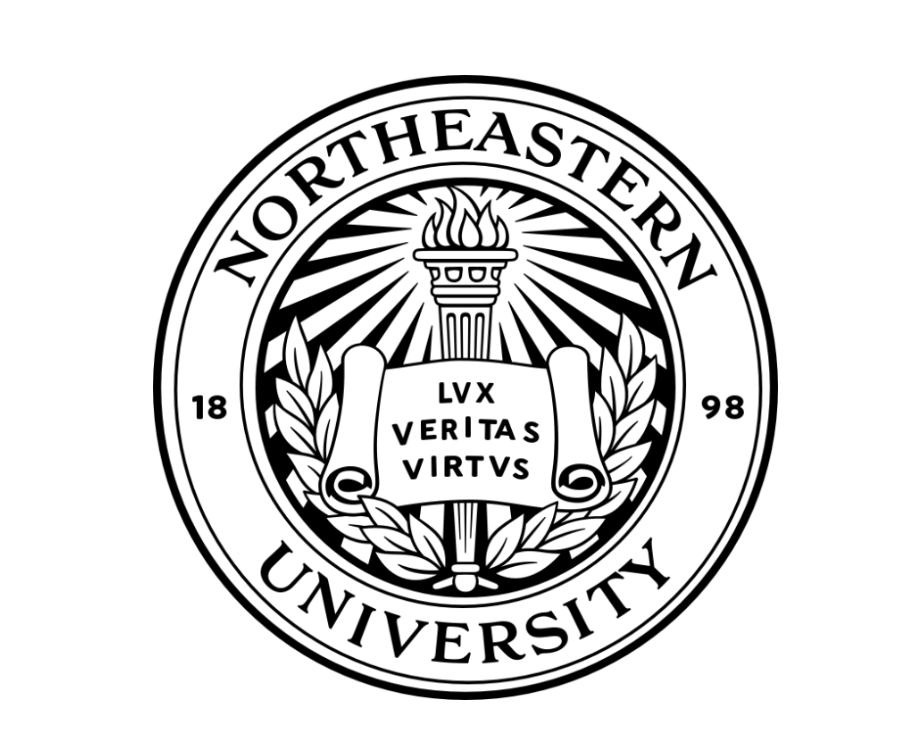
**Stock Market Analysis and Prediction**



**Group 11**

**Prathamesh Limaye**

**Prathamesh Verlekar**

**Saurabh Satra**

**Yash Pandya**

**Data Science Engineering Methods and Tools**

**Professor Handan Liu**

**Table of Contents:**

**1.0 Introduction**

**2.0 Objective**

**3.0 Methodology**

**4.0 Algorithms Used**

**5.0 Description of Data set**

**6.0 Data Set Link**

**7.0 Implementation**

**8.0 Conclusion**

**9.0 Future Scope**

**10.0 References**

1. **Introduction:**

Financial markets are one of the most fascinating inventions of our time. They have had a significant impact on many areas like business, education, jobs, technology and thus on the economy. Over the years, Stock market prediction has always caught the attention of many analysts and researchers. Predicting stock prices is a difficult problem in itself because of the number of variables which are involved. Therefore, investors and researchers have been interested in developing and testing models of stock price behavior. However, analyzing stock market movements and price behaviors is extremely challenging because of the markets dynamic, nonlinear, nonstationary, nonparametric, noisy, and chaotic nature. In short, stock markets are affected by many highly interrelated factors that include economic, political, psychological, and company-specific variables. Application of machine learning techniques and other algorithms for stock price analysis and forecasting is an area that shows great promise.

1. **Objective:**

In the past decades, there is an increasing interest in predicting markets among economists, policymakers, academics and market makers. Investment firms, hedge funds and even individuals have been using financial models to better understand market behavior and make profitable investments and trades. A wealth of information is available in the form of historical stock prices and company performance data, suitable for machine learning algorithms to process. Can we predict stock prices with machine learning? Investors make educated guesses by analyzing data. They'll read the news, study the company history, industry trends and other lots of data points that go into making a prediction.

The objective of the proposed project is to study, analyze and improve the supervised machine learning algorithms to accurately predict the future closing value of a given stock across a given period in the future.

1. **Methodology:**

Recent advancements in stock analysis and prediction fall under four categories—statistical, pattern recognition, machine learning (ML), and sentiment analysis. These categories mostly fall under the broader category of technical analysis, however, there are some machine learning techniques which also combine the broader categories of technical analysis with fundamental analysis approaches to predict the stock markets.



Before the advent of machine learning techniques, statistical techniques which often assumes linearity, stationarity, and normality provided a way to analyze and predict stocks. Time series in stock market analysis is a chronological collection of observations such as daily sales totals and prices of stocks. one group of statistical approaches which fall into the category of univariate analysis, due to their use of time series as input variables, are the Auto-Regressive Moving Average (ARMA), the Auto-Regressive Integrated Moving Average (ARIMA).

Machine learning has been extensively studied for its potentials in the prediction of financial markets. Machine learning tasks are broadly classified into supervised and unsupervised learning. In supervised learning, a set of labelled input data for training the algorithm and observed output data are available. However, in unsupervised learning, only the unlabeled or observed output data is available. The goal of supervised learning is to train an algorithm to automatically map the input data to the given output data. When trained, the machine would have learned to see an input data point and predict the expected output. The goal of unsupervised learning is to train an algorithm to find a pattern, correlation, or cluster in the given dataset. Several algorithms have been used in stock price direction prediction. Simpler techniques such as the single decision tree, discriminant analysis, and naïve Bayes have been replaced by better-performing algorithms such as Random Forest, logistic regression, and neural networks. With nonlinear, data-driven, and easy-to-generalize characteristics, multivariate analysis using deep Artificial Neural Networks (ANNs) has become a dominant and popular analysis tool in the financial market analysis. Recently, deep nonlinear neural network topologies are beginning to attract attention in time series prediction.

1. **Algorithms Used:**

We have made use of the following algorithms in our project:

**1) Linear Regression**

**2)** **Autocorrelation**

**3)** **ARIMA**

**4)** **RNN**

1. **Description of Data set:**

The Dataset Titled “Huge Stock Market Dataset” contains full historical data daily price and volume data for all US-based stocks and ETFs (Exchange Traded Funds) trading on the NYSE (The New York Stock Exchange), NASDAQ (National Association of Securities Dealers Automated Quotations) and NYSE MKT.

The dataset provided on Kaggle is around 1.53 GB that is quite big .The data is presented in CSV (Comma Separated Values) Format and has features such as Date, Opening Value, High Value, Low Value, Closing Value, Volume of Stocks and OpenInt Stock Values for over 1000 Companies in the United States of America.

1. **Data Set Link:**

Huge Stock Market Dataset – **Kaggle**

<https://www.kaggle.com/borismarjanovic/price-volume-data-for-all-us-stocks-etfs#aadr.us.txt>

1. **Implementation:**

**Autoregression**

A regression model, such as linear regression, models an output value based on a linear combination of input values.

Equation for linear regression: - y = b0 + b1\*x1

Where y is the prediction, b0 and b1 are coefficients found by optimizing the model on training data, and X is an input value. This technique can be used on time series where input variables are taken as observations at previous time steps, called lag variables. For example, we can predict the value for the next time step (t+1) given the observations at the last two-time steps (t-1 and t-2).

For a regression model,

this would look as follows: X(t+1) = b0 + b1\*X(t-1) + b2\*X(t-2)

Because the regression model uses data from the same input variable at previous time steps, it is referred to as an autoregression (regression of self).

## **Autocorrelation**

An autoregression model assumes that the observations at previous time steps are useful to predict the value at the next time step. This relationship between variables is called correlation.

If both variables change in the same direction (e.g. go up together or down together), this is called a positive correlation. If the variables move in opposite directions as values change (e.g. one goes up and one goes down), then this is called negative correlation.

We can use statistical measures to calculate the correlation between the output variable and values at previous time steps at various lags. The stronger the correlation between the output variable and a specific lagged variable, the more weight that autoregression model can put on that variable when modeling.

Again, because the correlation is calculated between the variable and itself at previous time steps, it is called an autocorrelation. It is also called serial correlation because of the sequenced structure of time series data.

The correlation statistics can also help to choose which lag variables will be useful in a model and which will not. If all lag variables show low or no correlation with the output variable, then it suggests that the time series problem may not be predictable.

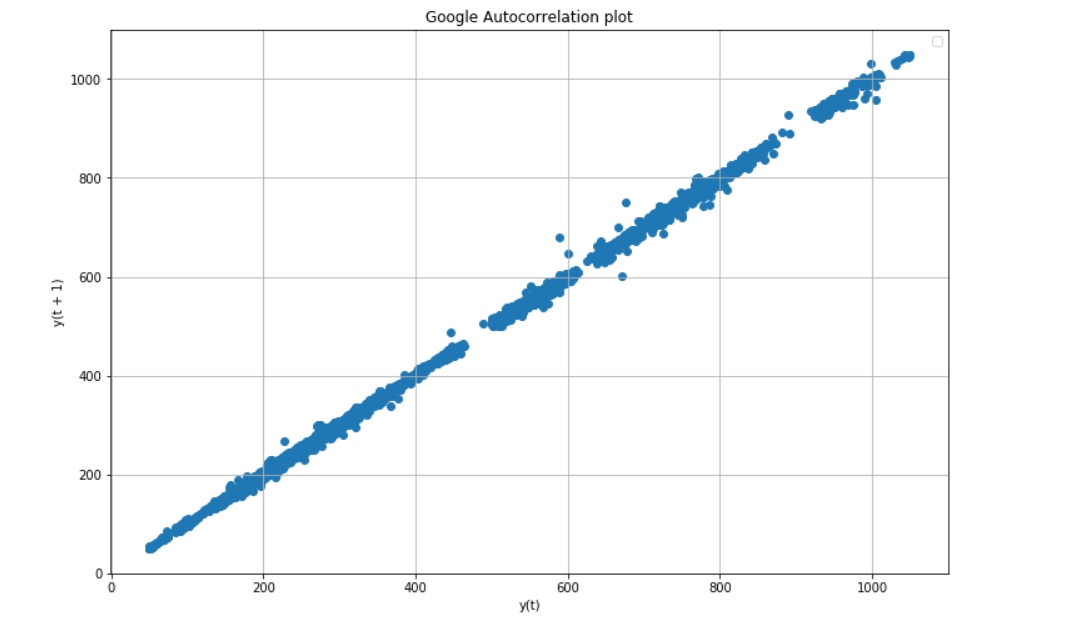
**Check for a Autocorrelation**

There are many ways by which we can check for autocorrelation in our time series dataset. We can plot the observation at the previous time step (t-1) with the observation at the next time step (t+1) as a scatter plot. This could be done manually by first creating a lag version of the time series dataset and using a built-in scatter plot function in the Pandas library. But we can plot this graph using simple built-in function provided in python library Pandas called as the lag\_plot () function.



Here we have used the statistical test for correlation called as “Pearson Correlation Coefficient Test” to demonstrate the correlation between the observation and the lag variable. This produces a number to summarize how correlated two variables are between -1 (negatively correlated) and +1 (positively correlated) with small values close to zero indicating low correlation and high values above 0.5 or below -0.5 showing high correlation. As we can see we have got the correlation values for t-1 and t+1 is above 0.5 for the Google Stock Data, shows a strong correlation (0.966) between the observation and the lag value =1.

We can plot the autocorrelation graph for the same using corr() function on the DataFrame of the lagged dataset, where we can see a large ball of observations along a diagonal line of the plot, which directly indicates the correlation between the parameters.



Here, we have plotted the Autocorrelation graph for the Google Stocks, y(t) as X-parameter while y(t+1) as a Y-parameter.

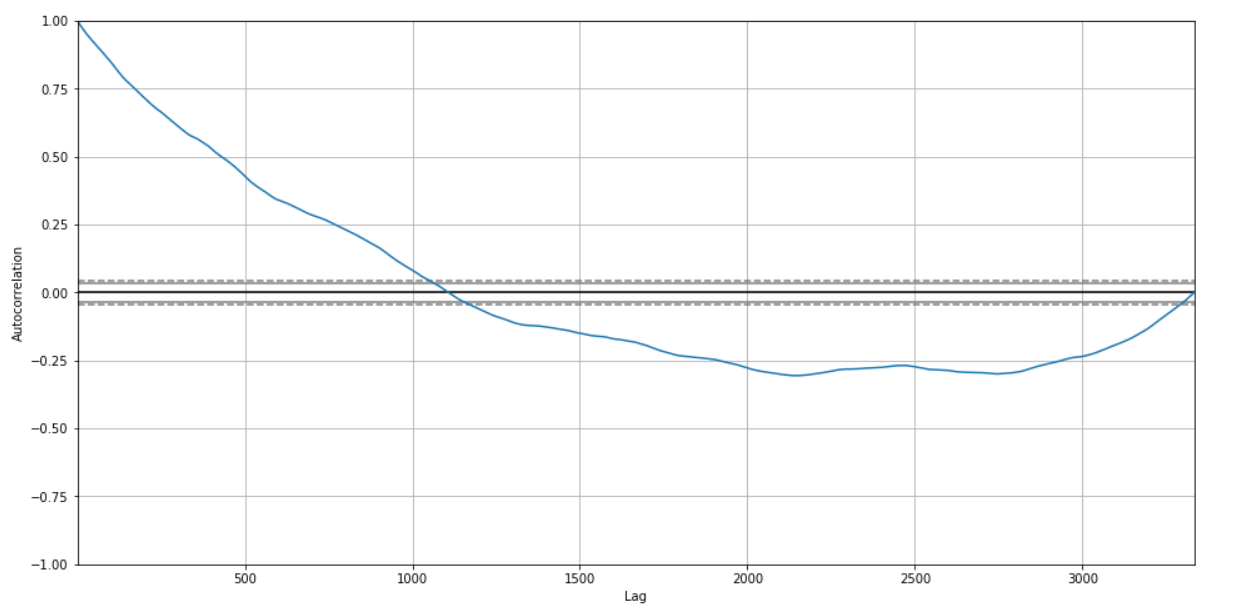
**Autocorrelation Plots**

We can plot the correlation coefficient for each lag variable. This gives an idea of which lag variables may be good candidates for use in a predictive model and how the relationship between the observation and its historic values changes over time.

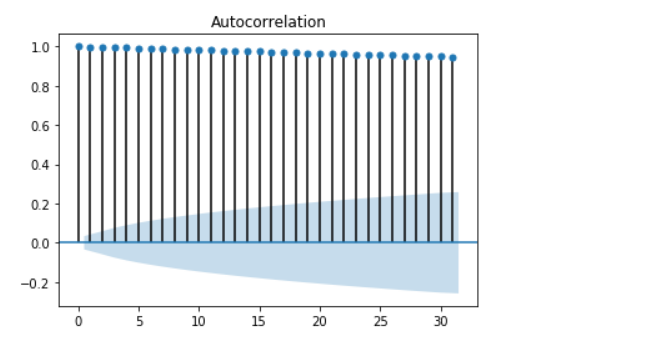
We could manually calculate the correlation values for each lag variable and plot the result. But it would be tedious job to do. There is built-in function called Autocorrelation plot () in python library- pandas.

