

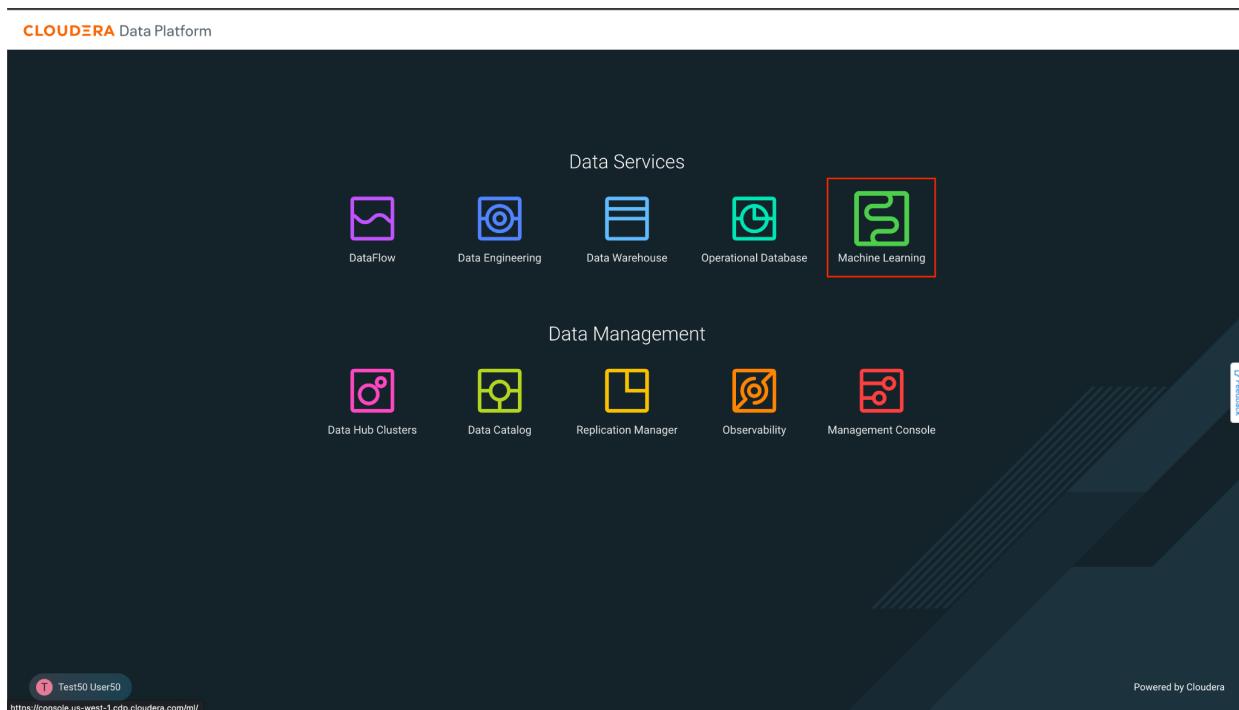
Data Lifecycle en CDP Public Cloud

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.

The screenshot shows the 'Machine Learning Workspaces' page. On the left, there's a sidebar with options like 'Workspaces', 'Workspace Backups', 'Model Registries', 'Help', and 'Test50 User50'. The main area has a table titled 'Machine Learning Workspaces' with columns: Status, Version, Workspace, Environment, Region, Creation Date, Cloud Provider, and Actions. A single row is listed: 'Ready' status, '2.0.38' version, 'ssa-cml-workspace' workspace (highlighted with a red box), 'ssa-hol' environment, 'unknown' region, '07/07/2023 11:42 PM CEST' creation date, 'aws' cloud provider, and an 'Actions' button. At the bottom, it says 'Displaying 1 - 1 of 1' and '25 / page'.

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**.

The screenshot shows the 'Projects' page. The sidebar includes 'Projects', 'Sessions', 'Experiments', 'Models', 'Jobs', 'Applications', 'User Settings', 'AMPs', 'Runtime Catalog', 'Site Administration', and 'Learning Hub'. The main area has a table for 'Search Projects' with 'Scope' set to 'My Projects' and 'Creator' set to 'All'. Below this, a message says 'You currently don't have any projects' with a small icon. A detailed description follows: 'Projects are the heart of Cloudera Machine Learning. They hold all the code, configuration, and libraries needed to reproducibly run analyses. Each project is independent, ensuring users can work freely without interfering with one another or breaking existing workloads.' A blue 'New Project' button is highlighted with a red box at the bottom of the message area. At the bottom of the page, it says 'Workspace: ssa-cml-workspace' and 'Cloud Provider: aws (AWS)'.

