

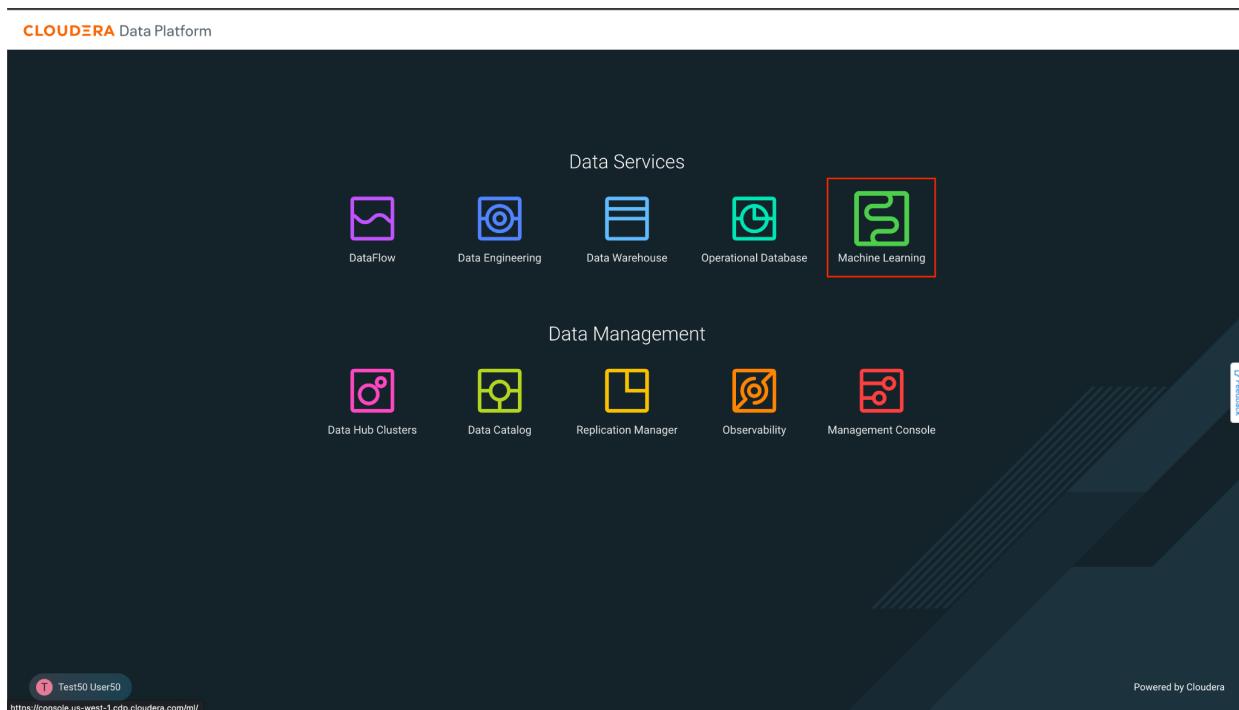
Data Lifecycle en CDP Public Cloud

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 very bottom, it shows '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/camposalex/TelcoChurn>

The screenshot shows the 'New Project' interface. The 'Project Name' field contains 'Telco Churn'. The 'Project Visibility' section shows 'Private - Only added collaborators can view the project' selected. Under 'Initial Setup', the 'Git' tab is active. In the 'Provide the Git URL of the project to clone...' section, 'HTTPS' is selected and the URL 'https://github.com/camposalex/TelcoChurn' is entered.

