Table Of Contents

Option B - Stock Price Prediction System

COS30018 - Intelligent Systems | https://github.com/gshiva53/COS30018-102262514/tree/main/pythonProject

Name: Shiva Gupta
Student ID: 102262514
Unit Code: COS30018
Wiki reports:https://github.com/gshiva53/COS30018-102262514/wiki

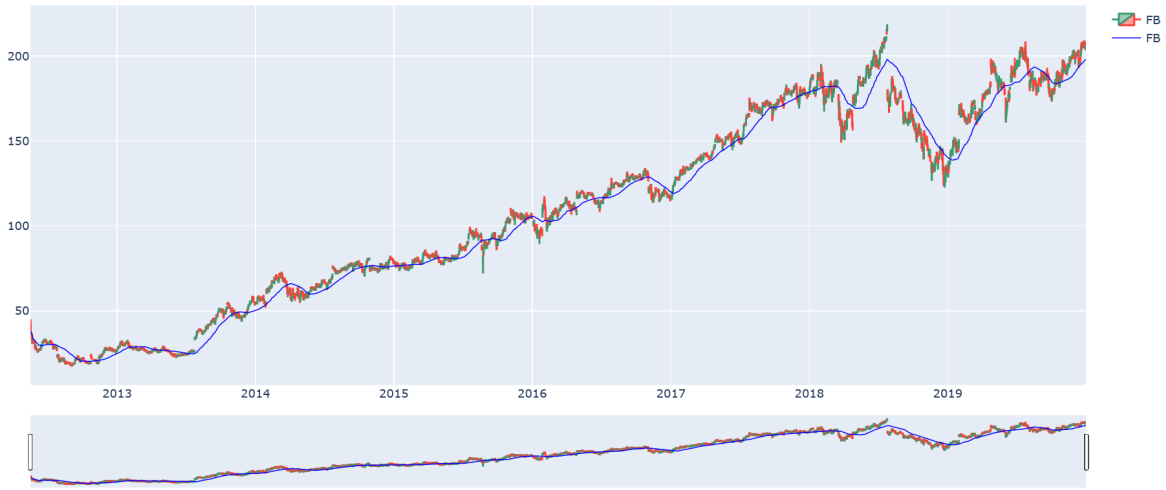
I declare this is an accurate description of team contributions of the team members

Team Member Name [Student ID]	Signature	Date
Shiva Gupta [102262514]	Shiva Gupta	11/02/2021

Introduction

This project aims to predict the stock price of a company using the historical stock price of the company. This system is built upon the existing system by NeuralNine. During this semester, there were many changes made to the existing system with many features added and the development was undertaken using the Iterative model. After a major change the codebase was tagged as a new version. The current system can predict the stock prices of a specified company using different machine learning models.

An image of **actual stock prices** vs **system predicted prices** for the company **FB**.



How to run the project on your machine

[Source: https://github.com/gshiva53/COS30018-102262514/wiki]

To run the project on your machine simply:

1. Create a virtual environment

```
$ python -m venv <name>                    // conventional name is 'env'
```

   To use the environment in Windows platform

```
$ <name>\Scripts\activate.bat
```

   NOTE: Since we need to run a batch file so we need to use the `cmd`, we can not use the `powershell`.

   To deactivate the virtual environment

```
![2_Stock Prediction System Architecture](D:\Semester 2 2021\COS30018 - Intelligent Systems\COS30018-102262514\snips\Project Summary Report\2_Stock Prediction System Architecture.png)>deactivate
```

