Question 1:Time Series Analysis

In [1]:
```python
# import the importent libraries
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
%matplotlib inline
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.arima_model import ARIMA
import statsmodels.api as sm
import statsmodels.tsa.api as smtsa

import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```python
# Set plot style
plt.style.use('seaborn')

# Set plot size
plt.rcParams['figure.figsize'] = [8,6]
```

In [3]:
```python
# Load the data from a CSV file
df=pd.read_csv('dataset/euro-daily-hist_1999_2022.csv')
```

In [4]:
```python
#See first 5 variables
df.head()
```

Out[4]:

	Period\Unit:	[Australian dollar]	[Bulgarian lev]	[Brazilian real]	[Canadian dollar]	[Swiss franc]	[Chinese yuan renminbi]	[Cypriot pound]	[Czech koruna]	[Danish krone]	...	[Romanian leu]	[Russian rouble]	[Swedish krona]
0	2022-04-08	1.4552	1.9558	5.1583	1.3675	1.0155	6.9115	NaN	24.479	7.4372	...	4.9425	NaN	10.2768
1	2022-04-07	1.4578	1.9558	5.1460	1.3704	1.0185	6.9448	NaN	24.512	7.4378	...	4.9419	NaN	10.3130
2	2022-04-06	1.4431	1.9558	5.0996	1.3647	1.0187	6.9498	NaN	24.441	7.4378	...	4.9433	NaN	10.2855
3	2022-04-05	1.4374	1.9558	5.0384	1.3647	1.0141	6.9783	NaN	24.338	7.4378	...	4.9438	NaN	10.2593
4	2022-04-04	1.4651	1.9558	5.1162	1.3749	1.0203	7.0026	NaN	24.320	7.4385	...	4.9432	NaN	10.3849

5 rows × 41 columns

In [5]:
```python
# Print the DataFrame info
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6022 entries, 0 to 6021
Data columns (total 41 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Period\Unit:               6022 non-null   object
 1   [Australian dollar ]       6022 non-null   object
 2   [Bulgarian lev ]           5620 non-null   object
 3   [Brazilian real ]          5754 non-null   object
 4   [Canadian dollar ]         6022 non-null   object
 5   [Swiss franc ]             6022 non-null   object
 6   [Chinese yuan renminbi ]   5754 non-null   object
 7   [Cypriot pound ]           2346 non-null   object
 8   [Czech koruna ]            6022 non-null   object
 9   [Danish krone ]            6022 non-null   object
 10  [Estonian kroon ]          3130 non-null   object
 11  [UK pound sterling ]       6022 non-null   object
 12  [Greek drachma ]           520 non-null    object
 13  [Hong Kong dollar ]        6022 non-null   object
 14  [Croatian kuna ]           5754 non-null   object
 15  [Hungarian forint ]        6022 non-null   object
 16  [Indonesian rupiah ]       6022 non-null   object
 17  [Israeli shekel ]          5754 non-null   object
 18  [Indian rupee ]            5754 non-null   object
 19  [Iceland krona ]           3615 non-null   float64
 20  [Japanese yen ]            6022 non-null   object
 21  [Korean won ]              6022 non-null   object
 22  [Lithuanian litas ]        4159 non-null   object
 23  [Latvian lats ]            3904 non-null   object
 24  [Maltese lira ]            2346 non-null   object
 25  [Mexican peso ]            6022 non-null   object
 26  [Malaysian ringgit ]       6022 non-null   object
 27  [Norwegian krone ]         6022 non-null   object
 28  [New Zealand dollar ]      6022 non-null   object
 29  [Philippine peso ]         6022 non-null   object
 30  [Polish zloty ]            6022 non-null   object
 31  [Romanian leu ]            5960 non-null   float64
 32  [Russian rouble ]          5994 non-null   object
 33  [Swedish krona ]           6022 non-null   object
 34  [Singapore dollar ]        6022 non-null   object
 35  [Slovenian tolar ]         2085 non-null   object
 36  [Slovak koruna ]           2608 non-null   object
 37  [Thai baht ]               6022 non-null   object
 38  [Turkish lira ]            5960 non-null   float64
 39  [US dollar ]               6022 non-null   object
 40  [South African rand ]      6022 non-null   object
dtypes: float64(3), object(38)
memory usage: 1.9+ MB
```

In [6]:
```python
#Seeing shape of dataframe
df.shape
```

Out[6]: (6022, 41)

In [7]:
```python
#Checking if any null values the DataFrame has
df.isnull().sum()
```

Out[7]:
```
Period\Unit:                   0
[Australian dollar ]           0
[Bulgarian lev ]             402
[Brazilian real ]            268
[Canadian dollar ]             0
[Swiss franc ]                 0
[Chinese yuan renminbi ]     268
[Cypriot pound ]            3676
[Czech koruna ]                0
[Danish krone ]                0
[Estonian kroon ]           2892
[UK pound sterling ]           0
[Greek drachma ]            5502
[Hong Kong dollar ]            0
[Croatian kuna ]             268
[Hungarian forint ]            0
[Indonesian rupiah ]           0
[Israeli shekel ]            268
[Indian rupee ]              268
[Iceland krona ]            2407
[Japanese yen ]                0
[Korean won ]                  0
[Lithuanian litas ]         1863
[Latvian lats ]             2118
[Maltese lira ]             3676
[Mexican peso ]                0
[Malaysian ringgit ]           0
[Norwegian krone ]             0
[New Zealand dollar ]          0
[Philippine peso ]             0
[Polish zloty ]                0
[Romanian leu ]               62
[Russian rouble ]             28
[Swedish krona ]               0
[Singapore dollar ]            0
[Slovenian tolar ]          3937
[Slovak koruna ]            3414
[Thai baht ]                   0
[Turkish lira ]               62
[US dollar ]                   0
[South African rand ]          0
dtype: int64
```

The Dataset has plentty of null values but the Turkish Lira column only has 62 which makes the time series less interrupted

In [8]:
```python
#Renaming the columns

df.rename(columns={'[Turkish lira ]': 'Turkish_Lira',
                   'Period\\Unit:': 'Date'},
          inplace=True)

# Store the 'Date' column into dataframe 'djia_df'
df['Date'] = pd.to_datetime(df['Date'])

#Sort values from earlier to latest
df.sort_values('Date', inplace=True)

#Print the first two variables
df.head(2)
```

Out[8]:

	Date	[Australian dollar]	[Bulgarian lev]	[Brazilian real]	[Canadian dollar]	[Swiss franc]	[Chinese yuan renminbi]	[Cypriot pound]	[Czech koruna]	[Danish krone]	...	[Romanian leu]	[Russian rouble]	[Swedish krona]	[S
6021	1999-01-04	1.9100	NaN	NaN	1.8004	1.6168	NaN	0.58231	35.107	7.4501	...	1.3111	25.2875	9.4696	
6020	1999-01-05	1.8944	NaN	NaN	1.7965	1.6123	NaN	0.58230	34.917	7.4495	...	1.3168	26.5876	9.4025	

2 rows × 41 columns

For better experiences the Time column and the value columns name was changed

In [9]:
```python
#Copy the new dataframe from the original dataframe
EU_TL=df[['Date','Turkish_Lira']].copy()
```

In [10]:
```python
# Attach your own 'Data' index to the dataframe
EU_TL.index = EU_TL['Date']

# Drop the 'Date' column from the dataframe
EU_TL.drop('Date', axis = 1, inplace = True)
EU_TL.head(5)
```

