Justin Jiang

Appendix

Introduction

Stocks is a time series where its price fluctuates over time. Stocks is the price to buy one share of a company. Investors buy stocks so that when its price does increase, it can be sold to make profit. There are factors that influence stock prices: company performance, economic status, and market sentiment. Market sentiment is defined as the accumulated feelings of investors towards a company or its stock.

The end goal of this project is to create models able to predict and forecast with the use of market sentiment as a factor. Obtaining a concrete measure of market sentiment is needed before modeling. After obtaining the market sentiment, the features used for the models will be the open price, the highest price, the lowest price, the volume of shares, and market sentiment will be used to predict the close price (target variable). Models that will be used support vector regression and a recurrent neural network. The model is selected based on the measure of RSME and ability to generalize. RSME was used to easily tell the discrepancy between the actual and predicted price.After selecting a model, it will be used to make a 5 day (stocks are open for 5 days in a week) forecast of the stock.

Dataset

There are two datasets used in this project. Both dataset contains the dates of between September 30, 2021 to September 30, 2022 and 25 different companies. The stock dataset contain the prices of the stock for each day and company. Description of relevant feature are:

- Open - price of stock beginning of the day
- Close - price of stock end of the day
- High - highest price of the stock during the day
- Low - lowest price of the stock during the day
- Volume - shares traded during the day

The tweets dataset contain tweets about a company during the day. Each tweet will represent how each investor felt during the day. The distribution of tweets for each company is shown in **Fig 1.0**. The number of tweets are fairly distributed for each company. The companies that are going to be experimented on are both Amazon (AMZN) and Microsoft (MSFT) because of the number of tweets being equal. Plotting

the close price during the year for both AMZN and MSFT are shown in **Fig 1.1** and **Fig 1.2**.

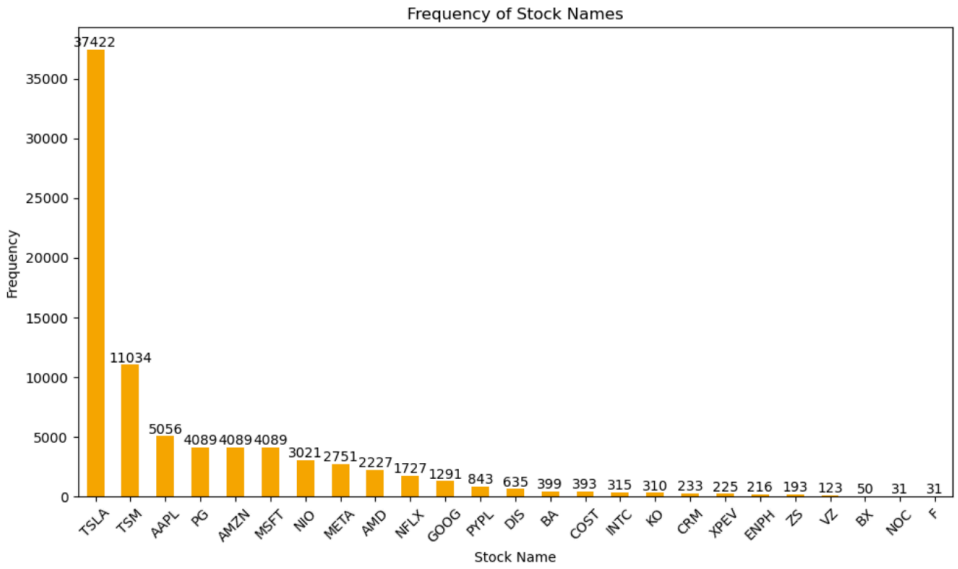

Fig 1.1

Fig 1.2

Fig 1.3

Due to the complexity of the price changing, predicting will prove to be a challenge.

Preprocessing and feature engineering

Each tweet needs to be preprocessed to properly evaluate a sentiment score of each tweet. The nltk library is used to pre-process tweets except for the operation of converting tweets. For nltk, the tweets are tokenized, have their special characters (@ and #) removed along with preceding words, punctuation removed, links removed (https and preceding characters), and lemmatized. The emoji library is used to convert emojis into words. After preprocessing the text, the tweets are evaluated for a sentiment score to see if a tweet is pessimistic or optimistic. The textblob library is used to find the sentiment of the preprocessed tweets. The scores range between -1 and 1, where -1 represents strong negative feelings, 1 represents strong positive feelings and 0 represents a neutral feeling. In terms of the stock market, 1 would mean a bullish attitude, and -1 would mean a bearish attitude. For the project the scores will be labeled accordingly shown below:

- [-1,-0.7] is strongly bearish
- (-0.7, -0.05) is bearish
- [-0.05, 0.05] is neutral
- (0.05, 0.7) is bearish
- [0.7, 1] is strongly bearish

The labels will be especially important for calculating the market sentiment. The distribution of the labels of the tweets are shown in **Fig 2.1**.