The plot provides the lag number along the x-axis and the correlation coefficient value between -1 and 1 on the y-axis. The plot also includes solid and dashed lines that indicate the 95% and 99% confidence interval for the correlation values. Correlation values above these lines are more significant than those below the line, providing a threshold or cutoff for selecting more relevant lag values.

The graph below shows the autocorrelation graph for the Google Stock data between the observation and lag variable. By doing this, we get the swing in positive and negative correlation as the stock data values change across the years.

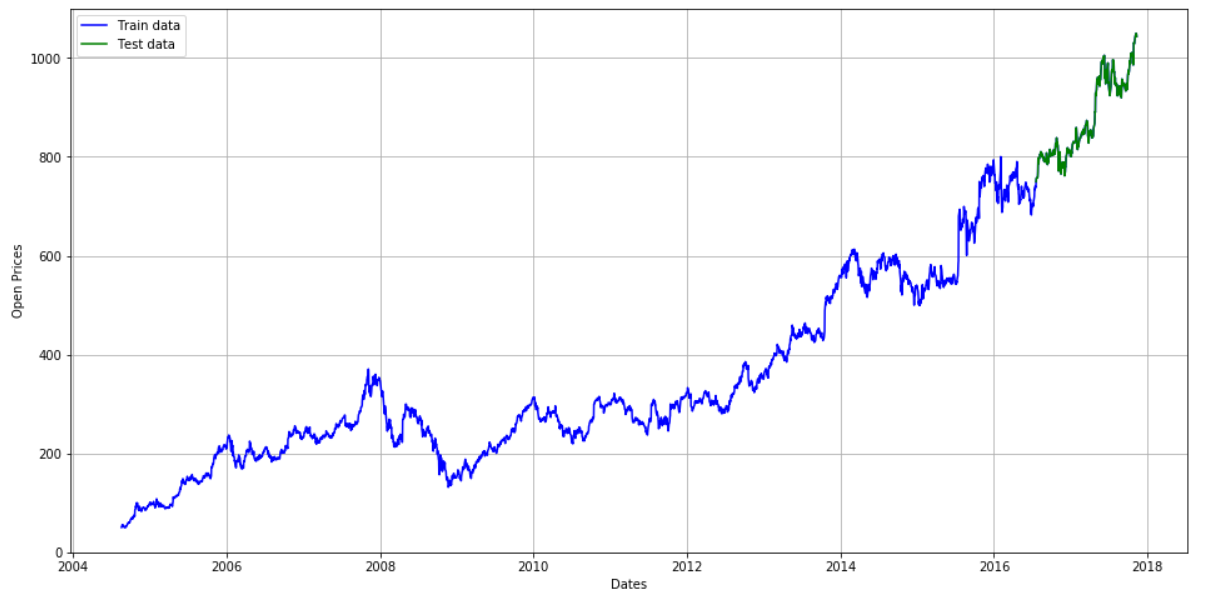


The statsmodels library in Python provides a different version of the plot in the plotacf() function as a line plot. The graph below shows evaluation of lag variables, but we have limited the lag number as 31.



**Splitting of the data**

We are splitting the data into training and prediction(testing) data, so we can compare our prediction with the actual stock market values.

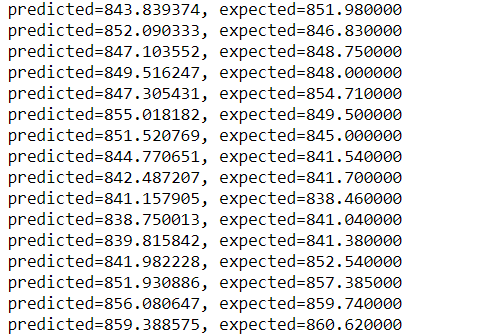


The visualization of Data vs Dates shows a splitting of the Google stock data into training and testing data. The Blue line shows the data for training and Green line data shows data for testing or prediction.

**Autoregression Model**

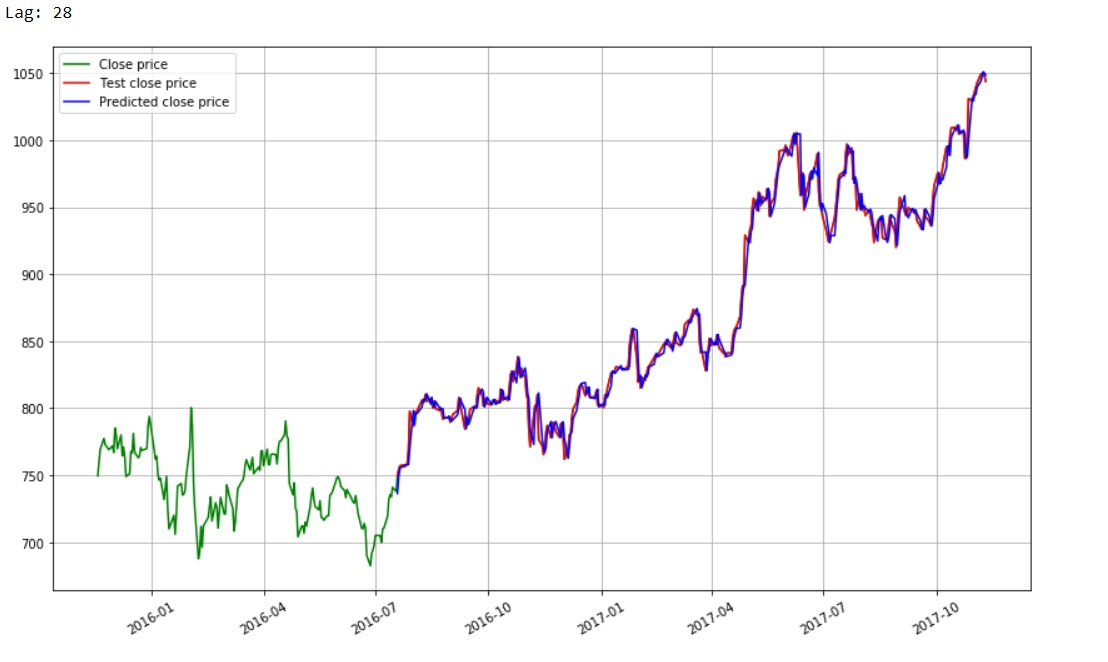
An autoregression model is a linear regression model that uses lagged variables as input variables. The statsmodels library provides an autoregression model that automatically selects an appropriate lag value using statistical tests and trains a linear regression model. It is provided in the [AR class](http://statsmodels.sourceforge.net/devel/generated/statsmodels.tsa.ar_model.AR.html). We can use this model by first creating the model AR () and then calling fit () to train it on our dataset. This returns an [AR Result](http://statsmodels.sourceforge.net/devel/generated/statsmodels.tsa.ar_model.ARResults.html) object. Once we fit the model, we can use the model to make a prediction by calling the predict () function for several observations in the future.

The following image shows the results for the Google stock data prediction using Autoregression model. The results show predicted values of the Google Stock followed by Expected values of Google Stock i.e. testing data.



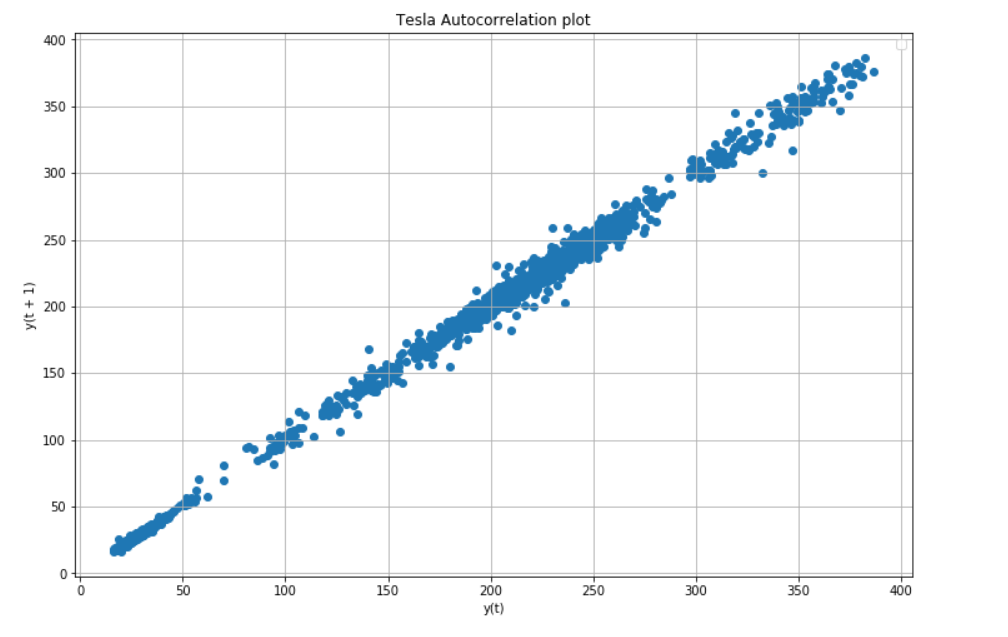
Along with the Stock prediction, we have also calculated the error values such as MSE (Mean Squared Error), RMSE (Root Mean Square Error), MAE (Mean Absolute Error). The error values we get during the Google Stock Market prediction are 78.49, 6.42, 8.85 respectively.

The following visualization shows the graph for predicted and Expected values of the Google Stock values.

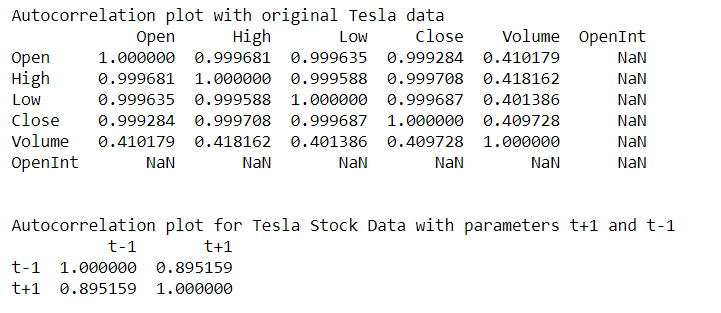


**Results of Stock Prediction for Tesla using Auto Regression Model**

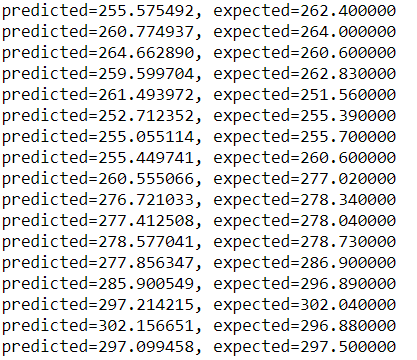
**Autocorrelation Plot**



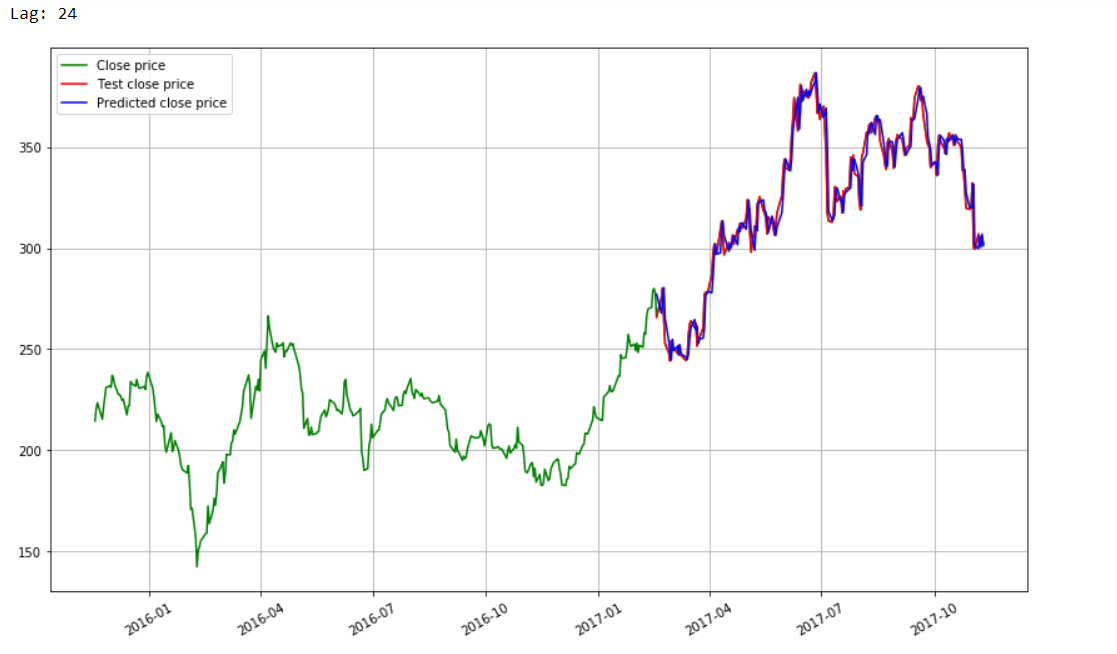
**Autocorrelation Results:**



**Prediction for Tesla Stocks:**

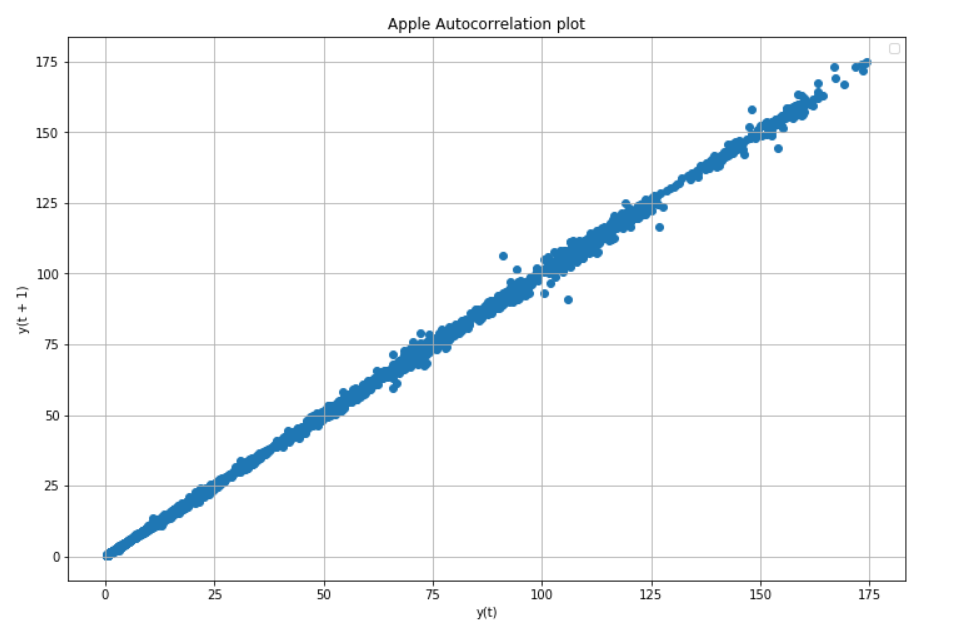


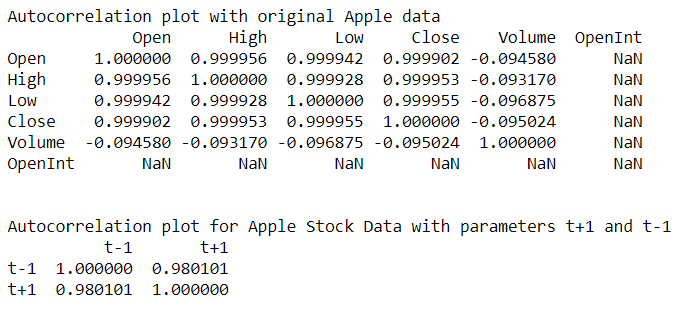
**Visualization for Tesla Stock prediction**



