

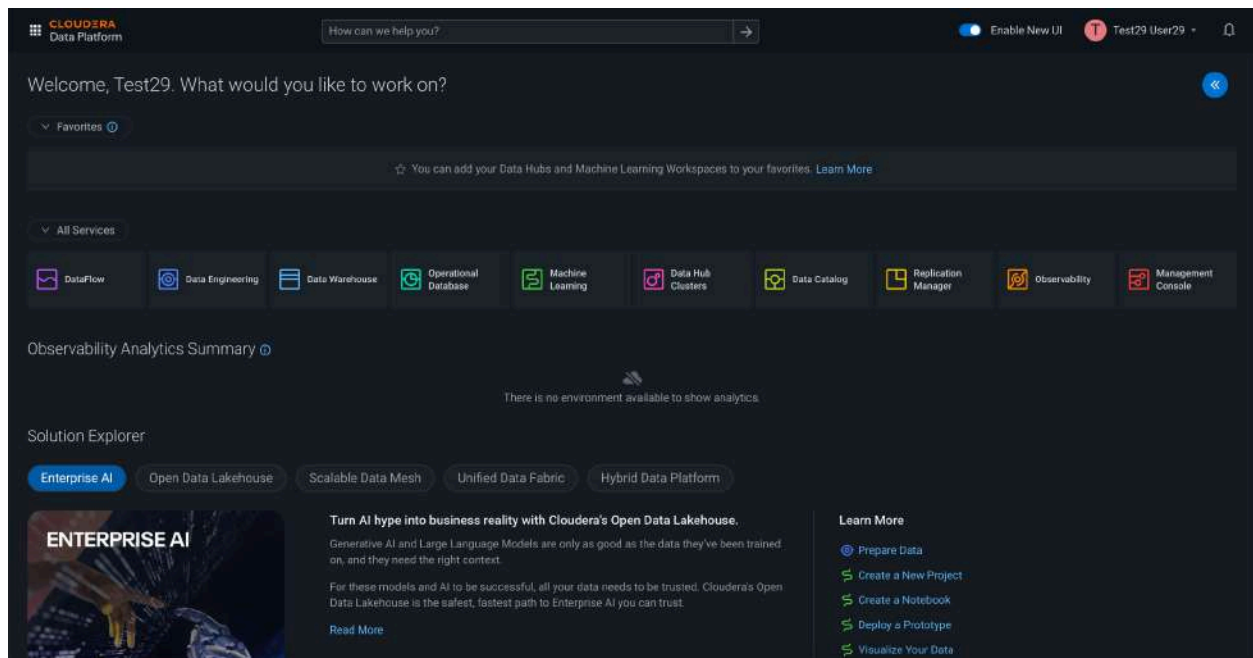
# Data Lifecycle on CDP Public Cloud

## Lab 004: Machine Learning Lab

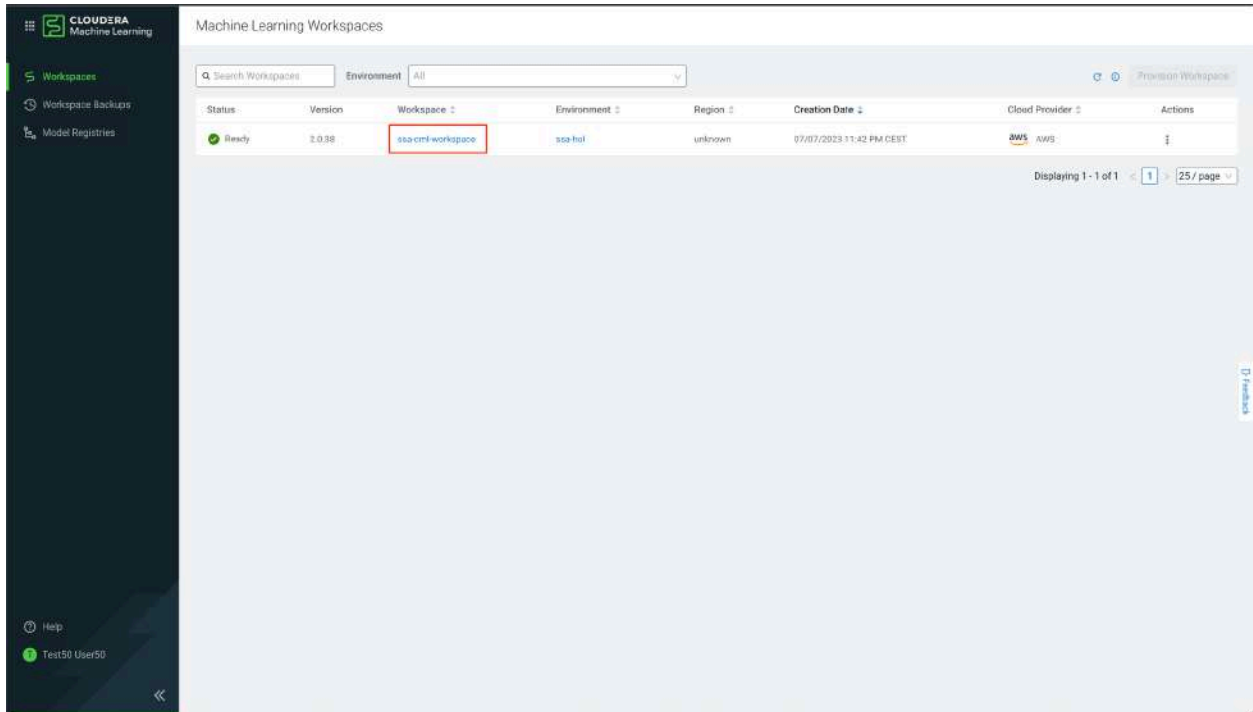
Goals:

- Train a model to predict if a customer will churn
- Deploy/expose model as REST API

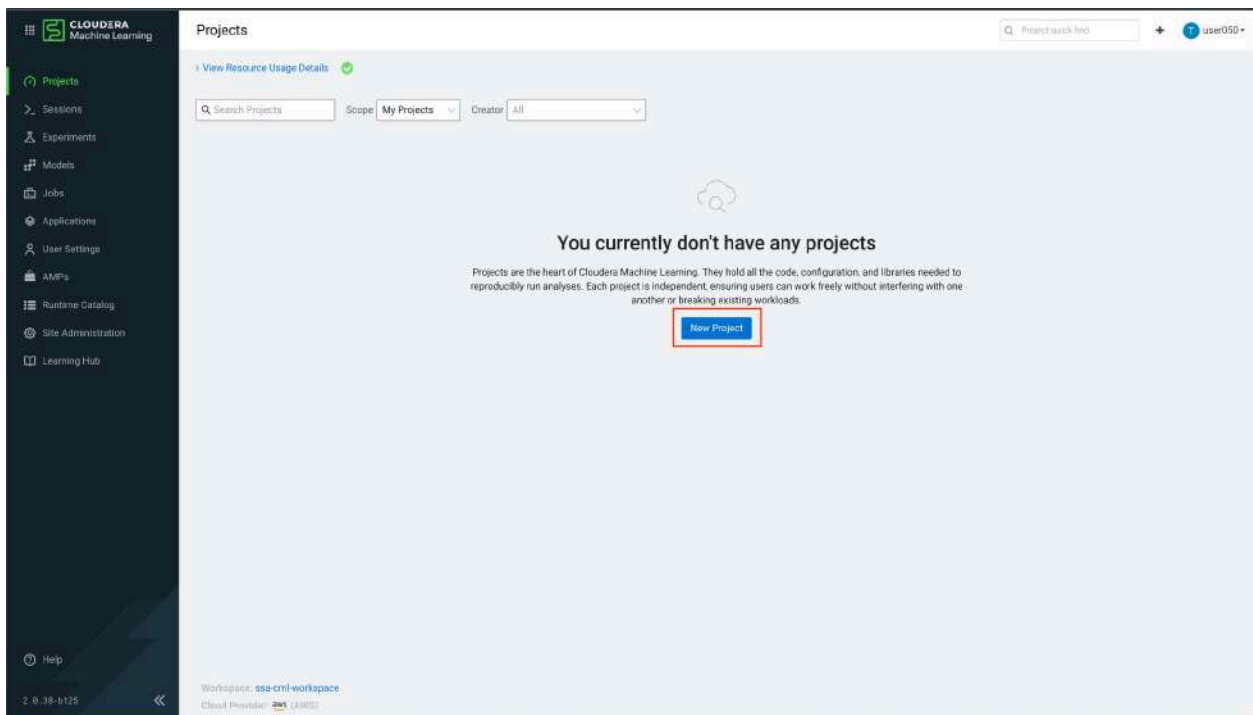
1. Click on Data Warehouse from CDP PC Home:



2. This is a screen to select a Workspace, which is compute resource allocation for Data Science related jobs. Click on the only Workspace that appears.



3. Once in the Workspace, you should see the following interface. Here are the projects you have created. It is time to create a new project. Click on the blue button **New Project**.



