (MyClass.attr1 など)。インスタンス属性に対する参照は、すべて self 変数で修飾する必要があります (self.text など)。クラスの外側では、クラス属性に対するすべての参照をクラス名で修飾するか (MyClass.attr1 など)、クラスのインスタンスで修飾する (x をクラスのインスタンスとすると x.attr1 などとする) 必要があります。クラスの外側では、インスタンス変数に対するすべての参照をクラスのインスタンスで修飾する必要があります (x.text など)。

非表示変数

プライベート 変数を作成することにより、データを隠蔽することができます。プライベート変数にアクセスできるのはそのクラス自体に限られます。__xxx または __xxx_yyy という形式で (2 個の下線を前に付けて) 名前を宣言すると、Python パーサーは、宣言された名前に自動的にクラス名を追加して隠蔽された変数を作成します。例を示します。

```
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
```

Java と異なり、Python では、インスタンス変数に対する参照はすべて `self` で修飾する必要があります。暗黙的な `this` の使用はありません。

継承

クラスを継承する機能は、オブジェクト指向プログラミングの根幹をなします。Python は、単一継承と多重継承の両方をサポートしています。単一継承 は、スーパークラスが 1 つしか存在できないことを意味します。多重継承 は、複数のスーパークラスが存在できることを意味します。

継承は、他のクラスのサブクラスを定義することで実装します。任意の数の Python クラスをスーパークラスにすることができます。Python の Jython 実装では、直接または間接に継承できる Java クラスは 1 つだけです。スーパークラスを提供する必要はありません。

スーパークラスのすべての属性やメソッドはいずれのサブクラスにも存在し、そのクラス自体によって使用できるほか、属性やメソッドが隠蔽されていなければ任意のクライアントから使用することもできます。サブクラスのインスタンスは任意の場所で使用でき、スーパークラスのインスタンスも使用できます。これがポリモアフィズム の一例です。これらの機能によって再利用が可能になり、拡張が容易になります。

例

```
class Class1: pass    #no inheritance

class Class2: pass

class Class3(Class1): pass     #single inheritance

class Class4(Class3, Class2): pass     #multiple inheritance
```

第 3 章 IBM SPSS Modeler でのスクリプト

スクリプトの種類

IBM SPSS Modeler には、以下の 3 種類のスクリプトがあります。

- ストリーム・スクリプト は、単一ストリームの実行を制御するために使用され、ストリーム内に格納されます。
- スーパーノード・スクリプト は、スーパーノードの動作を制御するために使用されます。
- スタンドアロン スクリプトまたはセッション・スクリプト は、さまざまなストリームにわたって実行を調整するために使用できます。

さまざまなメソッドを IBM SPSS Modeler のスクリプトで使用することができ、これらメソッドによって SPSS Modeler の広範な機能にアクセスできます。これらのメソッドは、より高度な機能を作成するために 39 ページの『第 4 章 スクリプト API』 でも使用されます。

ストリーム、スーパーノード・ストリーム、およびダイアグラム

多くの場合、ストリーム という語は、ファイルからロードされるストリームであれ、スーパーノード内で使用されるストリームであれ、同じ意味を持ちます。一般に、ストリームは、互いに接続された実行可能なノードの集合を意味します。しかし、スクリプトの場合は、あらゆる場所ですべての操作がサポートされるわけではありません。つまり、スクリプト作成者は、どのストリーム・バリアントを使用しているのかを認識している必要があります。

ストリーム

ストリームは、IBM SPSS Modeler の主なドキュメント・タイプです。ストリームは保存、ロード、編集、および実行することができます。ストリームには、パラメーター、グローバル値、スクリプト、およびその他の情報を関連付けることもできます。

スーパーノード・ストリーム

スーパーノード・ストリーム は、スーパーノード内で使用される種類のストリームです。通常のストリームと同様、互いにリンクされているノードが含まれています。スーパーノード・ストリームは、以下のような点で通常のストリームと異なっています。

- パラメーターおよびスクリプトは、スーパーノード・ストリームではなく、スーパーノード・ストリームを所有しているスーパーノードに関連付けられています。
- スーパーノード・ストリームには、スーパーノードの種類に応じて、追加の入力コネクター・ノードや出力コネクター・ノードがあります。これらのコネクター・ノードは、スーパーノード・ストリームに情報を渡したり、スーパーノード・ストリームから情報を取り出したりするために使用され、スーパーノードの作成時に自動的に作成されます。

ダイアグラム

ダイアグラム という用語は、通常のストリームとスーパーノード・ストリームの両方でサポートされる機能 (ノードの追加や削除、ノード間の接続の変更など) を含んでいます。

ストリームの実行

以下の例は、ストリーム内のすべての実行可能ノードを実行する最もシンプルなタイプのストリーム・スクリプトです。

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

以下の例も、ストリーム内のすべての実行可能ノードを実行します。

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

この例では、ストリームを変数 stream に格納しています。通常、スクリプトはストリームまたはストリーム内のノードを変更するために使用されるため、ストリームを変数に格納すると便利です。ストリームを格納する変数を作成することによって、スクリプトはより簡潔になります。

スクリプト・コンテキスト

modeler.script モジュールは、スクリプトが実行されるコンテキストを提供します。このモジュールは、実行時に SPSS Modeler スクリプトに自動的にインポートされます。このモジュールは、スクリプトがその実行環境にアクセスするための方法を提供する 4 つの関数を定義しています。

- session() 関数は、スクリプトのセッションを返します。セッションは、ストリームを実行するために使用されているロケールや、SPSS Modeler バックエンド (ローカル・プロセス、またはネットワーク SPSS Modeler Server) などの情報を定義します。
- stream() 関数は、ストリームとスーパーノード・スクリプトで使用できます。この関数は、実行中のストリーム・スクリプトまたはスーパーノード・スクリプトを所有しているストリームを返します。
- diagram() 関数は、スーパーノード・スクリプトで使用できます。この関数は、スーパーノード内のダイアグラムを返します。その他のスクリプトのタイプの場合、この関数は stream() 関数と同じ内容を返します。
- supernode() 関数は、スーパーノード・スクリプトで使用できます。この関数は、実行中のスクリプトを所有しているスーパーノードを返します。

これら 4 つの関数と出力を次の表に要約します。

表 *9. modeler.script* 関数の要約

スクリプト・タイプ	**session()**	**stream()**	**diagram()**	**supernode()**
スタンドアロン	セッションを返します	スクリプト起動時の現在の管理対象ストリーム (例えば、バッチ・モード -stream オプションによって渡されたストリーム) か、None を返します。	stream() と同じ	なし
ストリーム	セッションを返します	ストリームを返します	stream() と同じ	なし
スーパーノード	セッションを返します	ストリームを返します	スーパーノード・ストリームを返します	スーパーノードを返します

modeler.script モジュールは、終了コードでスクリプトを終了する方法も定義します。 exit(*exit-code*) 関数は、スクリプトの実行を停止し、指定された整数の終了コードを返します。

ストリーム用に定義されているメソッドの 1 つに runAll(List) があります。このメソッドは、すべての実行可能ノードを実行します。ノードを実行することで生成されるモデルまたは出力は、指定されたリストに追加されます。

通常、ストリームを実行すると、モデルやグラフなどの出力が生成されます。この出力をキャプチャーするために、スクリプトは、リストに初期化される変数を提供できます。例えば、以下のとおりです。

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

実行が完了すると、実行によって生成されたオブジェクトに results リストからアクセスできます。

既存のノードの参照

多くの場合、ストリームは、ストリームの実行前に変更する必要があるいくつかのパラメーターを使用して事前構築されています。これらのパラメーターを変更するには、以下の作業を行います。

1. 関連するストリーム内のノードを見つける。
2. ノードまたはストリーム (あるいは両方) の設定を変更する。

ノードの検索

ストリームでは、さまざまな方法で既存のノードを見つけることができます。これらのメソッドを次の表に要約します。

表 *10.* 既存のノードを見つけるためのメソッド

メソッド	戻り値の型	説明
s.findAll(type, label)	集計棒グラフ	指定したデータ型とラベルを持つすべてのノードのリストを返します。データ型またはラベルのいずれかが None の場合は、もう一方のパラメーターが使用されます。
s.findAll(filter, recursive)	集計棒グラフ	指定したフィルターで受け入れられるすべてのノードの集合を返します。recursive フラグが True の場合は、指定したストリーム内のスーパーノードも検索されます。
s.findByID(id)	ノード	指定した ID のノードを返すか、そのようなノードが存在しない場合は None を返します。検索は現行ストリームに限定されます。
s.findByType(type, label)	ノード	指定したデータ型またはラベルを持つノード、あるいはその両方を持つノードを返します。データ型または名前のいずれかが None の場合は、もう一方のパラメーターが使用されます。一致するノードが複数ある場合は、任意のノードが返されます。一致するノードがない場合、戻り値は None です。

表 10. 既存のノードを見つけるためのメソッド (続き)

メソッド	戻り値の型	説明
s.findDownstream(fromNodes)	集計棒グラフ	指定したノードのリストから検索し、指定したノードの下流にある一連のノードを返します。返されるリストには、最初に指定したノードも含まれます。
s.findUpstream(fromNodes)	集計棒グラフ	指定したノードのリストから検索し、指定したノードの上流にある一連のノードを返します。返されるリストには、最初に指定したノードも含まれます。

例えば、スクリプトがアクセスする必要のある単一のフィルター・ノードがストリームに含まれている場合、そのフィルター・ノードは、以下のスクリプトを使用して見つけることができます。

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

あるいは、ノードの ID (ノード・ダイアログ・ボックスの「注釈」タブに示されている) が分かる場合は、その ID を使用してノードを検索できます。例えば、以下のようにします。

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

プロパティーを設定する

ノード、ストリーム、モデル、および出力のすべてには、アクセス可能で、ほとんどの場合に設定可能なプロパティーがあります。通常、プロパティーは、オブジェクトの動作および外観を変更するために使用されます。オブジェクトのプロパティーのアクセスおよび設定に使用できるメソッドを次の表に要約します。

表 11. オブジェクトのプロパティーのアクセスおよび設定のためのメソッド

メソッド	戻り値の型	説明
p.getPropertyValue(propertyName)	オブジェクト	指定したプロパティーの値を返すか、そのようなプロパティーが存在しない場合は None を返します。
p.setPropertyValue(propertyName, value)	なし	指定したプロパティーの値を設定します。
p.setPropertyValues(properties)	なし	指定したプロパティーの値を設定します。プロパティー・マップの各項目は、プロパティー名を表すキーと、そのプロパティーに割り当てる必要がある値で構成されています。
p.getKeyedPropertyValue(propertyName, keyName)	オブジェクト	指定したプロパティーの値および関連付けられているキーを返すか、そのようなプロパティーまたはキーが存在しない場合は None を返します。
p.setKeyedPropertyValue(propertyName, keyName, value)	なし	指定したプロパティーおよびキーの値を設定します。

例えば、ストリームの先頭にある可変長ファイル・ノードの値を設定する場合は、以下のスクリプトを使用できます。

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

あるいは、フィルター・ノードからフィールドをフィルタリングできます。この場合は、フィールド名に対して値も入力します。例えば、以下のようにします。

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

ノードの作成とストリームの変更

新しいノードを既存のストリームに追加する場合があります既存のストリームにノードを追加するには、通常以下の作業を行います。

1. ノードを作成する。
2. ノードを既存のストリーム・フローにリンクする。

ノードの作成

ストリームでは、さまざまな方法でノードを作成できます。これらのメソッドを次の表に要約します。

表 *12.* ノードを作成するためのメソッド

メソッド	戻り値の型	説明
s.create(nodeType, name)	ノード	指定したデータ型のノードを作成して、指定したストリームに追加します。
s.createAt(nodeType, name, x, y)	ノード	指定したデータ型のノードを作成して、指定したストリームの指定した場所に追加します。x < 0 または y < 0 の場合、場所は設定されません。
s.createModelApplier(modelOutput, name)	ノード	提供されたモデル出力オブジェクトから派生したモデル・アプライヤー・ノードを作成します。

例えば、ストリーム内に新しいデータ型ノードを作成するには、以下のスクリプトを使用できます。

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

ノードのリンクとリンク解除

ストリーム内に新しいノードを作成する場合、そのノードを使用するにはノードのシーケンスに接続する必要があります。ストリームには、ノードをリンクおよびリンク解除するための多くのメソッドがあります。これらのメソッドを次の表に要約します。

表 13. ノードをリンクおよびリンク解除するためのメソッド

メソッド	戻り値の型	説明
s.link(source, target)	なし	ソース・ノードとターゲット・ノードの間に新しいリンクを作成します。
s.link(source, targets)	なし	ソース・ノードと指定されたリスト内の各ターゲットの間に新しいリンクを作成します。
s.linkBetween(inserted, source, target)	なし	他の 2 つのノード・インスタンス (ソース・ノートとターゲット・ノード) の間にノードを接続し、挿入したノードの位置がこれらのノードの間になるように設定します。ソース・ノートとターゲット・ノードの間の直接リンクが最初に削除されます。
s.linkPath(path)	なし	ノード・インスタンスの間の新しいパスを作成します。最初のノードが 2 番目のノードにリンクされ、2 番目のノードが 3 番のノードにリンクされ、以下同様にリンクされます。
s.unlink(source, target)	なし	ソース・ノードとターゲット・ノードの間の直接リンクを削除します。
s.unlink(source, targets)	なし	ソース・ノードと指定されたターゲット・リスト内の各オブジェクトの間の直接リンクを削除します。
s.unlinkPath(path)	なし	ノード・インスタンスの間に存在するパスをすべて削除します。
s.disconnect(node)	なし	指定されたノードと、指定したストリーム内の他のすべてのノードの間のリンクを削除します。
s.isValidLink(source, target)	*boolean*	指定したソース・ノードとターゲット・ノードの間にリンクを作成できる場合は True を返します。このメソッドは、指定したストリームに両方のオブジェクトが属していること、ソース・ノードがリンクを提供でき、ターゲット・ノードがリンクを受け取れること、このようなリンクを作成してもストリーム内に循環が発生しないことを検査します。

以下に示すサンプル・スクリプトは、以下の 5 つのタスクを実行します。

1. 可変長ファイル入力ノード、フィルター・ノード、およびテーブル出力ノードを作成する。
2. ノード同士を接続する。
3. 可変長ファイル入力ノードにファイル名を設定する。
4. 結果出力から「Drug」フィールドをフィルタリングする。
5. テーブル・ノードを実行する。

```
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)
```

ノードのインポート、置換、および削除

ノードの作成や接続だけでなく、多くの場合にストリームのノードの置換や削除も必要です。ノードのインポート、置換、および削除に使用できるメソッドを次の表に要約します。

表 *14.* ノードをインポート、置換、および削除するためのメソッド

メソッド	戻り値の型	説明
s.replace(originalNode, replacementNode, discardOriginal)	なし	指定したストリームの指定したノードを置換します。元のノードと置換ノードの両方が、指定したストリームによって所有されている必要があります。
s.insert(source, nodes, newIDs)	一覧	指定されたリスト内のノードのコピーを挿入します。指定されたリスト内のすべてのノードが、指定したストリームに含まれていると想定されます。newIDs フラグは、ノードごとに新しい ID を生成するのか、または既存の ID をコピーして使用するのかを示します。ストリーム内のすべてのノードの ID は固有であると想定されているため、指定したストリームとソース・ストリームが同じである場合、このフラグを True に設定する必要があります。このメソッドは新しく挿入されたノードのリストを返しますが、ノードの順序は定義されていません (つまり、順序は入力リストのノードの順序と必ずしも同じであるとは限りません)。
s.delete(node)	なし	指定したストリームから指定したノードを削除します。ノードは、指定したストリームによって所有されている必要があります。
s.deleteAll(nodes)	なし	指定したストリームから指定したすべてのノードを削除します。集合内のすべてのノードが、指定したストリームに属している必要があります。
s.clear()	なし	指定したストリームからすべてのノードを削除します。

ストリーム内のノードのトラバース

一般的な要件として、特定のノードの上流または下流にあるノードを識別したい場合があります。ストリームには、これらのノードを識別するために使用できる多くのメソッドがあります。これらのメソッドを次の表に要約します。

表 *15.* 上流または下流のノードを識別するためのメソッド

メソッド	戻り値の型	説明
`s.iterator()`	イテレーター	指定したストリームに含まれているノード・オブジェクトのイテレーターを返します。 `next()` 関数の呼び出しの間にストリームが変更される場合、イテレーターの動作は未定義です。
`s.predecessorAt(node, index)`	ノード	指定したノードの指定された直接の先行ノードを返すか、インデックスが境界を超えている場合は `None` を返します。
`s.predecessorCount(node)`	*int*	指定されたノードの直接の先行ノードの数を返します。
`s.predecessors(node)`	一覧	指定されたノードの直接の先行ノードを返します。
`s.successorAt(node, index)`	ノード	指定したノードの指定した直接の後続ノードを返すか、インデックスが境界を超えている場合は `None` を返します。
`s.successorCount(node)`	*int*	指定されたノードの直接の後続ノードの数を返します。
`s.successors(node)`	一覧	指定されたノードの直接の後続ノードを返します。

項目の消去または削除

従来のスクリプトでは、以下の例のような、`clear` コマンドのさまざまな使用法がサポートされています。

- `clear outputs` は、すべての出力項目をマネージャ パレットから削除します。
- `clear generated palette` は、「モデル」パレットからすべてのモデル ナゲットを消去します。
- `clear stream` は、ストリームの中身を削除します。

Python スクリプトでは、同様の関数セットがサポートされます。ストリーム マネージャ、出力マネージャ、およびモデル マネージャを消去するには、`removeAll()` コマンドを使用します。以下に例を示します。

- ストリーム マネージャを消去する場合:

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

- 出力マネージャを消去する場合:

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

- モデル マネージャを消去する場合:

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

ノードに関する情報の入手

ノードは、データ・インポート・ノードおよびデータ・エクスポート・ノード、モデル構築ノード、その他の種類のノードなど、さまざまなカテゴリーに分類されます。各ノードには、ノードに関する情報を見つけるために使用できる多くのメソッドがあります。

ノードの ID、名前、およびラベルを取得するために使用できるメソッドを次の表に要約します。

表 *16.* ノードの *ID*、名前、およびラベルを取得するためのメソッド

メソッド	戻り値の型	説明
`n.getLabel()`	*string*	指定したノードの表示ラベルを返します。ラベルがプロパティー `custom_name` の値となるのは、このプロパティーが空文字列ではなく、`use_custom_name` プロパティーが設定されていない場合のみです。これ以外の場合、ラベルは `getName()` の値になります。
`n.setLabel(label)`	なし	指定したノードの表示ラベルを設定します。新しいラベルが空文字列ではない場合、この文字列がプロパティー `custom_name` に割り当てられ、指定したラベルが優先されるようにプロパティー `use_custom_name` に `False` が割り当てられます。これ以外の場合は、空文字列が `custom_name` に割り当てられ、プロパティー `use_custom_name` に `True` が割り当てられます。
`n.getName()`	*string*	指定されたノードの名前を戻します。
`n.getID()`	*string*	指定したノードの ID を戻します。新しいノードが作成されるたびに、新しい ID が作成されます。この ID は、ストリームの一部としてノードが保存されるときに、ノードで永続化され、ストリームを開いたときにノード ID が保持されるようになります。ただし、保存したノードがストリームに挿入される場合、挿入されたノードは新しいオブジェクトと見なされ、新しい ID が割り当てられます。

ノードに関するその他の情報を取得するために使用できるメソッドを次の表に要約します。

表 17. ノードに関する情報を取得するためのメソッド

メソッド	戻り値の型	説明
n.getTypeName()	*string*	このノードのスクリプト名を戻します。これは、このノードの新しいインスタンスを作成するために使用できる名前と同じです。
n.isInitial()	*Boolean*	これが最初の ノード (ストリームの先頭にあるノード) である場合は、True を返します。
n.isInline()	*Boolean*	これがインライン・ノード (ストリームの中間にあるノード) である場合は、True を返します。
n.isTerminal()	*Boolean*	これが終端 ノード (ストリームの末尾にあるノード) である場合は、True を返します。
n.getXPosition()	*int*	ストリーム内のノードの x 位置オフセットを返します。
n.getYPosition()	*int*	ストリーム内のノードの y 位置オフセットを返します。
n.setXYPosition(x, y)	なし	ストリーム内のノードの位置を設定します。
n.setPositionBetween(source, target)	なし	指定されたノードの間に位置するようにストリーム内のノードの位置を設定します。
n.isCacheEnabled()	*Boolean*	キャッシュが有効な場合は True を返し、そうでない場合は False を返します。
n.setCacheEnabled(val)	なし	このオブジェクトのキャッシュを有効または無効にします。キャッシュがいっぱいの場合にキャッシュが無効になると、キャッシュはフラッシュされます。
n.isCacheFull()	*Boolean*	キャッシュがいっぱいの場合は True を返し、そうでない場合は False を返します。
n.flushCache()	なし	このノードのキャッシュをフラッシュします。キャッシュが有効でない場合やいっぱいでない場合、影響はありません。

第 4 章 スクリプト API

スクリプト API の概要

スクリプト API により、幅広い SPSS Modeler 機能にアクセスすることができます。ここまで説明してきたメソッドはいずれも API の一部であり、追加でインポートを行わなくてもスクリプト内から暗黙的にアクセスすることができます。ただし、API クラスを参照する必要がある場合は、以下のステートメントで明示的に API をインポートする必要があります。

```
import modeler.api
```

この import ステートメントは、多くのスクリプト API の例で必要になります。

スクリプト API を通じて使用可能なクラス、メソッド、およびパラメータの完全なガイドは、「*IBM SPSS Modeler Python Scripting API Reference Guide*」という文書に含まれています。

例 1: カスタム・フィルターを使用したノードの検索

31 ページの『ノードの検索』のセクションでは、検索基準としてノードのタイプ名を使用してストリームのノードを検索する例を示しました。場合によっては、より汎用的な検索が必要になります。そのような検索を実装するには、NodeFilter クラスおよびストリームの findAll() メソッドを使用します。この種の検索は以下の 2 段階で行います。

1. NodeFilter を拡張し、カスタム・バージョンの accept() メソッドを実装する新しいクラスを作成します。
2. この新しいクラスのインスタンスでストリームの findAll() メソッドを呼び出します。これにより、accept() メソッドで定義された基準を満たすすべてのノードが返されます。

ストリームのノードのうち、ノードのキャッシュが有効になっているノードを検索する方法を以下の例に示します。返されたノードのリストを使用して、それらのノードのキャッシュをフラッシュするか無効化することができます。

```
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)
```

例 2: ユーザーの権限に基づき、ディレクトリーまたはファイルの情報をユーザーが取得できるようにする

ユーザーに PSAPI を公開させないようにするために、PSAPI 関数の呼び出しを介して session.getServerFileSystem() というメソッドを使用し、ファイル・システム・オブジェクトを作成できます。

以下の例は、IBM SPSS Modeler Server に接続するユーザーの権限に基づいて、ディレクトリーまたはファイルの情報をユーザーが取得できるようにする方法を示しています。

```
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()
```

メタデータ: データに関する情報

ストリーム内では複数のノードが互いに接続されているため、各ノードで使用可能な列またはフィールドに関する情報を使用できます。これにより、例えば Modeler UI では、ソートまたは集計の基準となるフィールドを選択できます。この情報はデータ・モデルと呼ばれます。

スクリプトは、ノードを出入りするフィールドを調べることによって、データ・モデルにアクセスすることも可能です。一部のノードでは、入力データ・モデルと出力データ・モデルが同じです。例えば、ソート・ノードは、レコードを並べ替えるだけで、データ・モデルを変更することはありません。一部のノード (フィールド作成ノードなど) では、新しいフィールドを追加できます。他のノード (フィルター・ノードなど) は、フィールドの名前を変更したり、フィールドを削除したりすることができます。

以下の例では、スクリプトは標準の IBM SPSS Modeler druglearn.str ストリームを使用し、いずれかの入力フィールドが欠落した状態のモデルがフィールドごとに構築されます。これは、以下のように行われます。

1. データ型ノードから出力データ・モデルにアクセスする。
2. 出力データ・モデルの各フィールドをループする。
3. 各入力フィールドのフィルター・ノードを変更する。
4. 構築中のモデルの名前を変更する。
5. モデル構築ノードを実行する。

注: druglean.str ストリームのスクリプトを実行する前に、スクリプト言語を Python に設定することを忘れないでください (このストリームは IBM SPSS Modeler の旧バージョンで作成されているため、ストリームのスクリプト言語はレガシーに設定されます)。

```
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([])
```

DataModel オブジェクトには、データ・モデル内のフィールドまたは列に関する情報にアクセスするための多くのメソッドがあります。これらのメソッドを次の表に要約します。

表 *18.* フィールドまたは列に関する情報にアクセスするための *DataModel* オブジェクト・メソッド

メソッド	戻り値の型	説明
d.getColumnCount()	*int*	データ・モデル内の列の数を返します。
d.columnIterator()	イテレーター	各列を「ファイル順」の挿入順序で返すイテレーターを返します。イテレーターは列のインスタンスを返します。
d.nameIterator()	イテレーター	各列の名前を「ファイル順」の挿入順序で返すイテレーターを返します。
d.contains(name)	*Boolean*	指定した名前の列がこの DataModel 内に存在する場合は True を返し、存在しない場合は False を返します。
d.getColumn(name)	列	指定された名前の列を戻します。
d.getColumnGroup(name)	ColumnGroup	指定した列グループを返すか、指定した列グループが存在しない場合は None を返します。
d.getColumnGroupCount()	*int*	このデータ・モデル内の列グループの数を返します。
d.columnGroupIterator()	イテレーター	各列グループを順番に返すイテレーターを返します。
d.toArray()	Column[]	データ・モデルを列の配列として返します。列は「ファイル順」の挿入順序になります。

各フィールド (Column オブジェクト) には、列に関する情報にアクセスするための多くのメソッドが含まれています。以下の表に、これらのメソッドを示します。

表 *19.* 列に関する情報にアクセスするための *Column* オブジェクト・メソッド

メソッド	戻り値の型	説明
c.getColumnName()	*string*	列の名前を戻します。
c.getColumnLabel()	*string*	列のラベルを返すか、列にラベルが関連付けられていない場合は空文字列を返します。
c.getMeasureType()	MeasureType	列の測定タイプを返します。

表 19. 列に関する情報にアクセスするための *Column* オブジェクト・メソッド (続き)

メソッド	戻り値の型	説明
c.getStorageType()	StorageType	列のストレージ・タイプを返します。
c.isMeasureDiscrete()	*Boolean*	列が離散型の場合は True を返します。セット型またはフラグ型の列は、離散型と見なされます。
c.isModelOutputColumn()	*Boolean*	列がモデル出力列の場合は True を返します。
c.isStorageDatetime()	*Boolean*	列のストレージが、時刻、日付、またはタイム・スタンプの値の場合は True を返します。
c.isStorageNumeric()	*Boolean*	列のストレージが整数または実数の場合は True を返します。
c.isValidValue(value)	*Boolean*	指定した値がこのストレージで有効な場合は True を返し、有効な列の値が分かる場合は valid を返します。
c.getModelingRole()	ModelingRole	列のモデル作成の役割を返します。
c.getSetValues()	Object[]	列の有効な値の配列を返すか、値が分からない場合または列がセット型でない場合は None を返します。
c.getValueLabel(value)	*string*	列の値のラベルを返すか、値にラベルが関連付けられていない場合は空文字列を返します。
c.getFalseFlag()	オブジェクト	列の「false」標識値を返すか、値が分からない場合または列がフラグ型でない場合は None を返します。
c.getTrueFlag()	オブジェクト	列の「true」標識値を返すか、値が分からない場合または列がフラグ型でない場合は None を返します。
c.getLowerBound()	オブジェクト	列の値の下限値を返すか、値が分からない場合または列が連続型でない場合は None を返します。
c.getUpperBound()	オブジェクト	列の値の上限値を返すか、値が分からない場合または列が連続型でない場合は None を返します。

列に関する情報にアクセスするほとんどのメソッドには、DataModel オブジェクトに定義されている同等のメソッドがあります。たとえば、次の 2 つのステートメントは、同じものを指します。

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

生成されたオブジェクトへのアクセス

ストリームを実行するには、通常、追加の出力オブジェクトを生成する必要があります。これらの追加のオブジェクトは、新規モデル (以降の実行で使用する情報を提供する出力) にすることができます。

下記の例では、ストリームの開始点として druglearn.str ストリームを再度使用しています。この例では、ストリームのすべてのノードを実行し、結果をリストに格納します。次に、スクリプトでは結果全体についてループし、実行の結果として得られたモデル出力を IBM SPSS Modeler モデル (.gm) ファイルとして保存し、モデルを PMML エクスポートします。

```
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)
```

タスク実行クラスは、よく使用するさまざまな処理を実行するのに便利です。このクラスで使用可能なメソッドの要約を以下の表に示します。

表 *20.* よく使用する処理を実行するためのタスク実行クラスのメソッド

メソッド	戻り値の型	説明
t.createStream(name, autoConnect, autoManage)	ストリーム	新規ストリームを作成して返します。非公開でストリームを作成してユーザーから不可視にする必要があるコードでは、autoManage フラグを False に設定する必要があります。
t.exportDocumentToFile(documentOutput, filename, fileFormat)	なし	指定されたファイル形式を使用してストリームの説明をファイルにエクスポートします。
t.exportModelToFile(modelOutput, filename, fileFormat)	なし	指定されたファイル形式を使用してモデルをファイルにエクスポートします。
t.exportStreamToFile(stream, filename, fileFormat)	なし	指定されたファイル形式を使用してストリームをファイルにエクスポートします。

表 20. よく使用する処理を実行するためのタスク実行クラスのメソッド (続き)

メソッド	戻り値の型	説明
t.insertNodeFromFile(filename, diagram)	ノード	指定されたファイルからノードを読み込み、指定されたダイアグラムに挿入して返します。ノード・オブジェクトとスーパーノード・オブジェクトの両方の読み込みに使用することができます。
t.openDocumentFromFile(filename, autoManage)	DocumentOutput	指定されたファイルからドキュメントを読み込んで返します。
t.openModelFromFile(filename, autoManage)	ModelOutput	指定されたファイルからモデルを読み込んで返します。
t.openStreamFromFile(filename, autoManage)	ストリーム	指定されたファイルからストリームを読み込んで返します。
t.saveDocumentToFile(documentOutput, filename)	なし	指定されたファイルの場所にドキュメントを保存します。
t.saveModelToFile(modelOutput, filename)	なし	指定されたファイルの場所にモデルを保存します。
t.saveStreamToFile(stream, filename)	なし	指定されたファイルの場所にストリームを保存します。

エラーの処理

Python 言語には、try...except コード・ブロックによるエラー処理が備わっています。スクリプト内でこれを使用すると、例外をトラップし、対処しなければスクリプトが終了してしまう問題を処理することができます。

下記のスクリプト例では、IBM SPSS Collaboration and Deployment Services Repository からモデルを取得しようとしています。この操作では例外が発生する可能性があります (例えば、リポジトリーのログイン資格情報が正しく設定されていない場合や、リポジトリーのパスが誤っている場合が考えられます)。スクリプトでその事態が発生すると、ModelerException がスローされます (IBM SPSS Modeler によって生成される例外は、すべて modeler.api.ModelerException から派生しています)。

```
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()
```

注: スクリプト操作によっては、標準の Java 例外が発生する場合があります。それらの例外は ModelerException から派生していません。それらの例外をキャッチするために、追加の except ブロックを使用してすべての Java 例外をキャッチすることができます。以下に例を示します。

```
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()
except java.lang.Exception, e:
    print "A Java exception occurred:", e.getMessage()
```

ストリーム、セッション、およびスーパーノード・パラメーター

パラメーターは、直接スクリプトの中で値を固定的にコーディングするのではなく、実行時に渡す場合に便利です。パラメーターとその値は、ストリームの場合と同じ方法で定義します。つまり、ストリームまたはスーパーノードのパラメーター・テーブルの項目として、またはコマンド・ラインのパラメーターとして定義します。以下の表に示すように、Stream クラスおよび SuperNode クラスは、ParameterProvider オブジェクトによって定義される一連の関数を実装しています。セッションには `getParameters()` の呼び出しが用意されており、呼び出すと、それらの関数を定義するオブジェクトが返されます。

表 *21. ParameterProvider* オブジェクトによって定義されている関数

メソッド	戻り値の型	説明
`p.parameterIterator()`	イテレーター	このオブジェクトのパラメーター名の反復子を返します。
`p.getParameterDefinition(parameterName)`	ParameterDefinition	指定された名前を持つパラメーターのパラメーター定義を返します。該当するパラメーターがこのプロバイダーに存在しない場合は `None` を返します。結果は、メソッドが呼び出された時点での定義のスナップショットである可能性があり、その後にこのプロバイダーを通じてパラメーターに対して行われた変更が反映されているとは限りません。
`p.getParameterLabel(parameterName)`	*string*	指定されたパラメーターのラベルを返します。該当するパラメーターが存在しない場合は `None` を返します。
`p.setParameterLabel(parameterName, label)`	なし	指定されたパラメーターのラベルを設定します。
`p.getParameterStorage(parameterName)`	ParameterStorage	指定されたパラメーターのストレージを返します。該当するパラメーターが存在しない場合は `None` を返します。
`p.setParameterStorage(parameterName, storage)`	なし	指定されたパラメーターのストレージを設定します。
`p.getParameterType(parameterName)`	ParameterType	指定されたパラメーターのデータ型を返します。該当するパラメーターが存在しない場合は `None` を返します。
`p.setParameterType(parameterName, type)`	なし	指定されたパラメーターのデータ型を設定します。
`p.getParameterValue(parameterName)`	オブジェクト	指定されたパラメーターの値を返します。該当するパラメーターが存在しない場合は `None` を返します。

表 21. *ParameterProvider* オブジェクトによって定義されている関数 (続き)

メソッド	戻り値の型	説明
p.setParameterValue(parameterName, value)	なし	指定されたパラメーターの値を設定します。

以下の例では、スクリプトで通信データを集計して、平均収入データが最も低い領域を探します。次に、その領域でストリーム・パラメーターを設定します。さらに、そのストリーム・パラメーターを条件抽出ノードで使用してその領域をデータから除外した後、残りのデータに対する顧客離れモデルを作成します。

この例では、スクリプトで条件抽出ノード自体を生成するため、正しい値を条件抽出ノードの式に直接生成できたという点で、不自然な例になっています。しかし、通常ストリームは事前に作成されているため、この方法でパラメーターを設定すると便利です。

スクリプト例の最初の部分では、平均収入が最も低い領域を格納するストリーム・パラメーターを作成します。また、スクリプトでは集計ブランチとモデル作成ブランチにノードを作成し、相互に接続します。

```
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)
```

このスクリプト例では以下のストリームを作成します。

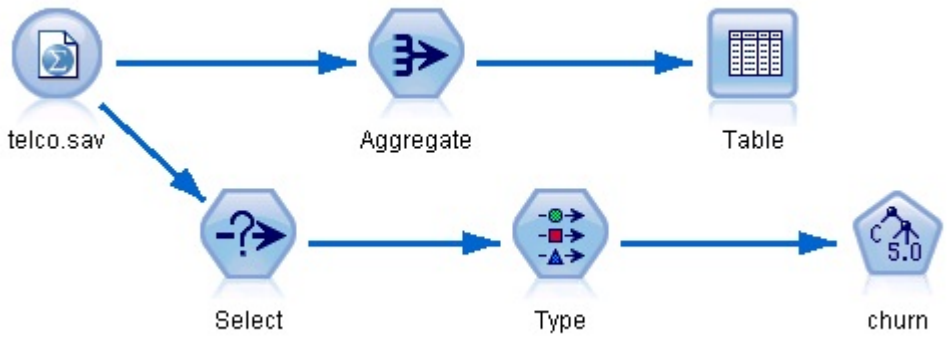

図 5. スクリプト例から得られるストリーム

スクリプト例の以下の部分では、集計ブランチの終端でテーブル・ノードを実行します。

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

スクリプト例の以下の部分では、テーブル・ノードの実行によって生成されたテーブル出力にアクセスします。スクリプトでは次に、テーブルの行全体について反復し、平均収入が最も低い領域を探します。

```
# 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
```

スクリプトの以下の部分では、平均収入が最も低い領域を使用して、以前に作成した「LowestRegion」ストリーム・パラメーターを設定します。スクリプトでは次に、指定の領域を学習データから除外してモデル・ビルダーを実行します。

```
# 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([])
```

スクリプト例全体を以下に示します。

```
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([])
```

グローバル値

グローバル値は、指定したフィールドの各種の要約統計量を計算するために使用します。これらの要約値には、ストリーム内の任意の場所からアクセスできます。グローバル値は、ストリームから名前でアクセスできるという点でストリーム・パラメーターと似ています。ストリーム・パラメーターとの相違点は、スクリプトやコマンド・ラインから代入するのではなく、グローバル値の設定ノードが実行されると関連付けられた値が自動的に更新されることです。ストリームのグローバル値にアクセスするには、ストリームの `getGlobalValues()` メソッドを呼び出します。

GlobalValues オブジェクトは、以下の表に示す関数を定義しています。

表 22. *GlobalValues* オブジェクトによって定義されている関数

メソッド	戻り値の型	説明
`g.fieldNameIterator()`	イテレーター	グローバル値を 1 つ以上持つ各フィールド名の反復子を返します。
`g.getValue(type, fieldName)`	オブジェクト	指定されたデータ型およびフィールド名のグローバル値を返します。値が見つからない場合は `None` を返します。返される値は一般に数値ですが、将来の実装では別の型の値を返すようになる可能性があります。
`g.getValues(fieldName)`	マップ	指定されたフィールド名の既知のエントリーを含むマップを返します。フィールドに既存のエントリーがない場合は `None` を返します。

`GlobalValues.Type` は、使用可能な要約統計量のタイプを定義します。以下の要約統計量が使用可能です。

- `MAX`: フィールドの最大値。
- `MEAN`: フィールドの平均値。
- `MIN`: フィールドの最小値。
- `STDDEV`: フィールドの標準偏差。
- `SUM`: フィールドの値の合計。

例えば、以下のスクリプトは「income」フィールドの平均値にアクセスします。このフィールドは、グローバル値の設定ノードによって計算されます。

```
import modeler.api

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

複数のストリームの処理: スタンドアロン スクリプト

複数のストリームを処理するには、スタンドアロン スクリプトを使用する必要があります。スタンドアロン スクリプトは、IBM SPSS Modeler UI 内で編集して実行するか、バッチ・モードでコマンド・ライン・パラメーターとして渡すことができます。

以下のスタンドアロン スクリプトは 2 つのストリームを開きます。一方のストリームはモデルを作成し、2 番目のストリームは予測値の分布をプロットします。

```
# Change to the appropriate location for your system
demosDir = "C:/Program Files/IBM/SPSS/Modeler/18.1/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()
```

さらに次の例は、開いているストリーム (「ストリーム」タブで開いているすべてのストリーム) を反復処理する方法を示しています。これは、スタンドアロン スクリプトでのみサポートされることに注意してください。

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

第 5 章 スクリプトのヒント

このセクションでは、スクリプトのヒントと使い方について概要を説明します。これには、ストリームの実行を修正したり、スクリプトで暗号化されたパスワードを使用したり、また、IBM SPSS Collaboration and Deployment Services Repository でオブジェクトにアクセスしたりする作業が含まれます。

ストリーム実行の変更

ストリームを実行すると、ターミナル・ノードがデフォルトの状況に最適化された順番で実行されます。状況に応じて、別の順序で実行させることもできます。ストリームの実行順序を変更するには、「ストリームのプロパティー」ダイアログ・ボックスの「実行」タブで、以下の手順を実行します。

1. 空のスクリプトを用意します。
2. ツールバーの 「デフォルト スクリプトを追加」 ボタンをクリックして、デフォルトのストリーム・スクリプトを追加します。
3. デフォルトのストリーム・スクリプトの文の順序を、実際に実行する順序に変更します。

ノードのループ

for ループを使用して、ストリーム内のすべてのノードをループできます。例えば、以下のスクリプト例はすべてのノードをループし、フィルター ノードにおけるフィールド名を大文字に変更します。

実際にフィルター処理されるフィールドがなくても、このスクリプトはフィルター ノードを持つどのようなストリームにおいても使用できます。フィールド名を全面的に大文字に変更するには、すべてのフィールドを渡すフィルター・ノードをただ単に追加するだけです。

```
# 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)
```

このスクリプトは現在のストリーム内のすべてのノードをループし、各ノードがフィルターであるかどうかをチェックします。ノードがフィルターである場合、そのノードの各フィールドをループし、field.upper() 関数または field.getColumnName().upper() 関数を使用して、名前を大文字に変更します。

IBM SPSS Collaboration and Deployment Services Repository 内のオブジェクトへのアクセス

IBM SPSS Collaboration and Deployment Services Repository のライセンスを保有している場合は、スクリプト コマンドを使用して、オブジェクトをリポジトリに格納したり、リポジトリから取得したりできます。リポジトリを使用して、エンタープライズ アプリケーション、ツール、およびソリューションのコンテキストで、データ マイニング モデルおよび関連する予測オブジェクトのライフサイクルを管理します。

IBM SPSS Collaboration and Deployment Services Repository への接続

リポジトリにアクセスするには、まず、SPSS Modeler ユーザー インターフェースの「ツール」メニューまたはコマンド ラインから、リポジトリに対して有効な接続を設定する必要があります。詳しくは、 69 ページの『IBM SPSS Collaboration and Deployment Services Repository 接続の引数』を参照してください。

リポジトリへのアクセスの実行

リポジトリにはセッションからアクセスできます。例えば、以下のようにします。

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

リポジトリからのオブジェクトの取得

スクリプト内で、ストリーム、モデル、出力、およびノードなど、さまざまなオブジェクトにアクセスするには、retrieve* 関数を使用します。取得関数の要約を以下の表に示します。

表 23. 取得スクリプト関数

オブジェクト タイプ	リポジトリ関数
ストリーム	repo.retrieveStream(String path, String version, String label, Boolean autoManage)
モデル	repo.retrieveModel(String path, String version, String label, Boolean autoManage)
出力	repo.retrieveDocument(String path, String version, String label, Boolean autoManage)
ノード	repo.retrieveProcessor(String path, String version, String label, ProcessorDiagram diagram)

例えば、以下の関数を使用してリポジトリからストリームを取得できます。

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

この例は、指定したフォルダーから risk_score.str ストリームを取得します。 ラベルの production はどのバージョンのストリームを取得するかを識別し、最後のパラメーターは SPSS Modeler がそのストリームを管理するかを指定します (例えば、その結果、SPSS Modeler ユーザー インターフェースが表示されている場合に、「ストリーム」タブにそのストリームが表示されます)。代わりに、特定の、ラベル付けのないバージョンを使用するには、以下のようにします。

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

注: バージョンとラベルの両方のパラメーターが None の場合、最新バージョンが返されます。

リポジトリへのオブジェクトの格納

スクリプトを使用してリポジトリにオブジェクトを格納するには、store* 関数を使用します。格納関数の要約を以下の表に示します。

表 24. 格納スクリプト関数

オブジェクト タイプ	リポジトリ関数
ストリーム	repo.storeStream(ProcessorStream stream, String path, String label)
モデル	repo.storeModel(ModelOutput modelOutput, String path, String label)
出力	repo.storeDocument(DocumentOutput documentOutput, String path, String label)
ノード	repo.storeProcessor(Processor node, String path, String label)

例えば、以下の関数を使用して risk_score.str ストリームの新規バージョンを格納できます。

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

この例は、新規バージョンのストリームを格納して、それに "test" ラベルを関連付けて、新規に作成されたバージョンのバージョン マーカーを返します。

注: ラベルを新規バージョンと関連付けたくない場合は、ラベルには None を渡してください。

リポジトリ フォルダの管理

リポジトリ内でフォルダーを使用することで、オブジェクトを論理グループ別に整理でき、オブジェクトの関連がわかりやすくなります。以下の例のように、createFolder() 関数を使用してフォルダーを作成してください。

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

この例は、「/projects」 フォルダーに「cross-sell」という新規フォルダーを作成します。この関数は、新規フォルダーの完全パスを返します。

フォルダーの名前を変更するには、以下のように renameFolder() 関数を使用してください。

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

最初のパラメーターは名前変更されるフォルダーの完全パスであり、2 番目のパラメーターはそのフォルダーにつける新しい名前です。

空のフォルダーを削除するには、以下のように deleteFolder() 関数を使用してください。

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

オブジェクトのロックおよびロック解除

スクリプトから、オブジェクトをロックして、ほかのユーザーが既存のバージョンを更新したり新しいバージョンを作成しないようにすることができます。ロックされたオブジェクトのロックを解除することもできます。

オブジェクトをロックおよびロック解除するシンタックスは次のとおりです。

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

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

オブジェクトの保存および取得同様、REPOSITORY_PATH によって、リポジトリー内のオブジェクトの場所が決められます。パスを引用符で囲み、区切り文字としてスラッシュを使用する必要があります。大文字と小文字は区別しません。

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

また、オブジェクトの場所を決めるには、リポジトリー・パスではなく URI (Uniform Resource Identifier) を使用できます。URI は接頭辞 `spsscr:` を含み、完全に引用符で囲まれている必要があります。パス区切り文字としてはスラッシュだけを使うことができ、スペースは暗号化する必要があります。つまり、パス内ではスペースの代わりに `%20` を使用します。URI では、大文字と小文字は区別しません。いくつか例を挙げると次の通りです。

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

オブジェクトのロックはすべてのバージョンのオブジェクトに適用されます。各バージョンをロックまたはロック解除することはできません。

暗号化パスワードの生成

場合によっては、スクリプトにパスワードを記述する必要があるかも知れません。例えば、パスワードで保護されたデータ・ソースにアクセスしたい場合などです。暗号化パスワードは、次の場所で使用することができます。

- データベース入力ノードおよび出力ノードのノード・プロパティー。
- サーバーにログインするためのコマンド・ライン引数。
- エクスポート・ノードの「公開」タブから生成するパラメーター・ファイル *.par* ファイルに保管されるデータベース接続プロパティー。

ユーザー・インターフェースから、Blowfish アルゴリズムに基づいた暗号化パスワードを生成することができます (詳細については、http://www.schneier.com/blowfish.html を参照してください)。パスワードを暗号化したら、そのパスワードをコピーしてスクリプト・ファイルやコマンド・ライン引数に指定することができます。`databasenode` および `databaseexportnode` に使用するノード・プロパティー `epassword` は暗号化パスワードを格納します。

1. 暗号化パスワードを生成するには、「ツール」メニューから次の項目を選択します。

   「パスワードのエンコード**...**」
2. 「パスワード」ボックスにパスワードを指定します。
3. 「暗号化」 をクリックすると、ランダムに暗号化されたパスワードが生成されます。
4. 「コピー」ボタンをクリックすると、暗号化されたパスワードがクリップボードにコピーされます。
5. パスワードを目的のスクリプトやパラメーターに貼り付けます。

スクリプトの検査

「スタンドアロン スクリプト」ダイアログ ボックスのツールバーにある赤い検査ボタンをクリックすれば、あらゆるタイプのスクリプトの構文を迅速に検査できます。

図 *6.* ストリーム・スクリプトのツールバー・アイコン

スクリプトの検査時にコードにエラーがあった場合、エラーを警告するメッセージと推奨する修正方法が表示されます。エラーのある行を表示するには、ダイアログ・ボックスの下部にあるフィードバック情報をクリックしてください。エラーが赤で強調表示されます。

コマンド・ラインからのスクリプト

通常はユーザー・インターフェースから行われるような操作を、スクリプトで実行することができます。IBM SPSS Modeler を起動する時には、コマンド・ライン上でスタンドアロン・ストリームを指定して実行してください。以下に例を示します。

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

-script フラグは指定されたスクリプトをロードすることを、-execute フラグはスクリプト・ファイル中のすべてのコマンドを実行することを示しています。

旧リリースとの互換性

以前の IBM SPSS Modeler のリリースで作成されたスクリプトは、通常現在のリリースでも変更なしで動作します。ただし、モデル・ナゲットがストリームに自動的に挿入され (デフォルト設定)、ストリーム内のその種類の既存ナゲットを置き換えまたは補足する場合があります。これが実際に行われるかどうかは、「モデルをストリームに追加」オプションおよび「前のモデルを置換」オプション (「ツール」>「オプション」>「ユーザー オプション」>「通知」) の設定によって異なります。例えば、既存のナゲットを削除して新しいナゲットを挿入し、ナゲットの置換を処理する旧リリースからのスクリプトの変更が必要な場合があります。

現在のリリースで作成したスクリプトは、以前のリリースでは動作しないことがあります。

古いリリースで作成されたスクリプトがあるコマンドを使用し、そのコマンドがリリースされてから他のコマンドに置き換えられて (または、廃止されて) いる場合は、古い形が依然としてサポートされますが、同時に警告メッセージも表示されます。例えば、古い generated キーワードは model に、clear generated は clear generated palette に置き換えられます。古い形を使うスクリプトは依然として動作しますが、警告も表示されます。

ストリーム実行結果へのアクセス

多くの IBM SPSS Modeler ノードで、モデル、グラフ、およびテーブル形式データなどの出力オブジェクトが生成されます。これらの出力の多くに、それ以降の実行の指針とするためにスクリプトが使用できる有用な値が含まれています。これらの値は、コンテンツ コンテナ (単にコンテナと呼ばれる) にグループ化されます。コンテナには、各コンテナを識別するタグまたは ID を使用してアクセスできます。これらの値にアクセスする方法は、そのコンテナが使用する形式 (「コンテンツ モデル」) によって異なります。

例えば、多くの予測モデル出力では、PMML という XML の一種を使用して、モデルに関する情報 (各分割でディシジョン ツリーが使用するフィールドや、ニューラル ネットワーク内のニューロンの接続方法とその強度など) を表現します。PMML を使用するモデル出力では、その情報にアクセスするために使用できる XML コンテンツ モデルを提供します。以下に例を示します。

```
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]

# Now that we have the C5.0 model output object, access the
# relevant content model
cm = modeloutput.getContentModel("PMML")

# 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")
```

IBM SPSS Modeler は、スクリプトで以下のコンテンツ モデルをサポートします。

- テーブル コンテンツ モデル: 行と列として表現される単純なテーブル形式データにアクセスできます。
- **XML** コンテンツ モデル: XML 形式で保管されたコンテンツにアクセスできます。
- **JSON** コンテンツ モデル: JSON 形式で保管されたコンテンツにアクセスできます。
- 列統計コンテンツ モデル: 特定のフィールドに関する統計の要約にアクセスできます。
- ペアごとの列統計コンテンツ モデル: 2 つのフィールドの間の統計の要約または 2 つの個別のフィールドの間にある値にアクセスできます。

テーブル コンテンツ モデル

テーブル コンテンツ モデルは、単純な行と列のデータにアクセスするための単純なモデルを提供します。特定の列内の値は、すべてストレージのタイプが同じでなければなりません (例えば、文字列または整数)。

API

表 *25. API*

戻り値	メソッド	説明
int	getRowCount()	このテーブル内の行の数を返します。
int	getColumnCount()	このテーブル内の列の数を返します。
String	getColumnName(int columnIndex)	指定された列インデックス位置にある列の名前を返します。列のインデックスは 0 から始まります。
StorageType	getStorageType(int columnIndex)	指定されたインデックス位置にある列のストレージ タイプを返します。列のインデックスは 0 から始まります。
Object	getValueAt(int rowIndex, int columnIndex)	指定された行インデックスおよび列インデックスの位置にある値を返します。行と列のインデックスは 0 から始まります。
void	reset()	このコンテンツ モデルに関連付けられた内部ストレージをすべて消去します。

ノードおよび出力

この表では、このタイプのコンテンツ モデルを含む出力を作成するノードをリストします。

表 *26.* ノードおよび出力

ノード名	出力名	コンテナ **ID**
table	table	"table"

スクリプトの例

```
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
```

スクリプトの「デバッグ」タブには、以下のような出力が表示されます。

```
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 コンテンツ モデル

XML コンテンツ モデルでは、XML ベースのコンテンツにアクセスできます。

XML コンテンツ モデルは、XPath 式に基づくコンポーネントにアクセスする機能をサポートします。XPath 式は、呼び出し元がどの要素または属性を必要とするかを定義する文字列です。XML コンテンツ モデルは、さまざまなオブジェクトの作成と、XPath のサポートで通常必要となる式のコンパイルについて、詳細な内容を隠します。これにより、Python スクリプトからの呼び出しが単純になります。

XML コンテンツ モデルには、XML 文書を文字列として返す関数が含まれています。これにより、Python スクリプト ユーザーは、自分にとって望ましい Python ライブラリを使用して XML を解析できます。

API

表 27. API

戻り値	メソッド	説明
String	getXMLAsString()	XML を文字列として返します。
数値	getNumericValue(String xpath)	パスを評価した結果を数値として返します (例えば、パス式に一致する要素の数をカウントします)。
boolean	getBooleanValue(String xpath)	指定されたパス式を評価した結果をブール値として返します。
String	getStringValue(String xpath, String attribute)	指定されたパスに一致する、属性値または XML ノード値のいずれかを返します。
文字列のリスト	getStringValues(String xpath, String attribute)	指定されたパスに一致するすべての属性値または XML ノード値のリストを返します。
文字列のリストのリスト	getValuesList(String xpath, <List of strings> attributes, boolean includeValue)	指定されたパスに一致するすべての属性値のリストを、必要な場合は XML ノード値と共に返します。
ハッシュ テーブル (key:string, value:list of string)	getValuesMap(String xpath, String keyAttribute, <List of strings> attributes, boolean includeValue)	キー属性または XML ノード値をキーとして使用するハッシュ テーブルを返し、また、指定された属性値のリストをテーブル値として返します。
boolean	isNamespaceAware()	XML パーサーが名前空間を認識している必要があるかどうかを返します。デフォルトは False です。
void	setNamespaceAware(boolean value)	XML パーサーが名前空間を認識している必要があるかどうかを設定します。このメソッドでは、後続の呼び出しで変更内容が取得されるようにするために reset() も呼び出します。
void	reset()	このコンテンツ モデルに関連付けられた内部ストレージをすべて消去します (キャッシュされた DOM オブジェクトなど)。

ノードおよび出力

この表では、このタイプのコンテンツ モデルを含む出力を作成するノードをリストします。

表 *28.* ノードおよび出力

ノード名	出力名	コンテナ **ID**
ほとんどのモデル ビルダー	ほとんどの生成されたモデル	"PMML"
"autodataprep"	なし	"PMML"

スクリプトの例

コンテンツにアクセスするための Python スクリプトのコードは、以下のようになります。

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

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

JSON コンテンツ モデル

JSON コンテンツ モデルは、JSON 形式のコンテンツのサポートを提供するために使用されます。このモデルでは、どの値にアクセスするかを呼び出し元が認識していることを前提として、呼び出し元が値を抽出できるようにする基本的な API が提供されます。

API

表 *29. API*

戻り値	メソッド	説明
String	getJSONAsString()	JSON コンテンツを文字列として返します。
Object	getObjectAt(<List of cbjecta> path, JSONArtifact artifact) throws Exception	指定されたパスのオブジェクトを返します。指定されたルート成果物がヌルである可能性があり、その場合はコンテンツのルートが使用されます。返される値は、リテラル文字列、整数、実数、またはブール値であるか、あるいは JSON 成果物 (JSON オブジェクトまたは JSON 配列のいずれか) である可能性もあります。
ハッシュ テーブル (key:object, value:object)	getChildValuesAt(<List of object> path, JSONArtifact artifact) throws Exception	パスが JSON オブジェクトを指す場合は、指定されたパスの子値を返します。それ以外の場合はヌルを返します。テーブル内のキーは文字列ですが、関連付けられている値は、リテラル文字列、整数、実数、またはブール値であるか、あるいは JSON 成果物 (JSON オブジェクトまたは JSON 配列のいずれか) である可能性もあります。

表 29. API (続き)

戻り値	メソッド	説明
オブジェクトのリスト	getChildrenAt(<List of object> path path, JSONArtifact artifact) throws Exception	パスが JSON 配列を指す場合は、指定されたパスのオブジェクトのリストを返します。それ以外の場合はヌルを返します。返される値は、リテラル文字列、整数、実数、またはブール値であるか、あるいは JSON 成果物 (JSON オブジェクトまたは JSON 配列のいずれか) である可能性もあります。
void	reset()	このコンテンツ モデルに関連付けられた内部ストレージをすべて消去します (キャッシュされた DOM オブジェクトなど)。

スクリプトの例

JSON 形式に基づいて出力を作成する出力ビルダーノードがある場合は、以下のコードを使用して、ブックのセットに関する情報にアクセスすることができます。

```
results = []
outputbuilder.run(results)
output = results[0]
cm = output.getContentModel("jsonContent")

bookTitle = cm.getObjectAt(["books", "ISIN123456", "title"], None)

# Alternatively, get the book object and use it as the root
# for subsequent entries
book = cm.getObjectAt(["books", "ISIN123456"], None)
bookTitle = cm.getObjectAt(["title"], book)

# Get all child values for aspecific book
bookInfo = cm.getChildValuesAt(["books", "ISIN123456"], None)

# Get the third book entry. Assumes the top-level "books" value
# contains a JSON array which can be indexed
bookInfo = cm.getObjectAt(["books", 2], None)

# Get a list of all child entries
allBooks = cm.getChildrenAt(["books"], None)
```

列統計コンテンツ モデルおよびペアごとの統計コンテンツ モデル

列統計コンテンツ モデルでは、フィールドごとに計算できる統計 (1 変量の統計) にアクセスできます。ペアごとの統計コンテンツ モデルでは、フィールドのペア間で計算できる統計またはフィールド内の値にアクセスできます。

統計の尺度には以下のものがあります。

- Count
- UniqueCount
- ValidCount

- 平均値
- Sum
- Min
- Max
- 範囲
- 分散
- StandardDeviation
- StandardErrorOfMean
- 歪度
- SkewnessStandardError
- 尖度
- KurtosisStandardError
- Median
- Mode
- Pearson
- Covariance (共分散)
- TTest
- FTest

一部の値は単一の列統計の場合のみに該当し、その他の値はペアごとの統計の場合のみに該当します。

これらを生成するノードを以下に示します。

- 記述統計ノード: 列統計を生成し、相関フィールドが指定されている場合はペアごとの統計を生成できます。
- データ検査ノード: 列を生成し、オーバーレイ フィールドが指定されている場合はペアごとの統計を生成できます。
- 平均ノード: フィールドのペアを比較するとき、またはあるフィールドの値を他のフィールド要約と比較するときに、ペアごとの統計を生成します。

使用可能なコンテンツ モデルと統計は、その特定のノードの機能とそのノード内の設定の両方によって決まります。

ColumnStatsContentModel API

表 30. *ColumnStatsContentModel API*:

戻り値	メソッド	説明
List<StatisticType>	getAvailableStatistics()	このモデルで使用可能な統計を返します。必ずしもすべてのフィールドがすべての統計の値を持つわけではありません。
List<String>	getAvailableColumns()	統計が計算された対象の列名を返します。
Number	getStatistic(String column, StatisticType statistic)	列に関連付けられた統計値を返します。

表 30. *ColumnStatsContentModel API (続き)*:

戻り値	メソッド	説明
void	reset()	このコンテンツ モデルに関連付けられた内部ストレージをすべて消去します。

PairwiseStatsContentModel API

表 31. *PairwiseStatsContentModel API*:

戻り値	メソッド	説明
List<StatisticType>	getAvailableStatistics()	このモデルで使用可能な統計を返します。必ずしもすべてのフィールドがすべての統計の値を持つわけではありません。
List<String>	getAvailablePrimaryColumns()	統計が計算された対象の 1 次列名を返します。
List<Object>	getAvailablePrimaryValues()	統計が計算された対象の 1 次列の値を返します。
List<String>	getAvailableSecondaryColumns()	統計が計算された対象の 2 次列名を返します。
Number	getStatistic(String primaryColumn, String secondaryColumn, StatisticType statistic)	列に関連付けられた統計値を返します。
Number	getStatistic(String primaryColumn, Object primaryValue, String secondaryColumn, StatisticType statistic)	1 次列値と 2 次列に関連付けられた統計値を返します。
void	reset()	このコンテンツ モデルに関連付けられた内部ストレージをすべて消去します。

ノードおよび出力

この表では、このタイプのコンテンツ モデルを含む出力を作成するノードをリストします。

表 32. ノードおよび出力:

ノード名	出力名	コンテナ **ID**	注
"means" (平均ノード)	"means"	"columnStatistics"	
"means" (平均ノード)	"means"	"pairwiseStatistics"	
"dataaudit" (データ検査ノード)	"means"	"columnStatistics"	
"statistics" (記述統計ノード)	"statistics"	"columnStatistics"	特定のフィールドが検証された場合のみ生成されます。

表 32. ノードおよび出力 (続き):

ノード名	出力名	コンテナ ID	注
"statistics" (記述統計ノード)	"statistics"	"pairwiseStatistics"	フィールドが相関している場合のみ生成されます。

スクリプトの例

```
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.getStatistic(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
```

第 6 章 コマンド・ライン引数

ソフトウェアの起動

オペレーティング・システムのコマンド・ラインを使用し、次のようにして IBM SPSS Modeler を起動できます。

1. IBM SPSS Modeler がインストールされているコンピューターで、DOS つまりコマンド・プロンプト・ウィンドウを開きます。
2. IBM SPSS Modeler インターフェースをインタラクティブ・モードで起動するには、`modelerclient` コマンドを入力し、続いて例えば次のような適切な引数を入力します。

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

使用可能な引数 (フラグ) により、サーバーへの接続、ストリームのロード、スクリプトの実行、または必要に応じて他のパラメーターの指定を行うことができます。

コマンド・ライン引数の使用

IBM SPSS Modeler の起動を変更するために、コマンド・ラインの引数 (フラグ とも呼ばれます) を初期の `modelerclient` コマンドに追加できます。

