



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

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

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

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

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

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

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

Table of Contents

DISCLAIMER	4
OBJECTIVE	4
SCENARIO	4
ENVIRONMENT ACCESS	5
STEP 1 – USE A JUPYTER NOTEBOOK.....	6
STEP 2 – BUILD MODEL PIPELINES	9
STEP 3 – USE YOUR 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.

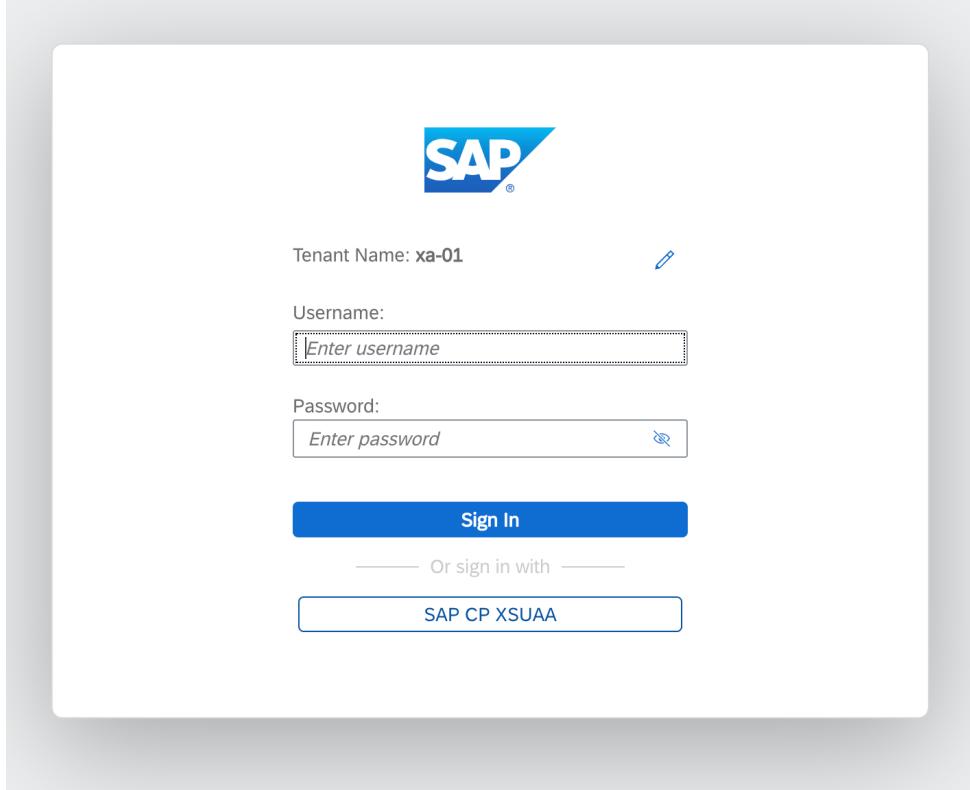
OBJECTIVE

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

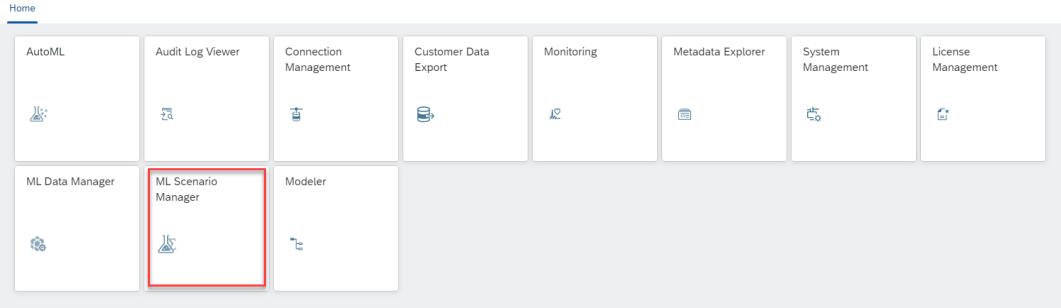
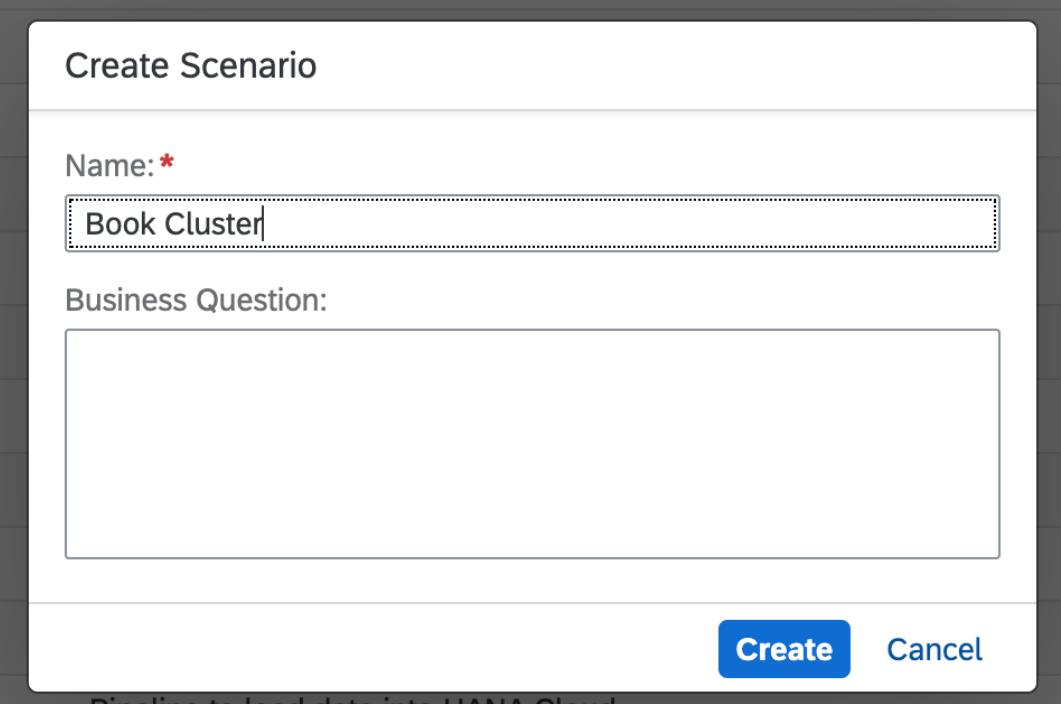
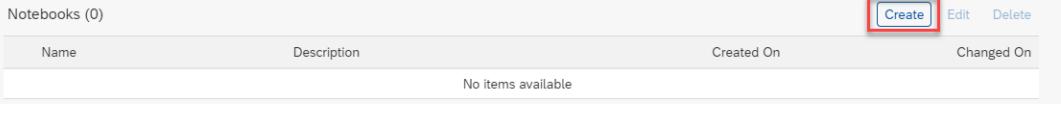
Books are grouped together in a bookshop based on their similarity, so that customers browsing for a book will find lots of similar books on the same shelf. This exercise analyzes the book description data using Python text mining algorithms and then uses this information to assign each book to a cluster.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

