INFORMATICS INSTITUTE OF TECHNOLOGY

In Collaboration with

ROBERT GORDON UNIVERSITY ABERDEEN

BSc. Artificial Intelligence & Data Science

Level 05

CM 2604 MACHINE LEARNING

COURSEWORK

Module Leader: Mr. Sahan Priyanayana

M.H.Jayashan De Silva

IIT ID: 20211295

RGU ID: 2312548

Table of Contents

1. Introduction

In this study, we used large census data, and the key aim was the prediction of whether an individual earns more than $50,000 a year. However, our perspective will be on machine learning, whose capabilities are now being enabled by the advent of very sophisticated algorithms and very large datasets. The paper presents a machine learning pipeline involving data preprocessing, model selection, and evaluation to obtain predictive models holding capacity for discerning income levels from demographic attributes with accuracy. The project is managed using GIT, and an open and transparent environment is established during the development phase.

The source code is available at: https://github.com/HJayashan/Income-Prediction-with-ML.git

2. Dataset

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

The 'Census Income' dataset obtained from the UCI Machine Learning Repository consists of demographic features and income levels of individuals. It contains 15 attributes like age, work class, education, occupation etc. including the target variable 'Income'.

	name	role	type	demographic
0	age	Feature	Integer	Age
1	workclass	Feature	Categorical	Income
2	fnlwgt	Feature	Integer	None
3	education	Feature	Categorical	Education Level
4	education-num	Feature	Integer	Education Level
5	marital-status	Feature	Categorical	Other
6	occupation	Feature	Categorical	Other
7	relationship	Feature	Categorical	Other
8	race	Feature	Categorical	Race
9	sex	Feature	Binary	Sex
10	capital-gain	Feature	Integer	None
11	capital-loss	Feature	Integer	None
12	hours-per-week	Feature	Integer	None
13	native-country	Feature	Categorical	Other
14	income	Target	Binary	Income

The dataset had 48842 rows and 15 columns before preprocessing.

```
df.shape
✓ 0.0s
(48842, 15)
```

3. Corpus Preparation

3.1 Preprocessing Techniques

Effective preprocessing of data is necessary to turn raw data in an appropriate form that is suitable in building reliable machine learning models. In our project, a set of preprocessing techniques was applied to make the census data ready for model training and guarantee high integrity and quality of this data.

3.1.1 Data Cleaning

An exhaustive initial scan was done to find out any internal inconsistencies within the data, such as missing values, duplicates, null values, or any form of data corruption. It included filling missing values where appropriate or removing affected records when such missing data was nontrivial.

1. **Remove duplicate values**

```
# Removing duplicates
df = df.drop_duplicates()

# Finding duplicate values in the dataset
print(df[df.duplicated()])
```
✓ 0.0s

Output:

```
Empty DataFrame
Columns: [age, workclass, fnlwgt, education, education-num, marital-status,
occupation, relationship, race, sex, capital-gain, capital-loss, hours-per-
week, native-country, income]
   Index: []
```

2. **Remove null values**

Before removing null values, let's check for null values and print them.

```
df.nunique()
```
✓ 0.0s

```
age                  74
workclass             9
fnlwgt            28523
education            16
education-num        16
marital-status        7
occupation           15
relationship          6
```

```
race                  5
sex                   2
capital-gain        123
capital-loss         99
hours-per-week       96
native-country       42
income                4
   dtype: int64
```

```python
for column in df.columns:
    unique_values = df[column].unique()
    print(f'{column} unique values: ')
    print(unique_values)
    print('\n')
```

✓ 0.0s

```
age unique values:
[39 50 38 53 28 37 49 52 31 42 30 23 32 40 34 25 43 54 35 59 56 19 20 45
 22 48 21 24 57 44 41 29 18 47 46 36 79 27 67 33 76 17 55 61 70 64 71 68
 66 51 58 26 60 90 75 65 77 62 63 80 72 74 69 73 81 78 88 82 83 84 85 86
 87 89]

workclass unique values:
['State-gov' 'Self-emp-not-inc' 'Private' 'Federal-gov' 'Local-gov' '?'
 'Self-emp-inc' 'Without-pay' 'Never-worked' nan]

fnlwgt unique values:
[ 77516  83311 215646 ... 173449  89686 350977]

education unique values:
['Bachelors' 'HS-grad' '11th' 'Masters' '9th' 'Some-college' 'Assoc-acdm'
 'Assoc-voc' '7th-8th' 'Doctorate' 'Prof-school' '5th-6th' '10th'
 '1st-4th' 'Preschool' '12th']

education-num unique values:
[13  9  7 14  5 10 12 11  4 16 15  3  6  2  1  8]

marital-status unique values:
['Never-married' 'Married-civ-spouse' 'Divorced' 'Married-spouse-absent'
 'Separated' 'Married-AF-spouse' 'Widowed']

occupation unique values:
['Adm-clerical' 'Exec-managerial' 'Handlers-cleaners' 'Prof-specialty'
 'Other-service' 'Sales' 'Craft-repair' 'Transport-moving'
 'Farming-fishing' 'Machine-op-inspct' 'Tech-support' '?'
 'Protective-serv' 'Armed-Forces' 'Priv-house-serv' nan]

relationship unique values:
['Not-in-family' 'Husband' 'Wife' 'Own-child' 'Unmarried' 'Other-relative']

race unique values:
```

```
['White' 'Black' 'Asian-Pac-Islander' 'Amer-Indian-Eskimo' 'Other']

sex unique values:
['Male' 'Female']

capital-gain unique values:
[ 2174     0 14084  5178  5013  2407 14344 15024  7688 34095  4064  4386
  7298  1409  3674  1055  3464  2050  2176   594 20051  6849  4101  1111
  8614  3411  2597 25236  4650  9386  2463  3103 10605  2964  3325  2580
  3471  4865 99999  6514  1471  2329  2105  2885 25124 10520  2202  2961
 27828  6767  2228  1506 13550  2635  5556  4787  3781  3137  3818  3942
   914   401  2829  2977  4934  2062  2354  5455 15020  1424  3273 22040
  4416  3908 10566   991  4931  1086  7430  6497   114  7896  2346  3418
  3432  2907  1151  2414  2290 15831 41310  4508  2538  3456  6418  1848
  3887  5721  9562  1455  2036  1831 11678  2936  2993  7443  6360  1797
  1173  4687  6723  2009  6097  2653  1639 18481  7978  2387  5060  1264
  7262  1731  6612]

capital-loss unique values:
[   0 2042 1408 1902 1573 1887 1719 1762 1564 2179 1816 1980 1977 1876
 1340 2206 1741 1485 2339 2415 1380 1721 2051 2377 1669 2352 1672  653
 2392 1504 2001 1590 1651 1628 1848 1740 2002 1579 2258 1602  419 2547
 2174 2205 1726 2444 1138 2238  625  213 1539  880 1668 1092 1594 3004
 2231 1844  810 2824 2559 2057 1974  974 2149 1825 1735 1258 2129 2603
 2282  323 4356 2246 1617 1648 2489 3770 1755 3683 2267 2080 2457  155
 3900 2201 1944 2467 2163 2754 2472 1411 1429 3175 1510 1870 1911 2465
 1421]

hours-per-week unique values:
[40 13 16 45 50 80 30 35 60 20 52 44 15 25 38 43 55 48 58 32 70  2 22 56
 41 28 36 24 46 42 12 65  1 10 34 75 98 33 54  8  6 64 19 18 72  5  9 47
 37 21 26 14  4 59  7 99 53 39 62 57 78 90 66 11 49 84  3 17 68 27 85 31
 51 77 63 23 87 88 73 89 97 94 29 96 67 82 86 91 81 76 92 61 74 95 79 69]

native-country unique values:
['United-States' 'Cuba' 'Jamaica' 'India' '?' 'Mexico' 'South'
 'Puerto-Rico' 'Honduras' 'England' 'Canada' 'Germany' 'Iran'
 'Philippines' 'Italy' 'Poland' 'Columbia' 'Cambodia' 'Thailand' 'Ecuador'
 'Laos' 'Taiwan' 'Haiti' 'Portugal' 'Dominican-Republic' 'El-Salvador'
 'France' 'Guatemala' 'China' 'Japan' 'Yugoslavia' 'Peru'
 'Outlying-US(Guam-USVI-etc)' 'Scotland' 'Trinadad&Tobago' 'Greece'
 'Nicaragua' 'Vietnam' 'Hong' 'Ireland' 'Hungary' 'Holand-Netherlands' nan]

income unique values:
['<=50K' '>50K' '<=50K.' '>50K.']
```

