Anomaly Detection and Failure Prediction workshop

1. Introduction
2. Log into the Red Hat OpenShift Data Science Platform
3. Start your Jupyter Notebook server
    a. The Jupyter environment
    b. Clone a github repository
    c. Introduction to Jupyter Notebooks
    d. Anomaly Detection
    e. Generate Synthetic Data
4. Deploy the Failure Prediction web application as a container
    a. Log into the Failure Prediction web application
    b. Stream the synthetic data & Perform a Failure Prediction
5. Conclusion

Introduction

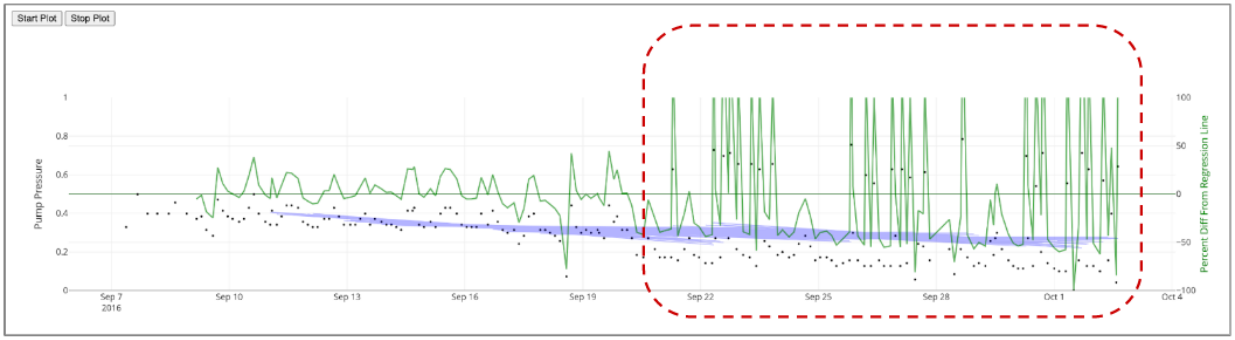
Welcome! In this introductory workshop, you'll learn how to:

1. Use the Red Hat OpenShift Data Science platform to detect anomalies in mechanical sensor data.
2. Generate Synthetic sensor data to mimic real-time data generated by an Edge device
3. Deploy a 'containerized web application' (in the Red Hat Openshift platform) to predict the failure of a mechanical device.

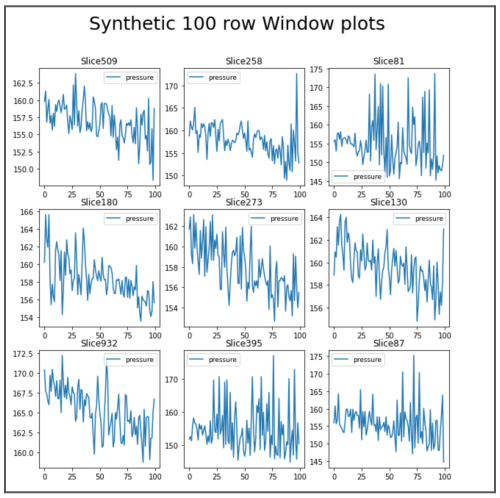
We'll start from a sensor dataset like this:

timestamp	sensor_25	sensor_11	sensor_36	sensor_34
2018-04-10 09:55:00	0.886128288849588	0.6936344767141810	0.21762059617623900	0.25318778143401500
2018-04-10 09:56:00	0.9204208188441080	0.7000403380548420	0.0968758054521226	0.2539990089674020
2018-04-10 09:57:00	0.9202596186121920	0.7037891972869380	0.09844494043538750	0.2719518531347420
2018-04-10 09:58:00	0.8767057041000580	0.7074345247993290	0.11346761648322900	0.26238610341377100
2018-04-10 09:59:00	0.907298589265789	0.7092217106877420	0.11872560793926700	0.24854953238656100
2018-04-10 10:00:00	0.9458851484833250	0.7088721741223050	0.12288425435926700	0.25818383654697300
2018-04-10 10:01:00	0.9458851484833250	0.7088721741223050	0.12288425435926700	0.25818383654697300
2018-04-10 10:02:00	0.8725669047299990	0.7112236359658490	0.12616730006050600	0.26549390466843700
2018-04-10 10:03:00	0.8795137723375100	0.7087605920157160	0.11858339137158600	0.2587340761266980
2018-04-10 10:04:00	0.8923600377327950	0.7036476602524060	0.126257236501765	0.27057234537967100

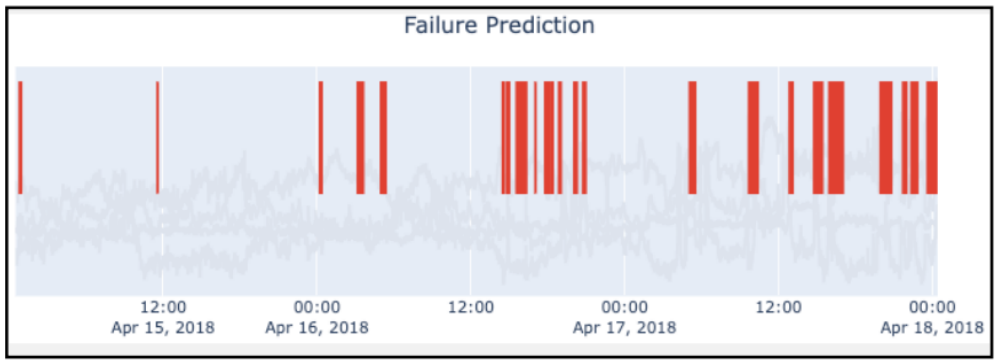
We will learn how to recognize anomalies in our data like this:



We will then generate synthetic data to mimic ‘live’ sensor data streaming from an Edge device connected to a Mechanical Pump. Plotted sensor data will look like this:



From this synthetic sensor data we will use our failure prediction model to predict when our mechanical device will fail based on sensor readings like this:



The failure prediction model will be accessible, via a Failure Prediction web application that we will deploy on the OpenShift (kubernetes) platform. We can do all of this without having to install anything on your computer, thanks to Red Hat OpenShift Data Science!

If you’re ready, let’s start!

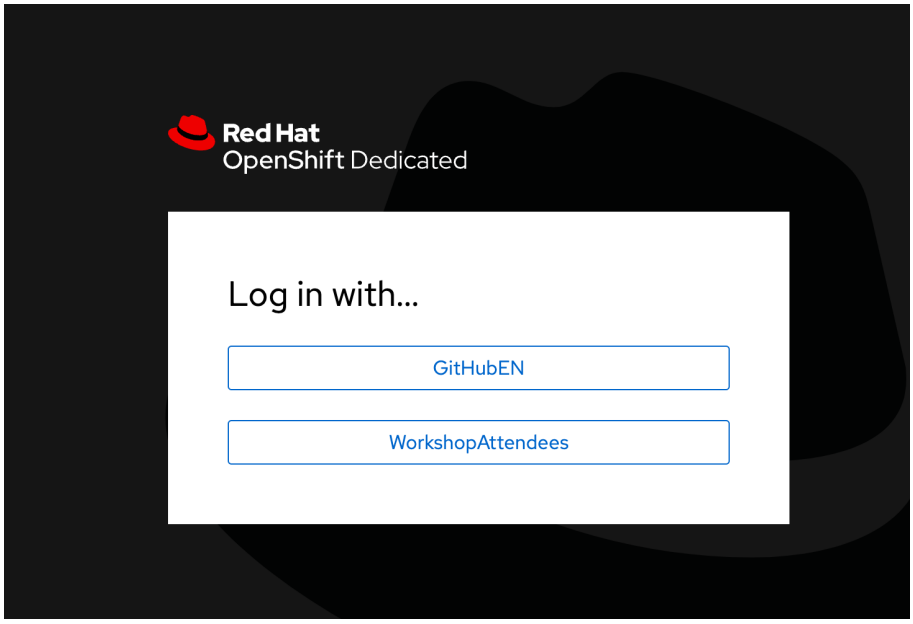
Log into the Red Hat OpenShift Data Science Platform

Red Hat OpenShift Data Science is a managed cloud service for data scientists and developers of artificial intelligence (AI) applications. It provides a fully supported environment in which to rapidly develop, train, and test machine learning models in the public cloud before deploying them in production. When you use OpenShift Data Science, you are running in Red Hat OpenShift Dedicated or Red Hat OpenShift Service on AWS, which simplifies cloud operations.

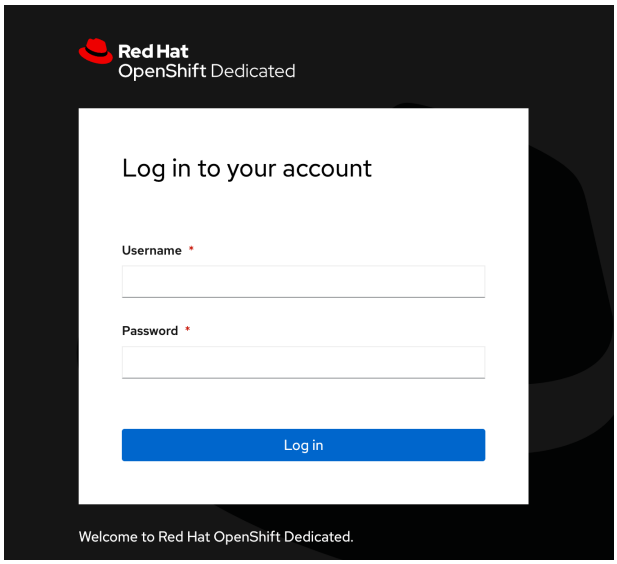
Let's get started!

Use the following url to access the Red Hat OpenShift Data Science platform:
https://console-openshift-console.apps.ieee.d7se.p1.openshiftapps.com

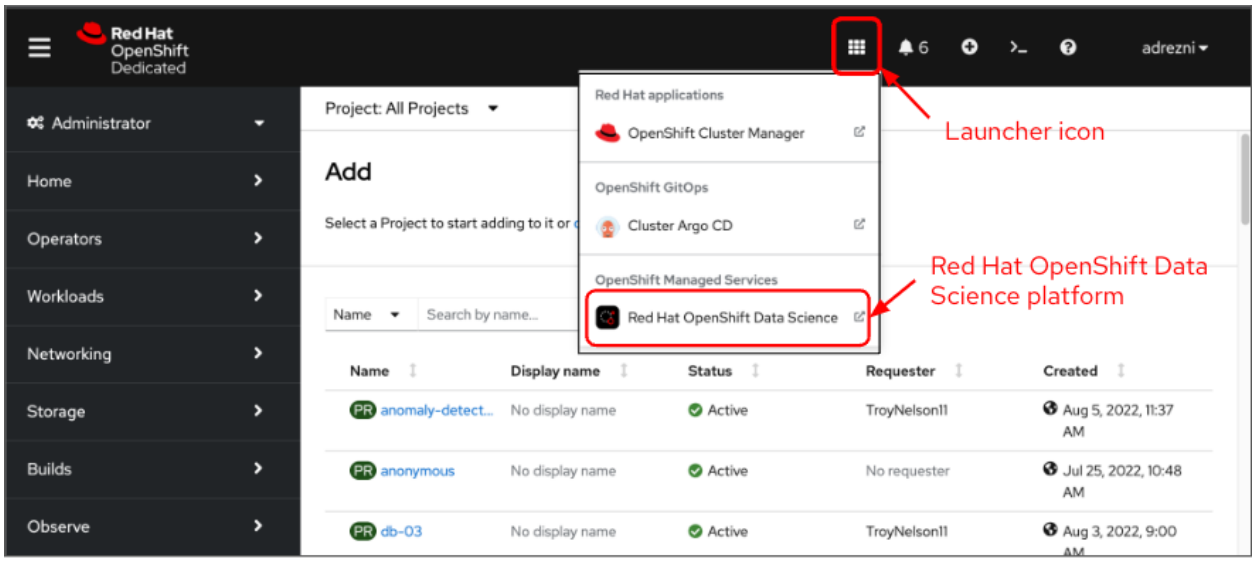
You will see the following login page appear. Click the 'WorkshopAttendees' button.



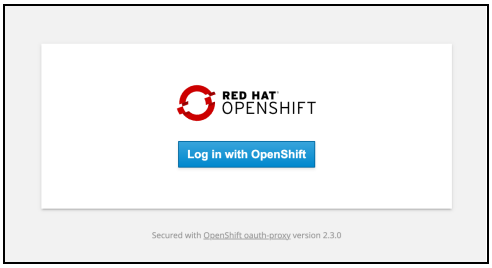
You will see the following login page appear. Use the username and password that you have been given. For example, user01 and password01



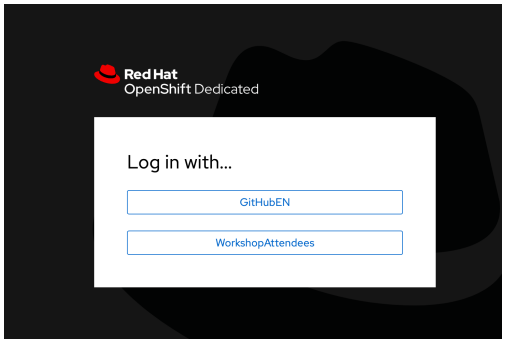
Upon successful login, you will see the Red Hat OpenShift Dedicated environment. This is a kubernetes platform from which you can create containerized applications. Part of this environment is the Red Hat OpenShift Data Science platform. It is accessed by selecting the launcher icon in the upper right corner. Select the option 'Red Hat OpenShift Data Science'



Upon clicking the Launcher Icon you will see another login screen. Click the 'Log in with OpenShift' icon.

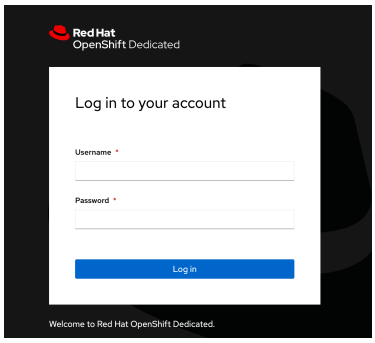


You will see another Log in page. Click the “Workshop Attendees” button.

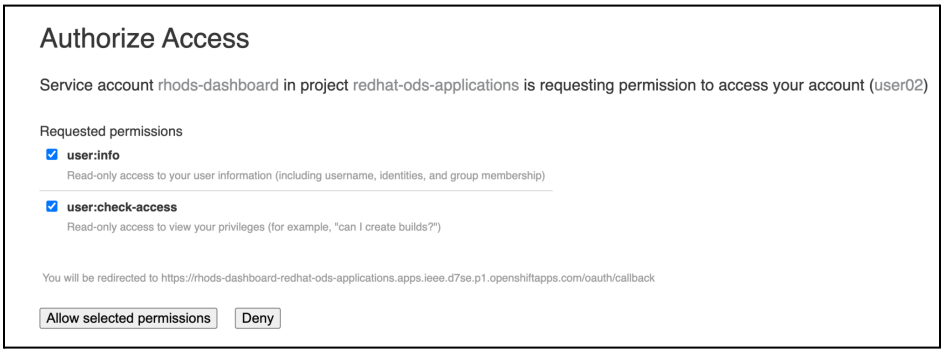


You will be prompted, again, for the username and password you have been given.

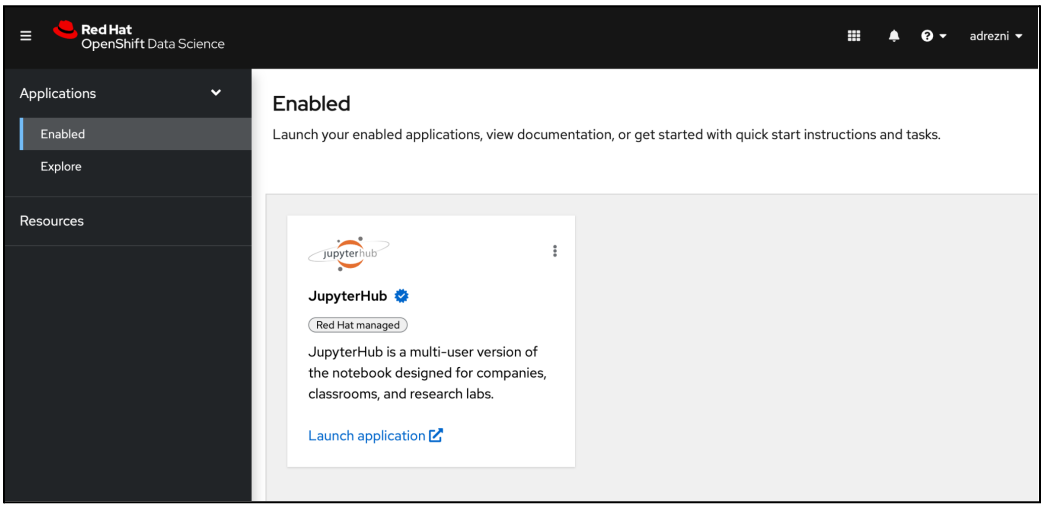
Enter them and click the ‘Log in’ button



You may see an ‘Authorize Access’ screen. Click the ‘Allow selected permissions’ button.



You will now be logged into the Red Hat OpenShift Data Science platform.



We are ready to start our JupyterHub notebook server!

Start your Jupyter notebook server

Red Hat OpenShift Data Science is a managed cloud service for data scientists and developers of artificial intelligence (AI) applications. It provides a fully supported environment in which to rapidly develop, train, and test machine learning models in the public cloud before deploying them in production. When you use OpenShift Data Science, you are running in Red Hat OpenShift Dedicated or Red Hat OpenShift Service on AWS, which simplifies cloud operations.

Let's launch the Jupyter Notebook server and set up options! Click the Launch application url.

