CLOUDERA

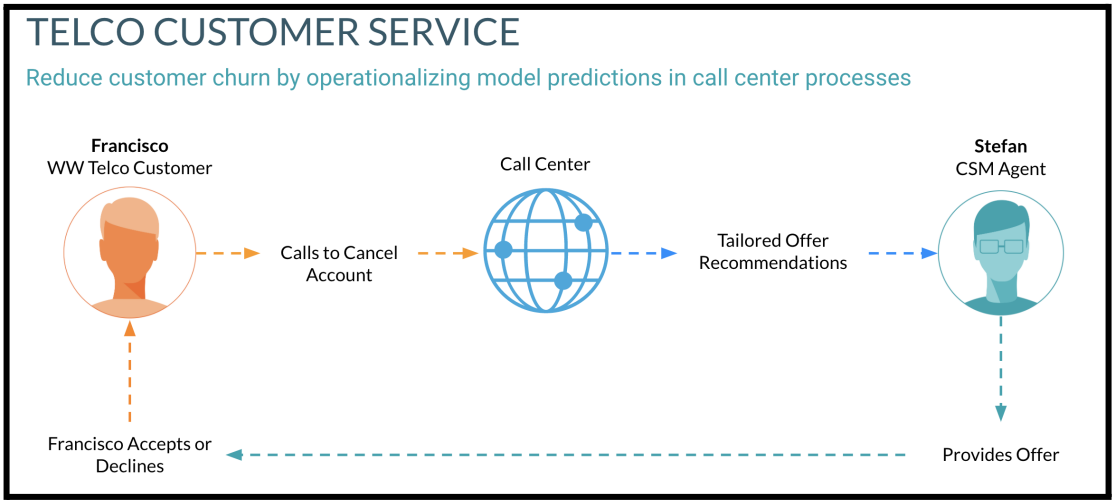
MACHINE LEARNING HANDS-ON

STUDENT GUIDE

Table of Contents

Introduction

In this workshop, we will introduce you to the World Wide Telco organization. They have many ML use cases, but we'll focus on a specific one for today. In this use case they are trying to reduce customer churn (i.e. cancellations) and most of the cancellations are coming in from their call center. In these hands-on labs, a customer, "Francisco" will call in with the intent to cancel. The call center application will - based on the predicted reasoning for Francisco to cancel - provide recommendations to Stefan who is the customer service manager for WW Telco. He'll then provide the offer to Francisco will choose or not to choose to accept - i.e. not cancel his account.



The Hands on Labs will take you through the whole process of logging in, sourcing the code and building fully working applications with deployed models.

Lab summary

1. Data Ingest.
2. Data Exploration.
3. Model Build Experiment.
4. Model Deploy.
5. Application Deploy

Lab 0 - Initial setup

Pre-requisites

1. Laptop with a supported OS (Windows 7 not supported) or Macbook.
2. A modern browser - Google Chrome (IE, Firefox, Safari not supported).
3. Firewall that prevents access to GitHub should not be present.

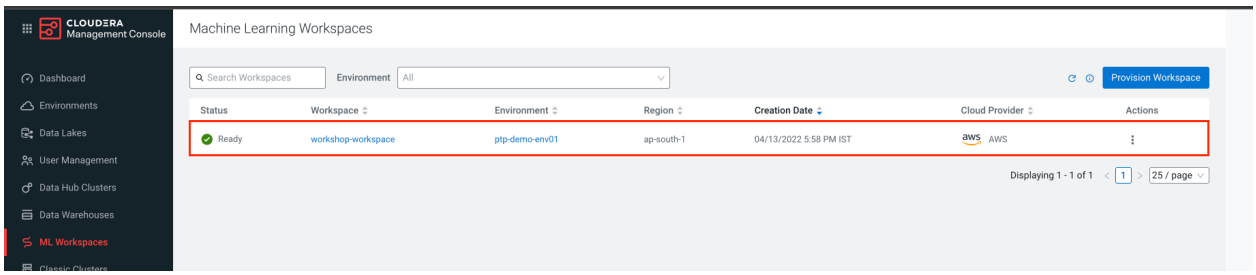
Getting Connected

You will be provided with a set of credentials along with a URL which will give you access to the CDP environment. The details will be shared by the instructor either before the lab starts or during the course of the labs

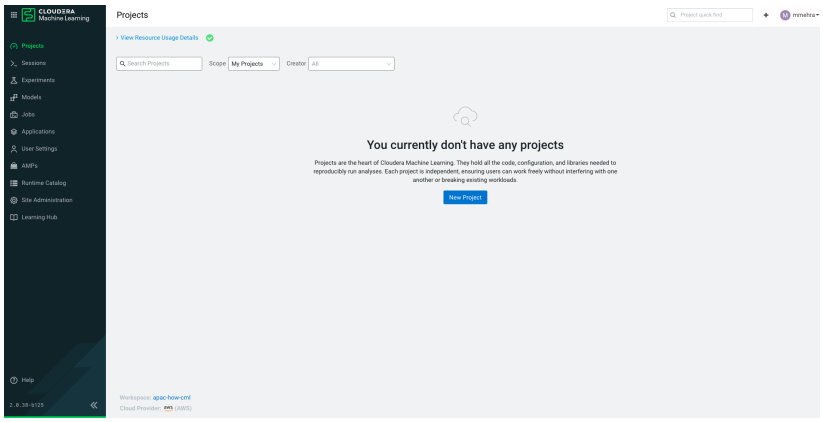
`Environment URL`	Will be shared by the instructor
`Login Email`	Will be shared by the instructor
`Workload User Name`	Will be shared by the instructor
`Workload Password`	Will be shared by the instructor
`GitHub URL`	Will be shared by the instructor

Create Project

Login into CDP using the URL above and the credentials assigned to you. After logging into CDP you will have access to the main console and then go to Cloudera Machine Learning workspace. Click on Workspace to proceed to executing the hands-on labs.



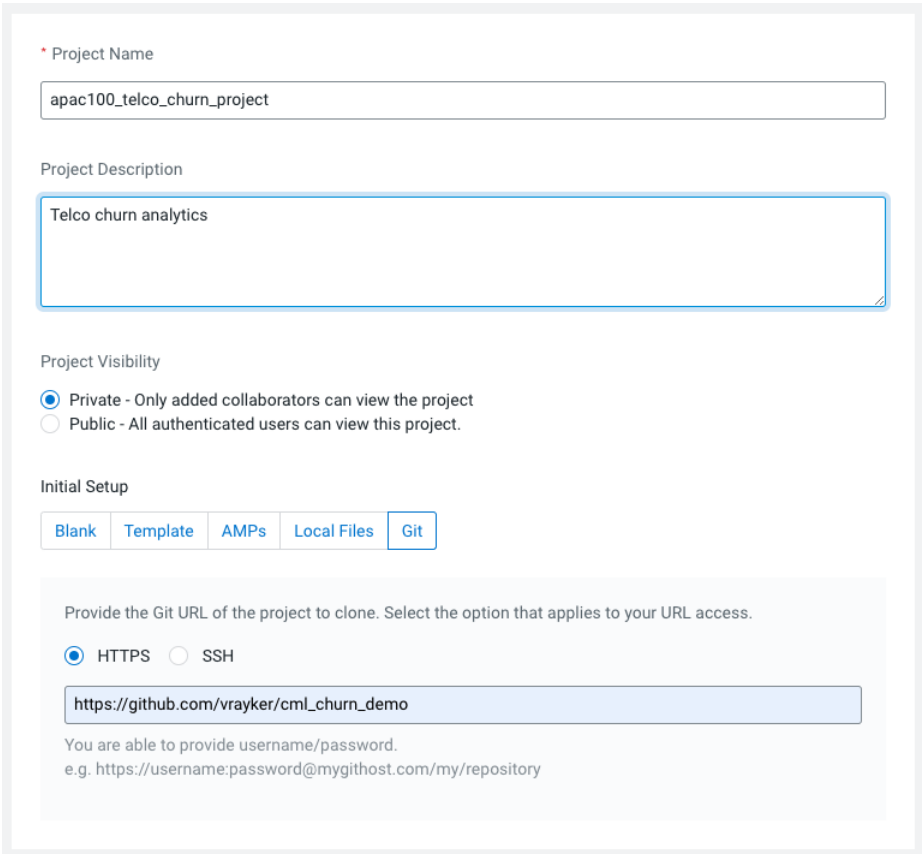
This will launch a ML workspace screen.

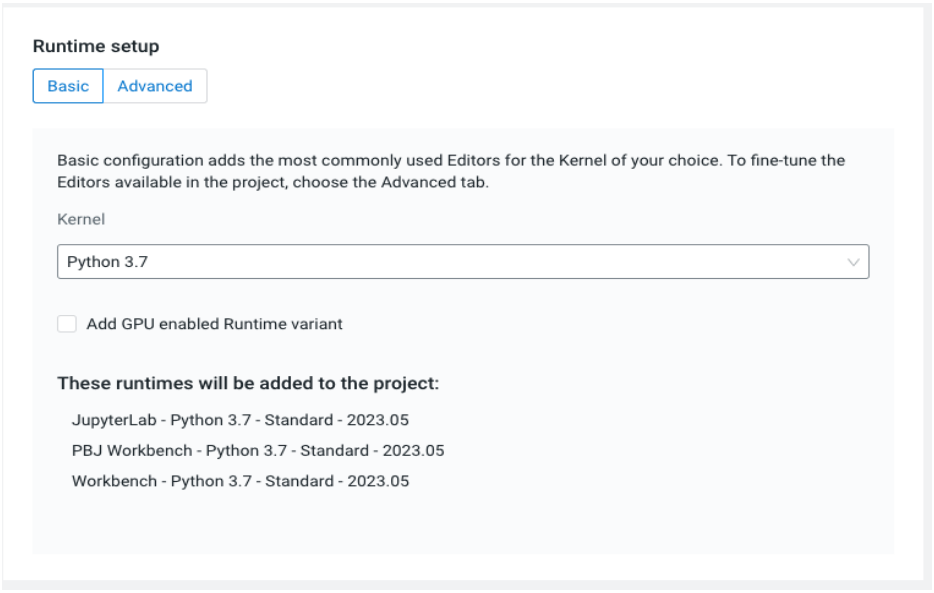


Click on “New Project” to start with the creation of our project.

Enter the following details in the New Project Page

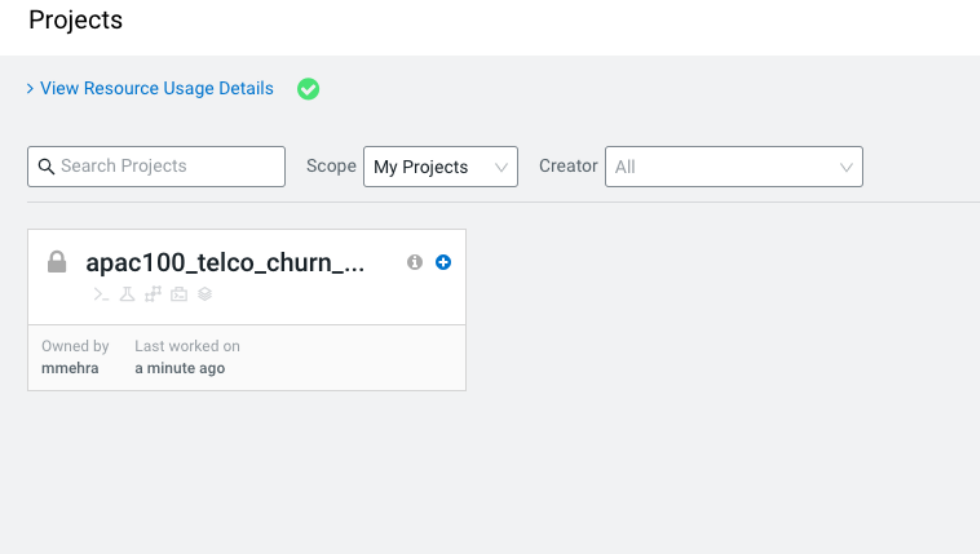
Project Name	<Workload_Username>_telco_churn_project
Project Description	Telco churn analytics
Project Visibility	Private
Initial Setup	Select “Git”
Protocol	HTTPS
Git URL of Project	https://github.com/mmehra12/cml_churn_demo
Runtime Setup	Basic
Kernel	Python 3.7



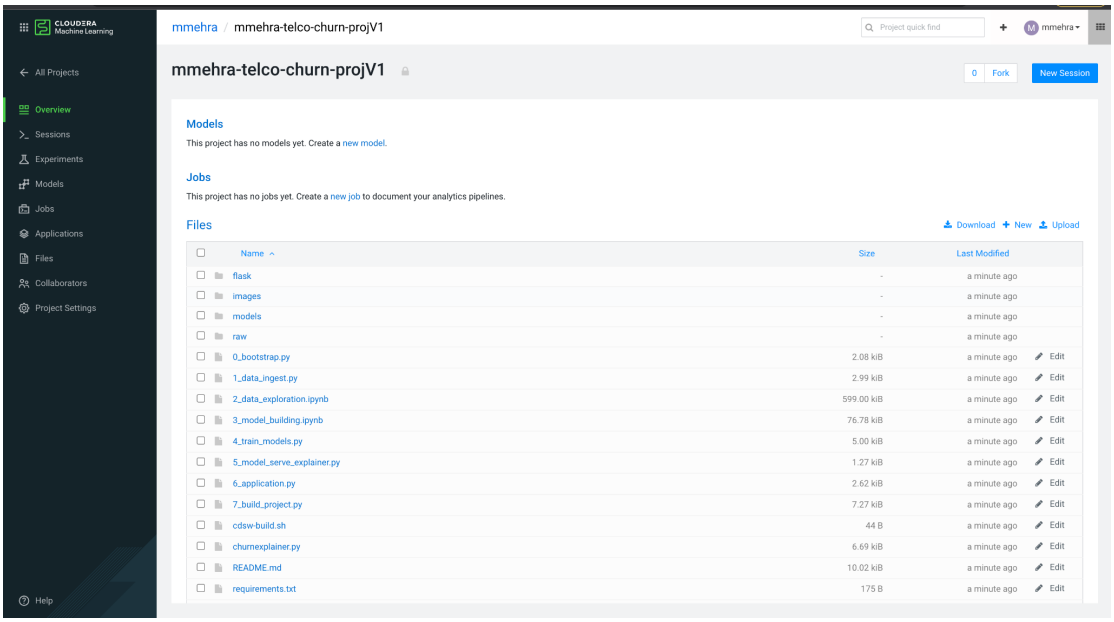


Click on **Create Project**

On successful creation you should now see the project on your Project page

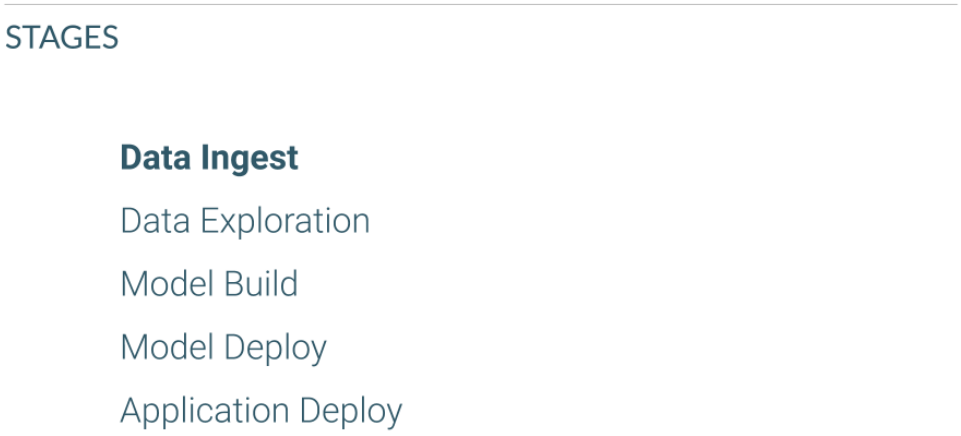


Clicking on it will take you to the Project that you just cloned from GitHub and you will be able to manage all the files from GitHub here.



Lab 1 - Data Ingest

In this lab, you will work on the Data Ingest Stage.

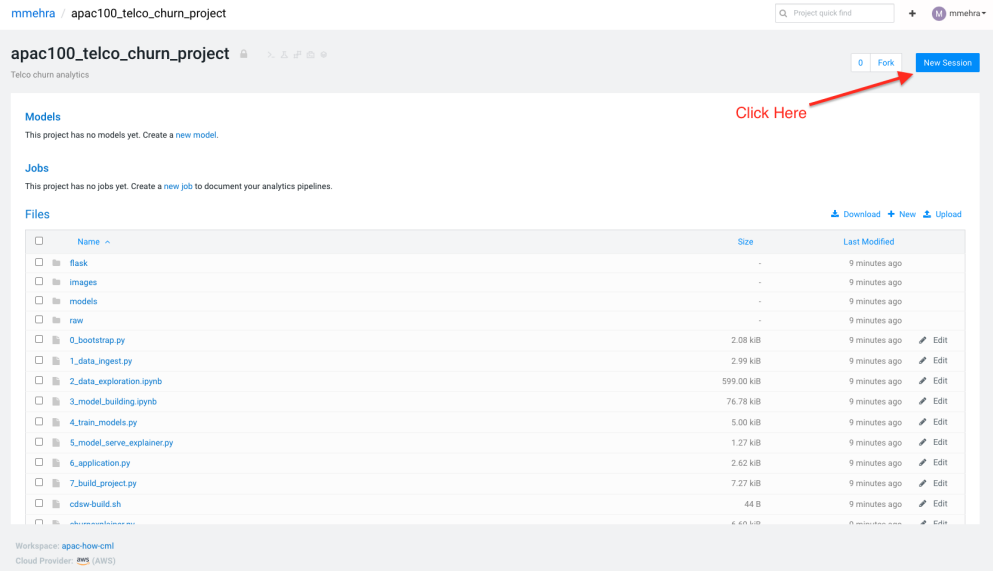


Bootstrap Script

We need to execute a bootstrap script at the start of the project. It will install the requirements, create the STORAGE environment variable and copy the data from raw/WA_Fn-UseC_-Telco-Customer-Churn-.csv into /datalake/data/churn of the STORAGE location, on AWS it will s3a://[something], on Azure it will be abfs://[something] and on CDSW cluster, it will be hdfs://[something]

Step 1 : Create New Session

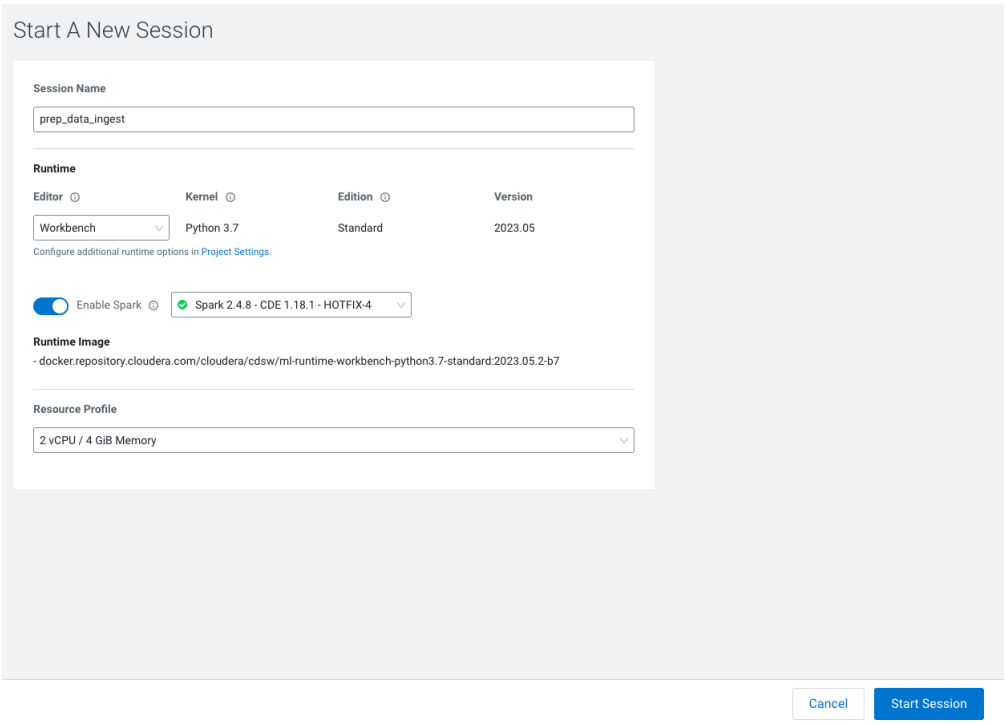
To create a new session you can go into your project and click on **New Session**



Start a "**NEW SESSION**" and use the below configuration.

