Intelligent Systems(COS 30018)

Vivek Saini(10382806)

Task B.1-Setup

Tutor: Mukesh Malani

In the First task that was given to us, we are presented with a model of stock prediction python code, which is used to predict the stock prices based on actual information from yahoo finance. Furthermore, we are required to install all the dependencies and libraries to run the code.

Youtube tutorial:

In the tutorial given to us, the person very well explains how the code is supposed to work, how to install the dependencies and what are the different blocks of the code are supposed to do. He also very well explains how the system is scaled and also the python program is required to make a graph and predict the stock prices based on the information in a certain time frame using the functions like:

# start = '2012-01-01', end='2017-01-01'

TRAIN\_START = '2015-01-01'

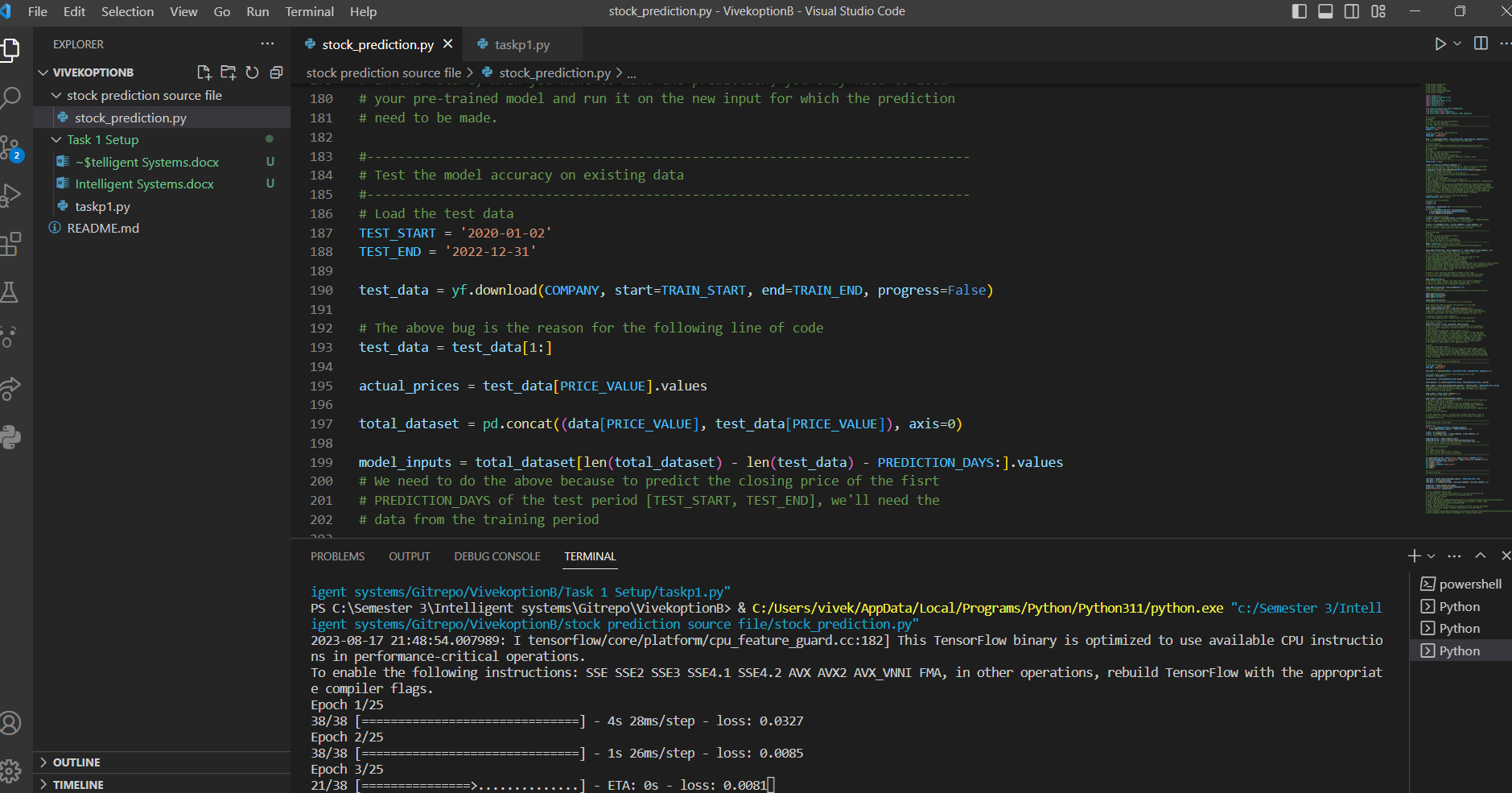
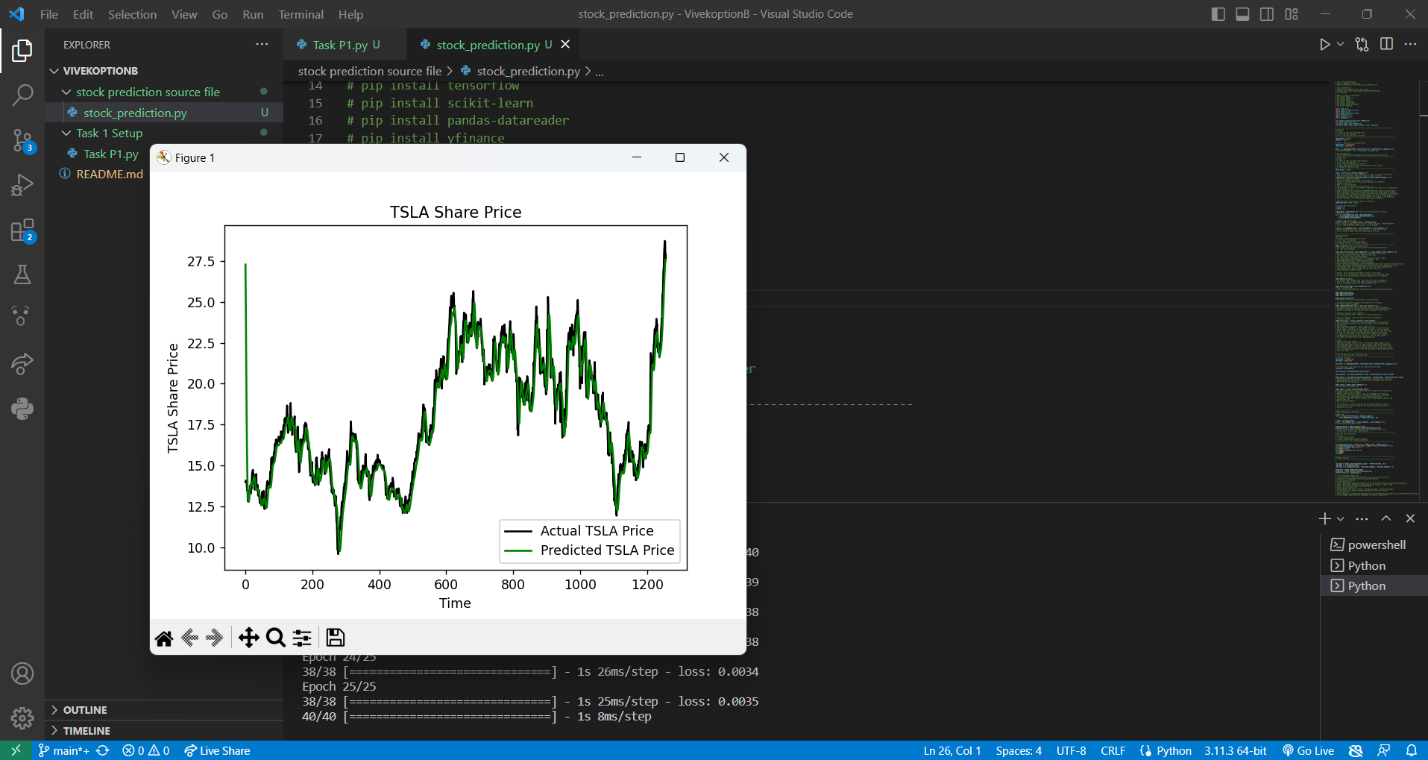
TRAIN\_END = '2020-01-01'

