



In [ ]:

```
1  #####
2  #
3  #           Assignment : Project On TimeSeries
4  #
5  #####
6  # Problem Statement : For Following Stocks Forecast to be done
7  # 1. NASDAQ.AAPL
8  # 2. NASDAQ.ADP
9  # 3. NASDAQ.CBOE
10 # 4. NASDAQ.CSCO
11 # 5. NASDAQ.EBAY
12
13 # Import the Packages needed
14 import pandas as pd
15 import numpy as np
16 import matplotlib.pyplot as plt
17 from pandas.tools.plotting import autocorrelation_plot
18 from statsmodels.graphics.tsaplots import plot_pacf
19 from statsmodels.tsa.arima_model import ARIMA, ARMAResults
20 import datetime
21 import sys
22 import seaborn as sns
23 import statsmodels
24 import statsmodels.stats.diagnostic as diag
25 from statsmodels.tsa.stattools import adfuller
26 from scipy.stats.mstats import normaltest
27 import statsmodels.api as sm
28
29 from matplotlib.pyplot import acorr
30 import warnings
31
32
33 # to remove the warnings coming in the output
34 if __name__ == '__main__':
35     warnings.filterwarnings(action='ignore', category=Warning)
36
37
38 plt.style.use('fivethirtyeight')
39 %matplotlib inline
40
41 # Loading the DataSet
```

```
42 print( " The Data Set of Stocks information for time frame \n",'-'*80)  
43 df = pd.read_csv('data_stocks.csv')  
44 df.head()  
45
```

```
In [ ]: 1 # Analysing the Data , to see if there are null , missing values ,as well to see fields having big variations  
2 # or constant variations  
3 print(df.describe())  
4 print('-'*80)  
5  
6 print(df.info())  
7 print('-'*80)  
8  
9 print(df.isnull().sum())  
10 print('-'*80)  
11  
12 print(df.shape)
```

```

In [ ]: 1 # to check the stationarity , we need to check the rolling mean and Dicker fuller test
2 from statsmodels.tsa.stattools import adfuller
3
4 # get_mean_var function gives the mean and the variance for the values
5 def get_mean_var(series, no_of_samples):
6     split_size = int(len(series) / no_of_samples)
7     start = 0
8     for i in range(no_of_samples):
9         sample_series = series[i*split_size:(i+1)*split_size]
10        #print(sample_series, "\n")
11        print('Mean= %.2f, Variance= %.2f' % (sample_series.mean(), sample_series.var()))
12
13
14 def plot_rolling_statistics(timeseries):
15     #Determining rolling statistics
16     rolmean = timeseries.rolling(window=500).mean()
17     rolstd = timeseries.rolling(window=500).std()
18
19     #Plot rolling statistics:
20     plt.plot(timeseries, color='blue', label='Original')
21     plt.plot(rolmean, color='red', label='Rolling Mean')
22     plt.plot(rolstd, color='black', label = 'Rolling Std')
23
24     plt.legend(loc='best')
25     plt.title('Rolling Mean & Standard Deviation')
26     plt.figure(figsize=(20,20))
27
28     plt.show()
29
30
31
32 def dickey_fuller_test(timeseries):
33     #Perform Dickey-Fuller test:
34     print('Results of Dickey-Fuller Test:')
35     dftest = adfuller(timeseries, autolag='AIC') # Akaike information criterion
36
37     # https://coolstatsblog.com/2013/08/14/using-aic-to-test-arma-models-2/
38     #https://www.statsmodels.org/dev/generated/statsmodels.tsa.stattools.adfuller.html
39     #statsmodels.tsa.stattools.adfuller(x, maxlag=None, regression='c', autolag='AIC',
40     # store=False, regresults=False)
41

```

```

42 dfoutput = pd.Series(dfctest[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Observations Used'])
43
44 for key,value in dfctest[4].items():
45     dfoutput['Critical Value (%s)'%key] = value
46 print(dfoutput)
47
48
49 def test_stationarity(timeseries):
50     no_of_samples = 10
51     print(" Printing the Mean and Variance for the Stock \n",'-'*80)
52     get_mean_var(timeseries, no_of_samples)
53     print(" \nPrinting results Rolling statistics and Dicker Fuller test Values\n",'-'*80)
54     plot_rolling_statistics(timeseries)
55     dickey_fuller_test(timeseries)
56
57
58

