

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.

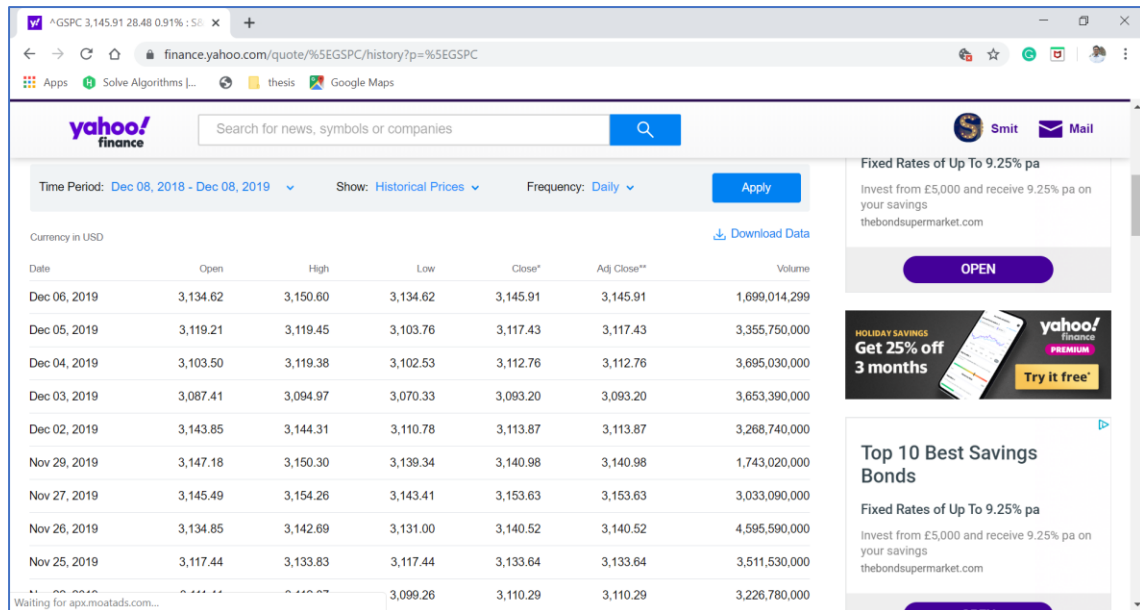


Figure 1: Yahoo!finance data set gathering

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.

```

Anaconda Prompt
(base) C:\Users\Smit>twitterscraper "#SP500" -l 10000 -bd 2018-01-01 -ed 2019-01-01 -o tweets.json
(base) C:\Users\Smit>

```

Figure 2: Twitterscraper, scraping tweets using Anaconda prompt

```

1 import codecs, json, csv
2 import pandas as pd
3
4 #read a json file downloaded with twitterscraper
5 with codecs.open('tweets18SnP500_3.json', 'r', 'utf-8') as f:
6     All_Tweets = json.load(f, encoding='utf-8')
7
8 #tweets is now a list of tweets
9 #save into a csv
10 file = "tweets18SnP500_3.csv" #file name
11
12 #open csv file for append
13 target_file = open(file, 'w', encoding='utf-8', newline='')
14 csv_file = csv.writer(target_file, delimiter=',', quotechar='"')
15 count=0 #a counter
16 i = 0 #a counter
17
18 eng_tweet = pd.DataFrame()
19 eng_tweet['timestamp'] = None
20 eng_tweet['likes'] = None
21 eng_tweet['content'] = None
22 eng_tweet['language'] = None
23
24 for tweet in All_Tweets:
25     eng_tweet.loc[i, 'timestamp'] = tweet['timestamp']
26     eng_tweet.loc[i, 'likes'] = tweet['likes']
27     eng_tweet.loc[i, 'content'] = tweet['text']
28     eng_tweet.loc[i, 'count'] = i
29     if (tweet['html'].find('lang="en"') != -1):
30         eng_tweet.loc[i, 'language'] = "ENGLISH!!!"
31     else:
32         eng_tweet.loc[i, 'language'] = "NON ENGLISH!!!"
33     i = i + 1
34     count=count+1
35
36 #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:

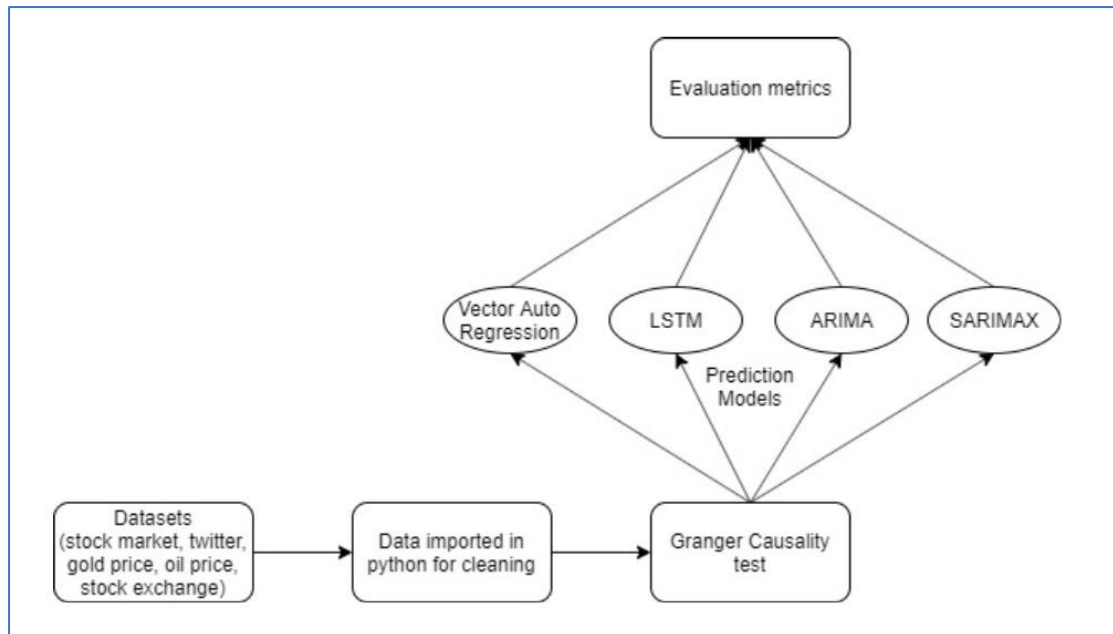


Figure 4: Design flowchart

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.