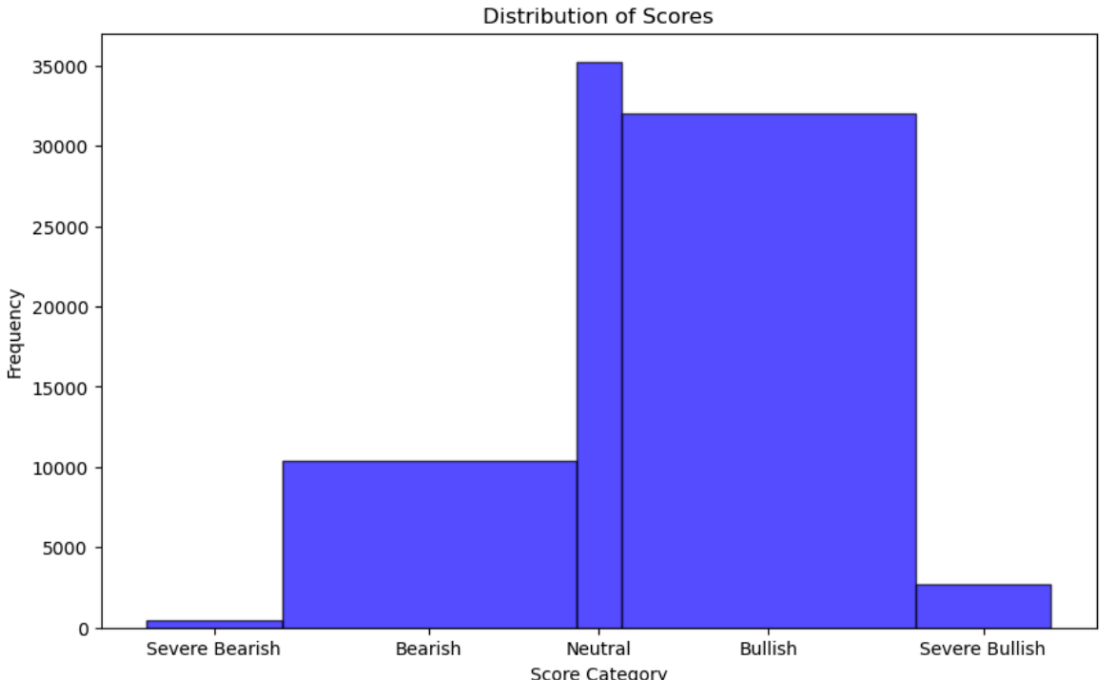

Fig 2.1

Before finding market sentiment, the local times need to be erased for each tweet since the tweets need to be in groups based on date and company. For finding the market sentiment, a weighted average of the tweets' sentiment for a day and company is used to find the weighted average. The weighted average is calculated by determining the majority predefined class from the portion of tweets. Each tweet is given a weight depending on the majority class. Tweets belonging to the majority class are weighted more, but tweets not belonging to the majority class don't have weight. Weighted average is used to help deal with the issue of sarcasm. Also, it represents how most investors feel towards a stock since a majority sentiment has more impact. After obtaining the market sentiment, it is added to the stock dataset as an additional feature. Market sentiment aligns with the company and day. The label of the market sentiment is added as well. **Fig 2.2** shows the result of the performed feature engineering.

	Date	Open	High	Low	Close	Adj Close	Volume	Stock Name	Average_sentiment	Sentiment_Label
0	2021-09-30	260.333344	263.043335	258.333344	258.493347	258.493347	53868000	TSLA	0.089583	Bullish
1	2021-10-01	259.466675	260.260010	254.529999	258.406677	258.406677	51094200	TSLA	0.139394	Bullish
2	2021-10-04	265.500000	268.989990	258.706665	260.510010	260.510010	91449900	TSLA	0.083333	Bullish
3	2021-10-05	261.600006	265.769989	258.066681	260.196655	260.196655	55297800	TSLA	0.494000	Bullish
4	2021-10-06	258.733337	262.220001	257.739990	260.916656	260.916656	43898400	TSLA	0.715000	Severe Bullish

Fig 2.2

AMZN is mainly testing our models, and MSFT is tested to compare with AMZN.

Support Vector Regression

Of the available regression models, support vector regression (SVR) is used because of its ability to capture non-linear relations, robustness and its ability to capture multiple features. This especially works well with complex data like stocks. With others, it fails to capture complicated relations in data. For the duration of this section, AMZN is experimented on.

For partitioning data, all features were standardized, and data is split 80% for training and the remaining 20% for testing (not random). A grid search is used to determine the best parameters from a predefined list. The parameters are shown in Fig 3.1. Since the data is a time series, the folds need to be approached differently. Sklearn TimeSeriesSplit is used to properly split the data since it tries to account for the time. Data is split with 5 splits. After performing a grid search, the best parameters were: linear kernel, and C=100. C is a regularization parameter that controls the trade-off between maximizing the margin and minimizing error. A linear kernel suggests a linear hyperplane is used to fit the data. Gamma isn't used since a linear kernel is used. The RSME for the training set is 0.963. When applying these parameters to the test data,

the resulting RSME is 1.244. The RSME scores suggest that although the model fits the data very well, it could mean that the model suffers from overfitting. This also means that it can not generalize well. A plot is shown in Fig 3.2 to show that the model captured too much of the data. In conclusion, SVR wouldn't be a good model to predict and future forecasts wouldn't be too accurate. A more complex model is needed.

```
# Hyperparameters
param_grid = {
    'C': [1, 10, 100, 1000],
    'gamma': [0.001, 0.01, 0.1, 1],
    'kernel': ['linear','rbf','poly']
}
```