```

In [ ]:

```

1  #####
2  # 1.          Forecasting for Stock NASDAQ.AAPL
3  #####
4
5  # Step 1. Visualizing the Stock Values
6  df['NASDAQ.AAPL'].plot(figsize=(10, 6))
7
8  # we observe from the plot , there is upward trend , so we will make it stationary
9  print(" We observe from the plot , Stock -NASDAQ.AAPL has upward trend\n " , '-'*80)

```

```
In [ ]: 1 # get_mean_var(df['NASDAQ.AAPL'].values, 10)
        2 # from the values , we observe that mean , variance are not constant
        3 test_stationarity(df['NASDAQ.AAPL'])
        4
        5 # we Observe from test stationarity results that
        6 # 1. Pvalue is greater than zero
        7 # 2. test statics need to be less than critical ,
        8 # as all conditons not met , we need to stationarize the series
        9 print('-'*80," \nWE Observe from tests that \n",'-'*80)
       10 print(''
       11 1. Pvalue is greater than zero
       12 2. test statics need to be less than critical ,
       13 as all conditons not met , we need to stationarize the series
       14
       15 ''')
```

```
In [ ]: 1 # Step2 : Stationarize the Stock by ,trying the shift technique
2 import statsmodels.api as sm
3 df['AAPL_SHIFT'] = df['NASDAQ.AAPL']-df['NASDAQ.AAPL'].shift()
4 print( " Stationarizing the Stock Series \n",'-'*80)
5
6 print(" Printing the First Difference Values\n",'-'*80)
7 print(df['AAPL_SHIFT'].head())
8
9 df['AAPL_SHIFT'].dropna(inplace=True)
10
11 # from the values , we observe that mean , variance are not constant
12 test_stationarity(df['AAPL_SHIFT'])
13 # we observe that Pvalue is zero , and test statistics is < Critical Value and
14 sm.stats.durbin_watson(df['AAPL_SHIFT'])
15 # the durbin_watson value shows no correlation as value =2
16 # series is stationary
17
18 print('' \nFrom Tests We observe \n'', '-'*80, ''
19 1. Pvalue is equal to zero
20 2. test statics less than critical ,
21 3. Durbin watson test =2 , shows no coorelation
22 as all conditons met , we can consider series stationarized '')
23
```

```
In [ ]: 1 # function to get the best model
2
3 #ararray = df['NASDAQ.AAPL']
4 def bst_TSMModel(ararray):
5     p=0
6     q=0
7     d=1
8
9     pdq=[]
10    aic=[]
11
12    for p in range(2):
13        for q in range(2):
14            try:
15                model = sm.tsa.ARIMA(ararray, (p,d,q)).fit()
16                x = model.aic
17                x1 = (p,d,q)
18
19                print (x1, x)
20                aic.append(x)
21                pdq.append(x1)
22            except:
23                print('error')
24                pass
25
26    #print(pdq,aic)
27    keys = pdq
28    values = aic
29    d = dict(zip(keys, values))
30
31    # Best Model
