Google_Stock_Forecasting

January 26, 2021

1 Introduction

This project aims to forecast Google stock price. The data is achieved from Kaggle (https://www.kaggle.com/rahulsah06/gooogle-stock-price?select=Google_Stock_Price_Test.csv). The art of forecasting stock prices has been a difficult task for many years. For a good and successful investment, many investors are keen on knowing the future situation of the stock market. Good and effective prediction systems for the stock market help traders, investors, and analyst by providing supportive information like the future direction of the stock market. In this work, we present a recurrent neural network (RNN) and Long Short-Term Memory (LSTM) approach to predict stock market indices.

1.1 Content

- Data Understanding and exploration
- Preprocessing
- Data Modelling and Forecasting
  - Create RNN Model
  - Create RNN Model
- Analysis
- Conclusion and Future Insights

2 Data Understanding and exploration

The data comes with two separate file for train and test data set. So as a first step we read these files and first few rows of these datas are shown.

[4]:

	Date	Open	High	Low	Close	Volume
0	1/3/2017	778.81	789.63	775.80	786.14	1,657,300
1	1/4/2017	788.36	791.34	783.16	786.90	1,073,000
2	1/5/2017	786.08	794.48	785.02	794.02	1,335,200
3	1/6/2017	795.26	807.90	792.20	806.15	1,640,200
4	1/9/2017	806.40	809.97	802.83	806.65	1,272,400

[5]:

	Date	Open	High	Low	Close	Volume
0	1/3/2012	325.25	332.83	324.97	663.59	7,380,500
1	1/4/2012	331.27	333.87	329.08	666.45	5,749,400
2	1/5/2012	329.83	330.75	326.89	657.21	6,590,300
3	1/6/2012	328.34	328.77	323.68	648.24	5,405,900

```
4  1/9/2012  322.04  322.29  309.46  620.76  11,688,800
```

Data have total of 6 columns Date,Open,High,Low,Close,Volume. In stock trading, the high and low refer to the maximum and minimum prices in a given time period. Open and close are the prices at which a stock began and ended trading in the same period. Volume is the total amount of trading activity.

Valuable information can be gleaned from understanding the open, high, low, close of a stock, and as well as it's trading volume. Traders often visualize price actions through bars and bar charts.

```
Shape of the test data is: (20, 6)
```

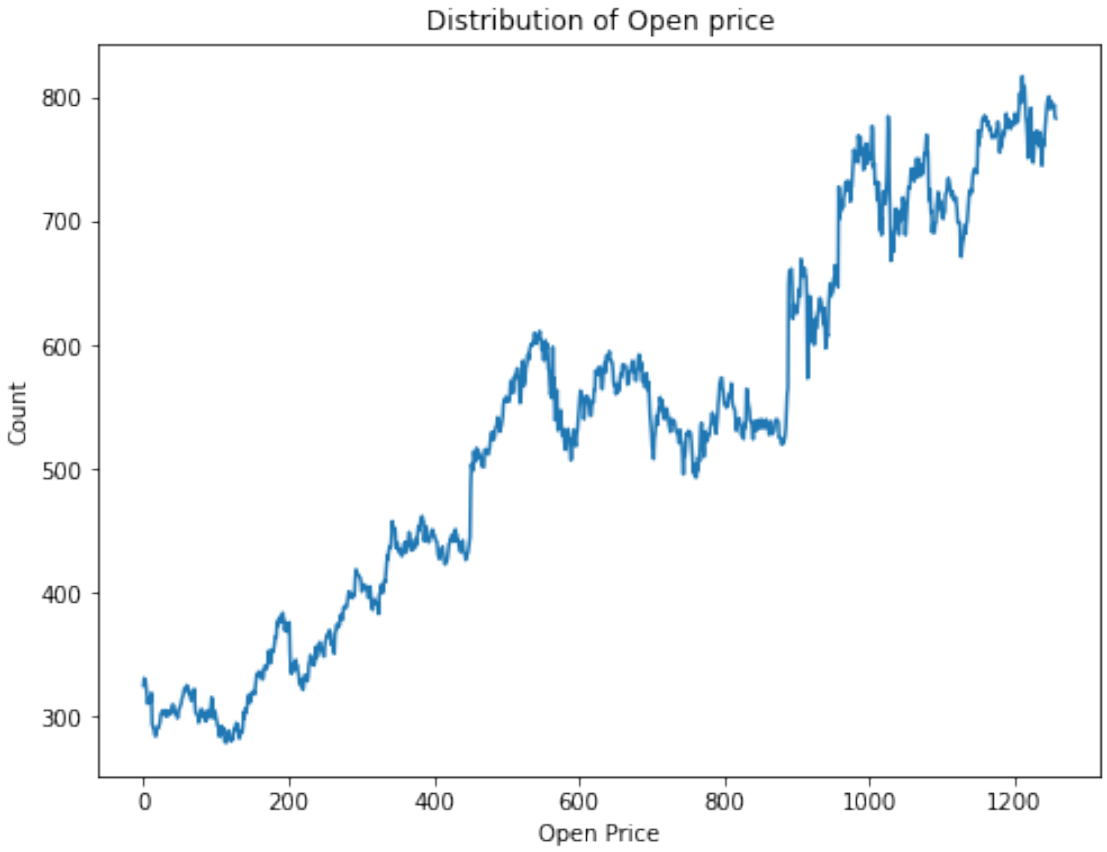
```
Shape of the test data is: (1258, 6)
```

The train data have total of 1258 rows and test have 20 rows. So the data is not that of big. Since the data have more than 1000 data we can use deep learning techniques for the forecasting.

```
|        |   0 |
|:-------|----:|
| Date   |   0 |
| Open   |   0 |
| High   |   0 |
| Low    |   0 |
| Close  |   0 |
| Volume |   0 |
```

We are interested in attempting to forecast the 'Open' series, which are prices at which a stock began. Since we don't have any missing values in the data, we do not need to interpolate it.

The distributuion of open values of the stocks is visualized below:



The distribution of open price shows that, the stocks with high open price are high in number and lower open price stocks are evry low.

