



INTERNAL

SAP Data Intelligence hands-on exercises

This document will guide you step-by-step through the process of training and implementing association rules to produce recommendation analysis using SAP HANA ML in SAP DI.

www.sap.com/contactsap

© 2018 SAP SE or an SAP affiliate company. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company.

The information contained herein may be changed without prior notice. Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors. National product specifications may vary.

These materials are provided by SAP SE or an SAP affiliate company for informational purposes only, without representation or warranty of any kind, and SAP or its affiliated companies shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP or SAP affiliate company products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

In particular, SAP SE or its affiliated companies have no obligation to pursue any course of business outlined in this document or any related presentation, or to develop or release any functionality mentioned therein. This document, or any related presentation, and SAP SE's or its affiliated companies' strategy and possible future developments, products, and/or platform directions and functionality are all subject to change and may be changed by SAP SE or its affiliated companies at any time for any reason without notice. The information in this document is not a commitment, promise, or legal obligation to deliver any material, code, or functionality. All forward-looking statements are subject to various risks and uncertainties that could cause actual results to differ materially from expectations. Readers are cautioned not to place undue reliance on these forward-looking statements, and they should not be relied upon in making purchasing decisions.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. All other product and service names mentioned are the trademarks of their respective companies. See www.sap.com/copyright for additional trademark information and notices.

Table of Contents

| | |
|--|----|
| DISCLAIMER | 4 |
| OBJECTIVE | 4 |
| SCENARIO | 4 |
| ENVIRONMENT ACCESS | 5 |
| STEP 1 – USE A JUPYTER NOTEBOOK..... | 6 |
| STEP 2 – BUILD MODEL PIPELINES | 9 |
| STEP 3 – USE YOUR ASSOCIATION RULES MODEL | 27 |
| APPENDIX 1 – INTRODUCTION TO ASSOCIATION RULES | 29 |
| APPENDIX 2 – APRIORI IN SAP HANA ML | 30 |

DISCLAIMER

The information shared in this document is confidential and proprietary to SAP and may not be disclosed without the permission of SAP. All functionality presented here is subject to change and may be changed by SAP at any time for any reason without notice.

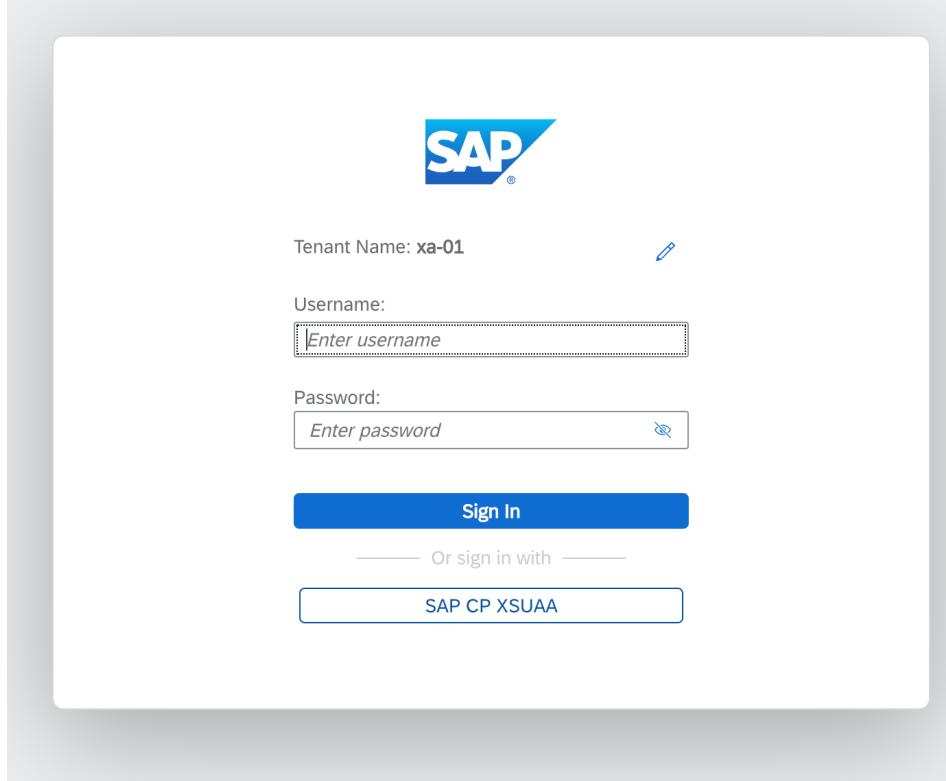
OBJECTIVE

The objective of this exercise is to give you an overview of how you can use the machine learning capabilities in SAP Data Intelligence. We will use the association rules APRIORI algorithm available in the SAP HANA ML PAL library.

SCENARIO

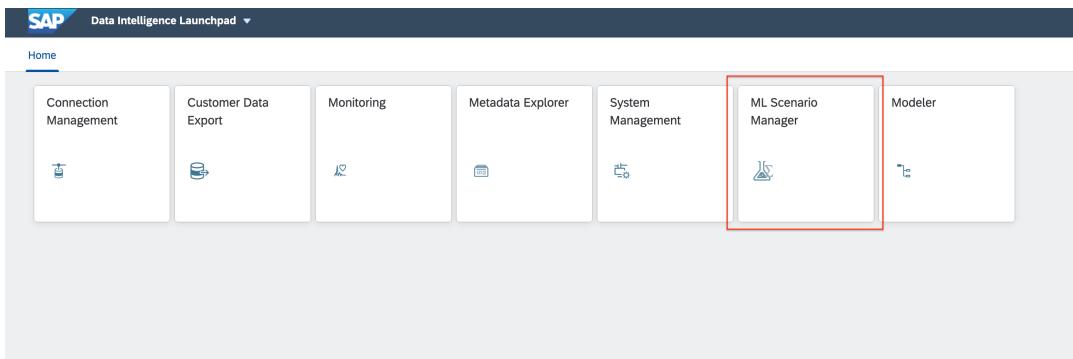
In this scenario, we want to build a book recommendation algorithm, using as input data source the history of book sales for a fictitious book shop having about 10 years of activity (2020 to 2021). The sales data are stored in a Data Warehouse Cloud instance. We will perform market-basket analysis on the historical combinations of books purchased, and build association rules that will be used to make book recommendations.

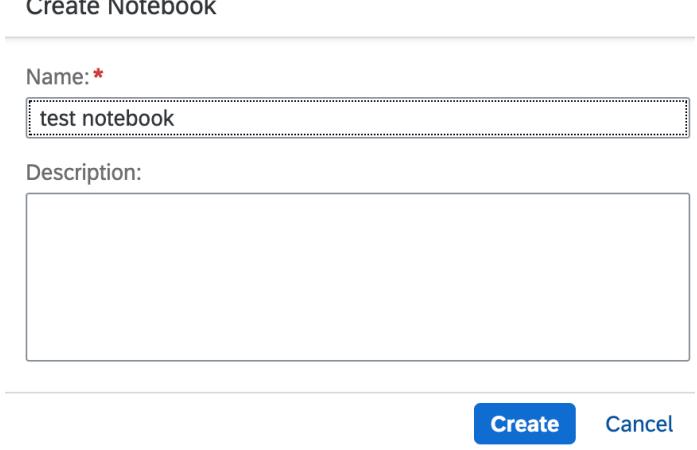
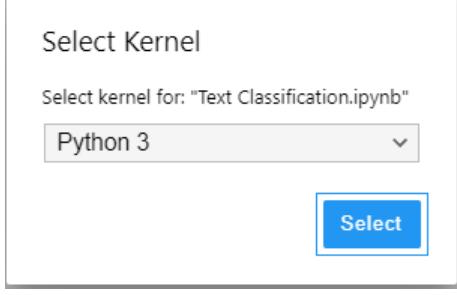
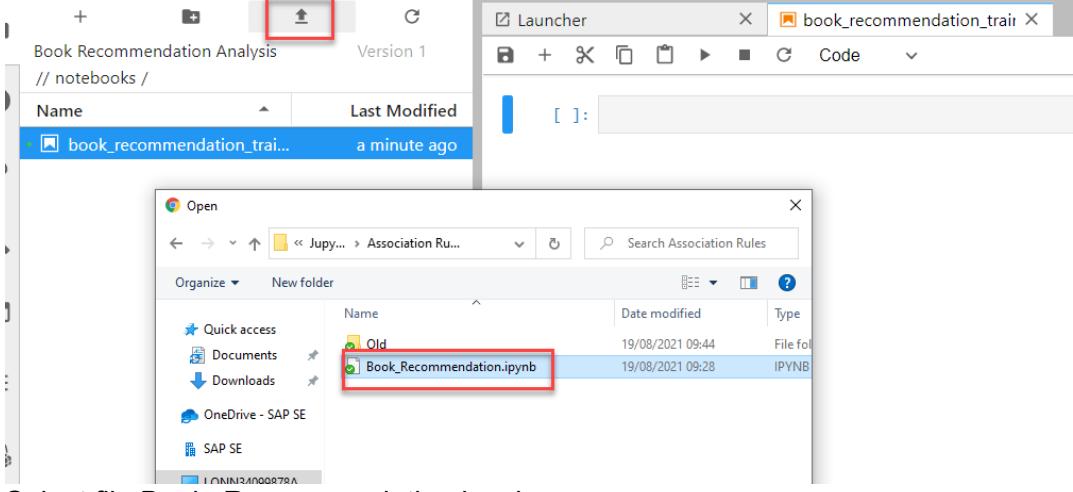
ENVIRONMENT ACCESS

