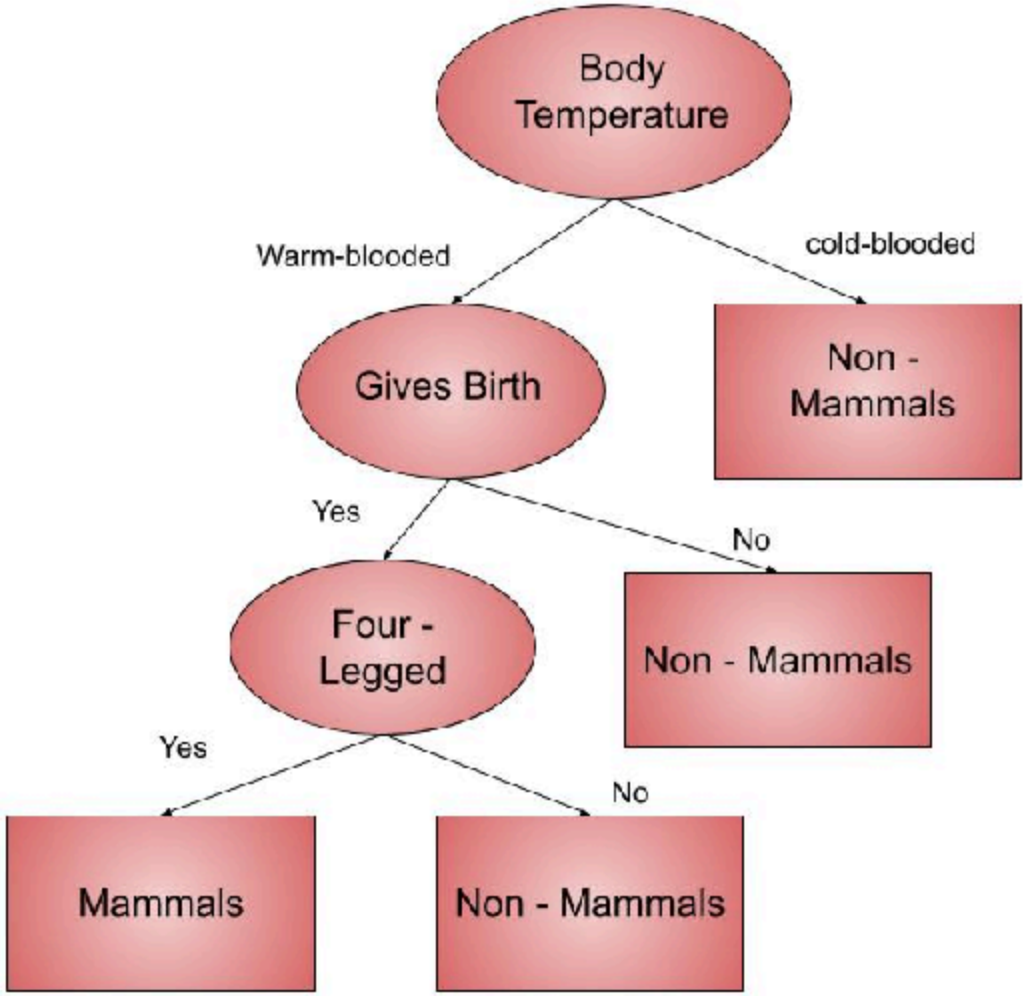
Supervised ML Assignment

Name: Rajesh Bisht

E-mail Id: rbisht.india@gmail.com

Git Link source code: https://github.com/RajeshBisht28/SuperVisedML_Assignment.git

Question: a) Describe the working of a Decision Tree algorithm. How does it decide on the best features to split the data? What are some advantages and disadvantages of using Decision Trees?



Decision Tree Overview

1. How a Decision Tree Works

- **Training**: The tree is built using a dataset by recursively splitting data based on feature values to maximize homogeneity in the subsets.
- **Prediction**: To make a prediction, the input instance is passed down the tree, starting from the root. At each node, a decision is made based on the feature test, and the instance follows the corresponding branch. This continues until a leaf node is reac

-##--

2. Selecting the Root Node

The root node is the first decision point in the tree. It is selected based on the feature that provides the **best split** of the data. To evaluate the quality of a split, criteria such as **information gain**, **Gini index**, or **variance reduction** (for regression) are u#sed.

Common Metrics for Selecting the Root Node:

1. **Information Gain (IG)**: Used in classification tasks, IG measures the reduction in entropy (uncertainty) after splitting.
   - **Entropy**: $H(S) = -\sum_{i=1}^{c} p_i \log_2(p_i)$
   - **Information Gain**: $IG = H(S) - \sum_{i=1}^{n} \frac{|S_i|}{|S|} H(S_i)$
2. **Gini Index**: Measures the impurity of a split (used in CART - Classification and Regression Trees).
   - **Gini**: $G = 1 - \sum_{i=1}^{c} p_i^2$
   - A lower Gini index indicates a better split.
3. **Variance Reduction**: Used in regression tasks to minimize the variance of the target variable in the subsets.
   - **Variance Reduction**: $\Delta = \text{Var}(S) - \left(\frac{|S_1|}{|S|} \text{Var}(S_1) + \frac{|S_2|}{|S|} \text{Var}(S_2) \right)$

---

3. How to Split a Tree Node

A split divides the dataset into two or more subsets. The goal is to choose a split that results in the most homogeneous subsets (purest possible).

Steps to Split a Node:

1. **Evaluate All Possible Splits**:

- For numerical features: Consider splitting at every unique value.
- For categorical features: Consider splitting based on category subsets.

2. **Calculate the Split Score**:
   - Use criteria like **information gain**, **Gini index**, or **variance reduction**.
3. **Select the Best Split**:
   - Choose the feature and split point that maximizes the chosen criterion.
4. **Create Sub-Nodes**:
   - The dataset is partitioned based on the best split, and sub-nodesar#e created for each partition.

Example of a Split (Classification)

Dataset:

Age	Income	Owns House	Default (Target)
25	Low	No	Yes
30	Medium	Yes	No
35	High	Yes	No
40	High	No	Yes

Possible Splits:

- **Split by Age**: Age < 30 or Age ≥ 30.
- **Split by Income**: {Low, Medium} or {High}.
- **Split by Owns House**: Yes or No.

Calculation of Best Split:

- Compute the information gain (or Gini index) for each feature and split point.
- Select the feature and value that gives the best score.

---

4. Stopping Criteria

The tree-building process stops when:

- The maximum tree depth is reached.
- All data points in a node belong to the same class (pure node).
- There are no more features to split on.
- The improvement in the split score is below a threshold.

In [ ]:

```

```

Question b) Use the Life Expectancy Prediction dataset from below Kaggle link and create an end to end project on Jupyter/Colab.

