Time Series Analysis and Forecasting of Economic Indicators (GDP) of India using SARIMA Model

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
```

In [2]:

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

In [3]:

```
df = pd.read_csv("India_GDP_1960-2022.csv")
```

In [4]:

```
df.head()
```

Out[4]:

	Unnamed: 0	India GDP - Historical Data	India GDP - Historical Data.1	India GDP - Historical Data.2	India GDP - Historical Data.3
0	NaN	Year	GDP in (Billion) $	Per Capita in rupees	Growth %
1	0.0	2021	3173.4	182160	8.95
2	1.0	2020	2667.69	154640	-6.6
3	2.0	2019	2831.55	165760	3.74
4	3.0	2018	2702.93	159840	6.45

In [5]:

```
df.tail()
```

Out[5]:

	Unnamed: 0	India GDP - Historical Data	India GDP - Historical Data.1	India GDP - Historical Data.2	India GDP - Historical Data.3
58	57.0	1964	56.48	9280	7.45
59	58.0	1963	48.42	8080	5.99
60	59.0	1962	42.16	7200	2.93
61	60.0	1961	39.23	6800	3.72
62	61.0	1960	37.03	6560	0

In [6]:

```
data = df.values.tolist()
```

In [7]:

```
column_headers = data[0]
```

In [8]:

```
df = pd.DataFrame(data[1:], columns=column_headers)
```

In [9]:

```
df
```

Out[9]:

	NaN	Year	GDP in (Billion) $	Per Capita in rupees	Growth %
0	0.0	2021	3173.4	182160	8.95
1	1.0	2020	2667.69	154640	-6.6
2	2.0	2019	2831.55	165760	3.74
3	3.0	2018	2702.93	159840	6.45
4	4.0	2017	2651.47	158480	6.8
...	...	...	...	...	...
57	57.0	1964	56.48	9280	7.45
58	58.0	1963	48.42	8080	5.99
59	59.0	1962	42.16	7200	2.93
60	60.0	1961	39.23	6800	3.72
61	61.0	1960	37.03	6560	0

62 rows × 5 columns

In [10]:

```
df.shape
```

Out[10]:

```
(62, 5)
```

In [11]:

```
df.columns
```

Out[11]:

```
Index([nan, 'Year', 'GDP in (Billion) $', 'Per Capita in rupees', 'Growth
%'], dtype='object')
```

In [12]:

```
df = df[['Year', 'GDP in (Billion) $', 'Per Capita in rupees', 'Growth %']]
```

In [13]:

```
df
```

Out[13]:

	Year	GDP in (Billion) $	Per Capita in rupees	Growth %
0	2021	3173.4	182160	8.95
1	2020	2667.69	154640	-6.6
2	2019	2831.55	165760	3.74
3	2018	2702.93	159840	6.45
4	2017	2651.47	158480	6.8
...	...	...	...	...
57	1964	56.48	9280	7.45
58	1963	48.42	8080	5.99
59	1962	42.16	7200	2.93
60	1961	39.23	6800	3.72
61	1960	37.03	6560	0

62 rows × 4 columns

In [14]:

```
df.shape
```

Out[14]:

```
(62, 4)
```

In [15]:

```
df.columns
```

Out[15]:

```
Index(['Year', 'GDP in (Billion) $', 'Per Capita in rupees', 'Growth %'],
dtype='object')
```

In [16]:

```
new_column_names = {
    'Year': 'Year',
    'GDP in (Billion) $': 'GDP',
    'Per Capita in rupees': 'Per Capita',
    'Growth %': 'Growth'
}
```

In [17]:

```
df.rename(columns=new_column_names, inplace=True)
```

In [18]:

```
df
```

Out[18]:

	Year	GDP	Per Capita	Growth
0	2021	3173.4	182160	8.95
1	2020	2667.69	154640	-6.6
2	2019	2831.55	165760	3.74
3	2018	2702.93	159840	6.45
4	2017	2651.47	158480	6.8
...	...	...	...	...
57	1964	56.48	9280	7.45
58	1963	48.42	8080	5.99
59	1962	42.16	7200	2.93
60	1961	39.23	6800	3.72
61	1960	37.03	6560	0

62 rows × 4 columns

In [19]:

```
df.duplicated().sum()
```

Out[19]:

```
0
```

In [20]:

```
df.isnull().sum()
```

Out[20]:

```
Year          0
GDP           0
Per Capita    0
Growth        0
dtype: int64
```

In [21]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 62 entries, 0 to 61
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   Year        62 non-null     object
 1   GDP         62 non-null     object
 2   Per Capita  62 non-null     object
 3   Growth      62 non-null     object
dtypes: object(4)
memory usage: 2.1+ KB
```

In [22]:

```
df = df[::-1]
```

In [23]:

```
df['Year'] = pd.to_numeric(df['Year'], errors='coerce')
df['GDP'] = pd.to_numeric(df['GDP'], errors='coerce')
df['Per Capita'] = pd.to_numeric(df['Per Capita'], errors='coerce')
df['Growth'] = pd.to_numeric(df['Growth'], errors='coerce')
```

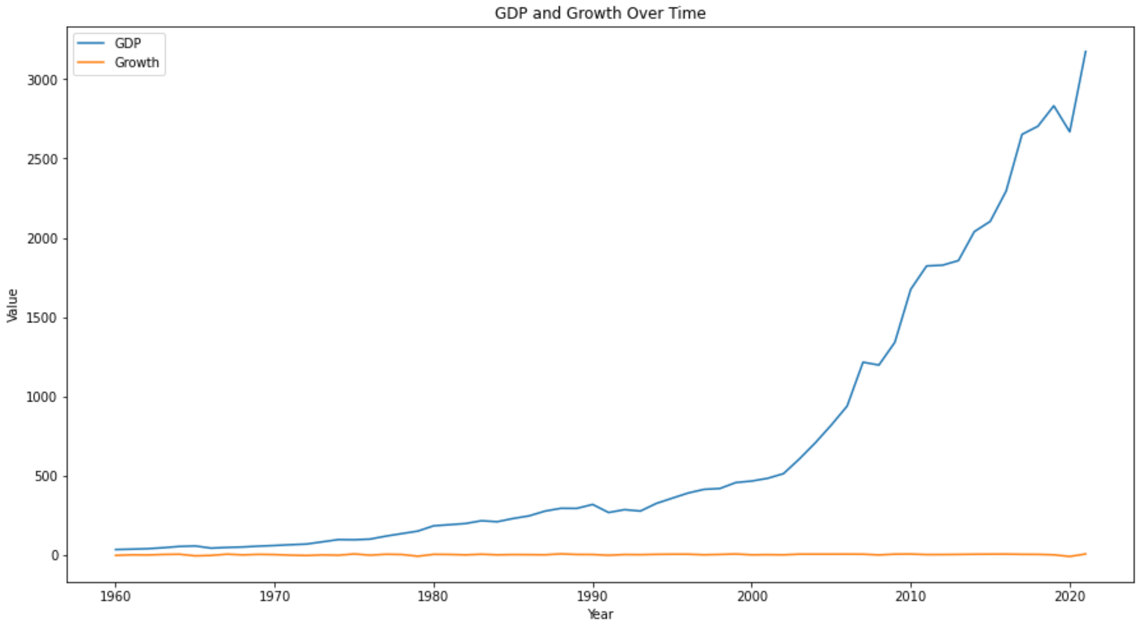
In [24]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 62 entries, 61 to 0
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Year        62 non-null     int64  
 1   GDP         62 non-null     float64
 2   Per Capita  62 non-null     int64  
 3   Growth      62 non-null     float64
dtypes: float64(2), int64(2)
memory usage: 2.1 KB
```

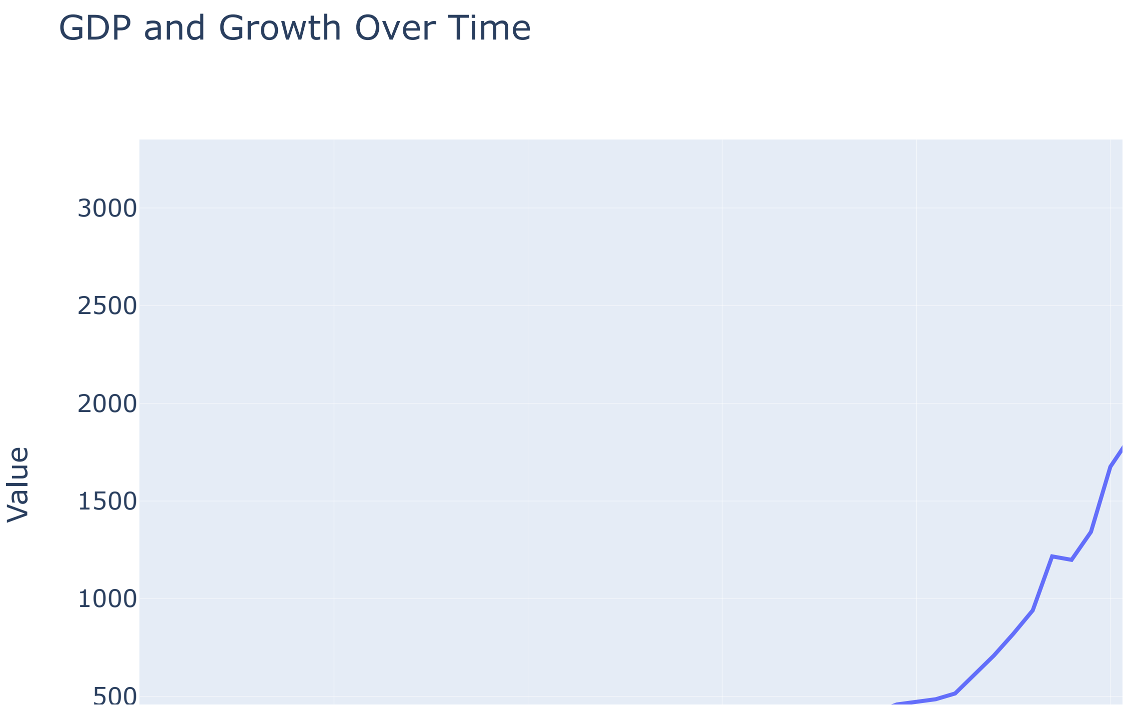
In [25]:

```
plt.figure(figsize=(15, 8))
plt.plot(df['Year'], df['GDP'], label='GDP')
plt.plot(df['Year'], df['Growth'], label='Growth')
plt.xlabel('Year')
plt.ylabel('Value')
plt.title('GDP and Growth Over Time')
plt.legend()
plt.show()
```



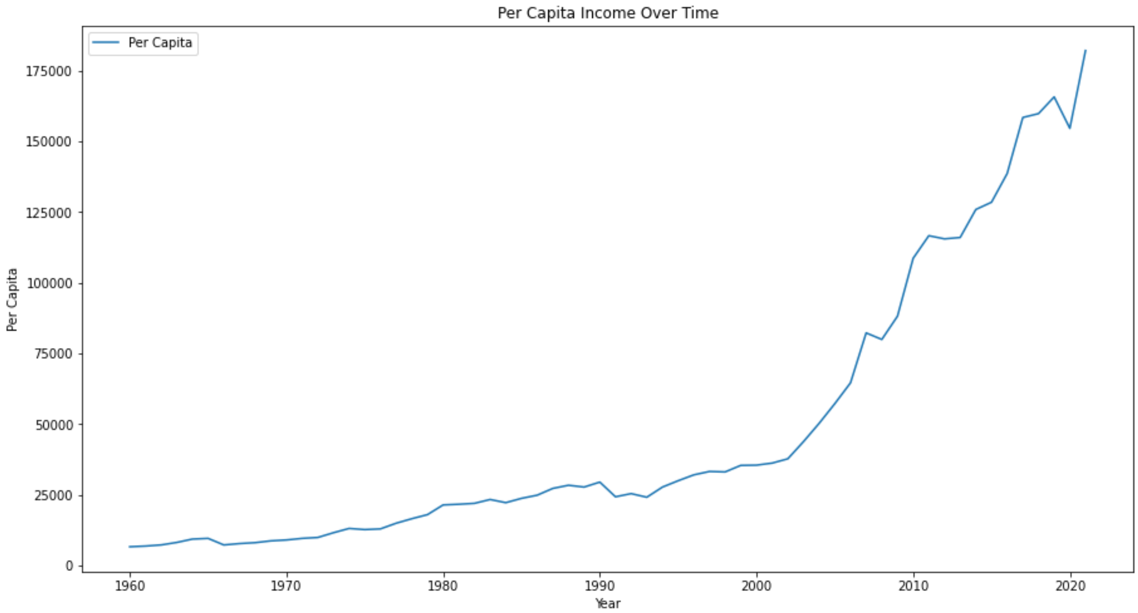
In [26]:

