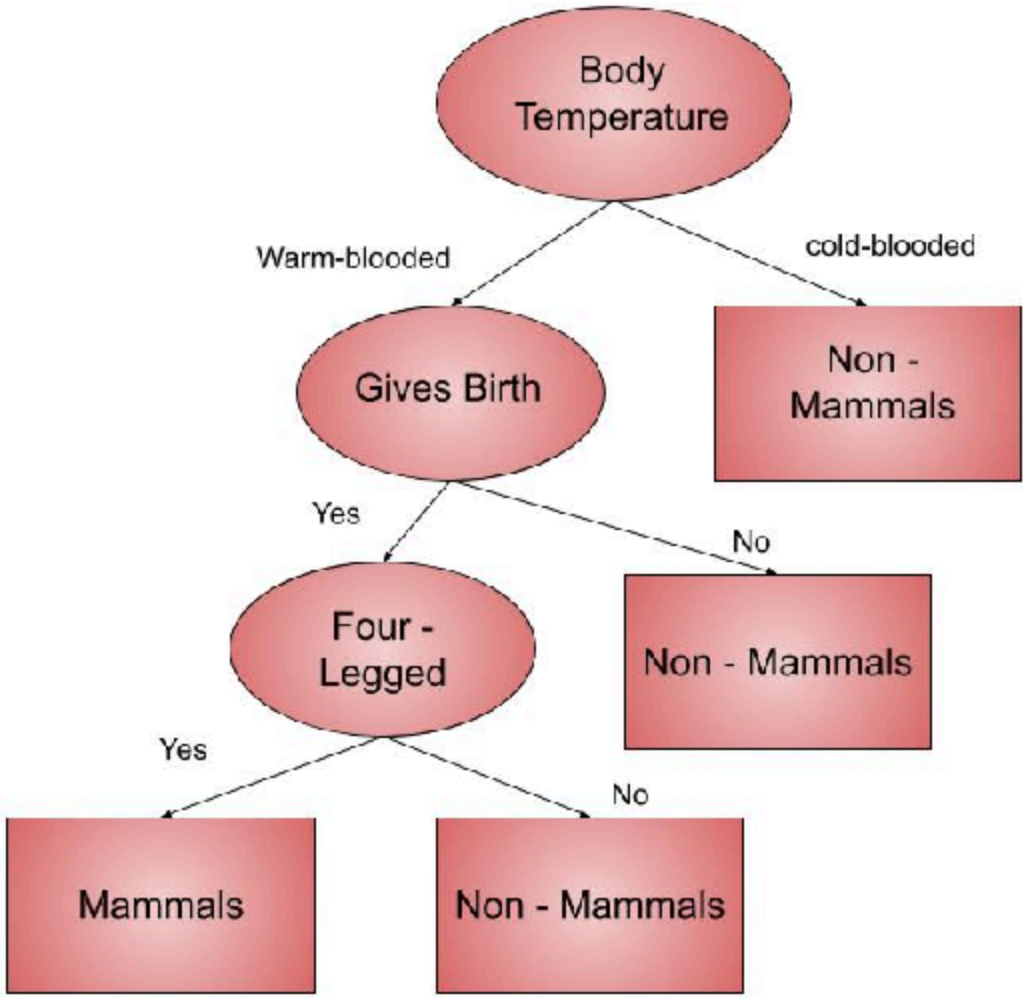
Supervised ML Assignment

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)
```

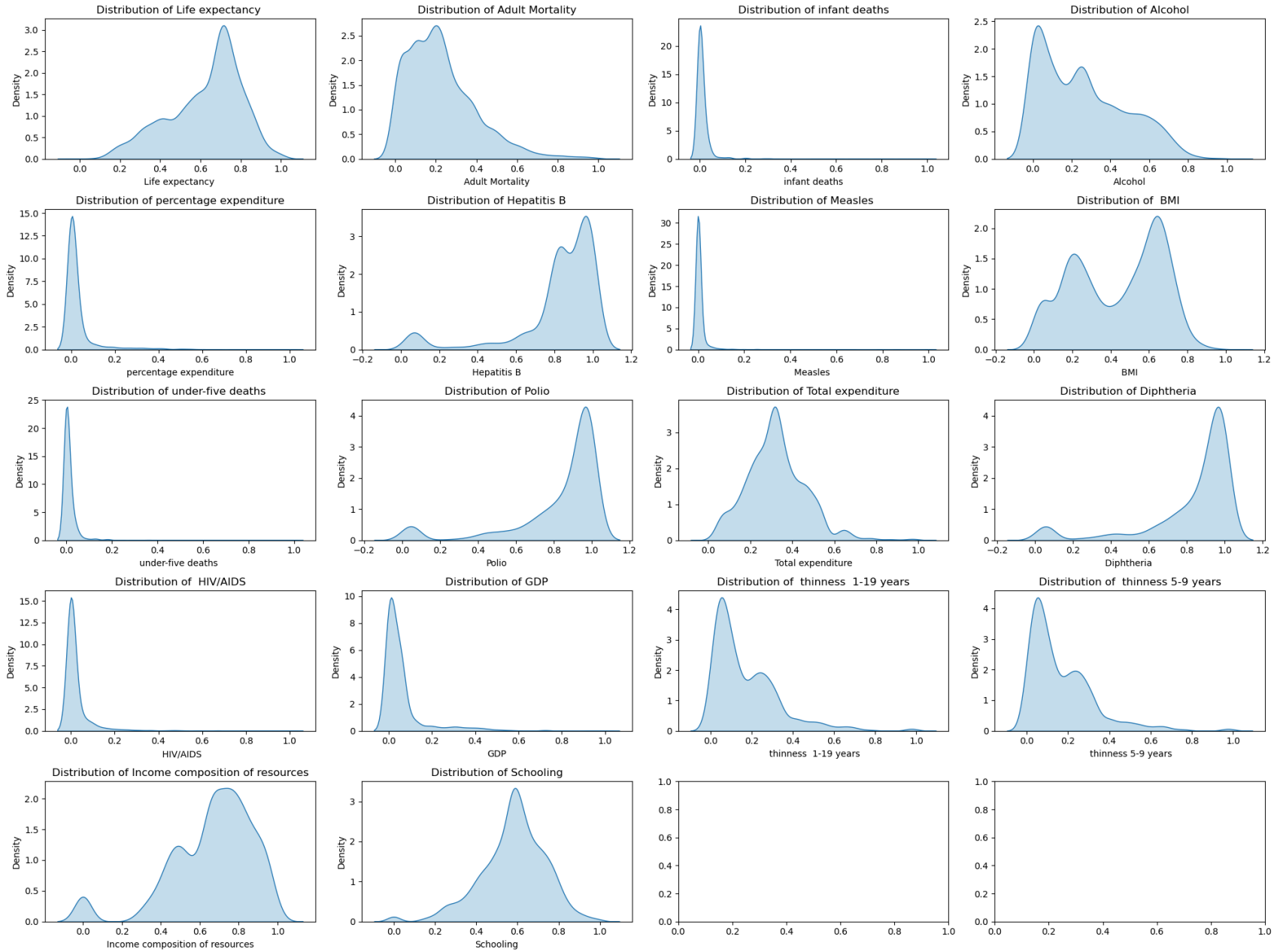
In [70]:
```
print(scaled_df.min())
```

```
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]:

```python
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…

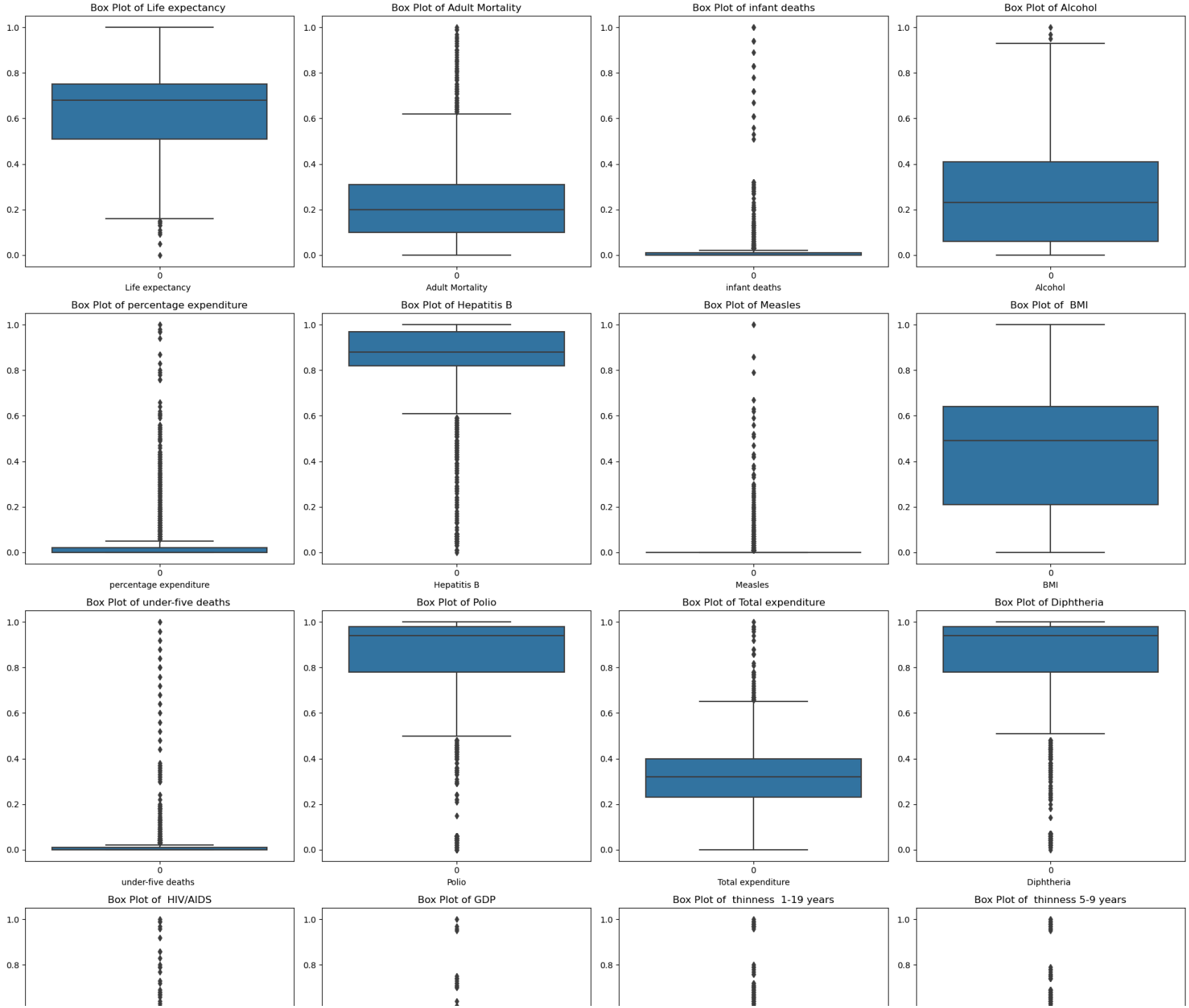
	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles	BMI	under-five deaths	Polio	Total expenditure	Diphtheria	HIV/
0	0.54	0.36	0.03	0.0	0.0	0.65	0.01	0.21	0.03	0.03	0.45	0.65	
1	0.45	0.37	0.04	0.0	0.0	0.62	0.00	0.20	0.03	0.57	0.45	0.62	
2	0.45	0.37	0.04	0.0	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	

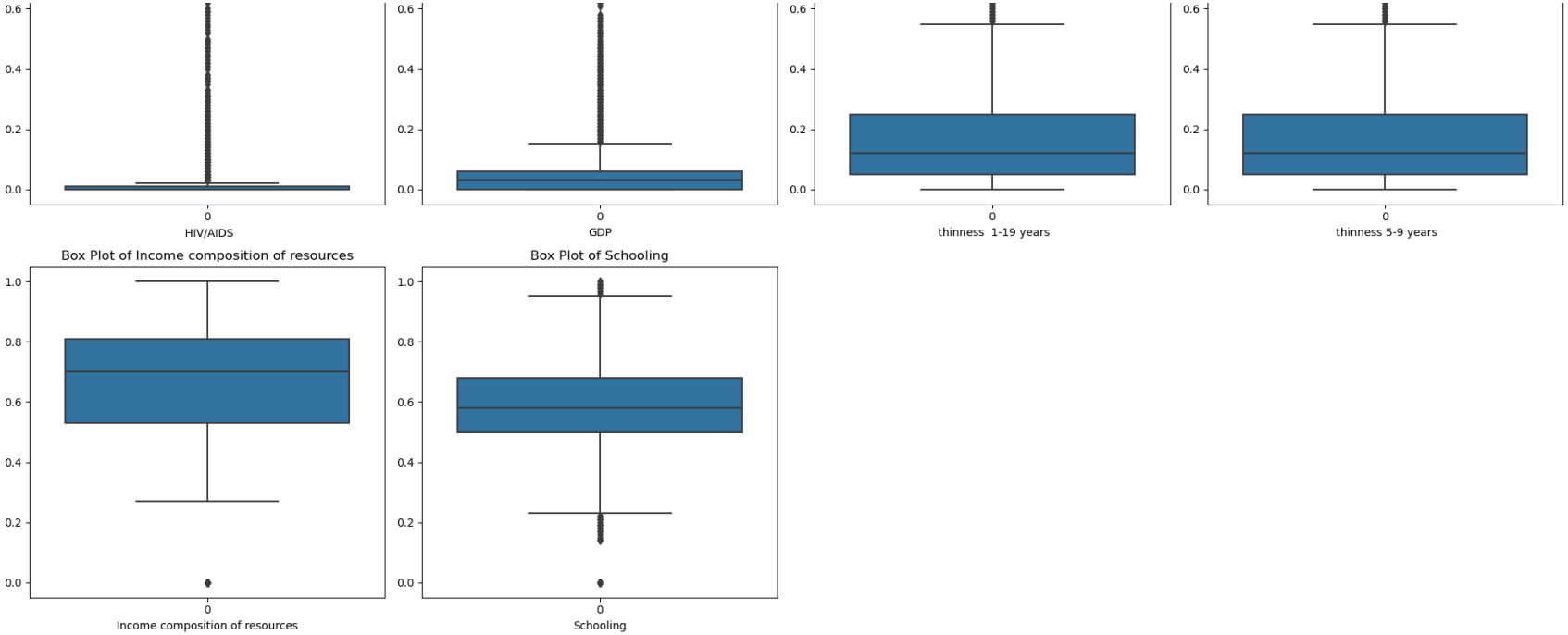
Outliers using by Box plots

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)
```

```
# 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…

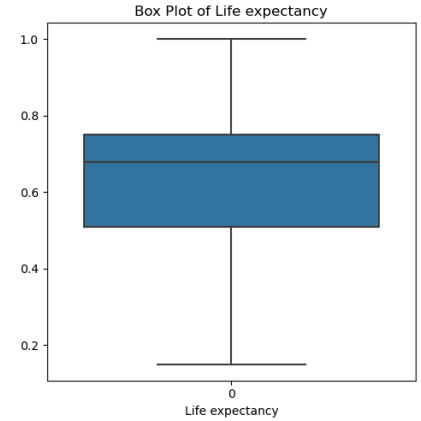
```
# Apply the function to each column
df_remove_outliers.columns = df_remove_outliers.columns.str.strip()
df_remove_outliers.columns
# Add 1 to each value in the 'Measles' column
df_remove_outliers['Measles'] = df_remove_outliers['Measles'] + 1
capped_df = df_remove_outliers.apply(cap_outliers)
```

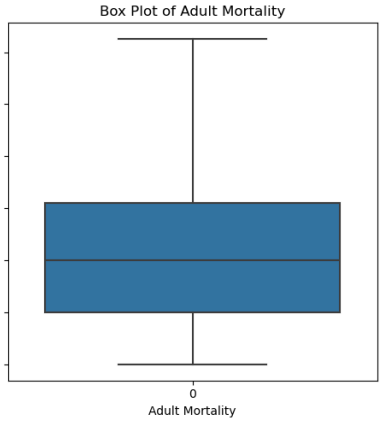
After Capping Outlier checks Box plot:

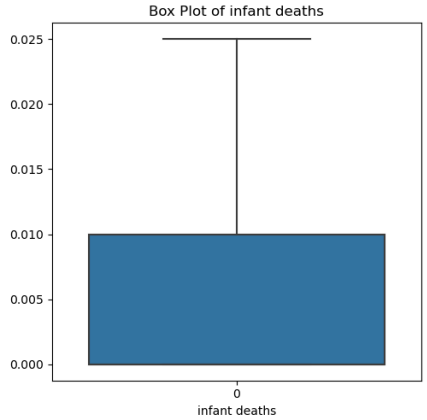
In [119…

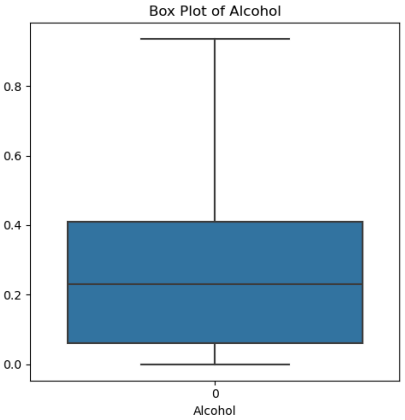
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

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()
```