```
1 print('STOCK PREDICTION USING RNN LSTM')
2 import numpy as np
3 import pandas as pd
4 from sklearn import preprocessing
5 from functools import reduce
6 import matplotlib.pyplot as plt
7 from statsmodels.tsa.api import VAR
8 from statsmodels.tsa.stattools import adfuller
9 from statsmodels.tools.eval_measures import rmse, aic
10 from sklearn.metrics import mean_absolute_error, mean_squared_error
11 import math
12 %matplotlib inline
13 #####
14 # Load the dataset
15 #####
16 SMdata= pd.read_csv('C:/Users/Smit/Dataset/yahoo/stockMarket.csv')
17 TWratio = pd.read_csv('C:/Users/Smit/PosRatioTweets.csv')
18 TWvol = pd.read_csv('C:/Users/Smit/TWvol.csv')
19 #merge all the datasets
20 data_frames = [SMdata, TWratio, TWvol]
21 data_csv = reduce(lambda left,right: pd.merge(left,right,on=['Date'],
22                                             how='outer'), data_frames)
23 data_csv = data_csv.drop(data_csv.index[1260:1640])
24 data_csv.isna().count()
25 data_csv = data_csv.drop('Unnamed: 0', axis=1)
26 data_csv = data_csv.drop('DateTime', axis=1)
27 data_csv = data_csv.fillna(0)
28 data_csv[['Close']].plot()
29 plt.show()
30 plt.clf()
31 #####
32 #VAR model
33 drop = ['Open', 'High', 'Low', 'Adj Close', 'Volume', 'Negative', 'Positive', 'Count']
34 data_csv = data_csv.drop(drop,axis=1)
35 data_csv = data_csv.set_index('Date')
36 data_csv.info()
37 nobs = 30
38 df_train, df_test = data_csv[0:-nobs], data_csv[-nobs:]
39 model = VAR(data_csv)
40 for i in [1,2,3,4,5,6,7,8,9]:
41     result = model.fit(i)
42     print('Lag Order =', i)
43     print('AIC : ', result.aic)
44     print('BIC : ', result.bic)
45     print('FPE : ', result.fpe)
46     print('HQIC : ', result.hqic, '\n')
47 x = model.select_order(maxlags=12)
48 x.summary()
49 model_fitted = model.fit(7)
50 model_fitted.summary()
51 forecast_input = df_train.values
52 fc = model_fitted.forecast(y=forecast_input, steps=nobs)
53 df_forecast = pd.DataFrame(fc, index=data_csv.index[-nobs:], columns=data_csv.columns + '_2d')
54 df_forecast
55 plt.plot(df_train['Close'])
56
57 mse = mean_squared_error(df_forecast['Close_2d'],df_test['Close'])
58 rmse = math.sqrt(mse)
59 mae = mean_absolute_error(df_forecast['Close_2d'],df_test['Close'])
60 MAPE = np.mean(np.abs((df_test['Close'] - df_forecast['Close_2d']) / df_forecast['Close_2d'])) * 100
61 print('The Mean Absolute Percentage Error is {:.2f}%'.format(MAPE))
62
63 msee = mean_squared_error(df_forecast['Ratio_2d'],df_test['Ratio'])
64 rmsee = math.sqrt(msee)
65 mae = mean_absolute_error(df_forecast['Ratio_2d'],df_test['Ratio'])
66 MAPEE = np.mean(np.abs((df_test['Ratio'] - df_forecast['Ratio_2d']) / df_forecast['Ratio_2d'])) * 100
67 print('The Mean Absolute Percentage Error is {:.2f}%'.format(MAPEE))
```

```

68
69 df_test['predicted_close'] = df_forecast['Close_2d']
70 df_test['predicted_ratio'] = df_forecast['Ratio_2d']
71
72 plt.plot(df_test['Close'], label='Actual Close')
73 plt.plot(df_test['predicted_close'], label='Predicted Close')
74 plt.legend()
75
76 plt.plot(df_test['Ratio'], label='Actual Ratio')
77 plt.plot(df_test['predicted_ratio'], label=['Predicted Ratio'])
78 plt.legend()

```

2. LSTM model:

The code snippet for the LSTM model is given below.

```

1 import numpy
2 import matplotlib.pyplot as plt
3 import pandas
4 import math
5 from keras.models import Sequential
6 from keras.layers import Dense
7 from keras.layers import LSTM
8 from sklearn.preprocessing import MinMaxScaler
9 from sklearn.metrics import mean_squared_error, mean_absolute_error
10 import time
11
12 numpy.random.seed(7)
13
14 # Load the dataset
15 dataframe = pandas.read_csv('C:/Users/Smit/Dataset/yahoo/stockMarket.csv',
16                             usecols=[1], engine='python', skipfooter=3)
17 dataset = dataframe.values
18 dataset = dataset.astype('float32')
19

```



```

20 # normalize the data
21 scaler = MinMaxScaler(feature_range=(0, 1))
22 dataset = scaler.fit_transform(dataset)
23
24 #train and test sets
25 train_size = int(len(dataset) * 0.85)
26 test_size = len(dataset) - train_size
27 train, test = dataset[0:train_size,:], dataset[train_size:len(dataset),:]
28 print(len(train), len(test))
29
30 # values into a dataset matrix
31 def create_dataset(dataset, look_back=1):
32     dataX, dataY = [], []
33     for i in range(len(dataset)-look_back-1):
34         a = dataset[i:(i+look_back), 0]
35         dataX.append(a)
36         dataY.append(dataset[i + look_back, 0])
37     return numpy.array(dataX), numpy.array(dataY)
38
39 # reshape into X=t and Y=t+1
40 look_back = 30
41 trainX, trainY = create_dataset(train, look_back)
42 testX, testY = create_dataset(test, look_back)
43
44 # reshape input to be [samples, time steps, features]
45 trainX = numpy.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
46 testX = numpy.reshape(testX, (testX.shape[0], 1, testX.shape[1]))
47 start_time = time.time()
48
49 # create and fit the LSTM model
50 model = Sequential()
51 model.add(LSTM(30, input_shape=(1, look_back)))
52 model.add(Dense(1))
53 model.compile(loss='mean_squared_error', optimizer='adam')
54 model.fit(trainX, trainY, epochs=10, batch_size=1, verbose=2)

```