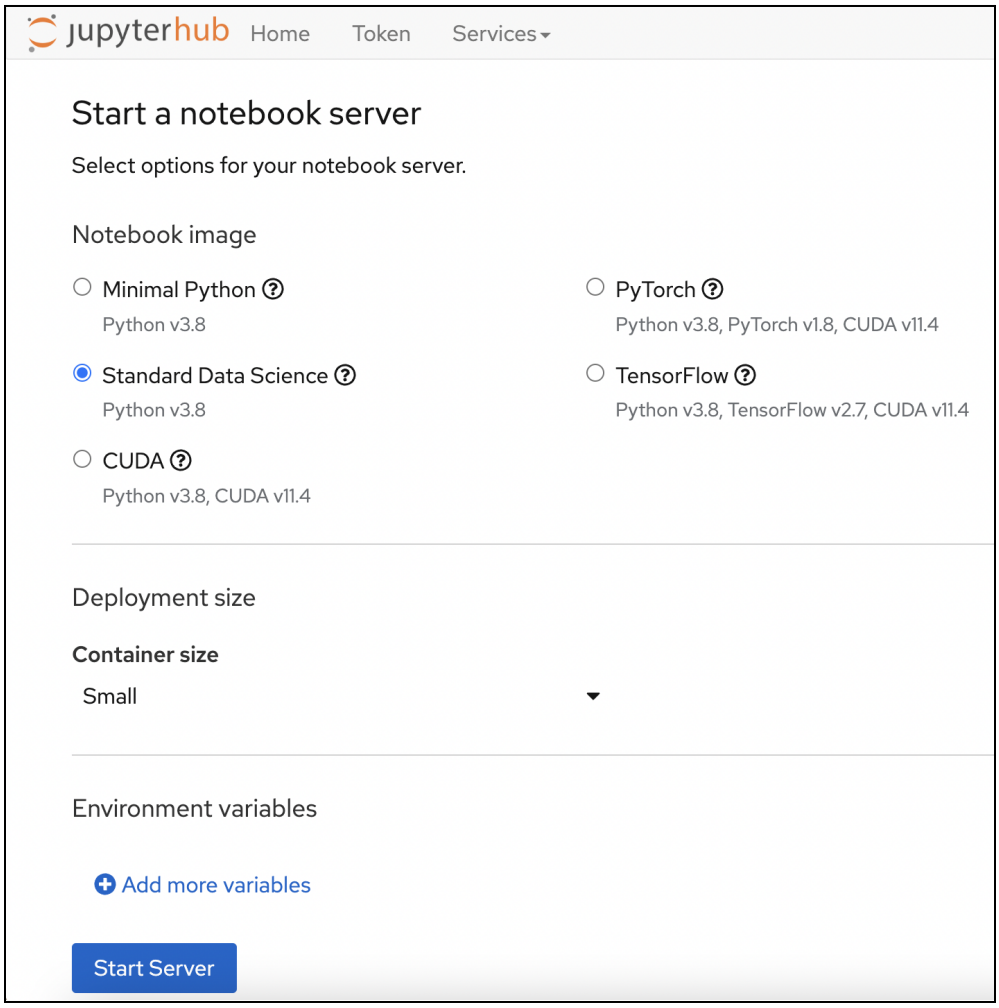
Note: You may need to log in again with your workshop username and password and then Authorize Access.

Select Options for your notebook server

When you first gain access to JupyterHub, a configuration screen gives you the opportunity to select a notebook image and configure the deployment size and environment variables for your data science project.

You can customize the following options:

- Notebook image
- Deployment size
- Environment variables



The following subsections list which prerequisite you need to choose for this activity.

Notebook image

You can choose from a number of predefined images. When you choose a predefined image, your JupyterLab instance has the associated libraries and packages that you need to do your work.

Available notebook images include:

- Minimal Python
- PyTorch
- **Standard Data Science**
- Tensorflow
- CUDA

For this learning path, choose the **Standard Data Science** notebook image.

Deployment size

You can choose different deployment sizes (resource settings) based on the type of data analysis and machine learning code you are working on. Each deployment size is pre-configured with specific CPU and memory resources.

For this learning path, select the **Small** deployment size.

Environment variables

The environment variables section is useful for injecting dynamic information that you don't want to save in your notebook.

This learning path does not use any environment variables.

If you are satisfied with your notebook server selections, click the **Start Server** button to start the notebook server. Once the new notebook server starts, JupyterLab opens and you are ready to experiment .

If this procedure is successful, you have started your Jupyter notebook server.

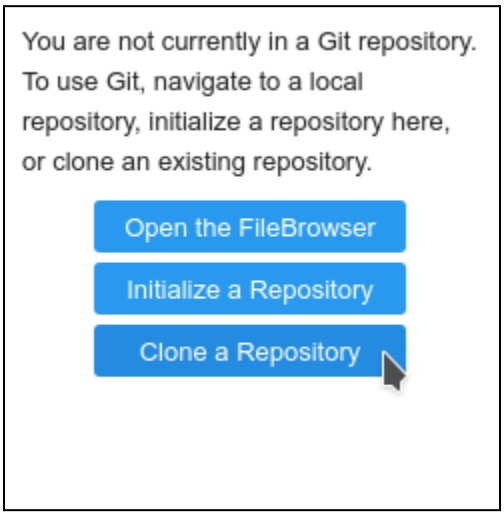
The Jupyter Environment

You are now inside your Jupyter environment. As you can see, it’s a web-based environment, but everything you’ll do here is in fact happening on the Red Hat OpenShift Data Science cluster. This means that without having to install and maintain anything on your own computer, and without disposing of lots of local resources like CPU and RAM, you can still conduct your Data Science work in this powerful and stable managed environment.

In the "file-browser" like window you’re in right now, you’ll find the files and folders that are saved inside your own personal space inside Red Hat OpenShift Data Science. It’s pretty empty right now though… So the first thing we will do is to bring the content of the workshop inside this environment.

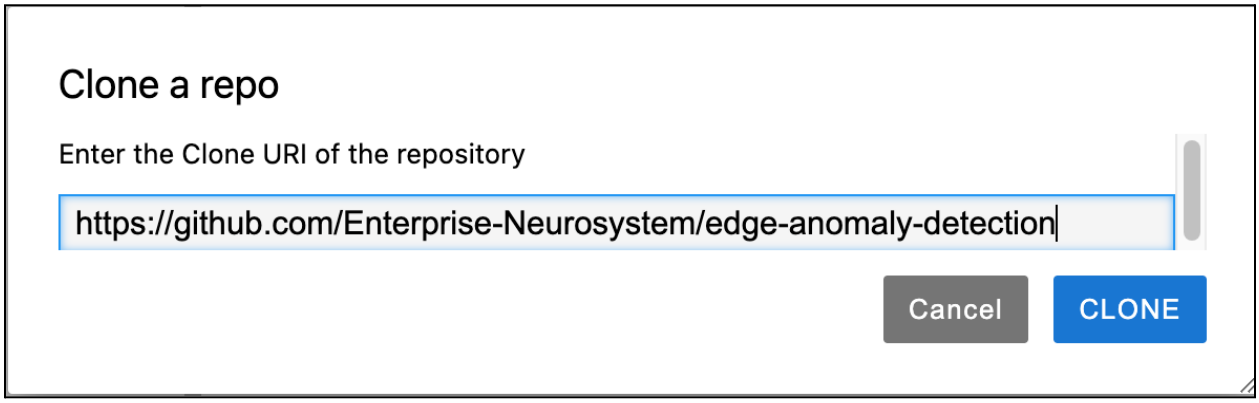
- On the left toolbar, click on the Git icon:

- Then click on Clone a Repository:



- Enter git the URL, `https://github.com/Enterprise-Neurosystem/edge-anomaly-detection`, then click CLONE:

```
https://github.com/Enterprise-Neurosystem/edge-anomaly-detection
```



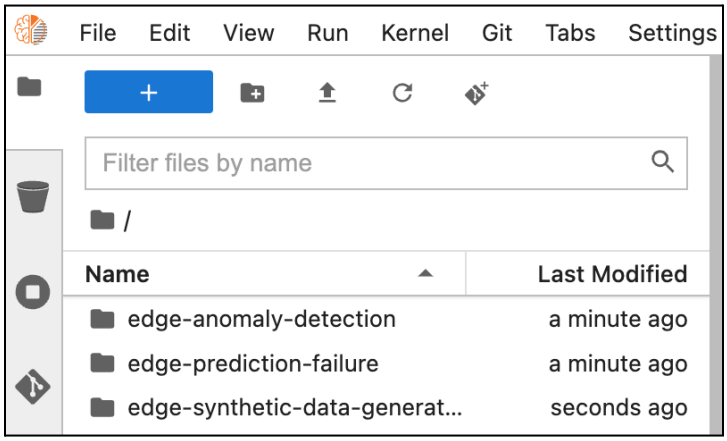
- It takes a few seconds, after which you can double-click and navigate to the newly-created folder, `edge-anomaly-detection`:

Note: After you have cloned the edge-anomaly-detection repository, proceed with cloning the following 2 repositories that will also be used later on in this workshop.

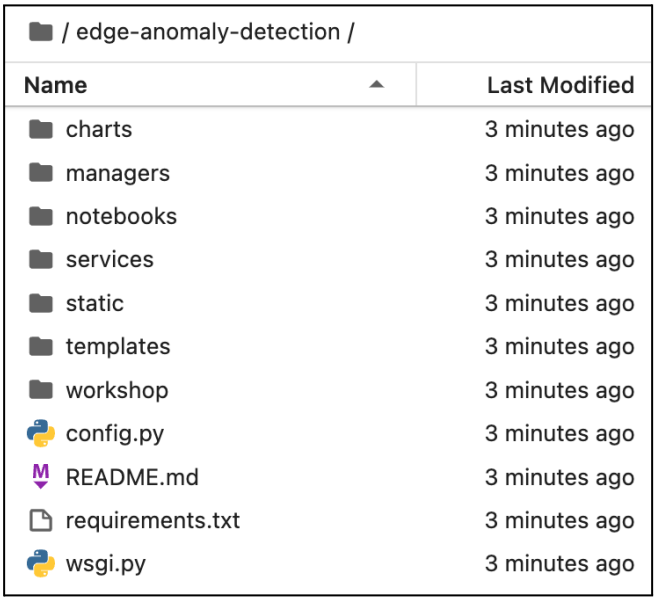
1. https://github.com/Enterprise-Neurosystem/edge-synthetic-data-generator

2. https://github.com/Enterprise-Neurosystem/edge-prediction-failure

After you've cloned the repositories, the `edge-anomaly-detection, edge-synthetic-data-generator and edge-failure-prediction` folders will appear under the **Name** pane.



Let's open up the edge-anomaly-detection folder by double clicking on the edge-anomaly-detection folder name.



The folder should contain the following sub-folders:

- `charts`
- `managers`
- `notebooks`
- `services`
- `static`
- `templates`
- `workshop`

Change into the notebooks sub-folder. You will see the following 2 files: `AnomalyDetectionNotebook.ipynb & Sandbox.ipynb`.

If you have not worked with Jupyter Notebooks before, continue to the next section - Introduction to Jupyter Notebooks.

If you are familiar with Jupyter Notebooks, continue to the section - Anomaly Detection.

Introduction to Jupyter Notebooks

This section provides a small introduction on how to use Jupyter Notebooks. If you're already at ease with Jupyter, you can directly head to the next section - Anomaly Detection.

What's a notebook?

- A notebook is an environment where you have *cells* that can display formatted text, or code.

This is an empty cell:

And a cell where we have entered some code:

- Code cells contain Python code that can be run interactively. Thats means you can modify the code, then run it. The code will not run on you computer or in the browser, but directly in the environment you are connected to, Red Hat OpenShift Data Science in our case.
- To run a code cell, just select it (click in the cell, or on the left side of it), and click the Run/Play button from the toolbar (you can also press CTRL+Enter to run a cell, or Shift+Enter to run the cell and automatically select the following one).

The Run button on the toolbar:

Our cell after pressing Run:

As you can see, you have the result of the code that was run in that cell, as well as information on when this particular cell has been run.

- When you save a notebook, the code as well as the results are saved! So you can always reopen it to look at the results without having to run all the program again, while still having access to the code.

Notebooks are so named because they are just like a physical Notebook: it's exactly like if you were taking notes about your experiments (which you will do), along with the code itself, including any parameters you set. You see the output of the experiment inline (this is the result from a cell once it's run), along with all the notes you want to take (to do that, switch the cell type from the menu from `Code` to `Markup`).

Time to play

Now that we have covered the basics, just give it a try!

- In your Jupyter environment (the file explorer-like interface), there is a file called `Sandbox.ipynb`. Double-click on it to launch the notebook (it will open another tab in the content section of the environment). Please feel free to experiment, run the cells, add some more and create functions. You can do what you want - it's your environment, and there is no risk of breaking anything or impacting other users. This environment isolation is also a great advantage brought by Red Hat OpenShift Data Science.
- You can also create a new notebook by selecting `File`->`New`->`Notebook`from the menu on the top left, then select a Python 3 kernel. This instructs Jupyter that we want to create a new notebook where the code cells will be run using a

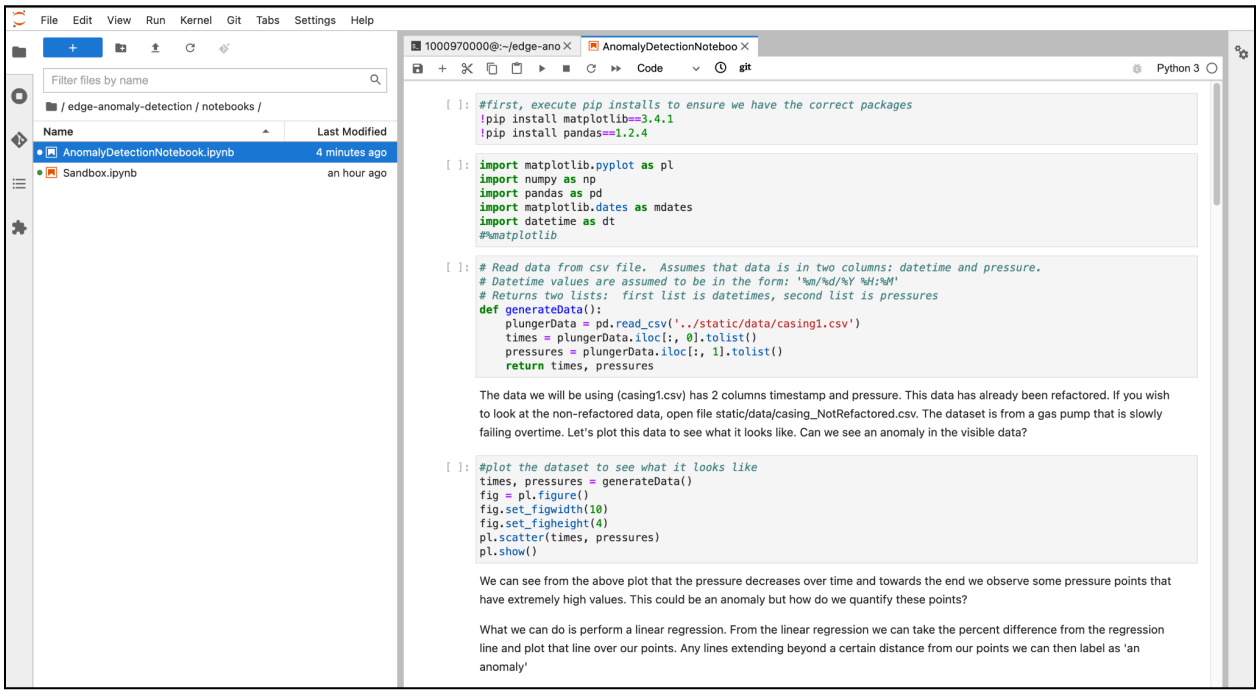
Python 3 kernel. We could have different kernels, with different languages or versions that we can run into notebooks, but that's a story for another time…

- You can also create a notebook by simply clicking on the icon in the launcher:

- If you want to learn more about notebooks, head to this page. Now that you're more familiar with notebooks, you're ready to go to the next section - Anomaly Detection.

Anomaly Detection

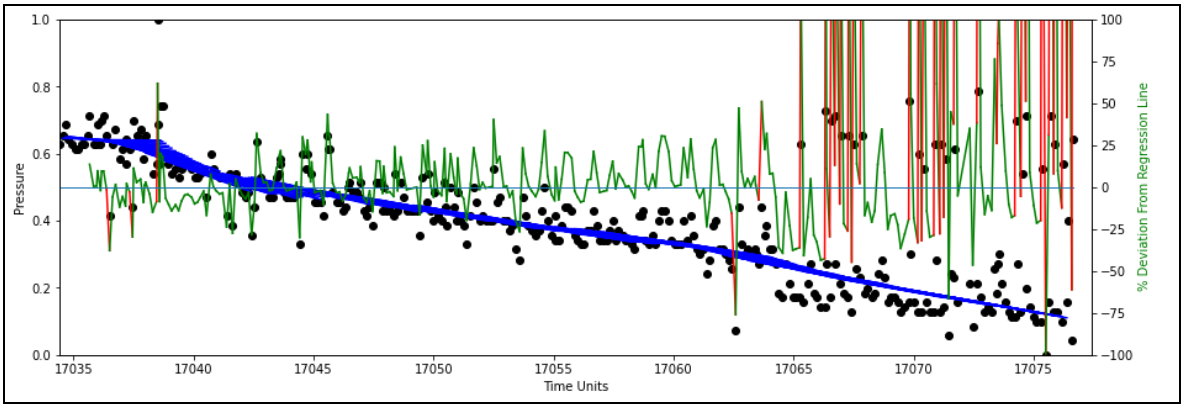
If you have not already done so, change into the notebooks sub-folder. You will see the following 2 files: `AnomalyDetectionNotebook.ipynb & Sandbox.ipynb`. Double click `AnomalyDetectionNotebook.ipynb` to open the notebook.



In this notebook, we'll explore a very small data set (2 columns consisting of timestamp and pressure) so that you can see what 'sensor' data looks like. The dataset is in csv file format.

You will work through this notebook, reading the explanations and executing the notebook's cells. Do this now.

The last notebook cell you execute will produce the following Anomaly Detection plot.



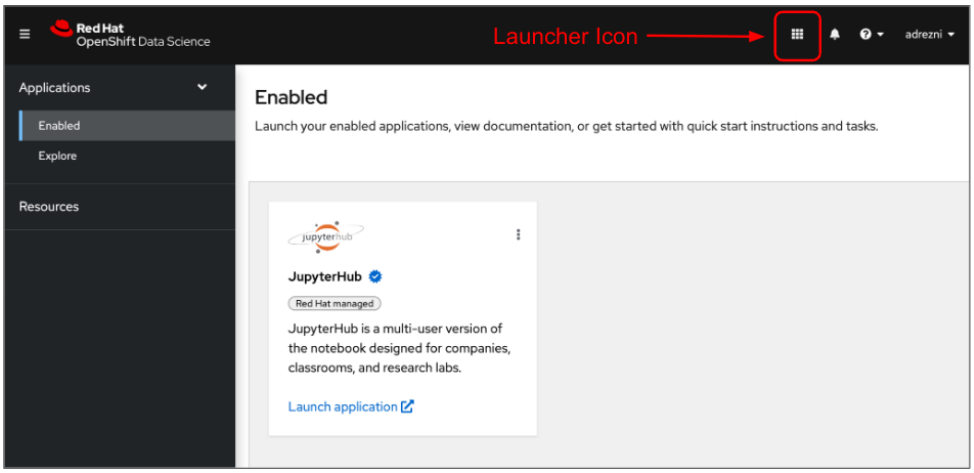
In this plot, the far right hand side of the plot is not only an anomaly, it is also the failure of a mechanical pump. Hypothesis: If we can detect anomalies we can then use this knowledge to predict when a mechanical pump will fail.

Now that you have an idea of what sensor anomalies look like and how to detect them, let's take a look at how we can predict failures. Continue to the next section, Failure Prediction Web Application.

Failure Prediction Web Application

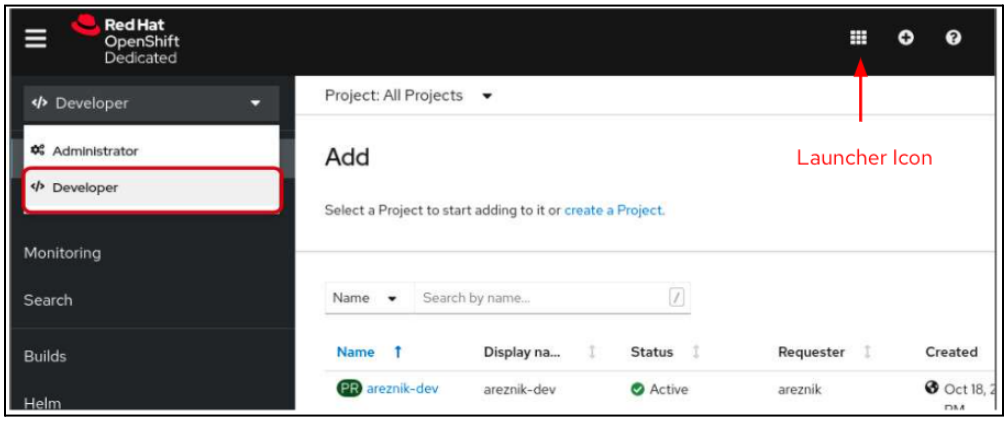
We need to containerize and deploy the Failure Prediction Web Application code. We will package the code as a container image and run it directly in OpenShift as a Web Application.

From the Red Hat OpenShift Data Science platform choose the Launcher Tool and Select "OpenShift Console"

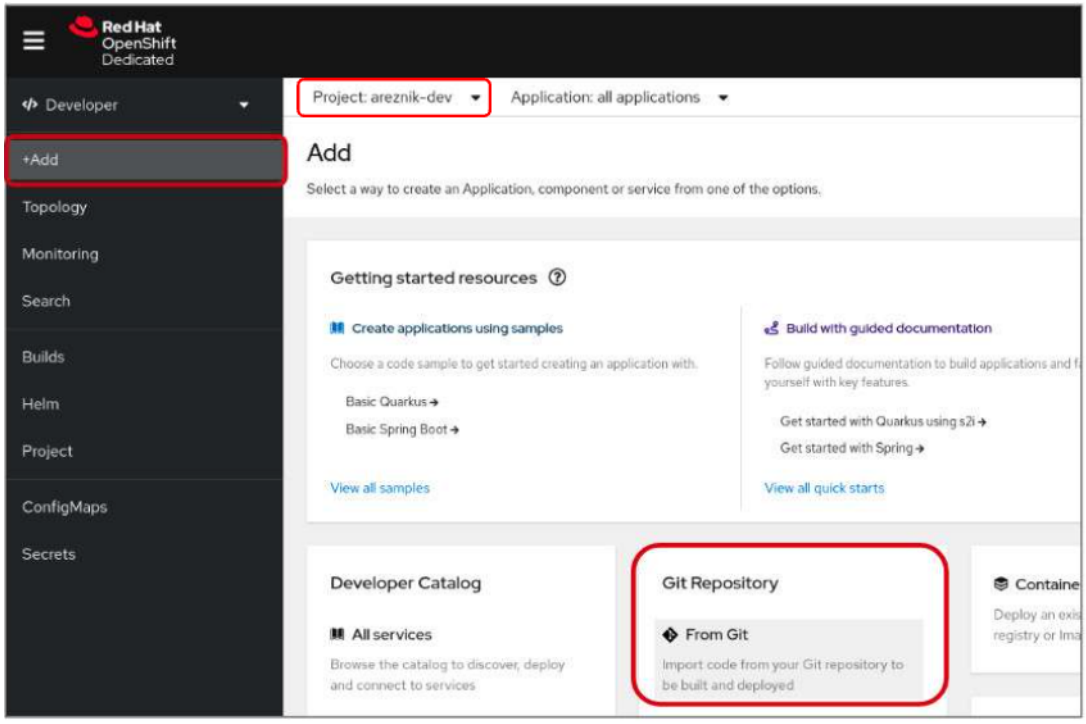


Within the OpenShift console, make certain you are in ‘Developer’ mode.

Note: The Launcher Icon is also visible in the OpenShift console. You can use this icon to move between the Red Hat OpenShift Data Science Platform and the Red Hat OpenShift Dedicated platform.

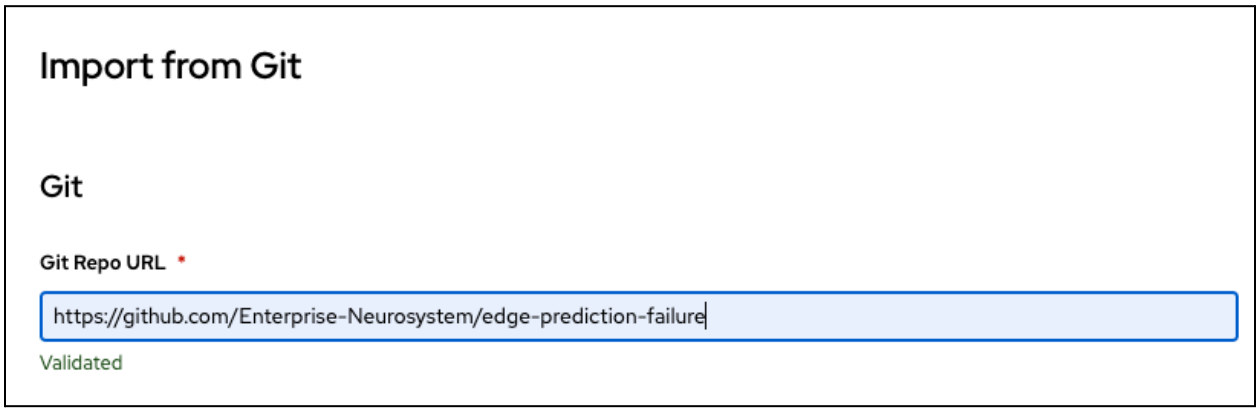


Next, make certain that you are in the ‘Project’ that is assigned to you. The ‘Project’ is your work area within OpenShift. The project name will be your userid that you were given at the beginning of the workshop. In the following screenshot, the project name is: areznik-dev

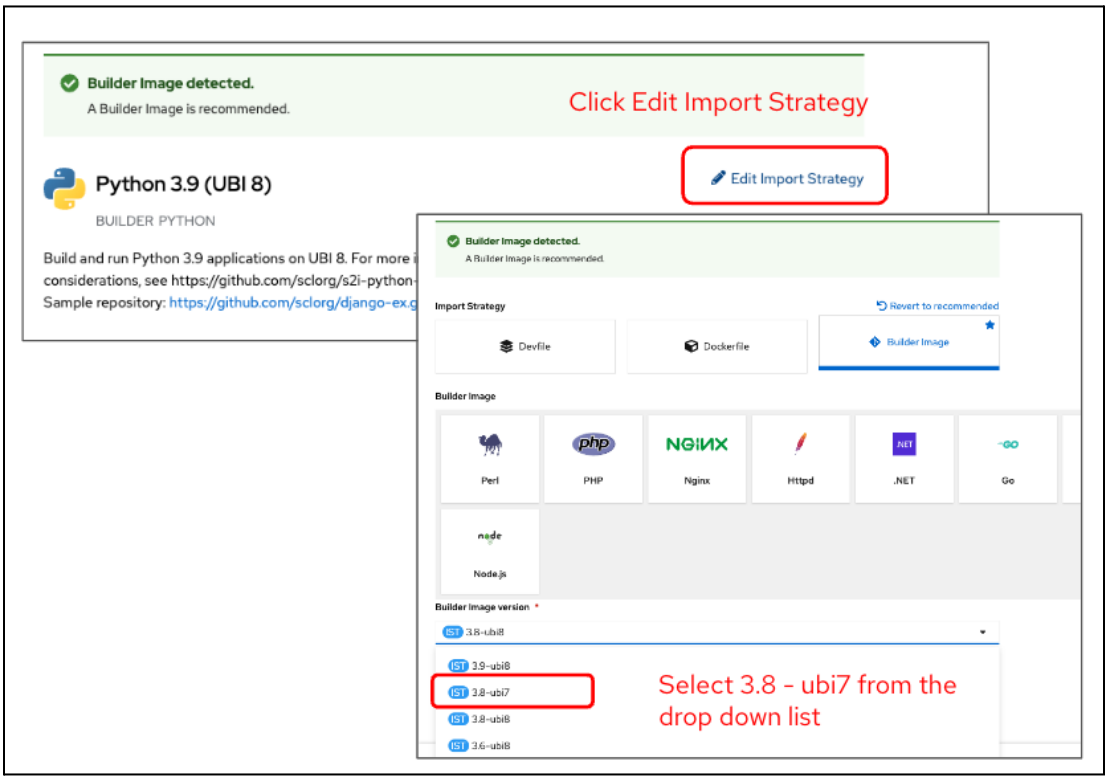


Once we are in the correct project we will begin to create our container. From the +Add menu, click the ‘From Git’ option.

In the Git Repo URL Field, enter:
https://github.com/Enterprise-Neurosystem/edge-prediction-failure.git

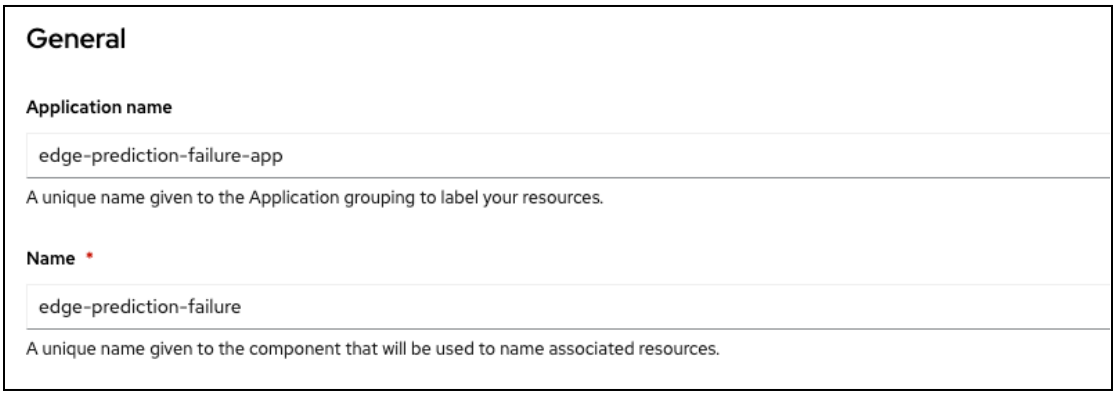


Next, change the BUILDER PYTHON to Python 3.8 (UBI7). Click ‘Edit Import Strategy’, then select `3.8 - ubi7` from the drop down list.

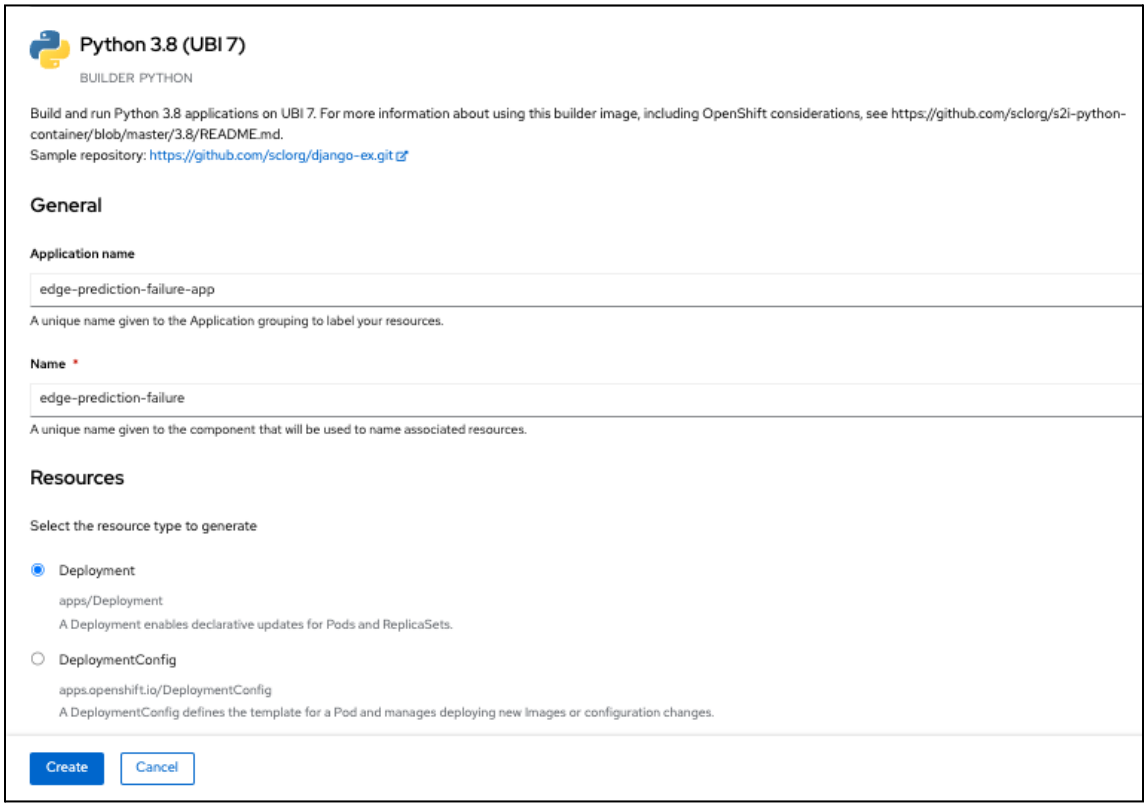


If you continue to scroll down the page, you will see that everything is automatically selected to create a deployment of your application, as well as a route through which you will be able to access the application.

