Watson Studio 2.1: What's New
Session ID: EDA23T023SH

Elena Lowery, WW Team, Data Science and IA
elowery@us.ibm.com

Table of contents

Contents

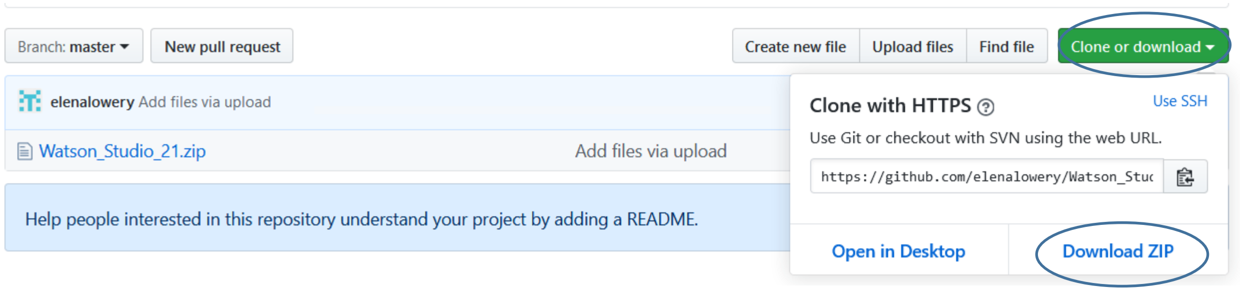
Overview

In this lab you will learn about the following new features in Watson Studio 2.1:

- Deployment
- Collaboration and Git integration
- Package and environment management
- Working with files in JupyterLab.

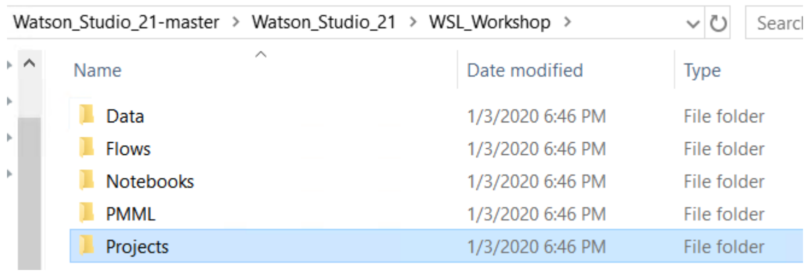
Required software, access, and files

- To complete this lab, you will need access to a **Cloud Pak for Data** (CP4D) cluster with **Watson Studio** and **WML**.
- You will need a Git account (for example, a free account on www.github.com) to complete *Collaboration and Git integration* as well as *JupyterLab* sections of the lab. You will also need a token for the git repo.
    - See **Appendix A** and **B** for Git setup instructions.
- You will also need to download and unzip this GitHub repository: https://github.com/elenalowery/Watson_Studio_21



Help people interested in this repository understand your project by adding a README.

- Unzip the files until you get to this directory structure:



Watson_Studio_21-master > Watson_Studio_21 > WSL_Workshop >

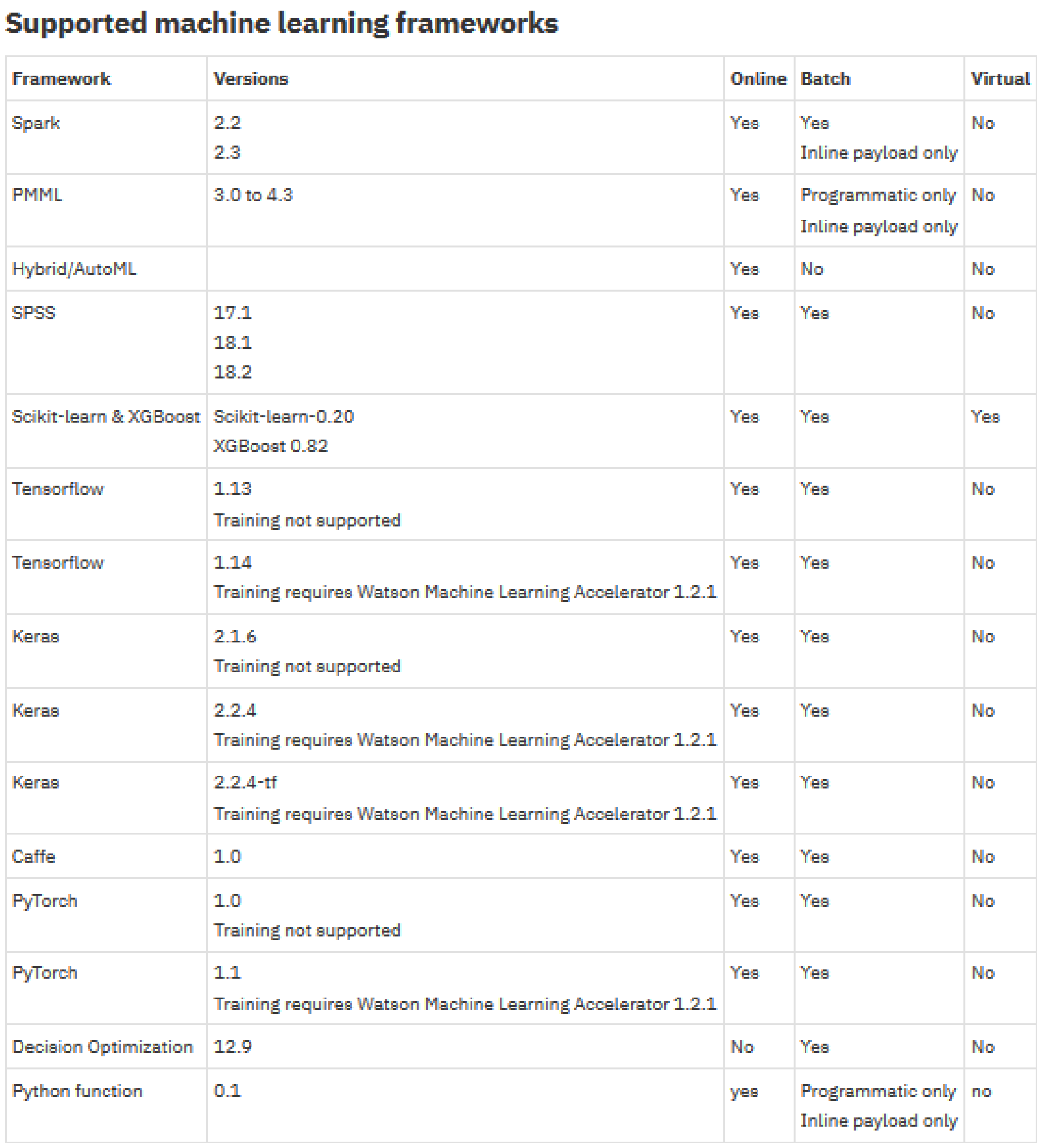
Name	Date modified	Type
Data	1/3/2020 6:46 PM	File folder
Flows	1/3/2020 6:46 PM	File folder
Notebooks	1/3/2020 6:46 PM	File folder
PMML	1/3/2020 6:46 PM	File folder
Projects	1/3/2020 6:46 PM	File folder

In the lab we will refer to this folder as the *git repo* folder.

Part 1: Deployment in Watson Studio

Deployment is the process of configuring an analytic asset for integration with other applications or access by business users.

Several types of analytics assets can be deployed in Watson Studio. The most current list of supported deployments can be found in documentation:
https://www.ibm.com/support/knowledgecenter/en/SSQNUZ_2.5.0/wsj/analyze-data/pm_service_supported_frameworks.html



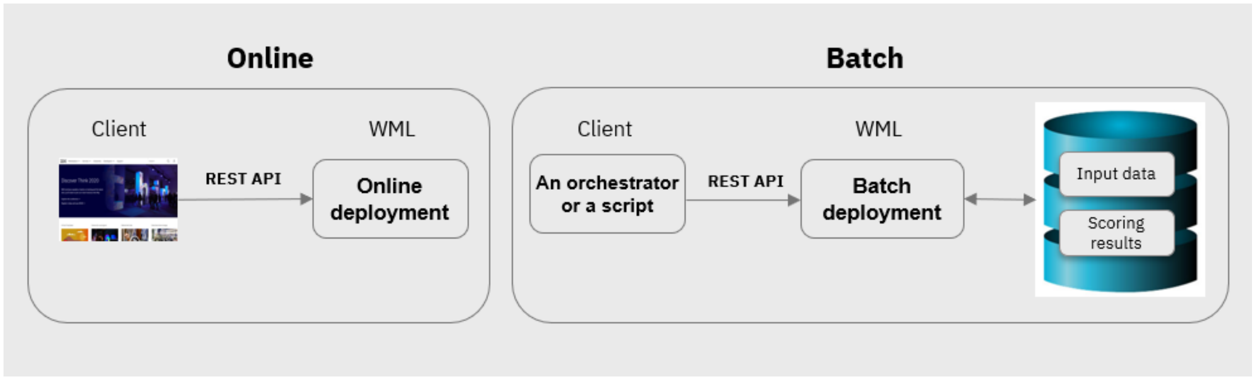
Supported machine learning frameworks

Framework	Versions	Online	Batch	Virtual
Spark	2.2 2.3	Yes	Yes Inline payload only	No
PMML	3.0 to 4.3	Yes	Programmatic only Inline payload only	No
Hybrid/AutoML		Yes	No	No
SPSS	17.1 18.1 18.2	Yes	Yes	No
Scikit-learn & XGBoost	Scikit-learn-0.20 XGBoost 0.82	Yes	Yes	Yes
Tensorflow	1.13 Training not supported	Yes	Yes	No
Tensorflow	1.14 Training requires Watson Machine Learning Accelerator 1.2.1	Yes	Yes	No
Keras	2.1.6 Training not supported	Yes	Yes	No
Keras	2.2.4 Training requires Watson Machine Learning Accelerator 1.2.1	Yes	Yes	No
Keras	2.2.4-tf Training requires Watson Machine Learning Accelerator 1.2.1	Yes	Yes	No
Caffe	1.0	Yes	Yes	No
PyTorch	1.0 Training not supported	Yes	Yes	No
PyTorch	1.1 Training requires Watson Machine Learning Accelerator 1.2.1	Yes	Yes	No
Decision Optimization	12.9	No	Yes	No
Python function	0.1	yes	Programmatic only Inline payload only	no

Watson Studio supports three deployment options:

- **Online**: a real time request/response deployment option. When this deployment option is used, models or functions are invoked with a REST API. A single row or multiple rows of data can be passed in with the REST request.
- **Batch**: a deployment option that reads and writes from/to a static data source. A batch deployment can be invoked with a REST API.
- **Virtual**: virtual deployment provides the capability to save a model for deployment in CoreML runtime. CoreML is not included with *Watson Studio*.

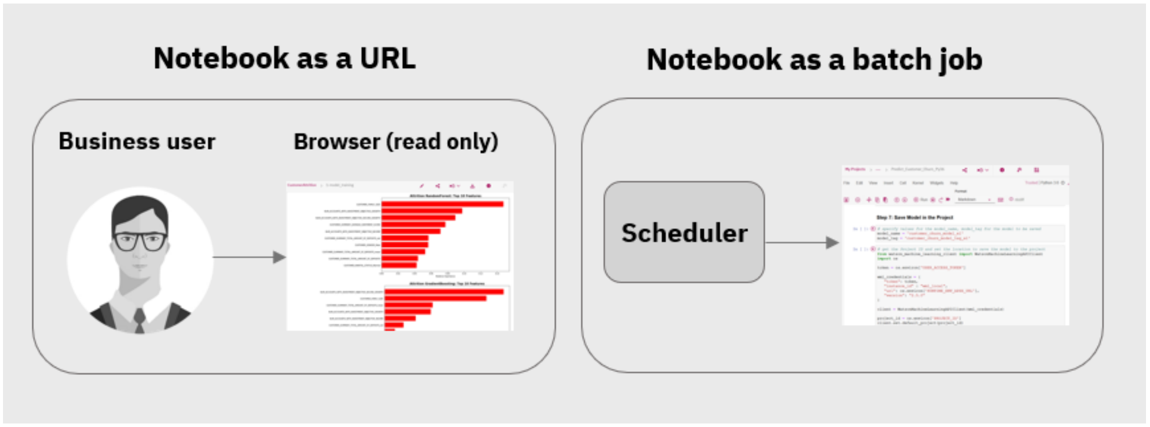
Figure 1: Model and function deployment



In addition to models and Python functions, it's also possible to deploy

- Notebooks as a read-only HTML page (a URL)
- Notebooks as a scheduled batch job.

Figure 2: Notebook deployment



IBM is planning to provide a consistent deployment experience for all analytic asset types, but since Watson Studio is a new product, the deployment experience is slightly different for models/functions and notebooks. Notebooks are deployed directly in the project, while the rest of the assets are deployed through a *Deployment Space*.

In this lab we will review deployment of a scikit-learn model, an SPSS flow, a Python function, and PMML. We will also deploy notebooks.

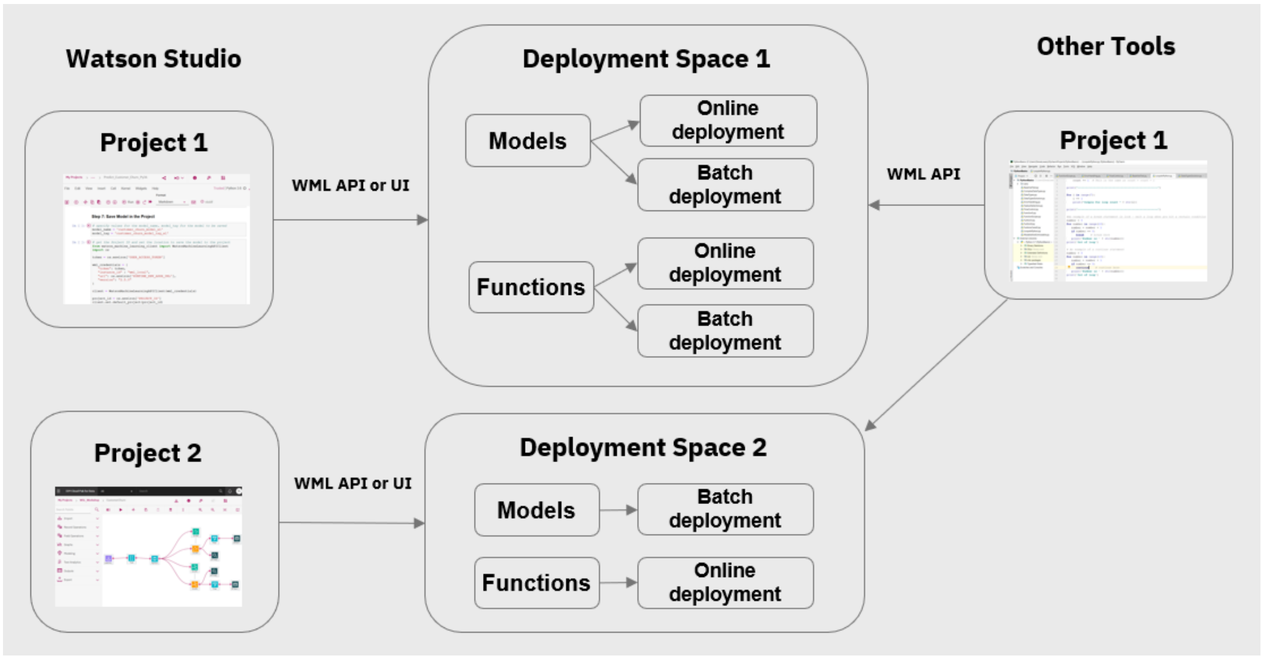
Deployment to Deployment Spaces

In this section we will deploy various analytics assets into a *deployment space*. As the name suggests, a *deployment space* is a container that's used to organize analytics assets for deployment.

Here are some important facts about deployment spaces:

- You can work with deployment spaces via UI and via API.
- In the UI there is a 1:1 relationship between a deployment space and a project.
- If you're using an API, many clients can publish to the same deployment space. For example, a notebook from another project or even a tool outside of Watson Studio.

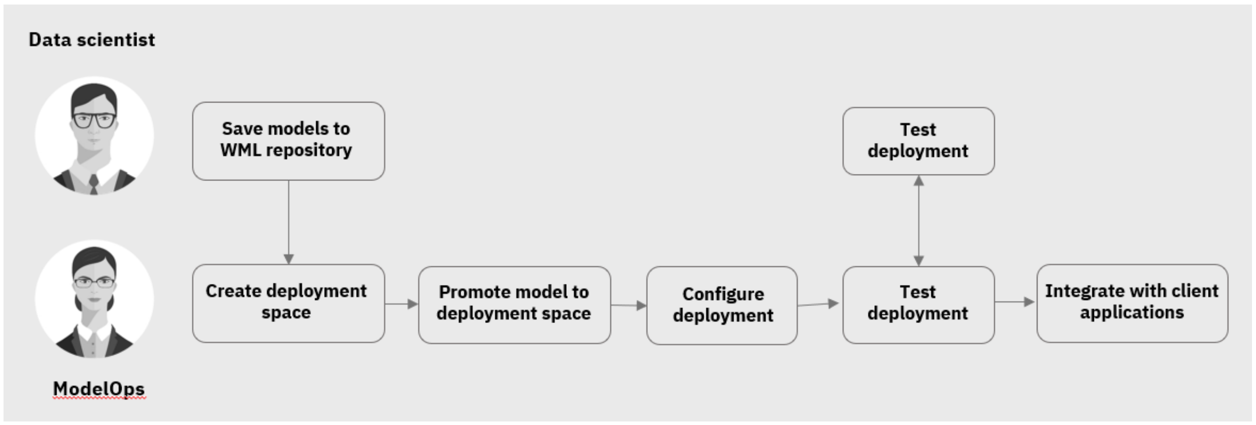
Figure 3: Deployment Spaces and projects



The deployment flow for most asset types consists of the following steps:

1. Create a deployment space.
2. Associate a deployment space with a project.
3. Save the asset (model, PMML, SPSS, etc.) into project repository.
4. Promote the asset to the deployment space.
5. Configure deployment (online or batch).
6. Test the deployed asset.
7. Integrate the deployed asset with another application (via REST API).

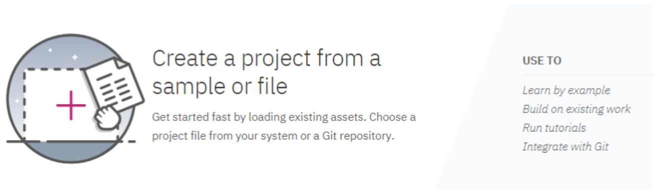
Figure 4: Deployment workflow in Watson Studio

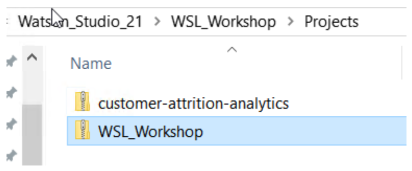

Data scientist
Save models to WML repository
Test deployment
ModelOps
Create deployment space
Promote model to deployment space
Configure deployment
Test deployment
Integrate with client applications

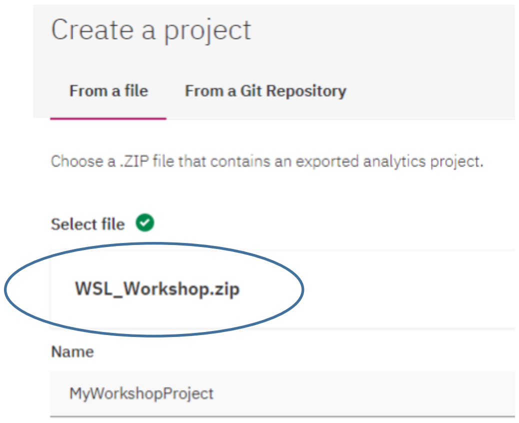
While it's possible to complete the entire deployment workflow with *WML client API*, we recommend that you start with configuring deployment via UI. Deployment with the API can provide flexibility and automation, but if you don't understand the implementation details of the deployment workflow, it may lead to poor governance of deployment.

First, we will walk through the deployment process via UI. Then we will review sample notebooks that use WML Client API for configuring deployment.

1. In Watson Studio create a new project from file – *Watson_Studio_21.zip*, located in the *git repo\Projects* folder. You can use any value for the project name.


Create a project from a sample or file
Get started fast by loading existing assets. Choose a project file from your system or a Git repository.
USE TO
Learn by example
Build on existing work
Run tutorials
Integrate with Git


Watson_Studio_21 > WSL_Workshop > Projects
Name
customer-attrition-analytics
WSL_Workshop


Create a project

From a file | From a Git Repository

Choose a .ZIP file that contains an exported analytics project.

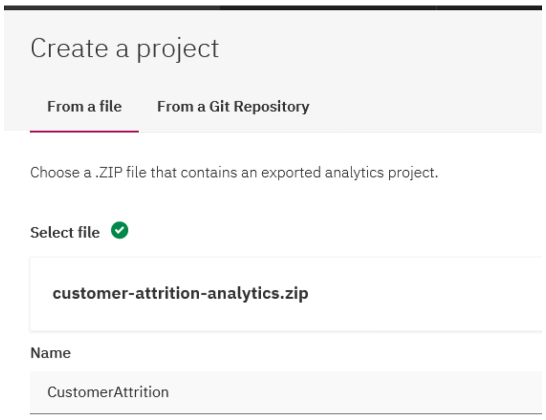
Select file

WSL_Workshop.zip

Name

MyWorkshopProject

2. In Watson Studio create a second project from file - *customer-attrition-analytics*, located in the *Projects* folder. You can specify any name for the project.


Create a project

From a file | From a Git Repository

Choose a .ZIP file that contains an exported analytics project.

Select file

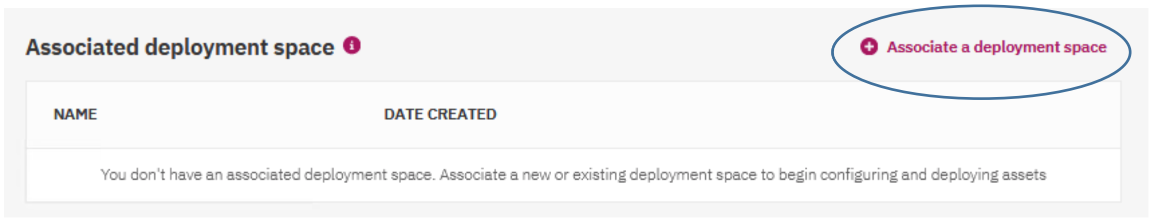
customer-attrition-analytics.zip

Name

CustomerAttrition

3. Navigate to the *Watson_Studio_21* project view (the project created in Step 1) and click on the **Settings** tab.

   At this time we don't have a deployment space associated with the project. Click **Associate a deployment space**.