Fig 3.1

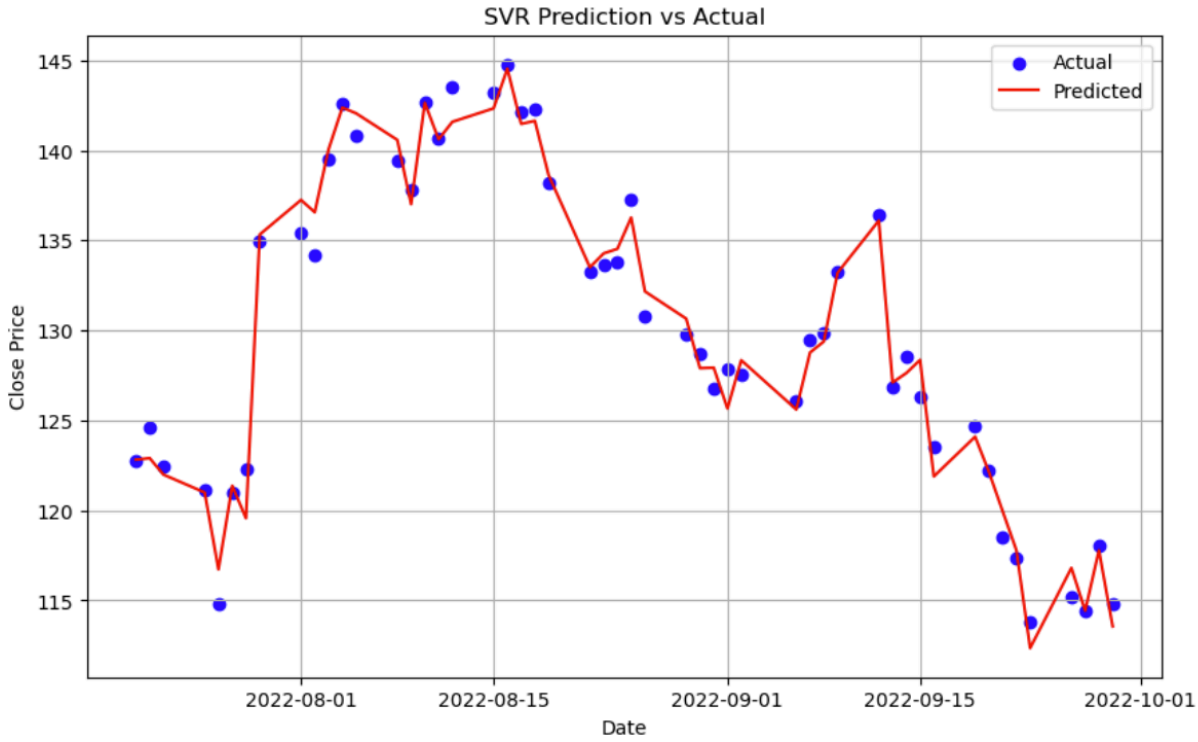

Fig 3.2

Recurrent Neural Network

Recurrent neural networks (RNN) is useful because it is one of few models that uses supervised learning on sequences. Before applying an RNN to the data, the data needs to be partitioned differently. Same thing as before, the data is split the same way and

standardized the same way too. After splitting the data, the train data of the features and the target is processed into a new dataset. In this process, a new dataset is created. This dataset contains sequences of 5 rows from the train data of the features. The creation is iterative until the first item in the sequence is the 5th to last row. The same is applied to the test data except it's just the target. This function is then applied to the test data too. The goal of creating sequences was to have the model able to learn smaller patterns. The feature data is reshaped for the RNN where the shape is: (number of rows, timestamp [length of sequence], number of features).

For the RNN, the goal is to implement a model where it contains a layer(s) using LSTM (long short term memory). This is because the model should learn and retain information about a sequence. LSTM in this case is also good because it avoids common problems like vanishing gradients. LSTM also controls what information should be forgotten since not all past stock prices are relevant to predicting future prices. Furthermore, the model should have an attention layer as well since learning patterns in stocks are complicated and want the model to pay attention and look for complex patterns, especially at a local level. This led to the architecture of the RNN to this:

1. Input layer
2. Bidirectional LSTM layer
3. Attention layer
4. LSTM layer
5. Dropout layer
6. Dense layer

```
# model
model = Sequential([
    Input(shape=(X_train.shape[1], X_train.shape[2])),
    Bidirectional(LSTM(100, return_sequences=True)),
    AttentionLayer(timesteps=X_train.shape[1]),
    LSTM(50, return_sequences=False),
    Dropout(0.3),
    Dense(100, activation='relu'),
    Dense(1)
])
```

Fig 4.1: Architecture of RNN.

Bidirectional was used to learn past and future sequences. The model should learn dependencies between past and future. The attention layer was constructed such that it assigns scores for each element in the inputted sequences. A dense layer equipped

with the softmax function is used to calculate the weights for each element. The weights are multiplied with each element to gain information about what is considered more relevant. After constructing an attention layer, another LSTM layer is added to add more complexity into the model. A dropout layer is used for regularization and a dense layer for regression. The units in the LSTM layer were fine-tuned and it was determined that the best number of units was 50. Bidirectional is twice the amount of units compared to the LSTM layer. Dropout layer (used for regularization) was tested with different parameters and was determined that dropout of 0.3 was a good parameter. The RSME from predicting with the test data is 5.945. **Fig 4.2** shows a visualization of the predicted price and actual price.

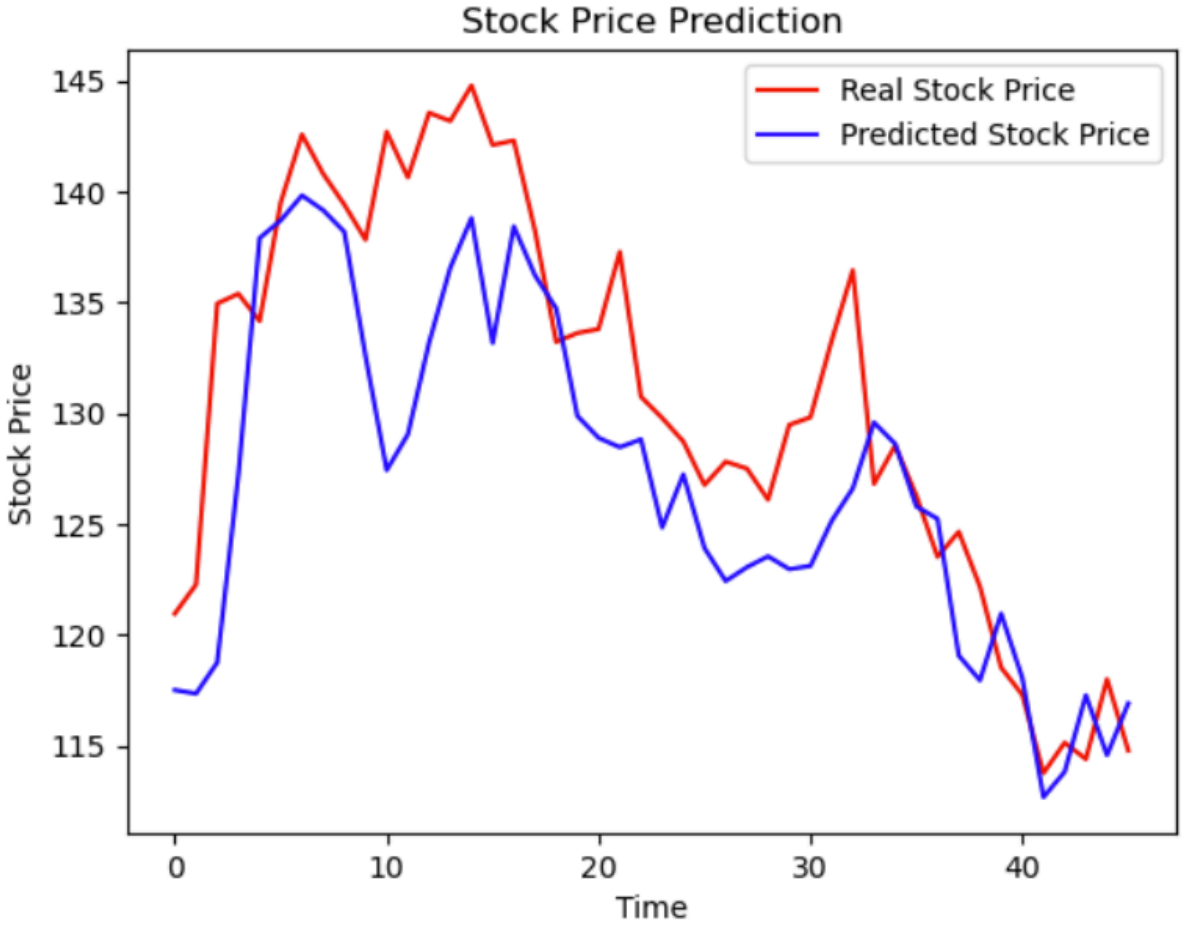

Fig 4.2

Forecasting

To see how well the model generalizes well, forecasting tasks were assigned to test its performance. Since market sentiment is a short term feeling, prices were forecasted in a 5 day interval because 5 days represent a week and to see performance in the short term. Anything beyond the timeframe, market sentiment loses predicting power.

For the project to make a forecast, it uses the same data predicted for the last 5 days as the input and calls the model to make a prediction for the next day based on the data. The oldest price is removed and similar to a sliding window the input is updated with the new forecast (both price and sequence). This ends until the list contains 5 new predicted prices. Data from Yahoo Finance is used to compare the forecasted price and actual price of the stock. RSME is used to measure the accuracy of the forecast. **Fig 4.4** shows the visualization of the forecast compared to the actual price. The RSME for the forecast is 3.806.