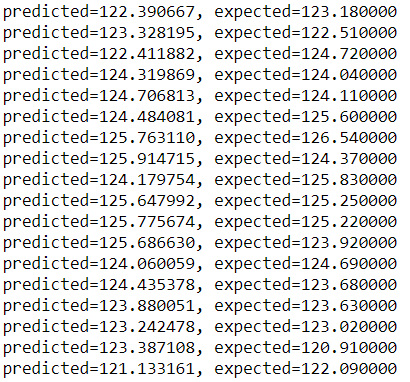
**Results of Stock Prediction for Apple using Auto Regression Model**

**Autocorrelation Plot**

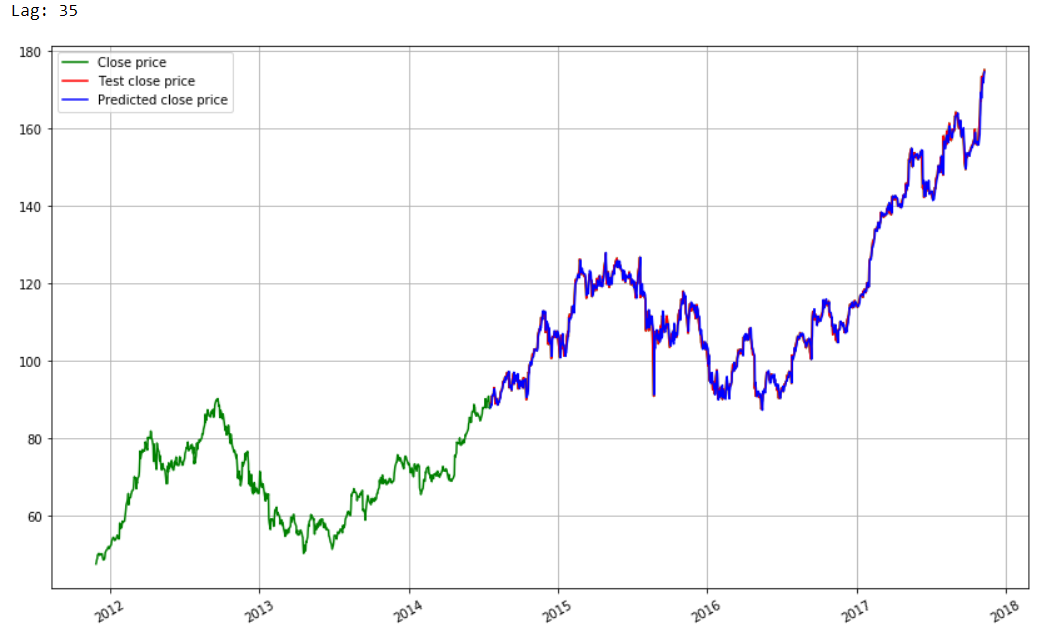


**Autocorrelation Results:**

**Prediction for Apple Stocks:**

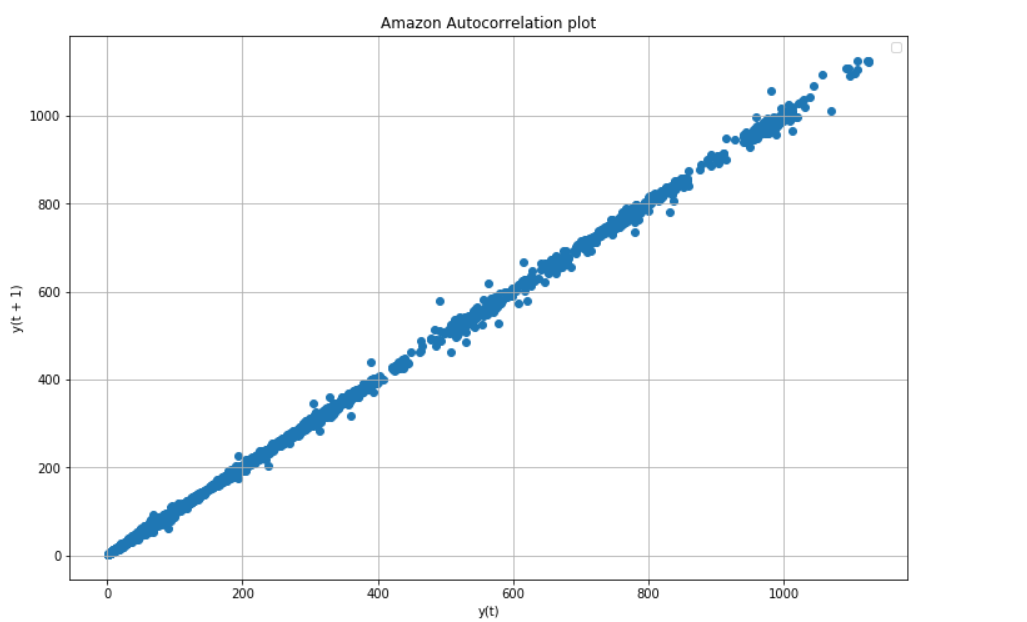


**Visualization for Apple Stock prediction**

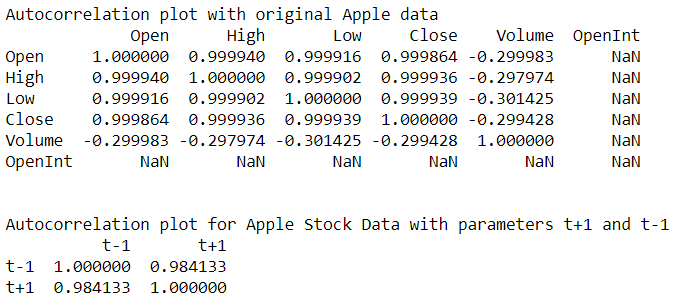


**Results of Stock Prediction for Amazon using Auto Regression Model**

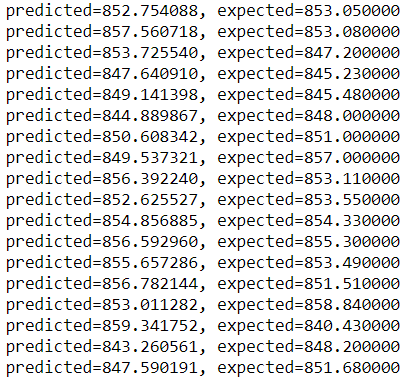
**Autocorrelation Plot**



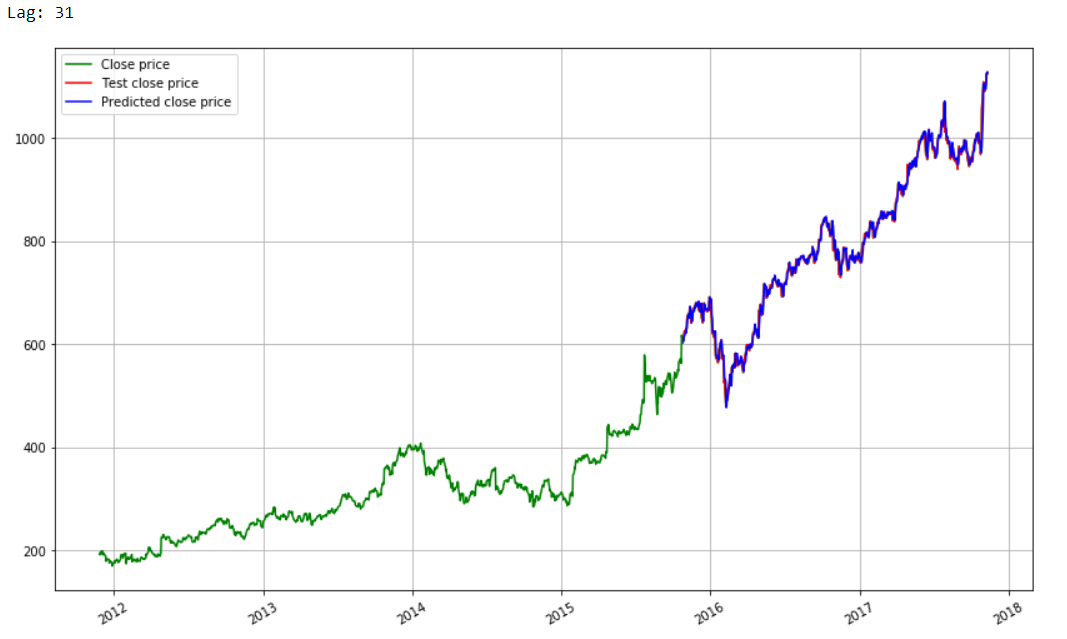
**Autocorrelation Results**



**Prediction for Amazon Stocks:**

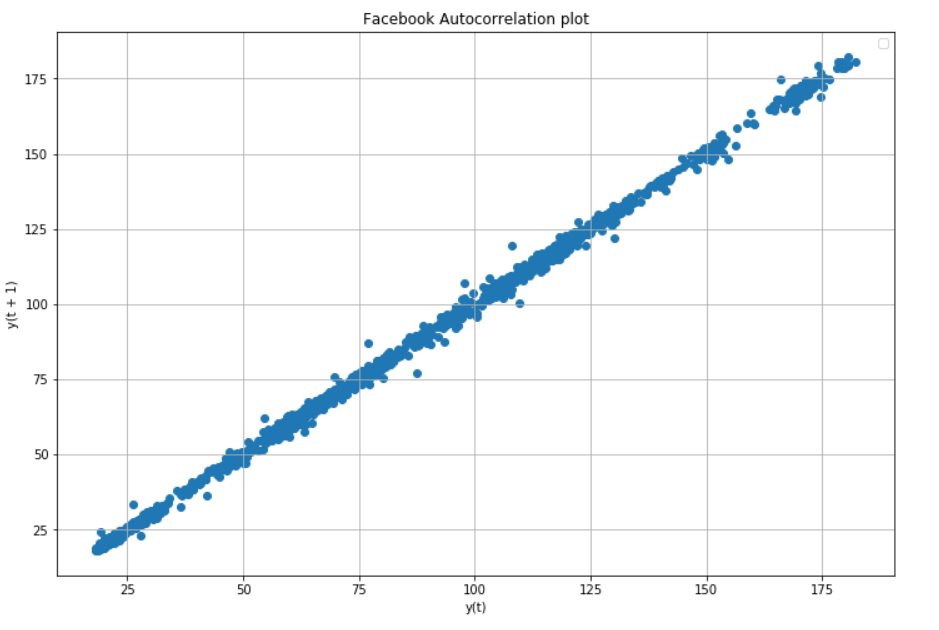


**Visualization for Amazon Stock prediction**

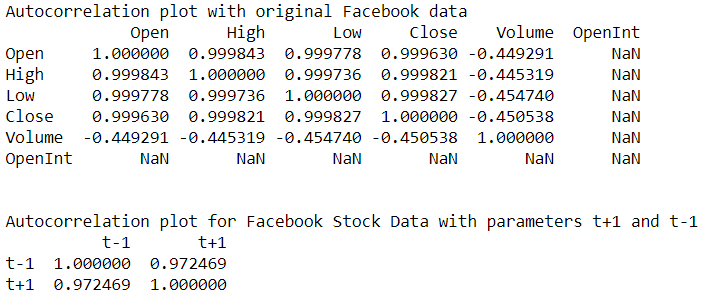


**Results of Stock Prediction for Facebook using Auto Regression Model**

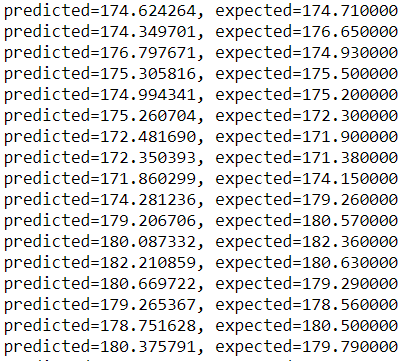
**Autocorrelation Plot**



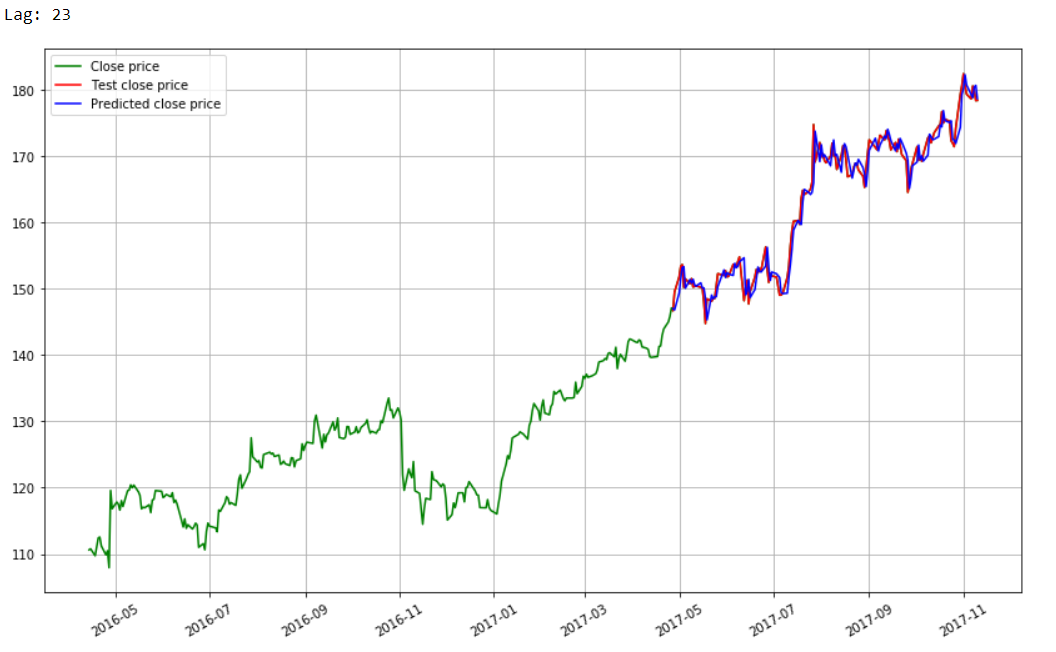
**Autocorrelation Results**



**Prediction for Facebook Stocks:**



**Visualization for Facebook Stock prediction**



**ARIMA:**

ARIMA is an acronym that stands for AutoRegressive Integrated Moving Average. It is a class of model that captures a suite of different standard temporal structures in time series data. [ARIMA model](https://en.wikipedia.org/wiki/Autoregressive_integrated_moving_average) is a class of statistical models for analyzing and forecasting time series data. It explicitly caters to a suite of standard structures in time series data, and as such provides a simple yet powerful method for making skillful time series forecasts.

