Informatics Institute of Technology

In affiliated with

Robert Gorden University Aberdeen

Artificial Intelligence & Data Science

Machine Learning

Coursework

Name: Gouri Medhavi Napevithanage

Student ID(IIT): 20210808

Student ID(RGU): 2237943

Module Coordinator

Mr. Sahan Priyanyana

Submitted partially fulfilling the requirements for BSc (Hons) in Artificial Intelligence and Data Science degree at Robert Gorden University.

March 2024

Table of Contents

Introduction

This coursework aims to train two machine learning models to solve a simple classification problem – predicting whether the income exceeds $50k/yr based on census data.

The major purpose of completion of this coursework is to prepare a dataset for machine learning, by employing data engineering and feature engineering techniques, as well as machine learning evaluation methodologies.

This course uses two machine learning models: Naïve Bayes Classifier Algorithm and Random Forest Classification.

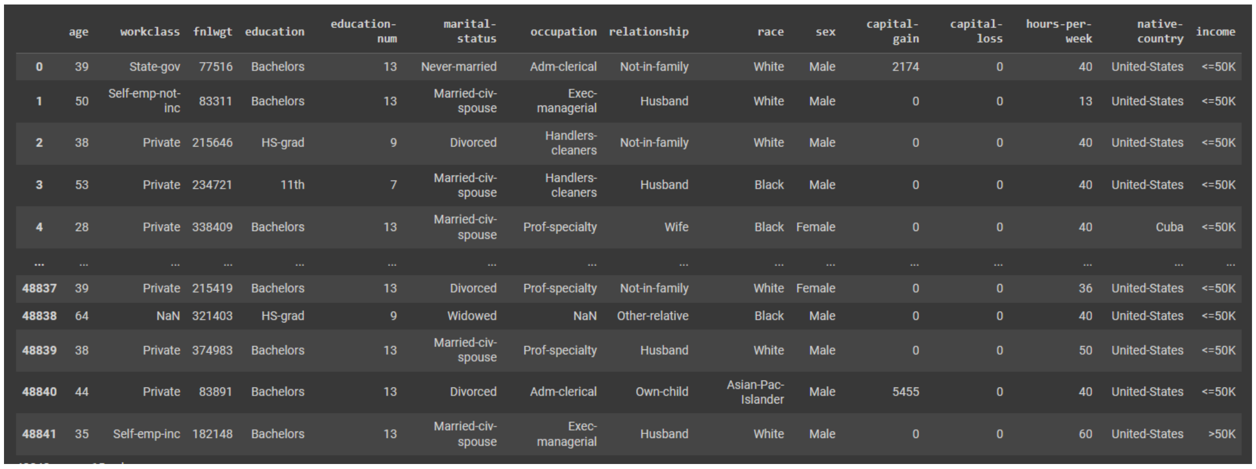
Datasets were obtained from UC Irvine Machine Learning Repository, as indicated in the coursework specification.

Link to dataset: https://archive.ics.uci.edu/dataset/2/adult

Corpus Preparation

DATASET PEPARATION

The first step involves defining column names and reading the "adult.data" and "adult.test" CSV files into different DataFrames. The "adult.test" set's income data is then cleaned by eliminating periods, which are probably meant to represent dollar signs. Ultimately, it creates a single, ready-to-analyze dataset that is saved in the "combined_dataframe" variable, merges the two DataFrames, and resets the index for a clean structure.

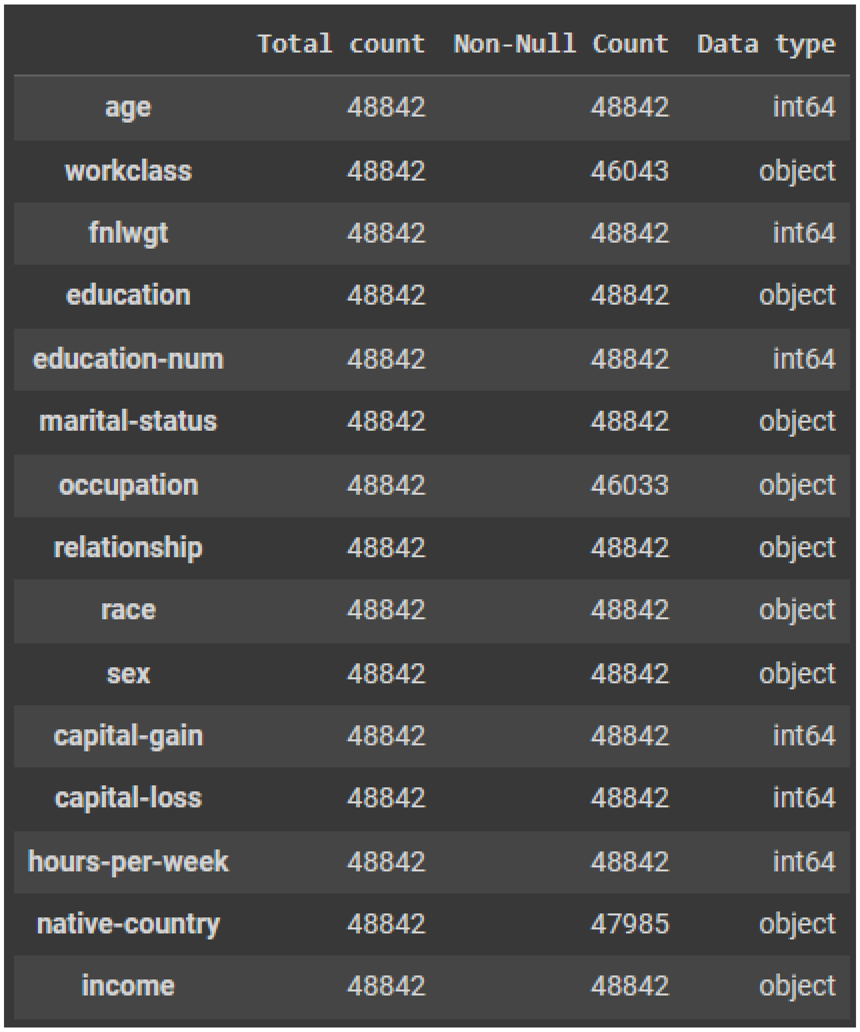


	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	income
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
48837	39	Private	215419	Bachelors	13	Divorced	Prof-specialty	Not-in-family	White	Female	0	0	36	United-States	<=50K
48838	64	NaN	321403	HS-grad	9	Widowed	NaN	Other-relative	Black	Male	0	0	40	United-States	<=50K
48839	38	Private	374983	Bachelors	13	Married-civ-spouse	Prof-specialty	Husband	White	Male	0	0	50	United-States	<=50K
48840	44	Private	83891	Bachelors	13	Divorced	Adm-clerical	Own-child	Asian-Pac-Islander	Male	5455	0	40	United-States	<=50K
48841	35	Self-emp-inc	182148	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	60	United-States	>50K

shows this final DataFrame's dimensions (row and column counts), giving important details about the amount of the data for additional investigation.

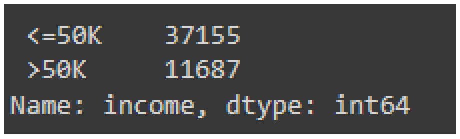
```
Number of rows and columns in the dataframe:(48842, 15)
```

Create a summary of ‘combined_dataframe’, showing the total entries, non-null values, and data types for each column. It’s a quick data quality check for your report.



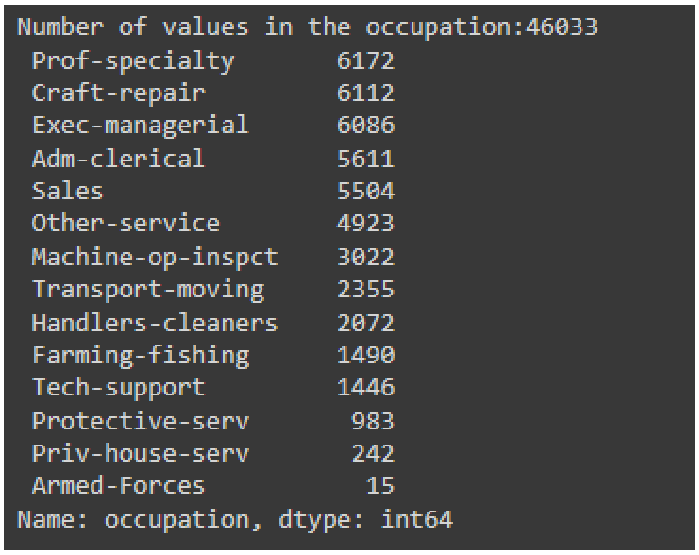
	Total count	Non-Null Count	Data type
age	48842	48842	int64
workclass	48842	46043	object
fnlwgt	48842	48842	int64
education	48842	48842	object
education-num	48842	48842	int64
marital-status	48842	48842	object
occupation	48842	46033	object
relationship	48842	48842	object
race	48842	48842	object
sex	48842	48842	object
capital-gain	48842	48842	int64
capital-loss	48842	48842	int64
hours-per-week	48842	48842	int64
native-country	48842	47985	object
income	48842	48842	object

Counts unique values in the ‘income’ column of ‘combined_dataframe’, providing a frequency distribution of income categories, useful for understanding the data’s composition.

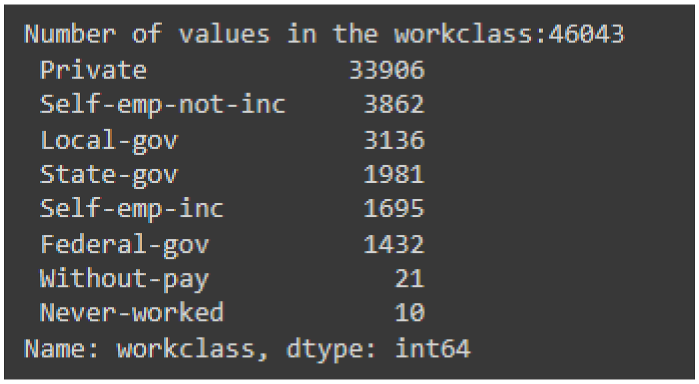


```
 <=50K    37155
 >50K     11687
Name: income, dtype: int64
```