Then we replace '?' with NaN.

```python
# Replace '?' with NaN
df.replace('?', pd.NA, inplace=True)
```
✓ 0.0s

Then we remove the values with ‘NaN’

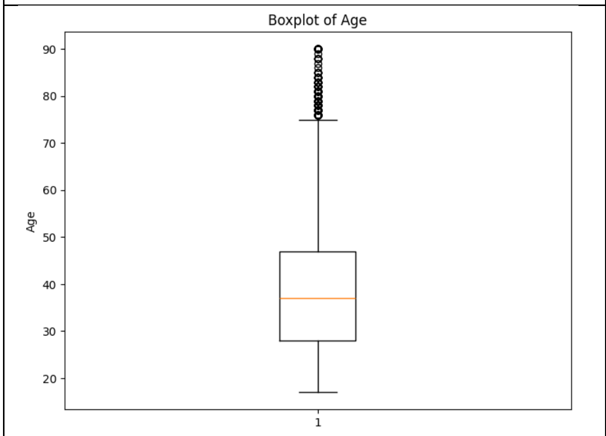
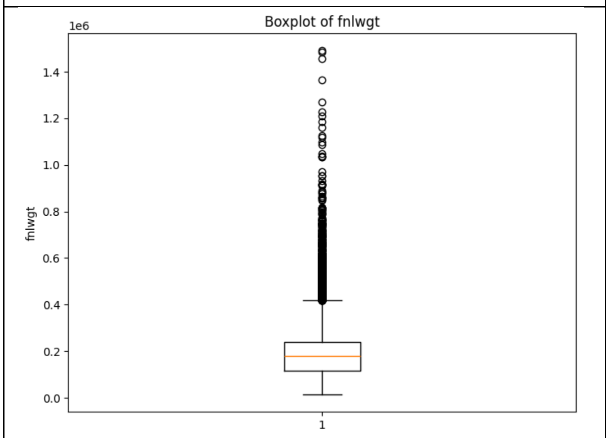
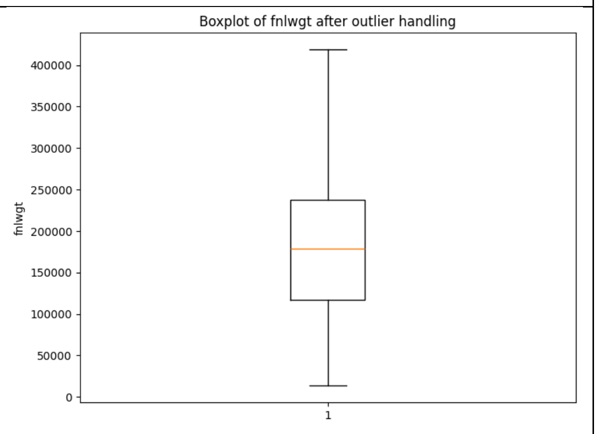
```python
# Remove NaN values
df.dropna(inplace=True)
```
✓ 0.0s