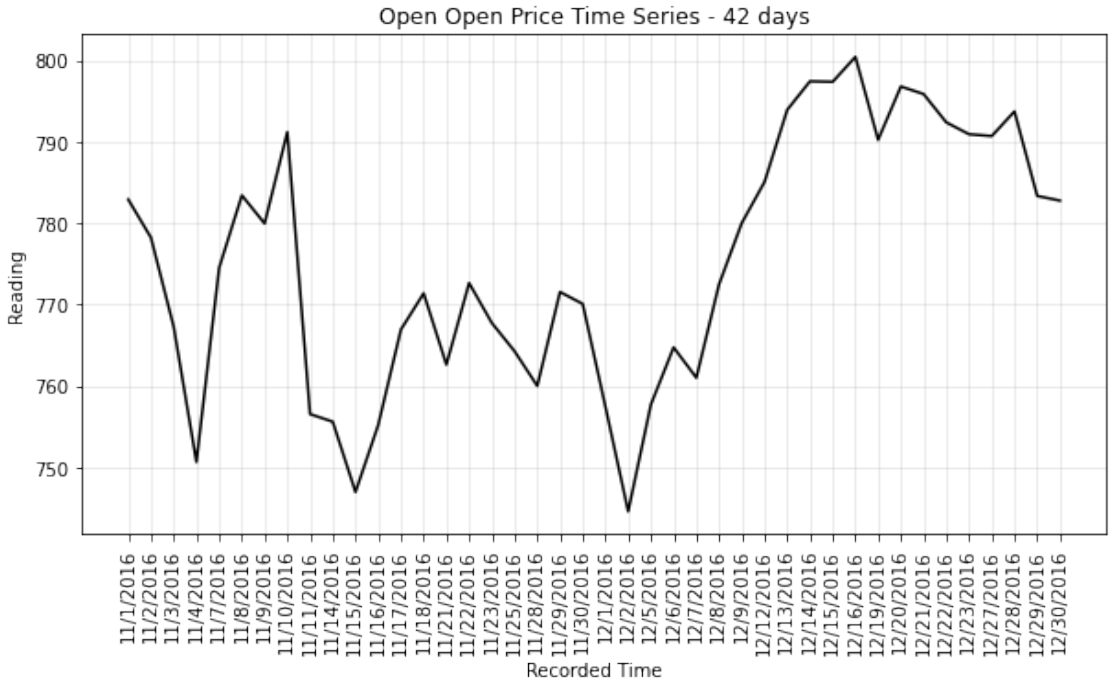
Since we are interested in the time series of the data we changed the index of the dataframe to Date column. Now the first few rows of the data is given below:

[47]:

Date	Open	High	Low	Close	Volume
1/3/2012	325.25	332.83	324.97	663.59	7,380,500
1/4/2012	331.27	333.87	329.08	666.45	5,749,400
1/5/2012	329.83	330.75	326.89	657.21	6,590,300
1/6/2012	328.34	328.77	323.68	648.24	5,405,900
1/9/2012	322.04	322.29	309.46	620.76	11,688,800

The time series changes of the Open price is given below:

So the open price has a trend as time proceeds, The proices tends to increase with short variations.



It is appear to be a periodic component present in the data.

3 Preprocessing

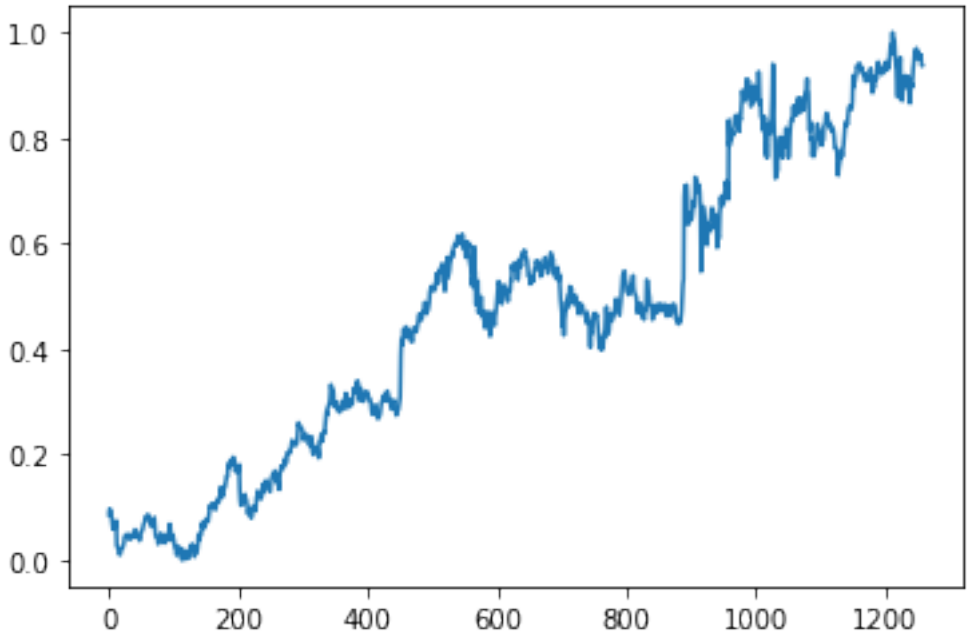
The statistical summary of the open price column is given below:

[64]:
```
count    1258.000000
mean      533.709833
std       151.904442
```

```
min        279.120000
25%        404.115000
50%        537.470000
75%        654.922500
max        816.680000
Name: Open, dtype: float64
```

We will redefine the train set to include only data and open price, since we need to forecast the opern price only. We need to scale this column for the ease of deplying deep learning models. Here we will apply the minimax scaling to the column.

Scaled the column to minimum of 0 to maximum of 1 and the distribution after the scaling is,



Before we can train a neural network with keras, we need to process the data into a format that the library accepts. In particular, for keras RNNs and LSTMs, training samples should be stored in a 3D numpy array of shape **(n_samples, time_steps, n_features)**. Since we'll be using only the series' Open Price' to predict its future, we'll only have 1 feature. Also, for the next-step prediction that we'll do in this notebook, target values can be stored in a simple list.