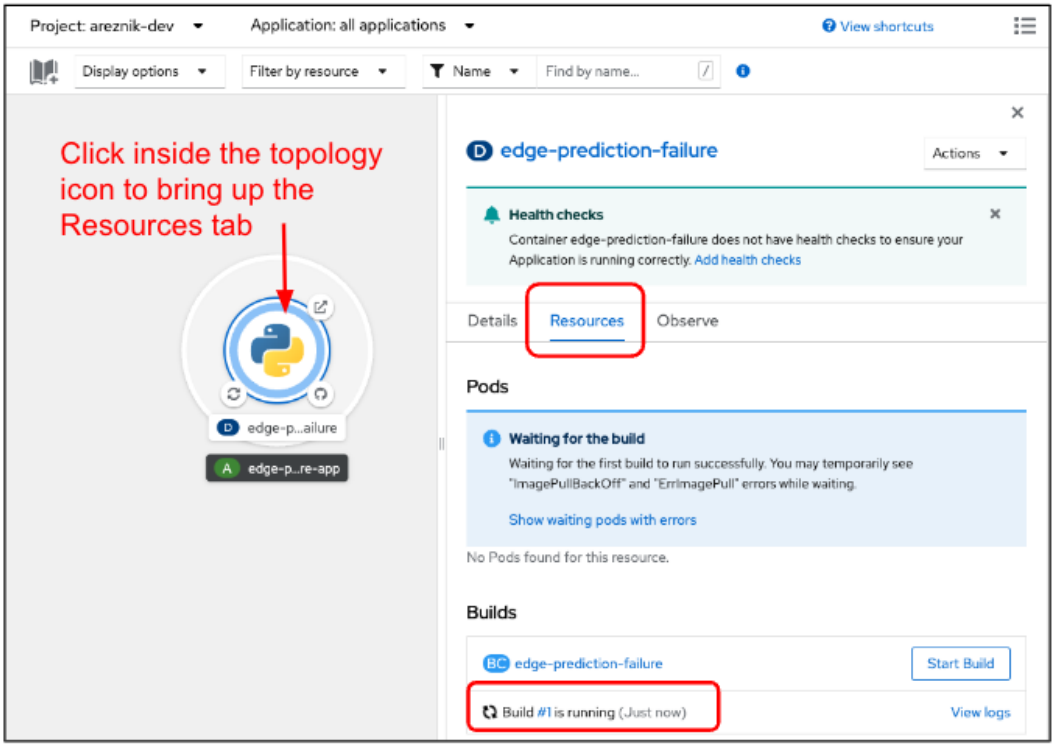
Make certain to name your app. For example, edge-failure-prediction



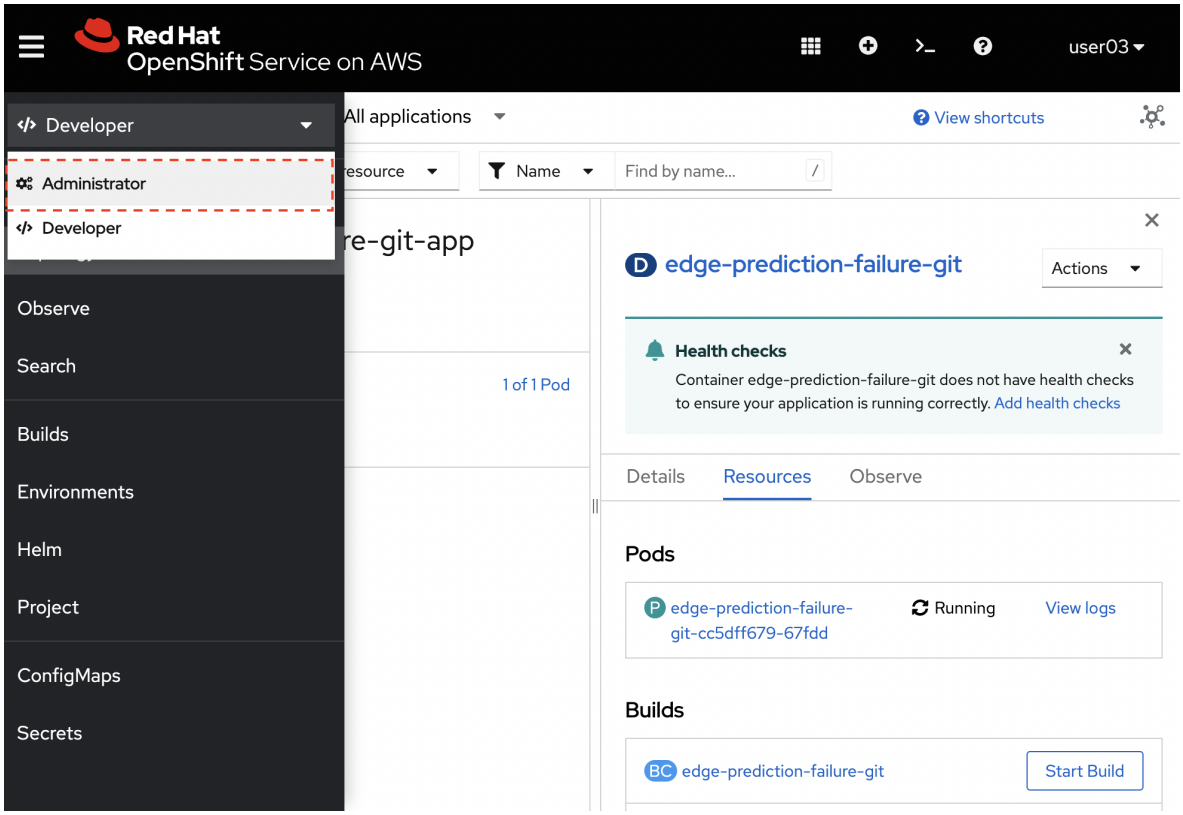
Now we are ready to press the ‘Create’ button to create our containerized application. Press the ‘Create’ button.



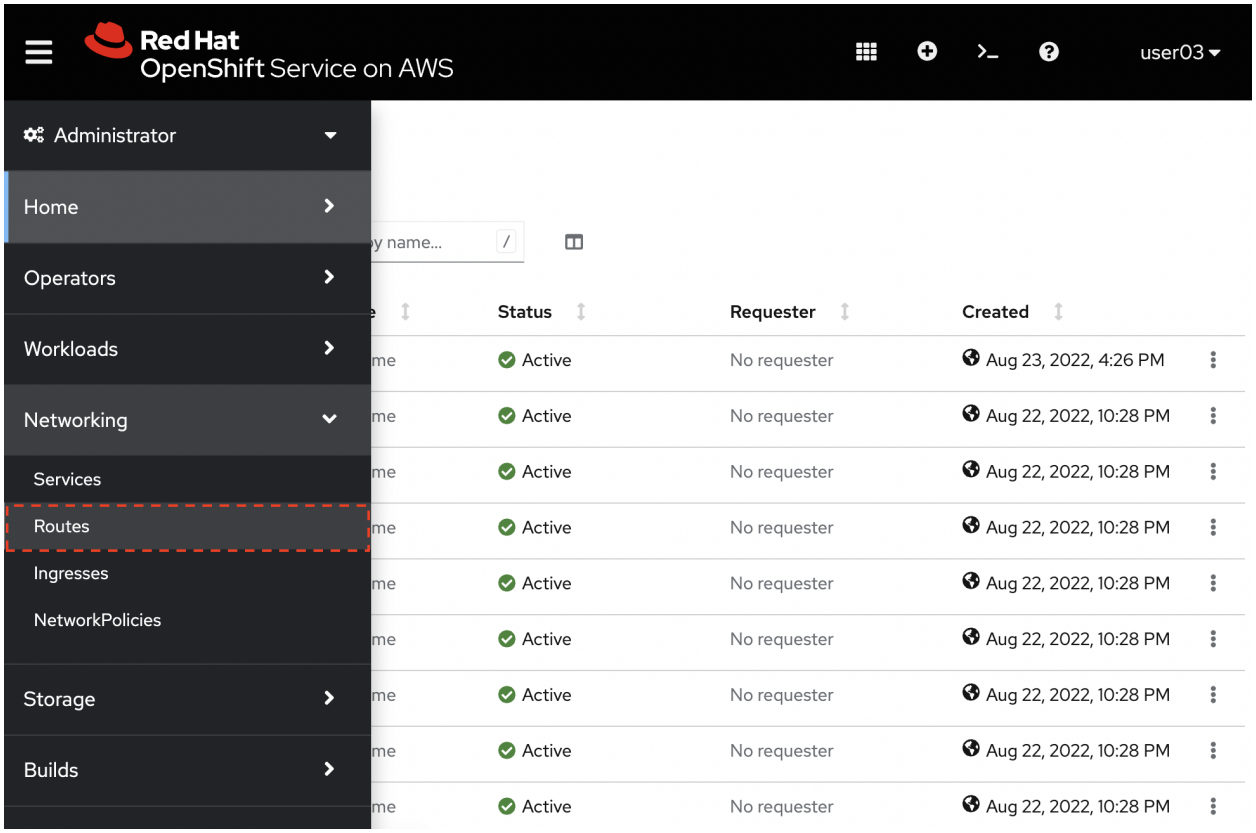
The automated process will take a few minutes. Some alerts may appear if OpenShift tries to deploy the application while the build is still running, but that's OK. Then OpenShift will deploy the application (rollout), and in the topology view, you should see a screen similar to the following screen capture.



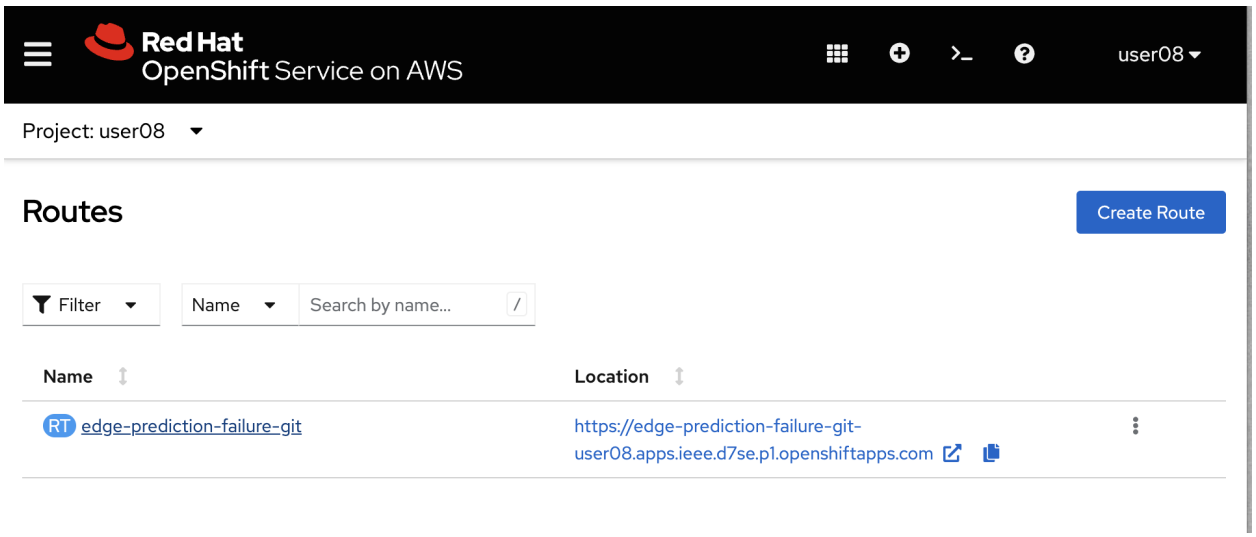
Before we view the containerized application, we first have to annotate the routes to increase the time out. Switch the view from ‘Developer’ to ‘Administrator’.



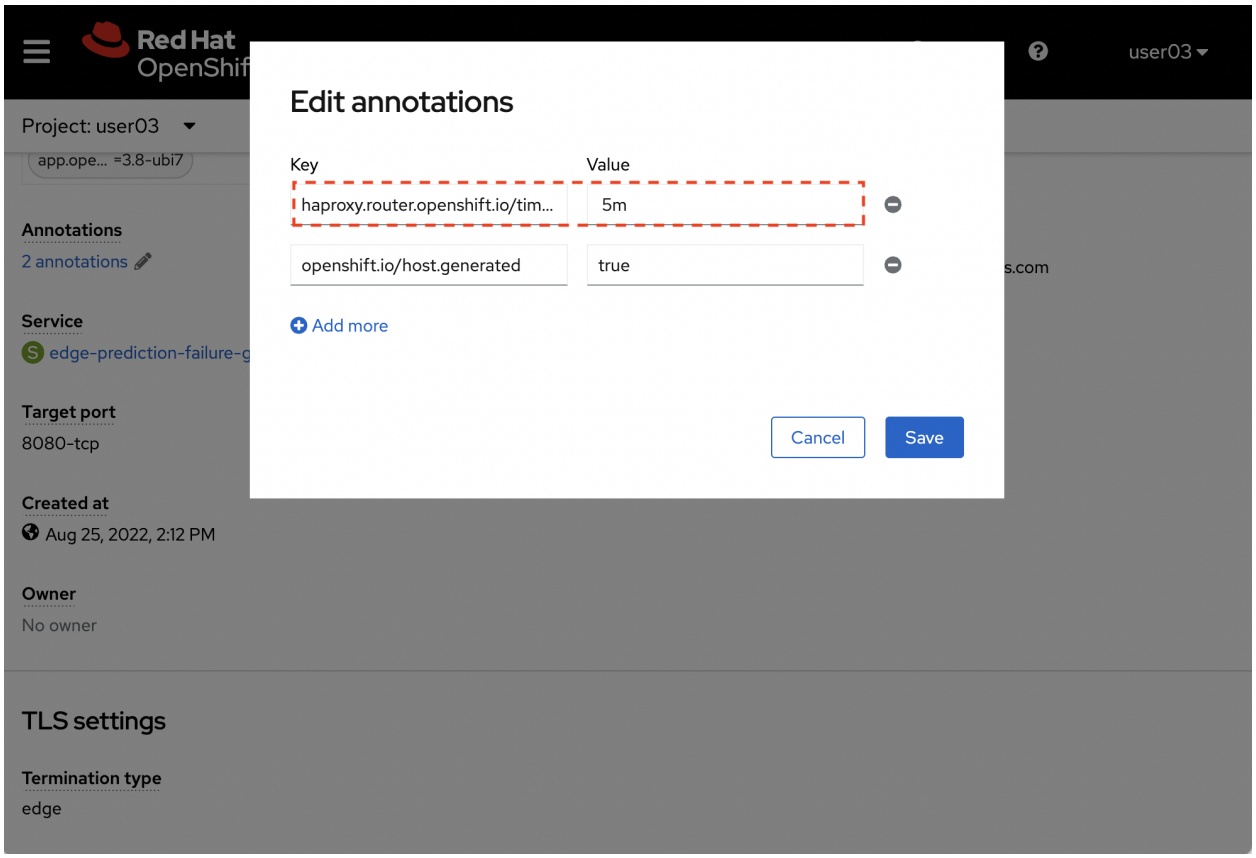
Navigate to ‘Networking’ and then ‘Routes’ on the side bar.



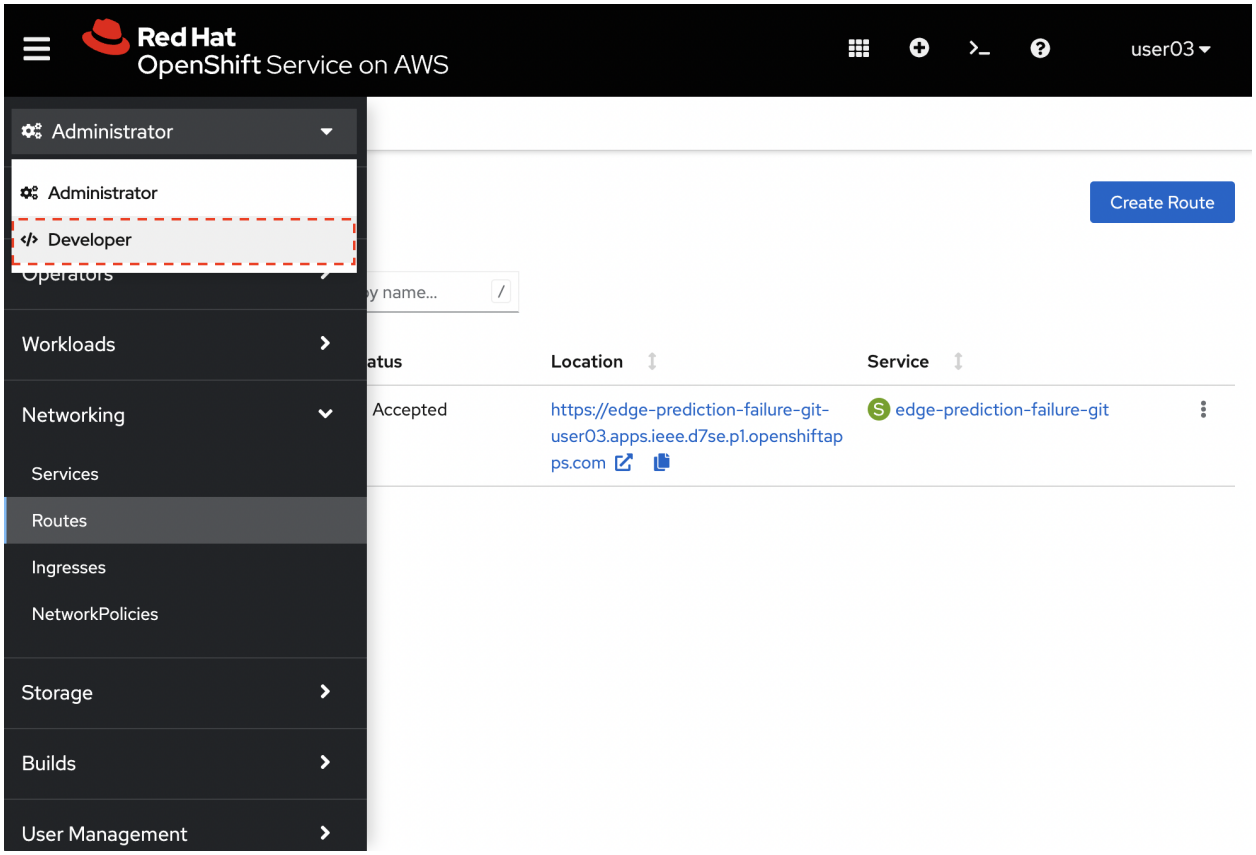
Click on the project name ‘edge-prediction-failure’ and then scroll down to ‘Annotations’.

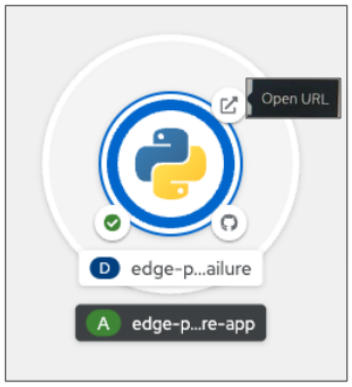


Click on the pencil icon to edit the annotations and then add the following key-value pair: haproxy.router.openshift.io/timeout **: 5m** and click on 'Save'. We are doing this to increase the timeout so that our web application does not timeout before the model is fully trained.

