```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from google.colab import drive
import os

drive.mount("/content/drive", force_remount=True)

# Make sure to replace 'MyDrive/path/to/your/file.txt' with the actual path
file_path = '/content/drive/MyDrive/Colab Notebooks/MSDS 422/Module 1/bangladesh_data.csv'
```

```
Mounted at /content/drive
```

1. EDA on Bangladesh's Socio-economic data

```
# Load dataset
data = pd.read_csv(file_path)
data.columns = data.columns.str.strip()  # Clean column names

# Display basic information
print("Dataset Shape:", data.shape)

print("\nBasic Info:")
data.info()

print("\nSummary Statistics:")
print(data.describe())

print("\nFirst Few Rows:")
print(data.head())

print("\nMissing Values:")
print(data.isnull().sum())
```

Show hidden output

```
# Columns of interest for analysis
columns_of_interest = [
                    "Year",
                    "Poverty Rate (National)",
                    "Rural Poverty Rate",
                    "Urban Poverty Rate",
                    "Literacy Rate(%)",
                    "GDP",
                    "Education Spending (% of GDP)",
                    "Healthcare Spending  Per Capita (US$)",
                    "Population",
                    "Urban Population % of Total",
                    "Unemployment Rate (%)",
                    "Inflation Rate (%)",
                    "Net Migration Rate",
                    "GDP Per Capita",
                    "Export Growth(%GDP)",
                    "Import Growth(%GDP)",
                    "Death Rate",
                    "Birth Rate",
                    "Infant Mortality Rate",
                    "Life Expectancy Growth Rate (%)",
                    "Annual % Crime Rate Change",
                    "green house gas emissions Annual % Change",
                    "Fossil Fuel consumption % of Total Energy Use",
                    "Maternal Mortality Rate Per 100K Live Births",
                    "Clean Water Access % of Population",
                    "Electricity Access % of Population",
                    "Suicide Rate"
]
```

```
len(columns_of_interest)
```

```
27
```

```
# calculate missing value % for each feature
missing_percentage = (data.isnull().sum() / len(data)) * 100
print(missing_percentage.sort_values())
```

```
Year                                          0.000000
Infant Mortality Rate                         0.000000
Growth Rate.2                                 0.000000
Unemployment Rate (%)                         0.000000
Annual Change (%).1                           0.000000
                                                ...
tourism exports                              63.636364
Tourism spending                             63.636364
Annual Change.4                              75.000000
Coal consumption % of Electricity from Coal  75.000000
Number of Private Vehichles                 100.000000
Length: 118, dtype: float64
```

1.1 Preping Bangladesh's Socio-economic data and Feature Engineering data for further analysis

```
# Check if columns exist in the dataset
missing_columns = [col for col in columns_of_interest if col not in data.columns]
if missing_columns:
    print(f"\nColumns not found in the dataset: {missing_columns}")
else:
    # Subset and copy relevant data
    data_subset = data[columns_of_interest].copy()
    print("\nData Subset:")
    print(data_subset.head())

    # Clean and preprocess data
    # Handle percentage columns
    percentage_columns = ["Poverty Rate (National)", "Rural Poverty Rate", "Urban Poverty Rate", "Literacy Rate(%)", "Education
                          "Unemployment Rate (%)",
                    "Inflation Rate (%)",
                    "Net Migration Rate",
                    "Export Growth(%GDP)",
                    "Import Growth(%GDP)",
                    "Death Rate",
                    "Birth Rate",
                    "Infant Mortality Rate",
                    "Life Expectancy Growth Rate (%)",
                    "Annual % Crime Rate Change",
                    "green house gas emissions Annual % Change",
                    "Fossil Fuel consumption % of Total Energy Use",
                    "Maternal Mortality Rate Per 100K Live Births",
                    "Clean Water Access % of Population",
                    "Electricity Access % of Population",
                    "Suicide Rate"]
    for col in percentage_columns:
        if data_subset[col].dtype == 'object':
            data_subset[col] = data_subset[col].str.rstrip('%').astype('float') / 100.0

    # Clean GDP column
    if data_subset["GDP"].dtype == 'object':
        data_subset["GDP"] = data_subset["GDP"].replace('[\$,B]', '', regex=True).astype('float')
    if data_subset["GDP Per Capita"].dtype == 'object':
      data_subset["GDP Per Capita"] = data_subset["GDP Per Capita"].replace('[\$,B]', '', regex=True).astype('float')

    # Clean Healthcare Spending column
    if data_subset["Healthcare Spending  Per Capita (US$)"].dtype == 'object':
        data_subset["Healthcare Spending  Per Capita (US$)"] = data_subset["Healthcare Spending  Per Capita (US$)"].replace('[\$

    # Clean Population column
    if data_subset["Population"].dtype == 'object':
        data_subset["Population"] = data_subset["Population"].replace('[,]', '', regex=True).astype('float')

    # Rename columns for consistency
    data_subset.columns = [
        "Year", "National Poverty Rate", "Rural Poverty Rate", "Urban Poverty Rate",
        "Literacy Rate", "GDP", "Education Spending", "Healthcare Spending",
        "Population", "Urban Population (%)",
        "Unemployment Rate",
        "Inflation Rate",
```

```
        "Net Migration Rate",
        "GDP Per Capita",
        "Export Growth",
        "Import Growth",
        "Death Rate",
        "Birth Rate",
        "Infant Mortality Rate",
        "Life Expectancy Growth Rate",
        "Annual Crime Rate Change",
        "Greenhouse Gas Emissions Change",
        "Fossil Fuel Consumption",
        "Maternal Mortality Rate",
        "Clean Water Access",
        "Electricity Access",
        "Suicide Rate"
    ]

    # Impute missing values
    imputer = SimpleImputer(strategy="mean")
    data_subset.iloc[:, 1:] = imputer.fit_transform(data_subset.iloc[:, 1:])

    # Feature Scaling
    scaler = StandardScaler()
    scaled_data = scaler.fit_transform(data_subset.iloc[:, 1:])
    scaled_data = pd.DataFrame(scaled_data, columns=data_subset.columns[1:])
    data_subset = pd.concat([data_subset["Year"], scaled_data], axis=1)

    # Feature Engineering
    data_subset["Urbanization × GDP Growth"] = data_subset["Urban Population (%)"] * data_subset["GDP"]
    data_subset["3-Year GDP Rolling Avg"] = data_subset["GDP"].rolling(window=3).mean()
    data_subset["3-Year Literacy Rolling Avg"] = data_subset["Literacy Rate"].rolling(window=3).mean()
```

Show hidden output

```
# do the same data processing but keep original values without scaling

# Check if columns exist in the dataset
missing_columns = [col for col in columns_of_interest if col not in data.columns]
if missing_columns:
    print(f"\nColumns not found in the dataset: {missing_columns}")
else:
    # Subset and copy relevant data
    data_original = data[columns_of_interest].copy()
    print("\nData Subset:")
    print(data_original.head())

    # Clean and preprocess data
    # Handle percentage columns
    percentage_columns = ["Poverty Rate (National)", "Rural Poverty Rate", "Urban Poverty Rate", "Literacy Rate(%)", "Education
                      "Unemployment Rate (%)",
                "Inflation Rate (%)",
                "Net Migration Rate",
                "Export Growth(%GDP)",
                "Import Growth(%GDP)",
                "Death Rate",
                "Birth Rate",
                "Infant Mortality Rate",
                "Life Expectancy Growth Rate (%)",
                "Annual % Crime Rate Change",
                "green house gas emissions Annual % Change",
                "Fossil Fuel consumption % of Total Energy Use",
                "Maternal Mortality Rate Per 100K Live Births",
                "Clean Water Access % of Population",
                "Electricity Access % of Population",
                "Suicide Rate"]
    for col in percentage_columns:
        if data_original[col].dtype == 'object':
            data_original[col] = data_original[col].str.rstrip('%').astype('float') / 100.0

    # Clean GDP column
    if data_original["GDP"].dtype == 'object':
        data_original["GDP"] = data_original["GDP"].replace('[\$,B]', '', regex=True).astype('float')
    if data_original["GDP Per Capita"].dtype == 'object':
      data_original["GDP Per Capita"] = data_original["GDP Per Capita"].replace('[\$,B]', '', regex=True).astype('float')

    # Clean Healthcare Spending column
```

```
    if data_original["Healthcare Spending  Per Capita (US$)"].dtype == 'object':
        data_original["Healthcare Spending  Per Capita (US$)"] = data_original["Healthcare Spending  Per Capita (US$)"].replace(

    # Clean Population column
    if data_original["Population"].dtype == 'object':
        data_original["Population"] = data_original["Population"].replace('[,]', '', regex=True).astype('float')

    # Rename columns for consistency
    data_original.columns = [
        "Year", "National Poverty Rate", "Rural Poverty Rate", "Urban Poverty Rate",
        "Literacy Rate", "GDP", "Education Spending", "Healthcare Spending",
        "Population", "Urban Population (%)",
        "Unemployment Rate",
        "Inflation Rate",
        "Net Migration Rate",
        "GDP Per Capita",
        "Export Growth",
        "Import Growth",
        "Death Rate",
        "Birth Rate",
        "Infant Mortality Rate",
        "Life Expectancy Growth Rate",
        "Annual Crime Rate Change",
        "Greenhouse Gas Emissions Change",
        "Fossil Fuel Consumption",
        "Maternal Mortality Rate",
        "Clean Water Access",
        "Electricity Access",
        "Suicide Rate"
    ]

    # Impute missing values
    imputer = SimpleImputer(strategy="mean")
    data_original.iloc[:, 1:] = imputer.fit_transform(data_original.iloc[:, 1:])

    # Feature Engineering
    data_original["Urbanization × GDP Growth"] = data_original["Urban Population (%)"] * data_original["GDP"]
    data_original["3-Year GDP Rolling Avg"] = data_original["GDP"].rolling(window=3).mean()
    data_original["3-Year Literacy Rolling Avg"] = data_original["Literacy Rate"].rolling(window=3).mean()
```

Show hidden output

```
print(data_subset.shape)
print(data_original.shape)
```

Show hidden output

```
# create a subset of data with columns_of_interest
data_original_2 = data[columns_of_interest]

# calculate missing value % for each feature
missing_percentage = (data_original_2.isnull().sum() / len(data_original_2)) * 100
print(missing_percentage)
```

```
Year                                           0.000000
Poverty Rate (National)                        0.000000
Rural Poverty Rate                             0.000000
Urban Poverty Rate                             0.000000
Literacy Rate(%)                               0.000000
GDP                                            0.000000
Education Spending (% of GDP)                 18.181818
Healthcare Spending  Per Capita (US$)         50.000000
Population                                     0.000000
Urban Population % of Total                    0.000000
Unemployment Rate (%)                          0.000000
Inflation Rate (%)                             0.000000
Net Migration Rate                             2.272727
GDP Per Capita                                 0.000000
Export Growth(%GDP)                            0.000000
Import Growth(%GDP)                            0.000000
Death Rate                                     0.000000
Birth Rate                                     0.000000
Infant Mortality Rate                          0.000000
Life Expectancy Growth Rate (%)                0.000000
Annual % Crime Rate Change                    56.818182
green house gas emissions Annual % Change     29.545455
Fossil Fuel consumption % of Total Energy Use 20.454545
Maternal Mortality Rate Per 100K Live Births  52.272727
```

```
Clean Water Access % of Population                47.727273
Electricity Access % of Population                27.272727
Suicide Rate                                      54.545455
dtype: float64
```

```
# calculate missing value % for each feature from imputed dataset
missing_percentage = (data_original.isnull().sum() / len(data_original)) * 100
print(missing_percentage)
```

```
Year                               0.000000
National Poverty Rate              0.000000
Rural Poverty Rate                 0.000000
Urban Poverty Rate                 0.000000
Literacy Rate                      0.000000
GDP                                0.000000
Education Spending                 0.000000
Healthcare Spending                0.000000
Population                         0.000000
Urban Population (%)               0.000000
Unemployment Rate                  0.000000
Inflation Rate                     0.000000
Net Migration Rate                 0.000000
GDP Per Capita                     0.000000
Export Growth                      0.000000
Import Growth                      0.000000
Death Rate                         0.000000
Birth Rate                         0.000000
Infant Mortality Rate              0.000000
Life Expectancy Growth Rate        0.000000
Annual Crime Rate Change           0.000000
Greenhouse Gas Emissions Change    0.000000
Fossil Fuel Consumption            0.000000
Maternal Mortality Rate            0.000000
Clean Water Access                 0.000000
Electricity Access                 0.000000
Suicide Rate                       0.000000
Urbanization × GDP Growth          0.000000
3–Year GDP Rolling Avg             4.545455
3–Year Literacy Rolling Avg        4.545455
dtype: float64
```

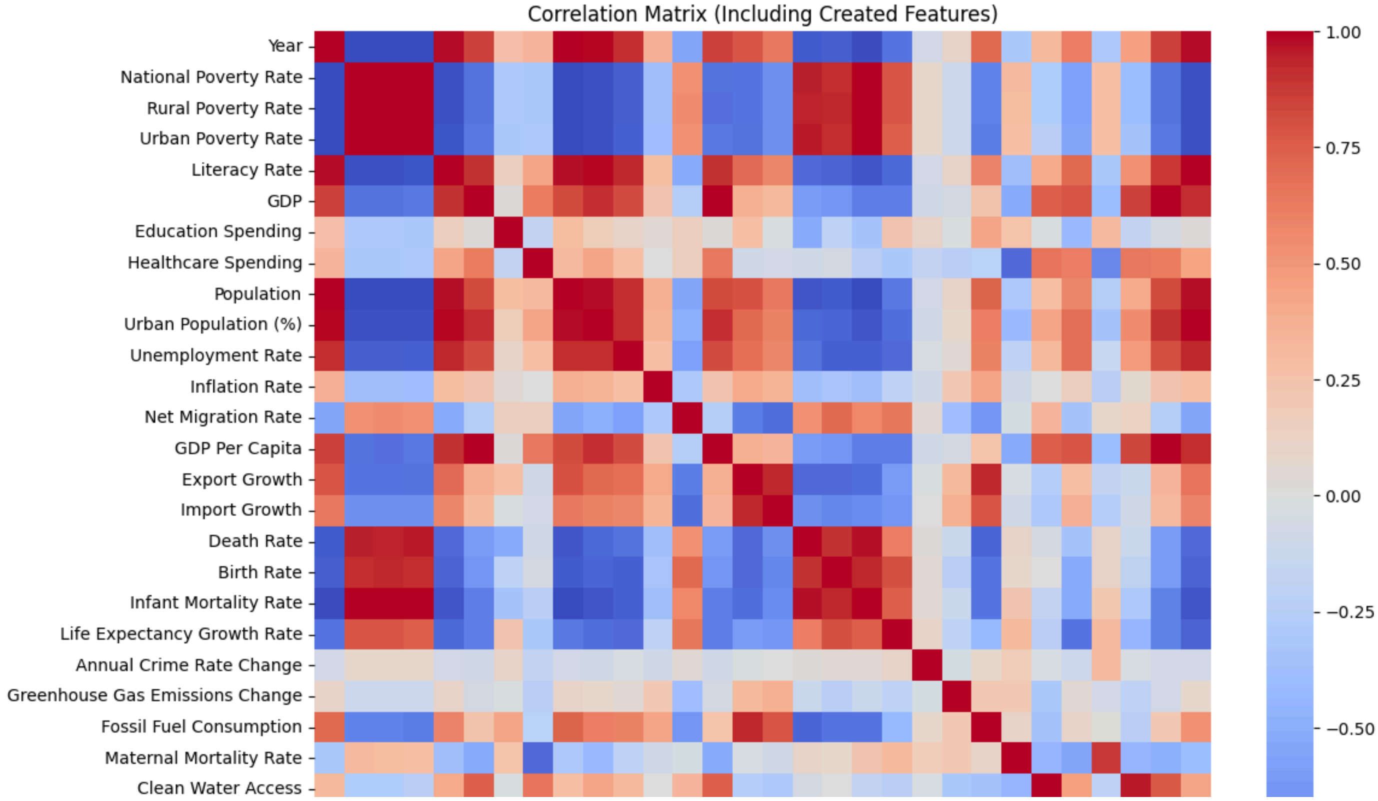
```
data_original['Healthcare Spending'].value_counts()
```