Here we uses all the values in the xtrain for the prediction with 100 timesteps, and defined y as just the next time step after training sample.

```
Training input shape: (1158, 100, 1)
Training output shape: (1158,)
```

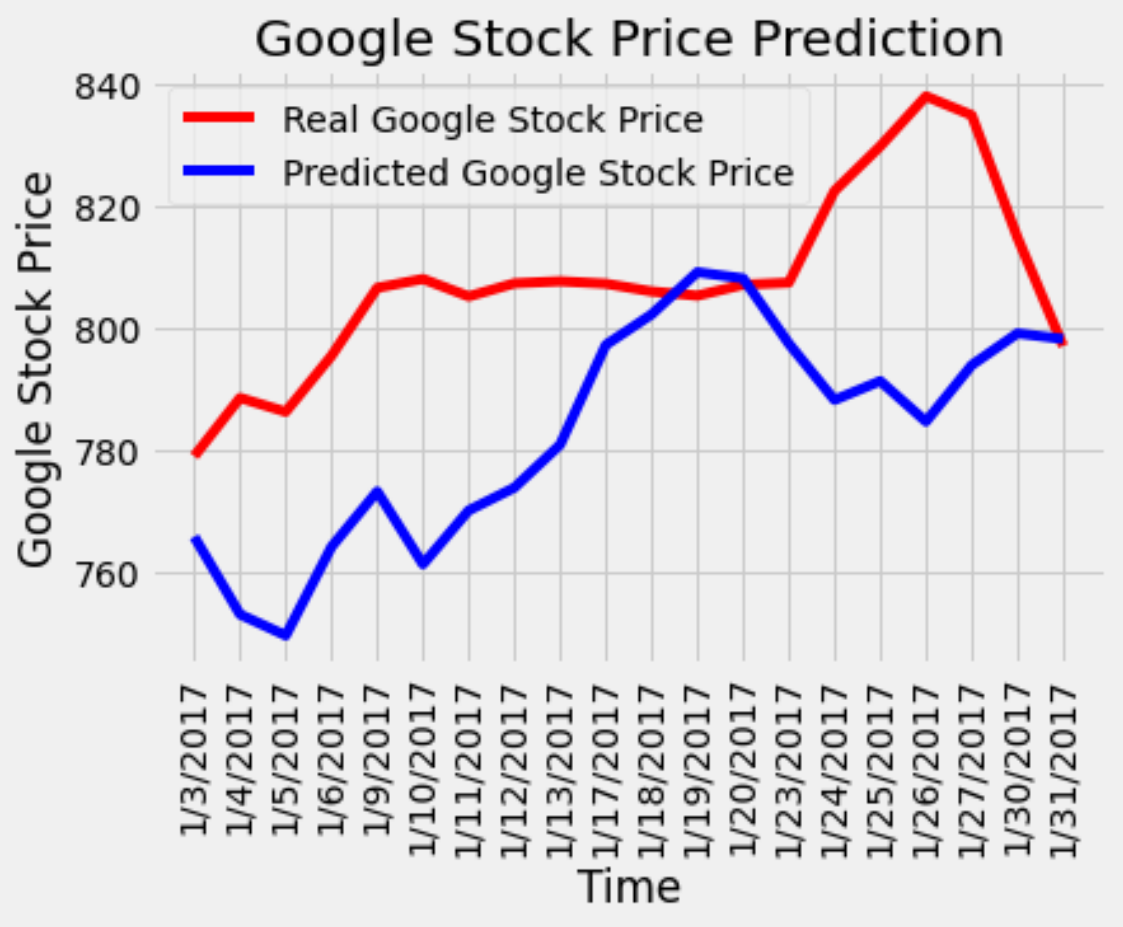
Like train set, test set also preprocessed to array of length 20 with 100 timesteps and 1 feauture.

```
Test input shape: (20, 100, 1)
Test output shape: (20,)
```

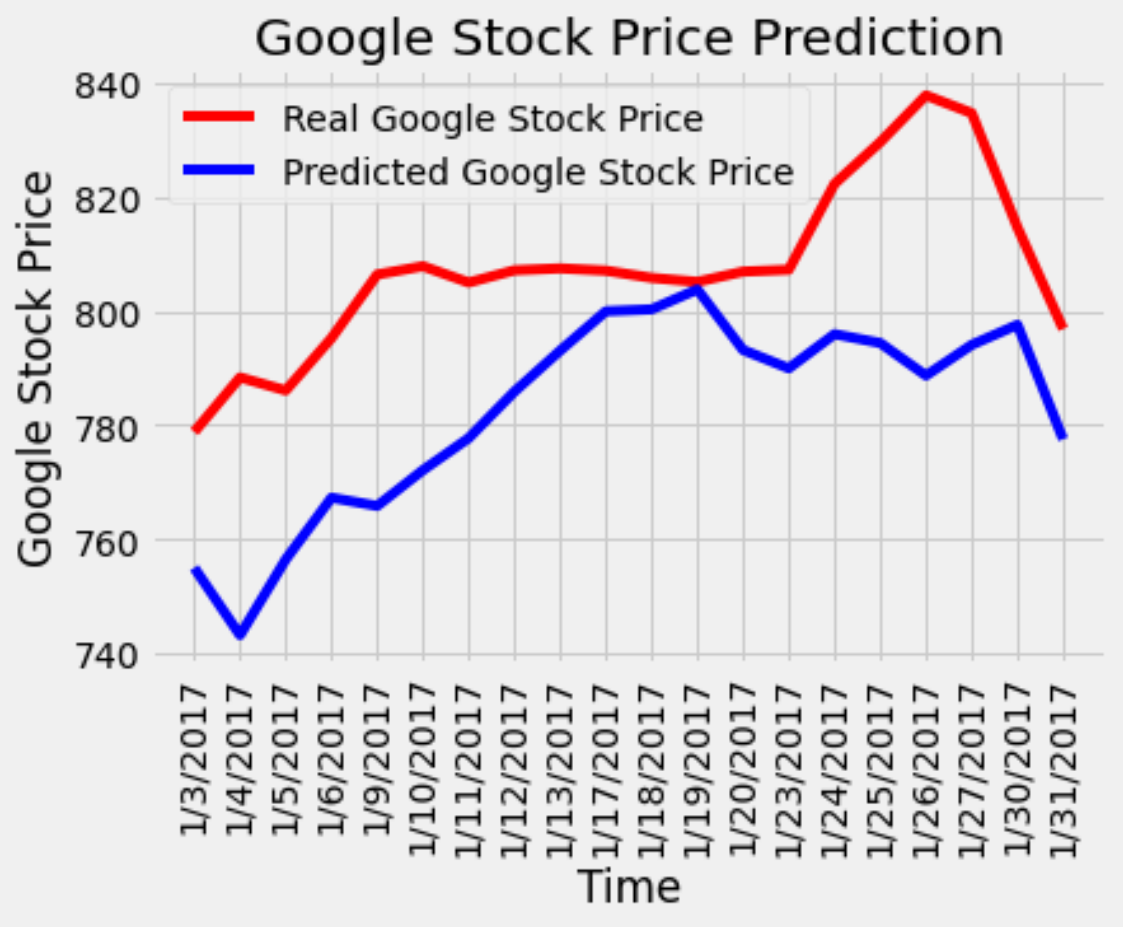
4 DataModelling and Forecasting

Now are data is perfect for modelling. First we will start with a simple RNN and evaluate the performance. Then move to more complex model and tune the hyperparameter for better performance. Finally deployes the LSTM model and carries out the performance evaluation.

4.1 RNN

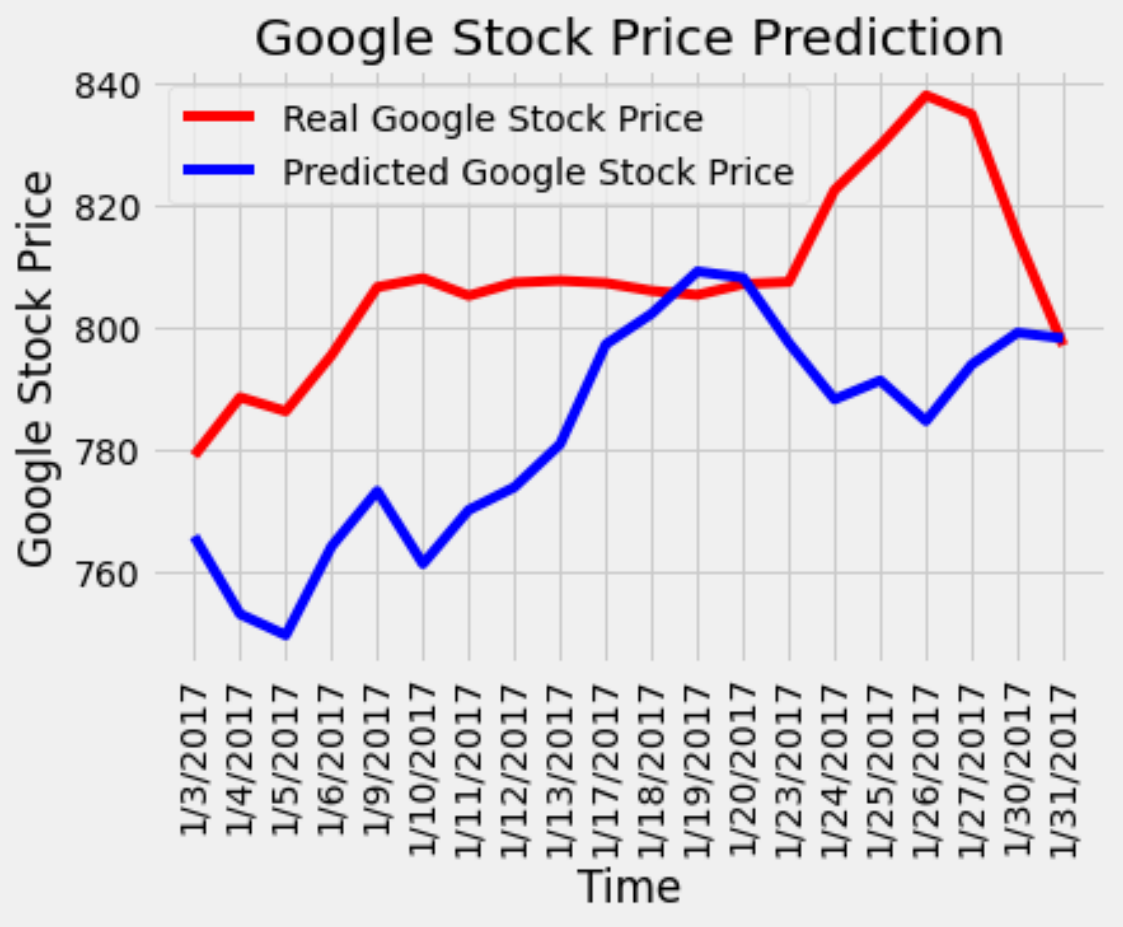



```
Root mean squared error is 613095.2730573008
```




```
Root mean squared error is 612606.5554208021

Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn_3 (SimpleRNN)     (None, 100)               10200
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 101
=================================================================
Total params: 10,301
Trainable params: 10,301
Non-trainable params: 0
_________________________________________________________________
```



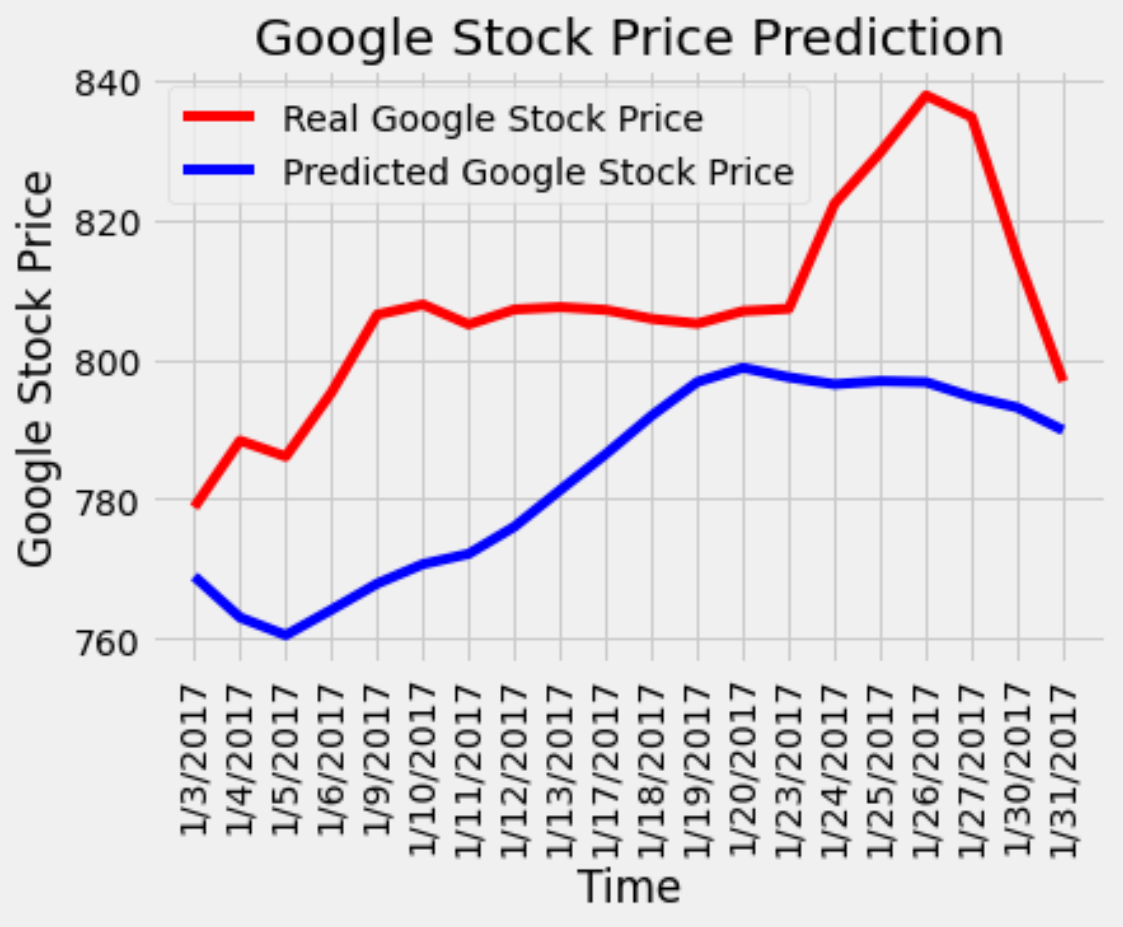

```
Root mean squared error is 613095.2730573008
```