- i. Download the dataset from above link and load it into your Python environment.
-

ii Perform the EDA and do the visualizations.- ii . Check the distributions/skewness in the variables and do the transformations if required- . iv. Check/Treat the outliers and do the feature scaling if requir

- d.

v. Create a ML model to predict the life expectancy based on the specifications giv- en. vi. Check for overfitting and treat them accordi

- gly.

vii. Use all the Supervised ML algorithms (DT, RF, SVM, XGBoost etc.) and compare the performances to get the best del.

Project (Data Set) Overview:

This project aims to predict the yield of different crops by analyzing historical (from 1997 to 2020) 24-years agricultural data. By using Machine earning and statistical method that models the relationship between a dependent variable and one or more independent variables, we can forecast future crop production. This is particularly valuable for farmers, policymakers, and agricultural businesses who need reliable predictions to make informed decisions about crop planning, resource allocation, and market strategies.

==== Exploratory Data Analysis ====

About the Data Set:

The dataset consists of 22 columns with the following notable details:

Key Features:

- **Country**: The country name (categorical).
- **Year**: The year of the record (numerical).
- **Status**: Developing/Developed status of the country (categorical).
- **Life expectancy**: The target variable to predict, representing the average life expectancy in years (numerical).

Potential Features:

Demographic and Health-Related Features:

- **Adult Mortality**, **infant deaths**, **under-five deaths**: Mortality statistics.
- **HIV/AIDS**, **BMI**, **thinness 1-19 years**, **thinness 5-9 years**: Health-related factors.

Economic and Resource Indicators:

- **percentage expenditure**, **GDP**, **Total expenditure**, **Income composition of resources**, **Schooling**.

Immunization Rates:

- **Hepatitis B**, **Measles**, **Polio**, **Diphtheria**.

Other Features:

- **Alcohol**: Alcohol consumption (units unknown).
- **Population**: Population size.

Data Issues:

- Missing values are present in columns like **Alcohol**, **Hepatitis B**, **Total expenditure**, **GDP**, and **Population**.
- The **Country** and **Year** columns are identifiers rather than predictive features and might not be directly useful for model training.

Unimportant Features:

- **Country**: Likely redundant unless regional grouping adds value.
- **Year**: Useful only for time-series analysis; otherwise, it might be less significant as a feature.
- **Population**: May add noise, depending on how it's processed.

In [ ]:

```
```

Load data set

In [16]:

```
import pandas as pd
import numpy as np
# Load dataset
df_original = pd.read_csv(r'LifeExpectancyData.csv')
```

Remove unimportant features: Country, Year and Population.

In [17]:

```
df_data = df_original.drop(columns=['Country', 'Year', 'Population'])
df_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2938 entries, 0 to 2937
Data columns (total 19 columns):
 #   Column                           Non-Null Count  Dtype
---  ------                           --------------  -----
 0   Status                           2938 non-null   object
 1   Life expectancy                  2928 non-null   float64
 2   Adult Mortality                  2928 non-null   float64
 3   infant deaths                    2938 non-null   int64
 4   Alcohol                          2744 non-null   float64
 5   percentage expenditure           2938 non-null   float64
 6   Hepatitis B                      2385 non-null   float64
 7   Measles                          2938 non-null   int64
 8    BMI                             2904 non-null   float64
 9   under-five deaths                2938 non-null   int64
 10  Polio                            2919 non-null   float64
 11  Total expenditure                2712 non-null   float64
 12  Diphtheria                       2919 non-null   float64
 13   HIV/AIDS                        2938 non-null   float64
 14  GDP                              2490 non-null   float64
 15   thinness  1-19 years            2904 non-null   float64
 16   thinness 5-9 years              2904 non-null   float64
 17  Income composition of resources  2771 non-null   float64
 18  Schooling                        2775 non-null   float64
dtypes: float64(15), int64(3), object(1)
memory usage: 436.2+ KB
```

In [ ]:

```
```

Label encoding Status feature.

In [19]:

```python
import pandas as pd
from sklearn.preprocessing import LabelEncoder
### Copy of original dataset with new data set : df_encode
df_encode = df_data.copy()
# Label encoding
label_encoder = LabelEncoder()
# Custom mapping to avoid zeros
df_encode['Status'] = label_encoder.fit_transform(df_original['Status'])+1
total_nulls = df_encode.isnull().sum().sum()
print(f"Total null values are: {total_nulls}")
```

```
Total null values are: 1911
```

Replace null values columns with their mean.

In [62]:

```python
# Replace null values in numerical columns with their mean
data_filled = df_encode.copy()
numerical_columns = df_encode.select_dtypes(include=['float64', 'int64']).columns
# Fill missing values with column means
data_filled[numerical_columns] = data_filled[numerical_columns].apply(lambda col: col.fillna(col.mean()))
total_nulls = data_filled.isnull().sum().sum()
print(f"Total null values are: {total_nulls}")
```

```
Total null values are: 0
```

In [63]:

```python
print(f"{data_filled['Status'].value_counts()}")
print("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~")
correlation_matrix = data_filled.corr()
print(correlation_matrix['Status'])
```

```
Status
2    2426
1     512
Name: count, dtype: int64
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Status                             1.000000
Life expectancy                   -0.481962
Adult Mortality                    0.315171
infant deaths                      0.112252
Alcohol                           -0.579371
percentage expenditure            -0.454261
Hepatitis B                       -0.095642
Measles                            0.076955
 BMI                              -0.310873
under-five deaths                  0.115195
Polio                             -0.220098
Total expenditure                 -0.289985
Diphtheria                        -0.216763
 HIV/AIDS                          0.148590
GDP                               -0.445911
 thinness  1-19 years              0.367934
 thinness 5-9 years                0.366297
Income composition of resources   -0.457302
Schooling                         -0.491444
Name: Status, dtype: float64
```

Status = 2 has 2,426 rows, while Status = 1 has only 512 rows its indicating Imbalanced Class Distribution. This imbalance could lead to bias or reduced importance for this feature in predictive models. Variables like Life expectancy, Alcohol, GDP, Schooling, and Income composition of resources are negatively correlated with Status. However, these variables are likely to convey the same information that Status represents (developed vs. developing countries). **Decision:** Given the overlap in information between Status and other features, as well as the class imbalance, the Status column can be dropped if its removal does not significantly impact model performance.

