"Now casting GDP Growth and Forecasting Inflation of Pakistan "

Project Report

Course:

Machine Learning-I

Submitted by:

Sarmad Zafar

Submitted to:

Sir Tariq Mahmood

Introduction:

This project consist of forecasting inflation and now casting GDP growth. The intuition behind this is, as pre-requisite for good macroeconomic policymaking is timely information on the current state of the economy. Different indicators are used by policy maker for policy making includes GDP growth, inflation, unemployment, trade, exchange rate etc. Of them GDP and inflation are the one with great impact.

Inflation data is released monthly in most of the countries known as CPI (Consumer Price Index), in Pakistan PBS release CPI data on first working day of each month. Policy maker always tried to forecast inflation as inflation targeting is main focus for sustainable growth. In this project I used different techniques to forecast CPI.

Next main indicator is GDP growth rapidly. Given that GDP is usually only available on a quarterly basis, whereas in the case of Pakistan it's available annually with first estimates typically published only 4 to 6 weeks or more after the end of the quarter or years. As it's a low frequency data as well as available with a considerable lag making dire need to make some proxy indicator for GDP as economic activity is changing rapidly which requires proactive policy making. So policymakers and forecasters have long made use of more timely higher frequency data, such as survey-based indicators like Purchasing Managers' Indices (PMIs) to gauge the economic activity but its have its own many issues. This problem has prompted a search for alternative high-frequency indicators of economic activity. In this project I used indicator based on Google Trends, which are used to construct a quarterly estimates of GDP growth for Pakistan

Background Knowledge:

1. **Theoretical Knowledge:**
   Although I haven't degree in economics but currently working in State Bank economics cluster, for around one year. So working hear I have learned a lot related to economics which also subject of my project. With practical experience, I have studied books and other stuff to get detailed understanding of field. Plus monthly and annual reports also add up significantly into y theoretical knowledge.

2. **Training/Workshops:**
   I have attended different workshops and training related to project domains which includes: IMF (Macroeconometric Forecasting, Monetary Policy Analysis and Forecasting), IBA(Introduction to machine learning for Economist), Banque de France (Big Data) etc.

3. **Data Source:**
   Some of the main data sources includes: Bloomberg, IMF Database and Google Trends from where a no of features are extracted these includes: Exchange Rate, Oil Price, Monetary Aggregate (M0, M2), LSM etc.

4. **Tools and Technique:**
   Python is used as main tool with different libraries. Some other tools used or data preparation and understanding the problem includes Stata, Haver Analytics and Tableau

5. **Research Papers:**
   I have gone through some research paper to understand and solve the problem at hand, these include:
   - Macroeconomic Nowcasting and Forecasting with Big Data
   - A comparison of time series and machine learning models for inflation forecasting: empirical evidence from the USA
   - Tracking activity in real time with Google Trends

6. **Current Market Practices:**
   I also gone through the Machine learning techniques which are used in different international organization and central banks.

Problems:

1) Inflation Forecasting:

Monthly CPI data is released by PBS on first working day of each month, giving the actual inflation for the current month. We have many different indicators including:

ER	Nominal Rs/US $ Exchange Rate (Average)
CPI	Consumer Price Index
M0	Reserve Money, Stock in million Rupees
M2	Monetary Assets, Stock in million Rupees
RES$	Liquid Foreign Exchange Reserves, Million US Dollars
Remit	Home Remittances, Million US Dollars
IPI-LSM	Quantum Index of Large-scale Manufacturing
3M-TB-Rate	3 month T-Bill weighted average rate
Dis Rate	Discount Rate
OIL	International Oil Price
USIPI	US Industrial Production 2010=100, seasonally adjusted
PSC	Private Sector Credit by all scheduled banks, Stock in million Rupees
PSB	Public Sector Borrrowing (From whole banking system by federal and provincial governments), Stock in million Rupees
LR	Nominal Lending Rates (Incremental)
REER	Real Effective Exchange Rate Indices
WPI	Wholesale Price Index
WCPI	World Consumer Price Index (2016=100)

- **Given all these data and business requirement the problem at hand is to forecast the inflation at some forecast horizon, I have generated the forecast for next 12 months**

2) GDP Nowcast

We have GDP growth rate is available annually with a lag of 1 to 2 months means if fiscal year is ending on 30th June, data for GDP will we released in August. First the frequency of this data is already very low i-e annual on top of that it's available with a considerable lag. When economic activity is changing rapidly, for good macroeconomic policymaking the timely information on the current state of the economy is crucial. So the problem is to made a proxy indicator with high frequency timely availability,

- **One solution is to make such indicator based on Google Trends, which are used to construct a GDP Tracker**

Results:

Problem-1

Sno.	Technique	Training RMSE	Testing RMSE
1)	Exponential Smoothing	0.569	1.918
2)	Moving Average	0.513	1.627
3)	AutoRegression	0.534	1.703
4)	ARMA	0.512	6.995
5)	ARIMA	0.199	0.044
6)	SARIMAX	0.222	0.072
7)	LSTM	-	2.352

Problem-2

Sno.	Technique	Mean Absolute Error	Mean Square Error
1)	Multi-Layer Perceptron	-	14.634
2)	Decision Tree	4.351	19.38
3)	Randomforest	3.44	14.19
4)	KNN	2.911	10.25
5)	Linear Regression	5.40	35.17
6)	SVM	4.19	17.84
7)	Gradient Boosting	3.62	14.06
8)	Adaptive Boosting	3.56	14.06

Multivariate LSTM Forecast Model

In [57]:
```
# import related libraries
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
import pandas as pd
import numpy as np
import seaborn as sns
from matplotlib import pyplot as plt
import math
from pandas import DataFrame
from pandas import concat
from numpy import concatenate
```

In [58]:
```
import warnings
warnings.filterwarnings("ignore")
```

In [59]:
```
# Reading File
df =  pd.read_excel('CPI_Urban.xls')
```

In [60]:
```
df = df.set_index('Date')
df = df.dropna()
```

In [61]:
```
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
    n_vars = 1 if type(data) is list else data.shape[1]
    df = DataFrame(data)
    cols, names = list(), list()
    # input sequence (t-n, ... t-1)
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
    # forecast sequence (t, t+1, ... t+n)
    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i == 0:
            names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
        else:
            names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
    # put it all together
    agg = concat(cols, axis=1)
    agg.columns = names
    # drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)
    return agg
```

In [62]:
```
# normalize features
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(df)
scaled
```

Out[62]:
```
array([[0.00000000e+00, 0.00000000e+00, 0.00000000e+00, ...,
        6.31178956e-03, 0.00000000e+00, 8.87167101e-02],
       [0.00000000e+00, 5.19262411e-04, 3.68649927e-04, ...,
        1.07514639e-02, 0.00000000e+00, 1.03937898e-01],
       [0.00000000e+00, 2.94389464e-03, 1.36683664e-03, ...,
        1.38152005e-02, 0.00000000e+00, 1.18357971e-01],
       ...,
       [9.82532343e-01, 9.69150692e-01, 9.66522536e-01, ...,
        4.97045362e-01, 1.00000000e+00, 9.29439743e-01],
       [9.85418303e-01, 9.18420718e-01, 9.68870117e-01, ...,
        5.04508927e-01, 1.00000000e+00, 9.49853142e-01],
       [1.00000000e+00, 8.99225593e-01, 1.00000000e+00, ...,
        5.33439387e-01, 1.00000000e+00, 7.87777515e-01]])
```

In [63]:
```
# frame as supervised learning
reframed = series_to_supervised(scaled, 1, 1)
```

In [64]:
```
reframed.head(5)
```

Out[64]:

	var1(t-1)	var2(t-1)	var3(t-1)	var4(t-1)	var5(t-1)	var6(t-1)	var7(t-1)	var8(t-1)	var9(t-1)	var10(t-1)	...
1	0.0	0.000000	0.000000	0.000000	0.003603	0.027522	0.102578	0.407773	0.298246	0.076283	...
2	0.0	0.000519	0.000369	0.000091	0.003603	0.022119	0.079005	0.407773	0.298246	0.035898	...
3	0.0	0.002944	0.001367	0.000165	0.003603	0.018717	0.087863	0.407773	0.298246	0.039161	...
4	0.0	0.007682	0.001359	0.000258	0.003603	0.018994	0.029573	0.472634	0.298246	0.043241	...
5	0.0	0.011053	0.001848	0.000369	0.003603	0.022585	0.022573	0.508494	0.298246	0.047728	...

5 rows × 38 columns

In [65]:
```
reframed.dtypes
```

Out[65]:
```
var1(t-1)     float64
var2(t-1)     float64
var3(t-1)     float64
var4(t-1)     float64
var5(t-1)     float64
var6(t-1)     float64
var7(t-1)     float64
var8(t-1)     float64
var9(t-1)     float64
var10(t-1)    float64
var11(t-1)    float64
var12(t-1)    float64
var13(t-1)    float64
var14(t-1)    float64
var15(t-1)    float64
var16(t-1)    float64
var17(t-1)    float64
var18(t-1)    float64
var19(t-1)    float64
var1(t)       float64
var2(t)       float64
var3(t)       float64
var4(t)       float64
var5(t)       float64
var6(t)       float64
var7(t)       float64
var8(t)       float64
var9(t)       float64
var10(t)      float64
var11(t)      float64
var12(t)      float64
var13(t)      float64
var14(t)      float64
var15(t)      float64
var16(t)      float64
var17(t)      float64
var18(t)      float64
var19(t)      float64
dtype: object
```

In [66]:
```
a = range(20,38,1)
reframed.drop(reframed.columns[a], axis=1, inplace=True)
reframed.dtypes
```

Out[66]:
```
var1(t-1)     float64
var2(t-1)     float64
var3(t-1)     float64
var4(t-1)     float64
var5(t-1)     float64
var6(t-1)     float64
var7(t-1)     float64
var8(t-1)     float64
var9(t-1)     float64
var10(t-1)    float64
var11(t-1)    float64
var12(t-1)    float64
var13(t-1)    float64
var14(t-1)    float64
var15(t-1)    float64
var16(t-1)    float64
var17(t-1)    float64
var18(t-1)    float64
var19(t-1)    float64
var1(t)       float64
dtype: object
```

In [73]:
```
values = reframed.values
train = values[:-12, :]
test = values[-12:]
# split into input and outputs
train_X, train_y = train[:, :-1], train[:, -1]
test_X, test_y = test[:, :-1], test[:, -1]
# reshape input to be 3D [samples, timesteps, features]
train_X = train_X.reshape((train_X.shape[0], 1, train_X.shape[1]))
test_X = test_X.reshape((test_X.shape[0], 1, test_X.shape[1]))
print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)
```

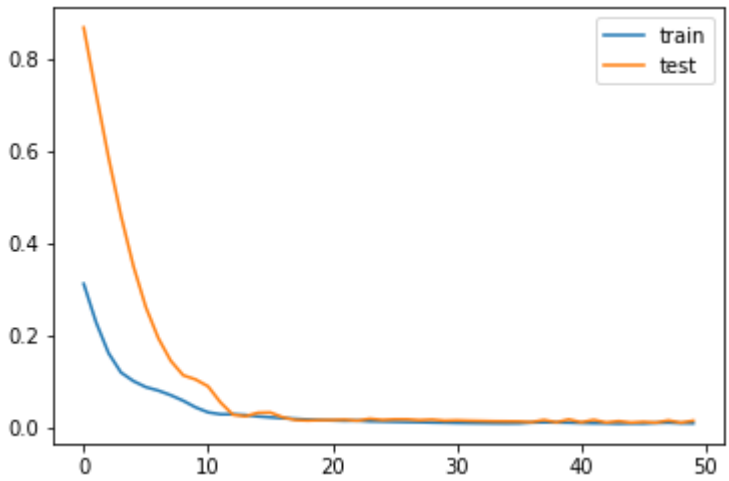
```
(351, 1, 19) (351,) (12, 1, 19) (12,)
```

In [68]:

```
model = Sequential()
model.add(LSTM(50, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(1))
model.compile(loss='mae', optimizer='adam')
# fit network
history = model.fit(train_X, train_y, epochs=50, batch_size=72, validation_data=(test
_X, test_y), verbose=2, shuffle=False)
# plot history
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend()
plt.show()
```

```
Train on 351 samples, validate on 12 samples
Epoch 1/50
 - 1s - loss: 0.3129 - val_loss: 0.8686
Epoch 2/50
 - 0s - loss: 0.2290 - val_loss: 0.7250
Epoch 3/50
 - 0s - loss: 0.1625 - val_loss: 0.5863
Epoch 4/50
 - 0s - loss: 0.1206 - val_loss: 0.4601
Epoch 5/50
 - 0s - loss: 0.1020 - val_loss: 0.3509
Epoch 6/50
 - 0s - loss: 0.0889 - val_loss: 0.2620
Epoch 7/50
 - 0s - loss: 0.0815 - val_loss: 0.1950
Epoch 8/50
 - 0s - loss: 0.0715 - val_loss: 0.1459
Epoch 9/50
 - 0s - loss: 0.0594 - val_loss: 0.1146
Epoch 10/50
 - 0s - loss: 0.0450 - val_loss: 0.1055
Epoch 11/50
 - 0s - loss: 0.0343 - val_loss: 0.0906
Epoch 12/50
 - 0s - loss: 0.0302 - val_loss: 0.0556
Epoch 13/50
 - 0s - loss: 0.0301 - val_loss: 0.0291
Epoch 14/50
 - 0s - loss: 0.0286 - val_loss: 0.0257
Epoch 15/50
 - 0s - loss: 0.0254 - val_loss: 0.0333
Epoch 16/50
 - 0s - loss: 0.0235 - val_loss: 0.0339
Epoch 17/50
 - 0s - loss: 0.0215 - val_loss: 0.0235
Epoch 18/50
 - 0s - loss: 0.0202 - val_loss: 0.0177
Epoch 19/50
 - 0s - loss: 0.0187 - val_loss: 0.0165
Epoch 20/50
 - 0s - loss: 0.0174 - val_loss: 0.0174
Epoch 21/50
 - 0s - loss: 0.0167 - val_loss: 0.0176
Epoch 22/50
 - 0s - loss: 0.0160 - val_loss: 0.0184
Epoch 23/50
 - 0s - loss: 0.0164 - val_loss: 0.0163
Epoch 24/50
 - 0s - loss: 0.0148 - val_loss: 0.0204
Epoch 25/50
 - 0s - loss: 0.0143 - val_loss: 0.0179
Epoch 26/50
 - 0s - loss: 0.0137 - val_loss: 0.0191
Epoch 27/50
 - 0s - loss: 0.0132 - val_loss: 0.0191
Epoch 28/50
 - 0s - loss: 0.0129 - val_loss: 0.0173
Epoch 29/50
 - 0s - loss: 0.0122 - val_loss: 0.0185
Epoch 30/50
 - 0s - loss: 0.0118 - val_loss: 0.0162
Epoch 31/50
 - 0s - loss: 0.0113 - val_loss: 0.0169
Epoch 32/50
 - 0s - loss: 0.0109 - val_loss: 0.0160
```

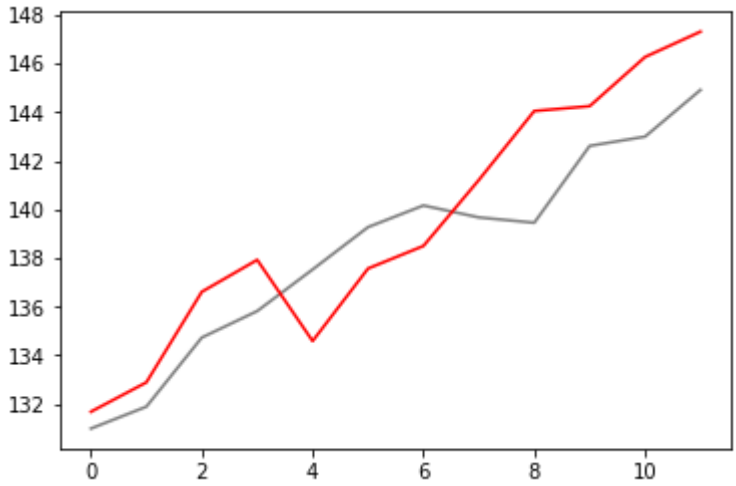
```
Epoch 33/50
 - 0s - loss: 0.0107 - val_loss: 0.0155
Epoch 34/50
 - 0s - loss: 0.0104 - val_loss: 0.0149
Epoch 35/50
 - 0s - loss: 0.0103 - val_loss: 0.0148
Epoch 36/50
 - 0s - loss: 0.0105 - val_loss: 0.0146
Epoch 37/50
 - 0s - loss: 0.0121 - val_loss: 0.0131
Epoch 38/50
 - 0s - loss: 0.0127 - val_loss: 0.0177
Epoch 39/50
 - 0s - loss: 0.0123 - val_loss: 0.0127
Epoch 40/50
 - 0s - loss: 0.0120 - val_loss: 0.0189
Epoch 41/50
 - 0s - loss: 0.0112 - val_loss: 0.0128
Epoch 42/50
 - 0s - loss: 0.0107 - val_loss: 0.0179
Epoch 43/50
 - 0s - loss: 0.0104 - val_loss: 0.0122
Epoch 44/50
 - 0s - loss: 0.0097 - val_loss: 0.0152
Epoch 45/50
 - 0s - loss: 0.0102 - val_loss: 0.0120
Epoch 46/50
 - 0s - loss: 0.0099 - val_loss: 0.0137
Epoch 47/50
 - 0s - loss: 0.0114 - val_loss: 0.0120
Epoch 48/50
 - 0s - loss: 0.0119 - val_loss: 0.0167
Epoch 49/50
 - 0s - loss: 0.0108 - val_loss: 0.0120
Epoch 50/50
 - 0s - loss: 0.0104 - val_loss: 0.0161
```



In [69]:
```
#make a prediction
yhat = model.predict(test_X)
test_X = test_X.reshape((test_X.shape[0], test_X.shape[2]))
# invert scaling for forecast
inv_yhat = concatenate((yhat, test_X[:, 1:]), axis=1)
inv_yhat = scaler.inverse_transform(inv_yhat)
inv_yhat = inv_yhat[:,0]
# invert scaling for actual
test_y = test_y.reshape((len(test_y), 1))
inv_y = concatenate((test_y, test_X[:, 1:]), axis=1)
inv_y = scaler.inverse_transform(inv_y)
inv_y = inv_y[:,0]
# calculate RMSE
rmse = math.sqrt(mean_squared_error(inv_y, inv_yhat))
print('Test RMSE: %.3f' % rmse)
```

```
Test RMSE: 2.352
```

In [70]:
```
#Ploting Fitted and Actual values
plt.plot(inv_y,color="gray")
plt.plot(inv_yhat,color="red")
plt.show()
```



In [ ]:

In [2]:
```
import pandas as pd
import numpy as np
import seaborn as sns
from matplotlib import pyplot as plt
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.graphics.tsaplots import plot_pacf
from pandas import datetime
from statsmodels.tsa.arima_model import ARIMA
from statsmodels.tsa.arima_model import ARMA
from sklearn.metrics import mean_squared_error
from statsmodels.tsa.seasonal import seasonal_decompose
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:8: FutureWarning: T
he pandas.datetime class is deprecated and will be removed from pandas in a future v
ersion. Import from datetime module instead.
```

In [3]:
```
import warnings
warnings.filterwarnings("ignore")
```

In [5]:
```
############################################ ML-Time Series Tempate #############
####################################
# Input Arguments file: File name with path (Eg: 'D:/data/dataread.csv'), f_type: Fil
e Type (Eg: 'csv',default csv)
def file_todataframe(file,f_type):
    if f_type == 'csv':
        return pd.read_csv(file)
    elif f_type == 'excel':
        return pd.read_excel(file)
    elif f_type == 'json':
        return pd.read_json(file)

# This function display shape, data type, data near head and tail of given data fram.
# Input Arguments df: dataframe, n: No f data points to display
def df_details(d_f,n):
    print('Data Types of Column: \n',d_f.dtypes)
    print('\n Size of Datarame: ',d_f.shape)
    print('\n Top and bottom ',n,' rows: \n')
    display(d_f.head(n).append(d_f.tail(n)))

# This function give deatials for missing values in data
# Input Arguments df: dataframe
def miss_ch(d_f):
    print('Available data with no nulls: ', d_f.dropna().shape[0])
    display('Deatils of Null values column wise',d_f.isnull().sum())


# Function for Canging column type
# Input Arguments df: dataframe, col_int: Columns of intrest (Eg: ['Sale','Custome
r'],['all'] default 'all'),
# dtyp: New data types of coloumn default int
def col_dtype(d_f,col_int,dtyp = int):
    d_f = deep_copy(d_f)
    av_fun = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64','int','floa
t','str','category']
    col_nam = d_f.columns
    #Checking Parameter Column names
    if not all(i in col_nam for i in col_int):
        print("Invalid column name")
        return
    #Checking Parameter available function
    if not dtyp in av_fun:
        print("Invalid data type")
        return

    d_f[col_int] = d_f[col_int].astype(dtyp,errors='ignore')
    return d_f

def data_num(d_f,col_int = 'all',func = 'all',scat = None):
    from scipy.stats import shapiro
    from statsmodels.graphics.gofplots import qqplot
    av_func = ['hist','boxplot','scatter','describe','normality']
    numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
    col_nam = d_f.select_dtypes(include=numerics).columns
    #Checking Parameter Column names
    if col_int == 'all':
        col_int = col_nam
    elif not all(i in col_nam for i in col_int):
        print("Invalid column name")
        return

    #Checking Parameter available function
    if func == 'all':
        func = av_func
    elif not all(i in av_gp for i in graph):
```

```
        print("Invalid Graph type, select 'distplot','boxpot','scatterplot','describe','normality'")
        return
    if scat is None:
        scat = d_f.columns[0]


    for fn in func:
        if fn == 'describe':
            display("Statictical Details",d_f[col_int].describe())
        else:
            for col in col_int:
                if fn == 'normality':
                    #qqplot(df[col])
                    print("Normality Test for: ",col)
                    stat, p = shapiro(d_f[col])
                    if p > 0.05:
                        print('Sample looks Gaussian. Statistics=%.3f, p=%.3f'% (stat, p))
                    else:
                        print('Sample does not look Gaussian. Statistics=%.3f, p=%.3f'% (stat, p))
                elif fn == 'scatter':
                    display(col)
                    plt.scatter(d_f[col],d_f[scat])
                    plt.show()
                else:
                    display(col)
                    getattr(plt, fn)(d_f[col])
                    plt.show()

## Function to return SL problem by taking a Time series data
# lag define the the how many and how much shifts to be included, for lag = [1,3] : will include one t-1 and t-3 data columns
def ts2sl(d_f,col, lag=[1,2]):
    d_f = d_f.copy(deep=True)
    x = 1
    for i in lag:
        xst = 'x'+ str(x)
        d_f[xst] = d_f[col].shift(i)
        x = x+1
    xst = 'x'+ str(x)
    d_f.rename(columns = {col: 'y'}, inplace = True)
    return d_f

# This function fill missing values
def fill_miss(d_f,col_int = 'all',metd = None):
    col_nam = d_f.columns
    av_method = ['bfill', 'pad', 'ffill', 'linear','mean']

    if not col_int in col_nam:
        print("Invalid column name")
        return d_f
    if metd == None:
        return d_f
    elif (metd == 'mean'):
        d_f[col_int] = d_f[col_int].fillna(value=d_f[col_int].mean())
        return d_f
    elif metd == 'linear':
        d_f[col_int] = d_f[col_int].interpolate(method = 'linear')
        return  d_f
    elif  metd in av_method:
        d_f[col_int] = d_f[col_int].fillna(method = metd)
        return d_f
    else:
```

```
        print("Invalid fill type")
        return d_f
```

Project Part#1

Using 20 different features forecasting inflation

Wrangling through data

In [72]:
```
# Reading File
df =  file_todataframe('CPI_Urban.xls','excel')
```

In [13]:
```
df_details(df,5)
```

```
Data Types of Column:
 Date     datetime64[ns]
CPI              float64
ER               float64
M0               float64
M2               float64
RES              float64
REM              float64
IPLSM            float64
TB               float64
Disr             float64
OIL              float64
USIPI            float64
PSC              float64
PSB              float64
LR               float64
REER             float64
WPI              float64
WCPI             float64
WP                 int64
Wfood            float64
dtype: object

 Size of Datarame:  (378, 20)

 Top and bottom  5  rows:
```

	Date	CPI	ER	M0	M2	RES	REM	IPLSM
0	1991-01-01	13.238078	22.129630	1.573830e+05	3.470580e+05	1451.000000	138.560000	49.270033
1	1991-02-01	13.238078	22.205375	1.603250e+05	3.490440e+05	1451.000000	123.950000	45.954554
2	1991-03-01	13.238078	22.559057	1.682910e+05	3.506650e+05	1451.000000	114.750000	47.200370
3	1991-04-01	13.238078	23.250155	1.682260e+05	3.527060e+05	1451.000000	115.500000	39.002094
4	1991-05-01	13.238078	23.742007	1.721270e+05	3.551370e+05	1451.000000	125.210000	38.017497
373	2022-02-01	NaN	171.900000	8.676084e+06	2.531512e+07	24534.193682	3040.084678	164.390900
374	2022-03-01	NaN	173.100000	8.676084e+06	2.531512e+07	24926.740781	3070.485525	167.913500
375	2022-04-01	NaN	174.300000	8.676084e+06	2.531512e+07	25325.568633	3101.190380	149.207000
376	2022-05-01	NaN	175.600000	8.676084e+06	2.531512e+07	25730.777731	3132.202284	144.323900
377	2022-06-01	NaN	176.800000	8.676084e+06	2.531512e+07	26142.470175	3163.524306	138.987100

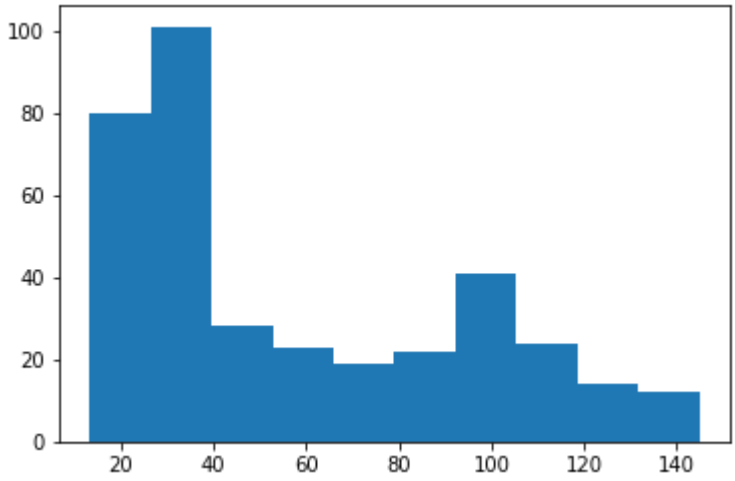
In [14]:
```
# Checking for missing values
miss_ch(df)
```

```
Available data with no nulls:  364

'Deatils of Null values column wise'

Date      0
CPI      14
ER        0
M0        0
M2        0
RES       0
REM       0
IPLSM     0
TB        0
Disr      0
OIL       0
USIPI     0
PSC       0
PSB       0
LR        0
REER      0
WPI       0
WCPI      0
WP        0
Wfood     0
dtype: int64
```

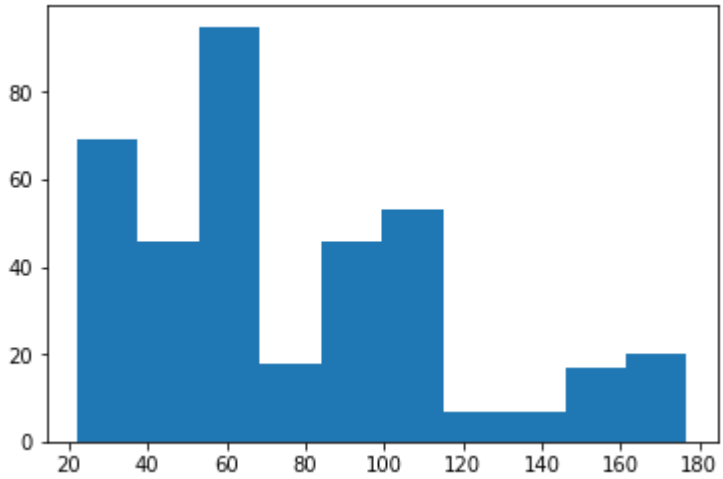
No missing values in data as CPI is the variable need to be forcasted

In [15]:
```
# Analysing data using template function by ploting the series
data_num(df,col_int = 'all',func = 'all',scat = None)
```

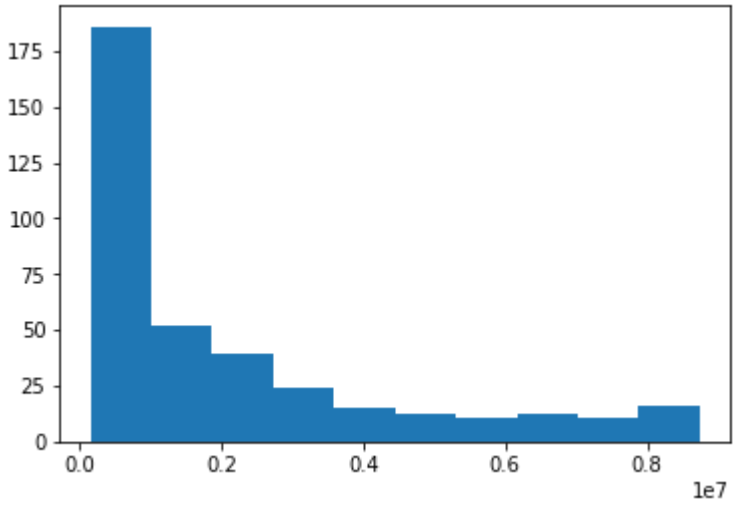
'CPI'


100
80
60
40
20
0
20
40
60
80
100
120
140

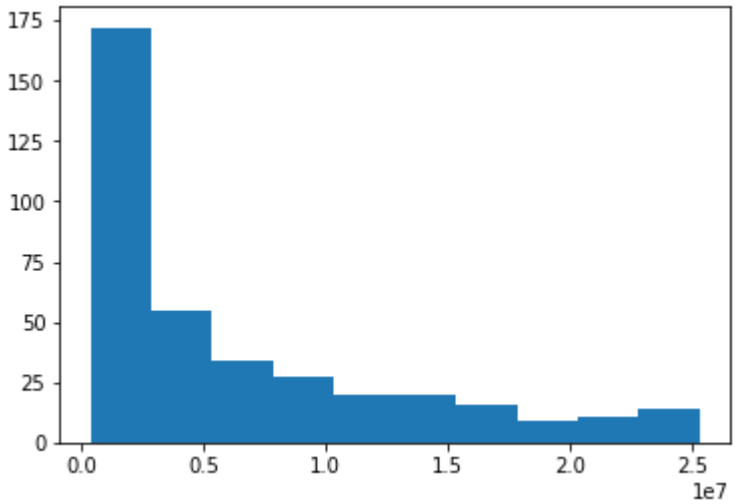
'ER'


80
60
40
20
0
20
40
60
80
100
120
140
160
180

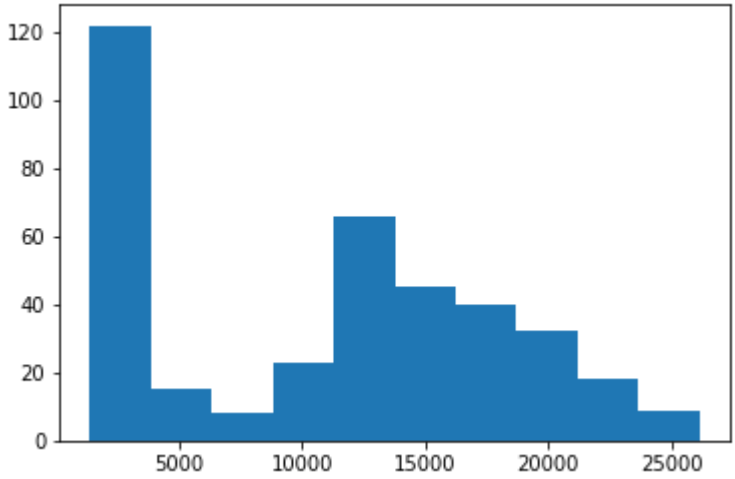
'M0'


175
150
125
100
75
50
25
0
0.0
0.2
0.4
0.6
0.8
1e7

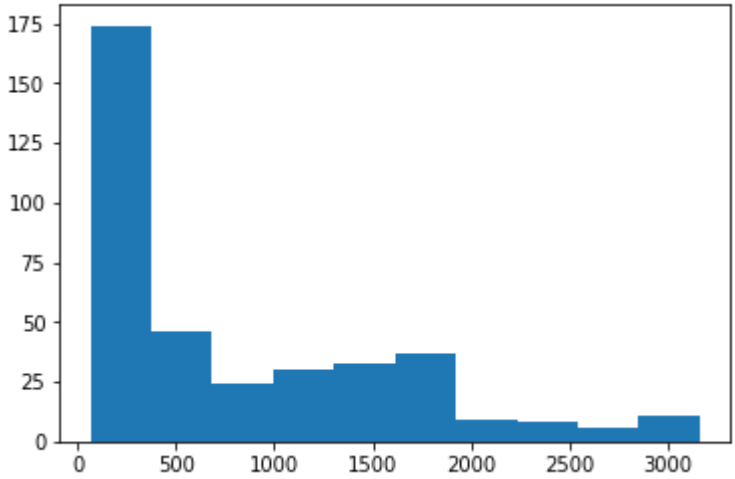
'M2'



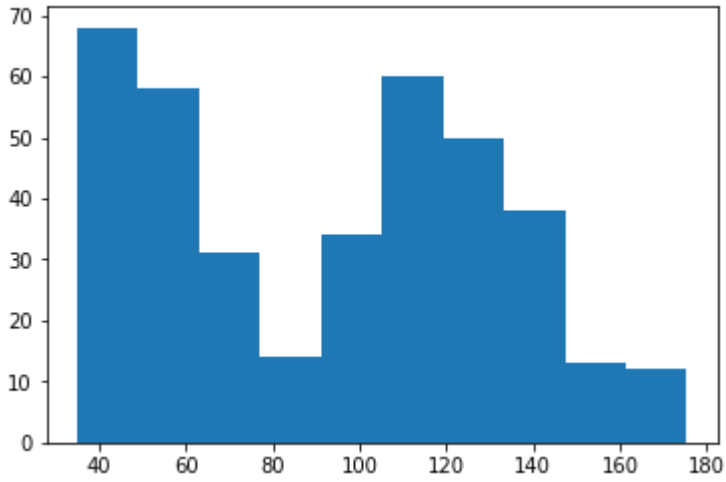
'RES'



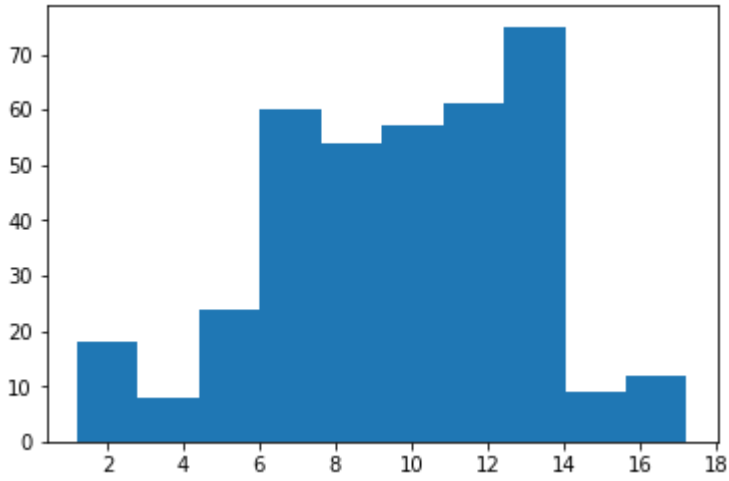
'REM'



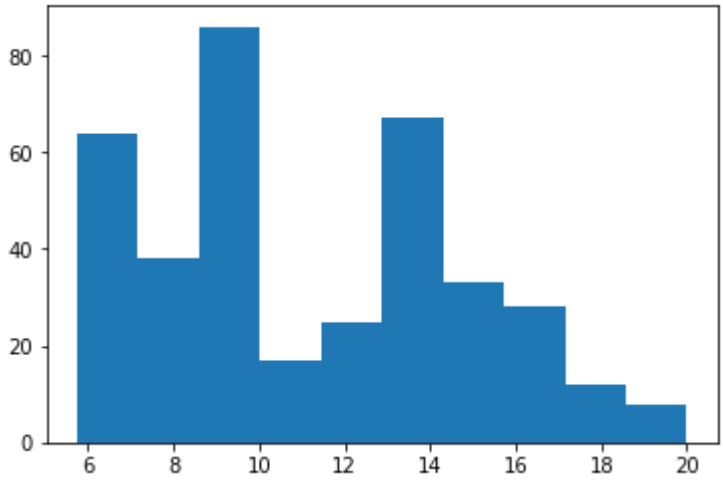
'IPLSM'


70
60
50
40
30
20
10
0
40
60
80
100
120
140
160
180

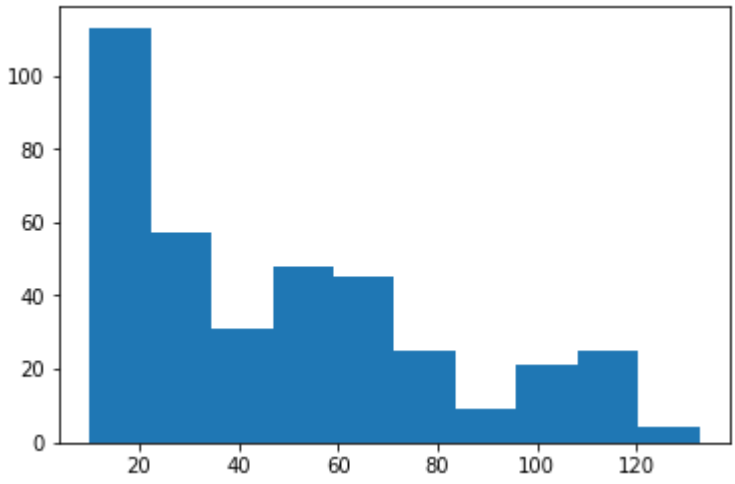
'TB'


70
60
50
40
30
20
10
0
2
4
6
8
10
12
14
16
18

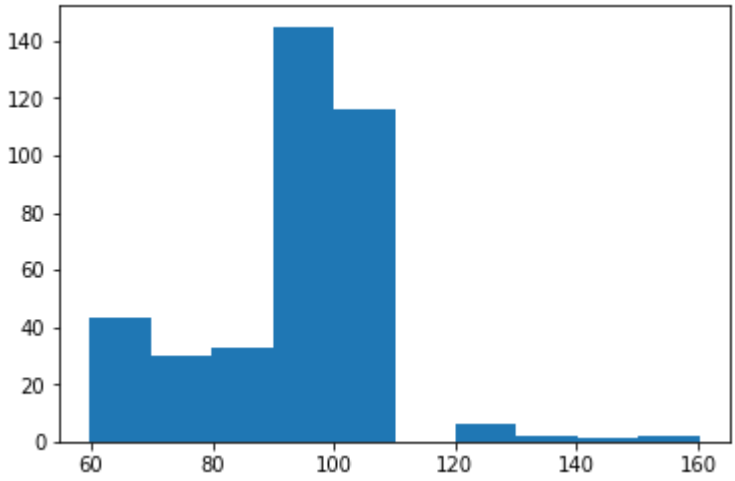
'Disr'


80
60
40
20
0
6
8
10
12
14
16
18
20

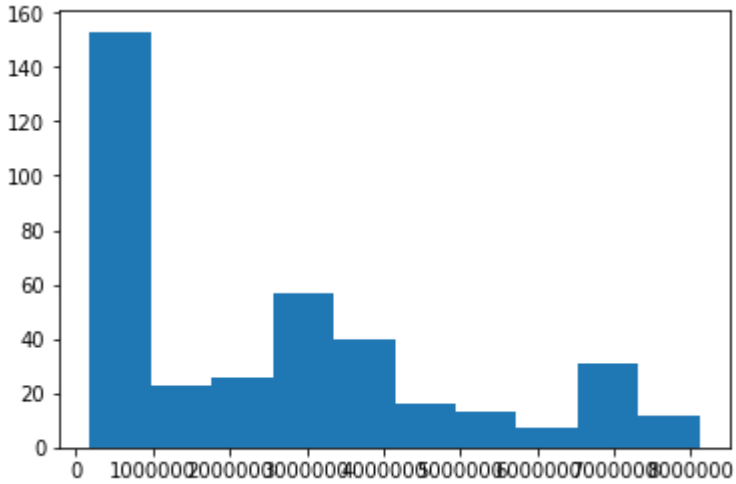
'OIL'



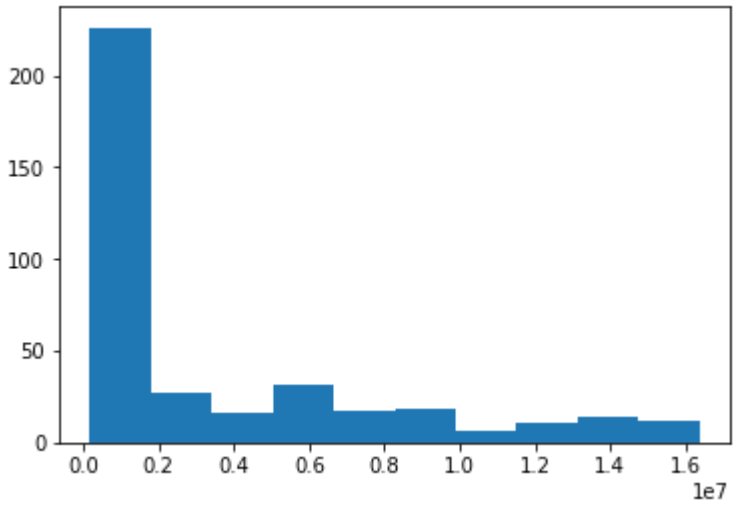
'USIPI'



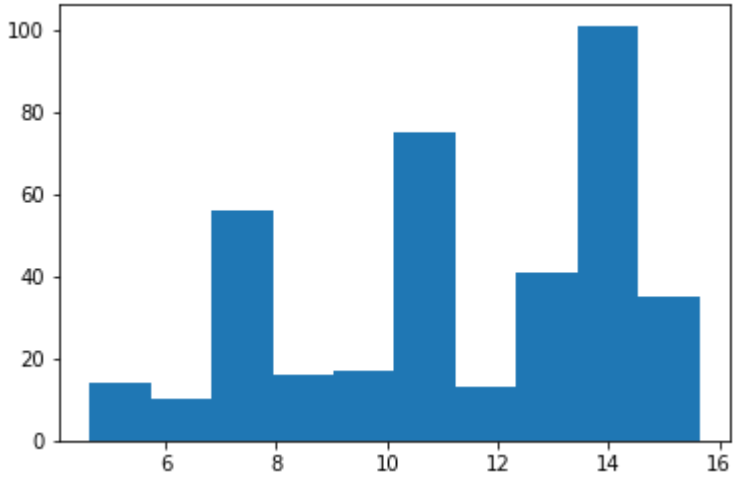
'PSC'



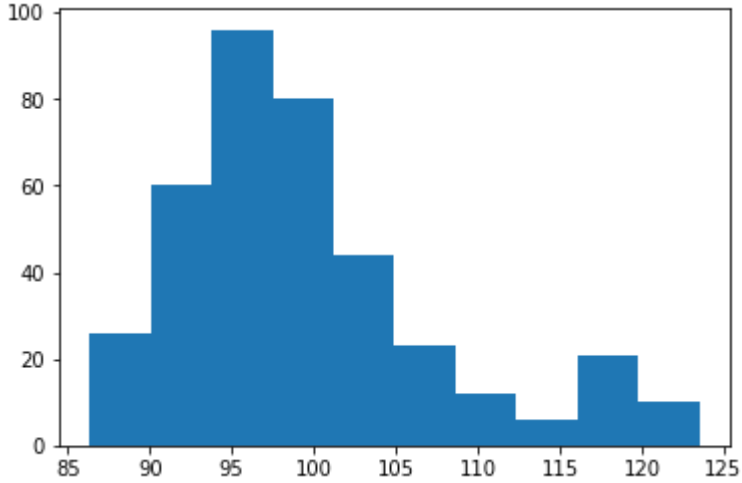
'PSB'


200
150
100
50
0
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
1e7

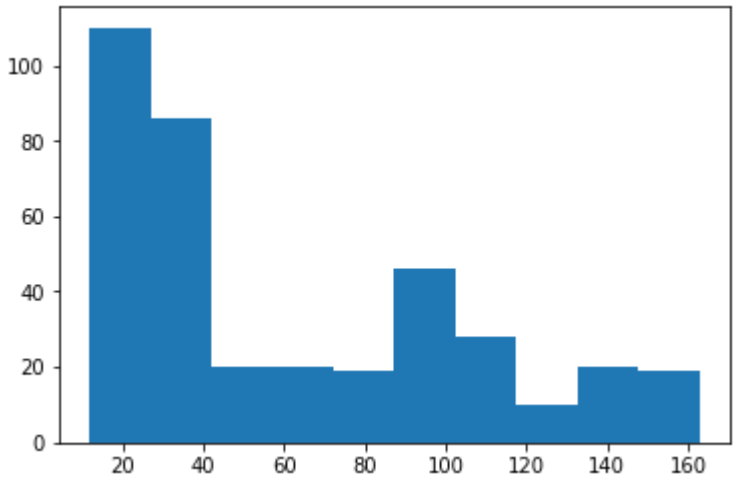
'LR'


100
80
60
40
20
0
6
8
10
12
14
16

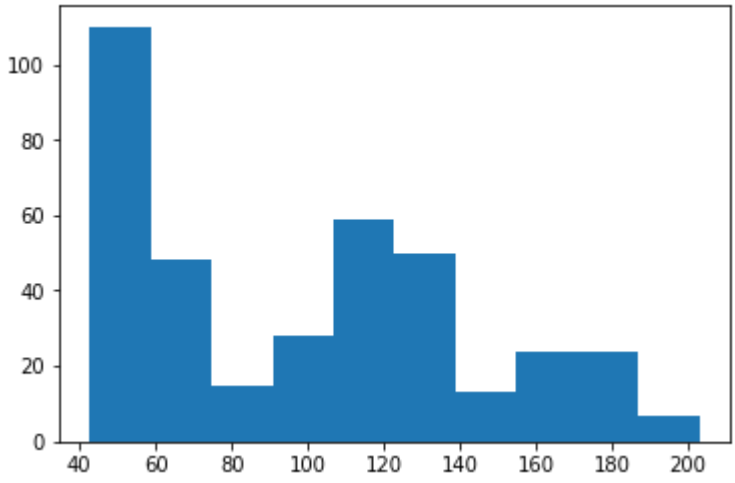
'REER'


100
80
60
40
20
0
85
90
95
100
105
110
115
120
125

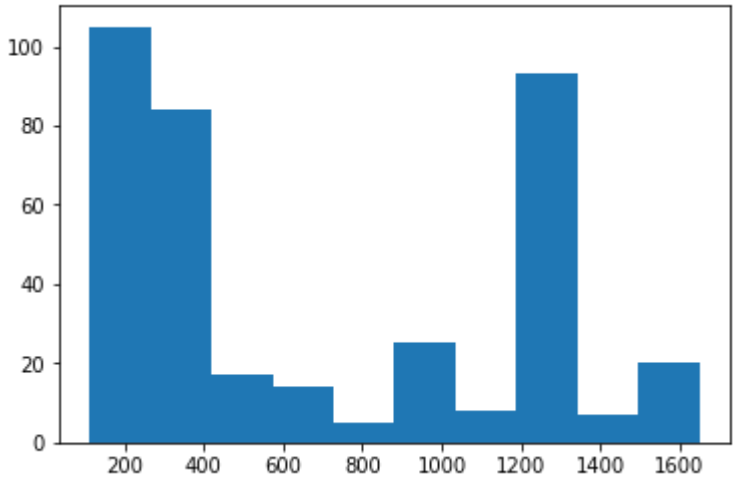
'WPI'



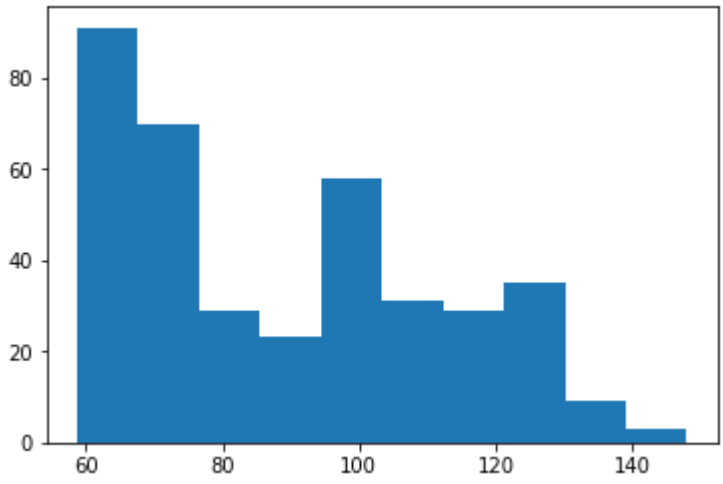
'WCPI'



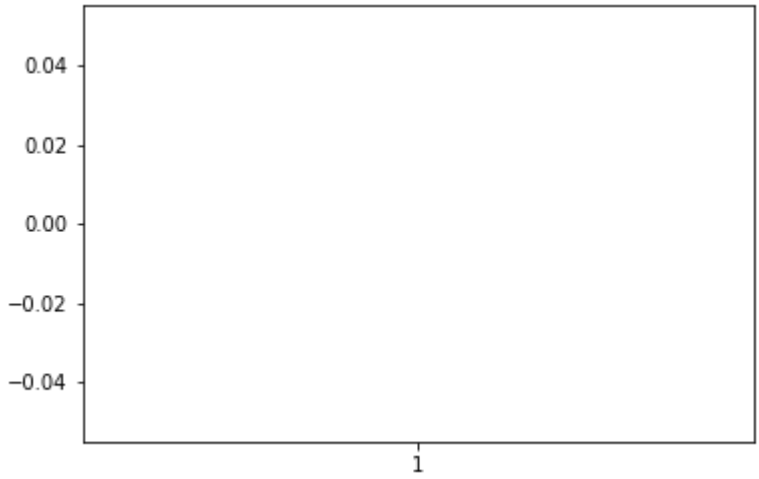
'WP'



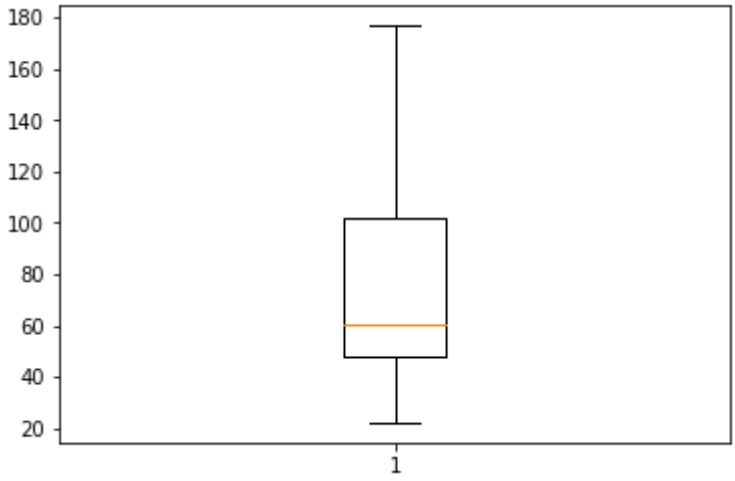
'Wfood'


80
60
40
20
0
60
80
100
120
140

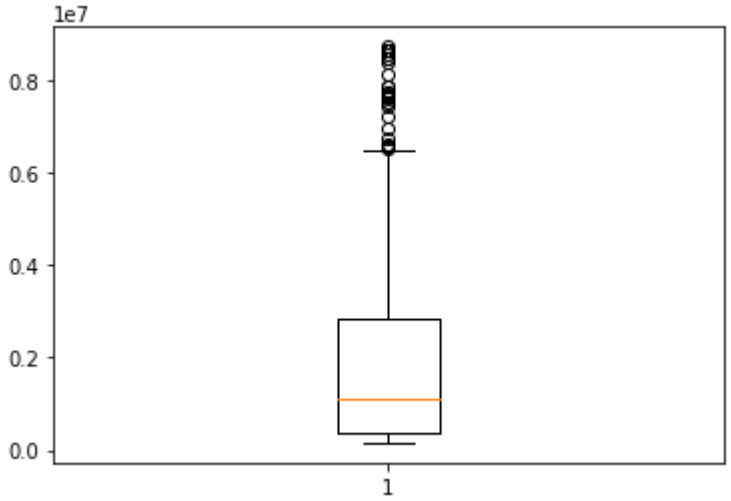
'CPI'


0.04
0.02
0.00
−0.02
−0.04
1

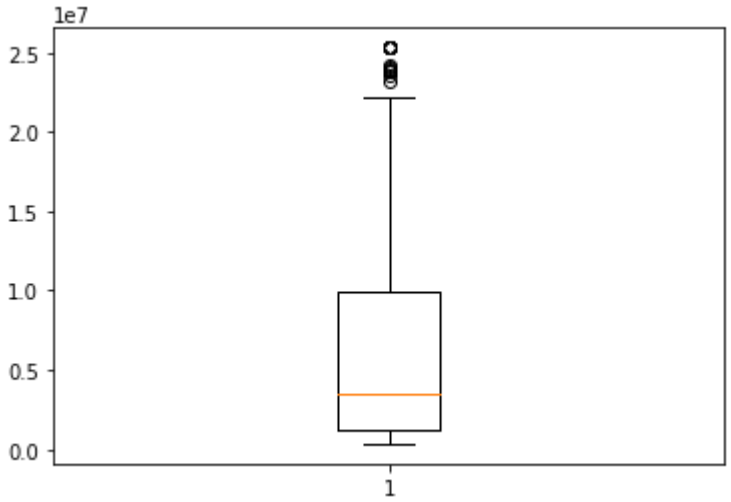
'ER'


180
160
140
120
100
80
60
40
20
1

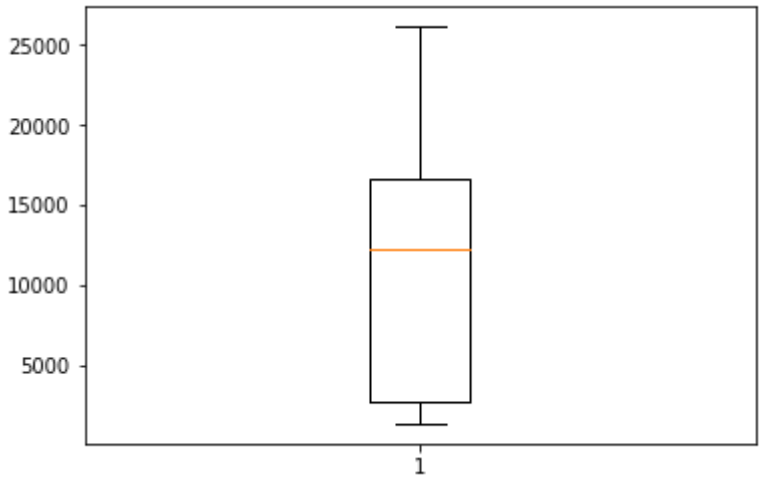
'M0'


