Stock Prediction for Apple

Prepared By:

Manthiramoorthy Cheranthian

For

World Data Science Institute

Steps Followed:

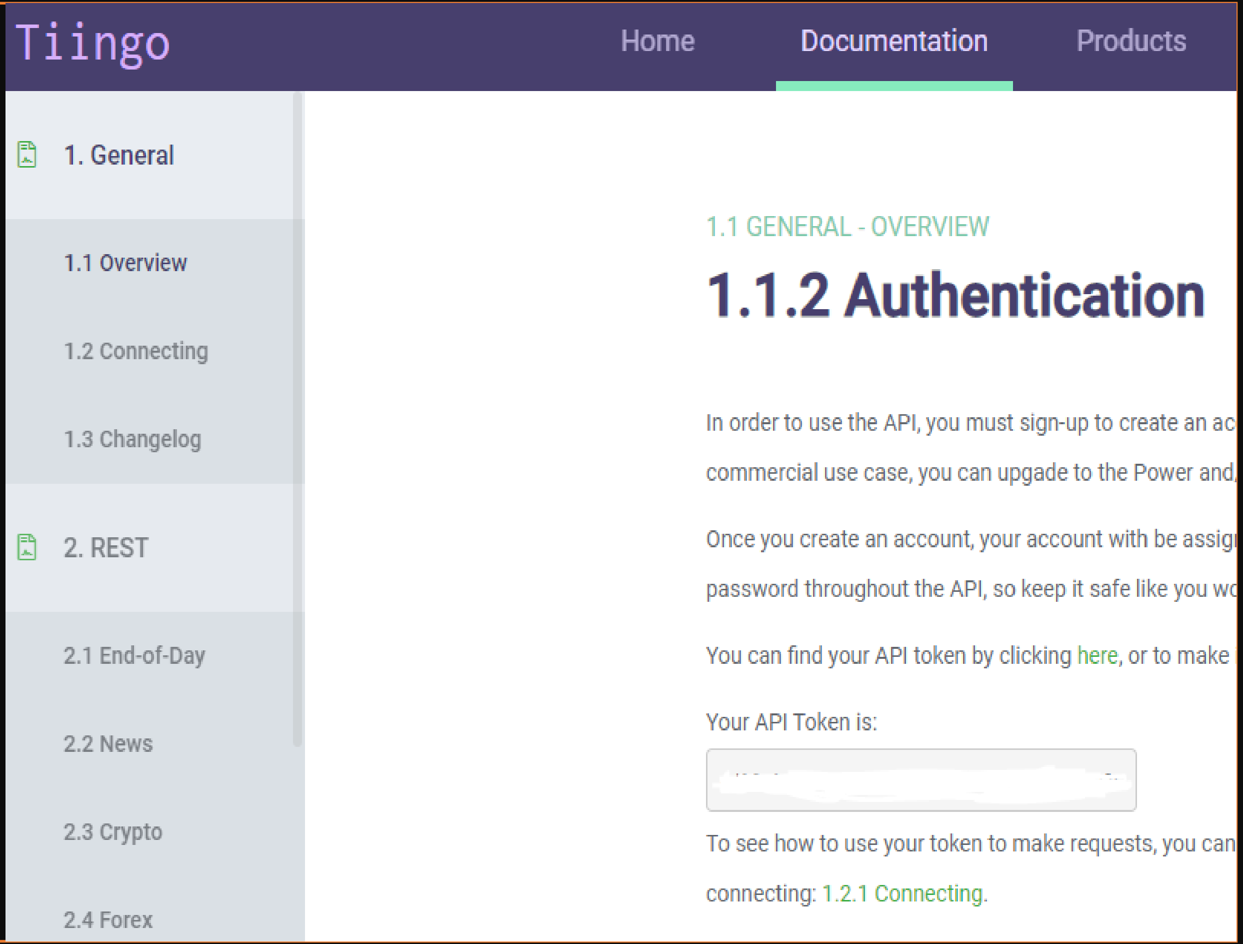
- Data Collection
- Data Preprocessing and data splitting
- Defining a stacked LSTM model
- Prediction and Performance metrics(RMSE) on train and test data.
- Forecast the stock data for 7 future days.