data =  yf.download(COMPANY, start=TRAIN\_START, end=TRAIN\_END, progress=False)

# yf.download(COMPANY, start = TRAIN\_START, end=TRAIN\_END)

Observations:

The observations that I made are as follows:

* There are different libraries that are required to be installed to collect the data and train the neural network model to produce the results as required.
* While testing the code, I encountered a problem where my keras model would not work, but I think there was some issue with the global version , I had to rework add few more lines to make it work , Here are the corrected libraries.
* import tensorflow as tf
* from tensorflow import keras
* from keras.models import Sequential
* from keras.layers import LSTM, Dense, Dropout, Bidirectional
* from sklearn import preprocessing
* from sklearn.model\_selection import train\_test\_split
* from yahoo\_fin import stock\_info as si
* from collections import deque
* Other than this I also observed that the model trains itself as explained in the tutorial and finds the most optimal route of predicting the stocks.
* 
* The predicted prices were really similar to the actual prices.

Task P1:

This task was a little challenging, I got a lot of errors while trying to to set up the environment but in the end, it was fun to play with , It was similar to the first task, we installed the libraries, loaded them up, and wrote down the code which preprocesses the information.

After this we move on to the training of the model. So, we're setting up a neural network to predict future prices, and here's how we're going about it. First, we've got this thing called TEST\_SIZE, which is basically the chunk of data we'll use for testing, and in this case, it's 20% of the whole dataset. Now, for the actual prediction part, we're picking out certain features from the data, and we call these FEATURE\_COLUMNS.

Next up, we're talking about the actual structure of the neural network. We've got these layers, and you can choose how many you want with N\_LAYERS. Inside these layers, we're using these cells, and by default, they're LSTM cells, which are good for sequences of data like this. You can decide how many cell units you want in each layer with UNITS.

But hey, we don't want the network to get too good at memorizing stuff and not really understanding, so we're adding this thing called DROPOUT. It's like telling some of the nodes in the layers to chill out during training – a dropout rate of 0.0 means no chill, while higher values mean more chill.

Now, BIDIRECTIONAL is a fancy term. It's about whether the network should consider the context from both past and future data points. So if it's set to true, it's like the network is reading the data in both directions.

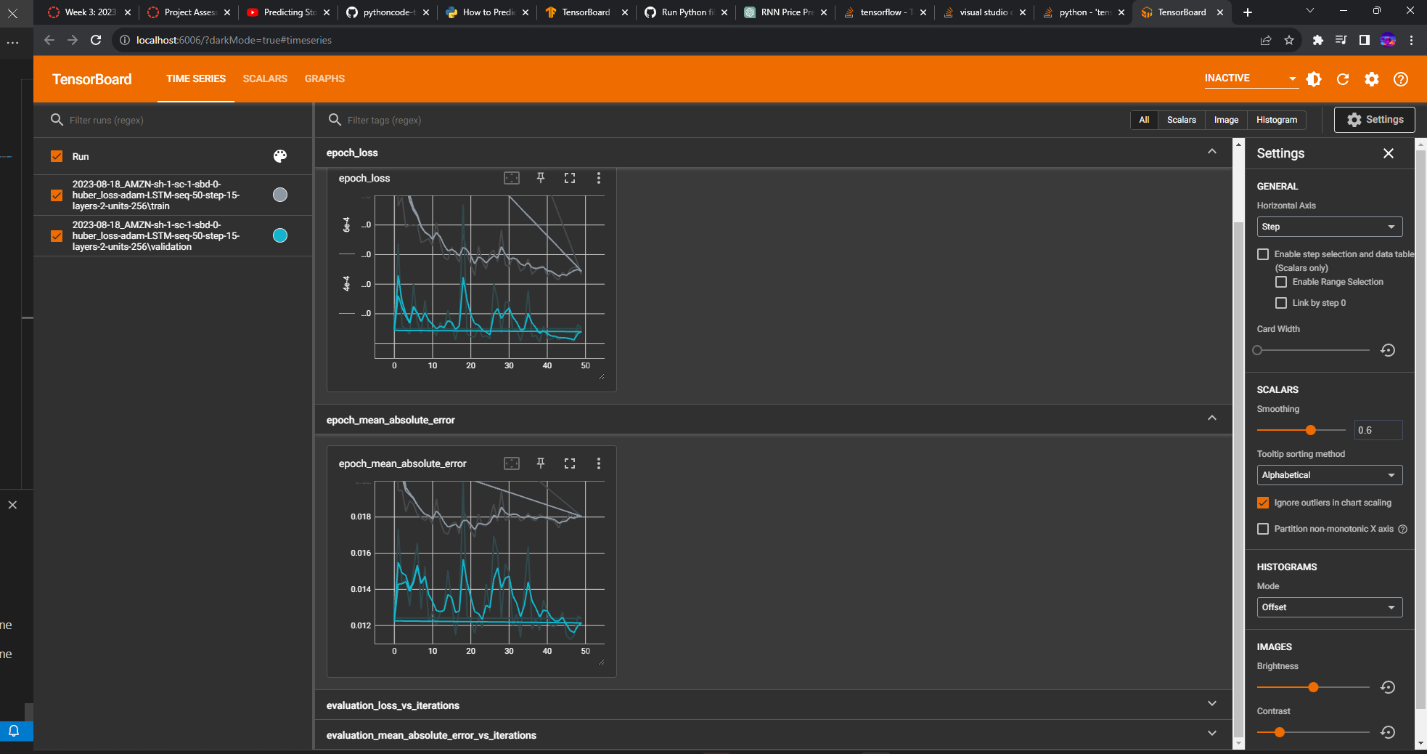
Okay, let's talk about the nitty-gritty of training. We've got this thing called LOSS, and we're going with Huber loss. It's a way to measure how wrong our predictions are, but it's not overly sensitive to those occasional big errors that can mess things up. Other options are mean absolute error (mae) and mean squared error (mse), which you might have heard of.