4. Enter the following information to create a new project:

**Project Name:** Telco Churn

**Project Visibility:** Private

**Initial Setup,** select Git

In the text field below HTTPS, enter the url of the git repo:

<https://github.com/camposalex/TelcoChurn>

New Project

Project quick find

user050

\* Project Name

Telco Churn

Project Description

Project Visibility

☒ Private - Only added collaborators can view the project

☐ Public - All authenticated users can view this project.

Initial Setup

Blank Template AMPs Local Files Git

Provide the Git URL of the project to clone. Select the option that applies to your URL access.

☒ HTTPS ☐ SSH

https://github.com/camposalex/TelcoChurn

You are able to provide username/password.  
e.g. https://username:password@mygithub.com/my/repository

Make sure to select **Python 3.7** in the Kernel selector. Click the button **Create Project**

Runtime setup

Basic Advanced

Basic configuration adds the most commonly used Editors for the Kernel of your choice. To fine-tune the Editors available in the project, choose the Advanced tab.

Kernel

Python 3.7

☐ Add GPU enabled Runtime variant

These runtimes will be added to the project:

JupyterLab - Python 3.7 - Standard - 2023.05

PBJ Workbench - Python 3.7 - Standard - 2023.05

Workbench - Python 3.7 - Standard - 2023.05

Cancel Create Project

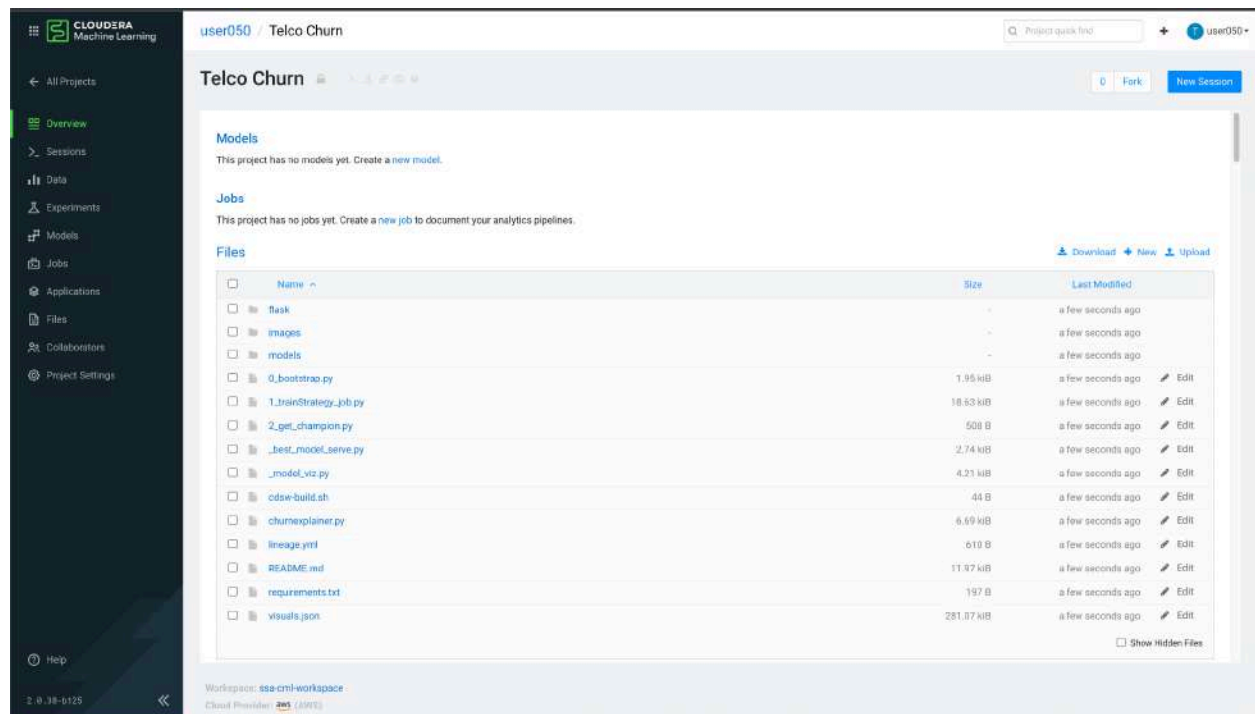
5. Once the project is created, you should see the following screen:

**Models**, deploy and manage models as REST APIs to serve predictions.

**Jobs**, automate and orchestrate the execution of batch analytics workloads

**Files**, assets that are part of the project, such as files, scripts and code

This Telco Churn project consists of running three scripts. The way of execution is through a session, which is the allocation of isolated compute resources for each user. For this, you must click on the blue button **New Session**, located in the upper right.



6. When starting a new session, make sure:

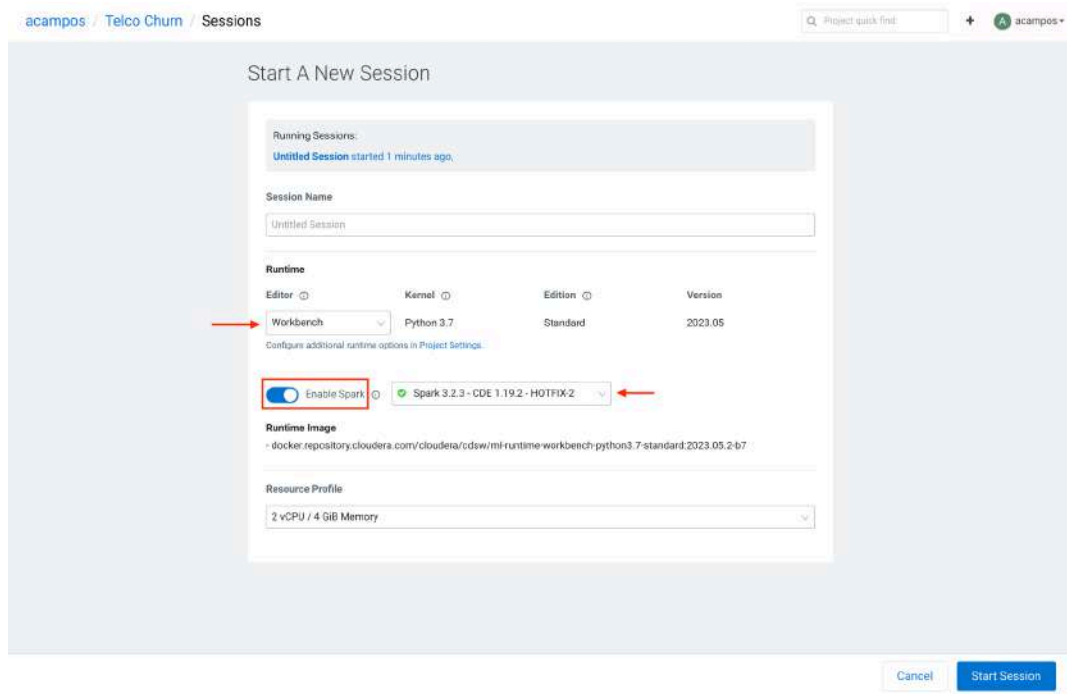
Select **Workbench** in the Editor selector.

Select **Python 3.7**, in the Spark version selector.

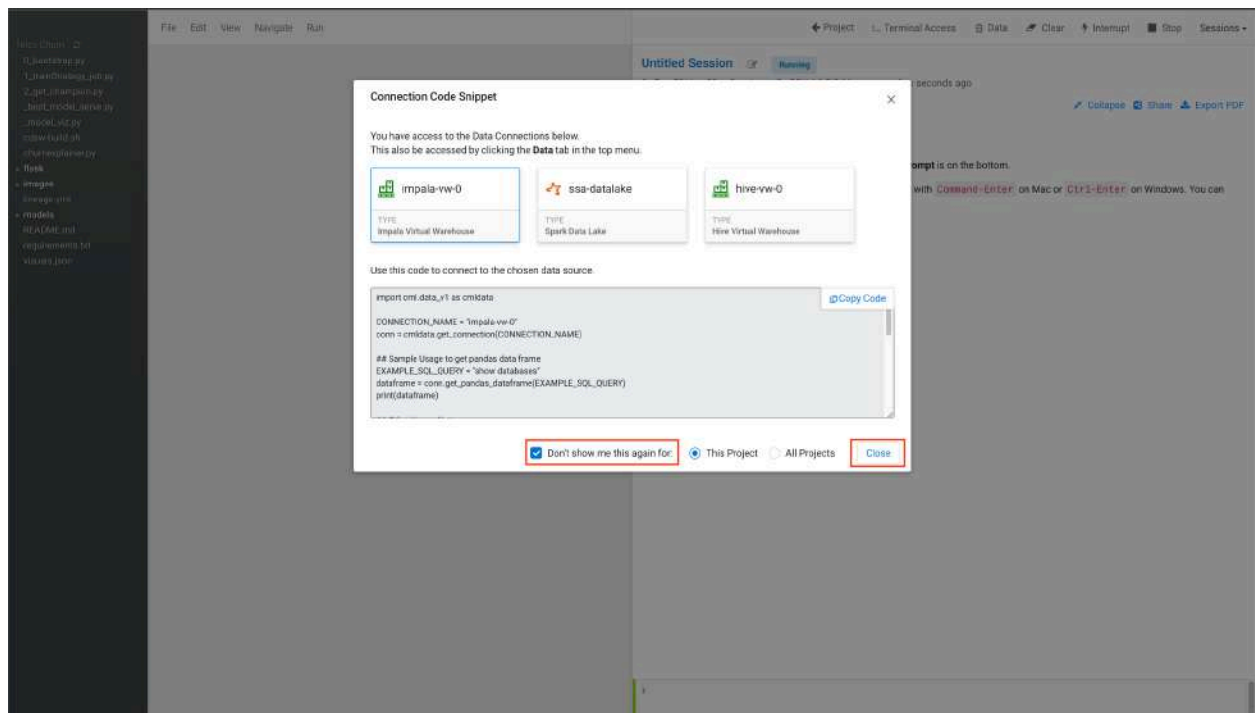
Enable **Spark**, marking the corresponding check.