**Kindly find the attachment for the .ipynb file for more details*

AAPL_Prediction_LSTM.ipynb

Data Collection:

- We use pandas reader library to collect data for apple stock by calling Tiingo API than using yahoo csv.
- To use get_data_tiingo method the user need to create an account with Tingo using this link (https://api.tiingo.com/).
- Once signed in get the API token from the below page.
- Collected data is stored in a pandas data frame and a new data frame is created using Closing values only.



Data Collection using pandas reader

```
[51] ##Data Collection for stocks using pandas_datareader library
     import pandas_datareader as pdr
```

```
##Create an account at https://api.tiingo.com/ to get stock data by hitting tingo API
##Storing the stock history data of Apple in a dataframe df by calling get_data_tiingo method
key='xxxxxxxxxxxxxxxxxxxxxxxxxxxx'
df=pdr.get_data_tiingo('AAPL',api_key=key)
```

```
[53] df.to_csv('AAPL.csv')
```

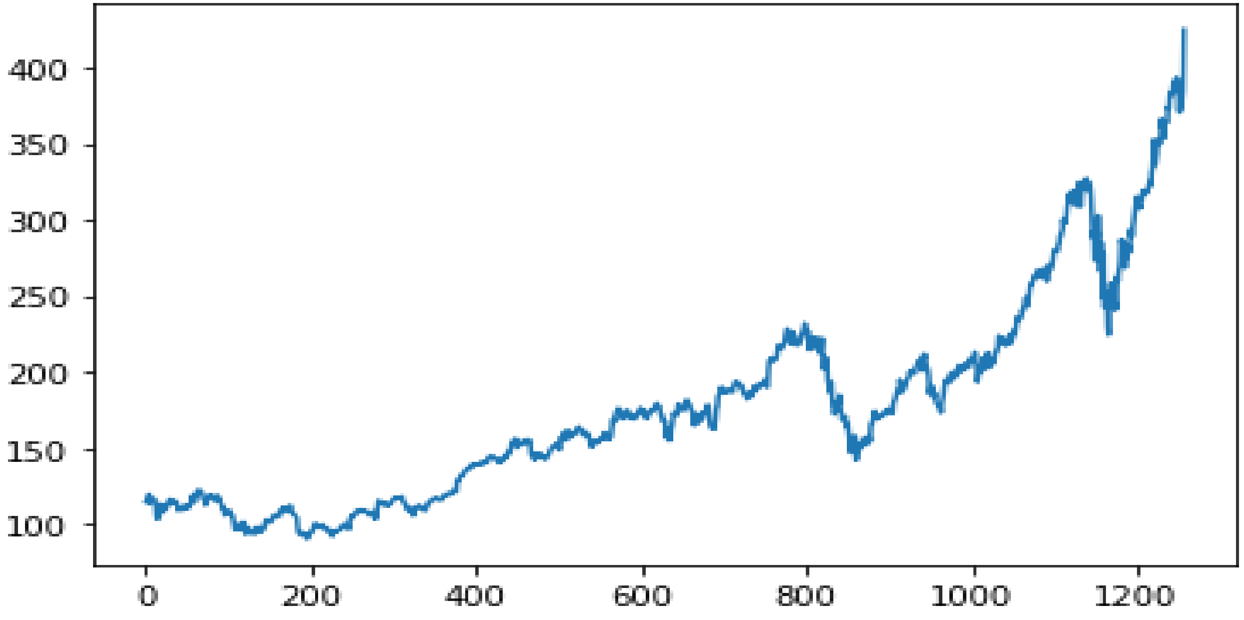
```
[54] import pandas as pd
     df=pd.read_csv('AAPL.csv')
```

```
[55] ##Subsetting the data, only using closing values
     df1=df.reset_index()['close']
```

Visualizing the stock Closing data:

```
##Plotting the closing values of the Stock
import matplotlib.pyplot as plt
plt.plot(df1)
```

```
[<matplotlib.lines.Line2D at 0x7fa4047c12e8>]
```



