

Question 1: Time Series Analysis

In [1]:

```
# Import the impor libraries
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import matplotlib
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]:

```
# Set plot style
plt.style.use('seaborn')

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

In [3]:

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

In [4]:

```
#See first 5 variables
df.head()
```

Out[4]:

	PeriodUnit:	[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	1.4481	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 x 14 columns

In [5]:

```
# Print the DataFrame Info
df.info()
```

Out[5]:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6022 entries, 0 to 6021
Data columns (total 14 columns):
 #   Column                Non-Null Count  Dtype  ---
 0   PeriodUnit:           6022 non-null   object
 1   [Australian dollar]    6022 non-null   object
 2   [Bulgarian lev]       6022 non-null   object
 3   [Brazilian real]       6022 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 kron]        3130 non-null   object
11   [UK pound sterling]    6022 non-null   object
12   [Greek drachma]        6022 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]         3804 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]         5994 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]        3414 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]:

```
#Seeing shape of dataframe
df.shape
```

Out[6]:

```
(6022, 14)
```

In [7]:

```
#Checking if any null values the DataFrame has
df.isnull().sum()
```

Out[7]:

```
PeriodUnit:           0
[Australian dollar]    0
[Bulgarian lev]       0
[Brazilian real]       0
[Canadian dollar]      0
[Swiss franc]          0
[Chinese yuan renminbi] 268
[Cypriot pound]        3676
[Czech koruna]         0
[Danish krone]         0
[Estonian kron]        2892
[UK pound sterling]    0
[Greek drachma]        3502
[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]         876
[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 plenty of null values but the Turkish Lira column only has 62 which makes the time series less interrupted

In [8]:

```
#Renaming the columns
df.rename(columns={'[Turkish lira]': 'Turkish_Lira',
                  '[Cypriot pound]': 'Date',
                  'PeriodUnit': 'Date',
                  inplace=True})

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

#Sort values from earlier to later
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]
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 x 14 columns

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

In [9]:

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

In [10]:

```
# Attach your own 'Date' 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]:

	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]:

```
# 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]:

```
#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]:

```
# plots is method to plot time series, ACF and PACF
def plotds(xt, nlag = 30, fig_size = (12, 10)):
    # Create instance of 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))

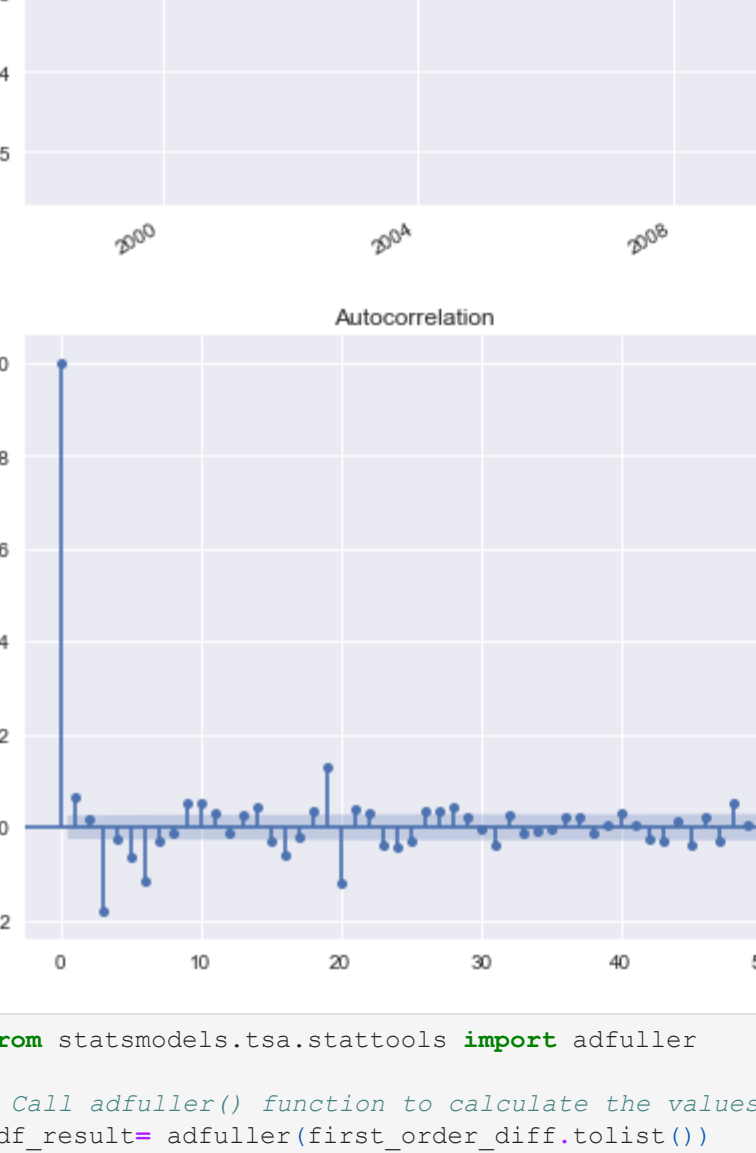
    xt.plot(ax = ax_xt)
    ax_xt.set_title('Time Series')
    plot_acf(xt, lags = 50, ax = ax_acf)
    plot_pacf(xt, lags = 50, ax = ax_pacf)
    plt.tight_layout()

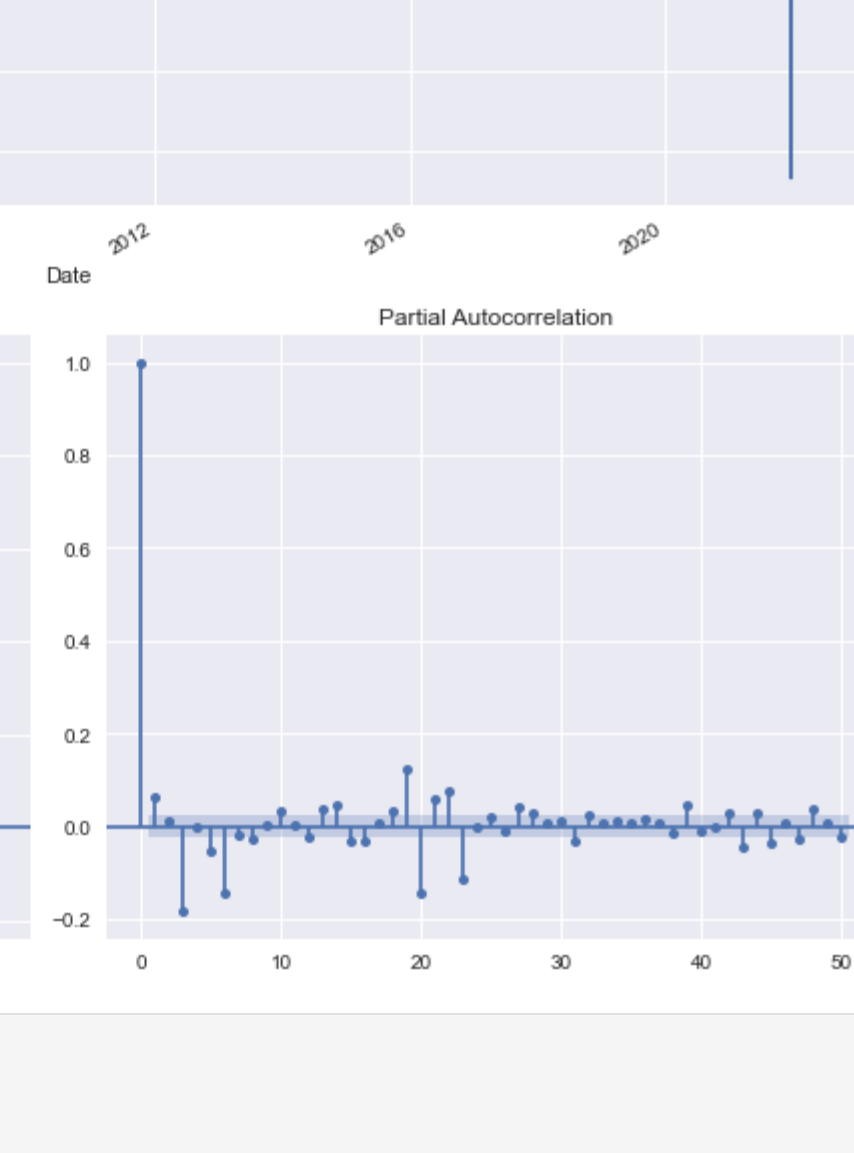
    return None
```

In [14]:

```
## Display plot of Turkish Lira column and Date index with ACF and PACF
plots(EU_TL['Turkish_Lira'], nlag = 50)
```







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

In [15]:

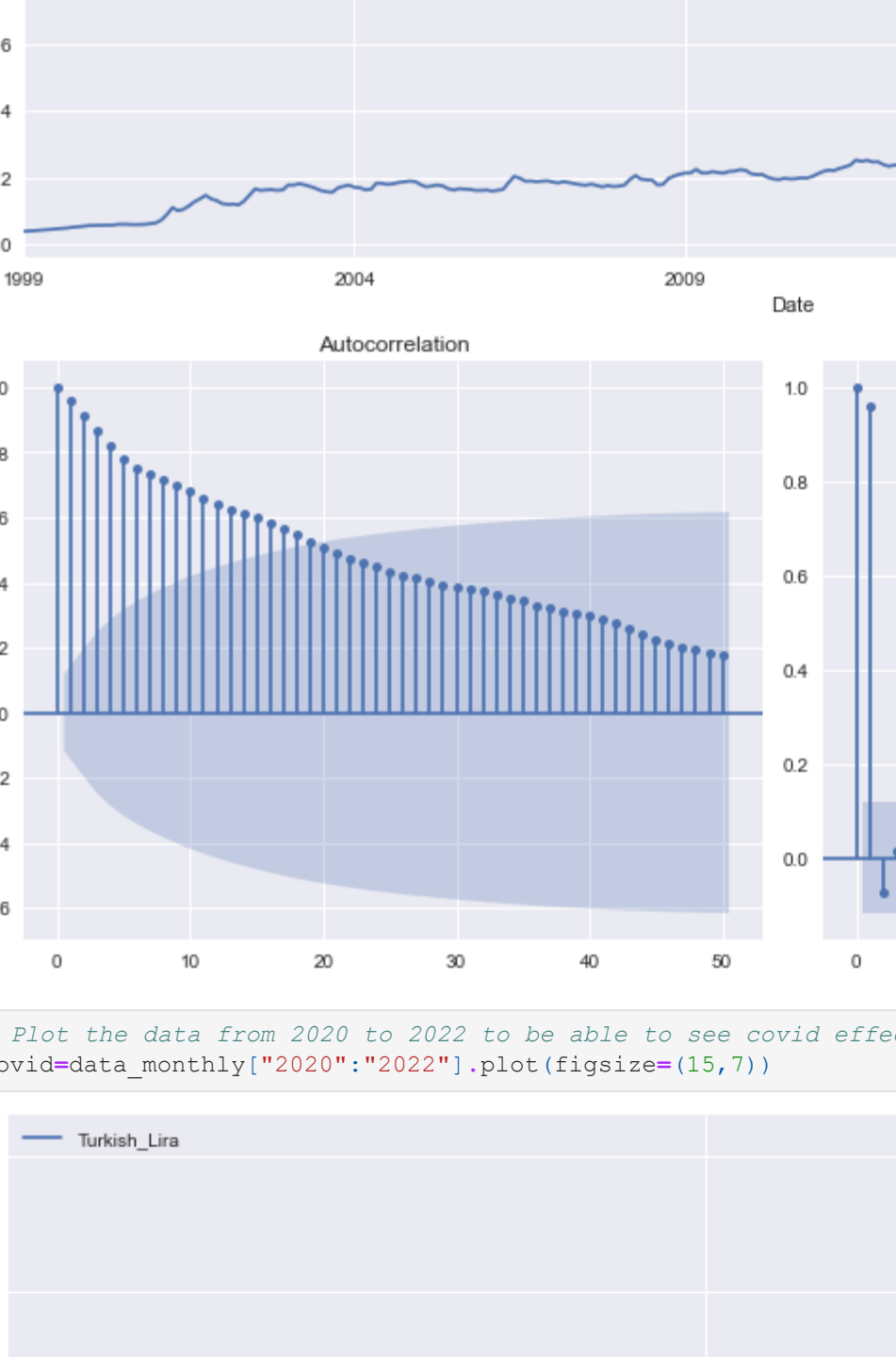
```
from statsmodels.tsa.stattools import adfuller

# 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]:

```
# qqplot for the 'Turkish Lira' 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 we can use to check if the dataset is stationary.

The Augmented Dickey-Fuller test:

In [17]:

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

Out[17]:

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

Out[17]:

```
Date      Turkish_Lira
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]:

```
# 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()
```

Out[18]:

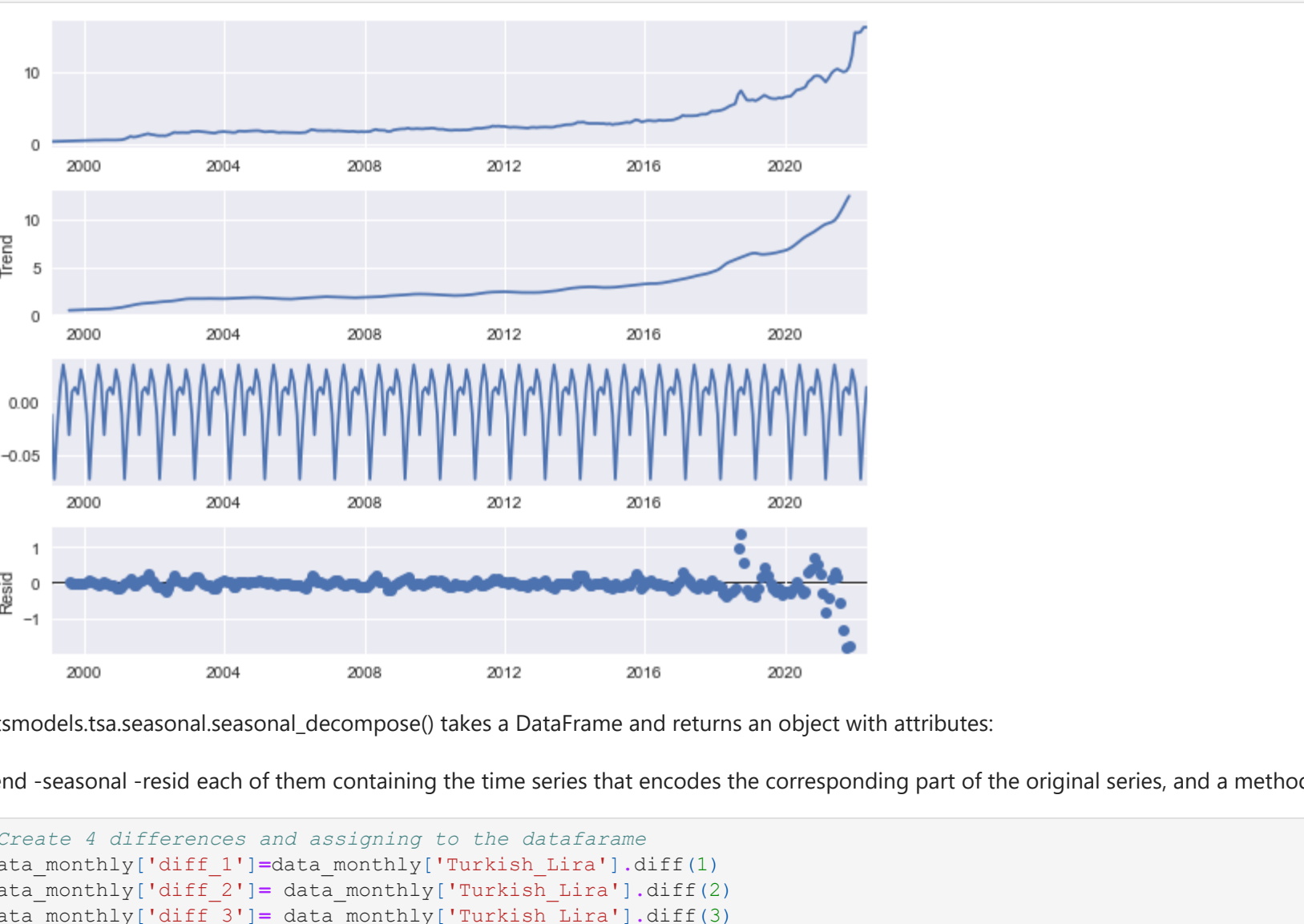
```
Display the first five records
first_order_diff.head()
```

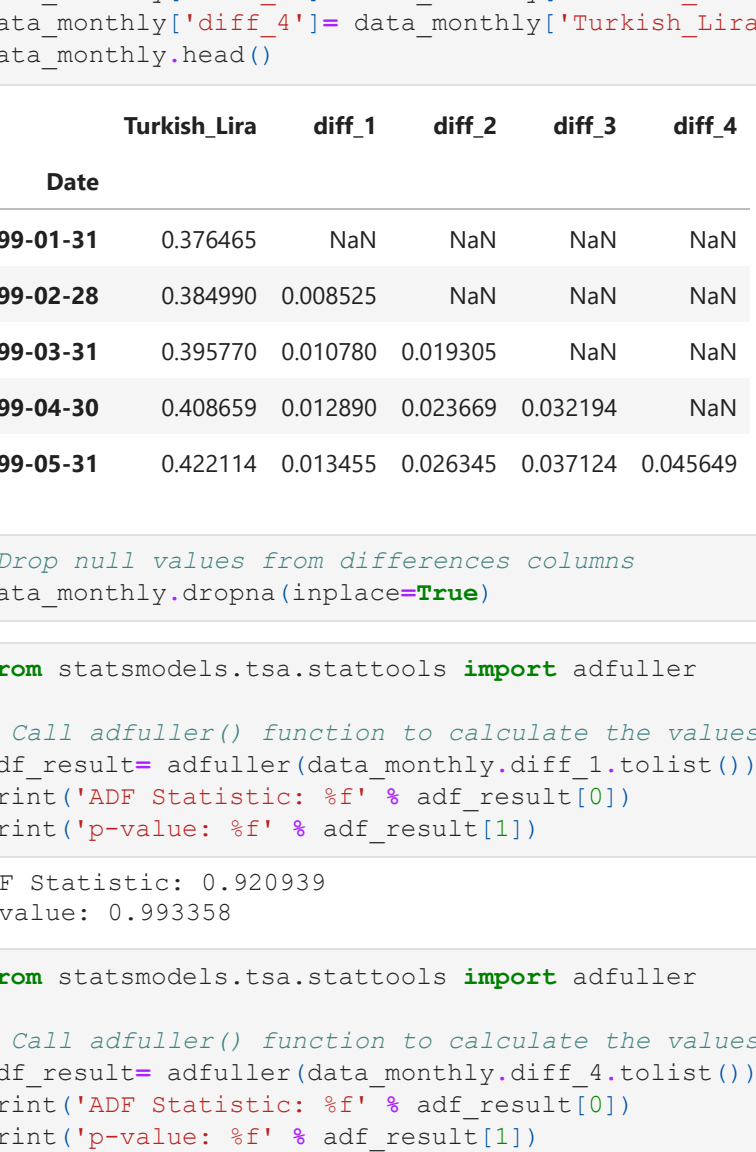
Out[18]:

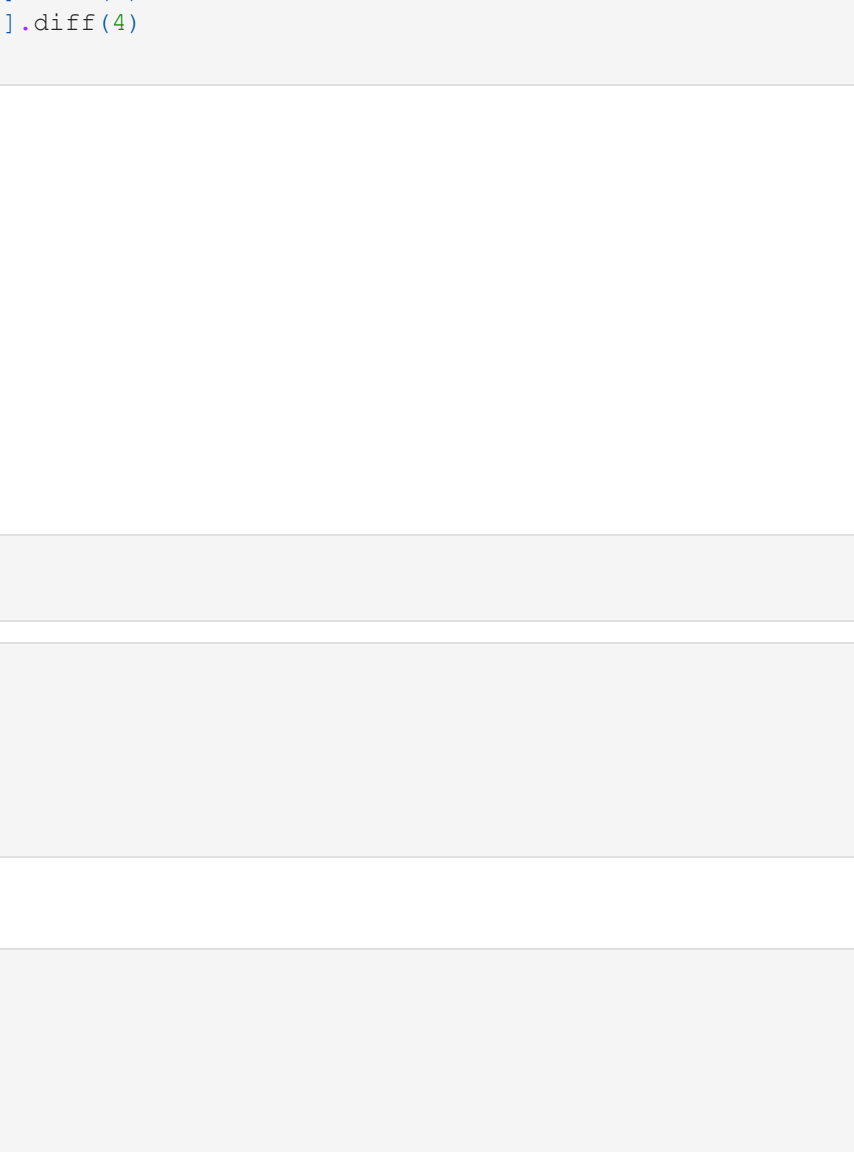
```
Date      Turkish_Lira
1999-01-05    0.0005
1999-01-06   -0.0006
1999-01-07   -0.0021
1999-01-08    0.0017
1999-01-11    0.0007
Name: Turkish_Lira, dtype: float64
```

In [19]:

```
# plot the data with 50 lags
plots(first_order_diff, nlag = 50)
```







In [20]:

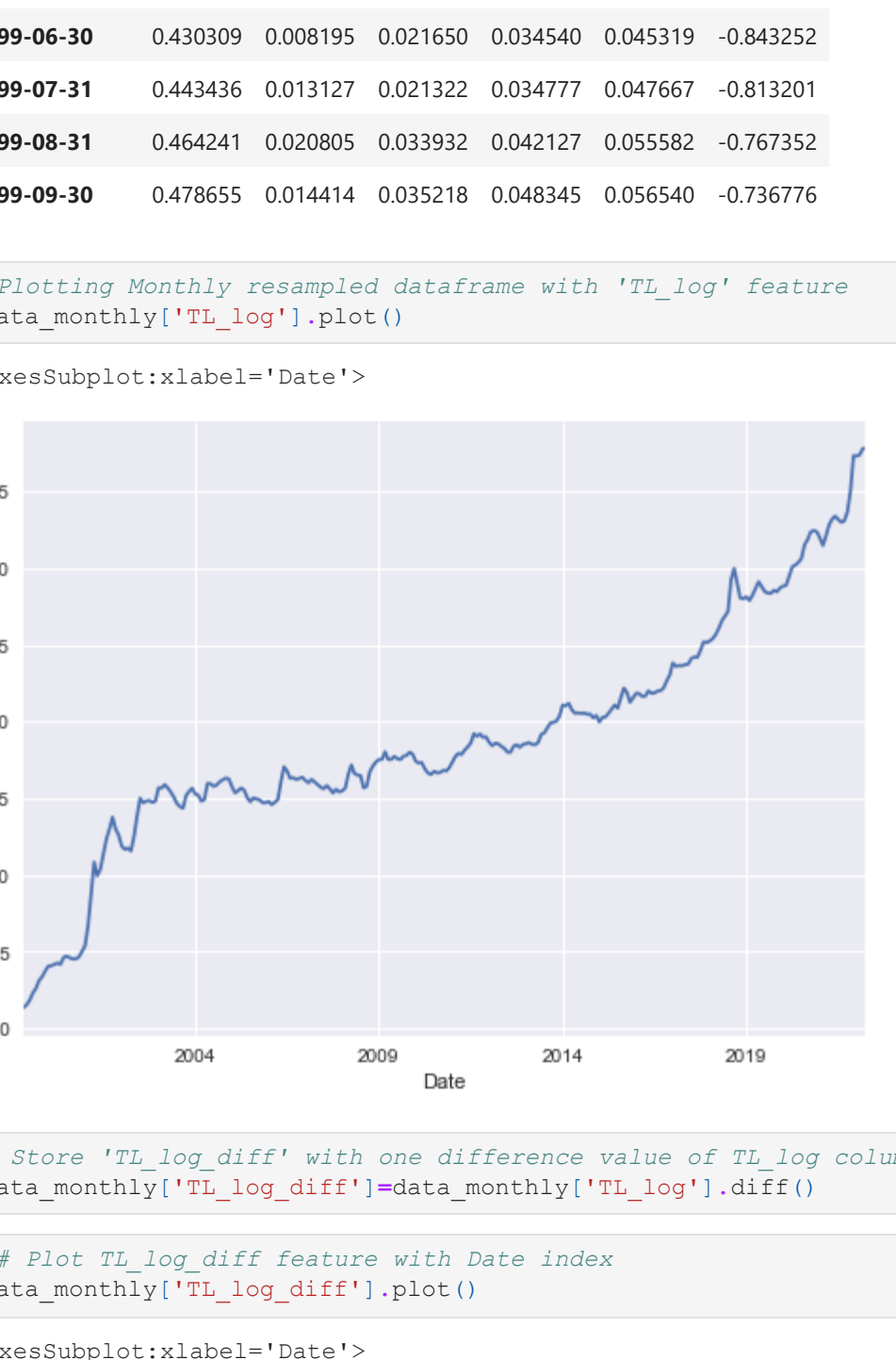
```
from statsmodels.tsa.stattools import adfuller

# 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]:

```
# qqplot for the 'Turkish Lira' column
x = sm.qqplot(first_order_diff, line = 's')
```



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

In [22]:

```
# Resample the data to the average monthly values
data_monthly = EU_TL.resample("m").mean()
```

In [23]:

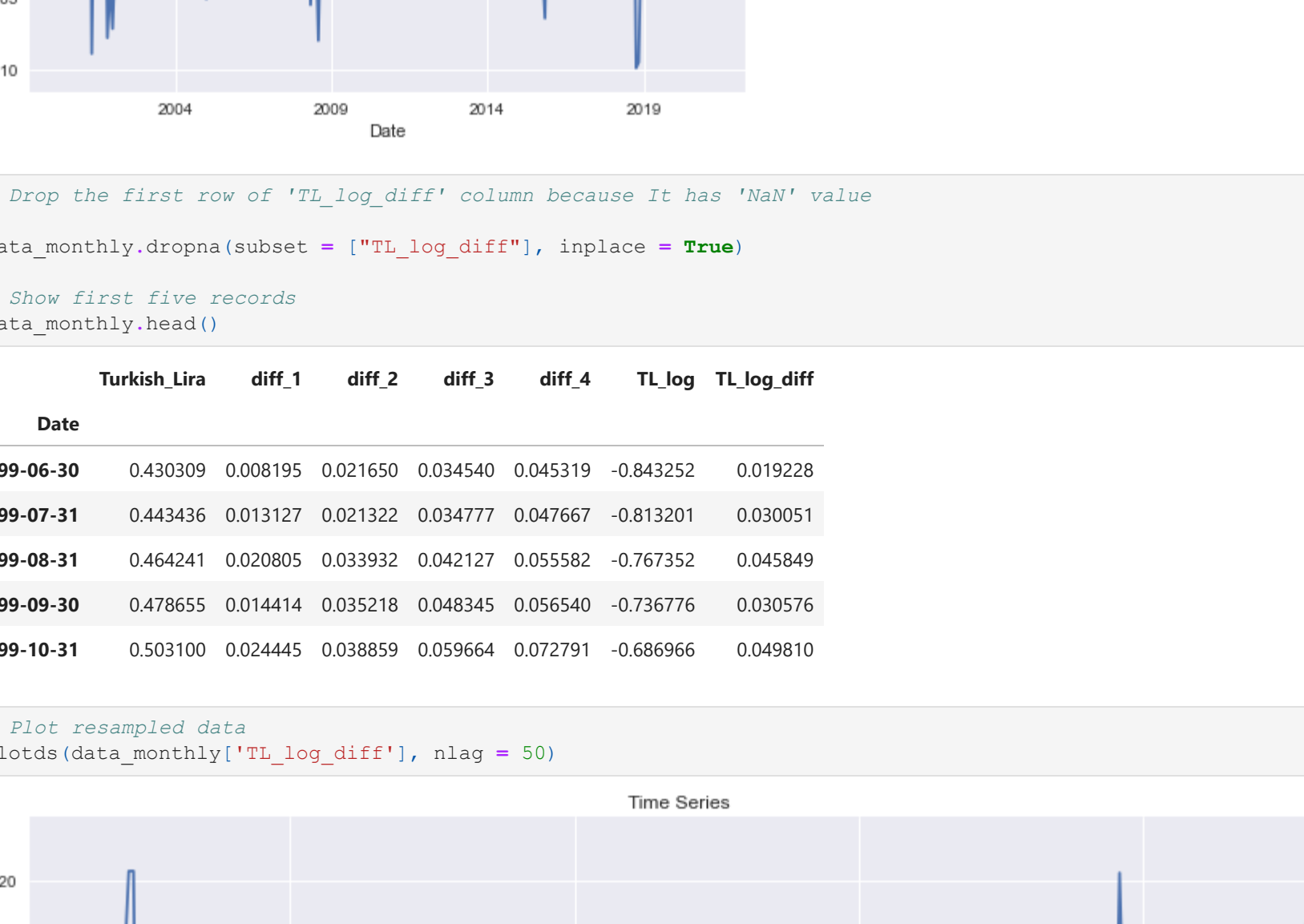
```
# Print the head of resampled data
data_monthly.head()
```

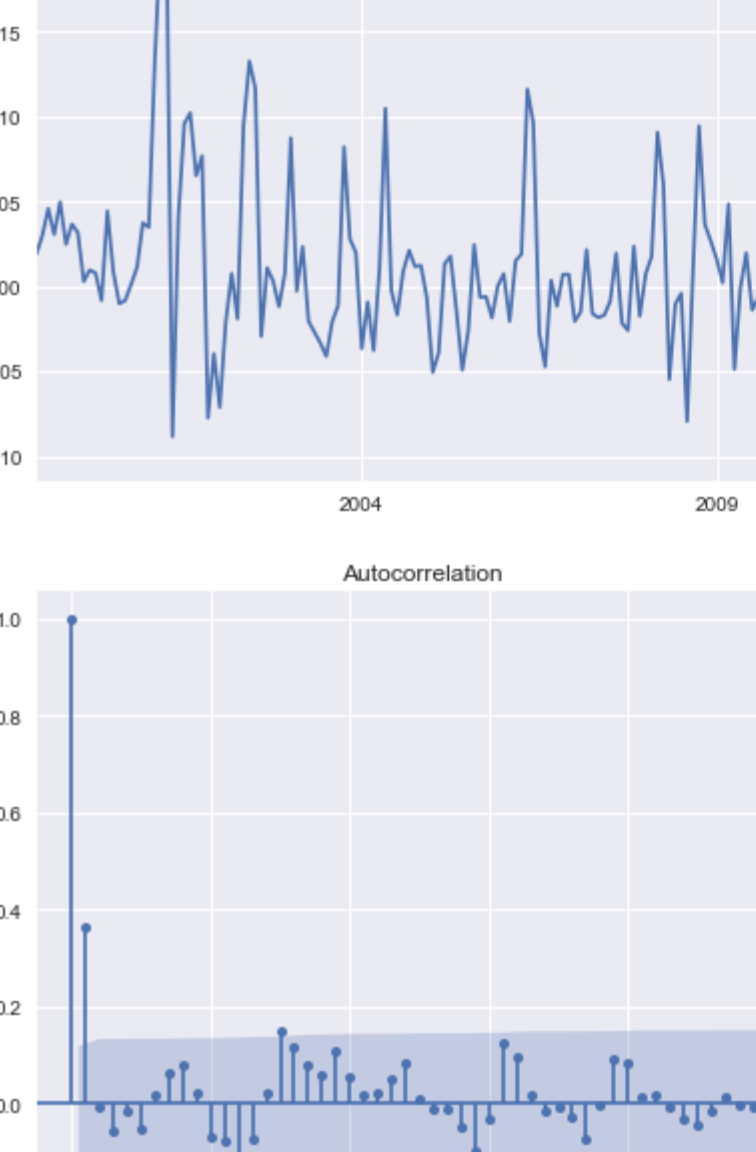
Out[23]:

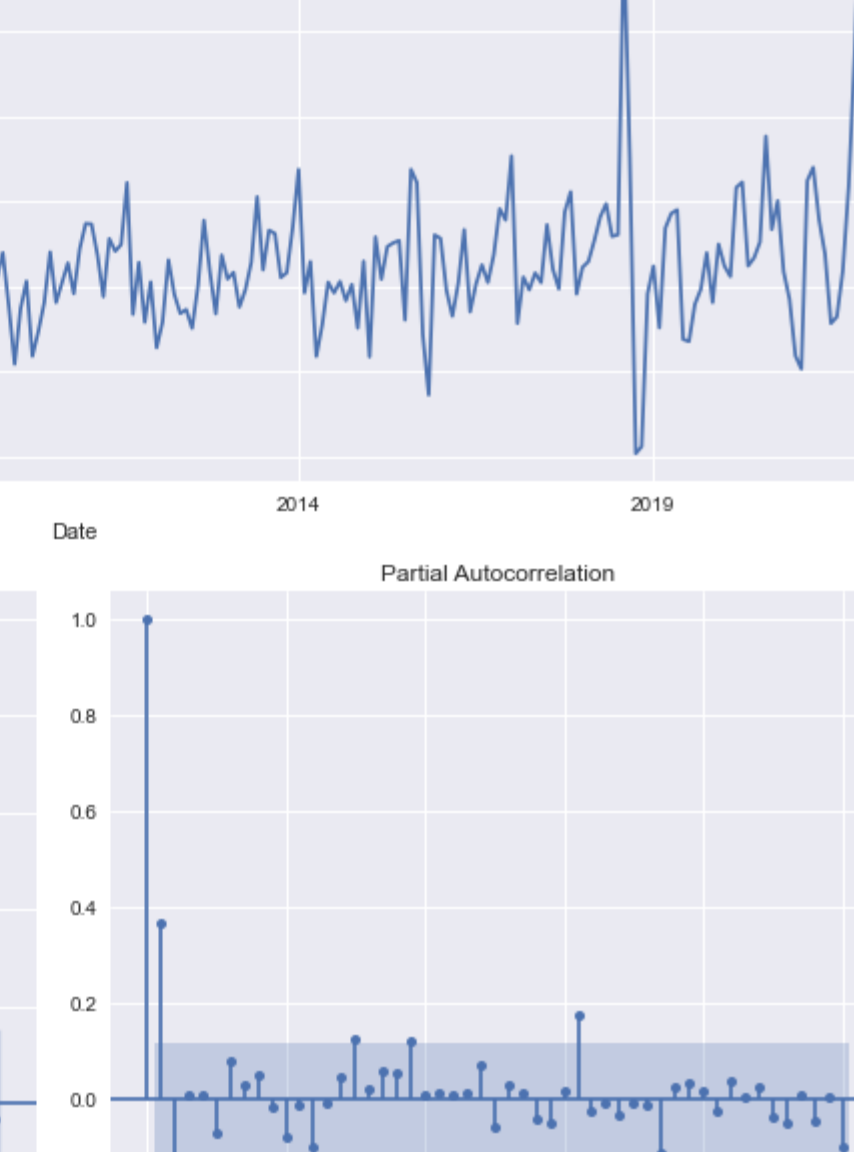
	Turkish_Lira
1999-01-31	0.374695
1999-02-28	0.384990
1999-03-31	0.395770
1999-04-30	0.408659
1999-05-31	0.422114

In [24]:

```
# Plot resampled data
plots(data_monthly['Turkish_Lira'], nlag = 50)
```

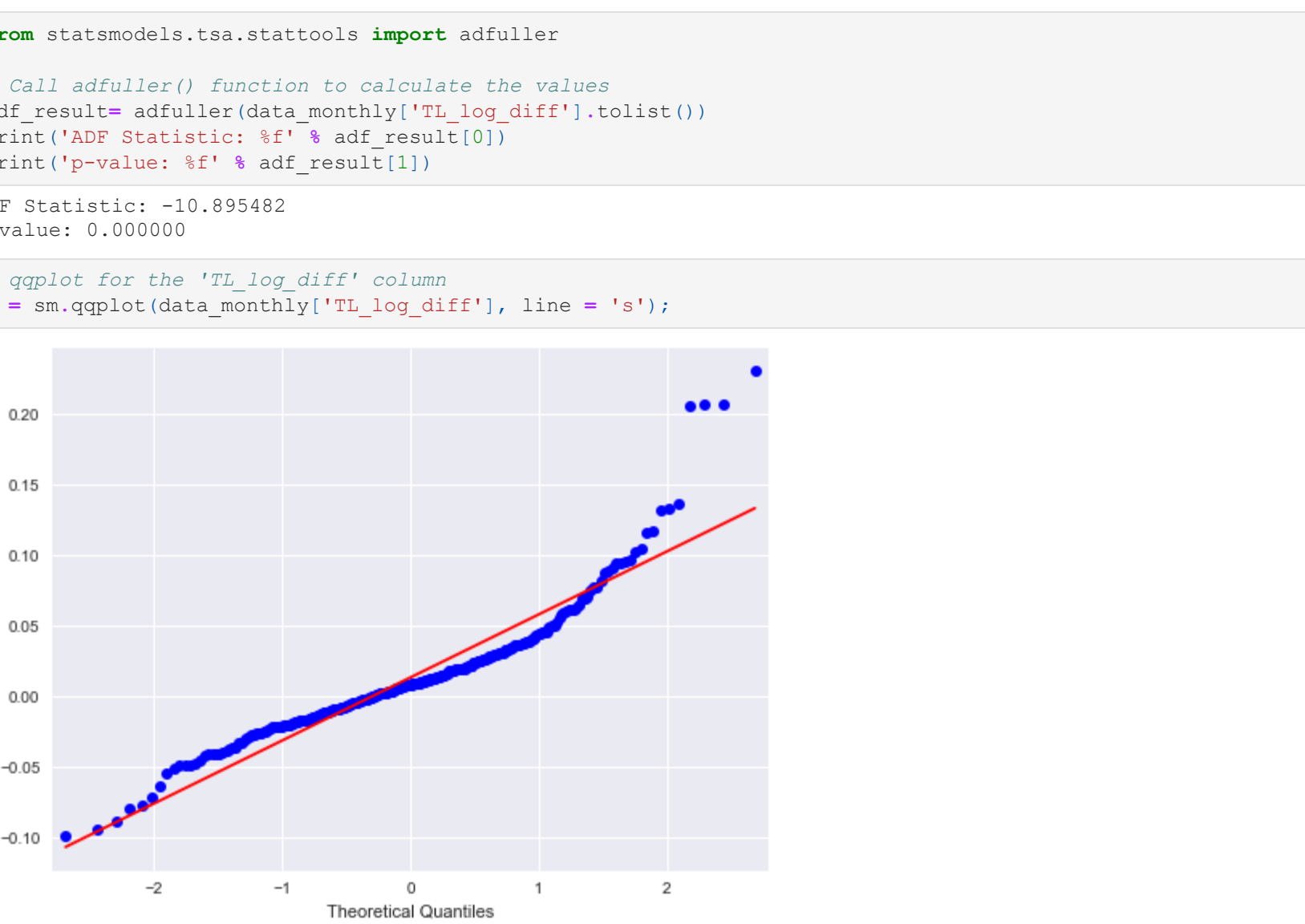






In [25]:

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

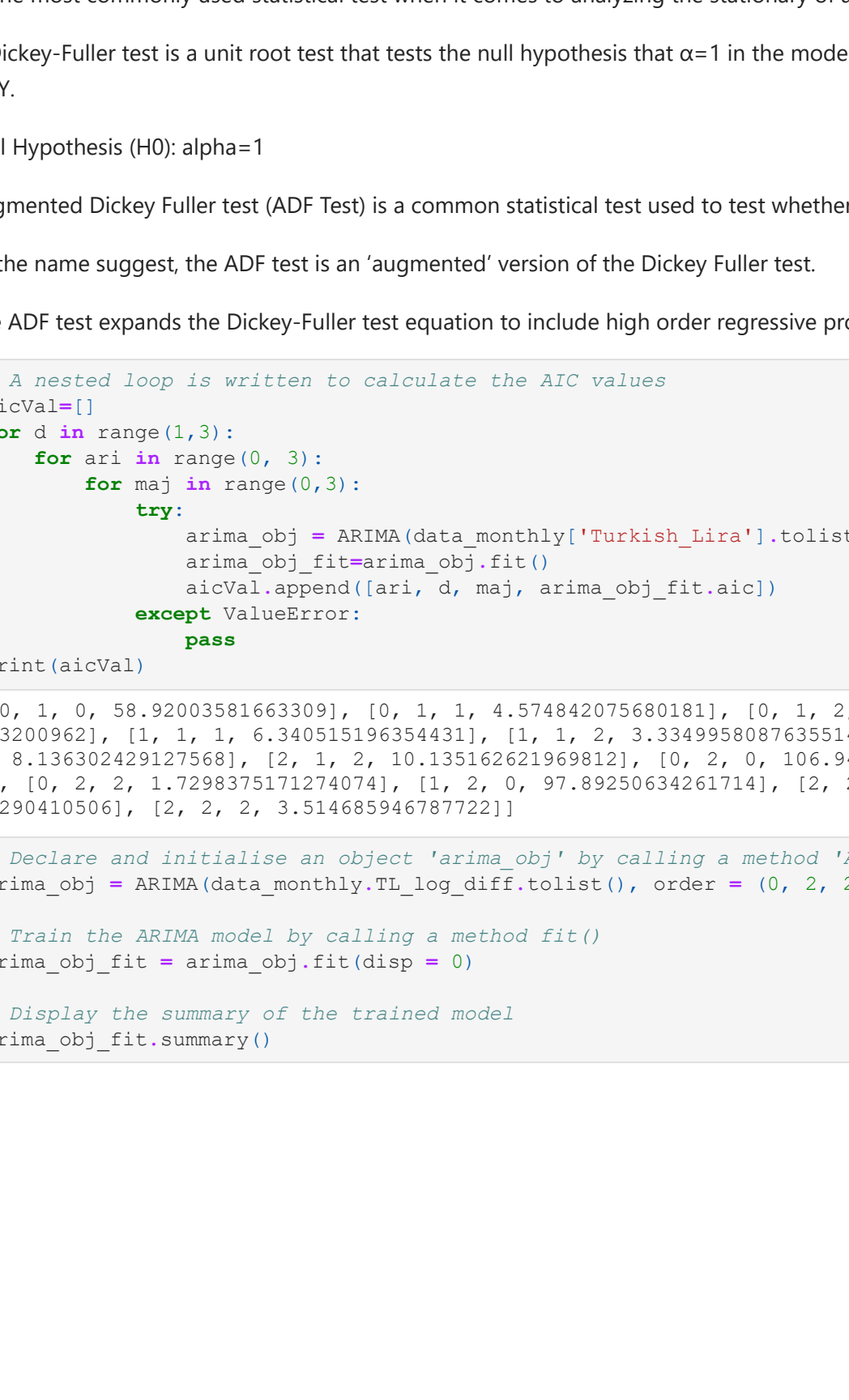


To discover effect of covid on the Euro/ Turkish Lira rate closely

In [26]:

```
# 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]:

```
# 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]:

	Turkish_Lira	diff_1	diff_2	diff_3	diff_4
1999-01-31	0.374695	NaN	NaN	NaN	NaN
1999-02-28	0.384990	0.008625	NaN	NaN	NaN
1999-03-31	0.395770	0.10780	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]:

```
#Drop null values from difference columns
data_monthly.dropna(inplace=True)
```

In [29]:

```
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: 0.920939
p-value: 0.993358
```

In [30]:

```
from statsmodels.tsa.stattools import adfuller

# Call adfuller() function to calculate the values
adf_result= adfuller(data_monthly['TL_log_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]:

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

In [32]:

```
#Printing first five variables
data_monthly.head(5)
```

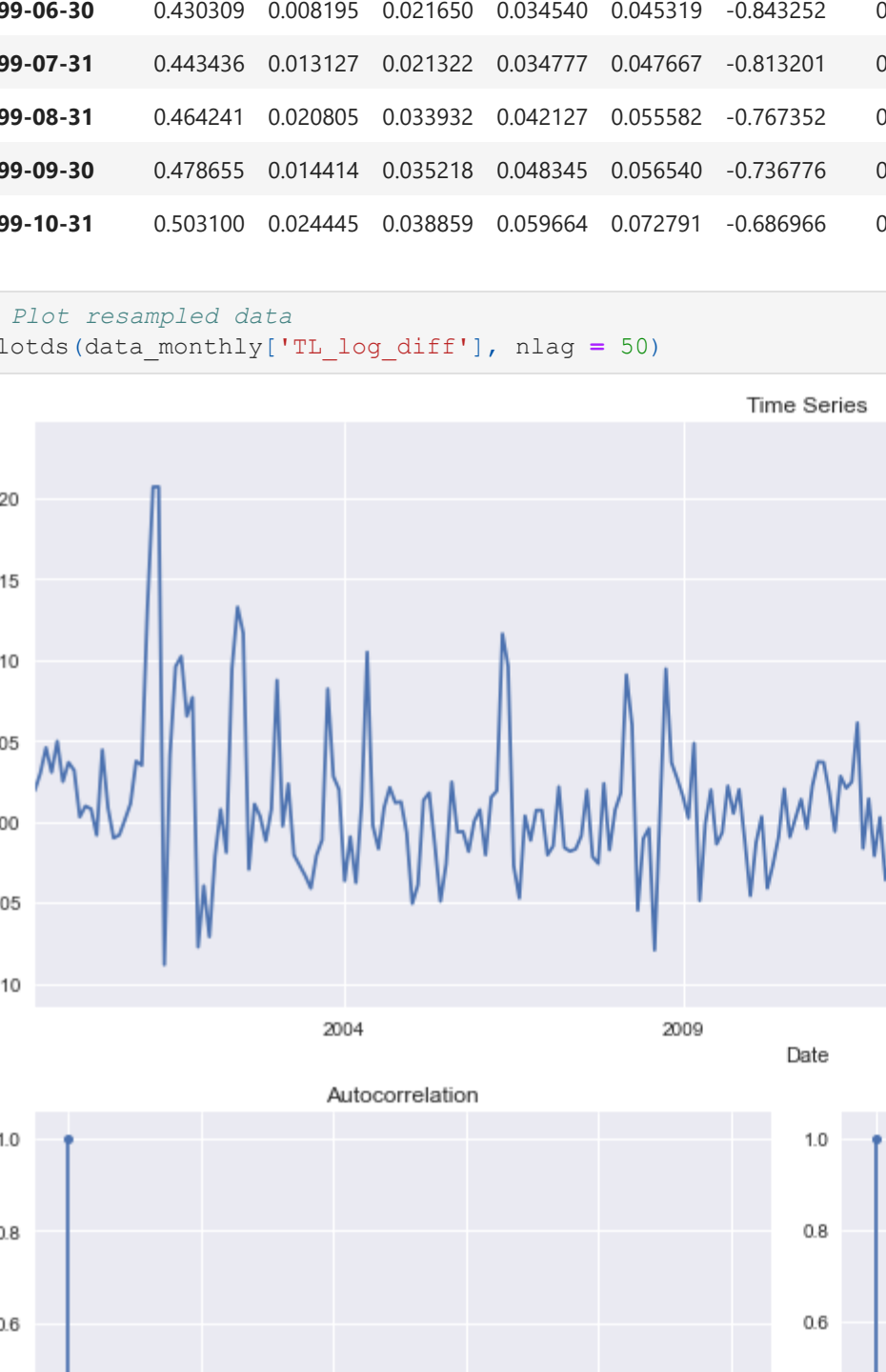
Out[32]:

	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	-0.842552
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	0.065540	-0.736776

In [33]:

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

Out[33]:



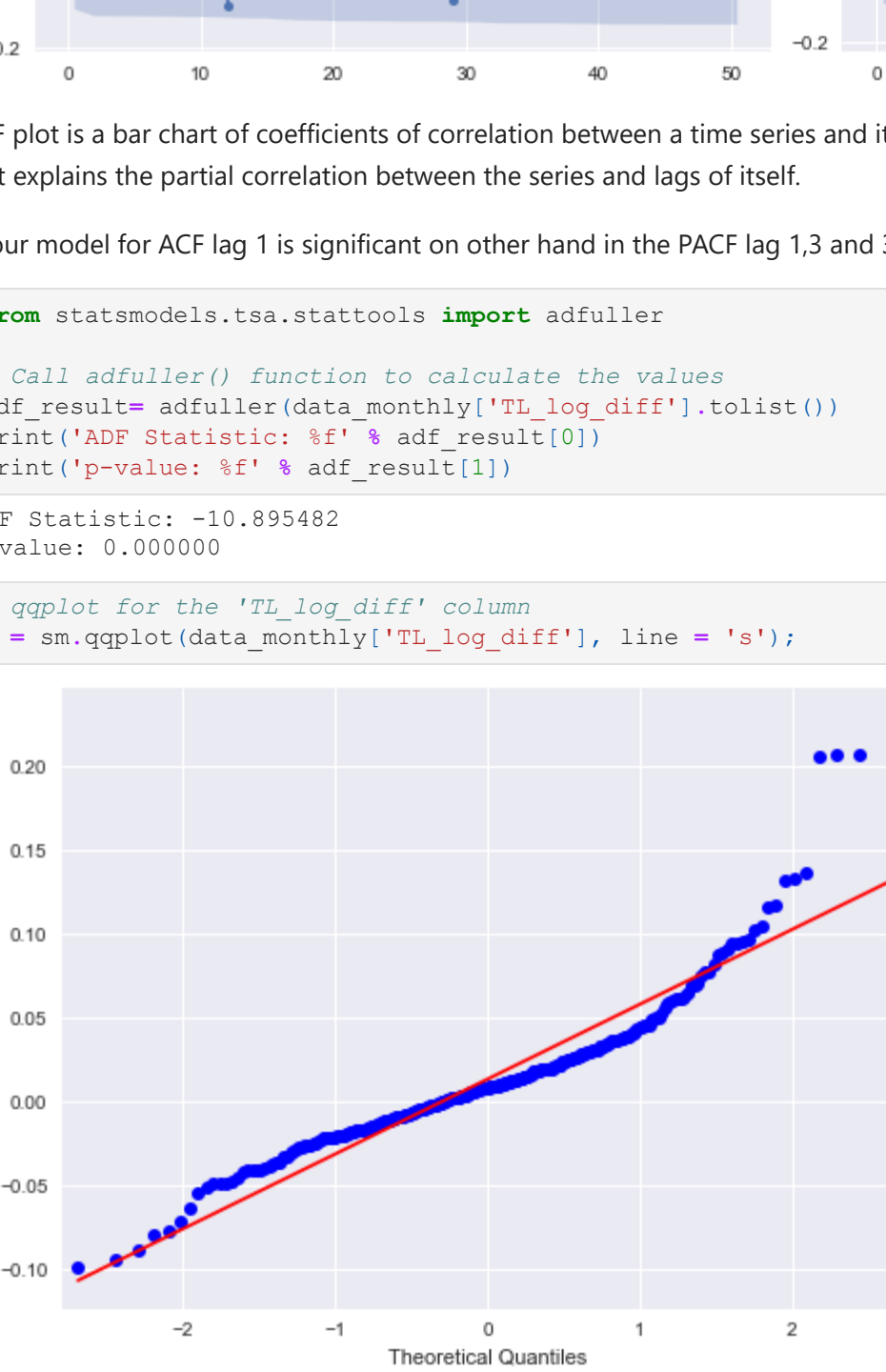
In [34]:

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

In [35]:

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

Out[35]:



In [36]:

```
# Drop the first row of 'TL_log_diff' column because it has 'NaN' value
data_monthly.dropna(subset = ['TL_log_diff'], inplace = True)
```

Out[36]:


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


Out[36]:


	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	-0.842552	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	0.065540	-0.736776	0.030576
1999-10-31	0.503100	0.024455	0.038595	0.059654	0.072791	-0.686966	0.049810

In [37]:

```
# Plot resampled data
plots(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]:

```
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]:

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


After applying 4 lated differences to make dataset stationary, there's another method 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 stationarity of a series.

A Dickey-Fuller test is a unit root test that tests the null hypothesis that $\alpha=1$ in the model equation. α 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]:

```
# A nested loop is written to calculate the AIC values
aicVal=[]
for ar1 in range(1,31):
    for ar2 in range(0,31):
        for ma3 in range(0,31):
            arma_obj = ARIMA(data_monthly['Turkish_Lira']).tolst(), order=(ar1, ar2, ma3))
            arma_obj_fit=arma_obj.fit()
            aicVal.append(aic(ar1, ar2, ma3, arma_obj_fit.aic))
        except ValueError:
            pass
```

Print(aicVal)

```
[10, 1, 0, 58.92003581663309], [0, 1, 1, 4.574842075680181], [0, 1, 2, 6.366776419441165], [1, 1, 0, 11.873448
1, 8.13630242912568], [2, 1, 1, 6.3405316354431], [1, 1, 2, 3.334898809763551], [2, 1, 0, 8.338454635400511], [2, 1,
1, 8.13630242912568], [2, 1, 2, 10.137162621969812], [0, 2, 0, 106.94089464664395], [0, 2, 1, 40.4958817448189
2], [0, 2, 2, 1.729837517674071], [1, 2, 0, 97.89250634261714], [2, 2, 0, 67.69595016077756], [2, 2, 1, 3.4782
1428410506], [2, 2, 2, 3.544885867871221]
```

In [41]:

```
# Declare and initialize an object 'arma_obj' by calling a method 'ARIMA()'
arma_obj = ARIMA(data_monthly['TL_log_diff'].tolist(), order = (0, 2, 2))

# Train the ARIMA model by calling a method fit()
arma_obj_fit = arma_obj.fit(display = 0)

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



```
ARIMA Model Results
Dep. Variable: ARIMA(0, 2, 2) No. Observations: 273
Method: csm-me S.O. of innovations: 0.044
Date: Fri, 03 Jun 2022 AIC: -907.617
Time: 21:21:31 BIC: -893.179
Sample: 2 HQIC: -901.821

coef std err z P>|z| [0.025 0.975]
const 3.70e+06 9.42e-07 3.926 0.000 1.85e+06 5.55e+06
ma.L1.D2.y -1.99e-21 0.012 -169.119 0.000 -2.015 -1.969
ma.L2.D2.y 0.9922 0.012 83.948 0.000 0.969 1.015

Roots
MA.1 1.0039 -0.0047j 1.0039 -0.0007
MA.2 1.0039 +0.0047j 1.0039 0.0007

After making the dataset stationary, there is a loop create to find AIC.
```

The Akaike information criterion (AIC) is a mathematical method for comparing how well a model fits the data it was generated from. It estimates models relatively, meaning that AIC scores are only useful in comparison with other AIC scores for the same dataset. A lower AIC score is better.

After finding the lowest AIC model the ARIMA model was applied;

Our best fit model is ARIMA(0,2,2), it means d=2 (second order differencing) and q=2 is the MA (moving-average) order.

ARIMA short for 'AutoRegressive Integrated Moving Average', is a forecasting algorithm based on the idea that the information in the past values of the time series can be used to predict the future values.

```
ARIMA(p,d,q)

p is the order of the AR term
q is the order of the MA term
d is the number of differencing required to make the time series stationary

In (42):
# Declare the array 'pred' and append the values
pred = np.append(0, 0), arima_obj_fit.fittedvalues.tolist()

# Add a new column 'ARIMA' into dataframe
data_monthly['ARIMA'] = pred

# Residues and fitted values are added and stored into dataframe
TL_log_diff = np.append(0, 0), arima_obj_fit.resid + arima_obj_fit.fittedvalues

# Add a column 'diffval' into dataframe
data_monthly['TL_log_diffval'] = TL_log_diff

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

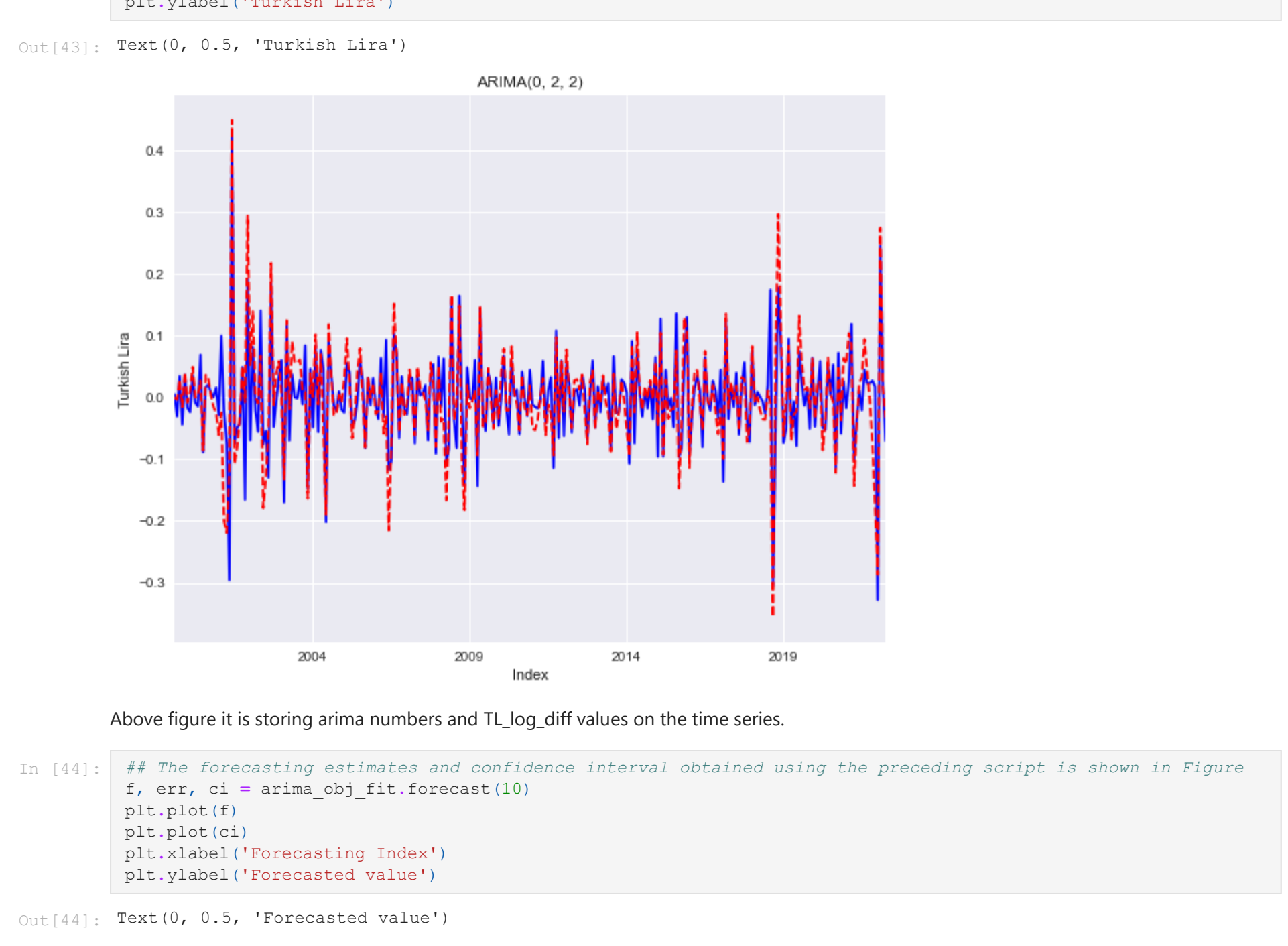
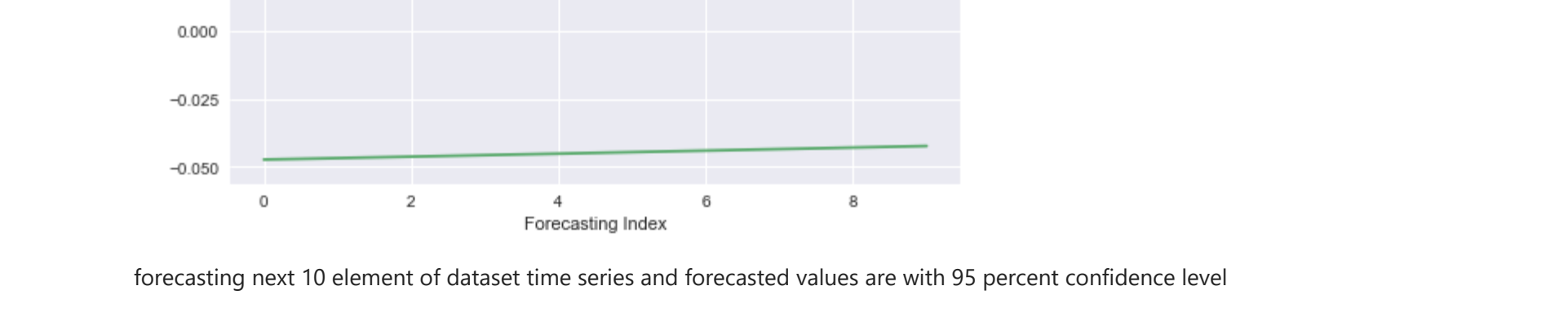


Figure 1 shows the forecasting estimates and confidence interval obtained using the preceding script is shown in Figure 1. The forecasted values are shown in green, and the confidence interval is shown in blue. The x-axis is labeled 'Forecasting Index' and ranges from 0 to 8. The y-axis is labeled 'Forecasted value' and ranges from -0.050 to 0.125.



forecasting next 10 element of dataset time series and forecasted values are with 95 percent confidence level

Question 2: Recommendation system

Content-Based Recommendation System

```
In (3):
#Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.metrics import pairwise_distances

In (2):
# Load the data from a CSV file
book_df = pd.read_csv('dataset/Books.csv', low_memory=False)

In (3):
# See the first 5 records
book_df.head()
```

	ISBN	Book-Title	Book-Author	Year-Of-Publication	Publisher	Image-URL-S
0	0191513448	Classical Mythology	Mark P. O. Morford	2002	Oxford University Press	http://images.amazon.com/images/P/0191513448.0...
1	0002005018	Clara Callan	Richard Bruce Wright	2001	HarperFlamingo Canada	http://images.amazon.com/images/P/0002005018.0...
2	0060973129	Decision in Normandy	Carlo D'Este	1991	HarperPerennial	http://images.amazon.com/images/P/0060973129.0...

3	0374157065	Flu: The Story of the Great Influenza Pandemic...	Gina Bari Kolata	1999	Farrar Straus Giroux	http://images.amazon.com/images/P/0374157065.0...
4	0393045218	The Mummies of Urunchi	E. J. W. Barber	1999	W. W. Norton & Company	http://images.amazon.com/images/P/0393045218.0...

```
In (46):
#Print the shape of dataset
book_df.shape

Out(46):
(271360, 8)

In (47):
#Drop duplicates
book_df = book_df.drop_duplicates()

In (48):
#After dropping duplicates print shape again
book_df.shape

Out(48):
(271360, 8)

In (49):
#Rename columns
book_df = book_df.rename(columns={'Book-Title': 'book_title', 'book-author': 'book_author', 'Year-Of-Publication': 'year_of_publication'})

In (50):
#Print book Authors
book_df['book_author']
```

0	Richard Bruce Wright
1	Richard Bruce Wright
2	Carlo D'Este
3	Gina Bari Kolata
4	E. J. W. Barber
...	...
271355	Paula Danziger
271356	Teri Sloat
271357	Christine Wicker
271358	Plato
271359	Christopher Bliffe
name: book_author, length: 271360, dtype: object	

```
In (51):
#Print title of books
book_df['book_title']

Out(51):
0 Classical Mythology
1 Clara Callan
2 Decision in Normandy
3 Flu: The Story of the Great Influenza Pandemic...
4 The Mummies of Urunchi
...
271355 There's a Bat in Bunk Five
271356 From One to One Hundred
271357 The True Story of the Town that Is...
271358 Republic (World's Classics)
271359 A Guided Tour of Rene Descartes' Meditations o...
Name: book_title, length: 271360, dtype: object
```

```
In (52):
from sklearn.feature_extraction.text import TfidfVectorizer

#Define a TF-IDF Vectorizer Object. Remove all english stop words such as 'the', 'a'
tfidf = TfidfVectorizer(stop_words='english')

#Replace NaN with an empty string
book_df['book_author'] = book_df['book_author'].fillna('')

#Construct the required TF-IDF matrix by fitting and transforming the data
tfidf_matrix = tfidf.fit_transform(book_df['book_author'])

#Output the shape of tfidf_matrix
tfidf_matrix.shape

Out(52):
(271360, 47609)
```

```
In (53):
# Array mapping from feature integer indices to feature name.
tfidf.get_feature_names() [1000:1010]
```

Out(53):

```
['amar',
 'amand',
 'amanda',
 'amanda',
 'amanda',
 'amand',
 'amara',
 'amarantha',
 'amarillas',
 'amarillo']
```

I used the text analytics to convert Book authors to numerical vectors you can observe that 47609 different vocabularies or words in our dataset.

```
In (54):
# Import linear kernel
from sklearn.metrics.pairwise import linear_kernel

# Compute the cosine similarity matrix
cosine_sim = linear_kernel(tfidf_matrix[1:10000], tfidf_matrix[1:10000])

To handle high dimensions we only used the first 10000 records
```

```
In (55):
# cosine_sim the first 10000 records shape
cosine_sim.shape

Out(55):
(10000, 10000)
```

```
In (56):
# Similarity outcome
cosine_sim[1]
```

Out(56): array([0., 0., ..., 0., 0., 0.])

Now it is time to calculate similarity score. There are several similarity metrics that you can use for this, such as the manhattan, euclidean, the Pearson, and the cosine similarity scores.

Here we used the cosine similarity to calculate a numeric quantity as relatively easy and fast to calculate and cosine similarity denotes the similarity between two books.

```
In (57):
# Construct a reverse map of indices and movie titles
indices = pd.Series(book_df.index, index=book_df['book_title']).drop_duplicates()

In (58):
indices.head()
```

book_title	0
Classical Mythology	1
Clara Callan	2
Decision in Normandy	3
Flu: The Story of the Great Influenza Pandemic...	4
The Mummies of Urunchi	5

```
In (59):
# Function that takes in movie title as input and outputs most similar movies
def get_recommendations(title, cosine_sim=cosine_sim):
    # Get the index of the movie that matches the title
    idx = indices(title)

    # Get the pairwise similarity scores of all movies with that movie
    sim_scores = list(enumerate(cosine_sim[idx]))

    # Sort the movies based on the similarity scores
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Get the scores of the 10 most similar movies
    sim_scores = sim_scores[1:11]

    # Get the movie indices
    book_indices = [i[0] for i in sim_scores]

    # Return the top 10 most similar movies
    return book_df[book_indices].iloc[book_indices]
```

In (60):

```
get_recommendations('Clara Callan')
Out(60):
4287 Spy Catcher: The Candid Autobiography of a Sen...
6206 Poison Apples (Worldwide Library Mysteries)
5479 Amsterdam: Amsterdam and the Hague (Art in Focus)
173 Always Daddy's Girl: Understanding Our Father...
2427 The Peculiar Memories of Thomas Penman
2813 Guide to Korean Characters: Reading and Writin...
676 The Mummies of Urunchi
Name: book_title, dtype: object
```

Collaborative Filtering

```
In (4):
#Read dataset with csv
rating_df = pd.read_csv('dataset/Ratings.csv')

In (75):
#Printing the first 5 records
rating_df.head()
```

	User-ID	ISBN	Book-Rating
0	276725	034545104X	0
1	276726	0155061224	5
2	276727	0446520802	0
3	276729	052165615X	3
4	276729	0521795028	6

```
In (5):
#Reading 'user_df' with csv
user_df = pd.read_csv('dataset/Users.csv')

In (77):
#Printing the first 5 records
user_df.head()
```

	User-ID	Location	Age
0	1	ny, ny, ny, ny, ny	NaN
1	2	stockton, california, usa	18.0
2	3	moscow, ykron territory, russia	NaN
3	4	porto, v.n.gala, portugal	17.0
4	5	farnborough, hants, united kingdom	NaN

```
In (6):
#Printing the first 5 records of 'book_df' dataset
book_df.head()
```

	ISBN	Book-Title	Book-Author	Year-Of-Publication	Publisher	Image-URL-S
0	0191513448	Classical Mythology	Mark P. O. Morford	2002	Oxford University Press	http://images.amazon.com/images/P/0191513448.0...
1	0002005018	Clara Callan	Richard Bruce Wright	2001	HarperFlamingo Canada	http://images.amazon.com/images/P/0002005018.0...
2	0060973129	Decision in Normandy	Carlo D'Este	1991	HarperPerennial	http://images.amazon.com/images/P/0060973129.0...

3	0374157065	Flu: The Story of the Great Influenza Pandemic...	Gina Bari Kolata	1999	Farrar Straus Giroux	http://images.amazon.com/images/P/0374157065.0...
4	0393045218	The Mummies of Urunchi	E. J. W. Barber	1999	W. W. Norton & Company	http://images.amazon.com/images/P/0393045218.0...

```
In (11):
rating_df.info()
book_df.info()

Out(11):
Out(12):
Out(13):
Out(14):
Out(15):
Out(16):
Out(17):
Out(18):
Out(19):
Out(20):
Out(21):
Out(22):
Out(23):
Out(24):
Out(25):
Out(26):
Out(27):
Out(28):
Out(29):
Out(30):
Out(31):
Out(32):
Out(33):
Out(34):
Out(35):
Out(36):
Out(37):
Out(38):
Out(39):
Out(40):
Out(41):
Out(42):
Out(43):
Out(44):
Out(45):
Out(46):
Out(47):
Out(48):
Out(49):
Out(50):
Out(51):
Out(52):
Out(53):
Out(54):
Out(55):
Out(56):
Out(57):
Out(58):
Out(59):
Out(60):
Out(61):
Out(62):
Out(63):
Out(64):
Out(65):
Out(66):
Out(67):
Out(68):
Out(69):
Out(70):
Out(71):
Out(72):
Out(73):
Out(74):
Out(75):
Out(76):
Out(77):
Out(78):
Out(79):
Out(80):
Out(81):
Out(82):
Out(83):
Out(84):
Out(85):
Out(86):
Out(87):
Out(88):
Out(89):
Out(90):
Out(91):
Out(92):
Out(93):
Out(94):
Out(95):
Out(96):
Out(97):
Out(98):
Out(99):
Out(100):
Out(101):
Out(102):
Out(103):
Out(104):
Out(105):
Out(106):
Out(107):
Out(108):
Out(109):
Out(110):
Out(111):
Out(112):
Out(113):
Out(114):
Out(115):
Out(116):
Out(117):
Out(118):
Out(119):
Out(120):
Out(121):
Out(122):
Out(123):
Out(124):
Out(125):
Out(126):
Out(127):
Out(128):
Out(129):
Out(130):
Out(131):
Out(132):
Out(133):
Out(134):
Out(135):
Out(136):
Out(137):
Out(138):
Out(139):
Out(140):
Out(141):
Out(142):
Out(143):
Out(144):
Out(145):
Out(146):
Out(147):
Out(148):
Out(149):
Out(150):
Out(151):
Out(152):
Out(153):
Out(154):
Out(155):
Out(156):
Out(157):
Out(158):
Out(159):
Out(160):
Out(161):
Out(162):
Out(163):
Out(164):
Out(165):
Out(166):
Out(167):
Out(168):
Out(169):
Out(170):
Out(171):
Out(172):
Out(173):
Out(174):
Out(175):
Out(176):
Out(177):
Out(178):
Out(179):
Out(180):
Out(181):
Out(182):
Out(183):
Out(184):
Out(185):
Out(186):
Out(187):
Out(188):
Out(189):
Out(190):
Out(191):
Out(192):
Out(193):
Out(194):
Out(195):
Out(196):
Out(197):
Out(198):
Out(199):
Out(200):
Out(201):
Out(202):
Out(203):
Out(204):
Out(205):
Out(206):
Out(207):
Out(208):
Out(209):
Out(210):
Out(211):
Out(212):
Out(213):
Out(214):
Out(215):
Out(216):
Out(217):
Out(218):
Out(219):
Out(220):
Out(221):
Out(222):
Out(223):
Out(224):
Out(225):
Out(226):
Out(227):
Out(228):
Out(229):
Out(230):
Out(231):
Out(232):
Out(233):
Out(234):
Out(235):
Out(236):
Out(237):
Out(238):
Out(239):
Out(240):
Out(241):
Out(242):
Out(243):
Out(244):
Out(245):
Out(246):
Out(247):
Out(248):
Out(249):
Out(250):
Out(251):
Out(252):
Out(253):
Out(254):
Out(255):
Out(256):
Out(257):
Out(258):
Out(259):
Out(260):
Out(261):
Out(262):
Out(263):
Out(264):
Out(265):
Out(266):
Out(267):
Out(268):
Out(269):
Out(270):
Out(271):
Out(272):
Out(273):
Out(274):
Out(275):
Out(276):
Out(277):
Out(278):
Out(279):
Out(280):
Out(281):
Out(282):
Out(283):
Out(284):
Out(285):
Out(286):
Out(287):
Out(288):
Out(289):
Out(290):
Out(291):
Out(292):
Out(293):
Out(294):
Out(295):
Out(296):
Out(297):
Out(298):
Out(299):
Out(300):
Out(301):
Out(302):
Out(303):
Out(304):
Out(305):
Out(306):
Out(307):
Out(308):
Out(309):
Out(310):
Out(311):
Out(312):
Out(313):
Out(314):
Out(315):
Out(316):
Out(317):
Out(318):
Out(319):
Out(320):
Out(321):
Out(322):
Out(323):
Out(324):
Out(325):
Out(326):
Out(327):
Out(328):
Out(329):
Out(330):
Out(331):
Out(332):
Out(333):
Out(334):
Out(335):
Out(336):
Out(337):
Out(338):
Out(339):
Out(340):
Out(341):
Out(342):
Out(343):
Out(344):
Out(345):
Out(346):
Out(347):
Out(348):
Out(349):
Out(350):
Out(351):
Out(352):
Out(353):
Out(354):
Out(355):
Out(356):
Out(357):
Out(358):
Out(359):
Out(360):
Out(361):
Out(362):
Out(363):
Out(364):
Out(365):
Out(366):
Out(367):
Out(368):
Out(369):
Out(370):
Out(371):
Out(372):
Out(373):
Out(374):
Out(375):
Out(376):
Out(377):
Out(378):
Out(379):
Out(380):
Out(381):
Out(382):
Out(383):
Out(384):
Out(385):
Out(386):
Out(387):
Out(388):
Out(389):
Out(390):
Out(391):
Out(392):
Out(393):
Out(394):
Out(395):
Out(396):
Out(397):
Out(398):
Out(399):
Out(400):
Out(401):
Out(402):
Out(403):
Out(404):
Out(405):
Out(406):
Out(407):
Out(408):
Out(409):
Out(410):
Out(411):
Out(412):
Out(413):
Out(414):
Out(415):
Out(416):
Out(417):
Out(418):
Out(419):
Out(420):
Out(421):
Out(422):
Out(423):
Out(424):
Out(425):
Out(426):
Out(427):
Out(428):
Out(429):
Out(430):
Out(431):
Out(432):
Out(433):
Out(434):
Out(435):
Out(436):
Out(437):
Out(438):
Out(439):
Out(440):
Out(441):
Out(442):
Out(443):
Out(444):
Out(445):
Out(446):
Out(447):
Out(448):
Out(449):
Out(450):
Out(451):
Out(452):
Out(453):
Out(454):
Out(455):
Out(456):
Out(457):
Out(458):
Out(459):
Out(460):
Out(461):
Out(462):
Out(463):
Out(464):
Out(465):
Out(466):
Out(467):
Out(468):
Out(469):
Out(470):
Out(471):
Out(472):
Out(473):
Out(474):
Out(475):
Out(476):
Out(477):
Out(478):
Out(479):
Out(480):
Out(481):
Out(482):
Out(483):
Out(484):
Out(485):
Out(486):
Out(487):
Out(488):
Out(489):
Out(490):
Out(491):
Out(492):
Out(493):
Out(494):
Out(495):
Out(496):
Out(497):
Out(498):
Out(499):
Out(500):
Out(501):
Out(502):
Out(503):
Out(504):
Out(505):
Out(506):
Out(507):
Out(508):
Out(509):
Out(510):
Out(511):
Out(512):
Out(513):
Out(514):
Out(515):
Out(516):
Out(517):
Out(518):
Out(519):
Out(520):
Out(521):
Out(522):
Out(523):
Out(524):
Out(525):
Out(526):
Out(527):
Out(528):
Out(529):
Out(530):
Out(531):
Out(532):
Out(533):
Out(534):
Out(535):
Out(536):
Out(537):
Out(538):
Out(539):
Out(540):
Out(541):
Out(542):
Out(543):
Out(544):
Out(545):
Out(546):
Out(547):
Out(548):
Out(549):
Out(550):
Out(551):
Out(552):
Out(553):
Out(554):
Out(555):
Out(556):
Out(557):
Out(558):
Out(559):
Out(560):
Out(561):
Out(562):
Out(563):
Out(564):
Out(565):
Out(566):
Out(567):
Out(568):
Out(569):
Out(570):
Out(571):
Out(572):
Out(573):
Out(574):
Out(575):
Out(576):
Out(577):
Out(578):
Out(579):
Out(580):
Out(581):
Out(582):
Out(583):
Out(584):
Out(585):
Out(586):
Out(587):
Out(588):
Out(589):
Out(590):
Out(591):
Out(592):
Out(593):
Out(594):
Out(595):
Out(596):
Out(597):
Out(598):
Out(599):
Out(600):
Out(601):
Out(602):
Out(603):
Out(604):
Out(605):
Out(606):
Out(607):
Out(608):
Out(609):
Out(610):
Out(611):
Out(612):
Out(613):
Out(614):
Out(615):
Out(616):
Out(617):
Out(618):
Out(619):
Out(620):
Out(621):
Out(622):
Out(623):
Out(624):
Out(625):
Out(626):
Out(627):
Out(628):
Out(629):
Out(630):
Out(631):
Out(632):
Out(633):
Out(634):
Out(635):
Out(636):
Out(637):
Out(638):
Out(639):
Out(640):
Out(641):
Out(642):
Out(643):
Out(644):
Out(645):
Out(646):
Out(647):
Out(648):
Out(649):
Out(650):
Out(651):
Out(652):
Out(653):
Out(654):
Out(655):
Out(656):
Out(657):
Out(658):
Out(659):
Out(660):
Out(661):
Out(662):
Out(663):
Out(664):
Out(665):
Out(666):
Out(667):
Out(668):
Out(669):
Out(670):
Out(671):
Out(672):
Out(673):
Out(674):
Out(675):
Out(676):
Out(677):
Out(678):
Out(679):
Out(680):
Out(681):
Out(682):
Out(683):
Out(684):
Out(685):
Out(686):
Out(687):
Out(688):
Out(689):
Out(690):
Out(691):
Out(692):
Out(693):
Out(694):
Out(695):
Out(696):
Out(697):
Out(698):
Out(699):
Out(700):
Out(701):
Out(702):
Out(703):
Out(704):
Out(705):
Out(706):
Out(707):
Out(708):
Out(709):
Out(710):
Out(711):
Out(712):
Out(713):
Out(714):
Out(715):
Out(716):
Out(717):
Out(718):
Out(719):
Out(720):
Out(721):
Out(722):
Out(723):
Out(724):
Out(725):
Out(726):
Out(727):
Out(728):
Out(729):
Out(730):
Out(731):
Out(732):
Out(733):
Out(734):
Out(735):
Out(736):
Out(737):
Out(738):
Out(739):
Out(740):
Out(741):
Out(742):
Out(743):
Out(744):
Out(745):
Out(746):
Out(747):
Out(748):
Out(749):
Out(750):
Out(751):
Out(752):
Out(753):
Out(754):
Out(755):
Out(756):
Out(757):
Out(758):
Out(759):
Out(760):
Out(761):
Out(762):
Out(763):
Out(764):
Out(765):
Out(766):
Out(767):
Out(768):
Out(769):
Out(770):
Out(771):
Out(772):
Out(773):
Out(774):
Out(775):
Out(776):
Out(777):
Out(778):
Out(779):
Out(780):
Out(781):
Out(782):
Out(783):
Out(784):
Out(785):
Out(786):
Out(787):
Out(788):
Out(789):
Out(790):
Out(791):
Out(792):
Out(793):
Out(794):
Out(795):
Out(796):
Out(797):
Out(798):
Out(799):
Out(800):
Out(801):
Out(802):
Out(803):
Out(804):
Out(805):
Out(806):
Out(807):
Out(808):
Out(809):
Out(810):
Out(811):
Out(812):
Out(813):
Out(814):
Out(815):
Out(816):
Out(817):
Out(818):
Out(819):
Out(820):
Out(821):
Out(822):
Out(823):
Out(824):
Out(825):
Out(826):
Out(827):
Out(828):
Out(829):
Out(830):
Out(831):
Out(832):
Out(833):
Out(834):
Out(835):
Out(836):
Out(837):
Out(838):
Out(839):
Out(840):
Out(841):
Out(842):
Out(843):
Out(844):
Out(845):
Out(846):
Out(847):
Out(848):
Out(849):
Out(850):
Out(851):
Out(852):
Out(853):
Out(854):
Out(855):
Out(856):
Out(857):
Out(858):
Out(859):
Out(860):
Out(861):
Out(862):
Out(863):
Out(864):
Out(865):
Out(866):
Out(867):
Out(868):
Out(869):
Out(870):
Out(871):
Out(872):
Out(873):
Out(874):
Out(875):
Out(876):
Out(877):
Out(878):
Out(879):
Out(880):
Out(881):
Out(882):
Out(883):
Out(884):
Out(885):
Out(886):
Out(887):
Out(888):
Out(889):
Out(890):
Out(891):
Out(892):
Out(893):
Out(894):
Out(895):
Out(896):
Out(897):
Out(898):
Out(899):
Out(900):
Out(901):
Out(902):
Out(903):
Out(904):
Out(905):
Out(906):
Out(907):
Out(908):
Out(909):
Out(910):
Out(911):
Out(912):
Out(913):
Out(914):
Out(915):
Out(916):
Out(917):
Out(918):
Out(919):
Out(920):
Out(921):
Out(922):
Out(923):
Out(924):
Out(925):
Out(926):
Out(927):
Out(928):
Out(929):
Out(930):
Out(931):
Out(932):
Out(933):
Out(934):
Out(935):
Out(936):
Out(937):
Out(938):
Out(939):
Out(940):
Out(941):
Out(942):
Out(943):
Out(944):
Out(945):
Out(946):
Out(947):
Out(948):
Out(949):
Out(950):
Out(951):
Out(952):
Out(953):
Out(954):
Out(955):
Out(956):
Out(957):
Out(958):
Out(959):
Out(960):
Out(961):
Out(962):
Out(963):
Out(964):
Out(965):
Out(966):
Out(967):
Out(968):
Out(969):
Out(970):
Out(971):
Out(972):
Out(973):
Out(974):
Out(975):
Out(976):
Out(977):
Out(978):
Out(979):
Out(980):
Out(981):
Out(982):
Out(983):
Out(984):
Out(985):
Out(986):
Out(987):
Out(988):
Out(989):
Out(990):
Out(991):
Out(992):
Out(993):
Out(994):
Out(995):
Out(996):
Out(997):
Out(998):
Out(999):
Out(1000):
Out
```



```
In [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 dataset
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: 906.7+ 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 white spaces or specified characters at the start and end of a string
Grocery_df['ItemDescription'] = Grocery_df['ItemDescription'].str.strip()

# str.strip is a Pandas function for DataFrames (and numpy for numpy arrays) that will cast the object to the appropriate dtype
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

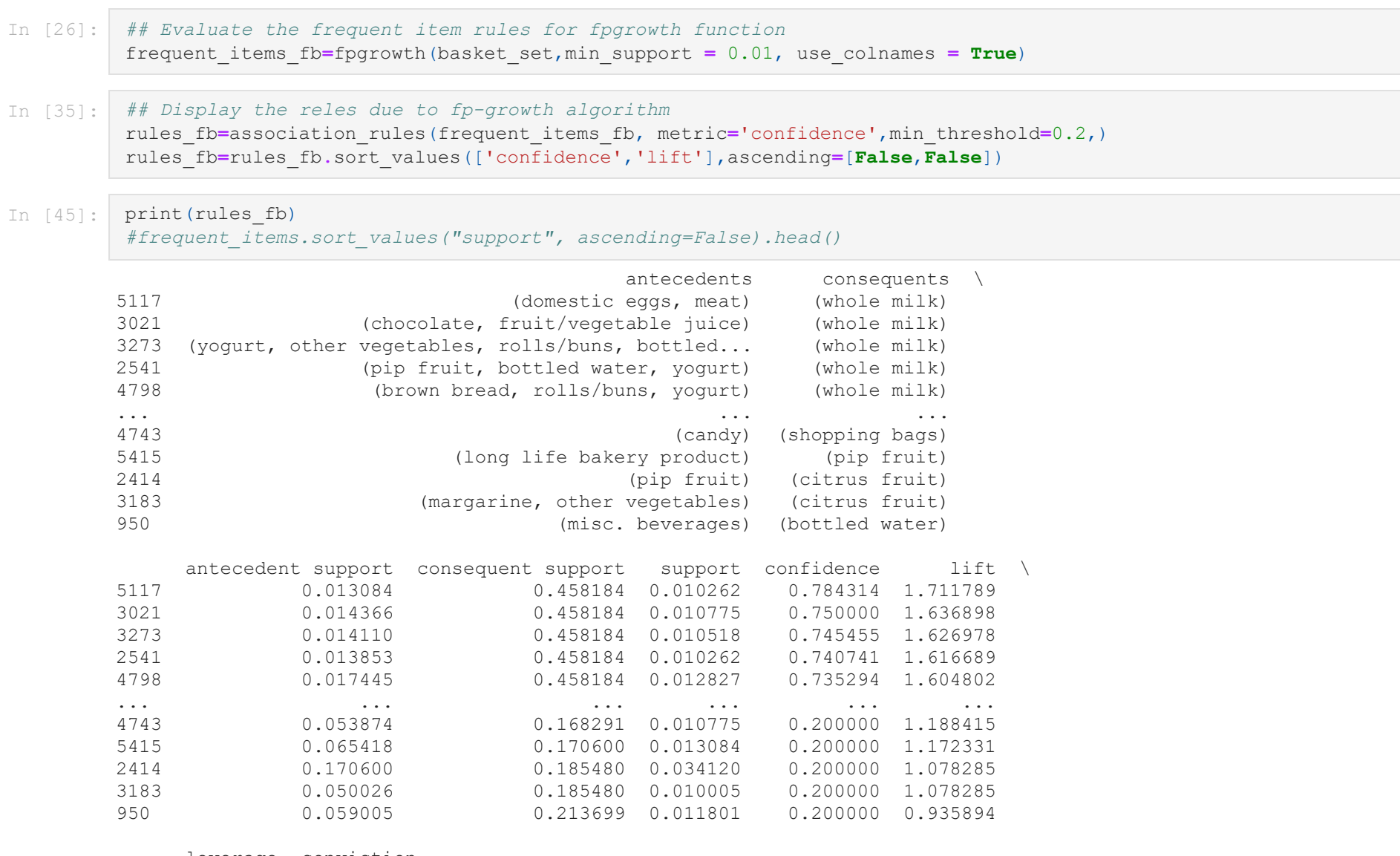
In [13]: Grocery_df.head()

Out[13]:
```



```
In [37]: #to show the number of items sold in basket
basket = Grocery_df.groupby(['Member_number', 'ItemDescription']).head(10).plot(kind='bar')

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



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

In [19]: 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	0.0	0.0
1001	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	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	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	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	0.0	0.0

5 rows x 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	0	0
1001	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
1002	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
1004	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

5 rows x 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 rules 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	confidence	lift
194	(bottled water, other vegetables)	(whole milk)	0.598361	1.305841
236	(yogurt, other vegetables)	(whole milk)	0.597015	1.393003
247	(rolls/buns, yogurt)	(whole milk)	0.592166	1.292420
206	(rolls/buns, other vegetables)	(whole milk)	0.559441	1.220996
254	(yogurt, soda)	(whole milk)	0.557895	1.217622
58	(whole milk)	(coffee)	0.120381	1.047420
232	(whole milk)	(yogurt, soda)	0.118701	1.217622
219	(whole milk)	(tropical fruit, other vegetables)	0.103022	1.207749
46	(whole milk)	(sausage, other vegetables)	0.109742	1.181702
	(whole milk)	(chicken)	0.109183	1.085698

	antecedent	support	consequent	support	confidence	lift
194	0.013162	1.349012	0.458184	0.056183	0.598361	1.305841
236	0.016704	1.344507	0.458184	0.071832	0.597015	1.393003
247	0.014917	1.328521	0.458184	0.065931	0.592166	1.292420
206	0.014859	1.229837	0.458184	0.050339	0.559441	1.220996
254	0.009720	1.225537	0.458184	0.054387	0.557895	1.217622
58	0.002497	1.006196	0.114931	0.055156	0.120381	1.047420
232	0.009720	1.024073	0.097486	0.054387	0.118701	1.217622
219	0.008693	1.023126	0.091329	0.050339	0.103022	1.207749
46	0.007732	1.018994	0.092868	0.050282	0.109742	1.181702
219	0.003949	1.009674	0.100564	0.050026	0.109183	1.085698

(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]: (2.4268898971155837, 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 indicate to put them together as they are popular

FP-growth algorithm for Market Basket Analysis

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

In [35]: # Display the rules 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	confidence	lift
5117	(domestic eggs, meat)	(whole milk)	0.784314	1.711789
3021	(chocolate, fruit/vegetable juice)	(whole milk)	0.750000	1.636898
3273	(yogurt, other vegetables, rolls/buns, bottled water)	(whole milk)	0.745455	1.624978
2541	(pip fruit, bottled water, yogurt)	(whole milk)	0.740741	1.616689
4798	(brown bread, rolls/buns, yogurt)	(whole milk)	0.735294	1.604602
...
4743	(candy)	(shopping bags)	0.200000	1.188415
5415	(long life bakery product)	(pip fruit)	0.200000	1.172331
2414	(pip fruit)	(citrus fruit)	0.200000	1.078285
3183	(margarine, other vegetables)	(citrus fruit)	0.200000	1.078285
950	(misc. beverages)	(bottled water)	0.200000	0.935894

	antecedent	support	consequent	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	0.010318	0.745455	1.624978	
2541	0.013853	0.458184	0.010262	0.740741	1.616689	
4798	0.017445	0.458184	0.012627	0.735294	1.604602	
...	
4743	0.053874	0.168291	0.010775	0.200000	1.188415	
5415	0.056418	0.170600	0.010384	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	1.039836
5415	0.001923	1.036750
2414	0.002477	1.018150
3183	0.000726	1.018150
950	-0.000808	0.982876

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

References:

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-9c5ba1c96ced>.

Rocca, B. (2019). Introduction to recommender systems. [online] Medium. Available at: <https://towardsdatascience.com/introduction-to-recommender-systems-6c66cf15ada9?text=The%20purpose%20of%20a%20recommender>.

```
In [ ]:
```