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/campossalex/TelcoChurn>

Keep the rest of the settings the same. Click the button **Create Project**

New Project

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/campossalex/TelcoChurn

Runtime setup
Basic Advanced

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

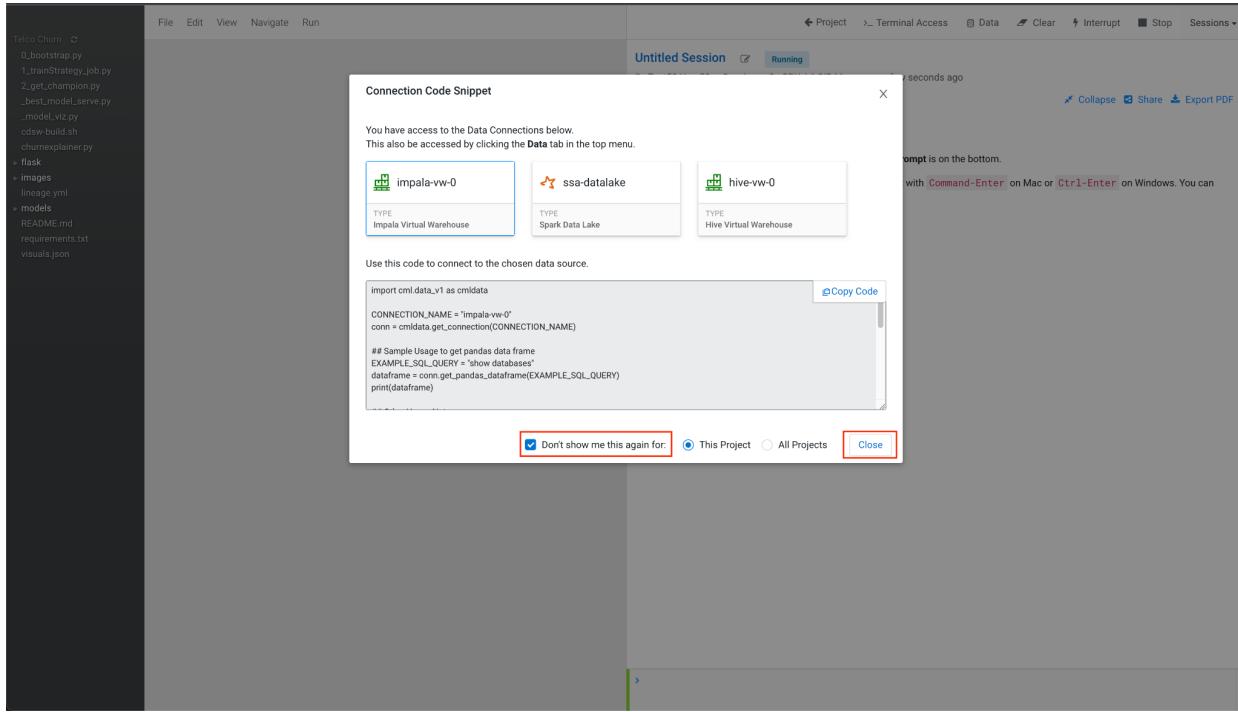
This Telco Churn project consists of running three scripts. The way of execution is through a session, which is the allocation of isolated compute resource for each user. For this, you must click on the blue button **New Session**, located in the upper right.

The screenshot shows the Cloudera Machine Learning interface. On the left, there's a sidebar with navigation links: All Projects, Overview, Sessions, Data, Experiments, Models, Jobs, Applications, Files, Collaborators, and Project Settings. The main area is titled 'Telco Churn'. It has sections for 'Models' (with a note that no models exist), 'Jobs' (with a note that no jobs exist), and 'Files'. The 'Files' section contains a table of contents for a directory structure. The table includes columns for Name, Size, and Last Modified. The files listed are flask, images, models, 0_bootstrap.py, 1_trainStrategy_job.py, 2_get_champion.py, _best_model_serve.py, _model_viz.py, cds-build.sh, chumexplainer.py, lineage.yml, README.md, requirements.txt, and visuals.json. At the bottom of the file list, there's a link to 'Show Hidden Files'. The bottom of the page shows workspace and cloud provider information: 'Workspace: ssa-cml-workspace' and 'Cloud Provider: AWS (AWS)'.

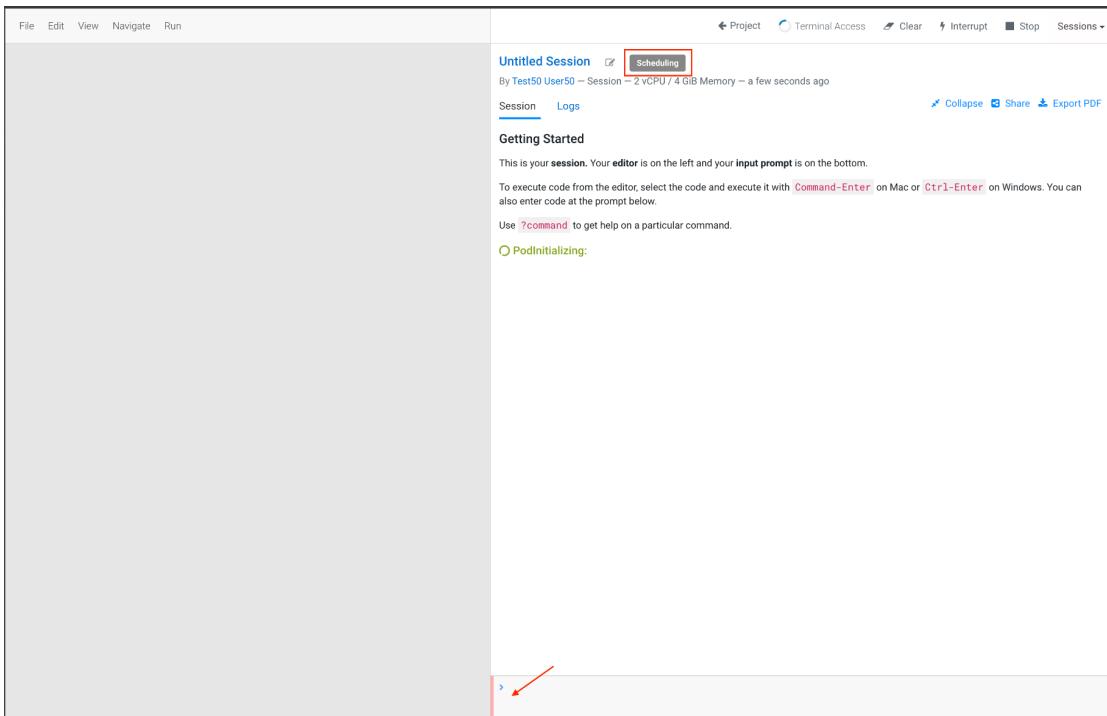
6. Make sure to enable Spark, marking the corresponding check, and click on the button **Start Session**.