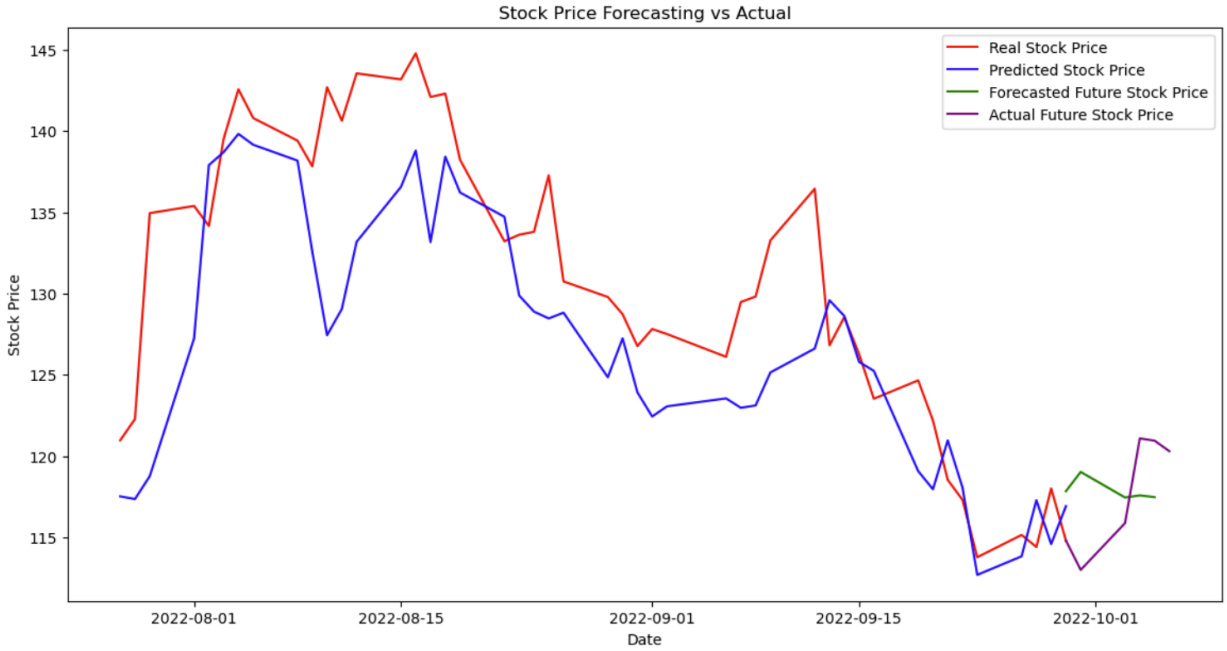

Fig 4.4

Microsoft

Fig 5.1 (model predictions) and **Fig** 5.2 (forecasts) show both the results of applying the same RNN model on the MSFT stock. The RSME in **Fig** 5.1 is 10.105 and **Fig** 5.2 is 15.130. This implies that the RNN doesn't generalize too well with other stocks. This leads to the conclusion that the current model isn't as sophisticated to capture complex patterns in stock prices. What might be considered successful is if the RSME is similar to the ones tested on AMZN.

Discussion

I think the project was a success to a certain extent. While the RNN model was able to capture the time series for AMZN, I don't think it was able to capture the other stocks

very well. Looking at the forecast I noticed that the direction of the plotted forecast moves in one direction after a certain period of time. This implies the extent of the RNN model can't predict well in further days in the future (5 days might be too hard for the model to predict). For the duration of the project, different layers were tested on the AMZN stock prices. One model only had LSTM layers and another model used a CNN layer. The RSME for those were close to 8 (predicted and actual). Given more time, more layers or different layers would be used. Although I did try to use more layers before, it increased the RSME of the forecasts; so more time will be needed to further investigate this. The methodology of the project showed it wasn't the best since it uses the summary of market sentiment for the day so it can't really reflect more instant changes in the market.

A problem I ran into with the project was properly plotting the forecasts. In the plot when connecting actual price or forecasting price, the lines would not connect. I tried attempting to fix it, but don't know what was causing the problem. Another problem was implementing a method to see the significance of each variable in the SVR. The use of shapley values was attempted, but wouldn't work with the SVM model. A grid search was attempted on the neural network, but when I implemented it, it gave me a high RSME. The grid search was scratched off and the goal was to try and fine tune the model instead. For additional metrics, cosine similarity was attempted, but the similarity indicated that the forecast and actual close price are very similar.

The inspiration of this project came from looking at a research paper a friend showed me about using models to predict stocks. It used SVR and LSTM to predict, and I wanted to attempt to tackle the project with the same idea. I was interested in sentiment analysis and thought this project was a good way to apply it. When looking at this dataset in Kaggle, there were notebooks on other coders' approaches to predicting stocks. Most commonly used was a GANs model. Given more time, In the future, I would like to try my own approach to a GANs model since it incorporates elements of an RNN. The RNN model was not as complex compared to other GANs models, but it does highlight the complexity of predicting stocks.

