Copy_of_fin_analytics_try_2_j_comp

April 19, 2024

[ ]:
```python
import pandas as pd

data = pd.read_excel('/content/transformed_data (1).xlsx', sheet_name='Sheet1')

# Display the first few rows of the dataset
print("First few rows of the dataset:")
data.head()
```

```
First few rows of the dataset:
```

[ ]:
```
       Country Name  Year  Exports of goods and services (annual % growth)  \
0  Egypt, Arab Rep.  2000                                          7.028999
1  Egypt, Arab Rep.  2001                                          3.290676
2  Egypt, Arab Rep.  2002                                          4.973599
3  Egypt, Arab Rep.  2003                                         13.832853
4  Egypt, Arab Rep.  2004                                         25.316456

   Food exports (% of merchandise exports)  \
0                                 7.996953
1                                 9.962074
2                                 8.753934
3                                 8.501811
4                                 9.769097

   Fuel imports (% of merchandise imports)  GDP growth (annual %)  \
0                                 7.584877               6.370004
1                                 4.924288               3.535252
2                                 3.975515               2.390204
3                                 5.168025               3.193455
4                                 8.572553               4.092072

   High-technology exports (% of manufactured exports)  \
0                                                0.0
1                                                0.0
2                                                0.0
3                                                0.0
4                                                0.0
```

```
   Inflation, GDP deflator (annual %)  Inflation, consumer prices (annual %)  \
0                            3.944407                               2.683805
1                            1.867700                               2.269757
2                            3.165579                               2.737239
3                            6.777494                               4.507776
4                           11.669908                              11.270619

   Military expenditure (% of GDP)  Trade (% of GDP)  \
0                         2.551265         39.017936
1                         2.985569         39.810427
2                         3.278455         40.987068
3                         3.177285         46.179641
4                         2.877531         57.819905

   Trade in services (% of GDP)
0                     17.344003
1                     16.630150
2                     18.731546
3                     21.854744
4                     28.199961
```

[ ]:
```
# Check the columns present in the dataset
print("\nColumns in the dataset:")
print(data.columns)
```

```
Columns in the dataset:
Index(['Country Name', 'Year',
       'Exports of goods and services (annual % growth)',
       'Food exports (% of merchandise exports)',
       'Fuel imports (% of merchandise imports)', 'GDP growth (annual %)',
       'High-technology exports (% of manufactured exports)',
       'Inflation, GDP deflator (annual %)',
       'Inflation, consumer prices (annual %)',
       'Military expenditure (% of GDP)', 'Trade (% of GDP)',
       'Trade in services (% of GDP)'],
      dtype='object')
```

[ ]:
```
# Check the data types of the columns
print("\nData types of the columns:")
print(data.dtypes)
```

```
Data types of the columns:
Country Name                                            object
Year                                                     int64
Exports of goods and services (annual % growth)        float64
Food exports (% of merchandise exports)                float64
```

```
Fuel imports (% of merchandise imports)                 float64
GDP growth (annual %)                                   float64
High-technology exports (% of manufactured exports)     float64
Inflation, GDP deflator (annual %)                      float64
Inflation, consumer prices (annual %)                   float64
Military expenditure (% of GDP)                         float64
Trade (% of GDP)                                        float64
Trade in services (% of GDP)                            float64
dtype: object
```

[ ]:
```
# Check for missing values in the dataset
print("\nMissing values in the dataset:")
print(data.isnull().sum())
```

```
Missing values in the dataset:
Country Name                                            0
Year                                                    0
Exports of goods and services (annual % growth)         0
Food exports (% of merchandise exports)                 0
Fuel imports (% of merchandise imports)                 0
GDP growth (annual %)                                   0
High-technology exports (% of manufactured exports)     0
Inflation, GDP deflator (annual %)                      0
Inflation, consumer prices (annual %)                   0
Military expenditure (% of GDP)                         0
Trade (% of GDP)                                        0
Trade in services (% of GDP)                            0
dtype: int64
```

[ ]:
```
column_means = data[data.columns[2:]].mean()

# Display the mean of each column
print("Mean of each column:")
print(column_means)
```

```
Mean of each column:
Exports of goods and services (annual % growth)          3.027259
Food exports (% of merchandise exports)                  6.186416
Fuel imports (% of merchandise imports)                  7.486914
GDP growth (annual %)                                    2.905349
High-technology exports (% of manufactured exports)      1.749638
Inflation, GDP deflator (annual %)                       8.945287
Inflation, consumer prices (annual %)                    6.084513
Military expenditure (% of GDP)                          3.469588
Trade (% of GDP)                                        68.498214
Trade in services (% of GDP)                            13.379571
dtype: float64
```

[ ]:
```
df = data

# Convert columns to numeric
cols_to_convert = df.columns[2:]
df[cols_to_convert] = df[cols_to_convert].apply(pd.to_numeric, errors='coerce')

# Drop rows with any NaN values
df.dropna(inplace=True)

# Print the first few rows of the DataFrame
print(df.head())
```

```
      Country Name  Year  Exports of goods and services (annual % growth)  \
0  Egypt, Arab Rep.  2000                                          7.028999
1  Egypt, Arab Rep.  2001                                          3.290676
2  Egypt, Arab Rep.  2002                                          4.973599
3  Egypt, Arab Rep.  2003                                         13.832853
4  Egypt, Arab Rep.  2004                                         25.316456

   Food exports (% of merchandise exports)  \
0                                 7.996953
1                                 9.962074
2                                 8.753934
3                                 8.501811
4                                 9.769097

   Fuel imports (% of merchandise imports)  GDP growth (annual %)  \
0                                 7.584877               6.370004
1                                 4.924288               3.535252
2                                 3.975515               2.390204
3                                 5.168025               3.193455
4                                 8.572553               4.092072

   High-technology exports (% of manufactured exports)  \
0                                                0.0
1                                                0.0
2                                                0.0
3                                                0.0
4                                                0.0

   Inflation, GDP deflator (annual %)  Inflation, consumer prices (annual %)  \
0                            3.944407                               2.683805
1                            1.867700                               2.269757
2                            3.165579                               2.737239
3                            6.777494                               4.507776
4                           11.669908                              11.270619
```

```
   Military expenditure (% of GDP)  Trade (% of GDP)  \
0                         2.551265         39.017936
1                         2.985569         39.810427
2                         3.278455         40.987068
3                         3.177285         46.179641
4                         2.877531         57.819905

   Trade in services (% of GDP)
0                     17.344003
1                     16.630150
2                     18.731546
3                     21.854744
4                     28.199961
```

[ ]:
```python
# Exclude non-numeric columns
numeric_df = df.select_dtypes(include='number')

# Calculate correlation
correlation = numeric_df.corr()

# Print correlation matrix
print(correlation)
```

```
                                                        Year  \
Year                                                1.000000
Exports of goods and services (annual % growth)    -0.057414
Food exports (% of merchandise exports)             0.059655
Fuel imports (% of merchandise imports)            -0.039049
GDP growth (annual %)                              -0.168757
High-technology exports (% of manufactured expo…  0.267271
Inflation, GDP deflator (annual %)                  0.051627
Inflation, consumer prices (annual %)               0.089499
Military expenditure (% of GDP)                    -0.271181
Trade (% of GDP)                                   -0.377001
Trade in services (% of GDP)                        0.001444

                                                    Exports of goods and
services (annual % growth)  \
Year
-0.057414
Exports of goods and services (annual % growth)
1.000000
Food exports (% of merchandise exports)
-0.009530
Fuel imports (% of merchandise imports)
0.074986
GDP growth (annual %)
0.666746
```

```
High-technology exports (% of manufactured expo…
-0.024633
Inflation, GDP deflator (annual %)
-0.028720
Inflation, consumer prices (annual %)
-0.013883
Military expenditure (% of GDP)
-0.017319
Trade (% of GDP)
0.098266
Trade in services (% of GDP)
0.003992

                                                    Food exports (% of
merchandise exports)  \
Year
0.059655
Exports of goods and services (annual % growth)
-0.009530
Food exports (% of merchandise exports)
1.000000
Fuel imports (% of merchandise imports)
0.546381
GDP growth (annual %)
-0.122148
High-technology exports (% of manufactured expo…
0.026871
Inflation, GDP deflator (annual %)
0.075261
Inflation, consumer prices (annual %)
0.112819
Military expenditure (% of GDP)
-0.160428
Trade (% of GDP)
-0.130517
Trade in services (% of GDP)
0.200942

                                                    Fuel imports (% of
merchandise imports)  \
Year
-0.039049
Exports of goods and services (annual % growth)
0.074986
Food exports (% of merchandise exports)
0.546381
Fuel imports (% of merchandise imports)
1.000000
```

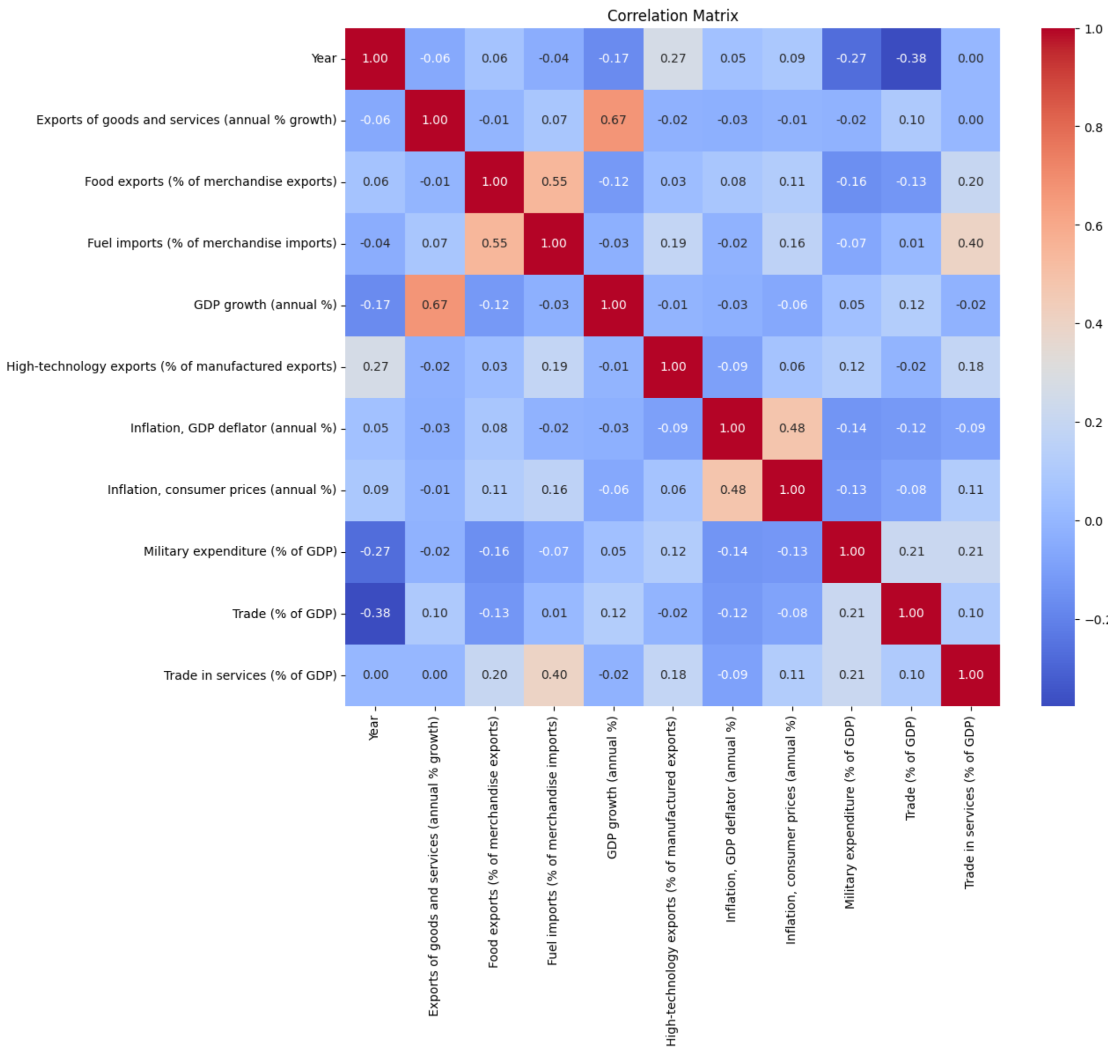
```
GDP growth (annual %)
-0.029291
High-technology exports (% of manufactured expo…
0.186013
Inflation, GDP deflator (annual %)
-0.017866
Inflation, consumer prices (annual %)
0.161518
Military expenditure (% of GDP)
-0.070252
Trade (% of GDP)
0.012419
Trade in services (% of GDP)
0.399689

                                                    GDP growth (annual %)  \
Year                                                            -0.168757
Exports of goods and services (annual % growth)                  0.666746
Food exports (% of merchandise exports)                         -0.122148
Fuel imports (% of merchandise imports)                         -0.029291
GDP growth (annual %)                                            1.000000
High-technology exports (% of manufactured expo…              -0.012234
Inflation, GDP deflator (annual %)                              -0.028594
Inflation, consumer prices (annual %)                           -0.063057
Military expenditure (% of GDP)                                  0.047418
Trade (% of GDP)                                                 0.120653
Trade in services (% of GDP)                                    -0.017465

                                                    High-technology exports (%
of manufactured exports)  \
Year
0.267271
Exports of goods and services (annual % growth)
-0.024633
Food exports (% of merchandise exports)
0.026871
Fuel imports (% of merchandise imports)
0.186013
GDP growth (annual %)
-0.012234
High-technology exports (% of manufactured expo…
1.000000
Inflation, GDP deflator (annual %)
-0.092827
Inflation, consumer prices (annual %)
0.059484
Military expenditure (% of GDP)
0.116086
```

```
Trade (% of GDP)
-0.016595
Trade in services (% of GDP)
0.182898

                                                    Inflation, GDP deflator
(annual %)  \
Year
0.051627
Exports of goods and services (annual % growth)
-0.028720
Food exports (% of merchandise exports)
0.075261
Fuel imports (% of merchandise imports)
-0.017866
GDP growth (annual %)
-0.028594
High-technology exports (% of manufactured expo…
-0.092827
Inflation, GDP deflator (annual %)
1.000000
Inflation, consumer prices (annual %)
0.483409
Military expenditure (% of GDP)
-0.140031
Trade (% of GDP)
-0.119524
Trade in services (% of GDP)
-0.090090

                                                    Inflation, consumer prices
(annual %)  \
Year
0.089499
Exports of goods and services (annual % growth)
-0.013883
Food exports (% of merchandise exports)
0.112819
Fuel imports (% of merchandise imports)
0.161518
GDP growth (annual %)
-0.063057
High-technology exports (% of manufactured expo…
0.059484
Inflation, GDP deflator (annual %)
0.483409
Inflation, consumer prices (annual %)
1.000000
```

```
Military expenditure (% of GDP)
-0.125875
Trade (% of GDP)
-0.081537
Trade in services (% of GDP)
0.113212

                                                    Military expenditure (% of
GDP)  \
Year
-0.271181
Exports of goods and services (annual % growth)
-0.017319
Food exports (% of merchandise exports)
-0.160428
Fuel imports (% of merchandise imports)
-0.070252
GDP growth (annual %)
0.047418
High-technology exports (% of manufactured expo…
0.116086
Inflation, GDP deflator (annual %)
-0.140031
Inflation, consumer prices (annual %)
-0.125875
Military expenditure (% of GDP)
1.000000
Trade (% of GDP)
0.214235
Trade in services (% of GDP)
0.212825

                                                    Trade (% of GDP)  \
Year                                                       -0.377001
Exports of goods and services (annual % growth)             0.098266
Food exports (% of merchandise exports)                    -0.130517
Fuel imports (% of merchandise imports)                     0.012419
GDP growth (annual %)                                       0.120653
High-technology exports (% of manufactured expo…         -0.016595
Inflation, GDP deflator (annual %)                         -0.119524
Inflation, consumer prices (annual %)                      -0.081537
Military expenditure (% of GDP)                             0.214235
Trade (% of GDP)                                            1.000000
Trade in services (% of GDP)                                0.102303

                                                    Trade in services (% of GDP)
Year                                                                    0.001444
Exports of goods and services (annual % growth)                         0.003992
```

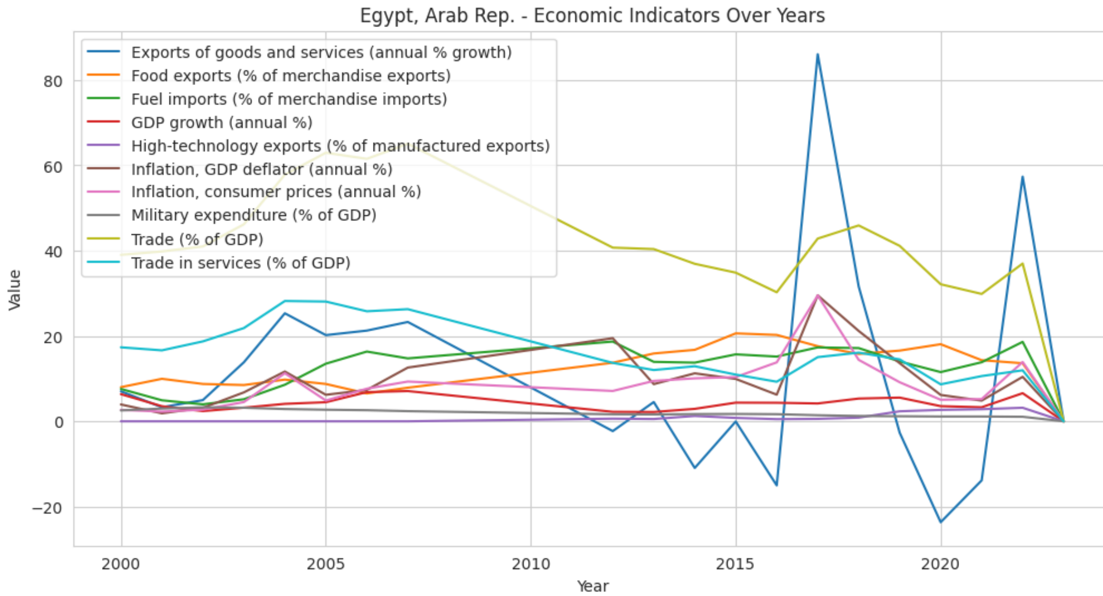
```
Food exports (% of merchandise exports)                          0.200942
Fuel imports (% of merchandise imports)                          0.399689
GDP growth (annual %)                                           -0.017465
High-technology exports (% of manufactured expo…               0.182898
Inflation, GDP deflator (annual %)                              -0.090090
Inflation, consumer prices (annual %)                            0.113212
Military expenditure (% of GDP)                                  0.212825
Trade (% of GDP)                                                 0.102303
Trade in services (% of GDP)                                     1.000000
```

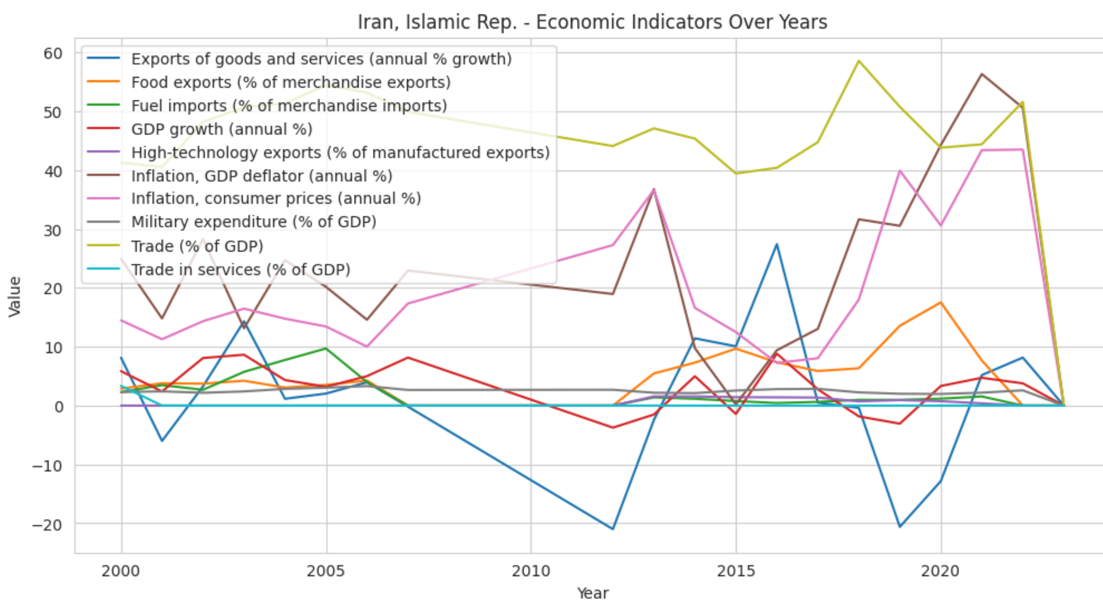
[ ]:
```python
import matplotlib.pyplot as plt
import seaborn as sns

# Create a heatmap of the correlation matrix
plt.figure(figsize=(12, 10))
sns.heatmap(correlation, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()
```

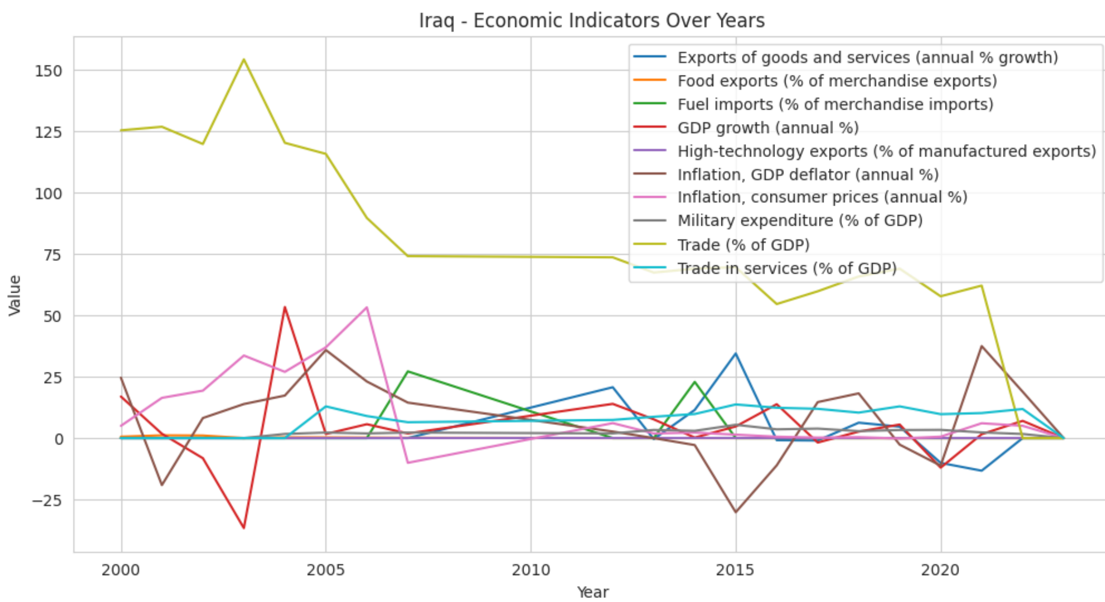


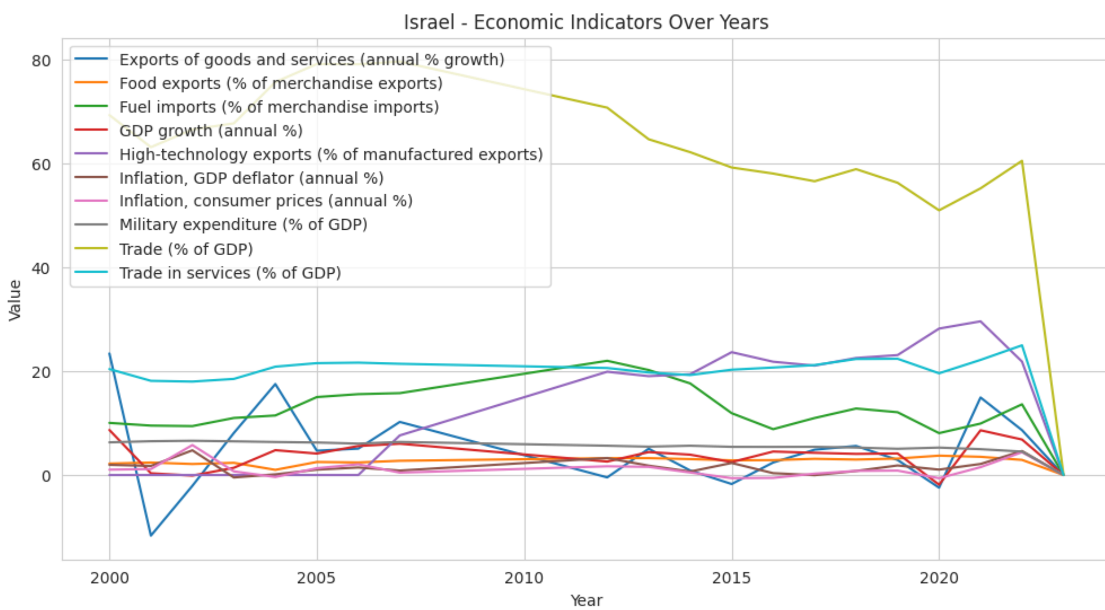
[ ]:

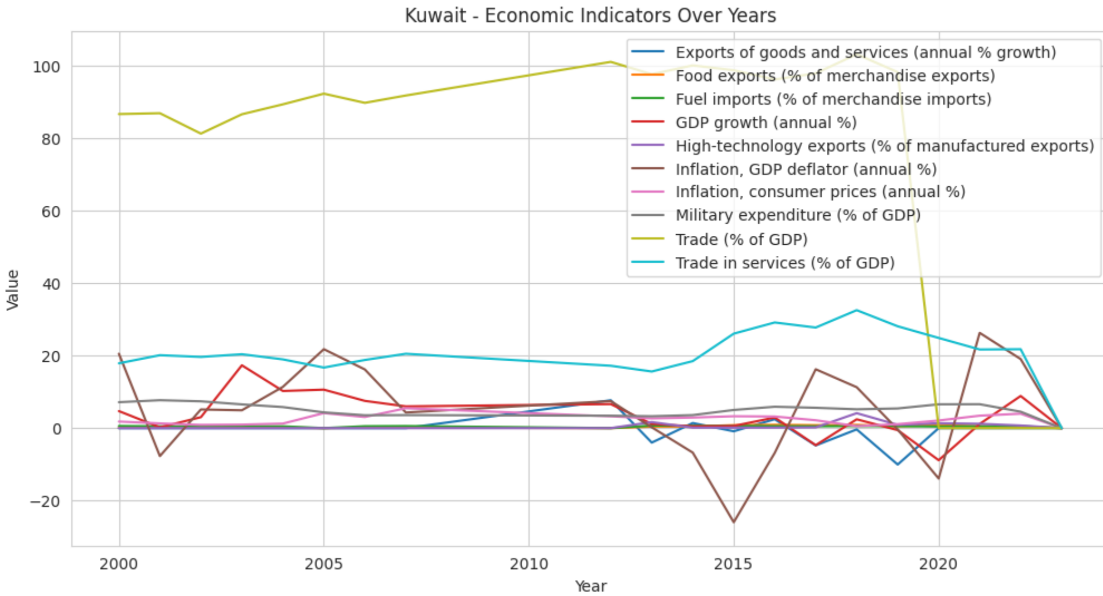
```
sns.set_style("whitegrid")

# Group data by country
grouped_data = df.groupby('Country Name')

# Plot each variable for each country
for country, data in grouped_data:
    plt.figure(figsize=(12, 6))
    plt.title(f"{country} - Economic Indicators Over Years")
    for column in data.columns[2:]:
        plt.plot(data['Year'], data[column], label=column)
    plt.xlabel('Year')
    plt.ylabel('Value')
    plt.legend()
    plt.show()
```

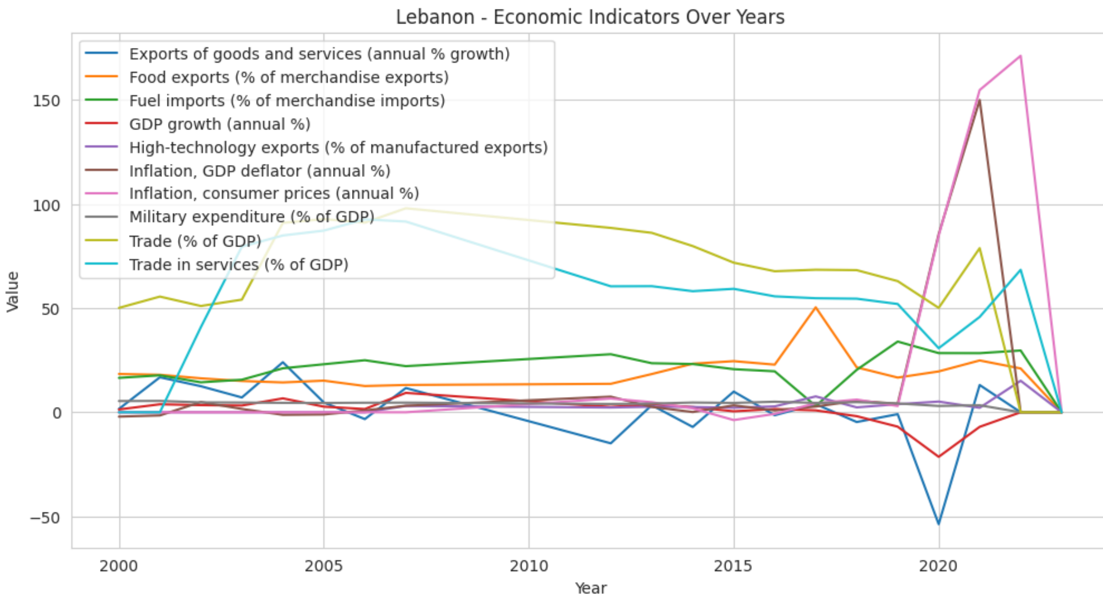

Egypt, Arab Rep. - Economic Indicators Over Years
Exports of goods and services (annual % growth)
Food exports (% of merchandise exports)
Fuel imports (% of merchandise imports)
GDP growth (annual %)
High-technology exports (% of manufactured exports)
Inflation, GDP deflator (annual %)
Inflation, consumer prices (annual %)
Military expenditure (% of GDP)
Trade (% of GDP)
Trade in services (% of GDP)
Value
Year
80
60
40
20
0
−20
2000
2005
2010
2015
2020

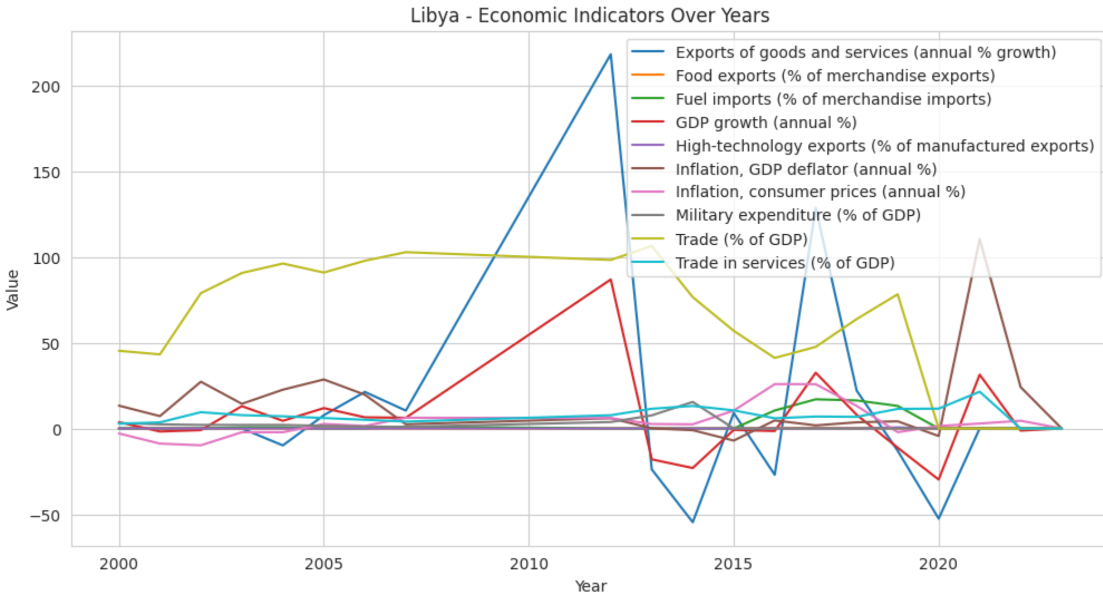

Iran, Islamic Rep. - Economic Indicators Over Years
Exports of goods and services (annual % growth)
Food exports (% of merchandise exports)
Fuel imports (% of merchandise imports)
GDP growth (annual %)
High-technology exports (% of manufactured exports)
Inflation, GDP deflator (annual %)
Inflation, consumer prices (annual %)
Military expenditure (% of GDP)
Trade (% of GDP)
Trade in services (% of GDP)
Value
Year
60
50
40
30
20
10
0
−10
−20
2000
2005
2010
2015
2020

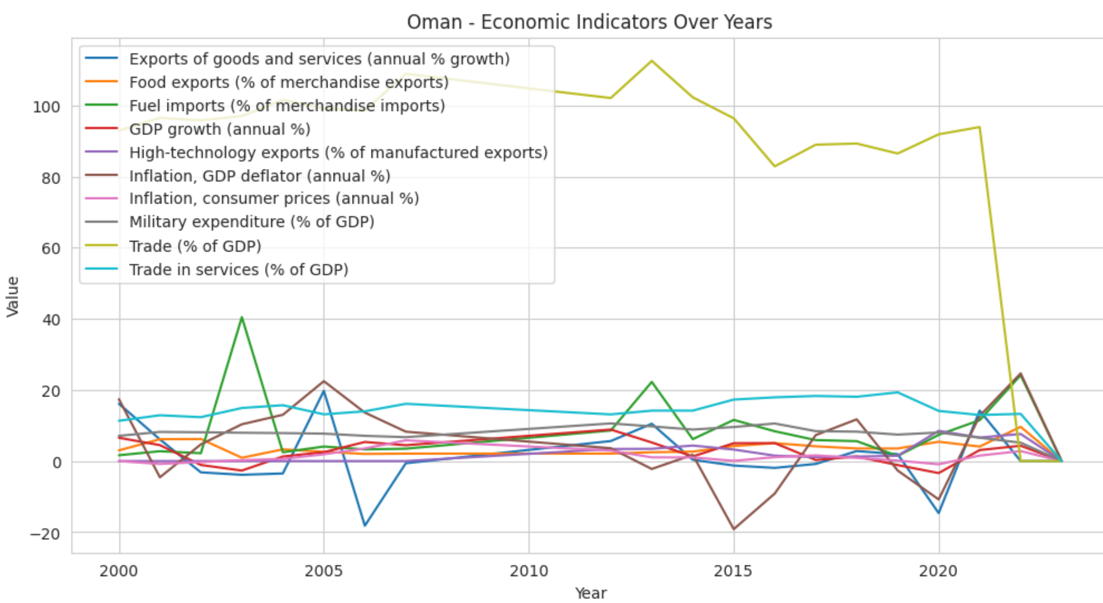

Iraq - Economic Indicators Over Years
Exports of goods and services (annual % growth)
Food exports (% of merchandise exports)
Fuel imports (% of merchandise imports)
GDP growth (annual %)
High-technology exports (% of manufactured exports)
Inflation, GDP deflator (annual %)
Inflation, consumer prices (annual %)
Military expenditure (% of GDP)
Trade (% of GDP)
Trade in services (% of GDP)
Value
Year
150
125
100
75
50
25
0
−25
2000
2005
2010
2015
2020

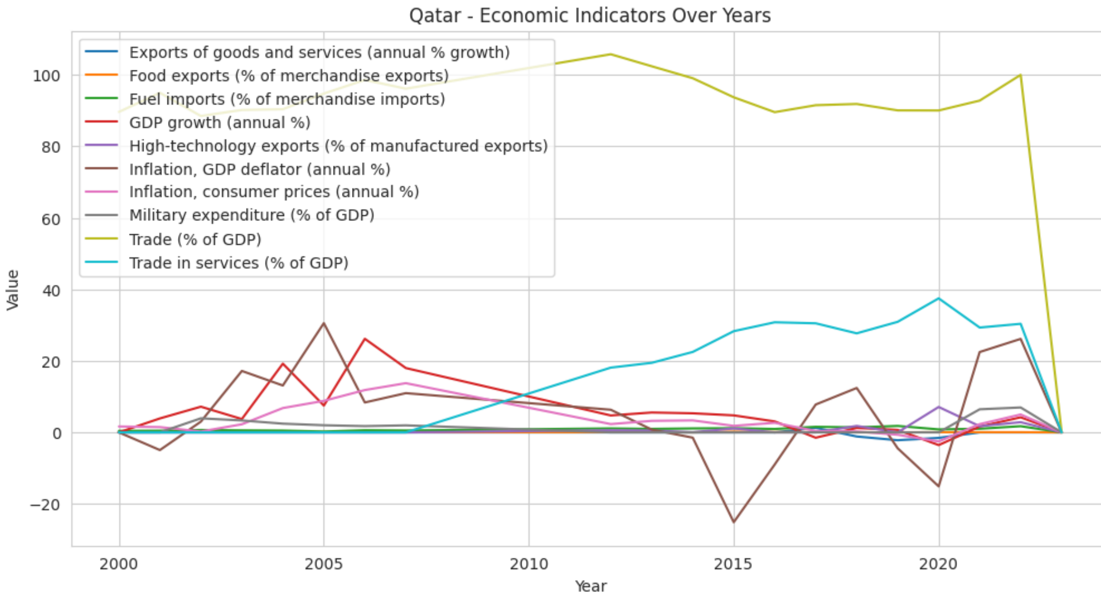

Israel - Economic Indicators Over Years
Exports of goods and services (annual % growth)
Food exports (% of merchandise exports)
Fuel imports (% of merchandise imports)
GDP growth (annual %)
High-technology exports (% of manufactured exports)
Inflation, GDP deflator (annual %)
Inflation, consumer prices (annual %)
Military expenditure (% of GDP)
Trade (% of GDP)
Trade in services (% of GDP)
Value
Year
80
60
40
20
0
2000
2005
2010
2015
2020

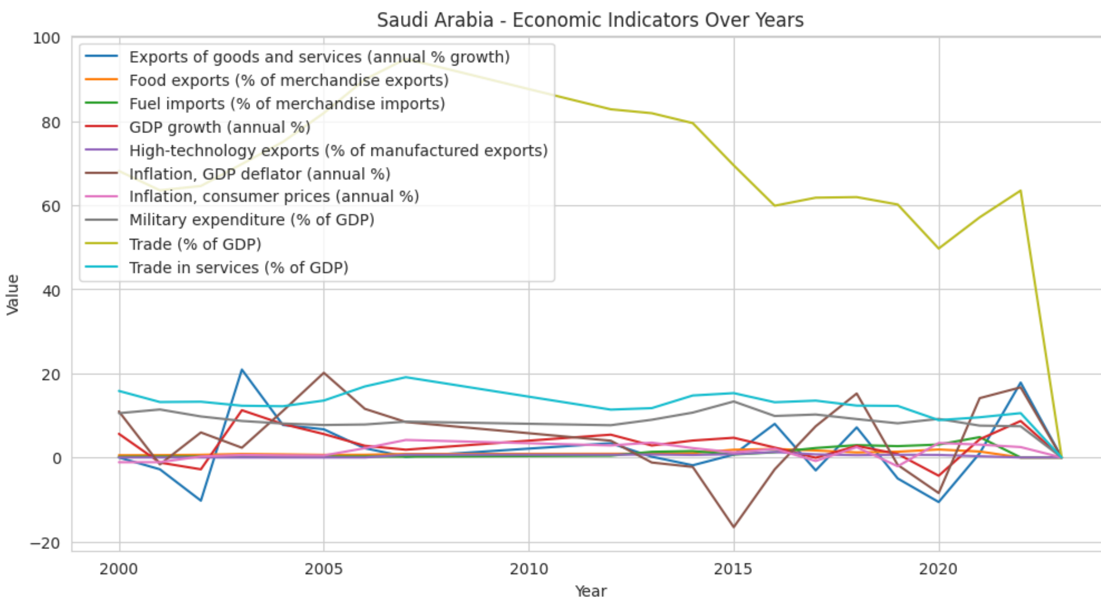

Kuwait - Economic Indicators Over Years
Exports of goods and services (annual % growth)
Food exports (% of merchandise exports)
Fuel imports (% of merchandise imports)
GDP growth (annual %)
High-technology exports (% of manufactured exports)
Inflation, GDP deflator (annual %)
Inflation, consumer prices (annual %)
Military expenditure (% of GDP)
Trade (% of GDP)
Trade in services (% of GDP)
Value
Year
100
80
60
40
20
0
−20
2000
2005
2010
2015
2020

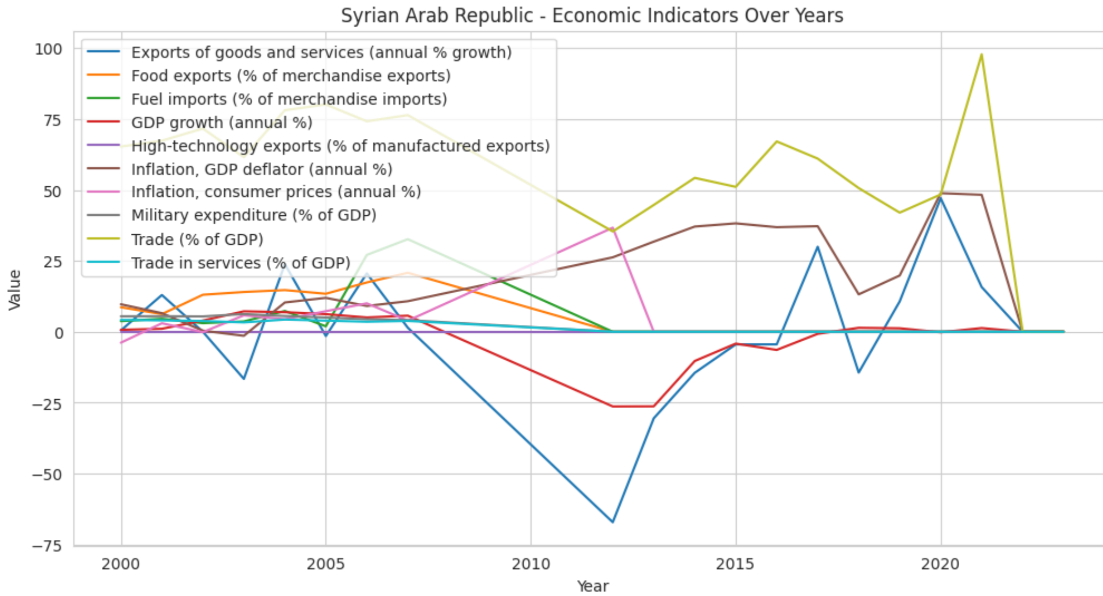

Lebanon - Economic Indicators Over Years
Exports of goods and services (annual % growth)
Food exports (% of merchandise exports)
Fuel imports (% of merchandise imports)
GDP growth (annual %)
High-technology exports (% of manufactured exports)
Inflation, GDP deflator (annual %)
Inflation, consumer prices (annual %)
Military expenditure (% of GDP)
Trade (% of GDP)
Trade in services (% of GDP)
Value
Year
150
100
50
0
−50
2000
2005
2010
2015
2020

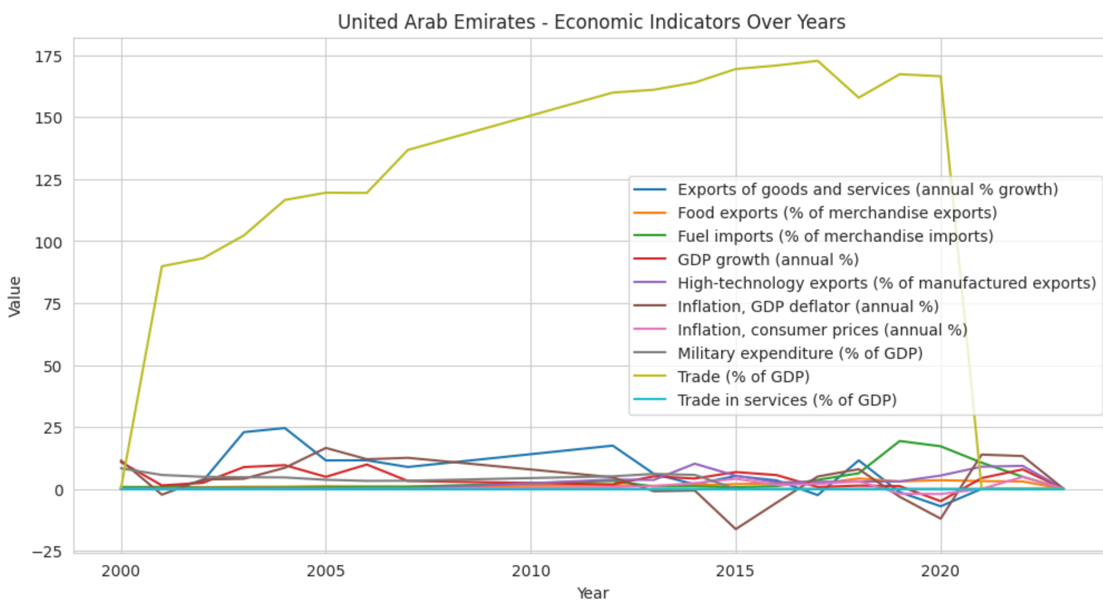

Libya - Economic Indicators Over Years
Exports of goods and services (annual % growth)
Food exports (% of merchandise exports)
Fuel imports (% of merchandise imports)
GDP growth (annual %)
High-technology exports (% of manufactured exports)
Inflation, GDP deflator (annual %)
Inflation, consumer prices (annual %)
Military expenditure (% of GDP)
Trade (% of GDP)
Trade in services (% of GDP)
Value
Year
200
150
100
50
0
−50
2000
2005
2010
2015
2020

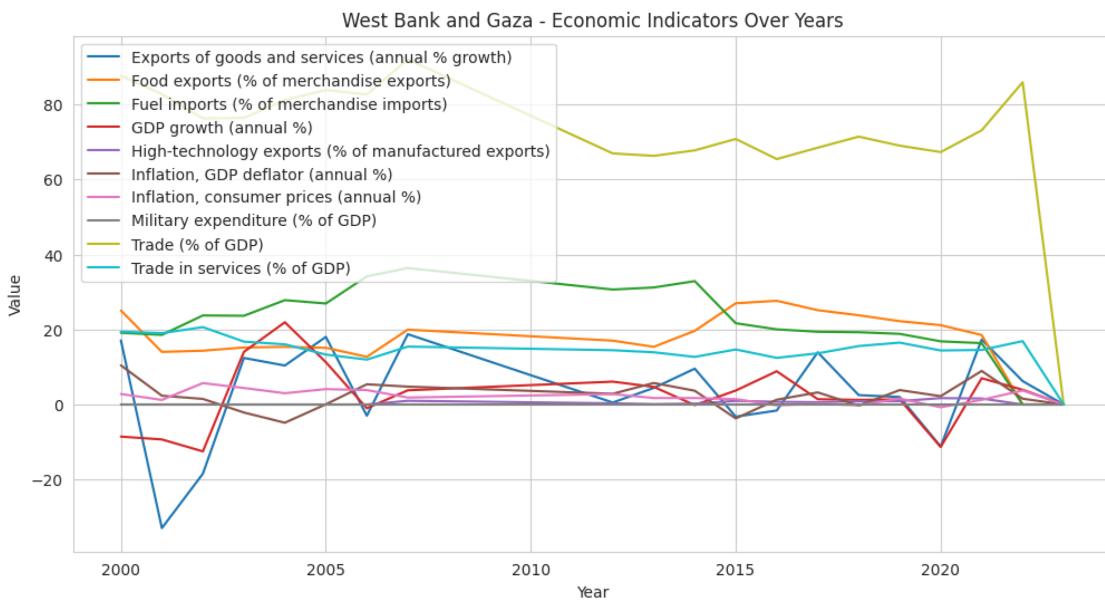

Oman - Economic Indicators Over Years
Exports of goods and services (annual % growth)
Food exports (% of merchandise exports)
Fuel imports (% of merchandise imports)
GDP growth (annual %)
High-technology exports (% of manufactured exports)
Inflation, GDP deflator (annual %)
Inflation, consumer prices (annual %)
Military expenditure (% of GDP)
Trade (% of GDP)
Trade in services (% of GDP)
Value
Year
100
80
60
40
20
0
−20
2000
2005
2010
2015
2020

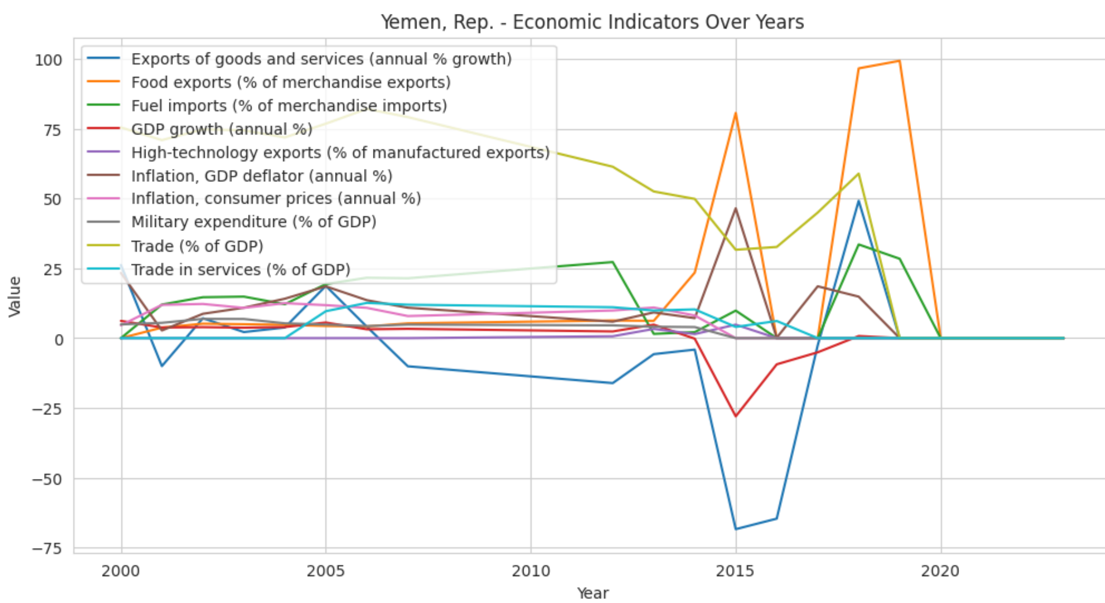

Qatar - Economic Indicators Over Years
Exports of goods and services (annual % growth)
Food exports (% of merchandise exports)
Fuel imports (% of merchandise imports)
GDP growth (annual %)
High-technology exports (% of manufactured exports)
Inflation, GDP deflator (annual %)
Inflation, consumer prices (annual %)
Military expenditure (% of GDP)
Trade (% of GDP)
Trade in services (% of GDP)
Value
Year


Saudi Arabia - Economic Indicators Over Years
Exports of goods and services (annual % growth)
Food exports (% of merchandise exports)
Fuel imports (% of merchandise imports)
GDP growth (annual %)
High-technology exports (% of manufactured exports)
Inflation, GDP deflator (annual %)
Inflation, consumer prices (annual %)
Military expenditure (% of GDP)
Trade (% of GDP)
Trade in services (% of GDP)
Value
Year