Healthcare Spending	count
25.545455	22
12.000000	2
9.000000	2
27.000000	1
51.000000	1
48.000000	1
45.000000	1
42.000000	1
40.000000	1
34.000000	1
30.000000	1
23.000000	1
24.000000	1
21.000000	1
18.000000	1
16.000000	1
14.000000	1
11.000000	1
10.000000	1
8.000000	1
58.000000	1

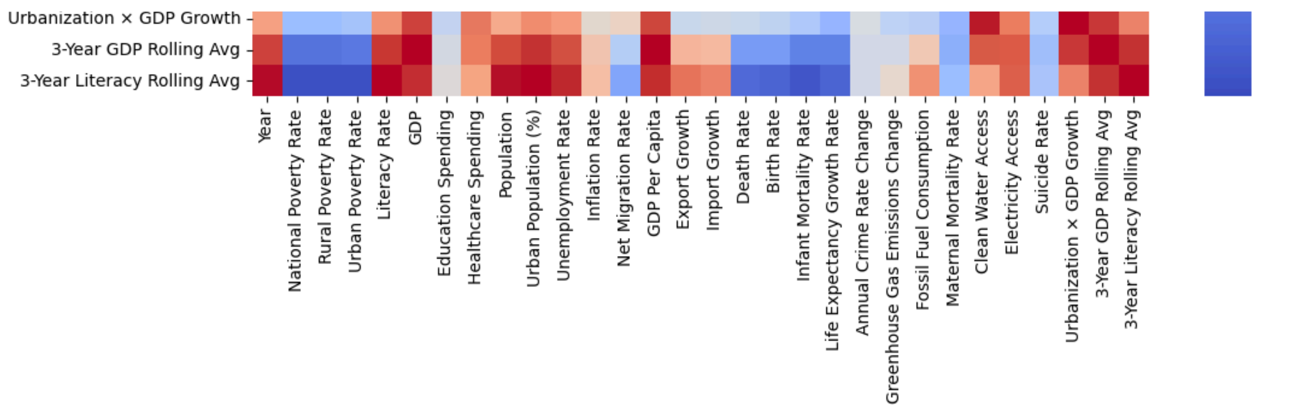
dtype: int64

1.2 Correlation Analysis on Features for data relations

```
# Exploratory Data Analysis
# Correlation Matrix
correlation_matrix = data_subset.corr()
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=False, cmap="coolwarm", fmt=".2f")
plt.title("Correlation Matrix (Including Created Features)")
plt.show()
```




```
# check correlation with National Poverty Rate
correlation_matrix["National Poverty Rate"].sort_values(ascending=False)
```



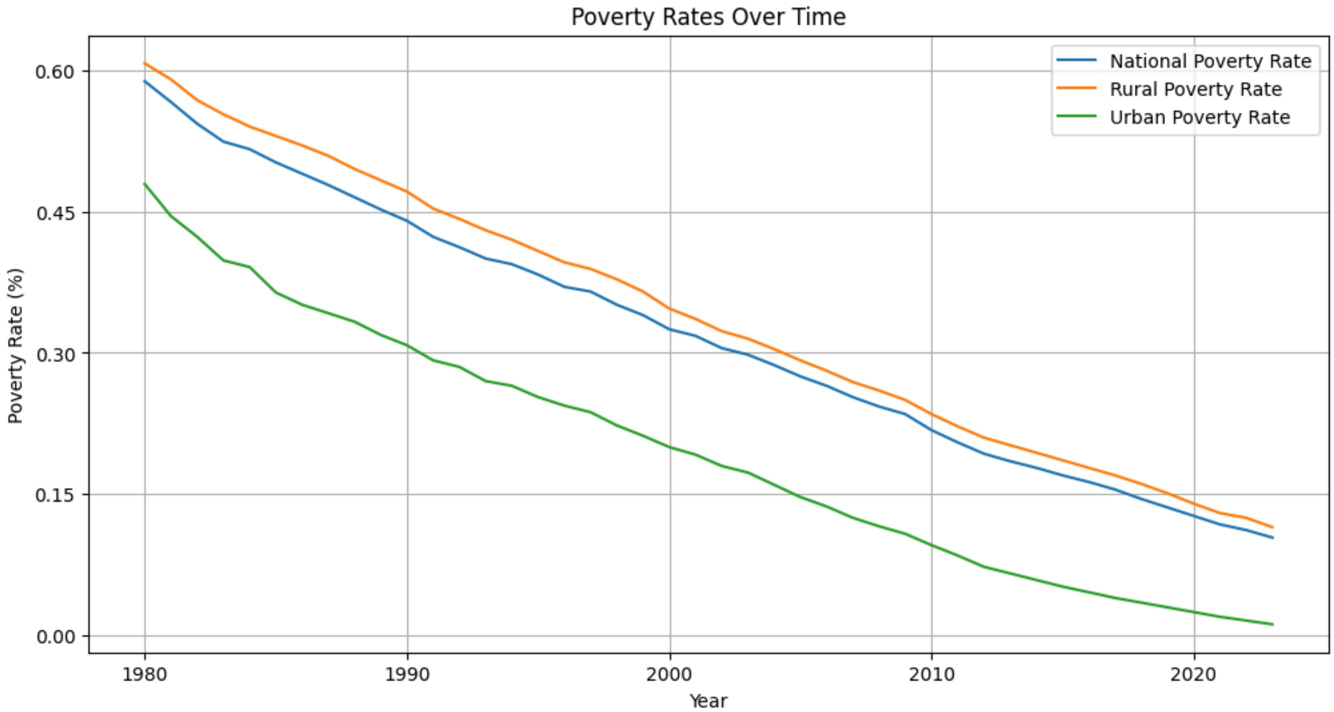
	National Poverty Rate
National Poverty Rate	1.000000
Rural Poverty Rate	0.999709
Urban Poverty Rate	0.998054
Infant Mortality Rate	0.992928
Death Rate	0.947170
Birth Rate	0.917431
Life Expectancy Growth Rate	0.775513
Net Migration Rate	0.535455
Maternal Mortality Rate	0.302007
Suicide Rate	0.276712
Annual Crime Rate Change	0.071631
Greenhouse Gas Emissions Change	-0.102458
Clean Water Access	-0.275660
Education Spending	-0.301818
Healthcare Spending	-0.311259
Inflation Rate	-0.373114
Urbanization × GDP Growth	-0.396160
Electricity Access	-0.574456
Import Growth	-0.652610
Fossil Fuel Consumption	-0.740503
Export Growth	-0.799517
GDP	-0.814500
GDP Per Capita	-0.818170
3-Year GDP Rolling Avg	-0.819651
Unemployment Rate	-0.907094
Literacy Rate	-0.968302
3-Year Literacy Rolling Avg	-0.973785
Urban Population (%)	-0.979221
Year	-0.997426
Population	-0.998727

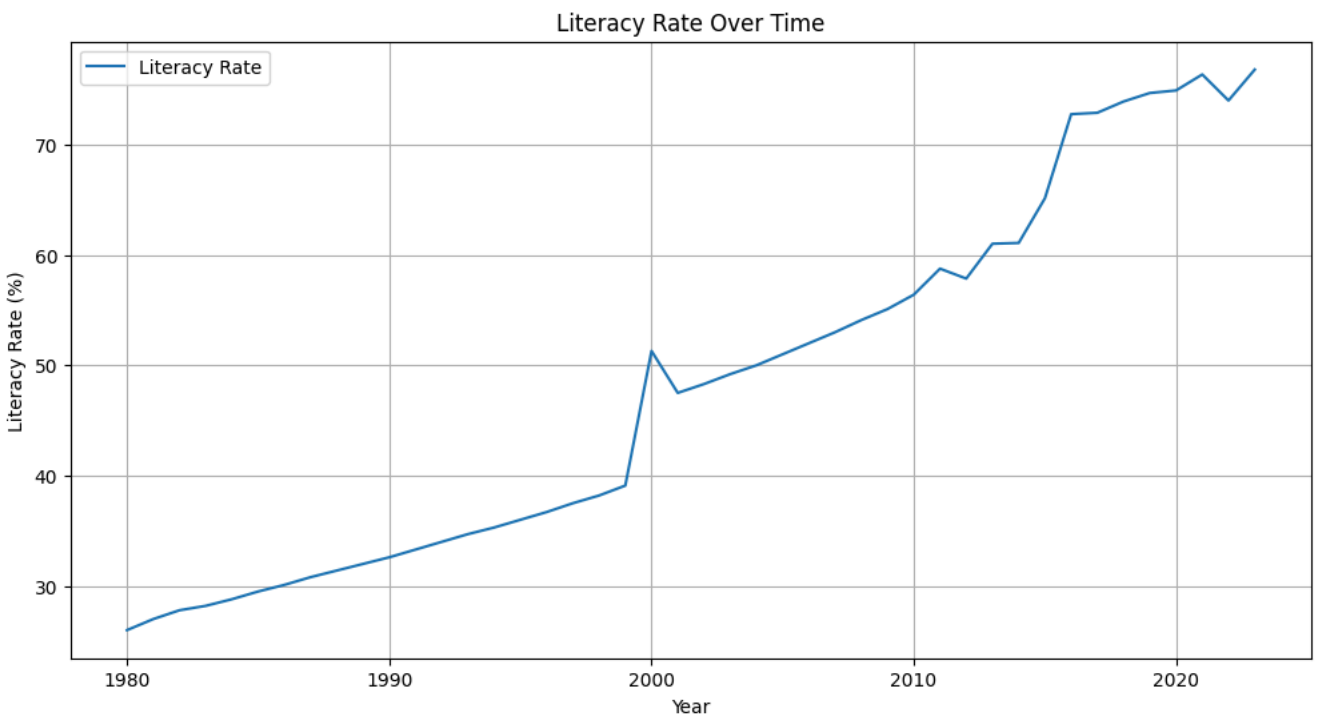
dtype: float64

1.3 Feature trend graph analysis

```
# Trend Analysis for original data
import numpy as np
from matplotlib.ticker import MaxNLocator

plt.figure(figsize=(12, 6))
sns.lineplot(data=data_original, x="Year", y="National Poverty Rate", label="National Poverty Rate")
sns.lineplot(data=data_original, x="Year", y="Rural Poverty Rate", label="Rural Poverty Rate")
sns.lineplot(data=data_original, x="Year", y="Urban Poverty Rate", label="Urban Poverty Rate")
plt.title("Poverty Rates Over Time")
plt.xlabel("Year")
plt.ylabel("Poverty Rate (%)")
plt.gca().yaxis.set_major_locator(MaxNLocator(integer=True, prune='lower', nbins=5))
plt.legend()
plt.grid(True)
plt.show()

plt.figure(figsize=(12, 6))
sns.lineplot(data=data_original, x="Year", y="Literacy Rate", label="Literacy Rate")
plt.title("Literacy Rate Over Time")
```

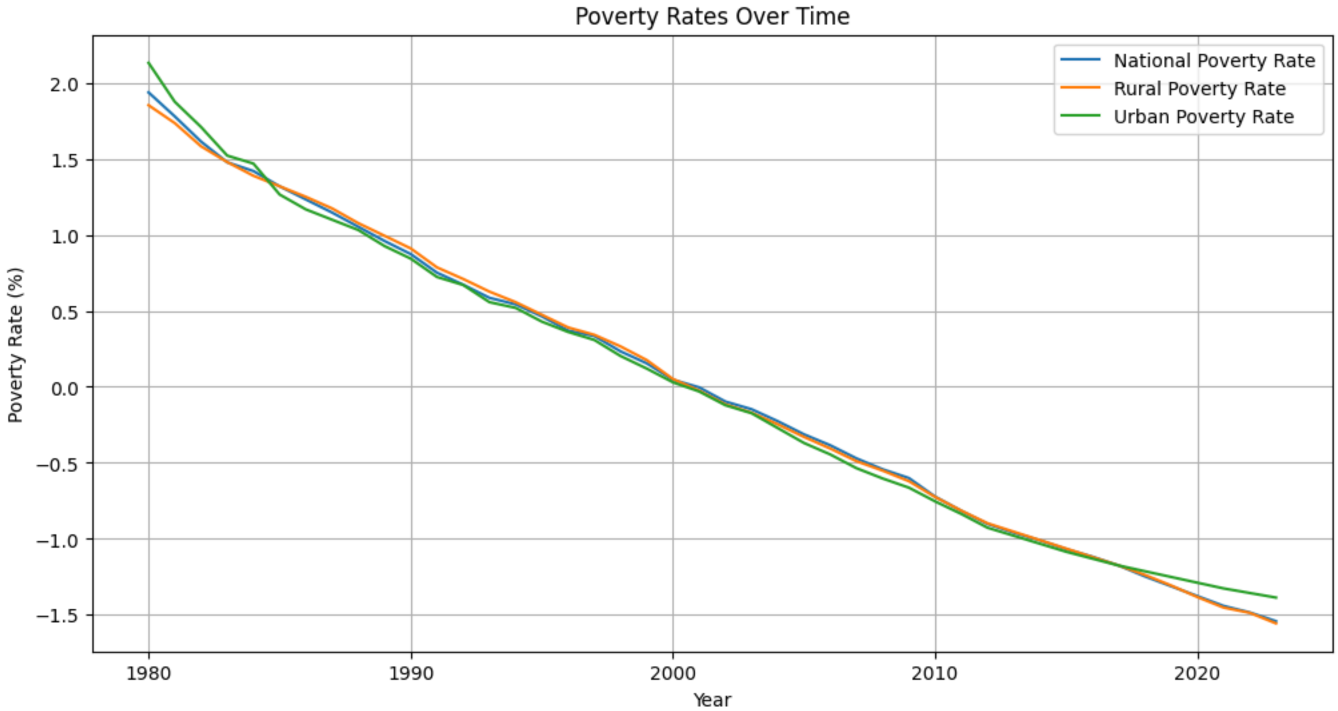
```
plt.xlabel("Year")
plt.ylabel("Literacy Rate (%)")
plt.grid(True)
plt.show()
```

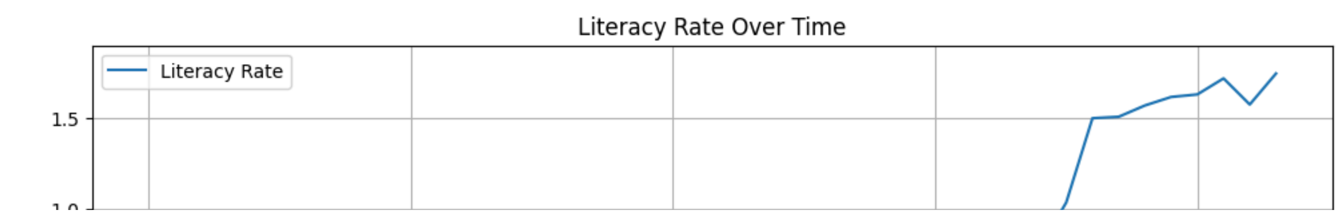




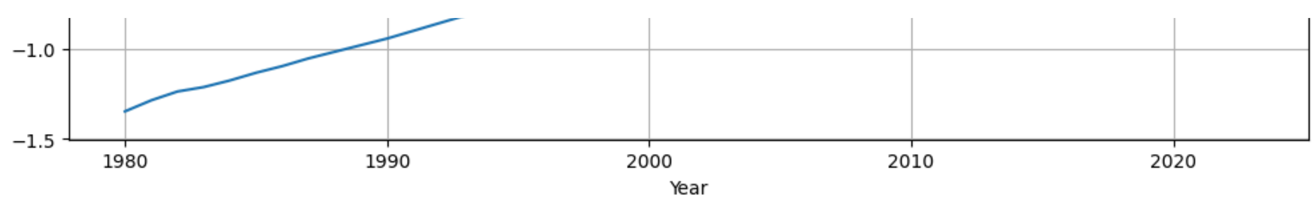

```
# Trend Analysis
plt.figure(figsize=(12, 6))
sns.lineplot(data=data_subset, x="Year", y="National Poverty Rate", label="National Poverty Rate")
sns.lineplot(data=data_subset, x="Year", y="Rural Poverty Rate", label="Rural Poverty Rate")
sns.lineplot(data=data_subset, x="Year", y="Urban Poverty Rate", label="Urban Poverty Rate")
plt.title("Poverty Rates Over Time")
plt.xlabel("Year")
plt.ylabel("Poverty Rate (%)")
plt.legend()
plt.grid(True)
plt.show()

plt.figure(figsize=(12, 6))
```

