Stock price prediction using Time series data

Submitted by,

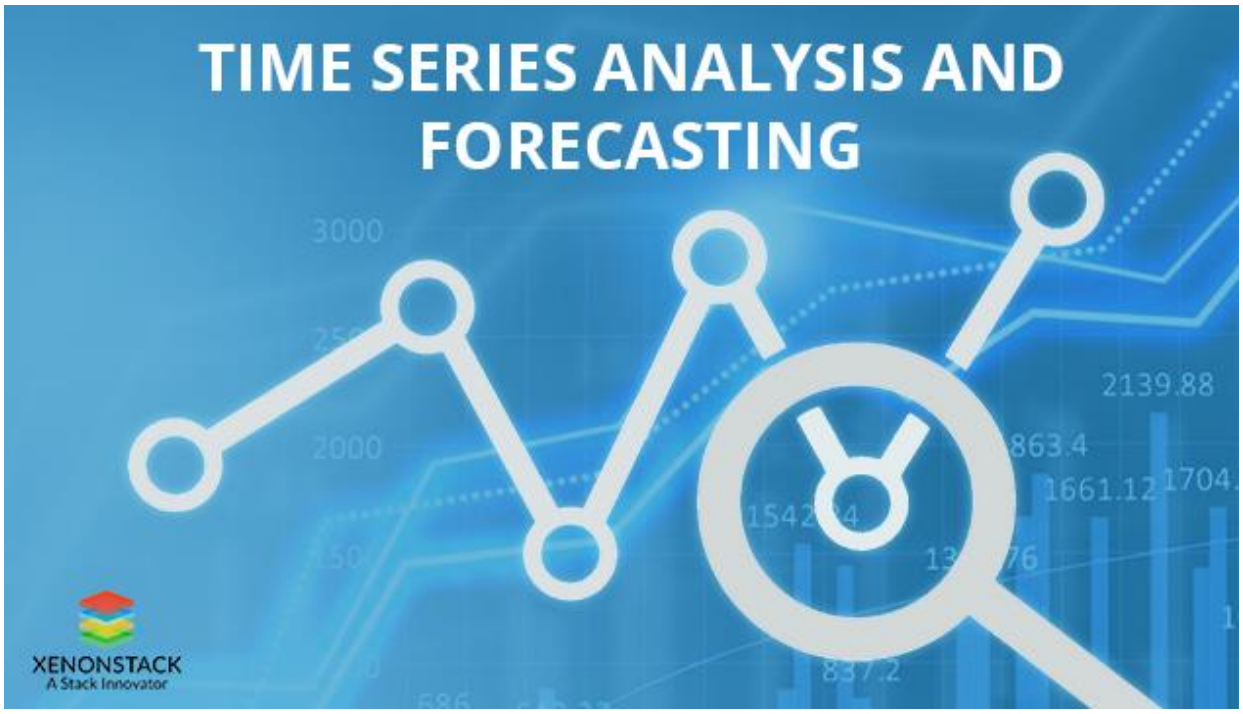
Hithul kannan

UG-student

BE-Mechanical

Content's

- Introduction
- Abstract
- Time series data
- ARIMA Model
- Data set used
- Implementation
- Conclusion



Introduction

Stock and financial markets tend to be unpredictable and even illogical, just like the outcome of the Brexit vote or the last US elections. Due to these characteristics, financial data should be necessarily possessing a rather turbulent structure which often makes it hard to find reliable patterns. Modeling turbulent structures requires machine learning algorithms capable of finding hidden structures within the data and predict how they will affect them in the future. The most efficient methodology to achieve this is Machine Learning and Deep Learning. Deep learning can deal with complex structures easily and extract relationships that further increase the accuracy of the generated results.

Machine learning has the potential to ease the whole process by analyzing large chunks of data, spotting significant patterns and generating a single output that navigates traders towards a particular decision based on predicted asset prices. Stock prices are not randomly generated values instead they can be treated as a discrete-time series model which is based on a set of well-defined numerical data items collected at successive points at regular intervals of time. Since it is essential to identify a model to analyze trends of stock prices with adequate information for decision making, it recommends that transforming the time series using ARIMA is a better algorithmic approach than forecasting directly, as it gives more authentic and reliable results.

Autoregressive Integrated Moving Average (ARIMA) Model converts non-stationary data to stationary data before working on it. It is one of the most popular models to predict linear time series data.

ARIMA model has been used extensively in the field of finance and economics as it is known to be robust, efficient and has a strong potential for short-term share market prediction.

Abstract

This is interesting machine learning project idea for Data Scientists/Machine Learning engineers working or planning to work with finance domain. Stock prices predictor is a system that learns about the performance of a company and predicts future stock prices. The challenges associated in working with stock prices data is that it is very granular, and moreover there are different types of data like volatility indices, prices, global macroeconomic indicators, fundamental indicators , and more. One good thing about working with stock market data is that the financial markets have shorter feedback cycles making it easier for data experts to validate their predictions on new data. There are different time series forecasting methods to forecast stock price, demand etc.

Time Series Data

Time series data is data that is collected at different points in time. This is opposed to cross-sectional data which observes individuals, companies, etc. at a single point in time. Because data points in time series are collected at adjacent time periods there is potential for correlation between observations. This is one of the features that distinguishes time series data from cross-sectional data.

Time series data is a collection of quantities that are assembled over even intervals in time and ordered chronologically. The time interval at which data is collection is generally referred to as the time series frequency.

Auto ARIMA

ARIMA is a very popular statistical method for time series forecasting. ARIMA models take into account the past values to predict the future values. There are three important parameters in ARIMA:

- p (past values used for forecasting the next value)
- q (past forecast errors used to predict the future values)
- d (order of differencing)

Dataset Used

I used the dataset of stock market data of Altaba Inc. The data shows the stock price of Altaba Inc from 1996–04–12 till 2017–11–10. The goal is to train an ARIMA model with optimal parameters that will forecast the closing price of the stocks on the test data.