```
fig = px.line(df, x='Year', y=['GDP', 'Growth'], title='GDP and Growth Over Time')
fig.update_xaxes(title_text='Year')
fig.update_yaxes(title_text='Value')
fig.show()
```



In [27]:

```
plt.figure(figsize=(15, 8))
plt.plot(df['Year'], df['Per Capita'], label='Per Capita')
plt.xlabel('Year')
plt.ylabel('Per Capita')
plt.title('Per Capita Income Over Time')
plt.legend()
plt.show()
```

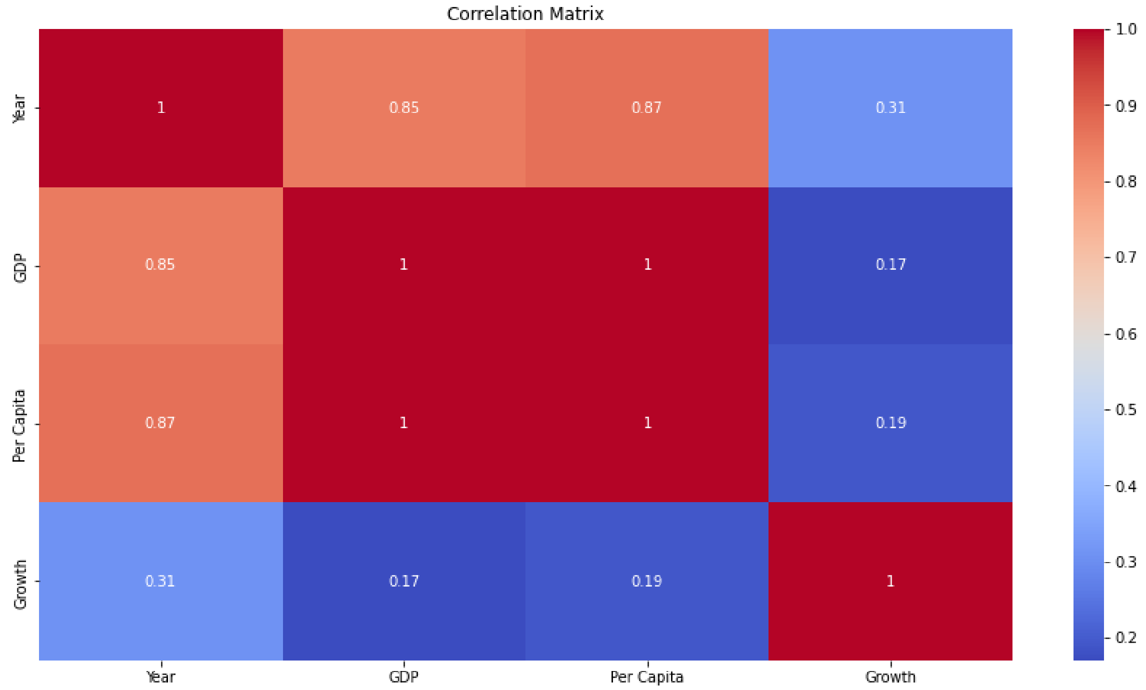


In [28]:

```
fig = px.line(df, x='Year', y='Per Capita', title='Per Capita Income Over Time')
fig.update_xaxes(title_text='Year')
fig.update_yaxes(title_text='Per Capita')
fig.show()
```

In [29]:

```
plt.figure(figsize=(15, 8))
correlation_matrix = df.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```



In [30]:

