Configuration Manual

MSc Research Project
MSc in Data Analytics

Smit Jain
x18135340

School of Computing
National College of Ireland

Submitted to:
Dr. Pierpaolo Dondio

National College of Ireland

Project Submission Sheet – 2019/2020

Student Name:	Smit Jain
Student ID:	18135340
Programme:	MSc in Data Analytics **Year:** 2019-2020
Module:	Configuration module for research project
Lecturer:	Dr. Pierpaolo Dondio
Submission Due Date:	12/12/2019
Project Title:	Analysing effect of Twitter Tweets, Oil Prices, Gold Prices and Foreign Exchange on S&P500 Using Machine Learning.
Word Count:	1201

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the references section. Students are encouraged to use the Harvard Referencing Standard supplied by the Library. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action. Students may be required to undergo a viva (oral examination) if there is suspicion about the validity of their submitted work.

Signature: Smit Jain

Date: 11/12/2019

PLEASE READ THE FOLLOWING INSTRUCTIONS:

1. Please attach a completed copy of this sheet to each project (including multiple copies).
2. Projects should be submitted to your Programme Coordinator.
3. **You must ensure that you retain a HARD COPY of ALL projects**, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer. Please do not bind projects or place in covers unless specifically requested.
4. You must ensure that all projects are submitted to your Programme Coordinator on or before the required submission date. **Late submissions will incur penalties.**
5. All projects must be submitted and passed in order to successfully complete the year. **Any project/assignment not submitted will be marked as a fail.**

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

1. Introduction to configuration manual:

This configuration manual can be used to replicate the work done by the author and get the desired results. The manual includes the system configuration requirements, the steps to gather the data, clean the gathered data and then use the data to implement the models and use evaluation criteria to compare the results of different models. The code snippets are attached in the last section.

2. Pre-requisites and system configuration:

The tools and software used for this thesis research work can be installed on a laptop or a PC. The basic configuration list is given below:

Operating system	Windows 10
RAM	8 GB
Hard Disk	512 SSD
Processor	Core i7 8^{th} gen

Getting started:

The basic toolset used in this research work for carrying out all the actions are listed below:

- Microsoft office tools
- Python 3.7
- Anaconda Spyder

The Microsoft office tools like Microsoft Excel and Word have been used. Python as a language has been used for this research work and all the processes like data gathering, data cleaning, transformation and analysis has been done in python language. The software version for python used is 3.7 and the latest version can be downloaded anytime for free from the website – 'https://www.python.org/downloads/'. The platform used for coding is Anaconda. Anaconda can be downloaded from the link – 'https://repo.anaconda.com/archive/Anaconda3-2019.10-Windows-x86_64.exe'.
Spyder has been used in the anaconda platform because it is very convenient to use, and all the variables can be seen in the variable explorer which makes it easy when data cleaning and manipulations are being done.

3. Dataset generation:

The datasets generated for this research work are –

1. Stock Market data set:
   This dataset has been taken from Yahoo finance (Figure 1). On the website (https://finance.yahoo.com/quote/%5EGSPC/history?p=%5EGSPC), we can select the time period and the daily data for the time period selected can be downloaded in csv format. The dataset downloaded is suitable for a time-series analysis.

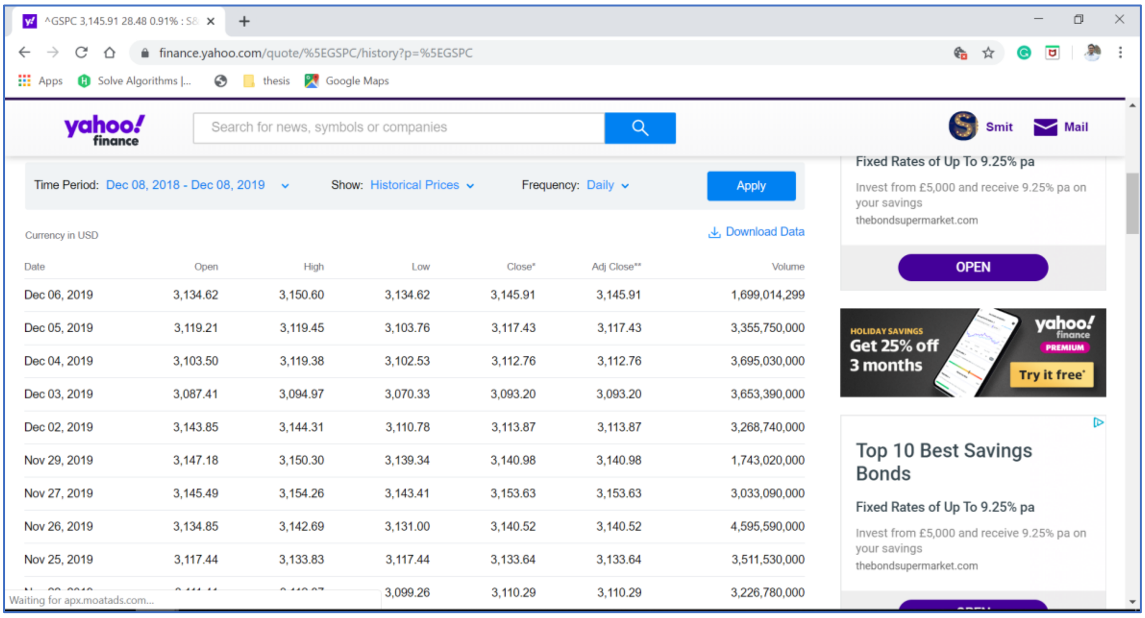


Figure 1: Yahoo!finance data set gathering

2. Twitter data set:
   This data set has been generated using twitter. The data has been scraped from twitter using the python command on anaconda command prompt. The command used is – 'twitterscraper "#SP500" -l 10000 -bd 2018-01-01 -ed 2019-01-01 -o tweets.json'. This command (Figure 2) will scrape the twitter data for the specified timeframe and a .json file will be created. Next, this json file is imported in python and converted into a csv and then operated and the dataset is cleaned (Figure 3). The dataset downloaded is suitable for a time-series analysis.

