



INTERNAL

## SAP Data Intelligence hands-on exercises

This document will guide you step-by-step through the process of training and implementing a text analysis and developing a cluster machine learning model using Python and SAP DI Pipelines.

[www.sap.com/contactsap](http://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](http://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 CLUSTER MODEL .....	23

## **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.

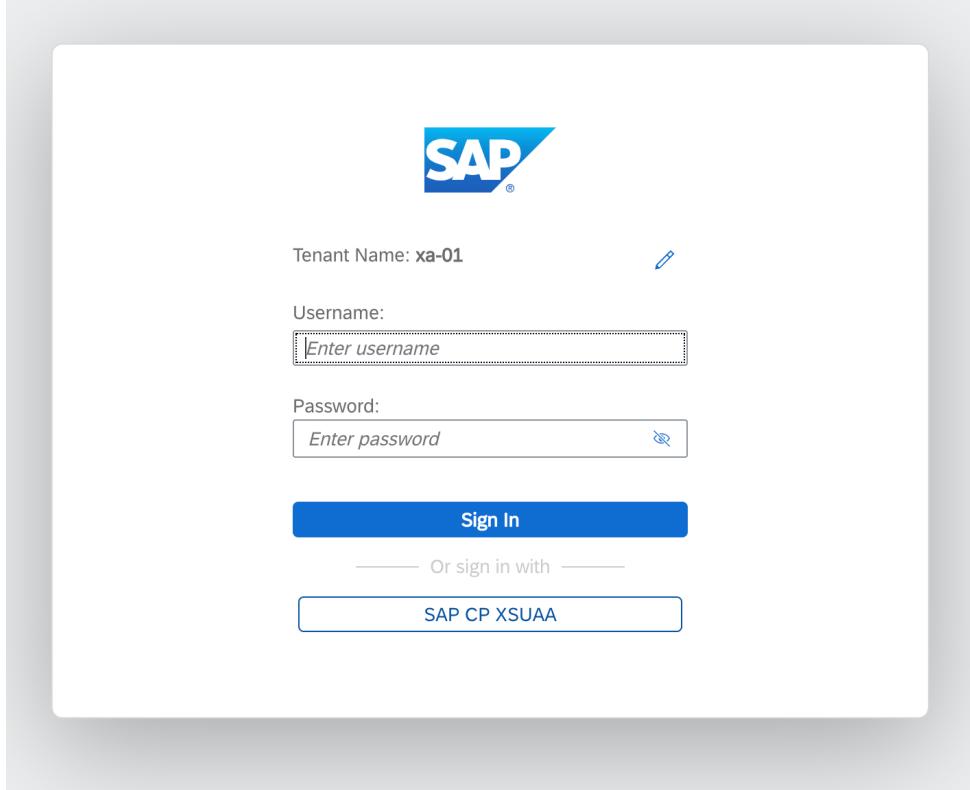
## **OBJECTIVE**

The objective of this exercise is to give you an overview of how you can use the machine learning capabilities in SAP Data Intelligence.

## **SCENARIO**

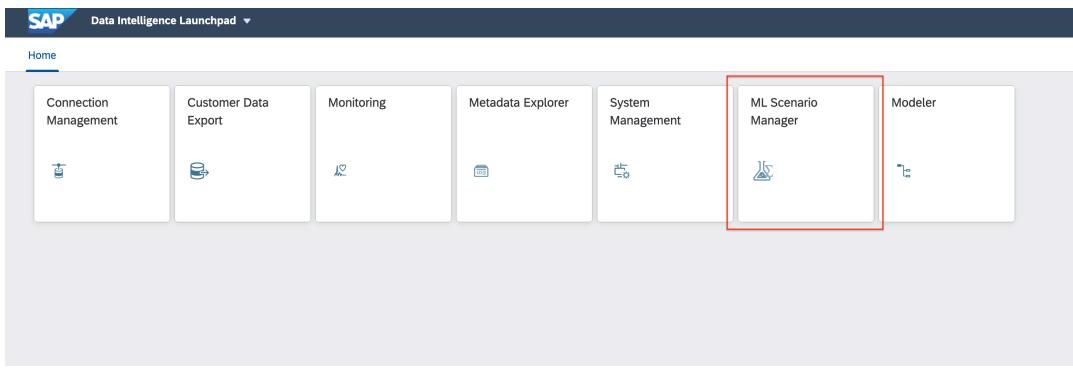
Books are grouped together in a bookshop based on their similarity, so that customers browsing for a book will find lots of similar books on the same shelf. This exercise analyzes the book description data using Python text mining algorithms and then uses this information to assign each book to a cluster. The books within a cluster are as similar as possible (based on the book description), so they are as homogeneous as possible, and there is as wide a difference as possible between clusters, so the different clusters are heterogeneous.