Implementation

So start with loading all the required libraries:

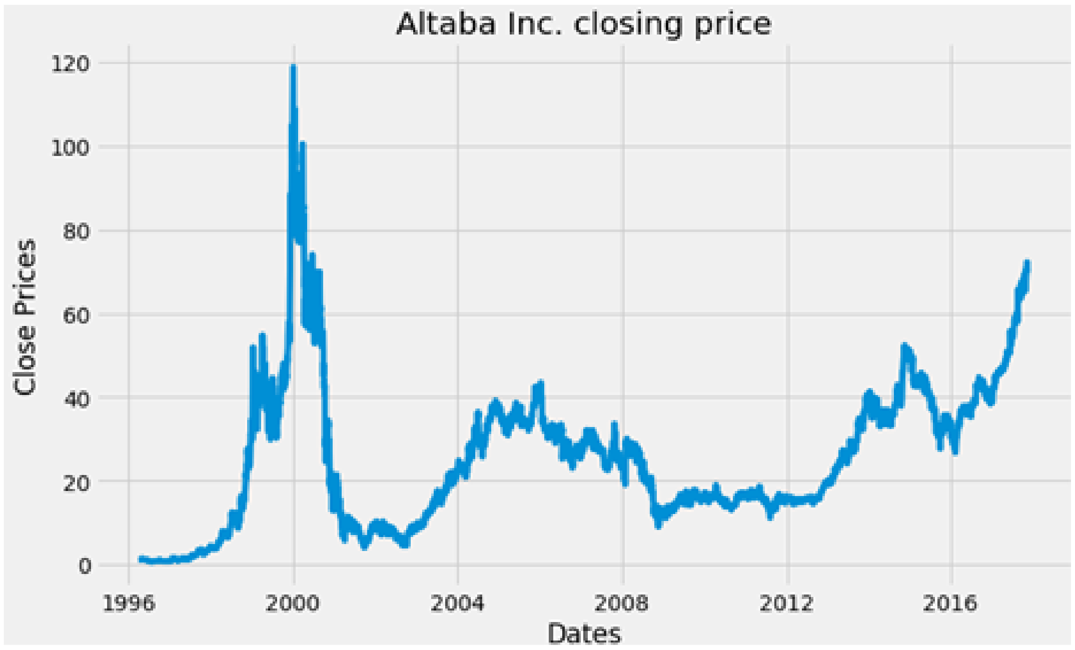
```
import os
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
from pylab import rcParams
rcParams['figure.figsize'] = 10, 6
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.arima_model import ARIMA
from pmdarima.arima import auto_arima
from sklearn.metrics import mean_squared_error, mean_absolute_error
import math
import numpy as np
```

Load the dataset:

```
dateparse = lambda dates: pd.datetime.strptime(dates, '%Y-%m-%d')

data = pd.read_csv('C:/Users/Sree Lakshmi/Desktop/data.txt',sep=',', index_col='Date',
parse_dates=['Date'], date_parser=dateparse).fillna(0)
```

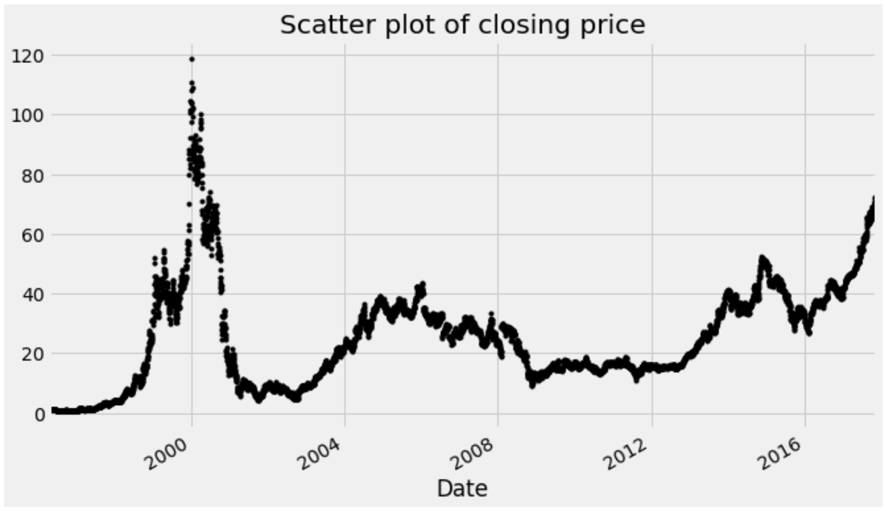
Visualize the per day closing price of the stock:

```
#plot close price

plt.figure(figsize=(10,6))

plt.grid(True)

plt.xlabel('Dates')

plt.ylabel('Close Prices')

plt.plot(data['Close'])

plt.title('Altaba Inc. closing price')

plt.show()
```

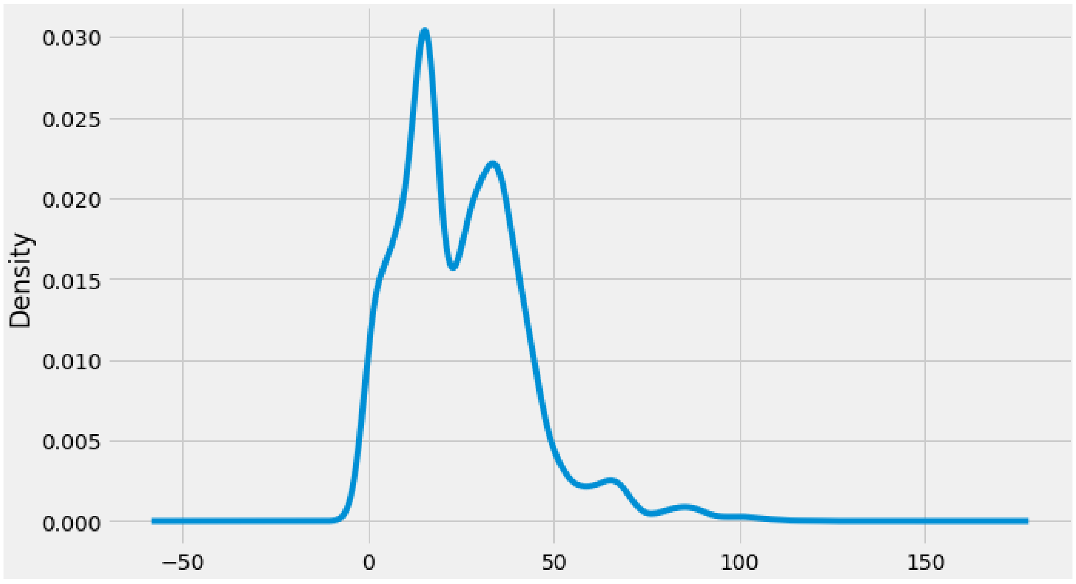


Lets us plot the scatterplot:

```
df_close = data['Close']

df_close.plot(style='k.')

plt.title('Scatter plot of closing price')

plt.show()
```



We can also visualize the data in our series through a probability distribution too:



Also, a given time series is thought to consist of three systematic components including level, trend, seasonality, and one non-systematic component called noise.

