SMARTBRIDGE
Let's Bridge the Gap

Smart Internz

Economic Growth

A Machine Learning Approach to GDP Per Capital Prediction

June 21, 2025

GitHub, GDPredict

Ujjwal Hirwani

Sanket Kashyap

Aradhya Kumar Mishra

Project Flow:

1. User interacts with the UI to enter the input.
2. Entered input is analysed by the model which is integrated.
3. Once model analyses the input the prediction is showcased on the UI To accomplish this, we have to complete all the activities listed below,
4. Define Problem / Problem Understanding
    a. Specify the business problem
    b. Business requirements
    c. Literature Survey
    d. Social or Business Impact.
5. Data Collection &Preparation
    a. Collect the dataset
    b. Data Preparation
6. Exploratory Data Analysis
    a. Descriptive statistical
    b. Visual Analysis
7. Model Building
    a. Training the model in multiplealgorithms
    b. Testing the model
8. Performance Testing & Hyperparameter Tuning
    a. Testing model with multiple evaluation metrics
    b. Comparing model accuracybefore & after applying hyperparameter tuning
9. Model Deployment
    a. Save the best model
    b. Integrate with Web Framework
10. Project Demonstration &Documentation
    a. Record explanation Videofor project end to end solution
    b. Project Documentation-Step by step projectdevelopment procedure

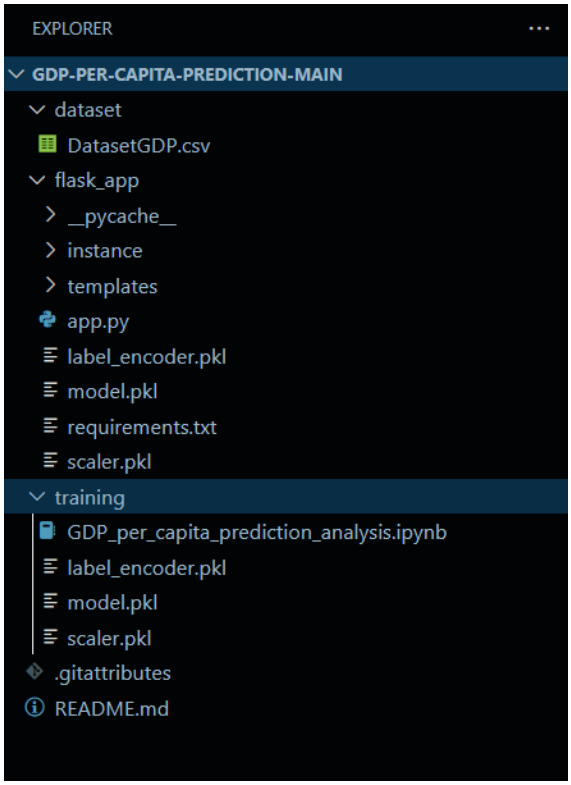
Prior Knowledge:

You must have prior knowledge of following topics to completethis project.

1. ML Concepts
    a. Supervised learning: https://www.javatpoint.com/supervised-machine-learning
    b. Unsupervised learning: https://www.javatpoint.com/unsupervised-machine-learning
2. Decision tree: https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm
3. Random forest: https://www.javatpoint.com/machine-learning-random-forest-algorithm
4. KNN: https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning
5. Xgboost: https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to- understand-the-math-behind-xgboost/
6. Evaluation metrics: https://www.analyticsvidhya.com/blog/2019/08/11-important-model- evaluation-error-metrics/

Flask Basics : https://www.youtube.com/watch?v=lj4I_CvBnt0

Project Structure:



1. We are building a flask application which needs HTML pages stored in the templates folder and a python script app.py for scripting.
2. Dtc_model.pkl is our saved model. Further we will use this model for flask integration.
3. Data Folder containsthe Dataset used
4. The Notebook file contains procedurefor building the model.

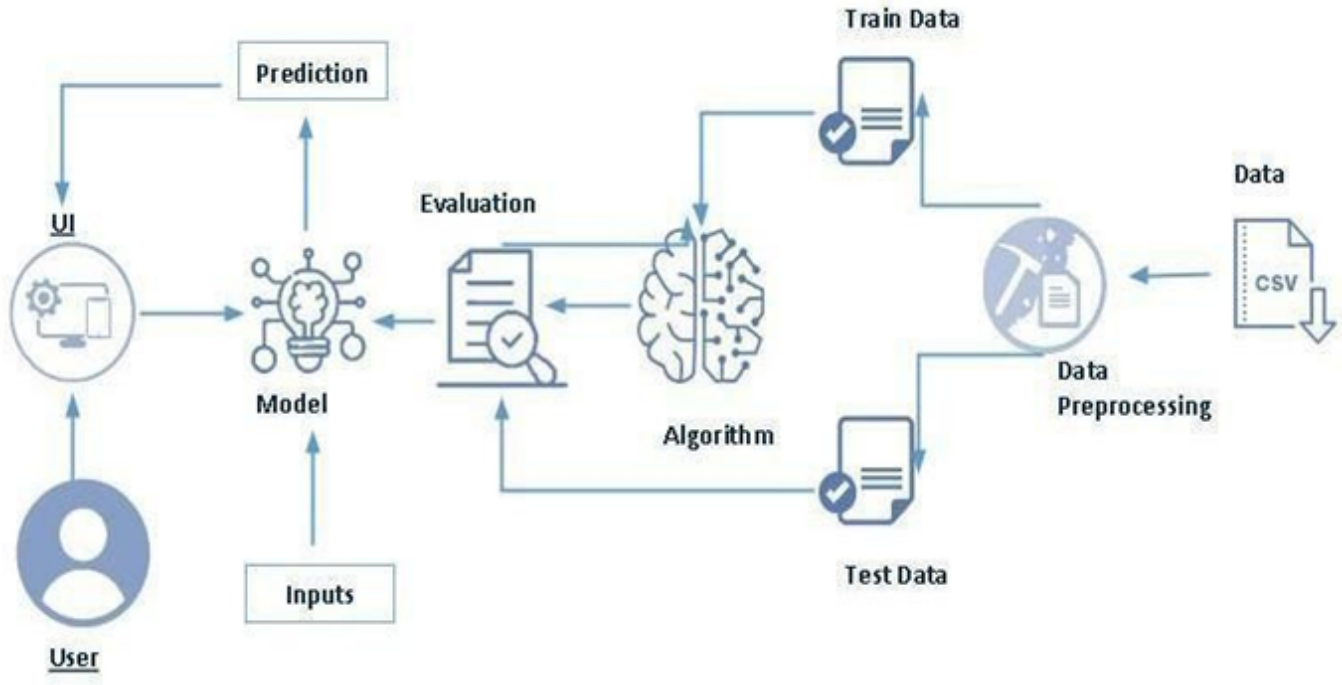
Economic Growth : A Machine Learning Approach to GDP per Capita Prediction

Economic growth is a vital indicator of a country's development and prosperity. Among various economic metrics, **GDP per capita** stands out as a crucial measure of the average economic output per person, reflecting living standards and productivity levels. Accurately forecasting GDP per capita can help governments, investors, and policymakers make informed decisions regarding resource allocation, policy reforms, and future investments.

In this project, we leverage **machine learning techniques** to build a predictive model that estimates GDP per capita using a wide range of socio-economic, demographic, and geographic features. These include factors such as population, literacy rate, birth and death rates, net migration, and economic sector contributions (agriculture, industry, and service). By applying data-driven insights, the model aims to capture complex relationships that influence economic performance, offering a modern approach to economic forecasting beyond traditional statistical methods.

This project demonstrates how machine learning can serve as a powerful tool in economic analysis, helping anticipate trends, simulate scenarios, and support decision-making in an increasingly data-rich world.

Technical Architecture:-



Milestone 1: Define Problem / ProblemUnderstanding

Business Problem

The project aims to utilize machine learning techniques for forecasting GDP per capita, a crucial indicator of economic prosperity. GDP per capita reflects the average income earned per person in a country and serves as a key measure of living standards and economic development. By leveraging historical GDP data along with a wide range of economic, social, and demographic factors, the machine learning model can generate forecasts that help policymakers, investors, and businesses understand and anticipate future economic trends.

Business requirements

Scenario 1:

A country implements significant policy reforms aimed at boosting entrepreneurship and innovation. The machine learning model, trained on historical data that includes the impact of similar policy changes in other regions, predicts a corresponding increase in GDP per capita over the forecast period.

Scenario 2:

An unexpected global economic downturn occurs due to factors such as a financial crisis or a pandemic. The machine learning model, trained on historical economic shocks and their effects on GDP per capita, provides forecasts reflecting the potential decline in economic prosperity, enabling stakeholders to take preemptive measures.

Scenario 3:

Technological advancements, such as automation and artificial intelligence adoption, lead to productivity gains in certain sectors of the economy. The machine learning model incorporates data on technological trends and their impact on productivity to forecast increases in GDP per capita, helping policymakers and businesses prepare for shifts in the economic landscape.

Literature Survey

Several studies have explored the application of machine learning techniques in economic forecasting, especially in predicting macroeconomic indicators like GDP per capita.

1. Traditional vs. ML Approaches:

Historically, GDP forecasting has relied on econometric models such as ARIMA, VAR, and linear regression. However, these models often struggle with high-dimensional, nonlinear relationships. Machine learning overcomes these limitations by automatically detecting complex patterns and interactions.

2. Socio-Economic Indicators in Prediction:

Studies like those by *Mayer et al. (2019)* and *Abedin et al. (2021)* demonstrate that incorporating social, demographic, and infrastructure variables — such as literacy rate, population, and telecommunication access — significantly improves the accuracy of GDP predictions.

3. Popular Algorithms Used:

Research has shown that algorithms like **Random Forest, XGBoost, and Support Vector Machines** outperform traditional models in predictive accuracy due to their ability to handle multicollinearity, non-linear patterns, and missing data (e.g., *Chen et al., 2020*).