32
33    minaic=min(d, key=d.get)
34    print ("\nBest Model is :", minaic)
35
```



```
In [ ]: 1 # Step3 : To optimize the parameters we use ACF and PACF
2 %matplotlib inline
3 fig = plt.figure(figsize=(12,8))
4
5 ax1 = fig.add_subplot(211)
6 fig = sm.graphics.tsa.plot_acf(df['NASDAQ.AAPL'].values.squeeze(), lags=30, ax=ax1)
7
8 ax2 = fig.add_subplot(212)
9 fig = sm.graphics.tsa.plot_pacf(df['NASDAQ.AAPL'].values.squeeze(), lags=70, ax=ax2)
10
11 bst_TSMModel(df['AAPL_SHIFT'])
12
13 print("\n The Optimised Paramters by using ACF and PACF for model \n", '-'*80)
14 print(" From the PACF we see that p= 1 (lags > 1 are all zeros)")
```

```
In [ ]: 1 # Step4 : to fit the model and predict
2 df['AAPL_SHIFT'].dropna(inplace=True)
3 model = sm.tsa.ARIMA(df['AAPL_SHIFT'], order=(1, 1, 0))
4 results = model.fit(dis=-1)
5
6 print( " The ARIMA Model Fitted and Forecasted Values plotted \n", '-'*80)
7
8 df['Forecast'] = results.fittedvalues
9
10 df[['AAPL_SHIFT', 'Forecast']].plot(figsize=(10, 6))
```

```
In [ ]: 1 #print(df['Forecast'].head())
2 predictions = pd.Series(results.fittedvalues,copy=True)
3
4 # taking cumulativeSum
5 predictions_ARIMA_cumsum = predictions.cumsum()
6
7 predictions_ARIMA_Forecast= pd.Series(df['NASDAQ.AAPL'].values)
8 predictions_ARIMA_Forecast = predictions_ARIMA_Forecast.add(predictions_ARIMA_cumsum,fill_value=0)
9
10 print("\n Plotting the Original and Forecasted Stocks \n",'-'*80)
11
12 predictions_ARIMA_Forecast.head()
13
14 df['NASDAQ.AAPL_FORECAST'] = predictions_ARIMA_Forecast
15 df[['NASDAQ.AAPL', 'NASDAQ.AAPL_FORECAST']].plot(figsize=(10, 6))
16
17
18
```

```
In [ ]: 1 # predictions = pd.Series(results.fittedvalues,copy=True)
2 # print(predictions.head())
3 from sklearn.metrics import mean_squared_error
4 def mean_forecast_err(y, yhat):
5     return y.sub(yhat).mean()
6
7 def mean_absolute_err(y, yhat):
8     return np.mean((np.abs(y.sub(yhat).mean()) / yhat)) # or percent error = * 100
9
10
11
12 print("Mean ForeCast Error - MFE = ",mean_forecast_err(df['NASDAQ.AAPL'], df['NASDAQ.AAPL_FORECAST']))
13 print("Mean Absolute Error - MAE = ",mean_absolute_err(df['NASDAQ.AAPL'], df['NASDAQ.AAPL_FORECAST']))
14 print("Mean Square Error - MSE = ",mean_squared_error(df['NASDAQ.AAPL'], df['NASDAQ.AAPL_FORECAST']))
15
```

```

In [ ]: 1 # Step1 : For Stock ADP , doing the Forecasting
        2 #####
        3 # 2.          Forecasting for Stock NASDAQ.ADP
        4 #####
        5 df['NASDAQ.ADP'].plot(figsize=(8, 6))
        6 # from the Graph We observe there is slight upward trend and stock Value is not stationary
        7 print(" We observe from the plot , Stock -NASDAQ.ADP has upward trend\n " , "-"*80)