ARIMA is an acronym that stands for AutoRegressive Integrated Moving Average. It is a generalization of the simpler AutoRegressive Moving Average and adds the notion of integration.

The descriptive form of ARIMA acronym is as follows:

* **AR**: Autoregression. A model that uses the dependent relationship between an observation and some number of lagged observations.
* **I**: Integrated. The use of differencing of raw observations (e.g. subtracting an observation from an observation at the previous time step) in order to make the time series stationary.
* **MA**: Moving Average. A model that uses the dependency between an observation and a residual error from a moving average model applied to lagged observations.

Each of these components are explicitly specified in the model as a parameter. A standard notation is used of ARIMA (p, d, q) where the parameters are substituted with integer values to quickly indicate the specific ARIMA model being used.

The parameters of the ARIMA model are defined as follows:

* **p**: The number of lag observations included in the model, also called the lag order.
* **d**: The number of times that the raw observations are differenced, also called the degree of differencing.
* **q**: The size of the moving average window, also called the order of moving average.

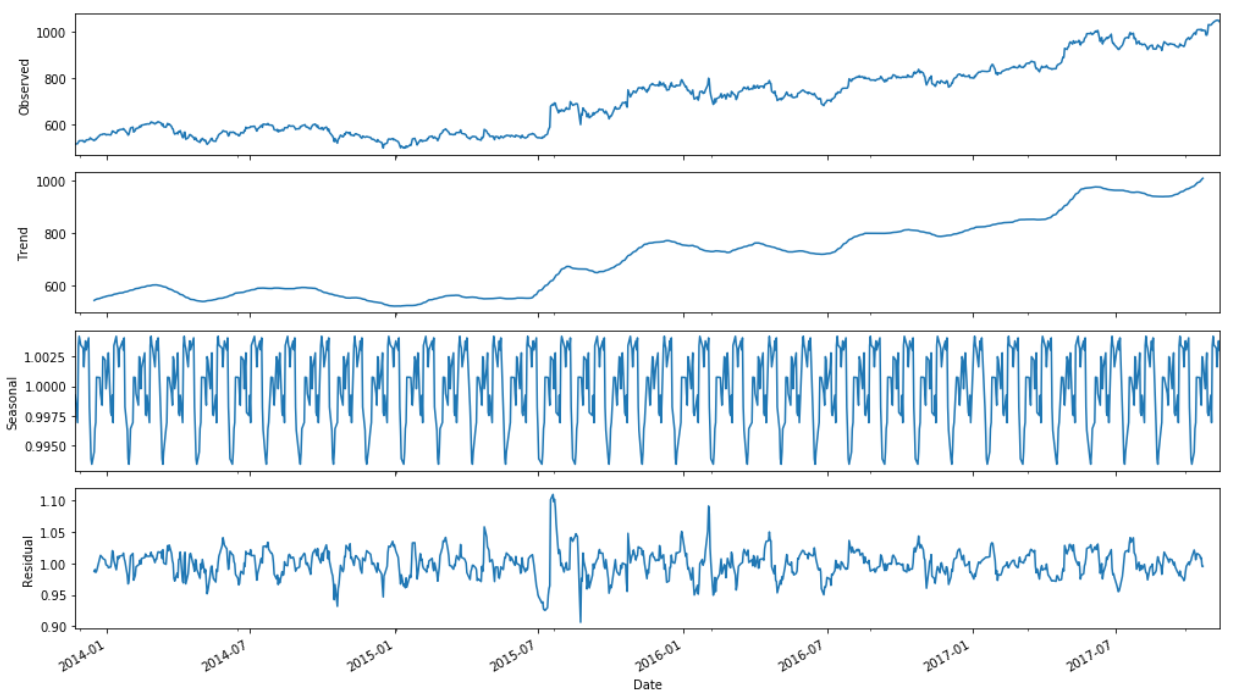
A value of 0 can be used for a parameter, which indicates to not use that element of the model. This way, the ARIMA model can be configured to perform the function of an ARIMA model, and even a simple AR, I, or MA model. Adopting an ARIMA model for a time series assumes that the underlying process that generated the observations is an ARIMA process.

The python statsmodels library provides the capability to fit an ARIMA model. An ARIMA model can be created using the statsmodels library as follows:

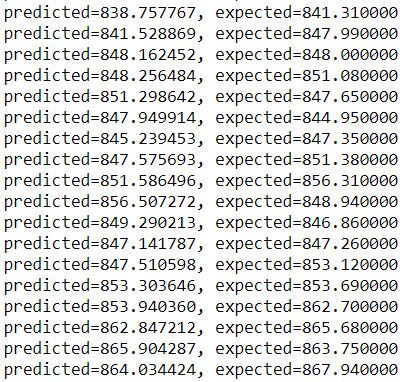
1. Define the model by calling [ARIMA()](http://statsmodels.sourceforge.net/devel/generated/statsmodels.tsa.arima_model.ARIMA.html) and passing in the p, d, and q parameters.
2. The model is prepared on the training data by calling the [fit()](http://statsmodels.sourceforge.net/devel/generated/statsmodels.tsa.arima_model.ARIMA.fit.html) function.
3. Predictions can be made by calling the [predict()](http://statsmodels.sourceforge.net/devel/generated/statsmodels.tsa.arima_model.ARIMA.predict.html) function and specifying the index of the time or times to be predicted.

First, we fit an ARIMA (1,1,0) model. This sets the lag value to 1 for autoregression, uses a difference order of 1 to make the time series stationary, and uses a moving average model of 0. When fitting the model, a lot of debug information is provided about the fit of the linear regression model. We can turn this off by setting the *disp* argument to 0.

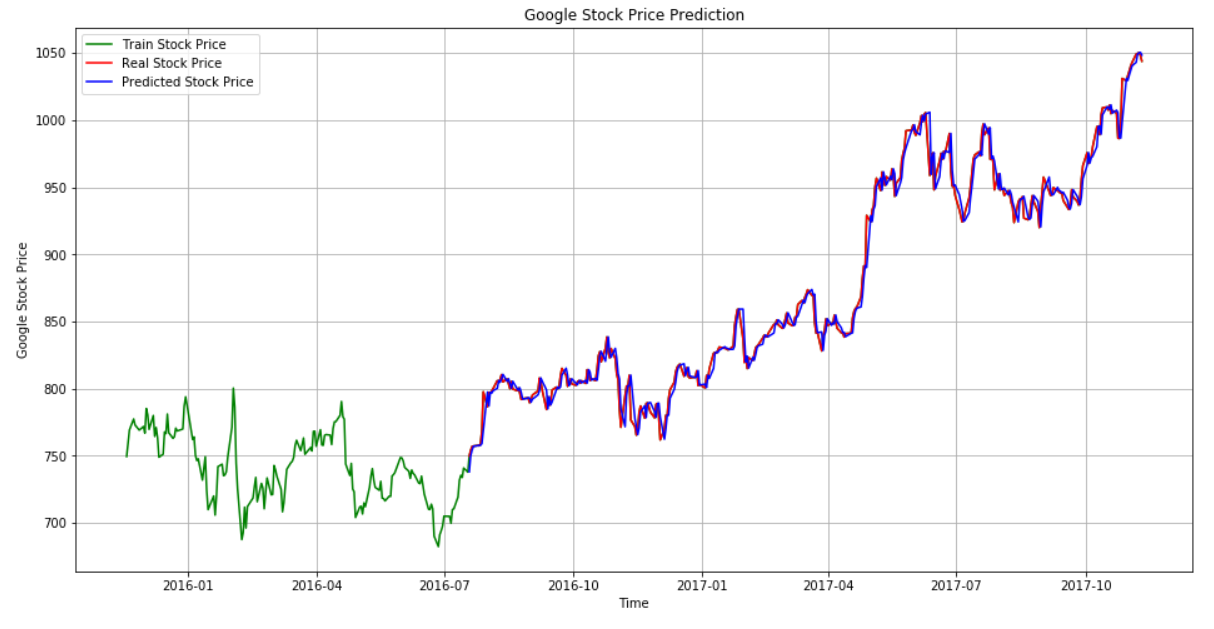
**Results of Stock Prediction for Google using ARIMA**



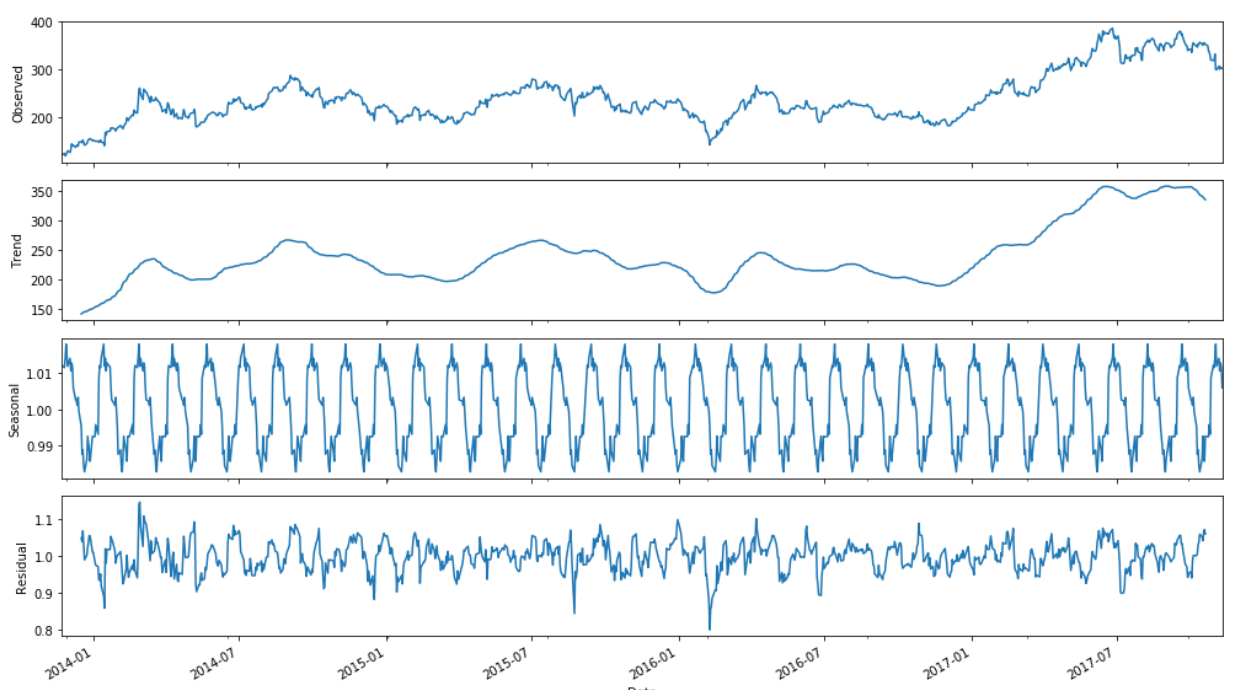
**Prediction for Google Stocks:**



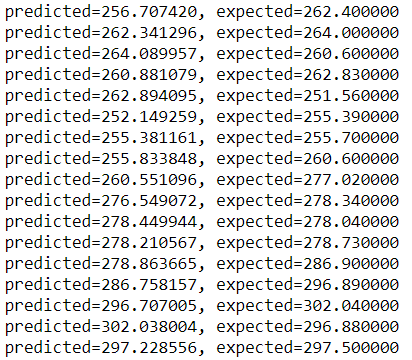
**Visualization for Google Stock prediction**



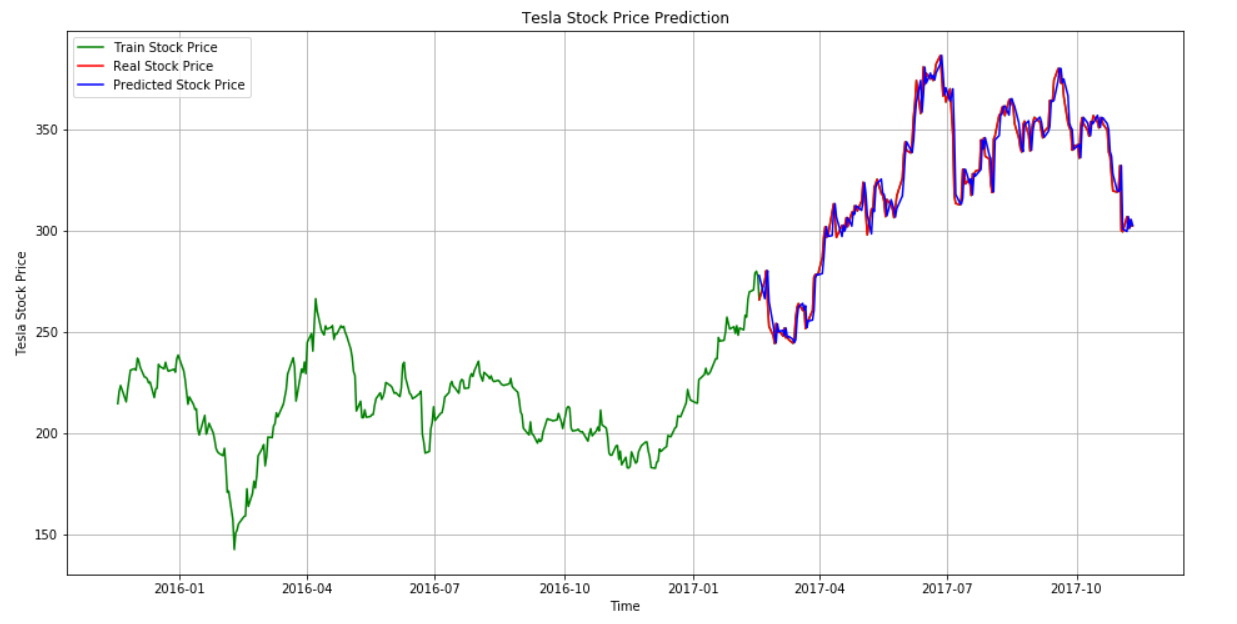
**Results of Stock Prediction for Tesla using ARIMA**



