INFORMATICS INSTITUTE OF TECHNOLOGY

In Collaboration with

ROBERT GORDON UNIVERSITY ABERDEEN

MACHINE LEARNING COURSEWORK

Manujaya Rajaguru

IIT ID – 20220654

RGU ID – 2312551

Table of Contents

1 INTRODUCTION

This assignment's main objective is to classify the income to determine whether it surpasses $50,000 annually using census data. To achieve this classification, the algorithms Naïve Bayes and Random Forest were used.

2 CORPUS COOPERATION

2.1 Data PRE-Processing

The "Adult" dataset from the UCI Machine Learning Repository comprises 14 attributes capturing demographic and employment details to predict whether an individual earns more than $50,000 annually. With features including age, education, and occupation, it facilitates binary classification based on income level. Preprocessing steps such as handling missing values and encoding categorical variables are necessary for modeling, while the dataset offers insights into socio-economic disparities and factors influencing earning potential.

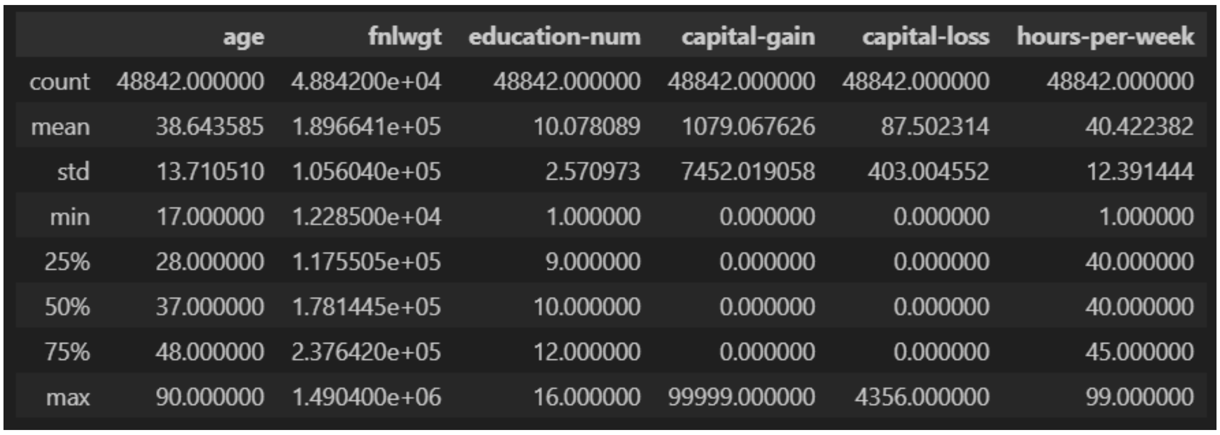
2.2 Finding the Null values and Dropping them

This looks for missing values in the dataset and handles them appropriately. It first looks for rows in the dataset that include the character '?' by returning True or False. If it is true it indicates the presence of null values. It then totals the number of such rows. This count indicates the number of missing or unknown values in the dataset, denoted by the symbol '?'.

Then it excludes the rows containing the symbol "?" and the null values are handled.

No rows in the dataset before removing duplicates	No rows in the dataset after removing duplicates
48790	45715

2.3 Statistics

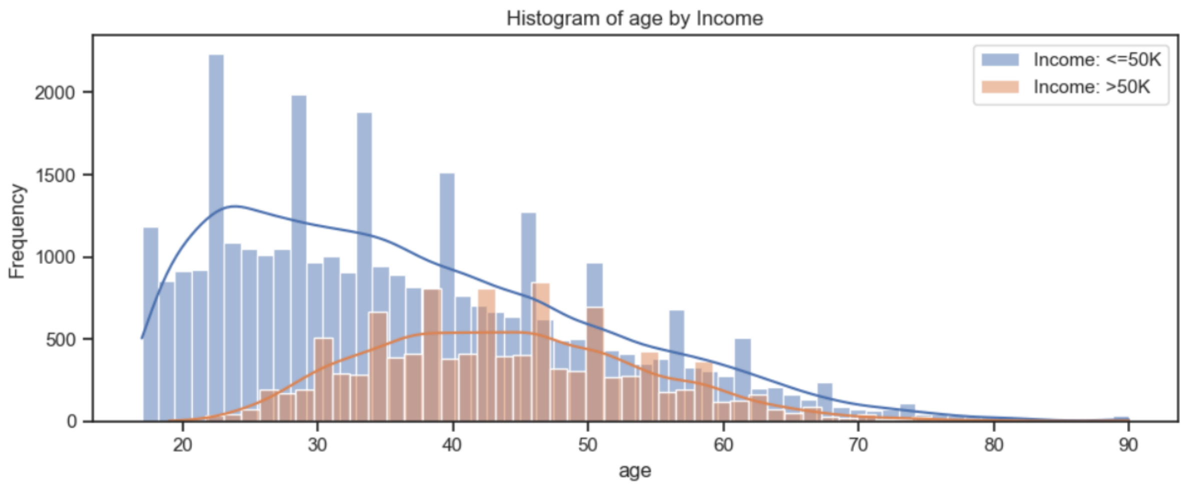


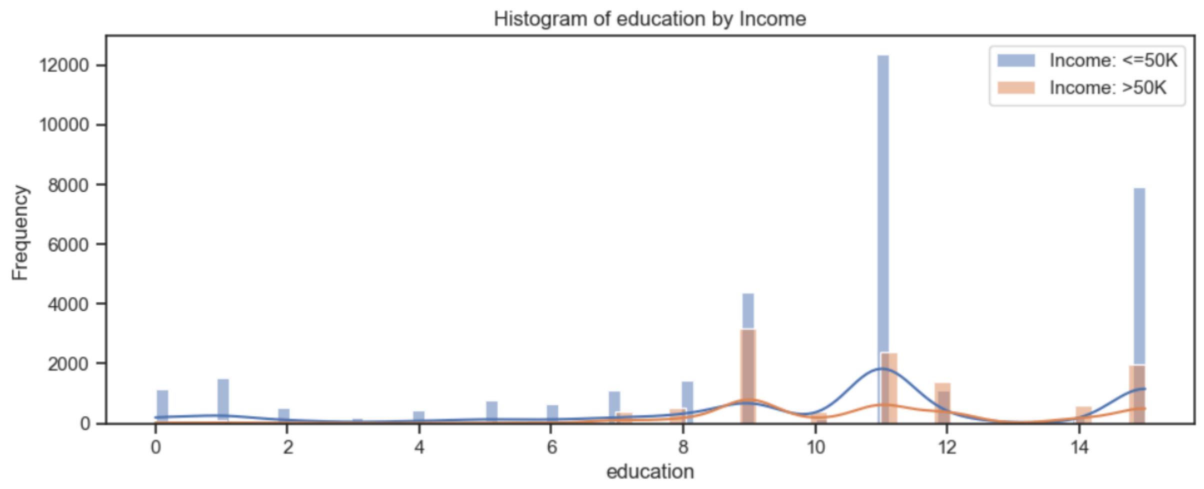
	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week
count	48842.000000	4.884200e+04	48842.000000	48842.000000	48842.000000	48842.000000
mean	38.643585	1.896641e+05	10.078089	1079.067626	87.502314	40.422382
std	13.710510	1.056040e+05	2.570973	7452.019058	403.004552	12.391444
min	17.000000	1.228500e+04	1.000000	0.000000	0.000000	1.000000
25%	28.000000	1.175505e+05	9.000000	0.000000	0.000000	40.000000
50%	37.000000	1.781445e+05	10.000000	0.000000	0.000000	40.000000
75%	48.000000	2.376420e+05	12.000000	0.000000	0.000000	45.000000
max	90.000000	1.490400e+06	16.000000	99999.000000	4356.000000	99.000000

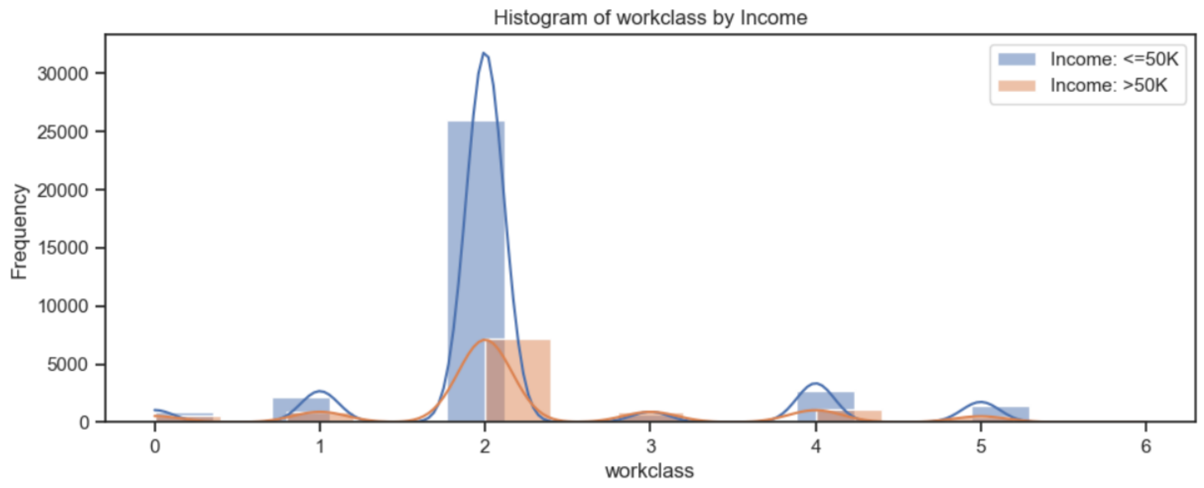
Both a statistical summary of the dataset's numerical features and a summary of its category attributes are shown here.