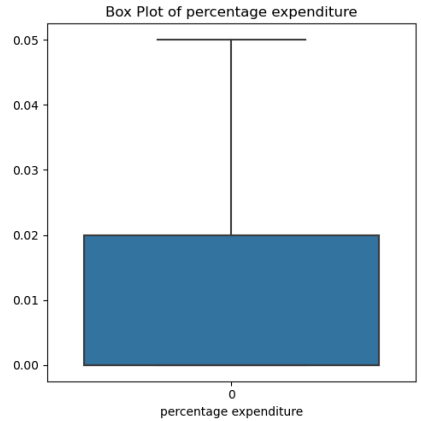
```
=== After Capping Outlier checks: Box plot ===
```

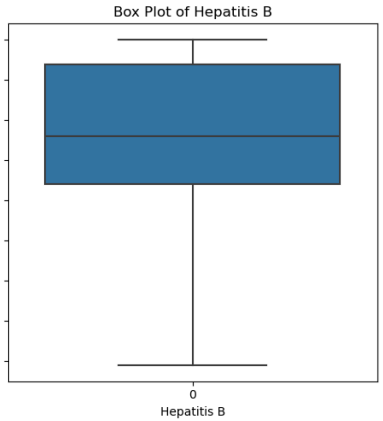

Box Plot of Life expectancy
1.0
0.8
0.6
0.4
0.2
0
Life expectancy

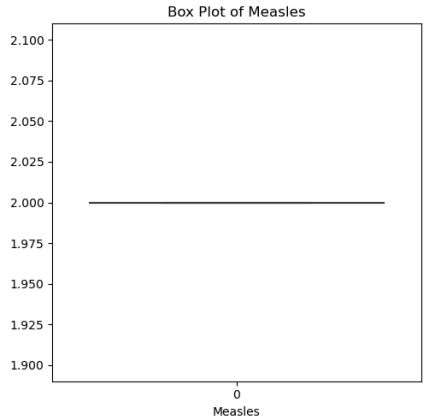

Box Plot of Adult Mortality
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
Adult Mortality

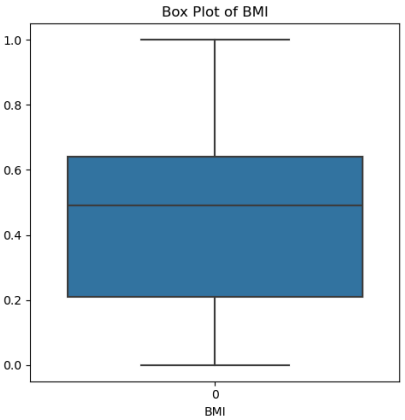

Box Plot of infant deaths
0.025
0.020
0.015
0.010
0.005
0.000
0
infant deaths

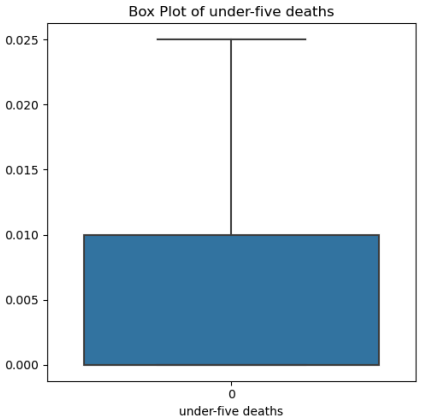

Box Plot of Alcohol
0.8
0.6
0.4
0.2
0.0
0
Alcohol

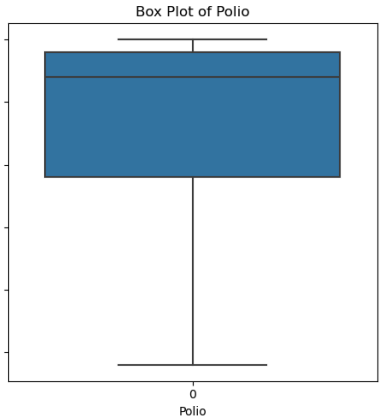

Box Plot of percentage expenditure
0.05
0.04
0.03
0.02
0.01
0.00
0
percentage expenditure

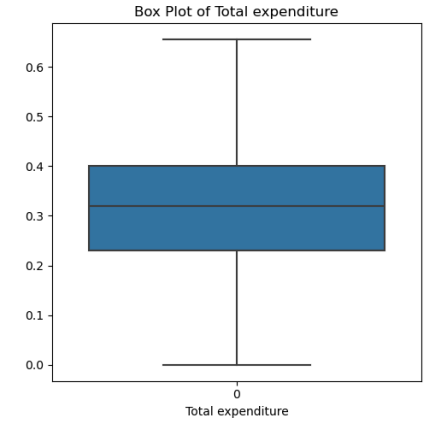

Box Plot of Hepatitis B
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0
Hepatitis B

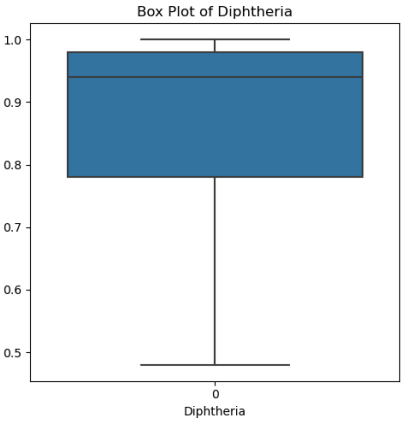

Box Plot of Measles
2.100
2.075
2.050
2.025
2.000
1.975
1.950
1.925
1.900
0
Measles

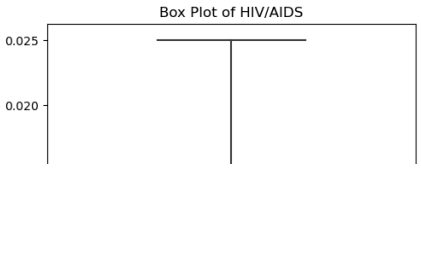

Box Plot of BMI
1.0
0.8
0.6
0.4
0.2
0.0
0
BMI

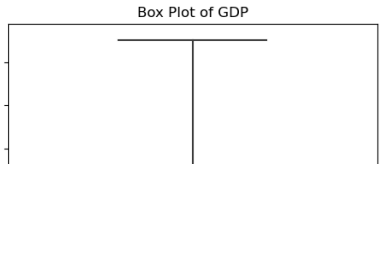

Box Plot of under-five deaths
0.025
0.020
0.015
0.010
0.005
0.000
0
under-five deaths

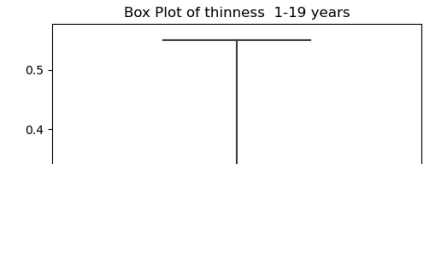

Box Plot of Polio
1.0
0.9
0.8
0.7
0.6
0.5
0
Polio

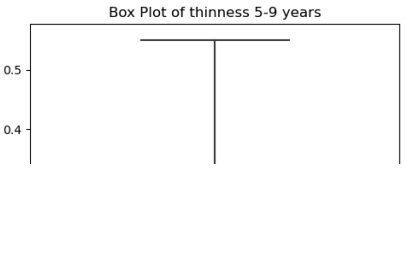

Box Plot of Total expenditure
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
Total expenditure