Out[10]:

Date	Turkish_Lira
1999-01-04	0.3723
1999-01-05	0.3728
1999-01-06	0.3722
1999-01-07	0.3701
1999-01-08	0.3718

In [11]:
```python
# Drop null values from the dataframe
EU_TL.dropna(inplace=True)
# Display the sum of null values
EU_TL.isnull().sum()
```

Out[11]:
```
Turkish_Lira    0
dtype: int64
```

In [12]:
```python
#See the min and max values
EU_TL.index.min(),EU_TL.index.max()
```

Out[12]: (Timestamp('1999-01-04 00:00:00'), Timestamp('2022-04-08 00:00:00'))

In [13]:
```python
# plotds is method to plot time series, ACF and PACF
def plotds(xt, nlag = 30, fig_size = (12, 10)):
    if not isinstance(xt, pd.Series):
        xt = pd.Series(xt)
    plt.figure(figsize=fig_size)
    layout = (2, 2)

    ax_xt = plt.subplot2grid(layout, (0, 0), colspan = 2)
    ax_acf= plt.subplot2grid(layout, (1, 0))
    ax_pacf = plt.subplot2grid(layout, (1, 1))

    xt.plot(ax = ax_xt)
    ax_xt.set_title('Time Series')
    plot_acf(xt, lags = 50, ax = ax_acf)
    plot_pacf(xt, lags = 50, ax = ax_pacf)
    plt.tight_layout()
    return None
```

In [14]:
```python
## Display plot of Turkish_Lira column and Date index with ACF and PACF
plotds(EU_TL['Turkish_Lira'], nlag = 50)
```

The dataset has 20 years records so that the plot looks very squeshed together so that we wont be able to see more in the detail

In [15]:
```python
from statsmodels.tsa.stattools import adfuller

# Call adfuller() function to calculate the values
adf_result= adfuller(EU_TL.Turkish_Lira.tolist())
print('ADF Statistic: %f' % adf_result[0])
print('p-value: %f' % adf_result[1])
```

```
ADF Statistic: 4.130176
p-value: 1.000000
```

In [16]:
```python
# qqplot for the 'Close' column
x = sm.qqplot(EU_TL['Turkish_Lira'], line = 's')
```

To be able to apply time series to the dataset, the data must be stationary. Stationarity is an important characteristic of time series. A time series is said to be stationary if its statistical properties do not change over time. In other words, it has constant mean and variance, and covariance is independent of time.(From class notes). We can check the plot with naked eye if the time series is stationary but it wont be accurate. The Dickey-Fuller test is one of most easy method to check if the datset is stationary.

The Augmented Dickey-Fuller test:

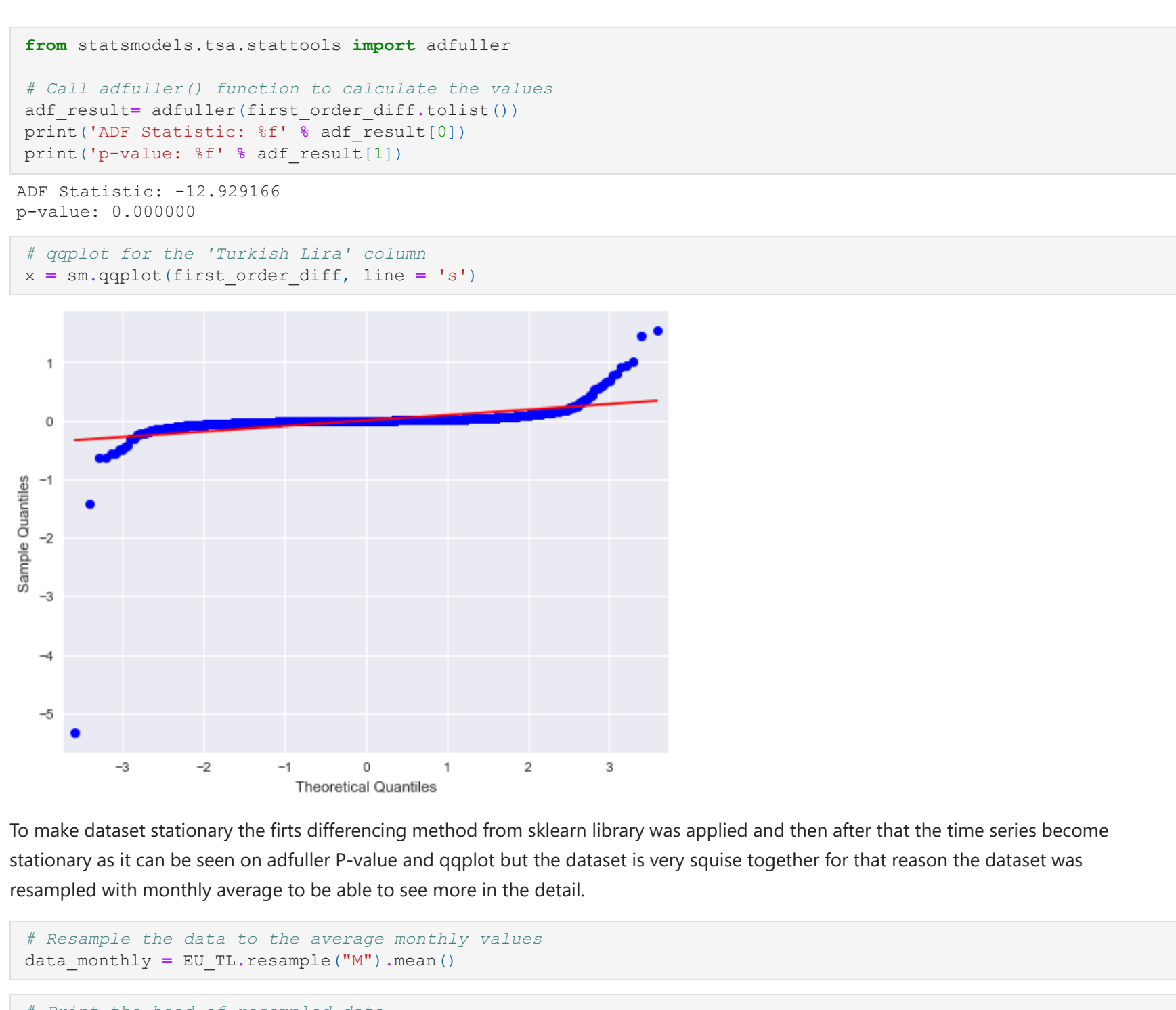
In [17]:
```python
# Store one difference value of the 'Turkish_Lira' column
first_order_diff = EU_TL['Turkish_Lira'].diff(1)

# Display the first five records
first_order_diff.head()
```

Out[17]:
```
Date
1999-01-04       NaN
1999-01-05    0.0005
1999-01-06   -0.0006
1999-01-07   -0.0021
1999-01-08    0.0017
Name: Turkish_Lira, dtype: float64
```

In [18]:
```python
# Drop the first row of 'Turkish_Lira' column because It has 'NaN' value and assign to the new DataFrame
first_order_diff=EU_TL['Turkish_Lira'].diff(1).dropna()

# Display the first five records
first_order_diff.head()
```

Out[18]:
```
Date
1999-01-05    0.0005
1999-01-06   -0.0006
1999-01-07   -0.0021
1999-01-08    0.0017
1999-01-11    0.0007
Name: Turkish_Lira, dtype: float64
```