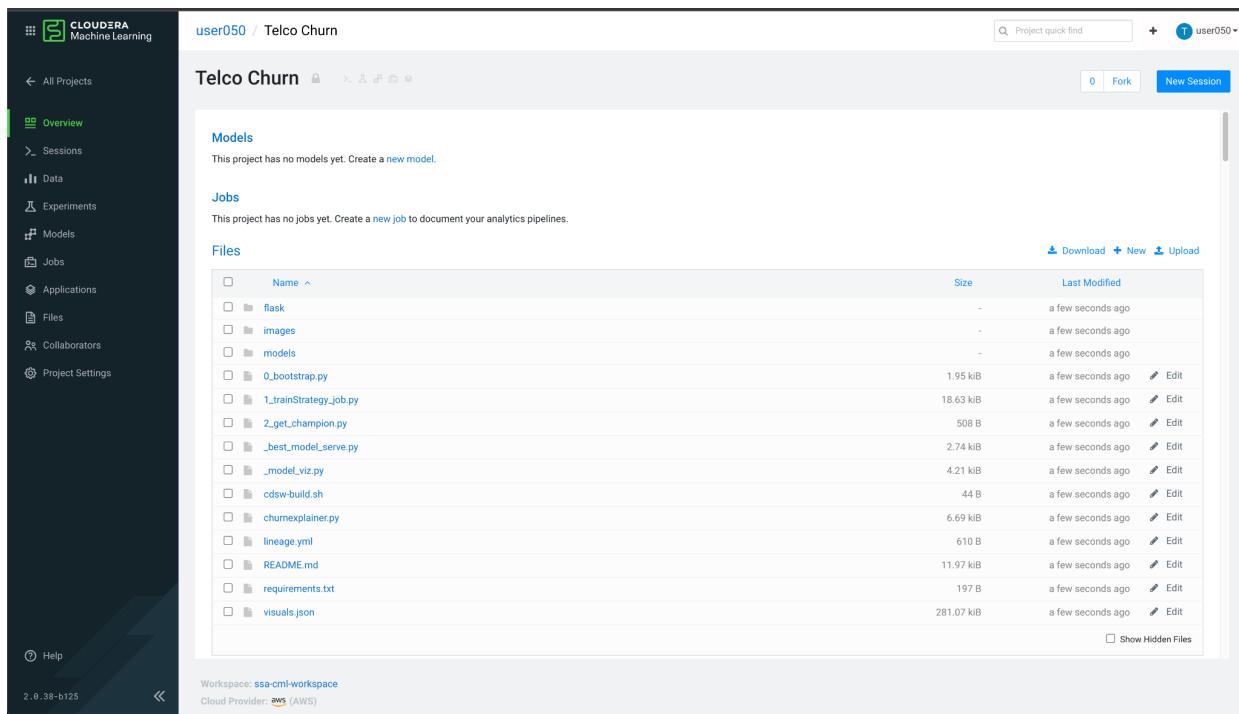
Make sure to select **Python 3.7** in the Kernel selector. Click the button **Create Project**

The screenshot shows the 'Runtime setup' dialog. The 'Kernel' dropdown is set to 'Python 3.7'. A red arrow points to this dropdown. Below it is a checkbox for 'Add GPU enabled Runtime variant'. A list of available runtimes is shown: JupyterLab - Python 3.7 - Standard - 2023.05, PBJ Workbench - Python 3.7 - Standard - 2023.05, and Workbench - Python 3.7 - Standard - 2023.05. At the bottom right are 'Cancel' and 'Create Project' buttons.

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.