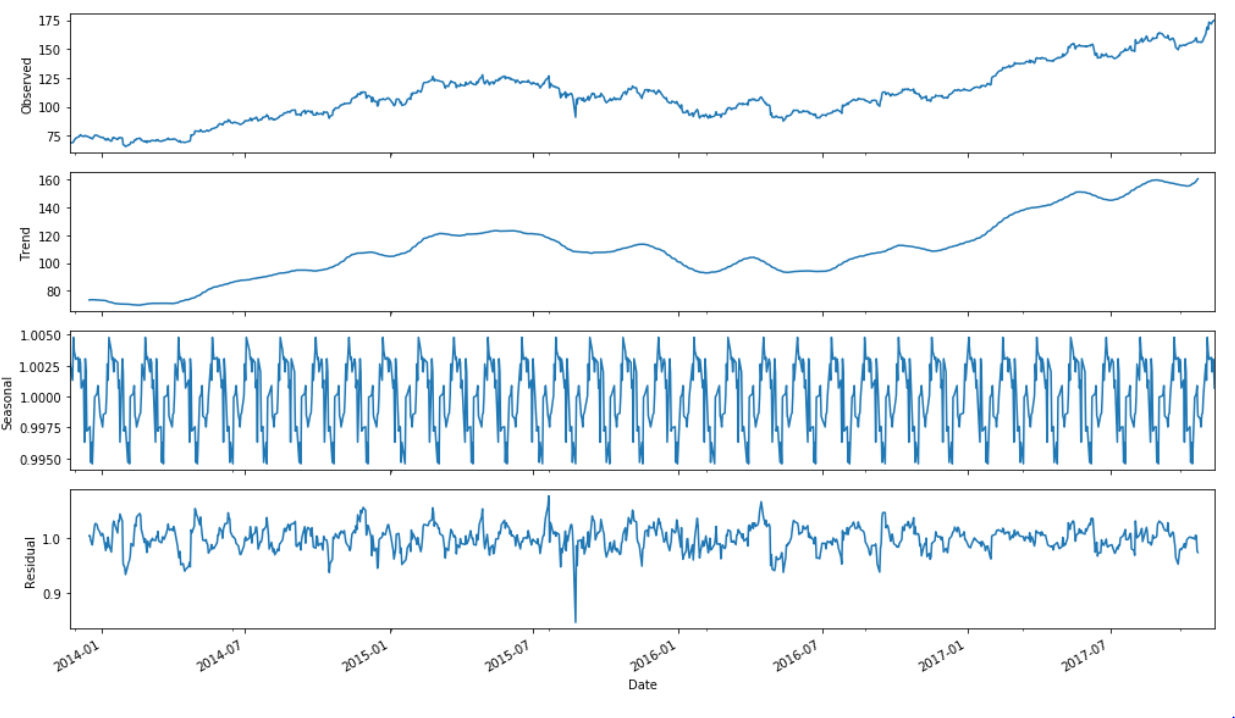
**Prediction for Tesla Stocks:**



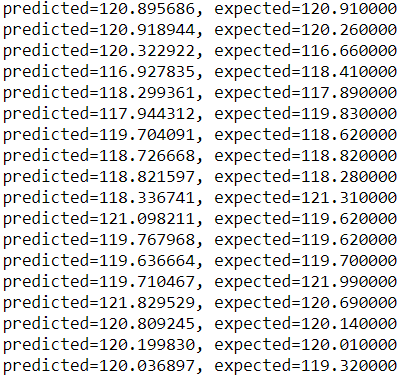
**Visualization for Tesla Stock prediction**



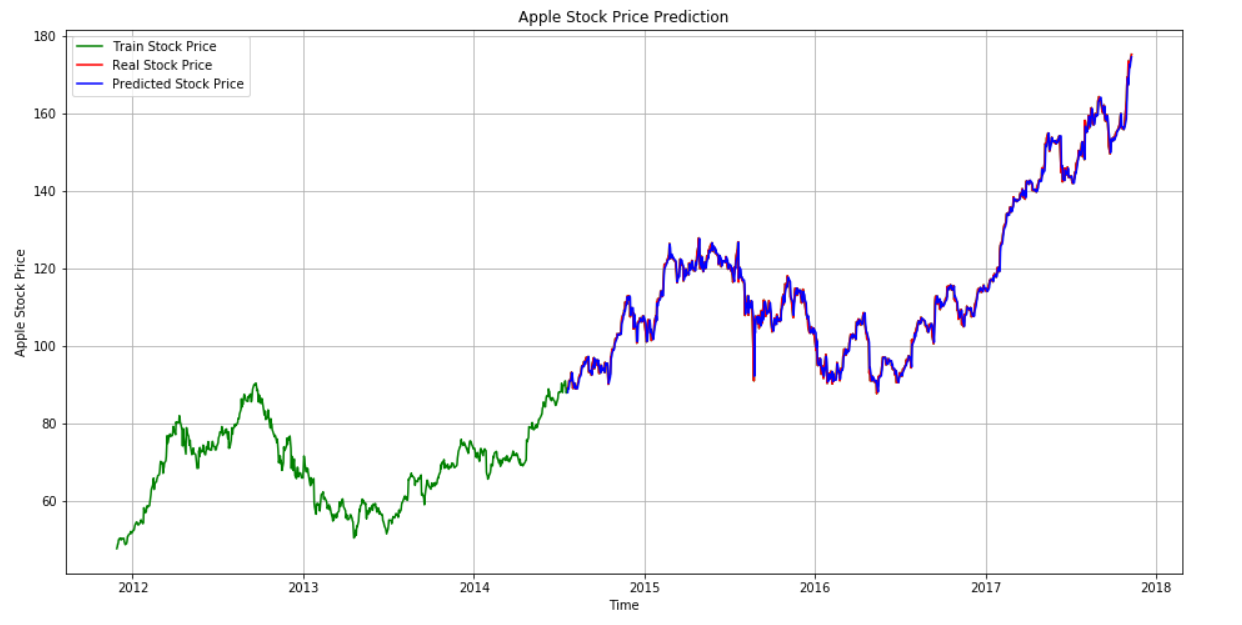
**Results of Stock Prediction for Apple using ARIMA**



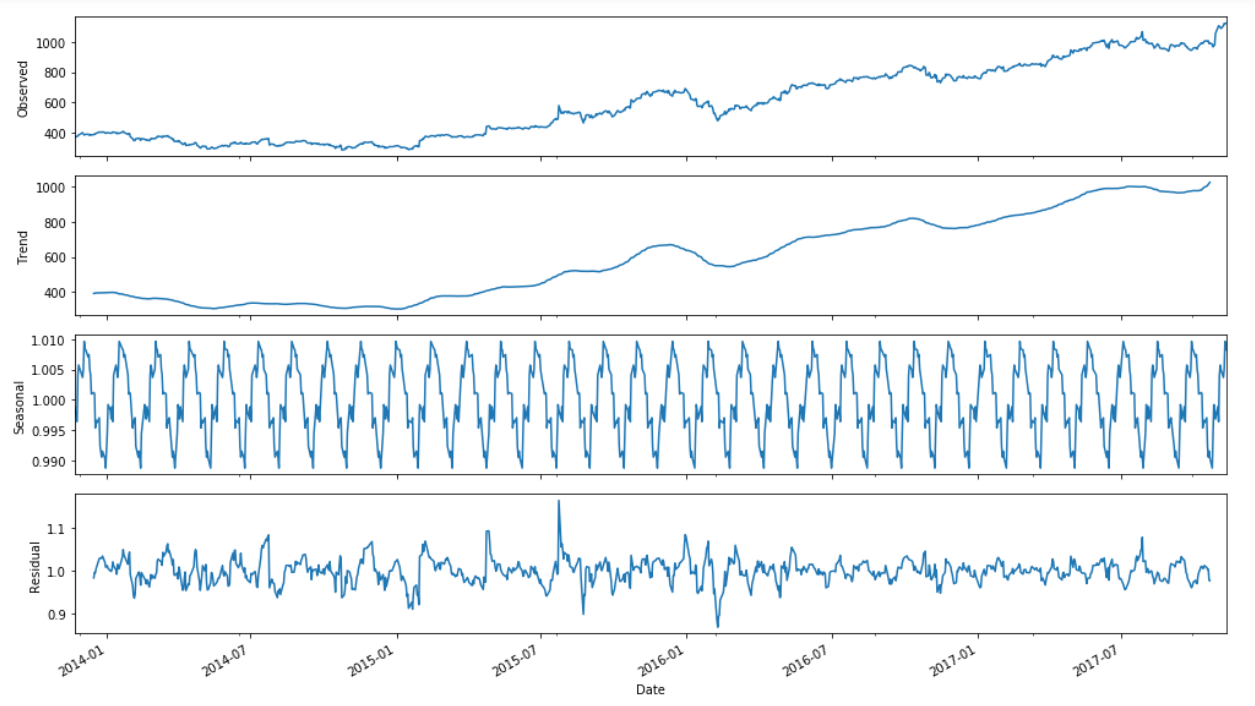
**Prediction for Apple Stocks:**



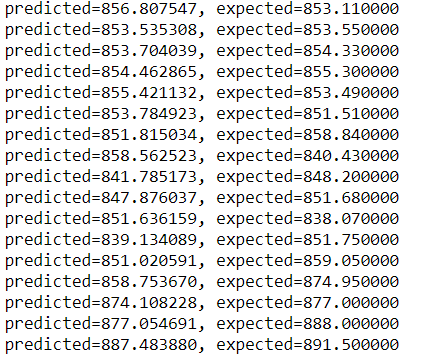
**Visualization for Apple Stock prediction**



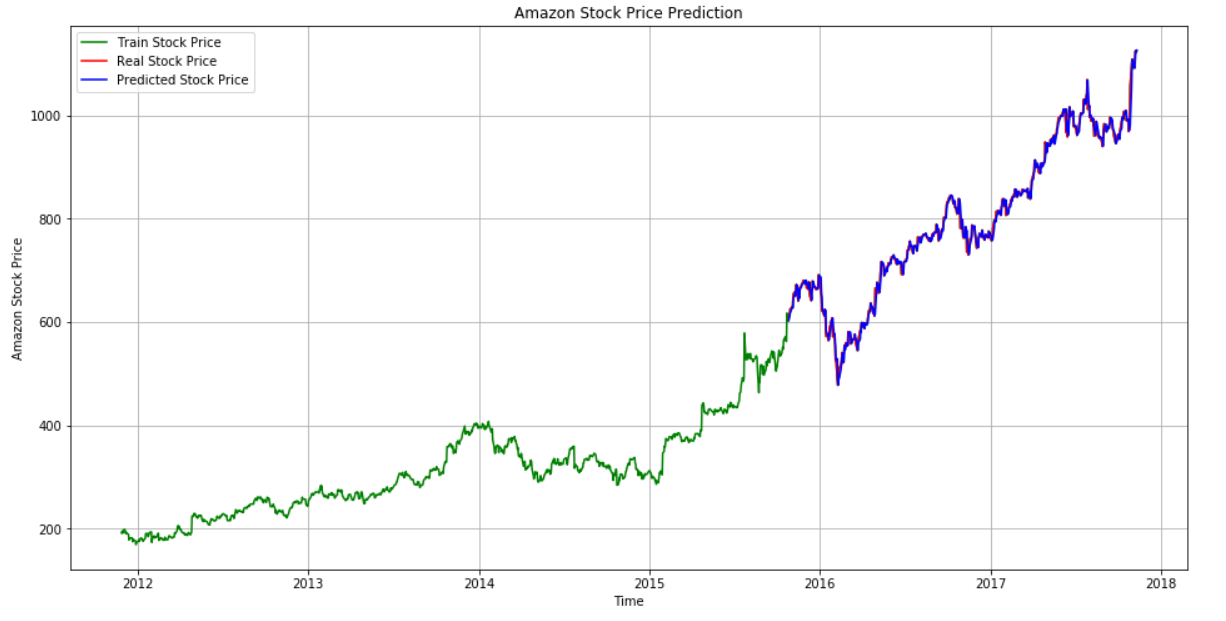
**Results of Stock Prediction for Amazon using ARIMA**



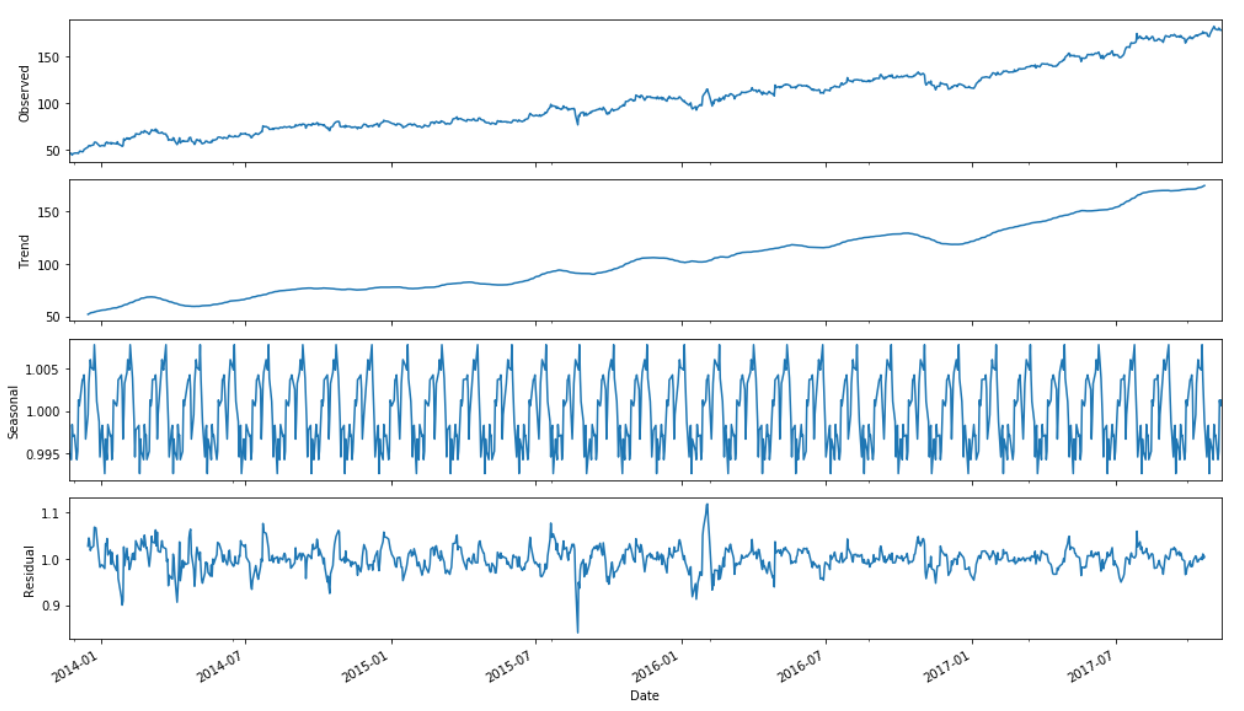
**Prediction for Amazon Stocks:**



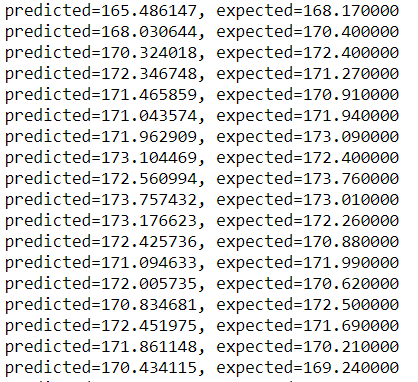
**Visualization for Amazon Stock prediction**