1e7
0.8
0.6
0.4
0.2
0.0
1

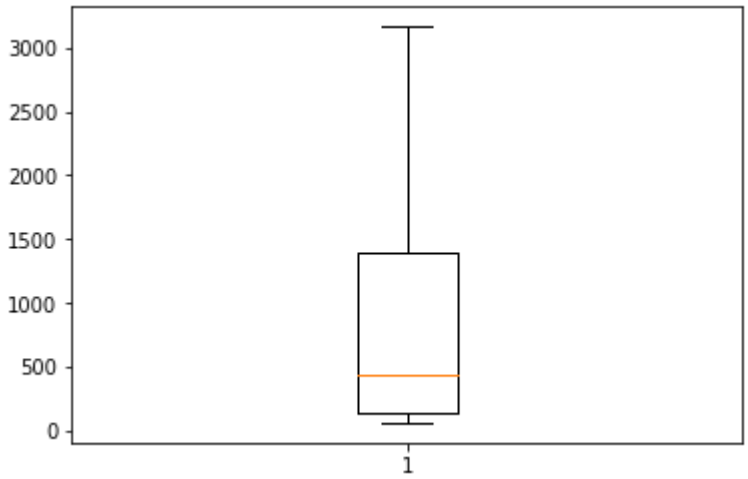
'M2'


1e7
2.5
2.0
1.5
1.0
0.5
0.0
1

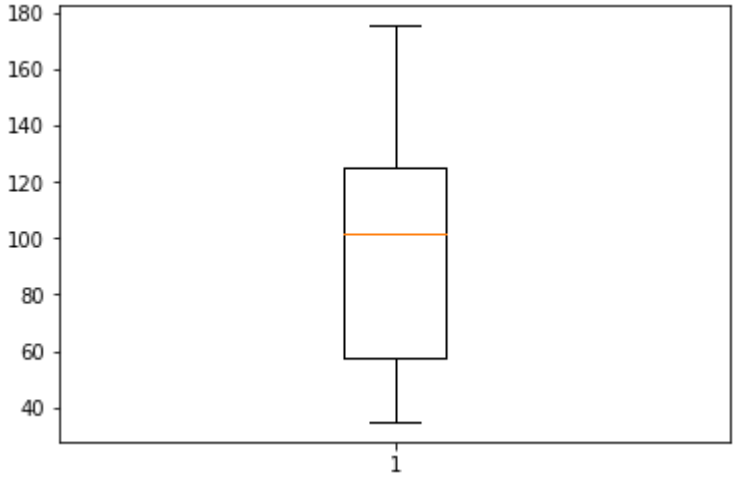
'RES'


25000
20000
15000
10000
5000
1

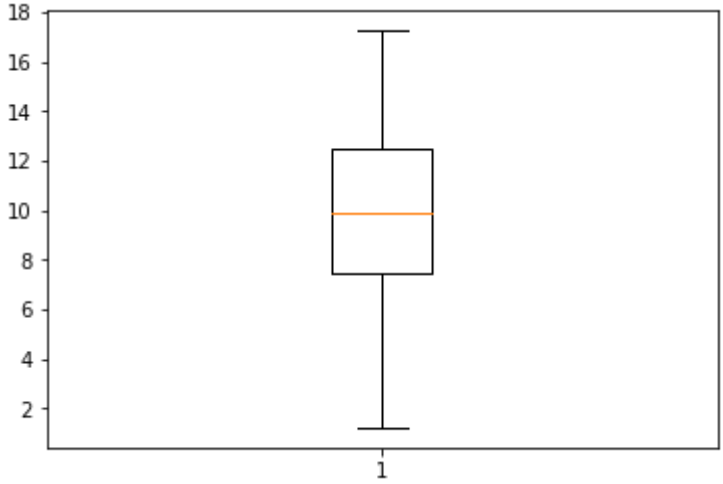
'REM'



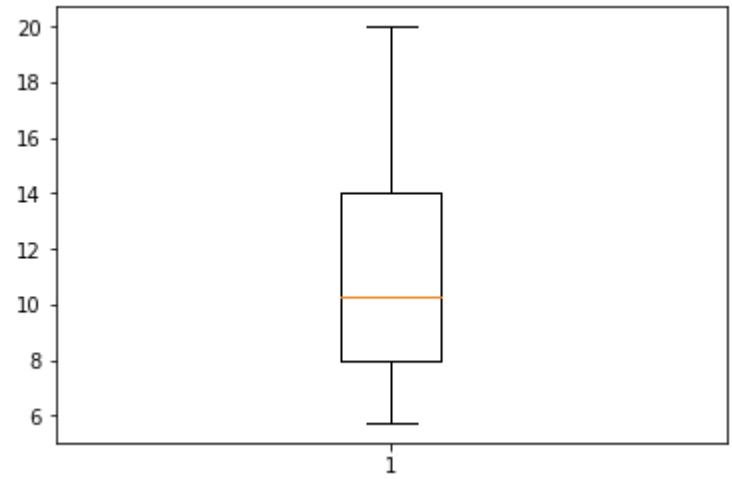
'IPLSM'



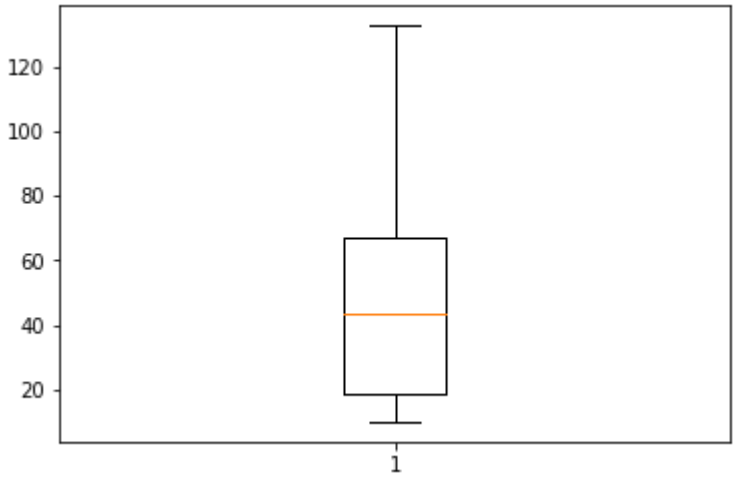
'TB'



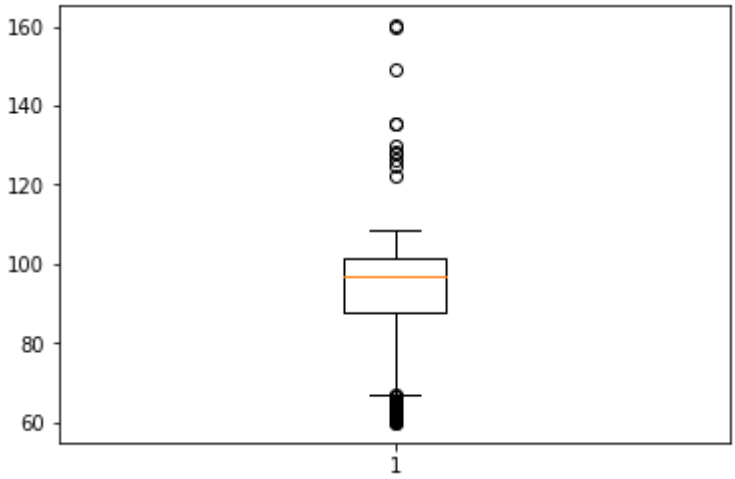
'Disr'



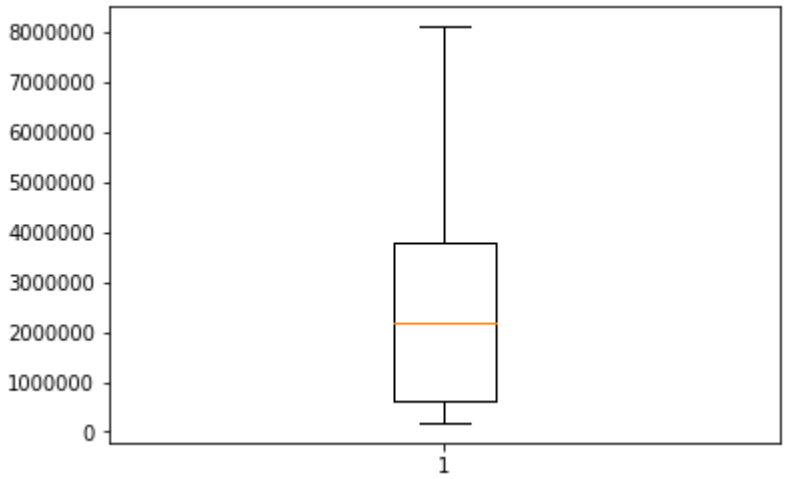
'OIL'



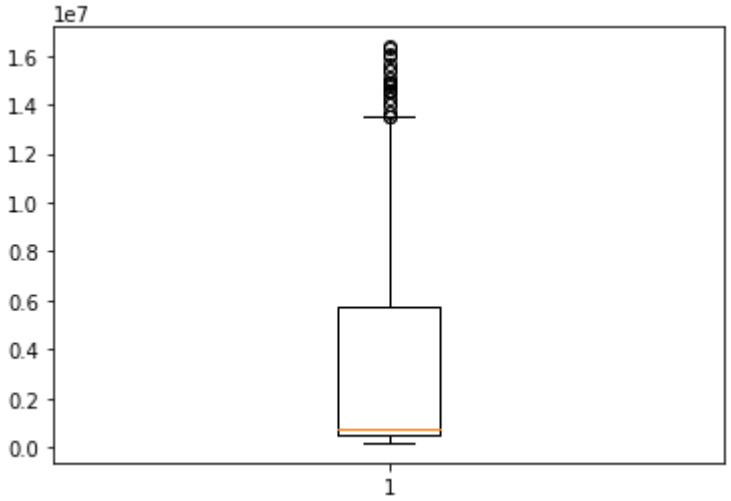
'USIPI'



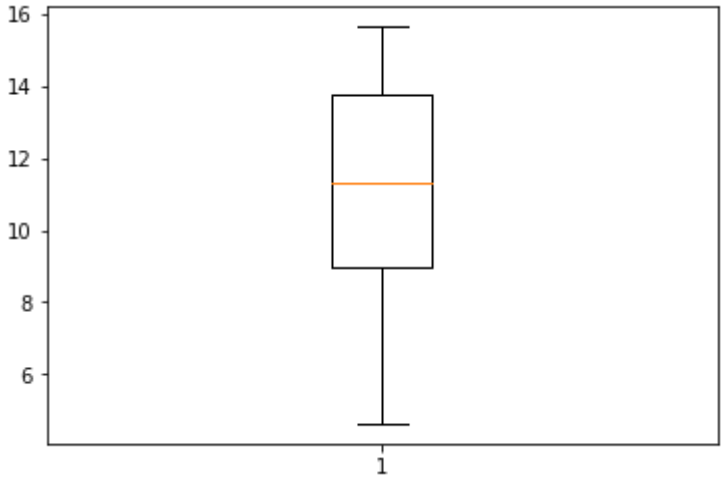
'PSC'


8000000
7000000
6000000
5000000
4000000
3000000
2000000
1000000
0
1

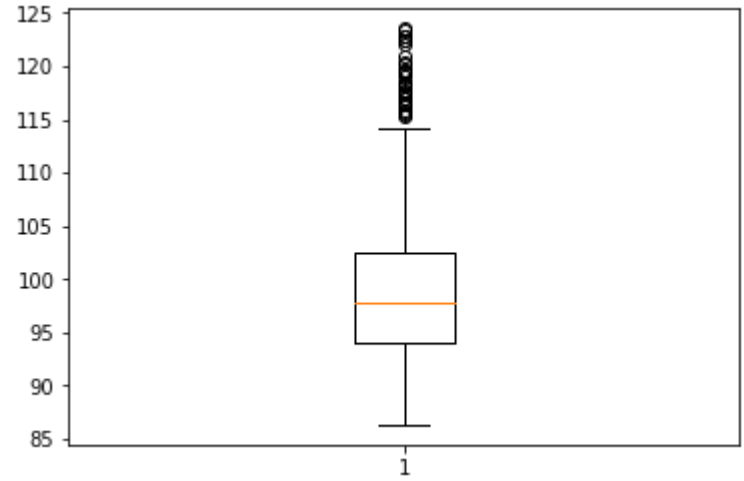
'PSB'


1e7
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
1

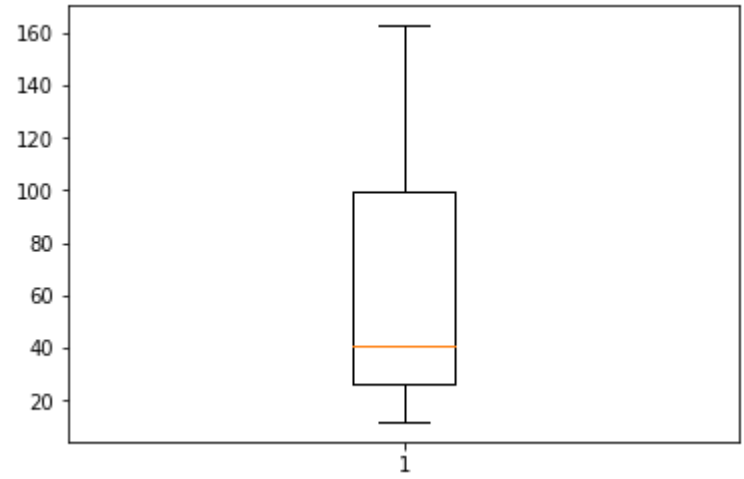
'LR'


16
14
12
10
8
6
1

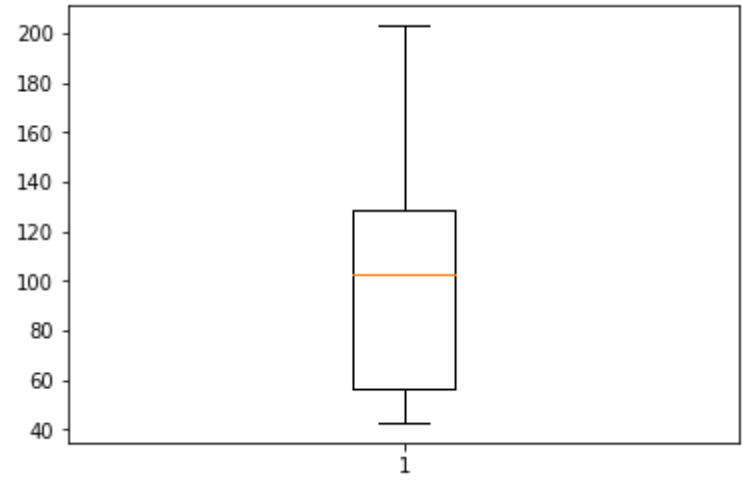
'REER'


125
120
115
110
105
100
95
90
85
1

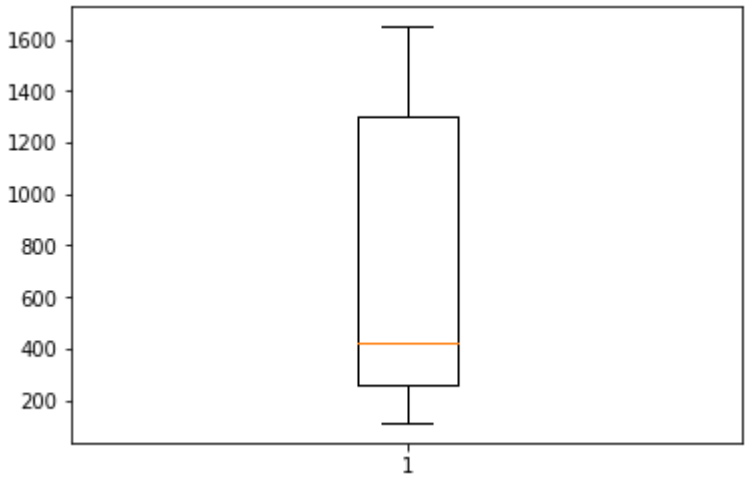
'WPI'


160
140
120
100
80
60
40
20
1

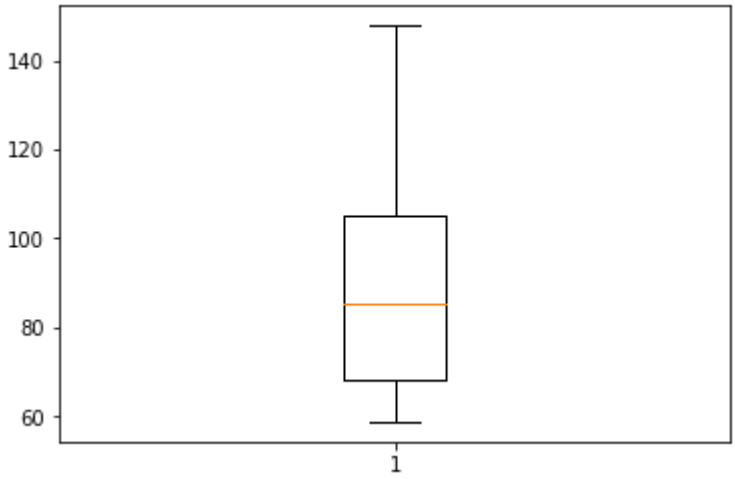
'WCPI'


200
180
160
140
120
100
80
60
40
1

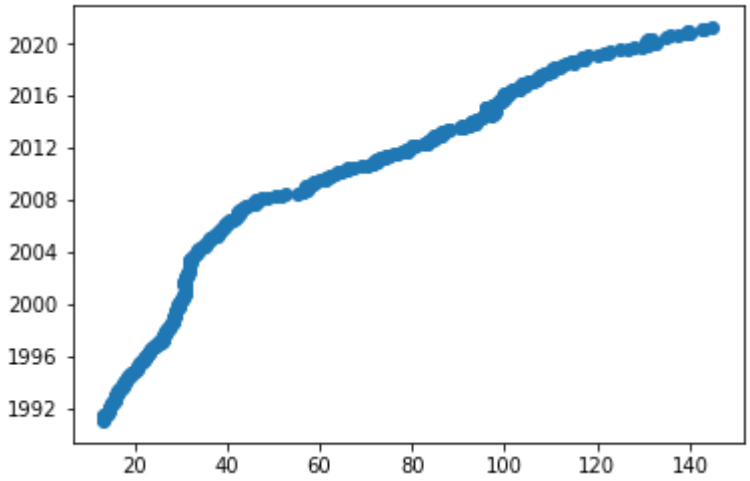
'WP'


1600
1400
1200
1000
800
600
400
200
1

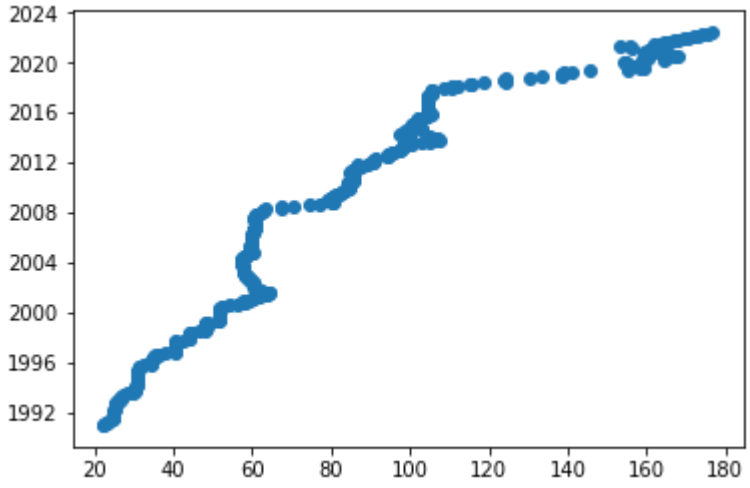
'Wfood'


140
120
100
80
60
1

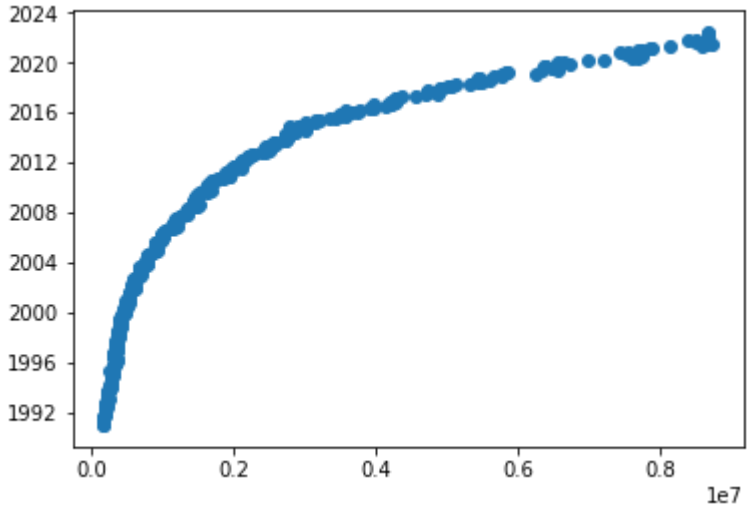
'CPI'


2020
2016
2012
2008
2004
2000
1996
1992
20
40
60
80
100
120
140

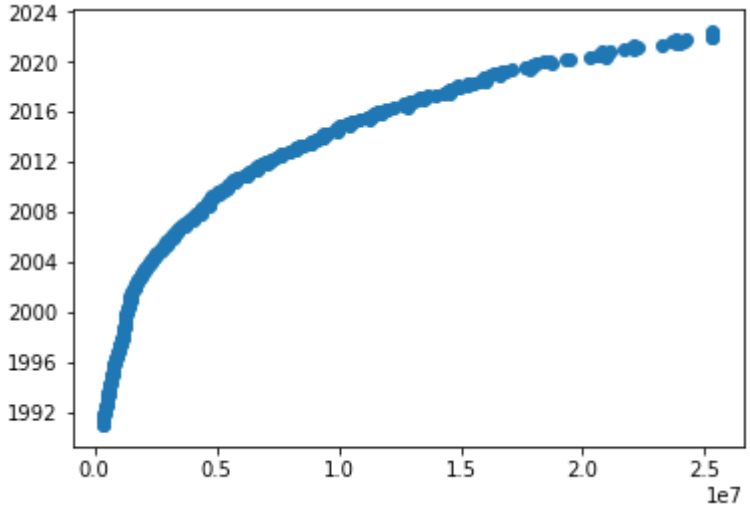
'ER'


2024
2020
2016
2012
2008
2004
2000
1996
1992
20
40
60
80
100
120
140
160
180

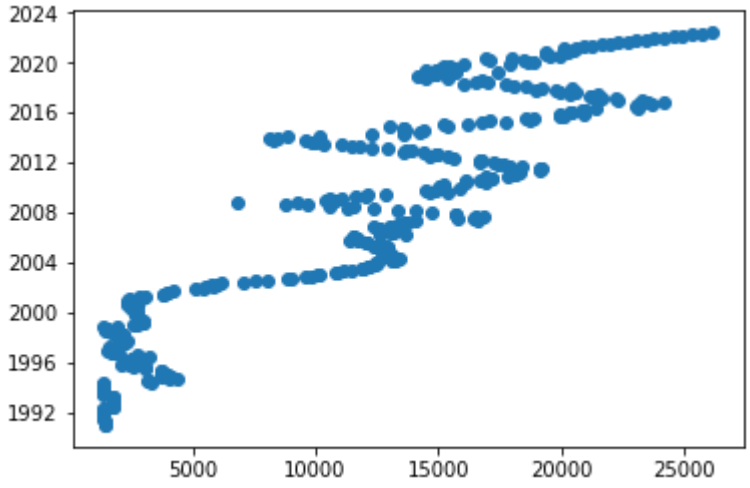
'M0'


2024
2020
2016
2012
2008
2004
2000
1996
1992
0.0
0.2
0.4
0.6
0.8
1e7

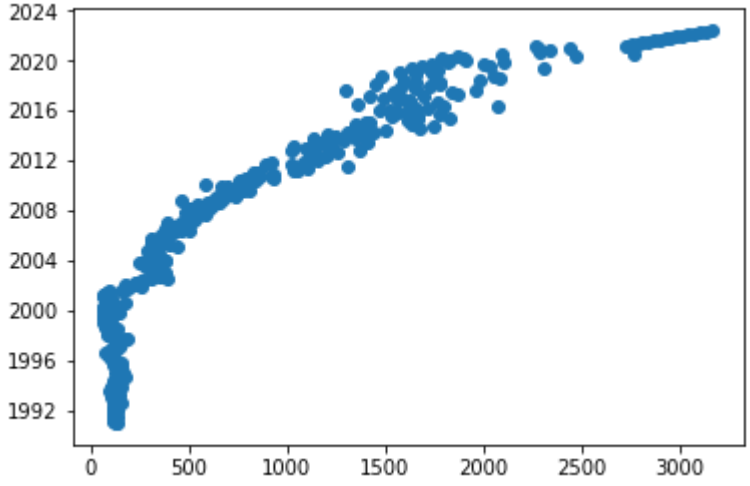
'M2'


2024
2020
2016
2012
2008
2004
2000
1996
1992
0.0
0.5
1.0
1.5
2.0
2.5
1e7

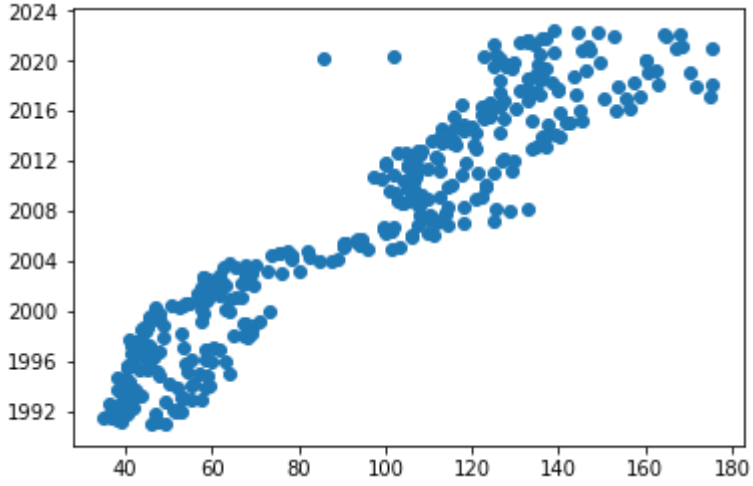
'RES'


2024
2020
2016
2012
2008
2004
2000
1996
1992
5000
10000
15000
20000
25000

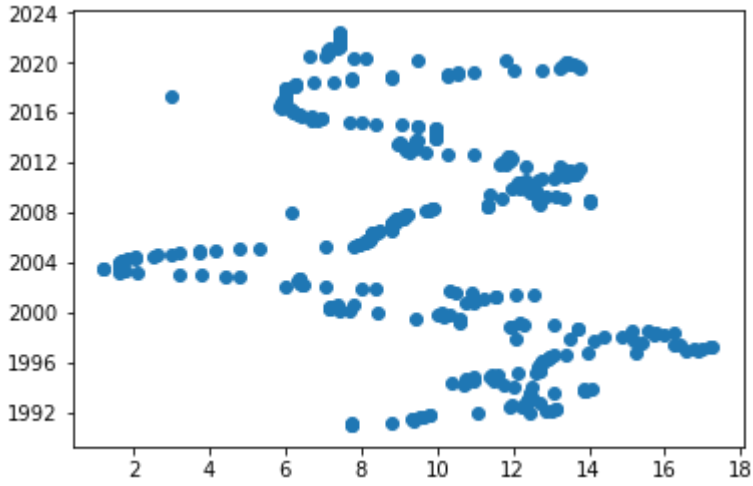
'REM'


2024
2020
2016
2012
2008
2004
2000
1996
1992
0
500
1000
1500
2000
2500
3000

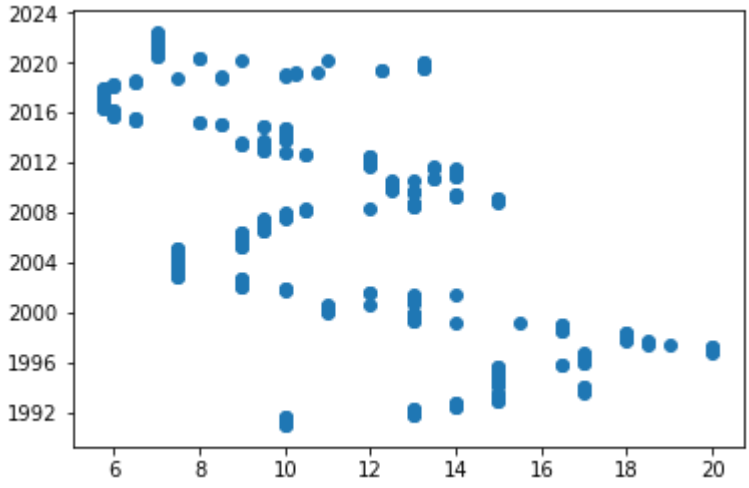
'IPLSM'


2024
2020
2016
2012
2008
2004
2000
1996
1992
40
60
80
100
120
140
160
180

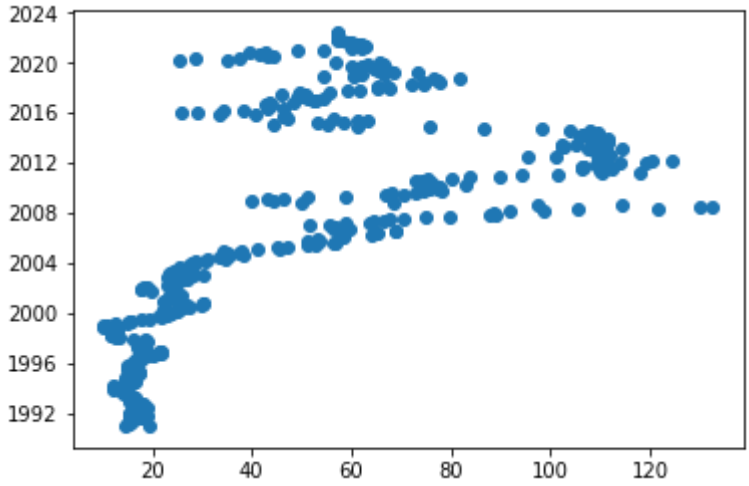
'TB'



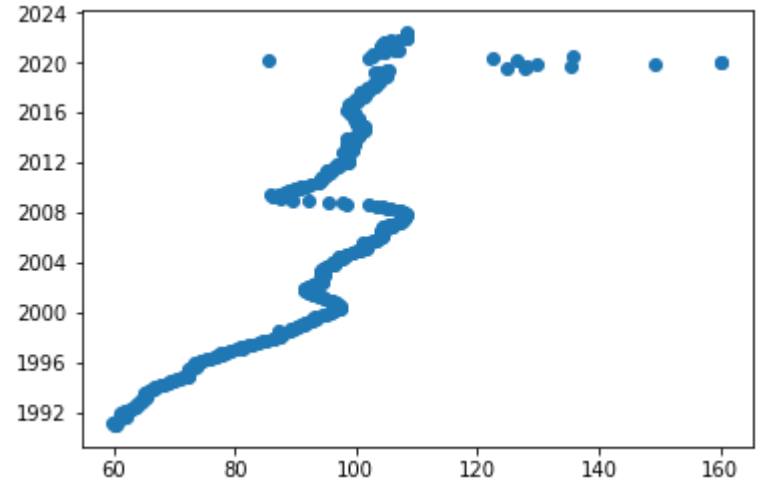
'Disr'



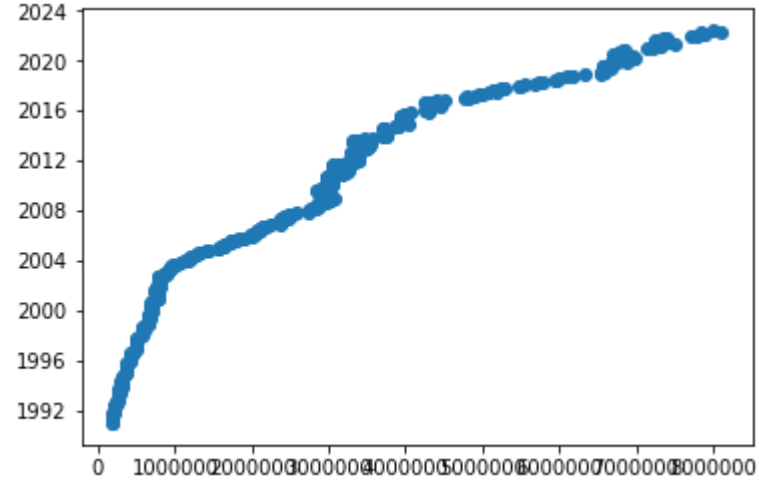
'OIL'



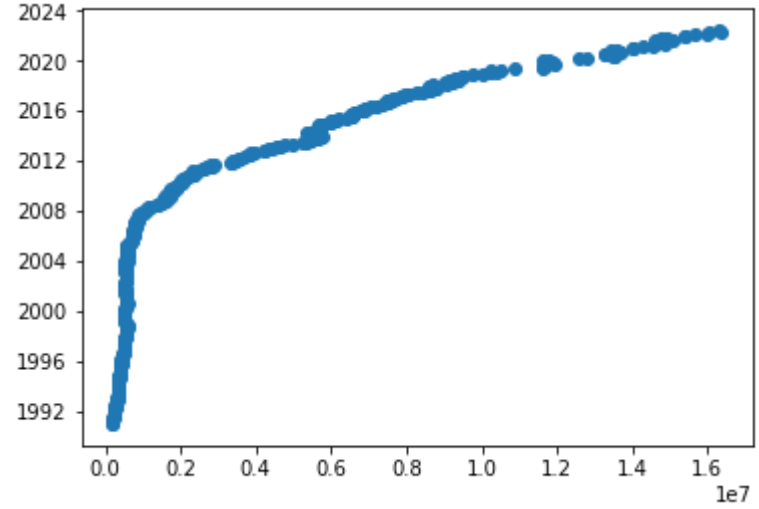
'USIPI'


2024
2020
2016
2012
2008
2004
2000
1996
1992
60
80
100
120
140
160

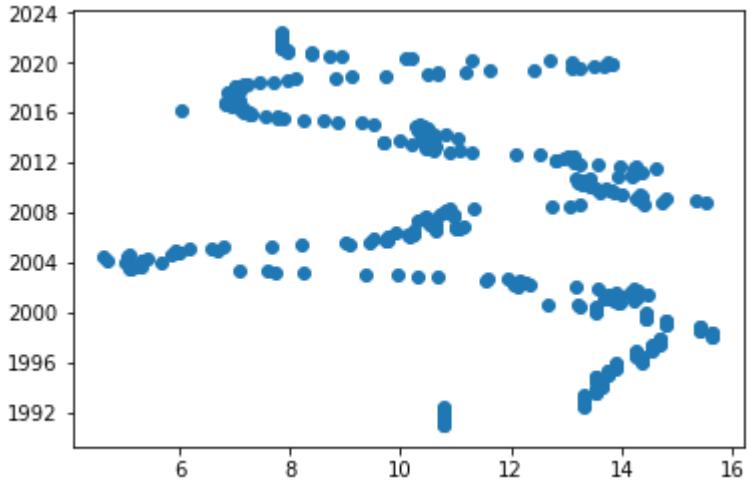
'PSC'


2024
2020
2016
2012
2008
2004
2000
1996
1992
0
1000000
2000000
3000000
4000000
5000000
6000000
7000000
8000000

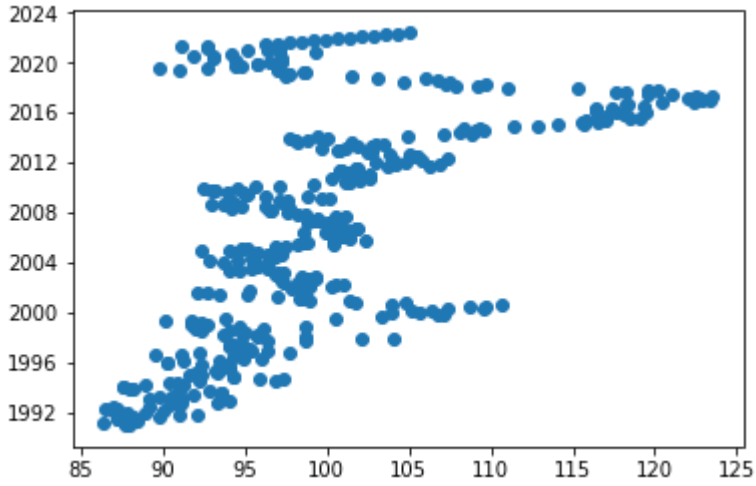
'PSB'


2024
2020
2016
2012
2008
2004
2000
1996
1992
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
1e7

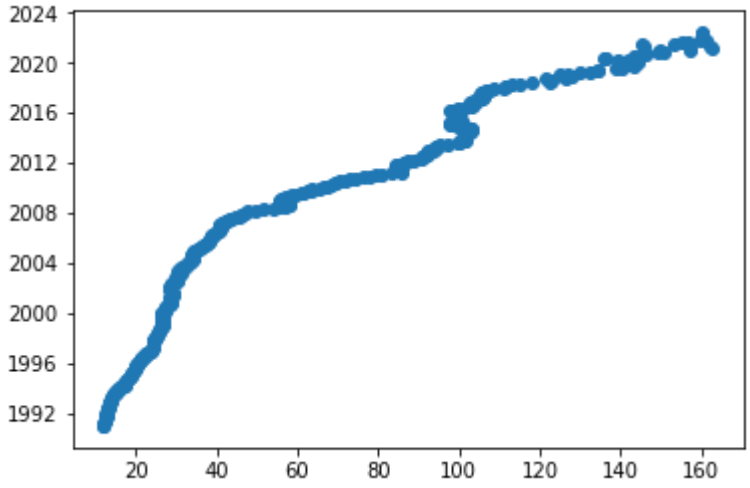
'LR'



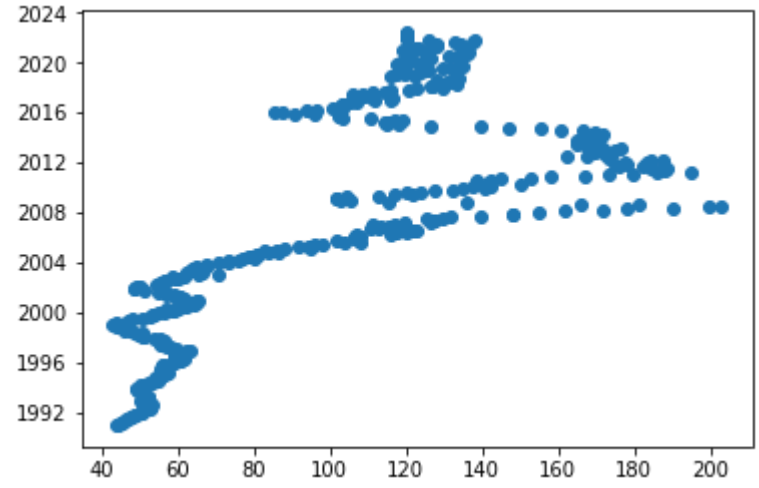
'REER'



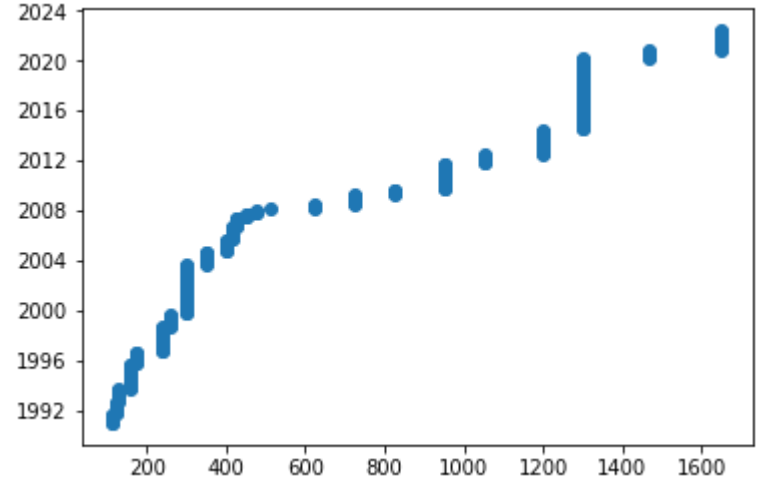
'WPI'



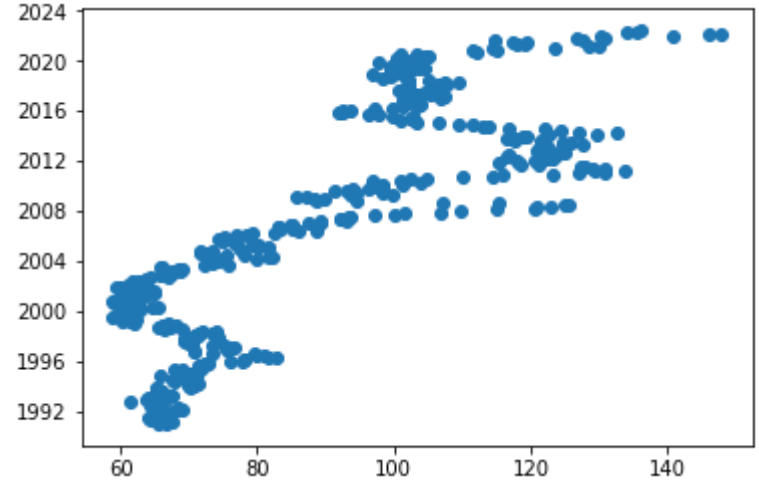
'WCPI'



'WP'



'Wfood'

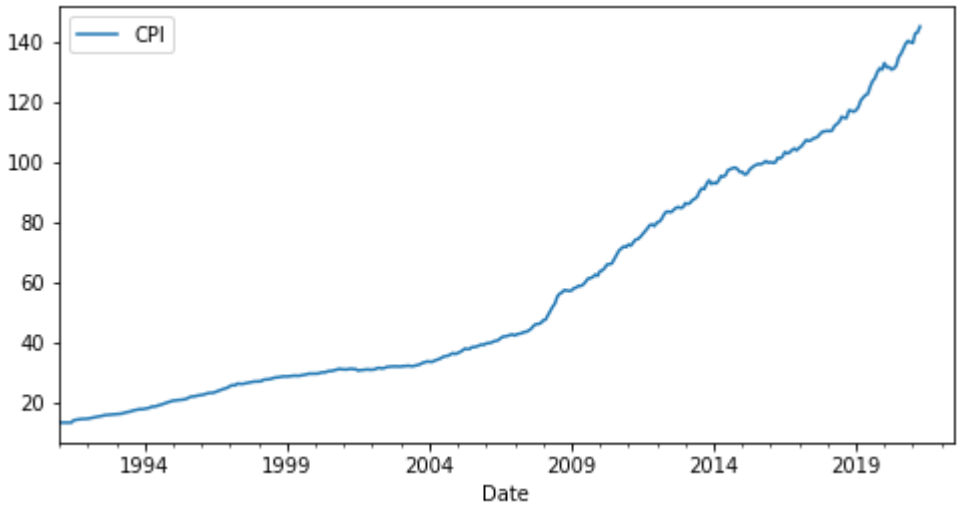


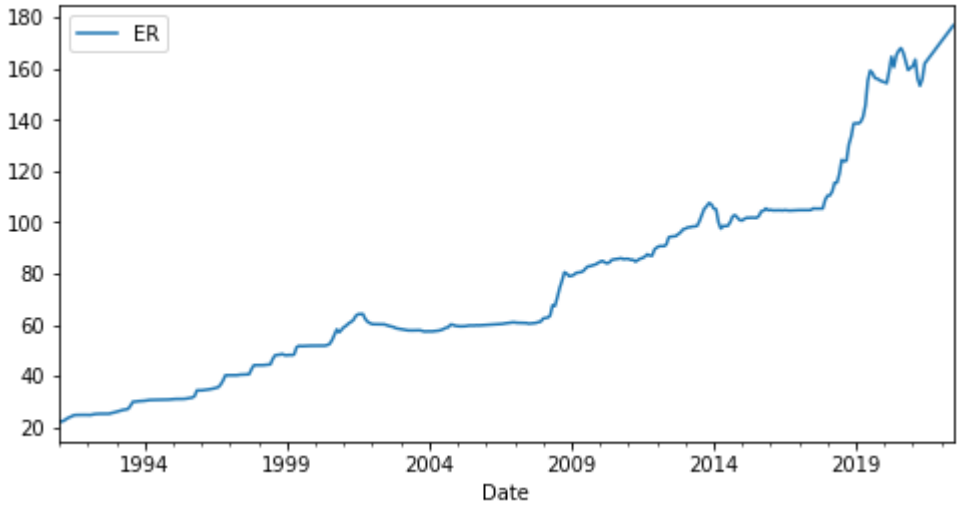
'Statictical Details'

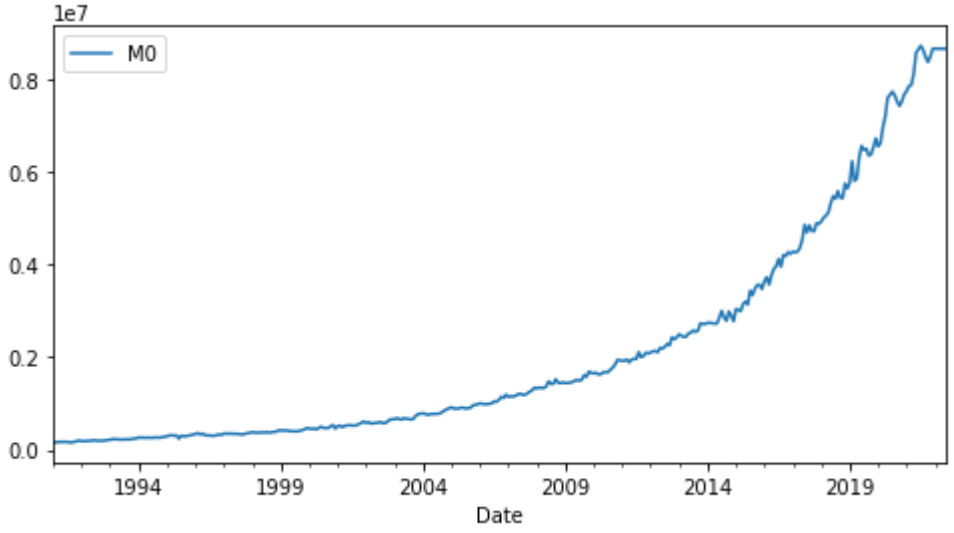
	CPI	ER	M0	M2	RES	REM	IPLSM	
count	364.000000	378.000000	3.780000e+02	3.780000e+02	378.000000	378.000000	378.000000	378.0
mean	56.986987	76.193293	2.120560e+06	6.403818e+06	10866.370398	806.636683	93.188547	9.7
std	37.195332	39.876103	2.354669e+06	6.741169e+06	7185.086582	793.234181	38.274582	3.4
min	13.238078	22.129630	1.573830e+05	3.470580e+05	1369.000000	64.140000	34.842674	1.2
25%	28.184862	48.121656	3.919688e+05	1.226660e+06	2756.000000	137.547500	57.548364	7.4
50%	39.737289	60.698947	1.098100e+06	3.456018e+06	12323.576366	446.155000	101.426862	9.8
75%	91.479375	101.769335	2.842353e+06	9.931084e+06	16647.862761	1399.040000	124.820632	12.4
max	144.910000	176.800000	8.741197e+06	2.531512e+07	26142.470175	3163.524306	175.490000	17.2

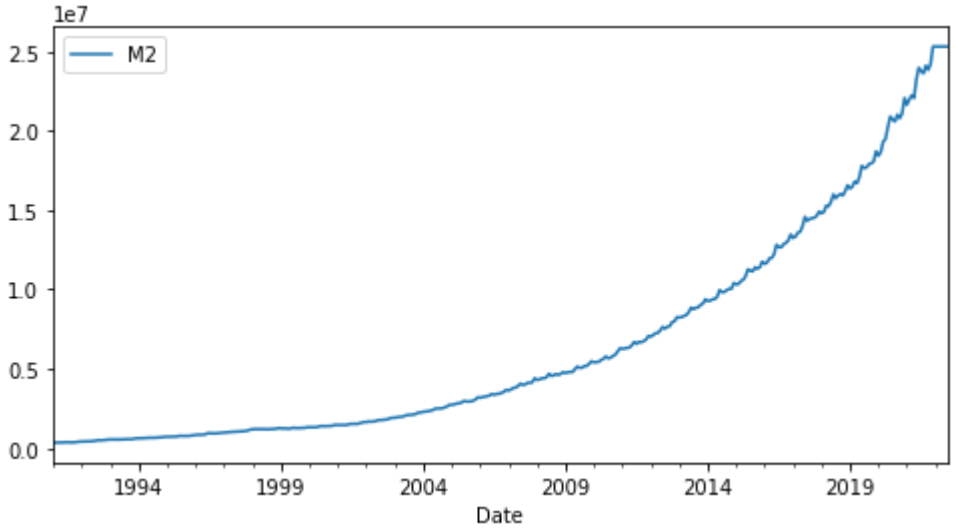
```
Normality Test for:  CPI
Sample looks Gaussian. Statistics=nan, p=1.000
Normality Test for:  ER
Sample does not look Gaussian. Statistics=0.914, p=0.000
Normality Test for:  M0
Sample does not look Gaussian. Statistics=0.777, p=0.000
Normality Test for:  M2
Sample does not look Gaussian. Statistics=0.817, p=0.000
Normality Test for:  RES
Sample does not look Gaussian. Statistics=0.912, p=0.000
Normality Test for:  REM
Sample does not look Gaussian. Statistics=0.840, p=0.000
Normality Test for:  IPLSM
Sample does not look Gaussian. Statistics=0.935, p=0.000
Normality Test for:  TB
Sample does not look Gaussian. Statistics=0.976, p=0.000
Normality Test for:  Disr
Sample does not look Gaussian. Statistics=0.955, p=0.000
Normality Test for:  OIL
Sample does not look Gaussian. Statistics=0.895, p=0.000
Normality Test for:  USIPI
Sample does not look Gaussian. Statistics=0.891, p=0.000
Normality Test for:  PSC
Sample does not look Gaussian. Statistics=0.878, p=0.000
Normality Test for:  PSB
Sample does not look Gaussian. Statistics=0.737, p=0.000
Normality Test for:  LR
Sample does not look Gaussian. Statistics=0.923, p=0.000
Normality Test for:  REER
Sample does not look Gaussian. Statistics=0.910, p=0.000
Normality Test for:  WPI
Sample does not look Gaussian. Statistics=0.874, p=0.000
Normality Test for:  WCPI
Sample does not look Gaussian. Statistics=0.906, p=0.000
Normality Test for:  WP
Sample does not look Gaussian. Statistics=0.851, p=0.000
Normality Test for:  Wfood
Sample does not look Gaussian. Statistics=0.919, p=0.000
```

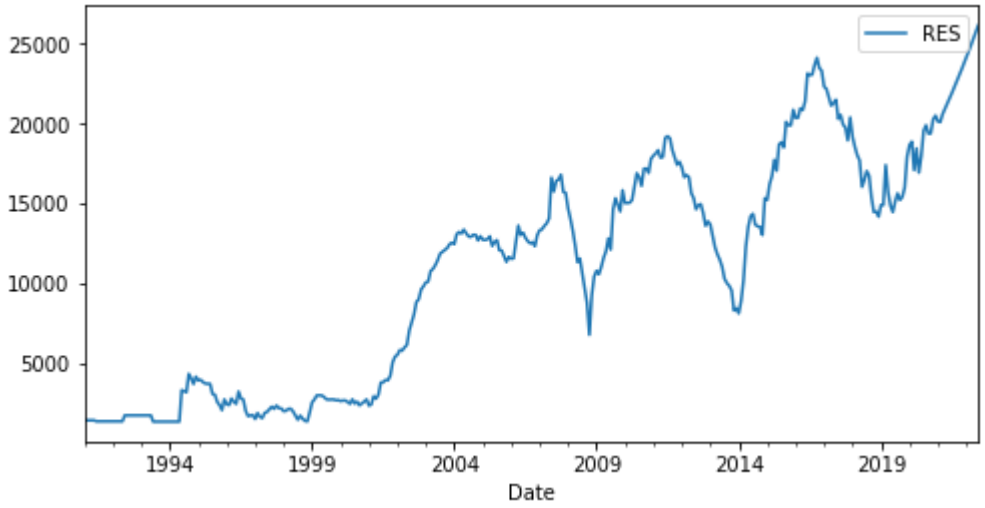
In [17]:
```
## Line plot all sereis
for (columnName, columnData) in df.iloc[:,1:].iteritems():
    df.plot(x = 'Date',y=columnName,figsize=(8,4))
    plt.show()
```