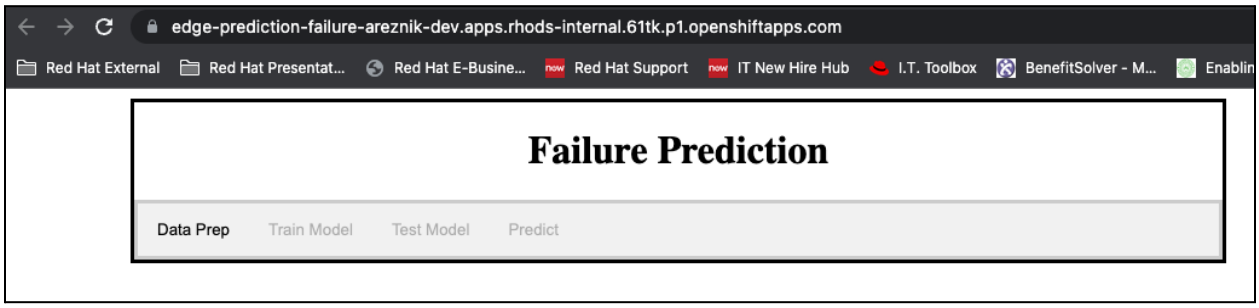


Now we can head back to our ‘Developer’ view and display our containerized application in a browser by clicking on the URL icon in the Topology view.





Your containerized Failure Prediction application will now appear in a browser window.



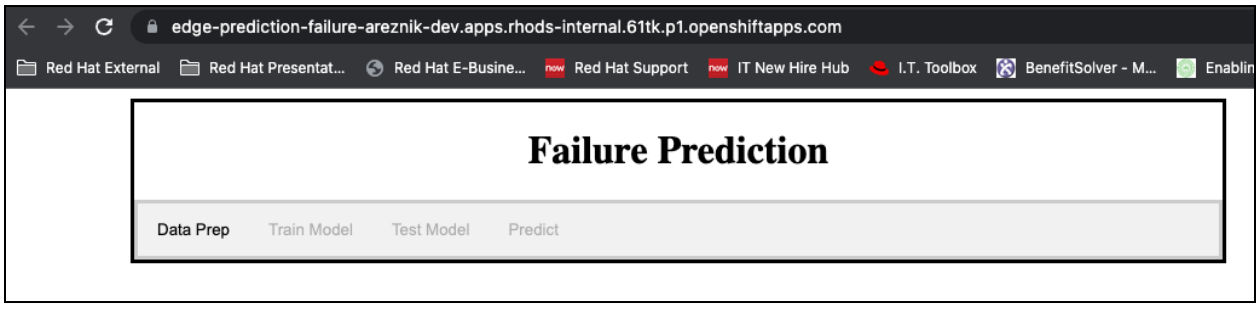
This application serves as a wizard that allows the user to:

- Prepare data for training, testing and prediction
- Train the model
- Test the model
- Make predictions on data that the user chooses

The goal of this application is to ultimately use a trained model that has learned what failures look like to specify there is a possibility of an imminent failure.

Opening Screen

When you first open the application, you will see **tabs**, some of which are initially disabled:



This user interface will guide you through all the processes from Data Preparation through Predictions and will make sure that you perform all processes in the correct order, just like a wizard.

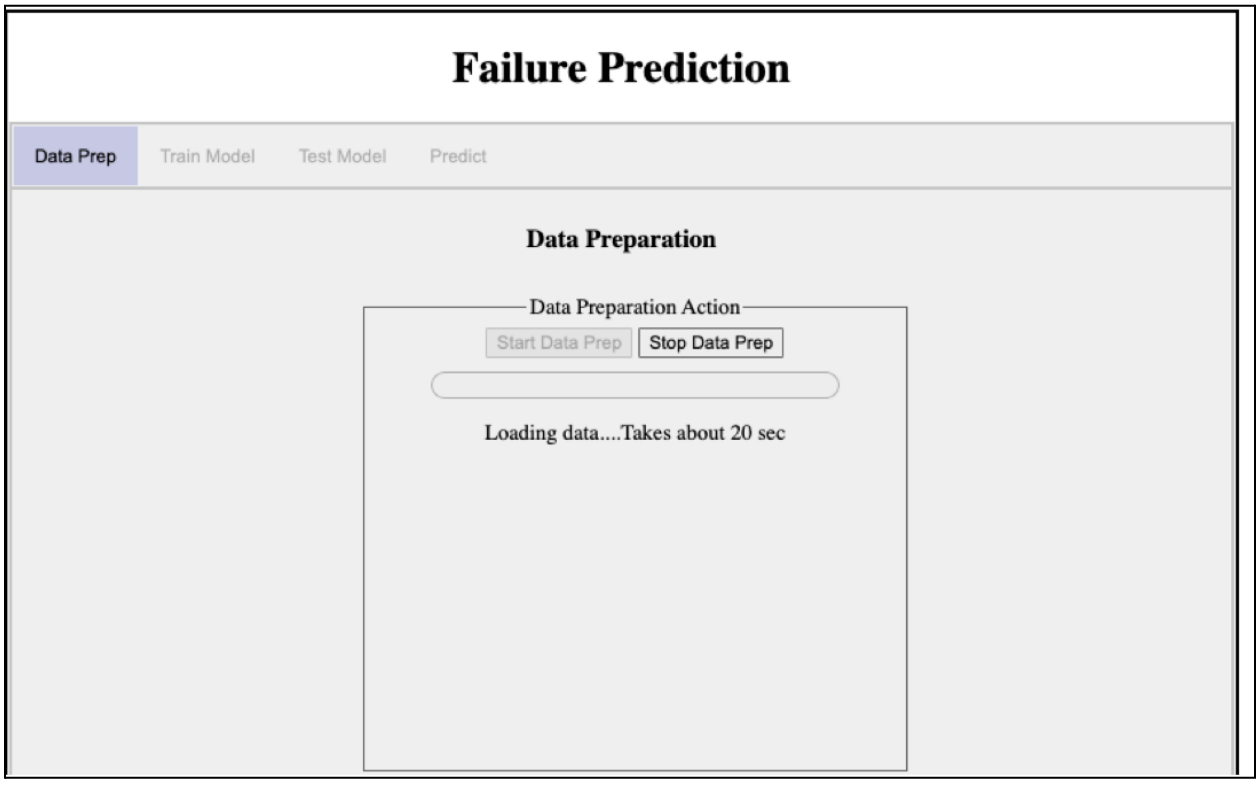
The gray bar with the labels Data Prep, Train Model, Test Model, Predict serve as navigation tabs. When you click on a tab, the relevant screen will appear. Think of the tabs as menu selections that change the interface to suit the job at hand.

Data Preparation

When the application first loads in your browser, a large amount of data is loaded.

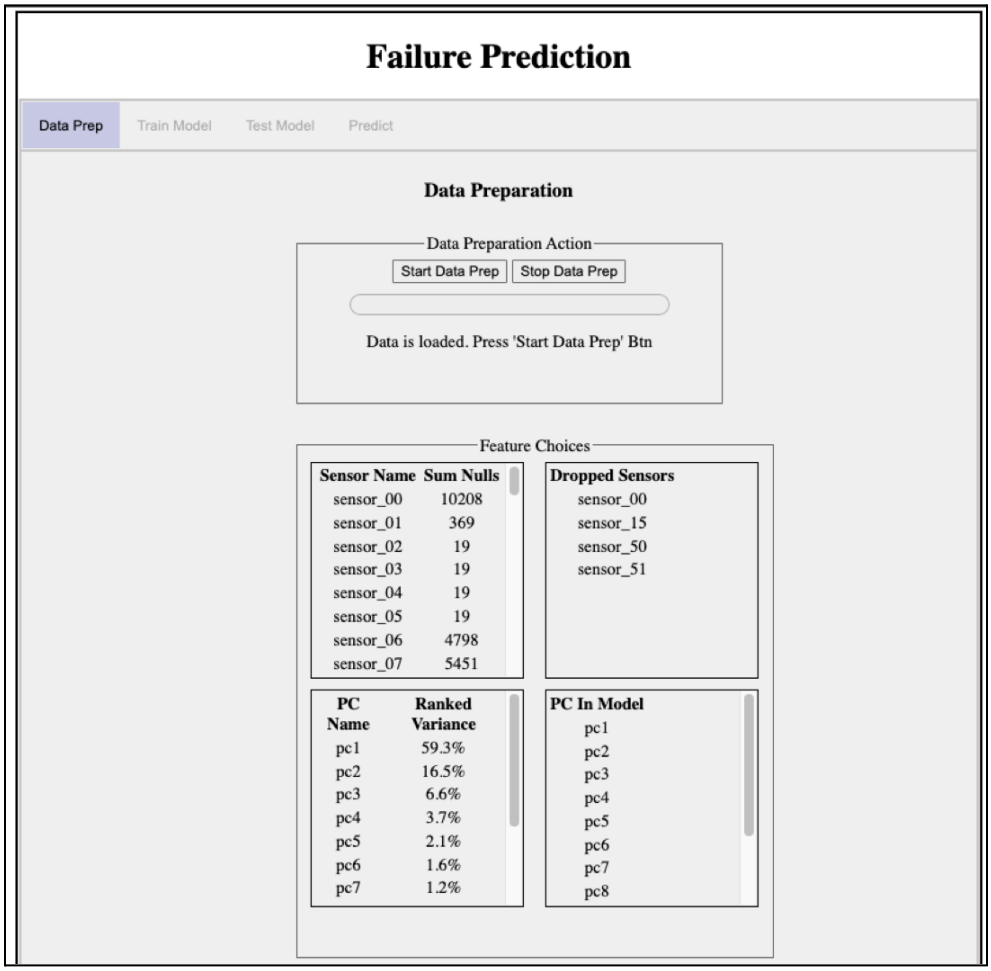
Click on the Data Prep tab.

If you click on the Data Prep tab before all the data has been loaded, you will see:



The message: Loading data…. means that you cannot proceed past this page until the data are loaded. Notice that the Start Data Prep button is disabled, and will stay disabled until the data are loaded.

Once loaded, the Start Data Prep button will be enabled.



The information that you see in the box labeled Feature Choices refers to what you discovered in the start of the workshop. That is, you did some data discovery. The box in the upper left shows how many nulls were found in each sensor column. That is why, for example, the box in the upper right shows a list of the sensors that will be dropped, including sensor_00.

The box in the lower left shows a list of results from the Principal Component Analysis (PCA) that you found in the data discovery. In the original data there were about 52 sensors. We would like to shorten this list to make the model training take less time. Instead of going through a long process of deciding which sensors to keep, we use the

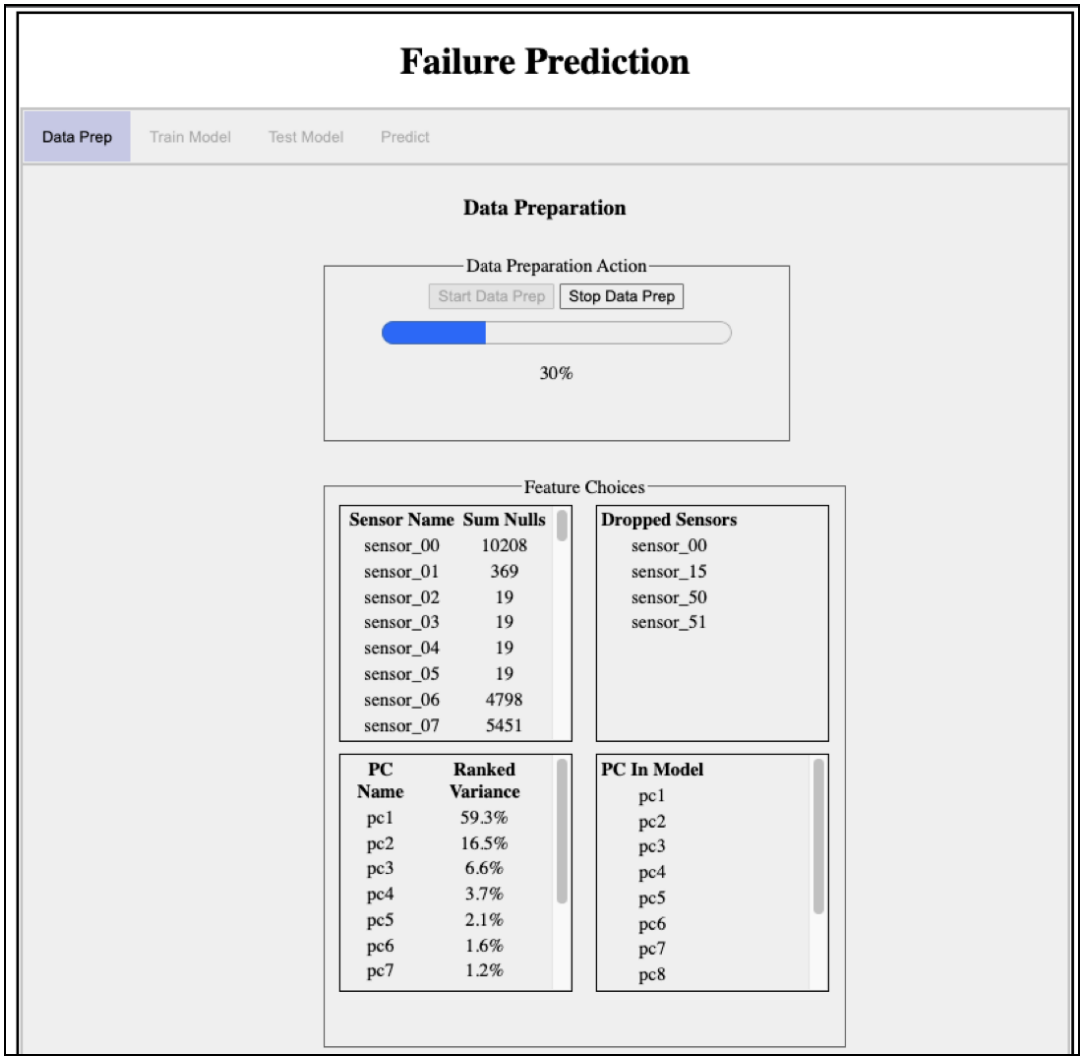
PCA that finds which linear combinations of all the sensors gives the most variance. The linear combinations are denoted by pc1, pc2, pc3, …, (called Principal Components) where pc1 is the first ranked. We then take however many pc's we need so that the sum of their ranked variances is at least 95%. In our case it takes 12 pc's to reach that goal. We then use those 12 pc's to represent a transformation of our original 52 sensors when we train the model.

Now that we have all the data loaded and the PCA transformation is complete, we are ready to proceed with preparing the data.

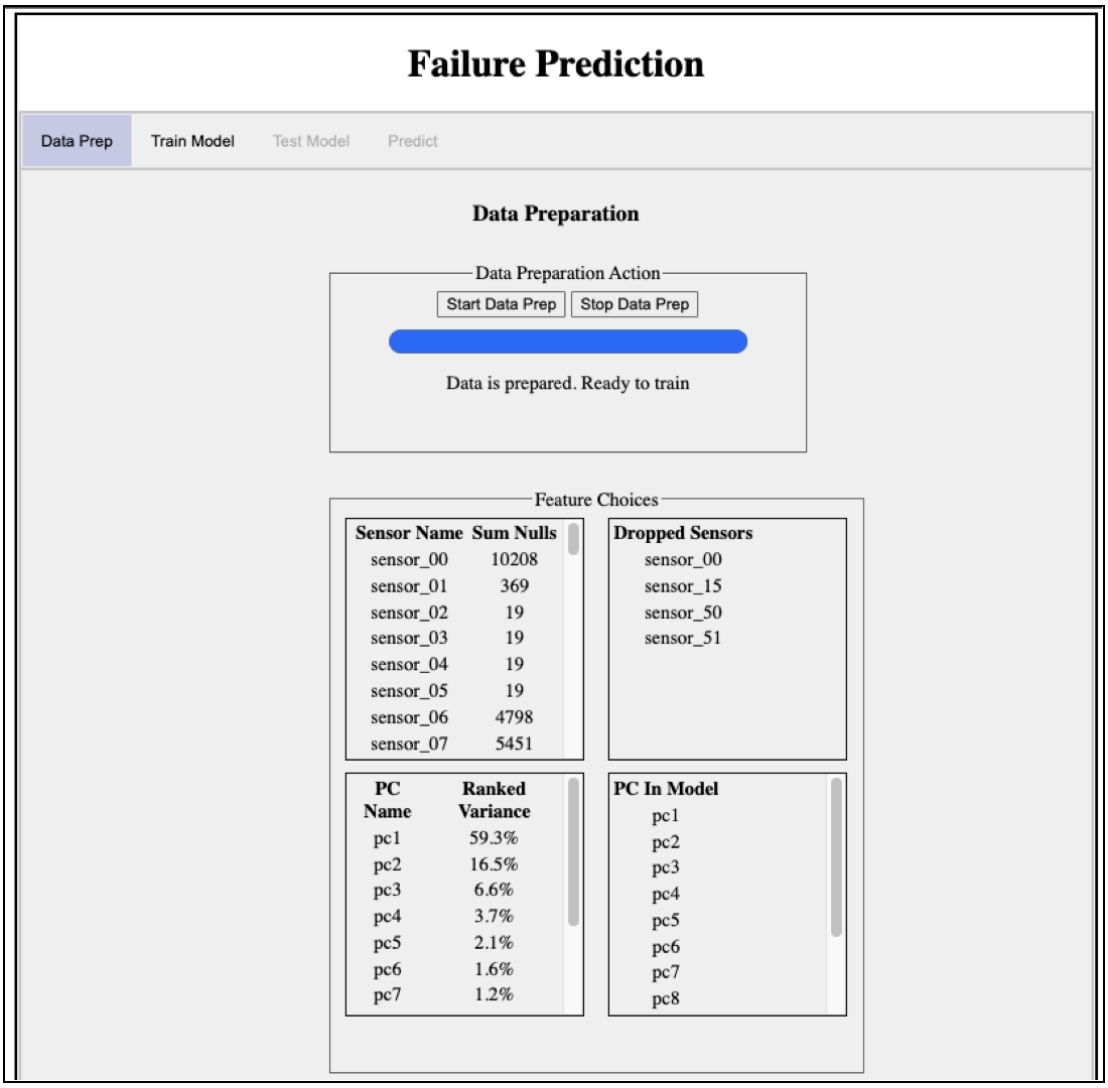
Click on the Start Data Prep button.

Each row of data that we will use has a timestamp together with the 12 pc's that we will now call model features. Data that is ordered by time is called a Time Series.

In the background, the application is reshaping the time series data so that it will be put into a form that the model understands. While this processing is taking place you will see:

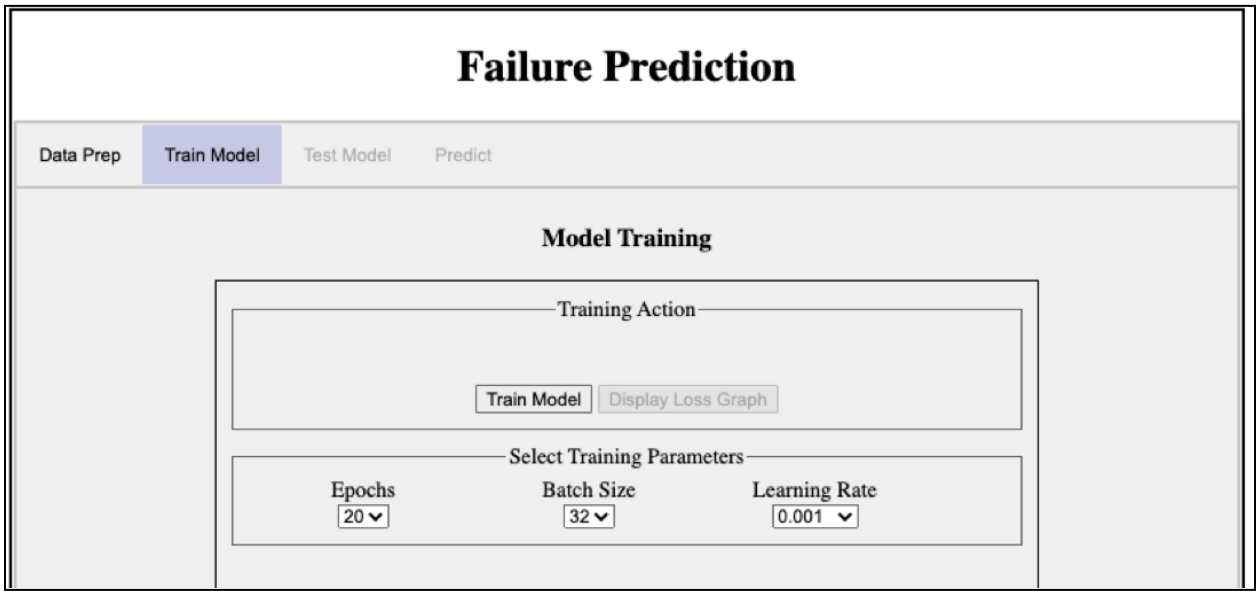


The progress bar will start slowly and at about 50% will speed up. When the data preparation is complete, a message will appear and the next tab, Train Model tab will be enabled:


Failure Prediction
Data Prep
Train Model
Test Model
Predict
Data Preparation
Data Preparation Action
Start Data Prep
Stop Data Prep
Data is prepared. Ready to train
Feature Choices
Sensor Name Sum Nulls
sensor_00 10208
sensor_01 369
sensor_02 19
sensor_03 19
sensor_04 19
sensor_05 19
sensor_06 4798
sensor_07 5451
Dropped Sensors
sensor_00
sensor_15
sensor_50
sensor_51
PC Name Ranked Variance
pc1 59.3%
pc2 16.5%
pc3 6.6%
pc4 3.7%
pc5 2.1%
pc6 1.6%
pc7 1.2%
PC In Model
pc1
pc2
pc3
pc4
pc5
pc6
pc7
pc8

Training the Model

Click on the Train Model tab.

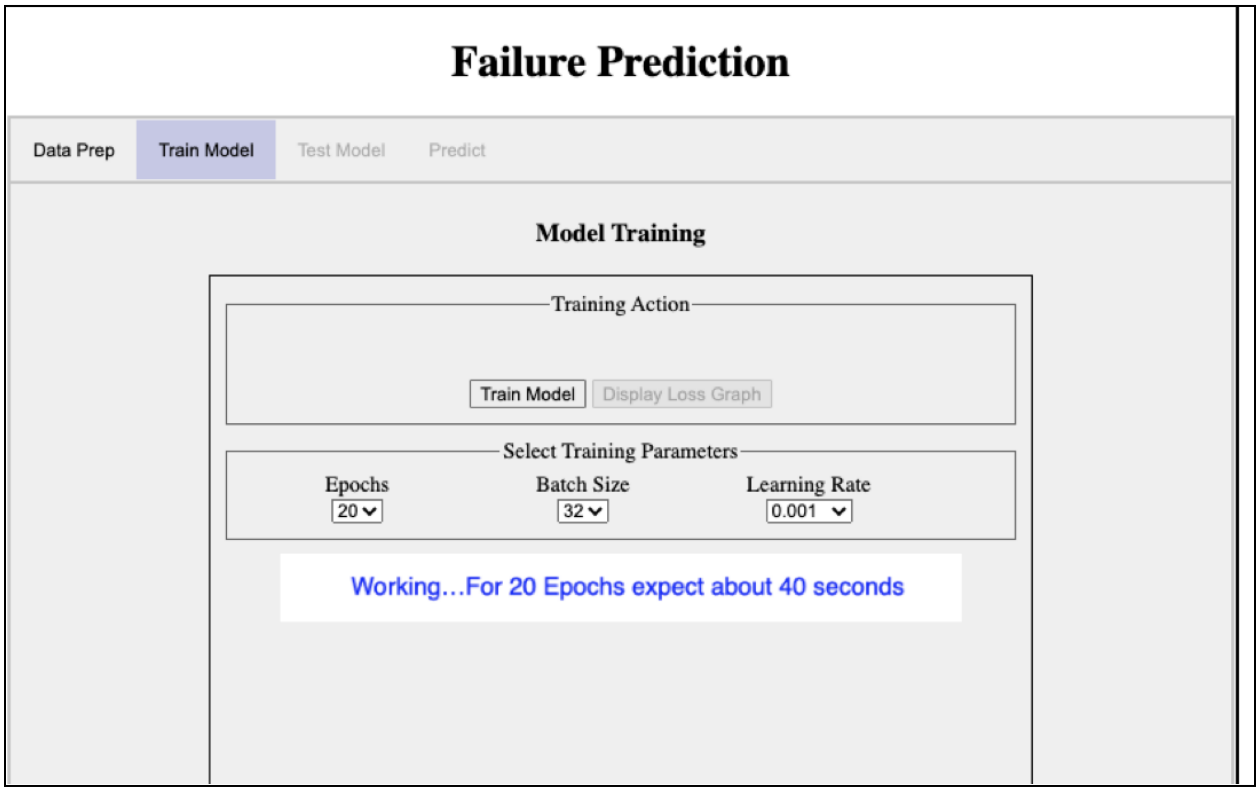


The training process of a model is sometimes called the learning process. This learning process is iterative. A model is a numerical approximation of an entity, and the training data are used to calculate coefficients of the approximation. The training data contain values for the features (input) together with values for the target (output). So during one pass (called an **epoch**) on the data, small samples of the training data (called a **batch**) are used to update the model's coefficients. During training, a parameter called the learning rate is used to determine the "pace" of the iterative steps used to update the coefficients. If the learning rate is too large, then the model may step over a critical part of the updating process. If too small, the updates may cause the model to converge too quickly and possibly miss the optimal approximation. In this training session, the learning rate starts as the value in the pull down, but after 10 epochs its value is divided by a factor of 10.

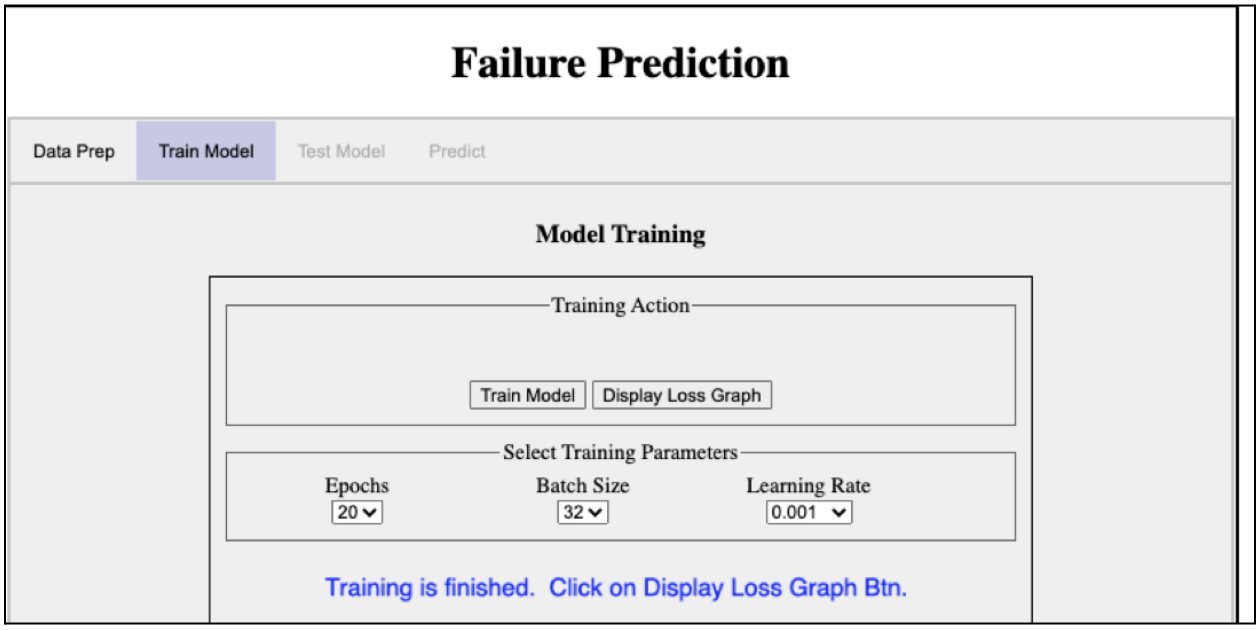
The default values in the pull downs are better left alone for this workshop.

Click on the Train Model button to start the training.

During training, there is no progress feedback. This is because the fit() function only gives progress in the form of text displayed in a terminal console. Since a web application has no console, there is no progress feedback available.

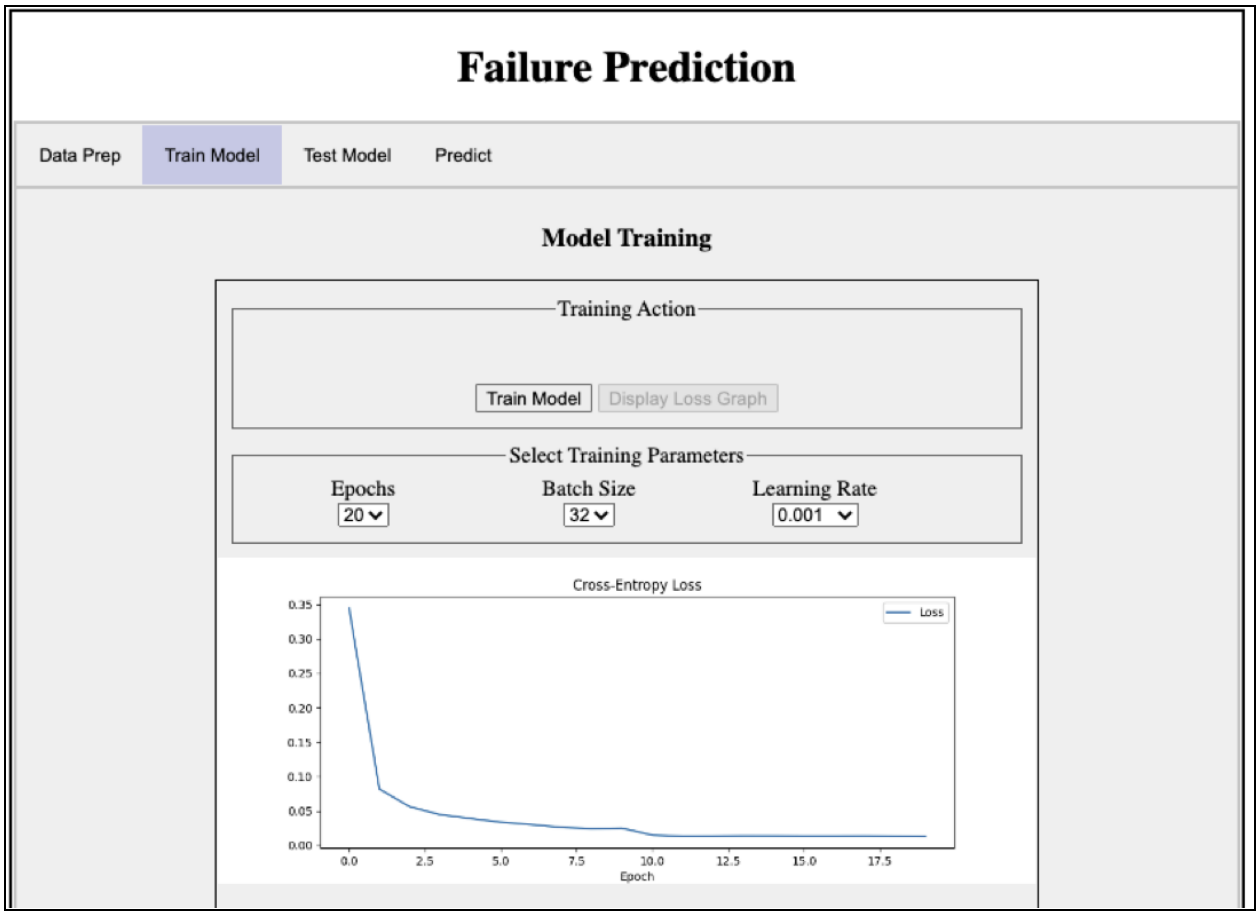


When the training is finished you will see:



Notice that the Display Loss Graph button is now enabled.

Click on the Display Loss Graph button and we will see a graph showing the Loss Graph:



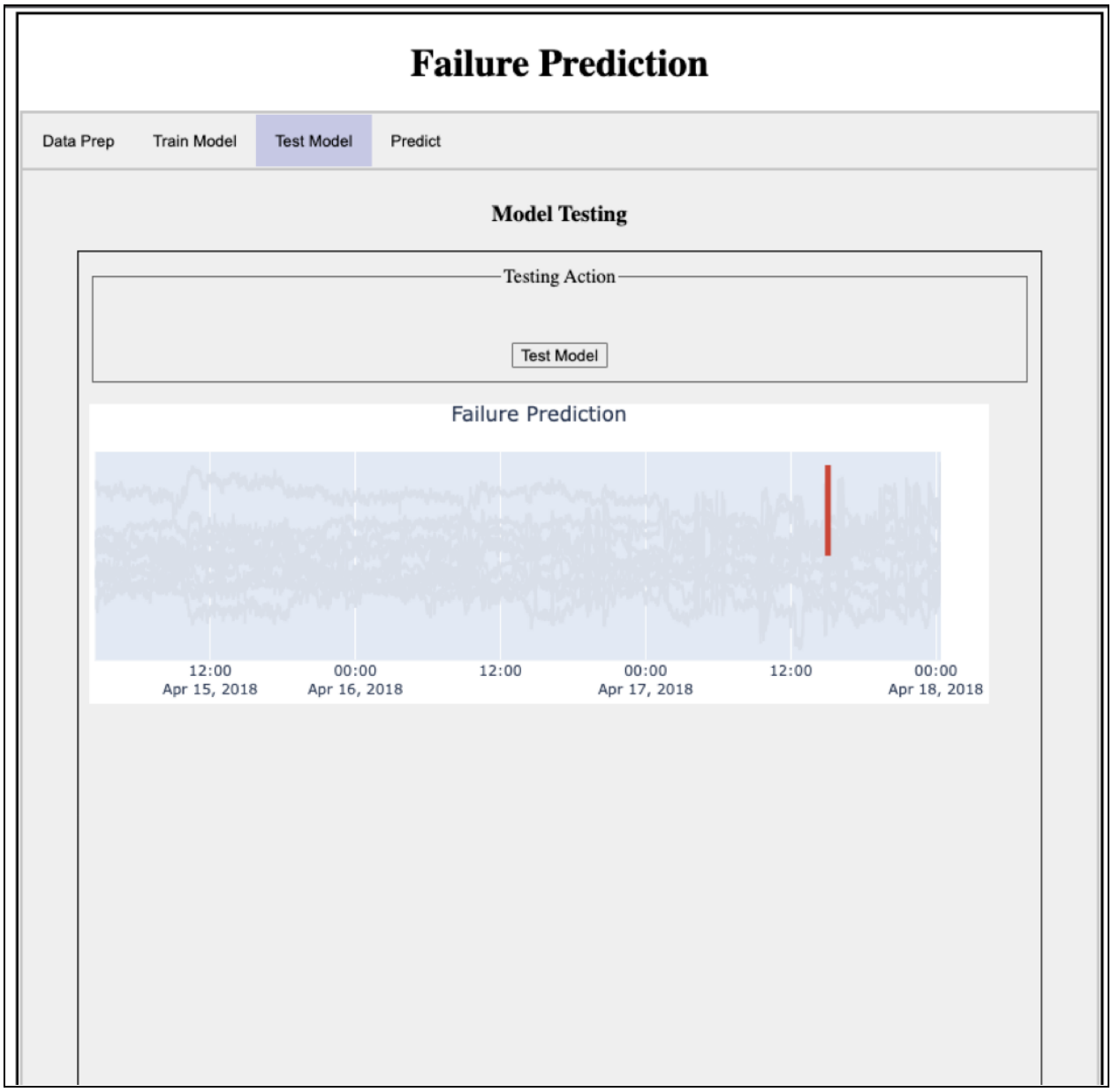
Besides the graph you will notice that the two tabs labeled Test Model and Predict have been enabled. We will run the Test first, although at this point we could just as well run a prediction.

Testing the Model

Click on the Test Model tab.

The Test Model tab is sparse because there are no parameters to define for training.

Just **click on the Test Model button.** The result is something like:



Do not be surprised if your neighbor gets a slightly different result. This is because of all the moving parts of the process, that is, because of the stochastic nature of training the model. Remember that a model is an approximation and the training involves uncertainties and randomness in order to make the approximation.

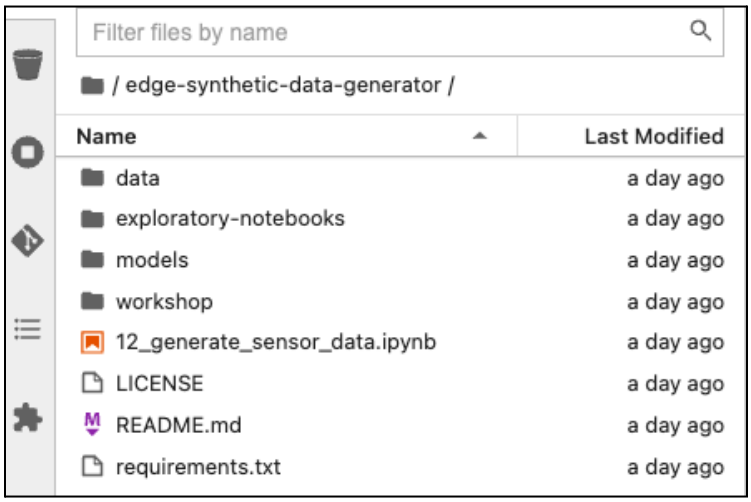
Before we can look at predicting mechanical pump failures in real time, we need to have real time mechanical pump sensor data to work with. For this workshop, we will not be connecting to a real time mechanical pump. Therefore; to simulate real time mechanical pump sensor data, we created a synthetic data generator. This generator

will allow you to create real-time synthetic data that our failure prediction model can ingest and determine if a mechanical failure is imminent. Let's look at how we generate Synthetic Data. Continue to the next section - Generate Synthetic Data.

Generate Synthetic Data

We are now going to generate synthetic data that we will use in our Failure Prediction Model. Navigate back to your JupyterLab environment.

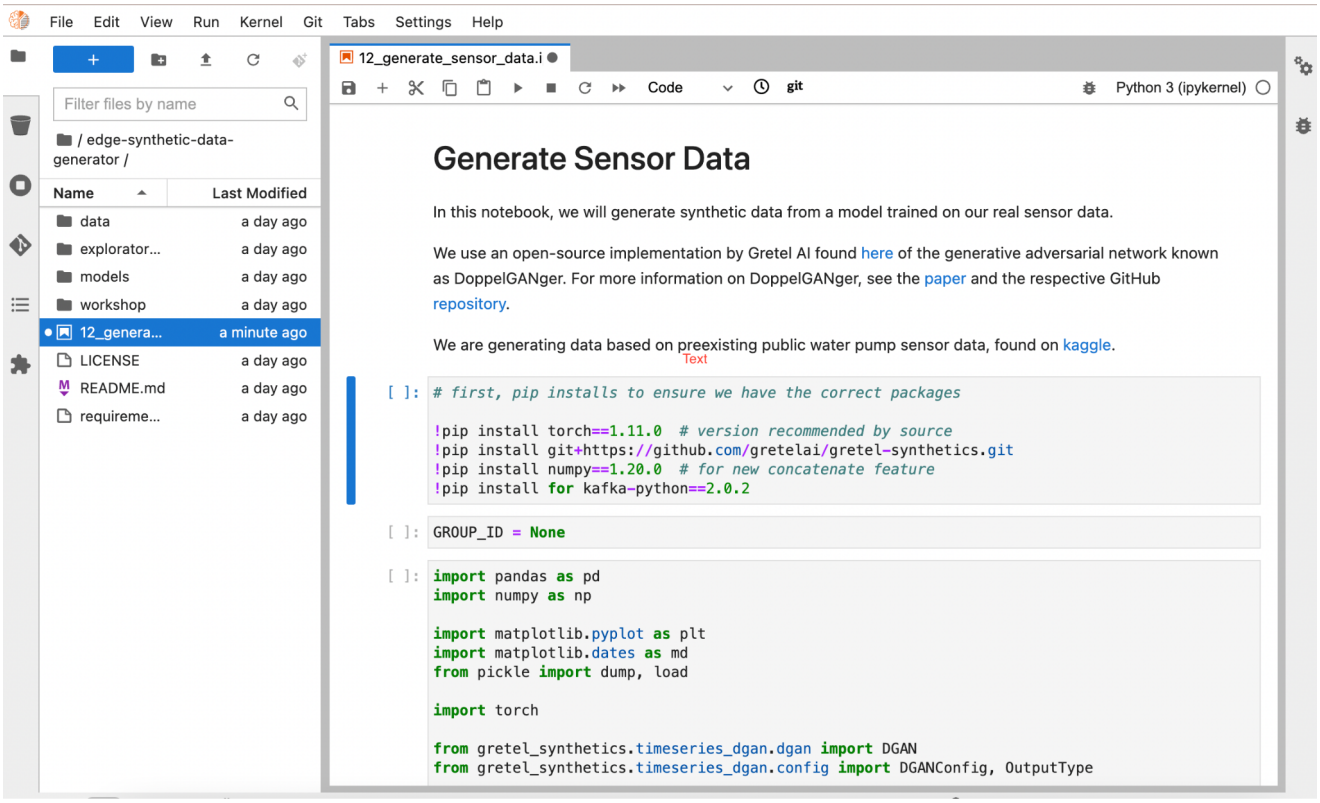
Let's open up the edge-synthetic-data-generator folder by double clicking on the edge-synthetic-data-generator folder name. **For the purposes of testing, we'll have one person from each group generate synthetic data and stream it. So if your name is in bold on the Workshop Signup spreadsheet you will be responsible for doing so.**



The folder should contain the following sub-folders:

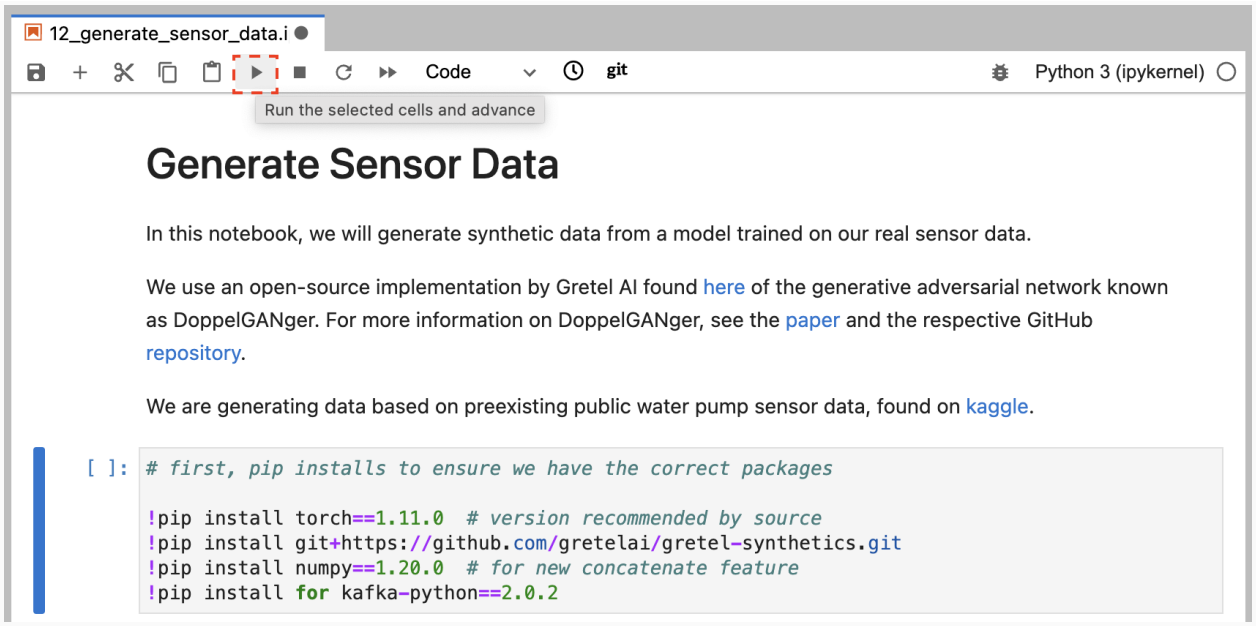
- `data`
- `exploratory-notebooks`
- `models`
- `workshop`

We will be working with the `12_generate_sensor_data.ipynb` jupyter notebook. Double click on the notebook name to open the notebook.



You will be running each of the cells, one at a time, in this notebook.

Run the first cell to install the required packages for this workshop by selecting it and clicking on the run button or press the <shift><enter> keys on your keyboard.

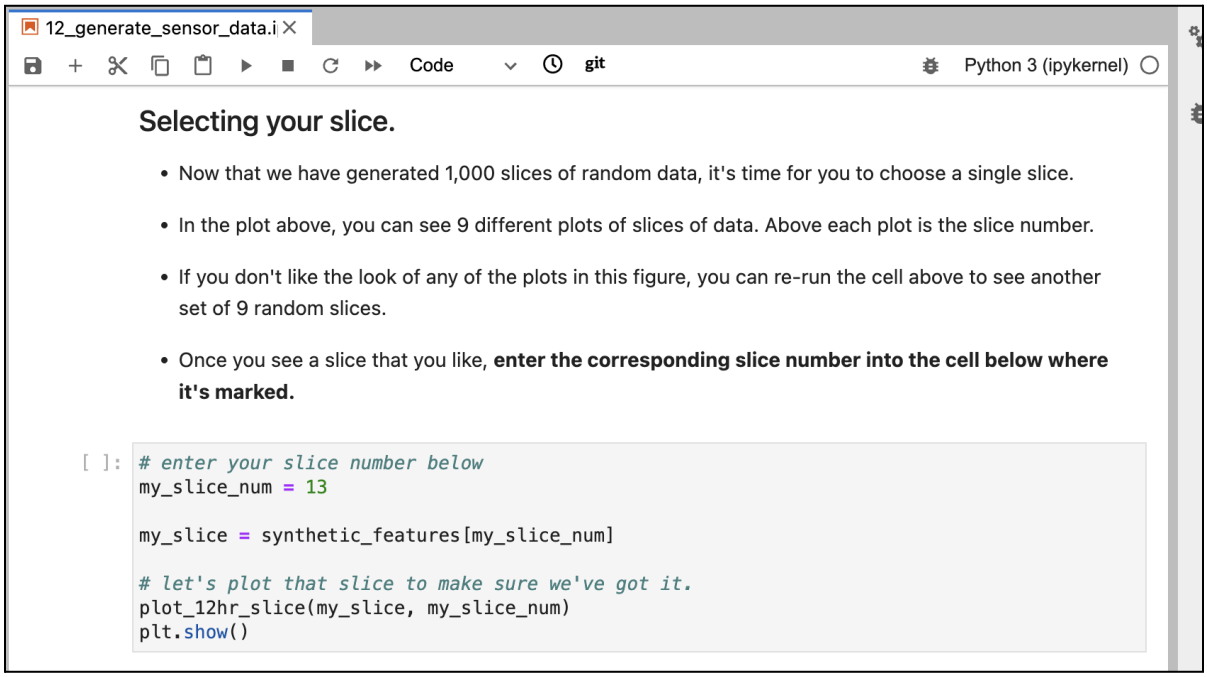


Next, in the following cell, replace ‘None’ with the GROUP_ID number provided to you and run the cell.

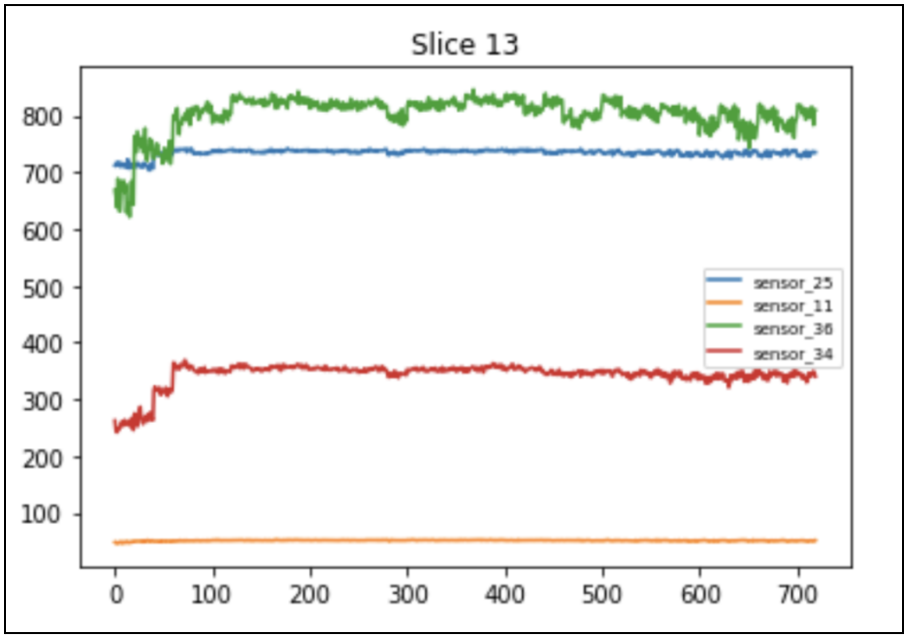
Afterwards, select and run cells 3-7 to generate synthetic data from a pre-trained model and plot some slices of the data.

Select a data slice

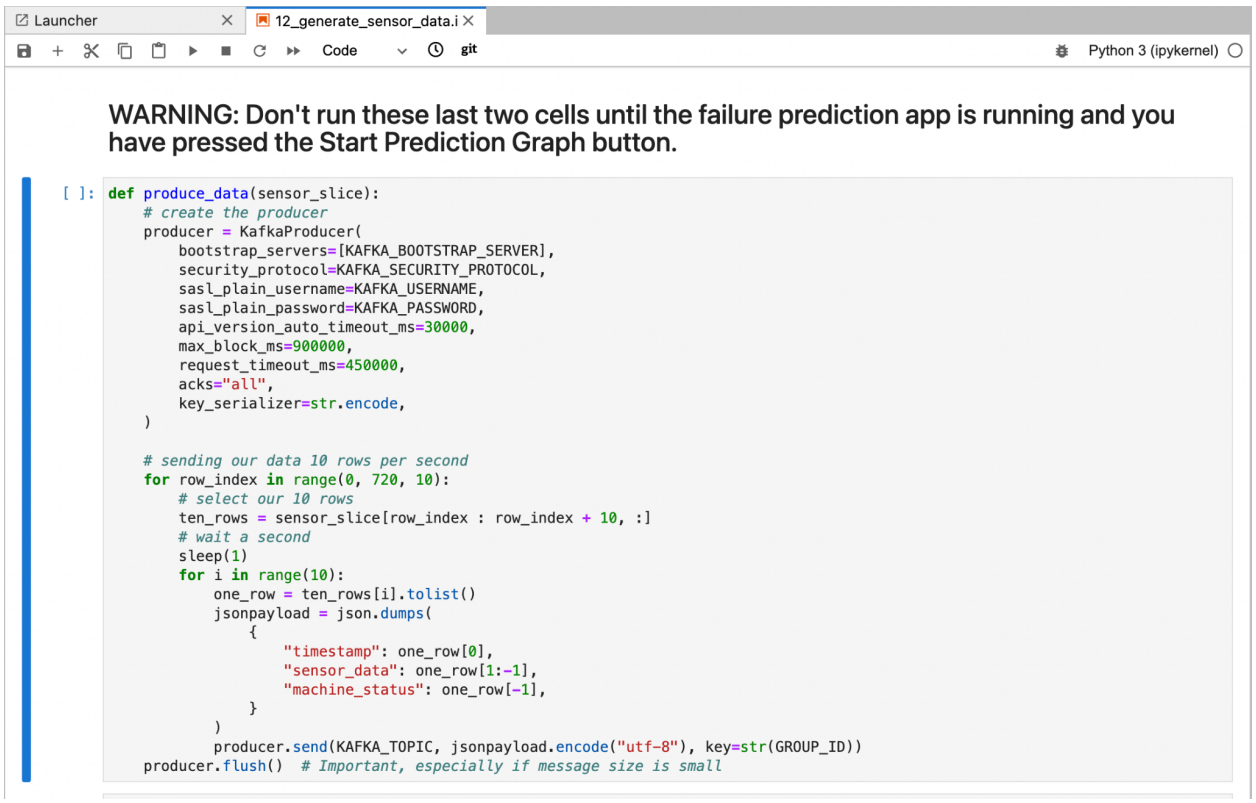
Once you have run the first 7 cells and reach the section titled ‘Selecting your slice’, enter in the slice number in the following cell. For example, if your slice number is 13, your code should resemble the following:



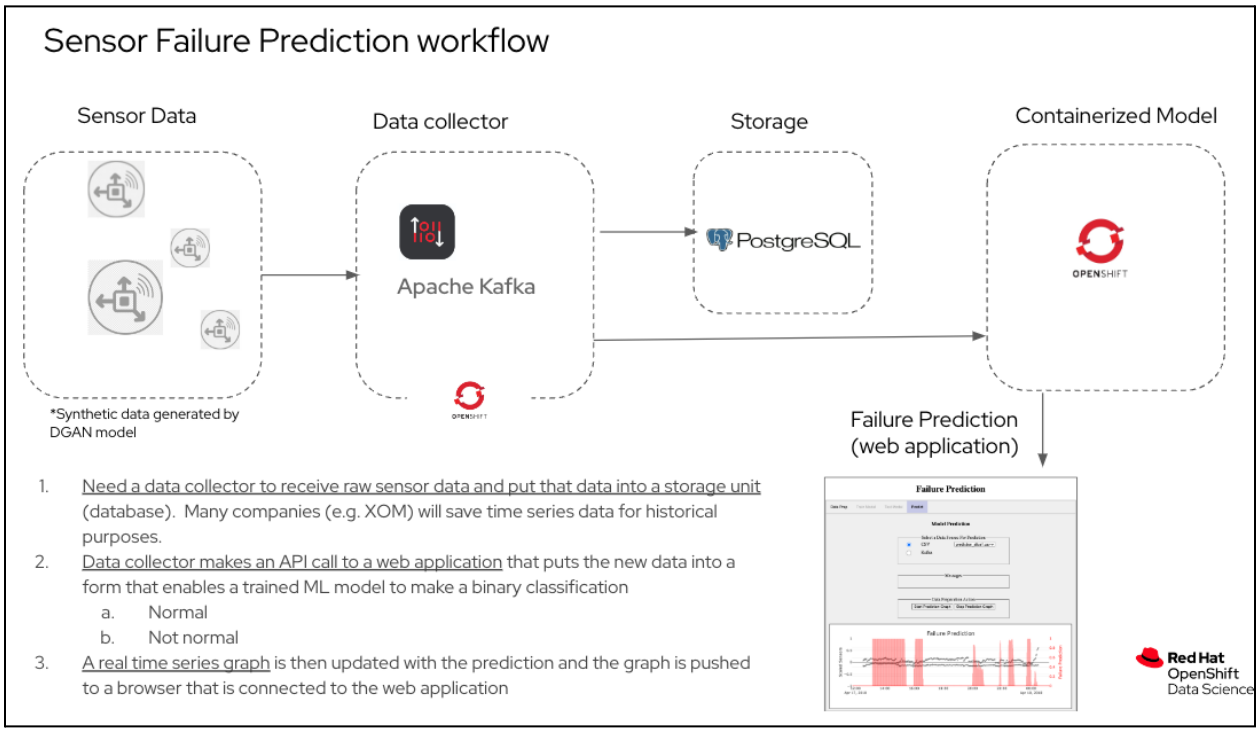
After inputting your slice number, run the cell to plot the sensor data for that slice:



Next we will prepare our sensor data, for streaming by Kafka, by running cells 9-11. Stop at the following warning message:



Before you run the remaining 2 cells, let's take a look at the Sensor Failure Prediction workflow so that we understand what is happening with our sensor data and in particular why we use Kafka.



