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 4**](#_Toc162663619)

[**Corpus Preparation 5**](#_Toc162663620)

[**DATASET PEPARATION 5**](#_Toc162663621)

[**BOX PLOTS AGAINST INCOME FOR NUMERICAL FEATURES 10**](#_Toc162663622)

[**INCOME DISTRIBUTION 11**](#_Toc162663623)

[**HISTOGRAMS FOR NUMIRICAL FEATURES 12**](#_Toc162663624)

[**DENSITY PLOT FOR NUMERICAL FEATURES 13**](#_Toc162663625)

[**COMPARISON OF MEAN AND STANDARD DEVIATION PRIOR TO AND FOLLOWING DATA CLEANING AND SCALING 14**](#_Toc162663626)

[**COMPARISON OF STANDARD DEVIATION AND STANDARD DEVIATION PRIOR TO AND FOLLOWING DATA CLEANING AND SCALING 15**](#_Toc162663627)

[**HANDLE OUTLIERS 16**](#_Toc162663628)

[**DATA SCALING 16**](#_Toc162663629)

[**CHECK FOR CORRELATION 17**](#_Toc162663630)

[**X,y SPLITTING & ENCODING CATEGORICAL FEATURES 18**](#_Toc162663631)

[**DATA BALANCING 19**](#_Toc162663632)

[**Solution Methodology 20**](#_Toc162663633)

[**Naïve Bayes Classifier 20**](#_Toc162663634)

[**Random Forest Classifier 20**](#_Toc162663635)

[**Model Evaluation 21**](#_Toc162663636)

[**Naïve Bayes Classifier Results 21**](#_Toc162663637)

[**Random Forest Classifier Results 22**](#_Toc162663638)

[**Summary 23**](#_Toc162663639)

[**Limitations 24**](#_Toc162663640)

[**Further Enhancements 25**](#_Toc162663641)

[**Code 26**](#_Toc162663642)

[**References 39**](#_Toc162663643)

# 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.

A screenshot of a computer screen

Description automatically generated

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



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.

A screenshot of a computer

Description automatically generated

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

A screenshot of a computer

Description automatically generated

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.

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

Remove the ‘education’ and ‘fnlwgt’ columns from combined\_dataframe because ‘education’ and ‘education\_num’ columns duplicate information.

A screenshot of a computer

Description automatically generated

Identify and print the count of duplicate rows in combined\_dataframe. Then, remove the duplicates to create clean\_data, ensuring the data is unique.



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.

A screenshot of a computer

Description automatically generated A screen shot of a computer

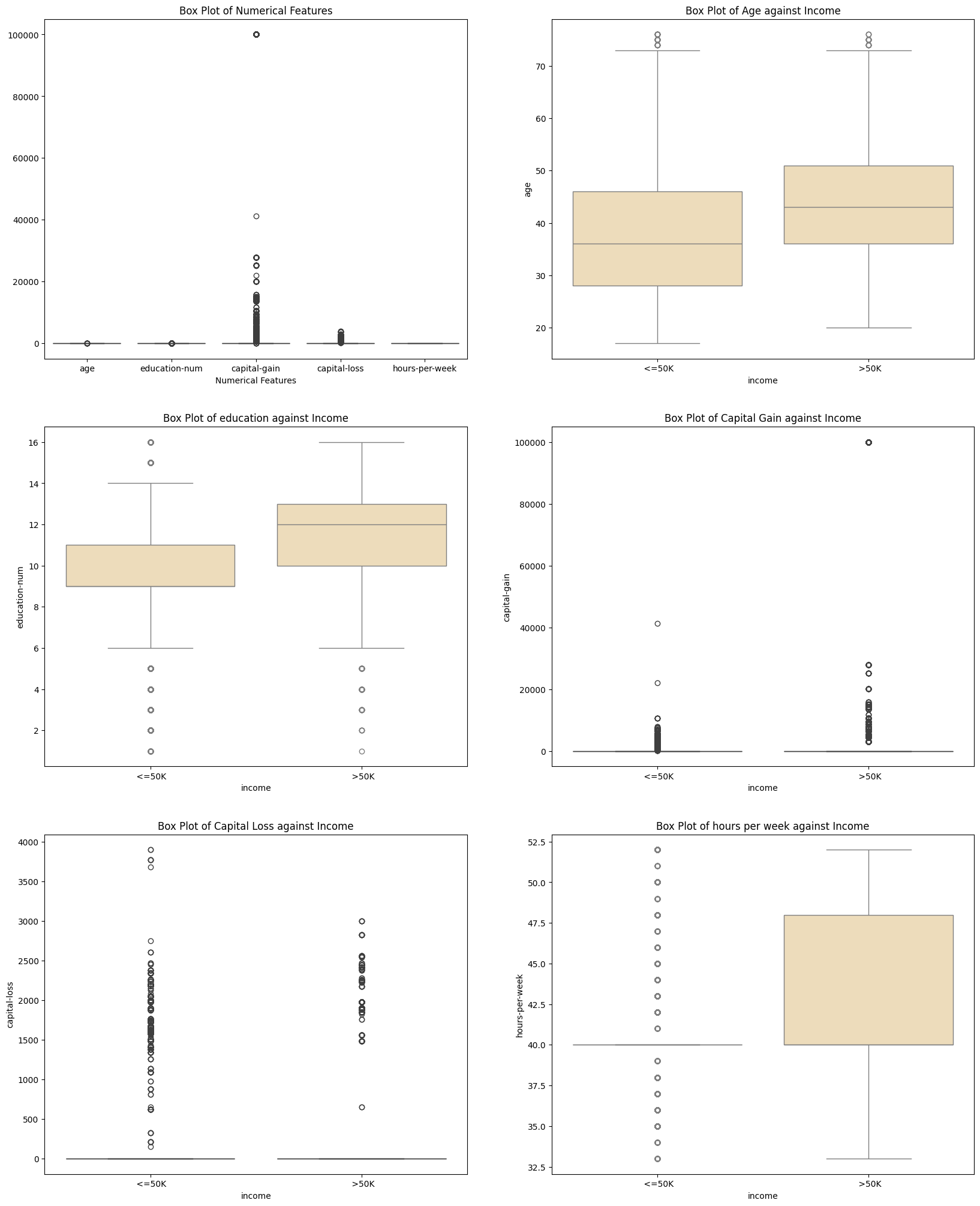
Description automatically generated

Generate a summary DataFrame named summary\_cd that provides a comprehensive overview.

A screenshot of a computer

Description automatically generated

## **BOX PLOTS AGAINST INCOME FOR NUMERICAL FEATURES**

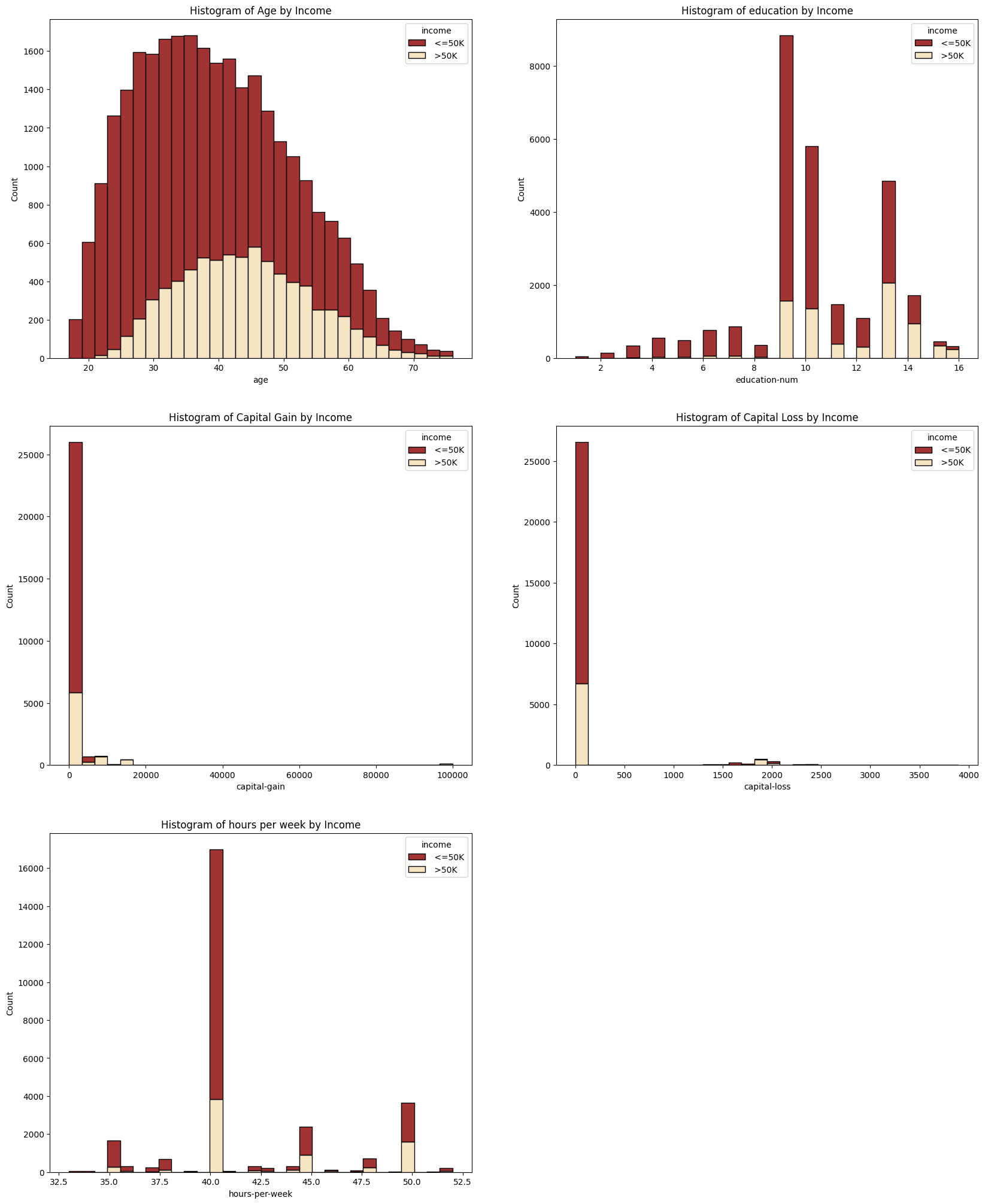


Box plots provide insights into the relationship between different numerical factors and income categories. By examining how these numerical characteristics are distributed for income levels below or above $50K, one can identify patterns that may be predictive.

## A graph showing the amount of income Description automatically generated**INCOME DISTRIBUTION**

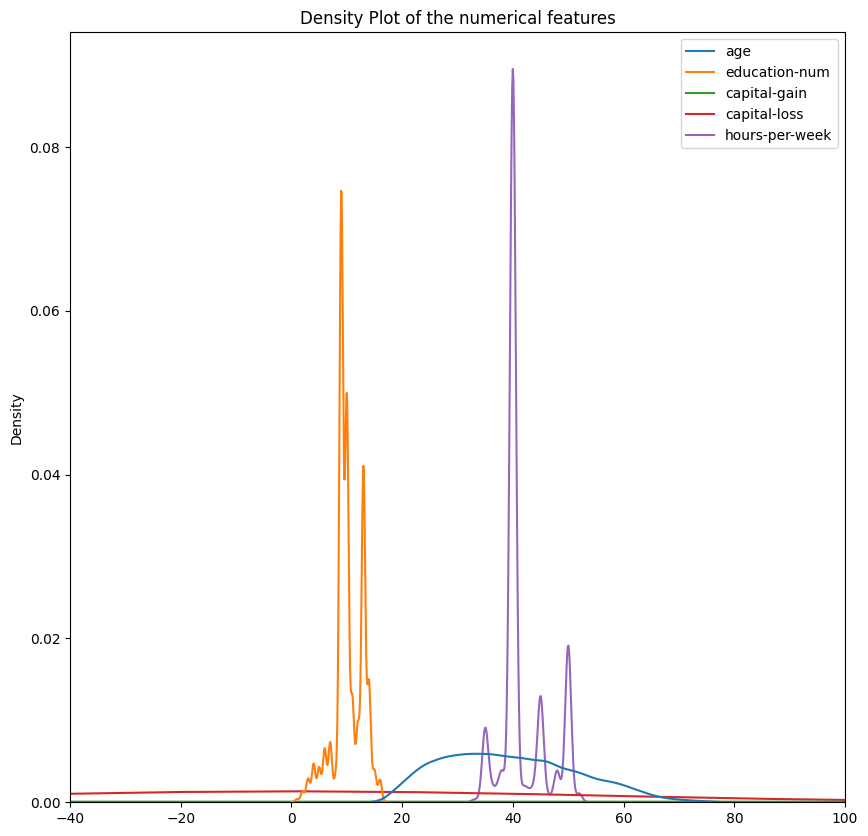
“Income Distribution Graph” compares the number of individuals with incomes less than or equal to 50K and those with incomes greater than 50K.

## **HISTOGRAMS FOR NUMIRICAL FEATURES**



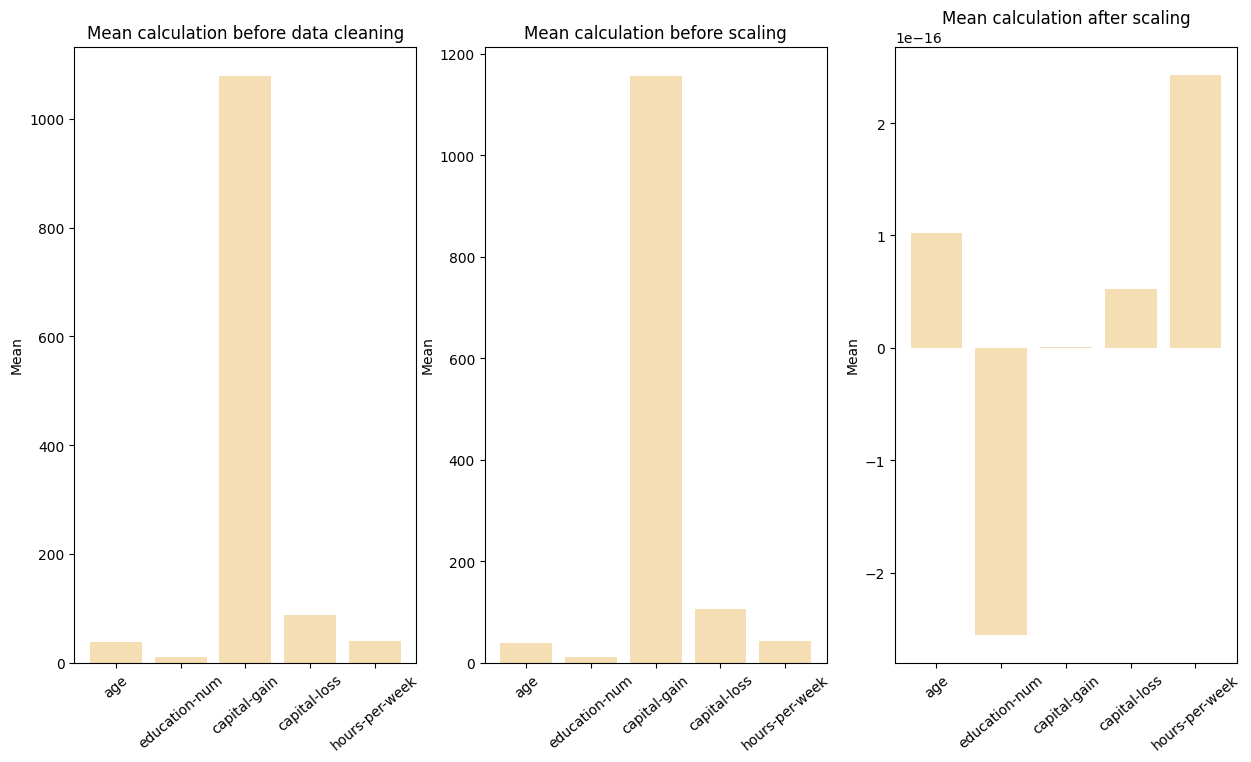
The histograms contrast income levels below and over $50,000 by showing the distribution of age, education, and work hours. Lower income workers tend to be younger and have higher levels of education, whereas higher earners are more evenly distributed in terms of age and full-time work hours. Low investment 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**



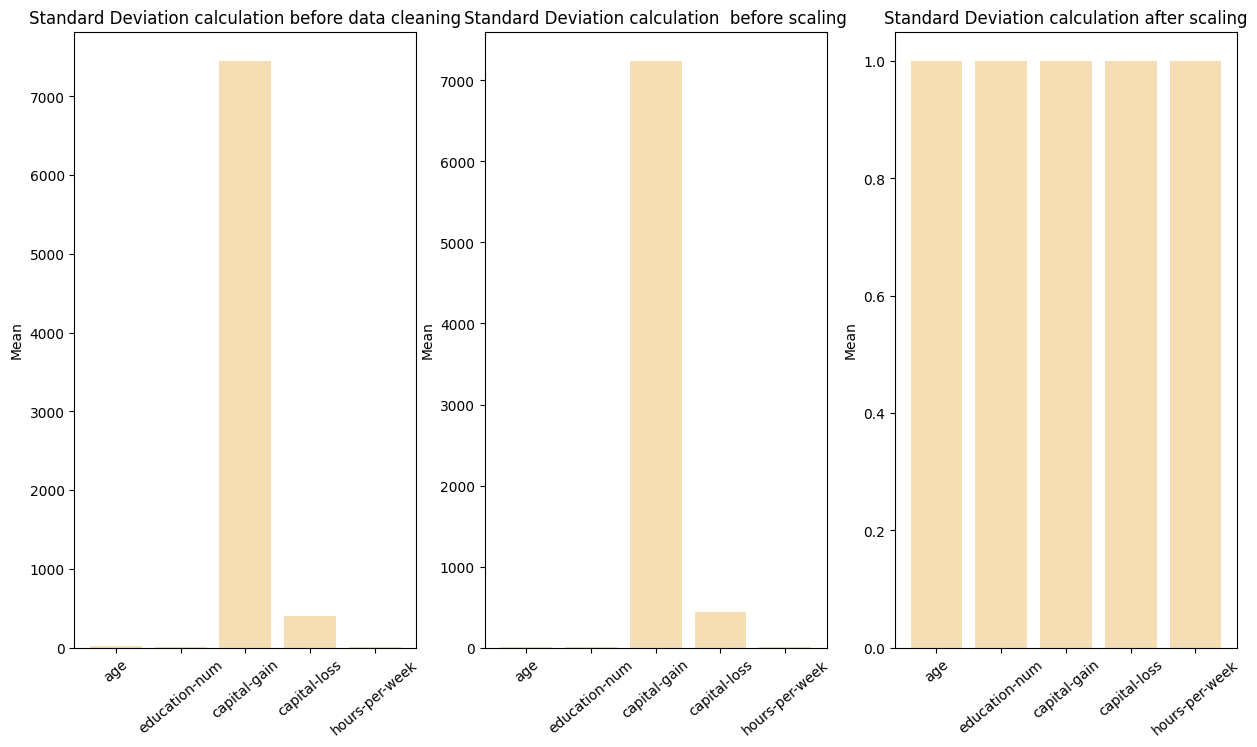
The distribution patterns across a range of values are displayed in the density plot for the numerical attributes in the dataset. Comparative study of the distributions of the variables is made possible by the overlaying of several variables. Plotting peaks indicate regions of concentrated data, whereas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



A screenshot of a computer

Description automatically generated



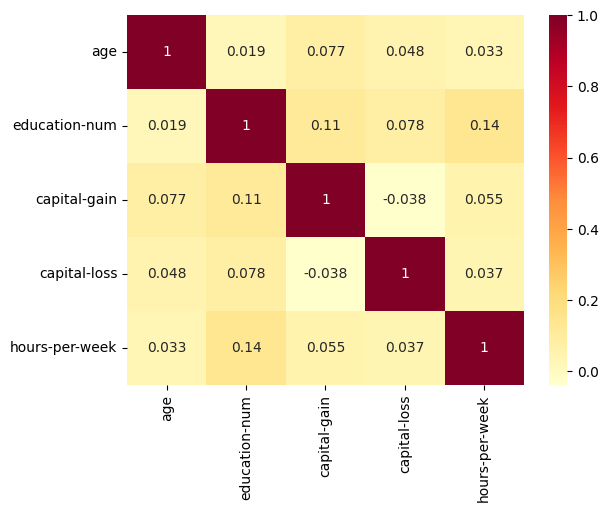
## **DATA SCALING**

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.

A screenshot of a computer

Description automatically generated

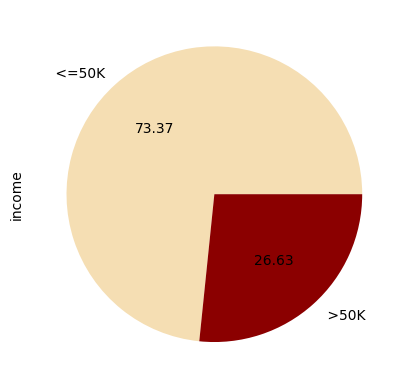
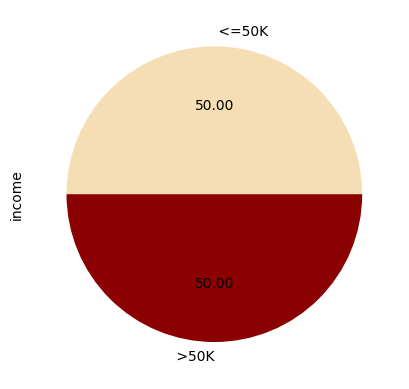
## **CHECK FOR CORRELATION**

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.

After Resampling

Before 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**

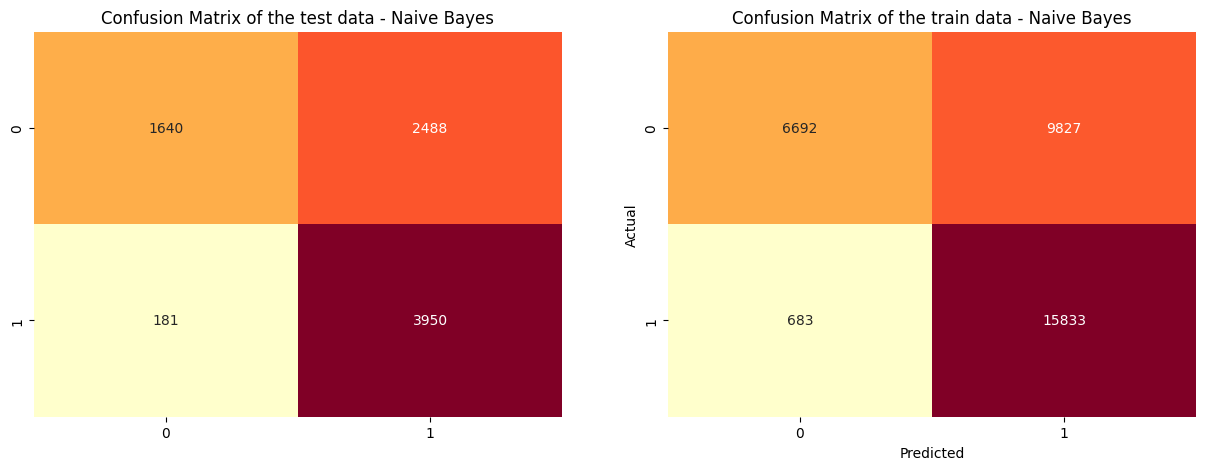
Random Forest is an ensemble method that builds multiple decision trees during training and aggregates their predictions to enhance accuracy and reduce overfitting. Each tree is trained on a random sample of data and features, allowing the model to capture complex, nonlinear relationships. The method 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**

A screenshot of a computer

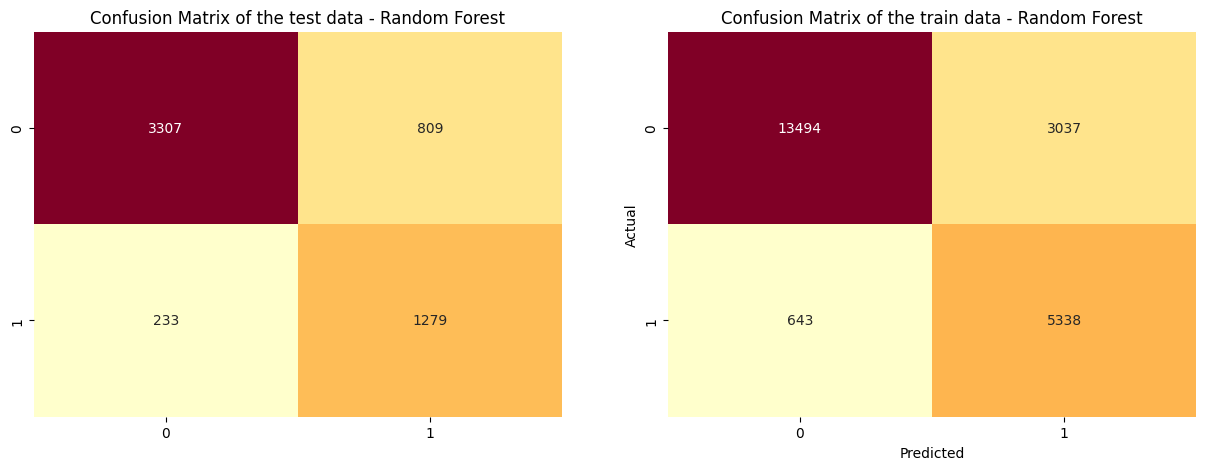
Description automatically generatedClassification report:



## **Random Forest Classifier Results**

A screenshot of a computer screen

Description automatically generatedClassification Report:



## **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','float64']).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/.](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/.](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/.](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/.](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/.](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/.](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/.](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/.](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].