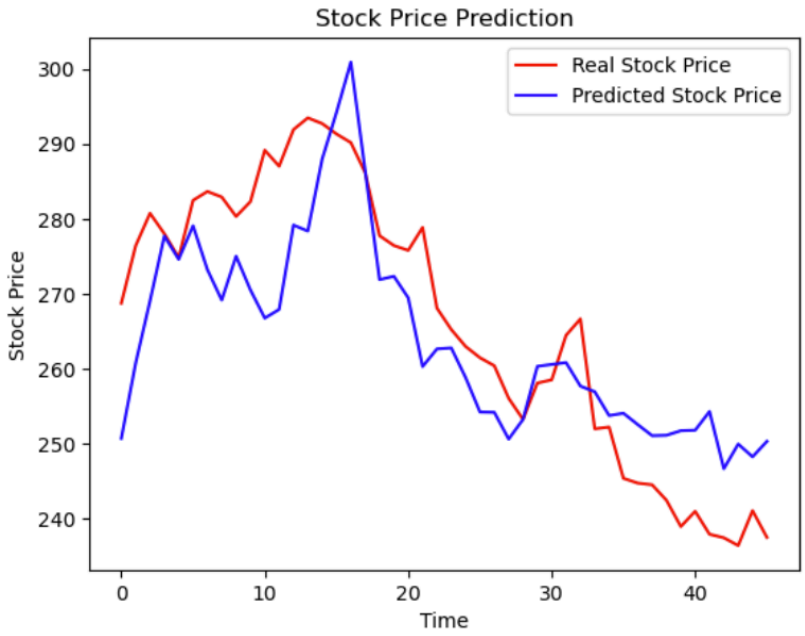
Stock Price Prediction
Real Stock Price
Predicted Stock Price
Stock Price
Time
300
290
280
270
260
250
240
0
10
20
30
40

Fig 5.1

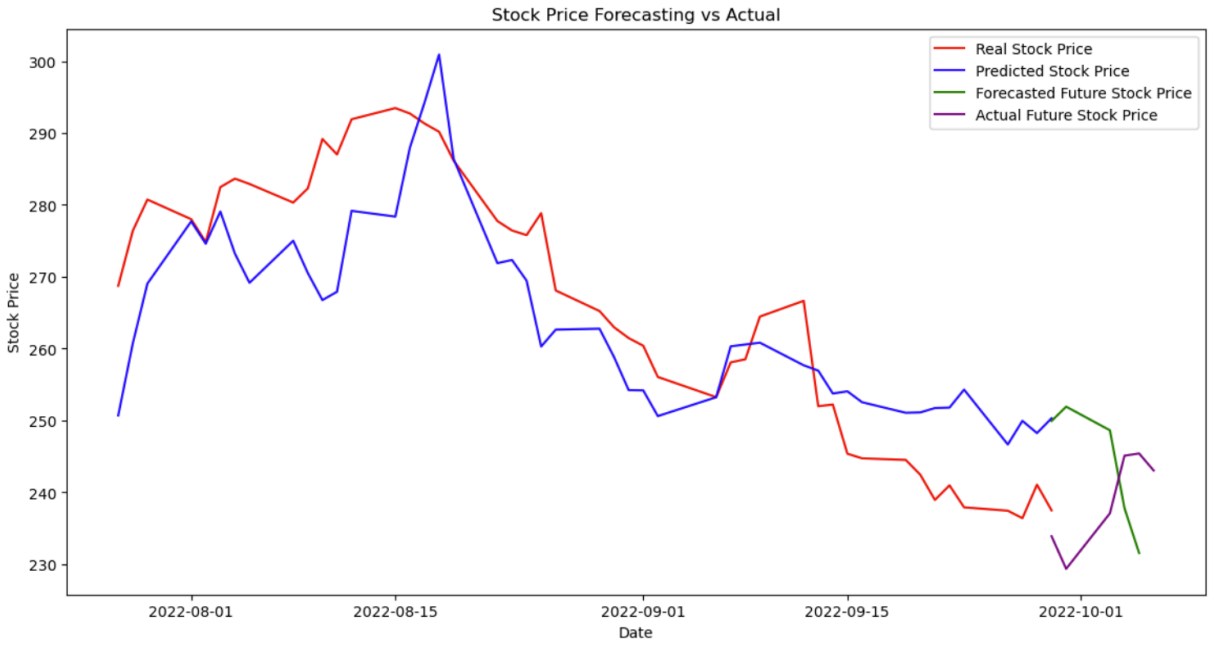
Stock Price Forecasting vs Actual
Real Stock Price
Predicted Stock Price
Forecasted Future Stock Price
Actual Future Stock Price
Stock Price
Date
300
290
280
270
260
250
240
230
2022-08-01
2022-08-15
2022-09-01
2022-09-15
2022-10-01

Fig 5.2

Resources

https://www.kaggle.com/datasets/equinxx/stock-tweets-for-sentiment-analysis-and-prediction/data