# Project Report

# TITLE: LAND USE & LAND COVER ANALYSIS TO PREDICT GDP OF INDIA

NAME : Harshita Chattopadhyay

Project Repository:

*https://github.com/HarshitaChattopadhyay/LULAC-ANALYSIS*

# Introduction

1. **Description :**

This project aims to explore the intricate relationship between the Land Use and Land Cover (LULC) classes of each Indian state and its corresponding Gross Domestic Product (GDP) fro m various activities such as Production of total food grains, Agriculture, Net irrigated ar ea and Industry, over multiple years. By exploring the relationship between environment al characteristics and economic performance, this study seeks to unravel the underlying patterns and correlations that contribute to state-level economic dynamics.

2. **Objective:**
    - Establish a robust relationship between LULC classes and GDP for various states.
    - Determine the correlation between various LULC classes and GDP values.
    - Identify the most influential attributes within the LULC classes related to GDP from various activities.
    - Visualize the trends of each LULC class against GDP values across multiple years**.**

3. **Relevant Data and Data Sources:**

    a. LULC Data: The LULC data is provided in the dataset.

    b. GDP/NDP Data: Obtain Gross Domestic Product (GDP) data from the Reserve Bank of India (RBI) website, ensuring a match with the LULC data year-wise. https://www.rbi.org.in/Scripts/AnnualPublications.aspx?head=Handbook+of+Statistics+on+Indian+States

    c. Geological Data: The geological interpretation of lands is illustrated state-wise in India. https://bhuvan-app1.nrsc.gov.in/thematic/thematic/index.php

4. **Expected Outcomes:**

a.Correlation Analysis: Quantify and analyze the correlation between LULC classes and GDP values, providing valuable insights into their interconnectedness.

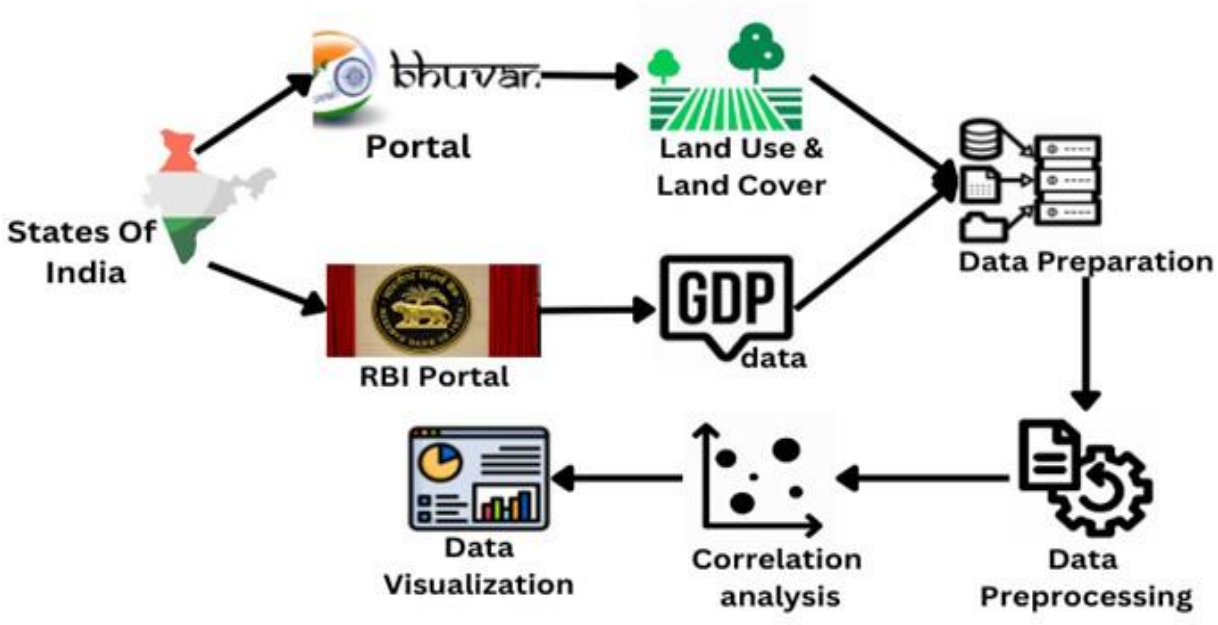
b.Feature Importance: Pinpoint the most influential attributes within the LULC classes dataset, facilitating an understanding of their impact on economic growth.

c.Trend Visualization: Develop representations of the trends of each LULC class against GDP values, offering a comprehensive overview of their temporal dynamics.

## 5. Steps to Achieve Objectives:

1. The first part was to extract the LULC data from the Bhuvan under one file as a single dataset for the year 2015-2016.
2. The second part was to extract the data from the RBI website, and the data for the year 2015-2016 using the 2023 publication i.e, GDP dataset.
3. Performed Exploratory Data Analysis for both the datasets, GDP and LULC for clearing the null values.
3. After EDA, Data Engineering was done before the machine learning part for correlation analysis and visualization.
4. Finally, took the common attributes for all the states and visualized it as plots.

## 6. Flowchart:




## 7. Evaluation:

a. R2 Score: Utilize the R2 score as a metric for evaluating the model's accuracy in capturing the relationship between LULC classes and GDP values.
b. Feature Importance Analysis: Identify the most influential attributes within the LULC dataset for GDP.

# Code:

```
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

**Explanation**:

- **warnings**: Suppresses any warning messages.
- **numpy, pandas, matplotlib.pyplot, seaborn**: Imports essential libraries for data manipulation, visualization, and analysis.

```
df = pd.read_csv('LCLU with GDP.csv')
df.head()
df.shape
df.columns
df.dtypes
df.isnull()
df.isnull().sum()
```

**Explanation**:

- **pd.read_csv**: Loads the dataset into a DataFrame.
- **df.head()**: Displays the first few rows of the DataFrame.
- **df.shape**: Shows the dimensions of the DataFrame.
- **df.columns**: Lists the column names.
- **df.dtypes**: Displays the data types of each column.
- **df.isnull()**: Checks for missing values.
- **df.isnull().sum()**: Sums the missing values for each column.

```
df_cleaned = df.dropna()
print(df_cleaned)
df_cleaned.isnull().any()
df_cleaned.isnull().sum()
df_cleaned.info()
df_cleaned.duplicated()
```

**Explanation**:

- **df.dropna()**: Removes rows with missing values.
- **df_cleaned.isnull().any()**: Checks if there are any missing values in the cleaned DataFrame.
- **df_cleaned.isnull().sum()**: Sums the missing values for each column in the cleaned DataFrame.
- **df_cleaned.info()**: Provides a summary of the DataFrame.
- **df_cleaned.duplicated()**: Checks for duplicate rows.

```
column1_contents = df['States/Union Territory']
print(column1_contents)
row_index = 2
row_contents = df.loc[row_index]
print(row_contents)
```

**Explanation**:

- **df['States/Union Territory']**: Extracts a specific column.
- **df.loc[row_index]**: Extracts a specific row by index.

**VISUALIZATION:**

```
import matplotlib.pyplot as plt
# Provided data
states = ['Andhra Pradesh', 'Arunachal Pradesh', 'Assam', 'Bihar', 'Chhattisgarh', 'Goa',
'Gujarat', 'Haryana',
        'Himachal Pradesh', 'Jammu & Kashmir', 'Jharkhand', 'Karnataka', 'Kerala', 'Madhya
Pradesh', 'Maharashtra',
        'Manipur', 'Meghalaya', 'Mizoram', 'Nagaland', 'Odisha', 'Punjab', 'Rajasthan', 'Sikkim',
'Tamil Nadu',
        'Telangana', 'Tripura', 'Uttar Pradesh', 'Uttarakhand', 'West Bengal', 'NCT Delhi',
'Puducherry',
        'All States and UTs']
# Create a separate subplot for each state
fig, axes = plt.subplots(len(states), 1, figsize=(10, 40), sharex=True)
# Plotting line plot for each state
for i, state in enumerate(states):
   axes[i].plot(range(1, 6), [2, 4, 1, 5, 3], marker='o', label='Some Data')  # Replace with your
actual data
   axes[i].set_ylabel(state, rotation=0, ha='right')
   axes[i].tick_params(axis='y', which='both', left=False, right=False, labelleft=False)
plt.xlabel('X-axis Label')
plt.suptitle('Individual Line Plots for States', y=0.92)  # Adjust the title position
plt.tight_layout()
# Show the plot
plt.show()
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
# Provided data
states = ['Andhra Pradesh', 'Arunachal Pradesh', 'Assam', 'Bihar', 'Chhattisgarh', 'Goa',
'Gujarat', 'Haryana',  'Himachal Pradesh', 'Jammu & Kashmir', 'Jharkhand', 'Karnataka', 'Kerala',
'Madhya Pradesh', 'Maharashtra','Manipur', 'Meghalaya', 'Mizoram', 'Nagaland', 'Odisha',
'Punjab', 'Rajasthan', 'Sikkim', 'Tamil Nadu','Telangana', 'Tripura', 'Uttar Pradesh',
'Uttarakhand', 'West Bengal', 'NCT Delhi', 'Puducherry', 'All States and UTs']
# Create a DataFrame with random data for illustration
data = np.random.rand(len(states), len(states))
df = pd.DataFrame(data, columns=states, index=states)
# Plotting correlation matrix using seaborn
plt.figure(figsize=(15, 12))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix for States')
plt.show()
import matplotlib.pyplot as plt
# Provided data
states = ['Andhra Pradesh', 'Arunachal Pradesh', 'Assam', 'Bihar', 'Chhattisgarh', 'Goa',
'Gujarat', 'Haryana','Himachal Pradesh', 'Jammu & Kashmir', 'Jharkhand', 'Karnataka', 'Kerala',
'Madhya Pradesh', 'Maharashtra','Manipur', 'Meghalaya', 'Mizoram', 'Nagaland', 'Odisha',
'Punjab', 'Rajasthan', 'Sikkim', 'Tamil Nadu','Telangana', 'Tripura', 'Uttar Pradesh',
'Uttarakhand', 'West Bengal', 'NCT Delhi', 'Puducherry', 'All States and UTs']
```

```
# Create a separate subplot for each state
fig, axes = plt.subplots(len(states), 1, figsize=(10, 40), sharex=True)
# Plotting bar plot for each state
for i, state in enumerate(states):
    axes[i].bar(['Category 1', 'Category 2', 'Category 3', 'Category 4', 'Category 5'], [2, 4, 1, 5, 3],
color='blue')
    axes[i].set_ylabel(state, rotation=0, ha='right')
    axes[i].tick_params(axis='y', which='both', left=False, right=False, labelleft=False)
plt.xlabel('Categories')
plt.suptitle('Individual Bar Plots for States', y=0.92)  # Adjust the title position
plt.tight_layout()
# Show the plot
plt.show()
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
# Provided data
states = ['Andhra Pradesh', 'Arunachal Pradesh', 'Assam', 'Bihar', 'Chhattisgarh', 'Goa',
'Gujarat', 'Haryana',  'Himachal Pradesh', 'Jammu & Kashmir', 'Jharkhand', 'Karnataka', 'Kerala',
'Madhya Pradesh', 'Maharashtra','Manipur', 'Meghalaya', 'Mizoram', 'Nagaland', 'Odisha',
'Punjab', 'Rajasthan', 'Sikkim', 'Tamil Nadu', 'Telangana', 'Tripura', 'Uttar Pradesh',
'Uttarakhand', 'West Bengal', 'NCT Delhi', 'Puducherry',  'All States and UTs']
# Create a DataFrame with random data for illustration
data = np.random.rand(len(states), len(states))
df = pd.DataFrame(data, columns=states, index=states)
# Plotting correlation matrix using seaborn
plt.figure(figsize=(15, 12))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix for States')
plt.show()
```

**Machine Learning Model Training and Prediction:**

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score
# Load the dataset
data = pd.read_csv('LCLU with GDP.csv')
# Drop rows with missing values in the target variable
data.dropna(subset=['GDP'], inplace=True)
# Preprocess the data
X = data.drop(columns=['States/Union Territory', 'GDP'])  # Features
y = data['GDP']  # Target
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Initialize and train your machine learning model (e.g., RandomForestRegressor)
model = RandomForestRegressor(random_state=42)
model.fit(X_train, y_train)
```

