Assignment

by

B20MT005 and M21AIE208

<u>Assignment Logistics</u>

a. The assignment is done using **python 3.8**
b. Dataset used is Euro/INR exchange rate and gathered from RBI website

https://dbie.rbi.org.in/BOE/OpenDocument/1608101729/OpenDocument/opendoc/openDocument.faces?logonSuccessful=true&shareId=0

c. We will be analysis daily timeframe data from period **1st Jan 2022** till **1st Jan 2023** that has 176 datapoints

Part 1

ARMA/ ARIMA and its variation

Step 1: Loading the data

```
df=pd.read_csv("euro.csv") #loading the file
```

```
df['Date'] = pd.to_datetime(df['Date'], format='%d/%m/%Y') #converting date column "str" to "datetime" object
df.set_index('Date', inplace=True)
```

```
df.head(5) #printing top-5 data on the table
```

	EURO
Date	
2016-04-04	75.3713
2016-04-05	75.5240
2016-04-06	75.6290
2016-04-07	75.8952
2016-04-11	75.7538

Step 2: Printing Basic Statistics

```
1 df.describe() # prinitng basic Data statistics
```

	EURO
count	788.000000
mean	77.765828
std	5.069230
min	68.250300
25%	74.216875
50%	76.521850
75%	80.691400
max	90.255500

Step 3: Plotting time series

```
1 mmr1= df.loc['2022-1-1':'2023-1-1'] #filtering data for 2022-1-1 till 2023-1-1
```

```
1 mmr1.plot()
```

```
<Axes: xlabel='Date'>
```

Step 4: Test for non-stationarity using ADF and PP

1. Augmented Dickey-Fuller test

```
result = adfuller(mmr1) #augmented Dickey-Fuller test
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])
print('Critical Values:')
for key, value in result[4].items():
 print('\t%s: %.3f' % (key, value))
```

```
ADF Statistic: -0.070607
p-value: 0.952289
Critical Values:
	1%: -3.468
	5%: -2.878
	10%: -2.576
```

2. Phillips Perron test

```
PhillipsPerron(mmr1)
```

```
Phillips-Perron Test
       (Z-tau)
Test Statistic 0.083
P-value        0.965
Lags           14


Trend: Constant
Critical Values: -3.47 (1%), -2.88 (5%), -2.58 (10%)
Null Hypothesis: The process contains a unit root.
Alternative Hypothesis: The process is weakly stationary.
```

ADF and PP statistics and p-value rejects the null hypothesis for stationarity, hence this series is non-stationary

Step 5: Test for non-stationarity at First Difference of the series using Autocorrelation and Partial Auto Correlation

Plotting Auto-correlation function

```
1 plot_acf(mmr1,lags=50) #plotting autocorrelation at First Difference
2 pyplot.show()
```

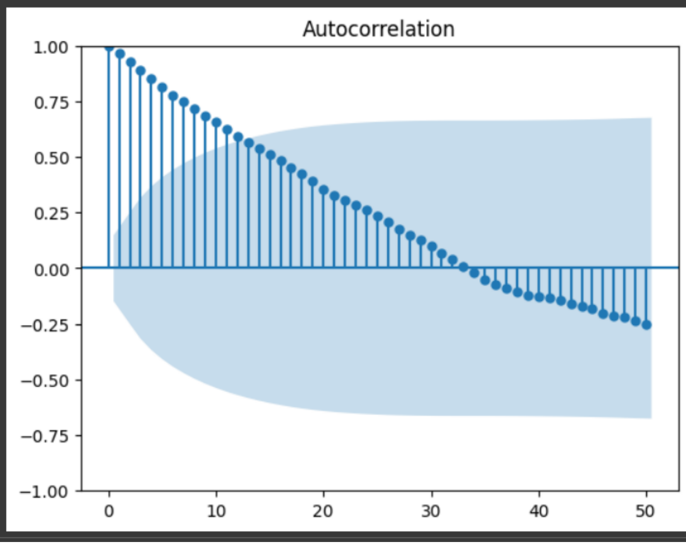


Figure 1 Autocorrelation plot of actual series with lag 50

```
1 plot_acf(dmmr1,lags=50) #plotting autocorelation of first difference
2 pyplot.show()
```