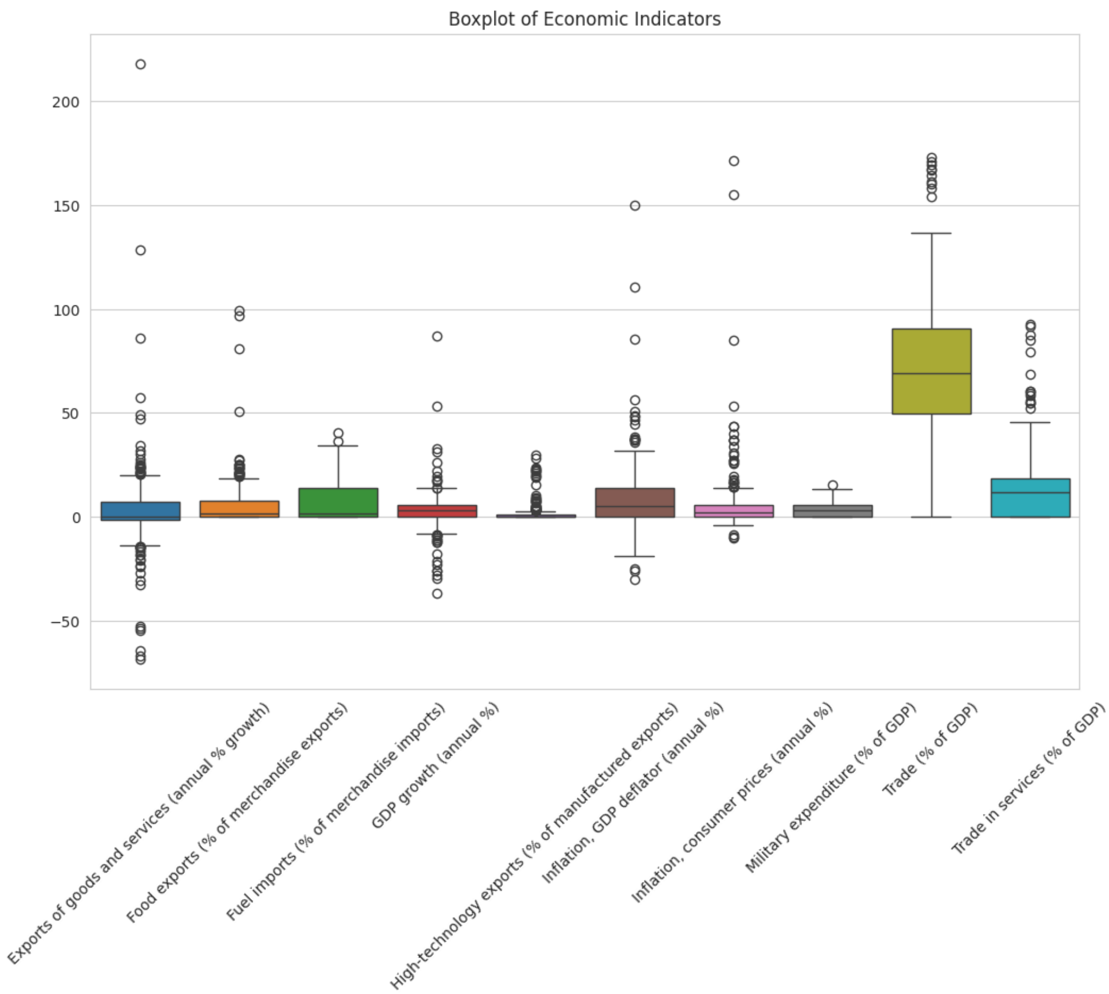

Syrian Arab Republic - Economic Indicators Over Years
Exports of goods and services (annual % growth)
Food exports (% of merchandise exports)
Fuel imports (% of merchandise imports)
GDP growth (annual %)
High-technology exports (% of manufactured exports)
Inflation, GDP deflator (annual %)
Inflation, consumer prices (annual %)
Military expenditure (% of GDP)
Trade (% of GDP)
Trade in services (% of GDP)
Value
Year
100
75
50
25
0
−25
−50
−75
2000
2005
2010
2015
2020


United Arab Emirates - Economic Indicators Over Years
Exports of goods and services (annual % growth)
Food exports (% of merchandise exports)
Fuel imports (% of merchandise imports)
GDP growth (annual %)
High-technology exports (% of manufactured exports)
Inflation, GDP deflator (annual %)
Inflation, consumer prices (annual %)
Military expenditure (% of GDP)
Trade (% of GDP)
Trade in services (% of GDP)
Value
Year
175
150
125
100
75
50
25
0
−25
2000
2005
2010
2015
2020

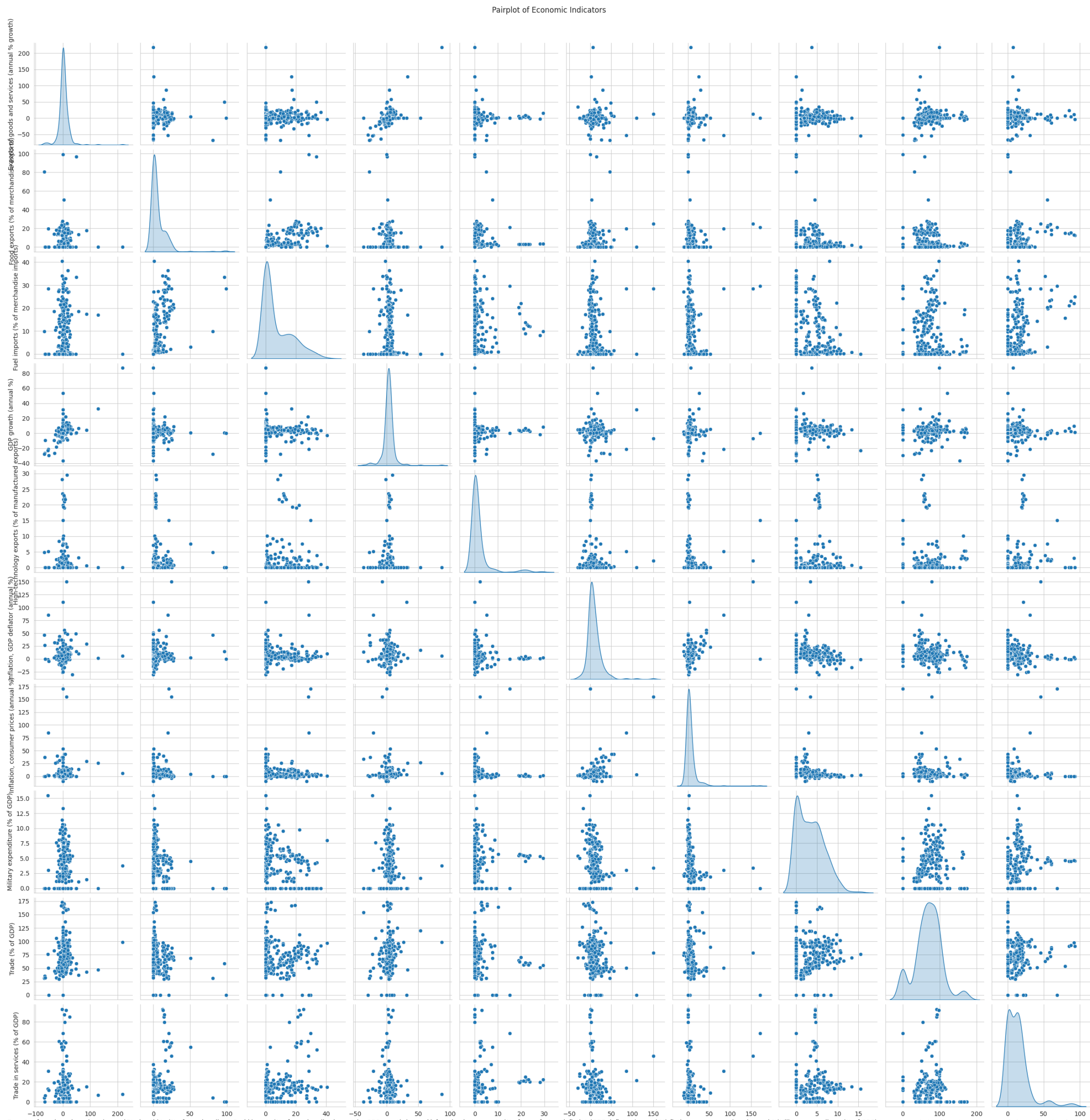




[ ]:
```
plt.figure(figsize=(12, 8))
sns.boxplot(data=df.iloc[:, 2:])
plt.xticks(rotation=45)
plt.title('Boxplot of Economic Indicators')
plt.show()
```



[ ]:
```
sns.pairplot(df, vars=df.columns[2:], diag_kind='kde')
plt.suptitle('Pairplot of Economic Indicators', y=1.02)
plt.show()
```



cluster countries based on their economic indicators,

[ ]:
```
data.head()
```

[ ]:
```
    Country Name  Year  Exports of goods and services (annual % growth)  \
260  Yemen, Rep.  2000                                        26.204317
261  Yemen, Rep.  2001                                       -10.013758
262  Yemen, Rep.  2002                                         7.132448
263  Yemen, Rep.  2003                                         2.171143
264  Yemen, Rep.  2004                                         3.811450

     Food exports (% of merchandise exports)  \
```

```
260                                 0.000000
261                                 3.771346
262                                 5.178558
263                                 4.985725
264                                 4.770815

     Fuel imports (% of merchandise imports)  GDP growth (annual %)  \
260                                 0.000000               6.181916
261                                12.024777               3.803646
262                                14.644732               3.935232
263                                14.864025               3.747398
264                                12.141815               3.972696

     High-technology exports (% of manufactured exports)  \
260                                                0.0
261                                                0.0
262                                                0.0
263                                                0.0
264                                                0.0

     Inflation, GDP deflator (annual %)  \
260                           23.346052
261                            2.748200
262                            8.711880
263                           10.892343
264                           14.113320

     Inflation, consumer prices (annual %)  Military expenditure (% of GDP)  \
260                               4.590000                         4.915392
261                              11.911591                         5.481015
262                              12.238534                         6.895608
263                              10.832361                         6.854552
264                              12.515095                         5.301366

     Trade (% of GDP)  Trade in services (% of GDP)
260         75.438839                           0.0
261         70.892398                           0.0
262         74.730456                           0.0
263         74.382617                           0.0
264         71.846674                           0.0
```

[ ]:
```python
from sklearn.cluster import KMeans
import pandas as pd

# Assuming df contains the loaded data from the file
# Grouping the data by 'Country Name' and calculating the mean GDP growth
```

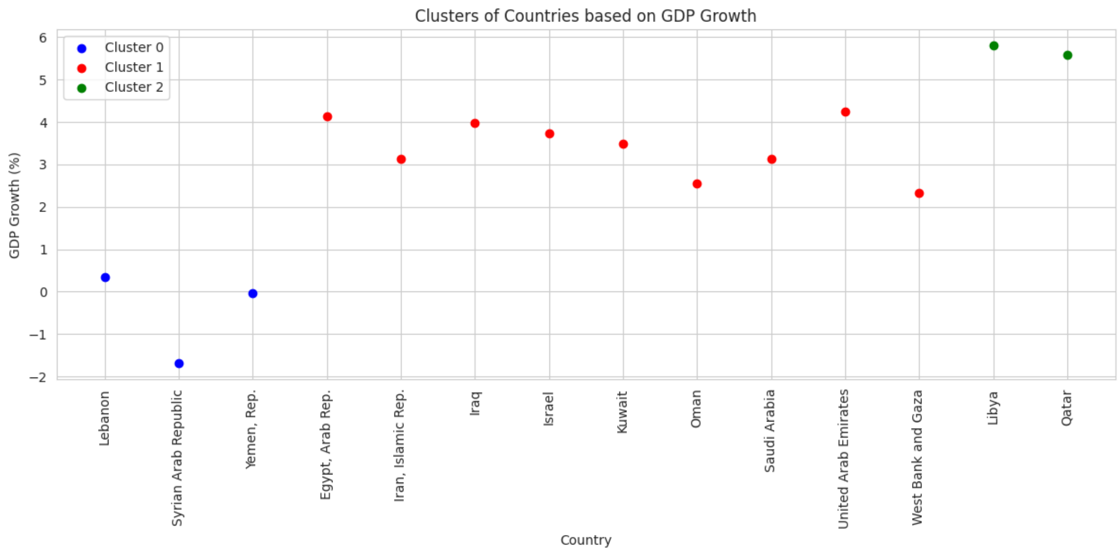
```
grouped_data = df.groupby('Country Name')['GDP growth (annual %)'].mean().
↪reset_index()

# Dropping any rows with missing values
grouped_data.dropna(inplace=True)

# Extracting the country names and GDP growth values
countries = grouped_data['Country Name']
gdp_growth = grouped_data['GDP growth (annual %)']

# Reshape the GDP growth values into a 2D array
gdp_growth_2d = gdp_growth.values.reshape(-1, 1)

# Clustering the countries based on GDP growth using KMeans
kmeans = KMeans(n_clusters=3, random_state=0)
clusters = kmeans.fit_predict(gdp_growth_2d)

# Create a DataFrame with countries and their corresponding clusters
country_clusters_df = pd.DataFrame({'Country': countries, 'Cluster': clusters})

# Print the country and its cluster
for cluster_num in range(3):
    countries_in_cluster = country_clusters_df[country_clusters_df['Cluster']␣
↪== cluster_num]['Country'].unique()
    print(f"Cluster {cluster_num}: {', '.join(countries_in_cluster)}")
```

```
Cluster 0: Lebanon, Syrian Arab Republic, Yemen, Rep.
Cluster 1: Egypt, Arab Rep., Iran, Islamic Rep., Iraq, Israel, Kuwait, Oman,
Saudi Arabia, United Arab Emirates, West Bank and Gaza
Cluster 2: Libya, Qatar
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```

[ ]:
```
# Plotting the clusters
plt.figure(figsize=(12, 6))
colors = ['blue', 'red', 'green']
for cluster_num in range(3):
    countries_in_cluster = country_clusters_df[country_clusters_df['Cluster']␣
↪== cluster_num]
    plt.scatter(countries_in_cluster['Country'],␣
↪grouped_data[grouped_data['Country Name'].
↪isin(countries_in_cluster['Country'])]['GDP growth (annual %)'],
                color=colors[cluster_num], label=f'Cluster {cluster_num}')
plt.title('Clusters of Countries based on GDP Growth')
```

```
plt.xlabel('Country')
plt.ylabel('GDP Growth (%)')
plt.xticks(rotation=90)
plt.legend()
plt.tight_layout()
plt.show()
```



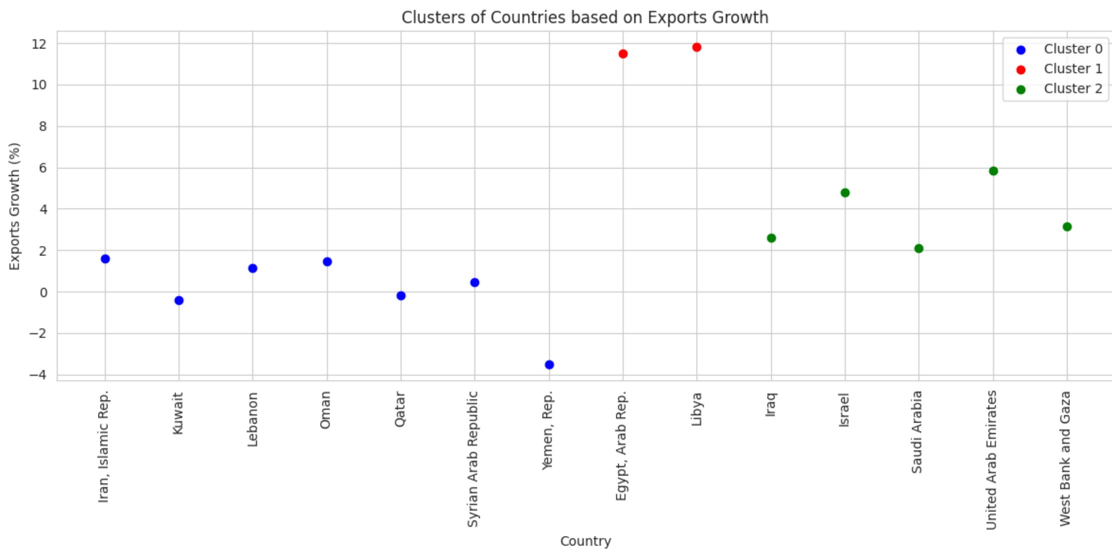
[ ]:
```
# Group data by country and calculate the average value for 'Exports of goods␣
↪and services (annual % growth)'
avg_exports_growth = df.groupby('Country Name')['Exports of goods and services␣
↪(annual % growth)'].mean().reset_index()

# Perform clustering
from sklearn.cluster import KMeans

# Select the feature for clustering
X = avg_exports_growth['Exports of goods and services (annual % growth)'].
↪values.reshape(-1, 1)

# Define the number of clusters
n_clusters = 3

# Fit KMeans clustering
kmeans = KMeans(n_clusters=n_clusters, random_state=42)
clusters = kmeans.fit_predict(X)

# Add the cluster labels to the DataFrame
avg_exports_growth['Cluster'] = clusters
```

```python
# Display the countries in each cluster
for cluster_num in range(n_clusters):
    countries_in_cluster = avg_exports_growth[avg_exports_growth['Cluster'] ==␣
↪cluster_num]['Country Name'].values
    print(f"Cluster {cluster_num}: {', '.join(countries_in_cluster)}")
```

```
Cluster 0: Iran, Islamic Rep., Kuwait, Lebanon, Oman, Qatar, Syrian Arab
Republic, Yemen, Rep.
Cluster 1: Egypt, Arab Rep., Libya
Cluster 2: Iraq, Israel, Saudi Arabia, United Arab Emirates, West Bank and Gaza
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```

[ ]:
```python
# Plotting the clusters
plt.figure(figsize=(12, 6))
colors = ['blue', 'red', 'green']
for cluster_num in range(n_clusters):
    countries_in_cluster = avg_exports_growth[avg_exports_growth['Cluster'] ==␣
↪cluster_num]
    plt.scatter(countries_in_cluster['Country Name'],␣
↪countries_in_cluster['Exports of goods and services (annual % growth)'],
                color=colors[cluster_num], label=f'Cluster {cluster_num}')
plt.title('Clusters of Countries based on Exports Growth')
plt.xlabel('Country')
plt.ylabel('Exports Growth (%)')
plt.xticks(rotation=90)
plt.legend()
plt.tight_layout()
plt.show()
```



[ ]:
```
india_data = pd.read_excel('transformed_data (2).xlsx')
west_asian_data = pd.read_excel('transformed_data (1).xlsx')

# Merge the two datasets based on the year column
merged_data = pd.merge(india_data, west_asian_data, on='Year',␣
↪suffixes=('_India', '_WestAsia'))

# Perform regression analysis
from sklearn.linear_model import LinearRegression

X = merged_data[[ 'Exports of goods and services (annual % growth)',
       'Food exports (% of merchandise exports)_WestAsia',
       'Fuel imports (% of merchandise imports)_WestAsia',
       'GDP growth (annual %)_WestAsia',
       'High-technology exports (% of manufactured exports)_WestAsia',
       'Inflation, GDP deflator (annual %)_WestAsia',
       'Inflation, consumer prices (annual %)_WestAsia',
       'Military expenditure (% of GDP)_WestAsia', 'Trade (% of GDP)_WestAsia',
       'Trade in services (% of GDP)_WestAsia']]
```

[ ]:
```
india_data.columns
```

[ ]:
```
Index(['Country Name', 'Year', 'Exports of goods and services (% of GDP)',
       'Food exports (% of merchandise exports)',
       'Fuel imports (% of merchandise imports)', 'GDP growth (annual %)',
       'High-technology exports (% of manufactured exports)',
       'Inflation, GDP deflator (annual %)',
       'Inflation, consumer prices (annual %)',
```

```
       'Military expenditure (% of GDP)', 'Trade (% of GDP)',
       'Trade in services (% of GDP)'],
      dtype='object')
```

[ ]:
```
from sklearn.preprocessing import StandardScaler
import pandas as pd

# Assuming merged_data is your DataFrame and the columns are as listed
features = ['Exports of goods and services (% of GDP)',
            'Fuel imports (% of merchandise imports)_India',
            'GDP growth (annual %)_India',
            'Inflation, GDP deflator (annual %)_India',
            'Inflation, consumer prices (annual %)_India']

# Extract the features into a new DataFrame
data_to_score = merged_data[features]

# Standardize the features
scaler = StandardScaler()
data_standardized = scaler.fit_transform(data_to_score)

# Create a composite score by taking the mean across the standardized features
# This assumes equal weighting of features
merged_data['composite_score'] = data_standardized.mean(axis=1)

# Now merged_data has a new column 'composite_score' which is the standardized␣
↪average of the selected features
```

[ ]:
```
# Define your custom weights for each feature
weights = {
    'Exports of goods and services (% of GDP)': 0.2,
    'Fuel imports (% of merchandise imports)_India': 0.2,
    'GDP growth (annual %)_India': 0.3,
    'Inflation, GDP deflator (annual %)_India': 0.15,
    'Inflation, consumer prices (annual %)_India': 0.15
}

# Apply weights to each column and sum to get the weighted score
merged_data['weighted_score'] = sum(data_standardized[:, i] * weights[feature]␣
↪for i, feature in enumerate(features))

# Now merged_data has a new column 'weighted_score' which is the weighted␣
↪average of the selected features
y = merged_data['weighted_score']
```

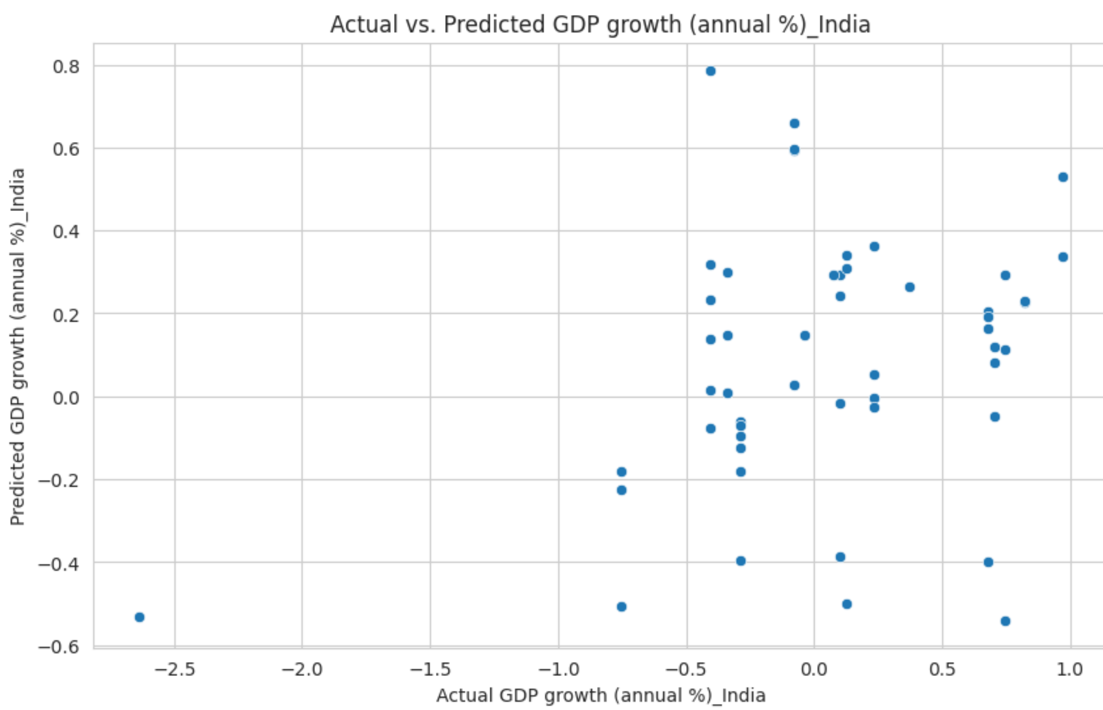
[ ]:
```
merged_data.columns
```

[ ]:
```
Index(['Country Name_India', 'Year',
       'Exports of goods and services (% of GDP)',
       'Food exports (% of merchandise exports)_India',
       'Fuel imports (% of merchandise imports)_India',
       'GDP growth (annual %)_India',
       'High-technology exports (% of manufactured exports)_India',
       'Inflation, GDP deflator (annual %)_India',
       'Inflation, consumer prices (annual %)_India',
       'Military expenditure (% of GDP)_India', 'Trade (% of GDP)_India',
       'Trade in services (% of GDP)_India', 'Country Name_WestAsia',
       'Exports of goods and services (annual % growth)',
       'Food exports (% of merchandise exports)_WestAsia',
       'Fuel imports (% of merchandise imports)_WestAsia',
       'GDP growth (annual %)_WestAsia',
       'High-technology exports (% of manufactured exports)_WestAsia',
       'Inflation, GDP deflator (annual %)_WestAsia',
       'Inflation, consumer prices (annual %)_WestAsia',
       'Military expenditure (% of GDP)_WestAsia', 'Trade (% of GDP)_WestAsia',
       'Trade in services (% of GDP)_WestAsia', 'composite_score',
       'weighted_score'],
      dtype='object')
```

[ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error
# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Fit the model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Calculate R-squared and RMSE
r2 = r2_score(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)

print("R-squared:", r2)
print("RMSE:", rmse)

# Plot the predictions against the actual values
```

```
plt.figure(figsize=(10, 6))
sns.scatterplot(x=y_test, y=y_pred)
plt.xlabel('Actual GDP growth (annual %)_India')
plt.ylabel('Predicted GDP growth (annual %)_India')
plt.title('Actual vs. Predicted GDP growth (annual %)_India')
plt.show()

# Get model summary
print("Coefficients:", model.coef_)
print("Intercept:", model.intercept_)
```

```
R-squared: 0.04464054699629516
RMSE: 0.5979636602836522
```




```
Coefficients: [-0.00262082  0.00043487  0.00270506  0.00051244 -0.02773105
-0.01789175
  0.00812982 -0.00433155  0.00267284 -0.00010062]
Intercept: -0.028594864479177244
```

The Linear Regression model demonstrates a modest fit with an **R-squared** of **0.0446** and a **RMSE** of **0.598**, indicating room for improvement in predictive accuracy. The **R^2 Score** of **0.1221** on the entire dataset suggests that the model explains a small portion of the variance in the data.