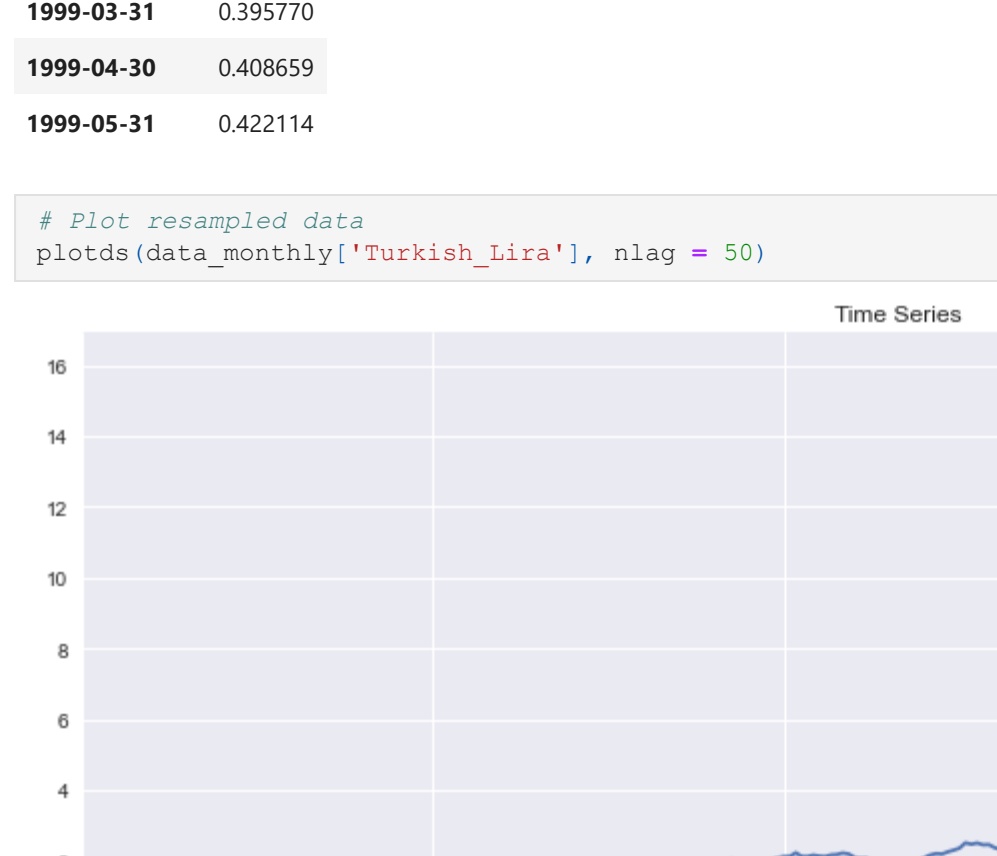
In [19]:
```python
# plot the data with 50 lags
plotds(first_order_diff, nlag = 50)
```



In [20]:
```python
from statsmodels.tsa.stattools import adfuller

# Call adfuller() function to calculate the values
adf_result= adfuller(first_order_diff.tolist())
print('ADF Statistic: %f' % adf_result[0])
print('p-value: %f' % adf_result[1])
```

```
ADF Statistic: -12.929166
p-value: 0.000000
```

In [21]:
```python
# qqplot for the 'Turkish Lira' column
x = sm.qqplot(first_order_diff, line = 's')
```



To make dataset stationary the firts differencing method from sklearn library was applied and then after that the time series become stationary as it can be seen on adfuller P-value and qqplot but the dataset is very squise together for that reason the dataset was resampled with monthly average to be able to see more in the detail.

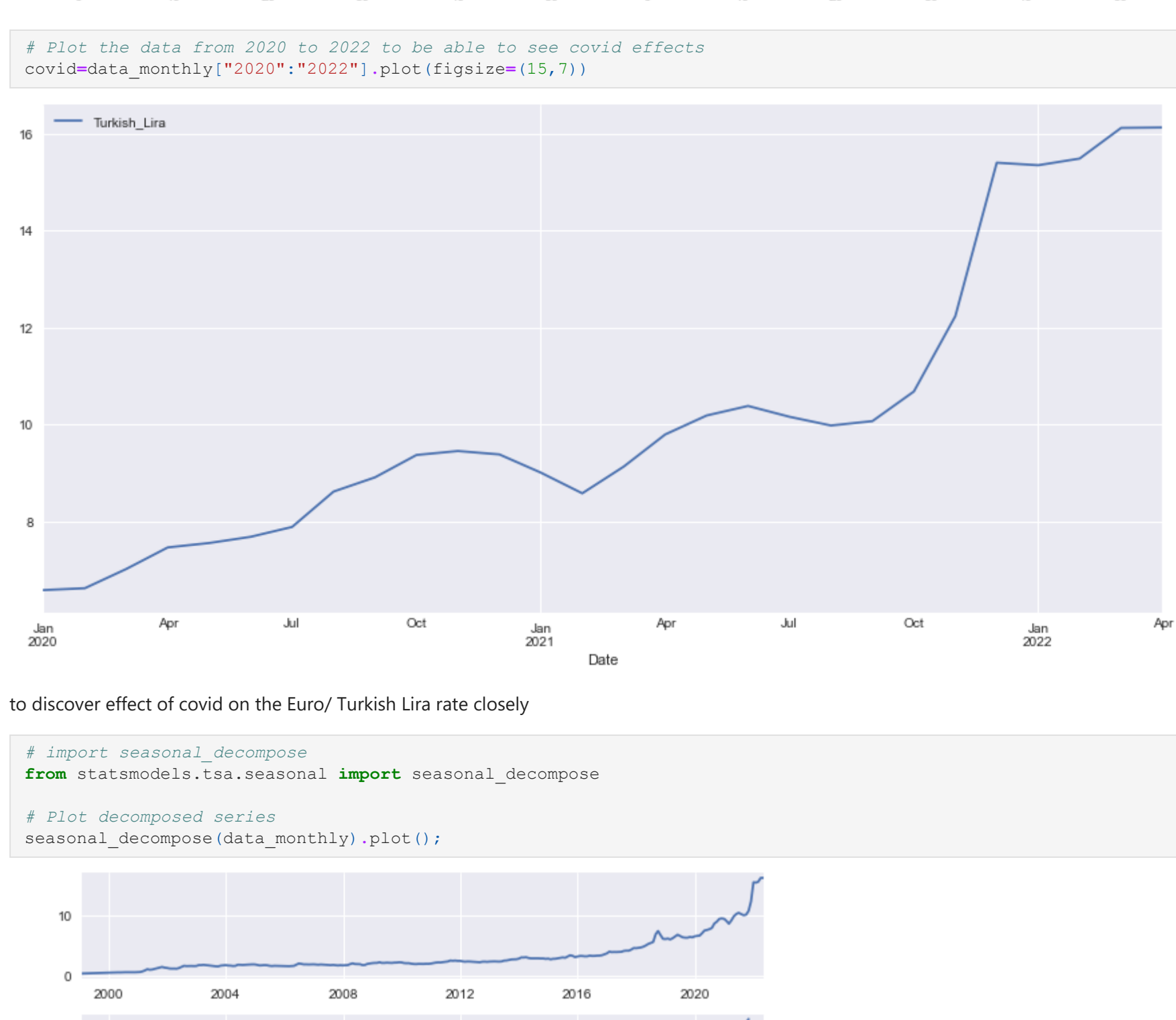
In [22]:
```python
# Resample the data to the average monthly values
data_monthly = EU_TL.resample("M").mean()
```