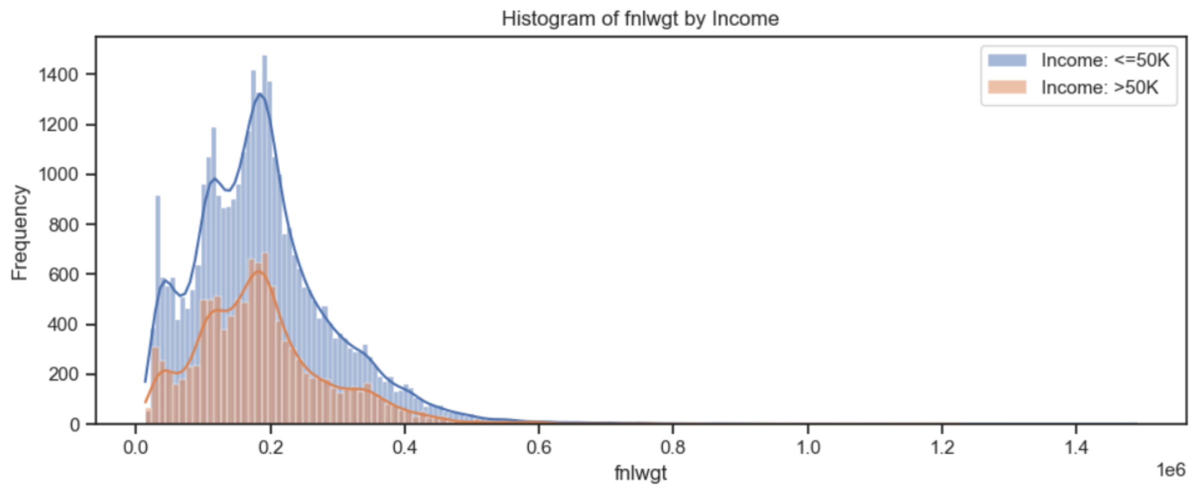
2.4 Histogram for Numerical Features

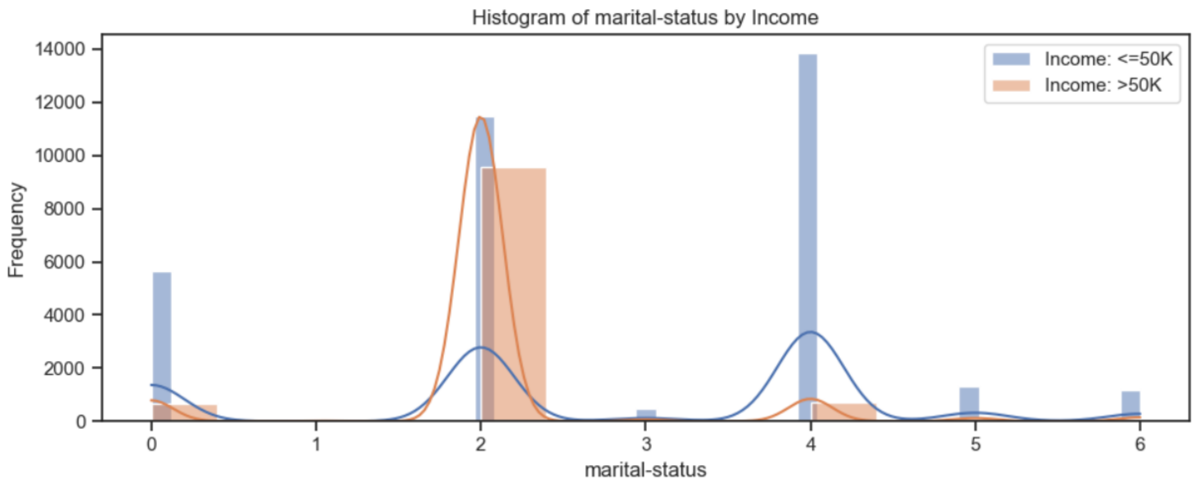
Bar plots allow us to visualize the distribution of numerical features across income categories. This reveals whether certain numerical properties have a varied distribution across income levels. It helps to Identify patterns or trends among numerical characteristics and income levels.