In [34]:

```
df_vs = data_filled.drop(columns=['Status'])
```

In [64]:

```
df_vs.head()
```

Out[64]:

	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles	BMI	under-five deaths	Polio	Total expenditure	Diphtheria	HIV/
0	65.0	263.0	62	0.01	71.279624	65.0	1154	19.1	83	6.0	8.16	65.0	
1	59.9	271.0	64	0.01	73.523582	62.0	492	18.6	86	58.0	8.18	62.0	
2	59.9	268.0	66	0.01	73.219243	64.0	430	18.1	89	62.0	8.13	64.0	
3	59.5	272.0	69	0.01	78.184215	67.0	2787	17.6	93	67.0	8.52	67.0	
4	59.2	275.0	71	0.01	7.097109	68.0	3013	17.2	97	68.0	7.87	68.0	

Diverse Range of Features So Min-Max Scaler required : Feature scaling

In [65]:

```
##pip install scikit-learn
import pandas as pd
from sklearn.preprocessing import MinMaxScaler

# Sample DataFrame
#df_scaled = df_vs.cop()
#df = pd.DataFrame(df_scaled)

# Initialize MinMaxScaler
scaler = MinMaxScaler()

# Apply the scaler to the DataFrame
scaled_data = scaler.fit_transform(df_vs)

# Convert the scaled data back to a DataFrame
scaled_df = pd.DataFrame(scaled_data, columns=df_vs.columns).round(2)

##print(scaled_df)
```

In [70]:

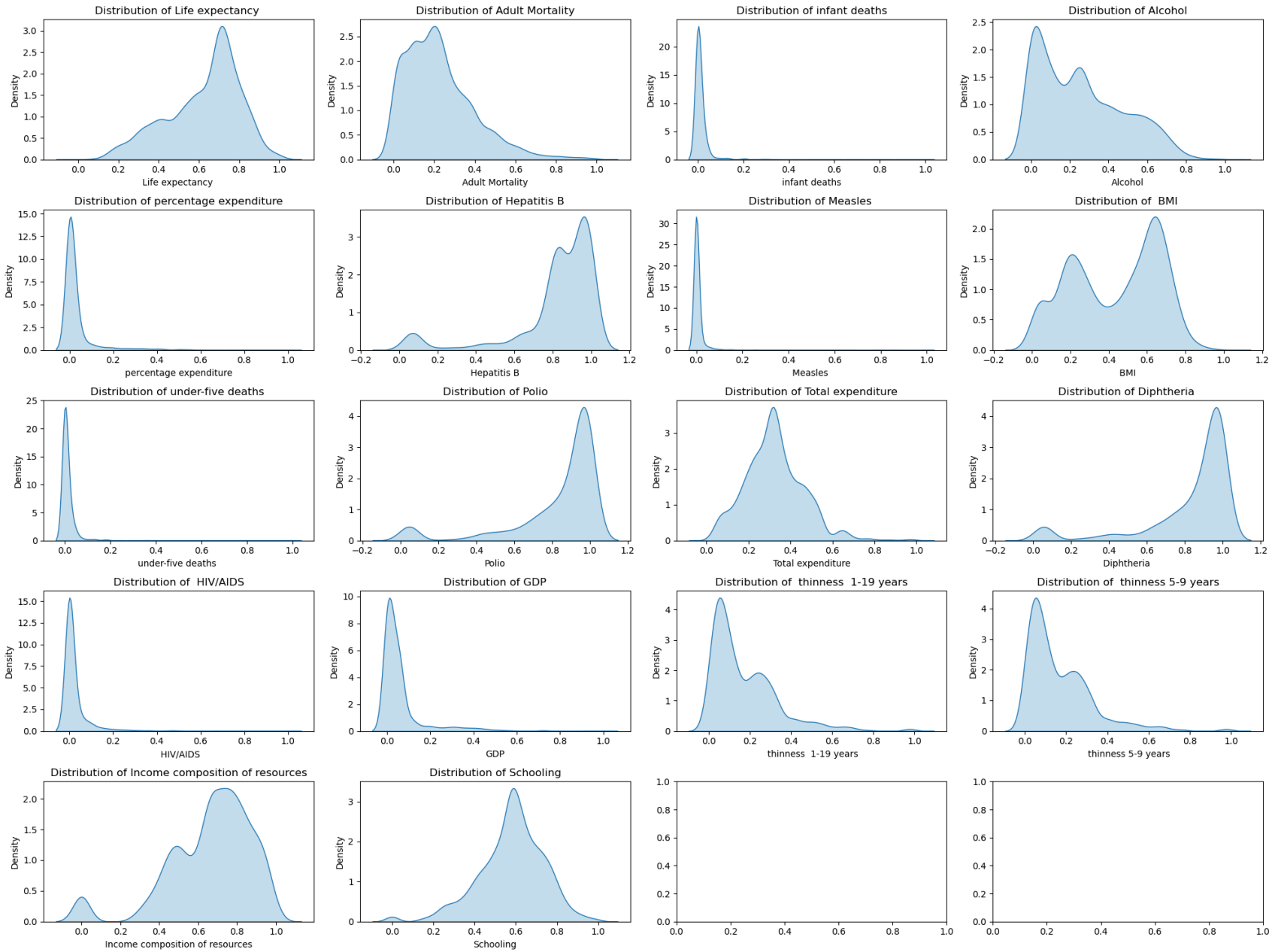
```
print(scaled_df.min())
```

```
Life expectancy                    0.0
Adult Mortality                    0.0
infant deaths                      0.0
Alcohol                            0.0
percentage expenditure             0.0
Hepatitis B                        0.0
Measles                            0.0
 BMI                               0.0
under-five deaths                  0.0
Polio                              0.0
Total expenditure                  0.0
Diphtheria                         0.0
 HIV/AIDS                          0.0
GDP                                0.0
 thinness  1-19 years              0.0
 thinness 5-9 years                0.0
Income composition of resources    0.0
Schooling                          0.0
dtype: float64
```

Plot the distribution curves for all features.

In [71]:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

features = scaled_df.columns
# Create a figure and axes
#fig, axes = plt.subplots(1, 4, figsize=(20, 5))
fig, axes = plt.subplots(5, 4, figsize=(20, 15))
# Plot each feature
for ax, feature in zip(axes.flatten(), features):
    sns.kdeplot(scaled_df[feature], shade=True, ax=ax)
    ax.set_title(f'Distribution of {feature}')
    ax.set_xlabel(feature)
    ax.set_ylabel('Density')

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



Skewness data : Need Robust Scaler

In [115…

```
from sklearn.preprocessing import RobustScaler

robust_scaler = RobustScaler()
robust_scaled_data = robust_scaler.fit_transform(scaled_df)
robust_scaled_df = pd.DataFrame(scaled_df, columns=scaled_df.columns)
robust_scaled_df.head()
```