2. Install the required libraries

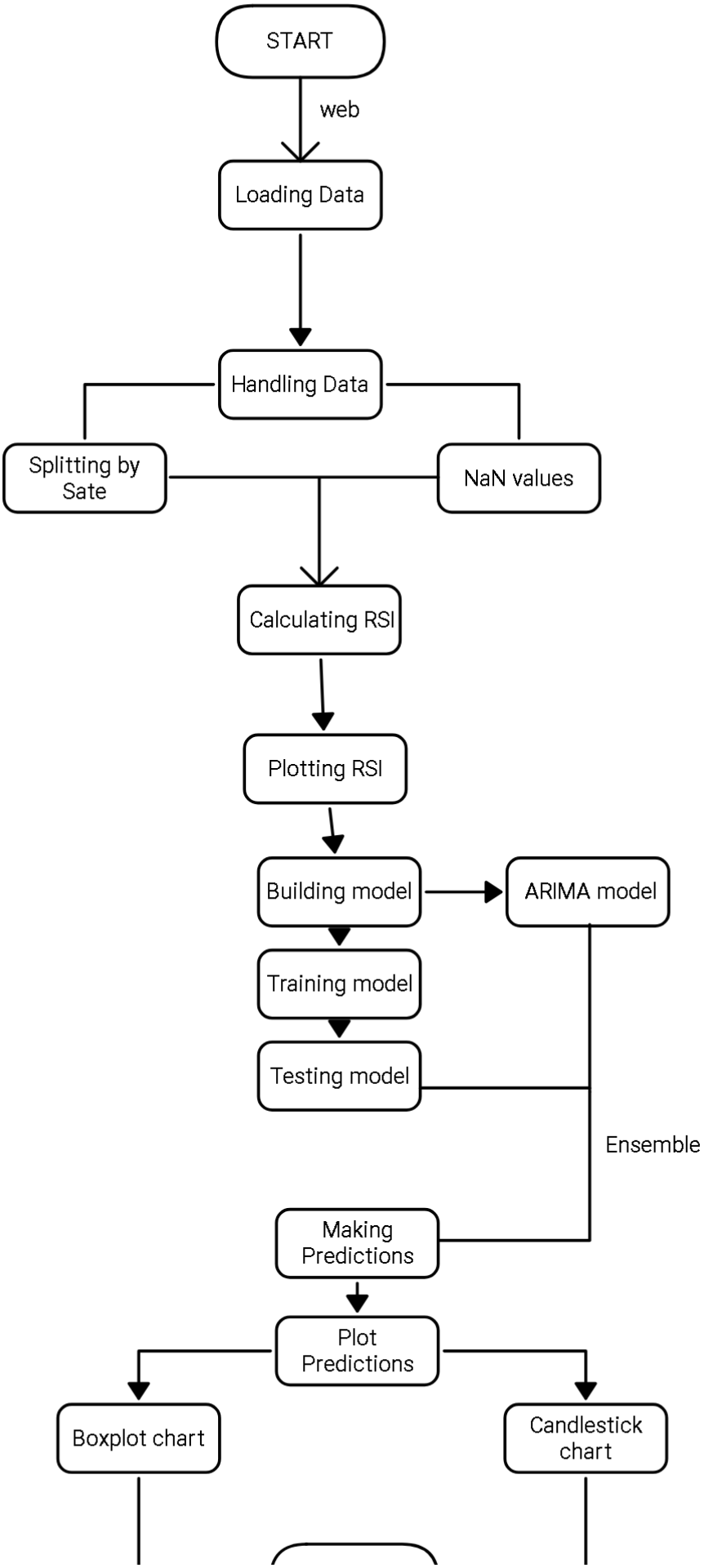
```
$ pip install numpy
$ pip install matplotlib
$ pip install pandas
$ pip install pandas-datareader
$ pip install tensorflow
$ pip install scikit-learn
```

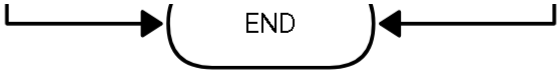
3. Run the code

```
>python ./main.py
```

System Architecture

The diagram below demonstrates the internal working of the stock prediction system. There are three major stages which happen one after another, namely, **Loading and Handling Data**, **Building the Model**, **Plotting the predictions**.


START
web
Loading Data
Handling Data
Splitting by Sate
NaN values
Calculating RSI
Plotting RSI
Building model
ARIMA model
Training model
Testing model
Ensemble
Making Predictions
Plot Predictions
Boxplot chart
Candlestick chart



Some of the stages will be visited in the upcoming sections.

Data Processing Techniques Used

[Source: https://github.com/gshiva53/COS30018-102262514/wiki/%5BB.2%5D-Data-processing-1]

[Source: https://github.com/gshiva53/COS30018-102262514/wiki/%5BB.3%5D-Data-processing-2]

The current system uses different techniques to process the data so that the model can be trained from quality data. Some of the techniques are described below, however, the detailed implementation details are mentioned in the weekly reports sourced above.