```

56 # make predictions
57 trainPredict = model.predict(trainX)
58 testPredict = model.predict(testX)
59
60 # invert predictions
61 trainPredict = scaler.inverse_transform(trainPredict)
62 trainY = scaler.inverse_transform([trainY])
63 testPredict = scaler.inverse_transform(testPredict)
64 testY = scaler.inverse_transform([testY])
65
66 # calculate root mean squared error
67 trainScore = math.sqrt(mean_squared_error(trainY[0], trainPredict[:,0]))
68 print('Train Score: %.2f RMSE' % (trainScore))
69 testScore = math.sqrt(mean_squared_error(testY[0], testPredict[:,0]))
70 print('Test Score: %.2f RMSE' % (testScore))
71 testScore_mse = mean_squared_error(testY[0], testPredict[:,0])
72 print('Test Score: %.2f MSE' % (testScore_mse))
73 mae = mean_absolute_error(testPredict[:,0], testY[0])
74 MAPE = numpy.mean(numpy.abs((testY[0] - testPredict[:,0]) / testPredict[:,0])) * 100
75 print('The Mean Absolute Percentage Error is {:.2f}%'.format(MAPE))
76 testY.mean()
77 print("--- %s seconds ---" % (time.time() - start_time))
78
79 # shift train predictions for plotting
80 trainPredictPlot = numpy.empty_like(dataset)
81 trainPredictPlot[:, :] = numpy.nan
82 trainPredictPlot[look_back:len(trainPredict)+look_back, :] = trainPredict
83
84 # shift test predictions for plotting
85 testPredictPlot = numpy.empty_like(dataset)
86 testPredictPlot[:, :] = numpy.nan
87 testPredictPlot[len(trainPredict)+(look_back*2)+1:len(dataset)-1, :] = testPredict
88

```

```

89 # plot baseline and predictions
90 '''
91 diff=[]
92 ratio=[]
93 p = model.predict(testX)
94 for u in range(len(testY)):
95     pr = p[u][0]
96     ratio.append((testY[u]/pr)-1)
97     diff.append(abs(testY[u]- pr))
98     #print(u, y_test[u], pr, (y_test[u]/pr)-1, abs(y_test[u]- pr))
99 ...
100
101 # plot baseline and predictions
102 plt.plot(scaler.inverse_transform(dataset), label = 'Actual Close')
103 plt.plot(testPredictPlot,color='red', label = 'Predicted Close')
104 plt.legend()
105 dataset.mean()

```

3. ARIMA model:

The code snippet for the ARIMA model is given below.

```

1 %matplotlib inline
2 import numpy as np
3 import pandas as pd
4 import matplotlib.pyplot as plt
5 import statsmodels.api as sm
6 import seaborn as sb
7 from scipy import stats
8 sb.set_style('darkgrid')
9 from pmdarima import auto_arima
10 from statsmodels.tsa.arima_model import ARIMA
11 from sklearn.metrics import mean_absolute_error, mean_squared_error
12 import math
13
14 #import the csv and store in a dataframe
15 stock_data = pd.read_csv('C:/Users/Smit/Dataset/yahoo/stockMarket.csv')
16 newdata = stock_data.set_index('Date')
17 newdata = newdata.iloc[:,3]
18 newdata = pd.DataFrame(newdata)
19 summary = auto_arima(newdata['Close'],start_p=0,
20                     start_q=0,max_p=3,max_q=3,seasonal=False,trace=True)
21 summary.summary()
22
23 trian = newdata.iloc[:1200]
24 test = newdata.iloc[1200:]
25 start = len(trian)
26 end = len(trian) + len(test) - 1
27 model_arima = ARIMA(trian['Close'],order=(0,1,0))
28 result_arima = model_arima.fit()
29 prediction= result_arima.predict(start=start,end=end,typ='levels')
30 prediction= pd.DataFrame(prediction)
31
32 test['prediction'] = prediction.values
33 test.plot()
34
35 mae = mean_absolute_error(prediction,test['Close'])
36 MAPE = np.mean(np.abs((test['Close'] - test['prediction']) / test['prediction'])) * 100
37 print('The Mean Absolute Percentage Error is {:.2f}%'.format(MAPE))
38 newdata.mean()
39 mse = mean_squared_error(prediction,test['Close'])
40 rmse = math.sqrt(mse)

```

4. SARIMAX model:

The code snippets for the SARIMAX model is given below.

```
1 # Import Libraries
2 import warnings
3 import itertools
4 import pandas as pd
5 import numpy as np
6 import matplotlib.pyplot as plt
7 import statsmodels.api as sm
8 from sklearn.metrics import mean_absolute_error, mean_squared_error
9 import math
10 from statsmodels.tools.eval_measures import rmse
11
12 plt.rcParams['figure.figsize'] = (20.0, 10.0)
13 plt.rcParams.update({'font.size': 12})
14 plt.style.use('ggplot')
15
16 #####
17 # import the data
18 dataRead = pd.read_csv('C:/Users/Smit/Dataset/yahoo/stockMarket.csv',
19                       engine='python', skipfooter=3)
20 data = pd.DataFrame()
21 data['Date'] = dataRead.Date
22 data['ClosingVal'] = dataRead.Close
23 data['Date'] = pd.to_datetime(data['Date'], format='%d/%m/%Y')
24 data.set_index(['Date'], inplace=True)
25
26 # Plot the data
27 data.plot()
28 plt.ylabel('Stock index')
29 plt.xlabel('Date')
30 plt.show()
```