These components are defined as follows:

Level: The average value in the series.

Trend: The increasing or decreasing value in the series.

Seasonality: The repeating short-term cycle in the series.

Noise: The random variation in the series.

First, we need to check if a series is stationary or not because time series analysis only works with stationary data.

ADF (Augmented Dickey-Fuller) Test

The Dickey-Fuller test is one of the most popular statistical tests. It can be used to determine the presence of unit root in the series, and hence help us understand if the series is stationary or not. The null and alternate hypothesis of this test is:

Null Hypothesis: The series has a unit root (value of a =1)

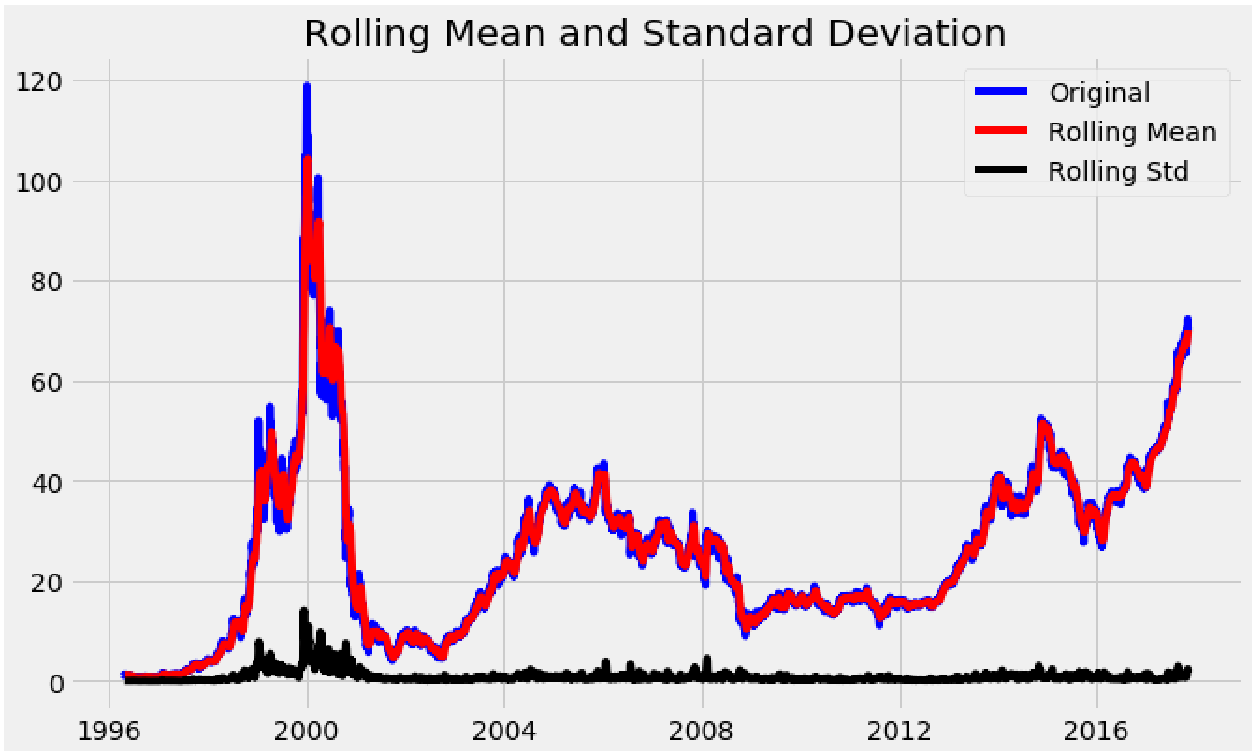
Alternate Hypothesis: The series has no unit root.

If we fail to reject the null hypothesis, we can say that the series is non-stationary. This means that the series can be linear or difference stationary.

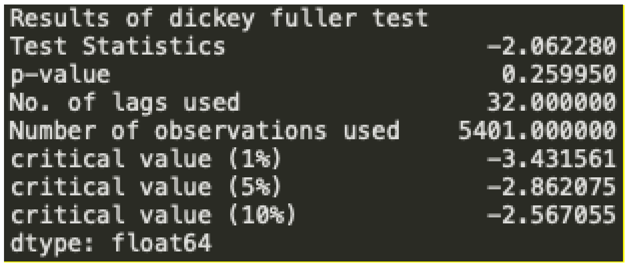
If both mean and standard deviation are flat lines (constant mean and constant variance), the series becomes stationary.

So let's check for stationarity:

```
#Test for staionarity
def test_stationarity(timeseries):
    #Determing rolling statistics
    rolmean = timeseries.rolling(12).mean()
    rolstd = timeseries.rolling(12).std()
    #Plot rolling statistics:
    plt.plot(timeseries, color='blue',label='Original')
    plt.plot(rolmean, color='red', label='Rolling Mean')
    plt.plot(rolstd, color='black', label = 'Rolling Std')
    plt.legend(loc='best')
    plt.title('Rolling Mean and Standard Deviation')
    plt.show(block=False)

    print("Results of dickey fuller test")
    adft = adfuller(timeseries,autolag='AIC')
    # output for dft will give us without defining what the values are.
    #hence we manually write what values does it explains using a for loop
    output = pd.Series(adft[0:4],index=['Test Statistics','p-value','No. of lags
used','Number of observations used'])
    for key,values in adft[4].items():
        output['critical value (%s)'%key] =  values
    print(output)
test_stationarity(df_close)
```



Through the above graph, we can see the increasing mean and standard deviation and hence our series is not stationary.



```
Results of dickey fuller test
Test Statistics                 -2.062280
p-value                          0.259950
No. of lags used                32.000000
Number of observations used   5401.000000
critical value (1%)             -3.431561
critical value (5%)             -2.862075
critical value (10%)            -2.567055
dtype: float64
```

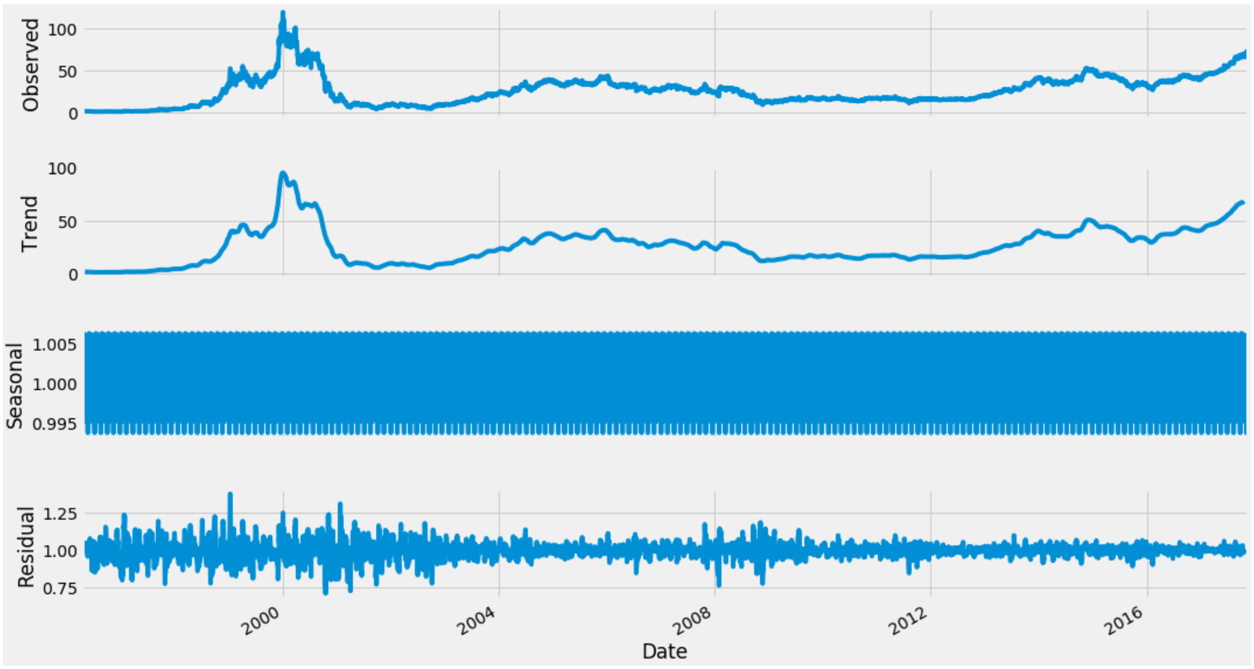
Results of Dickey-fuller test

We see that the p-value is greater than 0.05 so we cannot reject the Null hypothesis. Also, the test statistics is greater than the critical values. so the data is non-stationary.

In order to perform a time series analysis, we may need to separate seasonality and trend from our series. The resultant series will become stationary through this process.

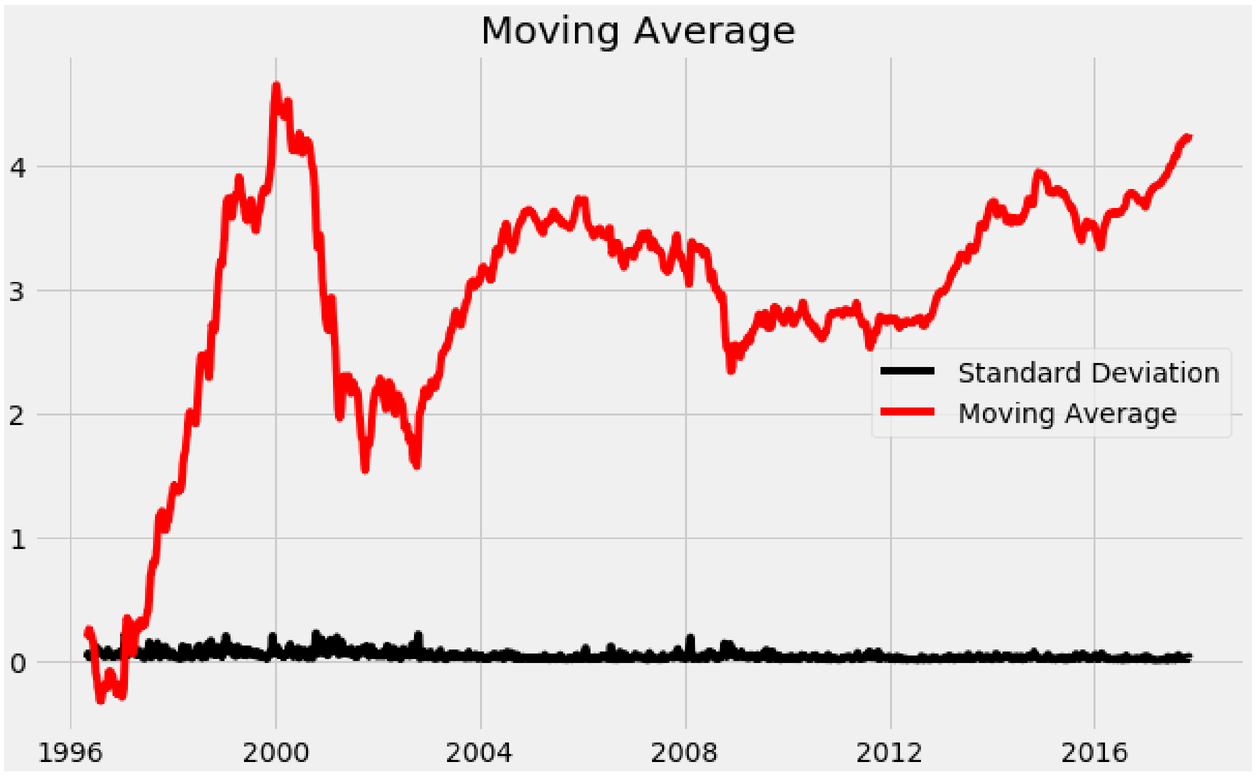
So let us separate Trend and Seasonality from the time series.

```
result = seasonal_decompose(df_close, model='multiplicative', freq = 30)

fig = plt.figure()

fig = result.plot()

fig.set_size_inches(16, 9)
```