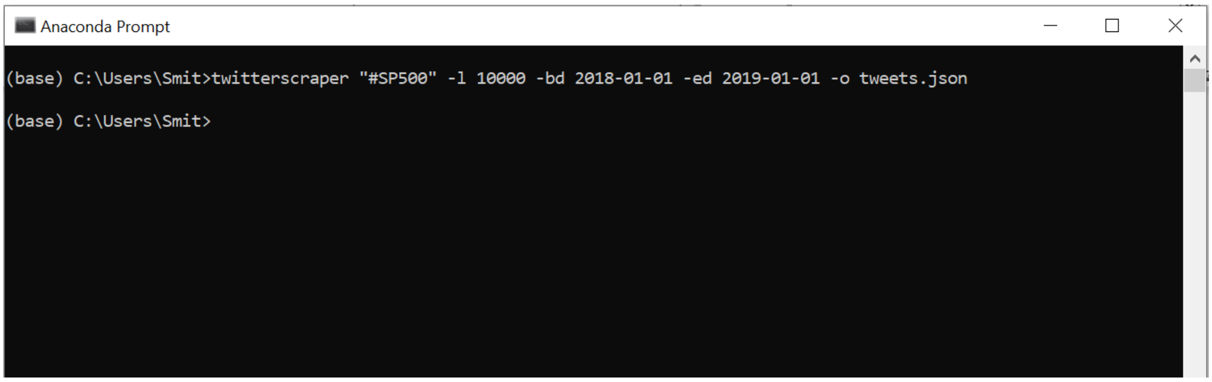
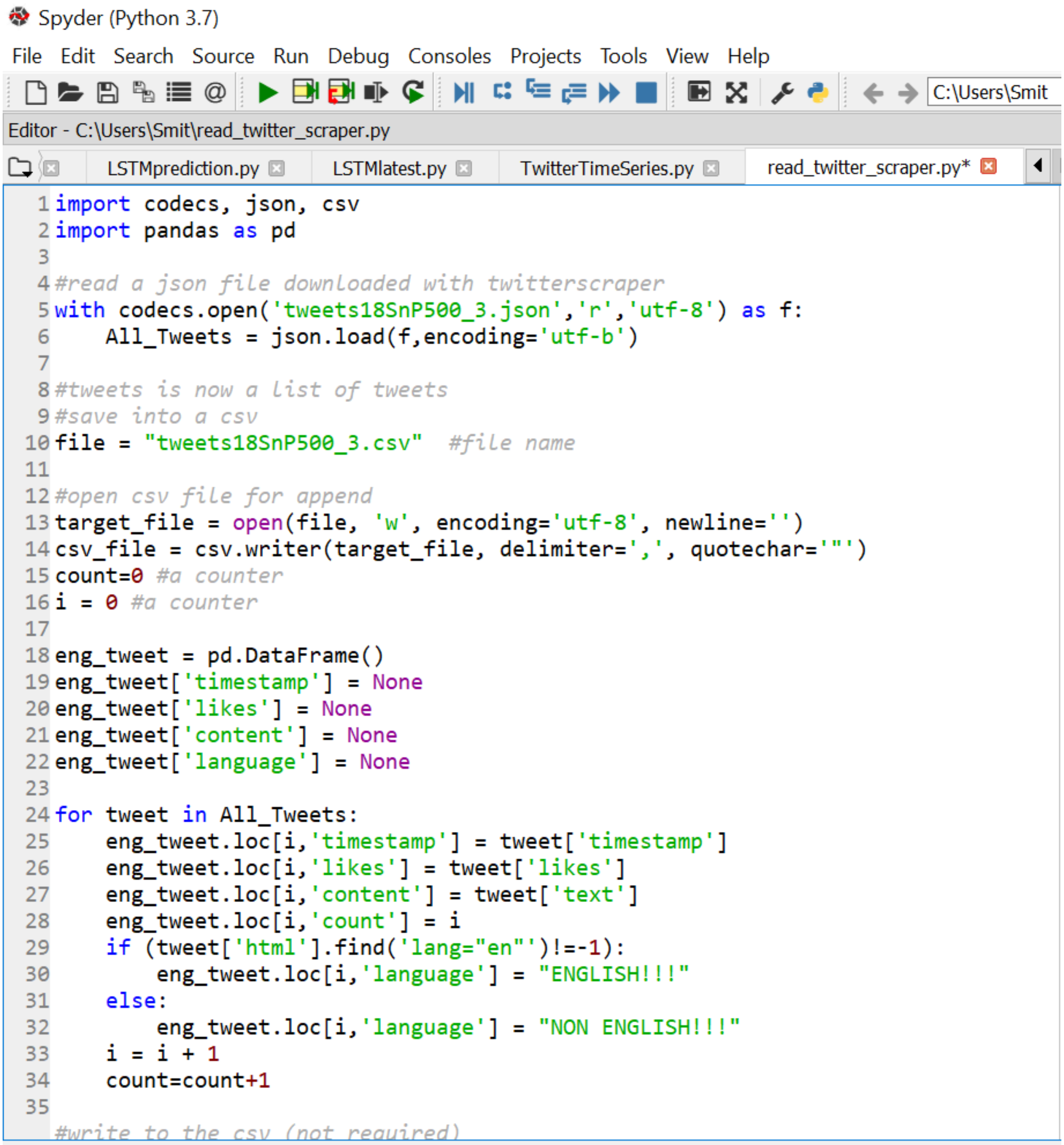


Figure 2: Twitterscraper, scraping tweets using Anaconda prompt



```python
import codecs, json, csv
import pandas as pd

#read a json file downloaded with twitterscraper
with codecs.open('tweets18SnP500_3.json','r','utf-8') as f:
    All_Tweets = json.load(f,encoding='utf-b')

#tweets is now a list of tweets
#save into a csv
file = "tweets18SnP500_3.csv"  #file name

#open csv file for append
target_file = open(file, 'w', encoding='utf-8', newline='')
csv_file = csv.writer(target_file, delimiter=',', quotechar='"')
count=0 #a counter
i = 0 #a counter

eng_tweet = pd.DataFrame()
eng_tweet['timestamp'] = None
eng_tweet['likes'] = None
eng_tweet['content'] = None
eng_tweet['language'] = None

for tweet in All_Tweets:
    eng_tweet.loc[i,'timestamp'] = tweet['timestamp']
    eng_tweet.loc[i,'likes'] = tweet['likes']
    eng_tweet.loc[i,'content'] = tweet['text']
    eng_tweet.loc[i,'count'] = i
    if (tweet['html'].find('lang="en"')!=-1):
        eng_tweet.loc[i,'language'] = "ENGLISH!!!"
    else:
        eng_tweet.loc[i,'language'] = "NON ENGLISH!!!"
    i = i + 1
    count=count+1

#write to the csv (not required)
```

Figure 3: code snippet to convert the json file and clean the dataset.

4. Research design workflow and methodology:

In this research work, the datasets have been first extracted from their sources and then imported in python for doing the cleaning. After the datasets are cleaned, a test known as Granger Causality test has been done to check the causality of different variables in the prediction of the stock market index. After the test, the twitter dataset is found to be a causal for the prediction of stock market hence, the twitter dataset along with the stock market dataset is used in various models for the prediction. After the predictions are made, the results are evaluated using MSE, RMSE, MAE and MAPE which are standard protocols for any time series prediction models. The design flowchart (Figure 4) is given below:

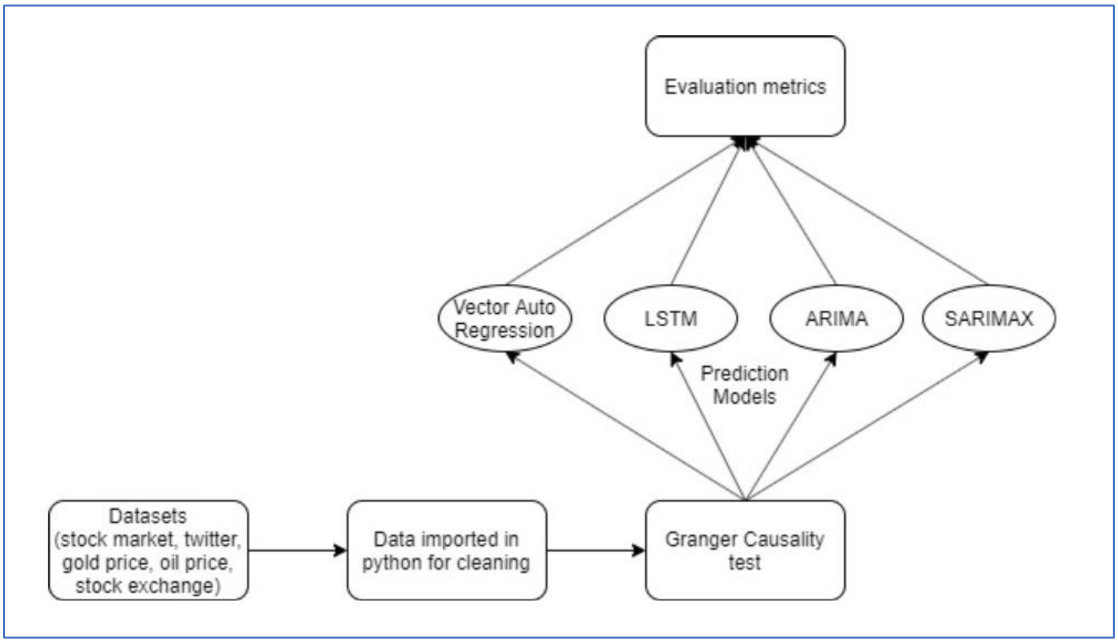


Figure 4: Design flowchart

5. Libraries used in code:

- import codecs, json, csv – these libraries have been used to deal with the tweets downloaded in json format and then convert them into a csv.
- import pandas as pd – Pandas library used for calculations and data manipulations.
- import warnings – library used to implement alerts
- import itertools – library used for iterating through loops
- import numpy as np – library used for performing mathematical functions
- import matplotlib.pyplot as plt – library used for plotting graphs
- import statsmodels.api as sm – library used for implementing statistical models
- from sklearn.metrics import mean_absolute_error,mean_squared_error – used for calculating MAE and MSE.
- import math – library for performing mathematical operations
- from statsmodels.tools.eval_measures import rmse, aic – library used for calculating RMSE values
- %matplotlib inline – the plots are plotted in line
- import os - imports miscellaneous interfaces of the operating system
- import seaborn as sb – library used for data visualization
- import re- the regex library
- from sklearn import preprocessing – library used for data preprocessing
- from functools import reduce – function used for performing computations on a list
- from statsmodels.tsa.api import VAR – for applying the VAR model
- from statsmodels.tsa.stattools import adfuller – used for conducting adfuller test
- from scipy import stats – for applying statistical functions
- sb.set_style('darkgrid') – for setting style in seaborn
- from pmdarima import auto_arima – for calculating the auto arima score
- from statsmodels.tsa.arima_model import ARIMA – to implement ARIMA model.

6. Implemented models (Code snippets):

The models implemented in this research work are VAR, LSTM, ARIMA and SARIMAX. The code snippets for each of the model implemented is given below.

1. Vector Auto Regressor (VAR) model:

The code snippet for this model is given below.

```
print('STOCK PREDICTION USING RNN LSTM')
import numpy as np
import pandas as pd
from sklearn import preprocessing
from functools import reduce
import matplotlib.pyplot as plt
from statsmodels.tsa.api import VAR
from statsmodels.tsa.stattools import adfuller
from statsmodels.tools.eval_measures import rmse, aic
from sklearn.metrics import mean_absolute_error,mean_squared_error
import math
%matplotlib inline
####
# load the dataset
####
SMdata= pd.read_csv('C:/Users/Smit/Dataset/yahoo/stockMarket.csv')
TWratio = pd.read_csv('C:/Users/Smit/PosRatioTweets.csv')
TWvol = pd.read_csv('C:/Users/Smit/TWvol.csv')
#merge all the datasets
data_frames = [SMdata, TWratio, TWvol]
data_csv = reduce(lambda  left,right: pd.merge(left,right,on=['Date'],
                                              how='outer'), data_frames)
data_csv = data_csv.drop(data_csv.index[1260:1640])
data_csv.isna().count()
data_csv = data_csv.drop('Unnamed: 0', axis=1)
data_csv = data_csv.drop('DateTime', axis=1)
data_csv = data_csv.fillna(0)
data_csv[['Close']].plot()
plt.show()
plt.clf()
########################################################
```

```
#VAR model
drop = ['Open','High','Low','Adj Close','Volume','Negative','Positive','Count']
data_csv = data_csv.drop(drop,axis=1)
data_csv = data_csv.set_index('Date')
data_csv.info()
nobs = 30
df_train, df_test = data_csv[0:-nobs], data_csv[-nobs:]
model = VAR(data_csv)
for i in [1,2,3,4,5,6,7,8,9]:
    result = model.fit(i)
    print('Lag Order =', i)
    print('AIC : ', result.aic)
    print('BIC : ', result.bic)
    print('FPE : ', result.fpe)
    print('HQIC: ', result.hqic, '\n')
x = model.select_order(maxlags=12)
x.summary()
model_fitted = model.fit(7)
model_fitted.summary()
forecast_input = df_train.values
fc = model_fitted.forecast(y=forecast_input, steps=nobs)
df_forecast = pd.DataFrame(fc, index=data_csv.index[-nobs:], columns=data_csv.columns + '_2d')
df_forecast
plt.plot(df_train['Close'])

mse = mean_squared_error(df_forecast['Close_2d'],df_test['Close'])
rmse = math.sqrt(mse)
mae = mean_absolute_error(df_forecast['Close_2d'],df_test['Close'])
MAPE = np.mean(np.abs((df_test['Close'] - df_forecast['Close_2d']) / df_forecast['Close_2d'])) * 100
print('The Mean Absolute Percentage Error is {:.2f}%'.format(MAPE))

msee = mean_squared_error(df_forecast['Ratio_2d'],df_test['Ratio'])
rmsee = math.sqrt(msee)
maee = mean_absolute_error(df_forecast['Ratio_2d'],df_test['Ratio'])
MAPEE = np.mean(np.abs((df_test['Ratio'] - df_forecast['Ratio_2d']) / df_forecast['Ratio_2d'])) * 100
print('The Mean Absolute Percentage Error is {:.2f}%'.format(MAPEE))
```

```

df_test['predicted_close'] = df_forecast['Close_2d']
df_test['predicted_ratio'] = df_forecast['Ratio_2d']

plt.plot(df_test['Close'], label='Actual Close')
plt.plot(df_test['predicted_close'], label='Predicted Close')
plt.legend()

plt.plot(df_test['Ratio'], label='Actual Ratio')
plt.plot(df_test['predicted_ratio'], label=['Predicted Ratio'])
plt.legend()
```

2. LSTM model:

The code snippet for the LSTM model is given below.

```
import numpy
import matplotlib.pyplot as plt
import pandas
import math
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error,mean_absolute_error
import time

numpy.random.seed(7)

# load the dataset
dataframe = pandas.read_csv('C:/Users/Smit/Dataset/yahoo/stockMarket.csv',
                            usecols=[1], engine='python', skipfooter=3)
dataset = dataframe.values
dataset = dataset.astype('float32')

```

```
# normalize the data
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)

#train and test sets
train_size = int(len(dataset) * 0.85)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size,:], dataset[train_size:len(dataset),:]
print(len(train), len(test))

# values into a dataset matrix
def create_dataset(dataset, look_back=1):
    dataX, dataY = [], []
    for i in range(len(dataset)-look_back-1):
        a = dataset[i:(i+look_back), 0]
        dataX.append(a)
        dataY.append(dataset[i + look_back, 0])
    return numpy.array(dataX), numpy.array(dataY)

# reshape into X=t and Y=t+1
look_back = 30
trainX, trainY = create_dataset(train, look_back)
testX, testY = create_dataset(test, look_back)

# reshape input to be [samples, time steps, features]
trainX = numpy.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
testX = numpy.reshape(testX, (testX.shape[0], 1, testX.shape[1]))
start_time = time.time()

# create and fit the LSTM model
model = Sequential()
model.add(LSTM(30, input_shape=(1, look_back)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(trainX, trainY, epochs=10, batch_size=1, verbose=2)
```

```
# make predictions
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)

# invert predictions
trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform([trainY])
testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform([testY])

# calculate root mean squared error
trainScore = math.sqrt(mean_squared_error(trainY[0], trainPredict[:,0]))
print('Train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(testY[0], testPredict[:,0]))
print('Test Score: %.2f RMSE' % (testScore))
testScore_mse = mean_squared_error(testY[0], testPredict[:,0])
print('Test Score: %.2f MSE' % (testScore_mse))
mae = mean_absolute_error(testPredict[:,0],testY[0])
MAPE = numpy.mean(numpy.abs((testY[0] - testPredict[:,0]) / testPredict[:,0])) * 100
print('The Mean Absolute Percentage Error is {:.2f}%'.format(MAPE))
testY.mean()
print("--- %s seconds ---" % (time.time() - start_time))

# shift train predictions for plotting
trainPredictPlot = numpy.empty_like(dataset)
trainPredictPlot[:, :] = numpy.nan
trainPredictPlot[look_back:len(trainPredict)+look_back, :] = trainPredict

# shift test predictions for plotting
testPredictPlot = numpy.empty_like(dataset)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(trainPredict)+(look_back*2)+1:len(dataset)-1, :] = testPredict
```

```
# plot baseline and predictions
'''
diff=[]
ratio=[]
p = model.predict(testX)
for u in range(len(testY)):
    pr = p[u][0]
    ratio.append((testY[u]/pr)-1)
    diff.append(abs(testY[u]- pr))
    #print(u, y_test[u], pr, (y_test[u]/pr)-1, abs(y_test[u]- pr))
'''

# plot baseline and predictions
plt.plot(scaler.inverse_transform(dataset), label = 'Actual Close')
plt.plot(testPredictPlot,color='red', label = 'Predicted Close')
plt.legend()
dataset.mean()
```