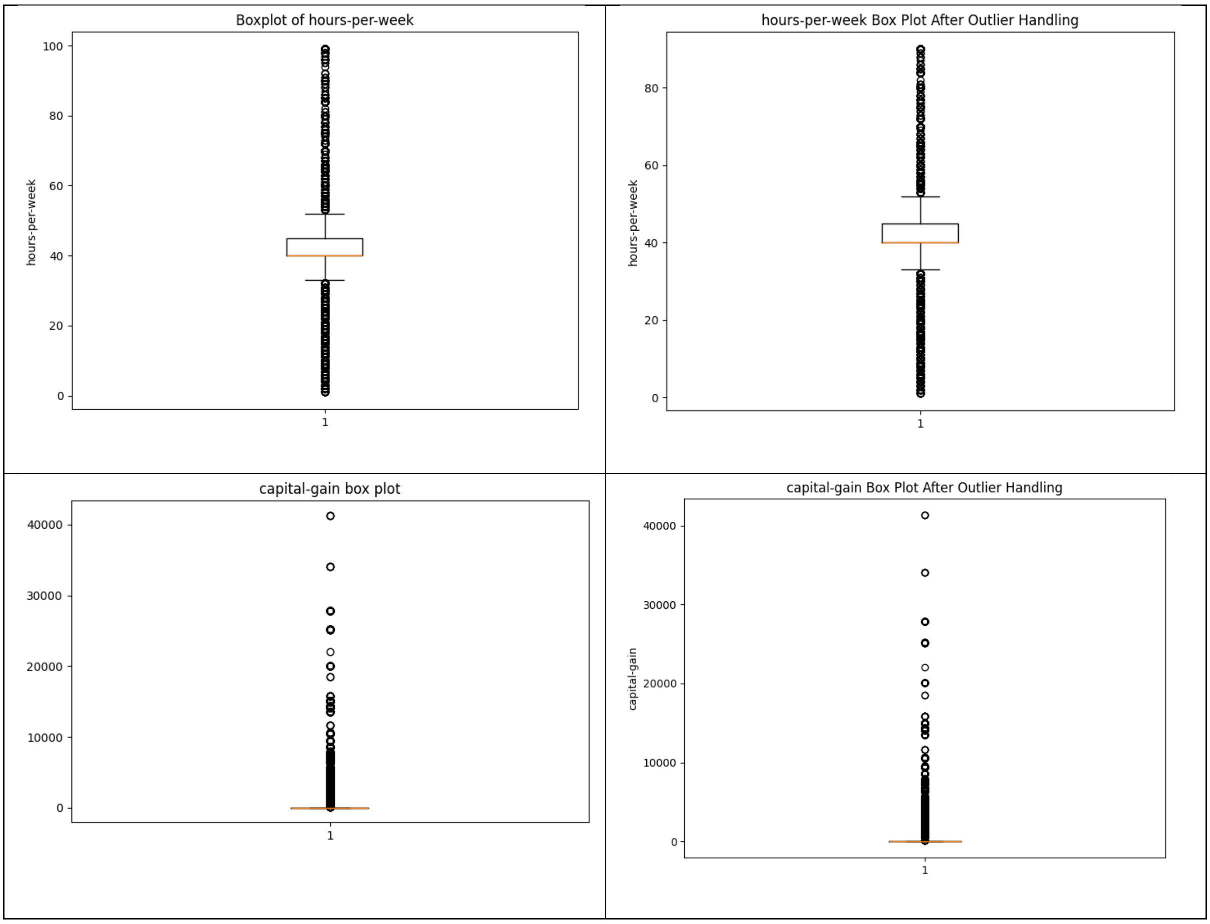
Up next we replace the income values which has the ‘.’

```python
# Replacing the income values which has the .
df['income'].replace({'<=50K.': '<=50K', '>50K.': '>50K'}, inplace=True)
```
✓ 0.0s

3.1.2 Outlier Handling

We have used statistical methods; for example, the Interquartile Range (IQR), through which we identified and removed the outliers of the important features.

<table>
<tr><th>Before</th><th>After</th></tr>
<tr><td>
</td><td>
</td></tr>
<tr><td>
</td><td>
</td></tr>
</table>



The below is the code used to handle outliers of ‘Age.’

```
# Calculate quartiles
Q1 = np.percentile(df['age'], 25)
Q3 = np.percentile(df['age'], 75)

# IQR
IQR = Q3 - Q1

# Outlier bounds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Identify outliers
outliers = df[(df['age'] < lower_bound) | (df['age'] > upper_bound)]

# Count the number of outliers
num_outliers = len(outliers)

print("Number of outliers:", num_outliers)
```

✓ 0.0s

```
Number of outliers: 268
```

```python
# Apply Winsorization to replace outliers
df['age'] = np.where(df['age'] < lower_bound, lower_bound, df['age'])
df['age'] = np.where(df['age'] > upper_bound, upper_bound, df['age'])
```
✓ 0.0s

Outliers of 'fnlwgt', 'Hours-per-week', 'Capital-gain' were also handled using a similar codes to the above code.

3.2 Feature Encoding

Feature encoding is used to convert the textual or categorical data into numerical format that algorithms can interpret. Label encoding assigns a unique integer to each category within a feature. The process was as follows:

```python
label_encoder = LabelEncoder()

encoding_columns = ['workclass', 'education','education-num', 'marital-status', 'occupation',
                    'relationship', 'race', 'sex', 'native-country', 'income']

for col in enc  (variable) label_encoder: LabelEncoder
    df[col] = label_encoder.fit_transform(df[col])
df.head()
```
✓ 0.0s

This approach is straightforward and effective, especially for datasets with a manageable number of categories. By converting textual and categorical information into a numerical format, we ensured that our dataset was fully compatible with the machine learning algorithms, facilitating their ability to learn from the data and make accurate predictions.

3.3 Data Normalization

```python
X = pp_data[['marital-status', 'education-num', 'relationship', 'sex', 'age', 'capital-gain']]
y = pp_data['income']
```
✓ 0.0s

```python
scaler = StandardScaler()
X = pd.DataFrame(scaler.fit_transform(X), columns = X.columns)
```
✓ 0.0s

The data was normalized using a standard scaler. This is the process of transforming the data such that the mean becomes zero and the standard deviation becomes one. It is done by subtracting the mean and dividing by the standard deviation for each feature.

3.4 Train/ Test Split

This would basically split the data into two parts: one for training and the other for testing the sets. This would basically allow the training of models in one set of the data and, in turn, allow for testing and validation using a totally different, unobserved subset of the data. Ensuring that the model generalizes well to new data is critical for its effectiveness in real-world applications.

I split the dataset using the train_test_split function from scikit-learn. The data is split such that 80% of the data belongs to the training set, and otherwise to the test set. This common splitting ratio helped us keep a balance in sufficiency with respect to the training data and enabled building a very robust test set. Below is a specific code snippet used for this purpose:

```
#splitting train, test variables
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
✓ 0.0s
```

```
X_train.shape, X_test.shape
✓ 0.0s
((36133, 6), (9034, 6))
```

```
X_train.head()
✓ 0.0s
```

	marital-status	education-num	relationship	sex	age	capital-gain
16570	-1.722256	-0.43865	-0.258480	-1.441101	0.798973	-0.23358
21745	-1.722256	-0.43865	-0.258480	0.693914	1.561715	-0.23358
14840	-0.389509	-0.43865	-0.884407	0.693914	0.112504	-0.23358
35253	-0.389509	-0.43865	-0.884407	0.693914	0.188778	-0.23358
40482	-0.389509	-0.43865	-0.884407	0.693914	0.570150	-0.23358

After the split, the training set had X_train.shape[0] instances, and the test set had X_test.shape[0] instances. This provides very wide training, where the model can learn sufficiently from the data, and tremendous testing scope should be wide enough to ensure that it is tested sufficiently so that it gives an estimation of performance with a high level of confidence.

4. Solution Methodology

4.1 Model Selection

Naïve Bayes and Random Forest Classifier algorithms were used to train the model.

4.1.1 Naïve Bayes Classifier

Naïve Bayes is a model that is simple, efficient, and effective for classification tasks, especially when the assumption of independence among the predictors holds reasonably well. The Naïve Bayes algorithm has particularly shown a lot of success and, therefore, fame in the context of text classification tasks but has also shown promising results in several other domains. Probabilistic in nature, Naive Bayes influences the interpretability brought out in the final results through prediction and, hence, providing meaningful probabilities.

```
# Creating a Gaussian Classifier
model = GaussianNB()

# Train the model using the training sets
gnb = model.fit(X_train,y_train)

# Predictions on the test set
gnb_predictions = gnb.predict(X_test)

# Evaluate Naïve Bayes Classifier
print("Naïve Bayes Classifier:")
print("Accuracy:", accuracy_score(y_test, gnb_predictions))
print("Classification Report:")
print(classification_report(y_test, gnb_predictions))

# Scores on training and test sets
print("Training Set Accuracy:", model.score(X_train, y_train))
print("Test Set Accuracy:", model.score(X_test, y_test))
```
✓ 0.0s

4.1.2 Random Forest Classifier

Random Forest is an ensemble learning method, combining many decision trees that can be called strong models. In other words, it is a learning model based on learning from several decision trees to minimize the risk of overfitting in capturing complex patterns within the data. This will allow the work for big data sets where many features are there—just like the data of the census. Here, different demographic variables will, in fact, show some significant role in the prediction of income.

```
# Initialize the classifier
random= RandomForestClassifier(random_state=42)

# Fit GridSearchCV
rf_model = random.fit(X_train, y_train)

y_pred = rf_model.predict(X_test)

# Evaluate Naïve Bayes Classifier
print("Random Forest Classifier:")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Scores on training and test sets
print("Training Set Accuracy:", random.score(X_train, y_train))
print("Test Set Accuracy:", random.score(X_test, y_test))
```
✓ 3.4s

5. Evaluation Criteria

The model was evaluated based on the criteria like accuracy, precision, recall, F1-score.

Accuracy: The ratio of correctly predicted instances to the total number of instances.

Precision value: Used to measure the accuracy of the positive predictions made by the model.

$$Precision = \frac{TP}{TP + FP}$$

Recall: Used to measure the ability of the model to correctly identify all positive instances in the dataset.

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

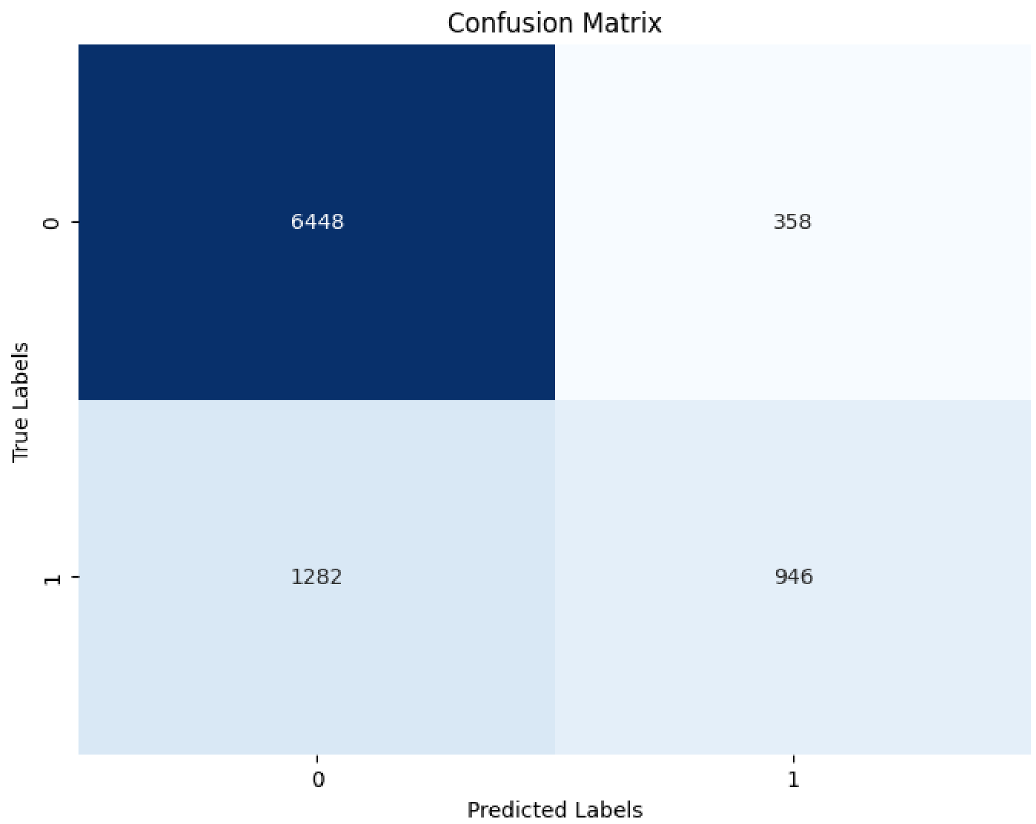
F1-score: The harmonic mean of precision and recall, providing a balance between the two metrics.

5.1 Naïve Bayes Classifier

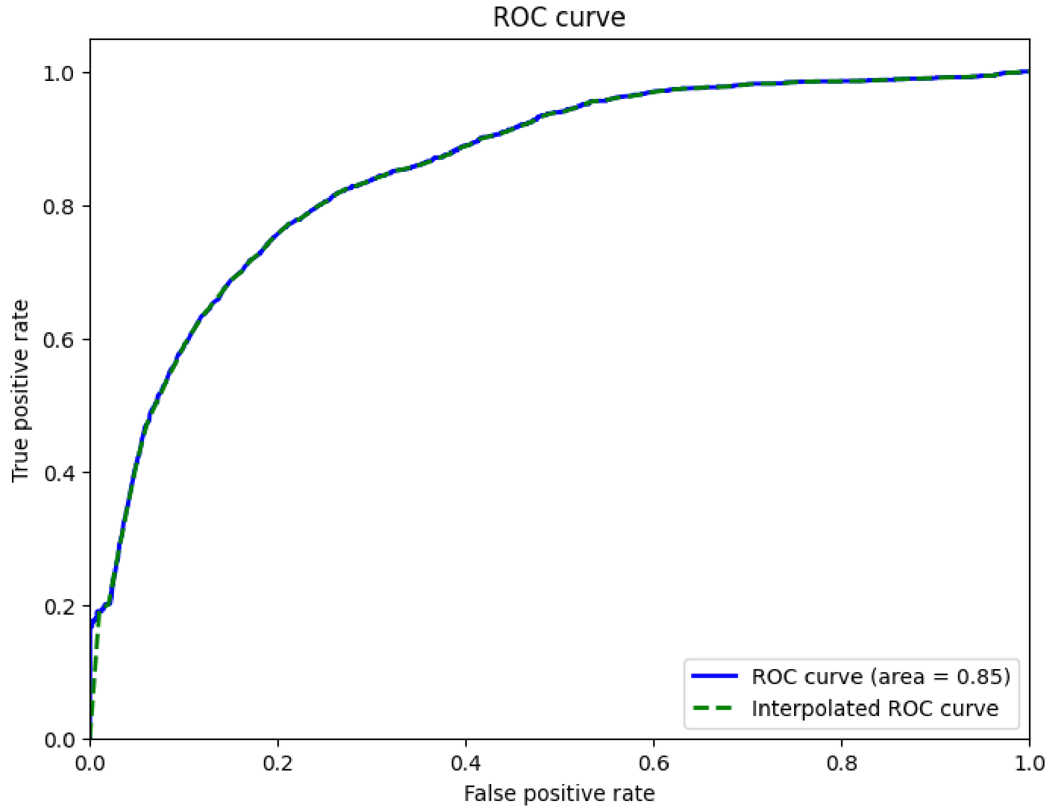
5.1.1 Classification Report

```
Naïve Bayes Classifier:
Accuracy: 0.8184635820234669
Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.95      0.89      6806
           1       0.73      0.42      0.54      2228

    accuracy                           0.82      9034
   macro avg       0.78      0.69      0.71      9034
weighted avg       0.81      0.82      0.80      9034

Training Set Accuracy: 0.8143802064594692
Test Set Accuracy: 0.8184635820234669
```

5.1.2 Confusion Matrix



5.1.3 ROC Curve

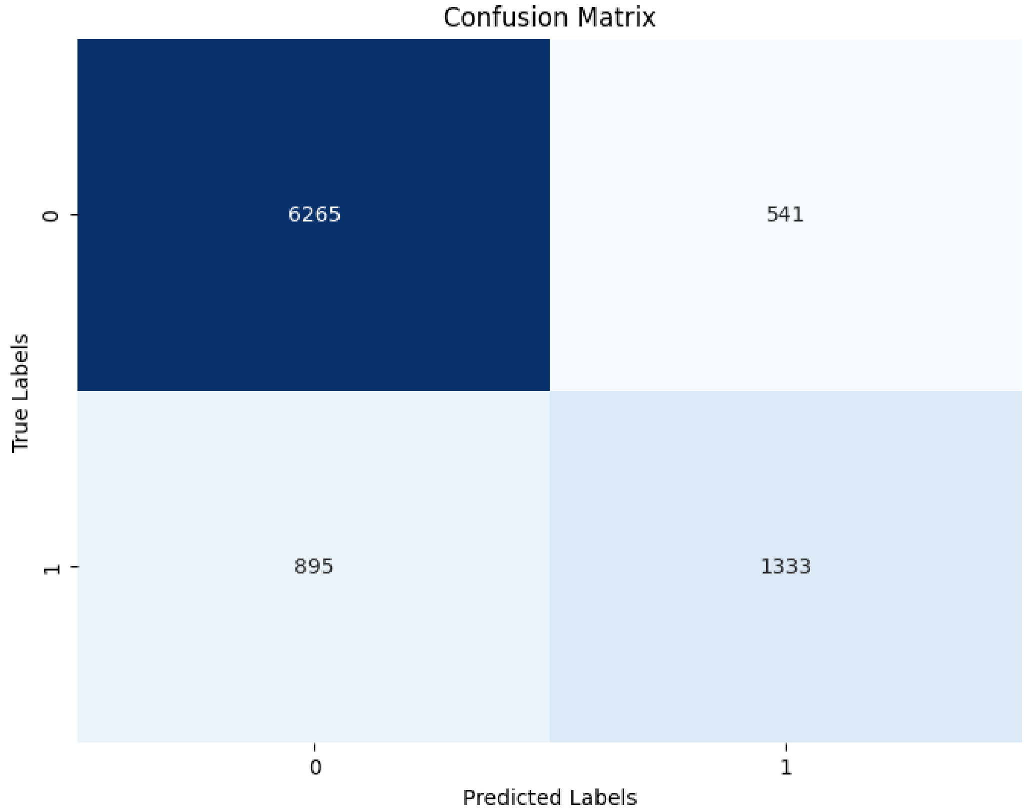


5.2 Random Forest Classifier

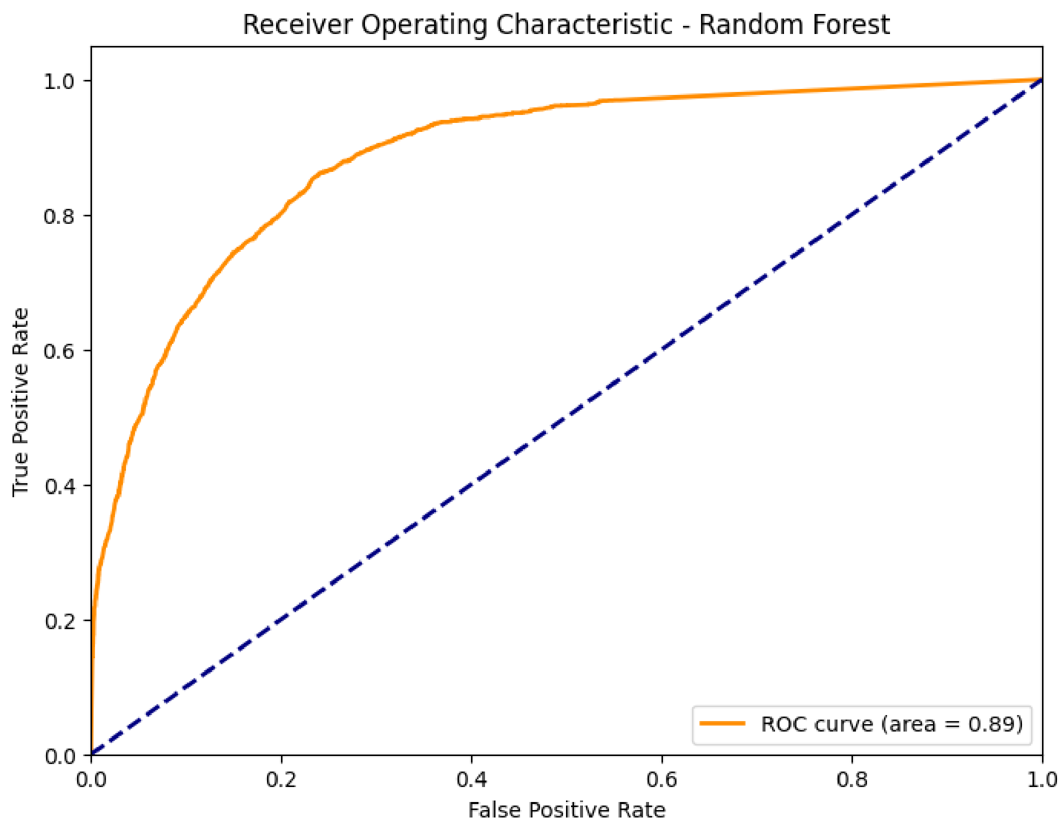
5.2.1 Classification Report

```
Random Forest Classifier:
Accuracy: 0.841044941332743
Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.92      0.90      6806
           1       0.71      0.60      0.65      2228

    accuracy                           0.84      9034
   macro avg       0.79      0.76      0.77      9034
weighted avg       0.83      0.84      0.84      9034

Training Set Accuracy: 0.8644452439598151
Test Set Accuracy: 0.841044941332743
```

5.2.2 Confusion Matrix



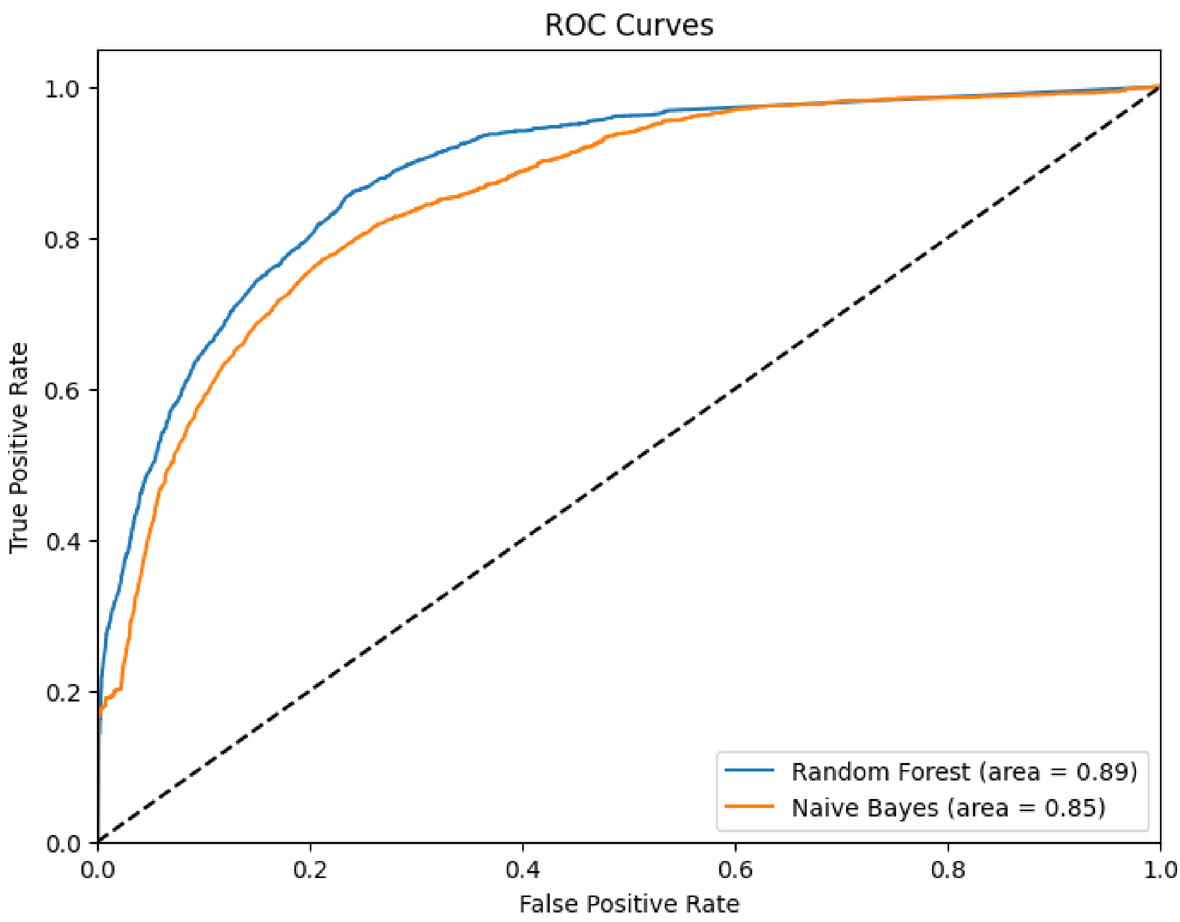
5.2.3 ROC Curve



6. Comparing the Models

```
Random Forest Classifier:
Accuracy: 0.841044941332743
Precision: 0.7113127001067235
Recall: 0.598294434470377
F1 Score: 0.6499268649439298
ROC AUC: 0.8854822890985936

Naive Bayes Classifier:
Accuracy: 0.8184635820234669
Precision: 0.7254601226993865
Recall: 0.4245960502692998
F1 Score: 0.535673839184598
ROC AUC: 0.8543754758052221
```

6.1 Comparing the ROC curves



7. Appendix: Source Code

7.1 Data Preprocessing.ipynb

```
!pip install ucimlrepo

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

import warnings
warnings.filterwarnings("ignore")
```

1. Importing the dataset

```
from ucimlrepo import fetch_ucirepo

# fetch dataset
adult = fetch_ucirepo(id=2)

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

# metadata
print(adult.metadata)

# variable information
print(adult.variables)

df = pd.concat([X,y],axis=1)
df
```

2. Data Exploration

```
# No of rows in dataset before preprocessing
len(df)

df.shape

df.info()

df.isna().sum()

df.describe()
```

3. Data Cleaning

Remove duplicate values

```
# Removing duplicates
df = df.drop_duplicates()

# Finding duplicate values in the dataset
print(df[df.duplicated()])

df.shape
```

Remove null values

```
df.nunique()

for column in df.columns:
    unique_values = df[column].unique()
    print(f'{column} unique values: ')
    print(unique_values)
    print('\n')

# Replace '?' with NaN
df.replace('?', pd.NA, inplace=True)

# Check for unique values again to make sure that we get rid of all the unnecessary things
for column in df:
    print(column)
    print(df[column].unique())
    print('\n')

# Remove NaN values
df.dropna(inplace=True)

# Check for unique values again to make sure that we get rid of all the unnecessary things
for column in df:
    print(column)
    print(df[column].unique())
    print('\n')

# Replacing the income values which has the .
df['income'].replace({'<=50K.': '<=50K', '>50K.': '>50K'}, inplace=True)

# Check for unique values again to make sure that we get rid of all the unnecessary things
for column in df:
    print(column)
    print(df[column].unique())
    print('\n')

# checking the count of NaN in all columns.
```

```
df.isna().sum()

df.info()

df.shape

df

# Removing duplicates
df = df.drop_duplicates()

# Finding duplicate values in the dataset
print(df[df.duplicated()])


#### Age
plt.figure(figsize=(10, 6))
plt.hist(df['age'], bins=20, color='skyblue', edgecolor='black')
plt.title('Distribution of Age')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()

#### Workclass distribution
# Workclass Distribution
plt.figure(figsize=(8, 6))
plt.title("workclass distribution in the dataset")
sns.histplot(df.workclass,label=column)

#### fnlwgt distribution
# fnlwgt
plt.figure(figsize=(8, 6))
sns.histplot(df['fnlwgt'], kde=False)
plt.xlabel('fnlwgt')
plt.ylabel('Frequency')
plt.title('Distribution of Weighting fnlwgt')
plt.show()

#### Marital distribution
# Marital Status Distribution
plt.figure(figsize=(16, 6))
plt.title("Marital Status Distribution in the dataset")
sns.histplot(df['marital-status'], label=column)

#### Race distribution
# Race Distribution
plt.figure(figsize=(12, 6))
plt.title("Race Distribution in the dataset")
sns.histplot(df['race'], label=column)
```

Sex distribution

```
# Sex Distribution
plt.figure(figsize=(8, 6))
plt.title("Race Distribution in the dataset")
sns.histplot(df['sex'], label=column)
```

4. Remove outliers

```
df.describe()
```

Age

```
# Box plot for age before outlier handling
plt.figure(figsize=(8, 6))
plt.boxplot(df['age'])
plt.title('Boxplot of Age')
plt.ylabel('Age')
plt.show()

# Calculate quartiles
Q1 = np.percentile(df['age'], 25)
Q3 = np.percentile(df['age'], 75)

# IQR
IQR = Q3 - Q1

# Outlier bounds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Identify outliers
outliers = df[(df['age'] < lower_bound) | (df['age'] > upper_bound)]

# Count the number of outliers
num_outliers = len(outliers)

print("Number of outliers:", num_outliers)

# Apply Winsorization to replace outliers
df['age'] = np.where(df['age'] < lower_bound, lower_bound, df['age'])
df['age'] = np.where(df['age'] > upper_bound, upper_bound, df['age'])

# Box plot for age after outlier handling
plt.figure(figsize=(8, 6))
plt.boxplot(df['age'])
plt.title('Boxplot of Age after outlier handling')
plt.ylabel('Age')
plt.show()
```

fnlwgt

```
# Box plot for fnlwgt before outlier handling
plt.figure(figsize=(8, 6))
plt.boxplot(df['fnlwgt'])

# Add title and labels
plt.title('Boxplot of fnlwgt')
plt.ylabel('fnlwgt')

# Show the plot
plt.show()

# Calculate quartiles
Q1 = np.percentile(df['fnlwgt'], 25)
Q3 = np.percentile(df['fnlwgt'], 75)

# IQR
IQR = Q3 - Q1

# Outlier bounds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Identify outliers
outliers = df[(df['fnlwgt'] < lower_bound) | (df['fnlwgt'] > upper_bound)]

# Count the number of outliers
num_outliers = len(outliers)

print("Number of outliers:", num_outliers)

# Apply Winsorization to replace outliers
df['fnlwgt'] = np.where(df['fnlwgt'] < lower_bound, lower_bound, df['fnlwgt'])
df['fnlwgt'] = np.where(df['fnlwgt'] > upper_bound, upper_bound, df['fnlwgt'])

# Box plot for fnlwgt after outlier handling
plt.figure(figsize=(8, 6))
plt.boxplot(df['fnlwgt'])

# Add title and labels
plt.title('Boxplot of fnlwgt after outlier handling')
plt.ylabel('fnlwgt')

# Show the plot
plt.show()
```

Hours per Week

```
# Box plot for Hours per week before outlier handling
plt.figure(figsize=(8, 6))
plt.boxplot(df['hours-per-week'])
```

```
plt.title('Boxplot of hours-per-week')
plt.ylabel('hours-per-week')
plt.show()

# Apply Winsorization to replace outliers
lower_bound = 0
upper_bound = 90
df['hours-per-week'] = np.where(df['hours-per-week'] < lower_bound, lower_bound, df['hours-per-week'])
df['hours-per-week'] = np.where(df['hours-per-week'] > upper_bound, upper_bound, df['hours-per-week'])

# Box plot for Hours per week after outlier handling
plt.figure(figsize=(8, 6))
plt.boxplot(df["hours-per-week"])
plt.title("hours-per-week Box Plot After Outlier Handling")
plt.ylabel('hours-per-week')
plt.show()
```

Capital gain

```
plt.figure(figsize=(8, 5))
plt.boxplot(df["capital-gain"])
plt.title("capital-gain box plot")
plt.show()

# Handle outliers by replacing values above 60000 with the mean of values below that threshold
outlier_threshold = 60000

capital_gain_mean_below_threshold = df.loc[df["capital-gain"] <= outlier_threshold, "capital-gain"].mean()
df.loc[df["capital-gain"] > outlier_threshold, "capital-gain"] = capital_gain_mean_below_threshold

# Box plot for Hours per week after outlier handling
plt.figure(figsize=(8, 6))
plt.boxplot(df["capital-gain"])
plt.title("capital-gain Box Plot After Outlier Handling")
plt.ylabel('capital-gain')
plt.show()
```

Capital loss

```
plt.figure(figsize=(8, 5))
plt.boxplot(x=df["capital-loss"])
plt.title("capital-loss box plot")
plt.show()
```

5. Data Visualization

Income above and belowe 50K

```
fig,axes = plt.subplots(1,1,figsize=(8,5))
sns.countplot(data = df, x='income')
plt.title('Income Above and Below 50K',fontsize=15)
plt.xlabel('Income',fontsize=15)
plt.ylabel('No. of people',fontsize=15)
plt.show()
```

Occupation vs income count

```
fig, axes = plt.subplots(1,1,figsize=(30,10))
sns.countplot(data=df,x='occupation',hue='income')
plt.title('Occupation vs Income')
plt.xlabel('Occupation')
plt.ylabel('No.of people')
plt.show()
```

sex vs income count

```
fig, axes = plt.subplots(1,1,figsize=(8,5))
sns.countplot(data=df,x='sex',hue='income')
plt.title('Sex vs Income',fontsize=15)
plt.xlabel('Sex',fontsize=15)
plt.ylabel('No.of people',fontsize=15)
plt.show()

#checking duplicates again and removing them
# Removing duplicates
df = df.drop_duplicates()

# Finding duplicate values in the dataset
print(df[df.duplicated()])
```

6. Encoding categorical values

```
df.info()

import os
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler

label_encoder = LabelEncoder()

encoding_columns = ['workclass', 'education','education-num', 'marital-status', 'occupation',
                'relationship', 'race', 'sex', 'native-country', 'income']

for col in encoding_columns:
    df[col] = label_encoder.fit_transform(df[col])

df.head()

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

```
y = df['income'] #target variable

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

X.head(5)

plt.figure(figsize = (18,15))
plt.title("Correlation between different features of the dataset", fontsize = 18, fontweight = 'bold')
sns.heatmap(X.corr(), cmap = 'Blues_r', annot = True)

# Comparing the correlation to the income
correlation_matrix = df.corr()['income'].sort_values(ascending=False)

print(correlation_matrix)


directory = 'PreprocessedData'

if not os.path.exists(directory):
    os.makedirs(directory)

csv_path = os.path.join(directory, 'Preprocessed.csv')

df.to_csv(csv_path, index=False)
print(f'CSV file saved: {csv_path}')
```

7.2 Model.ipynb

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

import warnings
warnings.filterwarnings('ignore')

pp_data = pd.read_csv('PreprocessedData/Preprocessed.csv')
pp_data
```

Normalization (Standard scaler)
```
X = pp_data[['marital-status', 'education-num', 'relationship', 'sex', 'age', 'capital-gain']]
y = pp_data['income']

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

Train/Test split
```
#splitting train, test variables
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)

X_train.shape, X_test.shape

X_train.head()
```

Model Training

NAÏVE BAYES

```
# Creating a Gaussian Classifier
model = GaussianNB()

# Train the model using the training sets
gnb = model.fit(X_train,y_train)

# Predictions on the test set
gnb_predictions = gnb.predict(X_test)

# Evaluate Naïve Bayes Classifier
print("Naïve Bayes Classifier:")
print("Accuracy:", accuracy_score(y_test, gnb_predictions))
print("Classification Report:")
print(classification_report(y_test, gnb_predictions))

# Scores on training and test sets
print("Training Set Accuracy:", model.score(X_train, y_train))
print("Test Set Accuracy:", model.score(X_test, y_test))
```

#ROC curve
```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
import numpy as np

# predict probabilities
nb_prob = model.predict_proba(X_test)[:, 1]

# calculate the ROC curve
fpr, tpr, thresholds = roc_curve(y_test, nb_prob)
```

```
# calculate area under curve
roc_auc = auc(fpr, tpr)

# plot the ROC curve
plt.figure(figsize=(8,6))
plt.plot(fpr, tpr,color='blue', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.legend(loc='lower right')

# approximate roc curve with straight lines

num_points = 100
fpr_interp = np.linspace(0, 1, num_points)
tpr_interp = np.interp(fpr_interp, fpr, tpr)

plt.plot(fpr_interp, tpr_interp, color='green', lw=2, linestyle='--', label='Interpolated ROC
curve')
plt.legend(loc='lower right')

plt.show()
```

 # Calculate confusion matrix

```
conf_matrix = confusion_matrix(y_test, gnb_predictions)

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

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.title("Confusion Matrix")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()
```

Random Forest

```
# Initialize the classifier
random= RandomForestClassifier(random_state=42)

# Fit GridSearchCV
rf_model = random.fit(X_train, y_train)
```

```
y_pred = rf_model.predict(X_test)

# Evaluate Naïve Bayes Classifier
print("Random Forest Classifier:")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Scores on training and test sets
print("Training Set Accuracy:", random.score(X_train, y_train))
print("Test Set Accuracy:", random.score(X_test, y_test))
```

#ROC curve

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Predict probabilities for the positive class (assuming the positive class is labeled as '1')
rf_prob = rf_model.predict_proba(X_test)[:, 1]

# Calculate the ROC curve
fpr_rf, tpr_rf, thresholds_rf = roc_curve(y_test, rf_prob)

# Calculate area under the curve
roc_auc_rf = auc(fpr_rf, tpr_rf)

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr_rf, tpr_rf, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % 
roc_auc_rf)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')  # Diagonal 45 degree line
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 - Random Forest')
plt.legend(loc="lower right")
plt.show()
```

 # Confusion Matrix

```
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.title("Confusion Matrix")
plt.xlabel("Predicted Labels")
```

```
plt.ylabel("True Labels")
plt.show()
```

Comparing the models

```
# Initialize classifiers
from sklearn.metrics import f1_score, precision_score, recall_score, roc_auc_score


random_forest = RandomForestClassifier(random_state=42)
naive_bayes = GaussianNB()

# Train Random Forest classifier
rf_model = random_forest.fit(X_train, y_train)
rf_predictions = rf_model.predict(X_test)

# Train Naive Bayes classifier
nb_model = naive_bayes.fit(X_train, y_train)
nb_predictions = nb_model.predict(X_test)

# Evaluate Random Forest Classifier
print("Random Forest Classifier:")
print("Accuracy:", accuracy_score(y_test, rf_predictions))
print("Precision:", precision_score(y_test, rf_predictions))
print("Recall:", recall_score(y_test, rf_predictions))
print("F1 Score:", f1_score(y_test, rf_predictions))
print("ROC AUC:", roc_auc_score(y_test, rf_model.predict_proba(X_test)[:,1]))

# Evaluate Naive Bayes Classifier
print("\nNaive Bayes Classifier:")
print("Accuracy:", accuracy_score(y_test, nb_predictions))
print("Precision:", precision_score(y_test, nb_predictions))
print("Recall:", recall_score(y_test, nb_predictions))
print("F1 Score:", f1_score(y_test, nb_predictions))
print("ROC AUC:", roc_auc_score(y_test, nb_model.predict_proba(X_test)[:,1]))
```

#ROC curves

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Assume X_train, X_test, y_train, y_test are already defined and contain your dataset

# Initialize and fit the Random Forest classifier
rf = RandomForestClassifier(random_state=42)
rf.fit(X_train, y_train)
# Predict probabilities for the test set
rf_probs = rf.predict_proba(X_test)[:, 1]
```

```
# Initialize and fit the Naive Bayes classifier
gnb = GaussianNB()
gnb.fit(X_train, y_train)
# Predict probabilities for the test set
gnb_probs = gnb.predict_proba(X_test)[:, 1]

# Calculate ROC curve and ROC area for Random Forest
fpr_rf, tpr_rf, _ = roc_curve(y_test, rf_probs)
roc_auc_rf = auc(fpr_rf, tpr_rf)

# Calculate ROC curve and ROC area for Naive Bayes
fpr_gnb, tpr_gnb, _ = roc_curve(y_test, gnb_probs)
roc_auc_gnb = auc(fpr_gnb, tpr_gnb)

# Plot the ROC curves
plt.figure(figsize=(8, 6))
plt.plot(fpr_rf, tpr_rf, label='Random Forest (area = %0.2f)' % roc_auc_rf)
plt.plot(fpr_gnb, tpr_gnb, label='Naive Bayes (area = %0.2f)' % roc_auc_gnb)
plt.plot([0, 1], [0, 1], 'k--')  # Dashed diagonal line
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curves')
plt.legend(loc="lower right")
plt.show()
```