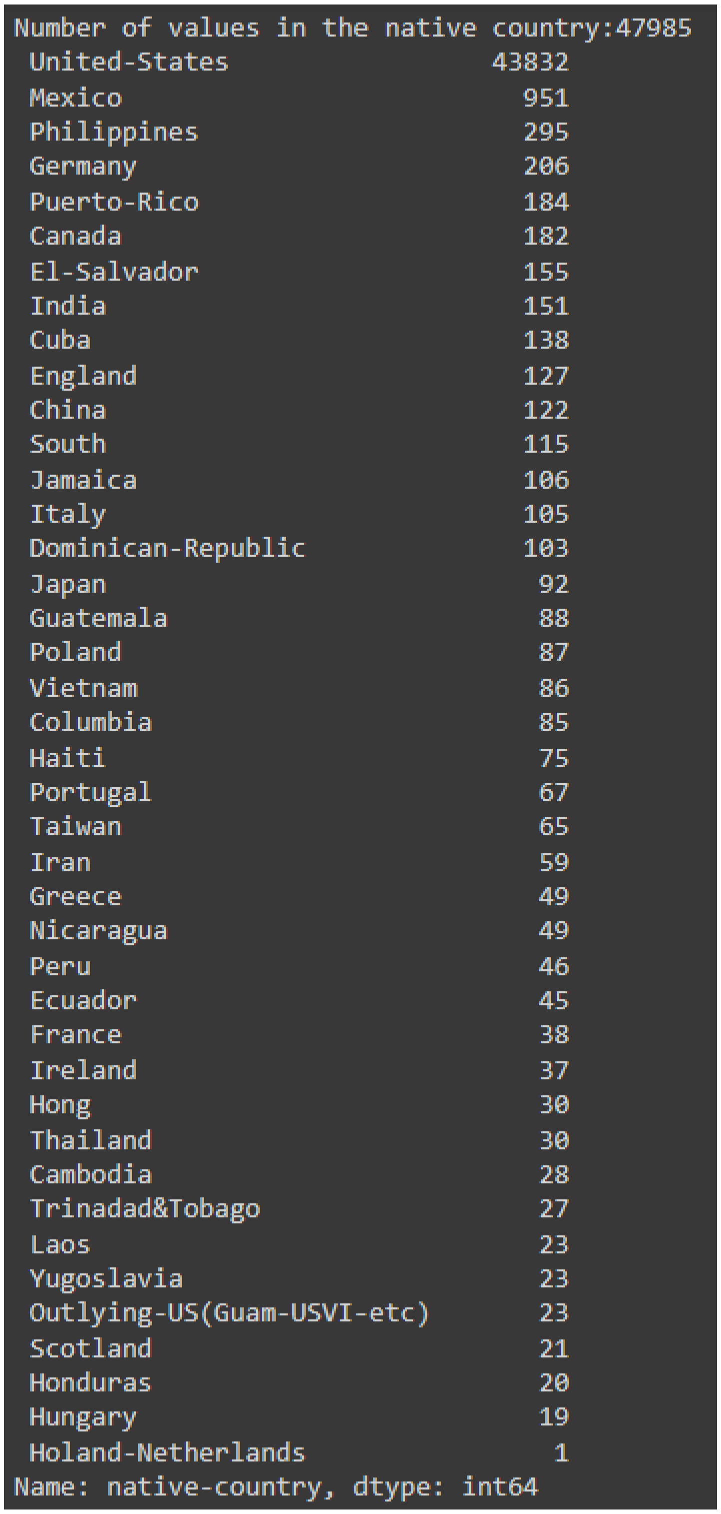
Calculatee and display the total number of non-null entries in the ‘occupation’, ‘native-country’, ‘workclass’ column of combined_dataframe. It then prints this number and shows the count of each unique occupation.



```
Number of values in the occupation:46033
 Prof-specialty       6172
 Craft-repair         6112
 Exec-managerial      6086
 Adm-clerical         5611
 Sales                5504
 Other-service        4923
 Machine-op-inspct    3022
 Transport-moving     2355
 Handlers-cleaners    2072
 Farming-fishing      1490
 Tech-support         1446
 Protective-serv       983
 Priv-house-serv       242
 Armed-Forces           15
Name: occupation, dtype: int64
```

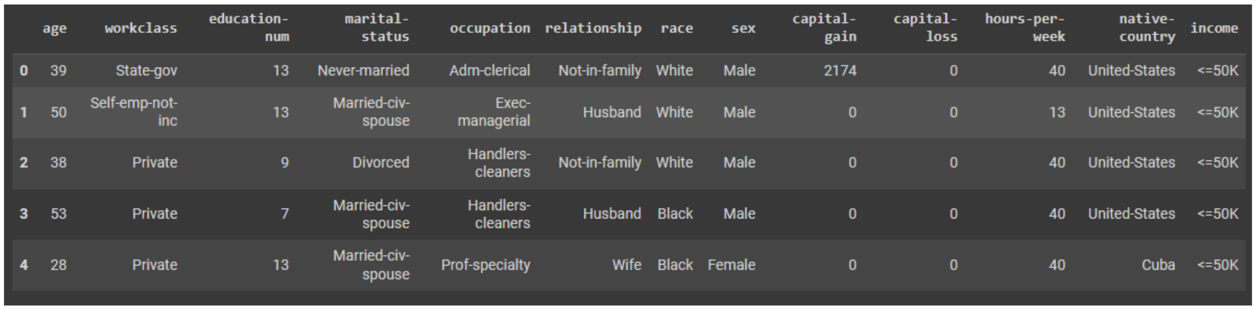


```
Number of values in the workclass:46043
 Private             33906
 Self-emp-not-inc     3862
 Local-gov            3136
 State-gov            1981
 Self-emp-inc         1695
 Federal-gov          1432
 Without-pay            21
 Never-worked           10
Name: workclass, dtype: int64
```



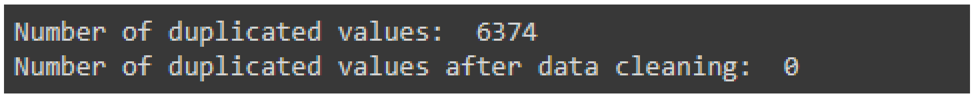
```
Number of values in the native country:47985
 United-States                 43832
 Mexico                          951
 Philippines                     295
 Germany                         206
 Puerto-Rico                     184
 Canada                          182
 El-Salvador                     155
 India                           151
 Cuba                            138
 England                         127
 China                           122
 South                           115
 Jamaica                         106
 Italy                           105
 Dominican-Republic              103
 Japan                            92
 Guatemala                        88
 Poland                           87
 Vietnam                          86
 Columbia                         85
 Haiti                            75
 Portugal                         67
 Taiwan                           65
 Iran                             59
 Greece                           49
 Nicaragua                        49
 Peru                             46
 Ecuador                          45
 France                           38
 Ireland                          37
 Hong                             30
 Thailand                         30
 Cambodia                         28
 Trinadad&Tobago                  27
 Laos                             23
 Yugoslavia                       23
 Outlying-US(Guam-USVI-etc)       23
 Scotland                         21
 Honduras                         20
 Hungary                          19
 Holand-Netherlands                1
Name: native-country, dtype: int64
```

Remove the ‘education’ and ‘fnlwgt’ columns from combined_dataframe because ‘education’ and ‘education_num’ columns duplicate information.



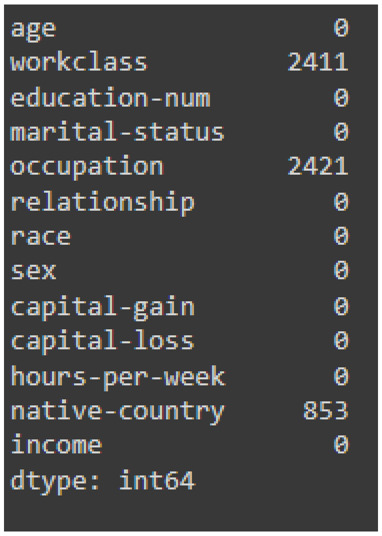
	age	workclass	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	income
0	39	State-gov	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
1	50	Self-emp-not-inc	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
2	38	Private	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
3	53	Private	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
4	28	Private	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K

Identify and print the count of duplicate rows in combined_dataframe. Then, remove the duplicates to create clean_data, ensuring the data is unique.

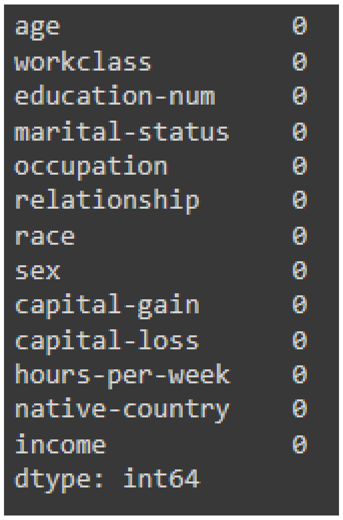


```
Number of duplicated values:  6374
Number of duplicated values after data cleaning:  0
```

Determine the total amount of missing values in every column of the DataFrame named "clean_data" and output the outcome. After deleting all rows with missing values, create a new DataFrame called "cleaned_data." After that, display the number of rows before and after duplicates and missing values have been eliminated. This will allow you to compare the size of the dataset at each step with clarity. To guarantee complete cleaning, lastly look for any missing values in the "cleaned_data" DataFrame.

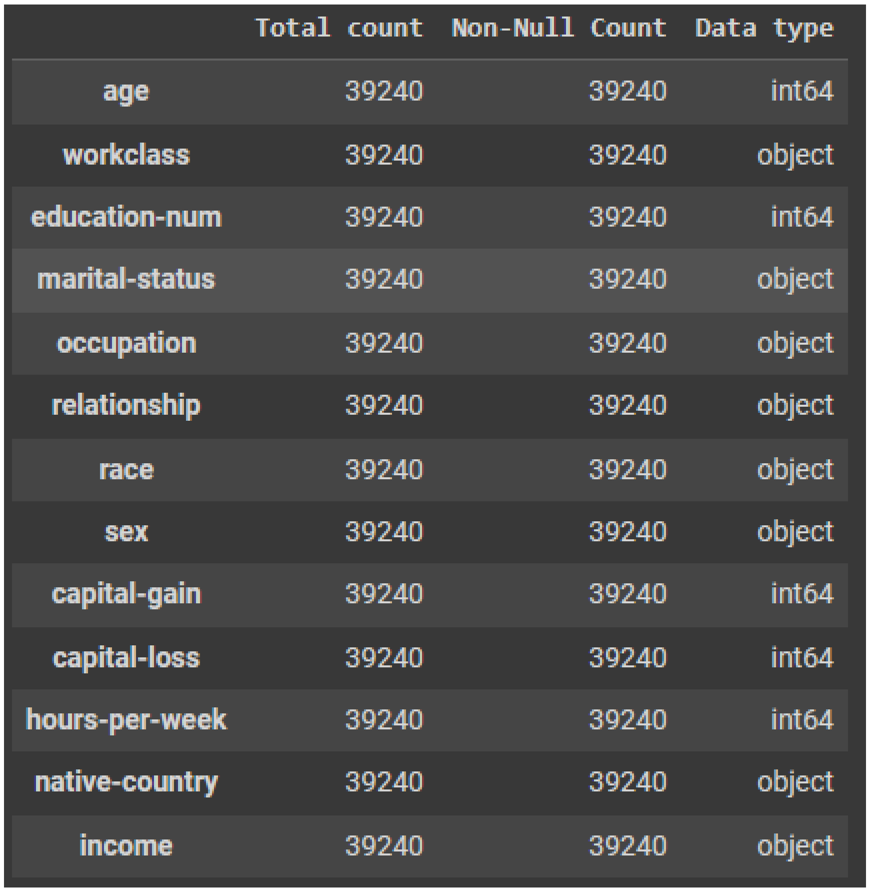


```
age                  0
workclass         2411
education-num        0
marital-status       0
occupation        2421
relationship         0
race                 0
sex                  0
capital-gain         0
capital-loss         0
hours-per-week       0
native-country     853
income               0
dtype: int64
```



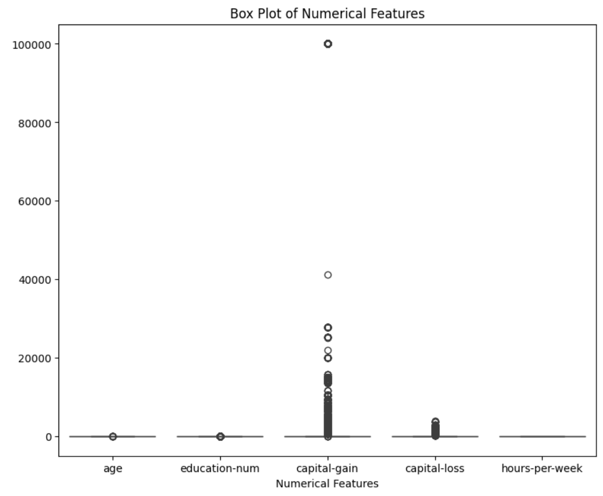
```
age               0
workclass         0
education-num     0
marital-status    0
occupation        0
relationship      0
race              0
sex               0
capital-gain      0
capital-loss      0
hours-per-week    0
native-country    0
income            0
dtype: int64
```

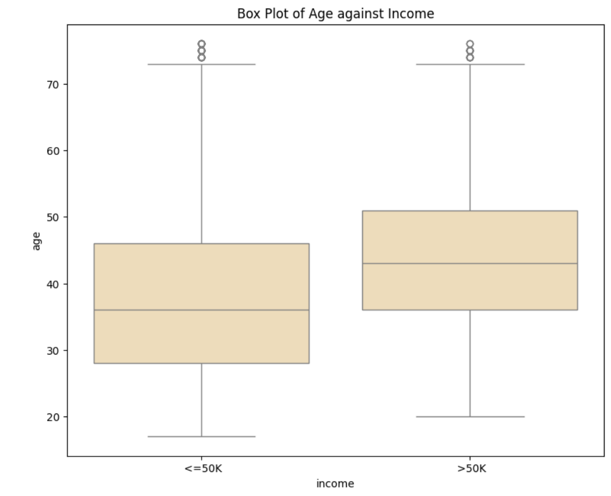
Generate a summary DataFrame named summary_cd that provides a comprehensive overview.