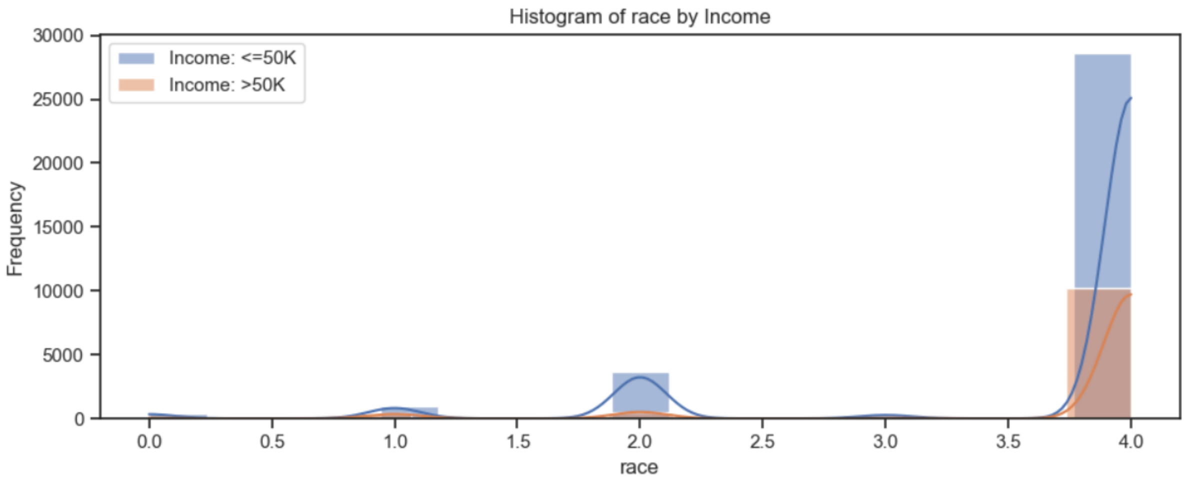


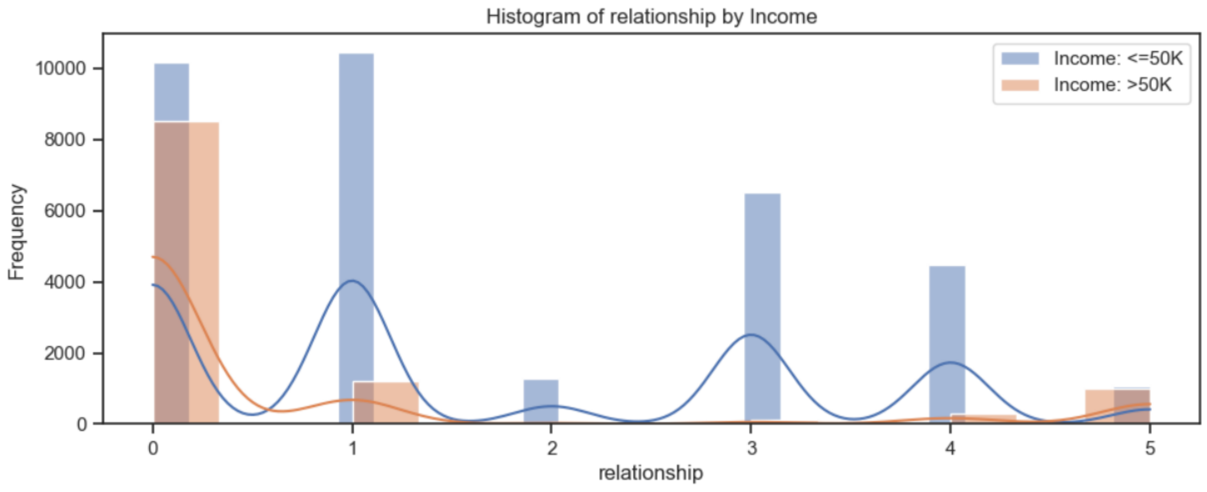

Histogram of education by Income
Frequency
education
Income: <=50K
Income: >50K

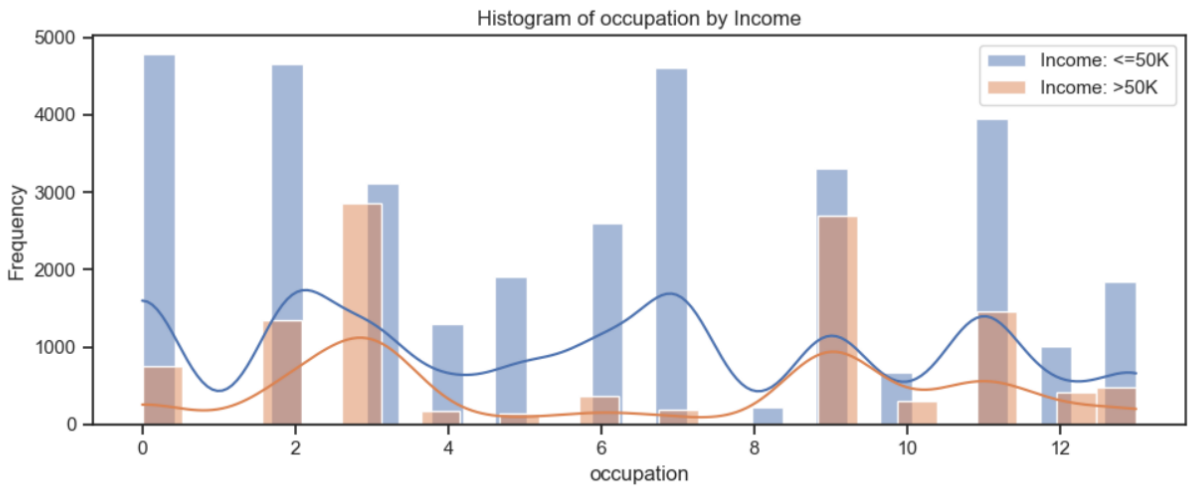

Histogram of workclass by Income
Frequency
workclass
Income: <=50K
Income: >50K

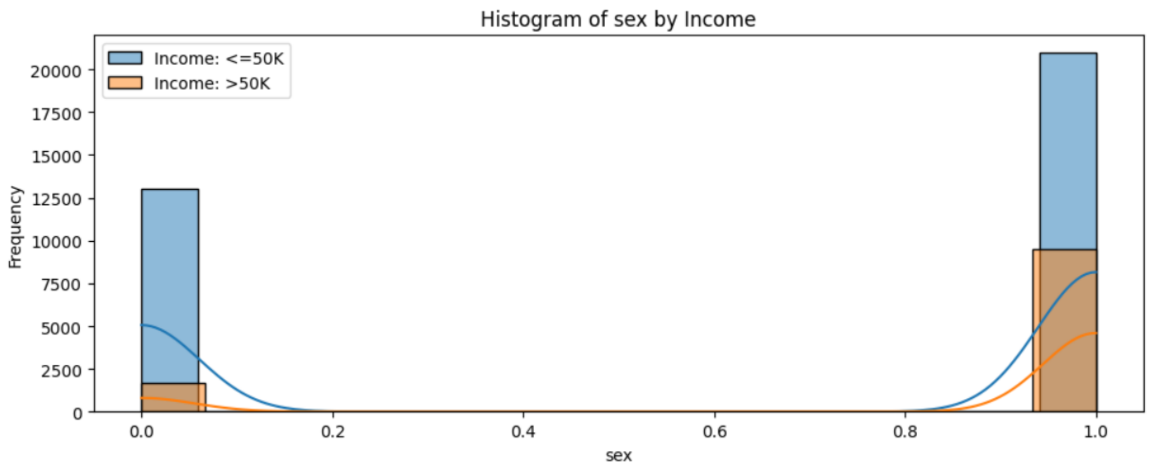

Histogram of fnlwgt by Income
Frequency
fnlwgt
1e6
Income: <=50K
Income: >50K

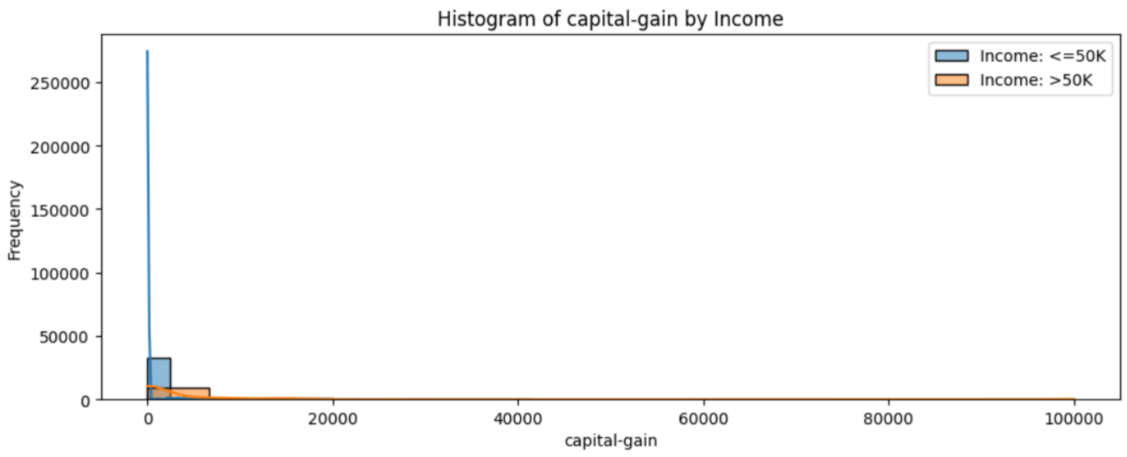

Histogram of marital-status by Income
Income: <=50K
Income: >50K
Frequency
marital-status
0
2000
4000
6000
8000
10000
12000
14000
0
1
2
3
4
5
6

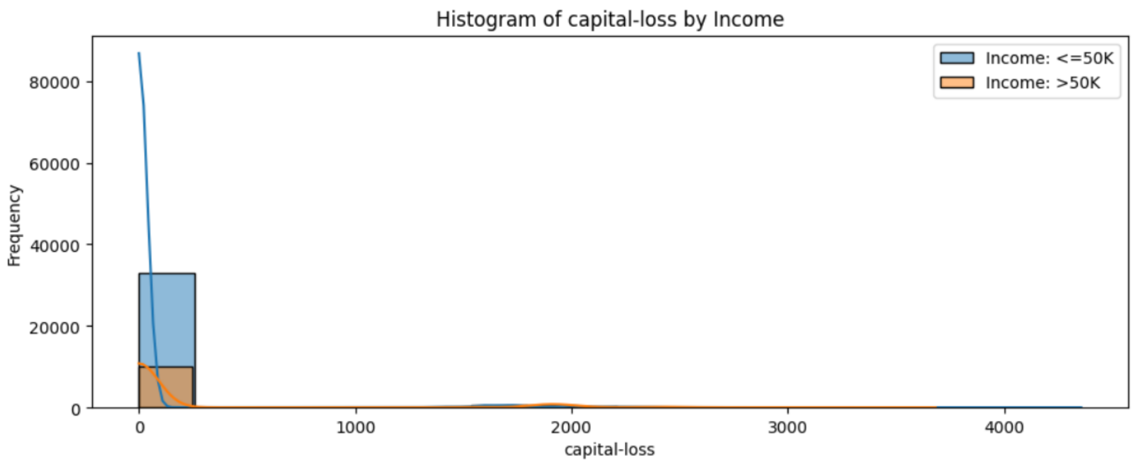

Histogram of race by Income
Income: <=50K
Income: >50K
Frequency
race
0
5000
10000
15000
20000
25000
30000
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

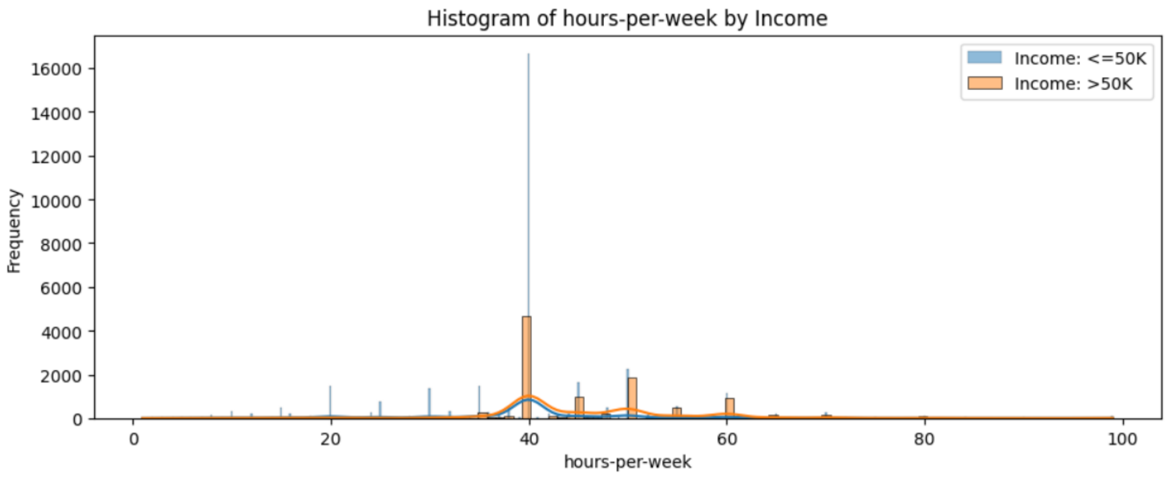

Histogram of relationship by Income
Income: <=50K
Income: >50K
Frequency
relationship
0
2000
4000
6000
8000
10000
0
1
2
3
4
5

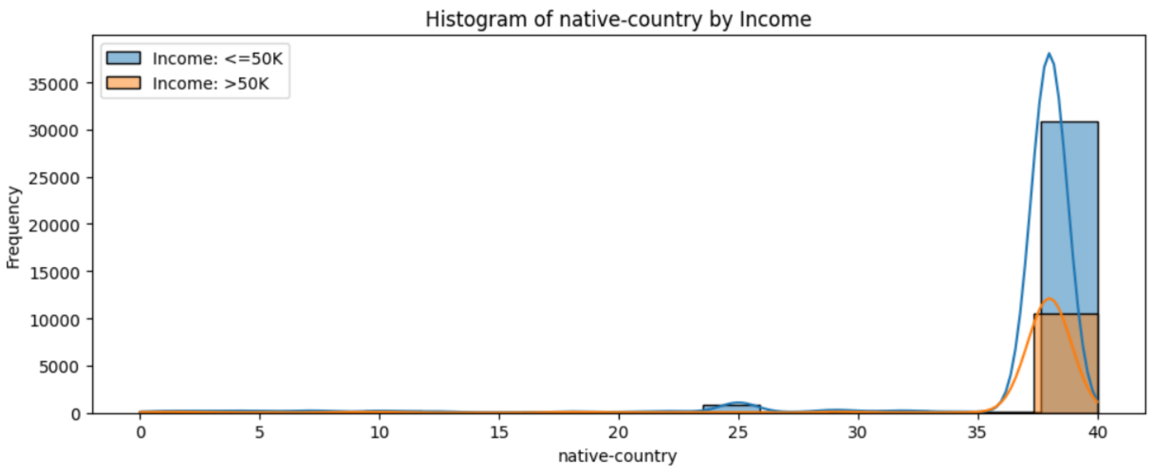

Histogram of occupation by Income
Frequency
occupation
Income: <=50K
Income: >50K


Histogram of sex by Income
Frequency
sex
Income: <=50K
Income: >50K


Histogram of capital-gain by Income
Frequency
capital-gain
Income: <=50K
Income: >50K


Histogram of capital-loss by Income
Income: <=50K
Income: >50K
Frequency
80000
60000
40000
20000
0
0
1000
2000
3000
4000
capital-loss


Histogram of hours-per-week by Income
Income: <=50K
Income: >50K
Frequency
16000
14000
12000
10000
8000
6000
4000
2000
0
0
20
40
60
80
100
hours-per-week


Histogram of native-country by Income
Income: <=50K
Income: >50K
Frequency
35000
30000
25000
20000
15000
10000
5000
0
0
5
10
15
20
25
30
35
40
native-country

2.5 Label Encoding

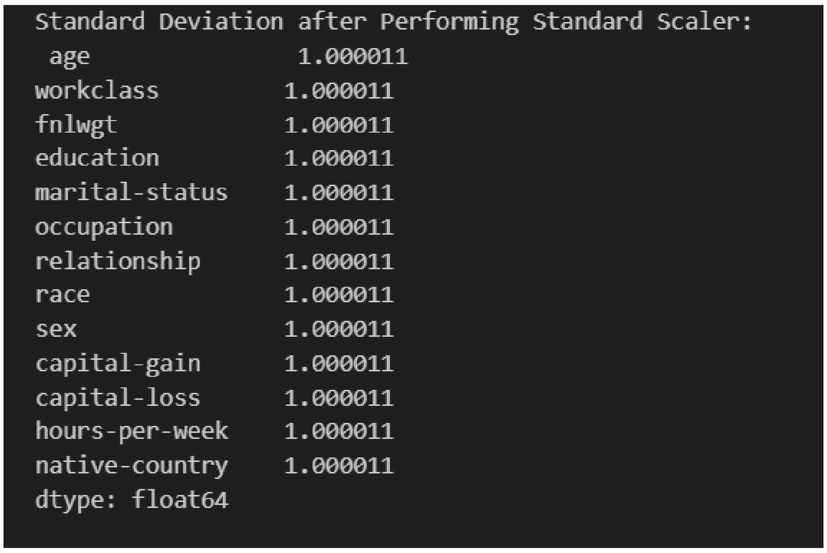
Categorical columns within the dataset are identified by their data type ('object'). The target variable, denoting income level, is separated from the dataset. Utilizing the LabelEncoder from the scikit-learn library, categorical columns are encoded into numerical values, facilitating their utilization in machine learning algorithms. This encoding process replaces the original categorical values with their corresponding numerical representations within the dataset, enabling the inclusion of categorical features in predictive models. The resulting dataset displays the transformed categorical columns alongside any remaining numerical columns, ensuring compatibility for subsequent modeling tasks.