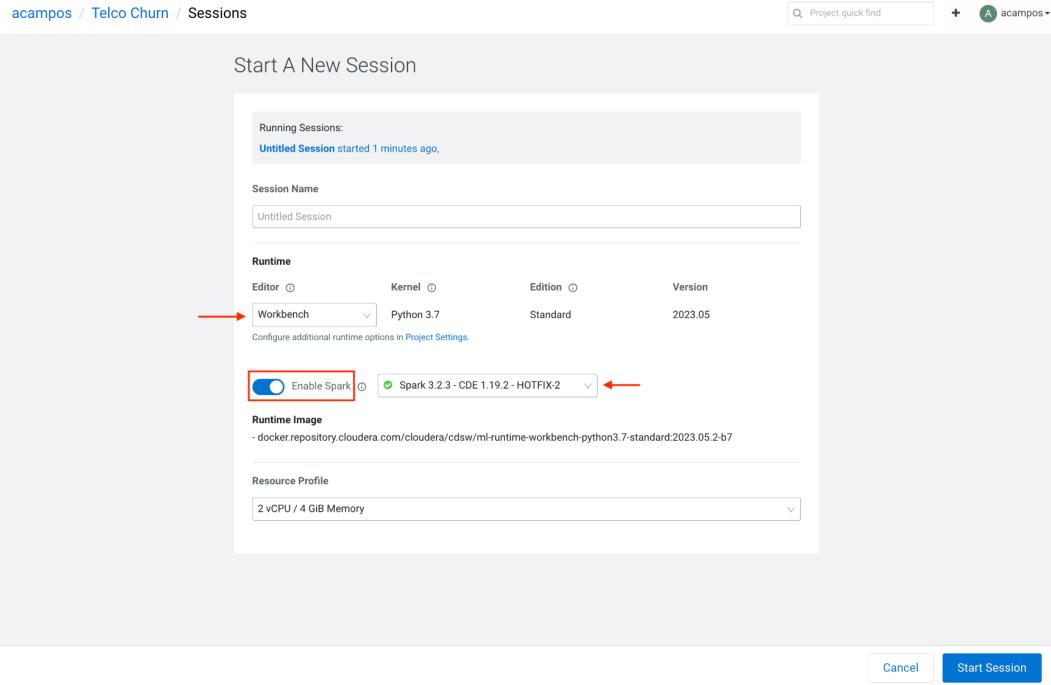


The screenshot shows the Cloudera Machine Learning interface. On the left is a sidebar with navigation links: All Projects, Overview (which is selected), Sessions, Data, Experiments, Models (selected), Jobs, Applications, Files, Collaborators, and Project Settings. At the bottom of the sidebar are Help and version information (2.0.38-b125). The main content area is titled "user050 / Telco Churn". It displays sections for "Models" (no models yet), "Jobs" (no jobs yet), and "Files". The "Files" section lists several Python scripts and files, including 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. Each file entry includes its size, last modified date, and an "Edit" link. A "Download" button is at the top of the file list, along with "New" and "Upload" buttons. At the bottom of the file list is a "Show Hidden Files" checkbox. The footer of the interface shows the workspace (ssa-cml-workspace) and cloud provider (aws).

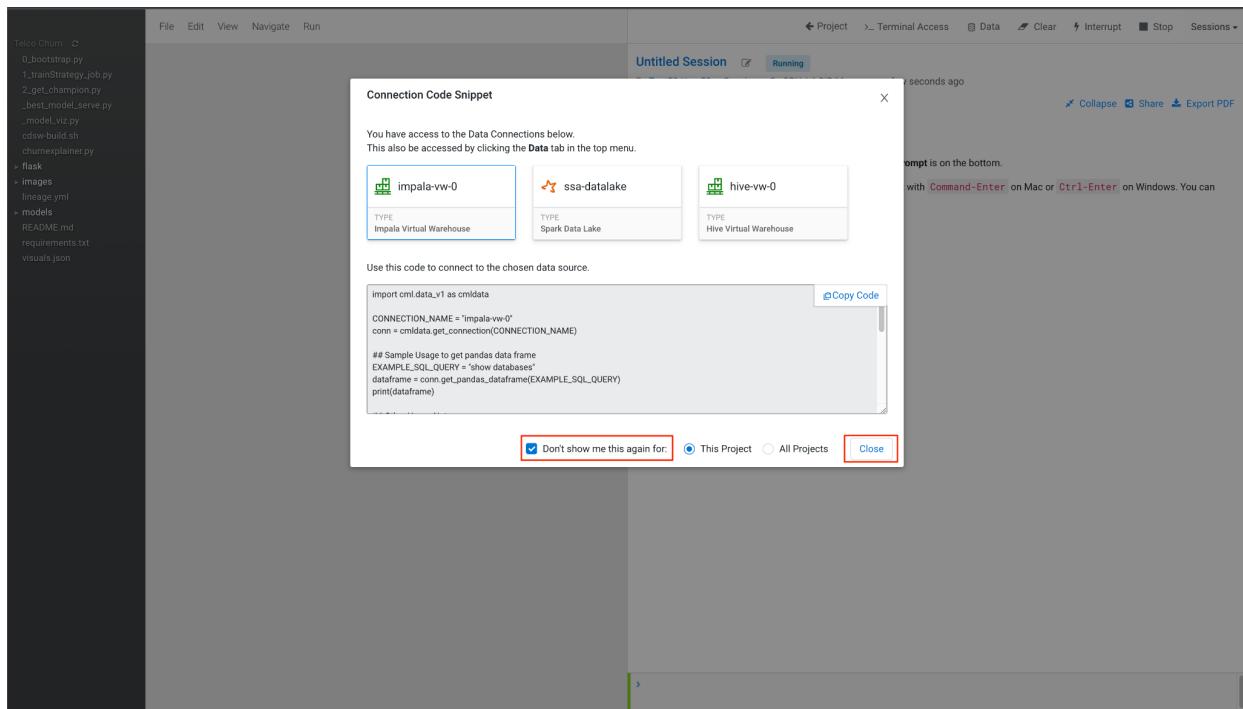
6. When starting a new session, make sure:

Select **Workbench** in the Editor selector.
Enable **Spark**, marking the corresponding check.
Select **Spark 3.2.x**, in the Spark version selector.

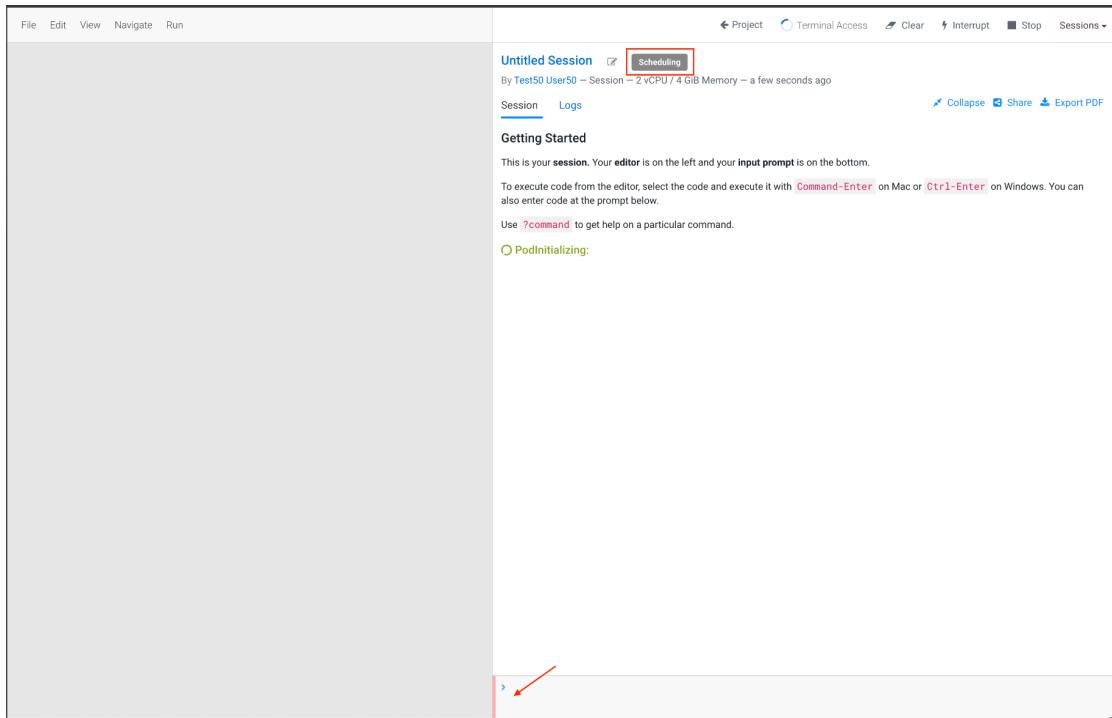
Click on the button **Start Session**



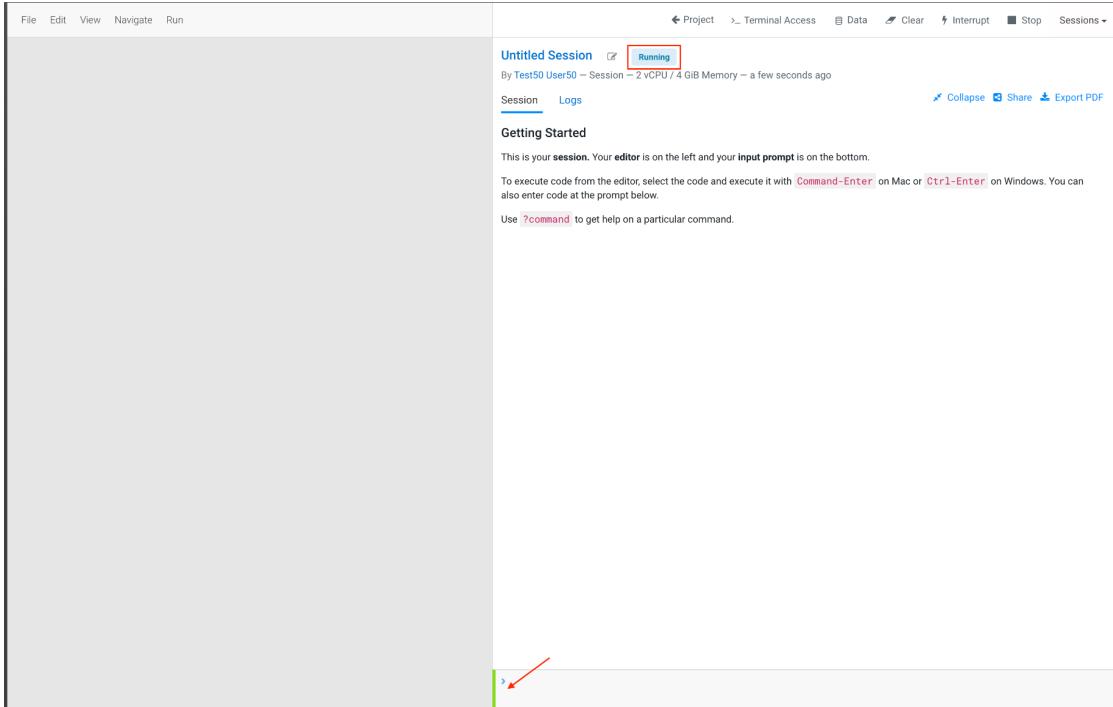
7. When you start a session for the first time, it will ask if you want to use a data connection. This project does not need this type of connection. mark the check of **Don't show me this again**, and then click the button **Close**, so this window will not appear anymore.



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.

```

File Edit View Navigate Run  0_bootstrap.py
Telco Churn
0_bootstrap.py
1_trainStrategy.job.py
2_get_champion.py
3_best_model_serve.py
4_mlflow_viz.py
cdsw-build.sh
churn_explainer.py
> flask
> images
> lineage.yml
> models
README.md
requirements.txt
visuals.json

0_bootstrap.py

1 # You have to set the impala host (line 57)
2 ## Part 0: Bootstrap File
3 # You need to set at the start of the project. It will install the requirements, create
4 # STORAGE environment variable and copy the data from
5 # raw/NA_Fn-Use-C-Telco-Customer-Churn-.csv into /datalake/data/churn of the STORA
6 #
7 # The STORAGE environment variable is the Cloud Storage location used by the Data
8 # to store hive data. On AWS it will s3://[something], on Azure it will be
9 # abfs://[something] and on CDSW cluster, it will be hdfs://[something]
10
11 # Install the requirements
12 pip3 install -r requirements.txt
13
14 # Create the directories and upload data
15
16 from cmlbootstrap import CMLBootstrap
17 from IPython.display import Javascript, HTML
18 import os
19 import time
20 import json
21 import requests
22 import xml.etree.ElementTree as ET
23 import datetime
24
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     ":" )[0] + "/" + os.getenv("CDSW_DOMAIN")
32 USERNAME = os.getenv("CDSW_PROJECT_URL").split(
33     "/" )[1] + os.getenv("CDSW_PROJECT_NAME") + "liba"
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 == 'metastore.warehouse.dir':
49                 storage=prop.find('value').text.split('/')[0] + "//" + prop.find('value').text.split('/')[1]
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

```

Line 1, Column 1 ★ 57 Lines Python Spaces 2

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
Telco Churn
0_bootstrap.py
1_trainStrategy.job.py
2_get_champion.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

0_bootstrap.py
1 # you have to set the impala host (line 57)
2 ## Part 0: Bootstrap File
3 # You need to at the start of the project. It will install the requirements, creat
4 # STORAG environment variable and copy the data from
5 # raw/MA_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 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     ":" )[0] + ":" + os.getenv("CDSW_DOMAIN")
32 USERNAME = os.getenv("CDSW_PROJECT_URL").split(
33     "/" )[6] # 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('/')[0] + "//" + prop.find('value')
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
Line 1, Column 1  ★ 57 Lines  Python  Spaces 2

```

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

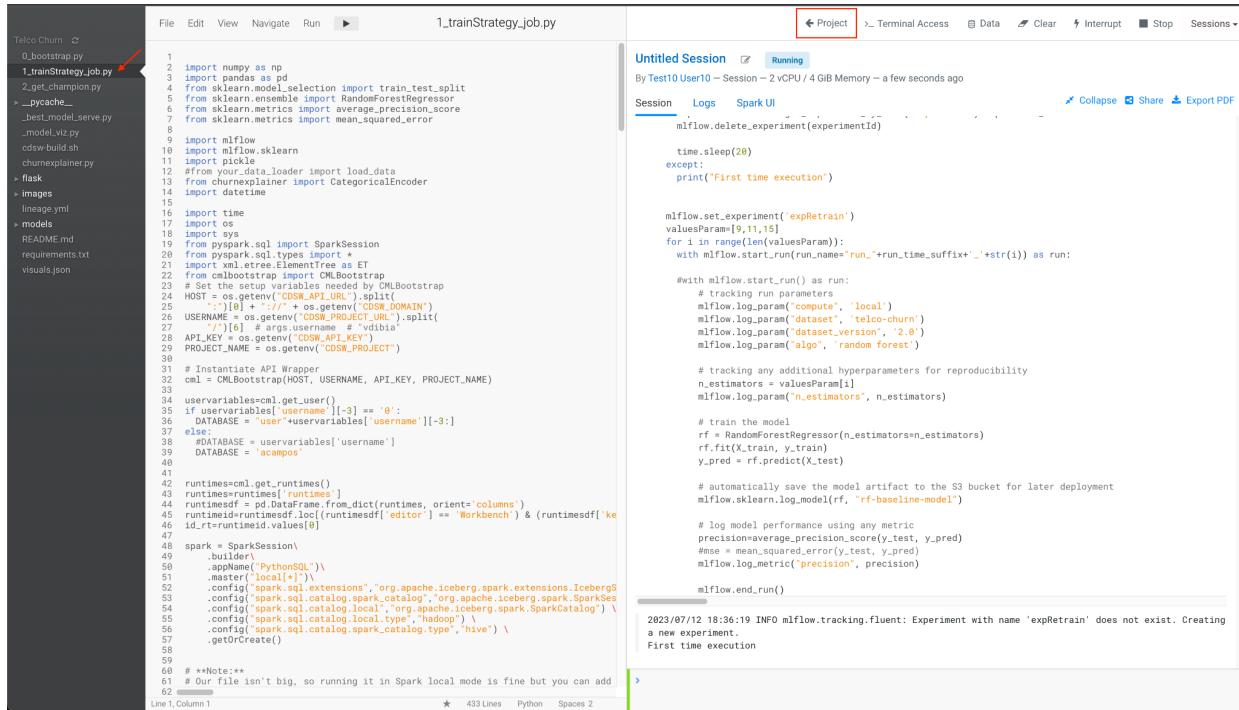
```

File Edit View Navigate Run ▶ 0_bootstrap.py
Telco Churn
0_bootstrap.py
1_trainStrategy.job.py
2_get_champion.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

0_bootstrap.py
1 # you have to set the impala host (line 57)
2 ## Part 0: Bootstrap File
3 # You need to at the start of the project. It will install the requirements, creat
4 # STORAG environment variable and copy the data from
5 # raw/MA_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 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     ":" )[0] + ":" + os.getenv("CDSW_DOMAIN")
32 USERNAME = os.getenv("CDSW_PROJECT_URL").split(
33     "/" )[6] # 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('/')[0] + "//" + prop.find('value')
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
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.



The screenshot shows the Jupyter Notebook interface. On the left, a sidebar lists files in the 'Telco Churn' directory, with '1_trainStrategy_job.py' highlighted by a red arrow. The main area displays the code for '1_trainStrategy_job.py'. At the top right, there are buttons for 'Project', 'Terminal Access', 'Data', 'Clear', 'Interrupt', 'Stop', and 'Sessions'. The 'Project' button is highlighted with a red border. Below these buttons, the status bar shows 'Untitled Session' and 'Running'. The code itself is a Python script that imports various libraries like numpy, pandas, sklearn, and mlflow. It sets up an experiment named 'expRetrain' with three runs, each with different hyperparameters. It then trains a Random Forest Regressor and saves the model. The log output at the bottom shows the creation of a new experiment and a 'First time execution' message.

```

File Edit View Navigate Run > Project Terminal Access Data Clear Interrupt Stop Sessions
Telco Churn
0_bootstrap.py
1_trainStrategy_job.py
2_get_champion.py
> _pycache_/_best_model_serve.py
_model_viz.py
cdsw-build.sh
churnexplainer.py
flask
images
lineage.yml
models
README.md
requirements.txt
visuals.json

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 import mlflow.sklearn
10 import pickle
11 #From your_data_loader import load_data
12 #From your_churn_explainer import CategoricalEncoder
13 import datetime
14
15 import time
16 import os
17 import sys
18 import pyspark.sql import SparkSession
19 from pyspark.ml import Pipeline
20 from pyspark.ml.type import *
21 from xml.etree.ElementTree as ET
22 from cmlbootstrap import CMLBootstrap
23 # Set the setup variables needed by CMLBootstrap
24 HOST = os.getenv('CDSW_HOSTNAME').split('.')[0]
25 PROJECT_NAME = os.getenv('CDSW_PROJECT_NAME')
26 DATABASE = os.getenv('CDSW_PROJECT_URL').split(
27    '/')[-1] # args.username # 'vdibia'
28 API_KEY = os.getenv("CDSW_API_KEY")
29 PROJECT_NAME = os.getenv('CDSW_PROJECT_NAME')
30
31 # Instantiate API Wrapper
32 #cal = CMLBootstrap(HOST, USERNAME, API_KEY, PROJECT_NAME)
33 uservariables=cal.get_user()
34 uservariables['username'][1:-3]
35 if uservariables['username'][1:-3] == '0':
36   DATABASE = uservariables['username'][1:-3]
37 else:
38   #DATABASE = uservariables['username']
39   DATABASE = 'acmpos'
40
41 runtimes=cal.get_runtimes()
42 runtimes=runtimes['runtimes']
43 runtimesdf = pd.DataFrame.from_dict(runtimes, orient='columns')
44 runtimeid=runtimesdf.loc[(runtimesdf['editor'] == 'Workbench') & (runtimesdf['ke
45 id']==runtimeid.values[0])
46
47 spark = SparkSession\
48 .builder\
49 .appName("PyIceberg")\
50 .master("local[*]")
51 .config("spark.sql.extensions", "org.apache.iceberg.spark.extensions.IcebergS
52 .config("spark.sql.catalog.spark_catalog", "org.apache.iceberg.spark.SparkSes
53 .config("spark.sql.catalog.local", "org.apache.iceberg.spark.SparkCatalog") \
54 .config("spark.sql.catalog.local.type", "hadoop") \
55 .config("spark.sql.catalog.spark_catalog.type", "hive") \
56 .getOrCreate()
57
58
59
60 # **Note:**#
61 # Our file isn't big, so running it in Spark local mode is fine but you can add
62

```

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:
        #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)
            #mean_squared_error(y_test, y_pred)
            mlflow.log_metric("precision", precision)

mlflow.end_run()

2023/07/12 10:36:19 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.

The screenshot shows the Cloudera Machine Learning interface. On the left, a dark sidebar menu lists various project components: All Projects, Overview, Sessions, Data, Experiments (which is highlighted with a green arrow), Models, Jobs, Applications, Files, Collaborators, and Project Settings. The main content area is titled "user010 / Telco Churn Experiments". It features a search bar labeled "Search Experiments" and a table with columns: Name, Creator, Created At, and Last Updated. A single experiment named "expRetrain" is listed, created by "Test10 User10" on "07/12/2023 8:36 PM". At the bottom right of the table, it says "Displaying 1 - 1 of 1".

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.

The screenshot shows the "Experiment" details page for "expRetrain". The sidebar on the left is identical to the previous screenshot. The main content area shows the experiment's name, ID, and artifact location. Below this, under "Runs (3)", there is a table of three runs. The table has columns: Status, Start Time, Run Name, Duration, User, Source, Version, Models, algo, compute, dataset, and precision. The runs are:

Status	Start Time	Run Name	Duration	User	Source	Version	Models	algo	compute	dataset	precision
Success	2023-07-12 08:36:19	run_3619_0	5.2s	user010	ipython3	8e811a	sklearn	random forest	local	telco-churn	1
Success	2023-07-12 08:36:25	run_3619_1	3.8s	user010	ipython3	8e811a	sklearn	random forest	local	telco-churn	1
Success	2023-07-12 08:36:28	run_3619_2	4.0s	user010	ipython3	8e811a	sklearn	random forest	local	telco-churn	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*.

The screenshot shows the Cloudera Machine Learning interface. On the left, there's a dark sidebar with various project management and data science tools like Overview, Sessions (which is highlighted with a red arrow), Data, Experiments, Models, Jobs, Applications, Files, Collaborators, and Project Settings. Below the sidebar, there's a help section and a footer indicating the version is 2.0.38-b125. The main area is titled "user010 / Telco Churn Sessions". It displays a table of sessions with the following columns: Status, Session, Kernel, Creator, Created At, and Duration. A single row is shown, labeled "Running" and "Untitled Session". The "Untitled Session" cell is highlighted with a red box. The table also includes filters for Creator (All), Status (Running), Session, Kernel, Creator, and Duration, along with buttons for Stop Selected, Delete Selected, and New Session. At the bottom, it says "Displaying 1 - 1" and "25 / page". The footer also shows the workspace is "ssa-cml-workspace" and the cloud provider is "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.