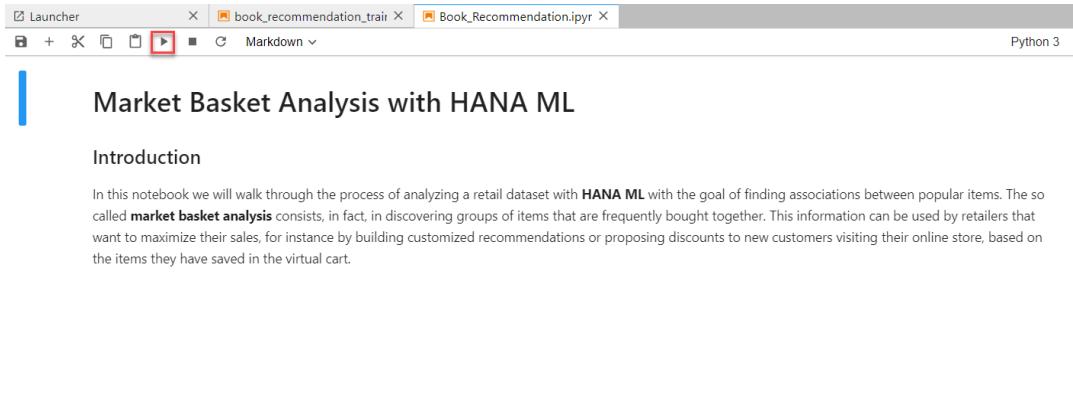
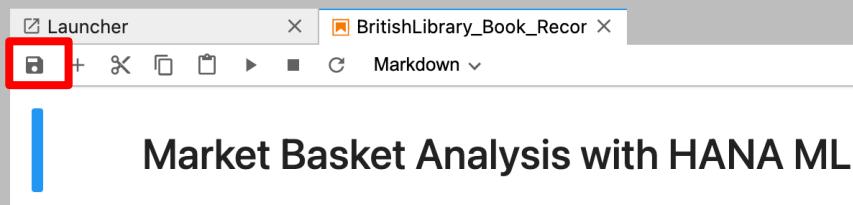
| Explanation | Screenshot |
|--|--|
| <p>Login to your SAP Data Intelligence tenant using the tenant name, username and password assigned to you</p> |  A screenshot of the SAP Data Intelligence login interface. At the top center is the SAP logo. Below it, the text "Tenant Name: xa-01" is displayed with a blue edit icon. The next section is for "Username:" with a placeholder "Enter username" in a text input field. The following section is for "Password:" with a placeholder "Enter password" in a text input field that includes a visibility toggle icon. A large blue "Sign In" button is centered below the input fields. Below the sign-in area, a horizontal line with the text "Or sign in with" is followed by a blue rectangular button labeled "SAP CP XSUAA". |

STEP 1 – USE A JUPYTER NOTEBOOK

A Jupyter Notebook environment is used to explore the data, and to test an association algorithm on the book sales dataset.

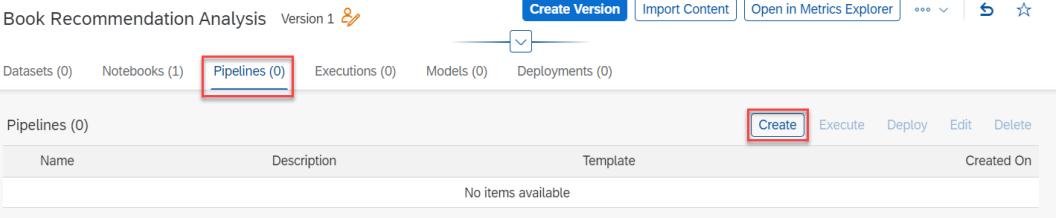
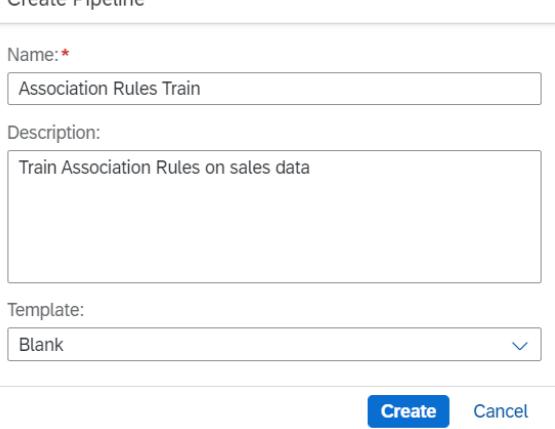
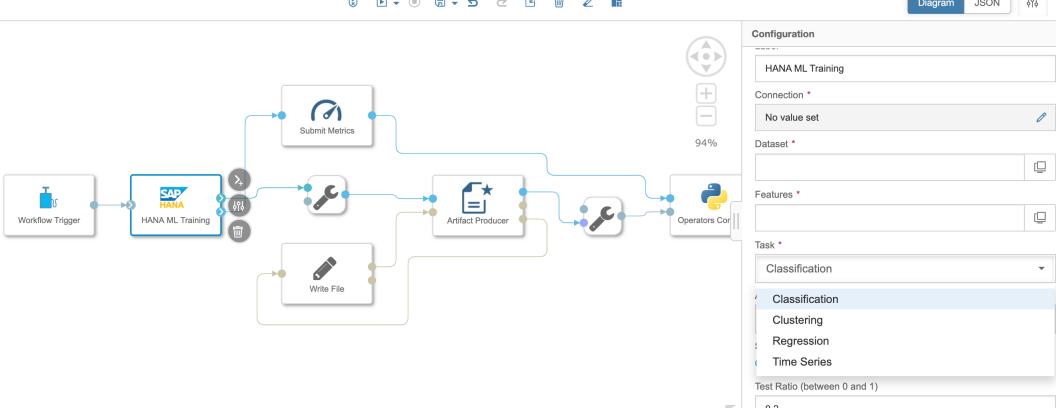
| Explanation | Screenshot | | | | | | | | |
|---|--|------------|-------------|------------|------------|--------------------|--|--|--|
| Click to open ML Scenario Manager |  | | | | | | | | |
| Click the Create button. Create a new scenario. Name the scenario according to your username: "D2VUXXXX Book Recommendation". | <p>Create Scenario</p> <p>Name: * D2V0001-Book Recommendation</p> <p>Business Question:</p> <p>Create Cancel</p> | | | | | | | | |
| In the Notebooks section, click Create to create a new notebook. | <p>Notebooks (0)</p> <table border="1"><thead><tr><th>Name</th><th>Description</th><th>Created On</th><th>Changed On</th></tr></thead><tbody><tr><td colspan="4">No items available</td></tr></tbody></table> <p>Create Edit Delete</p> | Name | Description | Created On | Changed On | No items available | | | |
| Name | Description | Created On | Changed On | | | | | | |
| No items available | | | | | | | | | |

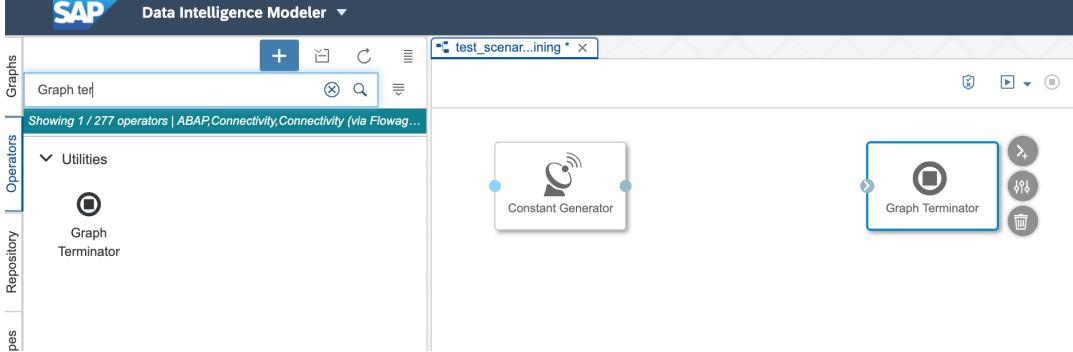
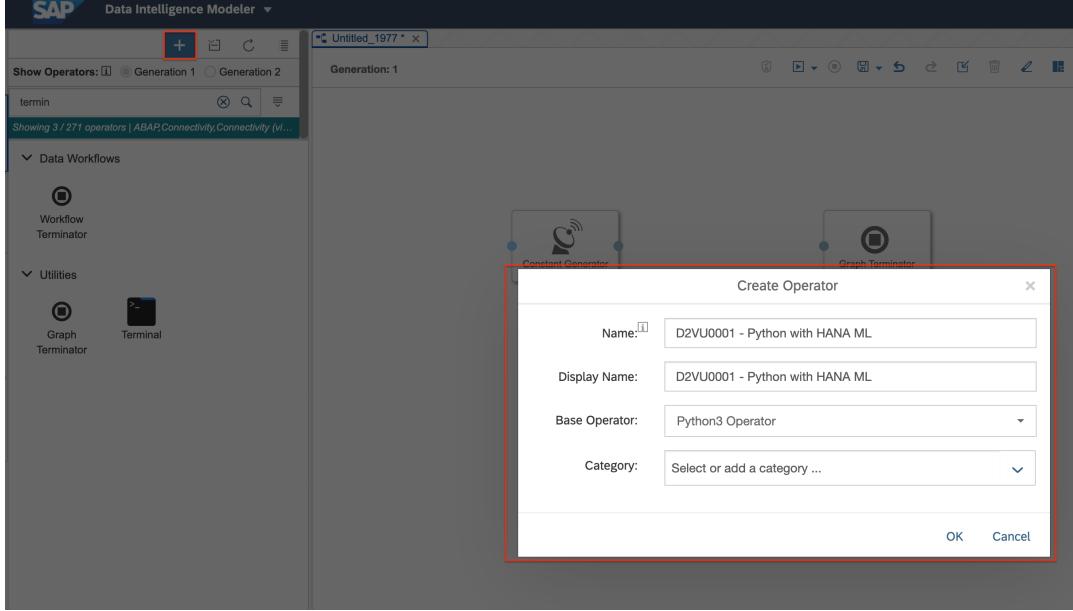
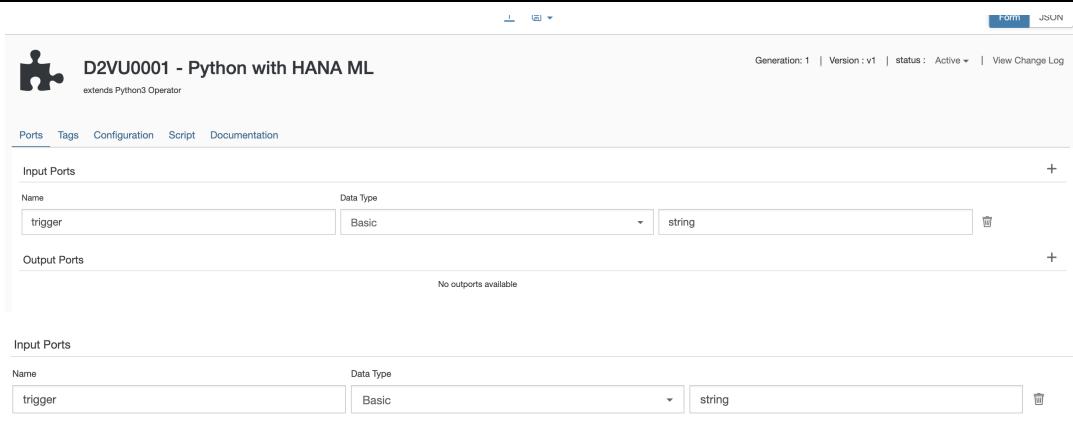
| Explanation | Screenshot |
|---|--|
| Name the notebook "test notebook" |  <p>Create Notebook</p> <p>Name:*</p> <p>test notebook</p> <p>Description:</p> <p>Create Cancel</p> |
| Select the Python 3 Kernel option. |  <p>Select Kernel</p> <p>Select kernel for: "Text Classification.ipynb"</p> <p>Python 3</p> <p>Select</p> |
| <p>Use the Upload Files function at the top left of the console to upload the Python notebook that we prepared for the exercise.</p> <p>Upload file Book_Recommendation.ipynb : you will find it in the course github repository.</p> <p>Double click on the notebook that is uploaded.</p> |  <p>Book Recommendation Analysis Version 1</p> <p>book_recommendation_train.ipynb</p> <p>Open</p> <p>Old</p> <p>Book_Recommendation.ipynb</p> <p>Select file Book_Recommendation.ipynb</p> |

| Explanation | Screenshot |
|--|--|
| <p>Step through the code to check it works correctly.</p> <p>Highlight each cell in sequence and click the arrow button to run the selected cell and advance.</p> <p>Analyze the results of your data analysis and understand the association rules.</p> |  |
| <p>Pay attention at cell 2 and 3, you will need to insert the correct name of your DWC connection, DWC schema and book sales order table</p> | <p>To proceed with our analysis, we need to create a HANA dataframe containing input data. These are different from a pandas dataframe, that reside and consume resources in the client domain (our jupyter notebook), a HANA dataframe exists only in memory in HANA. Therefore, we can access it as long as a connection with HANA is established. In the cell below, we create a connection to the HANA layer of the SAP Data Warehouse Cloud Tenant where the bookshop sales orders are stored. Please replace the connection id with the one related to your account.</p> <pre data-bbox="425 798 1498 846"><code>import hana_ml.dataframe as dataframe from notebook_hana_connector.notebook_hana_connector import NotebookConnectionContext conn = NotebookConnectionContext(connectionId = 'DWC_D2VUXXXX')</code></pre> <p>Now that the connection is set, we can access the sales order table in the form of a hana dataframe, and start to prepare the data for the association analysis. Please edit the schema input variable below according to your own DWC space.</p> <pre data-bbox="425 903 1498 952"><code>df_hana = (conn.table('All_BL_Sales_Order_Data', schema='D2VUXXXX')) df_hana.head(20).collect()</code></pre> |
| <p>Once you have stepped through to the end of the notebook, and there are no errors, save the notebook.</p> |  |

STEP 2 – BUILD MODEL PIPELINES

Model pipelines are used to operationalize the association rules model. Now that you have prepared the data, identified the best algorithm to use and which hyper-parameters work best, you can operationalize the model. We will build one pipeline to automate model training, and a second pipeline to deploy the book recommendation engine.

| Explanation | Screenshot |
|--|--|
| <p>To create the graphical pipeline to retrain the association rules, go to your ML Scenario's main page, select the "Pipelines" tab and click Create</p> |  <p>The screenshot shows the 'Book Recommendation Analysis' interface with 'Version 1'. The top navigation bar includes 'Create Version', 'Import Content', 'Open in Metrics Explorer', and other options. Below the navigation is a menu bar with 'Datasets (0)', 'Notebooks (1)', 'Pipelines (0)' (which is highlighted with a red box), 'Executions (0)', 'Models (0)', and 'Deployments (0)'. A sub-menu for 'Pipelines' is open, showing 'Pipelines (0)' with columns for 'Name', 'Description', 'Template', and 'Created On'. A 'Create' button is highlighted with a red box at the top right of the sub-menu. Below the sub-menu, a message says 'No items available'.</p> |
| <p>Name the pipeline "Association Rules Train" and select the "Blank" template to create a bespoke pipeline graph. Click Create.</p> |  <p>The screenshot shows the 'Create Pipeline' dialog. It has fields for 'Name:' (containing 'Association Rules Train') and 'Description:' (containing 'Train Association Rules on sales data'). A 'Template:' dropdown is set to 'Blank'. At the bottom are 'Create' and 'Cancel' buttons.</p> |
| <p>Notice that other non-blank templates are available in the menu. For instance, the HANA ML Training template (on the right), is very useful for the most common machine learning scenarios. Association rules are not covered by the HANA ML training operator, that's why we will need a customized pipeline.</p> |  <p>The screenshot shows a graphical pipeline editor. On the left, there is a 'Workflow Trigger' node connected to a 'HANA ML Training' node. The 'HANA ML Training' node has two outgoing arrows: one to a 'Submit Metrics' node and another to an 'Artifact Producer' node. The 'Submit Metrics' node connects to a 'Write File' node. The 'Artifact Producer' node connects to an 'Operators Config' node. On the right side of the editor, there is a 'Configuration' panel with tabs for 'Diagram' (selected) and 'JSON'. The configuration panel includes fields for 'Connection', 'Dataset', 'Features', 'Task' (set to 'Classification'), and a 'Test Ratio (between 0 and 1)' field set to '0.2'.</p> |

| Explanation | Screenshot |
|---|--|
| <p>In the blank pipeline, we'll start by adding two operators:</p> <ol style="list-style-type: none"> 1. Constant Generator 2. Graph Terminator <p>Select each, and double click or drag and drop to add to console. These will be the start and the end of our pipeline.</p> |  |
| <p>Then, we will build a custom operator to run association rules in Hana ML.</p> <p>Click on the plus icon at top left of the screen and complete the dialog box as in the picture. Include your username in the operator's name.</p> |  |
| <p>Create an input port like in the picture: we want the operator to start running as soon as it gets a message from the constant generator operator.</p> |  |

Go to the **Configuration** tab.
 We will configure the operator so that it can **connect to a HANA instance connected to SAP Data Intelligence**.
 Click the pencil to get into the edit mode and change to JSON.

In the editor, copy-paste the text on the right and click ok.

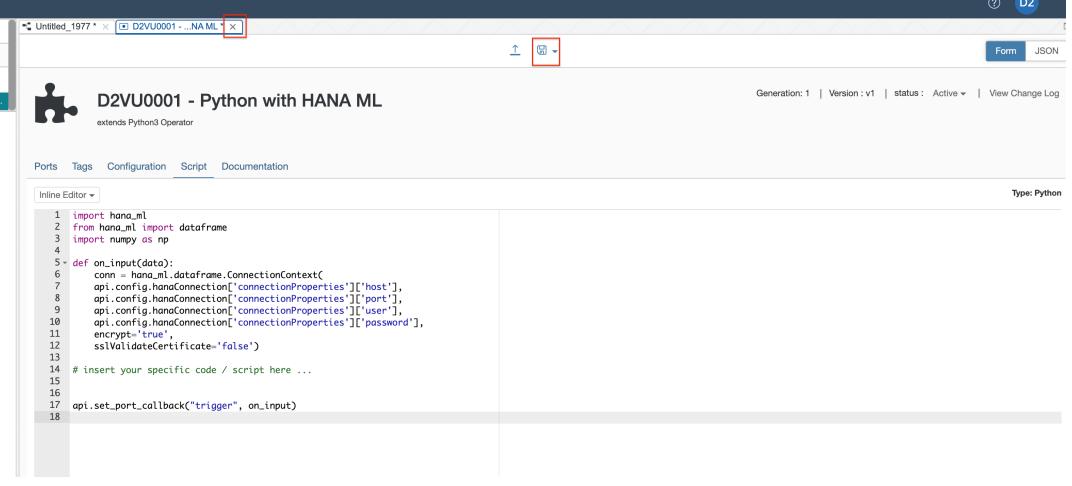
At the bottom of the modal are 'OK' and 'Cancel' buttons."/>

```

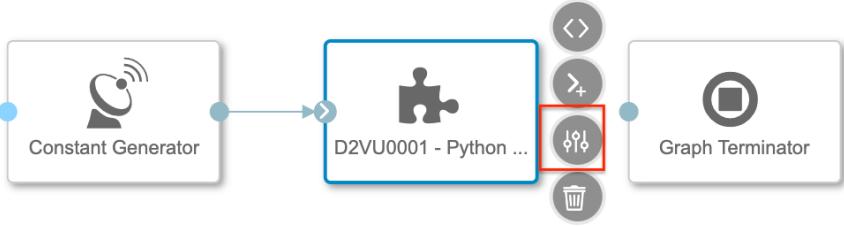
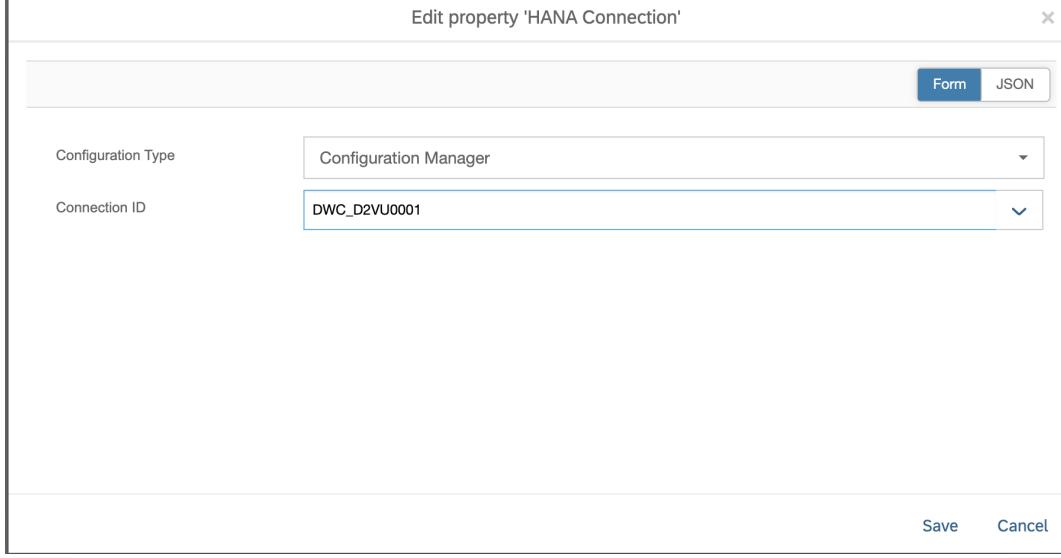
{
    "$schema": "http://json-schema.org/draft-06/schema#",
    "$id": "http://sap.com/vflow/HANA ML BLOG.configSchema.json",
    "type": "object",
    "properties": {
        "hanaConnection": {
            "title": "HANA Connection",
            "description": "HANA Connection",
            "type": "object",
            "properties": {
                "configurationType": {
                    "title": "Configuration Type",
                    "type": "string",
                    "enum": [
                        "",
                        "Configuration Manager",
                        "Manual"
                    ]
                },
                "connectionID": {
                    "title": "Connection ID",
                    "type": "string",
                    "format": "com.sap.dh.connection.id",
                    "sap_vflow_valuehelp": {
                        "url": "/app/datahub-app-
connection/connections?connectionTypes=HANA_DB",
                        "valuepath": "id",
                        "displayStyle": "autocomplete"
                    },
                    "sap_vflow_constraints": {
                        "ui_visibility": [
                            {
                                "name": "name",
                                "value": "Configuration Manager"
                            }
                        ]
                    }
                },
                "connectionProperties": {
                    "title": "Connection Properties",
                    "$ref": "http://sap.com/vflow/com.sap.dh.connections.hana_db.schema.json",
                    "sap_vflow_constraints": {
                        "ui_visibility": [
                            {
                                "name": "name",
                                "value": "Manual"
                            }
                        ]
                    }
                },
                "required": []
            },
            "codelanguage": {
                "type": "string"
            },
            "script": {
                "type": "string"
            }
        }
    }
}
    
```

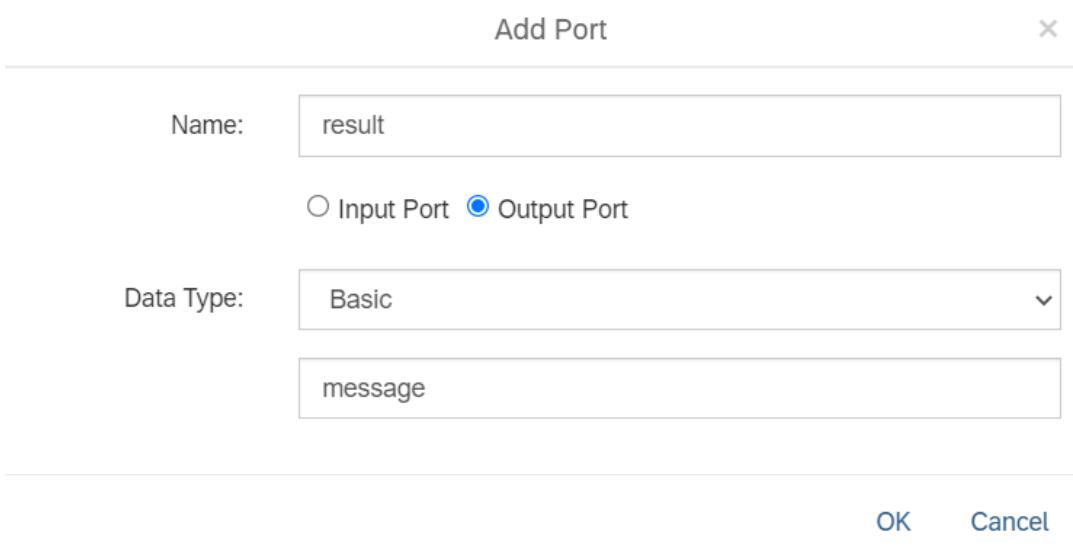
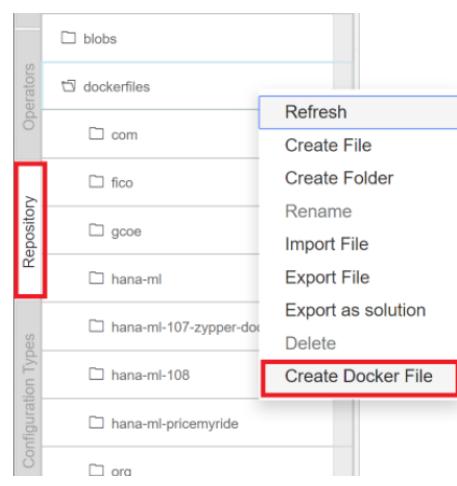
| | |
|--|--|
| | <pre> }, "required": [] } </pre> |
|--|--|

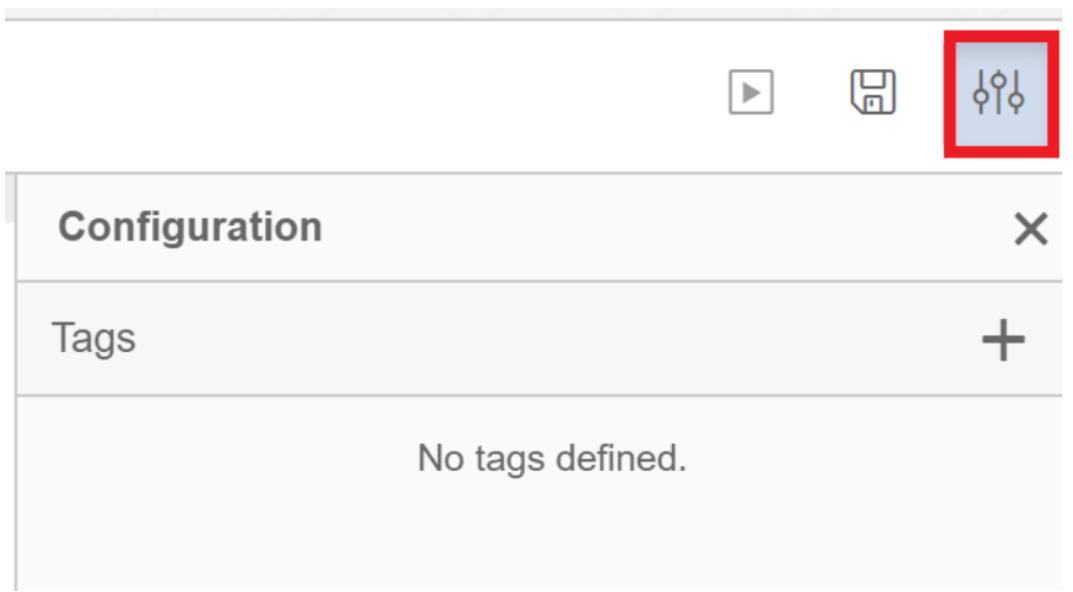
| | |
|---|--|
| <p>Go to the <i>Script</i> tab and copy paste the python code to the right.</p> | <pre> import hana_ml from hana_ml import dataframe import numpy as np def on_input(data): conn = hana_ml.dataframe.ConnectionContext(api.config.hanaConnection['connectionProperties']['host'], api.config.hanaConnection['connectionProperties']['port'], api.config.hanaConnection['connectionProperties']['user'], api.config.hanaConnection['connectionProperties']['password'], encrypt='true', sslValidateCertificate='false') # insert your specific code / script here ... api.set_port_callback("trigger", on_input) </pre> |
|---|--|

| | |
|---|---|
| <p>Save the custom operator and the close the window.</p> |  |
|---|---|

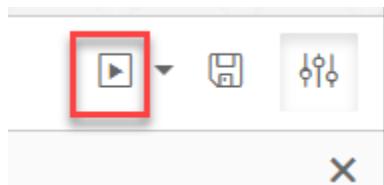
| | |
|---|--|
| <p>You should be now again in the pipeline window. Look for your new custom operator in the menu and add it to the pipeline. Connect the constant Generator with the trigger port of the custom operator.</p> |  |
|---|--|

| | |
|---|--|
| <p>Begin with configuring the HANA ML within Python operator. Highlight it and click on the Configuration icon.</p> |  |
| <p>Edit the Hana Connection field. Select Configuration Manager as Configuration Type and select your DWC connection in the Connection ID drop-down menu. Click save.</p> |  |
| <p>In the HANA ML within Python operator, click the script icon.</p>  <p>Customize the python template by adding the code to train the association algorithm. The complete script is available here to the right. Replace the schema value with your assigned user id for the exercise. Notice that python is</p> | <pre>import hana_ml from hana_ml import dataframe import numpy as np def on_input(data): conn = hana_ml.dataframe.ConnectionContext(api.config.hanaConnection['connectionProperties'][['host'], api.config.hanaConnection['connectionProperties'][['port'], api.config.hanaConnection['connectionProperties'][['user'], api.config.hanaConnection['connectionProperties'][['password'], encrypt='true', sslValidateCertificate='false') # insert your specific code / script here ... df_hana = (conn.table('All_BL_Sales_Order_Data', schema='<XXXXXXXXXX>')) df_hana=df_hana.select('Order_ID','Book_ID') from hana_ml.algorithms.pal.association import Apriori min_support=0.0005 min_confidence=0.05 min_lift=5 ap = Apriori(min_support=min_support, min_confidence=min_confidence, max_len = 2, min_lift=min_lift)</pre> |

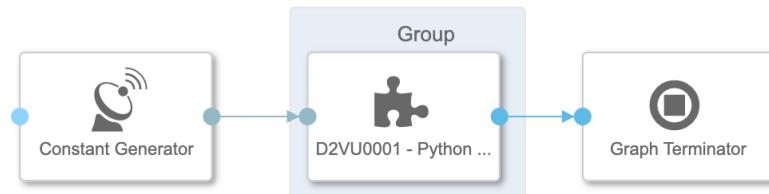
| | |
|--|--|
| <p>sensitive to indentation!</p> | <pre>) ap.fit(data=df_hana) ap.result_.save(where='APRIORI_BOOK_ASSOCIATION_PIPELINE_TRAINING',force=True) # output some quality metrics result = {"number_of_associations": str(ap.result_.shape[0])} api.send("result", api.Message(result)) api.set_port_callback("trigger", on_input) </pre> |
| <p>Return to your pipeline. Right-click on the custom operator and select Add Port. Add an output port. Enter the information here and then click OK.</p> |  <p>Add Port</p> <p>Name: result</p> <p><input type="radio"/> Input Port <input checked="" type="radio"/> Output Port</p> <p>Data Type: Basic</p> <p>message</p> <p>OK Cancel</p> |
| <p>Link the result port of the custom operator to the Graph Terminator.</p> |  <pre> graph LR CG[Constant Generator] --> D2VU[D2VU0001 - Python ...] D2VU --> GT[Graph Terminator] </pre> |
| <p>You need to create a Docker image for the Python operator. This allows you to specify the python libraries that your operator will use at runtime. You find the docker files by clicking into the "Repository" tab on the left, then right-click the "dockerfiles" folder</p> |  <p>Operators</p> <ul style="list-style-type: none"> blobs dockerfiles com fico gcoe hana-ml hana-ml-107-zypper-dox hana-ml-108 hana-ml-pricemryide ora <p>Repository</p> <p>Configuration Types</p> <p>Refresh</p> <p>Create File</p> <p>Create Folder</p> <p>Rename</p> <p>Import File</p> <p>Export File</p> <p>Export as solution</p> <p>Delete</p> <p>Create Docker File</p> |

| | | | |
|---|---|----------------------------|---|
| and select “Create Docker File”. | | | |
| Name the file according to your username D2VUxxxxBookRecommendation | <p style="text-align: right;">Create Docker File X</p> <p>Name: <input type="text" value="DVU0001BookRecommendation"/></p> <p style="text-align: right;">OK Cancel</p> | | |
| Enter this code into the Docker File window. This code leverages a base image that comes with SAP Data Intelligence and installs the necessary libraries on it. | <pre>FROM \$com.sap.sles.base RUN pip3.6 install --user numpy==1.16.4 RUN pip3.6 install --user pandas==0.24.0 RUN pip3.6 install --user hana_ml==2.9.21072600</pre> | | |
| Open the Configuration panel for the Docker File with the icon on the top-right hand corner. |  <p>Configuration X</p> <p>Tags +</p> <p>No tags defined.</p> | | |
| Assign a tag to the Dockerfile | <p>Configuration X</p> <p>Tags +</p> <table style="width: 100%; border-collapse: collapse;"> <tr> <td style="border: 1px solid #ccc; padding: 5px; width: 45%;">D2VU0001BookRecommendation</td> <td style="border: 1px solid #ccc; padding: 5px; width: 45%;">latest X</td> </tr> </table> | D2VU0001BookRecommendation | latest X |
| D2VU0001BookRecommendation | latest X | | |

Now save the Docker file and then click the “Build” icon to start building the Docker image. Wait a few minutes and you should receive a confirmation that the build completed successfully.



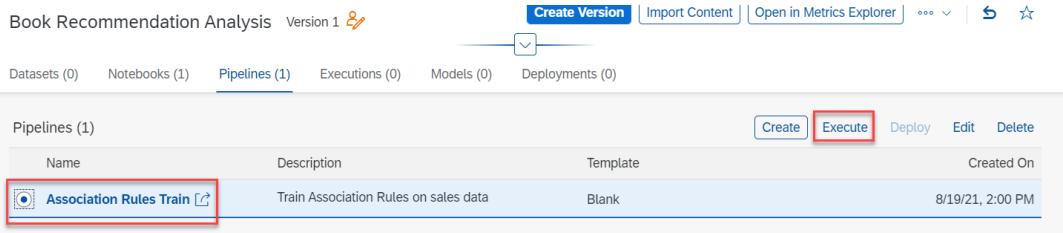
Now you need to configure the custom operator, which trains the model, to run in this Docker image. Click the tab to go back to your graphical pipeline. Right-click the custom operator and select “Group”.



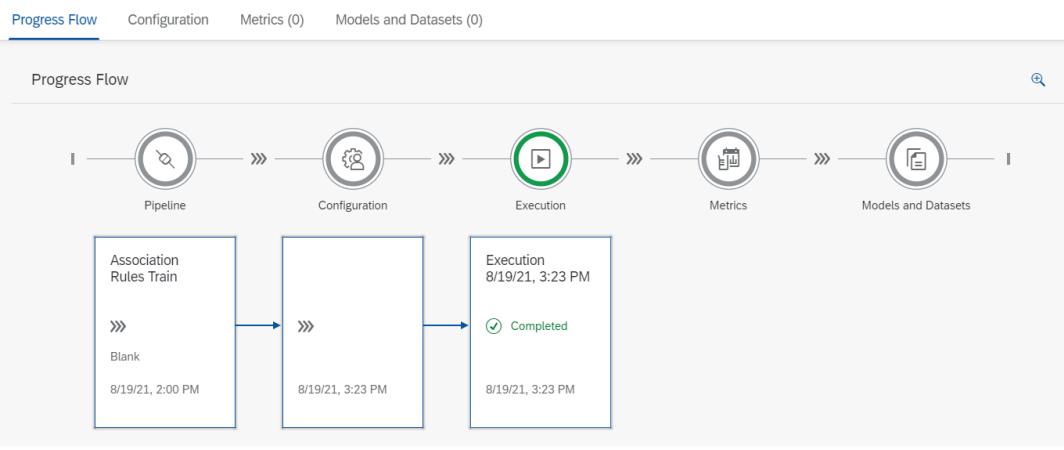
You specify which Docker image should be used. Select the group which surrounds the “Hana ML Within Python” operator. In the group’s Configuration select the docker tag. Save your pipeline.



The pipeline is now complete and you can run it. Go back to the ML Scenario. Select the pipeline in the ML Scenario and click the “Execute” button on the right.



Wait a few seconds until the pipeline executes and completes.



You will now use the model for real-time inference with REST-API.

Go back to the main page of your ML Scenario and create a second pipeline. This pipeline will provide the REST-API to obtain predictions in real-time.

Select the template "Python Consumer" to create a bespoke pipeline. Click Create.

The screenshot shows a 'Create Pipeline' dialog box. It contains fields for Name (set to 'Book Association Consumer'), Description (empty), Template (set to 'Python Consumer'), and buttons for 'Create' and 'Cancel'. Below the dialog, a 'Progress Flow' bar is visible.

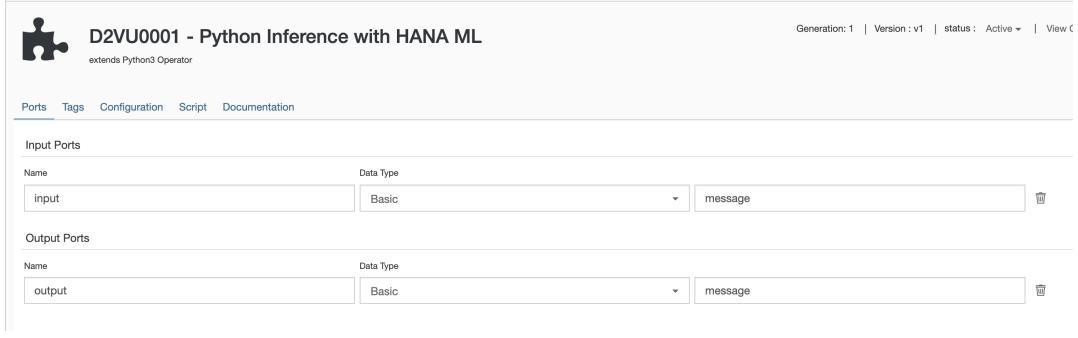
Take a moment to understand this template. The first portion of the graph, shown in red in the picture, has the function of reading a model artifact stored in a binary file and feeding it to the second portion of the graph. As a matter of fact, in machine learning scenarios, trained models such as regressors or



classifiers, are usually stored in binary format.

In our scenario, however, the results of Apriori are not stored in an artifact, but in a HANA table. We can delete the operators within the red box, we will not need them.

The blue portion of the graph, instead, is what we need to expose the book association table to an open API service. The OpenAPI Servlow operator reads requests coming from the openAPI and submits them to the python operator. These requests contains the ID of books that are about to be chosen by a customer. The role of the python operator will be querying the book association table to come up with a list of recommendations that will be sent back to the open API service.

| <p>Since we need to query a table stored in HANA, a simple python operator will not suffice. We will need a custom operator, similar to the one we built for the training pipeline, with the possibility to configure a HANA connection. Click on the plus button</p>  <p>to create a new operator. Configure the dialog box as shown here. Use your own username to name the operator.</p> | <p>Create Operator</p> <p>Name: <input type="text" value="D2VU0001 - Python Inference with HANA ML"/></p> <p>Display Name: <input type="text" value="D2VU0001 - Python Inference with HANA ML"/></p> <p>Base Operator: <input type="text" value="Python3 Operator"/></p> <p>Category: <input type="text" value="Select or add a category ..."/></p> <p>OK Cancel</p> | | | | | | | | | | | | |
|--|--|--------------------------------------|-----------|---------|-------|-------|--------------------------------------|------|-----------|---------|--------|-------|--------------------------------------|
| <p>Add two ports as shown in the picture</p> |  <p>D2VU0001 - Python Inference with HANA ML extends Python3 Operator</p> <p>Generation: 1 Version : v1 status : Active View C</p> <p>Ports Tags Configuration Script Documentation</p> <p>Input Ports</p> <table border="1"> <thead> <tr> <th>Name</th> <th>Data Type</th> <th>Message</th> </tr> </thead> <tbody> <tr> <td>input</td> <td>Basic</td> <td><input type="text" value="message"/></td> </tr> </tbody> </table> <p>Output Ports</p> <table border="1"> <thead> <tr> <th>Name</th> <th>Data Type</th> <th>Message</th> </tr> </thead> <tbody> <tr> <td>output</td> <td>Basic</td> <td><input type="text" value="message"/></td> </tr> </tbody> </table> | Name | Data Type | Message | input | Basic | <input type="text" value="message"/> | Name | Data Type | Message | output | Basic | <input type="text" value="message"/> |
| Name | Data Type | Message | | | | | | | | | | | |
| input | Basic | <input type="text" value="message"/> | | | | | | | | | | | |
| Name | Data Type | Message | | | | | | | | | | | |
| output | Basic | <input type="text" value="message"/> | | | | | | | | | | | |
| <p>Go to the Configuration tab and repeat the same steps you did for the training custom operator.</p> | | | | | | | | | | | | | |
| <p>In the Script session, enter the piece of code here.</p> <p>Save the operator and go back to your blank pipeline</p> | <pre>import json import hana_ml from hana_ml import dataframe # Global vars to keep track of model status model = None model_ready = False # Validate input data is JSON def is_json(data):</pre> | | | | | | | | | | | | |

```

try:
    json_object = json.loads(data)
except ValueError as e:
    return False
return True

# Load the Hana ML model
#def on_trigger(data):

    conn = hana_ml.dataframe.ConnectionContext(
        api.config.hanaConnection['connectionProperties'][['host'],
        api.config.hanaConnection['connectionProperties'][['port'],
        api.config.hanaConnection['connectionProperties'][['user'],
        api.config.hanaConnection['connectionProperties'][['password'],
        encrypt='true',
        sslValidateCertificate='false')

try:
    # Load your HANA_ML model here
    # model = .....
    model_ready = True
    api.logger.info("Model Received & Ready")
except Exception as e:
    api.logger.error(e)
    error_message = "An error occurred while loading the model: " + str(e)

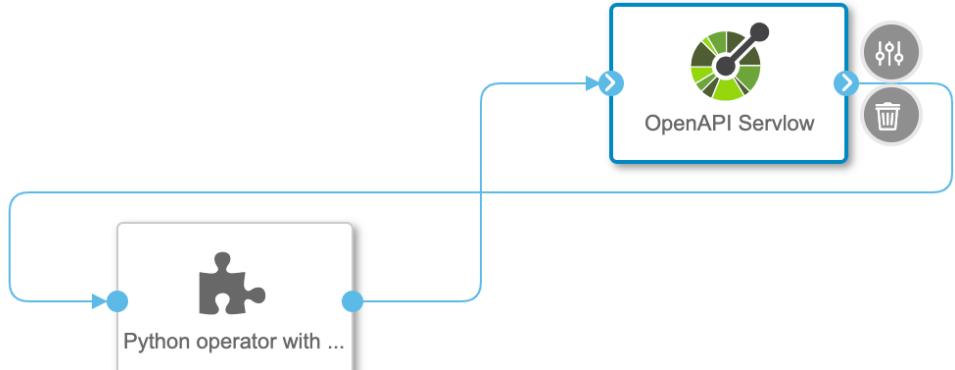
# Client POST request received
def on_input(msg):
    error_message = ""
    success = False
    prediction = None
    try:
        api.logger.info("POST request received from Client - checking if model is ready")
        if model_ready:
            api.logger.info("Model Ready")
            api.logger.info("Received data from client - validating json input")

            user_data = msg.body.decode('utf-8')
            # Received message from client, verify json data is valid
            if is_json(user_data):
                api.logger.info("Received valid json data from client - ready to use")

                # apply your model
                # obtain your results
                input_field = json.loads(user_data)[["input"]]
                #prediction = .....apply model here ...

                success = True
            else:
                api.logger.info("Invalid JSON received from client - cannot apply model.")
                error_message = "Invalid JSON provided in request: " + user_data
                success = False
        else:
            api.logger.info("Model has not yet reached the input port - try again.")
            error_message = "Model has not yet reached the input port - try again."
            success = False
    except:
        error_message = "An error occurred while processing the client request: " + str(e)
        success = False

```

| | |
|--|--|
| | <pre> except Exception as e: api.logger.error(e) error_message = "An error occurred: " + str(e) if success: # apply carried out successfully, send a response to the user msg.body = json.dumps({'prediction': prediction}) else: msg.body = json.dumps({'Error': error_message}) new_attributes = {'message.request.id': msg.attributes['message.request.id']} msg.attributes = new_attributes api.send('output', msg) api.set_port_callback("input", on_input) </pre> |
| Replace the Python Inference operator with the custom operator you just built. |  |
| Open the configuration menu for the custom operator and select your DWC connection as already done for the training pipeline | |
| In the Hana ML Inferencing within Python operator, click the “Script” icon. We need to complete the template by adding the lines highlighted on the right. Replace the schema value with your assigned user Id. | <pre> import json import hana_ml from hana_ml import dataframe import pandas as pd # Global vars to keep track of model status model = None model_ready = False # Validate input data is JSON def is_json(data): try: json_object = json.loads(data) except ValueError as e: </pre> |

Close the editor window.

```
return False
return True

# Load the Hana ML model
#def on_trigger(data):

    conn = hana_ml.dataframe.ConnectionContext(
        api.config.hanaConnection['connectionProperties']['host'],
        api.config.hanaConnection['connectionProperties']['port'],
        api.config.hanaConnection['connectionProperties']['user'],
        api.config.hanaConnection['connectionProperties']['password'],
        encrypt='true',
        sslValidateCertificate='false')

    try:
        # Load your HANA_ML model here
        model = (conn.table('APRIORI_BOOK_ASSOCIATION_PIPELINE_TRAINING',
schema='<XXXXXXXX#DWUSER>'))
        model_ready = True
        api.logger.info("Model Received & Ready")
    except Exception as e:
        api.logger.error(e)
        error_message = "An error occurred while loading the model: " + str(e)

# Client POST request received
def on_input(msg):
    error_message = ""
    success = False
    prediction = None
    try:
        api.logger.info("POST request received from Client - checking if model is ready")
        if model_ready:
            api.logger.info("Model Ready")
            api.logger.info("Received data from client - validating json input")

            user_data = msg.body.decode('utf-8')
            # Received message from client, verify json data is valid
            if is_json(user_data):
                api.logger.info("Received valid json data from client - ready to use")

                # apply your model
                # obtain your results
                book_ID = json.loads(user_data)["book"]
                filter_string='ANTECEDENT = '+str(book_ID)
                prediction =
model.filter(filter_string).sort('LIFT',desc=True).select('CONSEQUENT').collect().v
alues.tolist()
                if len(prediction) == 0:
                    prediction='No rule available for this book'
    
```

```

        success = True
    else:
        api.logger.info("Invalid JSON received from client - cannot apply
model.")
        error_message = "Invalid JSON provided in request: " + user_data
        success = False
    else:
        api.logger.info("Model has not yet reached the input port - try again.")
        error_message = "Model has not yet reached the input port - try again."
        success = False
except Exception as e:
    api.logger.error(e)
    error_message = "An error occurred: " + str(e)

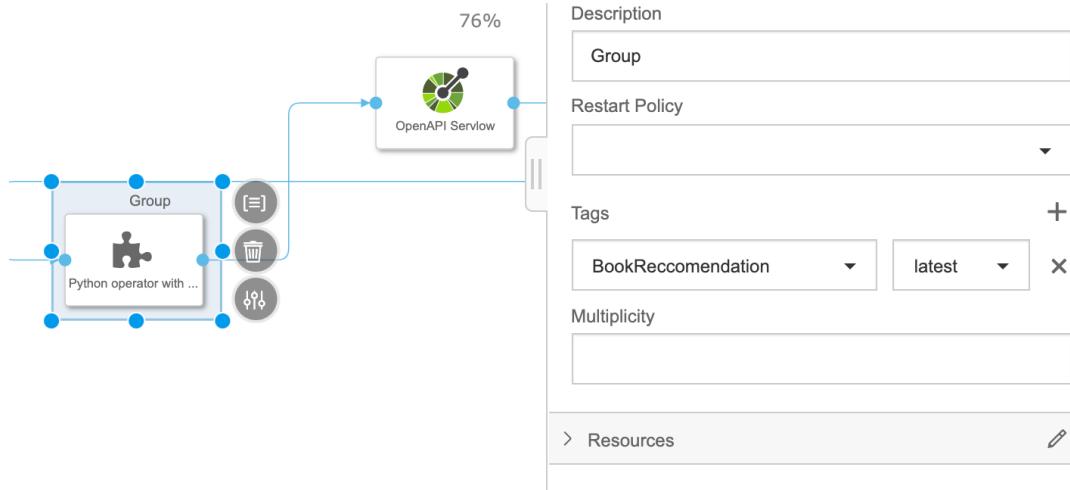
if success:
    # apply carried out successfully, send a response to the user
    msg.body = json.dumps({'recommendation': prediction})
else:
    msg.body = json.dumps({'Error': error_message})

new_attributes = {'message.request.id': msg.attributes['message.request.id']}
msg.attributes = new_attributes
api.send('output', msg)

api.set_port_callback("input", on_input)

```

Assign the Docker image. As before, right-click the custom operator and select “Group”.
Add the tag
Save the changes.



Click on OpenAPIServlow and have a look at the configuration

Configuration X

Properties Validate ↻

Id
openapiservlow1

Label
OpenAPI Servlow

Base Path
\${deployment}

Timeout
300000

One-Way
 True False

Swagger Specification

```
{  
  "schemes": [  
    "http",  
    "https"  
  ],  
  "swagger": "2.0",  
  "info": {  
    "description": "This is an example of using the OpenAPI Servlow to carry out inference with an existing model.",  
    "title": "OpenAPI demo",  
    "termsOfService": "http://www.sap.com/vora/terms/",  
    "contact": {  
      "name": "Apache 2.0",  
      "url": "http://www.apache.org/licenses/LICENSE-2.0"  
    },  
    "license": {  
      "name": "Apache 2.0",  
      "url": "http://www.apache.org/licenses/LICENSE-2.0"  
    }  
  }  
}
```

Websocket
 True False

Max Concurrency
32

Max Accepted
128

Notice in particular the content of the Swagger Specification.

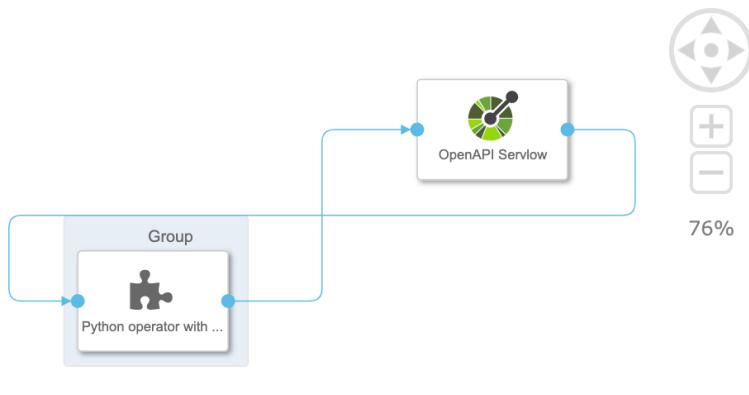
```
{  
  "schemes": [  
    "http",  
    "https"  
  ],  
  "swagger": "2.0",  
  "info": {  
    "description": "This is an example of using the OpenAPI Servlow to carry out inference with an existing model.",  
    "title": "OpenAPI demo",  
    "termsOfService": "http://www.sap.com/vora/terms/",  
    "contact": {  
      "name": "Apache 2.0",  
      "url": "http://www.apache.org/licenses/LICENSE-2.0"  
    },  
    "license": {  
      "name": "Apache 2.0",  
      "url": "http://www.apache.org/licenses/LICENSE-2.0"  
    }  
  }  
}
```

```

        "url":"http://www.apache.org/licenses/LICENSE-2.0.html"
    },
    "version":"1.0.0"
},
"basePath":"/$deployment",
"paths":{
    "/v1/uploadjson": {
        "post": {
            "description": "Upload data in json format",
            "consumes": [
                "application/json"
            ],
            "produces": [
                "application/json"
            ],
            "summary": "Upload JSON data to be used in the Python operator's script",
            "operationId": "upload",
            "parameters": [
                {
                    "type": "object",
                    "description": "json data",
                    "name": "body",
                    "in": "body",
                    "required": true
                }
            ],
            "responses": {
                "200": {
                    "description": "Data uploaded"
                },
                "500": {
                    "description": "Error during upload of json"
                }
            }
        }
    },
    "definitions": {
    },
    "securityDefinitions": {
        "UserSecurity": {
            "type": "basic"
        }
    }
}

```

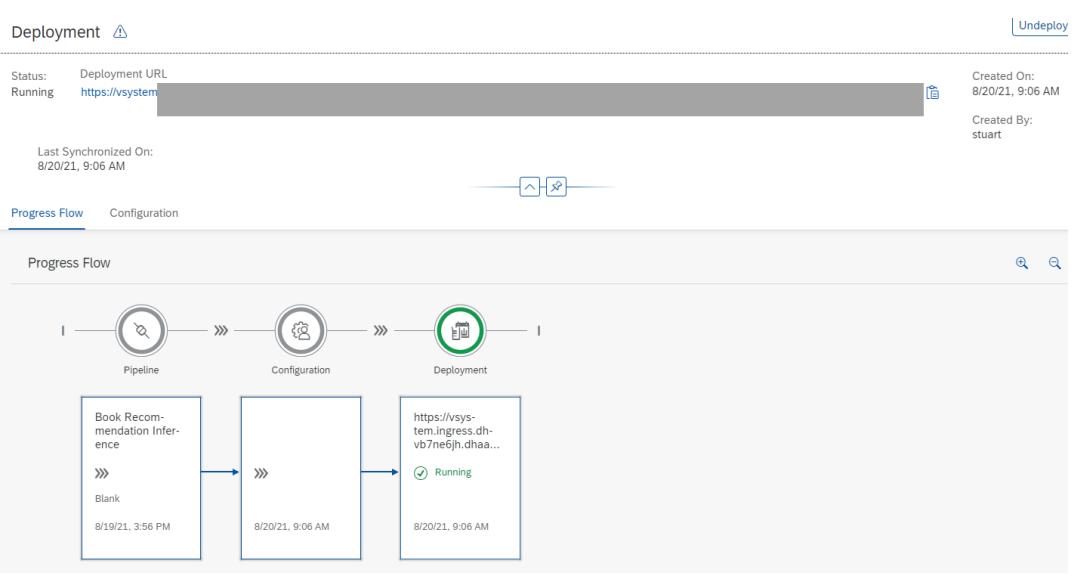
Save changes



Go back to the ML Scenario.
Select your pipeline and click "Deploy".

| Pipelines (4) | | | |
|--|-------------|-----------------|------------------|
| Name | Description | Template | Created On |
| Book Association C... [edit] | | Python Consumer | 10/9/21, 8:52 PM |

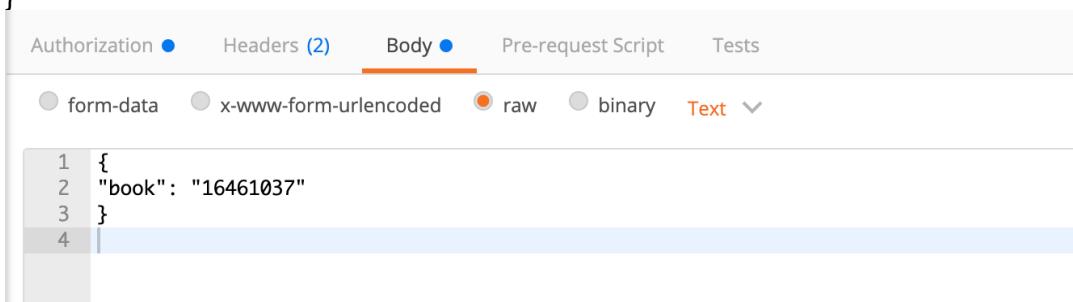
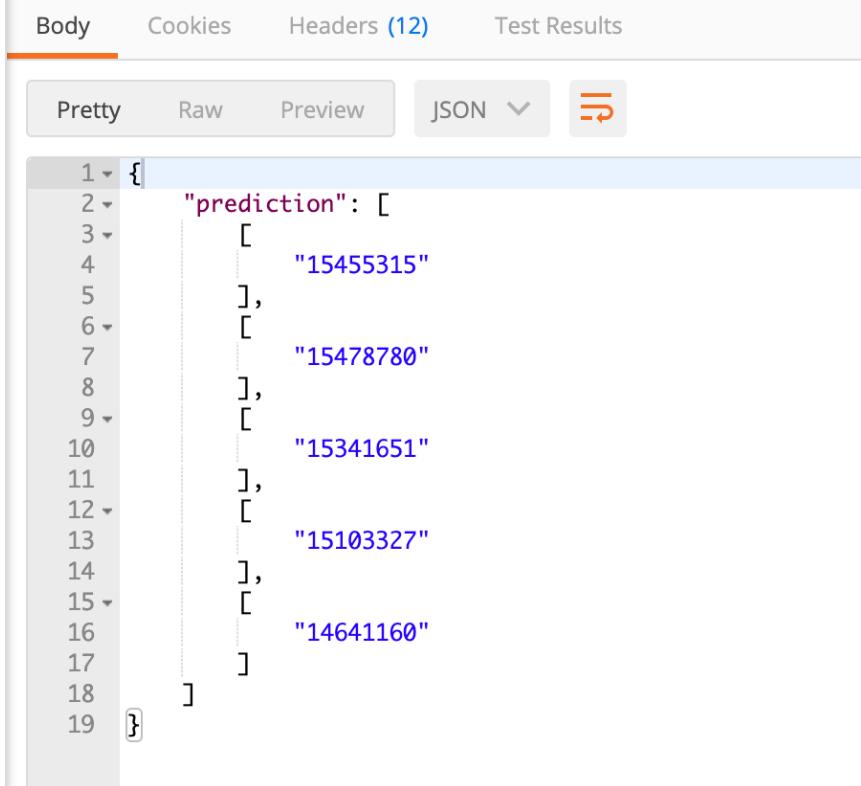
After a few seconds
the pipeline will start
running.



STEP 3 – USE YOUR ASSOCIATION RULES MODEL

Now that you have deployed your model, you can use it for real-time book recommendations. For this, you are going to use the Postman application.

| Explanation | Screenshot |
|---|--|
| <p>Open Postman. Copy the deployment URL from SAP DI. Enter the Deployment URL as request URL. Extend the URL with v1/uploadjson/, the path specified in the OpenAPI servlow operator. Change the request type from “GET” to “POST”.</p> | <p>Copy Deployment URL in SAP DI:</p> <p>To Postman:</p> |
| <p>Go to the “Authorization” tab. Select “Basic Auth” and enter your username and password for SAP Data Intelligence. The username starts with your tenant’s name, followed by a backslash and your actual username.</p> | |
| <p>Go to the “Headers” tab and enter the key “X-Requested-With” with value “XMLHttpRequest”.</p> | |
| <p>Finally, pass the input data to the REST-API. Select the “Body” tab, choose “raw” and enter the syntax given here. Replace <book_ID> with a</p> | <pre>{ "book": <book_ID> }</pre> |

| Explanation | Screenshot |
|--|---|
| <p>book ID such as 16461037. NB! The book_ID must be contained in the list of antecedents, so you can find some valid examples to test from the output you created when you ran the Python code analysis in Jupyter.</p> |  <pre data-bbox="425 270 1498 572">1 { 2 "book": "16461037" 3 } 4</pre> |
| <p>Press “Send” and you will see the book recommendation that comes from SAP Data Intelligence. Try the REST-API with different book IDs to see how the recommendations change.</p> |  <pre data-bbox="425 713 1286 1499">1 [{ 2 "prediction": [3 "15455315" 4], 5 "15478780" 6], 7 "15341651" 8], 9 "15103327" 10], 11 "14641160" 12] 13] 14] 15] 16] 17] 18] 19 }</pre> |
| <p>You have now completed the exercise.</p> | |
| | |

APPENDIX 1 – INTRODUCTION TO ASSOCIATION RULES

For a clearly presented tutorial on the concepts of association rules and the Apriori algorithm we use in this exercise, please see <https://www.kdnuggets.com/2016/04/association-rules-apriori-algorithm-tutorial.html>.

APPENDIX 2 – APRIORI IN SAP HANA ML

Apriori is a classic predictive analysis algorithm for finding association rules used in association analysis. Association analysis uncovers the hidden patterns, correlations or causal structures among a set of items or objects. For example, association analysis enables you to understand what products and services customers tend to purchase at the same time. By analyzing the purchasing trends of your customers with association analysis, you can predict their future behavior.

Apriori is designed to operate on databases containing transactions. As is common in association rule mining, given a set of items, the algorithm attempts to find subsets which are common to at least a minimum number of the item sets. Apriori uses a “bottom up” approach, where frequent subsets are extended one item at a time, a step known as candidate generation, and groups of candidates are tested against the data. The algorithm terminates when no further successful extensions are found. Apriori uses breadth-first search and a tree structure to count candidate item sets efficiently. It generates candidate item sets of length k from item sets of length k-1, and then prunes the candidates which have an infrequent sub pattern. The candidate set contains all frequent k-length item sets. After that, it scans the transaction database to determine frequent item sets among the candidates.

The Apriori function in PAL uses vertical data format to store the transaction data in memory. The function can take VARCHAR/NVARCHAR or INTEGER transaction ID and item ID as input. It supports the output of confidence, support, and lift value, but does not limit the number of output rules.

Prerequisites:

- The input data does not contain null value.
- There are no duplicated items in each transaction

Input Table

| Table | Column | Data Type | Description |
|-------|------------|-------------------------------|----------------|
| DATA | 1st column | INTEGER, VARCHAR, or NVARCHAR | Transaction ID |
| | 2nd column | INTEGER, VARCHAR, or NVARCHAR | Item ID |

Parameter Table

Mandatory Parameters

The following parameters are mandatory and must be given a value.

| Name | Data Type | Description |
|----------------|-----------|---|
| MIN_SUPPORT | DOUBLE | User-specified minimum support (actual value). |
| MIN_CONFIDENCE | DOUBLE | User-specified minimum confidence (actual value). |

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

| Name | Data Type | Default Value | Description |
|--------------------|-----------|------------------|--|
| MIN_LIFT | DOUBLE | 0.0 | User-specified minimum lift. |
| MAX_CONSEQUENT | INTEGER | 100 | Maximum length of dependent items. |
| MAXITEMLENGTH | INTEGER | 5 | Total length of leading items and dependent items in the output. |
| UBIQUITOUS | DOUBLE | 1.0 | Ignores items whose support values are greater than the UBIQUITOUS value during the frequent items mining phase. |
| IS_USE_PREFIX_TREE | INTEGER | 0 | Indicates whether to use the prefix tree, which can save memory. <ul style="list-style-type: none">• 0: Does not use the prefix tree.• 1: Uses the prefix tree. |
| LHS_RESTRICT | VARCHAR | No default value | Specifies that some items are only allowed on the left-hand side of the association rules. |
| RHS_RESTRICT | VARCHAR | No default value | Specifies that some items are only allowed on the right-hand side of the association rules. |

| Name | Data Type | Default Value | Description |
|--------------------------|-----------|---------------|--|
| LHS_IS_COMPLEMENTARY_RHS | INTEGER | 0 | <p>If you use RHS_RESTRICT to restrict some items to the right-hand side of the association rules, you can set this parameter to 1 to restrict the complementary items to the left-hand side.</p> <p>For example, if you have 1000 items (i1, i2, ..., i1000) and want to restrict i1 and i2 to the right-hand side, and i3, i4, ..., i1000 to the left-hand side, you can set the parameters similar to the following:</p> <pre>INSERT INTO PAL_CONTROL_TBL VALUES ('RHS_RESTRICT', NULL, NULL, 'i1'); INSERT INTO PAL_CONTROL_TBL VALUES ('RHS_RESTRICT', NULL, NULL, 'i2'); INSERT INTO PAL_CONTROL_TBL VALUES ('LHS_IS_COMPLEMENTARY_RHS', 1, NULL, NULL);</pre> |
| RHS_IS_COMPLEMENTARY_LHS | INTEGER | 0 | <p>If you use LHS_RESTRICT to restrict some items to the left-hand side of the association rules, you can set this parameter to 1 to restrict the complementary items to the right-hand side.</p> |

| Name | Data Type | Default Value | Description |
|--------------|-----------|---------------|---|
| THREAD_RATIO | DOUBLE | 0 | Specifies the ratio of total number of threads that can be used by this function. The value range is from 0 to 1, where 0 means only using 1 thread, and 1 means using at most all the currently available threads. Values outside the range will be ignored and this function heuristically determines the number of threads to use. |
| TIMEOUT | INTEGER | 3600 | Specifies the maximum run time in seconds. The algorithm will stop running when the specified timeout is reached. |
| PMML_EXPORT | INTEGER | 0 | <ul style="list-style-type: none"> • 0: Does not export Apriori model in PMML. • 1: Exports Apriori model in PMML in single row. • 2: Exports Apriori model in PMML in several rows, and the minimum length of each row is 5000 characters. |

Output Tables

| Table | Column | Data Type | Column Name | Description |
|--------|------------|----------------|---------------|------------------------------|
| RESULT | 1st column | NVARCHAR(1000) | ANTECEDENT | Leading items |
| | 2nd column | NVARCHAR(1000) | CONSEQUENT | Dependent items |
| | 3rd column | DOUBLE | SUPPORT | Support value |
| | 4th column | DOUBLE | CONFIDENCE | Confidence value |
| | 5th column | DOUBLE | LIFT | Lift value |
| MODEL | 1st column | INTEGER | ROW_INDEX | ID |
| | 2nd column | NVARCHAR(5000) | MODEL_CONTENT | Apriori model in PMML format |

