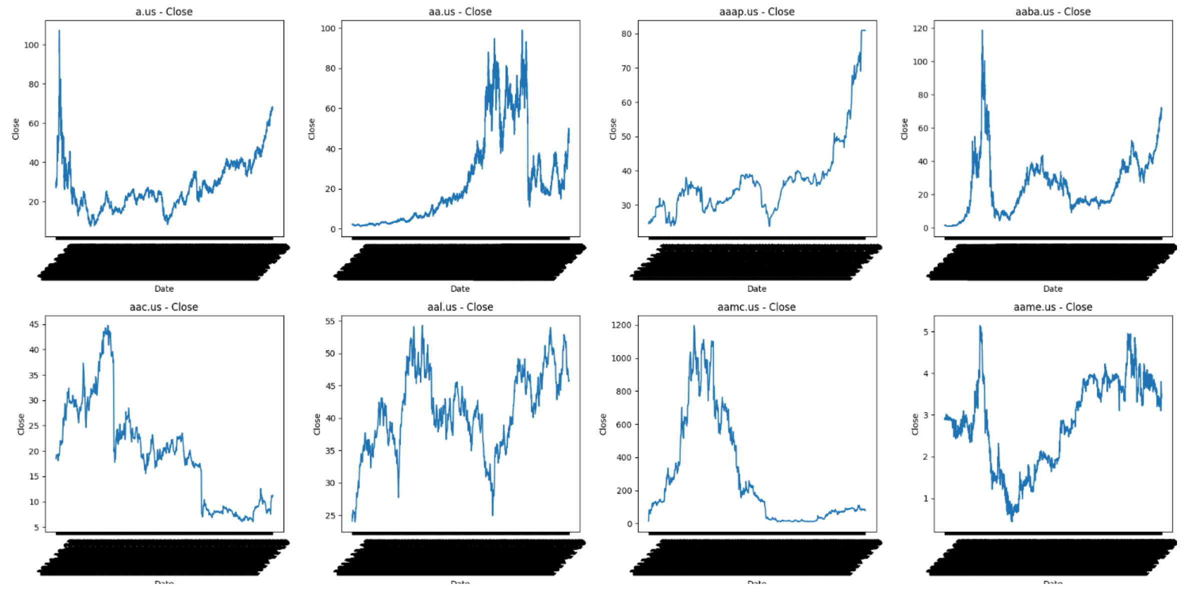
Introduction

This project works on stock prediction on randomly selected companies in the US. The model can be adjusted to different number of training days & prediction days. However, for my analysis, I decided to predict the stock price for 7 days following a 2 month span. I use a tensorflow keras LSTM model to accomplish this.

Analysis

The dataset I worked on was the “Huge Stock Market Dataset”. This dataset has over 8000 companies’ stock market data from 1999, or when the company was made, to 2017. Each company’s data is stored in a csv file in separate folders within a ‘Stocks’ folder. The dataset contains ETFs as well, but I will focus on Stocks. The csv’s contain the following features: Date, Opening, Closing, High, and Low amounts (in USD), volume, and open interest. I converted each csv into a data frame, and added a ticker column based on the name of the csv which was the company ticker. I made a function to convert the network of csv files into a dataframe. It selects a given amount of randomly selected companies, 50 in the case of training, and combines them into a single dataframe. It skips any empty csv files, and any companies with less than some minimum amount of given days, 60 days in the case of training. Here’s an example of 20 companies’ date vs closing prices:



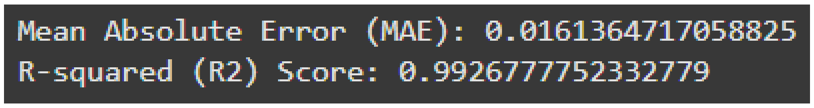
The next preprocessing step involves looping through each ticker and doing the following. Using a scaler to transform the data into a scaled version from 0 to 1. Looping from sequence length which was 60 days in this case to the length of the data minus the predicted number of days which was the 7 days following the 60 days. It makes sequences of 60 days that are appended to x, and the following 7 days of that sequence are appended to y. It converts x and y to numpy arrays, and appends the company’s data to a running numpy array of all the companies’ data. For training, I split the data into a 80% training 20% test split. Finally, the data is ready to be trained and inputted into a model.

Methods

The model I used for this project was a keras Sequential Model with a LSTM. I first created an empty model, then added a LSTM layer with 50 nodes with input shape of 60 days by 6 features. Then I added a 20% Dropout layer which drops 20% of the most inactive nodes. I added another LSTM with 50 nodes and another 20% dropout layer. Next, a 25 node fully connected dense layer, then outputted the results with a 7 node (for the next 7 predicted days) fully connected dense layer. I compiled the model with Adam optimizer with 0.001 learning rate, and mean squared error loss function. I fit the model on the training data over 20 epochs with 128 batch size using the test data as the validation set.

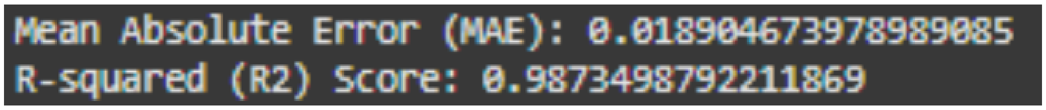
Results

Predictions were made of the x test set. The following Mean Absolute Error and R-squared score between the predicted and actual results was calculated



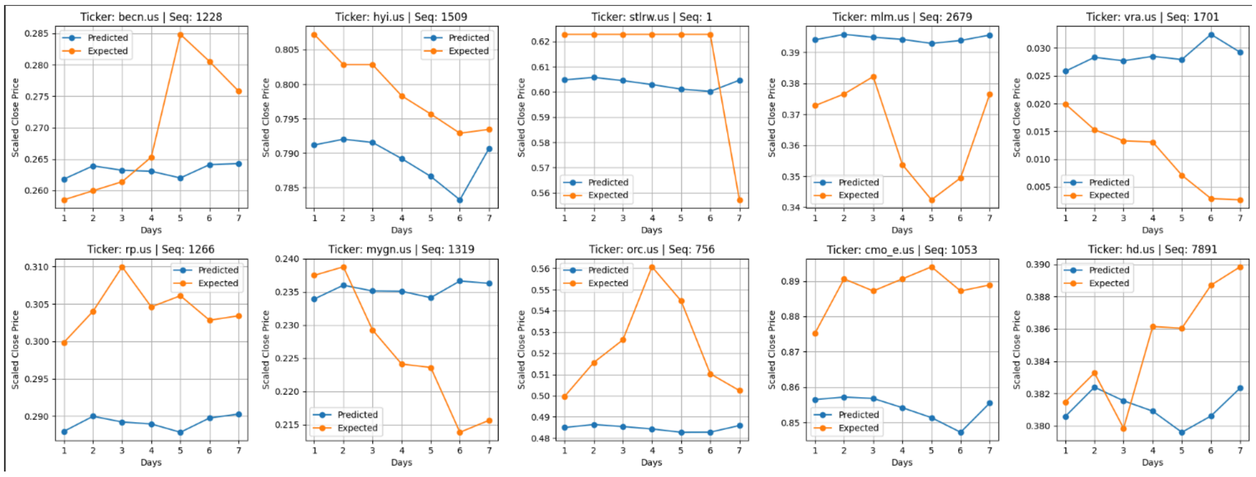
```
Mean Absolute Error (MAE): 0.0161364717058825
R-squared (R2) Score: 0.9926777752332779
```

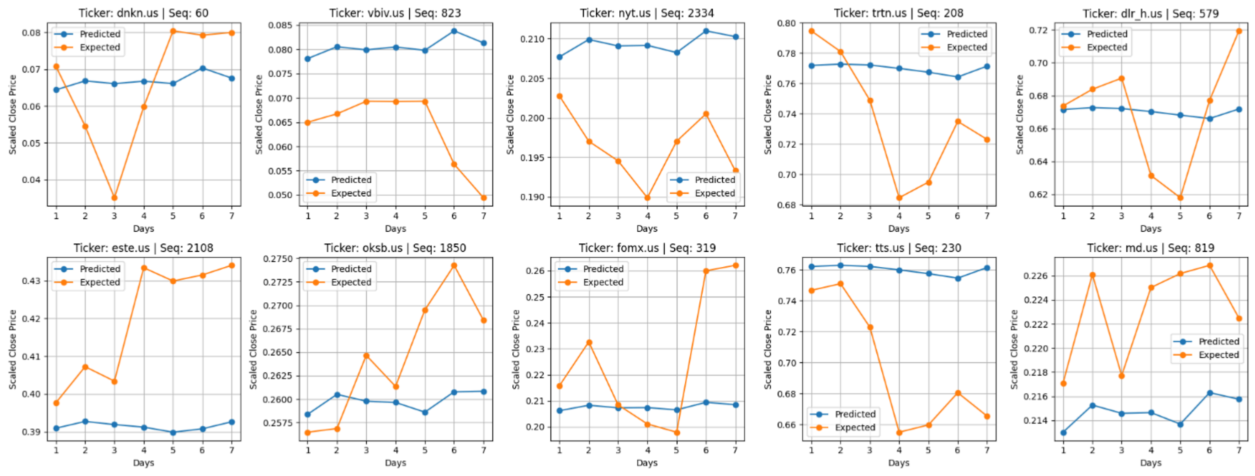
These values indicate a pretty well built model with predicted values that are very close to the actual. When testing the model on 20 new randomly selected companies using the same preprocessing step, the following results were calculated:



```
Mean Absolute Error (MAE): 0.018904673978989085
R-squared (R2) Score: 0.9873498792211869
```

Next, I converted the predictions into a data frame with the ticker column, sequence number, and 7 columns for each predicted day following the 60 days. Next, for each company in the 20 new companies, I made a graph of the predicted values vs actual values for the 7 days of a randomly selected sequence number. The following graphs were made:





This function could be rerun as many times as needed to generate graphs of randomly selected sequences. After analyzing many graphs, I've discovered that the lines appear to be quite different, but mostly because of the scale. The scaled values are kind-of relatively close, and follow a similar pattern, just not as volatile. It mostly stays around the same values and doesn't vary much. Something that might cause the spikes/dips in real time data that may not be predictable are black swan events in the company/industry/world, or noise trading.

Reflection:

One of the biggest things I learned through this project is the importance of preprocessing data. The downloaded data was in many folders of csv files, so it had to be preprocessed in a way that could take advantage of the data. Additionally, I was to figure out how to format the data in a way that would train on 60 days and predict the following 7 days. At first I did 1 day, then swapped to 7 days. Additionally, with such a large amount of data, I figured I had to train it in a way that could take advantage of the size of the dataset without it taking way too long, so I randomly selected companies. Luckily the model was pretty good after training three times after adjusting values, considering it took around 30 minutes per time. In the future, I hope to be able to predict more days following the 60 days, or include more than 60 initial days. Being able to predict long term stock prices will also be important as it won't be as volatile as 7 days. As i'm writing this right now, I realize that a stock market week is only 5 days so I should've done 5 or 10 predicted days, but I don't think it matters that much.