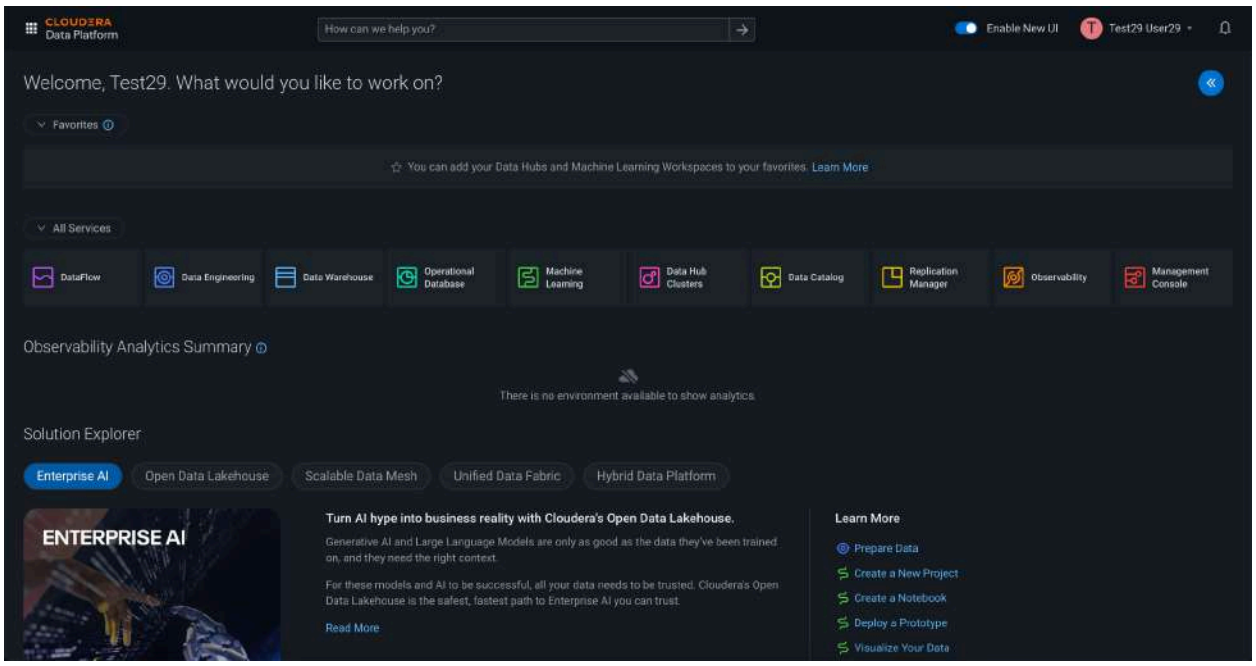
Data Lifecycle on CDP Public Cloud

Lab 004: Machine Learning Lab

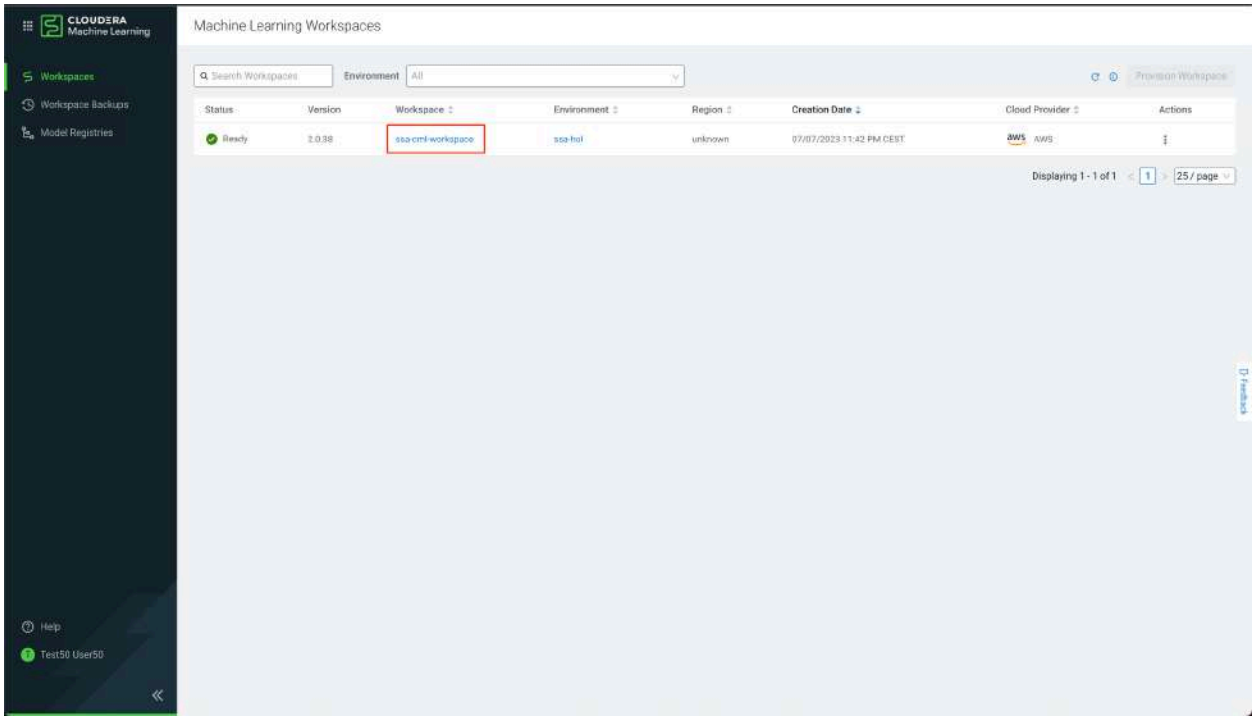
Goals:

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

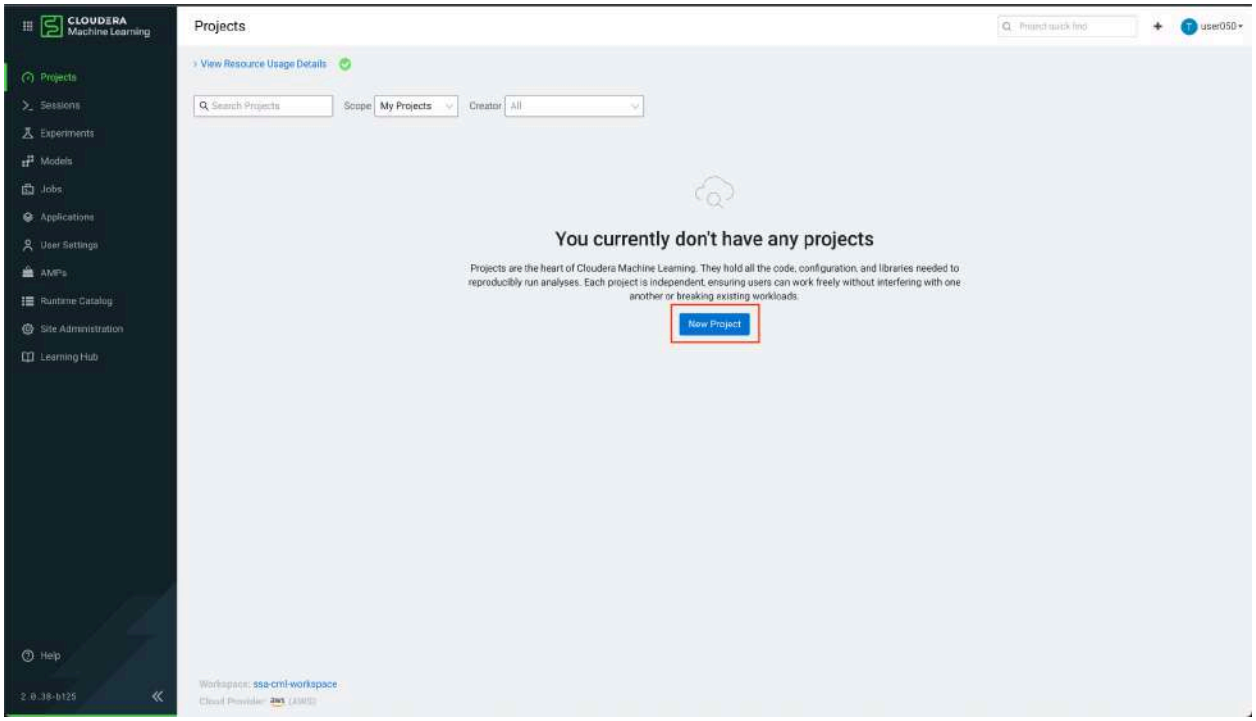
1. Click on Data Warehouse from CDP PC Home:



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

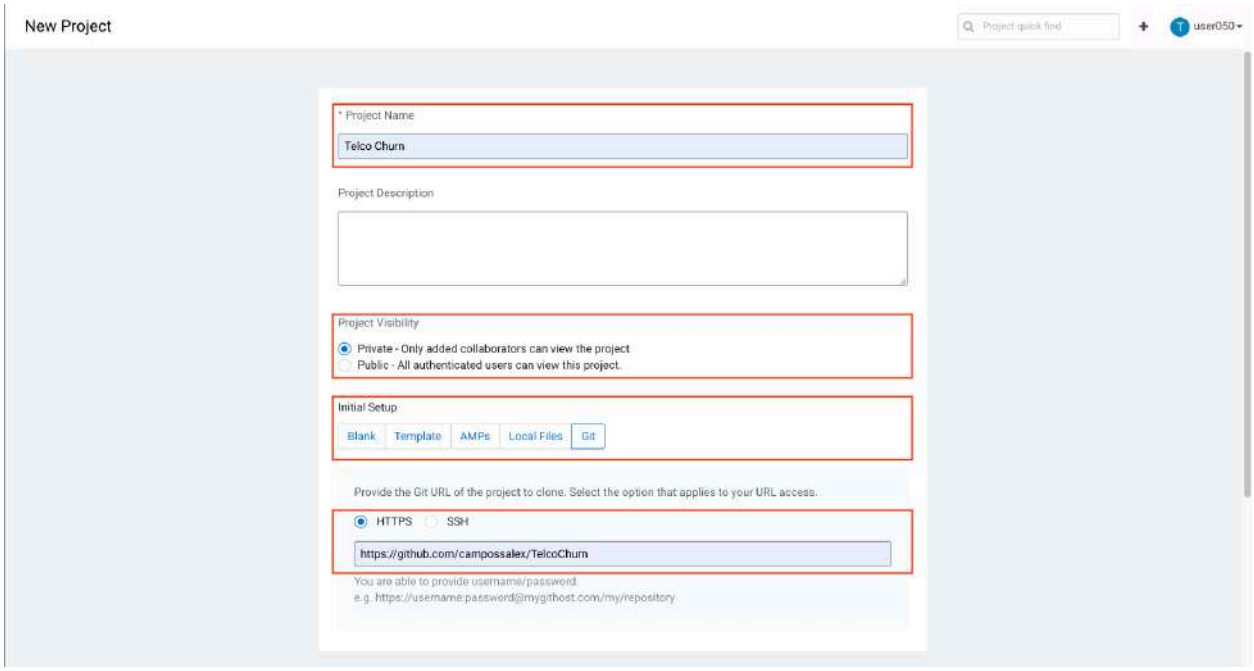


3. Once in the Workspace, you should see the following interface. Here are the projects you have created. It is time to create a new project. Click on the blue button **New Project**.

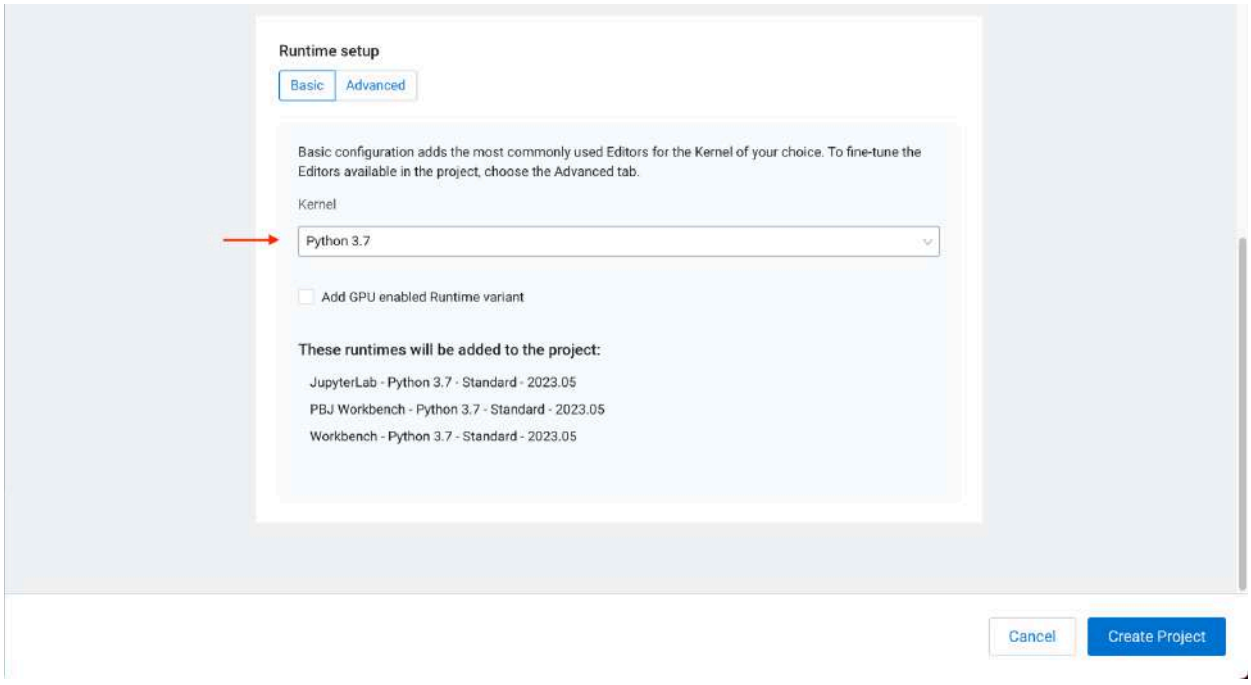


4. Enter the following information to create a new project:

Project Name: Telco Churn
Project Visibility: Private
Initial Setup, select Git
In the text field below HTTPS, enter the url of the git repo:
https://github.com/campossalex/TelcoChurn



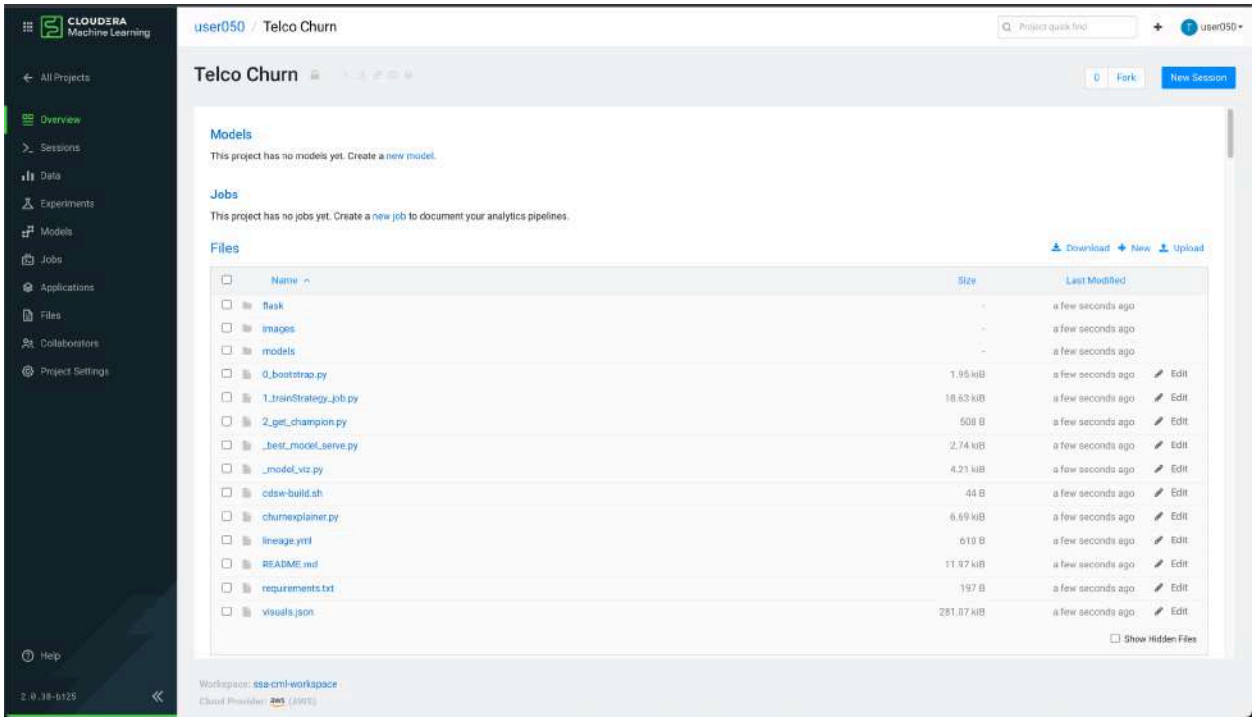
Make sure to select **Python 3.7** in the Kernel selector. Click the button **Create Project**



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

Models, deploy and manage models as REST APIs to serve predictions.
Jobs, automate and orchestrate the execution of batch analytics workloads
Files, assets that are part of the project, such as files, scripts and code

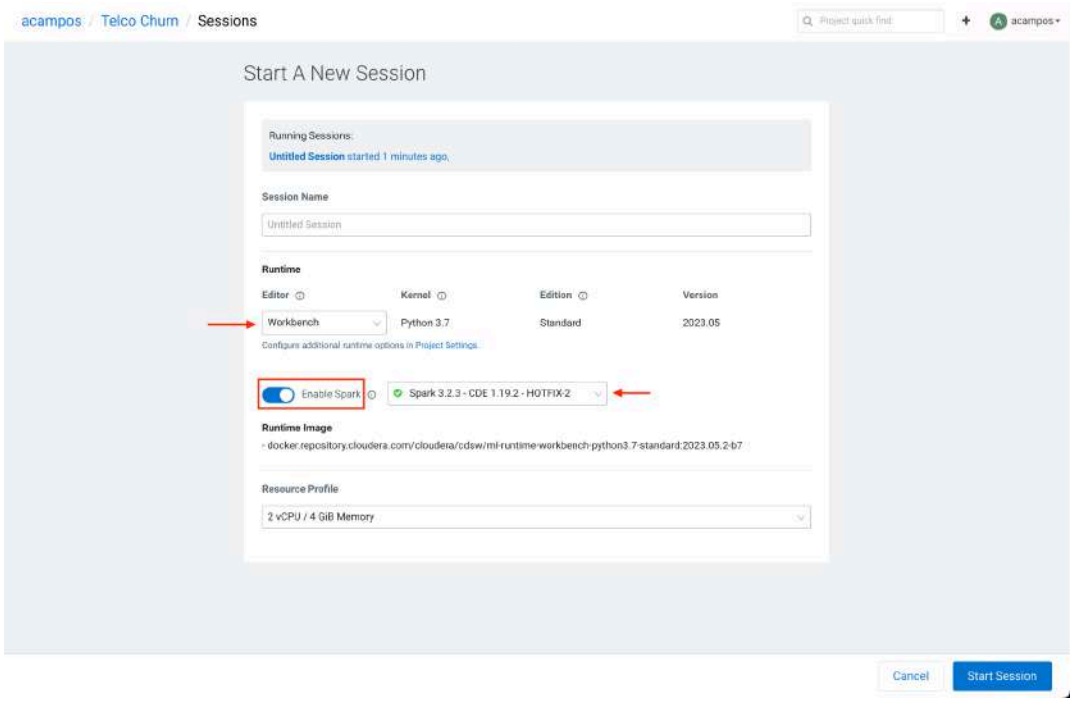
This Telco Churn project consists of running three scripts. The way of execution is through a session, which is the allocation of isolated compute resources for each user. For this, you must click on the blue button **New Session**, located in the upper right.



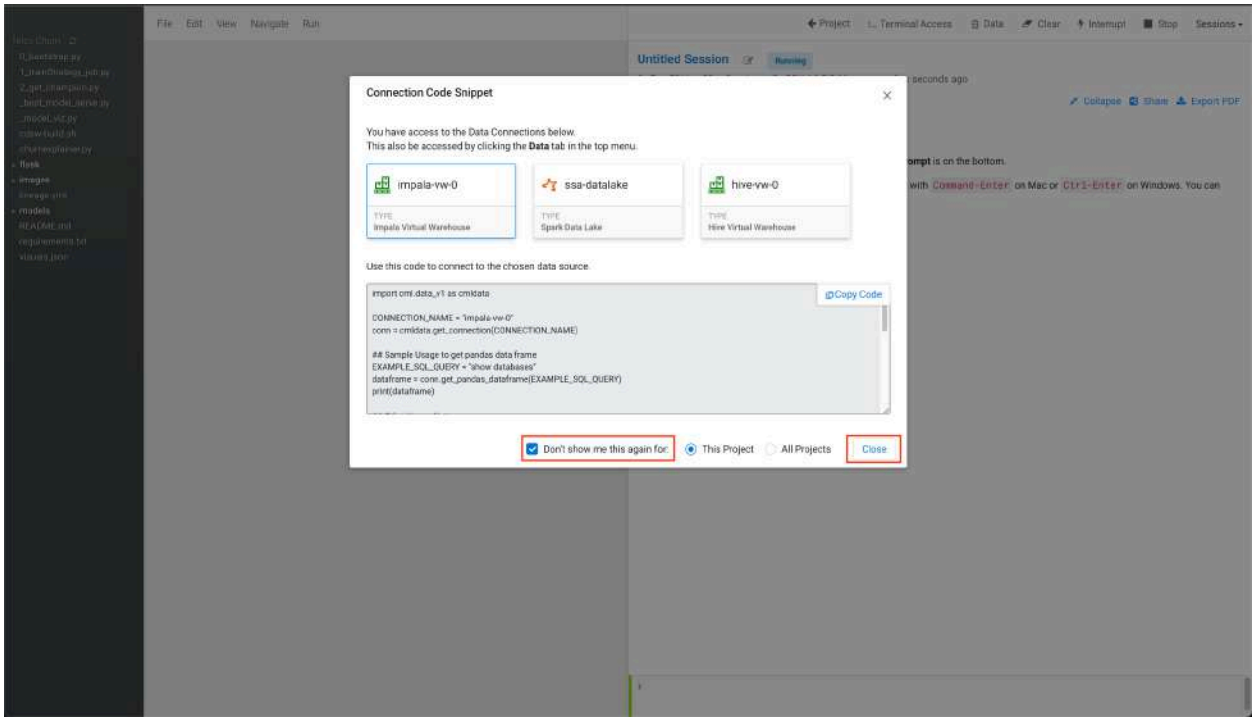
6. When starting a new session, make sure:

Select **Workbench** in the Editor selector.
Select **Python 3.7**, in the Spark version selector.
Enable **Spark**, marking the corresponding check.
Select **Spark 3.2.x**, in the Spark version selector.

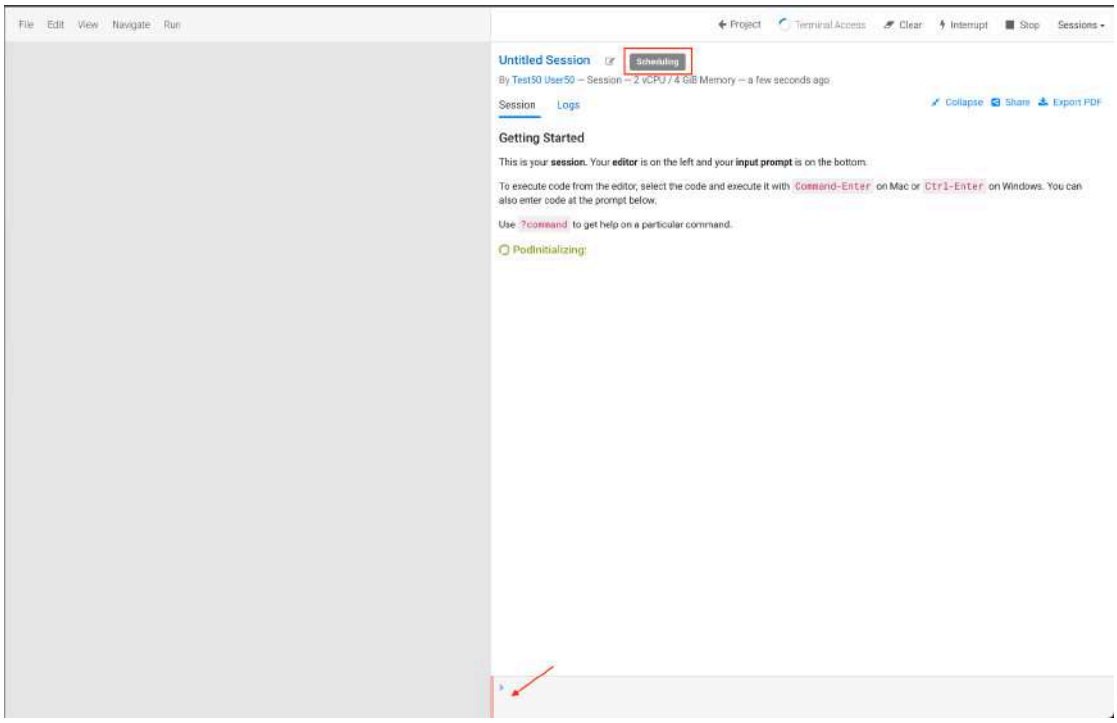
Click on the button **Start Session**



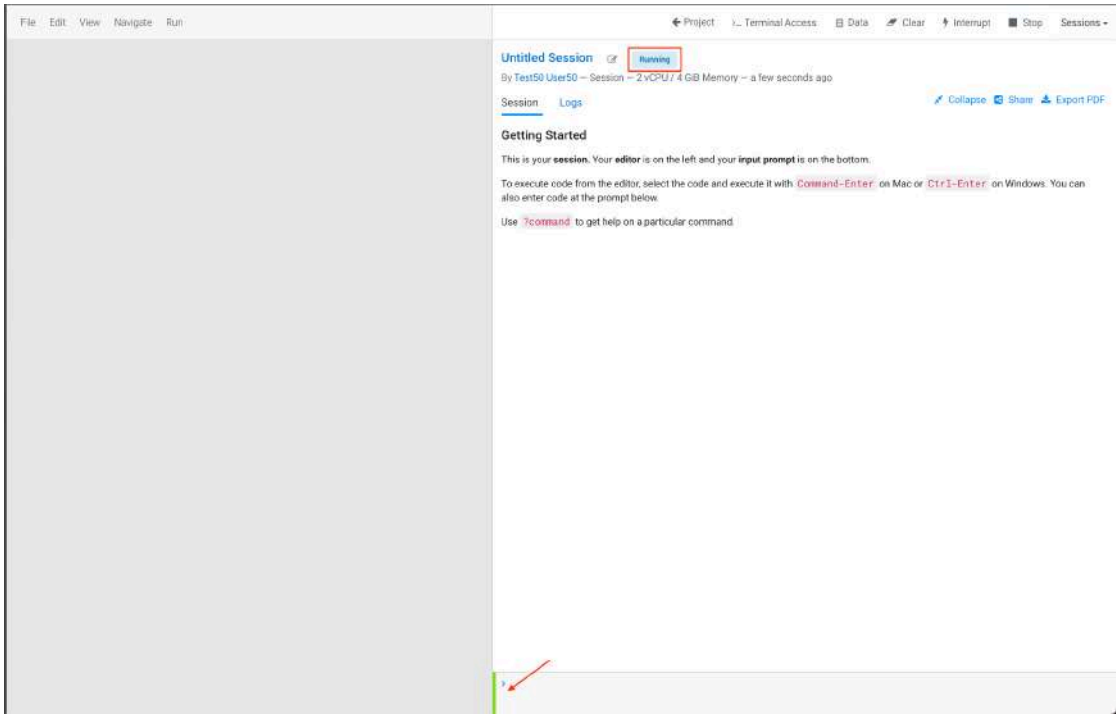
7. When you start a session for the first time, it will ask if you want to use a data connection. This project does not need this type of connection. mark the check of **Don't show me this again**, and then click the button **Close**, so this window will not appear anymore.



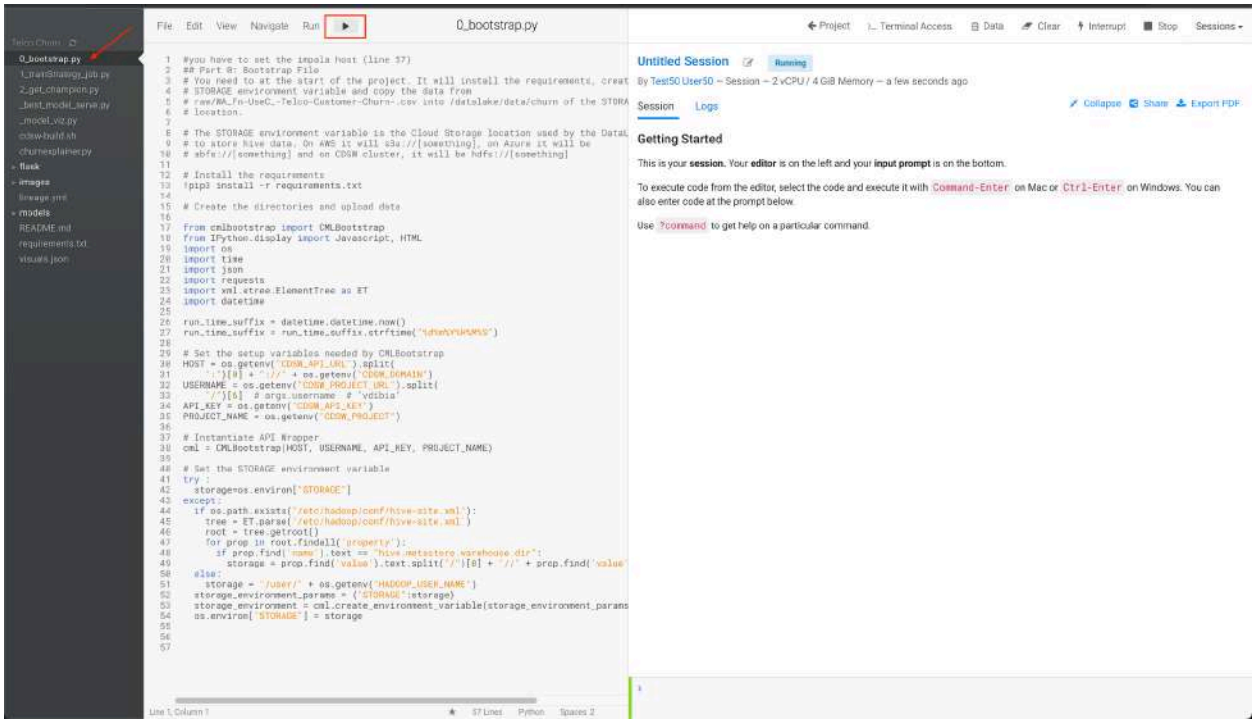
8. The editor/notebook located on the right side of the window will be in **Scheduling** status, and the bottom command bar flashing red. This means that CML is allocating computation for your session.



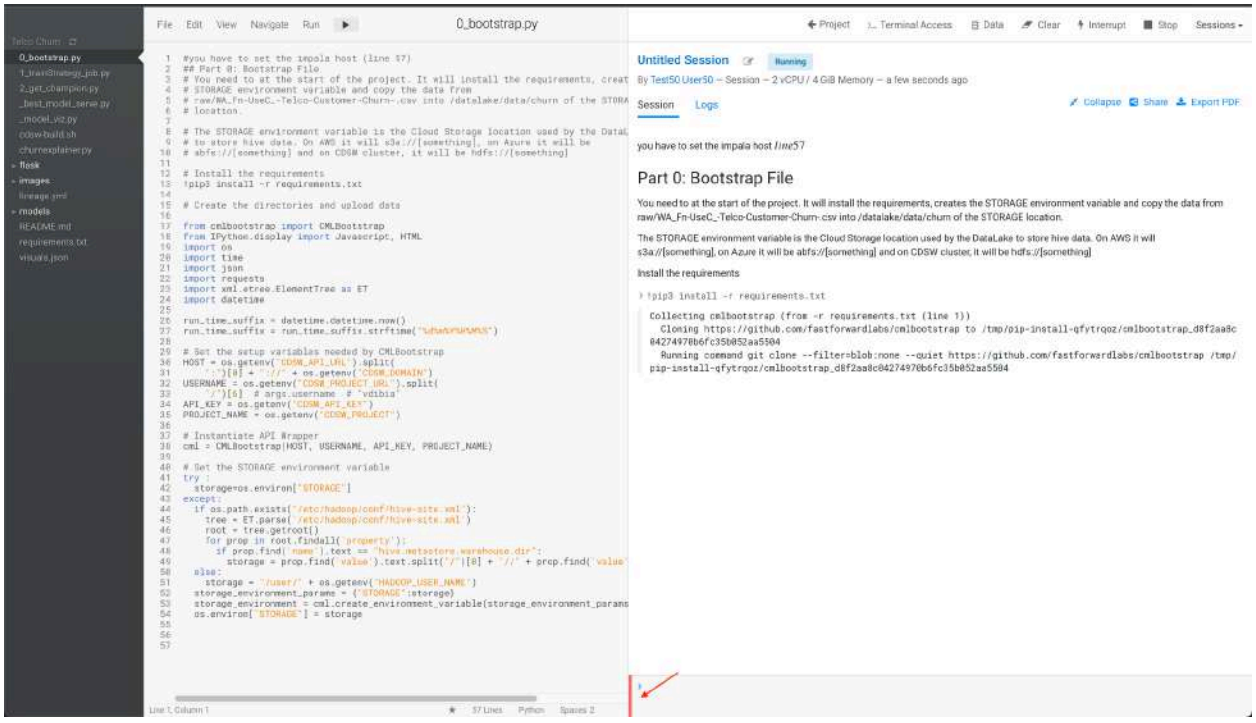
After a few seconds, the status changes to **Running**, and the command bar to green. This means that the session is ready to run code.



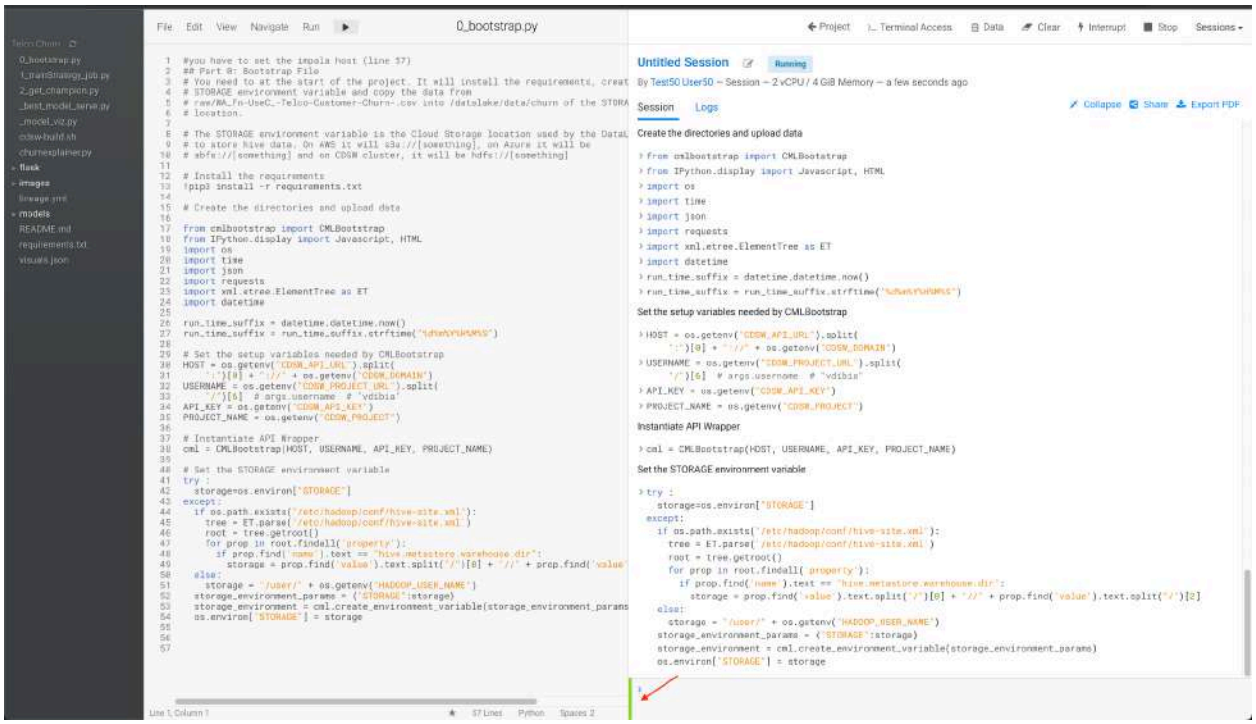
9. The first script/code to run is **0_bootstrap.py**. This Python code configures the libraries required for the project and integration with Lakehouse tables you populated before. Select (just one click) the file in the bar located on the left side of the interface, this will make the code appear in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



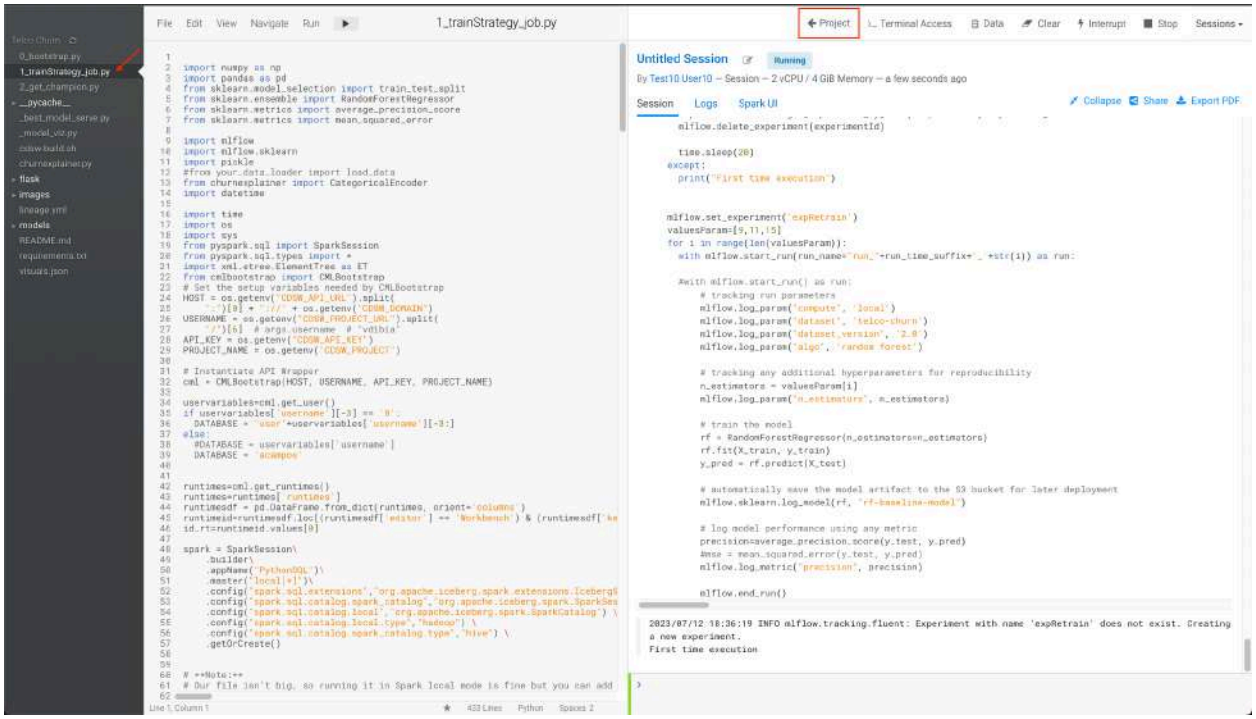
When you start execution, you will see code output on the right side of the interface, and the bottom command bar flashing red, indicating that it is busy.



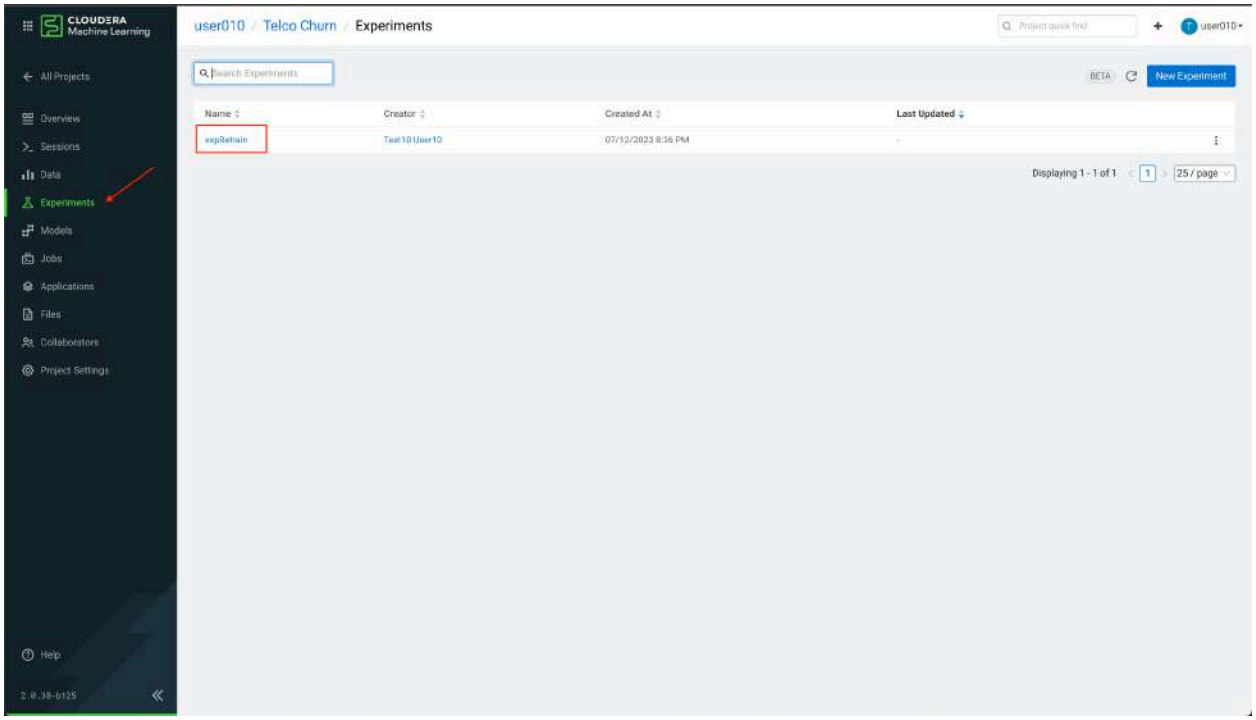
The green command bar indicates that the execution of the code has been finished. This bootstrap code takes 3-4 minutes to run.



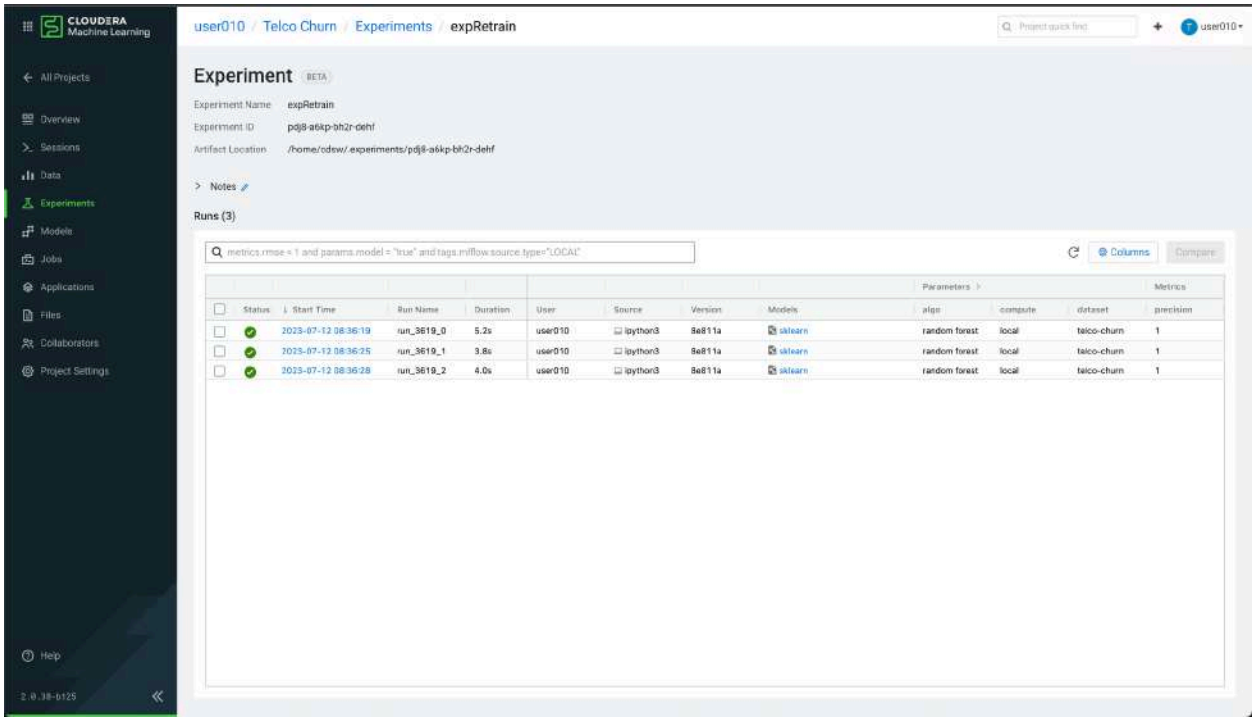
10. The second script/code to run is **1_trainStrategy_job.py**. This Python code will create the Experiment to run the model with three different hyper parameters and records the precision. Select (just one click) the file in the bar located on the left side of the interface, this will make the code 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.



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

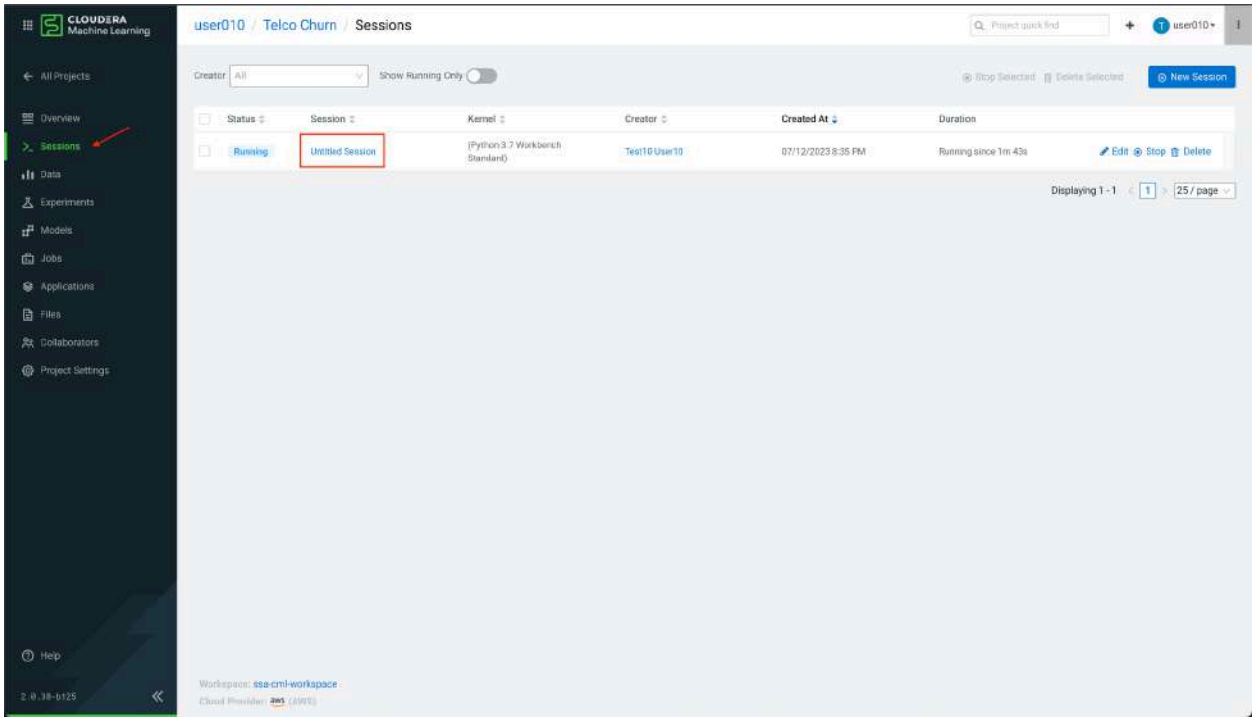


12. On this screen you will see the three runs of this experiment. Look at the last column, where **precision** attribute displays. This is the precision that each hyper parameter is delivering.

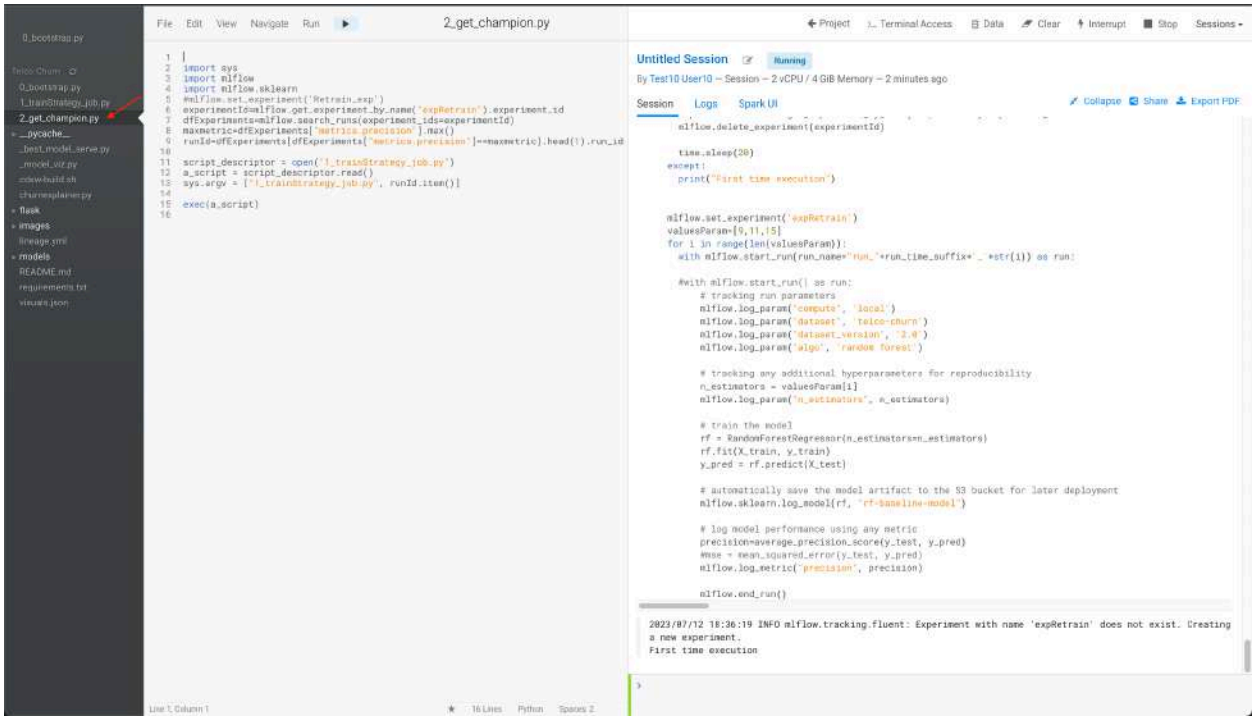


13. Let's go back to the session to run the last code. Since sessions run in Kubernetes containers, it's very easy to get back to where we were. Click on the option **Sessions** from the

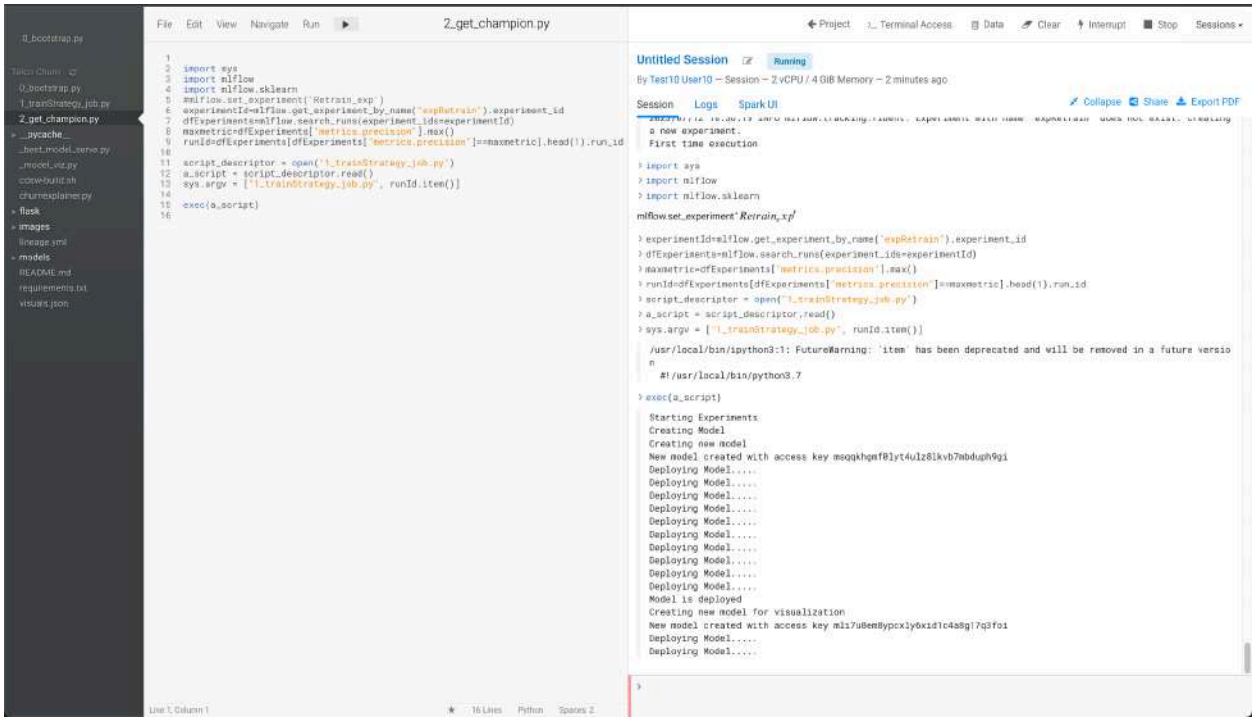
left menu, and later in the only session that will appear in the list. If you didn't name your session when you started it (step 6), it should be called *Untitled Session*.



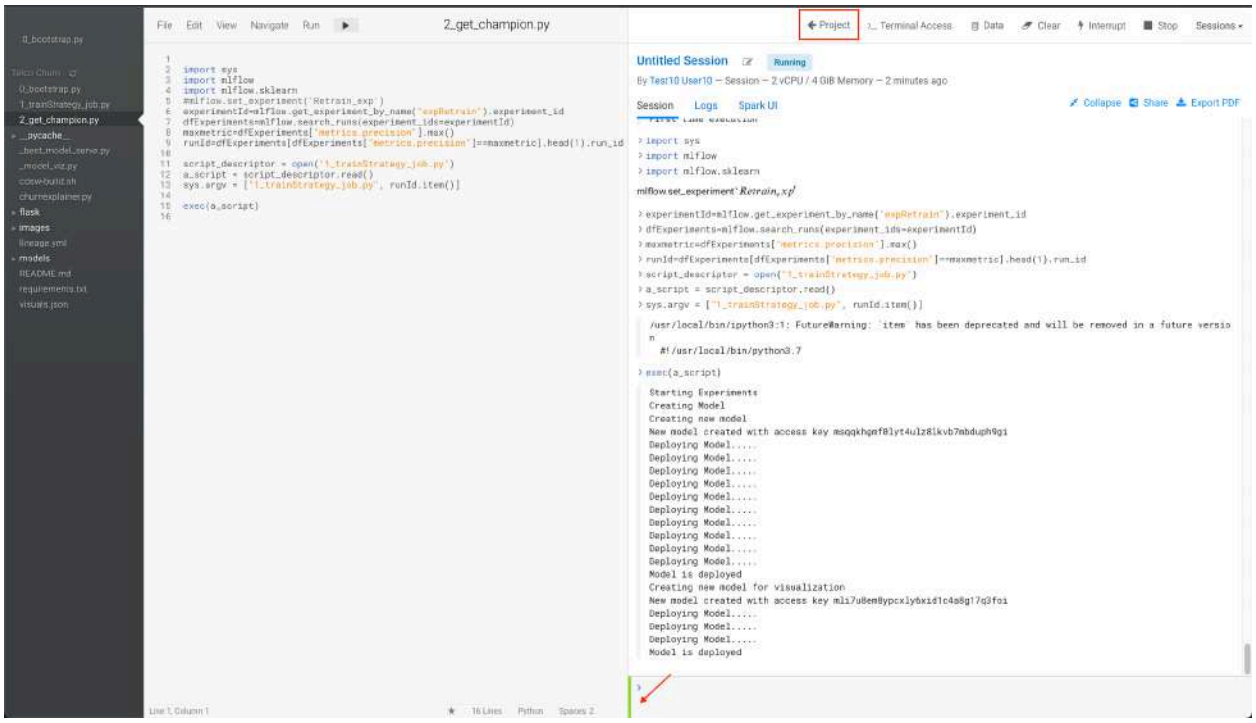
14. The third and last script/code to run is **2_get_champion.py**. This Python code takes the hyper parameter of the execution of the Experiment with better precision and deploys two Models as REST API, one to be integrated in Data Visualization and another for unit use for calls. Select (just one click)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



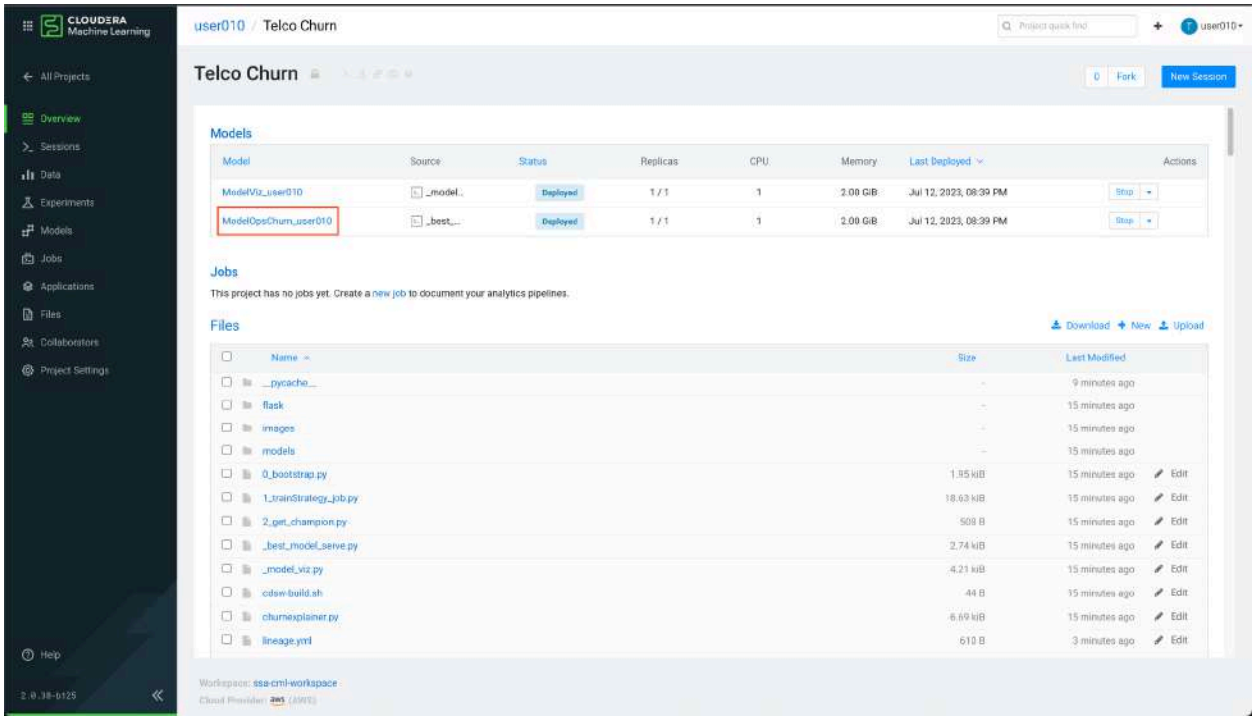
After a few seconds, you will see the following message “Deploying Model…” repeated several times, and the bottom command bar will be red.



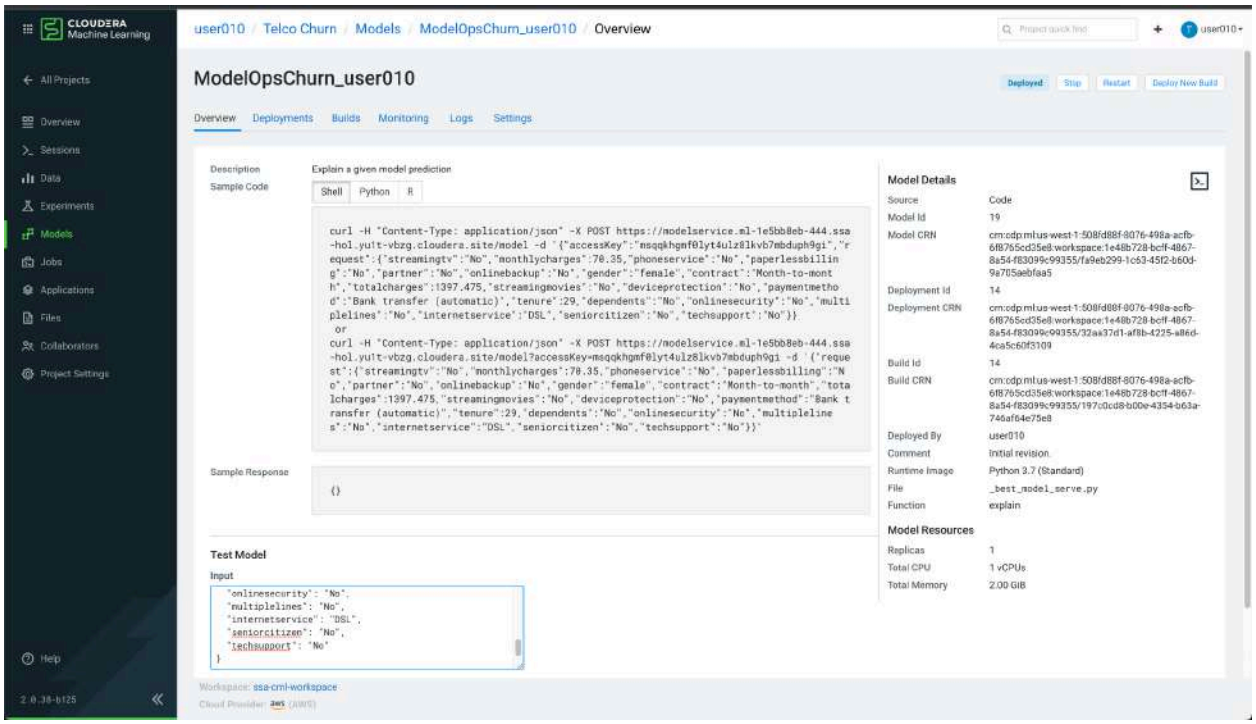
After about 2 minutes, the last message should be "Model is deployed", and the bar will be green. It means that the Deployment of the two Models is complete. Click on the button **Project**, located in the upper right bar of the session to return to the home page of the project.



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.