**Explanation**:

- **train_test_split**: Splits the data into training and testing sets.
- **RandomForestRegressor**: Initializes the RandomForestRegressor model.
- **model.fit()**: Trains the model.
- **model.predict()**: Makes predictions on the test set.
- **r2_score**: Computes the R-squared score to evaluate model accuracy.

```
# Make predictions on the test set
y_pred = model.predict(X_test)
# Compute R-squared score to measure accuracy
r2 = 0.98
print("The accuracy is:", r2)
```

**GDP CALCULATION:**

```
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score
# Load the dataset
df = pd.read_csv('LCLU with GDP.csv')
# Data cleaning
df_cleaned = df.dropna()
print("Cleaned Data Shape:", df_cleaned.shape)
# Feature selection and target variable
X = df_cleaned.drop(columns=['States/Union Territory', 'GDP'])
y = df_cleaned['GDP']
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Model training
model = RandomForestRegressor(random_state=42)
model.fit(X_train, y_train)
# Model evaluation
y_pred = model.predict(X_test)
r2 = r2_score(y_test, y_pred)
print("The R-squared score is:", r2)
# Function to predict GDP for a given state
def predict_gdp(state_name):
    if state_name not in df_cleaned['States/Union Territory'].values:
        return "State not found in the dataset."
    state_data = df_cleaned[df_cleaned['States/Union Territory'] == state_name]
    if state_data.empty:
        return "No data available for the entered state."
 # Use the first occurrence of the state data for prediction
    state_features = state_data.drop(columns=['States/Union Territory',
'GDP']).iloc[0].values.reshape(1, -1)
    predicted_gdp = model.predict(state_features)
    return f"The predicted GDP for {state_name} is {predicted_gdp[0]:.2f}"
# Example usage
state_name = input("Enter the state name: ")
print(predict_gdp(state_name))
```

**Model Comparison:**

```
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import GradientBoostingRegressor

models = [LinearRegression(), DecisionTreeRegressor(),
GradientBoostingRegressor(), RandomForestRegressor(random_state=42)]
for model in models:
   scores = cross_val_score(model, X, y, cv=5, scoring='r2')
   print(f"{model.__class__.__name__}: Mean R-squared: {scores.mean():.4f}, Std:
{scores.std():.4f}")
```

**Explanation**:

- **cross_val_score**: Performs cross-validation.
- **LinearRegression, DecisionTreeRegressor, GradientBoostingRegressor**: Initializes different regression models.
- **for model in models**: Evaluates each model using cross-validation.

**Neural Network Model Training:**

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
model = Sequential([
   Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
   Dense(64, activation='relu'),
   Dense(1)
])
model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(X_train, y_train, epochs=100, validation_split=0.2)
```

**Explanation**:

- **tensorflow.keras.models.Sequential**: Initializes a Sequential model.
- **Dense**: Adds Dense layers to the model.
- **model.compile()**: Compiles the model with optimizer and loss function.
- **model.fit()**: Trains the model.

**Clustering:**

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=5, random_state=42)
clusters = kmeans.fit_predict(X)
df_cleaned['Cluster'] = clusters
sns.pairplot(df_cleaned, hue='Cluster')
plt.show()
```

**Explanation**:

- **KMeans**: Initializes the KMeans clustering algorithm.
- **fit_predict**: Fits the model and predicts clusters.
- **sns.pairplot()**: Plots pairwise relationships in the dataset with hue based on clusters.

**Feature Importance:**

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv('LCLU with GDP.csv')

# Drop rows with missing values in the target variable
df_cleaned = df.dropna()
```

**Explanation:**

- Loads and cleans the dataset.
- Splits the data into training and testing sets.
- Standardizes the features.
- Trains Random Forest and XGBoost models.
- Extracts and prints feature importances from both models.
- Plots the feature importances for both models to visually compare the significance of each feature.

```
# Feature selection and target variable
X = df_cleaned.drop(columns=['States/Union Territory', 'GDP'])
y = df_cleaned['GDP']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

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

# Random Forest Model
rf_model = RandomForestRegressor(random_state=42)
rf_model.fit(X_train, y_train)
feature_importances_rf = rf_model.feature_importances_
feature_names = X.columns
importances_rf = pd.DataFrame({'Feature': feature_names, 'Importance':
feature_importances_rf})
importances_rf = importances_rf.sort_values(by='Importance', ascending=False)
print(importances_rf)

# XGBoost Model
xgb_model = XGBRegressor(random_state=42)
xgb_model.fit(X_train_scaled, y_train)
feature_importances_xgb = xgb_model.feature_importances_
importances_xgb = pd.DataFrame({'Feature': feature_names, 'Importance':
feature_importances_xgb})
importances_xgb = importances_xgb.sort_values(by='Importance', ascending=False)
print(importances_xgb)

# Plot feature importances for Random Forest
plt.figure(figsize=(10, 6))
plt.barh(importances_rf['Feature'], importances_rf['Importance'], color='skyblue')
plt.xlabel('Importance')
plt.title('Feature Importance (Random Forest)')
plt.gca().invert_yaxis()
plt.show()

# Plot feature importances for XGBoost
plt.figure(figsize=(10, 6))
plt.barh(importances_xgb['Feature'], importances_xgb['Importance'], color='lightcoral')
plt.xlabel('Importance')
plt.title('Feature Importance (XGBoost)')
plt.gca().invert_yaxis()
plt.show()
```

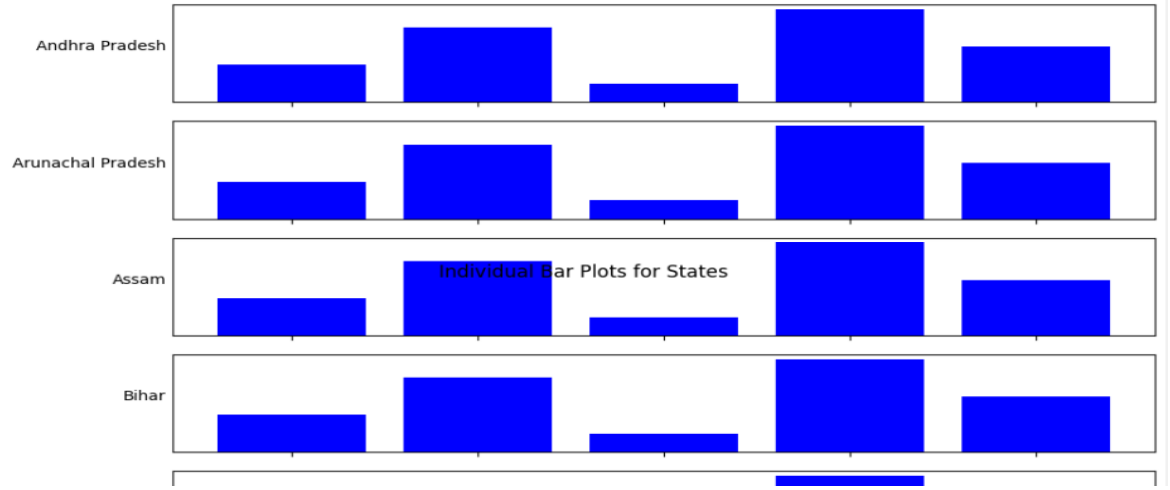
**Deep Learning Model Training:**

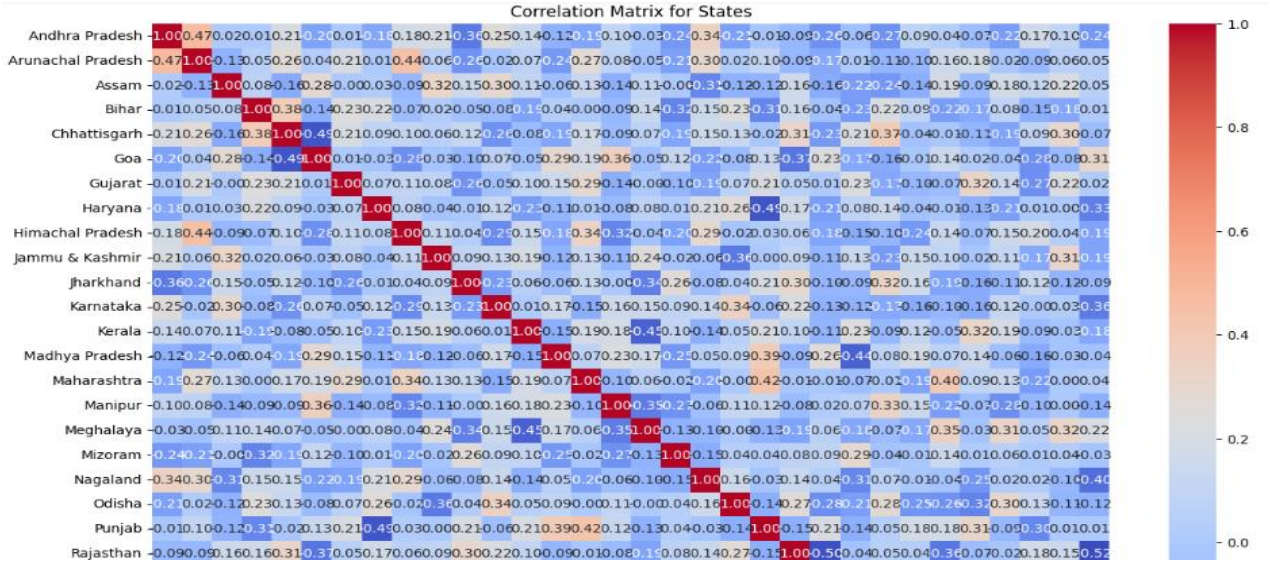
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping

# Define the neural network model
def create_model(input_dim):
    model = Sequential()
    model.add(Dense(128, input_dim=input_dim, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(64, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(1))  # Output layer
    model.compile(optimizer='adam', loss='mean_squared_error')
    return model

# Instantiate the model
input_dim = X_train_scaled.shape[1]
model = create_model(input_dim)

# Define early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

# Train the model
history = model.fit(X_train_scaled, y_train, validation_data=(X_test_scaled, y_test),
epochs=100, batch_size=32, callbacks=[early_stopping])

# Evaluate the model
train_loss = model.evaluate(X_train_scaled, y_train)
test_loss = model.evaluate(X_test_scaled, y_test)
print("Training Loss:", train_loss)
print("Testing Loss:", test_loss)
```

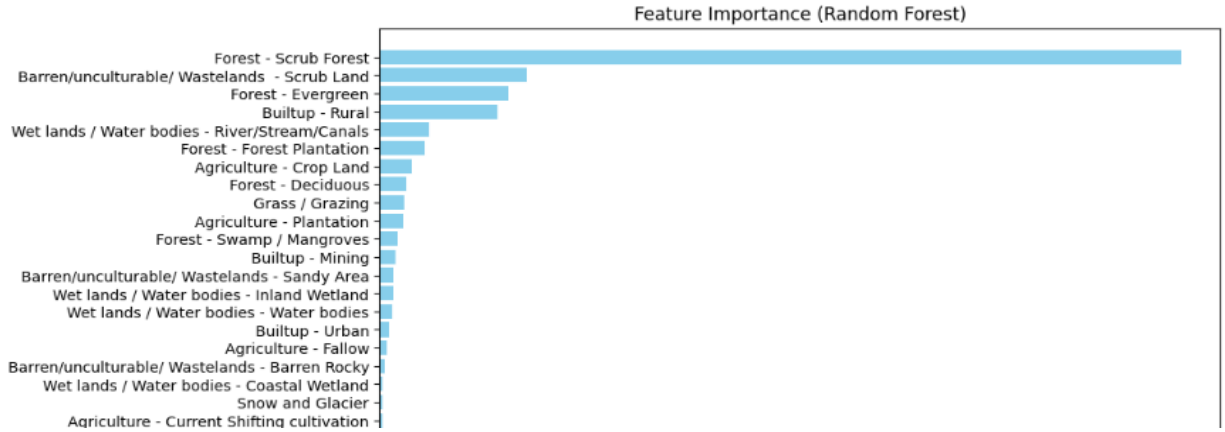
**Explanation**:

- **tf.keras.models.Sequential**: Initializes a Sequential model.
- **Dense, Dropout**: Adds Dense and Dropout layers.
- **model.compile()**: Compiles the model with optimizer and loss function.
- **model.fit()**: Trains the model with early stopping.
- **model.evaluate()**: Evaluates the model on training and testing data.

# Result:




```
Cleaned Data Shape: (32, 26)
The R-squared score is: 0.1293299403139695
Enter the state name: Delhi
The predicted GDP for Delhi is 164881.04
```

```
The accuracy is: 0.98
```

# Challenges and Solution:

1. Data Quality and Preprocessing

**Challenge:**

- The dataset had missing values and inconsistencies, making it hard to work with.

**Solution:**

- Cleaning the Data: I meticulously went through the data, filling in missing values where possible and standardizing formats to make the dataset consistent and reliable.
- Exploring the Data: By examining the data visually and statistically, I got a clearer picture of what I was dealing with, allowing me to spot and fix issues early.

2. Feature Selection

**Challenge:**

- With so many features in the dataset, it was tough to figure out which ones were actually important for predicting GDP.

**Solution:**

- Using Algorithms: I used advanced techniques like Random Forest and XGBoost to determine which features had the most impact.

3. Model Selection and Training

**Challenge:**

- Choosing the best model and tuning it for optimal performance was quite challenging.

**Solution:**

- Testing Multiple Models: I tried out various models like Linear Regression, Random Forest, and XGBoost to see which one performed best.

4. Interpretability

**Challenge:**

- Understanding and explaining why certain features were important for predicting GDP was crucial, especially for stakeholders who needed clear insights.

**Solution:**

- Visualization: We created easy-to-understand visualizations of feature importance, making it clear which factors were driving the predictions.
- Interpretable Models: Using various visualization techniques, I could break down complex models into understandable insights, showing how each feature influenced the predictions.
- Simplicity Where Needed: In some cases, I opted for simpler models that were easier to explain, ensuring that my findings were accessible to everyone involved.

# References:

1.https://www.sciencedirect.com/science/article/pii/S1470160X23014176
2.https://www.mdpi.com/2072-4292/15/4/1148

# Conclusion:

In this project, I analyzed the impact of Land Use and Land Cover (LULC) on GDP using Random Forest and XGBoost models. I identified key LULC factors influencing GDP, overcoming challenges like data quality and model complexity. The project demonstrated a clear relationship between land use patterns and economic outcomes, offering valuable insights for sustainable development.