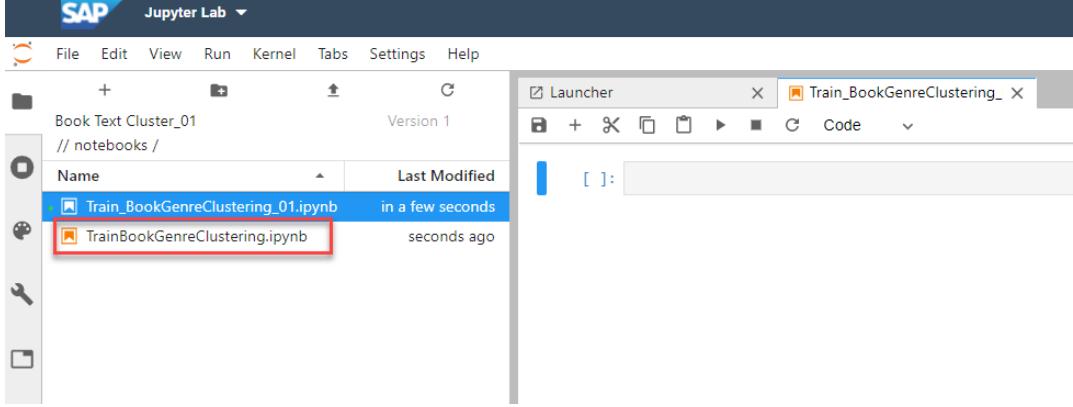
## ENVIRONMENT ACCESS

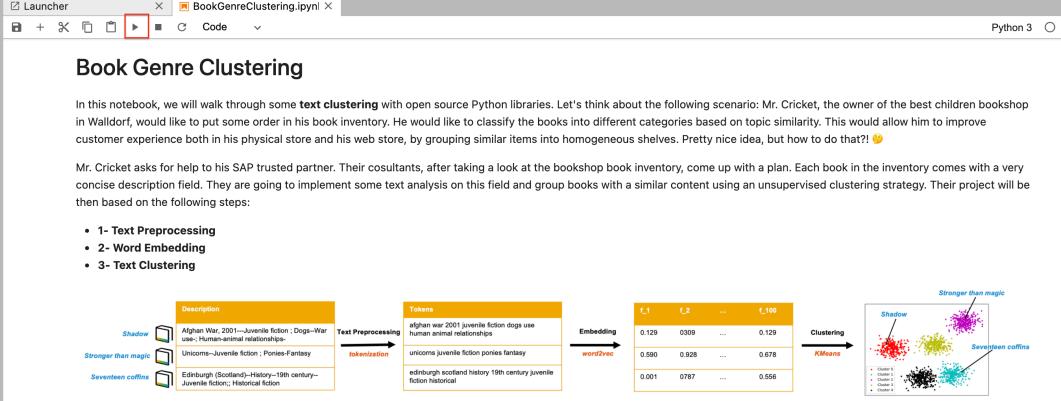
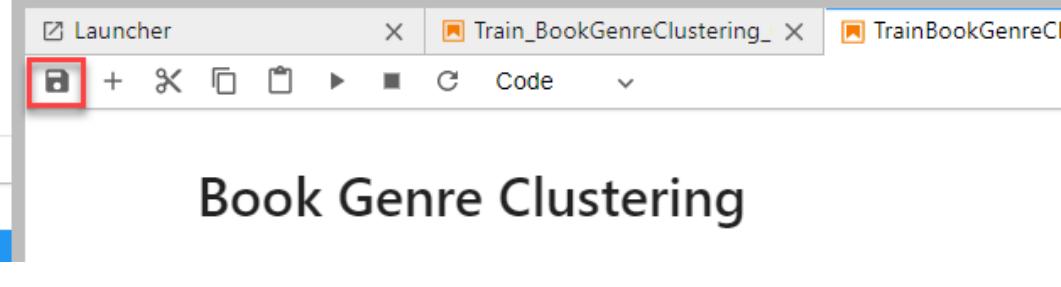
Explanation	Screenshot
Login to your SAP Data Intelligence tenant using the tenant name, username and password assigned to you	

## STEP 1 – USE A JUPYTER NOTEBOOK

A Jupyter Notebook environment is used to explore the data, and to run predictive model tests to compare the accuracy of different algorithms and the best settings for the hyper-parameters for the algorithms.

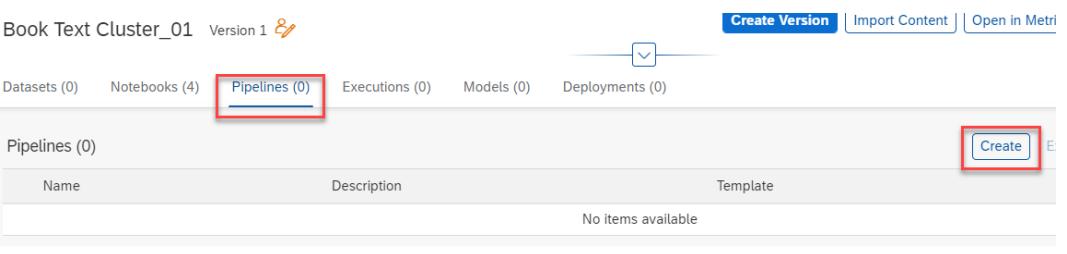
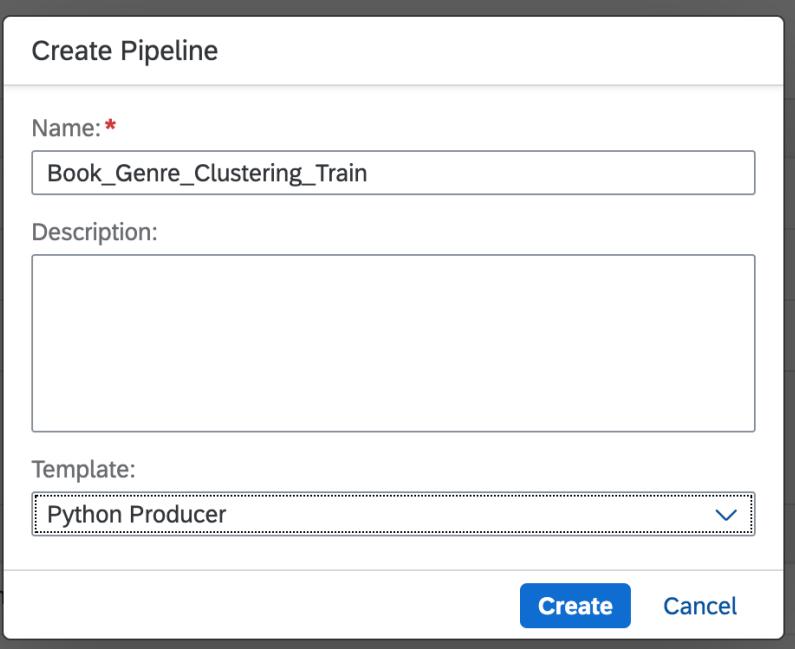
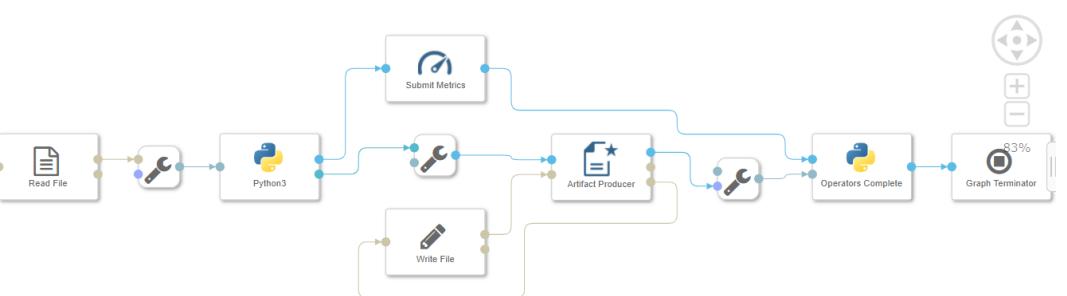
Explanation	Screenshot												
Click to open ML Scenario Manager													
Click the Create button. Create a new scenario. Name the scenario according to your username: "D2VUXXXX Book Clustering".	<p>Create Scenario</p> <p>Name:*</p> <input type="text" value="D2V0001 - Book Clustering"/> <p>Business Question:</p> <p><b>Create</b>   <b>Cancel</b></p>												
In the Notebooks section, click Create to create a new notebook.	<table border="1"> <thead> <tr> <th data-bbox="425 1389 1498 1419">Notebooks (0)</th> <th data-bbox="1318 1396 1367 1423"><b>Create</b></th> <th data-bbox="1372 1396 1449 1423">Edit</th> <th data-bbox="1388 1396 1449 1423">Delete</th> </tr> <tr> <th data-bbox="425 1431 584 1461">Name</th> <th data-bbox="584 1431 829 1461">Description</th> <th data-bbox="829 1431 1237 1461">Created On</th> <th data-bbox="1237 1431 1498 1461">Changed On</th> </tr> </thead> <tbody> <tr> <td colspan="4" data-bbox="425 1461 584 1491">No items available</td> </tr> </tbody> </table>	Notebooks (0)	<b>Create</b>	Edit	Delete	Name	Description	Created On	Changed On	No items available			
Notebooks (0)	<b>Create</b>	Edit	Delete										
Name	Description	Created On	Changed On										
No items available													

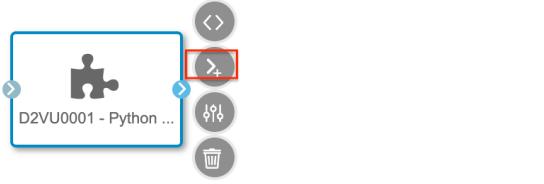
Explanation	Screenshot
Name the notebook with a name of your choice	<p>Create Notebook</p> <p>Name:*</p> <input type="text" value="test notebook"/> <p>Description:</p> <div style="border: 1px solid #ccc; height: 100px; width: 100%;"></div> <p style="text-align: right;"><span style="background-color: blue; color: white; padding: 2px 10px; border-radius: 5px;">Create</span> Cancel</p>
Select the Python 3 Kernel option.	<p>Select Kernel</p> <p>Select kernel for: "Text Classification.ipynb"</p> <div style="border: 1px solid #ccc; padding: 5px; width: fit-content;">         Python 3         <div style="float: right; margin-top: -10px;">▼</div> </div> <p style="text-align: center; margin-top: 10px;"><span style="background-color: blue; color: white; padding: 5px 20px; border-radius: 5px;">Select</span></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 <b>BookGenreClustering.ipynb</b>, you will find it in the course <a href="#">github repository</a>.</p>	 <p>Select file BookGenreClustering.ipynb</p>

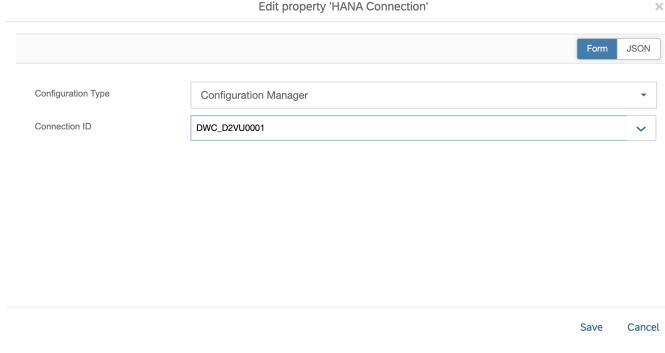
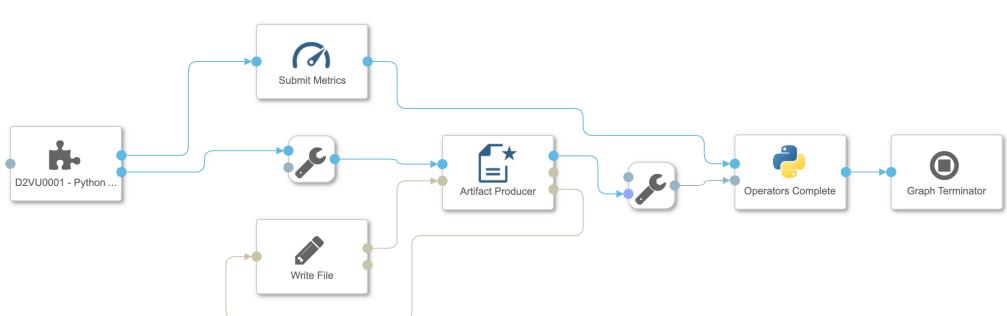
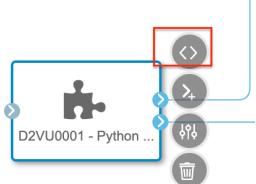
Explanation	Screenshot
<p>Double click on the notebook that is uploaded.</p> <p>Step through the code to check it works correctly.</p> <p>Highlight each cell in sequence and click the arrow button to run the selected cell and advance. Note that some of the steps can take a few minutes to complete.</p> <p>Analyze the results of your data analysis and understand the text analysis and cluster model results.</p>	 <p>The screenshot shows a Jupyter Notebook interface with the title 'Book Genre Clustering'. The notebook content includes a flow diagram illustrating the text analysis pipeline: 'Text Preprocessing' leads to 'Tokenization', which then leads to 'Embedding' using 'word2vec'. The resulting matrix has columns labeled L1, L2, ..., L199. Following this, a 'Clustering' step using 'KMeans' is shown, resulting in four distinct clusters: 'Shadow' (red), 'Stronger than magic' (yellow), 'Unicorn' (green), and 'Seventeen coffins' (blue). Below the diagram, a note says 'Let's put this into practice!' and provides the Python code for basic imports and text preprocessing.</p> <pre data-bbox="486 819 665 946"> [1]: # basic Python !pip install pandas !pip install numpy !pip install matplotlib !pip install seaborn  # text preprocessing !pip install regex !pip install nltk </pre>
<p>Adapt the notebook according to your own DWC connection and schema</p>	<pre data-bbox="445 1015 1367 1115"> import hana_ml.dataframe as dataframe from notebook_hana_connector.notebook_hana_connector import NotebookConnectionContext conn = NotebookConnectionContext(connectionId = 'DWC_D2VUXXXX') df_hana = (conn.table('Book_Author_Dimension_View', schema='D2VUXXXX')) </pre>
<p>Once you have stepped through to the end of the notebook, and there are no errors, save the notebook.</p>	 <p>The screenshot shows a Jupyter Notebook interface with the title 'Train_BookGenreClustering.ipynb'. The 'Save' icon in the toolbar is highlighted with a red box. The notebook content includes the same 'Book Genre Clustering' title and diagram as the previous screenshot.</p>

## STEP 2 – BUILD MODEL PIPELINES

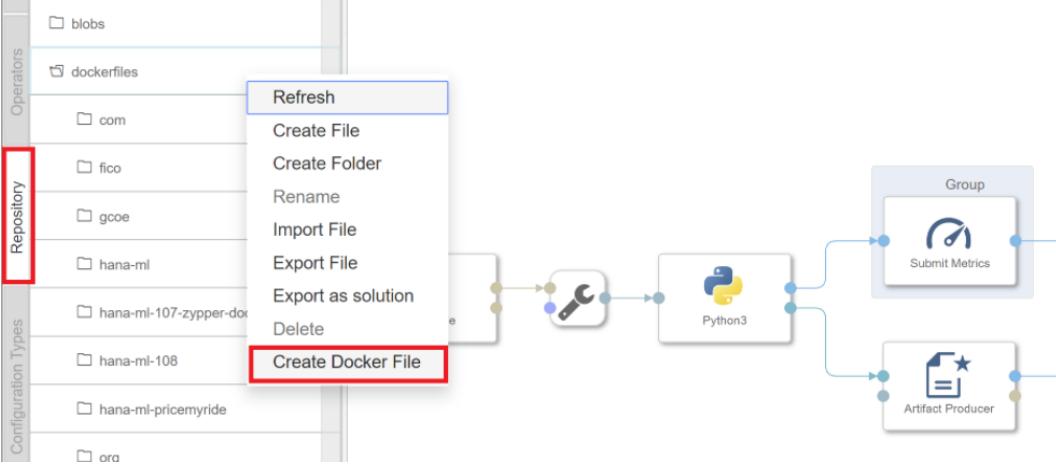
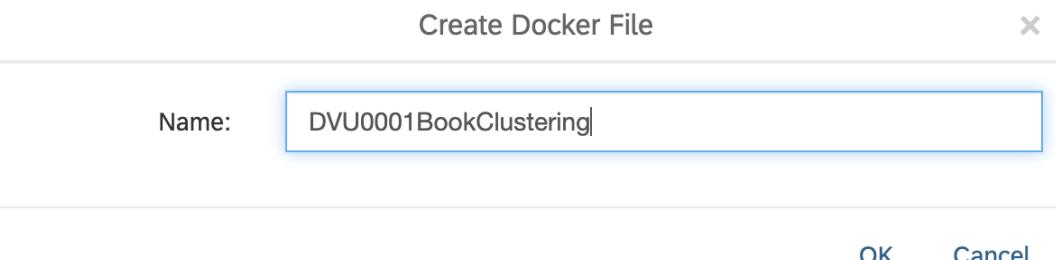
Model pipelines are used to operationalize the clustering model. Now that you have prepared the data, identify the best algorithm to use and which hyper-parameters work best, you want to take the model to production. You will build two pipelines. The first one is used to automate the model training, while the second one is used to inference the model on new data.

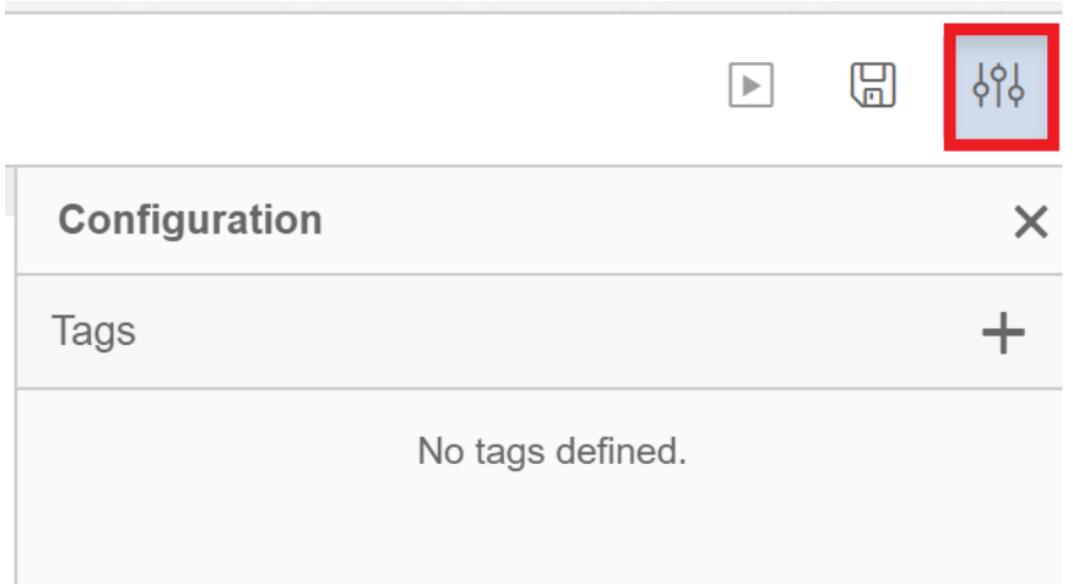
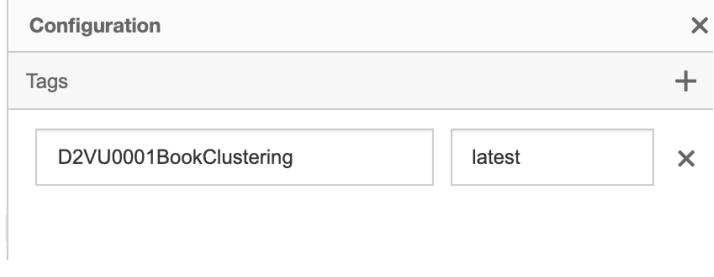
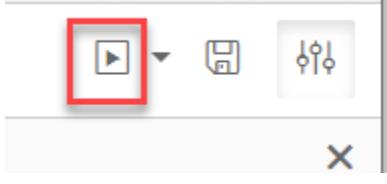
Explanation	Screenshot
<p>To create the graphical pipeline to retrain the model, go to your ML Scenario's main page, select the "Pipelines" tab and click Create.</p>	 <p>The screenshot shows the 'Book Text Cluster_01 Version 1' page. At the top, there are tabs for Datasets (0), Notebooks (4), Pipelines (0) (which is highlighted with a red box), Executions (0), Models (0), and Deployments (0). Below the tabs, there is a table header for 'Pipelines (0)' with columns for Name, Description, and Template. A large red box highlights the 'Create' button in the top right corner of the pipeline list area.</p>
<p>Name the pipeline "Text Clustering Train" and select the "Python Producer" template. Click Create.</p>	 <p>The screenshot shows the 'Create Pipeline' dialog box. It has fields for 'Name:' (set to 'Book_Genre_Clustering_Train') and 'Description:' (empty). Under 'Template:', it shows 'Python Producer' selected. At the bottom are 'Create' and 'Cancel' buttons, with a red box highlighting the 'Create' button.</p>
<p>You need to adjust the pipeline template. The pipeline loads data with the "Read File" operator. The data is passed to a Python operator, where the ML model is trained. The same Python-operator stores the model in the ML</p>	 <p>The screenshot shows the completed ML Pipeline graph. The flow starts with a 'Read File' operator, followed by a 'Python3' operator (where the ML model is trained). This is followed by a 'Submit Metrics' operator, another 'Python3' operator, and an 'Artifact Producer' operator. The pipeline then branches: one path goes through a 'Write File' operator and another 'Artifact Producer' operator; the other path goes through an 'Operators Complete' operator and a 'Graph Terminator' operator. Various status icons like '83%' are visible along the edges.</p>

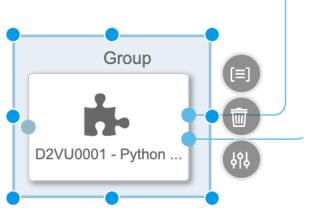
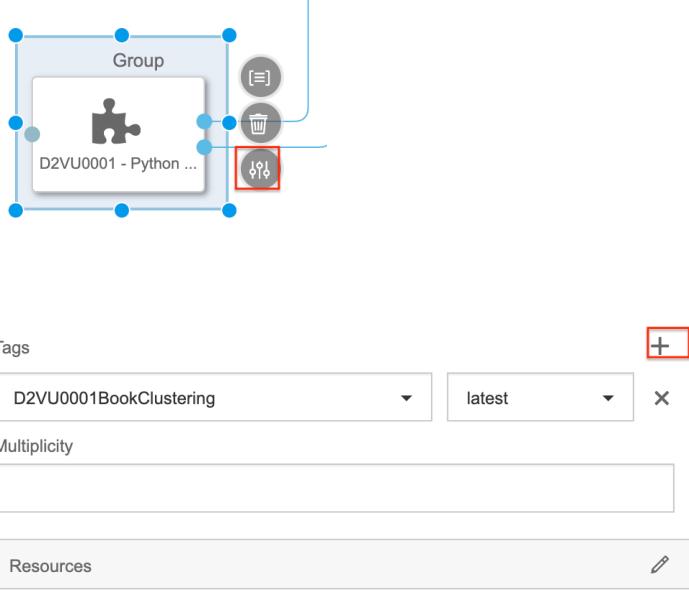
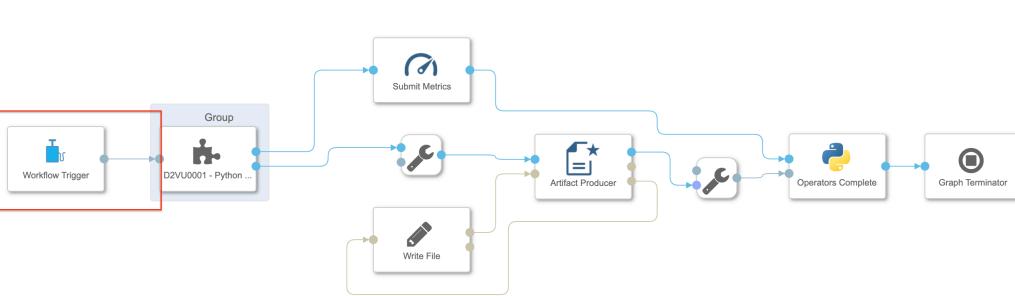
Explanation	Screenshot
<p>Scenario through the “Artifact Producer”. The Python-operator’s second output passes a model quality metric to the ML Scenario. Once both model and metric are saved, the pipeline’s execution is ended with the “Graph Terminator”.</p>	
<p>Since for us the input data are not stored in a file, but in HANA Cloud, we don’t need the Read File operator. We will adapt the custom python operator we built in the previous exercise.</p> <p>Insert the custom operator Python with HANA ML that you created in exercise 210 in the canvas. Click on the Add port symbol to add two output ports.</p> <p>Configure the ports with the parameter shown in the picture</p>	 <p>The screenshot shows a Python operator node named "D2VU0001 - Python ..." with a blue border. To its right are four circular icons: a double-headed arrow, a plus sign, a gear, and a trash can. The plus sign icon is highlighted with a red box. Below the node is a "Port details" dialog for the first output port. The dialog has fields for Name ("metrics"), Data Type ("Basic"), and message. Below it is another "Port details" dialog for the second output port, with Name ("modelBlob"), Data Type ("Basic"), and blob.</p>

Explanation	Screenshot
<p>In the Configuration panel, open the “Connection” configuration, set “Configuration type” to “Connection Management” and select from the drop-down menu your DWC connection</p>	
<p>Delete the Read File operator, and the Conversion operator. Replace the Python Operator with the custom operator that you have just configured. Your pipeline should look like in the picture.</p>	
<p>Next, adjust the Python code that analyzes the text data and trains the cluster model. Select the custom operator and click the Script option.</p>	
<p>The template code opens up. It shows how to pass the text analysis, model and metrics into the ML Scenario. <b>Carefully</b> copy and paste to replace the existing code with the code given here, so that we can operationalize the clustering model we developed in the</p>	<pre data-bbox="425 1474 1013 1938"> import hana_ml from hana_ml import dataframe import pandas as pd 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 ... import hana_ml.dataframe as dataframe df_hana = (conn.table('Book_Author_Dimension_View', schema=D2VUXXXX))  books= df_hana.select(['Book_ID','Book_Description']).collect() del df_hana books=books.dropna(axis=0,subset=['Book_Description'])  import nltk from nltk.corpus import stopwords </pre>

Explanation	Screenshot
<p>notebook. Adapt the highlighted line according to your own table and schema names.</p>	<pre> nltk.download("stopwords") stopwords=set(stopwords.words("english")) import regex as re  #Transform to lower case books["tokens"]=books["Book_Description"].apply(lambda x: x.lower()) #Remove punctuation books["tokens"]=books["tokens"].map(lambda x: re.sub("[.,!?:;'\(\)]", ' ', x)) #Remove stopwords books["tokens"]=books["tokens"].apply(lambda x: ''.join([t for t in x.split() if not t in stopwords])) # Remove short tokens books["tokens"]=books["tokens"].apply(lambda x:''.join([t for t in x.split() if len(t) &gt; 1])) #Remove extra spaces books["tokens"]=books["tokens"].map(lambda x: re.sub(' +', ' ', x)) # Remove duplicate tokens books["tokens"]=books["tokens"].apply(lambda x: ''.join(list(dict.fromkeys(x.split())))) books=books.drop_duplicates('tokens')  #download word embedding import gensim.downloader as gensim_api model = gensim_api.load("glove-wiki-gigaword-100") features=[] for i,book in books.iterrows():     tokens_features=[]     for word in book["tokens"].split():         try:             tokens_features.append(model[word])         except:             continue     features.append(np.mean(np.array(tokens_features),axis=0))  for i in range(100):     feature=f'_'+str(i)     books[feature]=[f[i] for f in features]  del features embedding=[f'_'+str(i) for i in range(100)]  #Fit Kmeans algorithm from sklearn.cluster import MiniBatchKMeans n_clus=10 km = MiniBatchKMeans(n_clusters = n_clus, batch_size=50 , random_state=42, max_iter=1000) y_kmeans = km.fit_predict(books[embedding])  # Silhouette from sklearn.metrics import silhouette_score score = silhouette_score(books[embedding], km.labels_)  # to send metrics to the Submit Metrics operator metrics_dict = {"SILHOUETTE": str(np.round_(score,2))}  # send the metrics to the output port - Submit Metrics operator will use this to persist the metrics api.send("metrics", api.Message(metrics_dict))  # create &amp; send the model blob to the output port - Artifact Producer operator will use this to persist the model and create an artifact ID import pickle model_blob = pickle.dumps(km) api.send("modelBlob", model_blob)  api.set_port_callback("trigger", on_input) </pre>
<p>Close the Script-window, then click “Save” in the menu bar.</p>	

Explanation	Screenshot
<p>You need to create a Docker image for the custom python operator. This gives the flexibility to leverage virtually any Python library. The Docker file installs the necessary libraries. You find the docker files by clicking into the “Repository” tab on the left, then right-click the “<b>dockerfiles</b>” folder and select “Create Docker File”.</p>	
<p>Give the dockerfile a name of the form D2VUXXXXBookClustering</p>	 <p>Create Docker File</p> <p>Name: DVU0001BookClustering</p> <p>OK Cancel</p>
<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.</p>	<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 sklearn RUN pip3.6 install --user hana_ml RUN pip3.6 install --user regex RUN pip3.6 install --user nltk RUN pip3.6 install --user gensim</pre>

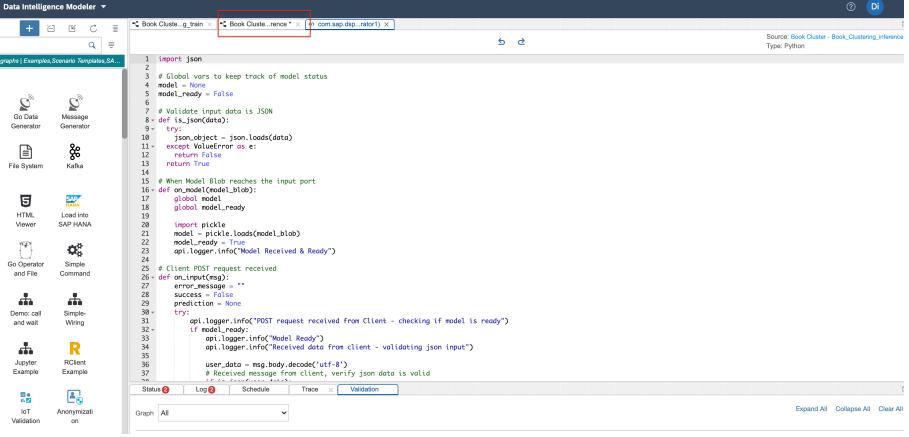
Explanation	Screenshot
<p>Open the Configuration panel for the Docker File with the icon on the top-right hand corner.</p>	 <p>The screenshot shows a window titled "Configuration". At the top right are three icons: a play button, a save disk, and a gear/cogwheel. The gear/cogwheel icon is highlighted with a red box. Below the title is a section titled "Tags" with a plus sign. Underneath it, the text "No tags defined." is displayed.</p>
<p>Assign a tag to the docker image</p>	 <p>The screenshot shows the same "Configuration" window. In the "Tags" section, there are two input fields: one containing "D2VU0001BookClustering" and another containing "latest". Both fields have an "X" icon at their right ends.</p>
<p>Now save the Docker file and 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.</p>	 <p>The screenshot shows the "Configuration" window again. This time, the play button icon at the top left is highlighted with a red box. The other two icons (save and configuration) are visible but not highlighted.</p>

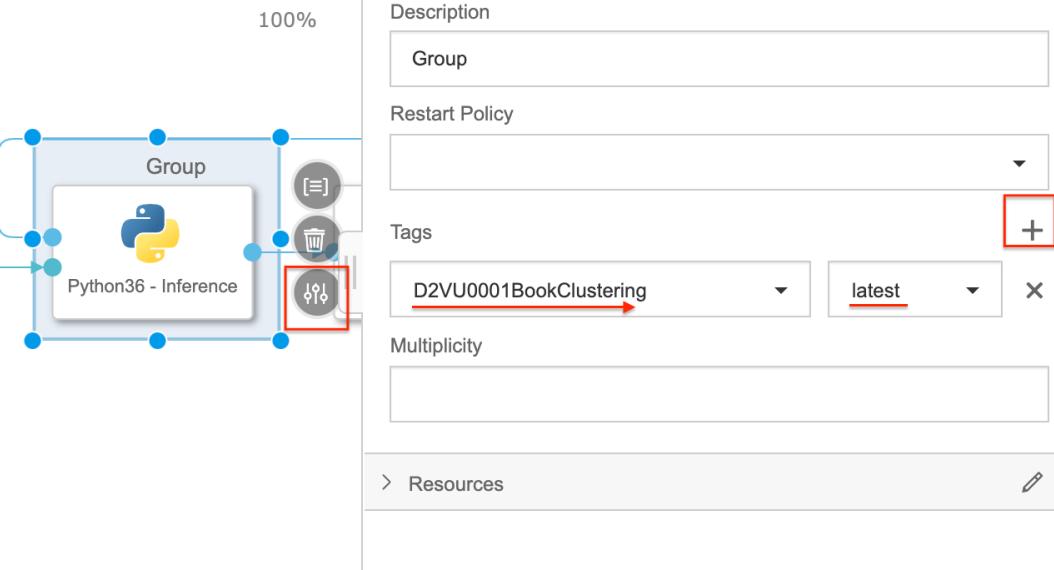
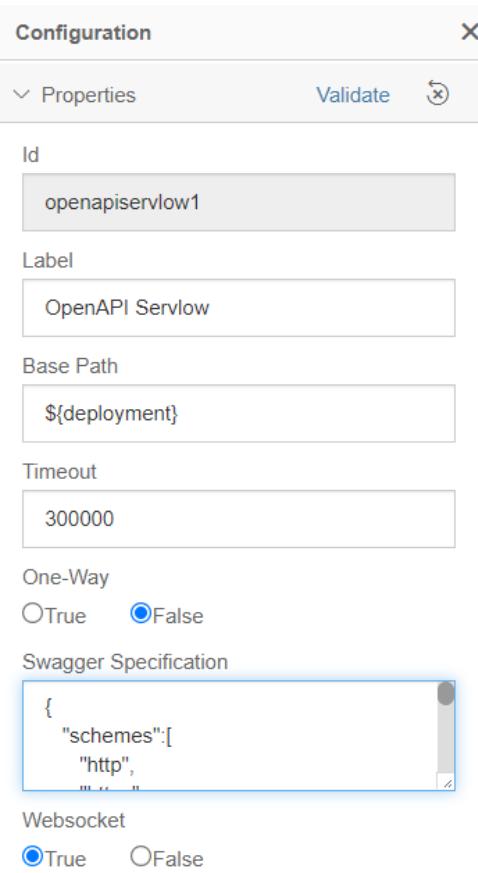
Explanation	Screenshot
<p>Now you need to configure the custom operator, which trains the model, to use this Docker image. Go back to the graphical pipeline and right-click the custom operator and select "Group".</p>	
<p>You specify which Docker image should be used. Select the group which surrounds the python custom operator. In the group's Configuration select the docker image you have built.</p>	
<p>Select the Workflow Trigger operator in the operators panel. Add it to the pipeline and connect it to the trigger port of the custom operator. Save your pipeline</p>	
<p>The pipeline is now complete and you can run it. Go back to the ML Scenario.</p>	

Explanation	Screenshot				
Select the pipeline in the ML Scenario and click the “Execute” button on the right.					
Click “Step 3” and then “Step 4” to skip the optional steps until you get to the “Enter your Pipeline Parameters”. Set “newArtifactName” Value to “kmeans”. Click Save. The trained model will be saved under this name.	<p>4. Pipeline Parameters</p> <p>Enter your Pipeline Parameters.</p> <table border="1" data-bbox="453 508 1498 587"> <thead> <tr> <th data-bbox="453 508 1302 536">Key</th> <th data-bbox="1302 508 1498 536">Value</th> </tr> </thead> <tbody> <tr> <td data-bbox="453 536 1302 587">newArtifactName*</td> <td data-bbox="1302 536 1498 587">kmeans</td> </tr> </tbody> </table> <p><b>Save</b></p>	Key	Value	newArtifactName*	kmeans
Key	Value				
newArtifactName*	kmeans				
Wait a few minutes until the pipeline executes and completes. The metrics section shows the trained model's silhouette metric. The model itself was saved successfully under the name “kmeans”. The model has a Technical Identifier.	<p>Status: <b>Completed</b>      Created On: 9/14/21, 10:19 AM      Last Synchronized On: 9/14/21, 10:22 AM Created By: stuart</p> <p>Progress Flow   Configuration   Metrics (1)   Models and Datasets (1)</p> <p>Progress Flow</p> <pre> graph LR     A[Text Clustering Train_00] --&gt; B[newArtifactName: Cluster_Model_01]     B --&gt; C[Execution 9/14/21, 10:19 AM]     C --&gt; D[Cluster_Model_01]     E[SILHOUETTE: 0.04 start: 2021-09...] --- C   </pre>				

Explanation	Screenshot
<p>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. Name the pipeline “Book Clustering Consumer”. Select the template “Python Consumer”. This template contains a pipeline that provides a REST-API.</p>	<p>Create Pipeline</p> <p>Name: * <input type="text" value="Book_Clustering_inference"/></p> <p>Description: <input type="text"/></p> <p>Template: <input type="text" value="Python Consumer"/></p> <p><b>Create</b> <b>Cancel</b></p>
<p>The “OpenAPI Servlow” operator provides the REST-API. The “Artifact Consumer” loads the trained model from your ML scenario. The “Python36 – Inference” operator ties the two operators together. It receives the input from the REST-API call (here the user’s text input book description) and uses the loaded model to assign the cluster, which is then returned by the “OpenAPI Servlow” to the client, which</p>	<pre> graph LR     A[Submit Artifact Name] --&gt; B{ }     B --&gt; C[Artifact Consumer]     C --&gt; D{ }     D --&gt; E[Read File]     E --&gt; F{ }     F --&gt; G[Python36 - Inference]     G --&gt; H[OpenAPI Servlow]   </pre>

Explanation	Screenshot
had called the REST-API.	
You only need to update the script of the Python3.6 Inference operator. Select the Python3.6 operator in the pipeline and click on the script icon	
<b>Carefully copy and paste to replace the whole code with the code given here.</b> Close the editor window.	<pre> import json  # 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  # When Model Blob reaches the input port def on_model(model_blob):     global model     global model_ready      import pickle     model = pickle.loads(model_blob)     model_ready = True     api.logger.info("Model Received &amp; Ready")  # 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                 # load new data                 books = json.loads(user_data)[\"book_description\"]                  # preprocessing                 import nltk                 from nltk.corpus import stopwords                 nltk.download("stopwords")                 stopwords=set(stopwords.words("english"))                 import regex as re                  #Transform to lower case                 books=[x.lower() for x in books ]                 #Remove punctuation                 books=[ re.sub("[.,!?.\"\\()]", ' ', x) for x in books ]                 #Remove stopwords                 books=[ ' '.join([t for t in x.split() if not t in stopwords]) for x in books]                 # Remove short tokens                 books=[ ' '.join([t for t in x.split() if len(t) &gt; 1]) for x in books]                 #Remove extra spaces                 books=[ re.sub(' +', ' ', x) for x in books]                 # Remove duplicate tokens                 books=[ ' '.join(list(dict.fromkeys(x.split()))) for x in books]                  # GloVe Vectorization     </pre>

Explanation	Screenshot
<pre> import gensim.downloader as gensim_api word_embedding = gensim_api.load("glove-wiki-gigaword-100") # load pre-trained word-vectors from gensim-data import numpy as np features=[] for book in books:     tokens_features=[]     for word in book.split():         try:             tokens_features.append(word_embedding[word])         except:             continue     features.append(np.mean(np.array(tokens_features),axis=0))  # deploy cluster model predictions = model.predict( features) 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({'Cluster': predictions.tolist()}) 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("model", on_model) api.set_port_callback("input", on_input) </pre>	
<p>Go back to the pipeline tab and save the pipeline</p> 	

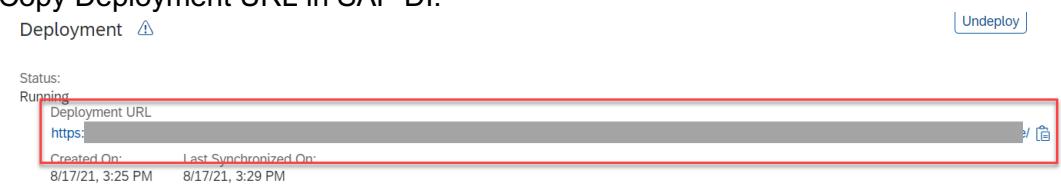
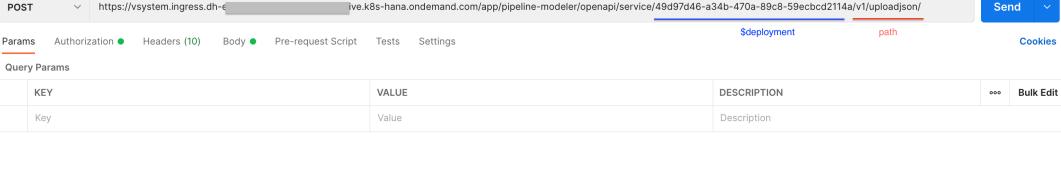
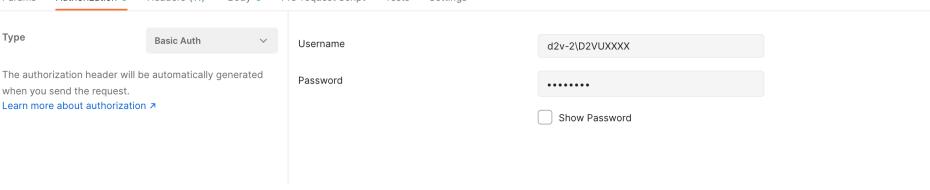
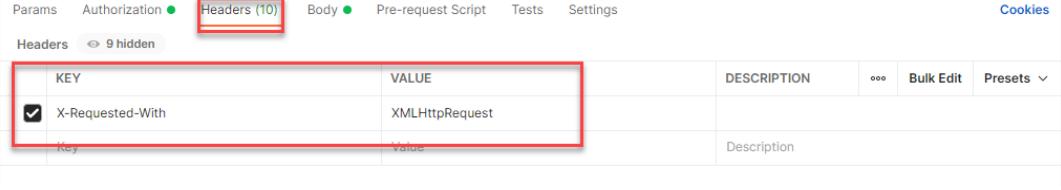
Explanation	Screenshot
<p>Finally, you need to assign the Docker image to the “Python36 – Inference” operator. As before, right-click the operator and select “Group”. Add the tag of your docker image Save the changes.</p>	 <p>Description: Group</p> <p>Restart Policy:</p> <p>Tags: D2VU0001BookClustering <span style="border: 1px solid red; padding: 2px;">+</span> latest</p> <p>Multiplicity:</p> <p>&gt; Resources</p>
<p>Click on OpenAPIServlow and have a look at the configuration</p>	 <p>Configuration</p> <p>Properties Validate</p> <p>Id: openapiservlow1</p> <p>Label: OpenAPI Servlow</p> <p>Base Path: \${deployment}</p> <p>Timeout: 300000</p> <p>One-Way: <input type="radio"/> True <input checked="" type="radio"/> False</p> <p>Swagger Specification:</p> <pre>{   "schemes": [     "http",     "https"   ] }</pre> <p>Websocket: <input checked="" type="radio"/> True <input type="radio"/> False</p>

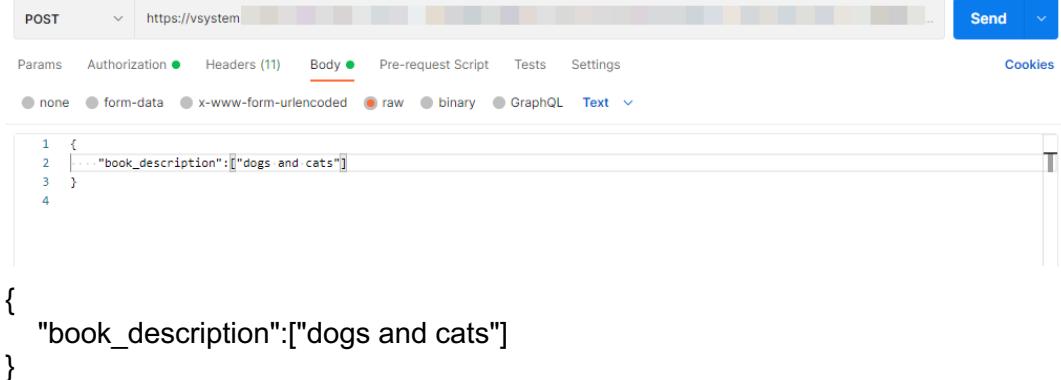
Explanation	Screenshot
	 <p>Max Concurrency 32</p> <p>Max Accepted 128</p>
<p>Notice in particular the content of the Swagger Specification.</p>	<pre>{   "schemes":[     "http",     "https"   ],   "swagger":"2.0",   "info":{      "description":"This is an example of using the OpenAPI Servlow to carry 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"           }         }       }     }   } }</pre>

Explanation	Screenshot								
	<pre>     "500":{       "description":"Error during upload of json"     }   },   "definitions":{},   "securityDefinitions":{     "UserSecurity":{       "type":"basic"     }   } } </pre>								
Go back to the ML Scenario. Now deploy the new pipeline. Select the pipeline and click “Deploy”.	<p>Pipelines (2)</p> <table border="1"> <thead> <tr> <th>Name</th> <th>Description</th> <th>Template</th> <th>Created On</th> </tr> </thead> <tbody> <tr> <td>Genre Clustering Serving_01</td> <td>REST-API for real time inference</td> <td>Python Consumer</td> <td>9/14/21, 10:28 AM</td> </tr> </tbody> </table>	Name	Description	Template	Created On	Genre Clustering Serving_01	REST-API for real time inference	Python Consumer	9/14/21, 10:28 AM
Name	Description	Template	Created On						
Genre Clustering Serving_01	REST-API for real time inference	Python Consumer	9/14/21, 10:28 AM						
Click through the screens until you can select the trained model from the drop-down. Click “Save”.	<p>Deploy Pipeline</p> <ol style="list-style-type: none"> <li>Pipeline</li> <li>Configuration Description (Optional)</li> <li>Previous Configuration (Optional)</li> <li>Pipeline Parameters</li> </ol> <p>4. Pipeline Parameters</p> <p>Enter your Pipeline Parameters.</p> <table border="1"> <tr> <td>Key</td> <td>Value</td> </tr> <tr> <td>ARTIFACT:MODEL*</td> <td>Cluster_Model_01 (9/14/21, 10:21 AM)</td> </tr> </table> <p><b>Save</b></p>	Key	Value	ARTIFACT:MODEL*	Cluster_Model_01 (9/14/21, 10:21 AM)				
Key	Value								
ARTIFACT:MODEL*	Cluster_Model_01 (9/14/21, 10:21 AM)								
After a few minutes the pipeline will start running.	<p>Deployment</p> <p>Status: Running Deployment URL: <a href="https://vsystem.ingress">https://vsystem.ingress</a></p> <p>Created On: 9/14/21, 10:34 AM La: 9/1 Created By: stuart</p> <p>Progress Flow Configuration</p> <p>Progress Flow</p> <pre> graph LR     Pipeline((Pipeline)) --&gt; Configuration((Configuration))     Configuration --&gt; Deployment((Deployment))     subgraph ProgressFlow [Progress Flow]         direction LR         Pipeline --- Configuration         Configuration --- Deployment     end </pre> <p>Pipeline: Genre Clustering Serving_01 (9/14/21, 10:28 AM)</p> <p>Configuration: ARTIFACT:MODEL: 765e9eee-276b-9/14/21, 10:34 AM</p> <p>Deployment: https://vsystem.ingress.dh-vb7neejn.dhaha... (Running) 9/14/21, 10:34 AM</p>								

## STEP 3 – USE YOUR CLUSTER MODEL

Now that you have deployed your model, you can use it for real-time cluster assignment. 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 <b>v1/uploadjson/</b>, 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>	

Explanation	Screenshot
<p>Finally, pass the input data to the REST-API. Select the “Body” tab, choose “raw” and enter the syntax given here.</p> <pre data-bbox="425 255 1486 635">{   "book_description":["dogs and cats"] }</pre>	
<p>Press “Send” and after a few minutes you will see the genre prediction that comes from SAP Data Intelligence. Try the REST-API with different text to see how the cluster allocations change.</p>	 <pre data-bbox="474 846 768 1015">1  { 2    "Cluster": [ 3      9 4    ] 5  }</pre>
<p>You have now completed the exercise.</p>	