3. ARIMA model:

The code snippet for the ARIMA model is given below.

```
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
import seaborn as sb
from scipy import stats
sb.set_style('darkgrid')
from pmdarima import auto_arima
from statsmodels.tsa.arima_model import ARIMA
from sklearn.metrics import mean_absolute_error,mean_squared_error
import math

#import the csv and store in a dataframe
stock_data = pd.read_csv('C:/Users/Smit/Dataset/yahoo/stockMarket.csv')
newdata = stock_data.set_index('Date')
newdata = newdata.iloc[:,3]
newdata = pd.DataFrame(newdata)
summary = auto_arima(newdata['Close'],start_p=0,
                     start_q=0,max_p=3,max_q=3,seasonal=False,trace=True)
summary.summary()

trian = newdata.iloc[:1200]
test = newdata.iloc[1200:]
start = len(trian)
end = len(trian) + len(test) - 1
model_arima = ARIMA(trian['Close'],order=(0,1,0))
result_arima = model_arima.fit()
prediction= result_arima.predict(start=start,end=end,typ='levels')
prediction= pd.DataFrame(prediction)

test['prediction'] = prediction.values
test.plot()

```

```
mae = mean_absolute_error(prediction,test['Close'])
MAPE = np.mean(np.abs((test['Close'] - test['prediction']) / test['prediction'])) * 100
print('The Mean Absolute Percentage Error is {:.2f}%'.format(MAPE))
newdata.mean()
mse = mean_squared_error(prediction,test['Close'])
rmse = math.sqrt(mse)
```

4. SARIMAX model:

The code snippets for the SARIMAX model is given below.

```
# Import Libraries
import warnings
import itertools
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.metrics import mean_absolute_error,mean_squared_error
import math
from statsmodels.tools.eval_measures import rmse
plt.rcParams['figure.figsize'] = (20.0, 10.0)
plt.rcParams.update({'font.size': 12})
plt.style.use('ggplot')
#################################################
# import the data
dataRead = pd.read_csv('C:/Users/Smit/Dataset/yahoo/stockMarket.csv',
                       engine='python', skipfooter=3)
data = pd.DataFrame()
data['Date'] = dataRead.Date
data['ClosingVal'] = dataRead.Close
data['Date']=pd.to_datetime(data['Date'], format='%d/%m/%Y')
data.set_index(['Date'], inplace=True)
# Plot the data
data.plot()
plt.ylabel('Stock index')
plt.xlabel('Date')
plt.show()
```

```
# Defining d and q parameters
q = d = range(0, 2)
# Defining p parameters
p = range(0, 4)
# Generating different combinations
pdq = list(itertools.product(p, d, q))
# different combinations of seasonal p, q and q
seasonal_pdq = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p, d, q))]
print('Examples of parameter combinations for Seasonal ARIMA...')
print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[1]))
print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[2]))
print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[3]))
print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[4]))
################################################
train_data = data['2013-06-28':'2017-06-28']
test_data = data['2017-06-29':'2018-06-26']
warnings.filterwarnings("ignore") # specify to ignore warning messages
```

```python
AIC = []
SARIMAX_model = []
for param in pdq:
    for param_seasonal in seasonal_pdq:
        try:
            mod = sm.tsa.statespace.SARIMAX(train_data,
                                            order=param,
                                            seasonal_order=param_seasonal,
                                            enforce_stationarity=False,
                                            enforce_invertibility=False)

            results = mod.fit()

            print('SARIMAX{}x{} - AIC:{}'.format(param, param_seasonal,
                  results.aic), end='\r')
            AIC.append(results.aic)
            SARIMAX_model.append([param, param_seasonal])
        except:
            continue

print('The smallest AIC is {} for model SARIMAX{}x{}'.format(min(AIC),
      SARIMAX_model[AIC.index(min(AIC))][0],
      SARIMAX_model[AIC.index(min(AIC))][1]))
```

```python
# model fitting
mod = sm.tsa.statespace.SARIMAX(train_data,
                                order=SARIMAX_model[AIC.index(min(AIC))][0],
                                seasonal_order=SARIMAX_model[AIC.index(min(AIC))][1],
                                enforce_stationarity=False,
                                enforce_invertibility=False)
results = mod.fit()

###########################################################
results.plot_diagnostics(figsize=(20, 14))
plt.show()
###########################################################
##### PREDICTIONS

#pred0 = results.get_prediction(start='2016-06-28', dynamic=False)
pred0 = results.get_prediction(start='2017-06-28', dynamic=False)
pred0_ci = pred0.conf_int()

#pred1 = results.get_prediction(start='2016-06-28', dynamic=True)
#pred1_ci = pred1.conf_int()

date1 = '2017-11-19'
date2 = '2018-07-26'
#date1 = '2018-06-28'
#date2 = '2019-03-04'
mydates = pd.date_range(date1, date2).tolist()

pred2 = results.get_forecast(steps=250,index=mydates) ## steps = 12
pred2_ci = pred2.conf_int()
print(pred2.predicted_mean['2017-11-19':'2018-07-26'])
#print(pred2.predicted_mean['2018-06-28':'2019-03-04'])
```

```
ax = data.plot(figsize=(20, 16))
pred0.predicted_mean.plot(ax=ax, label='1-step-ahead Forecast (get_predictions, dynamic=False)')
#pred1.predicted_mean.plot(ax=ax, label='Dynamic Forecast (get_predictions, dynamic=True)')
pred2.predicted_mean.plot(ax=ax, label='Dynamic Forecast (get_forecast)')
ax.fill_between(pred2_ci.index, pred2_ci.iloc[:, 0], pred2_ci.iloc[:, 1], color='k', alpha=.1)
plt.ylabel('Stock Market Index')
plt.xlabel('Date')
plt.legend()
plt.show()

###########################################################

#prediction = pred2.predicted_mean['2018-06-28':'2019-03-04'].values
prediction = pred2.predicted_mean['2017-06-27':'2018-03-03'].values
# flatten nested list
truth = list(itertools.chain.from_iterable(test_data.values))

error = rmse(truth, prediction)
error

# Mean Absolute Percentage Error
MAPE = np.mean(np.abs((truth - prediction) / truth)) * 100
mae = mean_absolute_error(prediction,truth)

print('The Mean Absolute Percentage Error for the forecast of year 2018-19 is {:.2f}%'.format(MAPE))

mse = mean_squared_error(prediction,truth)
rmse = math.sqrt(mse)

```