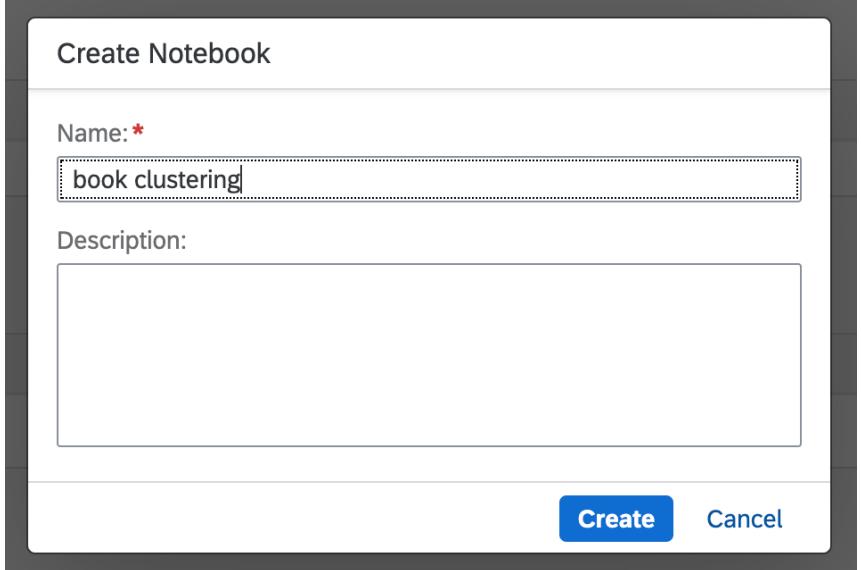
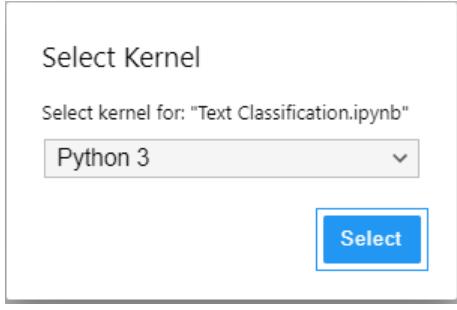
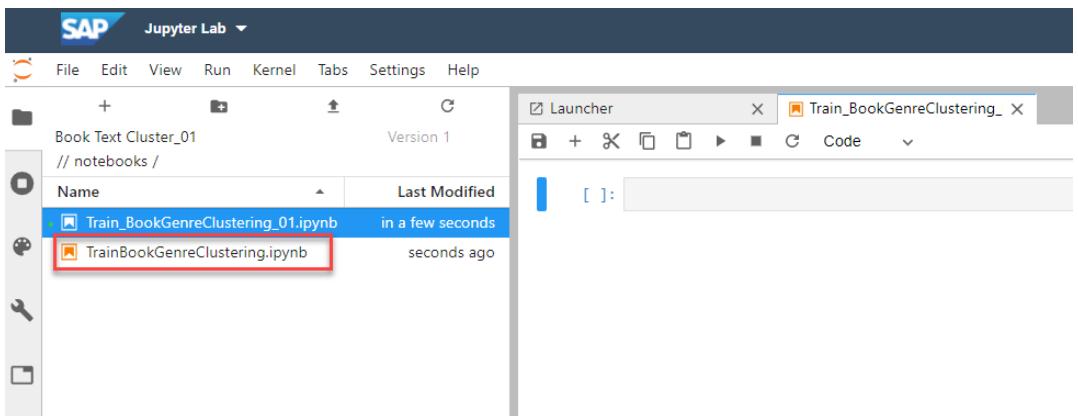
ENVIRONMENT ACCESS

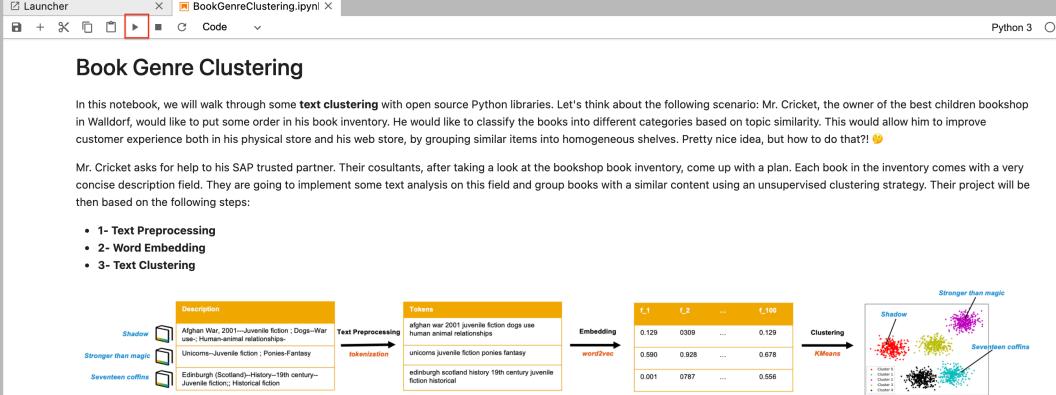
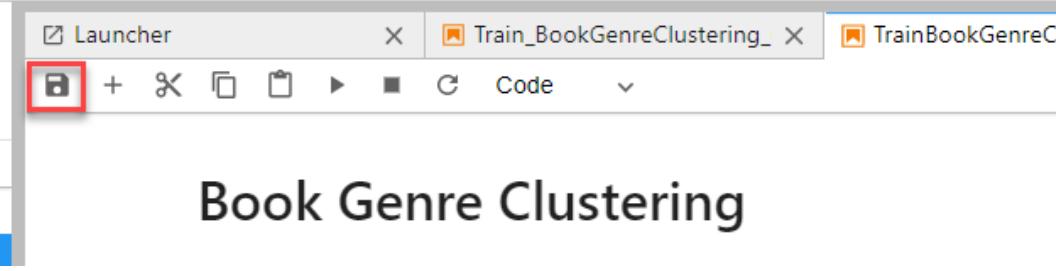
Explanation	Screenshot
Login to your SAP Data Intelligence Cloud tenant with the credentials provided by your instructor	

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.

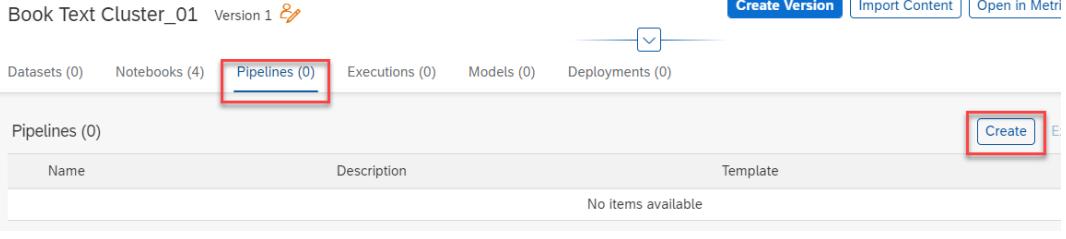
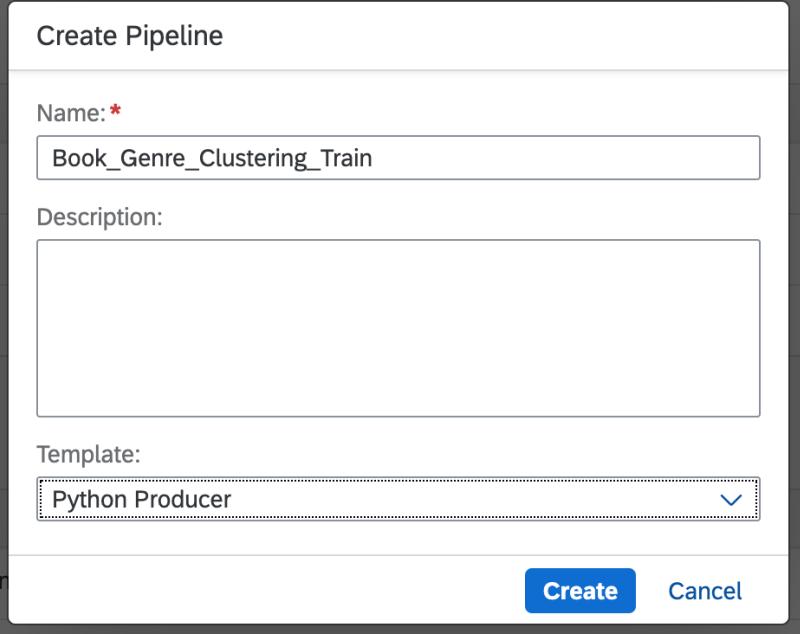
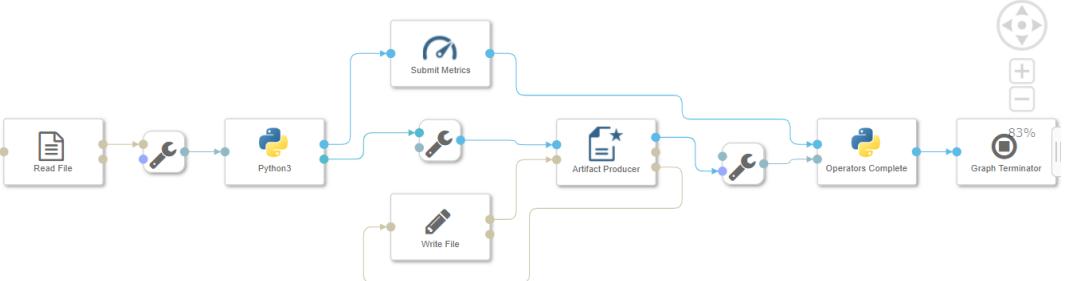
Explanation	Screenshot
Click to open ML Scenario Manager	
<p>Click the Create button. Create a new scenario. Name the scenario “Book Cluster <your user id>”</p> <p>You see the empty scenario. First, you will use the Notebooks to explore the data and to script the text analysis and cluster model in Python. Next, pipelines bring the code into production. Executions of these pipelines will create Machine Learning models, which are then deployed as REST-API for inference.</p>	
In the Notebooks section, click Create to create a new notebook.	

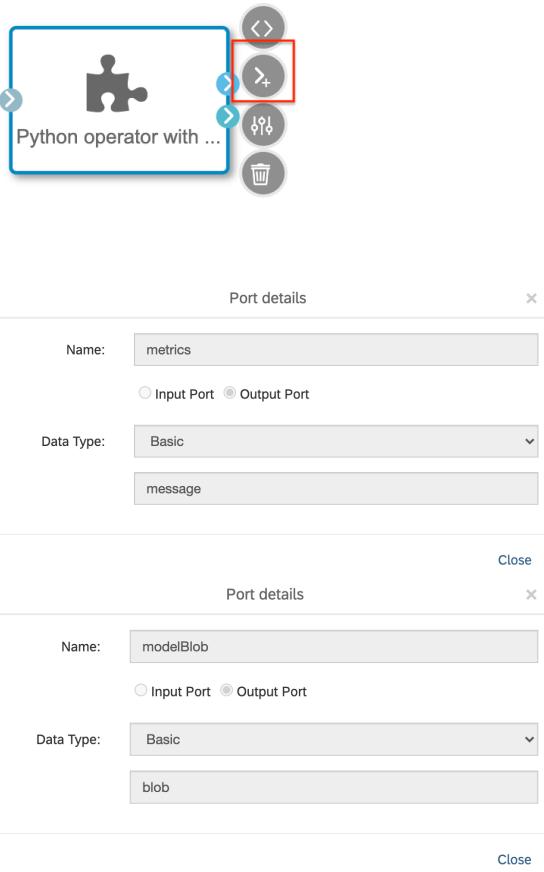
Explanation	Screenshot
Name the notebook "book clustering"	 <p>Create Notebook</p> <p>Name: *</p> <input type="text" value="book clustering"/> <p>Description:</p> <p>Create Cancel</p>
Select the Python 3 Kernel option.	 <p>Select Kernel</p> <p>Select kernel for: "Text Classification.ipynb"</p> <p>Python 3</p> <p>Select</p>
<p>Use the Upload Files function to upload the Python code into the console.</p> <p>Upload file BookGenreClustering.ipynb and text_clustering.png (you can find them in the bookcamp github repository)</p> <p>Double click on the notebook that is uploaded.</p>	 <p>SAP Jupyter Lab</p> <p>File Edit View Run Kernel Tabs Settings Help</p> <p>Book Text Cluster_01 Version 1</p> <p>Name Last Modified</p> <p>TrainBookGenreClustering.ipynb in a few seconds</p> <p>TrainBookGenreClustering.ipynb seconds ago</p> <p>Launcher Train_BookGenreClustering_ X</p> <p>Select file BookGenreClustering.ipynb</p>

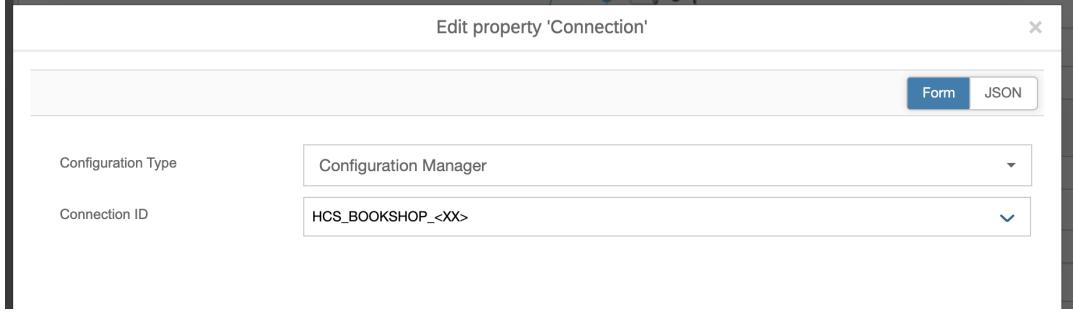
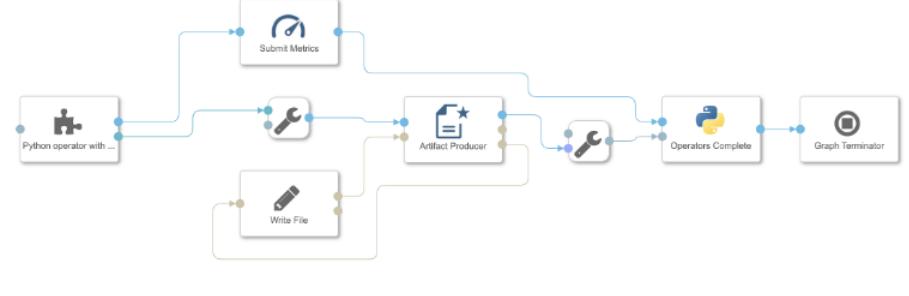
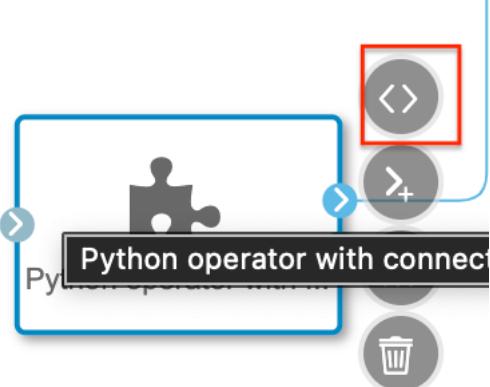
Explanation	Screenshot
<p>Double click on the notebook that is uploaded.</p> <p>Step through the code to check it works correctly.</p> <p>Highlight each cell in sequence and click the arrow button to run the selected cell and advance. Note that line 9 loads the word embedding data and line 19 runs a tsne model. Both of these steps will take a few minutes to complete. Analyze the results of your exploratory data analysis and understand the text analysis and cluster model results.</p>	 <p>Let's put this into practice!</p> <p>First, we will make sure the required libraries are installed. We will use a set of very common python libraries, for dataframe handling and visualization (pandas, numpy, matplotlib, seaborn), regex and nltk for text cleaning and preprocessing, gensim for the word embedding and sklearn for the clustering. hana_ml will be used in this notebook only to access the book inventory data, that are stored into Mr. Cricket HANA Cloud database.</p> <pre>[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>Once you have stepped through to the end of the notebook, and there are no errors, save the notebook.</p>	

STEP 2 – BUILD MODEL PIPELINES

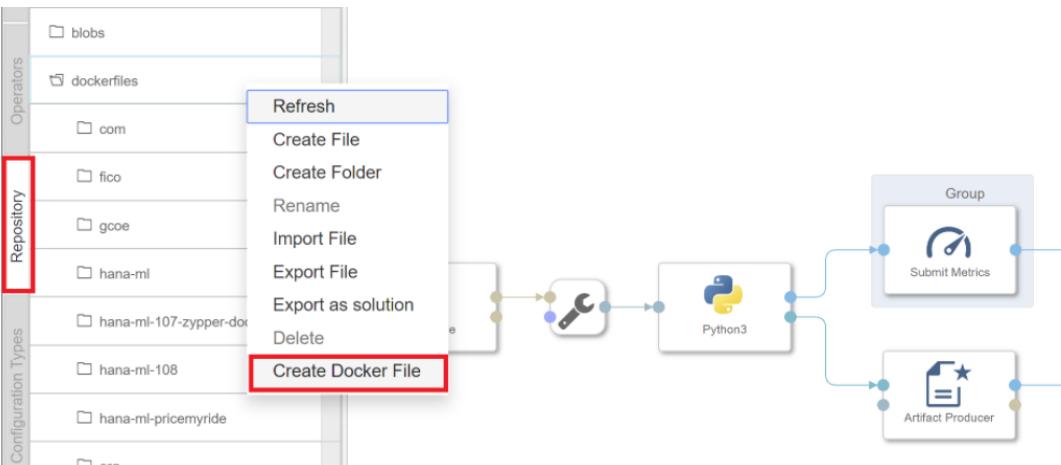
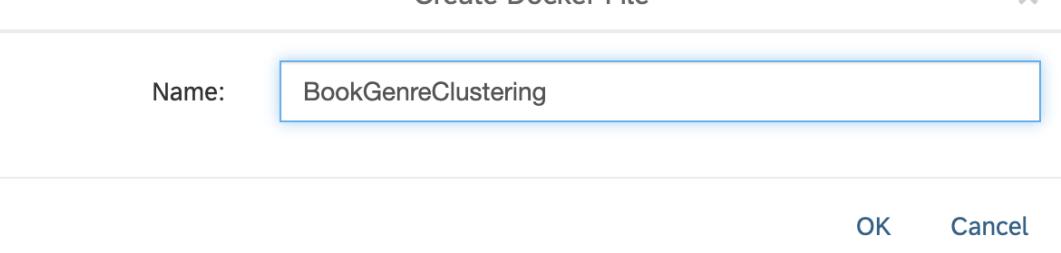
Model pipelines are used to operationalize the cluster model. Now that you have prepared the data, identify the best algorithm to use and which hyper-parameters work best, you want to take the model to production. You will build two pipeline. The first one is used to automate the model training, while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

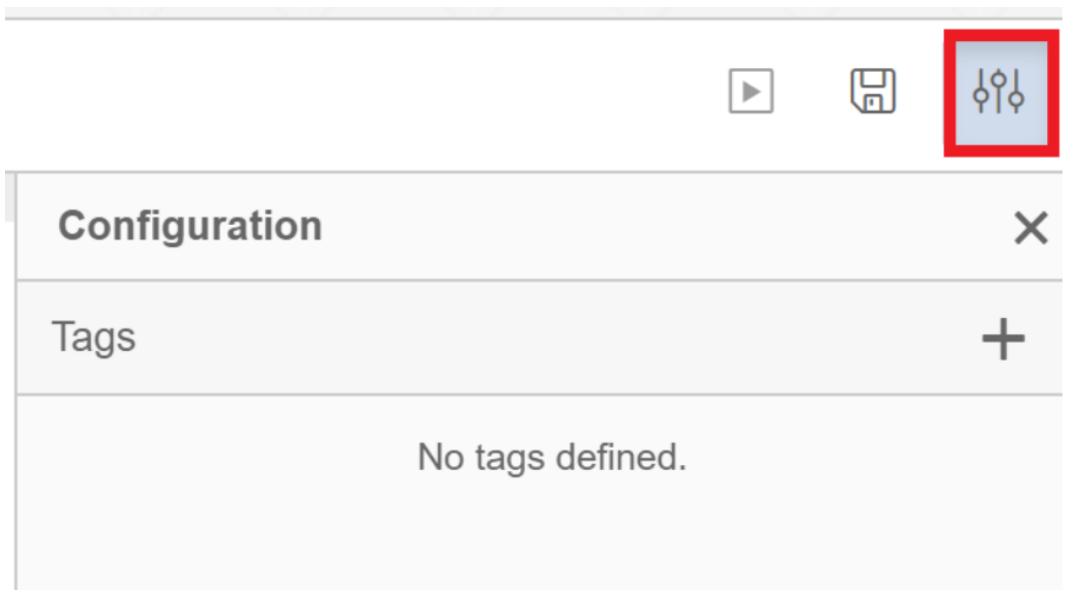
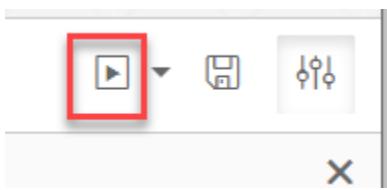
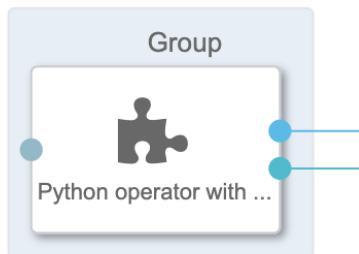
Explanation	Screenshot
<p>To create the graphical pipeline to retrain the model, go to your ML Scenario's main page, select the "Pipelines" tab and click Create.</p>	 <p>The screenshot shows the 'Book Text Cluster_01 Version 1' interface. At the top, there are tabs for Datasets (0), Notebooks (4), Pipelines (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:' (Book_Genre_Clustering_Train), 'Description:' (empty), and 'Template:' (Python Producer). A large red box highlights the 'Create' button at the bottom right of the dialog.</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. It consists of several operators connected by arrows: Read File → Python3 (ML Model Training) → Submit Metrics → Artifact Producer → Write File → Operators Complete → Graph Terminator. There are also feedback loops from the Write File and Operators Complete operators back to the Read File operator. A red box highlights the 'Graph Terminator' operator at the end of the pipeline.</p>

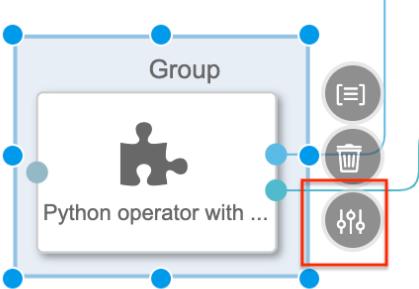
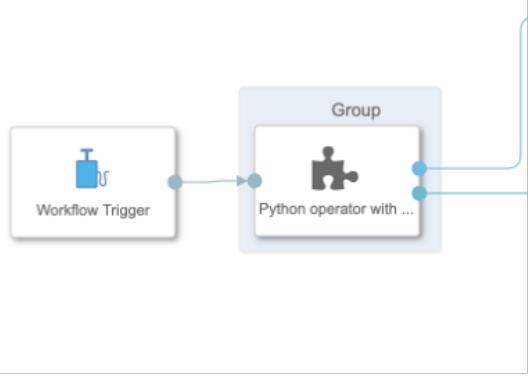
Explanation	Screenshot
<p>Scenario through the “Artifact Producer”. The Python-operator’s second output passes a model quality metric to the ML Scenario. Once both model and metric are saved, the pipeline’s execution is ended with the “Graph Terminator”.</p>	
<p>Since for us the input data are not stored 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 a Python with HANA ML connection operator 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>	 <div data-bbox="453 756 997 950"> <p>Python operator with ...</p> <p>Add port</p> </div> <div data-bbox="453 1034 997 1267"> <p>Port details</p> <p>Name: metrics</p> <p><input type="radio"/> Input Port <input checked="" type="radio"/> Output Port</p> <p>Data Type: Basic</p> <p>message</p> </div> <div data-bbox="453 1288 997 1584"> <p>Port details</p> <p>Name: modelBlob</p> <p><input type="radio"/> Input Port <input checked="" type="radio"/> Output Port</p> <p>Data Type: Basic</p> <p>blob</p> </div>

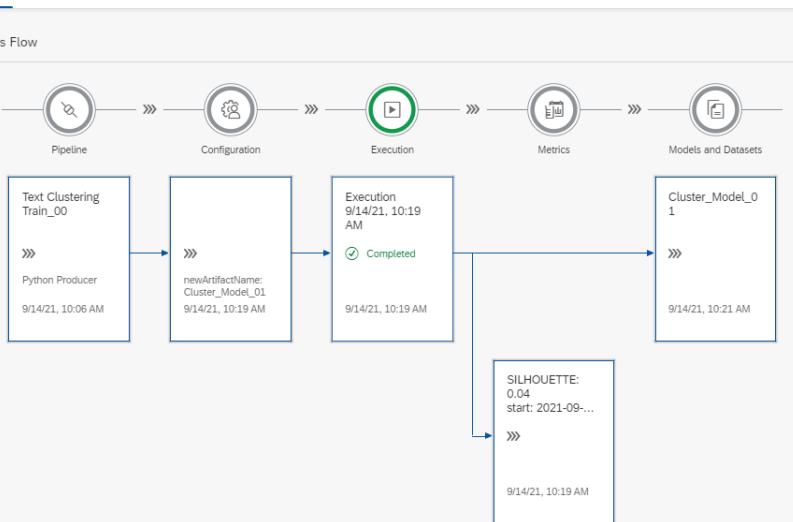
Explanation	Screenshot
<p>In the Configuration panel, open the “Connection” configuration, set “Configuration type” to “Connection Management” and select from the drop-down menu the connection ID related to your Hana Cloud instance</p>	
<p>Delete the Read File operator, and the Conversion operator. Replace 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.</p>	<pre data-bbox="425 1685 899 1928"> 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') </pre>