The above plot visualize the distribution of Google stock price for both original stock value and the predicted stok value using RNN. The predicted values seems to be not far away from original values and attained a Mean squared error of '613095.27'.

Now we will try whether we can improve this forecast prediction further more using more complex RNN, whcih have 3 hidden layers.

So first we deployes a RNN with 50 units, an activation of 50 for an input layer and three hidden layer.

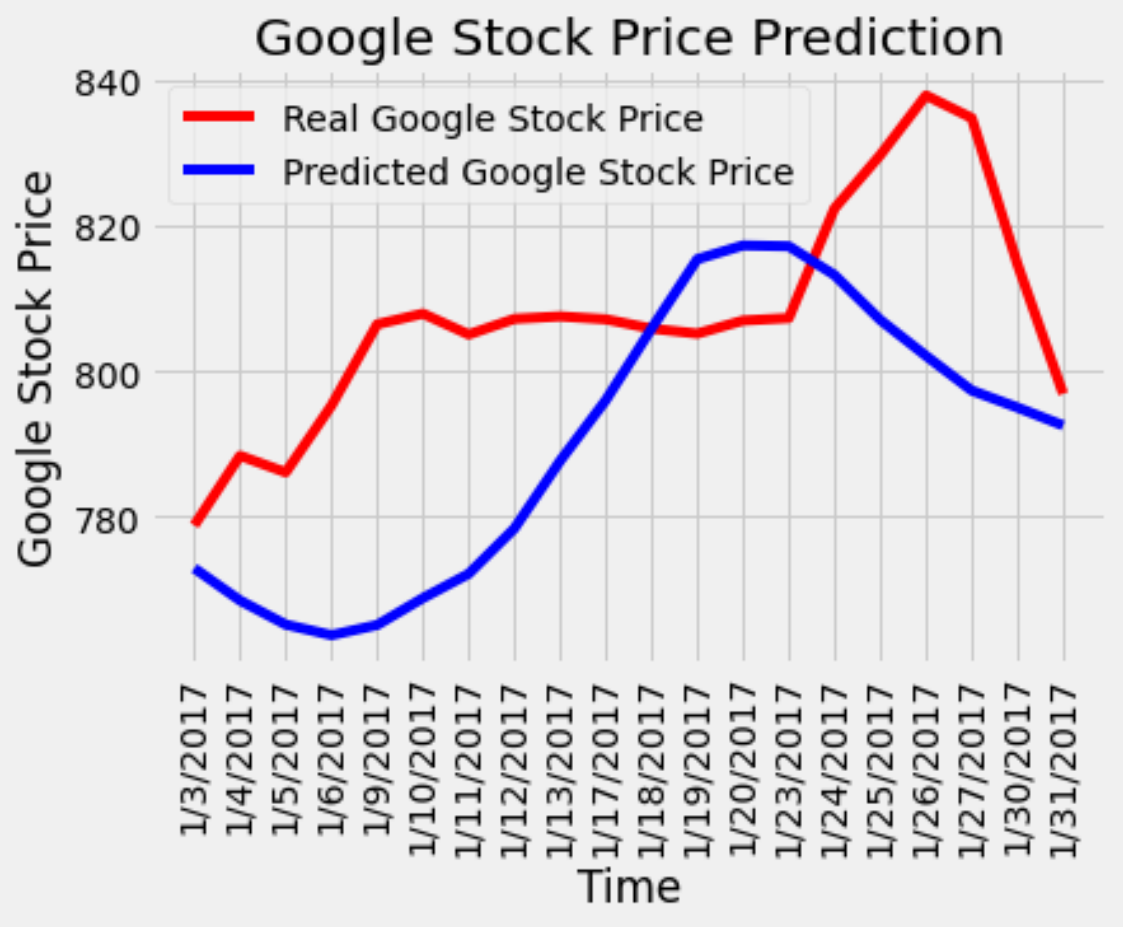
The plot of original and predicted values given below:




```
Mean Squared error is 613414.781542415
```

The above plot shows not much difference from the results of previous model and achieved an MSE of 613414.78.