2.6 Standard Scaler

The dataset's numerical columns are standardized using this. During preprocessing, features are standardized by scaling them to a mean of 0 and a standard deviation of 1 using the Standard Scaler. By guaranteeing that every feature contributes equally to the analysis, avoiding features with larger sizes from controlling the model, and enhancing algorithm efficiency and convergence, this transformation helps machine learning algorithms. Standardizing features also makes models more reliable and effective by enhancing optimization algorithms and regularization strategies. Thus, it initially constructs a new data frame and initializes, fits, and converts the numeric columns using standard scalar.

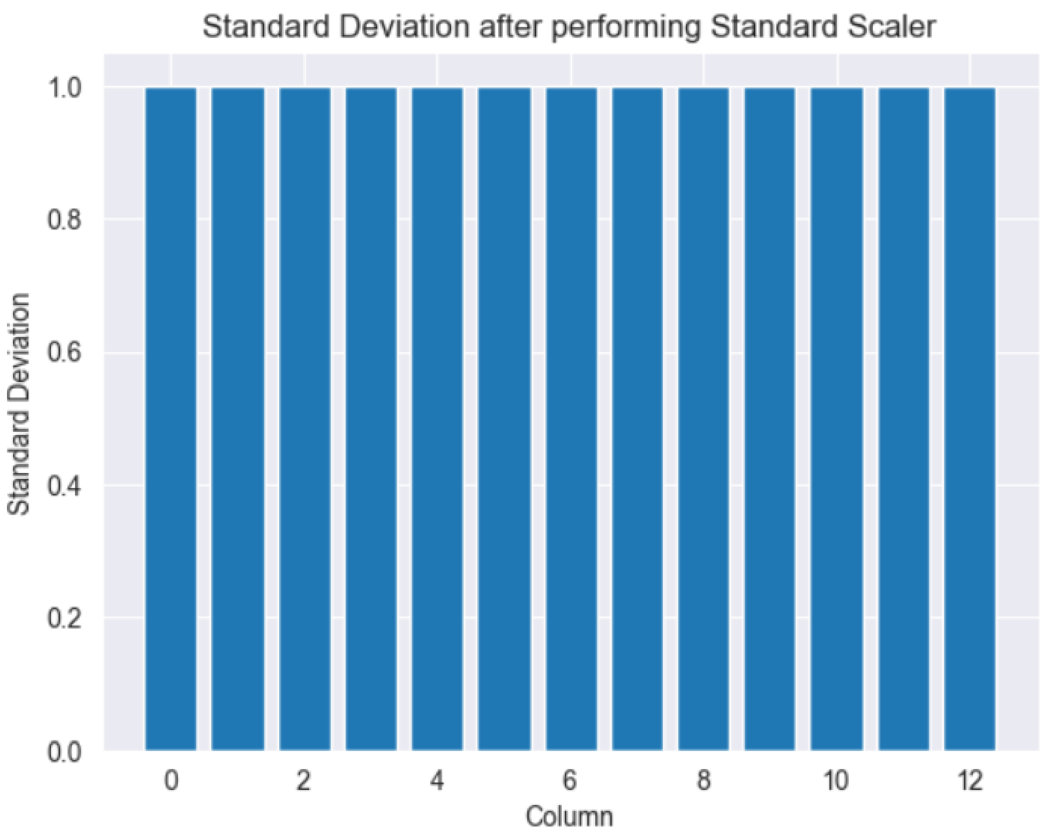
2.7 Standard Deviation after Standard Scalar

The standard deviation of each column in the dataset is computed after applying the Standard Scaler transformation. This operation is carried out to measure the spread or variability of values within each feature after normalization. The resulting standard deviations provide insights into the dispersion of data points from the mean, aiding in the understanding of the distributional characteristics and potential scale differences across different features.



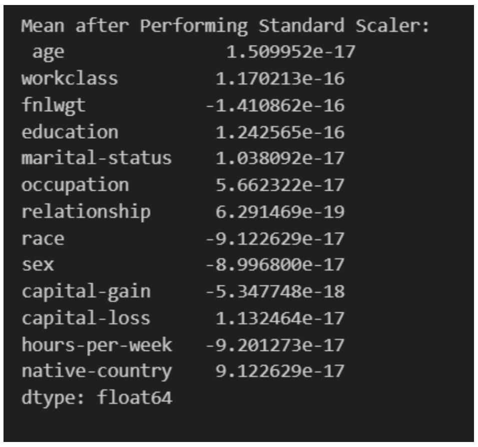
```
Standard Deviation after Performing Standard Scaler:
 age               1.000011
workclass         1.000011
fnlwgt            1.000011
education         1.000011
marital-status    1.000011
occupation        1.000011
relationship      1.000011
race              1.000011
sex               1.000011
capital-gain      1.000011
capital-loss      1.000011
hours-per-week    1.000011
native-country    1.000011
dtype: float64
```

The below visualization enables a visual comparison of the spread of values across different features in the dataset. Each bar on the plot represents the standard deviation of a specific feature, providing insights into the variability of values within each attribute.



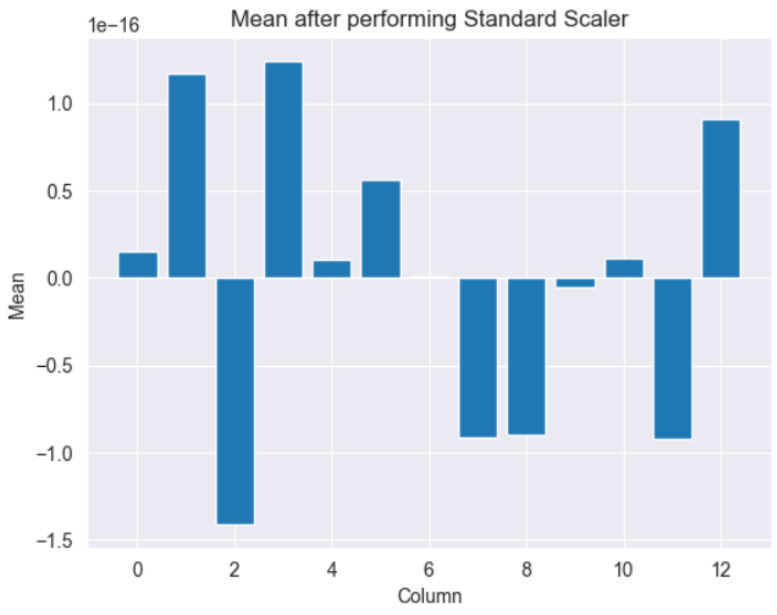
2.8 Mean after Standard Scaler

The mean of each column in the dataset is calculated after applying the Standard Scaler transformation. This computation allows for an assessment of the central tendency of the data within each feature, showcasing the average value around which the observations are distributed post-scaling.



```
Mean after Performing Standard Scaler:
 age               1.509952e-17
workclass          1.170213e-16
fnlwgt            -1.410862e-16
education          1.242565e-16
marital-status     1.038092e-17
occupation         5.662322e-17
relationship       6.291469e-19
race              -9.122629e-17
sex               -8.996800e-17
capital-gain      -5.347748e-18
capital-loss       1.132464e-17
hours-per-week    -9.201273e-17
native-country     9.122629e-17
dtype: float64
```

This is a bar plot illustrating the mean of each column after applying the Standard Scaler transformation. Each bar on the plot represents the mean value of a specific feature, offering insights into the central tendency of the data distribution within each attribute.



2.9 Splitting into Train and Test split

Splitting a dataset into training and testing sets is critical in machine learning for evaluating model performance on previously unknown data. It entails dividing the dataset into two subsets: one for training the model and another for assessing its performance. 20% of the data will be allocated to the testing set, while the remaining 80% will be used for training.

3 SOLUTION METHODOLOGY

3.1 DATA PREPROCESSING

First, the data is completed, meaning that duplicates and null values are found and managed, categorical variables are encoded using one-hot method, and numerical characteristics are standardized.

Following data exploration, a statistical summary of the categorical and numerical features was produced. To compare the distribution of the numerical features, box and density plots for each numerical feature were displayed. Next, each numerical column's mean and standard deviation are displayed and represented using a histogram. Following this, the Data Frame's numerical properties undergo standardization. It generates a bar plot to display the mean and standard deviation following standardization once the numerical characteristics have been scaled.

3.2 Model Development

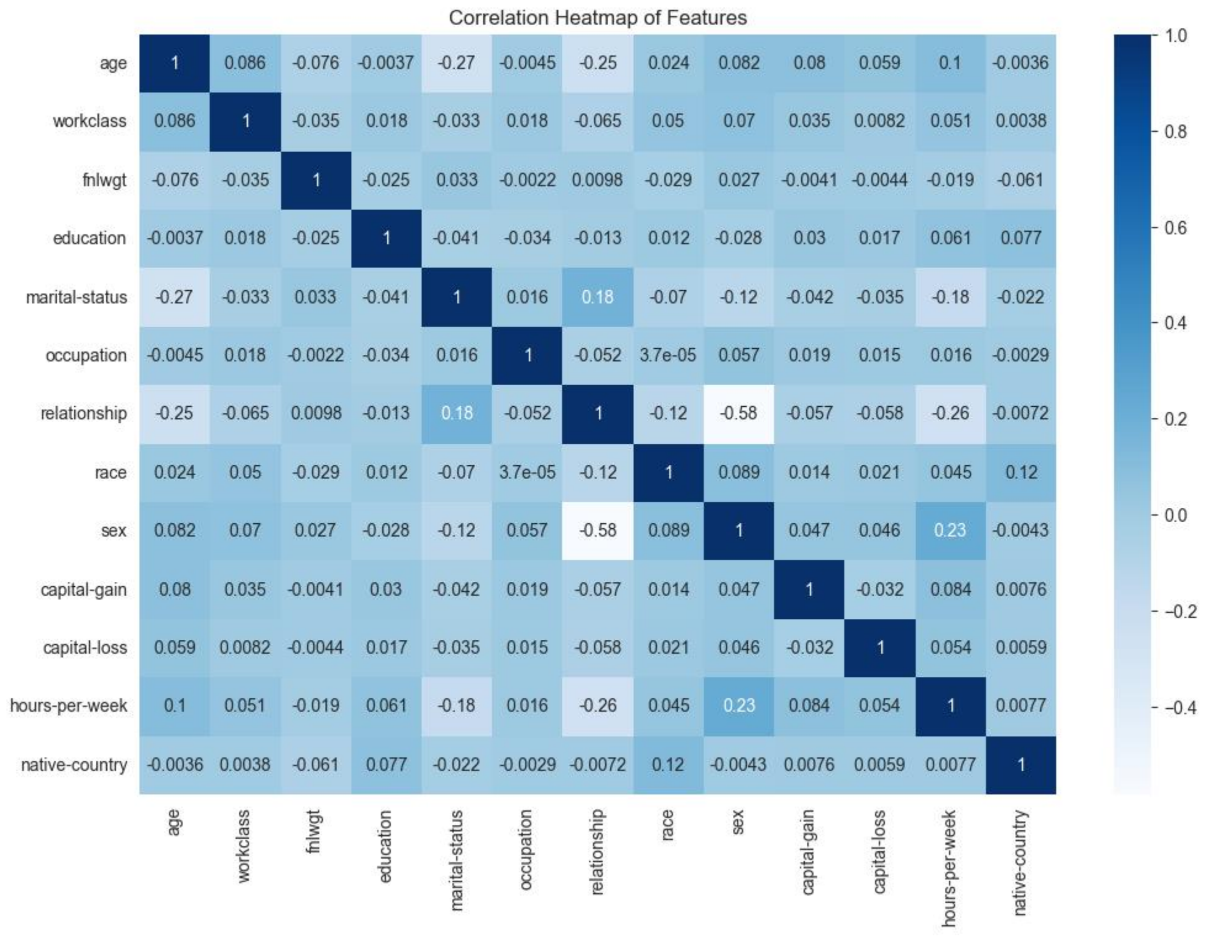
Prior to the building of the model, the dataset is split to train and test, the dataset is split into two, one for training the model and the other one for assessing the performance of the model.

3.2.1 NAÏVE BAYES

Gaussian Naïve Bayes is selected as the classifier for the prediction. A straightforward probabilistic classifier based on the Bayes theorem and feature independence is called Gaussian Naive Bayes (GNB). It assumes that a characteristic's likelihood of falling into a class is

Gaussian, or regularly distributed. Given a set of features, GNB calculates the likelihood of each class and utilizes the category with the highest probability as the prediction. In actuality, GNB usually performs better than its simplistic and naive feature independence assumption suggests, especially when dealing with continuous-valued features.

The Gaussian Naïve Bayes is used, because the features that are used to describe the instances in a dataset are independent from each other.



	age	workclass	fnlwgt	education	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country
age	1	0.086	-0.076	-0.0037	-0.27	-0.0045	-0.25	0.024	0.082	0.08	0.059	0.1	-0.0036
workclass	0.086	1	-0.035	0.018	-0.033	0.018	-0.065	0.05	0.07	0.035	0.0082	0.051	0.0038
fnlwgt	-0.076	-0.035	1	-0.025	0.033	-0.0022	0.0098	-0.029	0.027	-0.0041	-0.0044	-0.019	-0.061
education	-0.0037	0.018	-0.025	1	-0.041	-0.034	-0.013	0.012	-0.028	0.03	0.017	0.061	0.077
marital-status	-0.27	-0.033	0.033	-0.041	1	0.016	0.18	-0.07	-0.12	-0.042	-0.035	-0.18	-0.022
occupation	-0.0045	0.018	-0.0022	-0.034	0.016	1	-0.052	3.7e-05	0.057	0.019	0.015	0.016	-0.0029
relationship	-0.25	-0.065	0.0098	-0.013	0.18	-0.052	1	-0.12	-0.58	-0.057	-0.058	-0.26	-0.0072
race	0.024	0.05	-0.029	0.012	-0.07	3.7e-05	-0.12	1	0.089	0.014	0.021	0.045	0.12
sex	0.082	0.07	0.027	-0.028	-0.12	0.057	-0.58	0.089	1	0.047	0.046	0.23	-0.0043
capital-gain	0.08	0.035	-0.0041	0.03	-0.042	0.019	-0.057	0.014	0.047	1	-0.032	0.084	0.0076
capital-loss	0.059	0.0082	-0.0044	0.017	-0.035	0.015	-0.058	0.021	0.046	-0.032	1	0.054	0.0059
hours-per-week	0.1	0.051	-0.019	0.061	-0.18	0.016	-0.26	0.045	0.23	0.084	0.054	1	0.0077
native-country	-0.0036	0.0038	-0.061	0.077	-0.022	-0.0029	-0.0072	0.12	-0.0043	0.0076	0.0059	0.0077	1

Since the correlation coefficients are closer to zero, this visualization suggests that the features are relatively independent.

Every feature has a probability distribution that is Gaussian in nature. Since Gaussian Naïve Bayes is specifically designed for continuous features—such as the variables "workclass," "education," "marital-status," "occupation," "relationship," "race," "sex," and "native-country"—it was chosen over Multinomial Naive Bayes or Bernoulli Naive Bayes in this particular census dataset.

The Gaussian Naïve Bayes classifier was first initialized, and then it was fitted to the training data. The classifier uses the training data to determine the parameters that make up Gaussian distributions for each attribute in each class during the fitting process. After the classifier is trained, it can be used to make predictions about new data sets.

3.2.2 RANDOM FOREST Classifier

During training, the Random Forest ensemble learning technique generates a large number of decision trees. It then combines these predictions to maximize accuracy and reduce overfitting. Each decision tree in the forest is trained using a different subset of the training set of features and data, which is how it operates.

It is possible for random forests to identify nonlinear relationships between the target variable and features. The Random Forest technique minimizes overfitting by generating many decision trees and averaging their forecasts. Even with a high feature count, this ensemble approach reduces the risk of overfitting and increases generalizability to fresh data. Random Forest is robust against data outliers. Compared to other techniques, random forest's model performance is less affected by outliers generally.

To ensure that the findings are repeatable, it first initializes a random forest classifier and passes a random state of 42. Next, using the training set of data, the Random Forest Classifier model is used. Using various subsets of the training data and attributes, the Random Forest approach generates a large number of decision trees during the training process. To make forecasts that are more accurate, these decision trees are combined.

4 EVALUATION CRITERIA

The metrics—accuracy, f1 score, precision, and recall score—that are used to determine if an individual's annual income surpasses $50,000 are listed below, along with the reasons behind their application.

4.1 Accuracy

The percentage of individuals whose income is accurately determined to be over $50,000 annually or not exceeds that amount is known as accuracy.

$$\boxed{Accuracy = \frac{TP + TN}{TP + TN + FP + FN}}$$

4.2 Precision

The percentage of accurately identified individuals with yearly incomes over $50,000 among all those predicted to have that level of income would serve as the precision measure. High precision results in fewer false positives.

$$\boxed{Precison = \frac{TP}{TP + FP}}$$

4.3 Recall

The percentage of people with yearly incomes over $50,000 who were correctly identified out of all those with such incomes is known as the recall rate. A high recall rate implies fewer false negatives.

$$\boxed{Recall = \frac{TP}{TP + FN}}$$

4.4 F1 Scores

The F1 score will reveal how well the model strikes a compromise between reducing misclassification and accurately recognizing individuals earning more than $50,000 annually.

$$\boxed{F1\ score = \frac{2 \times (precision \times recall)}{precision + recall}}$$

5 MODEL EVALUATION

5.1 ACCURACY OF THE TRAINING DATASET

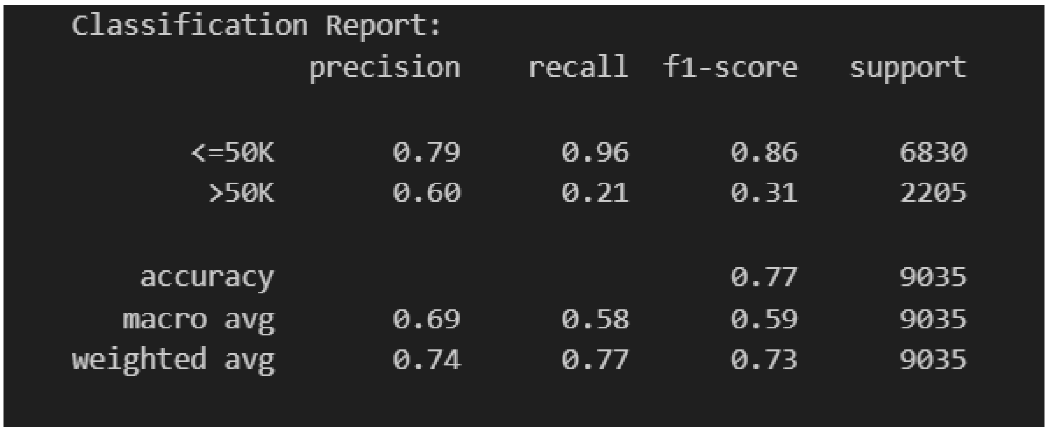
The way a model performs on the data it was trained on is known as its training accuracy. Determining how well the model fits the training data requires an understanding of training accuracy.

Accuracy score of income exceeding $50000 per year prediction	Naïve Bayes	Random Forest
	77.29%	83.24%

A 77.29% training accuracy was achieved with the Naive Bayes classifier. About 77.29% of the training dataset's instances were correctly classified by the Naive Bayes model. 83.24% was a higher training accuracy for the Random Forest classifier. As evidenced by the few misclassifications, Random Forest produced almost perfect predictions on the training dataset.

Using many decision trees, Random Forest is an ensemble learning technique. It has an excellent training accuracy because of its well-known capacity to identify intricate patterns and relationships in data. In comparison to Naive Bayes, Random Forest is a trickier model. Better performance on the training set could result from its ability to identify more complex patterns in the data.

5.2 Classification report (Precision, Recall, F1 scores) for NAÏVE BAYES

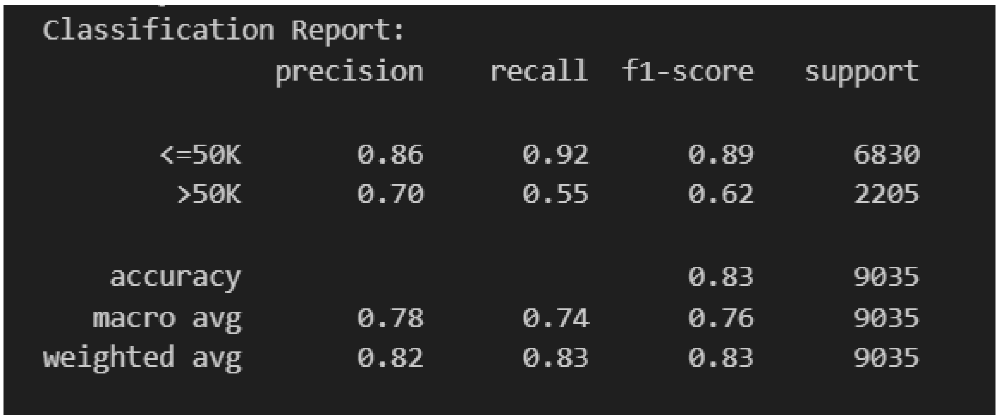


```
Classification Report:
              precision    recall  f1-score   support

       <=50K       0.79      0.96      0.86      6830
        >50K       0.60      0.21      0.31      2205

    accuracy                           0.77      9035
   macro avg       0.69      0.58      0.59      9035
weighted avg       0.74      0.77      0.73      9035
```

This produces a classification report that includes metrics for each class, including support, recall, f1 score, and precision. It offers a thorough rundown of how well the model classified test data.

The '<=50K' class has a higher precision model, which suggests fewer false positives. Still, recall for the '>50K' class is superior, suggesting that it captures more of the real '>50K' occurrences. In general, it seems that the model performs better for the '>50K' class in terms of recall and better for the '<=50K' class about precision.

5.3 Classification report (Precision, Recall, F1 scores) for RANDOM FOREST



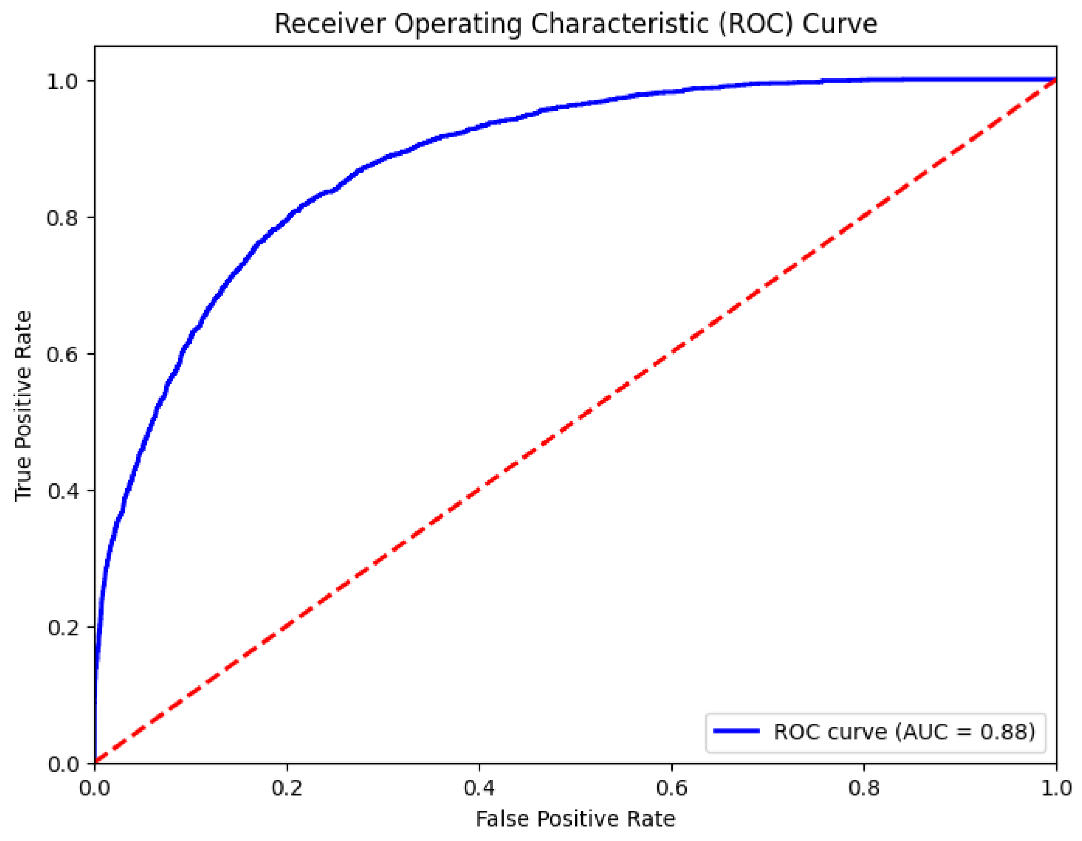
```
Classification Report:
              precision    recall  f1-score   support

       <=50K       0.86      0.92      0.89      6830
        >50K       0.70      0.55      0.62      2205

    accuracy                           0.83      9035
   macro avg       0.78      0.74      0.76      9035
weighted avg       0.82      0.83      0.83      9035
```

The model provides improved precision for the '<=50K' class, leading to fewer false positives. It has a higher recall for the '>50K' class, indicating that it captures a larger proportion of the actual '>50K' cases. The model's overall accuracy is 0.81, which means that about 81% of its predictions are right. For the '<=50K' class, the model offers increased precision, which reduces false positives. It catches a bigger percentage of the real '>50K' cases, as seen by its higher recall for the '>50K' class. With an overall accuracy of 0.81, the model makes around 81% of the correct predictions.

5.4 ROC CURVE

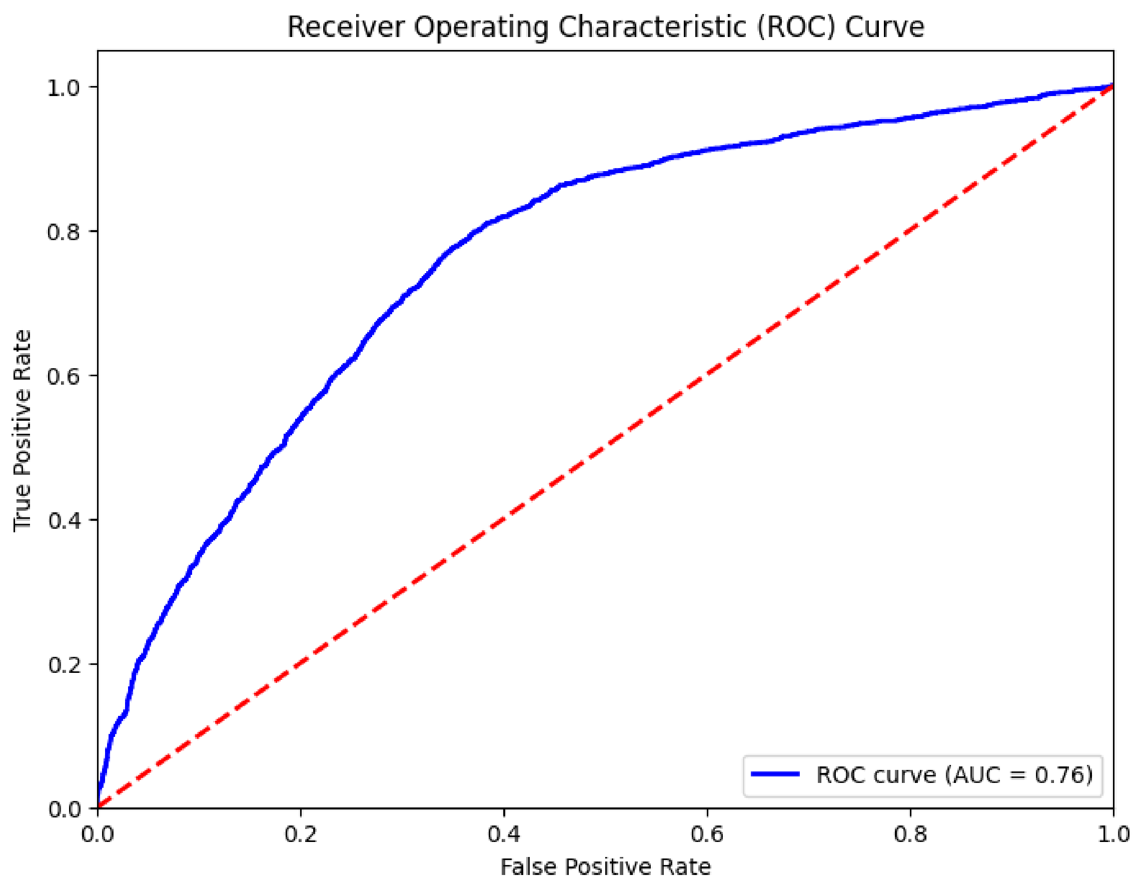
Graphical representations such as the Receiver Operating Characteristic (ROC) curve are frequently employed in performance analysis. The True Positive Rate and False Positive Rate are plotted on the ROC curve. An example showing how a classifier may distinguish between two separate groups (>50K$/year & <=50k$/year).

5.4.1 ROC Curve for RANDOM FOREST



- An AUC of 0.88 indicates that the Random Forest classifier works remarkably well in distinguishing between negative and positive samples.
- A randomly chosen positive instance is quite likely to receive a better score from the model than a randomly chosen negative one.
- This demonstrates how well the Random Forest model can identify the classes in the dataset and generate precise predictions.

5.4.2 ROC CURVE FOR NAÏVE BAYES



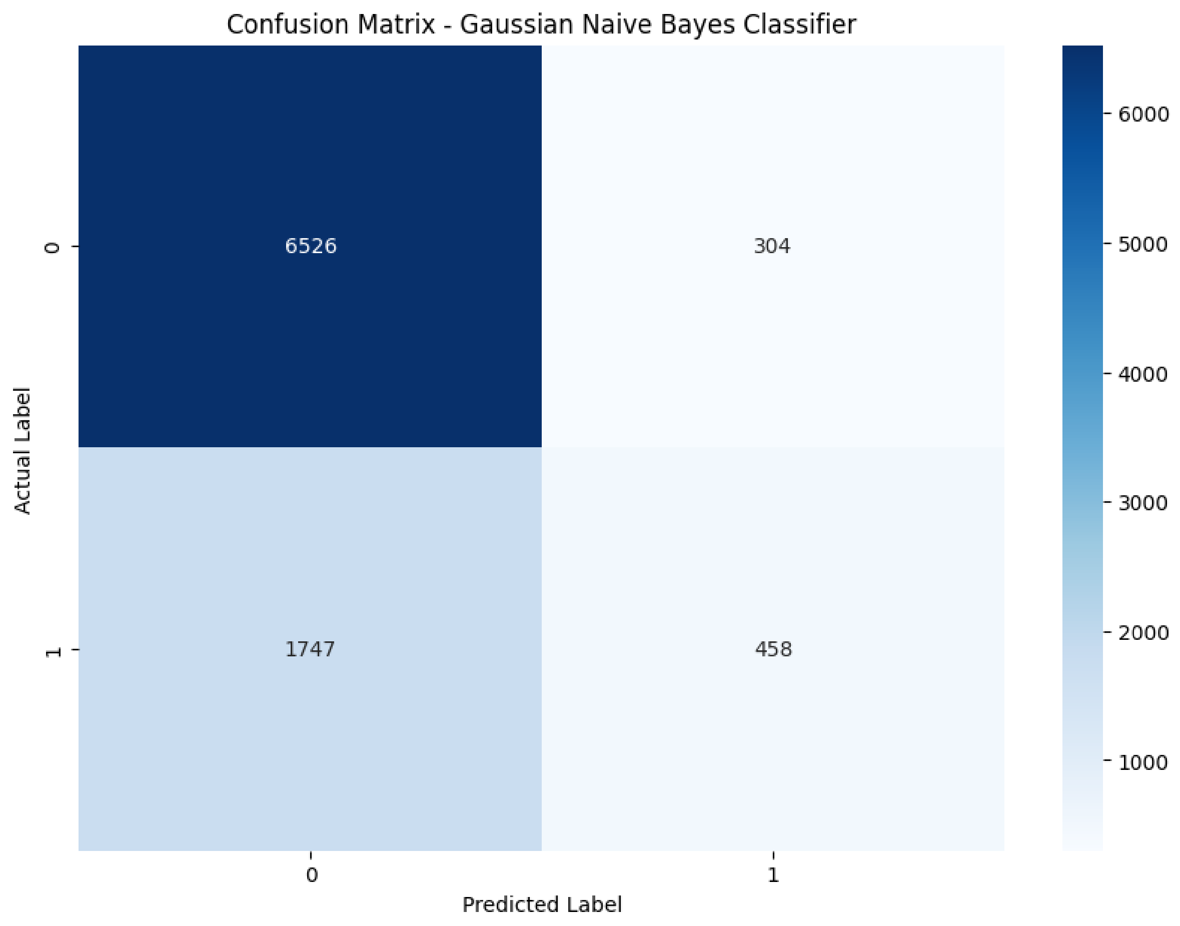
- Although a little lower than the Random Forest classifier, the AUC of 0.76 is still rather high.

- It demonstrates that, while still not as good as the Random Forest in this specific instance, the Naive Bayes classifier is also effective at distinguishing between positive and negative samples.

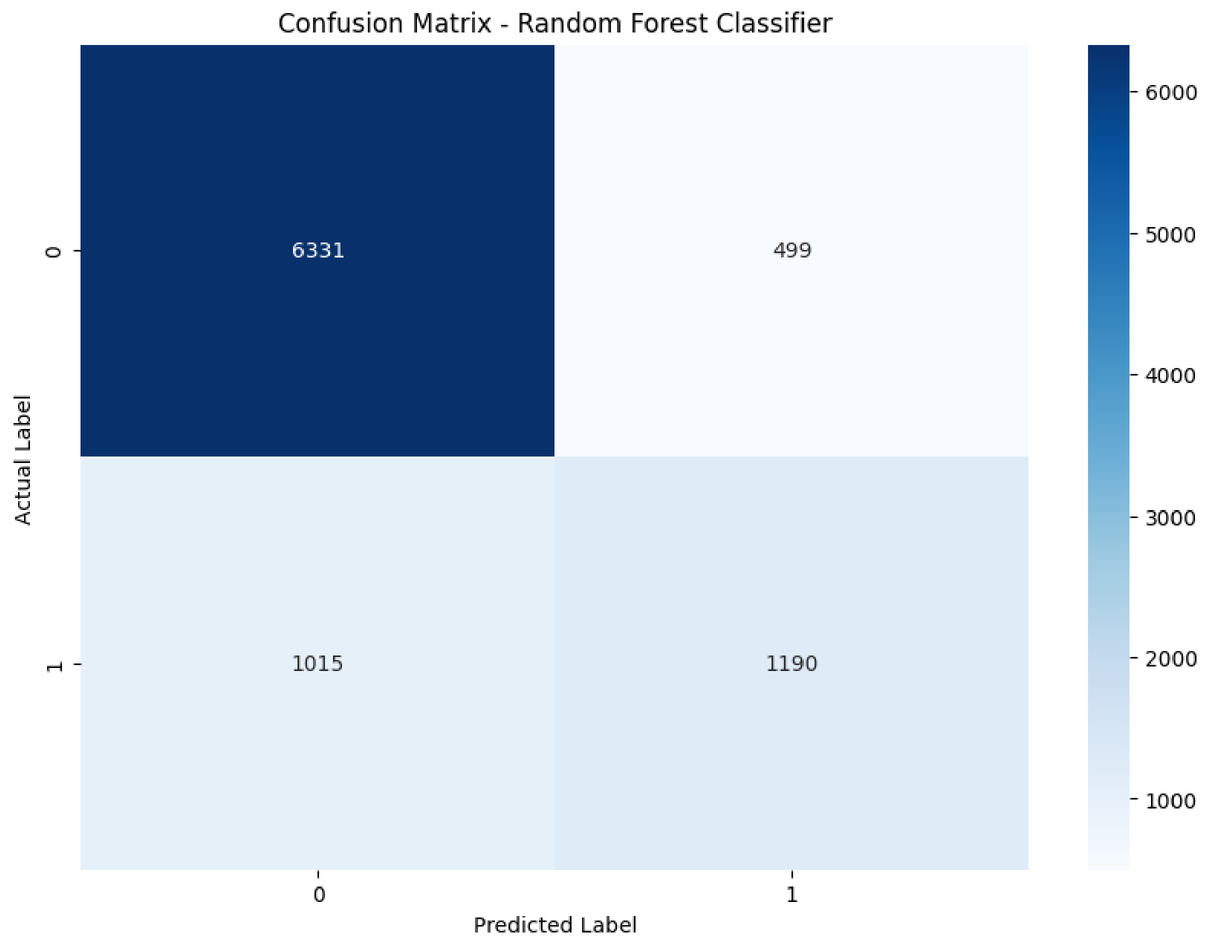
- The model has a little lower probability of ranking a randomly selected positive instance higher than a randomly selected negative instance as compared to the Random Forest.

The Random Forest and Naïve Bayes ROC Curves are displayed here. Since the Random Forest model's AUC value is larger than Naïve Bayes', it is more effective at differentiating between positive and negative data.

5.5 CONFUSION MATRIX OF NAÏVE BAYES



5.6 CONFUSION MATRIX OF RANDOM FOREST



6 LIMITATIONS

1. The technique was to oversample the minority class and undersample the majority class using RandomUnderSampler in order to rectify the class imbalance. Exercise caution while using this method, even though it can improve model performance on imbalanced datasets. Data loss and overfitting can be the outcomes of under- and oversampling techniques, respectively. Furthermore, the performance of the model may be significantly impacted by the oversampling and undersampling ratios that were employed.

2. When dealing with noisy datasets, Random Forests may overfit, especially when the number of trees (n_estimators) is large and the trees are deep. Overfitting can still occur even with ensemble techniques like feature randomization and bagging, particularly when working with noisy or redundant data or in high-dimensional feature spaces.

3. All features are regarded as equally meaningful for classification by naive Bayes classifiers. As a result, they may be susceptible to redundant or irrelevant features in the dataset. When irrelevant features are mistakenly believed to be independent of the class label, they might increase noise and impair model performance.

7 FURTHER ENHANCEMENTS

1. To obtain greater accuracy, choose other models or techniques such neural networks, K-Nearest Neighbors, and Support Vector Machine (SVM).

2. The number of trees in the forest overall, the trees' maximum depth, and the minimum number of samples required to split an internal node are just a few of the hyperparameters in random forest models that can have a significant impact on how well they work. To enhance the performance of the model, a methodical exploration of the hyperparameter space through techniques such as grid search or random searches could be helpful.
3. For categorical data, the algorithm use one-hot encoding; however, more feature engineering might be possible. This could involve looking into the links between existing features or creating new features based on domain expertise. Furthermore, feature selection techniques like feature significance ranking and recursive feature removal can help determine the most valuable aspects of the model.

8 The link to git repository

.https://github.com/manujayaRajaguru/MachineLearningCW_Income-Prediction.git

9 CODE

```
!pip install ucimlrepo
```

9.1 Data Pre-processing

9.1.1 Loading the data from Repo

```
# Importing libraries
from ucimlrepo import fetch_ucirepo
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# fetch dataset
adult = fetch_ucirepo(id=2)

# data (as pandas dataframes)
X = adult.data.features
y = adult.data.targets

# Concatenating feature and target data
data = pd.concat([X, y], axis=1)

# metadata
print(adult.metadata)

# variable information
print(adult.variables)

# Exporting loaded data to csv file
data.to_csv('adults.csv', index=False)
```

9.1.2 Cleaning the Dataset

```
data.describe()

data.info()

data['income'].value_counts()

# Replacing <=50k. to <=50k and >50k. to >50k

data["income"] = data["income"].str.replace('<=50K.', '<=50K')
data["income"] = data["income"].str.replace('>50K.', '>50K')

data['income'].value_counts()

data['sex'].value_counts()

data['native-country'].value_counts()

data['workclass'].value_counts()

data['occupation'].value_counts()

# Check NA values
data.isna().sum()

# Dropping Null Values

data = data.dropna()
data.isna().sum()

data.isin(['?']).sum()

data.drop("education-num", axis=1, inplace=True)
data

# Checking for Null values and handling

import numpy as np

data = data.replace('?', np.nan).dropna()
data.isin(['?']).sum()

data.duplicated().sum()

data.drop_duplicates(inplace=True)
data
```

```
data.shape

data.head()

# Label Encoding

from sklearn.preprocessing import LabelEncoder

# Get a list of columns with categorical data
categorical_columns = [col for col in data.columns if data[col].dtype == 'object']
target = data['income']

# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Encode categorical columns and replace original columns
for col in categorical_columns:
    data[col] = label_encoder.fit_transform(data[col])

data.head()

# Exporting Pre processed dataset
data['income'] = target
data.to_csv('cleaned_data.csv', index=False)
```

9.1.3 Data Visualization

```
# Reading the datset
c_data =  pd.read_csv('cleaned_data.csv')

# Create a bar plot for 'age' and 'income' columns using seaborn
plt.figure(figsize=(10, 6))  # Set the size of the plot
sns.barplot(x='income', y='age', hue='sex', data=c_data)  # Create the bar plot with seaborn

# Set labels and title
plt.xlabel('Income')  # Set x-axis label
plt.ylabel('Age')  # Set y-axis label
plt.title('Bar Plot of Income vs Age')  # Set title

plt.show()  # Display the bar plot
```

9.1.3.1 Histograms of Numerical features against income

```
# Drop non-numeric columns if any
```

```
numeric_data = c_data.select_dtypes(include=[float, int])

# Separate numerical features and income categories
numerical_features = numeric_data.columns
income_categories = c_data['income'].unique()

# Set up subplots
num_plots = len(numerical_features)
fig, axes = plt.subplots(nrows=num_plots, ncols=1, figsize=(10, num_plots*4))

# Plot histograms for each numerical feature against income category
for i, feature in enumerate(numerical_features):
    ax = axes[i]
    for category in income_categories:
        subset_data = numeric_data[c_data['income'] == category]
        sns.histplot(subset_data[feature], ax=ax, kde=True, label=f"Income: {category}", alpha=0.5)
    ax.set_title(f'Histogram of {feature} by Income')
    ax.legend()
    ax.set_xlabel(feature)
    ax.set_ylabel('Frequency')

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

9.1.3.2 Heat Map

```
# Plotting the Heat map
plt.figure(figsize=[10,10])

ct_counts = c_data.groupby(['education', 'income']).size()
ct_counts = ct_counts.reset_index(name = 'count')
ct_counts = ct_counts.pivot(index = 'education', columns = 'income', values = 'count').fillna(0)

sns.heatmap(ct_counts, annot = True, fmt = '.0f', cbar_kws = {'label' : 'Number of Individuals'})
plt.title('Number of People for Education Class relative to Income')
plt.xlabel('Income ($)')
plt.ylabel('Education Class')
```

9.1.3.3 Correlation Heatmap

```
# Plotting the Correlation Heatmap

numeric_cols = c_data.select_dtypes(include=['number']).columns
numeric_data = c_data[numeric_cols]

plt.figure(figsize=(10, 6))
```

```
sns.heatmap(numeric_data.corr(), annot=True, cmap='cividis')
plt.title('Correlation Heatmap')
plt.show()
```

9.2 Scaling and PCA

```
# Import Libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# Load the dataset
df = pd.read_csv('cleaned_data.csv')

df.head(10)
```

9.2.1 Standard Scaler

```
X = df.drop(['income'], axis=1) # Features
y = df['income'] # Label


scaler = StandardScaler()
X_scaler = pd.DataFrame(scaler.fit_transform(X), columns = X.columns)


X_scaler.head(10)


# Calculate the standard deviation of all columns
std_X = X_scaler.std()
print('Standard Deviation after Performing Standard Scaler: \n',std_X)

# Plot the Mean of all columns
plt.bar(range(len(mean_X)), mean_X)
plt.title("Mean after performing Standard Scaler")
plt.xlabel("Column")
plt.ylabel("Mean")
plt.show()

#Plot correlation heatmap
plt.figure(figsize = (9,8))
plt.title("Correlation Heatmap of Features")
sns.heatmap(X_scaler.corr(), cmap = 'Blues', annot = True)
plt.show()
```

9.2.2 Principal Component Analysis(PCA)

```
# Instantiate PCA
pca = PCA()

# Fit and transform the data
X_pca = pca.fit_transform(X_scaler)

# Get the explained variance ratio for each principal component
explained_variance = pca.explained_variance_ratio_

# Create a DataFrame to display attribute names and their contributions to explained variance
df_components = pd.DataFrame({'Attribute': X_scaler.columns, 'Explained Variance Ratio':
explained_variance})

print("Explained Variance Ratio for Each Attribute:\n")
print(df_components)

# Calculate the cumulative explained variance
cumulative_explained_variance = explained_variance.cumsum()

# Plot the cumulative explained variance
plt.figure(figsize=(10, 6))
plt.plot(range(1, len(cumulative_explained_variance)+1), cumulative_explained_variance, marker='o',
linestyle='--')
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Explained Variance Ratio')
plt.title('Cumulative Explained Variance Ratio vs. Number of Principal Components')
plt.grid(True)
plt.show()

pca = PCA(n_components=10)
X_pca = pca.fit_transform(X_scaler)
X_pca = pd.DataFrame(X_pca)
X_pca.head()

# Concatenate the target variable
df = pd.concat([X_pca, y], axis=1)
df

# Save the data
df.to_csv('cleaned_labelEncoded_PCA_adult.csv', index=False)
```

9.3 Naive Bayes model

```
# Importing libraries

from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd


# Reading the data from the Pre-processed file
data = pd.read_csv('cleaned_labelEncoded_PCA_adult.csv')
data.head()
```

9.3.1 NB Classification

```
# Naive Bayes Classification

# Extract features and target variable from the dataset
X = data.drop(columns=['income'])  # Features (excluding 'income' column)
y = data['income']  # Target variable

# Convert categorical variables into numerical representation (One-Hot Encoding)
X = pd.get_dummies(X)

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

# Create a Gaussian Naive Bayes classifier
nb_model = GaussianNB()

# Train the model on the training data
nb_model.fit(X_train, y_train)

# Make predictions on the test data
predictions = nb_model.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, predictions)
print("Accuracy:", accuracy)

# Generate classification report
report = classification_report(y_test, predictions)
print("Classification Report:")
print(report)
```

9.3.2 ROC Curve

```
# Importing necessary libraries
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt

# Get predicted probabilities for the positive class
y_probs = nb_model.predict_proba(X_test)[:, 1]

# Convert target variable to binary labels
y_test_binary = (y_test == '>50K').astype(int)

# Compute ROC curve
fpr, tpr, thresholds = roc_curve(y_test_binary, y_probs)

# Compute ROC AUC score
roc_auc = roc_auc_score(y_test_binary, y_probs)
print("ROC AUC Score:", roc_auc)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve (AUC = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='red', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```

9.3.3 Confusion Matrix

```
from sklearn.metrics import confusion_matrix

# Construct confusion matrix
conf_matrix = confusion_matrix(y_test, predictions)

# Display confusion matrix
print("Confusion Matrix:")
print(conf_matrix)

# Plotting the Confusion Matrix
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix - Gaussian Naive Bayes Classifier')
plt.ylabel('Actual Label')
plt.xlabel('Predicted Label')
plt.show()
```

9.4 Random Forest Model

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd

# Reading the data from the Pre-processed file
data = pd.read_csv('cleaned_labelEncoded_PCA_adult.csv')
data.head()
```

9.4.1 Classification

```
from sklearn.model_selection import GridSearchCV

# Assuming 'data' is your DataFrame containing features and target
# Drop the target column from features
X = data.drop(columns=['income'])
y = data['income']

# Convert categorical variables into numerical representation (One-Hot Encoding)
X = pd.get_dummies(X)

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

# Create a random forest classifier
rf_model = RandomForestClassifier(random_state=42)

# Parameter grid for GridSearch
parameter_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Grid search with cross-validation
```

```
grid_search = GridSearchCV(estimator=rf_model, param_grid=parameter_grid, cv=5, n_jobs=-1,
verbose=2, scoring='accuracy')
grid_search.fit(X_train, y_train)

# Best parameters from the GridSearchCV
print("Best Parameters:", grid_search.best_params_)
print("Best Score:", grid_search.best_score_)

# Train the model with the best parameters
best_rf_model = grid_search.best_estimator_

# Make predictions on the test data
predictions = best_rf_model.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, predictions)
print("Accuracy:", accuracy)

# Generate classification report
report = classification_report(y_test, predictions)
print("Classification Report:")
print(report)

# Get feature importance
feature_importance = pd.DataFrame({'Feature': X_train.columns, 'Importance':
best_rf_model.feature_importances_})
print("Feature Importance:")
print(feature_importance)
```

9.4.2 ROC Curve

```
from sklearn.preprocessing import LabelEncoder

# Encode target variable into binary labels
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)

# Get predicted probabilities for the positive class
y_probs = best_rf_model.predict_proba(X_test)[:, 1]

# Compute ROC curve
fpr, tpr, thresholds = roc_curve(y_test_encoded, y_probs)

# Compute ROC AUC score
roc_auc = roc_auc_score(y_test_encoded, y_probs)
```

```
print("ROC AUC Score:", roc_auc)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve (AUC = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='red', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```

9.4.3 Confusion Matrix

```
from sklearn.metrics import confusion_matrix

# Construct confusion matrix
conf_matrix = confusion_matrix(y_test, predictions)

# Display confusion matrix
print("Confusion Matrix:")
print(conf_matrix)

plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix - Random Forest Classifier')
plt.ylabel('Actual Label')
plt.xlabel('Predicted Label')
plt.show()
```