MINI PROJECT REPORT

On

STOCK PRICE PREDICTOR

Submitted by

ARYAVRAT
Roll No: 1615000133
RAKHI AGARWAL
Roll No: 161500436

ASHISH VARSHNEY
Roll No: 161500143
RAGHAV ARORA
Roll No: 1615000421

To

Mr. Vivek Kumar
Astt.Professor

Department of Computer Engineering & Applications

Institute of Engineering & Technology

GLA University
Mathura- 281406, INDIA
April, 2019

Department of Computer Engineering and Applications
GLA University, Mathura
17 km. Stone NH#2, Mathura-Delhi Road, P.O. – Chaumuha,
Mathura – 281406

Declaration

We hereby declare that the work which is being presented in the Mini Project Titled: ***"Stock Price Predictor",*** *in partial fulfillment of the requirements for Mini-Project LAB, is an authentic record of our own work carried under the supervision of* ***Mr. Vivek Kumar, Astt.Professor, GLA University, Mathura****.*

Aryavrat

Ashish Varshney

Ralkhi Agarwal

Raghav Arora

Department of Computer Engineering and Applications
GLA University, Mathura
17 km. Stone NH#2, Mathura-Delhi Road, P.O. – Chaumuha,
Mathura – 281406

CERTIFICATE

*This is to certify that the project entitled **"Stock Price Predictor"** carried out in Mini Project – II Lab is a bonafide work done by **Aryavrat (161500133), Ashish Varshney (161500143), Rakhi Agarwal (161500436) and Raghav Arora (161500421)** and is submitted in partial fulfillment of the requirements for the award of the degree Bachelor of Technology (Computer Science & Engineering).*

Signature of Supervisor:

Name of Supervisor: Mr. Vivek Kumar

Date:

ACKNOWLEDGEMENT

It gives us a great sense of pleasure to present the report of the B. Tech Mini Project undertaken during B. Tech. Third Year. This project in itself is an acknowledgement to the inspiration, drive and technical assistance contributed to it by many individuals. This project would never have seen the light of the day without the help and guidance that we have received.

Our heartiest thanks to ***Dr. (Prof). Anand Singh Jalal,*** *Head of Dept., Department of CEA for providing us with an encouraging platform to develop this project, which thus helped us in shaping our abilities towards a constructive goal.*

We owe special debt of gratitude to ***Mr. Vivek Kumar,*** *Assistant Professor Department of CEA, for his constant support and guidance throughout the course of our work. His sincerity, thoroughness and perseverance have been a constant source of inspiration for us. He has showered us with all his extensively experienced ideas and insightful comments at virtually all stages of the project & has also taught us about the latest industry-oriented technologies.*

We also do not like to miss the opportunity to acknowledge the contribution of all faculty members of the department for their kind guidance and cooperation during the development of our project. Last but not the least, we acknowledge our friends for their contribution in the completion of the project.

Aryavrat
AshishVarshney
Rakhi Agarwal
Raghav Arora

ABSTRACT

Stock Price Predictor is a Machine learning based project which uses the supervised learning .It provides the facility of prediction the future stock return based on the past return. It also provides the graphical representation of different models.

It uses various libraries of machine learning like pandas, numpy, matplotlib, statsmodels etc. and for the datasets we used the dataset from Google Finance, Qandl.

It not only gives the flexibility of predicting future stock return from past returns but also uses different model of supervised learning to reduce errors.

TABLE of CONTENTS

CHAPTER 1

INTRODUCTION

This section gives a scope of description and overview of everything included in this project report. Also it includes the system overview with goal and vision.

1.1 PURPOSE

The purpose of this document is to give detailed information regarding this Mini Project labeled “Stock Price Predictor” .It will illustrate the complete declaration of the project developed. In addition to this, it gives detailed description of the implementation of the system along with system constraints. This document is primarily intended to give an overview to anyone of how this application works and is beneficial to corporate sector.

1.2 SYSTEM OVERVIEW

Stock Price Predictor is a Machine Learning based project which displays the user RMSE (root mean square error) value between the predicted values of the stock return and the actual stock return and also displays the graphical representation of different predicting models and other process that are applied on the datasets like normalization, decomposition etc.

1.3 MOTIVATION

Nowadays, as the connections between worldwide economies are tightened by globalization, external perturbations to the financial markets are no longer domestic. With evolving capital markets, more and more data is being created daily.

The intrinsic value of a company’s stock is the value determined by estimating the expected future cash flows of a stock and discounting them to the present, which is known as the book value. This is distinct from the market value of the stock, that is determined by the company’s stock price. This market value of a stock can deviate from the intrinsic value due to reasons unrelated to the company’s fundamental operations, such as market sentiment.

The fluctuation of stock market is violent and there are many complicated financial indicators. Only few people with extensive experience and knowledge can understand the meaning of the indicators and use them to make good prediction to get fortune. Most people have to rely solely on luck to earn money from stock trading. However, the advancement in technology, provides an opportunity to gain steady fortune from stock market and also can help experts to find out the most informative indicators to make better prediction. The prediction of the market value is of paramount importance to help in maximizing the profit of stock option purchase while keeping the risk low.

CHAPTER 2

SOFTWARE REQUIREMENTS & ANALYSIS

2.1 PROBLEM STATEMENT

The aim of the project is to examine a number of different forecasting techniques to predict future stock returns based on past returns. We do this by applying supervised learning methods for stock price forecasting by interpreting the seemingly chaotic market data.

2.2 Techniques and their Functionalities

Moving Average

The moving average (MA) is a simple technical analysis tool that smoothen out price data by creating a constantly updated average price. The average is taken over a specific period of time, like 10 days, 20 minutes, 30 weeks or any time period the trader chooses. There are advantages to using a moving average in your trading, as well as options on what type of moving average to use. Moving average strategies are also popular and can be tailored to any time frame, suiting both long-term investors and short-term traders.

A moving average helps cut down the amount of "noise" on a price chart. Look at the direction of the moving average to get a basic idea of which way the price is moving. If it is angled up, the price is moving up (or was recently) overall; angled down, and the price is moving down overall; moving sideways, and the price is likely in a range.

2.2.1.1 Simple Moving Average Calculation

The simple moving average (SMA) calculates an average of the last **n** prices, where **n** represents the number of periods for which you want the average:

Simple moving average = (P1 + P2 + P3 + P4 + ... + Pn) / n

2.2.1.2 Exponential Weighted Moving Average

Exponential Moving Average Calculation

The exponential moving average (EMA) is a weighted average of the last **n** prices, where the weighting decreases exponentially with each previous price/period. In other words, the formula gives recent prices more weight than past prices.

Exponential moving average = [Close - previous EMA] * (2 / n+1) + previous EMA

$$EMA_{\text{today}}=[\text{Valuetoday}\times((s)/(1+d))]+EMA_{\text{yesterday}}\times[1-((s)/(+d))]$$

where:

s=smoothing

d=number of days

Weighted Moving Average Calculation

weighted moving average (WMA) gives you a weighted average of the last **n** prices, where the weighting decreases with each previous price. This works similarly to the EMA, but you calculate the WMA differently.

Weighted moving average calculation = (Price * weighting factor) + (Price previous period * weighting factor-1)...

2.2.2 Linear regression

Linear regression analyzes two separate variables in order to define a single relationship. In chart analysis, this refers to the variables of price and time. Investors and traders who use charts recognize the ups and downs of price printed horizontally from day-to-day, minute-to-minute or week-to-week, depending on the evaluated time frame. The different market approaches are what make linear regression analysis so attractive.

The equation for linear regression can be written as:

$$Y = \theta_1 X_1 + \theta_2 X_2 + \ldots \theta_n X_n$$

Here, x_1, x_2,….x_n represent the independent variables while the coefficients θ_1, θ_2, …. θ_n represent the weights.

2.2.3 ARIMA MODEL

ARIMA stands for Autoregressive Integrated Moving Average. ARIMA is also known as Box-Jenkins approach. Box and Jenkins claimed that non-stationary data can be made stationary by differencing the series, Y_t. The general model for Y_t is written as,

$$\mathbf{Y_t = \phi_1 Y_{t-1} + \phi_2 Y_{t-2} \ldots \phi_p Y_{t-p} + \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \ldots \theta_q \epsilon_{t-q}}$$

Where Y_t is the differenced time series value, ϕ and θ are unknown parameters and ϵ are independent identically distributed error terms with zero mean. Here, Y_t is expressed in terms of its past values and the current and past values of error terms.

The ARIMA model combines three basic methods:

- Auto Regression (AR) – In auto-regression, the values of a given time series data are regressed on their own lagged values, which is indicated by the “p” value in the ARIMA model.
- Differencing (I-for Integrated) – This involves differencing the time series data to remove the trend and convert a non-stationary time series to a stationary one. This is indicated by the “d” value in the ARIMA model. If d = 1, it looks at the difference between two-time series entries, if d = 2 it looks at the differences of the differences obtained at d =1, and so forth.
- Moving Average (MA) – The moving average nature of the ARIMA model is represented by the “q” value which is the number of lagged values of the error term.

This model is called Autoregressive Integrated Moving Average or ARIMA(p,d,q) of Y_t. We will follow the steps enumerated below to build our model.

Step 1: Testing and Ensuring Stationarity

To model a time series with the Box-Jenkins approach, the series has to be stationary. A stationary time series means a time series without trend, one having a constant mean and variance over time, which makes it easy for predicting values.

Testing for stationarity – We test for stationarity using the Augmented Dickey-Fuller unit root test. The p-value resulting from the ADF test has to be less than 0.05 or 5% for a time series to be stationary. If the p-value is greater than 0.05 or 5%, you conclude that the time series has a unit root which means that it is a non-stationary process.

Differencing – To convert a non-stationary process to a stationary process, we apply the differencing method. Differencing a time series means finding the differences between consecutive values of a time series data. The differenced values form a new time series dataset which can be tested to uncover new correlations or other interesting statistical properties.

We can apply the differencing method consecutively more than once, giving rise to the "first differences", "second order differences", etc.

We apply the appropriate differencing order (d) to make a time series stationary before we can proceed to the next step.

Step 2: Identification of p and q

In this step, we identify the appropriate order of Autoregressive (AR) and Moving average (MA) processes by using the Autocorrelation function (ACF) and Partial Autocorrelation function (PACF).

Identifying the p order of AR model

For AR models, the ACF will dampen exponentially and the PACF will be used to identify the order (p) of the AR model. If we have one significant spike at lag 1 on the PACF, then we have an AR model of the order 1, i.e. AR(1). If we have significant spikes at lag 1, 2, and 3 on the PACF, then we have an AR model of the order 3, i.e. AR(3).

Identifying the q order of MA model

For MA models, the PACF will dampen exponentially and the ACF plot will be used to identify the order of the MA process. If we have one significant spike at lag 1 on the ACF, then we have an MA model of the order 1, i.e. MA(1). If we have significant spikes at lag 1, 2, and 3 on the ACF, then we have an MA model of the order 3, i.e. MA(3).

Step 3: Estimation and Forecasting

Once we have determined the parameters (p,d,q) we estimate the accuracy of the ARIMA model on a training data set and then use the fitted model to forecast the values of the test data set using a forecasting function. In the end, we cross-check whether our forecasted values are in line with the actual values.

2.3 Hardware Requirements:

- multicore processor(i5-i7)
- 8GB RAM(minimum)

2.4 Software Requirements

- Windows 10, Linux or IOS
- Anaconda
- Jupyter Notebook
- Libraries: matplotlib, pandas, numpy, scikit-learn, sk-learn, statsmodels.

2.5 Specific Requirements

- The user must use a valid file format for dataset.
- Internet connection is mandatory.

CHAPTER 3

PROJECT DESIGN

The project design includes DFDs , UML diagrams which consists of Use case diagrams , Database including tables details and sequence diagrams.

3.1 DFD Data Flow Diagrams

Data flow diagram has one end user who provides the data set to the project and observe the result from the project.

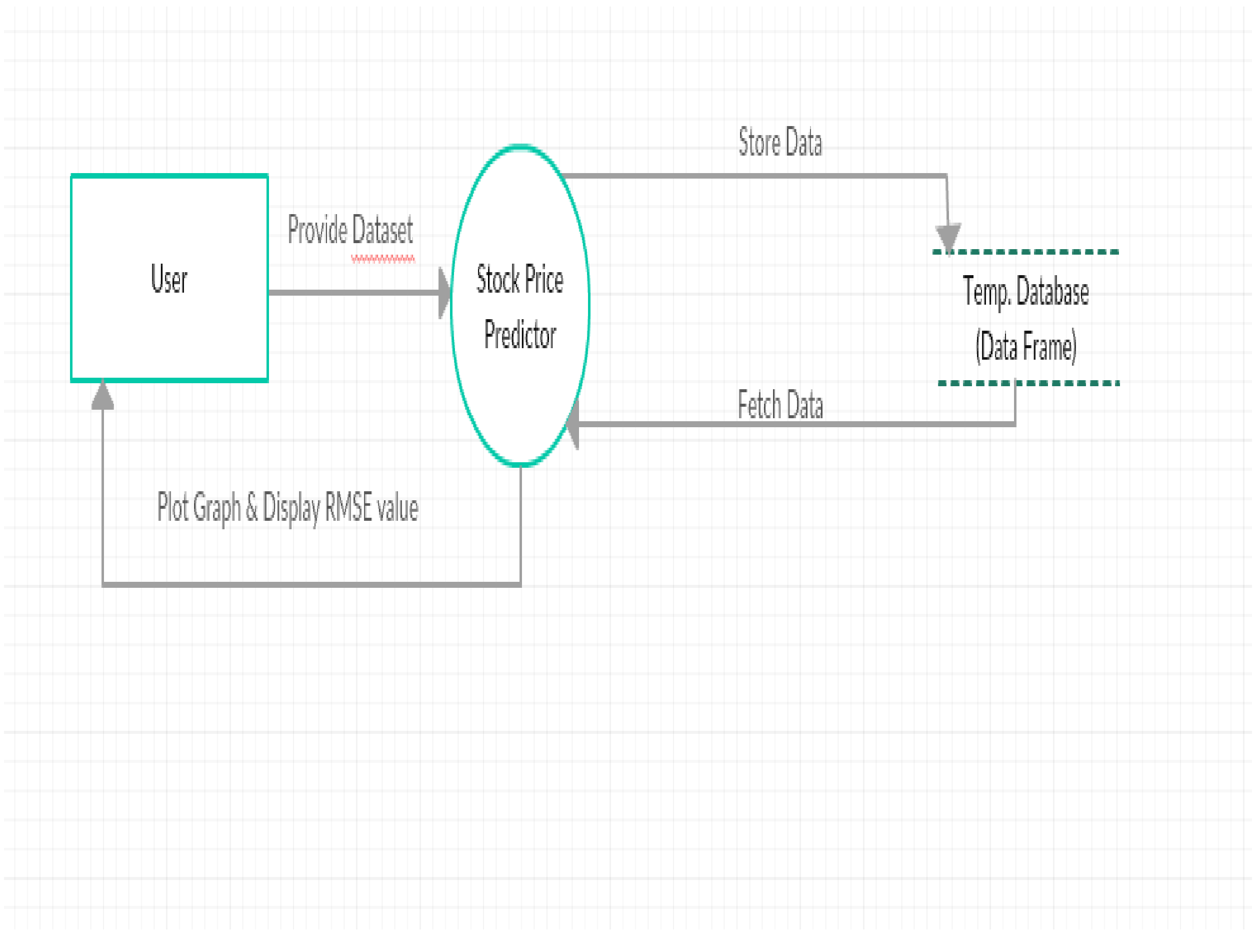


Fig 3.1 DFD Level zero

3.2 Use Case Diagrams

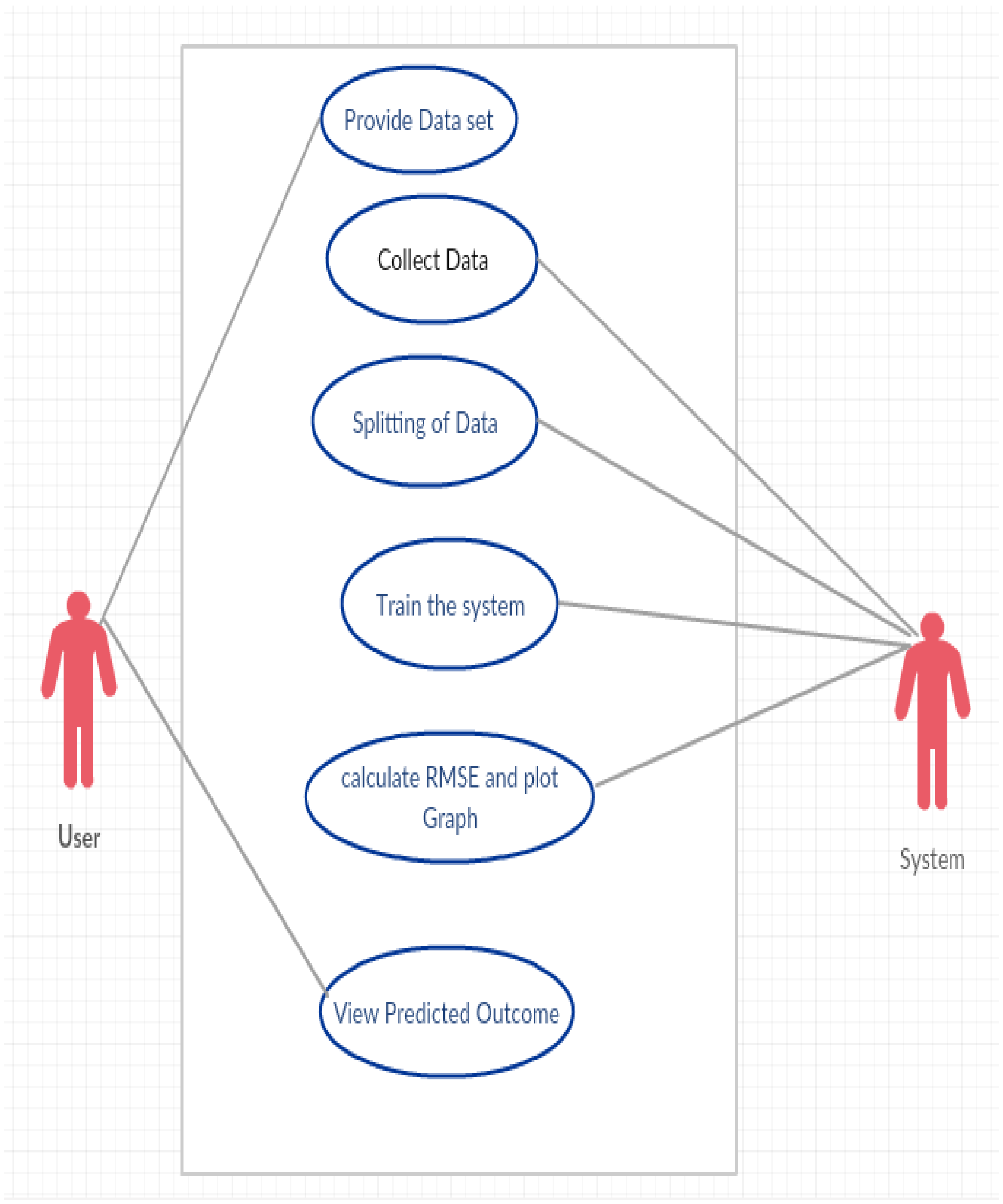


Fig 3.2 Use Case Diagram

3.3Sequence Diagram

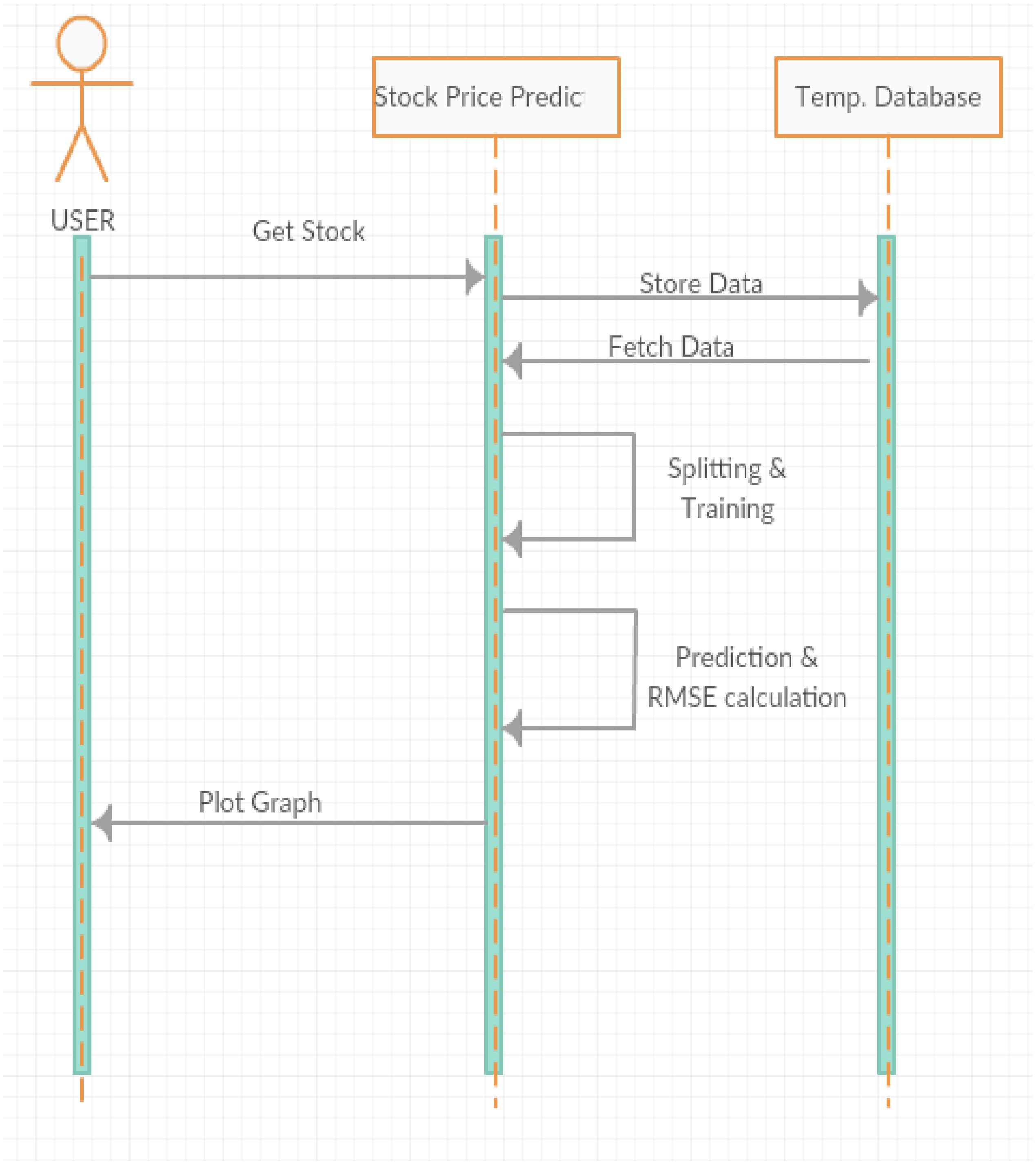


Fig 3.3 Sequence Diagram

3.4 Dataset Information

TABLE 1 (Only for Moving Average)

This table is for the data set used to implement the moving average model.

