

✓ Finance Midterm Project (DATA 690)

The exchange rate is a significant factor that affects the choices made by market participants, including investors, importers, exporters, bankers, financial institutions, businesses, tourists, and policymakers in both developed and developing nations. For practitioners and researchers in the field of international finance, forecasting the exchange rate is crucial, especially when the exchange rate is floating.

Autoregressive Integrated Moving Averages The general process for ARIMA models is the following:

- Visualize the Time Series Data
- Make the time series data stationary
- Plot the Correlation and AutoCorrelation Charts
- Construct the ARIMA Model
- Use the model to make predictions

```
from google.colab import drive
drive.mount('/content/drive')
```

↗ Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True)

```
!pip install yfinance #install the required libraries
```

↗ Looking in indexes: <https://pypi.org/simple>, <https://us-python.pkg.dev/colab-wheels/public/simple/>

```
Requirement already satisfied: yfinance in /usr/local/lib/python3.7/dist-packages (0.1.84)
Requirement already satisfied: multitasking>=0.0.7 in /usr/local/lib/python3.7/dist-packages (from yfinance) (0.0.11)
Requirement already satisfied: numpy>=1.15 in /usr/local/lib/python3.7/dist-packages (from yfinance) (1.21.6)
Requirement already satisfied: pandas>=0.24.0 in /usr/local/lib/python3.7/dist-packages (from yfinance) (1.3.5)
Requirement already satisfied: lxml>=4.5.1 in /usr/local/lib/python3.7/dist-packages (from yfinance) (4.9.1)
Requirement already satisfied: appdirs>=1.4.4 in /usr/local/lib/python3.7/dist-packages (from yfinance) (1.4.4)
Requirement already satisfied: requests>=2.26 in /usr/local/lib/python3.7/dist-packages (from yfinance) (2.28.1)
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-packages (from pandas>=0.24.0->yfinance) (2.8.1)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (from pandas>=0.24.0->yfinance) (2022.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from python-dateutil>=2.7.3->pandas>=0.24.0->yfinance) (1.16.0)
Requirement already satisfied: charset-normalizer<3,=>2 in /usr/local/lib/python3.7/dist-packages (from requests>=2.26->yfinance) (2.0.12)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests>=2.26->yfinance) (2022.9.24)
Requirement already satisfied: urllib3<1.27,=>1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests>=2.26->yfinance) (1.26.15)
Requirement already satisfied: idna<4,=>2.5 in /usr/local/lib/python3.7/dist-packages (from requests>=2.26->yfinance) (2.10)
Requirement already satisfied: pyparsing<3,=>2.4.7 in /usr/local/lib/python3.7/dist-packages (from lxml>=4.5.1->yfinance) (2.4.7)
```

```
#import all libraries
import pandas as pd
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (10,6)
import datetime
import yfinance as yf
import warnings
warnings.filterwarnings('ignore')
plt.style.use('seaborn')
```


```
#load the data
india = pd.read_csv("/content/drive/My Drive/Data/Finance-data/india.csv" )
india.head()
```

↗

	DATE	CCUSMA02INQ618N
0	2010-01-01	45.928067
1	2010-04-01	45.625182
2	2010-07-01	46.487500
3	2010-10-01	44.862500
4	2011-01-01	45.273600


```
india.rename({'CCUSMA02INQ618N': 'Rupee'}, axis=1, inplace=True) #rename column name
```

```
india = india.set_index('DATE') #make the date column as index
india.head()
```

 **Rupee**

DATE	
2010-01-01	45.928067
2010-04-01	45.625182
2010-07-01	46.487500
2010-10-01	44.862500
2011-01-01	45.273600

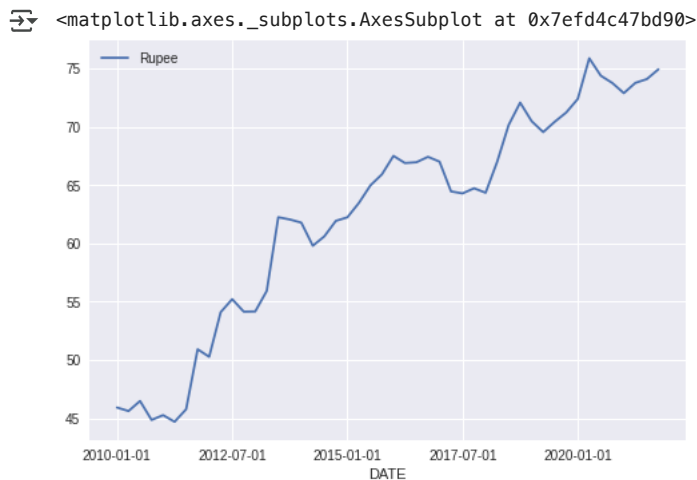
```
india.describe()
```

 **Rupee**

count	48.000000
mean	62.396424
std	9.514285
min	44.708233
25%	54.956283
50%	64.401164
75%	70.221093
max	75.874401

Visualize the data

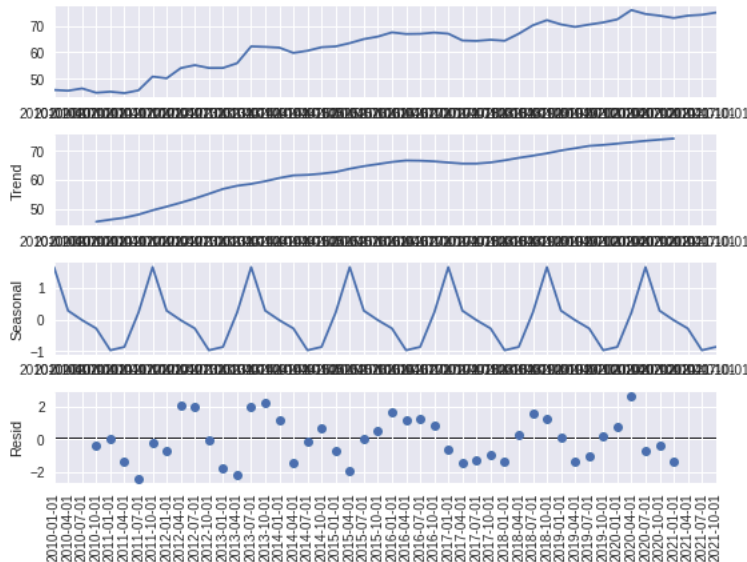
```
india.plot()
```



▼ Additive decomposition

```
from statsmodels.tsa.seasonal import seasonal_decompose
decomposition = seasonal_decompose(india, model='additive', period=7)
decomposition.plot()
plt.xticks(rotation=90)
plt.legend()
plt.show()
```

WARNING:matplotlib.legend:No handles with labels found to put in legend.