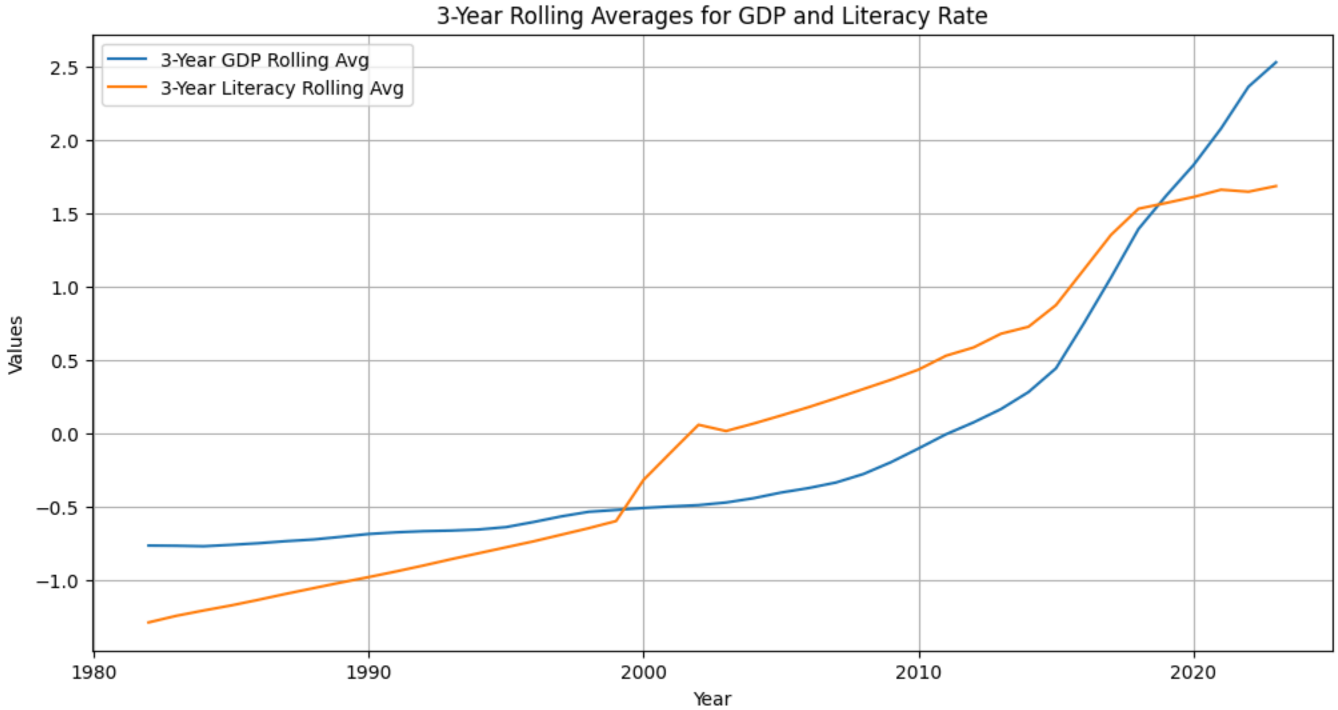
```
sns.lineplot(data=data_subset, x="Year", y="Literacy Rate", label="Literacy Rate")
plt.title("Literacy Rate Over Time")
plt.xlabel("Year")
plt.ylabel("Literacy Rate (%)")
plt.grid(True)
plt.show()
```



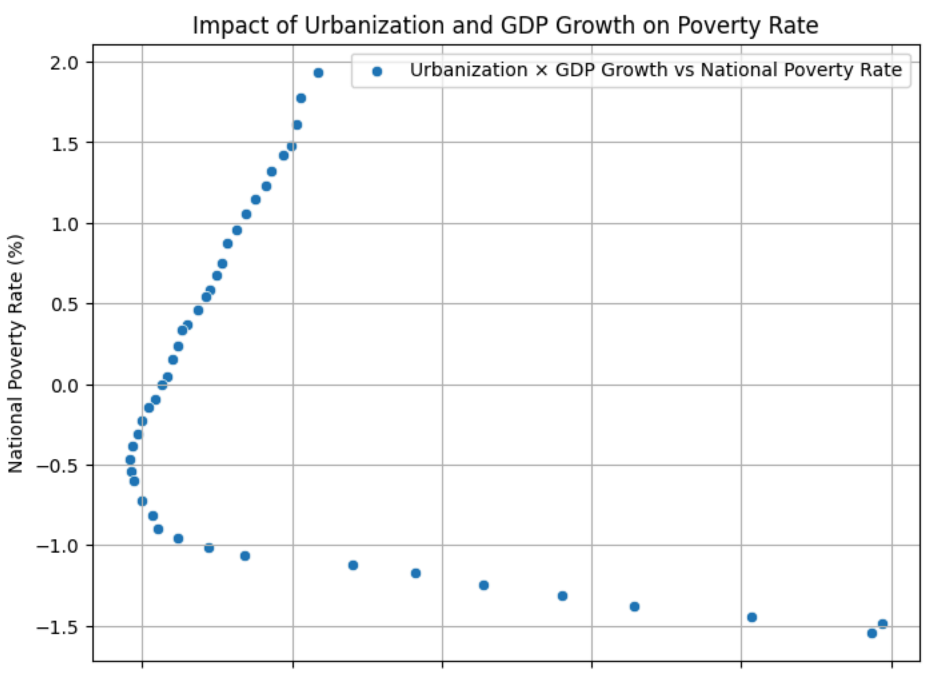



```
# Visualize Rolling Averages
plt.figure(figsize=(12, 6))
sns.lineplot(data=data_subset, x="Year", y="3-Year GDP Rolling Avg", label="3-Year GDP Rolling Avg")
sns.lineplot(data=data_subset, x="Year", y="3-Year Literacy Rolling Avg", label="3-Year Literacy Rolling Avg")
plt.title("3-Year Rolling Averages for GDP and Literacy Rate")
plt.xlabel("Year")
plt.ylabel("Values")
plt.legend()
plt.grid(True)
plt.show()
```



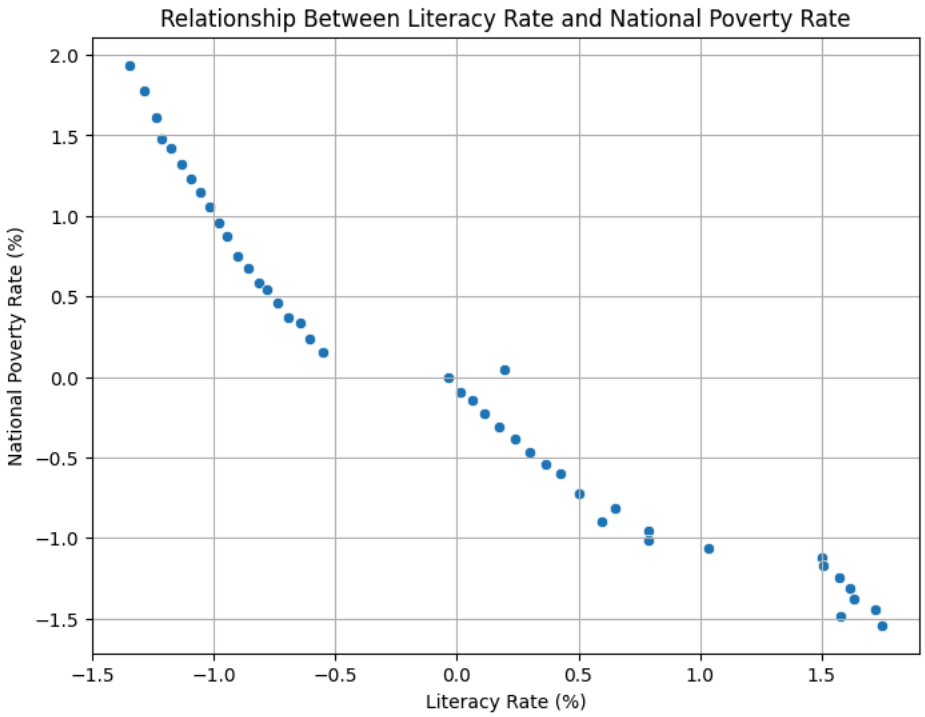


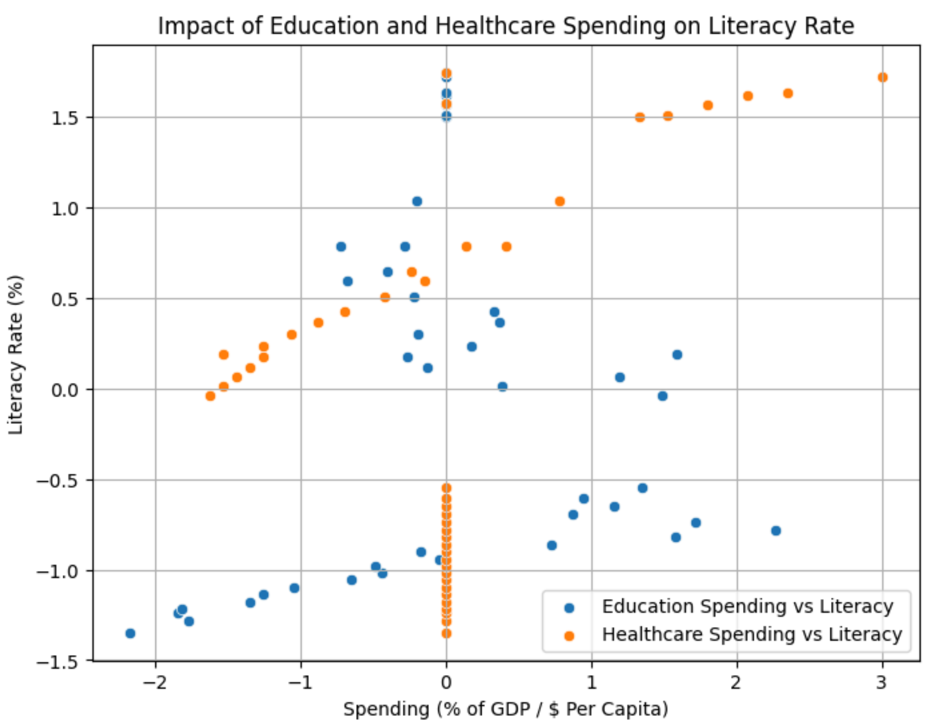

```
# Scatterplot for Interaction Term
plt.figure(figsize=(8, 6))
sns.scatterplot(data=data_subset, x="Urbanization × GDP Growth", y="National Poverty Rate", label="Urbanization × GDP Growth
plt.title("Impact of Urbanization and GDP Growth on Poverty Rate")
plt.xlabel("Urbanization × GDP Growth")
plt.ylabel("National Poverty Rate (%)")
plt.grid(True)
plt.show()
```




```
# Scatterplots to Analyze Relationships
plt.figure(figsize=(8, 6))
sns.scatterplot(data=data_subset, x="Literacy Rate", y="National Poverty Rate")
plt.title("Relationship Between Literacy Rate and National Poverty Rate")
plt.xlabel("Literacy Rate (%)")
plt.ylabel("National Poverty Rate (%)")
plt.grid(True)
```

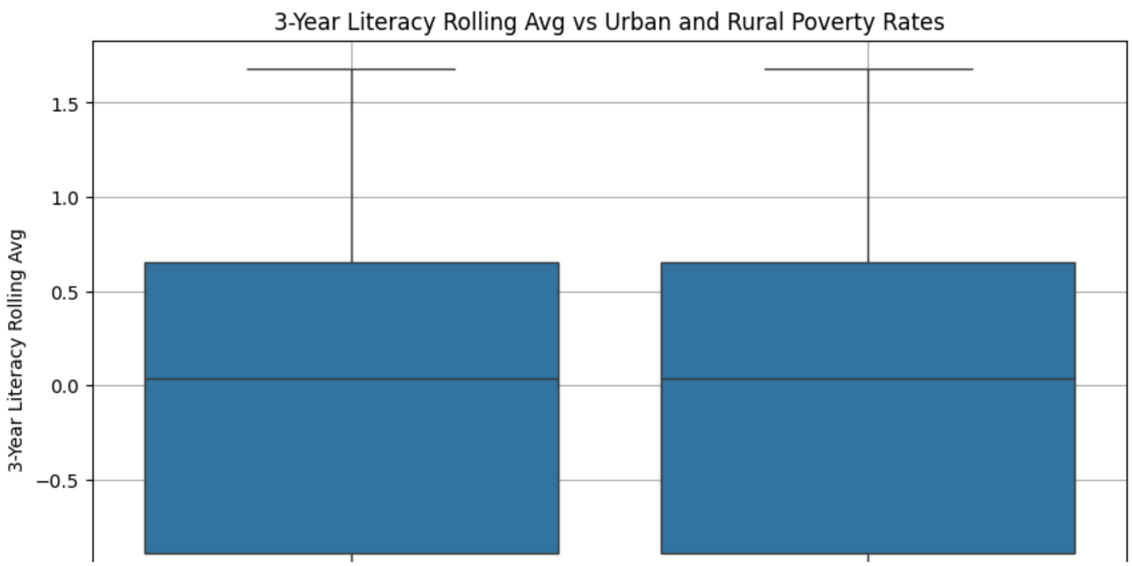
```
plt.show()

plt.figure(figsize=(8, 6))
sns.scatterplot(data=data_subset, x="Education Spending", y="Literacy Rate", label="Education Spending vs Literacy")
sns.scatterplot(data=data_subset, x="Healthcare Spending", y="Literacy Rate", label="Healthcare Spending vs Literacy")
plt.title("Impact of Education and Healthcare Spending on Literacy Rate")
plt.xlabel("Spending (% of GDP / $ Per Capita)")
plt.ylabel("Literacy Rate (%)")
plt.legend()
plt.grid(True)
plt.show()
```



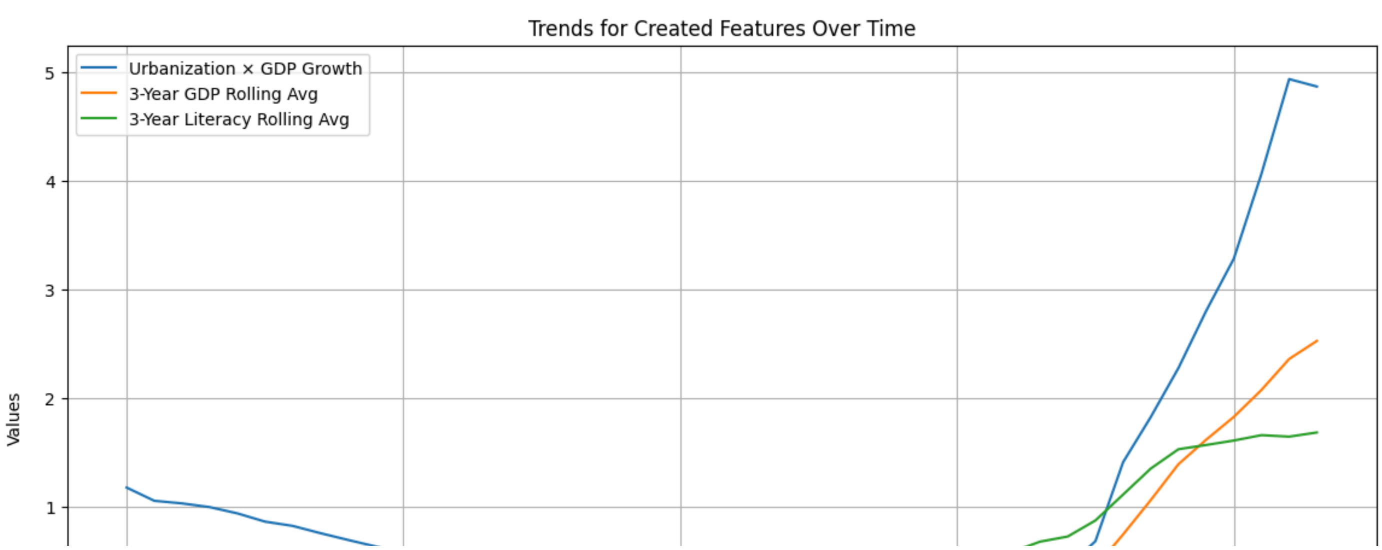



```
# Boxplot for Rolling Literacy Trends vs Poverty Rates
data_subset_melted = data_subset.melt(
    id_vars=["Year", "3-Year Literacy Rolling Avg"],
    value_vars=["Urban Poverty Rate", "Rural Poverty Rate"],
```

```
        var_name="Poverty Type",
        value_name="Poverty Rate"
    )
    plt.figure(figsize=(10, 6))
    sns.boxplot(data=data_subset_melted, x="Poverty Type", y="3-Year Literacy Rolling Avg")
    plt.title("3-Year Literacy Rolling Avg vs Urban and Rural Poverty Rates")
    plt.xlabel("Poverty Type")
    plt.ylabel("3-Year Literacy Rolling Avg")
    plt.grid(True)
    plt.show()
```