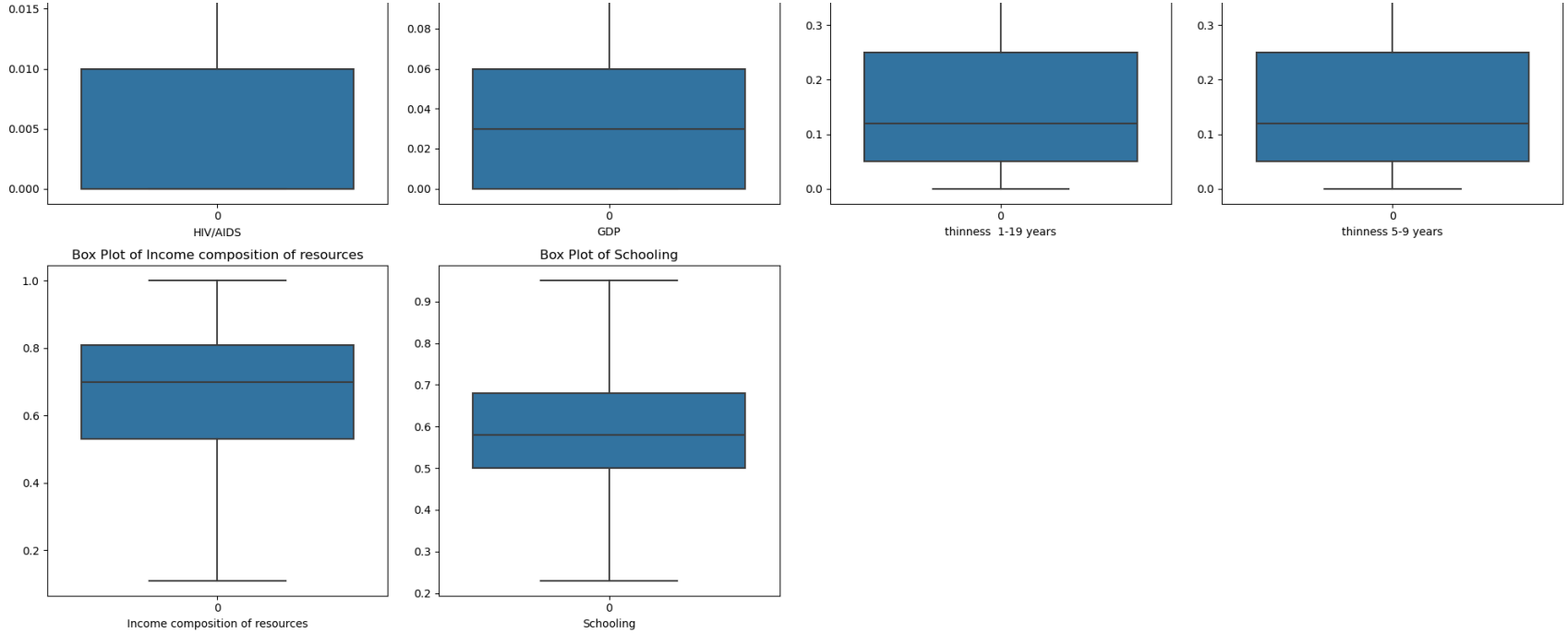

Box Plot of Diphtheria
1.0
0.9
0.8
0.7
0.6
0.5
0
Diphtheria


Box Plot of HIV/AIDS
0.025
0.020


Box Plot of GDP
0.14
0.12
0.10


Box Plot of thinness 1-19 years
0.5
0.4


Box Plot of thinness 5-9 years
0.5
0.4



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
```

```
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	HI
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(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']
```

```
# 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…

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

In [137…

```
# 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…

```
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…

```
# 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…

```
# 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…

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

```
# 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…

```
### !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…

```
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
```

```
# 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…
```
# 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…
```
# 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 [ ]:

Question C) Use the Income Classification dataset from below Kaggle link and create an end to end project on Jupyter/Colab.

https://www.kaggle.com/datasets/lodetomasi1995/income-classification/data

- ** i. Download the dataset from above link and load it into your Python environment. **
- ** ii. Perform the EDA and do the visualizations.
- ** iii. Check the distributions/skewness in the variables and do the transformations if required.
- ** iv. Check/Treat the outliers and do the feature scaling if required.
- ** v. Create a ML model to predict the life expectancy based on the specifications given.
- ** vi. Check for overfitting and treat them accordingly.
- ** vii. Use all the Supervised ML algorithms (DT, RF, SVM, XGBoost etc.) and compare the performances to get the best model.

Income Classification:(Data Set) Overview:

Column Names: age, workclass, fnlwgt, education, education-num, marital-status, occupation, relationship, race, sex, capital-gain, capital-loss, hours-per-week, native-country, income. Data Types: Numeric: age, fnlwgt, education-num, capital-gain, capital-loss, hours-per-week Categorical: workclass, education, marital-status, occupation, relationship, race, sex, native-country, income

In [ ]:

Income Classification

In [9]:
```
import pandas as pd
import numpy as np
# Load dataset
df_original = pd.read_csv(r'income_evaluation.csv')
```

In [10]:
```
### Truncate Leaft Right spaces of Feature name
df_original.columns = df_original.columns.str.strip()
```

In [11]:
```
df_original.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   age             32561 non-null  int64 
 1   workclass       32561 non-null  object
 2   fnlwgt          32561 non-null  int64 
 3   education       32561 non-null  object
 4   education-num   32561 non-null  int64 
 5   marital-status  32561 non-null  object
 6   occupation      32561 non-null  object
 7   relationship    32561 non-null  object
 8   race            32561 non-null  object
 9   sex             32561 non-null  object
 10  capital-gain    32561 non-null  int64 
 11  capital-loss    32561 non-null  int64 
 12  hours-per-week  32561 non-null  int64 
 13  native-country  32561 non-null  object
 14  income          32561 non-null  object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

In [12]:
```
total_nulls = df_original.isnull().sum().sum()
print(f"Total null values are: {total_nulls}")
```

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

Check Class imbalancing of Target Feature : income

In [14]:
```
import pandas as pd
import matplotlib.pyplot as plt
target_column = 'income'
class_counts = df_original[target_column].value_counts()
# Display the class distribution
print(class_counts)
# Plot the class distribution
class_counts.plot(kind='bar', color=['skyblue', 'salmon'])
plt.title('Class Distribution')
plt.xlabel('Class')
```

```
plt.ylabel('Frequency')
plt.show()
```

```
income
 <=50K    24720
 >50K      7841
Name: count, dtype: int64
```

Minority class : Upsampling => there is class imbalancing

In [15]:
```
import pandas as pd
from sklearn.utils import resample
```

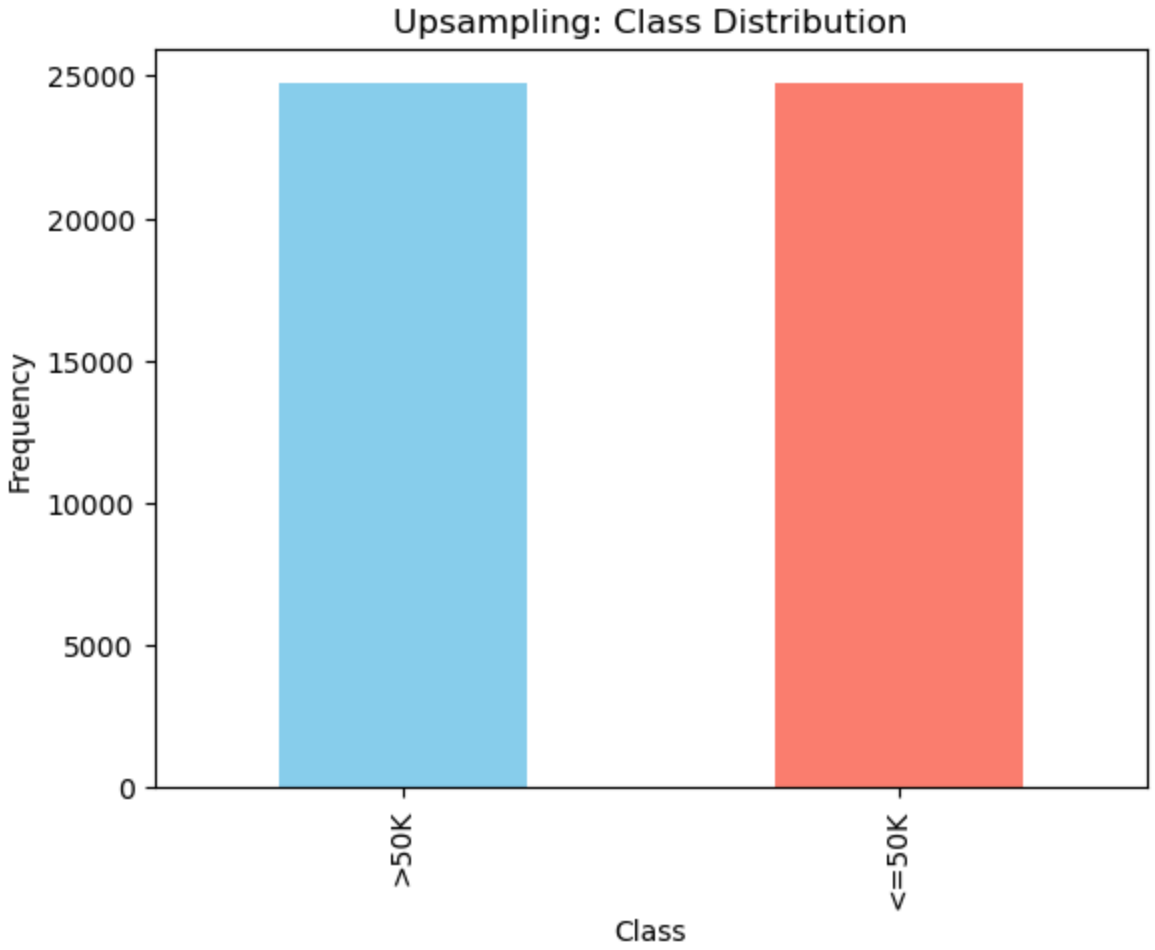
```
df_original['income'] = df_original['income'].str.strip()
df = df_original.copy()
target_feature = 'income'
# Separate majority and minority classes
df_majority = df[df['income'] == '<=50K']
df_minority = df[df[target_feature] == '>50K']
# Upsample minority class
df_minority_upsampled = resample(df_minority,
                                 replace=True,     # sample with replacement
                                 n_samples=len(df_majority),    # to match majority class
                                 random_state=42) # reproducible results

# Combine majority class with upsampled minority class
df_final = pd.concat([df_majority, df_minority_upsampled])

# Shuffle the final dataset
df_final = df_final.sample(frac=1, random_state=42).reset_index(drop=True)
# Display the final class distribution
print(df_final[target_feature].value_counts())
```

```
income
>50K     24720
<=50K    24720
Name: count, dtype: int64
```

In [ ]:

In [16]:
```
import pandas as pd
import matplotlib.pyplot as plt
target_column = 'income'
class_counts = df_final[target_column].value_counts()
# Display the class distribution
print(class_counts)
# Plot the class distribution
class_counts.plot(kind='bar', color=['skyblue', 'salmon'])
plt.title('Upsampling: Class Distribution')
plt.xlabel('Class')
plt.ylabel('Frequency')
plt.show()
```

```
income
>50K     24720
<=50K    24720
Name: count, dtype: int64
```



Label encoding all required features.

In [17]:
```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
df_encode = df_final.copy()
# Trim spaces from all string columns
df_encode = df_encode.apply(lambda x: x.str.strip() if x.dtype == "object" else x)
```

```
# Identify categorical columns
categorical_columns = df_encode.select_dtypes(include=['object']).columns
# Initialize the label encoder
label_encoders = {col: LabelEncoder() for col in categorical_columns}
# Apply label encoding to each categorical column
for col in categorical_columns:
    df_encode[col] = label_encoders[col].fit_transform(df_encode[col])
# Display the first few rows of the encoded dataset
print(df_encode.head(5))
```

```
   age  workclass  fnlwgt  education  education-num  marital-status  \
0   35          4  953588         11              9               2
1   36          4  150042          9             13               0
2   36          6   37778         12             14               2
3   40          5  102226         11              9               2
4   61          4  162397          9             13               2

   occupation  relationship  race  sex  capital-gain  capital-loss  \
0           4             5     4    0             0             0
1          10             3     4    0             0             0
2           5             0     4    1             0             0
3           4             0     4    1             0             0
4          12             0     4    1             0             0

   hours-per-week  native-country  income
0              40              39       1
1              40              39       0
2              70              39       0
3              40              39       0
4              40              39       1
```

Check Features min values : How much far values each others

In [18]:
```
print("===== Minimum values ======= ")
print(df_encode.min())
print("===== Maximum values ======= ")
print(df_encode.max())
```

```
===== Minimum values =======
age                   17
workclass              0
fnlwgt             12285
education              0
education-num          1
marital-status         0
occupation             0
relationship           0
race                   0
sex                    0
capital-gain           0
capital-loss           0
hours-per-week         1
native-country         0
income                 0
dtype: int64
===== Maximum values =======
age                    90
workclass               8
fnlwgt            1484705
education              15
education-num          16
marital-status          6
occupation             14
relationship            5
race                    4
sex                     1
capital-gain        99999
capital-loss         4356
hours-per-week         99
native-country         41
income                  1
dtype: int64
```

Apply Standardization (Z-Score Scaling) : Make all features equivalent in terms of scale.