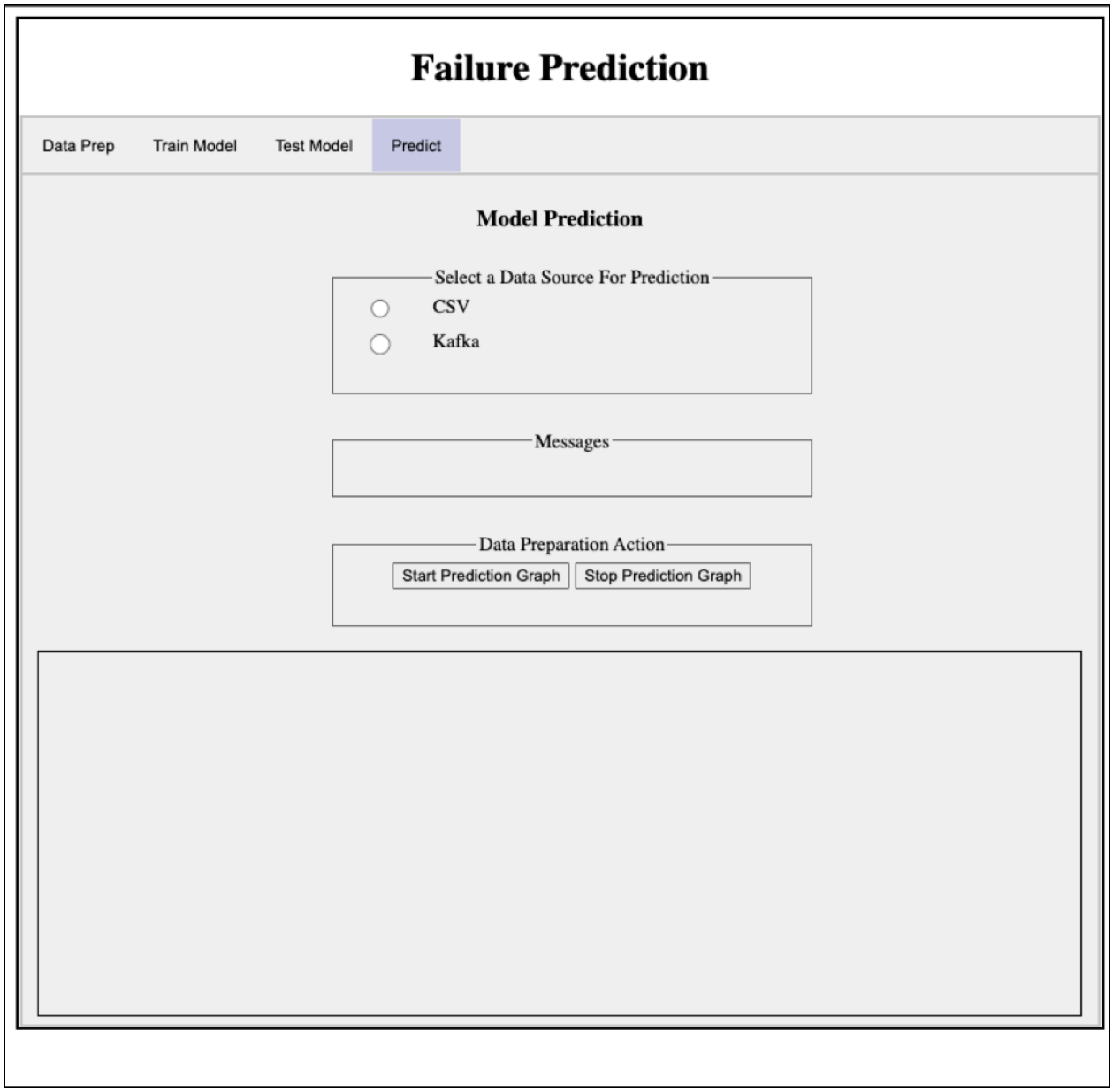
When we generate our synthetic data, it needs a service to push it to our model for failure prediction. The service that we use is called **Apache Kafka Streaming**. This service takes our generated synthetic data and pushes it to our deployed (and containerized) Failure Prediction web application.

We didn't have time in this workshop for you to set up this streaming service. Therefore we have set up the Kafka service for you.

Now that you understand what role the generated synthetic data plays, and what role Kafka plays, we will leave this notebook. Don't close it, as we will come back to the notebook to perform the actual data streaming. We will turn our attention back to the Failure Prediction Web Application that will be using our data.

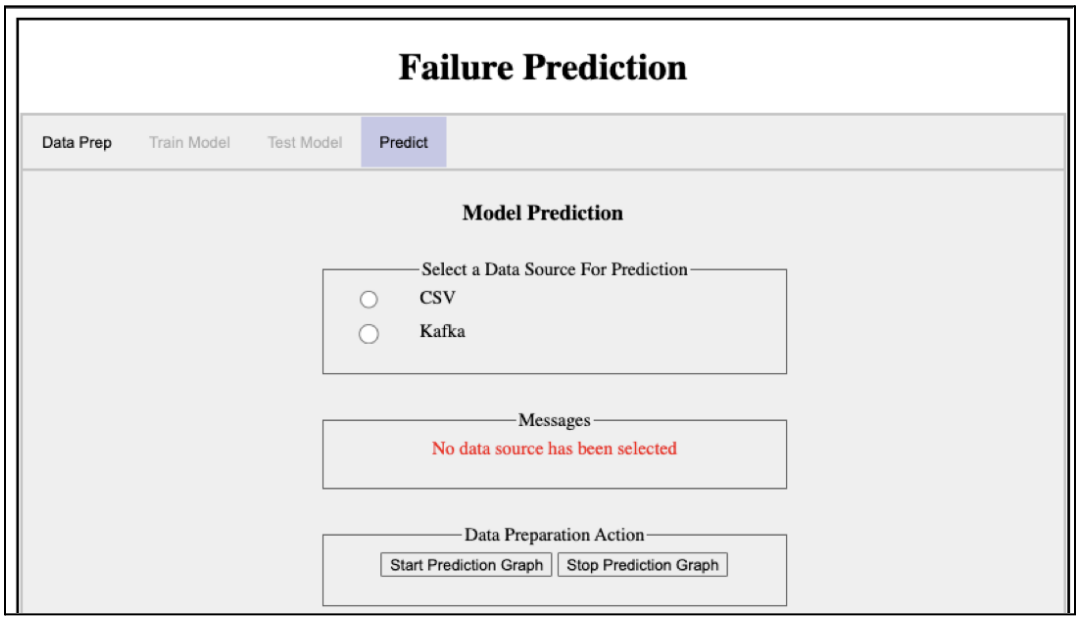
Prediction With the Model

Click on the Prediction tab:



Before we can make a prediction, we must first select a data source for the prediction data. There are two options:

1. Use a CSV file which will be streamed one point at a time, simulating real time generation. The data in the CSV file is taken from the original Kaggle data source as test data.
2. Use a data stream of *synthetic* data with the help of Apache Kafka that also simulates the production of real time data. NOTE: **<u>You cannot run a prediction if no data source has been selected.</u>** If you attempt to click on the Start Prediction Graph before selecting a data source you will get a red message:



If you choose the CSV radio button, a select box will list CSV file names available. Just **select a CSV filename**.

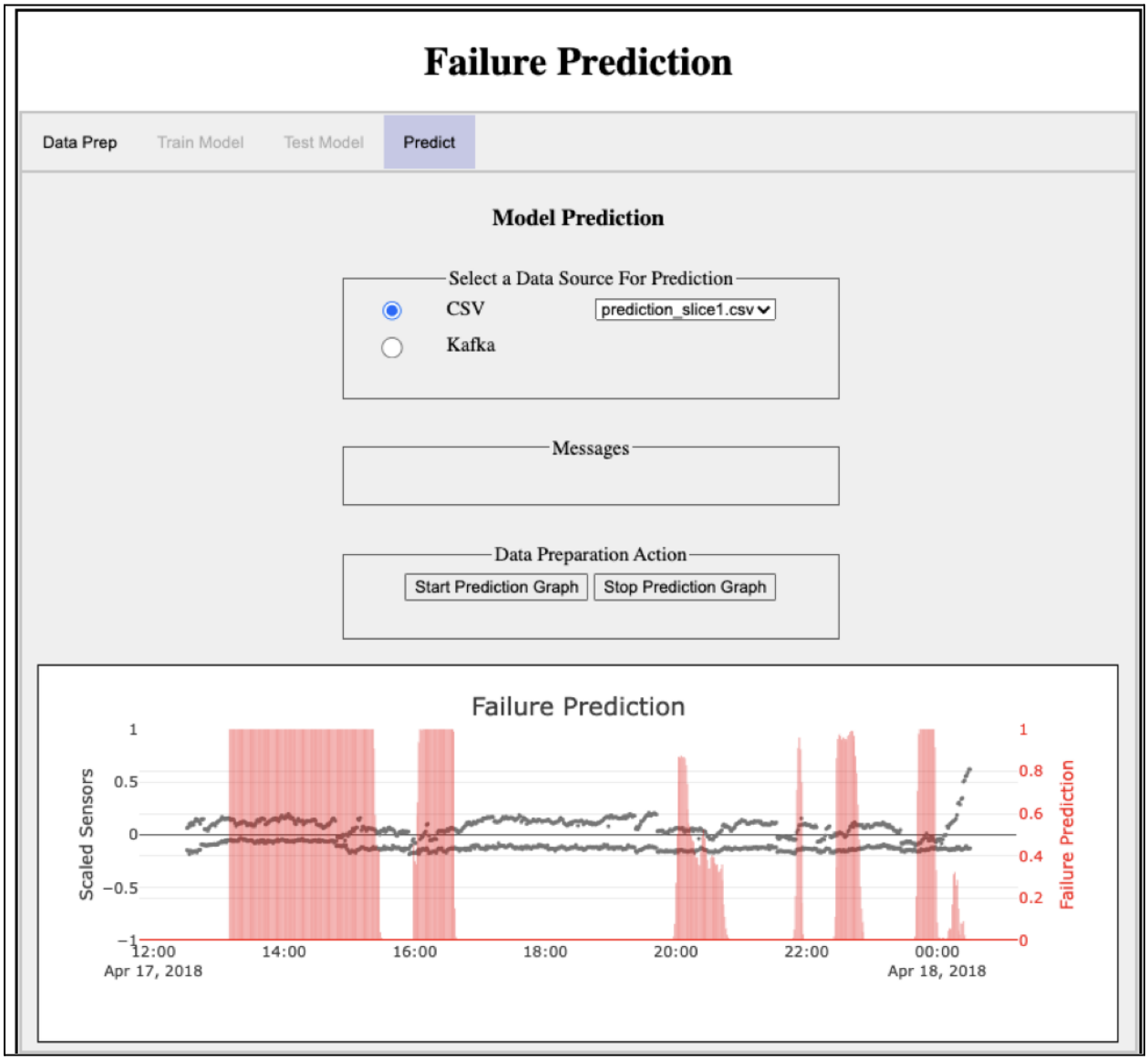
If you chose the Kafka radio button, **enter the Group Id** that you have been given. This Group Id allows your browser session to pull out messages that you were identified with.

The purpose of the button labeled Stop Prediction Graph is to allow you to stop a prediction process before it is finished. You can use this button if you wish to choose another data source and run the prediction again before the last prediction process has finished.

CSV Option

Click on the Start Prediction Graph button.

There will be a pause while the data stream is prepared. Then you will see points from the prediction data source as well as a red prediction alarm if the model predicts failure is imminent. Note that the prediction data is only available 12 hours before failure.

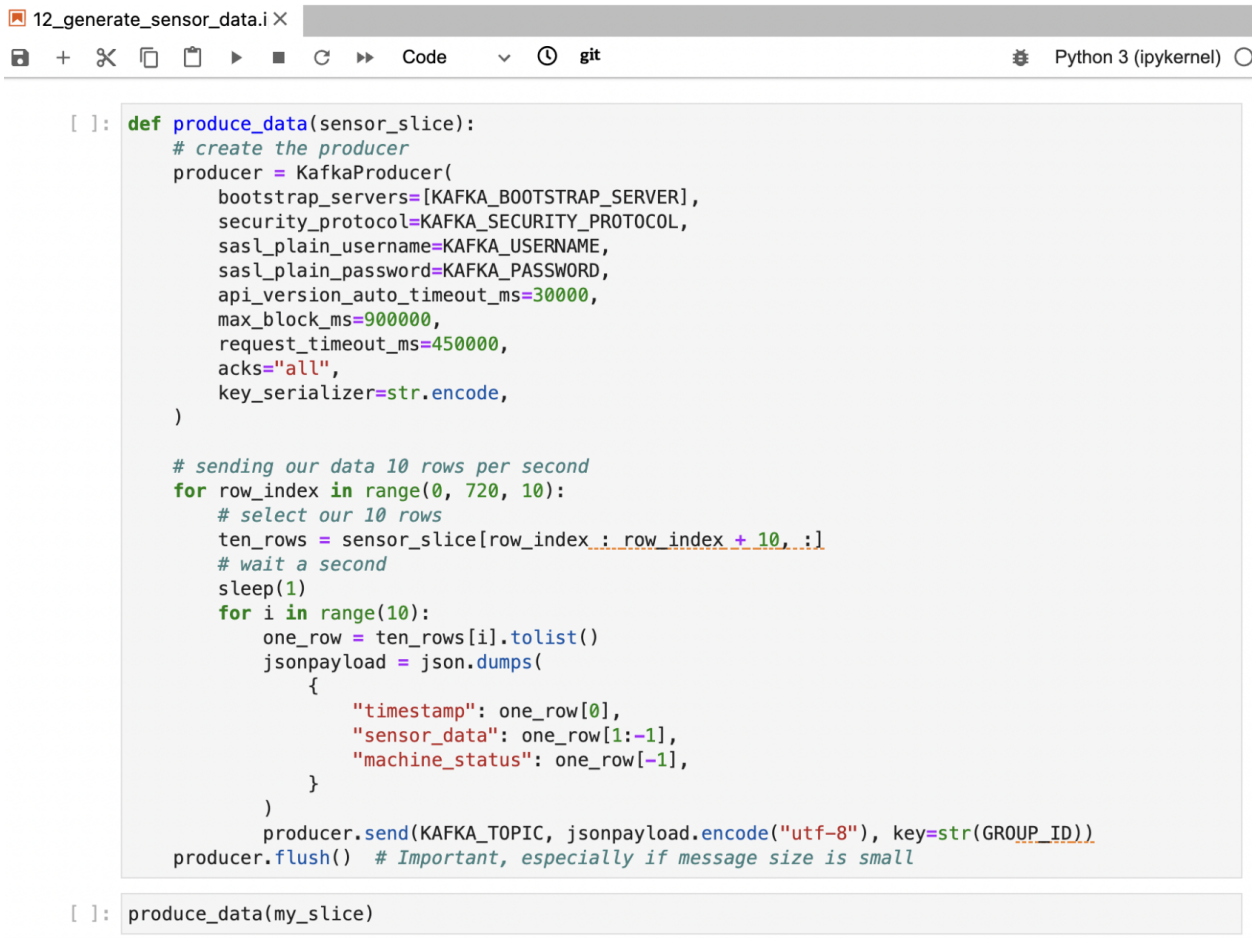


In this case actual failure was at 2018-04-18 00:30. The graph in your browser is interactive, so you can move your cursor to display the times.

Kafka Option

Click on the Start Prediction Graph button.

Head back to the Jupyter notebook where you generated synthetic data and run the remaining 2 cells which will (1) connect to the Kafka cluster based on the credentials you defined in the previous step, (2) initialize a KafkaProducer object, (3) stream your data to the sensor failure prediction model



Circle back to the web app to see your prediction graph updating as Kafka is streaming the synthetic data that you generated to your model.

This concludes our workshop.