The screenshot shows the 'Start A New Session' dialog box. It has fields for 'Session Name' (set to 'Untitled Session'), 'Runtime' (Editor: Workbench, Kernel: Python 3.7, Edition: Standard, Version: 2023.05), and 'Resource Profile' (2 vCPU / 4 GiB Memory). Below these, there are two radio buttons: 'Enable Spark' (which is selected) and 'Spark 2.4.8 - CDE 1.18.1 - HOTFIX-4'. The 'Runtime Image' dropdown is set to '- docker.repository.cloudera.com/cloudera/cds-wb/python3.7-standard 2023.05.2-b7'. At the bottom right, there are 'Cancel' and 'Start Session' buttons, with 'Start Session' being highlighted with a red box.

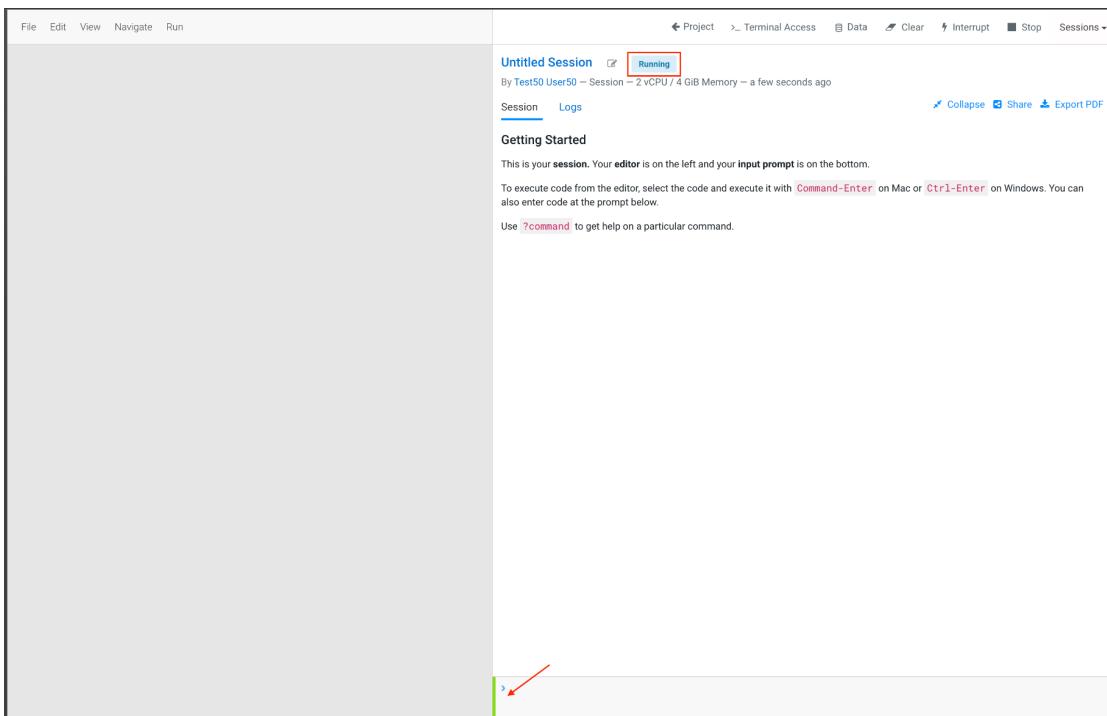
7. When you start a session for the first time, it will ask if you want to use a data connection. This project does not need this type of connection. mark the check of **Don't show me this again**, and then click the button **Close**, so this window will not appear anymore.



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.