In the first graph, we see the behavior of the raw data. Overall, it has a upward trend from 2010 to 2022.

The second plot exhibits the trend of the data. Trend shows the overall movement of a time series. We can say that there exist a trend if there is a long-term increase or decrease in the data. Here we see the trend of rise throughout the time period. Trend needs to be removed before the modeling stage and this process is known as de-trending process.

Seasonality is the third plot which shows periodical ups and downs in the data. If there is a seasonality in the data, it should also be removed. Here we see the ups and downs in the exchange rates.

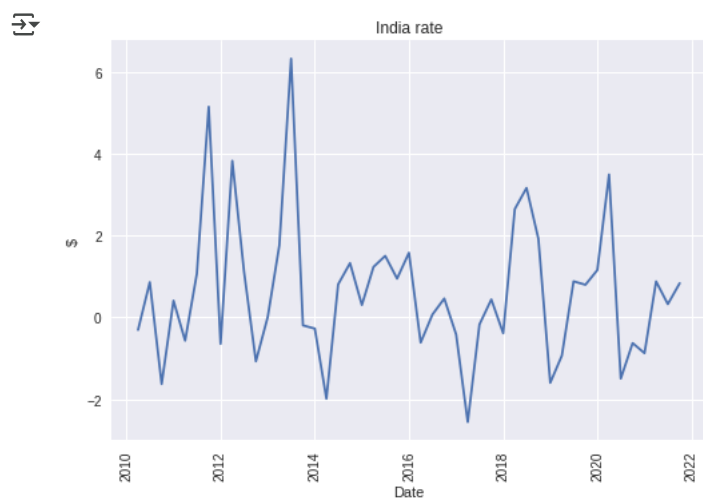
The last graph show the residuals. This is obtained after removing the trend and seasonal components from the time series. If we end up with a rather stable residual, it means that our data is stationary structure and ready to move on the modeling part. Here our data looks stationary.

Detecting and handling with stationarity

```
india_diff = india.diff().dropna()
india_diff.head()
```

Rupee	
DATE	
2010-04-01	-0.302885
2010-07-01	0.862318
2010-10-01	-1.625000
2011-01-01	0.411100
2011-04-01	-0.565367

```
plt.plot(india_diff)
plt.xlabel('Date')
plt.ylabel('$')
plt.title('India rate')
plt.xticks(rotation=90)
plt.show()
```



✓ Statistical Test for detecting stationarity: ADF

```
from statsmodels.tsa.stattools import adfuller
stat_test = adfuller(india)
print('The test statistic is {} and p-value is {}'.format(stat_test[0], stat_test[1]))
```

→ The test statistic is -1.7782341897007123 and p-value is 0.391320092850766

It suggests that the time series is non-stationary as it is higher p value than 0.05.

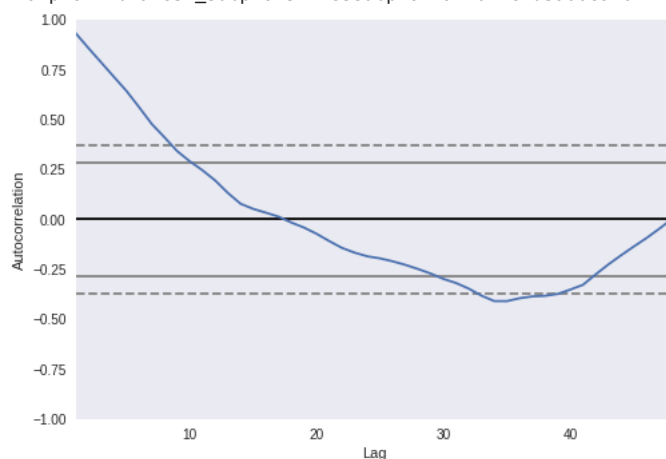
```
from statsmodels.tsa.stattools import adfuller
stat_test = adfuller(india_diff)
print('The test statistic is {} and p-value is {}'.format(stat_test[0], stat_test[1]))
```

→ The test statistic is -4.858874252646454 and p-value is 4.190602046423047e-05

After taking the first difference, the time series becomes stationary.

```
from pandas.plotting import autocorrelation_plot
autocorrelation_plot(india)
```

→ <matplotlib.axes._subplots.AxesSubplot at 0x7efd3a00e910>

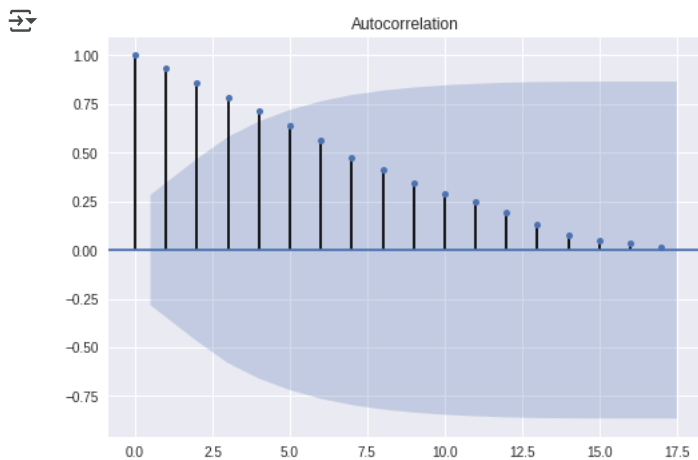


Running the example, we can see that there is a positive correlation with the first 1 to 18 lags that is perhaps significant for the first 17 lags.

A good starting point for the AR parameter of the model may be 17.