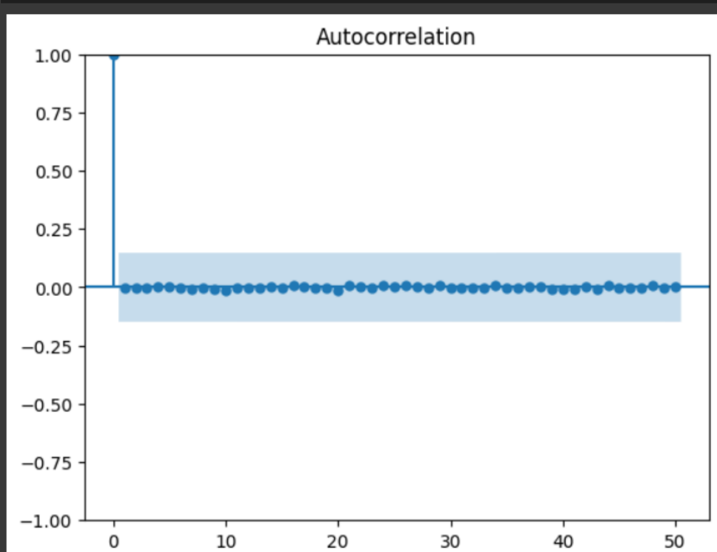


Figure 2 Autocorrelation of First Difference

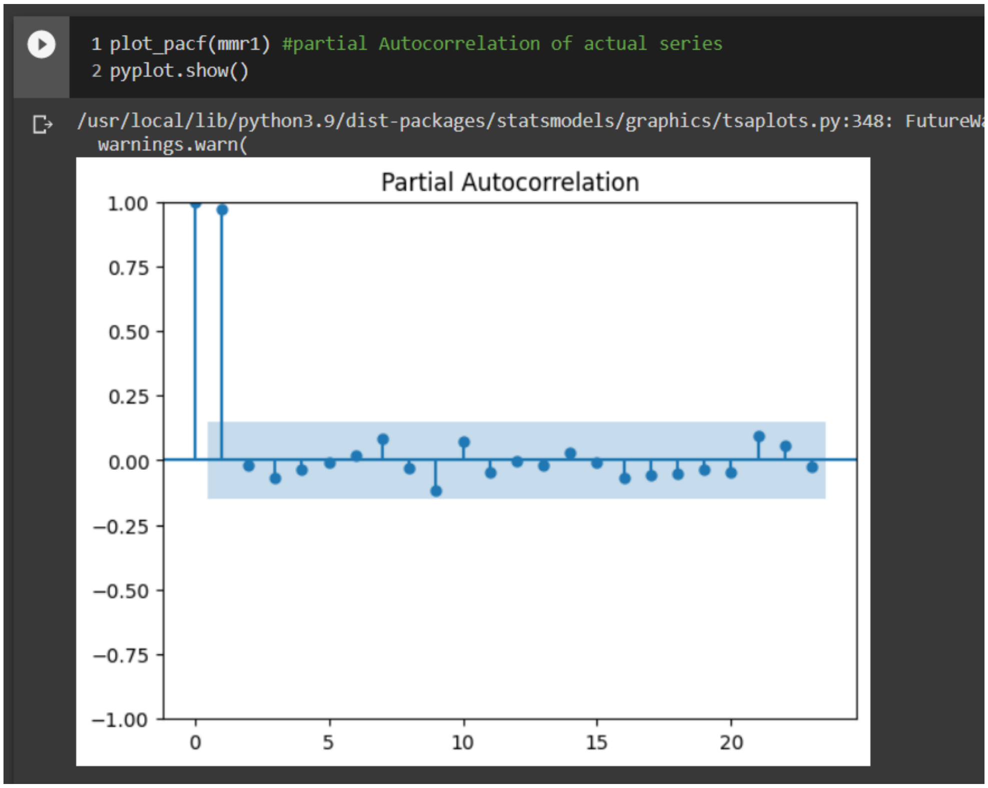


Figure 3 Partial Autocorrelation of the Actual Series

Since ACF plot shows that Auto-correlation is dropping immediately after first lag and we can use ARIMA model