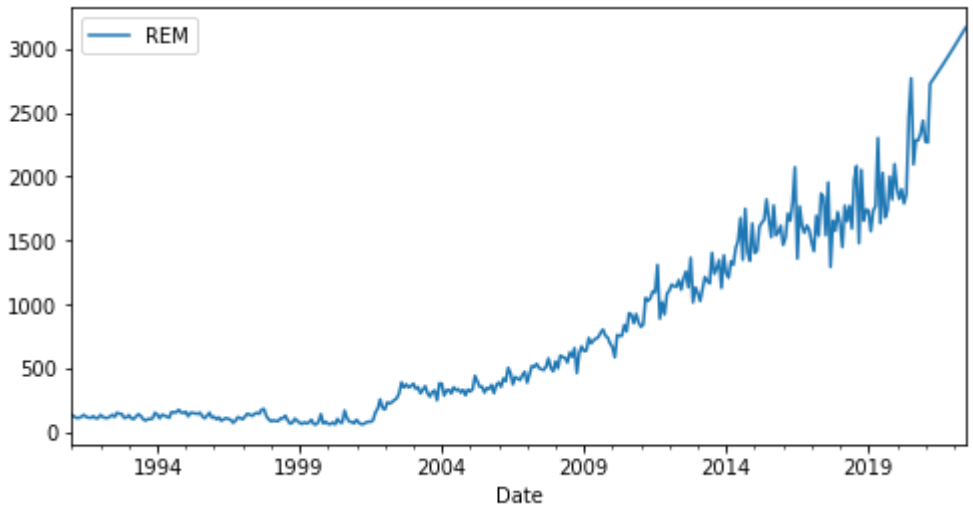

CPI
140
120
100
80
60
40
20
1994
1999
2004
2009
2014
2019
Date

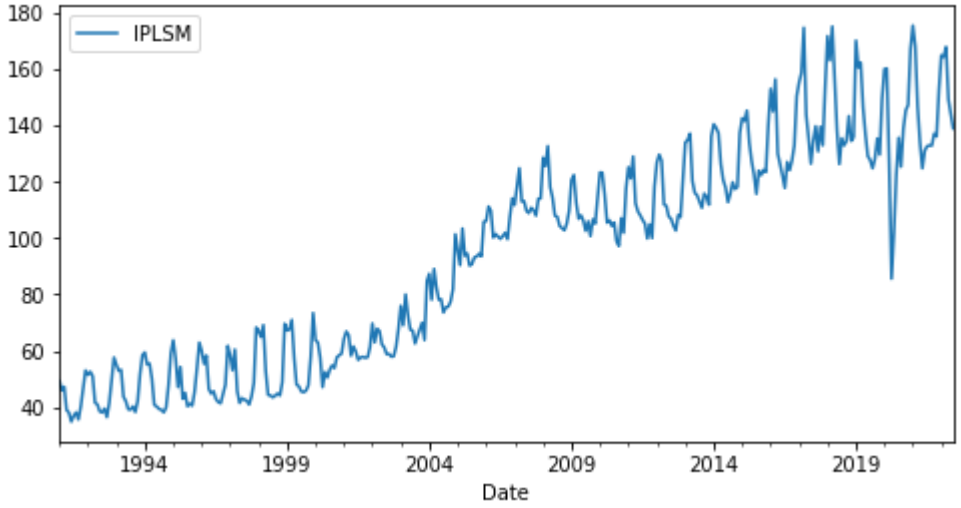

ER
180
160
140
120
100
80
60
40
20
1994
1999
2004
2009
2014
2019
Date

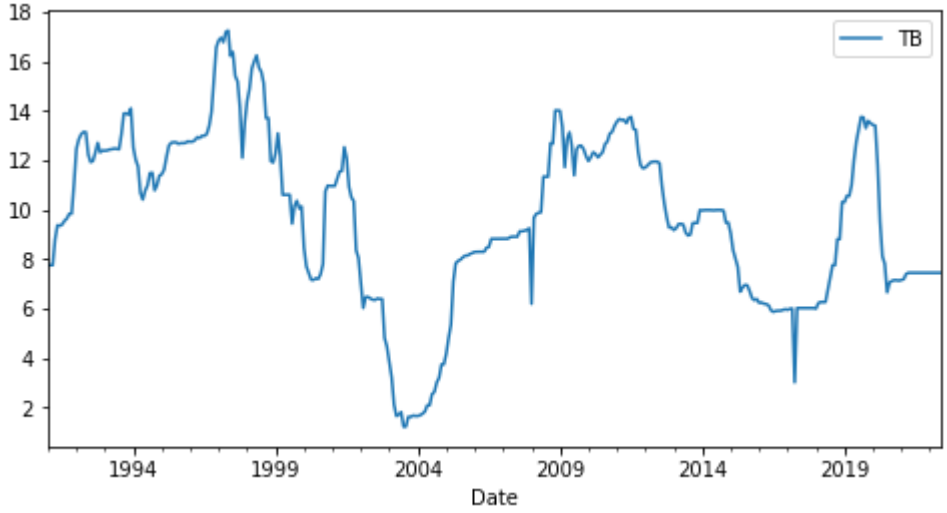

1e7
M0
0.8
0.6
0.4
0.2
0.0
1994
1999
2004
2009
2014
2019
Date

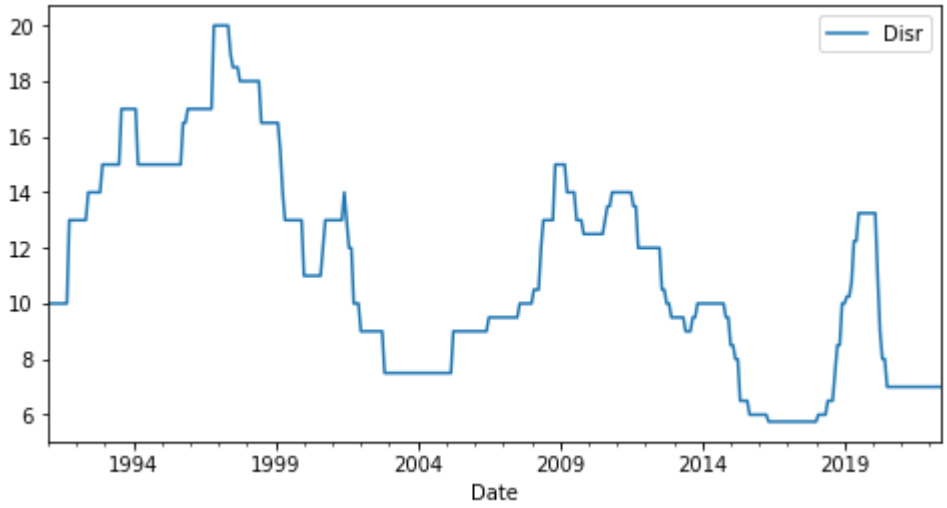

1e7
M2
2.5
2.0
1.5
1.0
0.5
0.0
1994
1999
2004
2009
2014
2019
Date

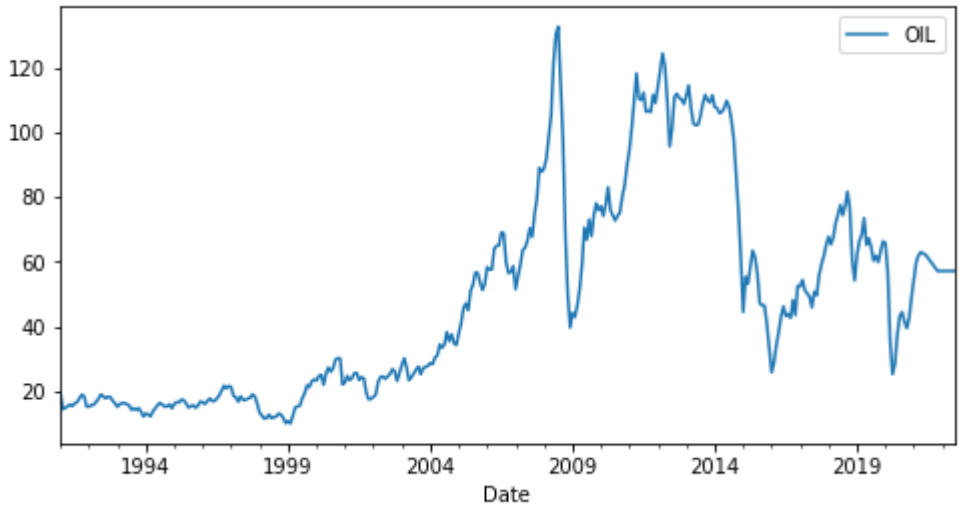

RES
25000
20000
15000
10000
5000
1994
1999
2004
2009
2014
2019
Date

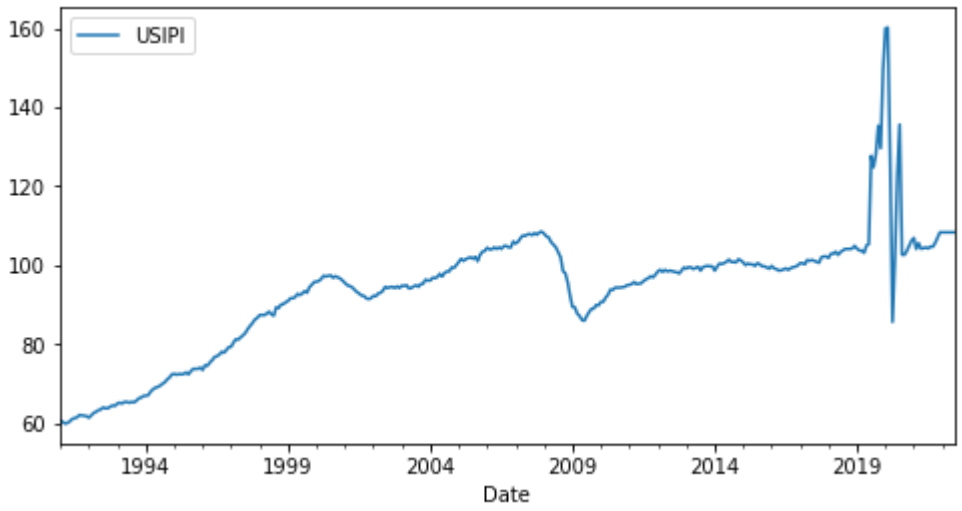

REM
3000
2500
2000
1500
1000
500
0
1994
1999
2004
2009
2014
2019
Date

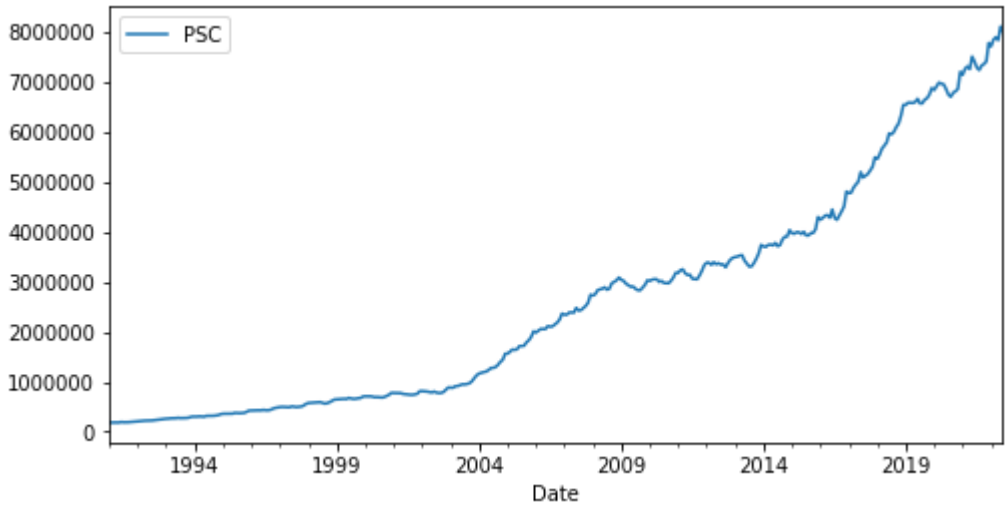

IPLSM
180
160
140
120
100
80
60
40
1994
1999
2004
2009
2014
2019
Date

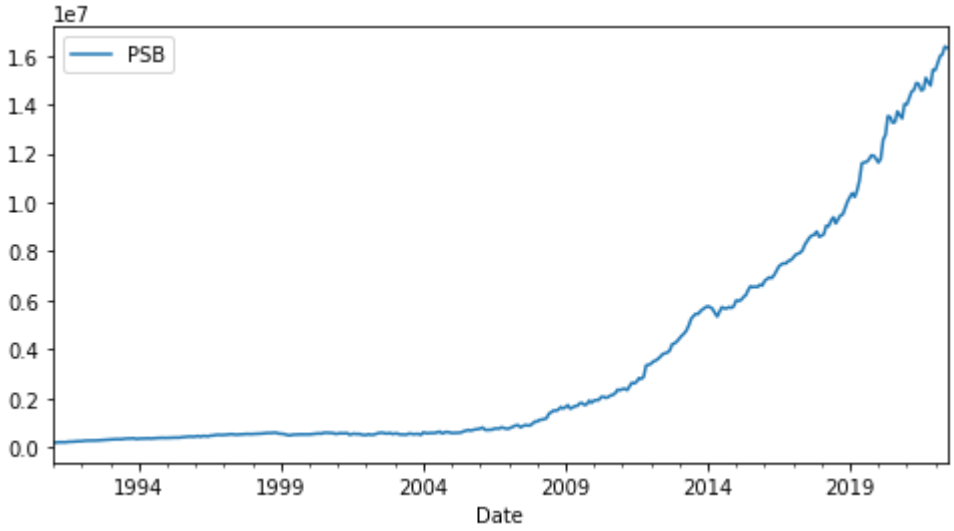

TB
18
16
14
12
10
8
6
4
2
1994
1999
2004
2009
2014
2019
Date

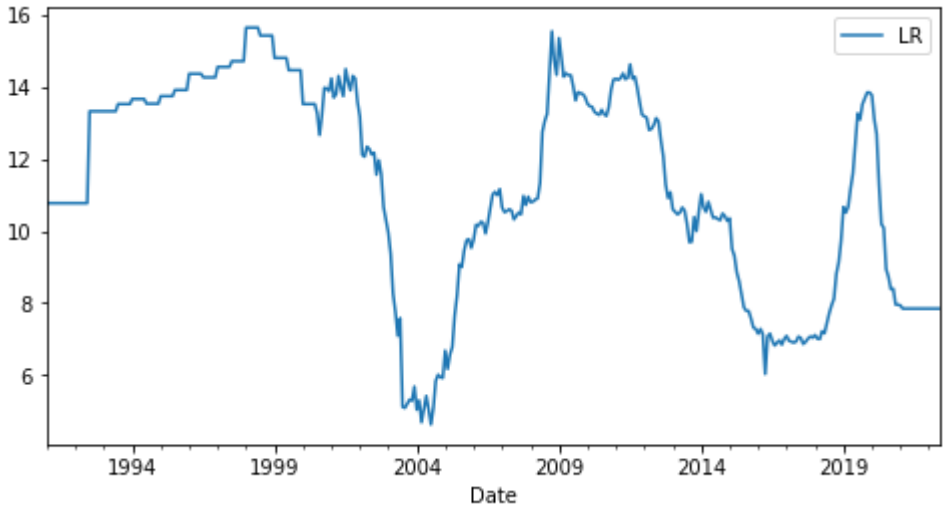

Disr
20
18
16
14
12
10
8
6
1994
1999
2004
2009
2014
2019
Date

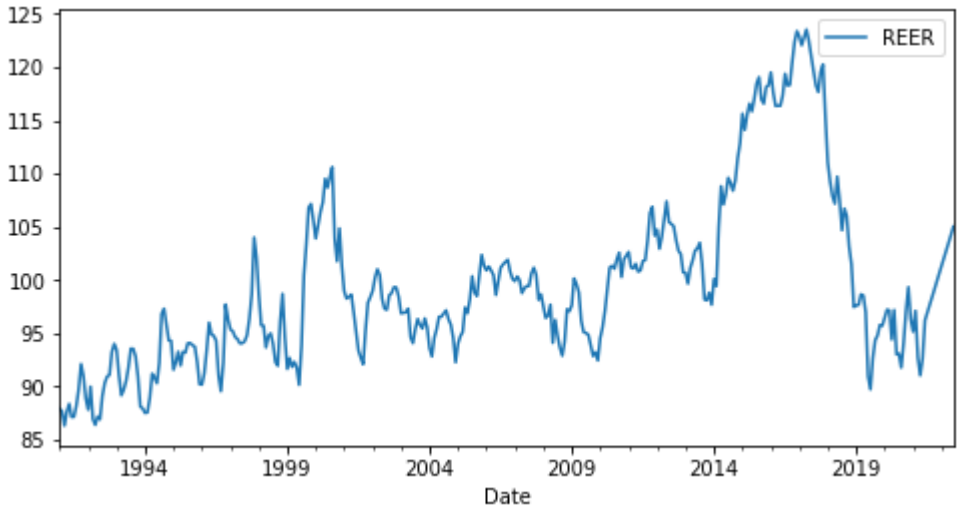

OIL
120
100
80
60
40
20
1994
1999
2004
2009
2014
2019
Date

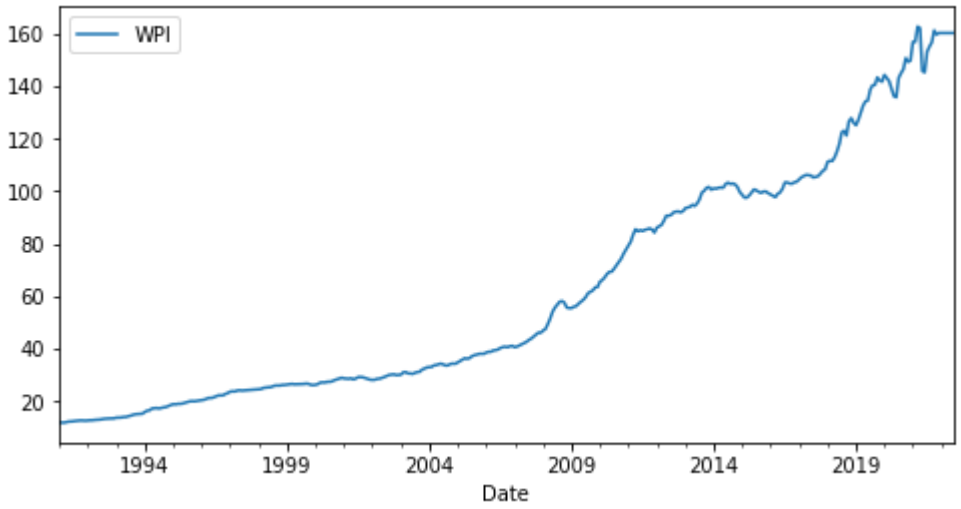

USIPI
160
140
120
100
80
60
1994
1999
2004
2009
2014
2019
Date

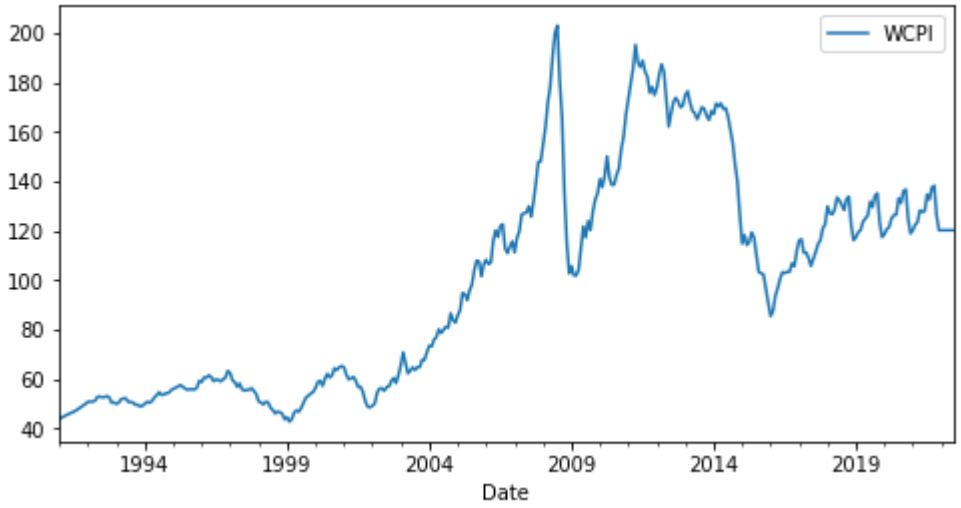

PSC
8000000
7000000
6000000
5000000
4000000
3000000
2000000
1000000
0
1994
1999
2004
2009
2014
2019
Date

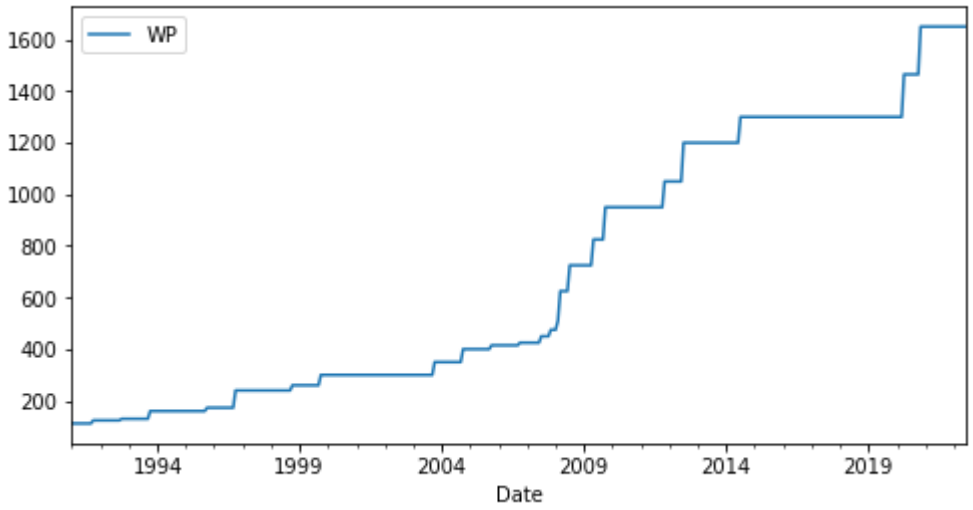

1e7
PSB
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
1994
1999
2004
2009
2014
2019
Date

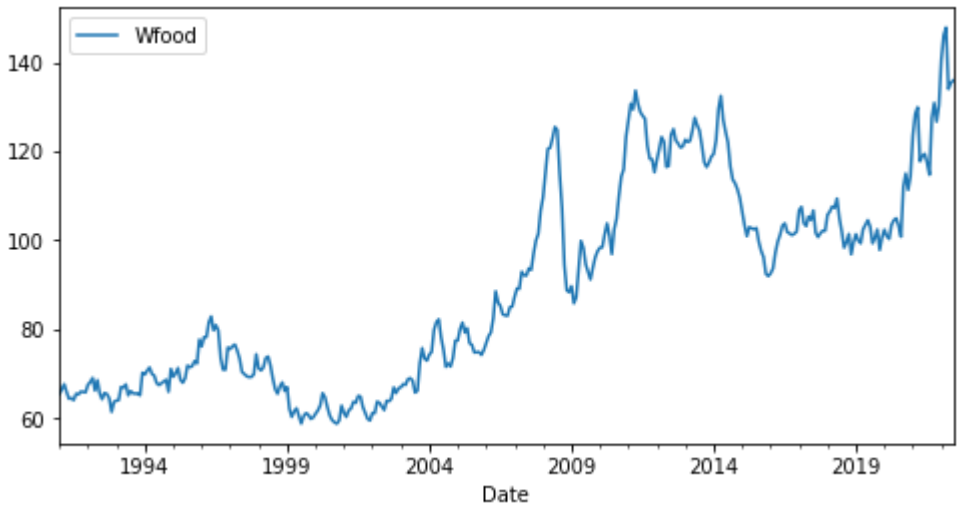

LR
16
14
12
10
8
6
1994
1999
2004
2009
2014
2019
Date


REER
125
120
115
110
105
100
95
90
85
1994
1999
2004
2009
2014
2019
Date


WPI
160
140
120
100
80
60
40
20
1994
1999
2004
2009
2014
2019
Date


WCPI
200
180
160
140
120
100
80
60
40
1994
1999
2004
2009
2014
2019
Date


WP
1600
1400
1200
1000
800
600
400
200
1994
1999
2004
2009
2014
2019
Date

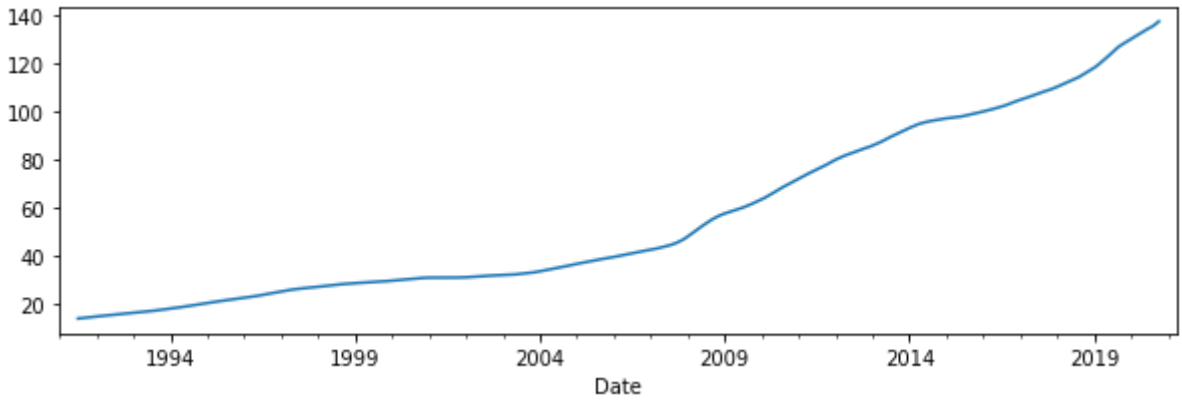

Wfood
140
120
100
80
60
1994
1999
2004
2009
2014
2019
Date

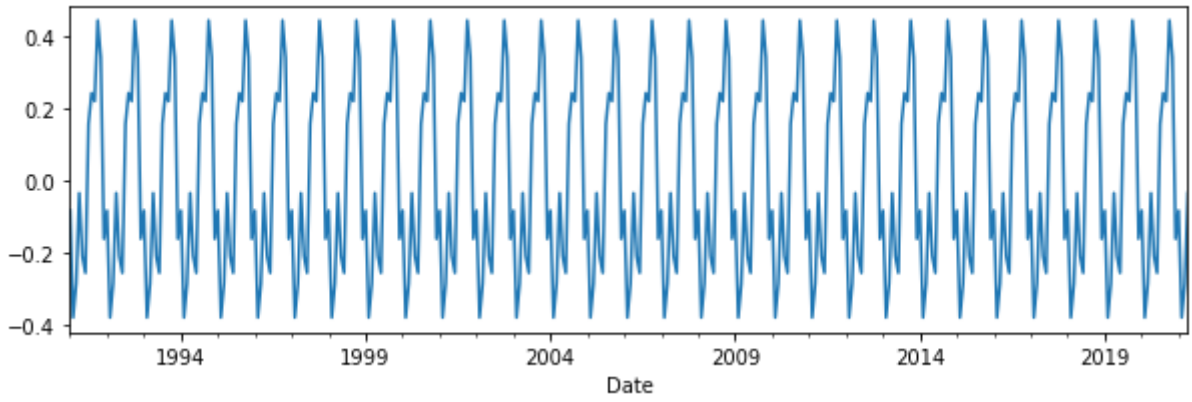
In time sereis problem we have to check if the series is stationary of not because if series is not statinary at levels then we have to take diference untill it gets stationary.

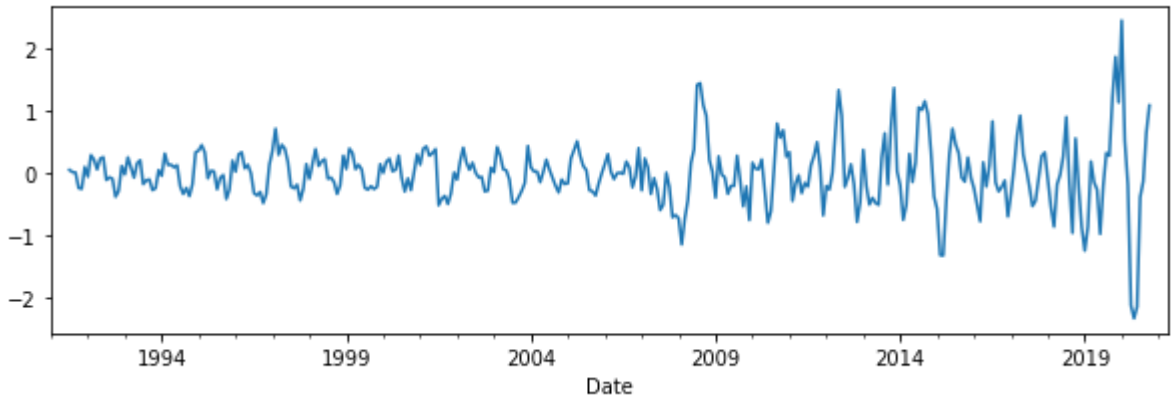
Also we will decompose all the series in seasonal, trend and residual and checkthe series.

In [73]:
```
# First seasonally decmposing all series.
df = df.set_index('Date')
for (columnName, columnData) in df.iteritems():
    print('Series:',columnName)
    result = seasonal_decompose(df[columnName].dropna(), model='additive')
    result.trend.plot(figsize=(10,3))
    plt.show()
    result.seasonal.plot(figsize=(10,3))
    plt.show()
    result.resid.plot(figsize=(10,3))
    plt.show()
```

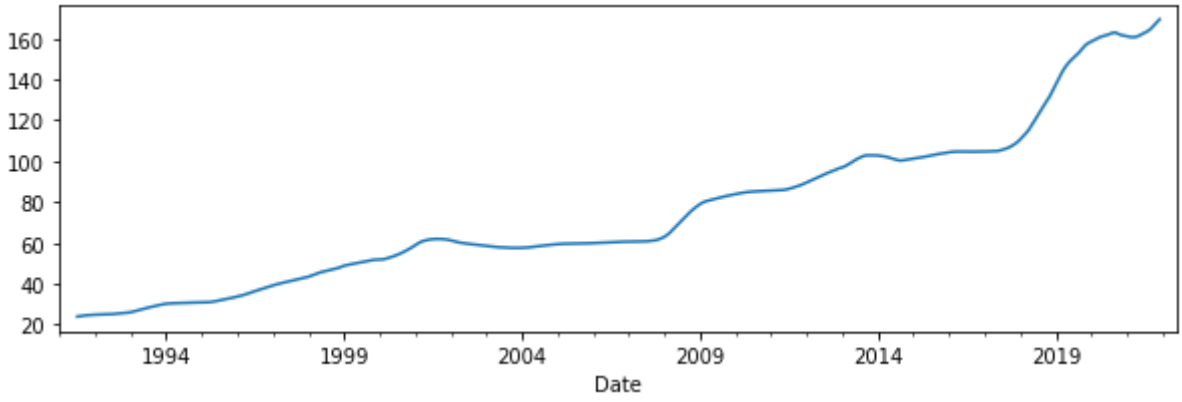
Series: CPI

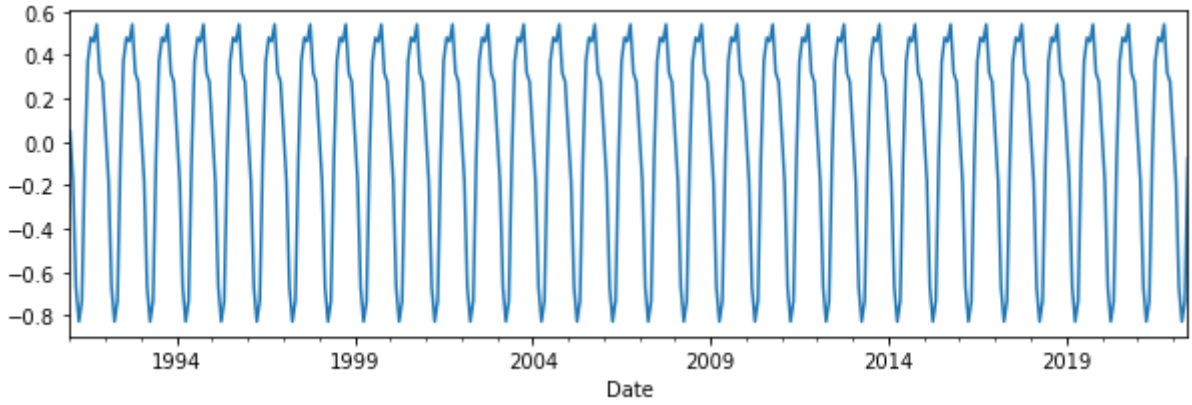


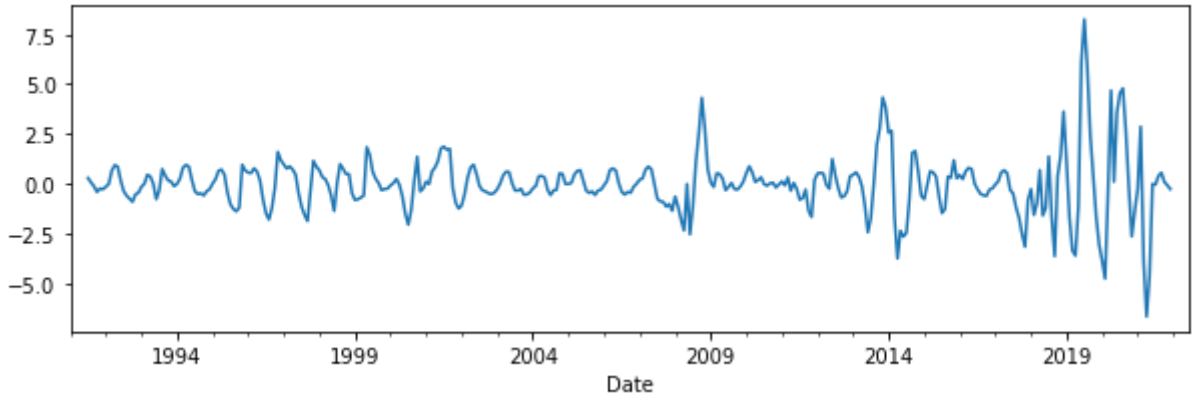




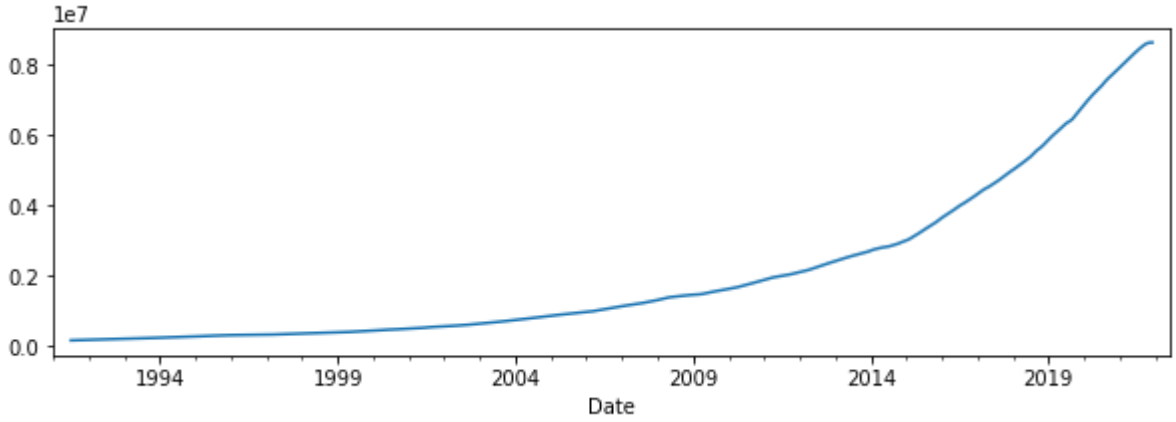
Series: ER

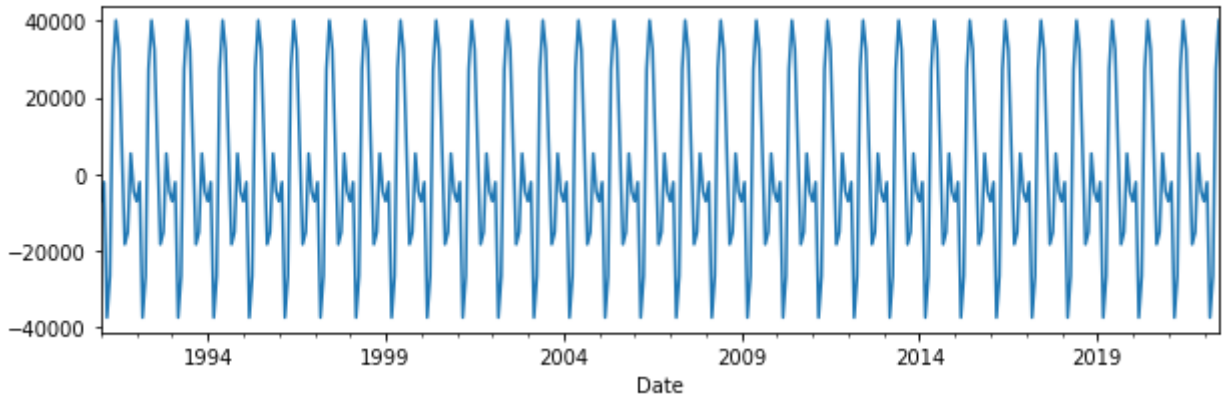


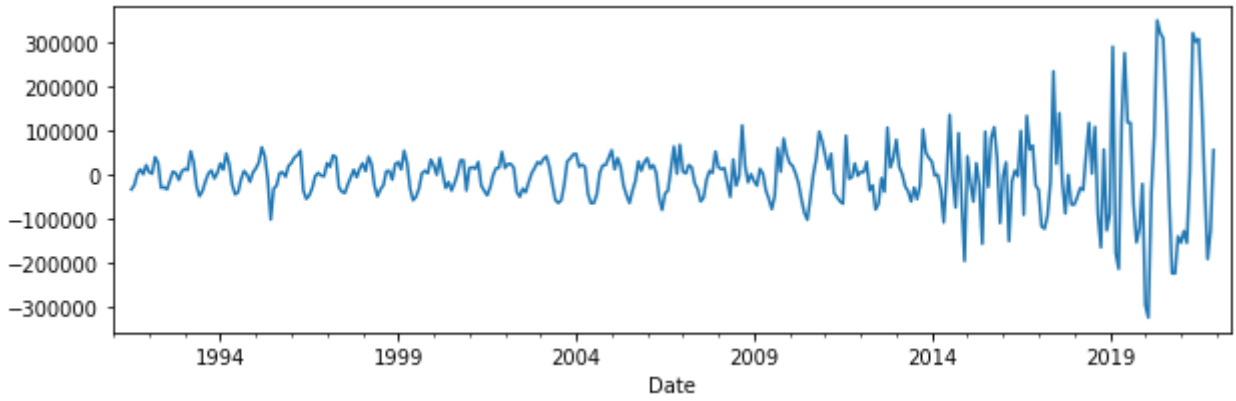

0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
1994
1999
2004
2009
2014
2019
Date


7.5
5.0
2.5
0.0
−2.5
−5.0
1994
1999
2004
2009
2014
2019
Date

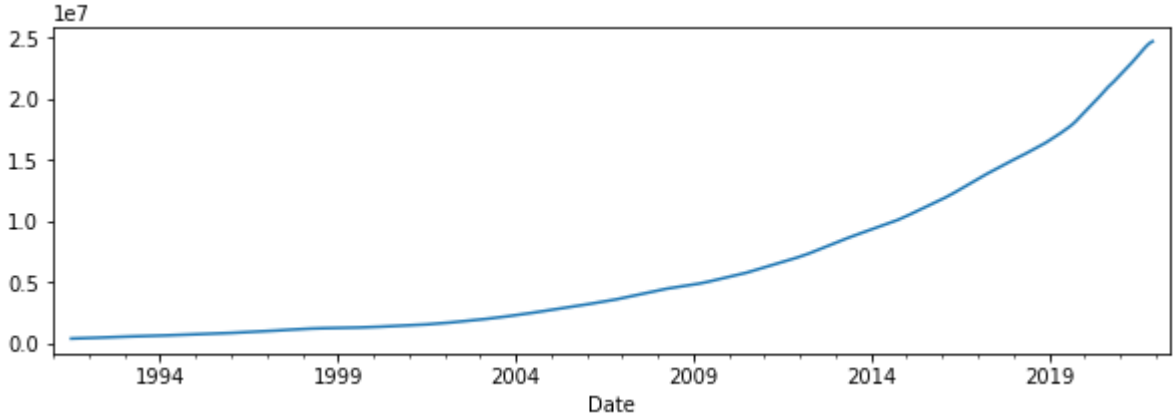
Series: M0

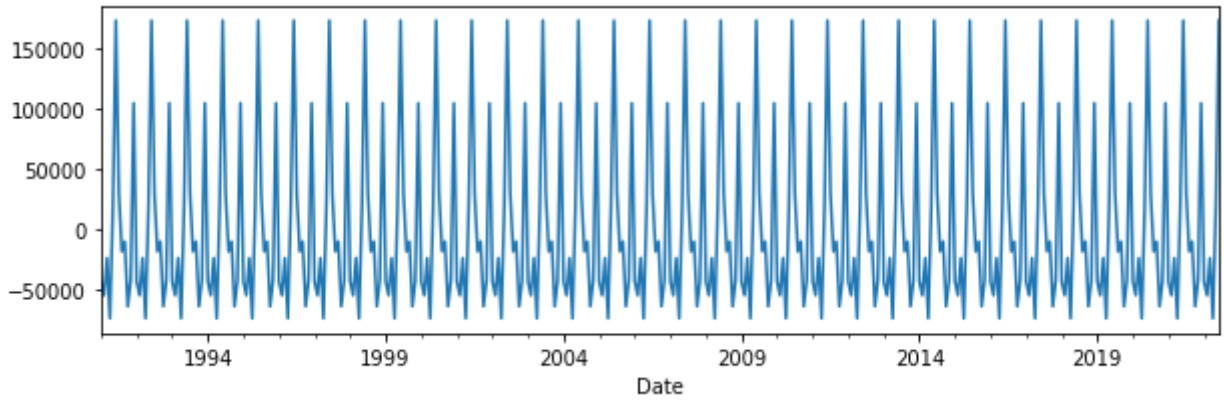

1e7
0.8
0.6
0.4
0.2
0.0
1994
1999
2004
2009
2014
2019
Date

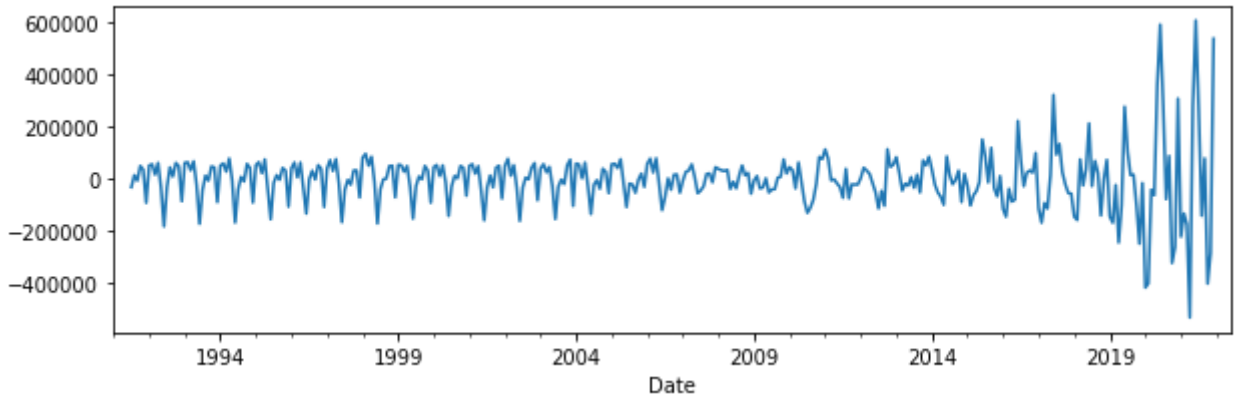

40000
20000
0
−20000
−40000
1994
1999
2004
2009
2014
2019
Date



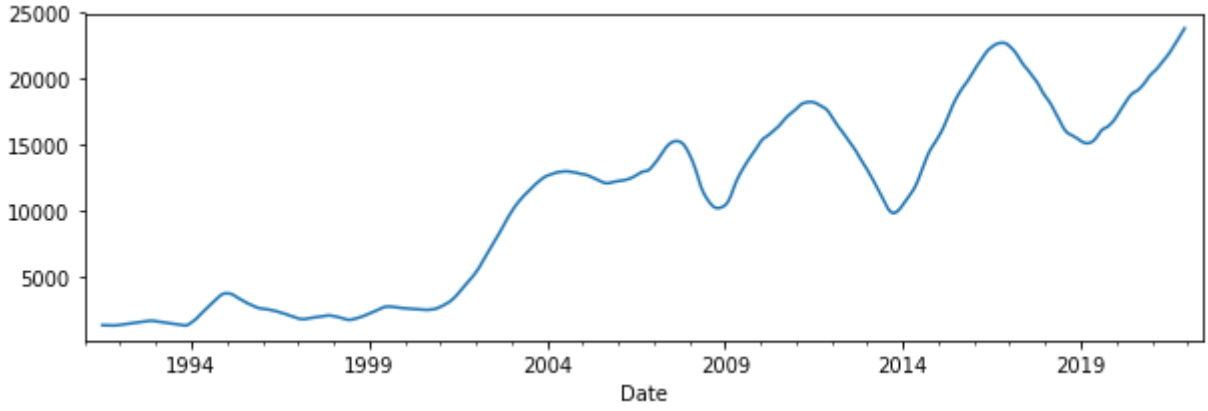
Series: M2

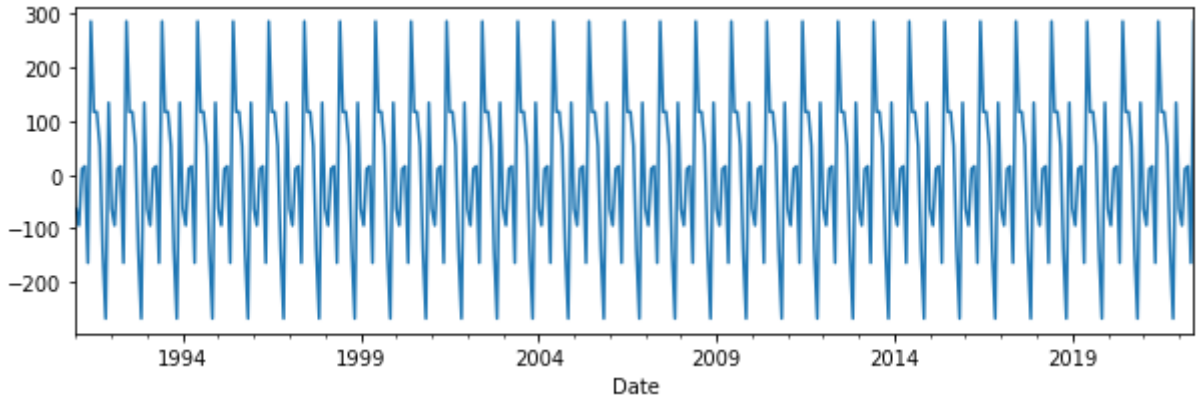


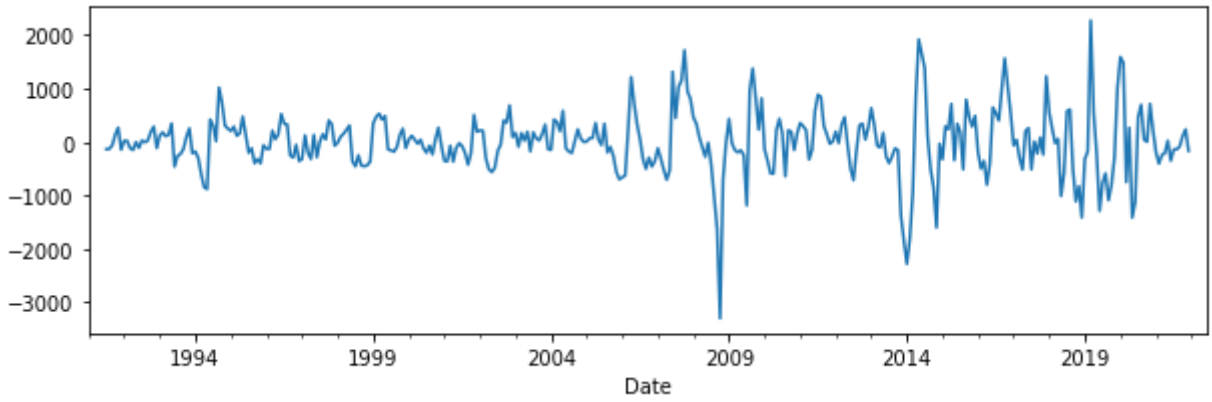




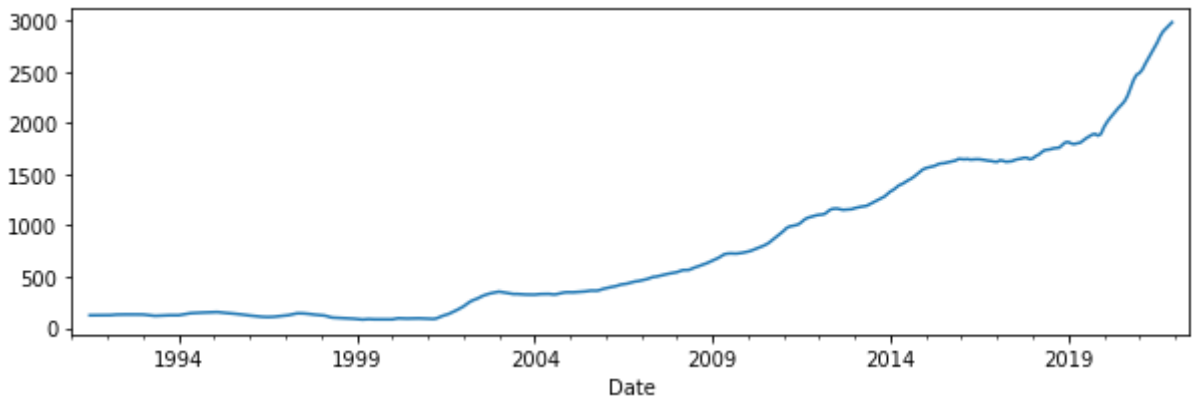
Series: RES

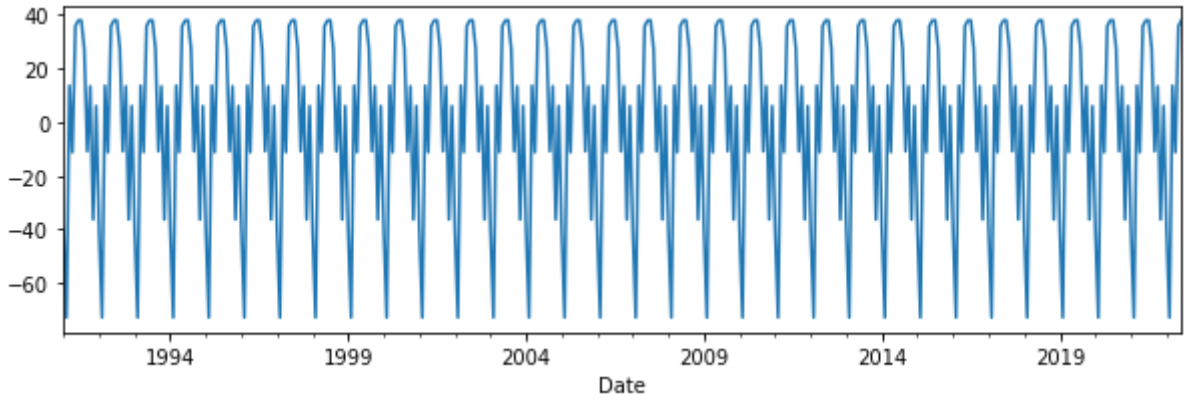

25000
20000
15000
10000
5000
1994
1999
2004
2009
2014
2019
Date

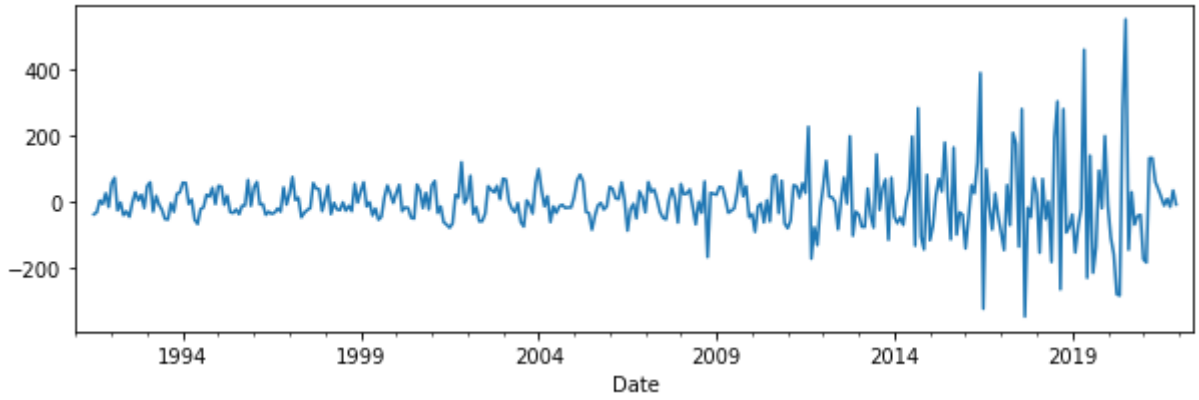

300
200
100
0
−100
−200
1994
1999
2004
2009
2014
2019
Date


2000
1000
0
−1000
−2000
−3000
1994
1999
2004
2009
2014
2019
Date

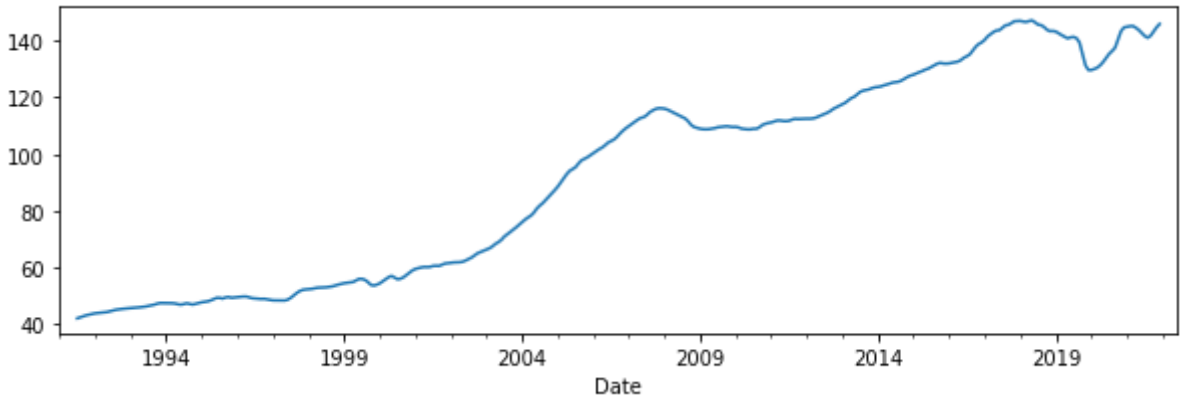
Series: REM

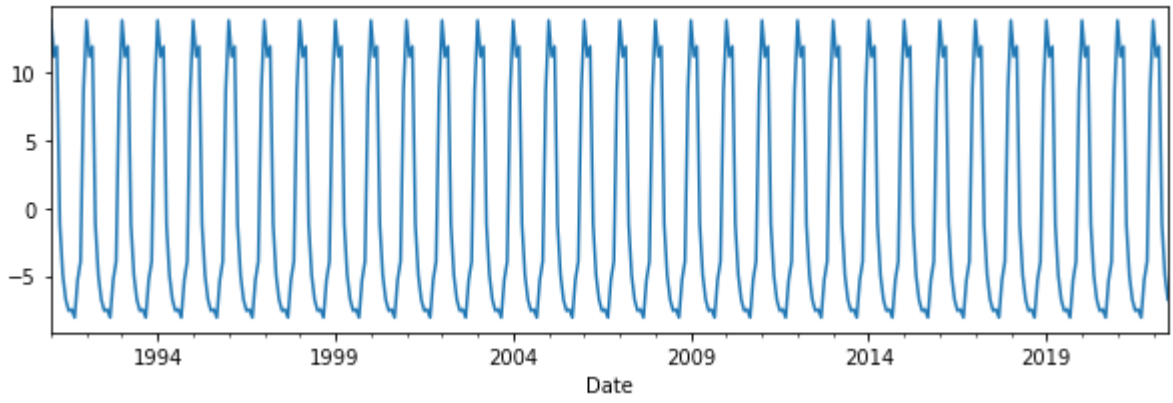

3000
2500
2000
1500
1000
500
0
1994
1999
2004
2009
2014
2019
Date

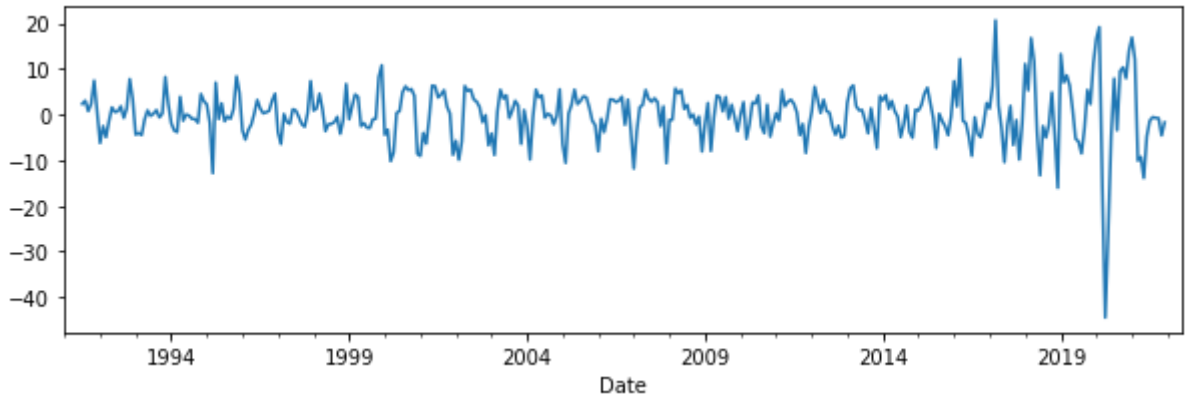




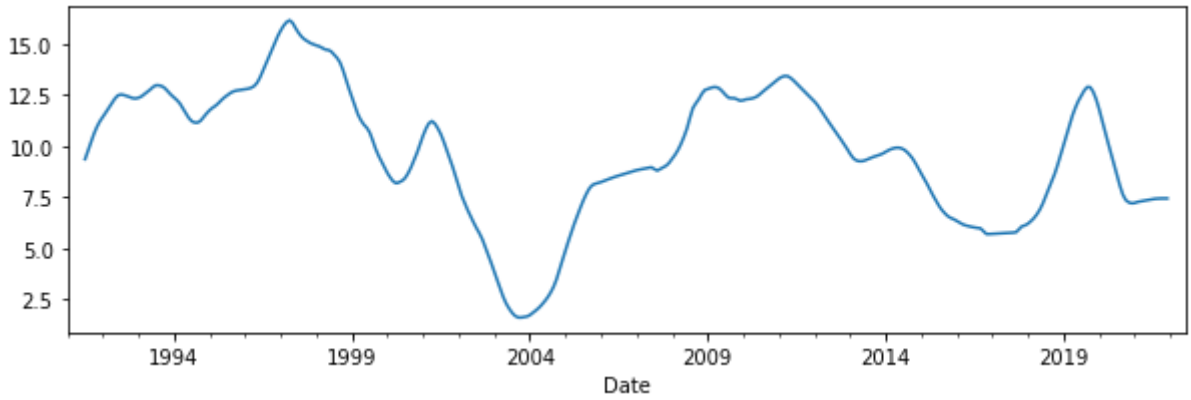
Series: IPLSM

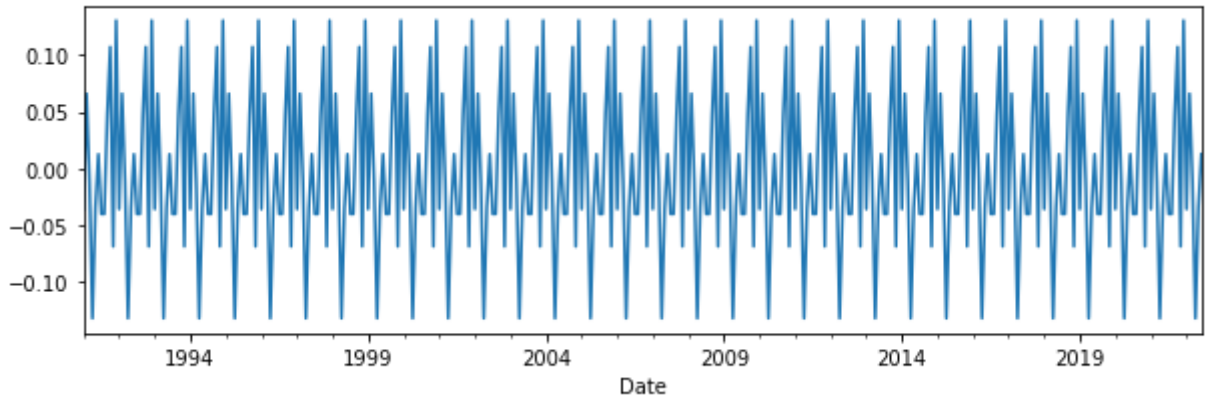


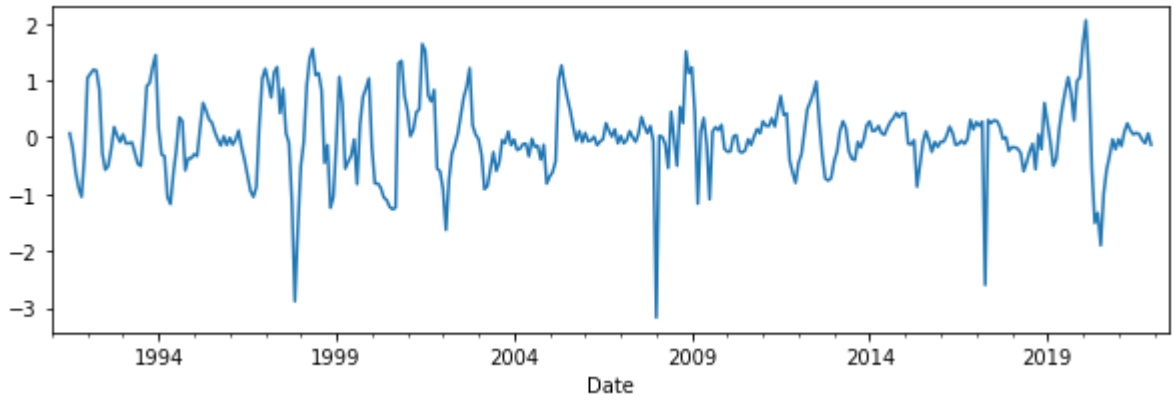




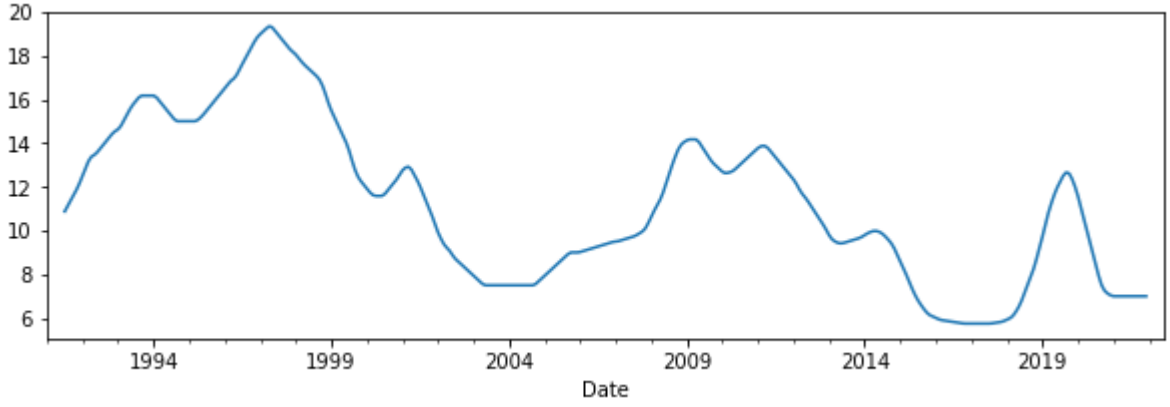
Series: TB

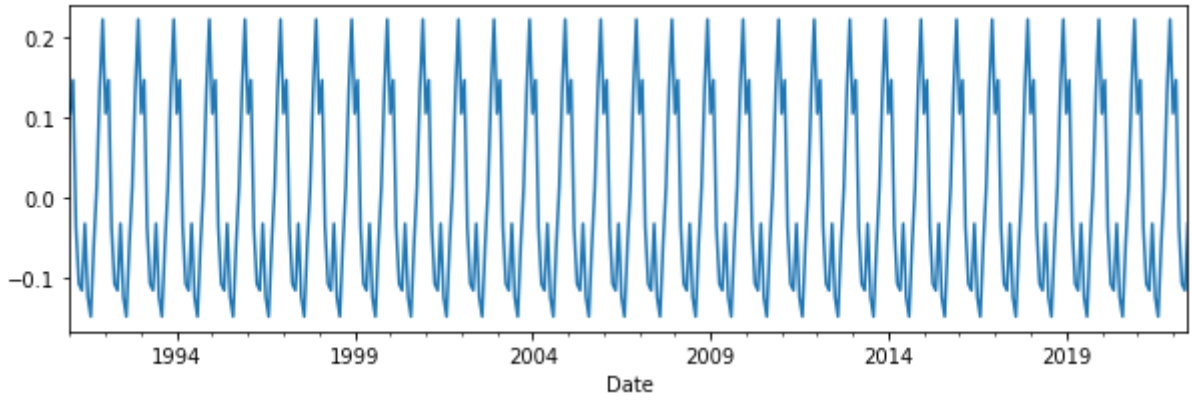


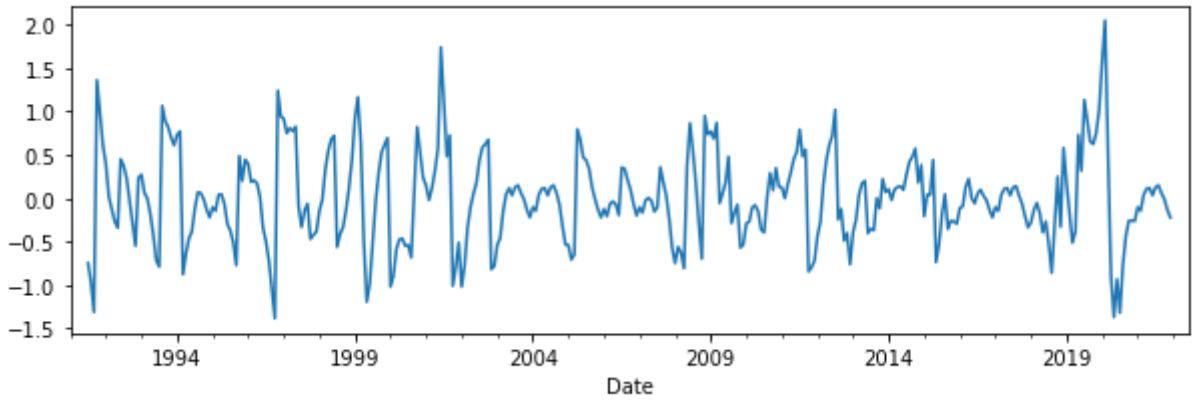




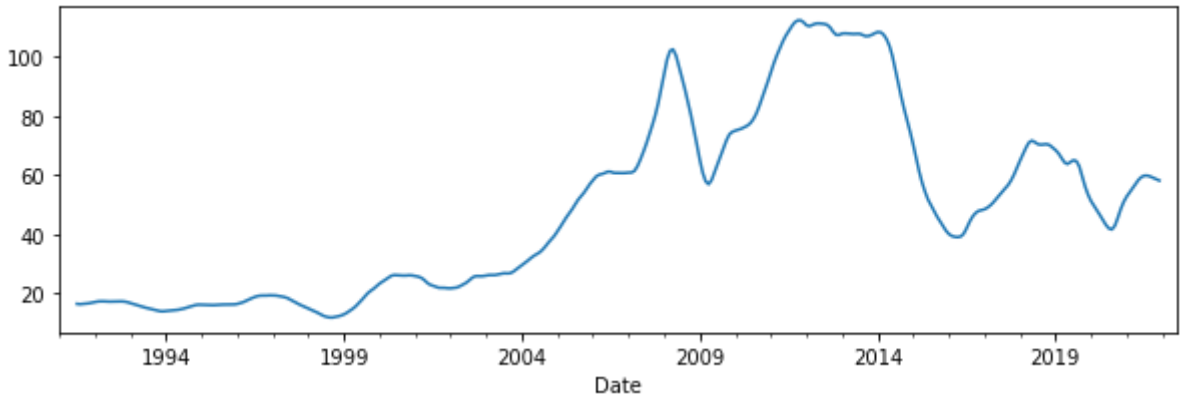
Series: Disr

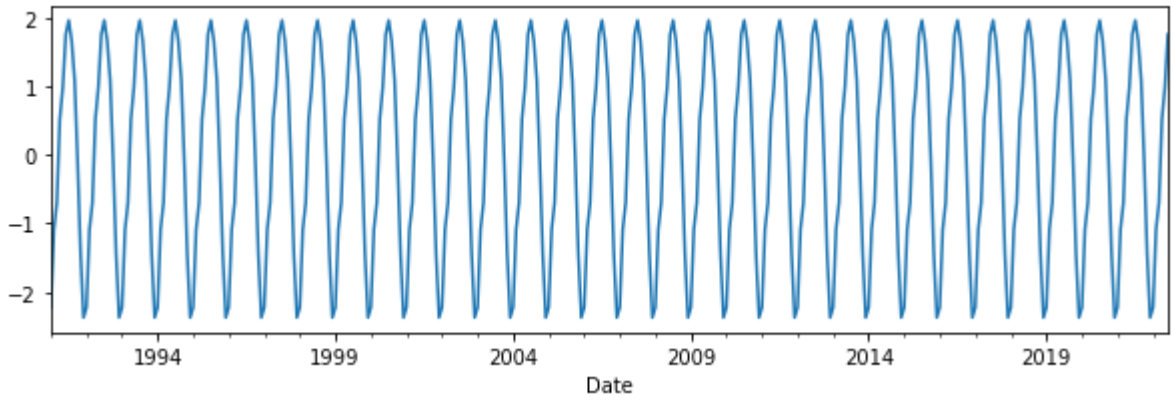

20
18
16
14
12
10
8
6
1994
1999
2004
2009
2014
2019
Date

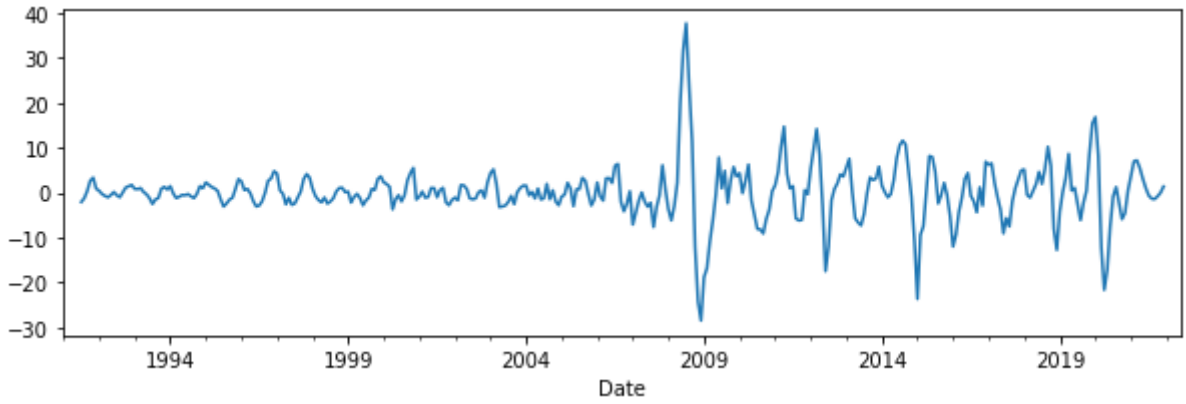

0.2
0.1
0.0
−0.1
1994
1999
2004
2009
2014
2019
Date


2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
1994
1999
2004
2009
2014
2019
Date

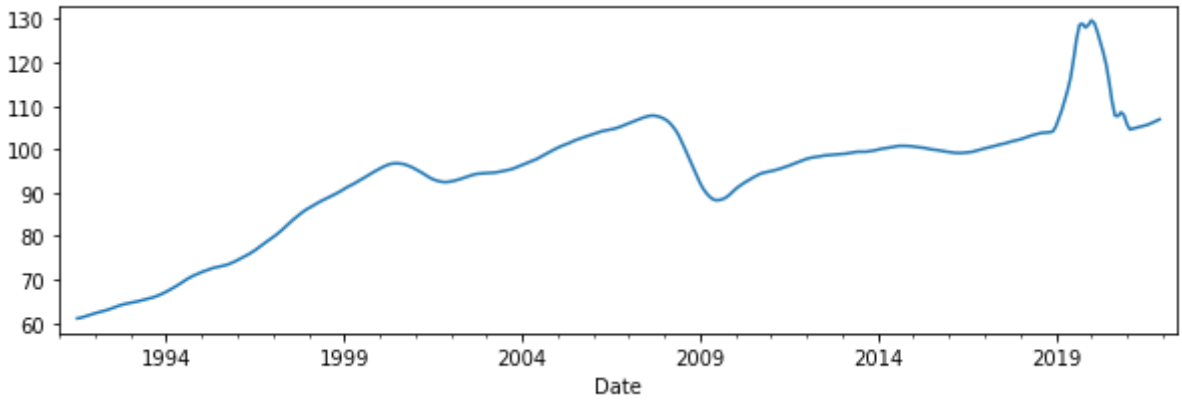
Series: OIL

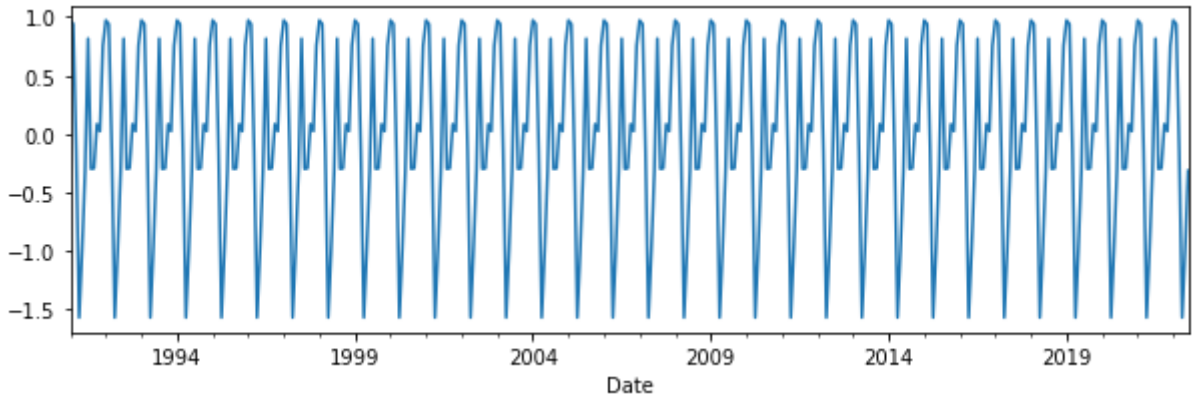

100
80
60
40
20
1994
1999
2004
2009
2014
2019
Date

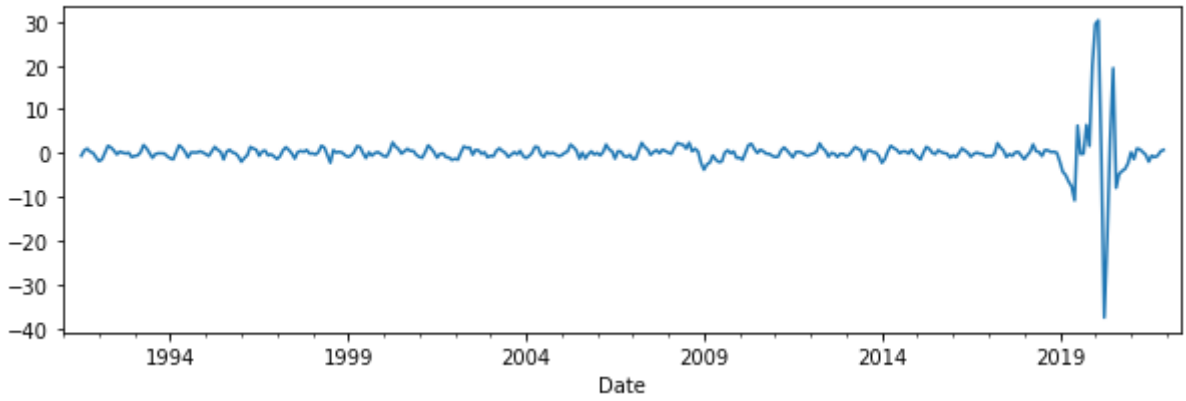




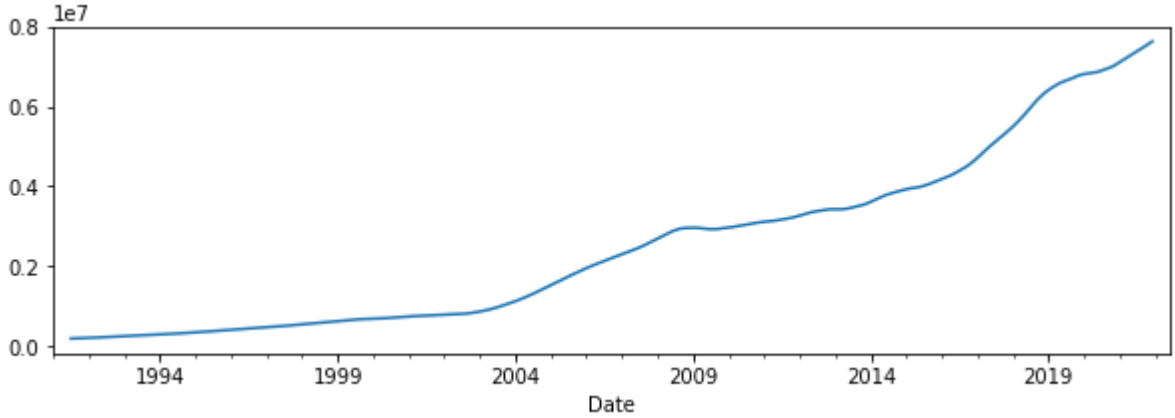
Series: USIPI

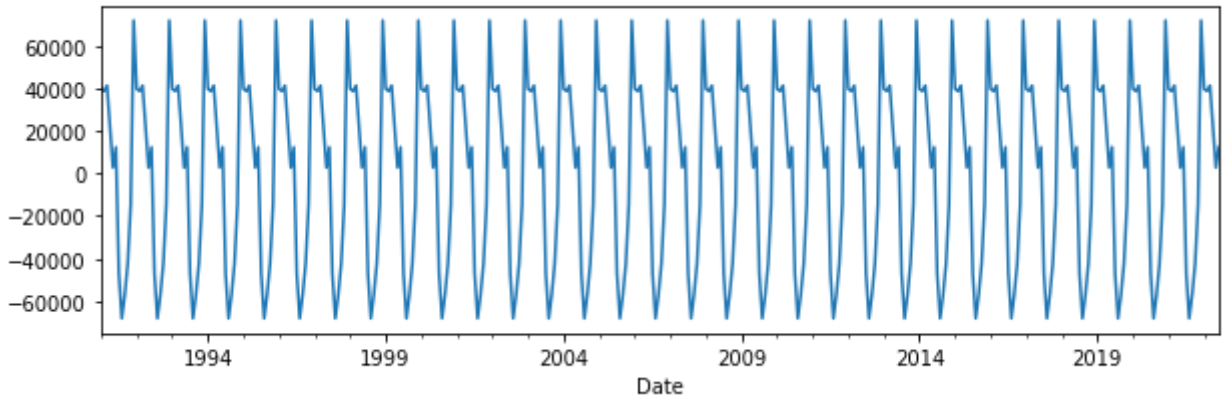


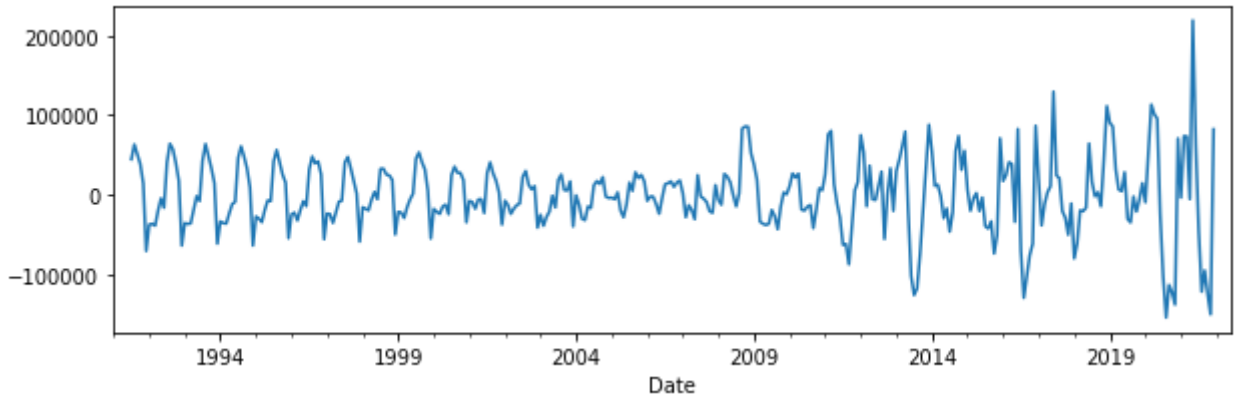




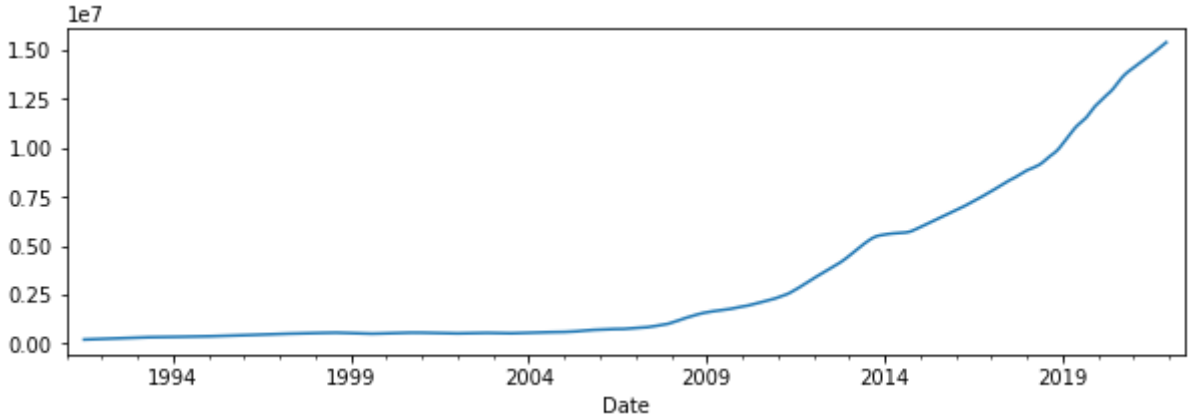
Series: PSC

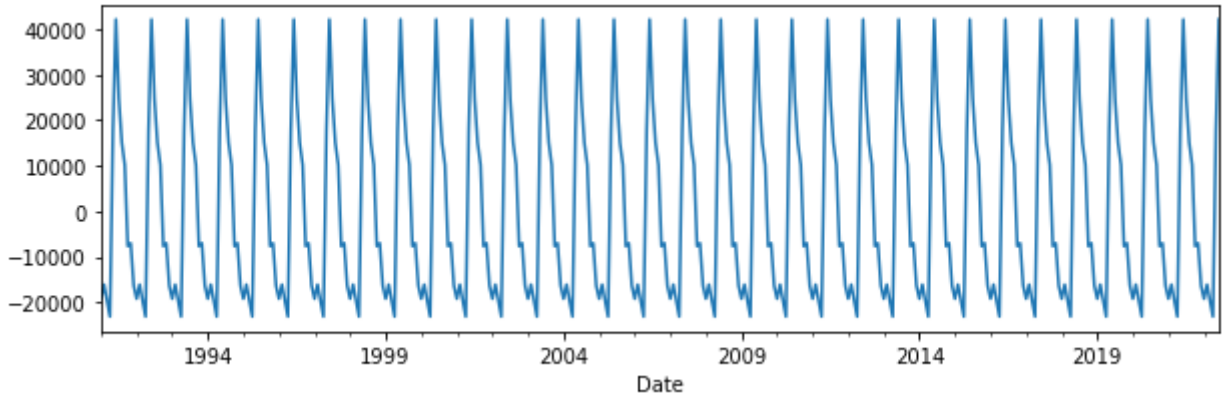


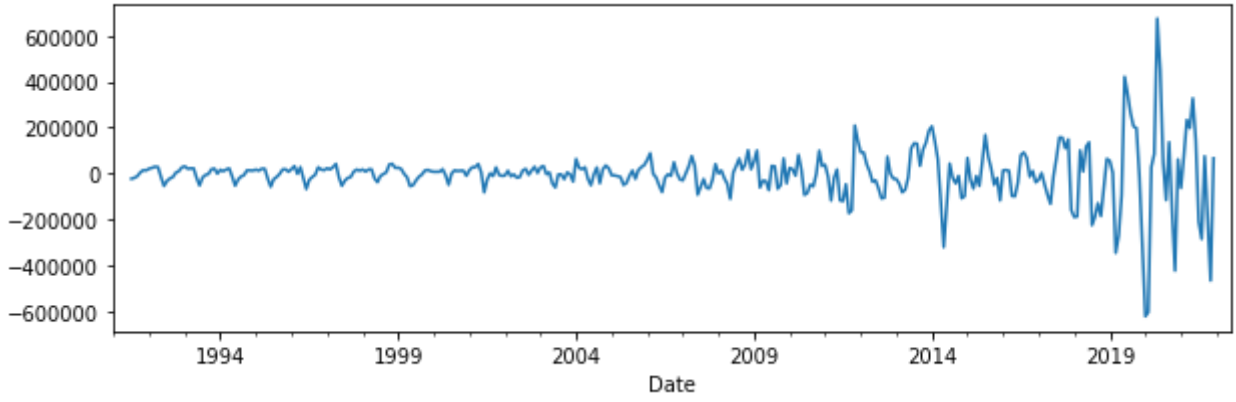




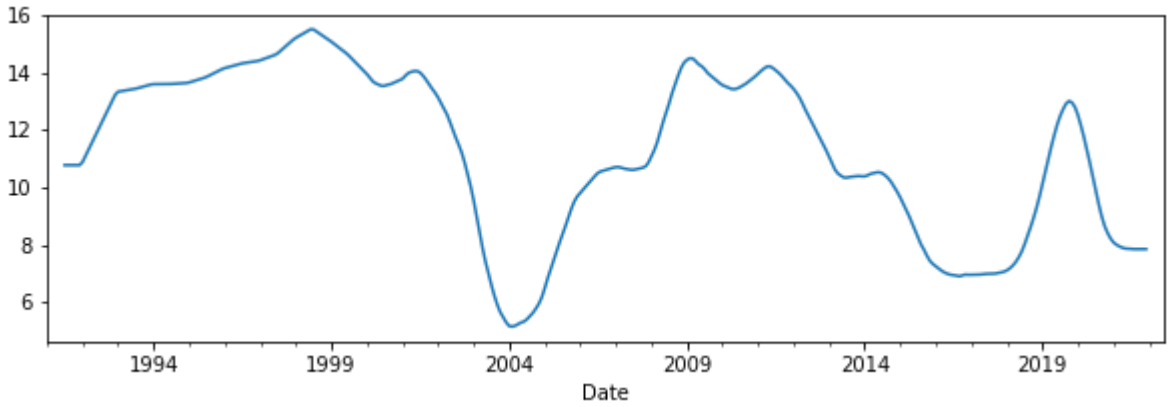
Series: PSB

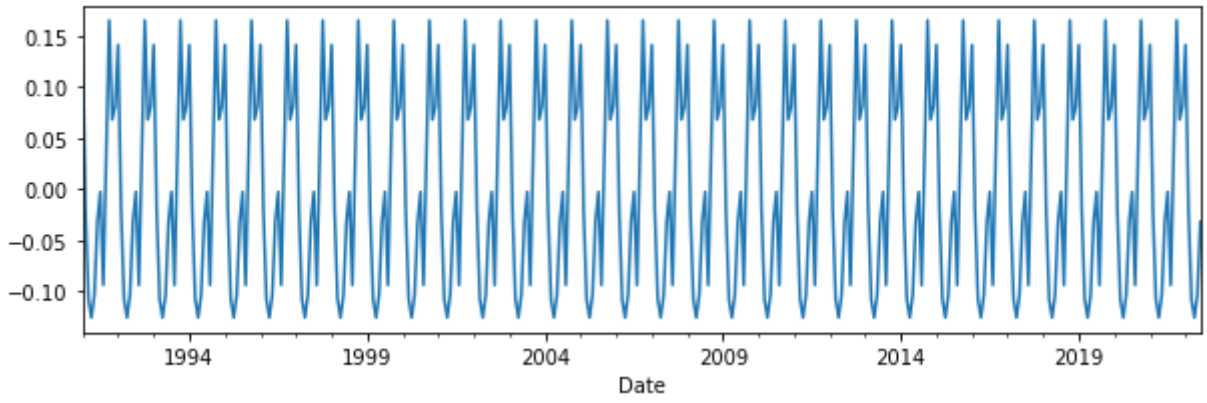

1e7
1.50
1.25
1.00
0.75
0.50
0.25
0.00
1994
1999
2004
2009
2014
2019
Date

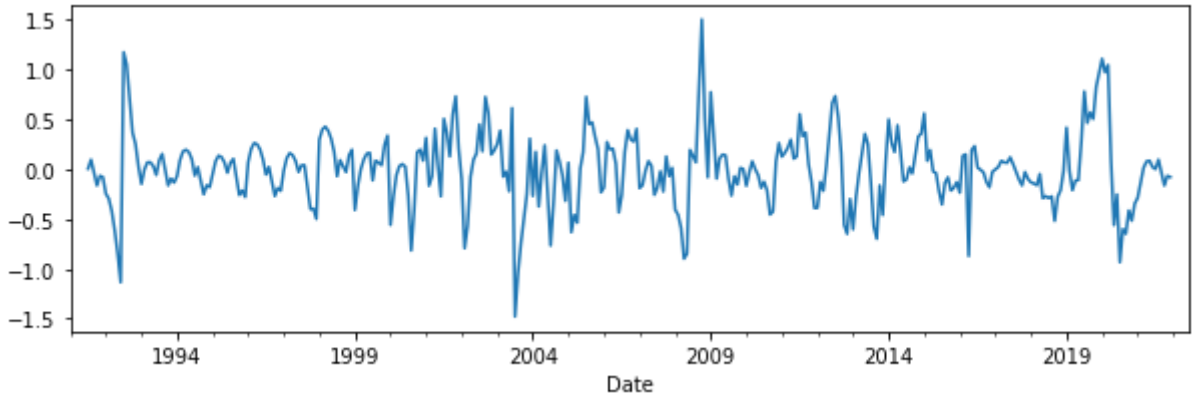

40000
30000
20000
10000
0
−10000
−20000
1994
1999
2004
2009
2014
2019
Date


600000
400000
200000
0
−200000
−400000
−600000
1994
1999
2004
2009
2014
2019
Date

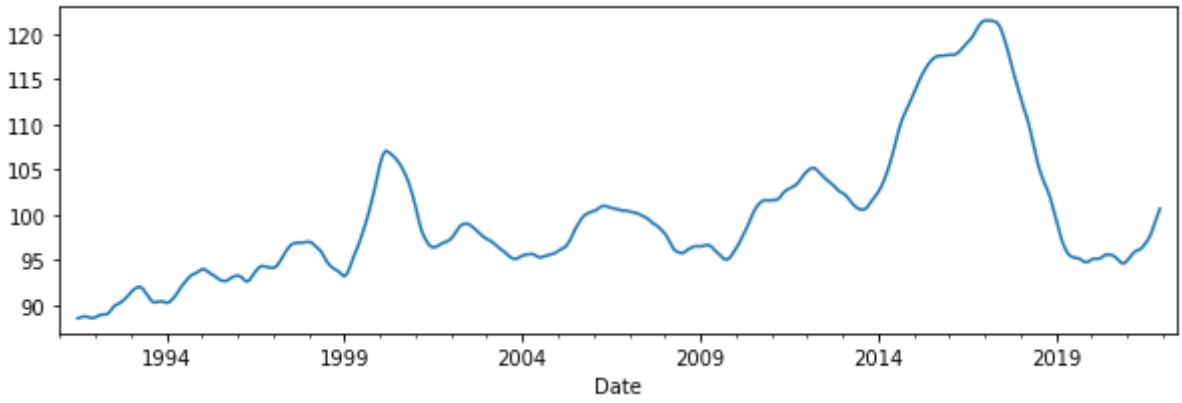
Series: LR

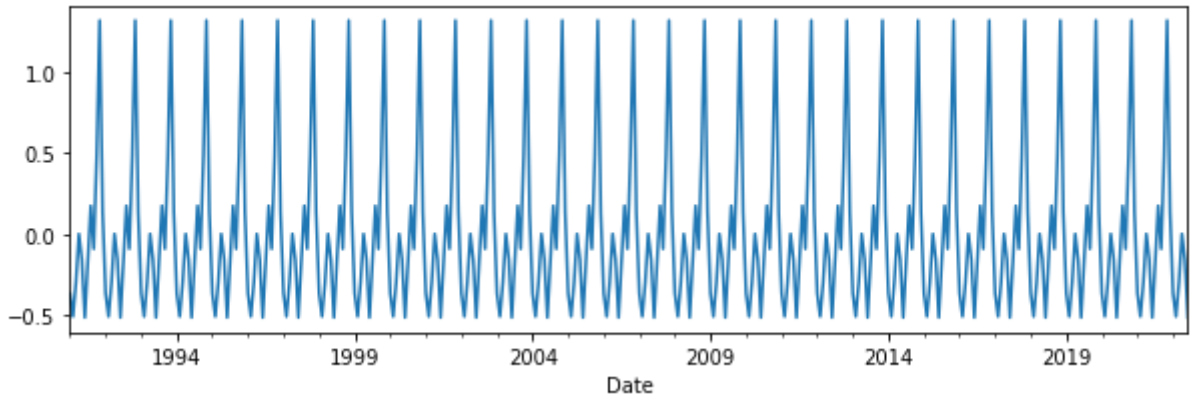

16
14
12
10
8
6
1994
1999
2004
2009
2014
2019
Date

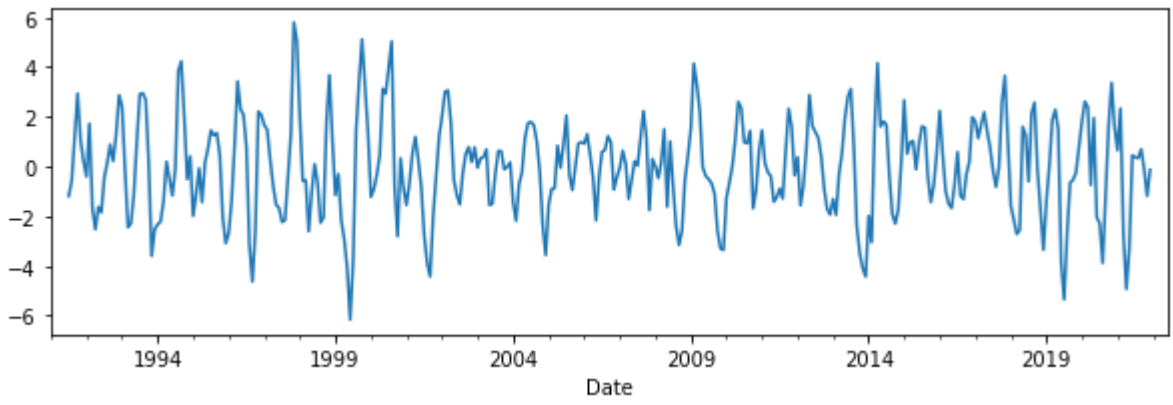

0.15
0.10
0.05
0.00
−0.05
−0.10
1994
1999
2004
2009
2014
2019
Date


1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
1994
1999
2004
2009
2014
2019
Date

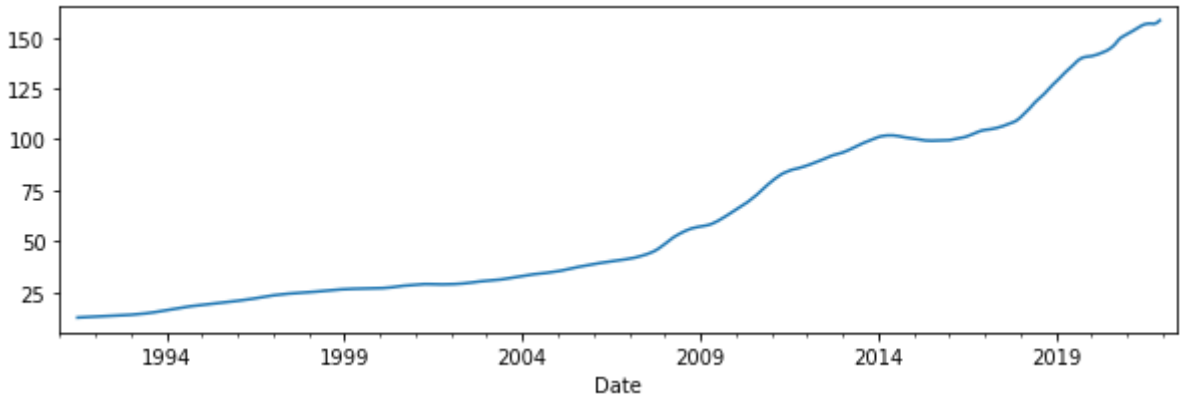
Series: REER

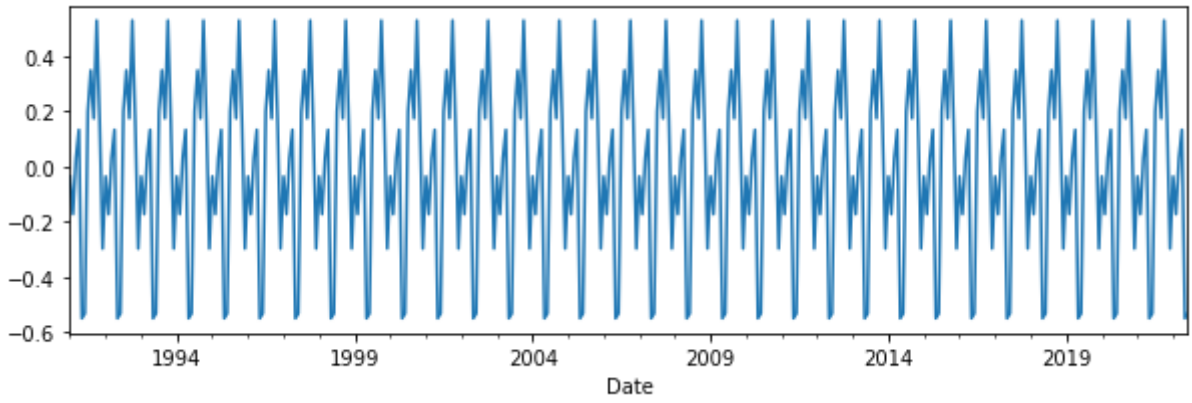

120
115
110
105
100
95
90
1994
1999
2004
2009
2014
2019
Date

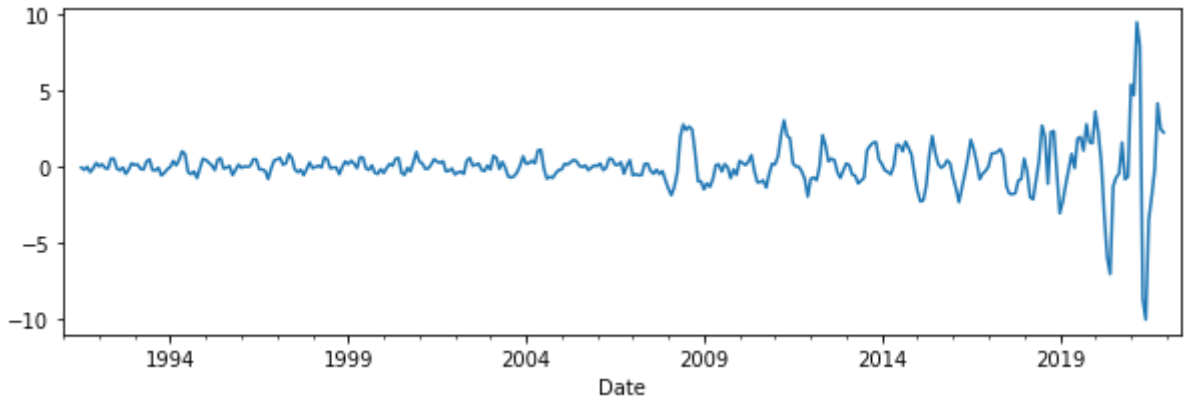

1.0
0.5
0.0
−0.5
1994
1999
2004
2009
2014
2019
Date



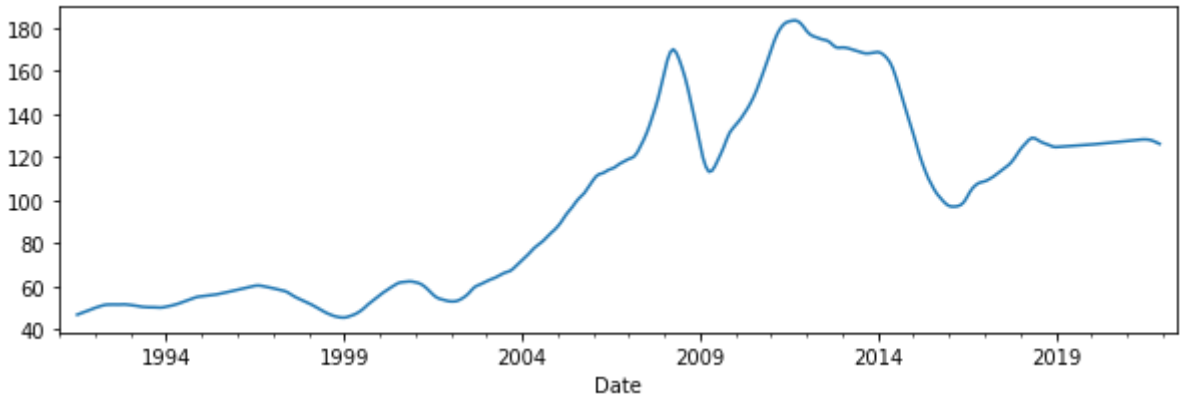
Series: WPI

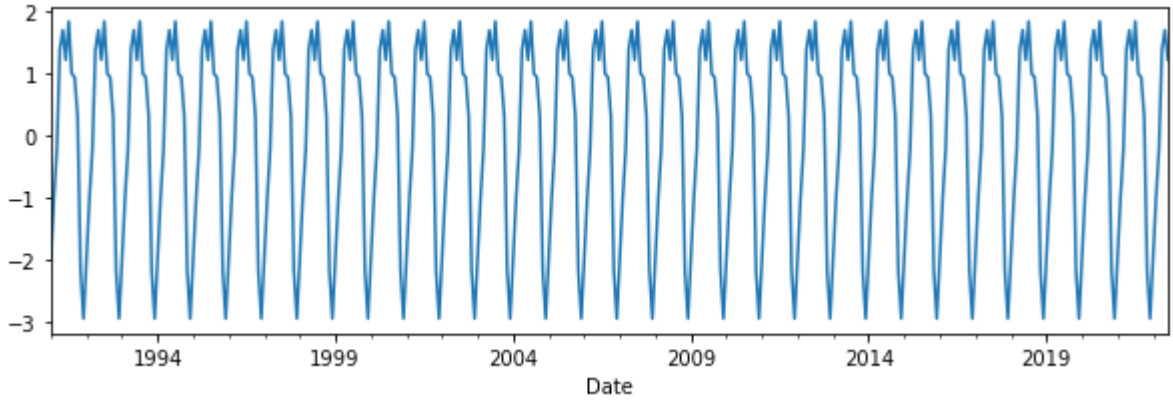


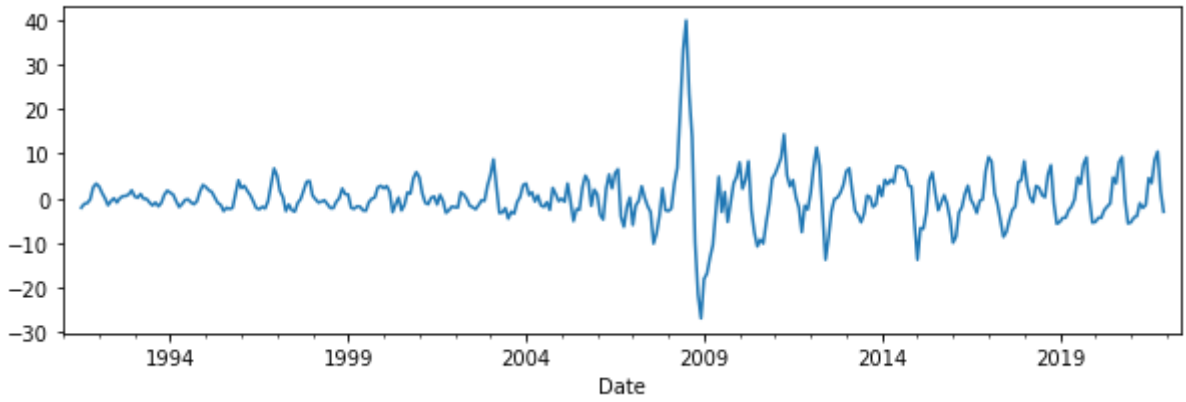




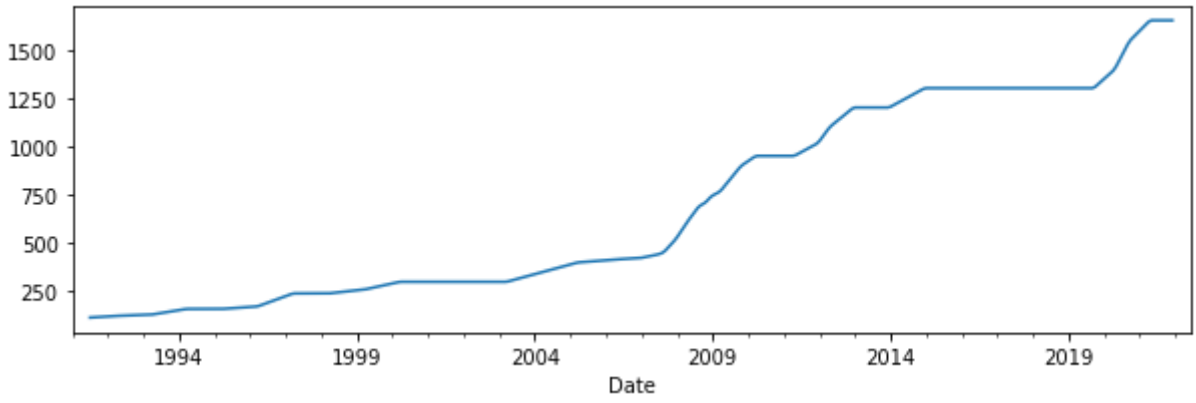
Series: WCPI

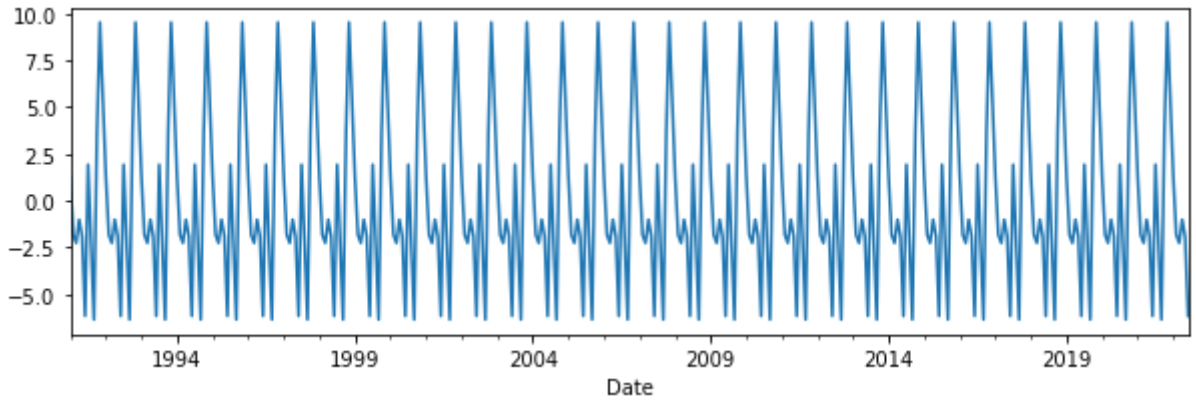

180
160
140
120
100
80
60
40
1994
1999
2004
2009
2014
2019
Date

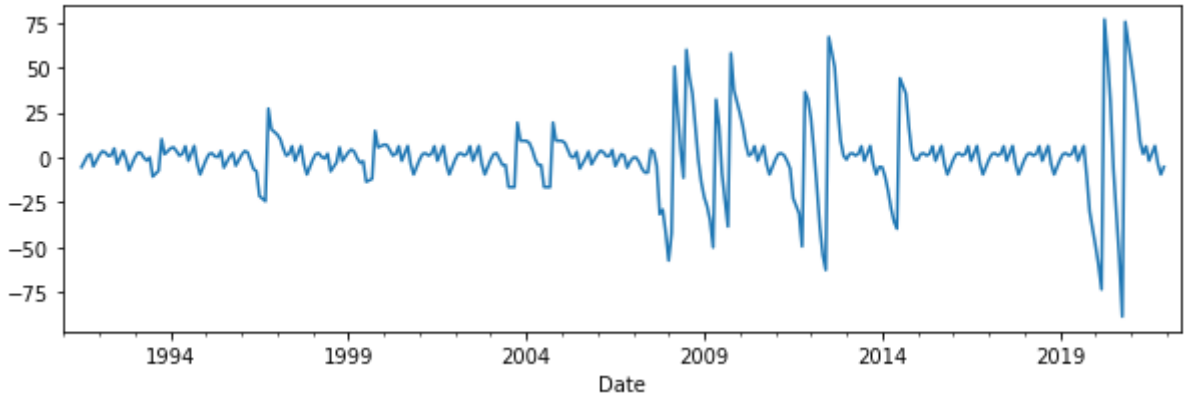

2
1
0
−1
−2
−3
1994
1999
2004
2009
2014
2019
Date


40
30
20
10
0
−10
−20
−30
1994
1999
2004
2009
2014
2019
Date

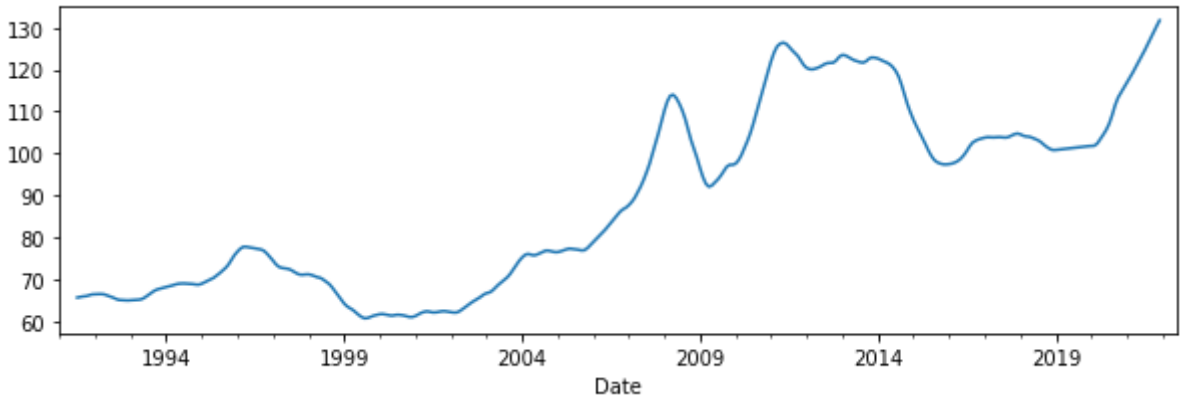
Series: WP

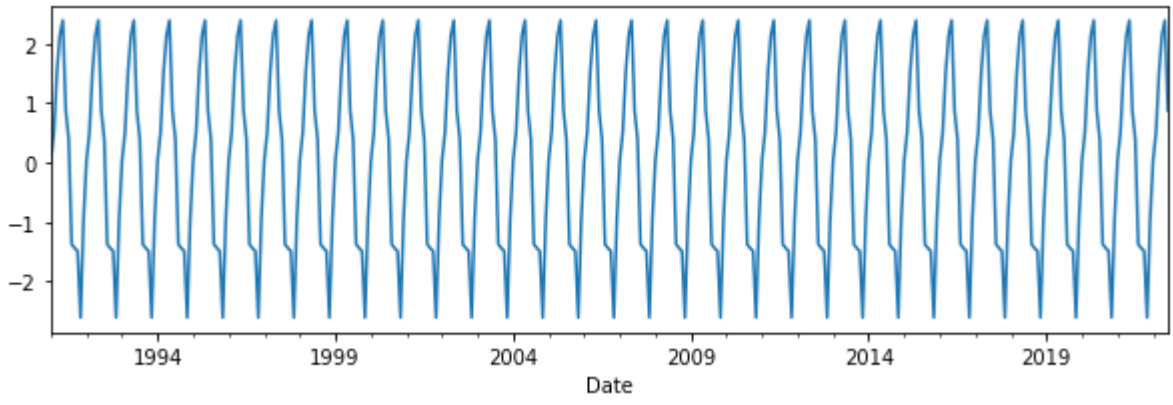

1500
1250
1000
750
500
250
1994
1999
2004
2009
2014
2019
Date

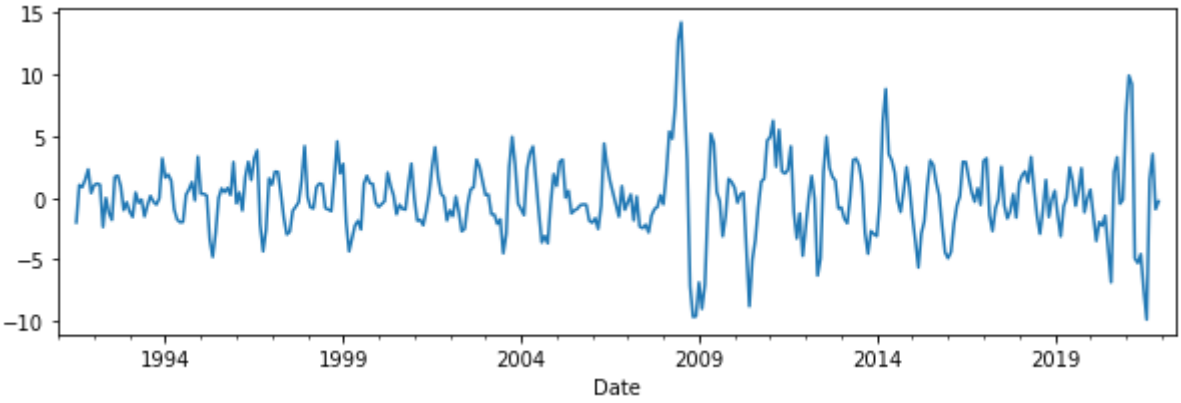




Series: Wfood






15
10
5
0
−5
−10
1994
1999
2004
2009
2014
2019
Date

In [74]:
```
## Testing statinarity using Agumented Dicky Fuler Unit Root Test
level = []
from statsmodels.tsa.stattools import adfuller
for (columnName, columnData) in df.iteritems():
    print('Series:',columnName)
    X = columnData.dropna().values
    result = adfuller(X)
    print('ADF Statistic: %f' % result[0])
    print('p-value: %f' % result[1])
    print('Critical Values:')
    for key, value in result[4].items():
        print('\t%s: %.3f' % (key, value))
    if (result[1]>0.05):
        print('\033[1m'+'\033[91m'+ "!Fail to reject null hypothesis, series not stat
ionary!"+'\033[0m')
    else:
        print('\033[1m'+'\033[92m'+ "!Null hypothesis rejected, series is stationar
y!"+'\033[0m')
        level.append(columnName)
    print()
    print()
```

```
Series: CPI
ADF Statistic: 2.644542
p-value: 0.999082
Critical Values:
        1%: -3.449
        5%: -2.870
        10%: -2.571
!Fail to reject null hypothesis, series not stationary!


Series: ER
ADF Statistic: 1.352190
p-value: 0.996886
Critical Values:
        1%: -3.448
        5%: -2.869
        10%: -2.571
!Fail to reject null hypothesis, series not stationary!


Series: M0
ADF Statistic: -1.207260
p-value: 0.670465
Critical Values:
        1%: -3.448
        5%: -2.870
        10%: -2.571
!Fail to reject null hypothesis, series not stationary!


Series: M2
ADF Statistic: 2.806514
p-value: 1.000000
Critical Values:
        1%: -3.449
        5%: -2.870
        10%: -2.571
!Fail to reject null hypothesis, series not stationary!


Series: RES
ADF Statistic: -0.898663
p-value: 0.788412
Critical Values:
        1%: -3.448
        5%: -2.869
        10%: -2.571
!Fail to reject null hypothesis, series not stationary!


Series: REM
ADF Statistic: 1.935820
p-value: 0.998584
Critical Values:
        1%: -3.449
        5%: -2.870
        10%: -2.571
!Fail to reject null hypothesis, series not stationary!


Series: IPLSM
ADF Statistic: -0.261657
p-value: 0.930762
Critical Values:
        1%: -3.448
```

```
        5%: -2.869
        10%: -2.571
!Fail to reject null hypothesis, series not stationary!


Series: TB
ADF Statistic: -2.511759
p-value: 0.112639
Critical Values:
        1%: -3.448
        5%: -2.869
        10%: -2.571
!Fail to reject null hypothesis, series not stationary!


Series: Disr
ADF Statistic: -1.730044
p-value: 0.415703
Critical Values:
        1%: -3.448
        5%: -2.869
        10%: -2.571
!Fail to reject null hypothesis, series not stationary!


Series: OIL
ADF Statistic: -1.999726
p-value: 0.286693
Critical Values:
        1%: -3.448
        5%: -2.869
        10%: -2.571
!Fail to reject null hypothesis, series not stationary!


Series: USIPI
ADF Statistic: -2.227164
p-value: 0.196534
Critical Values:
        1%: -3.448
        5%: -2.869
        10%: -2.571
!Fail to reject null hypothesis, series not stationary!


Series: PSC
ADF Statistic: 1.635438
p-value: 0.997960
Critical Values:
        1%: -3.448
        5%: -2.870
        10%: -2.571
!Fail to reject null hypothesis, series not stationary!


Series: PSB
ADF Statistic: 3.329441
p-value: 1.000000
Critical Values:
        1%: -3.449
        5%: -2.870
        10%: -2.571
!Fail to reject null hypothesis, series not stationary!


Series: LR
```

```
ADF Statistic: -2.383191
p-value: 0.146537
Critical Values:
        1%: -3.448
        5%: -2.869
        10%: -2.571
!Fail to reject null hypothesis, series not stationary!


Series: REER
ADF Statistic: -2.500290
p-value: 0.115404
Critical Values:
        1%: -3.448
        5%: -2.870
        10%: -2.571
!Fail to reject null hypothesis, series not stationary!


Series: WPI
ADF Statistic: 1.090616
p-value: 0.995136
Critical Values:
        1%: -3.449
        5%: -2.870
        10%: -2.571
!Fail to reject null hypothesis, series not stationary!


Series: WCPI
ADF Statistic: -1.795331
p-value: 0.382782
Critical Values:
        1%: -3.448
        5%: -2.869
        10%: -2.571
!Fail to reject null hypothesis, series not stationary!


Series: WP
ADF Statistic: 1.015790
p-value: 0.994430
Critical Values:
        1%: -3.448
        5%: -2.869
        10%: -2.571
!Fail to reject null hypothesis, series not stationary!


Series: Wfood
ADF Statistic: -0.639073
p-value: 0.861906
Critical Values:
        1%: -3.448
        5%: -2.870
        10%: -2.571
!Fail to reject null hypothesis, series not stationary!
```

All seen in the test all series are non stationary so taking first difference and then check the statinarity

In [75]:

```
first = []
for (columnName, columnData) in df.iteritems():
    print('Series:',columnName)
    X = columnData.diff().dropna().values
    result = adfuller(X)
    print('ADF Statistic: %f' % result[0])
    print('p-value: %f' % result[1])
    print('Critical Values:')
    for key, value in result[4].items():
        print('\t%s: %.3f' % (key, value))
    if (result[1]>0.05):
        print('\033[1m'+'\033[91m'+ "!Fail to reject null hypothesis, series not stat
ionary!"+'\033[0m')
    else:
        print('\033[1m'+'\033[92m'+ "!Null hypothesis rejected, series is stationar
y!"+'\033[0m')
        if columnName not in level:
            first.append(columnName)
    print()
    print()
```

```
Series: CPI
ADF Statistic: -1.396698
p-value: 0.583858
Critical Values:
        1%: -3.449
        5%: -2.870
        10%: -2.571
!Fail to reject null hypothesis, series not stationary!


Series: ER
ADF Statistic: -4.248181
p-value: 0.000546
Critical Values:
        1%: -3.448
        5%: -2.869
        10%: -2.571
!Null hypothesis rejected, series is stationary!


Series: M0
ADF Statistic: -1.692090
p-value: 0.435192
Critical Values:
        1%: -3.448
        5%: -2.870
        10%: -2.571
!Fail to reject null hypothesis, series not stationary!


Series: M2
ADF Statistic: -1.411675
p-value: 0.576666
Critical Values:
        1%: -3.449
        5%: -2.870
        10%: -2.571
!Fail to reject null hypothesis, series not stationary!


Series: RES
ADF Statistic: -4.624926
p-value: 0.000116
Critical Values:
        1%: -3.448
        5%: -2.869
        10%: -2.571
!Null hypothesis rejected, series is stationary!


Series: REM
ADF Statistic: -1.482494
p-value: 0.542153
Critical Values:
        1%: -3.449
        5%: -2.870
        10%: -2.571
!Fail to reject null hypothesis, series not stationary!


Series: IPLSM
ADF Statistic: -8.721265
p-value: 0.000000
Critical Values:
        1%: -3.448
```

```
        5%: -2.869
        10%: -2.571
!Null hypothesis rejected, series is stationary!


Series: TB
ADF Statistic: -8.800963
p-value: 0.000000
Critical Values:
        1%: -3.448
        5%: -2.869
        10%: -2.571
!Null hypothesis rejected, series is stationary!


Series: Disr
ADF Statistic: -8.954376
p-value: 0.000000
Critical Values:
        1%: -3.448
        5%: -2.869
        10%: -2.571
!Null hypothesis rejected, series is stationary!


Series: OIL
ADF Statistic: -10.916547
p-value: 0.000000
Critical Values:
        1%: -3.448
        5%: -2.869
        10%: -2.571
!Null hypothesis rejected, series is stationary!


Series: USIPI
ADF Statistic: -7.409308
p-value: 0.000000
Critical Values:
        1%: -3.448
        5%: -2.869
        10%: -2.571
!Null hypothesis rejected, series is stationary!


Series: PSC
ADF Statistic: -2.032761
p-value: 0.272396
Critical Values:
        1%: -3.448
        5%: -2.870
        10%: -2.571
!Fail to reject null hypothesis, series not stationary!


Series: PSB
ADF Statistic: -1.301949
p-value: 0.628280
Critical Values:
        1%: -3.449
        5%: -2.870
        10%: -2.571
!Fail to reject null hypothesis, series not stationary!


Series: LR
```

```
ADF Statistic: -6.094639
p-value: 0.000000
Critical Values:
        1%: -3.448
        5%: -2.869
        10%: -2.571
!Null hypothesis rejected, series is stationary!


Series: REER
ADF Statistic: -4.949283
p-value: 0.000028
Critical Values:
        1%: -3.448
        5%: -2.870
        10%: -2.571
!Null hypothesis rejected, series is stationary!


Series: WPI
ADF Statistic: -2.992795
p-value: 0.035573
Critical Values:
        1%: -3.449
        5%: -2.870
        10%: -2.571
!Null hypothesis rejected, series is stationary!


Series: WCPI
ADF Statistic: -12.032471
p-value: 0.000000
Critical Values:
        1%: -3.448
        5%: -2.869
        10%: -2.571
!Null hypothesis rejected, series is stationary!


Series: WP
ADF Statistic: -5.221231
p-value: 0.000008
Critical Values:
        1%: -3.448
        5%: -2.869
        10%: -2.571
!Null hypothesis rejected, series is stationary!


Series: Wfood
ADF Statistic: -5.834057
p-value: 0.000000
Critical Values:
        1%: -3.448
        5%: -2.870
        10%: -2.571
!Null hypothesis rejected, series is stationary!
```

In [76]:

```
sec = []
for (columnName, columnData) in df.iteritems():
    print('Series:',columnName)
    X = columnData.diff().diff().dropna().values
    result = adfuller(X)
    print('ADF Statistic: %f' % result[0])
    print('p-value: %f' % result[1])
    print('Critical Values:')
    for key, value in result[4].items():
        print('\t%s: %.3f' % (key, value))
    if (result[1]>0.1):
        print('\033[1m'+'\033[91m'+ "!Fail to reject null hypothesis, series not stat
ionary!"+'\033[0m')
    else:
        print('\033[1m'+'\033[92m'+ "!Null hypothesis rejected, series is stationar
y!"+'\033[0m')
        if (columnName not in first)&(columnName not in level):
            sec.append(columnName)
    print()
    print()
```

```
Series: CPI
ADF Statistic: -7.328748
p-value: 0.000000
Critical Values:
        1%: -3.449
        5%: -2.870
        10%: -2.571
!Null hypothesis rejected, series is stationary!


Series: ER
ADF Statistic: -10.329083
p-value: 0.000000
Critical Values:
        1%: -3.448
        5%: -2.869
        10%: -2.571
!Null hypothesis rejected, series is stationary!


Series: M0
ADF Statistic: -14.449297
p-value: 0.000000
Critical Values:
        1%: -3.448
        5%: -2.870
        10%: -2.571
!Null hypothesis rejected, series is stationary!


Series: M2
ADF Statistic: -4.512905
p-value: 0.000186
Critical Values:
        1%: -3.449
        5%: -2.870
        10%: -2.571
!Null hypothesis rejected, series is stationary!


Series: RES
ADF Statistic: -6.819986
p-value: 0.000000
Critical Values:
        1%: -3.449
        5%: -2.870
        10%: -2.571
!Null hypothesis rejected, series is stationary!


Series: REM
ADF Statistic: -11.243084
p-value: 0.000000
Critical Values:
        1%: -3.449
        5%: -2.870
        10%: -2.571
!Null hypothesis rejected, series is stationary!


Series: IPLSM
ADF Statistic: -11.203085
p-value: 0.000000
Critical Values:
        1%: -3.449
```

```
        5%: -2.870
        10%: -2.571
!Null hypothesis rejected, series is stationary!


Series: TB
ADF Statistic: -9.011405
p-value: 0.000000
Critical Values:
        1%: -3.448
        5%: -2.870
        10%: -2.571
!Null hypothesis rejected, series is stationary!


Series: Disr
ADF Statistic: -8.803752
p-value: 0.000000
Critical Values:
        1%: -3.448
        5%: -2.870
        10%: -2.571
!Null hypothesis rejected, series is stationary!


Series: OIL
ADF Statistic: -9.471137
p-value: 0.000000
Critical Values:
        1%: -3.449
        5%: -2.870
        10%: -2.571
!Null hypothesis rejected, series is stationary!


Series: USIPI
ADF Statistic: -8.702882
p-value: 0.000000
Critical Values:
        1%: -3.449
        5%: -2.870
        10%: -2.571
!Null hypothesis rejected, series is stationary!


Series: PSC
ADF Statistic: -12.607570
p-value: 0.000000
Critical Values:
        1%: -3.448
        5%: -2.870
        10%: -2.571
!Null hypothesis rejected, series is stationary!


Series: PSB
ADF Statistic: -9.787523
p-value: 0.000000
Critical Values:
        1%: -3.449
        5%: -2.870
        10%: -2.571
!Null hypothesis rejected, series is stationary!


Series: LR
```

```
ADF Statistic: -7.230152
p-value: 0.000000
Critical Values:
        1%: -3.449
        5%: -2.870
        10%: -2.571
!Null hypothesis rejected, series is stationary!


Series: REER
ADF Statistic: -9.200626
p-value: 0.000000
Critical Values:
        1%: -3.448
        5%: -2.870
        10%: -2.571
!Null hypothesis rejected, series is stationary!


Series: WPI
ADF Statistic: -7.385832
p-value: 0.000000
Critical Values:
        1%: -3.449
        5%: -2.870
        10%: -2.571
!Null hypothesis rejected, series is stationary!


Series: WCPI
ADF Statistic: -8.896420
p-value: 0.000000
Critical Values:
        1%: -3.449
        5%: -2.870
        10%: -2.571
!Null hypothesis rejected, series is stationary!


Series: WP
ADF Statistic: -11.194629
p-value: 0.000000
Critical Values:
        1%: -3.448
        5%: -2.869
        10%: -2.571
!Null hypothesis rejected, series is stationary!


Series: Wfood
ADF Statistic: -8.616358
p-value: 0.000000
Critical Values:
        1%: -3.449
        5%: -2.870
        10%: -2.571
!Null hypothesis rejected, series is stationary!
```

All series are stationary almost second difference

In [77]:
```
print('Series Stationary at Level',level)
print('Series Stationary at First difference',first)
print('Series Stationary at Second difference',sec)
```

```
Series Stationary at Level []
Series Stationary at First difference ['ER', 'RES', 'IPLSM', 'TB', 'Disr', 'OIL', 'U
SIPI', 'LR', 'REER', 'WPI', 'WCPI', 'WP', 'Wfood']
Series Stationary at Second difference ['CPI', 'M0', 'M2', 'REM', 'PSC', 'PSB']
```

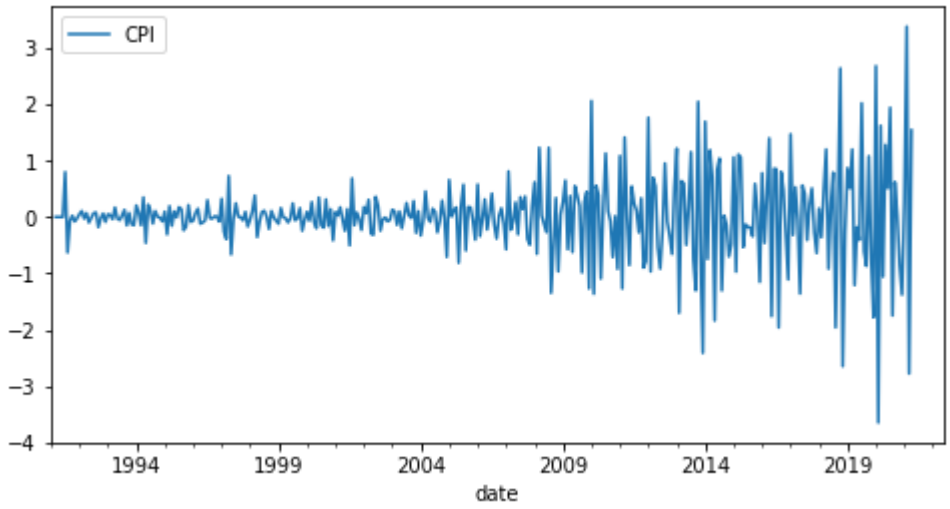
In [112]:
```
# Making another dataframe with statonary series
d = {'date':pd.Series(pd.period_range("1-1-1991","6-1-2022",freq="M"))}
dfnew = pd.DataFrame(data=d)
for (columnName, columnData) in df.iteritems():
    if columnName in level:
        dfnew[columnName] = columnData.values
    elif columnName in first:
        dfnew[columnName] = columnData.diff().values
    else:
        dfnew[columnName] = columnData.diff().diff().values
```

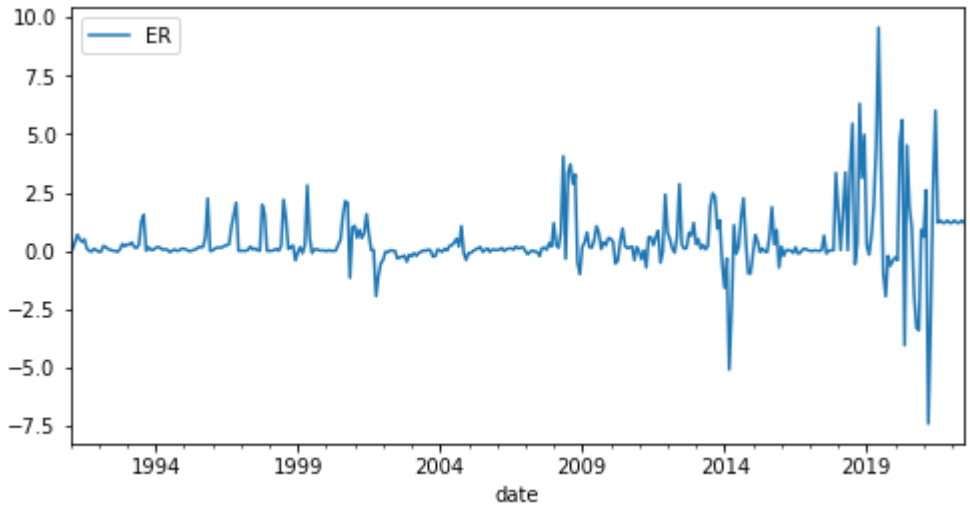
In [108]:
```
df_details(dfnew,5)
```

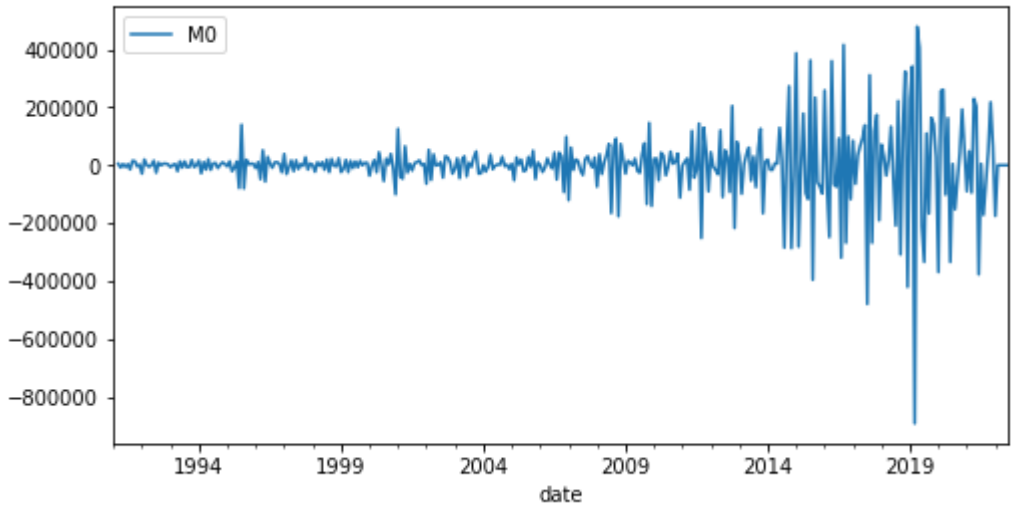
```
Data Types of Column:
 CPI      float64
ER       float64
M0       float64
M2       float64
RES      float64
REM      float64
IPLSM    float64
TB       float64
Disr     float64
OIL      float64
USIPI    float64
PSC      float64
PSB      float64
LR       float64
REER     float64
WPI      float64
WCPI     float64
WP       float64
Wfood    float64
dtype: object

 Size of Datarame:  (376, 19)

 Top and bottom  5  rows:
```

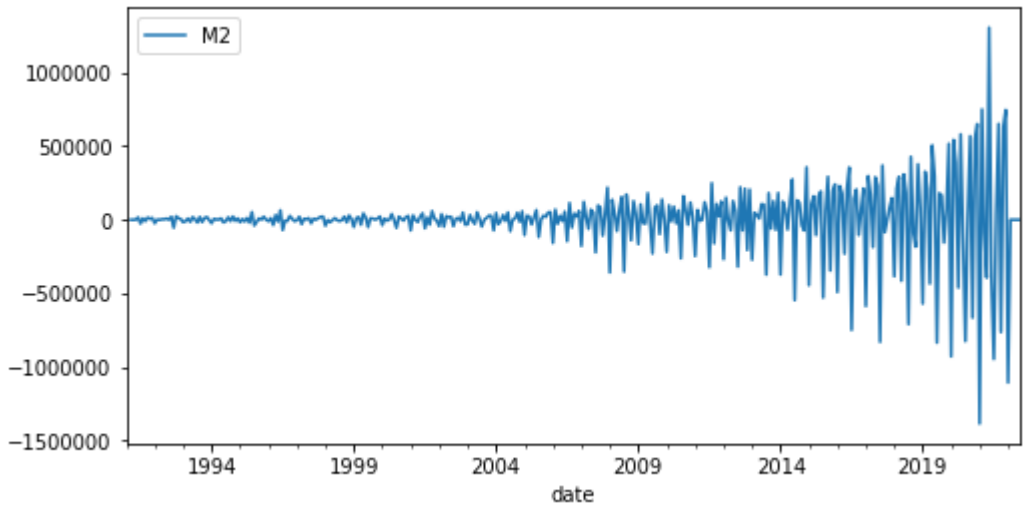
date	CPI	ER	M0	M2	RES	REM	IPLSM	TB	Disr	OIL	USI
1991-03	0.000000	0.353682	5024.0	-365.0	0.000000	5.410000	1.245816	0.0000	0.0	0.40	-(
1991-04	0.000000	0.691097	-8031.0	420.0	0.000000	9.950000	-8.198276	1.0400	0.0	0.50	(
1991-05	0.000000	0.491853	3966.0	390.0	0.000000	8.960000	-0.984597	0.5750	0.0	0.55	(
1991-06	0.000000	0.382066	-6180.0	16590.0	-61.000000	4.180000	-3.174823	-0.0217	0.0	-0.50	(
1991-07	0.798256	0.493317	3618.0	-25887.0	0.000000	-30.270000	2.049569	0.0517	0.0	0.85	(
2022-02	NaN	1.300000	0.0	0.0	386.365255	0.298018	-0.537200	0.0000	0.0	0.00	(
2022-03	NaN	1.200000	0.0	0.0	392.547099	0.300998	3.522600	0.0000	0.0	0.00	(
2022-04	NaN	1.200000	0.0	0.0	398.827852	0.304008	-18.706500	0.0000	0.0	0.00	(
2022-05	NaN	1.300000	0.0	0.0	405.209098	0.307049	-4.883100	0.0000	0.0	0.00	(
2022-06	NaN	1.200000	0.0	0.0	411.692444	0.310119	-5.336800	0.0000	0.0	0.00	(

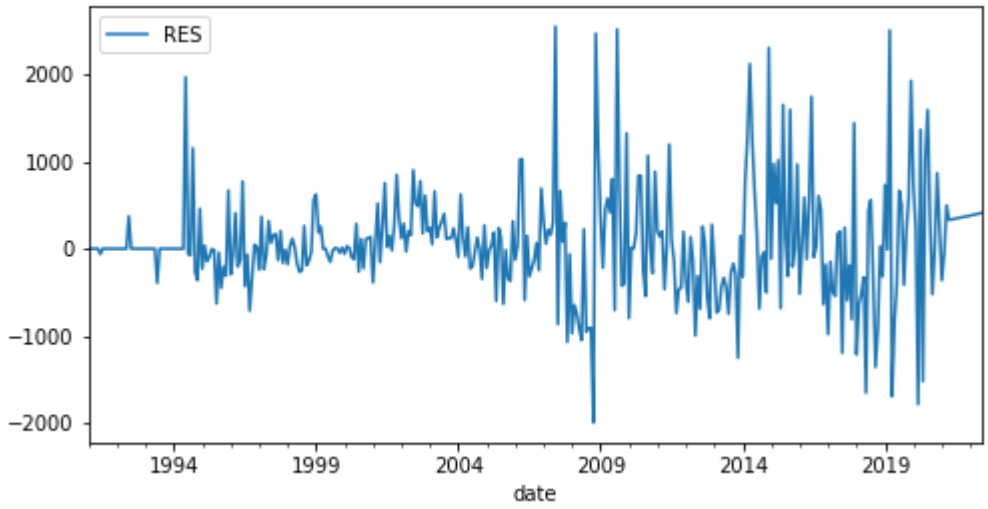
In [80]:
```
#Visiallizing all after converting them to stationary
## Line plot all sereis
for (columnName, columnData) in dfnew.iloc[:,1:].iteritems():
    dfnew.plot(x = 'date',y=columnName,figsize=(8,4))
    plt.show()
```

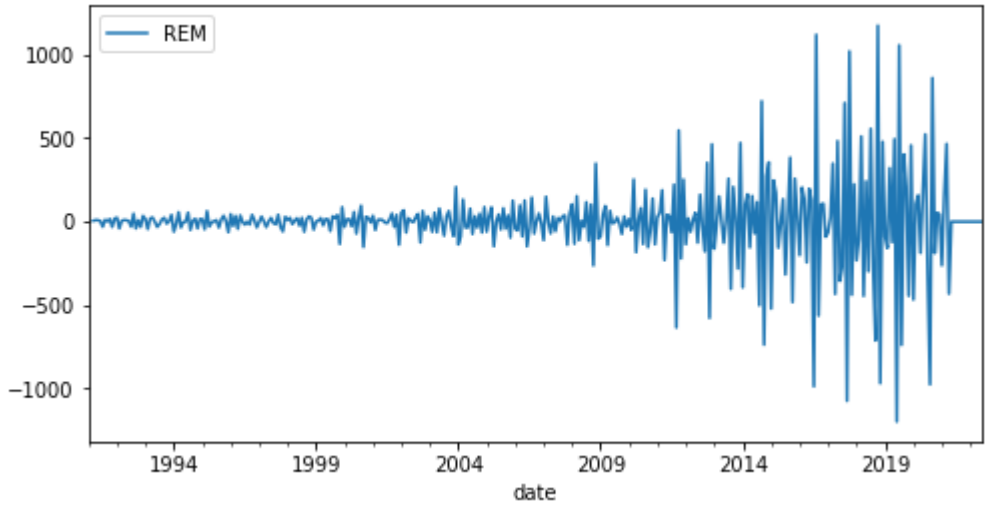

CPI
3
2
1
0
−1
−2
−3
−4
1994
1999
2004
2009
2014
2019
date

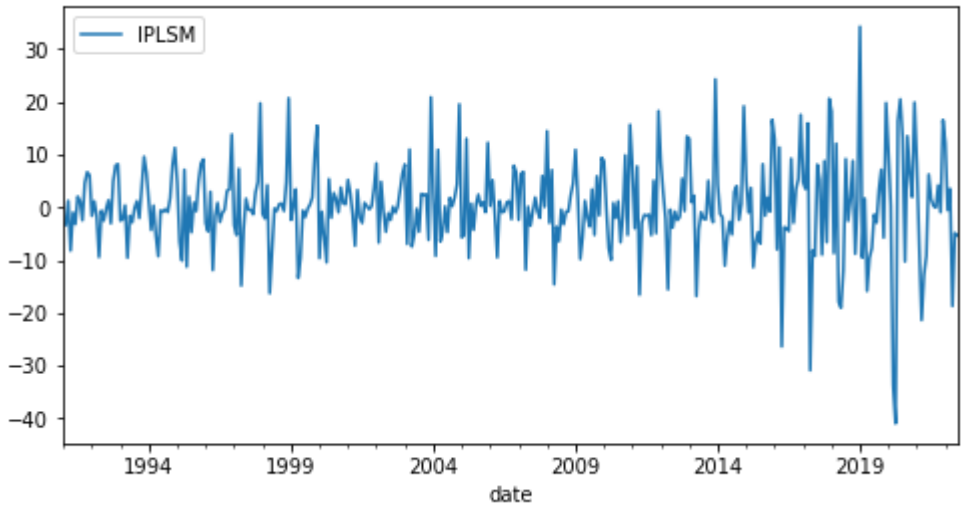

ER
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
1994
1999
2004
2009
2014
2019
date

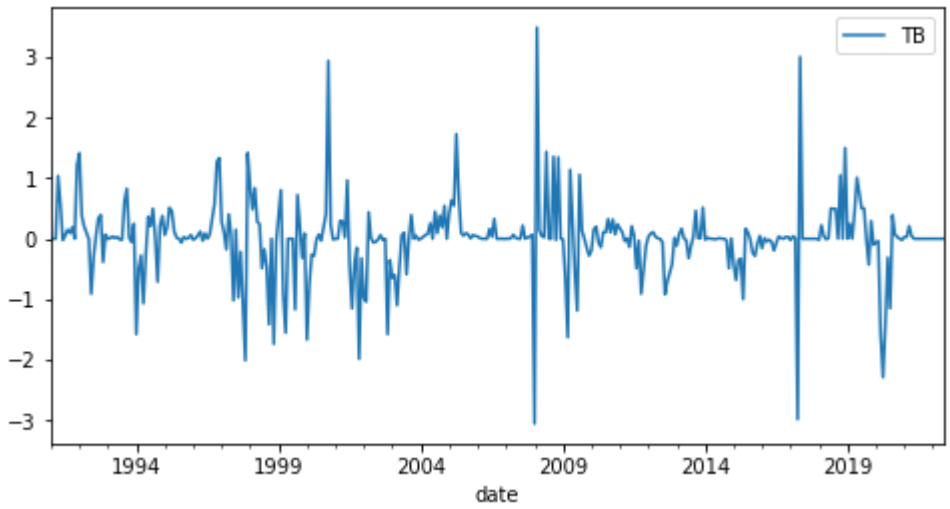

M0
400000
200000
0
−200000
−400000
−600000
−800000
1994
1999
2004
2009
2014
2019
date

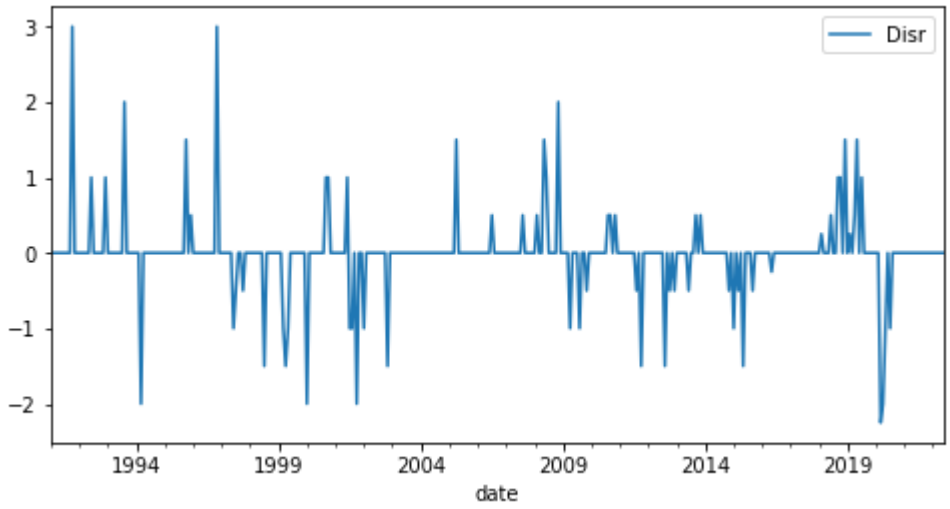

M2
1000000
500000
0
−500000
−1000000
−1500000
1994
1999
2004
2009
2014
2019
date

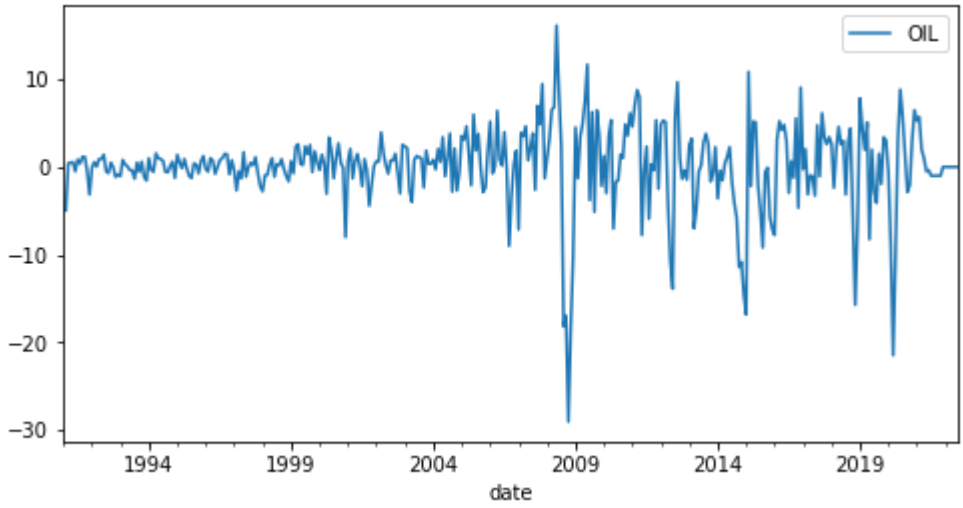

RES
2000
1000
0
−1000
−2000
1994
1999
2004
2009
2014
2019
date

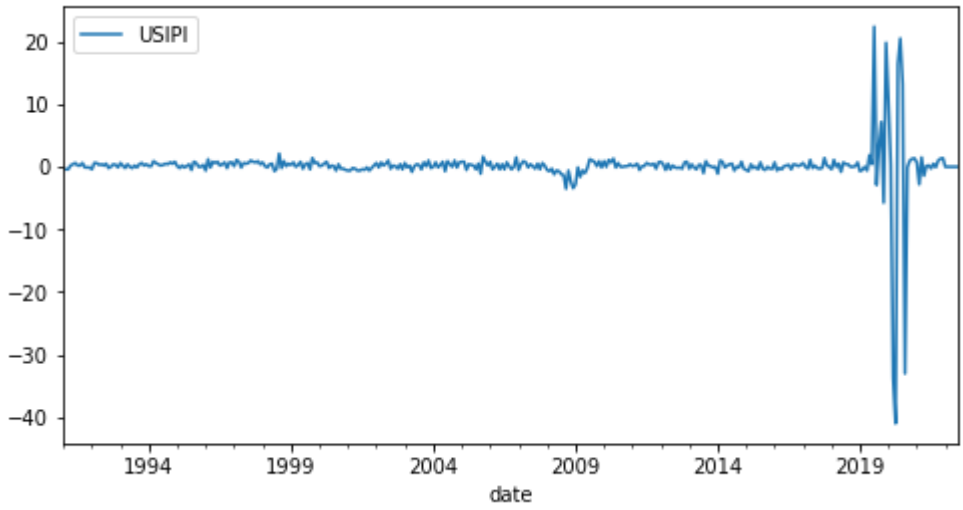

REM
1000
500
0
−500
−1000
1994
1999
2004
2009
2014
2019
date

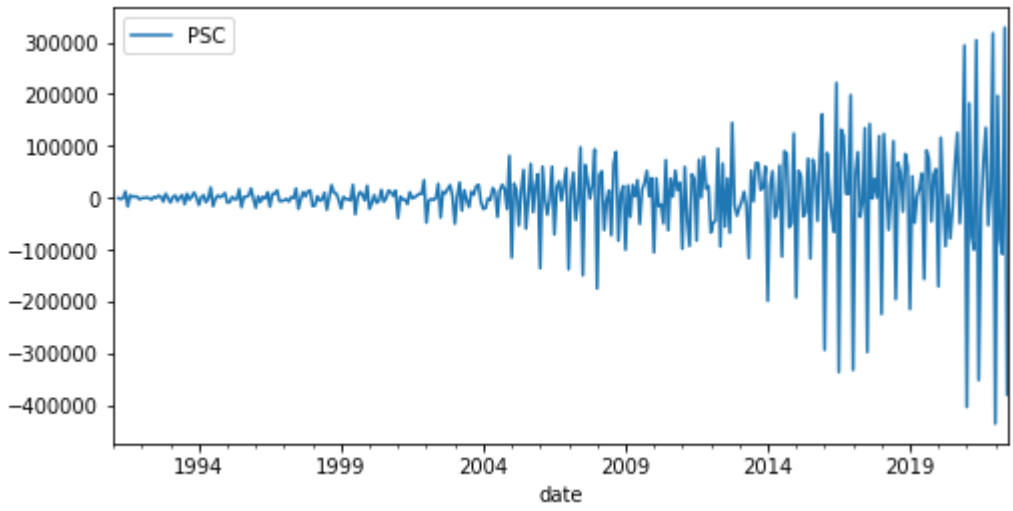

IPLSM
30
20
10
0
−10
−20
−30
−40
1994
1999
2004
2009
2014
2019
date

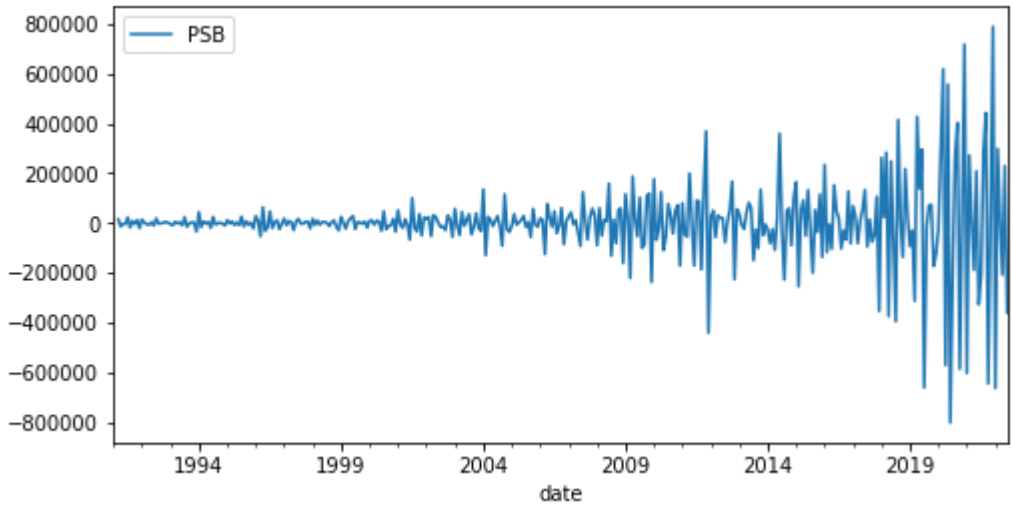

TB
3
2
1
0
−1
−2
−3
1994
1999
2004
2009
2014
2019
date

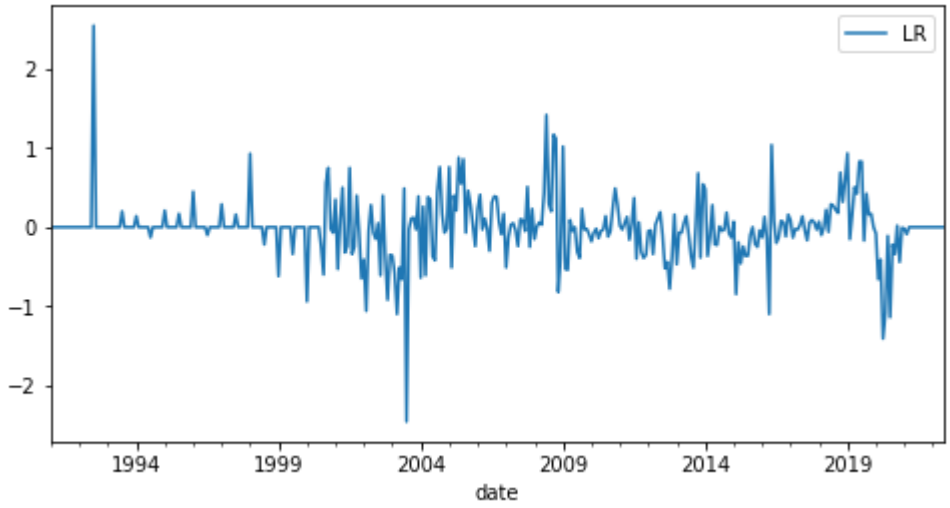

Disr
3
2
1
0
−1
−2
1994
1999
2004
2009
2014
2019
date

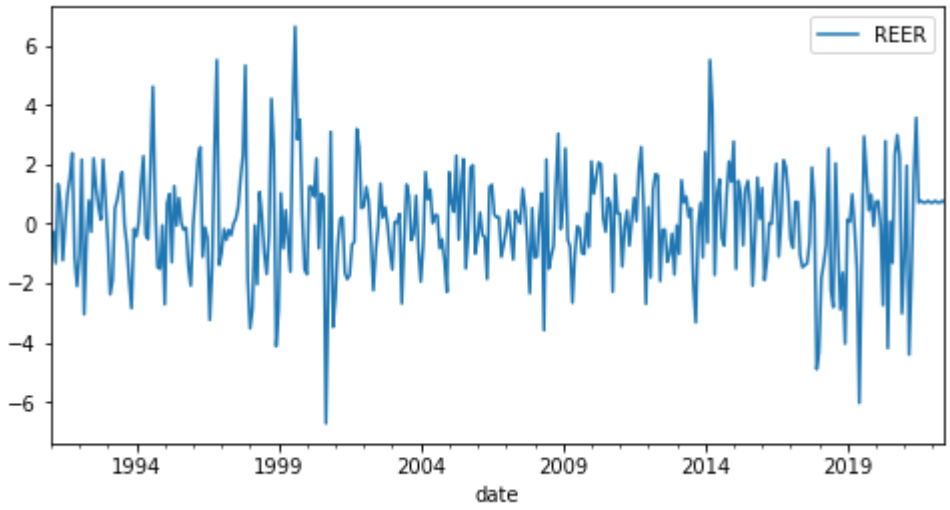

OIL
10
0
−10
−20
−30
1994
1999
2004
2009
2014
2019
date

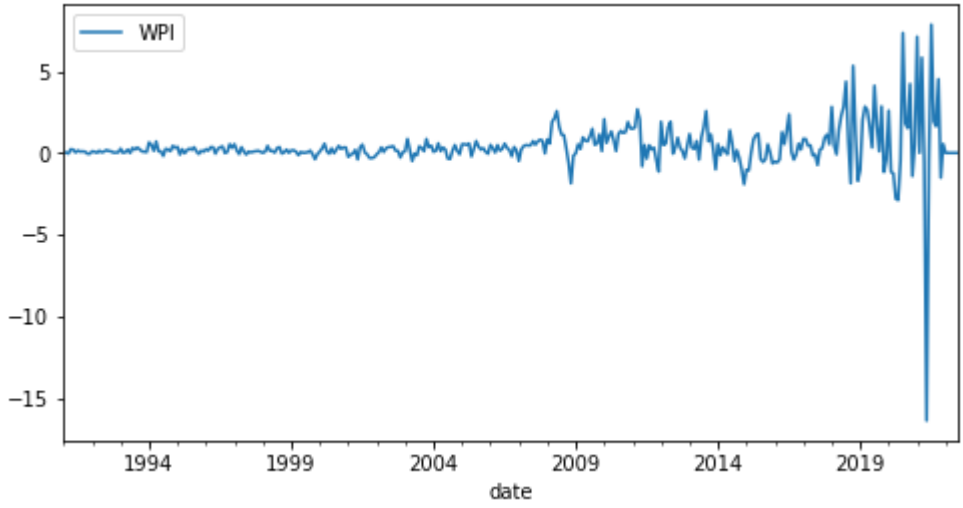

USIPI
20
10
0
−10
−20
−30
−40
1994
1999
2004
2009
2014
2019
date

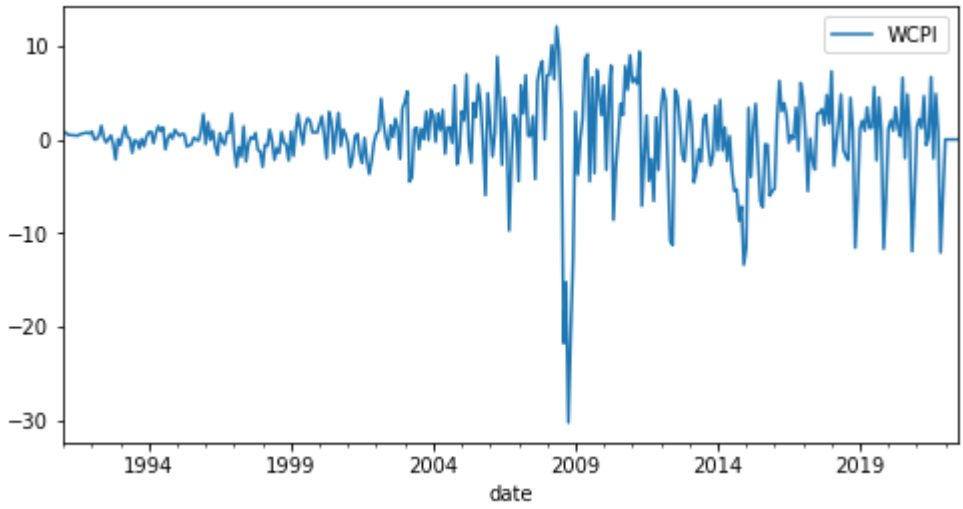

PSC
300000
200000
100000
0
−100000
−200000
−300000
−400000
1994
1999
2004
2009
2014
2019
date

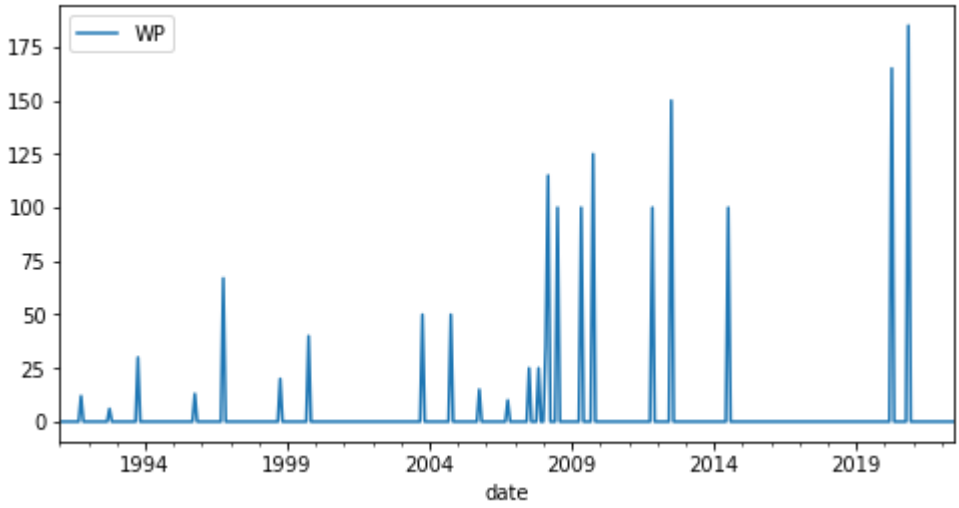

PSB
800000
600000
400000
200000
0
−200000
−400000
−600000
−800000
1994
1999
2004
2009
2014
2019
date

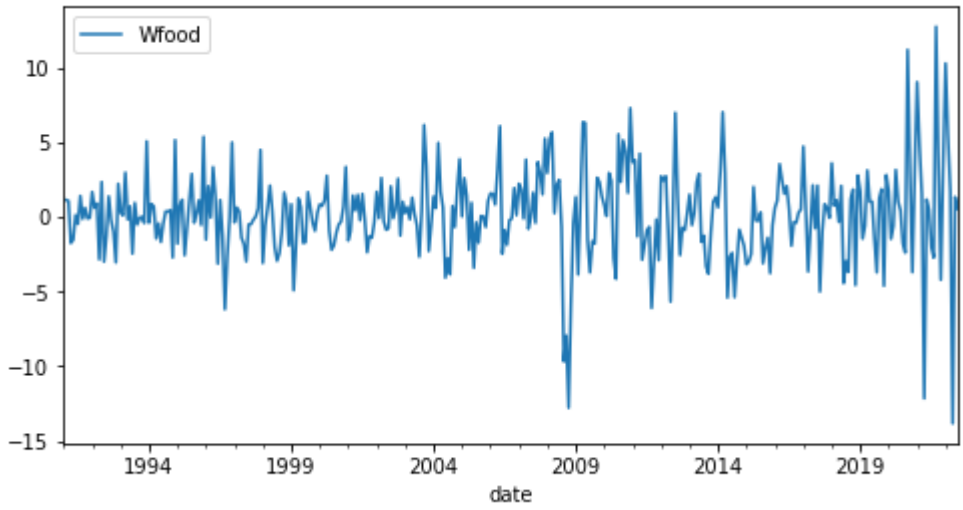

LR
2
1
0
−1
−2
1994
1999
2004
2009
2014
2019
date


REER
6
4
2
0
−2
−4
−6
1994
1999
2004
2009
2014
2019
date


WPI
5
0
−5
−10
−15
1994
1999
2004
2009
2014
2019
date

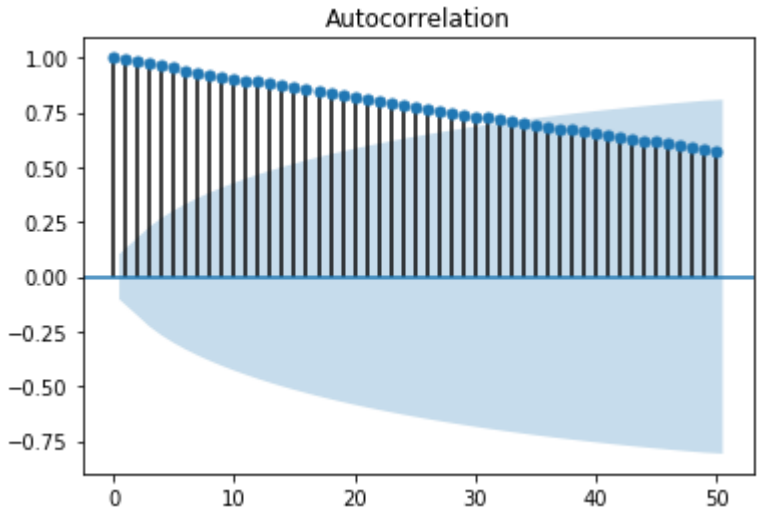




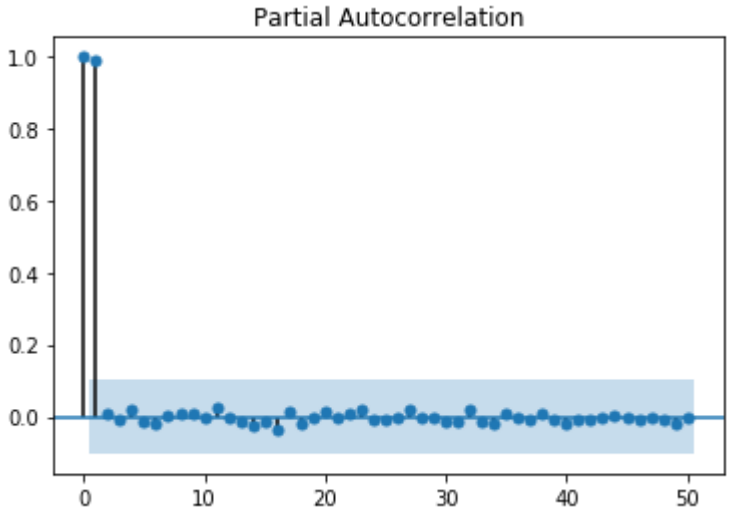


In [91]:
```
## Droping two columns with NA due to differencing
dfnew = dfnew.drop([0,1],axis=0)
```

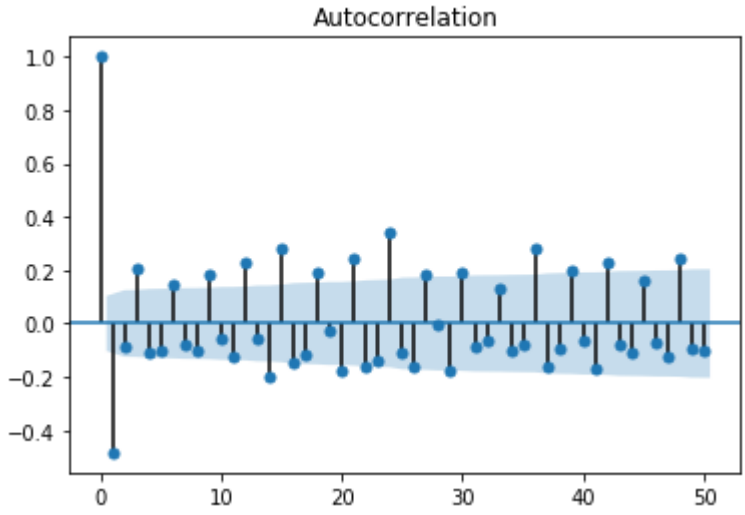
In [96]:
```
## Ploting Autocorelation Plot
plot_acf(df['CPI'].dropna(),lags=50)
plt.show()
```