```
[ ]: # Calculate R^2 score
r_squared = model.score(X, y)
print("R^2 Score:", r_squared)
```

```
R^2 Score: 0.12210827866227236
```

```
[ ]: import numpy as np
import pandas as pd
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Bidirectional

# Assuming X_train, y_train, X_test, y_test are your prepared datasets

# Reshape the data into 3D arrays (samples, time steps, features)
X_train = np.reshape(X_train, (X_train.shape[0], 1, X_train.shape[1]))
X_test = np.reshape(X_test, (X_test.shape[0], 1, X_test.shape[1]))

# Build LSTM model
model_lstm = Sequential()
model_lstm.add(LSTM(50, input_shape=(X_train.shape[1], X_train.shape[2])))
model_lstm.add(Dense(1))
model_lstm.compile(optimizer='adam', loss='mse')

# Build BiLSTM model
model_bilstm = Sequential()
model_bilstm.add(Bidirectional(LSTM(50, input_shape=(X_train.shape[1], X_train.
 ↪shape[2]))))
model_bilstm.add(Dense(1))
model_bilstm.compile(optimizer='adam', loss='mse')

# Train LSTM model
history_lstm = model_lstm.fit(X_train, y_train, epochs=100, batch_size=32,␣
 ↪validation_data=(X_test, y_test), verbose=1)

# Train BiLSTM model
history_bilstm = model_bilstm.fit(X_train, y_train, epochs=100, batch_size=32,␣
 ↪validation_data=(X_test, y_test), verbose=1)

# Evaluate LSTM model
lstm_loss = model_lstm.evaluate(X_test, y_test, verbose=0)
print("LSTM Loss:", lstm_loss)

# Evaluate BiLSTM model
bilstm_loss = model_bilstm.evaluate(X_test, y_test, verbose=0)
print("BiLSTM Loss:", bilstm_loss)

# Make predictions using LSTM and BiLSTM models
```

```
y_pred_lstm = model_lstm.predict(X_test)
y_pred_bilstm = model_bilstm.predict(X_test)

# Plot predictions vs. actual
plt.figure(figsize=(10, 6))
plt.plot(y_test, label='Actual')
plt.plot(y_pred_lstm, label='LSTM Predictions')
plt.plot(y_pred_bilstm, label='BiLSTM Predictions')
plt.legend()
plt.show()
```

```
Epoch 1/100
7/7 [==============================] - 3s 102ms/step - loss: 0.9265 - val_loss:
0.5617
Epoch 2/100
7/7 [==============================] - 0s 8ms/step - loss: 0.7490 - val_loss:
0.4566
Epoch 3/100
7/7 [==============================] - 0s 11ms/step - loss: 0.6735 - val_loss:
0.4206
Epoch 4/100
7/7 [==============================] - 0s 8ms/step - loss: 0.6314 - val_loss:
0.4007
Epoch 5/100
7/7 [==============================] - 0s 9ms/step - loss: 0.5974 - val_loss:
0.3845
Epoch 6/100
7/7 [==============================] - 0s 12ms/step - loss: 0.5674 - val_loss:
0.3659
Epoch 7/100
7/7 [==============================] - 0s 10ms/step - loss: 0.5395 - val_loss:
0.3571
Epoch 8/100
7/7 [==============================] - 0s 9ms/step - loss: 0.5159 - val_loss:
0.3507
Epoch 9/100
7/7 [==============================] - 0s 9ms/step - loss: 0.4974 - val_loss:
0.3433
Epoch 10/100
7/7 [==============================] - 0s 11ms/step - loss: 0.4821 - val_loss:
0.3417
Epoch 11/100
7/7 [==============================] - 0s 9ms/step - loss: 0.4679 - val_loss:
0.3454
Epoch 12/100
7/7 [==============================] - 0s 11ms/step - loss: 0.4525 - val_loss:
0.3418
```

```
Epoch 13/100
7/7 [==============================] - 0s 9ms/step - loss: 0.4416 - val_loss:
0.3378
Epoch 14/100
7/7 [==============================] - 0s 9ms/step - loss: 0.4319 - val_loss:
0.3350
Epoch 15/100
7/7 [==============================] - 0s 11ms/step - loss: 0.4196 - val_loss:
0.3423
Epoch 16/100
7/7 [==============================] - 0s 11ms/step - loss: 0.4061 - val_loss:
0.3312
Epoch 17/100
7/7 [==============================] - 0s 12ms/step - loss: 0.3922 - val_loss:
0.3272
Epoch 18/100
7/7 [==============================] - 0s 8ms/step - loss: 0.3832 - val_loss:
0.3268
Epoch 19/100
7/7 [==============================] - 0s 9ms/step - loss: 0.3717 - val_loss:
0.3321
Epoch 20/100
7/7 [==============================] - 0s 9ms/step - loss: 0.3617 - val_loss:
0.3329
Epoch 21/100
7/7 [==============================] - 0s 11ms/step - loss: 0.3566 - val_loss:
0.3362
Epoch 22/100
7/7 [==============================] - 0s 11ms/step - loss: 0.3466 - val_loss:
0.3336
Epoch 23/100
7/7 [==============================] - 0s 11ms/step - loss: 0.3401 - val_loss:
0.3329
Epoch 24/100
7/7 [==============================] - 0s 9ms/step - loss: 0.3322 - val_loss:
0.3384
Epoch 25/100
7/7 [==============================] - 0s 9ms/step - loss: 0.3242 - val_loss:
0.3373
Epoch 26/100
7/7 [==============================] - 0s 8ms/step - loss: 0.3192 - val_loss:
0.3392
Epoch 27/100
7/7 [==============================] - 0s 10ms/step - loss: 0.3127 - val_loss:
0.3421
Epoch 28/100
7/7 [==============================] - 0s 11ms/step - loss: 0.3051 - val_loss:
0.3674
```

```
Epoch 29/100
7/7 [==============================] - 0s 9ms/step - loss: 0.3026 - val_loss:
0.3585
Epoch 30/100
7/7 [==============================] - 0s 11ms/step - loss: 0.2983 - val_loss:
0.3407
Epoch 31/100
7/7 [==============================] - 0s 11ms/step - loss: 0.2885 - val_loss:
0.3515
Epoch 32/100
7/7 [==============================] - 0s 8ms/step - loss: 0.2810 - val_loss:
0.3453
Epoch 33/100
7/7 [==============================] - 0s 9ms/step - loss: 0.2784 - val_loss:
0.3422
Epoch 34/100
7/7 [==============================] - 0s 11ms/step - loss: 0.2768 - val_loss:
0.3374
Epoch 35/100
7/7 [==============================] - 0s 12ms/step - loss: 0.2667 - val_loss:
0.3760
Epoch 36/100
7/7 [==============================] - 0s 10ms/step - loss: 0.2640 - val_loss:
0.3664
Epoch 37/100
7/7 [==============================] - 0s 11ms/step - loss: 0.2558 - val_loss:
0.3624
Epoch 38/100
7/7 [==============================] - 0s 11ms/step - loss: 0.2503 - val_loss:
0.3704
Epoch 39/100
7/7 [==============================] - 0s 10ms/step - loss: 0.2478 - val_loss:
0.3667
Epoch 40/100
7/7 [==============================] - 0s 11ms/step - loss: 0.2377 - val_loss:
0.4081
Epoch 41/100
7/7 [==============================] - 0s 12ms/step - loss: 0.2379 - val_loss:
0.4040
Epoch 42/100
7/7 [==============================] - 0s 8ms/step - loss: 0.2321 - val_loss:
0.3917
Epoch 43/100
7/7 [==============================] - 0s 12ms/step - loss: 0.2224 - val_loss:
0.4017
Epoch 44/100
7/7 [==============================] - 0s 9ms/step - loss: 0.2166 - val_loss:
0.3910
```

```
Epoch 45/100
7/7 [==============================] - 0s 11ms/step - loss: 0.2125 - val_loss:
0.3897
Epoch 46/100
7/7 [==============================] - 0s 10ms/step - loss: 0.2084 - val_loss:
0.3939
Epoch 47/100
7/7 [==============================] - 0s 9ms/step - loss: 0.2038 - val_loss:
0.4008
Epoch 48/100
7/7 [==============================] - 0s 8ms/step - loss: 0.2007 - val_loss:
0.4011
Epoch 49/100
7/7 [==============================] - 0s 12ms/step - loss: 0.1957 - val_loss:
0.3875
Epoch 50/100
7/7 [==============================] - 0s 11ms/step - loss: 0.1943 - val_loss:
0.4225
Epoch 51/100
7/7 [==============================] - 0s 11ms/step - loss: 0.1885 - val_loss:
0.4156
Epoch 52/100
7/7 [==============================] - 0s 9ms/step - loss: 0.1848 - val_loss:
0.3902
Epoch 53/100
7/7 [==============================] - 0s 9ms/step - loss: 0.1819 - val_loss:
0.3955
Epoch 54/100
7/7 [==============================] - 0s 9ms/step - loss: 0.1782 - val_loss:
0.3998
Epoch 55/100
7/7 [==============================] - 0s 10ms/step - loss: 0.1727 - val_loss:
0.4024
Epoch 56/100
7/7 [==============================] - 0s 8ms/step - loss: 0.1689 - val_loss:
0.4049
Epoch 57/100
7/7 [==============================] - 0s 11ms/step - loss: 0.1649 - val_loss:
0.4256
Epoch 58/100
7/7 [==============================] - 0s 12ms/step - loss: 0.1623 - val_loss:
0.4037
Epoch 59/100
7/7 [==============================] - 0s 8ms/step - loss: 0.1581 - val_loss:
0.4004
Epoch 60/100
7/7 [==============================] - 0s 11ms/step - loss: 0.1547 - val_loss:
0.4056
```

```
Epoch 61/100
7/7 [==============================] - 0s 8ms/step - loss: 0.1521 - val_loss:
0.4035
Epoch 62/100
7/7 [==============================] - 0s 8ms/step - loss: 0.1489 - val_loss:
0.4044
Epoch 63/100
7/7 [==============================] - 0s 11ms/step - loss: 0.1461 - val_loss:
0.4246
Epoch 64/100
7/7 [==============================] - 0s 13ms/step - loss: 0.1501 - val_loss:
0.4138
Epoch 65/100
7/7 [==============================] - 0s 9ms/step - loss: 0.1426 - val_loss:
0.4643
Epoch 66/100
7/7 [==============================] - 0s 11ms/step - loss: 0.1453 - val_loss:
0.4400
Epoch 67/100
7/7 [==============================] - 0s 9ms/step - loss: 0.1365 - val_loss:
0.4272
Epoch 68/100
7/7 [==============================] - 0s 9ms/step - loss: 0.1337 - val_loss:
0.4389
Epoch 69/100
7/7 [==============================] - 0s 8ms/step - loss: 0.1328 - val_loss:
0.4567
Epoch 70/100
7/7 [==============================] - 0s 13ms/step - loss: 0.1294 - val_loss:
0.4546
Epoch 71/100
7/7 [==============================] - 0s 9ms/step - loss: 0.1267 - val_loss:
0.4485
Epoch 72/100
7/7 [==============================] - 0s 11ms/step - loss: 0.1252 - val_loss:
0.4454
Epoch 73/100
7/7 [==============================] - 0s 11ms/step - loss: 0.1231 - val_loss:
0.4429
Epoch 74/100
7/7 [==============================] - 0s 9ms/step - loss: 0.1201 - val_loss:
0.4669
Epoch 75/100
7/7 [==============================] - 0s 12ms/step - loss: 0.1189 - val_loss:
0.4555
Epoch 76/100
7/7 [==============================] - 0s 9ms/step - loss: 0.1178 - val_loss:
0.4623
```

```
Epoch 77/100
7/7 [==============================] - 0s 9ms/step - loss: 0.1137 - val_loss:
0.4981
Epoch 78/100
7/7 [==============================] - 0s 14ms/step - loss: 0.1188 - val_loss:
0.4746
Epoch 79/100
7/7 [==============================] - 0s 9ms/step - loss: 0.1123 - val_loss:
0.4970
Epoch 80/100
7/7 [==============================] - 0s 12ms/step - loss: 0.1124 - val_loss:
0.5134
Epoch 81/100
7/7 [==============================] - 0s 11ms/step - loss: 0.1114 - val_loss:
0.4749
Epoch 82/100
7/7 [==============================] - 0s 12ms/step - loss: 0.1101 - val_loss:
0.5156
Epoch 83/100
7/7 [==============================] - 0s 9ms/step - loss: 0.1052 - val_loss:
0.4837
Epoch 84/100
7/7 [==============================] - 0s 11ms/step - loss: 0.1034 - val_loss:
0.4958
Epoch 85/100
7/7 [==============================] - 0s 9ms/step - loss: 0.1030 - val_loss:
0.4894
Epoch 86/100
7/7 [==============================] - 0s 11ms/step - loss: 0.0989 - val_loss:
0.5112
Epoch 87/100
7/7 [==============================] - 0s 9ms/step - loss: 0.0997 - val_loss:
0.5186
Epoch 88/100
7/7 [==============================] - 0s 11ms/step - loss: 0.0979 - val_loss:
0.5011
Epoch 89/100
7/7 [==============================] - 0s 9ms/step - loss: 0.0968 - val_loss:
0.5116
Epoch 90/100
7/7 [==============================] - 0s 12ms/step - loss: 0.0963 - val_loss:
0.5098
Epoch 91/100
7/7 [==============================] - 0s 10ms/step - loss: 0.0922 - val_loss:
0.5151
Epoch 92/100
7/7 [==============================] - 0s 10ms/step - loss: 0.0906 - val_loss:
0.5159
```

```
Epoch 93/100
7/7 [==============================] - 0s 11ms/step - loss: 0.0952 - val_loss:
0.5396
Epoch 94/100
7/7 [==============================] - 0s 9ms/step - loss: 0.0958 - val_loss:
0.5438
Epoch 95/100
7/7 [==============================] - 0s 11ms/step - loss: 0.0894 - val_loss:
0.5274
Epoch 96/100
7/7 [==============================] - 0s 9ms/step - loss: 0.0880 - val_loss:
0.5510
Epoch 97/100
7/7 [==============================] - 0s 8ms/step - loss: 0.0877 - val_loss:
0.5463
Epoch 98/100
7/7 [==============================] - 0s 11ms/step - loss: 0.0879 - val_loss:
0.5252
Epoch 99/100
7/7 [==============================] - 0s 9ms/step - loss: 0.0841 - val_loss:
0.5570
Epoch 100/100
7/7 [==============================] - 0s 11ms/step - loss: 0.0820 - val_loss:
0.5376
Epoch 1/100
7/7 [==============================] - 6s 157ms/step - loss: 0.9102 - val_loss:
0.4865
Epoch 2/100
7/7 [==============================] - 0s 12ms/step - loss: 0.7198 - val_loss:
0.4059
Epoch 3/100
7/7 [==============================] - 0s 12ms/step - loss: 0.6747 - val_loss:
0.3776
Epoch 4/100
7/7 [==============================] - 0s 10ms/step - loss: 0.6316 - val_loss:
0.3615
Epoch 5/100
7/7 [==============================] - 0s 13ms/step - loss: 0.5838 - val_loss:
0.3468
Epoch 6/100
7/7 [==============================] - 0s 10ms/step - loss: 0.5569 - val_loss:
0.3353
Epoch 7/100
7/7 [==============================] - 0s 13ms/step - loss: 0.5266 - val_loss:
0.3135
Epoch 8/100
7/7 [==============================] - 0s 12ms/step - loss: 0.5015 - val_loss:
0.3022
```

```
Epoch 9/100
7/7 [==============================] - 0s 10ms/step - loss: 0.4822 - val_loss:
0.2901
Epoch 10/100
7/7 [==============================] - 0s 10ms/step - loss: 0.4640 - val_loss:
0.2825
Epoch 11/100
7/7 [==============================] - 0s 10ms/step - loss: 0.4469 - val_loss:
0.2710
Epoch 12/100
7/7 [==============================] - 0s 11ms/step - loss: 0.4308 - val_loss:
0.2649
Epoch 13/100
7/7 [==============================] - 0s 11ms/step - loss: 0.4165 - val_loss:
0.2562
Epoch 14/100
7/7 [==============================] - 0s 13ms/step - loss: 0.4079 - val_loss:
0.2557
Epoch 15/100
7/7 [==============================] - 0s 12ms/step - loss: 0.3954 - val_loss:
0.2529
Epoch 16/100
7/7 [==============================] - 0s 13ms/step - loss: 0.3856 - val_loss:
0.2534
Epoch 17/100
7/7 [==============================] - 0s 12ms/step - loss: 0.3763 - val_loss:
0.2541
Epoch 18/100
7/7 [==============================] - 0s 12ms/step - loss: 0.3646 - val_loss:
0.2673
Epoch 19/100
7/7 [==============================] - 0s 13ms/step - loss: 0.3577 - val_loss:
0.2660
Epoch 20/100
7/7 [==============================] - 0s 13ms/step - loss: 0.3508 - val_loss:
0.2568
Epoch 21/100
7/7 [==============================] - 0s 12ms/step - loss: 0.3400 - val_loss:
0.2696
Epoch 22/100
7/7 [==============================] - 0s 12ms/step - loss: 0.3365 - val_loss:
0.2751
Epoch 23/100
7/7 [==============================] - 0s 9ms/step - loss: 0.3264 - val_loss:
0.2632
Epoch 24/100
7/7 [==============================] - 0s 10ms/step - loss: 0.3177 - val_loss:
0.2678
```

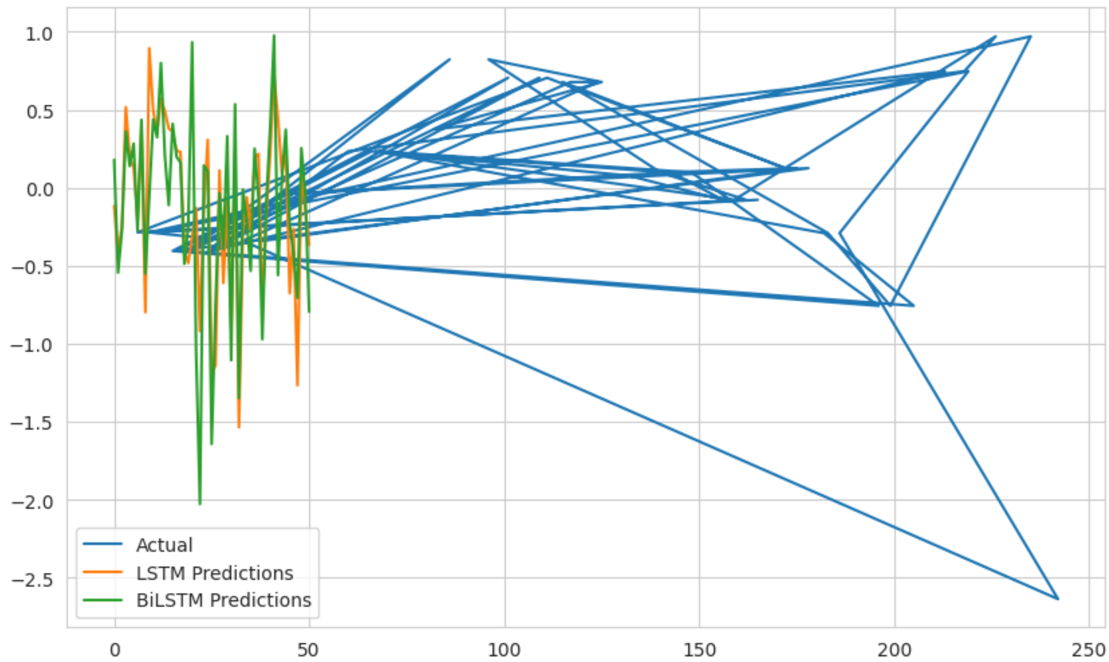
```
Epoch 25/100
7/7 [==============================] - 0s 13ms/step - loss: 0.3116 - val_loss:
0.2674
Epoch 26/100
7/7 [==============================] - 0s 12ms/step - loss: 0.3039 - val_loss:
0.2647
Epoch 27/100
7/7 [==============================] - 0s 10ms/step - loss: 0.2962 - val_loss:
0.2731
Epoch 28/100
7/7 [==============================] - 0s 11ms/step - loss: 0.2893 - val_loss:
0.2734
Epoch 29/100
7/7 [==============================] - 0s 13ms/step - loss: 0.2854 - val_loss:
0.2806
Epoch 30/100
7/7 [==============================] - 0s 12ms/step - loss: 0.2779 - val_loss:
0.2746
Epoch 31/100
7/7 [==============================] - 0s 13ms/step - loss: 0.2710 - val_loss:
0.2716
Epoch 32/100
7/7 [==============================] - 0s 10ms/step - loss: 0.2651 - val_loss:
0.2840
Epoch 33/100
7/7 [==============================] - 0s 12ms/step - loss: 0.2561 - val_loss:
0.2806
Epoch 34/100
7/7 [==============================] - 0s 12ms/step - loss: 0.2557 - val_loss:
0.2800
Epoch 35/100
7/7 [==============================] - 0s 10ms/step - loss: 0.2481 - val_loss:
0.3286
Epoch 36/100
7/7 [==============================] - 0s 10ms/step - loss: 0.2387 - val_loss:
0.3389
Epoch 37/100
7/7 [==============================] - 0s 10ms/step - loss: 0.2365 - val_loss:
0.3550
Epoch 38/100
7/7 [==============================] - 0s 10ms/step - loss: 0.2281 - val_loss:
0.3355
Epoch 39/100
7/7 [==============================] - 0s 12ms/step - loss: 0.2222 - val_loss:
0.3440
Epoch 40/100
7/7 [==============================] - 0s 12ms/step - loss: 0.2123 - val_loss:
0.3259
```

```
Epoch 41/100
7/7 [==============================] - 0s 13ms/step - loss: 0.2113 - val_loss:
0.3299
Epoch 42/100
7/7 [==============================] - 0s 10ms/step - loss: 0.2049 - val_loss:
0.3437
Epoch 43/100
7/7 [==============================] - 0s 15ms/step - loss: 0.2008 - val_loss:
0.3632
Epoch 44/100
7/7 [==============================] - 0s 10ms/step - loss: 0.1957 - val_loss:
0.3370
Epoch 45/100
7/7 [==============================] - 0s 9ms/step - loss: 0.1923 - val_loss:
0.3812
Epoch 46/100
7/7 [==============================] - 0s 12ms/step - loss: 0.1886 - val_loss:
0.3897
Epoch 47/100
7/7 [==============================] - 0s 10ms/step - loss: 0.1853 - val_loss:
0.3775
Epoch 48/100
7/7 [==============================] - 0s 13ms/step - loss: 0.1776 - val_loss:
0.3777
Epoch 49/100
7/7 [==============================] - 0s 10ms/step - loss: 0.1771 - val_loss:
0.3795
Epoch 50/100
7/7 [==============================] - 0s 10ms/step - loss: 0.1711 - val_loss:
0.3693
Epoch 51/100
7/7 [==============================] - 0s 10ms/step - loss: 0.1745 - val_loss:
0.3816
Epoch 52/100
7/7 [==============================] - 0s 12ms/step - loss: 0.1645 - val_loss:
0.4110
Epoch 53/100
7/7 [==============================] - 0s 10ms/step - loss: 0.1637 - val_loss:
0.3995
Epoch 54/100
7/7 [==============================] - 0s 13ms/step - loss: 0.1603 - val_loss:
0.4007
Epoch 55/100
7/7 [==============================] - 0s 10ms/step - loss: 0.1547 - val_loss:
0.4182
Epoch 56/100
7/7 [==============================] - 0s 15ms/step - loss: 0.1558 - val_loss:
0.4047
```

```
Epoch 57/100
7/7 [==============================] - 0s 14ms/step - loss: 0.1542 - val_loss:
0.3931
Epoch 58/100
7/7 [==============================] - 0s 10ms/step - loss: 0.1526 - val_loss:
0.4131
Epoch 59/100
7/7 [==============================] - 0s 10ms/step - loss: 0.1531 - val_loss:
0.3906
Epoch 60/100
7/7 [==============================] - 0s 10ms/step - loss: 0.1471 - val_loss:
0.4260
Epoch 61/100
7/7 [==============================] - 0s 16ms/step - loss: 0.1433 - val_loss:
0.3991
Epoch 62/100
7/7 [==============================] - 0s 17ms/step - loss: 0.1401 - val_loss:
0.4098
Epoch 63/100
7/7 [==============================] - 0s 16ms/step - loss: 0.1336 - val_loss:
0.4404
Epoch 64/100
7/7 [==============================] - 0s 16ms/step - loss: 0.1338 - val_loss:
0.4341
Epoch 65/100
7/7 [==============================] - 0s 17ms/step - loss: 0.1314 - val_loss:
0.4481
Epoch 66/100
7/7 [==============================] - 0s 18ms/step - loss: 0.1288 - val_loss:
0.4421
Epoch 67/100
7/7 [==============================] - 0s 14ms/step - loss: 0.1248 - val_loss:
0.4340
Epoch 68/100
7/7 [==============================] - 0s 14ms/step - loss: 0.1273 - val_loss:
0.4491
Epoch 69/100
7/7 [==============================] - 0s 15ms/step - loss: 0.1374 - val_loss:
0.4488
Epoch 70/100
7/7 [==============================] - 0s 17ms/step - loss: 0.1240 - val_loss:
0.4215
Epoch 71/100
7/7 [==============================] - 0s 17ms/step - loss: 0.1225 - val_loss:
0.4765
Epoch 72/100
7/7 [==============================] - 0s 16ms/step - loss: 0.1217 - val_loss:
0.4167
```

```
Epoch 73/100
7/7 [==============================] - 0s 14ms/step - loss: 0.1241 - val_loss:
0.4226
Epoch 74/100
7/7 [==============================] - 0s 14ms/step - loss: 0.1114 - val_loss:
0.4700
Epoch 75/100
7/7 [==============================] - 0s 17ms/step - loss: 0.1113 - val_loss:
0.4533
Epoch 76/100
7/7 [==============================] - 0s 16ms/step - loss: 0.1091 - val_loss:
0.4657
Epoch 77/100
7/7 [==============================] - 0s 17ms/step - loss: 0.1061 - val_loss:
0.4805
Epoch 78/100
7/7 [==============================] - 0s 14ms/step - loss: 0.1053 - val_loss:
0.4565
Epoch 79/100
7/7 [==============================] - 0s 17ms/step - loss: 0.1040 - val_loss:
0.4760
Epoch 80/100
7/7 [==============================] - 0s 17ms/step - loss: 0.1037 - val_loss:
0.4809
Epoch 81/100
7/7 [==============================] - 0s 17ms/step - loss: 0.1046 - val_loss:
0.4631
Epoch 82/100
7/7 [==============================] - 0s 17ms/step - loss: 0.0997 - val_loss:
0.4737
Epoch 83/100
7/7 [==============================] - 0s 16ms/step - loss: 0.0982 - val_loss:
0.4427
Epoch 84/100
7/7 [==============================] - 0s 16ms/step - loss: 0.0955 - val_loss:
0.4906
Epoch 85/100
7/7 [==============================] - 0s 20ms/step - loss: 0.0982 - val_loss:
0.4727
Epoch 86/100
7/7 [==============================] - 0s 14ms/step - loss: 0.0924 - val_loss:
0.4955
Epoch 87/100
7/7 [==============================] - 0s 12ms/step - loss: 0.0921 - val_loss:
0.4627
Epoch 88/100
7/7 [==============================] - 0s 12ms/step - loss: 0.0890 - val_loss:
0.4835
```

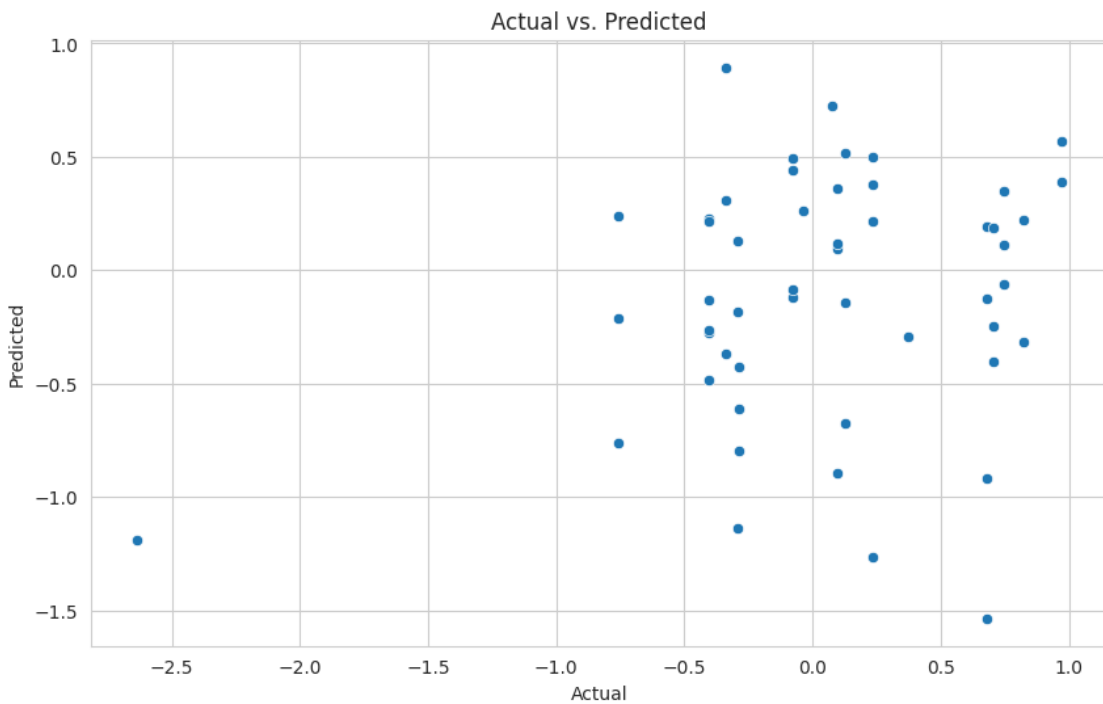
```
Epoch 89/100
7/7 [==============================] - 0s 10ms/step - loss: 0.0872 - val_loss:
0.4782
Epoch 90/100
7/7 [==============================] - 0s 12ms/step - loss: 0.0895 - val_loss:
0.4835
Epoch 91/100
7/7 [==============================] - 0s 13ms/step - loss: 0.0856 - val_loss:
0.4997
Epoch 92/100
7/7 [==============================] - 0s 10ms/step - loss: 0.0841 - val_loss:
0.5006
Epoch 93/100
7/7 [==============================] - 0s 12ms/step - loss: 0.0837 - val_loss:
0.4960
Epoch 94/100
7/7 [==============================] - 0s 10ms/step - loss: 0.0822 - val_loss:
0.5085
Epoch 95/100
7/7 [==============================] - 0s 10ms/step - loss: 0.0822 - val_loss:
0.5090
Epoch 96/100
7/7 [==============================] - 0s 10ms/step - loss: 0.0793 - val_loss:
0.4935
Epoch 97/100
7/7 [==============================] - 0s 13ms/step - loss: 0.0772 - val_loss:
0.4940
Epoch 98/100
7/7 [==============================] - 0s 12ms/step - loss: 0.0772 - val_loss:
0.5195
Epoch 99/100
7/7 [==============================] - 0s 13ms/step - loss: 0.0761 - val_loss:
0.5015
Epoch 100/100
7/7 [==============================] - 0s 10ms/step - loss: 0.0747 - val_loss:
0.4994
LSTM Loss: 0.5376214981079102
BiLSTM Loss: 0.49937736988067627
2/2 [==============================] - 0s 5ms/step
2/2 [==============================] - 1s 6ms/step
```



[ ]:
```python
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

def calculate_regression_metrics(y_true, y_pred):
    """
    Calculate and return various regression metrics.
    """
    mse = mean_squared_error(y_true, y_pred)
    rmse = mean_squared_error(y_true, y_pred, squared=False)
    mae = mean_absolute_error(y_true, y_pred)
    r2 = r2_score(y_true, y_pred)

    return {
        'Mean Squared Error (MSE)': mse,
        'Root Mean Squared Error (RMSE)': rmse,
        'Mean Absolute Error (MAE)': mae,
        'R-squared': r2
    }

def plot_actual_vs_predicted(y_true, y_pred):
    """
    Plot the actual vs. predicted values.
    """
    plt.figure(figsize=(10, 6))
    sns.scatterplot(x=y_true, y=y_pred)
    plt.xlabel('Actual')
```

```python
    plt.ylabel('Predicted')
    plt.title('Actual vs. Predicted')
    plt.show()

# Example usage
def evaluate_model(y_true, y_pred):
    metrics = calculate_regression_metrics(y_true, y_pred)
    print("LSTM Metrics:")
    for metric, value in metrics.items():
        print(f"{metric}: {value}")

    plot_actual_vs_predicted(y_true, y_pred)


y_pred_lstm_flat = y_pred_lstm.flatten()

# Assuming y_true is the actual values and y_pred_lstm_flat is the flattened␣
↪predicted values from your LSTM model
evaluate_model(y_test, y_pred_lstm_flat)
```

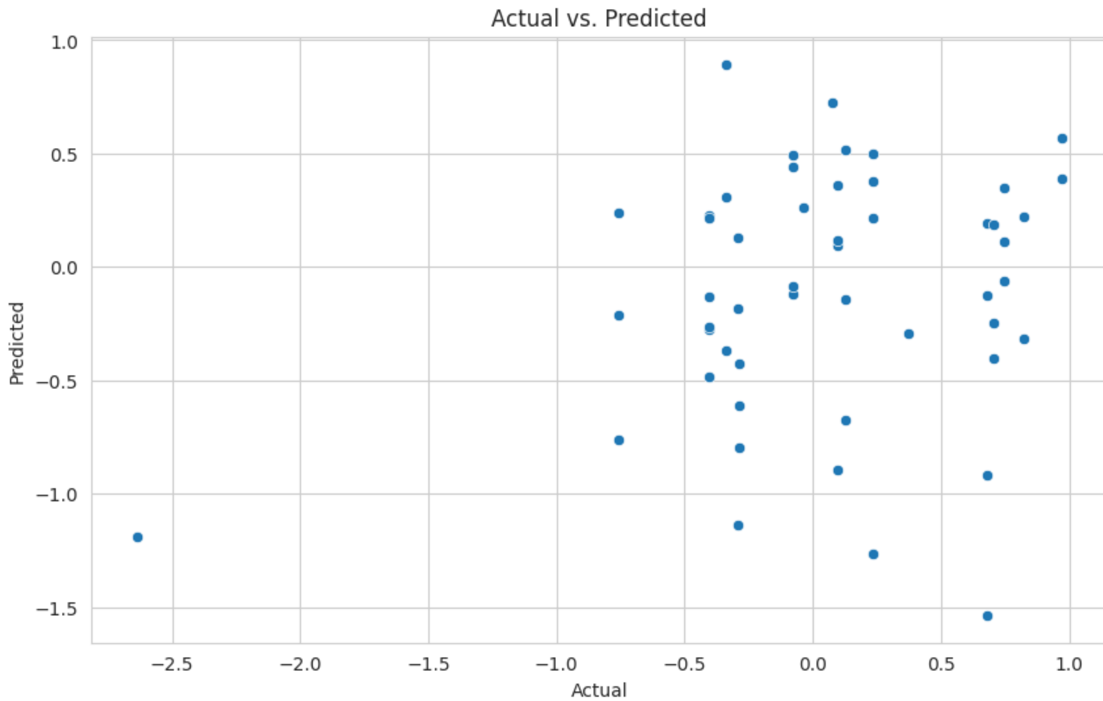
```
LSTM Metrics:
Mean Squared Error (MSE): 0.5376215368280253
Root Mean Squared Error (RMSE): 0.7332267976745157
Mean Absolute Error (MAE): 0.5642939971376032
R-squared: -0.4364611339803315
```



[ ]:

```python
# Modify the evaluate_model function to return the metrics
def evaluate_model(y_true, y_pred):
    metrics = calculate_regression_metrics(y_true, y_pred)
    print("Regression Metrics:")
    for metric, value in metrics.items():
        print(f"{metric}: {value}")

    plot_actual_vs_predicted(y_true, y_pred)
    return metrics

# Use the function to get the metrics
metrics = evaluate_model(y_test, y_pred_lstm_flat)

# Convert the metrics to a DataFrame
metrics_df = pd.DataFrame(list(metrics.items()), columns=['Metric', 'Value'])

# Create the barplot
plt.figure(figsize=(15, 6))
sns.barplot(x='Metric', y='Value', data=metrics_df)
plt.title('Regression Metrics')
plt.show()
```

```
Regression Metrics:
Mean Squared Error (MSE): 0.5376215368280253
Root Mean Squared Error (RMSE): 0.7332267976745157
Mean Absolute Error (MAE): 0.5642939971376032
R-squared: -0.4364611339803315
```



[ ]:
```
from tensorflow.keras.layers import GRU

# Build GRU model
model_gru = Sequential()
model_gru.add(GRU(50, input_shape=(X_train.shape[1], X_train.shape[2])))
model_gru.add(Dense(1))
model_gru.compile(optimizer='adam', loss='mse')
```

```
# Build BiGRU model
model_bigru = Sequential()
model_bigru.add(Bidirectional(GRU(50, input_shape=(X_train.shape[1], X_train.
↪shape[2]))))
model_bigru.add(Dense(1))
model_bigru.compile(optimizer='adam', loss='mse')

# Train GRU model
history_gru = model_gru.fit(X_train, y_train, epochs=100, batch_size=32,␣
↪validation_data=(X_test, y_test), verbose=1)

# Train BiGRU model
history_bigru = model_bigru.fit(X_train, y_train, epochs=100, batch_size=32,␣
↪validation_data=(X_test, y_test), verbose=1)

# Evaluate GRU model
gru_loss = model_gru.evaluate(X_test, y_test, verbose=0)
print("GRU Loss:", gru_loss)

# Evaluate BiGRU model
bigru_loss = model_bigru.evaluate(X_test, y_test, verbose=0)
print("BiGRU Loss:", bigru_loss)

# Make predictions using GRU and BiGRU models
y_pred_gru = model_gru.predict(X_test)
y_pred_bigru = model_bigru.predict(X_test)

# Plot predictions vs. actual
plt.figure(figsize=(15, 6))
plt.plot(y_test, label='Actual')
plt.plot(y_pred_gru, label='GRU Predictions')
plt.plot(y_pred_bigru, label='BiGRU Predictions')
plt.legend()
plt.show()
```

```
Epoch 1/100
7/7 [==============================] - 3s 89ms/step - loss: 1.5857 - val_loss:
0.7272
Epoch 2/100
7/7 [==============================] - 0s 9ms/step - loss: 0.9762 - val_loss:
0.5158
Epoch 3/100
7/7 [==============================] - 0s 11ms/step - loss: 0.7661 - val_loss:
0.4915
Epoch 4/100
7/7 [==============================] - 0s 9ms/step - loss: 0.6872 - val_loss:
```

```
0.4811
Epoch 5/100
7/7 [==============================] - 0s 10ms/step - loss: 0.6478 - val_loss:
0.4377
Epoch 6/100
7/7 [==============================] - 0s 9ms/step - loss: 0.6036 - val_loss:
0.3939
Epoch 7/100
7/7 [==============================] - 0s 9ms/step - loss: 0.5651 - val_loss:
0.3630
Epoch 8/100
7/7 [==============================] - 0s 11ms/step - loss: 0.5359 - val_loss:
0.3439
Epoch 9/100
7/7 [==============================] - 0s 11ms/step - loss: 0.5108 - val_loss:
0.3330
Epoch 10/100
7/7 [==============================] - 0s 9ms/step - loss: 0.4892 - val_loss:
0.3274
Epoch 11/100
7/7 [==============================] - 0s 11ms/step - loss: 0.4706 - val_loss:
0.3253
Epoch 12/100
7/7 [==============================] - 0s 10ms/step - loss: 0.4524 - val_loss:
0.3349
Epoch 13/100
7/7 [==============================] - 0s 11ms/step - loss: 0.4441 - val_loss:
0.3357
Epoch 14/100
7/7 [==============================] - 0s 9ms/step - loss: 0.4294 - val_loss:
0.3247
Epoch 15/100
7/7 [==============================] - 0s 11ms/step - loss: 0.4207 - val_loss:
0.3164
Epoch 16/100
7/7 [==============================] - 0s 12ms/step - loss: 0.4128 - val_loss:
0.3089
Epoch 17/100
7/7 [==============================] - 0s 11ms/step - loss: 0.4025 - val_loss:
0.3166
Epoch 18/100
7/7 [==============================] - 0s 9ms/step - loss: 0.3943 - val_loss:
0.3165
Epoch 19/100
7/7 [==============================] - 0s 12ms/step - loss: 0.3860 - val_loss:
0.3106
Epoch 20/100
7/7 [==============================] - 0s 10ms/step - loss: 0.3798 - val_loss:
```

```
0.3040
Epoch 21/100
7/7 [==============================] - 0s 9ms/step - loss: 0.3729 - val_loss:
0.3023
Epoch 22/100
7/7 [==============================] - 0s 8ms/step - loss: 0.3664 - val_loss:
0.3088
Epoch 23/100
7/7 [==============================] - 0s 8ms/step - loss: 0.3615 - val_loss:
0.3042
Epoch 24/100
7/7 [==============================] - 0s 9ms/step - loss: 0.3551 - val_loss:
0.2981
Epoch 25/100
7/7 [==============================] - 0s 9ms/step - loss: 0.3482 - val_loss:
0.3063
Epoch 26/100
7/7 [==============================] - 0s 11ms/step - loss: 0.3442 - val_loss:
0.3138
Epoch 27/100
7/7 [==============================] - 0s 8ms/step - loss: 0.3392 - val_loss:
0.3140
Epoch 28/100
7/7 [==============================] - 0s 9ms/step - loss: 0.3330 - val_loss:
0.3185
Epoch 29/100
7/7 [==============================] - 0s 9ms/step - loss: 0.3285 - val_loss:
0.3191
Epoch 30/100
7/7 [==============================] - 0s 9ms/step - loss: 0.3251 - val_loss:
0.3205
Epoch 31/100
7/7 [==============================] - 0s 11ms/step - loss: 0.3194 - val_loss:
0.3129
Epoch 32/100
7/7 [==============================] - 0s 11ms/step - loss: 0.3149 - val_loss:
0.3006
Epoch 33/100
7/7 [==============================] - 0s 11ms/step - loss: 0.3117 - val_loss:
0.3024
Epoch 34/100
7/7 [==============================] - 0s 9ms/step - loss: 0.3067 - val_loss:
0.2923
Epoch 35/100
7/7 [==============================] - 0s 9ms/step - loss: 0.3049 - val_loss:
0.2964
Epoch 36/100
7/7 [==============================] - 0s 9ms/step - loss: 0.2978 - val_loss:
```

```
0.3139
Epoch 37/100
7/7 [==============================] - 0s 9ms/step - loss: 0.2962 - val_loss:
0.3201
Epoch 38/100
7/7 [==============================] - 0s 12ms/step - loss: 0.2912 - val_loss:
0.3083
Epoch 39/100
7/7 [==============================] - 0s 15ms/step - loss: 0.2859 - val_loss:
0.3116
Epoch 40/100
7/7 [==============================] - 0s 16ms/step - loss: 0.2852 - val_loss:
0.3120
Epoch 41/100
7/7 [==============================] - 0s 15ms/step - loss: 0.2849 - val_loss:
0.3322
Epoch 42/100
7/7 [==============================] - 0s 14ms/step - loss: 0.2758 - val_loss:
0.3104
Epoch 43/100
7/7 [==============================] - 0s 11ms/step - loss: 0.2727 - val_loss:
0.3029
Epoch 44/100
7/7 [==============================] - 0s 15ms/step - loss: 0.2701 - val_loss:
0.2969
Epoch 45/100
7/7 [==============================] - 0s 14ms/step - loss: 0.2650 - val_loss:
0.2990
Epoch 46/100
7/7 [==============================] - 0s 11ms/step - loss: 0.2633 - val_loss:
0.3023
Epoch 47/100
7/7 [==============================] - 0s 15ms/step - loss: 0.2615 - val_loss:
0.2951
Epoch 48/100
7/7 [==============================] - 0s 13ms/step - loss: 0.2563 - val_loss:
0.3090
Epoch 49/100
7/7 [==============================] - 0s 14ms/step - loss: 0.2536 - val_loss:
0.3149
Epoch 50/100
7/7 [==============================] - 0s 11ms/step - loss: 0.2497 - val_loss:
0.3202
Epoch 51/100
7/7 [==============================] - 0s 17ms/step - loss: 0.2483 - val_loss:
0.3117
Epoch 52/100
7/7 [==============================] - 0s 11ms/step - loss: 0.2442 - val_loss:
```

```
0.3143
Epoch 53/100
7/7 [==============================] - 0s 13ms/step - loss: 0.2413 - val_loss:
0.3029
Epoch 54/100
7/7 [==============================] - 0s 11ms/step - loss: 0.2386 - val_loss:
0.3100
Epoch 55/100
7/7 [==============================] - 0s 15ms/step - loss: 0.2392 - val_loss:
0.3143
Epoch 56/100
7/7 [==============================] - 0s 13ms/step - loss: 0.2354 - val_loss:
0.3264
Epoch 57/100
7/7 [==============================] - 0s 15ms/step - loss: 0.2309 - val_loss:
0.3056
Epoch 58/100
7/7 [==============================] - 0s 15ms/step - loss: 0.2275 - val_loss:
0.3117
Epoch 59/100
7/7 [==============================] - 0s 15ms/step - loss: 0.2249 - val_loss:
0.3155
Epoch 60/100
7/7 [==============================] - 0s 15ms/step - loss: 0.2251 - val_loss:
0.3249
Epoch 61/100
7/7 [==============================] - 0s 15ms/step - loss: 0.2224 - val_loss:
0.3059
Epoch 62/100
7/7 [==============================] - 0s 17ms/step - loss: 0.2175 - val_loss:
0.3152
Epoch 63/100
7/7 [==============================] - 0s 14ms/step - loss: 0.2158 - val_loss:
0.3220
Epoch 64/100
7/7 [==============================] - 0s 15ms/step - loss: 0.2129 - val_loss:
0.3043
Epoch 65/100
7/7 [==============================] - 0s 14ms/step - loss: 0.2111 - val_loss:
0.3246
Epoch 66/100
7/7 [==============================] - 0s 15ms/step - loss: 0.2112 - val_loss:
0.3433
Epoch 67/100
7/7 [==============================] - 0s 8ms/step - loss: 0.2050 - val_loss:
0.3133
Epoch 68/100
7/7 [==============================] - 0s 9ms/step - loss: 0.2030 - val_loss:
```

```
0.3165
Epoch 69/100
7/7 [==============================] - 0s 11ms/step - loss: 0.2010 - val_loss:
0.3301
Epoch 70/100
7/7 [==============================] - 0s 9ms/step - loss: 0.1983 - val_loss:
0.3202
Epoch 71/100
7/7 [==============================] - 0s 11ms/step - loss: 0.1936 - val_loss:
0.3105
Epoch 72/100
7/7 [==============================] - 0s 12ms/step - loss: 0.1943 - val_loss:
0.3117
Epoch 73/100
7/7 [==============================] - 0s 12ms/step - loss: 0.1902 - val_loss:
0.3368
Epoch 74/100
7/7 [==============================] - 0s 12ms/step - loss: 0.1921 - val_loss:
0.3296
Epoch 75/100
7/7 [==============================] - 0s 9ms/step - loss: 0.1866 - val_loss:
0.3372
Epoch 76/100
7/7 [==============================] - 0s 12ms/step - loss: 0.1857 - val_loss:
0.3290
Epoch 77/100
7/7 [==============================] - 0s 9ms/step - loss: 0.1828 - val_loss:
0.3300
Epoch 78/100
7/7 [==============================] - 0s 12ms/step - loss: 0.1817 - val_loss:
0.3356
Epoch 79/100
7/7 [==============================] - 0s 11ms/step - loss: 0.1790 - val_loss:
0.3367
Epoch 80/100
7/7 [==============================] - 0s 9ms/step - loss: 0.1777 - val_loss:
0.3378
Epoch 81/100
7/7 [==============================] - 0s 12ms/step - loss: 0.1739 - val_loss:
0.3417
Epoch 82/100
7/7 [==============================] - 0s 9ms/step - loss: 0.1740 - val_loss:
0.3408
Epoch 83/100
7/7 [==============================] - 0s 9ms/step - loss: 0.1736 - val_loss:
0.3269
Epoch 84/100
7/7 [==============================] - 0s 9ms/step - loss: 0.1742 - val_loss:
```

```
0.3834
Epoch 85/100
7/7 [==============================] - 0s 12ms/step - loss: 0.1713 - val_loss:
0.3597
Epoch 86/100
7/7 [==============================] - 0s 10ms/step - loss: 0.1673 - val_loss:
0.3411
Epoch 87/100
7/7 [==============================] - 0s 9ms/step - loss: 0.1676 - val_loss:
0.3474
Epoch 88/100
7/7 [==============================] - 0s 11ms/step - loss: 0.1619 - val_loss:
0.3248
Epoch 89/100
7/7 [==============================] - 0s 9ms/step - loss: 0.1618 - val_loss:
0.3278
Epoch 90/100
7/7 [==============================] - 0s 9ms/step - loss: 0.1581 - val_loss:
0.3470
Epoch 91/100
7/7 [==============================] - 0s 9ms/step - loss: 0.1582 - val_loss:
0.3486
Epoch 92/100
7/7 [==============================] - 0s 10ms/step - loss: 0.1600 - val_loss:
0.3454
Epoch 93/100
7/7 [==============================] - 0s 9ms/step - loss: 0.1534 - val_loss:
0.3528
Epoch 94/100
7/7 [==============================] - 0s 9ms/step - loss: 0.1521 - val_loss:
0.3459
Epoch 95/100
7/7 [==============================] - 0s 11ms/step - loss: 0.1519 - val_loss:
0.3618
Epoch 96/100
7/7 [==============================] - 0s 9ms/step - loss: 0.1497 - val_loss:
0.3520
Epoch 97/100
7/7 [==============================] - 0s 9ms/step - loss: 0.1487 - val_loss:
0.3471
Epoch 98/100
7/7 [==============================] - 0s 9ms/step - loss: 0.1483 - val_loss:
0.3504
Epoch 99/100
7/7 [==============================] - 0s 9ms/step - loss: 0.1465 - val_loss:
0.3596
Epoch 100/100
7/7 [==============================] - 0s 12ms/step - loss: 0.1444 - val_loss:
```

```
0.3694
Epoch 1/100
7/7 [==============================] - 5s 148ms/step - loss: 2.0028 - val_loss:
0.5859
Epoch 2/100
7/7 [==============================] - 0s 10ms/step - loss: 0.9073 - val_loss:
0.5746
Epoch 3/100
7/7 [==============================] - 0s 10ms/step - loss: 0.8248 - val_loss:
0.4978
Epoch 4/100
7/7 [==============================] - 0s 10ms/step - loss: 0.6905 - val_loss:
0.3378
Epoch 5/100
7/7 [==============================] - 0s 9ms/step - loss: 0.6073 - val_loss:
0.3116
Epoch 6/100
7/7 [==============================] - 0s 12ms/step - loss: 0.5759 - val_loss:
0.2963
Epoch 7/100
7/7 [==============================] - 0s 12ms/step - loss: 0.5336 - val_loss:
0.2951
Epoch 8/100
7/7 [==============================] - 0s 12ms/step - loss: 0.5037 - val_loss:
0.2983
Epoch 9/100
7/7 [==============================] - 0s 10ms/step - loss: 0.4777 - val_loss:
0.2829
Epoch 10/100
7/7 [==============================] - 0s 11ms/step - loss: 0.4542 - val_loss:
0.2769
Epoch 11/100
7/7 [==============================] - 0s 13ms/step - loss: 0.4343 - val_loss:
0.2682
Epoch 12/100
7/7 [==============================] - 0s 11ms/step - loss: 0.4190 - val_loss:
0.2592
Epoch 13/100
7/7 [==============================] - 0s 12ms/step - loss: 0.4030 - val_loss:
0.2629
Epoch 14/100
7/7 [==============================] - 0s 10ms/step - loss: 0.3862 - val_loss:
0.2749
Epoch 15/100
7/7 [==============================] - 0s 12ms/step - loss: 0.3730 - val_loss:
0.2809
Epoch 16/100
7/7 [==============================] - 0s 10ms/step - loss: 0.3602 - val_loss:
```