Name of Field	Data type	Explanation
Date	Date	Date of the current day.
Open	float	Starting price of the day.
Low	float	Minimum share of the day.
High	float	Maximum share of the day.
Last	float	Last share of the day.
Close	float	Closing price of the day.
Total Trade Quantity	int	Number of shares bought or sold in the day and
Turnover (Lacs)	float	The turnover of the particular company on a given day/date.

Fig 3.4 (a) Data field information of the dataset 1(TATA GLOBAL BEVERAGES)

TABLE 2 (only for final script M.A., E.W.M.A., ARIMA)

This table is for our final code which consist of final different techniques.

Name of Field	Data type	Explanation
Date	Date	Date of the current day.
Open	float	Starting price of the day.
Low	float	Minimum share of the day.
High	float	Maximum share of the day.
Last	float	Last share of the day.
Close	float	Closing price of the day.
volume	int	Number of shares bought or sold in the day and
adj_close	float	-
prev_day_diff	float	-
50_day_moving_avg	float	-
10_day_volatility	float	-
s&p_index_open	float	-

s&p_index_low	float	-
s&p_index_close	float	-
s&p_index_close	float	-
s&p_index_adj_close	float	-

Fig 3.4 (b)   Data field information of the dataset 1(GOOGLE Finance)

CHAPTER 4

LIBRARIES

Different libraries are used to implement this project brief discussion of all libraries used in this project is given below:

4.1 Numpy

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

4.1.1 Simple financial functions

Function	Description
fv(rate, nper, pmt, pv[, when])	Compute the future value.
pv(rate, nper, pmt[, fv, when])	Compute the present value.
npv(rate, values)	Return the NET PRESENT VALUE of a cash flow series.
pmt(rate, nper, pv[, fv, when])	Compute the payment against loan principle plus interest .

ppmt(rate, per, nper, pv[, fv, when])	Compute the payment against loan principle.
ipmt(rate, per, nper, pv[, fv, when])	Compute the interest portion of a payment.
irr(values)	Return the Internal Rate of Return.
mirr(values, finance_rate, reinvest_rate)	Modified internal rate of return,
nper(rate, pmt, pv[, fv, when])	Compue the number of periodic payments.
rate(nper, pmt, pv, fv[, when, guess, tol, …])	Compare the rate of intrest per period.

4.2 pandas

pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language. Highlights ogf the pandas libraries are given below:

- A fast and efficient DataFrame object for data manipulation with integrated indexing;
- Tools for reading and writing data between in-memory data structures and different formats: CSV and text files, Microsoft Excel, SQL databases, and the fast HDF5 format;
- Intelligent data alignment and integrated handling of missing data: gain automatic label-based alignment in computations and easily manipulate messy data into an orderly form;
- Flexible reshaping and pivoting of data sets;
- Intelligent label-based slicing, fancy indexing, and subsetting of large data sets;
- Columns can be inserted and deleted from data structures for size mutability;
- Aggregating or transforming data with a powerful group by engine allowing split-apply-combine operations on data sets;
- High performance merging and joining of data sets;
- Hierarchical axis indexing provides an intuitive way of working with high-dimensional data in a lower-dimensional data structure;
- Time series-functionality: date range generation and frequency conversion, moving window statistics, moving window linear regressions, date shifting and lagging. Even create domain-specific time offsets and join time series without losing data;
- Highly optimized for performance, with critical code paths written in Cython or C.

- Python with *pandas* is in use in a wide variety of academic and commercial domains, including Finance, Neuroscience, Economics, Statistics, Advertising, Web Analytics, and more.

4.3 matplotlib

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter notebook, web application servers, and four graphical user interface toolkits.

Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, error charts, scatterplots, etc.

4.4 scikit-learn

Scikit-learn (formerly scikits.learn) is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, *k*-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

Function	**Description**
base.clone(estimator[, safe])	Constructs a new estimator with the same parameters.
base.is_classifier(estimator)	Returns True if the given estimator is (probably) a classifier.
base.is_regressor(estimator)	Returns True if the given estimator is (probably) a regressor.
config_context(**new_config)	Context manager for global scikit-learn configuration
get_config()	Retrieve current values for configuration set by set_config

set_config([assume_finite, working_memory])	Set global scikit-learn configuration
show_versions()	Print useful debugging information

4.5 statsmodels

statsmodels is a Python module that provides classes and functions for the estimation of many different statistical models, as well as for conducting statistical tests, and statistical data exploration. An extensive list of result statistics are available for each estimator. The results are tested against existing statistical packages to ensure that they are correct.

CHAPTER 5

TESTING

5.1 Theory of Testing

Software Testing is evaluation of the software against requirements gathered from users and system specifications. Testing is conducted at the phase level in software development life cycle or at module level in program code.

Black-box testing

It is carried out to test functionality of the program. It is also called ‘Behavioral’ testing. The tester in this case, has a set of input values and respective desired results. On providing input, if the output matches with the desired results, the program is tested ‘ok’, and problematic otherwise.

White-box testing

It is conducted to test program and its implementation, in order to improve code efficiency or structure. It is also known as ‘Structural’ testing. In this testing method, the design and structure of the code are known to the tester. Programmers of the code conduct this test on the code.

Manual Vs Automated Testing

Testing can either be done manually or using an automated testing tool:

- **Manual** - This testing is performed without taking help of automated testing tools. The software tester prepares test cases for different sections and levels of the code, executes the tests and reports the result to the manager.

  Manual testing is time and resource consuming. The tester needs to confirm whether or not right test cases are used. Major portion of testing involves manual testing.

- **Automated -**This testing is a testing procedure done with aid of automated testing tools. The limitations with manual testing can be overcome using automated test tools.

A test needs to check if a webpage can be opened in Internet Explorer. This can be easily done with manual testing. But to check if the web-server can take the load of 1 million users, it is quite impossible to test manually.

There are software and hardware tools which helps tester in conducting load testing, stress testing, regression testing.

Testing Levels

Testing itself may be defined at various levels of SDLC. The testing process runs parallel to software development. Before jumping on the next stage, a stage is tested, validated and verified.

Testing separately is done just to make sure that there are no hidden bugs or issues left in the software. Software is tested on various levels -

Unit Testing

While coding, the programmer performs some tests on that unit of program to know if it is error free. Testing is performed under white-box testing approach. Unit testing helps developers decide that individual units of the program are working as per requirement and are error free.

Integration Testing

Even if the units of software are working fine individually, there is a need to find out if the units if integrated together would also work without errors. For example, argument passing and data updation etc.

System Testing

The software is compiled as product and then it is tested as a whole. This can be accomplished using one or more of the following tests:

- **Functionality testing** - Tests all functionalities of the software against the requirement.
- **Performance testing** - This test proves how efficient the software is. It tests the effectiveness and average time taken by the software to do desired task. Performance

testing is done by means of load testing and stress testing where the software is put under high user and data load under various environment conditions.

- **Security & Portability** - These tests are done when the software is meant to work on various platforms and accessed by number of persons.

Acceptance Testing

When the software is ready to hand over to the customer it has to go through last phase of testing where it is tested for user-interaction and response. This is important because even if the software matches all user requirements and if user does not like the way it appears or works, it may be rejected.

- **Alpha testing** - The team of developer themselves perform alpha testing by using the system as if it is being used in work environment. They try to find out how user would react to some action in software and how the system should respond to inputs.
- **Beta testing** - After the software is tested internally, it is handed over to the users to use it under their production environment only for testing purpose. This is not as yet the delivered product. Developers expect that users at this stage will bring minute problems, which were skipped to attend.

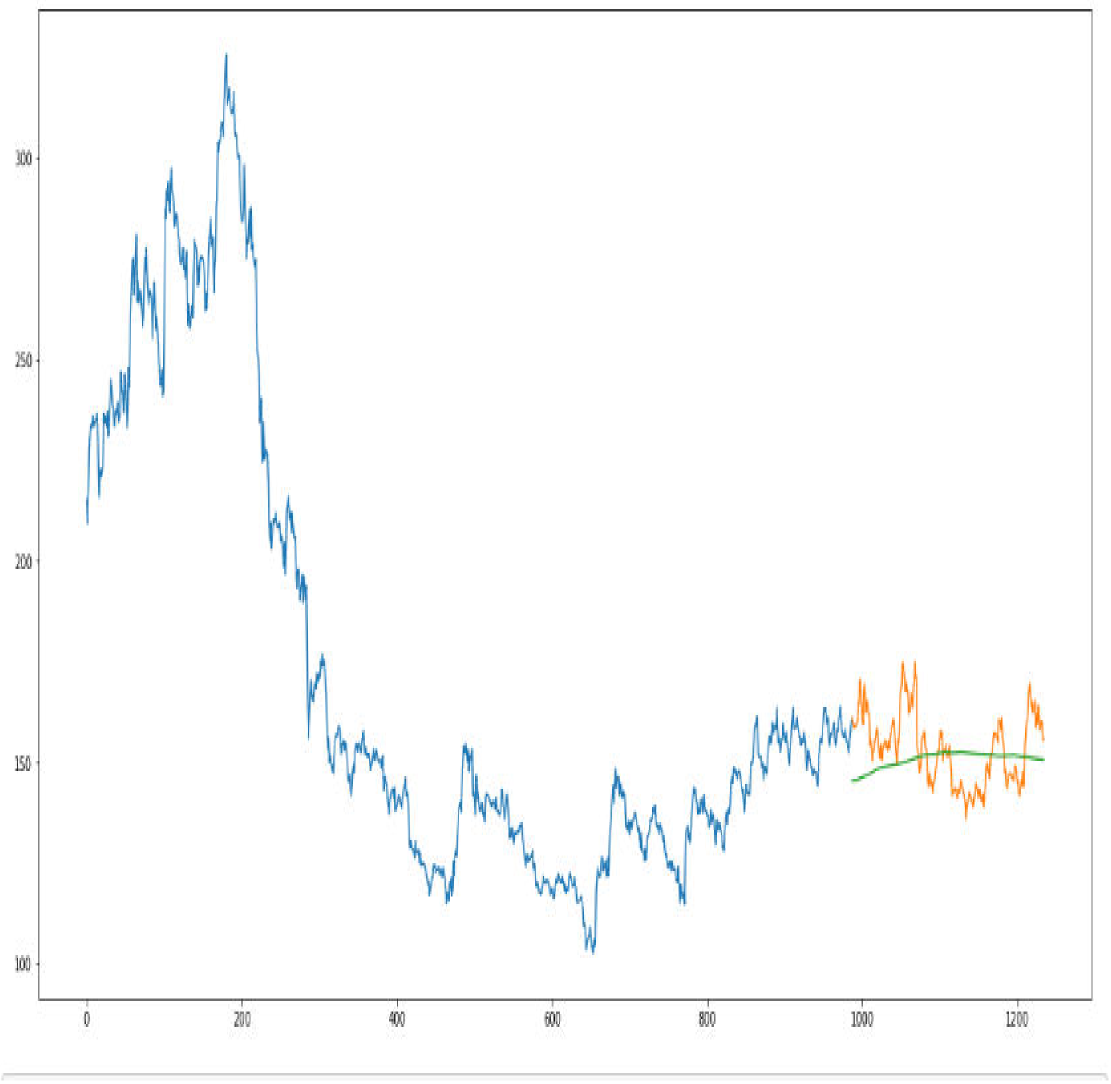
CHAPTER 6

Graphs & Charts

6.1 Graphs

6.1.1Tata Global Beverages Dataset : Moving Average

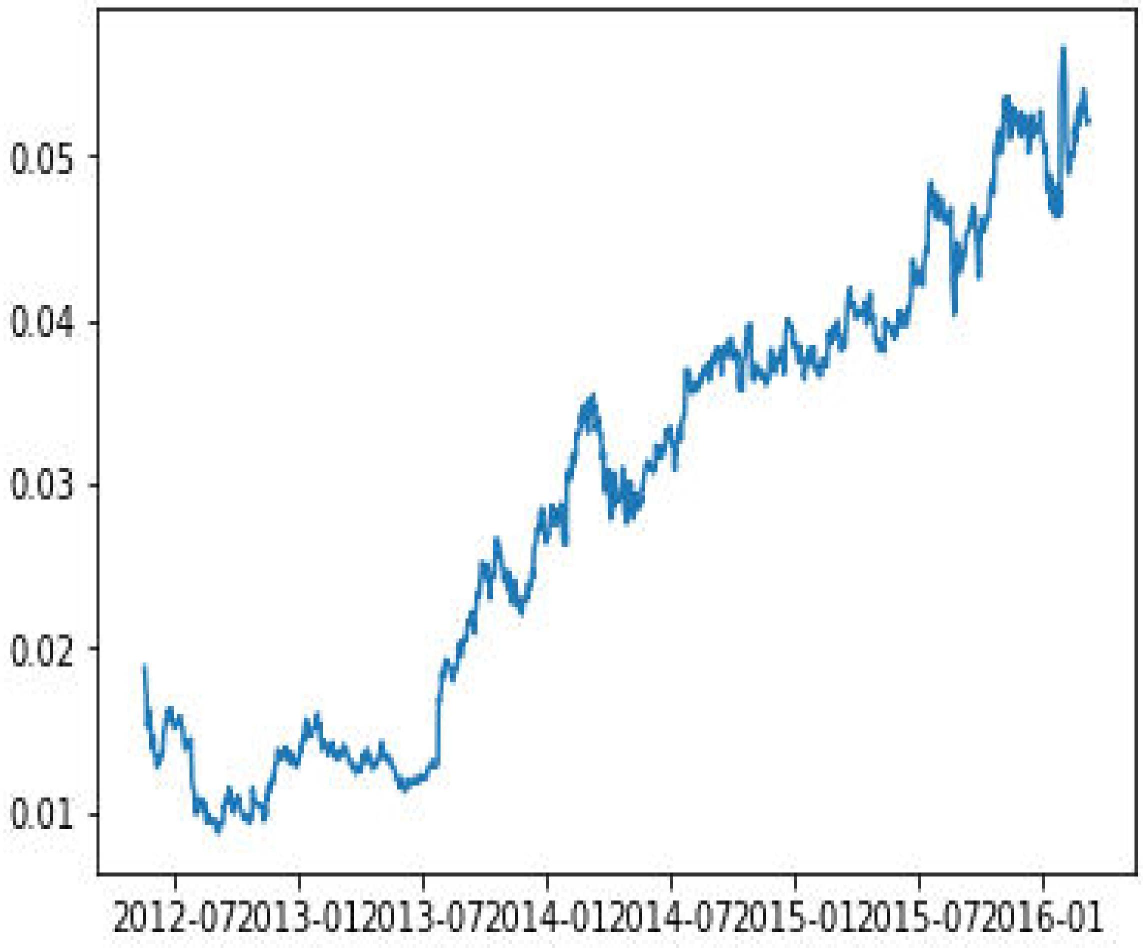
```
24]: [<matplotlib.lines.Line2D at 0x17c692036a0>,
     <matplotlib.lines.Line2D at 0x17c692037f0>]
```



The RMSE value is close to 105 but the results are not very promising (as you can gather from the plot). The predicted values are of the same range as the observed values in the train set (there is an increasing trend initially and then a slow decrease).

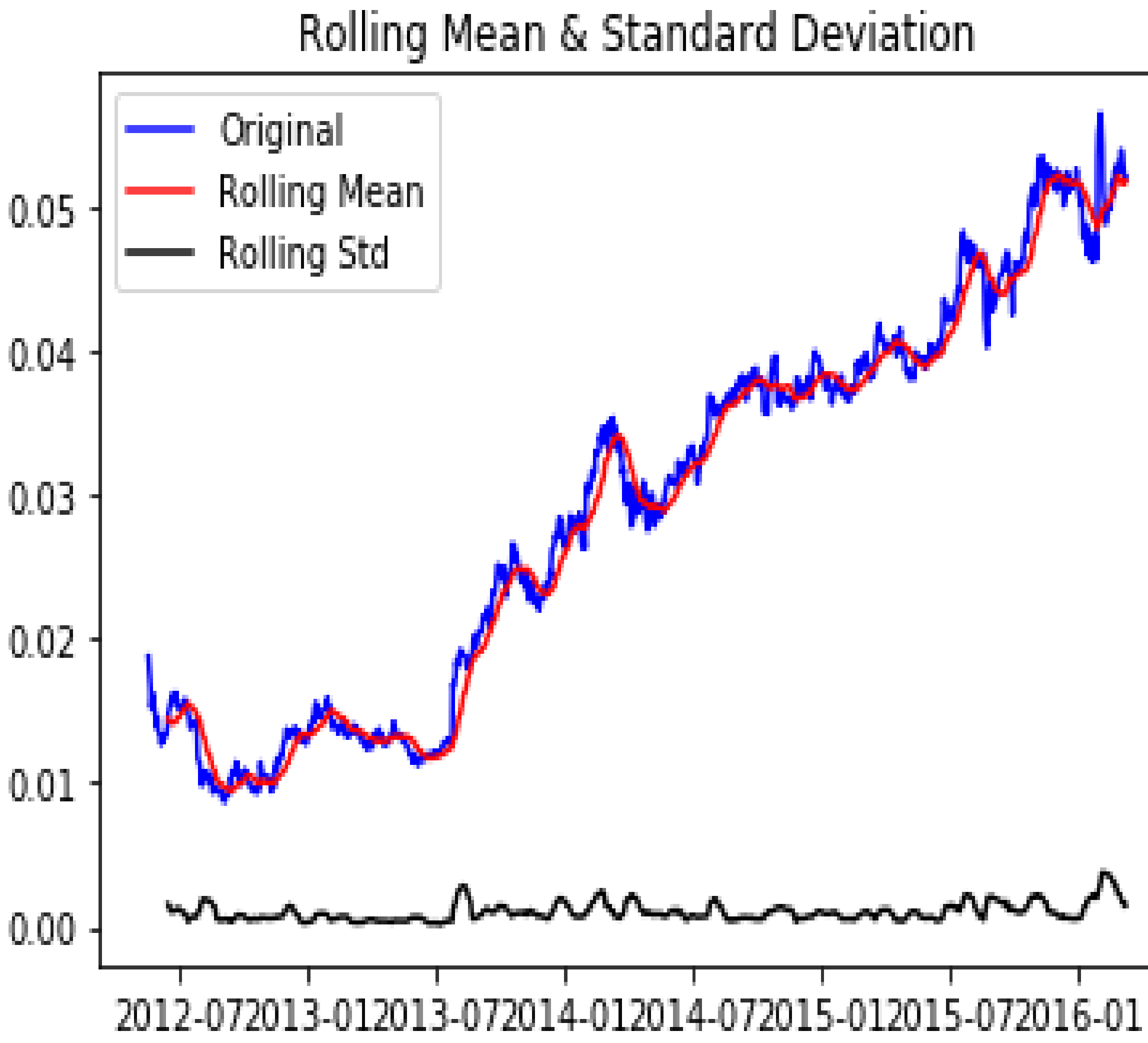
6.2.2 Google Finance Dataset

6.2.2.1 Time Series Plot



6.2.2.2 Dickey-Fuller Test for testing stationarity

Dickey-Fuller Test: This is one of the statistical tests for checking stationarity. Here the null hypothesis is that the TS is non-stationary. The test results comprise of a Test Statisti c and some Critical Values for difference confidence levels. If the 'Test Statistic' is less t han the 'Critical Value', we can reject the null hypothesis and say that the series is station ary.



6.2.2.3 Estimating & Eliminating Trend

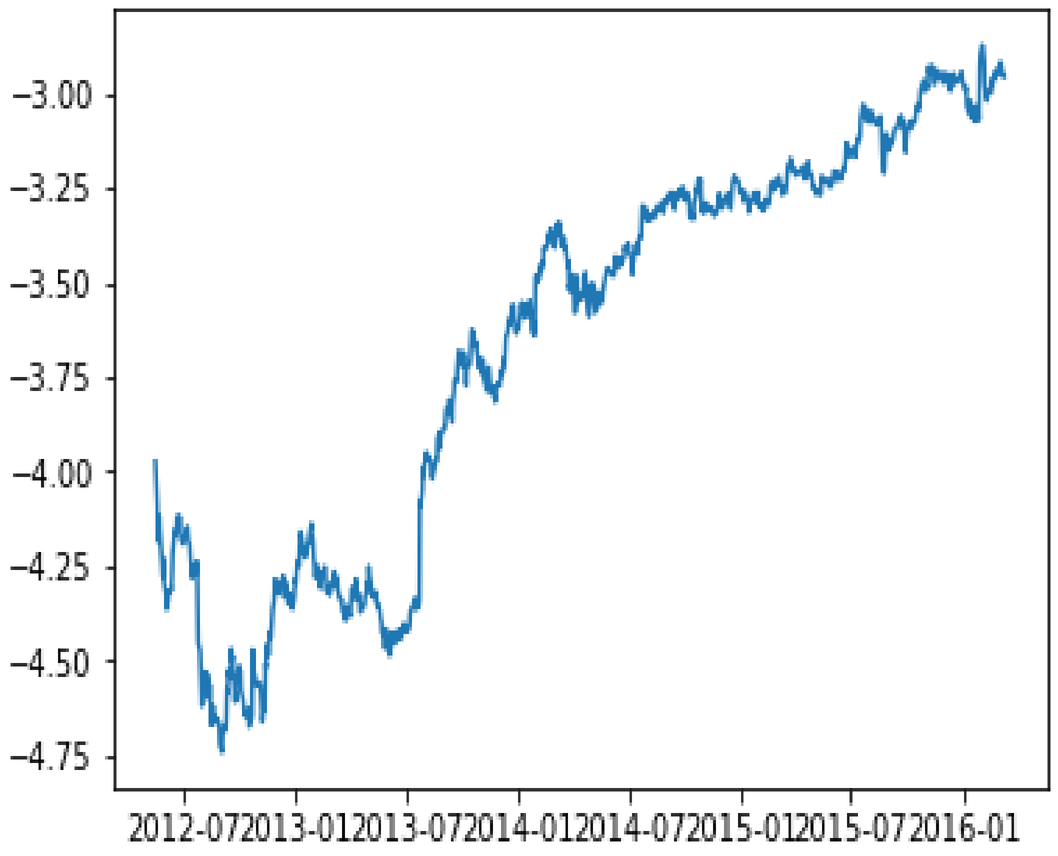
Though stationarity assumption is taken in many TS models, almost none of practical time series are stationary. So statisticians have figured out ways to make series stationary

Lets understand what is making a TS non-stationary. There are 2 major reasons behind non-stationarity of a TS:

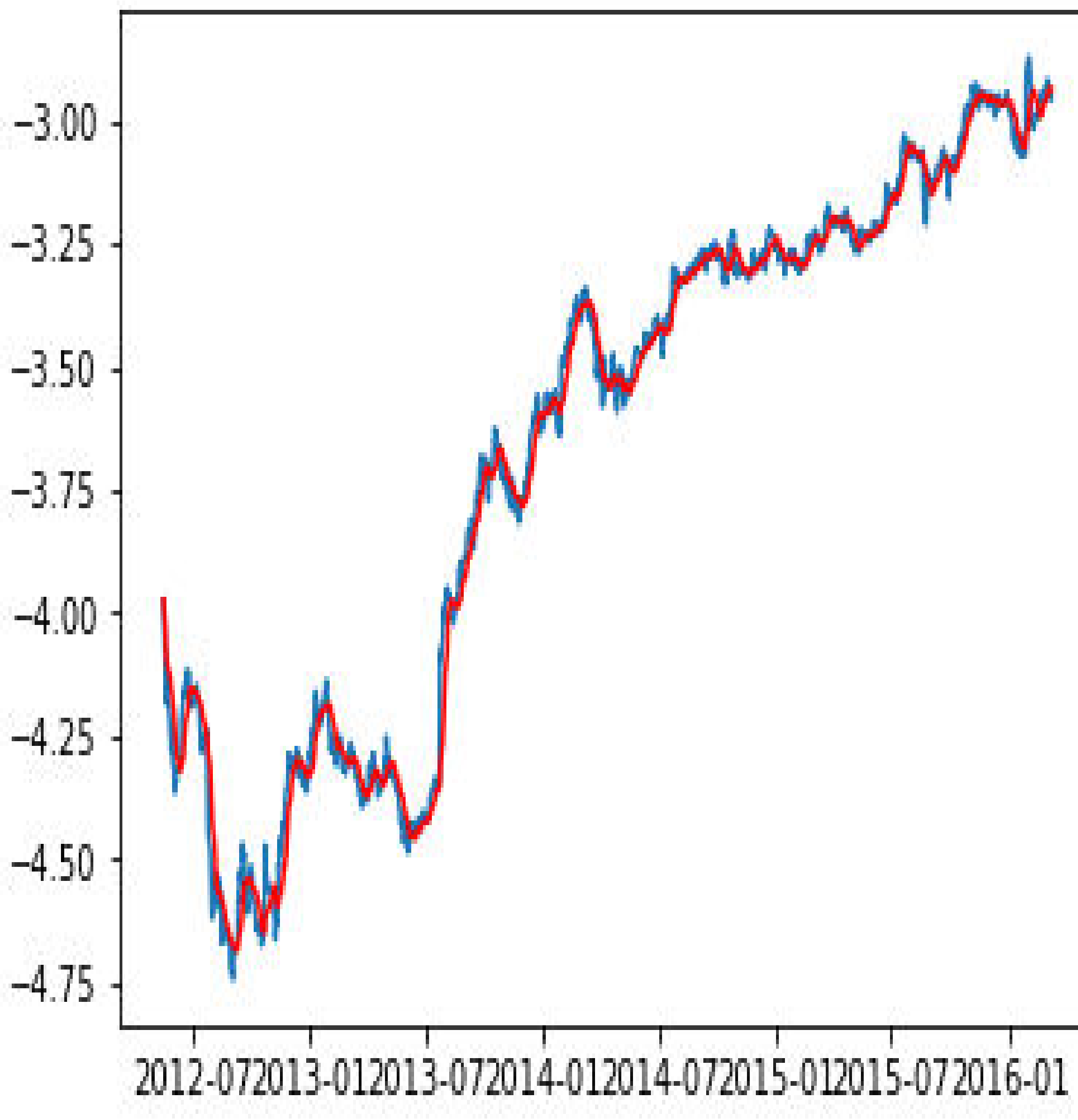
Trend – varying mean over time.

Seasonality – variations at specific time-frames.

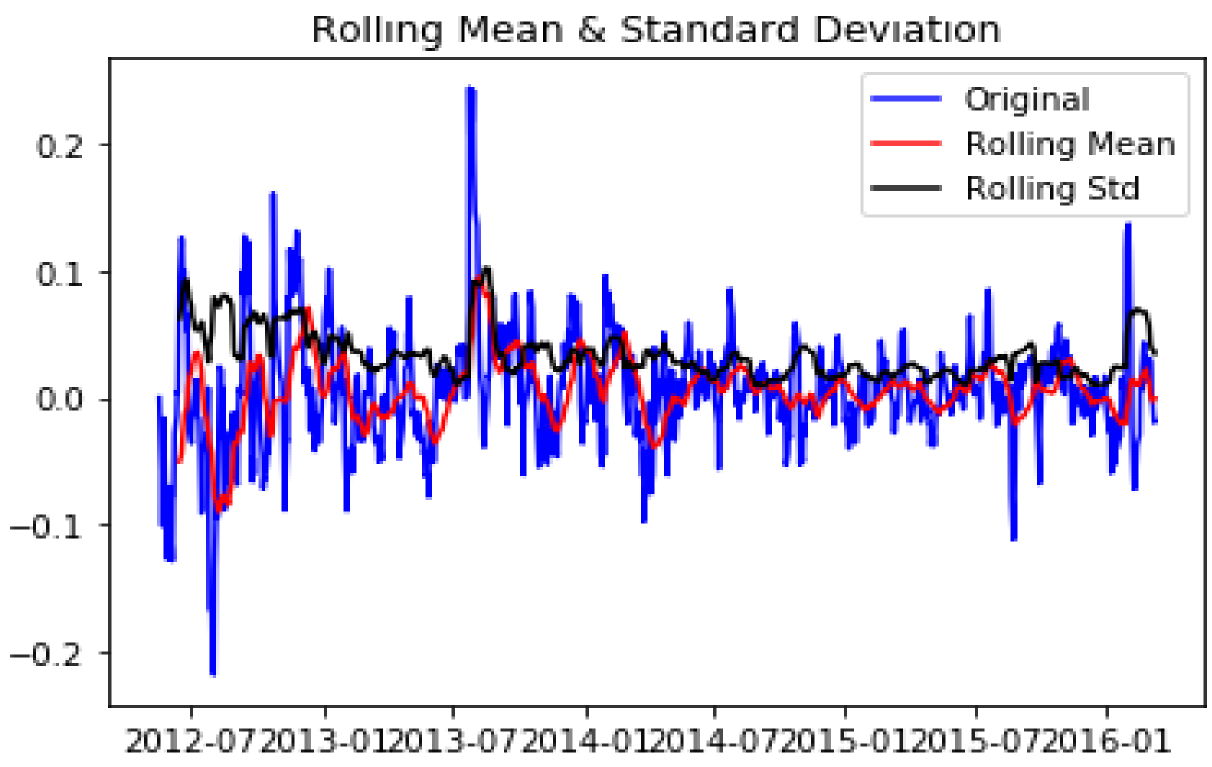
One of the first tricks to reduce trend can be transformation. We will apply transformation which penalize higher values more than smaller values. These can be taking a log, square root, cube root, etc. Lets take a log transform here for simplicity:



6.2.2.4 Smoothing (Moving Average)



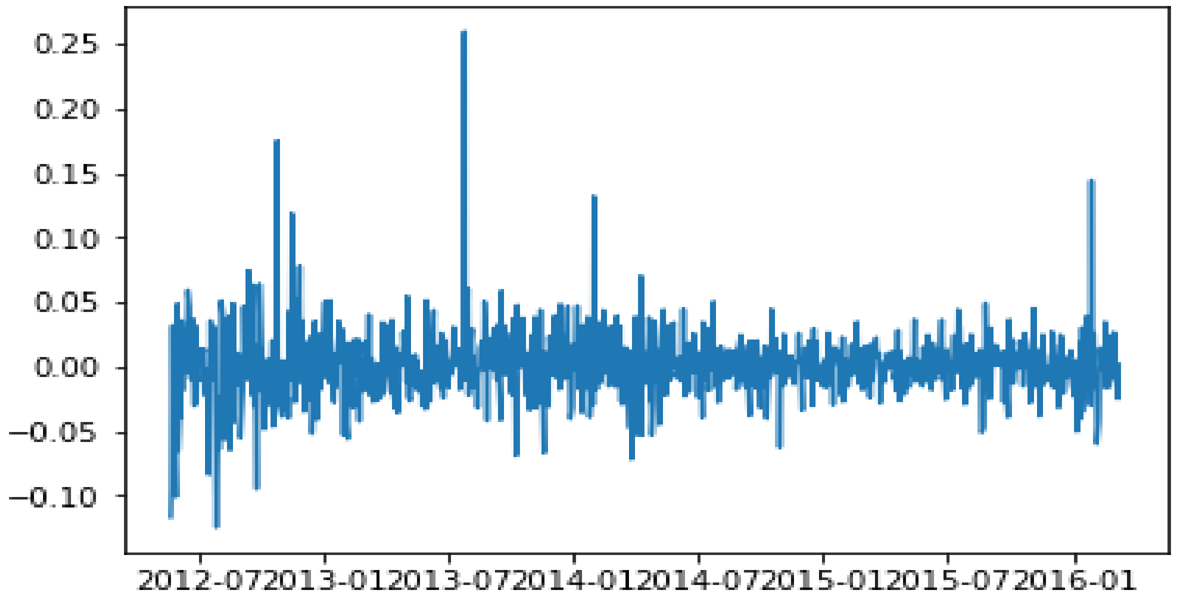
6.2.2.5 Dickey-Fuller Test for testing stationarity

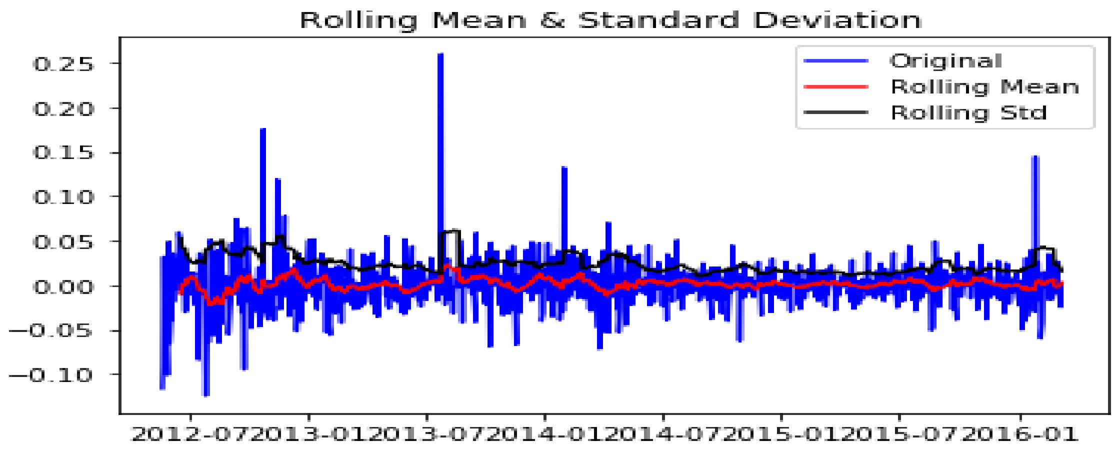


```
Results of Dickey-Fuller Test:
open                       -9.977879e+00
high                        2.153365e-17
10_day_volatility           5.000000e+00
50_day_moving_avg           9.500000e+02
Critical Value (1%)        -3.437252e+00
Critical Value (5%)        -2.864587e+00
Critical Value (10%)       -2.568392e+00
dtype: float64
```

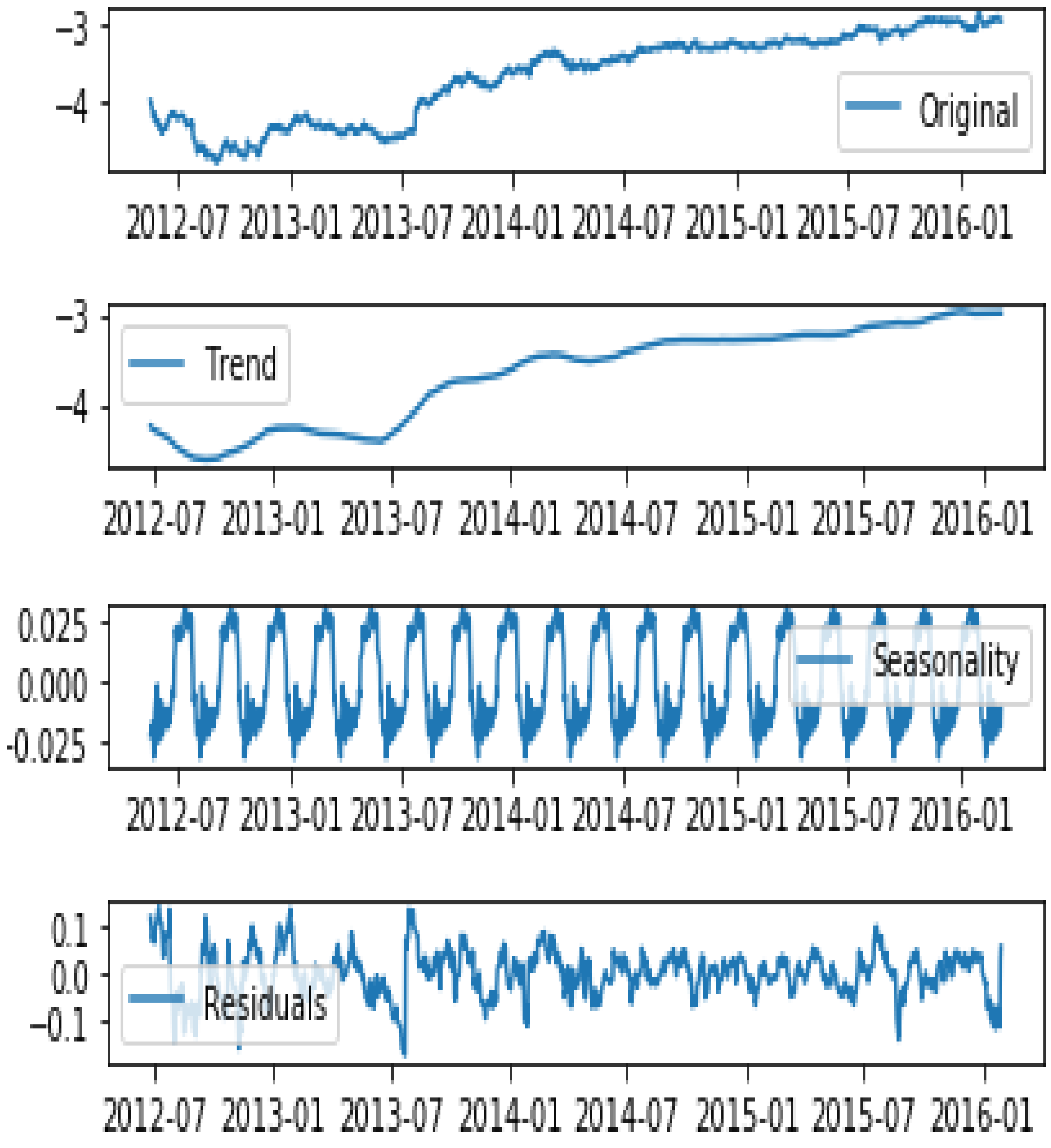
6.2.2.6 Eliminating Trend and Seasonality

Differencing : Differencing is a method of transforming a time series dataset. It can be used to remove the series dependence on time, so-called temporal dependence. This includes structures like trends and seasonality. Differencing is performed by subtracting the previous observation from the current observation.

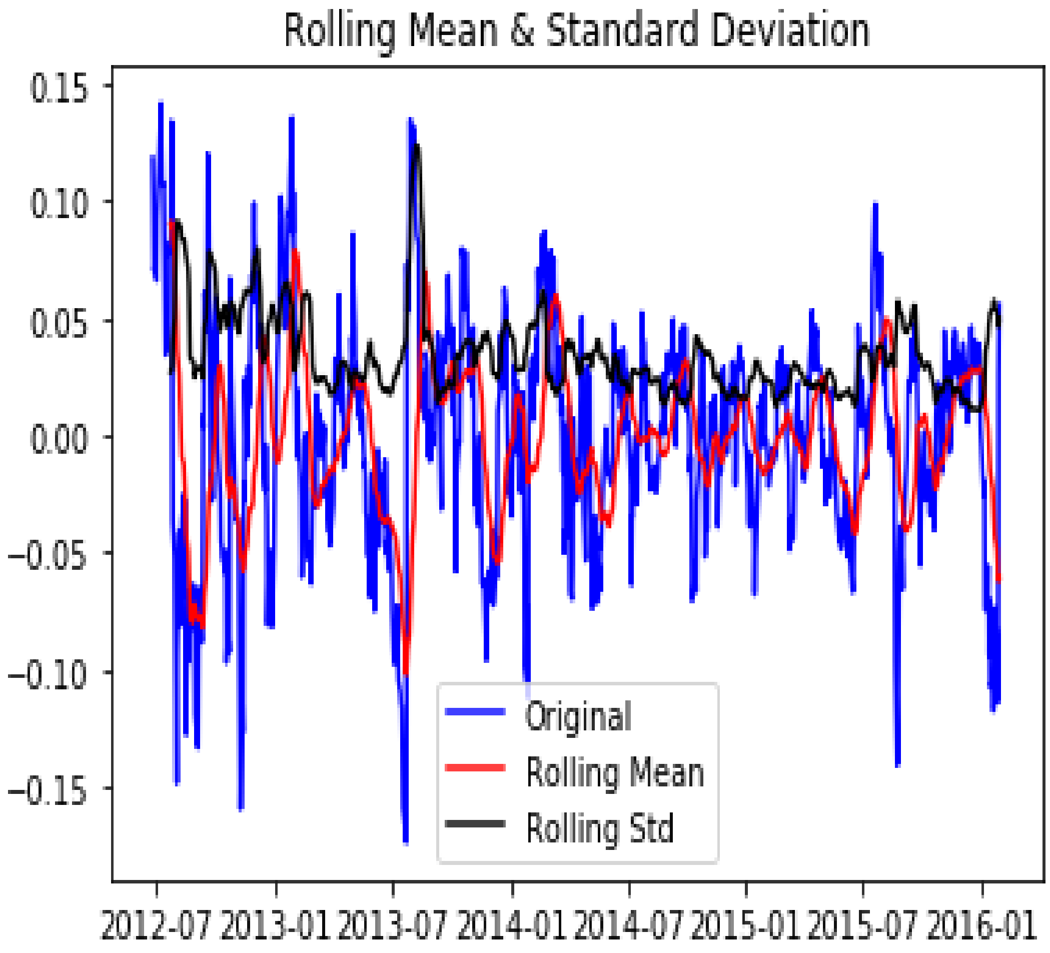


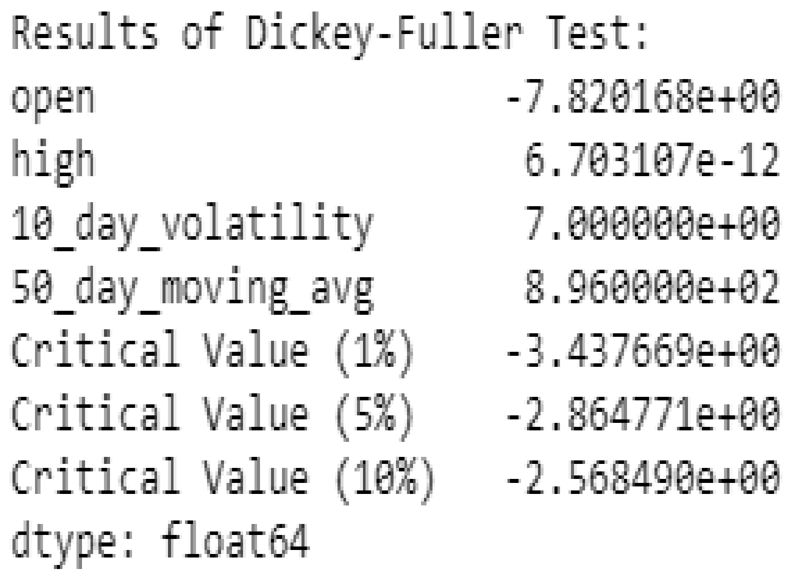


Decomposition



6.2.2.7 Dickey-Fuller Test



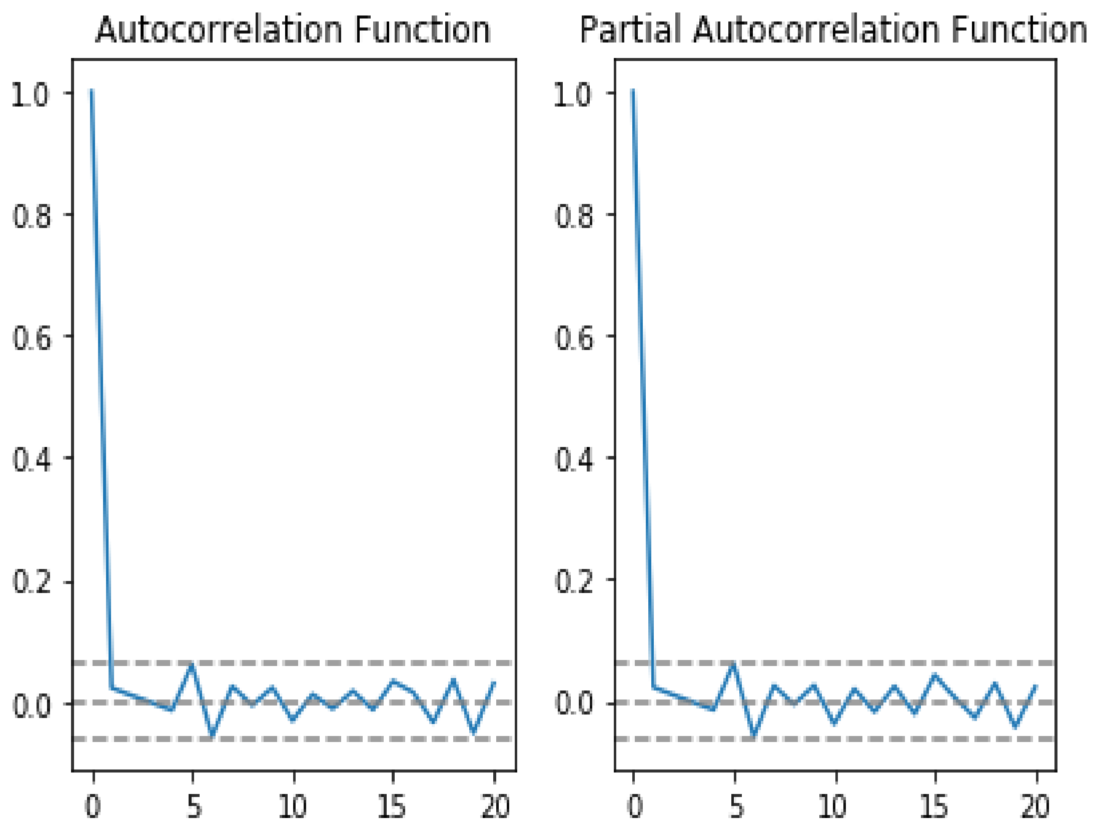


```
Results of Dickey-Fuller Test:
open                       -7.820168e+00
high                        6.703107e-12
10_day_volatility           7.000000e+00
50_day_moving_avg           8.960000e+02
Critical Value (1%)        -3.437669e+00
Critical Value (5%)        -2.864771e+00
Critical Value (10%)       -2.568490e+00
dtype: float64
```

6.2.2.8 Final Forecasting

ACF & PACF Plots

ACF and PACF plots: After a time series has been stationarized by differencing, the next step in fitting an ARIMA model is to determine whether AR or MA terms are needed to correct any autocorrelation that remains in the differenced series. Of course, with software like Statgraphics, you could just try some different combinations of terms and see what works best. But there is a more systematic way to do this. By looking at the autocorrelation function (ACF) and partial autocorrelation (PACF) plots of the differenced series, you can tentatively identify the numbers of AR and/or MA terms that are needed. You are already familiar with the ACF plot: it is merely a bar chart of the coefficients of correlation between a time series and lags of itself. The PACF plot is a plot of the partial correlation coefficients between the series and lags of itself.



6.2.2.9 AR Model

An autoregressive (AR) model predicts future behaviour based on past behaviour. It’s used for forecasting when there is some correlation between values in a time series and the values that precede and succeed them. You only use past data to model the behaviour, hence the name *auto*regressive (the Greek prefix auto– means “self.”). The process is basically a linear regression of the data in the current series against one or more past values in the same series.
In an AR model, the value of the outcome variable (Y) at some point *t* in time is — like “regular” linear regression — directly related to the predictor variable (X). Where simple linear regression and AR models differ is that Y is dependent on X and previous values for Y.

An AR(p) model is an autoregressive model where specific lagged values of y_t are used as predictor variables. Lags are where results from one time period affect following periods.
The value for “p” is called the *order*. For example, an AR(1) would be a “first order autoregressive process.” The outcome variable in a first order AR process at some point in time *t* is related only to time periods that are one period apart (i.e. the value of the variable at t – 1). A second or third order AR process would be related to data two or three periods apart.
The AR(p) model is defined by the equation:

$$\mathbf{y_t=\delta+\varphi_1 y_{t-1}+\varphi_2 y_{t-2}+\ldots+\varphi_p y_{t-1}+A_t}$$

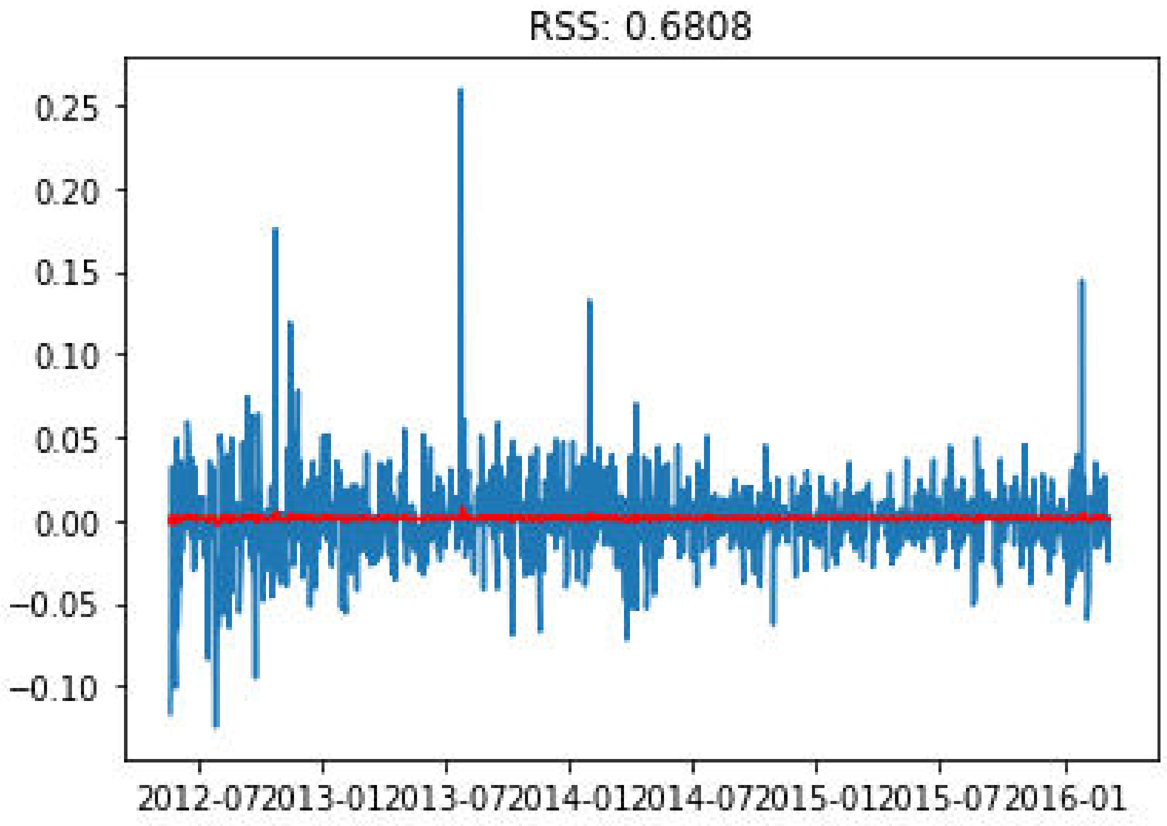
Where:

- $y_{t-1}, y_{t-2} \ldots y_{t-p}$ are the past series values (lags),
- A_t is white noise (i.e. randomness),

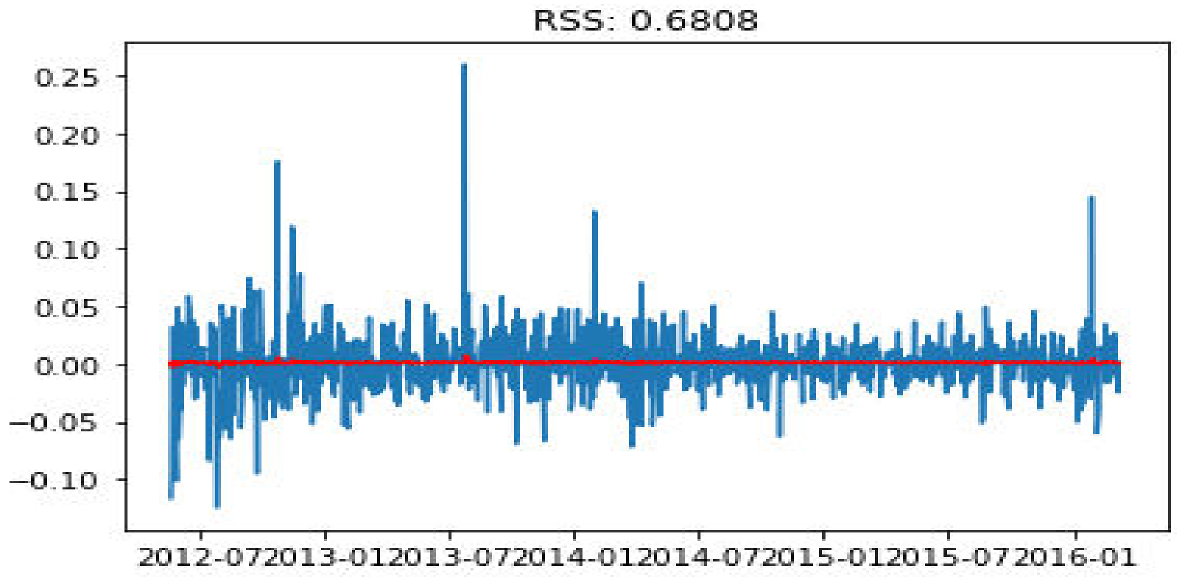
and δ is defined by the following equation:

$$\delta = \left(1 - \sum_{i=1}^{p} \phi_i\right) \mu \, ,$$

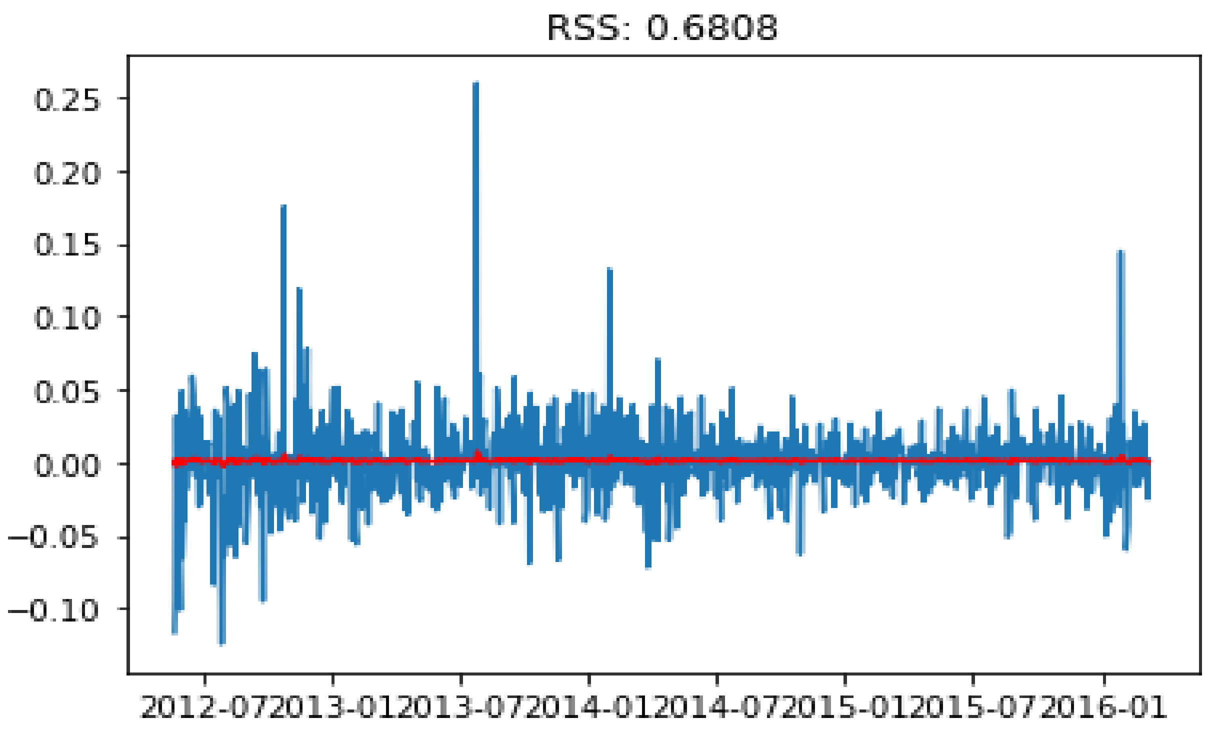
where μ is the process mean



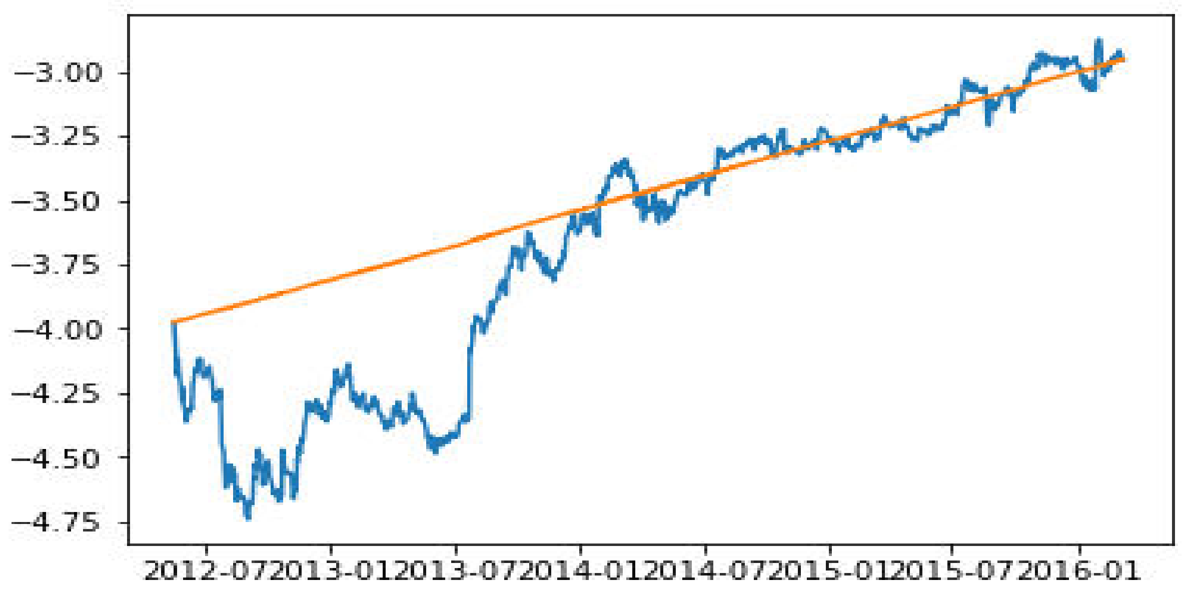
6.2.2.10 MA Model

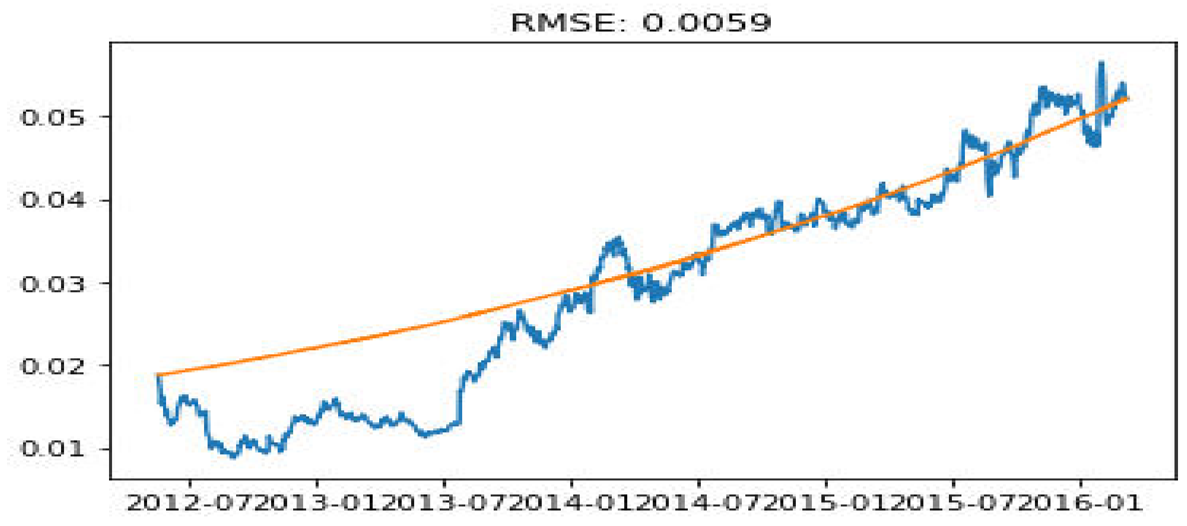


6.2.2.11 ARIMA Model



6.2.2.12 Conversion to Original Scale





6.3 Source Code

Main Script

```
import pandas as pd
import numpy as np
import matplotlib.pylab as plt
%matplotlib inline
from matplotlib.pylab import rcParams
rcParams['figure.figsize'] = 15, 6


data = pd.read_csv('GOOGL.csv')
print(data.head())
print('\n Data Types:')
print( data.dtypes)


dateparse = lambda dates:pd.datetime.strptime(dates, '%Y-%m-%d')
# dateparse('1962-01')
```

```
data = pd.read_csv('GOOGL.csv', parse_dates=['date'],index_col=['date'],date_parser=dateparse)
print (data.head())


#check datatype of index
data.index


#convert to time series:
ts = data['adj_close']
ts.head(10)


#1. Specific the index as a string constant:
ts['2012-05-18']


#2. Import the datetime library and use 'datetime' function:
from datetime import datetime
ts[datetime(2015, 3, 26)]


#1. Specify the entire range:
ts['2012-05-18':'2015-03-26']


from statsmodels.tsa.stattools import adfuller
def test_stationarity(timeseries):
    rolmean = pd.Series(timeseries).rolling(window=20).mean()
    rolstd = pd.Series(timeseries).rolling(window=20).std()



orig = plt.plot(timeseries, color='blue',label='Original')
mean = plt.plot(rolmean, color='red', label='Rolling Mean')
```

```
std = plt.plot(rolstd, color='black', label = 'Rolling Std')
plt.legend(loc='best')
plt.title('Rolling Mean & Standard Deviation')
plt.show(block=False)


 print ('Results of Dickey-Fuller Test:')
 dftest = adfuller(timeseries, autolag='AIC')
 dfoutput = pd.Series(dftest[0:4], index=['open','high','10_day_volatility', '50_day_moving_avg'])
  for key, value in dftest[4].items():
       dfoutput['Critical Value (%s)'%key] = value
       print(dfoutput)




test_stationarity(ts)


ts_log = np.log(ts)
plt.plot(ts_log)

moving_avg = pd.Series(ts_log).rolling(10, min_periods=1).mean()
plt.plot(ts_log)
plt.plot(moving_avg, color='red')

print(moving_avg)

ts_log_moving_avg_diff = ts_log - moving_avg
ts_log_moving_avg_diff.head(15)

ts_log_moving_avg_diff.dropna(inplace=True)
ts_log_moving_avg_diff.head()

test_stationarity(ts_log_moving_avg_diff)



expwighted_avg = pd.DataFrame.ewm(ts_log, halflife=12).mean() plt.plot(ts_log) plt.plot(expwi
ghted_avg, color='red')

ts_log_ewma_diff = ts_log - expwighted_avg
test_stationarity(ts_log_ewma_diff)

ts_log_diff = ts_log - ts_log.shift()
plt.plot(ts_log_diff)

ts_log_diff.dropna(inplace=True)
test_stationarity(ts_log_diff)



from statsmodels.tsa.seasonal import seasonal_decompose
decomposition = seasonal_decompose(ts_log, freq=52)

trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid

plt.subplot(411)
plt.plot(ts_log, label='Original')
plt.legend(loc='best')
plt.subplot(412)
plt.plot(trend, label='Trend')
plt.legend(loc='best')
plt.subplot(413)
plt.plot(seasonal,label='Seasonality')
plt.legend(loc='best')
plt.subplot(414)
plt.plot(residual, label='Residuals')
plt.legend(loc='best')
plt.tight_layout()



ts_log_decompose = residual
ts_log_decompose.dropna(inplace=True)
test_stationarity(ts_log_decompose)


from statsmodels.tsa.arima_model import ARIMA
#ACF and PACF plots:
from statsmodels.tsa.stattools import acf, pacf

lag_acf = acf(ts_log_diff, nlags=20)
lag_pacf = pacf(ts_log_diff, nlags=20, method='ols')

#Plot ACF:
plt.subplot(121)
plt.plot(lag_acf)
plt.axhline(y=0,linestyle='--',color='gray')
plt.axhline(y=-1.96/np.sqrt(len(ts_log_diff)),linestyle='--',color='gray')
plt.axhline(y=1.96/np.sqrt(len(ts_log_diff)),linestyle='--',color='gray')
plt.title('Autocorrelation Function')

#Plot PACF:
plt.subplot(122)
plt.plot(lag_pacf)
plt.axhline(y=0,linestyle='--',color='gray')
plt.axhline(y=-1.96/np.sqrt(len(ts_log_diff)),linestyle='--',color='gray')
plt.axhline(y=1.96/np.sqrt(len(ts_log_diff)),linestyle='--',color='gray')
plt.title('Partial Autocorrelation Function')
plt.tight_layout()


#MA model:
model = ARIMA(ts_log, order=(2, 1, 0))
results_AR = model.fit(disp=-1)
plt.plot(ts_log_diff)
plt.plot(results_AR.fittedvalues, color='red')
plt.title('RSS: %.4f'% sum((results_AR.fittedvalues-ts_log_diff)**2))



model = ARIMA(ts_log, order=(0, 1, 2))
results_MA = model.fit(disp=-1)
plt.plot(ts_log_diff)
plt.plot(results_MA.fittedvalues, color='red')
plt.title('RSS: %.4f'% sum((results_MA.fittedvalues-ts_log_diff)**2))



model = ARIMA(ts_log, order=(2, 1, 2))
results_ARIMA = model.fit(disp=-1)
plt.plot(ts_log_diff)
plt.plot(results_ARIMA.fittedvalues, color='red')
plt.title('RSS (Root Squared Sum): %.4f'% sum((results_ARIMA.fittedvalues-ts_log_diff)**2))


predictions_ARIMA_diff = pd.Series(results_ARIMA.fittedvalues, copy=True)
print(predictions_ARIMA_diff.head())
```

```
predictions_ARIMA_diff_cumsum = predictions_ARIMA_diff.cumsum()
print(predictions_ARIMA_diff_cumsum.head())

predictions_ARIMA_log = pd.Series(ts_log.ix[0], index=ts_log.index)
predictions_ARIMA_log = predictions_ARIMA_log.add(predictions_ARIMA_diff_cumsum, fill
_value=0)
predictions_ARIMA_log.head()

plt.plot(ts_log) plt.plot(predictions_ARIMA_log)
predictions_ARIMA = np.exp(predictions_ARIMA_log)
plt.plot(ts)
plt.plot(predictions_ARIMA)
plt.title('RMSE: %.4f'% np.sqrt(sum((predictions_ARIMA-ts)**2)/len(ts)))
```

CHAPTER 7

REFRENCES/BIBILIOGRAPHY

1. Stack Overflow

https://stackoverflow.com

2. Analytics Vidya

https://www.analyticsvidhya.com/blog/category/machine-learning/

3. Sent Desk

https://www.youtube.com/watch?v=OGxgnH8y2NM&list=PLQVvvaa0QuDfKTOs3Keq_kaG2P55YRn5.4.

4. Udemy

https://www.udemy.com/machinelearning/

5. Dataset

https://www.quandl.com/data/NSE/TATAGLOBAL-Tata-Global-Beverages-Limited

https://www.google.com/finance