Out[115…

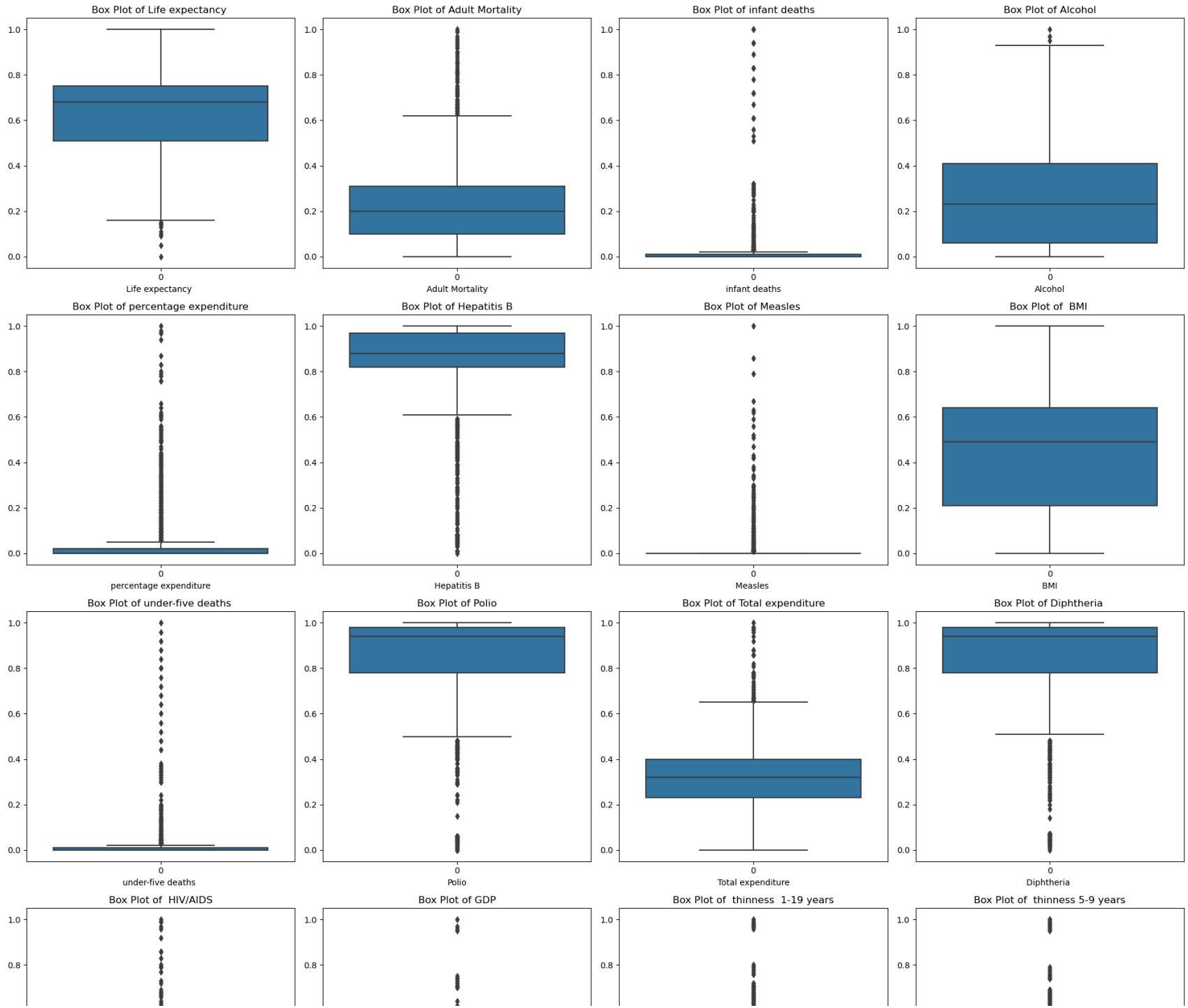
	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles	BMI	under-five deaths	Polio	Total expenditure	Diphtheria	HIV/
0	0.54	0.36	0.03	0.0	0.0	0.65	0.01	0.21	0.03	0.03	0.45	0.65	
1	0.45	0.37	0.04	0.0	0.0	0.62	0.00	0.20	0.03	0.57	0.45	0.62	
2	0.45	0.37	0.04	0.0	0.0	0.64	0.00	0.20	0.04	0.61	0.45	0.64	
3	0.44	0.38	0.04	0.0	0.0	0.67	0.01	0.19	0.04	0.67	0.47	0.67	
4	0.43	0.38	0.04	0.0	0.0	0.68	0.01	0.19	0.04	0.68	0.44	0.68	

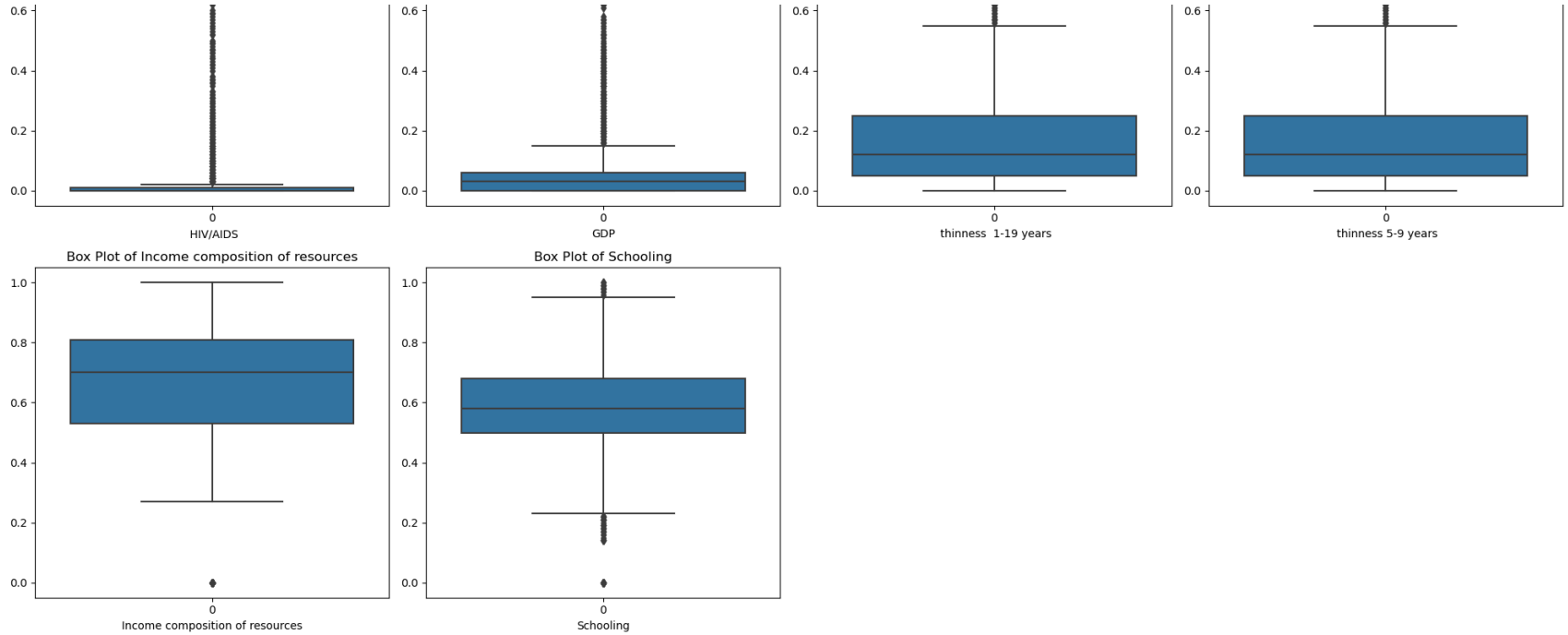
Outliers using by Box plots

In [116…

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

features = robust_scaled_df.columns
# Calculate the number of rows required for the grid
num_rows = len(features) // 4 + (len(features) % 4 > 0)
# Create a figure and axes
fig, axes = plt.subplots(num_rows, 4, figsize=(20, num_rows * 5))
# Flatten axes for easy iteration
axes = axes.flatten()
# Plot each feature
for ax, feature in zip(axes, features):
    sns.boxplot(data=robust_scaled_df[feature], ax=ax)
    ax.set_title(f'Box Plot of {feature}')
    ax.set_xlabel(feature)
```

