2 Hours to Data Innovation with Cloudera Machine Learning

November 15, 2023

2 Hours to Data Innovation with Cloudera Machine Learning

Objective

In this exercise we will implement an end-to-end machine learning workflow using Cloudera Machine Learning (CML), including:

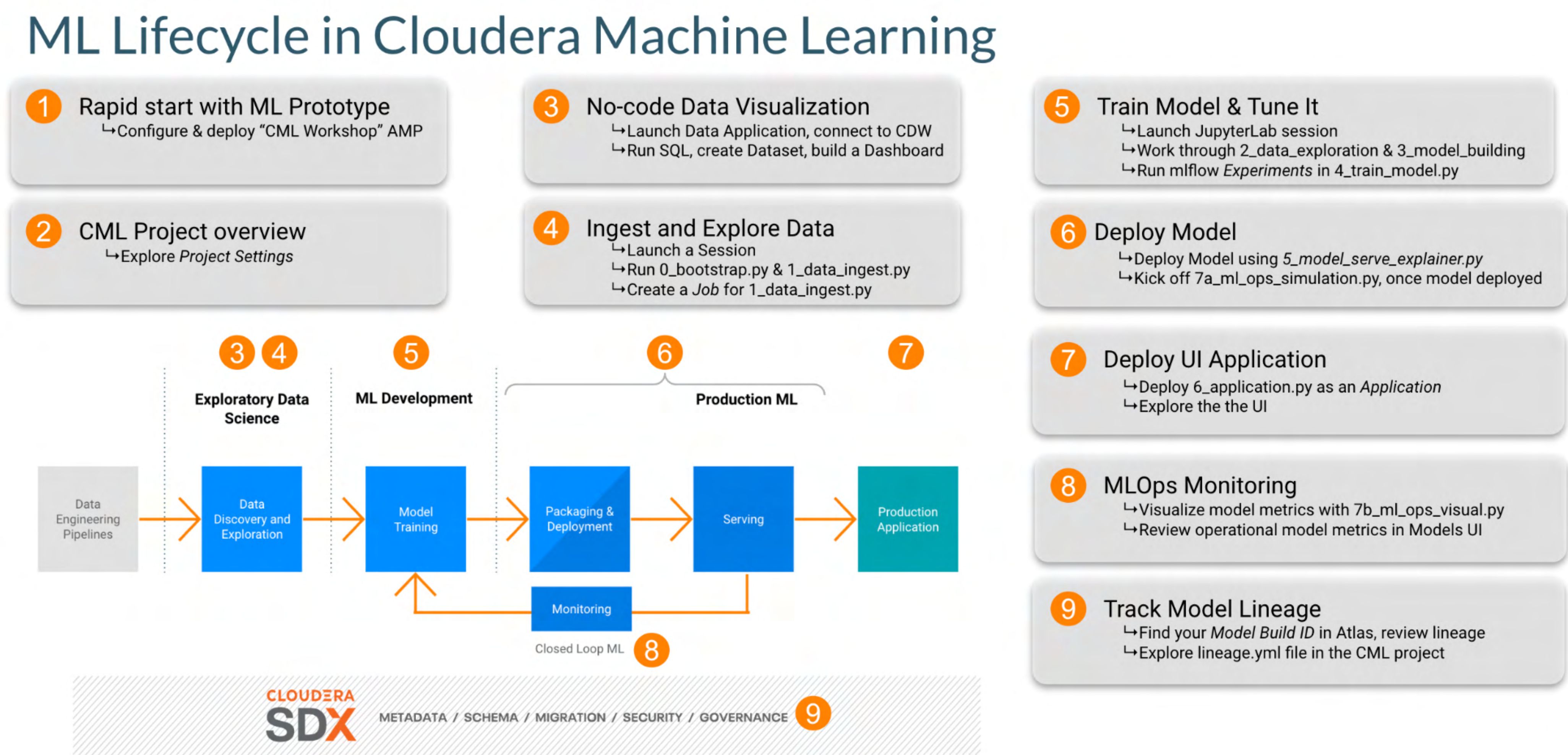
- Data ingest & Data exploration
- Model training and experimentation
- Model serving and model registry
- Business applications
- MLOps - model operations

Business Use Case

In this Hands On Lab you will create a model to predict customer churn and present model-driven insights to your stakeholders. You will use an interpretability technique to make your otherwise "black box model" explainable in an interactive dashboard. A mathematical explanation is beyond the scope of this lab but if you are interested in learning more we recommend the Fast Forward Labs Report on Model Interpretability. Finally, you will use basic ML Ops techniques to productionize and monitor your model.

Index

Visual Guide to CML Workshop

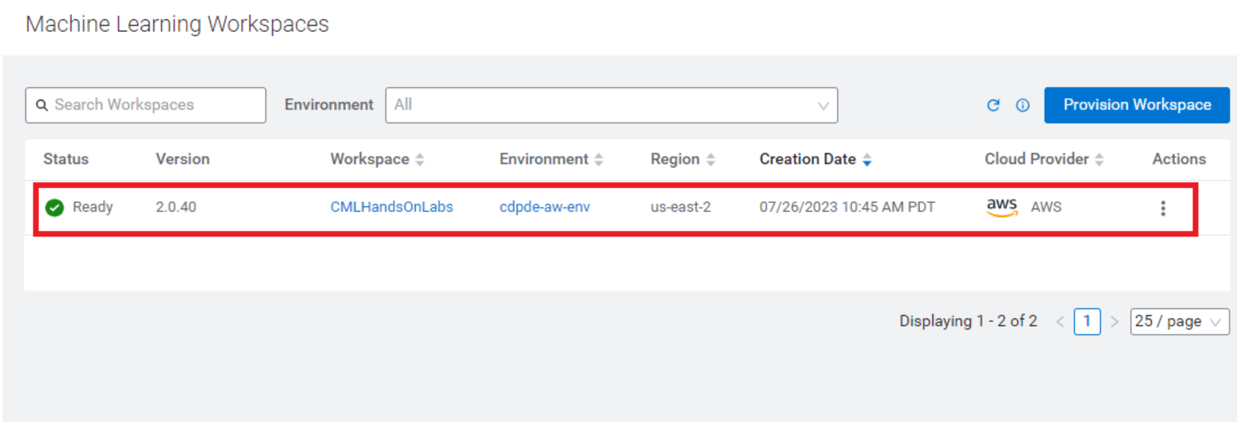


Step by Step Instructions

Part 1: Configure and deploy the Workshop Content as an [AMP](#) (10 min)

AMPs (Applied Machine Learning Prototypes) are reference Machine Learning projects that have been built by Cloudera Fast Forward Labs to provide quickstart examples and tutorials. AMPs are deployed into the Cloudera Machine Learning (CML) experience, which is a platform you can also build your own Machine Learning use cases on.

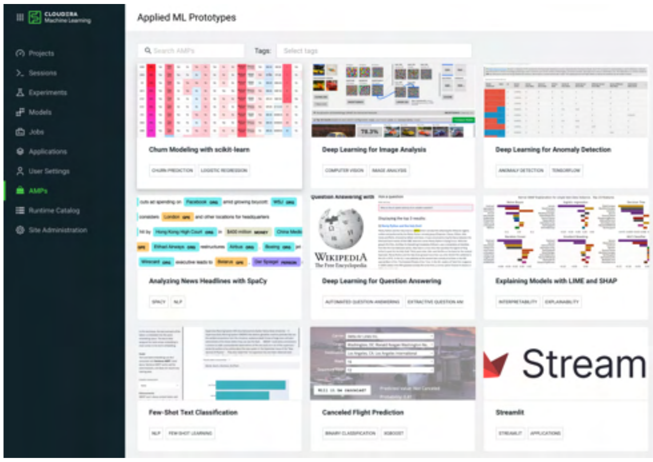
- [ ] **Go to Workshop CDP Tenant**
- [ ] **Navigate to the Machine Learning tile from the CDP Menu.**
- [ ] **Click into the Workspace by clicking the Workspace name.**



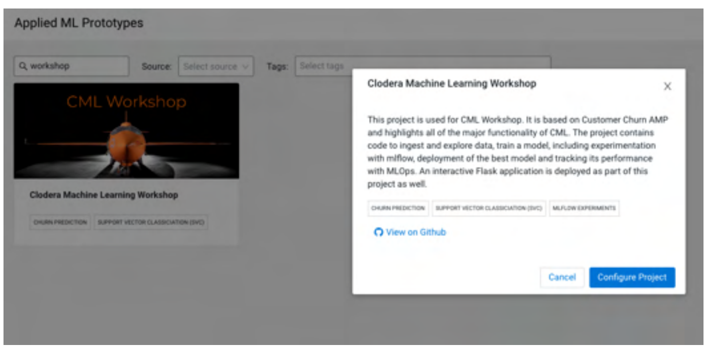
A Workspace is a cluster that runs on a kubernetes service to provide teams of data scientists a platform to develop, test, train, and ultimately deploy machine learning models. It is designed to deploy a small number of infra resources and then autoscale compute resources as needed when end users implement more workloads and use cases.