4. Visualization and EDA Importance:
As emphasized by *Kaggle competitions and World Bank analysis reports*, data preprocessing steps such as outlier detection, feature scaling, and visualization (via boxplots, heatmaps, and pairplots) are essential for meaningful model input and interpretation.

5. Model Deployment:
Recent studies have highlighted the importance of real-world integration using web technologies like **Flask** for deployment, making predictive models accessible for decision-makers (*Rahman et al., 2022*).

Social or Business Impact

1. Social Impact:
   - Policy Planning: Accurate GDP per capita forecasts can guide governments in creating effective economic and social policies, especially in areas like education, healthcare, and employment.
   - Resource Allocation: Helps ensure fair and efficient distribution of public resources by identifying regions or sectors with lagging growth.
   - Early Warning System: Predictive models can signal economic downturns, enabling timely interventions to protect vulnerable populations.
2. Business Impact:
   - Investment Decisions: Businesses and investors can use GDP forecasts to assess market potential, plan expansions, and minimize risks in emerging or unstable economies.
   - Strategic Planning: Companies can tailor their strategies based on anticipated economic trends, such as rising income levels or shifts in consumer demand.
   - Sectoral Insights: By analyzing feature contributions (e.g., agriculture, industry, services), businesses gain a deeper understanding of which sectors drive growth and where opportunities lie.

Overall, the integration of machine learning into economic forecasting democratizes data access, enhances decision-making, and drives both societal well-being and economic resilience.

Milestone 2 : Data Collection and Preperation

ML depends heavilyon data. It is the most crucialaspect that makesalgorithm training possible. So, this section allows you to download the required dataset

2.1 Collect the Dataset

There are many popularopen sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

In this project we have used .csv data. This data is downloaded from kaggle.com. Pleaserefer to the link given below to download the dataset.

Link: https://www.kaggle.com/datasets/fernandol/countries-of-the-world

As the dataset is downloaded. Let us read and understand the data properlywith the help of some visualisation techniques and some analysing techniques.

2.2 Importing the libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import cross_val_score
import pickle
```

2.3 Read the Dataset

We can read the datasetwith the help of pandas.

In pandas we have a function calledread_csv() to read the dataset.As a parameter we have to give the directory of the csv file.

Step 3: Read the Dataset

```
#data collection
df = pd.read_csv(r"D:\ujjwal\projects\GDP-per-capita-prediction\dataset\DatasetGDP.csv")
df.head()
```

[2] Python

	Country	Region	Population	Area (sq. mi.)	Pop. Density (per sq. mi.)	Coastline (coast/area ratio)	Net migration	Infant mortality (per 1000 births)	GDP ($ per capita)	Literacy (%)	Phones (per 1000)	Arable (%)	Crops (%)	Other (%)	Climate	Birthrate	Deathrate	Agriculture	Industry
0	Afghanistan	ASIA (EX. NEAR EAST)	31056997	647500	48.0	0.00	23.06	163.07	700.0	36.0	3.2	12.13	0.22	87.65	1.0	46.60	20.34	0.380	0.240
1	Albania	EASTERN EUROPE	3581655	28748	124.6	1.26	-4.93	21.52	4500.0	86.5	71.2	21.09	4.42	74.49	3.0	15.11	5.22	0.232	0.188
2	Algeria	NORTHERN AFRICA	32930091	2381740	13.8	0.04	-0.39	31.00	6000.0	70.0	78.1	3.22	0.25	96.53	1.0	17.14	4.61	0.101	0.600
3	American Samoa	OCEANIA	57794	199	290.4	58.29	-20.71	9.27	8000.0	97.0	259.5	10.00	15.00	75.00	2.0	22.46	3.27	NaN	NaN
4	Andorra	WESTERN EUROPE	71201	468	152.1	0.00	6.60	4.05	19000.0	100.0	497.2	2.22	0.00	97.78	3.0	8.71	6.25	NaN	NaN

```
df.info()
df.describe(include="all")
```

[5]

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 227 entries, 0 to 226
Data columns (total 20 columns):
 #   Column                              Non-Null Count  Dtype
---  ------                              --------------  -----
 0   Country                             227 non-null    object
 1   Region                              227 non-null    object
 2   Population                          227 non-null    int64
 3   Area (sq. mi.)                      227 non-null    int64
 4   Pop. Density (per sq. mi.)          227 non-null    float64
 5   Coastline (coast/area ratio)        227 non-null    float64
 6   Net migration                       224 non-null    float64
 7   Infant mortality (per 1000 births)  224 non-null    float64
 8   GDP ($ per capita)                  226 non-null    float64
 9   Literacy (%)                        209 non-null    float64
 10  Phones (per 1000)                   223 non-null    float64
 11  Arable (%)                          225 non-null    float64
 12  Crops (%)                           225 non-null    float64
 13  Other (%)                           225 non-null    float64
 14  Climate                             205 non-null    float64
 15  Birthrate                           224 non-null    float64
 16  Deathrate                           223 non-null    float64
 17  Agriculture                         212 non-null    float64
 18  Industry                            211 non-null    float64
 19  Service                             212 non-null    float64
```

2.3 Data Preperations

The download data set is not suitablefor training the machine learningmodel as it might have so much randomness so we need to clean the dataset properly in order to fetch good results. This activity includes the following steps.

a. Handling missing values
b. Handling Outliers

Note: These are the generalsteps of pre-processing the data beforeusing it for machine learning. Depending on the condition of your dataset, you may or may not have to go through all these steps.

2.3.1 Handling Missing Values

For checking the null values,df.isna().any( ) functionis used. To sum thosenull values we use .sum() function

Then we are going to fill the missing values using either mean(), median() or mode().
Otherwise we can even drop those rows entierly

Since the number of missing entries was relatively small, this approach helped maintain data quality without significantly reducing dataset size.

```
# Dropping uneccessary values
df = df.dropna(subset=['Net migration'])
df = df.dropna(subset=['GDP ($ per capita)'])
df = df.dropna(subset=['Phones (per 1000)'])
df = df.dropna(subset=['Arable (%)'])
df = df.dropna(subset=['Crops (%)'])
df = df.dropna(subset=['Birthrate'])
df = df.dropna(subset=['Deathrate'])
```

```
df['Literacy (%)'] = df['Literacy (%)'].fillna(df['Literacy (%)'].mean())
df['Climate'] = df['Climate'].fillna(df['Climate'].mean())
df['Service'] = df['Service'].fillna(df['Service'].mean())
df['Agriculture'] = df['Agriculture'].fillna(df['Agriculture'].mean())
df['Industry'] = df['Industry'].fillna(df['Industry'].median())
```

[19]