```python
# Remove unused subplots
for i in range(len(features), len(axes)):
    fig.delaxes(axes[i])

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


Box Plot of Life expectancy
Life expectancy
Box Plot of Adult Mortality
Adult Mortality
Box Plot of infant deaths
infant deaths
Box Plot of Alcohol
Alcohol
Box Plot of percentage expenditure
percentage expenditure
Box Plot of Hepatitis B
Hepatitis B
Box Plot of Measles
Measles
Box Plot of BMI
BMI
Box Plot of under-five deaths
under-five deaths
Box Plot of Polio
Polio
Box Plot of Total expenditure
Total expenditure
Box Plot of Diphtheria
Diphtheria
Box Plot of HIV/AIDS
Box Plot of GDP
Box Plot of thinness 1-19 years
Box Plot of thinness 5-9 years



Caping outliers using IQR method.

In [117…

```
import pandas as pd
def cap_outliers(series):
    Q1 = series.quantile(0.25)
    Q3 = series.quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    return series.clip(lower_bound, upper_bound)
```

Truncate spaces before-after column name, and increament 1 value of 'Measles' feature.

In [118…

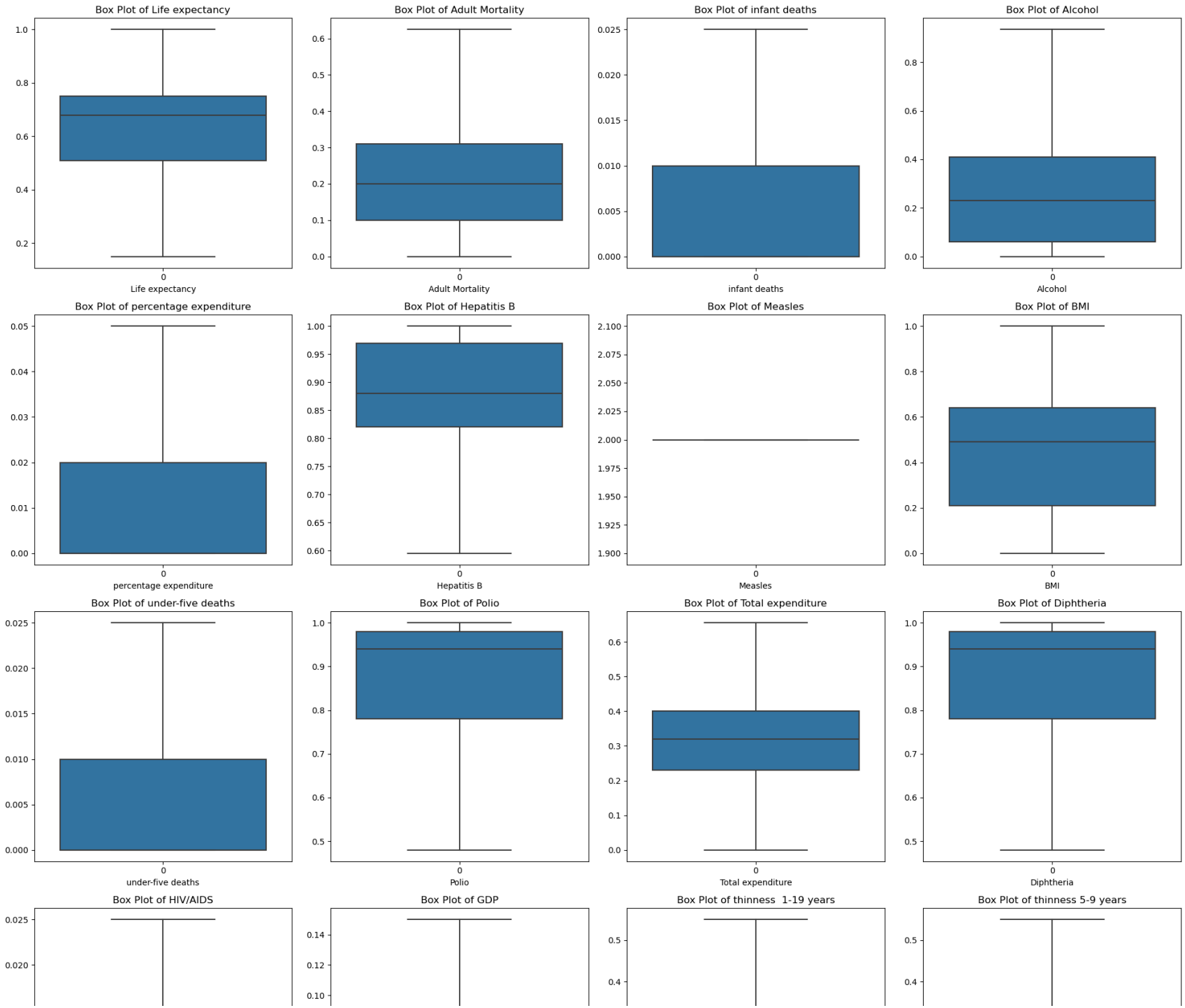
```
# Apply the function to each column
df_remove_outliers.columns = df_remove_outliers.columns.str.strip()
df_remove_outliers.columns
# Add 1 to each value in the 'Measles' column
df_remove_outliers['Measles'] = df_remove_outliers['Measles'] + 1
capped_df = df_remove_outliers.apply(cap_outliers)
```