Data Scaling:

```
##Scaling the data in the range of 0 to 1 before passing to the LSTM Model, Since LSTM is sensitive to scale
##We apply Min Max Scaler
import numpy as np
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler(feature_range=(0,1))
df1=scaler.fit_transform(np.array(df1).reshape(-1,1))
```

```
##Now the closing values are converted in the range between 0 and 1
print(df1)
```

```
[[0.07260233]
 [0.07487302]
 [0.07406633]
 ...
 [0.86590977]
 [0.87965342]
 [1.        ]]
```

Data Partition:

Data Partition (Train - Test Split)

Time Series Intuition: Time series data points are dependent on previous day's values. Hence Splitting the data to train and test should be splitted in such a way that the order is preserved based on date.

ex: Date - Closing Price

1. 01-jun - 332,
2. 02-jun - 331,
3. 03-jun - 335,
4. 04-jun - 334,
5. 05-jun - 333

while splitting the above data we could not use random split we can divide in a way that order is maintained

Train-

- 01-jun - 330
- 02-jun - 332
- 03-jun - 331
- 04-jun - 335

Test -

- 05-jun - 334,
- 06-jun - 333

```
##Splitting the data into train and test split
##First 70% goes to train data, remaining 30% test data
train_size=int(len(df1)*0.70)
test_size=len(df1)-train_size
train_data,test_data=df1[0:train_size,:],df1[train_size:len(df1),:]
```

Choosing Time Steps:

Time Steps Intuition: *Consider Train Data:* 330, 332, 331, 335 *Consider Test Data :* 334, 333, 334, 336

When we set Time_Steps =2 Our model takes into consideration two values before the current timestamp for predicting new stock value. It works like an iteration one step forward but consider 2 consecutive values since time_Step=2

Note: Usually choosing higher time_step gives better prediction

- x1 x2 y_train (Output)
- 330 332 331
- 332 331 335
- 331 335 333

Similarly for Test data

- x1 x2 y_test (Output)
- 334 333 334
- 333 334 336

Implementing Time Steps:

Splitting in a way such that every 1-step iterative 100 features forms a record in X_train and X_test

```
##Creating a function to implement the above logic through a for loop iteration
import numpy as np

def dataset(data,time_step):

  xdata,ydata=[],[]
  for i in range(0,len(data)-time_step-1):

    a=data[i:(i+time_step)]
    b=data[i+time_step]
    xdata.append(a)
    ydata.append(b)

  return np.array(xdata),np.array(ydata)
```

```
##Choosing a bigger time_step value usually provides a stable and better prediction
time_step=100
X_train,Y_train=dataset(train_data,time_step)
X_test,Y_test=dataset(test_data,time_step)
print((X_train.shape)) ##X_train has 779 records with time_step=100 features in it
print((Y_train.shape)) ##Y_train has just the 779 target records
```

```
(779, 100, 1)
(779, 1)
```

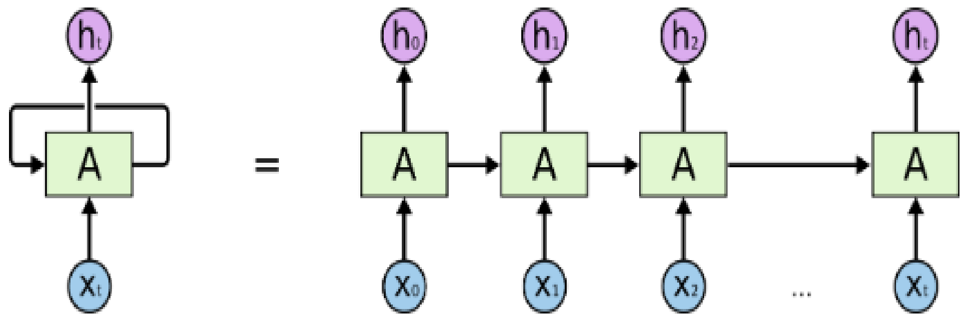
Data Reshaping:

- Reshaping the data such that the data is in required format for LSTM to work.
- If the data is already in 3-dimensional format this step could be skipped.

```
##Optional
##Make sure to Reshape the training data and testing data into a 3 dimensional if the data is in 2 dimensional form susceptible for LSTM to process
X_train=X_train.reshape(X_train.shape[0],X_train.shape[1],1)
X_test=X_test.reshape(X_test.shape[0],X_test.shape[1],1)
```

LSTM Model

- LSTM is also known as Long short-term memory.
- It follows Recurrent neural network generally used for time series data.
- LSTM allows some information to persist using their loops.
- LSTM can be assumed as multiple copies of the same network.
- LSTM has feedback connections
- LSTM can process entire sequences of data.