```
0.2836
Epoch 17/100
7/7 [==============================] - 0s 12ms/step - loss: 0.3472 - val_loss:
0.2791
Epoch 18/100
7/7 [==============================] - 0s 10ms/step - loss: 0.3375 - val_loss:
0.2797
Epoch 19/100
7/7 [==============================] - 0s 12ms/step - loss: 0.3259 - val_loss:
0.2805
Epoch 20/100
7/7 [==============================] - 0s 10ms/step - loss: 0.3169 - val_loss:
0.2812
Epoch 21/100
7/7 [==============================] - 0s 13ms/step - loss: 0.3082 - val_loss:
0.2988
Epoch 22/100
7/7 [==============================] - 0s 10ms/step - loss: 0.3046 - val_loss:
0.3006
Epoch 23/100
7/7 [==============================] - 0s 12ms/step - loss: 0.2934 - val_loss:
0.2870
Epoch 24/100
7/7 [==============================] - 0s 10ms/step - loss: 0.2876 - val_loss:
0.2838
Epoch 25/100
7/7 [==============================] - 0s 10ms/step - loss: 0.2810 - val_loss:
0.2935
Epoch 26/100
7/7 [==============================] - 0s 11ms/step - loss: 0.2756 - val_loss:
0.2954
Epoch 27/100
7/7 [==============================] - 0s 10ms/step - loss: 0.2692 - val_loss:
0.2781
Epoch 28/100
7/7 [==============================] - 0s 14ms/step - loss: 0.2649 - val_loss:
0.2887
Epoch 29/100
7/7 [==============================] - 0s 16ms/step - loss: 0.2594 - val_loss:
0.3010
Epoch 30/100
7/7 [==============================] - 0s 14ms/step - loss: 0.2560 - val_loss:
0.2873
Epoch 31/100
7/7 [==============================] - 0s 17ms/step - loss: 0.2506 - val_loss:
0.3133
Epoch 32/100
7/7 [==============================] - 0s 16ms/step - loss: 0.2487 - val_loss:
```

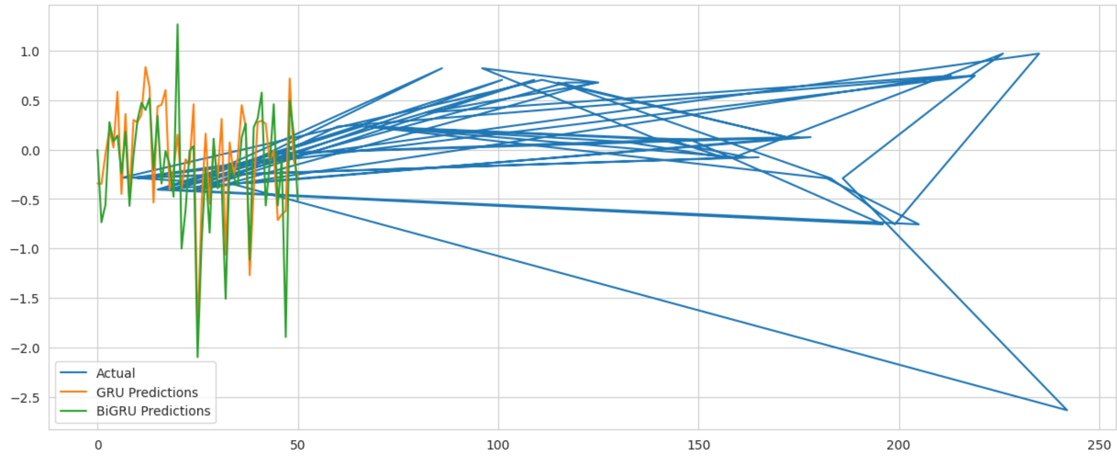
```
0.3205
Epoch 33/100
7/7 [==============================] - 0s 19ms/step - loss: 0.2423 - val_loss:
0.2996
Epoch 34/100
7/7 [==============================] - 0s 14ms/step - loss: 0.2405 - val_loss:
0.3106
Epoch 35/100
7/7 [==============================] - 0s 15ms/step - loss: 0.2333 - val_loss:
0.3115
Epoch 36/100
7/7 [==============================] - 0s 18ms/step - loss: 0.2301 - val_loss:
0.3160
Epoch 37/100
7/7 [==============================] - 0s 14ms/step - loss: 0.2268 - val_loss:
0.3255
Epoch 38/100
7/7 [==============================] - 0s 15ms/step - loss: 0.2291 - val_loss:
0.3446
Epoch 39/100
7/7 [==============================] - 0s 15ms/step - loss: 0.2187 - val_loss:
0.3198
Epoch 40/100
7/7 [==============================] - 0s 15ms/step - loss: 0.2179 - val_loss:
0.3359
Epoch 41/100
7/7 [==============================] - 0s 16ms/step - loss: 0.2122 - val_loss:
0.3188
Epoch 42/100
7/7 [==============================] - 0s 17ms/step - loss: 0.2096 - val_loss:
0.3269
Epoch 43/100
7/7 [==============================] - 0s 17ms/step - loss: 0.2017 - val_loss:
0.3501
Epoch 44/100
7/7 [==============================] - 0s 18ms/step - loss: 0.2017 - val_loss:
0.3399
Epoch 45/100
7/7 [==============================] - 0s 16ms/step - loss: 0.1998 - val_loss:
0.3616
Epoch 46/100
7/7 [==============================] - 0s 19ms/step - loss: 0.2017 - val_loss:
0.3373
Epoch 47/100
7/7 [==============================] - 0s 15ms/step - loss: 0.1920 - val_loss:
0.3841
Epoch 48/100
7/7 [==============================] - 0s 18ms/step - loss: 0.1898 - val_loss:
```

```
0.3464
Epoch 49/100
7/7 [==============================] - 0s 15ms/step - loss: 0.1839 - val_loss:
0.3558
Epoch 50/100
7/7 [==============================] - 0s 17ms/step - loss: 0.1810 - val_loss:
0.3523
Epoch 51/100
7/7 [==============================] - 0s 18ms/step - loss: 0.1776 - val_loss:
0.3399
Epoch 52/100
7/7 [==============================] - 0s 17ms/step - loss: 0.1749 - val_loss:
0.3512
Epoch 53/100
7/7 [==============================] - 0s 13ms/step - loss: 0.1733 - val_loss:
0.3541
Epoch 54/100
7/7 [==============================] - 0s 12ms/step - loss: 0.1695 - val_loss:
0.3581
Epoch 55/100
7/7 [==============================] - 0s 12ms/step - loss: 0.1667 - val_loss:
0.3567
Epoch 56/100
7/7 [==============================] - 0s 11ms/step - loss: 0.1664 - val_loss:
0.3537
Epoch 57/100
7/7 [==============================] - 0s 9ms/step - loss: 0.1600 - val_loss:
0.3879
Epoch 58/100
7/7 [==============================] - 0s 12ms/step - loss: 0.1594 - val_loss:
0.3545
Epoch 59/100
7/7 [==============================] - 0s 12ms/step - loss: 0.1633 - val_loss:
0.3435
Epoch 60/100
7/7 [==============================] - 0s 11ms/step - loss: 0.1535 - val_loss:
0.3908
Epoch 61/100
7/7 [==============================] - 0s 10ms/step - loss: 0.1523 - val_loss:
0.3762
Epoch 62/100
7/7 [==============================] - 0s 10ms/step - loss: 0.1513 - val_loss:
0.3556
Epoch 63/100
7/7 [==============================] - 0s 10ms/step - loss: 0.1501 - val_loss:
0.3867
Epoch 64/100
7/7 [==============================] - 0s 10ms/step - loss: 0.1457 - val_loss:
```

```
0.4200
Epoch 65/100
7/7 [==============================] - 0s 11ms/step - loss: 0.1459 - val_loss:
0.3950
Epoch 66/100
7/7 [==============================] - 0s 12ms/step - loss: 0.1417 - val_loss:
0.4208
Epoch 67/100
7/7 [==============================] - 0s 10ms/step - loss: 0.1394 - val_loss:
0.3903
Epoch 68/100
7/7 [==============================] - 0s 15ms/step - loss: 0.1365 - val_loss:
0.4049
Epoch 69/100
7/7 [==============================] - 0s 10ms/step - loss: 0.1364 - val_loss:
0.4120
Epoch 70/100
7/7 [==============================] - 0s 10ms/step - loss: 0.1409 - val_loss:
0.4414
Epoch 71/100
7/7 [==============================] - 0s 13ms/step - loss: 0.1324 - val_loss:
0.3949
Epoch 72/100
7/7 [==============================] - 0s 13ms/step - loss: 0.1325 - val_loss:
0.4378
Epoch 73/100
7/7 [==============================] - 0s 10ms/step - loss: 0.1321 - val_loss:
0.4377
Epoch 74/100
7/7 [==============================] - 0s 13ms/step - loss: 0.1272 - val_loss:
0.4049
Epoch 75/100
7/7 [==============================] - 0s 10ms/step - loss: 0.1254 - val_loss:
0.4147
Epoch 76/100
7/7 [==============================] - 0s 13ms/step - loss: 0.1273 - val_loss:
0.4198
Epoch 77/100
7/7 [==============================] - 0s 13ms/step - loss: 0.1223 - val_loss:
0.4190
Epoch 78/100
7/7 [==============================] - 0s 10ms/step - loss: 0.1215 - val_loss:
0.4398
Epoch 79/100
7/7 [==============================] - 0s 12ms/step - loss: 0.1170 - val_loss:
0.4389
Epoch 80/100
7/7 [==============================] - 0s 12ms/step - loss: 0.1173 - val_loss:
```

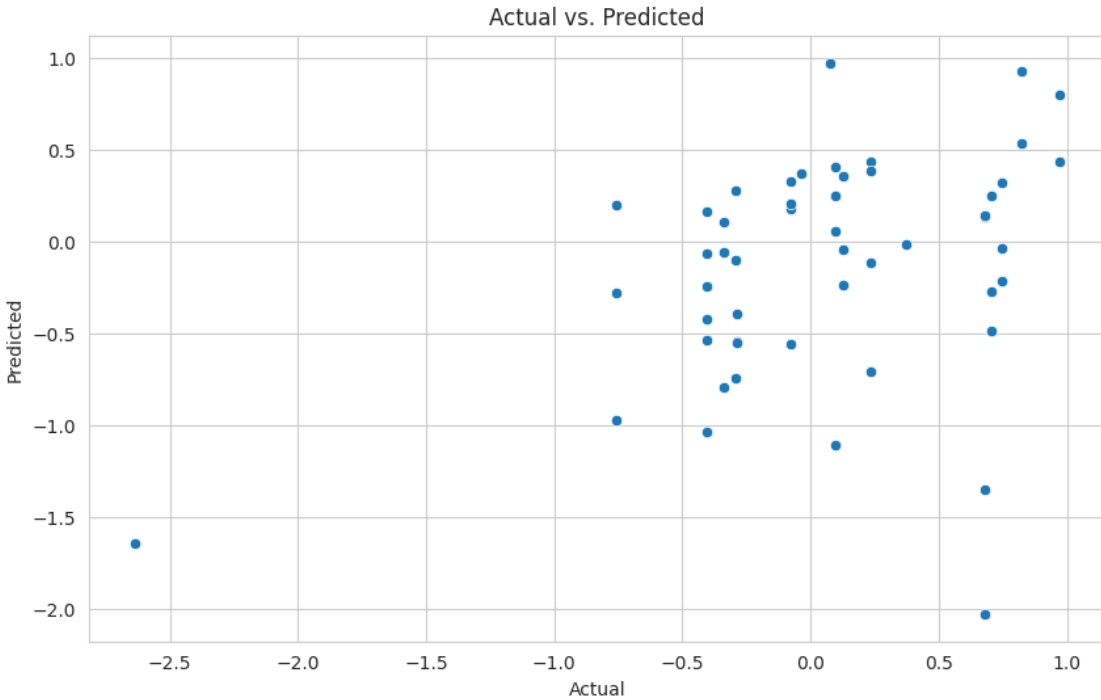
```
0.4123
Epoch 81/100
7/7 [==============================] - 0s 13ms/step - loss: 0.1172 - val_loss:
0.4472
Epoch 82/100
7/7 [==============================] - 0s 10ms/step - loss: 0.1171 - val_loss:
0.4490
Epoch 83/100
7/7 [==============================] - 0s 13ms/step - loss: 0.1138 - val_loss:
0.4354
Epoch 84/100
7/7 [==============================] - 0s 10ms/step - loss: 0.1136 - val_loss:
0.4400
Epoch 85/100
7/7 [==============================] - 0s 10ms/step - loss: 0.1141 - val_loss:
0.4414
Epoch 86/100
7/7 [==============================] - 0s 13ms/step - loss: 0.1084 - val_loss:
0.4338
Epoch 87/100
7/7 [==============================] - 0s 10ms/step - loss: 0.1139 - val_loss:
0.4577
Epoch 88/100
7/7 [==============================] - 0s 12ms/step - loss: 0.1095 - val_loss:
0.4786
Epoch 89/100
7/7 [==============================] - 0s 11ms/step - loss: 0.1116 - val_loss:
0.4513
Epoch 90/100
7/7 [==============================] - 0s 12ms/step - loss: 0.1073 - val_loss:
0.4935
Epoch 91/100
7/7 [==============================] - 0s 12ms/step - loss: 0.1082 - val_loss:
0.4809
Epoch 92/100
7/7 [==============================] - 0s 15ms/step - loss: 0.1064 - val_loss:
0.4559
Epoch 93/100
7/7 [==============================] - 0s 10ms/step - loss: 0.1030 - val_loss:
0.4684
Epoch 94/100
7/7 [==============================] - 0s 10ms/step - loss: 0.1030 - val_loss:
0.4498
Epoch 95/100
7/7 [==============================] - 0s 14ms/step - loss: 0.1021 - val_loss:
0.5100
Epoch 96/100
7/7 [==============================] - 0s 13ms/step - loss: 0.1001 - val_loss:
```

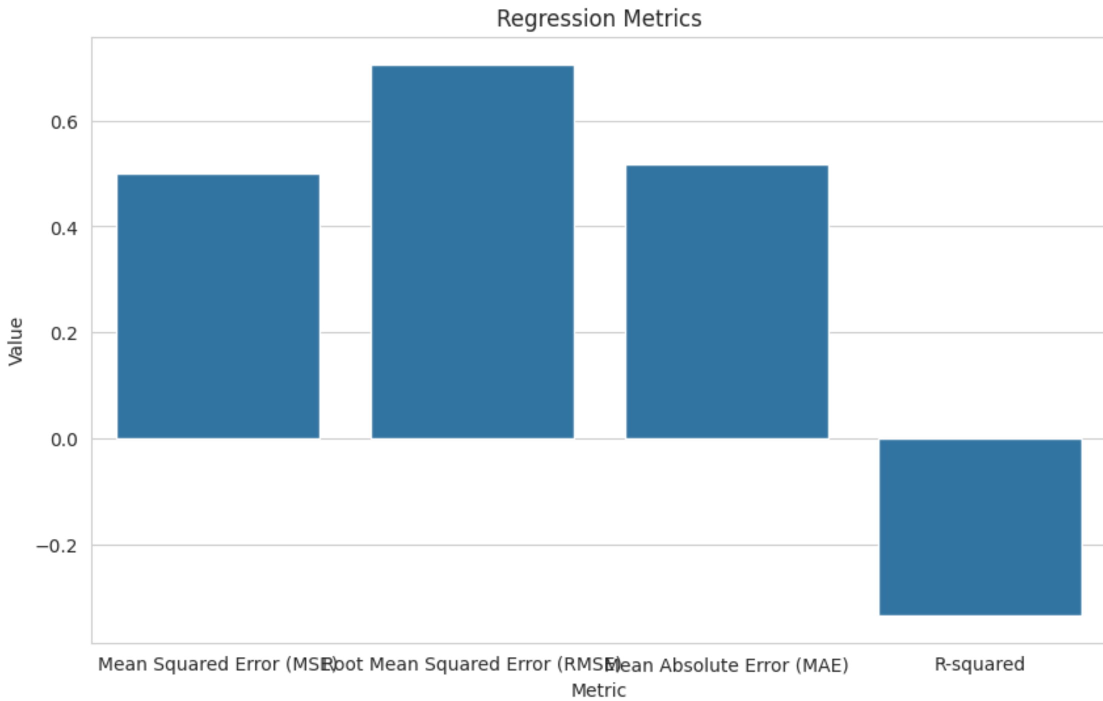
```
0.4570
Epoch 97/100
7/7 [==============================] - 0s 13ms/step - loss: 0.0999 - val_loss:
0.4987
Epoch 98/100
7/7 [==============================] - 0s 10ms/step - loss: 0.0991 - val_loss:
0.4855
Epoch 99/100
7/7 [==============================] - 0s 10ms/step - loss: 0.0988 - val_loss:
0.4716
Epoch 100/100
7/7 [==============================] - 0s 13ms/step - loss: 0.0962 - val_loss:
0.4940
GRU Loss: 0.3694315254688263
BiGRU Loss: 0.49399012327194214
2/2 [==============================] - 0s 6ms/step
2/2 [==============================] - 1s 9ms/step
```



[ ]:
```
y_pred_bilstm_flat = y_pred_bilstm.flatten()
print("BiLSTM METRICS")
# Use the function to get the metrics
metrics = evaluate_model(y_test, y_pred_bilstm_flat)

# Convert the metrics to a DataFrame
metrics_df = pd.DataFrame(list(metrics.items()), columns=['Metric', 'Value'])

# Create the barplot
plt.figure(figsize=(10, 6))
sns.barplot(x='Metric', y='Value', data=metrics_df)
plt.title('Regression Metrics')
plt.show()
```

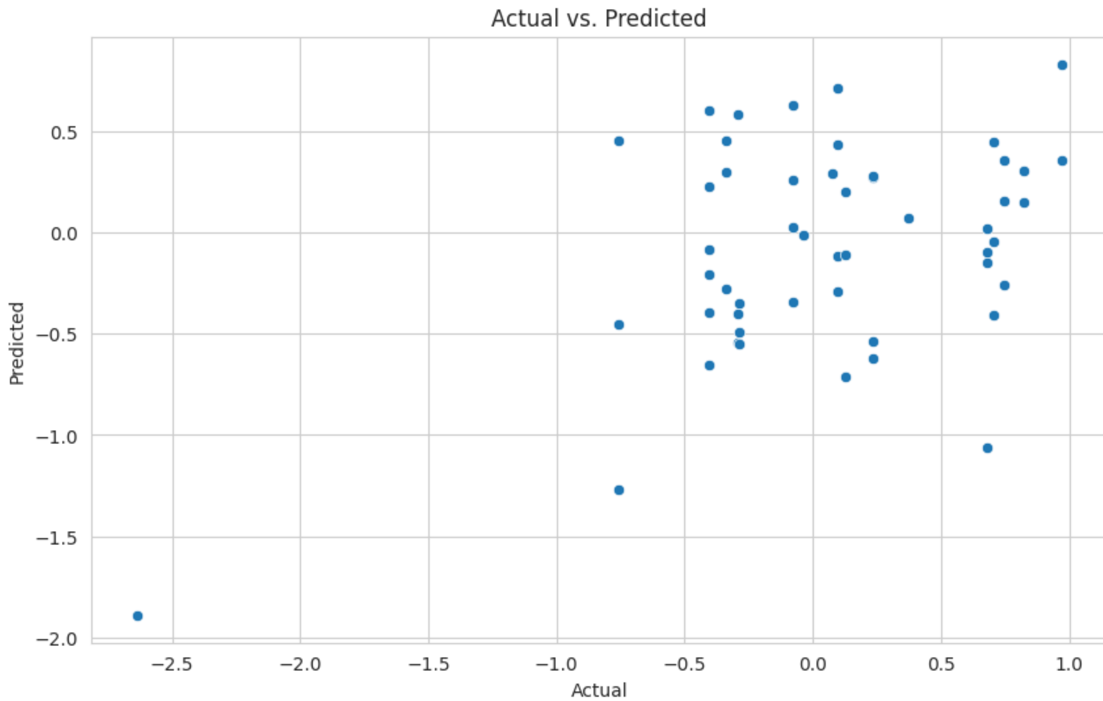
```
BiLSTM METRICS
Regression Metrics:
Mean Squared Error (MSE): 0.4993773689000158
Root Mean Squared Error (RMSE): 0.7066663773663041
Mean Absolute Error (MAE): 0.518508944506656
R-squared: -0.33427724240089884
```

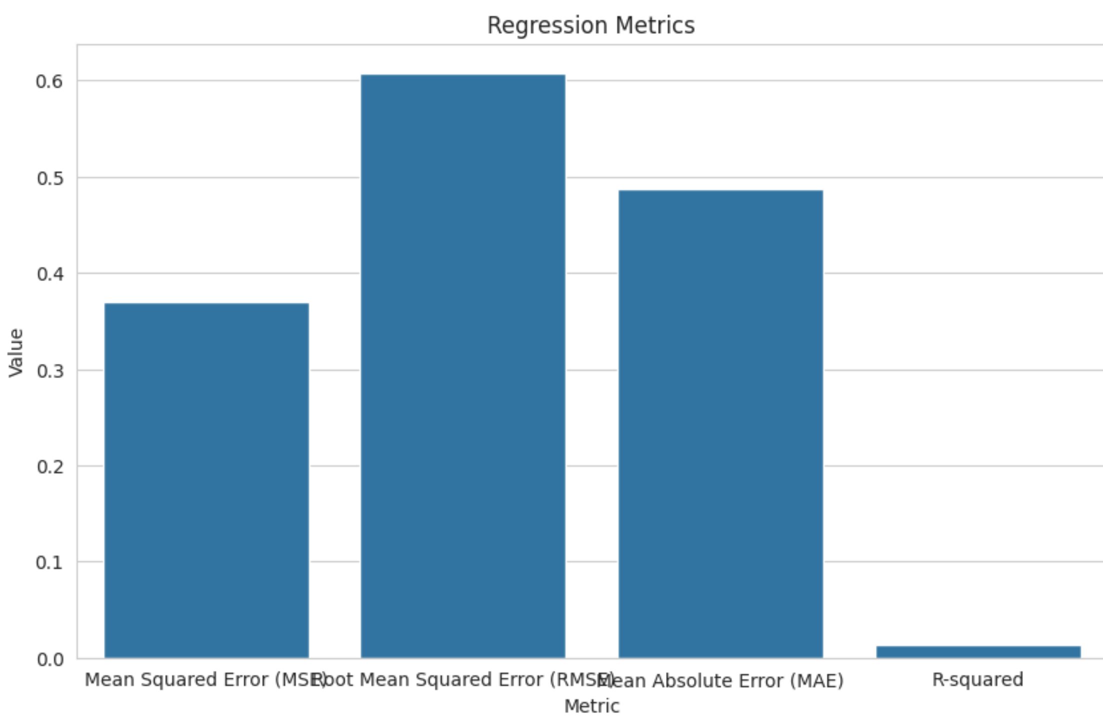




[ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

y_pred_gru_flat = y_pred_gru.flatten()
print("GRU METRICS")
# Use the function to get the metrics
metrics = evaluate_model(y_test, y_pred_gru_flat)

# Convert the metrics to a DataFrame
metrics_df = pd.DataFrame(list(metrics.items()), columns=['Metric', 'Value'])

# Create the barplot
plt.figure(figsize=(10, 6))
sns.barplot(x='Metric', y='Value', data=metrics_df)
plt.title('Regression Metrics')
plt.show()
```

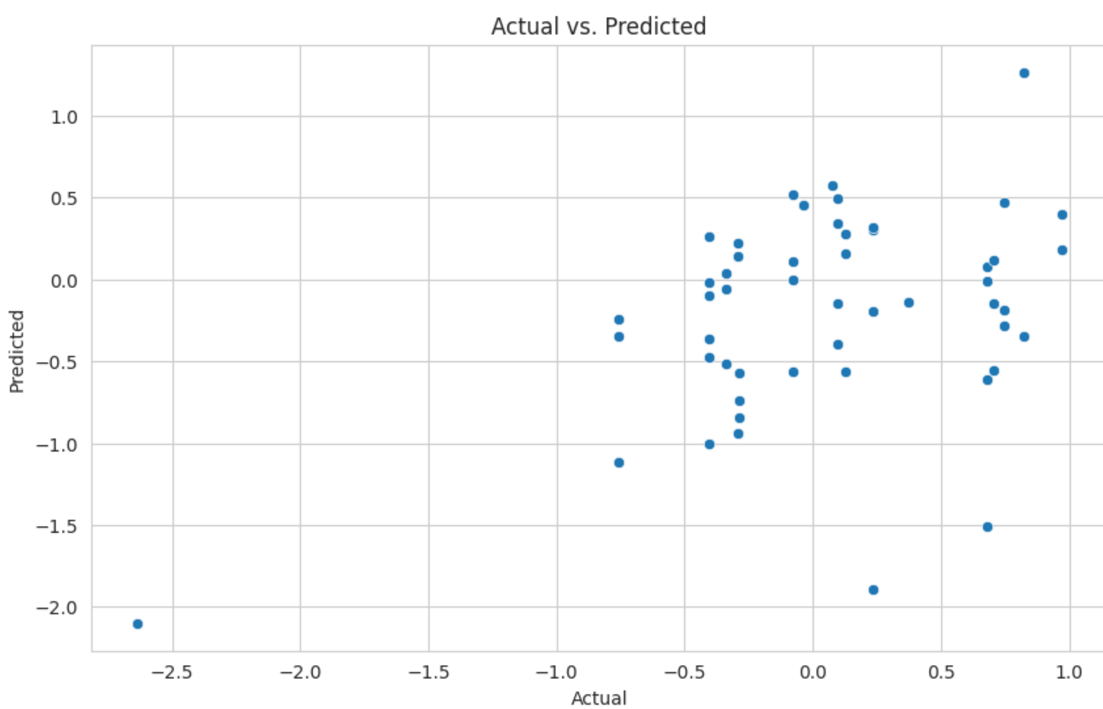
```
GRU METRICS
Regression Metrics:
Mean Squared Error (MSE): 0.3694315387835335
Root Mean Squared Error (RMSE): 0.6078088011731432
Mean Absolute Error (MAE): 0.4875229776845975
R-squared: 0.012922640227740286
```

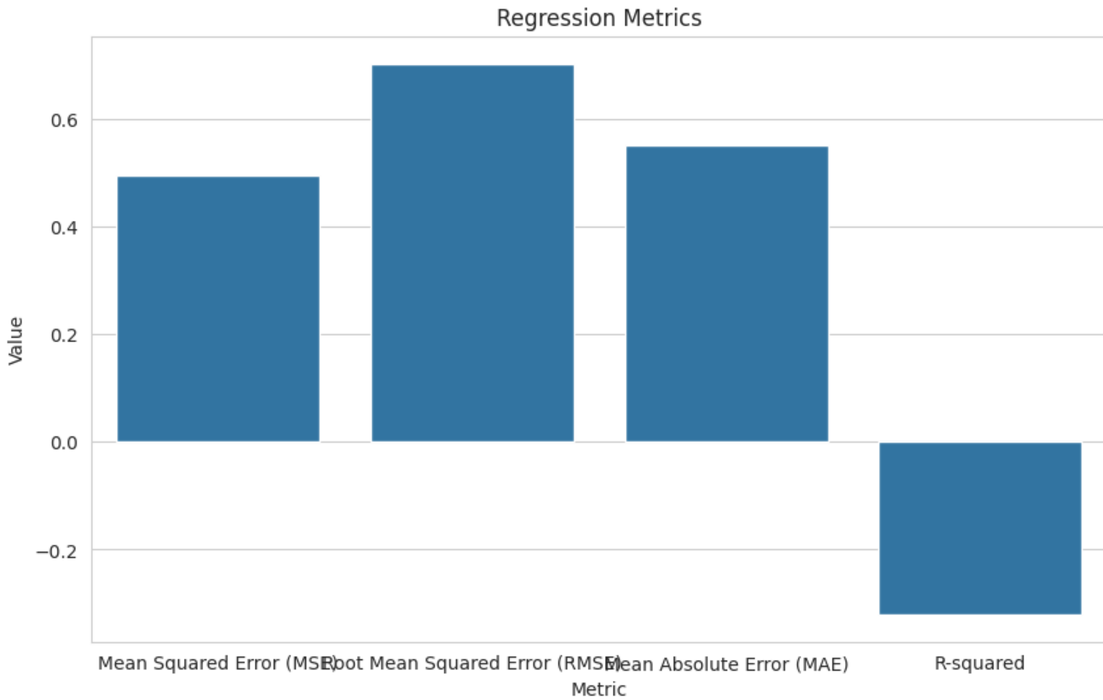

Actual vs. Predicted
Predicted
Actual
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0


Regression Metrics
Value
Metric
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Mean Squared Error (MSE)
Root Mean Squared Error (RMSE)
Mean Absolute Error (MAE)
R-squared

[ ]:
```python
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd


y_pred_bigru_flat = y_pred_bigru.flatten()
print("BiGRU METRICS")
# Use the function to get the metrics
metrics = evaluate_model(y_test, y_pred_bigru_flat)

# Convert the metrics to a DataFrame
metrics_df = pd.DataFrame(list(metrics.items()), columns=['Metric', 'Value'])

# Create the barplot
plt.figure(figsize=(10, 6))
sns.barplot(x='Metric', y='Value', data=metrics_df)
plt.title('Regression Metrics')
plt.show()
```

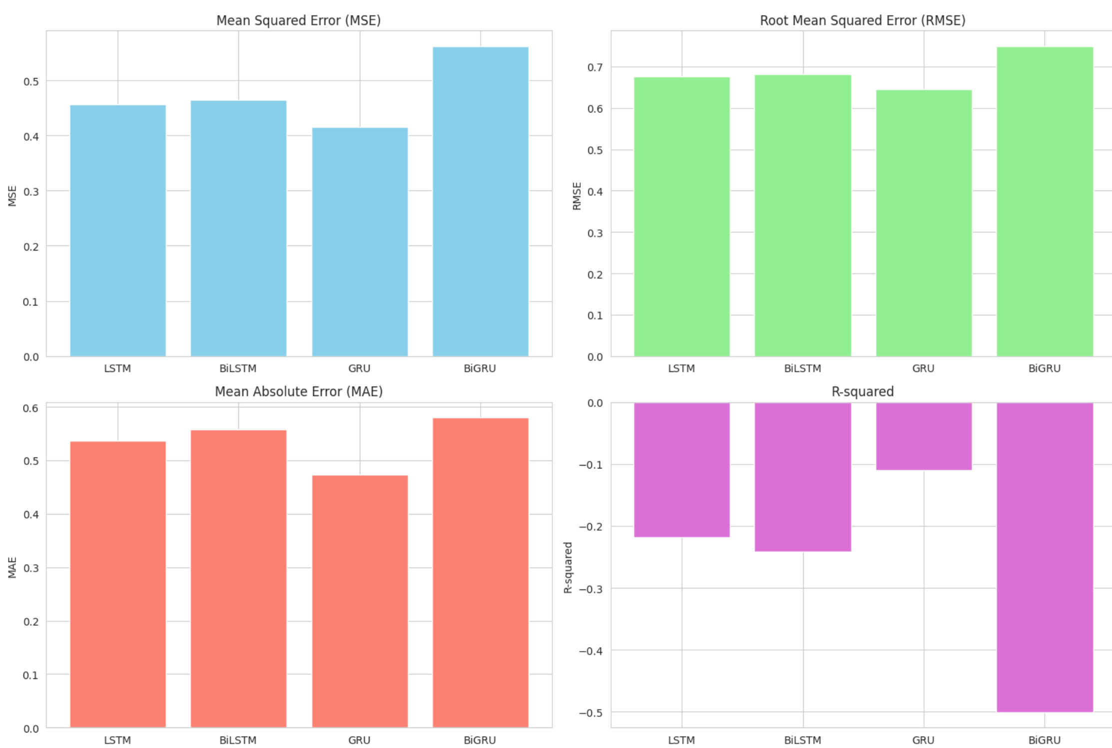
```
BiGRU METRICS
Regression Metrics:
Mean Squared Error (MSE): 0.4939901358552205
Root Mean Squared Error (RMSE): 0.7028443183630501
Mean Absolute Error (MAE): 0.5503988972015131
R-squared: -0.31988319313308033
```





[ ]:
```
import pandas as pd
import matplotlib.pyplot as plt

# Create a DataFrame with the metrics
metrics_data = {
    'Model': ['LSTM', 'BiLSTM', 'GRU', 'BiGRU'],
    'MSE': [0.4562067420626733, 0.46446107029808337, 0.41529824621674843, 0.
 ↪5616803301187765],
    'RMSE': [0.6754307825844728, 0.6815138078557788, 0.644436378719225, 0.
 ↪7494533541980959],
    'MAE': [0.5365917077435299, 0.5579181404310574, 0.4737416878096157, 0.
 ↪5797752033972754],
    'R-squared': [-0.21893043552391278, -0.24098502390078402, -0.
 ↪1096277749962633, -0.5007433829700807]
}

df_metrics = pd.DataFrame(metrics_data)

# Plotting the metrics
fig, axs = plt.subplots(2, 2, figsize=(15, 10))

# MSE plot
```

```python
axs[0, 0].bar(df_metrics['Model'], df_metrics['MSE'], color='skyblue')
axs[0, 0].set_title('Mean Squared Error (MSE)')
axs[0, 0].set_ylabel('MSE')

# RMSE plot
axs[0, 1].bar(df_metrics['Model'], df_metrics['RMSE'], color='lightgreen')
axs[0, 1].set_title('Root Mean Squared Error (RMSE)')
axs[0, 1].set_ylabel('RMSE')

# MAE plot
axs[1, 0].bar(df_metrics['Model'], df_metrics['MAE'], color='salmon')
axs[1, 0].set_title('Mean Absolute Error (MAE)')
axs[1, 0].set_ylabel('MAE')

# R-squared plot
axs[1, 1].bar(df_metrics['Model'], df_metrics['R-squared'], color='orchid')
axs[1, 1].set_title('R-squared')
axs[1, 1].set_ylabel('R-squared')

plt.tight_layout()
plt.show()
```



Based on the provided metrics for each model, here are the inferences and conclusions:

LSTM Model: - **MSE**: 0.4562 - **RMSE**: 0.6754 - **MAE**: 0.5366 - **R-squared**: -0.2189

The LSTM model has a moderate level of errors as indicated by the MSE, RMSE, and MAE. However, the negative R-squared value suggests that the model is not capturing the underlying pattern of the data and is performing worse than a simple horizontal line fit.

BiLSTM Model: - **MSE**: 0.4645 - **RMSE**: 0.6815 - **MAE**: 0.5579 - **R-squared**: -0.2410

The BiLSTM model shows slightly higher errors compared to the LSTM model. The negative R-squared value is also worse, indicating that the bidirectional nature of the model did not contribute to better performance and may have added complexity without benefit.

GRU Model: - **MSE**: 0.4153 - **RMSE**: 0.6444 - **MAE**: 0.4737 - **R-squared**: -0.1096

The GRU model has the lowest errors among the models and a less negative R-squared value, suggesting it performed better than both LSTM and BiLSTM models. However, the negative R-squared value still indicates a poor fit to the data.

BiGRU Model: - **MSE**: 0.5617 - **RMSE**: 0.7495 - **MAE**: 0.5798 - **R-squared**: -0.5007

The BiGRU model has the highest errors and the most negative R-squared value, which implies that it performed the worst among all the models. This suggests that the bidirectional GRU did not capture the sequential nature of the data effectively.

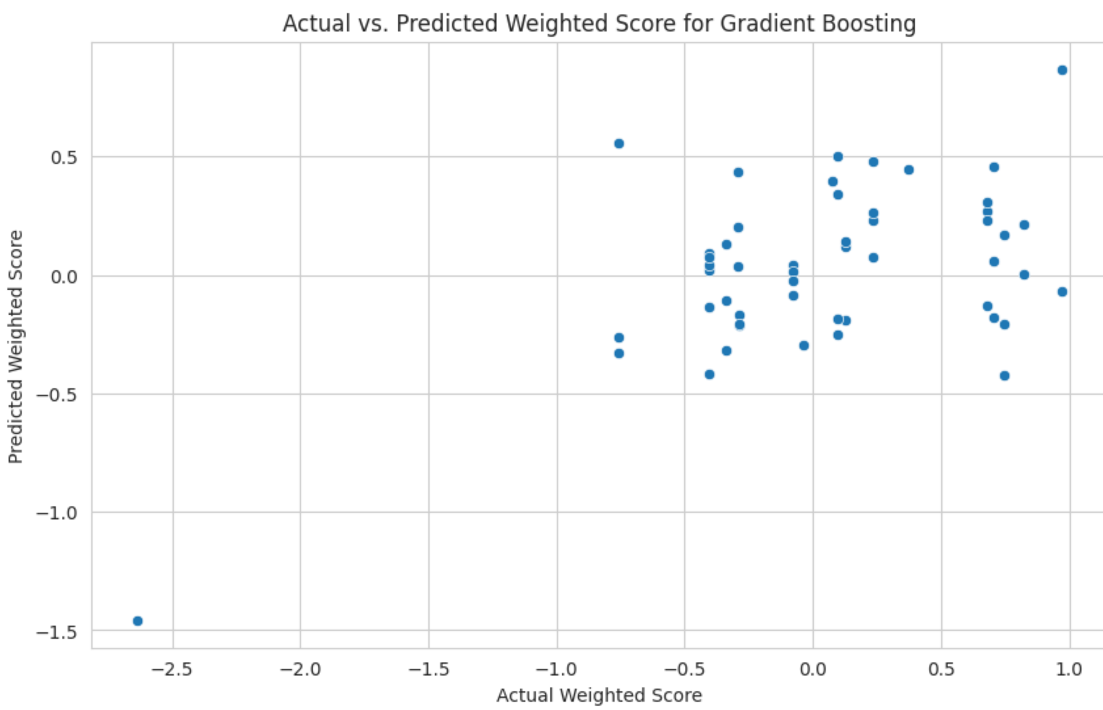
Conclusion: All models have negative R-squared values, which is a strong indicator that they are not suitable for the dataset in their current form. The GRU model shows relatively better performance, but still not a good fit. This could be due to several factors such as inadequate feature selection, need for hyperparameter tuning, or the complexity of the data itself. It is recommended to revisit the data preprocessing steps, consider feature engineering, and possibly explore different modeling approaches or algorithms. Additionally, evaluating the models on different metrics and conducting error analysis could provide further insights into improving model performance.

[ ]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge, Lasso
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
import seaborn as sns


# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

[ ]:
```
# Initialize machine learning models
ridge_model = Ridge(random_state=42)
lasso_model = Lasso(random_state=42)
rf_model = RandomForestRegressor(random_state=42)
gb_model = GradientBoostingRegressor(random_state=42)
```

```python
# Create a dictionary of models for easy access
models = {'Ridge': ridge_model, 'Lasso': lasso_model, 'Random Forest':
 ↪rf_model, 'Gradient Boosting': gb_model}

# Train and evaluate each model
model_metrics = {}
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    model_metrics[name] = {'MSE': mse, 'R-squared': r2}
    print(f'{name} - MSE: {mse}, R-squared: {r2}')

# Find the best model based on R-squared
best_model_name = max(model_metrics, key=lambda k:
 ↪model_metrics[k]['R-squared'])
best_model = models[best_model_name]
y_pred_best = best_model.predict(X_test)

# Plotting the results for the best model
plt.figure(figsize=(10, 6))
sns.scatterplot(x=y_test, y=y_pred_best)
plt.xlabel('Actual Weighted Score')
plt.ylabel('Predicted Weighted Score')
plt.title(f'Actual vs. Predicted Weighted Score for {best_model_name}')
plt.show()
```

```
Ridge - MSE: 0.3575427842636131, R-squared: 0.0446879856041108
Lasso - MSE: 0.33736560137653193, R-squared: 0.09859903087493371
Random Forest - MSE: 0.2772359066771713, R-squared: 0.25925846044939305
Gradient Boosting - MSE: 0.26687389134109546, R-squared: 0.286944539373613
```



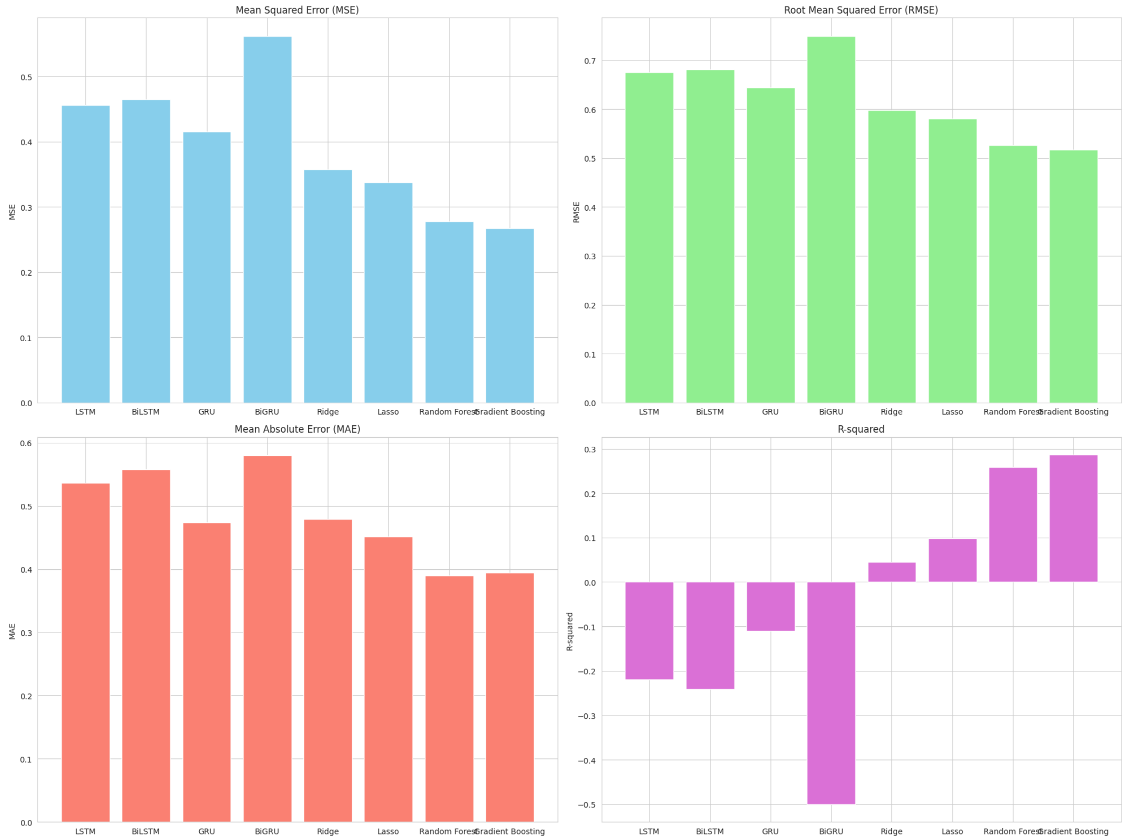
[ ]:
```
# Function to calculate metrics
def calculate_metrics(y_true, y_pred):
    mse = mean_squared_error(y_true, y_pred)
    rmse = np.sqrt(mse)
    mae = mean_absolute_error(y_true, y_pred)
    r2 = r2_score(y_true, y_pred)
    return mse, rmse, mae, r2

# Calculate metrics for Ridge
y_pred_ridge=ridge_model.predict(X_test)
ridge_mse, ridge_rmse, ridge_mae, ridge_r2 = calculate_metrics(y_test,␣
 ↪y_pred_ridge)

# Calculate metrics for Lasso
y_pred_lasso=lasso_model.predict(X_test)
lasso_mse, lasso_rmse, lasso_mae, lasso_r2 = calculate_metrics(y_test,␣
 ↪y_pred_lasso)

# Calculate metrics for Random Forest
y_pred_rf=rf_model.predict(X_test)
rf_mse, rf_rmse, rf_mae, rf_r2 = calculate_metrics(y_test, y_pred_rf)

# Calculate metrics for Gradient Boosting
y_pred_gb=gb_model.predict(X_test)
```

```
gb_mse, gb_rmse, gb_mae, gb_r2 = calculate_metrics(y_test, y_pred_gb)

# Print the metrics
print(f'Ridge - MSE: {ridge_mse}, RMSE: {ridge_rmse}, MAE: {ridge_mae},␣
↪R-squared: {ridge_r2}')
print(f'Lasso - MSE: {lasso_mse}, RMSE: {lasso_rmse}, MAE: {lasso_mae},␣
↪R-squared: {lasso_r2}')
print(f'Random Forest - MSE: {rf_mse}, RMSE: {rf_rmse}, MAE: {rf_mae},␣
↪R-squared: {rf_r2}')
print(f'Gradient Boosting - MSE: {gb_mse}, RMSE: {gb_rmse}, MAE: {gb_mae},␣
↪R-squared: {gb_r2}')
```

```
Ridge - MSE: 0.3575427842636131, RMSE: 0.5979488140832901, MAE:
0.47871089062785516, R-squared: 0.0446879856041108
Lasso - MSE: 0.33736560137653193, RMSE: 0.5808318184952783, MAE:
0.451416547604268, R-squared: 0.09859903087493371
Random Forest - MSE: 0.2772359066771713, RMSE: 0.5265319616862506, MAE:
0.3896422695461326, R-squared: 0.25925846044939305
Gradient Boosting - MSE: 0.26687389134109546, RMSE: 0.5165983849578853, MAE:
0.3946339132125805, R-squared: 0.286944539373613
```

[ ]:
```
# Existing metrics data
metrics_data = {
    'Model': ['LSTM', 'BiLSTM', 'GRU', 'BiGRU', 'Ridge', 'Lasso', 'Random␣
↪Forest', 'Gradient Boosting'],
    'MSE': [0.4562067420626733, 0.46446107029808337, 0.41529824621674843, 0.
↪5616803301187765, 0.3575427842636131, 0.33736560137653193, 0.
↪2772359066771713, 0.26687389134109546],
    'RMSE': [0.6754307825844728, 0.6815138078557788, 0.644436378719225, 0.
↪7494533541980959, 0.5979488140832901, 0.5808318184952783, 0.
↪5265319616862506, 0.5165983849578853],
    'MAE': [0.5365917077435299, 0.5579181404310574, 0.4737416878096157, 0.
↪5797752033972754, 0.47871089062785516, 0.451416547604268, 0.
↪3896422695461326, 0.3946339132125805],
    'R-squared': [-0.21893043552391278, -0.24098502390078402, -0.
↪1096277749962633, -0.5007433829700807, 0.0446879856041108, 0.
↪09859903087493371, 0.25925846044939305, 0.286944539373613]
}

df_metrics = pd.DataFrame(metrics_data)

# Plotting the updated metrics
fig, axs = plt.subplots(2, 2, figsize=(20, 15))

# MSE plot
axs[0, 0].bar(df_metrics['Model'], df_metrics['MSE'], color='skyblue')
```

```
axs[0, 0].set_title('Mean Squared Error (MSE)')
axs[0, 0].set_ylabel('MSE')

# RMSE plot
axs[0, 1].bar(df_metrics['Model'], df_metrics['RMSE'], color='lightgreen')
axs[0, 1].set_title('Root Mean Squared Error (RMSE)')
axs[0, 1].set_ylabel('RMSE')

# MAE plot
axs[1, 0].bar(df_metrics['Model'], df_metrics['MAE'], color='salmon')
axs[1, 0].set_title('Mean Absolute Error (MAE)')
axs[1, 0].set_ylabel('MAE')

# R-squared plot
axs[1, 1].bar(df_metrics['Model'], df_metrics['R-squared'], color='orchid')
axs[1, 1].set_title('R-squared')
axs[1, 1].set_ylabel('R-squared')

plt.tight_layout()
plt.show()
```



conclusions and inferences:

- **Best Model**: The **Gradient Boosting** model has the lowest MSE and MAE, and the highest R-squared value, indicating it is the best performing model among those evaluated.
- **Model Comparison**: **Random Forest** also performs well with low MSE and RMSE, and a good R-squared value, making it a close competitor to Gradient Boosting.
- **Performance Metrics**: Lower MSE, RMSE, and MAE values suggest better predictive accuracy, while a higher R-squared value indicates a better fit of the model to the data.
- **Model Fit**: Despite the negative R-squared values for some models, Ridge Regression shows a relative improvement, suggesting some potential in specific scenarios.

[ ]:
```
# Feature importance for Random Forest
rf_feature_importance = rf_model.feature_importances_

# Feature importance for Gradient Boosting
gb_feature_importance = gb_model.feature_importances_

# Coefficients for Ridge
ridge_coefficients = ridge_model.coef_

# Coefficients for Lasso
lasso_coefficients = lasso_model.coef_

# Create DataFrame for each model's feature importance/coefficients
df_rf_importance = pd.DataFrame({'Feature': X.columns, 'Importance': 
 ↪rf_feature_importance})
df_gb_importance = pd.DataFrame({'Feature': X.columns, 'Importance': 
 ↪gb_feature_importance})
df_ridge_coefficients = pd.DataFrame({'Feature': X.columns, 'Coefficient': 
 ↪ridge_coefficients})
df_lasso_coefficients = pd.DataFrame({'Feature': X.columns, 'Coefficient': 
 ↪lasso_coefficients})

# Sort DataFrame by importance/coefficient
df_rf_importance = df_rf_importance.sort_values(by='Importance', 
 ↪ascending=False)
df_gb_importance = df_gb_importance.sort_values(by='Importance', 
 ↪ascending=False)
df_ridge_coefficients = df_ridge_coefficients.sort_values(by='Coefficient', 
 ↪ascending=False)
df_lasso_coefficients = df_lasso_coefficients.sort_values(by='Coefficient', 
 ↪ascending=False)

# Print the top 5 most influential variables for each model
print("Random Forest Feature Importance:")
print(df_rf_importance)
```

```python
print("\nGradient Boosting Feature Importance:")
print(df_gb_importance)

print("\nRidge Coefficients:")
print(df_ridge_coefficients)

print("\nLasso Coefficients:")
print(df_lasso_coefficients)
```

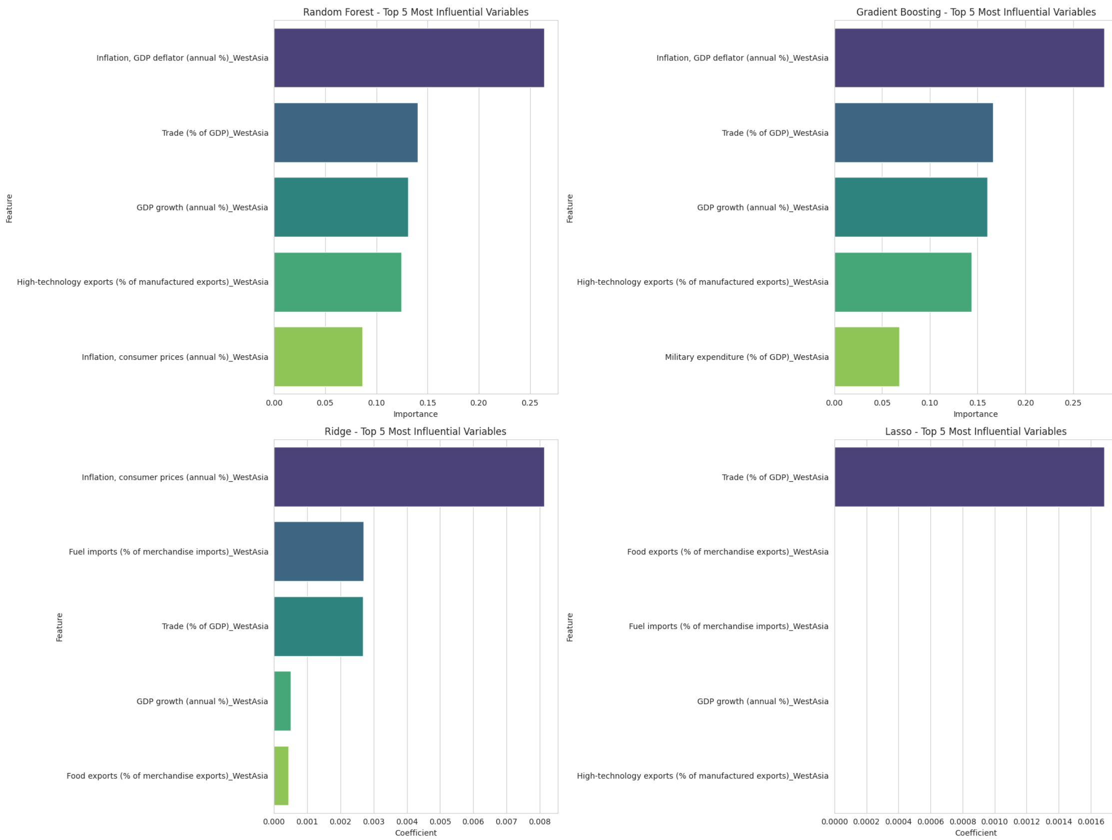
```
Random Forest Feature Importance:
                                            Feature  Importance
5        Inflation, GDP deflator (annual %)_WestAsia    0.263810
8                         Trade (% of GDP)_WestAsia    0.140702
3                    GDP growth (annual %)_WestAsia    0.130898
4  High-technology exports (% of manufactured exp…    0.124515
6     Inflation, consumer prices (annual %)_WestAsia    0.086614
0     Exports of goods and services (annual % growth)    0.067723
7           Military expenditure (% of GDP)_WestAsia    0.054579
2   Fuel imports (% of merchandise imports)_WestAsia    0.050327
1   Food exports (% of merchandise exports)_WestAsia    0.047328
9              Trade in services (% of GDP)_WestAsia    0.033505

Gradient Boosting Feature Importance:
                                            Feature  Importance
5        Inflation, GDP deflator (annual %)_WestAsia    0.282827
8                         Trade (% of GDP)_WestAsia    0.166055
3                    GDP growth (annual %)_WestAsia    0.160629
4  High-technology exports (% of manufactured exp…    0.143541
7           Military expenditure (% of GDP)_WestAsia    0.068283
6     Inflation, consumer prices (annual %)_WestAsia    0.058728
0     Exports of goods and services (annual % growth)    0.034823
1   Food exports (% of merchandise exports)_WestAsia    0.030417
9              Trade in services (% of GDP)_WestAsia    0.028838
2   Fuel imports (% of merchandise imports)_WestAsia    0.025860

Ridge Coefficients:
                                            Feature  Coefficient
6     Inflation, consumer prices (annual %)_WestAsia     0.008129
2   Fuel imports (% of merchandise imports)_WestAsia     0.002705
8                         Trade (% of GDP)_WestAsia     0.002673
3                    GDP growth (annual %)_WestAsia     0.000512
1   Food exports (% of merchandise exports)_WestAsia     0.000435
9              Trade in services (% of GDP)_WestAsia    -0.000101
0     Exports of goods and services (annual % growth)    -0.002621
7           Military expenditure (% of GDP)_WestAsia    -0.004330
5        Inflation, GDP deflator (annual %)_WestAsia    -0.017891
4  High-technology exports (% of manufactured exp…    -0.027723
```

```
Lasso Coefficients:
                                             Feature  Coefficient
8                           Trade (% of GDP)_WestAsia     0.001685
1   Food exports (% of merchandise exports)_WestAsia     0.000000
2   Fuel imports (% of merchandise imports)_WestAsia     0.000000
3                      GDP growth (annual %)_WestAsia    -0.000000
4  High-technology exports (% of manufactured exp…   -0.000000
6     Inflation, consumer prices (annual %)_WestAsia     0.000000
7           Military expenditure (% of GDP)_WestAsia    -0.000000
9              Trade in services (% of GDP)_WestAsia     0.000000
0    Exports of goods and services (annual % growth)    -0.000161
5        Inflation, GDP deflator (annual %)_WestAsia    -0.009962
```

[ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Set up the matplotlib figure
plt.figure(figsize=(20, 15))

# Plot Random Forest Feature Importance
plt.subplot(221)
sns.barplot(x='Importance', y='Feature', data=df_rf_importance.head(5),␣
 ↪palette='viridis')
plt.title('Random Forest - Top 5 Most Influential Variables')
plt.xlabel('Importance')
plt.ylabel('Feature')

# Plot Gradient Boosting Feature Importance
plt.subplot(222)
sns.barplot(x='Importance', y='Feature', data=df_gb_importance.head(5),␣
 ↪palette='viridis')
plt.title('Gradient Boosting - Top 5 Most Influential Variables')
plt.xlabel('Importance')
plt.ylabel('Feature')

# Plot Ridge Coefficients
plt.subplot(223)
sns.barplot(x='Coefficient', y='Feature', data=df_ridge_coefficients.head(5),␣
 ↪palette='viridis')
plt.title('Ridge - Top 5 Most Influential Variables')
plt.xlabel('Coefficient')
plt.ylabel('Feature')

# Plot Lasso Coefficients
plt.subplot(224)
```

```
sns.barplot(x='Coefficient', y='Feature', data=df_lasso_coefficients.head(5),␣
↪palette='viridis')
plt.title('Lasso - Top 5 Most Influential Variables')
plt.xlabel('Coefficient')
plt.ylabel('Feature')

# Adjust layout
plt.tight_layout()
```

```
<ipython-input-41-69f96f8ae2f0>:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x='Importance', y='Feature', data=df_rf_importance.head(5),
palette='viridis')
<ipython-input-41-69f96f8ae2f0>:16: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x='Importance', y='Feature', data=df_gb_importance.head(5),
palette='viridis')
<ipython-input-41-69f96f8ae2f0>:23: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x='Coefficient', y='Feature', data=df_ridge_coefficients.head(5),
palette='viridis')
<ipython-input-41-69f96f8ae2f0>:30: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x='Coefficient', y='Feature', data=df_lasso_coefficients.head(5),
palette='viridis')
```



[ ]:
```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Normalize the importance values to a probability scale (0 to 1) using min-max normalization
def normalize_importance(df):
    df['Importance_normalized'] = (df['Importance'] - df['Importance'].min()) / (df['Importance'].max() - df['Importance'].min())
    return df

# Assuming df_rf_importance, df_gb_importance, df_ridge_coefficients, and df_lasso_coefficients contain the feature importances or coefficients

# Create normalized dataframes
df_rf_importance_normalized = normalize_importance(df_rf_importance)
df_gb_importance_normalized = normalize_importance(df_gb_importance)
df_ridge_coefficients_normalized = normalize_importance(df_ridge_coefficients)
df_lasso_coefficients_normalized = normalize_importance(df_lasso_coefficients)
```

```python
# Set up the matplotlib figure
plt.figure(figsize=(20, 15))

# Plot Random Forest Feature Importance
plt.subplot(221)
sns.barplot(x='Importance_normalized', y='Feature',␣
 ↪data=df_rf_importance_normalized.head(5), palette='viridis')
plt.title('Random Forest - Top 5 Most Influential Variables')
plt.xlabel('Importance (Normalized)')
plt.ylabel('Feature')

# Plot Gradient Boosting Feature Importance
plt.subplot(222)
sns.barplot(x='Importance_normalized', y='Feature',␣
 ↪data=df_gb_importance_normalized.head(5), palette='viridis')
plt.title('Gradient Boosting - Top 5 Most Influential Variables')
plt.xlabel('Importance (Normalized)')
plt.ylabel('Feature')

# Plot Ridge Coefficients
plt.subplot(223)
sns.barplot(x='Importance_normalized', y='Feature',␣
 ↪data=df_ridge_coefficients_normalized.head(5), palette='viridis')
plt.title('Ridge - Top 5 Most Influential Variables')
plt.xlabel('Coefficient (Normalized)')
plt.ylabel('Feature')

# Plot Lasso Coefficients
plt.subplot(224)
sns.barplot(x='Importance_normalized', y='Feature',␣
 ↪data=df_lasso_coefficients_normalized.head(5), palette='viridis')
plt.title('Lasso - Top 5 Most Influential Variables')
plt.xlabel('Coefficient (Normalized)')
plt.ylabel('Feature')

# Adjust layout
plt.tight_layout()

# Show the plots
plt.show()
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in␣
 ↪get_loc(self, key)
   3652         try:
-> 3653             return self._engine.get_loc(casted_key)
```

```
   3654         except KeyError as err:

/usr/local/lib/python3.10/dist-packages/pandas/_libs/index.pyx in pandas._libs.
 ↪index.IndexEngine.get_loc()

/usr/local/lib/python3.10/dist-packages/pandas/_libs/index.pyx in pandas._libs.
 ↪index.IndexEngine.get_loc()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.
 ↪PyObjectHashTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.
 ↪PyObjectHashTable.get_item()

KeyError: 'Importance'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
<ipython-input-48-d2101d52c8dd> in <cell line: 15>()
     13 df_rf_importance_normalized = normalize_importance(df_rf_importance)
     14 df_gb_importance_normalized = normalize_importance(df_gb_importance)
---> 15 df_ridge_coefficients_normalized =␣
 ↪normalize_importance(df_ridge_coefficients)
     16 df_lasso_coefficients_normalized =␣
 ↪normalize_importance(df_lasso_coefficients)
     17

<ipython-input-48-d2101d52c8dd> in normalize_importance(df)
      5 # Normalize the importance values to a probability scale (0 to 1) using␣
 ↪min-max normalization
      6 def normalize_importance(df):
----> 7     df['Importance_normalized'] = (df['Importance'] - df['Importance'].
 ↪min()) / (df['Importance'].max() - df['Importance'].min())
      8     return df
      9

/usr/local/lib/python3.10/dist-packages/pandas/core/frame.py in␣
 ↪__getitem__(self, key)
   3759             if self.columns.nlevels > 1:
   3760                 return self._getitem_multilevel(key)
-> 3761             indexer = self.columns.get_loc(key)
   3762             if is_integer(indexer):
   3763                 indexer = [indexer]

/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in␣
 ↪get_loc(self, key)
   3653             return self._engine.get_loc(casted_key)
```

```
   3654         except KeyError as err:
-> 3655             raise KeyError(key) from err
   3656         except TypeError:
   3657             # If we have a listlike key, _check_indexing_error will raise

KeyError: 'Importance'
```

[ ]:
```
import pandas as pd

# Assuming india_data and west_asian_data are your DataFrames
# Group the data by country and calculate the average for each economic␣
 ↪indicator
india_grouped = india_data.groupby('Country Name').mean()
west_asian_grouped = west_asian_data.groupby('Country Name').mean()

# Merge the grouped data on 'Country Name'
merged_grouped = pd.merge(india_grouped, west_asian_grouped, on='Country Name',␣
 ↪suffixes=('_India', '_WestAsia'))

# Select columns for correlation analysis
columns_to_correlate = [
    'Exports of goods and services (% of GDP)_India',
    'Food exports (% of merchandise exports)_India',
    'Fuel imports (% of merchandise imports)_India',
    'GDP growth (annual %)_India',
    'High-technology exports (% of manufactured exports)_India',
    'Inflation, GDP deflator (annual %)_India',
    'Inflation, consumer prices (annual %)_India',
    'Military expenditure (% of GDP)_India',
    'Trade (% of GDP)_India',
    'Trade in services (% of GDP)_India',
    'Exports of goods and services (annual % growth)_WestAsia',
    'Food exports (% of merchandise exports)_WestAsia',
    'Fuel imports (% of merchandise imports)_WestAsia',
    'GDP growth (annual %)_WestAsia',
    'High-technology exports (% of manufactured exports)_WestAsia',
    'Inflation, GDP deflator (annual %)_WestAsia',
    'Inflation, consumer prices (annual %)_WestAsia',
    'Military expenditure (% of GDP)_WestAsia',
    'Trade (% of GDP)_WestAsia',
    'Trade in services (% of GDP)_WestAsia'
]

# Calculate correlation matrix
correlation_matrix = merged_grouped[columns_to_correlate].corr()
```

```
# Plot the correlation matrix
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(16, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', center=0)
plt.title('Correlation Matrix of Economic Indicators')
plt.show()
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-54-06fbc5d1942e> in <cell line: 36>()
     34
     35 # Calculate correlation matrix
---> 36 correlation_matrix = merged_grouped[columns_to_correlate].corr()
     37
     38 # Plot the correlation matrix

/usr/local/lib/python3.10/dist-packages/pandas/core/frame.py in␣
 ↪__getitem__(self, key)
   3765             if is_iterator(key):
   3766                 key = list(key)
-> 3767             indexer = self.columns._get_indexer_strict(key, "columns")[ ]
   3768
   3769         # take() does not accept boolean indexers

/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in␣
 ↪_get_indexer_strict(self, key, axis_name)
   5875             keyarr, indexer, new_indexer = self.
 ↪_reindex_non_unique(keyarr)
   5876
-> 5877         self._raise_if_missing(keyarr, indexer, axis_name)
   5878
   5879         keyarr = self.take(indexer)

/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in␣
 ↪_raise_if_missing(self, key, indexer, axis_name)
   5939
   5940             not_found = list(ensure_index(key)[missing_mask.
 ↪nonzero()[0]].unique())
-> 5941             raise KeyError(f"{not_found} not in index")
   5942
   5943     @overload

KeyError: "['Exports of goods and services (% of GDP)_India', 'Exports of goods␣
 ↪and services (annual % growth)_WestAsia'] not in index"
```

[ ]:
```
west_asian_data.shape
```

[ ]:
```
(280, 12)
```

[ ]:
```
!pip install nbconvert
```

```
Requirement already satisfied: nbconvert in /usr/local/lib/python3.10/dist-
packages (6.5.4)
Requirement already satisfied: lxml in /usr/local/lib/python3.10/dist-packages
(from nbconvert) (4.9.4)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-
packages (from nbconvert) (4.12.3)
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages
(from nbconvert) (6.1.0)
Requirement already satisfied: defusedxml in /usr/local/lib/python3.10/dist-
packages (from nbconvert) (0.7.1)
Requirement already satisfied: entrypoints>=0.2.2 in
/usr/local/lib/python3.10/dist-packages (from nbconvert) (0.4)
Requirement already satisfied: jinja2>=3.0 in /usr/local/lib/python3.10/dist-
packages (from nbconvert) (3.1.3)
Requirement already satisfied: jupyter-core>=4.7 in
/usr/local/lib/python3.10/dist-packages (from nbconvert) (5.7.2)
Requirement already satisfied: jupyterlab-pygments in
/usr/local/lib/python3.10/dist-packages (from nbconvert) (0.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.10/dist-packages (from nbconvert) (2.1.5)
Requirement already satisfied: mistune<2,>=0.8.1 in
/usr/local/lib/python3.10/dist-packages (from nbconvert) (0.8.4)
Requirement already satisfied: nbclient>=0.5.0 in
/usr/local/lib/python3.10/dist-packages (from nbconvert) (0.10.0)
Requirement already satisfied: nbformat>=5.1 in /usr/local/lib/python3.10/dist-
packages (from nbconvert) (5.10.4)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-
packages (from nbconvert) (24.0)
Requirement already satisfied: pandocfilters>=1.4.1 in
/usr/local/lib/python3.10/dist-packages (from nbconvert) (1.5.1)
Requirement already satisfied: pygments>=2.4.1 in
/usr/local/lib/python3.10/dist-packages (from nbconvert) (2.16.1)
Requirement already satisfied: tinycss2 in /usr/local/lib/python3.10/dist-
packages (from nbconvert) (1.2.1)
Requirement already satisfied: traitlets>=5.0 in /usr/local/lib/python3.10/dist-
packages (from nbconvert) (5.7.1)
Requirement already satisfied: platformdirs>=2.5 in
/usr/local/lib/python3.10/dist-packages (from jupyter-core>=4.7->nbconvert)
(4.2.0)
Requirement already satisfied: jupyter-client>=6.1.12 in
/usr/local/lib/python3.10/dist-packages (from nbclient>=0.5.0->nbconvert)
```

```
(6.1.12)
Requirement already satisfied: fastjsonschema>=2.15 in
/usr/local/lib/python3.10/dist-packages (from nbformat>=5.1->nbconvert) (2.19.1)
Requirement already satisfied: jsonschema>=2.6 in
/usr/local/lib/python3.10/dist-packages (from nbformat>=5.1->nbconvert) (4.19.2)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-
packages (from beautifulsoup4->nbconvert) (2.5)
Requirement already satisfied: six>=1.9.0 in /usr/local/lib/python3.10/dist-
packages (from bleach->nbconvert) (1.16.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-
packages (from bleach->nbconvert) (0.5.1)
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.10/dist-
packages (from jsonschema>=2.6->nbformat>=5.1->nbconvert) (23.2.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in
/usr/local/lib/python3.10/dist-packages (from
jsonschema>=2.6->nbformat>=5.1->nbconvert) (2023.12.1)
Requirement already satisfied: referencing>=0.28.4 in
/usr/local/lib/python3.10/dist-packages (from
jsonschema>=2.6->nbformat>=5.1->nbconvert) (0.34.0)
Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.10/dist-
packages (from jsonschema>=2.6->nbformat>=5.1->nbconvert) (0.18.0)
Requirement already satisfied: pyzmq>=13 in /usr/local/lib/python3.10/dist-
packages (from jupyter-client>=6.1.12->nbclient>=0.5.0->nbconvert) (23.2.1)
Requirement already satisfied: python-dateutil>=2.1 in
/usr/local/lib/python3.10/dist-packages (from jupyter-
client>=6.1.12->nbclient>=0.5.0->nbconvert) (2.8.2)
Requirement already satisfied: tornado>=4.1 in /usr/local/lib/python3.10/dist-
packages (from jupyter-client>=6.1.12->nbclient>=0.5.0->nbconvert) (6.3.3)
```

[62]:
```
!apt-get install texlive texlive-xetex texlive-latex-extra pandoc
```

```
Reading package lists… Done
Building dependency tree… Done
Reading state information… Done
pandoc is already the newest version (2.9.2.1-3ubuntu2).
texlive is already the newest version (2021.20220204-1).
texlive-latex-extra is already the newest version (2021.20220204-1).
texlive-xetex is already the newest version (2021.20220204-1).
0 upgraded, 0 newly installed, 0 to remove and 45 not upgraded.
```

[63]:
```
!jupyter nbconvert --to pdf Copy_of_fin_analytics_try_2_j_comp.ipynb
```

```
[NbConvertApp] Converting notebook Copy_of_fin_analytics_try_2_j_comp.ipynb to
pdf
[NbConvertApp] Support files will be in
Copy_of_fin_analytics_try_2_j_comp_files/
[NbConvertApp] Making directory ./Copy_of_fin_analytics_try_2_j_comp_files
[NbConvertApp] Making directory ./Copy_of_fin_analytics_try_2_j_comp_files
```

```
[NbConvertApp] Making directory ./Copy_of_fin_analytics_try_2_j_comp_files
[NbConvertApp] Making directory ./Copy_of_fin_analytics_try_2_j_comp_files
[NbConvertApp] Making directory ./Copy_of_fin_analytics_try_2_j_comp_files
[NbConvertApp] Making directory ./Copy_of_fin_analytics_try_2_j_comp_files
[NbConvertApp] Making directory ./Copy_of_fin_analytics_try_2_j_comp_files
[NbConvertApp] Making directory ./Copy_of_fin_analytics_try_2_j_comp_files
[NbConvertApp] Making directory ./Copy_of_fin_analytics_try_2_j_comp_files
[NbConvertApp] Making directory ./Copy_of_fin_analytics_try_2_j_comp_files
[NbConvertApp] Making directory ./Copy_of_fin_analytics_try_2_j_comp_files
[NbConvertApp] Making directory ./Copy_of_fin_analytics_try_2_j_comp_files
[NbConvertApp] Making directory ./Copy_of_fin_analytics_try_2_j_comp_files
[NbConvertApp] Making directory ./Copy_of_fin_analytics_try_2_j_comp_files
[NbConvertApp] Making directory ./Copy_of_fin_analytics_try_2_j_comp_files
[NbConvertApp] Making directory ./Copy_of_fin_analytics_try_2_j_comp_files
[NbConvertApp] Making directory ./Copy_of_fin_analytics_try_2_j_comp_files
[NbConvertApp] Making directory ./Copy_of_fin_analytics_try_2_j_comp_files
[NbConvertApp] Making directory ./Copy_of_fin_analytics_try_2_j_comp_files
[NbConvertApp] Making directory ./Copy_of_fin_analytics_try_2_j_comp_files
[NbConvertApp] Making directory ./Copy_of_fin_analytics_try_2_j_comp_files
[NbConvertApp] Making directory ./Copy_of_fin_analytics_try_2_j_comp_files
[NbConvertApp] Making directory ./Copy_of_fin_analytics_try_2_j_comp_files
[NbConvertApp] Making directory ./Copy_of_fin_analytics_try_2_j_comp_files
[NbConvertApp] Making directory ./Copy_of_fin_analytics_try_2_j_comp_files
[NbConvertApp] Making directory ./Copy_of_fin_analytics_try_2_j_comp_files
[NbConvertApp] Making directory ./Copy_of_fin_analytics_try_2_j_comp_files
[NbConvertApp] Making directory ./Copy_of_fin_analytics_try_2_j_comp_files
[NbConvertApp] Making directory ./Copy_of_fin_analytics_try_2_j_comp_files
[NbConvertApp] Making directory ./Copy_of_fin_analytics_try_2_j_comp_files
[NbConvertApp] Making directory ./Copy_of_fin_analytics_try_2_j_comp_files
[NbConvertApp] Making directory ./Copy_of_fin_analytics_try_2_j_comp_files
[NbConvertApp] Making directory ./Copy_of_fin_analytics_try_2_j_comp_files
[NbConvertApp] Making directory ./Copy_of_fin_analytics_try_2_j_comp_files
[NbConvertApp] Making directory ./Copy_of_fin_analytics_try_2_j_comp_files
[NbConvertApp] Writing 223293 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
Traceback (most recent call last):
  File "/usr/local/bin/jupyter-nbconvert", line 8, in <module>
    sys.exit(main())
  File "/usr/local/lib/python3.10/dist-packages/jupyter_core/application.py",
line 283, in launch_instance
    super().launch_instance(argv=argv, **kwargs)
  File "/usr/local/lib/python3.10/dist-
packages/traitlets/config/application.py", line 992, in launch_instance
    app.start()
  File "/usr/local/lib/python3.10/dist-packages/nbconvert/nbconvertapp.py", line
423, in start
    self.convert_notebooks()
```

```
  File "/usr/local/lib/python3.10/dist-packages/nbconvert/nbconvertapp.py", line
597, in convert_notebooks
    self.convert_single_notebook(notebook_filename)
  File "/usr/local/lib/python3.10/dist-packages/nbconvert/nbconvertapp.py", line
560, in convert_single_notebook
    output, resources = self.export_single_notebook(
  File "/usr/local/lib/python3.10/dist-packages/nbconvert/nbconvertapp.py", line
488, in export_single_notebook
    output, resources = self.exporter.from_filename(
  File "/usr/local/lib/python3.10/dist-
packages/nbconvert/exporters/exporter.py", line 189, in from_filename
    return self.from_file(f, resources=resources, **kw)
  File "/usr/local/lib/python3.10/dist-
packages/nbconvert/exporters/exporter.py", line 206, in from_file
    return self.from_notebook_node(
  File "/usr/local/lib/python3.10/dist-packages/nbconvert/exporters/pdf.py",
line 194, in from_notebook_node
    self.run_latex(tex_file)
  File "/usr/local/lib/python3.10/dist-packages/nbconvert/exporters/pdf.py",
line 164, in run_latex
    return self.run_command(
  File "/usr/local/lib/python3.10/dist-packages/nbconvert/exporters/pdf.py",
line 139, in run_command
    out, _ = p.communicate()
  File "/usr/lib/python3.10/subprocess.py", line 1141, in communicate
    stdout = self.stdout.read()
KeyboardInterrupt
^C
```

[ ]:
```
!jupyter nbconvert Copy_of_fin_analytics_try_2_j_comp.ipynb --to html
```

```
[NbConvertApp] Converting notebook Copy_of_fin_analytics_try_2_j_comp.ipynb to
html
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/nbformat/reader.py", line 19, in
parse_json
    nb_dict = json.loads(s, **kwargs)
  File "/usr/lib/python3.10/json/__init__.py", line 346, in loads
    return _default_decoder.decode(s)
  File "/usr/lib/python3.10/json/decoder.py", line 340, in decode
    raise JSONDecodeError("Extra data", s, end)
json.decoder.JSONDecodeError: Extra data: line 3506 column 2 (char 3906009)

The above exception was the direct cause of the following exception:

Traceback (most recent call last):
  File "/usr/local/bin/jupyter-nbconvert", line 8, in <module>
    sys.exit(main())
```

```
  File "/usr/local/lib/python3.10/dist-packages/jupyter_core/application.py",
line 283, in launch_instance
    super().launch_instance(argv=argv, **kwargs)
  File "/usr/local/lib/python3.10/dist-
packages/traitlets/config/application.py", line 992, in launch_instance
    app.start()
  File "/usr/local/lib/python3.10/dist-packages/nbconvert/nbconvertapp.py", line
423, in start
    self.convert_notebooks()
  File "/usr/local/lib/python3.10/dist-packages/nbconvert/nbconvertapp.py", line
597, in convert_notebooks
    self.convert_single_notebook(notebook_filename)
  File "/usr/local/lib/python3.10/dist-packages/nbconvert/nbconvertapp.py", line
560, in convert_single_notebook
    output, resources = self.export_single_notebook(
  File "/usr/local/lib/python3.10/dist-packages/nbconvert/nbconvertapp.py", line
488, in export_single_notebook
    output, resources = self.exporter.from_filename(
  File "/usr/local/lib/python3.10/dist-
packages/nbconvert/exporters/exporter.py", line 189, in from_filename
    return self.from_file(f, resources=resources, **kw)
  File "/usr/local/lib/python3.10/dist-
packages/nbconvert/exporters/exporter.py", line 207, in from_file
    nbformat.read(file_stream, as_version=4), resources=resources, **kw
  File "/usr/local/lib/python3.10/dist-packages/nbformat/__init__.py", line 174,
in read
    return reads(buf, as_version, capture_validation_error, **kwargs)
  File "/usr/local/lib/python3.10/dist-packages/nbformat/__init__.py", line 92,
in reads
    nb = reader.reads(s, **kwargs)
  File "/usr/local/lib/python3.10/dist-packages/nbformat/reader.py", line 75, in
reads
    nb_dict = parse_json(s, **kwargs)
  File "/usr/local/lib/python3.10/dist-packages/nbformat/reader.py", line 25, in
parse_json
    raise NotJSONError(message) from e
nbformat.reader.NotJSONError: Notebook does not appear to be JSON: '{\n
"cells": [\n    {\n      "cell_typ…
```

[ ]:

```

```