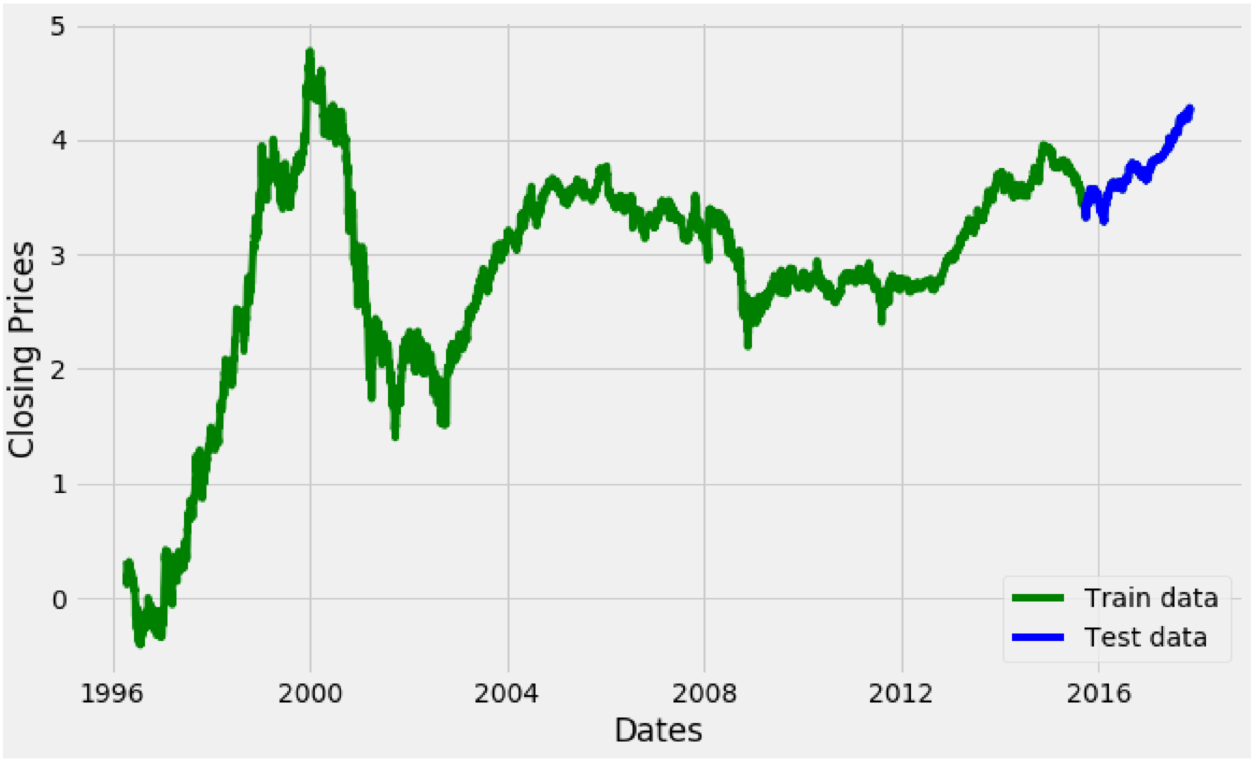
we start by taking a log of the series to reduce the magnitude of the values and reduce the rising trend in the series. Then after getting the log of the series, we find the rolling average of the series. A rolling average is calculated by taking input for the past 12 months and giving a mean consumption value at every point further ahead in series.

```
from pylab import rcParams

rcParams['figure.figsize'] = 10, 6

df_log = np.log(df_close)

moving_avg = df_log.rolling(12).mean()

std_dev = df_log.rolling(12).std()

plt.legend(loc='best')

plt.title('Moving Average')

plt.plot(std_dev, color ="black", label = "Standard Deviation")

plt.plot(moving_avg, color="red", label = "Mean")

plt.legend()

plt.show()
```



Now we are going to create an ARIMA model and will train it with the closing price of the stock on the train data. So let us split the data into training and test set and visualize it.

```
#split data into train and training set

train_data, test_data = df_log[3:int(len(df_log)*0.9)], df_log[int(len(df_log)*0.9):]

plt.figure(figsize=(10,6))

plt.grid(True)

plt.xlabel('Dates')

plt.ylabel('Closing Prices')

plt.plot(df_log, 'green', label='Train data')

plt.plot(test_data, 'blue', label='Test data')

plt.legend()
```



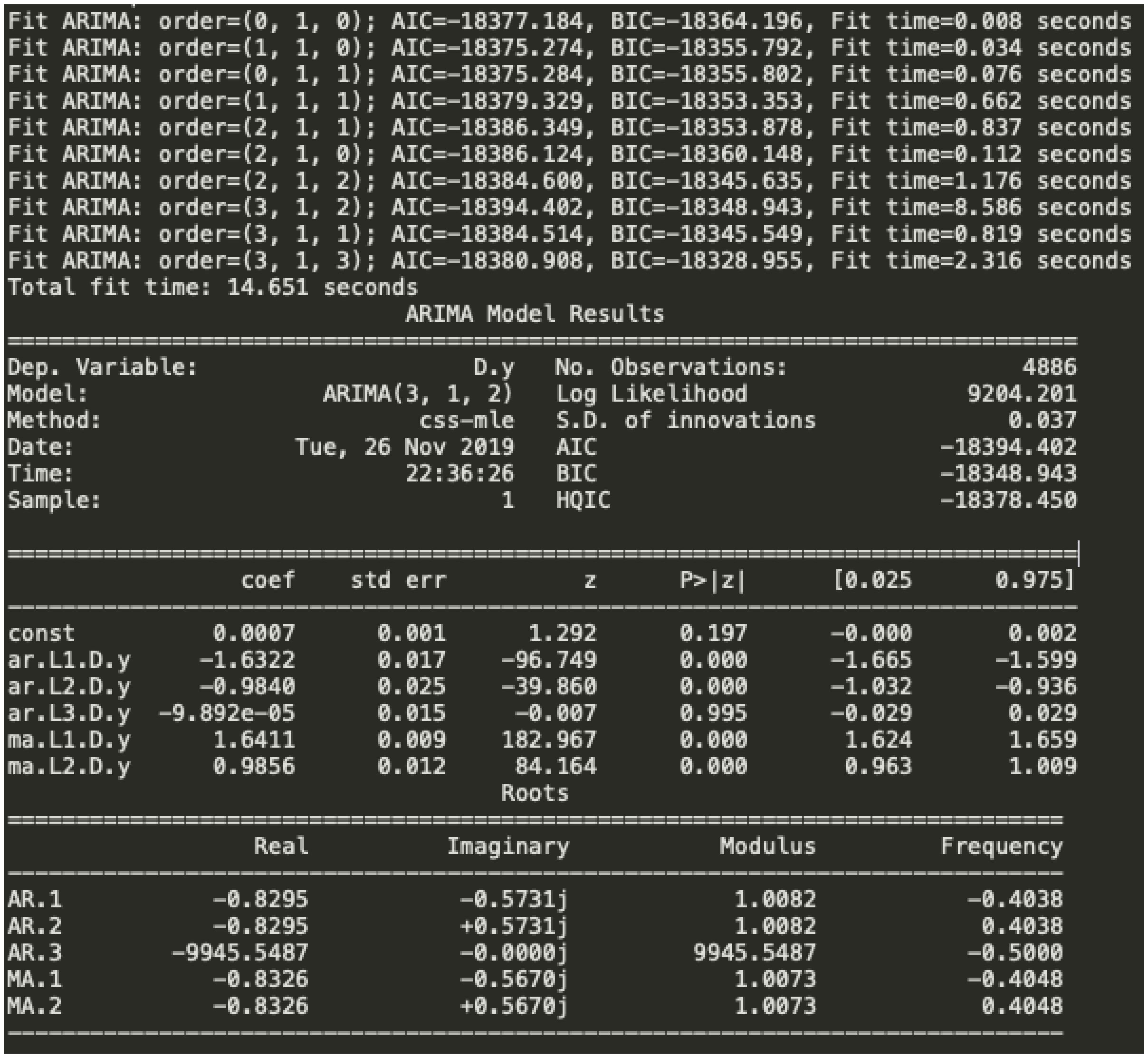
Its time to choose parameters p,q,d for ARIMA model. Last time we chose the value of p,d, and q by observing the plots of ACF and PACF but now we are going to use Auto ARIMA to get the best parameters without even plotting ACF and PACF graphs.