An unrolled recurrent neural network.

Library imports and Model Defining:

Create a Stacked LSTM Model

```
[13] ## Importing the required libraries to Create the Stacked LSTM Model
     from tensorflow.keras.models import Sequential
     from tensorflow.keras.layers import Dense
     from tensorflow.keras.layers import LSTM
     model=Sequential()
     model.add(LSTM(50,return_sequences=True,input_shape=(100,1))) ## Choosing 50 hidden layers, input shape as 100,1 as like our input dimension in previous step
     model.add(LSTM(50,return_sequences=True)) ##Since this is a stacked LSTM we add one LSTM after the other
     model.add(LSTM(50))
     model.add(Dense(1)) ##Final Output Layer
     model.compile(loss='mean_squared_error',optimizer='adam')
```

```
[14] model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 100, 50)           10400
_________________________________________________________________
lstm_1 (LSTM)                (None, 100, 50)           20200
_________________________________________________________________
lstm_2 (LSTM)                (None, 50)                20200
_________________________________________________________________
dense (Dense)                (None, 1)                 51
=================================================================
Total params: 50,851
Trainable params: 50,851
```

Fitting the Model:

```
##Fitting our LSTM model over the prepared training data
model.fit(X_train,Y_train,validation_data=(X_test,Y_test),epochs=100,batch_size=70,verbose=1)

##Decrease in Loss in every iteration is a good sign that our model reduces errors in every iteration
```

```
12/12 [==============================] - 2s 201ms/step - loss: 1.9691e-04 - val_loss: 0.0014
Epoch 73/100
12/12 [==============================] - 2s 199ms/step - loss: 1.9715e-04 - val_loss: 0.0027
Epoch 74/100
12/12 [==============================] - 2s 202ms/step - loss: 1.9017e-04 - val_loss: 0.0016
Epoch 75/100
12/12 [==============================] - 2s 200ms/step - loss: 2.0205e-04 - val_loss: 0.0023
Epoch 76/100
12/12 [==============================] - 2s 202ms/step - loss: 1.9491e-04 - val_loss: 0.0010
Epoch 77/100
12/12 [==============================] - 2s 201ms/step - loss: 1.6254e-04 - val_loss: 8.9429e-04
Epoch 78/100
12/12 [==============================] - 2s 208ms/step - loss: 1.5931e-04 - val_loss: 0.0011
Epoch 79/100
12/12 [==============================] - 2s 200ms/step - loss: 1.5851e-04 - val_loss: 9.2141e-04
Epoch 80/100
12/12 [==============================] - 2s 204ms/step - loss: 1.5681e-04 - val_loss: 9.0243e-04
Epoch 81/100
12/12 [==============================] - 2s 200ms/step - loss: 1.6878e-04 - val_loss: 0.0013
Epoch 82/100
12/12 [==============================] - 2s 196ms/step - loss: 1.5148e-04 - val_loss: 8.6561e-04
```

Prediction and Performance:

Prediction using Trained LSTM Model

```
[17] ##Predict using the model in both test and train data
     train_predict=model.predict(X_train)
     test_predict=model.predict(X_test)
```

```
[18] ##Converting the scaled values back to original values for comparison and metrics
     train_predict=scaler.inverse_transform(train_predict)
     test_predict=scaler.inverse_transform(test_predict)
```

```
[19] ##RMSE value for comparing predicted values and actual Y_train values
     ##We use math.sqrt to calculate Root value(RMSE) from the Mean squared error
     ##Train RMSE
     import math
     from sklearn.metrics import mean_squared_error
     math.sqrt(mean_squared_error(train_predict,Y_train))
```

```
152.57032125809138
```

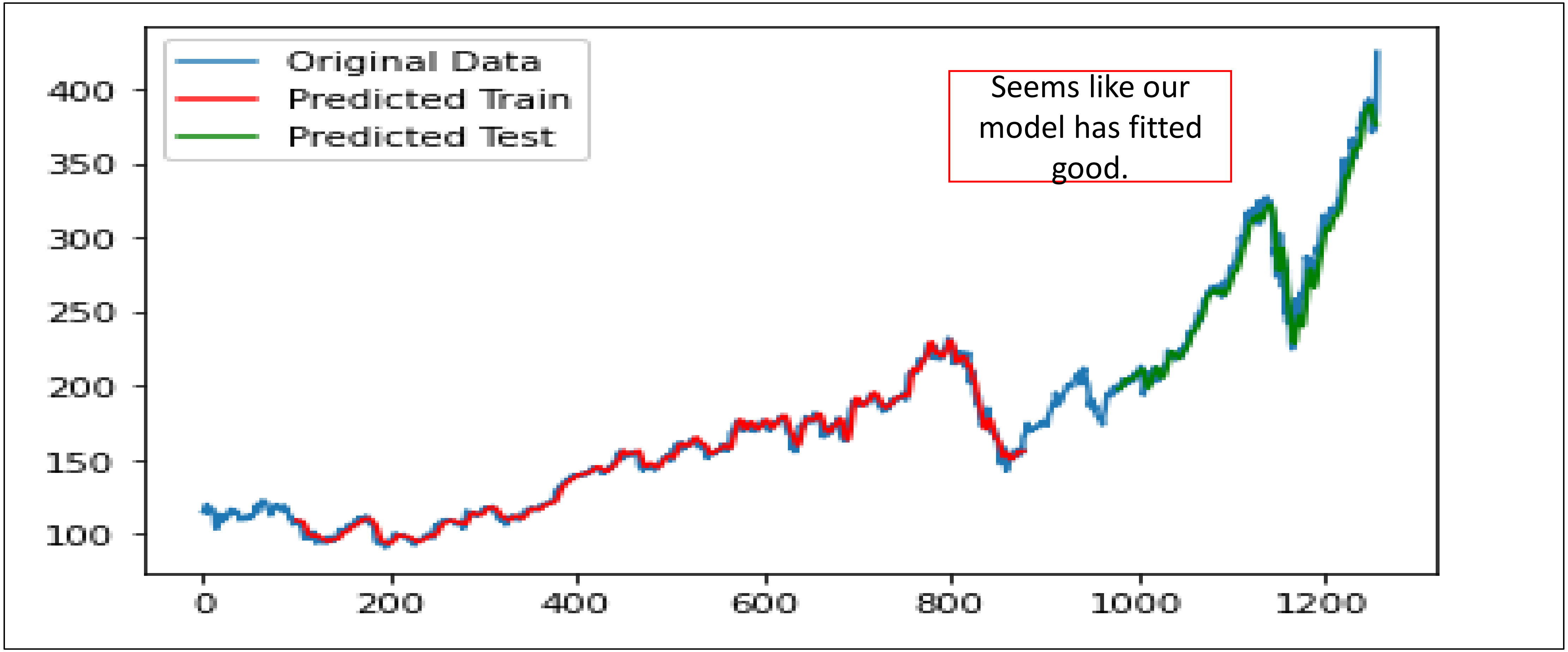
```
[20] ##RMSE value for comparing predicted values and actual Y_test values
     ##Test RMSE
     math.sqrt(mean_squared_error(Y_test,test_predict))
```

```
275.5391432373762
```

Prediction Plot vs Original Plot:

```
### Plotting
#shift train predictions for plotting
look_back=100 ##Since our time_Step is 100 we could not predict from 0th row hence plotting the train predict from 100
trainPredictPlot = np.empty_like(df1) ##Returns an array of same shape and size of input array which we pass into this function
trainPredictPlot[:, :] = np.nan        ##Replace all the values with nan
trainPredictPlot[look_back:len(train_predict)+look_back, :] = train_predict ##Replacing the NA values with predicted values
##for train from 100th row till final predicted values
# shift test predictions for plotting
testPredictPlot = np.empty_like(df1)
testPredictPlot[:, :] = np.nan
testPredictPlot[len(train_predict)+(look_back*2)+1:len(df1)-1, :] = test_predict ##Replacing the NA values with predicted values for test
# plot baseline and predictions
plt.plot(scaler.inverse_transform(df1))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
##The blue plot shows the actual stock data
##Green plot shows the test predicted values
##Orange plot shows the train predicted values
```