Associated deployment space

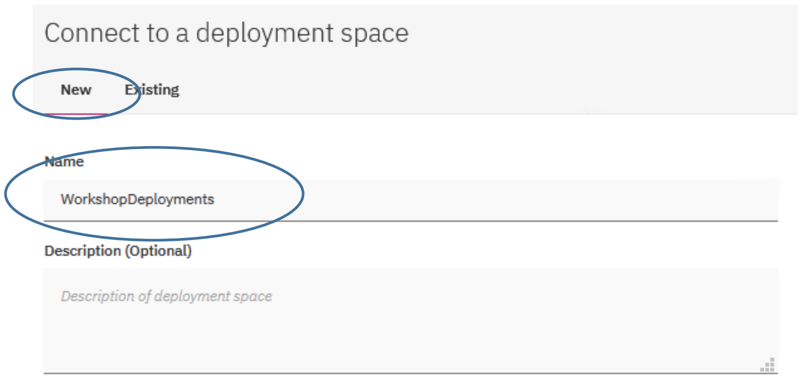
Associate a deployment space

NAME	DATE CREATED
You don't have an associated deployment space. Associate a new or existing deployment space to begin configuring and deploying assets	

   Important Note: *Once you associate a deployment space with a project, you can't modify or delete this association in Project UI. However, if you delete the deployment space from the* ***Analytics Deployments*** *view, it will be automatically removed from the project.*

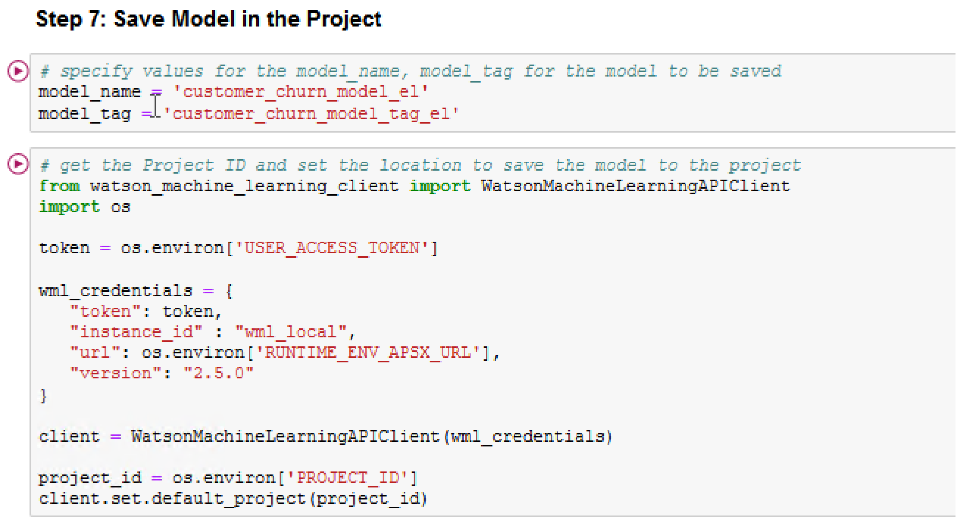
4. Provide a deployment space name and click **Associate**.

   Important Note: *While a unique deployment space name is not required, it will be easier to find your deployment space if you give it a unique id. Watson Studio allows creation of deployment spaces with the same name.*


Connect to a deployment space
New
Existing
Name
WorkshopDeployments
Description (Optional)
Description of deployment space

5. From the project view open the *Predict_Customer_Churn_Py36* notebook in *Edit* mode.

6. Review the notebook and run all cells in the notebook until you get to **Step 7**.



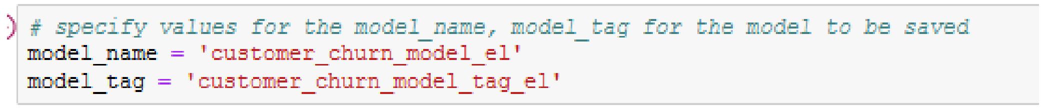
Step 7: Save Model in the Project

```
# specify values for the model_name, model_tag for the model to be saved
model_name = 'customer_churn_model_el'
model_tag = 'customer_churn_model_tag_el'
```

```
# get the Project ID and set the location to save the model to the project
from watson_machine_learning_client import WatsonMachineLearningAPIClient
import os

token = os.environ['USER_ACCESS_TOKEN']

wml_credentials = {
   "token": token,
   "instance_id" : "wml_local",
   "url": os.environ['RUNTIME_ENV_APSX_URL'],
   "version": "2.5.0"
}

client = WatsonMachineLearningAPIClient(wml_credentials)

project_id = os.environ['PROJECT_ID']
client.set.default_project(project_id)
```

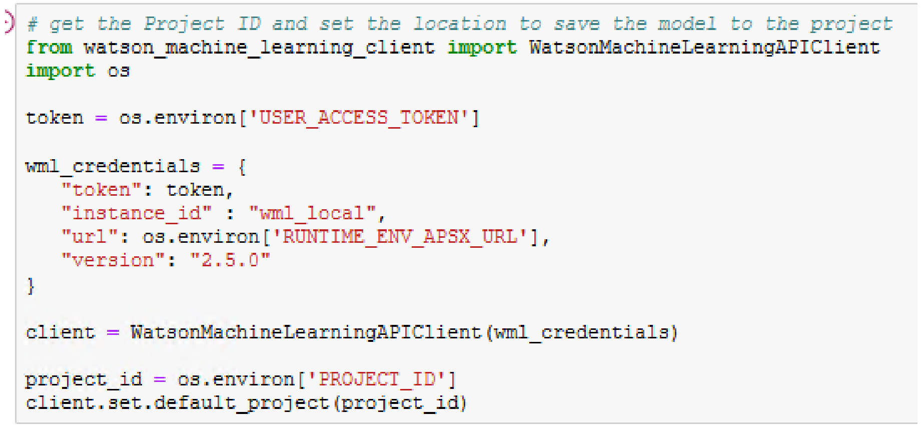
Let's review the code.

First, we specify the model name – it will be displayed in the **Assets** view of the project. If you wish, you can modify the model name and the tag.



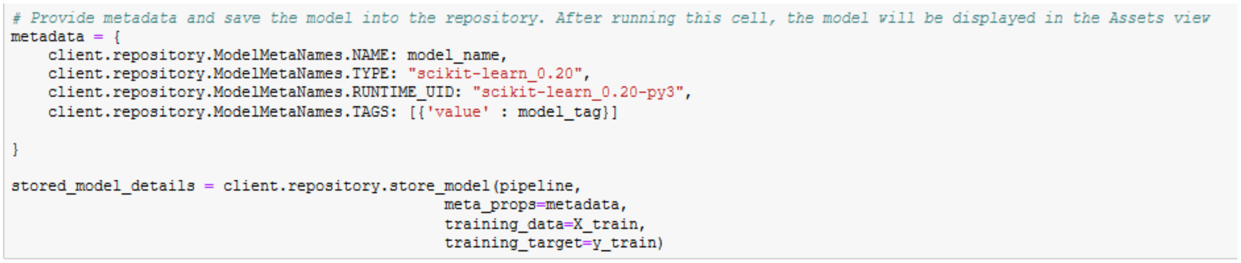
```
# specify values for the model_name, model_tag for the model to be saved
model_name = 'customer_churn_model_el'
model_tag = 'customer_churn_model_tag_el'
```

Next, we instantiate *WML client* object, which gives us access to variables that are needed to save the model into the project, specifically, the *project id.*



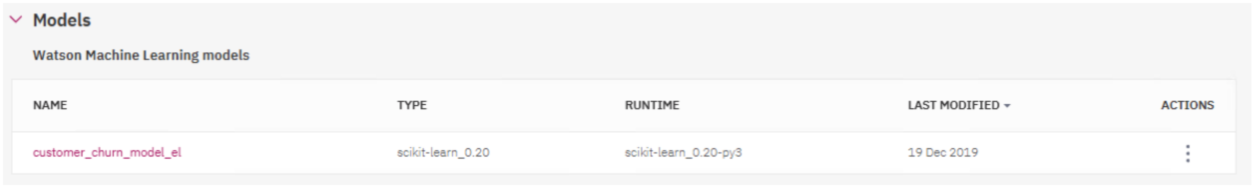
```
# get the Project ID and set the location to save the model to the project
from watson_machine_learning_client import WatsonMachineLearningAPIClient
import os

token = os.environ['USER_ACCESS_TOKEN']

wml_credentials = {
   "token": token,
   "instance_id" : "wml_local",
   "url": os.environ['RUNTIME_ENV_APSX_URL'],
   "version": "2.5.0"
}

client = WatsonMachineLearningAPIClient(wml_credentials)

project_id = os.environ['PROJECT_ID']
client.set.default_project(project_id)
```

Finally, we save the model.



```
# Provide metadata and save the model into the repository. After running this cell, the model will be displayed in the Assets view
metadata = {
    client.repository.ModelMetaNames.NAME: model_name,
    client.repository.ModelMetaNames.TYPE: "scikit-learn_0.20",
    client.repository.ModelMetaNames.RUNTIME_UID: "scikit-learn_0.20-py3",
    client.repository.ModelMetaNames.TAGS: [{'value' : model_tag}]

}

stored_model_details = client.repository.store_model(pipeline,
                                              meta_props=metadata,
                                              training_data=X_train,
                                              training_target=y_train)
```

7. Save the notebook and navigate back to the project **Assets** view. Notice that the saved model is displayed in the **Models** section.



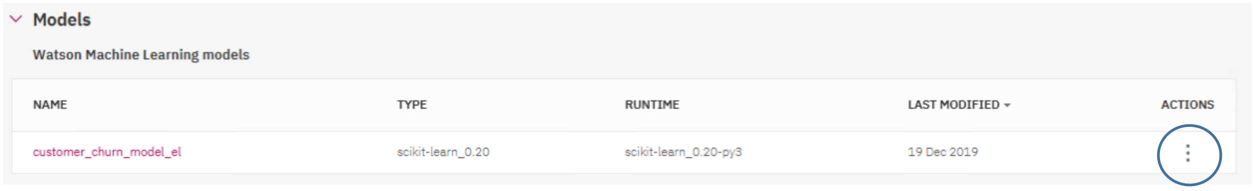
Models

Watson Machine Learning models

NAME	TYPE	RUNTIME	LAST MODIFIED	ACTIONS
customer_churn_model_el	scikit-learn_0.20	scikit-learn_0.20-py3	19 Dec 2019	⋮

Important Note: *At this time the API does not support model versioning. A different model with the same name will be saved each time you run the code to save a model. This issue will be fixed in the next release.*

8. Under **Actions** click on the ellipsis next to the model and select **Promote**.

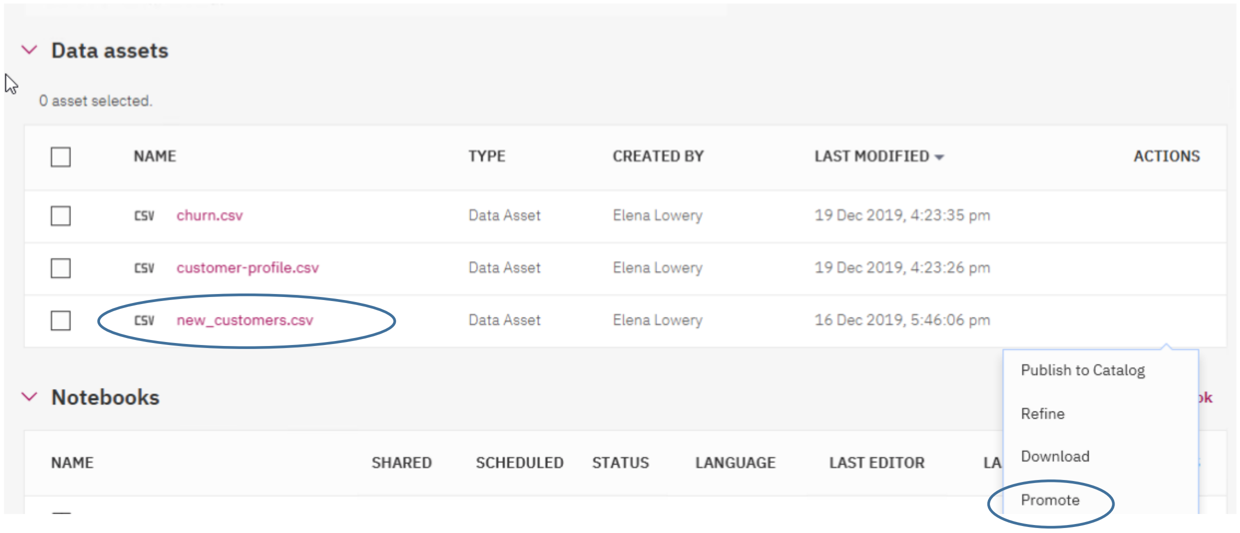


Models

Watson Machine Learning models

NAME	TYPE	RUNTIME	LAST MODIFIED	ACTIONS
customer_churn_model_el	scikit-learn_0.20	scikit-learn_0.20-py3	19 Dec 2019	⋮

9. From the same project view promote the *new_customers* and the *flow_customer_churn_batch_scoring.csv* data assets. We will use these files as input datasets for batch scoring.

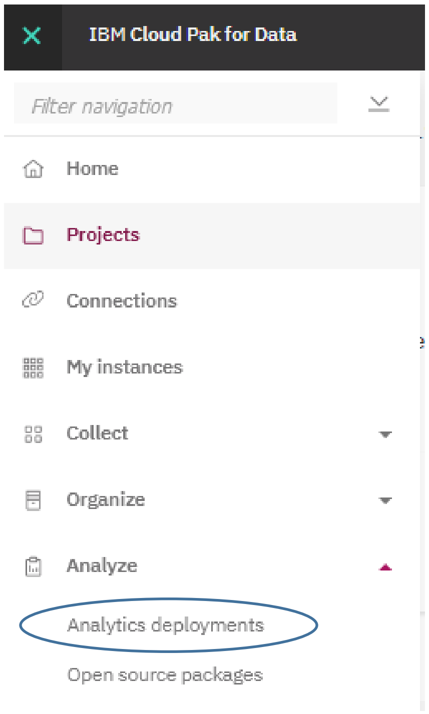


Data assets

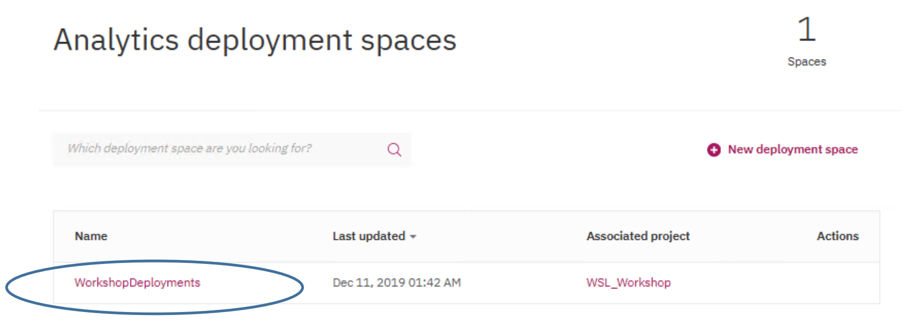
0 asset selected.

	NAME	TYPE	CREATED BY	LAST MODIFIED	ACTIONS
☐	CSV churn.csv	Data Asset	Elena Lowery	19 Dec 2019, 4:23:35 pm	
☐	CSV customer-profile.csv	Data Asset	Elena Lowery	19 Dec 2019, 4:23:26 pm	
☐	CSV new_customers.csv	Data Asset	Elena Lowery	16 Dec 2019, 5:46:06 pm	

10. From the *CP4D* main menu navigate to **Analytics Deployments**.



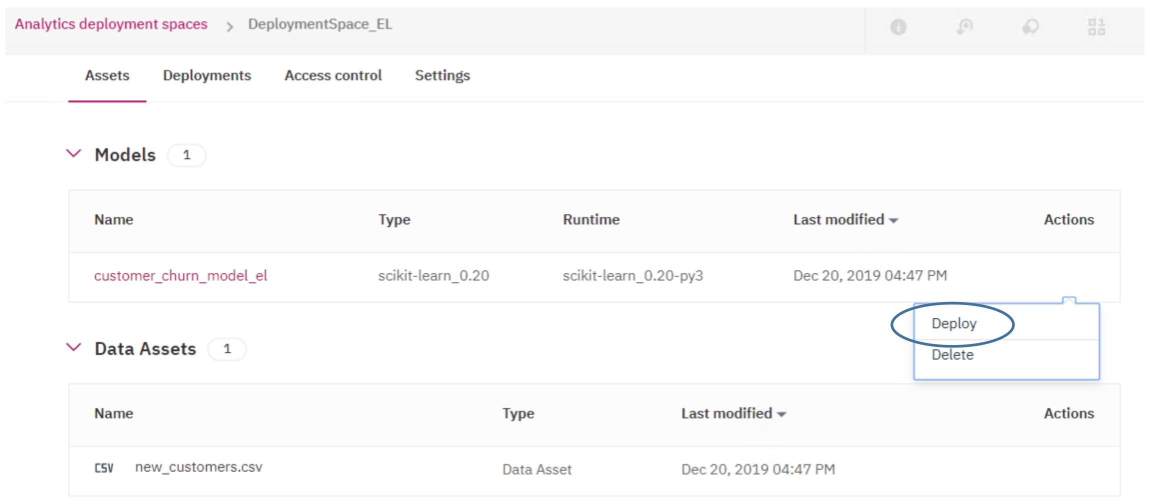
11. Click on your deployment space.


Analytics deployment spaces

1 Spaces

Name	Last updated	Associated project	Actions
WorkshopDeployments	Dec 11, 2019 01:42 AM	WSL_Workshop	

12. Click on the ellipsis under **Actions** and select **Deploy**.


Analytics deployment spaces > DeploymentSpace_EL

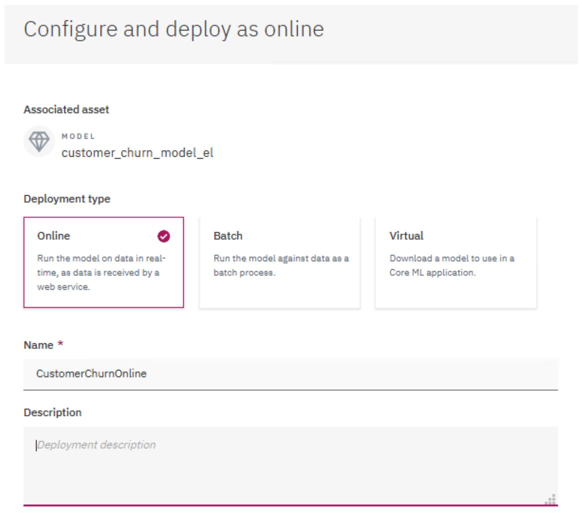
Models 1

Name	Type	Runtime	Last modified	Actions
customer_churn_model_el	scikit-learn_0.20	scikit-learn_0.20-py3	Dec 20, 2019 04:47 PM	

Data Assets 1

Name	Type	Last modified	Actions
CSV new_customers.csv	Data Asset	Dec 20, 2019 04:47 PM	

13. First, configure an *Online Deployment*. Select the **Online** box, provide deployment name and click **Create**.


Configure and deploy as online

Associated asset
MODEL
customer_churn_model_el

Deployment type

Online: Run the model on data in real-time, as data is received by a web service.

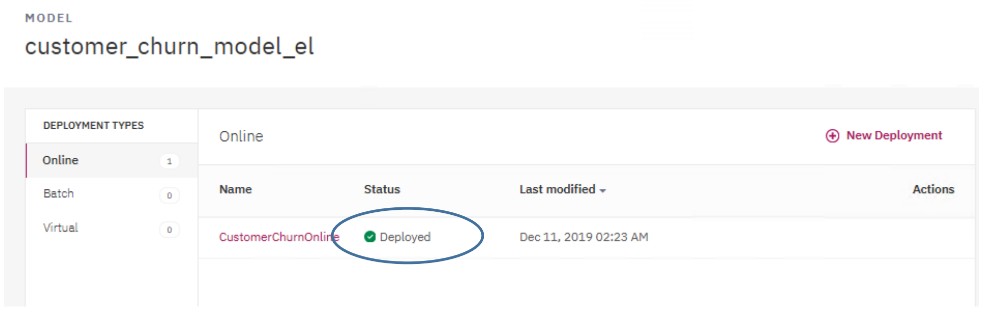
Batch: Run the model against data as a batch process.

Virtual: Download a model to use in a Core ML application.

Name *: CustomerChurnOnline

Description: Deployment description

14. After the deployment environment has been provisioned, you will see a success message and a green checkbox in the *Status* column.

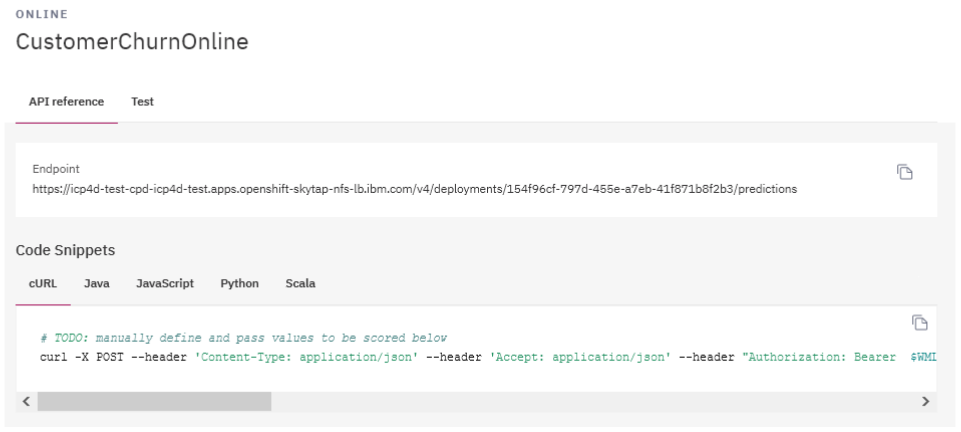


MODEL

customer_churn_model_el

Name	Status	Last modified	Actions
CustomerChurnOnline	Deployed	Dec 11, 2019 02:23 AM	

15. Click on the deployment to display sample invocation code and the test interface.



ONLINE

CustomerChurnOnline

Endpoint
https://icp4d-test-cpd-icp4d-test.apps.openshift-skytap-nfs-lb.ibm.com/v4/deployments/154f96cf-797d-455e-a7eb-41f871b8f2b3/predictions

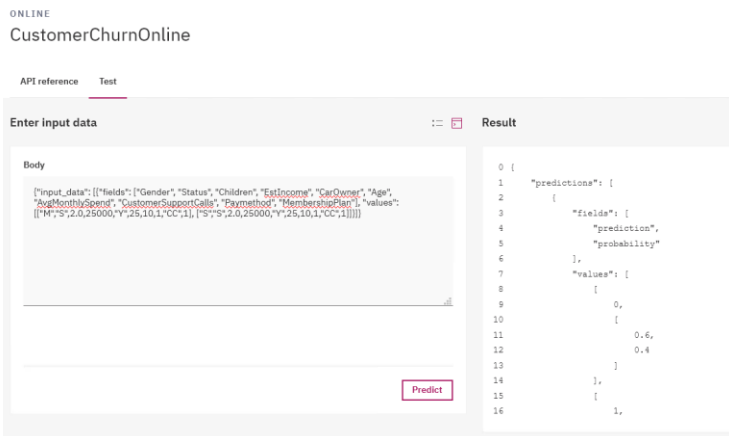
Code Snippets

```
# TODO: manually define and pass values to be scored below
curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json' --header "Authorization: Bearer  $WMI
```

16. Enter the following values for testing:

{"input_data": [{"fields": ["Gender", "Status", "Children", "EstIncome", "CarOwner", "Age", "AvgMonthlySpend", "CustomerSupportCalls", "Paymethod", "MembershipPlan"], "values": [["M","S",2.0,25000,"Y",25,10,1,"CC",1], ["S","S",2.0,25000,"Y",25,10,1,"CC",1]]}]}

You can copy this string from *data\SampleCustomerRecord.txt* file in of the unzipped git repo folder.