Auto ARIMA: Automatically discover the optimal order for an ARIMA model.

The auto_arima function seeks to identify the most optimal parameters for an ARIMA model, and returns a fitted ARIMA model. This function is based on the commonly-used R function, forecast::auto.arima.

The auro_arima function works by conducting differencing tests (i.e., Kwiatkowski–Phillips–Schmidt–Shin, Augmented Dickey-Fuller or Phillips–Perron) to determine the order of differencing, d, and then fitting models within ranges of defined start_p, max_p, start_q, max_q ranges. If the seasonal optional is enabled, auto_arima also seeks to identify the optimal P and Q hyper- parameters after conducting the Canova-Hansen to determine the optimal order of season differencing, D.

```
model_autoARIMA = auto_arima(train_data, start_p=0, start_q=0,
                      test='adf',       # use adftest to find           optimal 'd'
                      max_p=3, max_q=3, # maximum p and q
                      m=1,              # frequency of series
                      d=None,           # let model determine 'd'
                      seasonal=False,   # No Seasonality
                      start_P=0,
                      D=0,
                      trace=True,
                      error_action='ignore',
                      suppress_warnings=True,
                      stepwise=True)
print(model_autoARIMA.summary())
```

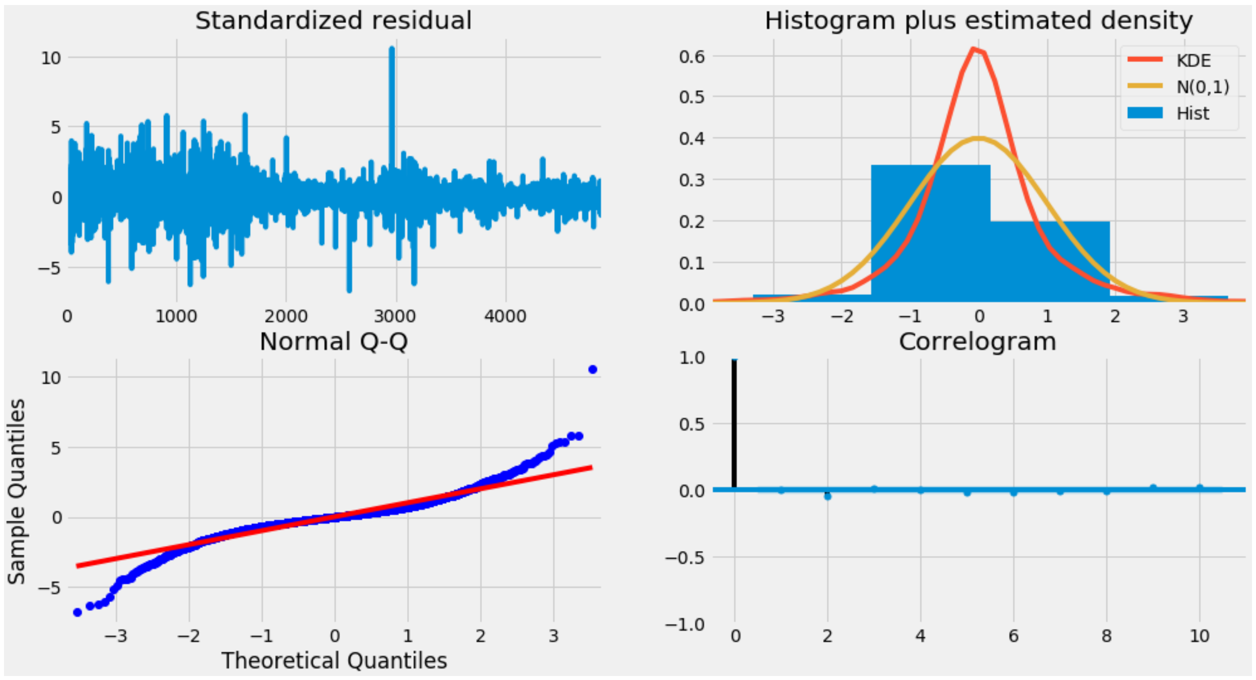


```
Fit ARIMA: order=(0, 1, 0); AIC=-18377.184, BIC=-18364.196, Fit time=0.008 seconds
Fit ARIMA: order=(1, 1, 0); AIC=-18375.274, BIC=-18355.792, Fit time=0.034 seconds
Fit ARIMA: order=(0, 1, 1); AIC=-18375.284, BIC=-18355.802, Fit time=0.076 seconds
Fit ARIMA: order=(1, 1, 1); AIC=-18379.329, BIC=-18353.353, Fit time=0.662 seconds
Fit ARIMA: order=(2, 1, 1); AIC=-18386.349, BIC=-18353.878, Fit time=0.837 seconds
Fit ARIMA: order=(2, 1, 0); AIC=-18386.124, BIC=-18360.148, Fit time=0.112 seconds
Fit ARIMA: order=(2, 1, 2); AIC=-18384.600, BIC=-18345.635, Fit time=1.176 seconds
Fit ARIMA: order=(3, 1, 2); AIC=-18394.402, BIC=-18348.943, Fit time=8.586 seconds
Fit ARIMA: order=(3, 1, 1); AIC=-18384.514, BIC=-18345.549, Fit time=0.819 seconds
Fit ARIMA: order=(3, 1, 3); AIC=-18380.908, BIC=-18328.955, Fit time=2.316 seconds
Total fit time: 14.651 seconds
                             ARIMA Model Results
==============================================================================
Dep. Variable:                    D.y   No. Observations:                 4886
Model:                 ARIMA(3, 1, 2)   Log Likelihood                9204.201
Method:                       css-mle   S.D. of innovations              0.037
Date:                Tue, 26 Nov 2019   AIC                         -18394.402
Time:                        22:36:26   BIC                         -18348.943
Sample:                             1   HQIC                        -18378.450

==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0007      0.001      1.292      0.197      -0.000       0.002
ar.L1.D.y     -1.6322      0.017    -96.749      0.000      -1.665      -1.599
ar.L2.D.y     -0.9840      0.025    -39.860      0.000      -1.032      -0.936
ar.L3.D.y  -9.892e-05      0.015     -0.007      0.995      -0.029       0.029
ma.L1.D.y      1.6411      0.009    182.967      0.000       1.624       1.659
ma.L2.D.y      0.9856      0.012     84.164      0.000       0.963       1.009
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1           -0.8295           -0.5731j            1.0082           -0.4038
AR.2           -0.8295           +0.5731j            1.0082            0.4038
AR.3        -9945.5487           -0.0000j         9945.5487           -0.5000
MA.1           -0.8326           -0.5670j            1.0073           -0.4048
MA.2           -0.8326           +0.5670j            1.0073            0.4048
-----------------------------------------------------------------------------
```

So the Auto ARIMA model provided the value of p,d, and q as 3,1 and 2 respectively.

Before moving forward, let’s review the residual plots from auto ARIMA.

```
model_autoARIMA.plot_diagnostics(figsize=(15,8))

plt.show()
```



So how to interpret the plot diagnostics?

Top left: The residual errors seem to fluctuate around a mean of zero and have a uniform variance.

Top Right: The density plot suggest normal distribution with mean zero.

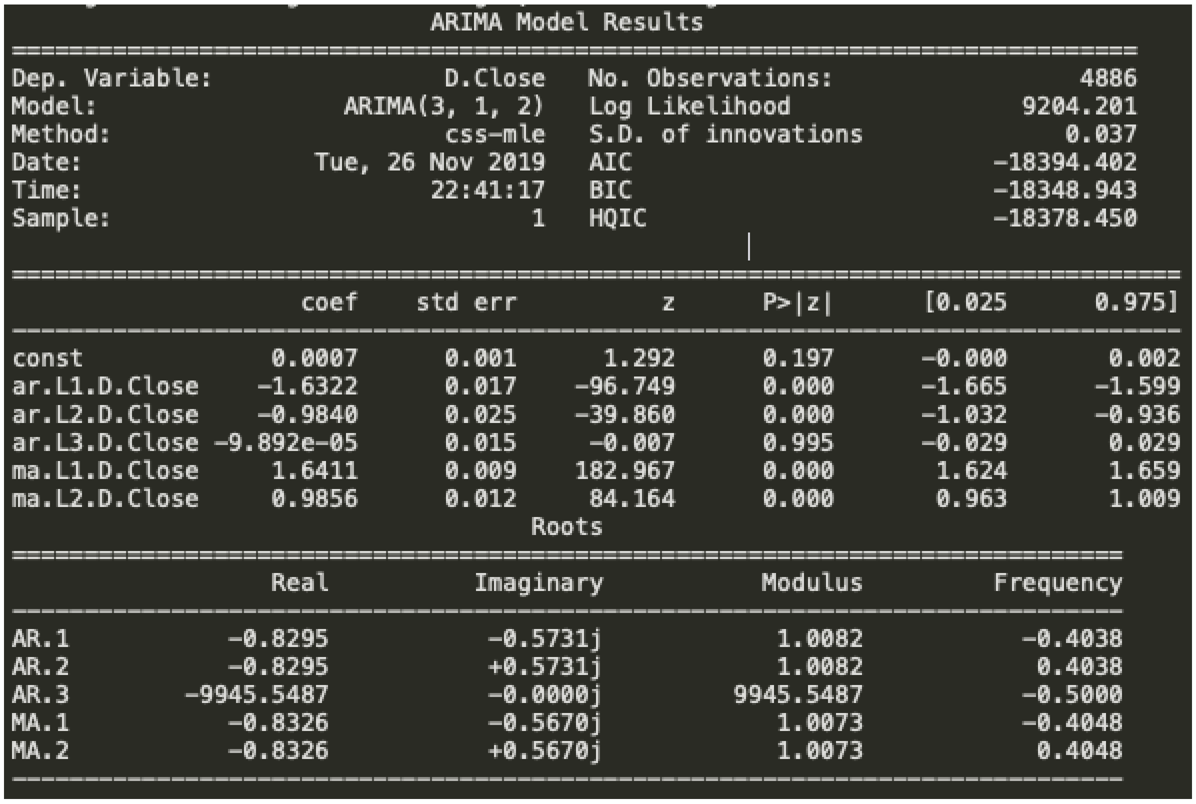
Bottom left: All the dots should fall perfectly in line with the red line. Any significant deviations would imply the distribution is skewed.

Bottom Right: The Correlogram, aka, ACF plot shows the residual errors are not autocorrelated. Any autocorrelation would imply that there is some pattern in the residual errors which are not explained in the model. So you will need to look for more X's (predictors) to the model.

Overall, it seems to be a good fit. Let's start forecasting the stock prices.

Next, create an ARIMA model with provided optimal parameters p, d and q.

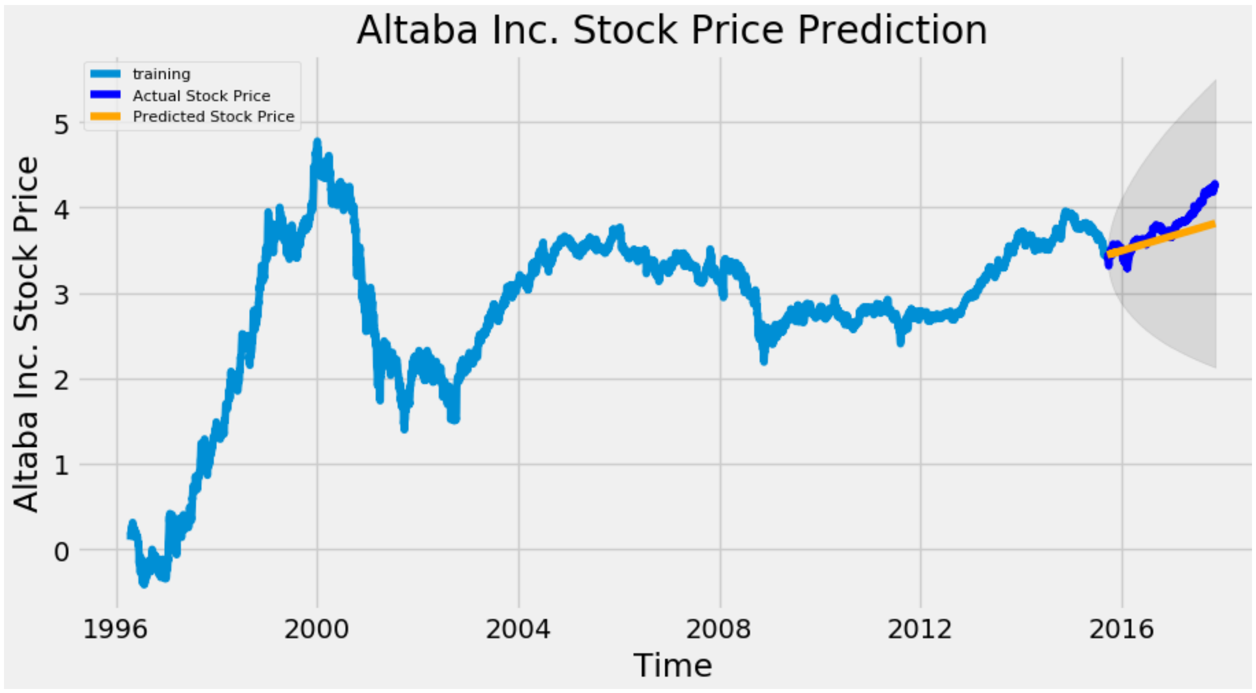
```
model = ARIMA(train_data, order=(3, 1, 2))

fitted = model.fit(disp=-1)

print(fitted.summary())
```



```
                              ARIMA Model Results
==============================================================================
Dep. Variable:                D.Close   No. Observations:                 4886
Model:                 ARIMA(3, 1, 2)   Log Likelihood                9204.201
Method:                       css-mle   S.D. of innovations              0.037
Date:                Tue, 26 Nov 2019   AIC                         -18394.402
Time:                        22:41:17   BIC                         -18348.943
Sample:                             1   HQIC                        -18378.450

=================================================================================
                    coef    std err          z      P>|z|      [0.025      0.975]
---------------------------------------------------------------------------------
const             0.0007      0.001      1.292      0.197      -0.000       0.002
ar.L1.D.Close    -1.6322      0.017    -96.749      0.000      -1.665      -1.599
ar.L2.D.Close    -0.9840      0.025    -39.860      0.000      -1.032      -0.936
ar.L3.D.Close -9.892e-05      0.015     -0.007      0.995      -0.029       0.029
ma.L1.D.Close     1.6411      0.009    182.967      0.000       1.624       1.659
ma.L2.D.Close     0.9856      0.012     84.164      0.000       0.963       1.009
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1           -0.8295           -0.5731j            1.0082           -0.4038
AR.2           -0.8295           +0.5731j            1.0082            0.4038
AR.3        -9945.5487           -0.0000j         9945.5487           -0.5000
MA.1           -0.8326           -0.5670j            1.0073           -0.4048
MA.2           -0.8326           +0.5670j            1.0073            0.4048
-----------------------------------------------------------------------------
```

Now let's start forecast the stock prices on the test dataset keeping 95% confidence level.

```
# Forecast
fc, se, conf = fitted.forecast(544, alpha=0.05)  # 95% confidence
fc_series = pd.Series(fc, index=test_data.index)
lower_series = pd.Series(conf[:, 0], index=test_data.index)
upper_series = pd.Series(conf[:, 1], index=test_data.index)
plt.figure(figsize=(12,5), dpi=100)
plt.plot(train_data, label='training')
plt.plot(test_data, color = 'blue', label='Actual Stock Price')
plt.plot(fc_series, color = 'orange',label='Predicted Stock Price')
plt.fill_between(lower_series.index, lower_series, upper_series,
                 color='k', alpha=.10)
plt.title('Altaba Inc. Stock Price Prediction')
plt.xlabel('Time')
plt.ylabel('Actual Stock Price')
plt.legend(loc='upper left', fontsize=8)
plt.show()
```



As you can see our model did quite handsomely. Let us also check the commonly used accuracy metrics to judge forecast results:

```
# report performance

mse = mean_squared_error(test_data, fc)

print('MSE: '+str(mse))

mae = mean_absolute_error(test_data, fc)

print('MAE: '+str(mae))

rmse = math.sqrt(mean_squared_error(test_data, fc))

print('RMSE: '+str(rmse))

mape = np.mean(np.abs(fc - test_data)/np.abs(test_data))

print('MAPE: '+str(mape))
```

Output:

```
MSE: 0.03330921053066402
MAE: 0.13801238336759786
RMSE: 0.18250811086267923
MAPE: 0.035328833278944705
```

Around 3.5% MAPE (Mean Absolute Percentage Error) implies the model is about 96.5% accurate in predicting the test set observations.

ARIMA MODEL SUCCESSFULLY BUILD

Conclusion

- From the end we had analyzed all sort of case study and had made a deep report made with the time series, ARIMA model and also examined with the help of an example
- The historical data from the year 1996 to 2017 were taken in to account for analysis. The ARIMA model is trained and predicted the stock prices on the test dataset.

ARIMA MODEL
Time Series Analysis
& Forecasting

END OF REPORT THANK YOU