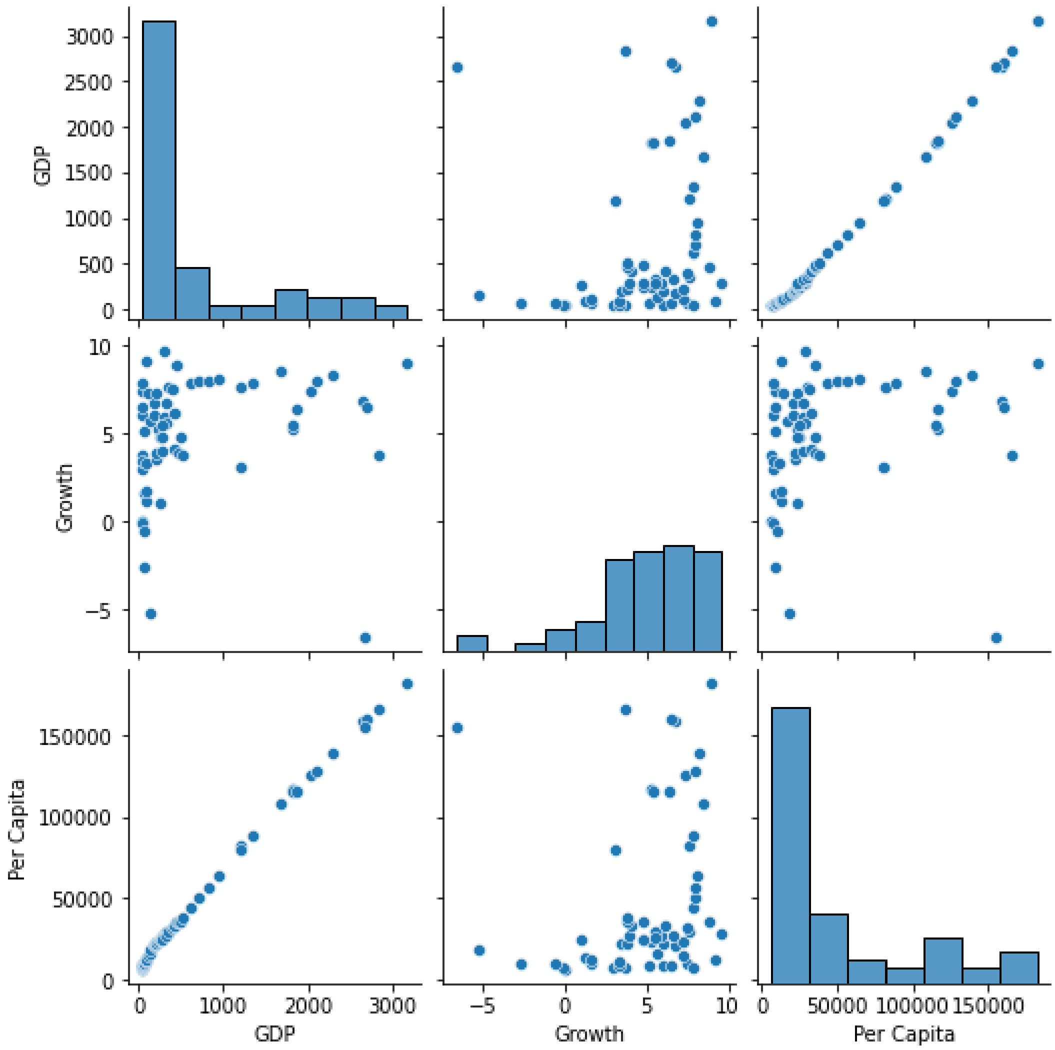
```
top_gdp_year = df[df['GDP'] == df['GDP'].max()]['Year'].values[0]
top_growth_year = df[df['Growth'] == df['Growth'].max()]['Year'].values[0]
print(f"Top GDP Year: {top_gdp_year}")
print(f"Top Growth Year: {top_growth_year}")
```

```
Top GDP Year: 2021
Top Growth Year: 1988
```

In [31]:

```
plt.figure(figsize=(20, 10))
sns.pairplot(df[['GDP', 'Growth', 'Per Capita']])
plt.show()
```

<Figure size 1440x720 with 0 Axes>



In [32]:

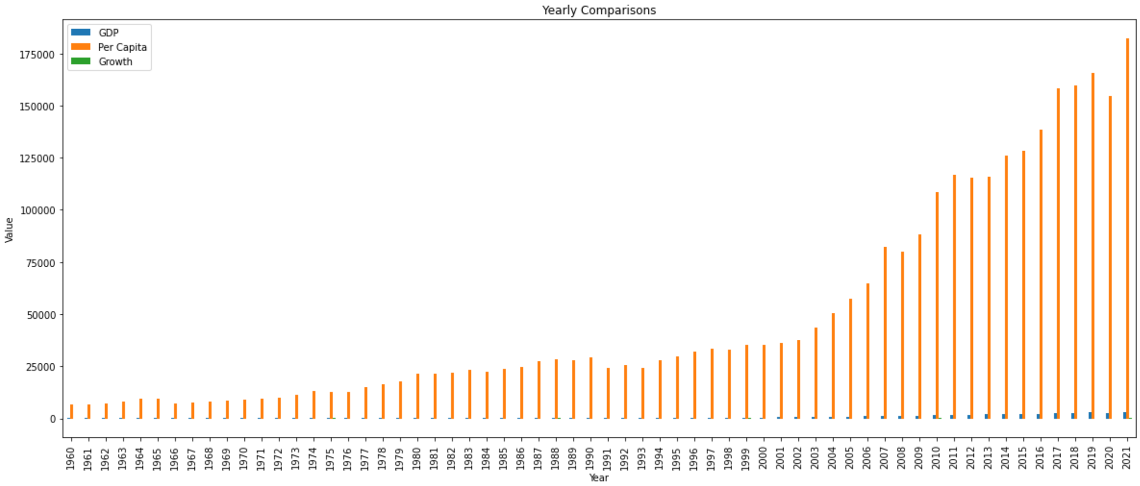
```
fig = px.scatter_matrix(df[['GDP', 'Growth', 'Per Capita']])
fig.update_layout(title='Pair Plot of GDP, Growth, and Per Capita')
fig.show()
```

In [33]:

```
df1 = df.copy()
```

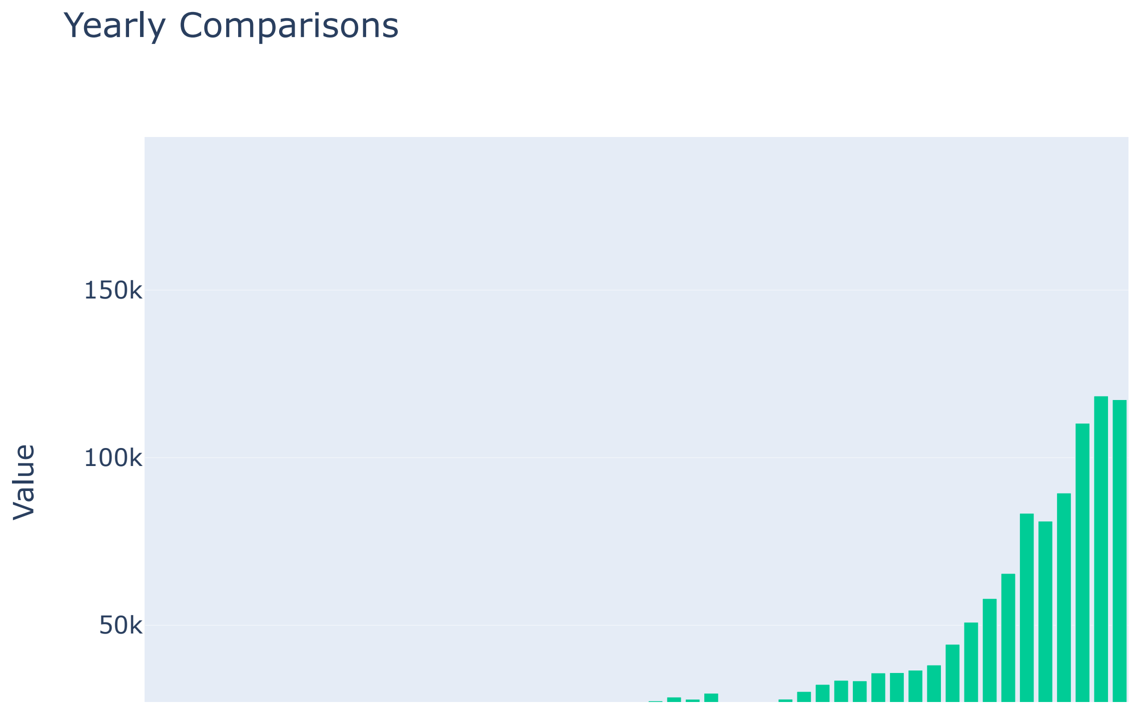
In [34]:

```
df1.set_index('Year').plot(kind='bar', figsize=(20, 8))
plt.xlabel('Year')
plt.ylabel('Value')
plt.title('Yearly Comparisons')
plt.show()
```



In [35]:

```
fig = px.bar(df1, x='Year', y=['GDP', 'Growth', 'Per Capita'],
             title='Yearly Comparisons', labels={'value': 'Value', 'variable': 'Variable
fig.update_xaxes(title_text='Year')
fig.update_yaxes(title_text='Value')
fig.show()
```



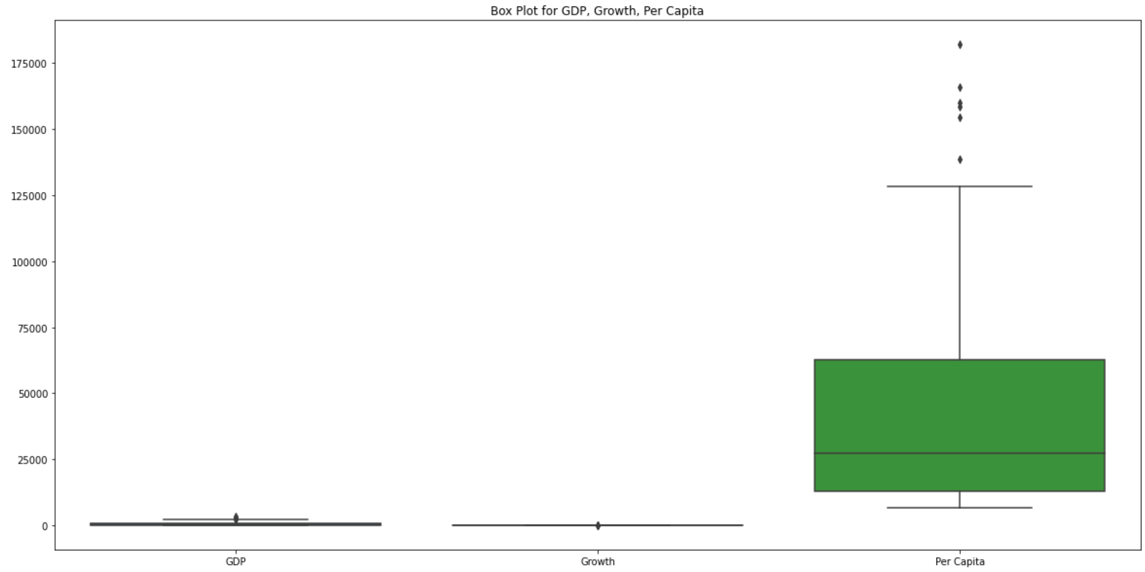
In [36]:

```
summary = df.describe()
print(summary)
```

```
              Year          GDP     Per Capita     Growth
count    62.000000    62.000000      62.000000  62.000000
mean   1990.500000   699.036452   48210.322581   5.007258
std      18.041619   867.228056   49386.668108   3.319231
min    1960.000000    37.030000    6560.000000  -6.600000
25%    1975.250000   100.327500   12920.000000   3.725000
50%    1990.500000   292.125000   27440.000000   5.620000
75%    2005.750000   910.290000   62720.000000   7.525000
max    2021.000000  3173.400000  182160.000000   9.630000
```

In [37]:

```
plt.figure(figsize=(20, 10))
sns.boxplot(data=df[['GDP', 'Growth', 'Per Capita']])
plt.title('Box Plot for GDP, Growth, Per Capita')
plt.show()
```



In [38]:

```
fig = px.box(df, y=['GDP', 'Growth', 'Per Capita'],
             title='Box Plot for GDP, Growth, and Per Capita', labels={'variable': 'Vari
fig.update_yaxes(title_text='Value')
fig.show()
```

In [39]:

```
df1['Year'] = pd.to_datetime(df1['Year'], format='%Y')
```

In [40]:

```
df1.set_index('Year', inplace=True)
```

In [41]:

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

In [42]:

```
def adfuller_test(data):
    result = adfuller(data)
    labels = ['ADF Test Statistic', 'p-value', '#Lags Used', 'Number of Observations']
    for value, label in zip(result, labels):
        print(label + ' : ' + str(value))
    if result[1] <= 0.05:
        print("Strong evidence against the null hypothesis (Ho), reject the null hypothe
    else:
        print("Weak evidence against the null hypothesis, indicating it is non-stationary

adfuller_test(df1['GDP'])
```

```
ADF Test Statistic : 2.148185119036238
p-value : 0.9988370923940496
#Lags Used : 10
Number of Observations : 51
Weak evidence against the null hypothesis, indicating it is non-stationary
```

In [43]:

```
df1['GDP First Difference'] = df1['GDP'] - df1['GDP'].shift(1)
```

In [44]:

```
adfuller_test(df1['GDP First Difference'].dropna())
```

```
ADF Test Statistic : 1.7367005046994224
p-value : 0.9982148515618909
#Lags Used : 11
Number of Observations : 49
Weak evidence against the null hypothesis, indicating it is non-stationary
```

In [45]:

```
df1['GDP Second Difference'] = df1['GDP First Difference'] - df1['GDP First Difference']
```

In [46]:

```
adfuller_test(df1['GDP Second Difference'].dropna())
```

```
ADF Test Statistic : -2.116870440421475
p-value : 0.23778524425550263
#Lags Used : 11
Number of Observations : 48
Weak evidence against the null hypothesis, indicating it is non-stationary
```

In [47]:

```
df1['Seasonal First Difference']= df1['GDP'] - df1['GDP'].shift(1)
```

In [48]:

```
adfuller_test(df1['Seasonal First Difference'].dropna())
```

```
ADF Test Statistic : 1.7367005046994224
p-value : 0.9982148515618909
#Lags Used : 11
Number of Observations : 49
Weak evidence against the null hypothesis, indicating it is non-stationary
```

In [49]:

```
df1['Seasonal Second Difference'] = df1['Seasonal First Difference'] - df1['Seasonal Fir
adfuller_test(df1['Seasonal Second Difference'].dropna())
```

```
ADF Test Statistic : -2.116870440421475
p-value : 0.23778524425550263
#Lags Used : 11
Number of Observations : 48
Weak evidence against the null hypothesis, indicating it is non-stationary
```

In [50]:

```
df1['Seasonal Third Difference'] = df1['Seasonal Second Difference'] - df1['Seasonal Sec
adfuller_test(df1['Seasonal Third Difference'].dropna())
```

```
ADF Test Statistic : -4.859517737791979
p-value : 4.178617461257609e-05
#Lags Used : 11
Number of Observations : 47
Strong evidence against the null hypothesis (Ho), reject the null hypothes
is. Data is stationary
```

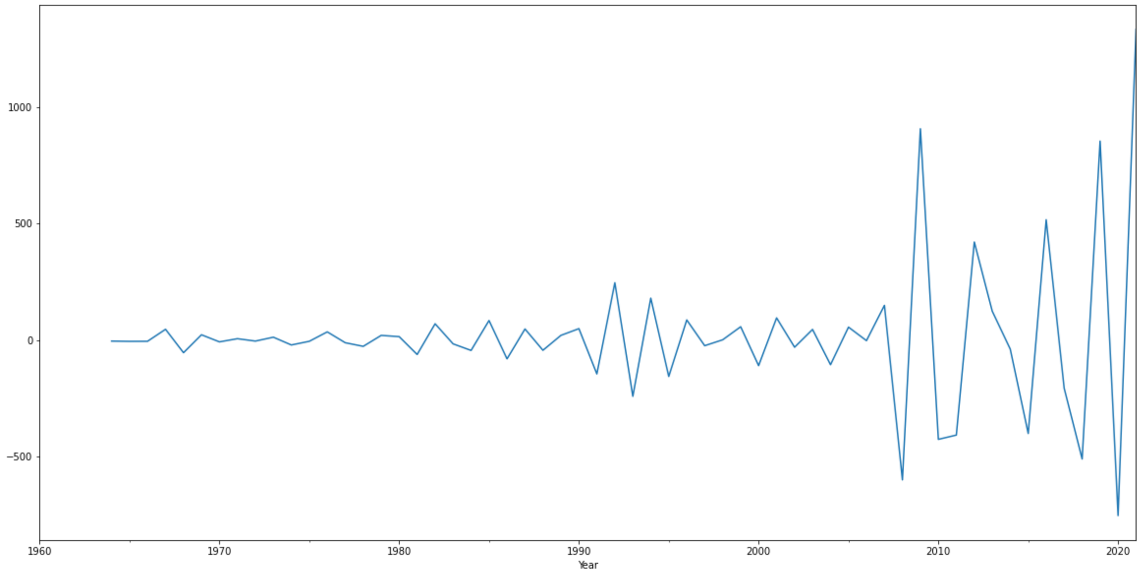
In [51]:

```
df1['Seasonal Fourth Difference'] = df1['Seasonal Third Difference'] - df1['Seasonal Thi
adfuller_test(df1['Seasonal Fourth Difference'].dropna())
```

```
ADF Test Statistic : -5.120695810316111
p-value : 1.2736063580398873e-05
#Lags Used : 11
Number of Observations : 46
Strong evidence against the null hypothesis (Ho), reject the null hypothes
is. Data is stationary
```