ACF

```
import statsmodels.api as sm
sm.graphics.tsa.plot_acf(india, lags=17);
```



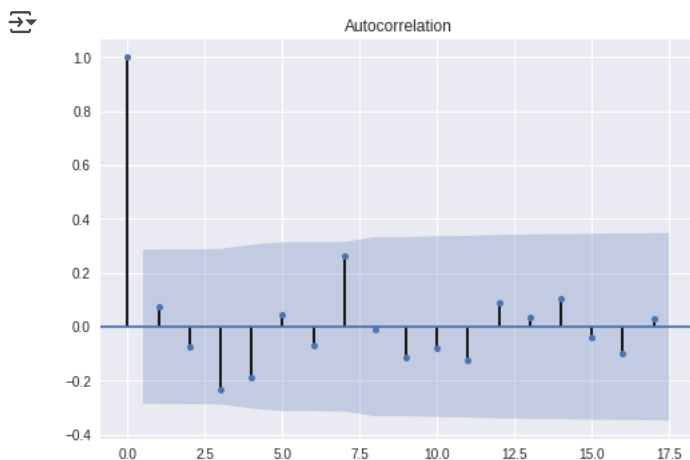
The acf plot indicates that acf is slowly decaying, which is an indicator of non-stationarity. Besides, the blue shaded area represents the confidence interval. If the bar is in this confidence interval, it indicates that there is no significant correlation between this lag and the current one and this information is used for identifying the optimal lag. As this data is non-stationary, it makes no sense to apply this information here.

Identification of an AR model is often best done with the PACF.

For an AR model, the theoretical PACF “shuts off” past the order of the model. The phrase “shuts off” means that in theory the partial autocorrelations are equal to 0 beyond that point. Put another way, the number of non-zero partial autocorrelations gives the order of the AR model. By the “order of the model” we mean the most extreme lag of x that is used as a predictor. Identification of an MA model is often best done with the ACF rather than the PACF.

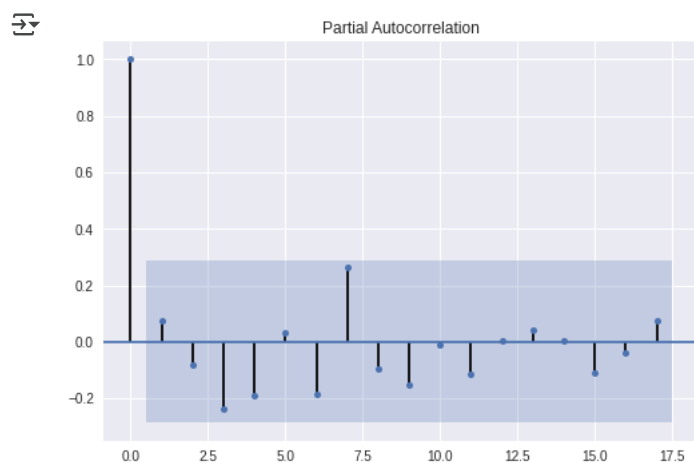
For an MA model, the theoretical PACF does not shut off, but instead tapers toward 0 in some manner. A clearer pattern for an MA model is in the ACF. The ACF will have non-zero autocorrelations only at lags involved in the model. p, d, q p AR model lags d differencing q MA lags

```
import statsmodels.api as sm
sm.graphics.tsa.plot_acf(india_diff, lags=17);
```



PACF

```
import statsmodels.api as sm
sm.graphics.tsa.plot_pacf(india_diff, lags=17);
```



✓ Autoregressive Integrated Moving Average - ARIMA

```
train_len = int(len(india_diff) * 0.8)
train = india_diff[:train_len]
test = india_diff[train_len:]
```

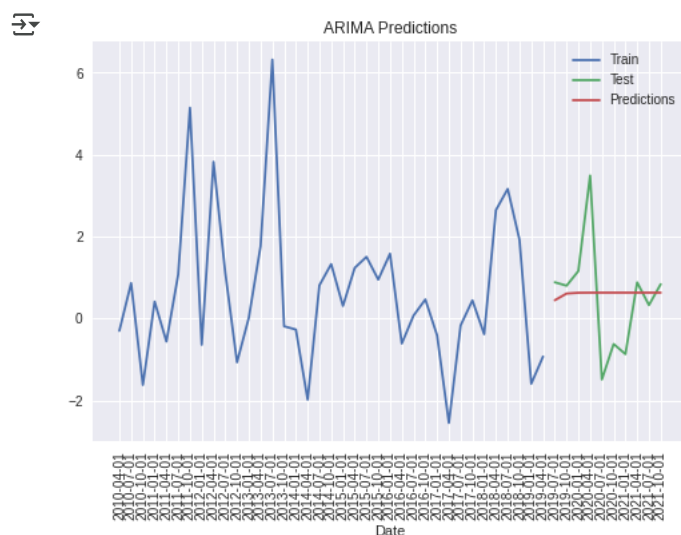
```
start = len(train)
end = len(train) + len(test) - 1
```

```
from statsmodels.tsa.arima.model import ARIMA
arima_model = ARIMA(train, order=(1,1,1))
arima_results = arima_model.fit()
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:527: ValueWarning: No frequency information was pro
% freq, ValueWarning)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:527: ValueWarning: No frequency information was pro
% freq, ValueWarning)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:527: ValueWarning: No frequency information was pro
% freq, ValueWarning)
```

```
arima_predict = arima_results.predict(start, end)
arima_predict.index = test.index
```

```
plt.plot(train, label='Train')
plt.plot(test, label='Test')
plt.plot(arima_predict, label='Predictions')
plt.title('ARIMA Predictions')
plt.legend()
plt.xticks(rotation = 90)
plt.xlabel('Date')
plt.show()
```



The above shows the ARIMA predictions for Indian Rupee.

```
arima_pred_diff = arima_predict.diff().dropna()
```

```
from sklearn.metrics import mean_absolute_error, mean_squared_error
mae_arima = mean_absolute_error(test.iloc[1:], arima_pred_diff)
mse_arima = mean_squared_error(test.iloc[1:], arima_pred_diff)
rmse_arima = np.sqrt(mean_squared_error(test.iloc[1:], arima_pred_diff))
```

```
arima_perf = {'mae_arima': mae_arima,
              'mse_arima': mse_arima,
              'rmse_arima': rmse_arima}
arima_perf = pd.DataFrame([arima_perf])
arima_perf
```

	mae_arima	mse_arima	rmse_arima
0	1.143369	2.090771	1.44595

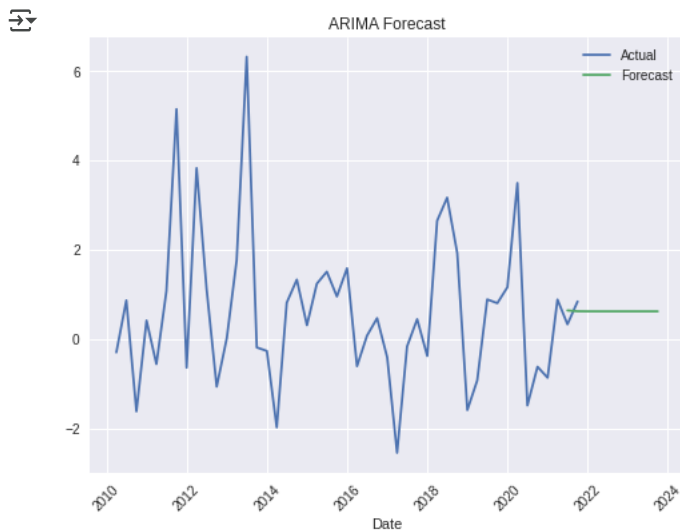
```
india_diff.index = pd.date_range(start='2010-04-01', freq = '3MS' , periods=47)
```

```
forecast_index = pd.date_range(start='2021-07-01', freq = '3MS' , periods=10)
forecast_index
```

```
DatetimeIndex(['2021-07-01', '2021-10-01', '2022-01-01', '2022-04-01',
                '2022-07-01', '2022-10-01', '2023-01-01', '2023-04-01',
                '2023-07-01', '2023-10-01'],
              dtype='datetime64[ns]', freq='3MS')
```

```
arima_model = ARIMA(india_diff, order=(1, 1, 1)).fit()
arima_forecast = arima_model.forecast(steps=10)
arima_forecast.index = forecast_index
```

```
plt.figure(figsize=(8, 6))
plt.plot(india_diff, label='Actual')
plt.plot(arima_forecast, label='Forecast')
plt.title('ARIMA Forecast')
plt.xlabel('Date')
plt.xticks(rotation=45)
plt.legend()
plt.show()
```



Above shows the forecast from 2021 to 2024 denoted by the straight line.

Forecasting of Euro

```
euro = pd.read_csv("/content/drive/My Drive/Data/Finance-data/europe.csv" )
euro.head()
```

	DATE	CCUSMA02EZQ618N
0	2010-01-01	0.722664
1	2010-04-01	0.786976
2	2010-07-01	0.773788
3	2010-10-01	0.736746
4	2011-01-01	0.731408

```
euro.rename({'CCUSMA02EZQ618N': 'euro'}, axis=1, inplace=True)
```

```
euro = euro.set_index('DATE')
euro.head()
```

	euro
DATE	
2010-01-01	0.722664
2010-04-01	0.786976
2010-07-01	0.773788
2010-10-01	0.736746
2011-01-01	0.731408

```
euro.describe()
```




```

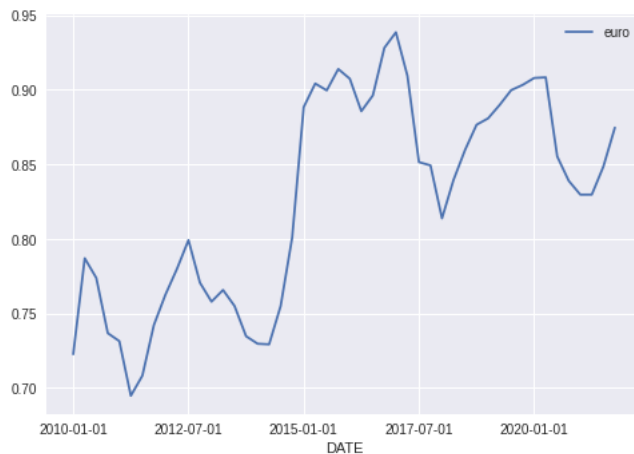
euro
count 48.000000
mean   0.826317
std    0.070725
min    0.694784
25%    0.761370
50%    0.839178
75%    0.891350
max    0.938476

```

```
euro.plot()
```



```
<matplotlib.axes._subplots.AxesSubplot at 0x7efd3164a790>
```

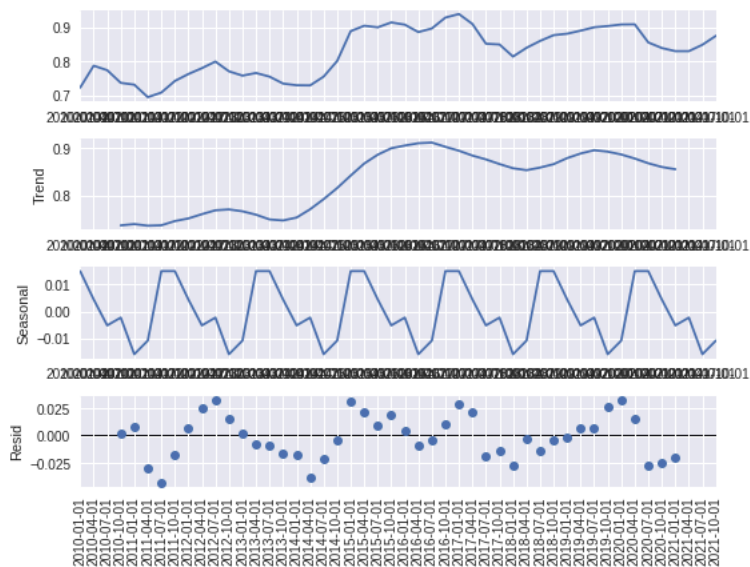


▼ Additive decomposition

```

from statsmodels.tsa.seasonal import seasonal_decompose
decomposition = seasonal_decompose(euro, model='additive', period=7)
decomposition.plot()
plt.xticks(rotation=90)
plt.show()

```



In the first graph, we see the behavior of the raw data. We can see that there have been ups and downs in the overall trend from 2010 to 2022.

The second plot exhibits the trend of the data. Trend shows the overall movement of a time series. We can say that there exist a trend if there is a long-term increase or decrease in the data. Here we see the trend of rise throughout the time period. first a gradual rise and followed by a huge rise and drop. Trend needs to be removed before the modeling stage and this process is known as de-trending process.

Seasonality is the third plot which shows periodical ups and downs in the data. If there is a seasonality in the data, it should also be removed. Here we see the ups and downs in the exchange *rates*.

The last graph show the residuals. This is obtained after removing the trend and seasonal components from the time series. If we end up with a rather stable residual, it means that our data is stationary structure and ready to move on the modeling part. Here our data looks stationary.

Detecting and handling with stationarity

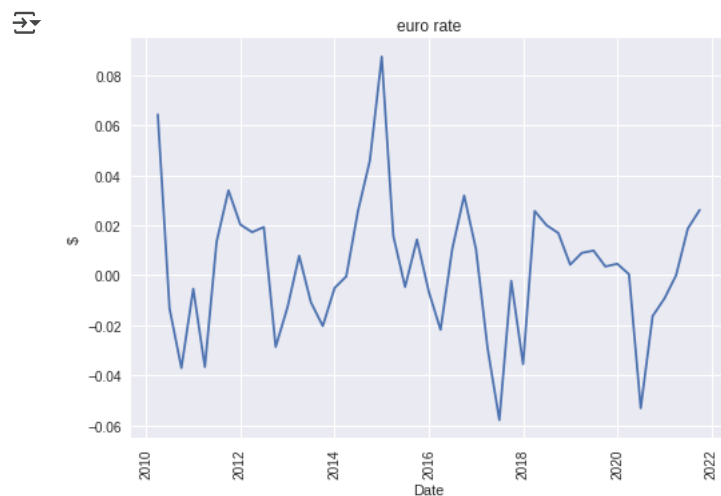
```
euro_diff = euro.diff().dropna()
euro_diff.head()
```

```

euro
DATE
2010-04-01    0.064312
2010-07-01   -0.013188
2010-10-01   -0.037042
2011-01-01   -0.005337
2011-04-01   -0.036625

```

```
plt.plot(euro_diff)
plt.xlabel('Date')
plt.xticks(rotation = 90)
plt.ylabel('$')
plt.title('euro rate')
plt.show()
```



```
from statsmodels.tsa.stattools import adfuller
stat_test = adfuller(euro)
print('The test statistic is {} and p-value is {}'.format(stat_test[0], stat_test[1]))
```

```

The test statistic is -1.6858741836030828 and p-value is 0.4384048239430989

```

It suggests that the time series is non-stationary as it is higher p value than 0.05.

```
from statsmodels.tsa.stattools import adfuller
stat_test = adfuller(euro_diff)
print('The test statistic is {} and p-value is {}'.format(stat_test[0], stat_test[1]))
```


```

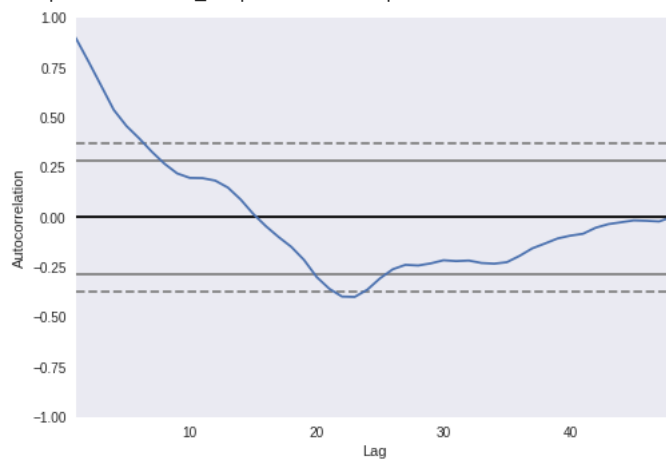
The test statistic is -4.974884648040398 and p-value is 2.4878442985924105e-05

```

After taking the first difference, the time series becomes stationary.

```
from pandas.plotting import autocorrelation_plot
autocorrelation_plot(euro)
```

 <matplotlib.axes._subplots.AxesSubplot at 0x7efd3eb29450>



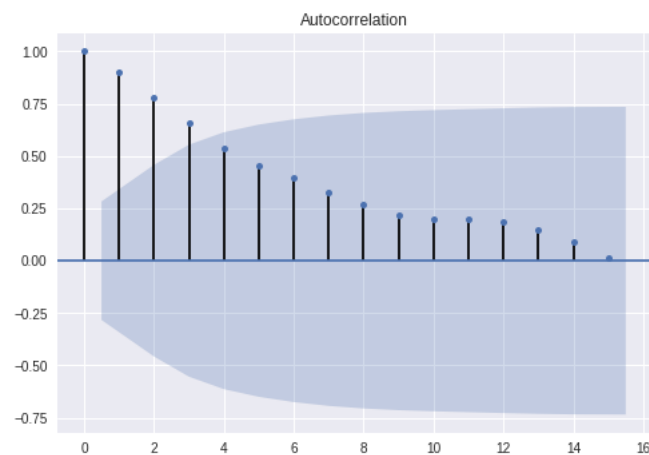
Running the example, we can see that there is a positive correlation with the first 1-to-15 lags that is perhaps significant for the first 15 lags.

A good starting point for the AR parameter of the model may be 15.

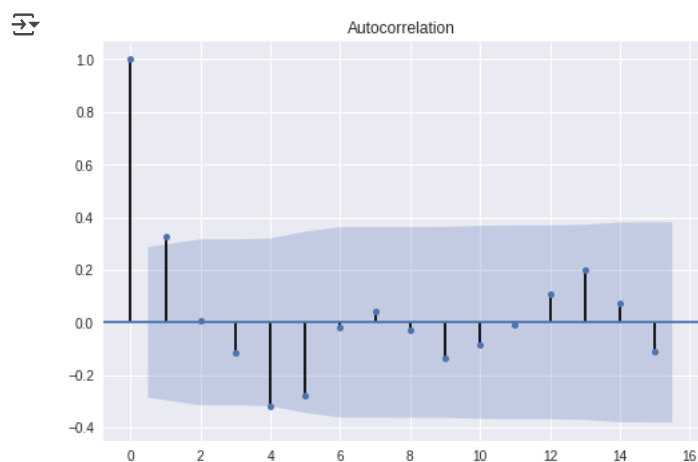
ACF

```
import statsmodels.api as sm
sm.graphics.tsa.plot_acf(euro, lags=15);
```



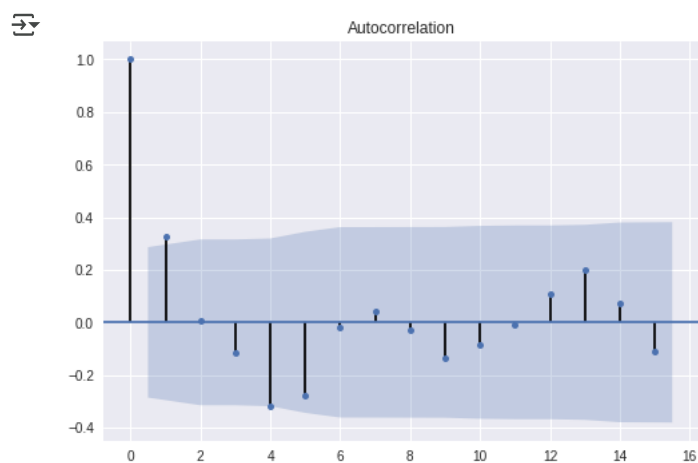


```
import statsmodels.api as sm
sm.graphics.tsa.plot_acf(euro_diff, lags=15);
```



PACF

```
import statsmodels.api as sm
sm.graphics.tsa.plot_acf(euro_diff, lags=15);
```



To find out the value of q we can use the ACF plot. Which will tell us how much moving average is required to remove the autocorrelation from the stationary time series.

Significant correlation in a stationary time series can be represented by adding auto regression terms. Using the PACF plot we can take the order of AR terms to be equal to the lags that can cross a significance limit.

Hence we are choosing p as 1 q as 1 and d value as 1.

```
train_len = int(len(euro_diff) * 0.8)
train = euro_diff[:train_len]
test = euro_diff[train_len:]
```

```
start = len(train)
end = len(train) + len(test) - 1
```

```
from statsmodels.tsa.arima.model import ARIMA
arima_model = ARIMA(train, order=(1,1,1))
arima_results = arima_model.fit()
```

```
ValueWarning: No frequency information was pro
% freq, ValueWarning)
ValueWarning: No frequency information was pro
% freq, ValueWarning)
ValueWarning: No frequency information was pro
% freq, ValueWarning)
```

```

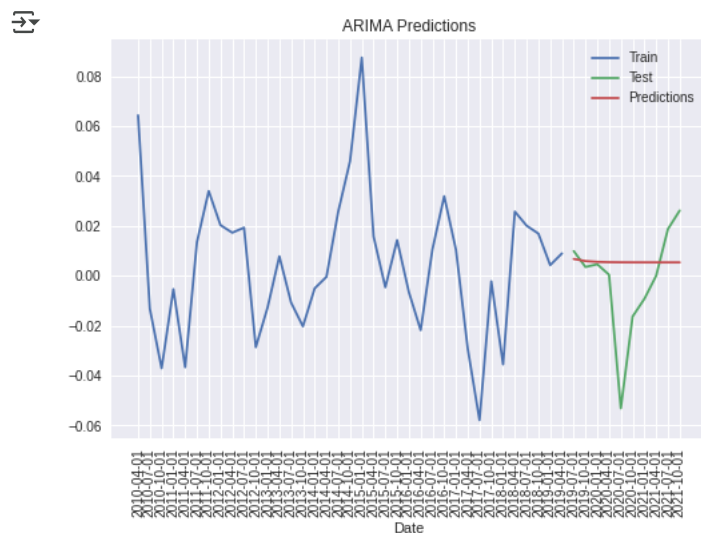
arima_predict = arima_results.predict(start, end)
arima_predict.index = test.index

```

```

plt.plot(train, label='Train')
plt.plot(test, label='Test')
plt.plot(arima_predict, label='Predictions')
plt.title('ARIMA Predictions')
plt.legend()
plt.xlabel('Date')
plt.xticks(rotation = 90)
plt.show()

```



The diagram shows that the test data follows a similar trend as that of the train data and the predictions have a slight fall then remain constant.

```

arima_pred_diff = arima_predict.diff().dropna()

```

```

from sklearn.metrics import mean_absolute_error, mean_squared_error
mae_arima = mean_absolute_error(test.iloc[1:], arima_pred_diff)
mse_arima = mean_squared_error(test.iloc[1:], arima_pred_diff)
rmse_arima = np.sqrt(mean_squared_error(test.iloc[1:], arima_pred_diff))

```

```

arima_perf = {'mae_arima': mae_arima,
              'mse_arima': mse_arima,
              'rmse_arima': rmse_arima}
arima_perf = pd.DataFrame([arima_perf])
arima_perf

```

```

mae_arima  mse_arima  rmse_arima
0    0.014817    0.000471    0.021712

```

```

euro_diff.index = pd.date_range(start='2010-04-01', freq = '3MS' , periods=47)

```

```

forecast_index = pd.date_range(start='2021-07-01', freq = '3MS' , periods=10)
forecast_index

```

```

DatetimeIndex(['2021-07-01', '2021-10-01', '2022-01-01', '2022-04-01',
               '2022-07-01', '2022-10-01', '2023-01-01', '2023-04-01',
               '2023-07-01', '2023-10-01'],
              dtype='datetime64[ns]', freq='3MS')

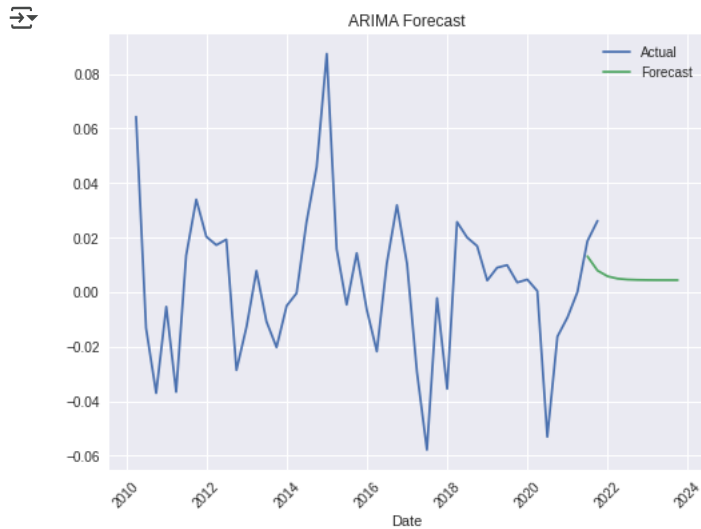
```

```

arima_model = ARIMA(euro_diff, order=(1, 1, 1)).fit()
arima_forecast = arima_model.forecast(steps=10)
arima_forecast.index = forecast_index

```

```
plt.figure(figsize=(8, 6))
plt.plot(euro_diff, label='Actual')
plt.plot(arma_forecast, label='Forecast')
plt.title('ARIMA Forecast')
plt.xlabel('Date')
plt.xticks(rotation=45)
plt.legend()
plt.show()
```



The chart above shows the forecast from 2021 to 2024, observing a fall in the initial month followed by a constant rate throughout the consecutive months.

Forecasting of Pound

```
pound = pd.read_csv("/content/drive/My Drive/Data/Finance-data/pound.csv" )
pound.head()
```

	DATE	CCUSMA02EZQ618N
0	2010-01-01	0.722664
1	2010-04-01	0.786976
2	2010-07-01	0.773788
3	2010-10-01	0.736746
4	2011-01-01	0.731408

```
pound.rename({'CCUSMA02EZQ618N': 'pound'}, axis=1, inplace=True)
```

```
pound = pound.set_index('DATE')
pound.head()
```

	pound
DATE	
2010-01-01	0.722664
2010-04-01	0.786976
2010-07-01	0.773788
2010-10-01	0.736746
2011-01-01	0.731408

```
pound.describe()
```



```
pound
count    48.000000
mean     0.826317
std      0.070725
min      0.694784
25%      0.761370
50%      0.839178
75%      0.891350
max      0.938476
```

```
pound.plot() #line plot
```

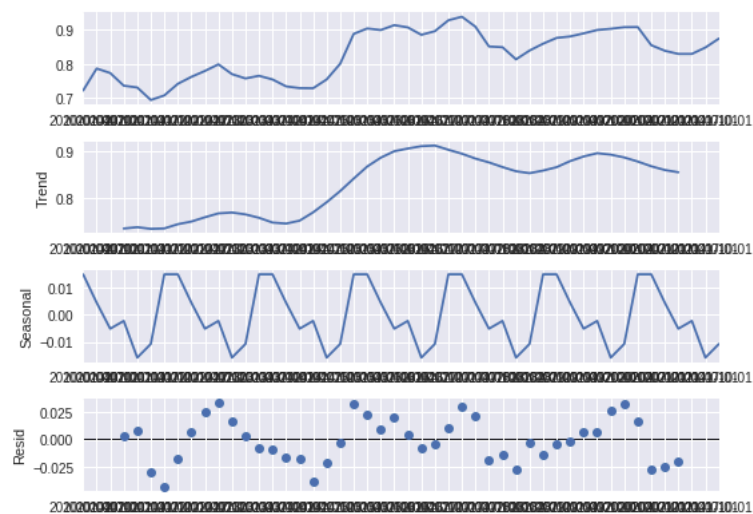


```
<matplotlib.axes._subplots.AxesSubplot at 0x7efd30907690>
```



▼ Additive decomposition

```
from statsmodels.tsa.seasonal import seasonal_decompose
decomposition = seasonal_decompose(pound, model='additive', period=7)
decomposition.plot()
plt.show()
```



This one is pretty similar to the Euro. In the first graph, we see the behavior of the raw data. We can see that there have been ups and downs in the overall trend from 2010 to 2022.

The second plot exhibits the trend of the data. Trend shows the overall movement of a time series. We can say that there exist a trend if there is a long-term increase or decrease in the data. Here we see the trend of rise throughout the time period. first a gradual rise and followed by a huge rise and drop. Trend needs to be removed before the modeling stage and this process is known as de-trending process.

Seasonality is the third plot which shows periodical ups and downs in the data. If there is a seasonality in the data, it should also be removed. Here we see the ups and downs in the exchange *rates*.

The last graph shows the residuals. This is obtained after removing the trend and seasonal components from the time series. If we end up with a rather stable residual, it means that our data is stationary structure and ready to move on the modeling part. Here our data looks stationary.

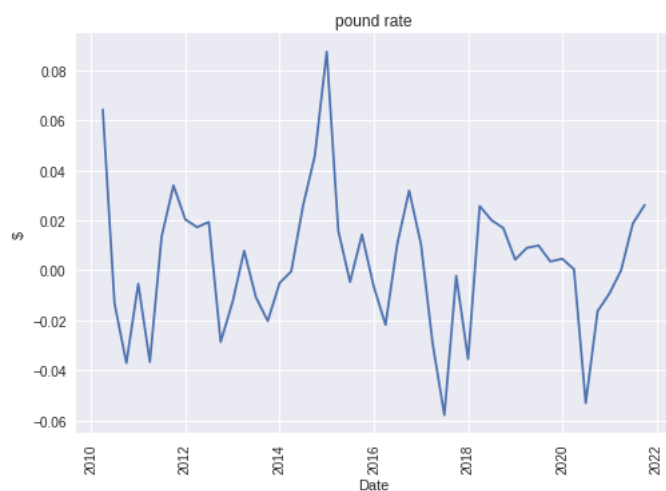
✓ Detecting and handling with stationarity

```
pound_diff = pound.diff().dropna()
pound_diff.head()
```



pound	
DATE	
2010-04-01	0.064312
2010-07-01	-0.013188
2010-10-01	-0.037042
2011-01-01	-0.005337
2011-04-01	-0.036625

```
plt.plot(pound_diff)
plt.xlabel('Date')
plt.ylabel('$')
plt.xticks(rotation=90)
plt.title('pound rate')
plt.show()
```



✓ Statistical Test for detecting stationarity: ADF

```
from statsmodels.tsa.stattools import adfuller
stat_test = adfuller(pound)
print('The test statistic is {} and p-value is {}'.format(stat_test[0], stat_test[1]))
```



The test statistic is -1.6858741836030828 and p-value is 0.4384048239430989

It suggests that the time series is non-stationary as it is higher p value than 0.05.


```
from statsmodels.tsa.stattools import adfuller
stat_test = adfuller(pound_diff)
print('The test statistic is {} and p-value is {}'.format(stat_test[0], stat_test[1]))
```

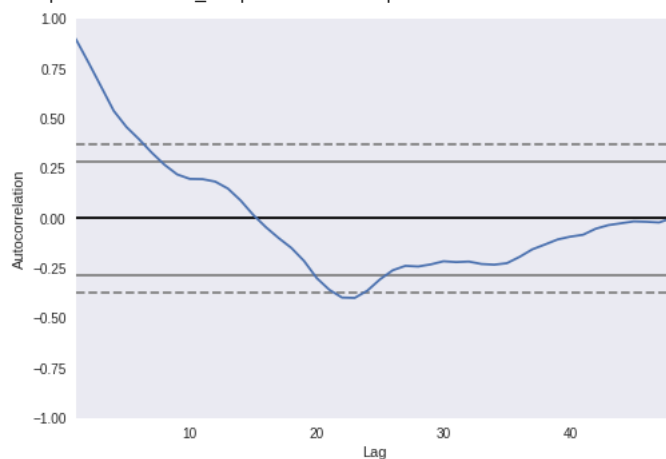


The test statistic is -4.974884648040398 and p-value is 2.4878442985924105e-05

After taking the first difference, the time series becomes stationary.

```
from pandas.plotting import autocorrelation_plot
autocorrelation_plot(pound)
```

 <matplotlib.axes._subplots.AxesSubplot at 0x7efd30aa1e90>

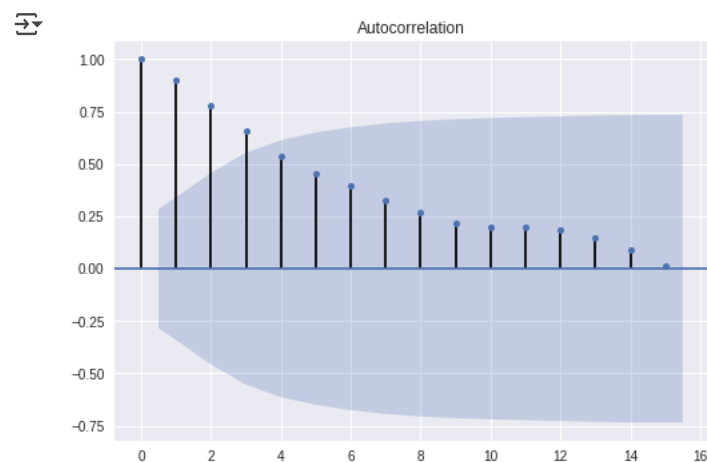


Running the example, we can see that there is a positive correlation with the first 10-to-15 lags that is perhaps significant for the first 15 lags.

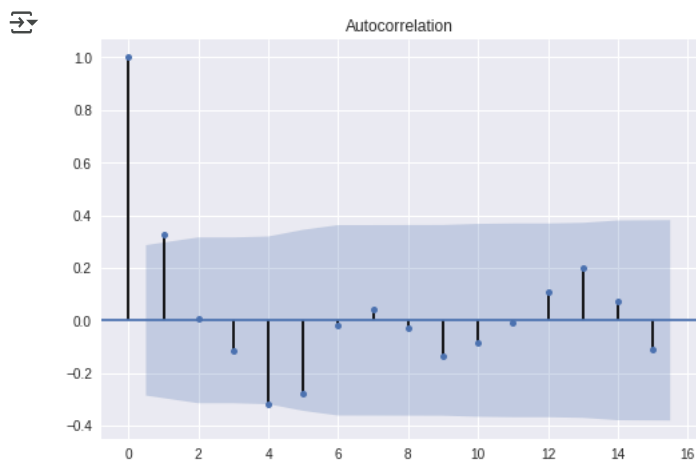
A good starting point for the AR parameter of the model may be 15.

ACF

```
import statsmodels.api as sm
sm.graphics.tsa.plot_acf(pound, lags=15);
```

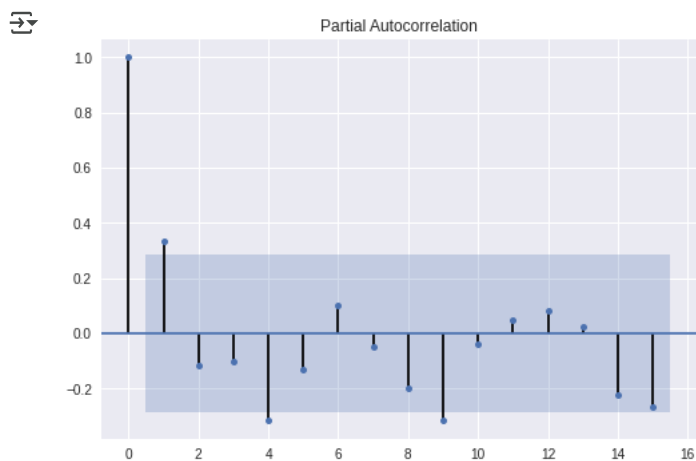


```
import statsmodels.api as sm
sm.graphics.tsa.plot_acf(pound_diff, lags=15);
```



PACF

```
import statsmodels.api as sm
sm.graphics.tsa.plot_pacf(pound_diff, lags=15);
```



To find out the value of q we can use the ACF plot. Which will tell us how much moving average is required to remove the autocorrelation from the stationary time series.

Significant correlation in a stationary time series can be represented by adding auto regression terms. Using the PACF plot we can take the order of AR terms to be equal to the lags that can cross a significance limit.

Here we can see that the 2nd lag goes to negative and the level of significance is seen on the 1st lag hence, we have taken 1 as the value for P . and similarly for value Q the 2nd lag value is 0, hence we could take either 1 or 2 and we have chosen 1 for this.

ARIMA

```
train_len = int(len(pound_diff) * 0.8)
train = pound_diff[:train_len]
test = pound_diff[train_len:]
```

```
start = len(train)
end = len(train) + len(test) - 1
```

```
from statsmodels.tsa.arima.model import ARIMA
arima_model = ARIMA(train, order=(1,1,1))
arima_results = arima_model.fit()
```

```

/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:527: ValueWarning: No frequency information was pro
% freq, ValueWarning)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:527: ValueWarning: No frequency information was pro
% freq, ValueWarning)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:527: ValueWarning: No frequency information was pro
% freq, ValueWarning)

```

```
arima_predict = arima_results.predict(start, end)
arima_predict.index = test.index
```

```
plt.plot(train, label='Train')
plt.plot(test, label='Test')
plt.plot(arima_predict, label='Predictions')
plt.title('ARIMA Predictions')
plt.legend()
plt.xlabel('Date')
plt.xticks(rotation = 90)
plt.show()
```