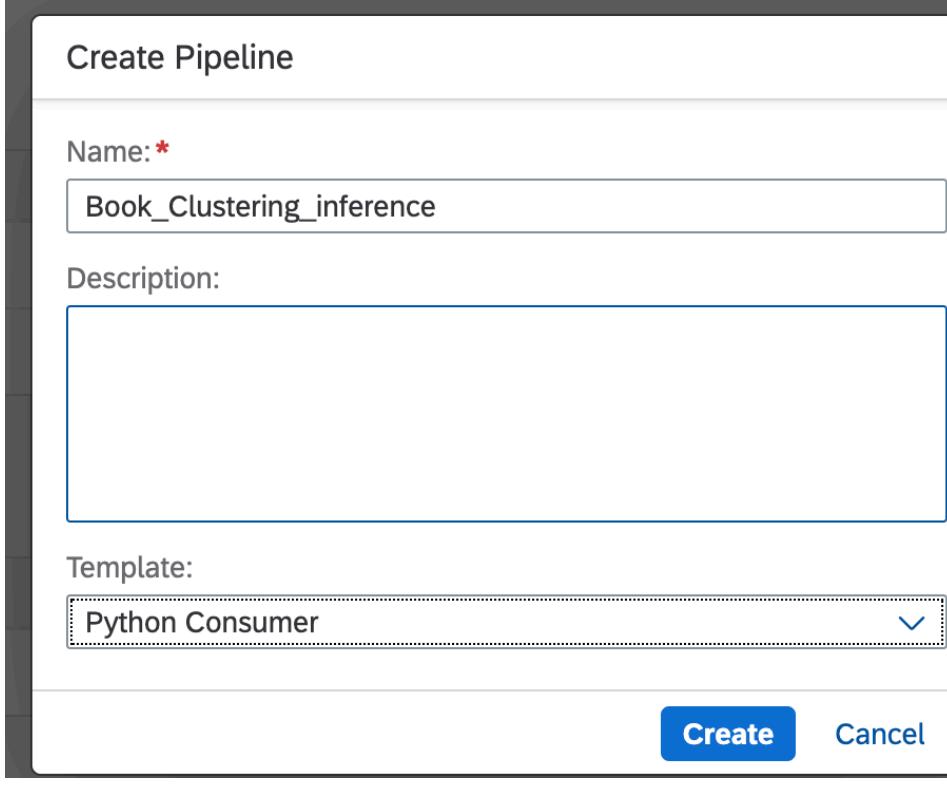
Explanation	Screenshot
<p>Carefully copy and paste to replace the existing code with the code given here, so that we can operationalize the clustering model we developed in the notebook.</p>	<pre> sslValidateCertificate='false') # insert your specific code / script here ... import hana_ml.dataframe as dataframe df_hana = (conn.table('SAP_CAPIRE_BOOKSHOP_BOOKS', schema='DATA2VALUE')) books= df_hana.select(['ID','DESCR']).collect() del df_hana books=books.dropna(axis=0,subset=['DESCR']) import nltk from nltk.corpus import stopwords nltk.download("stopwords") stopwords=set(stopwords.words("english")) import regex as re #Transform to lower case books["tokens"]=books["DESCR"].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) </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 “dockerfiles” folder and select “Create Docker File”.</p>	
<p>Name the file BookGenreClustering</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 data-bbox="437 1262 833 1537">FROM \$com.sap.sles.base RUN pip install --user numpy RUN pip install --user pandas RUN pip install --user sklearn RUN pip install --user hana_ml RUN pip install --user regex RUN pip install --user nltk RUN pip install --user genism</pre>

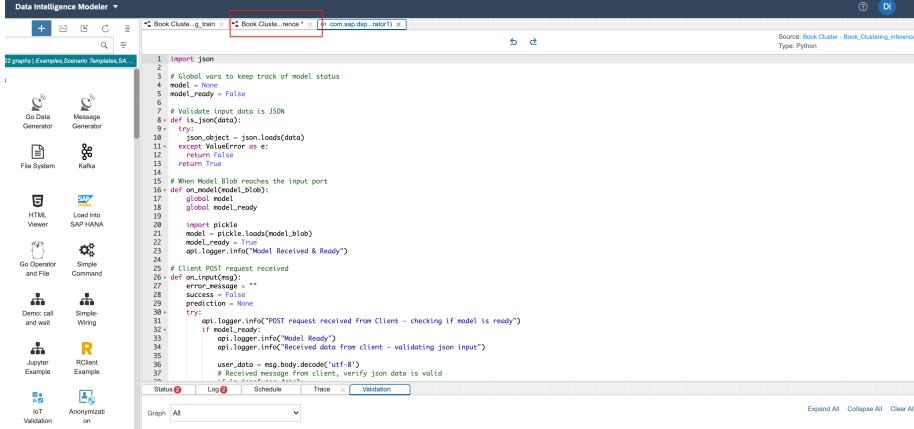
Explanation	Screenshot
<p>Open the Configuration panel for the Docker File with the icon on the top-right hand corner.</p>	
<p>Assign a tag to the docker image</p>	
<p>Now save the Docker file and click the “Build” icon to start building the Docker image. Wait a few minutes and you should receive a confirmation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

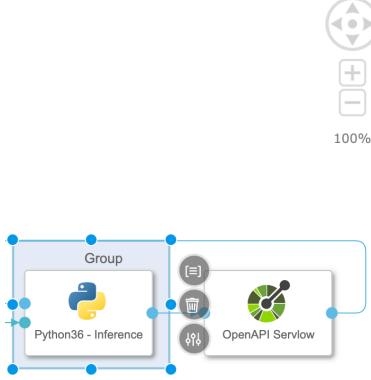
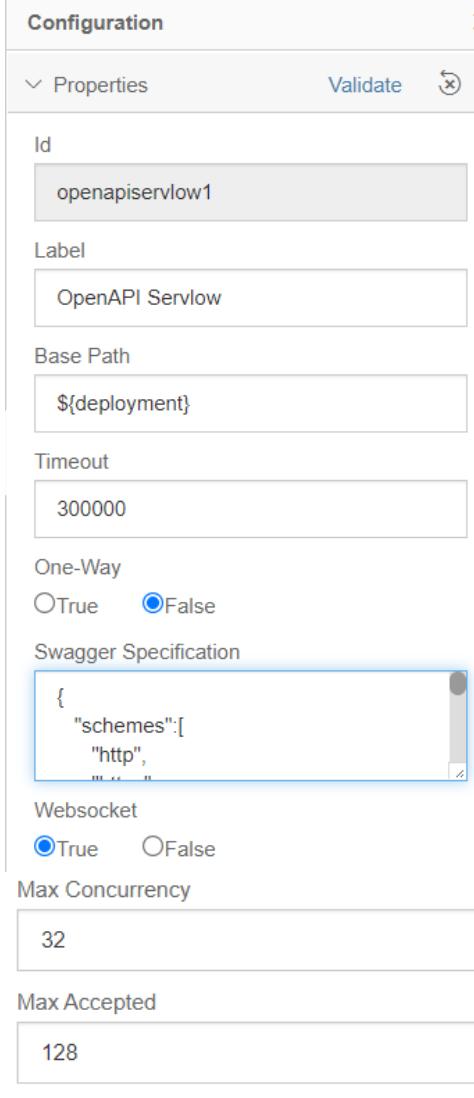
Explanation	Screenshot
<p>You specify which Docker image should be used. Select the group which surrounds the “Python 3” Operator. In the group’s Configuration select the docker image you have built.</p>	 <div data-bbox="430 599 1491 1417"> <p>Configuration X</p> <p>Properties Validate</p> <p>Id group1</p> <p>Description Group</p> <p>Restart Policy</p> <div style="border: 1px solid black; padding: 2px; display: inline-block;"> Name: metricsResponse Data Type: message </div> <p>Tags +</p> <div style="border: 1px solid red; padding: 2px; display: inline-block;"> BookGenreClustering ▼ </div> <div style="border: 1px solid red; padding: 2px; display: inline-block;"> latest ▼ X </div> <p>Multiplicity</p> </div>
<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>	