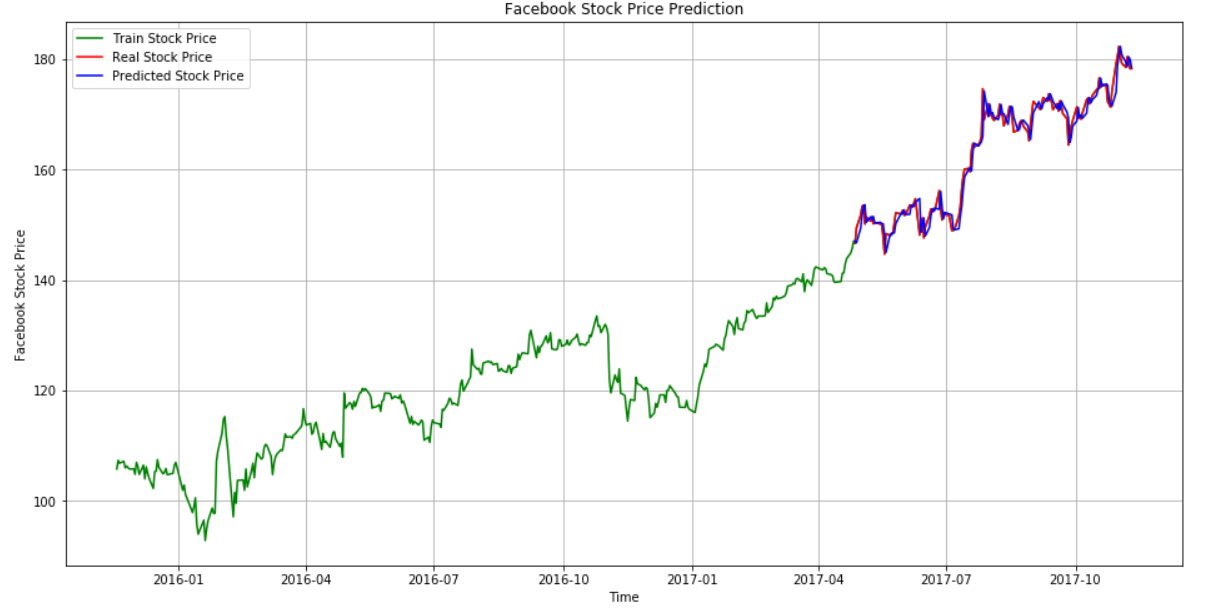
**Results of Stock Prediction for Facebook using ARIMA**



**Prediction for Facebook Stocks:**

ss

**Visualization for Facebook Stock prediction**



**Linear Regression**

Linear regression analysis is used to predict the value of a variable based on the value of another variable. The variable you want to predict is called the dependent variable. The variable you are using to predict the other variable's value is called the independent variable.

Linear regression performs the task to predict a dependent variable value (y) based on a given independent variable (x). So, this regression technique finds out a linear relationship between x (input) and y(output). Hence, the name is Linear Regression. If we plot the independent variable (x) on the x-axis and dependent variable (y) on the y-axis, linear regression gives us a straight line that best fits the data points, as shown in the figure below.

The equation for Linear Regression: **Y = a + bX**

On the other hand, the line of regression of X on Y is given by X = c + dY which is used to predict the unknown value of variable X using the known value of variable Y.

**Regression Performance:**

The variation of actual responses 𝑦ᵢ, 𝑖 = 1, …, 𝑛, occurs partly due to the dependence on the predictors 𝐱ᵢ. However, there is also an additional inherent variance of the output.

The coefficient of determination, denoted as 𝑅², tells you which amount of variation in 𝑦 can be explained by the dependence on 𝐱 using the particular regression model. Larger 𝑅² indicates a better fit and means that the model can better explain the variation of the output with different inputs.

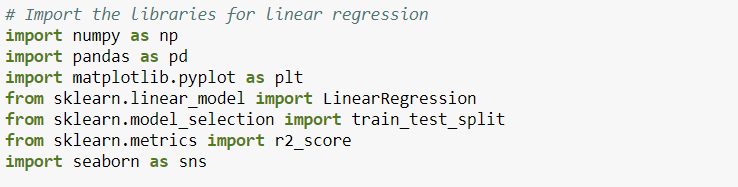
The value 𝑅² = 1 corresponds to SSR = 0, that is to the perfect fit since the values of predicted and actual responses fit completely to each other.

### **Linear Regression With scikit-learn**

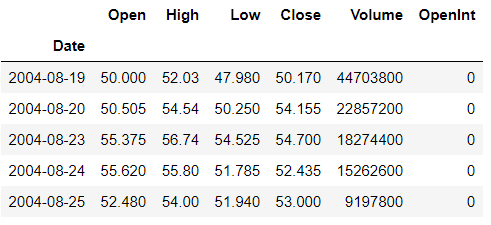
The steps involved in implementing Linear\_Regression on our dataset.

1. We imported the packages and classes we needed for Linear Regression
2. Provided the data we needed and created visualization for understanding data
3. Created a regression model and fitted it with existing data.
4. Checked the results of model fitting to know whether the model is satisfactory or not.
5. Calculated the Train and Test set by R2 – score.

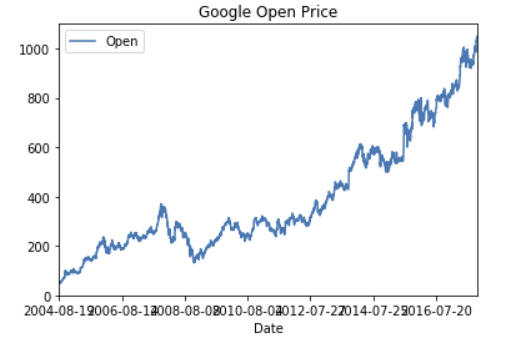
The packages and classes we imported for predicting using Linear Regression

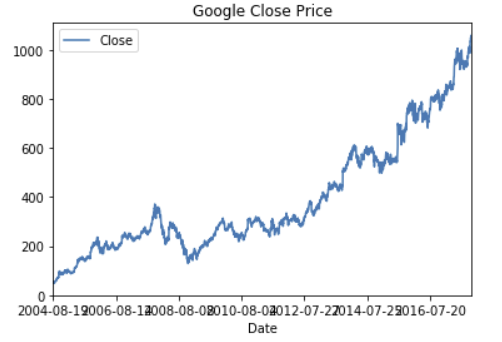


Glimpse of data set using head.

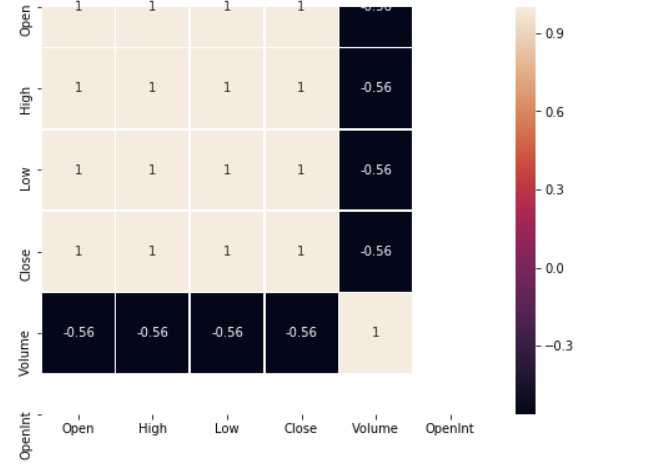


**Visualization of the Open and Close Price for Google Stock**

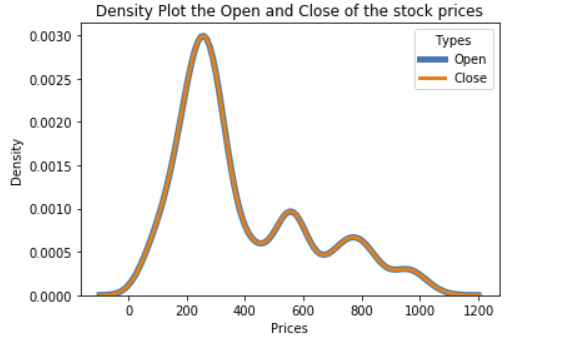




**Heatmap for visualization**



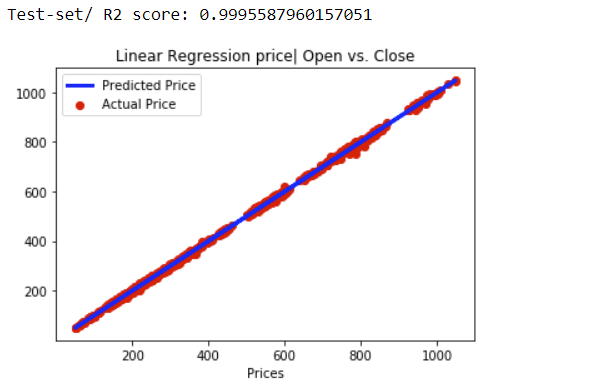
**Density Plot for Open and Close of the stock prices**



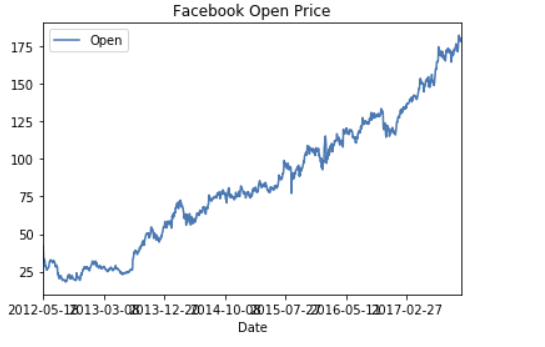
**Linear Regression Model for Train and Test Data Set**

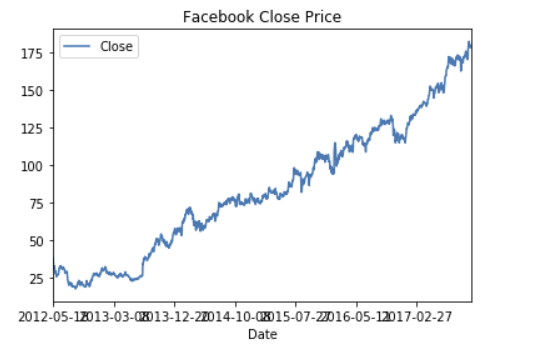
**Results of Stock Prediction for Google using Linear Regression**



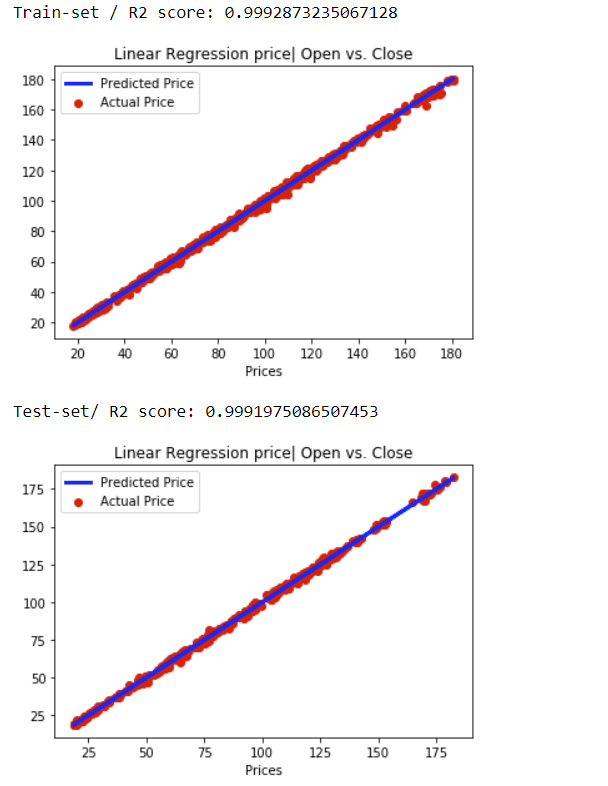


**Visualization of the Open and Close Price for Google Stock**

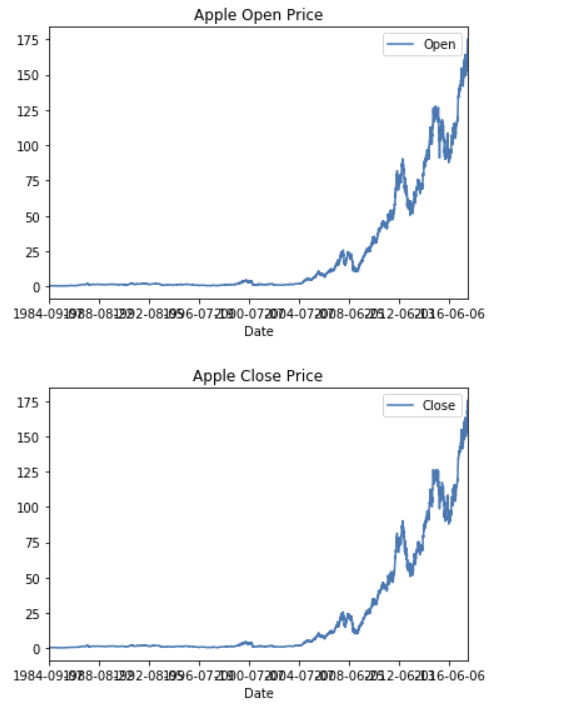




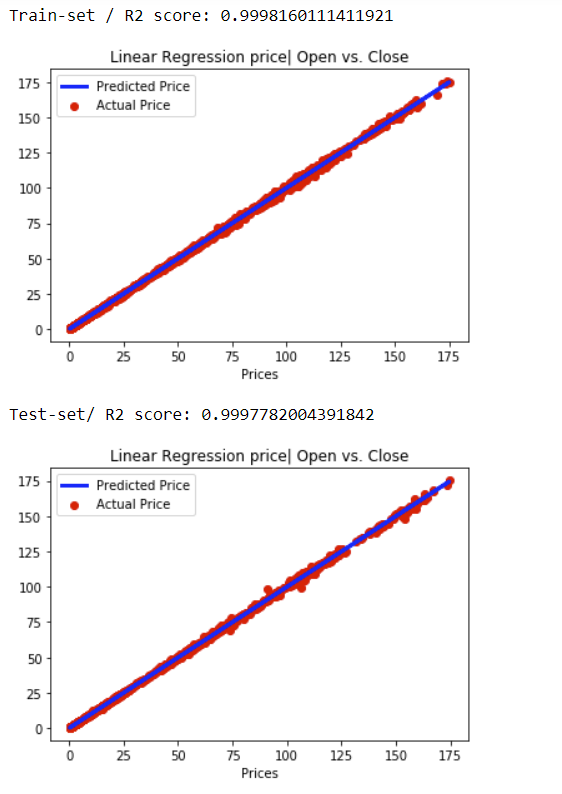
**Results of Stock Prediction for Facebook using Linear Regression**