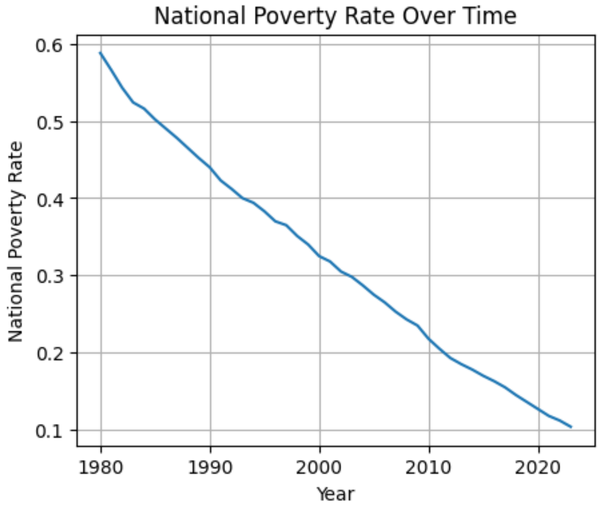


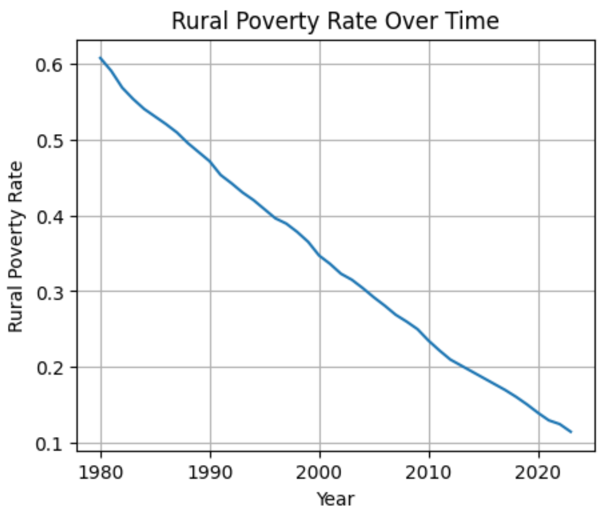

```
    # Overlay Trends for Created Features
    plt.figure(figsize=(14, 8))
    sns.lineplot(data=data_subset, x="Year", y="Urbanization × GDP Growth", label="Urbanization × GDP Growth")
    sns.lineplot(data=data_subset, x="Year", y="3-Year GDP Rolling Avg", label="3-Year GDP Rolling Avg")
    sns.lineplot(data=data_subset, x="Year", y="3-Year Literacy Rolling Avg", label="3-Year Literacy Rolling Avg")
    plt.title("Trends for Created Features Over Time")
    plt.xlabel("Year")
    plt.ylabel("Values")
    plt.legend()
    plt.grid(True)
    plt.show()
```

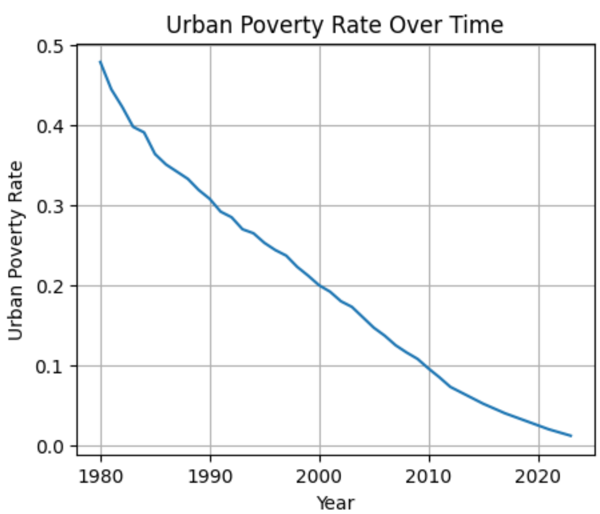


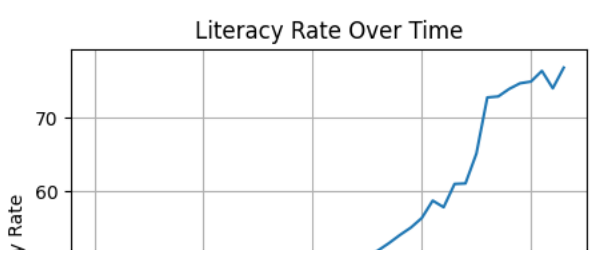

```
# create a chart for each feature over time
for col in data_subset.columns[1:]:
    plt.figure(figsize=(5, 4))
    sns.lineplot(data=data_original, x="Year", y=col)
    plt.title(f"{col} Over Time")
    plt.xlabel("Year")
    plt.ylabel(col)
    plt.grid(True)
```

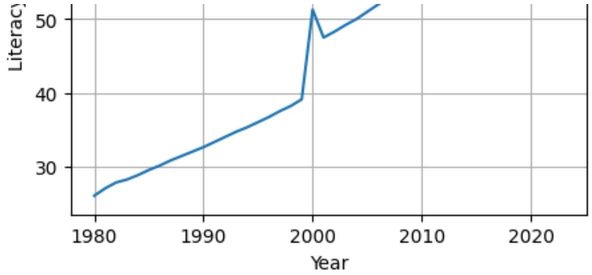
```
    plt.show()
    print("\n")
```

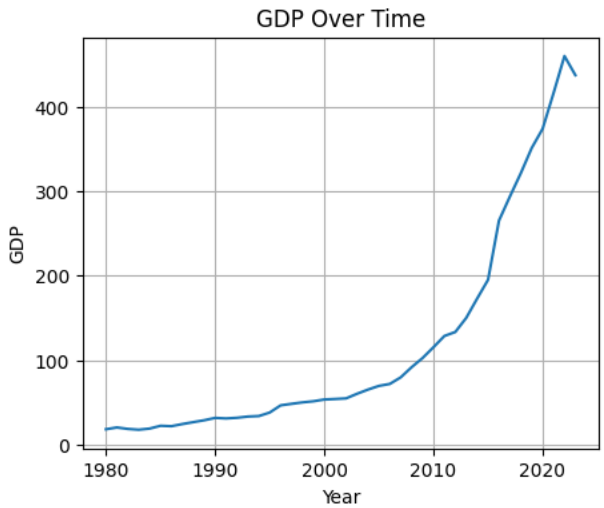

National Poverty Rate Over Time
National Poverty Rate
0.6
0.5
0.4
0.3
0.2
0.1
1980
1990
2000
2010
2020
Year

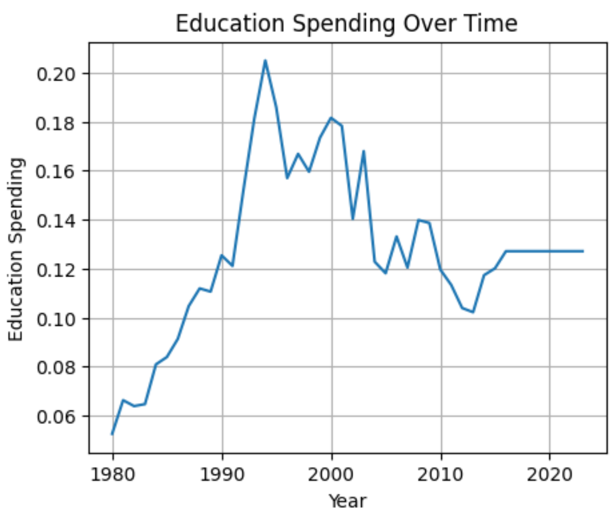

Rural Poverty Rate Over Time
Rural Poverty Rate
0.6
0.5
0.4
0.3
0.2
0.1
1980
1990
2000
2010
2020
Year

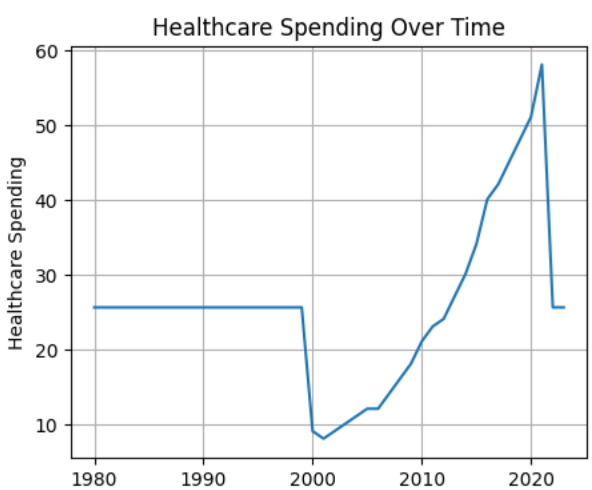

Urban Poverty Rate Over Time
Urban Poverty Rate
0.5
0.4
0.3
0.2
0.1
0.0
1980
1990
2000
2010
2020
Year


Literacy Rate Over Time
70
60

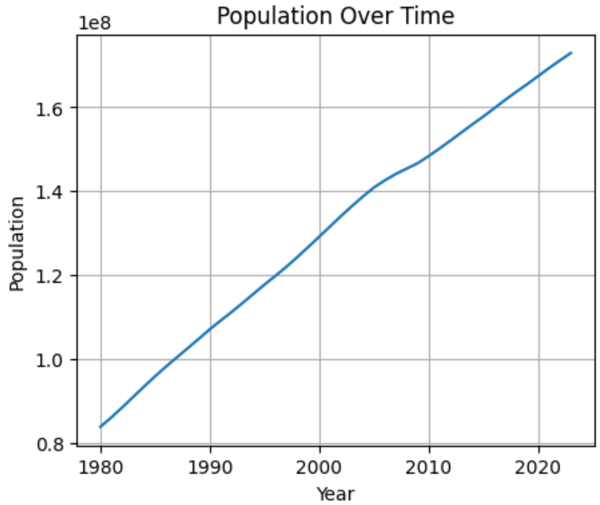

Literacy
50
40
30
1980 1990 2000 2010 2020
Year

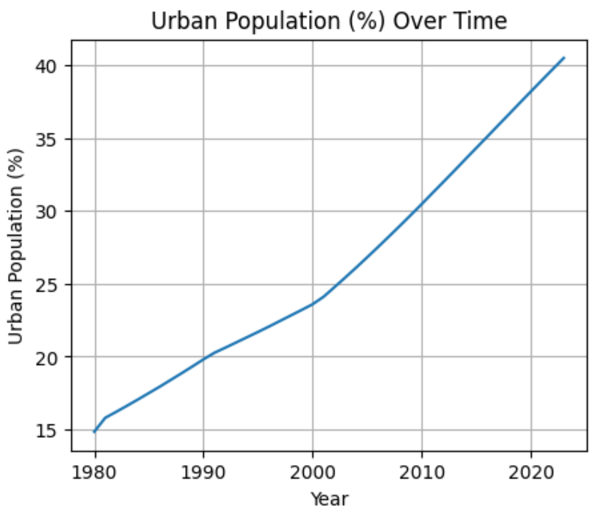

GDP Over Time
400
300
200
100
0
GDP
1980 1990 2000 2010 2020
Year

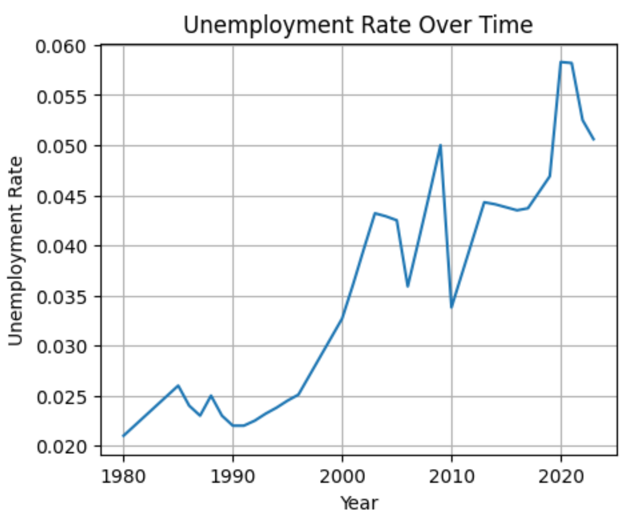

Education Spending Over Time
0.20
0.18
0.16
0.14
0.12
0.10
0.08
0.06
Education Spending
1980 1990 2000 2010 2020
Year

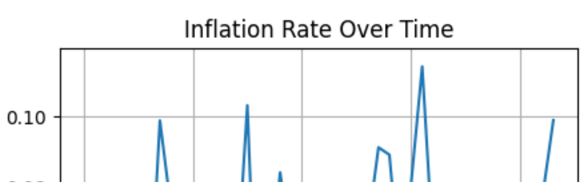

Healthcare Spending Over Time
60
50
40
30
20
10
Healthcare Spending
1980 1990 2000 2010 2020

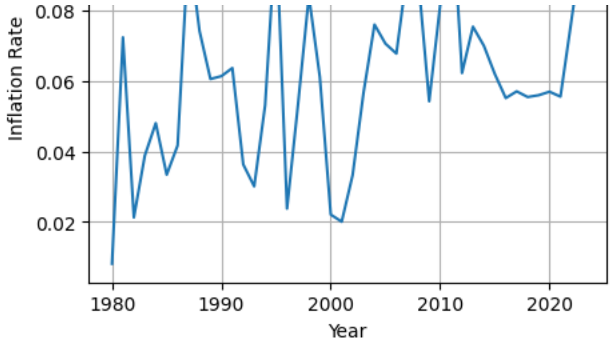
Year

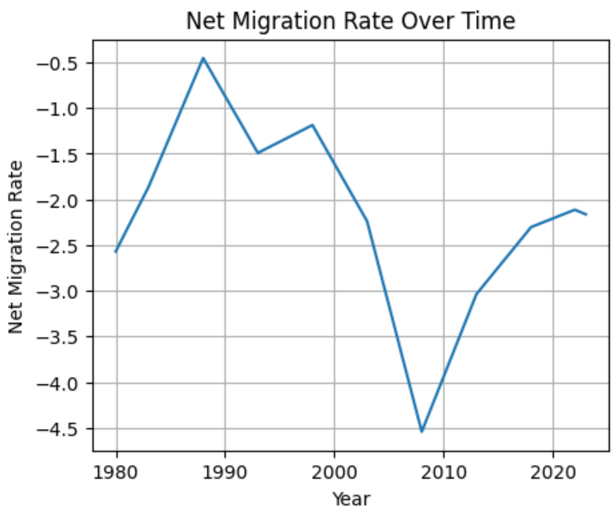

Population Over Time
1e8
Population
1.6
1.4
1.2
1.0
0.8
1980
1990
2000
2010
2020
Year