複数の種類のコマンド・ライン引数を使用できます。これらのコマンド・ライン引数についてはこのセクションで後述します。

表 *33.* コマンド・ライン引数の種類：

引数の種類	参照箇所
システムの引数	詳しくは、トピック 66 ページの『システムの引数』を参照してください。
パラメーターの引数	詳しくは、トピック 67 ページの『パラメーターの引数』を参照してください。
サーバー接続の引数	詳しくは、トピック 68 ページの『サーバー接続の引数』を参照してください。
IBM SPSS Collaboration and Deployment Services Repository 接続の引数	詳しくは、トピック 69 ページの『IBM SPSS Collaboration and Deployment Services Repository 接続の引数』を参照してください。
IBM SPSS Analytic Server 接続の引数	詳しくは、トピック 70 ページの『IBM SPSS Analytic Server 接続の引数』を参照してください。

例えば、以下のようにして `-server`、`-stream` および `-execute` のフラグ型を使用してサーバーに接続し、ストリームをロードおよび実行できます。

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

ローカル・クライアントのインストールと競合する場合、サーバー接続の引数は不要です。

スペースを含むパラメーター値は二重引用符で囲むことができます。例えば、次のようになります。

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

また、IBM SPSS Modeler のステートとスクリプトも、それぞれ -state フラグと -script フラグを使用して、この方法で実行できます。

注: コマンドで構造化パラメータを使用する場合は、引用符の前に円記号を置く必要があります。これにより、文字列の解釈中に引用符が削除されなくなります。

コマンド・ライン引数のデバッグ

コマンド・ラインをデバッグするには modelerclient コマンドを使用し、適切な引数を使用して IBM SPSS Modeler を起動します。これにより、コマンドが予定通りに実行されることを検証できます。また、「セッション パラメーター」ダイアログ・ボックス（「ツール」メニュー、セッション パラメーターの設定）のコマンド・ラインから渡されるパラメーターの値を確認することもできます。

システムの引数

ユーザー・インターフェースのコマンド・ラインによる起動で利用できるシステム引数を次の表に示します。

表 *34.* システムの引数

引数	動作説明
@ <commandFile>	@ 文字に続けてファイル名を記述することにより、コマンド・リストを指定することができます。modelerclient コマンドに @ から始まる引数を指定すると、その引数に指定されたコマンド・ファイル中のコマンドが、コマンド・ラインに指定されているのと同じように処理されます。詳しくは、トピック 70 ページの『複数の引数の組み合わせ』を参照してください。
-directory <dir>	デフォルトの作業ディレクトリーを設定します。ローカル・モードでは、このディレクトリーはデータと出力の両方で使用されます。例: -directory c:/ または -directory c:¥¥
-server_directory <dir>	デフォルトのデータ用サーバー・ディレクトリーを設定します。-directory フラグで指定された作業ディレクトリーは、出力に使用されます。
-execute	起動後に、起動時にロードされたストリーム、ステート、またはスクリプトを実行します。ストリームやステートではなくスクリプトがロードされた場合は、スクリプトだけが実行されます。
-stream <ストリーム>	起動時に、指定したストリームをロードします。複数のストリームを指定できますが、最後に指定したストリームが現在のストリームに設定されます。
-script <スクリプト>	起動時に、指定したスタンドアロン スクリプトをロードします。下で説明しているストリームやステートに加えてこれも指定できますが、起動時には 1 つのスクリプトしかロードできません。
-model <モデル>	起動時に、指定の生成モデル (.gm 形式ファイル) をロードします。
-state <ステート>	起動時に、指定した保存済みのステートをロードします。
-project <プロジェクト>	指定したプロジェクトをロードします。起動時には、プロジェクトを 1 つしかロードできません。
-output <出力>	起動時に、保存された出力オブジェクト (.cou 形式ファイル) をロードします。
-help	コマンド・ライン引数のリストを表示します。このオプションを指定すると、他の引数はすべて無視されて、ヘルプ画面が表示されます。
-P <name>=<value>	スタートアップ・パラメーターの設定に使用されます。ノードのプロパティー (スロット・パラメーター) の設定に使用することもできます。

注: ユーザー・インターフェースでデフォルト・ディレクトリーも設定できます。このオプションにアクセスするには、「ファイル」メニューの「作業ディレクトリーの設定」または「サーバー ディレクトリーの設定」を選択します。

複数ファイルのロード

ロードされた各オブジェクトに対応する引数を繰り返し指定して、起動時にコマンド・ラインから、複数のストリーム、ステート、および出力をロードすることができます。例えば、report.str と train.str の 2 種類のストリームをロード、実行するには、コマンド・ラインに次のコマンドを指定します。

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

IBM SPSS Collaboration and Deployment Services Repository からのオブジェクトのロード

ファイルまたは IBM SPSS Collaboration and Deployment Services Repository (ライセンスがある場合) から特定のオブジェクトを読み込むことができるため、ファイル名の接頭辞 spsscr: および、オプションで file: (ディスク上のオブジェクト) が IBM SPSS Modeler にオブジェクトの検索場所を示します。上記の接頭辞は、次のフラグに適用できます。

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

接頭辞を使用して、オブジェクトの場所を指定する URI を作成します。例えば、次のようになります。-stream "spsscr:///folder_1/scoring_stream.str"。spsscr: の接頭辞がある場合、IBM SPSS Collaboration and Deployment Services Repository への有効な接続を同じコマンドで指定する必要があります。そのため、例えば、フル・コマンドは次のようになります。

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

コマンド・ラインから URI を使用する必要がある ことに注意してください。単純な REPOSITORY_PATH はサポートされていません (その場合は、スクリプト内でのみ作動します)。IBM SPSS Collaboration and Deployment Services Repository 中のオブジェクトの URI 詳細については、 52 ページの『IBM SPSS Collaboration and Deployment Services Repository 内のオブジェクトへのアクセス』を参照してください。

パラメーターの引数

IBM SPSS Modeler のコマンド・ライン実行時に、パラメーターをフラグとして使用することができます。コマンド・ラインの引数に -P フラグを使用して、-P *<name>=<value>* の形式でパラメーターを表すことができます。

パラメーターは、次のいずれかになります。

- 単純なパラメーター（または、CLEM 式で直接使用されるパラメーター）。
- スロット・パラメーター (ノードのプロパティーと呼ばれることもある)。これらのパラメーターは、ストリーム中のノードの設定を変更するために使用されます。 詳しくは、トピック 75 ページの『ノードのプロパティーの概要』を参照してください。

- IBM SPSS Modeler の起動を変更するために用いられる、コマンド・ライン・パラメーター。

例えば、データ・ソースのユーザー名とパスワードを、次のようにコマンド・ラインのフラグとして指定することができます。

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

形式は、databasenode ノード・プロパティーの datasource パラメーターの形式と同じです。詳しくは、88 ページの『databasenode プロパティー』を参照してください。

注: ノードの名前を指定する場合、二重引用符でノード名を囲み、それらの引用符を円記号でエスケープする必要があります。例えば、直前の例のデータ ソース ノード名前が *Source_ABC* である場合、入力は以下のようになります。

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

以下の TM1 データ ソースの例のように、構造化パラメーターを示す引用符の前には円記号も必要です。

```
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%¥"}
```

サーバー接続の引数

-server フラグは、IBM SPSS Modeler にパブリック・サーバーに接続するよう指示し、-hostname、-use_ssl、-port、-username、-password、および -domain のフラグを使用して、IBM SPSS Modeler にパブリック・サーバーに接続する方法を指示します。-server 引数が指定されていない場合、デフォルト・サーバーまたはローカル・サーバーが使用されます。

例

パブリック・サーバーに接続するには

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

サーバー・クラスターに接続するには

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

サーバー・クラスターに接続するには、IBM SPSS Collaboration and Deployment Services を使用した Coordinator of Processes が必要です。したがって、-cluster 引数をリポジトリー接続オプション (spsscr_*) とともに使用する必要があります。 詳しくは、トピック 69 ページの『IBM SPSS Collaboration and Deployment Services Repository 接続の引数』を参照してください。

表 *35.* サーバー接続の引数：

引数	動作説明
-server	IBM SPSS Modeler をサーバー・モードで実行し、フラグ -hostname、-port、-username、-password、および -domain を使用してパブリック・サーバーに接続します。
-hostname <name>	サーバー・マシンのホスト名を指定します。サーバー・モードでしか利用できません。

表 35. サーバー接続の引数 (続き):

引数	動作説明
-use_ssl	接続で使用する SSL (secure socket layer) を指定します。このフラグはオプションです。SSL 使用時のデフォルト設定は *not* です。
-port <number>	指定したサーバーのポート番号。サーバー・モードでしか利用できません。
-cluster <name>	名前付きサーバーではなく、サーバー・クラスターへの接続を指定します。この引数は hostname、port 、および use_ssl 引数の代替です。name はクラスター名、または IBM SPSS Collaboration and Deployment Services Repository 内のクラスターを識別する一意の URI です。サーバー・クラスターは、IBM SPSS Collaboration and Deployment Services を使用して Coordinator of Processes で管理されます。 詳しくは、トピック『IBM SPSS Collaboration and Deployment Services Repository 接続の引数』を参照してください。
-username <name>	サーバーにログオンするためのユーザー名。サーバー・モードでしか利用できません。
-password <password>	サーバーにログオンするためのパスワード。サーバー・モードでしか利用できません。 **注:** -password 引数を使用しない場合、パスワードの入力を要求するプロンプトが表示されます。
-epassword <encodedpasswordstring>	サーバーにログオンするための暗号化パスワード。サーバー・モードでしか利用できません。 **注:** 暗号化パスワードは、IBM SPSS Modeler アプリケーションの「ツール」メニューから生成することができます。
-domain <name>	サーバーにログオンする際に使用するドメイン名。サーバー・モードでしか利用できません。
-P <name>=<value>	スタートアップ・パラメーターの設定に使用されます。ノードのプロパティー (スロット・パラメーター) の設定に使用することもできます。

IBM SPSS Collaboration and Deployment Services Repository 接続の引数

コマンド・ラインを経由して IBM SPSS Collaboration and Deployment Services でオブジェクトを保存したり取り出したりするには、IBM SPSS Collaboration and Deployment Services Repositoryに有効な接続を指定する必要があります。以下に例を示します。

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

接続を設定するために使用できる引数の一覧を次の表に示します。

表 36. *IBM SPSS Collaboration and Deployment Services Repository* 接続の引数

引数	動作説明
-spsscr_hostname <ホスト名または IP アドレス>	IBM SPSS Collaboration and Deployment Services Repository がインストールされているサーバーのホスト名または IP アドレスです。
-spsscr_port <number>	IBM SPSS Collaboration and Deployment Services Repository が接続を承認したポート番号です（通常、8080 がデフォルト値）。
-spsscr_use_ssl	接続で使用する SSL (secure socket layer) を指定します。このフラグはオプションです。SSL 使用時のデフォルト設定は *not* です。

表 36. *IBM SPSS Collaboration and Deployment Services Repository* 接続の引数 (続き)

引数	動作説明
-spsscr_username <name>	IBM SPSS Collaboration and Deployment Services Repository にログオンするためのユーザー名。
-spsscr_password <password>	IBM SPSS Collaboration and Deployment Services Repository にログオンするためのパスワード。
-spsscr_epassword <encoded password>	IBM SPSS Collaboration and Deployment Services Repository にログオンするためのエンコードされたパスワード。
-spsscr_providername <name>	IBM SPSS Collaboration and Deployment Services Repository (Active Directory または LDAP) へのログオンに使用する認証プロバイダー。これは、ネイティブ (ローカル・リポジトリー) のプロバイダーを使用する場合は不要です。

IBM SPSS Analytic Server 接続の引数

コマンド ラインを使用して IBM SPSS Analytic Server でオブジェクトを保存したり取り出したりするには、IBM SPSS Analytic Server への有効な接続を指定する必要があります。

注**:** Analytic Server のデフォルトの場所は、SPSS Modeler Server から取得されます。ユーザーは、「ツール」 > 「**Analytic Server** 接続」を使用して、独自の Analytic Server 接続を定義することもできます。

接続を設定するために使用できる引数の一覧を次の表に示します。

表 37. *IBM SPSS Analytic Server* 接続の引数

引数	動作説明
-analytic_server_username	IBM SPSS Analytic Server にログオンするためのユーザー名。
-analytic_server_password	IBM SPSS Analytic Server にログオンするためのパスワード。
-analytic_server_epassword	IBM SPSS Analytic Server にログオンするための暗号化パスワード。
-analytic_server_credential	IBM SPSS Analytic Server にログオンするために使用する資格情報。

複数の引数の組み合わせ

複数の引数を記述したコマンド・ファイルを作成し、起動時に @ 記号に続けてそのファイル名を指定することができます。こうすることによって、コマンド・ラインによる起動を短縮し、OS によるコマンド長の制限に関する問題を解決することができます。例えば、以下の起動コマンドは <commandFileName> が示すファイルに指定されている引数を使用します。

```
modelerclient @<commandFileName>
```

ファイル名やコマンド・ファイルへのパスにスペースがある場合は、以下のようにして引用符で囲みます。

```
modelerclient @ "C:¥Program Files¥IBM¥SPSS¥Modeler¥nn¥scripts¥my_command_file.txt"
```

このコマンド・ファイルには、スタートアップ時に個別に指定していたすべての引数を記述することができます。以下に例を示します。

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

コマンド・ファイルを記述して、コマンド・ファイル名を指定する場合の制限事項を次に示します。

- 1 行につき 1 つの引数またはコマンドを記述する必要があります。
- コマンド・ファイル内に、`@CommandFile` 引数を組み込まないでください。

第 7 章 プロパティー・リファレンス

プロパティー・リファレンスの概要

ノード、ストリーム、プロジェクト、スーパーノードに対して、数多くのさまざまなプロパティーを指定できます。名前、注釈、およびツールヒントなど、すべてのノードに共通のプロパティーもありますが、その一方で、ノードのタイプに固有なプロパティーもあります。キャッシングやスーパーノードの動作などの高レベルなストリーム操作を参照するプロパティーもあります。プロパティーは、標準のユーザー・インターフェースからアクセスでき (ノードのオプションを編集するダイアログ・ボックスを開く場合など)、また、多くの標準とは異なる方法でも使用できます。

- プロパティーは、このセクションで説明されているように、スクリプトからアクセスできます。詳しくは、『プロパティーのシンタックス』を参照してください。
- ノードのプロパティーは、スーパーノード・パラメーター中で使用することができます。
- ノードのプロパティーは、IBM SPSS Modeler の起動時にコマンド・ライン・オプションの一部として使用することもできます (-P フラグを使用)。

IBM SPSS Modeler のスクリプトでは、ノードおよびストリームのプロパティーは、よくスロット・パラメーターと呼ばれます。このガイドでは、スロット・パラメーターをノードまたはストリームのプロパティーと記載しています。

スクリプト言語の詳細は、スクリプト言語を参照してください。

プロパティーのシンタックス

プロパティーは、以下のシンタックスを使用して設定できます。

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

または

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

プロパティーの値は、以下のシンタックスを使用して取得できます。

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

または

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

ここで、OBJECT はノードまたは出力、PROPERTY は式で参照しているノード プロパティーの名前、KEY はキー プロパティーのキー値です。例えば、以下のシンタックスを使用して、フィルター ノードを検索し、すべてのフィールドを含むようにデフォルトを設定し、下流データから Age フィールドをフィルタリングします。

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

ストリームの findByType(TYPE, LABEL) 関数を使用すると、IBM SPSS Modeler で使用されているすべてのノードを検索することができます。少なくとも TYPE または LABEL のいずれかを指定する必要があります。

構造化プロパティー

スクリプト解析時の明確性を向上するために構造化プロパティーを使用するには、次の 2 種類の方法があります。

- データ型、フィルター、またはバランス・ノードなどの、複雑なノードのプロパティー名を構造化する。
- 複数のプロパティーを同時に指定する形式を提供する。

複雑なインターフェースの構造化

テーブルや他の複雑なインターフェースがあるノード、例えば、データ型、フィルター、およびバランス・ノードなどを対象とするスクリプトは、正しく解析されるために一定の構造を遵守する必要があります。これらの構造化プロパティーには、1 つの識別子名と比べてより複雑な名前が必要です。これらのプロパティーでは、単一の識別子の名前よりも複雑な名前が必要であり、この名前はキーと呼ばれます。この情報を参照するため、フィルター・ノードではフィールドごとに 1 つの情報項目 (各フィールドが真か偽か) が保存されます。この情報を参照するために、フィルター ノードはフィールドごとに 1 つの情報項目を保管します (各フィールドが true か false か)。このプロパティーには、真 (True) または偽 (False) の値が設定されているか、または指定される可能性があります。mynode というフィルター・ノード (上流側) に、Age というフィールドがある場合を考えてみましょう。これをオフにするには、次のように、キー Age と値 False を指定してプロパティー include を設定します。

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

複数のプロパティーの設定構造

多数のノードに対して、複数のノードおよびストリームのプロパティーを同時に割り当てることができます。これは、**multiset** コマンドまたはセット ブロックと呼ばれています。

場合によっては、構造化プロパティーがきわめて複雑なこともあります。以下に例を示します。

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

構造化プロパティーのもう 1 つの利点は、ノードが安定していなくてもそのノード上に複数のプロパティーが設定できることです。デフォルトでは、multiset はブロック内のすべてのプロパティーを設定してから、個別のプロパティー設定に基づいてアクションを実行します。例えば固定長ノードを定義するときに、フィールド・プロパティーを 2 ステップに分けて設定するとエラーが生じます。これは、両方の設定が有効になるまでノードが一貫しないためです。プロパティーを multiset として定義すれば、データ・モデルを更新する前に両方のプロパティーが設定でき、エラーが回避されます。

省略形

ノードのプロパティーのシンタックスでは、標準省略形が使用されています。省略形を覚えておけば、スクリプトの作成に役立ちます。

表 *38.* シンタックスで使用される標準省略形

省略形	意味
abs	絶対値
len	長さ
最小	最小値
最大	最大値
correl	相関
covar	共分散

表 38. シンタックスで使用される標準省略形 (続き)

省略形	意味
num	数字または数値
pct	パーセントまたは割合
transp	透過性
xval	交差検証
var	分散または変数 (入力ノードで)

ノードおよびストリームのプロパティーの例

ノードおよびストリームのプロパティーは、IBM SPSS Modeler のさまざまな場面で使用されます。一般的にこれらのプロパティーは、複数のストリームや操作を自動化するために用いられるスタンドアロン スクリプト、または単一のストリーム内のプロセスの自動化に用いられるストリーム・スクリプトなど、スクリプトの一部として使われます。スーパーノード内で、ノードのプロパティーを使用してノード・パラメーターを指定することもできます。もっとも基本的なレベルで、IBM SPSS Modeler の起動時にコマンド・ライン・オプションとしてプロパティーを指定することもできます。コマンド・ラインの起動時に、-p 引数を指定すれば、ストリーム・プロパティーを使用してストリームの設定を変更することができます。

表 39. ノードおよびストリームのプロパティーの例

プロパティー	意味
s.max_size	ノード s のプロパティー max_size を表します。
s:samplenode.max_size	ノード s のプロパティー max_size を表します。このノードは、サンプリング・ノードでなければなりません。
:samplenode.max_size	現在のストリーム中のサンプリング・ノードの、プロパティー max_size を表します (サンプリング・ノードは 1 つだけでなければなりません)。
s:sample.max_size	ノード s のプロパティー max_size を表します。このノードは、サンプリング・ノードでなければなりません。
t.direction.Age	データ型ノード t の *Age* フィールドの役割を表します。
:.max_size	*** 無効 *** ノード名またはノードの種類を指定する必要があります。

s:sample.max_size の例は、ノードの種類を完全に記述する必要がないことを示しています。

t.direction.Age の例は、1 つのノードの属性が個別の値を持つ単純な個々のスロットよりも複雑な場合に、一部のスロット名を構造化できることを示しています。このようなスロットは、構造化または複雑なプロパティーと呼ばれます。

ノードのプロパティーの概要

ノードの種類ごとに、独自の有効なプロパティーのセットが用意されています。また、各プロパティーにはデータ型があります。一般的なデータ型の数値、フラグ、または文字列の場合、プロパティーの設定は強制的に正しいデータ型に設定されます。強制的に設定できない場合はエラーが発生します。それに対し、プロパティー参照が、Discard、PairAndDiscard、および IncludeAsText のような有効な値の範囲を指定していることもあります。この場合、範囲外の値が使われた場合にエラーになります。フラグ型プロパティーは、true および false の値を使用して読み込まれるか、設定される必要があります (Off、OFF、off、No、

NO、no、n、N、f、F、false、False、FALSE、または 0 なども値の設定時に認識されますが、プロパティー値の読み込み時にエラーが発生する場合もあります。その他の値はすべて真と見なされます。true と false を使用すると、こうした混乱が避けられます)。このガイドにある参照テーブルでは、構造化プロパティーはそのまま「プロパティーの説明」欄に、使用形式とともに記載されています。

共通のノード・プロパティー

数多くのプロパティーが、IBM SPSS Modeler 中のすべてのノード (スーパーノードも含む) で共通に使われています。

表 *40.* 共通のノード・プロパティー：

プロパティー名	データ型	プロパティーの説明
use_custom_name	*flag*	
name	*string*	ストリーム領域上のノード名を対象とする読み込み専用プロパティーです (自動またはユーザー設定)。
custom_name	*string*	ノードのカスタム(ユーザー設定)名を指定します。
tooltip	*string*	
annotation	*string*	
keywords	*string*	オブジェクトに関連付けられているキーワードのリストを指定する構造化スロットです (例: ["Keyword1" "Keyword2"])。
cache_enabled	*flag*	
node_type	source_supernode process_supernode terminal_supernode スクリプト用に指定する すべてのノード名	ノードをタイプごとに参照するために使用される読み込み専用プロパティー。例えば、ノードを real_income のような名前だけで参照する代わりに、userinputnode または filternode のようなタイプで指定することもできます。

スーパーノード固有のプロパティーは、他のノードと同様に、個別に説明します。 詳しくは、トピック 365 ページの『第 20 章 スーパーノードのプロパティー』を参照してください。

第 8 章 Stream プロパティー

スクリプトにより、さまざまなストリームのプロパティーを制御することができます。ストリームのプロパティーを参照するには、以下のような、スクリプトを使用するための実行メソッドを設定する必要があります。

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

例

ノード プロパティーを使用して、現在のストリーム内の各ノードが参照されます。次のストリーム・スクリプトに、その例を示します。

```
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)
```

この例では、ノード プロパティーを使用して、ストリーム内のすべてのノードのリストを作成し、そのリストをストリームの注釈に書き込んでいます。この注釈は、次のようになります。

```
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"
```

ストリームのプロパティーを次の表に示します。

表 *41. Stream* プロパティー：

プロパティー名	データ型	プロパティーの説明
execute_method	Normal Script	

表 *41. Stream* プロパティー *(*続き*)*:

プロパティー名	データ型	プロパティーの説明
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*	

表 41. *Stream* プロパティー (続き):

プロパティー名	データ型	プロパティーの説明
ruleset_evaluation	Voting FirstHit	
refresh_source_nodes	*flag*	ストリーム実行時に、入力ノードを自動的にリフレッシュするために使用します。
script	*string*	
annotation	*string*	
name	*string*	**注:** このプロパティーは読み取り専用です。ストリーム名を変更する場合は、別名で保存する必要があります。
parameters		スタンドアロン スクリプト内からストリーム・パラメーターを更新する場合に、このプロパティーを使用します。
nodes		詳細は以下を参照してください。
encode	SystemDefault "UTF-8"	
stream_rewriting	*boolean*	
stream_rewriting_maximise_sql	*boolean*	
stream_rewriting_optimise_clem_execution	*boolean*	
stream_rewriting_optimise_syntax_execution	*boolean*	
enable_parallelism	*boolean*	
sql_generation	*boolean*	
database_caching	*boolean*	
sql_logging	*boolean*	
sql_generation_logging	*boolean*	
sql_log_native	*boolean*	
sql_log_prettyprint	*boolean*	
record_count_suppress_input	*boolean*	
record_count_feedback_interval	*integer*	
use_stream_auto_create_node_設定	*boolean*	true の場合はストリーム固有の設定が使用されます。それ以外の場合はユーザー設定が使用されます。
create_model_applier_for_new_モデル	*boolean*	true の場合、モデル・ビルダーが新しいモデルを作成するときにアクティブな更新リンクがなければ、新しいモデル・アプライヤーが追加されます。 **注:** IBM SPSS Modeler Batch バージョン 15 を使用している場合は、スクリプト内で明示的にモデル アプライヤを追加する必要があります。

表 *41. Stream* プロパティー *(*続き*)*:

プロパティー名	データ型	プロパティーの説明
create_model_applier_update_links	createEnabled createDisabled doNotCreate	モデル・アプライヤー・ノードの自動追加時に作成するリンクの種類を定義します。
create_source_node_from_builders	*boolean*	true の場合、ソース・ビルダーが新しいソース出力を作成するときにアクティブな更新リンクがなければ、新しい入力ノードが追加されます。
create_source_node_update_links	createEnabled createDisabled doNotCreate	入力ノードの自動追加時に作成するリンクの種類を定義します。
has_coordinate_system	*boolean*	これを true に設定すると、ストリーム全体に座標系が適用されます。
coordinate_system	*string*	選択された投影座標系の名前。
deployment_area	ModelRefresh Scoring None	ストリームの展開方法を選択します。この値を None に設定すると、他の展開エントリーは使用されません。
scoring_terminal_node_id	*string*	ストリームのスコアリングブランチを選択します。ストリーム内のどのターミナル・ノードでもスコアリング・ブランチとして使用できます。
scoring_node_id	*string*	スコアリングブランチのナゲットを選択します。
model_build_node_id	*string*	ストリームのモデル作成ノードを選択します。

第 9 章 入力ノードのプロパティー

ソース・ノードの共通プロパティー

すべての入力ノードに共通するプロパティーを次に一覧にします。その後に、特定のノードに関する情報が続きます。

例 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")
```

例 2

このスクリプトは、指定されたデータ ファイルに、複数行の文字列を表す Region というフィールドが含まれていることを前提とします。

```
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", StorageType.LIST)
# ...and specify the type if values in the list and the list depth
varfilenode.setKeyedPropertyValue("custom_list_storage_type", "Region", StorageType.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.GEOSPATIAL)
# ...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")
```

表 42. ソース・ノードの共通プロパティー：

プロパティー名	データ型	プロパティーの説明
direction	Input Target Both None Partition Split 頻度 RecordID	フィールドの役割のキープロパティー。 使用形式： NODE.direction.FIELDNAME **注:** 値 In と Out は廃止されました。今後のリリースではサポートが中断される場合があります。
type	Range Flag Set Typeless Discrete Ordered Set Default	フィールドのデータ型。このプロパティーを *Default* に設定すると、values プロパティーに関するすべての値は消去され、value_mode を *Specify* に設定すると、それが *Read* にリセットされます。value_mode が *Pass* または *Read* がすでに設定されている場合、type の設定によって影響を受けることはありません。 使用形式： NODE.type.FIELDNAME
storage	Unknown String Integer 実数 Time Date Timestamp	フィールドのストレージ・タイプ用読み込み専用キー・プロパティー。 使用形式： NODE.storage.FIELDNAME
check	None Nullify Coerce Discard Warn Abort	フィールド・タイプと範囲の検査用のキー・プロパティー。 使用形式： NODE.check.FIELDNAME
values	[*value value*]	連続型 (範囲) フィールドの場合、最初の値が最小値で最後の値が最大値になります。名義型 (セット型) フィールドの場合、すべての値を指定します。フラグ型の場合、最初の値が *false* (偽) を、最後の値が *true* (真) を表します。このプロパティーを設定すると、value_mode プロパティーの値が自動的に *Specify* に設定されます。ストレージは、リストの最初の値に基づいて決まります。例えば、最初の値が*string* の場合、ストレージは String に設定されます。 使用形式： NODE.values.FIELDNAME
value_mode	Read Pass Read+ Current Specify	次のデータの受け渡し時にフィールドに値を設定する方法を決定します。 使用形式： NODE.value_mode.FIELDNAME このプロパティーに *Specify* を直接には設定できないことに注意してください。特定の値を使用するには、values プロパティーを設定します。

表 42. ソース・ノードの共通プロパティー (続き) :

プロパティー名	データ型	プロパティーの説明
default_value_mode	Read Pass	すべてのフィールドに値を設定するためのデフォルトの方法を指定します。 使用形式 : NODE.default_value_mode この設定による特定のフィールドの設定は、value_mode プロパティーを使用するとオーバーライドされることがあります。
extend_values	*flag*	value_mode が *Read* に設定された場合に適用されます。新しく読み込んだ値を、フィールドの既存の値に追加する場合は、*T* を設定します。新しく読み込んだ値を優先して、既存の値を破棄する場合は、*F* を設定します。 使用形式 : NODE.extend_values.FIELDNAME
value_labels	*string*	値ラベルの指定に使用します。数値を先に指定します。
enable_missing	*flag*	*T* を設定した場合、フィールドの欠損値の追跡が有効になります。 使用形式 : NODE.enable_missing.FIELDNAME
missing_values	[*value value ...*]	欠損データを示すデータ値を指定します。 使用形式 : NODE.missing_values.FIELDNAME
range_missing	*flag*	プロパティーが *T* に設定されている場合、フィールドに欠損値 (空白) の範囲が定義されているかどうかを指定します。 使用形式 : NODE.range_missing.FIELDNAME
missing_lower	*string*	range_missing が真 (true) の場合、欠損値範囲の下限値を指定します。 使用形式 : NODE.missing_lower.FIELDNAME
missing_upper	*string*	range_missing が真 (true) の場合、欠損値範囲の上限値を指定します。 使用形式 : NODE.missing_upper.FIELDNAME
null_missing	*flag*	このプロパティーが *T* に設定されていると、ヌル (ソフトウェアでは $null$ として表示される未定義値) は欠損値と見なされます。 使用形式 : NODE.null_missing.FIELDNAME
whitespace_missing	*flag*	このプロパティーが *T* に設定されていると、空白値 (スペース、タブ、および改行) だけを含む値は欠損値とみなされます。 使用形式 : NODE.whitespace_missing.FIELDNAME
description	*string*	フィールドのラベルまたは説明の指定に使用します。

表 42. ソース・ノードの共通プロパティー (続き) :

プロパティー名	データ型	プロパティーの説明
default_include	*flag*	デフォルトの処理としてフィールドを通過させるかフィルターをかけるかの指定をするキー・プロパティー。 NODE.default_include 例: set mynode:filternode.default_include = false
include	*flag*	各フィールドを適用するかフィルターをかけるかを決定するキー・プロパティー: NODE.include.FIELDNAME.
new_name	*string*	
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	このキー付きプロパティーは、フィールドに関連付けられた尺度を定義するために使用できるという点で、type と類似しています。異なるのは、Python スクリプトで、getter 関数が常に MeasureType 値を返す一方で、setter 関数に MeasureType 値のうちの 1 つを渡すこともできるという点です。
collection_measure	Range / MeasureType.RANGE Flag / MeasureType.FLAG Set / MeasureType.SET OrderedSet / MeasureType.ORDERED_SET Typeless / MeasureType.TYPELESS	収集フィールド (深さが 0 のリスト) の場合、このキー付きプロパティーは、基礎となる値に関連付けられた尺度タイプを定義します。
geo_type	Point MultiPoint LineString MultiLineString Polygon MultiPolygon	地理空間フィールドの場合、このキー付きプロパティーにより、このフィールドが表す地理空間オブジェクトのタイプが定義されます。これは、値のリストの深さと整合している必要があります。
has_coordinate_system	*boolean*	地理空間フィールドの場合、このプロパティーにより、このフィールドに座標系があるかどうかが定義されます。
coordinate_system	*string*	地理空間フィールドの場合、このキー付きプロパティーにより、このフィールドの座標系が定義されます。

表 42. ソース・ノードの共通プロパティー (続き) :

プロパティー名	データ型	プロパティーの説明
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	このキー付きプロパティーは、フィールドのオーバーライド ストレージを定義するために使用できるという点で、custom_storage と類似しています。異なるのは、Python スクリプトで、getter 関数が常に StorageType 値を返す一方で、setter 関数に StorageType 値のうちの 1 つを渡すこともできるという点です。
custom_list_storage_type	String / MeasureType.STRING Integer / MeasureType.INTEGER Real / MeasureType.REAL Time / MeasureType.TIME Date / MeasureType.DATE Timestamp / MeasureType.TIMESTAMP	リスト フィールドの場合、このキー付きプロパティーにより、基礎となる値のストレージ タイプが指定されます。
custom_list_depth	*integer*	リスト フィールドの場合、このキー付きプロパティーにより、フィールドの深さが指定されます。
max_list_length	*integer*	地理空間または集合のいずれかの尺度を持つデータのみに使用できます。リストの最大長を設定するには、リストに入れることができる要素の数を指定します。
max_string_length	*integer*	データ型不明 のデータでのみ使用可能で、SQL を生成してテーブルを作成するときに使用されます。データの最大文字列の値を入力します。これにより、テーブルに生成される列が、その文字列を含めるのに十分な大きさになります。

asimport プロパティー

Analytic Server 入力により、Hadoop 分散ファイル・システム (HDFS) でストリームを実行することができます。

例

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

表 43. *asimport* プロパティー :

asimport プロパティー	データ型	プロパティーの説明
data_source	*string*	データ・ソースの名前。

表 43. *asimport* プロパティー (続き):

asimport プロパティー	データ型	プロパティーの説明
use_default_as	*boolean*	True に設定した場合、サーバーの options.cfg ファイルで構成されているデフォルトの Analytic Server 接続が使用されます。False に設定した場合、このノードの接続が使用されます。
connection	["文字列","文字列","文字列", "文字列","文字列","文字列","文字列", "文字列","文字列","文字列", "文字列" ,"文字列"]	Analytic Server 接続の詳細を含むリストのプロパティー。形式は次のとおりです: ["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"]。ここで、is_secure_connect はセキュア接続が使用されるかどうかを示し、値は true または false です。use-kerberos-auth は Kerberos 認証が使用されるかどうかを示し、値は true または false です。enable-kerberos-debug は Kerberos 認証のデバッグ・モードが使用されるかどうかを示し、値は true または false です。

cognosimport ノードのプロパティー

IBM Cognos ソース・ノードは、Cognos Analytics データベースからデータをインポートします。

例

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

表 44. *cognosimport* ノードのプロパティー：

cognosimport ノードのプロパティー	データ型	プロパティーの説明
mode	Data Report	Cognos データ (デフォルト) またはレポートをインポートするかどうかを指定します。
cognos_connection	*["文字列",フラグ,"文字列", "文字列" ,"文字列"]*	Cognos サーバーの接続の詳細を含むリストのプロパティー。形式は以下のとおりです。 ["Cognos_server_URL", login_mode, "namespace", "username", "password"] ここで、 Cognos_server_URL は、ソースが格納されている Cognos サーバーの URL です。 login_mode は、匿名ログインを使用するかどうかを示し、true または false のいずれかになります。true に設定する場合は、以下の各フィールドを必ず "" に設定してください。 namespace はサーバーへのログオンに使用するセキュリティー認証プロバイダを示します。 username および password は Cognos サーバーにログオンする際に使用するユーザー名とパスワードです。 login_mode の代わりに、以下のモードも使用可能です。 • anonymousMode。例: ['Cognos_server_url', 'anonymousMode', "namespace", "username", "password"] • credentialMode。例: ['Cognos_server_url', 'credentialMode', "namespace", "username", "password"] • storedCredentialMode。例: ['Cognos_server_url', 'storedCredentialMode', "stored_credential_name"] ここで、stored_credential_name は、リポジトリー内での Cognos の資格情報の名前です。
cognos_package_name	*string*	データ・オブジェクトをインポートしている Cognos データ・ソース (通常はデータベース) のパスおよび名前。次に例を示します。 /Public Folders/GOSALES **注:** スラッシュのみが有効です。
cognos_items	*["field","field", ... ,"field"]*	インポートする 1 つまたは複数のデータ・オブジェクトの名前。*field* の形式は、[*namespace*].[*query_subject*].[*query_item*] です。

表 44. *cognosimport* ノードのプロパティー (続き):

cognosimport ノードのプロパティー	データ型	プロパティーの説明
cognos_filters	*field*	データをインポートする前に適用するフィルターの名前。
cognos_data_parameters	*list*	データのプロンプト・パラメーターの値。名前と値のペアは大括弧で囲み、複数のペアはコンマで区切り、文字列全体は大括弧で囲みます。 書式: [["*param1*", "*value*"],...,["*paramN*", "*value*"]]
cognos_report_directory	*field*	レポートをインポートするフォルダーまたはパッケージの Cognos パス。次に例を示します。 /Public Folders/GOSALES **注:** スラッシュのみが有効です。
cognos_report_name	*field*	インポートするレポートのレポートの位置内にあるパスと名前。
cognos_report_parameters	*list*	レポート・パラメーターの値。名前と値のペアは大括弧で囲み、複数のペアはコンマで区切り、文字列全体は大括弧で囲みます。 書式: [["*param1*", "*value*"],...,["*paramN*", "*value*"]]

databasenode プロパティー

データベース・ノードは、Microsoft SQL Server、DB2、Oracle など ODBC (開放型データベース接続) を使用するさまざまなパッケージからデータをインポートするのに使用できます。

例

```
import modeler.api
stream = modeler.script.stream()
nnode = 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")
```

表 45. *databasenode* プロパティー：

databasenode プロパティー	データ型	プロパティーの説明
mode	Table Query	ダイアログ・ボックスのコントロールを使用してデータベースに接続するには、*Table* を指定します。SQL を使用して選択されたデータベースにクエリーを行うには、*Query* を指定します。
datasource	*string*	データベース名 (下記の注意を参照)。
username	*string*	データベース接続の詳細 (下記の注意を参照)。
password	*string*	
credential	*string*	IBM SPSS Collaboration and Deployment Services に保管されている資格情報の名前。このプロパティーは、username プロパティーや password プロパティーの代わりに使用することができます。資格情報のユーザー名とパスワードは、データベースにアクセスするためのユーザー名とパスワードに一致している必要があります。
use_credential		True または False に設定します。
epassword	*string*	スクリプト内でパスワードをハードコード化する代わりに、エンコードされたパスワードを指定します。 詳しくは、トピック 54 ページの『暗号化パスワードの生成』を参照してください。このプロパティーは、実行時に読み取り専用になります。
tablename	*string*	アクセスするテーブルの名前。
strip_spaces	None Left Right Both	文字列の前後のスペースを破棄するためのオプションです。
use_quotes	AsNeeded Always Never	クエリーをデータベースに送信するときにテーブル名と列名を引用符で囲むかどうかを指定します (例えば、テーブル名と列名にスペースや句読点が含まれているような場合)。
query	*string*	送信するクエリーを表す SQL コードを指定します。

注: データベース名 (datasource プロパティー内) に 1 つ以上のスペース、ピリオド (「終止符」とも呼ばれる)、または下線が含まれる場合は、「円記号と二重引用符」形式を使用して、それを文字列として扱うことができます。例えば、"{¥"db2v9.7.6_linux¥"}" または "{¥"TDATA 131¥"}"。さらに、datasource 文字列の値は常に次の例のように二重引用符と中括弧で囲みます。"{¥"SQL Server¥",spssuser,abcd1234,false}"。

注: データベース名 (datasource プロパティー内) にスペースが含まれる場合、datasource、username、および password の個別のプロパティーの代わりに、次の形式で単一のデータ ソース プロパティーを使用することもできます。

表 46. *databasenode* プロパティー - *datasource* 固有：

databasenode プロパティー	データ型	プロパティーの説明
datasource	*string*	書式： [database_name,username,password[,true | false]] 暗号化パスワードと使用しないパラメーターです。true に設定すると、パスワードが使用前に復号化されます。

データ・ソースを変更する場合、この形式を使用します。ただし、ユーザー名またはパスワードを変更する場合、username プロパティーまたは password プロパティーを使用できます。

datacollectionimportnode プロパティー

Data Collection データ・インポート・ノードは、市場調査製品で使用される Data Collection Data Model に基づいた調査データをインポートします。このノードを使用するには、Data Collection Data Library がインストールされている必要があります。

例

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

表 47. *datacollectionimportnode* プロパティー：

datacollectionimportnode プロパティー	データ型	プロパティーの説明
metadata_name	*string*	MDSC の名前。特殊な値の DimensionsMDD は、標準的な Data Collection メタデータ・ドキュメントが使用される必要のあることを示します。ほかに、次の値を指定できます。 mrADODsc mrI2dDsc mrLogDsc mrQdiDrsDsc mrQvDsc mrSampleReportingMDSC mrSavDsc mrSCDsc mrScriptMDSC 特殊な値の none は、MDSC がないことを示します。

表 47. *datacollectionimportnode* プロパティー (続き):

datacollectionimportnode プロパティー	データ型	プロパティーの説明
metadata_file	*string*	メタデータが格納されるファイルの名前。
casedata_name	*string*	CDSC の名前。使用できる値は以下のとおりです。 mrADODsc mrI2dDsc mrLogDsc mrPunchDSC mrQdiDrsDsc mrQvDsc mrRdbDsc2 mrSavDsc mrScDSC mrXmlDsc 特殊な値の none は、CDSC がないことを示します。
casedata_source_type	Unknown File Folder UDL DSN	CDSC のソース・タイプを示します。
casedata_file	*string*	casedata_source_type が *File* のときに、ケース・データが含まれるファイルを指定します。
casedata_folder	*string*	casedata_source_type が *Folder* のときに、ケース・データが含まれるフォルダーを指定します。
casedata_udl_string	*string*	casedata_source_type が *UDL* のときに、ケース・データが含まれるデータ・ソースのための OLD-DB 接続文字列を指定します。
casedata_dsn_string	*string*	casedata_source_type が *DSN* のときに、データ・ソースのための ODBC 接続文字列を指定します。
casedata_project	*string*	Data Collection データベースからケース・データを読み込むときに、プロジェクトの名前を入力できます。その他のケース・データのデータ型については、この設定を空白のままにしておく必要があります。
version_import_mode	All Latest Specify	各バージョンの取り扱い方法を定義します。
specific_version	*string*	version_import_mode が *Specify* のときに、インポートされるケース・データのバージョンを定義します。
use_language	*string*	特定言語のラベルが使用される必要があるかどうかを定義します。

表 47. *datacollectionimportnode* プロパティー (続き):

datacollectionimportnode プロパティー	データ型	プロパティーの説明
language	*string*	use_language が真 (true) の場合、入力に使用する言語コードを定義します。言語コードは、ケース・データ内で利用できる中の 1 つにする必要があります。
use_context	*string*	特定のコンテキストが入力される必要があるかどうかを定義します。コンテキストは、応答に関連する説明を多様化させるために使用されます。
context	*string*	use_context が真 (true) の場合、入力するコンテキストを定義します。コンテキストは、ケース・データ内で利用できる中の 1 つにする必要があります。
use_label_type	*string*	特定のラベル タイプが入力される必要があるかどうかを定義します。
label_type	*string*	use_label_type が真 (true) の場合、入力するラベル・タイプを定義します。ラベル・タイプは、ケース・データ内で利用できる中の 1 つにする必要があります。
user_id	*string*	明示的なログインが必要なデータベースの場合、データ・ソースにアクセスするためのユーザー ID とパスワードを提供できます。
password	*string*	
import_system_variables	Common None All	インポートされるシステム変数を指定します。
import_codes_variables	*flag*	
import_sourcefile_variables	*flag*	
import_multi_response	MultipleFlags Single	

excelimportnode プロパティー

Excel インポート ノードは、Microsoft Excel から .xlsx ファイル形式でデータをインポートします。ODBC データ・ソースは不要です。

例

```
#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")
```

表 *48. excelimportnode* プロパティー：

excelimportnode プロパティー	データ型	プロパティーの説明
excel_file_type	Excel2007	
full_filename	*string*	パスを含む、完全なファイル名。
use_named_range	*Boolean*	名前付けられた範囲を使用するかどうかを指定します。真の場合、読み込む範囲を指定するのに named_range プロパティーが使用され、その他のワークシートとデータ範囲の設定は無視されます。
named_range	*string*	
worksheet_mode	Index Name	ワークシートがインデックスで定義されているのか (Index)、または名前で定義されているのか (Name) を指定します。
worksheet_index	*integer*	読み込むべきワークシートのインデックス。最初のワークシートは 0、2 番目は 1、というようにインデックスが指します。
worksheet_name	*string*	読み込むべきワークシートの名前。
data_range_mode	FirstNonBlank ExplicitRange	範囲の決定方法を指定します。
blank_rows	StopReading ReturnBlankRows	data_range_mode が *FirstNonBlank* のときに、空白行の処理方法を指定します。
explicit_range_start	*string*	data_range_mode が *ExplicitRange* のときに、読み込む範囲の開始点を指定します。
explicit_range_end	*string*	
read_field_names	*Boolean*	指定された範囲の最初の行がフィールド (列) 名として使用されるかどうかを指定します。

extensionimportnode プロパティー

拡張インポート・ノードを使用すると、R スクリプトまたは Python for Spark スクリプトを実行して、データをインポートできます。

Python for Spark の例

```
##### 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)
```

R の例

```
#### 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/RJSONIO
# '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)
```

表 *49. extensionimportnode* プロパティー

extensionimportnode プロパティー	データ型	プロパティーの説明
syntax_type	*R* *Python*	R または Python のどちらのスクリプトを実行するか指定します (R がデフォルトです)。
r_syntax	*string*	実行する R スクリプト・シンタックス。
python_syntax	*string*	実行する Python スクリプト・シンタックス。

fixedfilenode プロパティー

固定長ノードで、固定長フィールド・テキスト・ファイルからデータをインポートします。ここで、ファイルのフィールドは区切られていませんが、同じ位置から始まって長さは固定されています。コンピューター生成のデータや、旧来のシステムのデータなどは、しばしば固定長フィールド形式で保存されています。

例

```
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)
```

表 *50. fixedfilenode* プロパティー :

fixedfilenode プロパティー	データ型	プロパティーの説明
record_len	*number*	各レコードの文字数を指定します。
line_oriented	*flag*	各レコードの末尾の改行文字をスキップします。
decimal_symbol	Default Comma Period	データ・ソースで使われている小数点記号。
skip_header	*number*	最初のレコードの先頭で無視する行数を指定します。列見出しを無視する場合などに役立ちます。
auto_recognize_datetime	*flag*	入力データの日付または時刻を自動的に特定するかどうかを指定します。

表 *50. fixedfilenode* プロパティー *(*続き*)*:

fixedfilenode プロパティー	データ型	プロパティーの説明
lines_to_scan	*number*	
fields	*list*	構造化プロパティー。
full_filename	*string*	読み込みファイルのディレクトリーを含む完全な名前。
strip_spaces	None Left Right Both	インポート時に文字列の前後のスペースを破棄します。
invalid_char_mode	Discard Replace	データ入力から不正な文字 (ヌル、0、または現在のエンコード中に存在していない文字) をデータ入力から削除するか (Discard)、指定された 1 文字の記号で不正な文字を置き換えます (Replace)。
invalid_char_replacement	*string*	
use_custom_values	*flag*	
custom_storage	Unknown String Integer 実数 Time Date Timestamp	

表 50. *fixedfilenode* プロパティー (続き):

fixedfilenode プロパティー	データ型	プロパティーの説明
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" "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	このプロパティーは、カスタム (ユーザー設定) ストレージが指定される場合のみ適用されます。
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"	このプロパティーは、カスタム (ユーザー設定) ストレージが指定される場合のみ適用されます。
custom_decimal_symbol	*field*	カスタム (ユーザー設定) ストレージが指定される場合のみ適用されます。

表 50. *fixedfilenode* プロパティー (続き) :

fixedfilenode プロパティー	データ型	プロパティーの説明
encode	StreamDefault SystemDefault "UTF-8"	テキストのエンコード方法を指定します。

gsdata_import ノードのプロパティー

マップ データや地理空間データをデータ マイニング セッションに取り込むには、地理空間入力ノードを使用します。

表 51. *gsdata_import* ノードのプロパティー

gsdata_import ノードのプロパティー	データ型	プロパティーの説明
full_filename	*string*	ロードしたい .shp ファイルのパスを入力します。
map_service_URL	*string*	接続先のマップ サービスの URL を入力します。
map_name	*string*	このプロパティーには、マップ サービスの最上位のフォルダー構造が格納されます (map_service_URL を使用する場合のみ)。

sasimportnode プロパティー

SAS インポート・ノードで、SAS データを IBM SPSS Modeler へインポートします。

例

```
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)
```

表 52. *sasimportnode* プロパティー :

sasimportnode プロパティー	データ型	プロパティーの説明
format	Windows UNIX Transport SAS7 SAS8 SAS9	インポートするファイルの形式。

表 52. *sasimportnode* プロパティー (続き):

sasimportnode プロパティー	データ型	プロパティーの説明
full_filename	*string*	パスも含めた、完全なファイル名。この名前を入力します。
member_name	*string*	指定した SAS トランスポート・ファイルからインポートするメンバーを指定します。
read_formats	*flag*	指定された形式ファイルから、データ形式 (変数ラベルなど) を読み込みます。
full_format_filename	*string*	
import_names	NamesAndLabels LabelsasNames	インポート時に変数名と変数ラベルをマッピングする方法を指定します。

simgennode プロパティー

シミュレーション生成ノードにより、シミュレーション対象のデータを容易に生成することができます。このとき、ユーザー指定の統計分布を使用して最初から生成するか、既存の履歴データに対してシミュレーション適合ノードを実行して得られた分布を使用して自動的に生成することができます。これは、モデルの入力に不確定性がある状況で予測モデルの結果を評価するときに便利です。

表 53. *simgennode* プロパティー:

simgennode プロパティー	データ型	プロパティーの説明
fields	構造化プロパティー	例を参照
correlations	構造化プロパティー	例を参照
keep_min_max_setting	*boolean*	
refit_correlations	*boolean*	
max_cases	*integer*	最小値は 1000、最大値は 2,147,483,647 です。
create_iteration_field	*boolean*	
iteration_field_name	*string*	
replicate_results	*boolean*	
random_seed	*integer*	
parameter_xml	*string*	パラメーター XML を文字列として返します。

fields の例

これは、以下の構文を使用する構造化されたスロット パラメータです。

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

distribution は、分布名の宣言と、それに続く、属性の名前と値のペアを含むリストです。各分布は次のように定義されます。

```
[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"]]], "", ""]])
```

例えば、二項分布の単一フィールドを生成するノードを作成するために、以下のスクリプトを使用する場合があります。

```
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]]], "", ""]])
```

二項分布では、n と prob の 2 つのパラメーターを使用します。二項分布では、最小値と最大値はサポートされず、空文字列として渡されます。

注: distribution を直接設定することはできません。これは、fields プロパティーとともに使用します。

以下の例では、考えられるすべての分布タイプを示します。NegativeBinomialFailures と NegativeBinomialTrial の両方でしきい値が thresh として入力されていることに注意してください。

```
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
])
```

correlations の例

これは、以下の構文を使用する構造化されたスロット パラメータです。

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

相関は、+1 から -1 までの任意の数字です。相関は必要な数だけ指定することができます。指定されていない相関は、すべて 0 に設定されます。不明なフィールドが存在する場合、相関値は相関行列 (または表)

上で設定する必要があり、赤いテキストで表示されます。不明なフィールドが存在する場合、ノードを実行することはできません。

statisticsimportnode プロパティー

IBM SPSS Statistics ファイル・ノードは、同じ形式を使用する IBM SPSS Statistics で使用される *.sav* ファイル形式のデータおよび IBM SPSS Modeler に保存されたキャッシュ・ファイルを読み込みます。

このノードのプロパティーについては、 355 ページの『statisticsimportnode プロパティー』に記載されています。

tm1odataimport ノードのプロパティー

IBM Cognos TM1 入力ノードは、Cognos TM1 データベースからデータをインポートします。

表 *54. tm1odataimport* ノードのプロパティー

tm1odataimport ノードのプロパティー	データ型	プロパティーの説明
admin_host	*string*	REST API のホスト名の URL。
server_name	*string*	admin_host から選択した TM1 サーバーの名前。
credential_type	*inputCredential* または *storedCredential*	資格情報のタイプを示すために使用されます。
input_credential	*list*	credential_type が inputCredential のときは、ドメイン、ユーザー名、およびパスワードを指定します。
stored_credential_name	*string*	credential_type が storedCredential のときは、C&DS サーバーの資格情報の名前を指定します。
selected_view	*["フィールド" "フィールド"]*	選択された TM1 キューブの詳細と、SPSS へのデータのインポートを行うキューブ ビューの名前を含むリストのプロパティー。以下に例を示します。 TM1_import.setPropertyValue("selected_view", ['plan_BudgetPlan', 'Goal Input'])
is_private_view	*flag*	selected_view が専用ビューであるかどうかを指定します。デフォルト値は false です。
selected_columns	*["フィールド"]*	選択した列を指定します。指定できる項目は 1 つのみです。 例: setPropertyValue("selected_columns", ["Measures"])

表 54. *tm1odataimport* ノードのプロパティー (続き)

tm1odataimport ノードのプロパティー	データ型	プロパティーの説明
selected_rows	*["フィールド" "フィールド"]*	選択した行を指定します。 例: setPropertyValue("selected_rows", ["Dimension_1_1", "Dimension_2_1", "Dimension_3_1", "Periods"])

tm1import ノードのプロパティー (廃止)

IBM Cognos TM1 入力ノードは、Cognos TM1 データベースからデータをインポートします。

注: このノードは、Modeler 18.0 で廃止されました。それに置き換わるノードのスクリプト名は *tm1odataimport* です。

表 55. *tm1import* ノードのプロパティー:

tm1import ノードのプロパティー	データ型	プロパティーの説明
pm_host	*string*	**注:** バージョン 16.0 および 17.0 の場合のみ ホスト名。以下に例を示します。 TM1_import.setPropertyValue("pm_host", 'http://9.191.86.82:9510/pmhub/pm')
tm1_connection	*["field","field", ... ,"field"]*	**注:** バージョン 16.0 および 17.0 の場合のみ TM1 サーバーの接続の詳細を含むリストのプロパティー。形式は次のとおりです: ["TM1_Server_Name","tm1_ username","tm1_ password"] 以下に例を示します。 TM1_import.setPropertyValue("tm1_connection", ['Planning Sample', "admin", "apple"])
selected_view	*["フィールド" "フィールド"]*	選択された TM1 キューブの詳細と、SPSS へのデータのインポートを行うキューブ ビューの名前を含むリストのプロパティー。以下に例を示します。 TM1_import.setPropertyValue("selected_view", ['plan_BudgetPlan', 'Goal Input'])
selected_column	*["フィールド"]*	選択した列を指定します。指定できる項目は 1 つのみです。 例: setPropertyValue("selected_columns", ["Measures"])
selected_rows	*["フィールド" "フィールド"]*	選択した行を指定します。 例: setPropertyValue("selected_rows", ["Dimension_1_1", "Dimension_2_1", "Dimension_3_1", "Periods"])

twcimport ノードのプロパティー

TWC ソース・ノードは、IBM ビジネスの 1 つである The Weather Company から気象データをインポートします。これを使用して、ある場所の過去または予報の気象データを取得できます。これにより、使用可能な最も正確で高精度の気象データを利用して、意思決定を向上させるための気象主導のビジネス・ソリューションの開発に役立てることができます。

表 56. *twcimport* ノードのプロパティー

twcimport ノードのプロパティー	データ型	プロパティーの説明
TWCDataImport.latitude	実数	緯度の値を形式 [-90.0■90.0] で指定します。
TWCDataImport.longitude	実数	経度の値を形式 [-180.0■180.0] で指定します。
TWCDataImport.licenseKey	*string*	The Weather Company から入手したライセンス・キーを指定します。
TWCDataImport.measurmentUnit	English Metric Hybrid	測定単位を指定します。指定できる値は、English、Metric、Hybrid です。Metric がデフォルトです。
TWCDataImport.dataType	Historical Forecast	入力する気象データのタイプを指定します。指定できる値は、Historical または Forecast です。Historical がデフォルトです。
TWCDataImport.startDate	*Integer*	TWCDataImport.dataType に Historical を指定した場合は、開始日を yyyyMMdd の形式で指定します。
TWCDataImport.endDate	*Integer*	TWCDataImport.dataType に Historical を指定した場合は、終了日を yyyyMMdd の形式で指定します。
TWCDataImport.forecastHour	6 12 24 48	TWCDataImport.dataType に Forecast を指定した場合は、時間に対して 6、12、24、または 48 を指定します。

userinputnode プロパティー

ユーザー入力ノードを利用すれば、最初から、あるいは既存のデータを変更して、合成データを簡単に作成できます。これは、モデル作成用の検定データセットを作成する場合などに役立ちます。

例

```
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")
```

表 57. *userinputnode* プロパティー：

userinputnode プロパティー	データ型	プロパティーの説明
data		
names		ノードにより生成されたフィールド名のリストを設定または返す構造化スロット。
custom_storage	Unknown String Integer 実数 Time Date Timestamp	フィールドのストレージを設定するか返す、キー・スロット。
data_mode	Combined Ordered	Combined が指定された場合、レコードは、セット値と最小/最大値のそれぞれ組み合わせについて生成されます。生成されたレコード数は、それぞれのフィールドの値の数値の積に等しくなります。Ordered が指定された場合、データ行を生成するために、各レコードの各列から 1 個の値が取られます。生成されるレコード数は、フィールドに関連付けられている最大の値に等しくなります。より小さいデータ値を持つフィールドは、ヌル値で埋められます。
values		**注:** このプロパティーは userinputnode.data に置き換えられたため、使用しないでください。

variablefilenode プロパティー

可変長ノードで、可変長フィールド・テキスト・ファイル、つまりフィールド数は一定でも各フィールド内の文字数が異なるレコードを含むファイルから、データを読み込みます。このノードは、固定長のヘッダー・テキストやある種の注釈があるファイルにも使用できます。

例

```
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])
```

表 *58. variablefilenode* プロパティー：

variablefilenode プロパティー	データ型	プロパティーの説明
skip_header	*number*	最初のレコードの先頭で無視する文字数を指定します。
num_fields_auto	*flag*	各レコードのフィールドの数を自動的に決定します。レコードは、改行文字で終わる必要があります。
num_fields	*number*	各レコードのフィールドの数を手動で指定します。
delimit_space	*flag*	ファイルのフィールドを区切る文字を指定します。
delimit_tab	*flag*	
delimit_new_line	*flag*	
delimit_non_printing	*flag*	
delimit_comma	*flag*	この場合、コンマはストリーム内でフィールドの区切り文字と桁区切り記号の両方であるため、delimit_other を *true* に設定し、other プロパティーを使用し、コンマを区切り記号として指定します。
delimit_other	*flag*	other プロパティーを使用して、カスタム区切り記号をユーザーが指定できます。
other	*string*	delimit_other が *true* に設定されているときに使用される区切り記号を指定します。
decimal_symbol	Default Comma Period	データ・ソースで使われている小数点記号を指定します。
multi_blank	*flag*	複数の隣接するブランク区切り文字を 1 つの区切り文字として扱います。
read_field_names	*flag*	データ・ファイル中の最初の行を列のラベルとして取り扱います。
strip_spaces	None Left Right Both	インポート時に文字列の前後のスペースを破棄します。
invalid_char_mode	Discard Replace	データ入力から不正な文字 (ヌル、0、または現在のエンコード中に存在していない文字) をデータ入力から削除するか (Discard)、指定された 1 文字の記号で不正な文字を置き換えます (Replace)。
invalid_char_replacement	*string*	
break_case_by_newline	*flag*	行区切り文字が改行文字であることを指定します。
lines_to_scan	*number*	指定したデータ型をスキャンする行数を指定します。

表 58. *variablefilenode* プロパティー (続き):

variablefilenode プロパティー	データ型	プロパティーの説明
auto_recognize_datetime	*flag*	入力データの日付または時刻を自動的に特定するかどうかを指定します。
quotes_1	Discard PairAndDiscard IncludeAsText	インポートでの単一引用符の処理方法を指定します。
quotes_2	Discard PairAndDiscard IncludeAsText	インポートでの二重引用符の処理方法を指定します。
full_filename	*string*	読み込みファイルのディレクトリーを含む完全な名前。
use_custom_values	*flag*	
custom_storage	Unknown String Integer 実数 Time Date Timestamp	
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" "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	カスタム (ユーザー設定) ストレージが指定される場合のみ適用されます。

表 58. *variablefilenode* プロパティー (続き):

variablefilenode プロパティー	データ型	プロパティーの説明
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"	カスタム (ユーザー設定) ストレージが指定される場合のみ適用されます。
custom_decimal_symbol	*field*	カスタム (ユーザー設定) ストレージが指定される場合のみ適用されます。
encode	StreamDefault SystemDefault "UTF-8"	テキストのエンコード方法を指定します。

xmlimportnode プロパティー

XML 入力ノードを使用して、XML 形式のデータをストリームにインポートできます。ディレクトリーの 1 つのファイルまたはすべてのファイルをインポートできます。オプションで、XML 構造を読み込むスキーマ ファイルを指定できます。

例