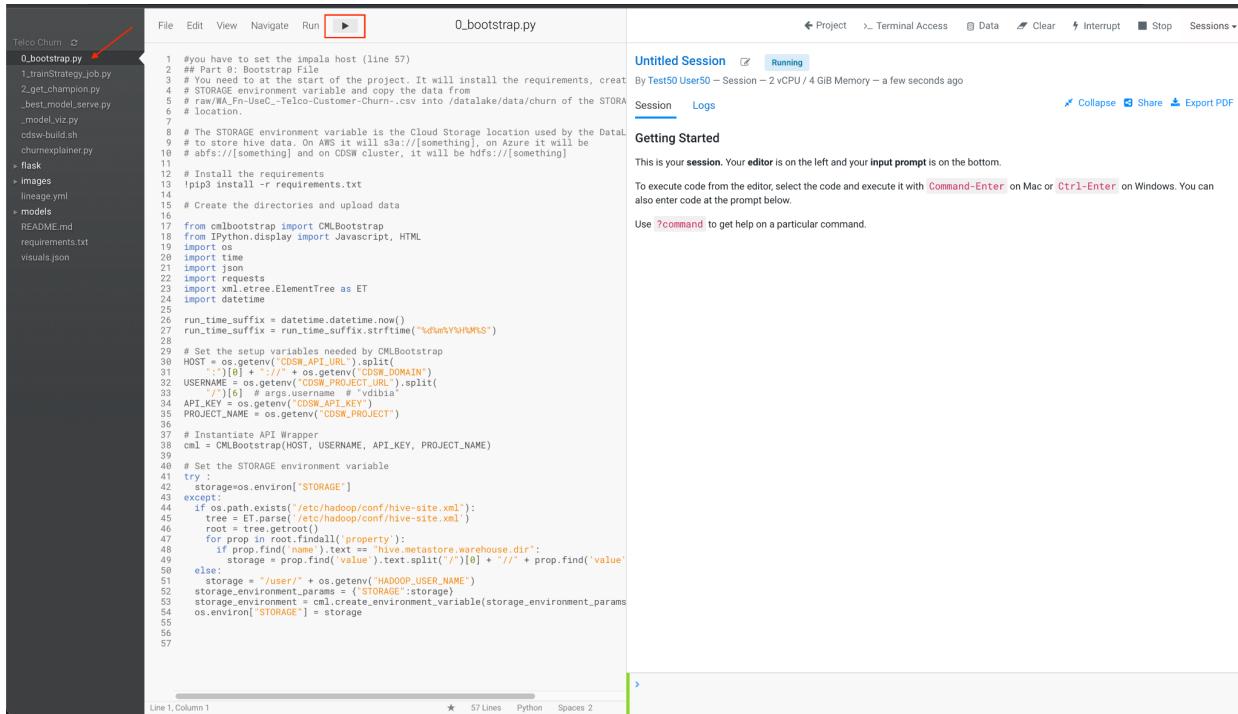


After a few seconds, the status changes to **Running**, and the command bar to green. This means that the session is ready to run code.



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.



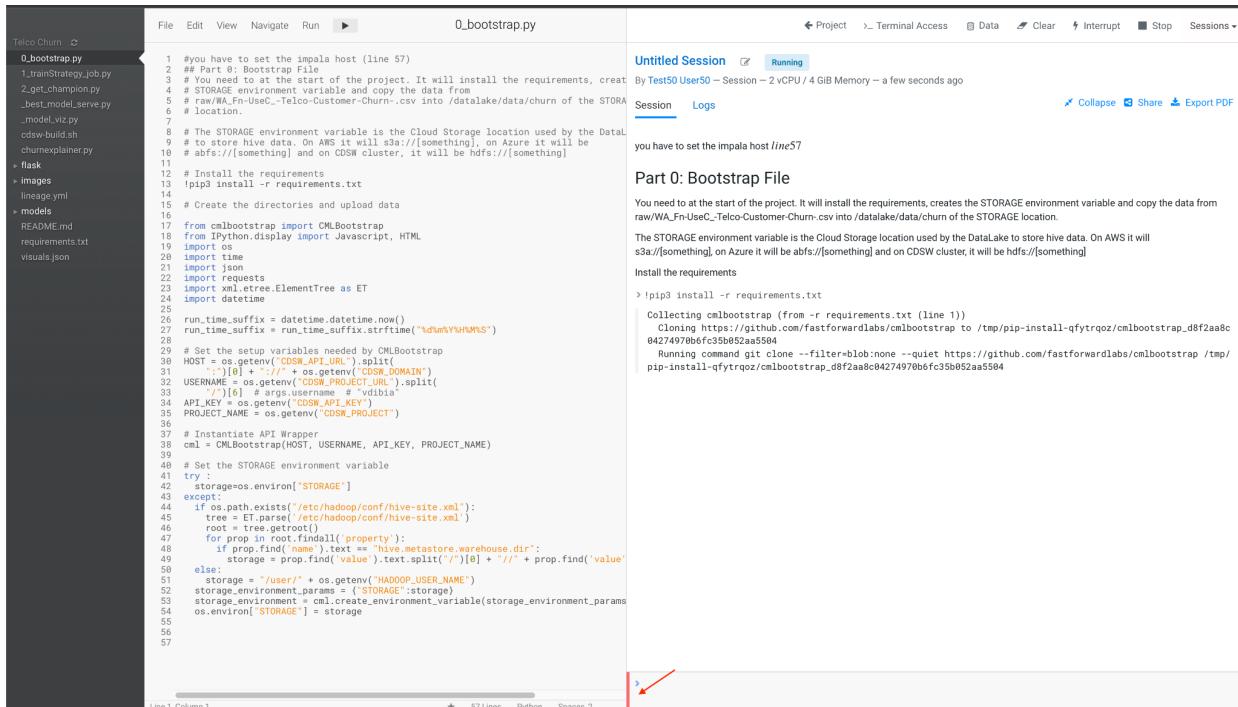
```

File Edit View Navigate Run ▶ 0_bootstrap.py

0_bootstrap.py
1 #you have to set the impala host (line 57)
2 # Part 0: Bootstrap File
3 # You need to be 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 Data
9 # to store hive data. On AWS it will s3a://[something], on Azure it will be
10 # abfs://[something] and on CDSW 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("%d%b%Y%H%M%S")
28
29 # Set the setup variables needed by CMLBootstrap
30 HOST = os.getenv("CDSW_API_URL").split(
31     "/")[-1] + "/" + os.getenv("CDSW_DOMAIN")
32 USERNAME = os.getenv("CDSW_PROJECT_URL").split(
33     "/")[-1] # args.username # 'vdibia'
34 API_KEY = os.getenv("CDSW_API_KEY")
35 PROJECT_NAME = os.getenv("CDSW_PROJECT")
36
37 # Instantiate API Wrapper
38 cml = CMLBootstrap(HOST, USERNAME, API_KEY, PROJECT_NAME)
39
40 # Set the STORAGE environment variable
41 try:
42     storage=os.environ["STORAGE"]
43 except:
44     if os.path.exists('/etc/hadoop/conf/hive-site.xml'):
45         tree = ET.parse('/etc/hadoop/conf/hive-site.xml')
46         root = tree.getroot()
47         for prop in root.findall('property'):
48             if prop.find('name').text == "hive.metastore.warehouse.dir":
49                 storage = prop.find('value').text.split('/')[-1] + "//" + prop.find('value').text
50             else:
51                 storage = "/user/" + os.getenv("HADOOP_USER_NAME")
52     storage_environment_params = {'STORAGE':storage}
53     storage_environment = cml.create_environment_variable(storage_environment_params)
54     os.environ["STORAGE"] = storage
55
56
57

```

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.