File Edit View Navigate Run ▶ 2_get_champion.py

Untitled Session Running

By Test010 User10 – Session – 2 vCPU / 4 GiB Memory – 2 minutes ago

Session Logs Spark UI

Collage Share Export PDF

```
1 | import sys
2 | import mlflow
3 | import mlflow.sklearn
4 | import os
5 | #mlflow.set_experiment('Retrain_exp')
6 | experimentId=mlflow.get_experiment_by_name("expRetrain").experiment_id
7 | dfExperiments=mlflow.search_runs(experiment_ids=experimentId)
8 | dfExperiments=dfExperiments.sort_values(['metrics.precision'], ascending=[True])
9 | runId=dfExperiments[dfExperiments['metrics.precision']==maxMetric].head(1).run_id
10 |
11 | script_descriptor = open('l_trainstrategy.job.py')
12 | a_script = script_descriptor.read()
13 | sys.argv = ['l_trainstrategy.job.py', runId.item()]
14 |
15 | exec(a_script)
16 
```

```
mlflow.set_experiment('expRetrain')
valuesParam=[0,1,15]
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", '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)

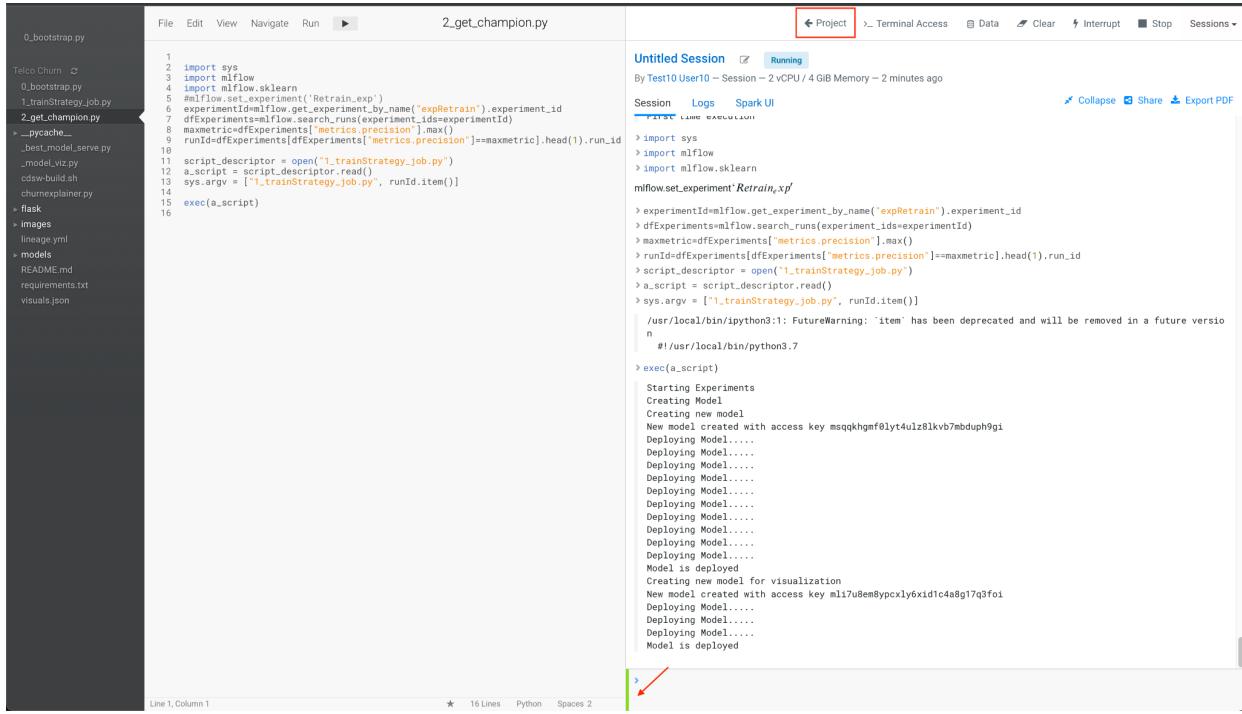
    mlflow.end_run()
```

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.

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.



The screenshot shows a Jupyter Notebook environment. On the left, a sidebar displays a file tree with files like `0_bootstrap.py`, `Telco Churn`, `1_trainStrategy.job.py`, `2_get_champion.py`, and `3_get_champion.py`. The main area shows a code cell for `2_get_champion.py` and its output in the terminal session. A red arrow points to the bottom of the terminal window, highlighting the final deployment log message.

```

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
▶ PASTE CODE EXECUTION
> import sys
> import mlflow
> import mlflow.sklearn
> mlflow.set_experiment('Retrain_exp')
> experimentId=mlflow.get_experiment_by_name("expRetrain").experiment_id
> dfExperiments=mlflow.search_runs(experiment_ids=experimentId)
> maxmetric=dfExperiments[["metrics.precision"]].max()
> runId=dfExperiments[dfExperiments["metrics.precision"]==maxmetric].head(1).run_id
> 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()]
> exec(a_script)
> 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
n
#!/usr/local/bin/python3.7

> exec(a_script)

Starting Experiments
Creating Model
Creating Model
New model created with access key msqqkhgmf0lyt4ulz8ikv7mbdph9gi
Deploying Model....
Model is deployed
Creating new model for visualization
New model created with access key ml17u8em8ypcxly6xid1c4a8g17q3foi
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**.

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

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 includes links for All Projects, Overview, Sessions, Data, Experiments, Models (which is selected and highlighted in green), Jobs, Applications, Files, Collaborators, Project Settings, Help, and a footer note about version 2.0.38-b125. The main content area displays the 'Overview' for 'ModelOpsChurn_user010'. It shows a 'Sample Response' with an empty JSON object {}, a 'Test Model' section with an 'Input' JSON object containing fields like 'onlinesecurity', 'multiplelines', 'internetservice', 'seniorcitizen', and 'techsupport', and a 'Test' button (which is red). Below this is a 'Result' section with a 'Status' of 'success' and a 'Response' JSON object containing a 'model_deployment_crn', a 'prediction' with a probability of 0.5555555555555556, and a 'uuid'. A 'Replica ID' field shows 'modelopschurn-user010-19-14-6c5d7947ff-52kzg'. The top right features a 'Project quick find' bar and a user profile for 'user010'. On the far right, there's a vertical scroll bar.