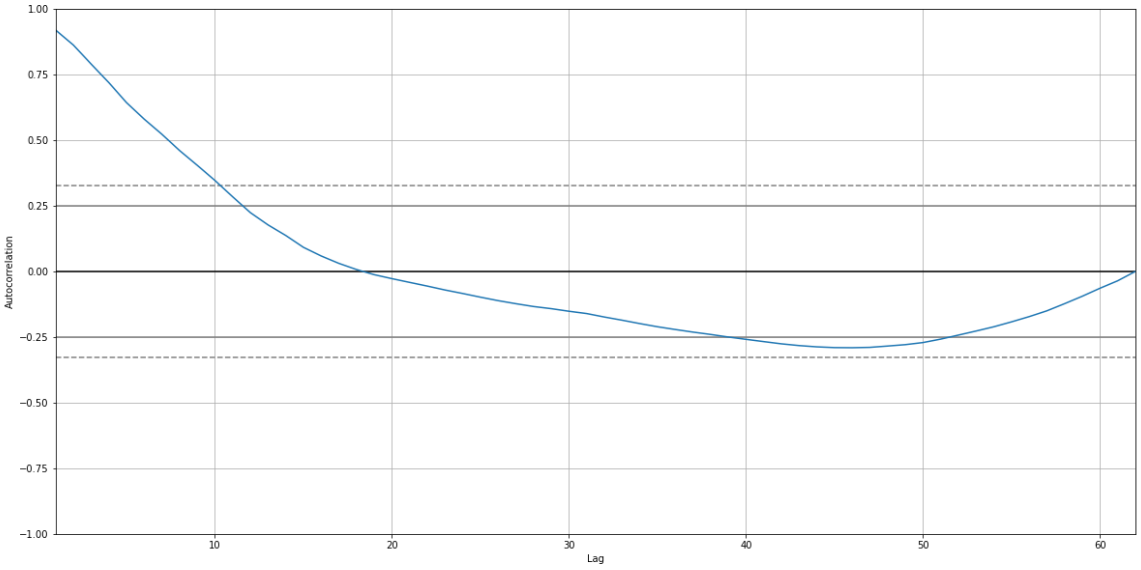
In [52]:

```
plt.figure(figsize=(20, 10))
df1['Seasonal Fourth Difference'].plot()
plt.show()
```



In [54]:

```
from pandas.plotting import autocorrelation_plot
plt.figure(figsize=(20, 10))
autocorrelation_plot(df1['GDP'])
plt.show()
```

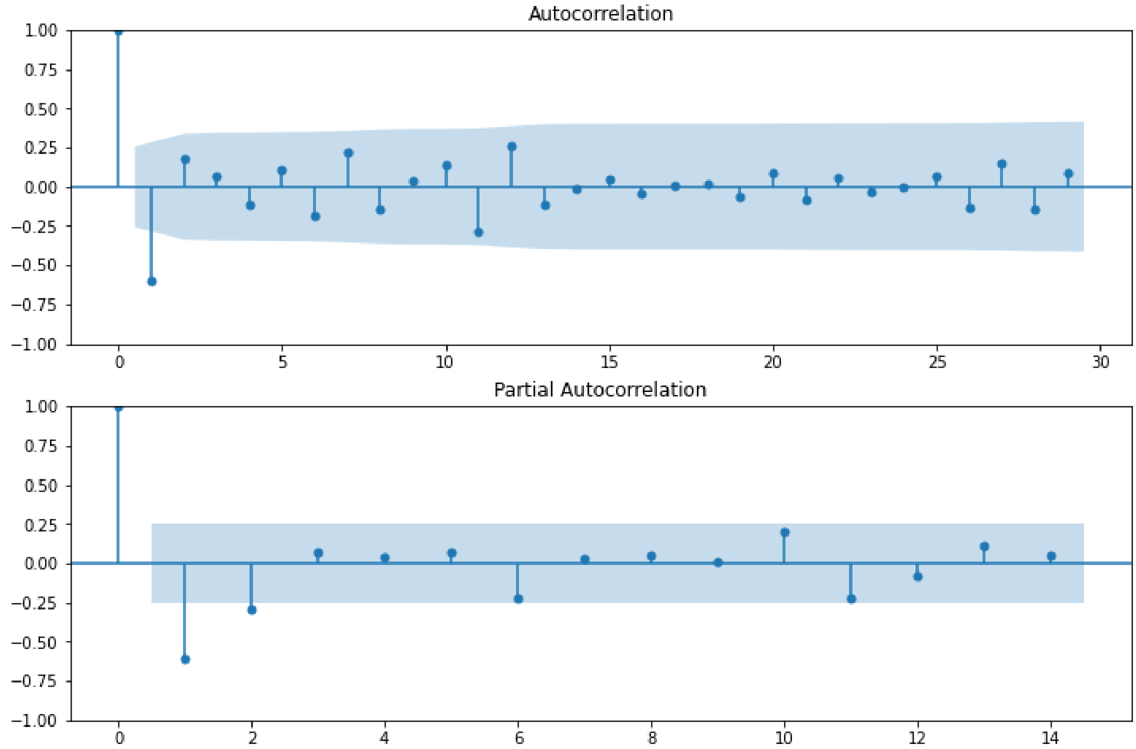


In [57]:

```
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
import statsmodels.api as sm
```

In [58]:

```
fig = plt.figure(figsize=(12, 8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(df1['Seasonal Fourth Difference'].dropna(), lags=29, ax=a
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(df1['Seasonal Fourth Difference'].dropna(), lags=14, ax=
```



In [60]:

```
# For non-seasonal data
#p=1, d=1, q=0 or 1

from statsmodels.tsa.arima.model import ARIMA
model = ARIMA(df1['GDP'], order=(1, 1, 1))
```

```
c:\pythonn\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueW
arning:

No frequency information was provided, so inferred frequency AS-JAN will b
e used.

c:\pythonn\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueW
arning:

No frequency information was provided, so inferred frequency AS-JAN will b
e used.

c:\pythonn\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueW
arning:

No frequency information was provided, so inferred frequency AS-JAN will b
e used.
```

In [61]:

```
model_fit=model.fit()
```

In [62]:

```
model_fit.summary()
```

Out[62]:

SARIMAX Results

Dep. Variable:	GDP	**No. Observations:**	62
Model:	ARIMA(1, 1, 1)	**Log Likelihood**	-363.933
Date:	Tue, 15 Aug 2023	**AIC**	733.865
Time:	13:59:20	**BIC**	740.198
Sample:	01-01-1960	**HQIC**	736.347
	- 01-01-2021		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.9897	0.028	35.677	0.000	0.935	1.044
ma.L1	-0.8536	0.082	-10.390	0.000	-1.015	-0.693
sigma2	8695.1295	905.764	9.600	0.000	6919.865	1.05e+04

Ljung-Box (L1) (Q):	3.39	**Jarque-Bera (JB):**	152.96
Prob(Q):	0.07	**Prob(JB):**	0.00
Heteroskedasticity (H):	325.20	**Skew:**	1.35
Prob(H) (two-sided):	0.00	**Kurtosis:**	10.27

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

In [63]:

```
forecast = model_fit.predict(start=62,end=70,dynamic=True)
```

In [64]:

```
forecast
```

Out[64]:

```
2022-01-01    3314.823803
2023-01-01    3454.796517
2024-01-01    3593.333031
2025-01-01    3730.448081
2026-01-01    3866.156252
2027-01-01    4000.471980
2028-01-01    4133.409552
2029-01-01    4264.983108
2030-01-01    4395.206644
Freq: AS-JAN, Name: predicted_mean, dtype: float64
```

In [65]:

```
forecast_index = pd.date_range(start='2022-01-01', end='2030-01-01', freq='YS')
forecast_df = pd.DataFrame({'Forecast': forecast.values}, index=forecast_index)
```

In [66]:

```
df1_forecast = pd.concat([df1, forecast_df], axis=1)
```

In [67]:

```
df1_forecast
```

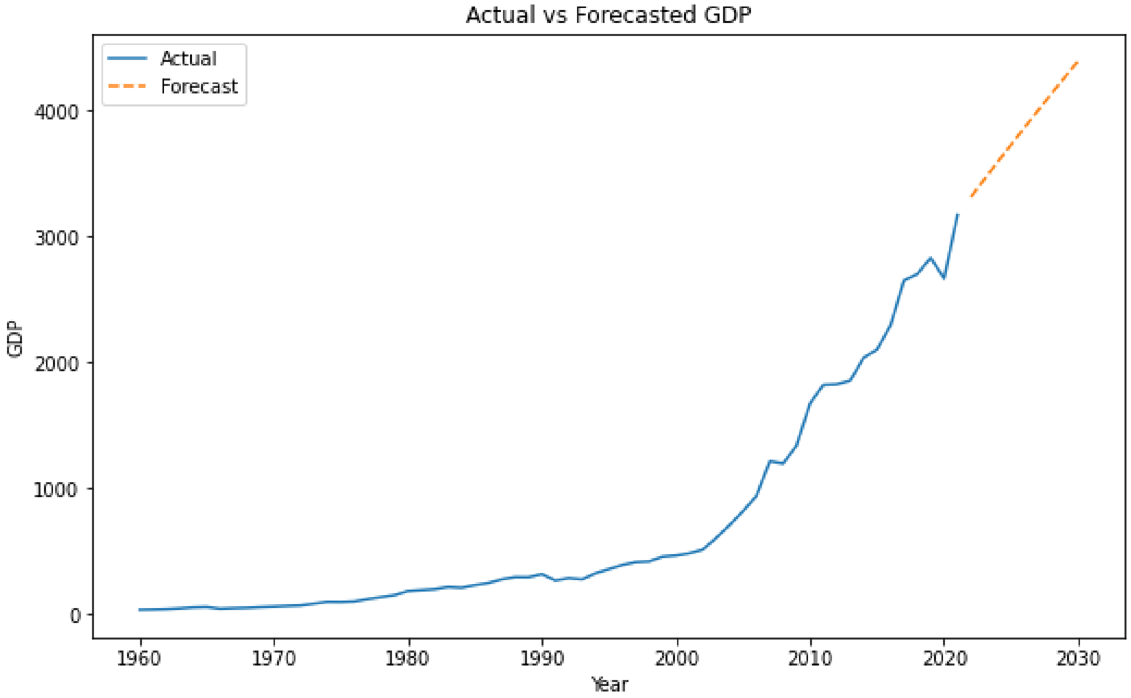
Out[67]:

	GDP	Per Capita	Growth	GDP First Difference	GDP Second Difference	Seasonal First Difference	Seasonal Second Difference	Seasonal Third Difference	Seas F Differ
1960-01-01	37.03	6560.0	0.00	NaN	NaN	NaN	NaN	NaN	
1961-01-01	39.23	6800.0	3.72	2.20	NaN	2.20	NaN	NaN	
1962-01-01	42.16	7200.0	2.93	2.93	0.73	2.93	0.73	NaN	
1963-01-01	48.42	8080.0	5.99	6.26	3.33	6.26	3.33	2.60	
1964-01-01	56.48	9280.0	7.45	8.06	1.80	8.06	1.80	-1.53	
...	...	...	...	...	...	...	...	...	
2026-01-01	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
2027-01-01	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
2028-01-01	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
2029-01-01	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
2030-01-01	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

71 rows × 10 columns

In [68]:

```
plt.figure(figsize=(10, 6))
plt.plot(df1_forecast.index, df1_forecast['GDP'], label='Actual')
plt.plot(df1_forecast.index, df1_forecast['Forecast'], label='Forecast', linestyle='--')
plt.xlabel('Year')
plt.ylabel('GDP')
plt.title('Actual vs Forecasted GDP')
plt.legend()
plt.show()
```



In [69]:

```
fig = go.Figure()
fig.add_trace(go.Scatter(x=df1_forecast.index, y=df1_forecast['GDP'], mode='lines', name
fig.add_trace(go.Scatter(x=df1_forecast.index, y=df1_forecast['Forecast'], mode='lines',
fig.update_layout(
    xaxis_title='Year',
    yaxis_title='GDP',
    title='Actual vs Forecasted GDP'
)
fig.show()
```