```

```

In [ ]: 1 #get_mean_var(df['NASDAQ.ADP'].values, 10)
        2 # from the values , we observe that mean , variance are not constant
        3 test_stationarity(df['NASDAQ.ADP'])
        4 # Durbin watson test for stationarity
        5 print('-'*80, '\n', 'DurbinWatson test : ', sm.stats.durbin_watson(df['NASDAQ.ADP']))
        6 # We observe durbin watson Value > 2 , so there is negative correlation
        7
        8
        9 # we Observe from test stationarity results that
       10 # 1. Pvalue is greater than zero
       11 # 2. test statics > than critical ,
       12 # as all conditons not met , we need to stationarize the series
       13 print('-'*80, " \nWE Observe from tests that \n", '-'*80)
       14 print(''
       15 1. Pvalue is greater than zero
       16 2. test statics need to be less than critical ,
       17 as all conditons not met , we need to stationarize the series
       18
       19 ''')

```

```
In [ ]: 1 # Step2 : Stationarize the Stock by ,trying the shift technique
2 import statsmodels.api as sm
3 df['ADP_SHIFT'] = df['NASDAQ.ADP']-df['NASDAQ.ADP'].shift()
4
5 print(" Printing the First Difference Values\n",'-'*80)
6 print( " Stationarizing the Stock Series \n",'-'*80)
7
8 print(df['ADP_SHIFT'].head())
9
10 df['ADP_SHIFT'].dropna(inplace=True)
11
12 #get_mean_var(df['ADP_SHIFT'].values, 10)
13 # from the values , we observe that mean , variance are not constant
14 test_stationarity(df['ADP_SHIFT'])
15 # we observe that Pvalue is zero , and test statistics is < Critical Value and
16 print('-'*80,'\nDurbin Watson Test : ',sm.stats.durbin_watson(df['ADP_SHIFT']))
17 # the durbin_watson value shows no correlation as value =2
18 # series is stationary
19
20 print('' \nFrom Tests We observe \n'', '-'*80, ''
21 1. Pvalue is equal to zero
22 2. test statics less than critical ,
23 3. Durbin Watson test =2 , no correlation
24 as all conditons met , we can consider series stationarized '')
25
```

```
In [ ]: 1 # Step3 : To optimize the parameters we use ACF and PACF
2 %matplotlib inline
3 fig = plt.figure(figsize=(12,8))
4
5 ax1 = fig.add_subplot(211)
6 fig = sm.graphics.tsa.plot_acf(df['NASDAQ.ADP'].values.squeeze(), lags=40, ax=ax1)
7
8 ax2 = fig.add_subplot(212)
9 fig = sm.graphics.tsa.plot_pacf(df['NASDAQ.ADP'].values.squeeze(), lags=40, ax=ax2)
10
11
12
13 # finding the best fit model
14 bst_TSMModel(df['ADP_SHIFT'])
15
16 print("\n The Optimised Paramters by using ACF and PACF for model \n", '-'*80)
```

```
In [ ]: 1 # Step4 : to fit the model and predict
2 df['ADP_SHIFT'].dropna(inplace=True)
3 model = sm.tsa.ARIMA(df['ADP_SHIFT'], order=(2, 1, 3))
4 results = model.fit(disp=-1)
5
6 print( " The ARIMA Model Fitted and Forecasted Values plotted \n", '-'*80)
7
8 df['Forecast'] = results.fittedvalues
9 df[['ADP_SHIFT', 'Forecast']].plot(figsize=(10, 8))
```

```
In [ ]: 1 print(df['Forecast'].head())
2 predictions = pd.Series(results.fittedvalues,copy=True)
3 # print(predictions_ARIMA_diff.head())
4 # print('-'*80)
5 # taking cumulativeSum
6 predictions_ARIMA_cumsum = predictions.cumsum()
7 print(predictions_ARIMA_cumsum.head())
8 predictions_ARIMA_Forecast= pd.Series(df['NASDAQ.ADP'].values)
9 predictions_ARIMA_Forecast = predictions_ARIMA_Forecast.add(predictions_ARIMA_cumsum,fill_value=0)
10 print(predictions_ARIMA_Forecast.head())
11
12 print(" Plotting the Original and Forecasted Stocks \n",'-'*80)
13 df['NASDAQ.ADP_FORECAST'] = predictions_ARIMA_Forecast
14
15 df[['NASDAQ.ADP', 'NASDAQ.ADP_FORECAST']].plot(figsize=(10, 8))
```

```
In [ ]: 1 #*****
2 #           Forecasting for Stock NASDAQ.CBOE
3 #*****
4 # Step 1. Visualizing the Stock Values
5
6 df['NASDAQ.CBOE'].plot(figsize=(8, 6))
7 # we observe from the plot , there is upward trend , so we will make it stationary
8 print(" We observe from the plot , Stock -NASDAQ.CBOE has upward trend\n " , '-'*80)
```

```
In [ ]: 1 # from the values , we observe that mean , variance are not constant
2 test_stationarity(df['NASDAQ.CBOE'])
3
4 print('-'*80," \nWE Observe from tests that \n",'-'*80)
5 print(''
6 1. Pvalue is greater than zero
7 2. test statics need to be less than critical ,
8 as all conditons not met , we need to stationarize the series
9
10 ''')
11
```

```

In [ ]: 1 # Step2 : Stationarize the Stock by ,trying the shift technique Log
        2
        3 print( " Stationarizing the Stock Series \n",'-'*80)
        4 import statsmodels.api as sm
        5 df['CBOE_SHIFT'] = df['NASDAQ.CBOE']-df['NASDAQ.CBOE'].shift()
        6
        7 print(" Printing the First Difference Values\n",'-'*80)
        8 print(df['CBOE_SHIFT'].head())
        9
       10 df['CBOE_SHIFT'].dropna(inplace=True)
       11
       12 # from the values , we observe that mean , variance are not constant
       13 test_stationarity(df['CBOE_SHIFT'])
       14 # we observe that Pvalue is zero , and test statistics is < Critical Value and
       15 print('-'*80,'\nDurbin Watson test :',sm.stats.durbin_watson(df['CBOE_SHIFT']))
       16 # the durbin_watson value shows no correlation as value =2
       17 # series is stationary
       18 print('' \nFrom Tests We observe \n'', '-'*80, ''
       19 1. Pvalue is equal to zero
       20 2. test statics less than critical ,
       21 3. Durbin watson test =2 , no correlation
       22 as all conditons met , we can consider series stationarized '')
       23

```

```

In [ ]: 1 # Step3 : To optimize the parameters we use ACF and PACF
        2 %matplotlib inline
        3 fig = plt.figure(figsize=(10,6))
        4
        5 ax1 = fig.add_subplot(211)
        6 fig = sm.graphics.tsa.plot_acf(df['NASDAQ.CBOE'].values.squeeze(), lags=40, ax=ax1)
        7
        8 ax2 = fig.add_subplot(212)
        9 fig = sm.graphics.tsa.plot_pacf(df['NASDAQ.CBOE'].values.squeeze(), lags=40, ax=ax2)
       10
       11 # finding the best fit model
       12 bst_TSMModel(df['NASDAQ.CBOE'])
       13 print("\n The Optimised Paramters by using ACF and PACF for model \n",'-'*80)

```

```

In [ ]: 1 # Step4 : to fit the model and predict
        2 df['CBOE_SHIFT'].dropna(inplace=True)
        3 #print(df['CBOE_SHIFT'].head())
        4 model = sm.tsa.ARIMA(df['CBOE_SHIFT'], order=(1, 1, 0))
        5 results = model.fit()
        6
        7 print( " The ARIMA Model Fitted and Forecasted Values plotted \n", '-'*80)
        8 df['Forecast'] = results.fittedvalues
        9 df[['CBOE_SHIFT', 'Forecast']].plot(figsize=(10, 6))
       10
       11

```

```

In [ ]: 1
        2 predictions = pd.Series(results.fittedvalues,copy=True)
        3 predictions_ARIMA_cumsum = predictions.cumsum()
        4
        5 predictions_ARIMA_Forecast= pd.Series(df['NASDAQ.CBOE'].values)
        6
        7 predictions_ARIMA = predictions_ARIMA_Forecast.add(predictions_ARIMA_cumsum,fill_value=0)
        8
        9 print(" Plotting the Original and Forecasted Stocks \n", '-'*80)
       10 df['NASDAQ.CBOE_FORECAST'] = predictions_ARIMA
       11 df[['NASDAQ.CBOE', 'NASDAQ.CBOE_FORECAST']][:20000].plot(figsize=(10, 6))
       12

```

```

In [ ]: 1 *****
        2 # 4.           Forecasting for Stock NASDAQ.CSCO
        3 *****
        4 # Step 1. Visualizing the Stock Values
        5
        6 df['NASDAQ.CSCO'].plot(figsize=(6, 4))
        7
        8 # we observe from the plot , there is downward trend ,and stock not stationary
        9 # so we will need to make it stationary
       10 print(" We observe from the plot , Stock -NASDAQ.CSCO has downward trend\n " , "-"*80)
       11