In [23]:
```python
# Print the head of resampled data
data_monthly.head()
```

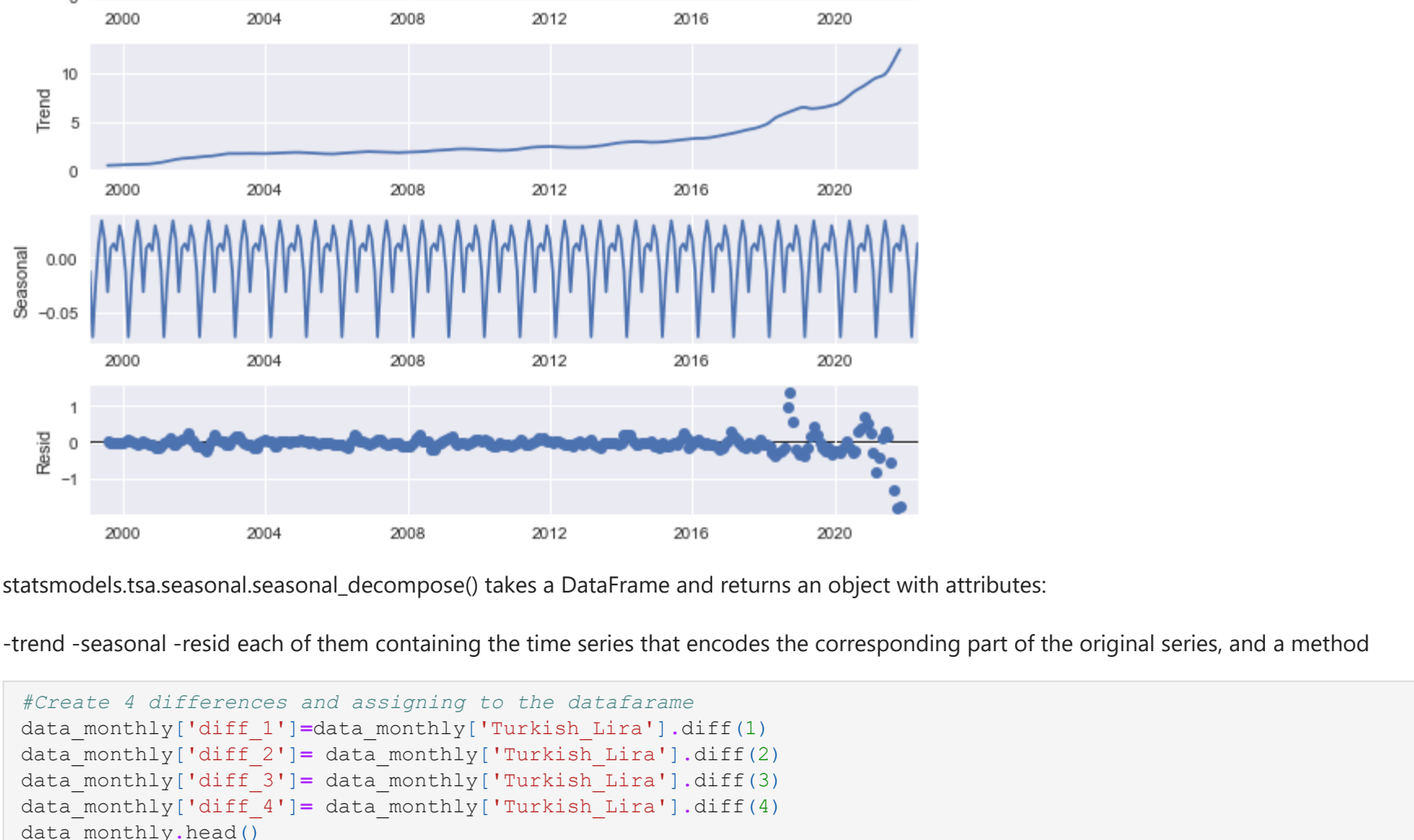
Out[23]:

Date	Turkish_Lira
1999-01-31	0.376465
1999-02-28	0.384990
1999-03-31	0.395770
1999-04-30	0.408659
1999-05-31	0.422114

In [24]:
```python
# Plot resampled data
plotds(data_monthly['Turkish_Lira'], nlag = 50)
```



In [25]:
```python
# Plot the data from 2020 to 2022 to be able to see covid effects
covid=data_monthly["2020":"2022"].plot(figsize=(15,7))
```



to discover effect of covid on the Euro/ Turkish Lira rate closely

In [26]:
```python
# import seasonal_decompose
from statsmodels.tsa.seasonal import seasonal_decompose

# Plot decomposed series
seasonal_decompose(data_monthly).plot();
```

statsmodels.tsa.seasonal.seasonal_decompose() takes a DataFrame and returns an object with attributes:

-trend -seasonal -resid each of them containing the time series that encodes the corresponding part of the original series, and a method

In [27]:
```python
#Create 4 differences and assigning to the dataframe
data_monthly['diff_1']=data_monthly['Turkish_Lira'].diff(1)
data_monthly['diff_2']= data_monthly['Turkish_Lira'].diff(2)
data_monthly['diff_3']= data_monthly['Turkish_Lira'].diff(3)
data_monthly['diff_4']= data_monthly['Turkish_Lira'].diff(4)
data_monthly.head()
```

Out[27]:

Date	Turkish_Lira	diff_1	diff_2	diff_3	diff_4
1999-01-31	0.376465	NaN	NaN	NaN	NaN
1999-02-28	0.384990	0.008525	NaN	NaN	NaN
1999-03-31	0.395770	0.010780	0.019305	NaN	NaN
1999-04-30	0.408659	0.012890	0.023669	0.032194	NaN
1999-05-31	0.422114	0.013455	0.026345	0.037124	0.045649

In [28]:
```python
#Drop null values from differences columns
data_monthly.dropna(inplace=True)
```

In [29]:
```python
from statsmodels.tsa.stattools import adfuller

# Call adfuller() function to calculate the values
adf_result= adfuller(data_monthly.diff_1.tolist())
print('ADF Statistic: %f' % adf_result[0])
print('p-value: %f' % adf_result[1])
```

```
ADF Statistic: 0.920939
p-value: 0.993358
```

In [30]:
```python
from statsmodels.tsa.stattools import adfuller

# Call adfuller() function to calculate the values
adf_result= adfuller(data_monthly.diff_4.tolist())
print('ADF Statistic: %f' % adf_result[0])
print('p-value: %f' % adf_result[1])
```

```
ADF Statistic: 2.803448
p-value: 1.000000
```

After resampling the dataset, 4 differences were applied to make dataset stationary but the dataset could not be stationary as we can see on P-value.

In [31]:
```python
#Applying log function to the monthly resample dataframe
data_monthly['TL_log']=np.log(data_monthly['Turkish_Lira'])
```

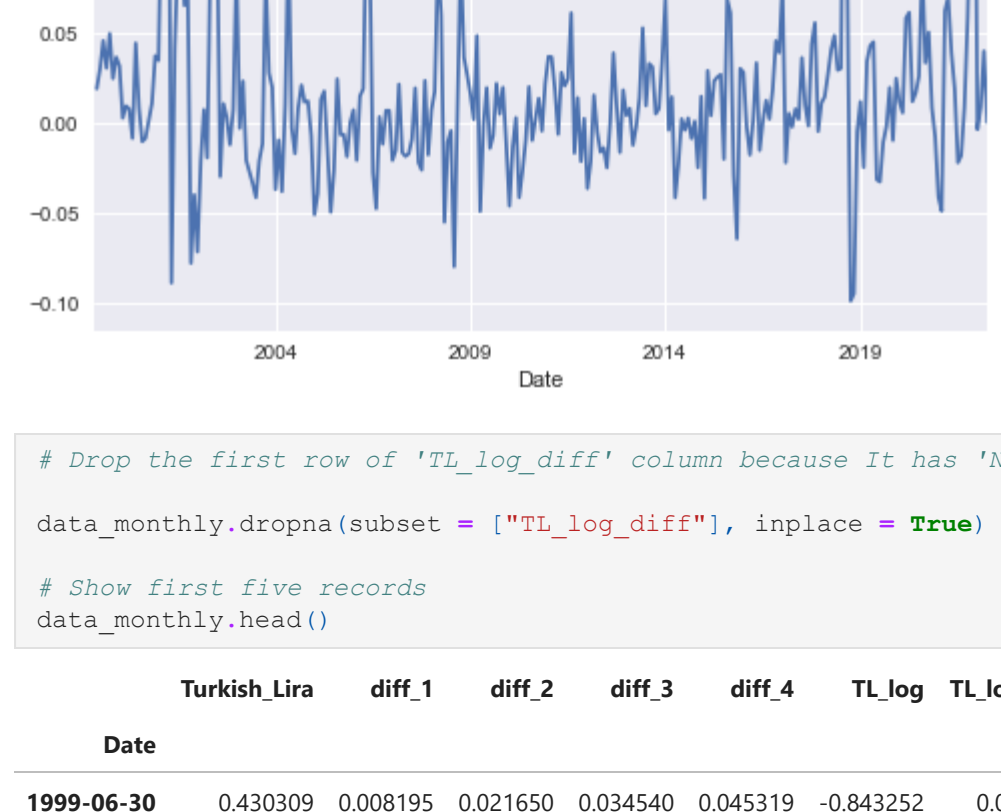
In [32]:
```python
#Printing first five variables
data_monthly.head(5)
```

Out[32]:

Date	Turkish_Lira	diff_1	diff_2	diff_3	diff_4	TL_log
1999-05-31	0.422114	0.013455	0.026345	0.037124	0.045649	-0.862479
1999-06-30	0.430309	0.008195	0.021650	0.034540	0.045319	-0.843252
1999-07-31	0.443436	0.013127	0.021322	0.034777	0.047667	-0.813201
1999-08-31	0.464241	0.020805	0.033932	0.042127	0.055582	-0.767352
1999-09-30	0.478655	0.014414	0.035218	0.048345	0.056540	-0.736776

In [33]:
```python
#Plotting Monthly resampled dataframe with 'TL_log' feature
data_monthly['TL_log'].plot()
```

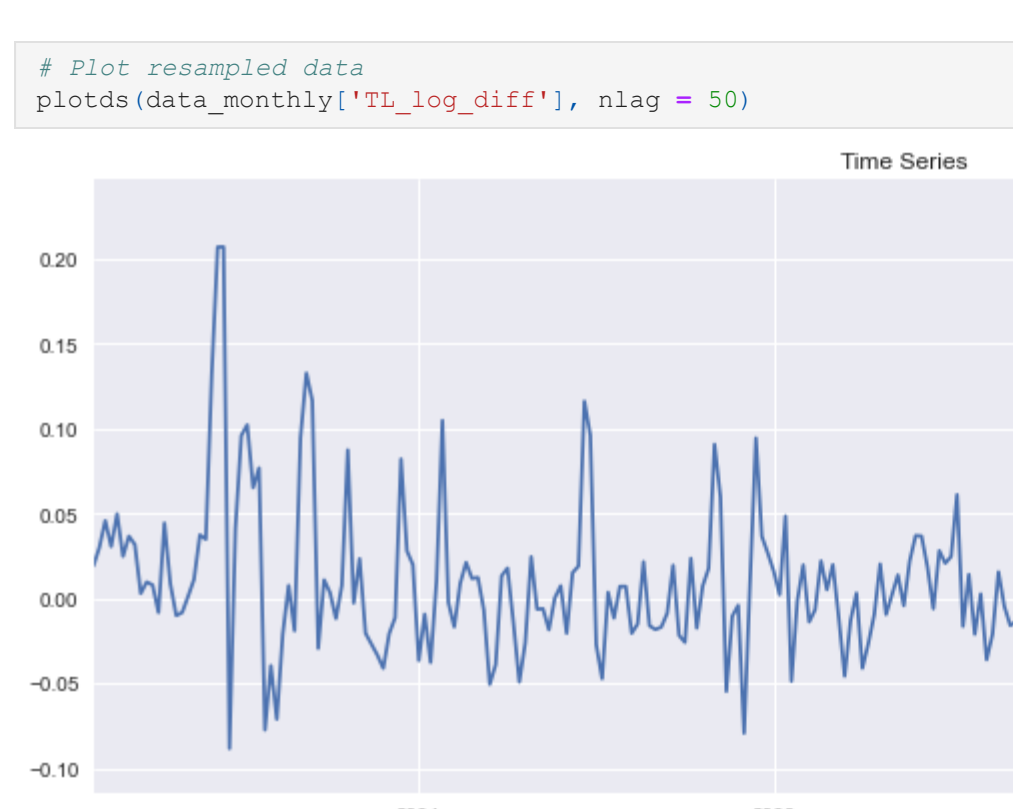
Out[33]: <AxesSubplot:xlabel='Date'>



In [34]:
```python
# Store 'TL_log_diff' with one difference value of TL_log column
data_monthly['TL_log_diff']=data_monthly['TL_log'].diff()
```

In [35]:
```python
## Plot TL_log_diff feature with Date index
data_monthly['TL_log_diff'].plot()
```

Out[35]: <AxesSubplot:xlabel='Date'>

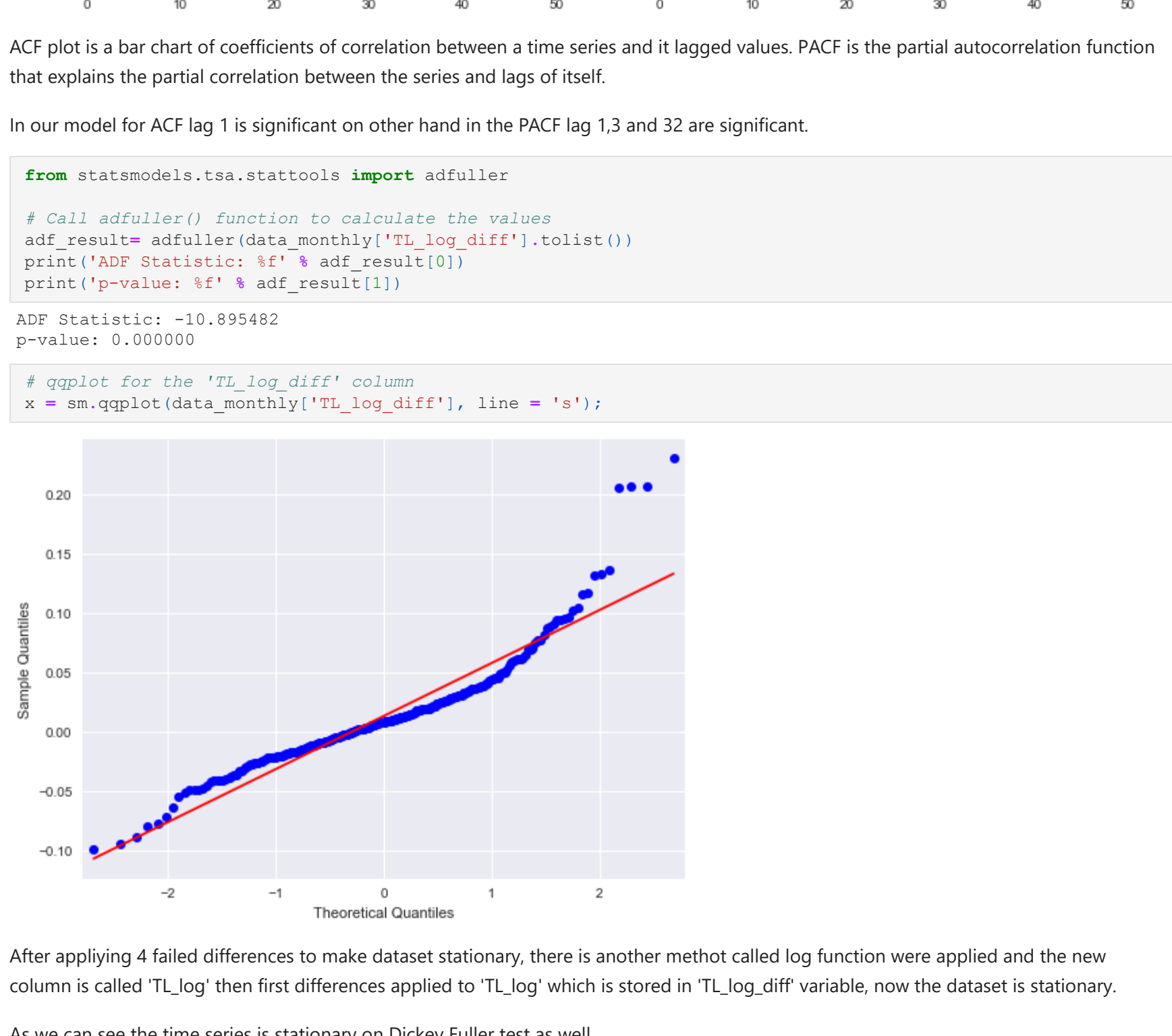


In [36]:
```python
# Drop the first row of 'TL_log_diff' column because It has 'NaN' value

data_monthly.dropna(subset = ["TL_log_diff"], inplace = True)

# Show first five records
data_monthly.head()
```

Out[36]:

Date	Turkish_Lira	diff_1	diff_2	diff_3	diff_4	TL_log	TL_log_diff
1999-06-30	0.430309	0.008195	0.021650	0.034540	0.045319	-0.843252	0.019228
1999-07-31	0.443436	0.013127	0.021322	0.034777	0.047667	-0.813201	0.030051
1999-08-31	0.464241	0.020805	0.033932	0.042127	0.055582	-0.767352	0.045849
1999-09-30	0.478655	0.014414	0.035218	0.048345	0.056540	-0.736776	0.030576
1999-10-31	0.503100	0.024445	0.038859	0.059664	0.072791	-0.686966	0.049810

In [37]:
```python
# Plot resampled data
plotds(data_monthly['TL_log_diff'], nlag = 50)
```



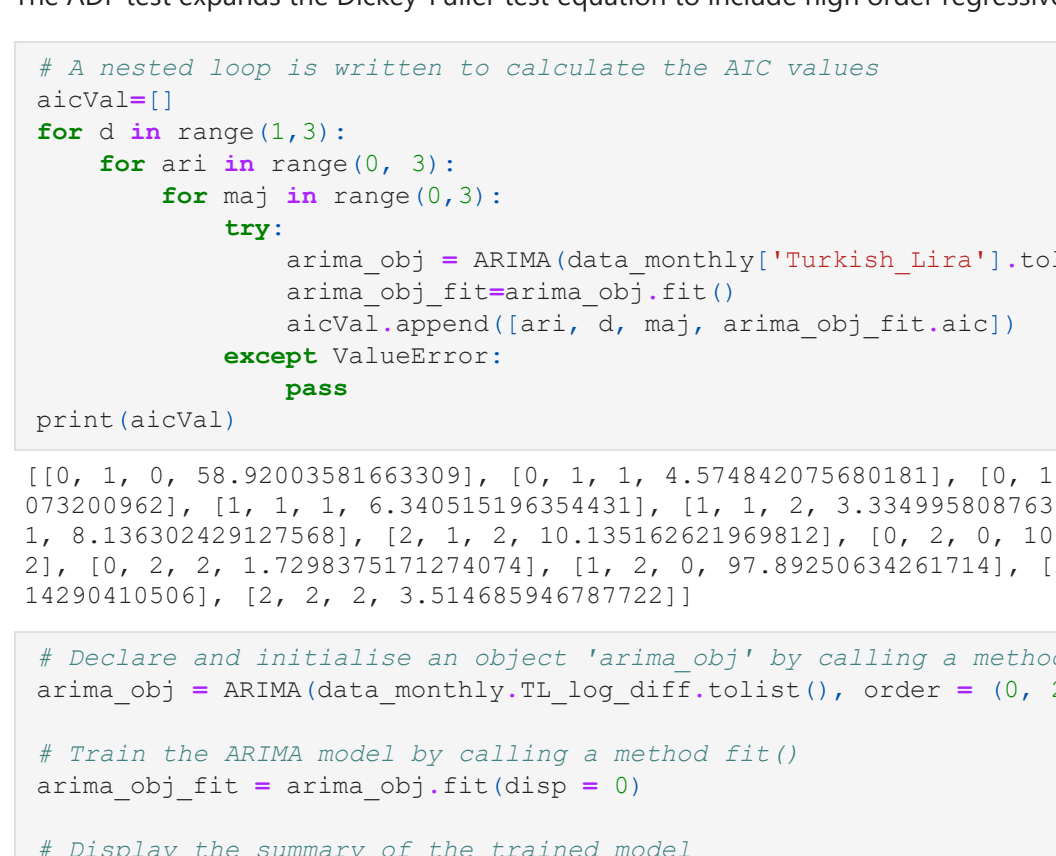
ACF plot is a bar chart of coefficients of correlation between a time series and it lagged values. PACF is the partial autocorrelation function that explains the partial correlation between the series and lags of itself.

In our model for ACF lag 1 is significant on other hand in the PACF lag 1,3 and 32 are significant.

In [38]:
```python
from statsmodels.tsa.stattools import adfuller

# Call adfuller() function to calculate the values
adf_result= adfuller(data_monthly['TL_log_diff'].tolist())
print('ADF Statistic: %f' % adf_result[0])
print('p-value: %f' % adf_result[1])
```

```
ADF Statistic: -10.895482
p-value: 0.000000
```

In [39]:
```python
# qqplot for the 'TL_log_diff' column
x = sm.qqplot(data_monthly['TL_log_diff'], line = 's');
```



After appliying 4 failed differences to make dataset stationary, there is another methot called log function were applied and the new column is called 'TL_log' then first differences applied to 'TL_log' which is stored in 'TL_log_diff' variable, now the dataset is stationary.

As we can see the time series is stationary on Dickey Fuller test as well.

Augmented Dickey Fuller test (ADF Test) is a common statistical test used to test whether a given Time series is stationary or not. It is one of the most commonly used statistical test when it comes to analyzing the stationary of a series.

A Dickey-Fuller test is a unit root test that tests the null hypothesis that α=1 in the model equation. alpha is the coefficient of the first lag on Y.

Null Hypothesis (H0): alpha=1

Augmented Dickey Fuller test (ADF Test) is a common statistical test used to test whether a given Time series is stationary or not.

As the name suggest, the ADF test is an 'augmented' version of the Dickey Fuller test.

The ADF test expands the Dickey-Fuller test equation to include high order regressive process in the model.

In [40]:
```python
# A nested loop is written to calculate the AIC values
aicVal=[]
for d in range(1,3):
    for ari in range(0, 3):
        for maj in range(0,3):
            try:
                arima_obj = ARIMA(data_monthly['Turkish_Lira'].tolist(), order=(ari,d,maj))
                arima_obj_fit=arima_obj.fit()
                aicVal.append([ari, d, maj, arima_obj_fit.aic])
            except ValueError:
                pass
print(aicVal)
```

```
[[0, 1, 0, 58.92003581663309], [0, 1, 1, 4.574842075680181], [0, 1, 2, 6.3667764191441165], [1, 1, 0, 11.873448073200962], [1, 1, 1, 6.340515196354431], [1, 1, 2, 3.3349958087635514], [2, 1, 0, 8.338545246190051], [2, 1, 1, 8.136302429127568], [2, 1, 2, 10.135162621969812], [0, 2, 0, 106.94089464664395], [0, 2, 1, 40.49588174481892], [0, 2, 2, 1.7298375171274074], [1, 2, 0, 97.89250634261714], [2, 2, 0, 67.69595016077756], [2, 2, 1, 3.478214290410506], [2, 2, 2, 3.514685946787722]]
```

In [41]:
```python
# Declare and initialise an object 'arima_obj' by calling a method 'ARIMA()'
arima_obj = ARIMA(data_monthly.TL_log_diff.tolist(), order = (0, 2, 2))

# Train the ARIMA model by calling a method fit()
arima_obj_fit = arima_obj.fit(disp = 0)

# Display the summary of the trained model
arima_obj_fit.summary()
```

Out[5]:

	Member_number	Date	itemDescription
0	1808	21-07-2015	tropical fruit
1	2552	05-01-2015	whole milk
2	2300	19-09-2015	pip fruit
3	1187	12-12-2015	other vegetables
4	3037	01-02-2015	whole milk

In [6]:
```
#display shape of datset
Grocery_df.shape
```

Out[6]:
```
(38765, 3)
```

In [7]:
```
Grocery_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 38765 entries, 0 to 38764
Data columns (total 3 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Member_number    38765 non-null  int64
 1   Date             38765 non-null  object
 2   itemDescription  38765 non-null  object
dtypes: int64(1), object(2)
memory usage: 908.7+ KB
```

In [8]:
```
Grocery_df.isnull().sum()
```

Out[8]:
```
Member_number      0
Date               0
itemDescription    0
dtype: int64
```

In [9]:
```
Grocery_df['itemDescription'].nunique()
```

Out[9]:
```
167
```

In [10]:
```
Grocery_df['Member_number'].nunique()
```

Out[10]:
```
3898
```

In [11]:
```
# Remove any whote spaces or specified characters at the start and end of a string
Grocery_df['itemDescription']=Grocery_df['itemDescription'].str.strip()

# astype is a Pandas function for DataFrames (and numpy for numpy arrays) that will cast the object to the spec
Grocery_df['itemDescription']=Grocery_df['itemDescription'].astype('str')
```

In [12]:
```
#Create and declare a method named as 'encode_units()
def encode_units(x):
    if x <= 0:
        return 0
    if x >= 1:
        return 1
```

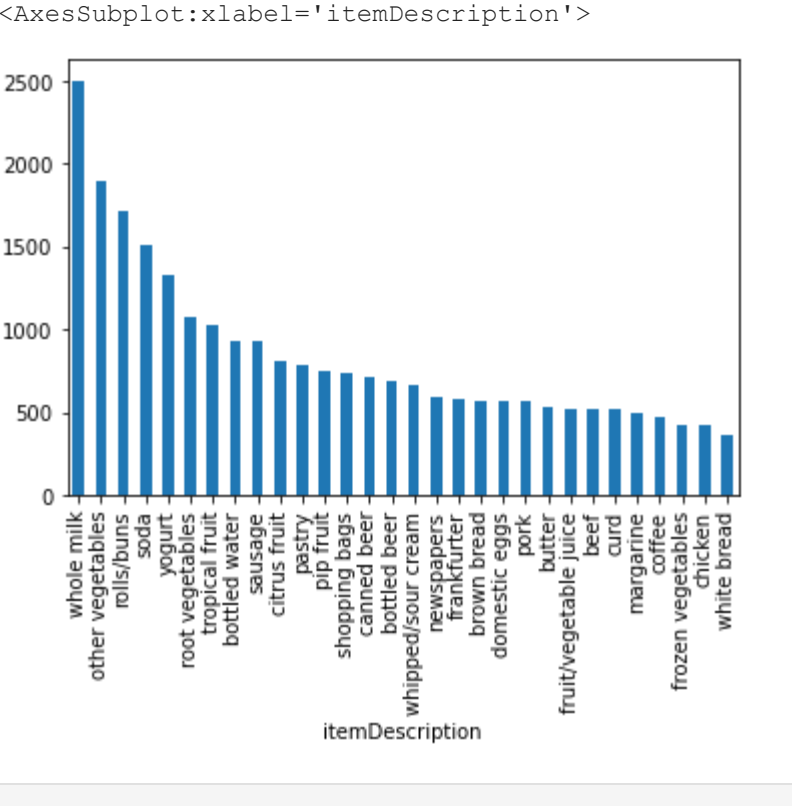
In [13]:
```
Grocery_df.head()
```

Out[13]:

	Member_number	Date	itemDescription
0	1808	21-07-2015	tropical fruit
1	2552	05-01-2015	whole milk
2	2300	19-09-2015	pip fruit
3	1187	12-12-2015	other vegetables
4	3037	01-02-2015	whole milk

In [37]:
```
#to show the number of items sold in barchart
Grocery_df.value_counts(ascending=False,subset='itemDescription').head(30).plot(kind='bar')
```

Out[37]:
```
<AxesSubplot:xlabel='itemDescription'>
```



In [ ]:

In [14]:
```
# store items within a new dataframe with groupby function
basket=Grocery_df.groupby(['Member_number','itemDescription'])['itemDescription'].count().unstack().reset_index
```

In [15]:
```
basket.head()
```

Out[15]:

itemDescription	Instant food products	UHT-milk	abrasive cleaner	artif. sweetener	baby cosmetics	bags	baking powder	bathroom cleaner	beef	berries	...	turkey	vinegar	waffles	whip
Member_number															
1000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
1001	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	...	0.0	0.0	0.0	
1002	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
1003	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
1004	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	

5 rows × 167 columns

In [17]:
```
#applying the function
basket_set = basket.applymap(encode_units)
```

In [18]:
```
basket_set.head()
```

Out[18]:

itemDescription	Instant food products	UHT-milk	abrasive cleaner	artif. sweetener	baby cosmetics	bags	baking powder	bathroom cleaner	beef	berries	...	turkey	vinegar	waffles	whip
Member_number															
1000	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
1001	0	0	0	0	0	0	0	0	1	0	...	0	0	0	
1002	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
1003	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
1004	0	0	0	0	0	0	0	0	0	0	...	0	0	0	

5 rows × 167 columns

In [ ]:
```
We applied the one hot encoding to dataset
```

In [19]:
```
#basket=Grocery_df.groupby(['itemDescription'])
```

Apriori Rule for Market Basket Analysis

In [56]:
```
# Evaluate the frequent item rules for apriori function
frequent_items_apr = apriori(basket_set, min_support = 0.05, use_colnames = True)
```

In [57]:
```
## Display the reles due to apriori algorithm
rules=association_rules(frequent_items_apr,metric='lift', min_threshold=1)
rules=rules.sort_values(['confidence','lift'],ascending=[False,False])
print(rules)
```

```
                         antecedents                        consequents  \
194  (bottled water, other vegetables)                       (whole milk)
236        (yogurt, other vegetables)                       (whole milk)
247              (rolls/buns, yogurt)                       (whole milk)
206    (rolls/buns, other vegetables)                       (whole milk)
254                   (yogurt, soda)                       (whole milk)
..                               ...                                ...
58                      (whole milk)                           (coffee)
255                     (whole milk)                     (yogurt, soda)
232                     (whole milk)  (tropical fruit, other vegetables)
219                     (whole milk)         (sausage, other vegetables)
46                      (whole milk)                          (chicken)

     antecedent support  consequent support   support  confidence      lift  \
194            0.093894            0.458184  0.056183    0.598361  1.305941
236            0.120318            0.458184  0.071832    0.597015  1.303003
247            0.111339            0.458184  0.065931    0.592166  1.292420
206            0.146742            0.458184  0.082093    0.559441  1.220996
254            0.097486            0.458184  0.054387    0.557895  1.217622
..                  ...                 ...       ...         ...       ...
58             0.458184            0.114931  0.055156    0.120381  1.047420
255            0.458184            0.097486  0.054387    0.118701  1.217622
232            0.458184            0.091329  0.050539    0.110302  1.207749
219            0.458184            0.092868  0.050282    0.109742  1.181702
46             0.458184            0.100564  0.050026    0.109183  1.085698

     leverage  conviction
194  0.013162    1.349012
236  0.016704    1.344507
247  0.014917    1.328521
206  0.014859    1.229837
254  0.009720    1.225537
..        ...         ...
58   0.002497    1.006196
255  0.009720    1.024073
232  0.008693    1.021326
219  0.007732    1.018954
46   0.003949    1.009674

[258 rows x 9 columns]
```

In Apriori algorithm (whole milk) goes with bottled water, other vegetables,yogurt, other vegetables and soda so we can see this from the Support, confidence(above 50%) and lift values(the value is bigger than 1 which is showing that some dependency exists between the items)

In [52]:
```
rules['lift'].max(),rules['confidence'].max()
```

Out[52]:
```
(2.4286889871155837, 0.7843137254901961)
```

In [53]:
```
rules['lift'].min(),rules['confidence'].min()
```

Out[53]:
```
(1.0001439344159704, 0.021836506159014557)
```

In [43]:
```
print('other vegetables',basket['other vegetables'].sum())
print('bottled water',basket['bottled water'].sum())
print('whole milk',basket['whole milk'].sum())
print('yogurt',basket['yogurt'].sum())
```

```
other vegetables 1898.0
bottled water 933.0
whole milk 2502.0
yogurt 1334.0
```

Also these figures showing the popularity these items which can be indicate to put them together as they are popular

FP-growth algorithm for Market Basket Analysis

In [26]:
```
## Evaluate the frequent item rules for fpgrowth function
frequent_items_fb=fpgrowth(basket_set,min_support = 0.01, use_colnames = True)
```

In [35]:
```
## Display the reles due to fp-growth algorithm
rules_fb=association_rules(frequent_items_fb, metric='confidence',min_threshold=0.2,)
rules_fb=rules_fb.sort_values(['confidence','lift'],ascending=[False,False])
```

In [45]:
```
print(rules_fb)
#frequent_items.sort_values("support", ascending=False).head()
```

```
                                          antecedents       consequents  \
5117                             (domestic eggs, meat)      (whole milk)
3021                 (chocolate, fruit/vegetable juice)      (whole milk)
3273  (yogurt, other vegetables, rolls/buns, bottled...      (whole milk)
2541                  (pip fruit, bottled water, yogurt)      (whole milk)
4798                  (brown bread, rolls/buns, yogurt)      (whole milk)
...                                                ...               ...
4743                                           (candy)  (shopping bags)
5415                          (long life bakery product)      (pip fruit)
2414                                       (pip fruit)   (citrus fruit)
3183                     (margarine, other vegetables)   (citrus fruit)
950                                 (misc. beverages)  (bottled water)

      antecedent support  consequent support   support  confidence      lift  \
5117            0.013084            0.458184  0.010262    0.784314  1.711789
3021            0.014366            0.458184  0.010775    0.750000  1.636898
3273            0.014110            0.458184  0.010518    0.745455  1.626978
2541            0.013853            0.458184  0.010262    0.740741  1.616689
4798            0.017445            0.458184  0.012827    0.735294  1.604802
...                  ...                 ...       ...         ...       ...
4743            0.053874            0.168291  0.010775    0.200000  1.188415
5415            0.065418            0.170600  0.013084    0.200000  1.172331
2414            0.170600            0.185480  0.034120    0.200000  1.078285
3183            0.050026            0.185480  0.010005    0.200000  1.078285
950             0.059005            0.213699  0.011801    0.200000  0.935894

      leverage  conviction
5117  0.004267    2.512057
3021  0.004192    2.167265
3273  0.004053    2.128564
2541  0.003914    2.089863
4798  0.004834    2.046862
...        ...         ...
4743  0.001708    1.039636
5415  0.001923    1.036750
2414  0.002477    1.018150
3183  0.000726    1.018150
950  -0.000808    0.982876

[5844 rows x 9 columns]
```

In the FP-growth algorithm it seems whole milk is being bought with domestic eggs, meat,chocolate and fruit/vegetable juice.

Also nn the FP-growth algorithm,antecedents are different such as domestic eggs, meat, chocolate etc and their confidences(above 70%) and lift are higher then Apriori algorithm items confidence(around 50%) and lift.

Referances:

Prabhakaran, S. (2019). Augmented Dickey Fuller Test (ADF Test) – Must Read Guide. [online] Machine Learning Plus. Available at: https://www.machinelearningplus.com/time-series/augmented-dickey-fuller-test/.

Zajic, A. (2019). Introduction to AIC — Akaike Information Criterion. [online] Medium. Available at: https://towardsdatascience.com/introduction-to-aic-akaike-information-criterion-9c9ba1c96ced.

Rocca, B. (2019). Introduction to recommender systems. [online] Medium. Available at: https://towardsdatascience.com/introduction-to-recommender-systems-6c66cf15ada#:~:text=The%20purpose%20of%20a%20recommender.

In [ ]: