This submission in particular focuses on fixing a model and then changing hyperparameters to look at how accuracy - and results in general - change according to the various set of hyperparameters
---

How to use:

This file contains two Jupyter notebooks - one with a data file creating code (data_gen) and one with the actual definition of the model (Model). The code also generates plots of multiple features and saves them for future reference. THe rest of the content of the notebooks are defined and designed to be self-explanatory.

First run the notebook 'data_gen' and then 'Model'.

While the other implementation studied the performance amongst models, here I fixed the model - a Multi-layer perceptron (MLP) with 3 linear layers, ReLU activation and SGD as optimizer for training.

The purpose in this method was to explore how - given sufficient data - the MLU model can accurately perform, given different slices of the data, one by one.
I mainly aimed to reduce the MSE loss of the training set and then tried predicting the values for the testing data, I tried different hyperparameters for different models and stocks (again, while not changing the model!) like learning rate, number of epochs, batch size etc. The aim was to minimize MSE loss. To understand the models performances, I plotted the predictions and ground truths of the test data as well.

I also tried understanding dependence between the features and for that I tried taking different number (as well as different sets/orders) of features to predict the close value, however from the experiment I observed that the given list of features in the question gave the best results.

The results achieved are not perfect but they are very close to actual ground truth and upon making the network more dense and having more data to train we might achieve much better results.