ONLINE

CustomerChurnOnline

```
{"input_data": [{"fields": ["Gender", "Status", "Children", "EstIncome", "CarOwner", "Age", "AvgMonthlySpend", "CustomerSupportCalls", "Paymethod", "MembershipPlan"], "values": [["M","S",2.0,25000,"Y",25,10,1,"CC",1], ["S","S",2.0,25000,"Y",25,10,1,"CC",1]]}]}
```

```
{
    "predictions": [
        {
            "fields": [
                "prediction",
                "probability"
            ],
            "values": [
                [
                    0,
                    [
                        0.6,
                        0.4
                    ]
                ],
                [
                    1,
```

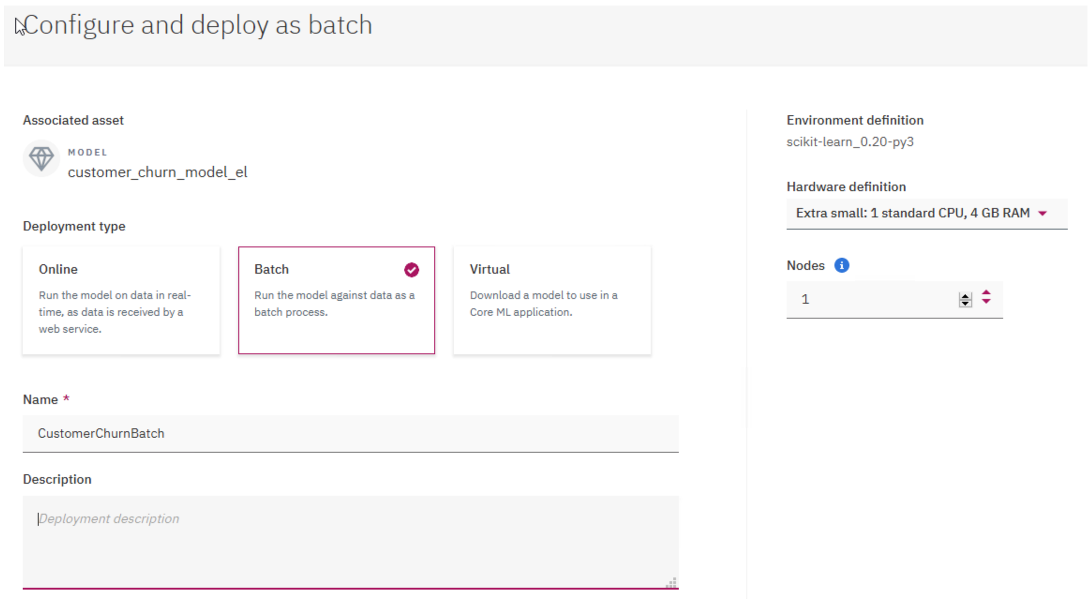
If you would like to test online scoring from a client outside of Watson Studio, you can use the code sample on the **API Reference** tab. You will also need to generate a token for authentication. You can find more information about token generation here: https://www.ibm.com/support/knowledgecenter/en/SSQNUZ_2.5.0/wsj/analyze-data/ml-authentication-local.html

You have finished configuring and testing an online deployment of an open source model.

Next, we will configure batch deployment. In the current release batch deployment supports only data assets (csv and other file types). The data assets must be in the same deployment space - that's why earlier in the lab we promoted *new_customers.csv* to our deployment space.

17. Navigate to the **Assets** page of the deployment space and click **Deploy** under **Actions**.

18. Select **Batch** and provide a *name* for the batch deployment.


Configure and deploy as batch

Associated asset
MODEL
customer_churn_model_el

Deployment type
- Online: Run the model on data in real-time, as data is received by a web service.
- Batch ✓: Run the model against data as a batch process.
- Virtual: Download a model to use in a Core ML application.

Name *
CustomerChurnBatch

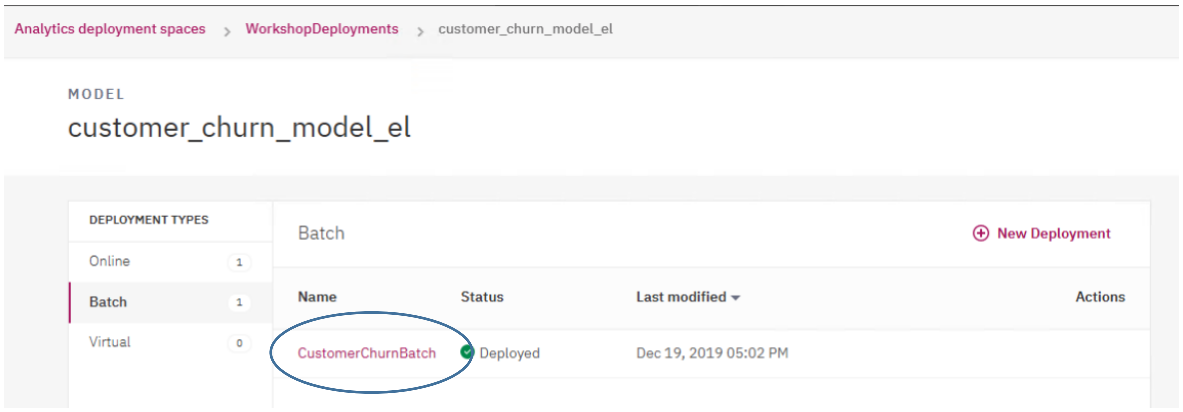
Description
Deployment description

Environment definition
scikit-learn_0.20-py3

Hardware definition
Extra small: 1 standard CPU, 4 GB RAM

Nodes
1

19. Click on the created batch deployment.


Analytics deployment spaces > WorkshopDeployments > customer_churn_model_el

MODEL
customer_churn_model_el

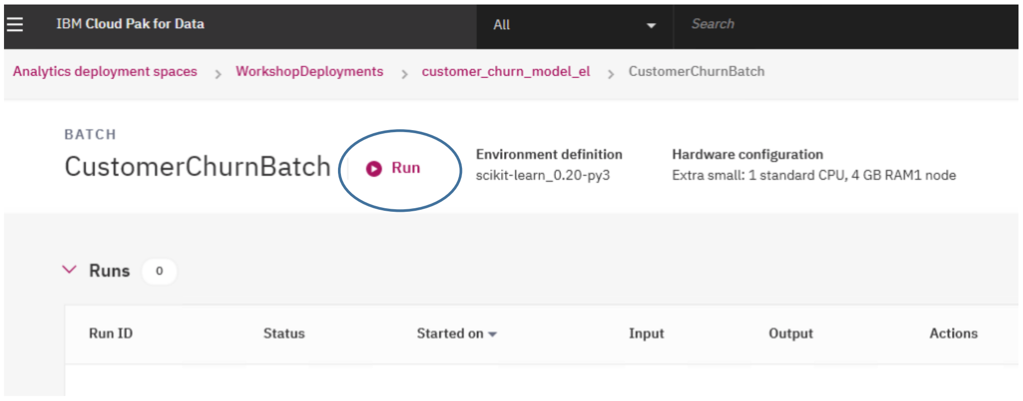
DEPLOYMENT TYPES	
Online	1
Batch	1
Virtual	0

Batch
New Deployment

Name	Status	Last modified	Actions
CustomerChurnBatch	Deployed	Dec 19, 2019 05:02 PM	

In the current release batch jobs can be invoked interactively in the deployment UI or with a REST API from an external asset. IBM is working on adding a scheduler to deployment spaces.

20. Click on the configured batch deployment, then click on the **Run** icon.



Analytics deployment spaces > WorkshopDeployments > customer_churn_model_el > CustomerChurnBatch

BATCH

CustomerChurnBatch Run

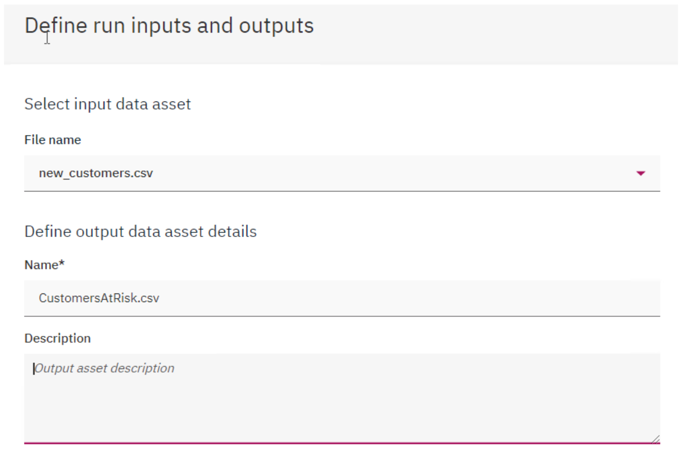
Environment definition: scikit-learn_0.20-py3

Hardware configuration: Extra small: 1 standard CPU, 4 GB RAM1 node

Runs 0

Run ID	Status	Started on	Input	Output	Actions

21. Select the input file name from the dropdown and provide an output file name. Click **Create**.



Define run inputs and outputs

Select input data asset

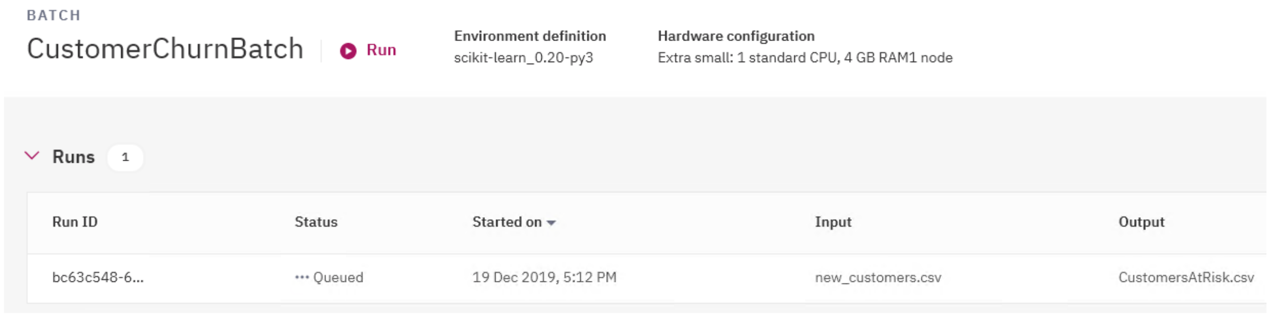
File name: new_customers.csv

Define output data asset details

Name*: CustomersAtRisk.csv

Description: Output asset description

22. Notice that the job status is *Queued*.



BATCH

CustomerChurnBatch Run

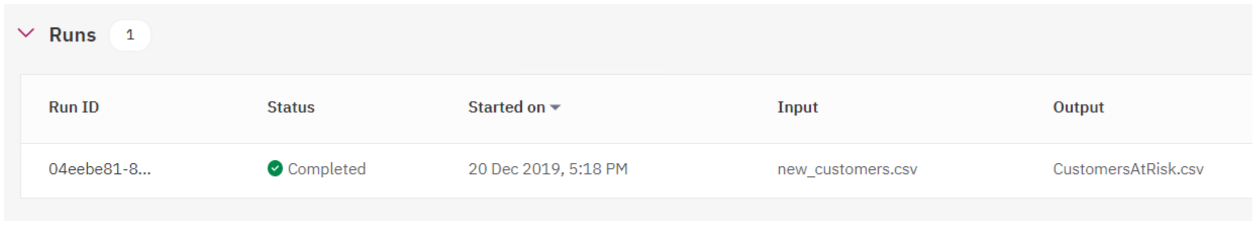
Environment definition: scikit-learn_0.20-py3

Hardware configuration: Extra small: 1 standard CPU, 4 GB RAM1 node

Runs 1

Run ID	Status	Started on	Input	Output
bc63c548-6...	••• Queued	19 Dec 2019, 5:12 PM	new_customers.csv	CustomersAtRisk.csv

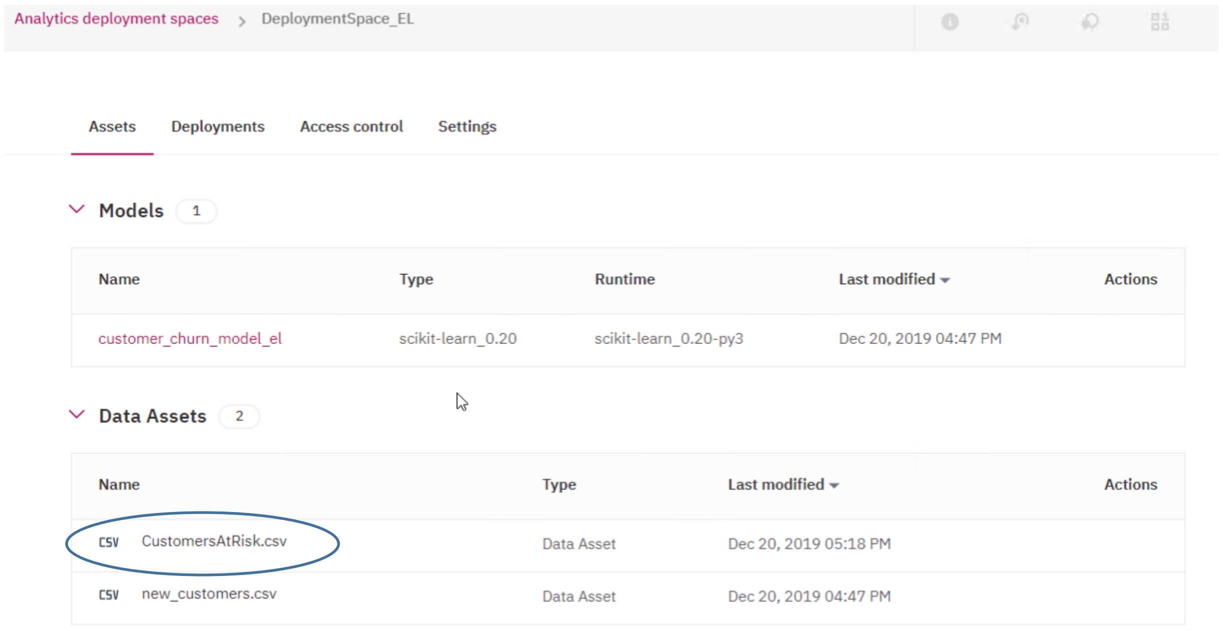
In a few seconds the job will complete and status will be updated.



Runs 1

Run ID	Status	Started on	Input	Output
04eebe81-8...	Completed	20 Dec 2019, 5:18 PM	new_customers.csv	CustomersAtRisk.csv

23. Navigate back to the deployment space view. The output file is now displayed under data assets.



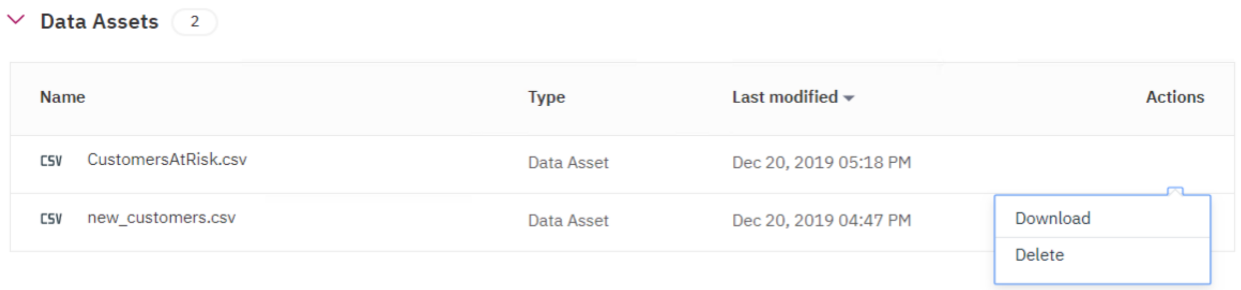
Models 1

Name	Type	Runtime	Last modified	Actions
customer_churn_model_el	scikit-learn_0.20	scikit-learn_0.20-py3	Dec 20, 2019 04:47 PM	

Data Assets 2

Name	Type	Last modified	Actions
CSV CustomersAtRisk.csv	Data Asset	Dec 20, 2019 05:18 PM	
CSV new_customers.csv	Data Asset	Dec 20, 2019 04:47 PM	

Important Note: *In this release the only action available for the output file is **Download**.*



Data Assets 2

Name	Type	Last modified	Actions
CSV CustomersAtRisk.csv	Data Asset	Dec 20, 2019 05:18 PM	
CSV new_customers.csv	Data Asset	Dec 20, 2019 04:47 PM	

You have finished configuring an open source model for batch scoring.

You can find additional documentation about deployment and scoring in the **CP4D Knowledge Center**: https://www.ibm.com/support/knowledgecenter/en/SSQNUZ_2.5.0/wsj/wmls/wmls-deploy-python.html#deploy-batch

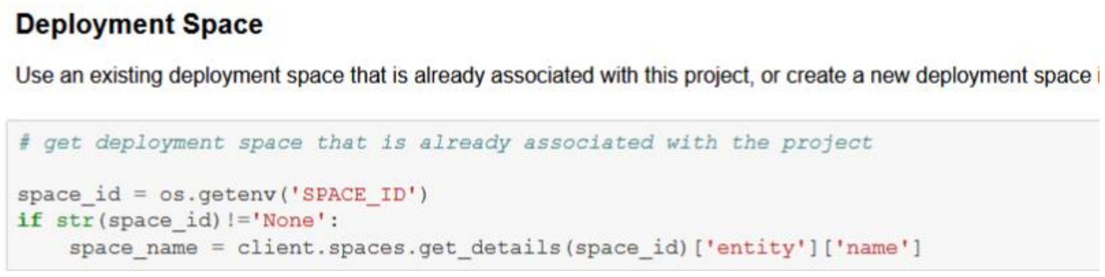
Next, we will review a notebook that completes the steps programmatically.

24. Navigate to the project view and open the *DeploymentWithAPI* notebook.

    We recommend that you save this notebook as a reference, you don't have to run it as a part of the lab.

    The *DeploymentWithAPI* notebook is the same as the notebook that we previously reviewed, but it has additional steps to publish to the deployment space and configure deployments.

    Under **Step 7**, find the section that saves a model into a deployment space.



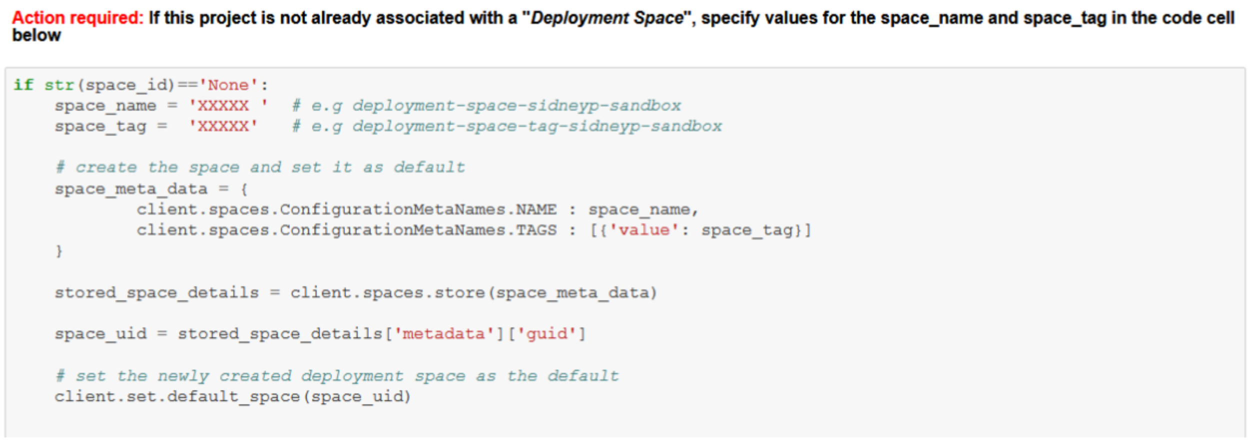
Deployment Space

Use an existing deployment space that is already associated with this project, or create a new deployment space i

```
# get deployment space that is already associated with the project

space_id = os.getenv('SPACE_ID')
if str(space_id)!='None':
    space_name = client.spaces.get_details(space_id)['entity']['name']
```

This sample code first checks if a deployment space is already associated with a project, and if it is, it saves the model into that deployment space.

Important Note: *in this release WML API allows creating a deployment space with the same name. If you run a cell which creates a deployment space several times, you will get several deployment spaces with the same name.*

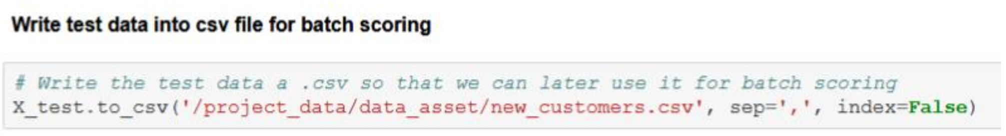


Action required: **If this project is not already associated with a "*Deployment Space*", specify values for the space_name and space_tag in the code cell below**

```
if str(space_id)=='None':
    space_name = 'XXXXX '   # e.g deployment-space-sidneyp-sandbox
    space_tag =  'XXXXX'    # e.g deployment-space-tag-sidneyp-sandbox

    # create the space and set it as default
    space_meta_data = {
            client.spaces.ConfigurationMetaNames.NAME : space_name,
            client.spaces.ConfigurationMetaNames.TAGS : [{'value': space_tag}]
    }

    stored_space_details = client.spaces.store(space_meta_data)

    space_uid = stored_space_details['metadata']['guid']

    # set the newly created deployment space as the default
    client.set.default_space(space_uid)
```

The notebook also provides examples of configuring a deployment and testing it.

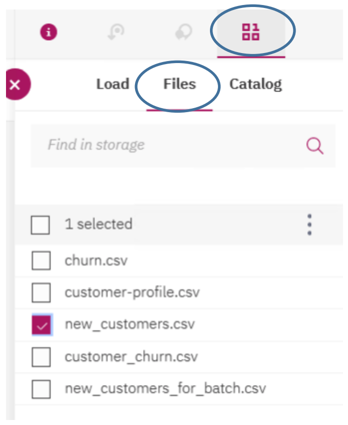
The last cell of the notebook saves the data that was used for testing as a csv file. This step is useful when you create demos because you can use this file to configure batch scoring.



Write test data into csv file for batch scoring

```
# Write the test data a .csv so that we can later use it for batch scoring
X_test.to_csv('/project_data/data_asset/new_customers.csv', sep=',', index=False)
```

Note: When using this API, new_customers.csv will be saved in the Files view. You will need to add it to Data Assets manually.

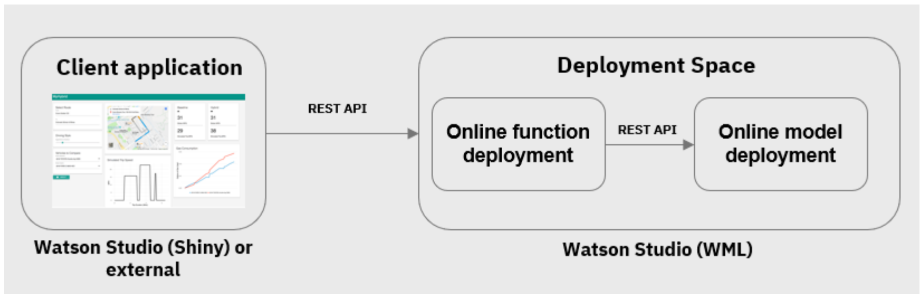


Next, we'll review deployment of *functions*.

Deployment of functions is an advanced topic that requires the knowledge of *WML Client API*. It's important to understand this capability because it allows us to create *deployment pipelines*. A deployment pipeline can contain multiple steps, not just scoring of the model. Most deployment pipelines contain data validation and preparation steps.

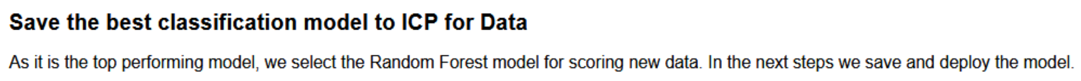
We will not write code in this section of the lab. Instead, we'll review code in one of the *industry accelerators* that are available on CP4D. An accelerator contains several assets, such as notebooks and sample data files. You can learn more about industry accelerators here:
https://www.ibm.com/support/knowledgecenter/en/SSQNUZ_2.5.0/cpd/svc/industry-accel-svc.html

This diagram shows the deployment of the function in the accelerator.



25. Open the *Customer Attrition* project that you imported earlier in the lab. Open the model training notebook, *1-model_training* in *Edit* mode.

Run the notebook until the *Save the best classification model to ICP for Data* section.