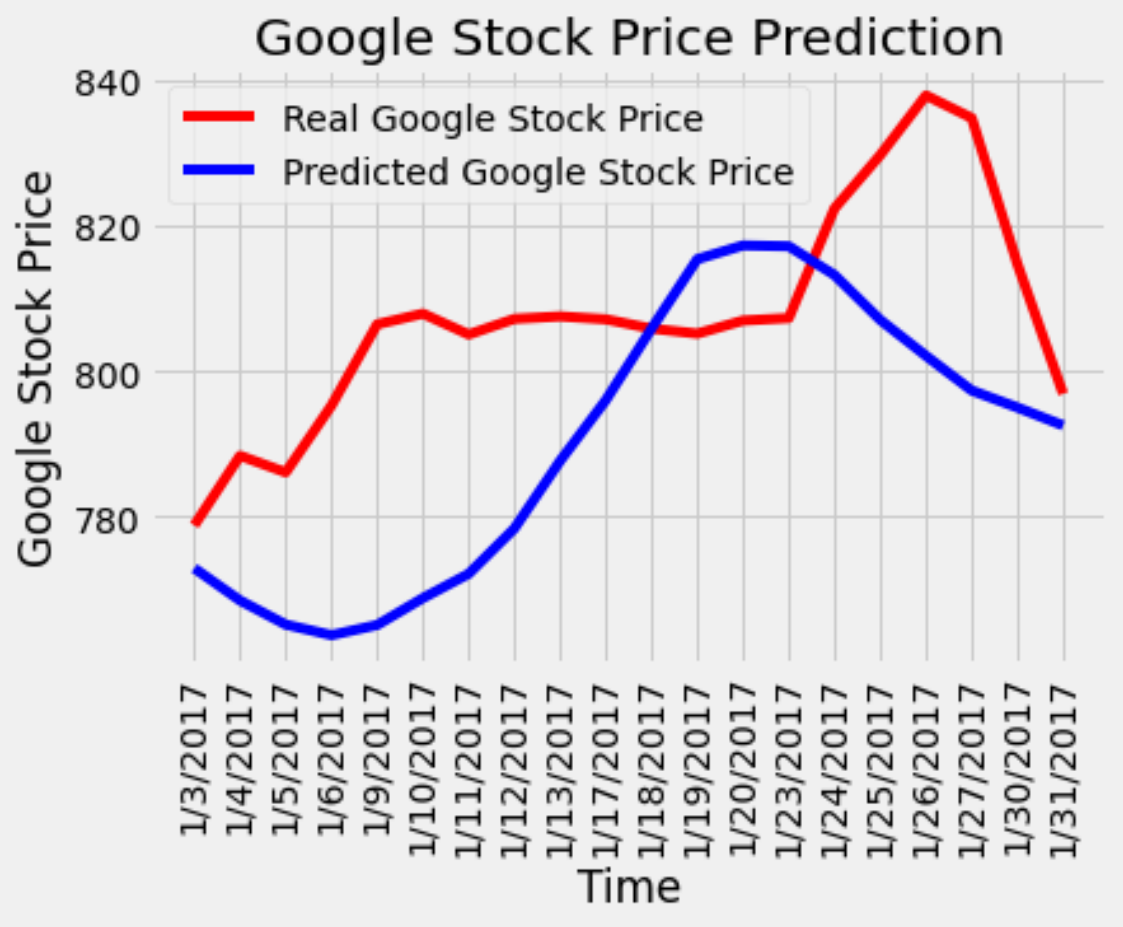
To inrease the accuracy of prediction, now we will increase the epoch of the above model from 100 to 200 and checks the



`Mean Square value with 3 hidden layer and 50 number of cell is 624400.442683162`

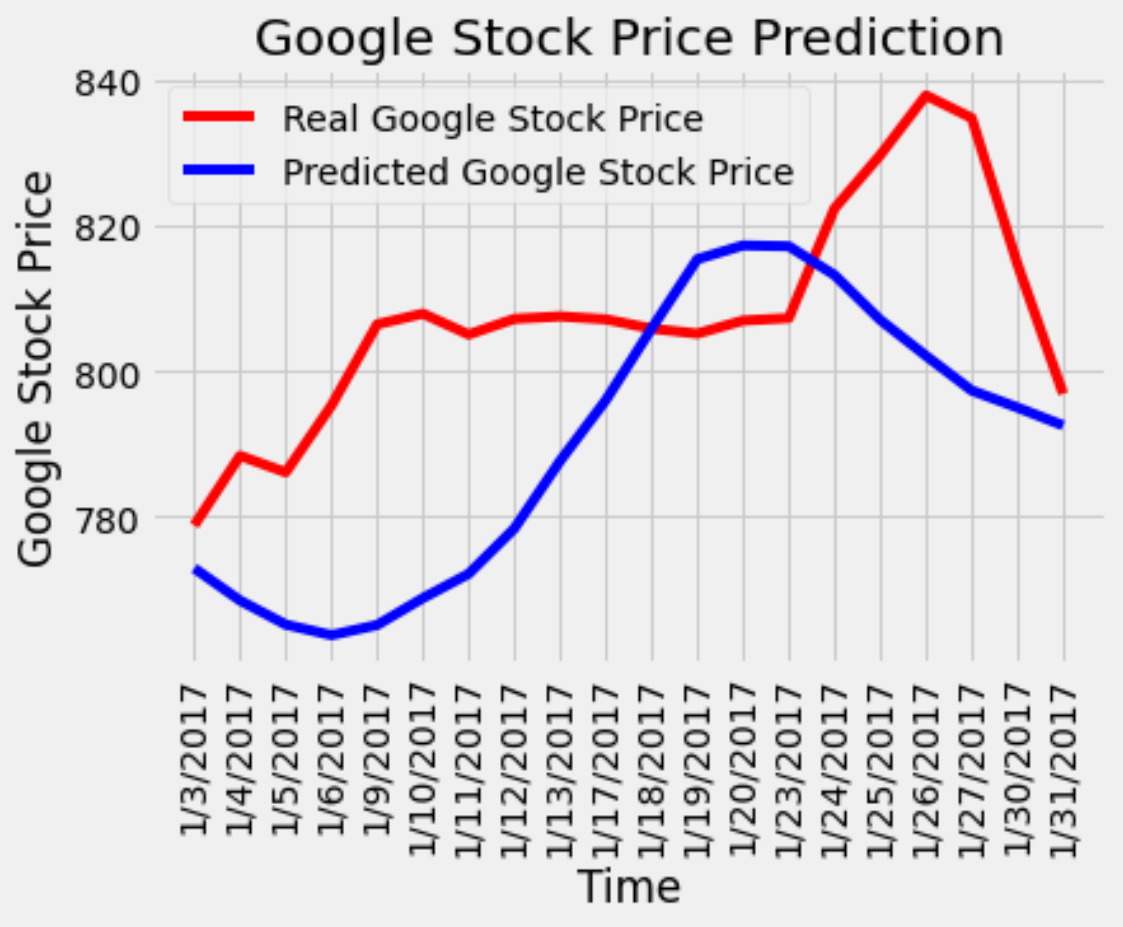
When increased the number of epochs , the model seems to have overfitted. The small dips and peaks of the predictions are missing in the new prediction. So it is not a good idea to increase the epoch to 200.

Now we will check changing the activation function to 'relu' would effect the forecasting performance.



`Mean Square value with 3 hidden layer and 50 number of cell is 623029.1496236795`

The above figure of the prediciton shows forecasting with 'relu' activation function worsened the prediction. So now we will try the activation function of 'elu'.



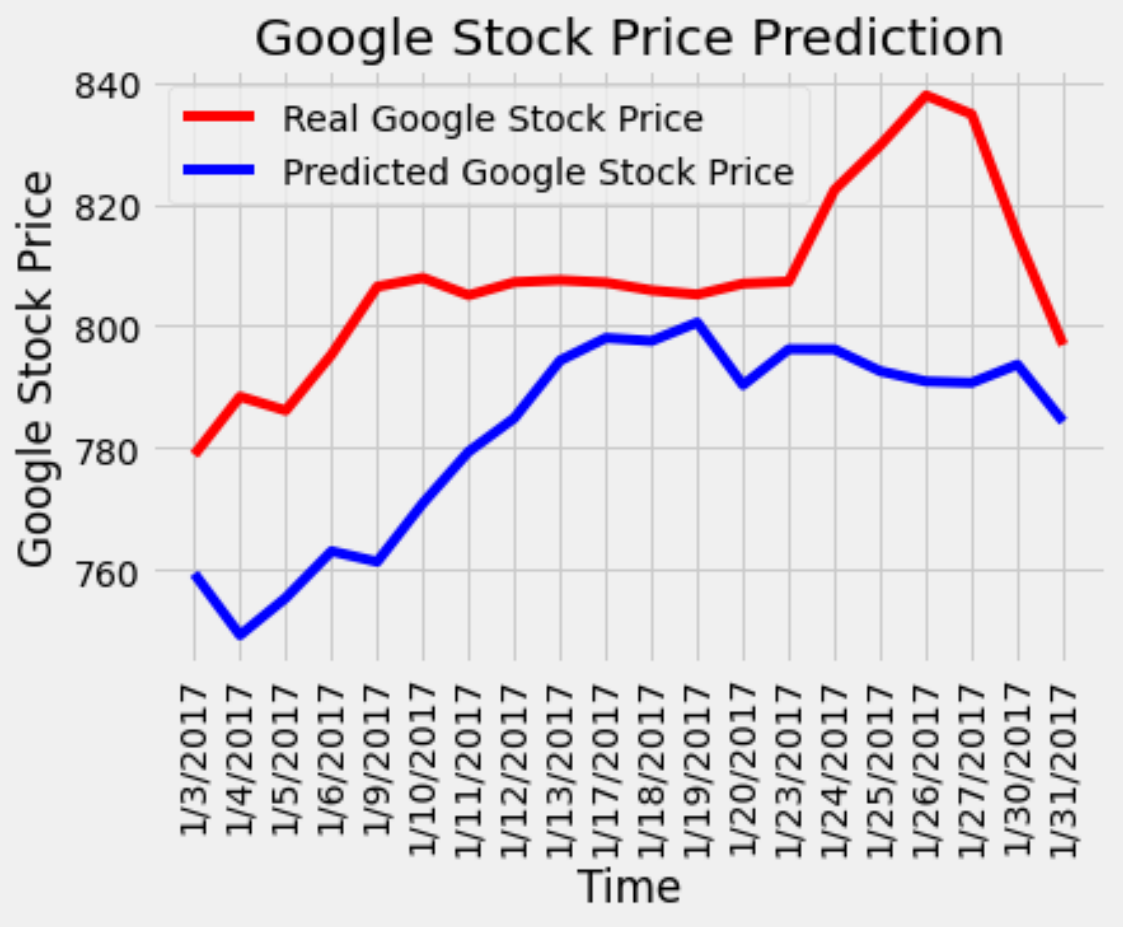

```
Mean Square value with 3 hidden layer and 50 number of cell is:
617661.3547055001
```

The prediciton using 'elu' seems to have done a decent job compared to 'relu'. But the performance is not better from model using 'tanh'.

4.2 LSTM

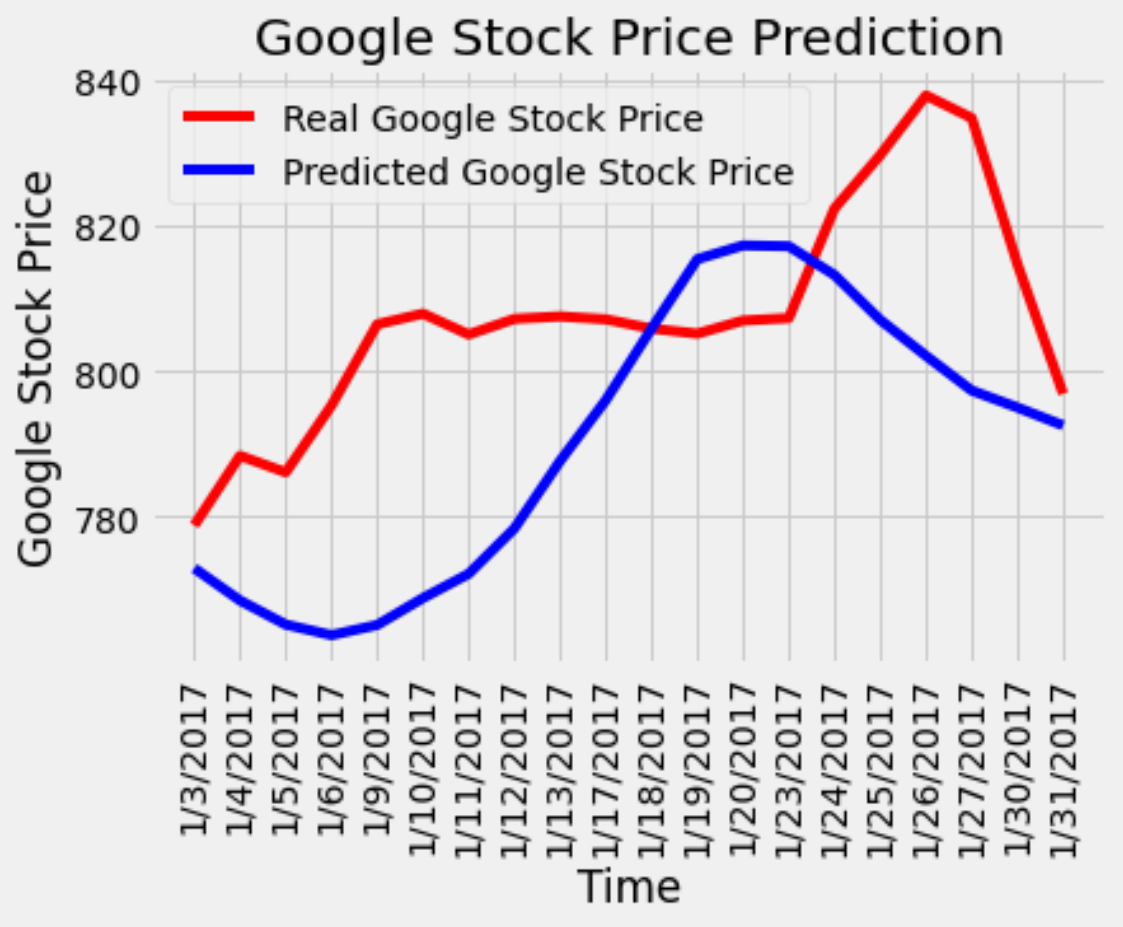
So far we have tuned hyperparameter of RNN and optimized the forecast prediction based on the Mean square error of the prediciton. Now we will deploy the LSTM model to our data. Since LSTM requires lot of computational power we will limit the tuning to just two steps and analyses the results.