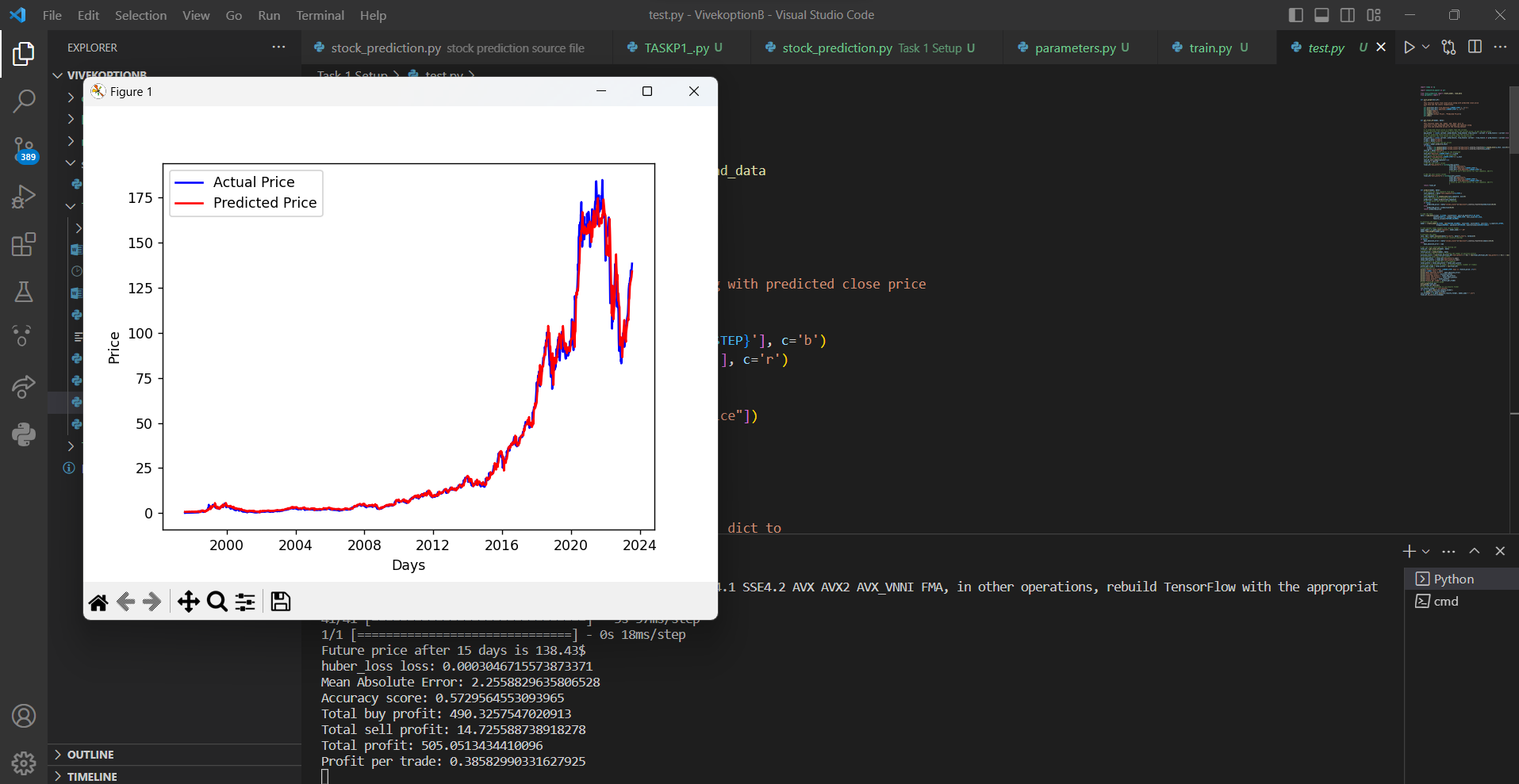
Then comes the OPTIMIZER, which is how the network tweaks itself to get better at predictions. We're defaulting to Adam here, which is a popular choice. During training, we don't want to feed the entire dataset at once – that's where BATCH\_SIZE comes in. It's like serving up chunks of data for the network to learn from.