```
node = stream.create("xmlimport", "My node")
node.setPropertyValue("full_filename", "c:/import/ebooks.xml")
node.setPropertyValue("records", "/author/name")
```

表 59. *xmlimportnode* プロパティー:

xmlimportnode プロパティー	データ型	プロパティーの説明
read	single directory	単独のデータ・ファイルを読み込む (デフォルト) か、ディレクトリー内のすべての XML ファイルを読み込みます。
recurse	*flag*	指定したディレクトリーのすべてのサブディレクトリーから XML ファイルを追加で読み込むかどうかを指定します。
full_filename	*string*	(必須) インポートする XML ファイルの完全パスおよびファイル名 (read = single の場合)。

表 59. *xmlimportnode* プロパティー (続き):

xmlimportnode プロパティー	データ型	プロパティーの説明
directory_name	*string*	(必須) XML ファイルをインポートするディレクトリーの完全パスおよび名前 (read = directory の場合)。
full_schema_filename	*string*	XML 構造を読み込む XSD ファイルまたは DTD ファイルの完全パスおよびファイル名。このパラメーターを使用すると、構造を XML 入力ファイルから読み込みます。
records	*string*	レコードの境界を定義する XPath 式 (例: /author/name)。入力ファイルにこの要素が出現するごとに、新しいレコードが作成されます。
mode	read specify	すべてのデータを読み込む (デフォルト) か、読み込む項目を指定します。
fields		インポートする項目 (要素と属性) のリスト。リスト内の各アイテムは XPath 式です。

dataviewimport プロパティー

データ ビュー ノードで、データ ビューのデータを IBM SPSS Modeler にインポートします。

例

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

dvnode = stream.createAt("dataviewimport", "Data View", 96, 96)
dvnode.setPropertyValue("analytic_data_source",
["","/folder/adv", "LATEST"])
dvnode.setPropertyValue("table_name", ["","com.ibm.spss.Table"])
dvnode.setPropertyValue("data_access_plan",
["","DataAccessPlan"])
dvnode.setPropertyValue("optional_attributes",
[["","NewDerivedAttribute"]])
dvnode.setPropertyValue("include_xml", True)
dvnode.setPropertyValue("include_xml_field", "xml_data")
```

表 60. *dataviewimport* プロパティー

dataviewimport プロパティー	データ型	プロパティーの説明
analytic_data_source	*string*	IBM SPSS Collaboration and Deployment Services に保管された分析データ ビュー オブジェクト。パス名と、使用するバージョンのバージョン ラベル。 ["Object ID","Full path", "Version"]

表 60. *dataviewimport* プロパティー (続き)

dataviewimport プロパティー	データ型	プロパティーの説明
table_name	*string*	分析データ ビューで使用されるデータ ビュー テーブル。テーブル名は、パッケージで修飾されている必要があります。IBM SPSS Collaboration and Deployment Services Deployment Manager クライアントから BOM をエクスポートし、エクスポートされた zip アーカイブ内の default.bom ファイルを調べることによって、パッケージを取得できます。パッケージ名は、BOM が IBM Operational Decision Management (iLOG) からインポートされた場合を除き、常に同じでなければなりません。 ["Object ID", "Name"]
data_access_plan	*string*	分析データ ビューにデータを提供するために使用されるデータ アクセス計画。 ["Object ID", "Name"]
optional_attributes	*string*	組み込む作成された属性のリスト。 [["ID1","Name1"], ["ID2", "Name2"]]
include_xml	*boolean*	XOM インスタンス データを持つフィールドを組み込む場合は True。IBM Analytical Decision Management の iLOG ノードが使用される場合を除き、推奨される設定は false です。これをオンにすると、多量の追加的な処理が発生することがあります。
include_xml_field	*string*	include_xml が true に設定された場合に追加するフィールドの名前。

第 10 章 レコード設定ノードのプロパティー

appendnode プロパティー

レコード追加ノードで、レコードのセットを連結します。レコード追加ノードは、構造が似ていながらデータが異なるデータ・セットを組み合せる場合に役立ちます。

例

```
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")
```

表 *61. appendnode* プロパティー：

appendnode プロパティー	データ型	プロパティーの説明
match_by	Position Name	メイン・データ・ソース中のフィールドの位置 (Position) 、または入力データセット中のフィールド名 (Name) を基準にして、データセットを追加できます。
match_case	*flag*	フィールド名を比較するときに大文字と小文字の区別を有効にします。
include_fields_from	Main All	
create_tag_field	*flag*	
tag_field_name	*string*	

aggregatenode プロパティー

レコード集計ノードで、一連の入力レコードを要約集計された出力レコードに置き換えます。

例

```
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")
```

表 62. *aggregatenode* プロパティー：

aggregatenode プロパティー	データ型	プロパティーの説明
keys	*list*	集計にキーとして使用できるフィールドが一覧表示されます。例えば、キー・フィールドが Sex と Region の場合、一意な M と F の、および地域 N と S のそれぞれの組み合わせに対して集計レコードが作成されます (4 つの一意な組み合わせ)。
contiguous	*flag*	同じキー値を持つすべてのレコードが入力にグループ化されている場合 (例えば、入力がキー・フィールドにソートされる場合)、このオプションを選択します。このオプションを選択すると、パフォーマンスが向上します。
aggregates		集計する数値フィールド、および選択されている集計モードを表示する構造化プロパティー。
aggregate_exprs		派生フィールドの名前を、そのフィールドを計算するために使用される集計式と共にキー化するキー プロパティー。以下に例を示します。 `aggregatenode.setKeyedPropertyValue ("aggregate_exprs", "Na_MAX", "MAX('Na')")`
extension	*string*	重複集計フィールドに対応させる接頭辞または接尾辞を指定します (下の例を参照)。
add_as	Suffix Prefix	
inc_record_count	*flag*	各集計レコードを作成するために集計された入力レコード数を指定する追加フィールドを作成します。
count_field	*string*	レコード度数フィールドの名前を指定します。
allow_approximation	*Boolean*	Analytic Server での集計の実行時に順序統計の近似を許可します。
bin_count	*integer*	近似で使用するビン数を指定します。

balancenode プロパティー

バランス・ノードで、データ・セットが指定した条件に合うように、データ・セットの不均衡を修正します。バランス式で、指定した比率によって条件が真 (true) の場合に、レコードの比率を調整します。

例

```
node = stream.create("balance", "My node")
node.setPropertyValue("training_data_only", True)
node.setPropertyValue("directives", [[1.3, "Age > 60"], [1.5, "Na > 0.5"]])
```

表 63. *balancenode* プロパティー:

balancenode プロパティー	データ型	プロパティーの説明
directives		指定された数値に基づいてフィールド値の割合を均衡にするための構造化プロパティー (次の例を参照してください)。
training_data_only	*flag*	学習データのみがバランス化されるよう指定します。データ区分フィールドがストリーム中で指定されていない場合、このオプションは無視されます。

このノードのプロパティーは次の形式を使用します。

[[*number*, 文字列] ¥ [*number*, 文字列] ¥ ... [*number*, 文字列]]

注: 文字列を式に埋め込む場合 (二重引用符を使用)、その先頭にエスケープ文字 " ¥ " を指定する必要があります。" ¥ " 文字は、行継続文字でもあります。これを使用して、引数を見やすく揃えて記述することができます。

derive_stbnode プロパティー

スペース-時間-ボックス・ノードは、緯度、経度、およびタイム・スタンプの各フィールドから、スペース-時間-ボックスを派生させます。頻度の高いスペース-時間-ボックスをハングアウトとして識別することもできます。

例

```
node = modeler.script.stream().createAt("derive_stb", "My node", 96, 96)

#「個々のレコード」モードの場合
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_")

#「ハングアウト」モードの場合
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)
```

表 64. スペース タイム ボックス ノードのプロパティー

derive_stbnode プロパティー	データ型	プロパティーの説明
mode	IndividualRecords Hangouts	
latitude_field	*field*	
longitude_field	*field*	
timestamp_field	*field*	
hangout_density	*density*	単一密度。有効な密度値については、「densities」を参照してください。
densities	[*density,density,..., density*]	各 density は、STB_GH8_1DAY などの文字列です。 **注:** どの density が有効であるかについては、制約があります。geohash の場合、GH1 から GH15 の値を使用できます。この部分では、以下の値を使用できます 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	*field*	
qualifying_duration	1DAY 12HOURS 8HOURS 6HOURS 4HOURS 3HOURS 2Hours 1HOUR 30MIN 15MIN 10MIN 5MIN 2MIN 1MIN 30SECS 15SECS 10SECS 5SECS 2SECS 1SECS	文字列でなければなりません。

表 64. スペース タイム ボックス ノードのプロパティー (続き)

derive_stbnode プロパティー	データ型	プロパティーの説明
min_events	*integer*	最小の有効な整数値は 2 です。
qualifying_pct	*integer*	1 から 100 の範囲でなければなりません。
add_extension_as	Prefix Suffix	
name_extension	*string*	

distinctnode プロパティー

重複レコード・ノードで、重複レコードを削除します。その場合、最初の重複するレコードをデータ・ストリームに渡すか、または、最初のレコードを破棄して、その後の重複レコードをデータ・ストリームに渡します。

例

```
node = stream.create("distinct", "My node")
node.setPropertyValue("mode", "Include")
node.setPropertyValue("fields", ["Age" "Sex"])
node.setPropertyValue("keys_pre_sorted", True)
```

表 65. *distinctnode* プロパティー:

distinctnode プロパティー	データ型	プロパティーの説明
mode	Include Discard	データ・ストリームに最初の重複レコードを含めるか、最初の重複レコードを破棄して、代わりにすべての重複レコードをデータ・ストリームに渡すことができます。
grouping_fields	*list*	レコードが同一であるかどうかを判断するために使われるフィールドを表示します。 **注:** このプロパティーは、IBM SPSS Modeler 16 以降では廃止されています。
composite_value	構造化スロット	下の例を参照してください。
composite_values	構造化スロット	下の例を参照してください。
inc_record_count	*flag*	各集計レコードを作成するために集計された入力レコード数を指定する追加フィールドを作成します。
count_field	*string*	レコード度数フィールドの名前を指定します。
sort_keys	構造化スロット。	**注:** このプロパティーは、IBM SPSS Modeler 16 以降では廃止されています。
default_ascending	*flag*	
low_distinct_key_count	*flag*	キー・フィールドに少ないレコードまたは少ない一意の値を持つよう指定します。
keys_pre_sorted	*flag*	同じキー値を持つすべてのレコードが入力で一緒にグループ化されるよう指定します。
disable_sql_generation	*flag*	

composite_value プロパティーの例

composite_value プロパティーは、以下の一般形式になっています。

```
node.setKeyedPropertyValue("composite_value", FIELD, FILLOPTION)
```

FILLOPTION は [FillType, Option1, Option2, ...] という形式になっています。

例:

```
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"])
```

カスタム オプションでは、複数の引数が必要であり、それらはリストとして追加されます。例えば、以下のようになります。

```
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"])
```

composite_values プロパティーの例

composite_values プロパティーは、以下の一般形式になっています。

```
node.setPropertyValue("composite_values", [
                        [FIELD1, [FILLOPTION1]],
                        [FIELD2, [FILLOPTION2]],
.
.
])
```

例:

```
node.setPropertyValue("composite_values", [
    ["Age", ["First"]],
    ["Name", ["MostFrequent", "First"]],
    ["Pending", ["IncludesValue", "T"]],
    ["Marital", ["FirstMatch", "Married", "Divorced", "Separated"]],
    ["Code", ["Concatenate", "Comma"]]
])
```

extensionprocessnode プロパティー

拡張変換ノードを使用すると、R スクリプトまたは Python for Spark スクリプトを使用して、ストリームからデータを取得し、取得したデータに変換を適用できます。

Python for Spark の例

```
#### 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)
```

R の例

```
#### 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)""")
```

表 66. *extensionprocessnode* プロパティー

extensionprocessnode プロパティー	データ型	プロパティーの説明
syntax_type	*R* *Python*	R または Python のどちらのスクリプトを実行するか指定します (R がデフォルトです)。
r_syntax	*string*	実行する R スクリプト・シンタックス。
python_syntax	*string*	実行する Python スクリプト・シンタックス。

表 66. *extensionprocessnode* プロパティー (続き)

extensionprocessnode プロパティー	データ型	プロパティーの説明
use_batch_size	*flag*	バッチ処理を使用可能にします。
batch_size	*integer*	各バッチに含めるデータ レコードの数を指定します。
convert_flags	StringsAndDoubles LogicalValues	フラグ型フィールドを変換するためのオプション。
convert_missing	*flag*	欠損値を R の NA 値に変換するためのオプション。
convert_datetime	*flag*	日付形式または日付/時刻形式の変数を R の日付/時刻形式に変換するためのオプション。
convert_datetime_class	POSIXct POSIXlt	日付形式または日付/時刻形式の変数のうち、どの形式の変数を変換するかを指定するためのオプション。

mergenode プロパティー

レコード結合ノードは、複数の入力レコードを取得し、入力フィールドの全部または一部を含む 1 つの出力レコードを作成します。この機能は、内部顧客データと購入人口データのような、異なるソースからのデータを結合する場合に役立ちます。

例

```
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"]])
```

表 67. *mergenode* プロパティー：

mergenode プロパティー	データ型	プロパティーの説明
method	Order Keys Condition Rankedcondition	データ ファイルでのリスト順にレコードを結合するかどうか、1 つ以上のキー フィールドを使用してキー フィールド内の同じ値にレコードを結合するかどうか、指定された条件を満たす場合にレコードを結合するかどうか、1 次データセットとすべての 2 次データセット内の各行のペアを結合するかどうかを指定します。いずれの場合も、ランク付け式を使用して、ランクの低い一致からランクの高い一致の順にすべての一致がソートされます。
condition	*string*	method が Condition に設定されている場合、レコードを含めるまたは破棄する条件を指定します。
key_fields	*list*	
common_keys	*flag*	
join	Inner FullOuter PartialOuter Anti	
outer_join_tag.n	*flag*	このプロパティーでは、*n* は「データセットの選択」ダイアログ・ボックスに表示されるタグ名です。どのようなデータセット数であっても不完全なレコードを作成する可能性があるので、複数のタグ名を指定できます。
single_large_input	*flag*	ほかの入力と比べて比較的大きな入力を指定し最適化を行うかどうかを指定します。
single_large_input_tag	*string*	「ラージ・データセットの選択」ダイアログ・ボックスに表示されるタグ名を指定します。このプロパティーの用途は、1 つの入力データセットしか指定できないという点で、outer_join_tag プロパティーとは若干異なることに注意してください (データ型がフラグと文字列という違いもあり)。
use_existing_sort_keys	*flag*	入力がすでにキー・フィールドでソート済みかどうかを指定します。
existing_sort_keys	[['文字列', 'Ascending'] ¥ ['文字列'', 'Descending']]	すでにソートされたフィールドとソート方向を指定します。
primary_dataset	*string*	method が Rankedcondition の場合は、結合内の 1 次データセットを選択します。これは、外部結合の左側と考えることができます
rename_duplicate_fields	*Boolean*	method が Rankedcondition の場合にこのプロパティーを Y に設定し、異なるデータ ソースから取得された同じ名前を持つ複数のフィールドが結果の結合データセットに含まれている場合、それらのデータ ソースの各タグがフィールドの列見出しの先頭に追加されます。

表 67. *mergenode* プロパティー (続き):

mergenode プロパティー	データ型	プロパティーの説明
merge_condition	*string*	
ranking_expression	*string*	
Num_matches	*integer*	merge_condition と ranking_expression に基づいて返される一致の数。最小値は 1、最大値は 100 です。

rfmaggregatenode プロパティー

リーセンシ、フリクエンシ、マネタリー (RFM) のレコード集計ノードを使用すると、顧客の過去のトランザクション・データを取得、未使用のデータを削除、残りのトランザクション・データをすべて単一行に結合することができます。これにより、最後のトランザクションの時期、トランザクション数、これらのトランザクションの合計金額が一覧表示されます。

例

```
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")
```

表 68. *rfmaggregatenode* プロパティー:

rfmaggregatenode プロパティー	データ型	プロパティーの説明
relative_to	Fixed Today	トランザクションのリーセンシが計算される日付を指定します。
reference_date	*date*	Fixed が relative_to に設定されている場合にのみ使用できます。
contiguous	*flag*	データ・ストリーム中で同じ ID を持つすべてのレコードが一緒に表示されるようにデータをソートしている場合、このオプションを選択すると処理を高速化することができます。
id_field	*field*	顧客およびトランザクションを識別するために使用するフィールドを指定します。
date_field	*field*	リーセンシを計算するために使用される日付フィールドを選択します。
value_field	*field*	マネタリー値を計算するために使用するフィールドを指定します。
extension	*string*	重複集計フィールドに対応させる接頭辞または接尾辞を指定します。
add_as	Suffix Prefix	extension を接尾辞として追加するか、または接頭辞として追加するかを指定します。

表 68. *rfmaggregatenode* プロパティー (続き):

rfmaggregatenode プロパティー	データ型	プロパティーの説明
discard_low_value_records	*flag*	discard_records_below 設定の使用を有効にします。
discard_records_below	*number*	RFM の合計を計算する場合に使用されないトランザクションの詳細の最小値を指定することができます。値の単位は、選択された「value」フィールドに関連します。
only_recent_transactions	*flag*	specify_transaction_date または transaction_within_last 設定の使用を有効にします。
specify_transaction_date	*flag*	
transaction_date_after	*date*	specify_transaction_date が選択されている場合にのみ使用できます。データが分析に含まれた後のトランザクションの日付を指定します。
transaction_within_last	*number*	transaction_within_last が選択されている場合にのみ使用できます。レコードが分析に含まれる後の「リーセンシ基準日」の日付からさかのぼった期間の数および種類 (日、週、月または年数) を指定します。
transaction_scale	Days Weeks Months Years	transaction_within_last が選択されている場合にのみ使用できます。レコードが分析に含まれる後の「リーセンシ基準日」の日付からさかのぼった期間の数および種類 (日、週、月または年数) を指定します。
save_r2	*flag*	各顧客の 2 番目に最近のトランザクションの日付を表示します。
save_r3	*flag*	save_r2 が選択されている場合にのみ使用できます。各顧客の 3 番目に最近のトランザクションの日付を表示します。

Rprocessnode プロパティー

R 変換ノードでは、IBM(r) SPSS(r) Modeler ストリームからデータを取得し、そのデータを独自のカスタム R スクリプトを使用して変更できます。データ変更後、データはストリームに返されます。

例

```
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",fieldMeasure="",fieldFormat="",
fieldRole="")
modelerDataModel<-data.frame(modelerDataModel,var1)""")
node.setPropertyValue("convert_datetime", "POSIXct")
```

表 69. *Rprocessnode* プロパティー：

Rprocessnode プロパティー	データ型	プロパティーの説明
syntax	*string*	
convert_flags	StringsAndDoubles LogicalValues	
convert_datetime	*flag*	
convert_datetime_class	POSIXct POSIXlt	
convert_missing	*flag*	
use_batch_size	*flag*	バッチ処理を使用可能にします
batch_size	*integer*	各バッチに含めるデータ レコードの数を指定します

samplenode プロパティー

サンプル・ノードでは、レコードのサブセットを選択します。層化サンプル、クラスター・サンプル、非無作為 (構造化) サンプルなど、さまざまなサンプルの種類がサポートされています。サンプリングは、パフォーマンスの向上、および分析のための関連するレコードまたはトランザクションのグループの選択に役に立ちます。

例

```
/* 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]])
```

表 70. *samplenode* プロパティー：

samplenode プロパティー	データ型	プロパティーの説明
method	Simple Complex	
mode	Include Discard	指定された条件を満たすレコードを含めるか (Include)、破棄 (Discard) します。
sample_type	First OneInN RandomPct	サンプリング方法を指定します。

表 70. *samplenode* プロパティー (続き) :

samplenode プロパティー	データ型	プロパティーの説明
first_n	*integer*	指定された分割点までのレコードを含めるか破棄します。
one_in_n	*number*	*n* 番目ごとにレコードを含めるか破棄します。
rand_pct	*number*	含めるか破棄するレコードのパーセンテージを指定します。
use_max_size	*flag*	maximum_size 設定の使用を有効にします。
maximum_size	*integer*	データ・ストリームに入れるまたはデータ・ストリームから破棄するサンプルの最大数を指定します。このオプションは冗長であり、そのため、First と Include が指定されているときは破棄されます。
set_random_seed	*flag*	ランダム・シード設定の使用を有効にします。
random_seed	*integer*	ランダム・シードとして使用する値を指定します。
complex_sample_type	Random Systematic	
sample_units	Proportions Counts	
sample_size_proportions	Fixed Custom Variable	
sample_size_counts	Fixed Custom Variable	
fixed_proportions	*number*	
fixed_counts	*integer*	
variable_proportions	*field*	
variable_counts	*field*	
use_min_stratum_size	*flag*	
minimum_stratum_size	*integer*	このオプションは、Sample units=Proportions によって複雑なサンプルが作成された場合にのみ適用されます。
use_max_stratum_size	*flag*	
maximum_stratum_size	*integer*	このオプションは、Sample units=Proportions によって複雑なサンプルが作成された場合にのみ適用されます。
clusters	*field*	
stratify_by	*[field1 ... fieldN]*	
specify_input_weight	*flag*	
input_weight	*field*	
new_output_weight	*string*	

表 70. *samplenode* プロパティー (続き):

samplenode プロパティー	データ型	プロパティーの説明
sizes_proportions	[[string *string value*][string *string value*]...]	sample_units=proportions および sample_size_proportions=Custom の場合、層化フィールドの値の考えられる組み合わせの値を指定します。
default_proportion	*number*	
sizes_counts	[[string *string value*][string *string value*]...]	層化フィールドの値の考えられる組み合わせの値を指定します。使用方法は sizes_proportions と似ていますが、割合ではなく整数を指定します。
default_count	*number*	

selectnode プロパティー

条件抽出ノードで、特定の条件に基づいて、データ・ストリームからレコードのサブセットを選択したり破棄したりできます。例えば、特定の営業地域に関連するレコードを選択できます。

例

```
node = stream.create("select", "My node")
node.setPropertyValue("mode", "Include")
node.setPropertyValue("condition", "Age < 18")
```

表 71. *selectnode* プロパティー:

selectnode プロパティー	データ型	プロパティーの説明
mode	Include Discard	選択したレコードを含めるか、または破棄するかを指定します。
condition	*string*	レコードを含めるか、または破棄かの条件。

sortnode プロパティー

ソート・ノードで、1 つまたは複数のフィールド値に基づいて、レコードを昇順または降順にソートします。

例

```
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"]])
```

表 72. *sortnode* プロパティー：

sortnode プロパティー	データ型	プロパティーの説明
keys	*list*	ソートの基準となるフィールドを指定します。ソートの方向が指定されていない場合、デフォルトが使用されます。
default_ascending	*flag*	デフォルトのソート順を指定します。
use_existing_keys	*flag*	前に使用されたフィールドのソート順を使用してソートを最適化するかどうかを指定します。
existing_keys		すでにソートされたフィールドとソート方向を指定します。keys プロパティーと同じ形式を使用します。

spacetimeboxes プロパティー

Space-Time-Box (STB) は、Geohash の空間的な場所を拡張したものです。具体的には、STB は英数字の文字列で、空間および時間を規則的に分割した領域です。

表 73. *spacetimeboxes* プロパティー

spacetimeboxes プロパティー	データ型	プロパティーの説明
mode	*IndividualRecords* *Hangouts*	
latitude_field	*field*	
longitude_field	*field*	
timestamp_field	*field*	

表 73. *spacetimeboxes* プロパティー (続き)

spacetimeboxes プロパティー	データ型	プロパティーの説明
densities	*[density, density, density...]*	各 density は、文字列です。例: STB_GH8_1DAY densities が有効であるための制限が存在することに注意してください。 geohash の場合、GH1-GH15 の値を使用できます。 この部分では、以下の値を使用できます 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	*Prefix* *Suffix*	
hangout_density	*density*	単一密度 (上記を参照)
id_field	*field*	
qualifying_duration	1DAY 12HOURS 8HOURS 6HOURS 4HOURS 2HOURS 1HOUR 30MIN 15MIN 10MIN 5MIN 2MIN 1MIN 30SECS 15SECS 10SECS 5SECS 2SECS 1SECS	これは、文字列にする必要があります。
min_events	*integer*	最小値は 2 です。

表 73. *spacetimeboxes* プロパティー (続き)

spacetimeboxes プロパティー	データ型	プロパティーの説明
qualifying_pct	*integer*	1 から 100 の範囲にする必要があります

streamingtimeseries プロパティー

ストリーミング時系列ノードは、1 つのステップで時系列モデルを作成してスコアリングします。
注: このストリーミング時系列ノードは、SPSS Modeler バージョン 18 で廃止されたオリジナルのストリーミング TS に置き換わるものです。

表 74. *streamingtimeseries* プロパティー

streamingtimeseries プロパティー	値	プロパティーの説明
targets	*field*	ストリーミング時系列ノードは、オプションで 1 つ以上の入力フィールドを予測値として使用し、1 つ以上の対象フィールドを予測します。度数フィールドおよび重みフィールドは使用しません。 詳しくは、トピック 187 ページの『一般的なモデル作成ノードのプロパティー』を参照してください。
candidate_inputs	[*field1 ... fieldN*]	モデルで使用される入力または予測変数フィールド。
use_period	*flag*	
date_time_field	*field*	
input_interval	None Unknown Year Quarter Month Week Day Hour Hour_nonperiod Minute Minute_nonperiod Second Second_nonperiod	
period_field	*field*	
period_start_value	*integer*	
num_days_per_week	*integer*	

表 74. *streamingtimeseries* プロパティー (続き)

streamingtimeseries プロパティー	値	プロパティーの説明
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	*list*	
output_interval	None Year Quarter Month Week Day Hour Minute Second	
is_same_interval	*flag*	
cross_hour	*flag*	
aggregate_and_distribute	*list*	
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	

表 74. *streamingtimeseries* プロパティー (続き)

streamingtimeseries プロパティー	値	プロパティーの説明
date_estimation	*list*	date_time_field を使用する場合にのみ使用可能です
period_estimation	*list*	use_period を使用する場合にのみ使用可能です
observations_type	Latest Earliest	
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*	
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*	

表 74. *streamingtimeseries* プロパティー (続き)

streamingtimeseries プロパティー	値	プロパティーの説明
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. *fieldname*	*integer*	伝達関数用。
tf_arima.d. *fieldname*	*integer*	伝達関数用。
tf_arima.q. *fieldname*	*integer*	伝達関数用。
tf_arima.sp. *fieldname*	*integer*	伝達関数用。
tf_arima.sd. *fieldname*	*integer*	伝達関数用。
tf_arima.sq. *fieldname*	*integer*	伝達関数用。
tf_arima.delay. *fieldname*	*integer*	伝達関数用。
tf_arima.transformation_type. *fieldname*	None SquareRoot NaturalLog	伝達関数用。
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	*real*	
events	*field*	
forecastperiods	*integer*	
extend_records_into_future	*flag*	
conf_limits	*flag*	
noise_res	*flag*	

streamingts プロパティー (廃止)

注: この元のストリーミング時系列ノードは SPSS Modeler のバージョン 18 で廃止され、新しいストリーミング時系列ノードに置き換えられました。この新しいノードは、IBM SPSS Analytic Server の機能を活用し、ビッグデータを処理するように設計されています。ストリーミング TS ノードは、1 つのステップで時系列モデルを作成してスコアリングします。時間間隔ノードは必要ありません。

例

```
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")
```

表 *75. streamingts* プロパティー：

streamingts プロパティー	データ型	プロパティーの説明
custom_fields	*flag*	custom_fields=false の場合は、上流のデータ型ノードの設定が使用されます。 custom_fields=true の場合は、targets と inputs を指定する必要があります。
targets	[*field 1...*フィールド *N*]	
inputs	[*field 1...*フィールド *N*]	
method	ExpertModeler Exsmooth Arima	
calculate_conf	*flag*	
conf_limit_pct	*real*	
use_time_intervals_node	*flag*	use_time_intervals_node=true の場合は、上流の時間間隔ノードの設定が使用されます。 use_time_intervals_node=false の場合は、interval_offset_position、interval_offset、および interval_type を指定する必要があります。
interval_offset_position	LastObservation LastRecord	LastObservation は、「最新の有効な観測値」を表します。LastRecord は、「最後のレコードから遡り設定」を表します。
interval_offset	*number*	
interval_type	Periods Years Quarters Months WeeksNonPeriodic DaysNonPeriodic HoursNonPeriodic MinutesNonPeriodic SecondsNonPeriodic	
events	*field*	
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*	

表 75. *streamingts* プロパティー (続き):

streamingts プロパティー	データ型	プロパティーの説明
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*	時系列モデル作成ノードの場合と同じプロパティー
arima_d	*integer*	時系列モデル作成ノードの場合と同じプロパティー
arima_q	*integer*	時系列モデル作成ノードの場合と同じプロパティー
arima_sp	*integer*	時系列モデル作成ノードの場合と同じプロパティー
arima_sd	*integer*	時系列モデル作成ノードの場合と同じプロパティー
arima_sq	*integer*	時系列モデル作成ノードの場合と同じプロパティー
arima_transformation_type	None SquareRoot NaturalLog	時系列モデル作成ノードの場合と同じプロパティー
arima_include_constant	*flag*	時系列モデル作成ノードの場合と同じプロパティー
tf_arima_p.*fieldname*	*integer*	時系列モデル作成ノードの場合と同じプロパティー。伝達関数用。
tf_arima_d.*fieldname*	*integer*	時系列モデル作成ノードの場合と同じプロパティー。伝達関数用。
tf_arima_q.*fieldname*	*integer*	時系列モデル作成ノードの場合と同じプロパティー。伝達関数用。
tf_arima_sp.*fieldname*	*integer*	時系列モデル作成ノードの場合と同じプロパティー。伝達関数用。
tf_arima_sd.*fieldname*	*integer*	時系列モデル作成ノードの場合と同じプロパティー。伝達関数用。
tf_arima_sq.*fieldname*	*integer*	時系列モデル作成ノードの場合と同じプロパティー。伝達関数用。
tf_arima_delay.*fieldname*	*integer*	時系列モデル作成ノードの場合と同じプロパティー。伝達関数用。

表 75. *streamingts* プロパティー (続き):

streamingts プロパティー	データ型	プロパティーの説明
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*	
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	<フィールド>	

cplexnode のプロパティー

CPLEX の最適化ノードにより、OPL (Optimization Programming Language) モデル・ファイルを介した複雑な数学 (CPLEX) ベースの最適化の機能が提供されます。この機能は IBM Analytical Decision Management 製品で使用可能ですが、IBM Analytical Decision Management の必要なしに、SPSS Modeler でも CPLEX ノードを使用できるようになりました。

CPLEX の最適化および OPL について詳しくは、IBM Analytical Decision Management の資料を参照してください。

表 76. *cplexnode* のプロパティー

cplexnode のプロパティー	データ型	プロパティーの説明
opl_model_text	*string*	CPLEX の最適化ノードによって実行され、最適化の結果を生成する OPL (Optimization Programming Language) スクリプト・プログラム。
opl_tuple_set_name	*string*	入力データに対応する、OPL モデルのタプル・セット名。
opl_data_text	*string*	OPL に使用する変数、つまりデータの定義。
output_value_mode	*string*	指定できる値は、raw または dvar です。dvar を指定した場合、ユーザーは「出力」タブで OPL の目的関数の変数名を出力用に指定する必要があります。raw を指定した場合、名前に関係なく、目的関数が直接出力されます。

表 *76. cplexnode* のプロパティー (続き)

cplexnode のプロパティー	データ型	プロパティーの説明
objective_function_value_fieldname	*string*	出力に使用するフィールド名。デフォルトは _OBJECTIVE です。

第 11 章 フィールド設定ノードのプロパティー

anonymizenode プロパティー

匿名化ノードは、フィールド名や値の下流の表示方法を変換し、元のデータを隠します。これは、他のユーザーが顧客名やその他の詳細情報をなどの重要情報を使用してモデルを構築できるようにする場合に有用です。

例

```
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")
```

表 77. *anonymizenode* プロパティー

anonymizenode プロパティー	データ型	プロパティーの説明
enable_anonymize	*flag*	これを True に設定すると、フィールド値の匿名化がアクティブになります (そのフィールドの「匿名値」列で「はい」を選択した場合も、同じ結果になります)。
use_prefix	*flag*	これを True に設定すると、ユーザー指定の接頭辞が使用されます (ユーザー指定の接頭辞が指定されている場合)。ハッシュ・メソッドによって匿名化されるフィールドに適用され、そのフィールドの「値を置換」ダイアログの「ユーザー設定」ラジオ・ボタンを選択することと同等です。
prefix	*string*	「値を置換」ダイアログ・ボックスのテキスト・ボックスに接頭辞を入力することと同等です。デフォルトの接頭辞は、何も他に指定されていない場合は、デフォルト値です。
transformation	Random Fixed	Transform メソッドにより匿名化されたフィールドの変換パラメーターが無作為 (Random) か固定 (Fixed) かを決定します。
set_random_seed	*flag*	これを True に設定すると、指定されたシード値が使用されます (transformation も Random に設定されている場合)。
random_seed	*integer*	set_random_seed が True に設定されている場合、このプロパティーは乱数のシードになります。
scale	*number*	transformation が Fixed に設定されている場合、この値はスケール用として使用されます。最大スケール値は通常 10 ですが、あふれを防止するために減少できます。

表 77. *anonymizenode* プロパティー (続き)

anonymizenode プロパティー	データ型	プロパティーの説明
translate	*number*	transformation が Fixed に設定されている場合、この値は変換用として使用されます。最大変換値は通常 1000 ですが、あふれを防止するために減少できます。

autodataprepnode プロパティー

自動データ準備 (ADP) ノードでは、データ分析、固定値の識別、問題のあるまたは役に立たない可能性のあるフィールドのスクリーニング、必要に応じた新しい属性の取得、詳細なスクリーニングおよびサンプリング手法を使用したパフォーマンスの向上などを行うことができます。完全に自動化された方法でノードを使用し、ノードで固定値を選択および適用できます。または必要に応じて変更の作成および承認、拒否または修正の前に変更をプレビューできます。

例

```
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")
```

表 78. *autodataprepnode* プロパティー

autodataprepnode プロパティー	データ型	プロパティーの説明
objective	Balanced Speed Accuracy Custom	
custom_fields	*flag*	真 (true) の場合は、現在のノードのターゲット、入力、その他フィールドなどを指定することができます。偽 (false) の場合は、上流のデータ型ノードから現在の設定が使用されます。
target	*field*	1 つの対象フィールドを指定します。
inputs	[*field1 ... fieldN*]	モデルで使用される入力または予測変数フィールド。
use_frequency	*flag*	
frequency_field	*field*	
use_weight	*flag*	
weight_field	*field*	
excluded_fields	Filter None	
if_fields_do_not_match	StopExecution ClearAnalysis	
prepare_dates_and_times	*flag*	すべての日付/時間フィールドへのアクセスを制御します。

表 78. *autodataprepnode* プロパティー (続き)

autodataprepnode プロパティー	データ型	プロパティーの説明
compute_time_until_date	*flag*	
reference_date	Today Fixed	
fixed_date	*date*	
units_for_date_durations	Automatic Fixed	
fixed_date_units	Years Months Days	
compute_time_until_time	*flag*	
reference_time	CurrentTime Fixed	
fixed_time	*time*	
units_for_time_durations	Automatic Fixed	
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_inputs	*flag*	
exclude_too_many_missing	*flag*	
maximum_percentage_missing	*number*	
exclude_too_many_categories	*flag*	
maximum_number_categories	*number*	
exclude_if_large_category	*flag*	
maximum_percentage_category	*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_continuous_inputs	*flag*	
replace_missing_continuous_target	*flag*	
replace_missing_nominal_inputs	*flag*	

表 78. *autodataprepnode* プロパティー (続き)

autodataprepnode プロパティー	データ型	プロパティーの説明
replace_missing_nominal_target	*flag*	
replace_missing_ordinal_inputs	*flag*	
replace_missing_ordinal_target	*flag*	
maximum_values_for_ordinal	*number*	
minimum_values_for_continuous	*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_input_fields	*flag*	
maximize_association_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_construction	*flag*	
transformed_target_name_extension	*string*	
transformed_inputs_name_extension	*string*	
constructed_features_root_name	*string*	
years_duration_ name_extension	*string*	
months_duration_ name_extension	*string*	
days_duration_ name_extension	*string*	
hours_duration_ name_extension	*string*	
minutes_duration_ name_extension	*string*	
seconds_duration_ name_extension	*string*	
year_cyclical_name_extension	*string*	
month_cyclical_name_extension	*string*	

表 78. *autodataprepnode* プロパティー (続き)

autodataprepnode プロパティー	データ型	プロパティーの説明
day_cyclical_name_extension	*string*	
hour_cyclical_name_extension	*string*	
minute_cyclical_name_extension	*string*	
second_cyclical_name_extension	*string*	

astimeintervalsnode プロパティー

間隔を指定し、新しい時間フィールドを作成して推定や予測を行う場合は、時間区分ノードを使用します。秒単位から年単位まで、すべての時間区分がサポートされます。

表 79. *astimeintervalsnode* プロパティー

astimeintervalsnode プロパティー	データ型	プロパティーの説明
time_field	*field*	1 つの連続型フィールドのみ許可されます。ノードは、このフィールドを集計キーとして使用して、間隔を変換します。ここで整数フィールドを使用すると、そのフィールドは時間インデックスとして認識されます。
dimensions	*[field1 field2 ... fieldn]*	これらのフィールドを使用して、各フィールドの値に基づき、個々の時系列が作成されます。
fields_to_aggregate	*[field1 field2 ... fieldn]*	これらのフィールドは、時間フィールドの期間変更処理の一部として集計されます。このピッカーに含まれていないすべてのフィールドが、ノードから送信されるデータから除外されます。

binningnode プロパティー

データ分割ノードで、既存の 1 つまたは複数の連続型 (数値範囲) フィールドの値に基づいて、自動的に新しい名義型 (セット型) フィールドを作成します。例えば、連続型収入フィールドを、平均からの偏差による収入グループを含む、新しいカテゴリー・フィールドに変換することができます。新規フィールドのビンを作成すると、分割点に基づいてフィールド作成ノードを生成することができます。

例

```
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)
```

表 *80. binningnode* プロパティー

binningnode プロパティー	データ型	プロパティーの説明
fields	*[field1 field2 ... fieldn]*	変換保留中の連続型 (数値範囲) フィールド。複数のフィールドを同時にビンに分割できます。
method	FixedWidth EqualCount Rank SDev Optimal	新規フィールドのビン (カテゴリー) の分割点を決める方法。
rcalculate_bins	Always IfNecessary	ノードが実行されるごとに、ビンが再計算され、適切なビンの中にデータが配置されるか、またはデータが既存のビンおよび追加された新規のビンに追加されるだけかを指定します。
fixed_width_name_extension	*string*	デフォルトの拡張子は *_BIN* です。
fixed_width_add_as	Suffix Prefix	拡張子をフィールド名の最後に追加するか (Suffix)、または先頭に追加するか (Prefix) を指定します。デフォルトの拡張子は *income_BIN* です。
fixed_bin_method	Width Count	
fixed_bin_count	*integer*	新規フィールドの固定幅ビン (カテゴリー) 数を決定するのに使用する整数を指定します。
fixed_bin_width	*real*	ビンの幅を算出するために使用する値 (整数または実数)。
equal_count_name_ extension	*string*	デフォルトの拡張子は *_TILE* です。
equal_count_add_as	Suffix Prefix	標準の分位を使用して生成されるフィールドに対して使用される拡張子が、Suffix (接頭辞) か Prefix (接尾辞) かを指定します。デフォルトの拡張子は、*_TILE* に *N* を付けたものになります。*N* は分位数です。
tile4	*flag*	それぞれが 25 % のケースを含む、4 分位のビンを生成します。
tile5	*flag*	5 つの 5 分位ビンを生成します。
tile10	*flag*	10 個の十分位 (デシル) ビンを生成します。
tile20	*flag*	20 個の二十分位ビンを生成します。
tile100	*flag*	100 個の百分位 (パーセンタイル) ビンを生成します。
use_custom_tile	*flag*	
custom_tile_name_extension	*string*	デフォルトの拡張子は *_TILEN* です。
custom_tile_add_as	Suffix Prefix	

表 *80. binningnode* プロパティー *(*続き*)*

binningnode プロパティー	データ型	プロパティーの説明
custom_tile	*integer*	
equal_count_method	RecordCount ValueSum	RecordCount の方法は、同じ数のレコードを各ビンに割り当てます。一方、ValueSum では、各ビンの値の合計が同じになるようにレコードを割り当てます。
tied_values_method	Next Current Random	可否同数の値のデータに配置されるビンを指定。
rank_order	Ascending Descending	このプロパティーには、Ascending (もっとも小さい値が 1 となる) または Descending (もっとも大きい値が 1 となる) が含まれます。
rank_add_as	Suffix Prefix	このオプションは、ランク、ランクの比率、およびランクのパーセンテージに適用されます。
rank	*flag*	
rank_name_extension	*string*	デフォルトの拡張子は *_RANK* です。
rank_fractional	*flag*	新規フィールドの値が、ランクを非欠損ケースの重みの合計で除算した値になるように、ケースをランク付けします。ランクの比率は 0 - 1 の範囲の値になります。
rank_fractional_name_ extension	*string*	デフォルトの拡張子は *_F_RANK* です。
rank_pct	*flag*	各ランクが、有効な値を持つレコード数で除算された後、100 倍されます。ランクのパーセンテージは、1 - 100 の範囲の値になります。
rank_pct_name_extension	*string*	デフォルトの拡張子は *_P_RANK* です。
sdev_name_extension	*string*	
sdev_add_as	Suffix Prefix	
sdev_count	One Two Three	
optimal_name_extension	*string*	デフォルトの拡張子は *_OPTIMAL* です。
optimal_add_as	Suffix Prefix	
optimal_supervisor_field	*field*	データ分割のために選択されたフィールドが関係する監督フィールドとして選ばれたフィールド。
optimal_merge_bins	*flag*	ケース度数が小さいビンをより大きな隣接ビンに追加することを指定します。
optimal_small_bin_threshold	*integer*	
optimal_pre_bin	*flag*	データセットの事前データ分割を実行することを示します。
optimal_max_bins	*integer*	過度に多数のビンを作成しないように、上限を指定します。

表 80. *binningnode* プロパティー (続き)

binningnode プロパティー	データ型	プロパティーの説明
optimal_lower_end_point	Inclusive Exclusive	
optimal_first_bin	Unbounded Bounded	
optimal_last_bin	Unbounded Bounded	

derivenode プロパティー

フィールド作成ノードで、1 つまたは複数の既存フィールドから、データ値を変更するか、新しいフィールドを作成します。これで、 タイプ式、フラグ、名義、ステート、カウント、および条件式の各フィールドが作成されます。

例 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¥"")
```

例 2

このスクリプトは、特定のポイント (特定のイベントが発生した場所など) の X 座標と Y 座標を表す XPos と YPos という 2 つの数値列があることを前提としています。このスクリプトにより、特定の座標系でその点を表す X 座標と Y 座標から地理空間列を計算するフィールド作成ノードが作成されます。

```
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")
```

表 81. *derivenode* プロパティー

derivenode プロパティー	データ型	プロパティーの説明
new_name	*string*	新しいフィールド名。
mode	Single Multiple	1 つのフィールドか (Single)、または複数フィールドか (Multiple) を指定します。
fields	*list*	複数フィールドを選択する場合にだけ、Multiple モードで使用。
name_extension	*string*	新しいフィールド名に使用する拡張子を指定します。
add_as	Suffix Prefix	拡張子をフィールド名の Prefix (先頭、接頭辞)、または Suffix (最後、接尾辞) として追加します。
result_type	Formula Flag Set State Count Conditional	作成可能な新しいフィールドの 6 つの種類。
formula_expr	*string*	フィールド作成ノードの新しいフィールド値を計算する式。
flag_expr	*string*	
flag_true	*string*	
flag_false	*string*	
set_default	*string*	
set_value_cond	*string*	特定の値に関連付けられた条件を提供するように構造化プロパティー。
state_on_val	*string*	オン (On) の条件を満たす場合の新規フィールドの値を指定します。
state_off_val	*string*	オフ (Off) の条件を満たす場合の新規フィールドの値を指定します。
state_on_expression	*string*	
state_off_expression	*string*	
state_initial	On Off	各レコードで新しいフィールドの初期値として On または Off を割り当てます。この値は、それぞれの条件が満たされるごとに変化します。
count_initial_val	*string*	
count_inc_condition	*string*	
count_inc_expression	*string*	
count_reset_condition	*string*	
cond_if_cond	*string*	
cond_then_expr	*string*	
cond_else_expr	*string*	

表 81. *derivenode* プロパティー (続き)

derivenode プロパティー	データ型	プロパティーの説明
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	このプロパティーを使用して、作成されたフィールドに関連付けられた尺度を定義することができます。setter 関数には、文字列か、MeasureType の値のいずれかを渡すことができます。getter は、常に MeasureType の値を返します。
collection_measure	Range / MeasureType.RANGE Flag / MeasureType.FLAG Set / MeasureType.SET OrderedSet / MeasureType.ORDERED_SET Typeless / MeasureType.TYPELESS	収集フィールド (深さが 0 のリスト) の場合、このプロパティーは、基礎となる値に関連付けられた尺度タイプを定義します。
geo_type	Point MultiPoint LineString MultiLineString Polygon MultiPolygon	地理空間フィールドの場合、このプロパティーにより、このフィールドが表す地理空間オブジェクトのタイプが定義されます。これは、値のリストの深さと整合している必要があります。
has_coordinate_system	*boolean*	地理空間フィールドの場合、このプロパティーにより、このフィールドに座標系があるかどうかが定義されます。
coordinate_system	*string*	地理空間フィールドの場合、このプロパティーにより、このフィールドの座標系が定義されます。

ensemblenode プロパティー

アンサンブル・ノードでは、2 つまたはそれ以上のモデル・ナゲットを組み合わせて 1 つのモデルよりもより正確な予測を取得します。

例

```
# 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")
```

表 82. *ensemblenode* プロパティー:

ensemblenode プロパティー	データ型	プロパティーの説明
ensemble_target_field	*field*	アンサンブルで使用されるすべてのモデルの対象フィールドを指定します。

表 82. *ensemblenode* プロパティー (続き):

ensemblenode プロパティー	データ型	プロパティーの説明
filter_individual_model_output	*flag*	個々のモデルのスコアリング結果を抑制するかどうかを指定します。
flag_ensemble_method	Voting ConfidenceWeightedVoting RawPropensityWeightedVoting AdjustedPropensityWeightedVoting HighestConfidence AverageRawPropensity AverageAdjustedPropensity	アンサンブル・スコアを決定するために使用する方法を指定します。この設定は、選択された対象がフラグ型フィールドである場合にのみ適用されます。
set_ensemble_method	Voting ConfidenceWeightedVoting HighestConfidence	アンサンブル・スコアを決定するために使用する方法を指定します。この設定は、選択された対象が名義型フィールドである場合にのみ適用されます。
flag_voting_tie_selection	Random HighestConfidence RawPropensity AdjustedPropensity	票決方法が選択された場合、可否同数の解決方法を指定します。この設定は、選択された対象がフラグ型フィールドである場合にのみ適用されます。
set_voting_tie_selection	Random HighestConfidence	票決方法が選択された場合、可否同数の解決方法を指定します。この設定は、選択された対象が名義型フィールドである場合にのみ適用されます。
calculate_standard_error	*flag*	対象フィールドが連続型の場合、標準誤差の計算がデフォルトで実施され、測定された値または推定された値と真の値との差異を計算し、それらの推定がどれほど近いかを示します。

fillernode プロパティー

置換ノードで、フィールド値の置換やストレージの変更を行います。@BLANK(@FIELD) のような、CLEM 条件に基づいて値を置換することができます。また、すべての空白値やヌル値を特定の値に置換することもできます。置換ノードは、データ型ノードと一緒に使用される場合が多く、欠損値の置き換えが行われます。

例

```
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¥"")
```

表 83. *fillernode* プロパティー

fillernode プロパティー	データ型	プロパティーの説明
fields	*list*	検査されて置換される値のデータセットのフィールド群。

表 83. *fillernode* プロパティー (続き)

fillernode プロパティー	データ型	プロパティーの説明
replace_mode	Always Conditional Blank Null BlankAndNull	すべての値、空白値、またはヌル値を置換できます。または、指定した条件に基づいて、置換できます。
condition	*string*	
replace_with	*string*	

filternode プロパティー

フィルター・ノードで、1 つの入力ノードから他の 1 つの入力ノードへ、フィールドをフィルタリング（破棄）し、フィールド名を変更し、また、フィールドを関連付けます。

例

```
node = stream.create("filter", "My node")
node.setPropertyValue("default_include", True)
node.setKeyedPropertyValue("new_name", "Drug", "Chemical")
node.setKeyedPropertyValue("include", "Drug", False)
```

default_include プロパティーの使用: default_include プロパティーの値を設定しても、すべてのフィールドが自動的に取り込まれたり除外されたりするわけではありません。単に、現在選択されている項目に対するデフォルトが決定されるだけです。これは、「フィルター・ノード」ダイアログ・ボックスで 「デフォルトでフィールドを含める」 をクリックすることと、機能的に同じです。例えば、次のスクリプトを実行すると想定します。

```
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)
```

これにより、*Age* (年齢) フィールドと *Sex* (性別) フィールドがノードを通過し、その他はすべて除外されます。次に、同じスクリプトを再び実行しますが、2 つの異なるフィールドを指定します。

```
node = modeler.script.stream().create("filter", "Filter")
node.setPropertyValue("default_include", False)
# Include these two fields in the list
for f in ["BP", "Na"]:
    node.setKeyedPropertyValue("include", f, True)
```

これにより、さらに 2 つのフィールドがフィルターに追加されたので、合計 4 フィールド (*Age* (年齢)、*Sex* (性別)、*BP* (血圧)、*Na* (ナトリウム値)) がフィルターを通過します。つまり、default_include の値を False にリセットしても、すべてのフィールドが自動的にリセットされるわけではありません。

その代わり、スクリプトを使用するか 「フィルター・ノード」ダイアログ・ボックス内で default_include を True にこの時点で変更すると、動作が反対になり、上記の 4 フィールドは上記の 4

フィールドは除外されます。「フィルター・ノード」ダイアログ・ボックス内のコントロールで実験することが、この相互関係を理解するうえで役に立ちます。

表 84. *filternode* プロパティー

filternode プロパティー	データ型	プロパティーの説明
default_include	*flag*	デフォルトの処理としてフィールドを通過させるかフィルターをかけるかの指定をするキー・プロパティー。 このプロパティーを設定しても、すべてのフィールドが自動的に取り込まれたり除外されたりするわけではありません。選択したフィールドが、デフォルトでは取り込まれるか除外されるかを決めるだけです。詳細は、下の例を参照してください。
include	*flag*	フィールドを取り込むか除外するかのキー・プロパティー。
new_name	*string*	

historynode プロパティー

時系列ノードにより、以前レコードのフィールドのデータを含む、新規フィールドが作成されます。時系列ノードは、多くの場合、時系列データなどの継続的なデータに使用されます。時系列ノードを使用する前に、ソート・ノードを使用して、データをソートしておくこともできます。

例

```
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")
```

表 85. *historynode* プロパティー

historynode プロパティー	データ型	プロパティーの説明
fields	*list*	履歴の対象となるフィールド。
offset	*number*	時系列フィールド値を抽出する最新レコードが、現在のレコードのいくつ前にあるかを指定します。
span	*number*	値を抽出する元になるレコードの前にあるレコード数を指定します。
unavailable	Discard Leave Fill	時系列として使用する前のレコードがないデータセットの先頭の数レコードを通常は指しますが、その時系列値がないレコードの取り扱い方法を指定します。

表 85. historynode プロパティー (続き)

historynode プロパティー	データ型	プロパティーの説明
fill_with	String Number	時系列値が利用できないレコードを充填するのに使用する値 (Number) または文字列 (String) を指定します。

partitionnode プロパティー

データ区分ノードで、モデル構築の学習、テスト、および検証の各ステージ用に、データを独立したサブセットに分割するデータ区分フィールドが生成されます。

例

```
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")
```

表 86. partitionnode プロパティー

partitionnode プロパティー	データ型	プロパティーの説明
new_name	*string*	ノードにより生成されたデータ区分フィールドの名前です。
create_validation	*flag*	検証用のデータ区分を作成するかどうかを指定します。
training_size	*integer*	学習用区分に割り当てるレコード数のパーセンテージ (0-100)。
testing_size	*integer*	テスト用区分に割り当てるレコード数のパーセンテージ (0-100)。
validation_size	*integer*	検証用区分に割り当てるレコード数のパーセンテージ (0-100)。検証用データ区分を生成しない場合は無視されます。
training_label	*string*	学習用データ区分のラベル。
testing_label	*string*	テスト用データ区分のラベル。
validation_label	*string*	検証用データ区分のラベル。検証用データ区分を生成しない場合は無視されます。
value_mode	System SystemAndLabel Label	データ中の各データ区分を表すために使用される値を指定します。例えば、学習用サンプルは、システム整数 1 、ラベル Training、またはこの 2 つを組み合わせた 1_Training のように表されます。
set_random_seed	*Boolean*	ユーザー指定のランダム・シードを使用するかどうかを指定します。

表 86. *partitionnode* プロパティー (続き)

partitionnode プロパティー	データ型	プロパティーの説明
random_seed	*integer*	ユーザー定義のランダム・シードの値。この値が使用されるようにするには、set_random_seed を True に設定する必要があります。
enable_sql_generation	*Boolean*	SQL プッシュバックを使用してレコードをデータ区分に割り当てるかどうかを指定します。
unique_field		レコードが無作為で繰り返し可能な方法でデータ区分に割り当てるよう、入力フィールドを指定します。この値が使用されるようにするには、enable_sql_generation を True に設定する必要があります。

reclassifynode プロパティー

データ分類ノードにより、あるカテゴリー値のセットが別のセットに変換されます。データ分類は、カテゴリーを再編成したり、分析用のデータをグループ化しなおす場合に役立ちます。

例

```
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"])
```

表 87. *reclassifynode* プロパティー

reclassifynode プロパティー	データ型	プロパティーの説明
mode	Single Multiple	1 つのフィールドのカテゴリーを再分類する場合、Single を使用します。Multiple (複数) を使用すると、一度に複数のフィールドを同時に変換できます。
replace_field	*flag*	
field	*string*	Single モードでしか使用できません。
new_name	*string*	Single モードでしか使用できません。
fields	*[field1 field2 ... fieldn]*	Multiple モードでしか使用できません。
name_extension	*string*	Multiple モードでしか使用できません。
add_as	Suffix Prefix	Multiple モードでしか使用できません。

表 87. *reclassifynode* プロパティー (続き)

reclassifynode プロパティー	データ型	プロパティーの説明
reclassify	*string*	フィールド値用構造化プロパティー。
use_default	*flag*	デフォルト値を使用します。
default	*string*	デフォルト値を指定します。
pick_list	*[string string ... string]*	ユーザーが、既知の新しい値をインポートしてテーブル内のドロップダウン・リストをデータで埋めることができるようにします。

reordernode プロパティー

フィールド順序ノードで、下流のフィールド表示に使用する順序を定義します。この順序は、テーブル、リスト、およびフィールド・ピッカーなど、さまざまな場所のフィールドの表示に適用されます。この操作は、さまざまなデータセットにおいて、特定のフィールドをより参照しやすくする場合に役立ちます。

例

```
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"])
```

表 88. *reordernode* プロパティー

reordernode プロパティー	データ型	プロパティーの説明
mode	Custom Auto	値を自動的に並び替えたり、ユーザー指定の順序を指定することができます。
sort_by	Name Type Storage	
ascending	*flag*	
start_fields	*[field1 field2 ... fieldn]*	新規フィールドは、これらのフィールドの後に挿入されます。
end_fields	*[field1 field2 ... fieldn]*	新規フィールドは、これらのフィールドの前に挿入されます。

reprojectnode プロパティー

SPSS Modeler では、式ビルダーの空間処理関数、時空間予測 (STP) ノード、マップ視覚化ノードなどの項目は、投影座標系を使用します。地理座標系を使用するインポート データの座標系を変更するには、投影ノードを使用してください。

表 89. *reprojectnode* プロパティー

reprojectnode プロパティー	データ型	プロパティーの説明
reproject_fields	*[field1 field2 ... fieldn]*	再投影されるすべてのフィールドをリストします。
reproject_type	Streamdefault Specify	フィールドの再投影方法を選択します。
coordinate_system	*string*	フィールドに適用される座標系の名前。例: set reprojectnode.coordinate_system = "WGS_1984_World_Mercator"

restructurenode プロパティー

再構成ノードで、名義型またはグラグ型フィールドを、これから別のフィールドの値で埋めることができるフィールドのグループへ変換します。例えば、*credit*、*cash*、および *debit* の値の *payment type* という名前のフィールドがある場合、3 つの新しいフィールド (*credit*、*cash*、*debit*) が作成されます。その各々には、実際の支払の値を含めることができます。

例

```
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"])
```

表 90. *restructurenode* プロパティー

restructurenode プロパティー	データ型	プロパティーの説明
fields_from	[*category category category*] all	
include_field_name	*flag*	再構成されるフィールド名に元のフィールド名を使用するかどうかを示します。
value_mode	OtherFields Flags	再構成されるフィールドの値を指定するためのモードを示します。OtherFields を指定すると、使用するフィールドを指定する必要があります (下を参照)。Flags を指定する場合、値は数値のフラグです。
value_fields	*list*	value_mode が OtherFields の場合は必須です。値のフィールドとして使用するフィールドを指定します。

rfmanalysisnode プロパティー

リーセンシ、フリクエンシ、マネタリー (RFM) の分析ノードを使用すると、最後に購入したのがどのくらい最近か (リーセンシ)、どのくらい頻繁に購入するか (フリクエンシ)、トランザクション全体でいくら消費したか (マネタリー) を検証することによって、最も良い顧客となると考えられるのはどの顧客かを量的に決定することができます。

例

```
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])
```

表 91. *rfmanalysisnode* プロパティー

rfmanalysisnode プロパティー	データ型	プロパティーの説明
recency	*field*	リーセンシ フィールドを指定します。このフィールドは日付、タイムスタンプまたは単純な数値です。
frequency	*field*	フリクエンシ フィールドを指定します。
monetary	*field*	マネタリー・フィールドを指定します。
recency_bins	*integer*	生成されるリーセンシ ビンの数を指定します。
recency_weight	*number*	リーセンシ データに適用される重みを指定します。The default is 100.
frequency_bins	*integer*	生成されるフリクエンシ ビンの数を指定します。
frequency_weight	*number*	フリクエンシ データに適用される重みを指定します。デフォルト値は 10 です。
monetary_bins	*integer*	生成されるマネタリー・ビンの数を指定します。
monetary_weight	*number*	マネタリー・データに適用される重みを指定します。デフォルトは 1 です。
tied_values_method	Next Current	可否同数の値のデータに配置されるビンを指定。
recalculate_bins	Always IfNecessary	
add_outliers	*flag*	recalculate_bins が IfNecessary に設定されている場合使用できます。設定されると、下限のビンの下にあるレコードが下限のビンに追加され、上限のビンの上にあるレコードが上限のビンに追加されます。
binned_field	Recency Frequency Monetary	

表 91. *rfmanalysisnode* プロパティー (続き)

rfmanalysisnode プロパティー	データ型	プロパティーの説明
recency_thresholds	値 値	recalculate_bins が Always に設定されている場合使用できます。リーセンシ ビンの上限および下限の閾値を指定します。あるビンの上限の閾値が次のビンの下限の閾値として使用されます。例えば、[10 30 60] は、最初のビンに 10 および 30 の上限および下限の閾値があり、2 番目のビンには 30 および 60 の閾値があると定義します。
frequency_thresholds	値 値	recalculate_bins が Always に設定されている場合使用できます。
monetary_thresholds	値 値	recalculate_bins が Always に設定されている場合使用できます。

settoflagnode プロパティー

フラグ設定ノードで、1 つ以上の名義型フィールドに定義されたカテゴリー値に基づいた、複数のフラグ型フィールドが派生します。

例

```
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"])
```

表 92. *settoflagnode* プロパティー

settoflagnode プロパティー	データ型	プロパティーの説明
fields_from	[*category category category*] all	
true_value	*string*	フラグを設定するときにノードが使用する真 (true) の値を指定します。デフォルトは T です。
false_value	*string*	フラグを設定するときにノードが使用する偽 (false) の値を指定します。デフォルトは F です。
use_extension	*flag*	新規フラグ型フィールドの接尾辞または接頭辞として、拡張子を使用します。
extension	*string*	

表 92. *settoflagnode* プロパティー (続き)

settoflagnode プロパティー	データ型	プロパティーの説明
add_as	Suffix Prefix	拡張子が接尾辞 (Suffix) または接頭辞 (Prefix) として追加されることを指定します。
aggregate	*flag*	キー・フィールドに基づいてレコードをグループ化します。真 (true) に設定されたレコードが 1 つでもあると、グループ内のすべてのフラグ型フィールドが有効になります。
keys	*list*	キー・フィールド。

statisticstransformnode プロパティー

Statistics 変換ノードは、IBM SPSS Modeler のデータ・ソースに対する IBM SPSS Statistics シンタックス・コマンドの選択を行います。このノードは、ライセンスが与えられた IBM SPSS Statistics のコピーが必要です。

このノードのプロパティーについては、 355 ページの『statisticstransformnode プロパティー』に記載されています。

timeintervalsnode プロパティー (廃止)

注: このノードは SPSS Modeler のバージョン 18 で廃止され、新しい時系列ノードに置き換えられました。時間区分ノードで、時系列データのモデル作成用に区分を指定し、必要に応じてラベルを作成します。値の間隔が均等に空けられていない場合は、レコード間に一律の間隔をとる必要に応じて、値を充填したり集計したりできます。

例

```
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")
```

表 *93. timeintervalsnode* プロパティー：

timeintervalsnode プロパティー	データ型	プロパティーの説明
interval_type	None Periods CyclicPeriods Years Quarters Months DaysPerWeek DaysNonPeriodic HoursPerDay HoursNonPeriodic MinutesPerDay MinutesNonPeriodic SecondsPerDay SecondsNonPeriodic	
mode	Label Create	レコードに連続してラベルを付ける (Label) か、または指定された日付、タイムスタンプ、または時間フィールドに基づいて系列を構築するか (Create) を指定します。
field	*field*	データから系列を構築する場合は、各レコードの日付または時刻を示すフィールドを指定します。
period_start	*integer*	期間または循環する期間の開始期間を指定します。
cycle_start	*integer*	循環する期間の開始サイクル。
year_start	*integer*	適用可能な区分タイプの、最初の区分が入る年。
quarter_start	*integer*	適用可能な区分タイプの、最初の区分が入る四半期。
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*	循環する期間の、各サイクル内の期間数。

表 93. *timeintervalsnode* プロパティー (続き):

timeintervalsnode プロパティー	データ型	プロパティーの説明
fiscal_year_begins	January February March April May June July August September October November December	四半期単位の区分の場合、会計年度が始まる月を指定します。
week_begins_on	Sunday Monday Tuesday Wednesday Thursday Friday Saturday Sunday	定期的な区分 (週当たりの日数、日当たりの時間数、日当たりの分数、日当たりの秒数) の、週が始まる曜日を指定します。
day_begins_hour	*integer*	定期的な区分 (日当たりの時間数、日当たりの分数、日当たりの秒数) の、日が始まる時間を指定します。day_begins_minute と day_begins_second を組み合わせて *8:05:01* のように、正確な時刻を指定できます。下の使用例を参照してください。
day_begins_minute	*integer*	定期的な区分 (日当たりの時間数、日当たりの分数、日当たりの秒数) の、日が始まる時間の分を指定します (例えば *8:05* の *5*)。
day_begins_second	*integer*	定期的な区分 (日当たりの時間数、日当たりの分数、日当たりの秒数) の、日が始まる時間の秒を指定します (例えば *08:05:17* の *17*)。
days_per_week	*integer*	定期的な区分 (週当たりの日数、日当たりの時間数、日当たりの分数、日当たりの秒数) の、週当たりの日数を指定します。
hours_per_day	*integer*	定期的な区分 (日当たりの時間数、日当たりの分数、日当たりの秒数) の、1 日の時間数を指定します。
interval_increment	1 2 3 4 5 6 10 15 20 30	日当たりの分数と日当たりの秒数について、各レコード用増分の分数または秒数を指定します。

表 *93. timeintervalsnode* プロパティー *(*続き*)*:

timeintervalsnode プロパティー	データ型	プロパティーの説明
field_name_extension	*string*	
field_name_extension_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"	
aggregate	Mean Sum Mode Min Max First Last TrueIfAnyTrue	フィールドの集計方法を指定します。
pad	Blank MeanOfRecentPoints True False	フィールドの充填方法を指定します。

表 93. *timeintervalsnode* プロパティー (続き):

timeintervalsnode プロパティー	データ型	プロパティーの説明
agg_mode	All Specify	必要に応じてデフォルトの関数ですべてのフィールドを集計または充填するかどうかを指定します。または、使用するフィールドと関数を指定します。
agg_range_default	Mean Sum Mode Min Max	連続型フィールドを集計するときに使用するデフォルトの関数を指定します。
agg_set_default	Mode First Last	名義型フィールドを集計するときに使用するデフォルトの関数を指定します。
agg_flag_default	TrueIfAnyTrue Mode First Last	
pad_range_default	Blank MeanOfRecentPoints	連続型フィールドをパディングするときに使用するデフォルトの関数を指定します。
pad_set_default	Blank MostRecentValue	
pad_flag_default	Blank True False	
max_records_to_create	*integer*	系列を充填するときに作成する最大レコード数を指定します。
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 プロパティー

行列入替ノードで、レコードがフィールドになり、フィールドがレコードになるように、行内と列内のデータを交換します。

例

```
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")
```

表 *94. transposenode* プロパティー

transposenode プロパティー	データ型	プロパティーの説明
transpose_method	*enum*	行列入替方法を、「通常」(normal)、「CASE から VAR へ」(casetovar)、「VAR から CASE へ」(vartocase) の中から指定します。
transposed_names	Prefix Read	「通常」行列入替方法のプロパティー。新しいフィールド名は、指定された接頭辞 (Prefix) に基づいて自動的に作成できます。または、既存のデータ内のフィールドからフィールド名を読み込むことができます (Read)。
prefix	*string*	「通常」行列入替方法のプロパティー。
num_new_fields	*integer*	「通常」行列入替方法のプロパティー。接頭辞を使用する場合は、作成する新しいフィールドの最大数を指定します。
read_from_field	*field*	「通常」行列入替方法のプロパティー。名前が読み込まれるフィールド。これはインスタンス化されたフィールドであることが必要です。そうでない場合は、ノードが実行されるときにエラーが発生します。
max_num_fields	*integer*	「通常」行列入替方法のプロパティー。フィールドから名前を読み込む場合は、異常に大量のフィールドを作成しないように、フィールド数の上限を指定します。
transpose_type	Numeric String Custom	「通常」行列入替方法のプロパティー。デフォルトでは連続型のフィールドのみの行列が入れ替えられますが、代わりに、数値フィールドのカスタム (ユーザー設定) サブセットを選択またはすべての文字列フィールドを入れ替えることもできます。
transpose_fields	*list*	「通常」行列入替方法のプロパティー。Custom (ユーザー設定) オプションを使用するときに、行列を入れ替えるフィールドを指定します。
id_field_name	*field*	「通常」行列入替方法のプロパティー。
index	*field*	「CASE から VAR へ (casetovar)」行列入替方法のプロパティー。複数のフィールドをインデックス・フィールドとして使用することを受け入れます。 field1 ... fieldN
column	*field*	「CASE から VAR へ (casetovar)」行列入替方法のプロパティー。複数のフィールドを列フィールドとして使用することを受け入れます。 field1 ... fieldN

表 94. *transposenode* プロパティー (続き)

transposenode プロパティー	データ型	プロパティーの説明
value	*field*	「CASE から VAR へ (casetovar)」行列入替方法のプロパティー。複数のフィールドを値フィールドとして使用することを受け入れます。 field1 ... fieldN
id_variables	*field*	「VAR から CASE へ (vartocase)」行列入替方法のプロパティー。複数のフィールドを ID 変数フィールドとして使用することを受け入れます。 field1 ... fieldN
value_variables	*field*	「VAR から CASE へ (vartocase)」行列入替方法のプロパティー。複数のフィールドを値変数フィールドとして使用することを受け入れます。 field1 ... fieldN

typenode プロパティー

データ型ノードで、フィールドのメタデータとプロパティーを指定します。例えば、各フィールドに、測定の尺度 (連続型、名義型、順序型、またはフラグ) を指定し、欠損値とシステムヌルの処理のためのオプションを設定し、モデル作成の目的に対するフィールドの役割を設定し、フィールドと値のラベルを指定し、フィールドの値を指定します。

例

```
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 Pressure"],
 ["NORMAL", "normal blood pressure"]])
```

ある種の場合、ほかのノードが正しく機能するように、フラグ設定ノードの fields from プロパティーのように、データ型ノードを完全にインスタンス化する必要がある場合があります。フィールドをインスタンス化するには、次のように、テーブル・ノードを接続して実行するだけです。

```
tablenode = stream.createAt("table", "Table node", 150, 50)
stream.link(node, tablenode)
tablenode.run(None)
stream.delete(tablenode)
```

表 *95. typenode* プロパティー：

typenode プロパティー	データ型	プロパティーの説明
direction	Input Target Both None Partition Split Frequency RecordID	フィールドの役割のキープロパティー。 **注:** 値 In と Out は廃止されました。今後のリリースではサポートが中断される場合があります。
Type	Range Flag Set Typeless Discrete OrderedSet default	フィールドの尺度 (以前はフィールドの「タイプ」と呼ばれていました)。type を Default に設定すると values パラメーター設定をクリアします。value_mode の値が Specify の場合、Read にリセットします。 value_mode が Pass または Read に設定される場合、type を設定しても value_mode には影響ありません。 **注:** 内部で使用されるデータ型は、データ型ノードに表示されるデータ型とは異なります。次のように対応します: 範囲型 > 連続セット型 -> 名義順序セット型 -> 順序離散型- > カテゴリー型
storage	Unknown String Integer Real Time Date Timestamp	フィールドのストレージ・タイプ用読み込み専用キー・プロパティー。
check	None Nullify Coerce Discard Warn Abort	フィールド・タイプと範囲の検査用のキー・プロパティー。
values	[*value value*]	連続型フィールドの場合、最初の値が最小値で最後の値が最大値になります。名義型フィールドの場合、すべての値を指定します。フラグ型の場合、最初の値が *false* (偽) を、最後の値が *true* (真) を表します。このプロパティーを設定すると、value_mode プロパティーの値が自動的に Specify に設定されます。
value_mode	Read Pass Read+ Current Specify	値の設定方法を決定します。このプロパティーに Specify を直接には設定できないことに注意してください。特定の値を使用するには、values プロパティーを設定します。

表 *95. typenode* プロパティー *(*続き*)*:

typenode プロパティー	データ型	プロパティーの説明
extend_values	*flag*	value_mode が Read に設定された場合に適用されます。新しく読み込んだ値を、フィールドの既存の値に追加する場合は、T を設定します。新しく読み込んだ値を優先して、既存の値を破棄する場合は、F を設定します。
enable_missing	*flag*	T を設定した場合、フィールドの欠損値の追跡が有効になります。
missing_values	[*value value ...*]	欠損データを示すデータ値を指定します。
range_missing	*flag*	フィールドに欠損値 (空白) の範囲が定義されているかどうかを指定します。
missing_lower	*string*	range_missing が真 (true) の場合、欠損値範囲の下限値を指定します。
missing_upper	*string*	range_missing が真 (true) の場合、欠損値範囲の上限値を指定します。
null_missing	*flag*	T を設定した場合、ヌル値 (ソフトウェアでは $null$ として表示される未定義値) は欠損値と見なされます。
whitespace_ missing	*flag*	T を設定した場合、空白類 (スペース、タブ、および改行) だけを含む値が欠損値と見なされます。
description	*string*	フィールドの説明を指定します。
value_labels	*[[Value LabelString] [Value LabelString] ...]*	値のペアのためのラベルを指定します。
display_places	*integer*	フィールドが表示されるときの小数部の桁数を設定します (REAL ストレージのフィールドにのみ適用)。-1 を設定すると、ストリームのデフォルトが使用されます。
export_places	*integer*	フィールドが表示されるときの小数部の桁数を設定します (REAL ストレージのフィールドにのみ適用)。-1 を設定すると、ストリームのデフォルトが使用されます。
decimal_separator	DEFAULT PERIOD COMMA	フィールドの小数点記号を指定します (REAL ストレージのフィールドにのみ適用)。

表 *95. typenode* プロパティー *(*続き*)*:

typenode プロパティー	データ型	プロパティーの説明
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	フィールドの日付形式を設定します (DATE または TIMESTAMP ストレージのフィールドにのみ適用されます)。
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"	フィールドの日付形式を設定します (TIME または TIMESTAMP ストレージのフィールドにのみ適用されます)。
number_format	DEFAULT STANDARD SCIENTIFIC CURRENCY	フィールドに数値の表示形式を設定します。
standard_places	*integer*	フィールドが標準形式で表示されるときの小数点以下の桁数を指定します。-1 を設定すると、ストリームのデフォルトが使用されます。既存の display_places スロットでもこの設定が変更されますが、現在は廃止されています。
scientific_places	*integer*	フィールドが科学系の形式で表示されるときの小数点以下の桁数を設定します。-1 を設定すると、ストリームのデフォルトが使用されます。

表 95. *typenode* プロパティー (続き):

typenode プロパティー	データ型	プロパティーの説明
currency_places	*integer*	フィールドが通貨の形式で表示されるときのフィールドの小数点以下の桁数を設定します。-1 を設定すると、ストリームのデフォルトが使用されます。
grouping_symbol	DEFAULT NONE LOCALE PERIOD COMMA SPACE	フィールドにグループ化シンボルを設定します。
column_width	*integer*	フィールドに列幅を設定します。-1 という値を指定すると、列幅は Auto に設定されます。
justify	AUTO CENTER LEFT RIGHT	フィールドに列調整を設定します。
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	このキー付きプロパティーは、フィールドに関連付けられた尺度を定義するために使用できるという点で、type と類似しています。異なるのは、Python スクリプトで、getter 関数が常に MeasureType 値を返す一方で、setter 関数に MeasureType 値のうちの 1 つを渡すこともできるという点です。
collection_measure	Range / MeasureType.RANGE Flag / MeasureType.FLAG Set / MeasureType.SET OrderedSet / MeasureType.ORDERED_SET Typeless / MeasureType.TYPELESS	収集フィールド (深さが 0 のリスト) の場合、このキー付きプロパティーは、基礎となる値に関連付けられた尺度タイプを定義します。
geo_type	Point MultiPoint LineString MultiLineString Polygon MultiPolygon	地理空間フィールドの場合、このキー付きプロパティーにより、このフィールドが表す地理空間オブジェクトのタイプが定義されます。これは、値のリストの深さと整合している必要があります。
has_coordinate_system	*boolean*	地理空間フィールドの場合、このプロパティーにより、このフィールドに座標系があるかどうかが定義されます。
coordinate_system	*string*	地理空間フィールドの場合、このキー付きプロパティーにより、このフィールドの座標系が定義されます。

表 95. *typenode* プロパティー (続き):

typenode プロパティー	データ型	プロパティーの説明
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	このキー付きプロパティーは、フィールドのオーバーライド ストレージを定義するために使用できるという点で、custom_storage と類似しています。異なるのは、Python スクリプトで、getter 関数が常に StorageType 値を返す一方で、setter 関数に StorageType 値のうちの 1 つを渡すこともできるという点です。
custom_list_storage_type	String / MeasureType.STRING Integer / MeasureType.INTEGER Real / MeasureType.REAL Time / MeasureType.TIME Date / MeasureType.DATE Timestamp / MeasureType.TIMESTAMP	リスト フィールドの場合、このキー付きプロパティーにより、基礎となる値のストレージタイプが指定されます。
custom_list_depth	*integer*	リスト フィールドの場合、このキー付きプロパティーにより、フィールドの深さが指定されます。
max_list_length	*integer*	地理空間または集合のいずれかの尺度を持つデータのみに使用できます。リストの最大長を設定するには、リストに入れることができる要素の数を指定します。
max_string_length	*integer*	データ型不明 のデータでのみ使用可能で、SQL を生成してテーブルを作成するときに使用されます。データの最大文字列の値を入力します。これにより、テーブルに生成される列が、その文字列を含めるのに十分な大きさになります。

第 12 章 グラフ作成ノードのプロパティー

グラフ・ノードの共通プロパティー

このセクションでは、グラフ作成ノードで使用できるプロパティーについて、共通なプロパティーとノード・タイプ固有のプロパティーも含めて説明します。

表 96. グラフ作成ノードの共通プロパティー

グラフ作成ノードの共通プロパティー	データ型	プロパティーの説明
title	*string*	タイトルを指定します。例:"This is a title."
caption	*string*	解説を指定します。例:"This is a caption."
output_mode	Screen File	グラフ作成ノードからの出力が表示されるか、ファイルへ書き込まれるかを指定します。
output_format	BMP JPEG ファイル PNG HTML output (*.cou*)	出力のタイプを指定します。出力可能なタイプは、各ノードに応じて変化します。
full_filename	*string*	グラフ作成ノードから生成されたグラフの、出力先のパスとファイル名を指定します。
use_graph_size	*flag*	下に説明する幅と高さのプロパティーを使用してグラフのサイズが明示して設定されるかどうかを制御します。画面に出力されるグラフにだけ影響します。棒グラフ・ノードには使用できません。
graph_width	*number*	use_graph_size が True の場合、グラフの幅をピクセル数で指定します。
graph_height	*number*	use_graph_size が True の場合、グラフの高さをピクセル数で指定します。

オプション フィールドの無効化

散布図のオーバーレイ・フィールドなどのオプション・フィールドは、次の例のようにプロパティー値に "" (空文字列) を設定することにより、無効化することができます。

```
plotnode.setPropertyValue("color_field", "")
```

色の指定

表題、解説、背景、およびラベルの色は、ハッシュ記号 (#) で始まる 16 進文字列で指定することができます。例えば、グラフの背景を空色にするには、次の文を指定します。

```
mygraphnode.setPropertyValue("graph_background", "#87CEEB")
```

ここで、最初の 2 桁 87 は赤色の量を、次の 2 桁 CE は緑の量を、最後の 2 桁 EB は青の量を示します。各桁は、0 から 9 または A から F の範囲の値になります。これらの値を使用して、赤-緑-青 (RGB) の色を指定します。

注: 色を RGB で指定する場合、ユーザー インターフェースのフィールド ピッカーを使用して正しい色コードを決定することができます。ピッカーを目的の色の上にかざせば、その色コードがツールヒントに表示されます。

collectionnode プロパティー

集計棒グラフ・ノードで、他の数値フィールドの値に相対的な数値フィールドの値の棒グラフを表示します（集計棒グラフ・ノードでは、ヒストグラムに似たグラフが作成されます）。集計棒グラフは、値が時間の経過とともに変化する変数やフィールドを表示する場合に役立ちます。3 次元グラフを使用して、分布をカテゴリー別に表示するシンボル値軸を追加することもできます。

例

```
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)
```

表 97. *collectionnode* プロパティー

collectionnode プロパティー	データ型	プロパティーの説明
over_field	*field*	
over_label_auto	*flag*	
over_label	*string*	
collect_field	*field*	
collect_label_auto	*flag*	
collect_label	*string*	
three_D	*flag*	
by_field	*field*	
by_label_auto	*flag*	
by_label	*string*	
operation	Sum Mean Min Max SDev	

表 97. *collectionnode* プロパティー (続き)

collectionnode プロパティー	データ型	プロパティーの説明
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*	標準のグラフ色は、このセクションの最初に説明されています。
page_background	*color*	標準のグラフ色は、このセクションの最初に説明されています。

distributionnode プロパティー

棒グラフ・ノードで、ローンの種類や性別など、シンボル値（カテゴリー) の出現頻度を表示します。通常、棒グラフ・ノードを使用してデータの不均衡を表示しますが、そのデータはモデルの作成前にバランス・ノードを使用して修正できます。

例

```
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)
```

表 98. *distributionnode* プロパティー

distributionnode プロパティー	データ型	プロパティーの説明
plot	SelectedFields Flags	
x_field	*field*	
color_field	*field*	オーバーレイ・フィールド。
normalize	*flag*	
sort_mode	ByOccurence Alphabetic	

表 98. *distributionnode* プロパティー (続き)

distributionnode プロパティー	データ型	プロパティーの説明
use_proportional_scale	*flag*	

evaluationnode プロパティー

評価ノードは、予測モデルの評価と比較に用いられます。評価グラフで、モデルが特定の結果をどの程度予測するかを表示します。それによって、予測値と予測の信頼度に基づいたレコードがソートされます。そして、レコードが等サイズ (分位) のグループに分割され、各分位のビジネスに関する基準の値が、高い方から降順で作図されます。作図には、複数のモデルが異なる線で示されます。

例

```
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")
```

表 99. *evaluationnode* プロパティー:

evaluationnode プロパティー	データ型	プロパティーの説明
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*	

表 99. *evaluationnode* プロパティー (続き):

evaluationnode プロパティー	データ型	プロパティーの説明
weight_value	*number*	
weight_field	*field*	
n_tile	Quartiles Quintles Deciles Vingtiles Percentiles 1000-tiles	
cumulative	*flag*	
style	Line Point	
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 ... fieldN]*	
target_field	*field*	
use_hit_condition	*flag*	
hit_condition	*string*	
use_score_expression	*flag*	
score_expression	*string*	
caption_auto	*flag*	

graphboardnode プロパティー

グラフボード・ノードでは、単一のノードにさまざまな種類のグラフを提供しています。このノードを使用して、検証するデータ・フィールドを選択肢、選択したデータに使用できるグラフを選択できます。選択したフィールドに適していないグラフの種類は、ノードによって自動的に除外されます。

注: グラフ タイプに対して無効なプロパティーを設定した場合 (例えば、ヒストグラフに対して y_field を指定した場合)、そのプロパティーは無視されます。

注: UI には、さまざまなグラフ タイプの「詳細」タブに「要約」フィールドがあります。このフィールドは、現在スクリプトではサポートされていません。

例

```
node = stream.create("graphboard", "My node")
node.setPropertyValue("graph_type", "Line")
node.setPropertyValue("x_field", "K")
node.setPropertyValue("y_field", "Na")
```

表 *100. graphboardnode* プロパティー

graphboard プロパティー	データ型	プロパティーの説明
graph_type	2DDotplot 3DArea 3DBar 3DDensity 3DHistogram 3DPie 3DScatterplot Area ArrowMap Bar BarCounts BarCountsMap BarMap BinnedScatter Boxplot Bubble ChoroplethMeans ChoroplethMedians ChoroplethSums ChoroplethValues ChoroplethCounts CoordinateMap CoordinateChoroplethMeans CoordinateChoroplethMedians CoordinateChoroplethSums CoordinateChoroplethValues CoordinateChoroplethCounts Dotplot Heatmap HexBinScatter Histogram Line LineChartMap LineOverlayMap Parallel Path Pie PieCountMap PieCounts PieMap PointOverlayMap PolygonOverlayMap Ribbon Scatterplot SPLOM Surface	グラフの種類を識別します。

表 100. graphboardnode プロパティー (続き)

graphboard プロパティー	データ型	プロパティーの説明
x_field	*field*	*x* 軸のカスタム (ユーザー設定) ラベルを指定します。ラベルでのみ使用できます。
y_field	*field*	*y* 軸のカスタム (ユーザー設定) ラベルを指定します。ラベルでのみ使用できます。
z_field	*field*	3 次元グラフの一部で使用します。
color_field	*field*	ヒート・マップで使用します。
size_field	*field*	バブル・プロットで使用します。
categories_field	*field*	
values_field	*field*	
rows_field	*field*	
columns_field	*field*	
fields	*field*	
start_longitude_field	*field*	参照マップの矢印で使用します。
end_longitude_field	*field*	
start_latitude_field	*field*	
end_latitude_field	*field*	
data_key_field	*field*	さまざまなマップで使用します。
panelrow_field	*string*	
panelcol_field	*string*	
animation_field	*string*	
longitude_field	*field*	マップ上の座標で使用します。
latitude_field	*field*	
map_color_field	*field*	

histogramnode プロパティー

ヒストグラム・ノードでは、数値フィールドの値の出現頻度が示されます。多くの場合、ヒストグラム・ノードは、操作やモデルの構築前にデータを調べるために使用されます。棒グラフ・ノードと同様、ヒストグラム・ノードにより、データ内の不均衡がしばしば明らかになります。

例

```
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)
```

表 *101. histogramnode* プロパティー

histogramnode プロパティー	データ型	プロパティーの説明
field	*field*	
color_field	*field*	
panel_field	*field*	
animation_field	*field*	
range_mode	Automatic UserDefined	
range_min	*number*	
range_max	*number*	
bins	ByNumber ByWidth	
num_bins	*number*	
bin_width	*number*	
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*	標準のグラフ色は、このセクションの最初に説明されています。
page_background	*color*	標準のグラフ色は、このセクションの最初に説明されています。
normal_curve	*flag*	正規分布のカーブを出力に表示するかどうかを指定します。

mapvisualization プロパティ

マップ視覚化ノードは、複数の入力接続を受け入れて、地理空間データを一連の層としてマップに表示することができます。各層は単一の地理空間フィールドです。例えば、基本層を国のマップとし、その上に道路の層、川の層、町の層を設けることができます。

表 102. *mapvisualization* プロパティ

mapvisualization プロパティ	データ型	プロパティーの説明
tag	*string*	入力用のタグの名前を設定します。デフォルトのタグは、入力がノードに接続された順序に基づく数値です (最初の接続タグは 1、2 番目の接続タグは 2 という方法で数値が設定されます)。
layer_field	*field*	マップ上に層として表示する、データ・セットからの地理フィールドを選択します。デフォルトの選択内容は、次のソート順に基づきます。 • 最初 - 点 • 行ストリング • 多角形 • 複数点 • 複数行ストリング • 最後 - 多角形群 同じ尺度タイプを持つ 2 つのフィールドがある場合、デフォルトでは、名前のアルファベット順で最初のフィールドが選択されます。
color_type	*boolean*	標準の色を地理フィールドのすべてのフィーチャーに適用するか、オーバーレイ フィールドを適用するかを指定します (オーバーレイ フィールドでは、データ・セットの他のフィールドの値に基づいて、フィーチャーに色が付けられます)。指定できる値は、standard または overlay です。デフォルトは standard です。
color	*string*	color_type に standard を選択した場合、ドロップダウンには、「ユーザー・オプション」の「表示」タブにある「グラフ カテゴリーの色順序」と同じ色パレットが含まれます。 デフォルトの「グラフ カテゴリーの色」は、1 です。
color_field	*field*	color_type に overlay を選択した場合、ドロップダウンには、層として選択された地理フィールドと同じデータ・セットからのすべてのフィールドが含まれます。
symbol_type	*boolean*	標準の記号を地理フィールドのすべてのレコードに適用するか、オーバーレイ記号を適用するかを指定します (オーバーレイ記号では、データ・セットの他のフィールドの値に基づいて、ポイントの記号アイコンが変更されます)。指定できる値は、standard または overlay です。デフォルトは standard です。
symbol	*string*	symbol_type に standard を選択した場合、ドロップダウンには、マップ上にポイントを表示するために使用される記号の選択項目が含まれます。

表 102. *mapvisualization* プロパティ (続き)

mapvisualization プロパティ	データ型	プロパティーの説明
symbol_field	*field*	symbol_type に overlay を選択した場合、ドロップダウンには、層として選択された地理フィールドと同じデータ・セットからの名義型フィールド、順序型フィールド、またはカテゴリー型フィールドがすべて含まれます。
size_type	*boolean*	標準のサイズを地理フィールドのすべてのレコードに適用するか、オーバーレイのサイズを適用するかを指定します (オーバーレイのサイズでは、データ・セットの他のフィールドの値に基づいて、記号アイコンのサイズまたは線の太さが変更されます)。指定できる値は、standard または overlay です。デフォルトは standard です。
size	*string*	size_type、point、または multipoint に standard を選択した場合、ドロップダウンには、選択した記号のサイズの選択項目が含まれます。linestring または multilinestring の場合、ドロップダウンには、線の太さの選択項目が含まれます。
size_field	*field*	size_type に overlay を選択した場合、ドロップダウンには、層として選択された地理フィールドと同じデータ・セットからのすべてのフィールドが含まれます。
transp_type	*boolean*	標準の透過度を地理フィールドのすべてのレコードに適用するか、オーバーレイ透過度を適用するかを指定します (オーバーレイ透過度では、データ・セットの他のフィールドの値に基づいて、記号、線、または多角形の透過度のレベルが変更されます)。指定できる値は、standard または overlay です。デフォルトは standard です。
transp	*integer*	transp_type に standard を選択した場合、ドロップダウンには、透過度レベルの選択項目が含まれます。この項目は、0% (不透明) から 100% (透明) まで 10% 刻みで増加します。マップ上のポイント、線、または多角形の透過度を設定します。 size_type に overlay を選択した場合、ドロップダウンには、層として選択された地理フィールドと同じデータ・セットからのすべてのフィールドが含まれます。 points、multipoints、linestrings、および multilinestrings、polygons および multipolygons (最下層) の場合、デフォルトは 0% です。最下層でない polygons および multipolygons の場合、デフォルトは 50% です (これらの多角形の下の層が覆い隠されることを避けるため)。

表 *102. mapvisualization* プロパティ *(*続き*)*

mapvisualization プロパティ	データ型	プロパティーの説明
transp_field	*field*	transp_type に overlay を選択した場合、ドロップダウンには、層として選択された地理フィールドと同じデータ・セットからのすべてのフィールドが含まれます。
data_label_field	*field*	マップのデータ ラベルとして使用するフィールドを指定します。例えば、この設定の適用先の層が多角形の層の場合は、データ ラベルを name フィールドにして、それぞれの多角形の名前を含めることができます。そのため、ここで name フィールドを選択すると、それらの名前がマップに表示されるようになります。
use_hex_binning	*boolean*	六角ビン分割を有効にし、すべての集計ドロップダウンを有効にします。デフォルトでは、この設定はオフになっています。

表 102. *mapvisualization* プロパティ (続き)

mapvisualization プロパティ	データ型	プロパティーの説明
color_aggregation および transp_aggregation	*string*	六角ビン分割を使用するポイント層に対して「オーバーレイ」フィールドを選択した場合は、六角形の中にあるすべてのポイントについて、そのフィールドのすべての値を集計する必要があります。したがって、マップに適用するすべての「オーバーレイ」フィールドについて、集計関数を指定する必要があります。 使用可能な集計関数を次に示します。 連続型 (実数または整数のストレージ): • 合計 • 平均値 • 最小値 • 最大値 • 中央値 • 第 1 四分位数 • 第 3 四分位数 連続型 (時間、日付、またはタイムスタンプのストレージ): • 平均値 • 最小値 • 最大値 名義型/カテゴリ型: • モード • 最小値 • 最大値 フラグ型: • いずれかが真の場合は真 • いずれかが偽の場合は偽
custom_storage	*string*	フィールド全体のストレージ タイプを設定します。デフォルトは List です。List を指定した場合は、次の custom_value_storage コントロールと list_depth コントロールが無効になります。
custom_value_storage	*string*	フィールド全体ではなく、リスト内の要素のストレージ タイプを設定します。デフォルトは Real です。

表 102. *mapvisualization* プロパティ (続き)

mapvisualization プロパティ	データ型	プロパティーの説明
list_depth	*integer*	リスト フィールドの深さを設定します。必要な深さは、地理フィールドのタイプによって異なり、次の基準に従います。 • ポイント - 0 • 行ストリング - 1 • 多角形 - 2 • 複数点 – 1 • 複数行ストリング - 2 • 多角形群 - 3 リストに変換し直す地理空間フィールドのタイプと、その種類のフィールドに必要な深さを把握しておく必要があります。設定に誤りがあると、フィールドを使用できません。 デフォルト値は 0、最小値は 0、最大値は 10 です。

multiplotnode プロパティー

線グラフ・ノードでは、1 つの *X* フィールドに対して複数の *Y* フィールドを表示する作図が作成されます。*Y* フィールドは色付きの線で作図され、それぞれ「スタイル」フィールドを「ライン」に、「X モード」フィールドを「ソート」に設定した散布図ノードに相当します。線グラフは、複数の変数の変動を長期にわたって調査するときに役立ちます。

例

```
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")
```

表 103. *multiplotnode* プロパティー

multiplotnode プロパティー	データ型	プロパティーの説明
x_field	*field*	
y_fields	*list*	
panel_field	*field*	
animation_field	*field*	

表 103. *multiplotnode* プロパティー (続き)

multiplotnode プロパティー	データ型	プロパティーの説明
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*	標準のグラフ色は、このセクションの最初に説明されています。
page_background	*color*	標準のグラフ色は、このセクションの最初に説明されています。

plotnode プロパティー

散布図ノードで、数値フィールド間の関係が示されます。作図は、点 (散布図) または折れ線を使用して作成できます。

例

```
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_output.jpeg")
```

表 *104. plotnode* プロパティー：

plotnode プロパティー	データ型	プロパティーの説明
x_field	*field*	*x* 軸のカスタム (ユーザー設定) ラベルを指定します。ラベルでのみ使用できます。
y_field	*field*	*y* 軸のカスタム (ユーザー設定) ラベルを指定します。ラベルでのみ使用できます。
three_D	*flag*	*y* 軸のカスタム (ユーザー設定) ラベルを指定します。3-D グラフのラベルでのみ使用できます。
z_field	*field*	
color_field	*field*	オーバーレイ・フィールド。
size_field	*field*	
shape_field	*field*	
panel_field	*field*	各カテゴリー個別のグラフの作成に使用する名義型またはフラグ型フィールドを指定します。グラフは「パネル化」され、複数のグラフが 1 つの出力ウィンドウに表示されます。
animation_field	*field*	アニメーションを使用して順番に表示する一連のグラフを作成してデータ値のカテゴリーを描画する、名義型またはフラグ型フィールドを指定します。
transp_field	*field*	カテゴリーごとに異なるレベルの透過度を使用して、データ値のカテゴリーを表すフィールドを指定します。折れ線グラフでは使用できません。
overlay_type	None Smoother Function	オーバーレイ関数が表示されるか、LOESS 平滑化が表示されるかを指定します。
overlay_expression	*string*	overlay_type が Function に設定されているときに使用される式を指定します。
style	Point Line	
point_type	Rectangle Dot Triangle Hexagon Plus Pentagon Star BowTie HorizontalDash VerticalDash IronCross Factory House Cathedral OnionDome ConcaveTriangle OblateGlobe CatEye FourSidedPillow RoundRectangle Fan	

表 104. *plotnode* プロパティー (続き):

plotnode プロパティー	データ型	プロパティーの説明
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	
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*	標準のグラフ色は、このセクションの最初に説明されています。
page_background	*color*	標準のグラフ色は、このセクションの最初に説明されています。
use_overlay_expr	*flag*	overlay_type の代わりに廃止される予定。

timeplotnode プロパティー

時系列ノードで、時系列データの 1 つ以上のセットを表示します。通常、最初に時間区分ノードを使用して *TimeLabel* フィールドを作成します。このフィールドは、*x* 軸にラベルを付けるために使用されます。

例

```
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)
```

表 *105. timeplotnode* プロパティー：

timeplotnode プロパティー	データ型	プロパティーの説明
plot_series	Series Models	
use_custom_x_field	*flag*	
x_field	*field*	
y_fields	*list*	
panel	*flag*	
normalize	*flag*	
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*	panel を True に設定した場合にのみ、平滑化を散布図に追加できます。
use_records_limit	*flag*	
records_limit	*integer*	
symbol_size	*number*	マーカー・サイズを指定します。
panel_layout	Horizontal Vertical	

webnode プロパティー

Web グラフ・ノードで、複数のシンボル値 (カテゴリー) フィールドの値の関係の強さが示されます。このグラフでは、接続の強さを示すためにさまざまな幅の線が使用されます。Web グラフ・ノードを使用して、例えば、E コマース・サイトで購入されたさまざまな商品の関係を調査できます。

例

```
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")
```

表 *106. webnode* プロパティー

webnode プロパティー	データ型	プロパティーの説明
use_directed_web	*flag*	
fields	*list*	
to_field	*field*	
from_fields	*list*	
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*	

表 106. *webnode* プロパティー (続き)

webnode プロパティー	データ型	プロパティーの説明
links_max_records	*number*	
weak_below	*number*	
strong_above	*number*	
link_size_continuous	*flag*	
web_display	Circular Network Directed Grid	
graph_background	*color*	標準のグラフ色は、このセクションの最初に説明されています。
symbol_size	*number*	マーカー・サイズを指定します。

第 13 章 モデル作成ノードのプロパティー

一般的なモデル作成ノードのプロパティー

次のプロパティーは、複数またはすべてのデータベース・モデル作成ードに共通です。個別のモデル作成ノードに関しては、必要に応じてドキュメント内に例外を記載しています。

表 *107.* 一般的なモデル作成ノードのプロパティー

プロパティー	値	プロパティーの説明
custom_fields	*flag*	真 (true) の場合は、現在のノードのターゲット、入力、その他フィールドなどを指定することができます。偽 (false) の場合は、上流のデータ型ノードから現在の設定が使用されます。
target または targets	*field* or [*field1 ... fieldN*]	モデルのタイプによって、単一の対象フィールドまたは複数の対象フィールドを指定します。
inputs	[*field1 ... fieldN*]	モデルで使用される入力または予測変数フィールド。
partition	*field*	
use_partitioned_data	*flag*	区分フィールドが定義される場合、このオプションは学習データ区分からのデータのみがモデル構築に使用されるようにします。
use_split_data	*flag*	
splits	*[field1 ... fieldN]*	分割モデル作成に使用する、フィールドを選択します。use_split_data が True に設定されている場合にのみ有効です。
use_frequency	*flag*	各モデル・タイプで言及するとおり、重みフィールドおよび度数フィールドが特定のモデルで使用されます。
frequency_field	*field*	
use_weight	*flag*	
weight_field	*field*	
use_model_name	*flag*	
model_name	*string*	ユーザーが指定する新規モデル名。
mode	Simple (単純) Expert	

anomalydetectionnode プロパティー

異常値検出ノードで、「正常な」データのパターンに合致しない異常ケースや外れ値を識別します。このノードで、外れ値が既知のパターンに当てはまらなかったり、何を探しているのかはっきりしなかったりする場合でも、外れ値を識別できます。

例

```
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)
```

表 *108. anomalydetectionnode* プロパティー

anomalydetectionnode プロパティー	値	プロパティーの説明
inputs	*[field1 ... fieldN]*	異常値検出モデルは、指定の入力フィールドに基づいてレコードをスクリーニングします。ターゲット・フィールドは使用しません。重みフィールドおよび度数フィールドも使用しません。 詳しくは、トピック 187 ページの『一般的なモデル作成ノードのプロパティー』を参照してください。
mode	Expert Simple	
anomaly_method	IndexLevel PerRecords NumRecords	レコードに異常としてフラグを設定するための、分割値を決めるのに使用される方法を指定します。
index_level	*number*	異常としてフラグを設定するための最小分割値を指定します。
percent_records	*number*	学習データ内のレコードの割合 (%) に基づいてレコードにフラグを設定するための、閾値を設定します。
num_records	*number*	学習データ内のレコードの数に基づいてレコードにフラグを設定するための、閾値を設定します。
num_fields	*integer*	各異常レコードに報告するフィールド数。
impute_missing_values	*flag*	
adjustment_coeff	*number*	距離の計算時,É連続型とカテゴリー・フィールド間に指定された関連の重みのバランスをとるために使用される値。
peer_group_num_auto	*flag*	ピア・グループ数を自動的に計算します。

表 108. *anomalydetectionnode* プロパティー (続き)

anomalydetectionnode プロパティー	値	プロパティーの説明
min_num_peer_groups	*integer*	peer_group_num_auto が True に設定されている場合に使用されるピア・グループの最小数を指定します。
max_num_per_groups	*integer*	ピア・グループの最大数を指定します。
num_peer_groups	*integer*	peer_group_num_auto が False に設定されている場合に使用されるピア・グループの数を指定します。
noise_level	*number*	クラスタリング中の外れ値の処理方法を決定します。0 から 0.5 までの値を指定してください。
noise_ratio	*number*	ノイズのバッファリングに使用されるコンポーネントに割り当てられる、メモリーの量を指定します。0 から 0.5 までの値を指定してください。

apriorinode プロパティー

Apriori ノードで、データからルール・セットを抽出し、情報内容が最も充実したルールを引き出します。Apriori には、5 種類のルール選択方法があり、高度なインデックス作成方法を使用して、大きなデータ・セットが効率的に処理されます。大きな問題の場合は、一般に、Apriori の方が高速に学習できます。保持できるルール数に特に制限はありません。また、最大 32 の前提条件を持つルールを処理できます。Apriori では、入力フィールドと出力フィールドのすべてがカテゴリーであることが必要ですが、この種類のデータに合わせて最適化されているので、よりよいパフォーマンスを実現します。

例

```
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)
```

表 109. *apriorinode* プロパティー

apriorinode プロパティー	値	プロパティーの説明
consequents	*field*	Apriori モデルは標準的な対象フィールドおよび入力フィールドの結果と条件を使用します。重みフィールドおよび度数フィールドは使用しません。 詳しくは、トピック 187 ページの『一般的なモデル作成ノードのプロパティー』を参照してください。
antecedents	[*field1 ... fieldN*]	
min_supp	*number*	
min_conf	*number*	
max_antecedents	*number*	
true_flags	*flag*	
optimize	Speed Memory	
use_transactional_data	*flag*	
contiguous	*flag*	
id_field	*string*	
content_field	*string*	
mode	Simple (単純) Expert	
evaluation	RuleConfidence DifferenceToPrior ConfidenceRatio InformationDifference NormalizedChiSquare	
lower_bound	*number*	
optimize	Speed Memory	モデル作成が速度とメモリーのどちらにより最適化されるかを指定します。

associationrulesnode プロパティー

アソシエーション ルール ノードは Apriori ノードに似ていますが、Apriori とは異なり、アソシエーション ルール ノードはリスト・データを処理できます。さらに、アソシエーション ルール ノードを IBM SPSS Analytic Server と共に使用すると、ビッグデータの処理や高速な並列処理の利用が可能になります。

表 *110. associationrulesnode* プロパティー

associationrulesnode プロパティー	データ型	プロパティーの説明
predictions	*field*	このリスト内の各フィールドは、ルールの予測フィールドとしてのみ表示することができます。
conditions	*[field1...fieldN]*	このリスト内の各フィールドは、ルールの条件としてのみ表示することができます。
max_rule_conditions	*integer*	1 つのルールに含めることができる条件の最大数。最小値は 1、最大値は 9 です。
max_rule_predictions	*integer*	1 つのルールに含めることができる予測の最大数。最小値は 1、最大値は 5 です。
max_num_rules	*integer*	ルール構築の一部としてみなすことができるルールの最大数。最小値は 1、最大値は 10,000 です。
rule_criterion_top_n	Confidence Rulesupport Lift Conditionsupport Deployability	値を判断するルール基準。この基準により、モデル内の上位 N 件のルールが選択されます。
true_flags	*Boolean*	これを *Y* に設定すると、ルールの構築時に、true の値を持つフラグ フィールドだけが処理対象になります。
rule_criterion	*Boolean*	これを *Y* に設定すると、モデルの構築時に、ルール基準の値を使用してルールが除外されます。
min_confidence	*number*	0.1 から 100: モデルによって生成されたルールについて最低限必要な確信度レベルのパーセント値。ここで指定された値よりも低い確信度レベルを持つルールがモデルによって生成された場合、そのルールは破棄されます。
min_rule_support	*number*	0.1 から 100: モデルによって生成されたルールについて最低限必要なルール サポートのパーセント値。ここで指定された値よりも低いルール サポート レベルを持つルールがモデルによって生成された場合、そのルールは破棄されます。
min_condition_support	*number*	0.1 から 100: モデルによって生成されたルールについて最低限必要な条件サポートのパーセント値。ここで指定された値よりも低い条件サポート レベルを持つルールがモデルによって生成された場合、そのルールは破棄されます。
min_lift	*integer*	1 から 10: モデルによって生成されたルールについて最低限必要なリフト レベルを表します。ここで指定された値よりも低いリフト レベルを持つルールがモデルによって生成された場合、そのルールは破棄されます。

表 *110. associationrulesnode* プロパティー *(*続き*)*

associationrulesnode プロパティー	データ型	プロパティーの説明
exclude_rules	*Boolean*	このプロパティーを使用して、モデルによるルールの作成元として使用しない関連フィールドのリストを選択します。 例: set :gsarsnode.exclude_rules = [[[field1,field2, field3]],[[field4, field5]]] - [] 内に指定されたフィールドのリストが、テーブル内の各行になります。
num_bins	*integer*	連続型フィールドのビン分割先となる自動ビンの数を設定します。最小値は 2、最大値は 10 です。
max_list_length	*integer*	最大長が不明なすべてのリスト フィールドに適用されます。ここで指定された数を上限として、リスト内の要素がモデルの構築で使用されます。ここで指定された数を超える要素については、すべて破棄されます。最小値は 1、最大値は 100 です。
output_confidence	*Boolean*	
output_rule_support	*Boolean*	
output_lift	*Boolean*	
output_condition_support	*Boolean*	
output_deployability	*Boolean*	
rules_to_display	upto all	出力テーブルに表示されるルールの最大数。
display_upto	*integer*	rules_to_display で upto を設定した場合は、出力テーブルに表示されるルールの数を指定します。最小値は 1 です。
field_transformations	*Boolean*	
records_summary	*Boolean*	
rule_statistics	*Boolean*	
most_frequent_values	*Boolean*	
most_frequent_fields	*Boolean*	
word_cloud	*Boolean*	
word_cloud_sort	Confidence Rulesupport Lift Conditionsupport Deployability	
word_cloud_display	*integer*	最小値は 1、最大値は 20 です。
max_predictions	*integer*	スコアに対する各入力に適用できるルールの最大数。

表 110. *associationrulesnode* プロパティー (続き)

associationrulesnode プロパティー	データ型	プロパティーの説明
criterion	Confidence Rulesupport Lift Conditionsupport Deployability	ルールの強度を判断するための尺度を選択します。
allow_repeats	*Boolean*	同じ予測を持つルールをスコア内に含めるかどうかを決定します。
check_input	NoPredictions Predictions NoCheck	

autoclassifiernode プロパティー

自動分類ノードは、2種類の結果 (yes/no、 churn/don't churn など) を生じる多くの異なるモデルを作成および比較し、与えられた分析への最善のアプローチを選ぶことができるようになります。多くのモデル作成アルゴリズムに対応し、希望する方法、各特定のオプション、そして結果を比較するための基準を選択することができます。このノードで、指定されたオプションに基づいてモデルのセットが生成され、指定された基準に基づいて最善の候補がランク付けされます。

例

```
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)
```

表 111. *autoclassifiernode* プロパティー:

autoclassifiernode プロパティー	値	プロパティーの説明
target	*field*	フラグ型対照の場合、自動分類ノードは 1 つの対象フィールドおよび 1 つ以上の入力フィールドを使用します。重みフィールドおよび度数フィールドも指定することができます。 詳しくは、トピック 187 ページの『一般的なモデル作成ノードのプロパティー』を参照してください。
ranking_measure	Accuracy Area_under_curve 利益 Lift Num_variables	

表 *111. autoclassifiernode* プロパティー *(*続き*)*:

autoclassifiernode プロパティー	値	プロパティーの説明
ranking_dataset	Training Test	
number_of_models	*integer*	モデル・ナゲットに含まれるモデルの数。1 と 100の間の整数を指定します。
calculate_variable_importance	*flag*	
enable_accuracy_limit	*flag*	
accuracy_limit	*integer*	0 と 100 の間の整数です。
enable_ area_under_curve _limit	*flag*	
area_under_curve_limit	*number*	0.0 と 1.0 の間の実数。
enable_profit_limit	*flag*	
profit_limit	*number*	1 以上の整数。
enable_lift_limit	*flag*	
lift_limit	*number*	1.0 を超える実数。
enable_number_of_variables_limit	*flag*	
number_of_variables_limit	*number*	1 以上の整数。
use_fixed_cost	*flag*	
fixed_cost	*number*	0.0 を超える実数。
variable_cost	*field*	
use_fixed_revenue	*flag*	
fixed_revenue	*number*	0.0 を超える実数。
variable_revenue	*field*	
use_fixed_weight	*flag*	
fixed_weight	*number*	0.0 を超える実数。
variable_weight	*field*	
lift_percentile	*number*	0 と 100 の間の整数です。
enable_model_build_time_limit	*flag*	
model_build_time_limit	*number*	個々のモデルのそれぞれを構築するためにかかる時間を制限するために分数を設定する整数。
enable_stop_after_time_limit	*flag*	
stop_after_time_limit	*number*	自動分類の実行のための全体経過時間を制限するために時間数を設定する実数。
enable_stop_after_valid_model_produced	*flag*	
use_costs	*flag*	
<algorithm>	*flag*	特定のアルゴリズムの使用の有効、無効を切り替えます。
<algorithm>.<property>	*string*	特定のアルゴリズムのプロパティー値を設定します。 詳しくは、トピック 195 ページの『アルゴリズム・プロパティーの設定』を参照してください。

アルゴリズム・プロパティーの設定

自動分類ノード、自動数値ノード、自動クラスター・ノードについては、ノードが使用する特定のアルゴリズムのプロパティーは、次の一般形式を使用して設定できます。

```
autonode.setKeyedPropertyValue(<algorithm>, <property>, <value>)
```

以下に例を示します。

```
node.setKeyedPropertyValue("neuralnetwork", "method", "MultilayerPerceptron")
```

自動分類ノードのアルゴリズム名は、cart、chaid、quest、c50、logreg、decisionlist、bayesnet、discriminant、svm および knn です。

自動数値ノードのアルゴリズム名は、cart、chaid、neuralnetwork、genlin、svm、regression、linear および knn です。

自動クラスター・ノードのアルゴリズム名は、twostep、k-means、および kohonen です。

プロパティー名は、各アルゴリズムノードのために文書化されている標準です。

ピリオドなどの句読点を含むアルゴリズム・プロパティーは、次のように一重引用符で囲む必要があります。

```
node.setKeyedPropertyValue("logreg", "tolerance", "1.0E-5")
```

次のように、複数の値をプロパティーに割り当てることもできます。

```
node.setKeyedPropertyValue("decisionlist", "search_direction", ["Up", "Down"])
```

特定のアルゴリズムの使用の有効、無効を切り替えるには、次のようにします。

```
node.setPropertyValue("chaid", True)
```

注: 自動分類ノードで特定のアルゴリズム・オプションが使用可能でない場合、または値の範囲ではなく、1 つの値だけを指定できるときは、標準の方法でノードにアクセスするときと同じ制限が、スクリプトにも適用されます。

---

autoclusternode プロパティー

自動クラスター・ノードは、同様の特性を持つレコードのグループを識別するクラスタリング・モデルを推定し、比較します。ノードは他の自動化モデル作成ノードと同じように動作し、複数の組み合わせのオプションを単一のモデル作成の実行で検証できます。モデルは、クラスター・モデルの有用性をフィルタリングおよびランク付けする基本的な指標を使用して比較し、特定のフィールドの重要度に基づいて指標を提供します。

例

```
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)
```

表 112. *autoclusternode* プロパティー

autoclusternode プロパティー	値	プロパティーの説明
evaluation	*field*	**注:** のみ。重要度の値を計算するフィールドを識別します。また、どれだけクラスターがフィールドの値を区別するか、どれだけ正確にモデルがこのフィールドを予測するかを識別するために使用することができます。
ranking_measure	Silhouette Num_clusters Size_smallest_cluster Size_largest_cluster Smallest_to_largest Importance	
ranking_dataset	Training Test	
summary_limit	*integer*	レポートに一覧するモデルの数。1 と 100 の間の整数を指定します。
enable_silhouette_limit	*flag*	
silhouette_limit	*integer*	0 と 100 の間の整数です。
enable_number_less_limit	*flag*	
number_less_limit	*number*	0.0 と 1.0 の間の実数。
enable_number_greater_limit	*flag*	
number_greater_limit	*number*	1 以上の整数。
enable_smallest_cluster_limit	*flag*	
smallest_cluster_units	Percentage Counts	
smallest_cluster_limit_percentage	*number*	
smallest_cluster_limit_count	*integer*	1 以上の整数。
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*	0 と 100 の間の整数です。
importance_limit_less_than	*number*	0 と 100 の間の整数です。
<algorithm>	*flag*	特定のアルゴリズムの使用の有効、無効を切り替えます。

表 112. *autoclusternode* プロパティー (続き)

autoclusternode プロパティー	値	プロパティーの説明
<algorithm>.<property>	*string*	特定のアルゴリズムのプロパティー値を設定します。 詳しくは、トピック 195 ページの『アルゴリズム・プロパティーの設定』を参照してください。

autonumericnode プロパティー

自動数値ノードでは、多くのさまざまな方法を使用し、連続する数値範囲の結果を求めてモデルを推定し比較します。このノードは、自動分類ノードと同じ方法で動作し、1 回のモデル作成のパスで、複数の組み合わせのオプションを使用し試すアルゴリズムを選択することができます。使用できるアルゴリズムには、ニューラル・ネットワーク、C&R Tree、CHAID、線型回帰、一般化線型回帰、サポート・ベクトル・マシン (SVM) が含まれています。モデルは、相関、相対エラー、または使用された変数の数に基づいて比較できます。

例

```
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)
```

表 113. *autonumericnode* プロパティー

autonumericnode プロパティー	値	プロパティーの説明
custom_fields	*flag*	真 (True) の場合、データ型ノード設定の代わりにカスタム・フィールド設定が使用されます。
target	*field*	自動数値ノードは 1 つの対象フィールドおよび 1 つ以上の入力フィールドを使用します。重みフィールドおよび度数フィールドも指定することができます。 詳しくは、トピック 187 ページの『一般的なモデル作成ノードのプロパティー』を参照してください。
inputs	*[field1 ... field2]*	
partition	*field*	
use_frequency	*flag*	
frequency_field	*field*	
use_weight	*flag*	
weight_field	*field*	
use_partitioned_data	*flag*	データ区分フィールドが定義されている場合、学習データだけがモデルの構築に使用されます。

表 113. *autonumericnode* プロパティー (続き)

autonumericnode プロパティー	値	プロパティーの説明
ranking_measure	Correlation NumberOfFields	
ranking_dataset	Test Training	
number_of_models	*integer*	モデル・ナゲットに含まれるモデルの数。1と 100の間の整数を指定します。
calculate_variable_importance	*flag*	
enable_correlation_limit	*flag*	
correlation_limit	*integer*	
enable_number_of_fields_limit	*flag*	
number_of_fields_limit	*integer*	
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*	特定のアルゴリズムの使用の有効、無効を切り替えます。
<algorithm>.<property>	*string*	特定のアルゴリズムのプロパティー値を設定します。 詳しくは、トピック 195 ページの『アルゴリズム・プロパティーの設定』を参照してください。

bayesnetnode プロパティー

ベイズ・ネットワーク・ノードを使用すると、観測された情報および記録された情報を実際の知識を組み合わせることによって確率モデルを作成し、発生の尤度を確立できます。ノードは主に分類に使用される Tree Augmented Naïve Bayes (TAN) および Markov Blanket ネットワークに焦点を当てています。

例

```
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")
```

表 *114. bayesnetnode* プロパティー

bayesnetnode プロパティー	値	プロパティーの説明
inputs	*[field1 ... fieldN]*	ベイズ・ネットワーク・モデルは単一の対象フィールドおよび 1 つ以上の入力フィールドを使用します。連続フィールドは自動的に分割されます。 詳しくは、トピック 187 ページの『一般的なモデル作成ノードのプロパティー』を参照してください。
continue_training_existing_model	*flag*	
structure_type	TAN MarkovBlanket	Bayesian ネットワークを構築時に使用する構造を選択します。
use_feature_selection	*flag*	
parameter_learning_method	Likelihood Bayes	親の値が認識されるノード間の条件付き確率テーブルを推定するために用いる方法を指定します。
mode	Expert Simple	
missing_values	*flag*	
all_probabilities	*flag*	
independence	Likelihood Pearson	2 つの変数のペアの観測がお互いに独立しているかどうかを評価するために用いる方法を指定します。
significance_level	*number*	独立性を判断するための分割値を指定します。
maximal_conditioning_set	*number*	独立性検定に使用する条件変数の最大数を指定します。
inputs_always_selected	*[field1 ... fieldN]*	ベイズ・ネットワーク構築時にデータセットのどのフィールドを常に使用するかを指定します。 **注:** 対象フィールドは必ず選択されます。
maximum_number_inputs	*number*	ベイズ・ネットワーク構築で使用する入力フィールドの最大数を指定します。
calculate_variable_importance	*flag*	
calculate_raw_propensities	*flag*	
calculate_adjusted_propensities	*flag*	
adjusted_propensity_partition	Test Validation	

buildr プロパティー

R 構築ノードを使用すると、IBM SPSS Modeler に展開されているモデル作成およびモデル・スコアリングを実行するためのカスタムの R スクリプトを入力できます。

例

```
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",fieldMeasure="",
fieldFormat="",fieldRole="")
modelerDataModel<-data.frame(modelerDataModel,var1)""")
```

表 *115. buildr* プロパティー：

buildr プロパティー	値	プロパティーの説明
build_syntax	*string*	モデル作成用の R スクリプト・シンタックス。
score_syntax	*string*	モデル・スコアリング用の R スクリプト・シンタックス。
convert_flags	StringsAndDoubles LogicalValues	フラグ型フィールドを変換するためのオプション。
convert_datetime	*flag*	日付形式または日付/時刻形式の変数を R の日付/時刻形式に変換するためのオプション。
convert_datetime_class	POSIXct POSIXlt	日付形式または日付/時刻形式の変数のうち、どの形式の変数を変換するかを指定するためのオプション。
convert_missing	*flag*	欠損値を R の NA 値に変換するためのオプション。
output_html	*flag*	R モデル・ナゲットのタブにグラフを表示するためのオプション。
output_text	*flag*	R モデル・ナゲットのタブに R コンソールのテキスト出力を書き込むためのオプション。

c50node プロパティー

C5.0 ノードは、ディシジョン・ツリーとルール・セットのどちらかを構築します。このモデルは、各レベルで最大の情報の対応をもたらすフィールドに基づいてサンプルを分割します。対象フィールドは、カテゴリーでなければなりません。複数の分割を 2 つ以上のサブグループに分割できます。

例

```
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]])
```

表 *116. c50node* プロパティー

c50node プロパティー	値	プロパティーの説明
target	*field*	C50 モデルは単一の対象フィールドおよび 1 つ以上の入力フィールドを使用します。重みフィールドも指定できます。 詳しくは、トピック 187 ページの『一般的なモデル作成ノードのプロパティー』を参照してください。
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	精度 (Accuracy) または一般化 (Generality) を選択。
expected_noise	*number*	
min_child_records	*number*	
pruning_severity	*number*	
use_costs	*flag*	
costs	*structured*	これは構造化されたプロパティーです。
use_winnowing	*flag*	
use_global_pruning	*flag*	デフォルトではオン (True)。
calculate_variable_importance	*flag*	
calculate_raw_propensities	*flag*	
calculate_adjusted_propensities	*flag*	
adjusted_propensity_partition	Test Validation	

carmanode プロパティー

CARMA モデルは、入力または対象フィールドを指定しなくても、データからルールのセットを抽出します。Apriori とは対照的に、CARMA ノードでは、前提条件サポートだけではなく、ルール・サポート (前提条件と結果の両方のサポート) を対象とした構築の設定が可能です。これは、生成されたルールをさまざまなアプリケーションで活用できることを意味します。例えば、この休暇シーズンに販売促進する項目を結果とする、商品またはサービス (前提条件) のリストを調べることができます。

例

```
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)
```

表 *117. carmanode* プロパティー

carmanode プロパティー	値	プロパティーの説明
inputs	*[field1 ... fieldn]*	CARMA モデルは対象フィールドでなく、入力フィールドのリストを使用します。重みフィールドおよび度数フィールドは使用しません。 詳しくは、トピック 187 ページの『一般的なモデル作成ノードのプロパティー』を参照してください。
id_field	*field*	モデル作成の ID フィールドとして使用するフィールド。
contiguous	*flag*	ID フィールドの ID が連続するかどうかを指定します。
use_transactional_data	*flag*	
content_field	*field*	
min_supp	*number(percent)*	前提条件範囲(サポート) ではなく、ルール範囲に関連します。デフォルト値は 20% です。
min_conf	*number(percent)*	デフォルト値は 20% です。
max_size	*number*	デフォルト値は 10 です。

表 117. *carmanode* プロパティー (続き)

carmanode プロパティー	値	プロパティーの説明
mode	Simple Expert	デフォルトは Simple です。
exclude_multiple	*flag*	複数の結果を持つルールを除外します。デフォルトは False です。
use_pruning	*flag*	デフォルトは False です。
pruning_value	*number*	デフォルトは 500 です。
vary_support	*flag*	
estimated_transactions	*integer*	
rules_without_antecedents	*flag*	

cartnode プロパティー

C&R Tree (分類と回帰ツリー) ノードは、ディシジョン・ツリーを生成し、将来の観測値を予測または分類できるようにします。この方法は再帰的なデータ区分を使用して学習レコードを複数のセグメントに分割し、各ステップで不純性を最小限に抑えます。ツリーのノードが「純粋」であると考えられるのは、ノード中にあるケースの 100% が、対象フィールドのある特定のカテゴリーに分類される場合です。対象フィールドおよび入力フィールドは、数値範囲またはカテゴリー (名義型、順序型、フラグ) が使用できます。すべての分岐は 2 分割です (2 つのサブグループのみ)。

例

```
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")
```

表 *118. cartnode* プロパティー

cartnode プロパティー	値	プロパティーの説明
target	*field*	C&R Tree モデルは 1 つの対象フィールドおよび 1 つ以上の入力フィールドを使用します。度数フィールドも指定できます。 詳しくは、トピック 187 ページの『一般的なモデル作成ノードのプロパティー』を参照してください。
continue_training_existing_model	*flag*	
objective	Standard Boosting Bagging psm	psm は非常に大きいデータセットに使用され、Server の接続が必要です。
model_output_type	Single InteractiveBuilder	
use_tree_directives	*flag*	
tree_directives	*string*	ツリーの成長のためのディレクティブ (式) を指定します。ディレクティブ (式) は、改行や引用符のエスケープ処理を回避するために、三重の引用符で囲むことができます。ディレクティブは、データやモデルリング・オプションの些細な変更に依存するため、他のデータセットに対しては一般化できません。
use_max_depth	Default Custom	
max_depth	*integer*	最大ツリー深さ (0 から 1000)。use_max_depth = Custom の場合にのみ使用します。
prune_tree	*flag*	オーバーフィットしないようにツリーを剪定します。
use_std_err	*flag*	リスクにおける最大差 (標準誤差) を使用します。
std_err_multiplier	*number*	最大差。
max_surrogates	*number*	最大代理変数。
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	*structured*	構造化プロパティー。
priors	Data Equal Custom	

表 118. *cartnode* プロパティー (続き)

cartnode プロパティー	値	プロパティーの説明
custom_priors	*structured*	構造化プロパティー。
adjust_priors	*flag*	
trails	*number*	ブーストまたはバグのコンポーネント・モデル数。
set_ensemble_method	Voting HighestProbability HighestMeanProbability	カテゴリー型対象のデフォルト結合ルール。
range_ensemble_method	Mean Median	連続型対象のデフォルト結合ルール。
large_boost	*flag*	特に大きなデータセットのブースティングを適用します。
min_impurity	*number*	
impurity_measure	Gini Twoing Ordered	
train_pct	*number*	オーバーフィット防止セット。
set_random_seed	*flag*	結果を再現オプション。
seed	*number*	
calculate_variable_importance	*flag*	
calculate_raw_propensities	*flag*	
calculate_adjusted_propensities	*flag*	
adjusted_propensity_partition	Test Validation	

chaidnode プロパティー

CHAID ノードはディシジョン・ツリーを生成し、カイ二乗統計値を使用して最適な分割を識別します。C&R ツリーおよび QUEST ノードと違って、CHAID は、非 2 分岐ツリーを生成できます。これは、ある分岐が 3 個以上のブランチを持つことを意味します。対象フィールドおよび入力フィールドは、数値範囲 (連続型) またはカテゴリーとなります。Exhaustive CHAID は CHAID の修正版で、可能性のある分割すべてを調べることで、よりよい結果を得られますが、計算時間も長くなります。

例

```
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)
```

表 *119. chaidnode* プロパティー

chaidnode プロパティー	値	プロパティーの説明
target	*field*	CHAID モデルは単一の対象フィールドおよび 1 つ以上の入力フィールドを使用します。度数フィールドも指定できます。 詳しくは、トピック 187 ページの『一般的なモデル作成ノードのプロパティー』を参照してください。
continue_training_existing_model	*flag*	
objective	Standard Boosting Bagging psm	psm は非常に大きいデータセットに使用され、Server の接続が必要です。
model_output_type	Single InteractiveBuilder	
use_tree_directives	*flag*	
tree_directives	*string*	
method	Chaid ExhaustiveChaid	
use_max_depth	Default Custom	
max_depth	*integer*	最大ツリー深さ (0 から 1000)。use_max_depth = Custom の場合にのみ使用します。
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	*structured*	構造化プロパティー。
trails	*number*	ブーストまたはバグのコンポーネント・モデル数。

表 119. *chaidnode* プロパティー (続き)

chaidnode プロパティー	値	プロパティーの説明
set_ensemble_method	Voting HighestProbability HighestMeanProbability	カテゴリー型対象のデフォルト結合ルール。
range_ensemble_method	Mean Median	連続型対象のデフォルト結合ルール。
large_boost	*flag*	特に大きなデータセットのブースティングを適用します。
split_alpha	*number*	分割の有意水準：
merge_alpha	*number*	結合の有意水準。
bonferroni_adjustment	*flag*	Bonferroni メソッドを使用して有意確率値を調整。
split_merged_categories	*flag*	マージしたカテゴリーの再分割を許可。
chi_square	Pearson LR	カイ 2 乗統計の計算に使用される方法 (Pearson または尤度比)
epsilon	*number*	期待されるセル度数の最小変化。
max_iterations	*number*	収束のための最大反復回数。
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 プロパティー

Cox 回帰ノードを使用すると、打ち切りレコードの存在下でイベントまでの時間のデータの生存モデルを構築します。モデルは、対象のイベントが入力変数の指定の値で指定の時間 (*t*) に発生する確率を予測する生存関数を作成します。

例

```
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)
```

表 *120. coxregnode* プロパティー

coxregnode プロパティー	値	プロパティーの説明
survival_time	*field*	Cox回帰モデルは 生存時間のある 1 つのフィールドを使用します。
target	*field*	Cox 回帰モデルは 1 つの対象フィールドおよび 1 つ以上の入力フィールドを使用します。 詳しくは、トピック 187 ページの『一般的なモデル作成ノードのプロパティー』を参照してください。
method	Enter Stepwise BackwardsStepwise	
groups	*field*	
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	
removal_criterion	LR Wald Conditional	
probability_entry	*number*	
probability_removal	*number*	
output_display	EachStep LastStep	
ci_enable	*flag*	

表 120. *coxregnode* プロパティー (続き)

coxregnode プロパティー	値	プロパティーの説明
ci_value	90 95 99	
correlation	*flag*	
display_baseline	*flag*	
survival	*flag*	
hazard	*flag*	
log_minus_log	*flag*	
one_minus_survival	*flag*	
separate_line	*field*	
value	*number*型 または *string*	フィールドに対して値の指定がない場合、デフォルト・オプションの「Mean」をそのフィールドで使用します。

decisionlistnode プロパティー

ディシジョン・リスト・ノードは、母集団に関連する与えられた 2 値の結果の高いもしくは低い尤度を示すサブグループまたはセグメントを識別します。例えば、離れる可能性の少ないもしくはキャンペーンに好意的に答える可能性のある顧客を探すことができます。顧客区分を追加し、結果を比較するために他のモデルを並べて表示することによって、ビジネスに関する知識をモデルに導入することができます。ディシジョン・リスト・モデルは、ルールのリストから構成され、各ルールには条件と結果が含まれます。ルールは順番に適用され、一致する最初のルールで、結果が決まります。

例

```
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)
```

表 121. *decisionlistnode* プロパティー

decisionlistnode プロパティー	値	プロパティーの説明
target	*field*	ディシジョン・リスト・モデルは 1 つの対象フィールドおよび 1 つ以上の入力フィールドを使用します。度数フィールドも指定できます。 詳しくは、トピック 187 ページの『一般的なモデル作成ノードのプロパティー』を参照してください。
model_output_type	Model InteractiveBuilder	
search_direction	Up Down	セグメントの検索に関連します。Up は、高い確率の検索、Down は低い確率の検索と同じです。

表 121. *decisionlistnode* プロパティー (続き)

decisionlistnode プロパティー	値	プロパティーの説明
target_value	*string*	指定しない場合は、フラグには真の値が想定されます。
max_rules	*integer*	残りを除外するセグメントの最大数
min_group_size	*integer*	最小セグメント・サイズ：
min_group_size_pct	*number*	最小セグメント・サイズ (パーセントとして)。
confidence_level	*number*	セグメント定義に追加するためにふさわしくするために、応答の尤度を向上するために入力フィールドが持つ最小しきい値。
max_segments_per_rule	*integer*	
mode	Simple Expert	
bin_method	EqualWidth EqualCount	
bin_count	*number*	
max_models_per_cycle	*integer*	リストの検索幅。
max_rules_per_cycle	*integer*	セグメント ルールの検索幅。
segment_growth	*number*	
include_missing	*flag*	
final_results_only	*flag*	
reuse_fields	*flag*	属性 (ルールに表示される入力フィールド)の再使用を許可します。
max_alternatives	*integer*	
calculate_raw_propensities	*flag*	
calculate_adjusted_propensities	*flag*	
adjusted_propensity_partition	Test Validation	

discriminantnode プロパティー

判別分析によって、ロジスティック回帰より厳密な仮説を立てることができますが、これらの仮説が一致した場合、ロジスティック回帰分析に対する様々な代替あるいは補足になります。

例

```
node = stream.create("discriminant", "My node")
node.setPropertyValue("target", "custcat")
node.setPropertyValue("use_partitioned_data", False)
node.setPropertyValue("method", "Stepwise")
```

表 122. *discriminantnode* プロパティー

discriminantnode プロパティー	値	プロパティーの説明
target	*field*	判別分析 モデルは単一の対象フィールドおよび 1 つ以上の入力フィールドを使用します。重みフィールドおよび度数フィールドは使用しません。 詳しくは、トピック 187 ページの『一般的なモデル作成ノードのプロパティー』を参照してください。
method	Enter Stepwise	
mode	Simple Expert	
prior_probabilities	AllEqual ComputeFromSizes	
covariance_matrix	WithinGroups SeparateGroups	
means	*flag*	「詳細出力」ダイアログ・ボックスの統計オプション
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*	「詳細出力」ダイアログ・ボックスの統計オプション
limit_to_first	*number*	デフォルト値は 10 です。
summary_table	*flag*	
leave_one_classification	*flag*	
combined_groups	*flag*	
separate_groups_covariance	*flag*	グループ別共分散行列オプション
territorial_map	*flag*	
combined_groups	*flag*	結合グループ散布図オプション
separate_groups	*flag*	グループ別散布図オプション
summary_of_steps	*flag*	
F_pairwise	*flag*	
stepwise_method	WilksLambda UnexplainedVariance MahalanobisDistance SmallestF RaosV	
V_to_enter	*number*	

表 122. *discriminantnode* プロパティー (続き)

discriminantnode プロパティー	値	プロパティーの説明
criteria	UseValue UseProbability	
F_value_entry	*number*	デフォルト値は 3.84 です。
F_value_removal	*number*	デフォルト値は 2.71 です。
probability_entry	*number*	デフォルト値は 0.05 です。
probability_removal	*number*	デフォルト値は 0.10 です。
calculate_variable_importance	*flag*	
calculate_raw_propensities	*flag*	
calculate_adjusted_propensities	*flag*	
adjusted_propensity_partition	Test Validation	

extensionmodelnode プロパティー

拡張モデル・ノードを使用すると、R スクリプトまたは Python for spark スクリプトを実行して、結果の作成およびスコアリングができます。

Python for Spark の例

```
#### 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)
```

R の例

```
#### script example for R
node.setPropertyValue("syntax_type", "R")
node.setPropertyValue("r_build_syntax", """modelerModel <- lm(modelerData$Na~modelerData$K,modelerData)
modelerDataModel
modelerModel
""")
```

表 123. *extensionmodelnode* プロパティ

extensionmodelnode プロパティ	値	プロパティーの説明
syntax_type	*R* *Python*	R または Python のどちらのスクリプトを実行するか指定します (R がデフォルトです)。
r_build_syntax	*string*	モデル作成用の R スクリプト・シンタックス。
r_score_syntax	*string*	モデル・スコアリング用の R スクリプト・シンタックス。
python_build_syntax	*string*	モデル作成用の Python スクリプト・シンタックス。
python_score_syntax	*string*	モデル・スコアリング用の Python スクリプト・シンタックス。
convert_flags	StringsAndDoubles LogicalValues	フラグ型フィールドを変換するためのオプション。
convert_missing	*flag*	欠損値を R の NA 値に変換するためのオプション。
convert_datetime	*flag*	日付形式または日付/時刻形式の変数を R の日付/時刻形式に変換するためのオプション。
convert_datetime_class	POSIXct POSIXlt	日付形式または日付/時刻形式の変数のうち、どの形式の変数を変換するかを指定するためのオプション。
output_html	*flag*	R モデル・ナゲットのタブにグラフを表示するためのオプション。
output_text	*flag*	R モデル・ナゲットのタブに R コンソールのテキスト出力を書き込むためのオプション。

factornode プロパティー

因子分析ノードには、データの複雑性を整理する強力なデータ分解手法が 2 種類あります。主成分分析 (PCA) : 入力フィールドの線型結合が検出されます。成分が互いに直交する (直角に交わる) 場合に、フィールドのセット全体の分散を把握するのに役立ちます。因子分析 : 一連の観測フィールド内の相関パターンを説明する基本因子が識別されます。どちらの手法でも、元のフィールド・セットの情報を効果的に要約する少数の派生フィールドの検出が目標です。

例

```
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)
```

表 *124. factornode* プロパティー

factornode プロパティー	値	プロパティーの説明
inputs	[*field1 ... fieldN*]	主成分分析/因子モデルは対象フィールドでなく、入力フィールドのリストを使用します。重みフィールドおよび度数フィールドは使用しません。 詳しくは、トピック 187 ページの『一般的なモデル作成ノードのプロパティー』を参照してください。
method	PC ULS GLS ML PAF Alpha (アルファ) Image	
mode	Simple (単純) Expert	
max_iterations	*number*	

表 124. *factornode* プロパティー (続き)

factornode プロパティー	値	プロパティーの説明
complete_records	*flag*	
matrix	Correlation Covariance (共分散)	
extract_factors	ByEigenvalues ByFactors	
min_eigenvalue	*number*	
max_factor	*number*	
rotation	None Varimax DirectOblimin Equamax Quartimax Promax	
delta	*number*	rotation で DirectOblimin を選択した場合、delta の値を指定できる。 値を指定しない場合は、delta のデフォルト値を使用。
kappa	*number*	rotation で Promax を選択した場合、kappa の値を指定できる。 値を指定しない場合は、kappa のデフォルト値を使用。
sort_values	*flag*	
hide_values	*flag*	
hide_below	*number*	

featureselectionnode プロパティー

フィールド選択ノードで、（欠損値の割合などの）諸基準に基づいて入力フィールドをスクリーニングして削除にかけ、指定した目標に相対的な残りの入力フィールドの重要度をランク付けします。例えば、数百の潜在的入力フィールドを含むデータセットがあるとして、患者予後のモデリングにはどれが役に立つのでしょう?

例

```
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)
```

変数選択モデルを作成して適用する詳細な例は、 5 ページの『スタンドアロン スクリプトの例 :変数選択モデルの生成』を参照してください。

表 125. *featureselectionnode* プロパティー

featureselectionnode プロパティー	値	プロパティーの説明
target	*field*	変数選択モデルは指定対象に関連した予測フィールドをランク付けします。重みフィールドおよび度数フィールドは使用しません。 詳しくは、トピック 187 ページの『一般的なモデル作成ノードのプロパティー』を参照してください。
screen_single_category	*flag*	True の場合、総レコード数に比べ同じカテゴリーに多くかたよったレコードを持つフィールドを選別します。
max_single_category	*number*	screen_single_category が True の場合に使用される閾値を指定します。
screen_missing_values	*flag*	True の場合、レコードの総数のパーセントで表すレコード数になるまで、多すぎる欠損値フィールドをスクリーニング (選別) します。
max_missing_values	*number*	
screen_num_categories	*flag*	True の場合、レコードの総数に対して多すぎるカテゴリーを減らす目的で、フィールドをスクリーニング (選別) します。
max_num_categories	*number*	
screen_std_dev	*flag*	True の場合、指定された最小値以下の標準偏差で、フィールドをスクリーニング (選別) します。
min_std_dev	*number*	
screen_coeff_of_var	*flag*	True の場合、指定された最小値以下の分散係数で、フィールドをスクリーニング (選別) します。
min_coeff_of_var	*number*	
criteria	Pearson Likelihood CramersV Lambda	カテゴリー対象に対するカテゴリー予測値のランク付けのときに、重要な値が基準とする測定単位を指定します。
unimportant_below	*number*	重要、境界、非重要として変数をランク付けするときに使用される閾値 *p* を指定します。0.0 から 1.0 の値を指定します。
important_above	*number*	0.0 から 1.0 の値を指定します。
unimportant_label	*string*	非重要ランクのラベルを指定します。
marginal_label	*string*	
important_label	*string*	

表 125. *featureselectionnode* プロパティー (続き)

featureselectionnode プロパティー	値	プロパティーの説明
selection_mode	ImportanceLevel ImportanceValue TopN	
select_important	*flag*	selection_mode が ImportanceLevel に設定されているときに、重要なフィールドを選択するかどうかを指定します。
select_marginal	*flag*	selection_mode が ImportanceLevel に設定されているときに、境界フィールドを選択するかどうかを指定します。
select_unimportant	*flag*	selection_mode が ImportanceLevel に設定されているときに、重要でないフィールドを選択するかどうかを指定します。
importance_value	*number*	selection_mode が ImportanceValue に設定されているときに、使用する分割値を指定します。0 から 100 の値を指定します。
top_n	*integer*	selection_mode が TopN に設定されているときに、使用する分割値を指定します。0 から 1000 の値を指定します。

genlinnode プロパティー

一般化線型モデルは、指定したリンク関数によって従属変数が因子および共変量と線型関係になるよう、一般線型モデルを拡張したものです。さらにこのモデルでは、非正規分布の従属変数を使用することができます。 線型回帰、ロジスティック回帰、カウント・データに関するログ線型モデル、そして区間打切り生存モデルなど、統計モデルの機能性が数多く含まれています。

例

```
node = stream.create("genlin", "My node")
node.setPropertyValue("model_type", "MainAndAllTwoWayEffects")
node.setPropertyValue("offset_type", "Variable")
node.setPropertyValue("offset_field", "Claimant")
```

表 126. *genlinnode* プロパティー

genlinnode プロパティー	値	プロパティーの説明
target	*field*	一般化線型モデルは、名義型またはフラグ型の 1 つの対象フィールドおよび 1 つ以上の入力フィールドが必要です。重みフィールドも指定できます。 詳しくは、トピック 187 ページの『一般的なモデル作成ノードのプロパティー』を参照してください。
use_weight	*flag*	
weight_field	*field*	フィールドのデータ型は連続型だけです。

表 126. *genlinnode* プロパティー (続き)

genlinnode プロパティー	値	プロパティーの説明
target_represents_trials	*flag*	
trials_type	Variable FixedValue	
trials_field	*field*	フィールドのデータ型はフラグ型または順序型です。
trials_number	*number*	デフォルト値は 10 です。
model_type	MainEffects MainAndAllTwoWayEffects	
offset_type	Variable FixedValue	
offset_field	*field*	フィールドのデータ型は連続型だけです。
offset_value	*number*	実数である必要があります。
base_category	Last First	
include_intercept	*flag*	
mode	Simple Expert	
distribution	BINOMIAL GAMMA IGAUSS NEGBIN NORMAL POISSON TWEEDIE MULTINOMIAL	IGAUSS: 逆ガウス。 NEGBIN: 負の 2 項分布。
negbin_para_type	Specify Estimate	
negbin_parameter	*number*	デフォルト値は 1 で、負でない実数を含む必要があります。
tweedie_parameter	*number*	

表 126. *genlinnode* プロパティー (続き)

genlinnode プロパティー	値	プロパティーの説明
link_function	IDENTITY CLOGLOG LOG LOGC LOGIT NEGBIN NLOGLOG ODDSPOWER PROBIT POWER CUMCAUCHIT CUMCLOGLOG CUMLOGIT CUMNLOGLOG CUMPROBIT	CLOGLOG: 補ログ・マイナス・ログ。 LOGC: 補対数。 NEGBIN: 負の 2 項分布。 NLOGLOG: 負ログ・マイナス・ログ。 CUMCAUCHIT: 累積コーチット。 CUMCLOGLOG: 累積補ログ・マイナス・ログ。 CUMLOGIT: 累積ロジット。 CUMNLOGLOG: 累積負ログ・マイナス・ログ。 CUMPROBIT: 累積プロビット。
power	*number*	値は 0 でない実数である必要があります。
method	Hybrid Fisher NewtonRaphson	
max_fisher_iterations	*number*	デフォルト値は 1 です。正の整数値だけが使用できます。
scale_method	MaxLikelihoodEstimate Deviance PearsonChiSquare FixedValue	
scale_value	*number*	デフォルト値は 1 です。0 を超える必要があります。
covariance_matrix	ModelEstimator RobustEstimator	
max_iterations	*number*	デフォルト値は 100 です。0 以上の整数だけを使用できます。
max_step_halving	*number*	デフォルト値は 5 です。正の整数値だけが使用できます。
check_separation	*flag*	
start_iteration	*number*	デフォルト値は 20 です。正の整数値だけが使用できます。
estimates_change	*flag*	
estimates_change_min	*number*	デフォルト値は 1E-006 です。正の数値だけが使用できます。
estimates_change_type	Absolute Relative	
loglikelihood_change	*flag*	
loglikelihood_change_min	*number*	正の数値だけが使用できます。
loglikelihood_change_type	Absolute Relative	

表 126. *genlinnode* プロパティー (続き)

genlinnode プロパティー	値	プロパティーの説明
hessian_convergence	*flag*	
hessian_convergence_min	*number*	正の数値だけが使用できます。
hessian_convergence_type	Absolute Relative	
case_summary	*flag*	
contrast_matrices	*flag*	
descriptive_statistics	*flag*	
estimable_functions	*flag*	
model_info	*flag*	
iteration_history	*flag*	
goodness_of_fit	*flag*	
print_interval	*number*	デフォルト値は 1 です。正の整数である必要があります。
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*	デフォルト値は 0.0001 です。
confidence_interval	*number*	デフォルト値は 95 です。
loglikelihood_function	Full Kernel	
singularity_tolerance	1E-007 1E-008 1E-009 1E-010 1E-011 1E-012	
value_order	Ascending Descending DataOrder	
calculate_variable_importance	*flag*	
calculate_raw_propensities	*flag*	
calculate_adjusted_propensities	*flag*	

表 126. genlinnode プロパティー (続き)

genlinnode プロパティー	値	プロパティーの説明
adjusted_propensity_partition	Test Validation	

glmmnode プロパティー

一般化線型混合モデル (GLMN) は線型モデルを拡張したため、対象が非正規分布となる場合があり、指定されたリンク関数を介して因子および共変量に線形に関連し、観測が相関できるようになりました。一般化線型混合モデルには、単純な線型回帰から、非正規分布の縦断的データを取り扱う複雑なマルチレベル・モデルまで、さまざまなモデルがあります。

表 127. glmmnode プロパティー:

glmmnode プロパティー	値	プロパティーの説明
residual_subject_spec	*structured*	指定したカテゴリー型フィールドの組み合わせにより、データセット内の被験者が一意に定義されることが必要です。
repeated_measures	*structured*	反復する観察の特定に使用されるフィールド。
residual_group_spec	[*field1 ... fieldN*]	反復効果共変量パラメーターの独立セットを定義するフィールド。
residual_covariance_type	Diagonal AR1 ARMA11 COMPOUND_SYMMETRY IDENTITY TOEPLITZ UNSTRUCTURED VARIANCE_COMPONENTS	残差の共変量構造を指定します。
custom_target	*flag*	上流のノードで定義された対象を使用するか (false) または target_field によって指定されたカスタム対象を使用するか (true) を定義します。
target_field	*field*	custom_target が true の場合対象として使用するフィールド。
use_trials	*flag*	試行回数を指定する追加フィールド又は値を、対象フィールドが一連の試行が発生する様々なイベントである場合に使用するかどうかを示します。デフォルトは false です。
use_field_or_value	field Value	フィールドまたは値を使用して試行回数を指定するかどうかを示します。
trials_field	*field*	試行回数の指定に使用するフィールド。
trials_value	*integer*	試行回数の指定に使用する値。指定する場合、最小値は 1 です。

表 127. *glmmnode* プロパティー (続き):

glmmnode プロパティー	値	プロパティーの説明
use_custom_target_reference	*flag*	カスタム参照カテゴリーをカテゴリー型対象に使用するかどうかを示します。デフォルトは false です。
target_reference_value	*string*	use_custom_target_reference が true の場合使用する参照カテゴリー。
dist_link_combination	Nominal (名義) Logit GammaLog BinomialLogit PoissonLog BinomialProbit NegbinLog BinomialLogC Custom	対象の値の分布に関する一般モデル。Custom を選択して、target_distribution で提供されたリストから分布を指定します。
target_distribution	Normal Binomial Multinomial Gamma (ガンマ) Inverse NegativeBinomial Poisson (ポワソン))	dist_link_combination が Custom の場合の対象の値の分布。
link_function_type	Identity LogC Log CLOGLOG Logit NLOGLOG PROBIT POWER CAUCHIT	対象値を予測値に関連付けるリンク関数。target_distribution が Binomial の場合、リストされているどのリンク関数でも使用できます。target_distribution が Multinomial の場合、CLOGLOG、CAUCHIT、LOGIT、NLOGLOG、または PROBIT を使用できます。target_distribution が Binomial 以外および Multinomial 以外の場合、IDENTITY、LOG、または POWER を使用できます。
link_function_param	*number*	使用するリンク関数パラメーター値。normal_link_function または link_function_type が POWER の場合のみ適用されます。
use_predefined_inputs	*flag*	固定効果フィールドを入力フィールドとして上流で定義されたフィールドとするか (true) fixed_effects_list のフィールドとするか (false) を指定します。デフォルトは false です。
fixed_effects_list	*structured*	use_predefined_inputs が false の場合、固定効果フィールドとして使用する入力フィールドを指定します。

表 127. *glmmnode* プロパティー (続き):

glmmnode プロパティー	値	プロパティーの説明
use_intercept	*flag*	true (デフォルト) の場合、モデルに定数項を含みます。
random_effects_list	*structured*	ランダム効果として指定するフィールドのリスト。
regression_weight_field	*field*	分析の重みフィールドとして使用するフィールド。
use_offset	None offset_value offset_field	オフセットを指定する方法を示します。値 None は、オフセットが使用されないことを意味します。
offset_value	*number*	use_offset が offset_value の場合オフセットに使用する値。
offset_field	*field*	use_offset が offset_field の場合オフセット値に使用する値。
target_category_order	Ascending Descending Data	カテゴリー型対象のソート順。値 Data は、データ内のソート順を使用するよう指定します。デフォルトは Ascending です。
inputs_category_order	Ascending Descending Data	Sorting order for categorical predictors. 値 Data は、データ内のソート順を使用するよう指定します。デフォルトは Ascending です。
max_iterations	*integer*	アルゴリズムで実行される反復の最大回数です。負の数ではない整数。デフォルト値は 100 です。
confidence_level	*integer*	モデル係数の区間推定の計算に使用する確信度。 負の数ではない整数。最小値は 100、デフォルト値は 95 です。
degrees_of_freedom_method	Fixed Varied	自由度が有意性検定に計算される方法を指定します。
test_fixed_effects_coeffecients	Model Robust	パラメーター推定共変量マトリックスを計算する方法。
use_p_converge	*flag*	パラメーター収束のオプション。
p_converge	*number*	空白または任意の正の値。
p_converge_type	絶対値 Relative	
use_l_converge	*flag*	対数尤度収束のオプション。
l_converge	*number*	空白または任意の正の値。
l_converge_type	絶対値 Relative	
use_h_converge	*flag*	Hessian 収束のオプション。
h_converge	*number*	空白または任意の正の値。
h_converge_type	絶対値 Relative	
max_fisher_steps	*integer*	
singularity_tolerance	*number*	

表 127. *glmmnode* プロパティー (続き):

glmmnode プロパティー	値	プロパティーの説明
use_model_name	*flag*	モデルのカスタム名を使用するか (true) システムによって生成された名前を使用するか (false) を指定します。デフォルトは false です。
model_name	*string*	use_model_name が true のときに、使用するモデルを指定します。
confidence	onProbability onIncrease	スコアリングの確信度を計算する基準 (最も高い予測確率、または最も高い予測確率と 2 番目に高い予測確率との差)。
score_category_probabilities	*flag*	true の場合、カテゴリー型対象の予測確率を生成します。デフォルトは false です。
max_categories	*integer*	score_category_probabilities が true のときに、使用するカテゴリーの最大数を指定します。
score_propensity	*flag*	true の場合、フィールドの「true」の結果の確率を示すフラグ型対象フィールドの傾向スコアを生成します。
emeans	*structure*	固定効果リストの各カテゴリー型フィールドについて、推定周辺平均を生成するかどうかを指定します。
covariance_list	*structure*	固定効果リストの各カテゴリー型フィールドについて、推定周辺平均を計算する場合に平均値を使用するかカスタム値を使用するかを指定します。
mean_scale	Original Transformed (変換)	対象の元の尺度に基づいて (デフォルト)、またはリンク関数変換に基づいて推定周辺平均を計算するかどうかを指定します。
comparison_adjustment_method	LSD SEQBONFERRONI SEQSIDAK	複数の対比で仮定検定を実行する場合に使用する調整方法。

gle プロパティー

GLE は、対象を非正規分布とできるように線型モデルを拡張したものであり、指定されたリンク関数を介して因子および共変量に線形に関連し、観測が相関できるようになりました。一般化線型混合モデルには、単純な線型回帰から、非正規分布の縦断的データを取り扱う複雑なマルチレベル・モデルまで、さまざまなモデルがあります。

表 128. *gle* プロパティー

gle プロパティー	値	プロパティーの説明
custom_target	*flag*	上流のノードで定義された対象を使用するか (false) または target_field によって指定されたカスタム対象を使用するか (true) を定義します。

表 *128. gle* プロパティー *(*続き*)*

gle プロパティー	値	プロパティーの説明
target_field	*field*	custom_target が true の場合対象として使用するフィールド。
use_trials	*flag*	試行回数を指定する追加フィールド又は値を、対象フィールドが一連の試行が発生する様々なイベントである場合に使用するかどうかを示します。デフォルトは false です。
use_trials_field_or_value	field Value	フィールドまたは値を使用して試行回数を指定するかどうかを示します。
trials_field	*field*	試行回数の指定に使用するフィールド。
trials_value	*integer*	試行回数の指定に使用する値。指定する場合、最小値は 1 です。
use_custom_target_reference	*flag*	カスタム参照カテゴリーをカテゴリー型対象に使用するかどうかを示します。デフォルトは false です。
target_reference_value	*string*	use_custom_target_reference が true の場合使用する参照カテゴリー。
dist_link_combination	NormalIdentity GammaLog PoissonLog NegbinLog TweedieIdentity NominalLogit BinomialLogit BinomialProbit BinomialLogC CUSTOM	対象の値の分布に関する一般モデル。target_distribution で提供されるリストから分布を選択するには CUSTOM を選択します。
target_distribution	Normal Binomial Multinomial Gamma (ガンマ) INVERSE_GAUSS NEG_BINOMIAL Poisson TWEEDIE UNKNOWN	dist_link_combination が Custom の場合の対象の値の分布。

表 *128. gle* プロパティー *(*続き*)*

gle プロパティー	値	プロパティーの説明
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	対象値を予測値に関連付けるリンク関数。 target_distribution が Binomial の場合、以下を使用できます。 UNKNOWN IDENTITY LOG LOGIT PROBIT COMPL_LOG_LOG POWER LOG_COMPL NEG_LOG_LOG ODDS_POWER target_distribution が NEG_BINOMIAL の場合、以下を使用できます。 NEG_BINOMIAL. target_distribution が UNKNOWN の場合、以下を使用できます。 GEN_LOGIT CUMUL_LOGIT CUMUL_PROBIT CUMUL_COMPL_LOG_LOG CUMUL_NEG_LOG_LOG CUMUL_CAUCHIT
link_function_param	*number*	使用する Tweedie パラメータ。 normal_link_function または link_function_type が POWER の場合のみ適用されます。
tweedie_param	*number*	使用するリンク関数パラメーター値。 dist_link_combination が TweedieIdentity に設定されているか、または link_function_type が TWEEDIE の場合にのみ適用できます。
use_predefined_inputs	*flag*	モデル効果フィールドを入力フィールドとして上流で定義されたフィールドとするか (true) fixed_effects_list のフィールドとするか (false) を指定します。
model_effects_list	*structured*	use_predefined_inputs が false の場合、モデル効果フィールドとして使用する入力フィールドを指定します。
use_intercept	*flag*	true (デフォルト) の場合、モデルに定数項を含みます。
regression_weight_field	*field*	分析の重みフィールドとして使用するフィールド。
use_offset	None Value 変数	オフセットを指定する方法を示します。値 None は、オフセットが使用されないことを意味します。

表 128. gle プロパティー (続き)

gle プロパティー	値	プロパティーの説明
offset_value	*number*	use_offset が offset_value の場合オフセットに使用する値。
offset_field	*field*	use_offset が offset_field の場合オフセット値に使用する値。
target_category_order	Ascending Descending	カテゴリー型対象のソート順。デフォルトは Ascending です。
inputs_category_order	Ascending Descending	Sorting order for categorical predictors. デフォルトは Ascending です。
max_iterations	*integer*	アルゴリズムで実行される反復の最大回数です。負の数ではない整数。デフォルト値は 100 です。
confidence_level	*number*	モデル係数の区間推定の計算に使用する確信度。負の数ではない整数。最小値は 100、デフォルト値は 95 です。
test_fixed_effects_coeffecients	Model 頑健	パラメーター推定共変量マトリックスを計算する方法。
detect_outliers	*flag*	true の場合、アルゴリズムで、多項分布を除くすべての分布に対する影響がある外れ値を検出します。
conduct_trend_analysis	*flag*	true の場合、アルゴリズムで散布図のトレンド分析を実行します。
estimation_method	FISHER_SCORING NEWTON_RAPHSON HYBRID	最尤法推定アルゴリズムを指定します。
max_fisher_iterations	*integer*	FISHER_SCORING estimation_method を使用している場合の、最大反復回数。最小値は 0、最大値は 20 です。
scale_parameter_method	MLE FIXED DEVIANCE PEARSON_CHISQUARE	スケール パラメータの推定に使用する方法を指定します。
scale_value	*number*	scale_parameter_method が Fixed に設定されている場合にのみ使用できます。
negative_binomial_method	MLE FIXED	負の二項分布補助パラメータの推定の使用する方法を指定します。
negative_binomial_value	*number*	negative_binomial_method が Fixed に設定されている場合にのみ使用できます。
use_p_converge	*flag*	パラメーター収束のオプション。
p_converge	*number*	空白または任意の正の値。
p_converge_type	*flag*	True = 絶対値、False = 相対値
use_l_converge	*flag*	対数尤度収束のオプション。
l_converge	*number*	空白または任意の正の値。
l_converge_type	*flag*	True = 絶対値、False = 相対値
use_h_converge	*flag*	Hessian 収束のオプション。
h_converge	*number*	空白または任意の正の値。

表 *128. gle* プロパティー *(*続き*)*

gle プロパティー	値	プロパティーの説明
h_converge_type	*flag*	True = 絶対値、False = 相対値
max_iterations	*integer*	アルゴリズムで実行される反復の最大回数です。負の数ではない整数。デフォルト値は 100 です。
sing_tolerance	*integer*	
use_model_selection	*flag*	パラメータしきい値とモデルの選択方法コントロールを有効にします。
method	LASSO ELASTIC_NET FORWARD_STEPWISE RIDGE	モデルの選択方法、または Ridge を使用している場合は正規化方法を決定します。
detect_two_way_interactions	*flag*	True の場合、モデルにより入力フィールド間の双方向交互作用が自動的に検出されます。 このコントロールは、モデルが主効果のみ (ユーザーが高次元効果を作成していない) であり、かつ選択された method が Forward Stepwise、Lasso、または Elastic Net の場合にのみ有効にしてください。
automatic_penalty_params	*flag*	モデル選択の method が Lasso または Elastic Net の場合のみ使用可能です。 この機能を使用して、Lasso または Elastic Net 変数選択方法に関連付けられたペナルティ パラメータを入力します。 True の場合、デフォルト値が使用されます。 False の場合、ペナルティ パラメータが有効になり、カスタム値を入力できます。
lasso_penalty_param	*number*	モデル選択の method が Lasso または Elastic Net であり、automatic_penalty_params が False の場合にのみ使用できます。Lasso のペナルティ パラメータを指定します。
elastic_net_penalty_param1	*number*	モデル選択の method が Lasso または Elastic Net であり、automatic_penalty_params が False の場合にのみ使用できます。Elastic Net パラメータ 1 のペナルティ パラメータを指定します。
elastic_net_penalty_param2	*number*	モデル選択の method が Lasso または Elastic Net であり、automatic_penalty_params が False の場合にのみ使用できます。Elastic Net パラメータ 2 のペナルティ パラメータを指定します。
probability_entry	*number*	選択された method が Forward Stepwise の場合にのみ使用できます。効果の包含に関する F 統計基準の有意水準を指定します。
probability_removal	*number*	選択された method が Forward Stepwise の場合にのみ使用できます。効果の除去に関する F 統計基準の有意水準を指定します。

表 128. *gle* プロパティー (続き)

gle プロパティー	値	プロパティーの説明
use_max_effects	*flag*	選択された method が Forward Stepwise の場合にのみ使用できます。 max_effects コントロールを有効にします。 False の場合、包含する効果のデフォルト数が、モデルに提供される効果の総数から切片を引いたものと等しくなければなりません。
max_effects	*integer*	変数増加ステップワイズ法作成方法を使用する場合の効果の最大数を指定します。
use_max_steps	*flag*	max_steps コントロールを有効にします。 False の場合、ステップのデフォルト数は、モデルに提供された効果の数から切片を除外したものの 3 倍と等しくなければなりません。
max_steps	*integer*	変数増加ステップワイズ法作成 method を使用する際に取るステップの最大数を指定します。
use_model_name	*flag*	モデルのカスタム名を使用するか (true) システムによって生成された名前を使用するか (false) を指定します。デフォルトは false です。
model_name	*string*	use_model_name が true のときに、使用するモデルを指定します。
usePI	*flag*	true の場合、予測変数の重要度が計算されます。

kmeansnode プロパティー

K-Means ノードで、データ・セットが異なるグループ (つまりクラスター) へ、クラスタリングされます。この方法で、固定数のクラスターを定義し、クラスターにレコードを繰り返し割り当てて、これ以上調整してもモデルが改善されなくなるまで、クラスターの中心を調整します。*K*-means では、結果を予測するのではなく、入力フィールドのセット内のパターンを明らかにするために、「非監視学習」として知られるプロセスが使用されます。

例

```
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)
```

表 129. *kmeansnode* プロパティー

kmeansnode プロパティー	値	プロパティーの説明
inputs	[*field1 ... fieldN*]	K-means モデルは入力フィールドのセットでクラスター分析を行いますが、対象フィールドは使用しません。重みフィールドおよび度数フィールドは使用しません。 詳しくは、トピック 187 ページの『一般的なモデル作成ノードのプロパティー』を参照してください。
num_clusters	*number*	
gen_distance	*flag*	
cluster_label	String Number	
label_prefix	*string*	
mode	Simple (単純) Expert	
stop_on	Default Custom	
max_iterations	*number*	
tolerance	*number*	
encoding_value	*number*	
optimize	Speed Memory	モデル作成が速度とメモリーのどちらにより最適化されるかを指定します。

knnnode プロパティー

k が整数である場合、*k* 最近傍 (KNN) ノードは、新しいケースを、予測領域の新しいケースに最も近い *k* 個のオブジェクトのカテゴリーまたは値と関連付けます。類似したケースはお互いに近く、類似していないケースはお互いに離れています。

例

```
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)
```

表 130. *knnnode* プロパティー

knnnode プロパティー	値	プロパティーの説明
analysis	PredictTarget IdentifyNeighbors	

表 130. *knnnode* プロパティー (続き)

knnnode プロパティー	値	プロパティーの説明
objective	Balance Speed Accuracy Custom	
normalize_ranges	*flag*	
use_case_labels	*flag*	次のオプションを有効化するチェック・ボックス。
case_labels_field	*field*	
identify_focal_cases	*flag*	次のオプションを有効化するチェック・ボックス。
focal_cases_field	*field*	
automatic_k_selection	*flag*	
fixed_k	*integer*	automatic_k_selectio が False の場合にのみ有効です。
minimum_k	*integer*	automatic_k_selectio が True の場合にのみ有効です。
maximum_k	*integer*	
distance_computation	Euclidean CityBlock	
weight_by_importance	*flag*	
range_predictions	Mean Median	
perform_feature_selection	*flag*	
forced_entry_inputs	[*field1 ... fieldN*]	
stop_on_error_ratio	*flag*	
number_to_select	*integer*	
minimum_change	*number*	
validation_fold_assign_by_field	*flag*	
number_of_folds	*integer*	validation_fold_assign_by_field が False の場合にのみ有効です。
set_random_seed	*flag*	
random_seed	*number*	
folds_field	*field*	validation_fold_assign_by_field が True の場合にのみ有効です。
all_probabilities	*flag*	
save_distances	*flag*	
calculate_raw_propensities	*flag*	
calculate_adjusted_propensities	*flag*	
adjusted_propensity_partition	Test Validation	

kohonennode プロパティー

Kohonen ノードは、ニューラル・ネットワークの一種であり、データ・セットをクラスター化して異なるグループを形成する目的で使用できます。ネットワークの学習が完了すると、類似のレコードは出力マップで互い近くに表示され、違いの大きいレコードほど離れたところに表示されます。強度の高いユニットを識別するために生成されたモデル内で、各ユニットが獲得した観察の数値を調べることができます。これは、適切なクラスター数についてのヒントになる場合があります。

例

```
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)
```

表 131. *kohonennode* プロパティー

kohonennode プロパティー	値	プロパティーの説明
inputs	[*field1 ... fieldN*]	Kohonen モデルは対象フィールドでなく、入力フィールドのリストを使用します。度数フィールドおよび重みフィールドは使用しません。 詳しくは、トピック 187 ページの『一般的なモデル作成ノードのプロパティー』を参照してください。
continue	*flag*	
show_feedback	*flag*	
stop_on	Default Time	
time	*number*	
optimize	Speed Memory	モデル作成が速度とメモリーのどちらにより最適化されるかを指定します。
cluster_label	*flag*	
mode	Simple (単純) Expert	
width	*number*	
length	*number*	

表 131. *kohonennode* プロパティー (続き)

kohonennode プロパティー	値	プロパティーの説明
decay_style	Linear Exponential	
phase1_neighborhood	*number*	
phase1_eta	*number*	
phase1_cycles	*number*	
phase2_neighborhood	*number*	
phase2_eta	*number*	
phase2_cycles	*number*	

linearnode プロパティー

線型回帰モデルは、対象と 1 つまたは複数の予測値との線型の関係に基づいて連続型対象を予測します。

例

```
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")
```

表 132. *linearnode* プロパティー:

linearnode プロパティー	値	プロパティーの説明
target	*field*	1 つの対象フィールドを指定します。
inputs	[*field1 ... fieldN*]	モデルで使用される入力または入力または予測変数フィールド。
continue_training_existing_model	*flag*	
objective	Standard Bagging Boosting psm	psm は非常に大きいデータセットに使用され、Server の接続が必要です。
use_auto_data_preparation	*flag*	
confidence_level	*number*	
model_selection	ForwardStepwise BestSubsets None	

表 132. *linearnode* プロパティー (続き) :

linearnode プロパティー	値	プロパティーの説明
criteria_forward_stepwise	AICC Fstatistics AdjustedRSquare ASE	
probability_entry	*number*	
probability_removal	*number*	
use_max_effects	*flag*	
max_effects	*number*	
use_max_steps	*flag*	
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*	

linearasnode プロパティー

線型回帰モデルは、対象と 1 つまたは複数の予測値との線型の関係に基づいて連続型対象を予測します。

表 133. *linearasnode* プロパティー

linearasnode プロパティー	値	プロパティーの説明
target	*field*	1 つの対象フィールドを指定します。
inputs	[*field1 ... fieldN*]	モデルで使用される入力または入力または予測変数フィールド。
weight_field	*field*	モデルで使用される分析フィールド。

表 133. *linearasnode* プロパティー (続き)

linearasnode プロパティー	値	プロパティーの説明
custom_fields	*flag*	デフォルト値は TRUE です。
intercept	*flag*	デフォルト値は TRUE です。
detect_2way_interaction	*flag*	双方向交互作用を考慮するかどうか。デフォルト値は TRUE です。
cin	*number*	モデル係数の推定値を計算するために使用する確信度の区間。0 より大きく、100 より小さい値を指定します。デフォルト値は 95 です。
factor_order	ascending descending	カテゴリ型予測フィールドの並び順。デフォルト値は ascending です。
var_select_method	ForwardStepwise BestSubsets none	使用するモデルの選択方法。デフォルト値は ForwardStepwise です。
criteria_for_forward_stepwise	AICC Fstatistics AdjustedRSquare ASE	モデルに効果を加えるべきか、またはモデルから効果を削除するべきかを決定するときに使用する統計。デフォルト値は AdjustedRSquare です。
pin	*number*	ここに指定された pin しきい値未満の最小 p 値を持つ効果がモデルに追加されます。デフォルト値は 0.05 です。
pout	*number*	ここに指定された pout しきい値より大きい p 値を持つモデル内のすべての効果が削除されます。デフォルト値は 0.10 です。
use_custom_max_effects	*flag*	最終モデルで最大数の効果を使用するかどうか。デフォルト値は FALSE です。
max_effects	*number*	最終モデルで使用する効果の最大数。デフォルト値は 1 です。
use_custom_max_steps	*flag*	最大数のステップを使用するかどうか。デフォルト値は FALSE です。
max_steps	*number*	ステップワイズ アルゴリズムが停止する最大ステップ数。デフォルト値は 1 です。
criteria_for_best_subsets	AICC AdjustedRSquare ASE	使用する基準のモード。デフォルト値は AdjustedRSquare です。

logregnode プロパティー

ロジスティック回帰は、入力フィールドの値に基づいてレコードを分類する統計手法です。線型回帰と似ていますが、数値範囲ではなくカテゴリー対象フィールドを使用します。

Multinomial Example

```
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")
```

Binomial Example

```
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)
```

表 *134. logregnode* プロパティー：

logregnode プロパティー	値	プロパティーの説明
target	*field*	ロジスティック回帰モデルは 1 つの対象フィールドおよび 1 つ以上の入力フィールドを使用します。度数フィールドおよび重みフィールドは使用しません。 詳しくは、トピック 187 ページの『一般的なモデル作成ノードのプロパティー』を参照してください。
logistic_procedure	Binomial Multinomial	
include_constant	*flag*	
mode	Simple (単純) Expert	
method	Enter Stepwise Forwards Backwards BackwardsStepwise	
binomial_method	Enter Forwards Backwards	

表 *134. logregnode* プロパティー *(*続き*)*:

logregnode プロパティー	値	プロパティーの説明
model_type	MainEffects FullFactorial Custom	モデル・タイプとして FullFactorial が指定されている場合、ステップ手法が指定されたとしても、実行されません。その代わりに、強制投入法 (Enter) が使用されます。 モデル・タイプに Custom が設定されてもユーザー設定フィールド (custom fields) が指定されていない場合は、主効果モデルが構築されます。
custom_terms	[*[BP Sex][BP][Age]*]	
multinomial_base_category	*string*	参照カテゴリーの決定方法を指定します。
binomial_categorical_input	*string*	
binomial_input_contrast	Indicator Simple Difference Helmert Repeated Polynomial Deviation	コントラストを決定する方法を指定するカテゴリー入力用のキー・プロパティー。
binomial_input_category	First Last	参照カテゴリーを決定する方法を指定するカテゴリー入力用のキー・プロパティー。
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*	
entry_criterion	Score LR	
removal_criterion	LR Wald	
probability_entry	*number*	
probability_removal	*number*	
binomial_probability_entry	*number*	

表 134. *logregnode* プロパティー (続き):

logregnode プロパティー	値	プロパティーの説明
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	
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_iteration_history	*flag*	
binomial_classification_plots	*flag*	
binomial_ci_enable	*flag*	
binomial_ci	*number*	
binomial_residual	outliers all	

表 134. logregnode プロパティー (続き):

logregnode プロパティー	値	プロパティーの説明
binomial_residual_enable	*flag*	
binomial_outlier_threshold	*number*	
binomial_classification_cutoff	*number*	
binomial_removal_criterion	LR Wald Conditional	
calculate_variable_importance	*flag*	
calculate_raw_propensities	*flag*	

lsvmnode プロパティー

線型サポート・ベクター・マシン (LSVM) ノードを使用すると、オーバーフィットすることなく、データを 2 つのグループのいずれかに分類することができます。LSVM は線型であり、極めて多数のレコードを含むデータセットなど、広範なデータセットを処理することができます。

表 135. lsvmnode プロパティー

lsvmnode プロパティー	値	プロパティーの説明
intercept	*flag*	モデルに切片を含めます。デフォルト値は True です。
target_order	Ascending Descending	カテゴリー型対象のソート順を指定します。連続型対象では無視されます。デフォルトは Ascending です。
precision	*number*	対象フィールドの尺度が Continuous の場合にのみ使用されます。回帰の損失の感度に関連するパラメーターを指定します。最小値は 0 で最大値はありません。デフォルト値は 0.1 です。
exclude_missing_values	*flag*	True にすると、欠損値が 1 つでもある場合はレコードが除外されます。デフォルト値は False です。
penalty_function	L1 L2	使用するペナルティ関数のタイプを指定します。デフォルト値は L2 です。
lambda	*number*	ペナルティ (正規化) パラメーター。

表 135. *lsvmnode* プロパティー (続き)

lsvmnode プロパティー	値	プロパティーの説明
calculate_variable_importance	*flag*	重要度の適切な測定を生成するモデルの場合、このオプションにより、モデルの推定における各予測値の相対重要度を示すグラフが表示されます。一部のモデルでは、特に大規模データセットを処理する場合、変数の重要度の計算には長い時間がかかるため、一部のモデルではデフォルトでオフになっています。変数の重要度は、ディシジョン リスト モデルでは使用できません。

neuralnetnode プロパティー

重要: 機能が拡張された新しいバージョンのニューラル・ネットワーク・モデル作成ノードがこのリリースで使用できます。新しいバージョンについては次の項で説明します (*neuralnetwork*)。旧バージョンでモデルを作成およびスコアリングできますが、新しいバージョンを使用するようスクリプトを更新することをお勧めします。以下は旧バージョンの詳細です。

例

```
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)
```

表 136. *neuralnetnode* プロパティー

neuralnetnode プロパティー	値	プロパティーの説明
targets	[*field1 ... fieldN*]	ニューラル・ノードには、 1 つ以上の対象フィールドと 1 つ以上の入力フィールドが必要です。度数および重みフィールドは無視されます。 詳しくは、トピック 187 ページの『一般的なモデル作成ノードのプロパティー』を参照してください。

表 136. *neuralnetnode* プロパティー (続き)

neuralnetnode プロパティー	値	プロパティーの説明
method	Quick Dynamic Multiple Prune ExhaustivePrune RBFN	
prevent_overtrain	*flag*	
train_pct	*number*	
set_random_seed	*flag*	
random_seed	*number*	
mode	Simple (単純) Expert	
stop_on	Default Accuracy Cycles Time	停止モード。
accuracy	*number*	停止精度。
cycles	*number*	学習サイクル。
time	*number*	学習時間 (分)。
continue	*flag*	
show_feedback	*flag*	
binary_encode	*flag*	
use_last_model	*flag*	
gen_logfile	*flag*	
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*	

表 136. *neuralnetnode* プロパティー (続き)

neuralnetnode プロパティー	値	プロパティーの説明
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*	
r_eta	*number*	
optimize	Speed Memory	モデル作成が速度とメモリーのどちらにより最適化されるかを指定します。
calculate_variable_importance	*flag*	注 : 前回のリリースで使用した sensitivity_analysis プロパティーは、このプロパティーにより廃止されます。古いプロパティーはまだサポートされますが、calculate_variable_importance をお勧めします。
calculate_raw_propensities	*flag*	
calculate_adjusted_propensities	*flag*	
adjusted_propensity_partition	Test Validation	

neuralnetworknode プロパティー

ニューラル・ネットワーク・ノードは、人間の脳が情報を処理する方法を単純化したモデルを使用します。ニューラル・ネットワーク・ノードは、連係する多数の単純な処理単位をシミュレートします。処理単位は、ニューロンを抽象化したものと表現できます。ニューラル・ネットワークは強力な一般関数推定法であり、学習させたり、適用するには、最低限の統計学および数学の知識しか必要ありません。

例

```
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")
```

表 *137. neuralnetworknode* プロパティー

neuralnetworknode プロパティー	値	プロパティーの説明
targets	[*field1 ... fieldN*]	対象フィールドを指定します。
inputs	[*field1 ... fieldN*]	モデルで使用される入力または入力または予測変数フィールド。
splits	[*field1 ... fieldN*	分割モデル作成に使用する、フィールドを選択します。
use_partition	*flag*	区分フィールドが定義される場合、このオプションは学習データ区分からのデータのみがモデル構築に使用されるようにします。
continue	*flag*	既存モデルの学習を継続 :
objective	Standard Bagging Boosting psm	psm は非常に大きいデータセットに使用され、Server の接続が必要です。
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*	
missing_values	listwiseDeletion missingValueImputation	
use_model_name	*boolean*	

表 137. *neuralnetworknode* プロパティー (続き)

neuralnetworknode プロパティー	値	プロパティーの説明
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 プロパティー

QUEST ノードには、ディシジョン・ツリーの構築用に2 分岐の方法が用意されています。これは、大規模な C&R ツリー分析が必要とする処理時間を短縮すると同時に、より多くの分割を可能にする入力値が優先される分類ツリー内の傾向を低減するように設計されています。入力フィールドは、数値範囲 (連続型) にできますが、目標変数はカテゴリーでなければなりません。すべての分割は 2 分岐です。

例

```
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)
```

表 138. *questnode* プロパティー

questnode プロパティー	値	プロパティーの説明
target	*field*	QUEST モデルは単一の対象フィールドおよび 1 つ以上の入力フィールドを使用します。度数フィールドも指定できます。 詳しくは、トピック 187 ページの『一般的なモデル作成ノードのプロパティー』を参照してください。
continue_training_existing_model	*flag*	

表 138. *questnode* プロパティー (続き)

questnode プロパティー	値	プロパティーの説明
objective	Standard Boosting Bagging psm	psm は非常に大きいデータセットに使用され、Server の接続が必要です。
model_output_type	Single InteractiveBuilder	
use_tree_directives	*flag*	
tree_directives	*string*	
use_max_depth	Default Custom	
max_depth	*integer*	最大ツリー深さ (0 から 1000)。use_max_depth = Custom の場合にのみ使用します。
prune_tree	*flag*	オーバーフィットしないようにツリーを剪定します。
use_std_err	*flag*	リスクにおける最大差 (標準誤差) を使用します。
std_err_multiplier	*number*	最大差。
max_surrogates	*number*	最大代理変数。
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	*structured*	構造化プロパティー。
priors	Data Equal Custom	
custom_priors	*structured*	構造化プロパティー。
adjust_priors	*flag*	
trails	*number*	ブーストまたはバグのコンポーネント・モデル数。
set_ensemble_method	Voting HighestProbability HighestMeanProbability	カテゴリー型対象のデフォルト結合ルール。
range_ensemble_method	Mean Median	連続型対象のデフォルト結合ルール。
large_boost	*flag*	特に大きなデータセットのブースティングを適用します。
split_alpha	*number*	分割の有意水準：
train_pct	*number*	オーバーフィット防止セット。
set_random_seed	*flag*	結果を再現オプション。

表 138. *questnode* プロパティー (続き)

questnode プロパティー	値	プロパティーの説明
seed	*number*	
calculate_variable_importance	*flag*	
calculate_raw_propensities	*flag*	
calculate_adjusted_propensities	*flag*	
adjusted_propensity_partition	Test Validation	

randomtrees プロパティー

ランダム ツリー ノードは、既存の C&RT ノードと似ていますが、ビッグデータを処理して単一のツリーを作成することを目的に設計されており、結果のモデルが SPSS Modeler バージョン 17 で追加された出力ビューアに表示されます。ランダム ツリー ノードが生成するディシジョン ツリーを使用して、将来の観測値を予測または分類できます。この方法は再帰的なデータ区分を使用して学習レコードを複数のセグメントに分割し、各ステップで不純性を最小限に抑えます。ツリーのノードが純粋 であると考えられるのは、ノード中にあるケースの 100% が、対象フィールドのある特定のカテゴリーに分類される場合です。対象フィールドおよび入力フィールドは、数値範囲またはカテゴリー (名義型、順序型、フラグ) が使用できます。すべての分岐は 2 分割です (2 つのサブグループのみ)。

表 139. *randomtrees* プロパティー

randomtrees プロパティー	値	プロパティーの説明
target	*field*	ランダム ツリー ノードでは、モデルには単一の対象フィールドおよび 1 つ以上の入力フィールドが必要になります。度数フィールドも指定できます。 詳しくは、トピック 187 ページの『一般的なモデル作成ノードのプロパティー』を参照してください。
number_of_models	*integer*	アンサンブル・モデル構築の一環として構築されるモデルの数を決定します。
use_number_of_predictors	*flag*	number_of_predictors を使用するかどうかを決定します。
number_of_predictors	*integer*	分割モデルの構築時に使用する予測値の個数を指定します。
use_stop_rule_for_accuracy	*flag*	精度を向上できない場合にモデル構築を中止するかどうかを決定します。
sample_size	*number*	極めて大規模なデータ・セットを処理する際にパフォーマンスを向上させるには、この値を小さくします。

表 139. *randomtrees* プロパティー (続き)

randomtrees プロパティー	値	プロパティーの説明
handle_imbalanced_data	*flag*	モデルの対象が特定のフラグの結果であり、望ましくない結果に対する望まれる結果の比率が非常に小さい場合は、データは不均衡になり、モデルによって実行されるブートストラップ・サンプリングがモデルの精度に影響を与える可能性があります。不均衡なデータの処理を有効にすると、モデルが収集する望ましい結果の比率が高まり、より強固なモデルが生成されます。
use_weighted_sampling	*flag*	*False* の場合、各ノードの変数は、同じ確率で無作為に選択されます。*True* の場合、変数には重みが付けられ、それに応じて選択されます。
max_node_number	*integer*	個々のツリーで許容されるノードの最大数。次の分割でこの数を超えることが予想される場合、ツリーの成長は停止します。
max_depth	*integer*	ツリーの最大の深さ。これに達すると成長は停止します。
min_child_node_size	*integer*	親ノードの分割後に子ノードで許容されるレコードの最小数を決定します。子ノードに含まれることになるレコードの数がここで指定した数よりも少ない場合、親ノードは分割されません。
use_costs	*flag*	
costs	*structured*	構造化プロパティー。形式は、実際の値、予測された値、およびコスト (予測が正しくない場合) の 3 つの値のリストです。以下に例を示します。 tree.setPropertyValue("costs", [["drugA", "drugB", 3.0], ["drugX", "drugY", 4.0]])
default_cost_increase	none linear square custom	**注:** 順序型対象に対してのみ有効です。 コスト行列にデフォルト値を設定します。
max_pct_missing	*integer*	いずれかの入力の欠損値の割合がここで指定した値より大きい場合、その入力は除外されます。最小値は 0、最大値は 100 です。
exclude_single_cat_pct	*integer*	いずれかのカテゴリー値がここで指定したレコードの割合より高い場合、そのフィールド全体がモデル構築から除外されます。最小値は 1、最大値は 99 です。
max_category_number	*integer*	フィールド内のカテゴリー数がこの値を超える場合、そのフィールドはモデル構築から除外されます。最大値は 2 です。

表 139. *randomtrees* プロパティー (続き)

randomtrees プロパティー	値	プロパティーの説明
min_field_variation	*number*	連続型フィールドの変動係数がこの値より小さい場合、そのフィールドはモデル構築から除外されます。
num_bins	*integer*	データが連続型入力で構成される場合にのみ使用されます。入力に対して使用する等しいフリクエンシ ビンの数を設定します。オプションは 2、4、5、10、20、25、50、または 100 です。
topN	*integer*	報告するルールの数を指定します。デフォルト値は 50 で、最小値は 1、最大値は 1000 です。

regressionnode プロパティー

線型回帰は、データを要約する一般的な統計手法であり、予測された出力値と実際の出力値の違いを最小限にする直線または面を当てはめることにより予測を行います。

注: 今後のリリースでは、線型回帰ノードは線型ノードに置き換えられる予定になっています。今後、線型回帰には線型モデルを使用することをお勧めします。

例

```
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)
```

表 *140. regressionnode* プロパティー

regressionnode プロパティー	値	プロパティーの説明
target	*field*	回帰モデルは単一の対象フィールドおよび 1 つ以上の入力フィールドを使用します。 重みフィールドも指定できます。 詳しくは、トピック 187 ページの『一般的なモデル作成ノードのプロパティー』を参照してください。
method	Enter Stepwise Backwards Forwards	
include_constant	*flag*	
use_weight	*flag*	
weight_field	*field*	
mode	Simple (単純) 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	引数には二重引用符を使用します。
stepping_method	useP useF	useP : F 値確率を使用 useF: F 値を使用
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*	

表 140. *regressionnode* プロパティー (続き)

regressionnode プロパティー	値	プロパティーの説明
regression_coefficients	*flag*	
exclude_fields	*flag*	
durbin_watson	*flag*	
model_fit	*flag*	
r_squared_change	*flag*	
p_correlations	*flag*	
descriptives	*flag*	
calculate_variable_importance	*flag*	

sequencenode プロパティー

シーケンス・ノードで、シーケンシャルな、または時間経過が伴うデータ内のアソシエーション・ルールを検出します。予測可能な順序で起こる傾向にあるアイテム・セットのリストを、シーケンスと呼びます。例えば、顧客がひげそりとアフター・シェーブ ローションを購入した場合、その顧客は次の購入時にシェービング クリームを購入する可能性があります。シーケンス・ノードは CARMA アソシエーション・ルール・アルゴリズムに基づいているため、効率的な 2 段階通過法でシーケンスが検出されます。

例

```
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)
```

表 141. *sequencenode* プロパティー

sequencenode プロパティー	値	プロパティーの説明
id_field	*field*	シーケンス・モデルを作成するには、ID フィールドを指定する必要があります。さらにオプションで時間フィールドと 1 つ以上の内容フィールドを指定します。重みフィールドおよび度数フィールドは使用しません。 詳しくは、トピック 187 ページの『一般的なモデル作成ノードのプロパティー』を参照してください。
time_field	*field*	
use_time_field	*flag*	
content_fields	[*field1 ... fieldn*]	
contiguous	*flag*	
min_supp	*number*	
min_conf	*number*	
max_size	*number*	
max_predictions	*number*	
mode	Simple (単純) 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 プロパティー

SLRM (自己学習応答モデル) ノードを使用するとモデルを構築でき、単一または少数の新しいケースを使用して全データを使用するモデルの保持をすることなく、モデルの再見積もりを行うことができます。

例

```
node = stream.create("slrm", "My node")
node.setPropertyValue("target", "Offer")
node.setPropertyValue("target_response", "Response")
node.setPropertyValue("inputs", ["Cust_ID", "Age", "Ave_Bal"])
```

表 142. slrmnode プロパティー

slrmnode プロパティー	値	プロパティーの説明
target	*field*	対象フィールドは名義型またはフラグ型である必要があります。度数フィールドも指定できます。 詳しくは、トピック 187 ページの『一般的なモデル作成ノードのプロパティー』を参照してください。
target_response	*field*	フラグ型である必要があります。
continue_training_existing_model	*flag*	
target_field_values	*flag*	すべて使用:ソースのすべての値を使用します。 指定:必要な値を選択します。
target_field_values_specify	*[field1 ... fieldN]*	
include_model_assessment	*flag*	
model_assessment_random_seed	*number*	実数である必要があります。
model_assessment_sample_size	*number*	実数である必要があります。
model_assessment_iterations	*number*	反復数。
display_model_evaluation	*flag*	
max_predictions	*number*	
randomization	*number*	
scoring_random_seed	*number*	
sort	Ascending Descending	高いスコアまたは低いスコアのどちらを持つオファーが最初に表示されるかを指定します。
model_reliability	*flag*	
calculate_variable_importance	*flag*	

statisticsmodelnode プロパティー

Statistics モデル・ノードを使用すると、PMML を作成する IBM SPSS Statistics 手続きを実行してデータを分析および使用することができます。このノードは、ライセンスが与えられた IBM SPSS Statistics のコピーが必要です。

このノードのプロパティーについては、 356 ページの『statisticsmodelnode プロパティー』に記載されています。

stpnode プロパティー

時空間予測 (STP) ノードは、ロケーション・データ、予測用の入力フィールド (予測値)、時間フィールド、および対象フィールドを使用します。各ロケーションには、それぞれの測定時の各予測値を表すデータの行が多数あります。データを分析すると、そのデータを使用して、分析で使用される形状データ内の任意のロケーションの対象値を予測できます。

表 *143. stpnode* プロパティー

stpnode プロパティー	データ型	プロパティーの説明
「フィールド」タブ		
target	*field*	これは対象フィールドです。
location	*field*	モデルの場所フィールド。地理空間フィールドのみ許可されます。
location_label	*field*	location で選択された場所にラベルを付けるために出力内で使用されるカテゴリー型フィールド。
time_field	*field*	モデルの時間フィールド。連続型の尺度を持つフィールドのみ許可されます。ストレージ タイプは、時間、日付、タイム スタンプ、整数のいずれかでなければなりません。
inputs	*[field1 ... fieldN]*	入力フィールドのリスト。
「時間区分」タブ		
interval_type_timestamp	Years Quarters Months Weeks Days Hours Minutes Seconds	
interval_type_date	Years Quarters Months Weeks Days	
interval_type_time	Hours Minutes Seconds	STP が計算で使用する時間インデックスの作成時に処理対象となる週あたりの日数を制限します。
interval_type_integer	Periods (時間インデックス フィールドの場合のみ、整数のストレージ)	データ セットを変換する間隔。選択できる項目は、モデルの time_field として選択されたフィールドのストレージ タイプによって異なります。
period_start	*integer*	

表 143. *stpnode* プロパティー (続き)

stpnode プロパティー	データ型	プロパティーの説明
start_month	January February March April May June July August September October November December	モデルがインデックス作成を開始する月です。例えば、March に設定した場合、データ セットの最初のレコードが January であるとしたら、モデルは最初の 2 つのレコードをスキップして 3 月からインデックス作成を開始します。
week_begins_on	Sunday Monday Tuesday Wednesday Thursday Friday Saturday	STP がデータから作成した時間インデックスの開始点。
days_per_week	*integer*	最小値は 1、最大値は 7、増分値は 1 です。
hours_per_day	*integer*	1 日のうちで、そのモデルが占める時間数。例えば、10 に設定した場合、モデルは day_begins_at の時刻に開始され、10 時間にわたってインデックス作成を続け、day_begins_at 値に一致する次の値までスキップします。
day_begins_at	00:00 01:00 02:00 03:00 ... 23:00	モデルがインデックス作成を開始する時間の値を設定します。
interval_increment	1 2 3 4 5 6 10 12 15 20 30	この増分の設定は分または秒に対応します。これは、モデルがデータのインデックス作成を開始する位置を決定します。つまり、増分が 30 で間隔のタイプが seconds の場合、モデルはデータのインデックス作成を 30 秒ごとに行います。

表 143. *stpnode* プロパティー (続き)

stpnode プロパティー	データ型	プロパティーの説明
data_matches_interval	*Boolean*	これを N に設定すると、モデルの構築前に、データが通常の interval_type に変換されます。 現在のデータがすでに正しい形式になっていて、interval_type とそれに関連するすべての設定がデータに一致している場合は、データの変換や集計が実行されないように、このプロパティーを Y に設定してください。 このプロパティーを Y に設定すると、すべての集計コントロールが無効になります。
agg_range_default	Sum Mean Min Max Median 1stQuartile 3rdQuartile	これは、連続型フィールドに使用されるデフォルトの集計方法を指定します。ユーザー指定の集計に明確に含まれていない連続型フィールドは、ここに指定した方法で集計されます。
custom_agg	[[field, aggregation method],[]..] デモ: [['x5' 'FirstQuartile']['x4' 'Sum']]	構造化プロパティー: スクリプト パラメーター: custom_agg 以下に例を示します。 set :stpnode.custom_agg = [[field1 function] [field2 function]] ここで、function は、当該フィールドで使用される集計関数です。
「基本」タブ		
include_intercept	*flag*	
max_autoregressive_lag	*integer*	最小値は 1、最大値は 5、増分値は 1 です。これは、予測に必要な以前のレコードの数を示します。したがって、例えば 5 に設定した場合は、以前の 5 件のレコードを使用して新しい予測が作成されます。ここに指定した、ビルド データからのレコード件数は、モデルに組み込まれます。したがって、ユーザーはモデルのスコアリング時にデータを再度提供する必要がありません。
estimation_method	Parametric Nonparametric	空間共分散行列のモデリング方法。
parametric_model	Gaussian Exponential PoweredExponential	Parametric 空間共分散モデルの順序パラメータ。

表 143. *stpnode* プロパティー (続き)

stpnode プロパティー	データ型	プロパティーの説明
exponential_power	*number*	PoweredExponential モデルのべき乗レベル。最小値は 1、最大値は 2 です。
「詳細」タブ		
max_missing_values	*integer*	モデル内で許可される、欠損値を持つレコードの最大パーセント値。
significance	*number*	モデル構築における仮説検証の有意水準。STP モデル推定のすべての検定 (2 つの適合度検定、効果 F 検定、係数 T 検定を含む) に使用する有意水準値を指定します。
「出力」タブ		
model_specifications	*flag*	
temporal_summary	*flag*	
location_summary	*flag*	場所の要約表がモデル出力に含まれるかどうかを指定します。
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*	
similarity_threshold	*number*	類似度のしきい値。この値を超えると、出力クラスターの類似度が十分に高いと判断され、1 つのクラスターに結合されます。
max_number_clusters	*integer*	モデル出力に含めることができるクラスターの上限値。
「モデル オプション」タブ		
use_model_name	*flag*	
model_name	*string*	
uncertainty_factor	*number*	最小値は 0、最大値は 100 です。将来の予測に適用される不確実性 (誤差) の増加を指定します。これは、予測の上限と下限です。

svmnode プロパティー

サポート・ベクター・マシン (SVM) ノードを使用すると、オーバーフィットすることなく、データを 2 つのグループのいずれかに分類することができます。SVM は、非常に多数の入力フィールドを含むデータセットなど、広範なデータセットを処理することができます。

例

```
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)
```

表 *144. svmnode* プロパティー:

svmnode プロパティー	値	プロパティーの説明
all_probabilities	*flag*	
stopping_criteria	1.0E-1 1.0E-2 1.0E-3 (default) 1.0E-4 1.0E-5 1.0E-6	最適化アルゴリズムをいつ停止するかを決定します。
regularization	*number*	C パラメーターとしても知られています。
precision	*number*	対象フィールドの尺度が Continuous の場合にのみ使用されます。
kernel	RBF (デフォルト) Polynomial Sigmoid Linear	変換に使用されるカーネル関数のタイプ。
rbf_gamma	*number*	kernel が RBF の場合にのみ使用されます。
gamma	*number*	kernel が Polynomial または Sigmoid の場合にのみ使用されます。
bias	*number*	
degree	*number*	kernel が Polynomial の場合にのみ使用されます。
calculate_variable_importance	*flag*	
calculate_raw_propensities	*flag*	
calculate_adjusted_ propensities	*flag*	
adjusted_propensity_partition	Test Validation	

tcmnode プロパティー

時間的因果モデリングでは、時系列データ内の重要な因果関係の検出が試行されます。時間的因果モデリングでは、一連の対象系列を指定し、それらの対象系列に対する一連の入力候補を指定します。その後、プロシージャーは、各対象系列について自己回帰の時系列モデルを構築し、対象系列との重要な因果関係を持つ入力だけを取り込みます。

表 145. *tcmnode* プロパティー

tcmnode プロパティー	値	プロパティーの説明
custom_fields	*Boolean*	
dimensionlist	[*dimension1 ... dimensionN*]	
data_struct	Multiple Single	
metric_fields	*field*	
both_target_and_input	[*f1 ... fN*]	
targets	[*f1 ... fN*]	
candidate_inputs	[*f1 ... fN*]	
forced_inputs	[*f1 ... fN*]	
use_timestamp	Timestamp Period	
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*	

表 145. *tcmnode* プロパティー (続き)

tcmnode プロパティー	値	プロパティーの説明
timestamp_increments	*integer*	
cyclic_increments	*integer*	
cyclic_periods	*list*	
output_interval	None Year Quarter Month Week Day Hour Minute Second	
is_same_interval	Same Notsame	
cross_hour	*Boolean*	
aggregate_and_distribute	*list*	
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_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	*Boolean*	
num_targets	*integer*	

表 145. tcmnode プロパティー (続き)

tcmnode プロパティー	値	プロパティーの説明
percent_targets	*integer*	
fields_display	*list*	
series_display	*list*	
network_graph_for_target	*Boolean*	
sign_level_for_target	*number*	
fit_and_outlier_for_target	*Boolean*	
sum_and_para_for_target	*Boolean*	
impact_diag_for_target	*Boolean*	
impact_diag_type_for_target	Effect Cause Both	
impact_diag_level_for_target	*integer*	
series_plot_for_target	*Boolean*	
res_plot_for_target	*Boolean*	
top_input_for_target	*Boolean*	
forecast_table_for_target	*Boolean*	
same_as_for_target	*Boolean*	
network_graph_for_series	*Boolean*	
sign_level_for_series	*number*	
fit_and_outlier_for_series	*Boolean*	
sum_and_para_for_series	*Boolean*	
impact_diagram_for_series	*Boolean*	
impact_diagram_type_for_series	Effect Cause Both	
impact_diagram_level_for_series	*integer*	
series_plot_for_series	*Boolean*	
residual_plot_for_series	*Boolean*	
forecast_table_for_series	*Boolean*	
outlier_root_cause_analysis	*Boolean*	
causal_levels	*integer*	
outlier_table	Interactive Pivot Both	
rmsp_error	*Boolean*	
bic	*Boolean*	
r_square	*Boolean*	
outliers_over_time	*Boolean*	
series_transormation	*Boolean*	
use_estimation_period	*Boolean*	

表 145. tcmnode プロパティー (続き)

tcmnode プロパティー	値	プロパティーの説明
estimation_period	Times Observation	
observations	*list*	
observations_type	Latest Earliest	
observations_num	*integer*	
observations_exclude	*integer*	
extend_records_into_future	*Boolean*	
forecastperiods	*integer*	
max_num_distinct_values	*integer*	
display_targets	FIXEDNUMBER PERCENTAGE	
goodness_fit_measure	ROOTMEAN BIC RSQUARE	
top_input_for_series	*Boolean*	
aic	*Boolean*	
rmse	*Boolean*	

ts プロパティー

時系列ノードは、時系列から指数平滑法、1 変量の自己回帰型統合移動平均法 (ARIMA)、および多変量 ARIMA (または伝達関数) モデルを推測し、将来のパフォーマンスの予測を作成します。この時系列ノードは、SPSS Modeler バージョン 18 で廃止された以前の時系列ノードと類似しています。ただし、この新しい時系列ノードは、IBM SPSS Analytic Server の機能を活用してビッグ データを処理するよう設計されており、結果モデルは SPSS Modeler バージョン 17 で追加された出力ビューアーに表示されます。

表 146. ts プロパティー

ts プロパティー	値	プロパティーの説明
targets	*field*	時系列ノードは、オプションで 1 つ以上の入力フィールドを予測値として使用しながら、1 つ以上の対象フィールドを予測します。度数フィールドおよび重みフィールドは使用しません。詳しくは、トピック 187 ページの『一般的なモデル作成ノードのプロパティー』を参照してください。
candidate_inputs	[*field1 ... fieldN*]	モデルで使用される入力または予測変数フィールド。
use_period	*flag*	

表 *146. ts* プロパティー *(*続き*)*

ts プロパティー	値	プロパティーの説明
date_time_field	*field*	
input_interval	None Unknown Year Quarter Month Week Day Hour Hour_nonperiod Minute Minute_nonperiod Second Second_nonperiod	
period_field	*field*	
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	*list*	
output_interval	None Year Quarter Month Week Day Hour Minute Second	
is_same_interval	*flag*	
cross_hour	*flag*	
aggregate_and_distribute	*list*	

表 146. ts プロパティー (続き)

ts プロパティー	値	プロパティーの説明
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	*list*	date_time_field を使用する場合にのみ使用可能です
period_estimation	*list*	use_period を使用する場合にのみ使用可能です
observations_type	Latest Earliest	
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*	

表 146. *ts* プロパティー (続き)

ts プロパティー	値	プロパティーの説明
expert_outlier_additive_patch	*flag*	
consider_newesmodels	*flag*	
exsmooth_model_type	Simple HoltsLinearTrend BrownsLinearTrend DampedTrend SimpleSeasonal WintersAdditive WintersMultiplicative DampedTrendAdditive DampedTrendMultiplicative MultiplicativeTrendAdditive MultiplicativeSeasonal MultiplicativeTrendMultiplicative MultiplicativeTrend	指数平滑法を指定します。デフォルトは Simple です。
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*	
arima.sq	*integer*	
arima_transformation_type	None SquareRoot NaturalLog	
arima_include_constant	*flag*	
tf_arima.p. *fieldname*	*integer*	伝達関数用。
tf_arima.d. *fieldname*	*integer*	伝達関数用。
tf_arima.q. *fieldname*	*integer*	伝達関数用。
tf_arima.sp. *fieldname*	*integer*	伝達関数用。
tf_arima.sd. *fieldname*	*integer*	伝達関数用。
tf_arima.sq. *fieldname*	*integer*	伝達関数用。
tf_arima.delay. *fieldname*	*integer*	伝達関数用。
tf_arima.transformation_type. *fieldname*	None SquareRoot NaturalLog	伝達関数用。
arima_detect_outliers	*flag*	
arima_outlier_additive	*flag*	
arima_outlier_level_shift	*flag*	

表 146. ts プロパティー (続き)

ts プロパティー	値	プロパティーの説明
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	*real*	
events	*field*	
continue	*flag*	
scoring_model_only	*flag*	多く (1 万単位) の時系列のモデルに使用します。
forecastperiods	*integer*	
extend_records_into_future	*flag*	
extend_metric_values	*field*	予測のための将来の値を提供できるようにします。
conf_limits	*flag*	
noise_res	*flag*	
max_models_output	*integer*	出力に表示するモデルの数を制御します。デフォルトは 10 個です。構築されたモデルの総数がこの値を超える場合、モデルは出力に表示されません。それでも、モデルはスコアリングに引き続き使用できます。

timeseriesnode プロパティー (廃止)

注: この元の時系列ノードは SPSS Modeler のバージョン 18 で廃止され、新しい時系列ノードに置き換えられました。この新しいノードは、IBM SPSS Analytic Server の機能を活用し、ビッグデータを処理するように設計されています。時系列ノードは、時系列から指数平滑法、1 変量の自己回帰型統合移動平均法 (ARIMA)、および多変量 ARIMA (または伝達関数) モデルを推測し、将来のパフォーマンスの予測を作成します。時系列ノードは、時間区分ノードによって常に先行される必要があります。

例

```
node = stream.create("timeseries", "My node")
node.setPropertyValue("method", "Exsmooth")
node.setPropertyValue("exsmooth_model_type", "HoltsLinearTrend")
node.setPropertyValue("exsmooth_transformation_type", "None")
```

表 147. *timeseriesnode* プロパティー

timeseriesnode プロパティー	値	プロパティーの説明
targets	*field*	時系列ノードは、オプションで 1 つ以上の入力フィールドを予測値として使用しながら、1 つ以上の対象フィールドを予測します。度数フィールドおよび重みフィールドは使用しません。 詳しくは、トピック 187 ページの『一般的なモデル作成ノードのプロパティー』を参照してください。
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*	
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*	

表 147. *timeseriesnode* プロパティー (続き)

timeseriesnode プロパティー	値	プロパティーの説明
arima_transformation_type	None SquareRoot NaturalLog	
arima_include_constant	*flag*	
tf_arima_p. *fieldname*	*integer*	伝達関数用。
tf_arima_d. *fieldname*	*integer*	伝達関数用。
tf_arima_q. *fieldname*	*integer*	伝達関数用。
tf_arima_sp. *fieldname*	*integer*	伝達関数用。
tf_arima_sd. *fieldname*	*integer*	伝達関数用。
tf_arima_sq. *fieldname*	*integer*	伝達関数用。
tf_arima_delay. *fieldname*	*integer*	伝達関数用。
tf_arima_transformation_type. *fieldname*	None SquareRoot NaturalLog	伝達関数用。
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	*real*	
max_lags	*integer*	
events	*field*	
scoring_model_only	*flag*	多く (1 万単位) の時系列のモデルに使用します。

treeas プロパティー

Tree-AS ノードは既存の CHAID ノードに似ていますが、Tree-AS ノードはビッグデータを処理して 1 つのツリーを作成することを目的に設計されており、結果モデルが SPSS Modeler バージョン 17 で追加された出力ビューアーに表示されます。このノードは、カイ 2 乗統計量 (CHAID) を使用して最適な分割を特定することで、ディシジョン・ツリーを生成します。CHAID をこのように使用することで、非 2 分岐ツリーを生成できます。これは、3 個以上のブランチを持つ分岐が存在することを意味します。対象フィールドおよび入力フィールドは、数値範囲 (連続型) またはカテゴリーとなります。Exhaustive CHAID は CHAID の修正版で、可能性のある分割すべてを調べることで、よりよい結果を得られますが、計算時間も長くなります。

表 148. *treeas* プロパティー

treeas プロパティー	値	プロパティーの説明
target	*field*	Tree-AS ノードでは、CHAID モデルには単一の対象フィールドおよび 1 つ以上の入力フィールドが必要になります。度数フィールドも指定できます。 詳しくは、トピック 187 ページの『一般的なモデル作成ノードのプロパティー』を参照してください。
method	chaid exhaustive_chaid	
max_depth	*integer*	最大ツリー深度 (0 から 20)。デフォルト値は 5 です。
num_bins	*integer*	データが連続型入力で構成される場合にのみ使用されます。入力に対して使用する等しいフリクエンシ ビンの数を設定します。オプションは 2、4、5、10、20、25、50、または 100 です。
record_threshold	*integer*	モデルでツリーを作成するときに、p 値の使用から効果サイズの使用に切り替えるレコード数。デフォルトは 1,000,000 です。増減は 10,000 の単位で行います。
split_alpha	*number*	分割の有意水準。この値は 0.01 から 0.99 までです。
merge_alpha	*number*	結合の有意水準。この値は 0.01 から 0.99 までです。
bonferroni_adjustment	*flag*	Bonferroni メソッドを使用して有意確率値を調整。
effect_size_threshold_cont	*number*	連続型対象を使用する際にノードの分割およびカテゴリの結合を行う効果サイズしきい値を設定します。この値は 0.01 から 0.99 までです。
effect_size_threshold_cat	*number*	カテゴリ型対象を使用する際にノードの分割およびカテゴリの結合を行う効果サイズしきい値を設定します。この値は 0.01 から 0.99 までです。
split_merged_categories	*flag*	マージしたカテゴリーの再分割を許可。
grouping_sig_level	*number*	ノード グループの形成方法または例外ノードの識別方法を決定するために使用されます。
chi_square	pearson likelihood_ratio	カイ 2 乗統計の計算に使用される方法 (Pearson または尤度比)
minimum_record_use	use_percentage use_absolute	
min_parent_records_pc	*number*	デフォルト値は 2 です。最小は 1、最大は 100、インクリメントは 1 です。親枝葉の値は子枝葉の値より大きくなければなりません。

表 148. *treeas* プロパティー (続き)

treeas プロパティー	値	プロパティーの説明
min_child_records_pc	*number*	デフォルト値は 1 です。最小は 1、最大は 100、インクリメントは 1 です。
min_parent_records_abs	*number*	デフォルト値は 100 です。最小は 1、最大は 100、インクリメントは 1 です。親枝葉の値は子枝葉の値より大きくなければなりません。
min_child_records_abs	*number*	デフォルト値は 50 です。最小は 1、最大は 100、インクリメントは 1 です。
epsilon	*number*	期待されるセル度数の最小変化。
max_iterations	*number*	収束のための最大反復回数。
use_costs	*flag*	
costs	*structured*	構造化プロパティー。形式は、実際の値、予測された値、およびコスト (予測が正しくない場合) の 3 つの値のリストです。以下に例を示します。 tree.setPropertyValue("costs", [["drugA", "drugB", 3.0], ["drugX", "drugY", 4.0]])
default_cost_increase	none linear square custom	**注:** 順序型対象に対してのみ有効です。 コスト行列にデフォルト値を設定します。
calculate_conf	*flag*	
display_rule_id	*flag*	フィールドが 1 つスコアリング出力に追加されますが、これは各レコードを割り当てるターミナル・ノードに ID を示すためのものです。

twostepnode プロパティー

TwoStep ノードで、2 段階のクラスター化手法が使用されます。最初のステップでは、データを 1 度通過させて、未処理の入力データを管理可能な一連のサブクラスターに圧縮します。2 番目のステップでは、階層クラスター化手法を使用して、サブクラスターをより大きなクラスターに結合させていきます。TwoStep には、学習データに最適なクラスター数を自動的に推定するという利点があります。また、フィールド・タイプの混在や大規模データ・セットも効率よく処理できます。

例

```
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)
```

表 *149. twostepnode* プロパティー

twostepnode プロパティー	値	プロパティーの説明
inputs	[*field1 ... fieldN*]	TwoStep モデルは対象フィールドでなく、入力フィールドのリストを使用します。重みフィールドおよび度数フィールドは認識されません。 詳しくは、トピック 187 ページの『一般的なモデル作成ノードのプロパティー』を参照してください。
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	

twostepAS のプロパティー

TwoStep クラスターは、通常ははっきりしない、データセット内での自然なグループ化 (またはクラスター) を明確にすることを目的として設計された探索ツールです。この手続きで使用されるアルゴリズムには、従来のクラスタリング手法とは異なる以下の優れた特徴があります (カテゴリー変数および連続変数の処理、クラスター数の自動選択、スケーラビリティなど)。

表 *150. twostepAS* のプロパティー

twostepAS のプロパティー	値	プロパティーの説明
inputs	[*f1 ... fN*]	TwoStepAS モデルは入力フィールドのリストを使用しますが、対象フィールドは使用しません。重みフィールドおよび度数フィールドは認識されません。
use_predefined_roles	Boolean	デフォルト=True

表 150. *twostepAS* のプロパティー (続き)

twostepAS のプロパティー	値	プロパティーの説明
use_custom_field_assignments	Boolean	デフォルト=False
cluster_num_auto	Boolean	デフォルト=True
min_num_clusters	整数	デフォルト=2
max_num_clusters	整数	デフォルト=15
num_clusters	整数	デフォルト=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	Boolean	
random_seed	整数	
distance_measure	Euclidean Loglikelihood	
include_outlier_clusters	Boolean	デフォルト=True
num_cases_in_feature_tree_leaf_is_less_than	整数	デフォルト=10
top_perc_outliers	整数	デフォルト=5
initial_dist_change_threshold	整数	デフォルト=0
leaf_node_maximum_branches	整数	デフォルト=8
non_leaf_node_maximum_branches	整数	デフォルト=8
max_tree_depth	整数	デフォルト=3
adjustment_weight_on_measurement_level	整数	デフォルト=6
memory_allocation_mb	数値	デフォルト=512
delayed_split	Boolean	デフォルト=True
fields_to_standardize	[*f1 ... fN*]	
adaptive_feature_selection	Boolean	デフォルト=True
featureMisPercent	整数	デフォルト=70
coefRange	数値	デフォルト=0.05
percCasesSingleCategory	整数	デフォルト=95
numCases	整数	デフォルト=24
include_model_specifications	Boolean	デフォルト=True
include_record_summary	Boolean	デフォルト=True
include_field_transformations	Boolean	デフォルト=True
excluded_inputs	Boolean	デフォルト=True
evaluate_model_quality	Boolean	デフォルト=True
show_feature_importance bar chart	Boolean	デフォルト=True
show_feature_importance_ word_cloud	Boolean	デフォルト=True

表 150. *twostepAS* のプロパティー (続き)

twostepAS のプロパティー	値	プロパティーの説明
show_outlier_clusters interactive_table_and_chart	Boolean	デフォルト=True
show_outlier_clusters_pivot_table	Boolean	デフォルト=True
across_cluster_feature_importance	Boolean	デフォルト=True
across_cluster_profiles_pivot_table	Boolean	デフォルト=True
withinprofiles	Boolean	デフォルト=True
cluster_distances	Boolean	デフォルト=True
cluster_label	String Number	
label_prefix	String	

第 14 章 モデル・ナゲット・ノードのプロパティー

モデル・ナゲット・ノードは、他のノードと同じ共通のプロパティーを共有しています。 詳しくは、トピック 76 ページの『共通のノード・プロパティー』を参照してください。

applyanomalydetectionnode プロパティー

異常値検出モデル作成ノードを使用して、異常値検出モデル・ナゲットを生成することができます。このモデル・ナゲットのスクリプト名は、*applyanomalydetectionnode* です。モデル作成ノード自体のスクリプトの詳細は、188 ページの『anomalydetectionnode プロパティー』を参照してください。

表 *151. applyanomalydetectionnode* プロパティー:

applyanomalydetectionnode プロパティー	値	プロパティーの説明
anomaly_score_method	FlagAndScore FlagOnly ScoreOnly	スコアリング用に、作成される出力を決めます。
num_fields	*integer*	報告するフィールド数。
discard_records	*flag*	レコードが出力から廃棄されるかどうかを示します。
discard_anomalous_records	*flag*	異常なレコードを廃棄するか、または異常でないレコードを廃棄するかの標識。デフォルトは、異常でない レコードが廃棄されることを示す off です。それに対し、on の場合は、異常なレコードが廃棄されます。このプロパティーは、discard_records が有効な場合にだけ、有効になります。

applyapriorinode プロパティー

Apriori モデル作成ノードを使用して、Apriori モデル・ナゲットを生成することができます。このモデル・ナゲットのスクリプト名は、*applyapriorinode* です。モデル作成ノード自体のスクリプトの詳細は、189 ページの『apriorinode プロパティー』を参照してください。

表 *152. applyapriorinode* プロパティー:

applyapriorinode プロパティー	値	プロパティーの説明
max_predictions	*number (*整数*)*	
ignore_unmatached	*flag*	
allow_repeats	*flag*	
check_basket	NoPredictions Predictions NoCheck	

表 152. *applyapriorinode* プロパティー (続き):

applyapriorinode プロパティー	値	プロパティーの説明
criterion	Confidence Support RuleSupport Lift Deployability	

applyassociationrulesnode プロパティー

アソシエーション ルール モデル作成ノードを使用して、アソシエーション ルール モデル ナゲットを作成することができます。このモデル ナゲットのスクリプト名は *applyassociationrulesnode* です。モデル作成ノード自体をスクリプト化する方法については、 190 ページの『associationrulesnode プロパティー』を参照してください。

表 153. *applyassociationrulesnode* プロパティー

applyassociationrulesnode プロパティー	データ型	プロパティーの説明
max_predictions	*integer*	スコアに対する各入力に適用できるルールの最大数。
criterion	Confidence Rulesupport Lift Conditionsupport Deployability	ルールの強度を判断するための尺度を選択します。
allow_repeats	*Boolean*	同じ予測を持つルールをスコア内に含めるかどうかを決定します。
check_input	NoPredictions Predictions NoCheck	

applyautoclassifiernode プロパティー

自動分類モデル作成ノードを使用して、自動分類モデル・ナゲットを生成することができます。このモデル・ナゲットのスクリプト名は、*applyautoclassifiernode* です。モデル作成ノードのスクリプト化の詳細は、193 ページの『autoclassifiernode プロパティー』を参照してください。

表 154. *applyautoclassifiernode* プロパティー:

applyautoclassifiernode プロパティー	値	プロパティーの説明
flag_ensemble_method	Voting ConfidenceWeightedVoting RawPropensityWeightedVoting HighestConfidence AverageRawPropensity	アンサンブル・スコアを決定するために使用する方法を指定します。この設定は、選択された対象がフラグ型フィールドである場合にのみ適用されます。

表 154. *applyautoclassifiernode* プロパティー (続き):

applyautoclassifiernode プロパティー	値	プロパティーの説明
flag_voting_tie_selection	Random HighestConfidence RawPropensity	票決方法が選択された場合、可否同数の解決方法を指定します。この設定は、選択された対象がフラグ型フィールドである場合にのみ適用されます。
set_ensemble_method	Voting ConfidenceWeightedVoting HighestConfidence	アンサンブル・スコアを決定するために使用する方法を指定します。この設定は、選択された対象がセット型フィールドである場合にのみ適用されます。
set_voting_tie_selection	Random HighestConfidence	票決方法が選択された場合、可否同数の解決方法を指定します。この設定は、選択された対象が名義型フィールドである場合にのみ適用されます。

applyautoclusternode プロパティー

自動クラスター・モデル作成ノードを使用して、自動クラスター・モデル・ナゲットを生成することができます。このモデル・ナゲットのスクリプト名は、*applyautoclusternode* です。このモデル・ナゲットの他のプロパティーはありません。モデル作成ノード自体のスクリプトの詳細は、 195 ページの『autoclusternode プロパティー』を参照してください。

applyautonumericnode プロパティー

自動数値モデル作成ノードを使用して、自動数値モデル・ナゲットを生成することができます。このモデル・ナゲットのスクリプト名は、*applyautonumericnode* です。モデル作成ノードのスクリプト化の詳細は、197 ページの『autonumericnode プロパティー』を参照してください。

表 155. *applyautonumericnode* プロパティー:

applyautonumericnode プロパティー	値	プロパティーの説明
calculate_standard_error	*flag*	

applybayesnetnode プロパティー

ベイズ・ネットワーク・モデル作成ノードを使用して、ベイズ・ネットワーク・モデル・ナゲットを生成することができます。このモデル・ナゲットのスクリプト名は、*applybayesnetnode* です。モデル作成ノード自体のスクリプトの詳細は、 198 ページの『bayesnetnode プロパティー』を参照してください。

表 156. *applybayesnetnode* プロパティー:

applybayesnetnode プロパティー	値	プロパティーの説明
all_probabilities	*flag*	
raw_propensity	*flag*	
adjusted_propensity	*flag*	
calculate_raw_propensities	*flag*	

表 156. *applybayesnetnode* プロパティー (続き):

applybayesnetnode プロパティー	値	プロパティーの説明
calculate_adjusted_propensities	*flag*	

applyc50node プロパティー

C5.0 モデル作成ノードを使用して、C5.0 モデル・ナゲットを生成することができます。このモデル・ナゲットのスクリプト名は、*applyc50node* です。モデル作成ノード自体のスクリプトの詳細は、 200 ページの『c50node プロパティー』を参照してください。

表 157. *applyc50node* プロパティー:

applyc50node プロパティー	値	プロパティーの説明
sql_generate	Never NoMissingValues	ルールセット実行時の SQL 生成オプションの設定に使用します。
calculate_conf	*flag*	SQL 生成が有効になっている場合に利用できます。このプロパティーには、生成されたツリー中の確信度計算が含まれています。
calculate_raw_propensities	*flag*	
calculate_adjusted_propensities	*flag*	

applycarmanode プロパティー

CARMA モデル作成ノードを使用して、CARMA モデル・ナゲットを生成することができます。このモデル・ナゲットのスクリプト名は、*applycarmanode* です。このモデル・ナゲットの他のプロパティーはありません。モデル作成ノード自体のスクリプトの詳細は、 202 ページの『carmanode プロパティー』を参照してください。

applycartnode プロパティー

C&R Tree モデル作成を使用して、C&R Tree モデル・ナゲットを生成することができます。このモデル・ナゲットのスクリプト名は、*applycartnode* です。モデル作成ノード自体のスクリプトの詳細は、 203 ページの『cartnode プロパティー』を参照してください。

表 158. *applycartnode* プロパティー:

applycartnode プロパティー	値	プロパティーの説明
enable_sql_generation	Never MissingValues NoMissingValues	ルールセット実行時の SQL 生成オプションの設定に使用します。
calculate_conf	*flag*	SQL 生成が有効になっている場合に利用できます。このプロパティーには、生成されたツリー中の確信度計算が含まれています。
display_rule_id	*flag*	フィールドが 1 つスコアリング出力に追加されますが、これは各レコードを割り当てるターミナル・ノードに ID を示すためのものです。

表 158. *applycartnode* プロパティー (続き) :

applycartnode プロパティー	値	プロパティーの説明
calculate_raw_propensities	*flag*	
calculate_adjusted_propensities	*flag*	

applychaidnode プロパティー

CHAID モデル作成ノードを使用して、CHAID モデル・ナゲットを生成することができます。このモデル・ナゲットのスクリプト名は、*applychaidnode* です。モデル作成ノード自体のスクリプトの詳細は、205 ページの『chaidnode プロパティー』を参照してください。

表 159. *applychaidnode* プロパティー :

applychaidnode Properties	値	プロパティーの説明
enable_sql_generation	Never MissingValues	ルールセット実行時の SQL 生成オプションの設定に使用します。
calculate_conf	*flag*	
display_rule_id	*flag*	フィールドが 1 つスコアリング出力に追加されますが、これは各レコードを割り当てるターミナル・ノードに ID を示すためのものです。
calculate_raw_propensities	*flag*	
calculate_adjusted_propensities	*flag*	

applycoxregnode プロパティー

Cox モデル作成ノードを使用して、Cox モデル・ナゲットを生成することができます。このモデル・ナゲットのスクリプト名は、*applycoxregnode* です。モデル作成ノード自体のスクリプトの詳細は、207 ページの『coxregnode プロパティー』を参照してください。

表 160. *applycoxregnode* プロパティー :

applycoxregnode プロパティー	値	プロパティーの説明
future_time_as	Intervals field	
time_interval	*number*	
num_future_times	*integer*	
time_field	*field*	
past_survival_time	*field*	
all_probabilities	*flag*	
cumulative_hazard	*flag*	

applydecisionlistnode プロパティー

ディシジョン・リスト・モデル作成ノードを使用して、ディシジョン・リスト・モデル・ナゲットを生成することができます。このモデル・ナゲットのスクリプト名は、*applydecisionlistnode* です。モデル作成ノード自体のスクリプトの詳細は、 209 ページの『decisionlistnode プロパティー』を参照してください。

表 *161. applydecisionlistnode* プロパティー：

applydecisionlistnode プロパティー	値	プロパティーの説明
enable_sql_generation	*flag*	真に設定したときは、ディシジョン・リスト・モデルが SQL へプッシュバックされるように IBM SPSS Modeler が試行します。
calculate_raw_propensities	*flag*	
calculate_adjusted_propensities	*flag*	

applydiscriminantnode プロパティー

判別分析モデル作成ノードを使用して、判別分析モデル・ナゲットを生成することができます。このモデル・ナゲットのスクリプト名は、*applydiscriminantnode* です。モデル作成ノード自体のスクリプトの詳細は、 210 ページの『discriminantnode プロパティー』を参照してください。

表 *162. applydiscriminantnode* プロパティー：

applydiscriminantnode プロパティー	値	プロパティーの説明
calculate_raw_propensities	*flag*	
calculate_adjusted_propensities	*flag*	

applyextension プロパティー

拡張モデル作成ノードを使用して、拡張モデル ナゲットを生成することができます。このモデル ナゲットのスクリプト名は *applyextension* です。モデル作成ノード自体をスクリプト化する方法については、 212 ページの『extensionmodelnode プロパティー』を参照してください。

Python for Spark の例

```
#### 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].
index(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)
```

R の例

```
#### 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)""")
```

表 163. *applyextension* プロパティー

applyextension プロパティー	値	プロパティーの説明
r_syntax	*string*	モデル・スコアリング用の R スクリプト・シンタックス。
python_syntax	*string*	モデル・スコアリング用の Python スクリプト・シンタックス。
use_batch_size	*flag*	バッチ処理を使用可能にします。
batch_size	*integer*	各バッチに含めるデータ レコードの数を指定します。
convert_flags	StringsAndDoubles LogicalValues	フラグ型フィールドを変換するためのオプション。
convert_missing	*flag*	欠損値を R の NA 値に変換するためのオプション。
convert_datetime	*flag*	日付形式または日付/時刻形式の変数を R の日付/時刻形式に変換するためのオプション。
convert_datetime_class	POSIXct POSIXlt	日付形式または日付/時刻形式の変数のうち、どの形式の変数を変換するかを指定するためのオプション。

applyfactornode プロパティー

因子分析モデル作成ノードを使用して、因子分析モデル・ナゲットを生成することができます。このモデル・ナゲットのスクリプト名は、*applyfactornode* です。このモデル・ナゲットの他のプロパティーはありません。モデル作成ノード自体のスクリプトの詳細は、 215 ページの『factornode プロパティー』を参照してください。

applyfeatureselectionnode プロパティー

変数選択モデル作成ノードを使用して、変数選択モデル・ナゲットを生成することができます。このモデル・ナゲットのスクリプト名は、*applyfeatureselectionnode* です。モデル作成ノード自体のスクリプトの詳細は、 216 ページの『featureselectionnode プロパティー』を参照してください。

表 164. *applyfeatureselectionnode* プロパティー:

applyfeatureselectionnode プロパティー	値	プロパティーの説明
selected_ranked_fields		モデル・ブラウザー内で検査されるランク付きのフィールドを指定します。
selected_screened_fields		モデル・ブラウザー内で検査されるスクリーニングされたフィールドを指定します。

applygeneralizedlinearnode プロパティー

一般化線型 (genlin) モデル作成ノードを使用して、一般化線型モデル・ナゲットを生成することができます。このモデル・ナゲットのスクリプト名は、*applygeneralizedlinearnode* です。モデル作成ノード自体のスクリプトの詳細は、 218 ページの『genlinnode プロパティー』を参照してください。

表 *165. applygeneralizedlinearnode* プロパティー：

applygeneralizedlinearnode プロパティー	値	プロパティーの説明
calculate_raw_propensities	*flag*	
calculate_adjusted_propensities	*flag*	

applyglmmnode プロパティー

GLMM モデル作成ノードを使用して、GLMM モデル・ナゲットを生成することができます。このモデル・ナゲットのスクリプト名は、*applyglmmnode* です。モデル作成ノード自体のスクリプトの詳細は、222 ページの『glmmnode プロパティー』を参照してください。

表 *166. applyglmmnode* プロパティー：

applyglmmnode プロパティー	値	プロパティーの説明
confidence	onProbability onIncrease	スコアリングの確信度を計算する基準 (最も高い予測確率、または最も高い予測確率と 2 番目に高い予測確率との差)。
score_category_probabilities	*flag*	True に設定された場合、カテゴリー対象の予測確率を生成します。カテゴリーごとにフィールドが作成されます。デフォルトは False です。
max_categories	*integer*	確率を予測するカテゴリーの最大数です。score_category_probabilities が True の場合にのみ使用されます。
score_propensity	*flag*	True に設定された場合、フラグ型対象を含むモデルに対して、未調整傾向スコア (「true」の結果の確率) を生成します。データ区分が有効な場合、テスト・データ区分に基づいて、調整済み傾向スコアも生成します。デフォルトは False です。
enable_sql_generation	udf native	ストリーム実行中の SQL 生成オプションを設定するために使用します。データベースにプッシュバックして SPSS® Modeler Server Scoring Adapter でスコアリングするか (スコアリング アダプターがインストール済みのデータベースに接続している場合)、SPSS Modeler 内でスコアリングするかを選択できます。 デフォルト値は udf です。

applygle プロパティー

GLE モデル作成ノードを使用して、GLE モデル ナゲットを生成できます。このモデル ナゲットのスクリプト名は *applygle* です。モデル作成ノード自体をスクリプト化する方法については、 225 ページの『gle プロパティー』を参照してください。

表 *167. applygle* プロパティー

applygle プロパティー	値	プロパティーの説明
enable_sql_generation	udf native	ストリーム実行中の SQL 生成オプションを設定するために使用します。データベースにプッシュバックして SPSS Modeler Server Scoring Adapter を使用してスコアリングするか (スコアリング アダプタがインストール済みのデータベースに接続している場合)、SPSS Modeler 内でスコアリングするかを選択します。

applykmeansnode プロパティー

K-means モデル作成ノードを使用して、K-means モデル・ナゲットを生成することができます。このモデル・ナゲットのスクリプト名は、*applykmeansnode* です。このモデル・ナゲットの他のプロパティーはありません。モデル作成ノード自体のスクリプトの詳細は、 230 ページの『kmeansnode プロパティー』を参照してください。

applyknnnode プロパティー

KNN モデル作成ノードを使用して、KNN モデル・ナゲットを生成することができます。このモデル・ナゲットのスクリプト名は、*applyknnnode* です。モデル作成ノード自体のスクリプトの詳細は、 231 ページの『knnnode プロパティー』を参照してください。

表 *168. applyknnnode* プロパティー :

applyknnnode プロパティー	値	プロパティーの説明
all_probabilities	*flag*	
save_distances	*flag*	

applykohonennode プロパティー

Kohonen モデル作成ノードを使用して、Kohonen モデル・ナゲットを生成することができます。このモデル・ナゲットのスクリプト名は、*applykohonennode*です。このモデル・ナゲットの他のプロパティーはありません。モデル作成ノード自体のスクリプトの詳細は、 200 ページの『c50node プロパティー』を参照してください。

applylinearnode プロパティー

線型モデル作成ノードを使用して、線型モデル・ナゲットを生成することができます。このモデル・ナゲットのスクリプト名は、*applylinearnode* です。モデル作成ノード自体のスクリプトの詳細は、 234 ページの『linearnode プロパティー』を参照してください。

表 *169. applylinearnode* プロパティー：

linear プロパティー	値	プロパティーの説明
use_custom_name	*flag*	
custom_name	*string*	
enable_sql_generation	udf native puresql	ストリーム実行中の SQL 生成オプションを設定するために使用します。データベースにプッシュバックして SPSS® Modeler Server Scoring Adapter でスコアリングするか (スコアリング アダプターがインストール済みのデータベースに接続している場合)、SPSS Modeler 内でスコアリングするか、またはデータベースにプッシュバックして SQL でスコアリングするかを選択できます。 デフォルト値は udf です。

applylinearasnode プロパティー

Linear-AS モデル作成ノードを使用して、Linear-AS モデル ナゲットを生成できます。このモデル ナゲットのスクリプト名は *applylinearasnode* です。モデル作成ノード自体のスクリプトの詳細は、 235 ページの『linearasnode プロパティー』を参照してください。

表 *170. applylinearasnode* プロパティー

applylinearasnode プロパティー	値	プロパティーの説明
enable_sql_generation	udf native	デフォルト値は udf です。

applylogregnode プロパティー

ロジスティック回帰モデル作成ノードを使用して、ロジスティック回帰モデル・ナゲットを生成することができます。このモデル・ナゲットのスクリプト名は、*applylogregnode* です。モデル作成ノード自体のスクリプトの詳細は、 236 ページの『logregnode プロパティー』を参照してください。

表 *171. applylogregnode* プロパティー：

applylogregnode プロパティー	値	プロパティーの説明
calculate_raw_propensities	*flag*	
calculate_conf	*flag*	
enable_sql_generation	*flag*	

applylsvmnode プロパティー

LSVM モデル作成ノードを使用して、LSVM モデル・ナゲットを生成することができます。このモデル・ナゲットのスクリプト名は *applylsvmnode* です。モデル作成ノード自体をスクリプト化する方法については、 241 ページの『lsvmnode プロパティー』を参照してください。

表 *172. applylsvmnode* プロパティー

applylsvmnode プロパティー	値	プロパティーの説明
calculate_raw_propensities	*flag*	未調整傾向スコアを計算するかどうかを指定します。
enable_sql_generation	udf native	Scoring Adapter (インストールされている場合) を使用またはインプロセスでスコアリングするか、データベースの外部でスコアリングするかを指定します。

applyneuralnetnode プロパティー

ニューラル・ネットワーク・モデル作成ノードを使用して、ニューラル・ネットワーク・モデル・ナゲットを生成することができます。このモデル・ナゲットのスクリプト名は、*applyneuralnetnode* です。モデル作成ノード自体のスクリプトの詳細は、 242 ページの『neuralnetnode プロパティー』を参照してください。

注意**:** 機能が拡張された新しいバージョンのニューラル・ネットワーク ナゲットがこのリリースで使用できます。新しいバージョンについては次の項で説明します (*applyneuralnetwork*)。以前のバージョンは現在も使用できますが、スクリプトを更新して新しいバージョンを使用することをお勧めします。旧バージョンの詳細を参照用に記載しておりますが、それに対するサポートは今後のリリースで廃止されます。

表 *173. applyneuralnetnode* プロパティー :

applyneuralnetnode プロパティー	値	プロパティーの説明
calculate_conf	*flag*	SQL 生成が有効になっている場合に利用できます。このプロパティーには、生成されたツリー中の確信度計算が含まれています。
enable_sql_generation	*flag*	
nn_score_method	Difference SoftMax	
calculate_raw_propensities	*flag*	
calculate_adjusted_propensities	*flag*	

applyneuralnetworknode プロパティー

ニューラル・ネットワーク・モデル作成ノードを使用して、ニューラル・ネットワーク・モデル・ナゲットを生成することができます。このモデル・ナゲットのスクリプト名は、*applyneuralnetworknode* です。モデル作成ノード自体のスクリプトの詳細は、neuralnetworknode プロパティーを参照してください。

表 *174. applyneuralnetworknode* プロパティー

applyneuralnetworknode プロパティー	値	プロパティーの説明
use_custom_name	*flag*	
custom_name	*string*	
confidence	onProbability onIncrease	
score_category_probabilities	*flag*	
max_categories	*number*	
score_propensity	*flag*	
enable_sql_generation	udf native puresql	ストリーム実行中の SQL 生成オプションを設定するために使用します。データベースにプッシュバックして SPSS® Modeler Server Scoring Adapter でスコアリングするか (スコアリング アダプターがインストール済みのデータベースに接続している場合)、SPSS Modeler 内でスコアリングするか、またはデータベースにプッシュバックして SQL でスコアリングするかを選択できます。 デフォルト値は udf です。

applyocsvmnode のプロパティー

One-Class SVM ノードを使用して、One-Class SVM モデル・ナゲットを作成することができます。このモデル・ナゲットのスクリプト名は、*applyocsvmnode* です。このモデル・ナゲットの他のプロパティーはありません。モデル作成ノード自体をスクリプト化する方法については、 362 ページの『ocsvmnode のプロパティー』を参照してください。

applyquestnode プロパティー

QUEST モデル作成ノードを使用して、QUEST モデル・ナゲットを生成することができます。このモデル・ナゲットのスクリプト名は、*applyquestnode* です。モデル作成ノード自体のスクリプトの詳細は、246 ページの『questnode プロパティー』を参照してください。

表 *175. applyquestnode* プロパティー：

applyquestnode プロパティー	値	プロパティーの説明
enable_sql_generation	Never MissingValues NoMissingValues	ルールセット実行時の SQL 生成オプションの設定に使用します。
calculate_conf	*flag*	

表 175. *applyquestnode* プロパティー (続き):

applyquestnode プロパティー	値	プロパティーの説明
display_rule_id	*flag*	フィールドが 1 つスコアリング出力に追加されますが、これは各レコードを割り当てるターミナル・ノードに ID を示すためのものです。
calculate_raw_propensities	*flag*	
calculate_adjusted_propensities	*flag*	

applyr プロパティー

R 作成ノードを使用して、R モデル・ナゲットを生成することができます。このモデル・ナゲットのスクリプト名は、*applyr* です。モデル作成ノード自体のスクリプトの詳細は、 200 ページの『buildr プロパティー』を参照してください。

表 176. *applyr* プロパティー

applyr プロパティー	値	プロパティーの説明
score_syntax	*string*	モデル・スコアリング用の R スクリプト・シンタックス。
convert_flags	StringsAndDoubles LogicalValues	フラグ型フィールドを変換するためのオプション。
convert_datetime	*flag*	日付形式または日付/時刻形式の変数を R の日付/時刻形式に変換するためのオプション。
convert_datetime_class	POSIXct POSIXlt	日付形式または日付/時刻形式の変数のうち、どの形式の変数を変換するかを指定するためのオプション。
convert_missing	*flag*	欠損値を R の NA 値に変換するためのオプション。
use_batch_size	*flag*	バッチ処理を使用可能にします
batch_size	*integer*	各バッチに含めるデータ レコードの数を指定します

applyrandomtrees プロパティー

ランダム ツリー モデル作成ノードを使用して、ランダム ツリー モデル ナゲットを生成できます。このモデル ナゲットのスクリプト名は *applyrandomtrees* です。モデル作成ノード自体をスクリプト化する方法については、 248 ページの『randomtrees プロパティー』を参照してください。

表 177. *applyrandomtrees* プロパティー

applyrandomtrees プロパティー	値	プロパティーの説明
calculate_conf	*flag*	このプロパティーには、生成されたツリー中の確信度計算が含まれています。

表 177. *applyrandomtrees* プロパティー (続き)

applyrandomtrees プロパティー	値	プロパティーの説明
enable_sql_generation	udf native	ストリーム実行中の SQL 生成オプションを設定するために使用します。データベースにプッシュバックして SPSS Modeler Server Scoring Adapter を使用してスコアリングするか (スコアリング アダプタがインストール済みのデータベースに接続している場合)、SPSS Modeler 内でスコアリングするかを選択します。

applyregressionnode プロパティー

線型回帰モデル作成ノードを使用して、線型回帰モデル・ナゲットを生成することができます。このモデル・ナゲットのスクリプト名は、*applyregressionnode* です。このモデル・ナゲットの他のプロパティーはありません。モデル作成ノード自体のスクリプトの詳細は、 250 ページの『regressionnode プロパティー』を参照してください。

applyselflearningnode プロパティー

自己学習応答モデル (SLRM) モデル作成ノードを使用して、SLRM モデル・ナゲットを生成することができます。このモデル・ナゲットのスクリプト名は、*applyselflearningnode* です。モデル作成ノード自体のスクリプトの詳細は、 253 ページの『slrmnode プロパティー』を参照してください。

表 178. *applyselflearningnode* プロパティー:

applyselflearningnode プロパティー	値	プロパティーの説明
max_predictions	*number*	
randomization	*number*	
scoring_random_seed	*number*	
sort	ascending descending	高いスコアまたは低いスコアのどちらを持つオファーが最初に表示されるかを指定します。
model_reliability	*flag*	「設定」タブでモデルの信頼性を考慮します。

applysequencenode プロパティー

シーケンス・モデル作成ノードを使用して、シーケンス・モデル・ナゲットを生成することができます。このモデル・ナゲットのスクリプト名は、*applysequencenode* です。このモデル・ナゲットの他のプロパティーはありません。モデル作成ノード自体のスクリプトの詳細は、 252 ページの『sequencenode プロパティー』を参照してください。

applysvmnode プロパティー

SVM モデル作成ノードを使用して、SVM モデル・ナゲットを生成することができます。このモデル・ナゲットのスクリプト名は、*applysvmnode* です。モデル作成ノード自体のスクリプトの詳細は、 259 ページの『svmnode プロパティー』を参照してください。

表 *179. applysvmnode* プロパティー：

applysvmnode プロパティー	値	プロパティーの説明
all_probabilities	*flag*	
calculate_raw_propensities	*flag*	
calculate_adjusted_propensities	*flag*	

applystpnode プロパティー

STP モデル作成ノードを使用して、関連するモデル ナゲットを生成することができます。このモデル ナゲットにより、出力ビューアにモデル出力が表示されます。このモデル ナゲットのスクリプト名は *applystpnode* です。モデル作成ノード自体をスクリプト化する方法については、 254 ページの『stpnode プロパティー』を参照してください。

表 *180. applystpnode* プロパティー

applystpnode プロパティー	データ型	プロパティーの説明
uncertainty_factor	*Boolean*	最小値は 0、最大値は 100 です。

applytcmnode プロパティー

時間的因果モデリング (TCM) モデル作成ノードを使用して、TCM モデル ナゲットを生成できます。このモデル ナゲットのスクリプト名は、*applytcmnode* です。モデル作成ノード自体をスクリプト化する方法については、 260 ページの『tcmnode プロパティー』を参照してください。

表 *181. applytcmnode* プロパティー

applytcmnode プロパティー	値	プロパティーの説明
ext_future	*boolean*	
ext_future_num	*integer*	
noise_res	*boolean*	
conf_limits	*boolean*	
target_fields	*list*	
target_series	*list*	

applyts プロパティー

時系列モデル作成ノードを使用して、時系列モデル・ナゲットを生成できます。このモデル ナゲットのスクリプト名は、*applyts* です。モデル作成ノード自体をスクリプト化する方法については、 263 ページの『ts プロパティー』を参照してください。

表 *182. applyts* プロパティー

applyts プロパティー	値	プロパティーの説明
extend_records_into_future	*Boolean*	
ext_future_num	*integer*	
compute_future_values_input	*Boolean*	
forecastperiods	*integer*	
noise_res	*boolean*	
conf_limits	*boolean*	
target_fields	*list*	
target_series	*list*	
includeTargets	*field*	

applytimeseriesnode プロパティー (廃止)

時系列モデル作成ノードを使用して、時系列モデル・ナゲットを生成できます。このモデル・ナゲットのスクリプト名は、*applytimeseriesnode* です。モデル作成ノード自体のスクリプトの詳細は、 267 ページの『timeseriesnode プロパティー (廃止)』を参照してください。

表 *183. applytimeseriesnode* プロパティー：

applytimeseriesnode プロパティー	値	プロパティーの説明
calculate_conf	*flag*	
calculate_residuals	*flag*	

applytreeas プロパティー

Tree-AS モデル作成ノードを使用して、Tree-AS モデル ナゲットを生成できます。このモデル ナゲットのスクリプト名は *applytreenas* です。モデル作成ノード自体をスクリプト化する方法については、 269 ページの『treeas プロパティー』を参照してください。

表 *184. applytreeas* プロパティー

applytreeas プロパティー	値	プロパティーの説明
calculate_conf	*flag*	このプロパティーには、生成されたツリー中の確信度計算が含まれています。
display_rule_id	*flag*	フィールドが 1 つスコアリング出力に追加されますが、これは各レコードを割り当てるターミナル・ノードに ID を示すためのものです。

表 *184. applytreeas* プロパティー *(*続き*)*

applytreeas プロパティー	値	プロパティーの説明
enable_sql_generation	udf native	ストリーム実行中の SQL 生成オプションを設定するために使用します。データベースにプッシュバックして SPSS Modeler Server Scoring Adapter を使用してスコアリングするか (スコアリング アダプタがインストール済みのデータベースに接続している場合)、SPSS Modeler 内でスコアリングするかを選択します。

applytwostepnode プロパティー

TwoStep モデル作成ノードを使用して、TwoStep モデル・ナゲットを生成することができます。このモデル・ナゲットのスクリプト名は、*applytwostepnode* です。このモデル・ナゲットの他のプロパティーはありません。モデル作成ノード自体のスクリプトの詳細は、 271 ページの『twostepnode プロパティー』を参照してください。

applytwostepAS のプロパティー

TwoStep AS モデル作成ノードを使用して、TwoStep AS モデル ナゲットを生成することができます。このモデル ナゲットのスクリプト名は *applytwostepAS* です。モデル作成ノード自体のスクリプトの詳細は、 272 ページの『twostepAS のプロパティー』を参照してください。

表 *185. applytwostepAS* のプロパティー

applytwostepAS のプロパティー	値	プロパティーの説明
enable_sql_generation	udf native	ストリーム実行中の SQL 生成オプションを設定するために使用します。データベースにプッシュバックして SPSS® Modeler Server Scoring Adapter でスコアリングするか (スコアリング アダプターがインストール済みのデータベースに接続している場合)、SPSS Modeler 内でスコアリングするかを選択できます。 デフォルト値は udf です。

applyxgboosttreenode のプロパティー

XGBoost Tree ノードを使用して、XGBoost Tree モデル・ナゲットを生成できます。このモデル・ナゲットのスクリプト名は、*applyxgboosttreenode* です。このモデル・ナゲットの他のプロパティーはありません。モデル作成ノード自体をスクリプト化する方法については、 359 ページの『xgboosttreenode のプロパティー』を参照してください。

applyxgboostlinearnode のプロパティー

XGBoost Linear ノードを使用して、XGBoost Linear モデル・ナゲットを生成できます。このモデル・ナゲットのスクリプト名は、*applyxgboostlinearnode* です。このモデル・ナゲットの他のプロパティーはありません。モデル作成ノード自体をスクリプト化する方法については、 361 ページの『xboostlinearnode のプロパティー』を参照してください。

第 15 章 データベース・モデル作成ノードのプロパティー

IBM SPSS Modeler は、データベース・ベンダーから入手できる、Microsoft SQL Server Analysis Services、Oracle Data Mining、IBM Netezza® Analytics などのデータ・マイニングおよびモデル作成のツールとの統合をサポートしています。IBM SPSS Modeler ネイティブ・データベース・アルゴリズムを使用して、アプリケーション内からのモデルの構築およびスコアリングがすべて可能です。データベース・モデルは、このセクションで説明するプロパティーを使用してスクリプトで作成および処理することも可能です。

例えば、次のスクリプトの引用は、IBM SPSS Modeler スクリプト・インターフェースを使用した Microsoft デシジョン・ツリー・モデルの作成を示します。

```
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([])
```

Microsoft モデル作成ノードのプロパティー

Microsoft モデル作成ノードのプロパティー

共通のプロパティー

次のプロパティーは、Microsoft データベース・モデル作成ノードに共通です。

表 *186.* 共通の *Microsoft* ノード・プロパティー

共通の **Microsoft** ノード・プロパティー	値	プロパティーの説明
analysis_database_name	*string*	Analysis Services データベースの名前。
analysis_server_name	*string*	Analysis Services ホストの名前。
use_transactional_data	*flag*	入力データがテーブル形式またはトランザクション形式かを指定します。

表 186. 共通の *Microsoft* ノード・プロパティー (続き)

共通の **Microsoft** ノード・プロパティー	値	プロパティーの説明
inputs	*list*	テーブル形式の入力フィールド。
target	*field*	予測フィールド (MS クラスタリング・ノードまたはシーケンス・クラスタリング・ノードには該当しない)。
unique_field	*field*	キー・フィールド。
msas_parameters	*structured*	アルゴリズム・パラメーター。 詳しくは、トピック 297 ページの『アルゴリズム・パラメーター』を参照してください。
with_drillthrough	*flag*	「ドリルスルーあり」オプション。

MS デシジョン・ツリー

mstreenode タイプのノードには、特定のプロパティーが定義されていません。このセクションの冒頭にある共通 Microsoft プロパティーを参照してください。

MS クラスタリング

msclusternode タイプのノードには、特定のプロパティーが定義されていません。このセクションの冒頭にある共通 Microsoft プロパティーを参照してください。

MS アソシエーション・ルール

次のプロパティーは、msassocnode タイプのノードで使用できます。

表 187. *msassocnode* プロパティー

msassocnode プロパティー	値	プロパティーの説明
id_field	*field*	データの各トランザクションを特定します。
trans_inputs	*list*	トランザクションデータの入力フィールド。
transactional_target	*field*	予測データ (トランザクション・データ)。

MS Naive Bayes

msbayesnode タイプのノードには、特定のプロパティーが定義されていません。このセクションの冒頭にある共通 Microsoft プロパティーを参照してください。

MS 線型回帰

msregressionnode タイプのノードには、特定のプロパティーが定義されていません。このセクションの冒頭にある共通 Microsoft プロパティーを参照してください。

MS ニューラル・ネットワーク

msneuralnetworknode タイプのノードには、特定のプロパティーが定義されていません。このセクションの冒頭にある共通 Microsoft プロパティーを参照してください。

MS ロジスティック回帰

mslogisticnode タイプのノードには、特定のプロパティーが定義されていません。このセクションの冒頭にある共通 Microsoft プロパティーを参照してください。

MS タイム・シリーズ

mstimeseriesnode タイプのノードには、特定のプロパティーが定義されていません。このセクションの冒頭にある共通 Microsoft プロパティーを参照してください。

MS シーケンス・クラスタリング

次のプロパティーは、mssequenceclusternode タイプのノードで使用できます。

表 *188. mssequenceclusternode properties*

mssequenceclusternode プロパティー	値	プロパティーの説明
id_field	*field*	データの各トランザクションを特定します。
input_fields	*list*	トランザクションデータの入力フィールド。
sequence_field	*field*	シーケンス ID。
target_field	*field*	予測フィールド (テーブル形式データ)。

アルゴリズム・パラメーター

各 Microsoft データベース・モデル・タイプには、msas_parameters プロパティーを使用して設定できる、次のような特定のパラメーターがあります。

```
stream = modeler.script.stream()
msregressionnode = stream.findByType("msregression", None)
msregressionnode.setPropertyValue("msas_parameters", [["MAXIMUM_INPUT_ATTRIBUTES", 255],
["MAXIMUM_OUTPUT_ATTRIBUTES", 255]])
```

これらのパラメーターは SQL Server から取得されます。各ノードに関連するパラメーターを見るには

1. キャンバスにデータベース入力ノードを配置します。
2. データベース入力ノードを開きます。
3. 「データ ソース」 ドロップダウン・リストから有効なソースを選択します。
4. 「テーブル名」 リストから有効なテーブルを選択します。
5. 「**OK**」 をクリックして、データベース入力ノードを閉じます。
6. プロパティーを一覧表示したい Microsoft データベース・モデル作成ノードを追加します。
7. データベース・モデル作成ノードを開きます。
8. 「エキスパート」 タブを選択します。

このノードの使用できる msas_parameters プロパティーが表示されます。

Microsoft モデル・ナゲットのプロパティー

Microsoft データベース・モデル作成ノードを使用して作成されるモデル・ナゲットのプロパティーを、次に示します。

MS デシジョン・ツリー

表 *189. MS* デシジョン・ツリーのプロパティー

applymstreenode プロパティー	値	説明
analysis_database_name	*string*	このノードは、ストリームの中で直接スコアされます。 このプロパティーは Analysis Services データベース名の識別に使用します。
analysis_server_name	*string*	Analysis サーバー・ホストの名前
datasource	*string*	SQL Server の ODBC データ・ソース 名 (DSN) の名前
sql_generate	*flag*	SQL 生成を有効にします。

MS 線型回帰

表 *190. MS* 線型回帰のプロパティー

applymsregressionnode プロパティー	値	説明
analysis_database_name	*string*	このノードは、ストリームの中で直接スコアされます。 このプロパティーは Analysis Services データベース名の識別に使用します。
analysis_server_name	*string*	Analysis サーバー・ホストの名前

MS ニューラル・ネットワーク

表 *191. MS* ニューラル・ネットワークのプロパティー

applymsneuralnetworknode プロパティー	値	説明
analysis_database_name	*string*	このノードは、ストリームの中で直接スコアされます。 このプロパティーは Analysis Services データベース名の識別に使用します。
analysis_server_name	*string*	Analysis サーバー・ホストの名前

MS ロジスティック回帰

表 *192. MS* ロジスティック回帰のプロパティー

applymslogisticnode プロパティー	値	説明
analysis_database_name	*string*	このノードは、ストリームの中で直接スコアされます。 このプロパティーは Analysis Services データベース名の識別に使用します。
analysis_server_name	*string*	Analysis サーバー・ホストの名前

MS タイム・シリーズ

表 *193. MS* タイム・シリーズのプロパティー

applymstimeseriesnode プロパティー	値	説明
analysis_database_name	*string*	このノードは、ストリームの中で直接スコアされます。 このプロパティーは Analysis Services データベース名の識別に使用します。
analysis_server_name	*string*	Analysis サーバー・ホストの名前
start_from	new_prediction historical_prediction	将来の予測を行うか過去の予測を行うかを指定します。
new_step	*number*	将来の予測の開始時間を定義します。
historical_step	*number*	過去の予測の開始時間を定義します。
end_step	*number*	予測の終了時間を定義します。

MS シーケンス・クラスタリング

表 *194. MS* シーケンス・クラスタリングのプロパティー

applymssequenceclusternode プロパティー	値	説明
analysis_database_name	*string*	このノードは、ストリームの中で直接スコアされます。 このプロパティーは Analysis Services データベース名の識別に使用します。
analysis_server_name	*string*	Analysis サーバー・ホストの名前

Oracle モデル作成ノードのプロパティー

Oracle モデル作成ノードのプロパティー

次のプロパティーは、各 Oracle データベース・モデリング・ノードに共通です。

表 *195. Oracle* ノードの共通プロパティー：

一般的な **Oracle** ノードのプロパティー	値	プロパティーの説明
target	*field*	
inputs	*field*のリスト	
partition	*field*	モデル構築の学習、テスト、および検証の各ステージ用に、データを独立したサブセット (サンプル) に分割するフィールド。
datasource		
username		
password		
epassword		
use_model_name	*flag*	
model_name	*string*	ユーザーが指定する新規モデル名。

表 195. *Oracle* ノードの共通プロパティー (続き):

一般的な **Oracle** ノードのプロパティー	値	プロパティーの説明
use_partitioned_data	*flag*	区分フィールドが定義される場合、このオプションは学習データ区分からのデータのみがモデル構築に使用されるようにします。
unique_field	*field*	
auto_data_prep	*flag*	Oracle 自動データ準備機能を有効化または無効化します (11g データベースのみ)。
costs	*structured*	構造化プロパティー、使用形式 : [[drugA drugB 1.5] [drugA drugC 2.1]]。[] 内の引数は実際の予測コストです。
mode	Simple (単純) Expert	Simple に設定されている場合、個々のノード・プロパティーに記述されているように、特定のプロパティーは無視されます。
use_prediction_probability	*flag*	
prediction_probability	*string*	
use_prediction_set	*flag*	

Oracle Naive Bayes

次のプロパティーは、oranbnode タイプのノードで使用できます。

表 196. *oranbnode* プロパティー

oranbnode プロパティー	値	プロパティーの説明
singleton_threshold	*number*	0.0–1.0.*
pairwise_threshold	*number*	0.0–1.0.*
priors	Data Equal Custom	
custom_priors	*structured*	構造化プロパティー、使用形式 : set :oranbnode.custom_priors = [[drugA 1][drugB 2][drugC 3][drugX 4][drugY 5]]

* mode が Simple に設定されている場合、プロパティーは無視されます。

Oracle Adaptive Bayes

次のプロパティーは、oraabnnode タイプのノードで使用できます。

表 197. *oraabnnode* プロパティー

oraabnnode プロパティー	値	プロパティーの説明
model_type	SingleFeature MultiFeature NaiveBayes	
use_execution_time_limit	*flag*	*
execution_time_limit	*integer*	値は 1 以上でなければなりません。*
max_naive_bayes_predictors	*integer*	値は 1 以上でなければなりません。*

表 197. *oraabnnode* プロパティー (続き)

oraabnnode プロパティー	値	プロパティーの説明
max_predictors	*integer*	値は 1 以上でなければなりません。*
priors	Data Equal Custom	
custom_priors	*structured*	構造化プロパティー、使用形式 : set :oraabnnode.custom_priors = [[drugA 1][drugB 2][drugC 3][drugX 4][drugY 5]]

* mode が Simple に設定されている場合、プロパティーは無視されます。

Oracle Support Vector Machines

次のプロパティーは、orasvmnode タイプのノードで使用できます。

表 198. *orasvmnode* プロパティー

orasvmnode プロパティー	値	プロパティーの説明
active_learning	Enable Disable	
kernel_function	Linear Gaussian System	
normalization_method	zscore minmax none	
kernel_cache_size	*integer*	Gaussian カーネル専用。値は 1 以上でなければなりません。*
convergence_tolerance	*number*	値は 1 以上でなければなりません。*
use_standard_deviation	*flag*	Gaussian カーネル専用。*
standard_deviation	*number*	値は 1 以上でなければなりません。*
use_epsilon	*flag*	回帰モデルのみです。*
epsilon	*number*	値は 1 以上でなければなりません。*
use_complexity_factor	*flag*	*
complexity_factor	*number*	*
use_outlier_rate	*flag*	単一バリアントのみです。*
outlier_rate	*number*	単一バリアントのみです。 0.0–1.0.*
weights	Data Equal Custom	
custom_weights	*structured*	構造化プロパティー、使用形式 : set :orasvmnode.custom_weights = [[drugA 1][drugB 2][drugC 3][drugX 4][drugY 5]]

* mode が Simple に設定されている場合、プロパティーは無視されます。

Oracle 一般化線型モデル

次のプロパティーは、oraglmnode タイプのノードで使用できます。

表 199. *oraglmnode* プロパティー

oraglmnode プロパティー	値	プロパティーの説明
normalization_method	zscore minmax none	
missing_value_handling	ReplaceWithMean UseCompleteRecords	
use_row_weights	*flag*	*
row_weights_field	*field*	*
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*	*

* mode が Simple に設定されている場合、プロパティーは無視されます。

Oracle デシジョン・ツリー

次のプロパティーは、oradecisiontreenode タイプのノードで使用できます。

表 200. *oradecisiontreenode* プロパティー

oradecisiontreenode プロパティー	値	プロパティーの説明
use_costs	*flag*	
impurity_metric	Entropy (エントロピー) Gini	
term_max_depth	*integer*	2–20.*
term_minpct_node	*number*	0.0–10.0.*
term_minpct_split	*number*	0.0–20.0.*
term_minrec_node	*integer*	値は 1 以上でなければなりません。*
term_minrec_split	*integer*	値は 1 以上でなければなりません。*
display_rule_ids	*flag*	*

* mode が Simple に設定されている場合、プロパティーは無視されます。

Oracle O-Cluster

次のプロパティーは、oraoclusternode タイプのノードで使用できます。

表 201. *oraoclusternode* プロパティー

oraoclusternode プロパティー	値	プロパティーの説明
max_num_clusters	*integer*	値は 1 以上でなければなりません。*
max_buffer	*integer*	値は 1 以上でなければなりません。*
sensitivity	*number*	0.0–1.0.*

* mode が Simple に設定されている場合、プロパティーは無視されます。

Oracle KMeans

次のプロパティーは、orakmeansnode タイプのノードで使用できます。

表 202. *orakmeansnode* プロパティー

orakmeansnode プロパティー	値	プロパティーの説明
num_clusters	*integer*	値は 1 以上でなければなりません。*
normalization_method	zscore minmax none	
distance_function	Euclidean Cosine	
iterations	*integer*	0–20.*
conv_tolerance	*number*	0.0–0.5.*
split_criterion	Variance Size	デフォルトは Variance です。*
num_bins	*integer*	値は 1 以上でなければなりません。*
block_growth	*integer*	1–5.*
min_pct_attr_support	*number*	0.0–1.0.*

* mode が Simple に設定されている場合、プロパティーは無視されます。

Oracle NMF

次のプロパティーは、oranmfnode タイプのノードで使用できます。

表 203. *oranmfnode* プロパティー

oranmfnode プロパティー	値	プロパティーの説明
normalization_method	minmax none	
use_num_features	*flag*	*
num_features	*integer*	0-1。デフォルト値はアルゴリズムによってデータから推定されます。
random_seed	*number*	*

表 203. *oranmfnode* プロパティー (続き)

oranmfnode プロパティー	値	プロパティーの説明
num_iterations	*integer*	0-500.*
conv_tolerance	*number*	0.0–0.5.*
display_all_features	*flag*	*

* mode が Simple に設定されている場合、プロパティーは無視されます。

Oracle Apriori

次のプロパティーは、oraapriorinode タイプのノードで使用できます。

表 204. *oraapriorinode* プロパティー

oraapriorinode プロパティー	値	プロパティーの説明
content_field	*field*	
id_field	*field*	
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 最小記述長 (MDL)

oramdlnode タイプのノードには、特定のプロパティーが定義されていません。このセクションの冒頭にある共通 Oracle プロパティーを参照してください。

Oracle Attribute Importance (AI)

次のプロパティーは、oraainode タイプのノードで使用できます。

表 205. *oraainode* プロパティー

oraainode プロパティー	値	プロパティーの説明
custom_fields	*flag*	真 (true) の場合は、現在のノードのターゲット、入力、その他フィールドなどを指定することができます。偽 (false) の場合は、上流のデータ型ノードから現在の設定が使用されます。
selection_mode	ImportanceLevel ImportanceValue TopN	
select_important	*flag*	selection_mode が ImportanceLevel に設定されているときに、重要なフィールドを選択するかどうかを指定します。
important_label	*string*	「重要」ランクのラベルを指定します。
select_marginal	*flag*	selection_mode が ImportanceLevel に設定されているときに、境界フィールドを選択するかどうかを指定します。
marginal_label	*string*	「境界」ランクのラベルを指定します。

表 205. *oraainode* プロパティー (続き)

oraainode プロパティー	値	プロパティーの説明
important_above	*number*	0.0–1.0.
select_unimportant	*flag*	selection_mode が ImportanceLevel に設定されているときに、重要でないフィールドを選択するかどうかを指定します。
unimportant_label	*string*	「非重要」ランクのラベルを指定します。
unimportant_below	*number*	0.0–1.0.
importance_value	*number*	selection_mode が ImportanceValue に設定されているときに、使用する分割値を指定します。0 から 100 の値を指定します。
top_n	*number*	selection_mode が TopN に設定されているときに、使用する分割値を指定します。0 から 1000 の値を指定します。

Oracle モデル・ナゲットのプロパティー

Oracle ノードを使用して作成されるモデル・ナゲットのプロパティーを、次に示します。

Oracle Naive Bayes

applyoranbnode タイプのノードには、特定のプロパティーが定義されていません。

Oracle Adaptive Bayes

applyoraabnnode タイプのノードには、特定のプロパティーが定義されていません。

Oracle Support Vector Machines

applyorasvmnode タイプのノードには、特定のプロパティーが定義されていません。

Oracle デシジョン・ツリー

次のプロパティーは、applyoradecisiontreenode タイプのノードで使用できます。

表 206. *applyoradecisiontreenode* プロパティー

applyoradecisiontreenode プロパティー	値	プロパティーの説明
use_costs	*flag*	
display_rule_ids	*flag*	

Oracle O-Cluster

applyoraoclusternode タイプのノードには、特定のプロパティーが定義されていません。

Oracle KMeans

applyorakmeansnode タイプのノードには、特定のプロパティーが定義されていません。

Oracle NMF

次のプロパティーは、applyoranmfnode タイプのノードで使用できます。

表 207. *applyoranmfnode* プロパティー

applyoranmfnode プロパティー	値	プロパティーの説明
display_all_features	*flag*	

Oracle Apriori

このモデル・ナゲットはスクリプトに適用できません。

Oracle MDL

このモデル・ナゲットはスクリプトに適用できません。

IBM Netezza Analytics モデル作成ノードのプロパティー

Netezza モデル作成ノードのプロパティー

次のプロパティーは、各 IBM Netezza データベース・モデリング・ノードに共通です。

表 208. 共通の *Netezza* ノード・プロパティー:

共通の **Netezza** ノード・プロパティー	値	プロパティーの説明
custom_fields	*flag*	真 (true) の場合は、現在のノードのターゲット、入力、その他フィールドなどを指定することができます。偽 (false) の場合は、上流のデータ型ノードから現在の設定が使用されます。
inputs	*[field1 ... fieldN]*	モデルで使用される入力または予測変数フィールド。
target	*field*	対象フィールド (連続型またはカテゴリー型)。
record_id	*field*	一意のレコード ID として使用されるフィールド。
use_upstream_connection	*flag*	true (デフォルト) の場合、 上流のノードで指定された接続の詳細。move_data_to_connection が指定されている場合は使用されません。
move_data_connection	*flag*	true の場合、データは connection に指定されたデータベースに移動します。use_upstream_connection が指定されている場合は使用されません。

表 208. 共通の *Netezza* ノード・プロパティー (続き):

共通の **Netezza** ノード・プロパティー	値	プロパティーの説明
connection	*structured*	モデルが保存される Netezza データベースの接続文字列。構造化プロパティー、使用形式 : ['odbc' '<dsn>' '<username>' '<psw>' '<catname>' '<conn_attribs>' [true|false]] ここで、 <dsn> は データ・ソース名です。 <username> と <psw> は、データベースのユーザー名とパスワードです。 <catname> はカタログ名です。 <conn_attribs> は接続の属性です。 true | false は、パスワードが必要かどうかを示します。
table_name	*string*	モデルが保存されるデータベース・テーブルの名前。
use_model_name	*flag*	true の場合、model_name によって指定された名前をモデルの名前として使用します。そうでない場合、モデル名はシステムによって作成されます。
model_name	*string*	ユーザーが指定する新規モデル名。
include_input_fields	*flag*	true の場合、すべての入力フィールドを下流に渡します。そうでない場合、record_id とモデルによって生成されたフィールドのみが渡されます。

Netezza ディシジョン・ツリー

次のプロパティーは、netezzadectreenode タイプのノードで使用できます。

表 209. *netezzadectreenode* プロパティー

netezzadectreenode プロパティー	値	プロパティーの説明
impurity_measure	Entropy (エントロピー) Gini	ツリーの分割に最も良い場所を評価するのに使用される、不純度の測定。
max_tree_depth	*integer*	ツリーが成長可能な最大レベル数。デフォルトは 62 です (可能な最大値)。
min_improvement_splits	*number*	分割が発生する不純度の改善の最小値。デフォルトは 0.01 です。
min_instances_split	*integer*	分割が発生する前に残る分割されていないレコードの最小数。デフォルトは 2 です (可能な最小値)。
weights	*structured*	クラスの相対的重み。構造化プロパティー、使用形式 : set :netezza_dectree.weights = [[drugA 0.3][drugB 0.6]] デフォルトの重みはすべてのクラスで 1 です。

表 209. *netezzadectreenode* プロパティー (続き)

netezzadectreenode プロパティー	値	プロパティーの説明
pruning_measure	Acc wAcc	デフォルトは Acc (精度) です。wAcc (重み付き精度) は、剪定を適用する際にクラスの重みを考慮します。
prune_tree_options	allTrainingData partitionTrainingData useOtherTable	デフォルトでは、allTrainingData を使用してモデルの精度を推定します。partitionTrainingData を使用して、使用する学習データの割合を、useOtherTable を使用して指定したデータベース・テーブルの学習データ・セットを使用します。
perc_training_data	*number*	prune_tree_options が partitionTrainingData に設定されている場合、学習に使用するデータの割合を指定します。
prune_seed	*integer*	prune_tree_options が partitionTrainingData に設定されている場合、分析結果を再現に使用するランダム・シード。デフォルトは 1 です。
pruning_table	*string*	モデルの精度を推定するために個別の剪定データセットのテーブル名。
compute_probabilities	*flag*	true の場合、予測フィールドのほか、確信度 (確率) フィールドを生成します。

Netezza K-Means

次のプロパティーは、netezzakmeansnode タイプのノードで使用できます。

表 210. *netezzakmeansnode properties*

netezzakmeansnode プロパティー	値	プロパティーの説明
distance_measure	Euclidean Manhattan Canberra maximum	データ・ポイント間の教理を測定する方法。
num_clusters	*integer*	作成するクラスター数。デフォルトは 3。
max_iterations	*integer*	モデルの学習を停止する前のアルゴリズムの反復数。デフォルトは 5。
rand_seed	*integer*	分析結果の反復に使用するランダム・シード。デフォルトは 12345。

Netezza ベイズ・ネットワーク

次のプロパティーは、netezzabayesnode タイプのノードで使用できます。

表 211. *netezzabayesnode* プロパティー

netezzabayesnode プロパティー	値	プロパティーの説明
base_index	*integer*	内部管理の最初の入力フィールドに割り当てられる数値の識別子。デフォルトは 777。
sample_size	*integer*	属性の値が非常に大きい場合に最小するサンプルのサイズ。デフォルトは 10,000。
display_additional_information	*flag*	true の場合、メッセージのダイアログ・ボックスに追加の進捗状況の情報を表示します。
type_of_prediction	best neighbors nn-neighbors	使用する予測アルゴリズムの種類: 最適 (相関度が最も高い近傍)、近傍 (近傍の重み付き予測)、NN 近傍 (null 以外の近傍)。

Netezza Naive Bayes

次のプロパティーは、netezzanaivebayesnode タイプのノードで使用できます。

表 212. *netezzanaivebayesnode* プロパティー

netezzanaivebayesnode プロパティー	値	プロパティーの説明
compute_probabilities	*flag*	true の場合、予測フィールドのほか、確信度 (確率) フィールドを生成します。
use_m_estimation	*flag*	true の場合、推定時に 0 の確立を回避する m 推定方法を使用します。

Netezza KNN

次のプロパティーは、netezzaknnnode タイプのノードで使用できます。

表 213. *netezzaknnnode* プロパティー

netezzaknnnode プロパティー	値	プロパティーの説明
weights	*structured*	重みを各クラスに割り当てる構造化プロパティー。例: set :netezzaknnnode.weights = [[drugA 0.3][drugB 0.6]]
distance_measure	Euclidean Manhattan Canberra Maximum	データ・ポイント間の教理を測定する方法。
num_nearest_neighbors	*integer*	特定のケースの最近傍数。デフォルトは 3。
standardize_measurements	*flag*	true の場合、距離の値を計算する前に連続型入力フィールドの測定を標準化します。
use_coresets	*flag*	true の場合、大規模なデータセットに対して計算を高速化するコアセット・サンプリングを使用しています

Netezza 分裂クラスタリング

次のプロパティーは、netezzadivclusternode タイプのノードで使用できます。

表 214. *netezzadivclusternode* プロパティー

netezzadivclusternode プロパティー	値	プロパティーの説明
distance_measure	Euclidean Manhattan Canberra Maximum	データ・ポイント間の教理を測定する方法。
max_iterations	*integer*	モデルの学習が停止する前に、実行するアルゴリズム反復の最大回数。デフォルトは 5 です。
max_tree_depth	*integer*	データセットを分割することができるレベルの最大数。デフォルトは 3 です。
rand_seed	*integer*	分析を複製するために使用されるランダムシード。デフォルトは 12345。
min_instances_split	*integer*	分割可能な最小レコード数。デフォルトは 5。
level	*integer*	レコードをスコアリングする階層レベル。デフォルトは -1。

Netezza PCA

次のプロパティーは、netezzapcanode タイプのノードで使用できます。

表 215. *netezzapcanode* プロパティー

netezzapcanode プロパティー	値	プロパティーの説明
center_data	*flag*	true（デフォルト）の場合、このオプションをチェックした場合、分析前にデータのセンタリングを（または「平均値減算」）を実行します。
perform_data_scaling	*flag*	true の場合、分析前にデータのスケーリングを行います。そうすることで、別の変数が異なる単位で測定されるとき、分析が恣意的でないようにします。
force_eigensolve	*flag*	true の場合、を計算する精度が低くなってもより高速な方法を使用します。
pc_number	*integer*	データセットを減少する主要成分の数。デフォルトは 1。

Netezza 回帰ツリー

次のプロパティーは、netezzaregtreenode タイプのノードで使用できます。

表 216. *netezzaregtreenode* プロパティー

netezzaregtreenode プロパティー	値	プロパティーの説明
max_tree_depth	*integer*	ルート・ノードの前にツリーが成長できるレベルの最大数。デフォルトは 10 です。

表 216. *netezzaregtreenode* プロパティー (続き)

netezzaregtreenode プロパティー	値	プロパティーの説明
split_evaluation_measure	分散	ツリーを分割するのに最適な場所を評価するために使用される、クラスの不純度の測定。デフォルト (現在唯一のオプション) は Variance。
min_improvement_splits	*number*	ツリー内に新しい分割が作成される前に純度を減少させる最小数。
min_instances_split	*integer*	分割可能な最小レコード数。
pruning_measure	mse r2 pearson spearman	剪定に使用する方法
prune_tree_options	allTrainingData partitionTrainingData useOtherTable	デフォルトでは、allTrainingData を使用してモデルの精度を推定します。partitionTrainingData を使用して、使用する学習データの割合を、useOtherTable を使用して指定したデータベース・テーブルの学習データ・セットを使用します。
perc_training_data	*number*	prune_tree_options が PercTrainingData に設定されている場合、学習に使用するデータの割合を指定します。
prune_seed	*integer*	prune_tree_options が PercTrainingData に設定されている場合、分析結果を再現に使用するランダム・シード。デフォルトは 1 です。
pruning_table	*string*	モデルの精度を推定するために個別の剪定データセットのテーブル名。
compute_probabilities	*flag*	true の場合、割り当てられたクラスの分散が出力に含まれるべきかどうかを指定ます。

Netezza 線型回帰

次のプロパティーは、netezzalineregressionnode タイプのノードで使用できます。

表 217. *netezzalineregressionnode* プロパティー

netezzalineregressionnode プロパティー	値	プロパティーの説明
use_svd	*flag*	true の場合、元のマトリックスの代わりに特異値分解マトリックスを使用して速度と数値の精度を向上させます。
include_intercept	*flag*	true (デフォルト) の場合、ソリューションの全体の精度が向上します。
calculate_model_diagnostics	*flag*	true の場合、モデルの診断を計算します。

Netezza 時系列

次のプロパティーは、netezzatimeseriesnode タイプのノードで使用できます。

表 218. *netezzatimeseriesnode* プロパティー

netezzatimeseriesnode プロパティー	値	プロパティーの説明
time_points	*field*	時系列の日付または時刻の値を含む入力フィールド。
time_series_ids	*field*	時系列 ID を含むフィールド。入力に複数の時系列が含まれる場合に使用します。
model_table	*field*	Netezza 時系列モデルが保存されるデータベース・テーブルの名前。
description_table	*field*	時系列名および説明を含む入力テーブルの名前。
seasonal_adjustment_table	*field*	指数平滑化または季節的傾向分解アルゴリズムによって計算された季節性調整値を保存する出力テーブル名。
algorithm_name	SpectralAnalysis または spectral ExponentialSmoothing または esmoothing ARIMA SeasonalTrendDecomposition または std	時系列モデリングに使用するアルゴリズム
trend_name	N A DA M DM	指数平滑化の傾向タイプ。 N - none A - 付加 DA - 付加減衰 M - 倍数 DM - 倍数減衰
seasonality_type	N A M	指数平滑化の季節性タイプ。 N - none A - 付加 M - 倍数
interpolation_method	linear cubicspline exponentialspline	使用する補間方法。
timerange_setting	SD SP	使用する時間範囲の設定。 SD - システム決定 (時系列データの全範囲を使用) SP - earliest_time および latest_time を使用したユーザー指定

表 218. *netezzatimeseriesnode* プロパティー (続き)

netezzatimeseriesnode プロパティー	値	プロパティーの説明
earliest_time latest_time	*integer* *date* *time* *timestamp*	timerange_setting が SP の場合の開始値および終了値。 形式は、time_points 値に従う必要があります。 例えば、time_points フィールドに日付が含まれる場合は、これも日付とする必要があります。 例: set NZ_DT1.timerange_setting = 'SP' set NZ_DT1.earliest_time = '1921-01-01' set NZ_DT1.latest_time = '2121-01-01'
arima_setting	SD SP	ARIMA アルゴリズムの設定 (algorithm_name が ARIMA に設定されている場合にのみ使用されます)。 SD - system-determined SP - user-specified arima_setting = SP の場合、次のパラメーターを使用して季節性の値および非季節性の値を設定します。例 (非季節性のみ): 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 - p、d、q、sp、sd および sq パラメーターの演算子です。 less - より小さい eq - 等しい lesseq - 次の値以下
p	*integer*	ARIMA - 自己相関の非季節性の度合い。
q	*integer*	ARIMA - 自己相関の非季節性導出値。
d	*integer*	ARIMA - モデル内の移動平均の非季節性数値。
sp	*integer*	ARIMA - 自己相関の季節性の度合い。

表 218. *netezzatimeseriesnode* プロパティー (続き)

netezzatimeseriesnode プロパティー	値	プロパティーの説明
sq	*integer*	ARIMA - 自己相関の季節性導出値。
sd	*integer*	ARIMA - モデル内の移動平均の季節性数値。
advanced_setting	SD SP	詳細設定の処理方法を決定します。 SD - system-determined SP - period、units_period および forecast_setting を使用したユーザー指定。 例: set NZ_DT1.advanced_setting = 'SP' set NZ_DT1.period = 5 set NZ_DT1.units_period = 'd'
period	*integer*	units_period と組み合わせて指定した季節性サイクルの長さ。スペクトル解析には適用できません。
units_period	ms s min h d wk q y	period が表現される単位。 ms - ミリ秒 s - 秒 min - 分 h - 時 d - 日 wk - 週 q - quarters y - years 例えば、1 週間は period に 1、units_period に wk を指定します。
forecast_setting	forecasthorizon forecasttimes	予測の作成方法を指定します。
forecast_horizon	*integer* *date* *time* *timestamp*	forecast_setting = forecasthorizon である場合、予測の終点の値を指定します。 形式は、time_points 値に従う必要があります。 例えば、time_points フィールドに日付が含まれる場合は、これも日付とする必要があります。
forecast_times	*integer* *date* *time* *timestamp*	forecast_setting = forecasttimes の場合、予測を作成するために使用する値を指定します。 形式は、time_points 値に従う必要があります。 例えば、time_points フィールドに日付が含まれる場合は、これも日付とする必要があります。
include_history	*flag*	過去の値を出力に含めるかどうかを示します。

表 218. *netezzatimeseriesnode* プロパティー (続き)

netezzatimeseriesnode プロパティー	値	プロパティーの説明
include_interpolated_values	*flag*	補間されたの値を出力に含めるかどうかを示します。include_history が false の場合は使用されません。

Netezza 一般化線型

次のプロパティーは、netezzaglmnode タイプのノードで使用できます。

表 219. *netezzaglmnode* プロパティー

netezzaglmnode プロパティー	値	プロパティーの説明
dist_family	bernoulli gaussian poisson negativebinomial wald gamma	分布のタイプ。デフォルトは bernoulli です。
dist_params	*number*	使用する分布パラメーター値。distribution が Negativebinomial の場合のみ適用されます。
trials	*integer*	distribution が Binomial の場合のみ適用されます。ターゲット応答が一連の試行が発生するさまざまなイベントの場合、target フィールドにはイベント数、trials フィールドには試行回数が含まれます。
model_table	*field*	Netezza 一般化線型モデルが保存されるデータベース・テーブルの名前。
maxit	*integer*	アルゴリズムが実行できる反復の最大回数。デフォルトは 20 です。
eps	*number*	アルゴリズムが適合度モデルの検索を停止する最大誤差の値 (科学的表記)。デフォルトは -3、つまり 1E-3 または 0.001 です。
tol	*number*	誤差が 0 として扱われる値 (科学的表記)。デフォルトは -7、つまり 1E-7 (または 0.0000001) を下回る誤差の値が有意でないとカウントされます。

表 219. *netezzaglmnode* プロパティー (続き)

netezzaglmnode プロパティー	値	プロパティーの説明
link_func	識別 inverse invnegative invsquare sqrt power oddspower log clog loglog cloglog logit probit gaussit cauchit canbinom cangeom cannegbinom	使用するリンク関数。デフォルトは logit です。
link_params	*number*	使用するリンク関数パラメーター値。link_function が power または oddspower の場合のみ適用されます。
interaction	[[[*colnames1*],[*levels1*]], [[*colnames2*],[*levels2*]], ...,[[*colnamesN*],[*levelsN*]],]	フィールド間の交互作用を指定します。*colnames* は、入力フィールドのリストです。また、各フィールドの *level* は常に 0 です。 例: [[["K","BP","Sex","K"],[0,0,0,0]], [["Age","Na"],[0,0]]]
intercept	*flag*	true の場合、モデルに定数項を含みます。

Netezza モデル・ナゲットのプロパティー

次のプロパティーは、Netezza データベース・モデリング ナゲットに共通です。

表 220. *Netezza* モデル・ナゲットの共通プロパティー

Netezza モデル・ナゲットの共通プロパティー	値	プロパティーの説明
connection	*string*	モデルが保存される Netezza データベースの接続文字列。
table_name	*string*	モデルが保存されるデータベース・テーブルの名前。

他のモデルナゲットのプロパティーは、対応するモデリングのノードの場合と同じです。

モデル・ナゲットのスクリプト名は以下の通りです。

表 221. *Netezza* モデル・ナゲットのスクリプト名

モデル・ナゲット	スクリプト名
デシジョン・ツリー	applynetezzadectreenode
K-Means	applynetezzakmeansnode
ベイズ・ネット	applynetezzabayesnode
Naive Bayes	applynetezzanaivebayesnode
KNN	applynetezzaknnnode
分裂クラスタリング	applynetezzadivclusternode
PCA	applynetezzapcanode
回帰ツリー	applynetezzaregtreenode
線型回帰	applynetezzalineregressionnode
時系列	applynetezzatimeseriesnode
一般化線型	applynetezzaglmnode

第 16 章 出力ノードのプロパティー

出力ノードのプロパティーは、ほかの種類のノードのプロパティーと少し異なっています。出力ノードのプロパティーは、特定のノード・オプションを参照するというよりは、参照を出力オブジェクトに格納します。このことはテーブルから値を取得して、それをストリーム・パラメーターとして設定するような場合などに役立ちます。

このセクションで、出力ノードで使用できるスクリプト用のプロパティーを説明します。

analysisnode プロパティー

精度分析ノードで、予測モデルの能力を評価して正確な予測を生成します。分析ノードでは、1 つ以上のモデル・ナゲットについて、予測値と実際値をさまざまな方法で比較します。また、分析ノードでは予測モデル同士を比較できます。

例

```
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")
```

表 222. *analysisnode* プロパティー:

analysisnode プロパティー	データ型	プロパティーの説明
output_mode	Screen File	出力ノードから生成される出力の、出力先を指定します。
use_output_name	*flag*	ユーザー設定の出力名が使用されるかどうかを指定します。
output_name	*string*	use_output_name が真 (true) のときに、使用する名前を指定します。
output_format	Text (*.txt*) HTML (*.html*) Output (*.cou*)	出力のタイプを指定します。
by_fields	*list*	

表 222. *analysisnode* プロパティー (続き):

analysisnode プロパティー	データ型	プロパティーの説明
full_filename	*string*	ディスク、データ、または HTML の出力を選択した場合の、出力ファイルの名前。
coincidence	*flag*	
performance	*flag*	
evaluation_binary	*flag*	
confidence	*flag*	
threshold (しきい値)	*number*	
improve_accuracy	*number*	
inc_user_measure	*flag*	
user_if	*expr*	
user_then	*expr*	
user_else	*expr*	
user_compute	[Mean Sum Min Max SDev]	

dataauditnode プロパティー

データ検査ノードでは、欠損値、外れ値、および極値に関する情報の他、各フィールドの要約統計量、ヒストグラムや棒グラフを含む、データを広範に検査するための手段を提供しています。結果は把握しやすい行列形式で表示され、ソートしたり、フルサイズのグラフやデータ準備ノードを生成することができます。

例

```
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")
```

表 223. *dataauditnode* プロパティー:

dataauditnode プロパティー	データ型	プロパティーの説明
custom_fields	*flag*	
fields	*[field1 ... fieldN]*	
overlay	*field*	

表 223. *dataauditnode* プロパティー (続き):

dataauditnode プロパティー	データ型	プロパティーの説明
display_graphs	*flag*	出力行列中のグラフ表示をオンまたはオフにするために使用されます。
basic_stats	*flag*	
advanced_stats	*flag*	
median_stats	*flag*	
calculate	Count Breakdown	欠損値の計算に使用します。計算方法のいずれか、または両方を選択するか、またはどちらも選択しません。
outlier_detection_method	std iqr	外れ値および極値の検出方法を指定します。
outlier_detection_std_outlier	*number*	outlier_detection_method が std の場合、外れ値の定義に使用する数値を指定します。
outlier_detection_std_extreme	*number*	outlier_detection_method が std の場合、外れ値の定義に使用する数値を指定します。
outlier_detection_iqr_outlier	*number*	outlier_detection_method が iqr の場合、外れ値の定義に使用する数値を指定します。
outlier_detection_iqr_extreme	*number*	outlier_detection_method が iqr の場合、外れ値の定義に使用する数値を指定します。
use_output_name	*flag*	ユーザー設定の出力名が使用されるかどうかを指定します。
output_name	*string*	use_output_name が真 (true) のときに、使用する名前を指定します。
output_mode	Screen File	出力ノードから生成される出力の、出力先を指定します。
output_format	Formatted (*.tab*) Delimited (*.csv*) HTML (*.html*) Output (*.cou*)	出力のタイプを指定します。
paginate_output	*flag*	output_format が HTML の場合、出力がページに分割されるようにします。
lines_per_page	*number*	paginate_output と共に使用する場合は、出力ページあたりの行数を指定します。
full_filename	*string*	

extensionoutputnode プロパティー

拡張出力ノードでは、独自のカスタム R スクリプトまたは Python for Spark スクリプトを使用して、データおよびモデル・スコアリングの結果を分析できます。分析はテキストまたはグラフィックで出力できます。出力はマネージャー領域の「出力」タブに追加されます。あるいは、出力をファイルにリダイレクトできます。

Python for Spark の例

```
#### 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)
```

R の例

```
#### script example for R
node.setPropertyValue("syntax_type", "R")
node.setPropertyValue("r_syntax", "print(modelerData$Age)")
```

表 224. *extensionoutputnode* プロパティー

extensionoutputnode プロパティー	データ型	プロパティーの説明
syntax_type	*R* *Python*	R または Python のどちらのスクリプトを実行するか指定します (R がデフォルトです)。
r_syntax	*string*	モデル・スコアリング用の R スクリプト・シンタックス。
python_syntax	*string*	モデル・スコアリング用の Python スクリプト・シンタックス。
convert_flags	StringsAndDoubles LogicalValues	フラグ型フィールドを変換するためのオプション。
convert_missing	*flag*	欠損値を R の NA 値に変換するためのオプション。
convert_datetime	*flag*	日付形式または日付/時刻形式の変数を R の日付/時刻形式に変換するためのオプション。
convert_datetime_class	POSIXct POSIXlt	日付形式または日付/時刻形式の変数のうち、どの形式の変数を変換するかを指定するためのオプション。

表 224. *extensionoutputnode* プロパティー (続き)

extensionoutputnode プロパティー	データ型	プロパティーの説明
output_to	Screen File	出力形式 (Screen または File) を指定します。
output_type	Graph Text	グラフィカル出力またはテキスト出力のどちらを作成するか指定します。
full_filename	*string*	生成された出力に使用するファイル名。
graph_file_type	HTML COU	出力ファイルのファイルの種類 (.html または .cou)。
text_file_type	HTML TEXT COU	テキスト出力のファイルの種類 (.html、.txt、または .cou) を指定します。

matrixnode プロパティー

クロス集計ノードで、フィールド間の関係を示すテーブルを作成します。一般的にこのノードは、2 つのシンボル値フィールドの関係を示す場合によく使用されますが、フラグ型フィールド間または数値型フィールド間の関係を示すこともできます。

例

```
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)
```

表 225. *matrixnode* プロパティー:

matrixnode プロパティー	データ型	プロパティーの説明
fields	Selected Flags Numerics	
row	*field*	

表 225. *matrixnode* プロパティー (続き):

matrixnode プロパティー	データ型	プロパティーの説明
column	*field*	
include_missing_values	*flag*	ユーザーによる欠損値 (空白) とシステムによる欠損値 (ヌル) が、行と列の出力に含まれるかどうかを指定します。
cell_contents	CrossTabs Function	
function_field	*string*	
function	Sum Mean Min Max SDev	
sort_mode	Unsorted Ascending Descending	
highlight_top	*number*	ゼロでない場合に真 (true) 。
highlight_bottom	*number*	ゼロでない場合に真 (true) 。
display	[Counts Expected Residuals (残差) RowPct ColumnPct TotalPct]	
include_totals	*flag*	
use_output_name	*flag*	ユーザー設定の出力名が使用されるかどうかを指定します。
output_name	*string*	use_output_name が真 (true) のときに、使用する名前を指定します。
output_mode	Screen File	出力ノードから生成される出力の、出力先を指定します。
output_format	Formatted (*.tab*) Delimited (*.csv*) HTML (*.html*) Output (*.cou*)	出力のタイプを指定します。Formatted と Delimited の両方が、テーブル内で行と列を入れ替える修飾子 transposed を伴うことができます。
paginate_output	*flag*	output_format が HTML の場合、出力がページに分割されるようにします。
lines_per_page	*number*	paginate_output と共に使用する場合は、出力ページあたりの行数を指定します。
full_filename	*string*	

meansnode プロパティー

平均比較ノードでは、独立したグループ間で、または関連するフィールドのペア間で著しい違いがあるかどうかを調べるために、平均を比較します。例えば、販売促進活動の前後で平均収益を比較したり、販売促進活動を受けなかった顧客と受けた顧客からの収益を比較することができます。

例

```
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")
```

表 226. *meansnode* プロパティー：

meansnode プロパティー	データ型	プロパティーの説明
means_mode	BetweenGroups BetweenFields	データに実行する平均統計処理の種類を指定します。
test_fields	[field1 ... fieldn]	means_mode が BetweenGroups に設定されているときのテスト・フィールドを指定します。
grouping_field	*field*	グループにまとめるフィールドを指定します。
paired_fields	[[field1 field2] [field3 field4] ...]	means_mode が BetweenFields に設定されているときに使用するフィールドのペアを指定します。
label_correlations	*flag*	相関ラベルが出力に表示されるかどうかを指定します。means_mode が BetweenFields に設定されているときにのみ、この設定が適用されます。
correlation_mode	Probability Absolute	確率 (Probability) または絶対値 (Absolute) のどちらかで相関にラベルを付けることを指定します。
weak_label	*string*	
medium_label	*string*	
strong_label	*string*	
weak_below_probability	*number*	correlation_mode が Probability に設定されているときに、弱い相関の分割値を指定します。この値は、例えば 0.90 のように、0 と 1 の間にする必要があります。
strong_above_probability	*number*	強い相関の分割値。

表 226. *meansnode* プロパティー (続き):

meansnode プロパティー	データ型	プロパティーの説明
weak_below_absolute	*number*	correlation_mode が Absolute に設定されているときに、弱い相関の分割値を指定します。この値は、例えば 0.90 のように、0 と 1 の間にする必要があります。
strong_above_absolute	*number*	強い相関の分割値。
unimportant_label	*string*	
marginal_label	*string*	
important_label	*string*	
unimportant_below	*number*	低いフィールド重要度の分割値。この値は、例えば 0.90 のように、0 と 1 の間にする必要があります。
important_above	*number*	
use_output_name	*flag*	ユーザー設定の出力名が使用されるかどうかを指定します。
output_name	*string*	使用する名前。
output_mode	Screen File	出力ノードから生成された出力の出力先を指定します。
output_format	Formatted (*.tab*) Delimited (*.csv*) HTML (*.html*) Output (*.cou*)	出力のタイプを指定します。
full_filename	*string*	
output_view	Simple Advanced	出力に単純な (Simple) ビューが表示されるか、または詳細な (Advanced) ビューが表示されるかを指定します。

reportnode プロパティー

レポート・ノードで、固定テキスト、およびデータやデータから導かれた他の式を含む、フォーマット済みレポートを作成します。レポートの書式は、固定テキストとデータの出力構成を定義するテキスト テンプレートを使用して指定します。テンプレート内の HTML タグを使用し、また「出力」タブでオプションを設定することで、カスタムのテキスト書式設定を提供できます。テンプレート内の CLEM 式を使用して、データ値やその他の条件出力を含めることができます。

例

```
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)
```

表 227. *reportnode* プロパティー：

reportnode プロパティー	データ型	プロパティーの説明
output_mode	Screen File	出力ノードから生成される出力の、出力先を指定します。
output_format	HTML (*.html*) Text (*.txt*) Output (*.cou*)	ファイル出力のタイプを指定します。
形式	Auto Custom	出力を自動的にフォーマット設定するか、テンプレートに含まれる HTML を使用してフォーマット設定するかを選択するために使用します。テンプレート内の HTML フォーマット設定を使用するには、Custom を指定します。
use_output_name	*flag*	ユーザー設定の出力名が使用されるかどうかを指定します。
output_name	*string*	use_output_name が真 (true) のときに、使用する名前を指定します。
text	*string*	
full_filename	*string*	
highlights	*flag*	
title	*string*	
lines_per_page	*number*	

routputnode のプロパティー

R 出力ノードでは、独自のカスタム R スクリプトを使用して、データおよびモデル・スコアリングの結果を分析できます。分析はテキストまたはグラフィックで出力できます。出力はマネージャー領域の「出力」タブに追加されます。あるいは、出力をファイルにリダイレクトできます。

表 228. *routputnode* のプロパティー

routputnode のプロパティー	データ型	プロパティーの説明
syntax	*string*	
convert_flags	StringsAndDoubles LogicalValues	
convert_datetime	*flag*	
convert_datetime_class	POSIXct POSIXlt	
convert_missing	*flag*	
output_name	Auto Custom	
custom_name	*string*	
output_to	Screen File	

表 228. *routputnode* のプロパティー (続き)

routputnode のプロパティー	データ型	プロパティーの説明
output_type	Graph Text	
full_filename	*string*	
graph_file_type	HTML COU	
text_file_type	HTML TEXT COU	

setglobalsnode プロパティー

グローバル・ノードで、データを走査し、CLEM 式で使用できる要約値を算出します。例えば、グローバル・ノードを使用して、「年齢」 という名前のフィールドの統計量を算出し、次に CLEM 式に @GLOBAL_MEAN(年齢) 関数を挿入して年齢 の全体的な平均を算出することができます。

例

```
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)
```

表 229. *setglobalsnode* プロパティー:

setglobalsnode プロパティー	データ型	プロパティーの説明
globals	[Sum Mean Min Max SDev]	フィールドを設定する構造化プロパティーは、次の形式で参照する必要があります。 node.setKeyedPropertyValue("globals", "Age", ["Max", "Sum", "Mean", "SDev"])
clear_first	*flag*	
show_preview	*flag*	

simevalnode プロパティー

シミュレーション評価ノードは、指定された予測される対象フィールドを評価し、対象フィールドの分布と相関情報を提供します。

表 230. *simevalnode* プロパティー：

simevalnode プロパティー	データ型	プロパティーの説明
target	*field*	
iteration	*field*	
presorted_by_iteration	*boolean*	
max_iterations	*number*	
tornado_fields	*[field1...fieldN]*	
plot_pdf	*boolean*	
plot_cdf	*boolean*	
show_ref_mean	*boolean*	
show_ref_median	*boolean*	
show_ref_sigma	*boolean*	
num_ref_sigma	*number*	
show_ref_pct	*boolean*	
ref_pct_bottom	*number*	
ref_pct_top	*number*	
show_ref_custom	*boolean*	
ref_custom_values	*[number1...numberN]*	
category_values	Category Probabilities Both	
category_groups	Categories Iterations	
create_pct_table	*boolean*	
pct_table	Quartiles Intervals Custom	
pct_intervals_num	*number*	
pct_custom_values	*[number1...numberN]*	

simfitnode プロパティー

シミュレーション・フィッティング・ノードは、各フィールドのデータの統計的な分布を調べ、最も適合する分布を各フィールドに割り当ててシミュレーション生成ノードを生成 (または更新) します。この後、シミュレーション生成ノードを使用して、シミュレートするデータを生成することができます。

表 231. *simfitnode* プロパティー：

simfitnode プロパティー	データ型	プロパティーの説明
build	Node XMLExport Both	
use_source_node_name	*boolean*	

表 231. *simfitnode* プロパティー (続き):

simfitnode プロパティー	データ型	プロパティーの説明
source_node_name	*string*	生成または更新される入力ノードのカスタム名。
use_cases	すべて LimitFirstN	
use_case_limit	*integer*	
fit_criterion	AndersonDarling KolmogorovSmirnov	
num_bins	*integer*	
parameter_xml_filename	*string*	
generate_parameter_import	*boolean*	

statisticsnode プロパティー

記述統計ノードでは、数値型フィールドに関する基本的な集計情報が提供されます。このノードで、個々のフィールドの要約統計量とフィールド間の相関が計算されます。

例

```
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")
```

表 232. *statisticsnode* プロパティー:

statisticsnode プロパティー	データ型	プロパティーの説明
use_output_name	*flag*	ユーザー設定の出力名が使用されるかどうかを指定します。
output_name	*string*	use_output_name が真 (true) のときに、使用する名前を指定します。
output_mode	Screen File	出力ノードから生成される出力の、出力先を指定します。
output_format	Text (*.txt*) HTML (*.html*) Output (*.cou*)	出力のタイプを指定します。

表 232. *statisticsnode* プロパティー (続き):

statisticsnode プロパティー	データ型	プロパティーの説明
full_filename	*string*	
examine	*list*	
correlate	*list*	
statistics	[count mean sum min max range variance sdev semean median mode]	
correlation_mode	Probability Absolute	確率 (Probability) または絶対値 (Absolute) のどちらかで相関にラベルを付けることを指定します。
label_correlations	*flag*	
weak_label	*string*	
medium_label	*string*	
strong_label	*string*	
weak_below_probability	*number*	correlation_mode が Probability に設定されているときに、弱い相関の分割値を指定します。この値は、例えば 0.90 のように、0 と 1 の間にする必要があります。
strong_above_probability	*number*	強い相関の分割値。
weak_below_absolute	*number*	correlation_mode が Absolute に設定されているときに、弱い相関の分割値を指定します。この値は、例えば 0.90 のように、0 と 1 の間にする必要があります。
strong_above_absolute	*number*	強い相関の分割値。

statisticsoutputnode プロパティー

Statistics 出力ノードを使用すると、IBM SPSS Statistics 手続きを呼び出し、IBM SPSS Modeler データを分析することができます。さまざまな IBM SPSS Statistics 分析手続きにアクセスできます。このノードは、ライセンスが与えられた IBM SPSS Statistics のコピーが必要です。

このノードのプロパティーについては、 357 ページの『statisticsoutputnode プロパティー』に記載されています。

tablenode プロパティー

テーブル・ノードで、データがテーブル形式で表示されます。このデータは、ファイルにも書き込めます。この機能は、データの値を調査したり、データを読みやすい形式でエクスポートする必要がある場合に役立ちます。

例

```
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)
```

表 233. *tablenode* プロパティー：

tablenode プロパティー	データ型	プロパティーの説明
full_filename	*string*	ディスク、データ、または HTML の出力を選択した場合の、出力ファイルの名前。
use_output_name	*flag*	ユーザー設定の出力名が使用されるかどうかを指定します。
output_name	*string*	use_output_name が真 (true) のときに、使用する名前を指定します。
output_mode	Screen File	出力ノードから生成される出力の、出力先を指定します。
output_format	Formatted (*.tab*) Delimited (*.csv*) HTML (*.html*) Output (*.cou*)	出力のタイプを指定します。
transpose_data	*flag*	エクスポート前にデータの行列を入れ替えて、行がフィールドを、列がレコードを表すようにします。
paginate_output	*flag*	output_format が HTML の場合、出力がページに分割されるようにします。
lines_per_page	*number*	paginate_output と共に使用する場合は、出力ページあたりの行数を指定します。
highlight_expr	*string*	
output	*string*	ノードで直前に構築されたテーブルへの参照を保持する、読み取り専用プロパティー。
value_labels	*[[Value LabelString] [Value LabelString] ...]*	値のペアのためのラベルを指定します。

表 233. *tablenode* プロパティー (続き):

tablenode プロパティー	データ型	プロパティーの説明
display_places	*integer*	フィールドが表示されるときの小数部の桁数を設定します (REAL ストレージのフィールドにのみ適用)。-1 を設定すると、ストリームのデフォルトが使用されます。
export_places	*integer*	フィールドが出力されるときの小数部の桁数を設定します (REAL ストレージのフィールドにのみ適用)。-1 を設定すると、ストリームのデフォルトが使用されます。
decimal_separator	DEFAULT PERIOD COMMA	フィールドの小数点記号を指定します (REAL ストレージのフィールドにのみ適用)。
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	フィールドの日付形式を設定します (DATE または TIMESTAMP ストレージのフィールドにのみ適用されます)。

表 233. *tablenode* プロパティー (続き):

tablenode プロパティー	データ型	プロパティーの説明
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"	フィールドの日付形式を設定します (TIME または TIMESTAMP ストレージのフィールドにのみ適用されます)。
column_width	*integer*	フィールドに列幅を設定します。-1 という値を指定すると、列幅は Auto に設定されます。
justify	AUTO CENTER LEFT RIGHT	フィールドに列調整を設定します。

transformnode プロパティー

変換ノードによって、選択フィールドに適用する前に変換の結果を選択し、視覚的に確認することができます。

例

```
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)
```

表 234. *transformnode* プロパティー:

transformnode プロパティー	データ型	プロパティーの説明
fields	[*field1... fieldn*]	変換で使用するフィールド。
formula	All Select	すべての変換を計算するか、選択した変換を計算するかを指定します。
formula_inverse	*flag*	逆変換を使用するかどうかを指定します。

表 234. *transformnode* プロパティー (続き):

transformnode プロパティー	データ型	プロパティーの説明
formula_inverse_offset	*number*	式で使用するデータ・オフセットを指定します。ユーザーが指定しない限り、デフォルトで 0 に設定されます。
formula_log_n	*flag*	log $_n$ 変換を使用するかどうかを指定します。
formula_log_n_offset	*number*	
formula_log_10	*flag*	$\log_{10}$ 変換を使用するかどうかを指定します。
formula_log_10_offset	*number*	
formula_exponential	*flag*	指数変換 (e^x) を使用するかどうかを指定します。
formula_square_root	*flag*	平方根変換を使用するかどうかを指定します。
use_output_name	*flag*	ユーザー設定の出力名が使用されるかどうかを指定します。
output_name	*string*	use_output_name が真 (true) のときに、使用する名前を指定します。
output_mode	Screen File	出力ノードから生成される出力の、出力先を指定します。
output_format	HTML (*.html*) Output (*.cou*)	出力のタイプを指定します。
paginate_output	*flag*	output_format が HTML の場合、出力がページに分割されるようにします。
lines_per_page	*number*	paginate_output と共に使用する場合は、出力ページあたりの行数を指定します。
full_filename	*string*	ファイル出力に使用するファイル名を指定します。

第 17 章 エクスポート・ノードのプロパティー

共通のエクスポート・ノード・プロパティー

次のプロパティーは、すべてのエクスポート・ノードに共通しています。

表 235. 共通のエクスポート・ノード・プロパティー

プロパティー	値	プロパティーの説明
publish_path	*string*	公開されたイメージおよびパラメーター・ファイルに使用するルート名を指定します。
publish_metadata	*flag*	イメージの入力および出力、それらのデータ・モデルを説明するメタデータ・ファイルを作成するかどうかを指定します。
publish_use_parameters	*flag*	ストリーム・パラメーターが *.par ファイルに含まれるかどうかを指定します。
publish_parameters	*string*のリスト	使用するパラメーターを指定します。
execute_mode	export_data publish	ストリームを公開せずにノードを実行するかどうか、ノードの実行時にストリームを自動的に公開するかどうかを指定します。

asexport プロパティー

Analytic Server エクスポートにより、Hadoop 分散ファイル・システム (HDFS) でストリームを実行することができます。

例

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

表 236. *asexport* プロパティー:

asexport プロパティー	データ型	プロパティーの説明
data_source	*string*	データ・ソースの名前。
export_mode	*string*	エクスポートしたデータを既存のデータ・ソースに追加する (**append**) か、既存のデータ・ソースを上書きする (**overwrite**) かを指定します。
use_default_as	*boolean*	True に設定した場合、サーバーの options.cfg ファイルで構成されているデフォルトの Analytic Server 接続が使用されます。False に設定した場合、このノードの接続が使用されます。

表 236. *asexport* プロパティー (続き):

asexport プロパティー	データ型	プロパティーの説明
connection	["文字列","文字列","文字列", "文字列","文字列","文字列","文字列", "文字列","文字列","文字列", "文字列" ,"文字列"]	Analytic Server 接続の詳細を含むリストのプロパティー。形式は次のとおりです: ["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"]。ここで、is_secure_connect はセキュア接続が使用されるかどうかを示し、値は true または false です。use-kerberos-auth は Kerberos 認証が使用されるかどうかを示し、値は true または false です。enable-kerberos-debug は Kerberos 認証のデバッグ・モードが使用されるかどうかを示し、値は true または false です。

cognosexportnode プロパティー

IBM Cognos エクスポート・ノードは、Cognos データベースで読み取ることができる形式でデータをエクスポートできます。

このノードの場合、Cognos 接続と ODBC 接続を定義する必要があります。

Cognos 接続

Cognos 接続のプロパティーは次のとおりです。

表 237. *cognosexportnode* プロパティー

cognosexportnode プロパティー	データ型	プロパティーの説明
cognos_connection	*["文字列","フラグ","文字列","文字列","文字列"]*	Cognos サーバーの接続の詳細を含むリストのプロパティー。形式は以下のとおりです。["Cognos_server_URL", login_mode, "namespace", "username", "password"] ここで、 Cognos_server_URL は、ソースが格納されている Cognos サーバーの URL です。 login_mode は、匿名ログインを使用するかどうかを示し、true または false のいずれかになります。true に設定する場合は、以下の各フィールドを必ず "" に設定してください。 namespace はサーバーへのログオンに使用するセキュリティー認証プロバイダを示します。 username および password は Cognos サーバーにログオンする際に使用するユーザー名とパスワードです。 login_mode の代わりに、以下のモードも使用可能です。 • anonymousMode。例: ['Cognos_server_url', 'anonymousMode', "namespace", "username", "password"] • credentialMode。例: ['Cognos_server_url', 'credentialMode', "namespace", "username", "password"] • storedCredentialMode。例: ['Cognos_server_url', 'storedCredentialMode', "stored_credential_name"] ここで、stored_credential_name は、リポジトリー内での Cognos の資格情報の名前です。
cognos_package_name	*string*	データをエクスポートしている Cognos データ・ソース (通常はデータベース) のパスおよび名前。次に例を示します。/Public Folders/MyPackage
cognos_datasource	*string*	

表 237. *cognosexportnode* プロパティー (続き)

cognosexportnode プロパティー	データ型	プロパティーの説明
cognos_export_mode	Publish ExportFile	
cognos_filename	*string*	

ODBC 接続

ODBC 接続のプロパティーは次のセクションの databaseexportnode に示されているものと同じです。ただし、datasource プロパティーは有効ではありません。

databaseexportnode プロパティー

データベース・エクスポート・ノードで、データを ODBC 対応のリレーショナル・データ・ソースに書き込みます。ODBC データ・ソースに書き込むには、データ・ソースが存在し、そのデータ・ソースに対する書き込み権限を取得している必要があります。

例

```
'''
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"]])
```

表 238. *databaseexportnode* プロパティー：

databaseexportnode プロパティー	データ型	プロパティーの説明
datasource	*string*	
username	*string*	
password	*string*	
epassword	*string*	このスロットは、実行時に読み込み用になります。暗号化パスワードを生成するには、「ツール」メニューの「パスワード暗号化ツール」を使用してください。 詳しくは、トピック 54 ページの『暗号化パスワードの生成』を参照してください。
table_name	*string*	
write_mode	Create Append Merge	
map	*string*	ストリーム・フィールド名をデータベース列名にマッピングします (write_mode が Merge の場合にのみ有効)。 結合の場合、すべてのフィールドをマッピングしてエクスポートする必要があります。データベース内に存在しないフィールド名が、新しい列として追加されます。
key_fields	*list*	キーに使用されるストリーム・フィールドを指定します。map プロパティーは、データベースでストリーム・フィールド内で対応する内容を表示します。
join	Database Add	
drop_existing_table	*flag*	
delete_existing_rows	*flag*	
default_string_size	*integer*	
type		スキーマ タイプの設定に用いられる構造化プロパティー。
generate_import	*flag*	
use_custom_create_table_command	*flag*	*custom_create_table* スロットを使用して、標準の CREATE TABLE SQL コマンドを変更します。
custom_create_table_command	*string*	標準の CREATE TABLE SQL コマンドの代わりに使用する文字列コマンドを指定します。

表 238. *databaseexportnode* プロパティー (続き):

databaseexportnode プロパティー	データ型	プロパティーの説明
use_batch	*flag*	次のプロパティーは、データベースのバルク・ロード用の詳細オプションです。use_batch に真 (True) の値を指定すると、行単位のデータベースへのコミットが無効になります。
batch_size	*number*	メモリーにコミットする前にデータベースに送信するレコード数を指定します。
bulk_loading	Off ODBC External	バルク・ロードの種類を指定します。ODBC および External 用の付加オプションを次に示します。
not_logged	*flag*	
odbc_binding	Row Column	ODBC 経由のバルク・ロードにおける、行方向または列方向のバインドを指定します。
loader_delimit_mode	Tab Space Other	外部プログラム経由のバルク・ロードの場合に、区切り文字の種類を指定します。Other は、loader_other_delimiter プロパティーと組み合わせて選択し、コンマ (,) のような区切り文字を指定します。
loader_other_delimiter	*string*	
specify_data_file	*flag*	真 (True) のフラグを設定すると、以下の data_file プロパティーが有効になります。このプロパティーには、データベースにバルク ロードする際の書き込み先のファイル名とパスを指定することができます。
data_file	*string*	
specify_loader_program	*flag*	真 (True) のフラグを設定すると、以下の loader_program プロパティーが有効になります。このプロパティーには、外部ローダー スクリプトまたはプログラムの名前と場所を指定することができます。
loader_program	*string*	
gen_logfile	*flag*	真 (True) のフラグを設定すると、以下の logfile_name が有効になります。このプロパティーには、エラーログを生成するための、サーバー上のファイル名を指定することができます。
logfile_name	*string*	

表 238. *databaseexportnode* プロパティー (続き):

databaseexportnode プロパティー	データ型	プロパティーの説明
check_table_size	*flag*	真 (True) のフラグを設定すると、IBM SPSS Modeler からエクスポートされる行数に対応してデータベースのテーブル サイズを確実に増加させるために、テーブル検査が実施されます。
loader_options	*string*	ローダー・プログラムに対して、-comment および -specialdir のような、他の引数を指定します。
export_db_primarykey	*flag*	指定されたフィールドがプライマリ キーかどうかを指定します。
use_custom_create_index_command	*flag*	true の場合、すべてのインデックスに対してカスタム SQL (ユーザー指定のSQL) を有効にします。
custom_create_index_command	*string*	カスタム SQL (ユーザー指定のSQL) が有効にされている場合、インデックスの作成に使用される SQL コマンドを指定します。(この値は、下に示す特定のインデックスに対して上書きできます。)
indexes.INDEXNAME.fields		必要な場合は指定されたインデックスを作成し、そのインデックスに含まれるフィールド名を一覧表示します。
INDEXNAME "use_custom_create_index_command"	*flag*	特定のインデックスに対してカスタム SQL (ユーザー指定のSQL) を有効または無効にするのに使用されます。後続の表の後にある例を参照してください。
INDEXNAME "custom_create_index_command"	*string*	指定されたインデックスに使用されるカスタム SQL (ユーザー指定のSQL) を使用します。 後続の表の後にある例を参照してください。
indexes.INDEXNAME.remove	*flag*	True の場合、指定されたインデックスをインデックスのセットから削除します。
table_space	*string*	作成されるテーブル・スペースを指定します。
use_partition	*flag*	分布ハッシュ・フィールドが使用されるよう指定します。
partition_field	*string*	分布ハッシュ・フィールドの内容を消去します。

注: 一部のデータベースでは、エクスポート用に圧縮されたデータベース テーブルを作成することができます (例えば、SQL で CREATE TABLE MYTABLE (...) COMPRESS YES; と指定します)。次のようにプロパティー use_compression および compression_mode を指定して、この機能をサポートします。

表 239. 圧縮機能を使用した *databaseexportnode* プロパティー:

databaseexportnode プロパティー	データ型	プロパティーの説明
use_compression	*Boolean*	True に設定した場合は、圧縮によるエクスポート用のテーブルを作成します。
compression_mode	Row Page	SQL Server データベースの圧縮レベルを設定します。
	Default Direct_Load_Operations All_Operations Basic OLTP Query_High Query_Low Archive_High Archive_Low	Oracle データベースの圧縮レベルを設定します。値 OLTP、Query_High、Query_Low、Archive_High、および Archive_Low には最低限 Oracle 11gR2 が必要です。

CREATE INDEX コマンドを特定のインデックス用に変更する方法を示す例:

