MACRO ECONOMIC DATA

OBJECTIVE:

To understand and analyze the big picture of an economy by studying key indicators and trends, such as GDP growth, inflation, unemployment, and trade, to make informed decisions and policies that support economic well-being and growth.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
df = pd.read_csv(r'/content/macrodata.csv_.csv')
df
```

	Date	year	quarter	realgdp	realcons	realinv	realgovt	realdpi	
0	3/31/1959	1959	1	2710.349	1707.4	286.898	470.045	1886.9	2
1	6/30/1959	1959	2	2778.801	1733.7	310.859	481.301	1919.7	2
2	9/30/1959	1959	3	2775.488	1751.8	289.226	491.260	1916.4	2
3	12/31/1959	1959	4	2785.204	1753.7	299.356	484.052	1931.3	2
4	3/31/1960	1960	1	2847.699	1770.5	331.722	462.199	1955.5	2
...	...	...	...	...	...	...	...	...	
198	9/30/2008	2008	3	13324.600	9267.7	1990.693	991.551	9838.3	21
199	12/31/2008	2008	4	13141.920	9195.3	1857.661	1007.273	9920.4	21
200	3/31/2009	2009	1	12925.410	9209.2	1558.494	996.287	9926.4	21
201	6/30/2009	2009	2	12901.504	9189.0	1456.678	1023.528	10077.5	21
202	9/30/2009	2009	3	12990.341	9256.0	1486.398	1044.088	10040.6	21

203 rows × 15 columns

```
df.head()
```

	Date	year	quarter	realgdp	realcons	realinv	realgovt	realdpi	cpi
0	3/31/1959	1959	1	2710.349	1707.4	286.898	470.045	1886.9	28.98
1	6/30/1959	1959	2	2778.801	1733.7	310.859	481.301	1919.7	29.15
2	9/30/1959	1959	3	2775.488	1751.8	289.226	491.260	1916.4	29.35
3	12/31/1959	1959	4	2785.204	1753.7	299.356	484.052	1931.3	29.37
4	3/31/1960	1960	1	2847.699	1770.5	331.722	462.199	1955.5	29.54

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 203 entries, 0 to 202
Data columns (total 15 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Date      203 non-null    object
 1   year      203 non-null    int64
 2   quarter   203 non-null    int64
 3   realgdp   203 non-null    float64
 4   realcons  203 non-null    float64
 5   realinv   203 non-null    float64
 6   realgovt  203 non-null    float64
 7   realdpi   203 non-null    float64
 8   cpi       203 non-null    float64
 9   m1        203 non-null    float64
 10  tbilrate  203 non-null    float64
 11  unemp     203 non-null    float64
 12  pop       203 non-null    float64
 13  infl      203 non-null    float64
 14  realint   203 non-null    float64
dtypes: float64(12), int64(2), object(1)
memory usage: 23.9+ KB
```

```
df['Date'] = pd.to_datetime(df['Date'])
```

```
df=df.set_index(['Date'])
```

```
df
```

Date	year	quarter	realgdp	realcons	realinv	realgovt	realdpi	cpi	
1959-03-31	1959	1	2710.349	1707.4	286.898	470.045	1886.9	28.980	13
1959-06-30	1959	2	2778.801	1733.7	310.859	481.301	1919.7	29.150	14
1959-09-30	1959	3	2775.488	1751.8	289.226	491.260	1916.4	29.350	14
1959-12-31	1959	4	2785.204	1753.7	299.356	484.052	1931.3	29.370	14
1960-03-31	1960	1	2847.699	1770.5	331.722	462.199	1955.5	29.540	13
...	...	...	...	...	...	...	...	...	
2008-09-30	2008	3	13324.600	9267.7	1990.693	991.551	9838.3	216.889	147
2008-12-31	2008	4	13141.920	9195.3	1857.661	1007.273	9920.4	212.174	157

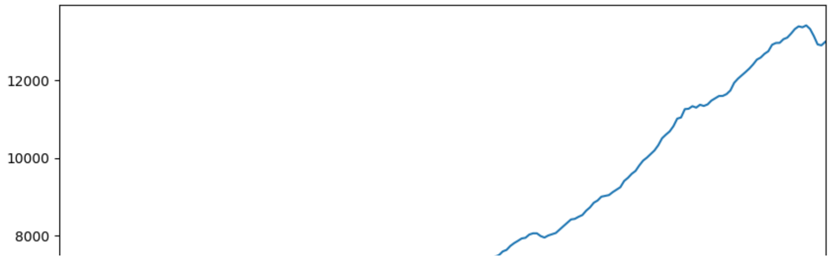
```
data = pd.DataFrame(df['realgdp'])
```

```
data
```

Date	realgdp
1959-03-31	2710.349
1959-06-30	2778.801
1959-09-30	2775.488
1959-12-31	2785.204
1960-03-31	2847.699
...	...
2008-09-30	13324.600
2008-12-31	13141.920
2009-03-31	12925.410
2009-06-30	12901.504
2009-09-30	12990.341

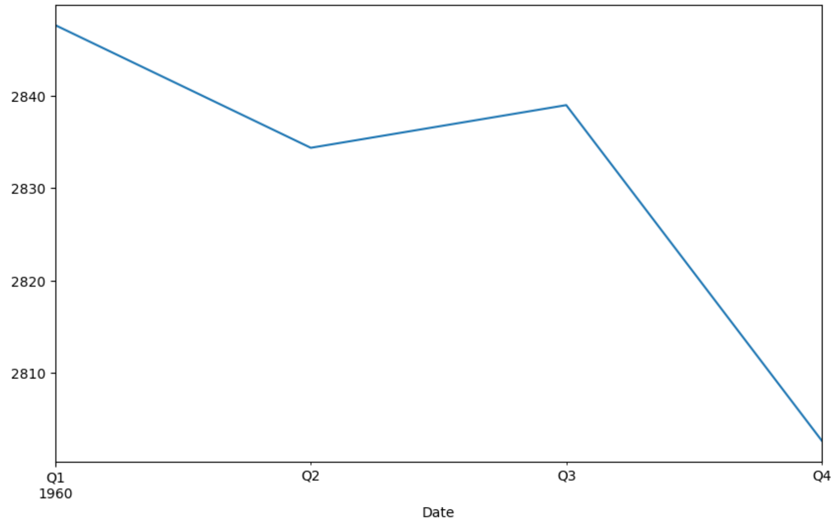
203 rows × 1 columns

```
data.realgdp.plot(figsize=(10,6));
```



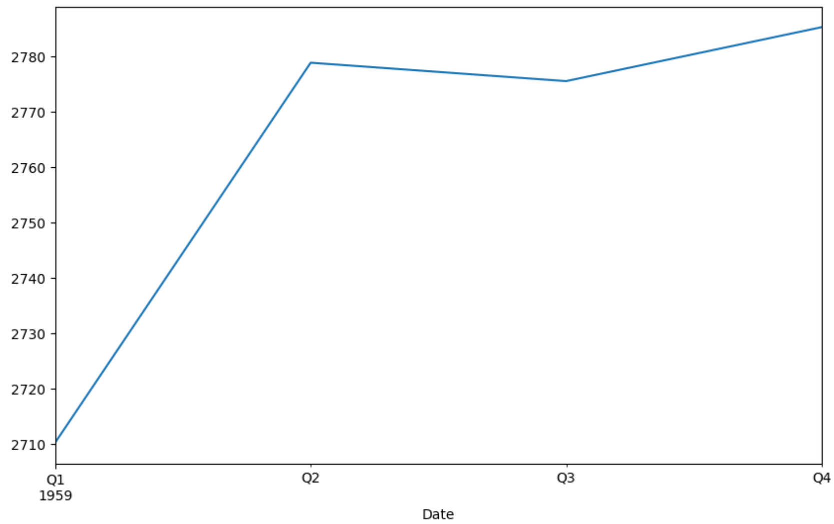

```
data['1960'].realgdp.plot(figsize=(10,6))
```

```
<ipython-input-12-3f8ea63c3e9e>:1: FutureWarning: Indexing a DataFrame with a date
  data['1960'].realgdp.plot(figsize=(10,6))
<Axes: xlabel='Date'>
```




```
data['1959-03-31':'1959-12-31'].realgdp.plot(figsize=(10,6))
```

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




```
data['1959'].mean()
```

```
<ipython-input-14-c65c702e4eab>:1: FutureWarning: Indexing a DataFrame with a datetimelike index using a single string to slice the
  data['1959'].mean()
realgdp    2762.4605
dtype: float64
```

```
data['1960'].mean()
```

```
<ipython-input-15-69875ea1518f>:1: FutureWarning: Indexing a DataFrame with a datetimelike index using a single string to slice the
  data['1960'].mean()
realgdp    2830.93175
dtype: float64
```

```
data['1970'].mean()
```

```
<ipython-input-16-bf22c9d7c288>:1: FutureWarning: Indexing a DataFrame with a datetimelike index using a single string to slice the
  data['1970'].mean()
realgdp    4269.9395
dtype: float64
```

```
data['d1']=df.realgdp.diff()
```

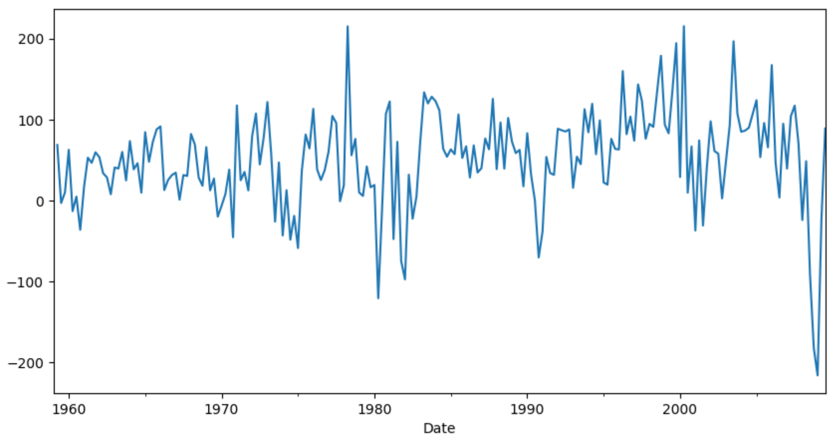
```
data
```

	realgdp	d1
Date		
1959-03-31	2710.349	NaN
1959-06-30	2778.801	68.452
1959-09-30	2775.488	-3.313
1959-12-31	2785.204	9.716
1960-03-31	2847.699	62.495
...	...	...
2008-09-30	13324.600	-90.666
2008-12-31	13141.920	-182.680
2009-03-31	12925.410	-216.510
2009-06-30	12901.504	-23.906
2009-09-30	12990.341	88.837

203 rows × 2 columns

```
data.d1.plot(figsize=(10,5))
```

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




```
data['1959'].d1.mean()
```

```
<ipython-input-20-3214ad96d64c>:1: FutureWarning: Indexing a DataFrame with a datetimelike index using a single string to slice the
  data['1959'].d1.mean()
24.95166666666667
```

```
data['1960'].d1.mean()
```

```
<ipython-input-21-c28e66afa8f8>:1: FutureWarning: Indexing a DataFrame with a datetimelike index using a single string to slice the
  data['1960'].d1.mean()
4.352999999999952
```

```
data['1970'].d1.mean()
```

```
<ipython-input-22-476cf48a175d>:1: FutureWarning: Indexing a DataFrame with a datetimelike index using a single string to slice the
  data['1970'].d1.mean()
-1.6560000000001764
```

```
round(data['1959'].d1.mean())
```

```
<ipython-input-23-719979ca2937>:1: FutureWarning: Indexing a DataFrame with a datetimelike index using a single string to slice the
  round(data['1959'].d1.mean())
25
```

```
round(data['1960'].d1.mean())
```

```
<ipython-input-24-0f30aceace94>:1: FutureWarning: Indexing a DataFrame with a datetimelike index using a single string to slice the
  round(data['1960'].d1.mean())
4
```

```
round(data['1970'].d1.mean())
```

```
<ipython-input-25-a07660c039a9>:1: FutureWarning: Indexing a DataFrame with a datetimelike index using a single string to slice the
  round(data['1970'].d1.mean())
-2
```

```
data['d2']=data.realgdp.diff(periods=2)
```

```
data
```

	realgdp	d1	d2
Date			
1959-03-31	2710.349	NaN	NaN
1959-06-30	2778.801	68.452	NaN
1959-09-30	2775.488	-3.313	65.139
1959-12-31	2785.204	9.716	6.403
1960-03-31	2847.699	62.495	72.211
...	...	...	...
2008-09-30	13324.600	-90.666	-42.265
2008-12-31	13141.920	-182.680	-273.346
2009-03-31	12925.410	-216.510	-399.190
2009-06-30	12901.504	-23.906	-240.416
2009-09-30	12990.341	88.837	64.931

203 rows × 3 columns

```
from pandas.core.arrays import period
data['dd1']=data.d1.diff()
```

```
data
```

Date	realgdp	d1	d2	dd1
1959-03-31	2710.349	NaN	NaN	NaN
1959-06-30	2778.801	68.452	NaN	NaN
1959-09-30	2775.488	-3.313	65.139	-71.765
1959-12-31	2785.204	9.716	6.403	13.029
1960-03-31	2847.699	62.495	72.211	52.779
...	...	...	...	...
2008-09-30	13324.600	-90.666	-42.265	-139.067
2008-12-31	13141.920	-182.680	-273.346	-92.014
2009-03-31	12925.410	-216.510	-399.190	-33.830

```
data['1959'].dd1.mean()
```

```
<ipython-input-30-ad3b13f5a78e>:1: FutureWarning: Indexing a DataFrame with a datetimelike index using a single string to slice the
  data['1959'].dd1.mean()
-29.36799999999971
```

```
data['1960'].dd1.mean()
```

```
<ipython-input-31-4da5448af4d0>:1: FutureWarning: Indexing a DataFrame with a datetimelike index using a single string to slice the
  data['1960'].dd1.mean()
-11.530500000000075
```

```
from statsmodels.tsa.seasonal import seasonal_decompose
```

```
data.index
```

```
DatetimeIndex(['1959-03-31', '1959-06-30', '1959-09-30', '1959-12-31',
               '1960-03-31', '1960-06-30', '1960-09-30', '1960-12-31',
               '1961-03-31', '1961-06-30',
               ...
               '2007-06-30', '2007-09-30', '2007-12-31', '2008-03-31',
               '2008-06-30', '2008-09-30', '2008-12-31', '2009-03-31',
               '2009-06-30', '2009-09-30'],
              dtype='datetime64[ns]', name='Date', length=203, freq=None)
```

```
data.realgdp.bfill(inplace=True)
```

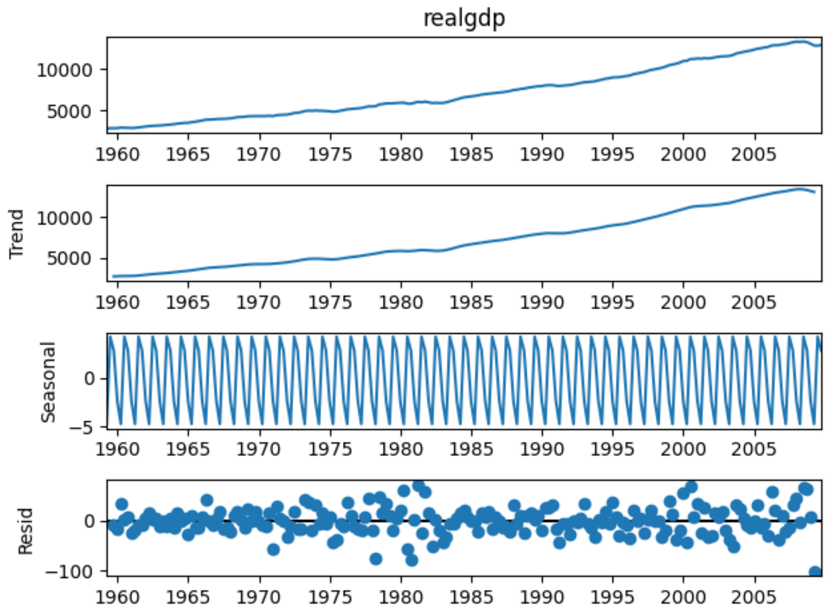
```
data
```

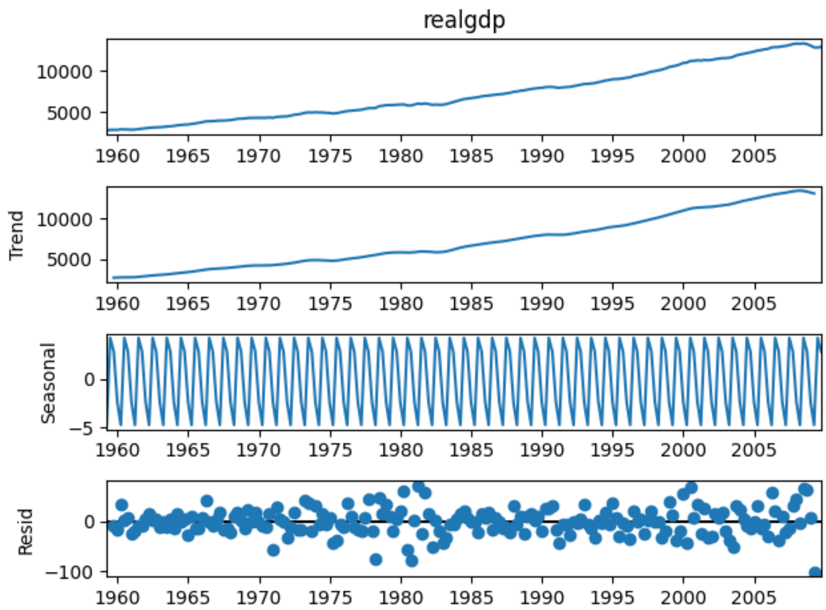
Date	realgdp	d1	d2	dd1
1959-03-31	2710.349	NaN	NaN	NaN
1959-06-30	2778.801	68.452	NaN	NaN
1959-09-30	2775.488	-3.313	65.139	-71.765
1959-12-31	2785.204	9.716	6.403	13.029
1960-03-31	2847.699	62.495	72.211	52.779
...	...	...	...	...
2008-09-30	13324.600	-90.666	-42.265	-139.067
2008-12-31	13141.920	-182.680	-273.346	-92.014
2009-03-31	12925.410	-216.510	-399.190	-33.830
2009-06-30	12901.504	-23.906	-240.416	192.604
2009-09-30	12990.341	88.837	64.931	112.743

203 rows × 4 columns

```
x = seasonal_decompose(data.realgdp)
```

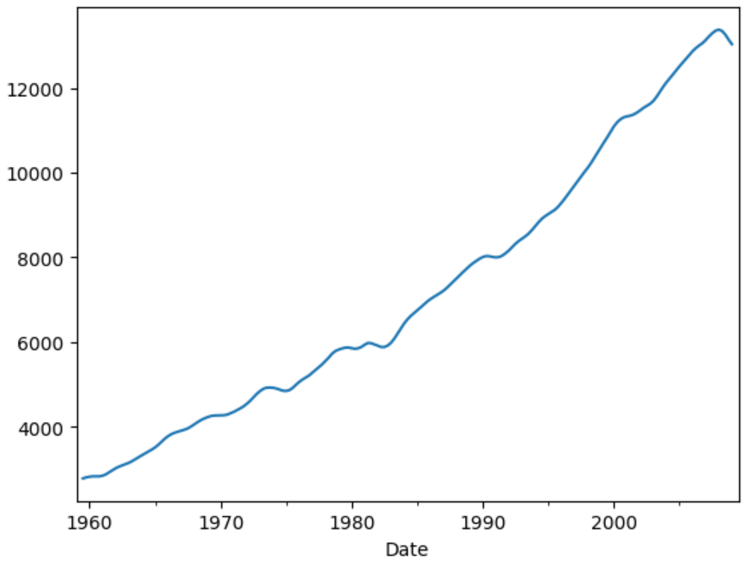
```
x.plot()
```





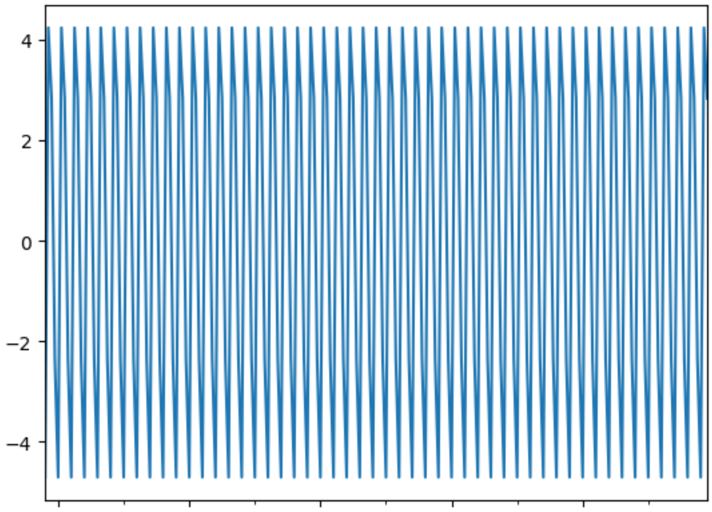

```
x.trend.plot()
```

<Axes: xlabel='Date'>



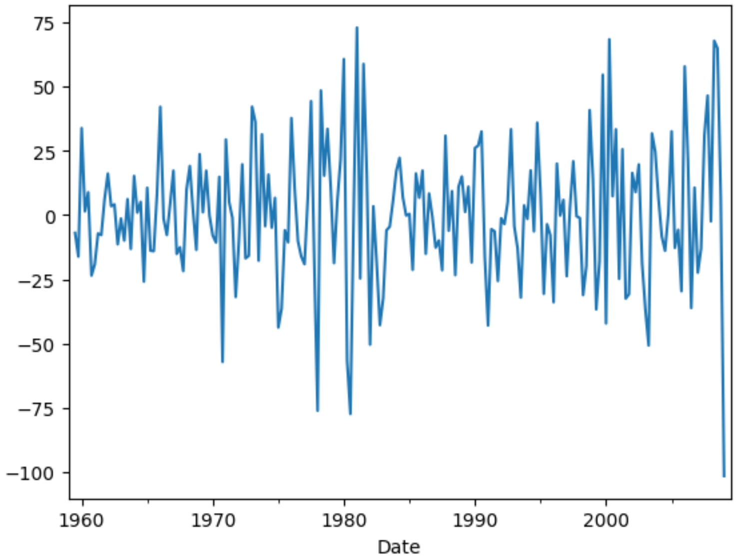

```
x.seasonal.plot()
```

<Axes: xlabel='Date'>




```
x.resid.plot()
```

<Axes: xlabel='Date'>




```
data
```

Date	realgdp	d1	d2	dd1
1959-03-31	2710.349	NaN	NaN	NaN
1959-06-30	2778.801	68.452	NaN	NaN
1959-09-30	2775.488	-3.313	65.139	-71.765
1959-12-31	2785.204	9.716	6.403	13.029
1960-03-31	2847.699	62.495	72.211	52.779
...	...	...	...	...
2008-09-30	13324.600	-90.666	-42.265	-139.067
2008-12-31	13141.920	-182.680	-273.346	-92.014
2009-03-31	12925.410	-216.510	-399.190	-33.830
2009-06-30	12901.504	-23.906	-240.416	192.604
2009-09-30	12990.341	88.837	64.931	112.743

203 rows × 4 columns

```
data = data.copy()
```

```
data.index
```

```
DatetimeIndex(['1959-03-31', '1959-06-30', '1959-09-30', '1959-12-31',
               '1960-03-31', '1960-06-30', '1960-09-30', '1960-12-31',
               '1961-03-31', '1961-06-30',
               ...
               '2007-06-30', '2007-09-30', '2007-12-31', '2008-03-31',
               '2008-06-30', '2008-09-30', '2008-12-31', '2009-03-31',
               '2009-06-30', '2009-09-30'],
              dtype='datetime64[ns]', name='Date', length=203, freq=None)
```

```
data=pd.DataFrame(data['realgdp'])
```

```
data.index
```

```
DatetimeIndex(['1959-03-31', '1959-06-30', '1959-09-30', '1959-12-31',
               '1960-03-31', '1960-06-30', '1960-09-30', '1960-12-31',
               '1961-03-31', '1961-06-30',
               ...
               '2007-06-30', '2007-09-30', '2007-12-31', '2008-03-31',
               '2008-06-30', '2008-09-30', '2008-12-31', '2009-03-31',
               '2009-06-30', '2009-09-30'],
              dtype='datetime64[ns]', name='Date', length=203, freq=None)
```

```
data = data['1959':'2009']
```

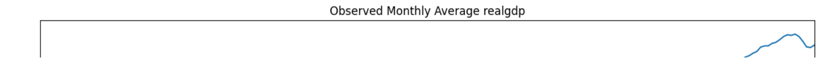
```
data.head()
```

	realgdp
Date	
1959-03-31	2710.349
1959-06-30	2778.801
1959-09-30	2775.488
1959-12-31	2785.204
1960-03-31	2847.699

```
Total = data.isnull().sum().sort_values(ascending = False)
Percent = (data.isnull().sum()*100/data.isnull().count()).sort_values(ascending = False)
missing_data = pd.concat([Total,Percent],axis = 1,keys=['Total','Percentage of missing value'])
missing_data
```

	Total	Percentage of missing value
realgdp	0	0.0

```
data.plot(figsize = (15,6),legend = None)
plt.xlabel('Date',fontsize = 14)
plt.ylabel('realgdp',fontsize =14)
plt.title('Observed Monthly Average realgdp')
plt.show()
```




```
from statsmodels.tsa.stattools import adfuller
```

```
def adf_test(timeseries):
  print('Result of Dickey_fuller Test: ')
  result=adfuller(timeseries,autolag="AIC")
  result=pd.Series(result[0:4], index=[" Test Statistic",'p-value','No. of lags Used','Number of Observations Used'])
  print(result)

  if result[1]<=0.05:
        print("Strong evidence against the null hypothesis")
        print("Reject the null hypothesis")
        print("Data has no unit root and is stationary")

  else:
        print("Strong evidence against the null hypothesis")
        print("Reject the null hypothesis")
        print("Data has no unit root and is stationary")


adf_test(data)
```

```
Result of Dickey_fuller Test:
 Test Statistic                   1.750463
p-value                          0.998246
No. of lags Used                12.000000
Number of Observations Used    190.000000
dtype: float64
Strong evidence against the null hypothesis
Reject the null hypothesis
Data has no unit root and is stationary
```

```
y = data['realgdp']
y
```

```
Date
1959-03-31     2710.349
1959-06-30     2778.801
1959-09-30     2775.488
1959-12-31     2785.204
1960-03-31     2847.699
                ...
2008-09-30    13324.600
2008-12-31    13141.920
2009-03-31    12925.410
2009-06-30    12901.504
2009-09-30    12990.341
Name: realgdp, Length: 203, dtype: float64
```

```
train = y[:'2005']
test = y['2007':]
```

```
pip install pmdarima
```

```
Collecting pmdarima
  Downloading pmdarima-2.0.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.manylinux_2_28_x86_64.whl (1.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.8/1.8 MB 9.1 MB/s eta 0:00:00
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (1.3.1)
Requirement already satisfied: Cython!=0.29.18,!=0.29.31,>=0.29 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (0.29.3
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (1.22.4)
Requirement already satisfied: pandas>=0.19 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (1.5.3)
Requirement already satisfied: scikit-learn>=0.22 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (1.2.2)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (1.10.1)
Requirement already satisfied: statsmodels>=0.13.2 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (0.13.5)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (1.26.16)
Requirement already satisfied: setuptools!=50.0.0,>=38.6.0 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (67.7.2)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.19->pmdarima) (2.8
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.19->pmdarima) (2022.7.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.22->pmdarima)
Requirement already satisfied: patsy>=0.5.2 in /usr/local/lib/python3.10/dist-packages (from statsmodels>=0.13.2->pmdarima) (0.5.3)
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.10/dist-packages (from statsmodels>=0.13.2->pmdarima) (23
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from patsy>=0.5.2->statsmodels>=0.13.2->pmdarima) (1
Installing collected packages: pmdarima
Successfully installed pmdarima-2.0.3
```

```
from pmdarima.arima import auto_arima
```

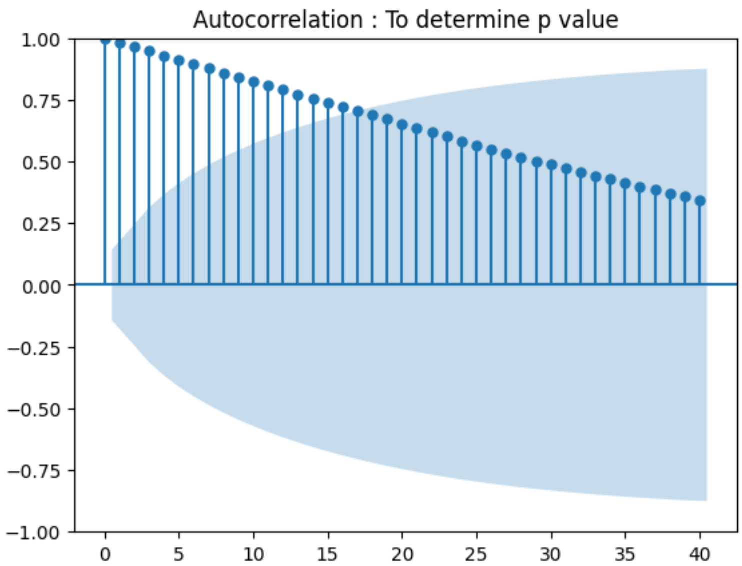
```
arima_model=auto_arima(train,Seasonal=True,stepwise=False,trace=1,random_state=10)
```

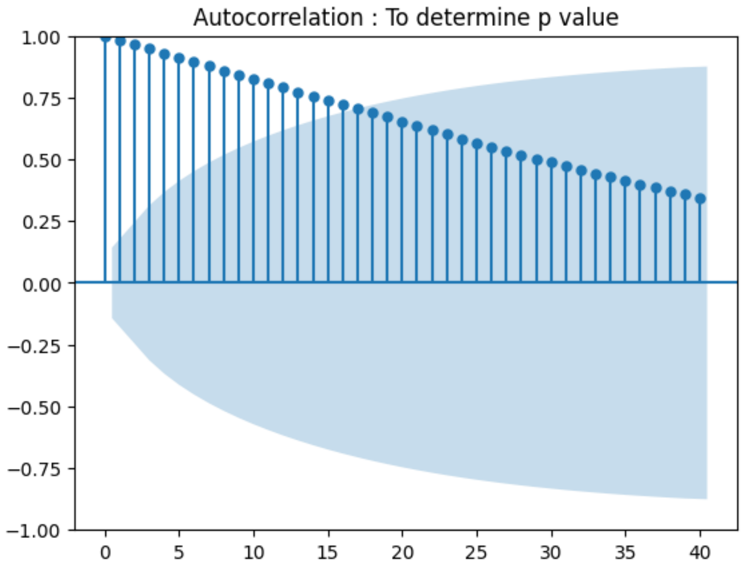
```
 ARIMA(0,2,0)(0,0,0)[1]             : AIC=2064.739, Time=0.07 sec
 ARIMA(0,2,1)(0,0,0)[1]             : AIC=1996.294, Time=0.06 sec
 ARIMA(0,2,2)(0,0,0)[1]             : AIC=1994.614, Time=0.14 sec
 ARIMA(0,2,3)(0,0,0)[1]             : AIC=1986.538, Time=0.19 sec
 ARIMA(0,2,4)(0,0,0)[1]             : AIC=1988.253, Time=0.31 sec
 ARIMA(0,2,5)(0,0,0)[1]             : AIC=1987.307, Time=0.37 sec
 ARIMA(1,2,0)(0,0,0)[1]             : AIC=2014.148, Time=0.04 sec
 ARIMA(1,2,1)(0,0,0)[1]             : AIC=1990.986, Time=0.13 sec
 ARIMA(1,2,2)(0,0,0)[1]             : AIC=1999.441, Time=0.22 sec
 ARIMA(1,2,3)(0,0,0)[1]             : AIC=1987.740, Time=0.62 sec
 ARIMA(1,2,4)(0,0,0)[1]             : AIC=1990.060, Time=0.85 sec
 ARIMA(2,2,0)(0,0,0)[1]             : AIC=2011.576, Time=0.12 sec
 ARIMA(2,2,1)(0,0,0)[1]             : AIC=1984.973, Time=0.43 sec
 ARIMA(2,2,2)(0,0,0)[1]             : AIC=1986.046, Time=0.71 sec
 ARIMA(2,2,3)(0,0,0)[1]             : AIC=inf, Time=1.54 sec
 ARIMA(3,2,0)(0,0,0)[1]             : AIC=2004.129, Time=0.13 sec
 ARIMA(3,2,1)(0,0,0)[1]             : AIC=1986.636, Time=0.64 sec
 ARIMA(3,2,2)(0,0,0)[1]             : AIC=1987.841, Time=0.56 sec
 ARIMA(4,2,0)(0,0,0)[1]             : AIC=2005.193, Time=0.08 sec
 ARIMA(4,2,1)(0,0,0)[1]             : AIC=1987.898, Time=0.37 sec
 ARIMA(5,2,0)(0,0,0)[1]             : AIC=2003.722, Time=0.15 sec

Best model:  ARIMA(2,2,1)(0,0,0)[1]
Total fit time: 7.816 seconds
```

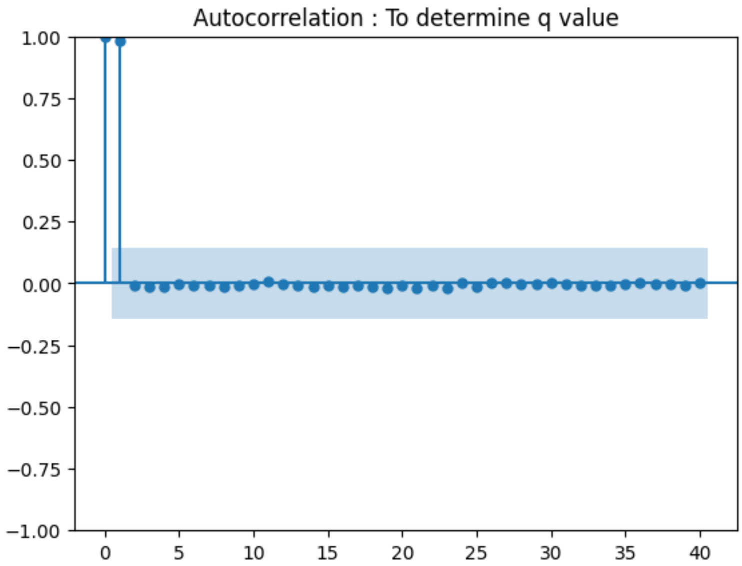
```
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
```

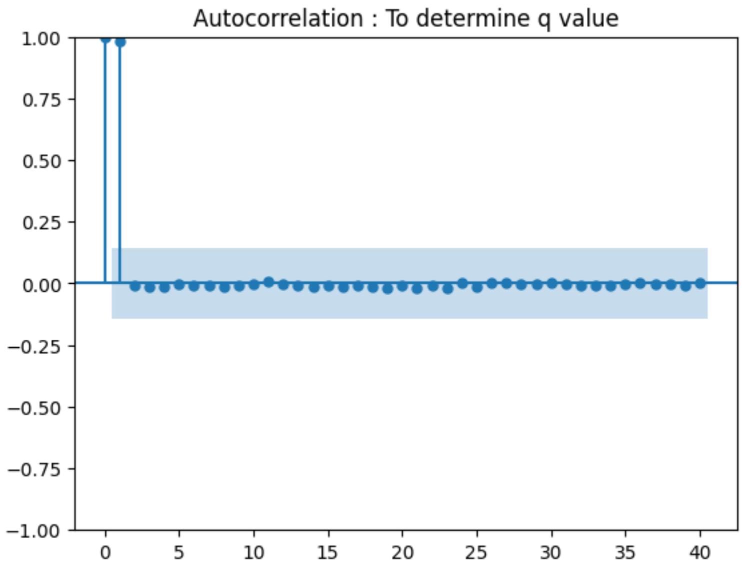
```
title='Autocorrelation : To determine p value'
lags=40
plot_acf(train,title=title,lags=lags)
```






```
title='Autocorrelation : To determine q value'
lags=40
plot_pacf(train,title=title,lags=lags,method='ywm')
```





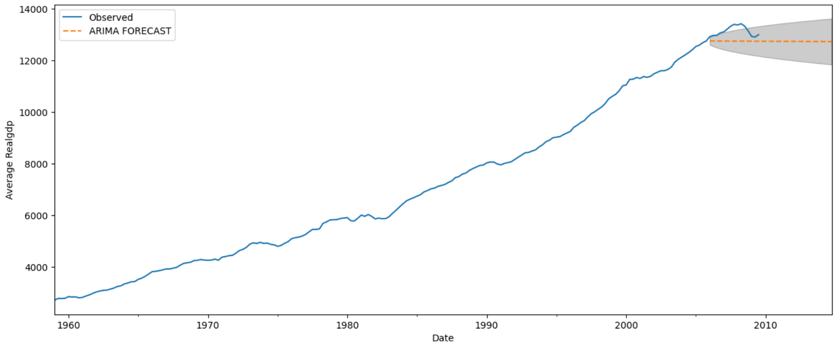

```
from statsmodels.tsa.arima.model import ARIMA,ARIMAResults
```

```
model=ARIMA(train,order=(1,0,0))
results=model.fit()
results.summary()
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:471: Val
  self._init_dates(dates, freq)
```

```
model=ARIMA(train,order=(1,0,0))
results=model.fit()
results.summary()
pred=results.get_forecast(steps=36)
ax1=y['1959':].plot(label='Observed')
pred.predicted_mean.plot(ax=ax1,label="ARIMA FORECAST",figsize=(15,6),linestyle='dashed')
pred_ci=pred.conf_int()
ax1.fill_between(pred_ci.index,pred_ci.iloc[:,0],pred_ci.iloc[:,1],color="k",alpha=0.2)
ax1.set_xlabel('Date')
ax1.set_ylabel('Average Realgdp')
plt.legend(loc= "upper left")
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:471: Val
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:471: Val
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:471: Val
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/sarimax.py:966:
  warn('Non-stationary starting autoregressive parameters'
```




```
from statsmodels.tsa.statespace.sarimax import SARIMAX
```

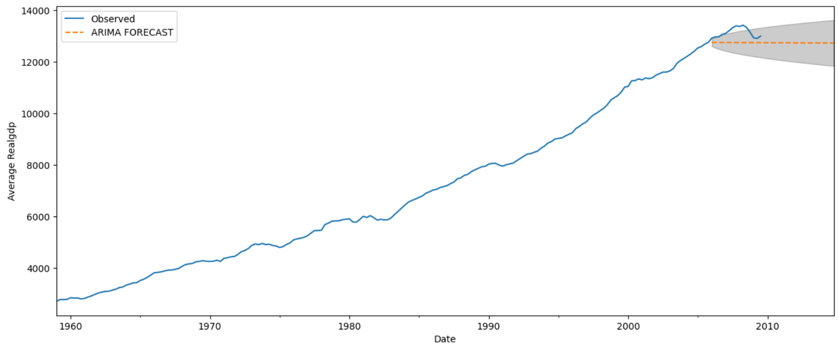
```
model = SARIMAX(train ,order=(1,1,2),seasonal_order=(1,0,1,12),enforece_stationarity=False,enforece_invertibility=False)
fitted_model=model.fit(maxiter=200)
print(fitted_model.summary())
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was provided,
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was provided,
  self._init_dates(dates, freq)
                                      SARIMAX Results
==========================================================================================
Dep. Variable:                              realgdp   No. Observations:                  188
Model:             SARIMAX(1, 1, 2)x(1, 0, [1], 12)   Log Likelihood                -998.489
Date:                              Tue, 25 Jul 2023   AIC                           2008.977
Time:                                      10:20:05   BIC                           2028.364
Sample:                                  03-31-1959   HQIC                          2016.833
                                       - 12-31-2005
Covariance Type:                                opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          1.0000      0.000   2619.704      0.000       0.999       1.001
ma.L1         -0.7721      0.068    -11.274      0.000      -0.906      -0.638
ma.L2         -0.1459      0.069     -2.118      0.034      -0.281      -0.011
ar.S.L12       0.8960      0.069     12.987      0.000       0.761       1.031
ma.S.L12      -0.9937      0.103     -9.664      0.000      -1.195      -0.792
sigma2      2434.1816   5.13e-05   4.75e+07      0.000    2434.182    2434.182
===================================================================================
Ljung-Box (L1) (Q):                   0.09   Jarque-Bera (JB):                 9.97
Prob(Q):                              0.76   Prob(JB):                         0.01
Heteroskedasticity (H):               1.98   Skew:                            -0.12
```

```
Prob(H) (two-sided):                  0.01   Kurtosis:                         4.10
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number 3.34e+22. Standard errors may be unstable.
```

```
pred=results.get_forecast(steps=36)
ax1=y['1950':].plot(label='Observed')
pred.predicted_mean.plot(ax=ax1,label="ARIMA FORECAST",figsize=(15,6),linestyle='dashed')
pred_ci=pred.conf_int()
ax1.fill_between(pred_ci.index,pred_ci.iloc[:,0],pred_ci.iloc[:,1],color="k",alpha=0.2)
ax1.set_xlabel('Date')
ax1.set_ylabel('Average Realgdp')
plt.legend(loc= "upper left")
plt.show()
```

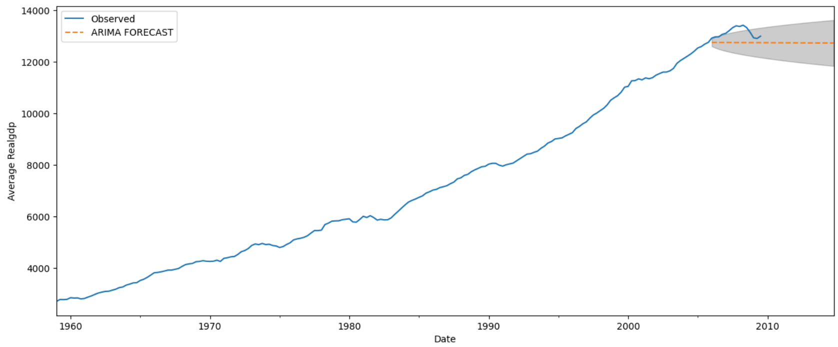



```
model2 = SARIMAX(train ,order=(0,0,1),seasonal_order=(1,0,1,12),enforece_stationarity=False,enforece_invertibility=False)
fitted_model=model2.fit(maxiter=200)
print(fitted_model.summary())
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was provided,
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was provided,
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/sarimax.py:978: UserWarning: Non-invertible starting MA paramete
  warn('Non-invertible starting MA parameters found.'
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/sarimax.py:997: UserWarning: Non-stationary starting seasonal au
  warn('Non-stationary starting seasonal autoregressive'
                                     SARIMAX Results
==========================================================================================
Dep. Variable:                            realgdp   No. Observations:                  188
Model:             SARIMAX(0, 0, 1)x(1, 0, 1, 12)   Log Likelihood               -1400.619
Date:                            Tue, 25 Jul 2023   AIC                           2809.237
Time:                                    10:20:06   BIC                           2822.183
Sample:                                03-31-1959   HQIC                          2814.482
                                     - 12-31-2005
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ma.L1          1.0000      4.089      0.245      0.807      -7.013       9.013
ar.S.L12       0.9905      0.006    160.129      0.000       0.978       1.003
ma.S.L12       0.9998      4.123      0.242      0.808      -7.081       9.081
sigma2      1.048e+05   3.73e-05   2.81e+09      0.000    1.05e+05    1.05e+05
===================================================================================
Ljung-Box (L1) (Q):                  44.04   Jarque-Bera (JB):                 2.34
Prob(Q):                              0.00   Prob(JB):                         0.31
Heteroskedasticity (H):               5.97   Skew:                             0.26
Prob(H) (two-sided):                  0.00   Kurtosis:                         3.19
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number 1.79e+26. Standard errors may be unstable.
```

```
pred=results.get_forecast(steps=36)
ax1=y['1950':].plot(label='Observed')
pred.predicted_mean.plot(ax=ax1,label="ARIMA FORECAST",figsize=(15,6),linestyle='dashed')
pred_ci=pred.conf_int()
ax1.fill_between(pred_ci.index,pred_ci.iloc[:,0],pred_ci.iloc[:,1],color="k",alpha=0.2)
ax1.set_xlabel('Date')
ax1.set_ylabel('Average Realgdp')
plt.legend(loc= "upper left")
plt.show()
```




```
y_forecasted_SARIMAX = pred.predicted_mean
y_truth = test
mse_SARIMAX = ((y_forecasted_SARIMAX - y_truth)**2).mean()
print('The Mean Squared Error of SARIMAX Forecast is {}'.format(round(mse_SARIMAX,2)))
print('The Root Mean squared Error of SARIMAX Forecast is {} '.format(round(np.sqrt(mse_SARIMAX),2)))
```

```
The Mean Squared Error of SARIMAX Forecast is 232698.8
The Root Mean squared Error of SARIMAX Forecast is 482.39
```

```
model3 = SARIMAX(train ,order=(0,0,2),seasonal_order=(1,0,1,12),enforece_stationarity=False,enforece_invertibility=False)
fitted_model=model3.fit(maxiter=200)
print(fitted_model.summary())
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was provided,
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency information was provided,
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/sarimax.py:978: UserWarning: Non-invertible starting MA paramete
  warn('Non-invertible starting MA parameters found.'
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/sarimax.py:997: UserWarning: Non-stationary starting seasonal au
  warn('Non-stationary starting seasonal autoregressive'
                                      SARIMAX Results
============================================================================================
Dep. Variable:                             realgdp   No. Observations:                  188
Model:             SARIMAX(0, 0, 2)x(1, 0, [1], 12)   Log Likelihood               -1298.603
Date:                              Tue, 25 Jul 2023   AIC                           2607.206
Time:                                      10:20:08   BIC                           2623.388
Sample:                                  03-31-1959   HQIC                          2613.762
                                       - 12-31-2005
Covariance Type:                                opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ma.L1          1.0001      0.042     24.087      0.000       0.919       1.081
ma.L2          0.9989      0.077     12.916      0.000       0.847       1.150
ar.S.L12       0.9957      0.003    313.629      0.000       0.989       1.002
ma.S.L12       0.4844      0.078      6.191      0.000       0.331       0.638
sigma2      4.339e+04   3665.420     11.839      0.000    3.62e+04    5.06e+04
===================================================================================
Ljung-Box (L1) (Q):                  43.36   Jarque-Bera (JB):                12.28
Prob(Q):                              0.00   Prob(JB):                         0.00
Heteroskedasticity (H):               3.94   Skew:                             0.02
Prob(H) (two-sided):                  0.00   Kurtosis:                         4.25
===================================================================================
```