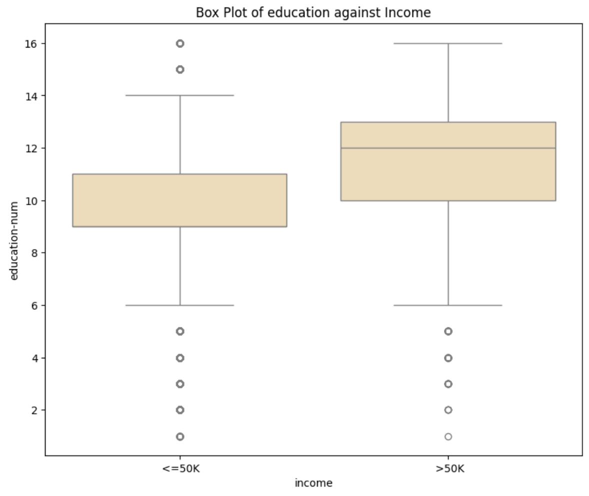


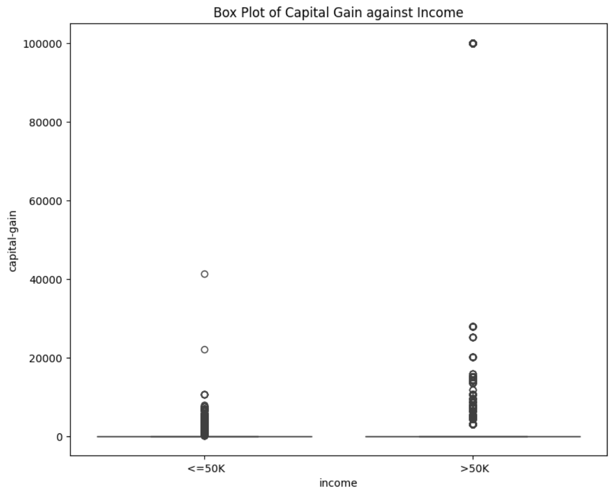
	Total count	Non-Null Count	Data type
age	39240	39240	int64
workclass	39240	39240	object
education-num	39240	39240	int64
marital-status	39240	39240	object
occupation	39240	39240	object
relationship	39240	39240	object
race	39240	39240	object
sex	39240	39240	object
capital-gain	39240	39240	int64
capital-loss	39240	39240	int64
hours-per-week	39240	39240	int64
native-country	39240	39240	object
income	39240	39240	object

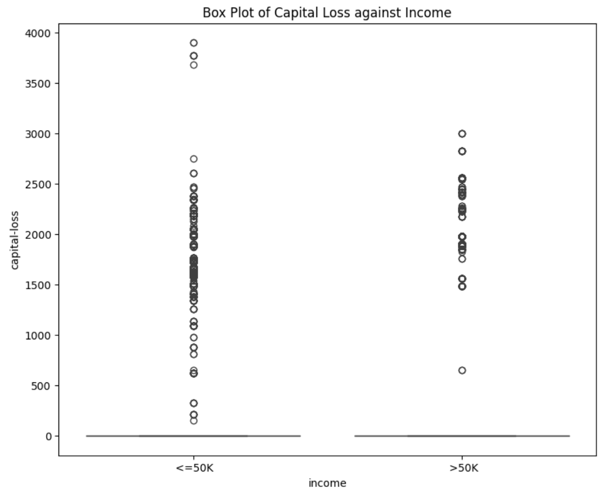
BOX PLOTS AGAINST INCOME FOR NUMERICAL FEATURES

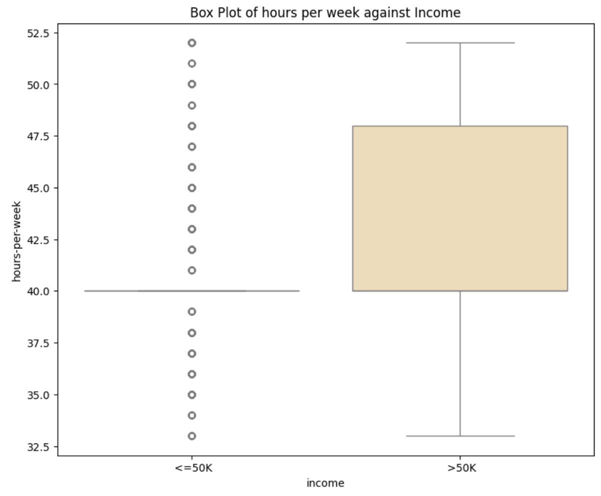






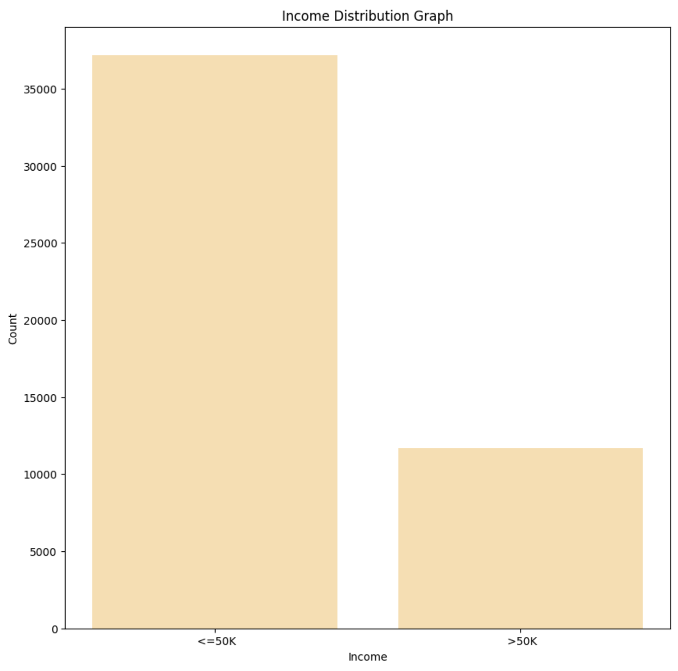






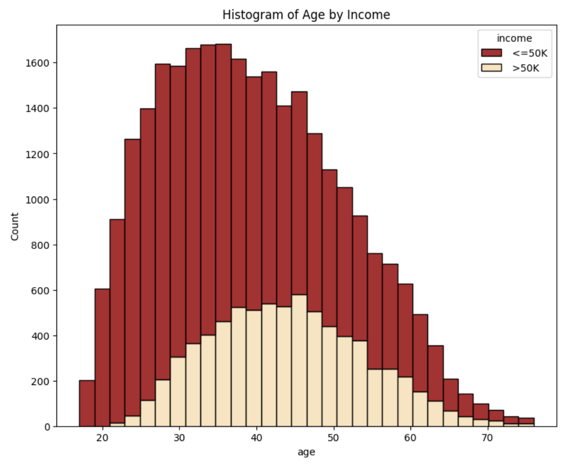
Box plots provide insights into the relationship between different numerical factors and income categories. By examining how these numerical characteristics are distributed for income levels below or above $50K, one can identify patterns that may be predictive.

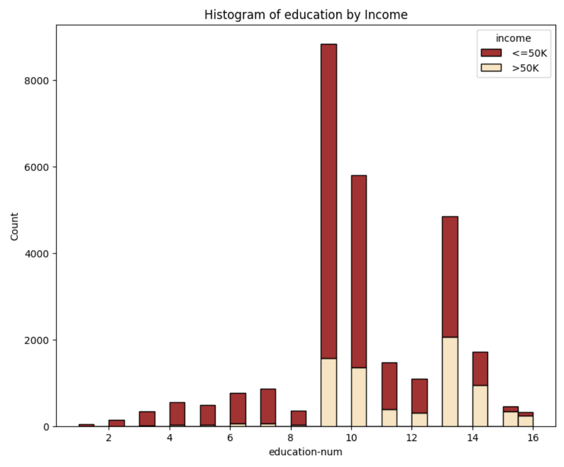
INCOME DISTRIBUTION

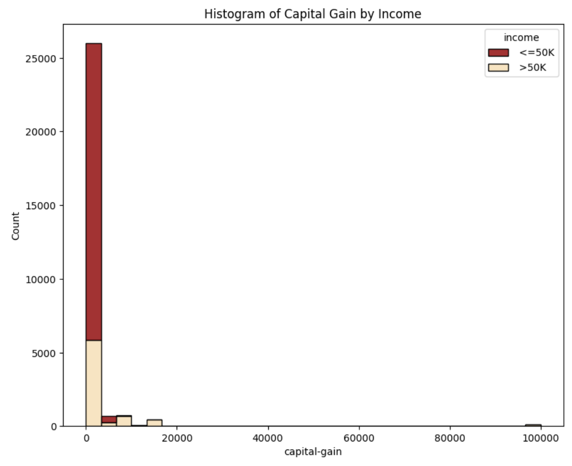


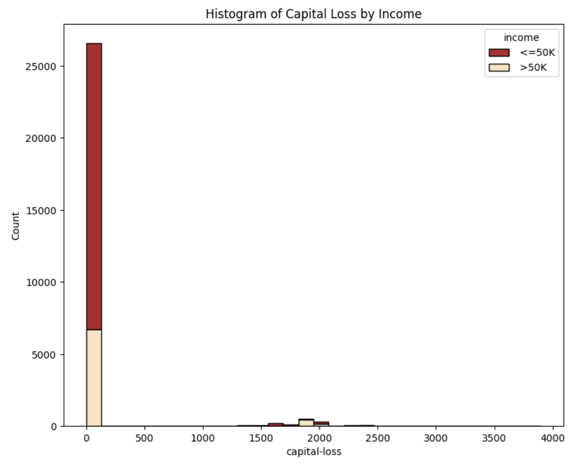
“Income Distribution Graph” compares the number of individuals with incomes less than or equal to 50K and those with incomes greater than 50K.

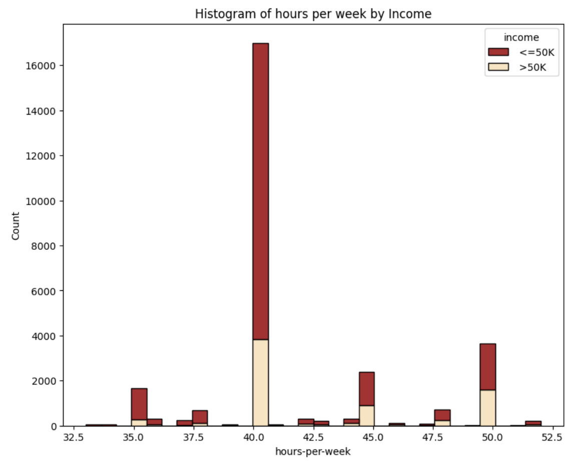
HISTOGRAMS FOR NUMIRICAL FEATURES





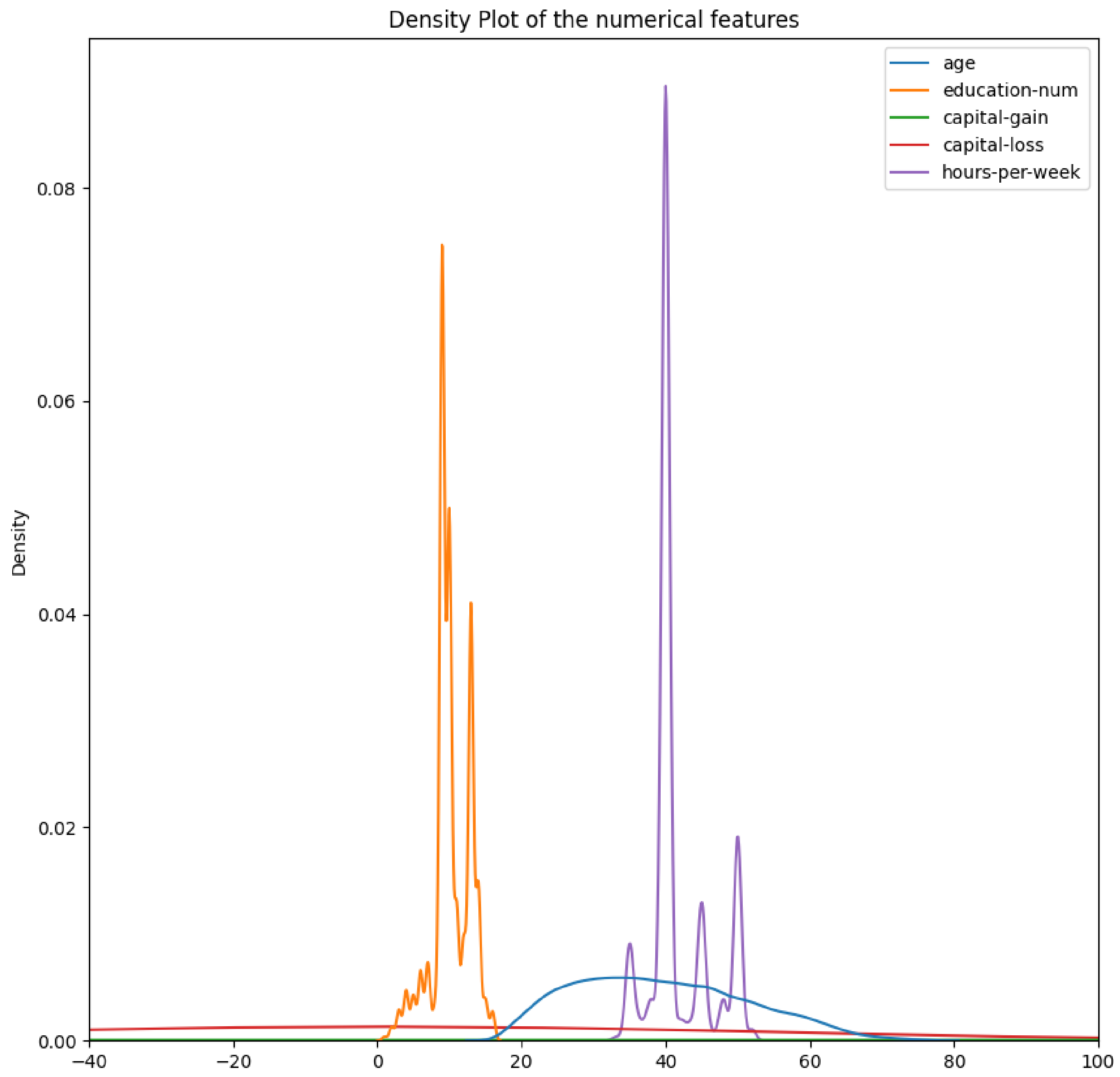






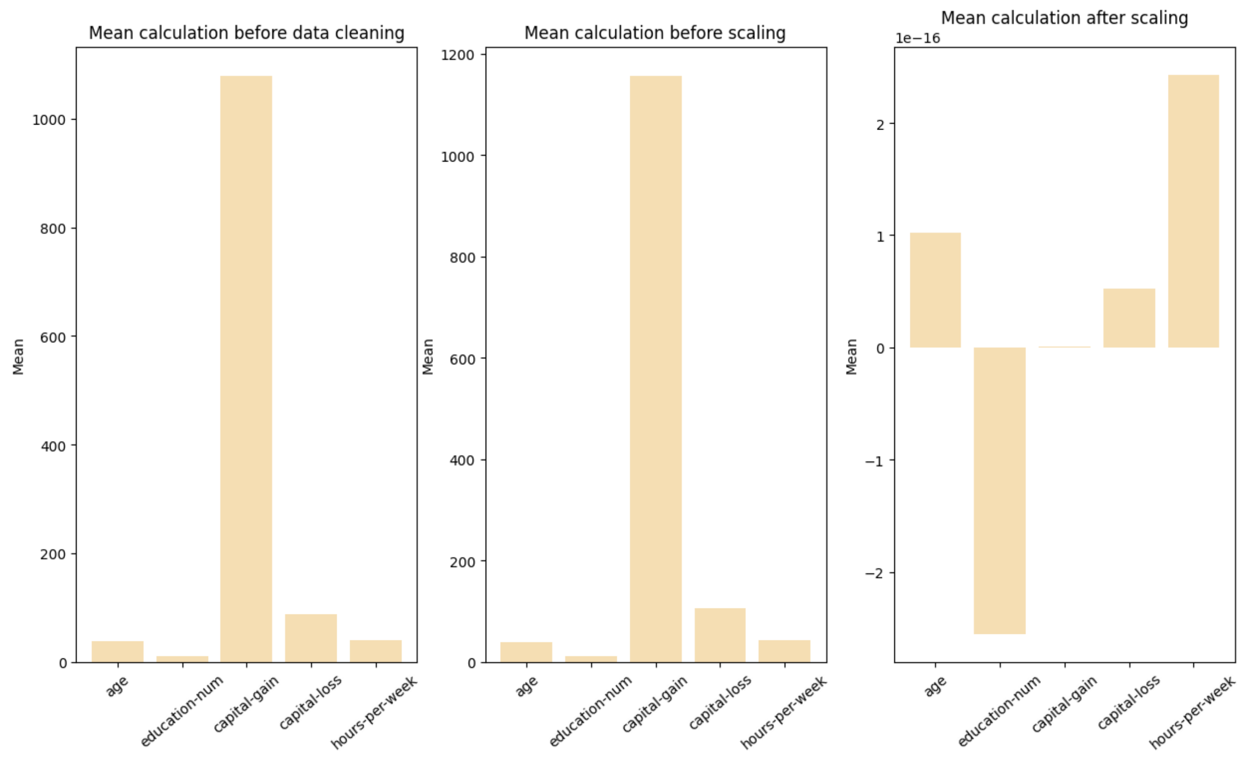
The histograms contrast income levels below and over $50,000 by showing the distribution of age, education, and work hours. Lower income workers tend to be younger and have higher levels of education, whereas higher earners are more evenly distributed in terms of age and full-time work hours. Low investment activity is suggested by both groups' negligible capital gains and losses. These trends might be useful in predicting income brackets.

DENSITY PLOT FOR NUMERICAL FEATURES



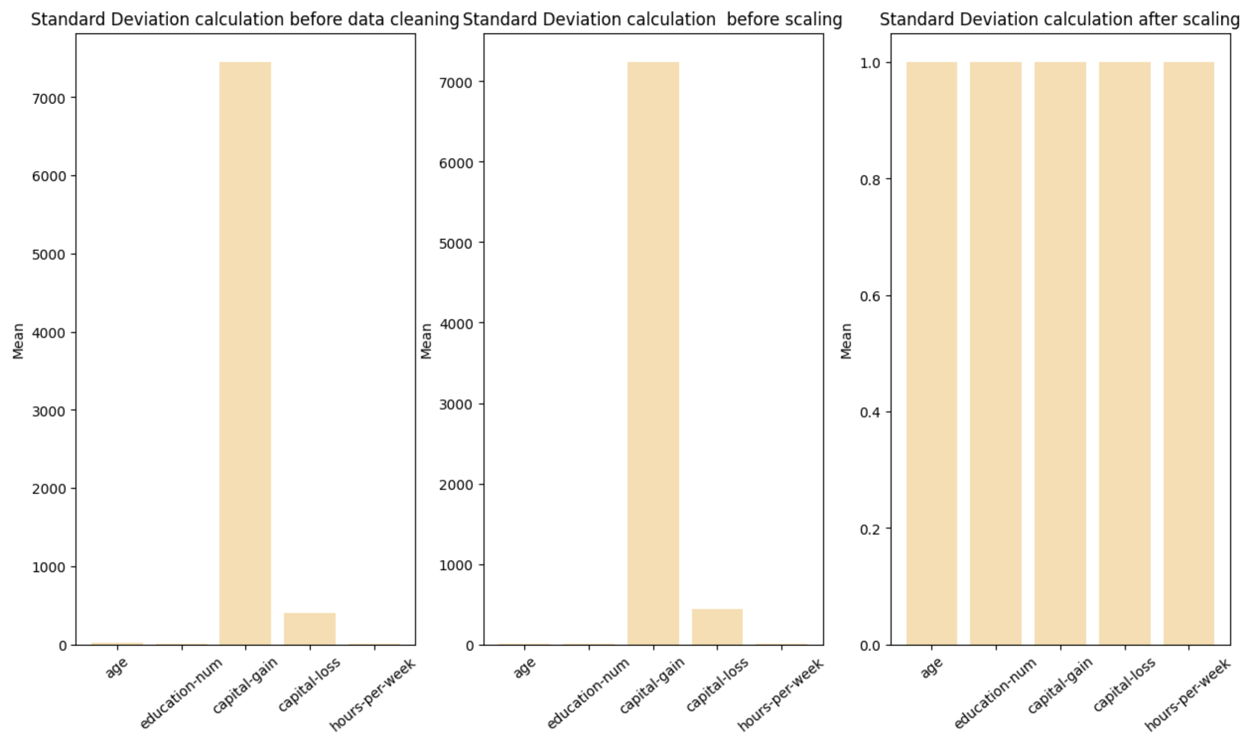
The distribution patterns across a range of values are displayed in the density plot for the numerical attributes in the dataset. Comparative study of the distributions of the variables is made possible by the overlaying of several variables. Plotting peaks indicate regions of concentrated data, whereas flatter parts indicate lower variability in those variables.

COMPARISON OF MEAN AND STANDARD DEVIATION PRIOR TO AND FOLLOWING DATA CLEANING AND SCALING



Three bar graphs that compare the mean values of various variables before and after data cleaning and scaling. These graphs are essential for understanding the impact of preprocessing steps on the dataset.

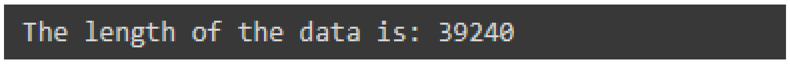
COMPARISON OF STANDARD DEVIATION AND STANDARD DEVIATION PRIOR TO AND FOLLOWING DATA CLEANING AND SCALING



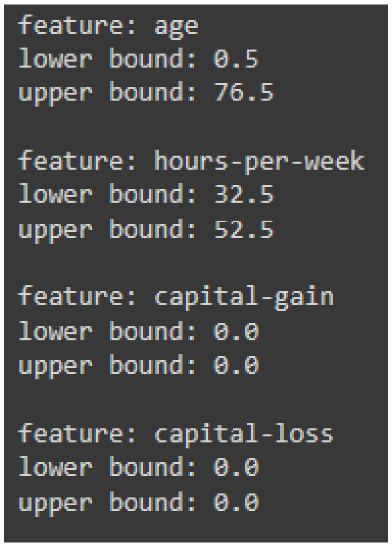
Three bar graphs that compare the standard deviation of different data attributes before cleaning, before scaling, and after scaling. The graphs highlight the significant reduction in variation after scaling, emphasizing the importance of data preprocessing in statistical analysis and machine learning.

HANDLE OUTLIERS

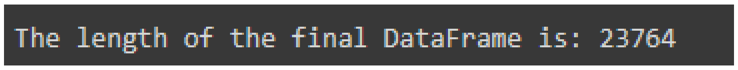
Calculate the interquartile range (IQR) for a specified column, define bounds to identify outliers, and filter the DataFrame to exclude them. Function is applied to multiple columns, effectively cleaning the data by removing statistical anomalies, which is crucial for accurate data analysis. After that display the number of rows remaining in the DataFrame after outlier removal.



```
The length of the data is: 39240
```



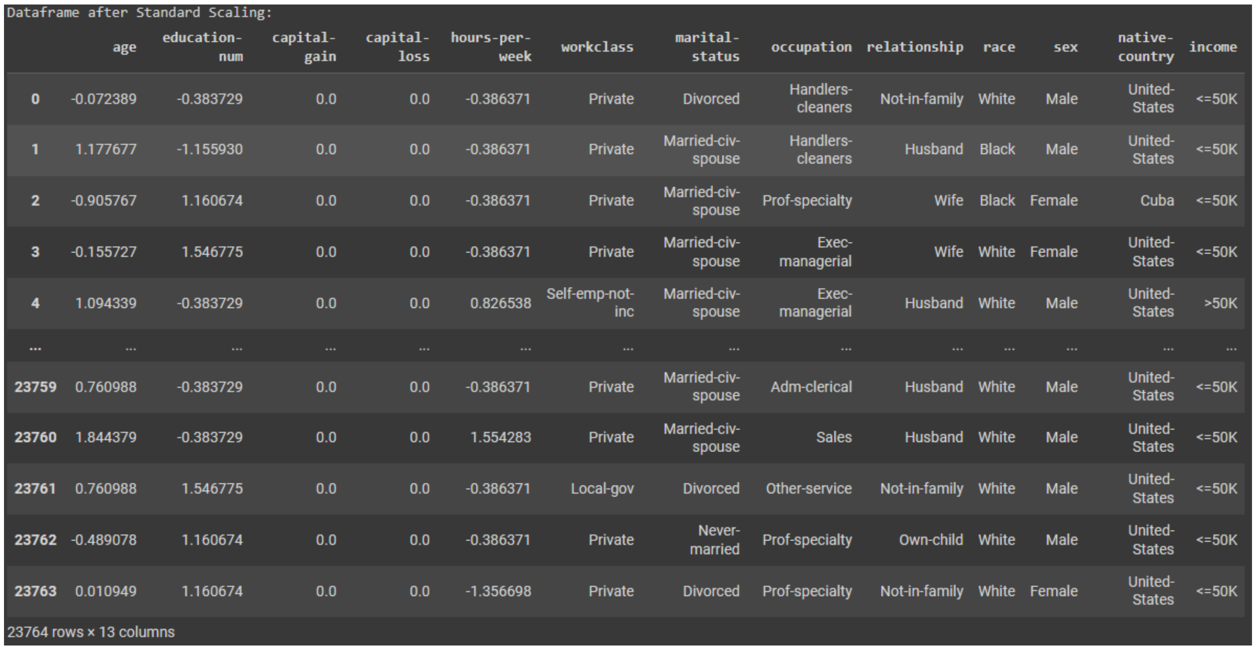
```
feature: age
lower bound: 0.5
upper bound: 76.5

feature: hours-per-week
lower bound: 32.5
upper bound: 52.5

feature: capital-gain
lower bound: 0.0
upper bound: 0.0

feature: capital-loss
lower bound: 0.0
upper bound: 0.0
```



```
The length of the final DataFrame is: 23764
```

DATA SCALING

To normalize numerical data inside the cleaned_data DataFrame, initialize a StandardScaler. The data will be scaled to have a zero mean and a one standard deviation, which is essential for models that require data to be normally distributed. The final_dataframe is subsequently created by combining the scaled numerical data with the initial category data.



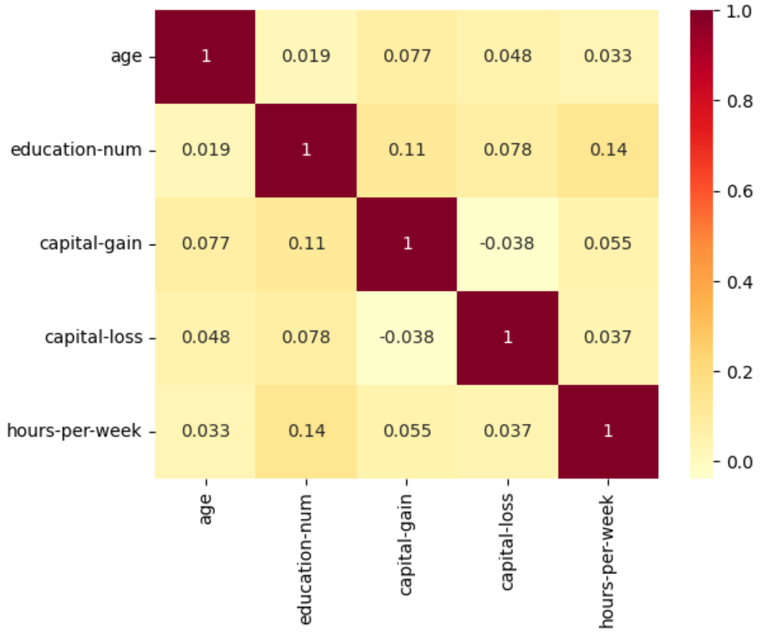
Dataframe after Standard Scaling:

	age	education-num	capital-gain	capital-loss	hours-per-week	workclass	marital-status	occupation	relationship	race	sex	native-country	income
0	-0.072389	-0.383729	0.0	0.0	-0.386371	Private	Divorced	Handlers-cleaners	Not-in-family	White	Male	United-States	<=50K
1	1.177677	-1.155930	0.0	0.0	-0.386371	Private	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	United-States	<=50K
2	-0.905767	1.160674	0.0	0.0	-0.386371	Private	Married-civ-spouse	Prof-specialty	Wife	Black	Female	Cuba	<=50K
3	-0.155727	1.546775	0.0	0.0	-0.386371	Private	Married-civ-spouse	Exec-managerial	Wife	White	Female	United-States	<=50K
4	1.094339	-0.383729	0.0	0.0	0.826538	Self-emp-not-inc	Married-civ-spouse	Exec-managerial	Husband	White	Male	United-States	>50K
...	...	...	...	...	...	...	...	...	...	...	...	...	...
23759	0.760988	-0.383729	0.0	0.0	-0.386371	Private	Married-civ-spouse	Adm-clerical	Husband	White	Male	United-States	<=50K
23760	1.844379	-0.383729	0.0	0.0	1.554283	Private	Married-civ-spouse	Sales	Husband	White	Male	United-States	<=50K
23761	0.760988	1.546775	0.0	0.0	-0.386371	Local-gov	Divorced	Other-service	Not-in-family	White	Male	United-States	<=50K
23762	-0.489078	1.160674	0.0	0.0	-0.386371	Private	Never-married	Prof-specialty	Own-child	White	Male	United-States	<=50K
23763	0.010949	1.160674	0.0	0.0	-1.356698	Private	Divorced	Prof-specialty	Not-in-family	White	Female	United-States	<=50K

23764 rows × 13 columns

CHECK FOR CORRELATION

Create a correlation matrix of the final_dataframe and visualizes it using a heatmap. Then selects the upper triangle of the correlation matrix to focus on pairwise correlations once. Features with a correlation coefficient greater than 0.8 are considered highly correlated and are listed to be dropped. These features are then removed from the DataFrame.

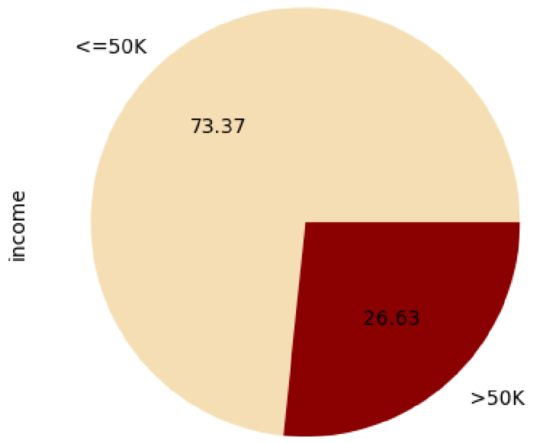


X,y SPLITTING & ENCODING CATEGORICAL FEATURES

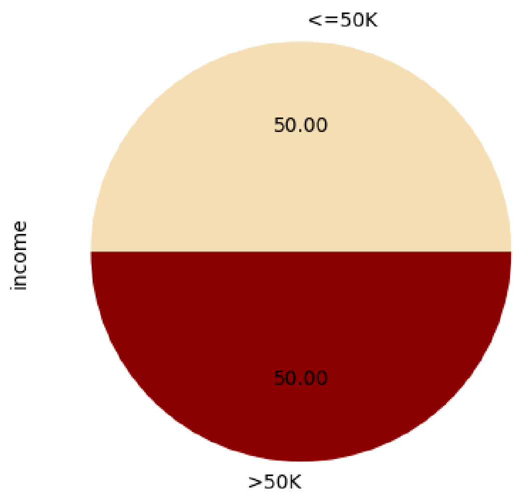
The DataFrame ‘final_dataframe’ is divided into two sections: y, which only includes the feature "income," and x, which includes all features other than "income." One-hot encoding is then used to convert the categorical features specified in label_categorical into numerical representation, resulting in binary columns for each category. To create the encoded_features DataFrame, the original categorical columns are removed from X and the newly one-hot encoded columns are concatenated back together. For machine learning algorithms that require numerical input, this translation is crucial.

DATA BALANCING

Visualize the class distribution in the dataset with a pie chart and then apply SMOTE to balance the classes. After resampling, a second pie chart shows the new, balanced distribution.



Before Resampling



After Resampling

Solution Methodology

Naïve Bayes Classifier

This simple probabilistic classifier assumes that each feature is independent and works using the ideas of the Bayes theorem. It assumes that the likelihood of features inside a class has a Gaussian distribution. It predicts the class with the highest probability by computing the likelihood of each class given the provided features. Gaussian Naïve Bayes frequently works effectively despite its fundamental assumptions of simplicity and feature independence, particularly when working with features that have continuous values. The independence of the features in the dataset is the basis for the selection of Gaussian Naïve Bayes.

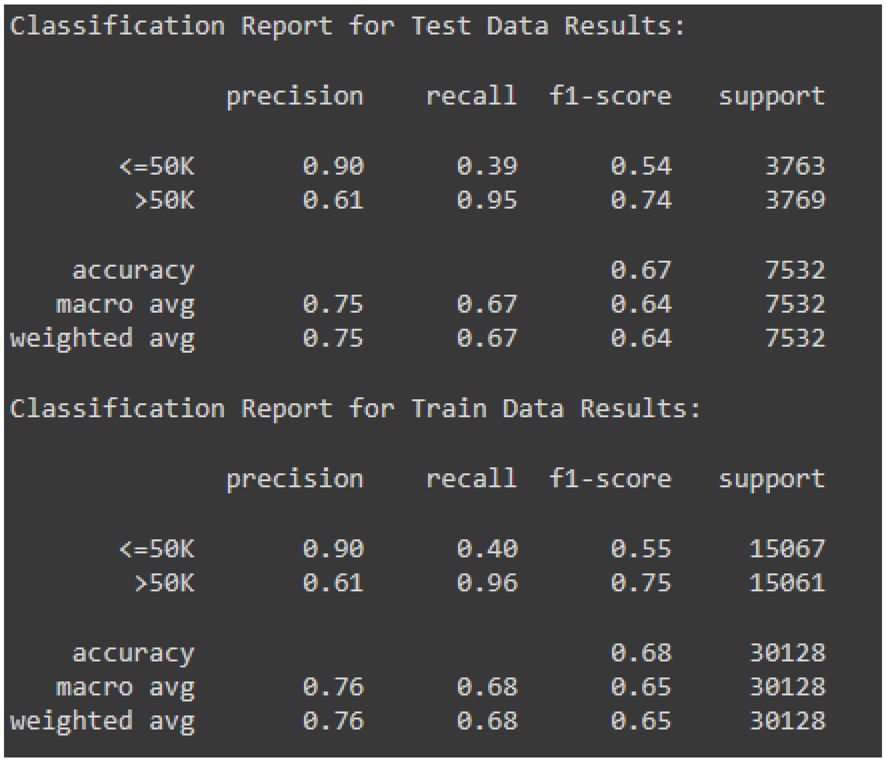
Random Forest Classifier

Random Forest is an ensemble method that builds multiple decision trees during training and aggregates their predictions to enhance accuracy and reduce overfitting. Each tree is trained on a random sample of data and features, allowing the model to capture complex, nonlinear relationships. The method averages the predictions from all trees, which mitigates overfitting and makes the model robust to outliers. By using a fixed random state, such as 42, the model's results are consistent and reproducible.

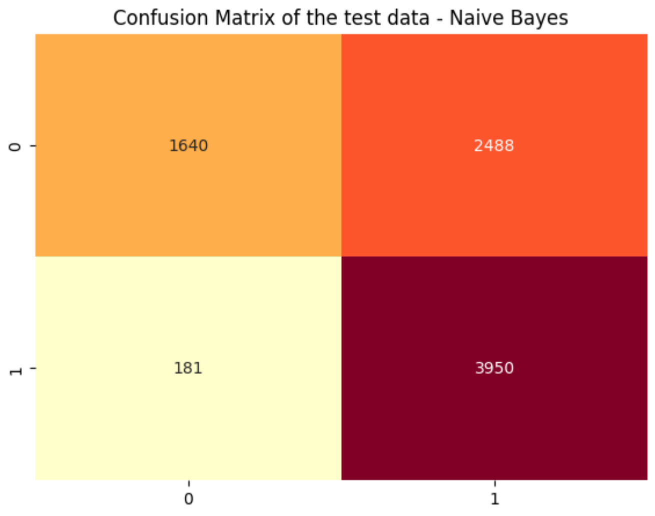
Model Evaluation

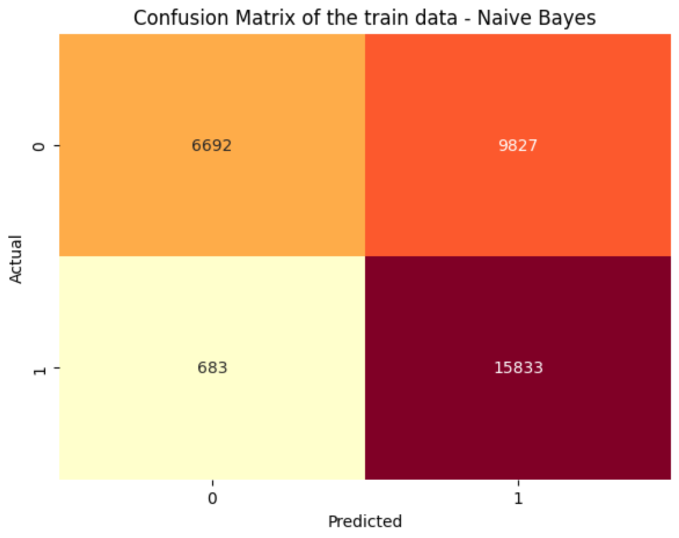
Naïve Bayes Classifier Results

Classification report:



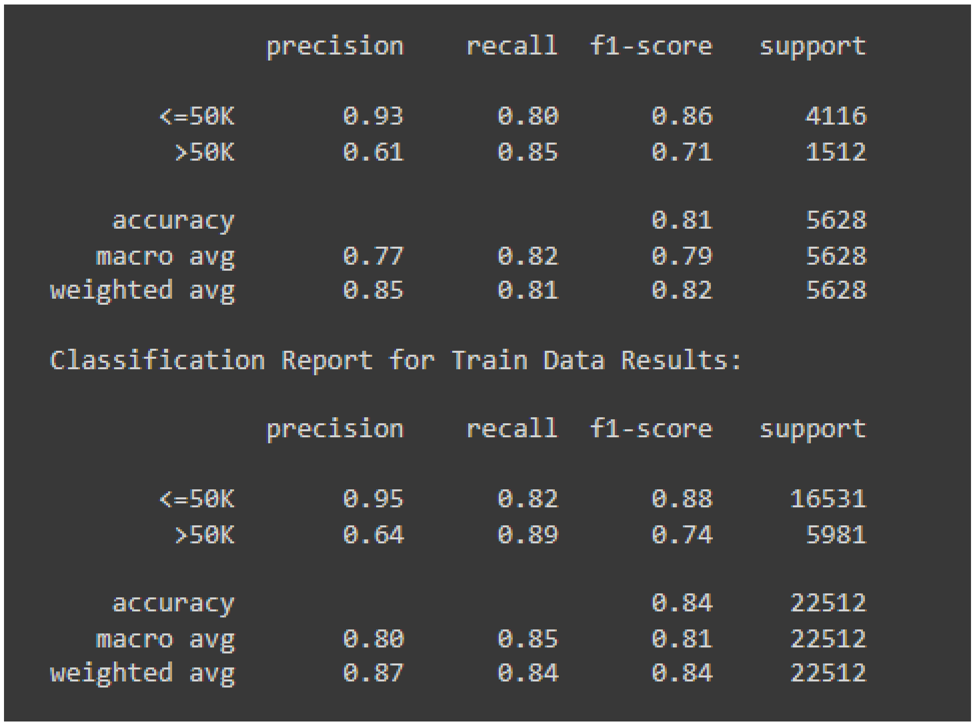
```
Classification Report for Test Data Results:

              precision    recall  f1-score   support

       <=50K       0.90      0.39      0.54      3763
        >50K       0.61      0.95      0.74      3769

    accuracy                           0.67      7532
   macro avg       0.75      0.67      0.64      7532
weighted avg       0.75      0.67      0.64      7532

Classification Report for Train Data Results:

              precision    recall  f1-score   support

       <=50K       0.90      0.40      0.55     15067
        >50K       0.61      0.96      0.75     15061

    accuracy                           0.68     30128
   macro avg       0.76      0.68      0.65     30128
weighted avg       0.76      0.68      0.65     30128
```



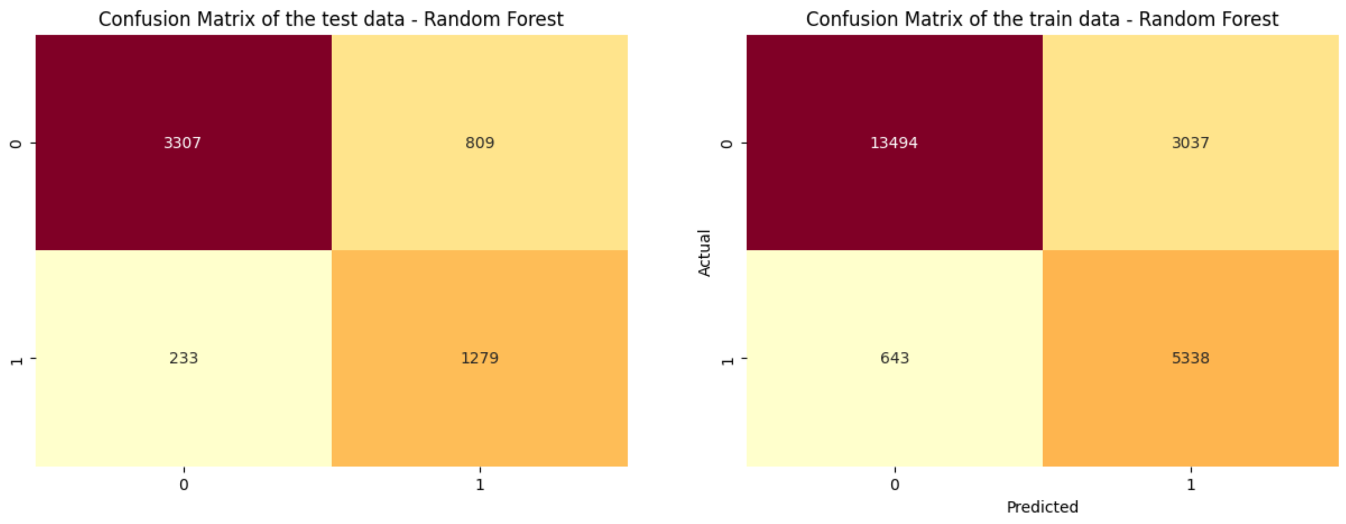


Random Forest Classifier Results

Classification Report:



```
              precision    recall  f1-score   support

       <=50K       0.93      0.80      0.86      4116
        >50K       0.61      0.85      0.71      1512

    accuracy                           0.81      5628
   macro avg       0.77      0.82      0.79      5628
weighted avg       0.85      0.81      0.82      5628

Classification Report for Train Data Results:

              precision    recall  f1-score   support

       <=50K       0.95      0.82      0.88     16531
        >50K       0.64      0.89      0.74      5981

    accuracy                           0.84     22512
   macro avg       0.80      0.85      0.81     22512
weighted avg       0.87      0.84      0.84     22512
```



Summary

This evaluation highlights key distinctions in the performance of Gaussian Naïve Bayes (GNB) and Random Forest models. While GNB demonstrates a high true positive rate (recall) for the ">50K" class, indicating effectiveness in identifying the majority of positive instances, it suffers from a lower positive predictive value (precision). This suggests an inclination towards misclassifying negative examples as positive.

Conversely, the Random Forest model exhibits a more balanced performance. It achieves a strong negative predictive value (precision) particularly for the "<=50K" class, signifying greater accuracy in predicting negative instances and subsequently leading to a higher overall classification accuracy. Additionally, Random Forest displays lower susceptibility to overfitting, as evidenced by the closer alignment of performance metrics between the training and test datasets compared to GNB.

In conclusion, for this specific classification task, the Random Forest model emerges as the more effective and reliable choice due to its superior balance, accuracy, and generalizability.

Limitations

- **Independence Assumption(Naïve Bayes):** It assumes that all features are independent of each other.
- **Complexity and Interpretability:** Random Forest models can become complex and are often considered "black boxes," making them difficult to interpret compared to simpler models.

.

Further Enhancements

- **Bagging and Boosting:** Explore techniques like bagging (where multiple Random Forests are trained on different subsets of data) or boosting (where models are trained sequentially, focusing on previously misclassified instances) to potentially improve overall accuracy and robustness.
- **ROC Curve and AUC:** Utilize the ROC curve and AUC (Area Under the Curve) to evaluate model performance across different classification thresholds.
- **L1 and L2 Regularization:** Implement L1 or L2 regularization to prevent overfitting by penalizing models with high parameter complexity.

Code

The GitHub Repo Link: https://github.com/GouriNapeVithanage/Census-Income-Prediction.git

# IMPORTS

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split, cross_validate
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score
from imblearn.over_sampling import SMOTE
from imblearn.combine import SMOTEENN
from imblearn.pipeline import Pipeline
from sklearn.model_selection import cross_val_score
```

# DATA PREPARATION

```
columns = ['age','workclass','fnlwgt','education','education-
num','marital-status', 'occupation', 'relationship','race', 'sex',
'capital-gain', 'capital-loss', 'hours-per-week', 'native-country',
'income']
#Load adult.data
ad = pd.read_csv('adult.data', names=columns, na_values=' ?')
#Load adult.test
adt = pd.read_csv('adult.test', names=columns, na_values=' ?', skiprows=1)
#Remove periods and dots from income column
adt.loc[:,'income']=adt['income'].str.replace('.','',regex=True)
#Combine Datasets
combined_dataframe= pd.concat([ad, adt], ignore_index=True)
combined_dataframe.reset_index(drop=True,inplace=True)
#Display
combined_dataframe
```

```
#Display the dimensions of the dataframe
print(f'Number of rows and columns in the
dataframe:{combined_dataframe.shape}')
```

```
#Display the summary of information about the dataframe
summary_cdf = pd.DataFrame({'Total count':
combined_dataframe.shape[0],'Non-Null Count':
combined_dataframe.count(),'Data type': combined_dataframe.dtypes})
summary_cdf
```

```
#Display the count of each unique value in 'income' column
income_count =combined_dataframe['income'].value_counts()
income_count
```

```
#Count the total number of values in the 'occupation' column
num_occupation =combined_dataframe['occupation'].value_counts().sum()
# Print the total number of values in the 'occupation' column
print(f'Number of values in the occupation:{num_occupation}')
#Display the count of each unique value in the 'occupation' column
combined_dataframe['occupation'].value_counts()
```

```
#Display the count of each unique value in 'native-country' column
num_native_country =combined_dataframe['native-
country'].value_counts().sum()
print(f'Number of values in the native country:{num_native_country}')
#Display the count of each unique value in the 'native-country' column
combined_dataframe['native-country'].value_counts()
```

```
#Display the count of each unique value in 'workclass' column
num_workclass =combined_dataframe['workclass'].value_counts().sum()
print(f'Number of values in the workclass:{num_workclass}')
#Display the count of each unique value in the 'workclass' column
combined_dataframe['workclass'].value_counts()
```

```
#Drop the 'education' and 'fnlwgt' columns from the dataframe
combined_dataframe = combined_dataframe.drop(['education', 'fnlwgt'], axis
= 1)
#Display the updated dataframe
combined_dataframe.head()
```

```
#Calculate and print the number of duplicated rows in the dataframe
print("Number of duplicated values: ",
combined_dataframe.duplicated().sum())
```

```
# Remove the duplicated rows from the dataframe
clean_data = combined_dataframe.drop_duplicates()
#Calculate and print the number of duplicated rows in the cleaned
dataframe
print("Number of duplicated values after data cleaning: ",
clean_data.duplicated().sum())
```

```
#Display the updated dataframe
clean_data
```

```
#Check for missing data in the entire dataframe
null_cd = clean_data.isnull().sum()
null_cd
```

```
#Remove rows with missing values from the dataframe
cleaned_data = clean_data.dropna()
#Print the total number of rows in the original dataframe
print(f'Number of rows: {combined_dataframe.shape[0]}')
#Print the total number of rows after handling duplicates
print(f'Number of rows after duplication handling: {clean_data.shape[0]}')
#Print the total number of remaining rows after removing rows with missing
values
print(f'Number of remaining rows: {cleaned_data.shape[0]} ')
```

```
#Check for missing data in the entire dataframe
null_cdf = cleaned_data.isnull().sum()
null_cdf
```

```
#Reset index
cleaned_data.reset_index(drop=True, inplace=True)
#Display the updated dataframe
cleaned_data
```

```
#Display summary of the entire dataframe
summary_cd = pd.DataFrame({'Total count': cleaned_data.shape[0],'Non-Null
Count': cleaned_data.count(),'Data type': cleaned_data.dtypes})
summary_cd
```

# HANDLE OUTLIERS

```
#Print the total number of rows in the cleaned dataframe
print(f'\nThe length of the data is: {len(cleaned_data)}')
```

```
#Define a function to detect and remove outliers in a given feature of a
dataframe
def outliers(feature,dataframe):
  #Calculate Quartiles
  q1 = dataframe[feature].quantile(0.25)
  q3 = dataframe[feature].quantile(0.75)
  #Calculate the IQR
  IQR = q3 - q1
  #Define the bounds
  lower_bound = q1 - (1.5 * IQR)
  upper_bound = q3 + (1.5 * IQR)
  #Remove outliers
  #Only keep rows where the feature value is within the lower and upper
bounds
  dataframe = dataframe[(dataframe[feature] >= lower_bound) &
(dataframe[feature] < upper_bound)]

  print(f'\nfeature: {feature}')
  print(f'lower bound: {lower_bound}')
  print(f'upper bound: {upper_bound}')
  return dataframe #return the dataframe without outliers
```

```
#Call the outliers function for the specified feature
cleaned_data = outliers('age',cleaned_data)
cleaned_data = outliers('hours-per-week',cleaned_data)
```

```
#Print the length of the dataframe after handling outliers
print(f'\nThe length of the final DataFrame is: {len(cleaned_data)}')
```

# DATA SCALING

```
#Initialize the StandardScaler
data_scaler = StandardScaler()
#Fit & Transform the numeric columns
```

```
scaled_features =
data_scaler.fit_transform(cleaned_data.select_dtypes(include
=['int64','float64']))
#New dataframe with scaled features
scaled_dataframe
=pd.DataFrame(scaled_features,columns=cleaned_data.select_dtypes(include=[
'int64','float64']).columns)
#Concatenate the scaled data
final_dataframe =
pd.concat([scaled_dataframe,cleaned_data.select_dtypes(exclude=['int64','f
loat64']).reset_index(drop=True)],axis=1)
```

```
#Display the updated dataframe
print('Dataframe after Standard Scaling: ')
final_dataframe
```

# MEAN AND STANDARD DEVIATION CALCULATION

```
# Select numerical columns
columns = ['age', 'education-num', 'capital-gain', 'capital-loss', 'hours-
per-week']
#Mean calculation before data cleaning
mean_cdf = combined_dataframe[columns].mean()
#Mean calculation before scaling
mean_cd = cleaned_data[columns].mean()
#Mean calculation after scaling
mean_fdf = final_dataframe[columns].mean()
#Display
print('Mean calculation before data cleaning: ')
print(mean_cdf)
print('\nMean calculation before scaling: ')
print(mean_cd)
print('\nMean calculation before after scaling: ')
print(mean_fdf)
```

```
#Standard Deviation calculation before data cleaning
std_cdf = combined_dataframe[columns].std()
#Standard Deviation calculation  before scaling
std_cd = cleaned_data[columns].std()
#Standard Deviation calculation after scaling
std_fdf = final_dataframe[columns].std()
```

```
#Display
print('Standard Deviation calculation  before data cleaning: ')
print(std_cdf)
print('\nStandard Deviation calculation before scaling: ')
print(std_cd)
print('\nStandard Deviation calculation before after scaling: ')
print(std_fdf)
```

# DATA VISUALIZATION

```
# Plot income class distribution
plt.figure(figsize=(10,10))
#Create a bar plot with 'income_count.index' as x and
'income_count.values' as y
plt.bar(income_count.index, income_count.values, color ='wheat')
plt.xlabel('Income')
plt.ylabel('Count')
plt.title('Income Distribution Graph')
plt.show()
```

```
#Create a new figure with specified size
fig,ax =plt.subplots(figsize=(10,10))
#Generate a Kernel Density Estimate (KDE) plot for the cleaned data
sns.kdeplot(data=cleaned_data,ax=ax)
ax.set_xlim(-40,100) #Set the x-axis limits
plt.title('Density Plot of the numerical features')
plt.show()
```

```
#Define the income categories to filter the dataset
income_categories = [' <=50K', ' >50K']
#Filter the dataset based on the defined income categories
filtered_data =
cleaned_data[cleaned_data['income'].isin(income_categories)]

# Create a subplot
fig,ax =plt.subplots(nrows=3,ncols=2,figsize=(20, 25))

# Create a box plot of all numerical features in the cleaned data
sns.boxplot(ax=ax[0,0], data=cleaned_data,color ='wheat')
ax[0,0].set_title('Box Plot of Numerical Features')
ax[0,0].set_xlabel('Numerical Features')
```

```
#Create a box plot of 'age' / 'income'
sns.boxplot(x='income', y='age',ax=ax[0,1], data=filtered_data,color
='wheat')
ax[0,1].set_title('Box Plot of Age against Income')

#Create a box plot of 'education-num' against 'income'
sns.boxplot(x='income', y='education-num',ax=ax[1,0],
data=filtered_data,color ='wheat')
ax[1,0].set_title('Box Plot of education against Income')

#Create a box plot of 'capital-gain' against 'income'
sns.boxplot(x='income', y='capital-gain',ax=ax[1,1], data=filtered_data)
ax[1,1].set_title('Box Plot of Capital Gain against Income')

#Create a box plot of 'capital-loss' against 'income'
sns.boxplot(x='income', y='capital-loss',ax=ax[2,0], data=filtered_data)
ax[2,0].set_title('Box Plot of Capital Loss against Income')

#Create a box plot of 'hours-per-week' against 'income'
sns.boxplot(x='income', y='hours-per-week',ax=ax[2,1],
data=filtered_data,color ='wheat')
ax[2,1].set_title('Box Plot of hours per week against Income')

plt.show()
```

```
#Define the colors for the two income categories
upper50='wheat'
lower50='darkred'

#Create a subplot with 3 rows and 2 columns, and set the figure size
fig,ax =plt.subplots(nrows=3,ncols=2,figsize=(20, 25))

#Create a stacked histogram of 'age' by 'income'
sns.histplot(ax=ax[0,0],data=cleaned_data, x='age', hue='income', bins=30,
palette={' >50K': upper50, ' <=50K': lower50}, alpha=0.8,
multiple='stack')
ax[0,0].set_title('Histogram of Age by Income')

#Create a stacked histogram of 'education-num' by 'income'
sns.histplot(ax=ax[0,1],data=cleaned_data, x='education-num',
hue='income', bins=30, palette={' >50K': upper50, ' <=50K': lower50},
alpha=0.8, multiple='stack')
ax[0,1].set_title('Histogram of education by Income')
```

```
#Create a stacked histogram of 'capital-gain' by 'income'
sns.histplot(ax=ax[1,0],data=cleaned_data, x='capital-gain', hue='income',
bins=30, palette={' >50K': upper50, ' <=50K': lower50}, alpha=0.8,
multiple='stack')
ax[1,0].set_title('Histogram of Capital Gain by Income')

#Create a stacked histogram of 'capital-loss' by 'income'
sns.histplot(ax=ax[1,1],data=cleaned_data, x='capital-loss', hue='income',
bins=30, palette={' >50K': upper50, ' <=50K': lower50}, alpha=0.8,
multiple='stack')
ax[1,1].set_title('Histogram of Capital Loss by Income')

#Create a stacked histogram of 'hours-per-week' by 'income'
sns.histplot(ax=ax[2,0],data=cleaned_data, x='hours-per-week',
hue='income', bins=30, palette={' >50K': upper50, ' <=50K': lower50},
alpha=0.8, multiple='stack')
ax[2,0].set_title('Histogram of hours per week by Income')

#remove the empty subplot
fig.delaxes(ax[2,1])

plt.show()
```

```
#Create a subplot with 1 row and 3 columns, and set the figure size
fig, axs = plt.subplots(ncols=3, figsize=(15, 8))

#Create a bar plot of the mean values before data cleaning
axs[0].bar(range(len(mean_cdf)), mean_cdf,color='wheat')
axs[0].set_title("Mean calculation before data cleaning")
axs[0].set_ylabel("Mean")
axs[0].set_xticks(range(len(mean_cdf)))
axs[0].set_xticklabels(combined_dataframe[columns].columns, rotation=40)

#Create a bar plot of the mean values before scaling
axs[1].bar(range(len(mean_cd)), mean_cd,color='wheat')
axs[1].set_title("Mean calculation before scaling")
axs[1].set_ylabel("Mean")
axs[1].set_xticks(range(len(mean_cd)))
axs[1].set_xticklabels(combined_dataframe[columns].columns, rotation=40)

#Create a bar plot of the mean values after scaling
```

```
axs[2].bar(range(len(mean_fdf)), mean_fdf,color='wheat')
axs[2].set_title("Mean calculation after scaling")
axs[2].set_ylabel("Mean")
axs[2].set_xticks(range(len(mean_fdf)))
axs[2].set_xticklabels(combined_dataframe[columns].columns, rotation=40)

plt.show()
```

```
#Create a subplot with 1 row and 3 columns, and set the figure size
fig, axs = plt.subplots(ncols=3, figsize=(15, 8))

#Create a bar plot of the mean values before data cleaning
axs[0].bar(range(len(mean_cdf)), mean_cdf,color='wheat')
axs[0].set_title("Mean calculation before data cleaning")
axs[0].set_ylabel("Mean")
axs[0].set_xticks(range(len(mean_cdf)))
axs[0].set_xticklabels(combined_dataframe[columns].columns, rotation=40)

#Create a bar plot of the mean values before scaling
axs[1].bar(range(len(mean_cd)), mean_cd,color='wheat')
axs[1].set_title("Mean calculation before scaling")
axs[1].set_ylabel("Mean")
axs[1].set_xticks(range(len(mean_cd)))
axs[1].set_xticklabels(combined_dataframe[columns].columns, rotation=40)

#Create a bar plot of the mean values after scaling
axs[2].bar(range(len(mean_fdf)), mean_fdf,color='wheat')
axs[2].set_title("Mean calculation after scaling")
axs[2].set_ylabel("Mean")
axs[2].set_xticks(range(len(mean_fdf)))
axs[2].set_xticklabels(combined_dataframe[columns].columns, rotation=40)

plt.show()


#Create a subplot with 1 row and 3 columns, and set the figure size
fig, axs = plt.subplots(ncols=3, figsize=(15, 8))

#Create a bar plot of the Standard Deviation values before data cleaning
axs[0].bar(range(len(std_cdf)), std_cdf,color='wheat')
axs[0].set_title("Standard Deviation calculation before data cleaning")
```

```
axs[0].set_ylabel("Mean")
axs[0].set_xticks(range(len(std_cdf)))
axs[0].set_xticklabels(combined_dataframe[columns].columns, rotation=40)

#Create a bar plot of the Standard Deviation values before scaling
axs[1].bar(range(len(std_cd)), std_cd,color='wheat')
axs[1].set_title("Standard Deviation calculation  before scaling")
axs[1].set_ylabel("Mean")
axs[1].set_xticks(range(len(std_cd)))
axs[1].set_xticklabels(combined_dataframe[columns].columns, rotation=40)

#Create a bar plot of the Standard Deviation values after scaling
axs[2].bar(range(len(std_fdf)), std_fdf,color='wheat')
axs[2].set_title("Standard Deviation calculation after scaling")
axs[2].set_ylabel("Mean")
axs[2].set_xticks(range(len(std_fdf)))
axs[2].set_xticklabels(combined_dataframe[columns].columns, rotation=40)

plt.show()
```

# CHECK FOR CORRELATION

```
#Compute the correlation matrix for the final dataframe
Correlation_matrix=final_dataframe.corr()
Correlation_matrix
```

```
#Plot the heatmap for correlation matrix
sns.heatmap(Correlation_matrix,annot=True,cmap='YlOrRd')
plt.show()
```

```
# Select upper triangle of correlation matrix
upper_tri =
Correlation_matrix.where(np.triu(np.ones(Correlation_matrix.shape),
k=1).astype(bool))
# Find index of feature columns with correlation greater than a threshold
to_drop = [column for column in upper_tri.columns if any(upper_tri[column]
> 0.8)]

print(f'The following columns are highly correlated: {to_drop}')

#Remove highly correlated features
final_dataframe = final_dataframe.drop(to_drop, axis=1)
```

# X y SPLITTING & ENCODING CATEGORICAL FEATURES

```
#Drop the 'income' column from the final dataframe to create the feature
set
X = final_dataframe.drop('income', axis=1)
#Use the 'income' column as the target variable
y = final_dataframe['income']
```

```
#Define the categorical features
label_categorical = ['workclass', 'marital-status', 'occupation',
'relationship', 'race', 'sex', 'native-country']
#Use one-hot encoding for the categorical features
X_categorical = pd.get_dummies(X[label_categorical], drop_first=True)
#Concatenate the original dataframe (without the categorical features)
with the one-hot encoded dataframe
encoded_features = pd.concat([X.drop(label_categorical, axis=1),
X_categorical], axis=1)
```

# DATA BALANCING

```
#Count the frequency of the unique values
y.value_counts()
```

```
#Plot the values
colors=['wheat', 'darkred']
y.value_counts().plot.pie(colors=colors,autopct='%.2f')
```

```
#Initialize Synthetic Minority Oversampling Technique(SMOT)
smote = SMOTE()
#Fit the smot to the data
X_resampled, y_resampled = smote.fit_resample(encoded_features, y)
#Plot the values
ax = y_resampled.value_counts().plot.pie(colors=colors,autopct='%.2f')
```

# NAIVE BAYES CLASSIFICATION MODEL

```
#Split the resampled data into training and test sets(The test will be 20%
of total data)
X_train, X_test, y_train, y_test = train_test_split(X_resampled,
y_resampled, test_size=0.2)
```

```
#Instantiate a Gaussian Naive Bayes classifier
naive_bayes = GaussianNB()
#Fit the classifier to the training data
naive_bayes.fit(X_train, y_train)
```

```
#Naive Bayes classifier to predict the target variable for the test data
y_pred_tnb = naive_bayes.predict(X_test)
print(f'Classification Report for Test Data
Results:\n\n{classification_report(y_test,y_pred_tnb)}')

#Naive Bayes classifier to predict the target variable for the training
data
y_pred_Tnb = naive_bayes.predict(X_train)
print(f'Classification Report for Train Data
Results:\n\n{classification_report(y_train,y_pred_Tnb)}')
```

```
#Compute the confusion matrix for the test data
conf_mat = confusion_matrix(y_test, y_pred_tnb)

#Create a subplot with 1 row and 2 columns, and set the figure size
fig, (ax1, ax2) = plt.subplots(ncols=2,figsize=(15,5))

#Create a heatmap of the confusion matrix for the test data
sns.heatmap(conf_mat, annot=True, fmt="d",
cbar=False,ax=ax1,cmap='YlOrRd')
ax1.set_title('Confusion Matrix of the test data - Naive Bayes')

#Compute the confusion matrix for the training data
conf_matrix = confusion_matrix(y_train, y_pred_Tnb)

#Create a heatmap of the confusion matrix for the training data
sns.heatmap(conf_matrix, annot=True, fmt="d",
cbar=False,ax=ax2,cmap='YlOrRd')
ax2.set_title('Confusion Matrix of the train data - Naive Bayes')


plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

# RANDOM FOREST CLASSIFICATION

```
#Split the encoded features and target variable into training and test
sets.The 'random_state' parameter ensures that the splits generate are
reproducible
X_train, X_test, y_train, y_test = train_test_split(encoded_features, y,
test_size=0.2, random_state=42)

random_forest = Pipeline([
    ('sampling', SMOTEENN(random_state=42)),
    ('classifier', RandomForestClassifier(random_state=42,
n_estimators=100))
])
```

```
random_forest.fit(X_train, y_train)
```

```
y_pred_trf = random_forest.predict(X_test)
print(f'Classification Report for Test Data
Results:\n\n{classification_report(y_test,y_pred_trf)}')

y_pred_Trf = random_forest.predict(X_train)
print(f'Classification Report for Train Data
Results:\n\n{classification_report(y_train,y_pred_Trf)}')
```

```
#Test data predict
con_matr = confusion_matrix(y_test, y_pred_trf)
#Plot the heatmap
fig, (ax1, ax2) = plt.subplots(ncols=2,figsize=(15,5))
sns.heatmap(con_matr, annot=True, fmt="d",
cbar=False,ax=ax1,cmap='YlOrRd')
ax1.set_title('Confusion Matrix of the test data - Random Forest')

#Training data predict
con_matri = confusion_matrix(y_train, y_pred_Trf)
#Plot the heatmap
sns.heatmap(con_matri, annot=True, fmt="d",
cbar=False,ax=ax2,cmap='YlOrRd')
ax2.set_title('Confusion Matrix of the train data - Random Forest')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

References

- www.youtube.com. (n.d.). Adult Sensus Income Kaggle Dataset Analysis | Kaggle | Who earns more than $50K/year ?? [online] Available at:. https://youtu.be/reVAGcwOxH8?si=aDL9SXw7NcoFXrmY
- GeeksforGeeks. (2019). ML | Handling Imbalanced Data with SMOTE and Near Miss Algorithm in Python. [online] Available at: https://www.geeksforgeeks.org/ml-handling-imbalanced-data-with-smote-and-near-miss-algorithm-in-python/.
- www.youtube.com. (n.d.). Machine Learning for Everybody – Full Course. [online] Available at: https://www.youtube.com/watch?v=i_LwzRVP7bg&t=6309s&pp=ygULbWwgbGVhcm5pbmc%3D [Accessed 24 Mar. 2024].
- Pydata.org. (2012). seaborn.kdeplot — seaborn 0.9.0 documentation. [online] Available at: https://seaborn.pydata.org/generated/seaborn.kdeplot.html.
- Stack Overflow. (n.d.). Multiple count plots in seaborn. [online] Available at: https://stackoverflow.com/questions/61000529/multiple-count-plots-in-seaborn[Accessed 29 Mar. 2024].
- matplotlib.org. (n.d.). List of named colors — Matplotlib 3.4.2 documentation. [online] Available at: https://matplotlib.org/stable/gallery/color/named_colors.html.
- Stack Overflow. (n.d.). Make the size of a heatmap bigger with seaborn. [online] Available at:https://stackoverflow.com/questions/38913965/make-the-size-of-a-heatmap-bigger-with-seaborn [Accessed 29 Mar. 2024].
- Stack Overflow. (n.d.). plot two seaborn heatmap graphs side by side. [online] Available at:https://stackoverflow.com/questions/50947776/plot-two-seaborn-heatmap-graphs-side-by-side [Accessed 29 Mar. 2024].
- www.w3schools.com. (n.d.). Matplotlib Subplot. [online] Available at: https://www.w3schools.com/python/matplotlib_subplot.asp.
- Brownlee, J. (2020). SMOTE for Imbalanced Classification with Python. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/.

- dashboards.mysidewalk.com. (n.d.). mySidewalk. [online] Available at: https://dashboards.mysidewalk.com/style-guide-for-dashboards/pie-charts.
- Gulati, A.P. (2022). Dealing with Outliers Using the IQR Method. [online] Analytics Vidhya. Available at: https://www.analyticsvidhya.com/blog/2022/09/dealing-with-outliers-using-the-iqr-method/.
- GeeksforGeeks. (2020). Interquartile Range to Detect Outliers in Data. [online] Available at: https://www.geeksforgeeks.org/interquartile-range-to-detect-outliers-in-data/.
- Bhandari, P. (2021). How to find and remove outliers. [online] Scribbr. Available at: https://www.scribbr.com/statistics/outliers/.
- Khan Academy (2016). Identifying outliers with the 1.5xIQR rule. [online] Khan Academy. Available at: https://www.khanacademy.org/math/statistics-probability/summarizing-quantitative-data/box-whisker-plots/a/identifying-outliers-iqr-rule.
- PennState: Statistics Online Courses. (n.d.). 3.2 - Identifying Outliers: IQR Method | STAT 200. [online] Available at: https://online.stat.psu.edu/stat200/lesson/3/3.2#:~:text=We%20can%20use%20the%20IQR.
- GeeksforGeeks. (2020). Seaborn Kdeplot – A Comprehensive Guide. [online] Available at: https://www.geeksforgeeks.org/seaborn-kdeplot-a-comprehensive-guide/?ref=lbp.
- GeeksforGeeks. (2017). Plotting graph using Seaborn | Python. [online] Available at: https://www.geeksforgeeks.org/plotting-graph-using-seaborn-python/.
- GeeksforGeeks. (2019). KDE Plot Visualization with Pandas and Seaborn. [online] Available at: https://www.geeksforgeeks.org/kde-plot-visualization-with-pandas-and-seaborn/.
- GeeksforGeeks. (2020). Density Plots with Pandas in Python. [online] Available at: https://www.geeksforgeeks.org/density-plots-with-pandas-in-python/.
- www.youtube.com. (n.d.). Random Forest Algorithm - Random Forest Explained | Random Forest in Machine Learning | Simplilearn. [online] Available at:https://www.youtube.com/watch?v=eM4uJ6XGnSM&ab_channel=Simplilearn [Accessed 29 Mar. 2024].

- www.youtube.com. (n.d.). Adult Sensus Income Kaggle Dataset Analysis | Kaggle | Who earns more than $50K/year ?? [online] Available at: https://www.youtube.com/watch?v=reVAGcwOxH8&ab_channel=AkshitMadan.
- www.youtube.com. (n.d.). Naive Bayes Classifier | Naive Bayes Algorithm | Naive Bayes Classifier With Example | Simplilearn. [online] Available at: https://www.youtube.com/watch?v=l3dZ6ZNFjo0&ab_channel=Simplilearn[Accessed 29 Mar. 2024].