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

THE BEST RUN SAP®

Table of Contents

DISCLAIMER

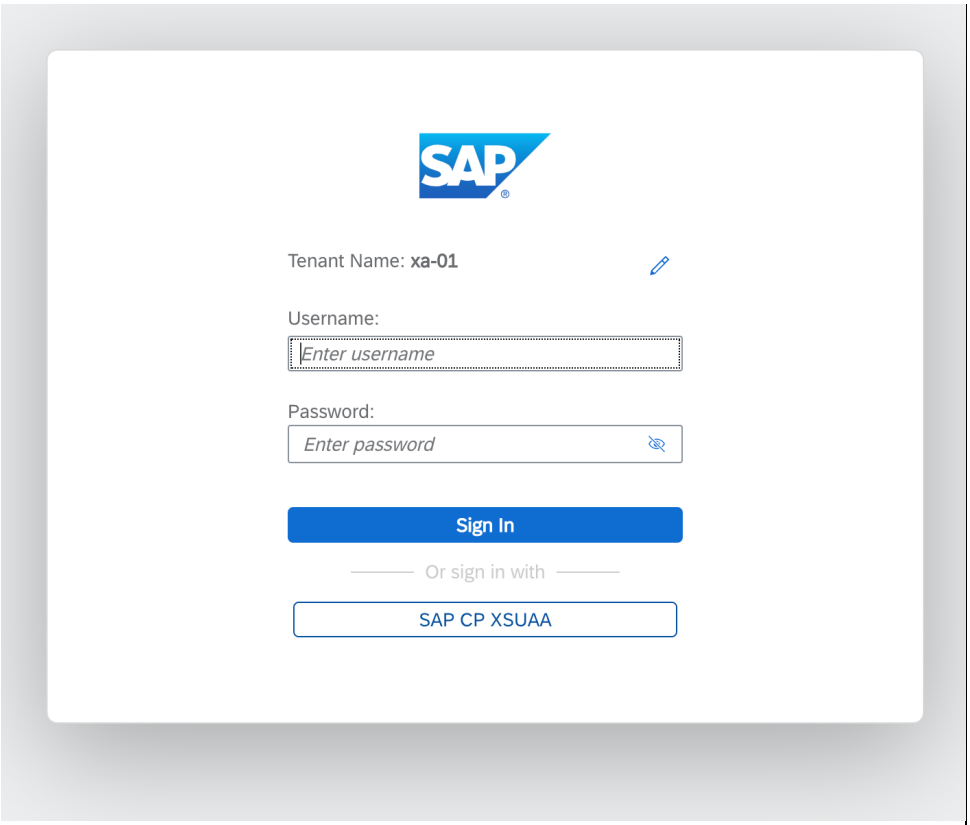
OBJECTIVE

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

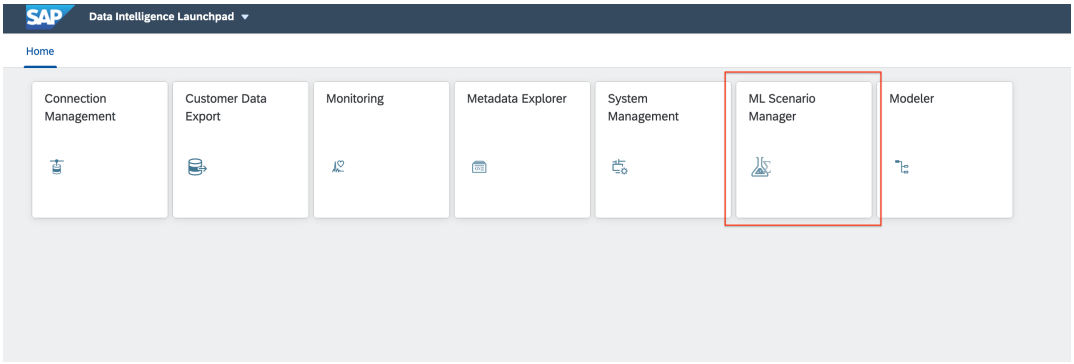
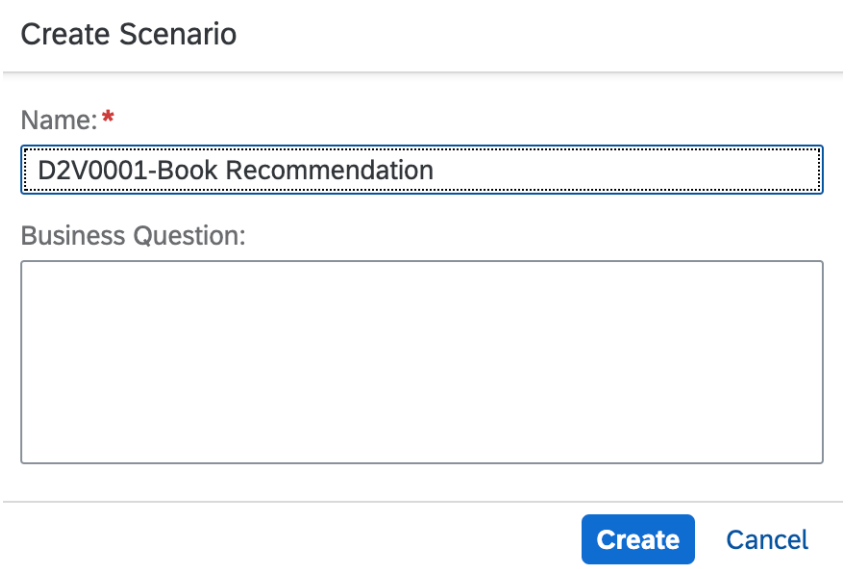
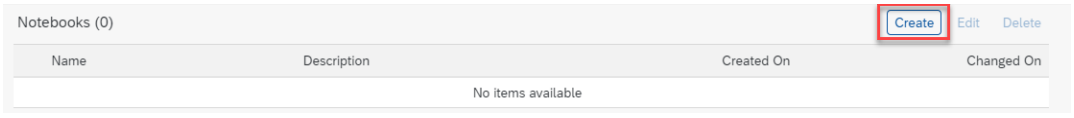
In this scenario, we want to build a book recommendation algorithm, using as input data source the history of book sales for a fictitious book shop having about 10 years of activity (2020 to 2021). The sales data are stored in a Data Warehouse Cloud instance. We will perform market-basket analysis on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

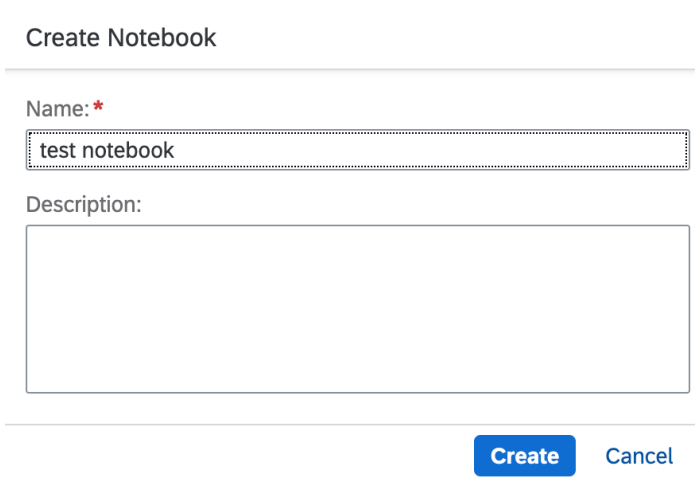
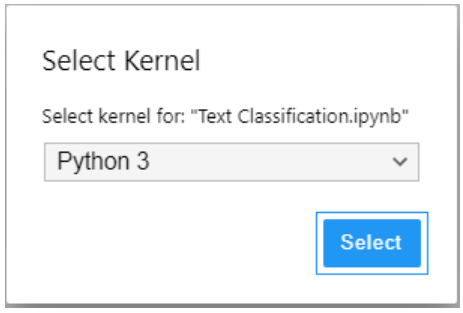
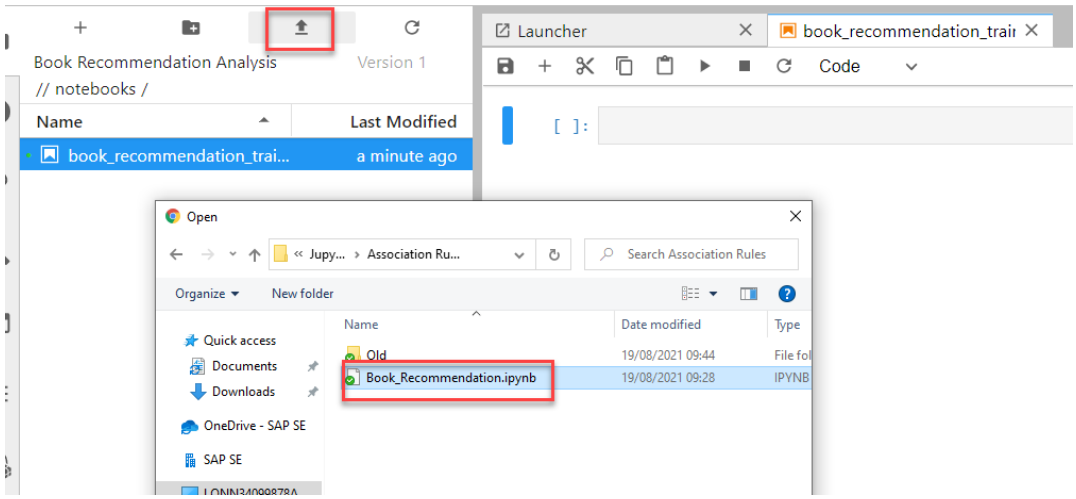
ENVIRONMENT ACCESS

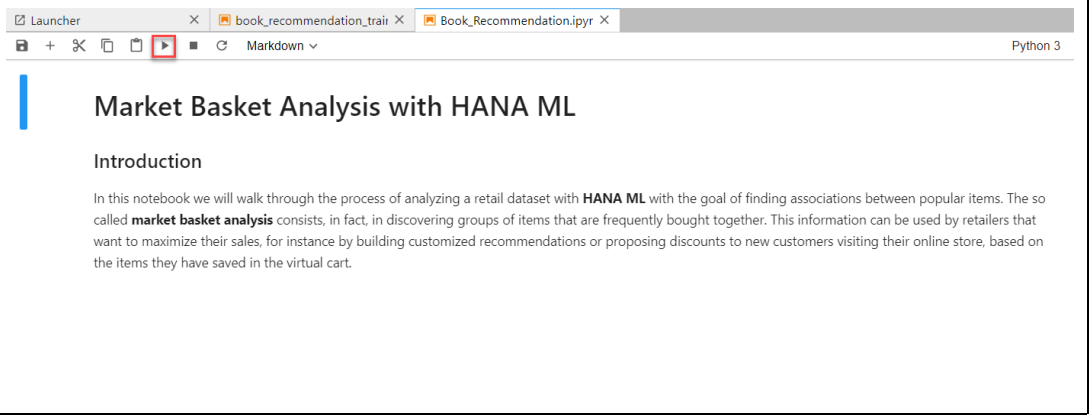
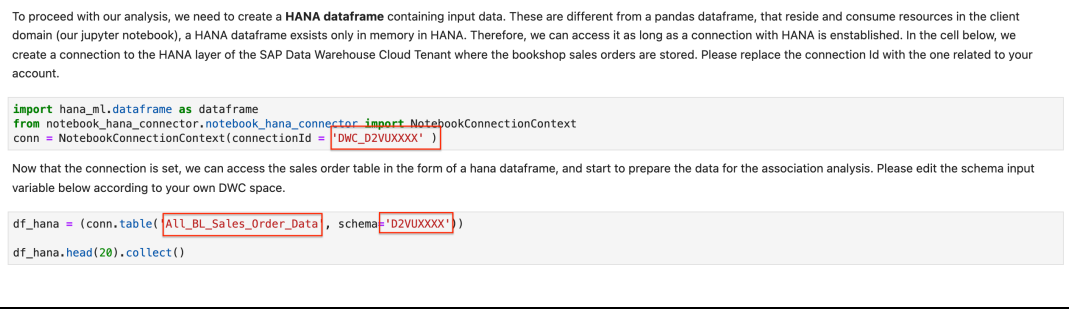
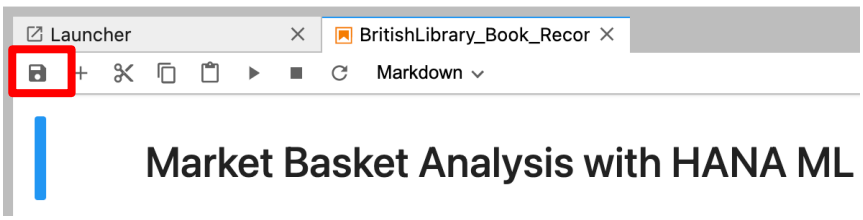
Explanation	Screenshot
Login to your SAP Data Intelligence tenant using the tenant name, username and password assigned to you	

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.

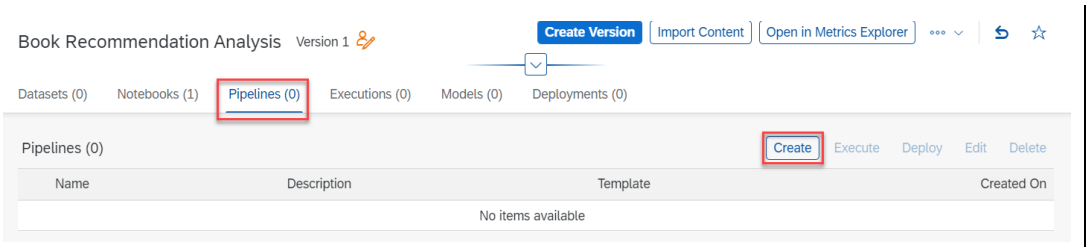
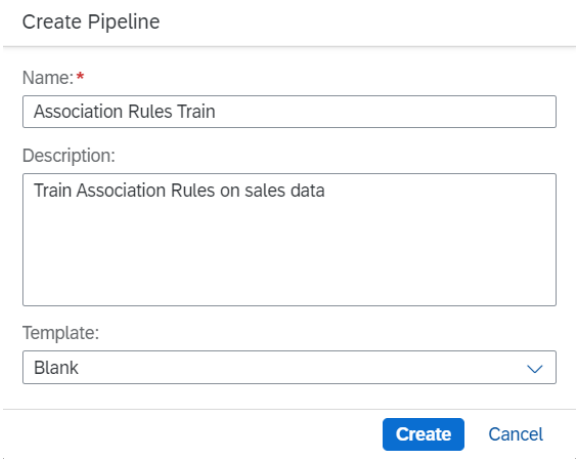
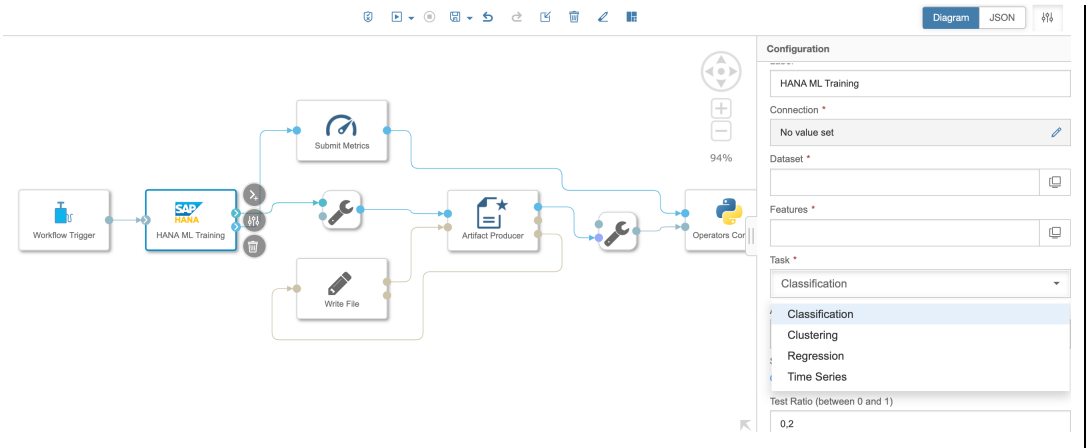
Explanation	Screenshot
Click to open ML Scenario Manager	
Click the Create button. Create a new scenario. Name the scenario according to your username: "D2VUXXXX Book Recommendation".	
In the Notebooks section, click Create to create a new notebook.	

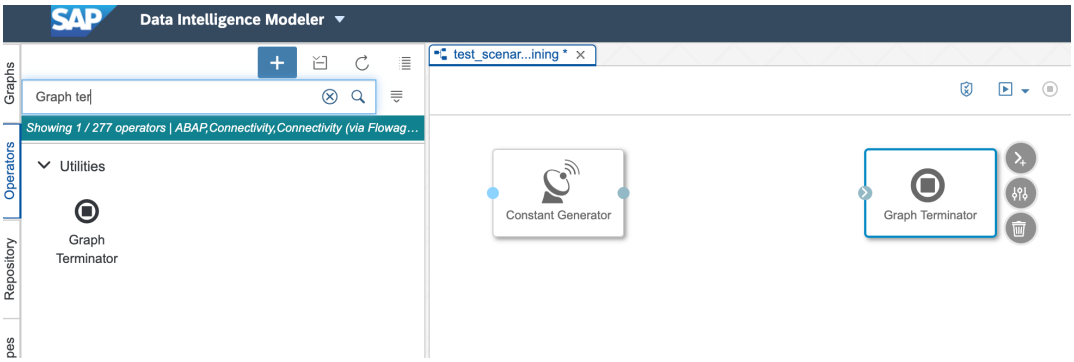
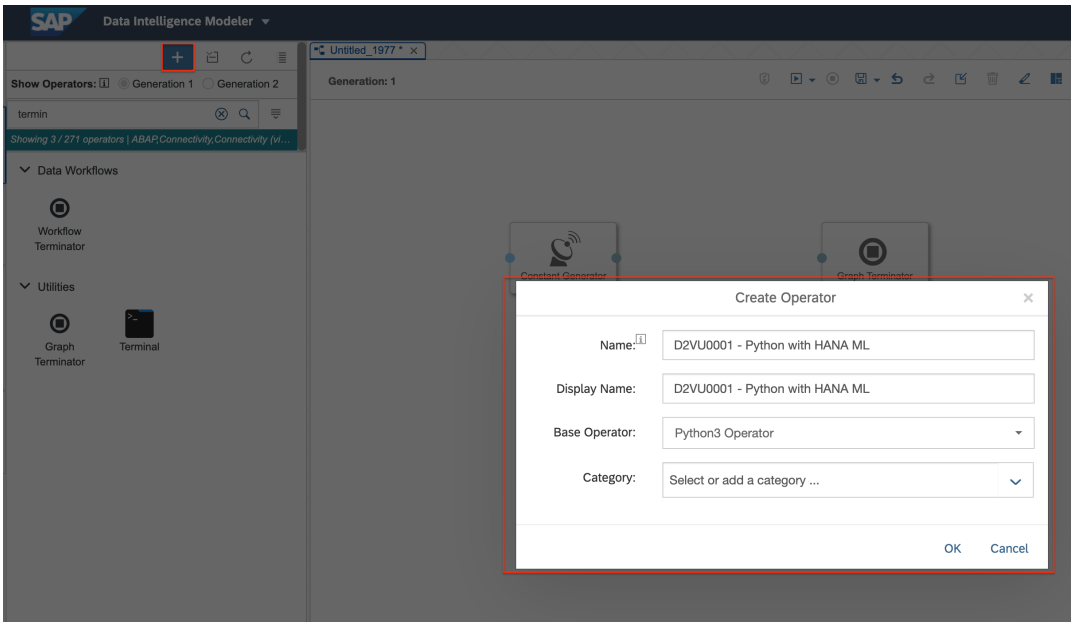
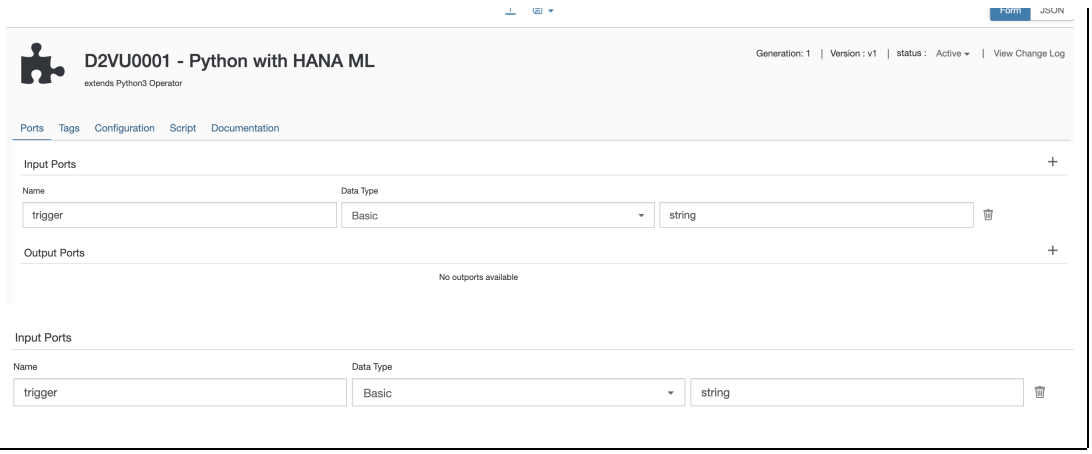
Explanation	Screenshot
Name the notebook “test notebook”	
Select the Python 3 Kernel option.	
Use the Upload Files function at the top left of the console to upload the Python notebook that we prepared for the exercise. Upload file Book_Recommendation.ipynb : you will find it in the course [github repository.](#) Double click on the notebook that is uploaded.	 Select file Book_Recommendation.ipynb

Explanation	Screenshot
Step through the code to check it works correctly. Highlight each cell in sequence and click the arrow button to run the selected cell and advance. Analyze the results of your data analysis and understand the association rules.	
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	
Once you have stepped through to the end of the notebook, and there are no errors, save the notebook.	

STEP 2 – BUILD MODEL PIPELINES

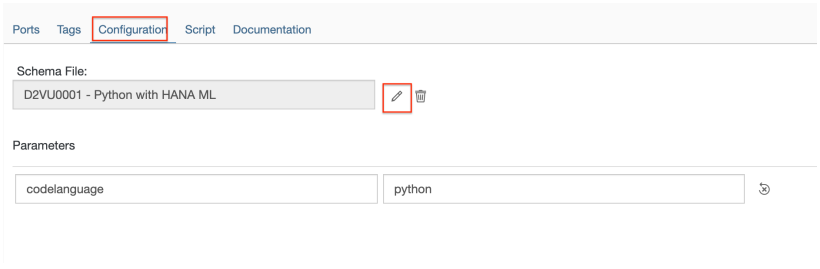
Model pipelines are used to operationalize the association rules model. Now that you have prepared the data, identified the best algorithm to use and which hyper-parameters work best, you can operationalize the model. We will build one pipeline to automate model training, and a second pipeline to deploy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

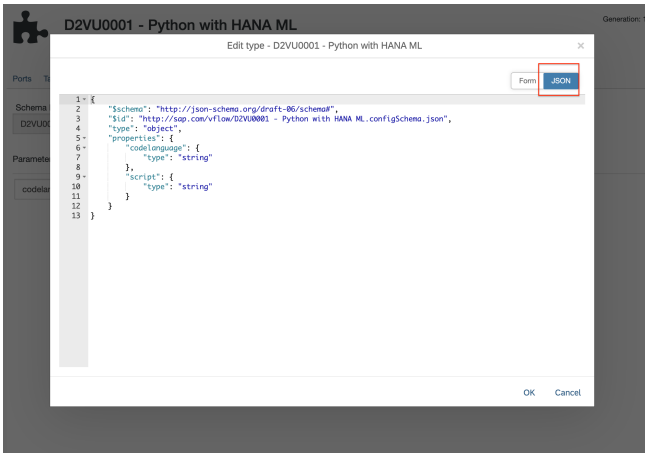
Explanation	Screenshot
To create the graphical pipeline to retrain the association rules, go to your ML Scenario's main page, select the "Pipelines" tab and click Create	
Name the pipeline "Association Rules Train" and select the "**Blank**" template to create a bespoke pipeline graph. Click Create.	
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 cover by the HANA ML training operator, that's why we will need a customized pipeline.	

Explanation	Screenshot
In the blank pipeline, we'll start by adding two operators: 1. Constant Generator 2. Graph Terminator Select each, and double click or drag and drop to add to console. These will be the start and the end of our pipeline.	
Then, we will build a custom operator to run association rules in Hana ML. 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**.	
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.	

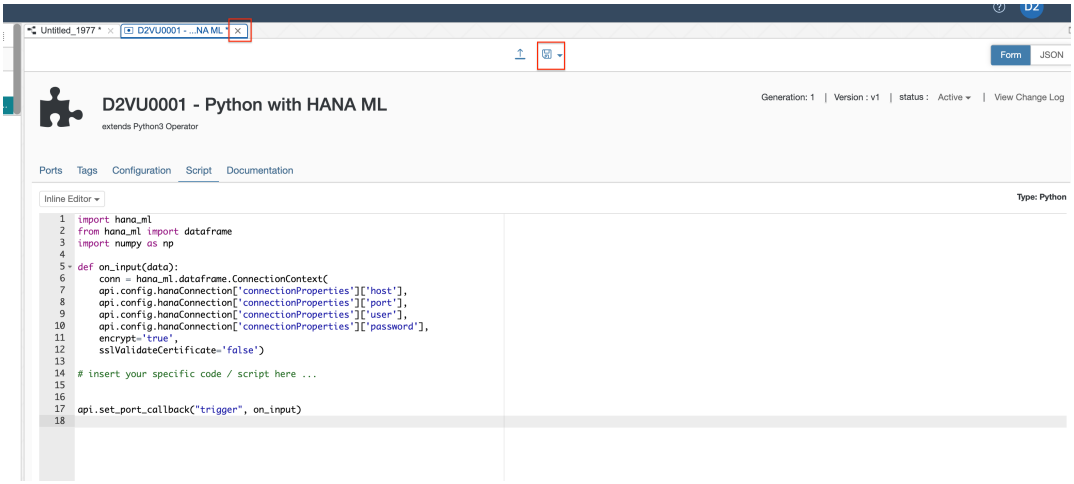
Go to the *Configuration* tab. We will configure the operator so that it can **connect to a HANA instance connected to SAP Data Intelligence**. Click the pencil to get into the edit mode and change to JSON.

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

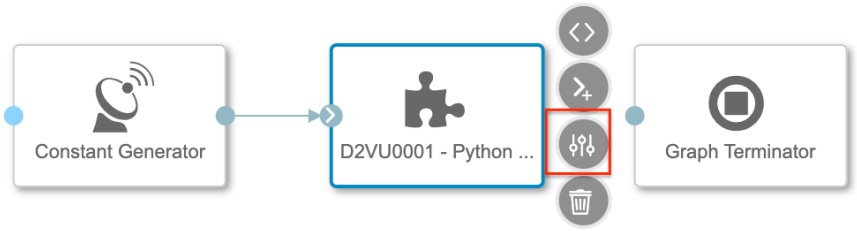




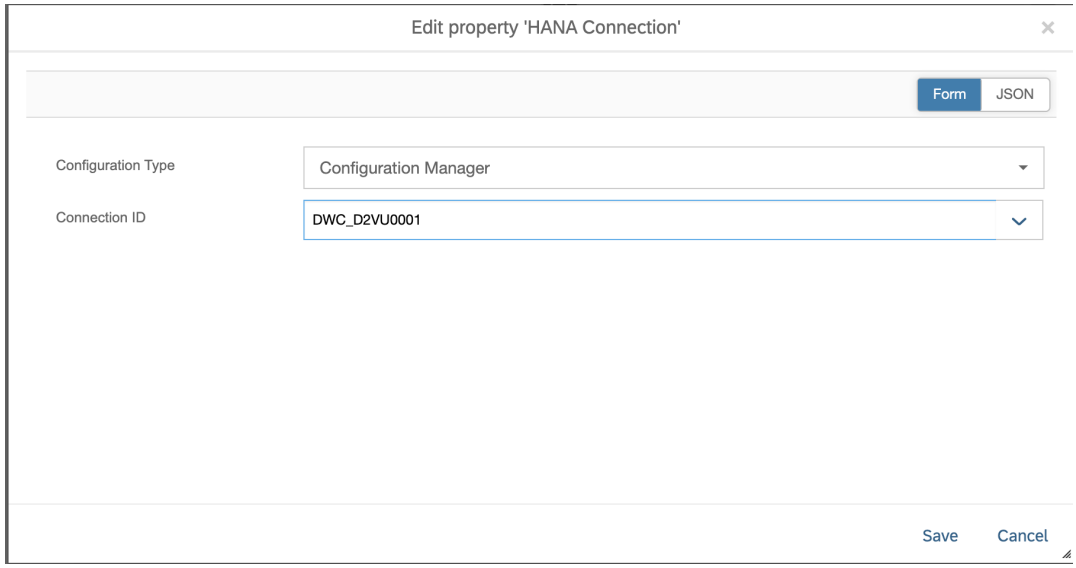

```
{
                "$schema": "http://json-schema.org/draft-06/schema#",
                "$id": "http://sap.com/vflow/HANA ML BLOG.configSchema.json",
                "type": "object",
                "properties": {
                                "hanaConnection": {
                                                "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":
"configurationType",
                                                                                                                                "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":
"configurationType",
                                                                                                                                "value":
"Manual"
                                                                                                                }
                                                                                                ]
                                                                                }
                                                                }
                                                },
                                                "required": []
                                },
                                "codelanguage": {
                                                "type": "string"
                                },
                                "script": {
                                                "type": "string"
                                }
```

	}, "required": [] }
Go to the *Script* tab and copy paste the python code to the right.	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)
Save the custom operator and the close the window.	
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.	

Begin with configuring the HANA ML within Python operator. Highlight it and click on the Configuration icon.



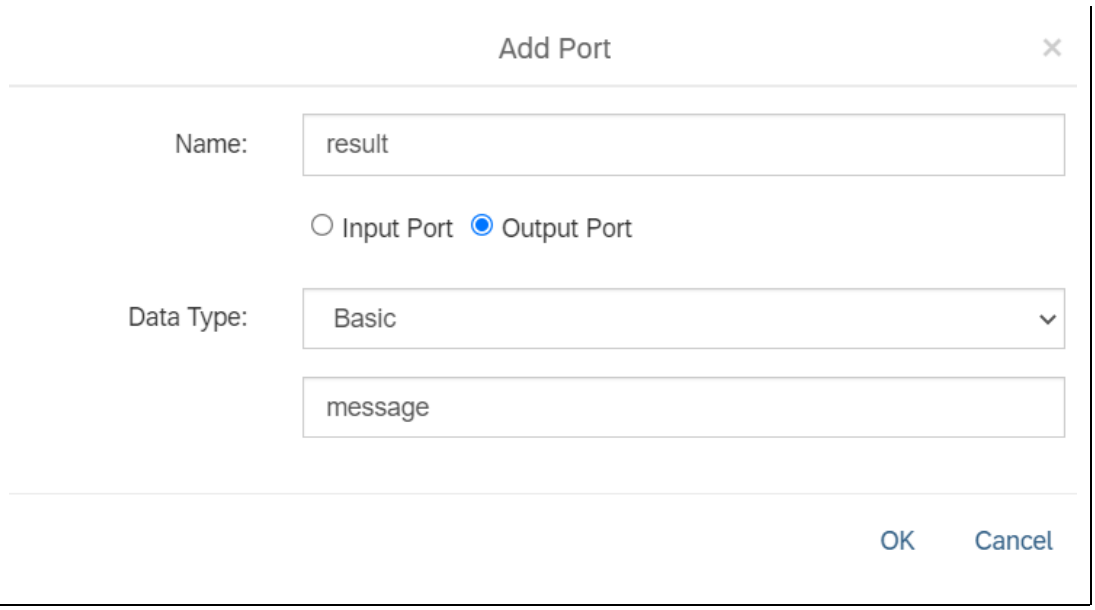
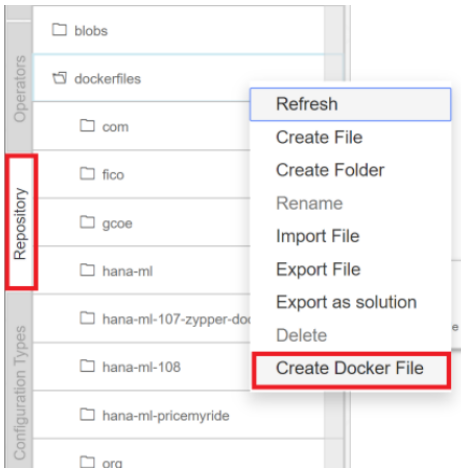
Edit the **Hana Connection** field. Select Configuration Manager as Configuration Type and select your DWC connection in the Connection ID drop-down menu. Click save.

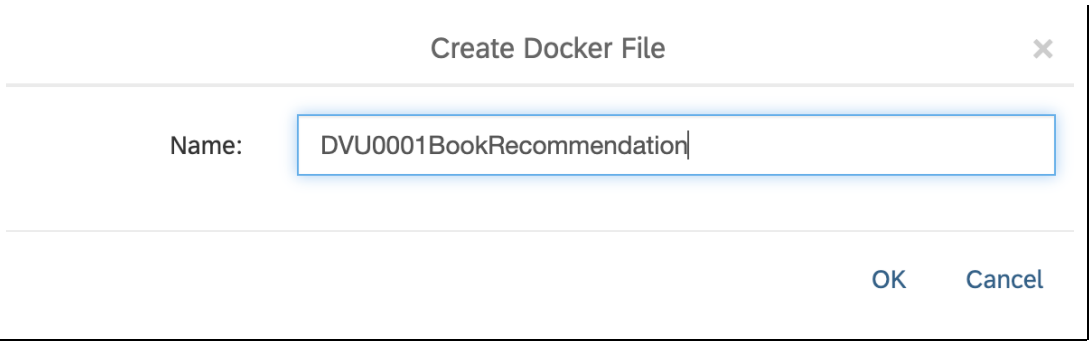
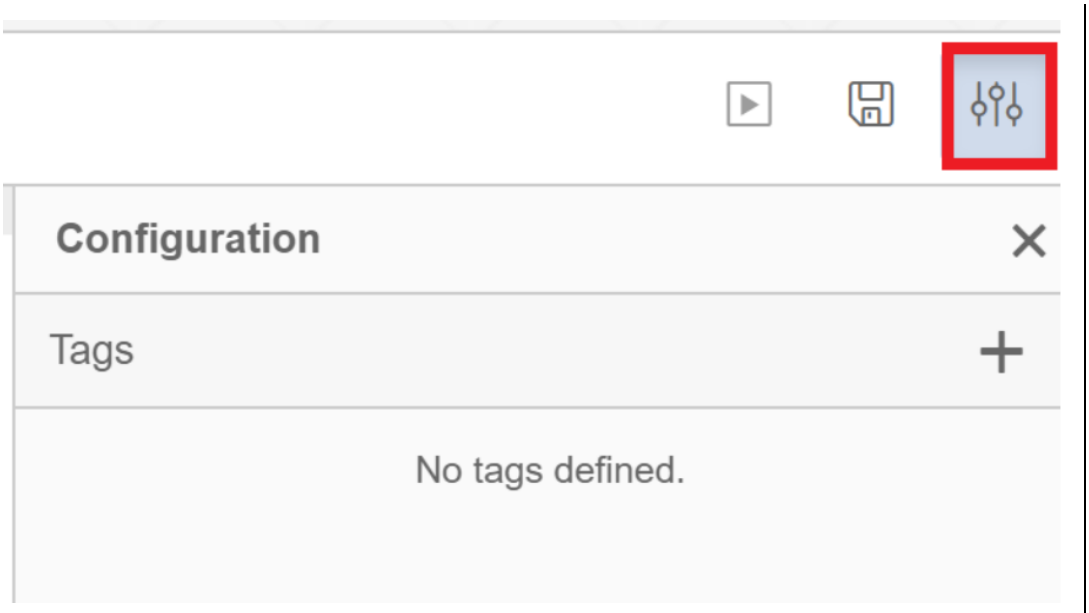
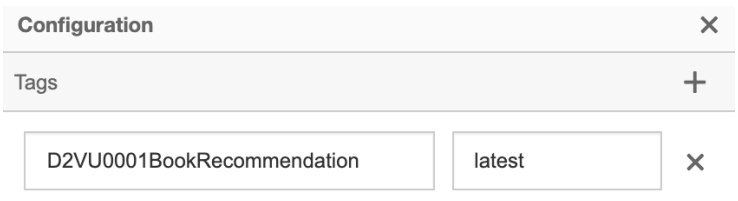


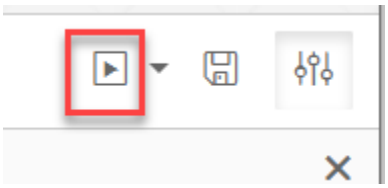
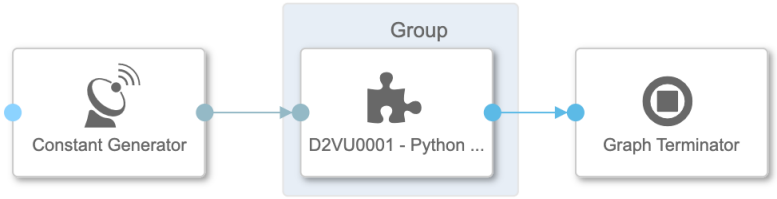
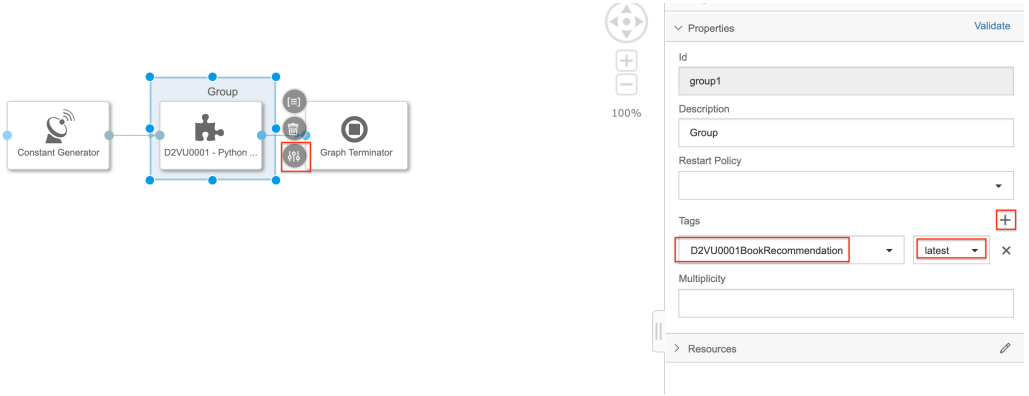
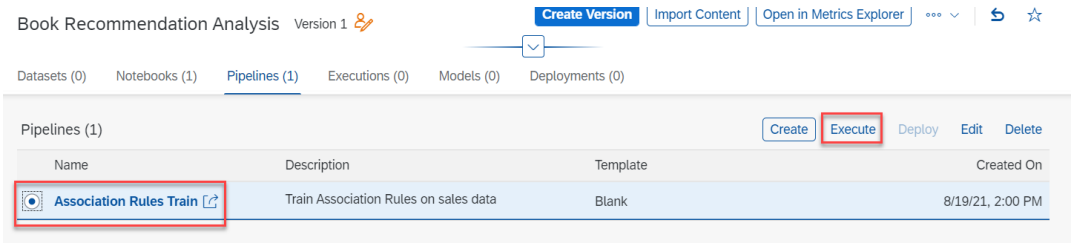
In the HANA ML within Python operator, click the script icon.

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

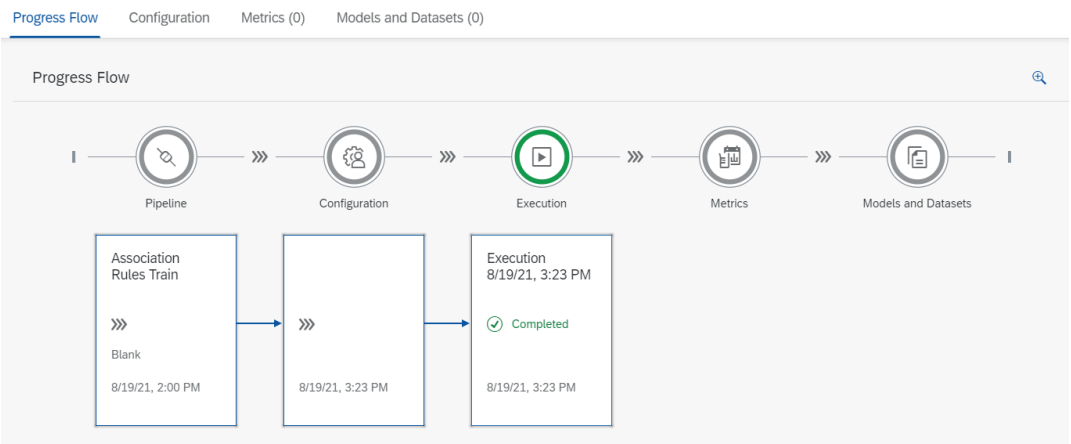
```
import hana_ml
from hana_ml import dataframe
import numpy as np

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='<XXXXXXXXX>'))
    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
```

sensitive to indentation!	```) 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) ```
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.	
Link the result port of the custom operator to the Graph Terminator.	
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	

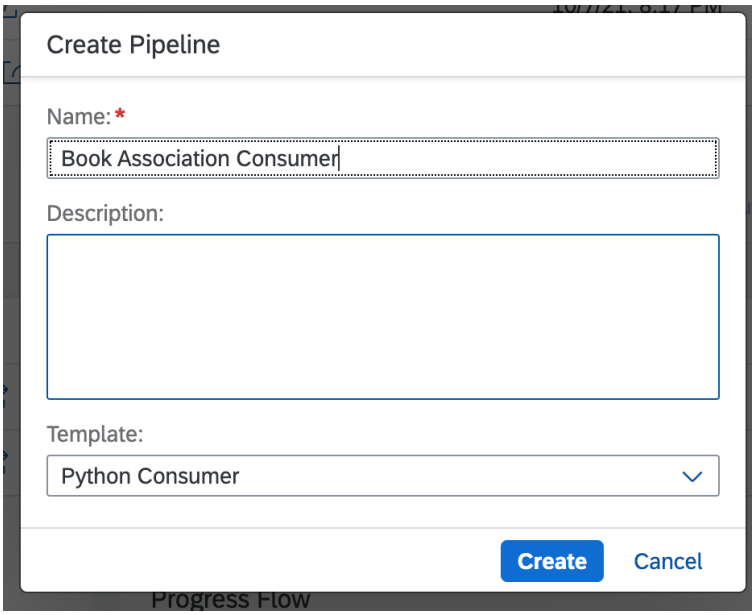
and select “Create Docker File”.	
Name the file according to your username D2VUxxxxBookRecommendation	
Enter this code into the Docker File window. This code leverages a base image that comes with SAP Data Intelligence and installs the necessary libraries on it.	FROM $com.sap.sles.base RUN pip3.6 install --user numpy==1.16.4 RUN pip3.6 install --user pandas==0.24.0 RUN pip3.6 install --user hana_ml==2.9.21072600
Open the Configuration panel for the Docker File with the icon on the top-right hand corner.	
Assign a tag to the Dockerfile	

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.

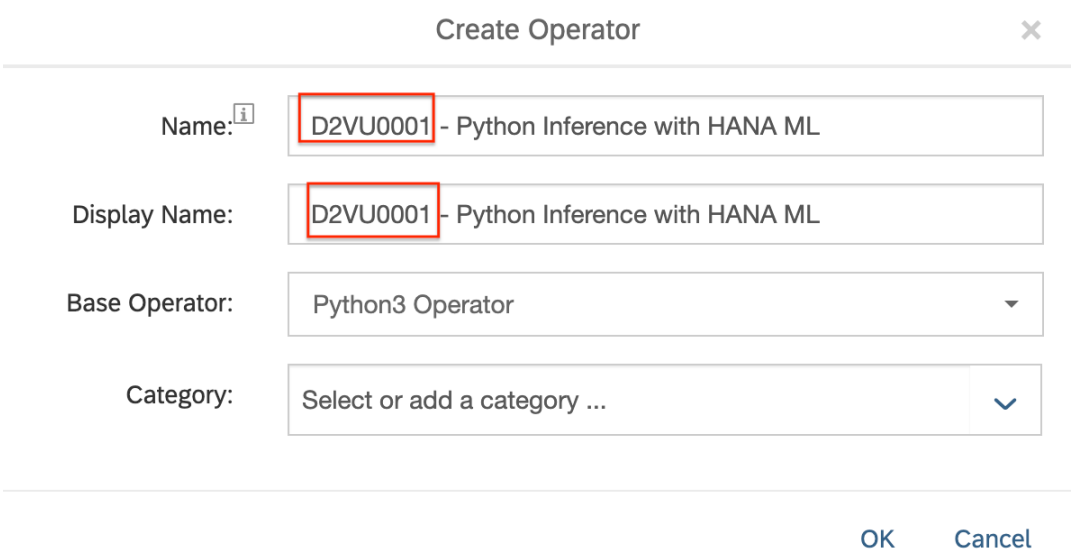


Take a moment to understand this template. The first portion of the graph, shown in red in the picture, has the function of reading a model artifact stored in a binary file and feeding it to the second portion of the graph. As a matter of fact, in machine learning scenarios, trained models such as regressors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

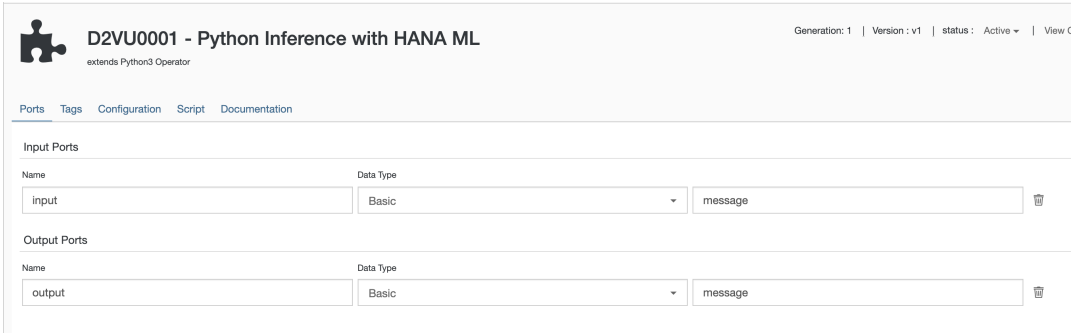
classifiers, are usually stored in binary format. In our scenario, however, the results of Apriori are not stored in an artifact, but in a HANA table. We can delete the operators within the red box, we will not need them. The blue portion of the graph, instead, is what we need to expose the book association 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.	

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

to create a new operator. Configure the dialog box as shown here. Use your own username to name the operator.



Add two ports as shown in the picture



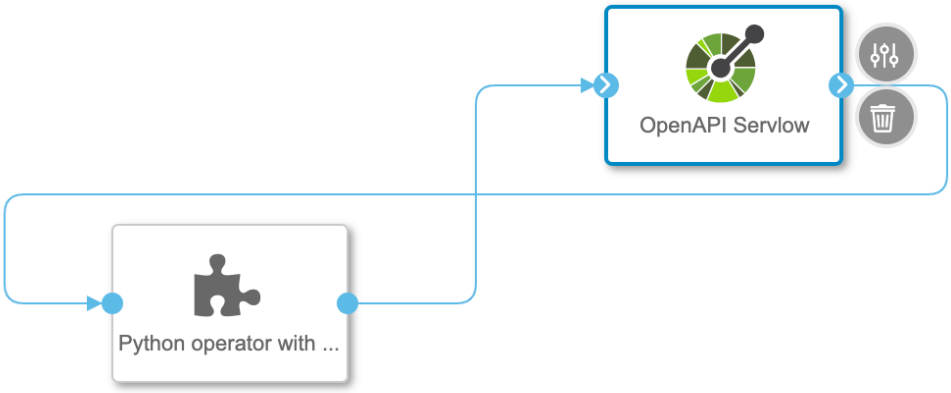
Go to the Configuration tab and repeat the same steps you did for the training custom operator.

In the Script session, enter the piece of code here.

Save the operator and go back to your blank pipeline

```
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):
```

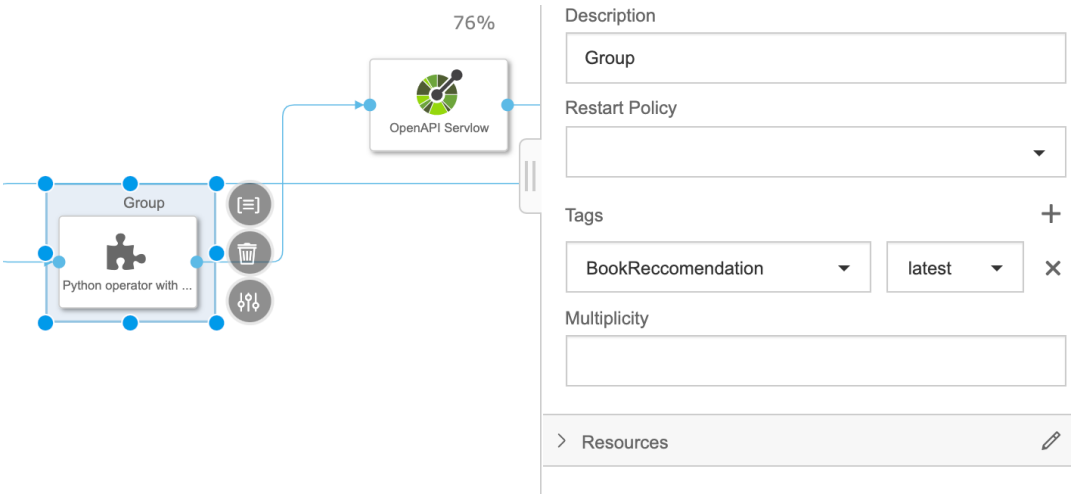
```
  try:
    json_object = json.loads(data)
  except ValueError as e:
    return False
  return True

# 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
```

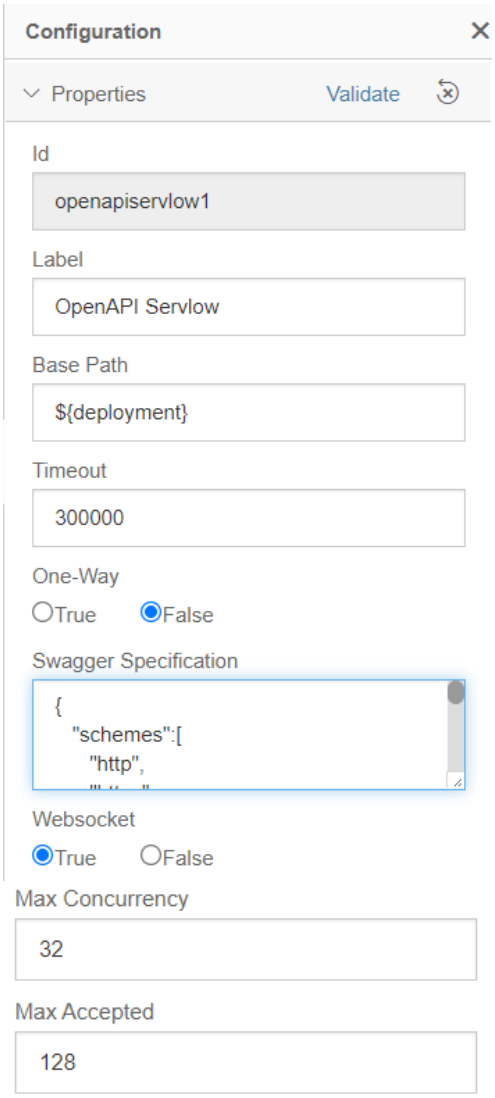
	except Exception as e: api.logger.error(e) error_message = "An error occurred: " + str(e) if success: # apply carried out successfully, send a response to the user msg.body = json.dumps({'prediction': prediction}) else: msg.body = json.dumps({'Error': error_message}) new_attributes = {'message.request.id': msg.attributes['message.request.id']} msg.attributes = new_attributes api.send('output', msg) api.set_port_callback("input", on_input)
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.	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:

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#DWCUSER>')) 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'

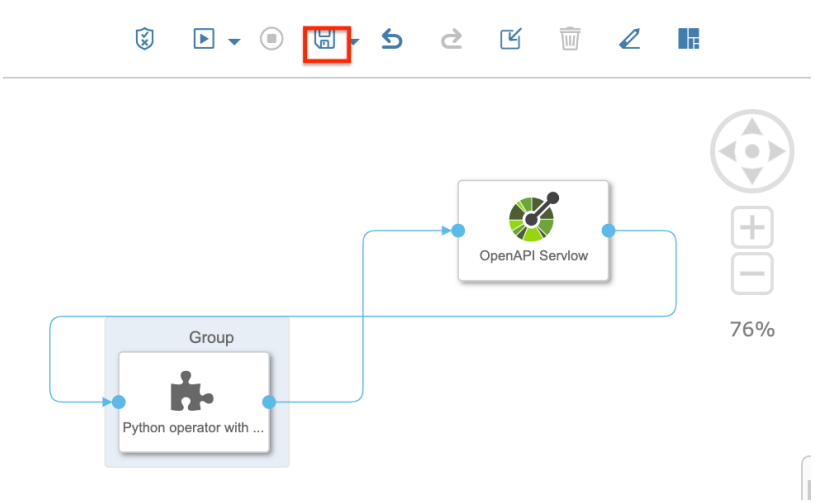
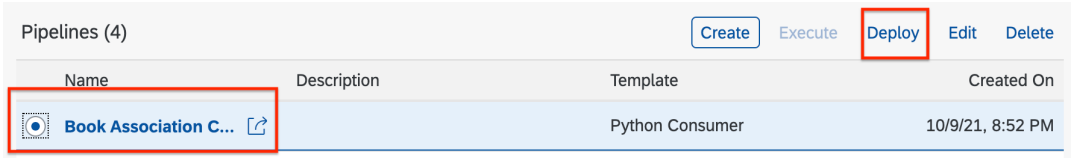
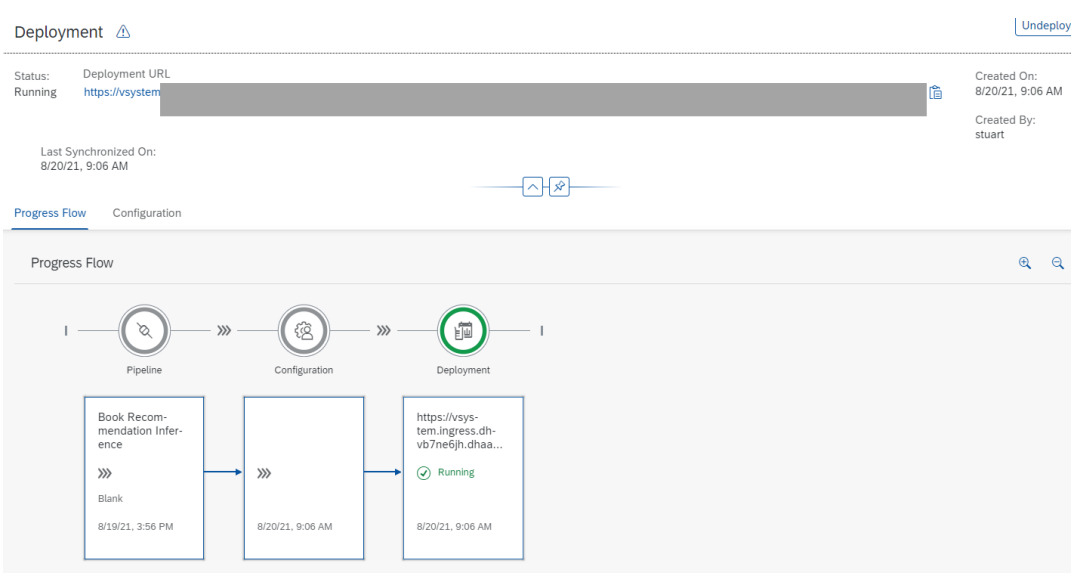
```
            success = True
        else:
            api.logger.info("Invalid JSON received from client - cannot apply
model.")
            error_message = "Invalid JSON provided in request: " + user_data
            success = False
    else:
        api.logger.info("Model has not yet reached the 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.



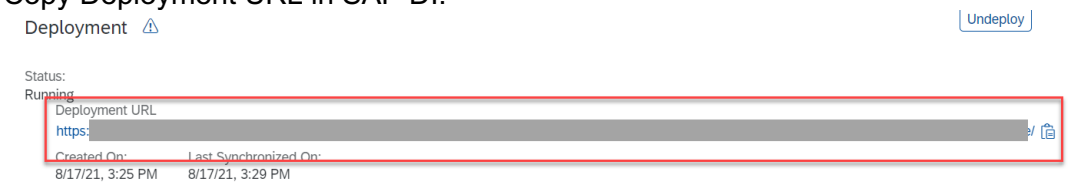
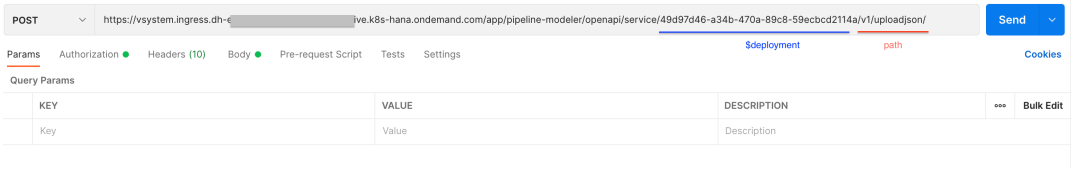
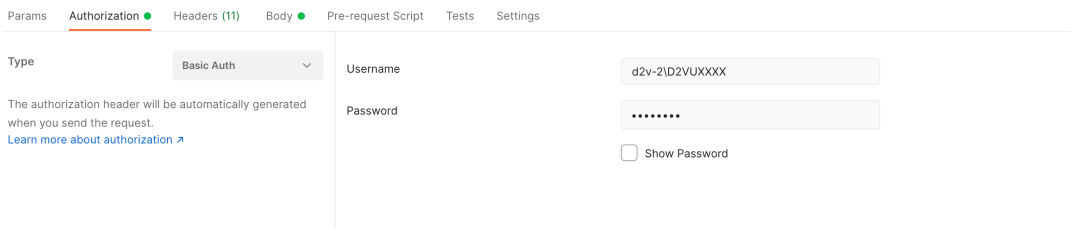
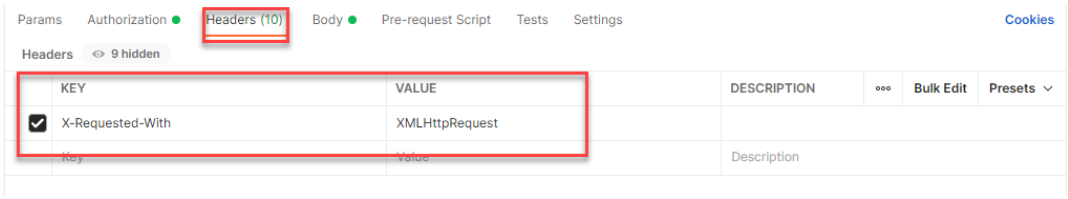
Click on OpenAPIServlow and have a look at the configuration	
Notice in particular the content of the Swagger Specification.	{ "schemes":["http", "https"], "swagger":"2.0", "info":{ "description":"This is an example of using the OpenAPI Servlow to carry out inference with an existing model.", "title":"OpenAPI demo", "termsOfService":"http://www.sap.com/vora/terms/", "contact":{ }, "license":{ "name":"Apache 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”.	
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
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”.	Copy Deployment URL in SAP DI:  To Postman: 
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.	
Go to the “Headers” tab and enter the key “X-Requested-With” with value “XMLHttpRequest”.	
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	{ "book”: <book_ID>

Explanation	Screenshot
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.	} Authorization ● Headers (2) Body ● Pre-request Script Tests form-data x-www-form-urlencoded raw binary Text 1 { 2 "book": "16461037" 3 } 4
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.	Body Cookies Headers (12) Test Results Pretty Raw Preview JSON 1 { 2 "prediction": [3 [4 "15455315" 5], 6 [7 "15478780" 8], 9 [10 "15341651" 11], 12 [13 "15103327" 14], 15 [16 "14641160" 17] 18] 19 }
You have now completed the exercise.	

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 casual 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. • 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	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. 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: `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);`
RHS_IS_COMPLEMENTARY_LHS	INTEGER	0	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.

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	• 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