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

```
y_forecasted_SARIMAX = pred.predicted_mean
y_truth = test
mse_SARIMAX = ((y_forecasted_SARIMAX - y_truth)**2).mean()
print('The Mean Squared Error of SARIMAX Forecast is {}'.format(round(mse_SARIMAX,2)))
print('The Root Mean squared Error of SARIMAX Forecast is {} '.format(round(np.sqrt(mse_SARIMAX),2)))
```

```
The Mean Squared Error of SARIMAX Forecast is 232698.8
The Root Mean squared Error of SARIMAX Forecast is 482.39
```

```
df1= pd.DataFrame(data.realgdp)
```

```
train = df1.iloc[:203]
test = df1.iloc[203:]
```

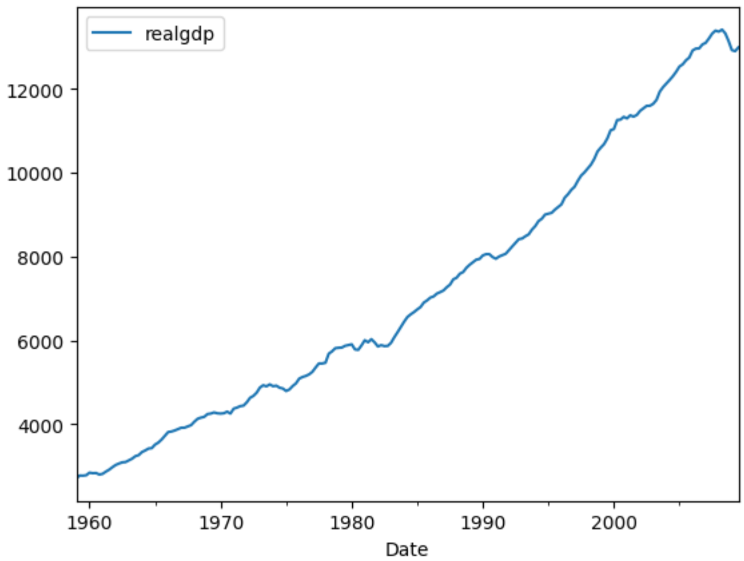
```
df1
```

	realgdp
Date	
1959-03-31	2710.349
1959-06-30	2778.801
1959-09-30	2775.488
1959-12-31	2785.204
1960-03-31	2847.699
...	...
2008-09-30	13324.600
2008-12-31	13141.920
2009-03-31	12925.410
2009-06-30	12901.504
2009-09-30	12990.341

203 rows × 1 columns

```
df1.plot()
```

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




```
df1.size
```

```
203
```

```
df1.size-5
```

```
198
```

```
train = df1.iloc[:194]
test = df1.iloc[194:]
```

```
train
```

Date	realgdp
1959-03-31	2710.349
1959-06-30	2778.801
1959-09-30	2775.488
1959-12-31	2785.204
1960-03-31	2847.699
...	...
2006-06-30	12962.462
2006-09-30	12965.916
2006-12-31	13060.679
2007-03-31	13099.901
2007-06-30	13203.977

194 rows × 1 columns

```
test.size
```

```
9
```

```
test
```

Date	realgdp
2007-09-30	13321.109
2007-12-31	13391.249
2008-03-31	13366.865
2008-06-30	13415.266
2008-09-30	13324.600
2008-12-31	13141.920
2009-03-31	12925.410
2009-06-30	12901.504
2009-09-30	12990.341

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
```

```
scaler.fit(train)
```

```
▾ MinMaxScaler
MinMaxScaler()
```

```
scaled_train =  scaler.transform(train)
```

```
scaled_train[:5]
```

```
array([[0.        ],
       [0.0065232 ],
       [0.00620748],
       [0.00713338],
       [0.0130889 ]])
```

```
scaled_test = scaler.transform(test)
```

```
scaled_test.max(),scaled_test.min()
```

```
(1.0201349809617797, 0.9711755552988919)
```

```
from keras.preprocessing.sequence import TimeseriesGenerator
```

```
scaled_train[:5]
```

```
array([[0.        ],
       [0.0065232 ],
       [0.00620748],
       [0.00713338],
       [0.0130889 ]])
```

```
n_input = 2
n_features = 1
```

```
generator= TimeseriesGenerator(scaled_train,scaled_train,length=n_input,batch_size=1)
```

```
generator[0]
```

```
(array([[[0.       ],
         [0.0065232]]]),
 array([[0.00620748]]))
```

```
x,y = generator[0]
```

```
x
```

```
array([[[0.       ],
        [0.0065232]]])
```

```
y
```

```
array([[0.00620748]])
```

```
len(scaled_train)
```

```
194
```

```
len(generator)
```

```
192
```

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
import tensorflow as tf
```

```
n_input = 5
n_features = 1
train_generator = TimeseriesGenerator(scaled_train,scaled_train,length=n_input,batch_size=1)
```

```
x.shape
```

```
(1, 2, 1)
```

```
model = tf.keras.models.Sequential([
    tf.keras.layers.LSTM(100, input_shape = (n_input,n_features), return_sequences= True),
    tf.keras.layers.LSTM(50, return_sequences = True),
    tf.keras.layers.LSTM(10),
    tf.keras.layers.Dense(64, activation = 'relu'),
    tf.keras.layers.Dense(32, activation = 'relu'),
    tf.keras.layers.Dense(1)
])

model.compile(optimizer = 'adam',loss = 'mse')
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 5, 100)            40800
```

```
 lstm_1 (LSTM)              (None, 5, 50)             30200

 lstm_2 (LSTM)              (None, 10)                2440

 dense (Dense)              (None, 64)                704

 dense_1 (Dense)            (None, 32)                2080

 dense_2 (Dense)            (None, 1)                 33

=================================================================
Total params: 76,257
Trainable params: 76,257
Non-trainable params: 0
_________________________________________________________________
```

CONCLUSION:

The analysis of real GDP data indicates the actual economic output. The data shows how the economy has performed over a specific period, reflecting its overall health and prosperity. By tracking real GDP, policymakers and businesses can assess the country's economic progress and make informed decisions to support future development