In a workspace, Projects view is the default and you'll be presented with all public (within your organization) and your own projects, if any. In this lab we will be creating a project based on Applied ML Prototype.

- ☐ **Click on** AMPs **in the side panel and search for "workshop"**



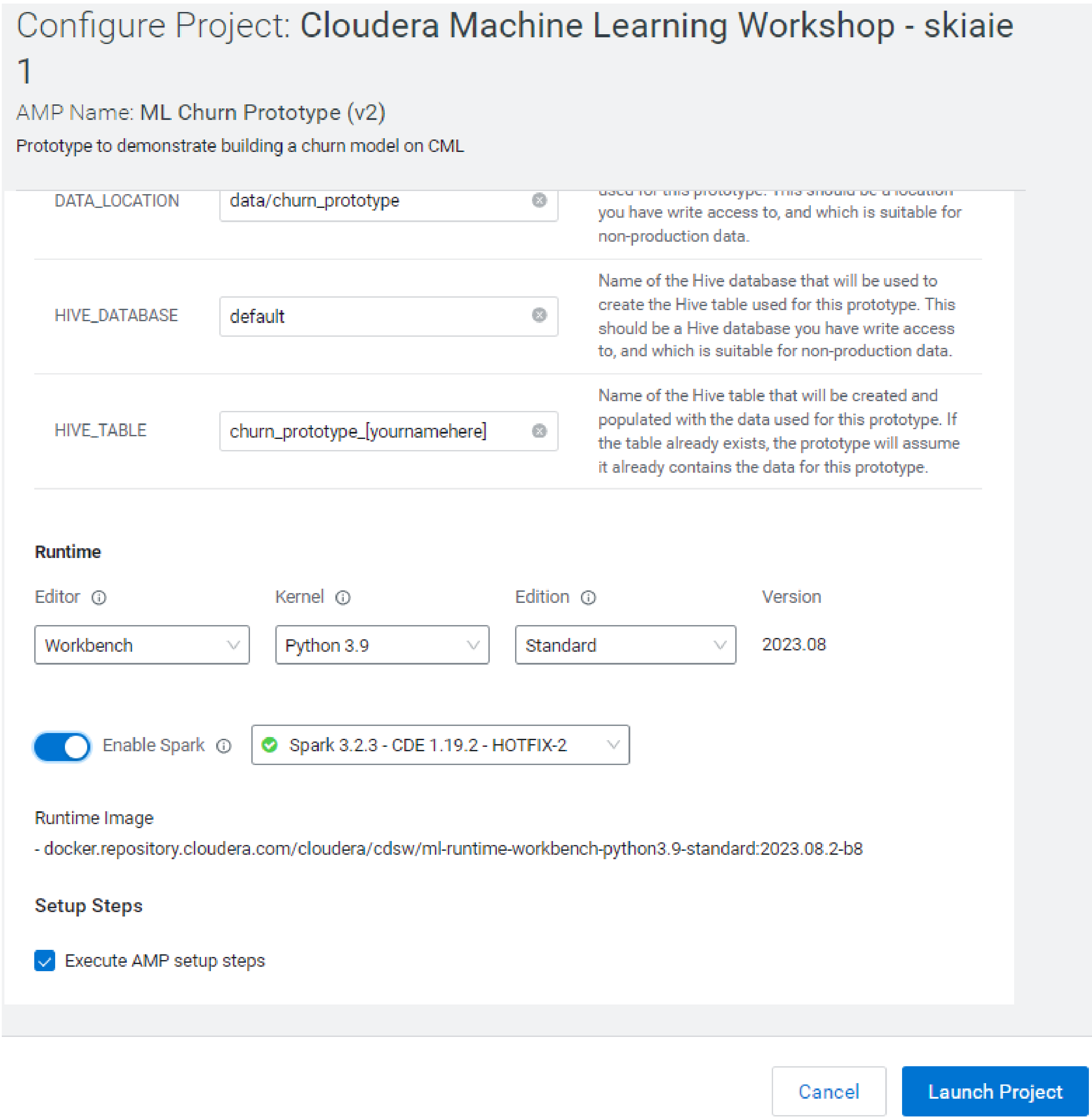
- ☐ **Click on the AMP card and then on** Configure Project



- ☐ IMPORTANT!
  - ☐ **In the Configure Project screen, change the HIVE_TABLE to have a unique suffix. Leave the other environment variables as is.**

DATA_LOCATION	data/churn_prototype
HIVE_DATABASE	default
HIVE_TABLE	churn_protype_<<YOUR LAST NAME>>

- [ ] **Select Runtime dropdowns**
    - **Workbench**
    - **Python 3.9**
    - **Enable Spark**
    - **Standard**



- [ ] **Click** Launch Project

Part 4: CML Sessions and Workbench (10 min)

Sessions allow you to perform actions such as run R, Scala or Python code. They also provide access to an interactive command prompt and terminal. Sessions will be built on a specified Runtime Image, which is a docker container that is deployed onto the ML Workspace. In addition you can specify how much compute you want the session to use.

As part of the AMP setup, the code in 0_bootsrap.py and 1_data_ingest.py was executed already. The code in those files installs every needed package for our Machine Learning project (see requirements.txt for complete listing) and loads the data into a Hive table. Additionally, the scripts sets up storage paths, uploads data files to the storage location, and finally writes out metadata information to be used in later steps of the workshop.

<u>Notebooks 2: Interactive Analysis with JupyterLab</u>

In the previous section you loaded a csv file with a python script. In this section you will perform more Python commands with Jupyter Notebooks. Notebooks have a ".ipynb" extension and need to be executed with a Session using the JupyterLabs editor.

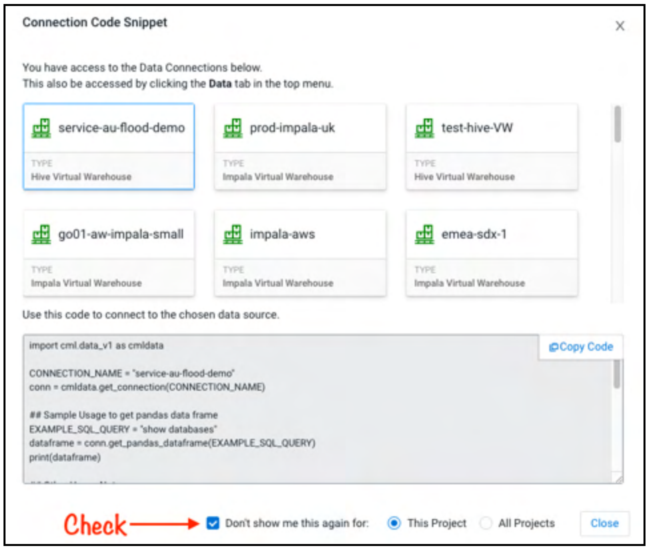
- ☐ **Click on** Overview **in the side panel**
- ☐ **Click** New Session **in the top right corner**
- ☐ **Launch the new Session with the following settings:**

Session Name: **<your session name_ lastname>**
Editor: **Jupyterlab**
Kernel: **Python 3.9**
Edition: **Standard**
Resource Profile: **1vCPU / 2GB Memory**
Runtime Version: **Any available version**

Enable Spark Add On: **enable any Spark version** Enable Spark

After a few moments the JupyterLab editor should have taken over the screen. You will be greeted with a pop-up window to get you started connecting to pre-populated Data Lake sources (e.g. virtual Data Warehouses). You could simply copy the code snippet provided and easily connect to, say, a Hive vDW. However, in this lab we won't be using this feature.