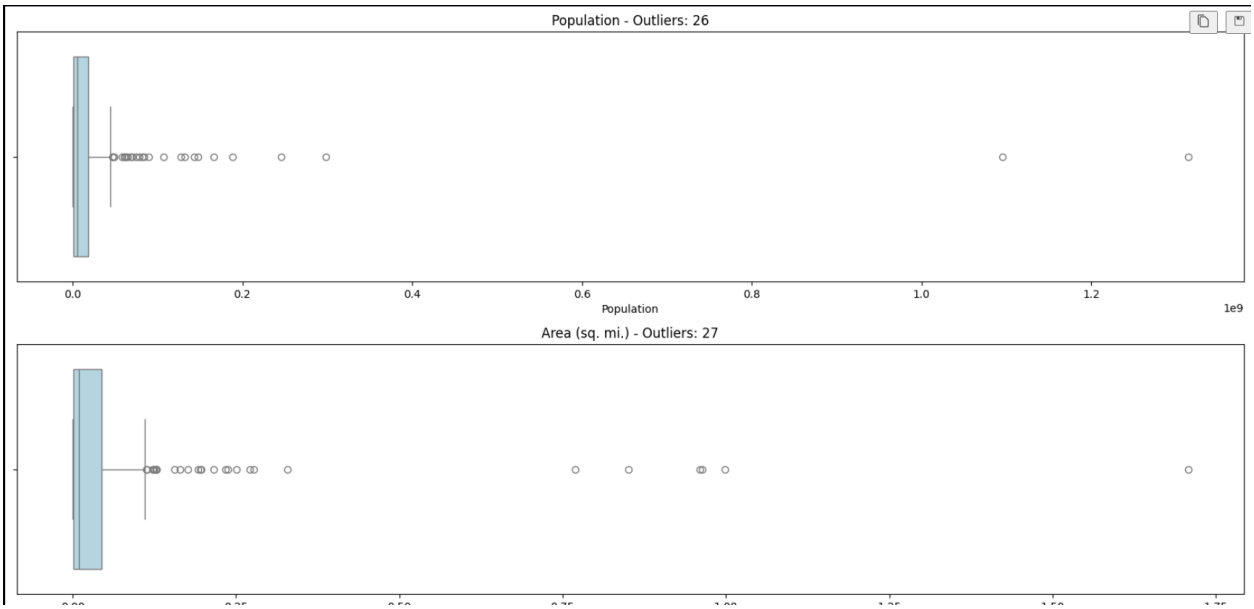
```
df.isnull().sum()
```

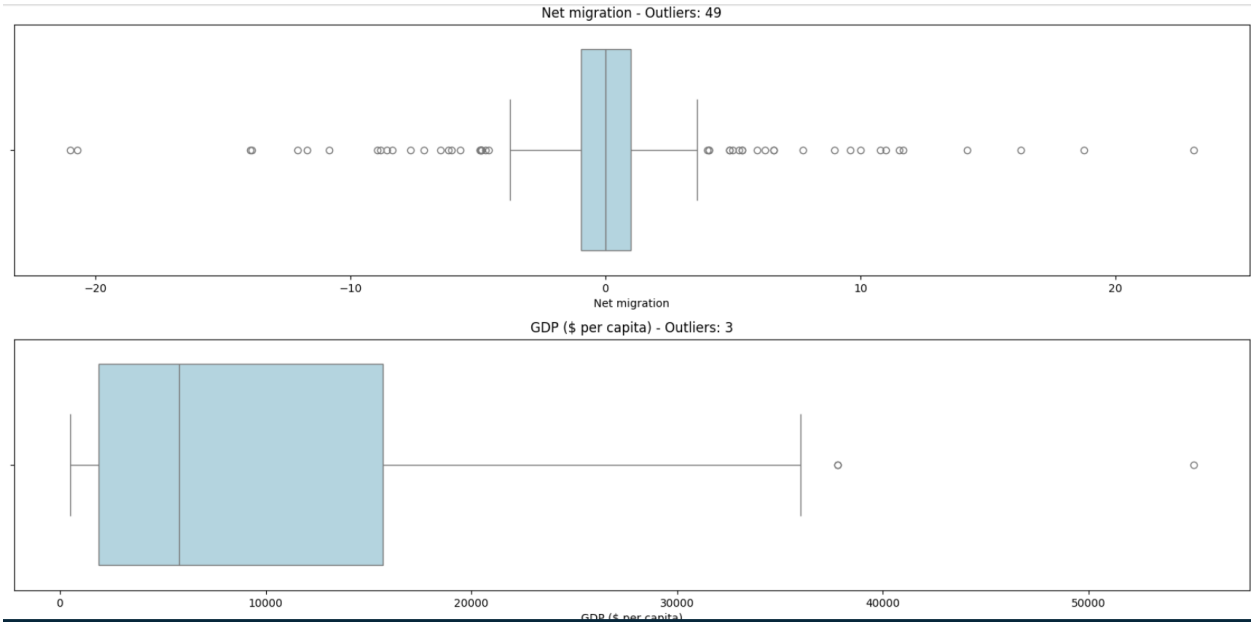
[20]

```
Region                            0
Population                        0
Area (sq. mi.)                    0
Pop. Density (per sq. mi.)        0
Coastline (coast/area ratio)      0
Net migration                     0
GDP ($ per capita)                0
Literacy (%)                      0
Phones (per 1000)                 0
Arable (%)                        0
Crops (%)                         0
Climate                           0
Birthrate                         0
Deathrate                         0
Agriculture                       0
Industry                          0
Service                           0
dtype: int64
```

2.3.2 Handling Outliers

Box plots help identify outliers by showing data distribution through quartiles. Any data point lying **below Q1 - 1.5×IQR** or **above Q3 + 1.5×IQR** is considered an outlier and appears as individual points outside the whiskers.





	Feature	Outlier Count
4	Net migration	49
3	Coastline (coast/area ratio)	37
10	Climate	34
9	Crops (%)	28
1	Area (sq. mi.)	27
0	Population	26
2	Pop. Density (per sq. mi.)	19
6	Literacy (%)	16
12	Deathrate	16
14	Industry	12
8	Arable (%)	9
13	Agriculture	7
5	GDP ($ per capita)	3
7	Phones (per 1000)	1
15	Service	1
11	Birthrate	0

Outliers were identified in many numerical features. To handle them without dropping valuable data, we applied two transformation techniques:

- Logarithmic Transformation (for reducing right skew)
- Square Root Transformation (for moderately skewed data)

For each feature:
- We calculated the skewness before and after applying both transformations.
- The transformation that minimized the skewness (closer to 0) was retained.

Features with negative or zero values were excluded from log transformation to avoid undefined results.

This step helps normalize the distribution, stabilize variance, and reduce the impact of outliers on model training.

GDP_per_capita_prediction_analysis.ipynb

C: > Users > sanke > Desktop > GDP-per-capita-prediction-main > training > GDP_per_capita_prediction_analysis.ipynb > M↓ S

Generate | + Code | + Markdown | Run All | Clear All Outputs | Outline | ···

```python
# DataFrames to store transformed versions
log_df = df.copy()
sqrt_df = df.copy()
final_df = df.copy()

for col in numerical_cols:
    # Avoid transformation for values <= 0 (to prevent log(0) issues)
    if (df[col] <= 0).any():
        continue

    # Apply transformations
    log_transformed = np.log(df[col])
    sqrt_transformed = np.sqrt(df[col])

    # Calculate skewness
    orig_skew = df[col].skew()
    log_skew = log_transformed.skew()
    sqrt_skew = sqrt_transformed.skew()

    # Select the transformation that brings skewness closer to 0
    if abs(log_skew) < abs(sqrt_skew) and abs(log_skew) < abs(orig_skew):
        final_df[col] = log_transformed
        best_transforms[col] = "log"
    elif abs(sqrt_skew) < abs(orig_skew):
        final_df[col] = sqrt_transformed
        best_transforms[col] = "sqrt"
    else:
        best_transforms[col] = "none"

# Print best transformation method for each feature
for feature, method in best_transforms.items():
    print(f"{feature}: {method} transformation applied")
```

```
Population: log transformation applied
Area (sq. mi.): log transformation applied
GDP ($ per capita): log transformation applied
Literacy (%): none transformation applied
Phones (per 1000): sqrt transformation applied
Climate: sqrt transformation applied
Birthrate: log transformation applied
Deathrate: log transformation applied
Industry: sqrt transformation applied
Service: none transformation applied
```

Milestone 3 : Exploratory Data Analysis

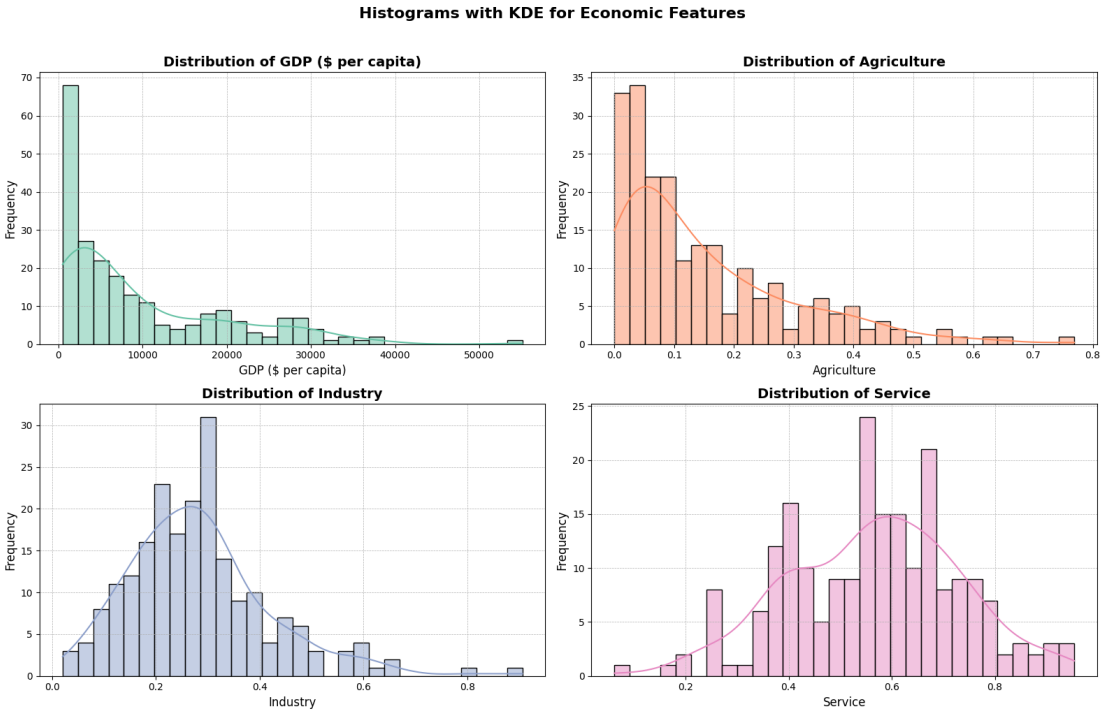
3.1 Descriptive Analysis

The `pandas.DataFrame.describe()` function provides summary statistics (like count, mean, std, min, max, and quartiles) for each numerical column, helping you quickly understand the distribution of your data.

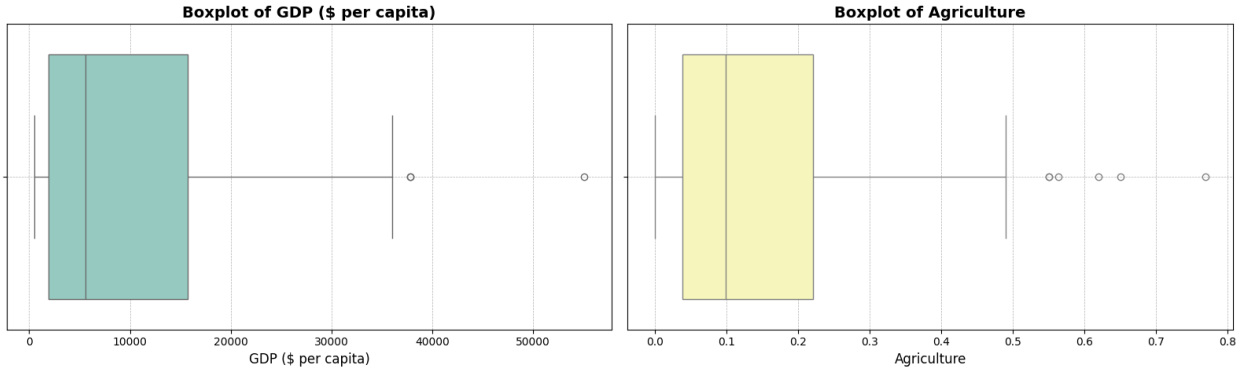
	Country	Region	Population	Area (sq. mi.)	Pop. Density (per sq. mi.)	Coastline (coast/area ratio)	Net migration	Infant mortality (per 1000 births)	GDP ($ per capita)	Literacy (%)	Phones (per 1000)	Arable (%)	Crops (%)	Other (%)	Climat
count	227	227	2.270000e+02	2.270000e+02	227.000000	227.000000	224.000000	224.000000	226.000000	209.000000	223.000000	225.000000	225.000000	225.000000	205.00000
unique	227	11	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal
top	Afghanistan	SUB-SAHARAN AFRICA	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal
freq	1	51	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal
mean	NaN	NaN	2.874028e+07	5.982270e+05	379.047137	21.165330	0.038125	35.506964	9689.823009	82.838278	236.061435	13.797111	4.564222	81.638311	2.13902
std	NaN	NaN	1.178913e+08	1.790282e+06	1660.185825	72.286863	4.889269	35.389899	10049.138513	19.722173	227.991829	13.040402	8.361470	16.140835	0.69939
min	NaN	NaN	7.026000e+03	2.000000e+00	0.000000	0.000000	-20.990000	2.290000	500.000000	17.600000	0.200000	0.000000	0.000000	33.330000	1.00000
25%	NaN	NaN	4.376240e+05	4.647500e+03	29.150000	0.100000	-0.927500	8.150000	1900.000000	70.600000	37.800000	3.220000	0.190000	71.650000	2.00000
50%	NaN	NaN	4.786994e+06	8.660000e+04	78.800000	0.730000	0.000000	21.000000	5550.000000	92.500000	176.200000	10.420000	1.030000	85.700000	2.00000
75%	NaN	NaN	1.749777e+07	4.418110e+05	190.150000	10.345000	0.997500	55.705000	15700.000000	98.000000	389.650000	20.000000	4.440000	95.440000	3.00000
max	NaN	NaN	1.313974e+09	1.707520e+07	16271.500000	870.660000	23.060000	191.190000	55100.000000	100.000000	1035.600000	62.110000	50.680000	100.000000	4.00000

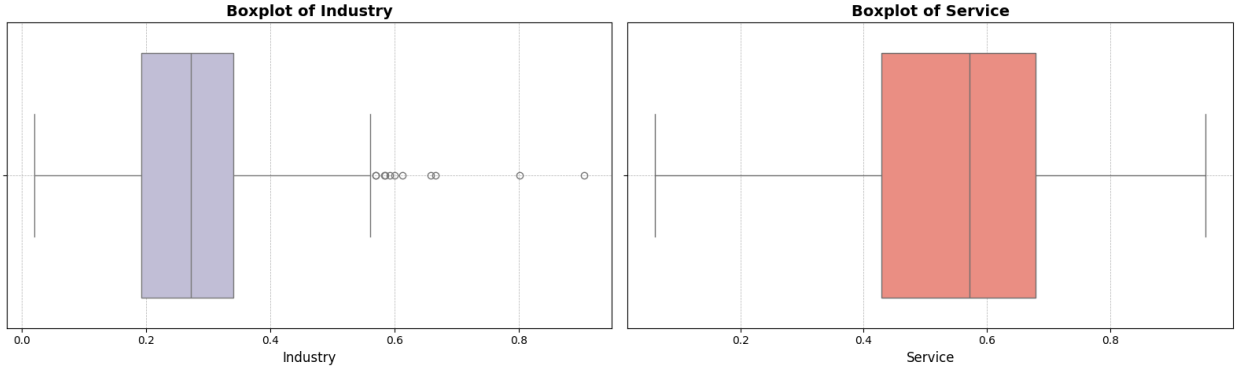
3.2 Univariate Analysis

Univariate analysis examines one variable at a time to understand its distribution, central tendency, and spread. **Histograms** are good for visualizing frequency distribution, while **boxplots** help detect spread, symmetry, and outliers in the data.

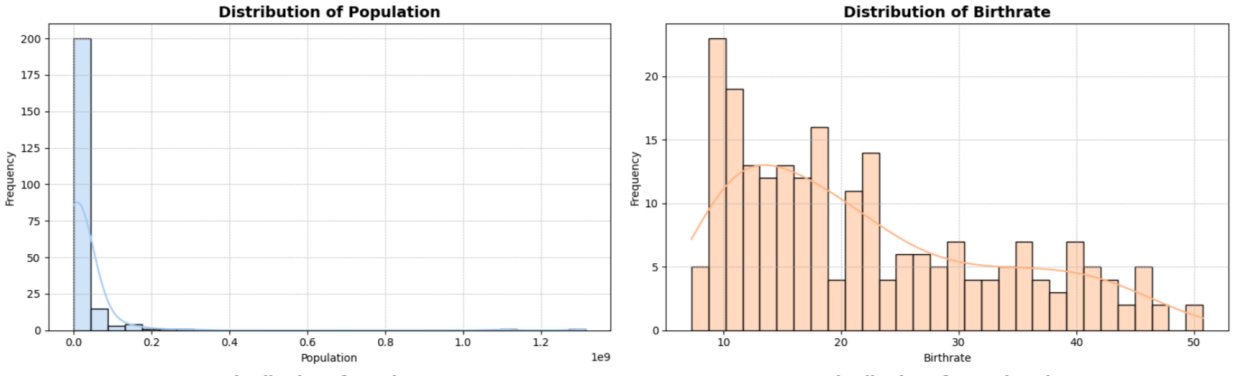


Boxplots for Economic Features

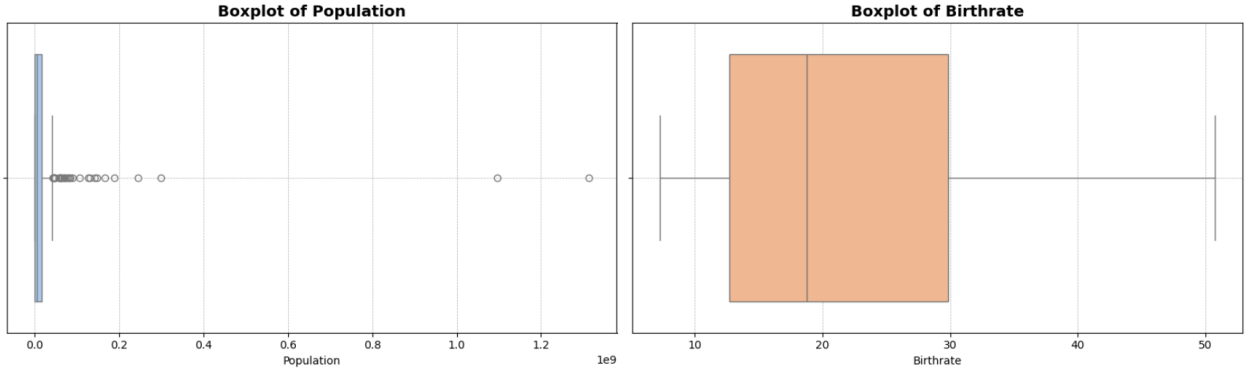




Demographic Features - Histograms with KDE



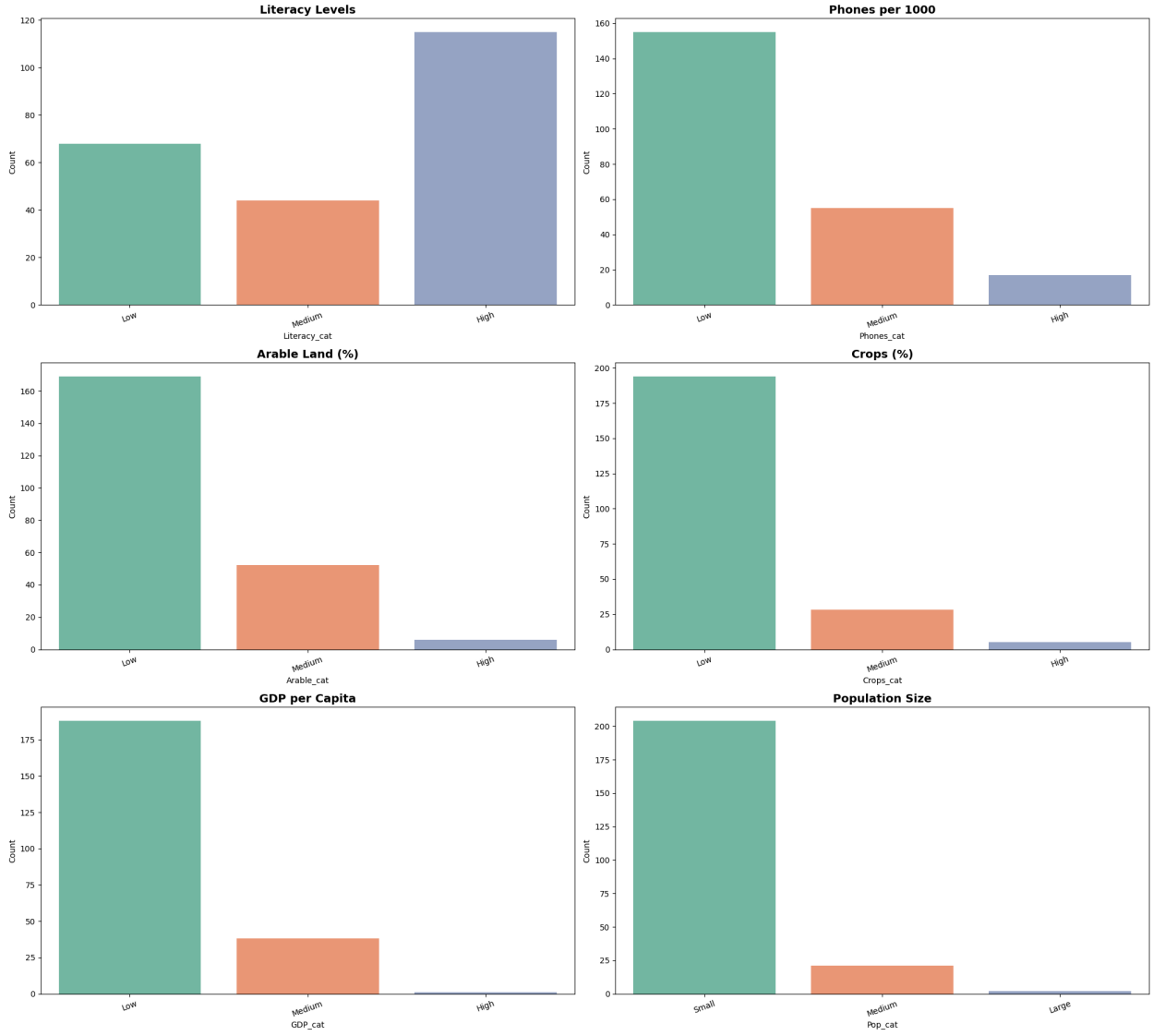
Demographic Features - Boxplots



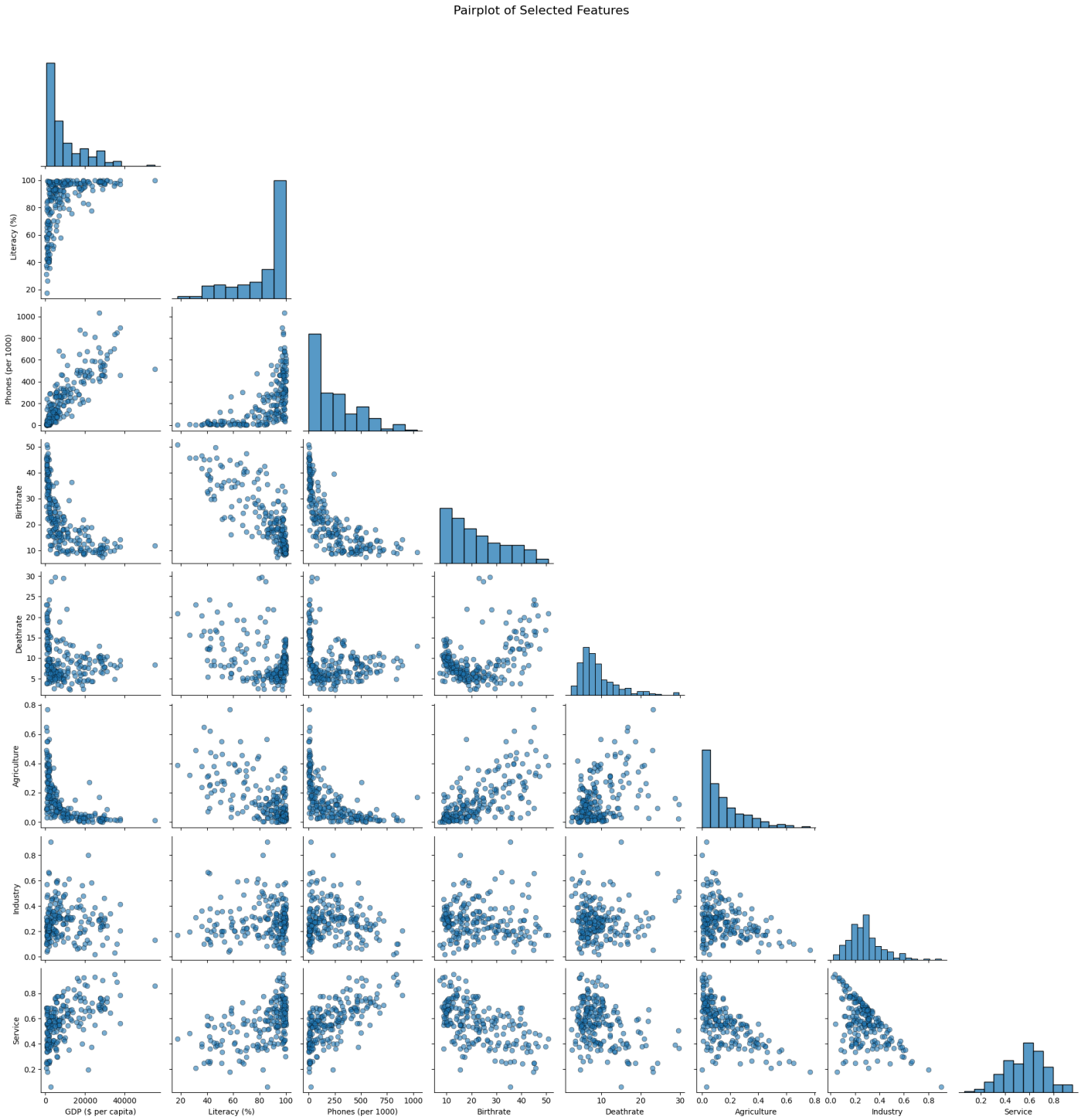
3.3 Bivariate Analysis

Bivariate analysis explores the relationship between two variables to identify patterns or correlations. **Countplots** are useful when one or both variables are categorical, as they clearly show the frequency of occurrences for each category, making comparisons easy.

```
plt.figure(figsize=(20, 18))
features = ['Literacy_cat', 'Phones_cat', 'Arable_cat', 'Crops_cat', 'GDP_cat', 'Pop_cat']
titles = ['Literacy Levels', 'Phones per 1000', 'Arable Land (%)', 'Crops (%)', 'GDP per Capita', 'Population Size']

for i, col in enumerate(features):
    plt.subplot(3, 2, i + 1)
    sns.countplot(x=col, hue=col, data=df, palette='Set2', legend=False)
    plt.title(f'{titles[i]}', fontsize=14, fontweight='bold')
    plt.xlabel(col)
    plt.ylabel("Count")
    plt.xticks(rotation=20)

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



Pairplots are excellent for bivariate analysis because they show scatter plots for every pair of numerical features, helping visualize relationships, patterns, and correlations. They also include histograms or KDE plots along the diagonal for univariate insights.

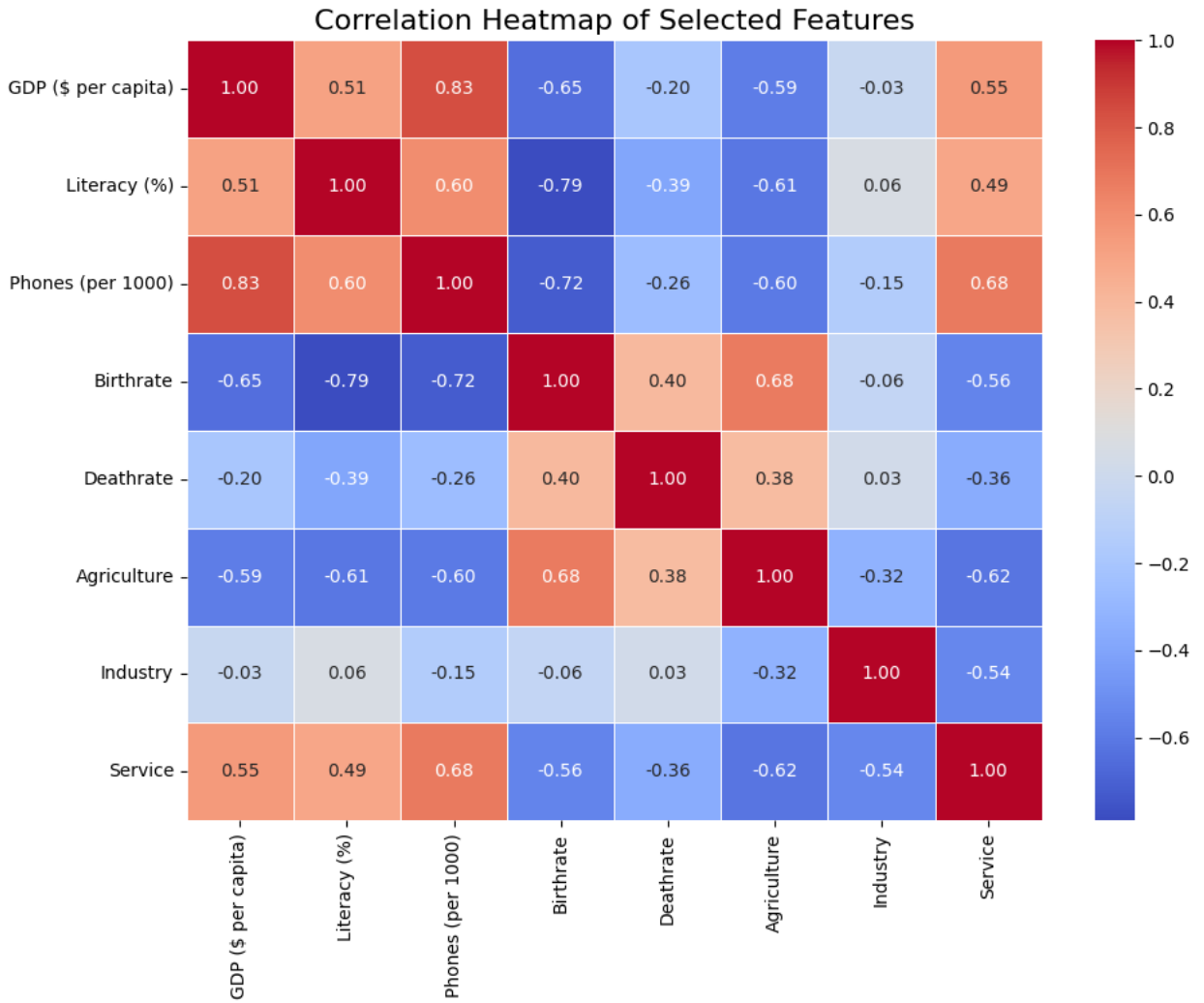


3.4 Multivariate Analysis

Multivariate analysis involves examining three or more variables to understand complex relationships and interactions among them.

A **correlation heatmap** is useful in multivariate analysis as it visually displays the strength and direction of linear relationships between numerical features, making it easier to identify strongly correlated variables and multicollinearity.

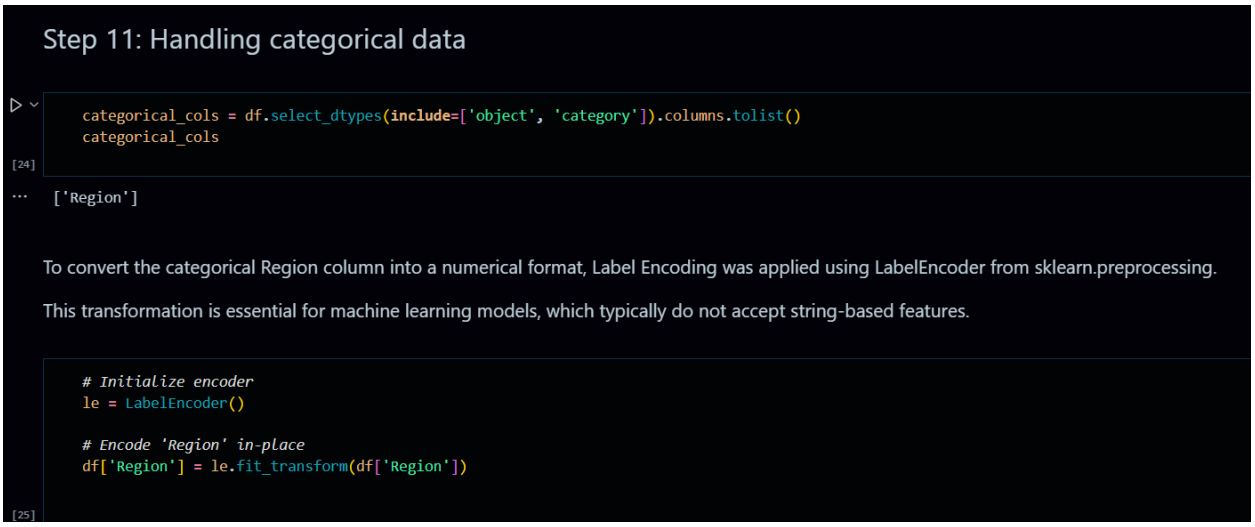
```
selected_features = ['GDP ($ per capita)', 'Literacy (%)', 'Phones (per 1000)',
                     'Birthrate', 'Deathrate', 'Agriculture', 'Industry', 'Service']