```
32 # Defining d and q parameters
33 q = d = range(0, 2)
34 # Defining p parameters
35 p = range(0, 4)
36 # Generating different combinations
37 pdq = list(itertools.product(p, d, q))
38
39 # different combinations of seasonal p, q and q
40 seasonal_pdq = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p, d, q))]
41 print('Examples of parameter combinations for Seasonal ARIMA...')
42 print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[1]))
43 print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[2]))
44 print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[3]))
45 print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[4]))
46 #####
47
48 train_data = data['2013-06-28': '2017-06-28']
49 test_data = data['2017-06-29': '2018-06-26']
50 warnings.filterwarnings("ignore") # specify to ignore warning messages
51
```

```

52 AIC = []
53 SARIMAX_model = []
54 for param in pdq:
55     for param_seasonal in seasonal_pdq:
56         try:
57             mod = sm.tsa.statespace.SARIMAX(train_data,
58                                             order=param,
59                                             seasonal_order=param_seasonal,
60                                             enforce_stationarity=False,
61                                             enforce_invertibility=False)
62         results = mod.fit()
63
64         print('SARIMAX{x{x}} - AIC:{}'.format(param, param_seasonal,
65                                             results.aic), end='\n')
66         AIC.append(results.aic)
67         SARIMAX_model.append([param, param_seasonal])
68     except:
69         continue
70
71
72 print('The smallest AIC is {} for model SARIMAX{x{x}}'.format(min(AIC),
73     SARIMAX_model[AIC.index(min(AIC))][0],
74     SARIMAX_model[AIC.index(min(AIC))][1]))

```

```

76 # model fitting
77 mod = sm.tsa.statespace.SARIMAX(train_data,
78     order=SARIMAX_model[AIC.index(min(AIC))][0],
79     seasonal_order=SARIMAX_model[AIC.index(min(AIC))][1],
80     enforce_stationarity=False,
81     enforce_invertibility=False)
82 results = mod.fit()
83
84 #####
85 results.plot_diagnostics(figsize=(20, 14))
86 plt.show()
87 #####
88 ##### PREDICTIONS
89
90 #pred0 = results.get_prediction(start='2016-06-28', dynamic=False)
91 pred0 = results.get_prediction(start='2017-06-28', dynamic=False)
92 pred0_ci = pred0.conf_int()
93
94 #pred1 = results.get_prediction(start='2016-06-28', dynamic=True)
95 #pred1_ci = pred1.conf_int()
96
97 date1 = '2017-11-19'
98 date2 = '2018-07-26'
99 #date1 = '2018-06-28'
100 #date2 = '2019-03-04'
101 mydates = pd.date_range(date1, date2).tolist()
102
103 pred2 = results.get_forecast(steps=250, index=mydates) ## steps = 12
104 pred2_ci = pred2.conf_int()
105 print(pred2.predicted_mean['2017-11-19': '2018-07-26'])
106 #print(pred2.predicted_mean['2018-06-28': '2019-03-04'])
107

```

```

110 ax = data.plot(figsize=(20, 16))
111 pred0.predicted_mean.plot(ax=ax, label='1-step-ahead Forecast (get_predictions, dynamic=False)')
112 #pred1.predicted_mean.plot(ax=ax, label='Dynamic Forecast (get_predictions, dynamic=True)')
113 pred2.predicted_mean.plot(ax=ax, label='Dynamic Forecast (get_forecast)')
114 ax.fill_between(pred2_ci.index, pred2_ci.iloc[:, 0], pred2_ci.iloc[:, 1], color='k', alpha=.1)
115 plt.ylabel('Stock Market Index')
116 plt.xlabel('Date')
117 plt.legend()
118 plt.show()
119
120 #####
121
122 #prediction = pred2.predicted_mean['2018-06-28':'2019-03-04'].values
123 prediction = pred2.predicted_mean['2017-06-27':'2018-03-03'].values
124 # flatten nested list
125 truth = list(itertools.chain.from_iterable(test_data.values))
126
127 error = rmse(truth, prediction)
128 error
129
130 # Mean Absolute Percentage Error
131 MAPE = np.mean(np.abs((truth - prediction) / truth)) * 100
132 mae = mean_absolute_error(prediction, truth)
133
134 print('The Mean Absolute Percentage Error for the forecast of year 2018-19 is {:.2f}%'.format(MAPE))
135
136 mse = mean_squared_error(prediction, truth)
137 rmse = math.sqrt(mse)
138

```