In [19]:
```python
from sklearn.preprocessing import StandardScaler

# Initialize the StandardScaler
scaler = StandardScaler()
```

```
# Fit and transform the data
scaled_data = scaler.fit_transform(df_encode)
# Convert the scaled data back to a DataFrame
df_scaled = pd.DataFrame(scaled_data, columns=df.columns)
# Display the first few rows of the scaled dataset
print("==== Display the first few rows of the scaled dataset  ======")
print("================================================================================\n")
print(df_scaled.head(5))
print("================================================================================")
```

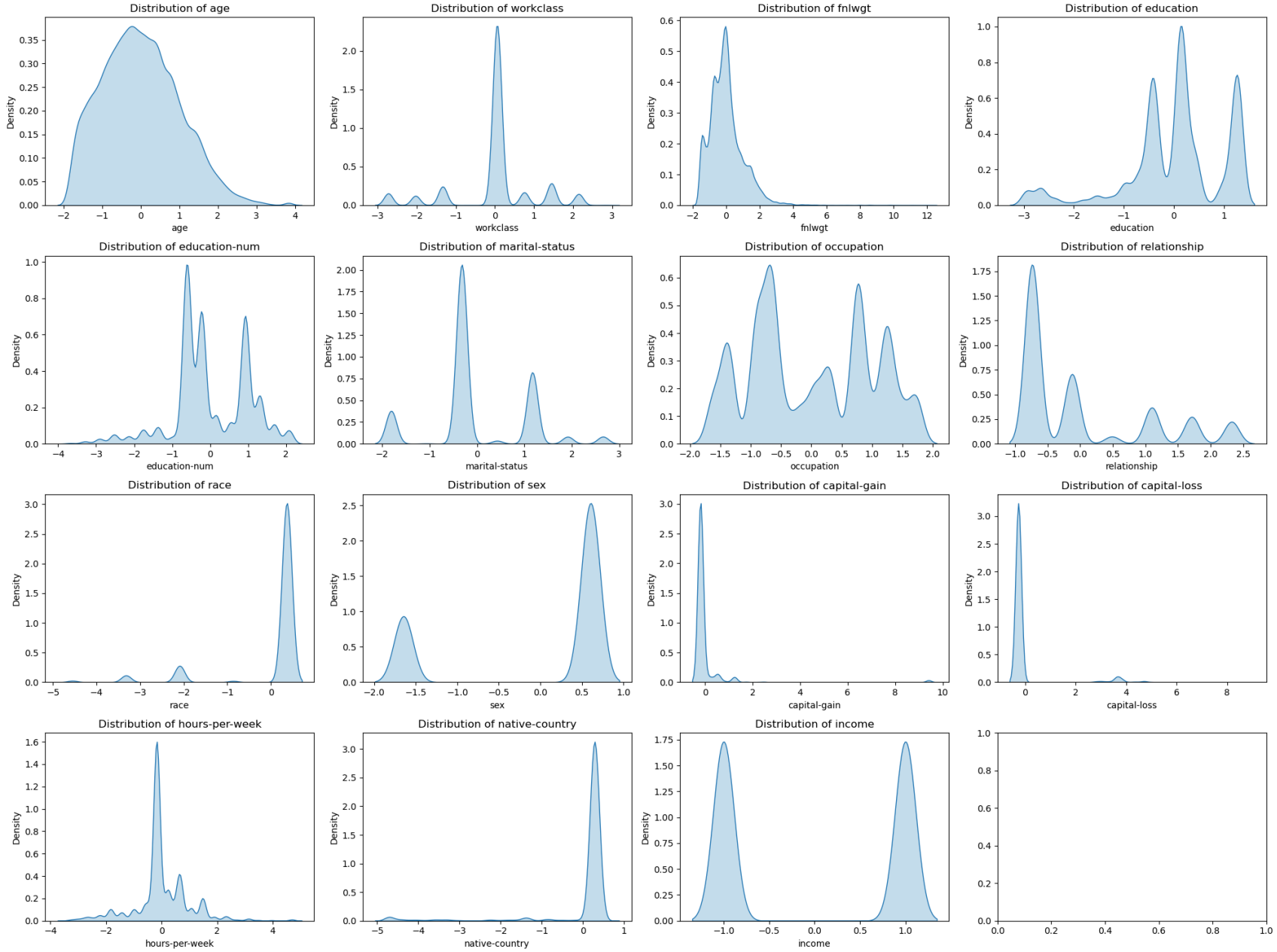
```
==== Display the first few rows of the scaled dataset  ======
================================================================================

        age  workclass    fnlwgt  education  education-num  marital-status  \
0 -0.421965   0.058001  7.214798   0.143631      -0.609271       -0.320142
1 -0.344544   0.058001 -0.376960  -0.416349       0.922407       -1.809609
2 -0.344544   1.449698 -1.437610   0.423621       1.305326       -0.320142
3 -0.034861   0.753850 -0.828717   0.143631      -0.609271       -0.320142
4  1.590975   0.058001 -0.260232  -0.416349       0.922407       -0.320142

   occupation  relationship      race       sex  capital-gain  capital-loss  \
0   -0.662863      2.322569  0.365828 -1.647066      -0.19418     -0.261603
1    0.767009      1.099110  0.365828 -1.647066      -0.19418     -0.261603
2   -0.424551     -0.736080  0.365828  0.607140      -0.19418     -0.261603
3   -0.662863     -0.736080  0.365828  0.607140      -0.19418     -0.261603
4    1.243632     -0.736080  0.365828  0.607140      -0.19418     -0.261603

   hours-per-week  native-country  income
0       -0.175082        0.283914     1.0
1       -0.175082        0.283914    -1.0
2        2.306827        0.283914    -1.0
3       -0.175082        0.283914    -1.0
4       -0.175082        0.283914     1.0
================================================================================
```

In [ ]:

Plot the distribution curves for all features.

In [20]:
```
import pandas as pd
import matplotlib.pyplot as plt
```

```
import seaborn as sns
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

features = df_scaled.columns
# Create a figure and axes

fig, axes = plt.subplots(4, 4, figsize=(20, 15))
# Plot each feature
for ax, feature in zip(axes.flatten(), features):
    sns.kdeplot(df_scaled[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()
```


Distribution of age
Distribution of workclass
Distribution of fnlwgt
Distribution of education
Distribution of education-num
Distribution of marital-status
Distribution of occupation
Distribution of relationship
Distribution of race
Distribution of sex
Distribution of capital-gain
Distribution of capital-loss
Distribution of hours-per-week
Distribution of native-country
Distribution of income
Density
age
workclass
fnlwgt
education
education-num
marital-status
occupation
relationship
race
sex
capital-gain
capital-loss
hours-per-week
native-country
income

In [ ]:

In [22]:
```
print(f"{df_scaled['income'].value_counts()}")
print("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~")
correlation_matrix = df_scaled.corr()
print(correlation_matrix['income'])
```

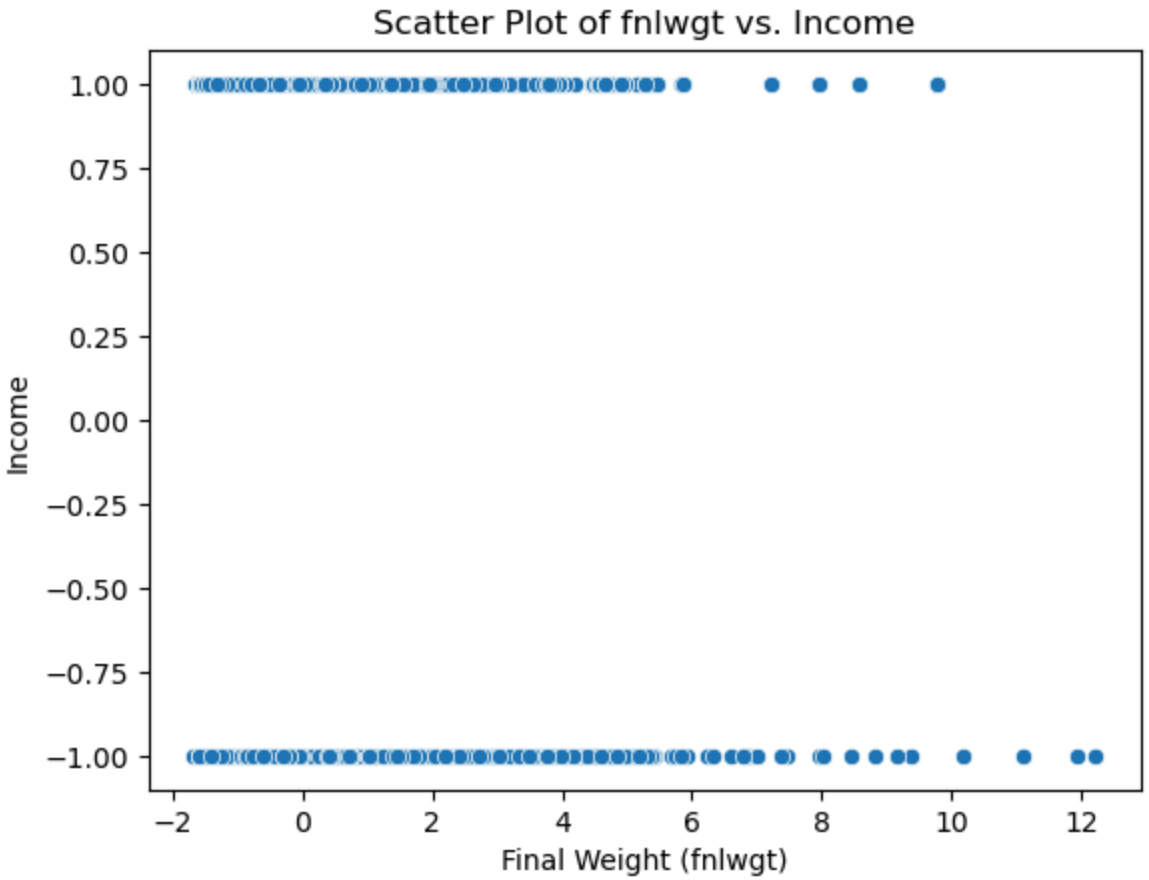
```
income
 1.0    24720
-1.0    24720
Name: count, dtype: int64
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
age               0.283867
workclass         0.062674
fnlwgt           -0.003776
education         0.101270
education-num     0.381409
marital-status   -0.261425
occupation        0.092582
relationship     -0.287602
race              0.087273
sex               0.267550
capital-gain      0.179897
capital-loss      0.151856
hours-per-week    0.271032
native-country    0.017224
income            1.000000
Name: income, dtype: float64
```

Three features have very weak correlation with target feature. So we can eliminate those features Fnlwgt (-0.003776): This near-zero correlation suggests that final weight (fnlwgt) has almost no association with income. Native-country (0.017224): This very weak positive correlation indicates that the native country has almost no association with income.
Workclass (0.062674): This weak positive correlation suggests that workclass has a slight association with income, but its impact may be minimal.

