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

THE BEST RUN SAP®

Table of Contents

DISCLAIMER

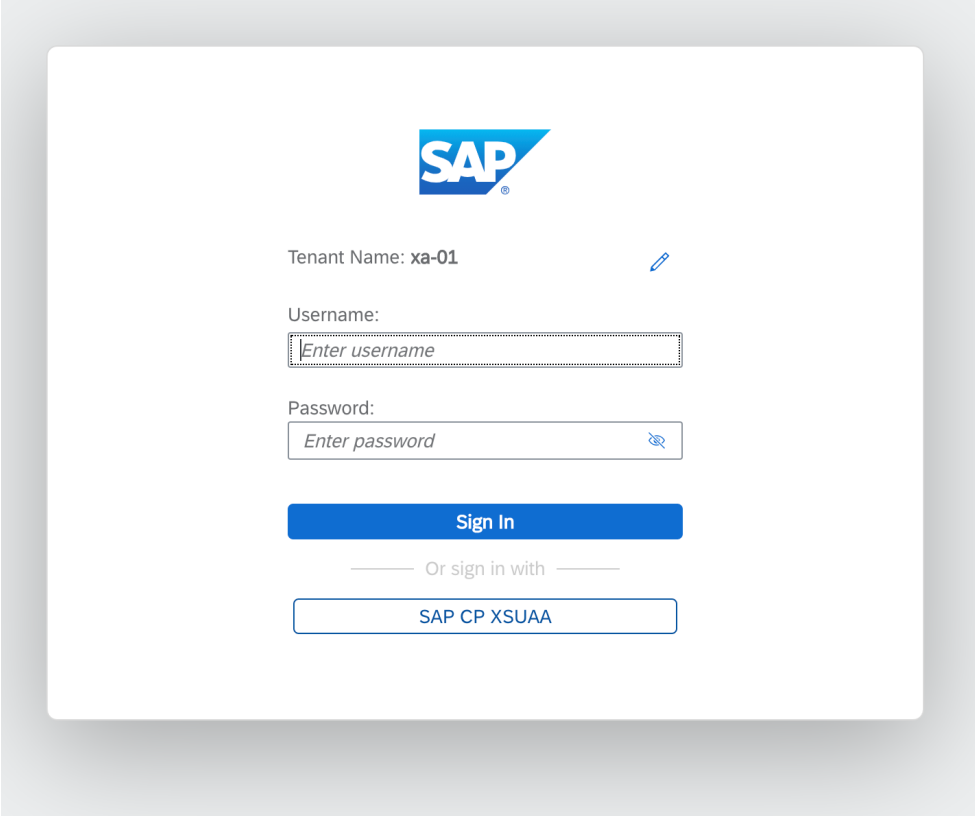
OBJECTIVE

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

SCENARIO

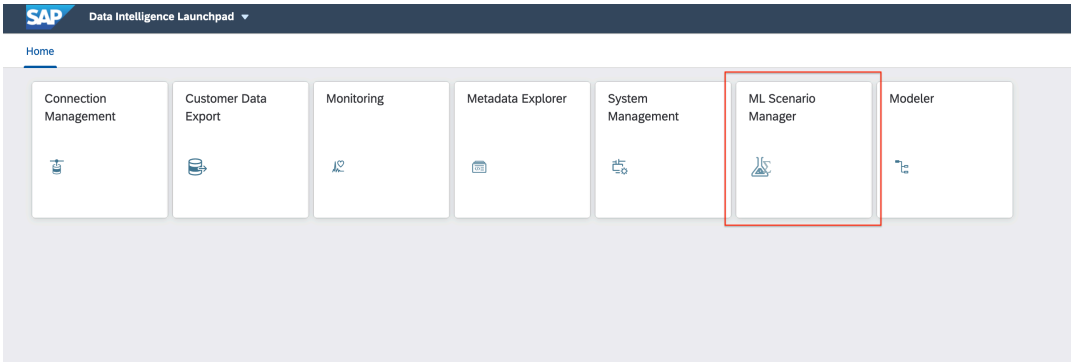
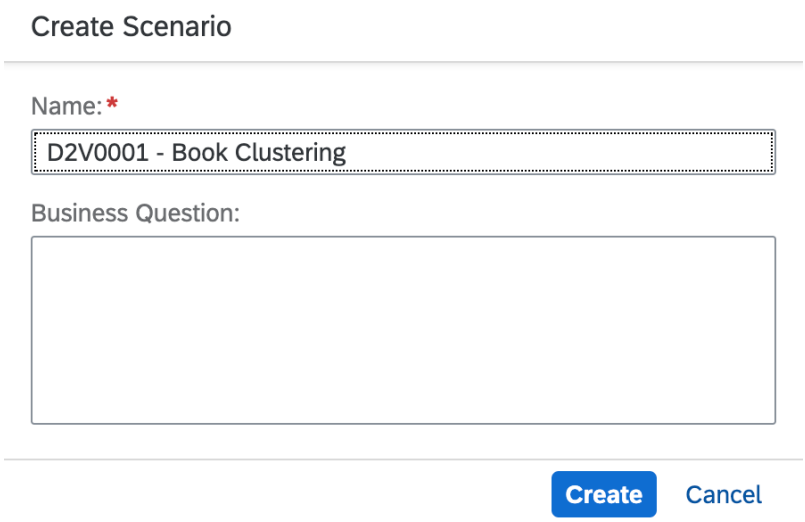
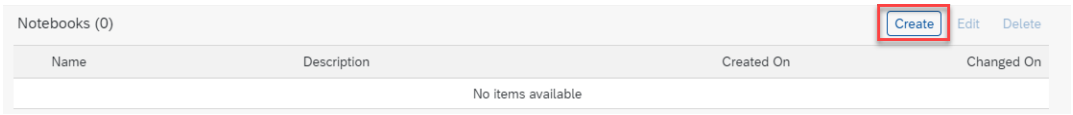
Books are grouped together in a bookshop based on their similarity, so that customers browsing for a book will find lots of similar books on the same shelf. This exercise analyzes the book description data using Python text mining algorithms and then uses this information to assign each book to a cluster. 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.

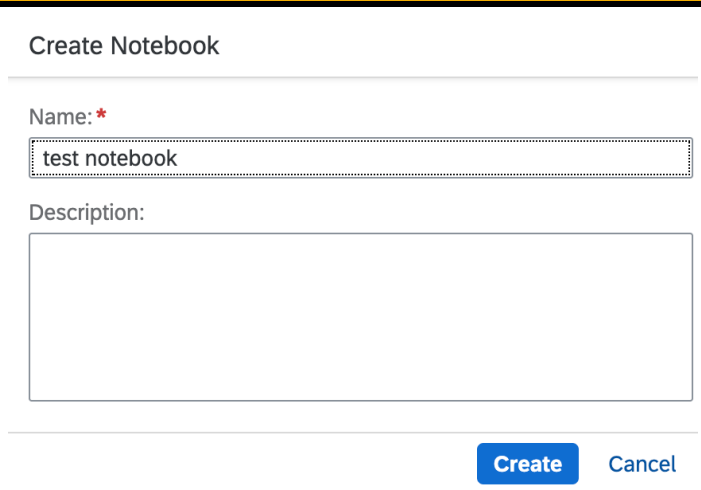
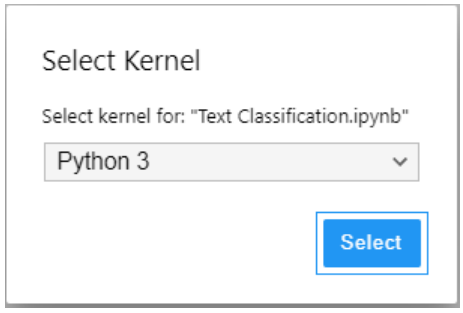
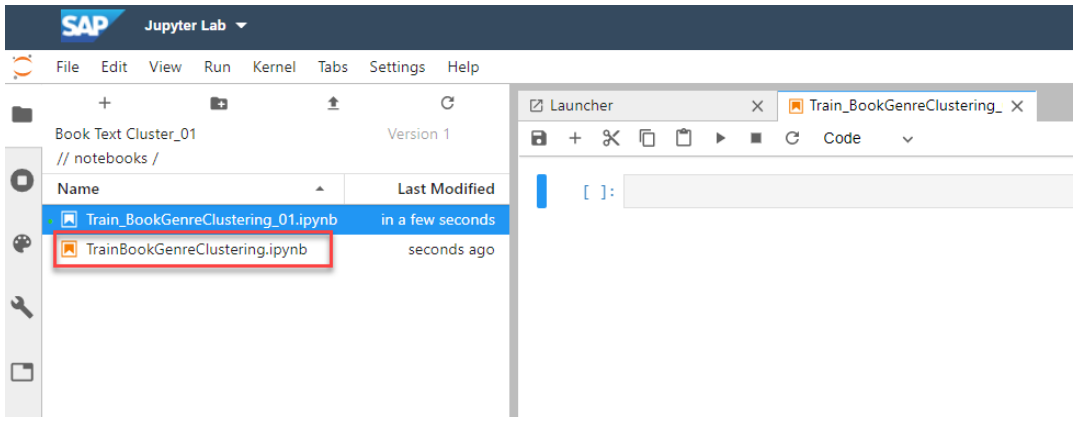
ENVIRONMENT ACCESS

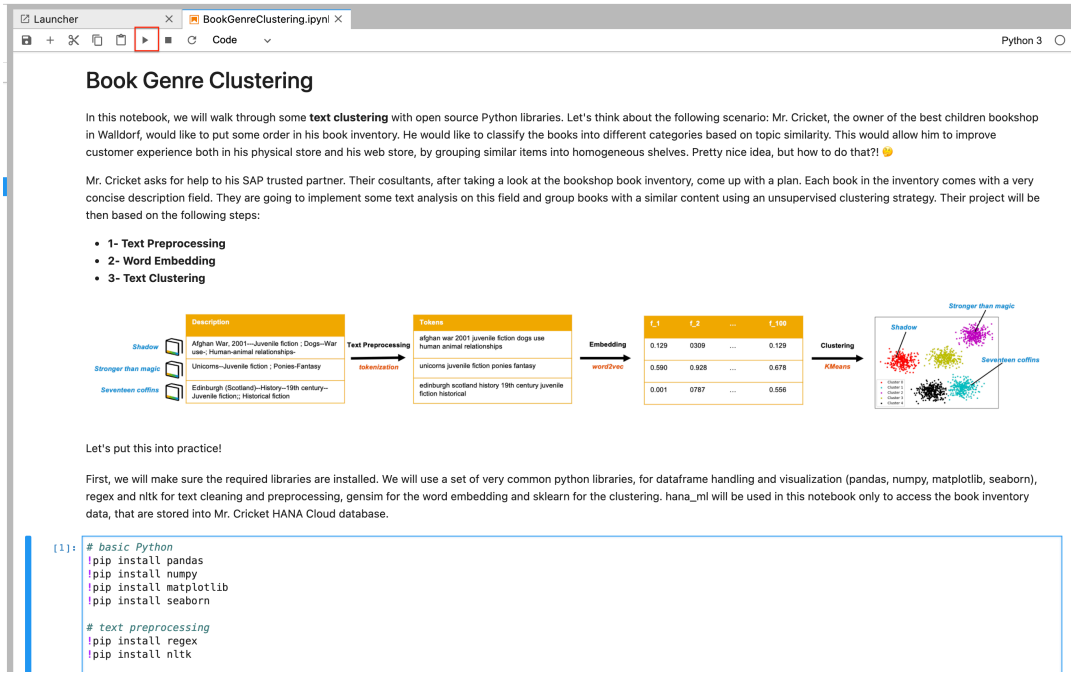
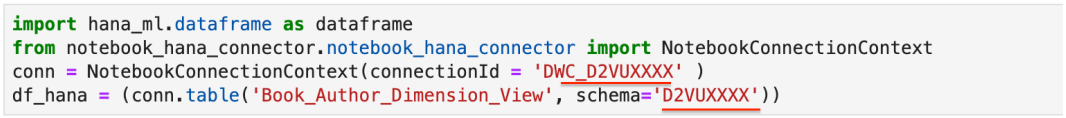
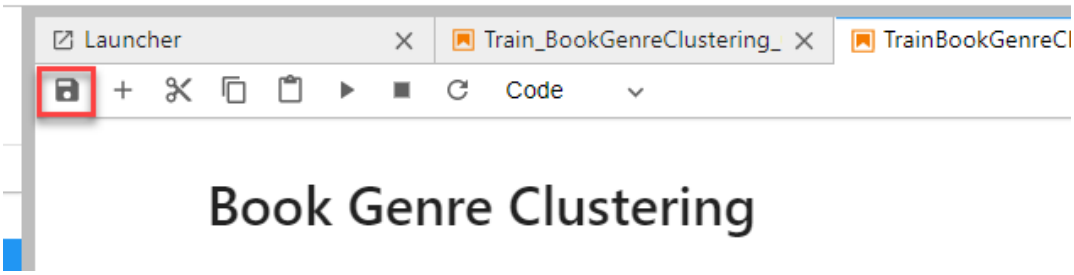
Explanation	Screenshot
Login to your SAP Data Intelligence tenant using the tenant name, username and password assigned to you	

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.

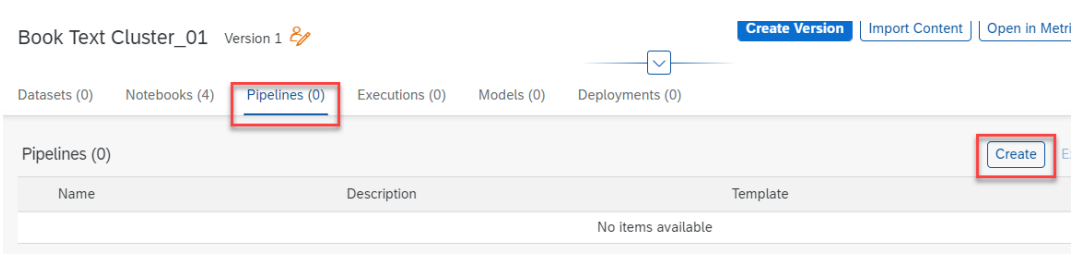
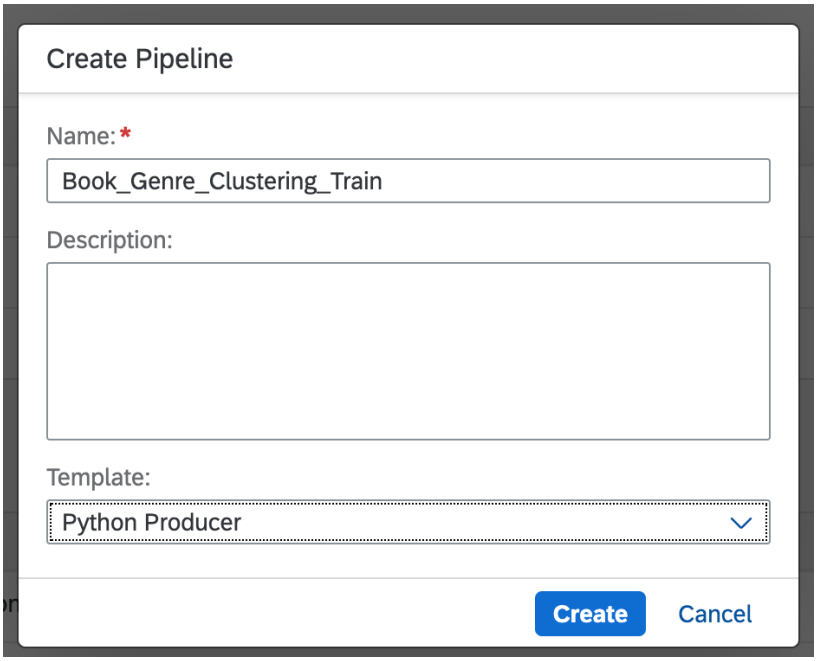
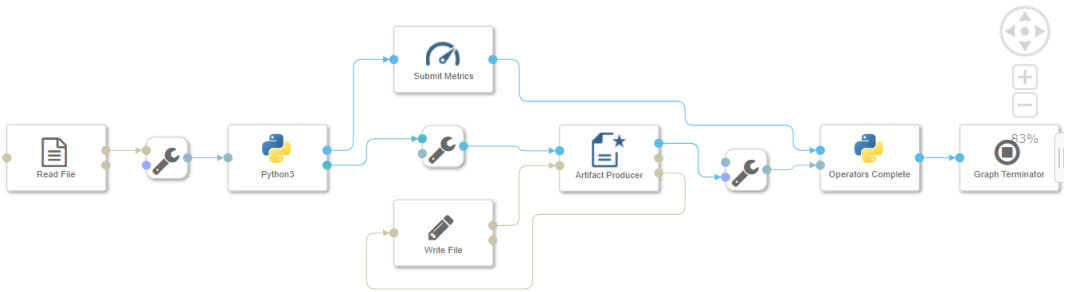
Explanation	Screenshot
Click to open ML Scenario Manager	
Click the Create button. Create a new scenario. Name the scenario according to your username: "D2VUXXXX Book Clustering".	
In the Notebooks section, click Create to create a new notebook.	

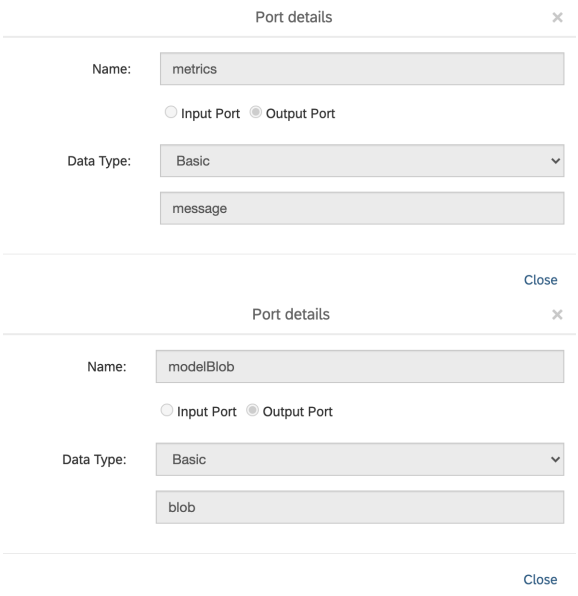
Explanation	Screenshot
Name the notebook with a name of your choice	
Select the Python 3 Kernel option.	
Use the Upload Files function at the top left of the console to upload the Python notebook that we prepared for the exercise. Upload file **BookGenreClustering.ipynb,** you will find it in the course github repository.	 Select file BookGenreClustering.ipynb

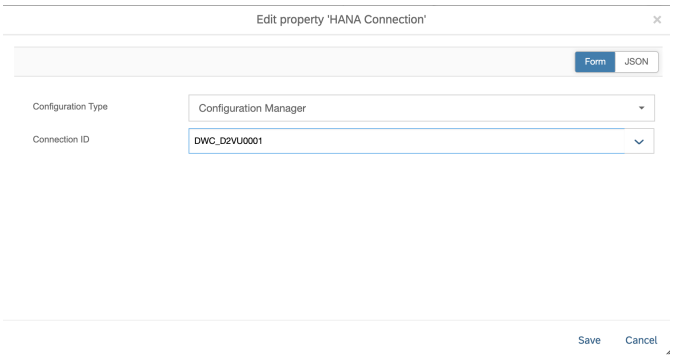
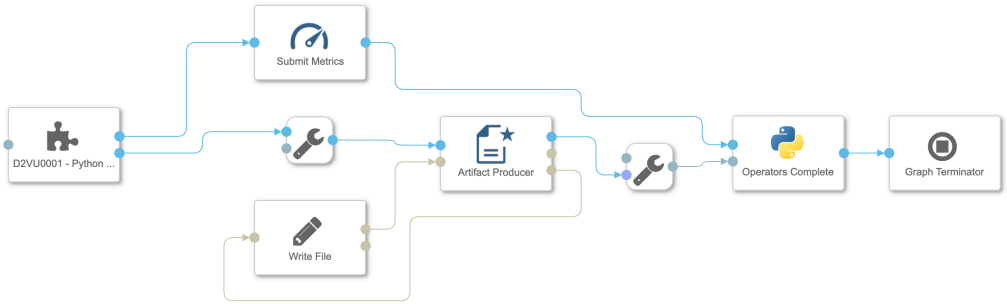
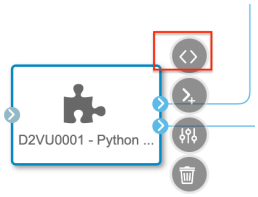
Explanation	Screenshot
Double click on the notebook that is uploaded. Step through the code to check it works correctly. Highlight each cell in sequence and click the arrow button to run the selected cell and advance.Note that some of the steps can take a few minutes to complete. Analyze the results of your data analysis and understand the text analysis and cluster model results.	
Adapt the notebook according to your own DWC connection and schema	
Once you have stepped through to the end of the notebook, and there are no errors, save the notebook.	

STEP 2 – BUILD MODEL PIPELINES

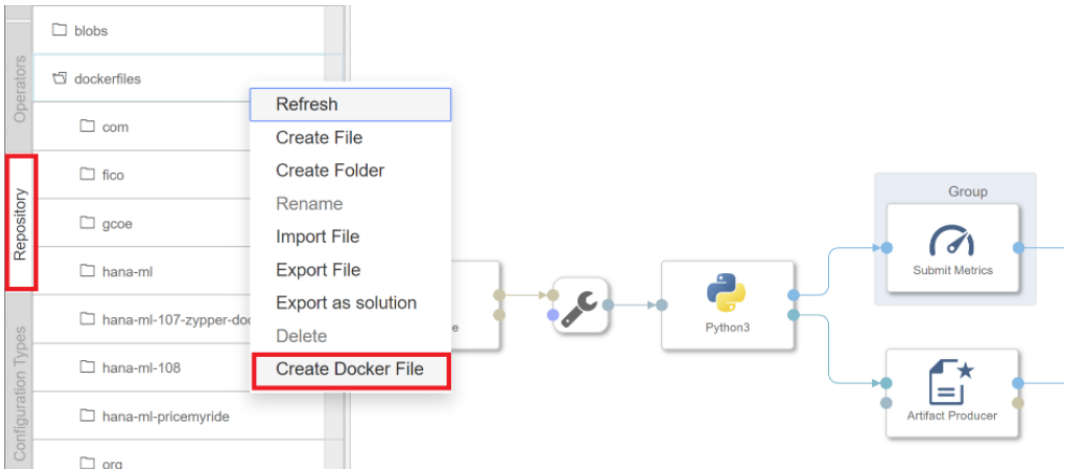
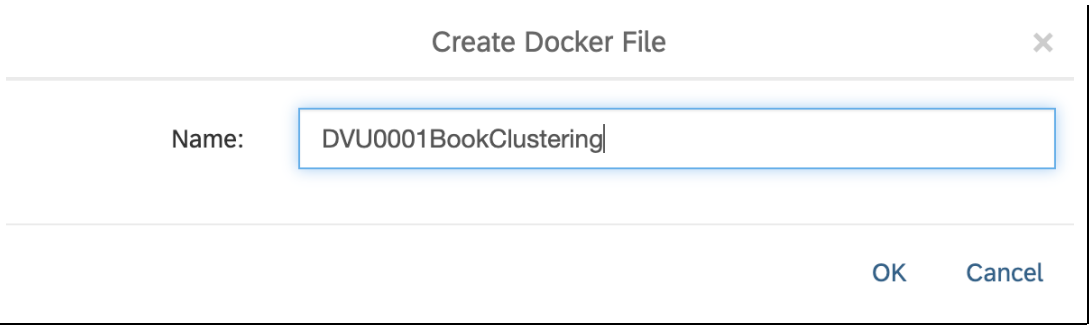
Model pipelines are used to operationalize the clustering model. Now that you have prepared the data, identify the best algorithm to use and which hyper-parameters work best, you want to take the model to production. You will build two pipelines. The first one is used to automate the model training,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

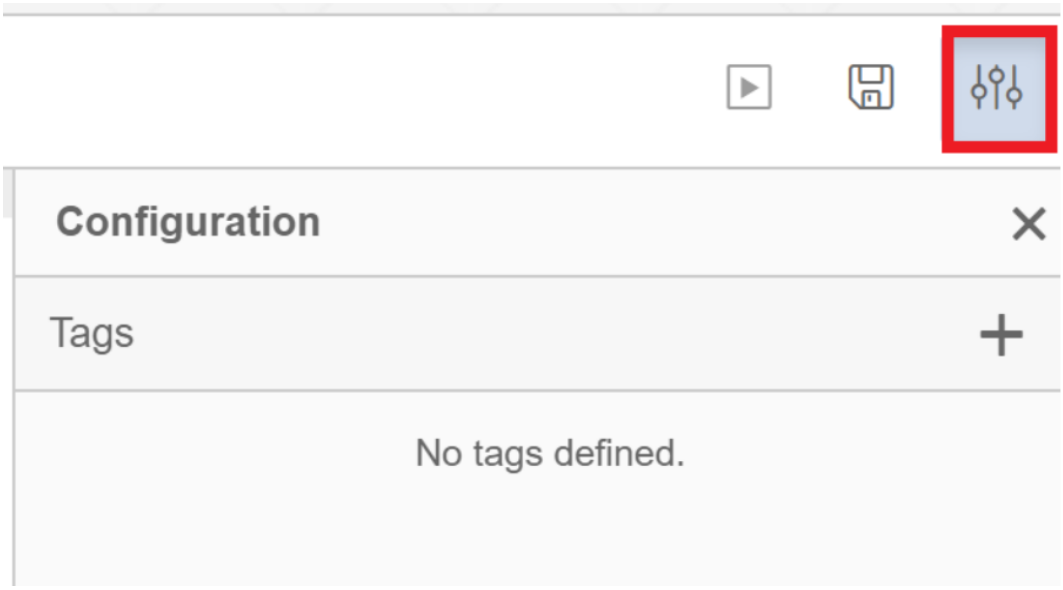
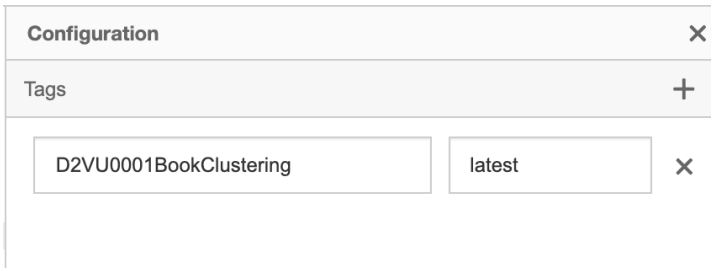
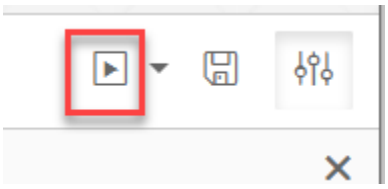
Explanation	Screenshot
To create the graphical pipeline to retrain the model, go to your ML Scenario's main page, select the "Pipelines" tab and click Create.	
Name the pipeline "Text Clustering Train" and select the "Python Producer" template. Click Create.	
You need to adjust the pipeline template. 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	

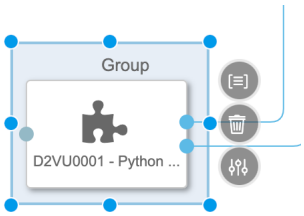
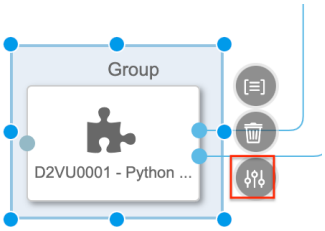
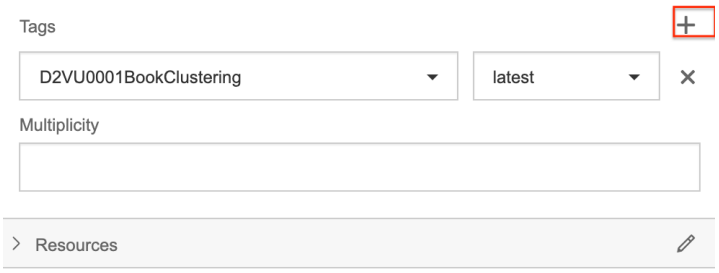
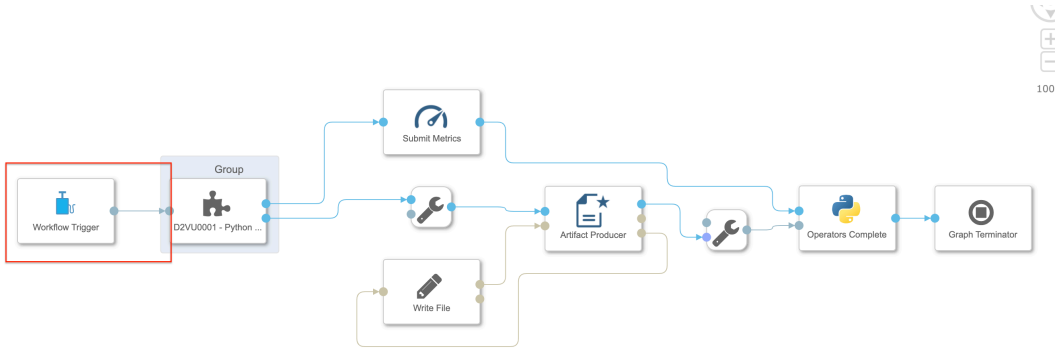
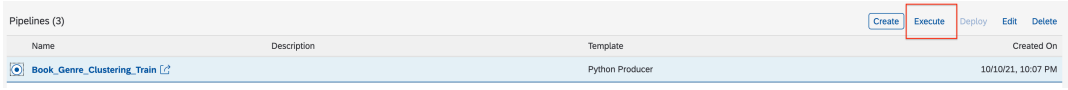
Explanation	Screenshot
Scenario through the “Artifact Producer”. The Python-operator’s second output passes a model quality metric to the ML Scenario. Once both model and metric are saved, the pipeline’s execution is ended with the “Graph Terminator”.	
Since for us the input data are not stored in a 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. 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. Configure the ports with the parameter shown in the picture	 

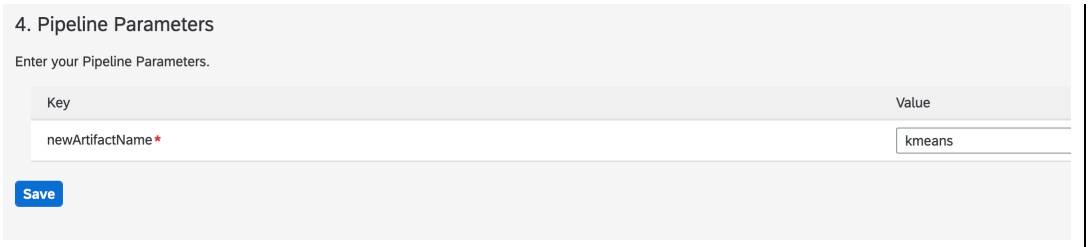
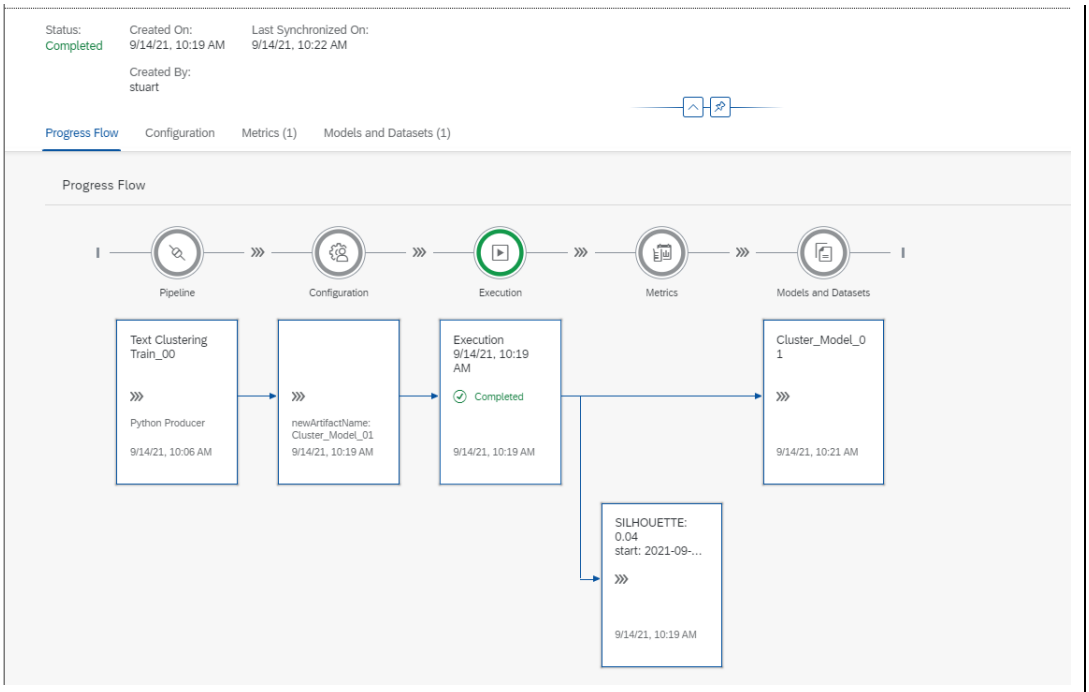
Explanation	Screenshot
In the Configuration panel, open the “Connection” configuration, set “Configuration type” to “Connection Management” and select from the drop -down menu your DWC connection	
Delete the Read File operator, and the Conversion operator. Replace the Python Operator with the custom operator that you have just configured. Your pipeline should look like in the picture.	
Next, adjust the Python code that analyzes the text data and trains the cluster model. Select the custom operator and click the Script option.	
The template code opens up. It shows how to pass the text analysis, model and metrics into the ML Scenario. **Carefully** 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	``` 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 ```

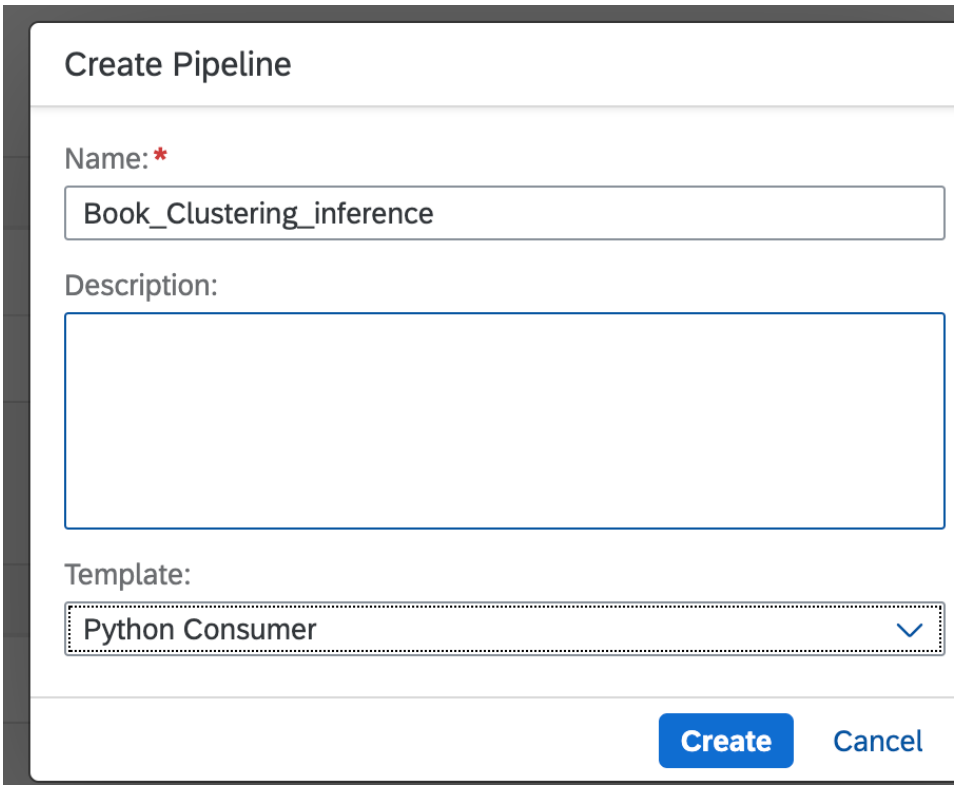
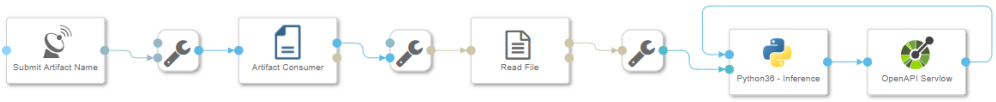
Explanation	Screenshot
notebook. Adapt the highlighted line according to your own table and schema names.	nltk.download("stopwords") stopwords=set(stopwords.words("english")) import regex as re #Transform to lower case books['tokens']=books['Book_Description'].apply(lambda x: x.lower()) #Remove punctuation 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) > 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 & 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)
Close the Script-window, then click “Save” in the menu bar.	

Explanation	Screenshot
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 “**dockerfiles**” folder and select “Create Docker File”.	
Give the dockerfile a name of the form D2VUXXXXBookClustering	
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 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

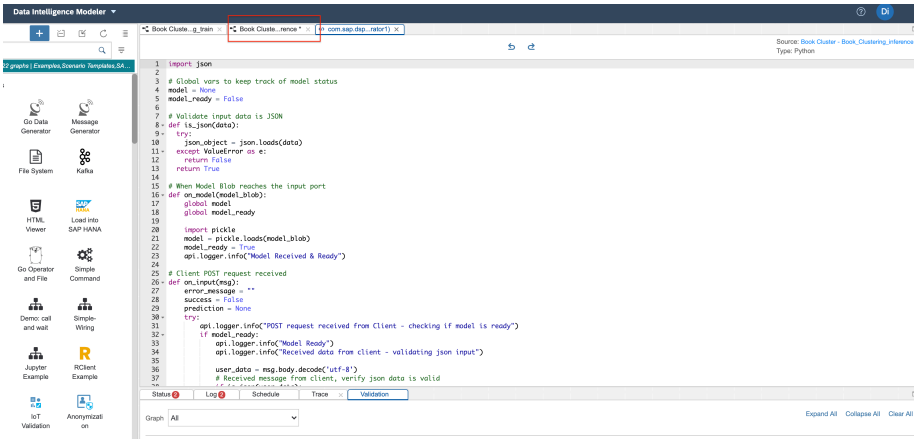
Explanation	Screenshot
Open the Configuration panel for the Docker File with the icon on the top-right hand corner.	
Assign a tag to the docker image	
Now save the Docker file and click the “Build” icon to start building the Docker image. Wait a few minutes and you should receive a confirmation that the build completed successfully.	

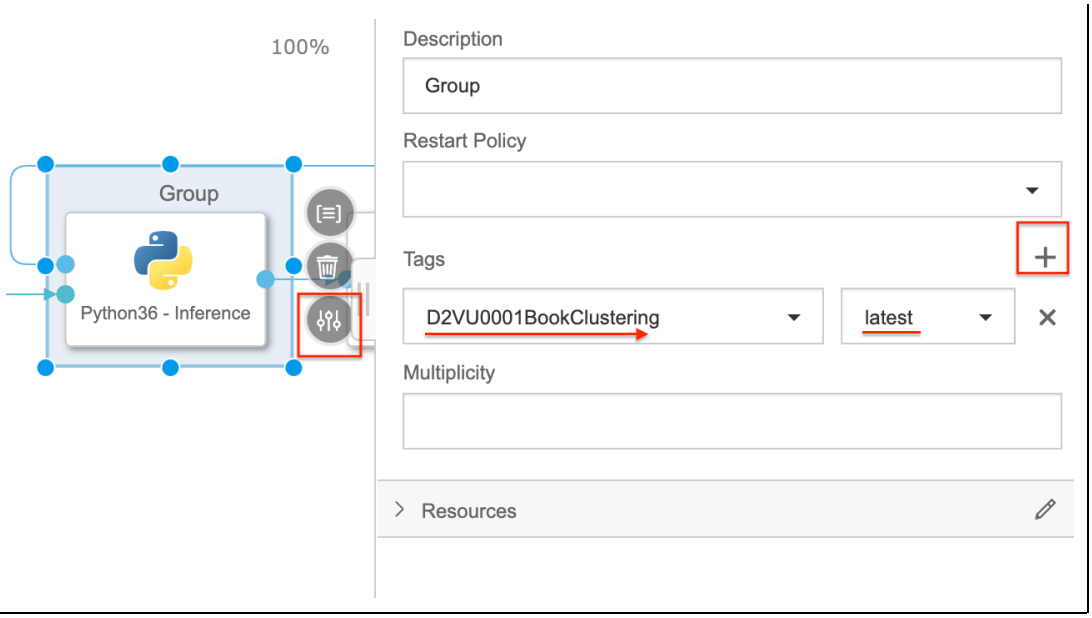
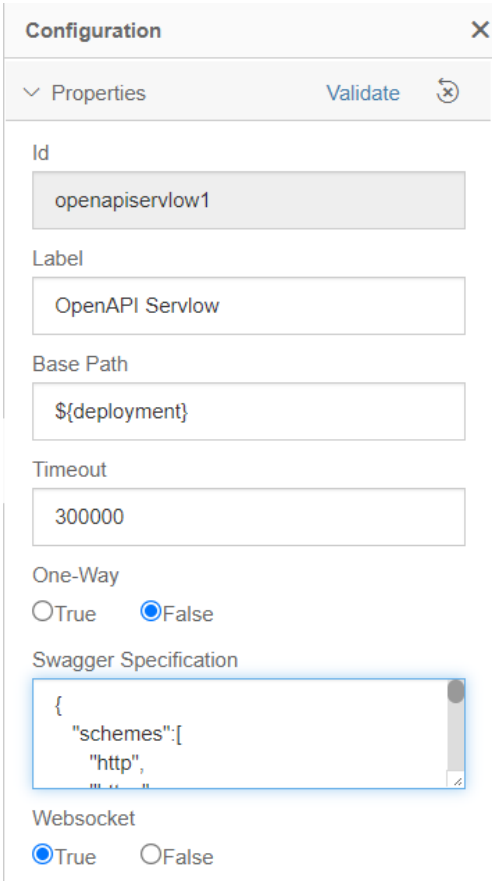
Explanation	Screenshot
Now you need to configure the custom operator, which trains the model, to use this Docker image. Go back to the graphical pipeline and right-click the custom operator and select “Group”.	
You specify which Docker image should be used. Select the group which surrounds the python custom operator. In the group’s Configuration select the docker image you have built.	 
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	
The pipeline is now complete and you can run it. Go back to the ML Scenario.	

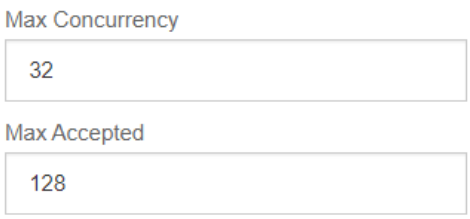
Explanation	Screenshot
Select the pipeline in the ML Scenario and click the “Execute” button on the right.	
Click “Step 3” and then “Step 4” to skip the optional steps until you get to the “Enter your Pipeline Parameters”. Set “newArtifactName” Value to “kmeans”. Click Save.The trained model will be saved under this name.	
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.	

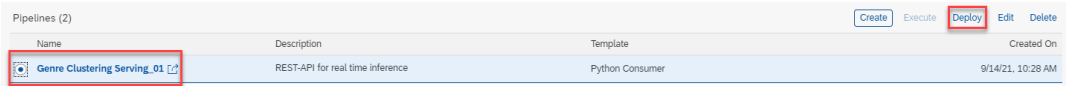
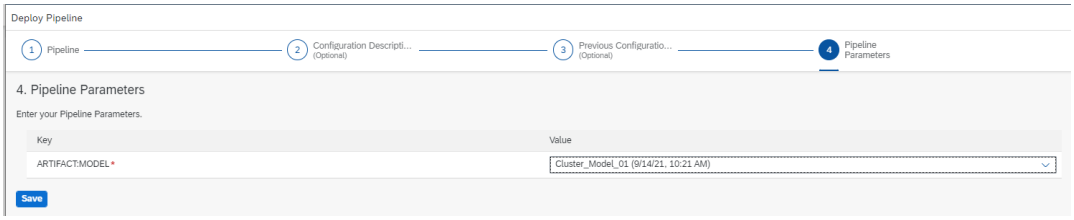
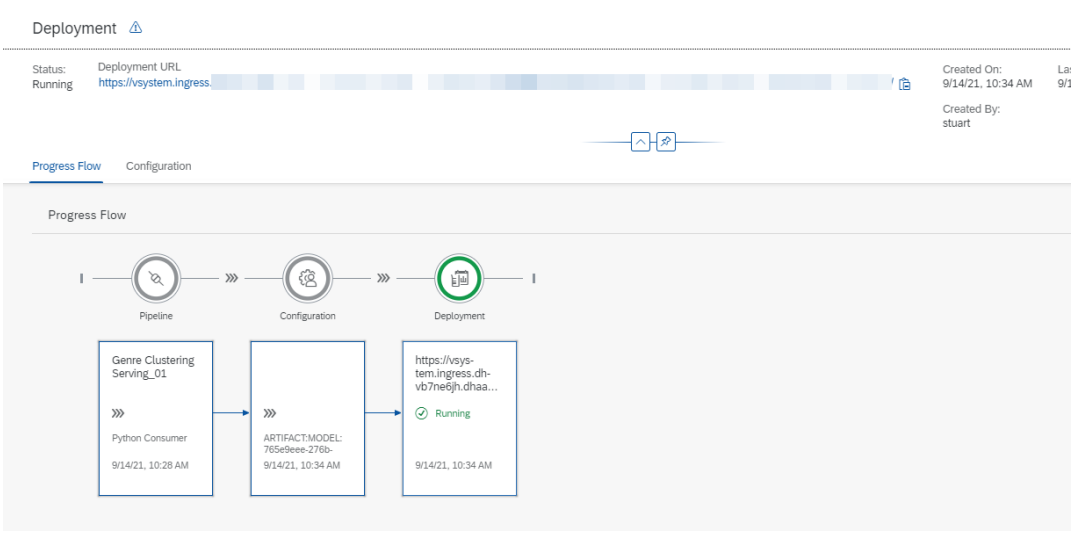
Explanation	Screenshot
You will now use the model for real-time inference with REST-API.Go back to the main page of your ML Scenario and create a second pipeline. This pipeline will provide the REST-API to obtain predictions in real-time. Name the pipeline “Book Clustering Consumer”. Select the template “Python Consumer”. This template contains a pipeline that provides a REST-API.	
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	

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	
Carefully copy and paste to replace the whole code with the code given here. Close the editor window.	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 & 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) > 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

Explanation	Screenshot
	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)
Go back to the pipeline tab and save the pipeline	

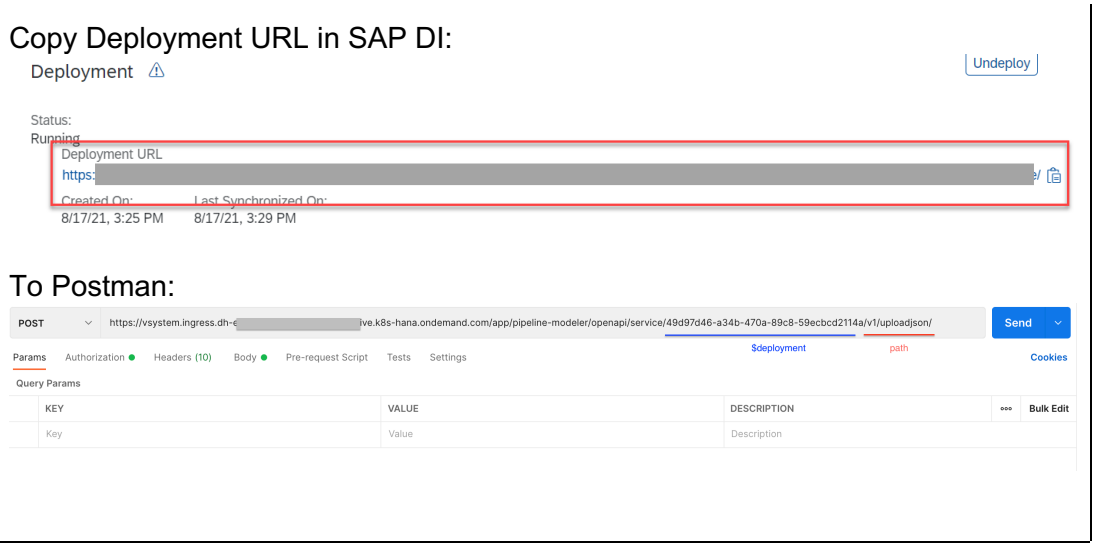
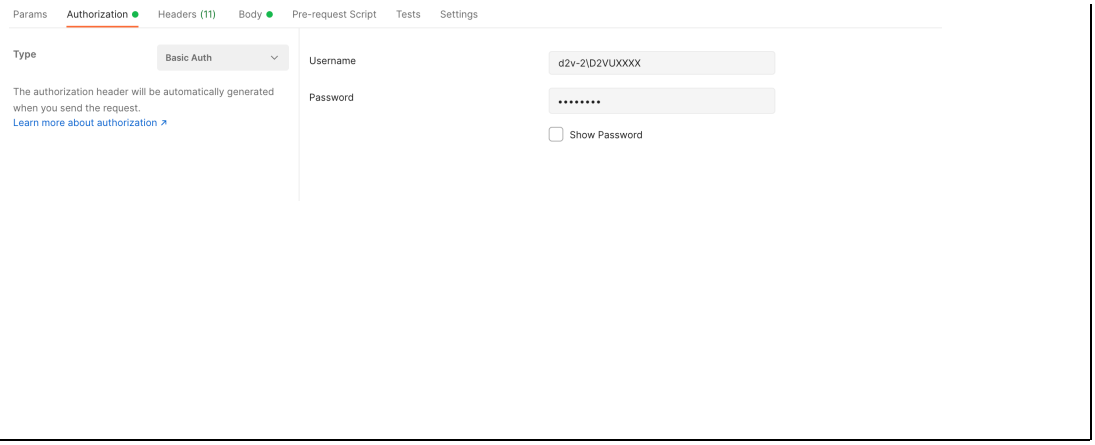
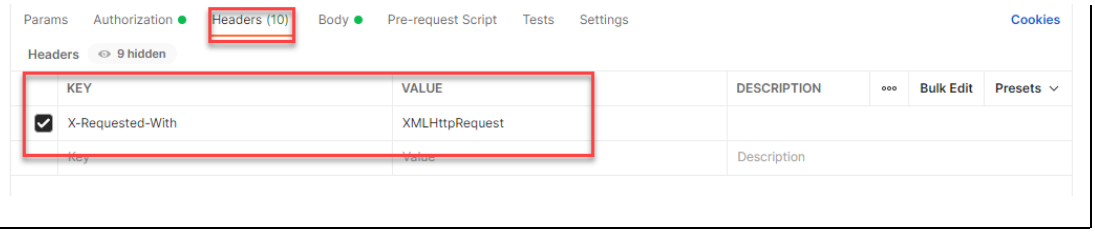
Explanation	Screenshot
Finally, you need to assign the Docker image to the “Python36 – Inference” operator. As before, right-click the operator and select “Group”. Add the tag of your docker image Save the changes.	
Click on OpenAPIServlow and have a look at the configuration	

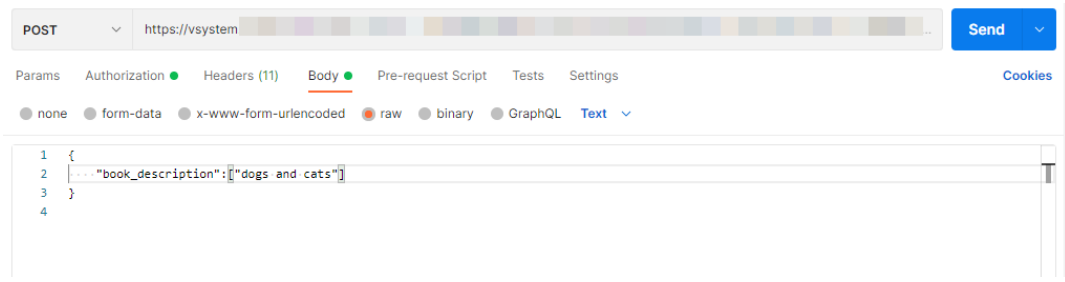
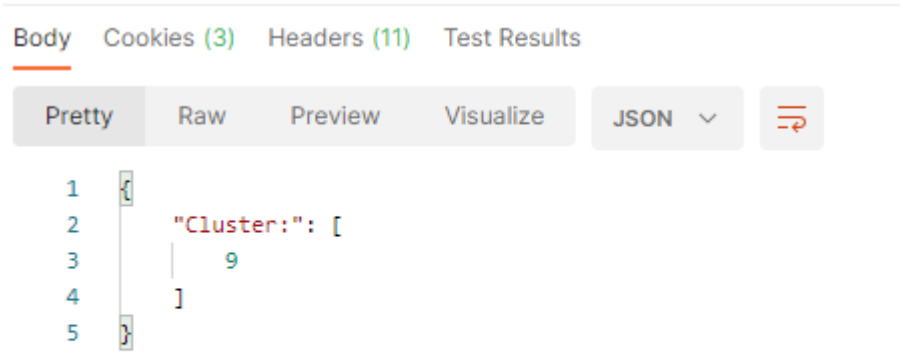
Explanation	Screenshot
	
Notice in particular the content of the Swagger Specification.	{ "schemes":["http", "https"], "swagger":"2.0", "info":{ "description":"This is an example of using the OpenAPI Servlow to carry out inference with an existing model.", "title":"OpenAPI demo", "termsOfService":"http://www.sap.com/vora/terms/", "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" },

Explanation	Screenshot
	"500":{ "description":"Error during upload of json" } } } } }, "definitions":{ }, "securityDefinitions":{ "UserSecurity":{ "type":"basic" } } }
Go back to the ML Scenario. Now deploy the new pipeline. Select the pipeline and click “Deploy”.	
Click through the screens until you can select the trained model from the drop-down. Click “Save”.	
After a few minutes the pipeline will start running.	

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
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”.	
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”.	

Explanation	Screenshot
Finally, pass the input data to the REST-API. Select the “Body” tab, choose “raw” and enter the syntax given here.	 { "book_description":["dogs and cats"] }
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.	
You have now completed the exercise.	