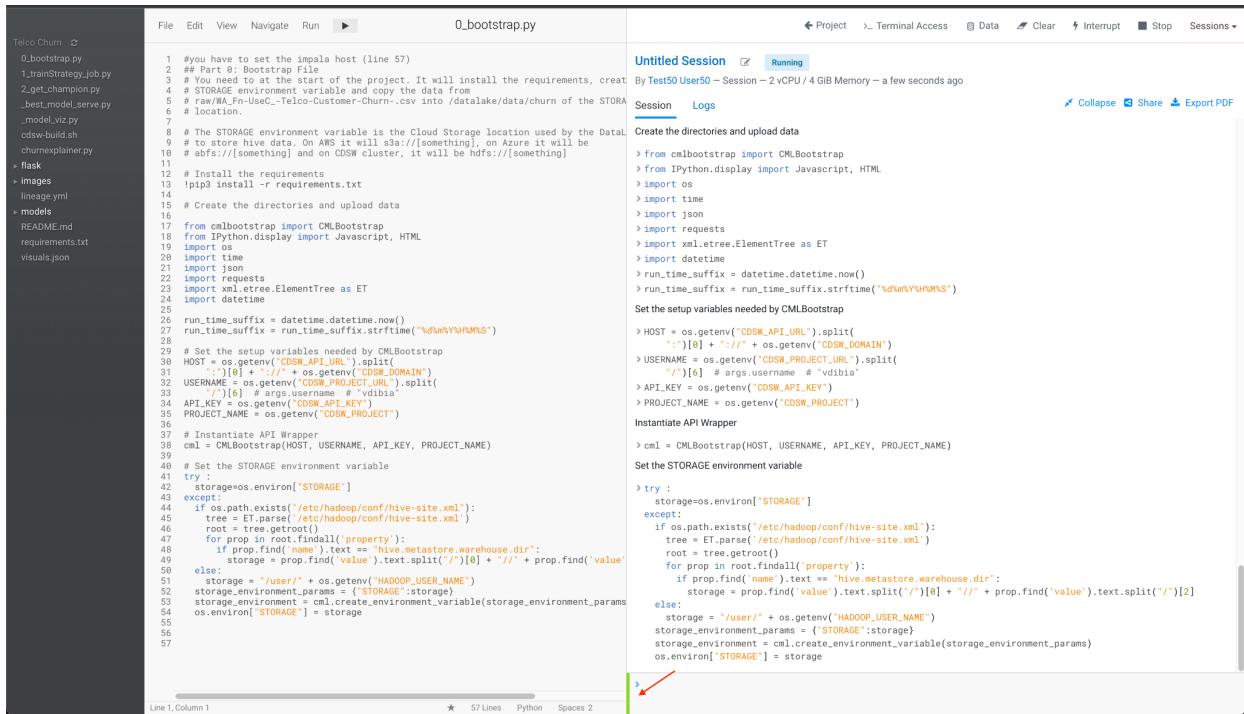
```

File Edit View Navigate Run ▶ 0_bootstrap.py

0_bootstrap.py
1 #you have to set the impala host (line 57)
2 # Part 0: Bootstrap File
3 # You need to be 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 Data
9 # to store hive data. On AWS it will s3a://[something], on Azure it will be
10 # abfs://[something] and on CDSW 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("%d%b%Y%H%M%S")
28
29 # Set the setup variables needed by CMLBootstrap
30 HOST = os.getenv("CDSW_API_URL").split(
31     "/")[-1] + "/" + os.getenv("CDSW_DOMAIN")
32 USERNAME = os.getenv("CDSW_PROJECT_URL").split(
33     "/")[-1] # args.username # 'vdibia'
34 API_KEY = os.getenv("CDSW_API_KEY")
35 PROJECT_NAME = os.getenv("CDSW_PROJECT")
36
37 # Instantiate API Wrapper
38 cml = CMLBootstrap(HOST, USERNAME, API_KEY, PROJECT_NAME)
39
40 # Set the STORAGE environment variable
41 try:
42     storage=os.environ["STORAGE"]
43 except:
44     if os.path.exists('/etc/hadoop/conf/hive-site.xml'):
45         tree = ET.parse('/etc/hadoop/conf/hive-site.xml')
46         root = tree.getroot()
47         for prop in root.findall('property'):
48             if prop.find('name').text == "hive.metastore.warehouse.dir":
49                 storage = prop.find('value').text.split('/')[-1] + "//" + prop.find('value').text
50             else:
51                 storage = "/user/" + os.getenv("HADOOP_USER_NAME")
52     storage_environment_params = {'STORAGE':storage}
53     storage_environment = cml.create_environment_variable(storage_environment_params)
54     os.environ["STORAGE"] = storage
55
56
57