Plot Comparison:



Data Forecasting for 7 days:

Forecast the Stock for future 7 Days

We see that the length of our test data is 378. Since our time step value is 100, we subset from 278 as we need the last 100 values from the test set to forecast our 1st day stock price and so on

```
[22] len(test_data)

378
```

```
predict_input=test_data[278:].reshape(1,-1) ##Reshape (1,-1) means the target variable will now have 1 row and unknown number of columns
##hence it takes automatically the number of available columns
predict_input.shape
forecast_input=list(predict_input)
forecast_input=forecast_input[0].tolist()##Making the input as a list
forecast_input
```

```
[0.5530026889752016,
 0.47173588288019114,
 0.5605915745443681,
 0.4537496265312219,
 0.4855691664176875,
 0.4670749925306244,
 0.46142814460711085,
 0.4149985061248879,
```

Forecasting Logic:

```
# Prediction for next 7 days
from numpy import array
output=[]
n_steps=100
i=0
while(i<7): ##Running the loop for 7 times

    if(len(forecast_input)>100): ##System goes inide this block only after the second iteration since lenght of forecast_input list will go
    ##above 100 only after 2nd iteration
        x_input=np.array(forecast_input[1:])#to shift the position 1 step to right such that subsequent 100 records are considered
        print("{} day input {}".format(i,x_input))
        x_input=x_input.reshape(1,-1)##Remembe to reshape for LSTM to predict
        x_input = x_input.reshape((1, n_steps, 1))
        yhat = model.predict(x_input, verbose=0)
        print("{} day output {}".format(i,yhat))
        forecast_input.extend(yhat[0].tolist())#appending the forecastinput with predicted value such that on next iteration 2:101 are considered for prediction
        forecast_input=forecast_input[1:]
        output.extend(yhat.tolist())
        i+=1
    else:    ##Systems goes inside this block during the first iteration since length of forecast_input is not greater than 100
        x_input = predict_input.reshape((1, n_steps,1)) ##Remembe to reshape for LSTM to predict
        yhat = model.predict(x_input, verbose=0)
        print(yhat[0])
        forecast_input.extend(yhat[0].tolist())#appending the forecastinput with predicted value such that on next iteration 2:101 are considered for prediction
        print(len(forecast_input))
        output.extend(yhat.tolist())##appending the output list with newly predicted values
        i+=1
print(output)
```

Forecasted values and Forecast Plot:

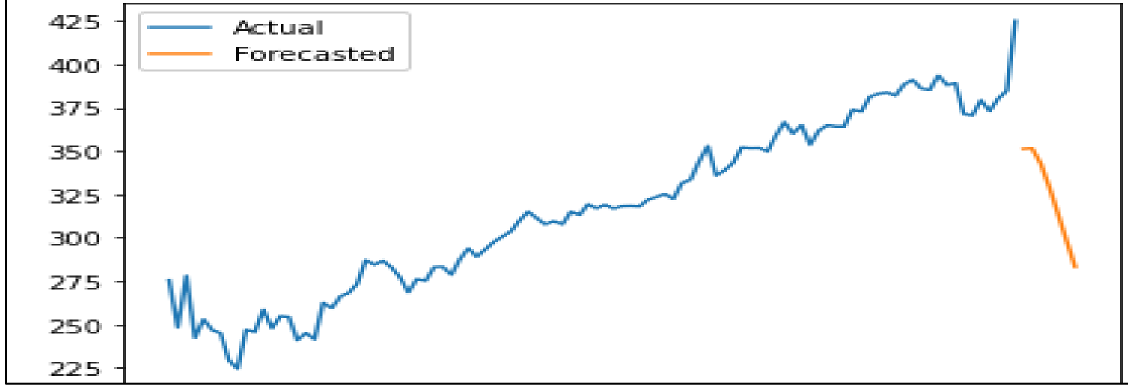
```
day_new=np.arange(1,101)
day_pred=np.arange(101,108)
len(df1)
scaler.inverse_transform(output)
```

```
array([[351.11241101],
       [351.42400498],
       [342.65051713],
       [329.10650349],
       [313.76103424],
       [298.17219891],
       [283.06444997]])
```

Forecast value and prediction accuracy could be scrutinized by trying with different values of Timesteps and Number of Epochs

```
plt.plot(day_new,scaler.inverse_transform(df1[1157:]),label='Actual')
plt.plot(day_pred,scaler.inverse_transform(output),label='Forecasted')
plt.legend(framealpha=1,frameon=True)
```

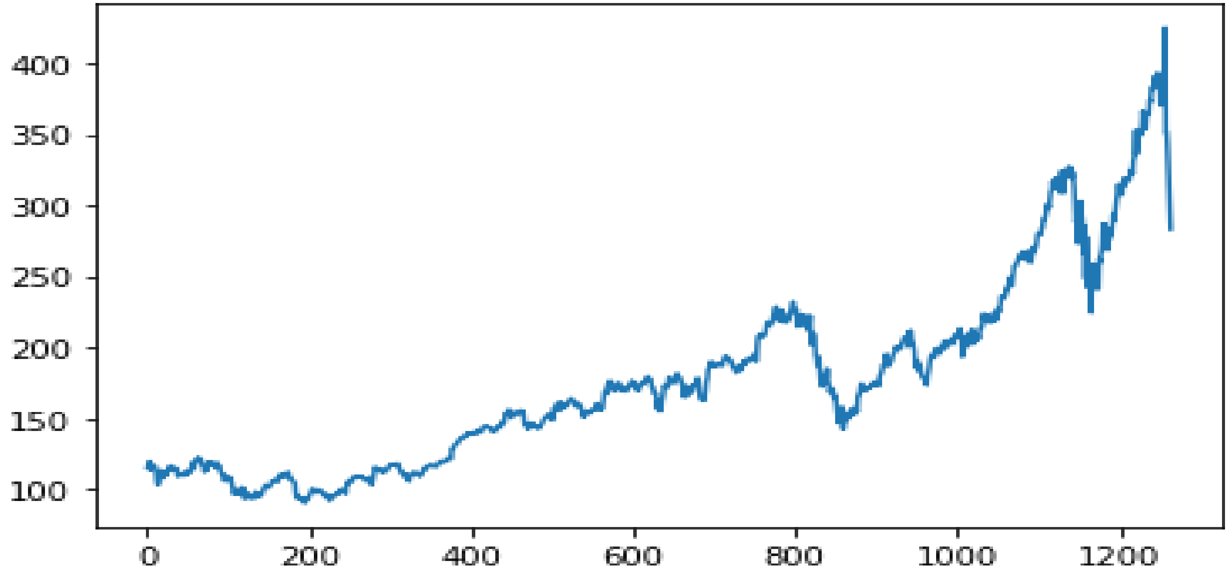
```
<matplotlib.legend.Legend at 0x7fbad8e52c50>
```



Combining both Forecasted and Original Plot:

```
df3=df1.tolist()
df3.extend(output)
df3=scaler.inverse_transform(df3).tolist()
plt.plot(df3)
```

[<matplotlib.lines.Line2D at 0x7fbad91cf710>]



Conclusion:

- To increase our accuracy we can increase the time step values as well as the number of epochs.
- Other possible options would be to add and try different hidden layers within the LSTM
- Other methods to try are ARMA and ARIMA.
- Autoregressive Moving Average (ARMA) In the statistical analysis of time series, autoregressive–moving-average models provide a close description of a stationary stochastic process in terms of two polynomials, one for the autoregression and the second for the moving average.
- Autoregressive Integrated Moving Average (ARIMA) in time series analysis, an autoregressive integrated moving average model is a generalization of an autoregressive moving average model. Both of these models are fitted to time series data either to better understand the data or to predict future points in the time series model.

Links:

- https://github.com/slydg/Stock-Quantamental-Investing-Analysis
- https://towardsdatascience.com/using-deep-learning-ai-to-predict-the-stock-market-9399cf15a312
- https://www.sciencedirect.com/science/article/pii/S1877050918307828
- https://medium.com/mlreview/a-simple-deep-learning-model-for-stock-price-prediction-using-tensorflow-30505541d877
- https://github.com/vansh123321/Projects/blob/master/Time_Series.ipynb