**Visualization of the Open and Close Price for Apple Stock**

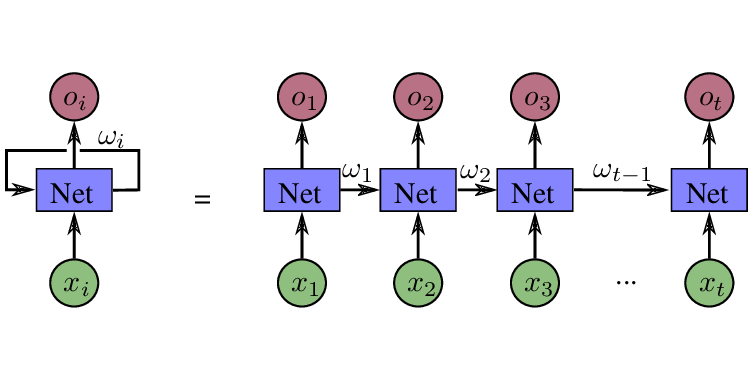


**Results of Stock Prediction for Apple using Linear Regression**



**Recurrent Neural Network (RNN)**

Recurrent Neural Network(RNN) are a type of [Neural Network](https://www.geeksforgeeks.org/tag/neural-network/) where the output from previous step are fed as input to the current step. In traditional neural networks, all the inputs and outputs are independent of each other, but in cases like when it is required to predict the next word of a sentence, the previous words are required and hence there is a need to remember the previous words. Thus, RNN came into existence, which solved this issue with the help of a Hidden Layer. The main and most important feature of RNN is Hidden state, which remembers some information about a sequence.



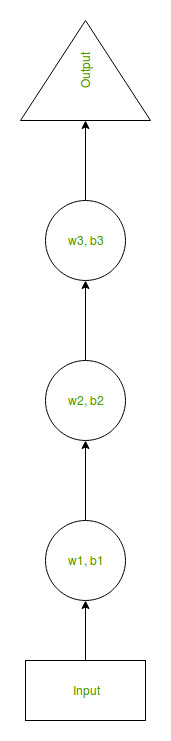
RNN have a “memory” which remembers all information about what has been calculated. It uses the same parameters for each input as it performs the same task on all the inputs or hidden layers to produce the output. This reduces the complexity of parameters, unlike other neural networks.

**To understand how RNN works**

The working of an RNN can be understood with the help of below example:

Example:

Suppose there is a deeper network with one input layer, three hidden layers and one output layer. Then like other neural networks, each hidden layer will have its own set of weights and biases, let’s say, for hidden layer 1 the weights and biases are (w1, b1), (w2, b2) for second hidden layer and (w3, b3) for third hidden layer. This means that each of these layers are independent of each other, i.e. they do not memorize the previous outputs.

****

Now the RNN will do the following:

* RNN converts the independent activations into dependent activations by providing the same weights and biases to all the layers, thus reducing the complexity of increasing parameters and memorizing each previous output by giving each output as input to the next hidden layer.
* Hence these three layers can be joined together such that the weights and bias of all the hidden layers is the same, into a single recurrent layer.

Formula for calculating current state:

rnn  
where:

ht -> current state

ht-1 -> previous state

xt -> input state

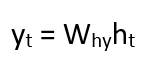
* Formula for applying Activation function(tanh):

rnn  
where:

whh -> weight at recurrent neuron

wxh -> weight at input neuron

* Formula for calculating output:



Yt -> output

Why -> weight at output layer

**Training through RNN**

1. A single time step of the input is provided to the network.
2. Then calculate its current state using set of current input and the previous state.
3. The current ht becomes ht-1 for the next time step.
4. One can go as many time steps according to the problem and join the information from all the previous states.
5. Once all the time steps are completed the final current state is used to calculate the output.
6. The output is then compared to the actual output i.e. the target output and the error is generated.
7. The error is then back propagated to the network to update the weights and hence the network (RNN) is trained.

**Advantages of Recurrent Neural Network**

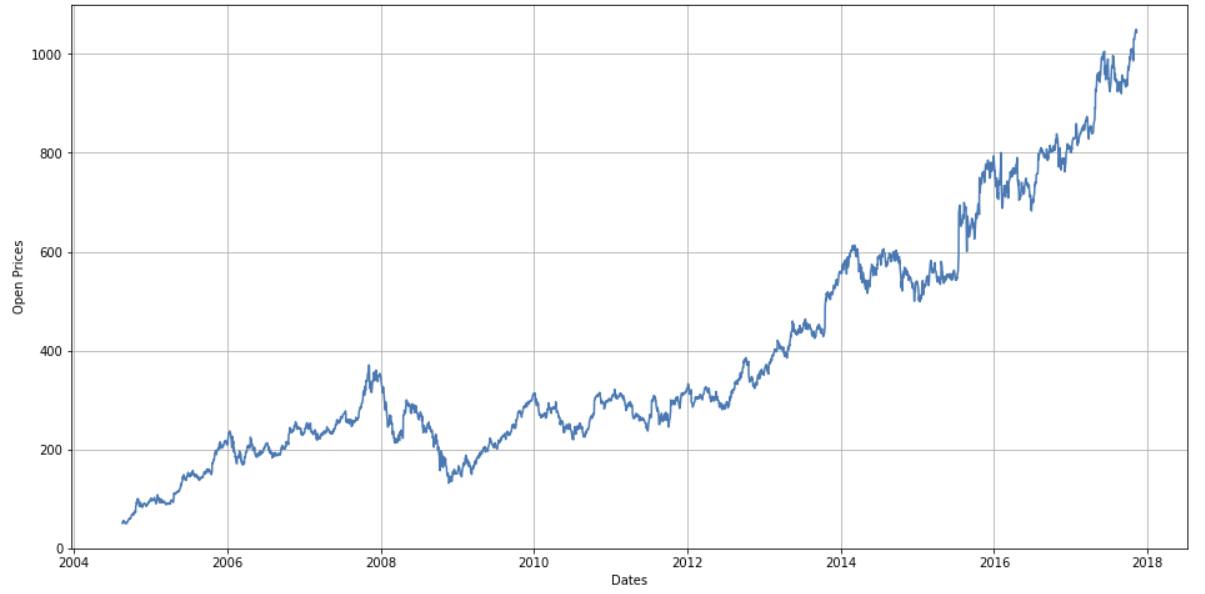
1. An RNN remembers each information through time. It is useful in time series prediction only because of the feature to remember previous inputs as well. This is called Long Short-Term Memory.
2. Recurrent neural network is even used with convolutional layers to extend the effective pixel neighborhood.

**Disadvantages of Recurrent Neural Network**

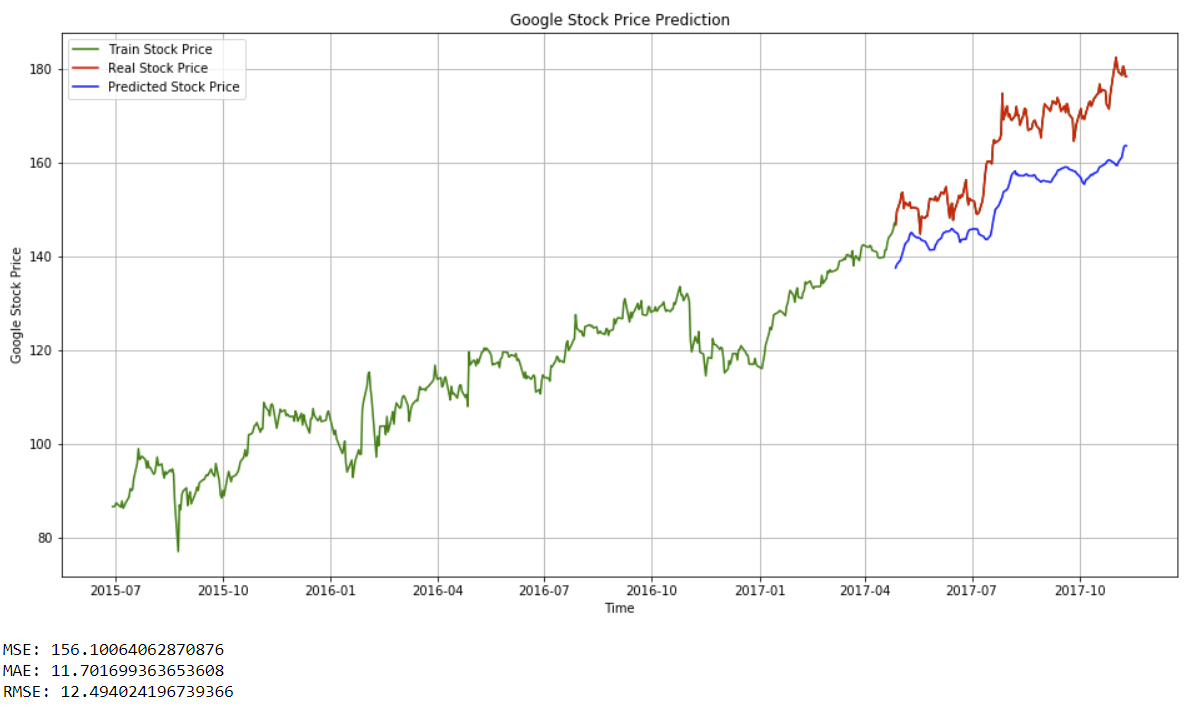
1. Gradient vanishing and exploding problems.
2. Training an RNN is a very difficult task.
3. It cannot process very long sequences if using tanh or relu as an activation function.