```

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



```

Telco Churn
File Edit View Navigate Run ▶ Project > Terminal Access Data Clear Interrupt Stop Sessions ▾

0_bootstrap.py
1_trainStrategy_job.py
2_getChampion.py
3_best_model_serve.py
4_model_viz.py
cdsw-build.sh
churnexplainer.py
> flask
> images
lineage.yml
> models
README.md
requirements.txt
visuals.json

File Edit View Navigate Run ▶ 0_bootstrap.py

1 # you have to set the impala host (line 57)
2 ## Part 0: Bootstrap File
3 # You need to setup the start of the project. It will install the requirements, create
4 # a CDSW environment variable and copy the data from
5 # raw/MA_Fn-UseC---Telco-Customer-Churn-.csv into /data/datalake/churn of the STORA
6 # location.
7
8 # The STORAGE environment variable is the Cloud Storage location used by the Data
9 # to store hive data. On AWS it will s3://[something], on Azure it will be
10 # abfs://[something] and on CDSW 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 cmldbootstrap 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("%d%b%Y%H%M%S")
28
29 # Set the setup variables needed by CMLBootstrap
30 HOST = os.getenv("CDSW_API_URL").split(
31     "/")[-1].split("api")[-1].split("CDSW_DOMAIN")
32 USERNAME = os.getenv("CDSW_PROJECT_URL").split(
33     "/")[-1] # args.username # "vdibia"
34 API_KEY = os.getenv("CDSW_API_KEY")
35 PROJECT_NAME = os.getenv("CDSW_PROJECT")
36
37 # Instantiates API Wrapper
38 cml = CMLBootstrap(HOST, USERNAME, API_KEY, PROJECT_NAME)
39
40 # Set the STORAGE environment variable
41 try:
42     storage=os.environ["STORAGE"]
43 except:
44     if os.path.exists("/etc/hadoop/conf/hive-site.xml"):
45         tree = ET.parse('/etc/hadoop/conf/hive-site.xml')
46         root = tree.getroot()
47         for prop in root.findall('property'):
48             if prop.find('name').text == "hive.metastore.warehouse.dir":
49                 storage = prop.find('value').text.split("/")[-1] + "//" + prop.find('value').text
50             else:
51                 storage = "/user/" + os.getenv("HADOOP_USER_NAME")
52             storage_environment_params = ('STORAGE':storage)
53             storage_environment = cml.create_environment_variable(storage_environment_params)
54             os.environ["STORAGE"] = storage
55
56
57
Untitled Session ▶ Running
By Test50 User50 - Session - 2 vCPU / 4 GiB Memory - a few seconds ago
Session Logs
Create the directories and upload data
> from cmldbootstrap import CMLBootstrap
> from IPython.display import Javascript, HTML
> import os
> import time
> import json
> import requests
> import xml.etree.ElementTree as ET
> import datetime
> run_time_suffix = datetime.datetime.now()
> run_time_suffix = run_time_suffix.strftime("%d%b%Y%H%M%S")
Set the setup variables needed by CMLBootstrap
> HOST = os.getenv("CDSW_API_URL").split(
...)[-1] + "/" + os.getenv("CDSW_DOMAIN")
> USERNAME = os.getenv("CDSW_PROJECT_URL").split(
...)[-1] # args.username # "vdibia"
> API_KEY = os.getenv("CDSW_API_KEY")
> PROJECT_NAME = os.getenv("CDSW_PROJECT")
Instantiate API Wrapper
> cml = CMLBootstrap(HOST, USERNAME, API_KEY, PROJECT_NAME)
Set the STORAGE environment variable
> try:
...     storage=os.environ["STORAGE"]
... except:
...     if os.path.exists('/etc/hadoop/conf/hive-site.xml'):
...         tree = ET.parse('/etc/hadoop/conf/hive-site.xml')
...         root = tree.getroot()
...         for prop in root.findall('property'):
...             if prop.find('name').text == 'hive.metastore.warehouse.dir':
...                 storage = prop.find('value').text.split('/')[-1] + "//" + prop.find('value').text
...             else:
...                 storage = '/user/' + os.getenv('HADOOP_USER_NAME')
...             storage_environment_params = ('STORAGE':storage)
...             storage_environment = cml.create_environment_variable(storage_environment_params)
...             os.environ['STORAGE'] = storage
Line 1, Column 1
★ 57 Lines Python Spaces 2

```

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.

```

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 from mlflow import sklearn
10 from mlflow import tracking
11 from mlflow import sys
12 #from your_data_loader import load_data
13 from churnexplainer import CategoricalEncoder
14 import datetime
15
16 import time
17 import os
18 import sys
19 from pyspark.sql import SparkSession
20 from pyspark.sql.types import *
21 import xml.etree.ElementTree as ET
22 from mlflow.tracking import MlflowClient
23 # Set the setup variables needed by CMLBootstrap
24 HOST = os.getenv("CDSW_API_URL").split(
25     ":" )[0] + ":" + os.getenv("CDSW_DOMAIN")
26 USERNAME = os.getenv("CDSW_USERNAME") + ":" + os.getenv("CDSW_PROJECT_URL").split(
27     "/" )[6] # args.username # "vdubla"
28 API_KEY = os.getenv("CDSW_API_KEY")
29 PROJECT_NAME = os.getenv("CDSW_PROJECT")
30
31 # Instantiates API Wrapper
32 cml = CMLBootstrap(HOST, USERNAME, API_KEY, PROJECT_NAME)
33
34 uservariables=cml.get_user()
35 if uservariables['username'][~-3:] == '0':
36     DATABASE = "user"+uservariables['username'][~-3:]
37 else:
38     DATABASE = uservariables['username']
39     DATABASE = 'acapmos'
40
41
42 runtimescml.get_runtimes()
43 runtimes=runtimes['runtimes']
44 runtimesdf = pd.DataFrame.from_dict(runtimes, orient='columns')
45 runtimesdfid=runtimesdf.loc[(runtimesdf['editor'] == 'Workbench') & (runtimesdf['ke
46 id_rf']==runtimesdf['values'][0])
47
48 spark = SparkSession(
49     .builder
50     .appName("PythonSQL")
51     .master("local[*]")
52     .config("spark.sql.extensions", "org.apache.iceberg.spark.extensions.IcebergS
53     config("spark.sql.catalog.spark_catalog", "org.apache.iceberg.spark.SparkCatalog
54     .config("spark.sql.catalog.local", "org.apache.iceberg.spark.SparkCatalog") \
55     .config("spark.sql.catalog.local.type", "hadoop") \
56     .config("spark.sql.catalog.spark_catalog.type", "hive") \
57     .getOrCreate()
58
59
60 # **Note:**
61 # Our file isn't big, so running it in Spark local mode is fine but you can add
62