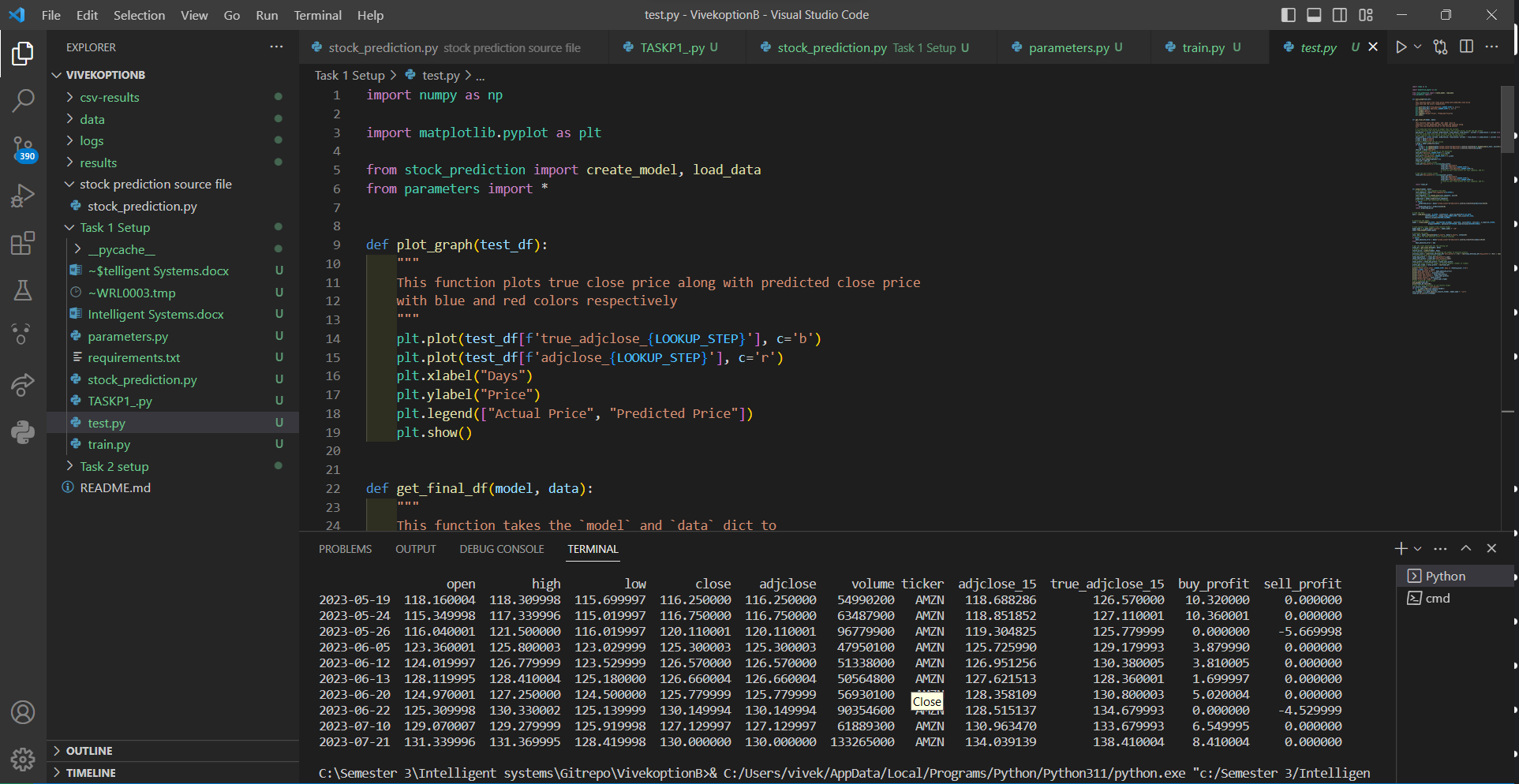
Lastly, we've got EPOCHS – it's the number of times the network goes through the entire training dataset. We're starting with 500 rounds, but you could try more to really let the network learn the patterns. So, in a nutshell, that's the setup we're going with to make our neural network predict those prices.

We also added the folders to make sure that results, logs , and data folders exist.

Then we moved on to check the progress by using the tensorboard command to check the mean squared or absolute error.



At last I would like to conclude this task with the final output after training the model.



Task P2:

I tried running the code in a virtual environment but was unsuccessful in running it, although I found the youtube tutorial for that code and had a good understanding of how it works, to summarize it is similar to the code I have done before. In this scenario we have different libraries to install but similar practice to load the data, train the data , and then testing out the data. In the end, we make a candlestick chart instead of a line graph which we did with matplotlib.