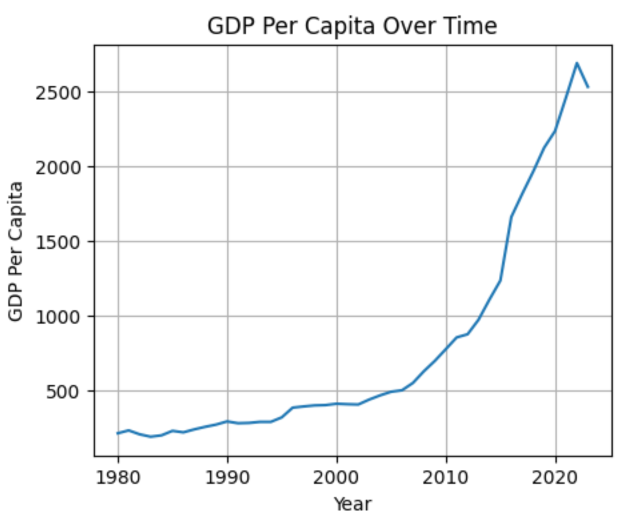

Urban Population (%) Over Time
Urban Population (%)
40
35
30
25
20
15
1980
1990
2000
2010
2020
Year

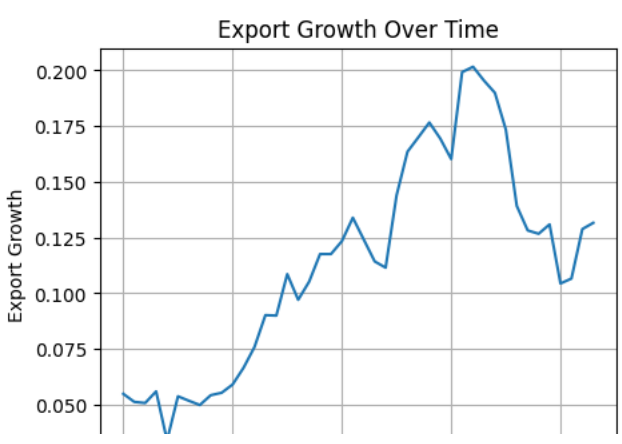

Unemployment Rate Over Time
Unemployment Rate
0.060
0.055
0.050
0.045
0.040
0.035
0.030
0.025
0.020
1980
1990
2000
2010
2020
Year

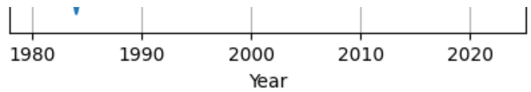

Inflation Rate Over Time
0.10

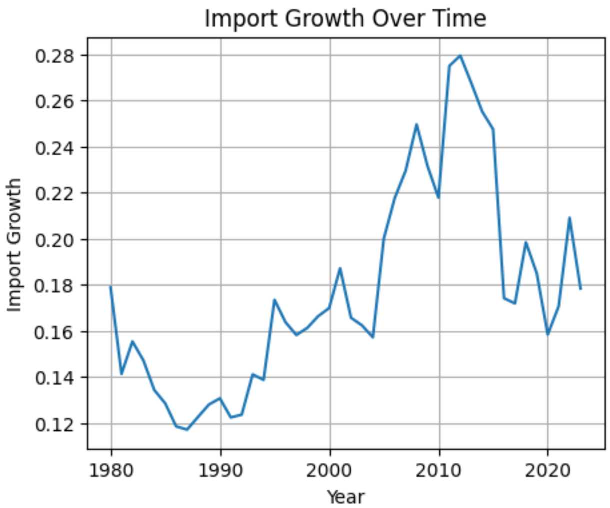

Inflation Rate
0.08
0.06
0.04
0.02
1980
1990
2000
2010
2020
Year

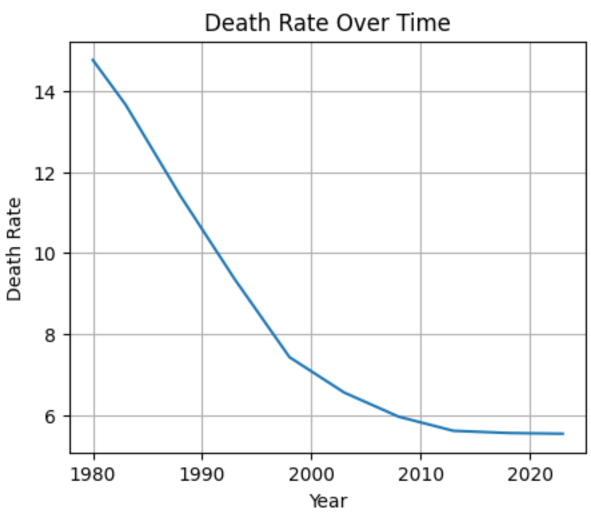

Net Migration Rate Over Time
Net Migration Rate
−0.5
−1.0
−1.5
−2.0
−2.5
−3.0
−3.5
−4.0
−4.5
1980
1990
2000
2010
2020
Year

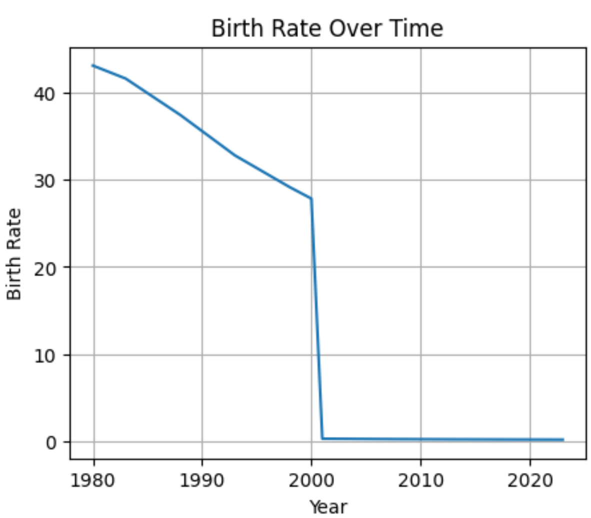

GDP Per Capita Over Time
GDP Per Capita
2500
2000
1500
1000
500
1980
1990
2000
2010
2020
Year

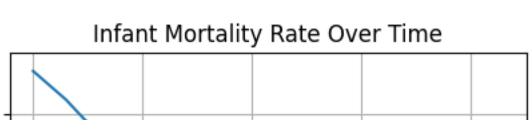

Export Growth Over Time
Export Growth
0.200
0.175
0.150
0.125
0.100
0.075
0.050

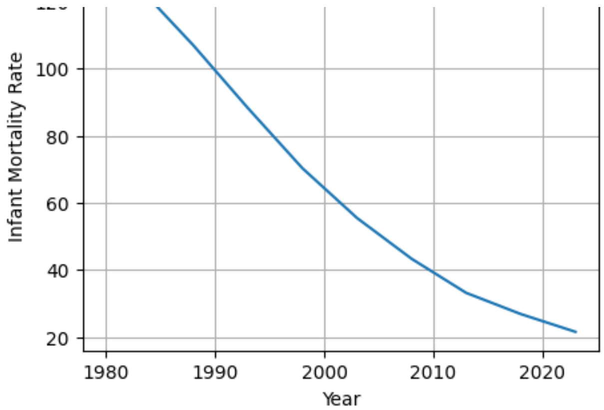

1980
1990
2000
2010
2020
Year

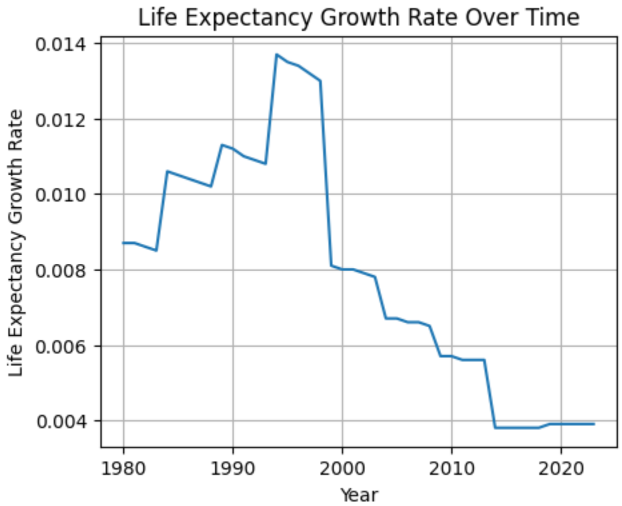

Import Growth Over Time
Import Growth
0.28
0.26
0.24
0.22
0.20
0.18
0.16
0.14
0.12
1980
1990
2000
2010
2020
Year

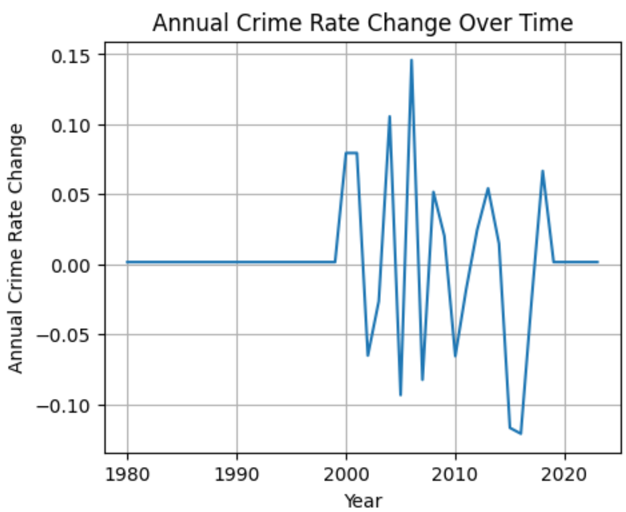

Death Rate Over Time
Death Rate
14
12
10
8
6
1980
1990
2000
2010
2020
Year

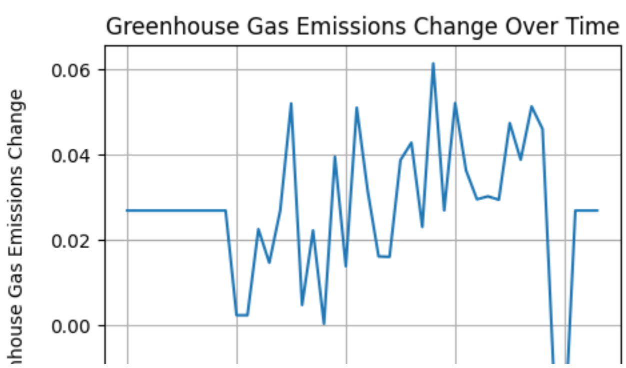

Birth Rate Over Time
Birth Rate
40
30
20
10
0
1980
1990
2000
2010
2020
Year

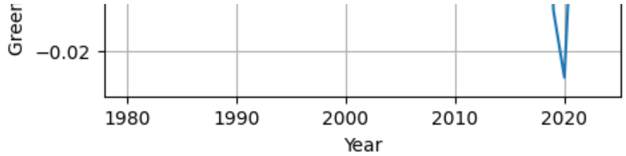

Infant Mortality Rate Over Time
120

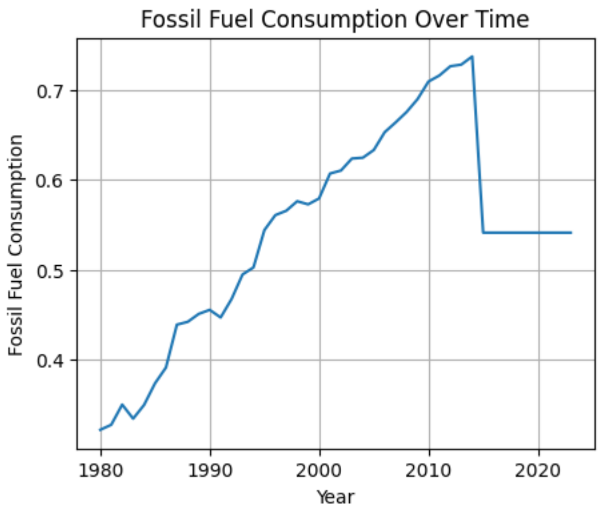

Infant Mortality Rate
Year

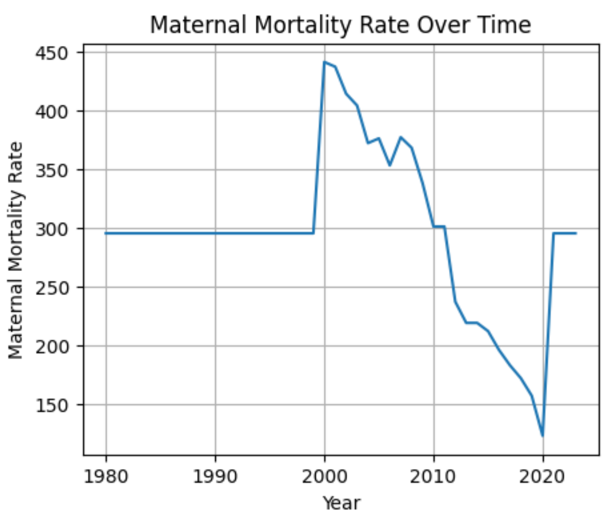

Life Expectancy Growth Rate Over Time
Life Expectancy Growth Rate
Year

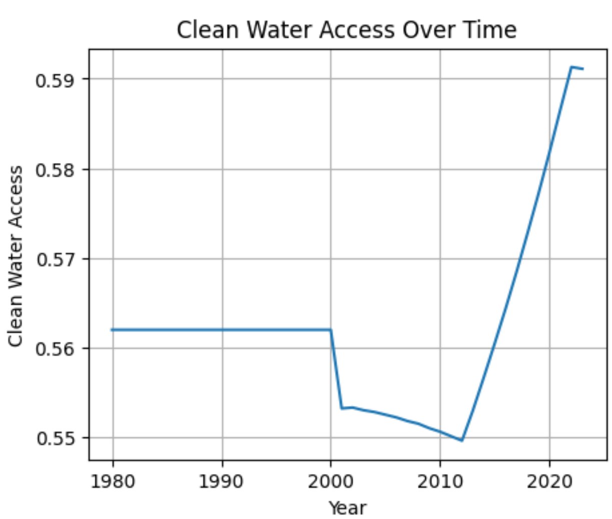

Annual Crime Rate Change Over Time
Annual Crime Rate Change
Year


Greenhouse Gas Emissions Change Over Time

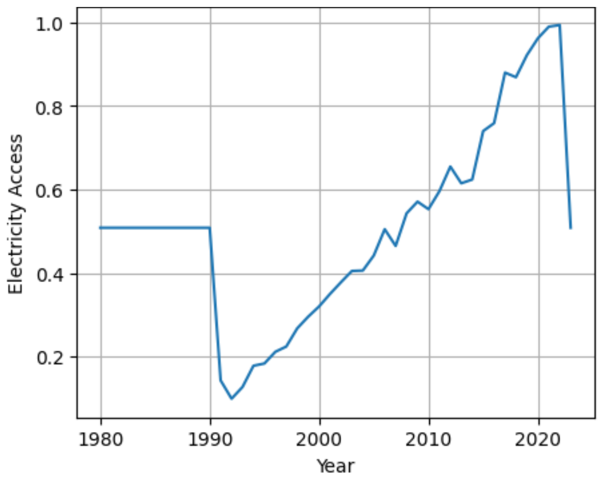

Greer
−0.02
1980
1990
2000
2010
2020
Year

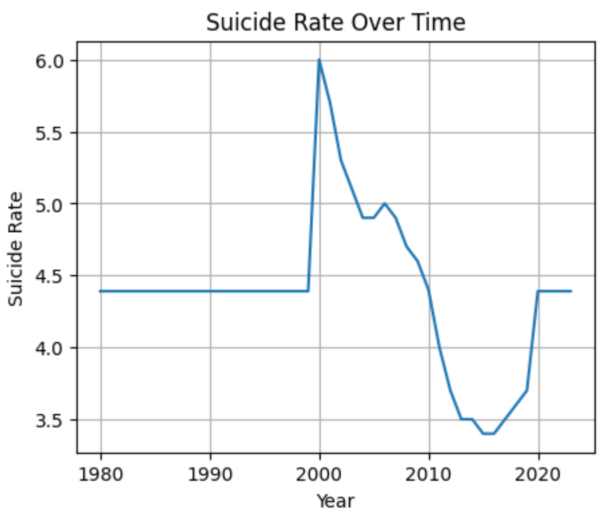

Fossil Fuel Consumption Over Time
Fossil Fuel Consumption
0.7
0.6
0.5
0.4
1980
1990
2000
2010
2020
Year

