In [ ]:

```
#*****************************************************************************
#
#                 Assignment : Project On TimeSeries
#
#*****************************************************************************
# Problem Statement : For Following Stocks Forecast to be done
# 1. NASDAQ.AAPL
# 2. NASDAQ.ADP
# 3. NASDAQ.CBOE
# 4. NASDAQ.CSCO
# 5. NASDAQ.EBAY

# Import the Packages needed
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from pandas.tools.plotting import autocorrelation_plot
from statsmodels.graphics.tsaplots import plot_pacf
from statsmodels.tsa.arima_model import ARIMA, ARMAResults
import datetime
import sys
import seaborn as sns
import statsmodels
import statsmodels.stats.diagnostic as diag
from statsmodels.tsa.stattools import adfuller
from scipy.stats.mstats import normaltest
import statsmodels.api as sm

from matplotlib.pyplot import acorr
import warnings


# to remove the warnings coming in the output
if __name__ == '__main__':
    warnings.filterwarnings(action='ignore', category=Warning)


plt.style.use('fivethirtyeight')
%matplotlib inline

# loading the DataSet
```

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

In [ ]:

```
# Analysing the Data , to see if there are null , missing values ,as well to see fields having big variations
# or constant variations
print(df.describe())
print('-'*80)

print(df.info())
print('-'*80)

print(df.isnull().sum())
print('-'*80)

print(df.shape)
```

In [ ]:

```
# to check the stationarity , we need to check the rolling mean and Dicker fuller test
from statsmodels.tsa.stattools import adfuller

# get_mean_var function gives the mean and the variance for the values
def get_mean_var(series, no_of_samples):
    split_size = int(len(series) / no_of_samples)
    start = 0
    for i in range(no_of_samples):
        sample_series = series[i*split_size:(i+1)*split_size]
        #print(sample_series,"\n")
        print('Mean= %.2f, Variance= %.2f' % (sample_series.mean(), sample_series.var()))


def plot_rolling_statistics(timeseries):
    #Determing rolling statistics
    rolmean = timeseries.rolling(window=500).mean()
    rolstd = timeseries.rolling(window=500).std()

    #Plot rolling statistics:
    plt.plot(timeseries, color='blue',label='Original')
    plt.plot(rolmean, color='red', label='Rolling Mean')
    plt.plot(rolstd, color='black', label = 'Rolling Std')

    plt.legend(loc='best')
    plt.title('Rolling Mean & Standard Deviation')
    plt.figure(figsize=(20,20))

    plt.show()



def dickey_fuller_test(timeseries):
    #Perform Dickey-Fuller test:
    print('Results of Dickey-Fuller Test:')
    dftest = adfuller(timeseries, autolag='AIC') # Akaike information criterion

    # https://coolstatsblog.com/2013/08/14/using-aic-to-test-arima-models-2/
    #https://www.statsmodels.org/dev/generated/statsmodels.tsa.stattools.adfuller.html
    #statsmodels.tsa.stattools.adfuller(x, maxlag=None, regression='c', autolag='AIC',
    # store=False, regresults=False)

```

```
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Observations Used'])

    for key,value in dftest[4].items():
        dfoutput['Critical Value (%s)'%key] = value
    print(dfoutput)


def test_stationarity(timeseries):
    no_of_samples = 10
    print(" Printing the Mean and Variance for the Stock \n",'-'*80)
    get_mean_var(timeseries, no_of_samples)
    print(" \nPrinting results Rolling statistics and Dicker Fuller test Values\n",'-'*80)
    plot_rolling_statistics(timeseries)
    dickey_fuller_test(timeseries)
```

In [ ]:

```
#*****************************************************************
# 1.            Forecasting for  Stock  NASDAQ.AAPL
#*****************************************************************

# Step 1. Visualizing the Stock Values
df['NASDAQ.AAPL'].plot(figsize=(10, 6))

# we observe from the plot , there is upward trend , so we will make it stationary
print(" We observe from the plot , Stock -NASDAQ.AAPL has upward trend\n " ,"-"*80)
```

In [ ]:

```
# get_mean_var(df['NASDAQ.AAPL'].values, 10)
# from the values , we observe that mean , variance are not constant
test_stationarity(df['NASDAQ.AAPL'])

# we Observe from test stationarity results that
# 1. Pvalue is greater than zero
# 2. test statics need to be less than critical ,
# as all conditons not met , we need to stationarize the series
print('-'*80," \nWE Observe from tests that \n",'-'*80)
print('''
1. Pvalue is greater than zero
2. test statics need to be less than critical ,
as all conditons not met , we need to stationarize the series

''')
```

In [ ]:

```
# Step2 : Stationarize the Stock by ,trying the shift technique
import statsmodels.api as sm
df['AAPL_SHIFT'] = df['NASDAQ.AAPL']-df['NASDAQ.AAPL'].shift()
print( " Stationarizing the Stock Series \n",'-'*80)

print(" Printing the First Difference Values\n",'-'*80)
print(df['AAPL_SHIFT'].head())

df['AAPL_SHIFT'].dropna(inplace=True)

# from the values , we observe that mean , variance are not constant
test_stationarity(df['AAPL_SHIFT'])
# we observe that Pvalue is zero , and test statistics is < Critical Value and
sm.stats.durbin_watson(df['AAPL_SHIFT'])
# the durbin_watson value shows no correlation as value =2
# series is stationary

print(''' \nFrom Tests We observe \n''','-'*80,'''
1. Pvalue is equal to zero
2. test statics  less than critical ,
3. Durbin watson test =2 , shows no coorelation
as all conditons  met , we can consider series stationarized ''')

```

In [ ]:

```
# function to get the best model

#ararray = df['NASDAQ.AAPL']
def bst_TSModel(ararray):
    p=0
    q=0
    d=1

    pdq=[]
    aic=[]

    for p in range(2):
        for q in range(2):
            try:
                model = sm.tsa.ARIMA(ararray, (p,d,q)).fit()
                x = model.aic
                x1 = (p,d,q)

                print (x1, x)
                aic.append(x)
                pdq.append(x1)
            except:
                print('error')
                pass

    #print(pdq,aic)
    keys = pdq
    values = aic
    d = dict(zip(keys, values))

    # Best Model

    minaic=min(d, key=d.get)
    print ("\nBest Model is :", minaic)

```

In [ ]:
```
# Step3 : To optimize the parameters we use ACF and PACF
%matplotlib inline
fig = plt.figure(figsize=(12,8))

ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(df['NASDAQ.AAPL'].values.squeeze(), lags=30, ax=ax1)

ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(df['NASDAQ.AAPL'].values.squeeze(), lags=70, ax=ax2)

bst_TSModel(df['AAPL_SHIFT'])

print("\n The Optimised Paramters by using ACF and PACF for model \n",'-'*80)
print(" From the PACF we see that p= 1 (lags > 1 are all zeros)")
```

In [ ]:
```
# Step4 :  to fit the model and predict
df['AAPL_SHIFT'].dropna(inplace=True)
model = sm.tsa.ARIMA(df['AAPL_SHIFT'], order=(1, 1, 0))
results = model.fit(disp=-1)

print( " The ARIMA Model Fitted and Forecasted Values plotted \n",'-'*80)

df['Forecast'] = results.fittedvalues

df[['AAPL_SHIFT', 'Forecast']].plot(figsize=(10, 6))
```

In [ ]:
```
#print(df['Forecast'].head())
predictions = pd.Series(results.fittedvalues,copy=True)

# taking cumulativeSum
predictions_ARIMA_cumsum = predictions.cumsum()

predictions_ARIMA_Forecast= pd.Series(df['NASDAQ.AAPL'].values)
predictions_ARIMA_Forecast = predictions_ARIMA_Forecast.add(predictions_ARIMA_cumsum,fill_value=0)

print("\n Plotting the Original and Forecasted Stocks \n",'-'*80)

predictions_ARIMA_Forecast.head()

df['NASDAQ.AAPL_FORECAST'] = predictions_ARIMA_Forecast
df[['NASDAQ.AAPL', 'NASDAQ.AAPL_FORECAST']].plot(figsize=(10, 6))
```

In [ ]:
```
# predictions = pd.Series(results.fittedvalues,copy=True)
# print(predictions.head())
from sklearn.metrics import mean_squared_error
def mean_forecast_err(y, yhat):
    return y.sub(yhat).mean()

def mean_absolute_err(y, yhat):
    return np.mean((np.abs(y.sub(yhat).mean()) / yhat)) # or percent error = * 100



print("Mean ForeCast Error -  MFE = ",mean_forecast_err(df['NASDAQ.AAPL'], df['NASDAQ.AAPL_FORECAST']))
print("Mean Absolute Error - MAE = ",mean_absolute_err(df['NASDAQ.AAPL'], df['NASDAQ.AAPL_FORECAST']))
print("Mean Square Error - MSE = ",mean_squared_error(df['NASDAQ.AAPL'], df['NASDAQ.AAPL_FORECAST']))

```

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

In [ ]:
```
#get_mean_var(df['NASDAQ.ADP'].values, 10)
# from the values , we observe that mean , variance are not constant
test_stationarity(df['NASDAQ.ADP'])
# Durbin watson test for stationarity
print('-'*80,'\n','DurbinWatson test : ',sm.stats.durbin_watson(df['NASDAQ.ADP']))
# We observe durbin watson Value > 2 , so there is negative correlation


# we Observe from test stationarity results that
# 1. Pvalue is greater than zero
# 2. test statics > than critical ,
# as all conditons not met , we need to stationarize the series
print('-'*80," \nWE Observe from tests that \n",'-'*80)
print('''
1. Pvalue is greater than zero
2. test statics need to be less than critical ,
as all conditons not met , we need to stationarize the series

''')
```

In [ ]:

```
# Step2 : Stationarize the Stock by ,trying the shift technique
import statsmodels.api as sm
df['ADP_SHIFT'] = df['NASDAQ.ADP']-df['NASDAQ.ADP'].shift()

print(" Printing the First Difference Values\n",'-'*80)
print( " Stationarizing the Stock Series \n",'-'*80)

print(df['ADP_SHIFT'].head())

df['ADP_SHIFT'].dropna(inplace=True)

#get_mean_var(df['ADP_SHIFT'].values, 10)
# from the values , we observe that mean , variance are not constant
test_stationarity(df['ADP_SHIFT'])
# we observe that Pvalue is zero , and test statistics is < Critical Value and
print('-'*80,'\nDurbin Watson Test : ',sm.stats.durbin_watson(df['ADP_SHIFT']))
# the durbin_watson value shows no correlation as value =2
# series is stationary

print(''' \nFrom Tests We observe \n''','-'*80,'''
1. Pvalue is equal to zero
2. test statics  less than critical ,
3. Durbin Watson test =2 , no correlation
as all conditons  met , we can consider series stationarized ''')

```

In [ ]:
```
# Step3 : To optimize the parameters we use ACF and PACF
%matplotlib inline
fig = plt.figure(figsize=(12,8))

ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(df['NASDAQ.ADP'].values.squeeze(), lags=40, ax=ax1)

ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(df['NASDAQ.ADP'].values.squeeze(), lags=40, ax=ax2)



# finding the best fit model
bst_TSModel(df['ADP_SHIFT'])

print("\n The Optimised Paramters by using ACF and PACF for model \n",'-'*80)
```

In [ ]:
```
# Step4 :  to fit the model and predict
df['ADP_SHIFT'].dropna(inplace=True)
model = sm.tsa.ARIMA(df['ADP_SHIFT'], order=(2, 1, 3))
results = model.fit(disp=-1)

print( " The ARIMA Model Fitted and Forecasted Values plotted \n",'-'*80)

df['Forecast'] = results.fittedvalues
df[['ADP_SHIFT', 'Forecast']].plot(figsize=(10, 8))
```

In [ ]:
```
print(df['Forecast'].head())
predictions = pd.Series(results.fittedvalues,copy=True)
# print(predictions_ARIMA_diff.head())
# print('-'*80)
# taking cumulativeSum
predictions_ARIMA_cumsum = predictions.cumsum()
print(predictions_ARIMA_cumsum.head())
predictions_ARIMA_Forecast= pd.Series(df['NASDAQ.ADP'].values)
predictions_ARIMA_Forecast = predictions_ARIMA_Forecast.add(predictions_ARIMA_cumsum,fill_value=0)
print(predictions_ARIMA_Forecast.head())

print(" Plotting the Original and Forecasted Stocks \n",'-'*80)
df['NASDAQ.ADP_FORECAST'] = predictions_ARIMA_Forecast

df[['NASDAQ.ADP', 'NASDAQ.ADP_FORECAST']].plot(figsize=(10, 8))
```

In [ ]:
```
#*****************************************************************
#             Forecasting for  Stock  NASDAQ.CBOE
#*****************************************************************
# Step 1. Visualizing the Stock Values

df['NASDAQ.CBOE'].plot(figsize=(8, 6))
# we observe from the plot , there is upward trend , so we will make it stationary
print(" We observe from the plot , Stock -NASDAQ.CBOE has upward trend\n " ,"-"*80)
```

In [ ]:
```
# from the values , we observe that mean , variance are not constant
test_stationarity(df['NASDAQ.CBOE'])

print('-'*80," \nWE Observe from tests that \n",'-'*80)
print('''
1. Pvalue is greater than zero
2. test statics need to be less than critical ,
as all conditons not met , we need to stationarize the series

''')
```

In [ ]:
```
# Step2 : Stationarize the Stock by ,trying the shift technique log

print( " Stationarizing the Stock Series \n",'-'*80)
import statsmodels.api as sm
df['CBOE_SHIFT'] = df['NASDAQ.CBOE']-df['NASDAQ.CBOE'].shift()

print(" Printing the First Difference Values\n",'-'*80)
print(df['CBOE_SHIFT'].head())

df['CBOE_SHIFT'].dropna(inplace=True)

# from the values , we observe that mean , variance are not constant
test_stationarity(df['CBOE_SHIFT'])
# we observe that Pvalue is zero , and test statistics is < Critical Value and
print('-'*80,'\nDurbin Watson test :',sm.stats.durbin_watson(df['CBOE_SHIFT']))
# the durbin_watson value shows no correlation as value =2
# series is stationary
print(''' \nFrom Tests We observe \n''','-'*80,'''
1. Pvalue is equal to zero
2. test statics  less than critical ,
3. Durbin watson test =2 , no correlation
as all conditons  met , we can consider series stationarized ''')

```

In [ ]:
```
# Step3 : To optimize the parameters we use ACF and PACF
%matplotlib inline
fig = plt.figure(figsize=(10,6))

ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(df['NASDAQ.CBOE'].values.squeeze(), lags=40, ax=ax1)

ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(df['NASDAQ.CBOE'].values.squeeze(), lags=40, ax=ax2)

# finding the best fit model
bst_TSModel(df['NASDAQ.CBOE'])
print("\n The Optimised Paramters by using ACF and PACF for model \n",'-'*80)

```

In [ ]:
```
# Step4 :  to fit the model and predict
df['CBOE_SHIFT'].dropna(inplace=True)
#print(df['CBOE_SHIFT'].head())
model = sm.tsa.ARIMA(df['CBOE_SHIFT'], order=(1, 1, 0))
results = model.fit()

print( " The ARIMA Model Fitted and Forecasted Values plotted \n",'-'*80)
df['Forecast'] = results.fittedvalues
df[['CBOE_SHIFT', 'Forecast']].plot(figsize=(10, 6))



```

In [ ]:
```

predictions = pd.Series(results.fittedvalues,copy=True)
predictions_ARIMA_cumsum = predictions.cumsum()

predictions_ARIMA_Forecast= pd.Series(df['NASDAQ.CBOE'].values)

predictions_ARIMA = predictions_ARIMA_Forecast.add(predictions_ARIMA_cumsum,fill_value=0)

print(" Plotting the Original and Forecasted Stocks \n",'-'*80)
df['NASDAQ.CBOE_FORECAST'] = predictions_ARIMA
df[['NASDAQ.CBOE', 'NASDAQ.CBOE_FORECAST']][:20000].plot(figsize=(10, 6))


```

In [ ]:
```
#*****************************************************************
# 4.            Forecasting for  Stock  NASDAQ.CSCO
#*****************************************************************
# Step 1. Visualizing the Stock Values

df['NASDAQ.CSCO'].plot(figsize=(6, 4))

# we observe from the plot , there is downward trend ,and stock not stationary
# so we will need to make it stationary
print(" We observe from the plot , Stock -NASDAQ.CSCO has downward trend\n " ,"-"*80)


```

In [ ]:

```
# from the values , we observe that mean , variance are not constant
test_stationarity(df['NASDAQ.CSCO'])

# we Observe from test stationarity results that
# 1. Pvalue is greater than zero
# 2. test statics need to be less than critical ,
# as all conditons not met , we need to stationarize the series
print('-'*80," \nWE Observe from tests that \n",'-'*80)
print('''
1. Pvalue is greater than zero
2. test statics need to be less than critical ,
as all conditons not met , we need to stationarize the series

''')

```

In [ ]:

```
# Step2 : Stationarize the Stock by ,trying the shift technique
import statsmodels.api as sm

# df['CBOE_log'] = df['NASDAQ.CBOE'].apply(lambda x: np.log(x))
# df['CBOE_log'].plot(figsize=(16, 12))

print( " Stationarizing the Stock Series \n",'-'*80)
df['CSCO_SHIFT'] = df['NASDAQ.CSCO']-df['NASDAQ.CSCO'].shift()

print(" Printing the First Difference Values\n",'-'*80)
print(df['CSCO_SHIFT'].head())

df['CSCO_SHIFT'].dropna(inplace=True)

#get_mean_var(df['CSCO_SHIFT'].values, 10)
# from the values , we observe that mean , variance are not constant
test_stationarity(df['CSCO_SHIFT'])
# we observe that Pvalue is zero , and test statistics is < Critical Value and
print('-'*80,'\n Durbin Watson Test : ',sm.stats.durbin_watson(df['CSCO_SHIFT']))
# the durbin_watson value shows no correlation as value =2
# series is stationary

print(''' \nFrom Tests We observe \n''','-'*80,'''
1. Pvalue is equal to zero
2. test statics  less than critical ,
3. The Durbin Watson almost = 2 , no co-relation
as all conditons  met , we can consider series stationarized ''')
```

In [ ]:
```
# Step3 : To optimize the parameters we use ACF and PACF
%matplotlib inline
fig = plt.figure(figsize=(6,4))

ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(df['NASDAQ.CSCO'].values.squeeze(), lags=40, ax=ax1)

ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(df['NASDAQ.CSCO'].values.squeeze(), lags=40, ax=ax2)

# finding the best fit model
bst_TSModel(df['NASDAQ.CSCO'])
print("\n The Optimised Paramters by using ACF and PACF for model \n",'-'*80)
```

In [ ]:
```
# Step4 :  to fit the model and predict
df['CSCO_SHIFT'].dropna(inplace=True)
print(df['CSCO_SHIFT'].head())
model = sm.tsa.ARIMA(df['CSCO_SHIFT'], order=(1, 1, 0))
results = model.fit(disp=-1)

print( " The ARIMA Model Fitted and Forecasted Values plotted \n",'-'*80)
df['Forecast'] = results.fittedvalues
df[['CSCO_SHIFT', 'Forecast']].plot(figsize=(6, 4))


```

In [ ]:
```
print(df['Forecast'].head())

predictions = pd.Series(results.fittedvalues,copy=True)
# print(predictions_ARIMA_diff.head())
# print('-'*80)
# taking cumulativeSum
predictions_ARIMA_cumsum = predictions.cumsum()
print(predictions_ARIMA_cumsum.head())


predictions_ARIMA_Forecast= pd.Series(df['NASDAQ.CSCO'].values)

predictions_ARIMA = predictions_ARIMA_Forecast.add(predictions_ARIMA_cumsum,fill_value=0)
# predictions_ARIMA_log.head()
# print(predictions_ARIMA_log.head())


# predictions_ARIMA = np.exp(predictions_ARIMA_log)
print(predictions_ARIMA.head())

print(" Plotting the Original and Forecasted Stocks \n",'-'*80)
df['NASDAQ.CSCO_FORECAST'] = predictions_ARIMA
df[['NASDAQ.CSCO', 'NASDAQ.CSCO_FORECAST']][:20000].plot(figsize=(6,4))

```

In [ ]:
```
#*********************************************************************
# 5.            Forecasting for  Stock  NASDAQ.EBAY
#*********************************************************************

# Step 1. Visualizing the Stock Values
df['NASDAQ.EBAY'].plot(figsize=(6,4))

# we observe from the plot , there is upward trend , so we will make it stationary
print(" We observe from the plot , Stock -NASDAQ.EBAY has upward trend\n " ,"-"*80)
```

In [ ]:

```
# get_mean_var(df['NASDAQ.EBAY'].values, 10)
# from the values , we observe that mean , variance are not constant
test_stationarity(df['NASDAQ.EBAY'])

print('-'*80," \nWE Observe from tests that \n",'-'*80)
print('''
1. Pvalue is greater than zero
2. test statics need to be less than critical ,
as all conditons not met , we need to stationarize the series

''')
```

In [ ]:

```
# Step2 : Stationarize the Stock by ,trying the shift technique
import statsmodels.api as sm

# df['CBOE_log'] = df['NASDAQ.CBOE'].apply(lambda x: np.log(x))
# df['CBOE_log'].plot(figsize=(16, 12))

print( " Stationarizing the Stock Series \n",'-'*80)
df['EBAY_SHIFT'] = df['NASDAQ.EBAY']-df['NASDAQ.EBAY'].shift()

print(" Printing the First Difference Values\n",'-'*80)
print(df['EBAY_SHIFT'].head())

df['EBAY_SHIFT'].dropna(inplace=True)

#get_mean_var(df['EBAY_SHIFT'].values, 10)
# from the values , we observe that mean , variance are not constant
test_stationarity(df['EBAY_SHIFT'])
# we observe that Pvalue is zero , and test statistics is < Critical Value and
print('-'*80,'\nDurbin Watson Test : ',sm.stats.durbin_watson(df['EBAY_SHIFT']))
# the durbin_watson value shows no correlation as value =2
# series is stationary

print(''' \nFrom Tests We observe \n''','-'*80,'''
1. Pvalue is equal to zero
2. test statics  less than critical ,
as all conditons  met , we can consider series stationarized ''')


```

In [ ]:
```
# Step3 : To optimize the parameters we use ACF and PACF
%matplotlib inline
fig = plt.figure(figsize=(6,4))

ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(df['NASDAQ.EBAY'].values.squeeze(), lags=40, ax=ax1)

ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(df['NASDAQ.EBAY'].values.squeeze(), lags=40, ax=ax2)

# finding the best fit model
bst_TSModel(df['NASDAQ.EBAY'])
print("\n The Optimised Paramters by using ACF and PACF for model \n",'-'*80)
```

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


print(df['Forecast'].head())


```

In [ ]:

```
predictions = pd.Series(results.fittedvalues,copy=True)
# print(predictions_ARIMA_diff.head())
# print('-'*80)
# taking cumulativeSum
predictions_ARIMA_cumsum = predictions.cumsum()
print(predictions_ARIMA_cumsum.head())


predictions_ARIMA= pd.Series(df['NASDAQ.EBAY'].values)

predictions_ARIMA = predictions_ARIMA.add(predictions_ARIMA_cumsum,fill_value=0)
# predictions_ARIMA_log.head()
# print(predictions_ARIMA_log.head())


# predictions_ARIMA = np.exp(predictions_ARIMA_log)
print(predictions_ARIMA.head())
print(" Plotting the Original and Forecasted Stocks \n",'-'*80)

df['NASDAQ.EBAY_FORECAST'] = predictions_ARIMA
# df[['NASDAQ.EBAY']].plot(figsize=(16, 12))
# df[['NASDAQ.EBAY_FORECAST']].plot(figsize=(16, 12))
df[['NASDAQ.EBAY', 'NASDAQ.EBAY_FORECAST']].plot(figsize=(6,4))


```