Explanation	Screenshot				
<p>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.</p>					
<p>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>	<p>4. Pipeline Parameters</p> <p>Enter your Pipeline Parameters.</p> <table border="1" data-bbox="437 656 1491 798"> <thead> <tr> <th data-bbox="437 656 1328 692">Key</th> <th data-bbox="1328 656 1491 692">Value</th> </tr> </thead> <tbody> <tr> <td data-bbox="437 692 1328 728">newArtifactName*</td> <td data-bbox="1328 692 1491 728">kmeans</td> </tr> </tbody> </table> <p>Save</p>	Key	Value	newArtifactName*	kmeans
Key	Value				
newArtifactName*	kmeans				
<p>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>	<p>Status: Completed 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> 				

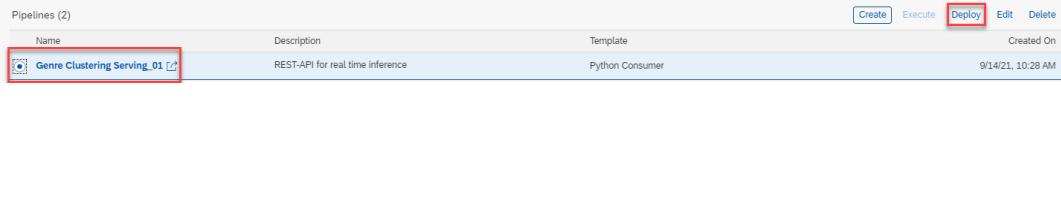
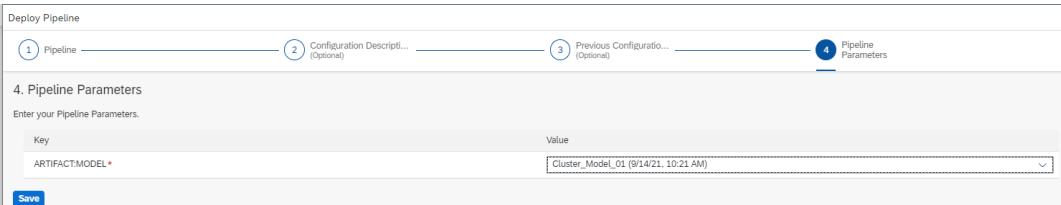
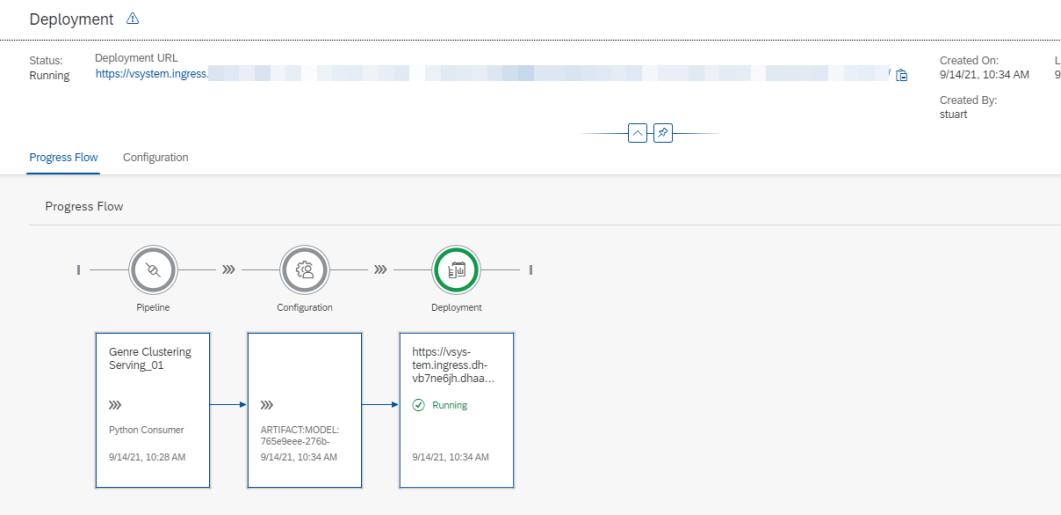
Explanation	Screenshot
<p>You will now use the model for real-time inference with REST-API. Go back to the main page of your ML Scenario and create a second pipeline. This pipeline will provide the REST-API to obtain predictions in real-time. Name the pipeline “Book Clustering Consumer”. Select the template “Python Consumer”. This template contains a pipeline that provides a REST-API.</p>	 <p>The screenshot shows the 'Create Pipeline' dialog box. The 'Name:' field is filled with 'Book_Clustering_inference'. The 'Description:' field is empty. The 'Template:' dropdown is set to 'Python Consumer'. At the bottom right are 'Create' and 'Cancel' buttons.</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] --> B{Artifact Consumer} B --> C[Read File] C --> D[Python36 - Inference] D --> E[OpenAPI Servlow] </pre> <p>The diagram illustrates the pipeline flow. It begins with the 'Submit Artifact Name' operator, followed by the 'Artifact Consumer' operator. The output of 'Artifact Consumer' goes to the 'Read File' operator. The output of 'Read File' goes to the 'Python36 - Inference' operator. Finally, the output of 'Python36 - Inference' goes to the 'OpenAPI Servlow' operator.</p>

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.	<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 & 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 </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>
Go back to the pipeline tab and save the pipeline	

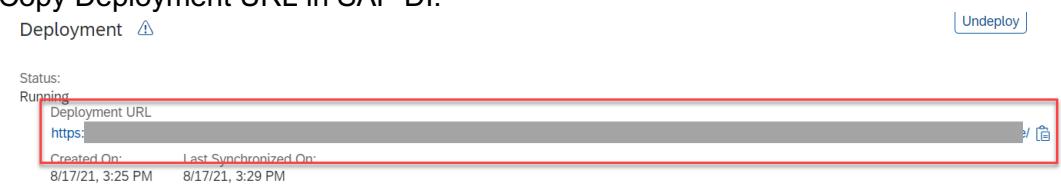
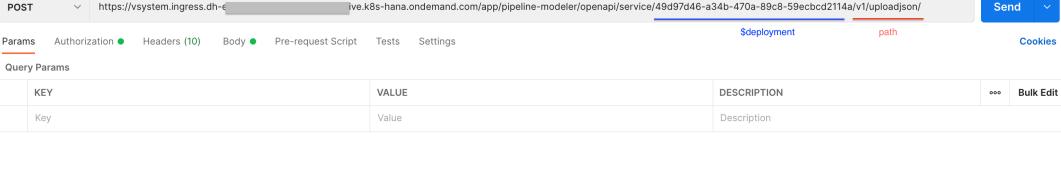
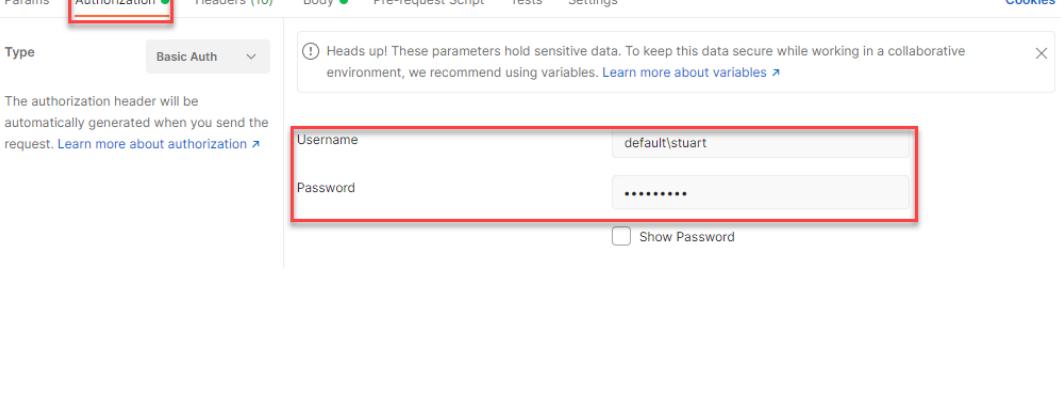
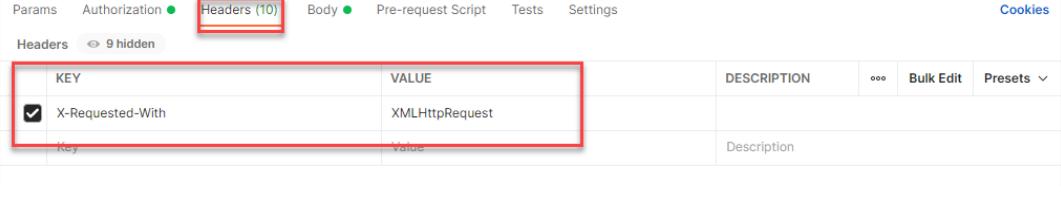
Explanation	Screenshot
<p>Finally, you need to assign the Docker image to the “Python36 – Inference” operator. As before, right-click the operator and select “Group”. Add the tag of your docker image Save the changes.</p>	 <div data-bbox="801 255 1269 762"> <p>Configuration</p> <p>Properties</p> <p>Id: group1</p> <p>Description: Group</p> <p>Restart Policy: (dropdown menu)</p> <p>Tags: BookGenreClustering latest</p> <p>Multiplicity: (dropdown menu)</p> <p>Resources: (button)</p> </div>
<p>Click on OpenAPIServlow and have a look at the configuration</p>	 <div data-bbox="425 819 899 1917"> <p>Configuration</p> <p>Properties</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", ...] }</pre> <p>Websocket: <input checked="" type="radio"/> True <input type="radio"/> False</p> <p>Max Concurrency: 32</p> <p>Max Accepted: 128</p> </div>

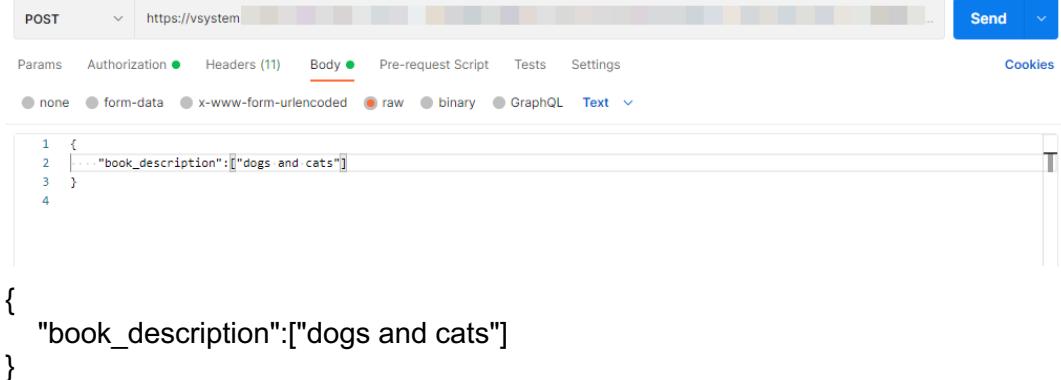
Explanation	Screenshot
<p>Notice in particular the content of the Swagger Specification.</p>	<pre>{ "schemes":["http", "https"], "swagger":"2.0", "info": { "description":"This is an example of using the OpenAPI Servlow to carry out inference with an existing model.", "title":"OpenAPI demo", "termsOfService":"http://www.sap.com/vora/terms/", "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" }, "500": { "description": "Error during upload of json" } } } }, "definitions": { </pre>

Explanation	Screenshot
	<pre data-bbox="430 255 703 466"> }, "securityDefinitions":{ "UserSecurity":{ "type":"basic" } } }</pre>
<p>Go back to the ML Scenario. Now deploy the new pipeline. Select the pipeline and click “Deploy”.</p>	
<p>Click through the screens until you can select the trained model from the drop-down. Click “Save”.</p>	
<p>After a few minutes the pipeline will start running.</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 v1/uploadjson/, the path specified in the OpenAPI servlow operator. Change the request type from “GET” to “POST”.</p>	<p>Copy Deployment URL in SAP DI:</p>  <p>To Postman:</p> 
<p>Go to the “Authorization” tab. Select “Basic Auth” and enter your username and password for SAP Data Intelligence. The username starts with your tenant’s name, followed by a backslash and your actual username.</p>	
<p>Go to the “Headers” tab and enter the key “X-Requested-With” with value “XMLHttpRequest”.</p>	

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="425 677 1241 1199">1 { 2 "Cluster": [3 9 4] 5 }</pre>
<p>You have now completed the exercise.</p>	