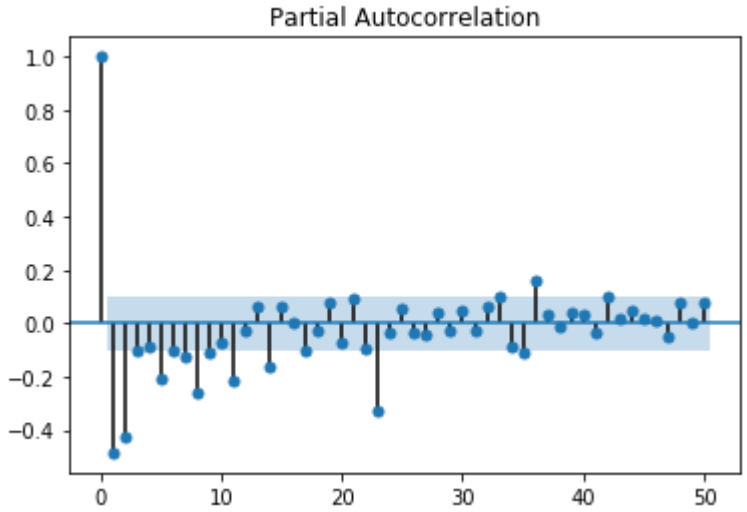
In [97]:
```
## Ploting Partial Autocorelation Plot
from statsmodels.graphics.tsaplots import plot_pacf
plot_pacf(df['CPI'].dropna(), lags=50)
plt.show()
```



In [98]:
```
## Ploting of stationary seires Autocorelation Plot
plot_acf(dfnew['CPI'].dropna(),lags=50)
plt.show()
```



In [99]:
```
## Ploting Partial Autocorelation Plot of stationary series
from statsmodels.graphics.tsaplots import plot_pacf
plot_pacf(dfnew['CPI'].dropna(), lags=50)
plt.show()
```



As per ACf the order of MA is 1 and order of AR by PACF is 2

In [120]:
```
# Index dates
dfnew = dfnew.set_index('date')
```

In [121]:
```
# Dropping NA values of CPI
dfnew.dropna(axis=0,inplace=True)
dfnew.tail(4)
```

Out[121]:

date	CPI	ER	M0	M2	RES	REM	IPLSM	TB	Disr	OIL	USIPI
2021-01	0.27	0.6	-90040.297941	-1.387183e+06	-362.600	-261.800	8.280000	0.03	0.0	5.27	0.6872
2021-02	3.38	2.6	48453.996922	7.517130e+05	-44.100	155.800	-7.280000	0.03	0.0	5.71	-2.8015
2021-03	-2.78	-7.4	-95054.407421	-8.648629e+04	498.700	466.500	-21.372833	0.21	0.0	2.00	1.4992
2021-04	1.54	-2.8	229696.325371	-3.941732e+05	329.656	-431.651	-12.812932	0.05	0.0	1.00	-1.4330

In [122]:
```
dfnew.iloc[:-12]
```

Out[122]:

date	CPI	ER	M0	M2	RES	REM	IPLSM	TB	D
1991-03	0.000000	0.353682	5024.000000	-365.000000	0.000000	5.41	1.245816	0.0000	0
1991-04	0.000000	0.691097	-8031.000000	420.000000	0.000000	9.95	-8.198276	1.0400	0
1991-05	0.000000	0.491853	3966.000000	390.000000	0.000000	8.96	-0.984597	0.5750	0
1991-06	0.000000	0.382066	-6180.000000	16590.000000	-61.000000	4.18	-3.174823	-0.0217	0
1991-07	0.798256	0.493317	3618.000000	-25887.000000	0.000000	-30.27	2.049569	0.0517	0
...	...	...	...	...	...	...	...	...	
2019-12	-1.780000	-0.421376	29323.605226	517451.168060	1926.642183	457.66	19.793368	-0.1002	0
2020-01	2.680000	-0.283081	-369295.278191	-930026.144812	728.104732	-466.71	10.590205	-0.0601	0
2020-02	-3.650000	-0.405715	256294.899081	543576.604159	222.887042	106.99	0.224914	-0.0397	0
2020-03	1.620000	4.719111	260987.822056	386034.868038	-1783.300000	158.79	-33.699820	-1.5899	-2
2020-04	-1.060000	5.598382	-101885.043803	-462362.176512	1365.300000	-187.12	-40.955268	-2.2900	-2

350 rows × 19 columns

Using Exponantial Smoothing

In [125]:
```
from statsmodels.tsa.holtwinters import ExponentialSmoothing
# prepare data
data = dfnew['CPI'].iloc[:-12].dropna()
# create class
model = ExponentialSmoothing(data,trend='add', seasonal='add',seasonal_periods=12)
# fit model
model_fit = model.fit()
predic = model_fit.forecast(12)
```

In [127]:
```
model_fit.summary()
```

Out[127]:

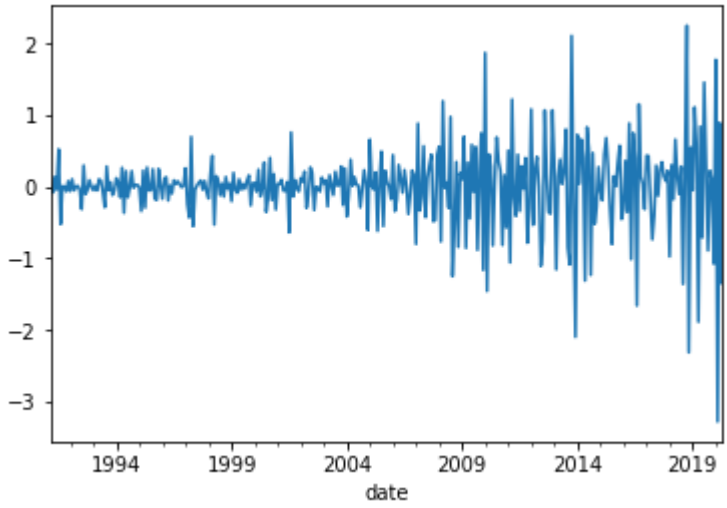
ExponentialSmoothing Model Results

Dep. Variable:	CPI	**No. Observations:**	350
Model:	ExponentialSmoothing	**SSE**	113.286
Optimized:	True	**AIC**	-362.806
Trend:	Additive	**BIC**	-301.079
Seasonal:	Additive	**AICC**	-360.739
Seasonal Periods:	12	**Date:**	Sat, 29 May 2021
Box-Cox:	False	**Time:**	21:25:08
Box-Cox Coeff.:	None		

	coeff	code	optimized
smoothing_level	1.4901e-08	alpha	True
smoothing_trend	1.4884e-08	beta	True
smoothing_seasonal	0.2082108	gamma	True
initial_level	0.3792064	l.0	True
initial_trend	-0.0001545	b.0	True
initial_seasons.0	-0.3454039	s.0	True
initial_seasons.1	-0.2987527	s.1	True
initial_seasons.2	-0.5167143	s.2	True
initial_seasons.3	-0.3395160	s.3	True
initial_seasons.4	-0.1048860	s.4	True
initial_seasons.5	-0.4863437	s.5	True
initial_seasons.6	-0.4577799	s.6	True
initial_seasons.7	-0.3572026	s.7	True
initial_seasons.8	-0.3946075	s.8	True
initial_seasons.9	-0.4808243	s.9	True
initial_seasons.10	-0.2668699	s.10	True
initial_seasons.11	-0.3777961	s.11	True

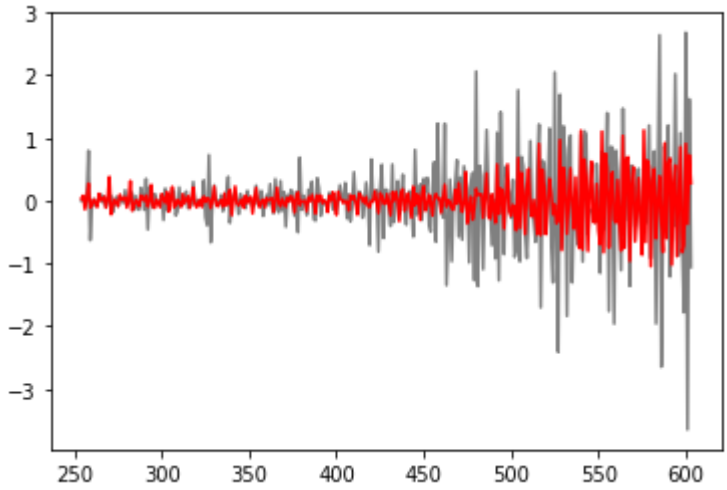
In [129]:
```
# Residual Plot
model_fit.resid.plot()
```

Out[129]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x15bef603390>
```



In [138]:
```
#Ploting Fitted and Actual values
plt.plot(dfnew.index[:-12],dfnew['CPI'].iloc[:-12],color="gray")
plt.plot(dfnew.index[:-12],model_fit.fittedvalues,color="red")
```

Out[138]:
```
[<matplotlib.lines.Line2D at 0x15bf2ec6550>]
```



In [133]:
```
from sklearn.metrics import mean_squared_error
rms = mean_squared_error(dfnew['CPI'].iloc[-12:], predic, squared=False)
rms1 = mean_squared_error(dfnew['CPI'].iloc[:-12], model_fit.fittedvalues, squared=Fa
lse)
print("RMSE of Fitted: ",rms1)
print("RMSE of out of Sample forecast: ",rms)
```

```
RMSE of Fitted:  0.5689239610259027
RMSE of out of Sample forecast:  1.918042278415232
```

Using ARIMA for forcasting

AR

In [140]:
```
from statsmodels.tsa.arima.model import ARIMA
data = dfnew['CPI'].iloc[:-12].dropna()
# AR model
model = ARIMA(data, order=(2,0,0))
model_fit = model.fit()
predic = model_fit.forecast(12)
model_fit.summary()
```

Out[140]:

SARIMAX Results

Dep. Variable:	CPI	**No. Observations:**	350
Model:	ARIMA(2, 0, 0)	**Log Likelihood**	-277.588
Date:	Sat, 29 May 2021	**AIC**	563.176
Time:	22:44:43	**BIC**	578.608
Sample:	03-31-1991	**HQIC**	569.319
	- 04-30-2020		
Covariance Type:	opg		

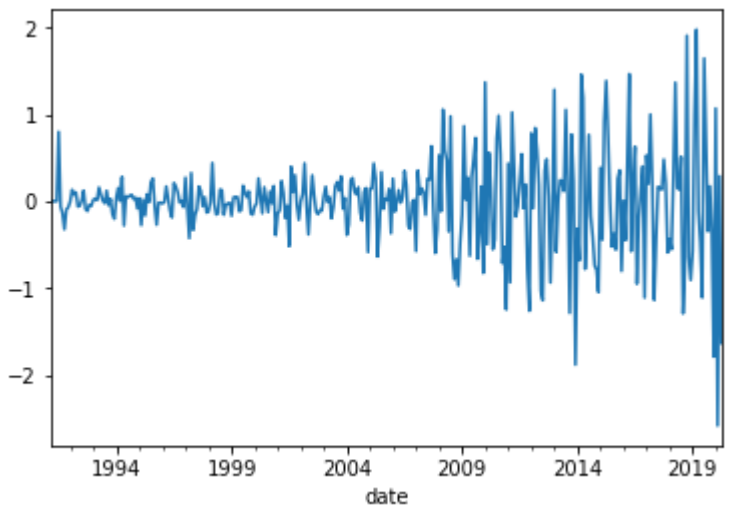
	coef	std err	z	P>|z|	[0.025	0.975]
const	-0.0019	0.013	-0.145	0.885	-0.028	0.024
ar.L1	-0.7121	0.035	-20.359	0.000	-0.781	-0.644
ar.L2	-0.4731	0.030	-15.838	0.000	-0.532	-0.415
sigma2	0.2854	0.014	20.638	0.000	0.258	0.312

Ljung-Box (L1) (Q):	0.38	**Jarque-Bera (JB):**	159.27
Prob(Q):	0.54	**Prob(JB):**	0.00
Heteroskedasticity (H):	27.35	**Skew:**	-0.17
Prob(H) (two-sided):	0.00	**Kurtosis:**	6.29

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

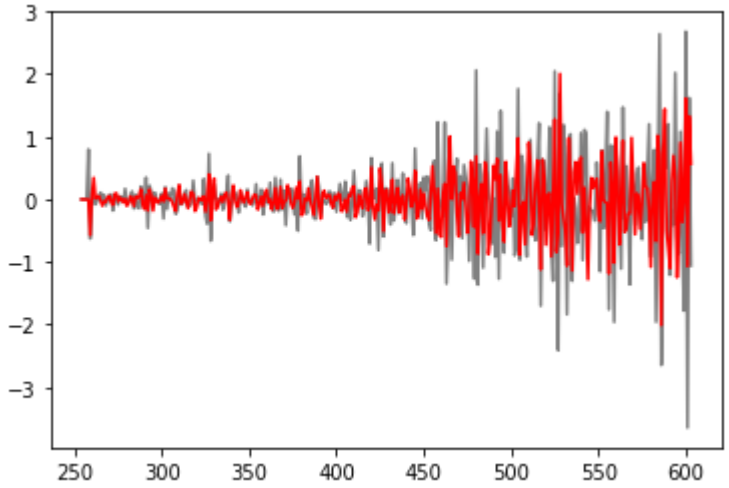
In [141]:
```
# Residual Plot
model_fit.resid.plot()
```

Out[141]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x15bf28737f0>
```



In [142]:
```
#Ploting Fitted and Actual values
plt.plot(dfnew.index[:-12],dfnew['CPI'].iloc[:-12],color="gray")
plt.plot(dfnew.index[:-12],model_fit.fittedvalues,color="red")
```

Out[142]:
```
[<matplotlib.lines.Line2D at 0x15bf3a48898>]
```



In [143]:
```
rms = mean_squared_error(dfnew['CPI'].iloc[-12:], predic, squared=False)
rms1 = mean_squared_error(dfnew['CPI'].iloc[:-12], model_fit.fittedvalues, squared=Fa
lse)
print("RMSE of Fitted: ",rms1)
print("RMSE of out of Sample forecast: ",rms)
```

```
RMSE of Fitted:  0.5342272496856852
RMSE of out of Sample forecast:  1.7029121116495982
```

MA

In [202]:
```
data = dfnew['CPI'].iloc[:-12].dropna()
# MA model
model = ARIMA(data, order=(0,0,1))
model_fit = model.fit()
predic = model_fit.forecast(12)
model_fit.summary()
```

Out[202]:

SARIMAX Results

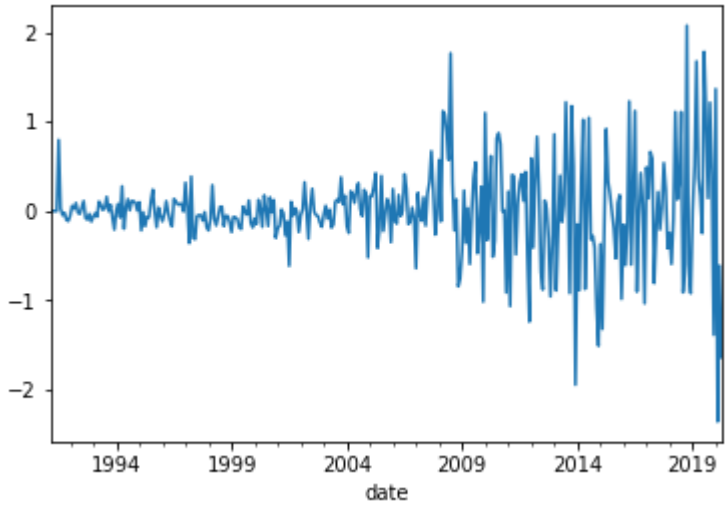
Dep. Variable:	CPI	**No. Observations:**	350
Model:	ARIMA(0, 0, 1)	**Log Likelihood**	-263.658
Date:	Sat, 29 May 2021	**AIC**	533.317
Time:	23:42:07	**BIC**	544.891
Sample:	03-31-1991	**HQIC**	537.924
	- 04-30-2020		
Covariance Type:	opg		

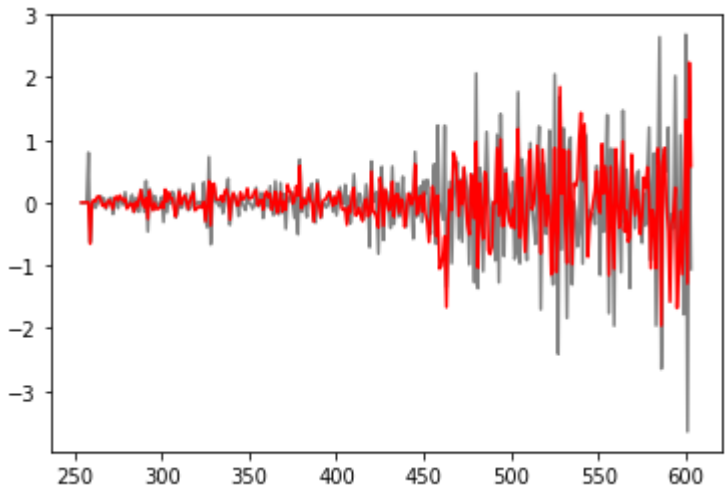
	coef	std err	z	P>|z|	[0.025	0.975]
const	0.0016	0.002	0.992	0.321	-0.002	0.005
ma.L1	-0.9452	0.012	-79.635	0.000	-0.969	-0.922
sigma2	0.2625	0.012	21.918	0.000	0.239	0.286

Ljung-Box (L1) (Q):	0.13	**Jarque-Bera (JB):**	193.44
Prob(Q):	0.72	**Prob(JB):**	0.00
Heteroskedasticity (H):	28.42	**Skew:**	-0.01
Prob(H) (two-sided):	0.00	**Kurtosis:**	6.64

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

In [203]:
```
# Residual Plot
model_fit.resid.plot()
plt.show()
#Ploting Fitted and Actual values
plt.plot(dfnew.index[:-12],dfnew['CPI'].iloc[:-12],color="gray")
plt.plot(dfnew.index[:-12],model_fit.fittedvalues,color="red")
plt.show()
```





In [204]:
```
rms = mean_squared_error(dfnew['CPI'].iloc[-12:], predic, squared=False)
rms1 = mean_squared_error(dfnew['CPI'].iloc[:-12], model_fit.fittedvalues, squared=False)
print("RMSE of Fitted: ",rms1)
print("RMSE of out of Sample forecast: ",rms)
```

```
RMSE of Fitted:  0.5125346531996787
RMSE of out of Sample forecast:  1.6270121741329326
```

ARMA

In [212]:
```
data = dfnew['CPI'].dropna().iloc[:-12]
# MA model
model = ARIMA(data, order=(2,0,1))
model_fit = model.fit()
predic = model_fit.forecast(12)
model_fit.summary()
```

Out[212]:

SARIMAX Results

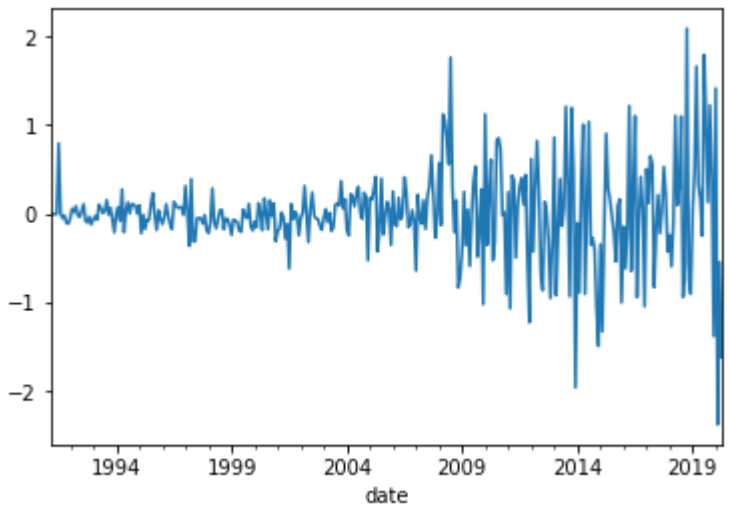
Dep. Variable:	CPI	**No. Observations:**	350
Model:	ARIMA(2, 0, 1)	**Log Likelihood**	-263.558
Date:	Sun, 30 May 2021	**AIC**	537.115
Time:	00:14:51	**BIC**	556.405
Sample:	03-31-1991	**HQIC**	544.793
	- 04-30-2020		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
const	0.0016	0.002	1.006	0.314	-0.002	0.005
ar.L1	0.0263	0.042	0.625	0.532	-0.056	0.109
ar.L2	0.0005	0.033	0.015	0.988	-0.064	0.065
ma.L1	-0.9489	0.013	-73.814	0.000	-0.974	-0.924
sigma2	0.2623	0.012	21.095	0.000	0.238	0.287

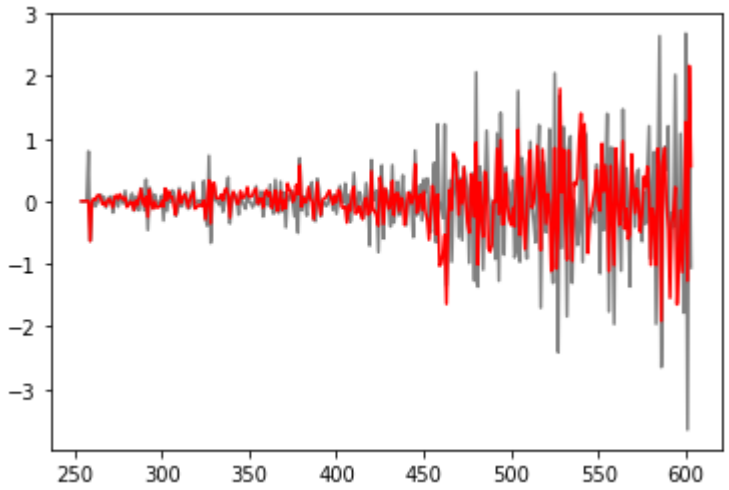
Ljung-Box (L1) (Q):	0.00	**Jarque-Bera (JB):**	200.46
Prob(Q):	0.96	**Prob(JB):**	0.00
Heteroskedasticity (H):	28.36	**Skew:**	-0.01
Prob(H) (two-sided):	0.00	**Kurtosis:**	6.71

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

In [213]:
```
# Residual Plot
model_fit.resid.plot()
plt.show()
```



In [215]:
```
#Ploting Fitted and Actual values
plt.plot(dfnew.index[:-12],data,color="gray")
plt.plot(dfnew.index[:-12],model_fit.fittedvalues,color="red")
plt.show()
```



In [216]:
```
rms = mean_squared_error(np.log2(df['CPI'].dropna().iloc[-12:]), predic, squared=Fals
e)
rms1 = mean_squared_error(data, model_fit.fittedvalues, squared=False)
print("RMSE of Fitted: ",rms1)
print("RMSE of out of Sample forecast: ",rms)
```

```
RMSE of Fitted:  0.5123773492289644
RMSE of out of Sample forecast:  6.994592339610631
```

ARIMA

In [182]:
```
data = np.log2(df['CPI'].dropna().iloc[:-12])
# ARIMA model
model = ARIMA(data, order=(2,1,1))
model_fit = model.fit()
predic = model_fit.forecast(12)
model_fit.summary()
```

Out[182]:

SARIMAX Results

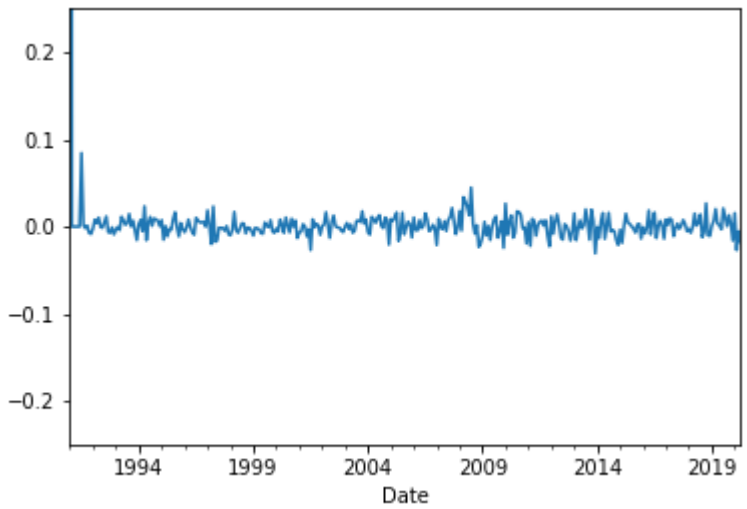
Dep. Variable:	CPI	**No. Observations:**	352
Model:	ARIMA(2, 1, 1)	**Log Likelihood**	1075.438
Date:	Sat, 29 May 2021	**AIC**	-2142.877
Time:	23:09:39	**BIC**	-2127.434
Sample:	01-01-1991	**HQIC**	-2136.731
	- 04-01-2020		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	1.0514	0.063	16.636	0.000	0.928	1.175
ar.L2	-0.0564	0.062	-0.909	0.363	-0.178	0.065
ma.L1	-0.9330	0.021	-43.446	0.000	-0.975	-0.891
sigma2	0.0001	5.3e-06	24.138	0.000	0.000	0.000

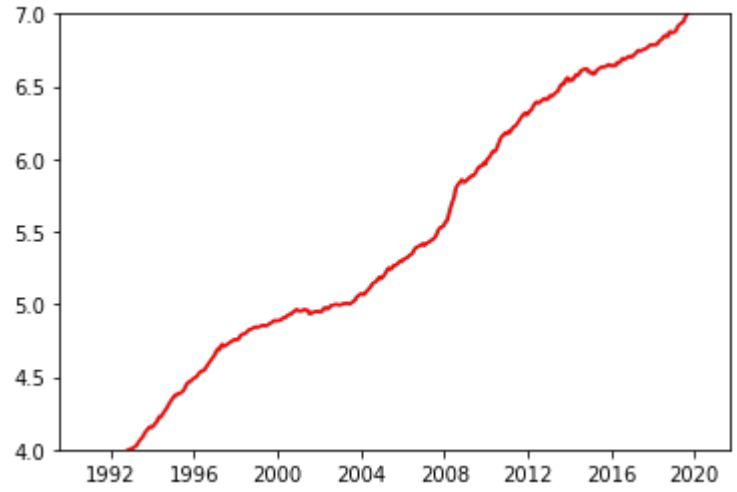
Ljung-Box (L1) (Q):	0.03	**Jarque-Bera (JB):**	890.07
Prob(Q):	0.86	**Prob(JB):**	0.00
Heteroskedasticity (H):	1.05	**Skew:**	1.14
Prob(H) (two-sided):	0.81	**Kurtosis:**	10.46

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

In [183]:
```
# Residual Plot
model_fit.resid.plot()
plt.ylim([-0.25,0.25])
plt.show()
```



In [191]:
```
#Ploting Fitted and Actual values
plt.plot(df.dropna().index[:-12],data,color="gray")
plt.plot(df.dropna().index[:-12],model_fit.fittedvalues,color="red")
plt.ylim([4,7])
plt.show()
```



In [194]:
```
rms = mean_squared_error(np.log2(df['CPI'].dropna().iloc[-12:]), predic, squared=Fals
e)
rms1 = mean_squared_error(data, model_fit.fittedvalues, squared=False)
print("RMSE of Fitted: ",rms1)
print("RMSE of out of Sample forecast: ",rms)
```

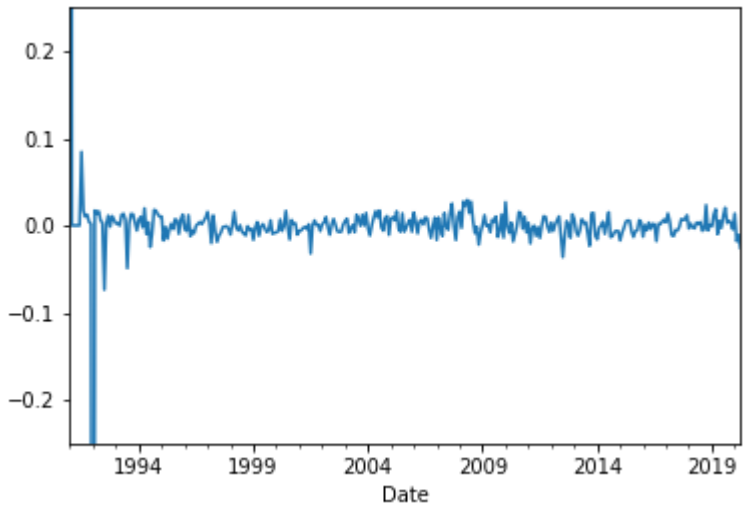
```
RMSE of Fitted:  0.1989522681713915
RMSE of out of Sample forecast:  0.04444677326928127
```

SARIMA

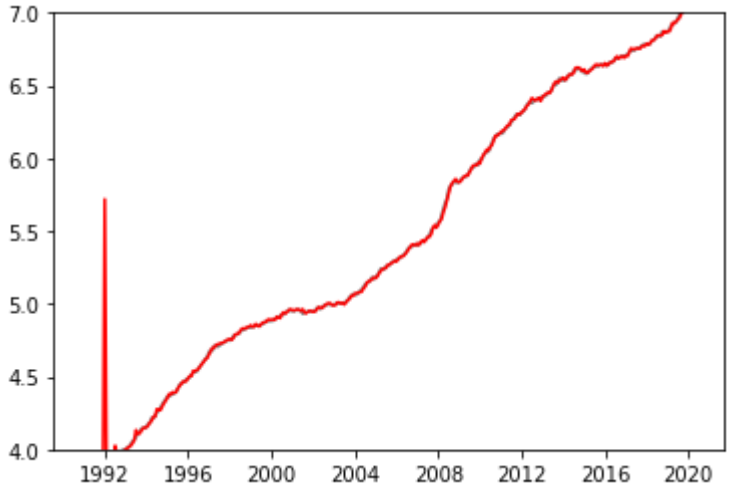
In [208]:
```
from statsmodels.tsa.statespace.sarimax import SARIMAX
data = np.log2(df['CPI'].dropna().iloc[:-12])

model=SARIMAX(endog=data,order=(2,1,1),seasonal_order=(2,1,1,12),trend='c',enforce_in
vertibility=False)
model_fit = model.fit()
predic = model_fit.forecast(12)
model_fit.summary()
print(model_fit.summary())
```

```
                                     SARIMAX Results
==========================================================================================
======
Dep. Variable:                                   CPI   No. Observations:
352
Model:             SARIMAX(2, 1, 1)x(2, 1, 1, 12)   Log Likelihood                10
52.833
Date:                            Sun, 30 May 2021   AIC                          -20
89.666
Time:                                    00:13:46   BIC                          -20
59.058
Sample:                                01-01-1991   HQIC                         -20
77.469
                                     - 04-01-2020
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept     -0.0001      0.000     -0.449      0.654      -0.001       0.000
ar.L1          0.4551      0.147      3.094      0.002       0.167       0.743
ar.L2          0.2382      0.063      3.783      0.000       0.115       0.362
ma.L1         -2.1724      0.701     -3.099      0.002      -3.547      -0.798
ar.S.L12      -0.6553      0.101     -6.484      0.000      -0.853      -0.457
ar.S.L24      -0.2765      0.079     -3.511      0.000      -0.431      -0.122
ma.S.L12      -0.3616      0.115     -3.149      0.002      -0.587      -0.137
sigma2       2.39e-05   1.56e-05      1.532      0.126   -6.69e-06    5.45e-05
==============================================================================
Ljung-Box (L1) (Q):                   6.12   Jarque-Bera (JB):                67.06
Prob(Q):                              0.01   Prob(JB):                         0.00
Heteroskedasticity (H):               0.87   Skew:                            -0.48
Prob(H) (two-sided):                  0.47   Kurtosis:                         4.96
==========================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-ste
p).
```

In [209]:
```
# Residual Plot
model_fit.resid.plot()
plt.ylim([-0.25,0.25])
plt.show()
```



In [210]:
```
#Ploting Fitted and Actual values
plt.plot(df.dropna().index[:-12],data,color="gray")
plt.plot(df.dropna().index[:-12],model_fit.fittedvalues,color="red")
plt.ylim([4,7])
plt.show()
```



In [211]:
```
rms = mean_squared_error(np.log2(df['CPI'].dropna().iloc[-12:]), predic, squared=Fals
e)
rms1 = mean_squared_error(data, model_fit.fittedvalues, squared=False)
print("RMSE of Fitted: ",rms1)
print("RMSE of out of Sample forecast: ",rms)
```

```
RMSE of Fitted:  0.22223971686606617
RMSE of out of Sample forecast:  0.07208052070229454
```

Multivariate LSTM Forecast Model

In [239]:

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
```

ERROR:root:Internal Python error in the inspect module.
Below is the traceback from this internal error.

```
Traceback (most recent call last):
  File "C:\ProgramData\Anaconda3\lib\site-packages\IPython\core\interactiveshell.p
y", line 3267, in run_code
    exec(code_obj, self.user_global_ns, self.user_ns)
  File "<ipython-input-239-7b363eb856ad>", line 4, in <module>
    from keras.models import Sequential
  File "C:\Users\Sarmad-PC\AppData\Roaming\Python\Python37\site-packages\keras\__ini
t__.py", line 21, in <module>
    from tensorflow.python import tf2
  File "<frozen importlib._bootstrap>", line 983, in _find_and_load
  File "<frozen importlib._bootstrap>", line 963, in _find_and_load_unlocked
  File "<frozen importlib._bootstrap>", line 906, in _find_spec
  File "<frozen importlib._bootstrap_external>", line 1280, in find_spec
  File "<frozen importlib._bootstrap_external>", line 1246, in _get_spec
  File "<frozen importlib._bootstrap_external>", line 1115, in __iter__
  File "<frozen importlib._bootstrap_external>", line 1103, in _recalculate
  File "<frozen importlib._bootstrap_external>", line 1099, in _get_parent_path
KeyError: 'tensorflow_core'

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "C:\ProgramData\Anaconda3\lib\site-packages\IPython\core\interactiveshell.p
y", line 2018, in showtraceback
    stb = value._render_traceback_()
AttributeError: 'KeyError' object has no attribute '_render_traceback_'

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "C:\Users\Sarmad-PC\AppData\Roaming\Python\Python37\site-packages\tensorflow_
core\python\pywrap_tensorflow.py", line 58, in <module>
    from tensorflow.python.pywrap_tensorflow_internal import *
  File "C:\Users\Sarmad-PC\AppData\Roaming\Python\Python37\site-packages\tensorflow_
core\python\pywrap_tensorflow_internal.py", line 28, in <module>
    _pywrap_tensorflow_internal = swig_import_helper()
  File "C:\Users\Sarmad-PC\AppData\Roaming\Python\Python37\site-packages\tensorflow_
core\python\pywrap_tensorflow_internal.py", line 24, in swig_import_helper
    _mod = imp.load_module('_pywrap_tensorflow_internal', fp, pathname, description)
  File "C:\ProgramData\Anaconda3\lib\imp.py", line 242, in load_module
    return load_dynamic(name, filename, file)
  File "C:\ProgramData\Anaconda3\lib\imp.py", line 342, in load_dynamic
    return _load(spec)
ImportError: DLL load failed: %1 is not a valid Win32 application.

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "C:\ProgramData\Anaconda3\lib\site-packages\IPython\core\ultratb.py", line 10
95, in get_records
    return _fixed_getinnerframes(etb, number_of_lines_of_context, tb_offset)
  File "C:\ProgramData\Anaconda3\lib\site-packages\IPython\core\ultratb.py", line 31
3, in wrapped
    return f(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\IPython\core\ultratb.py", line 34
7, in _fixed_getinnerframes
    records = fix_frame_records_filenames(inspect.getinnerframes(etb, context))
  File "C:\ProgramData\Anaconda3\lib\inspect.py", line 1500, in getinnerframes
    frameinfo = (tb.tb_frame,) + getframeinfo(tb, context)
  File "C:\ProgramData\Anaconda3\lib\inspect.py", line 1458, in getframeinfo
    filename = getsourcefile(frame) or getfile(frame)
  File "C:\ProgramData\Anaconda3\lib\inspect.py", line 696, in getsourcefile
    if getattr(getmodule(object, filename), '__loader__', None) is not None:
  File "C:\ProgramData\Anaconda3\lib\inspect.py", line 733, in getmodule
    if ismodule(module) and hasattr(module, '__file__'):
```

```
  File "C:\Users\Sarmad-PC\AppData\Roaming\Python\Python37\site-packages\tensorflow
\__init__.py", line 50, in __getattr__
  File "C:\Users\Sarmad-PC\AppData\Roaming\Python\Python37\site-packages\tensorflow
\__init__.py", line 44, in _load
  File "C:\ProgramData\Anaconda3\lib\importlib\__init__.py", line 127, in import_mod
ule
    return _bootstrap._gcd_import(name[level:], package, level)
  File "<frozen importlib._bootstrap>", line 1006, in _gcd_import
  File "<frozen importlib._bootstrap>", line 983, in _find_and_load
  File "<frozen importlib._bootstrap>", line 953, in _find_and_load_unlocked
  File "<frozen importlib._bootstrap>", line 219, in _call_with_frames_removed
  File "<frozen importlib._bootstrap>", line 1006, in _gcd_import
  File "<frozen importlib._bootstrap>", line 983, in _find_and_load
  File "<frozen importlib._bootstrap>", line 967, in _find_and_load_unlocked
  File "<frozen importlib._bootstrap>", line 677, in _load_unlocked
  File "<frozen importlib._bootstrap_external>", line 728, in exec_module
  File "<frozen importlib._bootstrap>", line 219, in _call_with_frames_removed
  File "C:\Users\Sarmad-PC\AppData\Roaming\Python\Python37\site-packages\tensorflow_
core\__init__.py", line 28, in <module>
    from tensorflow.python import pywrap_tensorflow  # pylint: disable=unused-import
  File "C:\Users\Sarmad-PC\AppData\Roaming\Python\Python37\site-packages\tensorflow_
core\python\pywrap_tensorflow.py", line 74, in <module>
    raise ImportError(msg)
ImportError: Traceback (most recent call last):
  File "C:\ProgramData\Anaconda3\lib\site-packages\IPython\core\interactiveshell.p
y", line 3267, in run_code
    exec(code_obj, self.user_global_ns, self.user_ns)
  File "<ipython-input-239-7b363eb856ad>", line 4, in <module>
    from keras.models import Sequential
  File "C:\Users\Sarmad-PC\AppData\Roaming\Python\Python37\site-packages\keras\__ini
t__.py", line 21, in <module>
    from tensorflow.python import tf2
  File "<frozen importlib._bootstrap>", line 983, in _find_and_load
  File "<frozen importlib._bootstrap>", line 963, in _find_and_load_unlocked
  File "<frozen importlib._bootstrap>", line 906, in _find_spec
  File "<frozen importlib._bootstrap_external>", line 1280, in find_spec
  File "<frozen importlib._bootstrap_external>", line 1246, in _get_spec
  File "<frozen importlib._bootstrap_external>", line 1115, in __iter__
  File "<frozen importlib._bootstrap_external>", line 1103, in _recalculate
  File "<frozen importlib._bootstrap_external>", line 1099, in _get_parent_path
KeyError: 'tensorflow_core'

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "C:\ProgramData\Anaconda3\lib\site-packages\IPython\core\interactiveshell.p
y", line 2018, in showtraceback
    stb = value._render_traceback_()
AttributeError: 'KeyError' object has no attribute '_render_traceback_'

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "C:\Users\Sarmad-PC\AppData\Roaming\Python\Python37\site-packages\tensorflow_
core\python\pywrap_tensorflow.py", line 58, in <module>
    from tensorflow.python.pywrap_tensorflow_internal import *
  File "C:\Users\Sarmad-PC\AppData\Roaming\Python\Python37\site-packages\tensorflow_
core\python\pywrap_tensorflow_internal.py", line 28, in <module>
    _pywrap_tensorflow_internal = swig_import_helper()
  File "C:\Users\Sarmad-PC\AppData\Roaming\Python\Python37\site-packages\tensorflow_
core\python\pywrap_tensorflow_internal.py", line 24, in swig_import_helper
    _mod = imp.load_module('_pywrap_tensorflow_internal', fp, pathname, description)
  File "C:\ProgramData\Anaconda3\lib\imp.py", line 242, in load_module
    return load_dynamic(name, filename, file)
  File "C:\ProgramData\Anaconda3\lib\imp.py", line 342, in load_dynamic
    return _load(spec)
```

```
ImportError: DLL load failed: %1 is not a valid Win32 application.


Failed to load the native TensorFlow runtime.

See https://www.tensorflow.org/install/errors

for some common reasons and solutions.  Include the entire stack trace
above this error message when asking for help.
```

In another workbook as require tensor and its install on another Python environment (Python 3.6) so will run LSTM on that note book

In [ ]:

In [4]:

```
# Importing basic functions
import pandas as pd
#import missingno as mano
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import math as mt
import statistics as st
from numpy.random import seed
from numpy.random import randn
from scipy.stats import shapiro# Importing basic functions
import pandas as pd
#import missingno as mano
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import math as mt
import statistics as st
from numpy.random import seed
from numpy.random import randn
from scipy.stats import shapiro
import matplotlib
from matplotlib.pyplot import figure
# Importinf Clasification Regression Related Functions
from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import log_loss
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.utils import resample
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn import svm
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.model_selection import StratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn import metrics
import xgboost as xgb
# This function read data from excel and csv files to dataframe.
# Input Arguments file: File name with path (Eg: 'D:/data/dataread.csv'), f_type: Fil
e Type (Eg: 'csv',default csv)
def file_todataframe(file,f_type):
    if f_type == 'csv':
        return pd.read_csv(file)
```

```
    elif f_type == 'excel':
        return pd.read_excel(file)
    elif f_type == 'json':
        return pd.read_json(file)

# This function display shape, data type, data near head and tail of given data fram.
# Input Arguments df: dataframe, n: No f data points to display
def df_details(d_f,n):
    print('Data Types of Column: \n',d_f.dtypes)
    print('\n Size of Datarame: ',d_f.shape)
    print('\n Top and bottom ',n,' rows: \n')
    display(d_f.head(n).append(d_f.tail(n)))

# This function drops irrelevant columns
# Input Arguments df: dataframe, col_del: Value or index array of column to delete (E
g: [1,3,5] or ['Names',Sales]),
# typ: 1 for column index and 0 for column name in col_del
def col_drop(d_f,col_del,typ=0):
    if typ == 0:
        d_f = d_f.drop(col_del,axis=1)
    elif typ == 1:
        d_f = d_f.drop(df.columns[col_del],axis=1)
    return d_f

# This function drops rows with particular column values
# Input Arguments df: dataframe, row_del: Delete if row with given value,
# col_ref: Name of column to check for row values (Eg:['Names']),
def row_drop(d_f,row_del,col_ref):
    d_f = d_f.drop(d_f[d_f[col_ref] == row_del].index)
    return d_f

# This function give deatials for missing values in data
# Input Arguments df: dataframe
def miss_ch(d_f):
    print('Available data with no nulls: ', d_f.dropna().shape[0])
    display('Deatils of Null values column wise',d_f.isnull().sum())

# This function give deatials for missing values in data
# Input Arguments df: dataframe, col_int: Columns of intrest (Eg: ['Sale','Custome
r'],['all'] default 'all'),
# graph: Types of graph to display. (Eg: ['bar','matrix','heatmap','dendrogram'],['al
l'] default 'all')
def miss_viz(d_f,col_int = 'all',graph = 'all'):
    import missingno as mano
    av_gp = ['bar','matrix','heatmap','dendrogram']
    col_nam = d_f.columns
    if col_int == 'all':
        col_int = col_nam
    elif not all(i in col_nam for i in col_int):
        print("Invalid column name")
        return

    if graph == 'all':
        graph = av_gp
    elif not all(i in av_gp for i in graph):
        print("Invalid Graph type, select 'bar','matrix','heatmap','dendrogram'")
        return

    for gp in graph:
        getattr(mano, gp)(d_f[col_int])

# This function fill missing values
# Input Arguments df: dataframe, col_int: Columns of intrest (Eg: ['Sale','Custome
r'],['all'] default 'all'),
# metd: Types of graph to display. (Option: {0(float),'backfill', 'bfill', 'pad', 'ff
ill', 'linear'} default None)
```

```
def fill_miss(d_f,col_int = 'all',metd = None):
    col_nam = d_f.columns
    av_method = ['backfill', 'bfill', 'pad', 'ffill', 'linear']
    if col_int == 'all':
        col_int = col_nam
    elif not all(i in col_nam for i in col_int):
        print("Invalid column name")
        return d_f

    if metd == None:
        return d_f
    elif (type(metd) == int) | (type(metd) == float):
        print('yes')
        return d_f[col_int].fillna(metd)
    elif metd == 'linear':
        return d_f[col_int].interpolate(method = 'linear')
    elif  metd in av_method:
        return d_f[col_int].fillna(method = 'ffill')
    else:
        print("Invalid fill type")
        return d_f

# Function for Numerical Data Analysis
# Input Arguments df: dataframe, col_int: Columns of intrest (Eg: ['Sale','Custome
r'],['all'] default 'all'),
# func: Types of graph and function. (Option: {'distplot','boxpot','scatterplot','des
cribe','normality'} default all)
def data_num(d_f,col_int = 'all',func = 'all',scat = None):
    from scipy.stats import shapiro
    from statsmodels.graphics.gofplots import qqplot
    av_func = ['hist','boxplot','scatter','describe','normality']
    numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
    col_nam = d_f.select_dtypes(include=numerics).columns
    #Checking Parameter Column names
    if col_int == 'all':
        col_int = col_nam
    elif not all(i in col_nam for i in col_int):
        print("Invalid column name")
        return

    #Checking Parameter available function
    if func == 'all':
        func = av_func
    elif not all(i in av_gp for i in graph):
        print("Invalid Graph type, select 'distplot','boxpot','scatterplot','describ
e','normality'")
        return
    if scat is None:
        scat = d_f.columns[0]


    for fn in func:
        if fn == 'describe':
            display("Statictical Details",d_f[col_int].describe())
        else:
            for col in col_int:
                if fn == 'normality':
                    #qqplot(df[col])
                    print("Normality Test for: ",col)
                    stat, p = shapiro(d_f[col])
                    if p > 0.05:
                        print('Sample looks Gaussian. Statistics=%.3f, p=%.3f'% (stat
, p))
                    else:
                        print('Sample does not look Gaussian. Statistics=%.3f, p=%.3f
'% (stat, p))
```

```
                elif fn == 'scatter':
                    display(col)
                    plt.scatter(d_f[col],d_f[scat])
                    plt.show()
                else:
                    display(col)
                    getattr(plt, fn)(d_f[col])
                    plt.show()

# Function for Categorical Data Analysis
# Input Arguments df: dataframe, col_int: Columns of intrest (Eg: ['Sale','Custome
r'],['all'] default 'all')
def data_cat(d_f,col_int = 'all',bar = None):
    import matplotlib.pyplot as plt
    col_nam = d_f.select_dtypes(include=['object','category']).columns
    numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
    col_num = d_f.select_dtypes(include=numerics).columns
    #Checking Parameter Column names
    if col_int == 'all':
        col_int = col_nam
    elif not all(i in col_nam for i in col_int):
        print("Invalid column name")
        return

    if not ((bar in col_num) | (bar is None)):
        print("Only numeric column for Bar")
        return
    if not (bar is None):
        for col in col_int:
            plt.figure(figsize=(8,5))
            plt.bar(d_f[col], d_f[bar])
            plt.show()
        return
    for col in col_int:
        plt.suptitle(col)
        d_f[col].value_counts().plot(kind='bar')
        plt.show()

# Function for Canging column type
# Input Arguments df: dataframe, col_int: Columns of intrest (Eg: ['Sale','Custome
r'],['all'] default 'all'),
# dtyp: New data types of coloumn default int
def col_dtype(d_f,col_int,dtyp = int):
    d_f = deep_copy(d_f)
    av_fun = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64','int','floa
t','str','category']
    col_nam = d_f.columns
    #Checking Parameter Column names
    if not all(i in col_nam for i in col_int):
        print("Invalid column name")
        return d_f
    #Checking Parameter available function
    if not dtyp in av_fun:
        print("Invalid data type")
        return d_f

    d_f[col_int] = d_f[col_int].astype(dtyp,errors='ignore')
    return d_f

# Function for Canging column type
# Input Arguments df: dataframe, col_int: Columns of intrest (Eg: ['Sale','Custome
r'],['all'] default 'all'),
# opr: Operation to be performed on coloumn values, val = [old value, new value]
def col_opre(d_f,col_int,opr = None,val = None):
    col_nam = d_f.select_dtypes(include=['object']).columns
    av_opr = ['str_replace','rm_space','chg_value']
```

```
    #Checking Parameter Column names
    if not (col_int in col_nam):
        print("Invalid column name")
        return d_f

    if opr == 'str_replace':
        d_f[col_int] = d_f[col_int].str.replace(val[0], val[1])
        return d_f
    elif opr == 'rm_space':
        d_f[col_int] = d_f[col_int].str.replace(' ', '')
        return d_f
    elif opr == 'chg_value':
        d_f[col_int] = d_f[col_int].replace(val[0], val[1])
        return d_f
    else:
        print("Invalid Opeation")
        return d_f

# Function for Performing numerical operations to column values
# Input Arguments df: dataframe, col_int: Columns of intrest (Eg: ['Sale','Custome
r'],['all'] default 'all'),
# opr: Operation to be performed on coloumn values, val: value to apply on column
def col_opre(d_f,col_int = 'all',opr = None,val = None):

    numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64',]
    col_num = d_f.select_dtypes(include=numerics).columns


    if opr == 'add':
        d_f[col_int] = d_f[col_int]+val
        return d_f
    elif opr == 'sub':
        d_f[col_int] = d_f[col_int]-val
        return d_f
    elif opr == 'mul':
        d_f[col_int] = d_f[col_int]*val
        return d_f
    elif opr == 'div':
        d_f[col_int] = d_f[col_int]/val
        return d_f
    else:
        print("Invalid Opeation")
        return d_f

# Function for creating a deep copy
# Input Arguments df: dataframe, Output data frame copy
def deep_copy(d_f):
    return d_f.copy(deep=True)

# Function for coding Categorical variable  to Numeric
# Input Arguments df: dataframe, col_int: Columns of intrest (Eg: ['Sale','Custome
r'],['all'] default 'all'{must be string}),
# coding_type: Coding type to apply{'label','binary','ordinal','onehot'} default,labe
l, contain: For binary
def cat_num(d_f,col_int,coding_type = 'label',contain=None,X = None):
    from sklearn.preprocessing import OrdinalEncoder
    from sklearn.preprocessing import OneHotEncoder
    d_f = deep_copy(d_f)
    col_nam = d_f.select_dtypes(include=['object']).columns
    if not all(i in col_nam for i in col_int):
        print("Invalid column name")
        return d_f
    else:
        d_f[col_int] =  d_f[col_int].astype('str')
```

```
    if coding_type == 'label':
        for col in col_int:
            d_f[col] = d_f[col].astype('category')
            d_f[col] = d_f[col].cat.codes
        return d_f
    elif coding_type == 'binary':
        for col in col_int:
            d_f[col] = np.where(d_f[col].str.contains(contain), 1, 0)
        return d_f
    elif coding_type == 'ordinal':
        for col in col_int:
            ord_enc = OrdinalEncoder()
            d_f[col] = ord_enc.fit_transform(d_f[[col]])
        return d_f
    elif coding_type == 'onehot':
        for col in col_int:
            temp = pd.get_dummies(d_f[col], prefix=col)
            d_f = d_f.join(temp)
            d_f = col_drop(d_f,col)
        return d_f
    else:
        display("Invalid Ending Method")
        return d_f

# This function scatter Plot
# Input Arguments df: dataframe, col_int: Two columns of intrest (Eg: ['Sale','Custom
er'],['all']),
def scatter_plot(d_f,col_int = None):
    numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
    col_nam = d_f.select_dtypes(include=numerics).columns
    if (col_int is None) & (not all(i in col_nam for i in col_int)) & (len(col_int) !
= 2):
        print("Invalid column name")
        return
    x = d_f[col_int[0]]
    y = d_f[col_int[1]]
    xv = col_int[0]+' X- Value'
    yv = col_int[1]+' Y- Value'
    tit = col_int[0] + ' vs ' + col_int[1] +' Scatter plot'

    plt.scatter(x, y)
    plt.rcParams.update({'figure.figsize':(10,8), 'figure.dpi':100})
    plt.title(tit)
    plt.xlabel(xv)
    plt.ylabel(yv)
    plt.show()

# Function for applyng Machine learning Models
# Input Arguments df with last raw as labels, task: regression or classification, alg
o: Descion Tree or Random Forest
def anova(d_f,col_int='all',col_main=None):
    from scipy import stats
    import statsmodels.api as sm
    from statsmodels.formula.api import ols

    numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64',]
    col_num = d_f.select_dtypes(include=numerics).columns

    #Checking Parameter Column names
    if col_int == 'all':
        col_int = col_nam
    elif not all(i in col_nam for i in col_int):
        print("Invalid column name")
        return

    for col in col_int:
```

```
        model = ols(col_main+'~ C(Q("'+col+'"))', data=d_f).fit()
        anova_table = sm.stats.anova_lm(model, typ=2)
        print ("\nAnova =>",col_main," - ",col)
        display(anova_table)

## T Test
def t_test(d_f,col_ind=[1,2]):
    data1 = d_f.iloc[:,col_ind[0]].values
    data2 = d_f.iloc[:,col_ind[1]].values
    # calculate means
    mean1, mean2 = st.mean(data1), st.mean(data2)
    # calculate sample standard deviations
    std1, std2 = st.stdev(data1), st.stdev(data2)
    # calculate standard errors
    n1, n2 = len(data1), len(data2)
    se1, se2 = std1/mt.sqrt(n1), std2/mt.sqrt(n2)
    # standard error on the difference between the samples
    sed = mt.sqrt(se1**2.0 + se2**2.0)
    # calculate the t statistic
    t_stat = (mean1 - mean2) / sed
    print('T Test Statistics=%.3f' % (t_stat))

## Normality Test
def norm(d_f,col_ind=[1]):
    for col in col_ind:
        data = d_f.iloc[:,col]
        stat, p = shapiro(data)
        print('Statistics=%.3f, p=%.3f' % (stat, p))
        # interpret
        alpha = 0.05
        if p > alpha:
            print('Sample looks Gaussian (fail to reject H0)')
        else:
            print('Sample does not look Gaussian (reject H0)')
        data.hist()

# This function Correlation Heat Map
# Input Arguments d_f: dataframe
def corr_hmap(d_f):
    import seaborn as sns
    sns.set(rc={'figure.figsize':(15,8)})
    corr = d_f.corr().dropna(1,'all').dropna(0,'all')
    ax = sns.heatmap(
        corr,
        vmin=-1, vmax=1, center=0,
        cmap=sns.diverging_palette(20, 220, n=200),
        square=True
    )
    ax.set_xticklabels(
        ax.get_xticklabels(),
        rotation=90,
        horizontalalignment='right'
    );
    ax.set_yticklabels(
        ax.get_yticklabels(),
        rotation=0,
        horizontalalignment='right'
    );

# Function for applyng Machine learning Models
# Input Arguments df with last raw as labels, task: regression or classification, alg
o: Descion Tree or Random Forest
def ml_algo(X_train, X_test, y_train, y_test,algo = 'decisiontree',task = 'Reg',n=3):

    #X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.33, random
_state=1)
```

```
    if task == 'Reg':
        if algo == 'decisiontree':
            model = DecisionTreeRegressor(max_depth=n)

        elif algo == 'randomforest':
            model = RandomForestRegressor(n_estimators = n, random_state = 42)

        elif algo == 'knn':
            model = KNeighborsRegressor(n_neighbors=n)

        elif algo == 'lreg':
            model = LinearRegression()

        elif algo == 'svm':
            model = SVR(kernel="rbf")

        elif algo == 'gboost':
            model = GradientBoostingRegressor()

        elif algo == 'adaboost':
            model = AdaBoostRegressor(random_state=0, n_estimators=100)

        else:
            print("Invalid Algorithm")
            return

        model.fit(X_train,y_train)
        y_pred = model.predict(X_test)
        mae = mean_absolute_error(y_test, y_pred)
        mse = mean_squared_error(y_test, y_pred)
        r_2 = r2_score(y_test, y_pred)
        r_2adj = r2_score(y_test, y_pred,multioutput='variance_weighted')
        return mae,mse,r_2,r_2adj,model

    elif task == 'Class':
        if algo == 'decisiontree':
            model = DecisionTreeClassifier(max_depth=n)

        elif algo == 'randomforest':
            model=RandomForestClassifier(n_estimators=n)

        elif algo == 'knn':
            model = KNeighborsClassifier(n_neighbors=n)

        elif algo == 'gboost':
            model = GradientBoostingClassifier()

        elif algo == 'adaboost':
            model = AdaBoostClassifier(n_estimators=100, random_state=0)

        elif algo == 'svc':
            model = svm.SVC()

        elif algo == 'naive':
            model = GaussianNB()

        elif algo == 'xgboost':
            model = xgb.XGBClassifier(random_state=1,learning_rate=0.01)

        elif algo == 'mlp':
            model = MLPClassifier(hidden_layer_sizes=5)

        elif algo == 'logit':
            model = LogisticRegression()
        else:
            print("Invalid Algorithm")
```

```
            return


        model.fit(X_train,y_train)
        y_pred = model.predict(X_test)
        c_m =confusion_matrix(y_test, y_pred)
        c_r = classification_report(y_test, y_pred)
        acc_sc = accuracy_score(y_test, y_pred)

        fpr , tpr, _ = roc_curve(y_test, y_pred)
        auc_roc = metrics.auc(fpr, tpr)

        logloss = log_loss(y_test, y_pred)
        #Precision
        pre_l = precision_score(y_test, y_pred)*100
        #Recall
        recall_l = recall_score(y_test, y_pred)*100
        return c_m,c_r,acc_sc,auc_roc,logloss,model,pre_l,recall_l

# Data frame with output variable at first location
# f_s features to select
def feature_selectk(d_f,f_s,col):
    from sklearn.feature_selection import SelectKBest
    from sklearn.feature_selection import f_classif
    from sklearn.model_selection import train_test_split

    x = col_drop(d_f,col,typ=0)
    y = d_f[col]
    fs = SelectKBest(score_func=f_classif, k=f_s)
    # learn relationship from training data
    fs.fit(x, y)
    # transform train input data
    x_fs = fs.transform(x)
    return x_fs,y,fs

def feature_select_rffs(x,y,reg=True):
    from sklearn.feature_selection import SelectFromModel
    if reg:
        sel = SelectFromModel(RandomForestRegressor(n_estimators = 50))
    else:
        sel = SelectFromModel(RandomForestClassifier(n_estimators = 50))
    sel.fit(x, y)
    selected_feat= x.columns[(sel.get_support())]
    return selected_feat

def importantfeat_rffs(d_f,reg=True):
    # random forest for feature importance on a regression problem
    from sklearn.datasets import make_regression
    from sklearn.ensemble import RandomForestRegressor
    from matplotlib import pyplot
    # define dataset
    X = d_f.iloc[:,1:len(d_f.columns)]
    y = d_f.iloc[:,:1]
    # define the model
    model = RandomForestRegressor()
    # fit the model
    model.fit(X, y)
    # get importance
    importance = model.feature_importances_
    # summarize feature importance
    for i,v in enumerate(importance):
        print('Feature: %0d, Score: %.5f' % (i,v))
    # plot feature importance
    pyplot.bar([x for x in range(len(importance))], importance)
    pyplot.xticks(range(1,len(importance)),df.iloc[:,1:len(df)-1].columns,rotation=90
)
```

```
    pyplot.show()

# kfoldtype = KFold , StratifiedKFold
def ml_algo_cv(X,y,algo = 'decisiontree',task = 'Reg',n=3,split=10,kftype ='skfold'):
    accf = 0
    aucf = 0
    pref = 0
    recallf = 0
    i= 0
    if kftype == 'skfold':
        kf = StratifiedKFold(n_splits=split, random_state=None, shuffle=False)
    elif kftype == 'kfold':
        kf = KFold(n_splits=split, random_state=None, shuffle=False)

    for train_index, test_index in kf.split(X,y):
        trainX, testX = X.iloc[train_index], X.iloc[test_index]
        trainY, testY = y.iloc[train_index], y.iloc[test_index]
        cm,cr,acc,auc,ll,mdl,pre,recall = ml_algo(trainX, testX, trainY, testY,algo =
algo ,task = task,n=3)
        accf = accf + acc
        aucf = aucf + auc
        pref = pref + pre
        recallf = recallf + recall
        i = i + 1
    accf = accf/i
    aucf = auc/i
    pref = pref/i
    recallf = recallf/i
    return accf,aucf,pref,recallf,cm

## Upsampling minority class and down sampling majority class
def class_imbalance(d_f,resmpl=0.75,minor=0,col=None):
    df_majority = row_drop(df,minor+1,col[0])
    df_minority = row_drop(df,minor,col[0])
    resmplc = mt.ceil(mt.ceil(df[col].value_counts()[minor]* 0.75) * resmpl)

    # Upsample minority class
    df_minority_upsampled = resample(df_minority,
                                     replace=True,     # sample with replacement
                                     n_samples=resmplc,    # to match majority class
                                     random_state=123) # reproducible results
    df_majority_under = resample(df_majority,
                                 replace=False,     # sample with replacement
                                 n_samples=resmplc,    # to match majority class
                                 random_state=123) # reproducible results
    # Combine majority class with upsampled minority class
    df_smp = pd.concat([df_majority_under, df_minority_upsampled])
    # Display new class counts
    #print(df_smp[col].value_counts())
    return df_smp
## Upsampling minority class and down sampling majority class
def class_imbalance_no(d_f,minr,maj,minor=0,col=None):
    df_majority = row_drop(df,minor+1,col[0])
    df_minority = row_drop(df,minor,col[0])


    # Upsample minority class
    df_minority_upsampled = resample(df_minority,
                                     replace=True,     # sample with replacement
                                     n_samples=minr,    # to match majority class
                                     random_state=123) # reproducible results
    df_majority_under = resample(df_majority,
                                 replace=False,     # sample with replacement
                                 n_samples=maj,    # to match majority class
                                 random_state=123) # reproducible results
    # Combine majority class with upsampled minority class
```

```
    df_smp = pd.concat([df_majority_under, df_minority_upsampled])
    # Display new class counts
    #print(df_smp[col].value_counts())
    return df_smp

def con_mat(con_mat,lab = ['x-label','y-label']):
    import seaborn as sns
    import matplotlib.pyplot as plt
    ax= plt.subplot()
    sns.heatmap(cm, annot=True, fmt="d", ax = ax,cmap="YlGnBu"); #annot=True to annot
ate cells

    # labels, title and ticks
    ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
    ax.set_title('Confusion Matrix');
    ax.xaxis.set_ticklabels([lab[0],lab[1]]); ax.yaxis.set_ticklabels([lab[0],lab[1
]]);

# FS: Feature Selection
# CV: Cross Validiation
# CI: CLass Imbalance
def ml_com(d_f,colum,algor='decisiontree',fs=True,cv=True,ci=True,resmpl = 0.75):
    x = col_drop(d_f,colum,typ=0)
    y = d_f[colum]
    if fs & cv & ci:
        df_samp = class_imbalance(d_f,resmpl = resmpl,minor=0,col=colum)
        fet = feature_select_rffs(x,y,reg=False)
        x = col_drop(df_samp,colum,typ=0)
        x = x[fet]
        y = df_samp[colum]
        acc,auc,pre,recall,cm = ml_algo_cv(x,y,algo = algor,task = 'Class',n=3,split=
10,kftype ='skfold')

    elif fs & ~cv & ~ci:
        fet = feature_select_rffs(x,y,reg=False)
        x = col_drop(d_f,colum,typ=0)
        x = x[fet]
        y = d_f[colum]
        X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.33, ran
dom_state=1)
        cm,cr,acc,auc,ll,model,pre,recall = ml_algo(X_train, X_test, y_train, y_test,
algo = algor,task = 'Class',n=3)
    elif ~fs & ~cv & ci:
        df_samp = class_imbalance(d_f,resmpl = resmpl,minor=0,col=colum)
        x = col_drop(df_samp,colum,typ=0)
        y = df_samp[colum]
        X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.33, ran
dom_state=1)
        cm,cr,acc,auc,ll,model,pre,recall = ml_algo(X_train, X_test, y_train, y_test,
algo = algor,task = 'Class',n=3)
    elif ~fs & ~cv & ~ci:
        X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.33, ran
dom_state=1)
        cm,cr,acc,auc,ll,model,pre,recall = ml_algo(X_train, X_test, y_train, y_test,
algo = algor,task = 'Class',n=3)

    return acc,auc,pre,recall,cm
```

Project Part#1

Using Google Trends and Mobility data for Nowcasting Quaterley GDP

Wrangling through data

In [5]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [6]:
```
# Loading trends data
gt =  file_todataframe('Google Trend & Mobility Data.xlsx','excel')
```

In [7]:
```
df_details(gt,5)
```

```
Data Types of Column:
 date                datetime64[ns]
Economic crisis              int64
Crisis                       int64
Recession                    int64
Financial crisis             int64
Inflation                    int64
Unemployment                 int64
BISP                         int64
ehsaas program               int64
USAID                        int64
Credit                       int64
Loan                         int64
Interest                     int64
House Loan                   int64
Car Loan                     int64
Food                         int64
Cinema                       int64
Cars                         int64
Birthday                     int64
Travel                       int64
Weddings                     int64
Fitness                      int64
Cigarette                    int64
Tourism                      int64
Hotels                       int64
Fast Food                    int64
House for sale               int64
Construction                 int64
Investment                   int64
Jobs                         int64
Agriculture                  int64
FMCG                         int64
Aviation                     int64
Manufacturing                int64
Textile                      int64
Economy News                 int64
Business News                int64
World News                   int64
Politics                     int64
Newspapers                   int64
mehngai                      int64
Real estate                  int64
deficit                      int64
elections                    int64
parliament                   int64
taxes                        int64
government                   int64
budget                       int64
economic growth              int64
subsidy                      int64
current account              int64
trade                        int64
protest                      int64
stock market                 int64
revenue                      int64
LSM                        float64
M0                         float64
PSB                        float64
CPI                        float64
dtype: object

 Size of Datarame:  (171, 59)
```

```
Top and bottom  5  rows:
```

	date	Economic crisis	Crisis	Recession	Financial crisis	Inflation	Unemployment	BISP	ehsaas program	USAID	...
0	2007-01-01	0	25	0	0	0	2	0	0	2	...
1	2007-02-01	6	24	0	0	12	5	0	0	7	...
2	2007-03-01	0	16	0	0	10	2	0	0	6	...
3	2007-04-01	0	11	0	0	11	15	0	0	2	...
4	2007-05-01	0	14	0	0	32	9	0	0	9	...
166	2020-11-01	0	7	2	1	10	2	2	6	0	...
167	2020-12-01	1	10	2	0	9	3	3	18	0	...
168	2021-01-01	1	10	1	0	8	2	2	10	0	...
169	2021-02-01	0	9	1	0	7	2	4	9	0	...
170	2021-03-01	1	9	1	0	7	2	1	11	0	...

10 rows × 59 columns

In [8]:
```
# Loading Quaterly GDP Data
qgdp = file_todataframe('GDP_Q.xlsx','excel')
```

In [9]:
```
df_details(qgdp,5)
```

```
Data Types of Column:
 Date    datetime64[ns]
GDPQ           float64
dtype: object

 Size of Datarame:  (84, 2)

 Top and bottom  5  rows:
```

	Date	GDPQ
0	2000-03-01	1379.900000
1	2000-06-01	1418.500000
2	2000-09-01	1430.000000
3	2000-12-01	1426.200000
4	2001-03-01	1415.800000
79	2019-12-01	3189.820283
80	2020-03-01	3080.249332
81	2020-06-01	3114.324411
82	2020-09-01	3154.097321
83	2020-12-01	3177.588749

In [10]:
```
# Checking for missing values
miss_ch(qgdp)
```

```
Available data with no nulls:  84

'Deatils of Null values column wise'

Date    0
GDPQ    0
dtype: int64
```

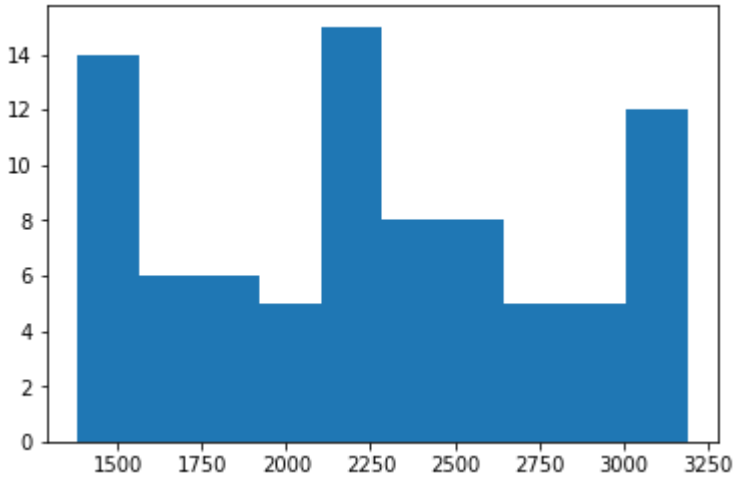
In [11]:
```
# Checking for missing values in trends data
miss_ch(gt)
```

```
Available data with no nulls:  171

'Deatils of Null values column wise'

date                0
Economic crisis     0
Crisis              0
Recession           0
Financial crisis    0
Inflation           0
Unemployment        0
BISP                0
ehsaas program      0
USAID               0
Credit              0
Loan                0
Interest            0
House Loan          0
Car Loan            0
Food                0
Cinema              0
Cars                0
Birthday            0
Travel              0
Weddings            0
Fitness             0
Cigarette           0
Tourism             0
Hotels              0
Fast Food           0
House for sale      0
Construction        0
Investment          0
Jobs                0
Agriculture         0
FMCG                0
Aviation            0
Manufacturing       0
Textile             0
Economy News        0
Business News       0
World News          0
Politics            0
Newspapers          0
mehngai             0
Real estate         0
deficit             0
elections           0
parliament          0
taxes               0
government          0
budget              0
economic growth     0
subsidy             0
current account     0
trade               0
protest             0
stock market        0
revenue             0
LSM                 0
M0                  0
PSB                 0
CPI                 0
dtype: int64
```

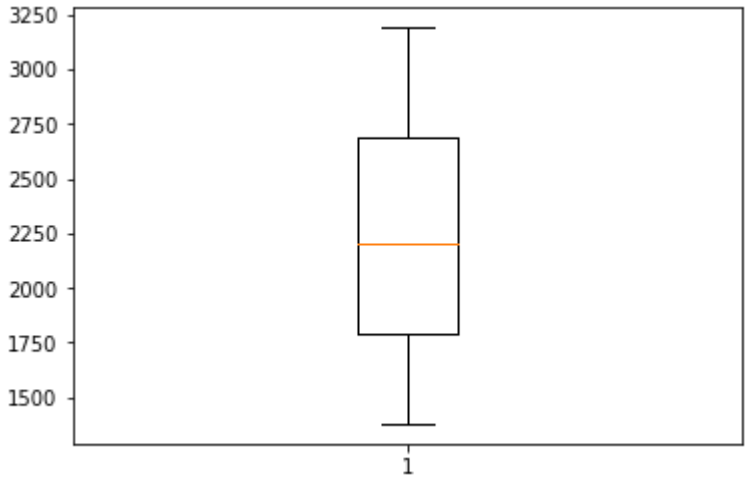
No missing values we can go for further data analysis

In [11]:
```
# Analysing data using template function by ploting the series
data_num(qgdp,col_int = 'all',func = 'all',scat = None)
```

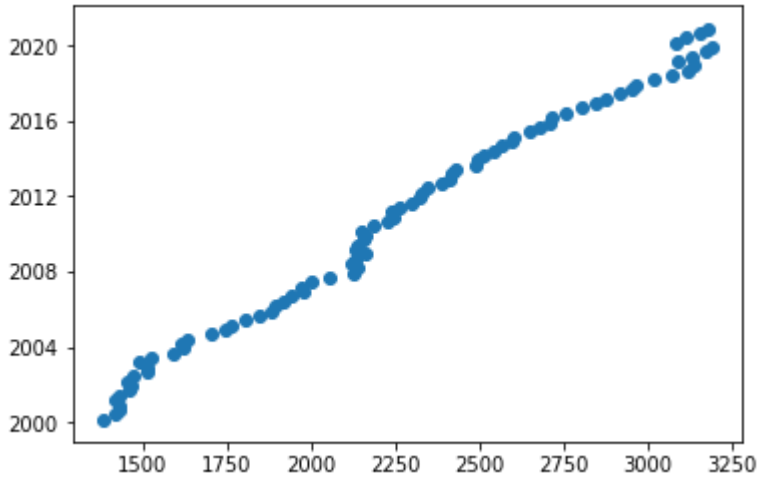
'GDPQ'



'GDPQ'



'GDPQ'

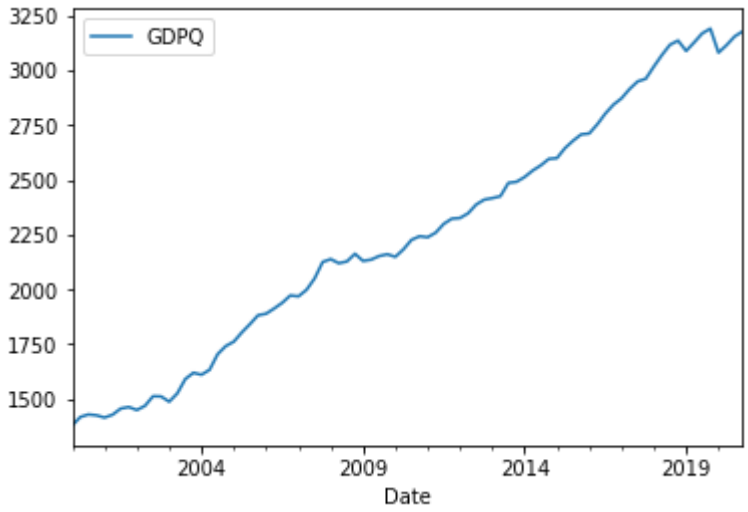


'Statictical Details'

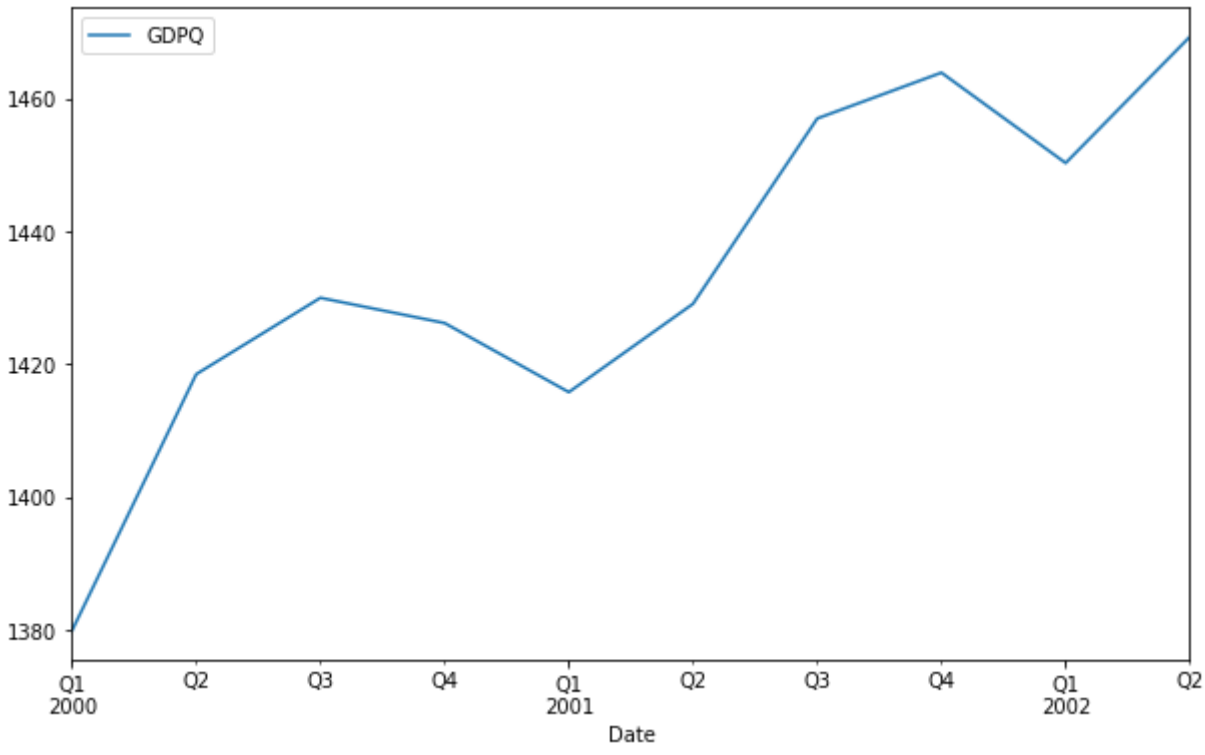
	GDPQ
count	84.000000
mean	2248.413290
std	557.873968
min	1379.900000
25%	1793.950000
50%	2204.800000
75%	2685.650000
max	3189.820283

```
Normality Test for:  GDPQ
Sample does not look Gaussian. Statistics=0.945, p=0.001
```

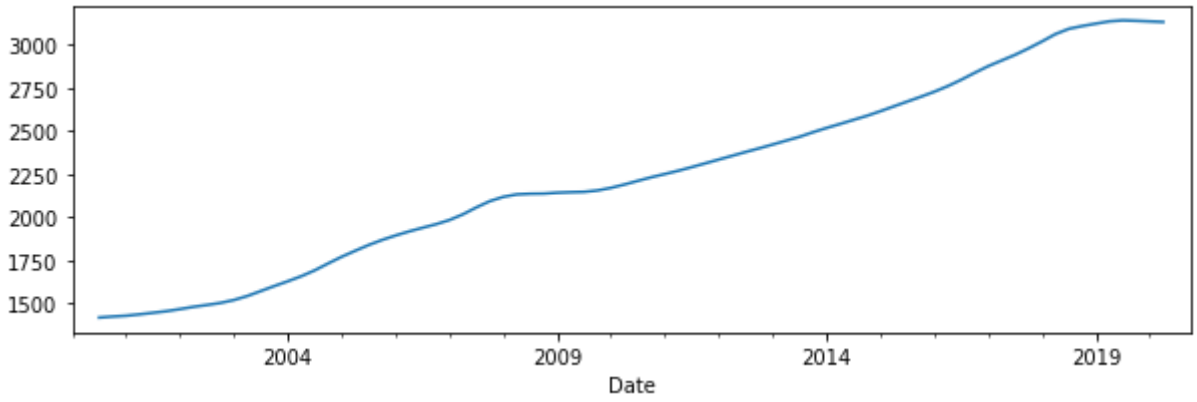
In [12]:
```
# Ploting GDP data
qgdp.plot(x='Date',y='GDPQ')
plt.show()
```

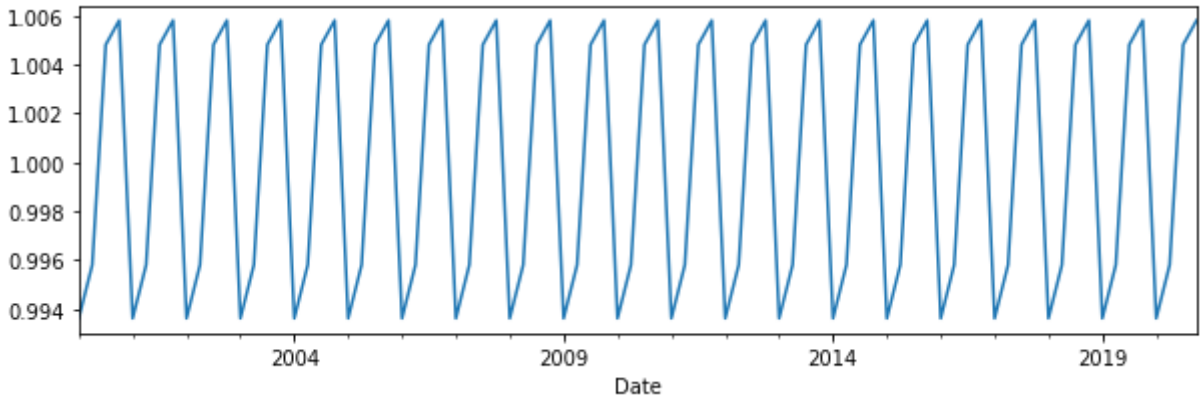


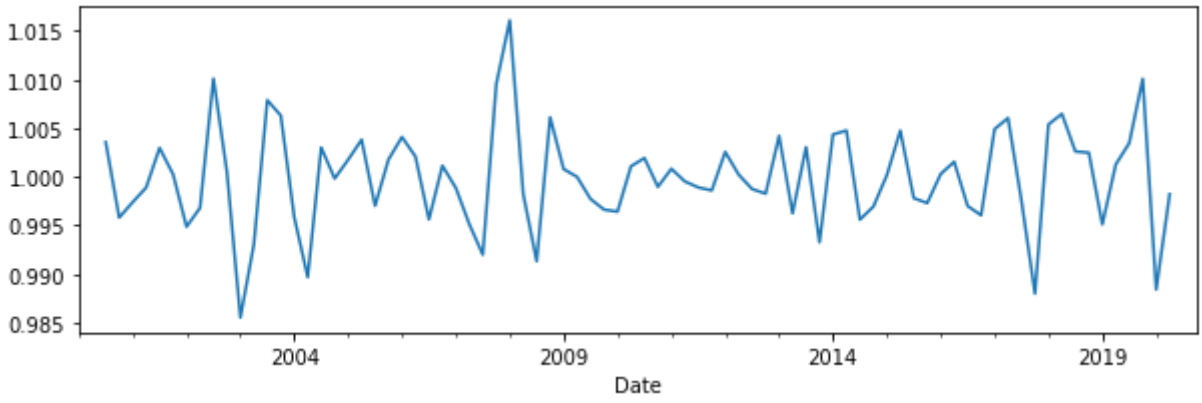
In [13]:
```
qgdp.iloc[0:10].plot(x='Date',y='GDPQ',figsize=(10,6))
plt.show()
```



In [13]:
```
# Decompasing this into seasonal, trend and residual
from statsmodels.tsa.seasonal import seasonal_decompose
qgdp = qgdp.set_index('Date')
result = seasonal_decompose(qgdp, model='multiplicative')
result.trend.plot(figsize=(10,3))
plt.show()
result.seasonal.plot(figsize=(10,3))
plt.show()
result.resid.plot(figsize=(10,3))
plt.show()
```







As the above data is of GDP sereies with obvouis visible trend and seasonality as visible in seasonal decomposition graph so converting it into growth

In [14]:
```
# Taking quater on quarter change and multipling by 100 to make it percent
qgdp['GDPG'] = qgdp['GDPQ'].pct_change(periods=4) * 100
qgdp.head(10)
```

Out[14]:

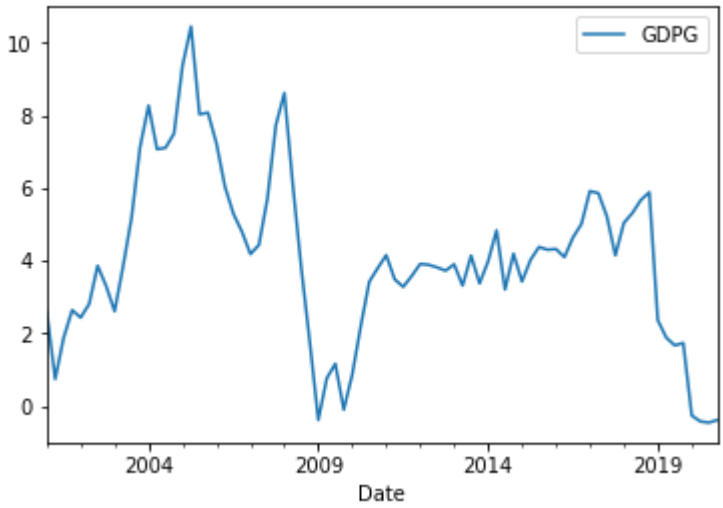
Date	GDPQ	GDPG
2000-03-01	1379.9	NaN
2000-06-01	1418.5	NaN
2000-09-01	1430.0	NaN
2000-12-01	1426.2	NaN
2001-03-01	1415.8	2.601638
2001-06-01	1429.1	0.747268
2001-09-01	1457.0	1.888112
2001-12-01	1463.9	2.643388
2002-03-01	1450.3	2.436785
2002-06-01	1469.3	2.812959

In [15]:
```
# Dropping missing value as after claculating growth data first four quater are NA.
qgdp.dropna(inplace=True)
qgdp.head(10)
```

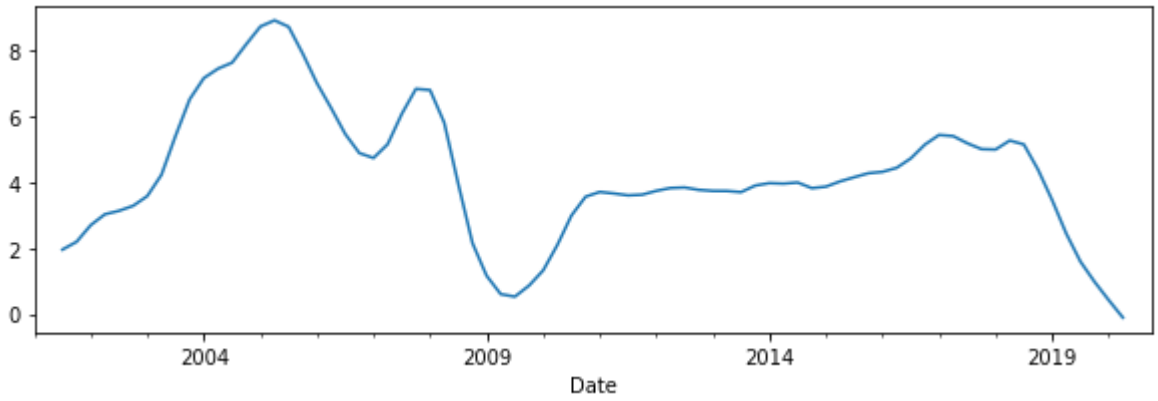
Out[15]:

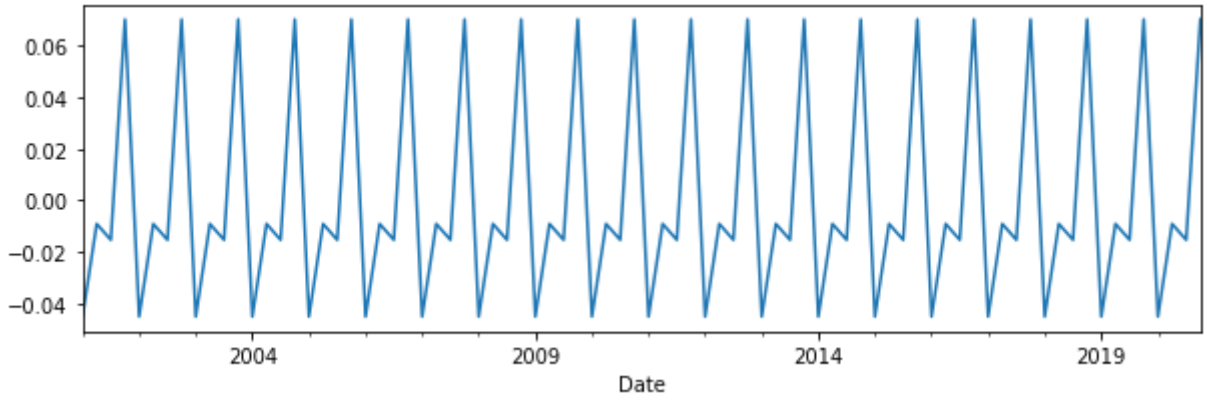
Date	GDPQ	GDPG
2001-03-01	1415.8	2.601638
2001-06-01	1429.1	0.747268
2001-09-01	1457.0	1.888112
2001-12-01	1463.9	2.643388
2002-03-01	1450.3	2.436785
2002-06-01	1469.3	2.812959
2002-09-01	1513.3	3.864104
2002-12-01	1512.3	3.306237
2003-03-01	1488.2	2.613252
2003-06-01	1525.9	3.852175

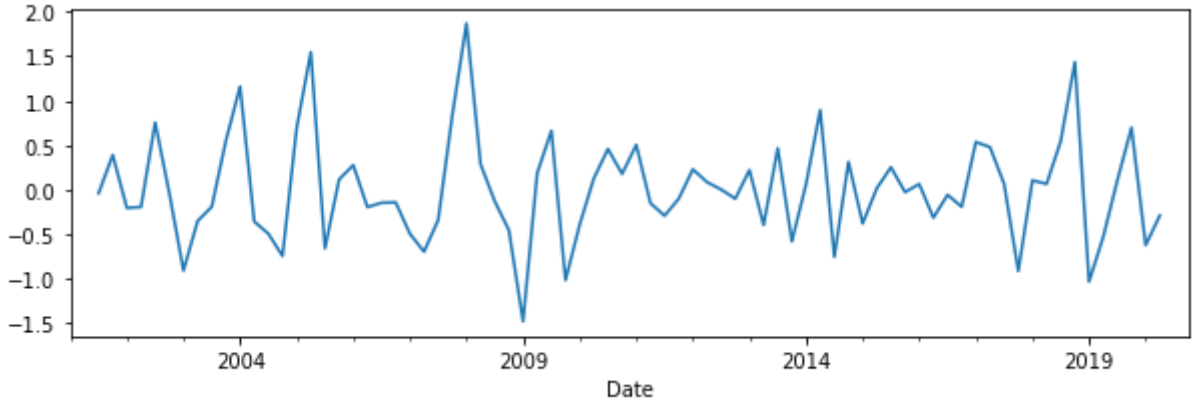
In [17]:
```
# Ploting GDP growth data
qgdp.plot(y='GDPG')
plt.show()
```



In [18]:
```
# Decompasing this into seasonal, trend and residual
result = seasonal_decompose(qgdp['GDPG'], model='additive')
result.trend.plot(figsize=(10,3))
plt.show()
result.seasonal.plot(figsize=(10,3))
plt.show()
result.resid.plot(figsize=(10,3))
plt.show()
```







No trend in data so we may use it for forecasting, but seasonality which will be catered in the Neural Network

Now wrangling through search trends date

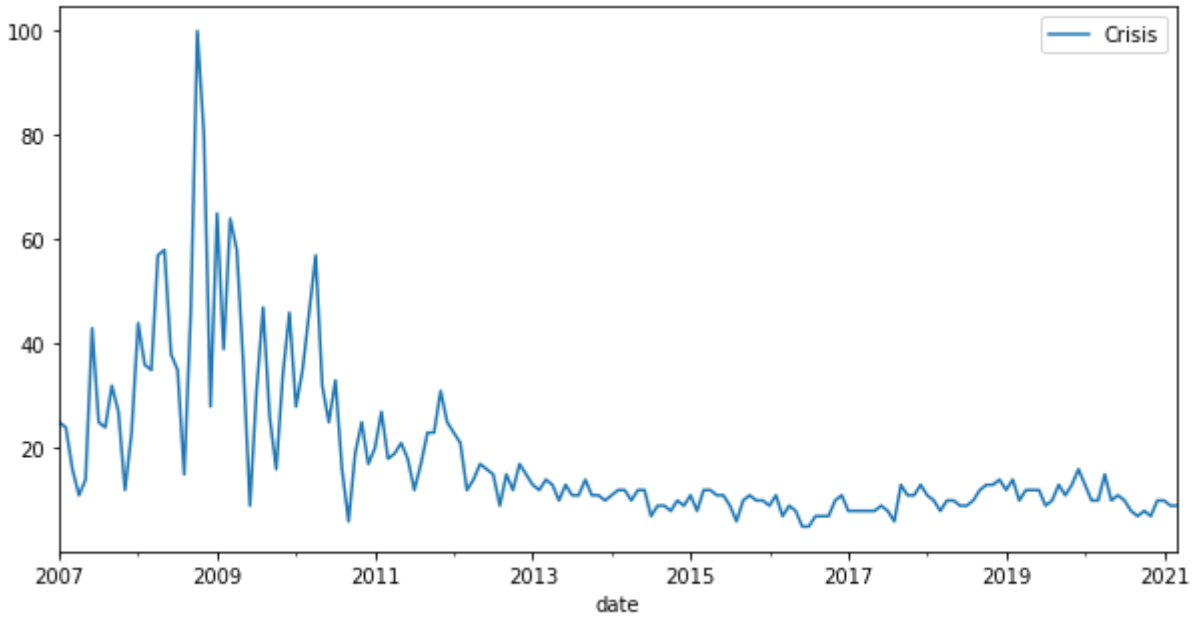
In [19]:
```
gt.head(4)
```

Out[19]:

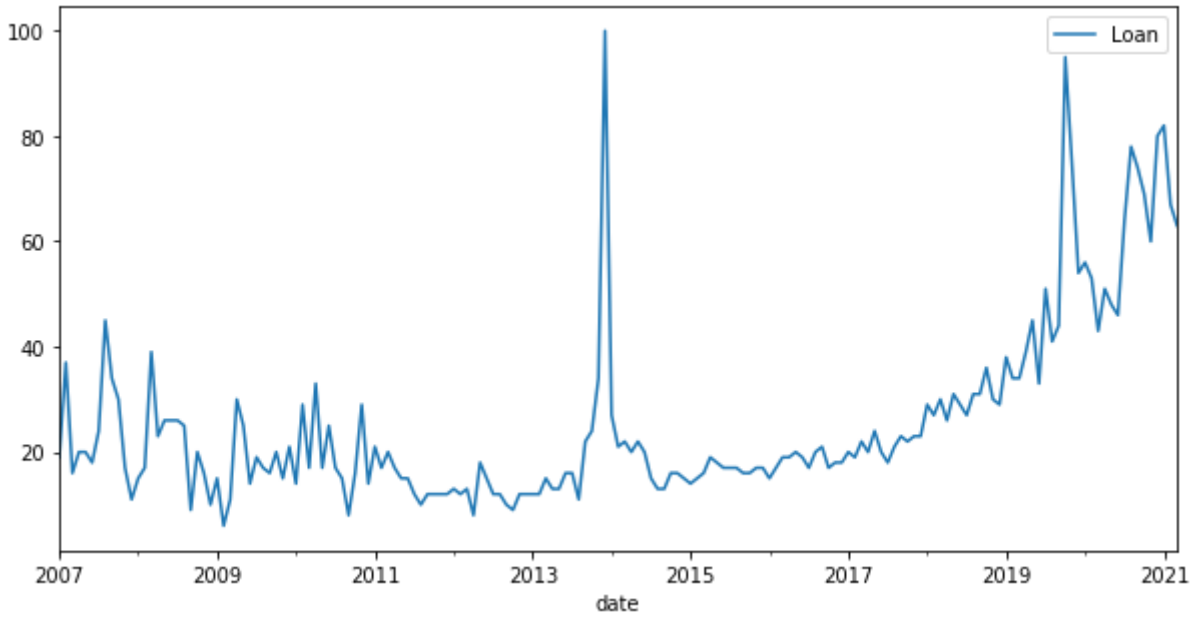
	date	Economic crisis	Crisis	Recession	Financial crisis	Inflation	Unemployment	BISP	ehsaas program	USAID	...	t
0	2007-01-01	0	25	0	0	0	2	0	0	2	...	
1	2007-02-01	6	24	0	0	12	5	0	0	7	...	
2	2007-03-01	0	16	0	0	10	2	0	0	6	...	
3	2007-04-01	0	11	0	0	11	15	0	0	2	...	

4 rows × 55 columns

In [20]:
```
# Ploting GDP data
gt.plot(x='date',y='Crisis',figsize=(10,5))
plt.show()
```



In [21]:
```
# Ploting GDP data
gt.plot(x='date',y='Loan',figsize=(10,5))
plt.show()
```



The intuition behinde using the google trends data for GDP forecasting is that human search behaviour is mostly driven by the underling situtaion as visible in the above graphs that in time of booms people are searching for the loan, and in crices like in 2008 people searching for crisis as so on

In [16]:
```
# As this data is monthly frequency but GDP is quaterlyso we have to convert it into
 quarterly data.
# Making a new data fram with all data quaterley
d = {'Quaters':pd.Series(pd.period_range("1-1-2007","12-1-2020",freq="Q"))}
df = pd.DataFrame(data=d)
df['GDP'] = qgdp.loc['2007-01-01':'2020-12-01',:]['GDPG'].values
```

In [11]:
```
df.head(10)
```

Out[11]:

	Quaters	GDP
0	2007Q1	4.185850
1	2007Q2	4.436896
2	2007Q3	5.707362
3	2007Q4	7.734807
4	2008Q1	8.619464
5	2008Q2	6.079864
6	2008Q3	3.784812
7	2008Q4	1.754881
8	2009Q1	-0.378770
9	2009Q2	0.768904

In [24]:
```
df.shape
```

Out[24]:
```
(56, 2)
```

In [17]:

```
#Resampling the monthly trends data to quarterly data
gt['date'] = pd.date_range('1/1/2007','3/31/2021', freq='M')
gt['date'] = gt[gt['date'] < '12/31/2020']
gt = gt.set_index('date')
gtq = gt.resample('Q', convention='start',).sum()
gtq
```

Out[17]:

date	Economic crisis	Crisis	Recession	Financial crisis	Inflation	Unemployment	BISP	ehsaas program	USAID	Credit
2007-03-31	6	65	0	0	22	9	0	0	15	138
2007-06-30	0	68	0	0	81	28	0	0	21	135
2007-09-30	0	81	0	0	55	19	2	0	12	226
2007-12-31	4	62	0	0	75	15	0	0	9	153
2008-03-31	0	115	8	0	56	18	0	0	13	146
2008-06-30	0	153	8	8	113	19	0	0	30	167
2008-09-30	9	96	9	5	103	19	2	0	16	114
2008-12-31	39	208	30	76	125	7	4	0	14	143
2009-03-31	32	168	60	42	90	20	2	0	9	121
2009-06-30	13	103	31	26	111	14	4	0	33	118
2009-09-30	24	104	33	28	47	4	12	0	37	124
2009-12-31	3	96	17	17	65	11	22	0	20	97
2010-03-31	3	109	18	13	58	16	23	0	25	126
2010-06-30	6	114	14	8	32	12	20	0	28	140
2010-09-30	3	55	14	5	32	5	5	0	26	100
2010-12-31	4	61	13	3	67	11	9	0	29	97
2011-03-31	4	65	7	7	43	10	35	0	13	78
2011-06-30	2	58	3	4	50	10	136	0	15	64
2011-09-30	1	52	6	4	24	5	230	0	17	66
2011-12-31	3	79	6	3	32	7	87	0	14	67
2012-03-31	2	56	5	4	33	7	78	0	13	53
2012-06-30	3	47	4	5	35	6	107	0	16	62
2012-09-30	2	39	3	2	14	5	86	0	13	58
2012-12-31	4	44	5	3	25	7	95	0	19	65
2013-03-31	2	39	3	3	26	6	81	0	12	67

date	Economic crisis	Crisis	Recession	Financial crisis	Inflation	Unemployment	BISP	ehsaas program	USAID	Credit
2013-06-30	1	36	2	2	30	7	48	0	14	58
2013-09-30	1	36	3	3	15	4	73	0	11	53
2013-12-31	1	32	4	3	18	5	29	0	8	48
2014-03-31	2	35	3	4	21	6	41	0	7	58
2014-06-30	1	34	4	2	23	6	40	0	11	53
2014-09-30	1	25	1	0	12	3	29	0	11	49
2014-12-31	0	27	2	2	16	5	26	0	9	55
2015-03-31	0	31	2	2	17	5	20	0	6	59
2015-06-30	1	34	3	2	22	6	14	0	9	57
2015-09-30	1	25	2	3	11	3	10	0	6	50
2015-12-31	2	31	3	2	16	6	6	0	6	61
2016-03-31	0	27	2	2	18	4	7	0	6	50
2016-06-30	0	22	3	1	19	5	7	0	5	50
2016-09-30	2	19	3	0	10	3	6	0	4	48
2016-12-31	0	28	3	0	16	5	5	0	6	50
2017-03-31	0	24	3	2	17	4	4	0	4	55
2017-06-30	1	25	1	2	14	5	8	0	4	59
2017-09-30	0	27	3	0	11	3	6	0	3	62
2017-12-31	2	35	5	2	19	5	5	0	5	67
2018-03-31	0	29	3	1	17	5	3	0	3	64
2018-06-30	0	29	3	2	17	6	4	0	3	67
2018-09-30	2	31	4	1	15	3	3	0	3	63
2018-12-31	3	40	3	2	22	5	4	0	3	86
2019-03-31	2	36	4	2	23	5	4	0	3	78
2019-06-30	3	36	4	2	28	5	5	0	3	87
2019-09-30	3	32	5	0	19	3	3	0	3	88

date	Economic crisis	Crisis	Recession	Financial crisis	Inflation	Unemployment	BISP	ehsaas program	USAID	Credit
2019-12-31	2	40	5	2	27	7	6	5	2	100
2020-03-31	2	33	5	3	26	6	10	7	1	95
2020-06-30	2	36	9	3	20	6	19	153	2	87
2020-09-30	1	25	5	1	21	6	9	62	1	113
2020-12-31	0	15	4	2	20	5	6	15	0	73

56 rows × 58 columns

In [18]:
```
# Merging this trends data to newly created dataframe.
for (columnName, columnData) in gtq.iteritems():
    df[columnName] = columnData.values
```

In [19]:
```
df.head(10)
```

Out[19]:

	Quaters	GDP	Economic crisis	Crisis	Recession	Financial crisis	Inflation	Unemployment	BISP	ehsaas program
0	2007Q1	4.185850	6	65	0	0	22	9	0	0
1	2007Q2	4.436896	0	68	0	0	81	28	0	0
2	2007Q3	5.707362	0	81	0	0	55	19	2	0
3	2007Q4	7.734807	4	62	0	0	75	15	0	0
4	2008Q1	8.619464	0	115	8	0	56	18	0	0
5	2008Q2	6.079864	0	153	8	8	113	19	0	0
6	2008Q3	3.784812	9	96	9	5	103	19	2	0
7	2008Q4	1.754881	39	208	30	76	125	7	4	0
8	2009Q1	-0.378770	32	168	60	42	90	20	2	0
9	2009Q2	0.768904	13	103	31	26	111	14	4	0

10 rows × 60 columns

Data frame is ready for furthe processing and training

In [20]:
```
df_details(df,5)
```

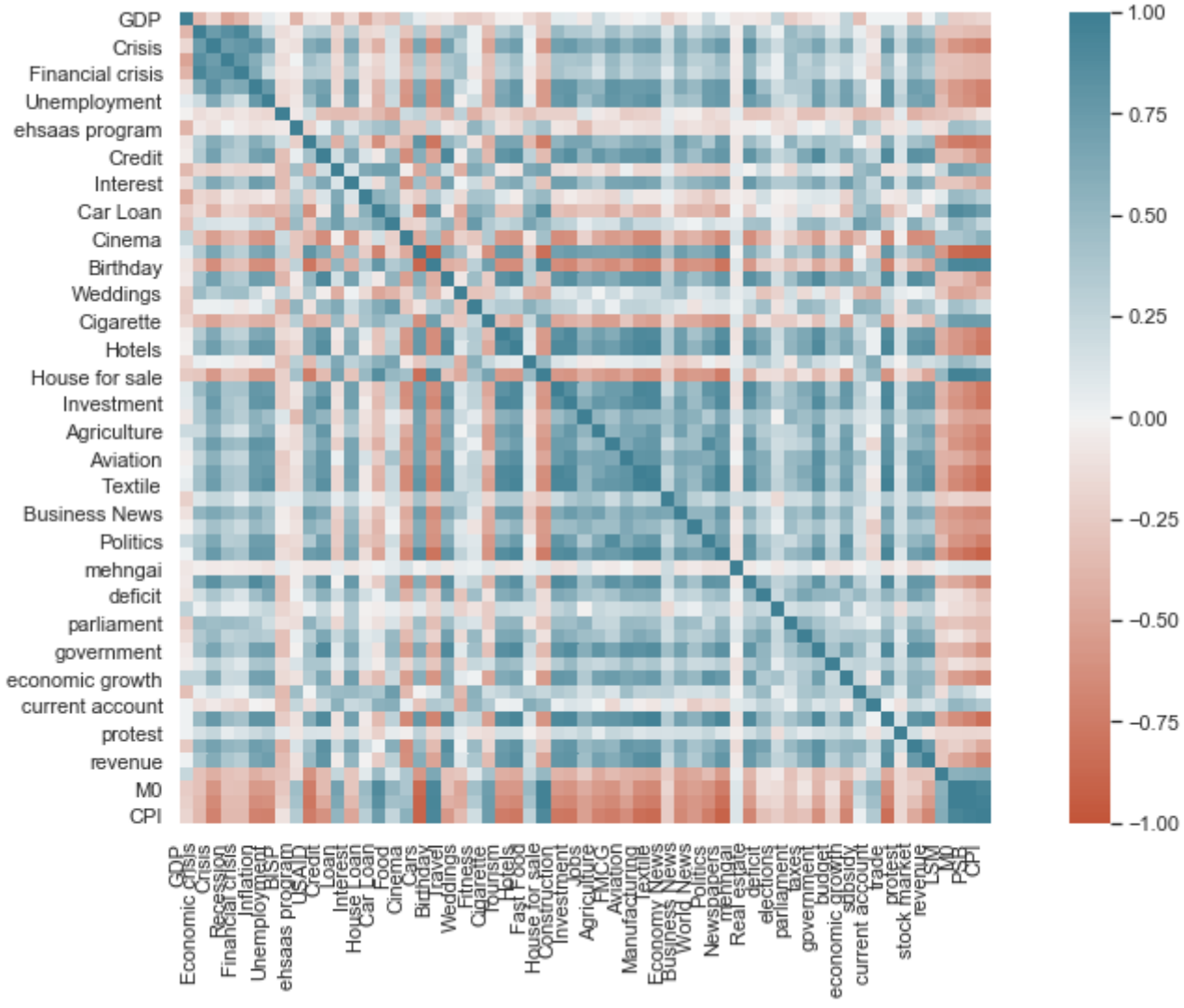
```
Data Types of Column:
 Quaters             period[Q-DEC]
GDP                          float64
Economic crisis                int64
Crisis                         int64
Recession                      int64
Financial crisis               int64
Inflation                      int64
Unemployment                   int64
BISP                           int64
ehsaas program                 int64
USAID                          int64
Credit                         int64
Loan                           int64
Interest                       int64
House Loan                     int64
Car Loan                       int64
Food                           int64
Cinema                         int64
Cars                           int64
Birthday                       int64
Travel                         int64
Weddings                       int64
Fitness                        int64
Cigarette                      int64
Tourism                        int64
Hotels                         int64
Fast Food                      int64
House for sale                 int64
Construction                   int64
Investment                     int64
Jobs                           int64
Agriculture                    int64
FMCG                           int64
Aviation                       int64
Manufacturing                  int64
Textile                        int64
Economy News                   int64
Business News                  int64
World News                     int64
Politics                       int64
Newspapers                     int64
mehngai                        int64
Real estate                    int64
deficit                        int64
elections                      int64
parliament                     int64
taxes                          int64
government                     int64
budget                         int64
economic growth                int64
subsidy                        int64
current account                int64
trade                          int64
protest                        int64
stock market                   int64
revenue                        int64
LSM                          float64
M0                           float64
PSB                          float64
CPI                          float64
dtype: object

 Size of Datarame:  (56, 60)
```

Top and bottom 5 rows:

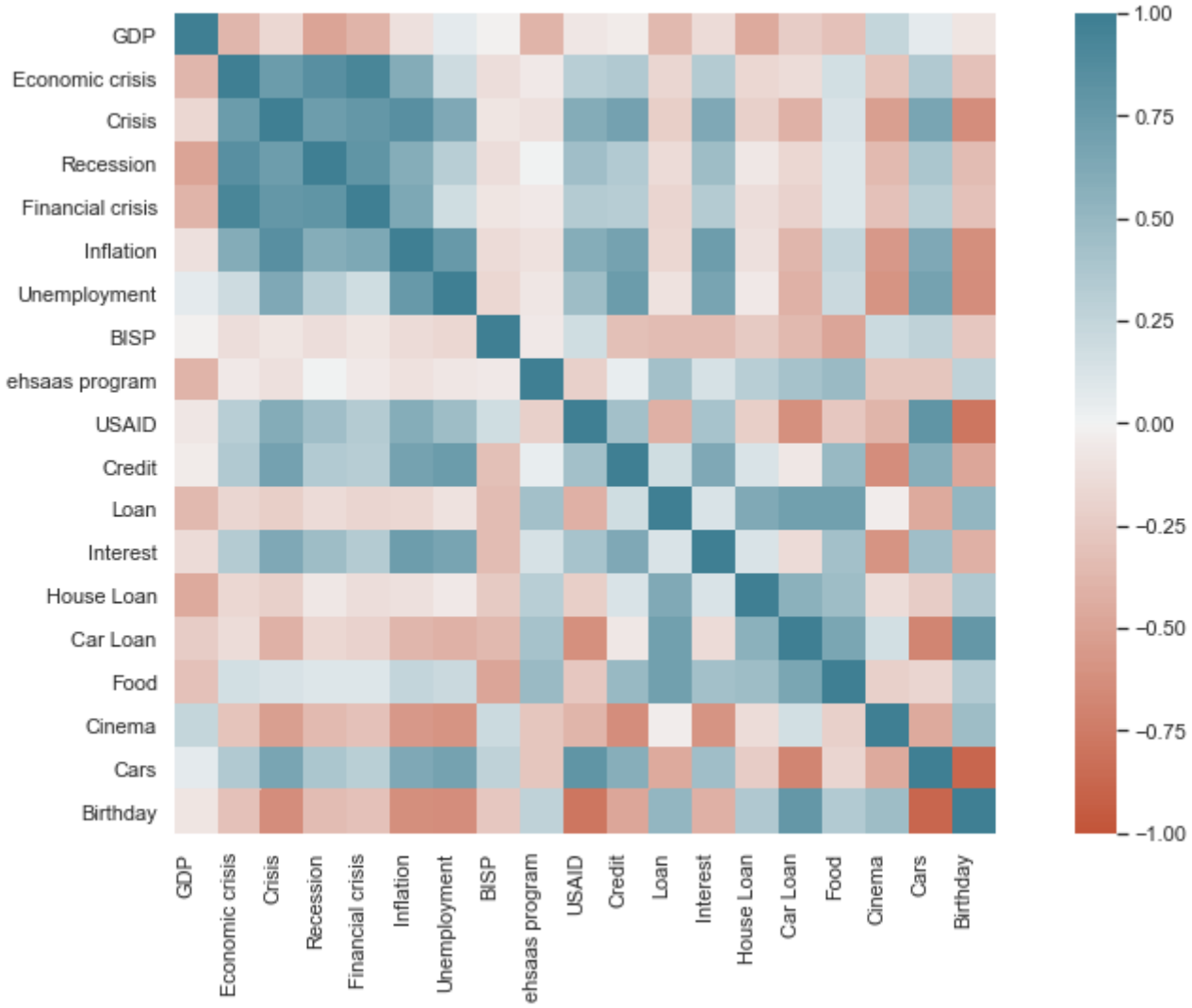
	Quaters	GDP	Economic crisis	Crisis	Recession	Financial crisis	Inflation	Unemployment	BISP	ehsaas program
0	2007Q1	4.185850	6	65	0	0	22	9	0	0
1	2007Q2	4.436896	0	68	0	0	81	28	0	0
2	2007Q3	5.707362	0	81	0	0	55	19	2	0
3	2007Q4	7.734807	4	62	0	0	75	15	0	0
4	2008Q1	8.619464	0	115	8	0	56	18	0	0
51	2019Q4	1.736338	2	40	5	2	27	7	6	5
52	2020Q1	-0.258280	2	33	5	3	26	6	10	7
53	2020Q2	-0.419317	2	36	9	3	20	6	19	153
54	2020Q3	-0.454332	1	25	5	1	21	6	9	62
55	2020Q4	-0.383455	0	15	4	2	20	5	6	15

10 rows × 60 columns

In [21]:
```
# Ploting correlation heat map to see the relation between the data in trend values with trends data
corr_hmap(df)
```



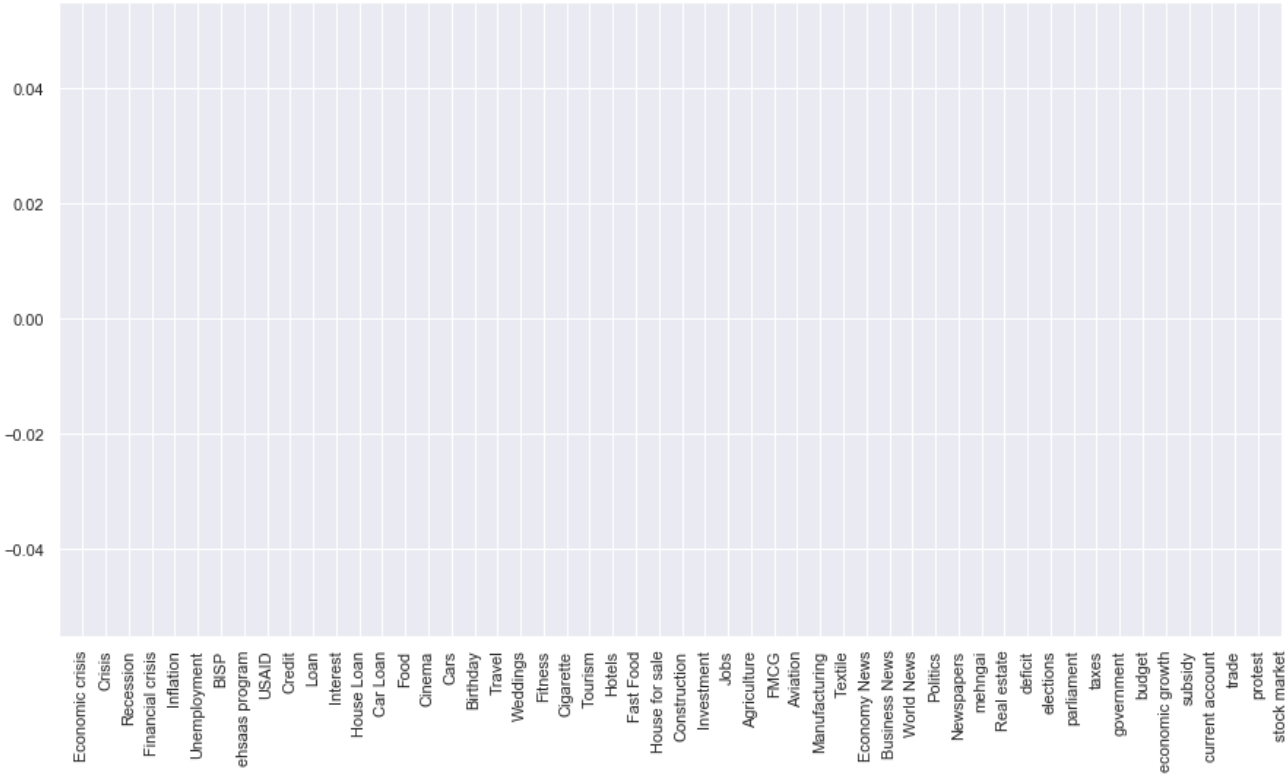
In [187]:
```
corr_hmap(df.iloc[:,:20])
```



In [30]:

```
x_fs,y,fs = feature_selectk(df.iloc[:,1:],54,'GDP')
# Displaying the fcore of calculating for each varible
for i in range(len(fs.scores_)):
    print('Feature %d: %f' % (i, fs.scores_[i]))
# plot the scores
plt.bar([i for i in range(len(fs.scores_))], fs.scores_)
plt.xticks(range(1,len(fs.scores_)),df.iloc[:,2:len(df)-1].columns,rotation=90)
plt.show()
```

```
Feature 0: nan
Feature 1: nan
Feature 2: nan
Feature 3: nan
Feature 4: nan
Feature 5: nan
Feature 6: nan
Feature 7: nan
Feature 8: nan
Feature 9: nan
Feature 10: nan
Feature 11: nan
Feature 12: nan
Feature 13: nan
Feature 14: nan
Feature 15: nan
Feature 16: nan
Feature 17: nan
Feature 18: nan
Feature 19: nan
Feature 20: nan
Feature 21: nan
Feature 22: nan
Feature 23: nan
Feature 24: nan
Feature 25: nan
Feature 26: nan
Feature 27: nan
Feature 28: nan
Feature 29: nan
Feature 30: nan
Feature 31: nan
Feature 32: nan
Feature 33: nan
Feature 34: nan
Feature 35: nan
Feature 36: nan
Feature 37: nan
Feature 38: nan
Feature 39: nan
Feature 40: nan
Feature 41: nan
Feature 42: nan
Feature 43: nan
Feature 44: nan
Feature 45: nan
Feature 46: nan
Feature 47: nan
Feature 48: nan
Feature 49: nan
Feature 50: nan
Feature 51: nan
Feature 52: nan
Feature 53: nan
```

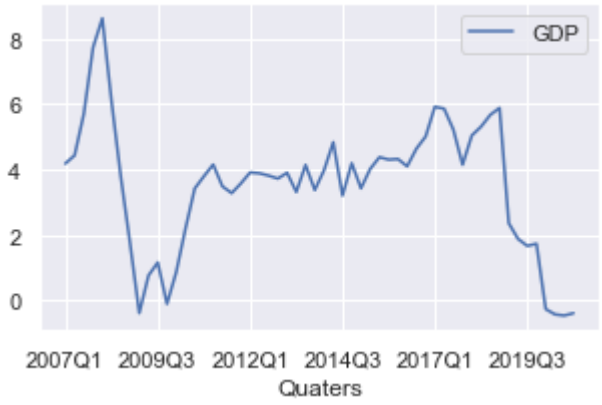


In [22]:
```
x = df.iloc[:,2:]
y = df['GDP']
a = feature_select_rffs(x,y,reg=True).values
a
```

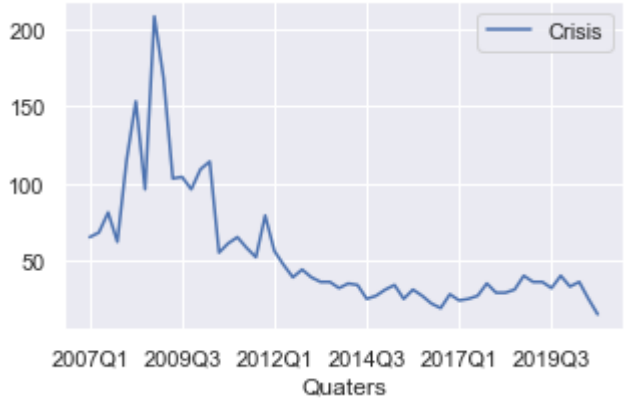
Out[22]:
```
array(['Economic crisis', 'Recession', 'Financial crisis', 'BISP',
       'ehsaas program', 'USAID', 'Credit', 'Loan', 'House Loan',
       'Car Loan', 'Food', 'Cinema', 'Cigarette', 'Agriculture',
       'Newspapers', 'deficit', 'subsidy'], dtype=object)
```

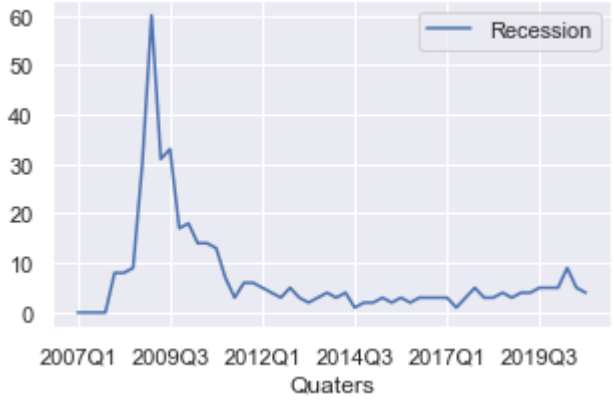
In [32]:
```
df.Quaters=df.Quaters.astype(str)
```

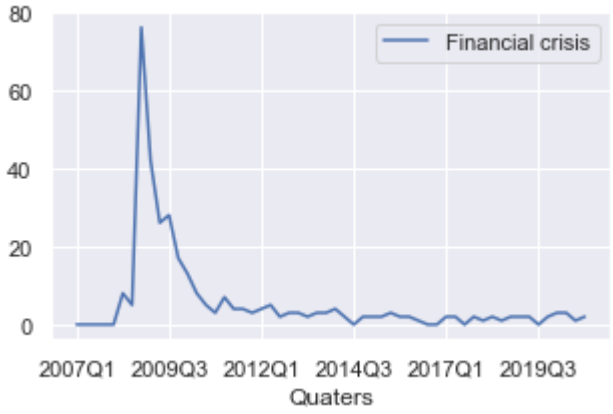
In [33]:
```
# Line plots of all features
for (columnName, columnData) in df.iloc[:,1:].iteritems():
    df.plot(x = 'Quaters',y=columnName,figsize=(5,3))
    plt.show()
```

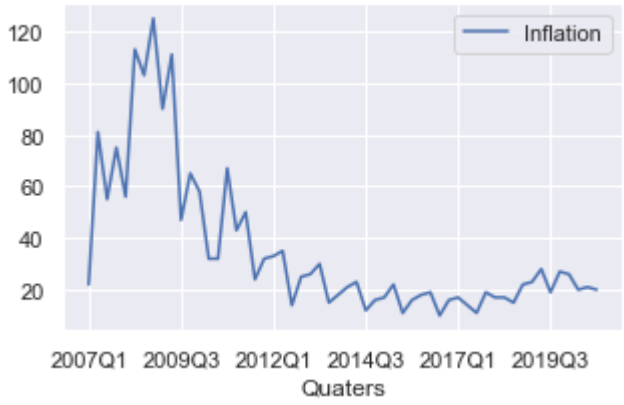

GDP
8
6
4
2
0
2007Q1
2009Q3
2012Q1
2014Q3
2017Q1
2019Q3
Quaters

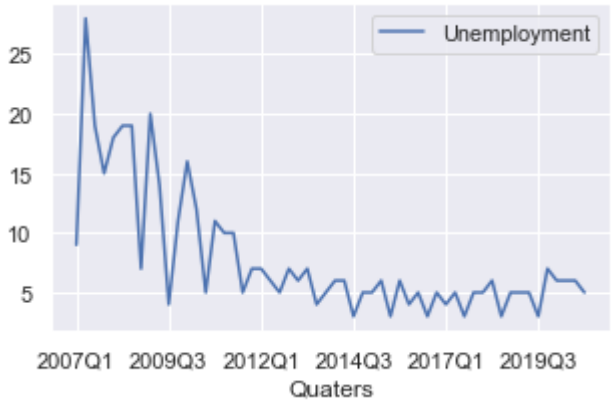

Economic crisis
40
30
20
10
0
2007Q1
2009Q3
2012Q1
2014Q3
2017Q1
2019Q3
Quaters

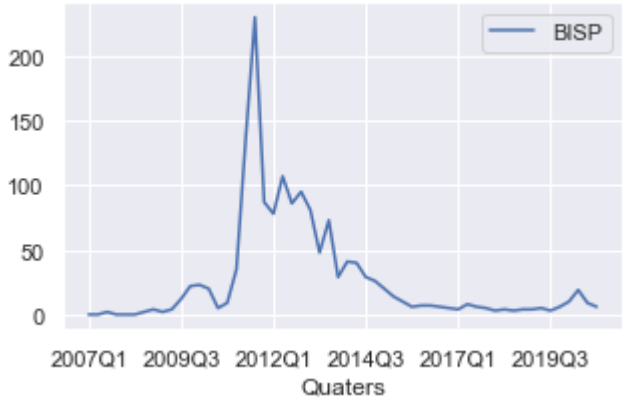

Crisis
200
150
100
50
2007Q1
2009Q3
2012Q1
2014Q3
2017Q1
2019Q3
Quaters

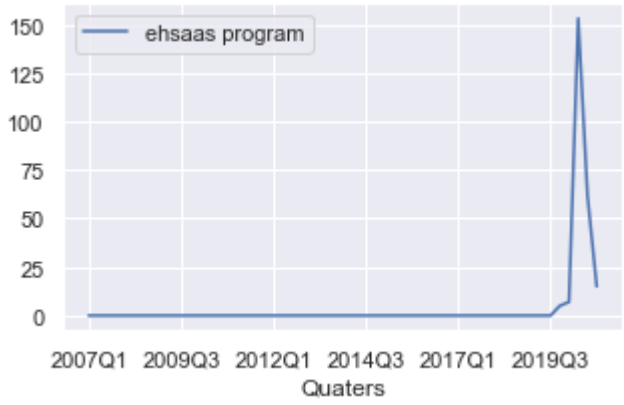

Recession
60
50
40
30
20
10
0
2007Q1
2009Q3
2012Q1
2014Q3
2017Q1
2019Q3
Quaters

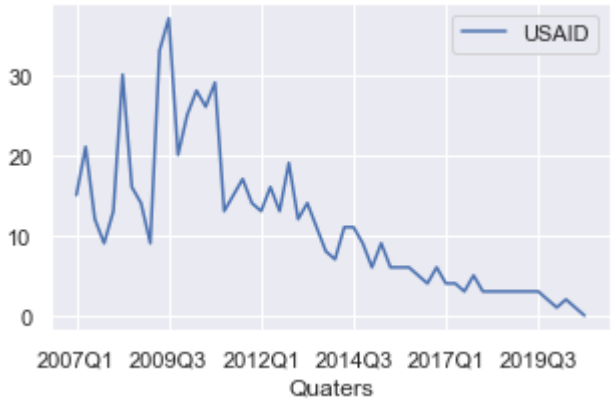

Financial crisis
80
60
40
20
0
2007Q1
2009Q3
2012Q1
2014Q3
2017Q1
2019Q3
Quaters

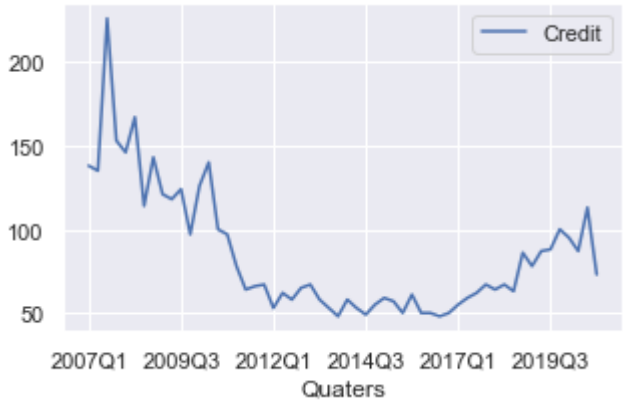

Inflation
120
100
80
60
40
20
2007Q1
2009Q3
2012Q1
2014Q3
2017Q1
2019Q3
Quaters


Unemployment
25
20
15
10
5
2007Q1
2009Q3
2012Q1
2014Q3
2017Q1
2019Q3
Quaters


BISP
200
150
100
50
0
2007Q1
2009Q3
2012Q1
2014Q3
2017Q1
2019Q3
Quaters

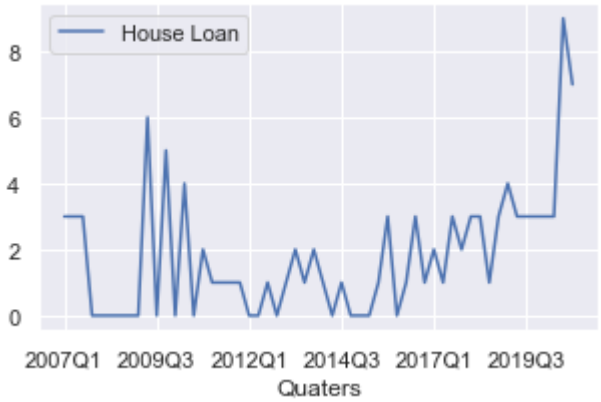

ehsaas program
150
125
100
75
50
25
0
2007Q1
2009Q3
2012Q1
2014Q3
2017Q1
2019Q3
Quaters


USAID
30
20
10
0
2007Q1
2009Q3
2012Q1
2014Q3
2017Q1
2019Q3
Quaters


Credit
200
150
100
50
2007Q1
2009Q3
2012Q1
2014Q3
2017Q1
2019Q3
Quaters

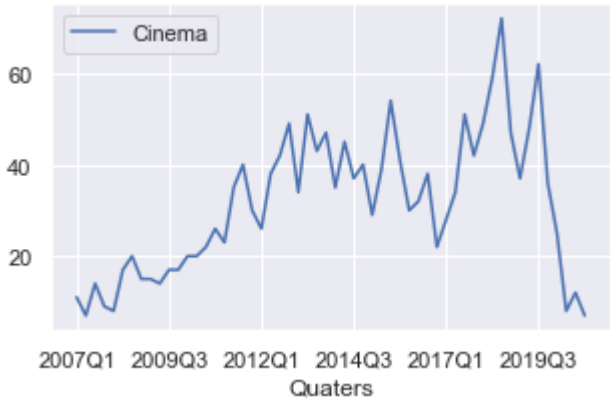

Loan
200
150
100
50
2007Q1
2009Q3
2012Q1
2014Q3
2017Q1
2019Q3
Quaters


Interest
70
60
50
40
30
20
2007Q1
2009Q3
2012Q1
2014Q3
2017Q1
2019Q3
Quaters


House Loan
8
6
4
2
0
2007Q1
2009Q3
2012Q1
2014Q3
2017Q1
2019Q3
Quaters

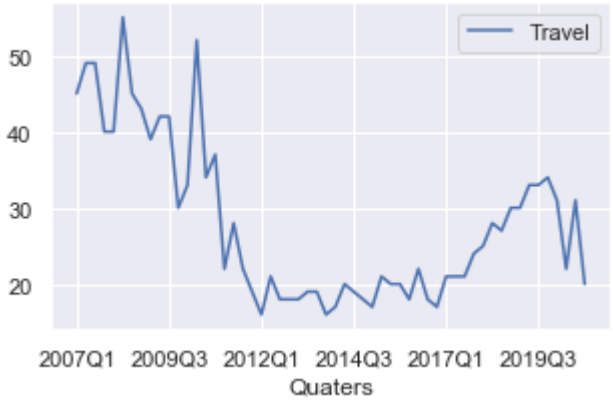

Car Loan
20
15
10
5
0
2007Q1
2009Q3
2012Q1
2014Q3
2017Q1
2019Q3
Quaters

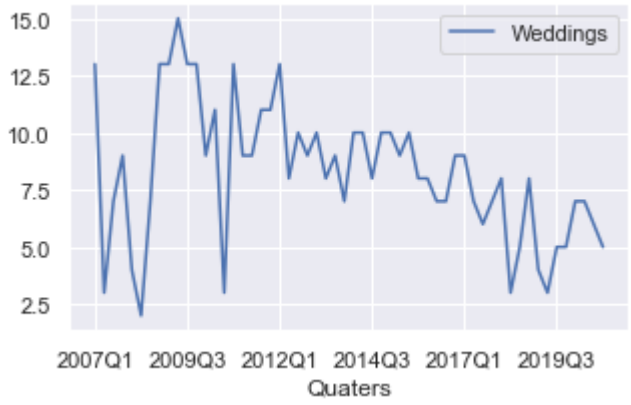

Food
140
120
100
80
60
2007Q1
2009Q3
2012Q1
2014Q3
2017Q1
2019Q3
Quaters


Cinema
60
40
20
2007Q1
2009Q3
2012Q1
2014Q3
2017Q1
2019Q3
Quaters

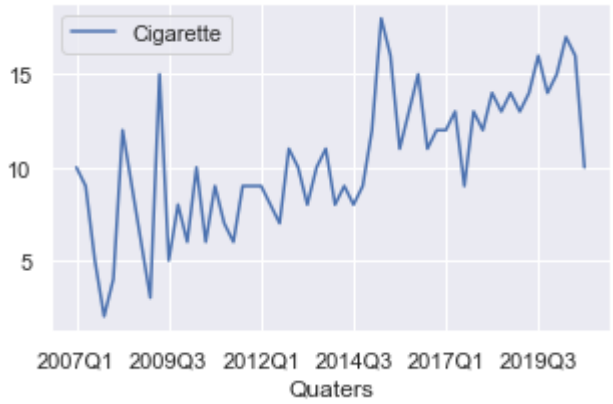

Cars
140
120
100
80
60
40
2007Q1
2009Q3
2012Q1
2014Q3
2017Q1
2019Q3
Quaters

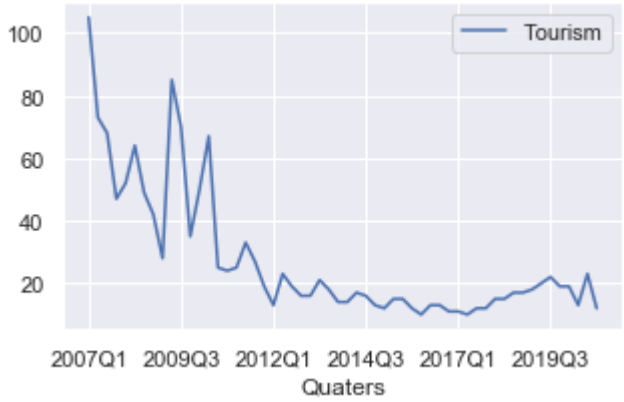

Birthday
250
200
150
100
50
2007Q1
2009Q3
2012Q1
2014Q3
2017Q1
2019Q3
Quaters

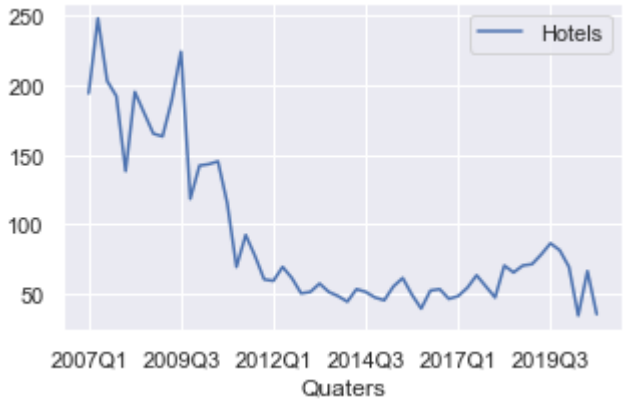

Travel
50
40
30
20
2007Q1
2009Q3
2012Q1
2014Q3
2017Q1
2019Q3
Quaters

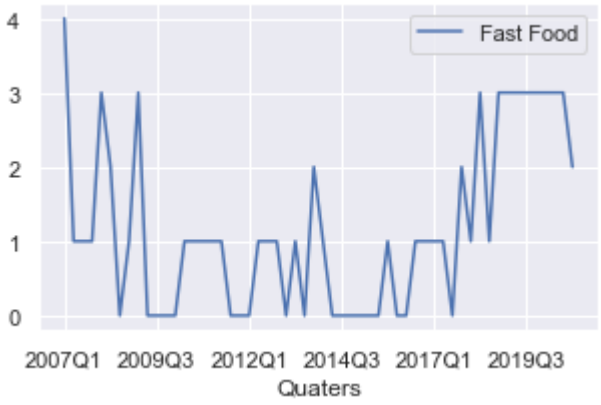

Weddings
15.0
12.5
10.0
7.5
5.0
2.5
2007Q1
2009Q3
2012Q1
2014Q3
2017Q1
2019Q3
Quaters

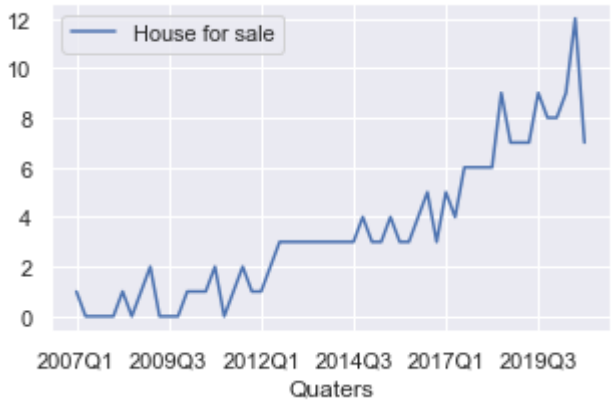

Fitness
45
40
35
30
25
20
15
2007Q1
2009Q3
2012Q1
2014Q3
2017Q1
2019Q3
Quaters

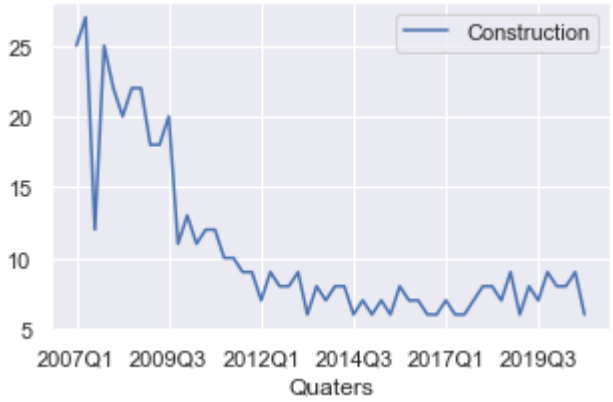

Cigarette
15
10
5
2007Q1
2009Q3
2012Q1
2014Q3
2017Q1
2019Q3
Quaters

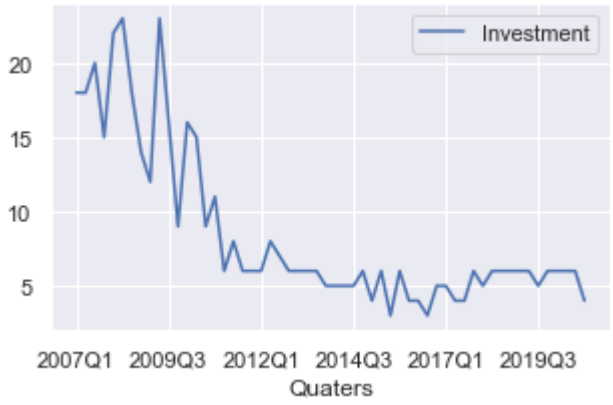

Tourism
100
80
60
40
20
2007Q1
2009Q3
2012Q1
2014Q3
2017Q1
2019Q3
Quaters


Hotels
250
200
150
100
50
2007Q1
2009Q3
2012Q1
2014Q3
2017Q1
2019Q3
Quaters

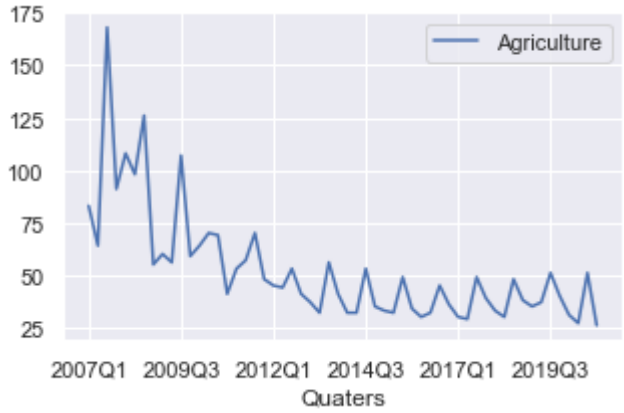

Fast Food
4
3
2
1
0
2007Q1
2009Q3
2012Q1
2014Q3
2017Q1
2019Q3
Quaters

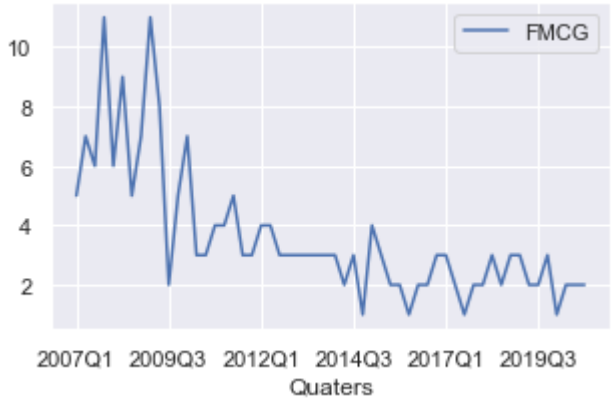

House for sale
12
10
8
6
4
2
0
2007Q1
2009Q3
2012Q1
2014Q3
2017Q1
2019Q3
Quaters

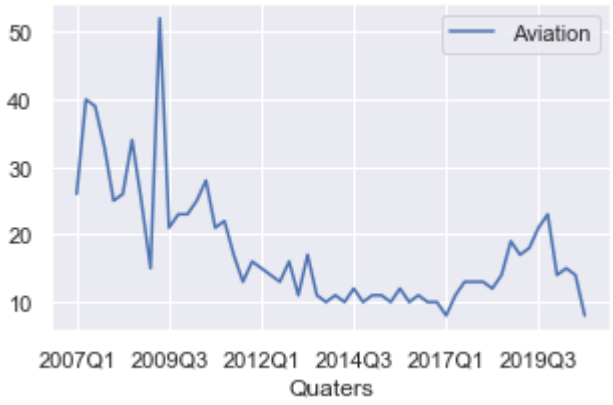

Construction
25
20
15
10
5
2007Q1
2009Q3
2012Q1
2014Q3
2017Q1
2019Q3
Quaters


Investment
20
15
10
5
2007Q1
2009Q3
2012Q1
2014Q3
2017Q1
2019Q3
Quaters


Jobs
275
250
225
200
175
150
125
2007Q1
2009Q3
2012Q1
2014Q3
2017Q1
2019Q3
Quaters


Agriculture
175
150
125
100
75
50
25
2007Q1
2009Q3
2012Q1
2014Q3
2017Q1
2019Q3
Quaters

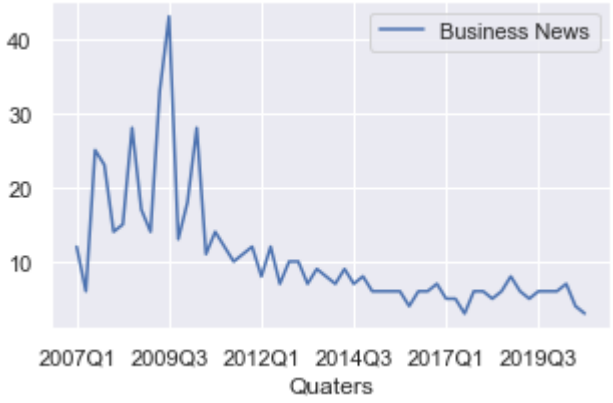

FMCG
10
8
6
4
2
2007Q1
2009Q3
2012Q1
2014Q3
2017Q1
2019Q3
Quaters

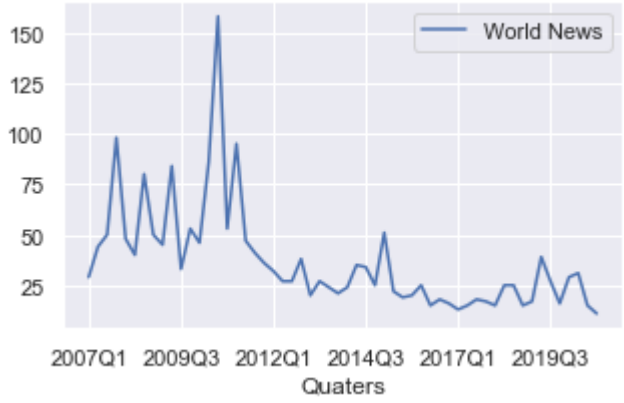

Aviation
50
40
30
20
10
2007Q1
2009Q3
2012Q1
2014Q3
2017Q1
2019Q3
Quaters

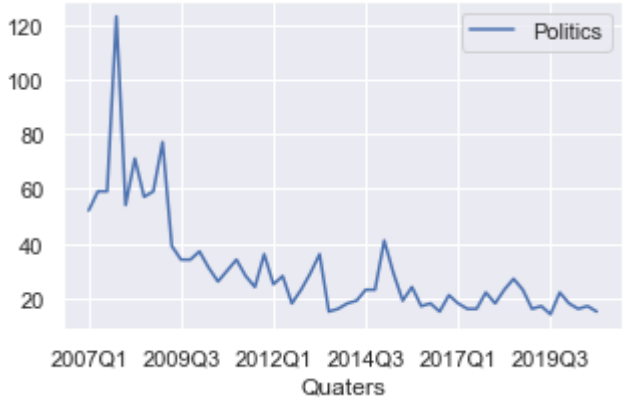

Manufacturing
100
80
60
40
20
2007Q1
2009Q3
2012Q1
2014Q3
2017Q1
2019Q3
Quaters

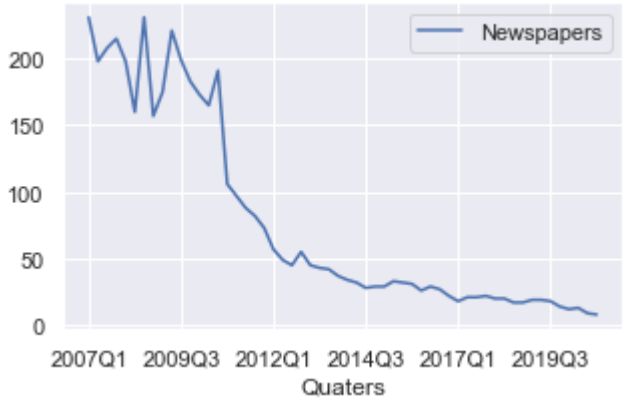

Textile
250
200
150
100
50
2007Q1
2009Q3
2012Q1
2014Q3
2017Q1
2019Q3
Quaters

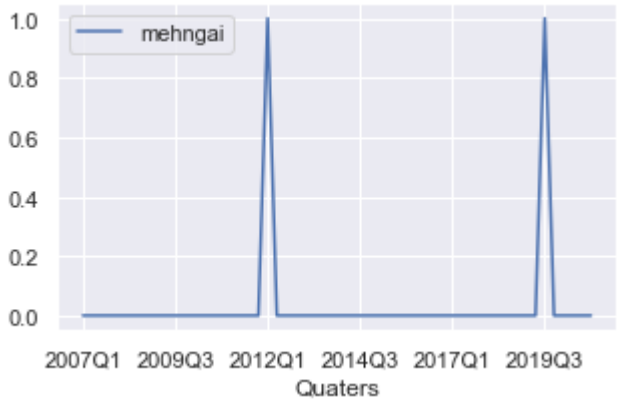

Economy News
3.0
2.5
2.0
1.5
1.0
0.5
0.0
2007Q1
2009Q3
2012Q1
2014Q3
2017Q1
2019Q3
Quaters


Business News
40
30
20
10
2007Q1
2009Q3
2012Q1
2014Q3
2017Q1
2019Q3
Quaters

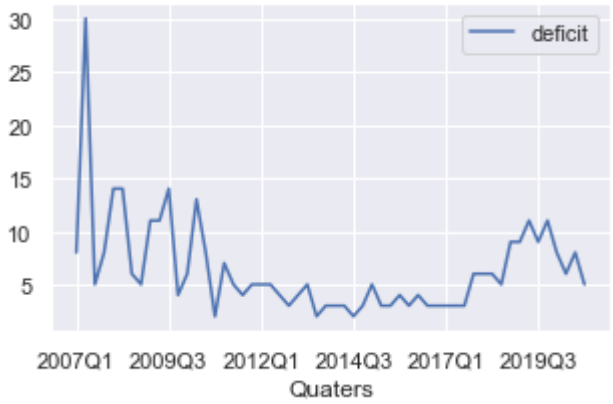

World News
150
125
100
75
50
25
2007Q1
2009Q3
2012Q1
2014Q3
2017Q1
2019Q3
Quaters

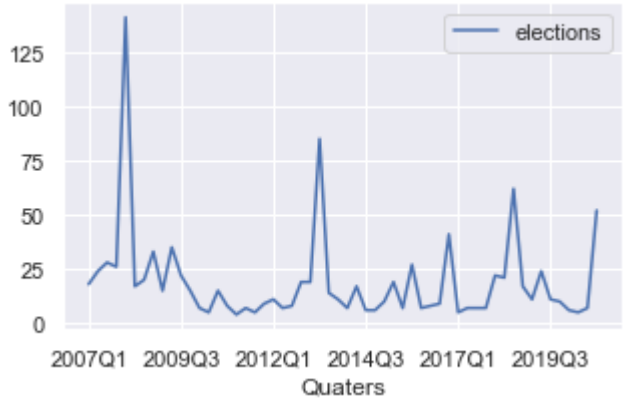

Politics
120
100
80
60
40
20
2007Q1
2009Q3
2012Q1
2014Q3
2017Q1
2019Q3
Quaters

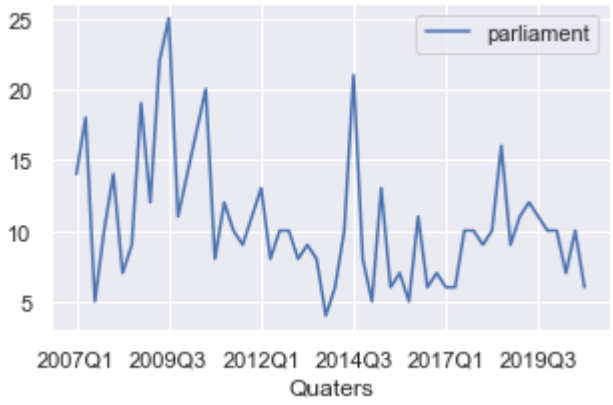

Newspapers
200
150
100
50
0
2007Q1
2009Q3
2012Q1
2014Q3
2017Q1
2019Q3
Quaters

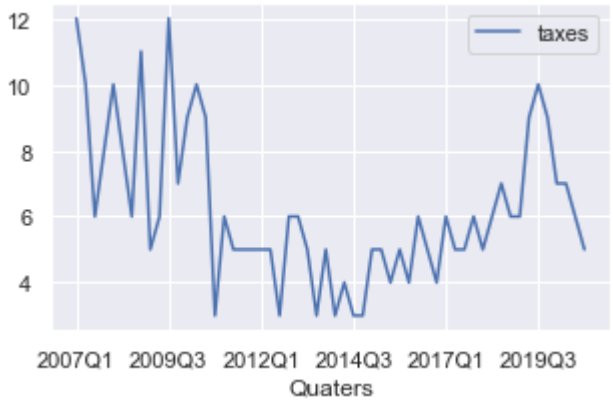

mehngai
1.0
0.8
0.6
0.4
0.2
0.0
2007Q1
2009Q3
2012Q1
2014Q3
2017Q1
2019Q3
Quaters

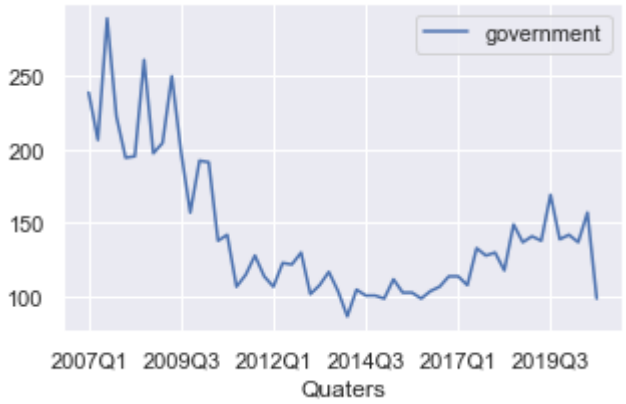

Real estate
70
60
50
40
30
20
10
2007Q1
2009Q3
2012Q1
2014Q3
2017Q1
2019Q3
Quaters

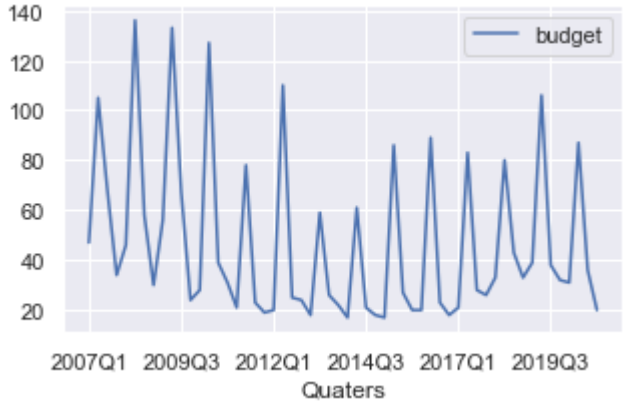

deficit
30
25
20
15
10
5
2007Q1
2009Q3
2012Q1
2014Q3
2017Q1
2019Q3
Quaters


elections
125
100
75
50
25
0
2007Q1
2009Q3
2012Q1
2014Q3
2017Q1
2019Q3
Quaters

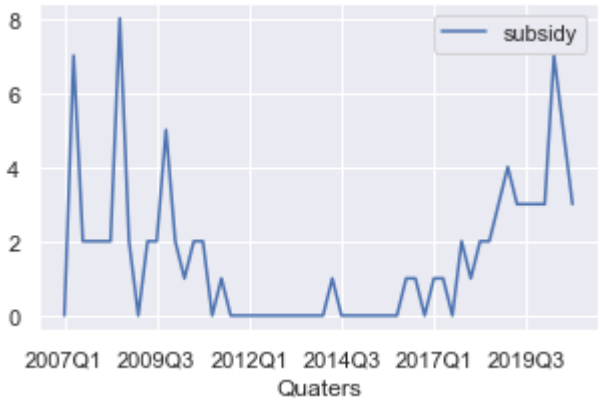

parliament
25
20
15
10
5
2007Q1
2009Q3
2012Q1
2014Q3
2017Q1
2019Q3
Quaters

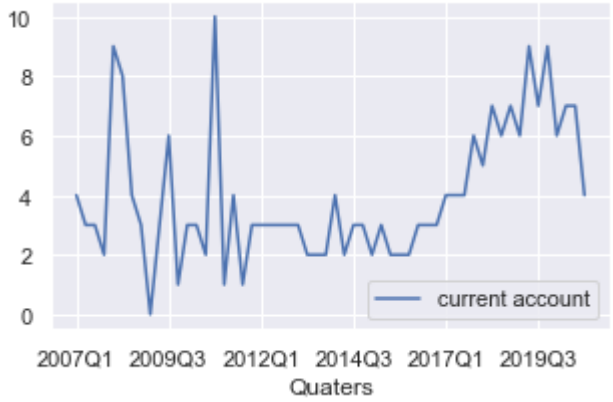

taxes
12
10
8
6
4
2007Q1
2009Q3
2012Q1
2014Q3
2017Q1
2019Q3
Quaters


government
250
200
150
100
2007Q1
2009Q3
2012Q1
2014Q3
2017Q1
2019Q3
Quaters

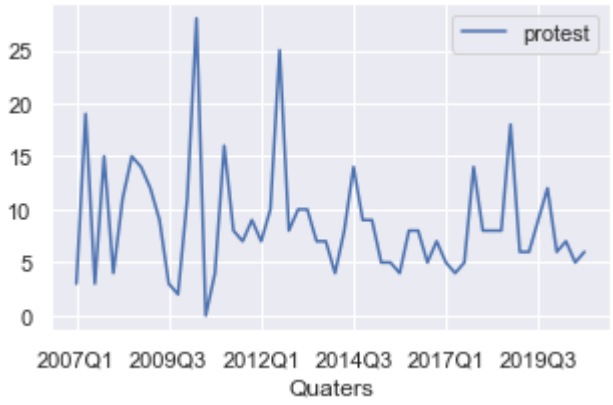

budget
140
120
100
80
60
40
20
2007Q1
2009Q3
2012Q1
2014Q3
2017Q1
2019Q3
Quaters

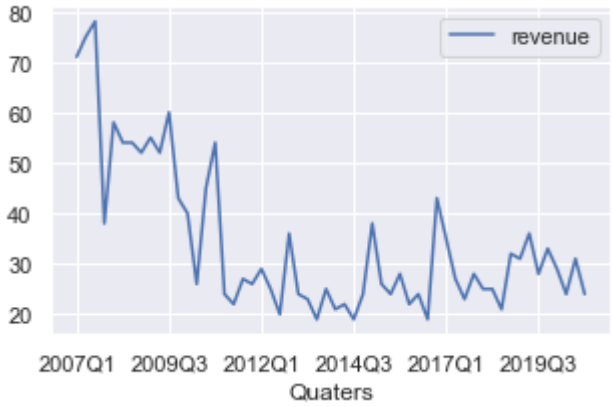

economic growth
30
25
20
15
10
5
2007Q1
2009Q3
2012Q1
2014Q3
2017Q1
2019Q3
Quaters


subsidy
8
6
4
2
0
2007Q1
2009Q3
2012Q1
2014Q3
2017Q1
2019Q3
Quaters


current account
10
8
6
4
2
0
2007Q1
2009Q3
2012Q1
2014Q3
2017Q1
2019Q3
Quaters


trade
200
150
100
50
2007Q1
2009Q3
2012Q1
2014Q3
2017Q1
2019Q3
Quaters





Training using Multi Layer Preceptron

Now training trends data on the available GDP data for prediction using the trained model.

In [15]:
```
# import related libraries
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
import pandas as pd
import numpy as np
import seaborn as sns
from matplotlib import pyplot as plt
import math
from pandas import DataFrame
from pandas import concat
from numpy import concatenate
```

```
Using TensorFlow backend.
```

In [100]:
```
df1 = df.pop('GDP')
df['GDP'] = df1
df.head(5)
```

Out[100]:

	Quaters	Economic crisis	Crisis	Recession	Financial crisis	Inflation	Unemployment	BISP	ehsaas program	USAID	...
0	2007Q1	6	65	0	0	22	9	0	0	15	...
1	2007Q2	0	68	0	0	81	28	0	0	21	...
2	2007Q3	0	81	0	0	55	19	2	0	12	...
3	2007Q4	4	62	0	0	75	15	0	0	9	...
4	2008Q1	0	115	8	0	56	18	0	0	13	...

5 rows × 60 columns

In [101]:
```
# normalize features
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(df.iloc[:,1:])
scaled
```

Out[101]:
```
array([[0.15384615, 0.25906736, 0.        , ..., 0.        , 0.        ,
        0.51138265],
       [0.        , 0.2746114 , 0.        , ..., 0.00539886, 0.00761773,
        0.53904981],
       [0.        , 0.34196891, 0.        , ..., 0.00663499, 0.02186097,
        0.67906471],
       ...,
       [0.05128205, 0.10880829, 0.15      , ..., 0.9873426 , 0.94793699,
        0.00385891],
       [0.02564103, 0.05181347, 0.08333333, ..., 1.        , 1.        ,
        0.        ],
       [0.        , 0.        , 0.06666667, ..., 0.64981478, 0.54123926,
        0.00781116]])
```

In [102]:
```
scal = pd.DataFrame(scaled, columns = df.iloc[:,1:].columns)
scal.head(5)
```

Out[102]:

	Economic crisis	Crisis	Recession	Financial crisis	Inflation	Unemployment	BISP	ehsaas program	USAID	
0	0.153846	0.259067	0.000000	0.0	0.104348	0.24	0.000000	0.0	0.405405	0.5
1	0.000000	0.274611	0.000000	0.0	0.617391	1.00	0.000000	0.0	0.567568	0.4
2	0.000000	0.341969	0.000000	0.0	0.391304	0.64	0.008696	0.0	0.324324	1.0
3	0.102564	0.243523	0.000000	0.0	0.565217	0.48	0.000000	0.0	0.243243	0.5
4	0.000000	0.518135	0.133333	0.0	0.400000	0.60	0.000000	0.0	0.351351	0.5

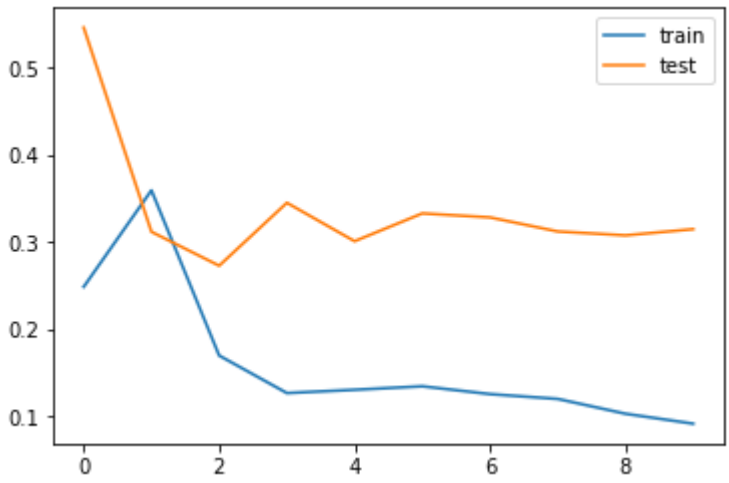
5 rows × 59 columns

In [103]:
```
values = scal.values
train = values[:-12, :]
test = values[-12:]
# split into input and outputs
train_X, train_y = train[:, :-1], train[:, -1]
test_X, test_y = test[:, :-1], test[:, -1]
# reshape input to be 3D [samples, timesteps, features]
train_X = train_X.reshape((train_X.shape[0], 1, train_X.shape[1]))
test_X = test_X.reshape((test_X.shape[0], 1, test_X.shape[1]))
print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)
```

```
(44, 1, 58) (44,) (12, 1, 58) (12,)
```

In [104]:
```
model = Sequential()
model.add(LSTM(50, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(300))
model.add(Dense(10))
model.add(Dense(1))
model.compile(loss='mae', optimizer='adam')
# fit network
history = model.fit(train_X, train_y, epochs=10, batch_size=6, validation_data=(test_
X, test_y), verbose=2, shuffle=False)
# plot history
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend()
plt.show()
```

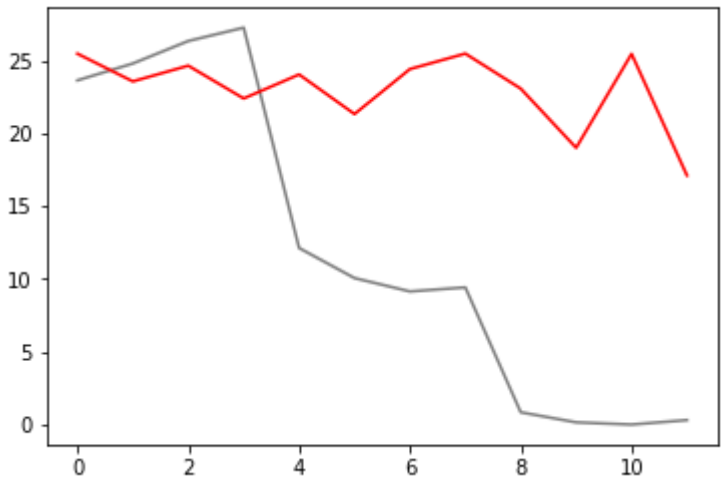
```
Train on 44 samples, validate on 12 samples
Epoch 1/10
 - 1s - loss: 0.2488 - val_loss: 0.5462
Epoch 2/10
 - 0s - loss: 0.3596 - val_loss: 0.3120
Epoch 3/10
 - 0s - loss: 0.1700 - val_loss: 0.2731
Epoch 4/10
 - 0s - loss: 0.1270 - val_loss: 0.3453
Epoch 5/10
 - 0s - loss: 0.1308 - val_loss: 0.3012
Epoch 6/10
 - 0s - loss: 0.1348 - val_loss: 0.3331
Epoch 7/10
 - 0s - loss: 0.1258 - val_loss: 0.3285
Epoch 8/10
 - 0s - loss: 0.1203 - val_loss: 0.3124
Epoch 9/10
 - 0s - loss: 0.1033 - val_loss: 0.3079
Epoch 10/10
 - 0s - loss: 0.0919 - val_loss: 0.3150
```



In [105]:
```
#make a prediction
yhat = model.predict(test_X)
test_X = test_X.reshape((test_X.shape[0], test_X.shape[2]))
# invert scaling for forecast
inv_yhat = concatenate((yhat, test_X[:, :]), axis=1)
inv_yhat = scaler.inverse_transform(inv_yhat)
inv_yhat = inv_yhat[:,0]
# invert scaling for actual
test_y = test_y.reshape((len(test_y), 1))
inv_y = concatenate((test_y, test_X[:, :]), axis=1)
inv_y = scaler.inverse_transform(inv_y)
inv_y = inv_y[:,0]
# calculate RMSE
rmse = math.sqrt(mean_squared_error(inv_y, inv_yhat))
print('Test RMSE: %.3f' % rmse)
```

```
Test RMSE: 14.634
```

In [106]:
```
#Ploting Fitted and Actual values
plt.plot(inv_y,color="gray")
plt.plot(inv_yhat,color="red")
plt.show()
```



In [35]:
```
from sklearn.neural_network import MLPRegressor
from sklearn.datasets import make_regression
X_train, X_test, y_train, y_test = df.iloc[:51,2:],df.iloc[52:,2:],df.iloc[:51,1],df.
iloc[52:,1]

network = MLPRegressor(hidden_layer_sizes=(300, 10),
                       solver = "adam",
                       activation = "relu",
                       learning_rate_init = 0.001,
                       tol=1e-4,
                       max_iter=700000,
                       early_stopping = False,
                       random_state=0)
network.fit(X_train, y_train)
network.predict(X_test)
```

Out[35]:
```
array([2.41256983, 2.58384271, 0.25254688, 1.69471641])
```

As Data size is not large so MLP is not performing wel lets try other technique

Applying All Machine Learning Algorithms from Template

In [32]:
```
## Changing Class balance and checking resuts
all_algo = ['decisiontree','randomforest','knn','lreg','svm', 'gboost','adaboost']
#all_algo = ['decisiontree']
colum = ['GDP']
X_train, X_test, y_train, y_test = df.iloc[:-4,2:],df.iloc[-4:,2:],df.iloc[:-4,1],df.
iloc[-4:,1]
for alg in all_algo:
    mae,mse,r2,r2adj,model = ml_algo(X_train, X_test, y_train, y_test,algo = alg,task
= 'Reg',n=3)
    print(alg)
    print("Mean Absolute Error: ",mae)
    print("Mean Square Error :",mse)
    print("Rbarsquare: ",r2)
    print("Rbarsquare Adjusted: ",r2adj)
    print()
```

```
decisiontree
Mean Absolute Error:  4.351181697155505
Mean Square Error : 19.38532637173128
Rbarsquare:  -3540.7720958125064
Rbarsquare Adjusted:  -3540.772095812506

randomforest
Mean Absolute Error:  3.442877146211465
Mean Square Error : 14.193509574291022
Rbarsquare:  -2592.2076245658805
Rbarsquare Adjusted:  -2592.2076245658805

knn
Mean Absolute Error:  2.9112758047130054
Mean Square Error : 10.256509385539346
Rbarsquare:  -1872.9028709423826
Rbarsquare Adjusted:  -1872.9028709423828

lreg
Mean Absolute Error:  5.403826103530117
Mean Square Error : 35.17371968946646
Rbarsquare:  -6425.370983557364
Rbarsquare Adjusted:  -6425.3709835573645

svm
Mean Absolute Error:  4.198142329562928
Mean Square Error : 17.84115903566346
Rbarsquare:  -3258.6469111612196
Rbarsquare Adjusted:  -3258.6469111612196

gboost
Mean Absolute Error:  3.628727740984974
Mean Square Error : 13.960304262951635
Rbarsquare:  -2549.6001363833975
Rbarsquare Adjusted:  -2549.6001363833975

adaboost
Mean Absolute Error:  3.5630710475611127
Mean Square Error : 14.065197121168572
Rbarsquare:  -2568.76445640354
Rbarsquare Adjusted:  -2568.76445640354
```

In [ ]: