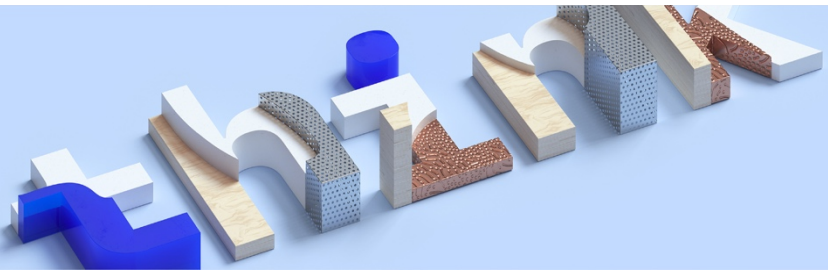


think 2018

IBM



Lab Center – Hands-on Lab

4188

Automation in IBM SPSS Modeler Using Python

Jim Mott, Ph.D., IBM, jcmott@us.ibm.com

Jos den Ronden, IBM, jdenronden@nl.ibm.com

Table of Contents

| | |
|---|----|
| Disclaimer | 5 |
| Workshop Preparation..... | 7 |
| Workshop 1: Use Modeler's Native Tools to Automate Tasks..... | 8 |
| Task 1. Import and examine the data..... | 8 |
| Task 2. Create a loop through row fields in the Matrix node..... | 9 |
| Workshop 2: Use Python to Automate Tasks | 18 |
| Task 1. Examine and run a pasted Python script. | 18 |
| Task 2. Examine and run an existing Python script..... | 21 |
| Task 3. Use Modeler's data model to identify categorical variables. | 24 |
| Task 4. Use Modeler's data model to identify categorical variables and their role..... | 27 |
| Workshop 3: Explore Scripts | 30 |
| Task 1. Change field names downstream from an Aggregate node. | 30 |
| Task 2. Control the order of execution. | 33 |
| Task 3. Create standardized fields. | 36 |
| Task 4. Assess the importance of a predictor by leaving it out. | 38 |
| Task 5. List all nodes in a stream..... | 40 |
| Task 6. Recode a string field into a numeric field, with value labels. | 41 |
| We Value Your Feedback!..... | 46 |

Disclaimer

IBM's statements regarding its plans, directions, and intent are subject to change or withdrawal without notice at IBM's sole discretion. Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision.

The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract.

The development, release, and timing of any future features or functionality described for our products remains at our sole discretion I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results like those stated here.

Information in these presentations (including information relating to products that have not yet been announced by IBM) has been reviewed for accuracy as of the date of initial publication and could include unintentional technical or typographical errors. IBM shall have no responsibility to update this information. **This document is distributed "as is" without any warranty, either express or implied. In no event, shall IBM be liable for any damage arising from the use of this information, including but not limited to, loss of data, business interruption, loss of profit or loss of opportunity.** IBM products and services are warranted per the terms and conditions of the agreements under which they are provided.

IBM products are manufactured from new parts or new and used parts.

In some cases, a product may not be new and may have been previously installed. Regardless, our warranty terms apply."

Any statements regarding IBM's future direction, intent or product plans are subject to change or withdrawal without notice.

Performance data contained herein was generally obtained in controlled, isolated environments. Customer examples are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual performance, cost, savings or other results in other operating environments may vary.

References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business.

Workshops, sessions and associated materials may have been prepared by independent session speakers, and do not necessarily reflect the views of IBM. All materials and discussions are provided for informational purposes only, and are neither intended to, nor shall constitute legal or other guidance or advice to any individual participant or their specific situation.

It is the customer's responsibility to insure its own compliance with legal requirements and to obtain advice of competent legal counsel as to the identification and interpretation of any relevant laws and regulatory requirements that may affect the customer's business and any actions the customer may need to take to comply with such laws. IBM does not provide legal advice or represent or warrant that its services or products will ensure that the customer follows any law.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products about this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products. IBM does not warrant the quality of any third-party products, or the ability of any such third-party products to interoperate with IBM's products. **IBM expressly disclaims all warranties, expressed or implied, including but not limited to, the implied warranties of merchantability and fitness for a purpose.**

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents, copyrights, trademarks or other intellectual property right.

IBM, the IBM logo, and ibm.com are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at: www.ibm.com/legal/copytrade.shtml.

© 2018 International Business Machines Corporation. No part of this document may be reproduced or transmitted in any form without written permission from IBM.

U.S. Government Users Restricted Rights — use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM.

Workshop Preparation

IBM SPSS Modeler will be used in the workshops. This section will start the software and set its working folder.

1. Double-click the **IBM SPSS Modeler 18.1.1** shortcut on the desktop to start IBM SPSS Modeler.

It will take about two minutes for IBM SPSS Modeler to start; a Welcome window will display, asking you what you want to do. You will not make use of this option.

2. Click **Cancel** to close the **Welcome** window.

You will set the working folder for IBM SPSS Modeler.

3. From the **File** menu, click **Set Directory**, navigate to **C:\Training\4188**, and then click **Set**.

You are ready to start the workshops.

Workshop 1: Use Modeler's Native Tools to Automate Tasks

You work for a telecommunications firm and want to use the Matrix node to assess the importance of a number of fields to predict CHURN. You will use the Looping feature to cycle through these predictors

Stream file: workshop_1_start.str

Data file: telco churn data.txt

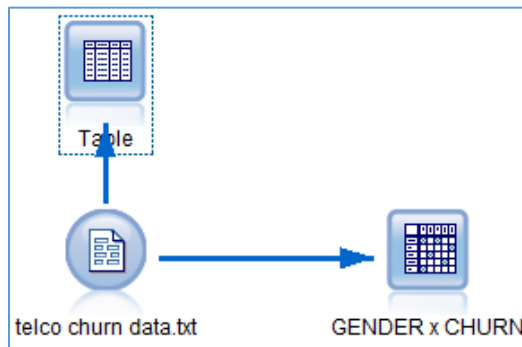
Folder C:\Training\4188

Task 1. Import and examine the data.

You will import and examine the data.

1. From the **File** menu, click **Open Stream**, click **workshop_1_start.str**, located in **C:\Training\4188** and then click **Open**.

The results appear as follows:



The stream imports data, and includes a Table and Matrix node to examine the data.

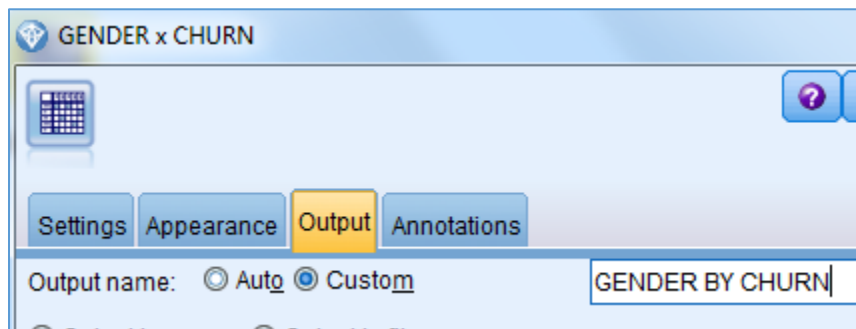
2. Run the **Table** node.
The dataset contains customers of a telecommunications firm.
3. Click **OK** to close the **Table** output window.

Task 2. Create a loop through row fields in the Matrix node.

For the purpose of demonstration later, you will assign a custom name to the Matrix output table.

1. Edit the **Matrix** node.
2. Click the **Output** tab.
3. Beside **Output name**, select the **Custom** option.
4. In the **Custom** text box, type **GENDER BY CHURN**.

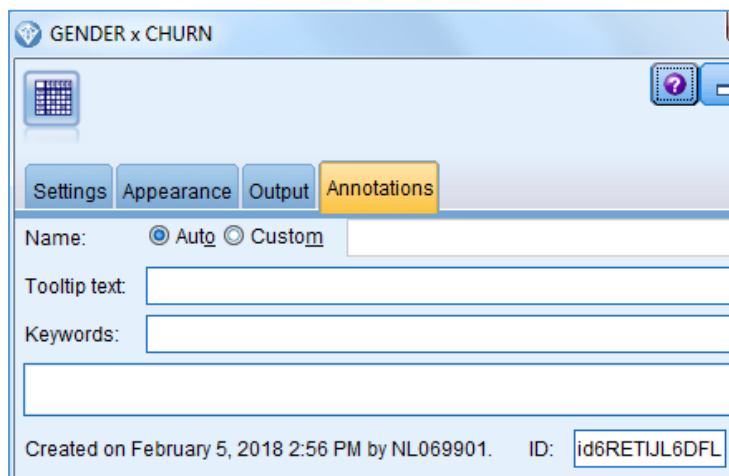
The results appear as follows:



You will not annotate the node, but will examine the settings on the Annotations tab.

5. Click the **Annotations** tab.

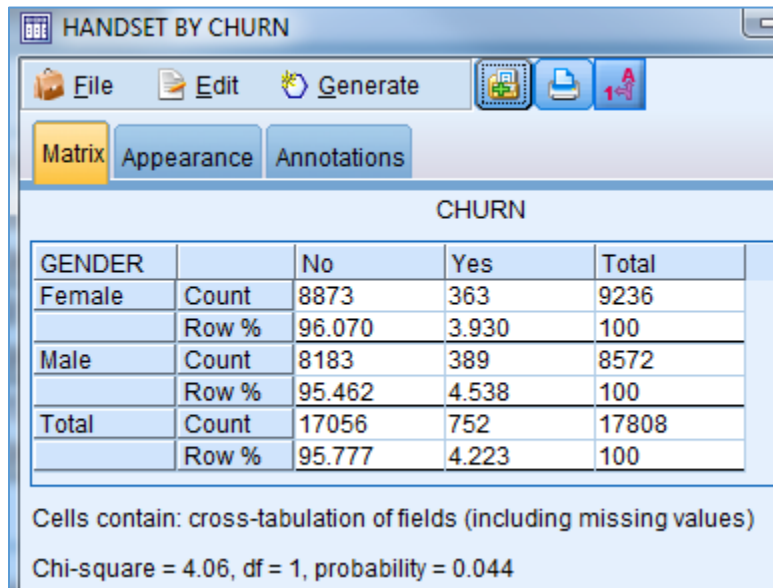
The results appear similar to the following:



Each IBM SPSS Modeler node has an Annotations tab, where you can enter some text.

Notice the ID in the right lower corner. Each IBM SPSS Modeler node has an ID that uniquely identifies the node.

6. Click the **Run**  button.
The results appear as follows:



| GENDER | | No | Yes | Total |
|--------|-------|--------|-------|-------|
| Female | Count | 8873 | 363 | 9236 |
| | Row % | 96.070 | 3.930 | 100 |
| Male | Count | 8183 | 389 | 8572 |
| | Row % | 95.462 | 4.538 | 100 |
| Total | Count | 17056 | 752 | 17808 |
| | Row % | 95.777 | 4.223 | 100 |

Cells contain: cross-tabulation of fields (including missing values)
Chi-square = 4.06, df = 1, probability = 0.044

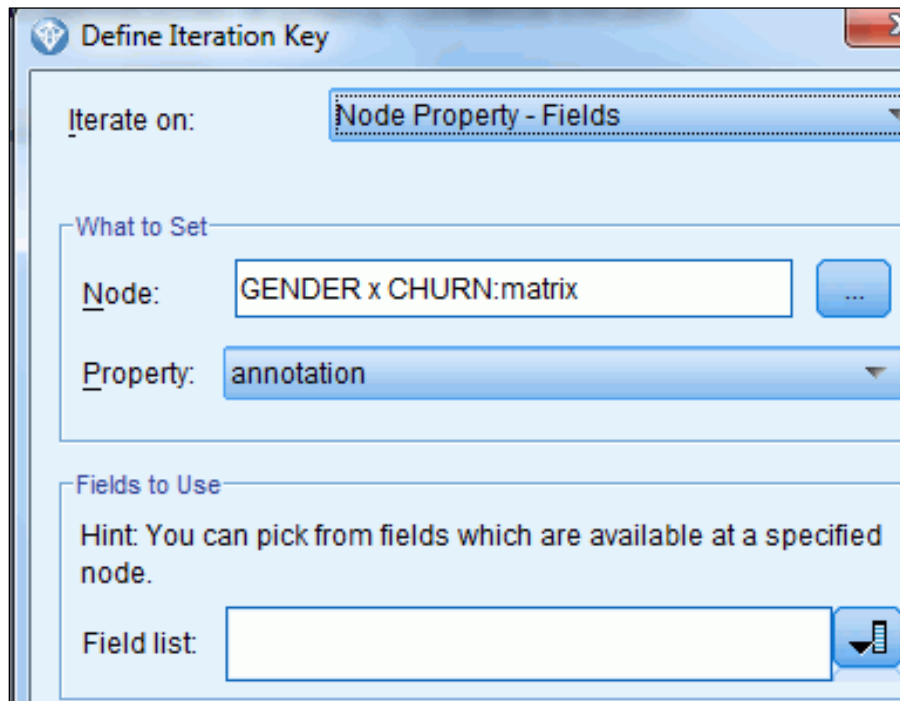
There is small, but significant, difference in churn rates between men and women. Men are somewhat more inclined to cancel their account.

Notice that the title bar of the Matrix output reads GENDER BY CHURN, as specified on the Output name tab in the Matrix dialog box.

7. Click **OK** to close the **Matrix** output window.
You want to cross tabulate other categorical fields by CHURN. The most efficient way is to iterate through the row fields in the Matrix node.

8. Right-click the **Matrix** node, click **Looping/Conditional Execution**, and then click **Define Iteration Key (Fields)**.

The results appear as follows:

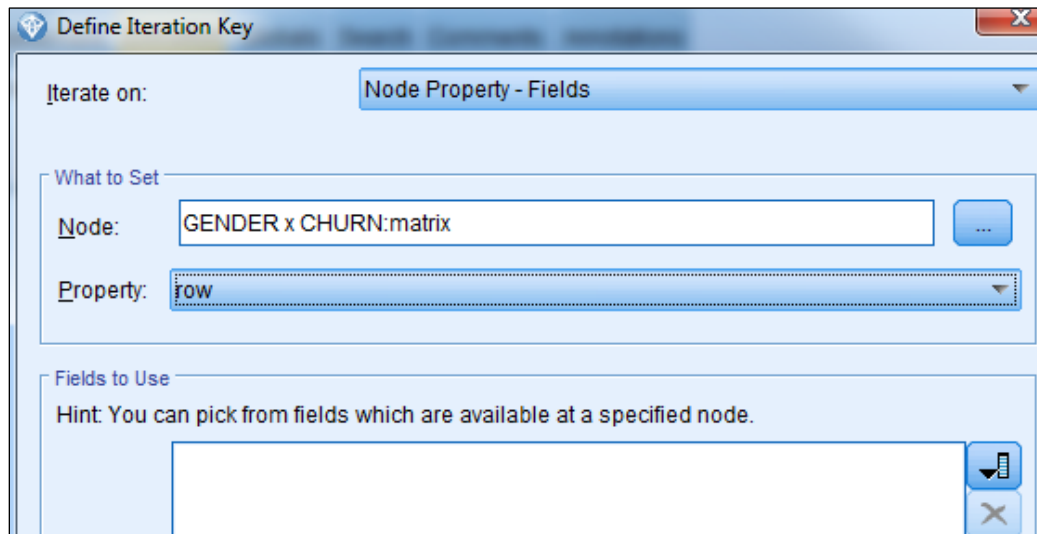


You want to cycle through fields in the Matrix node, so the Iterate on and Node settings are correct.

The Property setting at this point specifies that you want to change the Matrix node's annotation in each run. Recall, each IBM SPSS Modeler node has an Annotations tab, where you can enter some text. You do not want to change the annotation of the Matrix node, but you want to cycle through the row fields. This property is named row, so you will select that property.

- Click the pull-down menu for **Property**, scroll down in the list of properties, and then select **row**.

The results appear as follows:

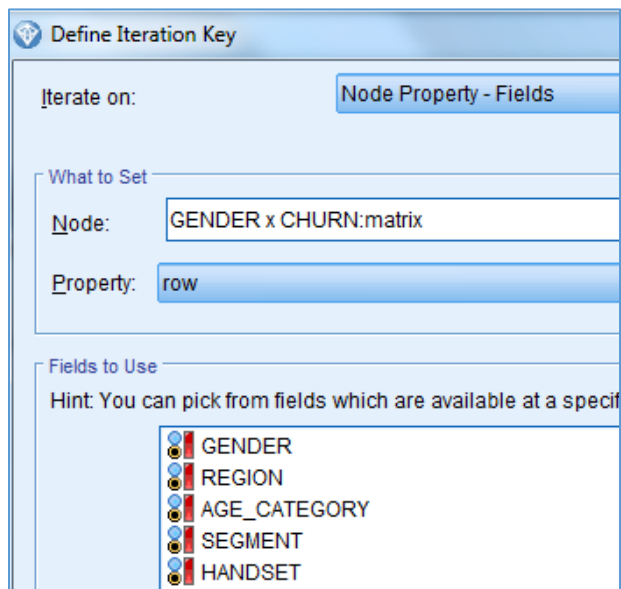


The image shows a 'Define Iteration Key' dialog box. The 'Iterate on:' dropdown is set to 'Node Property - Fields'. Under the 'What to Set' section, the 'Node:' field contains 'GENDER x CHURN:matrix' and the 'Property:' dropdown is set to 'row'. The 'Fields to Use' section has a hint: 'Hint: You can pick from fields which are available at a specified node.' and an empty list box with a 'Field selector' button (a button with a list icon) and a close button (an 'X' icon).

In the Fields to Use area you will select the (categorical) fields that you want to cross tabulate by CHURN.

- In the **Fields to Use** area, click the **Field selector**  button, select **GENDER, REGION, AGE_CATEGORY, SEGMENT, and HANDSET**, and then click **OK**.

The results appear as follows:

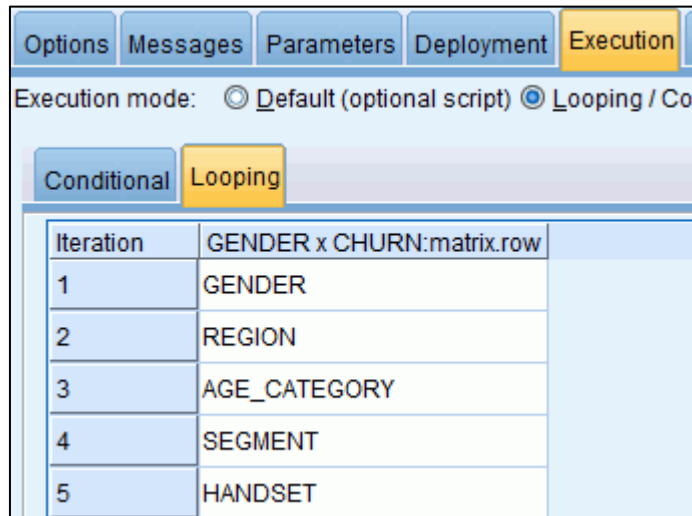


The image shows the 'Define Iteration Key' dialog box with the 'Fields to Use' section expanded. The list box now contains five selected fields: GENDER, REGION, AGE_CATEGORY, SEGMENT, and HANDSET. Each field has a small icon to its left, indicating it is selected. The 'Node:' field still contains 'GENDER x CHURN:matrix' and the 'Property:' dropdown is still set to 'row'.

The setup to loop through the row fields in the Matrix node is now complete.

11. Click **OK**.

The results appear as follows:



| Iteration | GENDER x CHURN:matrix.row |
|-----------|---------------------------|
| 1 | GENDER |
| 2 | REGION |
| 3 | AGE_CATEGORY |
| 4 | SEGMENT |
| 5 | HANDSET |

The Iteration column shows that there will be five iterations. The iteration key is in the column named GENDER x CHURN:matrix.row, which tells you that the iterations affect the row field in the Matrix node.

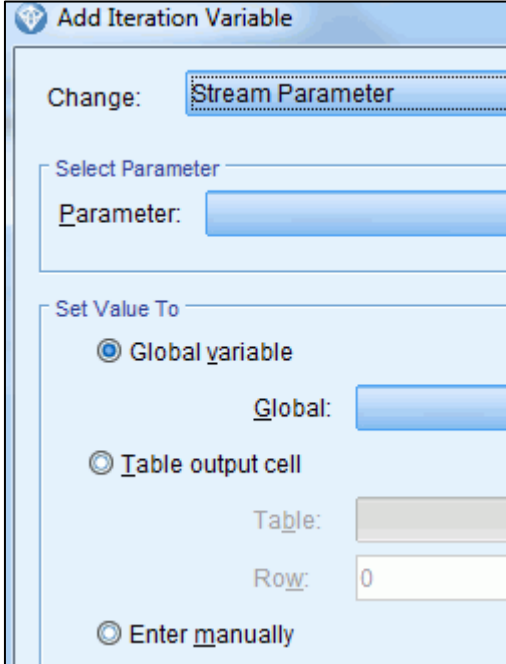
The iterations will use the iteration key values GENDER, REGION, AGE_CATEGORY, SEGMENT and HANDSET, respectively.

To attach clear names to the various Matrix tables in the output, you will give a Matrix output table a name of the form "<row field> BY CHURN", where <row field> is the row field in question.

In the previous task you have customized the name for the Matrix output table on the Output tab, in the Output name text box. When you want to specify the output name in a loop, you will look for the property "output name".


12. Click the **Add Variable**  button.

The results appear as follows:



The image shows the 'Add Iteration Variable' dialog box in IBM SPSS Modeler. The 'Change:' dropdown menu is set to 'Stream Parameter'. Below this, there is a 'Select Parameter' section with a 'Parameter:' text box. The 'Set Value To' section has three radio buttons: 'Global variable' (selected), 'Table output cell', and 'Enter manually'. The 'Global variable' section has a 'Global:' text box. The 'Table output cell' section has 'Table:' and 'Row:' text boxes, with 'Row:' set to '0'. The 'Enter manually' section is currently inactive.

By default, IBM SPSS Modeler will change the value of a Stream Parameter. In this example, however, you want to change a property of a node (the output name of the Matrix node).

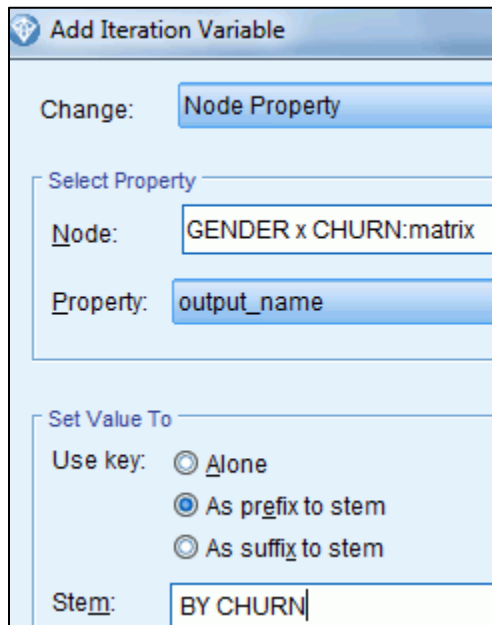
13. In the **Change** drop-down menu, select **Node Property**.
14. For **Node**, click the **Ellipses**  button, select the **GENDER x CHURN** Output Node, and then click **OK**.

You will select the appropriate property to specify the name for the output.

15. Click the pull-down menu for **Property**, scroll down, and then select **output_name**.
The Matrix output table name must be of the form <value of the key> BY CHURN, so you need to have the value of the key as a prefix to the stem " BY CHURN" (with a leading blank).

- Under **Set Value To**, select **As prefix to stem**, for **Stem** press the **Spacebar**, and then type **BY CHURN**.

The results appear as follows:



Add Iteration Variable

Change: Node Property

Select Property

Node: GENDER x CHURN:matrix

Property: output_name

Set Value To

Use key: ☐ Alone ☒ As prefix to stem ☐ As suffix to stem

Stem: BY CHURN

- Click **OK**.

The results appear as follows:

Options Messages Parameters Deployment Execution Globals Search Comments

Execution mode: ☐ Default (optional script) ☒ Looping / Conditional Execution

Conditional Looping

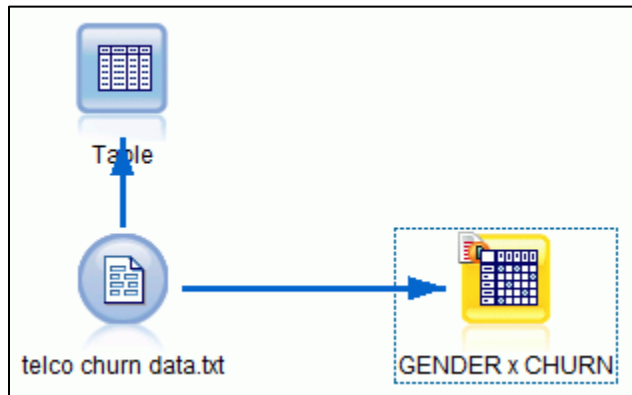
| Iteration | GENDER x CHURN:matrix.row | GENDER x CHURN:matrix.output_name |
|-----------|---------------------------|-----------------------------------|
| 1 | GENDER | GENDER BY CHURN |
| 2 | REGION | REGION BY CHURN |
| 3 | AGE_CATEGORY | AGE_CATEGORY BY CHURN |
| 4 | SEGMENT | SEGMENT BY CHURN |
| 5 | HANDSET | HANDSET BY CHURN |


The GENDER x CHURN.matrix.output_name column shows the custom name that will be attached to the Matrix output table in a run.

Notice that you can only have one iteration key, here defined by the GENDER x CHURN.matrix.row column. The other variables follow this lead. For example, you cannot use the Looping feature to create a double loop in order to iterate through a number of column fields in addition to cycling through a number of row fields.

18. Click **OK** to close the **Looping Execution** dialog box.

The results appear as follows:

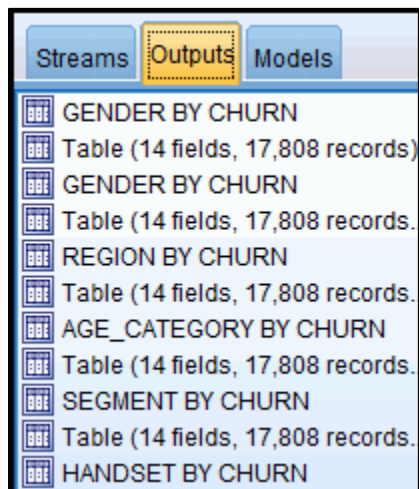


The Matrix node has an icon  in the left-upper corner that tells you that a loop has been set on the node.

In order to make looping work you need to run the entire stream, and not only the node that has the looping icon on it.

19. Click **the Run the current stream**  button in the toolbar.
20. Click the **Outputs** tab.

The results appear as follows:



Five Matrix tables have been produced and they are identified by their output name, as specified in the Looping settings.

Notice that the same Table node has also been executed in each iteration.

This completes the first workshop. You will clear the output, and then save the stream.

21. Right-click any output item on the **Outputs** tab, and then click **Delete All**.
You will close the stream without saving it.
22. From the **File** menu, click **Close Stream**; click **No** if asked to save your changes.
Leave IBM SPSS Modeler open for the next workshop.

This completes the first workshop. You have used IBM SPSS Modeler's Looping feature to cycle through a number of fields in the row dimension of a Matrix node, and you have customized the name for the output. In order to do so, you have set values for two properties of the Matrix node: the row property and the output_name property. Notice that these properties correspond to options in the dialog box.

You also observed that the Looping feature will not let you create a double loop, through both rows and columns. Using Python will enable you to create double loop, and more.

Workshop 2: Use Python to Automate Tasks

You work for a telecommunications firm and want to use the Matrix node to assess the importance of a number of fields to predict CHURN. In the past, you have created a stream using the Looping feature and now you want to paste the Python code and customize the pasted code.

Data file: telco churn data.txt

Stream file: workshop_2_start.str

Python files: workshop_2_task_2.py
workshop_2_task_3.py
workshop_2_task_4.py

Folder C:\Training\4188

Task 1. Examine and run a pasted Python script.

1. From the **File** menu, click **Open Stream**, click **workshop_2_start.str**, located in **C:\Training\4188** and then click **Open**.

This is the result of the previous workshop. A loop is set on the Matrix node.

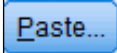
You will paste the Python code associated with the Looping feature, and then run the Python code.

2. Right-click the **Matrix** node, click **Looping/Conditional Execution**, and then click **Edit Looping Settings**.

The results appear as follows:

| Options | Messages | Parameters | Deployment | Execution | Globals | Search | Comments |
|--|----------------------------|------------------------------------|------------|-----------|---------|--------|----------|
| Execution mode: <input type="radio"/> Default (optional script) <input checked="" type="radio"/> Looping / Conditional Execution | | | | | | | |
| <div>Conditional Looping</div> | | | | | | | |
| Iteration | HANDSET x CHURN:matrix.row | HANDSET x CHURN:matrix.output_name | | | | | |
| 1 | GENDER | GENDER BY CHURN | | | | | |
| 2 | REGION | REGION BY CHURN | | | | | |
| 3 | AGE_CATEGORY | AGE_CATEGORY BY CHURN | | | | | |
| 4 | SEGMENT | SEGMENT BY CHURN | | | | | |
| 5 | HANDSET | HANDSET BY CHURN | | | | | |

You will examine the associated script by pasting the corresponding code.

3. Click the **Paste**  button, in the right lower corner of the dialog box.
The Script Editor opens, showing the Python code. The code might be quite overwhelming. Actually, the code is quite long because it is not optimal. You will examine a part of the code to verify that.
4. Scroll to the beginning of the script.
The results appear as follows:



The screenshot shows the Script Editor interface with the following elements:

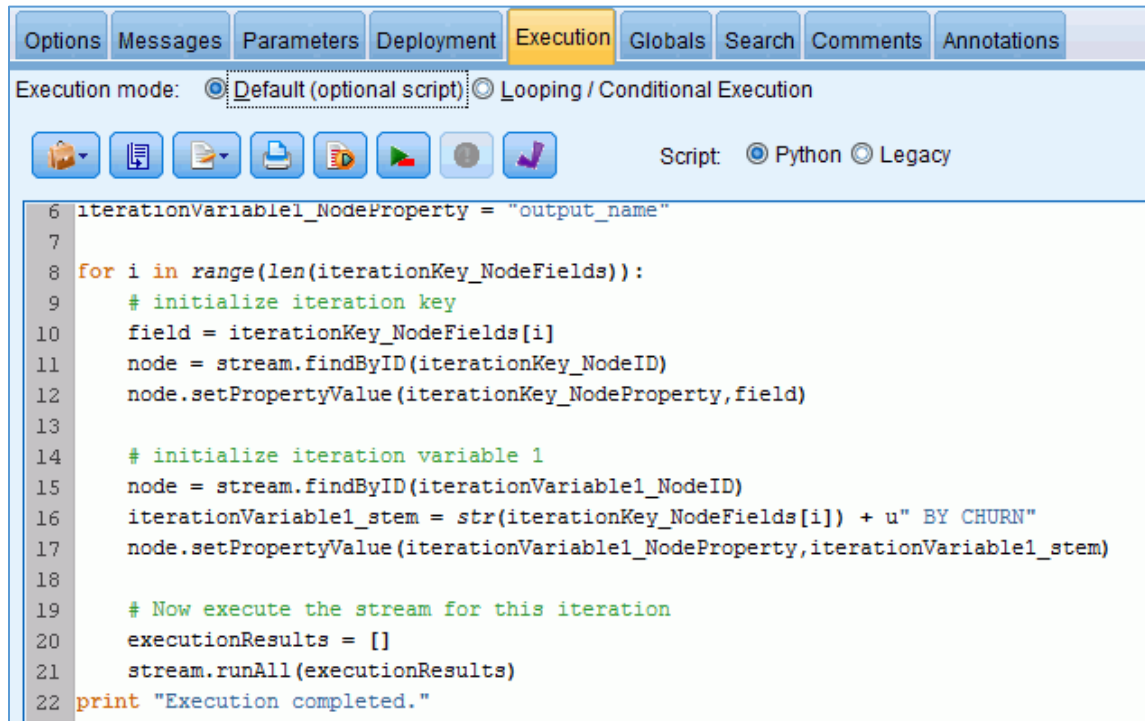
- Tabbed interface: Options, Messages, Parameters, Deployment, Execution (selected), Globals, Search, Comments, Annotations.
- Execution mode: ☒ Default (optional script) ☐ Looping / Conditional Execution
- Script type: ☒ Python ☐ Legacy
- Python code in the editor:

```
1 stream = modeler.script.stream()
2 iterationKey_NodeID = "id6RETIJL6DFL" # "HANDSET x CHURN":matrix
3 iterationKey_NodeProperty = "row"
4 iterationKey_NodeFields = [u'GENDER',u'REGION',u'AGE_CATEGORY',u'SEGMENT',u'HANDSET']
5 iterationVariable1_NodeID = "id6RETIJL6DFL" # "HANDSET x CHURN":matrix
6 iterationVariable1_NodeProperty = "output_name"
7
```

Line #2 and #5 both refer to the ID of the same Matrix node, so that two variables store the same content. (The node ID is used later in the script to locate the Matrix node in the stream.)

5. Scroll to **line #8** of the script.

The results appear similar to the following.



```
6 iterationVariable1_NodeProperty = "output_name"
7
8 for i in range(len(iterationKey_NodeFields)):
9     # initialize iteration key
10    field = iterationKey_NodeFields[i]
11    node = stream.findByID(iterationKey_NodeID)
12    node.setPropertyValue(iterationKey_NodeProperty, field)
13
14    # initialize iteration variable 1
15    node = stream.findByID(iterationVariable1_NodeID)
16    iterationVariable1_stem = str(iterationKey_NodeFields[i]) + u" BY CHURN"
17    node.setPropertyValue(iterationVariable1_NodeProperty, iterationVariable1_stem)
18
19    # Now execute the stream for this iteration
20    executionResults = []
21    stream.runAll(executionResults)
22    print "Execution completed."
```

On line #8, the `for i in range ...` defines a loop, iterating through the fields (GENDER, REGION, and so forth).

The indented lines are executed in each iteration. Now, note:

- Line #15: This locates the Matrix node in the stream, by finding the Matrix node by its node ID. This is done in each iteration, which is not efficient because it suffices to do that only once.
- Line #21: `stream.runAll(executionResults)`. This will run all the terminal nodes in each iteration. Rather than executing all terminal nodes, only the Matrix node should be executed. (This was the reason why the Table node is executed in each iteration.)

Now that you have walked through some of the code, you will execute it.

6. Click **Run this script**  in the toolbar of the Script Editor.

A series of Table and Matrix outputs tables is generated. Again, the Table node output is the same for all iterations.


You will close the Script Editor.

7. Set the focus on the Script Editor, and then click **OK** to close the Script Editor.
Note: If the Script Editor is hidden, click Tools\Stream Properties\Execution.
You will delete all output.
8. Right-click any output item on the **Outputs** tab, and then click **Delete All**. If asked to save output, click **Delete All**.
Leave the stream open for the next task.

Task 2. Examine and run an existing Python script.

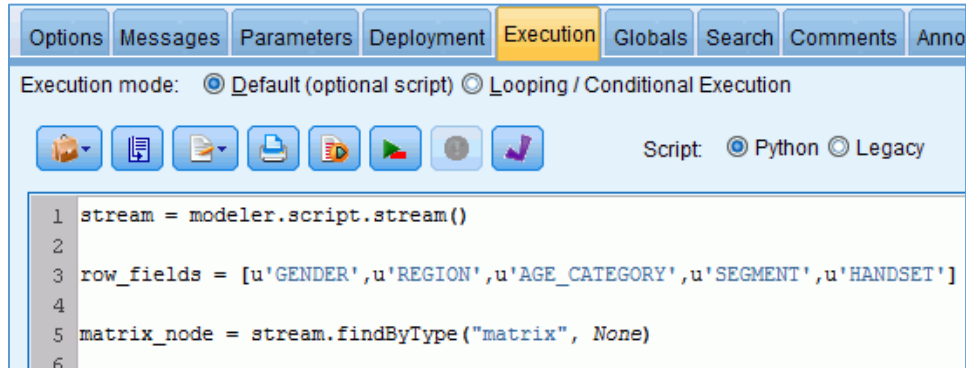
In the previous task you have pasted a Python script from a dialog box, walked through the code, and noticed that the code could be more efficient at some points (no duplication of code, nor running all terminal nodes in the stream in each iteration).

In this task you will open a script that will also run a Matrix node five times, but which is more efficient.

1. From the **Tools** menu, click **Stream Properties**, and then click **Execution**.
The Script Editor contains the pasted script from the previous task. You will delete the content.
2. Press **Ctrl+A**, and then press the **Delete** key.
The content of Script Editor is cleared.
You will open an existing script.
3. Click the **File options**  button in the toolbar of the Script Editor, click **Import**, click **workshop_2_task_2.py**, located in the **C:\Training\4188** folder, and then click **Open**.
You will walk through some lines.

4. Scroll to the beginning of the script.

The results appear as follows:



| Line(s) | Description |
|---------|---|
| 1 | Gets the stream information, and stores it in a variable named stream. |
| 3 | Specifies the fields to cycle through: GENDER, REGION, and so forth. |
| 5 | Locates the Matrix node, now by its type, rather than by its ID as in the pasted code in the previous task. In general, it is easier to locate a node by its type than by its node ID. |

Notice that the statement to find the Matrix node is outside the `for` loop, because the Matrix node has to be located only once.


5. Scroll to **line #7**.

The results appear as follows:

```
6
7 for row_field in row_fields:
8     matrix_node.setPropertyValue ("row", row_field)
9     text = str (row_field) + u" BY CHURN"
10    matrix_node.setPropertyValue ("output_name",text)
11
12    executionResults = []
13    matrix_node.run (executionResults)
```

| Line(s) | Description |
|---------|---|
| 7 | Initiates a loop using <code>row_field</code> as variable name to cycle through the row fields (GENDER, REGION, and so forth). |
| 8-10 | Sets the two properties for the Matrix node, the row field and the name for the output (<code>row_field</code> and <code>output_name</code> , respectively). |
| 12 | Declares and initializes a variable that will store the output. |
| 13 | Runs the Matrix node. Only the node of interest will be executed, not all terminal nodes as the pasted script did. |

The code is simpler and more efficient than the pasted code. You will verify that this code also runs the Matrix node for the specified row fields.

6. Click the **Run this script**  button in the toolbar of the Script Editor.
The matrices will be generated.
You will close the Script Editor.
7. If necessary, set focus on the Script Editor, and then click **OK** to close the Script Editor.
You will delete all output.
8. Right-click any output item on the **Outputs** tab, and then click **Delete All**. If asked to save output, click **Delete All**.

Task 3. Use Modeler's data model to identify categorical variables.

In the previous task you have hard coded the categorical fields in the script.

Alternatively, you can get the categorical fields from the data model. The data model stores information about the fields. Because this is node-specific information, the data model method is defined for each node.

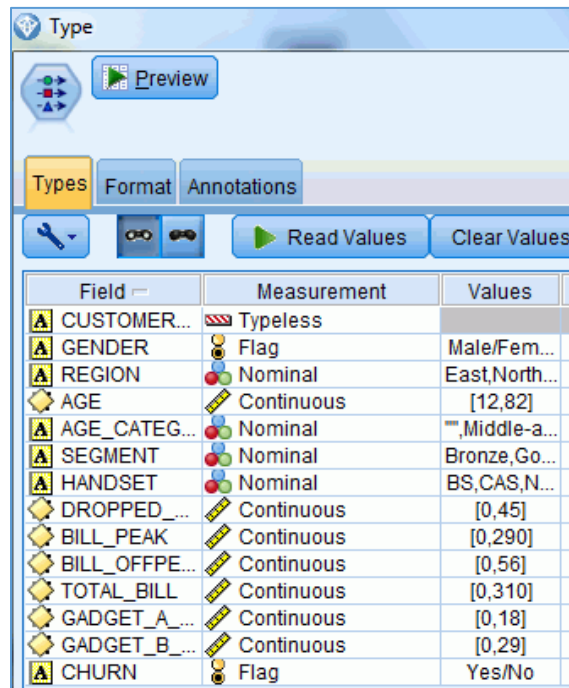
Also, the field information that goes into a node can be different from the information that goes out from of a node. For example, when you use the Filter node to rename or exclude fields, the field information that goes into the Filter node is different from the field information that goes out from the Filter node.

In this task you will open a script that uses the data model and iterates through all categorical fields and cross tabulates them by CHURN. Of course, one of the categorical fields is CHURN itself, and therefore a condition is needed to prevent cross tabulating CHURN by itself.

You will first examine the measurement levels of the fields.


1. Edit the **Type** node.

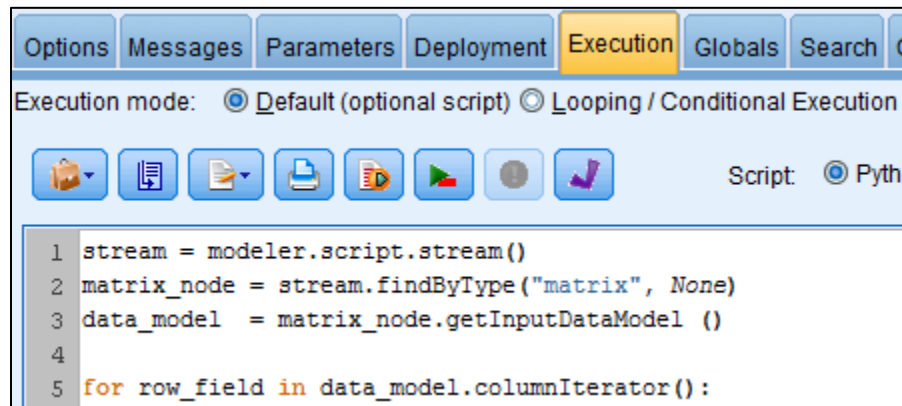
The results appear as follows:



| Field | Measurement | Values |
|---------------|-------------|---------------|
| CUSTOMER... | Typeless | |
| GENDER | Flag | Male/Fem... |
| REGION | Nominal | East,North... |
| AGE | Continuous | [12,82] |
| AGE_CATEG... | Nominal | -,Middle-a... |
| SEGMENT | Nominal | Bronze,Go... |
| HANDSET | Nominal | BS,CAS,N... |
| DROPPED_... | Continuous | [0,45] |
| BILL_PEAK | Continuous | [0,290] |
| BILL_OFFPE... | Continuous | [0,56] |
| TOTAL_BILL | Continuous | [0,310] |
| GADGET_A_... | Continuous | [0,18] |
| GADGET_B_... | Continuous | [0,29] |
| CHURN | Flag | Yes/No |

The Measurement column shows that GENDER, REGION, AGE_CATEGORY, SEGMENT, HANDSET, and CHURN are typed as categorical (flag, nominal) fields.

2. Click **OK** to close the **Type** dialog box.
You will open a script that uses the data model information to cycle through the row fields and executes the Matrix node for all categorical fields, except CHURN itself.
3. From the **Tools** menu, click **Stream Properties**, and then click **Execution**.
The editor contains the script from the previous task. You will delete the content.
4. Press **Ctrl+A**, and then press the **Delete** key.
You will open the script for this task.
5. Click the **File options**  button in the toolbar of the Script Editor, click **Import**, click **workshop_2_task_3.py**, located in the **C:\Training\4188** folder, and then click **Open**.
6. Scroll to the beginning of the script.
The results appear as follows:



| Line(s) | Description |
|---------|---|
| 3 | Stores the data model in a variable named <code>data_model</code> . The data model will be obtained from the point where the fields go into the Matrix node by use of the <code>getInputDataModel</code> method. Because the fields are instantiated in the upstream Type node, all the fields' meta data (storages, measurement levels, roles) is known and contained in the <code>data_model</code> variable. |

7. Scroll to **line #5**.

The results appear as follows:

```

3 data_model = matrix_node.getInputDataModel ()
4
5 for row_field in data_model.columnIterator():
6     if row_field.isMeasureDiscrete() and str (row_field) != "CHURN":
7         matrix_node.setPropertyValue("row", row_field)
8         text = str (row_field) + u" BY CHURN"
9         matrix_node.setPropertyValue("output_name",text)
10
11     executionResults = []
12     matrix_node.run (executionResults)

```

| Line(s) | Description |
|---------|--|
| 5 | Initiates a loop. The word "column" in <code>columnIterator</code> is a synonym for "field"; thus, this will iterate through the fields in the dataset (as defined at the entry point of the Matrix node). |
| 6 | Specifies the conditions for running the Matrix node. Two conditions must be fulfilled: The row field: <ul style="list-style-type: none"> • must be categorical; this is checked by the <code>isMeasureDiscrete</code> method, which returns true if the field is flag, nominal or ordinal. • must not be CHURN itself; inequality in Python is represented by the <code>!=</code> operator. |
| 7 - 12 | Set the properties of the Matrix node, and run the Matrix node. Notice that the lines are indented from line #7 onwards, telling you that these lines should only be executed when the conditions in line #6 are met. |



8. Click the **Run this script** button in the toolbar of the script editor.

The Matrix tables will be produced.

You will close the Script Editor.

9. If necessary, set focus on the Script Editor, and then click **OK** to close the Script Editor.
You will delete all output.
10. Right-click any output item on the **Outputs** tab, and then click **Delete All**. . If asked to save output, click **Delete All**.

Task 4. Use Modeler's data model to identify categorical variables and their role.

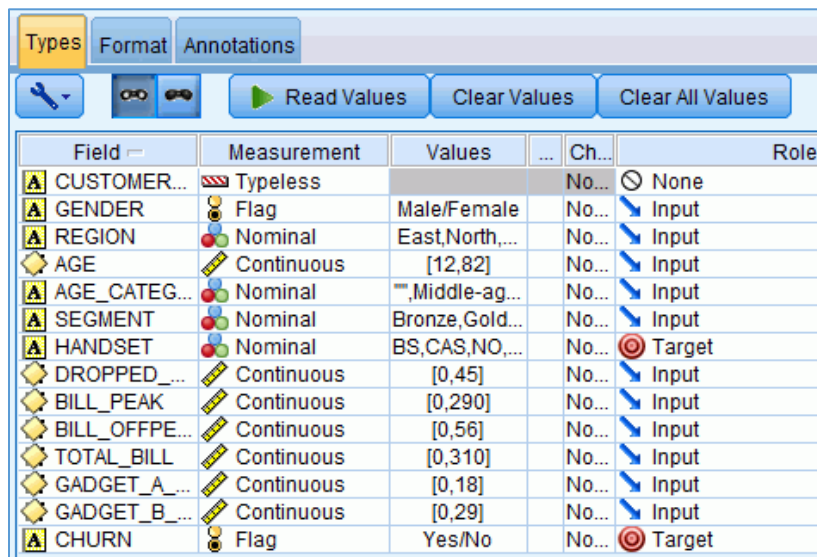
In the previous task you have hard coded that the column field should not be CHURN, to not execute a CHURN by CHURN Matrix node.

Instead of hard coding the field, you could write code to cross tabulate all categorical fields with role Input by all categorical fields with role Target. When GENDER, REGION, and so forth have role Input, CHURN has role Target, all required tables will be generated.

You will first examine the roles of the fields.

1. Edit the **Type** node.

The results appear as follows:



| Field | Measurement | Values | Ch... | Role |
|---------------|-------------|----------------|-------|--------|
| CUSTOMER... | Typeless | | No... | None |
| GENDER | Flag | Male/Female | No... | Input |
| REGION | Nominal | East,North,... | No... | Input |
| AGE | Continuous | [12,82] | No... | Input |
| AGE_CATEG... | Nominal | ""Middle-ag... | No... | Input |
| SEGMENT | Nominal | Bronze,Gold... | No... | Input |
| HANDSET | Nominal | BS,CAS,NO... | No... | Target |
| DROPPED_... | Continuous | [0,45] | No... | Input |
| BILL_PEAK | Continuous | [0,290] | No... | Input |
| BILL_OFFPE... | Continuous | [0,56] | No... | Input |
| TOTAL_BILL | Continuous | [0,310] | No... | Input |
| GADGET_A_... | Continuous | [0,18] | No... | Input |
| GADGET_B_... | Continuous | [0,29] | No... | Input |
| CHURN | Flag | Yes/No | No... | Target |

The Role column shows that GENDER, REGION, AGE_CATEGORY, and SEGMENT have role Input, HANDSET and CHURN have role Target.


Cross tabulating all categorical fields with role Input by all categorical fields with role Target will generate $4 * 2 = 8$ tables.

2. Click **OK** to close the **Type** dialog box.
3. From the **Tools** menu, click **Stream Properties**, and then click **Execution**.

The editor contains the script from the previous task. You will delete the content.

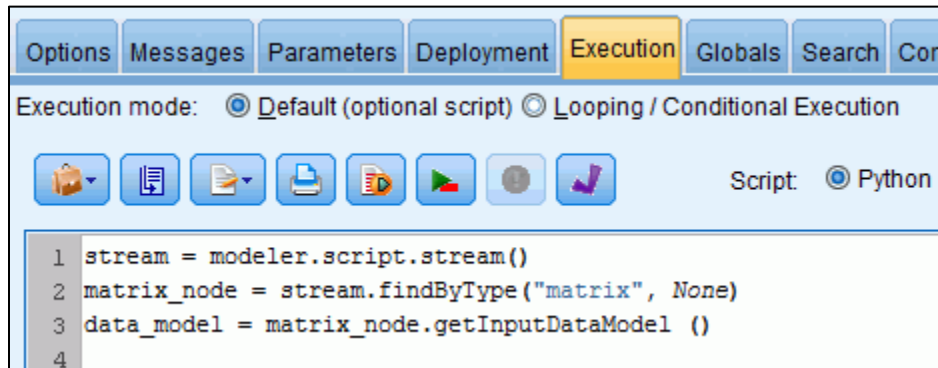
4. Press **Ctrl+A**, and then press the **Delete** key.

You will open the script for this task.

5. Click the **File options**  button in the toolbar of the Script Editor, click **Import**, click **workshop_2_task_4.py**, located in the **C:\Training\4188** folder, and then click **Open**.

6. Scroll to the beginning of the script.

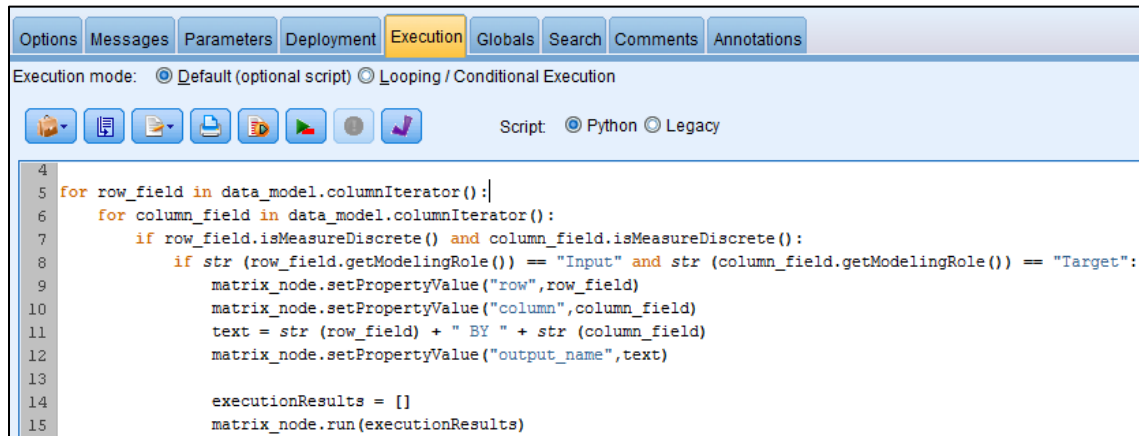
The results appear as follows:



| Line(s) | Description |
|---------|---|
| 3 | Same as previous. Stores the data model in a variable named <code>data_model</code> . The data model will be obtained from the point where the fields go into the Matrix node. Because the fields are instantiated in the upstream Type node, all the fields' meta data (storages, measurement levels, roles) is known and contained in the <code>data_model</code> variable. |

7. Scroll to **line #5**.

The results appear as follows:



```

4
5 for row_field in data_model.columnIterator():
6     for column_field in data_model.columnIterator():
7         if row_field.isMeasureDiscrete() and column_field.isMeasureDiscrete():
8             if str(row_field.getModelingRole()) == "Input" and str(column_field.getModelingRole()) == "Target":
9                 matrix_node.setPropertyValue("row", row_field)
10                matrix_node.setPropertyValue("column", column_field)
11                text = str(row_field) + " BY " + str(column_field)
12                matrix_node.setPropertyValue("output_name", text)
13
14                executionResults = []
15                matrix_node.run(executionResults)

```

| Line(s) | Description |
|---------|---|
| 5 | Initiates a loop through the fields. Fields in this loop will act as row fields in the Matrix node. Recall, the word "column" in <code>columnIterator</code> is a synonym for "field"; it does not refer to the column field in the Matrix node. |
| 6 | Initiates a loop through the fields. Fields in this loop will act as column fields in the Matrix node. |
| 7 | Specifies the first condition for running the Matrix node: both row and column field must be categorical. |
| 8 | Specifies the second condition for running the Matrix node: the row field must have role Input, the column field role Target. Notice the operator for equality <code>==</code> . |



8. Click the **Run this script** button in the toolbar of the Script Editor.
Eight Matrix tables are generated, as was required.
You will close the Script Editor.
9. If necessary, set focus on the Script Editor, and then click **OK** to close the Script Editor.
You will delete all output.
10. Right-click any output item on the **Outputs** tab, and then click **Delete All**.
This completes the second workshop. You have pasted a default script and explored alternative ways, using the data model property.
You will close the stream without saving it.
11. From the **File** menu, click **Close Stream**; click **No** if asked to save your changes.
Leave IBM SPSS Modeler open for the next workshop.

Workshop 3: Explore Scripts

You will run a number of scripts to examine whether they could be of use in your own work.

Data file: **telco churn data.txt**

Stream file: **workshop_3_task_1_start.str**

workshop_3_task_2_start.str

workshop_3_task_3_start.str

workshop_3_task_4_start.str

workshop_3_task_5_start.str

workshop_3_task_6_start.str

Folder **C:\Training\4188**

Task 1. Change field names downstream from an Aggregate node.

A common situation is that you have to aggregate your data. For example, you have transactional data, with multiple records per customer, storing fields such as number of peak calls, number of off peak calls, and number of text messages, all recorded per day. In this case, you want to create a dataset with one record per customer, storing the sum of each field (sum of peak calls, sum of off peak calls, sum of text messages, and so forth).

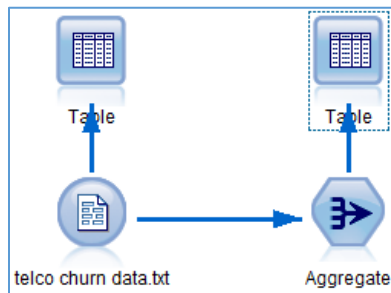
IBM SPSS Modeler's Aggregate procedure will automatically assign names to the new fields. For example, when you take the sum of a field peakcalls, the aggregated field will be named peakcalls_Sum.

Downstream from the Aggregate node, you can rename the fields, but because the number of aggregated fields might be substantial, this could be quite time consuming.

The following script will help out.

1. From the **File** menu, click **Open Stream**, and then open **workshop_3_task_1.str**, located in the **C:\Training\4188** folder.

The results appear as follows:



2. Run the **Table** node downstream from the **Aggregate** node.
The aggregated fields are derived from the original field names, suffixed by **_Sum**.
3. Click **OK** to close the **Table** output window.
Rather than adding a Filter node and renaming the fields yourself, you can use a script to automate this task.
4. From the **Tools** menu, click **Stream Properties**, and then click **Execution**.
5. Scroll to the beginning of the script.

The results appear as follows:

```

1 stream = modeler.script.stream()
2 aggr_node = stream.findByType("aggregate", None)
3
4 x = aggr_node.getXPosition()
5 y = aggr_node.getYPosition()
6
7 filter_node = stream.createAt ("filter", "Filter", x + 100, y)
8 stream.link(aggr_node, filter_node)
9

```

| Line(s) | Description |
|---------|---|
| 2 | Locates the aggregate node in the stream. |
| 4-5 | Gets the x and y coordinates of the aggregate node on the stream canvas. |
| 7 | Creates a Filter node on the stream canvas, and places it to the right of the Aggregate node. |
| 8 | Creates a connection from the Aggregate node to the Filter node. |

6. Scroll down to **line #10**.

The results appear as follows:

```

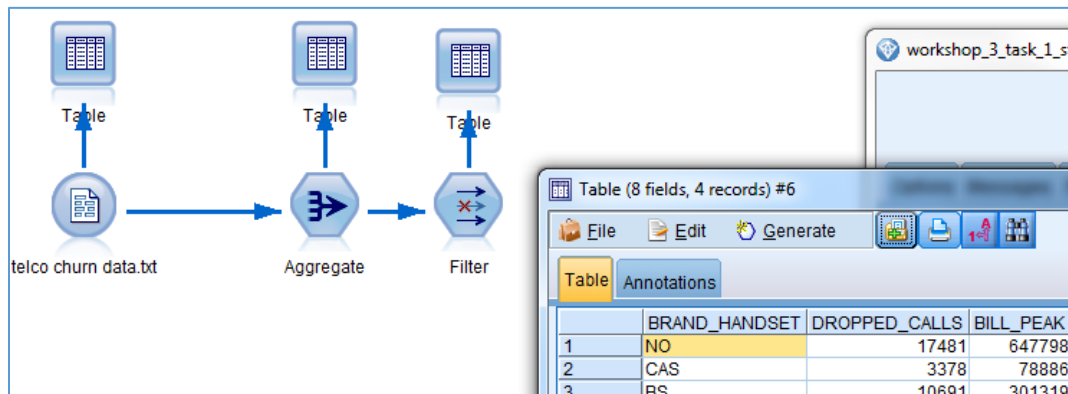
9
10 data_model = filter_node.getInputDataModel ()
11
12 for field in data_model.columnIterator():
13     current_name = field.getColumnNames()
14     if current_name.endswith("_Sum"):
15         new_name = current_name.replace("_Sum","")
16         filter_node.setKeyedPropertyValue("new_name", current_name, new_name)
17

```

| Line(s) | Description |
|---------|--|
| 10 | Gets the data model, at the point where the fields go into the Filter node. |
| 12 | Initiates a loop through the fields. |
| 13 - 16 | If the current field name ends with "_Sum", replaces "_Sum" by an empty string (""). |

7. Click the **Run this script**  button.

The results appear similar to the following:



A Filter and Table node have been added to the stream, and the Table node is executed. The Table output window shows that the suffix "_Sum" has been dropped from the field names. The Filter node has taken care of that. To verify, edit the Filter node.

8. Click **OK** to close the **Table** output window.

9. Edit the **Filter** node.

The results appear as follows:

| Filter Annotations | | |
|--|--------|-------------------|
| Fields: 7 in, 0 filtered, 6 renamed, 7 | | |
| Field | Filter | Field |
| HANDSET | → | HANDSET |
| DROPPED_CALLS_Sum | → | DROPPED_CALLS |
| BILL_PEAK_Sum | → | BILL_PEAK |
| BILL_OFFPEAK_Sum | → | BILL_OFFPEAK |
| TOTAL_BILL_Sum | → | TOTAL_BILL |
| GADGET_A_REVENUES_Sum | → | GADGET_A_REVENUES |
| GADGET_B_REVENUES_Sum | → | GADGET_B_REVENUES |

The script has replaced the field names.

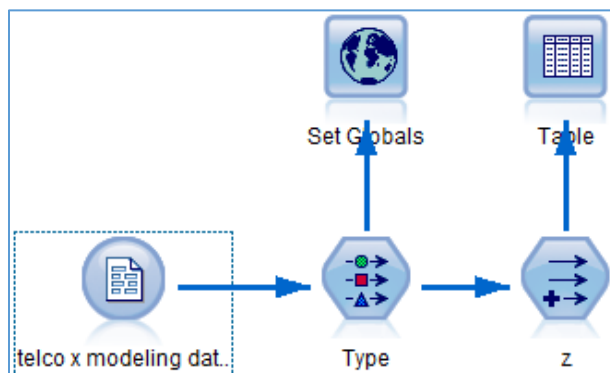
10. Click **OK** to close the **Filter** dialog box.
This completes this task.
11. From the **File** menu, click **Close Stream**; click **No** if asked to save your changes.

Task 2. Control the order of execution.

When you execute a stream that includes multiple terminal nodes, you do not have control over the order of execution. Using scripting will let you determine the order in which the terminal nodes must be executed.

1. From the **File** menu, click **Open Stream**, and then open **workshop_3_task_2_start.str**, located in the **C:\Training\4188** folder.

The results appear as follows:

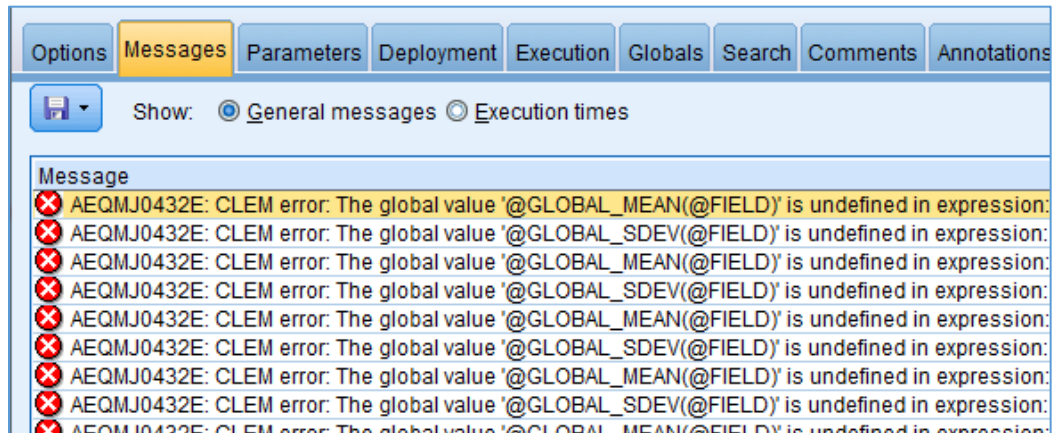


2. Edit the **Derive** node named **z**.


Standardized fields are computed by subtracting the mean of the field in question, and dividing the difference by the standard deviation of the field. Means and standard deviations are computed in the Set Globals node.

3. Click **OK** to close the **Derive** dialog box.
4. Run the **Table** node.

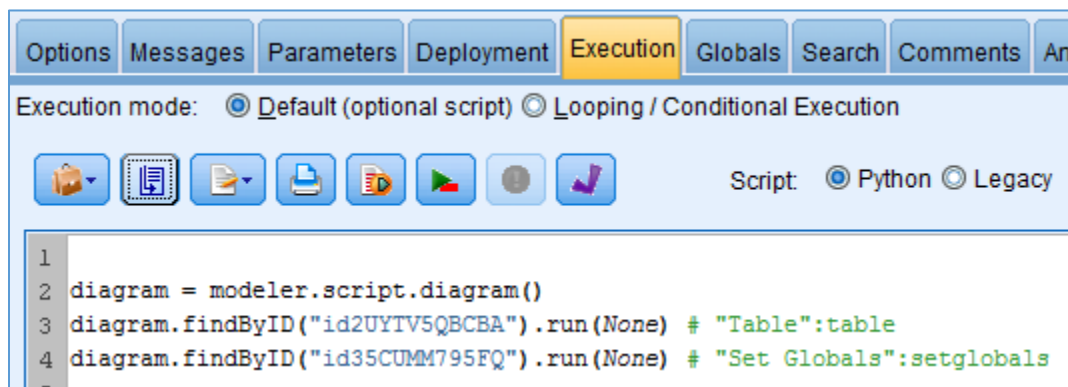
The results appear as follows:




The reason is that the Derive node uses values from global variables. So, that node should always be executed first.

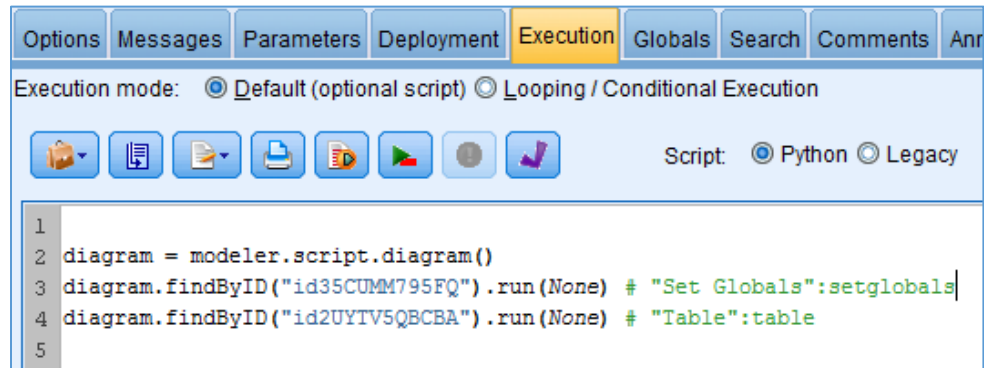
5. Click **OK** to close the **Messages** window.
6. From the **Tools** menu, click **Stream Properties**, and then click **Execution**.
7. Click the **Append default script**  button.


The results appear as follows:



The Append default script  button has pasted the code to run all terminal nodes. Here, the order is not correct because the Set Globals node should be run first.

8. Cut and paste **line #4**, so that it appears before running the **Table** node.
The results appear as follows:



9. Click the **Run this script**  button.
10. Scroll to the right in the **Table** output window, to locate one of the standardized fields (they start with "z").
The standardized fields are created.
11. Click **OK** to close the **Table** output window.
12. Click **OK** to close the Script Editor.
This completes this task.
13. From the **File** menu, click **Close Stream**; click **No** if asked to save your changes.

Task 3. Create standardized fields.

In the previous stream, you had a Set Globals and Derive node already included in the stream.

Now, suppose that you do not want to add these nodes yourself, but that you want to automatically derive standardized fields for all (continuous) fields in the dataset.

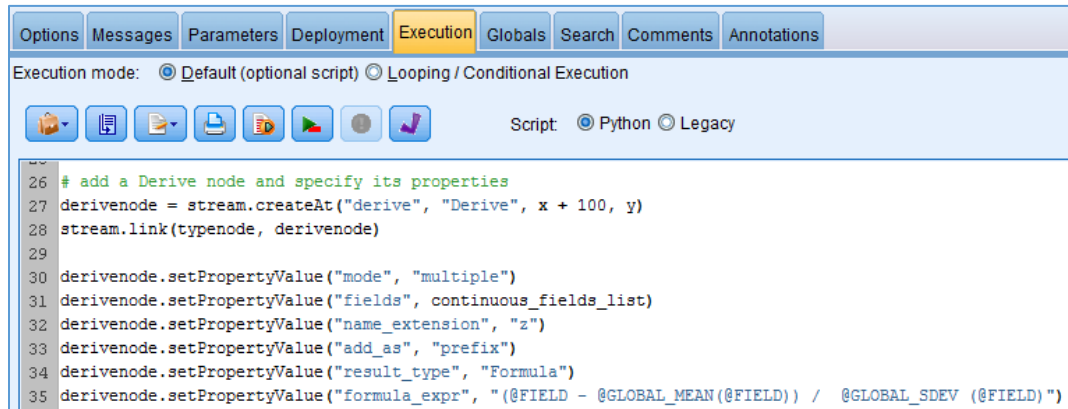
Scripting will enable you to do that.

1. From the **File** menu, click **Open Stream**, and then open **workshop_3_task_3_start.str**, located in the **C:\Training\4188** folder.

The objective is to derive standardized fields, for all continuous fields.


2. From the **Tools** menu, click **Stream Properties**, and then click **Execution**.
3. Scroll to **line #26**.

The results appear as follows:

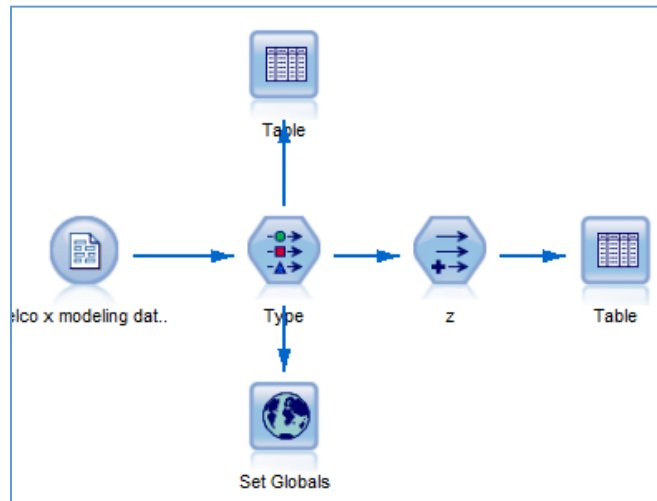


```
26 # add a Derive node and specify its properties
27 derivenode = stream.createAt("derive", "Derive", x + 100, y)
28 stream.link(typenode, derivenode)
29
30 derivenode.setPropertyValue("mode", "multiple")
31 derivenode.setPropertyValue("fields", continuous_fields_list)
32 derivenode.setPropertyValue("name_extension", "z")
33 derivenode.setPropertyValue("add_as", "prefix")
34 derivenode.setPropertyValue("result_type", "Formula")
35 derivenode.setPropertyValue("formula_expr", "(@FIELD - @GLOBAL_MEAN(@FIELD)) / @GLOBAL_SDEV (@FIELD)")
```

| Line(s) | Description |
|---------|---|
| 27-28 | A Derive node is created, and connected downstream from the Type node. |
| 30-35 | All the properties for the Derive node are specified, such as multiple mode, and the formula. |

4. Click the **Run this script**  button.
The script executes a Table node to verify that the new fields have been created.
5. Scroll all to the right in the **Table** output window, to verify that standardized fields have been created (they start with "z").

6. Click **OK** to close the **Table** output window.
The results appear as follows:



The Set Globals, Derive node (named z), and Table node have been added to the stream.

7. Click **OK** to close the Script Editor.
This completes this task.
8. From the **File** menu, click **Close Stream**; click **No** if asked to save your changes.

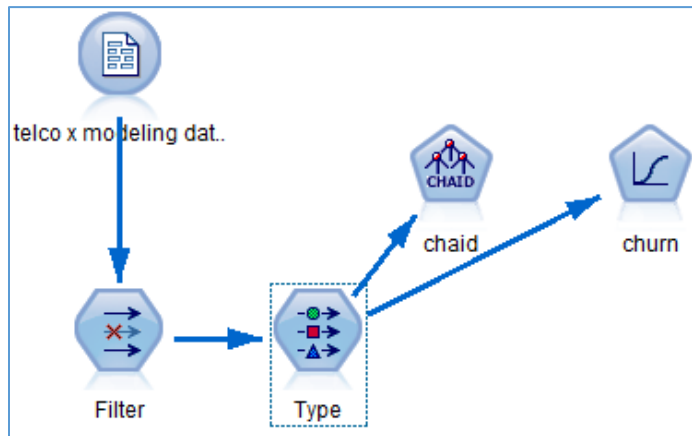
Task 4. Assess the importance of a predictor by leaving it out.

To illustrate how to automate running models, assume that you have a set of predictors X1 to X10 for a certain target Y.

You want to assess the importance of a predictor by leaving that predictor out. For example, assume you want to predict Y with X1 to X10, and want to assess the importance of each X field by excluding it from the model. So, you will first predict Y using X2 to X10, then using X1, X3 to X10, up to the tenth iteration, where you will use X1 to X9 and leave out X10.

1. From the **File** menu, click **Open Stream**, and then open **workshop_3_task_4_start.str**, located in the **C:\Training\4188** folder.

The results appear as follows:



In this example, we want to run the Logistic model multiple times, each time with leaving one predictor out.

2. From the **Tools** menu, click **Stream Properties**, and then click **Execution**.
If preferred, you can walk through the script. Here, you will just run the script.

3. Click the **Run this script**  button.


It will take about 2 minutes to complete execution.


The script will execute an Analysis node to evaluate the models.


A section of the results appears follows:


Analysis


Annotations

 Collapse All


 Expand All

 Results for output field churn


 Individual Models

 Comparing \$L-churn with churn


| | | |
|---------|--------|--------|
| Correct | 26,958 | 84.86% |
| Wrong | 4,811 | 15.14% |
| Total | 31,769 | |

 Comparing \$L1-churn with churn


| | | |
|---------|--------|--------|
| Correct | 26,957 | 84.85% |
| Wrong | 4,812 | 15.15% |
| Total | 31,769 | |

 Comparing \$L2-churn with churn

| | | |
|---------|--------|--------|
| Correct | 25,836 | 81.32% |
| Wrong | 5,933 | 18.68% |
| Total | 31,769 | |

 Comparing \$L3-churn with churn

| | | |
|---------|--------|--------|
| Correct | 26,721 | 84.11% |
| Wrong | 5,048 | 15.89% |
| Total | 31,769 | |

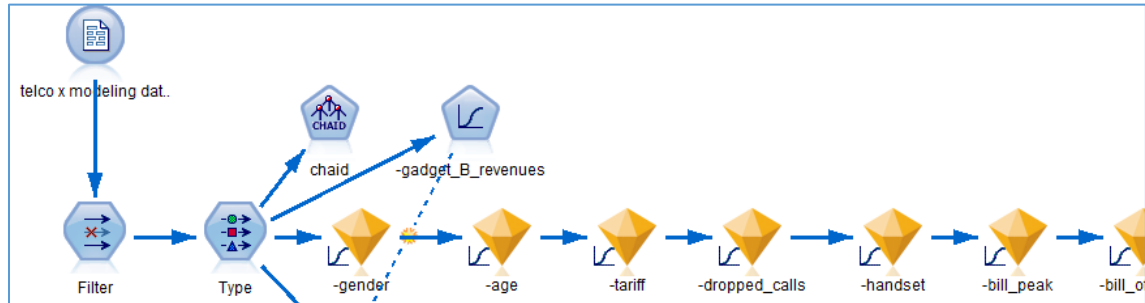
 Comparing \$L4-churn with churn

| | | |
|---------|--------|--------|
| Correct | 19,312 | 60.79% |
| Wrong | 12,457 | 39.21% |
| Total | 31,769 | |

The fifth model is only 60.79% accurate. To identify what model that is, you can examine the stream.

4. Click **OK** to close the **Analysis** output window.

- Click **OK** to close the Script Editor.
A section of the results appears as follows:



The fifth model excluded handset.

This completes this task.

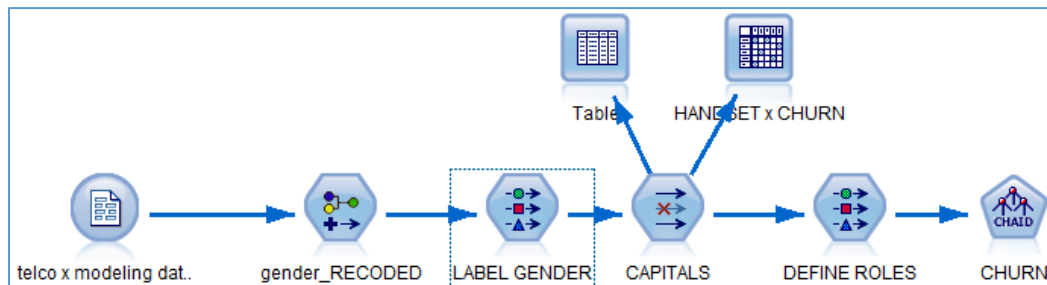
- From the **File** menu, click **Close Stream**; click **No** if asked to save your changes.

Task 5. List all nodes in a stream.

This example is taken from the scripting documentation that is installed with the software (*ModelerScriptingAutomation.pdf*).


- From the **File** menu, click **Open Stream**, and then open **workshop_3_task_5_start.str**, located in the **C:\Training\4188** folder.

The results appear as follows:



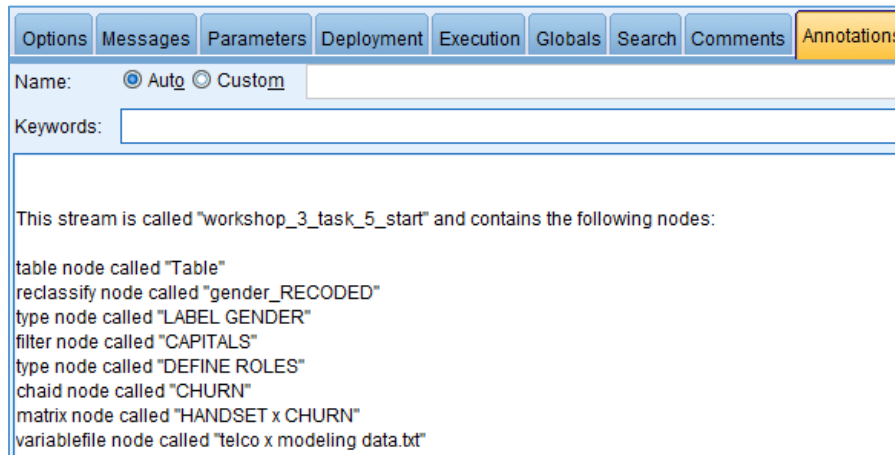
This stream recodes the string field named gender into numbers, and defines the original values as value labels.

The script contained in the stream will list all nodes.

- From the **Tools** menu, click **Stream Properties**, and then click **Execution**.
If preferred, you can walk through the script. Here, you will just run the script.
- Click the **Run this script**  button.
The stream has created an annotation for the stream.

- Click the **Annotations** tab.

The results appear as follows:



Options Messages Parameters Deployment Execution Globals Search Comments Annotations

Name: ☒ Auto ☐ Custom

Keywords:

This stream is called "workshop_3_task_5_start" and contains the following nodes:

- table node called "Table"
- reclassify node called "gender_RECODED"
- type node called "LABEL GENDER"
- filter node called "CAPITALS"
- type node called "DEFINE ROLES"
- chaid node called "CHURN"
- matrix node called "HANDSET x CHURN"
- variablefile node called "telco x modeling data.txt"

The stream annotation lists the nodes.

- Click **OK** to close the **Stream Properties** dialog box.
This completes this task.
- From the **File** menu, click **Close Stream**; click **No** if asked to save your changes.
This script could serve as a start to list all nodes in all streams that you have, to subsequently locate the node you are looking for.

Task 6. Recode a string field into a numeric field, with value labels.

IBM SPSS Statistics has a procedure to automatically recode a string field into a numeric field, with the original string values as value labels.

The script in this task mimics this functionality.

- From the **File** menu, click **Open Stream**, and then open **workshop_3_task_6_start.str**, located in the **C:\Training\4188** folder.

The results appear as follows:

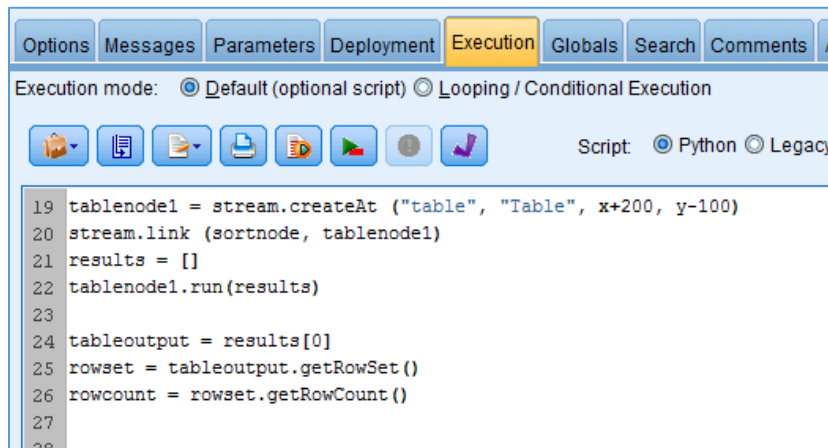


This stream imports data from a text file, and then filters fields.

You will examine the data by previewing it in the Filter node.

2. Edit the **Filter** node, and then click **Preview**.
Handset is a string field. We want to "automatically recode" this field. The alphabetically first handset must be assigned the code 1, and so forth.
3. Click **OK** to close the **Preview** output window.
4. Click **OK** to close the **Filter** dialog box.
5. From the **Tools** menu, click **Stream Properties**, and then click **Execution**.
6. Scroll to **line #19**.

The results appear as follows:



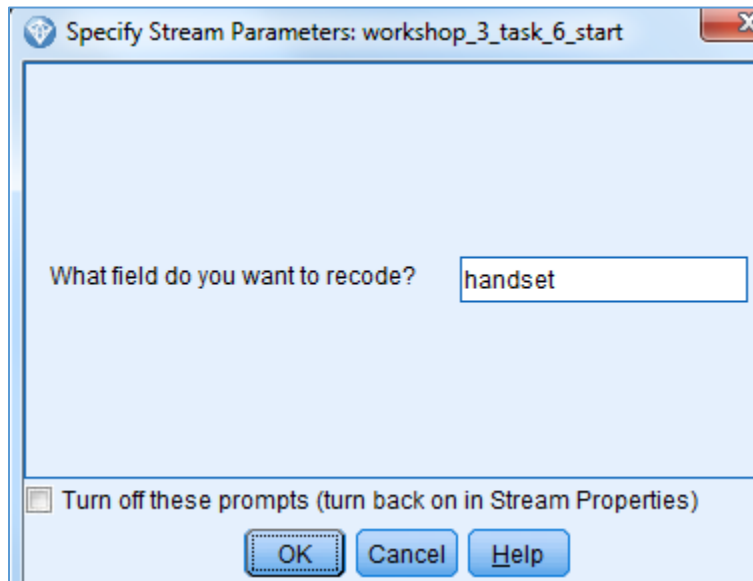
```
19 tablename = stream.createAt ("table", "Table", x+200, y-100)
20 stream.link (sortnode, tablename)
21 results = []
22 tablename.run(results)
23
24 tableoutput = results[0]
25 rowset = tableoutput.getRowSet()
26 rowcount = rowset.getRowCount()
27
28
```

A Table node is created and executed in lines 19 - 22. So, this will produce table output. You can read the content in this table, which is demonstrated in lines 24 - 26: this code stores the output in a variable named `tableoutput`, gets the rows from this table (actually the values of the field that must automatically be recoded), and determines the number of rows in the table (actually the number of values that must be recoded).

7. Click **OK** to close the Script Editor. (You will not run the script from the Script Editor, but run the stream instead.)

8. Click the **Run the current stream**  button.

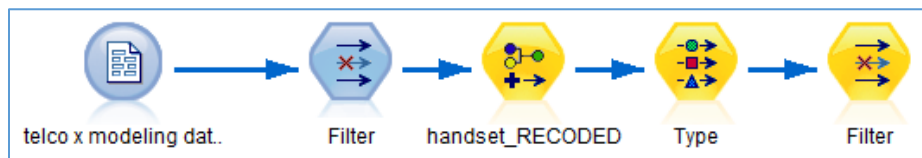
The results appear similar to the following:



You are prompted to enter the name of the field that you want to recode.

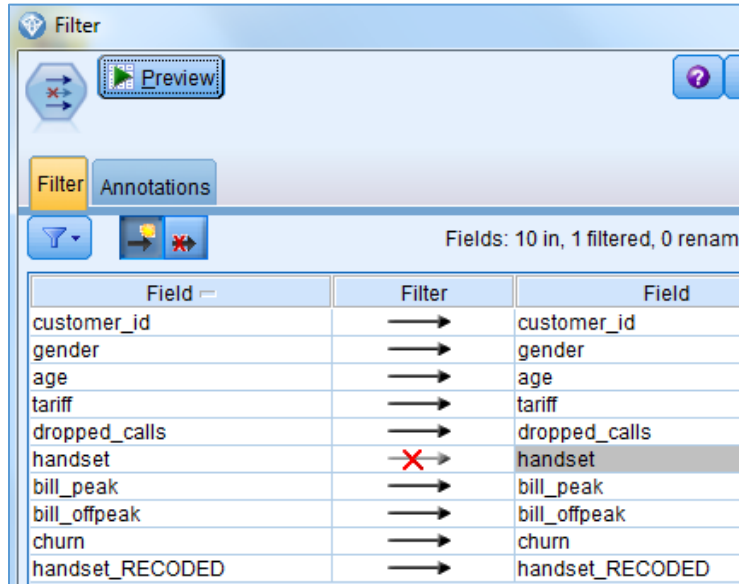
9. Type **handset** in the text box, if that field is not yet specified.
10. Click **OK** to close the prompt dialog box.
11. Close any **Table** output window that is displayed.

The results appear as follows:



Nodes have been generated that will recode handset.

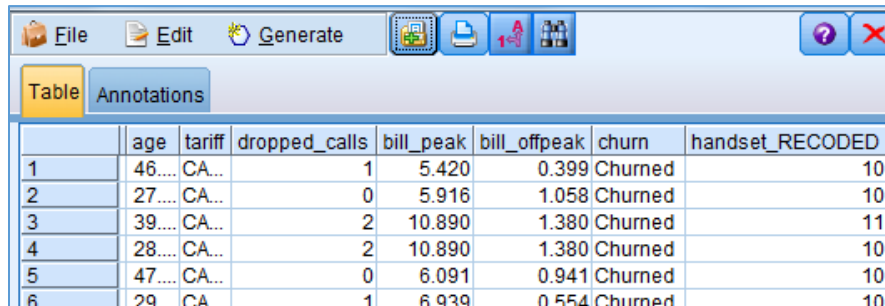
12. Edit the **Filter** node that is downstream from the **Type** node.
The results appear as follows:



| Field | Filter | Field |
|-----------------|--------|-----------------|
| customer_id | → | customer_id |
| gender | → | gender |
| age | → | age |
| tariff | → | tariff |
| dropped_calls | → | dropped_calls |
| handset | ✗ | handset |
| bill_peak | → | bill_peak |
| bill_offpeak | → | bill_offpeak |
| churn | → | churn |
| handset_RECODED | → | handset_RECODED |

The original field is removed from the dataset, and a new field is added to the dataset.

13. Click **Preview**, and then scroll all the way to the right in the **Preview** output window.
The results appear similar to the following:



| | age | tariff | dropped_calls | bill_peak | bill_offpeak | churn | handset_RECODED |
|---|-------|--------|---------------|-----------|--------------|---------|-----------------|
| 1 | 46... | CA... | 1 | 5.420 | 0.399 | Churned | 10 |
| 2 | 27... | CA... | 0 | 5.916 | 1.058 | Churned | 10 |
| 3 | 39... | CA... | 2 | 10.890 | 1.380 | Churned | 11 |
| 4 | 28... | CA... | 2 | 10.890 | 1.380 | Churned | 10 |
| 5 | 47... | CA... | 0 | 6.091 | 0.941 | Churned | 10 |
| 6 | 29 | CA | 1 | 6.939 | 0.554 | Churned | 10 |

The new field is numeric, but because the script assigned value labels to the values in the upstream Type node, you can also display the handsets.

14. Click the **Display field and value labels**  button.

15. Scroll all the way to the right in the Table output window to locate **handset_RECODED**, if necessary.

The results appear as follows:

| Table | Annotations | | | | | | |
|-------|-------------|--------|---------------|-----------|--------------|---------|-----------------|
| | age | tariff | dropped_calls | bill_peak | bill_offpeak | churn | handset_RECODED |
| 1 | 46.... | CA... | 1 | 5.420 | 0.399 | Churned | SOP10 |
| 2 | 27.... | CA... | 0 | 5.916 | 1.058 | Churned | SOP10 |
| 3 | 39.... | CA... | 2 | 10.890 | 1.380 | Churned | SOP20 |
| 4 | 28.... | CA... | 2 | 10.890 | 1.380 | Churned | SOP10 |
| 5 | 47.... | CA... | 0 | 6.091 | 0.941 | Churned | SOP10 |
| 6 | 29.... | CA... | 1 | 6.939 | 0.554 | Churned | SOP10 |
| 7 | 38.... | CA... | 1 | 8.456 | 0.201 | Churned | SOP20 |
| 8 | 27.... | CA... | 0 | 7.656 | 1.740 | Churned | SOP20 |
| 9 | 38.... | CA... | 1 | 14.850 | 2.655 | Churned | SOP20 |
| 10 | 48.... | CA... | 0 | 9.810 | 1.470 | Churned | SOP20 |

The value labels rather than values are displayed.

16. Click **OK** to close the **Table** output window.
17. From the **File** menu, click **Close Stream**; click **No** if asked to save your changes.
This completes the workshops. You can review the scripts that you think could be useful in your work. When done, please exit IBM SPSS Modeler.
18. From the **File** menu, click **Exit**; click **No** if asked to save your changes.

We Value Your Feedback!

- Don't forget to submit your Think 2018 session and speaker feedback! Your feedback is very important to us – we use it to continually improve the conference.
- Access the Think 2018 agenda tool to quickly submit your surveys from your smartphone, laptop or conference kiosk.