```

Line 1, Column 1 ★ 433 Lines Python Spaces 2

Untitled Session Running
By Test10 User10 — Session — 2 vCPU / 4 GiB Memory – a few seconds ago

Session Logs Spark UI

```

mlflow.delete_experiment(experimentId)

time.sleep(20)
except:
    print('First time execution')

mlflow.set_experiment('expRetrain')
valuesParam=[9,11,15]
for i in range(len(valuesParam)):
    with mlflow.start_run(run_name="run_."+str(i)) as run:
        # tracking run parameters
        mlflow.log.param("compute", 'local')
        mlflow.log.param("dataset", 'telco-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()

```

2023/07/12 18:36:19 INFO mlflow.tracking.fluent: Experiment with name 'expRetrain' does not exist. Creating a new experiment.

First time execution

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.

Name	Creator	Created At	Last Updated
expRetrain	Test10 User10	07/12/2023 8:36 PM	

Displaying 1 - 1 of 1 < 1 > 25 / page

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.

user010 / Telco Churn / Experiments / expRetrain

Experiment BETA

Experiment Name: expRetrain
Experiment ID: pdj8-a6kp-bh2r-dehf
Artifact Location: /home/cdsu/experiments/pdj8-a6kp-bh2r-dehf

Runs (3)

	Status	Start Time	Run Name	Duration	User	Source	Version	Models	Parameters >	Metrics
<input type="checkbox"/>	✓	2023-07-12 08:36:19	run_3619_0	5.2s	user010	ipython3	8e811a	sklearn	algo: random forest, compute: local, dataset: telco-churn, precision: 1	
<input type="checkbox"/>	✓	2023-07-12 08:36:25	run_3619_1	3.8s	user010	ipython3	8e811a	sklearn	algo: random forest, compute: local, dataset: telco-churn, precision: 1	
<input type="checkbox"/>	✓	2023-07-12 08:36:28	run_3619_2	4.0s	user010	ipython3	8e811a	sklearn	algo: random forest, compute: local, dataset: telco-churn, precision: 1	

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*.

user010 / Telco Churn Sessions

Sessions

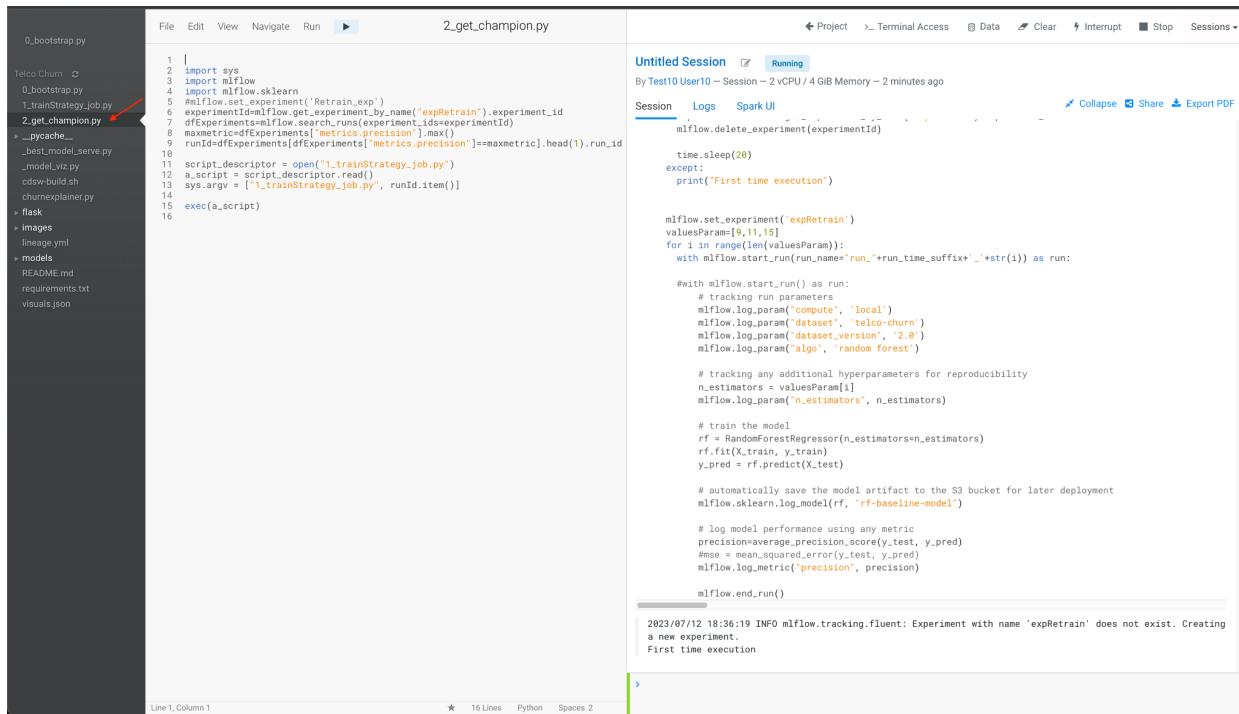
Creator: All | Show Running Only:

Status	Session	Kernel	Creator	Created At	Duration	Actions
Running	Untitled Session	(Python 3.7 Workbench Standard)	Test10 User10	07/12/2023 8:35 PM	Running since 1m 43s	<input type="button" value="Edit"/> <input type="button" value="Stop"/> <input type="button" value="Delete"/>

Displaying 1 - 1 < 1 > 25 / page

Workspace: ssa-cml-workspace
Cloud Provider: AWS (AWS)

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 the 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 Jupyter Notebook interface. On the left, a sidebar lists files: 0_bootstrap.py, Telco Churn, 0_bootstrap.py, 1_trainStrategy.job.py, 2_get_champion.py (with a red arrow pointing to it), __pycache__, __best_model_server.py, _model_viz.py, cds-w-build.sh, churnexplainer.py, flask, images, lineage.yml, models, README.md, requirements.txt, and visuals.json. The main area shows the code for '2_get_champion.py'. The code imports sys, mlflow, and sklearn, then retrieves the experiment ID for 'expRetrain', finds the run ID with the highest precision, and executes the corresponding script. It then sets up a new experiment named 'expRetrain', logs parameters, trains a Random Forest Regressor, and saves the model. The session output shows the command being run and a message indicating a new experiment is being created. The bottom status bar shows 'Line 1, Column 1', '16 Lines', 'Python', and 'Spaces 2'.

```

File Edit View Navigate Run ▶ 2_get_champion.py
File Edit View Navigate Run ▶ 2_get_champion.py
Untitled Session [Running]
By Test10 User10 - Session - 2 vCPU / 4 GiB Memory - 2 minutes ago
Session Logs Spark UI
  mlflow.delete_experiment(experimentId)
  time.sleep(20)
except:
  print("First time execution")

mlflow.set_experiment('expRetrain')
valuesParam=[9,11,15]
for i in range(len(valuesParam)):
  with mlflow.start_run(run_name="run_."+str(i)) as run:
    #with mlflow.start_run() as run:
      # tracking run parameters
      mlflow.log.param("compute", 'local')
      mlflow.log.param("dataset", 'telco-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()

2023/07/12 18:36:19 INFO mlflow.tracking.fluent: Experiment with name 'expRetrain' does not exist. Creating a new experiment.
First time execution

```

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

```

File Edit View Navigate Run ▶ 2_get_champion.py
Untitled Session [Running]
By Test10 User10 - Session - 2 vCPU / 4 GiB Memory - 2 minutes ago
Session Logs Spark UI ⌂ First Execution
Creating Model
Creating new model
New model created with access key msqqkhgmf0lyt4ulz8lkbv7mbdph9gi
Deploying Model.....
Model is deployed
Creating new model for visualization
New model created with access key ml17u8em8ypcxly6xidc4a8g17q3foi
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.

```

File Edit View Navigate Run ▶ 2_get_champion.py
Untitled Session [Running]
By Test10 User10 - Session - 2 vCPU / 4 GiB Memory - 2 minutes ago
Session Logs Spark UI ⌂ First Execution
Creating Model
Creating new model
New model created with access key msqqkhgmf0lyt4ulz8lkbv7mbdph9gi
Deploying Model.....
Model is deployed
Creating Model for visualization
New model created with access key ml17u8em8ypcxly6xidc4a8g17q3foi
Deploying Model.....
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**.

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	<button>Stop</button>
ModelOpsChurn_user010	best..	Deployed	1 / 1	1	2.00 GiB	Jul 12, 2023, 08:39 PM	<button>Stop</button>

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

Model Details	
Source	Code
Model Id	19
Model CRN	cm:cdp:ml:us-west:1:508fd88f:8076-498a-acfb-6f8765cd3d5e8/workspace:1#4b728:bcf#4867-8a54-f8309c99355/fa9eb299-1c63-452b-60d9-70aefba5
Deployment Id	14
Deployment CRN	cm:cdp:ml:us-west:1:508fd88f:8076-498a-acfb-6f8765cd3d5e8/workspace:1#4b728:bcf#4867-8a54-f8309c99355/32aa371-af8b-4225-b8d4-4ca5c60f3109
Build Id	14
Build CRN	cm:cdp:ml:us-west:1:508fd88f:8076-498a-acfb-6f8765cd3d5e8/workspace:1#4b728:bcf#4867-8a54-f8309c99355/197c0d8-b00e-4354-b63a-746af64e795e8
Deployed By	user010
Comment	Initial revision.
Runtime Image	Python 3.7 (Standard)
File	_best_model_.serve.py
Function	explain
Model Resources	
Replicas	1
Total CPU	1 vCPUs
Total Memory	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 shows the Cloudera Machine Learning interface for a project named 'user010'. The left sidebar has a 'Models' section selected. The main area displays the 'Overview' of a model named 'ModelOpsChurn_user010'. In the 'Test Model' section, the 'Input' field contains the following JSON:

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

The 'Test' button is highlighted with a red box. Below it, the 'Result' section shows a status of 'success' with a green circle icon. The 'Response' field displays the JSON output from the model:

```
{
  "model_deployment_crn": "crn:cdp:ml:us-west-1:508fd88f-8076-498a-acfb-6f8765cd35e8:workspace:1e48b728-bcff-4067-8a54-f83099c99355/32aa37d1-af8b-4225-a86d-4ca5",
  "prediction": {
    "probability": 0.5555555555555556
  },
  "uuid": "95a97cf3-36d3-459e-9372-b2b51334ca63"
}
```

The 'Replica ID' field shows 'modelopschurn-user010-19-14-6c5d7947ff-52kzg'. On the right side, there are sections for 'Comment', 'Runtime Image', 'File', and 'Function', and a 'Model Resources' section showing 1 replica, 1 vCPU, and 2.00 GiB memory.