Starts with a simple network without any hidden layers and 10 unit of cells and epoch number of 50 and batch size =1.



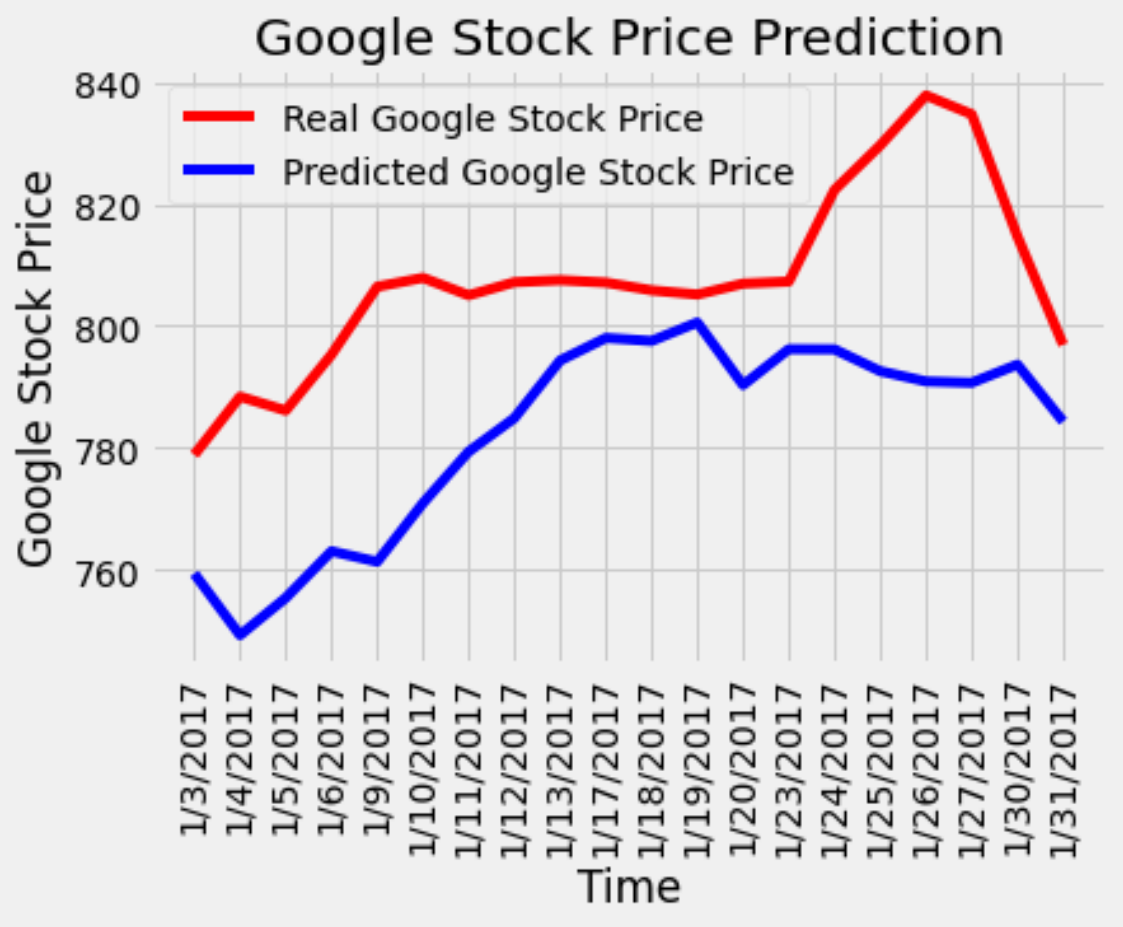
As expected LSTM gave a better forecasting than the RSTM model. The new MSE is 612266.340 and the plot shows that predicted values are in good resemblence with the original value.

Now we increase the epoch and check whether it increases the precsion of prediction.



The MSE didnt get better compared to the previous model and the model seems to be overfitted from the plot. The MSE is 612825.626.

To increase the prediction better, added three hidden layer to the system




```
Mean squred error is: 612266.3402352287
```

Even though we added some more hidden layer to it, the MSE of the prediction didnt improve. It is same as the previous LSTM Model. Also this model needed high computational power.

5 Analysis

The table showing the Meamn squared errror of all the models, we have deployed is given below:

[259]:

	Model	MSE
0	Simple_RNN_100_epoch_100cell	612606.5500
1	Simple_RNN_50_epoch_100cell	613095.2730
2	RNN_3_layer_100epoch	613414.7800
3	RNN_3_layer_200epoch	624400.4426
4	RNN_3_layer_100epoch_relu	623029.1496
5	RNN_3_layer_100epoch_elu	617661.3540
6	LSTM 50 epoch	612266.3400
7	LSTM 100 epoch	612825.6260
8	LSTM 3 hidden layer 100 epoch	612266.3400

The Mean squared error table above shows the brief information of the performance of the models. Surprisingly the Simple RNN model without any hidden layer has better performance than RNN with 3 hidden layers. The minimum Mean squared error achieved was from the model with 100 cells and 100 epochs.

The LSTM model forecasted the model more precisely compared to RNN, But the computational it need has to taken into account. so the output of these models is not worth its huge computational model. A simple RNN model with minimum computational need provides almost the same prediction result as LSTM.

So considering both the predicting accuracy and Computational need, I would choose a simple RNN model with 100 cell numbers and 100 epochs. Even though this model consumed little computational power, it could provide a result nearly good as LSTM. The huge computational expense of LSTM is a drawback of this model if one doesn't have GPU for the computation.

6 Conclusion and Future Insights

This project aimed to forecast the stock price of Google. For the forecasting, we deployed RNN and LSTM with different hyperparameter tuning. The RNN model with 100 cell numbers and 100 epochs and LSTM model predicted with better accuracy. Since LSTM needs more computational power, a simple RNN is most suitable for future predictions.

We could further tune cell number, epoch, and batch size to check for better performance. Also, we can forecast the close price, low price, and stock price of the stock this way.