Select **Spark 3.2.x**, in the Spark version selector.

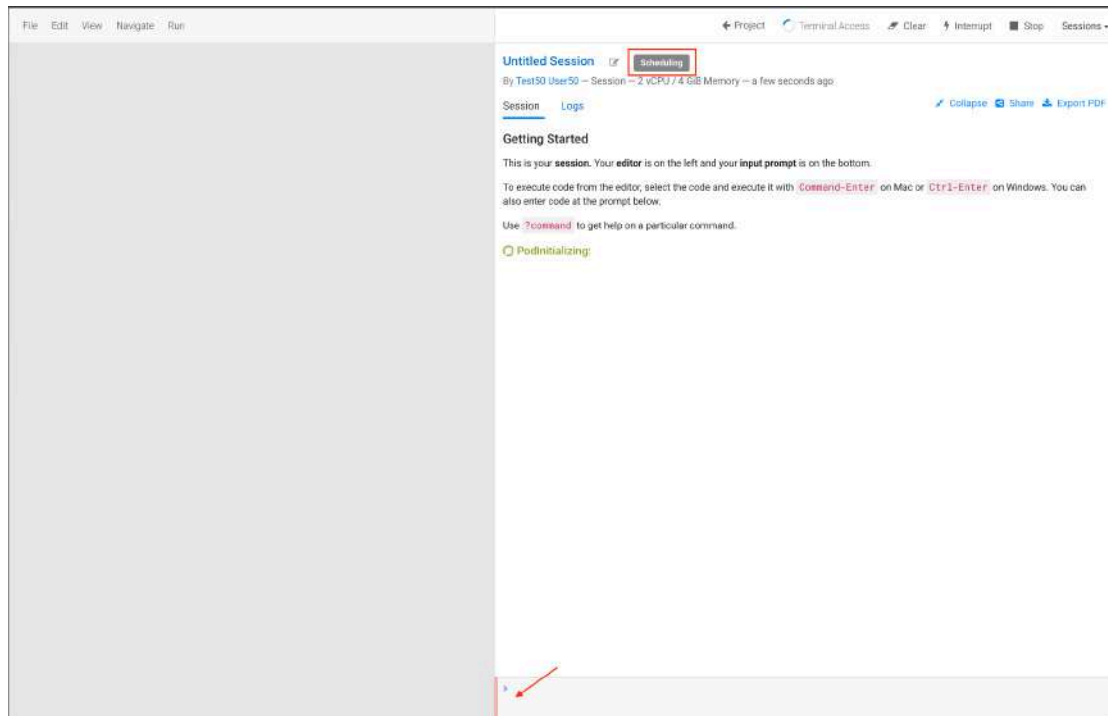
Click on the button **Start Session**



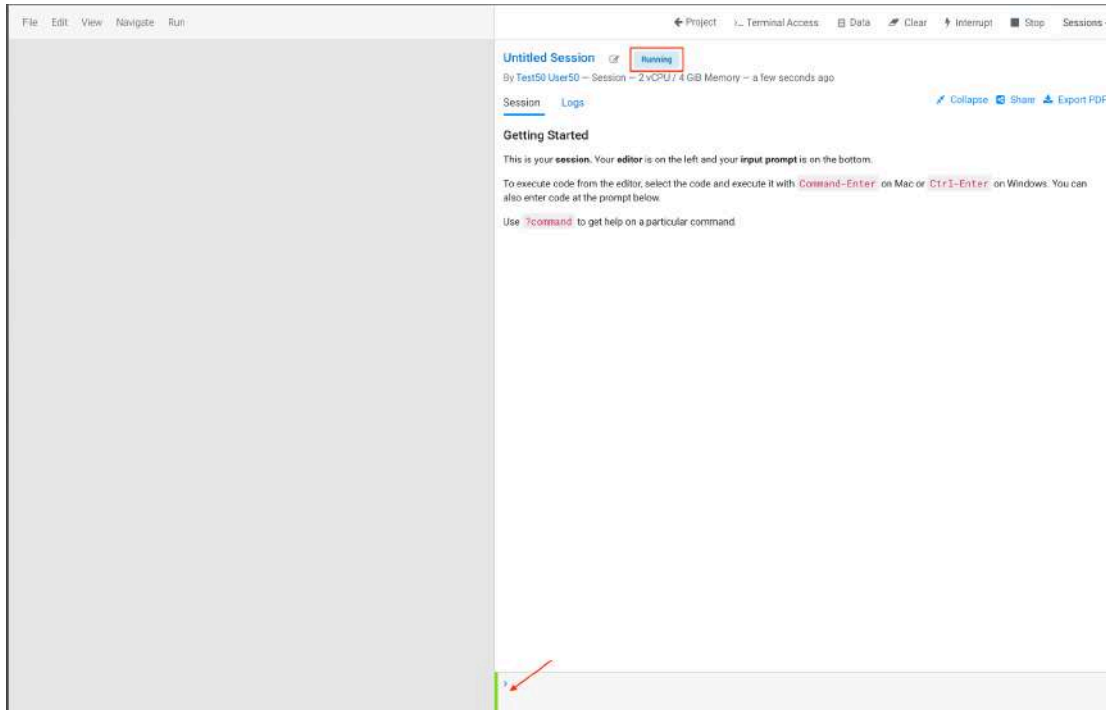
7. When you start a session for the first time, it will ask if you want to use a data connection. This project does not need this type of connection. mark the check of **Don't show me this again**, and then click the button **Close**, so this window will not appear anymore.




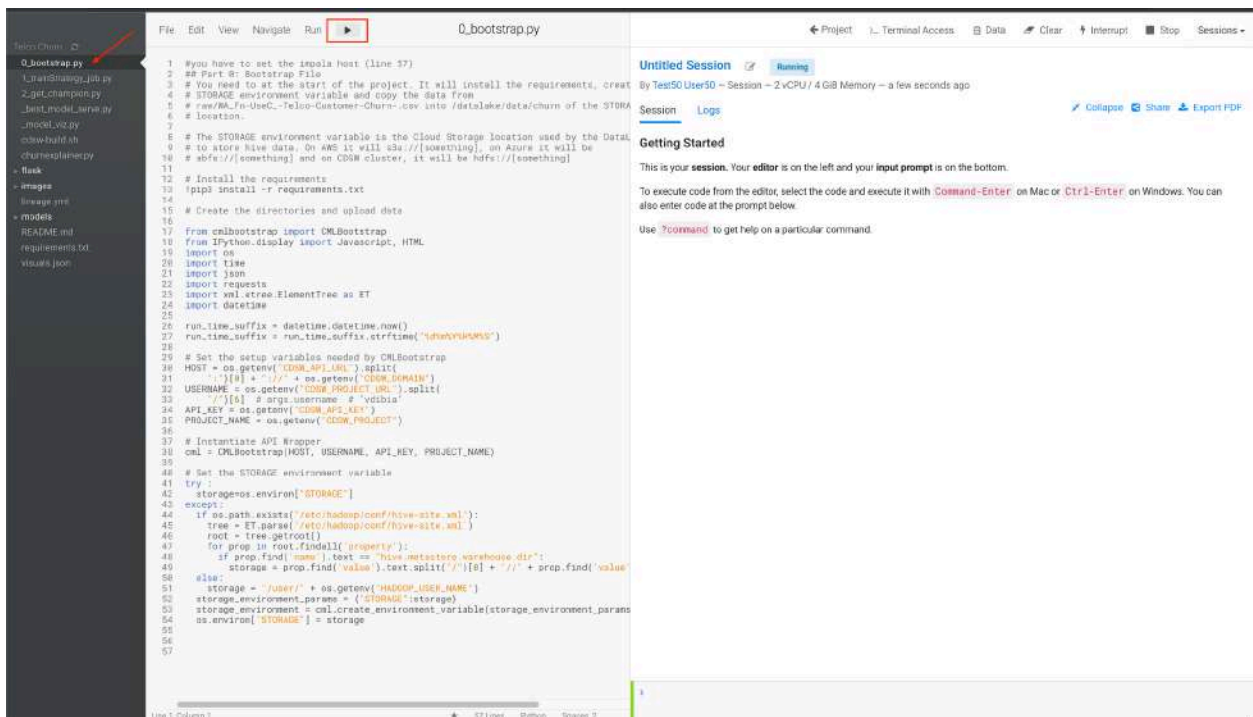
8. The editor/notebook located on the right side of the window will be in **Scheduling** status, and the bottom command bar flashing red. This means that CML is allocating computation for your session.



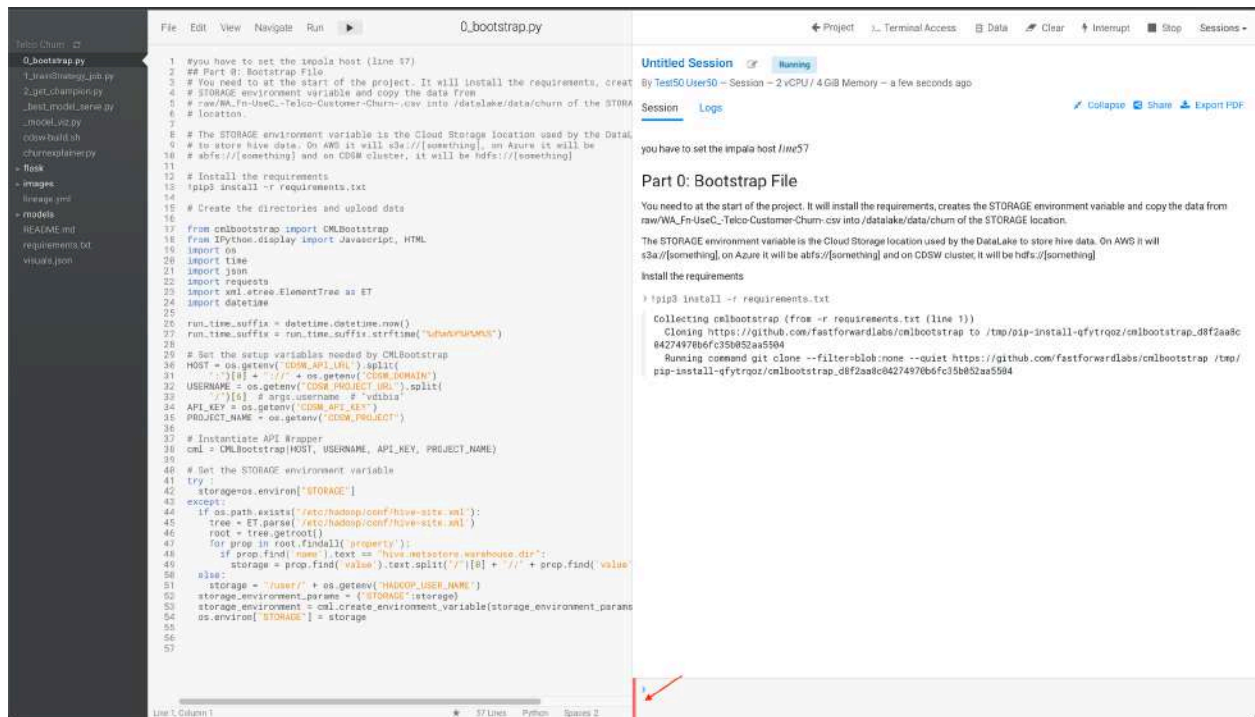
After a few seconds, the status changes to **Running**, and the command bar to green. This means that the session is ready to run code.



9. The first script/code to run is **0\_bootstrap.py**. This Python code configures the libraries required for the project and integration with Lakehouse tables you populated before. Select (just one click) the file in the bar located on the left side of the interface, this will make the code appear in the editor. Once the file is selected, click on the button  to run the code.

