# Data Collection and Preprocessing Phase

| | |
|---|---|
| Date | 15 March 2024 |
| Team ID | xxxxxx |
| Project Title | Forecasting Economic Prosperity: Leveraging Machine Learning For GDP Per Capita Prediction |
| Maximum Marks | 6 Marks |

## Data Exploration and Preprocessing Template

Identifies data sources, assesses quality issues like missing values and duplicates, and implements resolution plans to ensure accurate and reliable analysis.

<table>
<tr><th>Section</th><th>Description</th></tr>
<tr><td>Data Overview</td><td>Structure : 55x15</td></tr>
<tr><td>Univariate Analysis</td><td><pre># Calculate mean, median, and mode for each numerical variable
numerical_columns = data.select_dtypes(include=['number']).columns

mean_values = data[numerical_columns].mean()
median_values = data[numerical_columns].median()
mode_values = data[numerical_columns].mode().iloc[0]  # mode() returns a DataFrame; use .iloc[0] to get the first mode

# Print the results
print("Mean values:\n", mean_values)
print("\nMedian values:\n", median_values)
print("\nMode values:\n", mode_values)</pre></td></tr>
</table>

```
Mean values:
 Population                            8.464170e+06
Area (sq. mi.)                        1.538324e+05
Pop. Density (per sq. mi.)            1.098327e+02
Net migration                        -6.529091e-01
Coastline (coast/area ratio)          4.427091e+00
Phones (per 1000)                              NaN
Arable (%)                            2.500000e+01
Crops (%)                             0.000000e+00
Climate                               2.009091e+00
Birthrate                             2.634509e+01
Deathrate                             8.560727e+00
Agriculture                           1.983636e-01
Industry                              2.448182e-01
Service                               5.480000e-01
GDP ($ per capita)                    4.883636e+03
dtype: float64
```

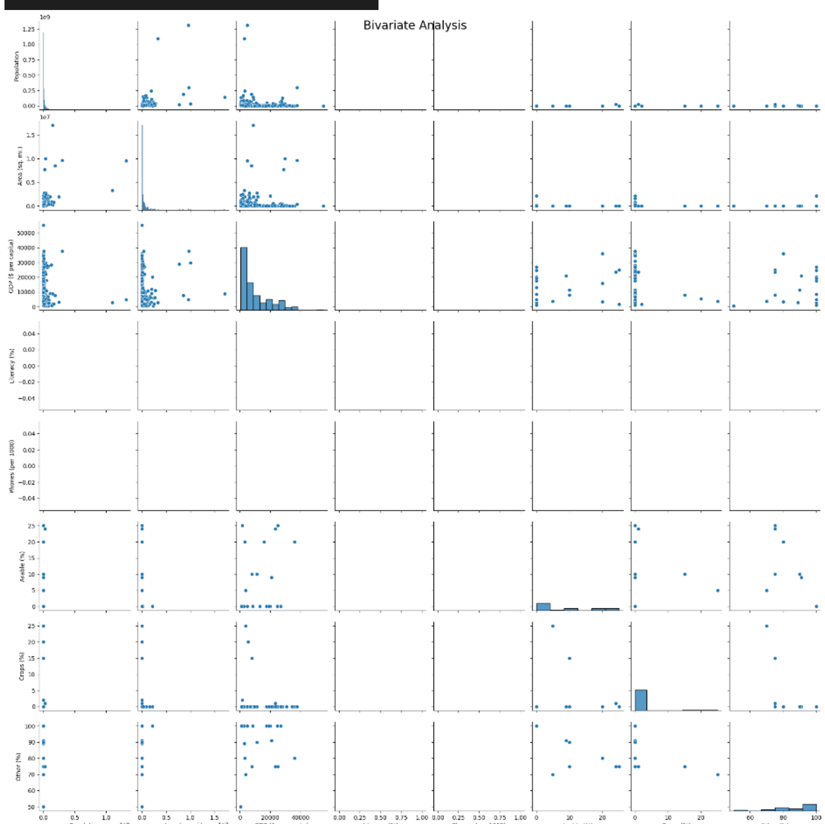
```
Median values:
 Population                            5548702.000
Area (sq. mi.)                          65610.000
Pop. Density (per sq. mi.)                 70.800
Net migration                              -0.060
Coastline (coast/area ratio)                0.710
Phones (per 1000)                             NaN
Arable (%)                                 25.000
Crops (%)                                   0.000
Climate                                     2.000
Birthrate                                  24.510
Deathrate                                   7.820
Agriculture                                 0.172
Industry                                    0.210
Service                                     0.555
GDP ($ per capita)                       2500.000
dtype: float64
```

<table>
<tr>
<td></td>
<td>

```
Mode values:
 Population                         7502.000
Area (sq. mi.)                      413.000
Pop. Density (per sq. mi.)            3.600
Net migration                         0.000
Coastline (coast/area ratio)          0.000
Phones (per 1000)                       NaN
Arable (%)                           25.000
Crops (%)                             0.000
Climate                               2.000
Birthrate                            18.790
Deathrate                            10.310
Agriculture                           0.010
Industry                              0.110
Service                               0.684
GDP ($ per capita)                 1400.000
Name: 0, dtype: float64
```

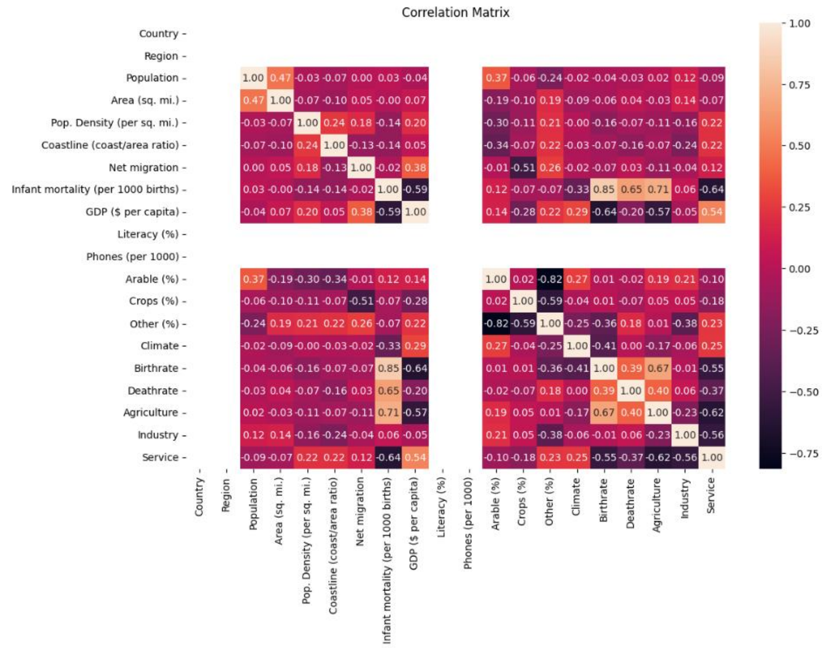
</td>
</tr>
<tr>
<td>Bivariate Analysis</td>
<td>

```python
# Now you can plot the heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(data.corr(), annot=True, fmt=".2f")
plt.title('Correlation Matrix')
plt.show()
```




</td>
</tr>
</table>

Multivariate Analysis




Outliers and Anomalies

```
[22] # Handling outliers (example using IQR method)
     Q1 = data.quantile(0.25)
     Q3 = data.quantile(0.75)
     IQR = Q3 - Q1
     data = data[~((data < (Q1 - 1.5 * IQR)) | (data > (Q3 + 1.5 * IQR))).any(axis=1)]
```

**Data Preprocessing Code Screenshots**

Loading Data

```
data = pd.read_csv('countries of the world.csv')
```

Handling Missing Data

Handling missing values

```
[ ] # Fill missing values in numeric columns with mean of each column
    numeric_cols = data.select_dtypes(include='number').columns
    data[numeric_cols] = data[numeric_cols].fillna(data[numeric_cols].mean())

    # Fill missing values in categorical columns with mode (most frequent value)
    categorical_cols = data.select_dtypes(include='object').columns

    for col in categorical_cols:
        if not data[col].mode().empty:  # Check if mode is not empty
            data[col] = data[col].fillna(data[col].mode().iloc[0])
        else:
            data[col] = data[col].fillna('missing')  # Placeholder for entirely NaN col
```

Data Transformation

```
[ ] # Convert necessary columns to numeric
    columns_to_convert = ['Literacy (%)', 'Phones (per 1000)', 'Arable (%)', 'Crops (%)', 'Other (%)', 'GDP ($ per capita)', 'Population']
    for column in columns_to_convert:
        data[column] = pd.to_numeric(data[column], errors='coerce')
```

```
[ ] ting continuous values into categories

    cy is defined as being able to read and write, or having knowledge about a specific subject
     = ['0-20%' if x<=20.0 else '20-40%' if x>20.0 and x<=40.0 else '40-60%' if x>40.0 and x<=60.0 else '60-80%' if x>60.0 and x<=80.0 else '80-100%' for x in data['Literacy (%)']]


    ['0-200' if x<=200.0 else '200-400' if x>200.0 and x<=400.0 else '400-600' if x>400.0 and x<=600.0 else '600-800' if x>600.0 and x<=800.0 else '800-1000' for x in data['Phones (per 10

     farming is growing crops in fields, which have usually been ploughed before planting
     ['0-20%' if x<=20.0 else '20-40%' if x>20.0 and x<=40.0 else '40-60%' if x>40.0 and x<=60.0 else '60-80%' if x>60.0 and x<=80.0 else '80-100%' for x in data['Arable (%)']]


    ['0-20%' if x<=20.0 else '20-40%' if x>20.0 and x<=40.0 else '40-60%' if x>40.0 and x<=60.0 else '60-80%' if x>60.0 and x<=80.0 else '80-100%' for x in data['Crops (%)']]


     ['0-20%' if x<=20.0 else '20-40%' if x>20.0 and x<=40.0 else '40-60%' if x>40.0 and x<=60.0 else '60-80%' if x>60.0 and x<=80.0 else '80-100%' for x in data['Other (%)']]


    0-10000$' if x<=10000.0 else '10000-20000$' if x>10000.0 and x<=20000.0 else '20000-30000$' if x>20000.0 and x<=30000.0 else '30000-40000$' if x>30000.0 and x<=40000.0 else '40000-50

    tion
    on = ['Below 1 million' if x<=1000000 else '1-20 million' if x>1000000 and x<=20000000 else '20-60 million' if x>20000000 and x<=60000000 else '60-100 million' if x>60000000 and x<=10
```

```
] # Convert all numeric columns that might have commas as decimal separators
  for col in data.columns:
      # Check if the column is of type object
      if data[col].dtype == 'object':
          # Replace commas with dots
          data[col] = data[col].str.replace(',', '.')
          # Convert the column to numeric, forcing errors to NaN
          data[col] = pd.to_numeric(data[col], errors='coerce')

  # Fill any new NaN values that resulted from the conversion
  data = data.fillna(data.mean(numeric_only=True))
```

```
] # Identify categorical columns
  # Replace 'Country' and 'Region' with actual categorical columns in your dataset
  categorical_columns = ['Country', 'Region']  # Example columns
  for col in categorical_columns:
      if col in data.columns:
          data[col] = LabelEncoder().fit_transform(data[col])

  # If there are other categorical features, use OneHotEncoder (example for 'Region')
  if 'Region' in data.columns:
      data = pd.get_dummies(data, columns=['Region'], drop_first=True)
```

Feature Engineering

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
print("Scaled Training Data:")
print(X_train_scaled)
print("Scaled Test Data:")
print(X_test_scaled)
```

```
Scaled Training Data:
[[-6.47219291e-01 -5.16195636e-01 -5.39040510e-01  4.69112293e-01
  -5.84388360e-01            nan  0.00000000e+00  0.00000000e+00
   0.00000000e+00  6.86037311e-01  1.25574014e+00  4.02588627e-01
   1.13967408e+00 -1.27081909e+00]
 [ 2.68692521e+00  1.29694268e+00 -2.86256441e-01 -2.65921213e-01
  -5.24934457e-01            nan  0.00000000e+00  0.00000000e+00
   0.00000000e+00 -5.13249239e-01 -8.80204430e-01  9.79452196e-02
   9.54727805e-01 -8.22694104e-01]
 [ 2.15343342e+00 -6.15791697e-02  8.63471870e-01  4.69112293e-01
  -5.84388360e-01            nan  0.00000000e+00  0.00000000e+00
   0.00000000e+00  4.11650414e-01  2.38766531e-01  1.30908852e+00
  -2.81049550e-01 -9.36503624e-01]
 [-8.18309170e-01 -7.19631979e-01  2.09518420e+00  4.69112293e-01
   1.68791080e+00            nan  0.00000000e+00  0.00000000e+00
   0.00000000e+00  1.02311185e+00 -9.42248275e-02  1.45769506e+00
  -1.71017982e+00  1.30460629e-01]
 [-7.15940504e-01 -6.78200486e-01  4.05727746e-01  1.64666597e+00
  -4.81431602e-01            nan  0.00000000e+00  0.00000000e+00
```

<table>
<tr>
<td>Save Processed Data</td>
<td>

Save processed data

```
## Step 3: Save the processed data to a CSV file
data.to_csv('processed_data.csv', index=False)

# To save in Excel format
data.to_excel('processed_data.xlsx', index=False)

# If you prefer to save in a database, you can use SQLAlchemy
from sqlalchemy import create_engine



print("Data saved successfully.")
```

```
Data saved successfully.
```

</td>
</tr>
</table>