```
db_exportnode.setKeyedPropertyValue("indexes", "MYINDEX", ["use_custom_create_index_command",
 True])db_exportnode.setKeyedPropertyValue("indexes", "MYINDEX", ["custom_create_index_command",
 "CREATE BITMAP INDEX <index-name> ON <table-name> <(index-columns)>"])
```

あるいは、同じ処理をハッシュ テーブルを用いて行うこともできます。

```
db_exportnode.setKeyedPropertyValue("indexes", "MYINDEX", ["fields":["id", "region"],
 "use_custom_create_index_command":True, "custom_create_index_command":"CREATE INDEX <index-name> ON
 <table-name> <(index-columns)>"])
```

datacollectionexportnode プロパティー

Data Collection エクスポート・ノードは、Data Collection の市場調査ソフトウェアで使用する形式でデータを出力します。このノードを使用するには、Data Collection Data Library がインストールされている必要があります。

例

```
stream = modeler.script.stream()
datacollectionexportnode = stream.createAt("datacollectionexport", "Data Collection", 200, 200)
datacollectionexportnode.setPropertyValue("metadata_file", "c:¥¥museums.mdd")
datacollectionexportnode.setPropertyValue("merge_metadata", "Overwrite")
datacollectionexportnode.setPropertyValue("casedata_file", "c:¥¥museumdata.sav")
datacollectionexportnode.setPropertyValue("generate_import", True)
datacollectionexportnode.setPropertyValue("enable_system_variables", True)
```

表 240. *datacollectionexportnode* プロパティー

datacollectionexportnode プロパティー	データ型	プロパティーの説明
metadata_file	*string*	出力するメタデータ・ファイルの名前。
merge_metadata	Overwrite MergeCurrent	

表 240. *datacollectionexportnode* プロパティー (続き)

datacollectionexportnode プロパティー	データ型	プロパティーの説明
enable_system_variables	*flag*	エクスポートされた *.mdd* ファイルに Data Collection システム変数を含むかどうかを指定します。
casedata_file	*string*	ケース・データがエクスポートされる *.sav* ファイルの名前。
generate_import	*flag*	

excelexportnode プロパティー

Excel エクスポート ノードでは、データを Microsoft Excel .xlsx ファイル形式で出力します。オプションで、ノードが実行されるときに自動的に Excel が起動し、エクスポートするファイルを開けるように選択できます。

例

```
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)
```

表 241. *excelexportnode* プロパティー

excelexportnode プロパティー	データ型	プロパティーの説明
full_filename	*string*	
excel_file_type	Excel2007	
export_mode	Create Append	
inc_field_names	*flag*	フィールド名がワークシートの最初の行に表示されるかどうかを指定します。
start_cell	*string*	エクスポートの開始セルを指定します。
worksheet_name	*string*	書き込むワークシートの名前。
launch_application	*flag*	Excel が結果のファイルで呼び出されるかどうかを指定します。Excel を起動するパスは、「ヘルパー・アプリケーション」ダイアログ・ボックス (「ツール」メニューから「ヘルパー・アプリケーション」) 内で指定する必要があります。

表 241. *excelexportnode* プロパティー (続き)

excelexportnode プロパティー	データ型	プロパティーの説明
generate_import	*flag*	出力されたデータ・ファイルを読み込む Excel 入力ノードが生成されるかどうかを指定します。

extensionexportnode プロパティー

拡張エクスポート・ノードを使用すると、R スクリプトまたは Python for Spark スクリプトを実行して、データをエクスポートできます。

Python for Spark の例

```
#### 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)
```

R の例

```
#### script example for R
node.setPropertyValue("syntax_type", "R")
node.setPropertyValue("r_syntax", """write.csv(modelerData, "C:/export.csv")""")
```

表 242. *extensionexportnode* プロパティー

extensionexportnode プロパティー	データ型	プロパティーの説明
syntax_type	*R* *Python*	R または Python のどちらのスクリプトを実行するか指定します (R がデフォルトです)。
r_syntax	*string*	実行する R スクリプト・シンタックス。
python_syntax	*string*	実行する Python スクリプト・シンタックス。

表 242. *extensionexportnode* プロパティー (続き)

extensionexportnode プロパティー	データ型	プロパティーの説明
convert_flags	StringsAndDoubles LogicalValues	フラグ型フィールドを変換するためのオプション。
convert_missing	*flag*	欠損値を R の NA 値に変換するためのオプション。
convert_datetime	*flag*	日付形式または日付/時刻形式の変数を R の日付/時刻形式に変換するためのオプション。
convert_datetime_class	POSIXct POSIXlt	日付形式または日付/時刻形式の変数のうち、どの形式の変数を変換するかを指定するためのオプション。

outputfilenode プロパティー

ファイル・ノードでは、データが区切り文字で区切られたテキスト・ファイルへ出力されます。このことは、他の分析ソフトウェアや表計算ソフトウェアに読み込める形式でデータをエクスポートする場合に、役立ちます。

例

```
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)
```

表 243. *outputfilenode* プロパティー

outputfilenode プロパティー	データ型	プロパティーの説明
full_filename	*string*	出力ファイルの名前。
write_mode	Overwrite Append	
inc_field_names	*flag*	
use_newline_after_records	*flag*	
delimit_mode	Comma Tab Space Other	
other_delimiter	*char* 型	

表 243. *outputfilenode* プロパティー (続き)

outputfilenode プロパティー	データ型	プロパティーの説明
quote_mode	None Single Double Other	
other_quote	*flag*	
generate_import	*flag*	
encoding	StreamDefault SystemDefault "UTF-8"	

sasexportnode プロパティー

SAS エクスポート・ノードで、SAS または SAS 互換ソフトウェア・パッケージで読み込むデータを、SAS 形式で出力できます。3 つの SAS ファイル形式が利用可能です。SAS for Windows/OS2、SAS for UNIX、または SAS バージョン 7/8

例

```
stream = modeler.script.stream()
sasexportnode = stream.createAt("sasexport", "SAS Export", 200, 200)
sasexportnode.setPropertyValue("full_filename", "c:/output/SAS_output.sas7bdat")
sasexportnode.setPropertyValue("format", "SAS8")
sasexportnode.setPropertyValue("export_names", "NamesAndLabels")
sasexportnode.setPropertyValue("generate_import", True)
```

表 244. *sasexportnode* プロパティー

sasexportnode プロパティー	データ型	プロパティーの説明
形式	Windows UNIX SAS7 SAS8	バリアント・プロパティー・ラベル・フィールド。
full_filename	*string*	
export_names	NamesAndLabels NamesAsLabels	エクスポート時にフィールド名を IBM SPSS Modeler から IBM SPSS Statistics または SAS変数名に関連付けます。
generate_import	*flag*	

statisticsexportnode プロパティー

Statistics エクスポート・ノードでは、IBM SPSS Statistics *.sav* または *.zsav* フォーマットでデータを出力します。*.sav* または *.zsav* ファイルは、IBM SPSS Statistics Base およびその他の製品で読み込むことができます。この形式は、IBM SPSS Modeler のキャッシュ・ファイルでも使用されます。

このノードのプロパティーについては、 357 ページの『statisticsexportnode プロパティー』に記載されています。

tm1odataexport ノードのプロパティー

IBM Cognos TM1 エクスポート・ノードは、Cognos TM1 データベースで読み取ることができる形式でデータをエクスポートできます。

表 *245. tm1odataexport* ノードのプロパティー

tm1odataexportノードのプロパティー	データ型	プロパティーの説明
admin_host	*string*	REST API のホスト名の URL。
server_name	*string*	admin_host から選択した TM1 サーバーの名前。
credential_type	inputCredential または storedCredential	資格情報のタイプを示すために使用されます。
input_credential	*list*	credential_type が inputCredential のときは、ドメイン、ユーザー名、およびパスワードを指定します。
stored_credential_name	*string*	credential_type が storedCredential のときは、C&DS サーバーの資格情報の名前を指定します。
selected_cube	*field*	データのエクスポート先のキューブの名前。以下に例を示します。 TM1_export.setPropertyValue ("selected_cube", "plan_BudgetPlan")

表 245. *tm1odataexport* ノードのプロパティー (続き)

tm1odataexportノードのプロパティー	データ型	プロパティーの説明
spss_field_to_tm1_element_mapping	*list*	マップされる tm1 要素は、選択されたキューブ ビューの列ディメンションの一部でなければなりません。形式は次のとおりです: [[[Field_1, Dimension_1, False], [Element_1, Dimension_2, True], ...], [[Field_2, ExistMeasureElement, False], [Field_3, NewMeasureElement, True], ...]] マッピング情報を示す 2 つのリストがあります。ディメンションへの葉要素のマッピングは、以下の例 2 に対応しています。 例 1: 最初のリスト: ([[Field_1, Dimension_1, False], [Element_1, Dimension_2, True], ...]) は、TM1 ディメンジョンのマップ情報に使用されます。 3 つの値を持つそれぞれのリストは、ディメンション マッピング情報を示します。3 番目のブール値は、ディメンションの要素を選択するかどうかを示すために使用されます。例: "[Field_1, Dimension_1, False]" は、Field_1 が Dimension_1 にマップされることを示します。"[Element_1, Dimension_2, True]" は、Element_1 が Dimension_2 に対して選択されることを示します。 例 2: 2 番目のリスト: ([[Field_2, ExistMeasureElement, False], [Field_3, NewMeasureElement, True], ...]) は、TM1 数値データ ディメンション要素のマップ情報に使用されます。 3 つの値を持つそれぞれのリストは、測定要素のマッピング情報を示します。3 番目のブール値は、新しい要素を作成する必要があることを示すために使用されます。"[Field_2, ExistMeasureElement, False]" は、Field_2 が ExistMeasureElement にマップされることを示します。"[Field_3, NewMeasureElement, True]" は、NewMeasureElement が selected_measure で選択された数値データ ディメンションである必要があり、Field_3 がそれにマップされることを示します。
selected_measure	*string*	数値データ ディメンションを指定します。 例: setPropertyValue("selected_measure", "Measures")

tm1export ノードのプロパティー (廃止)

IBM Cognos TM1 エクスポート・ノードは、Cognos TM1 データベースで読み取ることができる形式でデータをエクスポートできます。

注: このノードは、Modeler 18.0 で廃止されました。それに置き換わるノードのスクリプト名は *tm1odataexport* です。

表 *246. tm1export* ノードのプロパティー:

tm1export ノードのプロパティー	データ型	プロパティーの説明
pm_host	*string*	**注:** バージョン 16.0 および 17.0 の場合のみ ホスト名。以下に例を示します。 `TM1_export.setPropertyValue("pm_host", 'http://9.191.86.82:9510/pmhub/pm')`
tm1_connection	*["field","field", ... ,"field"]*	**注:** バージョン 16.0 および 17.0 の場合のみ TM1 サーバーの接続の詳細を含むリストのプロパティー。形式は次のとおりです: `[ "TM1_Server_Name", "tm1_ username", "tm1_password"]` 以下に例を示します。 `TM1_export.setPropertyValue("tm1_connection", ['Planning Sample', "admin" "apple"])`
selected_cube	*field*	データのエクスポート先のキューブの名前。以下に例を示します。 `TM1_export.setPropertyValue ("selected_cube", "plan_BudgetPlan")`

表 246. *tm1export* ノードのプロパティー (続き):

tm1export ノードのプロパティー	データ型	プロパティーの説明
spssfield_tm1element_mapping	*list*	マップされる tm1 要素は、選択されたキューブ ビューの列ディメンションの一部でなければなりません。形式は次のとおりです: [[[Field_1, Dimension_1, False], [Element_1, Dimension_2, True], ...], [[Field_2, ExistMeasureElement, False], [Field_3, NewMeasureElement, True], ...]] マッピング情報を示す 2 つのリストがあります。ディメンションへの葉要素のマッピングは、以下の例 2 に対応しています。 例 1: 最初のリスト: ([[Field_1, Dimension_1, False], [Element_1, Dimension_2, True], ...]) は、TM1 ディメンジョンのマップ情報に使用されます。 3 つの値を持つそれぞれのリストは、ディメンション マッピング情報を示します。3 番目のブール値は、ディメンションの要素を選択するかどうかを示すために使用されます。例: "[Field_1, Dimension_1, False]" は、Field_1 が Dimension_1 にマップされることを示します。"[Element_1, Dimension_2, True]" は、Element_1 が Dimension_2 に対して選択されることを示します。 例 2: 2 番目のリスト: ([[Field_2, ExistMeasureElement, False], [Field_3, NewMeasureElement, True], ...]) は、TM1 数値データ ディメンション要素のマップ情報に使用されます。 3 つの値を持つそれぞれのリストは、測定要素のマッピング情報を示します。3 番目のブール値は、新しい要素を作成する必要があることを示すために使用されます。"[Field_2, ExistMeasureElement, False]" は、Field_2 が ExistMeasureElement にマップされることを示します。"[Field_3, NewMeasureElement, True]" は、NewMeasureElement が selected_measure で選択された数値データ ディメンションである必要があり、Field_3 がそれにマップされることを示します。
selected_measure	*string*	数値データ ディメンションを指定します。 例: setPropertyValue("selected_measure", "Measures")

xmlexportnode プロパティー

XML エクスポート・ノードでは、XML 形式のファイルにデータを出力します。オプションで、エクスポートしたデータをストリームに読み込む XML 入力ノードを作成できます。

例

```
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"]])
```

表 247. *xmlexportnode* プロパティー

xmlexportnode プロパティー	データ型	プロパティーの説明
full_filename	*string*	(必須) XML エクスポート・ファイルの完全パスおよびファイル名。
use_xml_schema	*flag*	XML スキーマ (XSD ファイルまたは DTD ファイル) を使用して、エクスポートされたデータの構造を制御するかどうかを指定します。
full_schema_filename	*string*	使用する XSD ファイルまたは DTD ファイルの完全パスおよびファイル名。use_xml_schema が true に設定されている場合にのみ必須です。
generate_import	*flag*	エクスポートされたデータ・ファイルをストリームに読み込む XML 入力ノードを、自動的に生成します。
records	*string*	レコードの境界を示す XPath 式。
map	*string*	XML 構造にフィールド名をマッピングします。

第 18 章 IBM SPSS Statistics ノードのプロパティー

statisticsimportnode プロパティー

Statistics ファイル・ノードは、同じ形式を使用する IBM SPSS Statistics で使用される *.sav* または *.zsav* ファイル形式のデータおよび IBM SPSS Modeler に保存されたキャッシュ・ファイルを読み込みます。

例

```
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)
```

表 *248. statisticsimportnode* プロパティー：

statisticsimportnode プロパティー	データ型	プロパティーの説明
full_filename	*string*	パスを含む、完全なファイル名。
password	*string*	パスワード。password パラメータは、file_encrypted パラメータよりも前に設定する必要があります。
file_encrypted	*flag*	ファイルがパスワード保護されているかどうか。
import_names	NamesAndLabels LabelsAsNames	変数名と変数ラベルを処理する方法。
import_data	DataAndLabels LabelsAsData	値とラベルを処理する方法。
use_field_format_for_storage	*Boolean*	インポート時に IBM SPSS Statistics フィールド形式情報を使用するかどうかを指定します。

statisticstransformnode プロパティー

Statistics 変換ノードは、IBM SPSS Modeler のデータ・ソースに対する IBM SPSS Statistics シンタックス・コマンドの選択を行います。このノードは、ライセンスが与えられた IBM SPSS Statistics のコピーが必要です。

例

```
stream = modeler.script.stream()
statisticstransformnode = stream.createAt("statisticstransform", "Transform", 200, 200)
statisticstransformnode.setPropertyValue("syntax", "COMPUTE NewVar = Na + K.")
statisticstransformnode.setKeyedPropertyValue("new_name", "NewVar", "Mixed Drugs")
statisticstransformnode.setPropertyValue("check_before_saving", True)
```

表 249. *statisticstransformnode* プロパティー

statisticstransformnode プロパティー	データ型	プロパティーの説明
syntax	*string*	
check_before_saving	*flag*	項目を保存する前に、入力されたシンタックスを検証します。シンタックスが無効な場合は、エラー・メッセージを表示します。
default_include	*flag*	詳しくは、トピック 146 ページの『filternode プロパティー』を参照してください。
include	*flag*	詳しくは、トピック 146 ページの『filternode プロパティー』を参照してください。
new_name	*string*	詳しくは、トピック 146 ページの『filternode プロパティー』を参照してください。

statisticsmodelnode プロパティー

Statistics モデル・ノードを使用すると、PMML を作成する IBM SPSS Statistics 手続きを実行してデータを分析および使用することができます。このノードは、ライセンスが与えられた IBM SPSS Statistics のコピーが必要です。

例

```
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")
```

statisticsmodelnode プロパティー	データ型	プロパティーの説明
syntax	*string*	
default_include	*flag*	詳しくは、トピック 146 ページの『filternode プロパティー』を参照してください。
include	*flag*	詳しくは、トピック 146 ページの『filternode プロパティー』を参照してください。
new_name	*string*	詳しくは、トピック 146 ページの『filternode プロパティー』を参照してください。

statisticsoutputnode プロパティー

Statistics 出力ノードを使用すると、IBM SPSS Statistics 手続きを呼び出し、IBM SPSS Modeler データを分析することができます。さまざまな IBM SPSS Statistics 分析手続きにアクセスできます。このノードは、ライセンスが与えられた IBM SPSS Statistics のコピーが必要です。

例

```
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")
```

表 250. *statisticsoutputnode* プロパティー

statisticsoutputnode プロパティー	データ型	プロパティーの説明
mode	Dialog 構文	「IBM SPSS Statistics ダイアログ」オプションまたはシンタックス・エディターを選択します。
syntax	*string*	
use_output_name	*flag*	
output_name	*string*	
output_mode	Screen File	
full_filename	*string*	
file_type	HTML SPV SPW	

statisticsexportnode プロパティー

Statistics エクスポート・ノードでは、IBM SPSS Statistics *.sav* または *.zsav* フォーマットでデータを出力します。*.sav* または *.zsav* ファイルは、IBM SPSS Statistics Base およびその他の製品で読み込むことができます。この形式は、IBM SPSS Modeler のキャッシュ・ファイルでも使用されます。

例

```
stream = modeler.script.stream()
statisticsexportnode = stream.createAt("statisticsexport", "Export", 200, 200)
statisticsexportnode.setPropertyValue("full_filename", "c:/output/SPSS_Statistics_out.sav")
statisticsexportnode.setPropertyValue("field_names", "Names")
statisticsexportnode.setPropertyValue("launch_application", True)
statisticsexportnode.setPropertyValue("generate_import", True)
```

表 *251. statisticsexportnode* プロパティー：

statisticsexportnode プロパティー	データ型	プロパティーの説明
full_filename	*string*	
file_type	sav zsav	ファイルを *sav* または *zsav* 形式で保存します。以下に例を示します。 statisticsexportnode.setPropertyValue("file_type","sav")
encrypt_file	*flag*	ファイルがパスワード保護されているかどうか。
password	*string*	パスワード。
launch_application	*flag*	
export_names	NamesAndLabels NamesAsLabels	エクスポート時にフィールド名を IBM SPSS Modeler から IBM SPSS Statistics または SAS変数名に関連付けます。
generate_import	*flag*	

第 19 章 Python ノードのプロパティー

smotenode のプロパティー

SMOTE (Synthetic Minority Over-sampling Technique) ノードは不均衡データ・セットを扱うためのオーバーサンプリング・アルゴリズムを提供します。これにより、データの均衡化のための高度な手法が提供されます。SPSS Modeler の SMOTE プロセス ノードは Python で実装されており、imbalanced-learn© Python ライブラリーを必要とします。

表 252. *smotenode* のプロパティー

smotenode のプロパティー	データ型	プロパティーの説明
target_field	*field*	対象フィールド。
sample_ratio	*string*	カスタムの比率の値を使用できるようにします。オプションは、自動 (sample_ratio_auto) と比率の設定 (sample_ratio_manual) の 2 つです。
sample_ratio_value	浮動小数点	これは、マジョリティー クラスのサンプル数に対するマイノリティー クラスのサンプル数の比率です。 0 より大きく 1 以下でなければなりません。デフォルトは auto です。
random_seed	*integer*	乱数発生ルーチンによって使用されるシード。
k_neighbours	*integer*	合成サンプルを作成するために使用する最近傍の数。デフォルトは 5 です。
m_neighbours	*integer*	マイノリティー サンプルが危険な状況にあるかをどうか判断するために使用する最近傍の数。このオプションは、SMOTE アルゴリズムの種類が borderline1 および borderline2 の場合にのみ使用可能です。デフォルトは 10 個です。
algorithm_kind	*string*	SMOTE アルゴリズムの種類: regular、borderline1、または borderline2。
use_partition	*Boolean*	true に設定した場合は、モデルの構築に学習データのみが使用されます。デフォルトは true です。

xgboosttreenode のプロパティー

XGBoost Tree© は、ツリー モデルを基本モデルとして使用する勾配ブースティング・アルゴリズムの高度な実装です。ブースティング・アルゴリズムでは、弱い分類子に繰り返し学習させ、それを最終的な強い分類子に追加します。XGBoost Tree は柔軟性が極めて高く、多くのユーザーを圧倒するほどの多数のパラメータが用意されています。このため、SPSS Modeler の XGBoost Tree ノードでは、コア・フィーチャーおよびよく使用されるパラメータが公開されています。このノードは Python で実装されています。

表 253. *xgbooststreenode* のプロパティー

xgbooststreenode のプロパティー	データ型	プロパティーの説明
TargetField	*field*	対象フィールド。
InputFields	*field*	入力フィールド。
treeMethod	*string*	モデルの構築用のツリー手法。指定できる値は、auto、exact、または approx です。デフォルトは auto です。
numBoostRound	*integer*	モデル作成用の num boost round 値。1 から 1000 の間の値を指定します。デフォルトは 10 個です。
maxDepth	*integer*	ツリーの成長の最大深度。1 以上の値を指定します。デフォルトは 6 です。
minChildWeight	*Double*	ツリーの成長のための子の重みの最小値。0 以上の値を指定します。デフォルトは 1 です。
maxDeltaStep	*Double*	ツリーの成長の差分ステップの最大数。0 以上の値を指定します。デフォルトは 0 です。
objectiveType	*string*	学習タスクの目的タイプ。指定できる値は reg:linear、reg:logistic、reg:gamma、reg:tweedie、count:poisson、■ rank:pairwise、binary:logistic、または multi です。フラグ型対象の場合、binary:logistic または multi のみを使用できます。multi を使用する場合、スコア結果には XGBoost 目的タイプ multi:softmax および multi:softprob が表示されます。
random_seed	*integer*	乱数シード。0 から 9999999 の範囲の任意の数値です。デフォルトは 0 です。
sampleSize	*Double*	過剰適合を制御するためのサブサンプル。0.1 から 1.0 の間の値を指定します。デフォルトは 0.1 です。
eta	*Double*	過剰適合を制御するためのイータ。0 から 1 の間の値を指定します。デフォルトは 0.3 です。
gamma	*Double*	過剰適合を制御するためのガンマ。0 以上の任意の数値を指定してください。デフォルトは 6 です。
colsSampleRatio	*Double*	過剰適合を制御するためのツリー別の列サンプル。0.01 から 1 の間の値を指定します。デフォルトは 1 です。
colsSampleLevel	*Double*	過剰適合を制御するためのレベル別の列サンプル。0.01 から 1 の間の値を指定します。デフォルトは 1 です。
lambda	*Double*	過剰適合を制御するためのラムダ。0 以上の任意の数値を指定してください。デフォルトは 1 です。

表 253. *xgbooosttreenode* のプロパティー (続き)

xgbooosttreenode のプロパティー	データ型	プロパティーの説明
alpha	*Double*	過剰適合を制御するためのアルファ。0 以上の任意の数値を指定してください。デフォルトは 0 です。
scalePosWeight	*Double*	不均衡データ・セットを扱うためのスケールの正の重み。デフォルトは 1 です。

xboostlinearnode のプロパティー

XGBoost Linear© は、線型モデルを基本モデルとして使用する勾配ブースティング・アルゴリズムの高度な実装です。ブースティング・アルゴリズムでは、弱い分類子に繰り返し学習させ、それを最終的な強い分類子に追加します。SPSS Modeler の XGBoost Linear ノードは Python で実装されています。

表 254. *xgboostlinearnode* のプロパティー

xgboostlinearnode のプロパティー	データ型	プロパティーの説明
TargetField	*field*	
InputFields	*field*	
alpha	*Double*	アルファ線型ブースティング パラメータ。0 以上の任意の数値を指定してください。デフォルトは 0 です。
lambda	*Double*	ラムダ線型ブースティング パラメータ。0 以上の任意の数値を指定してください。デフォルトは 1 です。
lambdaBias	*Double*	ラムダ バイアス線型ブースティング パラメータ。任意の数値を指定します。デフォルトは 0 です。
numBoostRound	*integer*	モデル作成用の num boost round 値。1 から 1000 の間の値を指定します。デフォルトは 10 個です。
objectiveType	*string*	学習タスクの目的タイプ。指定できる値は reg:linear、reg:logistic、reg:gamma、reg:tweedie、count:poisson、■ rank:pairwise、binary:logistic、または multi です。フラグ型対象の場合、binary:logistic または multi のみを使用できます。multi を使用する場合、スコア結果には XGBoost 目的タイプ multi:softmax および multi:softprob が表示されます。
random_seed	*integer*	乱数シード。0 から 9999999 の範囲の任意の数値です。デフォルトは 0 です。

ocsvmnode のプロパティー

One-Class SVM ノードでは、非監視学習アルゴリズムを使用します。このノードは、新規性検知の目的で使用できます。このノードは、与えられたサンプル・セットのソフト境界を検知し、新規ポイントがこのセットに属するか、属さないかを分類します。SPSS Modeler の One-Class SVM モデリング ノードは Python で実装されており、scikit-learn© Python ライブラリーを必要とします。

表 255. *ocsvmnode* のプロパティー

ocsvmnode のプロパティー	データ型	プロパティーの説明
role_use	*string*	定義済みの役割を使用するには predefined を指定し、ユーザー設定フィールドの割り当てを使用するには custom を指定します。デフォルトは predefined です。
inputs	*field*	入力用のフィールド名のリスト。
splits	*field*	分割用のフィールド名のリスト。
use_partition	*Boolean*	true または false を指定します。デフォルトは true です。true に設定した場合は、モデルの構築時に学習データのみが使用されます。
mode_type	*string*	モード。指定できる値は、simple または expert です。simple を指定した場合は、「エキスパート」タブのすべてのパラメータが無効になります。
stopping_criteria	*string*	指数表記の文字列。指定できる値は、1.0E-1、1.0E-2、1.0E-3、1.0E-4、1.0E-5、または 1.0E-6 です。デフォルトは 1.0E-3 です。
precision	*float*	回帰精度 (ニュー)。学習誤差およびサポート・ベクターの小数部の範囲です。 0 より大きく 1.0 以下の数値を指定してください。デフォルトは 0.1 です。
kernel	*string*	アルゴリズムで使用するカーネル タイプ。指定できる値は linear、poly、rbf、sigmoid、または precomputed です。デフォルトは rbf です。
enable_gamma	*Boolean*	gamma パラメータを有効にします。true または false を指定します。デフォルトは true です。
gamma	*float*	このパラメータはカーネル rbf、poly、および sigmoid の場合にのみ有効です。enable_gamma パラメータを false に設定した場合、このパラメータは auto に設定されます。true に設定した場合、デフォルトは 0.1 です。
coef0	*float*	カーネル関数の独立した項目。このパラメータは、poly カーネルおよび sigmoid カーネルの場合にのみ有効です。デフォルト値は 0.0 です。

表 255. *ocsvmnode* のプロパティー (続き)

ocsvmnode のプロパティー	データ型	プロパティーの説明
degree	*integer*	多項式のカーネル関数の次数。このパラメータは、poly カーネルの場合にのみ有効です。任意の整数を指定します。デフォルトは 3 です。
shrinking	*Boolean*	収縮ヒューリスティック・オプションを使用するかどうかを指定します。true または false を指定します。デフォルトは false です。
enable_cache_size	*Boolean*	cache_size パラメータを有効にします。true または false を指定します。デフォルトは false です。
cache_size	*float*	カーネル キャッシュのサイズ (MB)。デフォルトは 200 です。
enable_random_seed	*Boolean*	random_seed パラメータを有効にします。true または false を指定します。デフォルトは false です。
random_seed	*integer*	確率推定のためにデータをシャッフルする際に使用する乱数シード。任意の整数を指定します。
pc_type	*string*	平行座標グラフィックスのタイプ。指定できるオプションは independent または general です。
lines_amount	*integer*	グラフィックに含める最大行数。1 から 1000 の整数を指定します。
lines_fields_custom	*Boolean*	lines_fields パラメータを有効にし、グラフ出力にカスタム・フィールドを表示できるようにします。false に設定した場合、すべてのフィールドが表示されます。true に設定した場合、lines_fields パラメータで指定したフィールドのみが表示されます。パフォーマンス上の理由から、最大で 20 個のフィールドが表示されます。
lines_fields	*field*	グラフィックに垂直軸として含めるフィールド名のリスト。

第 20 章 スーパーノードのプロパティー

スーパーノード固有のプロパティーを次の表に示します。共通のノード・プロパティーもスーパーノードに適用されることに注意してください。

表 256. ターミナル・スーパーノードのプロパティー

プロパティー名	プロパティーの種類/値のリスト	プロパティーの説明
execute_method	Script 正規	
script	*string*	

スーパーノードのパラメーター

次の一般形式を使用して、スーパーノードのパラメーターを作成または設定するためにスクリプトを使用できます。

```
mySuperNode.setParameterValue("minvalue", 30)
```

以下を使用して、パラメーター値を取得することができます。

```
value mySuperNode.getParameterValue("minvalue")
```

既存のスーパーノードの検索

findByType() 関数を使用して、ストリーム内のスーパーノードを検索できます。

```
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", None)
```

カプセル化ノードのプロパティー設定

スーパーノード内の子ダイアグラムにアクセスすることにより、スーパーノードの中にカプセル化された特定のノードのプロパティーを設定できます。例えば、データを読み込むためにカプセル化された可変長ファイルのある入力スーパーノードがあるとします。以下のようにして、子ダイアグラムにアクセスし、関連ノードを検索することにより、読み込みファイルの名前 (full_filename プロパティーを使用して指定) を渡すことができます。

```
childDiagram = source_supernode.getChildDiagram()
varfilenode = childDiagram.findByType("variablefile", None)
varfilenode.setPropertyValue("full_filename", "c:/mydata.txt")
```

スーパーノードの作成

スーパーノードとその中身を初めから作成する場合、同様の方法で行うことができます。このためには、スーパーノードを作成し、子ダイアグラムにアクセスして、目的のノードを作成します。スーパーノードのダイアグラム内のすべてのノードを、入力コネクター・ノードや出力コネクター・ノードとリンクさせるようにすることも必要です。例えば、プロセス スーパーノードを作成する場合は、次のようにします。

```
process_supernode = modeler.script.stream().createAt("process_super", "My SuperNode", 200, 200)
childDiagram = process_supernode.getChildDiagram()
filternode = childDiagram.createAt("filter", "My Filter", 100, 100)
childDiagram.linkFromInputConnector(filternode)
childDiagram.linkToOutputConnector(filternode)
```

付録 A. ノード名のリファレンス

ここでは、IBM SPSS Modeler のノードのスクリプト名のリファレンスを提供します。

モデル・ナゲット名

モデル・ナゲット (生成されたモデル) は、ノード・オブジェクトと出力オブジェクトと同様に、その種類で参照できます。 次の表に、モデル・オブジェクトの参照名を一覧表示します。

これらの名前は、IBM SPSS Modeler ウィンドウの右上隅にある「モデル」パレット内のモデル・ナゲットを参照するために、特に使用されます。 スコアリングの目的でストリームに追加されたモデル・ノードを参照するには、`apply...` の接頭辞が付いた別の名前セットが使用されます。 詳しくは、トピックモデル・ナゲット・ノードのプロパティーを参照してください。

注: 通常の状況では、名前および 種類の両方でモデルを参照することが、混乱を避けるために推奨されます。

表 257. モデル・ナゲット名 (「モデル作成」パレット) :

モデル名	モデル
anomalydetection	異常値
Apriori	Apriori
autoclassifier	自動分類
autocluster	自動クラスター
autonumeric	自動数値
bayesnet	ベイズ・ネットワーク
c50	C5.0
carma	Carma
cart	C&R Tree
chaid	CHAID
coxreg	Cox 回帰
decisionlist	ディシジョン・リスト
discriminant	判別
factor	因子分析
featureselection	変数選択
genlin	一般化線型回帰
glmm	GLMM
kmeans	K-Means
knn	*k* 最近傍法
kohonen	Kohonen
線型	線型
logreg	ロジスティック回帰
neuralnetwork	ニューラル・ネットワーク

表 257. モデル・ナゲット名 (「モデル作成」パレット) (続き) :

モデル名	モデル
quest	QUEST
回帰	線型回帰
sequence	シーケンス
slrm	自己学習応答モデル
statisticsmodel	IBM SPSS Statistics モデル
svm	Support Vector Machine
timeseries	時系列
TwoStep	TwoStep

表 258. モデル・ナゲット名 (「データベース・モデリング」パレット) :

モデル名	モデル
db2imcluster	IBM ISW クラスタリング
db2imlog	IBM ISW ロジスティック回帰
db2imnb	IBM ISW Naive Bayes
db2imreg	IBM ISW 回帰
db2imtree	IBM ISW デシジョン・ツリー
msassoc	MS アソシエーション・ルール
msbayes	MS Naive Bayes
mscluster	MS クラスタリング
mslogistic	MS ロジスティック回帰
msneuralnetwork	MS ニューラル・ネットワーク
msregression	MS 線型回帰
mssequencecluster	MS シーケンス・クラスタリング
mstimeseries	MS タイム・シリーズ
mstree	MS デシジョン・ツリー
netezzabayes	Netezza ベイズ・ネットワーク
netezzadectree	Netezza ディシジョン・ツリー
netezzadivcluster	Netezza 分裂クラスタリング
netezzaglm	Netezza 一般化線型
netezzakmeans	Netezza K-Means
netezzaknn	Netezza KNN
netezzalineregression	Netezza 線型回帰
netezzanaivebayes	Netezza Naive Bayes
netezzapca	Netezza PCA
netezzaregtree	Netezza 回帰ツリー
netezzatimeseries	Netezza 時系列
oraabn	Oracle Adaptive Bayes
oraai	Oracle AI
oradecisiontree	Oracle デシジョン・ツリー
oraglm	Oracle GLM

表 258. モデル・ナゲット名 (「データベース・モデリング」パレット) (続き) :

モデル名	モデル
orakmeans	Oracle *k*-Means
oranb	Oracle Naive Bayes
oranmf	Oracle NMF
oraocluster	Oracle O-Cluster
orasvm	Oracle SVM

重複するモデル名の回避

生成されたモデルを操作するのにスクリプトを使用する場合、重複するモデル名を使用していると、スクリプトがあいまいになることに注意する必要があります。 これを避けるために、スクリプト作成時に、生成されたモデルには一意の名前を使用することをお勧めします。

重複するモデル名に関するオプションを設定するには

1. メニューから次の項目を選択します。

   「ツール」 > 「ユーザー オプション」
2. 「通知」タブをクリックします。
3. 生成されたモデルに対して重複する名前を禁止するには、「前のモデルを置換」を選択します。

あいまいなモデルの参照がある場合、スクリプト実行の動作は SPSS Modeler と IBM SPSS Collaboration and Deployment Services との間で異なります。SPSS Modeler クライアントには自動的に同じ名前を持つモデルを置き換えるオプション「以前のモデルを置き換える」があります (例えば、スクリプトをループで反復して随時異なる名前を作成)。しかし、このオプションは、同じスクリプトが IBM SPSS Collaboration and Deployment Services で実行される場合は使用できません。ループの終了前に、モデルに対するあいまいな参照を回避するために各反復で生成されるモデルの名前を変更するか、現在のモデルをクリアすることにより (`clear generated palette` 文の追加など)、この状況を回避することができます。

出力形式名

次の表に、すべての出力オブジェクトの形式と、それを作成するノードを一覧表示します。 各タイプの出力オブジェクトで使用できるエクスポート形式の完全なリストについては、出力タイプを作成するノードのプロパティーの説明 (グラフ作成ノードの共通のプロパティーと出力ノードのプロパティー) を参照してください。

表 259. 出力オブジェクトの種類と、そのオブジェクトを作成するノード :

出力オブジェクトの種類	ノード
analysisoutput	分析
collectionoutput	集計棒グラフ
dataauditoutput	データ検査
distributionoutput	分布
evaluationoutput	評価
histogramoutput	ヒストグラム
matrixoutput	クロス集計

表 259. 出力オブジェクトの種類と、そのオブジェクトを作成するノード (続き) :

出力オブジェクトの種類	ノード
meansoutput	平均値
multiplotoutput	マルチ散布図
plotoutput	作図
qualityoutput	品質
reportdocumentoutput	このオブジェクトの種類はノードからのものではなく、プロジェクト・レポートに作成された出力です。
reportoutput	レポート
statisticsprocedureoutput	StatisticsOutput
statisticsoutput	記述統計
tableoutput	表
timeplotoutput	時系列グラフ
weboutput	Web

付録 B. 従来のスクリプトから Python スクリプトへの移行

従来のスクリプトの移行の概要

ここでは、IBM SPSS Modeler での Python スクリプトと従来のスクリプトの違いを要約し、従来のスクリプトを Python スクリプトに移行する方法について説明します。また、SPSS Modeler の標準的な従来のコマンドと、同等の Python コマンドのリストも示します。

一般的な差異

従来のスクリプトの設計の大部分は、OS コマンド・スクリプトが基になっています。従来のスクリプトは、行指向であり、一部のブロック構造 (`if...then...else...endif` や、`for...endfor` など) があるとしても、インデントには一般に意味がありません。

Python スクリプトでは、インデントには意味があり、同一の論理ブロックに属する複数の行は、同じレベルにインデントされている必要があります。

注: Python コードをコピーして貼り付ける場合は、注意が必要です。タブを使用してインデントされている行は、エディター上で、スペースを使用してインデントされている行と同じように見える場合があります。しかし、これらの行が同じインデントであるとは見なされないため、Python スクリプトはエラーを生成します。

スクリプト・コンテキスト

スクリプト・コンテキストは、スクリプトを実行する環境 (例えば、スクリプトを実行するストリームやスーパーノード) を定義します。従来のスクリプトでは、コンテキストは暗黙的です。つまり、例えば、ストリーム・スクリプト内のノード参照は、そのスクリプトを実行するストリーム内にあると想定されます。

Python スクリプトでは、スクリプト・コンテキストは、`modeler.script` モジュールによって明示的に提供されます。例えば、Python ストリーム・スクリプトは、以下のコードを使用して、スクリプトを実行するストリームにアクセスできます。

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

ストリームに関連した関数は、返されたオブジェクトによって呼び出すことができます。

コマンドと関数

従来のスクリプトは、コマンド指向です。つまり、スクリプトの各行は、実行する必要があるコマンドが先頭にあり、パラメーターが後に続きます。例えば、以下のとおりです。

```
connect 'Type':typenode to :filternode
rename :derivenode as "Compute Total"
```

Python は、通常、関数を定義するオブジェクト (モジュール、クラス、またはオブジェクト) によって起動される関数を使用します。例えば、以下のとおりです。

```
stream = modeler.script.stream()
typenode = stream.findByType("type", "Type)
filternode = stream.findByType("filter", None)
stream.link(typenode, filternode)
derive.setLabel("Compute Total")
```

リテラルとコメント

IBM SPSS Modeler でよく使用される一部のリテラル・コマンドおよびコメント・コマンドには、Python スクリプトの同等コマンドがあります。これは、SPSS Modeler の既存の従来のスクリプトを、IBM SPSS Modeler 17 で使用できるように、Python スクリプトに変換するのに役立ちます。

表 *260.* リテラルとコメントの従来のスクリプトから *Python* スクリプトへのマッピング：

従来のスクリプト	**Python** スクリプト
整数。例: `4`	同じ
浮動小数点数。例: `0.003`	同じ
単一引用符で囲まれた文字列。例: `'Hello'`	同じ **注:** 非 ASCII 文字が含まれている文字列リテラルには、接頭辞 `u` を付けて、Unicode として表します。
二重引用符で囲まれた文字列。例: `"Hello again"`	同じ **注:** 非 ASCII 文字が含まれている文字列リテラルには、接頭辞 `u` を付けて、Unicode として表します。

長い文字列。例:
`"""This is a string`
`that spans multiple`
`lines"""`
	同じ

リスト。例: `[1 2 3]`	`[1, 2, 3]`
変数の参照。例: `set x = 3`	`x = 3`

行の継続 (¥)。例:
`set x = [1 2 ¥`
`  3 4]`
	`x = [ 1, 2,¥`
	`  3, 4]`

ブロックのコメント。例:
`/* This is a long comment`
`over a line. */`
	`""" This is a long comment`
	`over a line. """`

行のコメント。例: `set x = 3  # make x 3`	`x = 3  # make x 3`
`undef`	`None`
`true`	`True`
`false`	`False`

演算子

IBM SPSS Modeler でよく使用される一部の演算子コマンドには、Python スクリプトの同等コマンドがあります。これは、SPSS Modeler の既存の従来のスクリプトを、IBM SPSS Modeler 17 で使用できるように、Python スクリプトに変換するのに役立ちます。

表 *261.* 演算子の従来のスクリプトから *Python* スクリプトへのマッピング：

従来のスクリプト	**Python** スクリプト
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

条件とループ

IBM SPSS Modeler でよく使用される一部の条件コマンドおよびループ・コマンドには、Python スクリプトの同等コマンドがあります。これは、SPSS Modeler の既存の従来のスクリプトを、IBM SPSS Modeler 17 で使用できるように、Python スクリプトに変換するのに役立ちます。

表 *262.* 条件とループの従来のスクリプトから *Python* スクリプトへのマッピング：

従来のスクリプト	**Python** スクリプト

for VAR from INT1 to INT2
  ...
endfor
	for VAR in range(INT1, INT2):
	  ...
	
	or
	
	VAR = INT1
	while VAR <= INT2:
	  ...
	  VAR += 1

for VAR in LIST
  ...
endfor
	for VAR in LIST:
	  ...

表 262. 条件とループの従来のスクリプトから *Python* スクリプトへのマッピング (続き):

従来のスクリプト	**Python** スクリプト

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	同等機能なし
var VAR1	変数宣言は不要

変数

従来のスクリプトでは、変数は参照される前に宣言します。例えば、以下のとおりです。

```
var mynode
set mynode = create typenode at 96 96
```

Python スクリプトでは、変数は初回の参照時に作成されます。例えば、以下のとおりです。

```
mynode = stream.createAt("type", "Type", 96, 96)
```

従来のスクリプトでは、変数の参照は ^ 演算子を使用して明示的に削除する必要があります。例えば、以下のとおりです。

```
var mynode
set mynode = create typenode at 96 96
set ^mynode.direction."Age" = Input
```

ほとんどのスクリプト言語と同様、Python スクリプトでは、これは不要です。例えば、以下のとおりです。

```
mynode = stream.createAt("type", "Type", 96, 96)
mynode.setKeyedPropertyValue("direction","Age","Input")
```

ノード、出力、およびモデルの各タイプ

従来のスクリプトのさまざまなオブジェクト・タイプ (ノード、出力、およびモデル) では、通常、タイプがオブジェクトのタイプに追加された形になっています。例えば、フィールド作成 (Derive) ノードのタイプは、derivenode です。

```
set feature_name_node = create derivenode at 96 96
```

Python の IBM SPSS Modeler API には、node 接尾辞が含まれないため、フィールド作成ノード (Derive) のタイプは、derive です。例えば、以下のとおりです。

```
feature_name_node = stream.createAt("derive", "Feature", 96, 96)
```

従来のスクリプトと Python スクリプトでのタイプ名の唯一の違いは、タイプ接尾辞がないことです。

プロパティー名

プロパティー名は、従来のスクリプトと Python スクリプトで同じです。例えば、可変長ファイル・ノードでは、ファイルの場所を定義するプロパティーは、両方のスクリプト環境で full_filename です。

ノードの参照

多くの従来のスクリプトは、暗黙の検索を使用して、変更するノードを見つけてアクセスします。例えば、以下のコマンドは、ラベル「Type」を使用して、現行ストリームの中でデータ型ノードを検索し、「Age」フィールドの方向 (またはモデル作成の役割) を Input に、「Drug」フィールドを Target (予測される値) に設定します。

```
set 'Type':typenode.direction."Age" = Input
set 'Type':typenode.direction."Drug" = Target
```

Python スクリプトでは、プロパティー値を設定するための関数を呼び出す前に、ノード・オブジェクトを明示的に位置指定する必要があります。例えば、以下のとおりです。

```
typenode = stream.findByType("type", "Type")
typenode.setKeyedPropertyValue("direction", "Age", "Input")
typenode.setKeyedPropertyValue("direction", "Drug", "Target")
```

注: この場合、"Target" を文字列引用符で囲む必要があります。

Python スクリプトは、ModelingRole 列挙を modeler.api パッケージで使用することもできます。

Python スクリプトのバージョンは、より冗長な場合がありますが、ノードの検索は通常 1 回のみ行われるため、ランタイム・パフォーマンスが良くなります。従来のスクリプトの例では、ノードの検索は、コマンドごとに行われます。

ID によるノードの検索もサポートされています (ノード ID は、ノード・ダイアログの「注釈」タブで確認できます)。例えば、従来のスクリプトでは、以下のようにします。

```
# id65EMPB9VL87 is the ID of a Type node
set @id65EMPB9VL87.direction."Age" = Input
```

以下のスクリプトは、Python スクリプトを使用した場合の同じ例です。

```
typenode = stream.findByID("id65EMPB9VL87")
typenode.setKeyedPropertyValue("direction", "Age", "Input")
```

プロパティーの取得と設定

従来のスクリプトは、set コマンドを使用して、値を割り当てます。 set コマンドの後ろに、プロパティー定義を続けることができます。以下のスクリプトは、プロパティーを設定するための 2 つの有効な形式を示しています。

```
set <node reference>.<property> = <value>
set <node reference>.<keyed-property>.<key> = <value>
```

Python スクリプトでは、関数 setPropertyValue() と setKeyedPropertyValue() を使用して、同じ結果が得られます。例えば、以下のとおりです。

```
object.setPropertyValue(property, value)
object.setKeyedPropertyValue(keyed-property, key, value)
```

従来のスクリプトでは、get コマンドを使用して、プロパティー値にアクセスできます。例えば、以下のとおりです。

```
var n v
set n = get node :filternode
set v = ^n.name
```

Python スクリプトでは、関数 getPropertyValue() を使用して、同じ結果が得られます。例えば、以下のとおりです。

```
n = stream.findByType("filter", None)
v = n.getPropertyValue("name")
```

ストリームの編集

従来のスクリプトでは、create コマンドを使用して、新しいノードを作成します。例えば、以下のとおりです。

```
var agg select
set agg = create aggregatenode at 96 96
set select = create selectnode at 164 96
```

Python スクリプトでは、ノードを作成するためのさまざまなメソッドがストリームに用意されています。例えば、以下のとおりです。

```
stream = modeler.script.stream()
agg = stream.createAt("aggregate", "Aggregate", 96, 96)
select = stream.createAt("select", "Select", 164, 96)
```

従来のスクリプトでは、connect コマンドを使用して、ノード間のリンクを作成します。例えば、以下のとおりです。

```
connect ^agg to ^select
```

Python スクリプトでは、link メソッドを使用して、ノード間のリンクを作成します。例えば、以下のとおりです。

```
stream.link(agg, select)
```

従来のスクリプトでは、disconnect コマンドを使用して、ノード間のリンクを削除します。例えば、以下のとおりです。

```
disconnect ^agg from ^select
```

Python スクリプトでは、unlink メソッドを使用して、ノード間のリンクを削除します。例えば、以下のとおりです。

```
stream.unlink(agg, select)
```

従来のスクリプトでは、position コマンドを使用して、ストリーム・キャンバスにノードを配置したり、他のノード間にノードを配置したりします。例えば、以下のとおりです。

```
position ^agg at 256 256
position ^agg between ^myselect and ^mydistinct
```

Python スクリプトでは、2 つの異なるメソッド setXYPosition と setPositionBetween を使用して、同じ結果が得られます。以下に例を示します。

```
agg.setXYPosition(256, 256)
agg.setPositionBetween(myselect, mydistinct)
```

ノード操作

IBM SPSS Modeler でよく使用される一部のノード操作コマンドには、Python スクリプトの同等コマンドがあります。これは、SPSS Modeler の既存の従来のスクリプトを、IBM SPSS Modeler 17 で使用できるように、Python スクリプトに変換するのに役立ちます。

表 *263.* ノード操作の従来のスクリプトから *Python* スクリプトへのマッピング：

従来のスクリプト	**Python** スクリプト
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*)

ループ

従来のスクリプトでは、サポートされている主なループ・オプションが 2 つあります。

- カウント型 ループ。インデックス変数が、2 つの整数の境界の間で変化します。
- シーケンス型 ループ。一連の値をループして、現在の値をループ変数にバインドします。

以下のスクリプトは、従来のスクリプトでのカウント型ループの例です。

```
for i from 1 to 10
 println ^i
endfor
```

以下のスクリプトは、従来のスクリプトでのシーケンス型ループの例です。

```
var items
set items = [a b c d]

for i in items
 println ^i
endfor
```

以下のような他のタイプのループも使用可能です。

- モデル・パレットのモデル、または出力パレットの出力を反復する。
- ノードに入るフィールドまたはノードから出るフィールドを反復する。

Python スクリプトでも、さまざまなタイプのループをサポートしています。以下のスクリプトは、Python スクリプトでのカウント型ループの例です。

```
i = 1
while i <= 10:
 print i
 i += 1
```

以下のスクリプトは、Python スクリプトでのシーケンス型ループの例です。

```
items = ["a", "b", "c", "d"]
for i in items:
 print i
```

シーケンス型ループは非常に柔軟であり、IBM SPSS Modeler API メソッドと組み合わせることにより、従来のスクリプトの大部分のユース・ケースをサポートできます。以下の例は、Python スクリプトでシーケンス型ループを使用して、ノードから出るフィールドを反復する方法を示しています。

```
node = modeler.script.stream().findByType("filter", None)
for column in node.getOutputDataModel().columnIterator():
 print column.getColumnName()
```

ストリームの実行

ストリームの実行中に、生成されたモデルまたは出力オブジェクトが、いずれかのオブジェクト・マネージャーに追加されます。従来のスクリプトでは、スクリプトは、作成されたオブジェクトをオブジェクト・マネージャーから位置指定するか、生成された最新の出力に、その出力を生成したノードからアクセスする必要があります。

Python でのストリームの実行は、実行により生成されたモデルまたは出力オブジェクトが、実行関数に渡されるリストに返されるという点で異なります。このため、ストリームの実行結果に、より簡単にアクセスできます。

従来のスクリプトは、以下の 3 つのストリーム実行コマンドをサポートしています。

- `execute_all` は、ストリーム内のすべての実行可能ターミナル・ノードを実行します。
- `execute_script` は、スクリプト実行の設定に関係なく、ストリーム・スクリプトを実行します。
- `execute` *`node`* は、指定したノードを実行します。

Python スクリプトは、以下のような同様の関数をサポートしています。

- *`stream`*`.runAll(`*`results-list`*`)` は、ストリーム内のすべての実行可能ターミナル・ノードを実行します。
- *`stream`*`.runScript(`*`results-list`*`)` は、スクリプト実行の設定に関係なく、ストリーム・スクリプトを実行します。
- *`stream`*`.runSelected(`*`node-array`*`, `*`results-list`*`)` は、指定したノードのセットを、指定した順に実行します。
- *`node`*`.run(`*`results-list`*`)` は、指定したノードを実行します。

従来のスクリプトでは、オプションの整数コードを指定した `exit` コマンドを使用して、ストリームの実行を終了できます。例えば、以下のとおりです。

```
exit 1
```

Python スクリプトでは、以下のスクリプトを使用して、同じ結果が得られます。

```
modeler.script.exit(1)
```

ファイル・システムおよびリポジトリーによるオブジェクトへのアクセス

従来のスクリプトでは、open コマンドを使用して、既存のストリーム、モデル、または出力オブジェクトを開くことができます。例えば、以下のとおりです。

```
var s
set s = open stream "c:/my streams/modeling.str"
```

Python スクリプトには、セッションからアクセス可能で、同じような作業を実行できる TaskRunner クラスがあります。例えば、以下のとおりです。

```
taskrunner = modeler.script.session().getTaskRunner()
s = taskrunner.openStreamFromFile("c:/my streams/modeling.str", True)
```

従来のスクリプトを使用してオブジェクトを保存するには、save コマンドを使用します。例えば、以下のとおりです。

```
save stream s as "c:/my streams/new_modeling.str"
```

同等の Python スクリプトのアプローチでは、TaskRunner クラスを使用します。例えば、以下のとおりです。

```
taskrunner.saveStreamToFile(s, "c:/my streams/new_modeling.str")
```

IBM SPSS Collaboration and Deployment Services Repository ベースの操作は、retrieve および store コマンドを使用することよって、従来のスクリプトでサポートされています。例えば、以下のとおりです。

```
var s
set s = retrieve stream "/my repository folder/my_stream.str"
store stream ^s as "/my repository folder/my_stream_copy.str"
```

Python スクリプトでは、セッションに関連付けられているリポジトリー・オブジェクトによって、同等の機能にアクセスできます。

```
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)
```

注: リポジトリーにアクセスするには、有効なリポジトリー接続を使用してセッションが構成されている必要があります。

ストリーム操作

IBM SPSS Modeler でよく使用される一部のストリーム操作コマンドには、Python スクリプトの同等コマンドがあります。これは、SPSS Modeler の既存の従来のスクリプトを、IBM SPSS Modeler 17 で使用できるように、Python スクリプトに変換するのに役立ちます。

表 *264.* ストリーム操作の従来のスクリプトから *Python* スクリプトへのマッピング:

従来のスクリプト	**Python** スクリプト
create stream *DEFAULT_FILENAME*	*taskrunner*.createStream(*name*, *autoConnect*, *autoManage*)
close stream	*stream*.close()
clear stream	*stream*.clear()
get stream *stream*	同等機能なし
load stream *path*	同等機能なし
open stream *path*	*taskrunner*.openStreamFromFile(*path*, *autoManage*)

表 *264.* ストリーム操作の従来のスクリプトから *Python* スクリプトへのマッピング *(*続き*)*:

従来のスクリプト	**Python** スクリプト
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*)

モデルの操作

IBM SPSS Modeler でよく使用される一部のモデル操作コマンドには、Python スクリプトの同等コマンドがあります。これは、SPSS Modeler の既存の従来のスクリプトを、IBM SPSS Modeler 17 で使用できるように、Python スクリプトに変換するのに役立ちます。

表 *265.* モデル操作の従来のスクリプトから *Python* スクリプトへのマッピング:

従来のスクリプト	**Python** スクリプト
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*)

ドキュメント出力操作

IBM SPSS Modeler でよく使用される一部のドキュメント出力操作コマンドには、Python スクリプトの同等コマンドがあります。これは、SPSS Modeler の既存の従来のスクリプトを、IBM SPSS Modeler 17 で使用できるように、Python スクリプトに変換するのに役立ちます。

表 *266.* ドキュメント出力操作の従来のスクリプトから *Python* スクリプトへのマッピング:

従来のスクリプト	**Python** スクリプト
open output *path*	*taskrunner*.openDocumentFromFile(*path*, *autoManage*)
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*)

従来のスクリプトと **Python** スクリプトのその他の違い

レガシー・スクリプトは、IBM SPSS Modeler プロジェクトの操作をサポートしています。 Python スクリプトは、現在、これをサポートしていません。

従来のスクリプトは、ステート型 オブジェクト (ストリームおよびモデルの組み合わせ) をいくらかサポートしています。ステート型オブジェクトは、IBM SPSS Modeler 8.0 以降、廃止されました。 Python スクリプトは、ステート型オブジェクトをサポートしていません。

Python スクリプトは、従来のスクリプトでは使用できない、以下の追加の機能を提供しています。

- クラス定義と関数定義

- エラー処理
- より高度な入出力サポート
- 外部のサード・パーティー・モジュール

特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものです。この資料は、IBM から他の言語でも提供されている可能性があります。ただし、これを入手するには、本製品または当該言語版製品を所有している必要がある場合があります。

本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒*103-8510*
東京都中央区日本橋箱崎町*19*番*21*号
日本アイ・ビー・エム株式会社
法務・知的財産
知的財産権ライセンス渉外

IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、IBM より提供されます。

記載されている性能データとお客様事例は、例として示す目的でのみ提供されています。実際の結果は特定の構成や稼働条件によって異なります。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確証できません。 IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

IBM の将来の方向性および指針に関する記述は、予告なく変更または撤回される場合があります。これらは目標および目的を提示するものにすぎません。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名称はすべて架空のものであり、類似する個人や企業が実在しているとしても、それは偶然にすぎません。

商標

IBM、IBM ロゴおよび ibm.com は、世界の多くの国で登録された International Business Machines Corporation の商標です。他の製品名およびサービス名等は、それぞれ IBM または各社の商標である場合があります。現時点での IBM の商標リストについては、http://www.ibm.com/legal/copytrade.shtml をご覧ください。

Adobe、Adobe ロゴ、PostScript、PostScript ロゴは、Adobe Systems Incorporated の米国およびその他の国における登録商標または商標です。

インテル、Intel、Intel ロゴ、Intel Inside、Intel Inside ロゴ、Centrino、Intel Centrino ロゴ、Celeron、Xeon、Intel SpeedStep、Itanium、および Pentium は、Intel Corporation または子会社の米国およびその他の国における商標または登録商標です。

Linux は、Linus Torvalds の米国およびその他の国における登録商標です。

Microsoft、Windows、Windows NT および Windows ロゴは、Microsoft Corporation の米国およびその他の国における商標です。

UNIX は The Open Group の米国およびその他の国における登録商標です。

Java およびすべての Java 関連の商標およびロゴは Oracle やその関連会社の米国およびその他の国における商標または登録商標です。

製品資料に関するご使用条件

これらの資料は、以下のご使用条件に同意していただける場合に限りご使用いただけます。

適用条件

IBM Web サイトの「ご利用条件」に加えて、以下のご使用条件が適用されます。

個人的使用

これらの資料は、すべての著作権表示その他の所有権表示をしていただくことを条件に、非商業的な個人による使用目的に限り複製することができます。ただし、IBM の明示的な承諾をえずに、これらの資料またはその一部について、二次的著作物を作成したり、配布 (頒布、送信を含む) または表示 (上映を含む) することはできません。

商業的使用

これらの資料は、すべての著作権表示その他の所有権表示をしていただくことを条件に、お客様の企業内に限り、複製、配布、および表示することができます。ただし、IBM の明示的な承諾をえずにこれらの資料の二次的著作物を作成したり、お客様の企業外で資料またはその一部を複製、配布、または表示することはできません。

権利

ここで明示的に許可されているもの以外に、資料や資料内に含まれる情報、データ、ソフトウェア、またはその他の知的所有権に対するいかなる許可、ライセンス、または権利を明示的にも黙示的にも付与するものではありません。

資料の使用が IBM の利益を損なうと判断された場合や、上記の条件が適切に守られていないと判断された場合、IBM はいつでも自らの判断により、ここで与えた許可を撤回できるものとさせていただきます。

お客様がこの情報をダウンロード、輸出、または再輸出する際には、米国のすべての輸出入 関連法規を含む、すべての関連法規を遵守するものとします。

IBM は、これらの資料の内容についていかなる保証もしません。これらの資料は、特定物として現存するままの状態で提供され、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任なしで提供されます。

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

［ア行］

［カ行］

［サ行］

[タ行]

[ナ行]

[ハ行]

[マ行]

[ヤ行]

[ラ行]

A

B

C

N

O

P

Q

R

S

T

U

V

W

X

Printed in Japan

日本アイ・ビー・エム株式会社
〒103-8510 東京都中央区日本橋箱崎町19-21