correlation_matrix = df[selected_features].corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title("Correlation Heatmap of Selected Features", fontsize=16)
plt.tight_layout()
plt.show()
```



3.5 Handling Categorical Columns

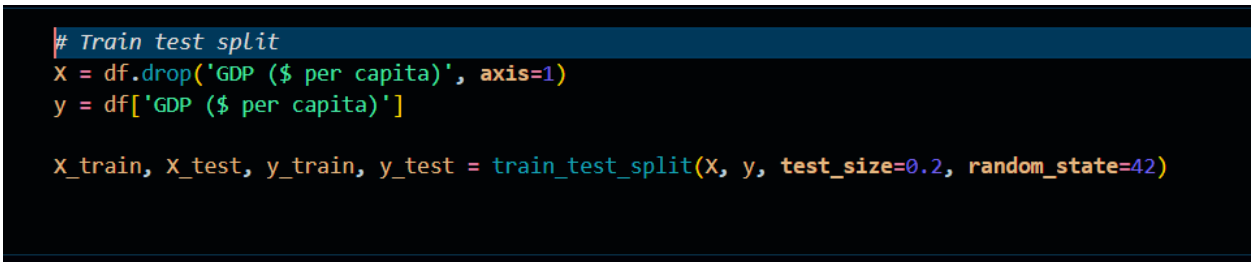
Handling categorical columns is important because machine learning models require numerical input. Unprocessed categorical data can't be interpreted by most algorithms.

LabelEncoder converts categorical values into numeric labels, making them usable by models while preserving category information. It's best for ordinal data or when categories are not one-hot encoded.



3.6 Training and Testing Data Split

Train-test split is a technique used to evaluate machine learning models by dividing the dataset into two parts: a **training set** (used to train the model) and a **testing set** (used to assess its performance).
This helps ensure the model generalizes well to unseen data and avoids overfitting. Typically, 70–80% of the data is used for training, and the rest for testing.



3.7 Scaling the Data

Scaling the data is crucial because many machine learning algorithms are sensitive to feature magnitudes. Features with larger ranges can dominate the learning process and lead to biased models.

StandardScaler standardizes features by removing the mean and scaling to unit variance (z = (x - μ) / σ). It's widely used as it keeps the distribution centered and is effective for algorithms like linear regression.

```
scaler = StandardScaler()

# Fit on training data and transform
X_train_scaled = scaler.fit_transform(X_train)

# Only transform test data
X_test_scaled = scaler.transform(X_test)
```

Milestone 4 : Model Training

4.1 Linear Regression

A function named linear_reg is created and train and test data are passed as the parameters. Inside the function, Linear Regression() algorithm is initialized and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in a new variable. Model score is calculated by r2_score() and mean_squared_error() is used to find error.

```
def linear_reg(X_train_scaled, X_test_scaled, y_train, y_test):
    lr = LinearRegression()
    lr.fit(X_train_scaled, y_train)
    y_pred = lr.predict(X_test_scaled)
    score = r2_score(y_test, y_pred)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))

    print("*** Linear Regression Model ***")
    print("Score for Linear Regression model is {}".format(score))
    print("RMSE for Linear Regression model is {}".format(rmse))
```

4.2 Random Forest Regressor

A function named random_forest_regressor is created and train and test data are passed as the parameters. Inside the function, RandomForestRegressor() algorithm is initialized and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in a new variable. Model score is calculated by r2_score() and mean_squared_error() is used to find error.

```
def random_forest_regressor(X_train_scaled, X_test_scaled, y_train, y_test):
    rf = RandomForestRegressor()
    rf.fit(X_train_scaled, y_train)
    y_pred = rf.predict(X_test_scaled)
    score = r2_score(y_test, y_pred)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))

    print("*** Random Forest Regressor Model ***")
    print("Score for Random Forest Regressor Model is {}".format(score))
    print("RMSE for Random Forest Regressor Model is {}".format(rmse))
```

4.3 Support Vector Regression

A function named svr_model is created and train and test data are passed as the parameters. Inside the function, SVR() algorithm is initialized and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in a new variable. Model score is calculated by r2_score() and mean_squared_error() is used to find error.

```
def svr_model(X_train_scaled, X_test_scaled, y_train, y_test):
    svr = SVR()
    svr.fit(X_train_scaled, y_train)
    y_pred = svr.predict(X_test_scaled)

    score = r2_score(y_test, y_pred)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))

    print("*** SVR Model ***")
    print("Score for SVR Model is {}".format(score))
    print("RMSE for SVR Model is {}".format(rmse))
```

Milestone 5 : Performance Testing and HyperParamter Tuning

5.1 Model Comparison

For comparing the above three models compareModel function is defined.
After calling the function, the results of models are displayed as output. From the three model random forest regression is performing well. From the below image, we can see the accuracy of the models and error of the models. Random forest regression has high accuracy and less error.

```
def model_compare(X_train_scaled, X_test_scaled, y_train, y_test):
    linear_reg(X_train_scaled, X_test_scaled, y_train, y_test)
    print('-' * 100)

    random_forest_regressor(X_train_scaled, X_test_scaled, y_train, y_test)
    print('-' * 100)

    svr_model(X_train_scaled, X_test_scaled, y_train, y_test)
```

```
model_compare(X_train_scaled, X_test_scaled, y_train, y_test)
```

```
*** Linear Regression Model ***
Score for Linear Regression model is 0.7826114237194834
RMSE for Linear Regression model is 4649.544639823302
----------------------------------------------------------------------------------------------------
*** Random Forest Regressor Model ***
Score for Random Forest Regressor Model is 0.9068044568724314
RMSE for Random Forest Regressor Model is 3044.3153283097695
----------------------------------------------------------------------------------------------------
*** SVR Model ***
Score for SVR Model is -0.26118547459767205
RMSE for SVR Model is 11199.059258238174
```

5.2 HyperParameter Tuning

```python
# Initialize model with tuned hyperparameters
rf = RandomForestRegressor(
    n_estimators=150,
    max_depth=20,
    min_samples_split=4,
    random_state=42
)

# Fit the model
rf.fit(X_train_scaled, y_train)

# Predict on test set
y_pred = rf.predict(X_test_scaled)

# Evaluate model
r2 = r2_score(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred)

print(f"R² Score: {r2:.4f}")
print(f"RMSE: {rmse:.2f}")
```

```
R² Score: 0.9115
RMSE: 8804849.01
```

We performed hyperparameter tuning on the Random Forest Regressor using parameters like `n_estimators`, `max_depth`, and `min_samples_split` in an attempt to improve model performance. However, the results after tuning were not satisfactory:

Before Tuning

R² Score: 0.9157

RMSE: 2895.98

After Tuning

R² Score: 0.9115

RMSE: 8804849.01 ×

Despite a similar R² score, the RMSE increased drastically after tuning, indicating that the model's

predictions became significantly less accurate in terms of absolute error. This suggests potential issues such as improper scaling, overfitting, or unsuitable hyperparameters.

Conclusion:

We chose to proceed with the **original Random Forest Regressor model** (before tuning), as it offers better predictive accuracy and lower error in GDP per capita estimation.

5.3 Evaluating Performance and Saving the Model

From sklearn, cross_val_score is used to evaluate the score of the model. On the parameters, we have given rf (model name), x_sc, y, cv (as 5 folds). Our model is performing well. So, we are saving the model by pickle.dump().

Before evaluating the mode, we are transforming data x with transform() function.

```
rf = RandomForestRegressor()
rf.fit(X_train_scaled, y_train)
y_pred = rf.predict(X_test_scaled)

X_sc = scaler.transform(X)

cv = cross_val_score(rf, X_sc, y, cv=5)

np.mean(cv)
```

```
np.float64(0.834938893135182)
```

```
pickle.dump(rf, open('model.pkl', 'wb'))
pickle.dump(scaler, open('scaler.pkl', 'wb'))
pickle.dump(le, open('label_encoder.pkl', 'wb'))
```

Milestone 6: Model Deployment

6.1 Save the best model

From the below image, we can see the accuracy of the models and error of the models. Random forest regression has high accuracy and less error. We chose to proceed with the original Random Forest Regressor model (before tuning), as it offers better predictive accuracy and lower error in GDP per capita estimation.

```
*** Linear Regression Model ***
Score for Linear Regression model is 0.7826114237194834
RMSE for Linear Regression model is 4649.544639823302
----------------------------------------------------------------------------------------------------
*** Random Forest Regressor Model ***
Score for Random Forest Regressor Model is 0.9068044568724314
RMSE for Random Forest Regressor Model is 3044.3153283097695
----------------------------------------------------------------------------------------------------
*** SVR Model ***
Score for SVR Model is -0.26118547459767205
RMSE for SVR Model is 11199.059258238174
```

```
pickle.dump(rf, open('model.pkl', 'wb'))
pickle.dump(scaler, open('scaler.pkl', 'wb'))
pickle.dump(le, open('label_encoder.pkl', 'wb'))
```

6.2 Integrate with Web Framework

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI.

This section has the following tasks

- Building HTML Pages
- Building server-side script
- Run the web application

6.2.1: Building HTML Page

For the project we create the following HTML pages:

- **`index.html`**: Serves as the homepage with navigation, project introduction, and sections like About, Predict, and Feedback.
- **`predict.html`**: Contains the input form where users enter country parameters for GDP prediction.
- **`result.html`**: Displays the predicted GDP per capita based on the input provided.

6.2.2: Build Python code

- Import the libraries.

```
from flask import Flask, render_template, request, url_for, redirect
from flask_sqlalchemy import SQLAlchemy
import pickle
import pandas as pd
```

- Initialise flask app and the database.

```
app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = "sqlite:///feedback.db"
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

db = SQLAlchemy(app)

class Feedback(db.Model):
    feedback_id = db.Column(db.Integer, primary_key = True)
    name = db.Column(db.String(20), nullable=False)
    email = db.Column(db.String(50), nullable=False)
    message = db.Column(db.String(1000), nullable=False)

    def __repr__(self):
        return f"{self.name} - {self.email} - {self.message}"
```

- Load the saved model. Importing the flask module in the project is mandatory. An object of Flask class is our WSGI application. Flask constructor takes the name of the current module (name) as argument.

```
# Load model
model = pickle.load(open('model.pkl', 'rb'))
le_region = pickle.load(open('label_encoder.pkl', 'rb'))
scaler = pickle.load(open('scaler.pkl', 'rb'))
```

- Render HTML page

```
@app.route('/', methods=['GET', 'POST'])
def home():
    return render_template('index.html')
```

- Here we will be using a declared constructor to route to the HTML page which we have created earlier. In the above example, '/' URL is bound with the index.html function. Hence, when the home page of the web server is opened in the browser, the html page will be rendered. Whenever you enter the values from the html page the values can be retrieved using POST Method

- Retrieves the value from UI:

```
@app.route('/predict', methods=['GET', 'POST'])
def predict():
    if request.method == 'POST':
        try:
            data = request.form

            region = data['region']
            region_encoded = le_region.transform([region])[0]

            input_values = [
                region_encoded,
                int(data['Population']),
                int(data['Area']),
                float(data['Density']),
                float(data['Coastline']),
                float(data['NetMigration']),
                float(data['Literacy']),
                float(data['Phones']),
                float(data['Arable']),
                float(data['Crops']),
                float(data['Climate']),
                float(data['Birthrate']),
                float(data['Deathrate']),
                float(data['Agriculture']),
                float(data['Industry']),
                float(data['Service'])
            ]
```

- Here we are routing our app to predict() function. This function retrieves all the values from the HTML page using Post request. That is stored in an array. This array is passed to the model.predict() function. This function returns the prediction. And this prediction value will be rendered to the text that we have mentioned in the

result.html page earlier

- Main Function

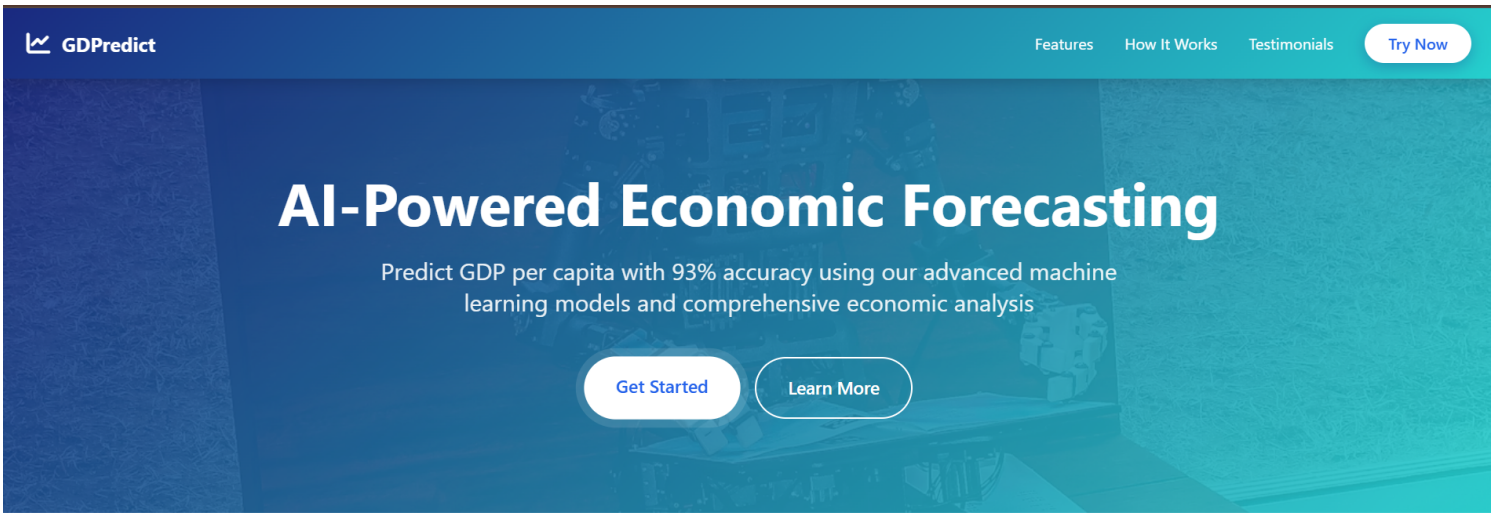
```python
if __name__ == '__main__':
    app.run(debug=False)
```

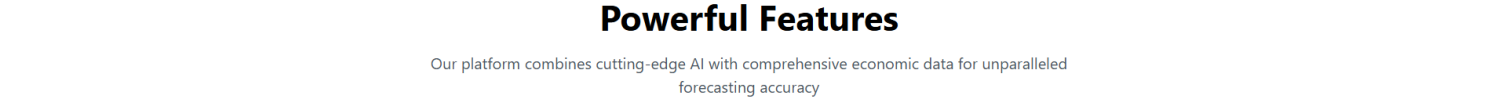
6.2.3 Run the web application

1. **Open terminal** (e.g., Command Prompt or Powershellt).
2. **Navigate to project folder**
3. **Activate virtual environment**: `env\Scripts\activate`.
4. **Run the app**: `python app.py`

```
D:\ujjwal\projects\GDP-per-capita-prediction\flask_app>python app.py
 * Serving Flask app 'app'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
127.0.0.1 - - [21/Jun/2025 11:11:01] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [21/Jun/2025 11:11:01] "GET /favicon.ico HTTP/1.1" 404 -
```

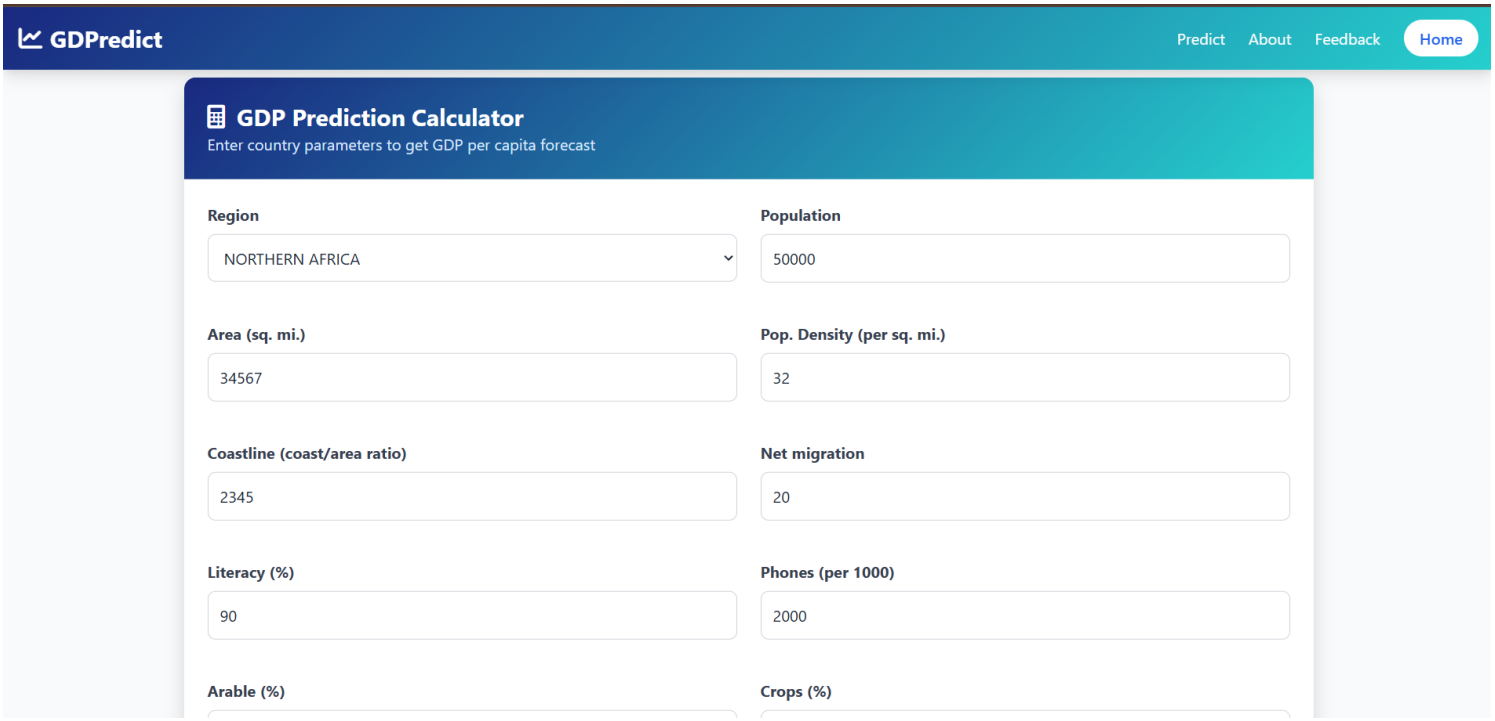
5. **Open browser** and go to `http://127.0.0.1:5000`.



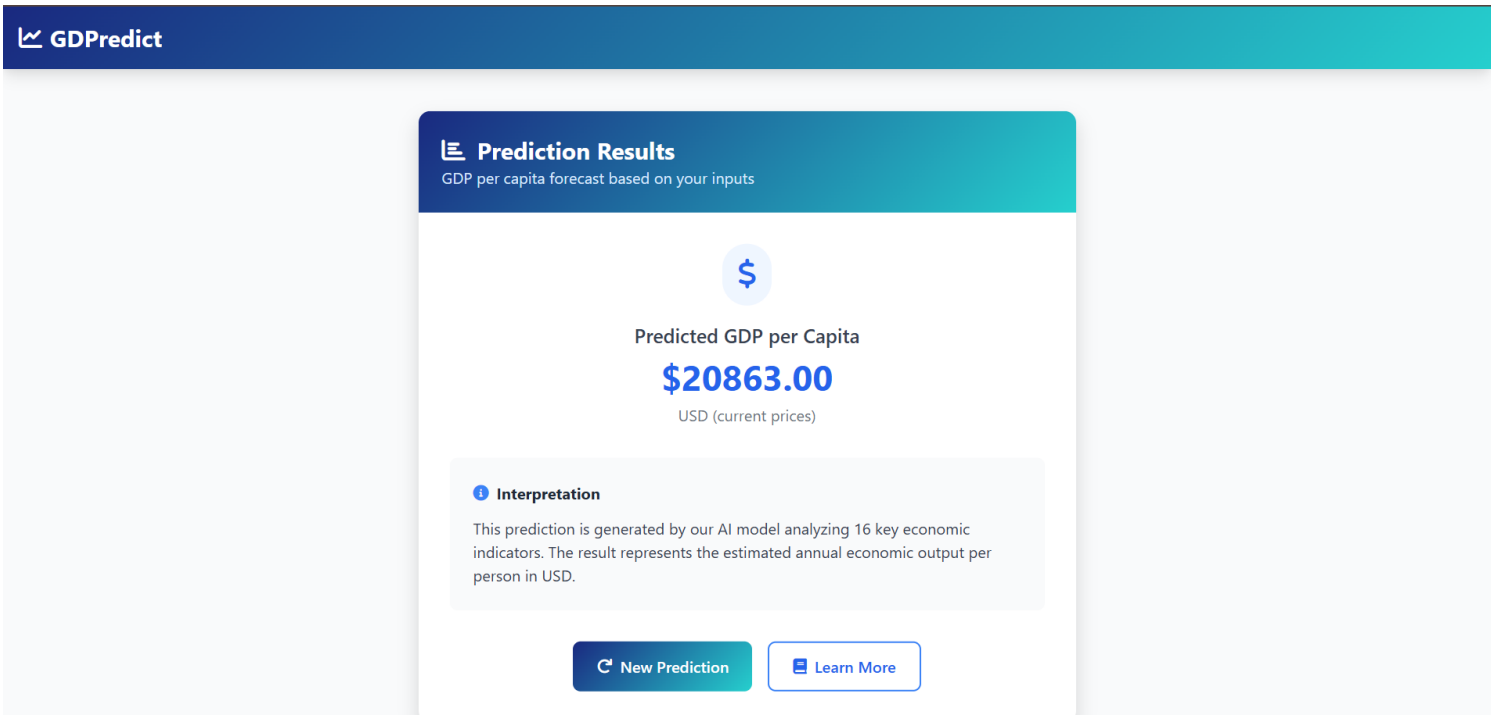


Home Page

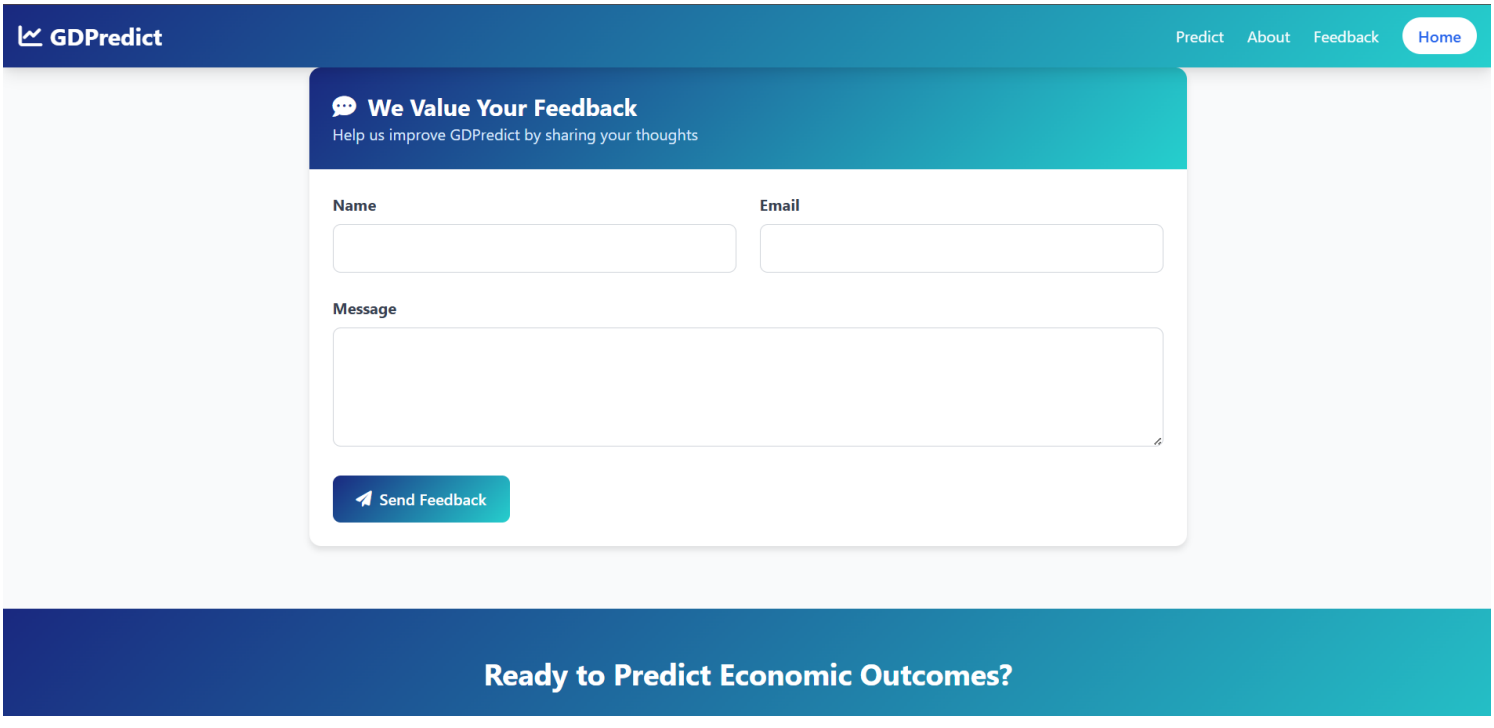
Understanding GDP per Capita



Input Form



Result



Feedback

Milestone 7: Project Demonstration & Documentation

7.1 Record explanation Video for project end to end solution

- GDPredict Demo Video
- GitHub
- GDPredict

7.2 Project Documentation-Step by step project development procedure

📊 Model Training

1. **Downloaded and Loaded the Dataset** into a pandas DataFrame using `read_csv()`.
2. **Performed Univariate Analysis** using `histplot()` and `boxplot()` to detect distributions and outliers.
3. **Carried Out Bivariate Analysis** with `countplot()` and `pairplot()` to understand relationships across features and regions.
4. **Visualized Correlations** using a heatmap to identify feature importance and redundancy.
5. **Summarized the Data** using descriptive statistics like `.info()`, `.describe()` to understand data types and missing values.
6. **Dropped Unnecessary Columns** like `Country`, which were identifiers and not useful for prediction.
7. **Handled Missing Values** using `dropna()` for critical values and `fillna()` with mean/median/mode for others depending on distribution and outliers.
8. **Encoded the Categorical Feature**`Region` using `LabelEncoder`.
9. **Detected and Handled Outliers** by choosing appropriate imputations based on outlier count per column.
10. **Split Data** into train and test sets using `train_test_split()`.
11. **Scaled the Data** using `StandardScaler()` to normalize feature ranges.
12. **Trained Models** including Linear Regression, Random Forest, and SVR, and compared performance.
13. **Selected the Best Model** based on R^2 Score and RMSE—Random Forest gave the best performance (≈91% accuracy).
14. **Saved the Model, Scaler, and Encoder** using `pickle.dump()` for deployment.

🌐 Application Building

15. **Created HTML Frontend** with Tailwind CSS containing a form to input 16 parameters and submit for prediction.
16. **Used Flask Framework** in Python to handle form requests and return predictions.
17. **Loaded the Pickled Model, Scaler, and Encoder** inside the Flask app.
18. **Built Prediction Route (`/`)** that accepts POST requests, processes input, scales it, and outputs prediction.
19. **Integrated a Feedback Form** and stored submissions using SQLite + SQLAlchemy ORM.
20. **Validated Form Inputs and Handled Errors** gracefully using Flask error handling.

🚀 Deployment

21. **Created a `requirements.txt`** and `Procfile` for deployment readiness.
22. **Pushed Project Code to GitHub** including all templates and model files.
23. **Deployed the App on Render.com**, configuring build and start commands (`gunicorn app:app`) and linking the GitHub repo.
24. **Tested the Web App** on the Render-deployed URL, confirming both prediction and feedback storage worked.
25. **Verified Prediction Accuracy** and ensured the database connection was functioning properly post-deployment.