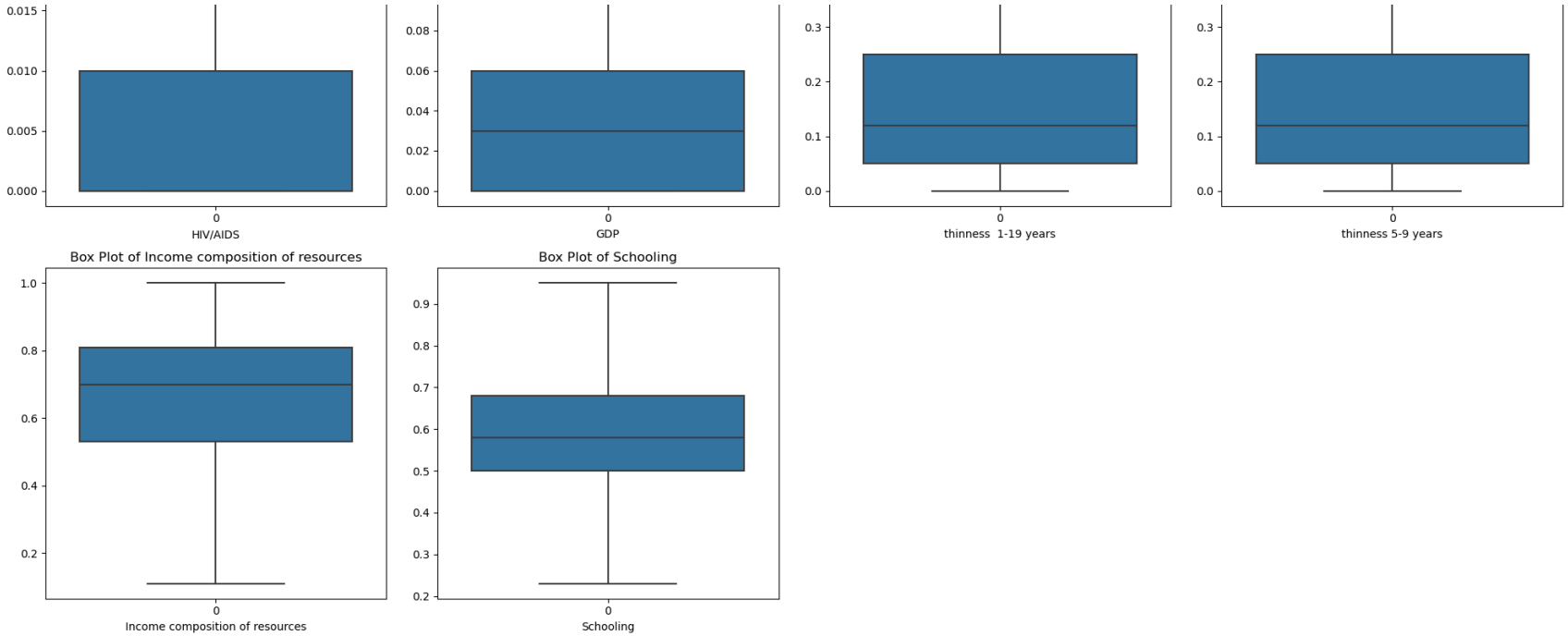
After Capping Outlier checks Box plot:

In [119…

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

features = capped_df.columns
# Calculate the number of rows required for the grid
num_rows = len(features) // 4 + (len(features) % 4 > 0)
# Create a figure and axes
fig, axes = plt.subplots(num_rows, 4, figsize=(20, num_rows * 5))
# Flatten axes for easy iteration
axes = axes.flatten()
# Plot each feature
for ax, feature in zip(axes, features):
    sns.boxplot(data=capped_df[feature], ax=ax)
    ax.set_title(f'Box Plot of {feature}')
    ax.set_xlabel(feature)

# Remove unused subplots
for i in range(len(features), len(axes)):
    fig.delaxes(axes[i])

print("\n === After Capping Outlier checks: Box plot === \n")
plt.tight_layout()
plt.show()
```

```
=== After Capping Outlier checks: Box plot ===
```


Box Plot of Life expectancy
Life expectancy
Box Plot of Adult Mortality
Adult Mortality
Box Plot of infant deaths
infant deaths
Box Plot of Alcohol
Alcohol
Box Plot of percentage expenditure
percentage expenditure
Box Plot of Hepatitis B
Hepatitis B
Box Plot of Measles
Measles
Box Plot of BMI
BMI
Box Plot of under-five deaths
under-five deaths
Box Plot of Polio
Polio
Box Plot of Total expenditure
Total expenditure
Box Plot of Diphtheria
Diphtheria
Box Plot of HIV/AIDS
Box Plot of GDP
Box Plot of thinness 1-19 years
Box Plot of thinness 5-9 years



Measles Feature have very thin line in Box-plot, Check for Zero Variance: Verify if all values in the "Measles" column are the same:

In [124…

```
total_nulls = capped_df.isnull().sum().sum()
print(f"Total null values are: {total_nulls}\n")
print(f"Measles feature have Null: {capped_df['Measles'].isna().sum()} \n")
print(f"Measles feature have all values same: {capped_df['Measles'].nunique()}")
```

```
Total null values are: 0

Measles feature have Null: 0

Measles feature have all values same: 1
```

'Measles' Column has zero variance. Correlation is not meaningful in this case, and may consider dropping the column.

In [125…

```
capped_df = capped_df.drop(columns=['Measles'])
capped_df.head()
```

Out[125…

	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	BMI	under-five deaths	Polio	Total expenditure	Diphtheria	HIV/AIDS	GD
0	0.54	0.36	0.025	0.0	0.0	0.65	0.21	0.025	0.48	0.45	0.65	0.0	0.0
1	0.45	0.37	0.025	0.0	0.0	0.62	0.20	0.025	0.57	0.45	0.62	0.0	0.0
2	0.45	0.37	0.025	0.0	0.0	0.64	0.20	0.025	0.61	0.45	0.64	0.0	0.0
3	0.44	0.38	0.025	0.0	0.0	0.67	0.19	0.025	0.67	0.47	0.67	0.0	0.0
4	0.43	0.38	0.025	0.0	0.0	0.68	0.19	0.025	0.68	0.44	0.68	0.0	0.0

In [ ]:

In [59]:

```
import pandas as pd

# Assuming df is your DataFrame
# Calculate skewness for all columns
skewness = scaled_df.skew().reset_index()
skewness.columns = ['Column', 'Skewness']

# Create a styled DataFrame
styled_skewness = skewness.style.set_table_styles(
    [
        {'selector': 'thead th', 'props': [('background-color', '#3E4149'), ('color', 'white'), ('text-align', 'cente
        {'selector': 'tbody td', 'props': [('border', '1px solid black'), ('text-align', 'center')]}
    ]
).set_properties(**{'text-align': 'center'})

# Display the styled DataFrame
styled_skewness
```

Out[59]:

	Column	Skewness
0	Life expectancy	-0.639745
1	Adult Mortality	1.176303
2	infant deaths	9.733551
3	Alcohol	0.606758
4	percentage expenditure	4.631135
5	Hepatitis B	-2.154028
6	Measles	9.406339
7	BMI	-0.220313
8	under-five deaths	9.464405
9	Polio	-2.105095
10	Total expenditure	0.651358
11	Diphtheria	-2.079076
12	HIV/AIDS	5.381845
13	GDP	3.474777
14	thinness 1-19 years	1.720626
15	thinness 5-9 years	1.783263
16	Income composition of resources	-1.174475
17	Schooling	-0.616065

Correlation Table among all features. (After removes outliers)

In [127…

```
#  Heatmap to show correlations
import pandas as pd
```

```python
correlation_matrix = capped_df.corr()
correlation_matrix_styled = correlation_matrix.style.background_gradient(cmap='coolwarm').format(precision=2)
correlation_matrix_styled
```

Out[127…

	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	BMI	under-five deaths	Polio	Total expenditure	Diphtheria	HI
Life expectancy	1.00	-0.69	-0.56	0.39	0.49	0.30	0.56	-0.60	0.57	0.21	0.57	
Adult Mortality	-0.69	1.00	0.38	-0.20	-0.32	-0.21	-0.39	0.41	-0.35	-0.13	-0.35	
infant deaths	-0.56	0.38	1.00	-0.32	-0.35	-0.31	-0.43	0.97	-0.42	-0.13	-0.41	
Alcohol	0.39	-0.20	-0.32	1.00	0.39	0.11	0.32	-0.32	0.26	0.30	0.26	
percentage expenditure	0.49	-0.32	-0.35	0.39	1.00	0.13	0.34	-0.35	0.27	0.18	0.27	
Hepatitis B	0.30	-0.21	-0.31	0.11	0.13	1.00	0.19	-0.32	0.59	0.06	0.63	
BMI	0.56	-0.39	-0.43	0.32	0.34	0.19	1.00	-0.45	0.34	0.23	0.34	
under-five deaths	-0.60	0.41	0.97	-0.32	-0.35	-0.32	-0.45	1.00	-0.45	-0.15	-0.43	
Polio	0.57	-0.35	-0.42	0.26	0.27	0.59	0.34	-0.45	1.00	0.16	0.84	
Total expenditure	0.21	-0.13	-0.13	0.30	0.18	0.06	0.23	-0.15	0.16	1.00	0.17	
Diphtheria	0.57	-0.35	-0.41	0.26	0.27	0.63	0.34	-0.43	0.84	0.17	1.00	
HIV/AIDS	-0.78	0.59	0.39	-0.20	-0.30	-0.27	-0.48	0.44	-0.47	-0.11	-0.47	
GDP	0.50	-0.32	-0.32	0.39	0.72	0.15	0.35	-0.33	0.30	0.14	0.28	
thinness 1-19 years	-0.51	0.34	0.42	-0.44	-0.34	-0.13	-0.56	0.43	-0.28	-0.28	-0.29	
thinness 5-9 years	-0.51	0.35	0.43	-0.43	-0.34	-0.13	-0.56	0.44	-0.28	-0.29	-0.28	
Income composition of resources	0.73	-0.48	-0.42	0.44	0.52	0.25	0.51	-0.44	0.46	0.17	0.47	

	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	BMI	under-five deaths	Polio	Total expenditure	Diphtheria	HIV
Schooling	0.74	-0.46	-0.50	0.52	0.54	0.27	0.53	-0.53	0.50	0.25	0.50	

In [ ]:

In [130…

```
def evaluate_model(y_true, y_pred):
    mse = mean_squared_error(y_true, y_pred)
    rmse = np.sqrt(mse)
    mae = mean_absolute_error(y_true, y_pred)
    r2 = r2_score(y_true, y_pred)
    return mse, rmse, mae, r2
```

In [ ]:

In [131…

```
from sklearn.preprocessing import RobustScaler, PowerTransformer, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

X = capped_df.drop(columns=['Life expectancy'])  # Features
y = capped_df['Life expectancy']  # Target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Train a Decision Tree Regressor
model = DecisionTreeRegressor(random_state=42)
model.fit(X_train, y_train)
```

Out[131…

```
▾ DecisionTreeRegressor
DecisionTreeRegressor(random_state=42)
```

In [ ]:

In [132…

```
# Predict on training and testing sets
y_train_pred = model.predict(X_train)
```

```
y_test_pred = model.predict(X_test)
train_metrics = evaluate_model(y_train, y_train_pred)
test_metrics = evaluate_model(y_test, y_test_pred)
```

In [133…

```
# Create a DataFrame for tabular display
compare_df = pd.DataFrame({
    'Metric': ['Mean Squared Error', 'Root Mean Squared Error', 'Mean Absolute Error', 'R-squared'],
    'Training': [round(val, 2) for val in train_metrics],
    'Testing': [round(val, 2) for val in test_metrics]
})
print("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~")
print("       Decision Tree Training and Testing Metrics ")
print("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n")

print(compare_df)
print("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~")
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
       Decision Tree Training and Testing Metrics
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

                   Metric  Training  Testing
0       Mean Squared Error       0.0     0.00
1  Root Mean Squared Error       0.0     0.05
2      Mean Absolute Error       0.0     0.03
3                R-squared       1.0     0.92
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Accuracy Training 100 and Testing 92 : may be overfitting So try to remove overfitting probllem using Random forest.

Apply Random Forest Regression.

In [135…

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# Distribute Feature and Target
X = capped_df.drop(columns=['Life expectancy'])
y = capped_df['Life expectancy']
```

```python
# Split into training and testing data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train the Random Forest model
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Predictions for random forest
y_train_pred = rf.predict(X_train)
y_test_pred = rf.predict(X_test)
```

In [136…

```python
train_rf_metrics = evaluate_model(y_train, y_train_pred)
test_rf_metrics = evaluate_model(y_test, y_test_pred)
```

In [137…

```python
# Create a DataFrame for tabular display
compare_rf = pd.DataFrame({
    'Metric': ['Mean Squared Error', 'Root Mean Squared Error', 'Mean Absolute Error', 'R-squared'],
    'Training': [round(val, 2) for val in train_rf_metrics],
    'Testing': [round(val, 2) for val in test_rf_metrics]
})
print("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~")
print("       Random Forest: Training and Testing Metrics ")
print("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n")

print(compare_rf)
print("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~")
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
       Random Forest: Training and Testing Metrics
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

                    Metric  Training  Testing
0       Mean Squared Error      0.00     0.00
1  Root Mean Squared Error      0.01     0.03
2      Mean Absolute Error      0.01     0.02
3                R-squared      0.99     0.97
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Train the SVM model : Support Vector Regression (SVR)

In [148…

```python
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# Distribute Feature and Target
X = capped_df.drop(columns=['Life expectancy'])
y = capped_df['Life expectancy']

# Split into training and testing data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [ ]:

In [149…

```python
# Initialize the SVR model
svr_model = SVR(kernel='linear')
# Train the model
svr_model.fit(X_train, y_train)
```

Out[149…

```
▾ SVR
SVR(kernel='linear')
```

In [ ]:

In [150…

```python
# Make predictions on the test set
y_pred = svr_model.predict(X_test)
# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f'Testing Mean Squared Error: {mse}')
print(f'R-squared: {r2}')
```

```
Mean Squared Error: 0.005509200569119356
R-squared: 0.8235385620048019
```

In [164…

```python
# Make predictions on the training set
y_train_pred = svr_model.predict(X_train)
```

```python
# Make predictions on the test set
y_test_pred = svr_model.predict(X_test)

# Evaluate the model on the training set
train_mse = mean_squared_error(y_train, y_train_pred)
train_r2 = r2_score(y_train, y_train_pred)

print('SVM: Training set evaluation:')
print(f'Mean Squared Error: {train_mse}')
print(f'R-squared: {train_r2}')

# Evaluate the model on the test set
test_mse = mean_squared_error(y_test, y_test_pred)
test_r2 = r2_score(y_test, y_test_pred)

print('SVM: Test set evaluation:')
print(f'Mean Squared Error: {test_mse}')
print(f'R-squared: {test_r2}')
```

```
SVM: Training set evaluation:
Mean Squared Error: 0.006389483904286667
R-squared: 0.8045792639863596
SVM: Test set evaluation:
Mean Squared Error: 0.005509200569119356
R-squared: 0.8235385620048019
```

In [158…

```python
### !pip install xgboost scikit-learn
```

```
Requirement already satisfied: xgboost in c:\users\hp\anaconda3\lib\site-packages (2.1.3)
Requirement already satisfied: scikit-learn in c:\users\hp\anaconda3\lib\site-packages (1.2.2)
Requirement already satisfied: numpy in c:\users\hp\anaconda3\lib\site-packages (from xgboost) (1.26.4)
Requirement already satisfied: scipy in c:\users\hp\anaconda3\lib\site-packages (from xgboost) (1.11.4)
Requirement already satisfied: joblib>=1.1.1 in c:\users\hp\anaconda3\lib\site-packages (from scikit-learn) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\hp\anaconda3\lib\site-packages (from scikit-learn)
(2.2.0)
```

Train XGBoost (Extreme Gradient Boosting)

In [184…

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import xgboost as xgb
```

```python
# Distribute Feature and Target
X = capped_df.drop(columns=['Life expectancy'])
y = capped_df['Life expectancy']

# Split into training and testing data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [ ]:

In [161…

```python
# Initialize the XGBoost regressor
xgb_reg = xgb.XGBRegressor(objective='reg:squarederror', n_estimators=100, learning_rate=0.1)
# Train the model
xgb_reg.fit(X_train, y_train)
```

Out[161…

XGBRegressor

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.1, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
```

In [ ]:

In [163…

```python
# Make predictions on the training set
y_train_pred = xgb_reg.predict(X_train)

# Make predictions on the test set
y_test_pred = xgb_reg.predict(X_test)

# Evaluate the model on the training set
train_mse = mean_squared_error(y_train, y_train_pred)
train_r2 = r2_score(y_train, y_train_pred)
```

```
print('XGBOOST: Training set evaluation:')
print(f'Mean Squared Error: {train_mse}')
print(f'R-squared: {train_r2}')

# Evaluate the model on the test set
test_mse = mean_squared_error(y_test, y_test_pred)
test_r2 = r2_score(y_test, y_test_pred)

print('XGBOOST: Test set evaluation:')
print(f'Mean Squared Error: {test_mse}')
print(f'R-squared: {test_r2}')
```

```
XGBOOST: Training set evaluation:
Mean Squared Error: 0.0003112305047796156
R-squared: 0.9904810943692769
XGBOOST: Test set evaluation:
Mean Squared Error: 0.0010522123447912657
R-squared: 0.9662973055512034
```

XGBOOST : with some parameter and Early stopping, max depth, learning rate etc.

In [217…

```
# Convert to DMatrix format
dtrain = xgb.DMatrix(X_train, label=y_train)
dval = xgb.DMatrix(X_test, label=y_test)

# Define parameters
params = {
    'objective': 'binary:logistic',
    'max_depth': 15,
    'learning_rate': 0.1,
    'eval_metric': ['logloss', 'error']
}

# Train with early stopping
model = xgb.train(
    params=params,
    dtrain=dtrain,
    num_boost_round=800,
    early_stopping_rounds=10,  # Stop if no improvement for 20 rounds
    evals=[(dtrain, 'train'), (dval, 'val')],  # Datasets to evaluate
    verbose_eval=10  # Print evaluation every 10 rounds
```

```
)
# Get the best iteration
print(f"Best iteration: {model.best_iteration}")
print(f"Best score: {model.best_score}")

# Use best model for predictions
predictions = model.predict(dval)
```

```
[0]	train-logloss:0.64942	train-error:0.37406	val-logloss:0.65232	val-error:0.37918
[10]	train-logloss:0.60171	train-error:0.31065	val-logloss:0.60704	val-error:0.31901
[20]	train-logloss:0.59282	train-error:0.30751	val-logloss:0.59942	val-error:0.31578
[30]	train-logloss:0.59055	train-error:0.30703	val-logloss:0.59792	val-error:0.31605
[40]	train-logloss:0.58970	train-error:0.30675	val-logloss:0.59757	val-error:0.31571
[45]	train-logloss:0.58946	train-error:0.30671	val-logloss:0.59751	val-error:0.31571
Best iteration: 36
Best score: 0.31527210690942753
```

In [218…

```
train_predictions = model.predict(dtrain)
test_predictions = model.predict(dval)

# Calculate R² for training set
r2_train = r2_score(y_train, train_predictions)

# Calculate R² for test set
r2_test = r2_score(y_test, test_predictions)

print("Model Performance:")
print(f"R² Score (Training): {r2_train:.4f}")
print(f"R² Score (Testing): {r2_test:.4f}")

# Calculate the difference to check for overfitting
r2_difference = r2_train - r2_test
print(f"\nDifference between Train and Test R²: {r2_difference:.4f}")

# Simple interpretation
if r2_difference > 0.1:
    print("Warning: Model might be overfitting (R² difference > 0.1)")
elif r2_train < 0.5:
    print("Warning: Model might be underfitting (Training R² < 0.5)")
else:
    print("Model R² scores suggest good fit")
```

```
Model Performance:
R² Score (Training): 0.9916
R² Score (Testing): 0.9664

Difference between Train and Test R²: 0.0251
Model R² scores suggest good fit
```

In [ ]:

:: Final Notes ::

Comparison: Decision Tree: Training : Testing :: 100 : 92 Random Forest: Training : Testing :: 99 : 97 SVM: Training : Testing :: 80 : 82 XGB : Training : Testing :: 99 : 96 XGB with Hypertuning: Training : Testing :: 99 : 97

Acceptance: Random Forest have 99% accuracy on the training set and 97% on the testing set are both quite high. This means the model is accurately predicting most of the data in both sets. A small gap (2%) between training and testing accuracy suggests that the model is generalizing well and is not overfitting. This is a good sign, as the model performs similarly on both the training and testing sets.

Acceptability: Yes, the model can be considered acceptable for deployment, as it achieves high accuracy on both training and testing data.

Interpretation of Accuracy: In terms of variance explained, it can approximate the testing performance as "97% accurate" and training as "99% accurate."

In [ ]: