**Objective:** To allow the user to train various models on a variety of time series data and compare those models for forecasting and computational performance.

The pipeline consists of 5 main containers (EDA, Preprocess, Train/NonML, Inference) and several helper services (MinIO, MLflow, Druid, Superset). There is still no dynamic input for data so the user must upload the data to MinIO and point to it manually. The project would greatly benefit from an input dashboard that would allow the user to easily upload datasets and apply the preprocessing/training jobs to the cluster. The exploratory data analysis (EDA) container runs a statistical profile on the data and creates various plots (PCA, PACF, Correlation pair plot, etc.) to assist the user in visualizing and identifying important qualities of their data. This will allow the user to make the necessary choices in the preprocessing and training steps to better suit their situation.

There are essentially two options for performing long forecasts with deep learning models, configuring the model to have a long output sequence length, or predicting recursively, wherein the model’s predictions are fed back into the model to generate the next predictions. The former option tends to dramatically reduce accuracy as the model must predict N orders of change for N output sequence length. As N increases, the model begins to flatten out and only predicts a unimodal or bimodal average. Recursive inference has the advantage in accuracy for very short forecasts but the disadvantage of compounding error. You can expect to see the model be very accurate for a few data points but then rapidly converge to a large nonsensical value. Classical statistical models have an advantage for long forecasts because they don’t break, however, they are usually much less accurate overall. If short term accuracy is paramount, consider deep learning models but if long term estimates are more important, use non deep learning models.

The dataset that was used during development came pre-split. This has the benefit of not importing extra data unnecessarily, however, any future data input mechanism must also split upon intake. This is also not to be confused with a further split that happens when training the deep learning models.

Make sure to change the pytorch versions in requirements.txt for inference and train containers to the version that includes gpu support and rebuild the images.

StatsForecast Models (and probably Prophet as well) can only predict from the last datapoint they were trained on). For those not well versed in some of the statistical models, ARIMA is incredibly slow and a complete memory hog. Don’t try to fit with long season lengths unless you have a lot of memory. ETS is sensitive to long season lengths as well as needing no negative values (use MinMax Scaler). Generally reducing season lengths should be the first debugging step if a model breaks, followed by down-sampling if necessary.

Both the deep learning models and non-deep learning models could benefit from being tuned. Additionally, the deep learning models are rather simple and could be improved.

.kubernetes directory is deprecated, all up to date Kubernetes can be found as templates in the helm chart. Commands to use helm version:

minikube start --cpus \_ --memory \_ (tested with 14, 22GiB)  
*build or import docker images*  
helm install flts ./.helm

Adding resource requests and limits can help minikube manage resources on limited systems.

Note: the program only overwrites to Druid, it does not delete and recreate, meaning data might be duplicated if periodicity is changed, a different forecasting window is used, etc.

Druid SQLalchemy URI: druid://broker:8082/druid/v2/sql/

Superset SQL Lab template to create virtual dataset:  
WITH j AS (

SELECT

COALESCE(a.\_\_time, b.\_\_time) AS \_t,

a.down AS a\_down,

a.up AS a\_up,

a.rnti\_count AS a\_rnti\_count,

a.mcs\_down AS a\_mcs\_down, a.mcs\_down\_var AS a\_mcs\_down\_var,

a.mcs\_up AS a\_mcs\_up, a.mcs\_up\_var AS a\_mcs\_up\_var,

a.rb\_down AS a\_rb\_down, a.rb\_down\_var AS a\_rb\_down\_var,

a.rb\_up AS a\_rb\_up, a.rb\_up\_var AS a\_rb\_up\_var,

b.down AS b\_down,

b.up AS b\_up,

b.rnti\_count AS b\_rnti\_count,

b.mcs\_down AS b\_mcs\_down, b.mcs\_down\_var AS b\_mcs\_down\_var,

b.mcs\_up AS b\_mcs\_up, b.mcs\_up\_var AS b\_mcs\_up\_var,

b.rb\_down AS b\_rb\_down, b.rb\_down\_var AS b\_rb\_down\_var,

b.rb\_up AS b\_rb\_up, b.rb\_up\_var AS b\_rb\_up\_var

FROM "PobleSec\_test" a

FULL OUTER JOIN "AUTOARIMA" b ON a.\_\_time = b.\_\_time

)

SELECT

CAST(\_t AS TIMESTAMP) AS "\_\_time",

a\_down, a\_up, a\_rnti\_count,

a\_mcs\_down, a\_mcs\_down\_var, a\_mcs\_up, a\_mcs\_up\_var,

a\_rb\_down, a\_rb\_down\_var, a\_rb\_up, a\_rb\_up\_var,

b\_down, b\_up, b\_rnti\_count,

b\_mcs\_down, b\_mcs\_down\_var, b\_mcs\_up, b\_mcs\_up\_var,

b\_rb\_down, b\_rb\_down\_var, b\_rb\_up, b\_rb\_up\_var

FROM j;