Specifying the Dates for the Data set

In the existing code base we have saved the dates and use them to load the data for training and testing, however we can do this in a load function and assign the existing values as default values so the code still runs as previous but it can be changed.

```
// Default values for the funtion
TRAIN_START = dt.datetime(2012, 5, 23)     # Start date to read
TRAIN_END = dt.datetime(2020, 1, 7)       # End date to read

// Function definition
def load_data(... , start_date = TRAIN_START, end_date = TRAIN_END);

//function usages
load_data()
//function usage with dates
load_data(dt.datetime(2015, 8, 27), dt.datetime(2020, 8, 27))
```

The dates are then used to read the data from the `web.dataReader()` so we need to change it there.

```
data = web.DataReader(ticker, DATA_SOURCE, start_date, end_date)
```

Dealing with NaN

[Resource: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.dropna.html]
`NAN` means not a number, so we need to deal with the values that are not numbers in the data.
We can use the `.dropna()` function provided in the pandas library in the DataFrame module.
This function removes the missing values from the DataFrame object.
The `implace=True` flag is true then it performs the function inplace and returns none.

```
data = data = web.DataReader(ticker, DATA_SOURCE, start_date, end_date)
    ...

//Remove the NAN values
data.dropna(inplace = True)
```

Storing and Loading Data

For our context the data is the data frame being read from the web using the pandas library. We are directly loading and using the data but we can load and store it locally using the function below

```
    if isinstance(ticker, str):
        data = web.DataReader(ticker, DATA_SOURCE, start_date, end_date)
    elif isinstance(ticker, pd.DataFrame):
        data = ticker
    else:
        raise TypeError("ticker can be either a str or a 'pd.DataFrame'
instance")
```

Scaling Feature Columns

We are required to save the scaler that we are using to scale our data so that the scaler can be used later and also to scale our feature columns to include different types of prices like when the stock market is 'Open' etc.
We can do this by using a list of string values for different feature prices like

```
FEATURE_COLUMNS = ['Open', 'High', 'Low', 'Volume', 'Close', 'Adj Close']
    ...
def load_data(... , feature_columns = FEATURE_COLUMNS, ...)
    ...
```

Now, for different feature columns we can also check if the data has columns for every feature price using the assert statement.

```
for col in feature_columns:
    assert col in data.columns, f"'{col}' does not exist in the dataframe."
```

Visualizing the Data using different charts

`trace` can be thought of as the options for plotting graphs. Multiple traces can be used consecutively within the same graph.

The requirements state that we need to express the data for variable trading days and this can be achieved by plotting the average for specified days.

```
# avg_window is the input the argument to the function
avg = data['Close'].rolling(window=avg_window, min_periods=1).mean()
```

This `avg` can be plot in the y-axis and it can display the **moving averages**.

```
trace2 = {
        'x': data.index,
        'y': avg,
        'type': 'scatter',
        'mode': 'lines',
        'line': {
            'width': 1,
            'color': 'blue'
        },
        'name': company,
        'showlegend': True
    }
```

Plot the graph

```
fig = go.Figure(data=[trace1, trace2])
fig.show()
```

Machine Learning Techniques and Ensemble Models

The system follows a modular approach for building a machine learning model. Arguments can be specified while calling the function and different models can be built thereof.
Arguments for Creating the model

There are multiple arguments needed for building the model like the number of layers, type of network, length of the sequence etc. In the current code we are building a layer then adding the dropout and then building another layer and again adding the dropout. This process can be automated in the code. Also, the current network type is LSTM.

list all the arguments that are needed to build the model. Here we are creating a skeleton method in which we will later implement building the model.

```
def create_model(sequence_length, n_features, units, cell, n_layers, dropout,
                 loss, optimizer, bidirectional):
    ...

create_model()
```

Building the Layers using Arguments

The model is sequential and we create it first. Then for building the first layer we check if we want the layer to be bidirectional, if it is True then the first layer is a bidirectional layer otherwise it is a LSTM layer by default.

```
model = Sequential()
    for i in range(n_layers):
        if i == 0:
            # first layer
            if bidirectional:
                model.add(Bidirectional(cell(units, return_sequences=True),
batch_input_shape=(None,                          sequence_length, n_features)))
            else:
                model.add(cell(units, return_sequences=True, batch_input_shape=
(None, sequence_length,
```

After we are done building the first layer then we build the next layer. Note that the number of layers are variable so we are building the first layer by ourselves and the last layers will all be of the same type

For last layer or layers because all the layers will be of the same type after first layer so

```
        elif i == n_layers - 1:
            # last layer
            if bidirectional:
                model.add(Bidirectional(cell(units, return_sequences=False)))
            else:
                model.add(cell(units, return_sequences=False))
```

We are also going to add a hidden layer to the model which will have the return_sequences argument set to true.

```
      else:
          # Hidden layers
          if bidirectional:
              model.add(bidirectional(cell(units, return_sequences=True))
          else:
              model.add(cell(units, return_sequences=True))
```

After every layer we are going to add a dropout layer.

```
 model.add(Dropout(dropout))
```

Then we will compile the model

```
model.compile(optimizer=optimizer, loss=loss)
```

Then we return the model back to the caller and this way we have automated the model building process using some extra arguments.

Call the create_model function and fix that model to predict the stock prices.

```
model = create_model(50, 1)
model.fit(x_train, y_train, epochs=3, batch_size=32)
```

ARIMA model

building the model on existing data. The data is separated in 70:30. major chunk is used for building the model and then the model is tested on the minor chunk.

```
for time_point in range(N_test_observations):
    model = ARIMA(history, order=(4, 1, 0))
    model_fit = model.fit()
    output = model_fit.forecast()
    yhat = output[0]
    model_predictions.append(yhat)
    true_test_value = test_data[time_point]
    history.append(true_test_value)
```

Calculating Errors

After predicting, we will calculate the mean squared error for the actual price vs the predicted price.

```
MSE_error = mean_squared_error(test_data, model_predictions)
print('Testing Mean Squared Error is {}'.format(MSE_error))
```

Ensemble LSTM AND ARIMA model

We simply take the predictions from both the models and average them out for the specified range.

```
combined_prediction = (prediction[-1] + model_predictions[-1])/2
print(f"Prediction using ensemble approach: {combined_prediction}")
```

Extensions Implemented

[Source: https://github.com/gshiva53/COS30018-102262514/wiki/%5BB.7%5D-Extension]

The current model was extended to calculate the RSI - Relative Strength Indicator which can predict bearish or bullish momentum. It can also be further extended to work with several different indicators like trendline application, Advanced RSI etc. The importance and implementation is documented in detail at the source link.
A brief view at how the RSI is being implemented in the system

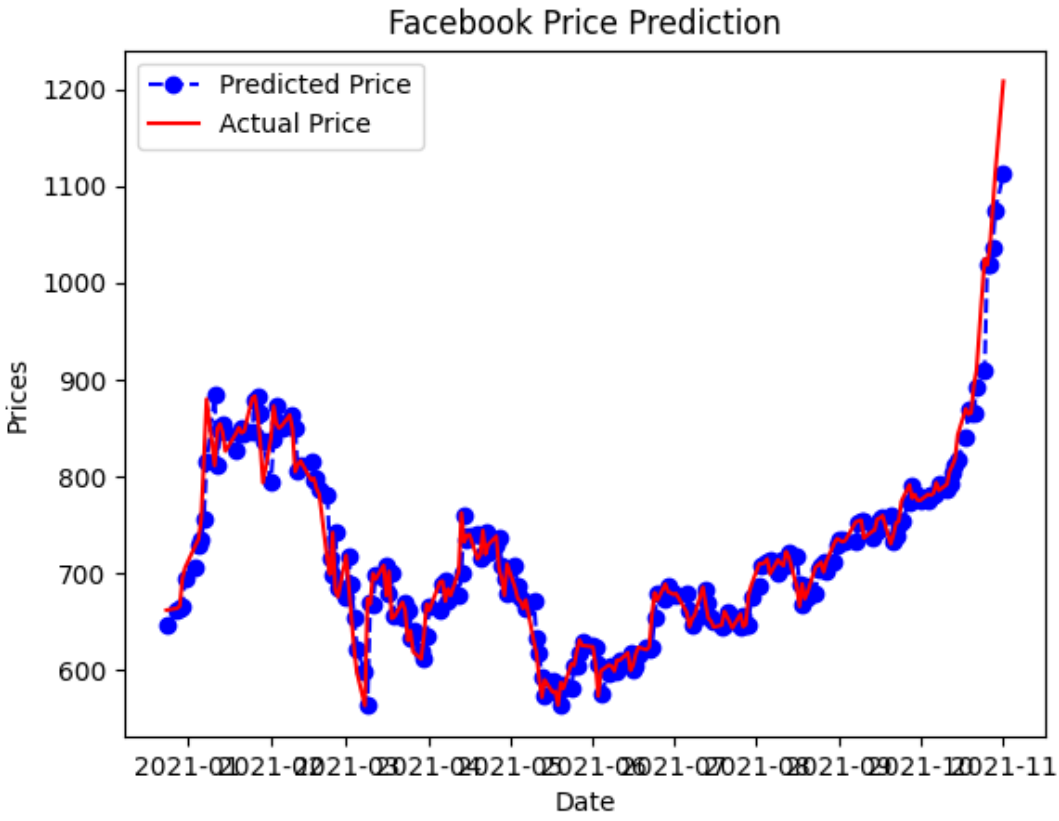
```
def Plot_RSI(days):
    delta = data['Adj Close'].diff(1)
    delta.dropna(inplace=True)

    positive = delta.copy()
    negative = delta.copy()

    positive[positive < 0] = 0
    negative[negative > 0] = 0

    days = days  # default is 14
```

```
    average_gain = positive.rolling(window=days).mean()
    average_loss = abs(negative.rolling(window=days).mean())

    relative_strength = average_gain / average_loss
    RSI = 100.0 - (100.0 / (1.0 + relative_strength))

    combined = pd.DataFrame()
    combined['Adj Close'] = data['Adj Close']
    combined['RSI'] = RSI
    # ----- plotting -----
```

The RSI is then plotted with the adjusted close columns of the data and the plotted graphs can then be analyzed to understand **overbought** and **oversold** conditions.

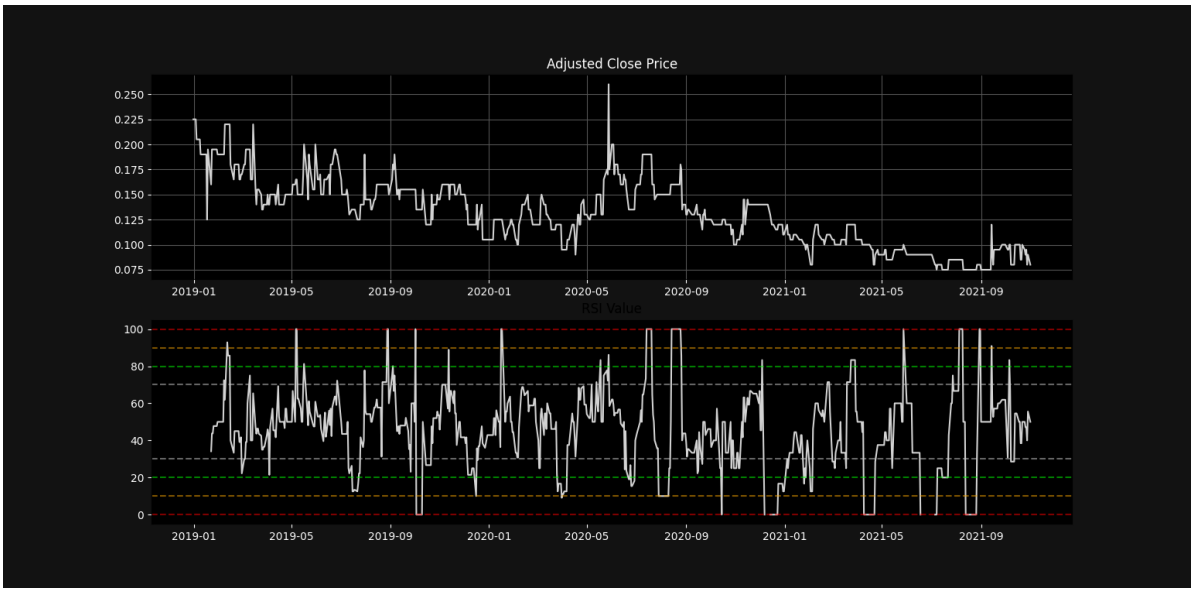
Examples of the system working

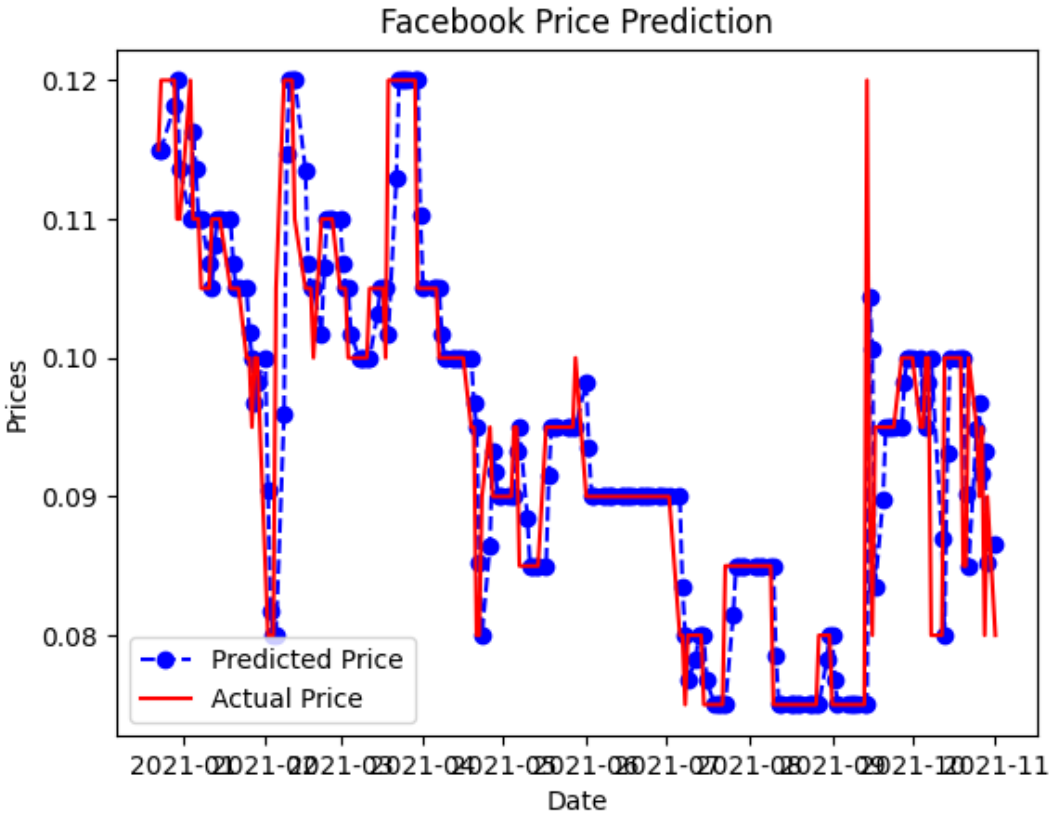
These are some of the screenshots for the company Tesla('TSLA') from January 1, 2019 till date.




```
Prediction using LSTM model: [[809.51044 864.7991  782.2467  816.45416 855.6156  855.9849  835.69464
  823.3504  908.8454  819.2185  840.20905 842.8835  858.60486 779.6244
  790.96405 822.2808  896.1395  826.90625 787.29047 789.07745 833.70496
  827.35266 808.58936 840.4298  843.90283 810.7834  825.94226 878.7425
  855.7844  847.3764  804.33923 800.8054  755.3626  853.11334 814.5612
  848.62225 845.6264  898.65656 816.8827  876.51965 864.723   793.2319
  866.9588  840.6239  867.3453  832.9964  768.0025  854.02576 873.44025
  849.1101 ]]
Prediction using ARIMA model: [645.8111162837743, 661.3190387559694, 663.6362628154969, 665.9257728571993,
Prediction using ensemble approach: [ 961.05994  988.7043   947.4281   964.53186  984.11255  984.29724
  974.1521   967.98    1010.7274   965.914    976.4093   977.74646
  985.6072   946.11694  951.78674  967.4452  1004.3745   969.7579
  949.94995  950.8435   973.1572   969.9811   960.5994   976.51965
  978.25616  961.6964   969.2759   995.676    984.19696  979.9929
  958.47437  956.70746  933.9861   982.86145  963.5853   980.61584
  979.1179  1005.63306  964.7461   994.5646   988.66626  952.92065
  989.7842   976.6167   989.9774   972.803    940.306    983.3176
  993.0249   980.8598 ]
```

These are some of the screenshots for the company Commbank ('CBA') from January 1, 2019 till date.






```
Prediction using LSTM model: [[0.09172091 0.09468768 0.09810299 0.09560612 0.09475182 0.0917671
  0.09239723 0.09223039 0.09373176 0.09573295 0.09209907 0.09254591
  0.09322361 0.09264181 0.09265426 0.0969482  0.0941699  0.09270103
  0.09669383 0.09527474 0.09339685 0.09422071 0.09484297 0.09461667
  0.09482034 0.09279595 0.09296869 0.09731873 0.09406314 0.09503617
  0.09385629 0.0932472  0.09283087 0.09087756 0.09400148 0.0932722
  0.09402364 0.09273911 0.0932033  0.09249821 0.09476183 0.09338212
  0.09768928 0.09359765 0.09020522 0.08688834 0.09386557 0.0924694
  0.09161488 0.09442657]]
Prediction using ARIMA model: [0.11500000208616257, 0.11500000208616257, 0.11818067861165461, 0.11999999731779099,
Prediction using ensemble approach: [0.08913325 0.09061664 0.09232429 0.09107585 0.09064871 0.08915634
 0.08947141 0.089388   0.09013867 0.09113927 0.08932233 0.08954576
 0.0898846  0.0895937  0.08959992 0.0917469  0.09035775 0.08962331
 0.09161972 0.09091017 0.08997122 0.09038315 0.09069428 0.09058113
 0.09068297 0.08967077 0.08975714 0.09193216 0.09030437 0.09079088
 0.09020095 0.0898964  0.08968823 0.08871157 0.09027354 0.0899089
 0.09028462 0.08964235 0.08987445 0.0895219  0.09065372 0.08996385
 0.09211744 0.09007162 0.0883754  0.08671696 0.09020558 0.0895075
 0.08908024 0.09048608]
```

Some other examples of the system working are:

- Facebook FB from January 1 2020 till date.
- Amazon AMZN from January 1 2019 till date.

The results for these are available in the Git Repo under snips.

Critical Analysis

Although the system is revamped with many new features like safely handling data, using momentum indicators and use of different machine learning models it still has some shortcomings. A few of the shortcomings are discussed about below.

Inaccurate Ensemble Approach

The current ensemble approach averages the predictions from the two models, namely, LSTM and ARIMA model and makes a prediction but this means that the final value depends equally on both the values to be accurate for the final prediction to be accurate. But the model currently can't make accurate predictions.
A better approach would be to use the calculated mean squared error. We can calculate how much every value contributes to the total error and then use the individual error contribution and subtract it to get an accurate value.

Training the model on spikes and dips

As evident from the screenshots of the TSLA price prediction the model is highly inaccurate while predicting stocks that are comparatively volatile. So, the model needs to be trained on such scenarios.

Better predictions for average rise or fall in stocks

The current system is comparatively accurate in predicting the prices for an average rise or fall of stocks as seen in the case of FB with the ARIMA model predicting with only 12.65 mean squared error.

Conclusion

Reviewing this semester and the system that I worked on, I have learnt important lessons in the field of AI especially in machine learning. I learned about plethora of concepts ranging from how to build, train, test the model to honing my problem solving skills. However, there are many shortfalls of the current system, it is definitely an improvement from the old codebase.