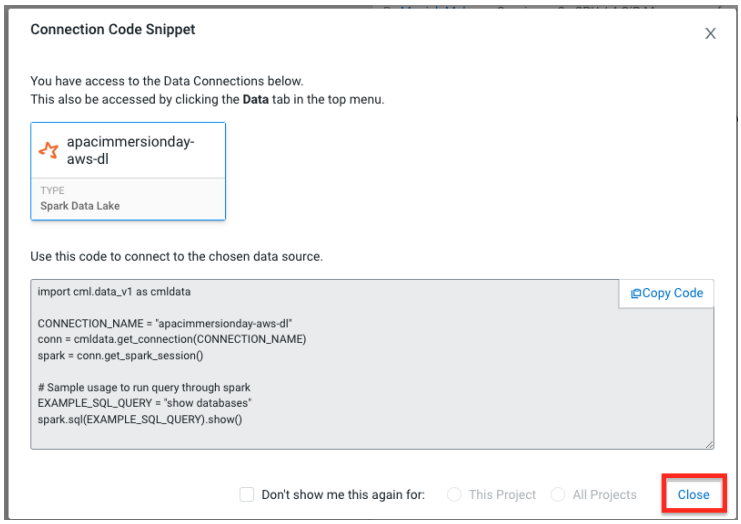
Session Name	prep_data_ingest
Runtime Editor	Workbench
Enable Spark	Yes - Spark version 2.4.8
Resource Profile	2 vCPU / 4 GiB

Click on **START SESSION**



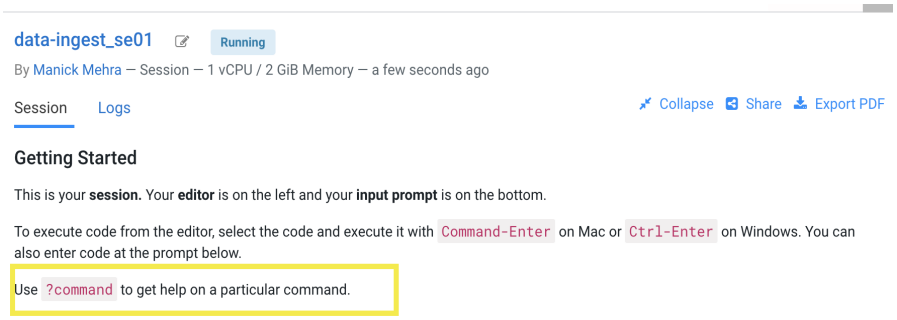
***IMPORTANT :* Please do not use the higher resource configurations.**

On successful creation of the session you will get a Dialog box with a code snippet to connect to this session from an application. For now we can click on Close

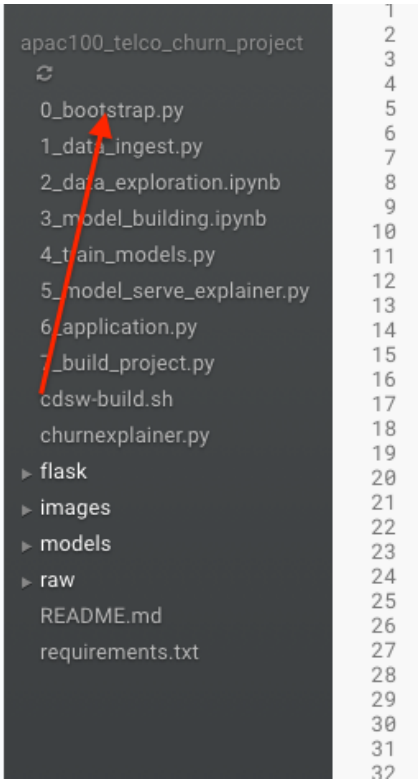


Step 2 : Execute Bootstrap script

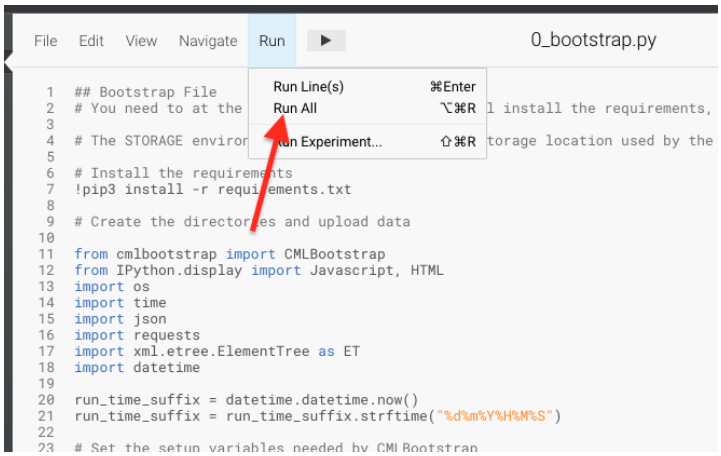
Once the session is ready you should get a similar message



Select the **0_bootstrap.py** on the left file browser



Select **Run -> Rull All**



As this will install all the dependencies and the first execution will take a bit tad longer as it needs to download all the binaries. You will start to see the execution logs on the right side of the screen.

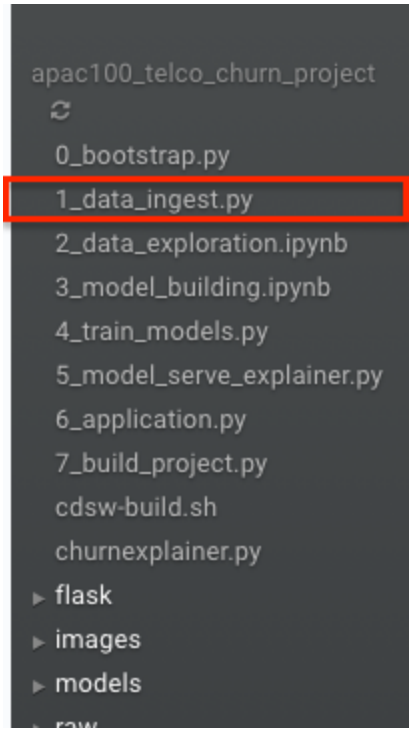
```
Collecting dill==0.3.1.1 (from -r requirements.txt (line 2))
  Downloading dill-0.3.1.1.tar.gz (151 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 152.0/152.0 kB 2.3 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Collecting lime==0.1.1.36 (from -r requirements.txt (line 3))
  Downloading lime-0.1.1.36.tar.gz (275 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 275.9/275.9 kB 4.7 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Collecting scikit-learn==0.21.3 (from -r requirements.txt (line 4))
  Downloading scikit_learn-0.21.3-cp37-cp37m-manylinux1_x86_64.whl (6.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6.7/6.7 MB 15.8 MB/s eta 0:00:00
Collecting xlrd==1.2.0 (from -r requirements.txt (line 5))
  Downloading xlrd-1.2.0-py2.py3-none-any.whl (103 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 103.3/103.3 kB 2.6 MB/s eta 0:00:00
Collecting pandas==0.25.1 (from -r requirements.txt (line 6))
  Downloading pandas-0.25.1-cp37-cp37m-manylinux1_x86_64.whl (10.4 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 10.4/10.4 MB 45.3 MB/s eta 0:00:00
Collecting numpy==1.17.2 (from -r requirements.txt (line 7))
  Downloading numpy-1.17.2-cp37-cp37m-manylinux1_x86_64.whl (20.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 20.3/20.3 MB 40.5 MB/s eta 0:00:00
Collecting flask==2.1.0 (from -r requirements.txt (line 8))
  Downloading Flask-2.1.0-py3-none-any.whl (95 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 95.2/95.2 kB 2.1 MB/s eta 0:00:00
```

This execution will take a couple of minutes. The last command to be executed is this and post this the bootstrap step is completed, and you can move to the next step.

```
!hdfs dfs -copyFromLocal /home/cdsw/raw/WA_Fn-UseC_-Telco-Customer-Churn-.csv $STORAGE/datalake/data/churn/WA_
22/04/19 10:35:57 WARN impl.MetricsConfig: Cannot locate configuration: tried hadoop-metrics2-s3a-file-syste
m.properties,hadoop-metrics2.properties
22/04/19 10:35:57 INFO impl.MetricsSystemImpl: Scheduled Metric snapshot period at 10 second(s).
22/04/19 10:35:57 INFO impl.MetricsSystemImpl: s3a-file-system metrics system started
22/04/19 10:35:57 INFO s3a.IDBDelegationTokenBinding: Maybe renewing Knox Token when deploying unbonded token
22/04/19 10:35:57 INFO s3a.IDBDelegationTokenBinding: There is no Knox Token available, fetching one from IDB
roker...
22/04/19 10:35:57 INFO idbroker.AbstractIDBClient: Authenticating with IDBroker requires Kerberos
22/04/19 10:35:57 INFO idbroker.AbstractIDBClient: Kerberos credentials are available, using Kerberos to esta
blish a session. UGI=mmehra/7e588bef-8ef4-45dd-81d3-084ef2499ba1.mlx.cloudera.site@PTP-DEMO.DP5I-5VKQ.CLOUDER
A.SITE (auth:KERBEROS)
Apr 19, 2022 10:35:57 AM org.apache.knox.gateway.shell.KnoxSession createClient
INFO: Using default JAAS configuration
22/04/19 10:35:58 INFO s3a.IDBDelegationTokenBinding: Bonded to Knox token eyJqa3...SSlPkw
22/04/19 10:35:58 INFO Configuration.deprecation: No unit for fs.s3a.connection.request.timeout(0) assuming S
ECONDS
22/04/19 10:35:58 INFO s3a.IDBDelegationTokenBinding: Maybe renewing Knox Token when fetching AWS credentials
22/04/19 10:35:58 INFO s3a.IDBDelegationTokenBinding: Using existing Knox Token: eyJqa3...SSlPkw
22/04/19 10:35:58 INFO idbroker.AbstractIDBClient: Creating Knox CAB session using Knox DT eyJqa3...SSlPkw
...
22/04/19 10:35:58 INFO s3a.IDBDelegationTokenBinding: Maybe renewing Knox Token when fetching AWS credentials
22/04/19 10:35:58 INFO s3a.IDBDelegationTokenBinding: Using existing Knox Token: eyJqa3...SSlPkw
22/04/19 10:35:58 INFO s3a.IDBDelegationTokenBinding: Maybe renewing Knox Token when fetching AWS credentials
22/04/19 10:35:58 INFO s3a.IDBDelegationTokenBinding: Using existing Knox Token: eyJqa3...SSlPkw
22/04/19 10:35:58 INFO s3a.IDBDelegationTokenBinding: Maybe renewing Knox Token when fetching AWS credentials
22/04/19 10:35:58 INFO s3a.IDBDelegationTokenBinding: Using existing Knox Token: eyJqa3...SSlPkw
copyFromLocal: `s3a://ot-ptp-demo/datalake/data/churn/WA_Fn-UseC_-Telco-Customer-Churn-.csv': File exists
22/04/19 10:35:59 INFO impl.MetricsSystemImpl: Stopping s3a-file-system metrics system...
22/04/19 10:35:59 INFO impl.MetricsSystemImpl: s3a-file-system metrics system stopped.
22/04/19 10:35:59 INFO impl.MetricsSystemImpl: s3a-file-system metrics system shutdown complete.
```

Execute Data Ingest Script

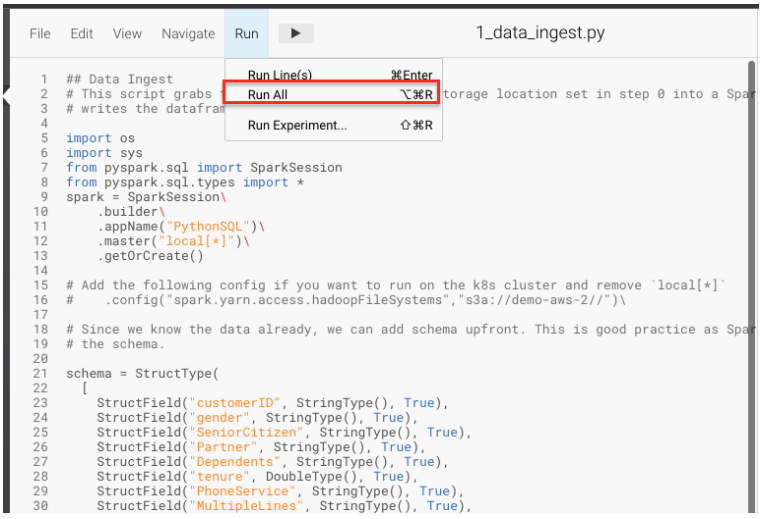
In the same Workbench, open the script “*1_data_ingest.py*”



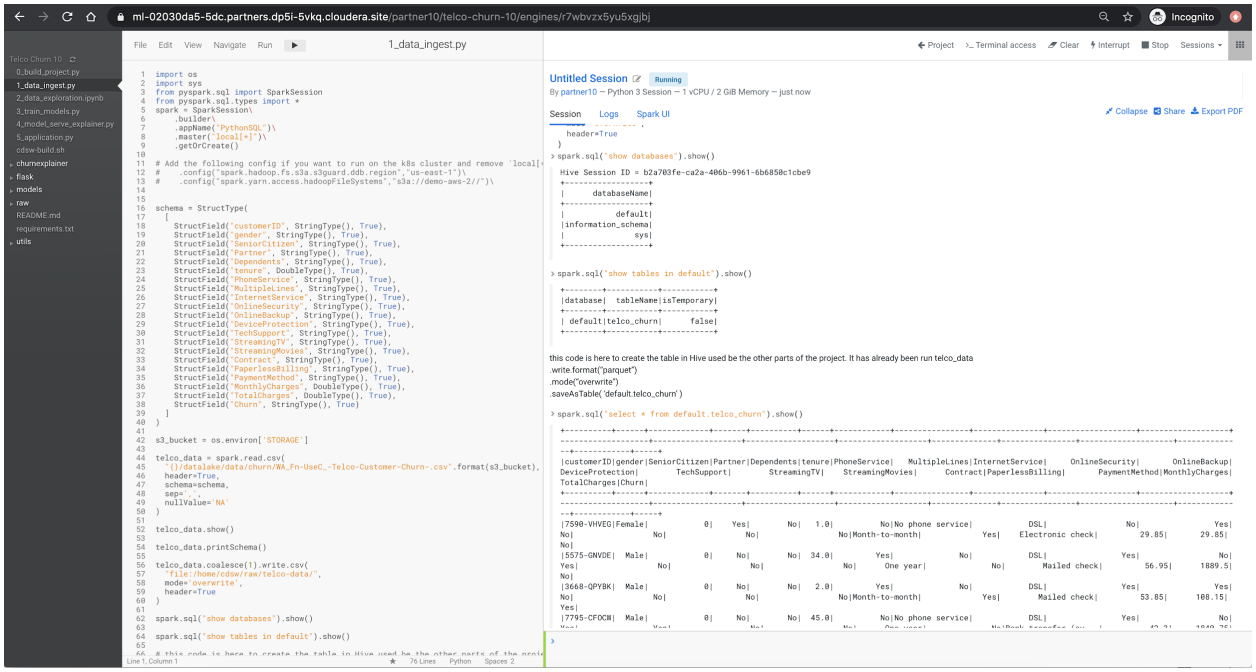
This script will load the data from an S3 bucket using Spark.

It demonstrates how to read from files and tables using Spark file and SQL operators.

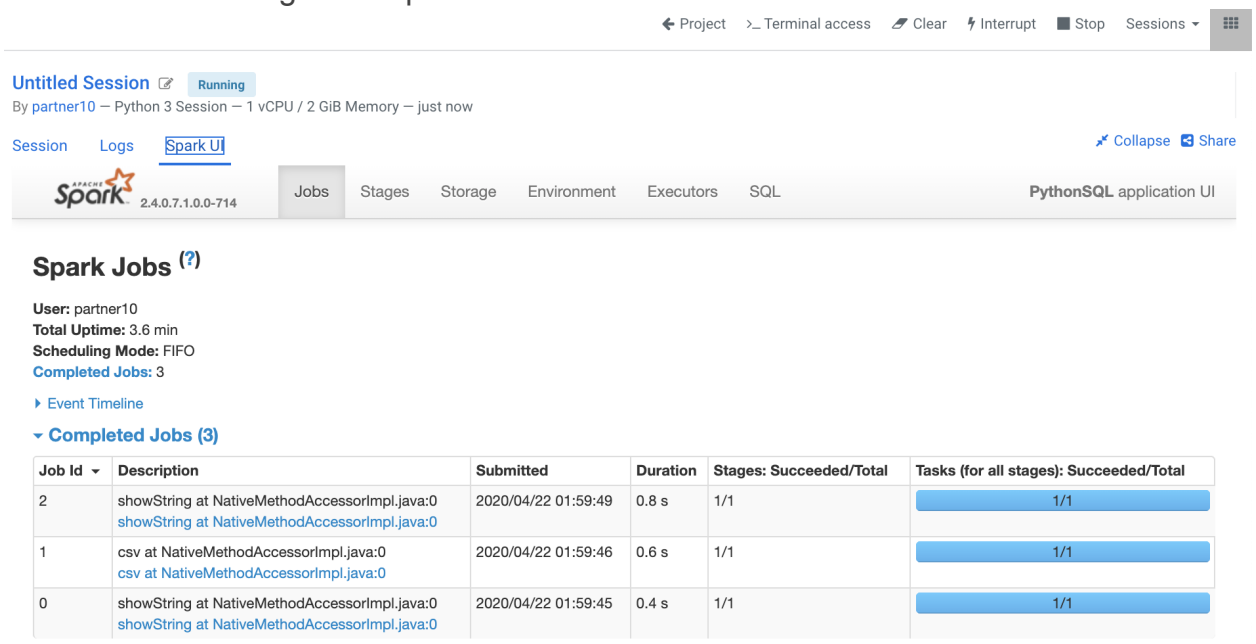
Click on **Run → Run All.**



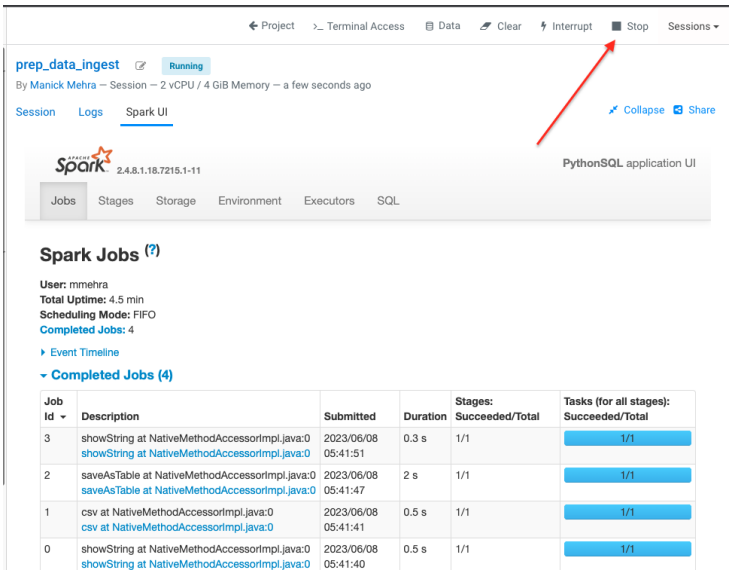
Session output will show the code execution results. Observe the database, table, and data from the table.



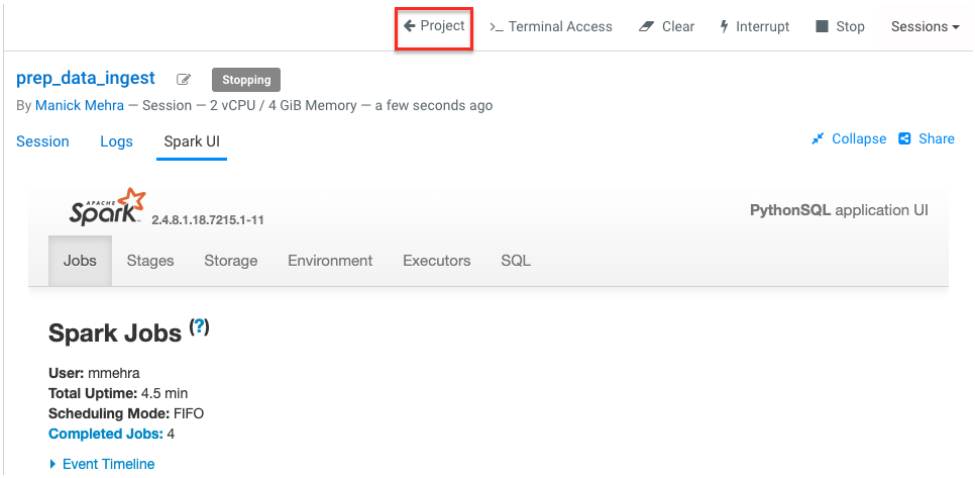
Also examine the logs and Spark UI for details of the run.



Stop the session once you data ingestion completes



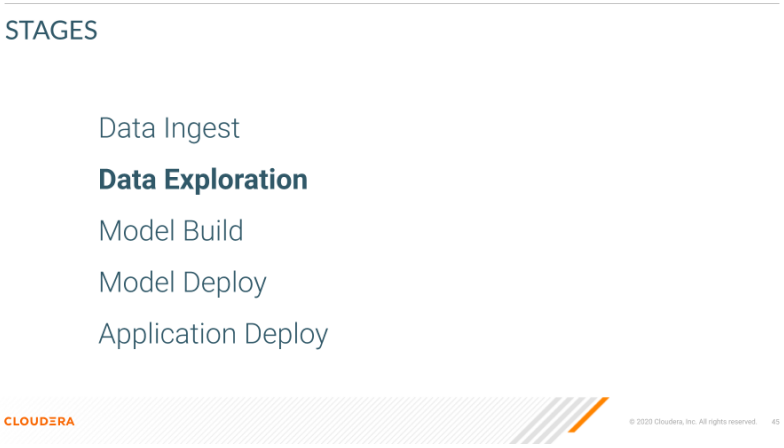
Go back to the Project page



Lab 2 - Data Exploration

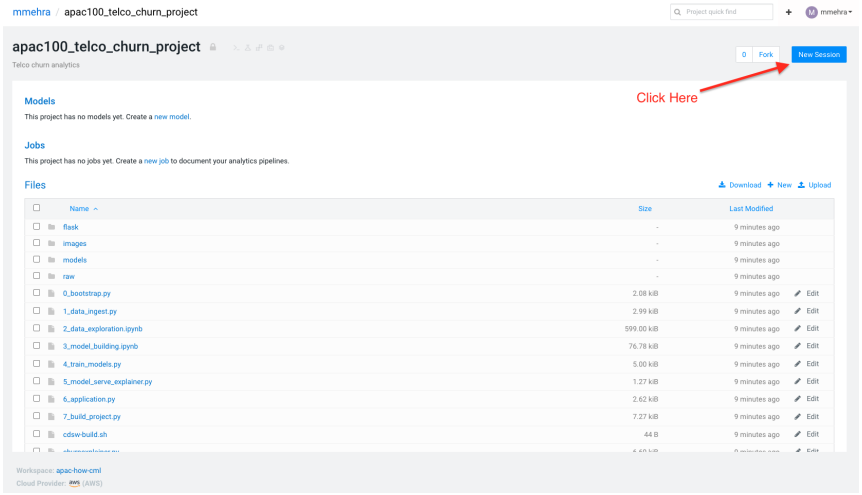
In this lab, you will explore some dataset using a different editor from the previous lab.

In fact, in this lab we are going to use a popular notebook, Jupyter, to show the flexibility of CML that allows you to bring your own editor.



Step 1 : Create New Session

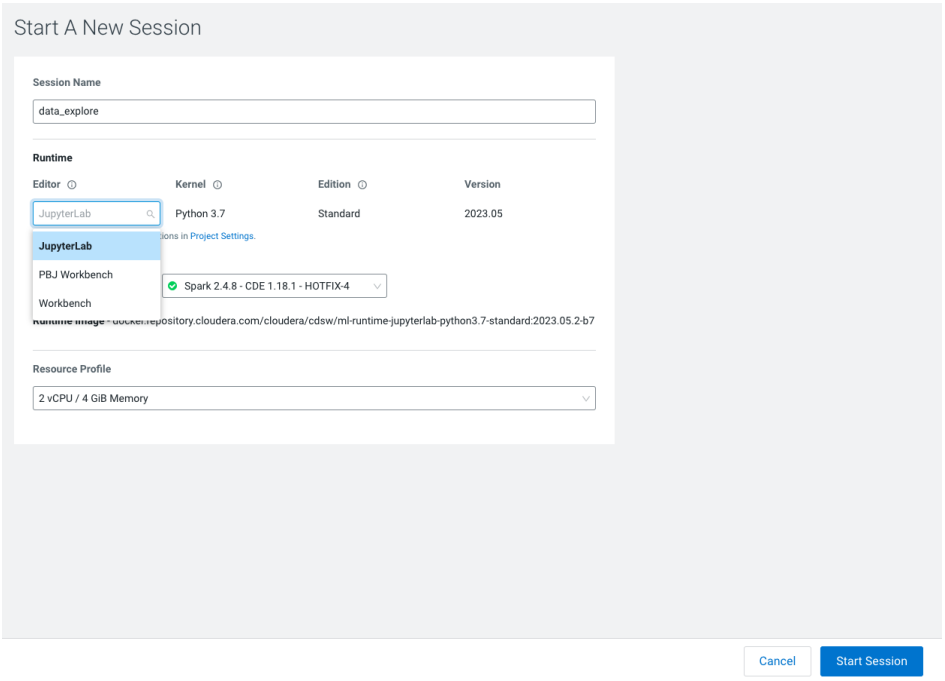
To create a new session you can go into your project and click on **New Session**



Start a “**NEW SESSION**” and use the below configuration.

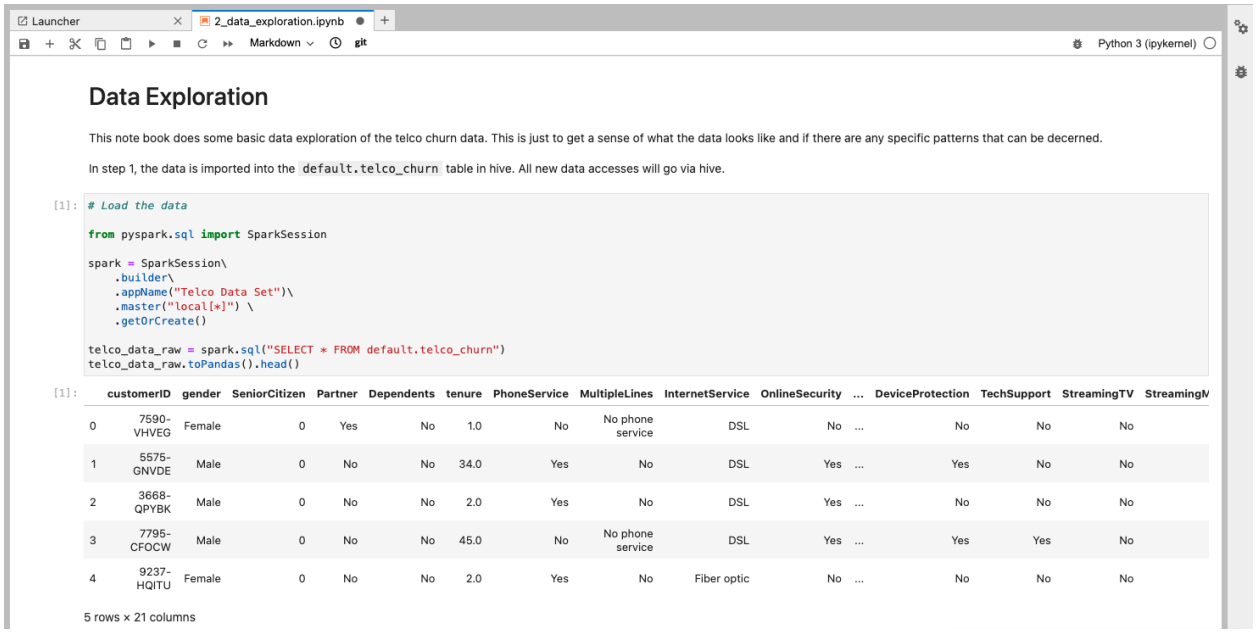
Session Name	data_explore
Runtime Editor	JupyterLab
Enable Spark	Yes - Spark version 2.4.8
Resource Profile	2 vCPU / 4 GiB

Click on **START SESSION**



Step 2 : Execute Data Exploration script

Double-Click on **2_data_exploration.ipynb** it will take you into the notebook



As you notice we are interacting with the data lake, in particular with the database previously created

At this point the data scientist realized that they forgot to add a dependency at the time of bootstrap process. They can still do that from here. Let's see how that can be done.

For our data exploration, if you run the script without making any changes you will see that it will fail at a point because of a missing dependency.

Feature Distributions

We want to examine the distribution of our features, so start with them one at a time.

Seaborn has a standard function called distplot() that allows us to easily examine the distribution of a column of a pandas dataframe or a numpy array.

```
import matplotlib.pyplot as plt
import seaborn as sns

plt.grid(b=None)
plt.title("Tenure Distribution",color='grey')
plt.xticks(color='grey')
plt.yticks(color='grey')

sns.set_style("whitegrid")
sns.despine(left=True,bottom=True)


def plotter():
    sns.distplot(sample_data['tenure'], kde=False, axlabel=False)
plotter()
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
/tmp/ipykernel_214/1760403484.py in <module>
      1 import matplotlib.pyplot as plt
----> 2 import seaborn as sns
      3
      4 plt.grid(b=None)
      5 plt.title("Tenure Distribution",color='grey')

ModuleNotFoundError: No module named 'seaborn'
```

This can be taken care of by adding the missing dependency before we make use of it.

Add this command at the start of your script and Run All Cells again.

To add a new command block got to the start of the script and select the first block of code, Click on the + sign at the top of the editor and enter the following command

!pip install seaborn

Data Exploration

This note book does some basic data exploration of the telco churn data. This is just to get a sense of what the data looks like and if there are any spec

In step 1, the data is imported into the `default.telco_churn` table in hive. All new data accesses will go via hive.

[1]:
```
# Load the data

from pyspark.sql import SparkSession

spark = SparkSession\
    .builder\
    .appName("Telco Data Set")\
    .master("local[*]") \
    .getOrCreate()

telco_data_raw = spark.sql("SELECT * FROM default.telco_churn")
telco_data_raw.toPandas().head()
```

```
Hive Session ID = e0db9aef-8440-4611-a2c2-993b66c671fc
```

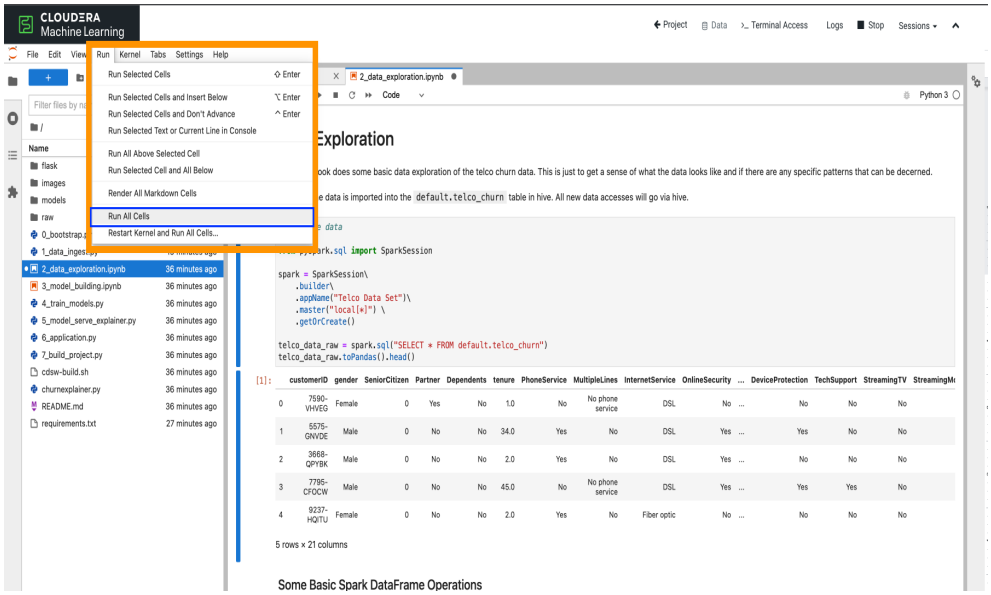
[1]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	...	Device
0	7590-VHVEG	Female	0	Yes	No	1.0	No	No phone service	DSL	No	...	
1	5575-GNVDE	Male	0	No	No	34.0	Yes	No	DSL	Yes	...	
2	3668-QPYBK	Male	0	No	No	2.0	Yes	No	DSL	Yes	...	
3	7795-CFOCW	Male	0	No	No	45.0	No	No phone service	DSL	Yes	...	
4	9237-HQITU	Female	0	No	No	2.0	Yes	No	Fiber optic	No	...	

5 rows × 21 columns

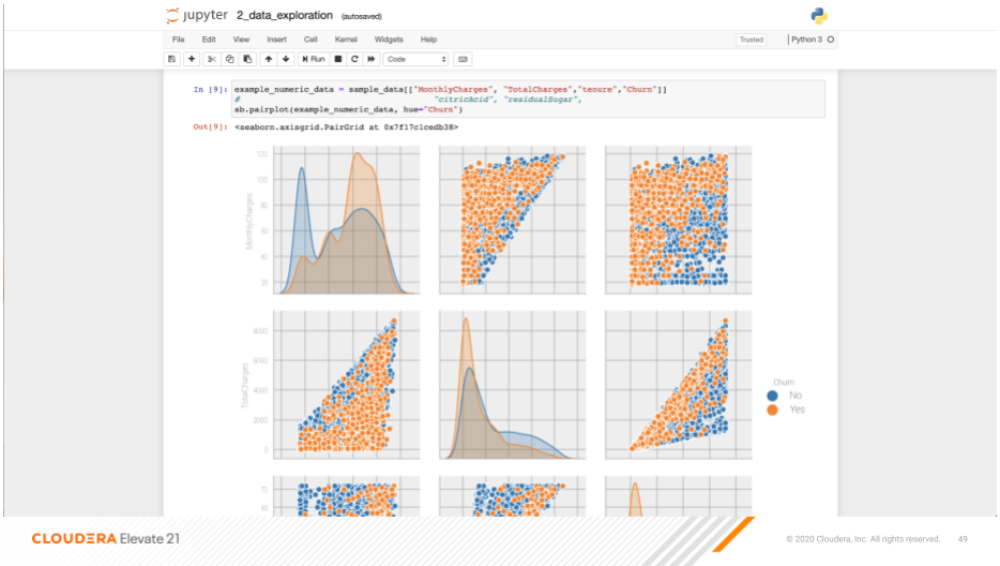
[ ]:
```
!pip install seaborn
```

Some Basic Spark DataFrame Operations



You are ready to run the notebook, go to *Cell*, *Run All*

And you can analyze the plotted graphs



Now we can go back to *Project*

This concludes this lab.

Lab 3 - Model Build/Train Experiment

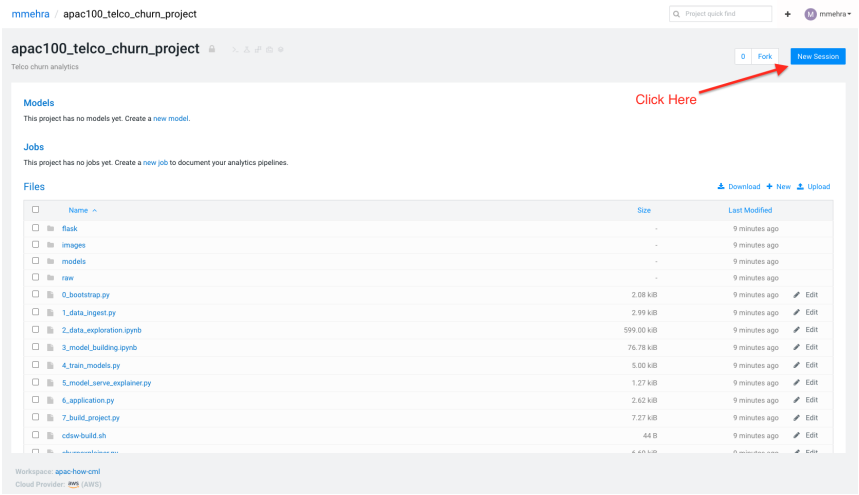
In this lab, you will build and train the model, using the Experiment feature form CML that allows you to run offline different training sessions, with different parameters configuration, for your model so that you could promote in “Production” that configuration that showed the best results, KPIs.

Model Building

Step 1 : Create Session

We will use a Jupyter Notebook to show the process of selecting and building the model to predict churn. It also shows more details on how the LIME model is created and a bit more on what LIME is actually doing.

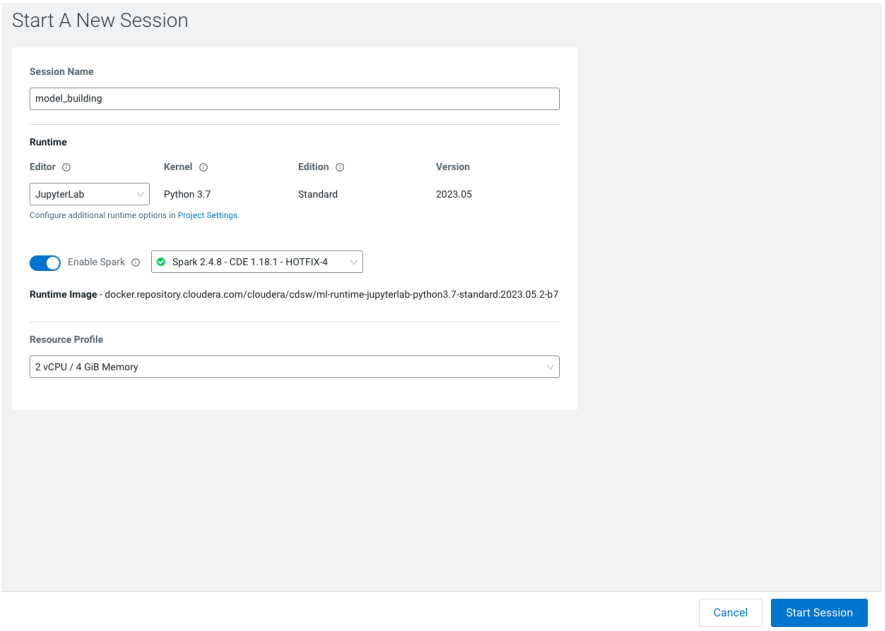
To create a new session you can go into your project and click on **New Session**



Start a “**NEW SESSION**” and use the below configuration.

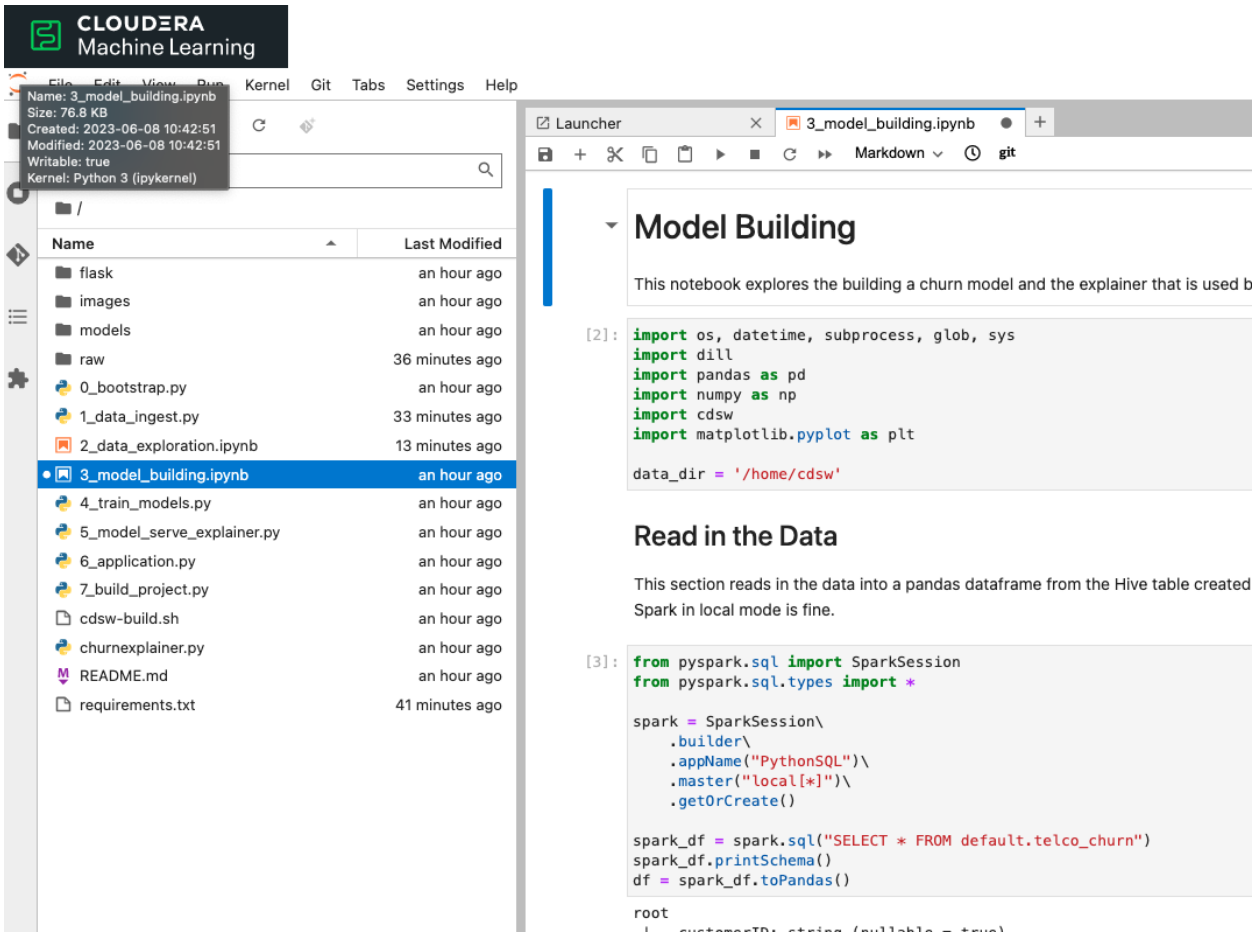
Session Name	model_building
Runtime Editor	JupyterLab
Enable Spark	Yes - Spark version 2.4.8
Resource Profile	2 vCPU / 4 GiB

Click on **START SESSION**



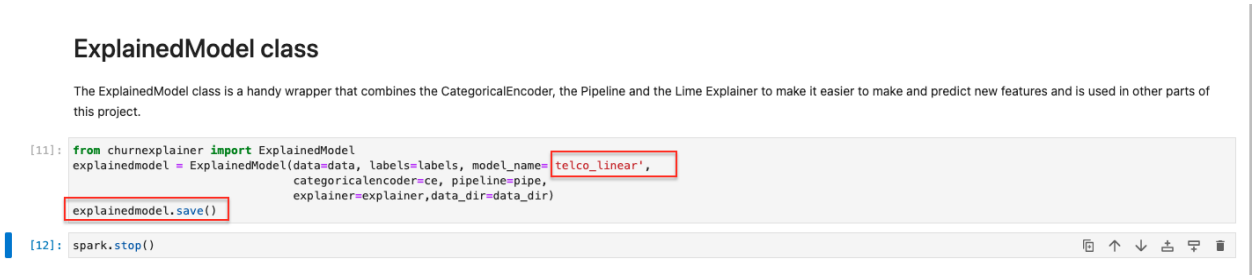
Step 2 : Execute model building script

Open the *3_model_building.ipynb* file.

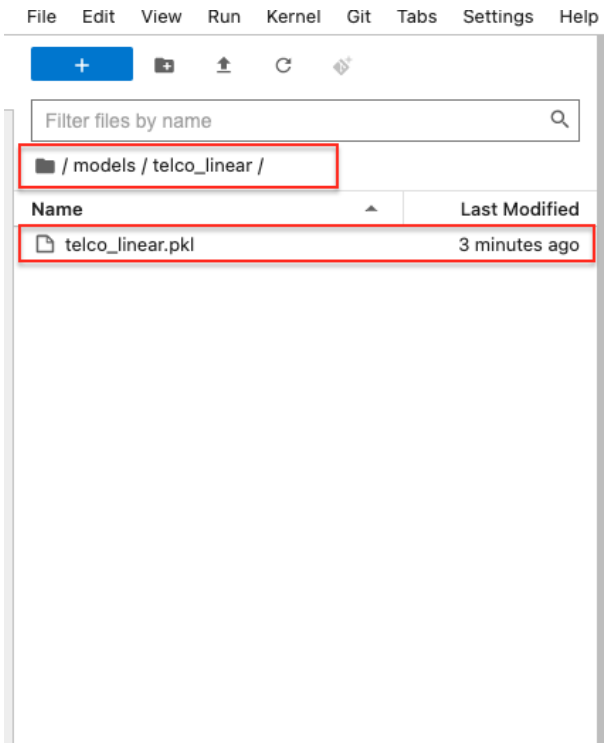


At the top of the page click **Run > Run All Cells**.

Once the script finishes executing you can scroll to the bottom of the screen and check the part of the code where the built model is getting saved in pickle format.



You can now see this pickle file created in your project directory as shown below.

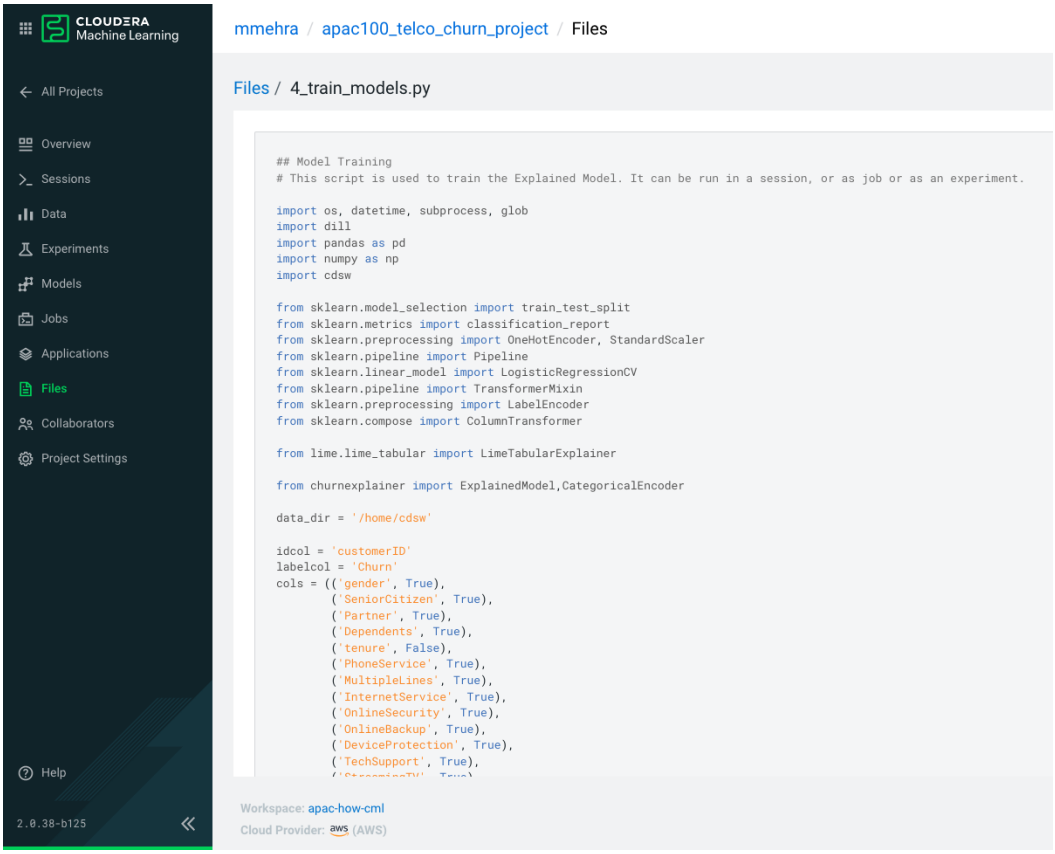


Model Training

Step 1 : Create a session

For the training portion of the lab we will use the file ***4_train_models.py***

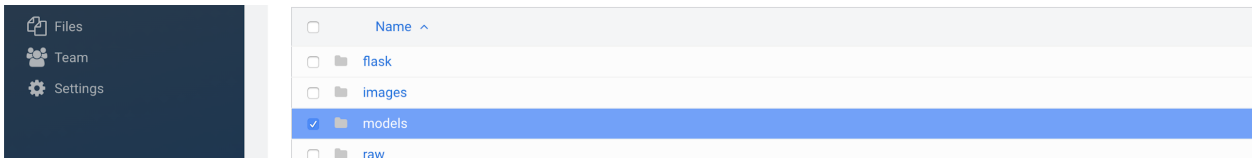
Click on it and familiarize yourself with the code. This can be done by going into your project, clicking on ***Files*** in the left pane, and viewing the file ***4_train_models.py***

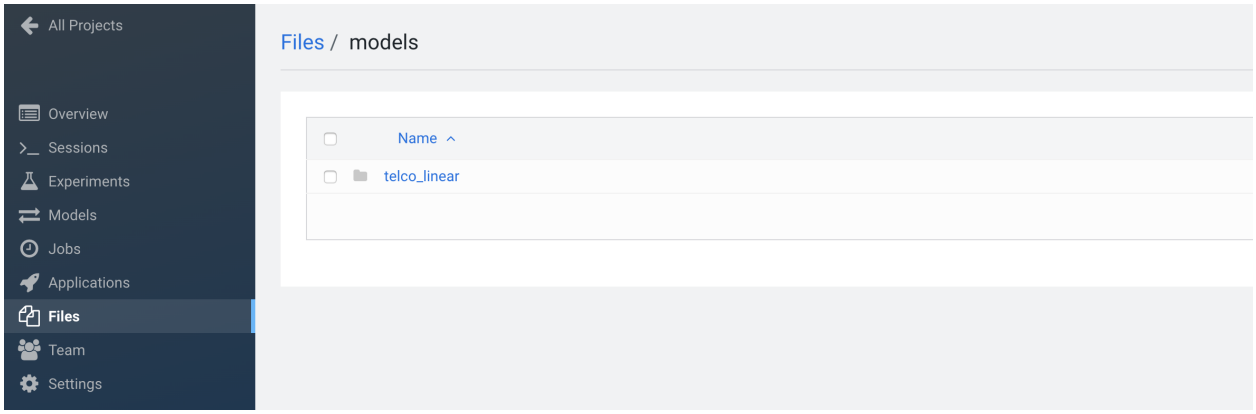


The code also keeps track of the metrics associated to a particular train configuration:

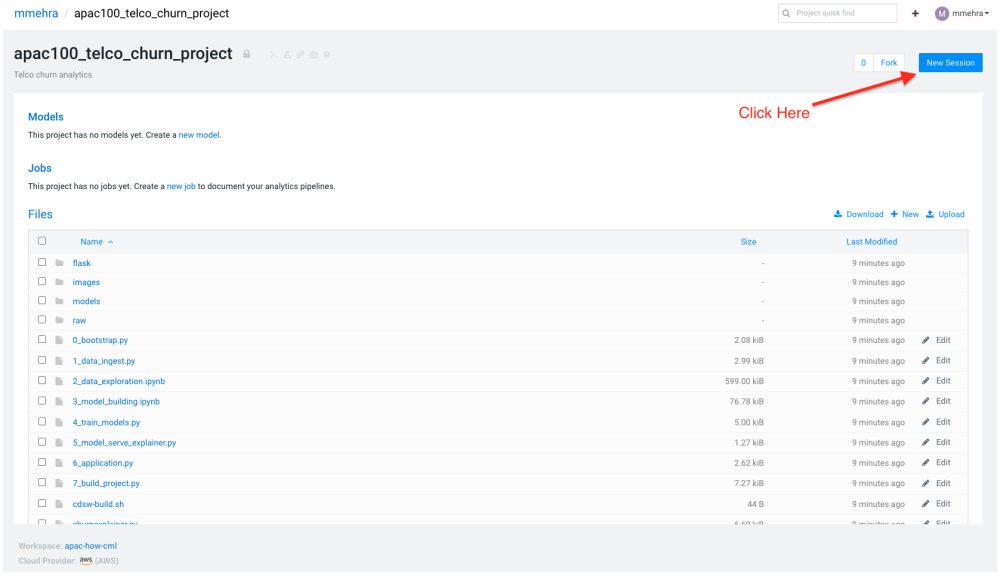
```
mlflow.log_metric("train_score", round(train_score,2))
mlflow.log_metric("test_score", round(test_score,2))
```

The real model that is being trained can be seen here. Go to the `models` folder:





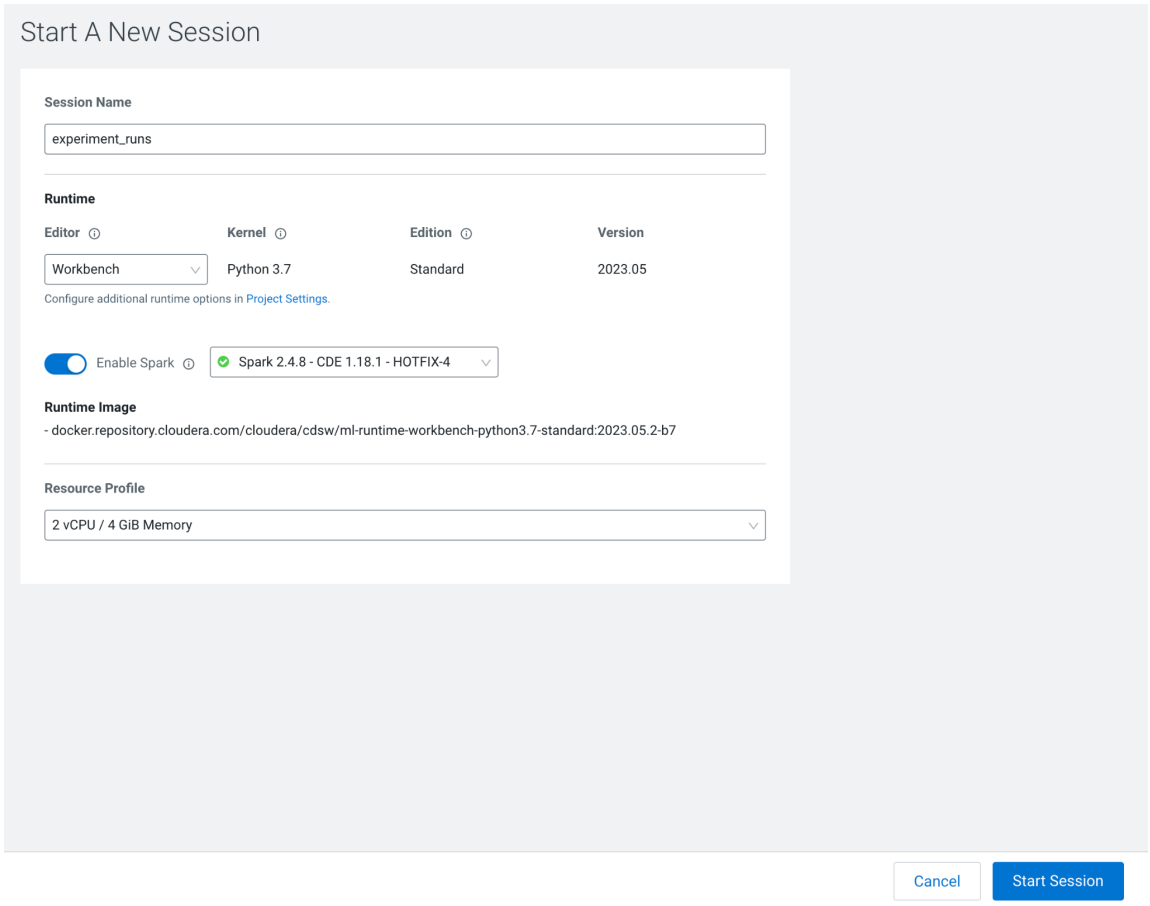
To create a new session you can go into your project, select ***Sessions*** in the left pane, and click on **New Session**



Start a "**NEW SESSION**" and use the below configuration.

Session Name	experiment_runs
Runtime Editor	Workbench
Enable Spark	Yes - Spark version 2.4.8
Resource Profile	2 vCPU / 4 GiB

Click on **START SESSION**

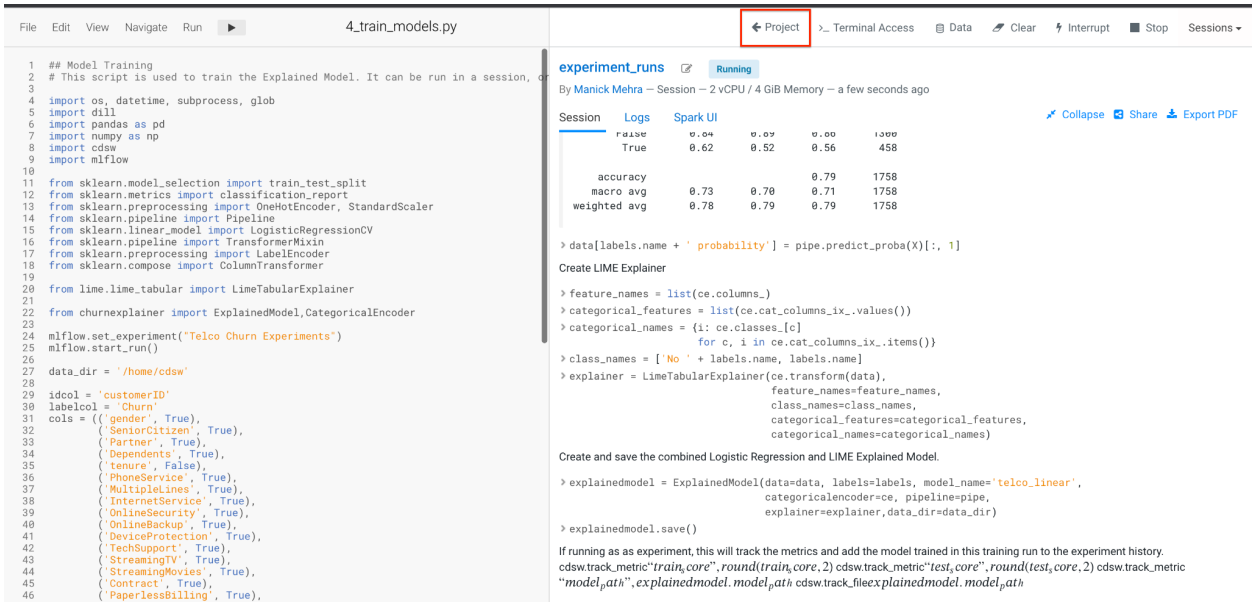


Step 2 : Run Experiments

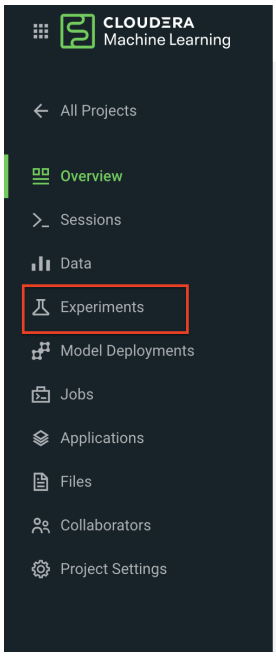
Select the file ***4_train_models.py***

Select ***Run -> Rull All***

Once this runs successfully Go back to the **Project Page**

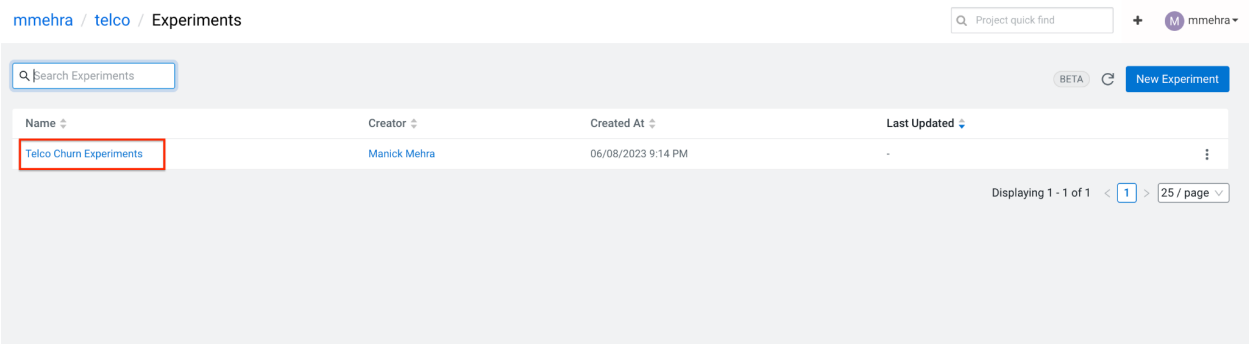


Select ***Experiments*** from the left tab



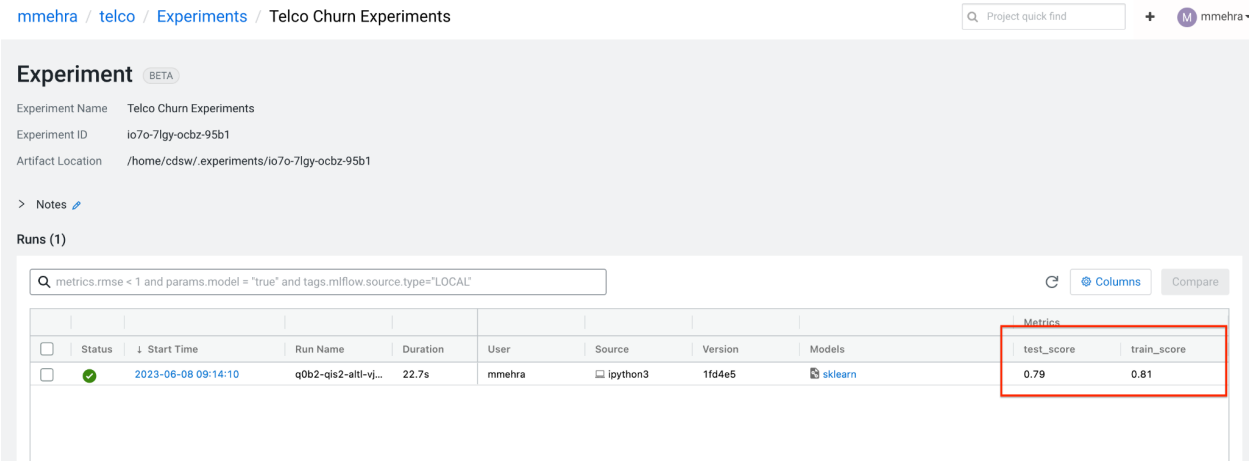
You will see that the experiment we ran from the session shows up here as it has the same name that we specified in the script.

```
import dill
import pandas as pd
import numpy as np
import cdsw
import mlflow

from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegressionCV
from sklearn.pipeline import TransformerMixin
from sklearn.preprocessing import LabelEncoder
from sklearn.compose import ColumnTransformer

from lime.lime_tabular import LimeTabularExplainer

from churnexplainer import ExplainedModel,CategoricalEncoder

mlflow.set_experiment("Telco Churn Experiments")
mlflow.start_run()

data_dir = '/home/cdsw'

idcol = 'customerID'
labelcol = 'Churn'
cols = (('gender', True),
        ('SeniorCitizen', True),
```

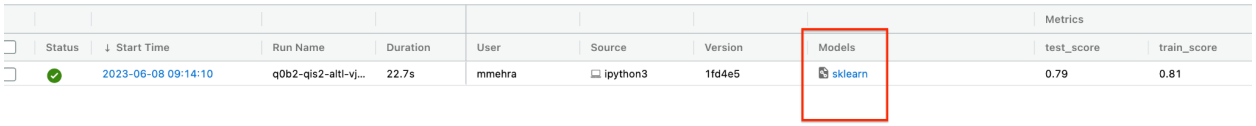


Step 3 : Verify Experiment metrics

Click on the Experiment you just ran and look for the train score and test score



We can run multiple experiments and based on the best score we can decide which model to go with. The model is also one of the columns in the Experiment details.

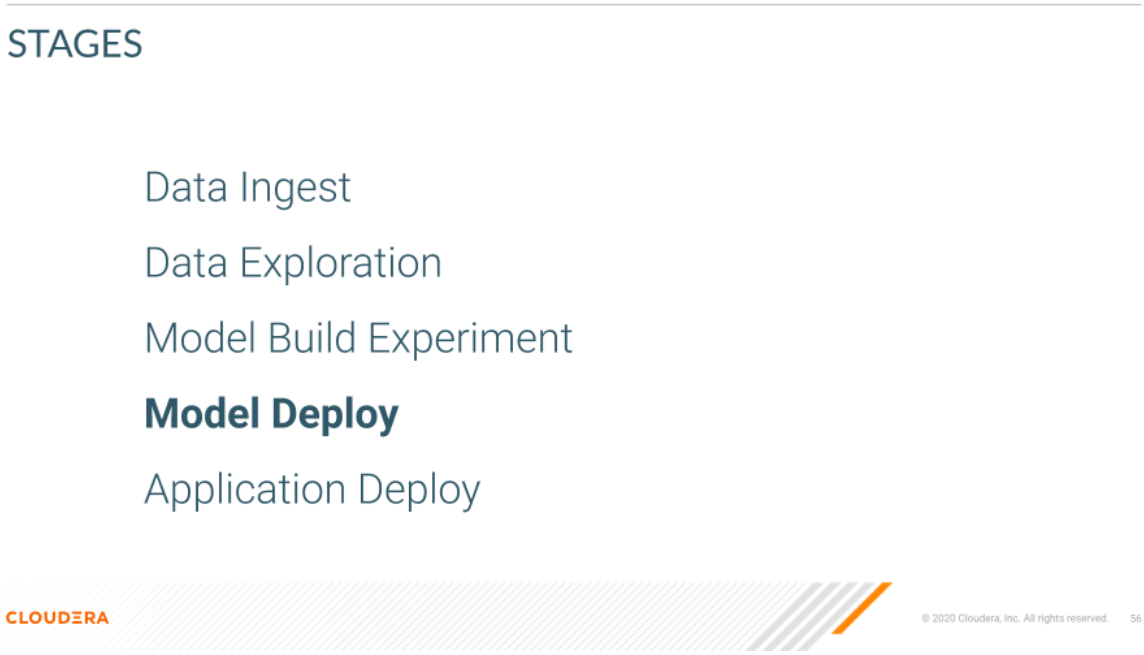


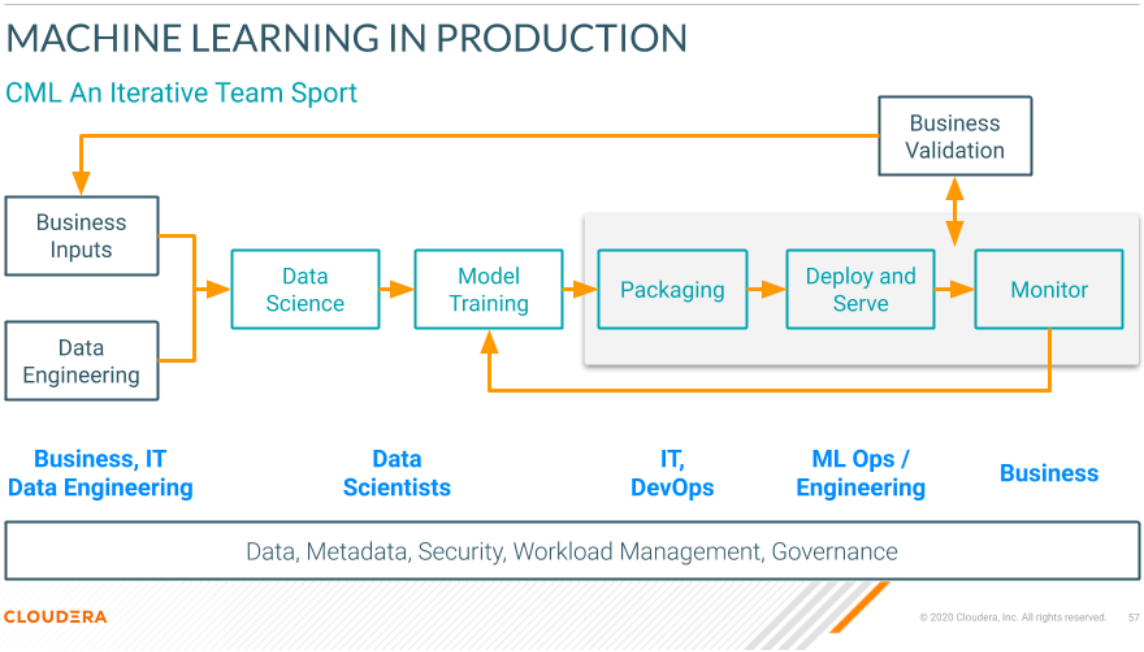
									Metrics	
	Status	↓ Start Time	Run Name	Duration	User	Source	Version	Models	test_score	train_score
	✓	2023-06-08 09:14:10	q0b2-qis2-altl-vj...	22.7s	mmehra	ipython3	1fd4e5	sklearn	0.79	0.81

This concludes the Lab

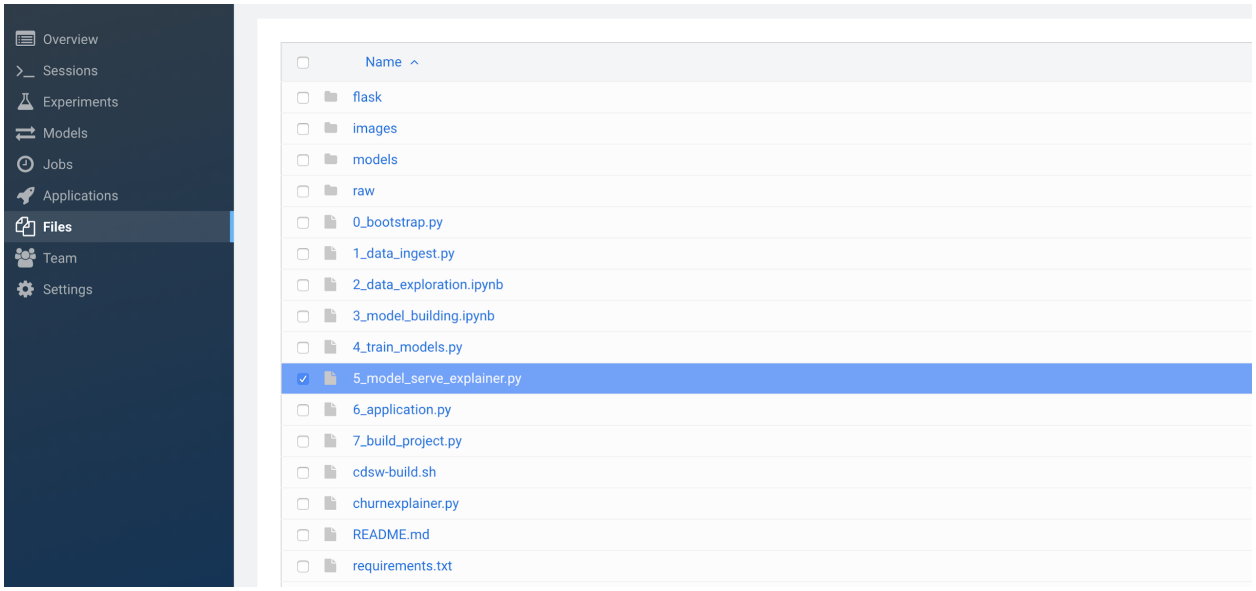
Lab 4 - Model Deploy/Serve

In this lab, you will deploy/serve the model that you have trained in the Lab 3 as a REST endpoint. The model can be invoked as-needed, in real-time or batch fashion, by external services that need to score the prediction implemented by the model.





Click on ***Files*** in the left tab, and go to the file ***5_model_serve_explainer.py***:



This is the script for serving the model, and the line below is loading the pickel model we have generated in the Lab 3

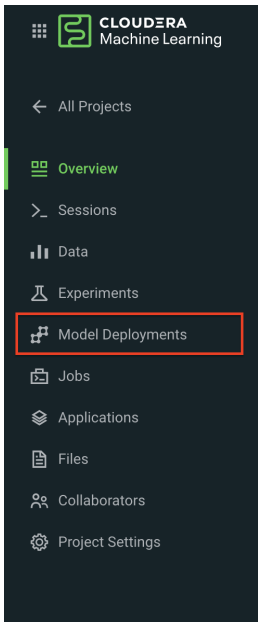
and the function is the one that takes the input arguments, passes them to the model for scoring and gives back the result of the score.

```
def explain(args):
    #data = dict(ChainMap(request.args, em.default_data))
    data = dict(ChainMap(args, em.default_data))
    data = em.cast_dct(data)
    probability, explanation = em.explain_dct(data)
    return jsonify({'data': dict(data),
                    'probability': probability,
                    'explanation': explanation})
```

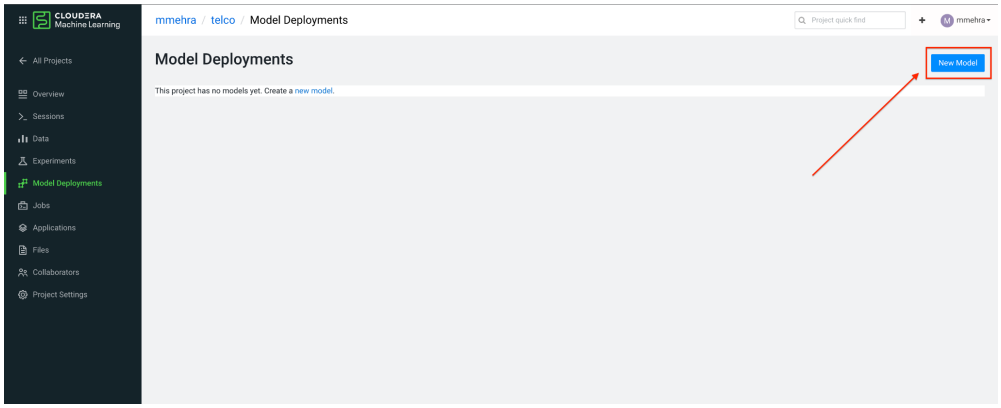
Copy the entire string as shown below because we are going to use it in the future and a sample JSON input parameters for the model.

```
{"StreamingTV":"No","MonthlyCharges":70.35,"PhoneService":"No","PaperlessBilli
ng":"No","Partner":"No","OnlineBackup":"No","gender":"Female","Contract":"Mont
h-to-month","TotalCharges":1397.475,"StreamingMovies":"No","DeviceProtection":
"No","PaymentMethod":"Bank transfer
(automatic)","tenure":29,"Dependents":"No","OnlineSecurity":"No","MultipleLine
s":"No","InternetService":"DSL","SeniorCitizen":"No","TechSupport":"No"}
```

Now go to ***Model Deployments***

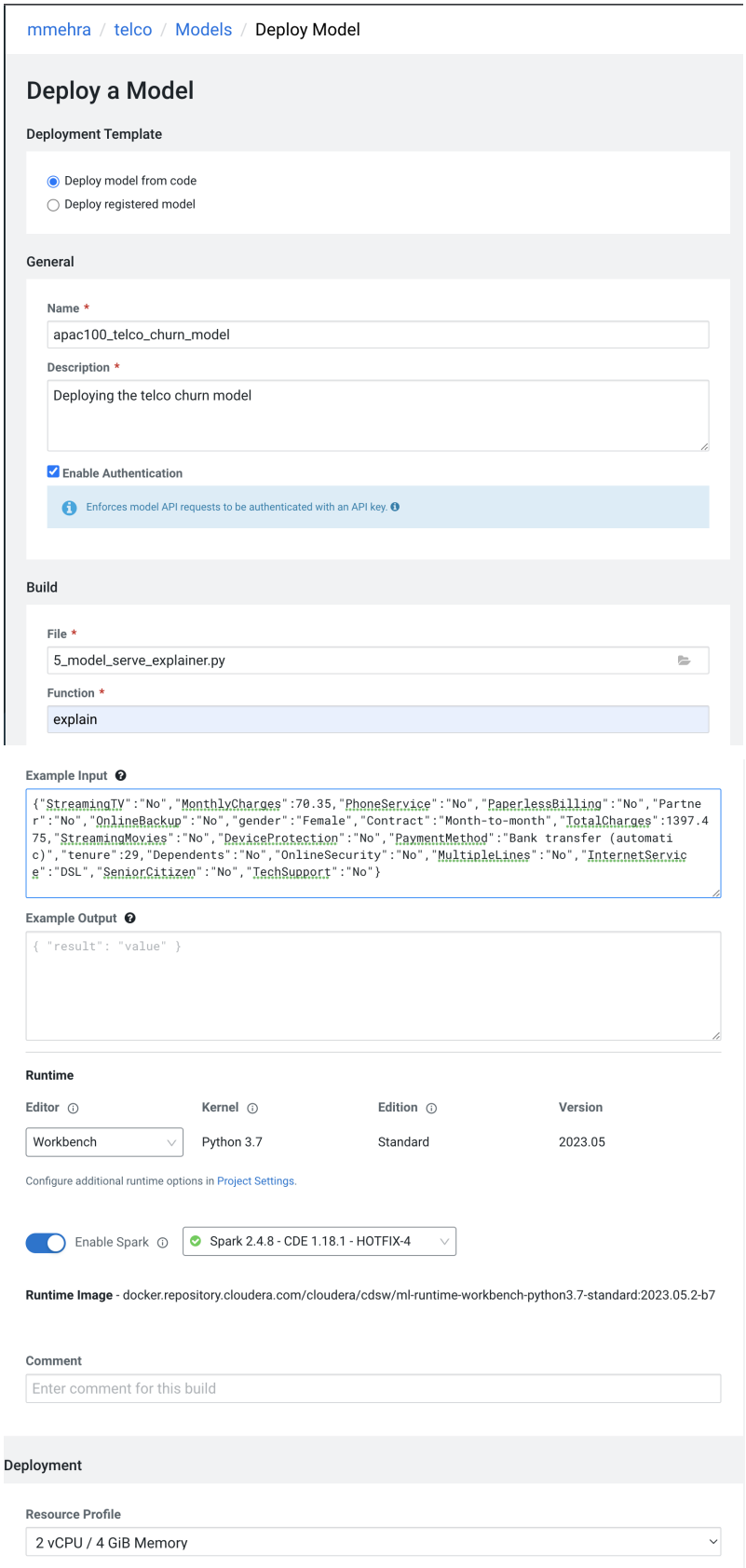


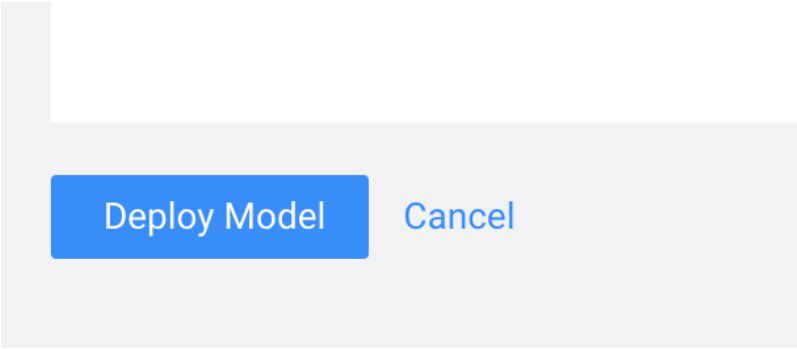
Click ***New Model***



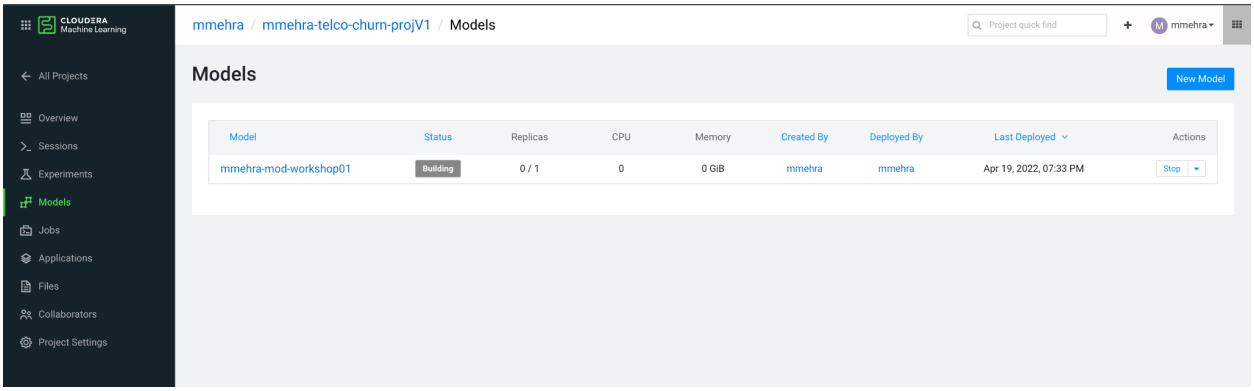
Enter the following details

Deployment Template	Deploy Model from Code
Name	<workload_username>_telco_churn_mod
Description	Deploying the telco churn model
Enable Authentication	**False (Disable)**
File	5_model_serve_explainer.py
Function	explain
Example Input	`{"StreamingTV":"No","MonthlyCharges":70.35,"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"}`
Runtime Editor	Workench
Enable Spark	TRUE - Spark 2.4.8
Resource Profile	2 vCPU / 4 GiB

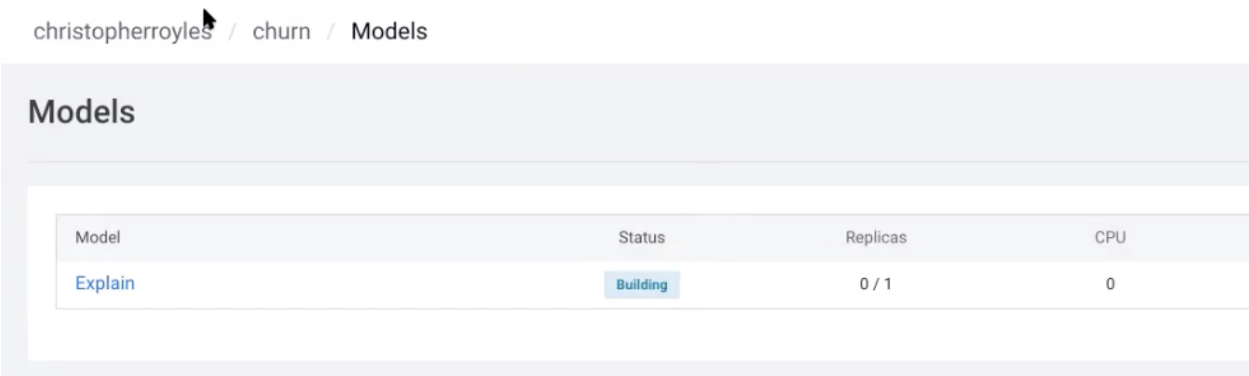

mmehra / telco / Models / Deploy Model
Deploy a Model
Deployment Template
Deploy model from code
Deploy registered model
General
Name *
apac100_telco_churn_model
Description *
Deploying the telco churn model
Enable Authentication
Enforces model API requests to be authenticated with an API key.
Build
File *
5_model_serve_explainer.py
Function *
explain
Example Input
{"StreamingTV":"No","MonthlyCharges":70.35,"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"}
Example Output
{ "result": "value" }
Runtime
Editor
Kernel
Edition
Version
Workbench
Python 3.7
Standard
2023.05
Configure additional runtime options in Project Settings.
Enable Spark
Spark 2.4.8 - CDE 1.18.1 - HOTFIX-4
Runtime Image - docker.repository.cloudera.com/cloudera/cdsw/ml-runtime-workbench-python3.7-standard:2023.05.2-b7
Comment
Enter comment for this build
Deployment
Resource Profile
2 vCPU / 4 GiB Memory



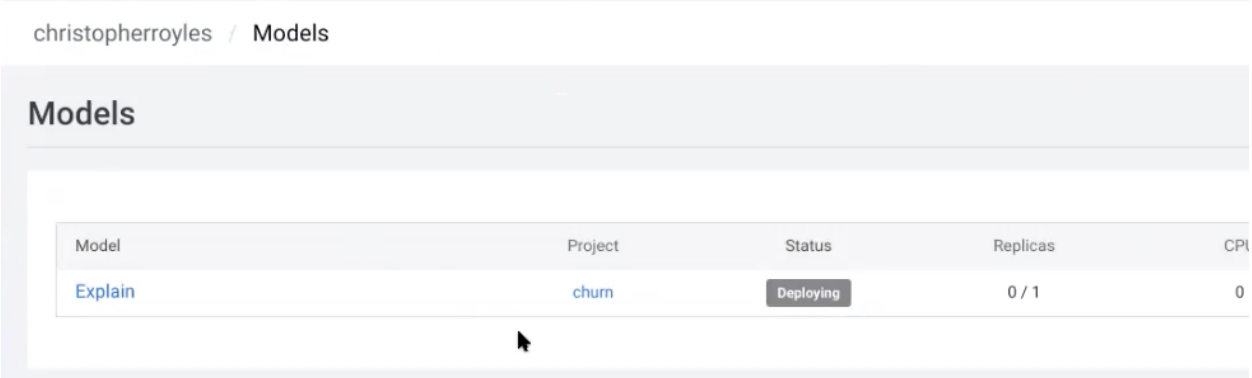
At this point you can deploy the model by clicking ***Deploy model***



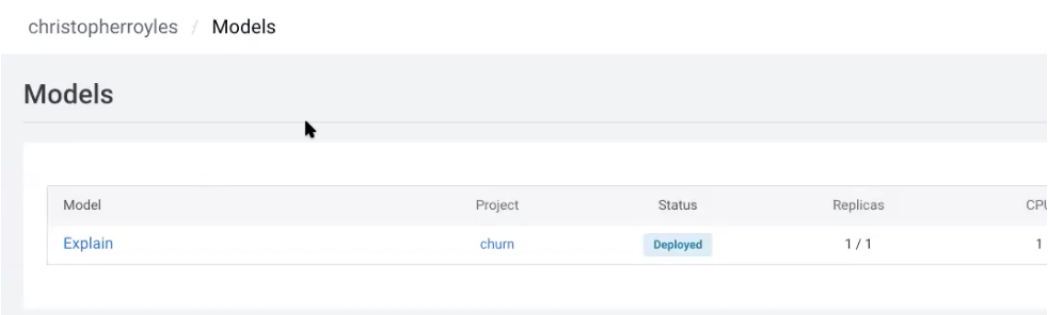
The status will go thru the life-cycle of the container *Pending -> Building*



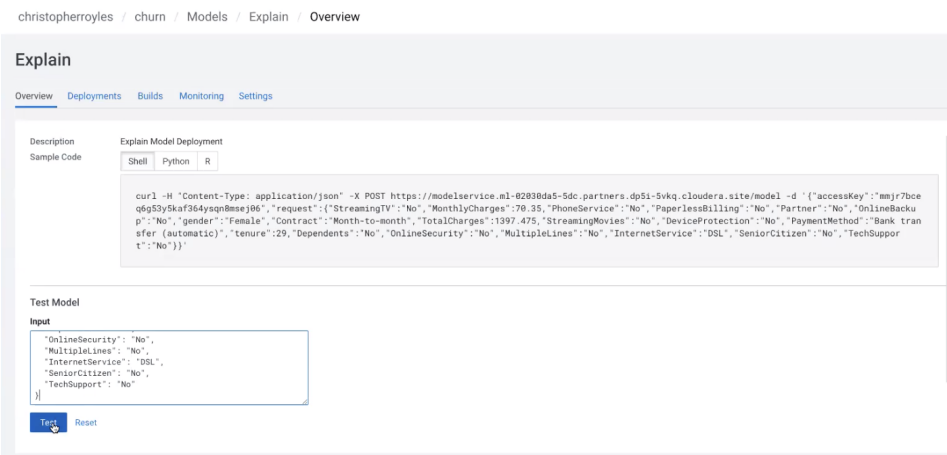
Building -> Deploying



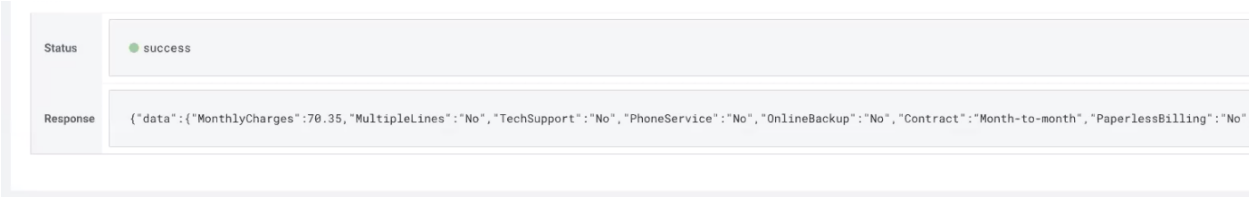
And finally *Deployed*



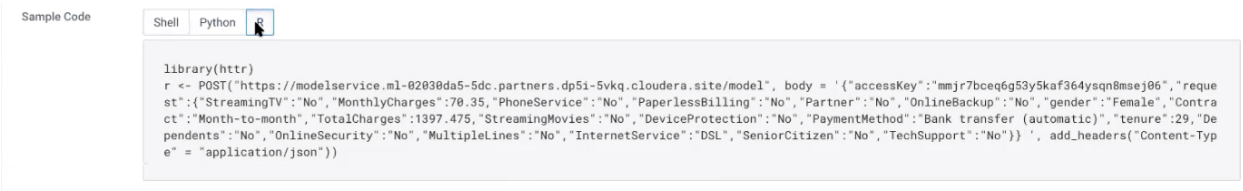
Now you can click on the model name and test it ...



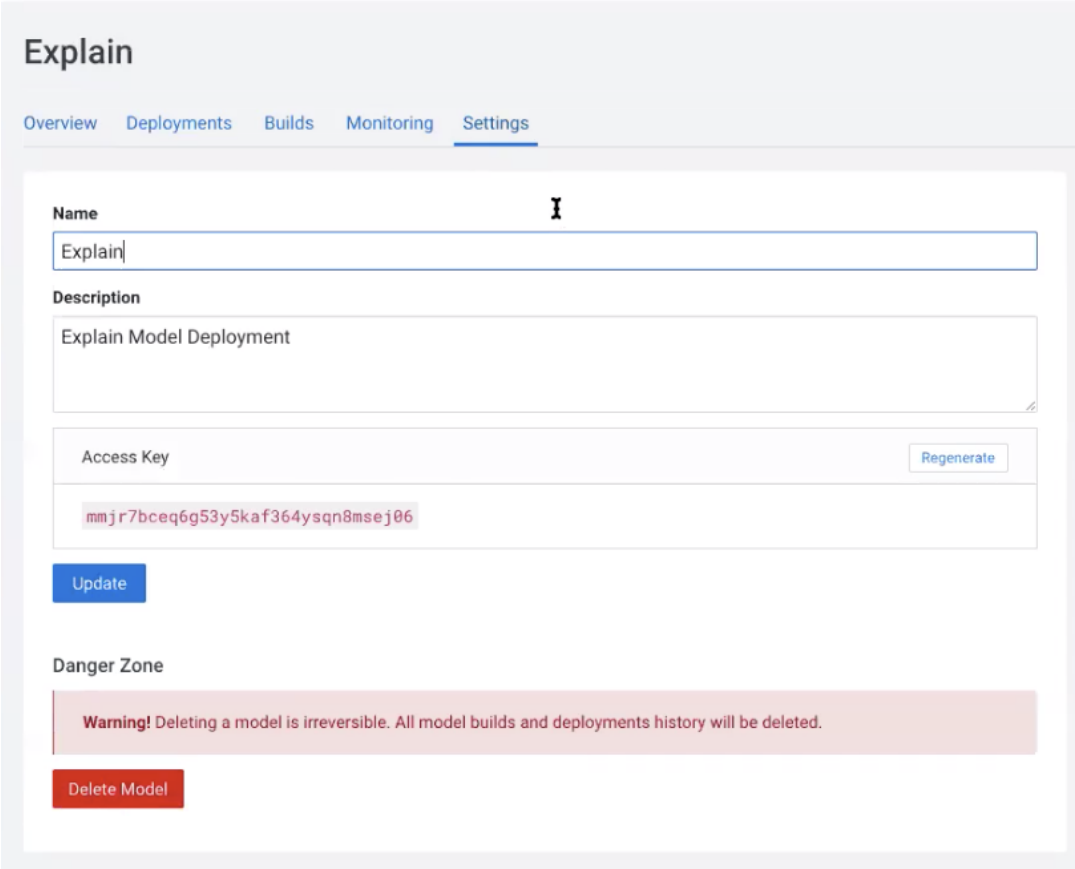
… you should get the following:



If you want to call the model from external services, the sample codes for invoking this REST endpoint are provided in Shell, Python and R

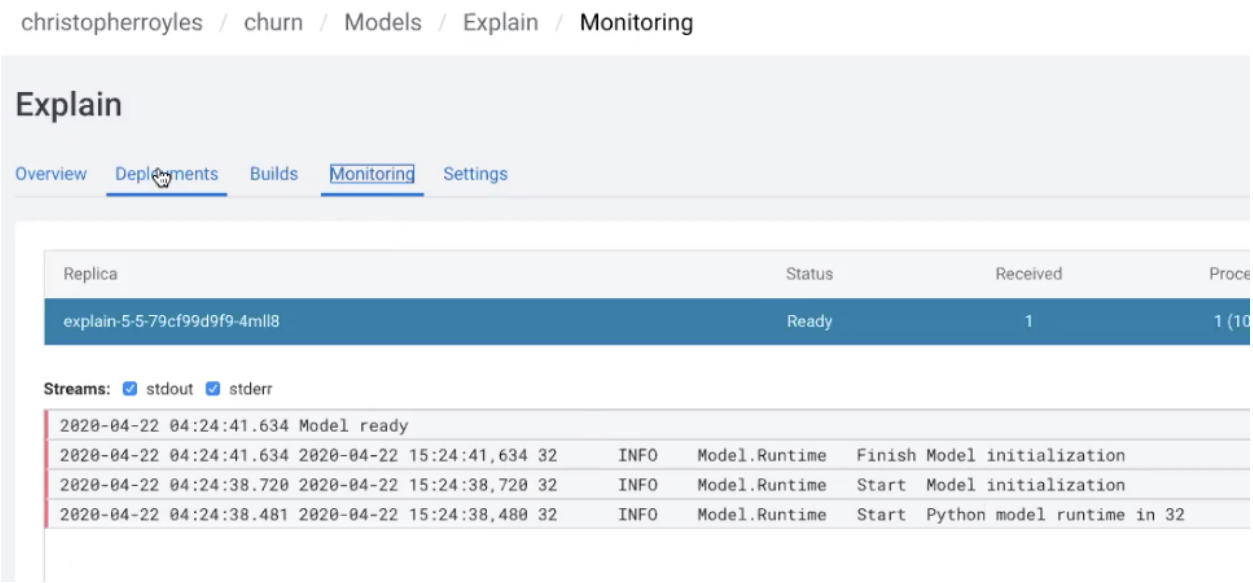


As you can see the sample codes also provide an *accesskey* for invoking this model, so only the services that provide a correct *accesskey* can invoke it. You can manage the access key by going to ***Settings***:



Copy the Access Key for the next lab

The model can be monitored going to the ***Monitoring*** tab

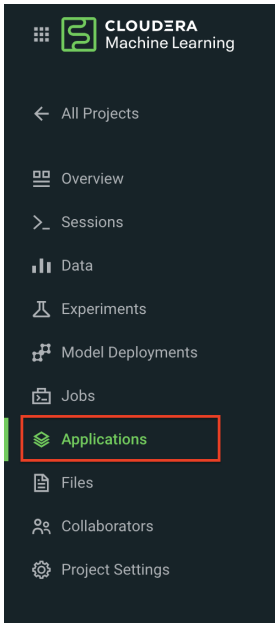


This concludes this lab.

Lab 5 - Application Deploy

In this lab, you will create an application that embeds the model deployed in the previous lab, allowing business users, end-users that are not Data Scientists to interact and to get insight about the context of these analyses.

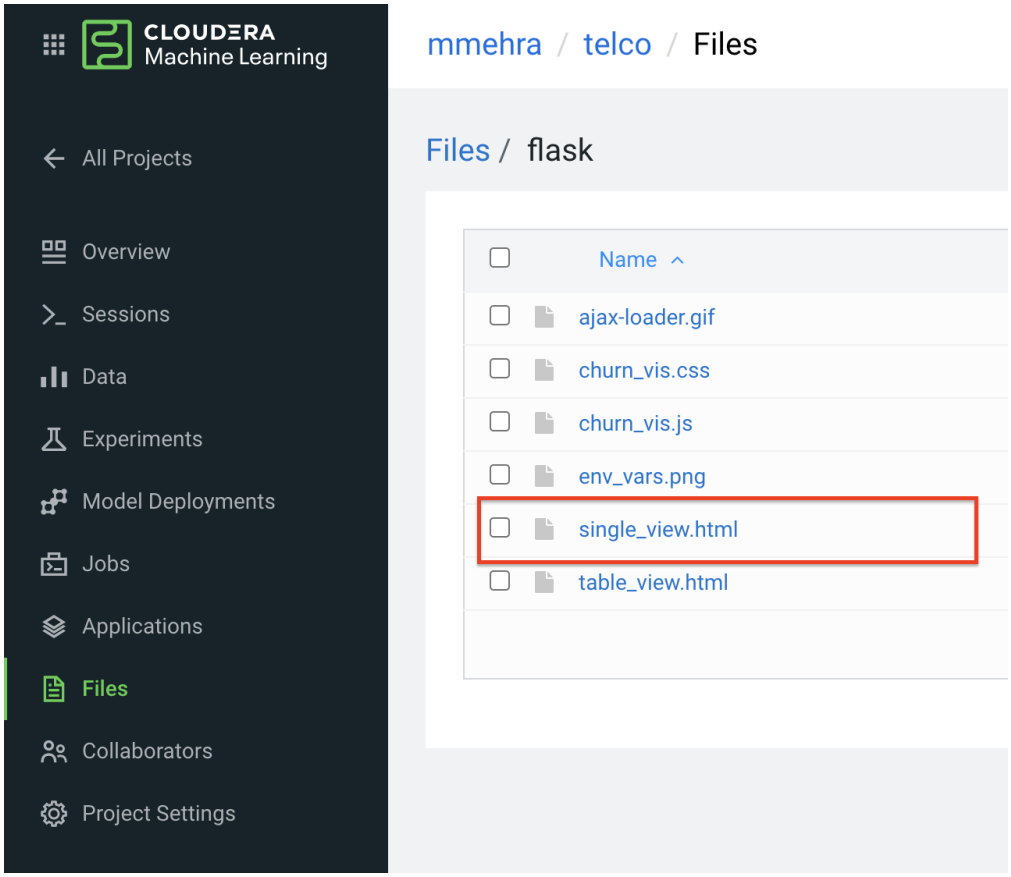
In the left tab, go to ***Applications***



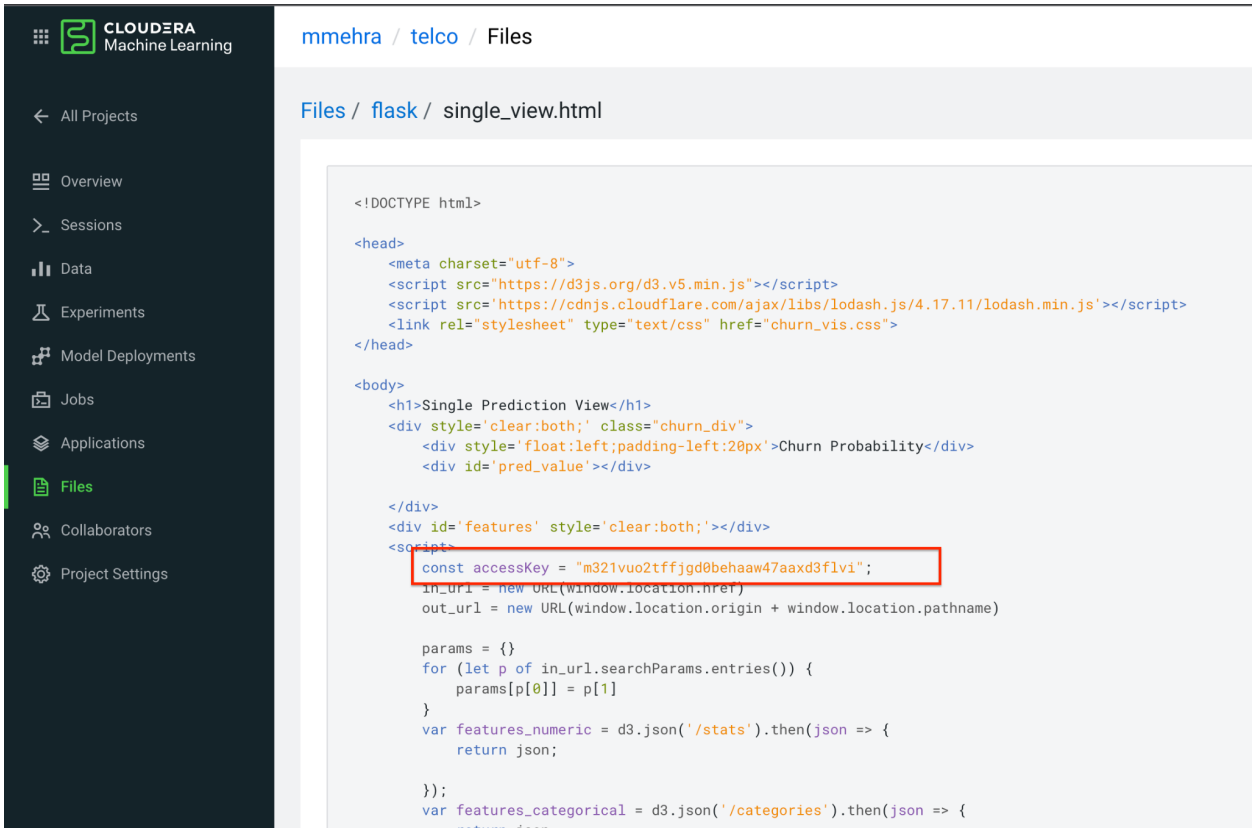
And as you can see we do not have any applications available yet. Go back to Files, here you can see that we provide the code on an application that is a Flask application as front-end

And the back-end is provided by the 6_application.py code

And now update the access key in the **single_view.html** (this file is under the **flask** folder)



Click on **single_view.html** and you can see the accesskey that we need to substitute to the one we copied in the previous lab



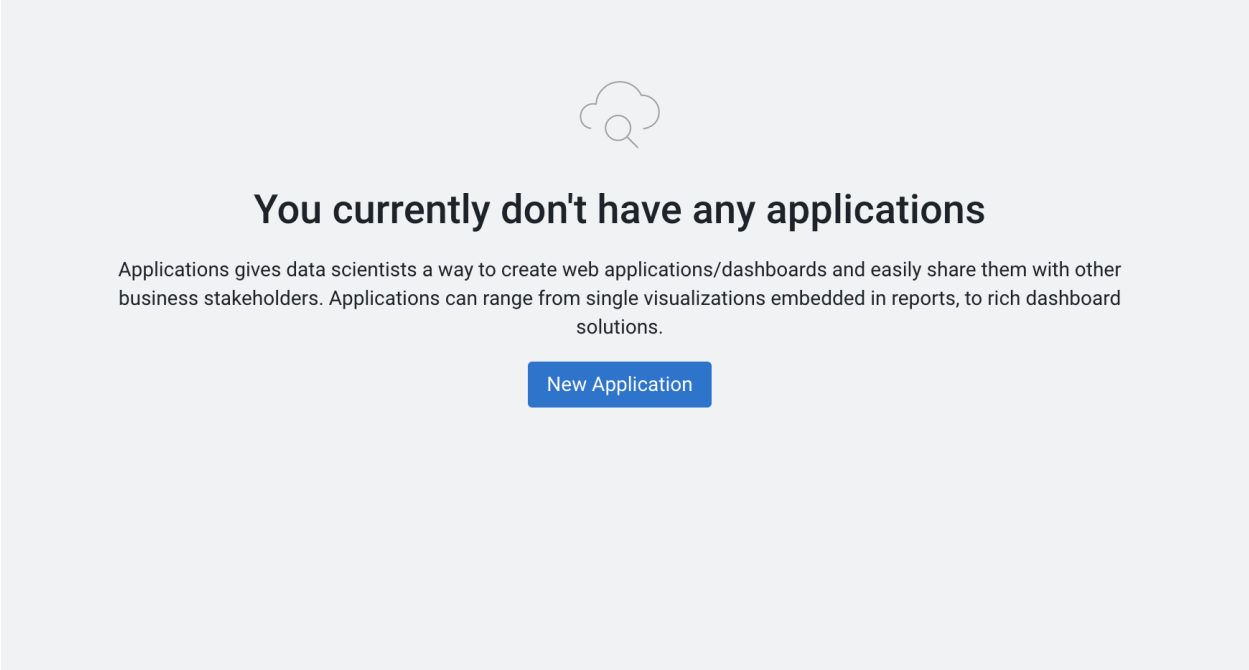
In order to edit this file, open it in a workbench (click Open in Session)

You do not need to launch a session. Just substitute the access key with yours and select **File > Save**.

```
    <div style='float:left;padding-left:20px'>Churn Probability</div>
    <div id='pred_value'></div>

</div>
<div id='features' style='clear:both;'></div>
<script>
    const accessKey = 'msj7n0gwhll55j1ouy8yv15cpeqa9re';
    in_url = new URL(window.location.href)
    out_url = new URL(window.location.origin + window.location.pathname)

    params = {}
    for (let p of in_url.searchParams.entries()) {
```

This is going to be used to call our deployed model when the end-user of this application will interact with it.

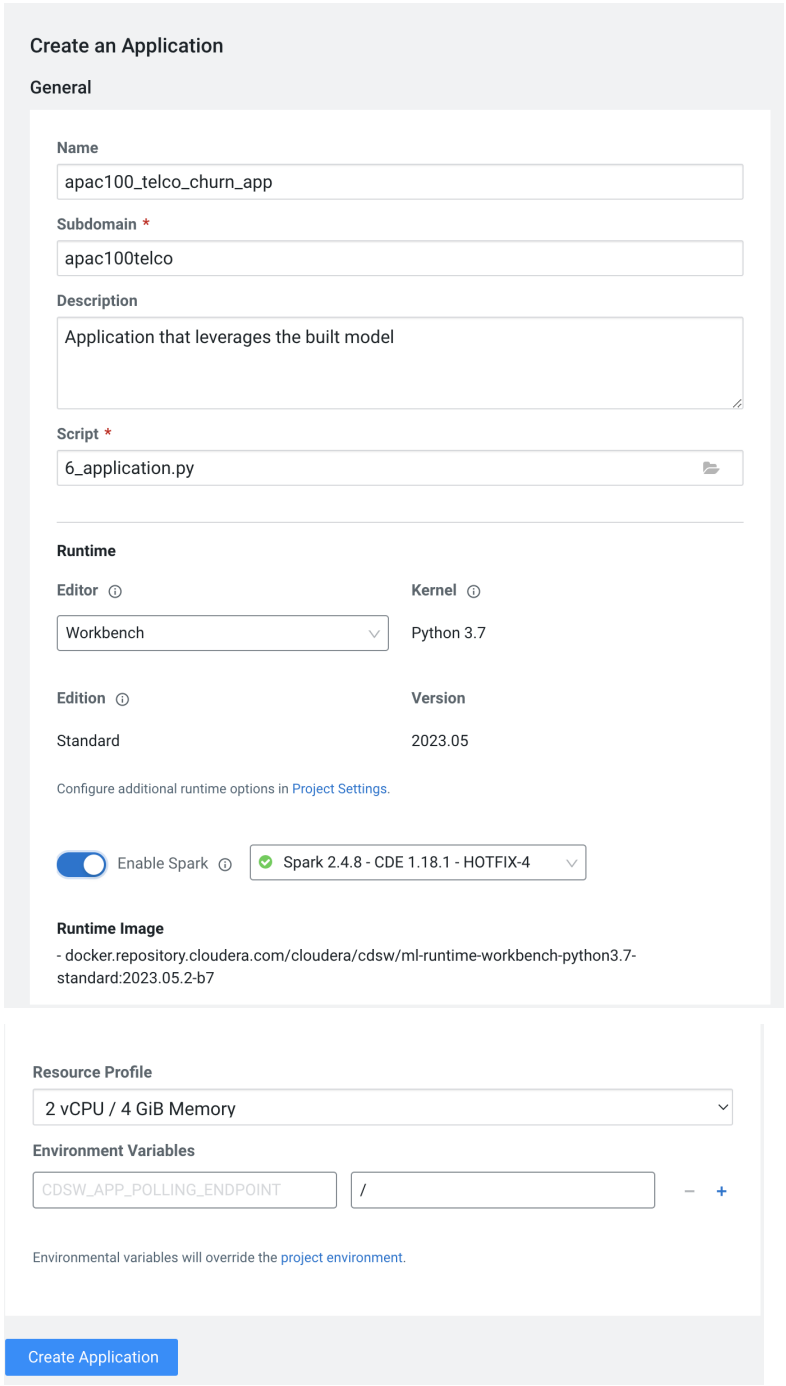
Now go back to Application, and click the New Application button.



Provide the following details.

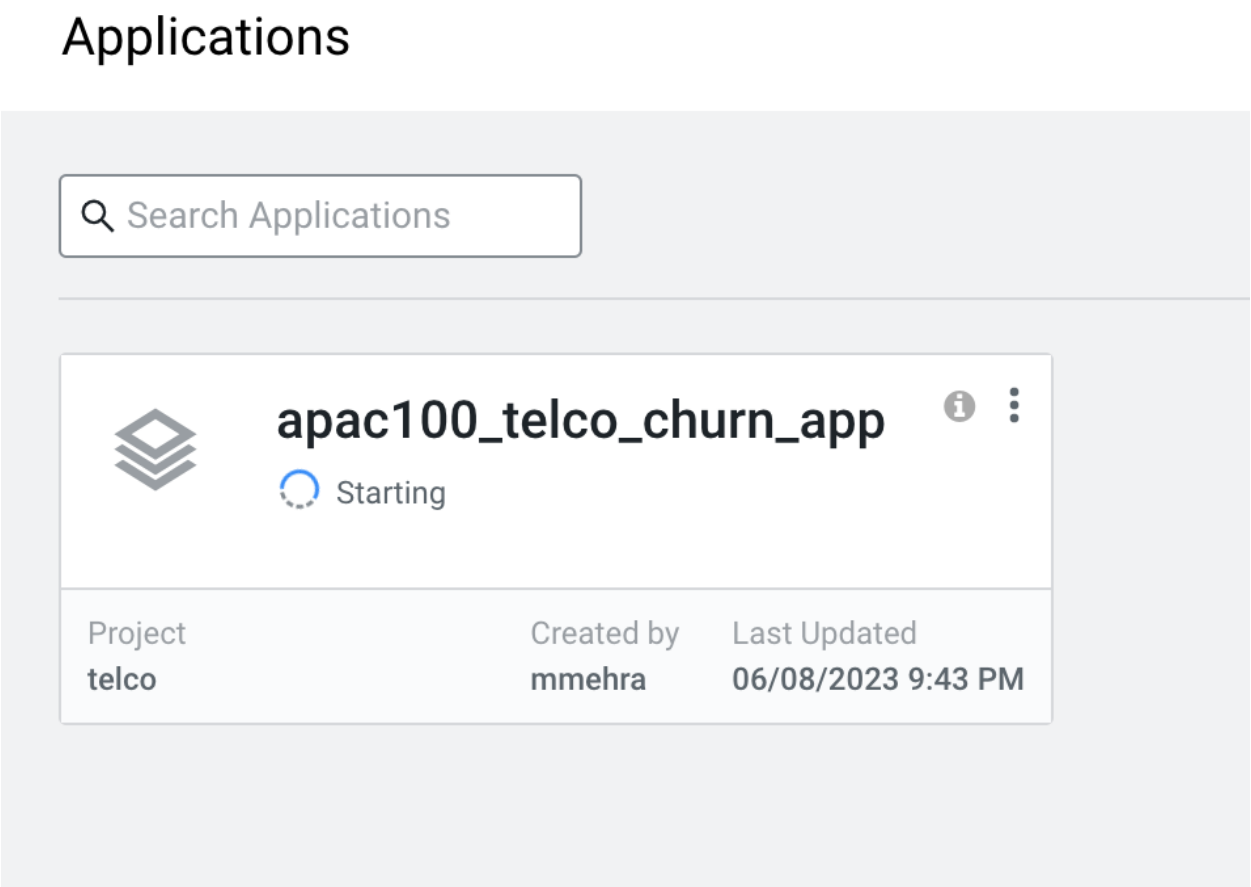
Name	<username>_telco_churn_app
Subdomain	<username>telco
Description	Application that leverages the built model
Script	6_application.py

Runtime - Editor	Workbench
Enable Spark	TRUE - Spark 2.4.8
Resource Profile	2 vCPU / 4 GiB

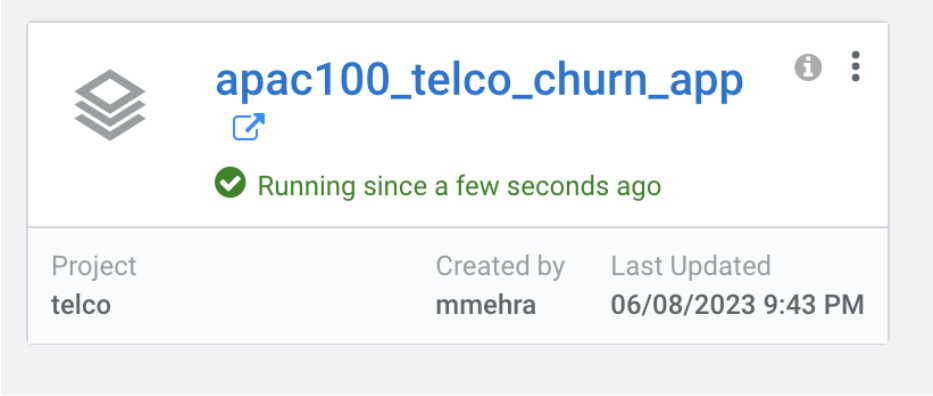


And then click **Create Application**

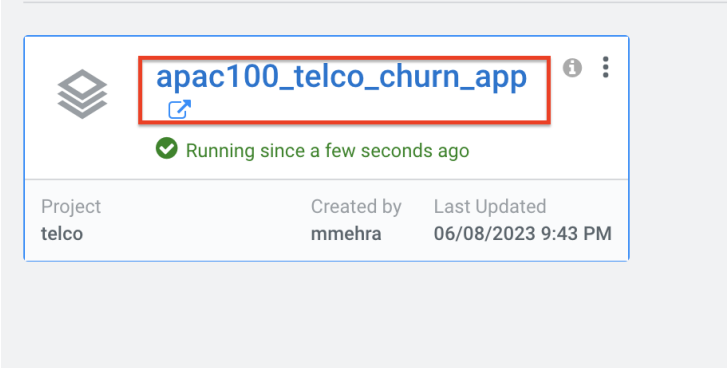
Then you should see the status **Starting** state



After a while it will change to **Running**

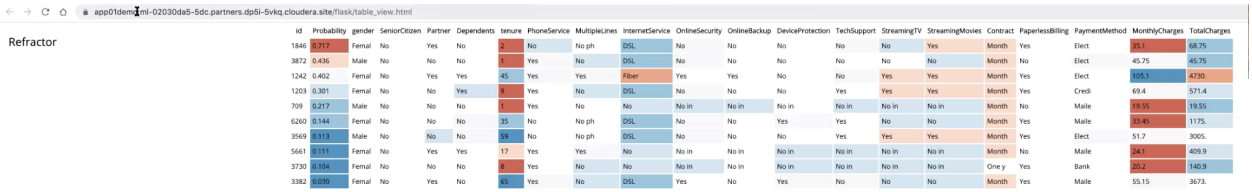


Click now in your newly created application

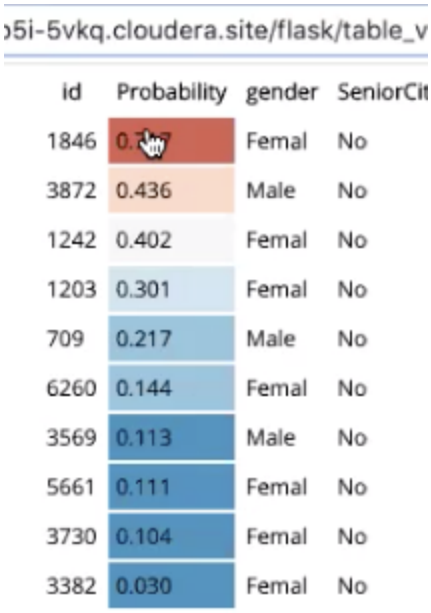


You can see the subdomain we have specified before as a prefix of your application url.

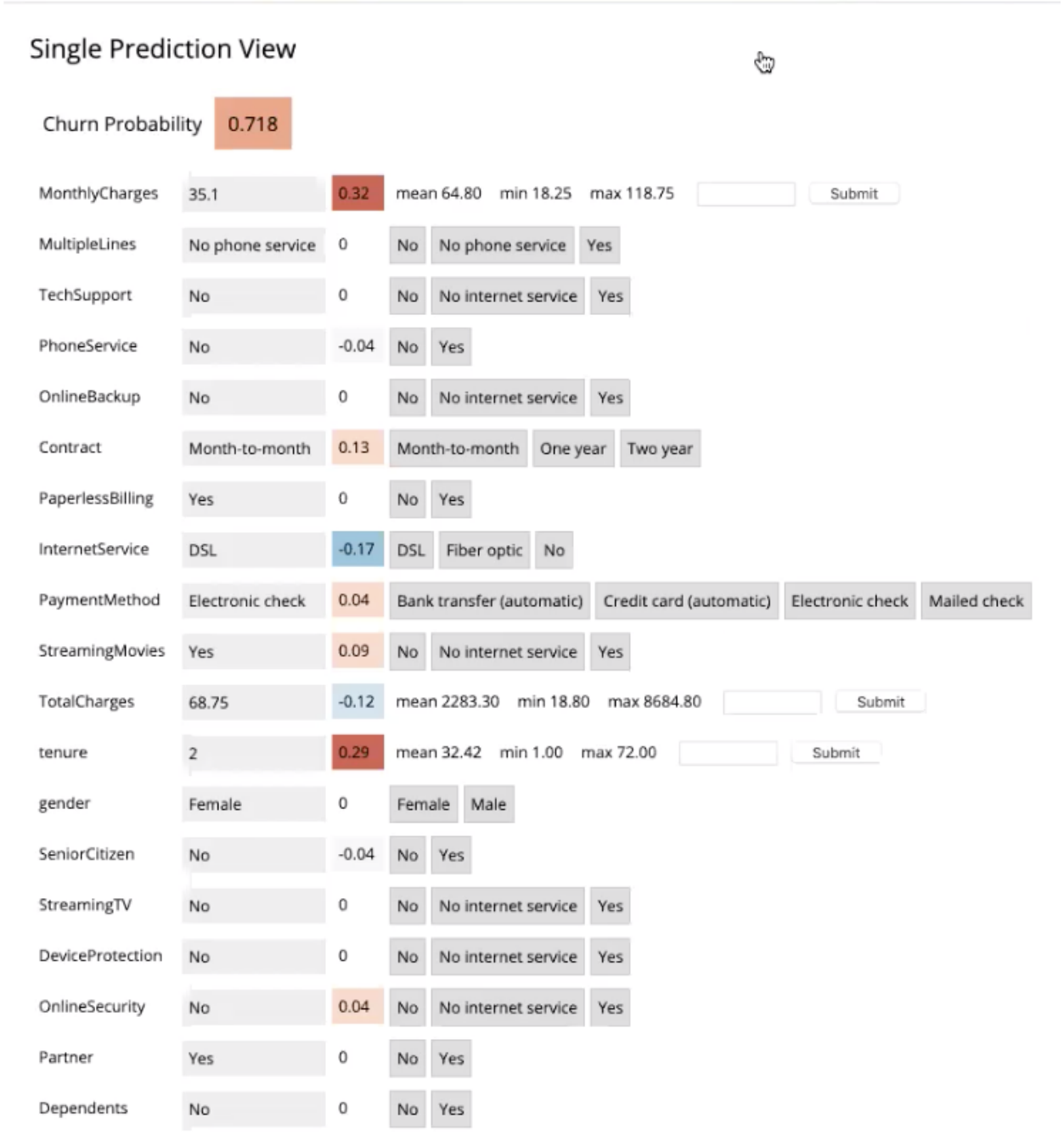
Once the application is loaded



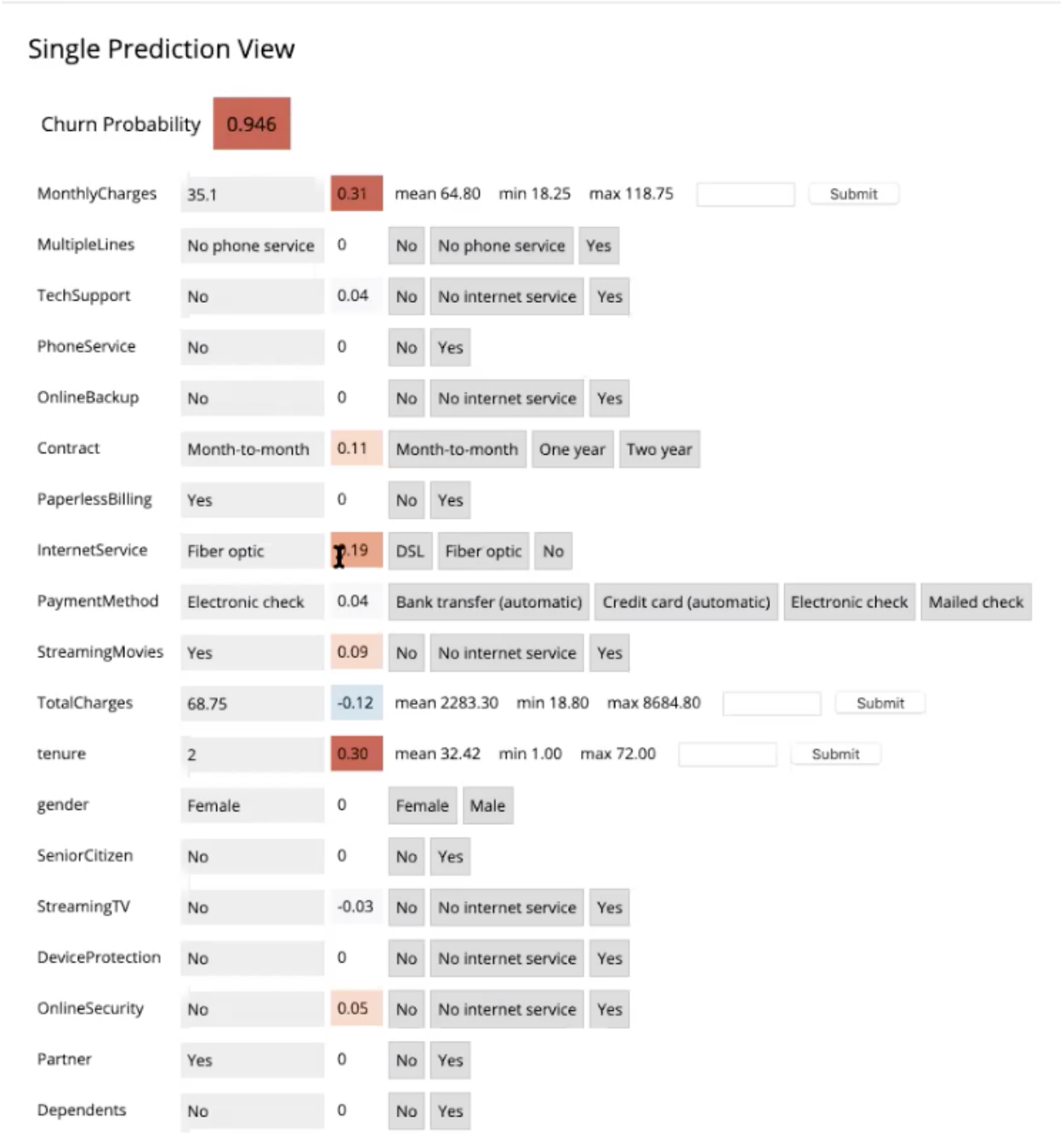
Click on one of the item in the Probability column



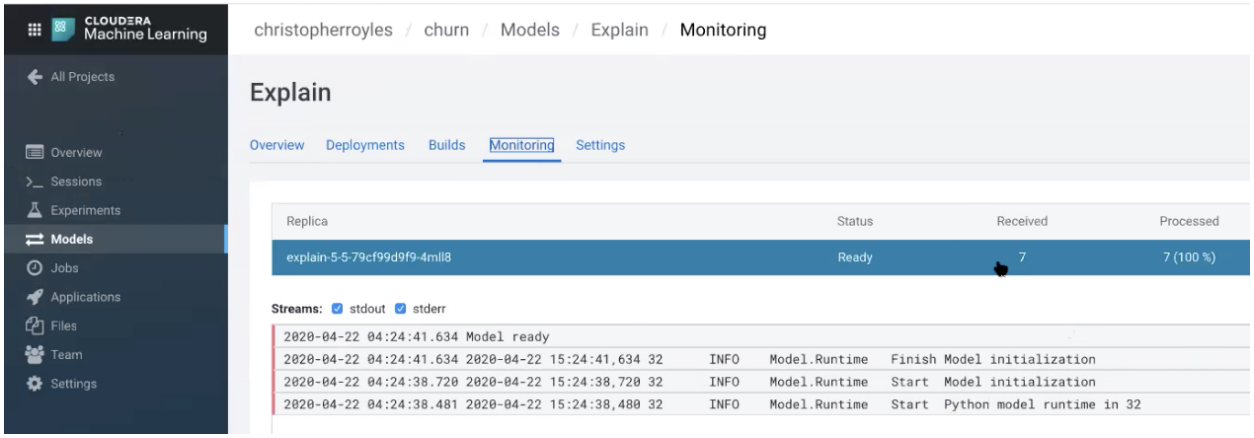
To get the detailed view



If you change some of the values, that will also change the churn probability by calling the model we have deployed in the previous lab.



Everytime you click and change a value, the application will call our model. To check this go back to your model deployed, click Monitoring bd you should see that the Receive value is increased



This concludes the lab.