**Results of Stock Prediction for Google using RNN Model**

**Visualization of Google Stock Open Price**

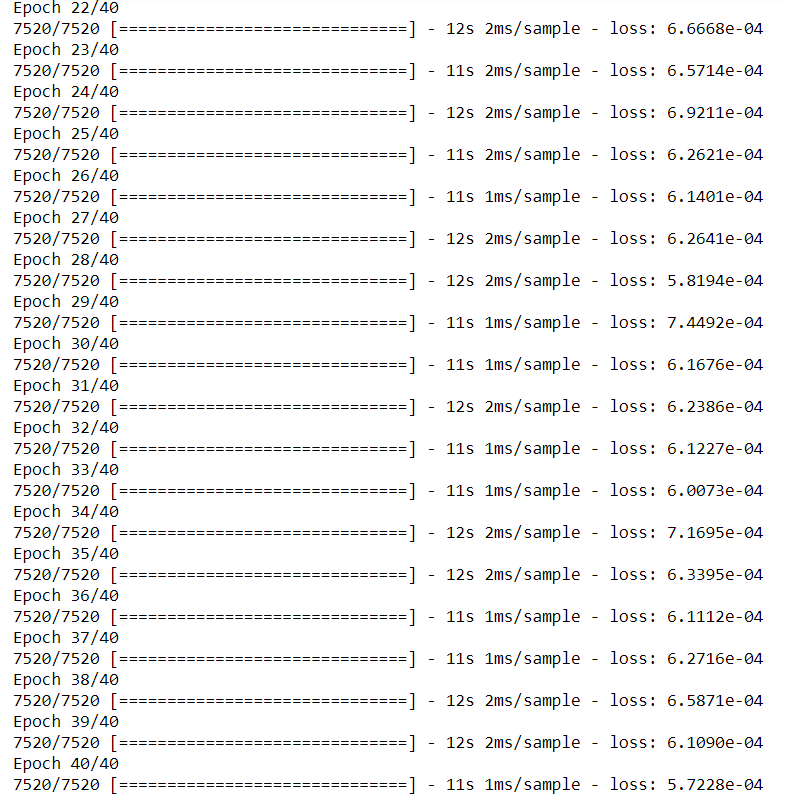


**Google Stock Price Prediction**

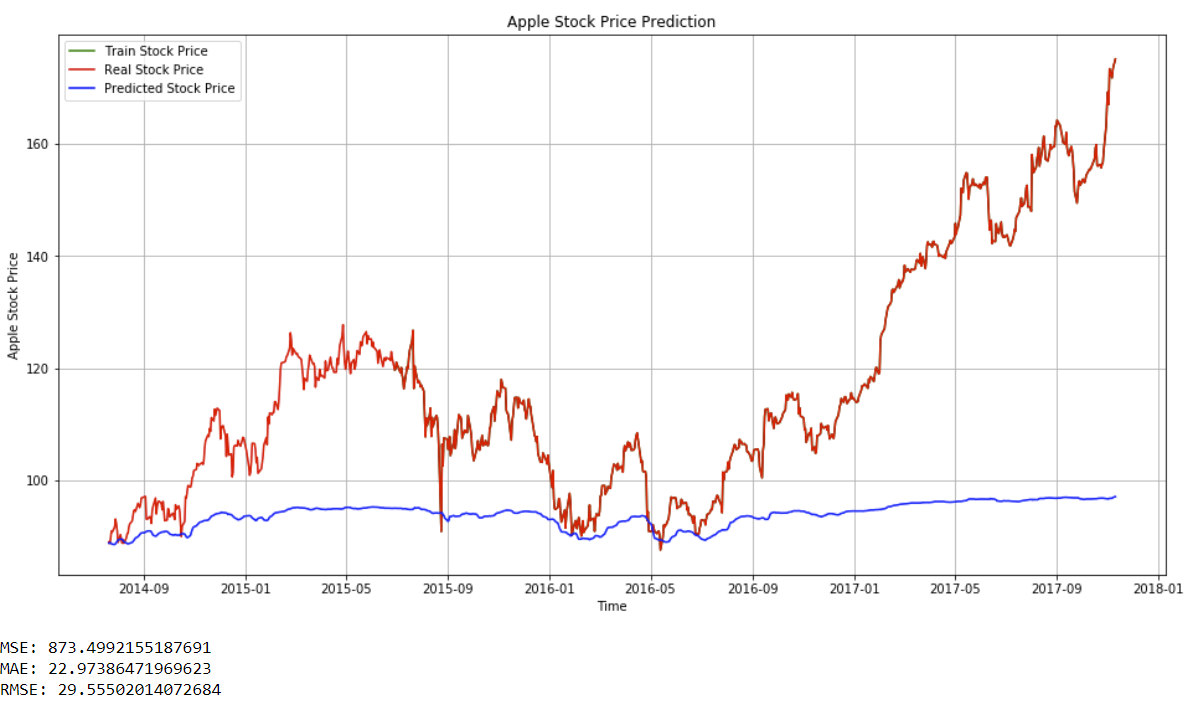


**Results of Stock Prediction for Apple using RNN Model**

**Taking Epoch value as 40 and training on 7520 samples**

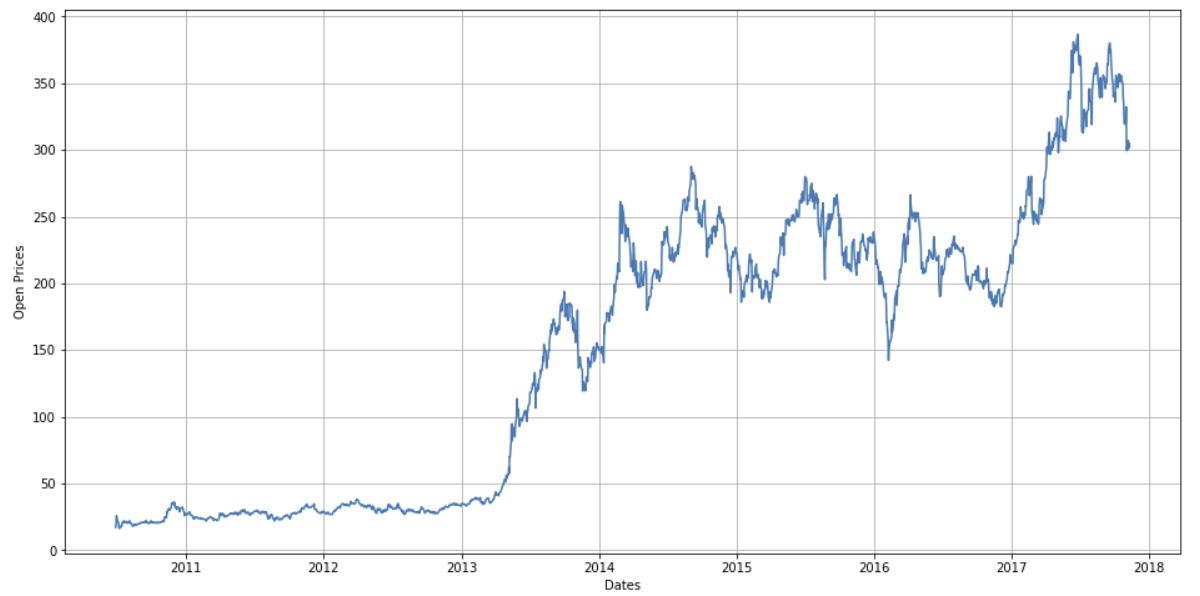


**Apple Stock Price Prediction**

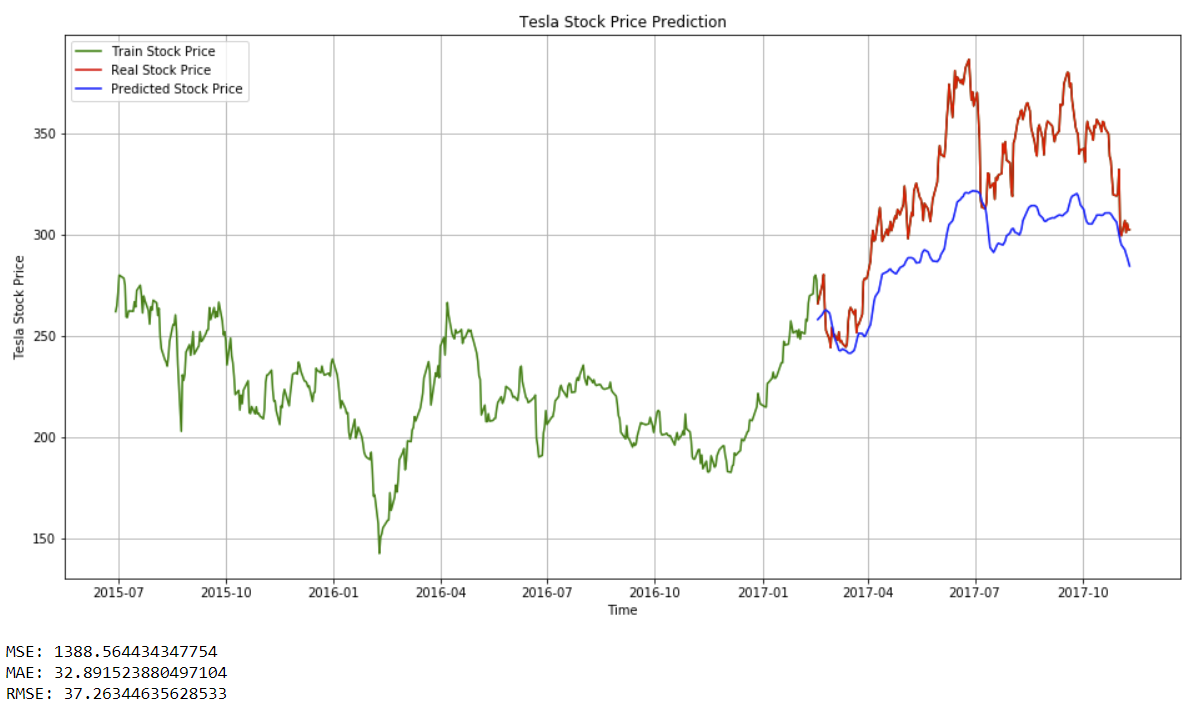


**Results of Stock Prediction for Tesla using RNN Model**

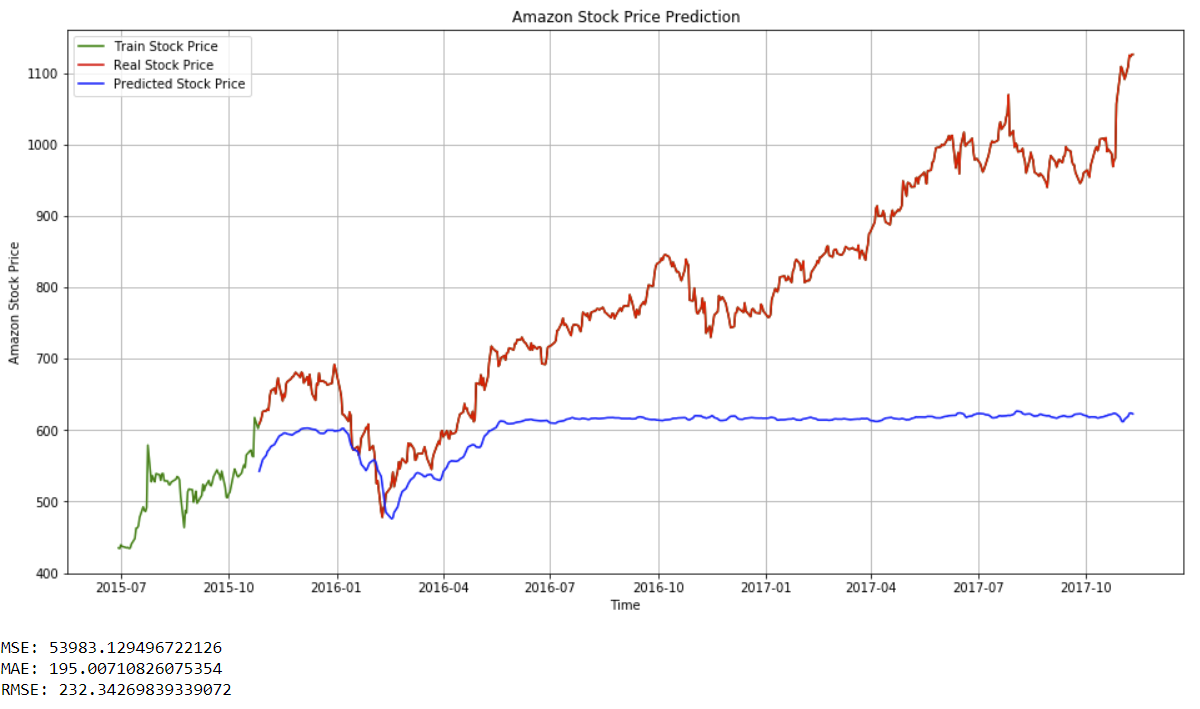
**Visualization of Tesla Open Price**



**Tesla Stock Price Prediction**

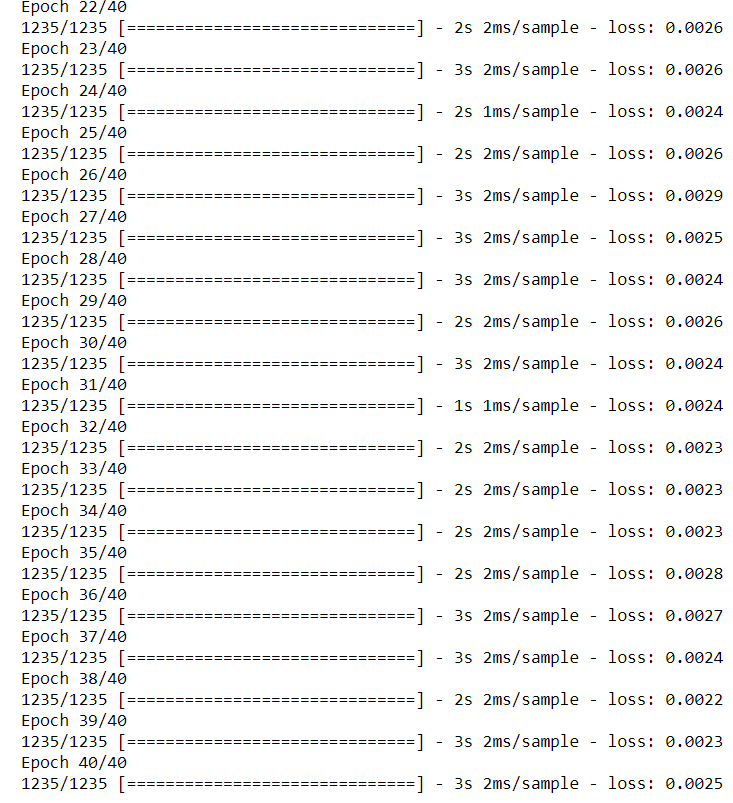


**Amazon Stock Price Prediction**

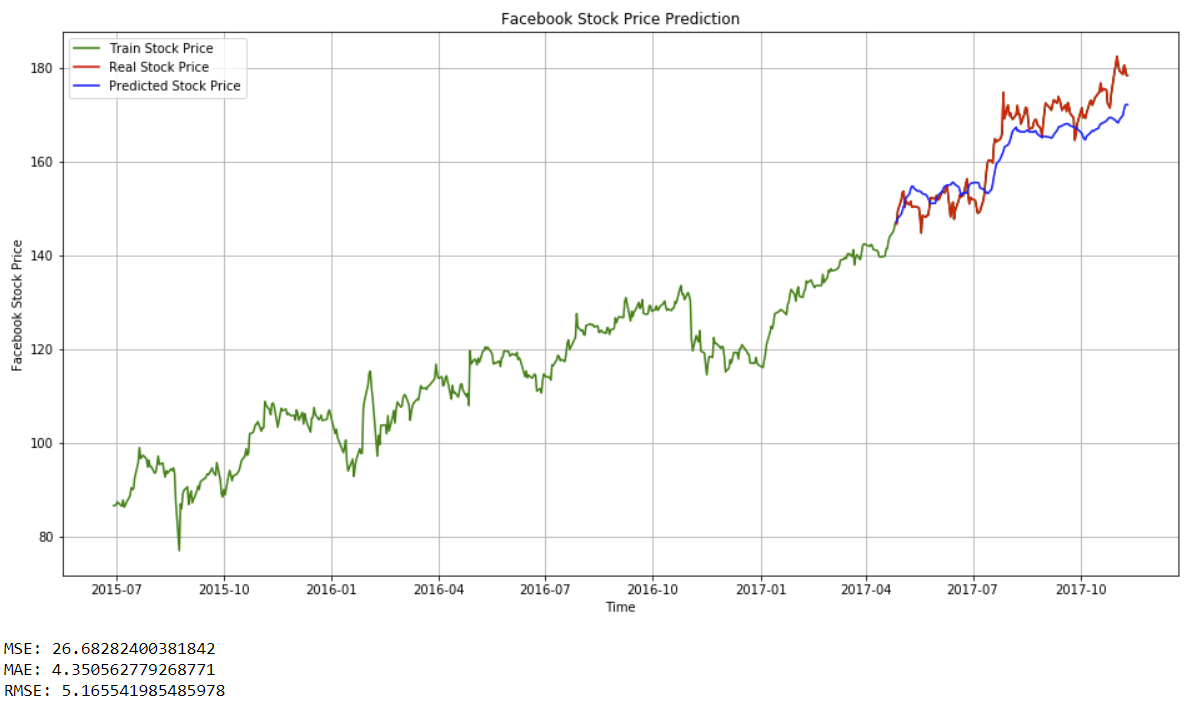


**Results of Stock Prediction for Facebook using RNN Model**

**Taking Epoch value as 40 and training on 1235 samples**



**Facebook Stock Price Prediction**



1. **Conclusion:**

* Determining the stock market forecasts is always been challenging work.
* Thus, as we can see above in our proposed method, we train the data with multiple Machine Learning and Time Series algorithms using existing stock dataset that is available. We used this data to predict the stock prices of the several companies across United States.

**9.0 Future Scope:**

* The proposed model does not predict well for sudden changes in the trend of stock data
* This occurs due to external factors and real-world changes affecting the stock market.
* So, we can overcome this problem by implementing Sentiment Analysis on News feeds and Advanced Neural Network model to improve the results.
* We can also try to modify the same system to an online-learning system that captures in real-time

**10.0 References:**

* <https://www.kaggle.com/>
* <https://machinelearningmastery.com/arima-for-time-series-forecasting-with-python/>
* <https://machinelearningmastery.com/autoregression-models-time-series-forecasting-python/>
* Stock Market Analysis: A Review and Taxonomy of Prediction Techniques- International Journal of Financial Studies