```



```
In [ ]: 1 # from the values , we observe that mean , variance are not constant
2 test_stationarity(df['NASDAQ.CSCO'])
3
4 # we Observe from test stationarity results that
5 # 1. Pvalue is greater than zero
6 # 2. test statics need to be less than critical ,
7 # as all conditons not met , we need to stationarize the series
8 print('-'*80," \nWE Observe from tests that \n",'-'*80)
9 print(''
10 1. Pvalue is greater than zero
11 2. test statics need to be less than critical ,
12 as all conditons not met , we need to stationarize the series
13
14 ''')
15
```

```
In [ ]: 1 # Step2 : Stationarize the Stock by ,trying the shift technique
2 import statsmodels.api as sm
3
4 # df['CBOE_Log'] = df['NASDAQ.CBOE'].apply(lambda x: np.Log(x))
5 # df['CBOE_Log'].plot(figsize=(16, 12))
6
7 print( " Stationarizing the Stock Series \n",'-'*80)
8 df['CSCO_SHIFT'] = df['NASDAQ.CSCO']-df['NASDAQ.CSCO'].shift()
9
10 print(" Printing the First Difference Values\n",'-'*80)
11 print(df['CSCO_SHIFT'].head())
12
13 df['CSCO_SHIFT'].dropna(inplace=True)
14
15 #get_mean_var(df['CSCO_SHIFT'].values, 10)
16 # from the values , we observe that mean , variance are not constant
17 test_stationarity(df['CSCO_SHIFT'])
18 # we observe that Pvalue is zero , and test statistics is < Critical Value and
19 print('-'*80,'\n Durbin Watson Test : ',sm.stats.durbin_watson(df['CSCO_SHIFT']))
20 # the durbin_watson value shows no correlation as value =2
21 # series is stationary
22
23 print('' \nFrom Tests We observe \n'', '-'*80, ''
24 1. Pvalue is equal to zero
25 2. test statics less than critical ,
26 3. The Durbin Watson almost = 2 , no co-relation
27 as all conditons met , we can consider series stationarized '')
28
29
```

```
In [ ]: 1 # Step3 : To optimize the parameters we use ACF and PACF
2 %matplotlib inline
3 fig = plt.figure(figsize=(6,4))
4
5 ax1 = fig.add_subplot(211)
6 fig = sm.graphics.tsa.plot_acf(df['NASDAQ.CSCO'].values.squeeze(), lags=40, ax=ax1)
7
8 ax2 = fig.add_subplot(212)
9 fig = sm.graphics.tsa.plot_pacf(df['NASDAQ.CSCO'].values.squeeze(), lags=40, ax=ax2)
10
11 # finding the best fit model
12 bst_TSMModel(df['NASDAQ.CSCO'])
13 print("\n The Optimised Paramters by using ACF and PACF for model \n", '-'*80)
```

```
In [ ]: 1 # Step4 : to fit the model and predict
2 df['CSCO_SHIFT'].dropna(inplace=True)
3 print(df['CSCO_SHIFT'].head())
4 model = sm.tsa.ARIMA(df['CSCO_SHIFT'], order=(1, 1, 0))
5 results = model.fit(dis=-1)
6
7 print( " The ARIMA Model Fitted and Forecasted Values plotted \n", '-'*80)
8 df['Forecast'] = results.fittedvalues
9 df[['CSCO_SHIFT', 'Forecast']].plot(figsize=(6, 4))
10
11
```

```

In [ ]: 1 print(df['Forecast'].head())
        2
        3 predictions = pd.Series(results.fittedvalues,copy=True)
        4 # print(predictions_ARIMA_diff.head())
        5 # print('- '*80)
        6 # taking cumulativeSum
        7 predictions_ARIMA_cumsum = predictions.cumsum()
        8 print(predictions_ARIMA_cumsum.head())
        9
       10
       11 predictions_ARIMA_Forecast= pd.Series(df['NASDAQ.CSCO'].values)
       12
       13 predictions_ARIMA = predictions_ARIMA_Forecast.add(predictions_ARIMA_cumsum,fill_value=0)
       14 # predictions_ARIMA_Log.head()
       15 # print(predictions_ARIMA_Log.head())
       16
       17
       18 # predictions_ARIMA = np.exp(predictions_ARIMA_Log)
       19 print(predictions_ARIMA.head())
       20
       21 print(" Plotting the Original and Forecasted Stocks \n",'- '*80)
       22 df['NASDAQ.CSCO_FORECAST'] = predictions_ARIMA
       23 df[['NASDAQ.CSCO', 'NASDAQ.CSCO_FORECAST']][:20000].plot(figsize=(6,4))
       24

```

```

In [ ]: 1 #*****
        2 # 5.           Forecasting for Stock NASDAQ.EBAY
        3 #*****
        4
        5 # Step 1. Visualizing the Stock Values
        6 df['NASDAQ.EBAY'].plot(figsize=(6,4))
        7
        8 # we observe from the plot , there is upward trend , so we will make it stationary
        9 print(" We observe from the plot , Stock -NASDAQ.EBAY has upward trend\n " , "-"*80)

```

```
In [ ]: 1 # get_mean_var(df['NASDAQ.EBAY'].values, 10)
        2 # from the values , we observe that mean , variance are not constant
        3 test_stationarity(df['NASDAQ.EBAY'])
        4
        5 print('-'*80," \nWE Observe from tests that \n",'-'*80)
        6 print(''
        7 1. Pvalue is greater than zero
        8 2. test statics need to be less than critical ,
        9 as all conditons not met , we need to stationarize the series
       10
       11 ''')
```

```
In [ ]: 1 # Step2 : Stationarize the Stock by ,trying the shift technique
2 import statsmodels.api as sm
3
4 # df['CBOE_Log'] = df['NASDAQ.CBOE'].apply(lambda x: np.Log(x))
5 # df['CBOE_Log'].plot(figsize=(16, 12))
6
7 print( " Stationarizing the Stock Series \n",'-'*80)
8 df['EBAY_SHIFT'] = df['NASDAQ.EBAY']-df['NASDAQ.EBAY'].shift()
9
10 print(" Printing the First Difference Values\n",'-'*80)
11 print(df['EBAY_SHIFT'].head())
12
13 df['EBAY_SHIFT'].dropna(inplace=True)
14
15 #get_mean_var(df['EBAY_SHIFT'].values, 10)
16 # from the values , we observe that mean , variance are not constant
17 test_stationarity(df['EBAY_SHIFT'])
18 # we observe that Pvalue is zero , and test statistics is < Critical Value and
19 print('-'*80,'\nDurbin Watson Test : ',sm.stats.durbin_watson(df['EBAY_SHIFT']))
20 # the durbin_watson value shows no correlation as value =2
21 # series is stationary
22
23 print('' \nFrom Tests We observe \n'', '-'*80, ''
24 1. Pvalue is equal to zero
25 2. test statics less than critical ,
26 as all conditons met , we can consider series stationarized '')
27
28
```

```
In [ ]: 1 # Step3 : To optimize the parameters we use ACF and PACF
2 %matplotlib inline
3 fig = plt.figure(figsize=(6,4))
4
5 ax1 = fig.add_subplot(211)
6 fig = sm.graphics.tsa.plot_acf(df['NASDAQ.EBAY'].values.squeeze(), lags=40, ax=ax1)
7
8 ax2 = fig.add_subplot(212)
9 fig = sm.graphics.tsa.plot_pacf(df['NASDAQ.EBAY'].values.squeeze(), lags=40, ax=ax2)
10
11 # finding the best fit model
12 bst_TSModel(df['NASDAQ.EBAY'])
13 print("\n The Optimised Paramters by using ACF and PACF for model \n", '-'*80)
```

```
In [ ]: 1 # Step4 : to fit the model and predict
2 df['EBAY_SHIFT'].dropna(inplace=True)
3 print(df['EBAY_SHIFT'].head())
4 model = sm.tsa.ARIMA(df['EBAY_SHIFT'], order=(1, 1, 0))
5 results = model.fit()
6 print( " The ARIMA Model Fitted and Forecasted Values plotted \n", '-'*80)
7 df['Forecast'] = results.fittedvalues
8 df[['EBAY_SHIFT', 'Forecast']].plot(figsize=(6,4))
9
10
11 print(df['Forecast'].head())
12
13
```

```
In [ ]: 1 predictions = pd.Series(results.fittedvalues,copy=True)
2 # print(predictions_ARIMA_diff.head())
3 # print('- '*80)
4 # taking cumulativeSum
5 predictions_ARIMA_cumsum = predictions.cumsum()
6 print(predictions_ARIMA_cumsum.head())
7
8
9 predictions_ARIMA= pd.Series(df['NASDAQ.EBAY'].values)
10
11 predictions_ARIMA = predictions_ARIMA.add(predictions_ARIMA_cumsum,fill_value=0)
12 # predictions_ARIMA_Log.head()
13 # print(predictions_ARIMA_Log.head())
14
15
16 # predictions_ARIMA = np.exp(predictions_ARIMA_Log)
17 print(predictions_ARIMA.head())
18 print(" Plotting the Original and Forecasted Stocks \n",'- '*80)
19
20 df['NASDAQ.EBAY_FORECAST'] = predictions_ARIMA
21 # df[['NASDAQ.EBAY']].plot(figsize=(16, 12))
22 # df[['NASDAQ.EBAY_FORECAST']].plot(figsize=(16, 12))
23 df[['NASDAQ.EBAY', 'NASDAQ.EBAY_FORECAST']].plot(figsize=(6,4))
24
25
```