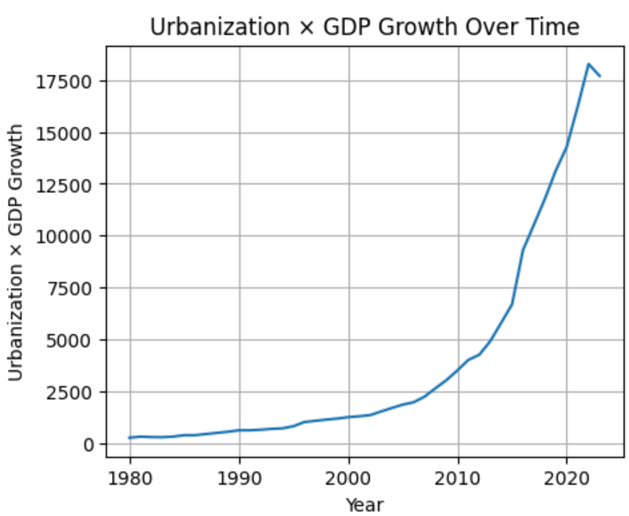

Maternal Mortality Rate Over Time
Maternal Mortality Rate
450
400
350
300
250
200
150
1980
1990
2000
2010
2020
Year

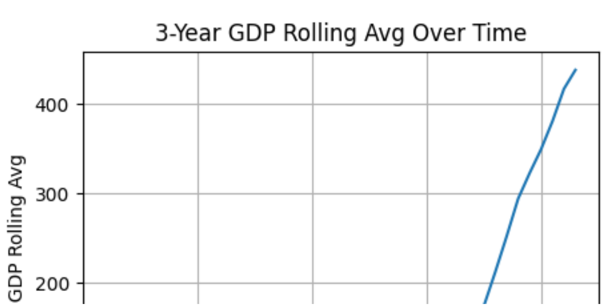

Clean Water Access Over Time
Clean Water Access
0.59
0.58
0.57
0.56
0.55
1980
1990
2000
2010
2020
Year

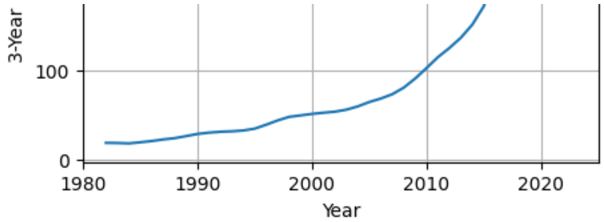

Electricity Access Over Time

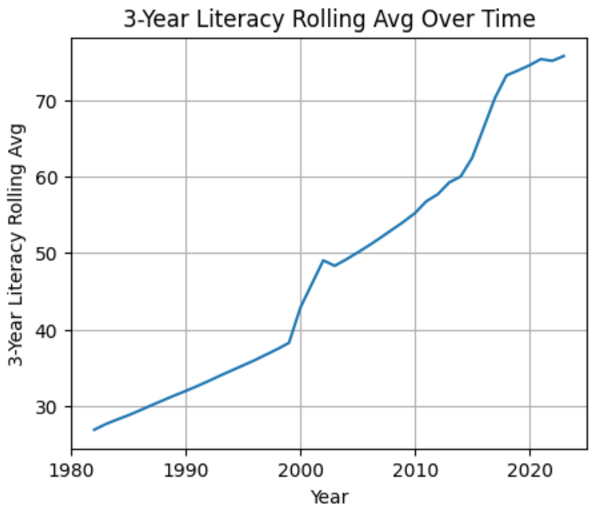

1.0
0.8
0.6
0.4
0.2
Electricity Access
1980
1990
2000
2010
2020
Year


Suicide Rate Over Time
6.0
5.5
5.0
4.5
4.0
3.5
Suicide Rate
1980
1990
2000
2010
2020
Year

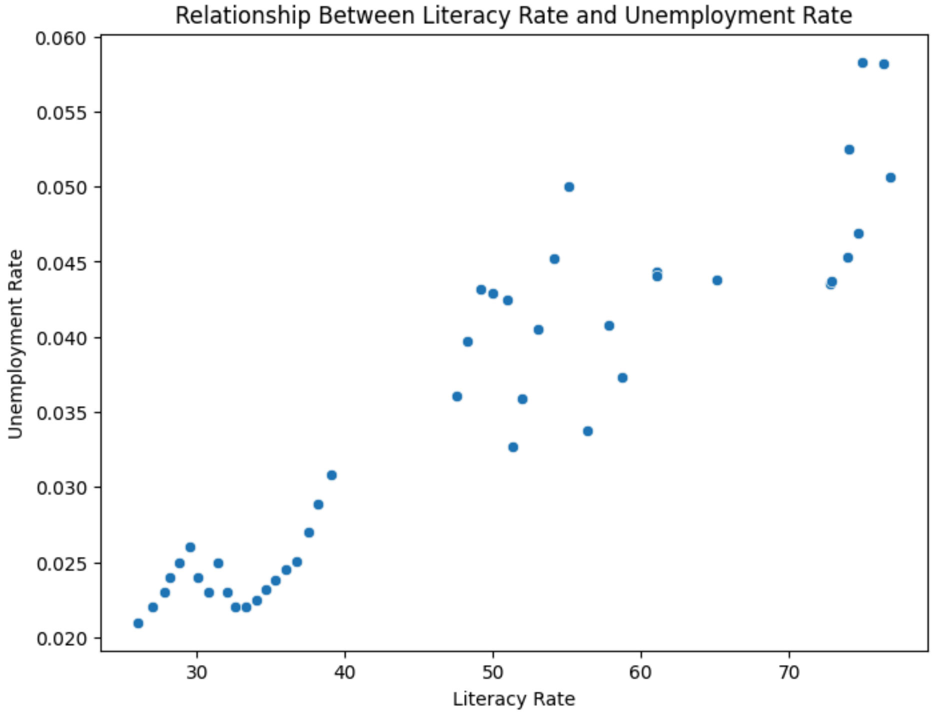

Urbanization × GDP Growth Over Time
17500
15000
12500
10000
7500
5000
2500
0
Urbanization × GDP Growth
1980
1990
2000
2010
2020
Year


3-Year GDP Rolling Avg Over Time
400
300
200
GDP Rolling Avg


3-Year
100
0
1980
1990
2000
2010
2020
Year


3-Year Literacy Rolling Avg Over Time
3-Year Literacy Rolling Avg
70
60
50
40
30
1980
1990
2000
2010
2020
Year

```
#scatter plot of literacy rate and unemployment rate
plt.figure(figsize=(8, 6))
sns.scatterplot(data=data_original, x="Literacy Rate", y="Unemployment Rate")
plt.title("Relationship Between Literacy Rate and Unemployment Rate")
plt.xlabel("Literacy Rate")
plt.ylabel
plt.show()
```

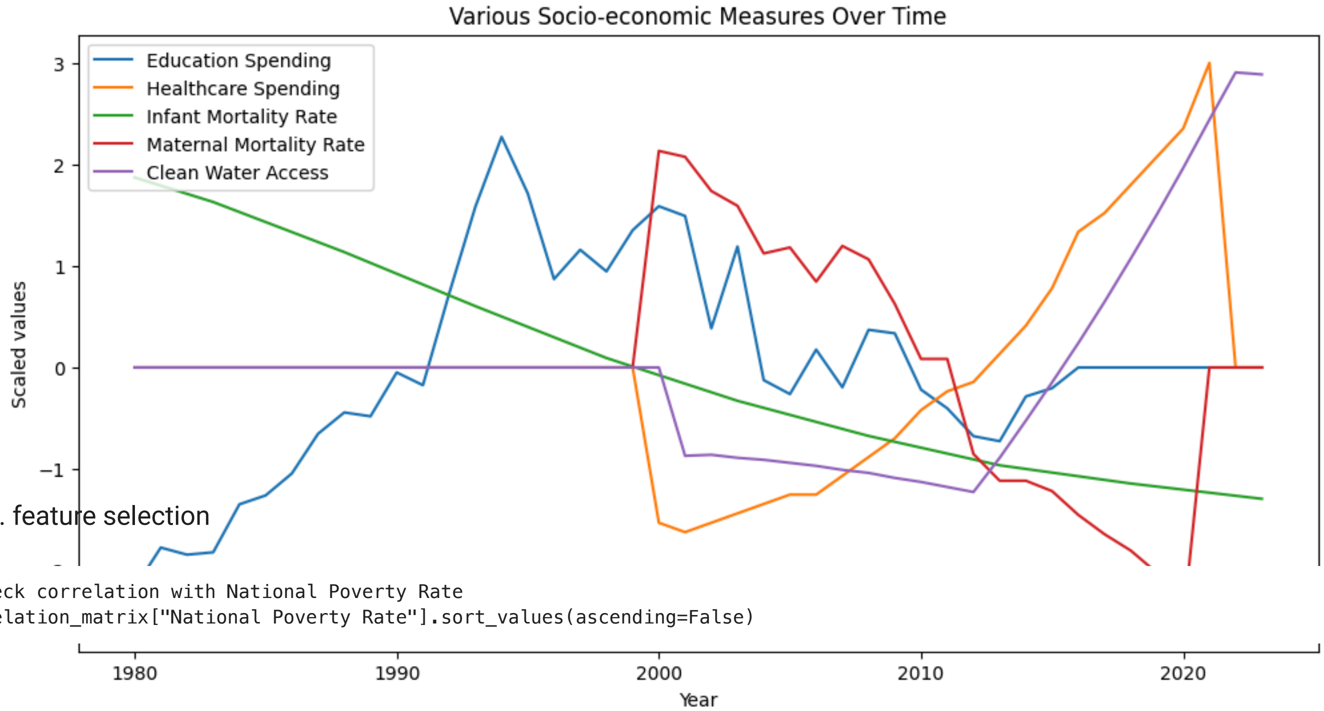



```
#scatter plot of literacy rate and Life expectancy growth rate
plt.figure(figsize=(8, 6))
sns.scatterplot(data=data_original, x="Literacy Rate", y="Life Expectancy Growth Rate")
plt.title("Relationship Between Literacy Rate and Life Expectancy Growth Rate")
plt.xlabel("Literacy Rate")
```

```
Text(0.5, 0, 'Literacy Rate')
```


```
plt.figure(figsize=(12, 6))
sns.lineplot(data=data_subset, x="Year", y="Education Spending", label="Education Spending")
sns.lineplot(data=data_subset, x="Year", y="Healthcare Spending", label="Healthcare Spending")
sns.lineplot(data=data_subset, x="Year", y="Infant Mortality Rate", label="Infant Mortality Rate")
```

```
sns.lineplot(data=data_subset, x="Year", y="Maternal Mortality Rate", label="Maternal Mortality Rate")
sns.lineplot(data=data_subset, x="Year", y="Clean Water Access", label="Clean Water Access")
plt.title("Various Socio-economic Measures Over Time")
plt.xlabel("Year")
plt.ylabel("Scaled values")
plt.legend()
plt.show()
```



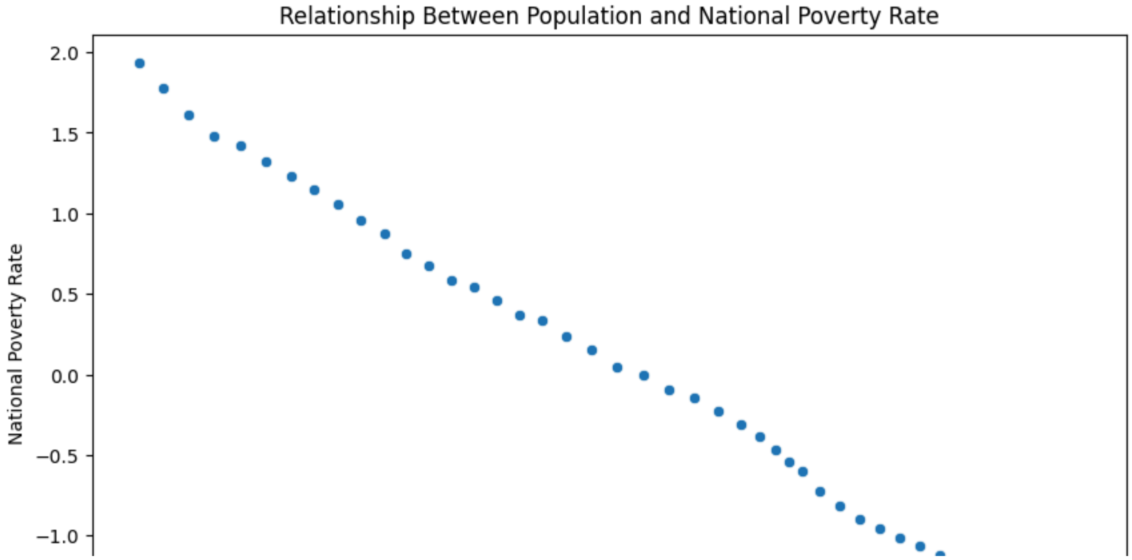
2. feature selection

```
# check correlation with National Poverty Rate
correlation_matrix["National Poverty Rate"].sort_values(ascending=False)
```

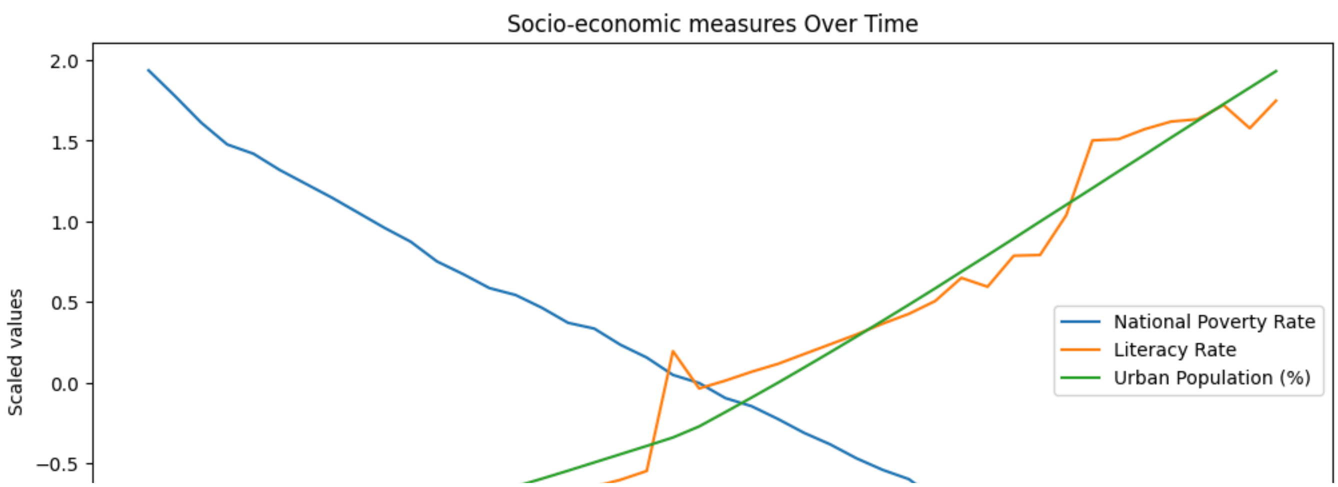
	National Poverty Rate
National Poverty Rate	1.000000
Rural Poverty Rate	0.999709
Urban Poverty Rate	0.998054
Infant Mortality Rate	0.992928
Death Rate	0.947170
Birth Rate	0.917431
Life Expectancy Growth Rate	0.775513
Net Migration Rate	0.535455
Maternal Mortality Rate	0.302007
Suicide Rate	0.276712
Annual Crime Rate Change	0.071631
Greenhouse Gas Emissions Change	-0.102458
Clean Water Access	-0.275660
Education Spending	-0.301818
Healthcare Spending	-0.311259
Inflation Rate	-0.373114
Urbanization × GDP Growth	-0.396160
Electricity Access	-0.574456
Import Growth	-0.652610
Fossil Fuel Consumption	-0.740503
Export Growth	-0.799517
GDP	-0.814500
GDP Per Capita	-0.818170
3-Year GDP Rolling Avg	-0.819651
Unemployment Rate	-0.907094
Literacy Rate	-0.968302
3-Year Literacy Rolling Avg	-0.973785
Urban Population (%)	-0.979221
Year	-0.997426
Population	-0.998727

dtype: float64

```
#scatter plot for National Poverty Rate vs Population
plt.figure(figsize=(10, 6))
sns.scatterplot(data=data_subset, x="Population", y="National Poverty Rate")
plt.title("Relationship Between Population and National Poverty Rate")
plt.xlabel("Population")
plt.ylabel
plt.show()
```




```
plt.figure(figsize=(12, 6))
sns.lineplot(data=data_subset, x="Year", y="National Poverty Rate", label="National Poverty Rate")
sns.lineplot(data=data_subset, x="Year", y="Literacy Rate", label="Literacy Rate")
sns.lineplot(data=data_subset, x="Year", y="Urban Population (%)", label="Urban Population (%)")
#sns.lineplot(data=data_subset, x="Year", y="Maternal Mortality Rate", label="Maternal Mortality Rate")
#sns.lineplot(data=data_subset, x="Year", y="Clean Water Access", label="Clean Water Access")
plt.title("Socio-economic measures Over Time")
plt.xlabel("Year")
plt.ylabel("Scaled values")
plt.legend()
plt.show()
```




```
# correlation with Literacy Rate
correlation_matrix = data_subset.corr()
correlation_with_literacy = correlation_matrix["Literacy Rate"].sort_values(ascending=False)
print("\nCorrelation with Literacy Rate:")
print(correlation_with_literacy)
```

```
Correlation with Literacy Rate:
Literacy Rate                  1.000000
3-Year Literacy Rolling Avg    0.995405
Urban Population (%)           0.988159
Year                           0.980226
Population                     0.972824
Unemployment Rate              0.928044
GDP Per Capita                 0.901279
GDP                            0.897481
3-Year GDP Rolling Avg         0.888713
Electricity Access             0.711008
Export Growth                  0.695975
Fossil Fuel Consumption        0.589086
Import Growth                  0.585368
Urbanization × GDP Growth      0.535440
Healthcare Spending            0.416864
Clean Water Access             0.397836
```

```
Inflation Rate                    0.294528
Education Spending                0.154902
Greenhouse Gas Emissions Change   0.086434
Annual Crime Rate Change         -0.083553
Suicide Rate                     -0.305661
Maternal Mortality Rate          -0.370832
Net Migration Rate               -0.517146
Death Rate                       -0.855011
Life Expectancy Growth Rate      -0.861326
Birth Rate                       -0.883872
Infant Mortality Rate            -0.944369
Urban Poverty Rate               -0.955698
National Poverty Rate            -0.968302
Rural Poverty Rate               -0.970977
Name: Literacy Rate, dtype: float64
```

```
# correlation with National Poverty Rate
correlation_matrix = data_subset.corr()
correlation_with_literacy = correlation_matrix["National Poverty Rate"].sort_values(ascending=False)
print("\nCorrelation with National Poverty Rate:")
print(correlation_with_literacy)
```

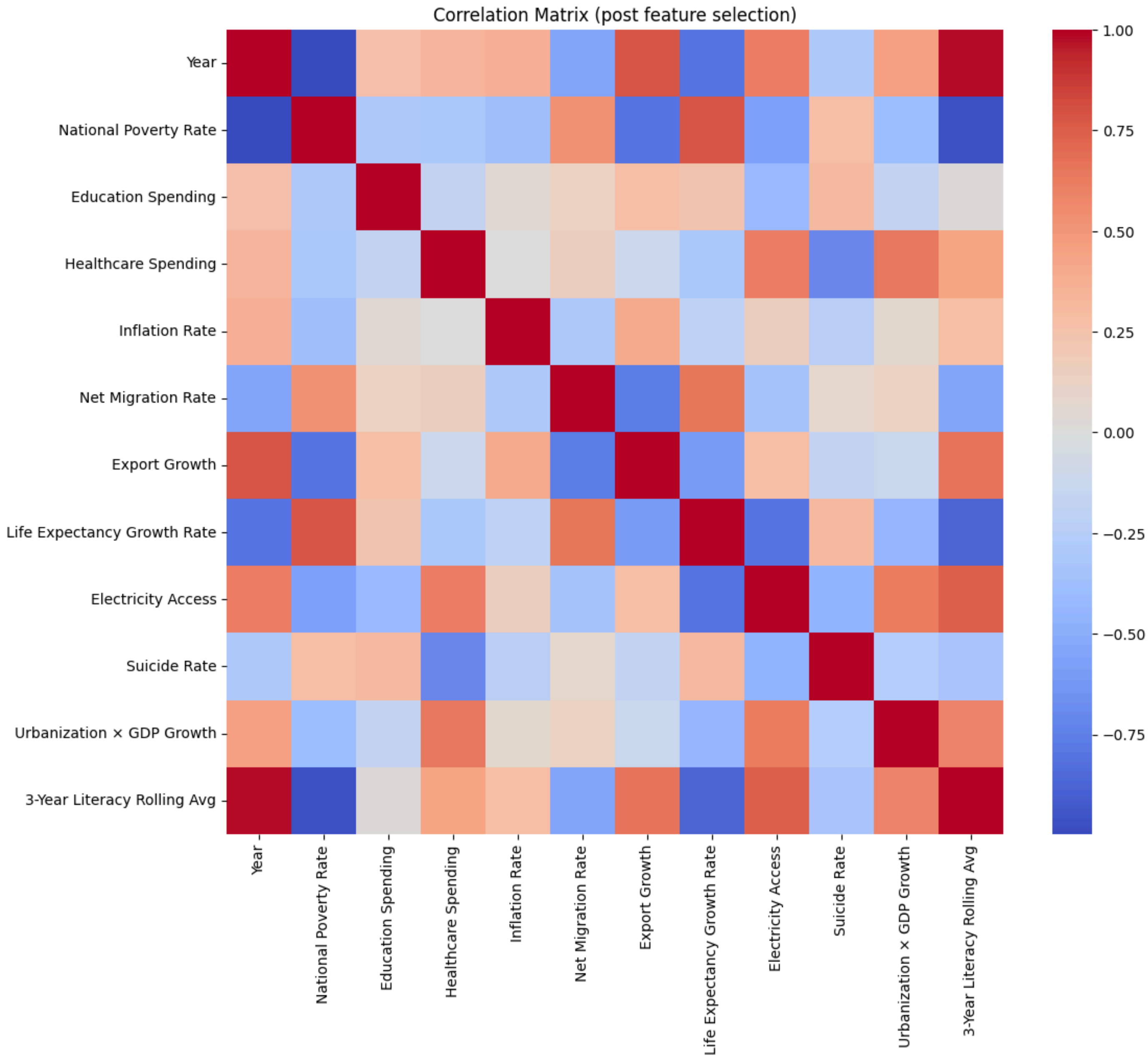
```
Correlation with National Poverty Rate:
National Poverty Rate             1.000000
Rural Poverty Rate                0.999709
Urban Poverty Rate                0.998054
Infant Mortality Rate             0.992928
Death Rate                        0.947170
Birth Rate                        0.917431
Life Expectancy Growth Rate       0.775513
Net Migration Rate                0.535455
Maternal Mortality Rate           0.302007
Suicide Rate                      0.276712
Annual Crime Rate Change          0.071631
Greenhouse Gas Emissions Change  -0.102458
Clean Water Access               -0.275660
Education Spending               -0.301818
Healthcare Spending              -0.311259
Inflation Rate                   -0.373114
Urbanization × GDP Growth        -0.396160
Electricity Access               -0.574456
Import Growth                    -0.652610
Fossil Fuel Consumption          -0.740503
Export Growth                    -0.799517
GDP                              -0.814500
GDP Per Capita                   -0.818170
3-Year GDP Rolling Avg           -0.819651
Unemployment Rate                -0.907094
Literacy Rate                    -0.968302
3-Year Literacy Rolling Avg      -0.973785
Urban Population (%)             -0.979221
Year                             -0.997426
Population                       -0.998727
Name: National Poverty Rate, dtype: float64
```

```
# drop features with multicollinearity
drop_multicollinearity = ["GDP", "GDP Per Capita", "Urban Population (%)", "Urban Poverty Rate", "Rural Poverty Rate", "Populati
                          "Fossil Fuel Consumption", "Import Growth", "Death Rate", "Infant Mortality Rate", "Maternal Mortality
                          "Clean Water Access", "Unemployment Rate", "Birth Rate", "3-Year GDP Rolling Avg", "Literacy Rate"]
data_subset2 = data_subset.drop(columns=drop_multicollinearity)

# drop features with weak correlation
drop_weak = ["Greenhouse Gas Emissions Change", "Annual Crime Rate Change"]
data_subset3 = data_subset2.drop(columns=drop_weak)

data_subset3.columns
```

```
Index(['Year', 'National Poverty Rate', 'Education Spending',
       'Healthcare Spending', 'Inflation Rate', 'Net Migration Rate',
       'Export Growth', 'Life Expectancy Growth Rate', 'Electricity Access',
       'Suicide Rate', 'Urbanization × GDP Growth',
       '3-Year Literacy Rolling Avg'],
      dtype='object')
```

```
  # Correlation Matrix
correlation_matrix = data_subset3.corr()
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=False, cmap="coolwarm", fmt=".2f")
```

```
plt.title("Correlation Matrix (post feature selection)")
plt.show()
```



3. Assumption test

```
# missing variable, impute with mean
from sklearn.impute import SimpleImputer

data_subset4 = data_subset3.copy()
imputer = SimpleImputer(strategy="mean")
data_subset4.iloc[:, 1:] = imputer.fit_transform(data_subset3.iloc[:, 1:])

# assumption test - stationarity for each variable
from statsmodels.tsa.stattools import adfuller

for col in data_subset4.columns[1:]:
    print(f"Testing for Stationarity of {col}:")
    result = adfuller(data_subset4[col])
    print(f'ADF Statistic: {result[0]}')
    print(f'p-value: {result[1]}')  # p < 0.05 → Stationary, p > 0.05 → Not Stationary
```

```
    print("\n")

# create a list of features that are not stationary
non_stationary_features = [col for col in data_subset4.columns[1:] if adfuller(data_subset4[col])[1] > 0.05]
print("Non-Stationary Features:", non_stationary_features)
```

```
Testing for Stationarity of National Poverty Rate:
ADF Statistic: -4.763823203330587
p-value: 6.374826725818937e-05


Testing for Stationarity of Education Spending:
ADF Statistic: -3.544900041245629
p-value: 0.006906122959191104


Testing for Stationarity of Healthcare Spending:
ADF Statistic: -1.8008294343679248
p-value: 0.38004998028956016


Testing for Stationarity of Inflation Rate:
ADF Statistic: -5.145502940672432
p-value: 1.1347447558833673e-05


Testing for Stationarity of Net Migration Rate:
ADF Statistic: -2.371684077838328
p-value: 0.14988835272038342


Testing for Stationarity of Export Growth:
ADF Statistic: -1.3197905082476749
p-value: 0.6200702019437832


Testing for Stationarity of Life Expectancy Growth Rate:
ADF Statistic: -0.6543480750939475
p-value: 0.8582228137788708


Testing for Stationarity of Electricity Access:
ADF Statistic: -1.4191514377955283
p-value: 0.5730610243262942


Testing for Stationarity of Suicide Rate:
ADF Statistic: -1.807758642953694
p-value: 0.37661660313197487


Testing for Stationarity of Urbanization × GDP Growth:
ADF Statistic: -2.276533624733398
p-value: 0.17964806285176094


Testing for Stationarity of 3-Year Literacy Rolling Avg:
ADF Statistic: 0.28434878043769235
p-value: 0.9766149123955452


Non-Stationary Features: ['Healthcare Spending', 'Net Migration Rate', 'Export Growth', 'Life Expectancy Growth Rate', 'Elec
```

4. predictive model

```
print(data_subset4.shape)
print(data_subset4.head())
print(data_subset4.info())
```

```
(44, 12)
   Year  National Poverty Rate  Education Spending  Healthcare Spending  \
0  1980               1.936390           -2.174833                  0.0
1  1981               1.778304           -1.774808                  0.0
2  1982               1.613032           -1.844885                  0.0
3  1983               1.476503           -1.821526                  0.0
4  1984               1.419017           -1.348503                  0.0

   Inflation Rate  Net Migration Rate  Export Growth  \
0       -2.175351           -0.404870      -1.244337
1        0.536530           -0.170065      -1.320995
```

```
2      -1.621129           0.063745      -1.331642
3      -0.880755           0.297555      -1.220914
4      -0.491531           0.578128      -1.689379

   Life Expectancy Growth Rate  Electricity Access  Suicide Rate  \
0                     0.252218                 0.0  1.634136e-15
1                     0.252218                 0.0  1.634136e-15
2                     0.220236                 0.0  1.634136e-15
3                     0.188255                 0.0  1.634136e-15
4                     0.859867                 0.0  1.634136e-15

   Urbanization × GDP Growth  3-Year Literacy Rolling Avg
0                   1.174562                    -0.008660
1                   1.052558                    -0.008660
2                   1.028819                    -1.290739
3                   0.994492                    -1.246053
4                   0.936896                    -1.209492
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 44 entries, 0 to 43
Data columns (total 12 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   Year                         44 non-null     int64
 1   National Poverty Rate        44 non-null     float64
 2   Education Spending           44 non-null     float64
 3   Healthcare Spending          44 non-null     float64
 4   Inflation Rate               44 non-null     float64
 5   Net Migration Rate           44 non-null     float64
 6   Export Growth                44 non-null     float64
 7   Life Expectancy Growth Rate  44 non-null     float64
 8   Electricity Access           44 non-null     float64
 9   Suicide Rate                 44 non-null     float64
 10  Urbanization × GDP Growth    44 non-null     float64
 11  3-Year Literacy Rolling Avg  44 non-null     float64
dtypes: float64(11), int64(1)
memory usage: 4.3 KB
None
```

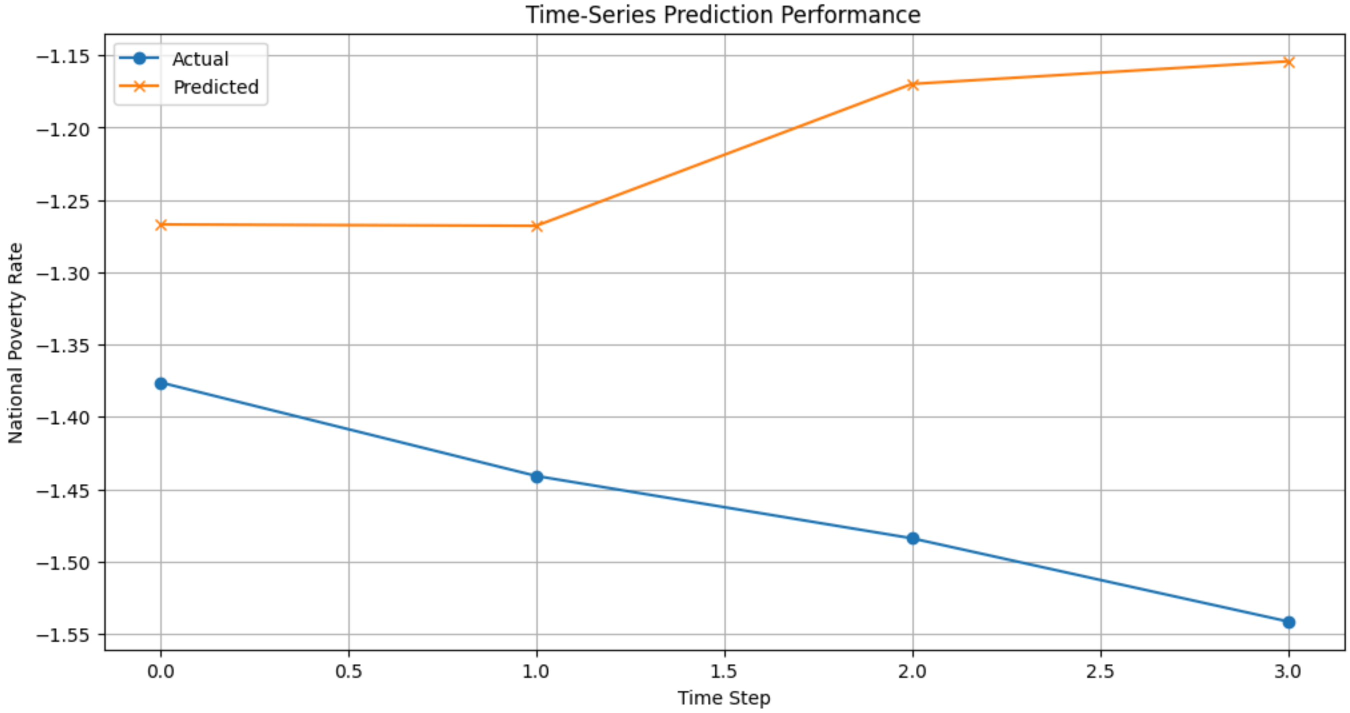
```
#!pip install --upgrade scikit-learn
```

4.1 Random Forest Model

```
import numpy as np
import pandas as pd
from sklearn.model_selection import TimeSeriesSplit, GridSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import matplotlib.pyplot as plt


def preprocess_time_series_data(df):
    """Time-series specific preprocessing"""
    # Clean column names and sort by year
    df = df.rename(columns={'Urbanization × GDP Growth': 'Urbanization_GDP_Growth'})
    df = df.sort_values('Year').reset_index(drop=True)

    # Handle zero values in Healthcare Spending
    df['Healthcare Spending'] = np.where(
        (df['Healthcare Spending'] == 0) & (df['Year'] < 2000),
        np.nan,
        df['Healthcare Spending']
    )
    df['Healthcare Spending'] = df['Healthcare Spending'].ffill()

    # Remove non-informative columns
    df = df.drop(columns=['Year', 'Electricity Access', 'Suicide Rate'])

    return df.dropna()

processed_data = preprocess_time_series_data(data_subset4)

# Split features and target
X = processed_data.drop(columns=['National Poverty Rate'])
y = processed_data['National Poverty Rate']

# Time-series aware train-test split (last 20% as test)
test_size = int(len(X) * 0.2)
X_train, X_test = X[:-test_size], X[-test_size:]
```

```
y_train, y_test = y[:-test_size], y[-test_size:]

# Time-series cross-validation configuration
tscv = TimeSeriesSplit(n_splits=5)

# Hyperparameter grid with regularization parameters
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 5],
    'max_features': ['sqrt', 0.8],
    'min_impurity_decrease': [0.0, 0.1]
}

# Initialize and tune Random Forest
rf = RandomForestRegressor(random_state=42, n_jobs=-1)
grid_search = GridSearchCV(
    estimator=rf,
    param_grid=param_grid,
    cv=tscv,
    scoring='neg_mean_squared_error',
    verbose=1,
    n_jobs=-1
)

print("Starting grid search with time-series validation...")
grid_search.fit(X_train, y_train)

# Best model evaluation
best_rf = grid_search.best_estimator_

def time_series_evaluate(model, X_train, X_test, y_train, y_test):
    """Time-series specific evaluation"""
    # Generate predictions
    y_pred_train = model.predict(X_train)
    y_pred_test = model.predict(X_test)

    # Calculate metrics
    metrics = {
        'Train RMSE': np.sqrt(mean_squared_error(y_train, y_pred_train)),
        'Test RMSE': np.sqrt(mean_squared_error(y_test, y_pred_test)),
        'Train MAE': mean_absolute_error(y_train, y_pred_train),
        'Test MAE': mean_absolute_error(y_test, y_pred_test),
        'Train R²': r2_score(y_train, y_pred_train),
        'Test R²': r2_score(y_test, y_pred_test)
    }

    # Plot actual vs predicted
    plt.figure(figsize=(12, 6))
    plt.plot(y_test.values, label='Actual', marker='o')
    plt.plot(y_pred_test, label='Predicted', marker='x')
    plt.title('Time-Series Prediction Performance')
    plt.xlabel('Time Step')
    plt.ylabel('National Poverty Rate')
    plt.legend()
    plt.grid(True)
    plt.show()

    return pd.Series(metrics)

# Evaluate and show results
metrics = time_series_evaluate(best_rf, X_train, X_test, y_train, y_test)
print("\nBest Parameters:", grid_search.best_params_)
print("\nPerformance Metrics:\n", metrics.to_string())

# Feature importance analysis
feature_importance = pd.DataFrame({
    'Feature': X.columns,
    'Importance': best_rf.feature_importances_
}).sort_values('Importance', ascending=False)

print("\nFeature Importance:\n", feature_importance.to_string(index=False))
```

```
Starting grid search with time-series validation...
Fitting 5 folds for each of 48 candidates, totalling 240 fits
```




```
Best Parameters: {'max_depth': None, 'max_features': 0.8, 'min_impurity_decrease': 0.0, 'min_samples_split': 2, 'n_estimator

Performance Metrics:
 Train RMSE     0.021702
Test RMSE      0.269561
Train MAE      0.017697
Test MAE       0.245967
Train R²       0.997422
Test R²      -18.908087

Feature Importance:
                     Feature  Importance
3-Year Literacy Rolling Avg    0.346535
        Healthcare Spending    0.320461
Life Expectancy Growth Rate    0.182795
    Urbanization_GDP_Growth    0.054611
          Net Migration Rate    0.031819
               Export Growth    0.023393
          Education Spending    0.021718
              Inflation Rate    0.018670
```

4.2 Lasso Regression

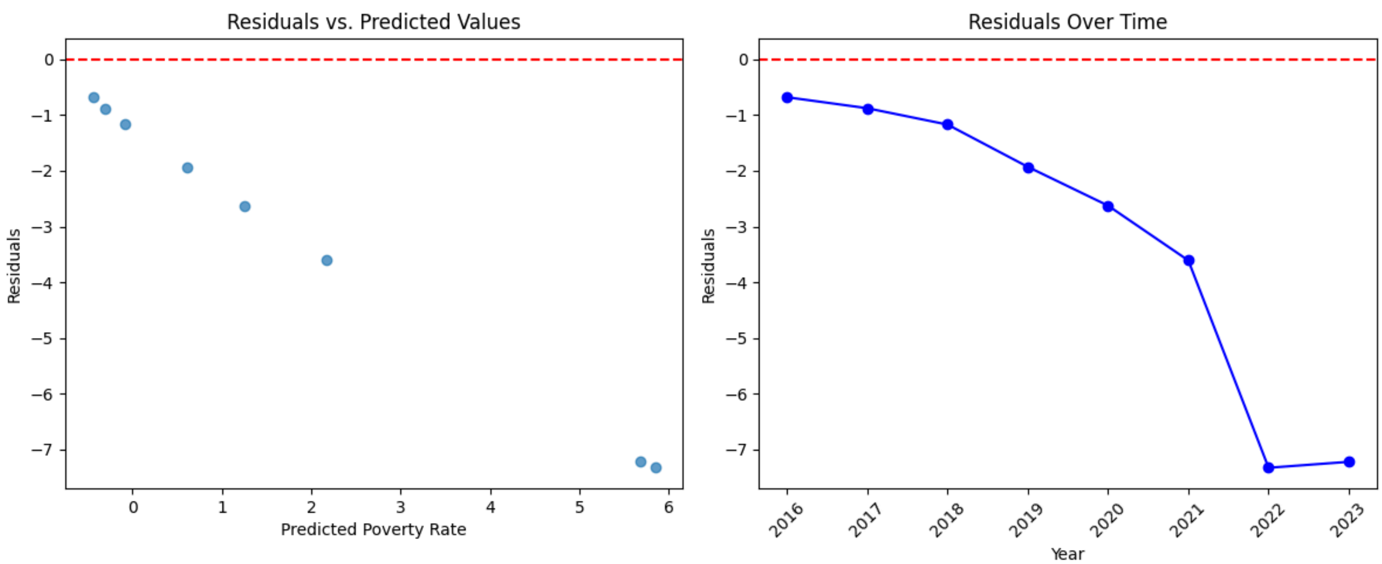
```
import numpy as np
import pandas as pd
import warnings
from sklearn.model_selection import TimeSeriesSplit, train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.exceptions import ConvergenceWarning
from sklearn.model_selection import GridSearchCV

# Load data and sort by year
data_subset = data_subset4.sort_values('Year')

# Feature Engineering
data_subset['Poverty_Rate_Lag1'] = data_subset['National Poverty Rate'].shift(1)
data_subset['Poverty_Rate_Lag2'] = data_subset['National Poverty Rate'].shift(2)
data_subset['Poverty_Rate_Rolling3'] = (
    data_subset['National Poverty Rate']
    .rolling(window=3, min_periods=1, closed='left')  # Ensure no future leakage
    .mean()
)
```

```
# Impute missing values (instead of dropping)
data_subset.fillna(0, inplace=True)  # Or use forward-fill

# Define target and predictors
X = data_subset.drop(columns=['National Poverty Rate', 'Year'])
y = data_subset['National Poverty Rate']

# Train-test split (last 20% as test)
test_size = int(0.2 * len(X))
X_train, X_test = X[:-test_size], X[-test_size:]
y_train, y_test = y[:-test_size], y[-test_size:]

# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Time-series cross-validation
tscv = TimeSeriesSplit(n_splits=5)

# Hyperparameter tuning with TimeSeriesSplit
param_grid = {'alpha': np.logspace(-5, 3, 50)}  # Wider alpha range

with warnings.catch_warnings():
    warnings.filterwarnings("ignore", category=ConvergenceWarning)
    lasso_cv = GridSearchCV(
        Lasso(max_iter=10000),
        param_grid,
        cv=tscv,  # Use TimeSeriesSplit
        scoring='neg_mean_squared_error'
    )
    lasso_cv.fit(X_train_scaled, y_train)

# Best model
best_lasso = lasso_cv.best_estimator_

# Predictions and evaluation
y_pred_train = best_lasso.predict(X_train_scaled)
y_pred_test = best_lasso.predict(X_test_scaled)

print("Train Metrics:")
print(f"MAE: {mean_absolute_error(y_train, y_pred_train):.4f}")
print(f"MSE: {mean_squared_error(y_train, y_pred_train):.4f}")
print(f"R²: {r2_score(y_train, y_pred_train):.4f}\n")

print("Test Metrics:")
print(f"MAE: {mean_absolute_error(y_test, y_pred_test):.4f}")
print(f"MSE: {mean_squared_error(y_test, y_pred_test):.4f}")
print(f"R²: {r2_score(y_test, y_pred_test):.4f}")

# Feature importance
feature_importance = pd.DataFrame({
    'Feature': X.columns,
    'Coefficient': best_lasso.coef_
}).sort_values('Coefficient', key=abs, ascending=False)
print("\nFeature Importance:\n", feature_importance)
```

```
Train Metrics:
MAE: 0.0736
MSE: 0.0085
R²: 0.9884

Test Metrics:
MAE: 3.1801
MSE: 16.5092
R²: -830.0116

Feature Importance:
                        Feature  Coefficient
9   3-Year Literacy Rolling Avg    -1.559530
11            Poverty_Rate_Lag2    -1.258370
10            Poverty_Rate_Lag1     0.962675
12        Poverty_Rate_Rolling3    -0.841784
8     Urbanization × GDP Growth     0.660744
1           Healthcare Spending    -0.426611
3             Net Migration Rate    -0.217992
4                 Export Growth    -0.154719
0            Education Spending    -0.129683
```

```
     7                 Suicide Rate    -0.090553
     5   Life Expectancy Growth Rate    0.082975
     6            Electricity Access    0.019748
     2                Inflation Rate   -0.004280
```

```
# Residual Plot
import matplotlib.pyplot as plt

# Predictions and residuals
y_pred_test = best_lasso.predict(X_test_scaled)
residuals = y_test - y_pred_test

# Plot residuals vs. predicted values
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.scatter(y_pred_test, residuals, alpha=0.7)
plt.axhline(0, color='r', linestyle='--')
plt.title('Residuals vs. Predicted Values')
plt.xlabel('Predicted Poverty Rate')
plt.ylabel('Residuals')

# Plot residuals over time (test set years)
test_years = data_subset['Year'].iloc[-len(y_test):]
plt.subplot(1, 2, 2)
plt.plot(test_years, residuals, marker='o', linestyle='-', color='b')
plt.axhline(0, color='r', linestyle='--')
plt.title('Residuals Over Time')
plt.xlabel('Year')
plt.ylabel('Residuals')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



4.3 VAR Model

```
import numpy as np
import pandas as pd
from statsmodels.tsa.api import VAR
from statsmodels.tsa.stattools import adfuller, grangercausalitytests
from statsmodels.stats.stattools import durbin_watson
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import matplotlib.pyplot as plt

# ============================================
# 1. Data Preparation & Cleaning
# ============================================
data = data_subset4.sort_values('Year').set_index('Year')
data.replace(0, np.nan, inplace=True)
```

```
data.fillna(method='ffill', inplace=True)
data = data.drop(columns=['Electricity Access', 'Suicide Rate'])

# ==============================================
# 2. Stationarity Transformation (Dynamic Handling)
# ==============================================
def make_stationary(df, target_col):
    stationary_df = df.copy()
    # Check if target needs differencing
    p_value_target = adfuller(df[target_col].dropna())[1]
    if p_value_target > 0.05:
        stationary_df[f"{target_col}_diff"] = df[target_col].diff()
        stationary_df.drop(target_col, axis=1, inplace=True)
        target_var = f"{target_col}_diff"
    else:
        target_var = target_col

    # Check other features
    for col in df.columns:
        if col != target_col:
            p_value = adfuller(df[col].dropna())[1]
            if p_value > 0.05:
                stationary_df[f"{col}_diff"] = df[col].diff()
                stationary_df.drop(col, axis=1, inplace=True)
    return stationary_df.dropna(), target_var

stationary_data, target_var = make_stationary(data, 'National Poverty Rate')

# ==============================================
# 3. Feature Selection (Granger Causality) - Updated
# ==============================================
candidate_features = [col for col in stationary_data.columns if col != target_var]
significant_features = []

print("\nGranger Causality Results:")
for feature in candidate_features:
    test_result = grangercausalitytests(stationary_data[[target_var, feature]], maxlag=2, verbose=False
    p_values = [test_result[i+1][0]['ssr_chi2test'][1] for i in range(2)]
    min_p = min(p_values)
    print(f"{feature}: min p-value = {min_p:.4f}")
    if min_p < 0.1:  # Relaxed threshold
        significant_features.append(feature)

final_features = [target_var] + significant_features
filtered_data = stationary_data[final_features]

# Add variance filtering
from sklearn.feature_selection import VarianceThreshold
selector = VarianceThreshold(threshold=0.01)
filtered_data = pd.DataFrame(
    selector.fit_transform(filtered_data),
    columns=filtered_data.columns[selector.get_support()]
)

print("\nFinal Features After Filtering:", filtered_data.columns.tolist())

# ==============================================
# 4. Train-Test Split & Model Fitting
# ==============================================
test_size = 5
train = filtered_data.iloc[:-test_size]
test = filtered_data.iloc[-test_size:]

model = VAR(train)
lag_order = model.select_order(maxlags=3).selected_orders['aic']
var_model = model.fit(lag_order)

# ==============================================
# 5. Feature Importance Analysis (Updated)
# ==============================================
def calculate_var_feature_importance(model, target_index):
    """Calculate and format feature importance with full feature listing"""
    coeffs = model.coefs
    importance = {feature: 0.0 for feature in train.columns if feature != target_var}

    # Calculate raw importance scores
    for lag in range(lag_order):
```