- ☐ **Check the box** ☑ Don't show me this again and click Close

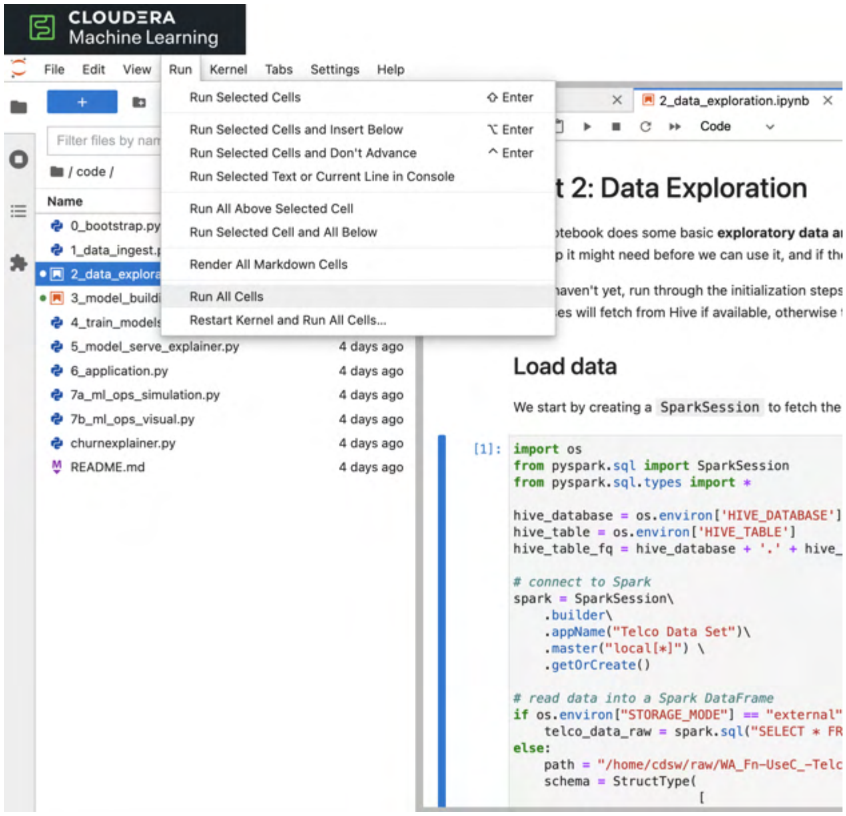


☐ **Open Notebook *code/2_data_exploration.ipynb* from the left side menu and investigate the code.**

Notebook cells are meant to be executed individually and give a more interactive flavor for coding and experimentation.

As before, no code changes are required and more detailed instructions are included in the comments. There are two ways to run each cell. Click on the cell you want to run. Hit “Shift” + “Enter” on your keyboard. Use this approach if you want to execute each cell individually. If you use this approach, make sure to run cells top to bottom, as they depend on each other.

☐ **Alternatively, open the “Run” menu from the top bar and then select “Run All”. Use this approach if you want to execute the entire notebook in bulk.**



With CML Runtimes, you can easily switch between different editors and work with multiple editors or programming environments in parallel if needed. You retrieved the data in notebook “2_data_exploration.ipynb” using a JupyterLab session via Spark SQL. Spark SQL allows you to easily exchange files across sessions. Your Spark table was tracked as a Hive External Table and automatically made available in Atlas, the Data Catalog, and CDW. This is powered by SDX integration and requires no work on the CDP Admin or Users. We will see more on this in Part 7.

Part 5: Model Training with JupyterLab (15 min)

When you are finished with notebook “2_data_exploration.ipynb” go ahead and move on to notebook “3_model_building.ipynb”. As before, no code changes are required.

- ☐ **While still in JupyterLab session, navigate to *code/3_model_building.ipynb***
- ☐ **Execute all code in *3_model_building.ipynb***

In this notebook “3_model_building.ipynb” you create a model with SciKit Learn and Lime, and then store it in your project. Optionally, you could have saved it to Cloud Storage. CML allows you to work with any other libraries of your choice. This is the power of CML… any open source library and framework is one pip install away.

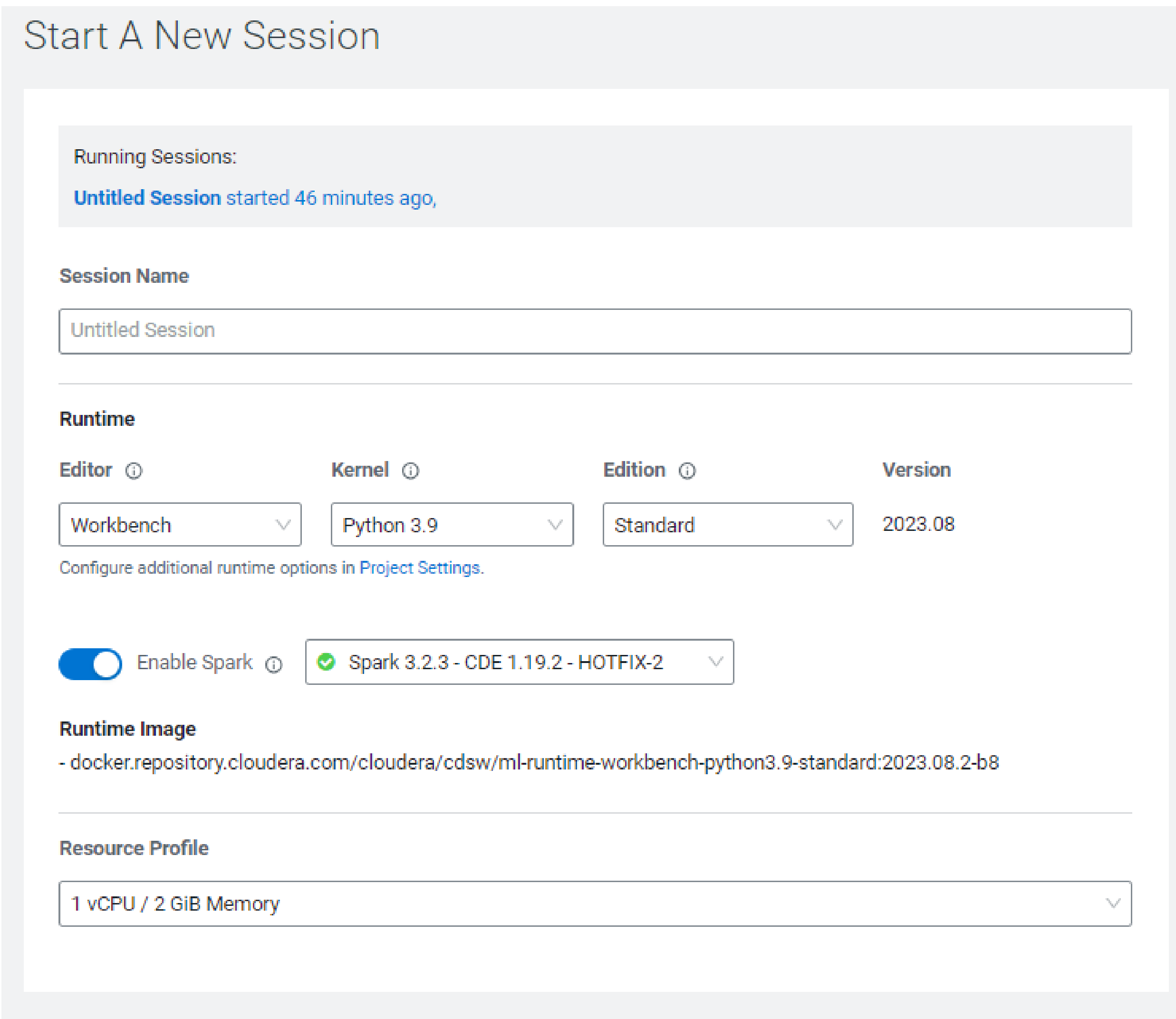
☐ Click ■ Stop to terminate your JupyterLab session

Part 5: Model Training with Workbench (15 min)

☐ Click on Overview in the side panel

☐ Click New Session in the top right corner

☐ Start a Workbench session with the following configuration



- [ ] **Ensure that** Enable Spark **add on is enabled**
- [ ] **Leave all other default settings as is and click** Start Session

Once you see the flashing red line on the bottom of the session pane turn steady green the container has been successfully started.
Model training and mlflow Experiments