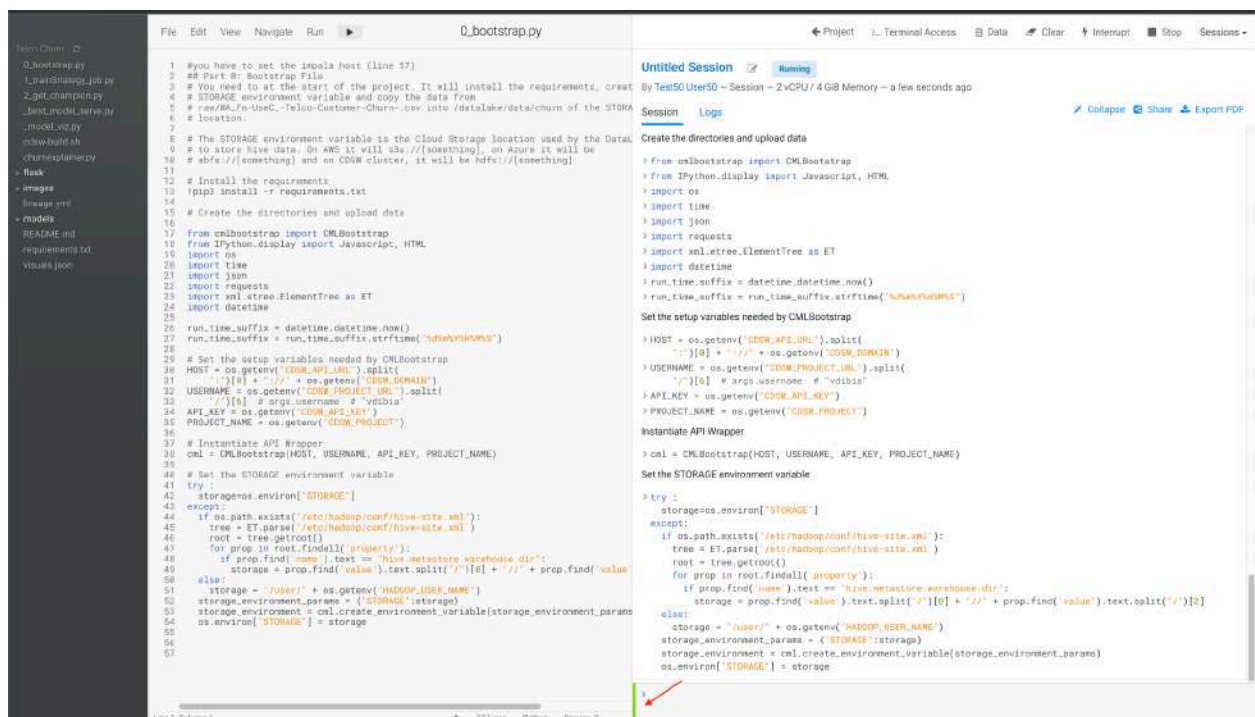


When you start execution, you will see code output on the right side of the interface, and the bottom command bar flashing red, indicating that it is busy.




```
1 #You have to set the impala host (line 17)
2 ## Part 0: Bootstrap File
3 # You need to at the start of the project. It will install the requirements, creat
4 # STORAGE environment variable and copy the data from
5 # raw/WA_Fn-UseC-Telco-Customer-Churn-csv into /datalake/data/churn of the STORA
6 # location.
7
8 # The STORAGE environment variable is the Cloud Storage location used by the DataL
9 # to store hive data. On AWS it will s3a://[something], on Azure it will be
10 # abfs://[something] and on CDGW cluster, it will be hdfs://[something]
11
12 # Install the requirements
13 pip3 install -r requirements.txt
14
15 # Create the directories and upload data
16
17 from cmlbootstrap import CMLBootstrap
18 from IPython.display import Javascript, HTML
19 import os
20 import time
21 import json
22 import requests
23 import xml.etree.ElementTree as ET
24 import datetime
25
26 run_time_suffix = datetime.datetime.now()
27 run_time_suffix = run_time_suffix.strftime("%H%M%S")
28
29 # Set the setup variables needed by CMLBootstrap
30 HOST = os.getenv("CDGW_API_URL").split(
31     ":[0] + "/" + os.getenv("CDGW_DOMAIN")
32 )
33 USERNAME = os.getenv("CDGW_PROJECT_URL").split(
34     ":[0] # args.username # 'vdiola'
35 )
36 API_KEY = os.getenv("CDGW_API_KEY")
37 PROJECT_NAME = os.getenv("CDGW_PROJECT")
38
39 # Instantiate API Wrapper
40 cml = CMLBootstrap(HOST, USERNAME, API_KEY, PROJECT_NAME)
41
42 # Set the STORAGE environment variable
43 try:
44     storageos.environ["STORAGE"]
45 except:
46     if os.path.exists("/etc/hadoop/conf/hive-site.xml"):
47         tree = ET.parse("/etc/hadoop/conf/hive-site.xml")
48         root = tree.getroot()
49         for prop in root.findall('property'):
50             if prop.find('name').text == "hive.metastore.warehouse.dir":
51                 storage = prop.find('value').text.split("/")[-1] + "/" + prop.find('value')
52             else:
53                 storage = "/user/" + os.getenv("HADOOP_USER_NAME")
54                 storage_environment_params = {"STORAGE":storage}
55                 storage_environment = cml.create_environment_variable(storage_environment_params)
56                 os.environ["STORAGE"] = storage
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

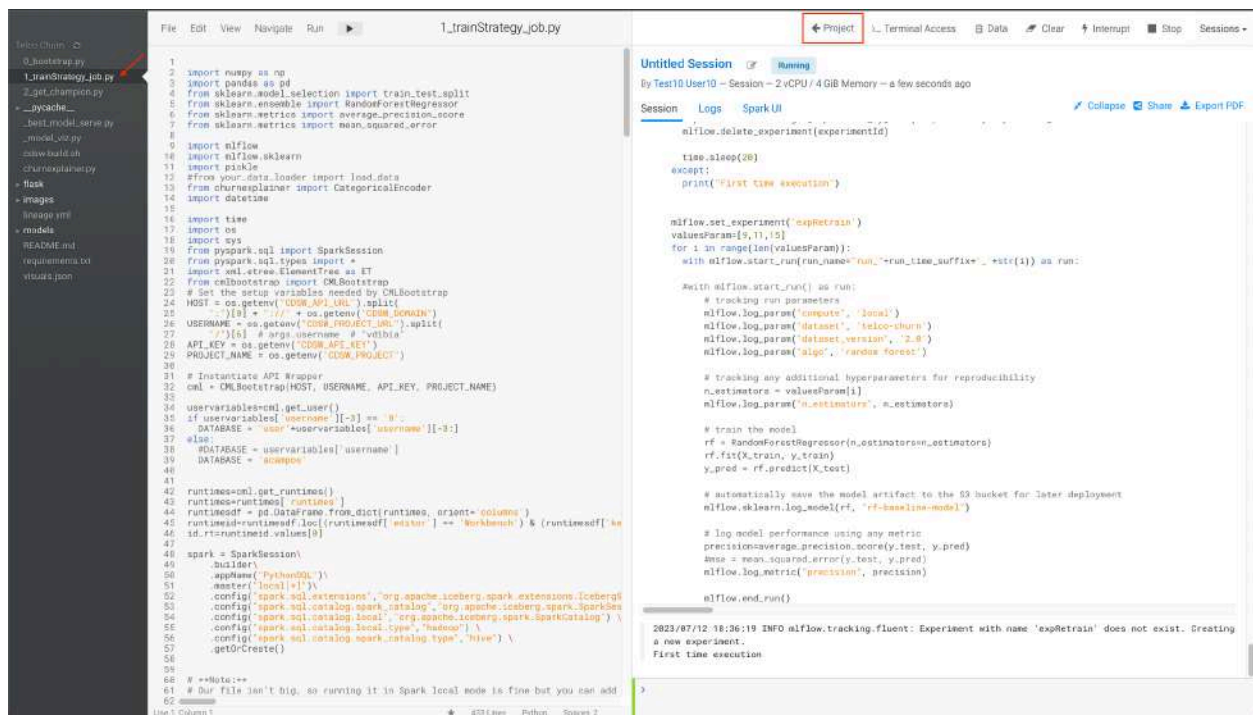
The green command bar indicates that the execution of the code has been finished. This bootstrap code takes 3-4 minutes to run.



```
1 #You have to set the impala host (line 17)
2 ## Part 0: Bootstrap File
3 # You need to at the start of the project. It will install the requirements, creat
4 # STORAGE environment variable and copy the data from
5 # raw/WA_Fn-UseC-Telco-Customer-Churn-csv into /datalake/data/churn of the STORA
6 # location.
7
8 # The STORAGE environment variable is the Cloud Storage location used by the DataL
9 # to store hive data. On AWS it will s3a://[something], on Azure it will be
10 # abfs://[something] and on CDGW cluster, it will be hdfs://[something]
11
12 # Install the requirements
13 pip3 install -r requirements.txt
14
15 # Create the directories and upload data
16
17 from cmlbootstrap import CMLBootstrap
18 from IPython.display import Javascript, HTML
19 import os
20 import time
21 import json
22 import requests
23 import xml.etree.ElementTree as ET
24 import datetime
25
26 run_time_suffix = datetime.datetime.now()
27 run_time_suffix = run_time_suffix.strftime("%H%M%S")
28
29 # Set the setup variables needed by CMLBootstrap
30 HOST = os.getenv("CDGW_API_URL").split(
31     ":[0] + "/" + os.getenv("CDGW_DOMAIN")
32 )
33 USERNAME = os.getenv("CDGW_PROJECT_URL").split(
34     ":[0] # args.username # 'vdiola'
35 )
36 API_KEY = os.getenv("CDGW_API_KEY")
37 PROJECT_NAME = os.getenv("CDGW_PROJECT")
38
39 # Instantiate API Wrapper
40 cml = CMLBootstrap(HOST, USERNAME, API_KEY, PROJECT_NAME)
41
42 # Set the STORAGE environment variable
43 try:
44     storageos.environ["STORAGE"]
45 except:
46     if os.path.exists("/etc/hadoop/conf/hive-site.xml"):
47         tree = ET.parse("/etc/hadoop/conf/hive-site.xml")
48         root = tree.getroot()
49         for prop in root.findall('property'):
50             if prop.find('name').text == "hive.metastore.warehouse.dir":
51                 storage = prop.find('value').text.split("/")[-1] + "/" + prop.find('value')
52             else:
53                 storage = "/user/" + os.getenv("HADOOP_USER_NAME")
54                 storage_environment_params = {"STORAGE":storage}
55                 storage_environment = cml.create_environment_variable(storage_environment_params)
56                 os.environ["STORAGE"] = storage
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```



10. The second script/code to run is **1\_trainStrategy\_job.py**. This Python code will create the Experiment to run the model with three different hyper parameters and records the precision. Select (just one click) the file in the bar located on the left side of the interface, this will make the code appear in the editor. Once the file is selected, click on the button  to run the code. Once the execution is finished (approximately 1 minute), click on the button **Project**, located in the upper right bar of the session to go back to the project home.



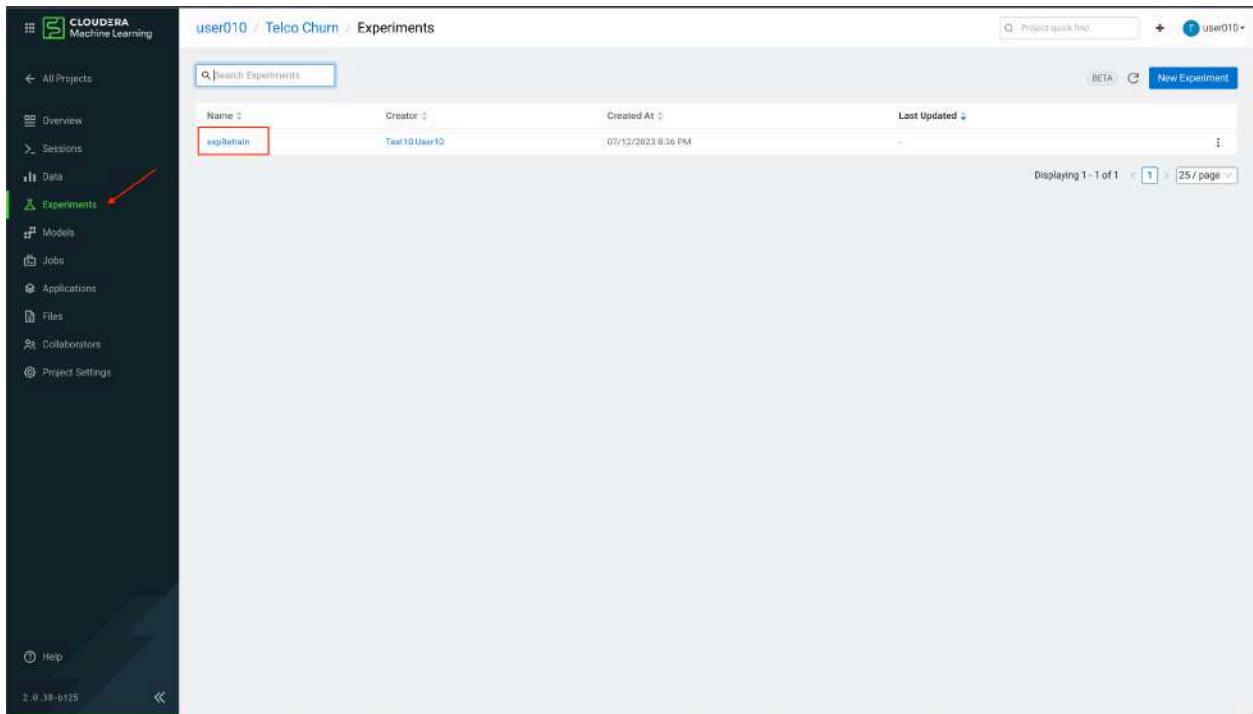
The screenshot displays the Databricks workspace interface. On the left, a file explorer shows the project structure with files like 0\_bootstrap.py, 1\_trainStrategy\_job.py, 2\_get\_champion.py, and others. The file 1\_trainStrategy\_job.py is selected. The main editor area shows the Python code for training a model using mlflow and Spark. The code includes imports for numpy, pandas, sklearn, mlflow, and SparkSession, followed by data loading and model training logic. On the right, the 'Untitled Session' is shown in a 'Running' state. The session log displays the execution progress, including the creation of the experiment 'expRetrain' and the first time execution. The session bar at the top right has a 'Project' button highlighted with a red box.

```

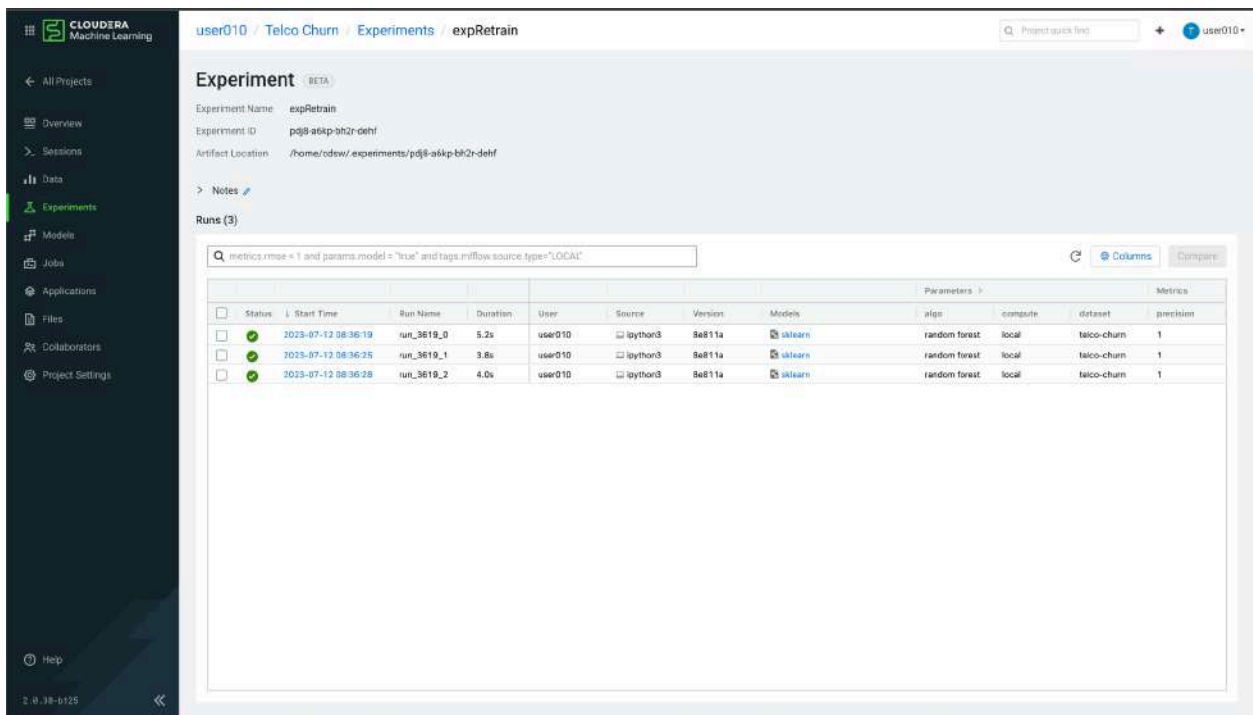
1 import numpy as np
2 import pandas as pd
3 from sklearn.model_selection import train_test_split
4 from sklearn.ensemble import RandomForestRegressor
5 from sklearn.metrics import average_precision_score
6 from sklearn.metrics import mean_squared_error
7
8 import mlflow
9
10 import mlflow.sklearn
11 import pickle
12
13 #from your_data_loader import load_data
14 from churnexplainer import CategoricalEncoder
15 import database
16
17 import time
18 import os
19 import sys
20 from pyspark.sql import SparkSession
21 from pyspark.sql.types import *
22 import xml.etree.ElementTree as ET
23 from mlbootstrap import MLBootstrap
24 # Set the setup variables needed by CMLBootstrap
25 HOST = os.getenv('CML_API_URL').split(
26     '/')[-1] + '/' + os.getenv('CML_DOMAIN')
27
28 //[[[1] # args: username # 'volbia'
29
30 API_KEY = os.getenv('CML_API_KEY')
31 PROJECT_NAME = os.getenv('CML_PROJECT')
32
33 # Instantiate API wrapper
34 cml = MLBootstrap(HOST, USERNAME, API_KEY, PROJECT_NAME)
35
36 user_variables=cml.get_user()
37
38 if user_variables['username'][-3:] == 'B':
39     DATABASE = 'user'+user_variables['username'][-3:]
40 else:
41     DATABASE = user_variables['username']
42     DATABASE = 'scmpops'
43
44 runtime=cml.get_runtime()
45 runtime=runtime['runtime']
46 runtime_df = pd.DataFrame.from_dict(runtime, orient='columns')
47 runtime_id=runtime['id']
48 id=runtime['id']
49
50 spark = SparkSession(
51     builder()
52     .appName('PythonDGL')
53     .master('local[*]')
54     .config('spark.sql.extensions','org.apache.iceberg.spark.extensions.IcebergSparkExtensions')
55     .config('spark.sql.catalog.spark_catalog','org.apache.iceberg.spark.SparkCatalog')
56     .config('spark.sql.catalog.local','org.apache.iceberg.spark.SparkCatalog')
57     .config('spark.sql.catalog.local.type','hive')
58     .config('spark.sql.catalog.spark_catalog.type','hive')
59     .getOrCreate()
60
61 # **Note:**
62 # Our file isn't big, so running it in Spark local mode is fine but you can add
63

```

11. Once back in project home, click on the **Experiments** option, from the left menu, and then on **expRetrain** in the list of Experiments that appears.

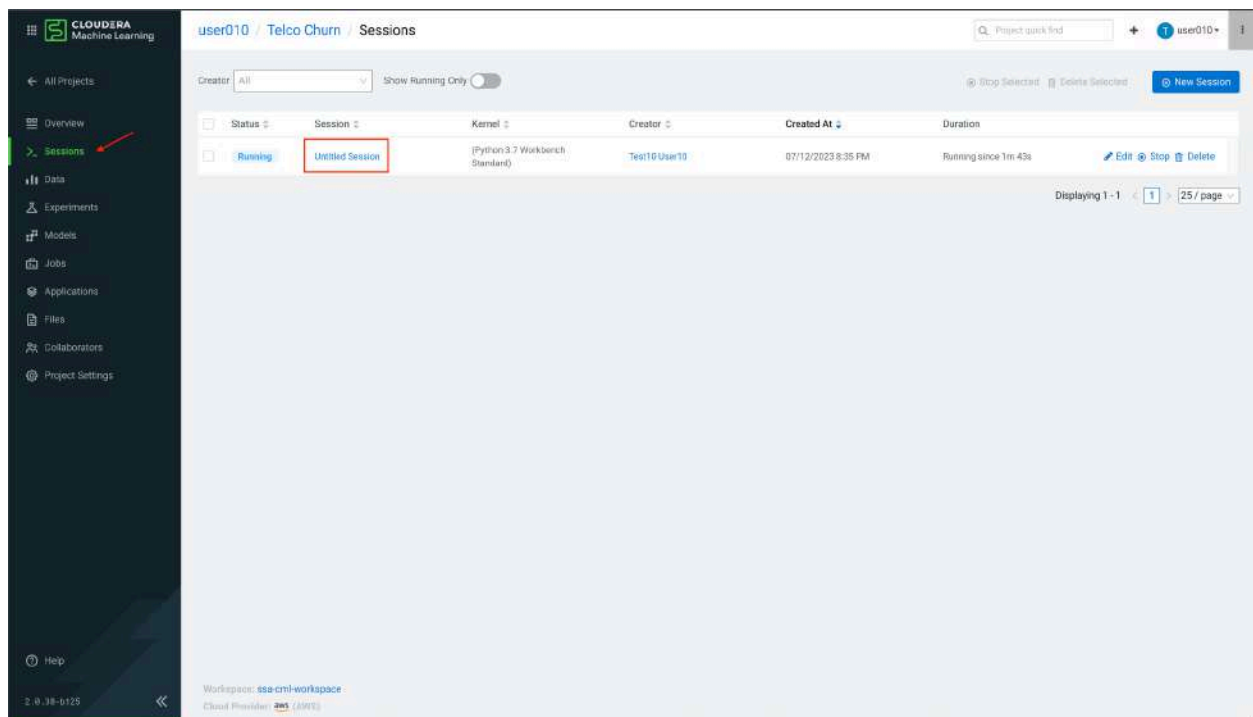



12. On this screen you will see the three runs of this experiment. Look at the last column, where **precision** attribute displays. This is the precision that each hyper parameter is delivering.



13. Let's go back to the session to run the last code. Since sessions run in Kubernetes containers, it's very easy to get back to where we were. Click on the option **Sessions** from the

left menu, and later in the only session that will appear in the list. If you didn't name your session when you started it (step 6), it should be called *Untitled Session*.



14. The third and last script/code to run is **2\_get\_champion.py**. This Python code takes the hyper parameter of the execution of the Experiment with better precision and deploys two Models as REST API, one to be integrated in Data Visualization and another for unit use for calls. Select (just one click) the file in the bar located on the left side of the interface, this will make the code appear in the editor. Once the file is selected, click on the button  to run the code.

The screenshot shows a JupyterLab environment with a file explorer on the left containing files like 0\_bootstrap.py, 1\_train\_strategy\_job.py, 2\_get\_champion.py, and others. The main editor displays the 2\_get\_champion.py script, which imports sys, mlflow, and sklearn, sets up an experiment named 'exp@retrain', and runs a search for the best model using mlflow.search\_runs. The script also includes a loop to delete the experiment and a function to train a model and log it to mlflow.

```
1 |
2 | import sys
3 | import mlflow
4 | import mlflow.sklearn
5 | mlflow.set_experiment('Retrain_exp')
6 | experiment_id=mlflow.get_experiment_by_name('exp@retrain').experiment_id
7 | dfExperiments=mlflow.search_runs(experiment_id=experiment_id)
8 | maxmetric=dfExperiments['metrics.precision'].max()
9 | runId=dfExperiments[dfExperiments['metrics.precision']==maxmetric].head(1).run_id
10 |
11 | script_descriptor = open('1_train_strategy_job.py')
12 | a_script = script_descriptor.read()
13 | sys.argv = ['1_train_strategy_job.py', runId.item()]
14 |
15 | exec(a_script)
16 |
```

The terminal output shows the execution of the script, including the creation of a new experiment, the first time execution, and the logging of model performance metrics.

```
2023/07/12 15:36:19 INFO mlflow.tracking.fluent: Experiment with name 'exp@retrain' does not exist. Creating a new experiment.
First time execution

mlflow.set_experiment('exp@retrain')
valuesParam=[9,11,15]
for i in range(len(valuesParam)):
    with mlflow.start_run(run_name='Run_'+run_time_suffix+str(i)) as run:
        #with mlflow.start_run() as run:
            # tracking run parameters
            mlflow.log_param('compute', 'local')
            mlflow.log_param('dataset', 'talca-churn')
            mlflow.log_param('dataset_version', '2.0')
            mlflow.log_param('algo', 'random forest')

            # tracking any additional hyperparameters for reproducibility
            n_estimators = valuesParam[i]
            mlflow.log_param('n_estimators', n_estimators)

            # train the model
            rf = RandomForestRegressor(n_estimators=n_estimators)
            rf.fit(X_train, y_train)
            y_pred = rf.predict(X_test)

            # automatically save the model artifact to the S3 bucket for later deployment
            mlflow.sklearn.log_model(rf, 'rf-baseline-model')

            # log model performance using any metric
            precision=average_precision_score(y_test, y_pred)
            mse = mean_squared_error(y_test, y_pred)
            mlflow.log_metric('precision', precision)

mlflow.end_run()
```

After a few seconds, you will see the following message “Deploying Model...” repeated several times, and the bottom command bar will be red.

The screenshot shows the same JupyterLab environment, but the terminal output is different. It shows the execution of the script, including the creation of a new experiment, the first time execution, and the logging of model performance metrics. The output also shows a FutureWarning about the 'item' attribute being deprecated, and a series of 'Deploying Model...' messages.

```
1 |
2 | import sys
3 | import mlflow
4 | import mlflow.sklearn
5 | mlflow.set_experiment('Retrain_exp')
6 | experiment_id=mlflow.get_experiment_by_name('exp@retrain').experiment_id
7 | dfExperiments=mlflow.search_runs(experiment_id=experiment_id)
8 | maxmetric=dfExperiments['metrics.precision'].max()
9 | runId=dfExperiments[dfExperiments['metrics.precision']==maxmetric].head(1).run_id
10 |
11 | script_descriptor = open('1_train_strategy_job.py')
12 | a_script = script_descriptor.read()
13 | sys.argv = ['1_train_strategy_job.py', runId.item()]
14 |
15 | exec(a_script)
16 |
```

The terminal output shows the execution of the script, including the creation of a new experiment, the first time execution, and the logging of model performance metrics. The output also shows a FutureWarning about the 'item' attribute being deprecated, and a series of 'Deploying Model...' messages.

```
2023/07/12 15:36:19 INFO mlflow.tracking.fluent: Experiment with name 'exp@retrain' does not exist. Creating a new experiment.
First time execution

> import sys
> import mlflow
> import mlflow.sklearn

mlflow.set_experiment('Retrain_exp')

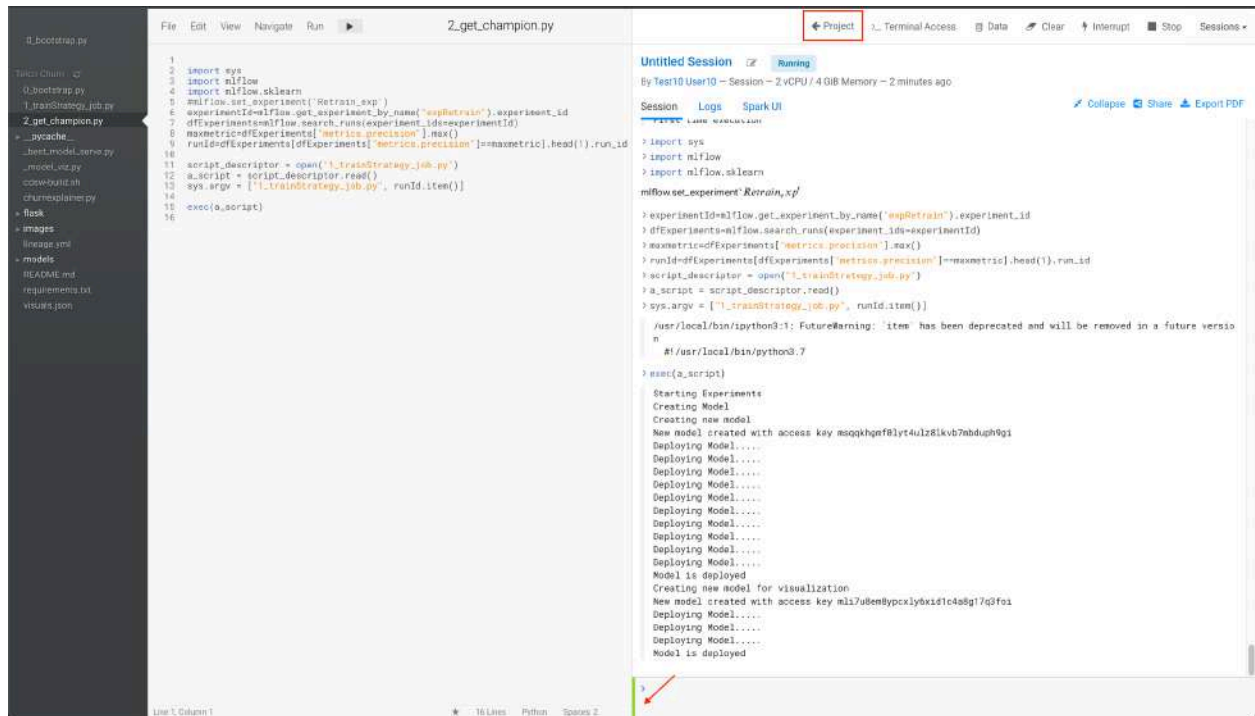
> experiment_id=mlflow.get_experiment_by_name('exp@retrain').experiment_id
> dfExperiments=mlflow.search_runs(experiment_id=experiment_id)
> maxmetric=dfExperiments['metrics.precision'].max()
> runId=dfExperiments[dfExperiments['metrics.precision']==maxmetric].head(1).run_id
> script_descriptor = open('1_train_strategy_job.py')
> a_script = script_descriptor.read()
> sys.argv = ['1_train_strategy_job.py', runId.item()]

/usr/local/bin/python3.11: FutureWarning: 'item' has been deprecated and will be removed in a future version
#1/usr/local/bin/python3.7

> exec(a_script)

Starting Experiments
Creating Model
Creating new model
New model created with access key msqkqh9f81yt4ul281kxb7abdh9gi
Deploying Model.....
Deploying Model.....
Deploying Model.....
Deploying Model.....
Deploying Model.....
Deploying Model.....
Deploying Model.....
Deploying Model.....
Model is deployed
Creating new model for visualization
New model created with access key ml2u0B8nypxixy6xid1c48sg7q3foi
Deploying Model.....
Deploying Model.....
```

After about 2 minutes, the last message should be "Model is deployed", and the bar will be green. It means that the Deployment of the two Models is complete. Click on the button **Project**, located in the upper right bar of the session to return to the home page of the project.



```
1 import sys
2 import mlflow
3 import mlflow.sklearn
4 mlflow.set_experiment('Retrain_exp')
5 experimentId=mlflow.get_experiment_by_name('exp@Retrain').experiment_id
6 dfExperiments=mlflow.search_runs(experiment_id=experimentId)
7 maxMetric=dfExperiments['metrics.precision'].max()
8 runId=dfExperiments[dfExperiments['metrics.precision']==maxMetric].head(1).run_id
9
10 script_descriptor = open('1_trainStrategy_job.py')
11 a_script = script_descriptor.read()
12 sys.argv = ['1_trainStrategy_job.py', runId.item()]
13
14 exec(a_script)
15
16
```

```
> import sys
> import mlflow
> import mlflow.sklearn
mlflow.set_experiment('Retrain_exp')
> experimentId=mlflow.get_experiment_by_name('exp@Retrain').experiment_id
> dfExperiments=mlflow.search_runs(experiment_id=experimentId)
> maxMetric=dfExperiments['metrics.precision'].max()
> runId=dfExperiments[dfExperiments['metrics.precision']==maxMetric].head(1).run_id
> script_descriptor = open('1_trainStrategy_job.py')
> a_script = script_descriptor.read()
> sys.argv = ['1_trainStrategy_job.py', runId.item()]
> sys.argv = ['1_trainStrategy_job.py', runId.item()]
/usr/local/bin/python3.7: FutureWarning: 'item' has been deprecated and will be removed in a future version
#!/usr/local/bin/python3.7
> exec(a_script)
Starting Experiments
Creating Model
Creating new model
New model created with access key msqkqh7f81y4u1z8ikv7bhdup9gi
Deploying Model....
Deploying Model....
Deploying Model....
Deploying Model....
Deploying Model....
Deploying Model....
Deploying Model....
Deploying Model....
Deploying Model....
Deploying Model....
Model is deployed
Creating new model for visualization
New model created with access key ml17u8en8pckly6x1d1c4d8g17d3foi
Deploying Model....
Deploying Model....
Deploying Model....
Deploying Model....
Model is deployed
```

15. Once on the home page of the project, you will see the Models displayed, which are two. Click on the one that starts with **ModelOpsChurn**.

**Models**

Model	Source	Status	Replicas	CPU	Memory	Last Deployed	Actions
ModelViz_user010	_model...	Deployed	1 / 1	1	2.00 GiB	Jul 12, 2023, 08:39 PM	Stop
<b>ModelOpsChurn_user010</b>	_best...	Deployed	1 / 1	1	2.00 GiB	Jul 12, 2023, 08:39 PM	Stop

**Jobs**

This project has no jobs yet. Create a [new job](#) to document your analytics pipelines.

**Files**

Name	Size	Last Modified
._pycache_	-	9 minutes ago
flask	-	15 minutes ago
images	-	15 minutes ago
models	-	15 minutes ago
0_bootstrap.py	1.35 KiB	15 minutes ago
1_trainStrategy_job.py	18.63 KiB	15 minutes ago
2_get_champion.py	508 B	15 minutes ago
_best_model_serve.py	2.74 KiB	15 minutes ago
_model_viz.py	4.21 KiB	15 minutes ago
cdsw-build.sh	44 B	15 minutes ago
churnexplainer.py	6.69 KiB	15 minutes ago
lineage.yml	610 B	3 minutes ago

16. Here you will see Model information and settings in the Overview tab.

**ModelOpsChurn\_user010**

[Overview](#) [Deployments](#) [Builds](#) [Monitoring](#) [Logs](#) [Settings](#)

**Description**

Explain a given model prediction

Sample Code

```
curl -H 'Content-Type: application/json' -X POST https://modelservice.sl-1e5bb8eb-444.ssa-hol-yult-vbzg.cloudera.site/model -d '{"accessKey":"msqghgmF0lyt4ulz8lkv7mbdph9gi","request":{"streamingtv":"No","monthlycharges":78.35,"phoneservice":"No","paperlessbilling":"No","partner":"No","onlinebackup":"No","gender":"female","contract":"Month-to-month","totalcharges":1397.475,"streamingmovies":"No","deviceprotection":"No","paymentmethod":"Bank transfer (automatic)","tenure":29,"dependents":"No","onlinesecurity":"No","multiplelines":"No","internetservice":"DSL","seniorcitizen":"No","techsupport":"No"}}'
or
curl -H 'Content-Type: application/json' -X POST https://modelservice.sl-1e5bb8eb-444.ssa-hol-yult-vbzg.cloudera.site/model?accessKey=msqghgmF0lyt4ulz8lkv7mbdph9gi -d '{"request":{"streamingtv":"No","monthlycharges":78.35,"phoneservice":"No","paperlessbilling":"No","partner":"No","onlinebackup":"No","gender":"female","contract":"Month-to-month","totalcharges":1397.475,"streamingmovies":"No","deviceprotection":"No","paymentmethod":"Bank transfer (automatic)","tenure":29,"dependents":"No","onlinesecurity":"No","multiplelines":"No","internetservice":"DSL","seniorcitizen":"No","techsupport":"No"}}'
```

Sample Response

```
{}
```

**Test Model**

Input

```
{
  "onlinesecurity": "No",
  "multiplelines": "No",
  "internetservice": "DSL",
  "seniorcitizen": "No",
  "techsupport": "No"
}
```

**Model Details**

Source	Code
Model Id	19
Model CRN	cm:cdpmlus-west-1:508f488f-8076-498a-acfb-6f8765cd35e8:workspace:1e48b728-bc0f-4867-8a54-fb309e-09355/19770cd9-b00e-4354-963a-746af4c79e8
Deployment Id	14
Deployment CRN	cm:cdpmlus-west-1:508f488f-8076-498a-acfb-6f8765cd35e8:workspace:1e48b728-bc0f-4867-8a54-fb309e-09355/32aw37d1-afbb-4225-afed-46a5c60f3109
Build Id	14
Build CRN	cm:cdpmlus-west-1:508f488f-8076-498a-acfb-6f8765cd35e8:workspace:1e48b728-bc0f-4867-8a54-fb309e-09355/19770cd9-b00e-4354-963a-746af4c79e8
Deployed By	user010
Comment	Initial revision.
Runtime Image	Python 3.7 (Standard)
File	_best_model_serve.py
Function	explain

**Model Resources**

Replicas	Total CPU	Total Memory
1	1 vCPUs	2.00 GiB

To test it and make a request to the model, scroll down, and click on the button **Test**, which will take the value in JSON format that is in the field **Input** and will make the request call to the model. What you see in the field **Result** is the response from the model in JSON format. If you

wish, you can change some of the parameters of the **Input** field (for example, change some values from *Not* to *Yes*), and call the model again, and observe the value of the attribute *probability* of the response to see if there were any changes.

The screenshot displays the Cloudera Machine Learning interface for a project named 'Telco Churn'. The 'Test Model' section is active, showing the input parameters for a model. The input is a JSON object with the following values: 'onlinesecurity': 'No', 'multiplesites': 'No', 'internetservice': 'DSL', 'senderemailsize': 'No', and 'techsupport': 'No'. The 'Test' button is highlighted with a red box. Below the input, the 'Result' section shows a 'success' status. The 'Response' is a JSON object containing 'model\_deployment\_arn', 'prediction' (with a 'probability' value of 0.5555555555555556), and 'uuid'. The 'Replica ID' is also displayed. The interface includes a sidebar with navigation options like 'All Projects', 'Overview', 'Sessions', 'Data', 'Experiments', 'Models', 'Jobs', 'Applications', 'Files', 'Collaborators', and 'Project Settings'. The top right shows the user 'user010' and a search bar.

Sample Response

```
{}
```

Test Model

Input

```
{  "onlinesecurity": "No",  "multiplesites": "No",  "internetservice": "DSL",  "senderemailsize": "No",  "techsupport": "No"}
```

**Test** [Reset](#)

Result

Status: ● success

Response

```
{  "model_deployment_arn": "arn:cdp:ml:us-west-1:508f480f-8076-498a-acfb-6f8765cd35e8:workspace:1e48b728-bc9f-4867-8a54-f93899c99355/32aa37d1-af8b-4225-a66d-4ca5",  "prediction": {    "probability": 0.5555555555555556  },  "uuid": "95a97cf3-36d3-459e-9372-b2b51334ca63"}
```

Replica ID: [modelopschurn-user818-19-14-6c5d7947ff-52kzg](#)

Workspace: [ss2-cml-workspace](#)  
Cloud Provider: [aws](#) (AWS)