Relationship against income of : Fnlwgt, Native-country and Workclass

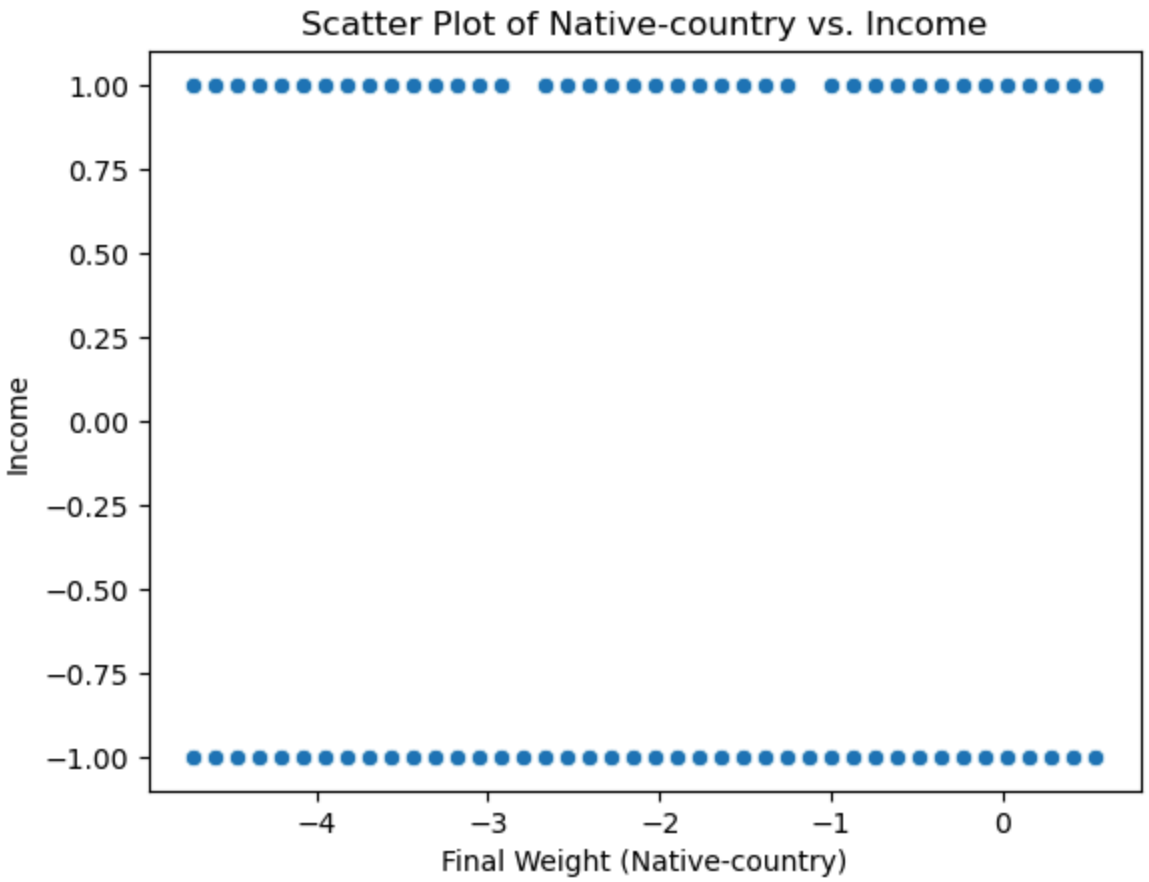
In [23]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming you have a pandas DataFrame named df with 'fnlwgt' and 'income' columns
```

```
sns.scatterplot(data=df_scaled, x='fnlwgt', y='income')
plt.title('Scatter Plot of fnlwgt vs. Income')
plt.xlabel('Final Weight (fnlwgt)')
plt.ylabel('Income')
plt.show()
```



In [24]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming you have a pandas DataFrame named df with 'fnlwgt' and 'income' columns
sns.scatterplot(data=df_scaled, x='native-country', y='income')
plt.title('Scatter Plot of Native-country vs. Income')
plt.xlabel('Final Weight (Native-country)')
```

```
plt.ylabel('Income')
plt.show()
```



In [ ]:

In [25]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming you have a pandas DataFrame named df with 'fnlwgt' and 'income' columns
sns.scatterplot(data=df_scaled, x='workclass', y='income')
plt.title('Scatter Plot of Work Class vs. Income')
plt.xlabel('Final Weight (Work Class)')
plt.ylabel('Income')
plt.show()
```

Eliminate all three features : Fnlwgt, Native-country and Workclass

In [26]:
```
df_rc = df_scaled.drop(columns=['fnlwgt'])
df_rc = df_rc.drop(columns=['workclass'])
df_rc = df_rc.drop(columns=['native-country'])
df_rc.head(5)
```

Out[26]:

	age	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	in
0	-0.421965	0.143631	-0.609271	-0.320142	-0.662863	2.322569	0.365828	-1.647066	-0.19418	-0.261603	-0.175082	
1	-0.344544	-0.416349	0.922407	-1.809609	0.767009	1.099110	0.365828	-1.647066	-0.19418	-0.261603	-0.175082	
2	-0.344544	0.423621	1.305326	-0.320142	-0.424551	-0.736080	0.365828	0.607140	-0.19418	-0.261603	2.306827	
3	-0.034861	0.143631	-0.609271	-0.320142	-0.662863	-0.736080	0.365828	0.607140	-0.19418	-0.261603	-0.175082	
4	1.590975	-0.416349	0.922407	-0.320142	1.243632	-0.736080	0.365828	0.607140	-0.19418	-0.261603	-0.175082	

Applyy Robust scalling

In [27]:
```
from sklearn.preprocessing import RobustScaler
# Assuming you have a pandas DataFrame named df with your features
scaler = RobustScaler()
scaled_features = scaler.fit_transform(df_rc)

# Converting scaled features back to DataFrame for better handling
df_scaled = pd.DataFrame(scaled_features, columns=df_rc.columns)
df_scaled.head(5)
```

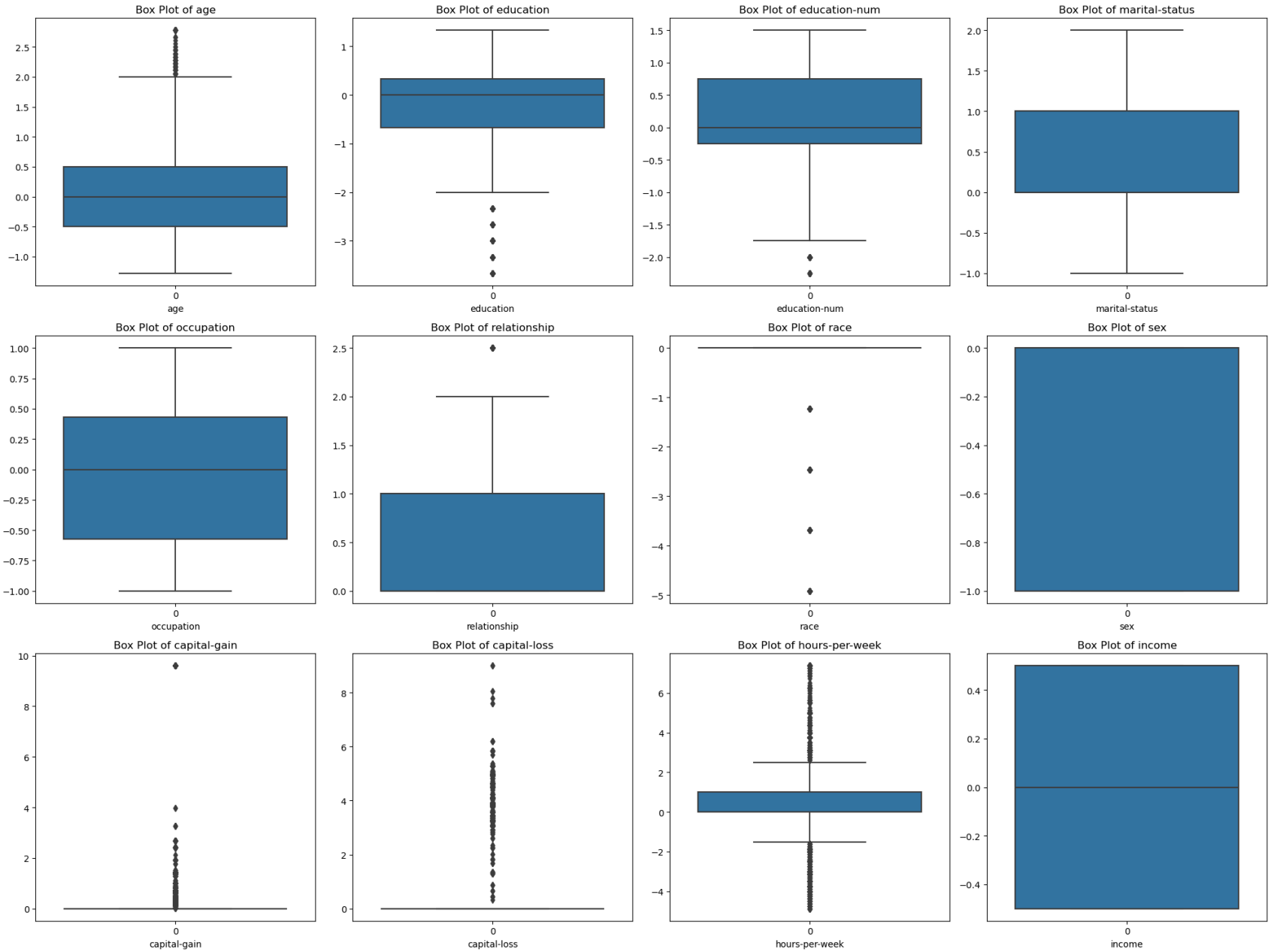
Out[27]:

	age	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	income
0	-0.277778	0.000000	-0.25	0.0	-0.428571	2.5	0.0	-1.0	0.0	0.0	0.00	0.5
1	-0.222222	-0.666667	0.75	-1.0	0.428571	1.5	0.0	-1.0	0.0	0.0	0.00	-0.5
2	-0.222222	0.333333	1.00	0.0	-0.285714	0.0	0.0	0.0	0.0	0.0	3.75	-0.5
3	0.000000	0.000000	-0.25	0.0	-0.428571	0.0	0.0	0.0	0.0	0.0	0.00	-0.5
4	1.166667	-0.666667	0.75	0.0	0.714286	0.0	0.0	0.0	0.0	0.0	0.00	0.5

Check Outliers using by Box plots.

In [28]:

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

features = df_scaled.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=df_scaled[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])

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



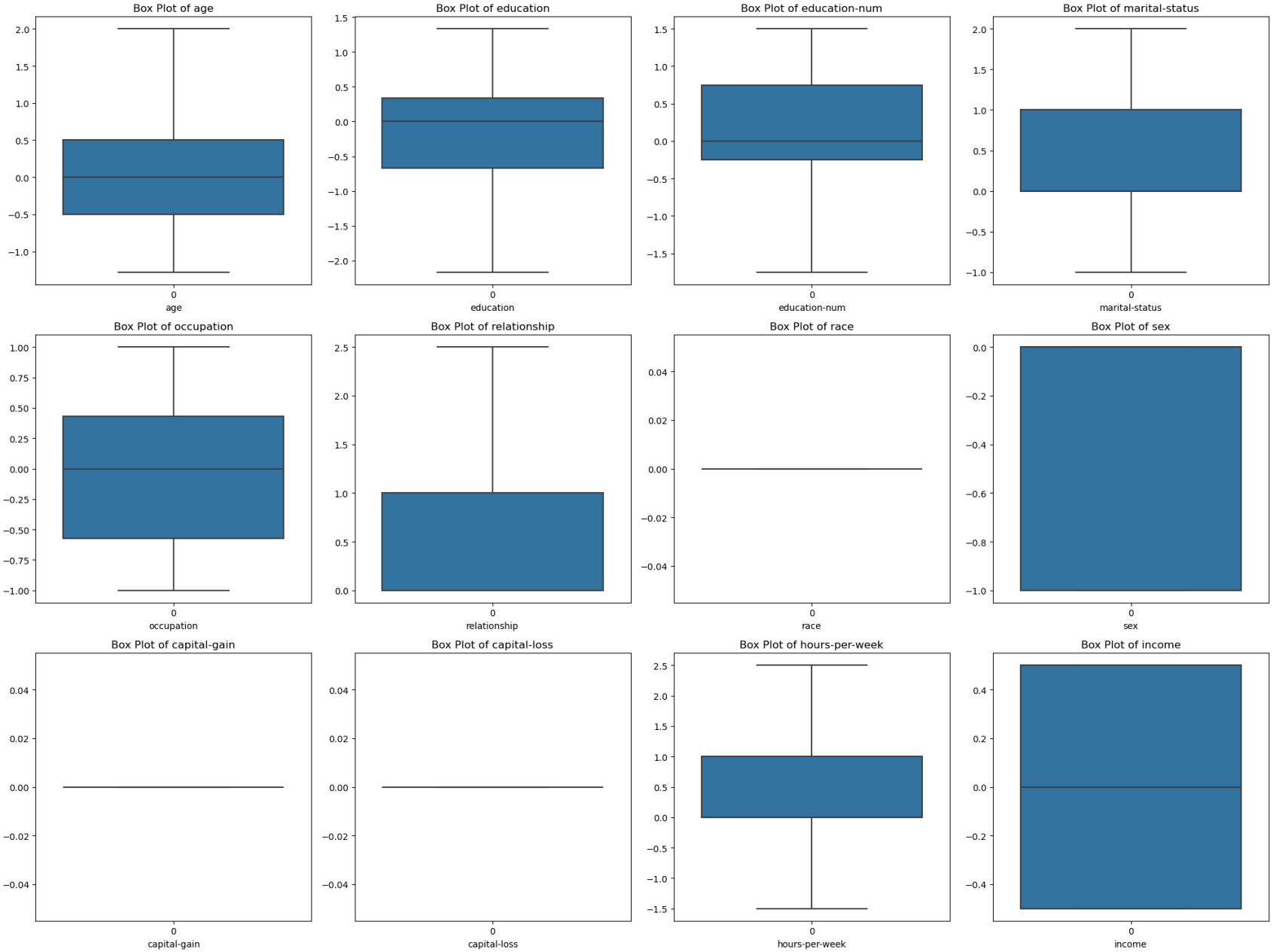
Caping outliers using IQR method.

In [29]:
```
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)

capped_df = df_scaled.apply(cap_outliers)
```

After capping Outlier via Box-plot

In [30]:
```
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("=========== After capping Outlier via Box-plot ===========")
print("============================================================")
plt.tight_layout()
plt.show()
```