Step 5: Building ARIMA model

1. **ARIMA model [Order (5,0,1)]**

```
[ ]  from statsmodels.tsa.arima.model import ARIMA #loading ARIMA model in python
```

```
model = ARIMA(mmr1, order=(5,1,0)) #initialisation of ARIMA model
model_fit = model.fit() # model fitting
# summary of fit model
print(model_fit.summary())
# line plot of residuals
residuals = pd.DataFrame(model_fit.resid)
residuals.plot()
pyplot.show()
```

```
/usr/local/lib/python3.9/dist-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarni
  self._init_dates(dates, freq)
/usr/local/lib/python3.9/dist-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarni
  self._init_dates(dates, freq)
/usr/local/lib/python3.9/dist-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarni
  self._init_dates(dates, freq)
                               SARIMAX Results
==============================================================================
Dep. Variable:                   EURO   No. Observations:                  176
Model:                 ARIMA(5, 1, 0)   Log Likelihood                -113.331
Date:                Sat, 01 Apr 2023   AIC                            238.661
Time:                        09:19:19   BIC                            257.650
Sample:                             0   HQIC                           246.364
                                - 176
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.0237      0.069      0.341      0.733      -0.112       0.160
ar.L2          0.0247      0.088      0.280      0.779      -0.148       0.198
ar.L3          0.0317      0.088      0.359      0.719      -0.141       0.205
ar.L4          0.0187      0.072      0.260      0.795      -0.122       0.159
ar.L5         -0.0642      0.078     -0.822      0.411      -0.217       0.089
sigma2         0.2138      0.024      8.909      0.000       0.167       0.261
===================================================================================
Ljung-Box (L1) (Q):                   0.05   Jarque-Bera (JB):                 0.35
Prob(Q):                              0.83   Prob(JB):                         0.84
Heteroskedasticity (H):               1.09   Skew:                             0.08
Prob(H) (two-sided):                  0.76   Kurtosis:                         2.86
===================================================================================
```

```
[ ]
# density plot of residuals
residuals.plot(kind='kde')
pyplot.show()
```

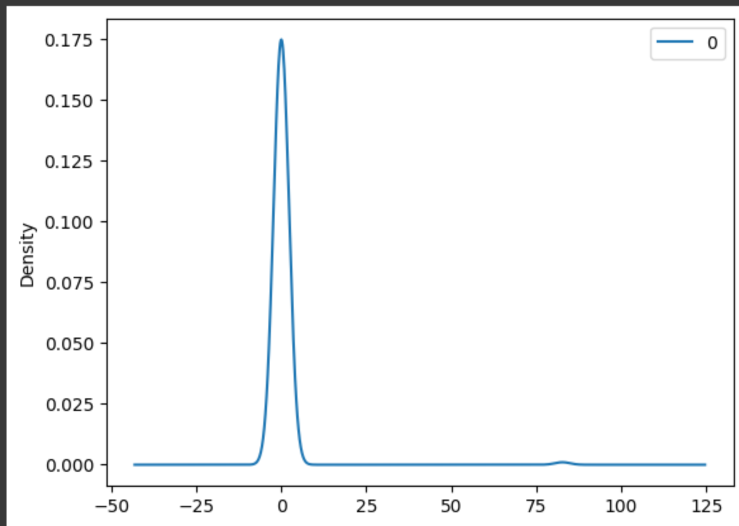
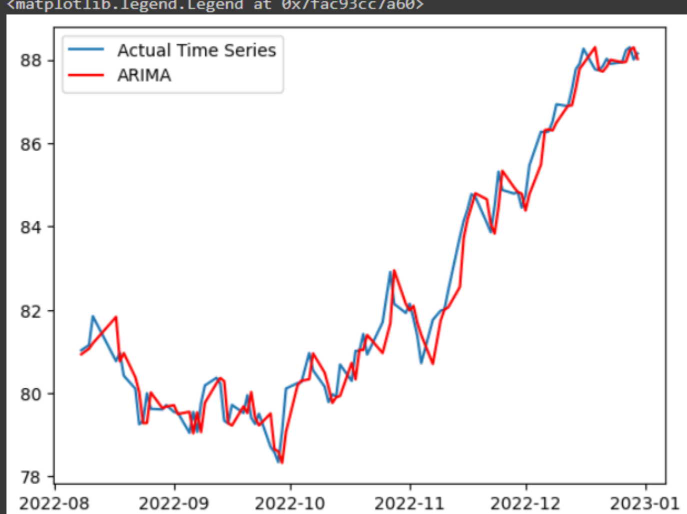


Figure 4 Density plots for Residuals

Step 6: Checking the plot of the forecast and the actual time series

```
[ ]
import matplotlib.pyplot as plt
plt.plot(mmr1[80:],label="Actual Time Series")
plt.plot(model_fit.fittedvalues[80:], color='red',label="ARIMA")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fac93cc7a60>
```



Step 7: Building 2 different ARIMA model

a. **ARIMA [Order(5,0,1)]**
b. **ARIMA [Order(10,0,1)]**

```
predicted = []
predicted2 = []
for i in range(0,10): #Rolling Forecast ARIMA model
  mmr1= df.loc['2022-1-1':f'2023-1-{1+i}']
  # # fit model
  model = ARIMA(mmr1, order=(5,1,0)) #-> ARIMA model 1
  model_fit = model.fit()
  predicted.append(model_fit.forecast(steps=1).to_list()[0]) #First period ahead forecast for Model1

  #prediction by model 2
  model2 = ARIMA(mmr1, order=(10,1,0)) #-> ARIMA model 2
  model_fit2 = model2.fit()
  predicted2.append(model_fit2.forecast(steps=1).to_list()[0]) #First period ahead forecast for Model1


```

We are predicting the first period ahead forecast for 10 points i.e **2nd Jan 2023** till **11th Jan 2023**

Predicted Results from 2 forecasters

```
test['Model 1 predicted']=predicted
test['Model 2 predicted']=predicted2
```

```
test
```

Date	EURO	Model 1 predicted	Model 2 predicted
2023-01-02	88.3753	88.150863	88.195150
2023-01-03	88.2508	88.357854	88.371293
2023-01-04	87.6546	88.247153	88.160069
2023-01-05	87.7992	87.666760	87.674117
2023-01-06	86.9235	87.783660	87.857475
2023-01-09	87.9528	86.871424	86.802108
2023-01-10	88.2829	86.871424	86.802108
2023-01-11	87.8551	86.871424	86.802108
2023-01-12	87.9566	87.916897	87.783534
2023-01-13	88.2628	88.337581	88.497731

Step 8: Evaluating Model 1 and Model 2 with MSPE and Debol Mariano Test

```
[ ]  from sklearn.metrics import mean_absolute_percentage_error
     mean_absolute_percentage_error(test['EURO'],test['predicted']) #MSPE for Model 1

     0.006269505931084722

[ ]  mean_absolute_percentage_error(test['EURO'],test['predicted2']) #MSPE for Model 2

     0.006781108218092335
```

Figure 5 MSPE calculation for Model1 and Model2

Debol Mariano Test

```
[ ]  from dieboldmariano import dm_test

[ ]  dm_test(test["EURO"], test['predicted'], test['predicted2'], one_sided=True)

     (-2.0061652704341872, 0.03789774276997642)
```

Figure 6 Forecaster comparison using Debold Mariano Test

Model	Order	MSPE
ARIMA model 1	[5,0,1]	0.0062
ARIMA model 2	[10,0,1]	0.0067

Since we can reject the null hypothesis of Debold Mariano test at 3% which suggest both forecaster are similar and MSPE show **model 1 is performing better**.

Step 9: Exchange Forecast for next 10 days using ARIMA model 1

```python
import matplotlib.pyplot as plt
plt.plot(test['EURO'],label="Actual Time Series")
plt.plot(test['Model 1 predicted'], color='red',label="ARIMA model 1")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fbd05b8deb0>
```

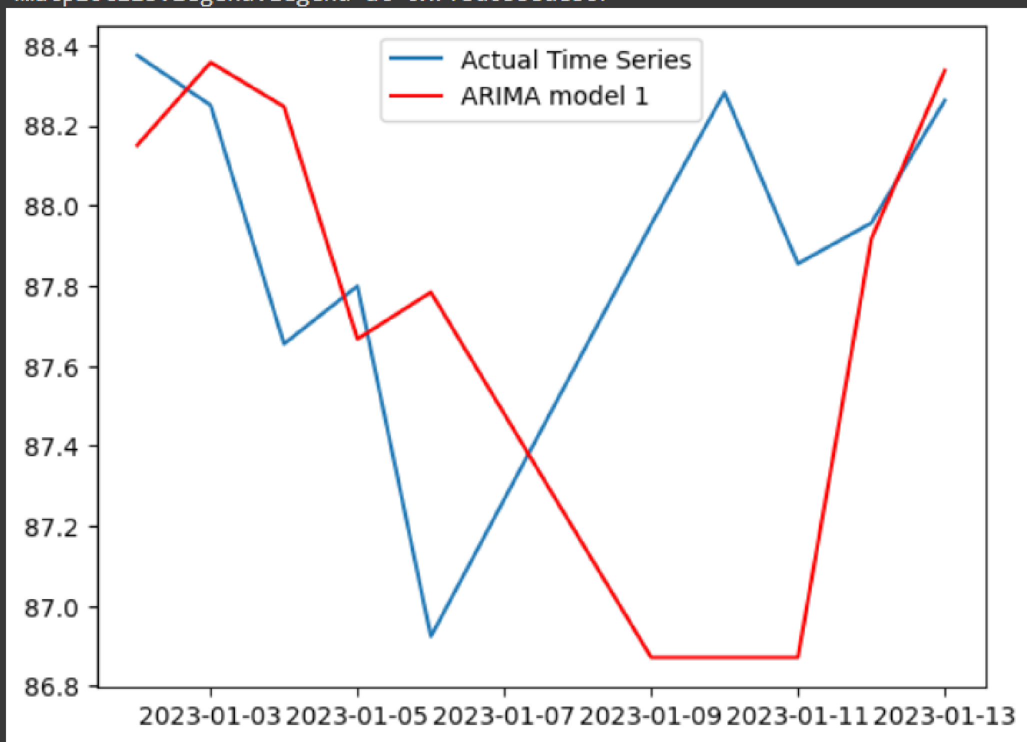


Figure 7 Prediction for next 10 days Exchange rate [EURO]

Part 2

ARCH/ GARCH and its variation

Stationarity of the data is already checked in Step1 till Step4

Step 5: Building 2 different ARCH model

1. ARCH model with lag 5
2. ARCH model with lag 10

```
model = arch_model(dmmr1, mean='Zero', vol='ARCH', p=10)
model_fit = model.fit() # model fitting
# summary of fit model
print(model_fit.summary())
# line plot of residuals
residuals = pd.DataFrame(model_fit.resid)
residuals.plot()
pyplot.show()
```

```
Optimization terminated successfully    (Exit mode 0)
            Current function value: 141.09663326174086
            Iterations: 35
            Function evaluations: 401
            Gradient evaluations: 31
                        Zero Mean - ARCH Model Results
==============================================================================
Dep. Variable:                   EURO   R-squared:                       0.000
Mean Model:                 Zero Mean   Adj. R-squared:                  0.006
Vol Model:                       ARCH   Log-Likelihood:               -141.097
Distribution:                  Normal   AIC:                           304.193
Method:            Maximum Likelihood   BIC:                           339.069
                                        No. Observations:                  176
Date:                Sun, Apr 02 2023   Df Residuals:                      176
Time:                        06:43:22   Df Model:                            0
                               Volatility Model
=============================================================================
                 coef    std err          t      P>|t|       95.0% Conf. Int.
-----------------------------------------------------------------------------
omega          0.1339  1.853e-02      7.231  4.811e-13    [9.764e-02,  0.170]
alpha[1]       0.5798      0.261      2.225  2.605e-02    [6.917e-02,  1.091]
alpha[2]       0.0000  1.736e-06      0.000      1.000 [-3.403e-06,3.403e-06]
alpha[3]       0.0000  1.252e-07      0.000      1.000 [-2.454e-07,2.454e-07]
alpha[4]       0.0000  1.136e-06      0.000      1.000 [-2.227e-06,2.227e-06]
alpha[5]       0.0000  3.570e-05      0.000      1.000 [-6.998e-05,6.998e-05]
alpha[6]       0.0000  6.116e-06      0.000      1.000 [-1.199e-05,1.199e-05]
alpha[7]       0.0000  1.684e-07      0.000      1.000 [-3.300e-07,3.300e-07]
alpha[8]       0.0000  6.868e-05      0.000      1.000 [-1.346e-04,1.346e-04]
alpha[9]       0.0000  4.159e-07      0.000      1.000 [-8.151e-07,8.151e-07]
alpha[10]      0.0000  1.450e-05      0.000      1.000 [-2.842e-05,2.842e-05]
=============================================================================
```

Figure 8 ARCH model fit with lag 5

```python
residual_1 = []
residual_2 = []
for i in range(0,10): #Rolling Forecast ARCH model
  mmr1= df.loc['2022-1-1':f'2023-1-{1+i}']
  dmmr1 = mmr1.diff()
  dmmr1.iloc[0]=mmr1.iloc[0]
  # # fit model
  model = arch_model(dmmr1, mean='Zero', vol='ARCH', p=5) #-> ARCH model 1
  model_fit = model.fit()
  forecasts1=model_fit.forecast(horizon=1, reindex=False)

  residual_1.append(forecasts1.residual_variance.iloc[-3:]['h.1'].to_list()[0]) #First period ahead forecast for Model1

  #prediction by model 2
  model2 = arch_model(dmmr1, mean='Zero', vol='ARCH', p=10) #-> ARCH model 2
  model_fit2 = model2.fit()
  forecasts2=model_fit2.forecast(horizon=1, reindex=False)

  residual_2.append(forecasts2.residual_variance.iloc[-3:]['h.1'].to_list()[0]) #First period ahead forecast for Model2
```

Predicted Results from both models ARCH model 1 and ARCH model 2

```python
test['Model 1 predicted']=test['EURO']+residual_1
test['Model 2 predicted']=test['EURO']+residual_2
```

```python
test
```

Date	EURO	Model 1 predicted	Model 2 predicted
2023-01-02	88.3753	88.553317	88.521539
2023-01-03	88.2508	88.437938	88.384635
2023-01-04	87.6546	87.831894	87.791520
2023-01-05	87.7992	88.128233	88.262203
2023-01-06	86.9235	87.098801	87.115259
2023-01-09	87.9528	88.572730	88.644720
2023-01-10	88.2829	88.902830	88.974820
2023-01-11	87.8551	88.475030	88.547020
2023-01-12	87.9566	88.741521	88.620423
2023-01-13	88.2628	88.509308	88.489496

Step 6: Evaluating Model 1 and Model 2 with MSPE and Debol Mariano Test

MSPE

```
1 from sklearn.metrics import mean_absolute_percentage_error
2 mean_absolute_percentage_error(test['EURO'],test['Model 1 predicted']) #MSPE for ARCH Model 1
```

```
0.004476541832959443
```

```
1 mean_absolute_percentage_error(test['EURO'],test['Model 2 predicted']) #MSPE for ARCH Model 2
```

```
0.004590856538003515
```

Debold Mariano Test

Debol Mariano Test

```
[146] 1 from dieboldmariano import dm_test
```

```
[148] 1 dm_test(test["EURO"], test['Model 1 predicted'], test['Model 2 predicted'], one_sided=True)
```

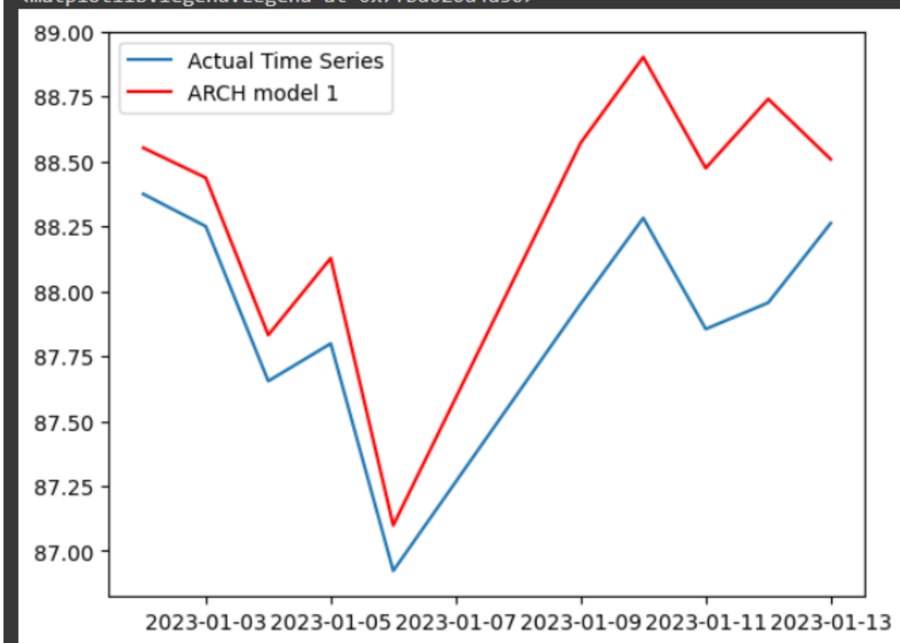
```
(-0.6271153276408591, 0.27308072449981224)
```

Model	**Lag**	**MSPE**
ARCH model 1	5	0.0044
ARCH model 2	10	0.0045

Step 7: Exchange Forecast for next 10 days using ARCH model 1

```
import matplotlib.pyplot as plt
plt.plot(test['EURO'],label="Actual Time Series")
plt.plot(test['Model 1 predicted'], color='red',label="ARCH model 1")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fbd020d4d30>
```



Comparing ARIMA and ARCH model

Model	**MSPE**
ARIMA model 1	0.0062
ARIMA model 2	0.0067
ARCH model 1	0.0044
ARCH model 2	0.0045

Conclusion: ARCH model have better forecast accuracy than ARIMA models