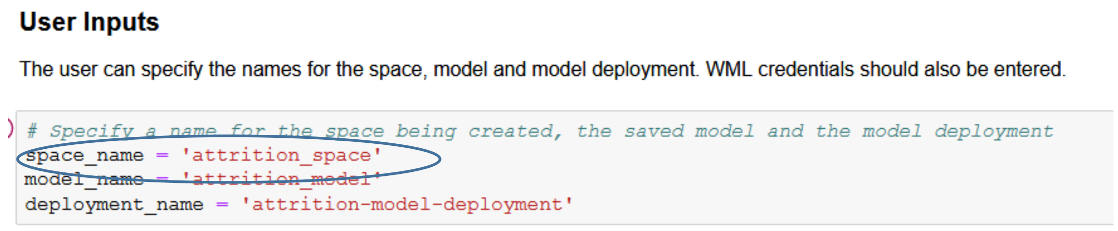

Save the best classification model to ICP for Data

As it is the top performing model, we select the Random Forest model for scoring new data. In the next steps we save and deploy the model.

Review the code that saves the model into the deployment space.

Notice that the code in this notebook defines the deployment space and creates deployments. These steps are completed programmatically to simplify the setup of the accelerator. A customer can modify this notebook to save the model into the project, and complete the rest of the steps manually, like we've done in the beginning of this lab.

26. Change the deployment space to a unique name.


User Inputs

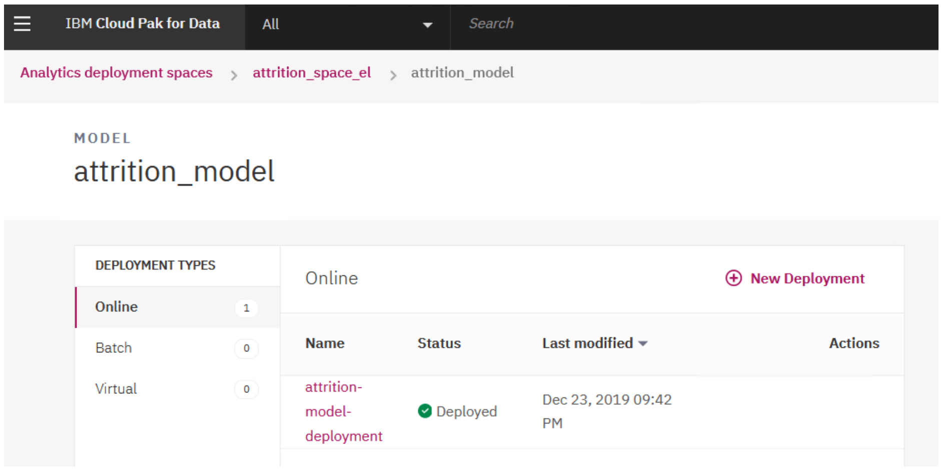
The user can specify the names for the space, model and model deployment. WML credentials should also be entered.

```
# Specify a name for the space being created, the saved model and the model deployment
space_name = 'attrition_space'
model_name = 'attrition_model'
deployment_name = 'attrition-model-deployment'
```

As mentioned earlier, while you will not get an error if the same deployment space is used by multiple users, it will be difficult to find your deployment space.

27. Run the remaining cells in the notebook.

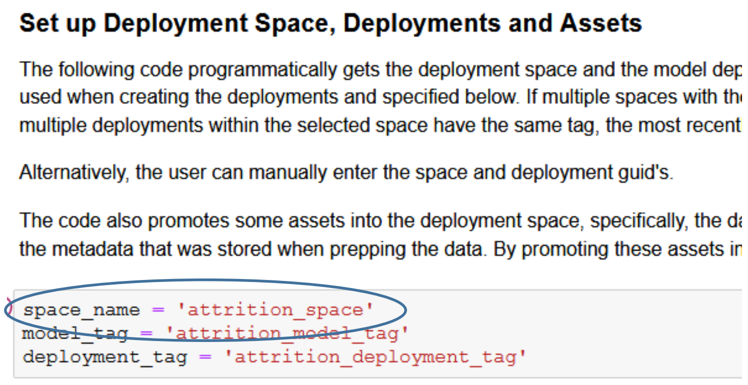
28. Navigate to the **Analytics Deployment** view to verify that the model was saved and that deployments were created.



Name	Status	Last modified	Actions
attrition-model-deployment	Deployed	Dec 23, 2019 09:42 PM	

29. Navigate back to the project view and open the model scoring notebook, *2-model_scoring*, in *Edit* mode.

30. Change the deployment space name to the name that you specified in the previous step.



Set up Deployment Space, Deployments and Assets

The following code programmatically gets the deployment space and the model dep used when creating the deployments and specified below. If multiple spaces with th multiple deployments within the selected space have the same tag, the most recent

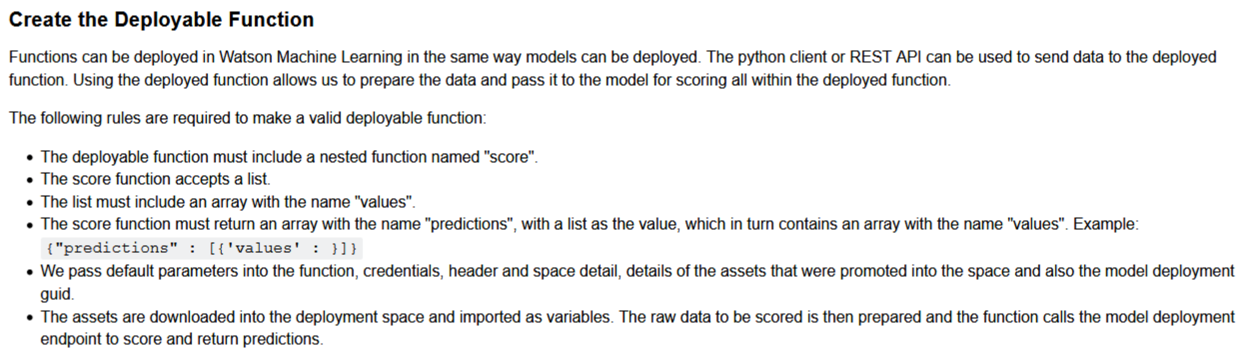
Alternatively, the user can manually enter the space and deployment guid's.

The code also promotes some assets into the deployment space, specifically, the da the metadata that was stored when prepping the data. By promoting these assets in

```
space_name = 'attrition_space'
model_tag = 'attrition_model_tag'
deployment_tag = 'attrition_deployment_tag'
```

31. Review the *Create the Deployable Function* section.

    This section explains the specific rules for writing a function that can be deployed in Watson Studio.



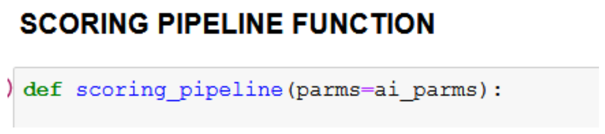
Create the Deployable Function

Functions can be deployed in Watson Machine Learning in the same way models can be deployed. The python client or REST API can be used to send data to the deployed function. Using the deployed function allows us to prepare the data and pass it to the model for scoring all within the deployed function.

The following rules are required to make a valid deployable function:

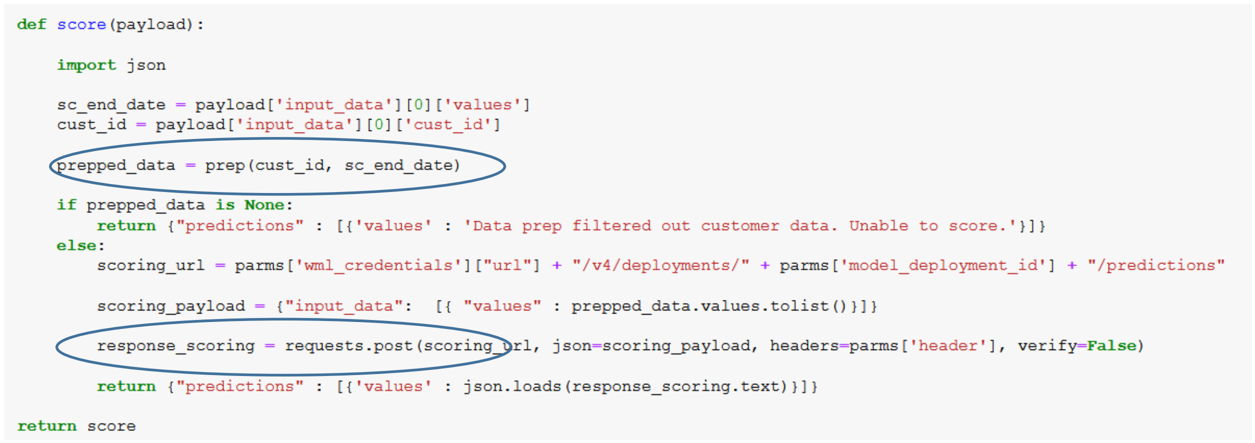
- The deployable function must include a nested function named "score".
- The score function accepts a list.
- The list must include an array with the name "values".
- The score function must return an array with the name "predictions", with a list as the value, which in turn contains an array with the name "values". Example: `{"predictions" : [{'values' : }]}`
- We pass default parameters into the function, credentials, header and space detail, details of the assets that were promoted into the space and also the model deployment guid.
- The assets are downloaded into the deployment space and imported as variables. The raw data to be scored is then prepared and the function calls the model deployment endpoint to score and return predictions.

The main implementation of the function is in the **SCORING PIPELINE FUNCTION** cell: *scoring_pipeline* is the nested function that performs data preparation and invokes the *score()* function, which, in turn, calls the model deployed in the model training notebook.



SCORING PIPELINE FUNCTION

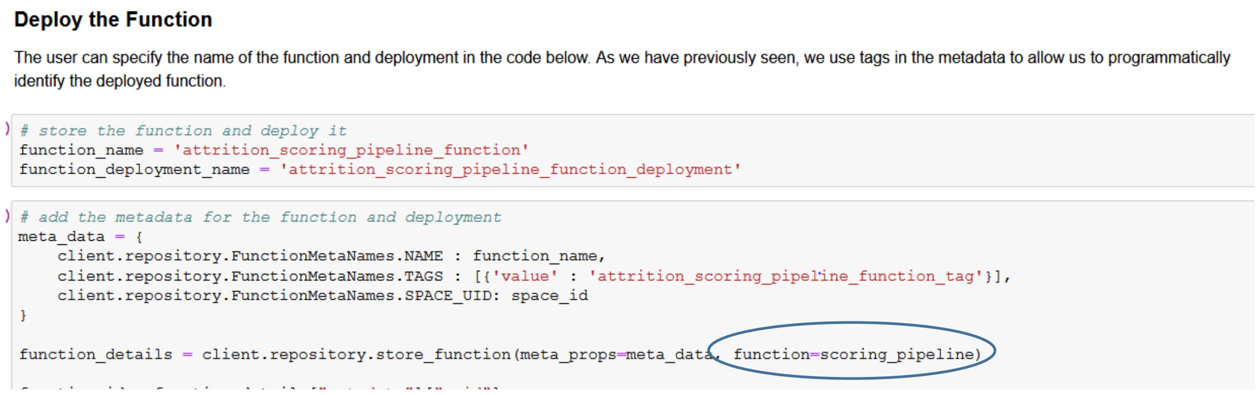
```
def scoring_pipeline(parms=ai_parms):
```



```
def score(payload):

    import json

    sc_end_date = payload['input_data'][0]['values']
    cust_id = payload['input_data'][0]['cust_id']

    prepped_data = prep(cust_id, sc_end_date)

    if prepped_data is None:
        return {"predictions" : [{'values' : 'Data prep filtered out customer data. Unable to score.'}]}
    else:
        scoring_url = parms['wml_credentials']["url"] + "/v4/deployments/" + parms['model_deployment_id'] + "/predictions"

        scoring_payload = {"input_data":  [{ "values" : prepped_data.values.tolist()}]}

        response_scoring = requests.post(scoring_url, json=scoring_payload, headers=parms['header'], verify=False)

        return {"predictions" : [{'values' : json.loads(response_scoring.text)}]}

return score
```

32. Review the code that deploys the function.

Notice that we have to provide function name as one of the parameters.



Deploy the Function

The user can specify the name of the function and deployment in the code below. As we have previously seen, we use tags in the metadata to allow us to programmatically identify the deployed function.

```
# store the function and deploy it
function_name = 'attribution_scoring_pipeline_function'
function_deployment_name = 'attrition_scoring_pipeline_function_deployment'
```

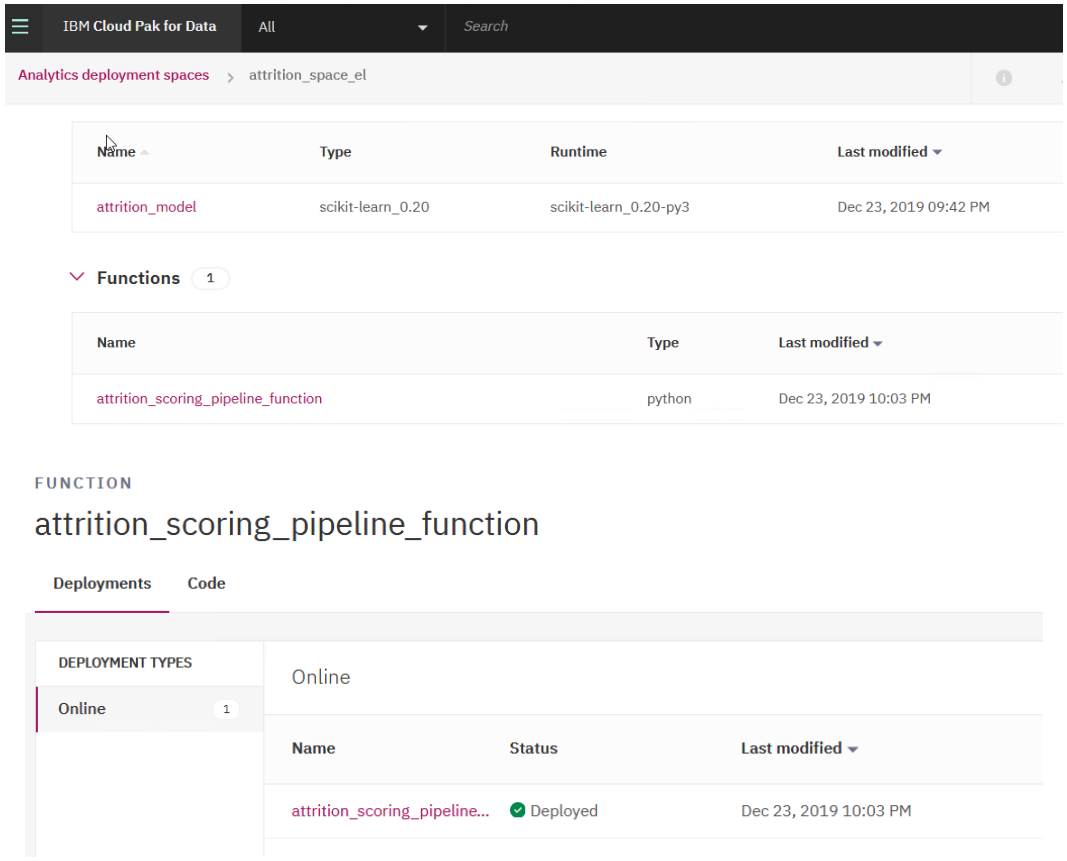
```
# add the metadata for the function and deployment
meta_data = {
    client.repository.FunctionMetaNames.NAME : function_name,
    client.repository.FunctionMetaNames.TAGS : [{'value' : 'attrition_scoring_pipeline_function_tag'}],
    client.repository.FunctionMetaNames.SPACE_UID: space_id
}

function_details = client.repository.store_function(meta_props=meta_data, function=scoring_pipeline)
```

33. Run the notebook.

34. Test deployment of the function.

Note: The notebook states that an R Shiny application can be used to invoke the deployed function. R Studio may not be installed on the CP4D cluster that you're using for the lab. If R Studio is installed, you can test model scoring from the R application.

- Navigate to the **Analytics Deployment** view to verify that the function was saved and deployed



Name	Type	Runtime	Last modified
attrition_model	scikit-learn_0.20	scikit-learn_0.20-py3	Dec 23, 2019 09:42 PM

Functions 1

Name	Type	Last modified
attrition_scoring_pipeline_function	python	Dec 23, 2019 10:03 PM

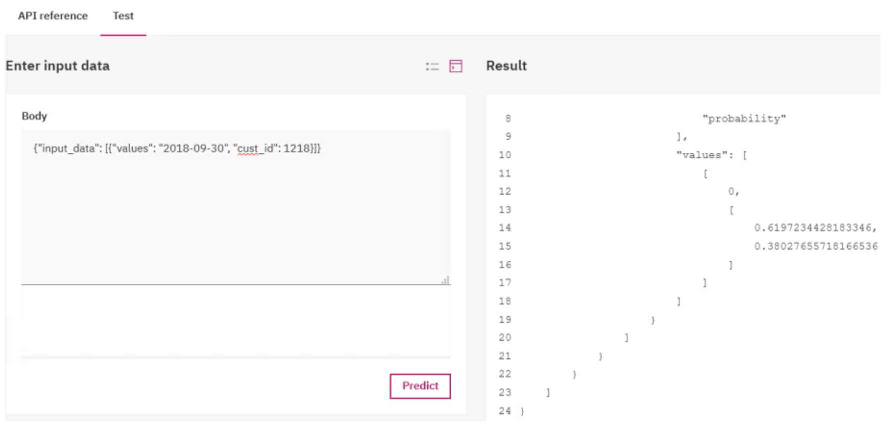
FUNCTION

attrition_scoring_pipeline_function

Name	Status	Last modified
attrition_scoring_pipeline...	Deployed	Dec 23, 2019 10:03 PM

- Click on the function deployment to bring up the **Test** tab.
- You can test the function with the following values:

{"input_data": [{"values": "2018-09-30", "cust_id": 1218}]}



{"input_data": [{"values": "2018-09-30", "cust_id": 1218}]}

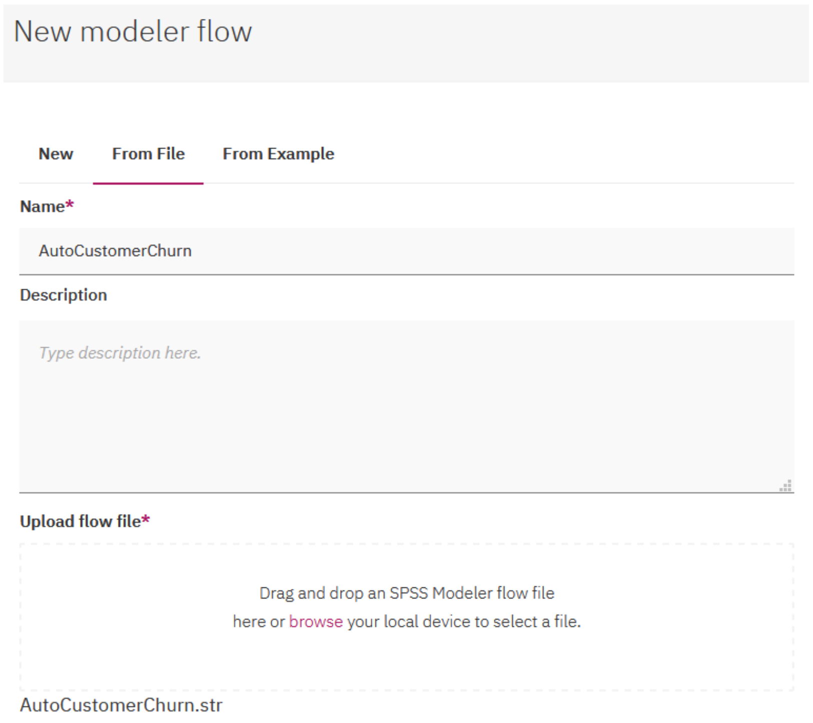
```
                            "probability"
                        ],
                        "values": [
                            [
                                0,
                                [
                                    0.6197234428183346,
                                    0.3802765571816653
                                ]
                            ]
                        ]
                    }
                ]
            }
        }
    ]
}
```

You have finished deploying and testing a scoring pipeline function.

Next, we will deploy as SPSS flow.

35. Navigate to the original *WSL_Workshop* project (created in Step 1) view and add a Modeler flow from file.

    - Select **Add to Project -> Modeler flow**
    - Navigate to the unzipped *git repo* folder and select *CustomerChurn.str* from the *Flows* folder.
    - Click **Create**.


New modeler flow

New | From File | From Example

Name*
AutoCustomerChurn

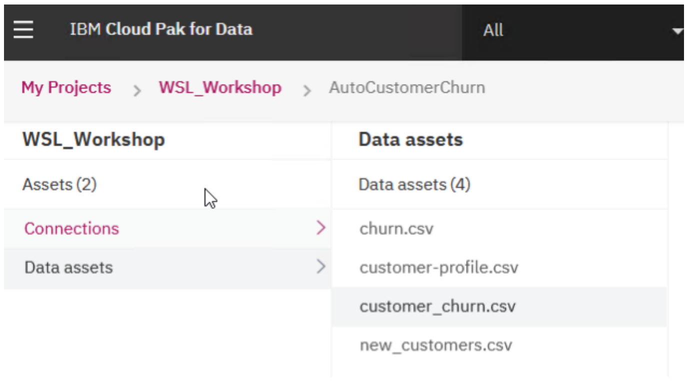
Description
Type description here.

Upload flow file*
Drag and drop an SPSS Modeler flow file here or browse your local device to select a file.

AutoCustomerChurn.str

36. Double click on the input data asset.

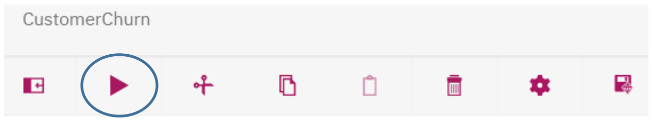
37. Click **Change data asset** in the node properties view, then select *customer_churn.csv*. Make sure to click **Save** in node properties before closing it.



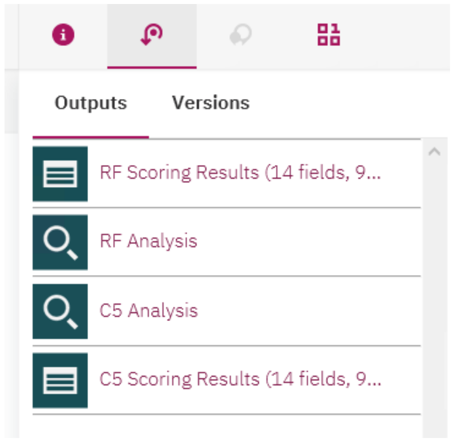
Let’s review this Modeler flow before we deploy it.

Similar to the notebook we reviewed earlier in the lab, the flow builds models to predict customer churn. The flow builds 2 models – a *C5* and a *Random Forest* model.

38. Click the **Run** icon in the menu bar.



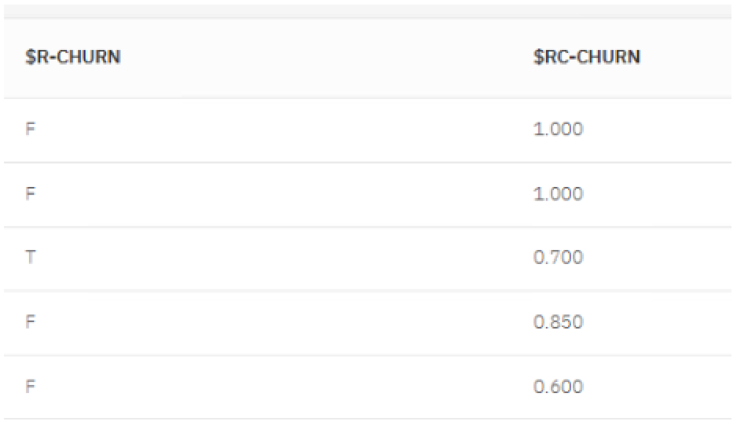
39. Click on the **Output** icon to review model evaluation (*Analysis* nodes) and output (*Table* nodes)


Outputs
Versions
RF Scoring Results (14 fields, 9...
RF Analysis
C5 Analysis
C5 Scoring Results (14 fields, 9...

The *random forest* model performed slightly better than the *C5* model. We will use it for deployment.

40. Double click on *RF Scoring Results* output.

When we invoke model scoring, the two values that start with the *$* will be returned*. $R-CHURN* is the prediction (true of false), and *$RC-CHURN* is the confidence in the prediction. The results will be returned for each row of input data.

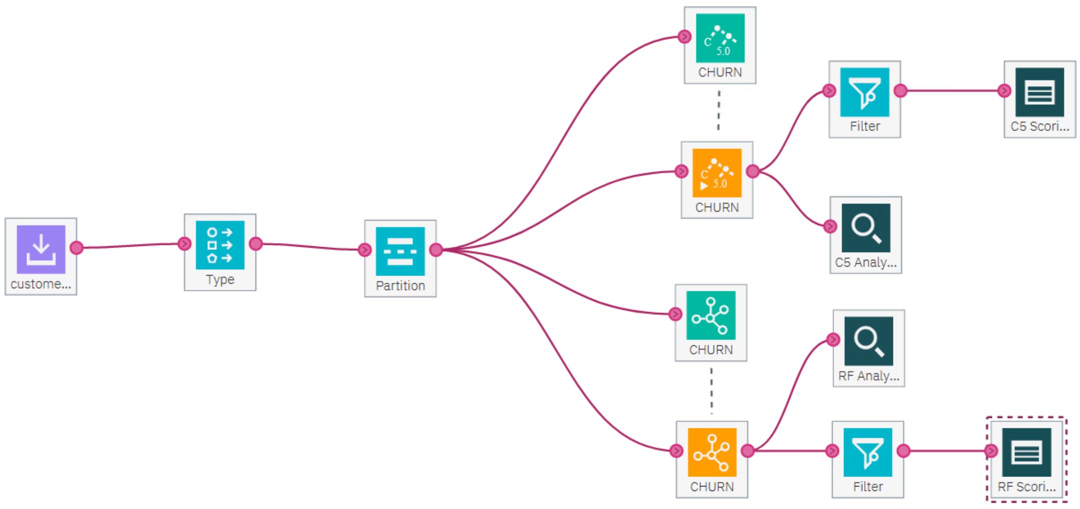


$R-CHURN	$RC-CHURN
F	1.000
F	1.000
T	0.700
F	0.850
F	0.600

Modeler flows have a concept of *branches*. You can identify the number of branches by counting the end nodes. In our example we have *4 branches*.

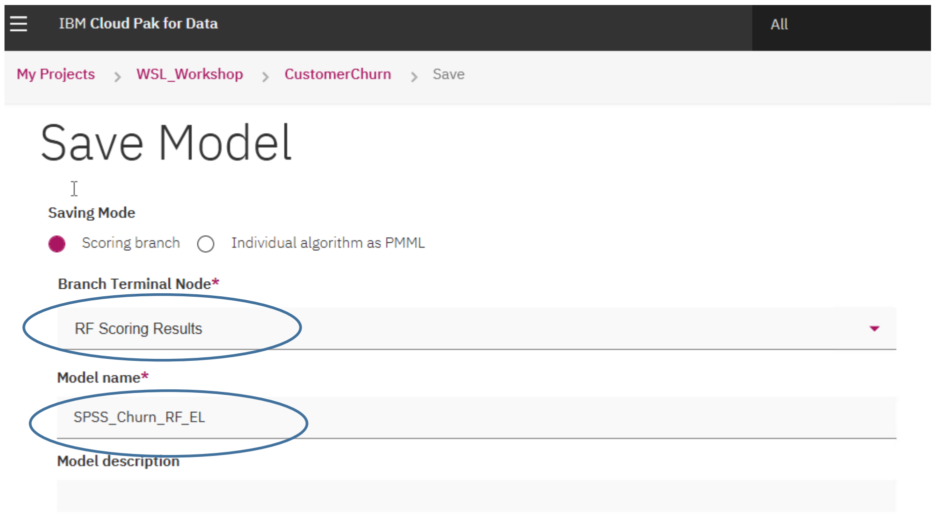
When we deploy a flow, we need to specify which branch we want to deploy. Deploying a branch means all the previous nodes connected to the selected node will be deployed as one pipeline. The nodes that are not applicable for scoring - for example, *Partition*, model building (algorithm), display nodes (*Graph* and *Table*) - will not run when deployed for scoring.

Important Note: *When you're not sure if an end node will run as a part of the scoring flow, add a Table node to it. For deployment, select a branch that ends with that Table node. The Table node will not run in deployment because it does not perform a "scoring function" – it's a "display only" node.*



41. Right mouse click on the *RF Scoring Results* node and select **Save branch as a model**.

42. The branch terminal node is pre-selected. Provide a unique model name and click **Create**.



Save Model

Saving Mode

[✓ Scoring branch] Individual algorithm as PMML

Branch Terminal Node*

RF Scoring Results

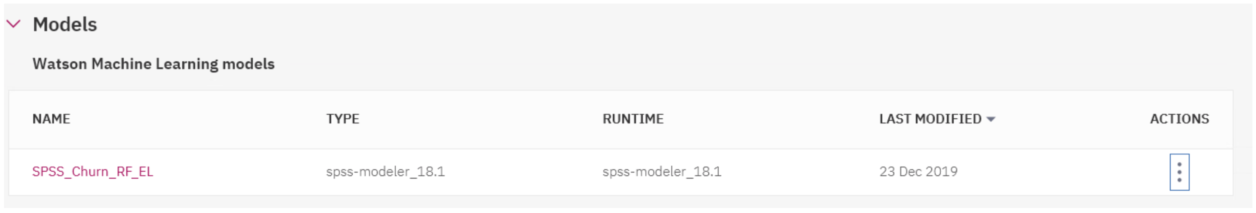
Model name*

SPSS_Churn_RF_EL

Model description

43. Navigate back to the project **Assets** view.

44. Click on the **Actions** menu and select **Promote**.



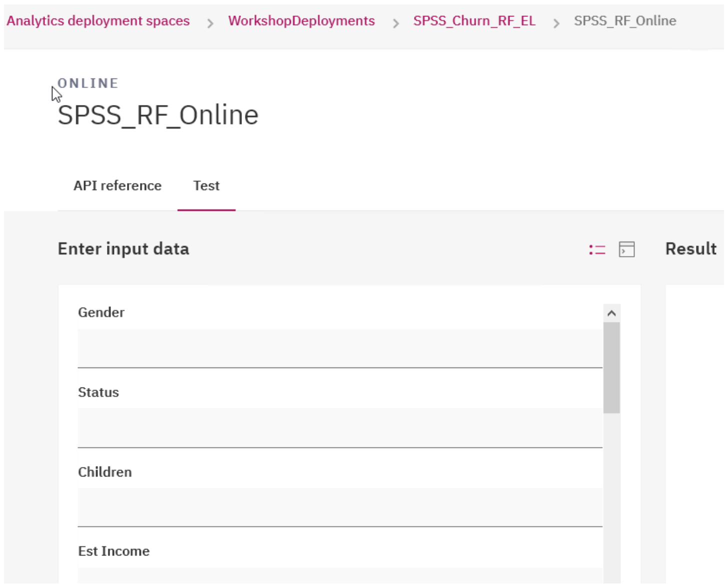
Models

Watson Machine Learning models

NAME	TYPE	RUNTIME	LAST MODIFIED	ACTIONS
SPSS_Churn_RF_EL	spss-modeler_18.1	spss-modeler_18.1	23 Dec 2019	⋮

45. Navigate to the deployment space associated with your projects and create an *online* deployment.

46. Click on the configured deployment to bring up the **Test** tab.

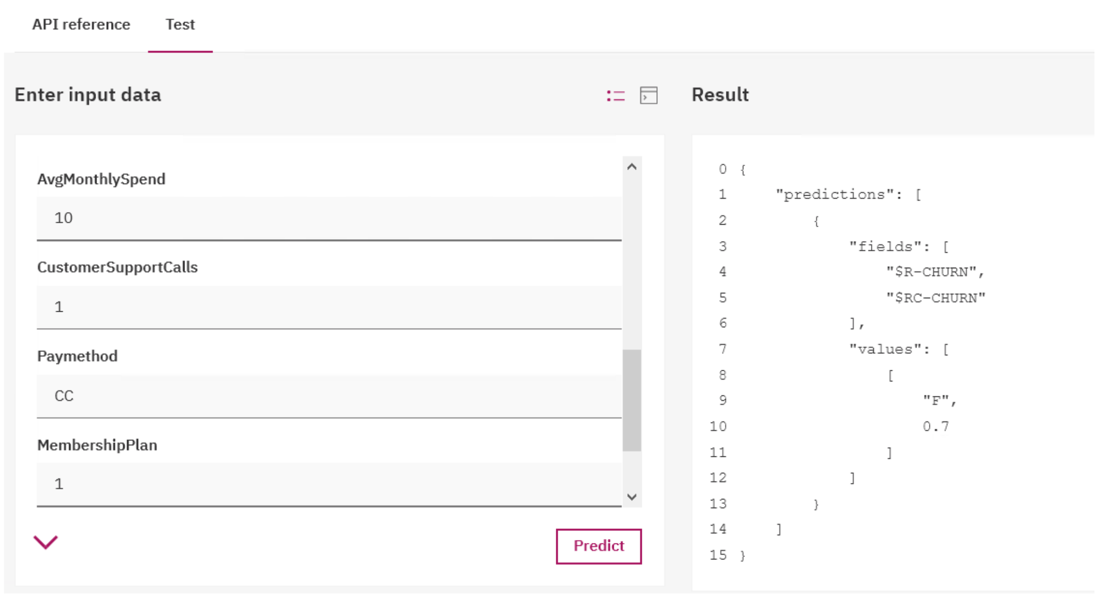


47. You can use the following values for testing:

Gender: M
Status: S
Children: 2
Est Income: 25000
Car Owner: Y
Age: 25
AvgMonthlySpend: 10
CustomerSupportCalls: 1
Paymethod: CC
MembershipPlan: 1

Do not specify the CHURN value.

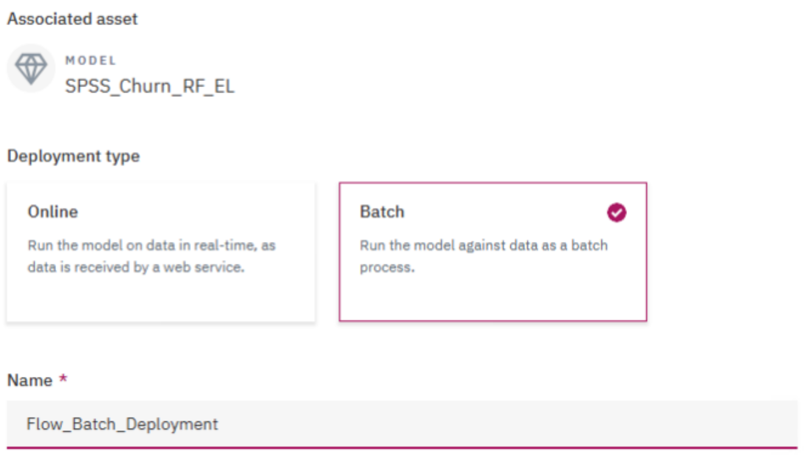
Scoring results should be similar to this output.



AvgMonthlySpend: 10
CustomerSupportCalls: 1
Paymethod: CC
MembershipPlan: 1

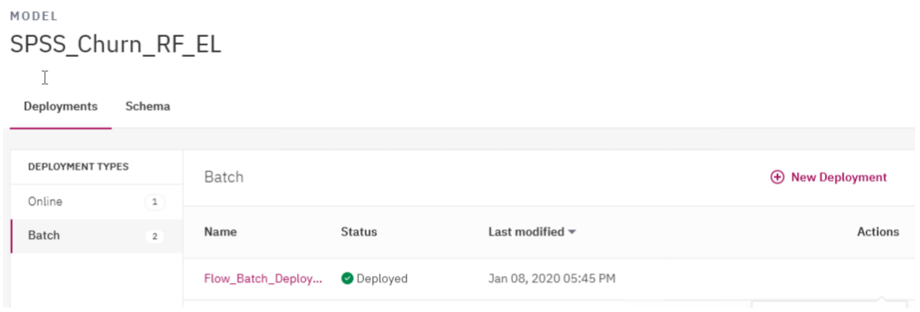
```
{
    "predictions": [
        {
            "fields": [
                "$R-CHURN",
                "$RC-CHURN"
            ],
            "values": [
                [
                    "F",
                    0.7
                ]
            ]
        }
    ]
}
```

48. Create a batch deployment for the same model.


Associated asset
MODEL
SPSS_Churn_RF_EL
Deployment type
Online
Run the model on data in real-time, as data is received by a web service.
Batch
Run the model against data as a batch process.

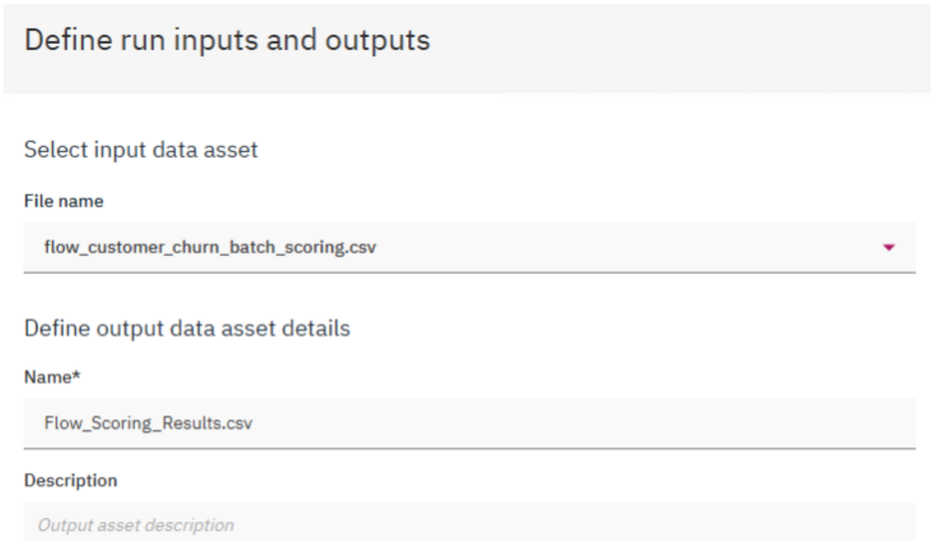
Name *: Flow_Batch_Deployment

49. Click on the created batch deployment and configure a *Run*.



Name	Status	Last modified	Actions
Flow_Batch_Deploy...	Deployed	Jan 08, 2020 05:45 PM	

Use the *flow_customer_churn_batch_scoring.csv* file as the input file for scoring.


Define run inputs and outputs

Select input data asset

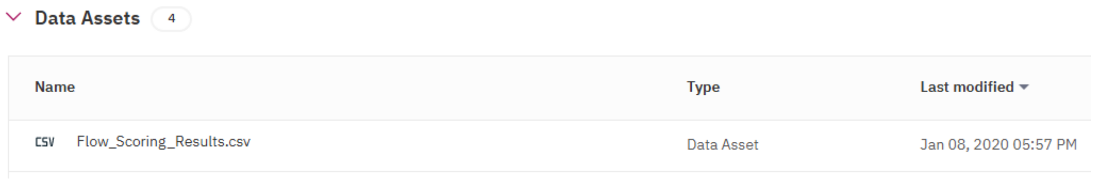
File name: flow_customer_churn_batch_scoring.csv

Define output data asset details

Name*: Flow_Scoring_Results.csv

Description: Output asset description

50. Verify that the output file was generated. If you wish, you can download and review the output file.


Data Assets 4

Name	Type	Last modified
CSV Flow_Scoring_Results.csv	Data Asset	Jan 08, 2020 05:57 PM

You have finished deploying of an SPSS model.

Next, we will deploy PMML.

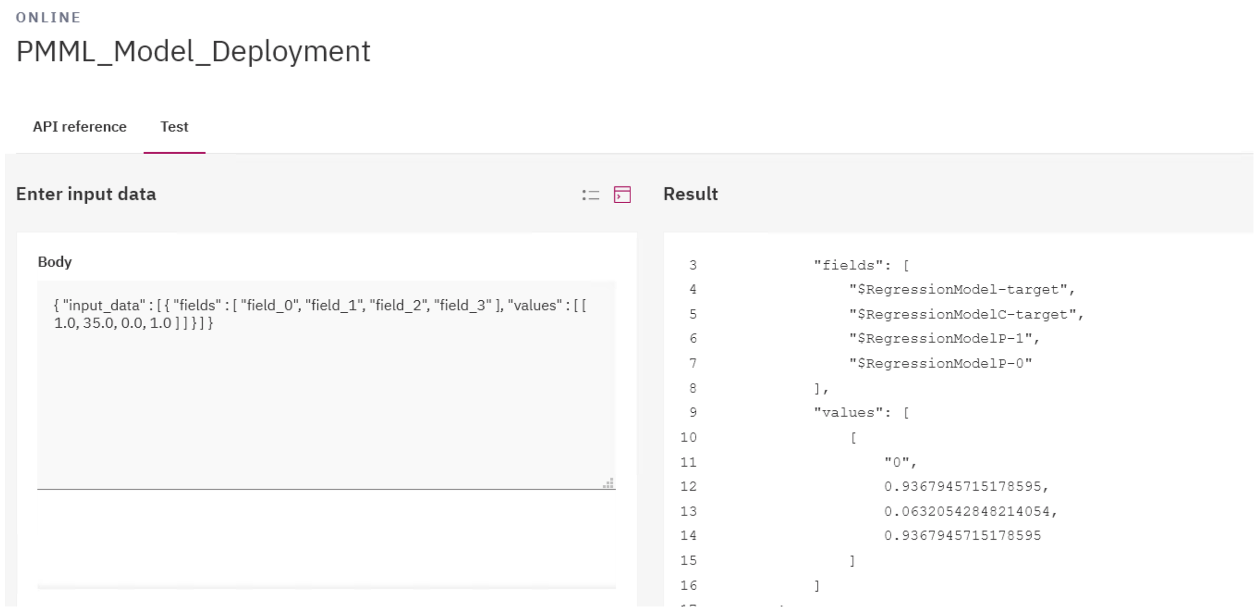
51. In Watson Studio open the imported WSL Workshop project, then open the *PMML_Example* notebook in *Edit* mode.

52. Review the notebook and run it.

    As indicated in the notebook, you can deploy the model manually after it has been saved in the deployment space or run through all the cells that create deployment automatically.

    If you choose to deploy the model manually, you can use the following values for testing:

    { "input_data" : [{ "fields" : ["field_0", "field_1", "field_2", "field_3"], "values" : [[1.0, 35.0, 0.0, 1.0]] }] }


ONLINE

PMML_Model_Deployment

Enter input data

Body

```
{ "input_data" : [ { "fields" : [ "field_0", "field_1", "field_2", "field_3" ], "values" : [ [
1.0, 35.0, 0.0, 1.0 ] ] } ] }
```

Result

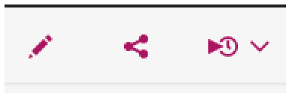
```
        "fields": [
            "$RegressionModel-target",
            "$RegressionModelC-target",
            "$RegressionModelP-1",
            "$RegressionModelP-0"
        ],
        "values": [
            [
                "0",
                0.9367945715178595,
                0.06320542848214054,
                0.9367945715178595
            ]
        ]
```

You have finished PMML model deployment.

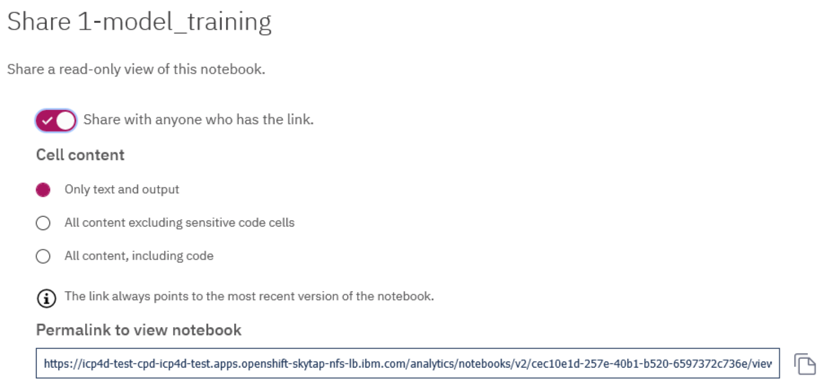
Deployment of Notebooks

In this section we will deploy a notebook as a URL.

1. Open the *customer attrition* project.
2. Open *the 1_model_training* notebook by clicking on it (it does not need to be in *Edit* mode).
3. Click on the *share* icon in the menu bar.



4. Select the preferred sharing options and copy the URL of the shared notebook.



Share 1-model_training

Share a read-only view of this notebook.

[✓] Share with anyone who has the link.

Cell content

[✓ Only text and output]

All content excluding sensitive code cells

All content, including code

The link always points to the most recent version of the notebook.

Permalink to view notebook

https://icp4d-test-cpd-icp4d-test.apps.openshift-skytap-nfs-lb.ibm.com/analytics/notebooks/v2/cec10e1d-257e-40b1-b520-6597372c736e/viev

Note: While "data exploration" and "results" notebooks may seem like a more obvious choice for this type of deployment, sharing model training notebooks may be a good fit for customers who are looking for more transparency in data science. The notebook that we have chosen provides useful information about model training.

5. Paste the copied URL into another browser (a browser that hasn't cached your login userid).

   Notice that log in is not required if you selected *Share with anyone who has the link*.

   Note: as stated on in the sharing configuration window, the notebooks are published as "read-only", i.e. the user will not be able to run notebook cells.

You have finished the Deployment section of the lab.

Part 2: Collaboration and Git Integration

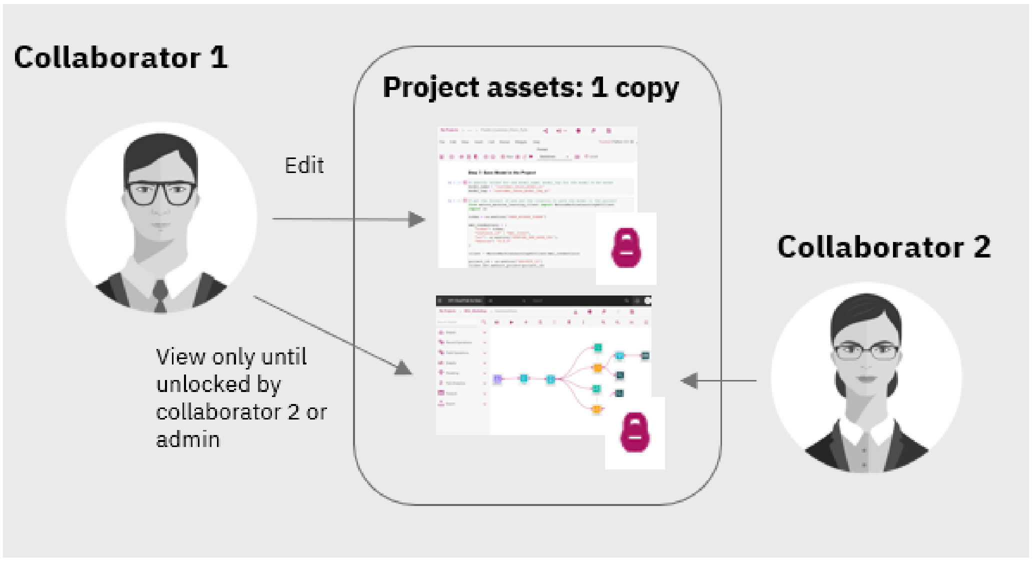
Watson Studio 2.1 supports three modes of collaboration:

- **Option 1:** Local collaboration (no Git)
- **Option 2**: Collaboration via Git for all assets with the exception of JupyterLab
- **Option 3:** JupyterLab collaboration with Git

Let's review collaboration options in more detail.

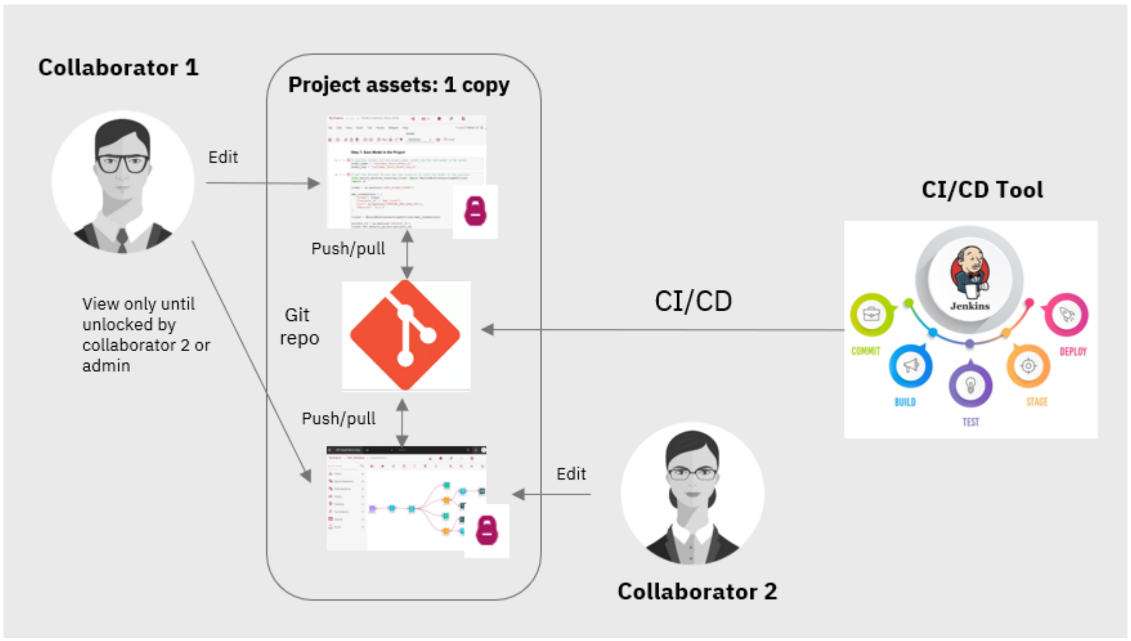
When we work in *local collaboration* mode, all collaborators work on one copy of assets in the project. When a user works on an asset, for example, a notebook, it becomes locked. Only the current user or the admin of the project can unlock the asset. Since only one version of the asset exists, changes are immediately available to all collaborators. Local collaboration mode is enabled when we create a project **without** a connection to Git.

Figure 1: Option 1 – local collaboration



The second option is *collaboration via Git for all assets with the exception of JupyterLab*. JupyterLab is an exception because it has its own Git integration that works differently than integration for other assets in the project. This option is enabled when we connect the project to a Git repo. Once the project is connected, we can use the **Pull** and the **Push and Pull** options to synchronize with the repo.

Figure 2: Option 2 - Collaboration with Git



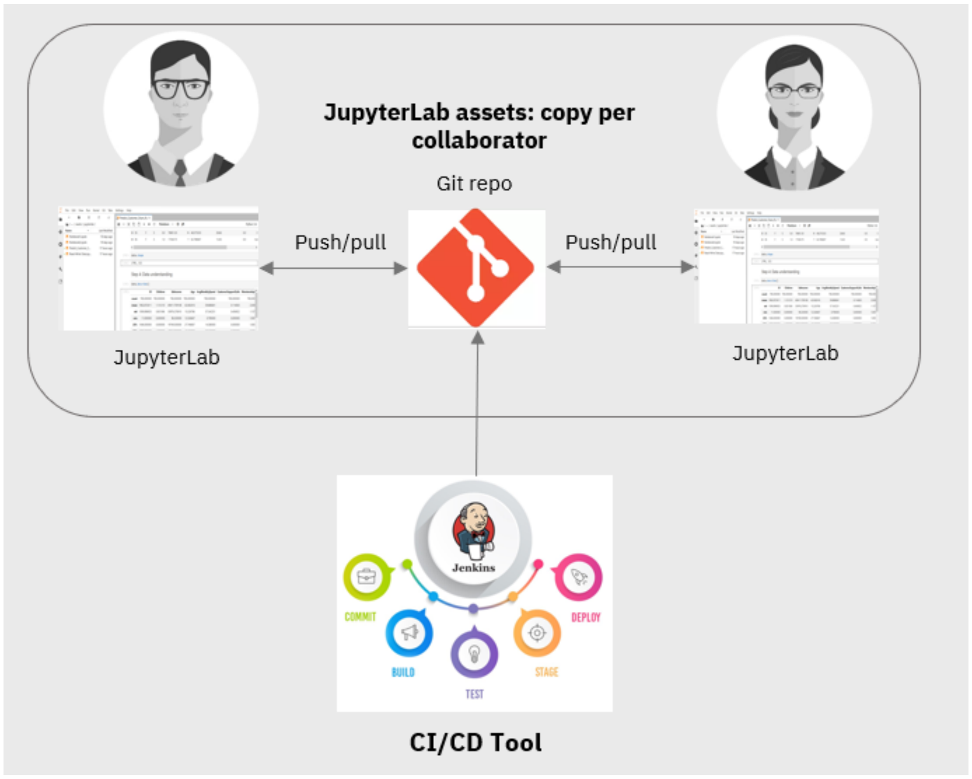
Since there is only one copy of the assets, *Pull and Push* and *Pull* options are not directly applicable to collaboration (because collaborators will see the changes immediately). However, these options are needed for 2 reasons:

- *Push* option will publish assets to a git repo. A git repo can be used by a CI/CD tool for automating deployment.
- *Pull* option is needed for getting assets from JupyterLab into the project (explained in the next option).

The third option is *collaboration with JupyterLab*. JupyterLab is an IDE that's used for editing notebooks, and Git integration is provided by the JupyterLab Git extension.

While conceptually Git integration in JupyterLab is similar to Option 2, the UI and the steps are different.

Figure 3: Option 3 - Collaboration in JupyterLab



Unlike Jupyter Notebook environment, JupyterLab includes a file management component, which means that notebooks and data files can be stored in the JupyterLab IDE. We refer to these files as JupyterLab files. All other files in the project are called "project assets".

Since a project may include not just JupyterLab assets, but also SPSS flows, data assets, RStudio files, and Data Refinery flows, there are two types of Git integration in a project with a JupyterLab – Option 2 for all project assets and Option 3 for JupyterLab files. Please note that only one notebook IDE – either Jupyter Notebook OR JupyterLab can be configured in one project.

While the details of Git integration may seem confusing, if the projects are organized by "asset type", then the user doesn't need to understand the complexity and the different options for integration.

Option 1, local collaboration, is a good option for any company that does not require Git as a version and collaboration management system. High availability is configured by default in a CP4D cluster, and backup can be performed either manually or with scripts.

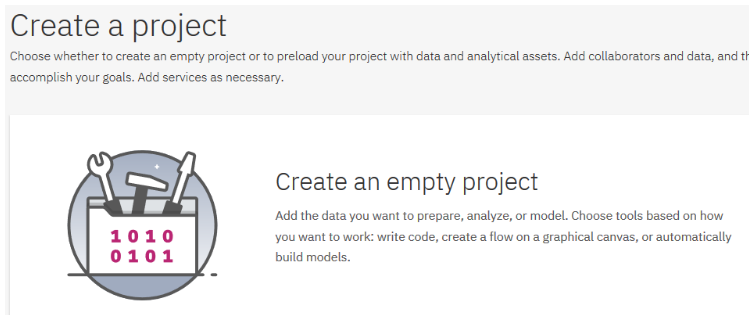
The customer can create a project just with JupyterLab IDE and keep all notebooks and data assets in JupyterLab for a consistent collaboration experience.

Finally, the "mixed asset" projects can be reserved for analytical assets that will eventually be deployed to production. These projects will require more involvement from an admin who can create best practices and a recommended workflow for working with Git.

In this section of the lab we will review the three collaboration options. You will need to work with a colleague to test some aspects of collaboration. If you're running this lab by yourself, create a 2nd userid.

Option 1: Local collaboration

1. Login to CP4D instance and create a new *empty project*.



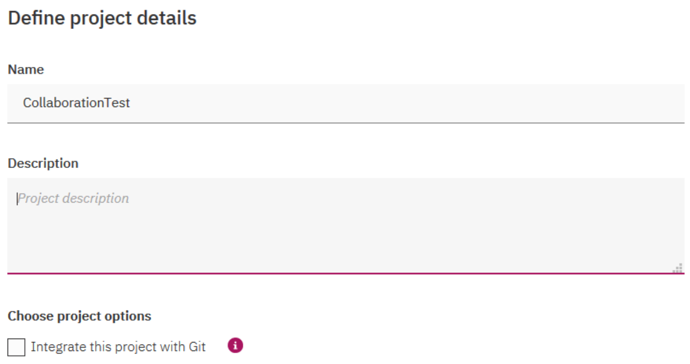
Create a project

Choose whether to create an empty project or to preload your project with data and analytical assets. Add collaborators and data, and th accomplish your goals. Add services as necessary.

Create an empty project

Add the data you want to prepare, analyze, or model. Choose tools based on how you want to work: write code, create a flow on a graphical canvas, or automatically build models.

2. When providing project details, ***do not check*** the *Integrate this project with Git* option.



Define project details

Name

CollaborationTest

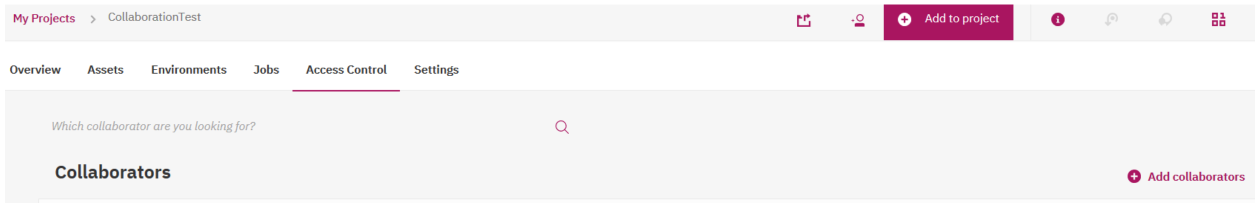
Description

Project description

Choose project options

☐ Integrate this project with Git

3. In the project click on the **Access Control** tab and add a collaborator – one of your colleagues or a 2nd userid. Give the user **Editor** role.



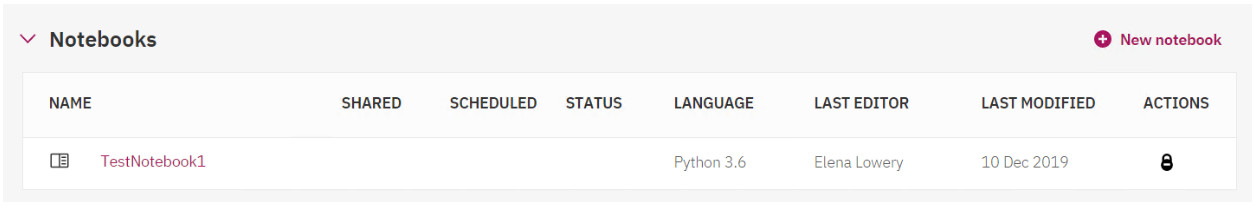
My Projects > CollaborationTest

4. Create a notebook and add simple code, for example a markdown cell or a print statement.

5. Next, we'll test the collaborator experience.

    - If you're working with a colleague, take a look at the project from their browser.
    - If you're using a second userid, log in to CP4D with that userid in a different browser (i.e *Chrome* or *Firefox*), not a different tab in the same browser (browser may refresh and log in as 1 userid).

   The collaborator will see the notebook in locked status. Notice that they can't unlock it. After you unlock the notebook (click on the lock icon and select **Unlock**), ask the collaborator to save changes, close the notebook and unlock it.



NAME	SHARED	SCHEDULED	STATUS	LANGUAGE	LAST EDITOR	LAST MODIFIED	ACTIONS
TestNotebook1				Python 3.6	Elena Lowery	10 Dec 2019	

   Open the notebook – you will see the changes made by the collaborator.

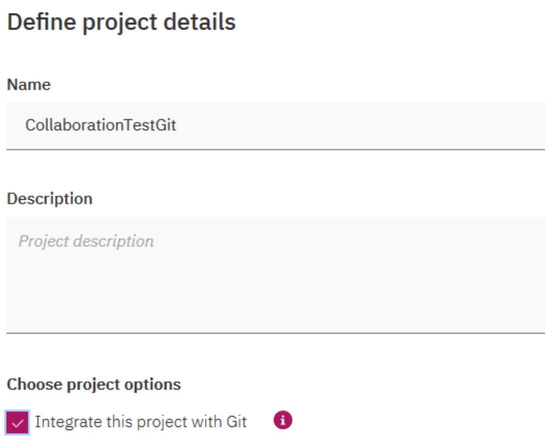
Option 2: Git Collaboration for non-JupyterLab assets

1. Create an empty repo in your Git application. Save the URL and the token.

   *Note: If need instructions for completing these steps, see **Appendix A** and **B.***

2. In CP4D create a new *empty project*.

3. When providing project details, ***check*** the *Integrate this project with Git* option.



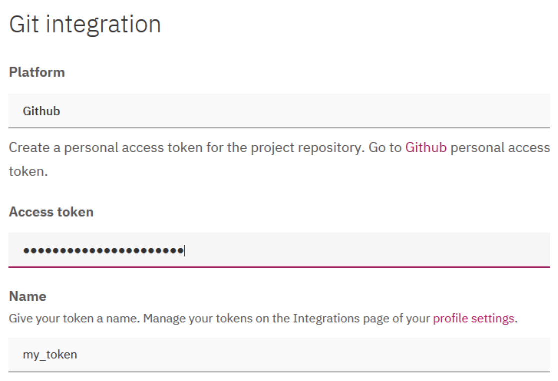
Name: CollaborationTestGit

Description: Project description

Choose project options

[✓] Integrate this project with Git

4. Click **New token**. Paste the token and give it a unique name.


Git integration

Platform

Github

Create a personal access token for the project repository. Go to Github personal access token.

Access token

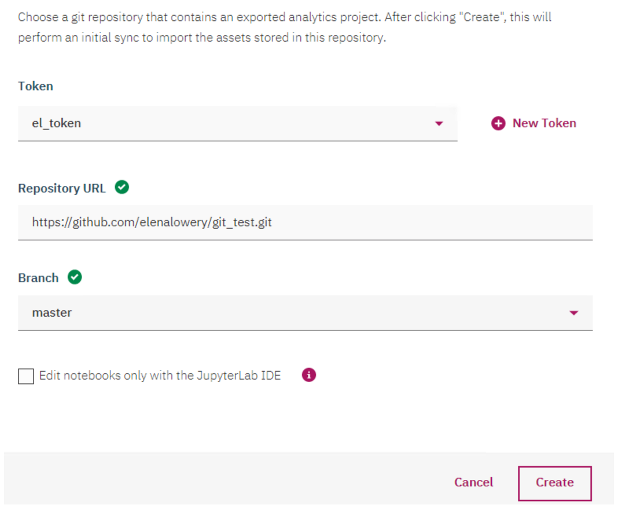
••••••••••••••••••••••

Name

Give your token a name. Manage your tokens on the Integrations page of your profile settings.

my_token

5. Enter the *Repository URL* and select the *master* branch. **Do not check** the *Edit notebooks only with the JupyterLab IDE* checkbox.


Choose a git repository that contains an exported analytics project. After clicking "Create", this will perform an initial sync to import the assets stored in this repository.

Token

el_token

Repository URL

https://github.com/elenalowery/git_test.git

Branch

master

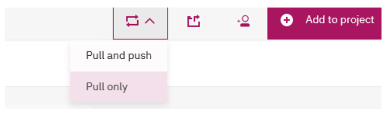
Edit notebooks only with the JupyterLab IDE

6. In the project click on the **Access Control** tab and add a collaborator – one of your colleagues. Give the user **Editor** role.

7. Create 2 notebooks, *Notebook1* and *Notebook2* when logged in as your userid. Add simple code to both notebooks.

8. Ask your colleague to create *Notebook3* or a notebook with a unique name. Add simple code to this notebook.

   - Notice that all assets are visible to collaborators immediately
   - Only the user with *Admin* role on the project can push assets to the git repository (see the Git icon on the menu bar).



Important Note: *While the Pull option is available, new assets will never be pulled from the Git repo for two reasons:*

- *The Git repo should never be modified from outside of Watson Studio. If the Git repo is modified either manually or through another tool, integration from Watson Studio will no longer work. The only exception to this rule is putting files into the JuptyerLab folder, which is explained in the next section.*
- *Changes are immediately available to all collaborators, so the collaborators don't need to Push/Pull to the repo.*

Option 3: Git Collaboration for JupyterLab assets

While JupyterLab is in a project, it has some differences compared to other assets that are a part of a Watson Studio project:

- *JupyterLab* maintains a separate file structure for each user. In other words, users **are not** working on one copy of a notebook when working in JupyterLab, which is the case for all other assets in Watson Studio.

- If you want to commit notebooks to Git, you need to place notebooks into a pre-created directory in *JupyterLab* (*project_git_repo/<repo_name>/assets/jupyter_lab*)

- Notebooks that are saved outside of that directory can't be pushed to Git.

- Data assets can also be uploaded directly into Git. Uploading data assets into Git is not required (project data assets can still be used). If you want to commit data assets from JupyterLab to Git, you need to place them into a a pre-created directory in *JupyterLab* (*project_data_assets/data_asset*).

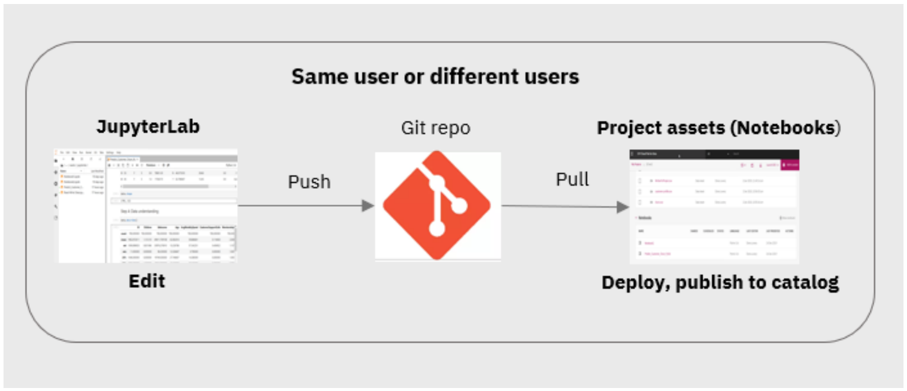
There are two types of Git integration in a project that has JupyterLab:

1. *JupyterLab* integration with Git via the JupyterLab Git extension
2. Project-level integration with Git.

The integration in *JupyterLab* is done with the Git extension (open source extension to JupyterLab). This integration is needed for collaboration. Collaborators won't see notebooks unless they go through the Commit/Pull steps.

Project-level integration with Git is still needed for performing notebook-related tasks, such as scheduling a notebook run or publishing a notebook as a URL. In order to perform these tasks, the notebooks must become a "project assets" – they should be pushed to the Git repo from *JupyterLab* Git extension, and pulled from the same repo on the project level.

Figure 4: Using Git to get assets from JupyterLab to Project.


Same user or different users
JupyterLab
Git repo
Project assets (Notebooks)
Push
Pull
Edit
Deploy, publish to catalog

We will review these steps in the lab.

1. Create an empty repo in your Git application

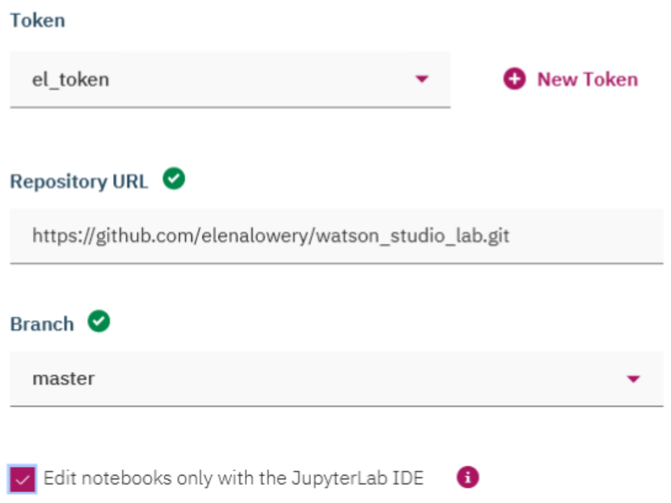
    - Note the repo URL
    - You will be able to reuse the token that you created in the previous section.

2. In CP4D create a new *empty project*.

3. When providing project details, ***check*** the *Integrate this project with Git* option.

4. Provide Git repo information.

5. Enter the *Repository URL* and select the *master* branch. **Check** the *Edit notebooks only with the JupyterLab IDE* checkbox.



Token: el_token

New Token

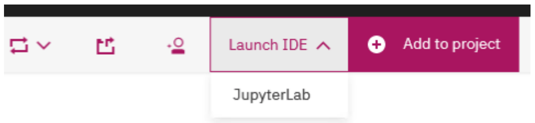
Repository URL: https://github.com/elenalowery/watson_studio_lab.git

Branch: master

[✓] Edit notebooks only with the JupyterLab IDE

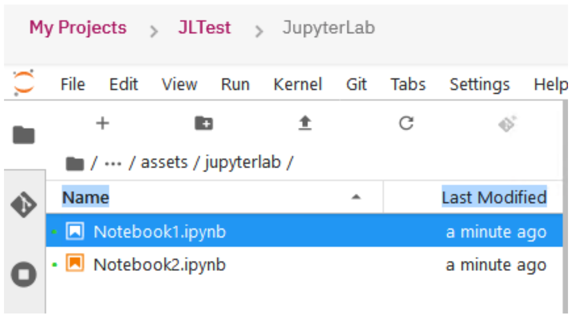
6. In the project click on the **Access Control** tab and add a collaborator – one of your colleagues. Give the user **Editor** role.

7. Notice that we now have an option to launch *JupyterLab* IDE. Launch the IDE.

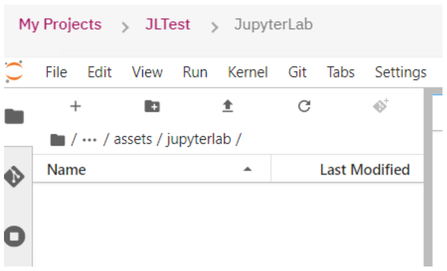


8. In *JupyterLab*, navigate to the *project_git_repo/<your_repo>/assets/jupyterlab* directory and create 2 notebooks.

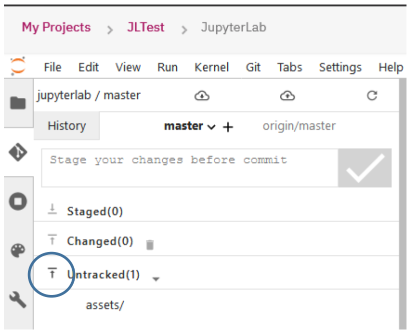
   Important note: *If you create notebooks in any other folder in JupyterLab, you will not be able to push them to the git repo.*


My Projects > JLTest > JupyterLab
File Edit View Run Kernel Git Tabs Settings Help
/ ··· / assets / jupyterlab /
Name
Last Modified
Notebook1.ipynb a minute ago
Notebook2.ipynb a minute ago

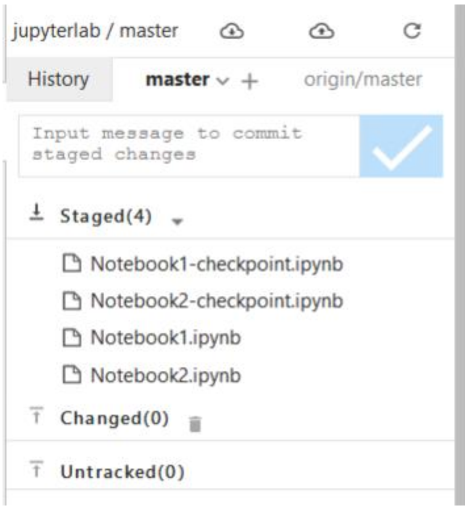
9. Ask your colleague to open the same project and launch *JupyterLab*. While they will see the git repo, they will not see any notebooks because you haven't committed them yet.


My Projects > JLTest > JupyterLab
File Edit View Run Kernel Git Tabs Settings
/ ··· / assets / jupyterlab /
Name
Last Modified

10. In your project, click on the *Git* icon. Then click on the arrow next to *Untracked*.


My Projects > JLTest > JupyterLab
File Edit View Run Kernel Git Tabs Settings Help
jupyterlab / master
History
master
origin/master
Stage your changes before commit
Staged(0)
Changed(0)
Untracked(1)
assets/

11. All assets are moved to the *Staged* section. You can remove the assets that you don't want committed from *Staged* section by selecting them and then clicking the down arrow.


jupyterlab / master

History | **master** | origin/master

Input message to commit staged changes

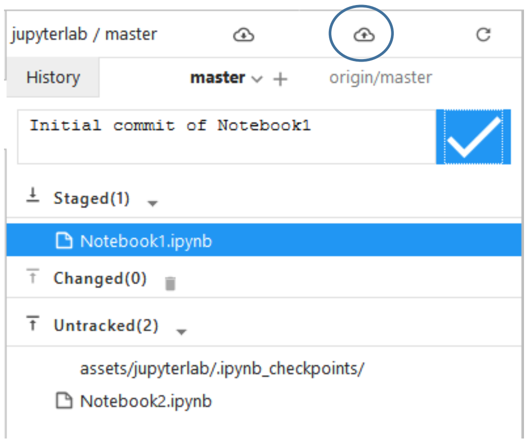
Staged(4)
- Notebook1-checkpoint.ipynb
- Notebook2-checkpoint.ipynb
- Notebook1.ipynb
- Notebook2.ipynb

Changed(0)

Untracked(0)

Let's commit only *Notebook1*.

- Remove all checkpoints and *Notebook2*
- Provide a commit message and click the *blue checkbox*
- Click the *push* icon (cloud with an up arrow).


jupyterlab / master

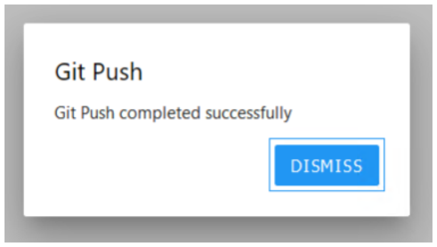
History | **master** | origin/master

Initial commit of Notebook1

Staged(1)
- Notebook1.ipynb

Changed(0)

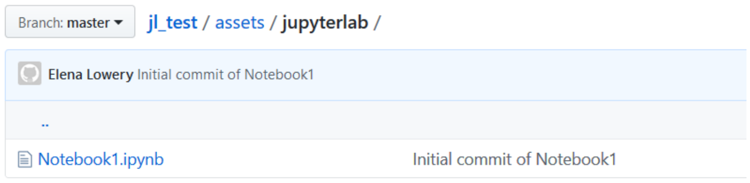
Untracked(2)
- assets/jupyterlab/.ipynb_checkpoints/
- Notebook2.ipynb


Git Push

Git Push completed successfully

DISMISS

12. Check your git repo (in *Github*). You should now see the notebook.


Branch: master ▾ jl_test / assets / jupyterlab /

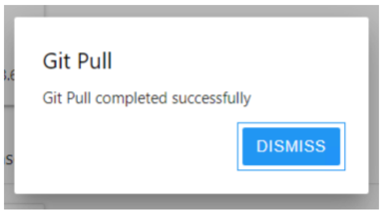
Elena Lowery Initial commit of Notebook1

..

Notebook1.ipynb	Initial commit of Notebook1

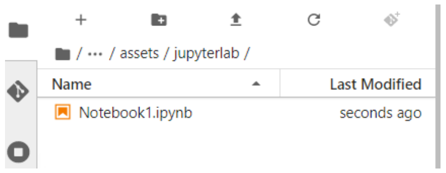
13. Ask your colleague to pull the changes into their project

- Open the Git extension in the JupyterLab environment
- Click the **Pull** icon (cloud with down arrow).


Git Pull

Git Pull completed successfully

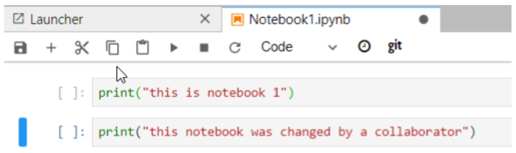
When they switch to the **File** view, they will see the notebook.


/ ··· / assets / jupyterlab /

Name	Last Modified
Notebook1.ipynb	seconds ago

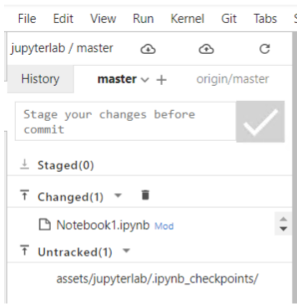
Ask your colleague to change the notebook.

In our example, we added a 2nd cell with a *print* statement.



```
[ ]: print("this is notebook 1")
[ ]: print("this notebook was changed by a collaborator")
```

Ask your colleague to click on the Git icon. JupyterLab already determined that the notebook has changed.



Stage your changes before commit

Staged(0)

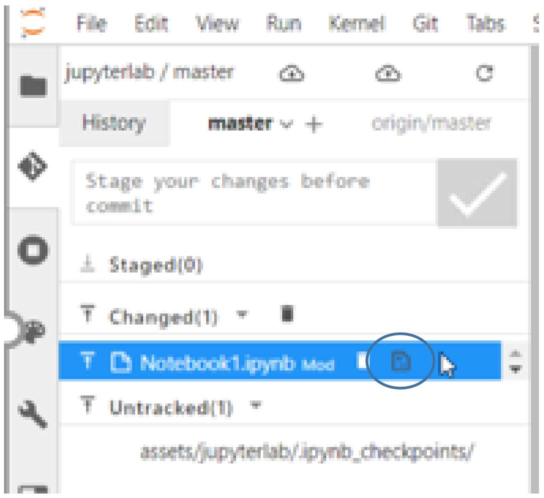
Changed(1)

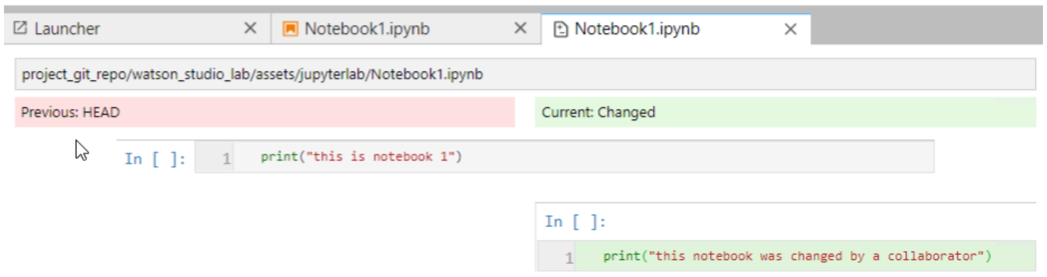
Notebook1.ipynb Mod

Untracked(1)

assets/jupyterlab/.ipynb_checkpoints/

The Git extension provides access to another extension, *nbdime*, that highlights differences between the versions of the notebook. To display this extension, highlight the notebook and click on the icon that appears when you hover over the highlighted notebook.



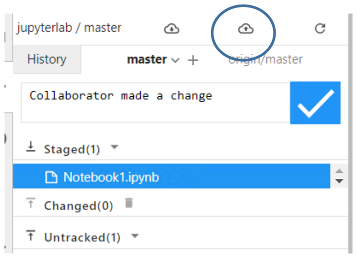

Launcher
Notebook1.ipynb
Notebook1.ipynb
project_git_repo/watson_studio_lab/assets/jupyterlab/Notebook1.ipynb
Previous: HEAD
Current: Changed

```
In [ ]:  print("this is notebook 1")
```

```
In [ ]:
print("this notebook was changed by a collaborator")
```

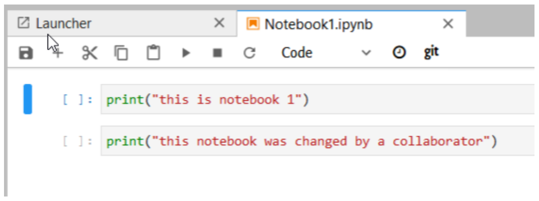
Ask your colleague to commit the change. They will need to follow the same process as we did in Steps 10 and 11:

- Remove all checkpoints and *Notebook2*
- Provide a commit message and click the *blue checkbox*
- Click the *push* icon (cloud with an up arrow).



14. In your *JupyterLab* environment open Git extension and pull from Git. You should now see the changes that were made by your colleague.

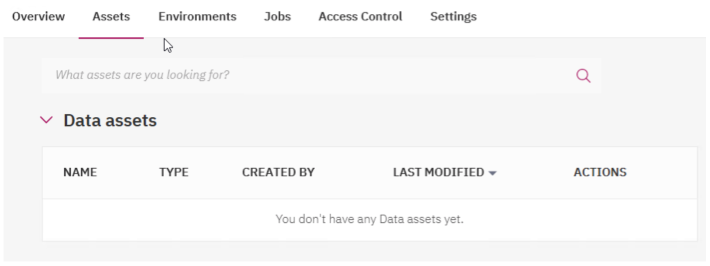
Note: If the notebook was open in your JupyterLab environment, you will have to close it and open it again.



```
[ ]: print("this is notebook 1")
[ ]: print("this notebook was changed by a collaborator")
```

Next, we will pull the notebook into the project so that we can perform notebook functions such as publishing a notebook as a URL or configuring it to run as a batch job.

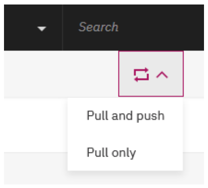
15. Navigate up to the **Project** view. Notice that there are no assets in the project (because notebooks are stored in *JupyterLab* IDE).



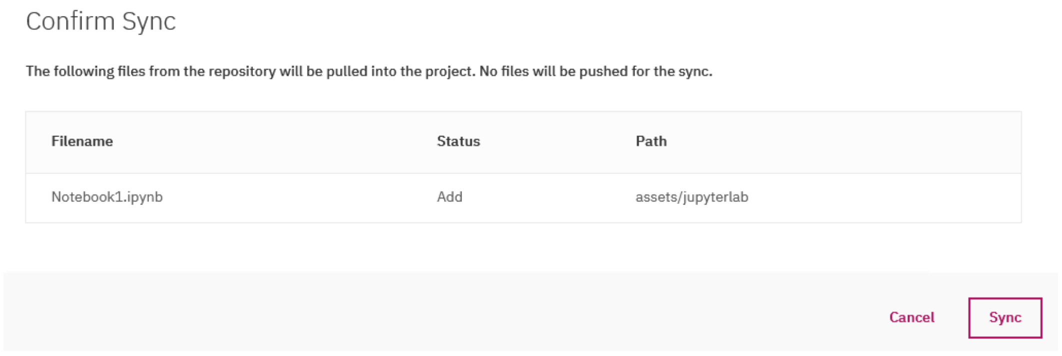
NAME	TYPE	CREATED BY	LAST MODIFIED	ACTIONS

You don't have any Data assets yet.

16. Click on the **Git** icon, then select **Pull only**.



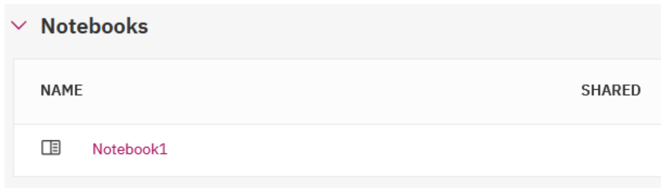
17. Click **Sync**, then **View Project Assets**.



The following files from the repository will be pulled into the project. No files will be pushed for the sync.

Filename	Status	Path
Notebook1.ipynb	Add	assets/jupyterlab

The notebook is now displayed under project assets.


Notebooks

NAME	SHARED
Notebook1	

Notice that you can't edit this notebook. This is by design – a project is always configured with a *JupyterLab IDE* OR *Jupyter Notebook IDE* (not both). If you want to make changes to this notebook, you will need to do it in *JupyterLab*, commit changes to the Git repo, and pull changes into the project.

Click on the notebook and notice that several action items are now available. We tested these options in the **Deployment** section of the lab.

Part 3: Package and Environment Management

In this section you will learn how to load packages and configure environments in Watson Studio.

Package management

While it's possible to load packages using *!conda* and *!pip* commands in notebooks, installation of packages with these commands is not persisted. The packages will not be available to collaborators and even to the same user after environment restart.

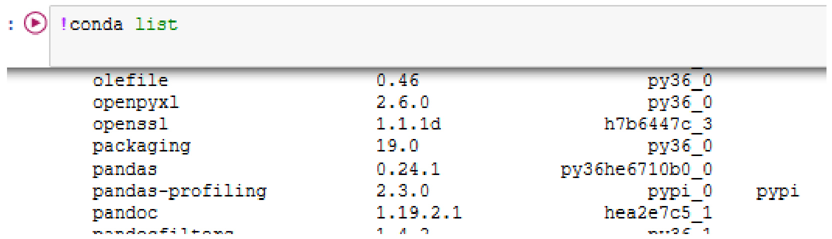
The recommended approach is to specify which packages should be loaded in the *Environment Definition* view. The packages will be installed each time the environment starts, which will provide a consistent experience for each user.

First, let's test what happens if we use *!conda* install in a notebook. You will need to work with a colleague to complete this part of the lab.

1. Open the *Predict_Customer_Churn_Py36* notebook that you used in the previous section of the lab.

2. Add a cell in the beginning of the notebook and run the *!conda list* command.

   Notice that the *pandas-profiling* library is installed. It was installed because you ran the notebook earlier, and the notebook has the code to install the pandas-profiling package.



```
!conda list
```

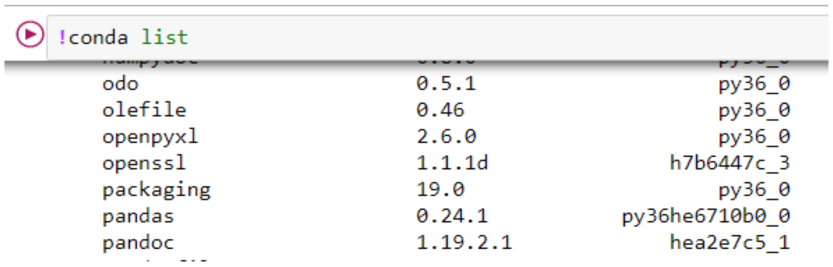
```
olefile                   0.46                     py36_0
openpyxl                  2.6.0                    py36_0
openssl                   1.1.1d               h7b6447c_3
packaging                 19.0                     py36_0
pandas                    0.24.1           py36he6710b0_0
pandas-profiling          2.3.0                    pypi_0    pypi
pandoc                    1.19.2.1             hea2e7c5_1
```

3. Stop the notebook and unlock it.

4. Add a collaborator.

5. Ask the collaborator to open the notebook in *Edit* mode and run the *!conda* list command.

   The same package is not loaded for the collaborator.



```
!conda list
     odo                       0.5.1                    py36_0
     olefile                   0.46                     py36_0
     openpyxl                  2.6.0                    py36_0
     openssl                   1.1.1d               h7b6447c_3
     packaging                 19.0                     py36_0
     pandas                    0.24.1           py36he6710b0_0
     pandoc                    1.19.2.1             hea2e7c5_1
```

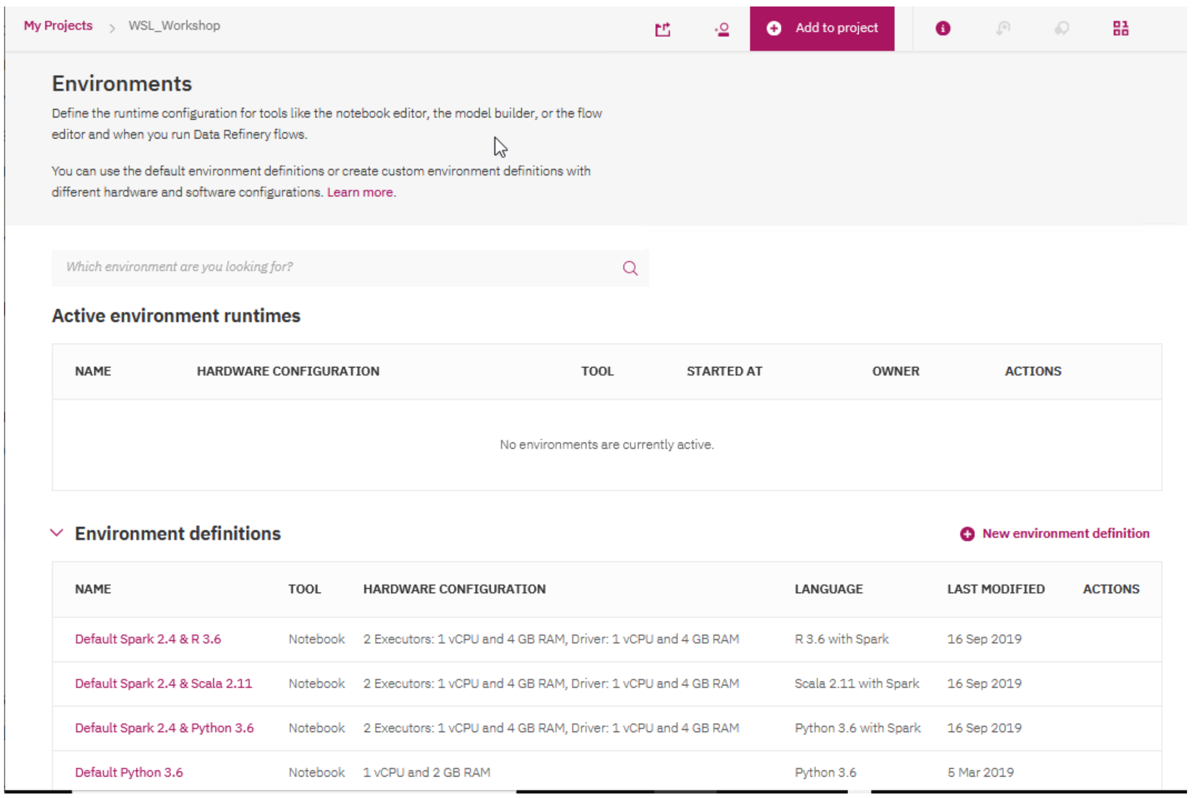
6. Ask the collaborator to close the notebook, stop the environment, and unlock the notebook.

   Note: if the collaborator is not an Admin of your project, then the environment is stopped from the Environments tab, not from the notebook menu.

   Next, we'll use the recommended "*declarative*" approach for package management.

7. In your project click on the **Environments** tab.

   - Make sure the notebook environment is not running.
   - Click on Python 3.6 environment



My Projects > WSL_Workshop
Add to project

Environments

Define the runtime configuration for tools like the notebook editor, the model builder, or the flow editor and when you run Data Refinery flows.

You can use the default environment definitions or create custom environment definitions with different hardware and software configurations. Learn more.

Which environment are you looking for?

Active environment runtimes

NAME	HARDWARE CONFIGURATION	TOOL	STARTED AT	OWNER	ACTIONS
No environments are currently active.					

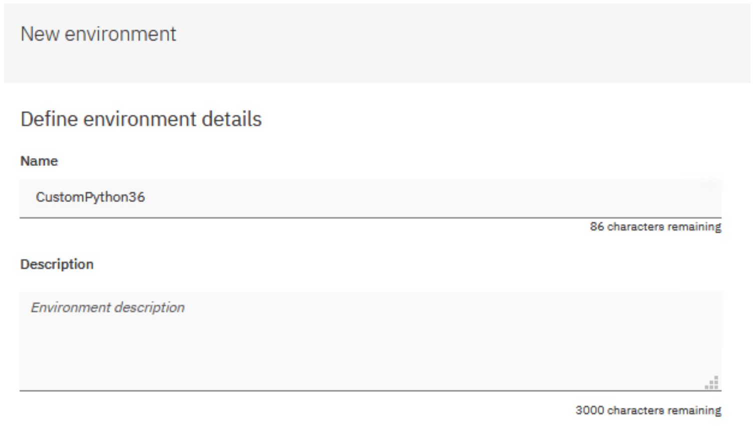
Environment definitions

NAME	TOOL	HARDWARE CONFIGURATION	LANGUAGE	LAST MODIFIED	ACTIONS
Default Spark 2.4 & R 3.6	Notebook	2 Executors: 1 vCPU and 4 GB RAM, Driver: 1 vCPU and 4 GB RAM	R 3.6 with Spark	16 Sep 2019	
Default Spark 2.4 & Scala 2.11	Notebook	2 Executors: 1 vCPU and 4 GB RAM, Driver: 1 vCPU and 4 GB RAM	Scala 2.11 with Spark	16 Sep 2019	
Default Spark 2.4 & Python 3.6	Notebook	2 Executors: 1 vCPU and 4 GB RAM, Driver: 1 vCPU and 4 GB RAM	Python 3.6 with Spark	16 Sep 2019	
Default Python 3.6	Notebook	1 vCPU and 2 GB RAM	Python 3.6	5 Mar 2019	

8. Click on **Default Python 3.6**. On this screen you can review the packages that are loaded by default. Notice that *pandas-profiling* is not one of the packages in the list.

9. Return to the **Environments** tab and click **New environment definition**.

10. Provide the environment name and click **Create**.



New environment

Define environment details

Name

CustomPython36

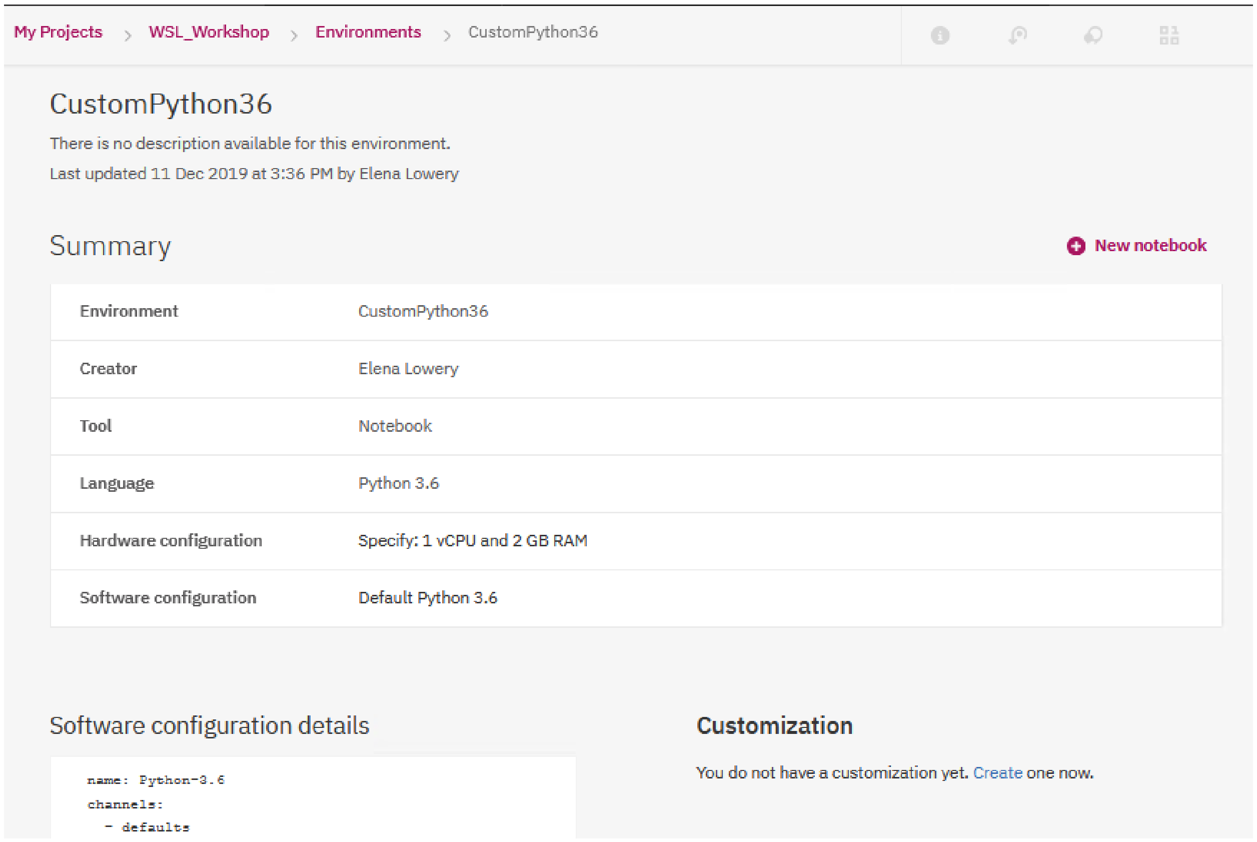
86 characters remaining

Description

Environment description

3000 characters remaining

11. Click **Create** under **Customization**.



CustomPython36

There is no description available for this environment.

Last updated 11 Dec 2019 at 3:36 PM by Elena Lowery

Summary

Environment	CustomPython36
Creator	Elena Lowery
Tool	Notebook
Language	Python 3.6
Hardware configuration	Specify: 1 vCPU and 2 GB RAM
Software configuration	Default Python 3.6

Software configuration details

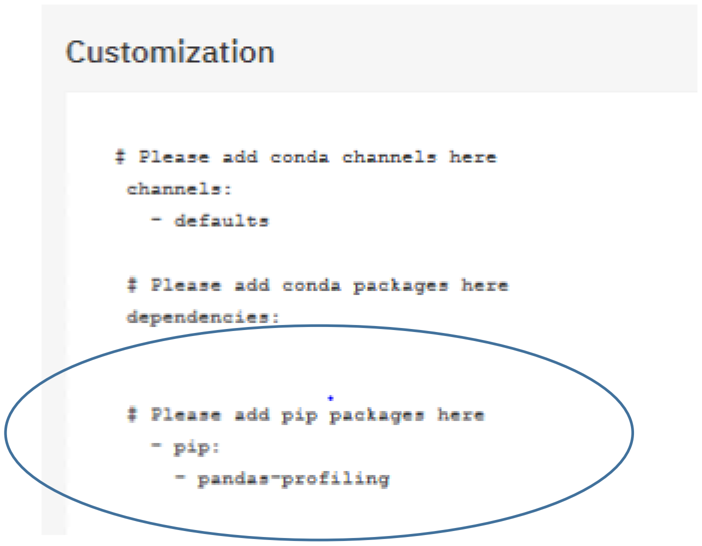
```
name: Python-3.6
channels:
  - defaults
```

Customization

You do not have a customization yet. Create one now.

12. Make the following changes to the yml file:

- Add – *pandas-profiling* under pip
- Make sure the indentation is exactly as shown in the screenshot. If indentation is not correct, you will get an error when the environment starts.



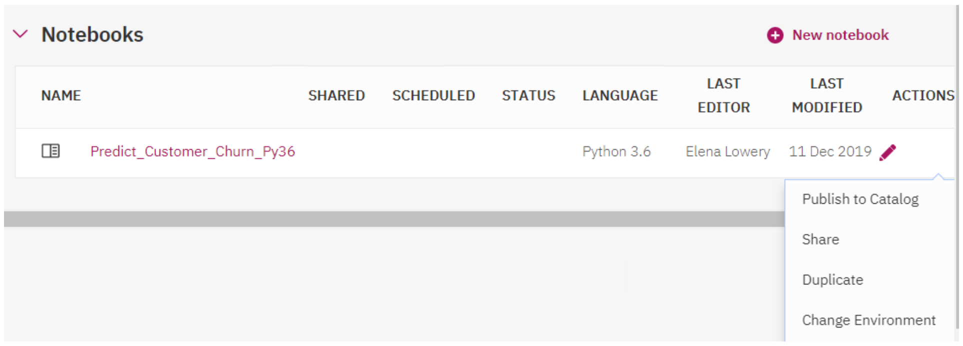
Customization

```
# Please add conda channels here
channels:
  - defaults

# Please add conda packages here
dependencies:


# Please add pip packages here
  - pip:
    - pandas-profiling
```

The customization screen is in the *.yml* format (a format that's a standard for Conda environments). You can find more details about customization details here: https://www.ibm.com/support/knowledgecenter/en/SSQNUZ_2.5.0/wsj/analyze-data/customize-envs.html

To test loading of this package you'll need to work with a colleague. Alternatively, you can uninstall the package from your environment (*!conda remove*), and then follow the same steps.

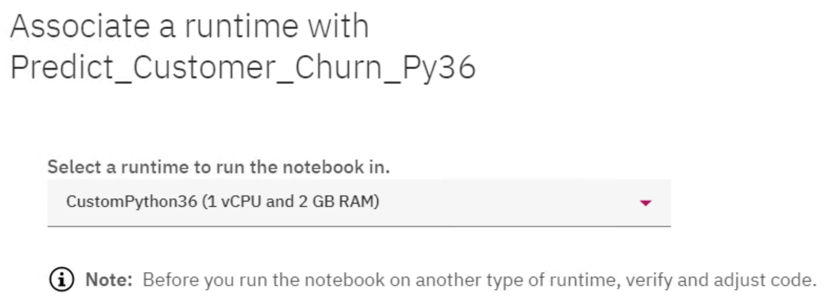
13. Ask your colleague to navigate to the project and select **Change Environment** from the notebook **Actions** menu.



Notebooks

NAME	SHARED	SCHEDULED	STATUS	LANGUAGE	LAST EDITOR	LAST MODIFIED	ACTIONS
Predict_Customer_Churn_Py36				Python 3.6	Elena Lowery	11 Dec 2019	

14. Select the *Custom Python* environment and click **Associate**.


Associate a runtime with Predict_Customer_Churn_Py36

Select a runtime to run the notebook in.

CustomPython36 (1 vCPU and 2 GB RAM)

Note: Before you run the notebook on another type of runtime, verify and adjust code.

15. Open the notebook in **Edit** mode and run *!conda list* command.

    Pandas-profiling package is now installed because it was loaded during environment startup.

Part 4: Importing Notebooks into JupyterLab

In this section you will learn how to import existing notebooks into JupyterLab.

Importing notebooks

You may have noticed that JupyterLab does not provide *Create from existing* option for notebooks. We can get existing notebooks into JupyterLab by importing them into the git repo.

Notebooks and data files have to be placed in specific folders in the git repo. The easiest way to make sure that the directory structure is correct is to do a *push* from JupyterLab.

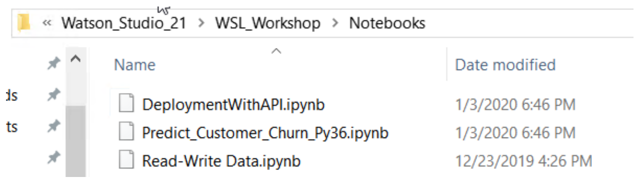
We already completed some of these steps in **Part 2**, let's review them:

1. Create an empty git repo.
2. Associate a git repo with a Watson Studio project.
3. Create a simple notebook.
4. Commit changes to the repo - this will create the folder structure.

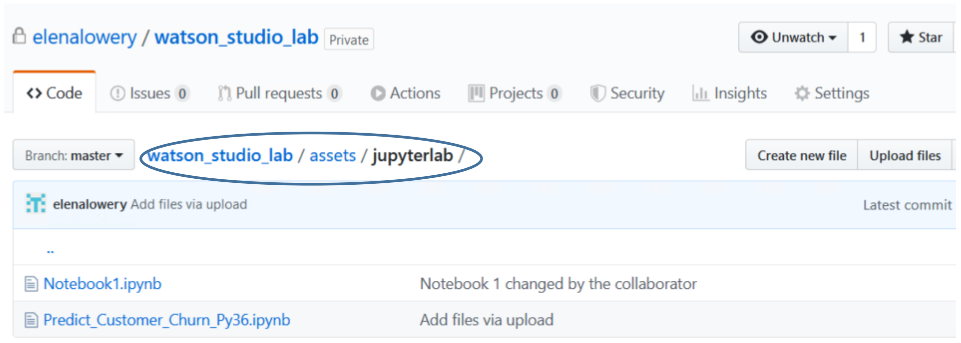
Next, we'll complete the steps to import a notebook into JupyterLab.

1. Open the git repo which you connected to your JupyterLab project.

2. Import the notebook that we have used for model deployment, *Predict_Customer_Churn_Py36,* into the *<your_git_repo>/assets/jupyterlab* folder*.* This notebook can be found in the *git repo\Notebooks* folder*.*



Name	Date modified
DeploymentWithAPI.ipynb	1/3/2020 6:46 PM
Predict_Customer_Churn_Py36.ipynb	1/3/2020 6:46 PM
Read-Write Data.ipynb	12/23/2019 4:26 PM

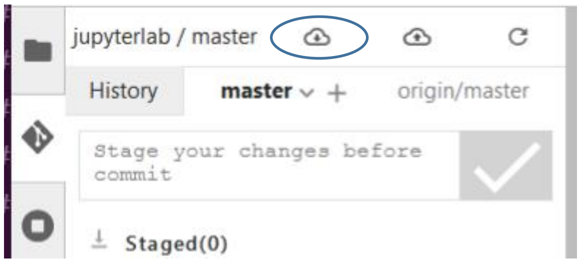


Notebook1.ipynb	Notebook 1 changed by the collaborator
Predict_Customer_Churn_Py36.ipynb	Add files via upload

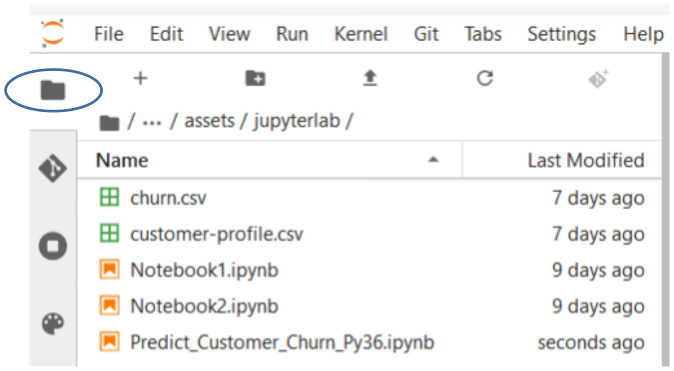
Important note*: notebooks must be imported into this folder. If they are imported into a different folder, they will not be available in JupyterLab. Importing assets into other folders will also break git integration on the project level (as explained in Part 2).*

3. In Watson Studio open your project, then open JupyterLab.

4. Open the git extension and click on the **Pull** icon.



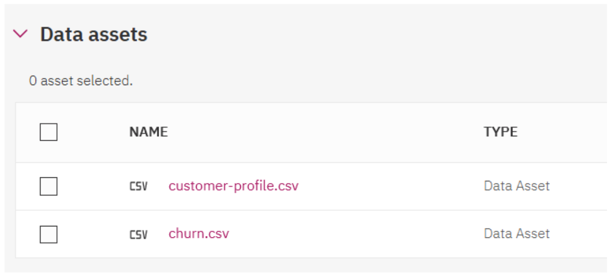
5. After synchronization is done, click on the **File** icon. The notebook is now available in JupyterLab.



Name	Last Modified
churn.csv	7 days ago
customer-profile.csv	7 days ago
Notebook1.ipynb	9 days ago
Notebook2.ipynb	9 days ago
Predict_Customer_Churn_Py36.ipynb	seconds ago

6. If you wish, you can test model deployment from JuptyerLab. You will need to configure a deployment space similar to the way you'd done it in **Part 1** of the lab (see page 6).

   You will also need to import *churn.csv* and *customer-profile.csv* files into the project (not JupyterLab) data assets. These files can be found in *git repo\Data* folder.



NAME	TYPE
customer-profile.csv	Data Asset
churn.csv	Data Asset

7. Repeat steps 1-4 to import another notebook, *Read-Write Data* from the *git repo\Notebooks* folder.

   This notebook is an example of how to read/write data from file system in Watson Studio. It shows the differences between using pandas API and *Project API* (provided by Watson Studio).

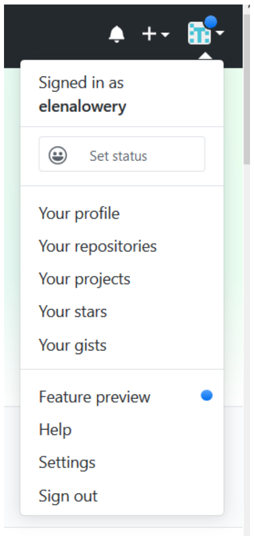
8. Review the code in the notebook and run it.

You have completed the JupyterLab section of the lab.

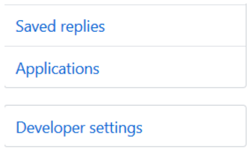
Appendix A: Getting a Git Repo Token

Note: The following instructions are specific to getting a token from a github.com account.

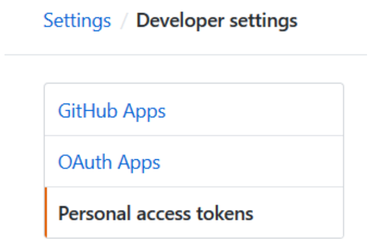
1. If you don't already have a git account, set it up on *github.com* or another git site.
2. Create a git repo. If you need instructions – see **Appendix B.**
3. Click on the **Profile** icon (top right corner), then click on **Settings**.



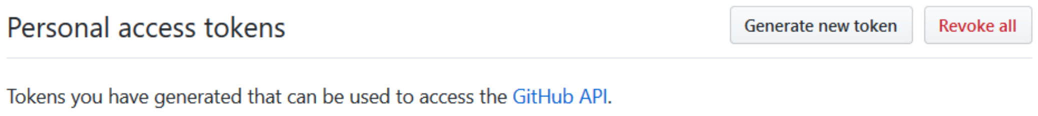
4. Click on **Developer Settings** (in the bottom of the left-side menu).



5. Click on **Personal access tokens**.



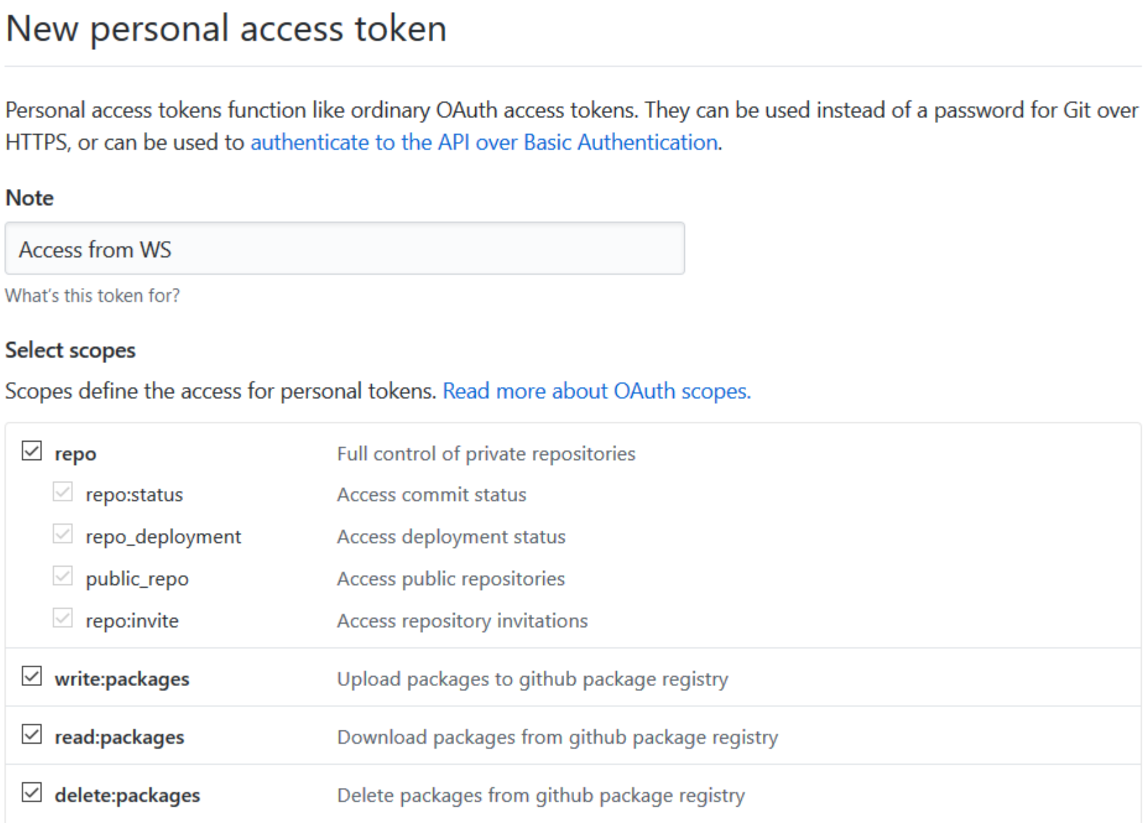
6. Click **Generate new token.**



Tokens you have generated that can be used to access the GitHub API.

7. Provide a token name, for example, *Access from WS*.

   Check the *scopes* checkboxes – as a minimum, select the scopes listed in the screenshot.



New personal access token

Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to authenticate to the API over Basic Authentication.

Note

Access from WS

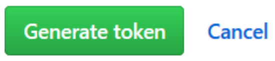
What's this token for?

Select scopes

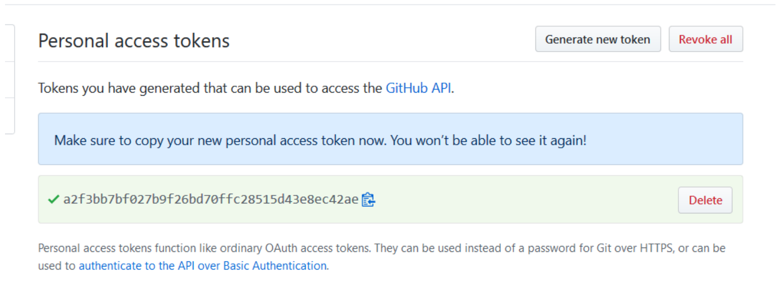
Scopes define the access for personal tokens. Read more about OAuth scopes.

☑ repo	Full control of private repositories
☑ repo:status	Access commit status
☑ repo_deployment	Access deployment status
☑ public_repo	Access public repositories
☑ repo:invite	Access repository invitations
☑ write:packages	Upload packages to github package registry
☑ read:packages	Download packages from github package registry
☑ delete:packages	Delete packages from github package registry

8. Click Generate token.



9. Save the token in a notepad. You will use this value in Watson Studio.



Personal access tokens

Tokens you have generated that can be used to access the GitHub API.

Make sure to copy your new personal access token now. You won't be able to see it again!

✓ a2f3bb7bf027b9f26bd70ffc28515d43e8ec42ae

Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to authenticate to the API over Basic Authentication.

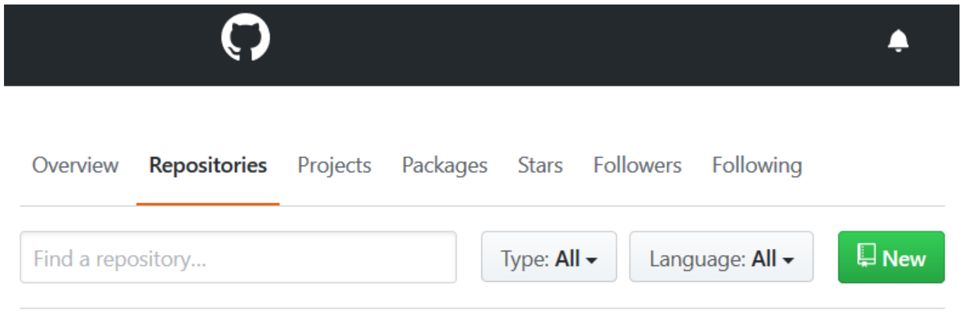
Appendix B: Setting up a Git repo

1. If you don't already have a git account, set it up on *github.com* or another git site.

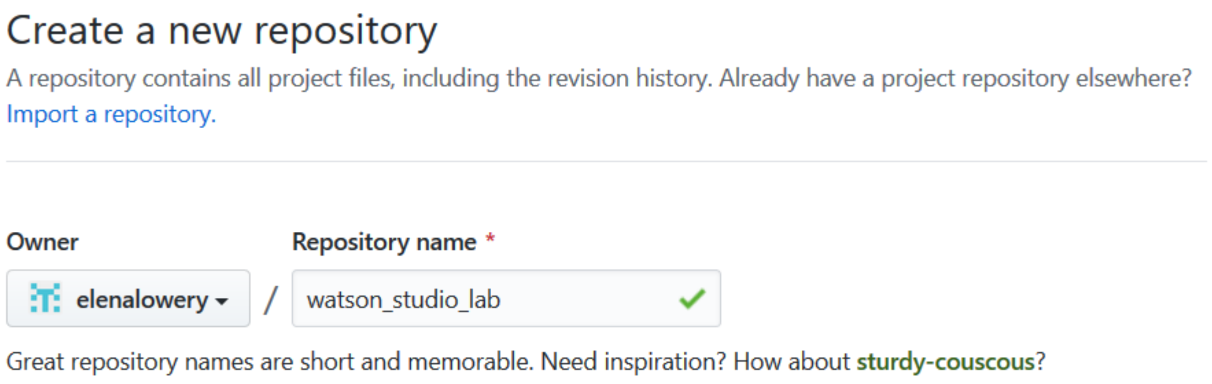
   Note: The following instructions are specific to getting a token from a github.com account.

2. Log in to your git account.

3. Click on the **Repositories** tab (top right corner), then click **New**.



4. Provide *Repository name*. You can keep the default settings or you can change the security setting to *Private*.



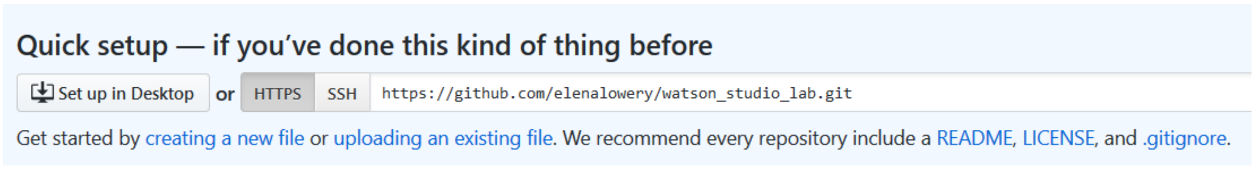
Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Owner: elenalowery / Repository name *: watson_studio_lab

Great repository names are short and memorable. Need inspiration? How about sturdy-couscous?

5. Note the *https URL* – you will use it in Watson Studio.



Quick setup — if you've done this kind of thing before

Set up in Desktop or HTTPS SSH `https://github.com/elenalowery/watson_studio_lab.git`

Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.