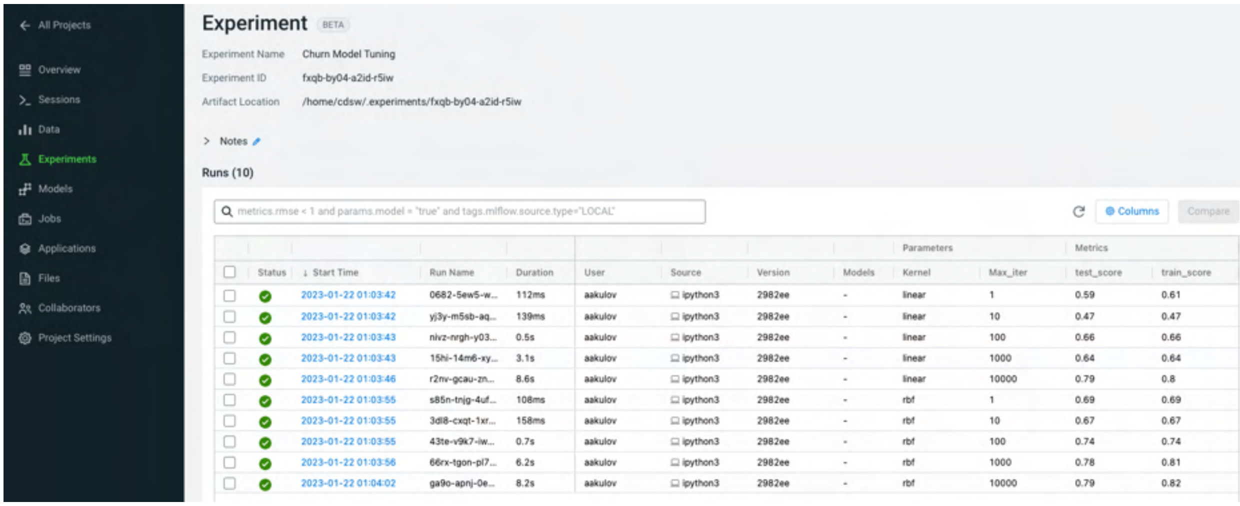
After exploring the data and building an initial, baseline model the work of optimization (a.k.a. hyperparameter tuning) can start to take place. In this phase of an ML project, model training script is made to be more robust. Further, it is now time to find model parameters that provide the "best" outcome. Depending on the model type and business use case "best" may mean use of different metrics. For instance, in a model that is built to diagnose ailments, the rate of false negatives may be especially important to determine "best" model. In cybersecurity use cae, it may be the rate of false positives that's of most interest.

To give Data Scientists flexibility to collect, record, and compare experiment runs, CML provides out-of-the-box mlflow Experiments as a framework to achieve this.

- [ ] **Inside a running Workbench session, navigate to code/4_train_model.py**
- [ ] **Click ▶ in the top menu**

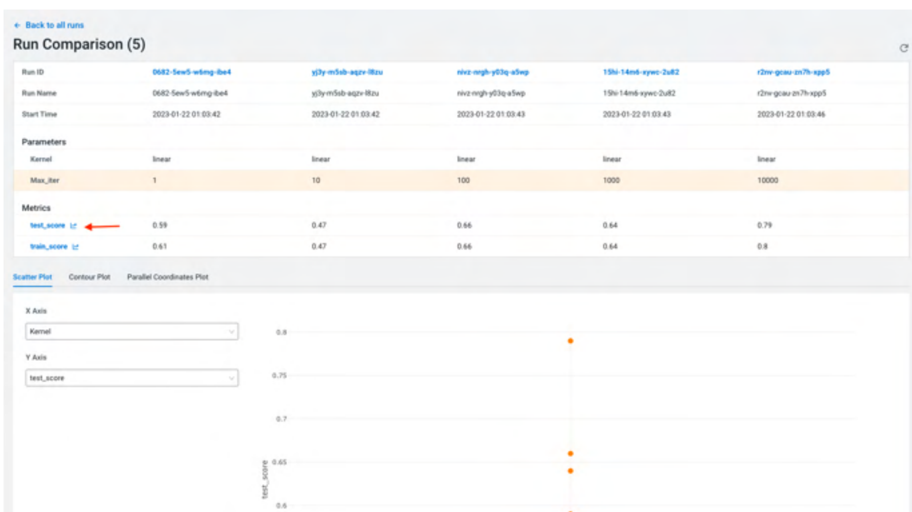
This script uses "kernel" and "max_iter" as the two parameters to manipulate during model training in order to achieve the best result. In our case, we'll define "best" as the highest "test_score".

- [ ] **While your script is running, click on** ← Project **in the top panel of the REPL**
- [ ] **Click on** Experiments **in the side bar**
- [ ] **Click on *Churn Model Tuning***



As expected, higher number of max_iterations produces better result (higher test_score). Interestingly, the choice of kernel does not make a difference at higher max_iter values. We can choose linear as it allows for faster model training.

- ☐ **Select all runs with "linear" Kernel**
- ☐ **Click** Compare
- ☐ **Click** **test_score**



Built-in visualizations in mlflow allow for more detailed comparison of various experiment runs and outcomes. You can also access the same experiment data via mlflow API from a running session. There there visualization possibilities are limitless.

Part 6: CML Model Deployment (15 min)

Once a model is trained its predictions and insights must be put to use so they can add value to the organization. Generally this means using the model on new, unseen data in a production environment that offers key ML Ops capabilities.

One such example is Batch Scoring via CML Jobs. The model is loaded in a script and the predict function provided by the ML framework is applied to data in batch. The script is scheduled and orchestrated to perform the scoring on a regular basis. In case of failures, the script or data are manually updated so the scoring can resume.

This pattern is simple and reliable but has one pitfall. It requires the user or system waiting for the scoring job to run at its scheduled time. What if predictions are required on a short notice? Perhaps when a prospect navigates on an online shopping website or a potential anomaly is flagged by a third party business system?

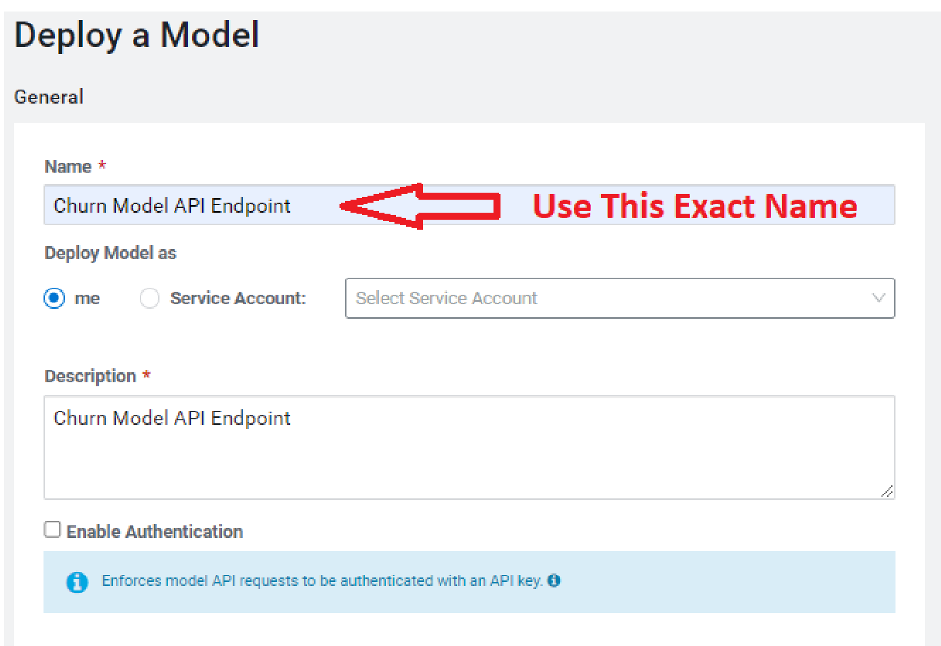
- CML Models allow you to deploy the same model script and model file in a REST Endpoint so the model can now serve responses in real time. The endpoint is hosted by a container.
- CML Models provides tracking, metadata and versioning features that allow you to manage models in production.
- Similarly, CML Applications allows you to deploy visual tools in an endpoint container. This is typically used to host apps with open source libraries such as Flask, Shiny, Streamlit and more.
- Once a model is deployed to a CML Models container, a CML Application can forward requests to the Model endpoint to provide visual insights powered by ML models.

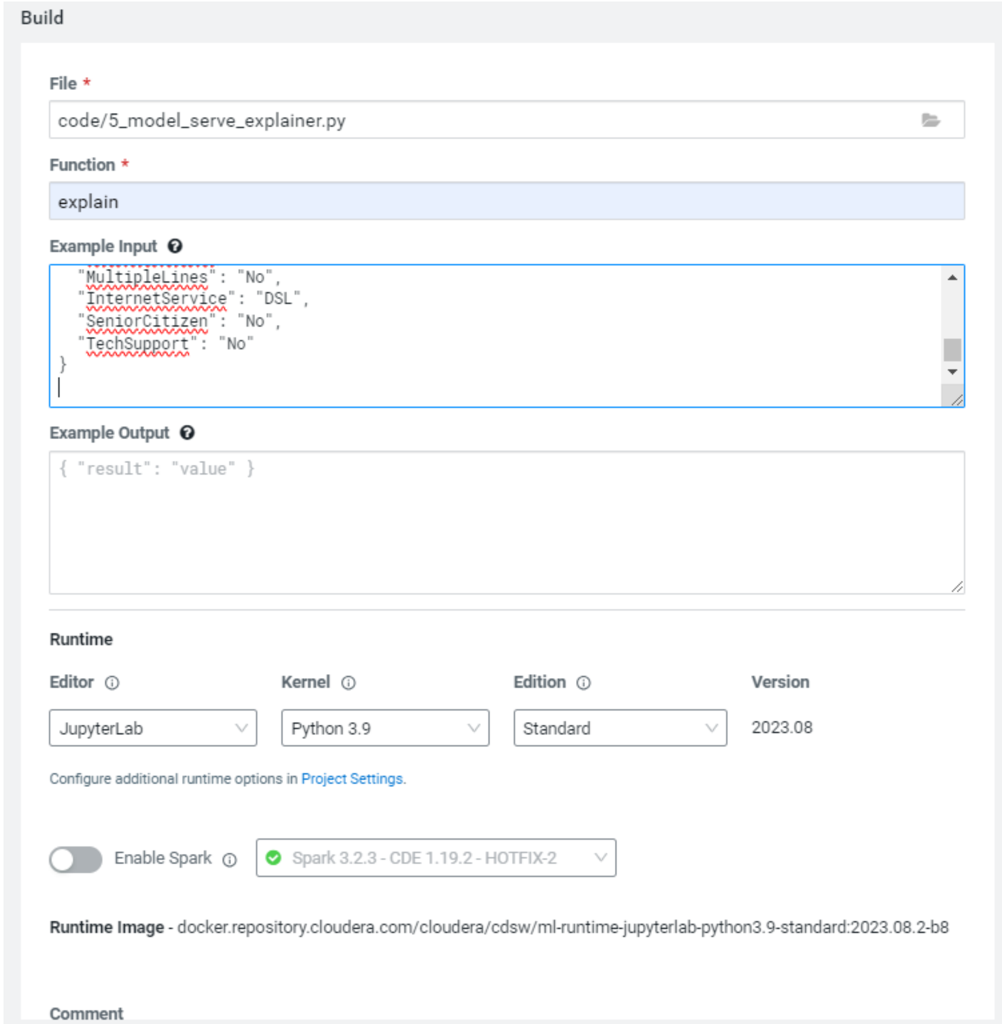
Below are the steps to deploy a near-real-time scoring model:

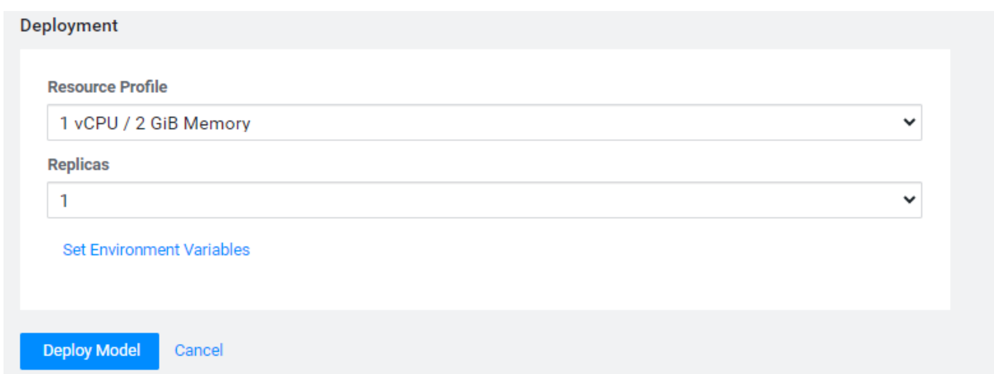
- [ ] **Click on** Models **in the side panel**
- [ ] **Click** New Model
- [ ] **Important! Name your model *Churn Model API Endpoint***
  Any other name will cause issues with downstream scripts.
- [ ] **Give your model any description**
- [ ] **Important! Uncheck** ☐ Enable Authentication
- [ ] **Under File select *code/5_model_serve_explainer.py***
- [ ] **Under Function enter *explain***
- [ ] **For Example Input enter the following JSON**
- [ ] **You do not need to Enable Spark for model serving in this case**

This JSON is a set of key value pairs representing a customer's attributes. For example, a customer who is currently on a DSL Internet Service plan.

```
Unset
{
  "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"
}
```



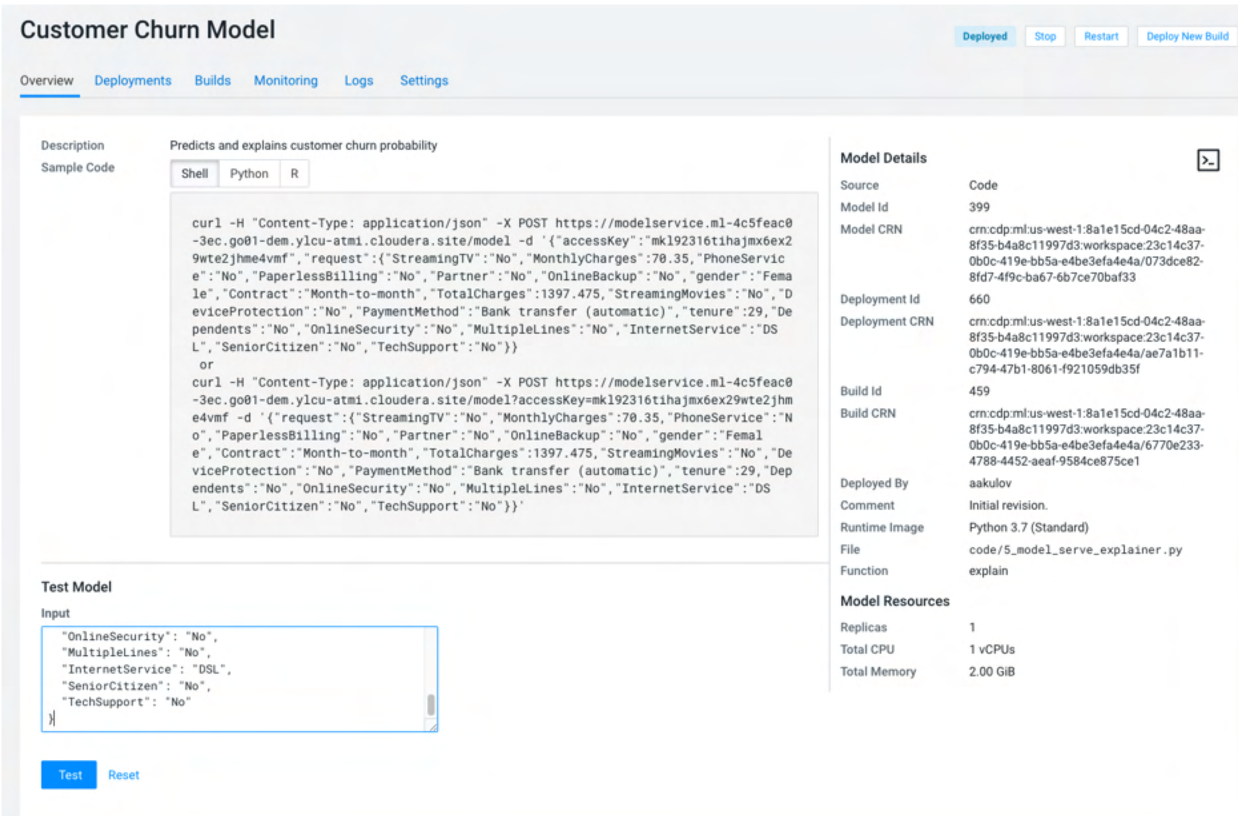




- [ ] **Scroll to the bottom of the page and click** Deploy Model

Model deployment may take a minute or two, meanwhile you can click on the Model name and explore the UI. The code for a sample request is provided on the left side. On the right side observe the model's metadata. Each model is assigned a number of attributes including Model Name, Deployment, Build and creation timestamp.

- ☐ **Note down the *Build Id* of your model, we will need it in MLOps part of the workshops**



- ☐ **Once your model is Deployed, click** Test

The test simulates a request submission to the Model endpoint. The model processes the input and returns the output along with metadata and a prediction for the customer. In addition, the request is assigned a unique identifier. We will use this metadata for ML Ops later in part 6.

<u>Script 5: Inspecting a Model Script</u>

- ☐ **Navigate back to the Project Overview page and open the "5_model_serve_explainer.py" script. Scroll down and familiarize yourself with the code.**

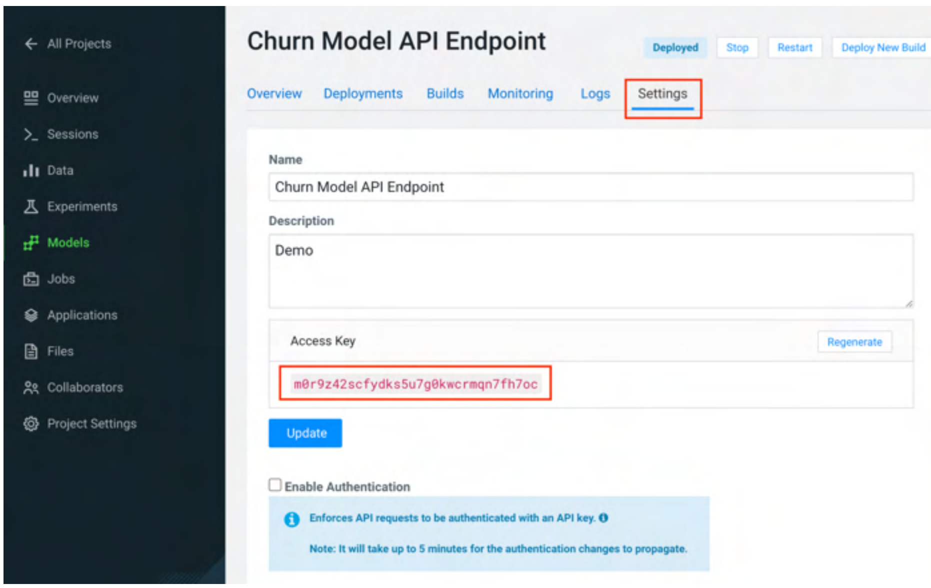
- Notice the method “explain” method. This is the Python function whose purpose is to receive the Json input as a request and return a Json output as a response.
- Within the method, the classifier object is used to apply the model object’s predict method.
- In addition, notice that a decorator named “@cdsw.model_metrics” is applied to the “explain” method. Thanks to the decorator you can use the “cdsw.track_metric” methods inside the “explain” method to register each scalar value associated with each request.
- The values are saved in the Model Metrics Store, a built in database used for tracking model requests.

Navigate back to the Project Overview page. Under *Files* listing, open the “models/telco_linear” subfolder and notice the presence of the “telco_linear.pkl” file. This is the physical model file loaded by the .py script you just inspected above.

Part 7: Interacting with the Visual Application (10 min)

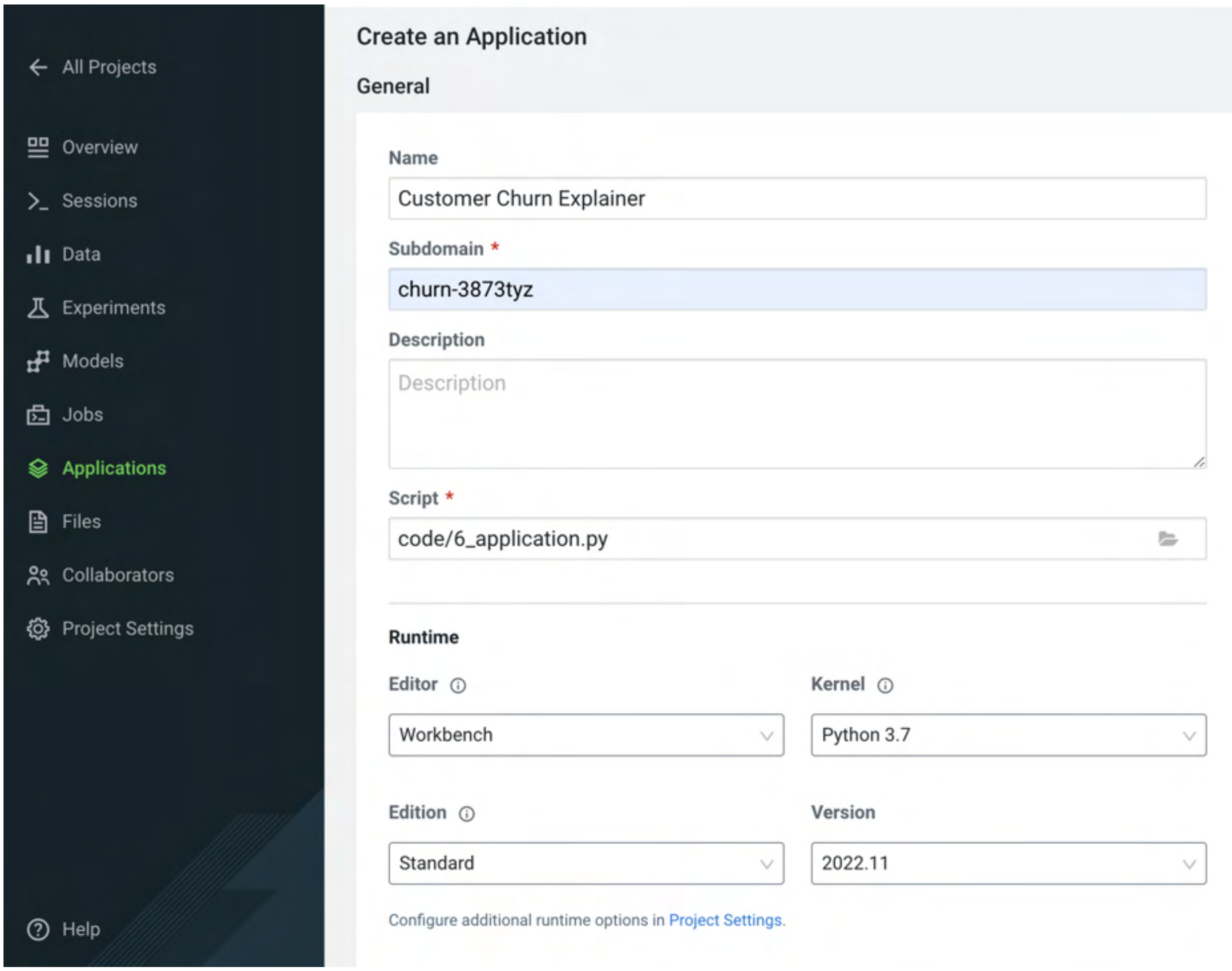
Any custom, UI app can be hosted within CML. These can be streamlit, Django, or Rshiny (or other frameworks) apps that deliver custom visualization or incorporate a real-time model scoring. In the following steps we will deploy an Application for the Churn Customer project:

- ☐ **Go to Models and click on the model that you've deployed**
- ☐ **Go to the *Settings* tab and copy the *Access Key* string**



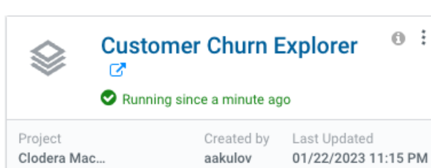
- ☐ **Navigate to *Files* > *flask* > *single_view.html***
- ☐ **Click Open In Session in the top right corner**
- ☐ **Important! On line 61 of the file, update the access key value with the Access Key you got earlier. Click *File* > *Save* (or ⌘+S)**
- ☐ **Click on Applications in the side panel**
- ☐ **Click on New Application**

- ☐ **Give your application a name, and provide a *unique* subdomain (e.g. your last name, no spaces or punctuation)**
- ☐ **Under *Scripts* select code/6_application.py**
- ☐ **Ensure that a *Workbench* editor is selected and** Enable Spark **toggle is turned on**



- ☐ **Scroll the bottom of the page and click** Create Application

Application startup can take up to 2 minutes, and once the application is ready you'll see a card similar to this:



- ☐ **Click on the application in order to open it.**

This will automatically redirect you to the Visual Application landing page where the same data you worked with earlier is presented in an interactive table.

On the left side notice the probability column. This is the target variable predicted by the Machine Learning Model. It reflects the probability of each customer churning. The value is between 0 and 1. A value of 0.49 represents a 49% probability of the customer churning. By default, if the probability is higher than 50% the classifier will label the customer as “will churn” and otherwise as “will not churn”.

The 50% threshold can be increased or decreased implying customers previously assigned a “will churn” label may flip to “will not churn” and vice versa. This has important implications as it provides an avenue for tuning the level selectivity based on business considerations but a detailed explanation is beyond the scope of this content.

Next, click on the customer at the top of the table to investigate further.

Refractor

id	Probability	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	D
2672	0.584	Femal	No	No	No	4	Yes	No	Fiber	No	No	N
1540	0.123	Male	No	No	No	63	Yes	Yes	Fiber	No	Yes	Y
1724	0.061	Male	No	Yes	No	64	Yes	Yes	Fiber	Yes	Yes	Y
2085	0.060	Femal	No	Yes	Yes	48	No	No ph	DSL	No	No	N
2434	0.045	Male	No	Yes	Yes	17	Yes	No	No	No in	No in	N
5073	0.009	Male	No	Yes	Yes	53	Yes	No	DSL	Yes	Yes	N
1150	0.009	Femal	No	Yes	Yes	45	Yes	Yes	No	No in	No in	N
4151	0.008	Male	No	Yes	Yes	39	Yes	Yes	No	No in	No in	N
4249	0.003	Femal	No	Yes	No	70	Yes	Yes	No	No in	No in	N
1712	0.001	Male	No	Yes	No	72	Yes	No	No	No in	No in	N

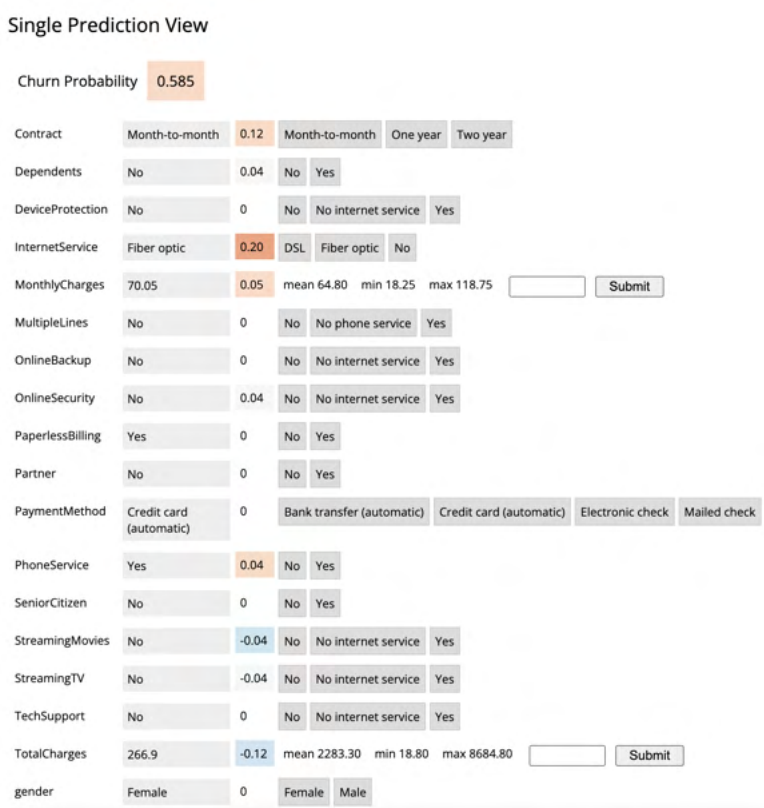
A more detailed view of the customer is automatically loaded. The customer has a 58% chance of churning.

The Lime model applied to the classifier provides a color coding scheme highlighting the most impactful features in the prediction label being applied to this specific customer.

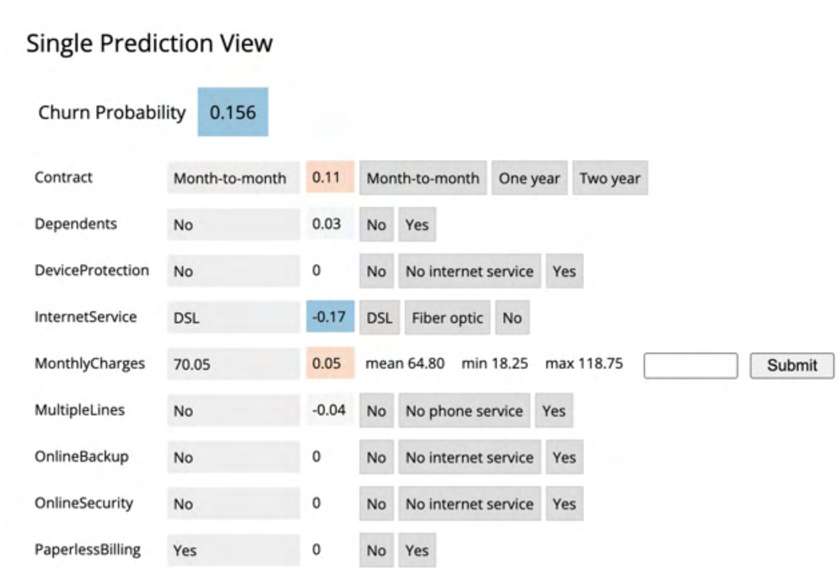
For example, this customer’s prediction of “will churn” is more significantly influenced by the “Internet Service” feature.

- The dark red color coding signals that the customer is negatively impacted by the current value for the feature.

- The current values of Monthly Charges and Phone Service also increase the likelihood of churn while the values of the Streaming Movies and Total Charges features decrease the likelihood of churn.



Let's see what happens if we change the value for the most impactful feature in this given scenario i.e. "Internet Service". Currently the value is set to "Fiber Optic". Hover over the entry in the table and select "DSL".



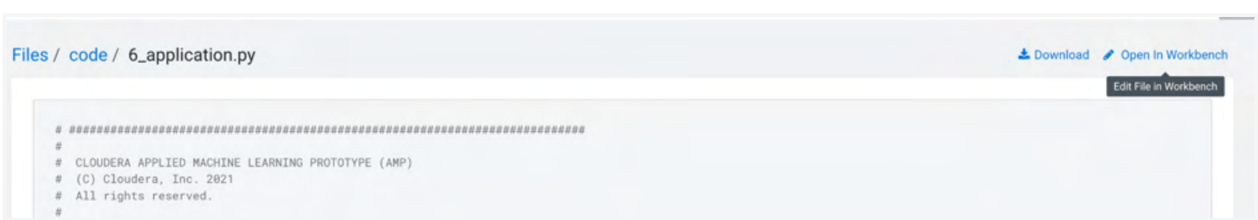
The table has now reloaded and the churn probability for this customer has dramatically decreased to roughly 15%.

This simple analysis can help the marketer optimize strategy in accordance to different business objectives. For example, the company could now tailor a proactive marketing offer based on this precious information. In addition, a more thorough financial analysis could be tied to the above simulation perhaps after adjusting the 50% threshold to increase or decrease selectivity based on business constraints or customer lifetime value assigned to each customer.

<u>Script 6: Exploring the Application Script</u>

Navigate back to the CML Project Home folder (Overview). Open the “Code” folder and then script “6_application.py”. This is a basic Flask application that serves the HTML and some specific data used for.

Click on “Open in Session” to visualize the code in a more reader friendly-mode.



Now you will be able to explore the code with the Workbench Editor. The “Launch Session” form will automatically load on the right side of your screen. There is no need to launch a session so you can just minimize it.

As always no code changes are required. Here are some key highlights::

- At lines 177 - 191 we load the model and use the “Explain” method to load a small dataset in the file. This is similar to what you did in script 5. If you want to display more data or fast changing data there are other ways to do this, for example with Cloudera SQL Stream Builder.
- At line 248 we run the app on the "CDSW_APP_PORT". This value is already preset for you as this is a default environment variable. You can reuse this port for other applications.

Part 8: CML Models Operations (15 min)

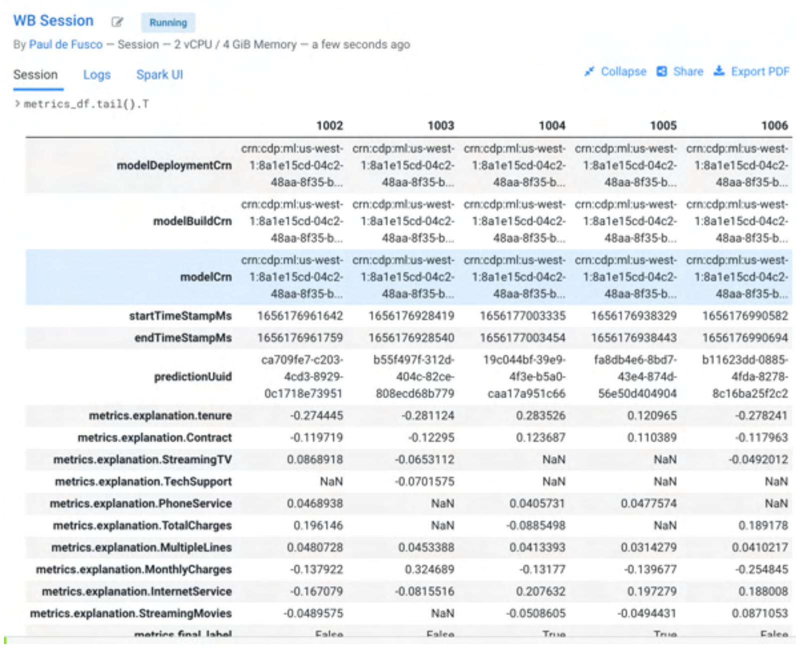
- ☐ **Navigate back to the project overview and launch a new session with the following configurations.**

Session Name: ***<your session name>***
Editor: **Workbench**
Kernel: **Python 3.9**
Resource Profile: **1vCPU/2 GiB Memory**
Runtime Edition: **Standard**
Runtime Version: **Any available version**
Enable Spark Add On: **any Spark version**

- ☐ **Once the session is running, open script "7a_ml_ops_simulation.py" and execute the whole script end to end. Wait for this script to complete.**
- ☐ **Open script "7b_ml_ops_visual.py" and explore the code in the editor.**
- ☐ **Execute the whole script end to end**

Observe the code outputs on the right side. Here are the key highlights:

- Model predictions are tracked in the CML Models Metrics Store. This is enabled by the use of the Python decorator and the use of "cdsw.track_metrics" methods in script 5. What is being tracked is completely up to the script developer.
- You can then extract the predictions and related metadata and put the information in a Pandas dataframe. Again, the Python library you use does not matter and is entirely up to the developer.
- This is exactly what the first diagram on the right side of your screen shows. Each column represents a prediction request reaching your CML Model endpoint. Each row represents a metric you are tracking in the CML Models Metrics Store.



	1002	1003	1004	1005	1006
modelDeploymentCrn	crn:cdp:ml:us-west-1:8a1e15cd-04c2-48aa-8f35-b...	crn:cdp:ml:us-west-1:8a1e15cd-04c2-48aa-8f35-b...	crn:cdp:ml:us-west-1:8a1e15cd-04c2-48aa-8f35-b...	crn:cdp:ml:us-west-1:8a1e15cd-04c2-48aa-8f35-b...	crn:cdp:ml:us-west-1:8a1e15cd-04c2-48aa-8f35-b...
modelBuildCrn	crn:cdp:ml:us-west-1:8a1e15cd-04c2-48aa-8f35-b...	crn:cdp:ml:us-west-1:8a1e15cd-04c2-48aa-8f35-b...	crn:cdp:ml:us-west-1:8a1e15cd-04c2-48aa-8f35-b...	crn:cdp:ml:us-west-1:8a1e15cd-04c2-48aa-8f35-b...	crn:cdp:ml:us-west-1:8a1e15cd-04c2-48aa-8f35-b...
modelCrn	crn:cdp:ml:us-west-1:8a1e15cd-04c2-48aa-8f35-b...	crn:cdp:ml:us-west-1:8a1e15cd-04c2-48aa-8f35-b...	crn:cdp:ml:us-west-1:8a1e15cd-04c2-48aa-8f35-b...	crn:cdp:ml:us-west-1:8a1e15cd-04c2-48aa-8f35-b...	crn:cdp:ml:us-west-1:8a1e15cd-04c2-48aa-8f35-b...
startTimeStampMs	1656176961642	1656176928419	1656177003335	1656176938329	1656176990582
endTimeStampMs	1656176961759	1656176928540	1656177003454	1656176938443	1656176990694
predictionUuid	ca709fe7-c203-4cd3-8929-0c1718e73951	b55f497f-312d-404c-82ce-808ecd68b779	19c044bf-39e9-4f3e-b5a0-caa17a951c66	fa8db4e6-8bd7-43e4-874d-56e50d404904	b11623dd-0885-4fda-8278-8c16ba25f2c2
metrics.explanation.tenure	-0.274445	-0.281124	0.283526	0.120965	-0.278241
metrics.explanation.Contract	-0.119719	-0.12295	0.123687	0.110389	-0.117963
metrics.explanation.StreamingTV	0.0868918	-0.0653112	NaN	NaN	-0.0492012
metrics.explanation.TechSupport	NaN	-0.0701575	NaN	NaN	NaN
metrics.explanation.PhoneService	0.0468938	NaN	0.0405731	0.0477574	NaN
metrics.explanation.TotalCharges	0.196146	NaN	-0.0885498	NaN	0.189178
metrics.explanation.MultipleLines	0.0480728	0.0453388	0.0413393	0.0314279	0.0410217
metrics.explanation.MonthlyCharges	-0.137922	0.324689	-0.13177	-0.139677	-0.254845
metrics.explanation.InternetService	-0.167079	-0.0815516	0.207632	0.197279	0.188008
metrics.explanation.StreamingMovies	-0.0489575	NaN	-0.0508605	-0.0494431	0.0871053
metrics.final_label	False	False	True	True	False

- Once the tracked metrics have been saved to a Python data structure they can be used for all sorts of purposes.
- For example, the second diagram shows a basic line plot in Seaborn where the models' output probabilities are plotted as a function of time. On the X axis you can see the timestamp associated with each request. On the Y axis you can find the associated output probability.

- Similarly, you can plot processing time as shown in the third diagram. This represents the time duration required to process a particular request.

- As an example, this information could be used to trigger the deployment of more resources to support this model endpoint when a particular threshold is passed. You can deploy more resources manually via the UI, or programmatically and in an automated CI/CD pipeline with CML APIv2 and CML Jobs.

- You can also monitor the model's accuracy over time. For example, the below diagram shows a line plot of prediction accuracy sorted over time. As you can see, the trend is negative and the model is making increasingly less accurate predictions.
- Just like with processing time and other metrics, CML allows you to implement ML Ops pipelines that automate actions related to model management. For example, you could use a combination of CML Jobs and CML APIv2 to trigger the retraining and redeployment of a model when its accuracy reaches a particular threshold over a particular time period.
- As always this is a relatively basic example. CML is an open platform for hands-on developers which gives users the freedom to implement more complex ML Ops pipelines.

- Ground truth metrics can be collected with the cdsw.track_delayed_metrics method. This allows you to compare your predictions with the actual event after the prediction was output. In turn, this allows you to calculate the model accuracy and create visualizations such as the one above.
- For an example of the cdsw.track_delayed_metrics method open the “7a_ml_ops_simulation.py” script and review lines 249 - 269. Keep in mind that this is just a simulation.
- In a real world scenario the requests would be coming from an external system or be logged in a SQL or NoSQL database. In turn, the script above would be used to set ground truth values in batch via a CML Job or in real time with a CML Model endpoint.