```
=========== After capping Outlier via Box-plot ===========
============================================================
```



In [31]:
```
print(f"{capped_df['income'].value_counts()}")
```

```
income
 0.5    24720
-0.5    24720
Name: count, dtype: int64
```

income : target feature make it categorical

In [32]:
```python
# Map income values to categorical labels
income_mapping = {0.5: '<=50K', -0.5: '>50K'}
data_cat = capped_df.copy()
###data_cat['income'] = data_cat['income'].map(income_mapping)
data_cat['income'] = data_cat['income'].map(income_mapping).astype('category')

# Confirm the transformation
print(data_cat['income'].value_counts())
print(data_cat['income'].dtype)
```

```
income
<=50K    24720
>50K     24720
Name: count, dtype: int64
category
```

In [ ]:
```python

```

Split data and split column with target and independent features

In [33]:
```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, accuracy_score
data = data_cat.copy()
# Step 2: Define features and target
X = data.drop(columns=['income'])  # Features
y = data['income']                 # Target

# Step 3: Split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

Decision Tree for classify target 'income'

In [45]:
```
# Step 4: Train the decision tree classifier
dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(X_train, y_train)

# Step 5: Evaluate the model
y_pred = dt_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)
report
```

Out[45]:
```
'              precision    recall  f1-score   support\n\n       <=50K       0.82      0.95      0.88      7350\n
>50K       0.94      0.79      0.86      7482\n\n    accuracy                           0.87     14832\n   macro avg
0.88      0.87      0.87     14832\nweighted avg       0.88      0.87      0.87     14832\n'
```

In [35]:
```
import pandas as pd
from sklearn.metrics import confusion_matrix

# Assuming y_test and y_pred are your true labels and predicted labels
conf_matrix = confusion_matrix(y_test, y_pred)

# Create a pandas DataFrame for better visualization
conf_matrix_df = pd.DataFrame(conf_matrix,
                              index=['True <=50K', 'True >50K'],
                              columns=['Predicted <=50K', 'Predicted >50K'])

print(conf_matrix_df)
```

```
           Predicted <=50K  Predicted >50K
True <=50K             6975             375
True >50K              1573            5909
```

Random Forest : Classifier

In [36]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import precision_score, recall_score, classification_report, accuracy_score

data = data_cat.copy()
# Step 2: Define features and target
```

```
X = data.drop(columns=['income'])  # Features
y = data['income']                 # Target

# Step 3: Split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Train a Random Forest Classifier
rf_model = RandomForestClassifier(random_state=42, n_estimators=100)
rf_model.fit(X_train, y_train)

# Predict and evaluate the model
y_pred_rf = rf_model.predict(X_test)

# Calculate precision, recall, and accuracy
precision_rf = precision_score(y_test, y_pred_rf, average=None)
recall_rf = recall_score(y_test, y_pred_rf, average=None)
accuracy_rf = accuracy_score(y_test, y_pred_rf)
report_rf = classification_report(y_test, y_pred_rf)

print(f"Accuracy: {accuracy_rf}")
print(f"Precision for each class: {precision_rf}")
print(f"Recall for each class: {recall_rf}")
print("Classification Report:")
print(report_rf)
```

```
Accuracy: 0.8801240560949298
Precision for each class: [0.83655472 0.9351541 ]
Recall for each class: [0.94217687 0.819166  ]
Classification Report:
              precision    recall  f1-score   support

       <=50K       0.84      0.94      0.89      7350
        >50K       0.94      0.82      0.87      7482

    accuracy                           0.88     14832
   macro avg       0.89      0.88      0.88     14832
weighted avg       0.89      0.88      0.88     14832
```

In [89]:

```
import pandas as pd
from sklearn.metrics import confusion_matrix
```

```
# Assuming y_test and y_pred are your true labels and predicted labels
conf_matrix = confusion_matrix(y_test, y_pred_rf)

# Create a pandas DataFrame for better visualization
conf_matrix_df = pd.DataFrame(conf_matrix,
                              index=['True <=50K', 'True >50K'],
                              columns=['Predicted <=50K', 'Predicted >50K'])

print(conf_matrix_df)
```

```
            Predicted <=50K  Predicted >50K
True <=50K             6925             425
True >50K              1353            6129
```

GridsearchCV -hyperparameter tuning with : Random Forest

In [38]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix

# Define the model
print("Define the model")
rf = RandomForestClassifier()
# Define the parameter grid
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [10, 20],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2],
    'max_features': ['sqrt'],
    'bootstrap': [True]
}

print("Start.............")
# Implement Grid Search
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5, scoring='accuracy')

print("Starting grid search...")
grid_search.fit(X_train, y_train)
print("Grid search completed.")
```

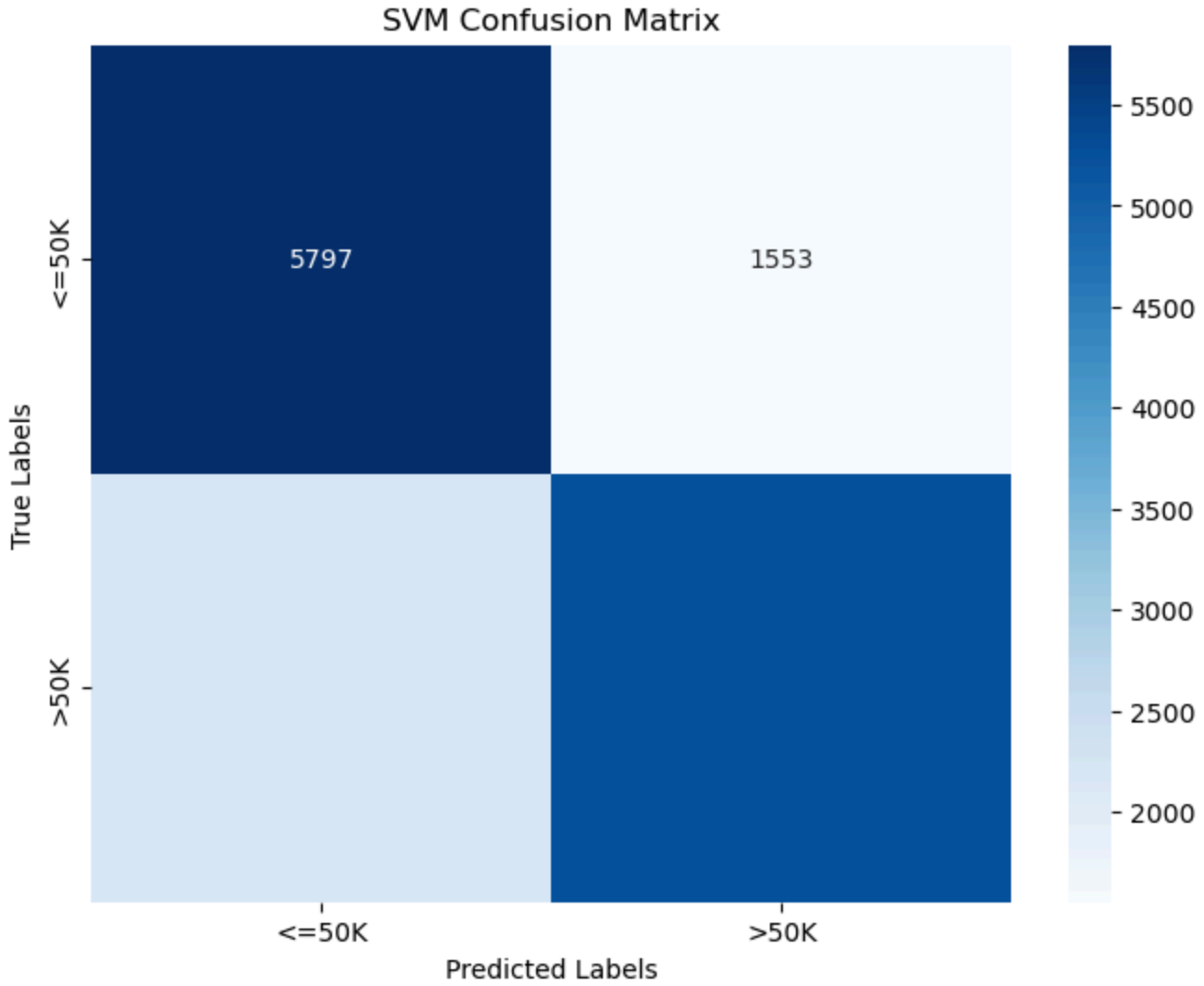
```python
# Get the best parameters
best_params = grid_search.best_params_
print(f"Best parameters: {best_params}")

# Use the best parameters to fit the model
best_rf = RandomForestClassifier(**best_params)
best_rf.fit(X_train, y_train)
y_pred = best_rf.predict(X_test)
print("Model fitting and prediction completed.")

# Evaluate the model

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Define the model
Start.............
Starting grid search...
Grid search completed.
Best parameters: {'bootstrap': True, 'max_depth': 20, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_spl
it': 2, 'n_estimators': 100}
Model fitting and prediction completed.
Confusion Matrix:
[[6990  360]
 [1411 6071]]
Classification Report:
              precision    recall  f1-score   support

       <=50K       0.83      0.95      0.89      7350
        >50K       0.94      0.81      0.87      7482

    accuracy                           0.88     14832
   macro avg       0.89      0.88      0.88     14832
weighted avg       0.89      0.88      0.88     14832

2222222222222222222
```

SVM : Support Vector Machine Classifier

In [40]:
```
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
import seaborn as sns
import matplotlib.pyplot as plt

# Train an SVM classifier
svm_model = SVC(kernel='linear', random_state=42)
svm_model.fit(X_train, y_train)

# Predict using the SVM model
y_pred_svm = svm_model.predict(X_test)

# Evaluate the model
accuracy_svm = accuracy_score(y_test, y_pred_svm)
conf_matrix_svm = confusion_matrix(y_test, y_pred_svm)
report_svm = classification_report(y_test, y_pred_svm)

print(f"Accuracy: {accuracy_svm}")
print("Classification Report:")
print(report_svm)

# Display the confusion matrix as a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_svm, annot=True, fmt="d", cmap="Blues",
            xticklabels=['<=50K', '>50K'], yticklabels=['<=50K', '>50K'])
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("SVM Confusion Matrix")
plt.show()
```

```
Accuracy: 0.7465614886731392
Classification Report:
              precision    recall  f1-score   support

       <=50K       0.72      0.79      0.76      7350
        >50K       0.77      0.71      0.74      7482

    accuracy                           0.75     14832
   macro avg       0.75      0.75      0.75     14832
weighted avg       0.75      0.75      0.75     14832
```



In [ ]:

In [42]:
```
import pandas as pd
from sklearn.metrics import confusion_matrix

# Assuming y_test and y_pred are your true labels and predicted labels
conf_matrix = confusion_matrix(y_test, y_pred_svm)
```

```
# Create a pandas DataFrame for better visualization
conf_matrix_df = pd.DataFrame(conf_matrix,
                              index=['True <=50K', 'True >50K'],
                              columns=['Predicted <=50K', 'Predicted >50K'])

print(conf_matrix_df)
```

```
           Predicted <=50K  Predicted >50K
True <=50K             5797            1553
True >50K              2206            5276
```

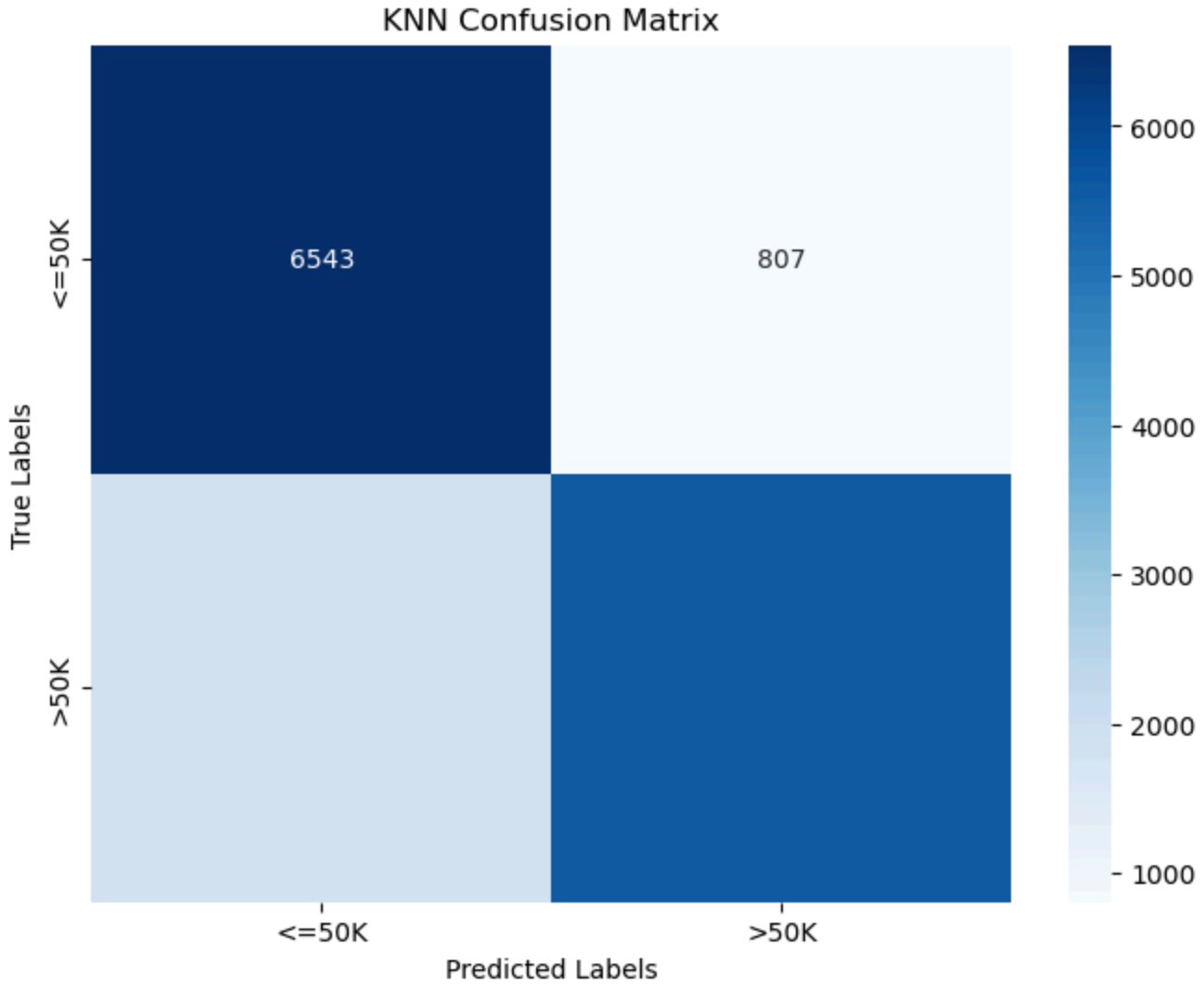
KNN Classifier

In [43]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
import seaborn as sns
import matplotlib.pyplot as plt

# Train a KNN classifier
knn_model = KNeighborsClassifier(n_neighbors=5)  # Default n_neighbors=5
knn_model.fit(X_train, y_train)

# Predict using the KNN model
y_pred_knn = knn_model.predict(X_test)

# Evaluate the model
accuracy_knn = accuracy_score(y_test, y_pred_knn)
conf_matrix_knn = confusion_matrix(y_test, y_pred_knn)
report_knn = classification_report(y_test, y_pred_knn)

print(f"Accuracy: {accuracy_knn}")
print("Classification Report:")
print(report_knn)

# Display the confusion matrix as a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_knn, annot=True, fmt="d", cmap="Blues",
            xticklabels=['<=50K', '>50K'], yticklabels=['<=50K', '>50K'])
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
```

```
plt.title("KNN Confusion Matrix")
plt.show()
```

```
Accuracy: 0.8183656957928802
Classification Report:
              precision    recall  f1-score   support

       <=50K       0.78      0.89      0.83      7350
        >50K       0.87      0.75      0.81      7482

    accuracy                           0.82     14832
   macro avg       0.83      0.82      0.82     14832
weighted avg       0.83      0.82      0.82     14832
```



In [44]:
```
import pandas as pd
from sklearn.metrics import confusion_matrix

# Assuming y_test and y_pred are your true labels and predicted labels
conf_matrix = confusion_matrix(y_test, y_pred_knn)

# Create a pandas DataFrame for better visualization
conf_matrix_df = pd.DataFrame(conf_matrix,
                              index=['True <=50K', 'True >50K'],
```

```
                               columns=['Predicted <=50K', 'Predicted >50K'])

print(conf_matrix_df)
```

```
           Predicted <=50K  Predicted >50K
True <=50K             6543             807
True >50K              1887            5595
```

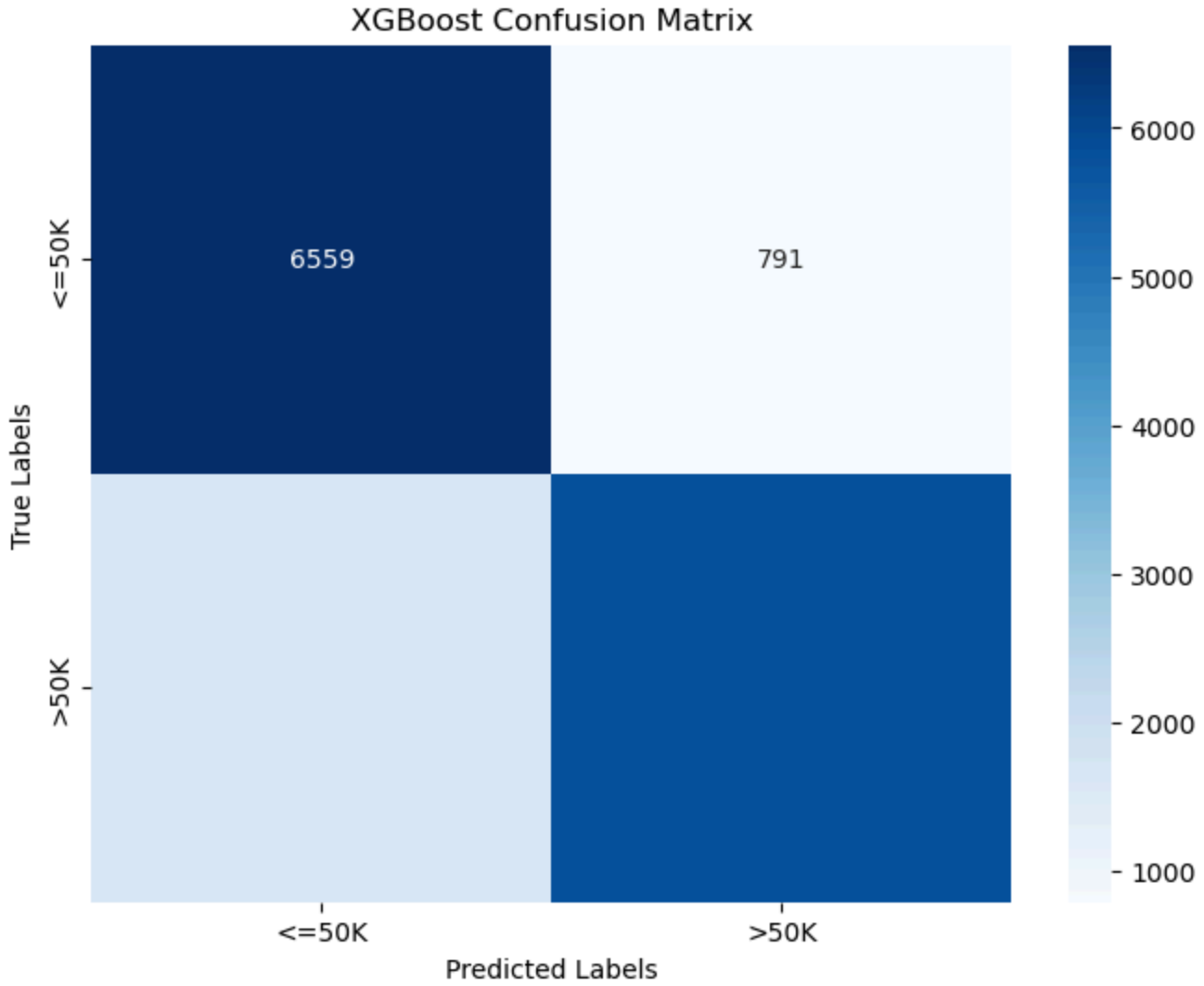
XGBoost : Classifier

In [48]:

```
# Map target labels to numeric values
y_train_numeric = y_train.map({'<=50K': 0, '>50K': 1})
y_test_numeric = y_test.map({'<=50K': 0, '>50K': 1})

# Train an XGBoost classifier
xgb_model = XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42)
xgb_model.fit(X_train, y_train_numeric)

# Predict using the XGBoost model
y_pred_xgb = xgb_model.predict(X_test)

# Evaluate the model
accuracy_xgb = accuracy_score(y_test_numeric, y_pred_xgb)
conf_matrix_xgb = confusion_matrix(y_test_numeric, y_pred_xgb)
report_xgb = classification_report(y_test_numeric, y_pred_xgb)

print(f"Accuracy: {accuracy_xgb}")
print("Classification Report:")
print(report_xgb)

# Display the confusion matrix as a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_xgb, annot=True, fmt="d", cmap="Blues",
            xticklabels=['<=50K', '>50K'], yticklabels=['<=50K', '>50K'])
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("XGBoost Confusion Matrix")
plt.show()
```

```
C:\Users\hp\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning: [18:23:24] WARNING: C:\buildkite-agent\buil
ds\buildkite-windows-cpu-autoscaling-group-i-0c55ff5f71b100e98-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
```

```
Accuracy: 0.8365695792880259
Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.89      0.84      7350
           1       0.88      0.78      0.83      7482

    accuracy                           0.84     14832
   macro avg       0.84      0.84      0.84     14832
weighted avg       0.84      0.84      0.84     14832
```

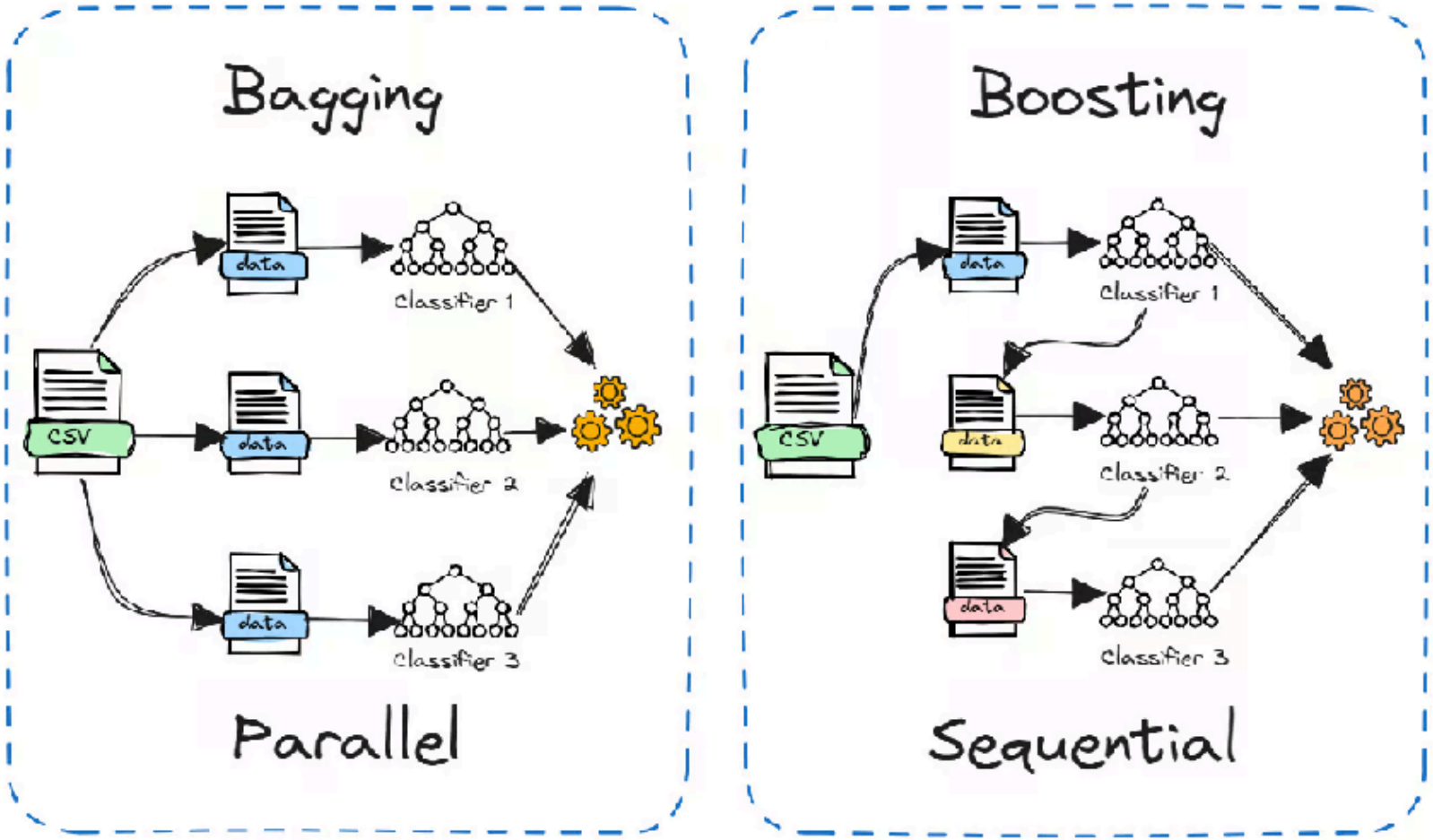


For 'Income Classification', We have used, Decision Tree, Random Forest, GridsearchCV with Random Forest, XGBoost, SVM and KNN classifier

their is Best result from GridSearchCV with Random Forest 84 percision rate and 92 is recall approximate.

In [ ]:

Question: d) What is the difference between Bagging Classifier and voting classifier. Explain the working of both the algorithms.



Bagging Classifier :

- **Ensemble Learning:** Bagging is a type of ensemble learning method. Ensemble methods combine the predictions of multiple base models to produce a more accurate and stable prediction than any individual model could achieve.

- **Bootstrap Aggregating:** "Bagging" is short for "Bootstrap Aggregating".
  - **Bootstrap:** It involves creating multiple subsets of the original training data by sampling *with replacement*.
  - **Aggregating:** Each subset trains a separate instance of the same base classifier (e.g., a decision tree). The final prediction aggregates these predictions, typically by majority voting (classification) or averaging (regression).

How does it work?

1. **Bootstrap Sampling:** Create 'n' bootstrap samples from the original dataset (sampling with replacement).
2. **Base Model Training:** Train a base classifier on each of the 'n' bootstrap samples independently.
3. **Prediction Aggregation:**
   - **Classification:** The final prediction is the class with the most votes.
   - **Regression:** The final prediction is the average of all predictions.

When to use Bagging Classifiers?

- **High Variance Models:** Effective for models with high variance (e.g., decision trees).
- **Complex Datasets:** Improves accuracy and stability on complex datasets.
- **Improving Generalization:** Helps the model perform well on unseen data.

Where are Bagging Classifiers used?

- **Machine Learning:** Widely used in various applications.
- **Random Forests:** An extension of bagging with decision trees and feature randomness.
- **Various Domains:** Image classification, medical diagnosis, financial modeling, NLP, etc.

Advantages and Limitations

Advantages:

- Improved Accuracy
- Reduced Overfitting

- Robustness
- Easy Parallelization

Limitations:

- Reduced Interpretability
- Higher Computational Cost

Example Baggging Classifier using scikit-learn (Python):

In [7]:
```python
# In a Code cell:
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_classification
from sklearn.metrics import accuracy_score
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)

X, y = make_classification(n_samples=1000, n_features=20, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

bagging_clf = BaggingClassifier(base_estimator=DecisionTreeClassifier(), n_estimators=100, random_state=42)
bagging_clf.fit(X_train, y_train)
y_pred = bagging_clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.8633333333333333
```

Voting Classifier

A Voting Classifier is an ensemble machine learning algorithm that combines the predictions from multiple other classification models. Instead of relying on a single model's prediction, it aggregates the predictions of several models to make a final prediction. This often leads to improved accuracy and robustness compared to using individual models.

There are two main types:

1. **Hard Voting:** Each classifier "votes" for a class, and the class with the majority of votes is the final prediction.
2. **Soft Voting:** Each classifier provides a probability or confidence score for each class. The probabilities are then averaged or summed, and the class with the highest average probability is chosen. Soft voting often performs better.

How it works:

1. **Train multiple classifiers:** Train several different classification models on the same training data.
2. **Combine predictions:**
    - **Hard Voting:** The class with the most votes is the final prediction.
    - **Soft Voting:** The class with the highest average probability is the final prediction.

When to use Voting Classifiers:

- **Diverse Models:** Most effective when base models have different strengths and weaknesses.
- **Improved Accuracy:** Can often achieve higher accuracy than individual models.
- **Robustness:** More robust to noise and outliers.
- **Ensemble Learning:** A type of ensemble learning method.

Advantages:

- Increased Accuracy
- Improved Robustness
- Simple Implementation

Disadvantages:

- Computational Cost
- Interpretability
- Performance Dependence (on base models)

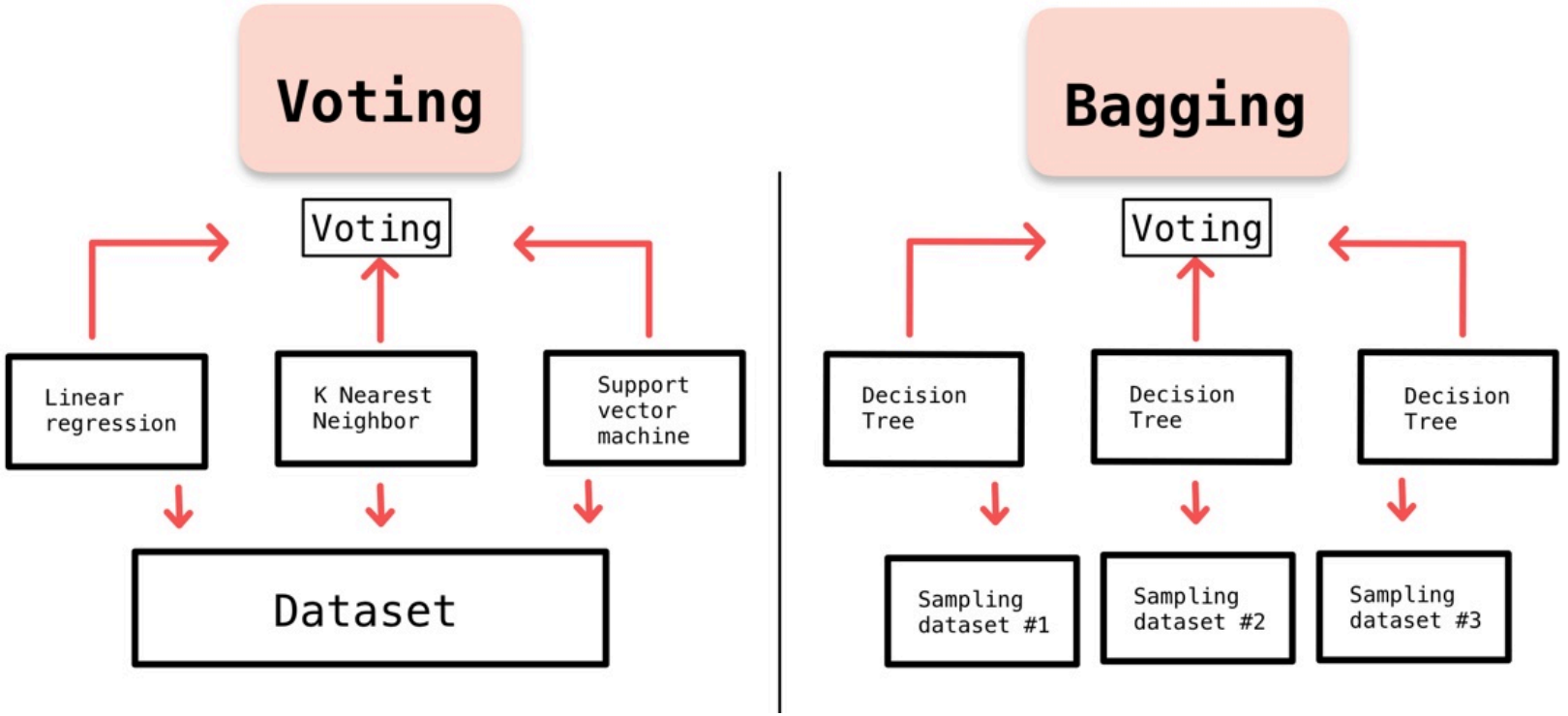
In [3]:

```python
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_classification
from sklearn.metrics import accuracy_score

# Generate a sample dataset
X, y = make_classification(n_samples=1000, n_features=20, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Create individual classifiers
clf1 = LogisticRegression(random_state=1)
clf2 = DecisionTreeClassifier(random_state=1)
clf3 = SVC(probability=True, random_state=1)  # probability=True is needed for soft voting

# Create the voting classifier (soft voting)
eclf_soft = VotingClassifier(estimators=[('lr', clf1), ('dt', clf2), ('svc', clf3)], voting='soft')
eclf_soft.fit(X_train, y_train)
y_pred_soft = eclf_soft.predict(X_test)
accuracy_soft = accuracy_score(y_test, y_pred_soft)
print(f"Soft Voting Accuracy: {accuracy_soft}")

# Create the voting classifier (hard voting)
eclf_hard = VotingClassifier(estimators=[('lr', clf1), ('dt', clf2), ('svc', clf3)], voting='hard')
eclf_hard.fit(X_train, y_train)
y_pred_hard = eclf_hard.predict(X_test)
accuracy_hard = accuracy_score(y_test, y_pred_hard)
print(f"Hard Voting Accuracy: {accuracy_hard}")

print("""
This example demonstrates how to create both hard and soft voting classifiers using `scikit-learn`. Remember that for
""")
```

```
Soft Voting Accuracy: 0.85
Hard Voting Accuracy: 0.8433333333333334

This example demonstrates how to create both hard and soft voting classifiers using `scikit-learn`. Remember that for soft voting, your classifiers must be able to provide probability estimates (e.g., using `probability=True` in `SVC`).
```

Difference between Voting Classifier and Bagging Classifier

Aspect	Voting Classifier	Bagging Classifier
Definition	Combines the predictions of multiple models (could be different models or the same model with different parameters) to make a final prediction.	Uses multiple instances of the same model trained on different subsets of the data to make a final prediction.
Model Diversity	Can use a variety of different models (e.g., Logistic Regression, Decision Trees, SVM, etc.).	Uses the same model but with different subsets of data (e.g., multiple decision trees in the case of a Bagging Classifier).
Aggregation Method	Voting (majority voting for classification or averaging for regression).	Aggregation of predictions (e.g., averaging the outputs for regression or majority voting for classification).
Bias-Variance Tradeoff	Reduces variance by combining different models but doesn't necessarily reduce bias.	Reduces both bias and variance by combining predictions from multiple instances



Quuestion: e) Explain the Bayes theorem in Naïve Bayes Algorithm.

Bayes' Theorem in Naïve Bayes Algorithm

Bayes' Theorem is a mathematical formula used to determine the conditional probability of events. It's a key component of the Naïve Bayes algorithm, which is widely used for classification tasks in machine learning. Let’s break down how it works in this context:

Bayes' Theorem

The formula for Bayes' Theorem is:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

Where:

- $P(A|B)$ is the probability of event A occurring given that B is true.
- $P(B|A)$ is the probability of event B occurring given that A is true.
- $P(A)$ and $P(B)$ are the probabilities of observing A and B independently of each other.

Naïve Bayes Classifier

In the context of Naïve Bayes, the theorem helps us calculate the posterior probability of a class (target) given a set of features (predictors). The term "naïve" comes from the assumption that all predictors are independent of each other, which simplifies the calculations.

Here's a step-by-step outline:

1. **Calculate Prior Probability**: This is the initial probability of each class. For example, if we're classifying emails as spam or not spam, we look at the probability of an email being spam or not based on historical data.
2. **Calculate Likelihood**: For each feature, we calculate the likelihood, which is the probability of the feature given the class. If the features are words in an email, we compute the likelihood of each word appearing in spam and not spam emails.
3. **Calculate Evidence**: This is the total probability of the observed data. It can be seen as a normalizing constant ensuring that the probabilities sum up to 1.
4. **Apply Bayes' Theorem**: Using Bayes' Theorem, we combine these probabilities to find the posterior probability of each class given the features. The class with the highest posterior probability is the predicted class.

In [ ]: