**INFORMATICS INSTITUTE OF TECHNOLOGY**

**In Collaboration with**

**ROBERT GORDON UNIVERSITY ABERDEEN**

**Artificial Intelligence and Data Science**

**Module Leader: Sahan Priyanayana**

**CM2604 - Machine Learning**

**Assignment Type: Individual Coursework**

**Student Name: Vanuja Thihansith Sooriyaarachchi**

**IIT ID - 20222408**

**RGU ID - 2311130**

© The copyright for this project and all its associated products resides with Informatics Institute of Technology

# Table of Contents

[Table of Contents 2](#_Toc162643600)

[Abstract 3](#_Toc162643601)

[Acknowledgement 4](#_Toc162643602)

[Introduction 5](#_Toc162643603)

[Corpus Preparation 6](#_Toc162643604)

[Dataset Description 6](#_Toc162643605)

[Data Preprocessing 6](#_Toc162643606)

[Data Columns with Unsuitable Values 6](#_Toc162643607)

[Data Columns with No Values 7](#_Toc162643608)

[Outlier Handling 7](#_Toc162643609)

[Correlation Matrix 9](#_Toc162643610)

[Solution Methodology 11](#_Toc162643611)

[Model Selection 11](#_Toc162643612)

[Model Implementation 11](#_Toc162643613)

[Evaluation Criteria 12](#_Toc162643614)

[Model Evaluation 13](#_Toc162643615)

[Experimental Results 15](#_Toc162643616)

[Limitations and Further Enhancements 19](#_Toc162643617)

[Definitions 19](#_Toc162643618)

[Appendix: Source Code 20](#_Toc162643619)

[Comparing models 28](#_Toc162643620)

[Accuracy 28](#_Toc162643621)

[Classification Report 28](#_Toc162643622)

[ROC Curves for Naïve Bayes and Random Forest 28](#_Toc162643623)

[Reference 29](#_Toc162643624)

# Abstract

This is the research project in the Machine Learning module (CM2604) with a view of analyzing in depth basic Machine Learning Models. Led by Mr. Sahan Priyanayana, the focus takes the path of two leading models in this area.

Models: The Random Forest Classifier and the Naïve Bayes Classifier. These models are carefully built and tested with the most common metrics for evaluation: accuracy, precision, recall, and F1 score. The study empirically determines each model's performance in making predictions for the income level and details its strengths and weaknesses in each of the methods used. The conclusion of the study, with perceptive suggestions for different areas that can further be investigated for improvements down the avenue of machine learning for income predictions.

# Acknowledgement

I would like to express my sincere gratitude to Mr. Sahan Priyanayana and Miss Sachithra Vinodani for their invaluable advice, inspiration, and help during this project’s completion. Their knowledge and perceptions have greatly influenced how this inquiry has proceeded. I express my gratitude to my peers and associates for their collaboration and helpful criticism throughout the creation and assessment of machine learning models. Moreover, I would like to express my gratitude to the developers of the scikit-learn framework and the ‘Census Income’ dataset for offering the resources and instruments required to carry out this study. Finally, I want to sincerely thank my family and friends for their unwavering understanding and support during this endeavor. Their support has served as a continual source of inspiration.

# Introduction

In this study, the census data was used to deploy the power of machine learning to predict if a person makes more than $50,000 a year. Nowadays, modern development of state-of-the-art techniques and big datasets in machine learning has really come to be a powerful predictive analytics tool. We applied data preprocessing, model selection, and evaluation, among other methods that can help develop predictive models robustly capable of discerning income levels, depending on demographic attributes.

The project is managed using Git, fostering transparency and collaboration throughout the development phase. The source code is openly accessible to the public on [GitHub](https://github.com/VanujaSooriyaarachchi) at <https://github.com/VanujaSooriyaarachchi/Census-Data-Income-Prediction.git>. This initiative promotes knowledge sharing and invites contributions from the wider community, thereby. advancing the field of income prediction through machine learning.

# Corpus Preparation

## Dataset Description

The “Census Income” dataset was used in this task and comes from the UCI Machine Learning Repository. It contains some predetermined demographic features with people’s income. There are 12 attributes in the dataset, and the target variable is “Income”. The two main data files that the dataset is spread over are adult.data and adult.test. The data files contain important data to be studied for analysis and model training; thus, enabling research in the fields of income prediction and demographic trends.

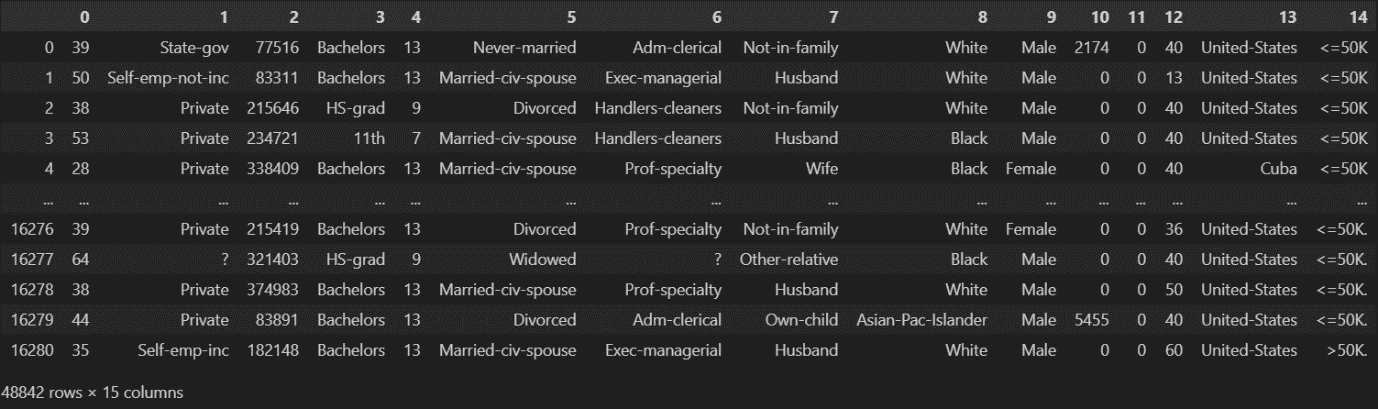
## Data Preprocessing

The data pre-processing pipeline included a few important steps: missing value identification in the data and replacing the corresponding values with NA, followed by dropping such rows that have NA values in them. The categorical variables were one-hot encoded, providing a way to represent categorical data in a numerical format so that ML models can work on it. Similarly, outliers in numeric columns were checked for and then treated with the appropriate method.

All these preprocessing steps are taken on our dataset separately in the domain, ensuring in-depth cleaning and transformation of data on each stage. Cleaned and pre-processed data from all such steps were then concatenated into one big CSV file, which was later used in further analysis and model training. This comprehensive preprocessing approach lays firm ground to build up accurate and reliable machine learning models.

## Data Columns with Unsuitable Values

Since we are unsure of the actual value that needs to be assigned to that row, we are forced to remove the unknown symbols at this point and are unable to replace anything.



**Code**

for col in['workclass', 'occupation', 'native-country']:

    df[col].replace('?', pd.NA, inplace=True)

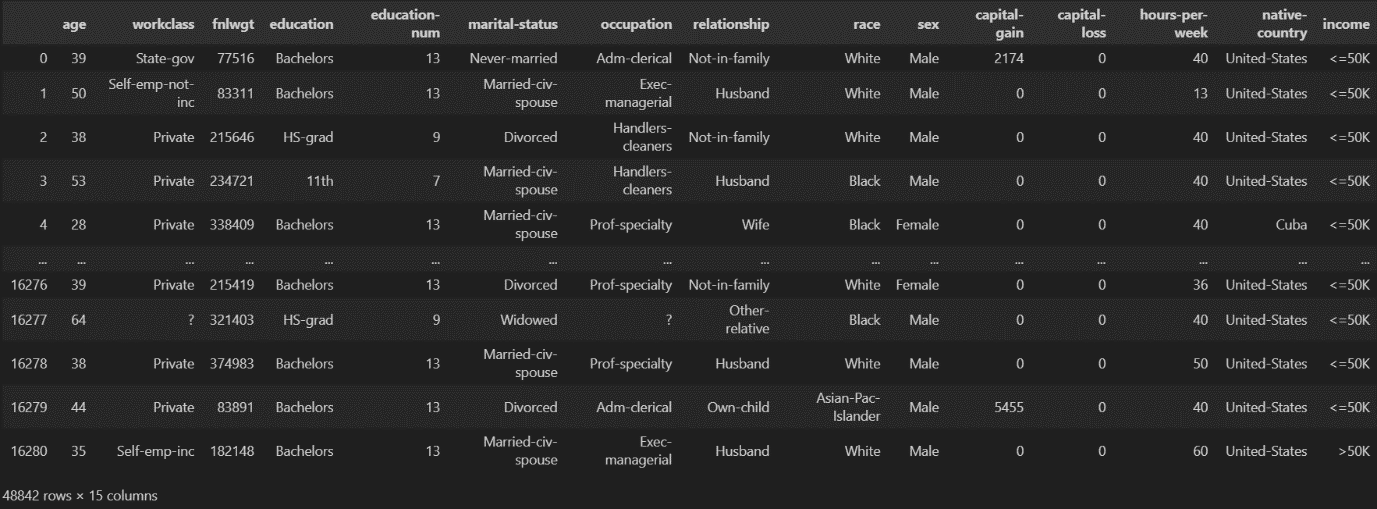
df = df.dropna(subset = ['workclass', 'occupation', 'native-country'])

for col in['workclass', 'occupation', 'native-country']:

    print(f"(col) after remove'?':")

    print(df[col].unique())

## Data Columns with No Values



**Code**

mean = df[df['capital-gain'] != 0]['capital-gain'].mean()

df['capital-gain'] = df['capital-gain'].replace(0, mean)

print(df['capital-gain'])

print(df['capital-gain'].unique())

mean = df[df['capital-loss'] != 0]['capital-loss'].mean()

df['capital-loss'] = df['capital-loss'].replace(0, mean)

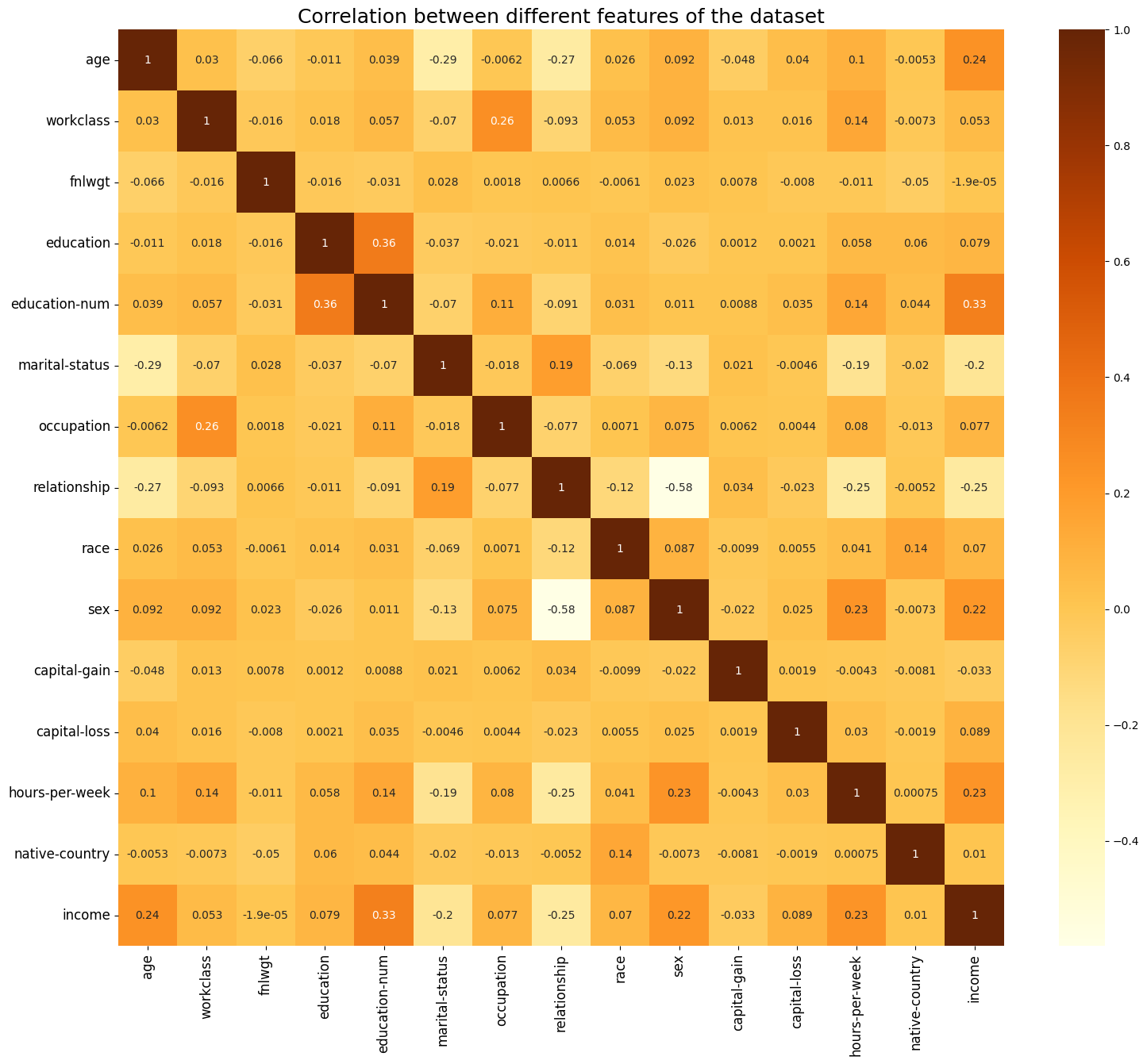
print(df['capital-loss'])

print(df['capital-loss'].unique())

## Outlier Handling

|  |  |
| --- | --- |
| Before | After |
|  |  |
|  |  |
|  |  |
|  |  |

## Correlation Matrix



The correlation matrices are useful tools in feature selection, multicollinearity detection, model building, and data preprocessing for this data analysis course.

* **Feature Selection:** Correlation matrices aid in selecting relevant features for predictive models. This gauging of the relationship will help the predictor come out as those variables showing either weak or strong relationships with the target variable of the model. The approach further sieves the model development procedure by focusing on those features most likely to be important for predictive accuracy (Vogt, 2015).
* **Detection of Multicollinearity:** A correlation matrix proves to be a handy tool in detecting multicollinearity amongst the predictor variables. Multicollinearity is said to happen when two or more predictors are highly correlated with each other, thus leading to an increase in the standard errors, and consequently, the model’s estimate becomes unstable. The analysis of the correlation matrix enables the identification of the pairs of variables with a high correlation value. They help the researcher to take an appropriate remedy either by transforming the variable or dropping it as and when required to overcome the multicollinearity problem so that a good fit model is obtained.
* **Model building:** This follows the correlation matrix and forms the basis for considering one feature at a time for model building. An attempt to identify the significant predictors which have high correlation among themselves and with the response variable, such a predictor is accorded priority during model development. This focused approach is likely to improve the efficiency of the model in the sense that influencing predictors are included, while those that are irrelevant or redundant would be sidestepped, therefore improving efficiency and interpretability of the model.
* **Data Preprocessing:** Correlation matrices find an important place during data preprocessing. They are not used only for feature selection; rather, they reduce multi-collinearity among the features. Prior to building the model, the correlation matrices can be used by the analysts to remove the redundant features or those that have very high correlation with others, hence simplifying the dataset and reducing chances of problems related to multicollinearity. Such a preprocessing step ensures that all subsequent analyses are being done with validity, and model performance gets optimized.

# Solution Methodology

## Model Selection

Two machine learning models were selected for this task:

1. **Random Forest Classifier:** An ensemble learning method that constructs many decision trees for classification, then aggregates their predictions through voting, summing, or averaging to improve the accuracy and overfitting mitigation of the wide classifications.
2. **Nave Bayes Classifier:** Naïve Bayes is an extremely powerful and simple probabilistic classifier, assuming all features are independent of each other and making use of Bayes’ theorem to predict in a very efficient manner. It is very popular and often used in tasks of text classification.

# Model Implementation

The models in this code were developed using the Python programming language with the scikit-learn library. The GaussianNB class implements a simple and effectual algorithm, which is reputed for being applicable in classification assignments. The Random Forest classifier is implemented with the use of the Random Forest Classifier class (Cutler, 2010), which is the strongest ensembled learning model so far, and at the time of creation, this class had been proved as very robust, having an outstanding capability to deal with complex data sets. So, the implementation of these classifiers is done inside the code in order to solve the classification problem.

|  |
| --- |
| **Random Forest Classifier** |
| # Random Forest Classifier  rf\_model = RandomForestClassifier(random\_state=42)  # Perform grid search  grid\_search.fit(X\_train, y\_train)  # Predictions on the training set  best\_rf\_train\_predictions = best\_rf\_model.predict(X\_train)  # Calculate accuracy for the training set  best\_rf\_train\_accuracy = accuracy\_score(y\_train, best\_rf\_train\_predictions)  print("Best Random Forest Training Accuracy:", best\_rf\_train\_accuracy)  # Predictions on the test set  best\_rf\_predictions = best\_rf\_model.predict(X\_test)  # Calculate accuracy for the test set  best\_rf\_test\_accuracy = accuracy\_score(y\_test, best\_rf\_predictions)  print("Best Random Forest Test Accuracy:", best\_rf\_test\_accuracy)  # Generate classification report  report = classification\_report(y\_test, best\_rf\_predictions)  # Print classification report  print ("Classification Report:")  print(report) |
| **Naïve Bayes Classifier** |
| # Create Naïve Bayes model object using Gaussian Naïve Bayes  nb\_model = GaussianNB()  # Train the Naïve Bayes model on the training data  nb\_model.fit(X\_train, y\_train)  # Generate predictions on the training set using the trained Naïve Bayes model  nb\_train\_predictions = nb\_model.predict(X\_train)  # Generate predictions on the test set using the trained Naïve Bayes model  nb\_test\_predictions = nb\_model.predict(X\_test)  # Calculate training accuracy  nb\_train\_accuracy = accuracy\_score(y\_train, nb\_train\_predictions)  print("Naïve Bayes Training Accuracy:", nb\_train\_accuracy)  # Calculate test accuracy  nb\_test\_accuracy = accuracy\_score(y\_test, nb\_test\_predictions)  print("Naïve Bayes Test Accuracy:", nb\_test\_accuracy)  # Generate classification report  report = classification\_report(y\_test, nb\_test\_predictions)  # Print classification report  print("Classification Report:")  print(report) |

# Evaluation Criteria

In evaluating models, the following criteria were considered:

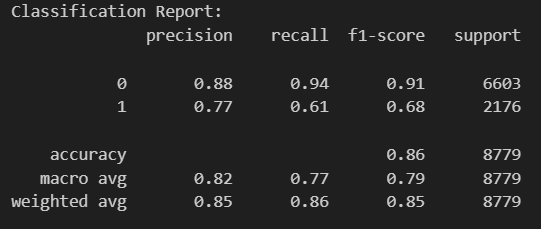
* **Accuracy:** It calculates the percentage of all instances that were accurately predicted.
* **Precision:** The ratio of correctly predicted positive observations to all predicted positive observations is computed using this metric.
* **Recall:** The ratio of accurately predicted positive observations to the total number of actual positive observations is quantified.
* **F1-score:** The harmonic mean of recall and precision, or this score, provides a fair evaluation of both measures.

## Model Evaluation

**Random Forest Classifier**

The Random Forest Classifier achieved the following evaluation metrics:

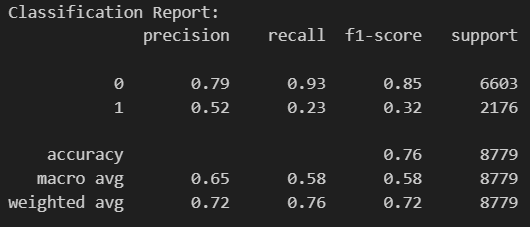
* Accuracy: 0.86
* Precision: 0.88
* Recall: 0.94
* F1-score: 0.91

****

**Naïve Bayes Classifier**

The Naïve Bayes Classifier achieved the following evaluation metrics:

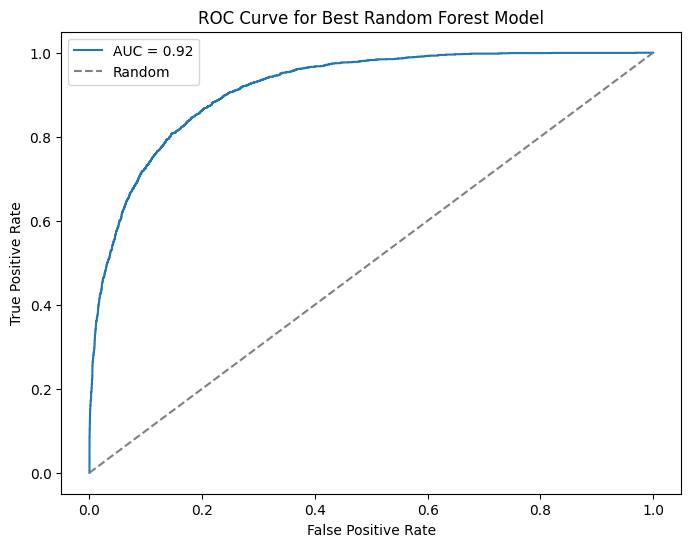
* Accuracy: 0.76
* Precision: 0.79
* Recall: 0.93
* F1-score: 0.85

****

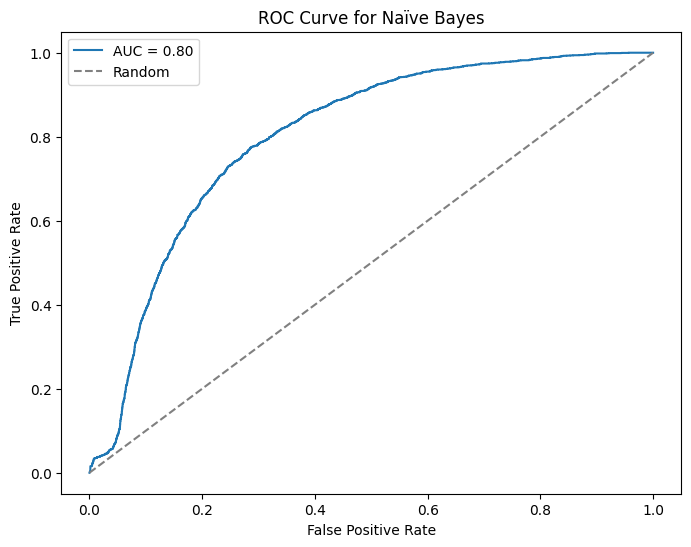
# Experimental Results

**ROC curve**

The Receiver Operating Characteristic (ROC) curve plots sensitivity against (1 - specificity) at various thresholds to assess binary classification models. Better performance is indicated by a higher curve, with the top-left corner being the ideal location. AUC (Area Under the Curve) values closer to 1 indicate superior discrimination. AUC measures overall performance. ROC curves are useful for threshold selection, model comparison, and model evaluation.

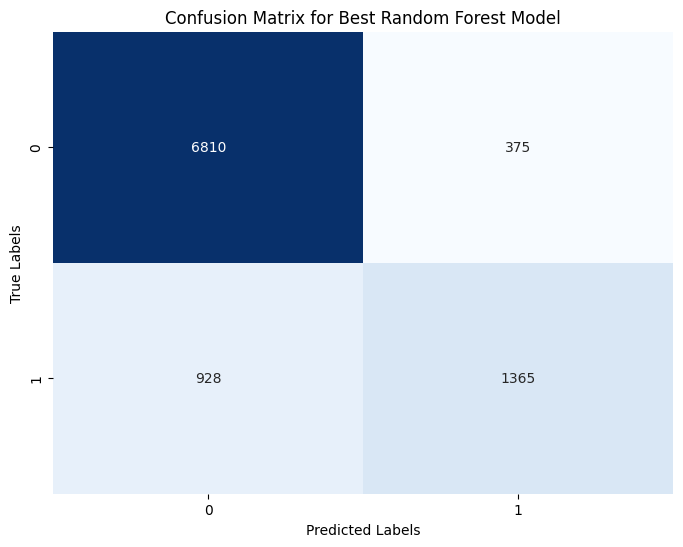
**Random Forest Classifier**

**Naïve Bayes Classifier**

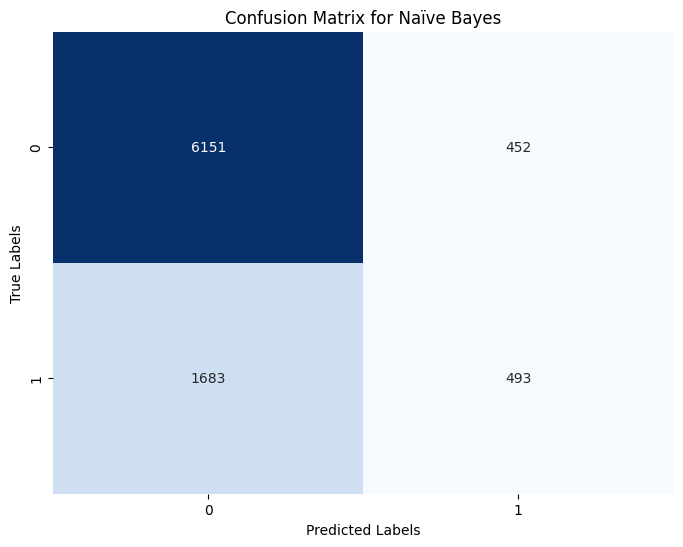
****

**Confusion Matrix**

It is a confusion matrix that accumulates true positives, true negatives, false positives, and false negatives of the predictions, together with a summary of how a classification model performed. It gives information based on recall, accuracy, precision, F1 score of the model, and is important in threshold optimization. Endowed with the power to realize classification errors, one could understand the behavior of the model.

**Random Forest Classifier**

**Naïve Bayes Classifier**



# Limitations and Further Enhancements

**Limitations**

* **Limited Dataset Size:** Small datasets may not fully represent the population, introducing bias.
* **Imbalanced Class Distribution:** Unequal class proportions in the ‘Income’ variable can hinder model performance.
* **Simplified Feature Engineering:** Using more advanced techniques could enhance model accuracy by capturing complex relationships.

**Further improvements**

* **Larger, Diverse Datasets:** Augment data by diversifying sources or increasing sample size for better generalization.
* **Advanced Feature Engineering:** Get to know it through approaches like polynomial features or feature selection to refine input variables.
* **Experiment with state-of-the-art algorithms:** Try using contemporary models, such as Support Vector Machines or Gradient Boosting Machines, on varied architectures where higher performance might be achieved.

# Definitions

* **Machine Learning:** A branch of artificial intelligence that allows computers to deduct general patterns from data and make predictions for themselves when deciding or predicting something, irrespective of the fact that explicit programming guides are to be followed.
* **Naïve Bayes Classifier:** A probabilistic classifier following Bayes’ theorem with the conditional independence assumption between features and the probability estimation through observed data.
* **Random forest classifier:** It is an ensemble method that grows a number of decision trees and then averages the set of output probabilities from individual trees or takes the most voted prediction to make the final prediction.
* **Accuracy:** The proportion of correct values of the instance to all values of the instance which were predicted by the model.
* **Precision:** This metric focuses on the testing ability of a model not to predict negatives as positives. In quantified form, this is the ratio of positive instances correctly predicted over the sum of the positives that have been predicted.
* **Recall:** Is a measure that relates to the model’s ability in capturing all positive instances, calculated by determining the number of correctly predicted positive instances divided by the total actual number of positive instances.
* **F1-score:** Is the harmonic mean of precision and recall, giving a fair outcome of the model’s output, especially with unbalanced datasets.

# Appendix: Source Code

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import os

from sklearn.preprocessing import LabelEncoder

import warnings

warnings.filterwarnings("ignore")

# Dataset Analysis

adult\_data = pd.read\_csv(r"D:\IIT\2 nd Year\2nd Sem\Machine Learning\Course Work\MachineLearningCW\Machine-Learning-Course-Work\CSV file\adult.data", header=None)

with open(r"D:\IIT\2 nd Year\2nd Sem\Machine Learning\Course Work\MachineLearningCW\Machine-Learning-Course-Work\CSV file\adult.test", "r") as file:

    lines = file.readlines()

lines = lines[1:]

with open(r"D:\IIT\2 nd Year\2nd Sem\Machine Learning\Course Work\MachineLearningCW\Machine-Learning-Course-Work\CSV file\adult.test", "w") as file:

    file.writelines(lines)

adult\_test = pd.read\_csv(r"D:\IIT\2 nd Year\2nd Sem\Machine Learning\Course Work\MachineLearningCW\Machine-Learning-Course-Work\CSV file\adult2.test", header=None)

df = pd.concat([adult\_data,adult\_test])

df

column\_names = ['age', 'workclass', 'fnlwgt', 'education', 'education-num', 'marital-status', 'occupation', 'relationship', 'race', 'sex', 'capital-gain', 'capital-loss', 'hours-per-week', 'native-country', 'income']

df.columns = column\_names

df

df.shape

df.describe()

df.isna().sum()

categorical\_columns = []

for col in df.columns:

    if df[col].dtype == "O":

        categorical\_columns.append(col)

categorical\_columns

categorical\_columns.append('education-num')

df[categorical\_columns]

for col in categorical\_columns:

    print(df[col].value\_counts())

    print()

num\_cols = set(df.columns).difference(categorical\_columns)

num\_cols = list(num\_cols)

num\_cols

plt.figure(figsize=(10, 30))

fig, axes = plt.subplots(nrows=3, ncols=2, figsize=(10, 15))

axes = axes.flatten()

for i, col in enumerate(num\_cols):

    sns.histplot(df[col], kde=True, ax=axes[i])

plt.show()

# Data Cleaning

df.nunique()

for column in df.columns:

    unique\_values = df[column].unique()

    print(f"{column} unique values:")

    print(unique\_values)

    print("\n")

df.isin([' ?']).sum(axis=0)

for col in['workclass', 'occupation', 'native-country']:

    df[col].replace('?', pd.NA, inplace=True)

df = df.dropna(subset = ['workclass', 'occupation', 'native-country'])

for col in['workclass', 'occupation', 'native-country']:

    print(f"(col) after remove'?':")

    print(df[col].unique())

df['income'].replace({' <=50K.': ' <=50K', ' >50K.': ' >50K'}, inplace=True)

for column in df.columns:

    unique\_values = df[column].unique()

    print(f"{column} unique values:")

    print(unique\_values)

    print("\n")

adult\_data.isna().sum()

df

mean = df[df['capital-gain'] != 0]['capital-gain'].mean()

df['capital-gain'] = df['capital-gain'].replace(0, mean)

print(df['capital-gain'])

print(df['capital-gain'].unique())

mean = df[df['capital-loss'] != 0]['capital-loss'].mean()

df['capital-loss'] = df['capital-loss'].replace(0, mean)

print(df['capital-loss'])

print(df['capital-loss'].unique())

df.head(10)

df.drop\_duplicates()

plt.figure(figsize=(8, 5))

sns.boxplot(x=df["age"])

plt.title("Age Box Plot")

plt.show()

mean\_age = df.loc[df["age"] < 75, "age"].mean()

df.loc[df["age"] > 75, "age"] = mean\_age

plt.figure(figsize=(8,5))

sns.boxplot(x=df["age"])

plt.title("Age box plot after outlier handling")

plt.show()

plt.figure(figsize=(8,5))

sns.boxplot(x=df["fnlwgt"])

plt.title("fnlwgt box plot")

plt.show()

# Handling the outlier in Final-Weight

Q1 = df['fnlwgt'].quantile(0.25)

Q3 = df['fnlwgt'].quantile(0.75)

IQR = Q3 - Q1

# Define bounds

lower\_bound = Q1 - 1.5 \* IQR

upper\_bound = Q3 + 1.5 \* IQR

# Filter out the outliers

df = df[(df["fnlwgt"] >= lower\_bound) & (df["fnlwgt"] <= upper\_bound)]

plt.figure(figsize=(8, 5))

sns.boxplot(x=df["fnlwgt"])

plt.title("fnlwgt box plot after outlier handling")

plt.show()

plt.figure(figsize=(8, 5))

sns.boxplot(x=df["capital-gain"])

plt.title("capital-gain box plot")

plt.show()

outlier\_threshold = 60000

mean\_threshold = df.loc[df["capital-gain"] <= outlier\_threshold, "capital-gain"].mean()

df.loc[df["capital-gain"] > outlier\_threshold, "capital-gain"] = mean\_threshold

plt.figure(figsize=(8,5))

sns.boxplot(x=df["capital-gain"])

plt.title("capital-gain box plot after outlier handling")

plt.show()

df

plt.figure(figsize=(8, 5))

sns.boxplot(x=df["capital-loss"])

plt.title("capital-loss box plot")

plt.show()

# Define bounds

# Define the thresholds

lower\_threshold = 800

upper\_threshold = 3200

# Filter out the outliers

# Calculate the mean of values within the thresholds

mean\_within\_threshold = df.loc[(df["capital-loss"] >= lower\_threshold) &

                                      (df["capital-loss"] <= upper\_threshold),

                                      "capital-loss"].mean()

# Replace values below the lower threshold and above the upper threshold with the calculated mean

df.loc[df["capital-loss"] < lower\_threshold, "capital-loss"] = mean\_within\_threshold

df.loc[df["capital-loss"] > upper\_threshold, "capital-loss"] = mean\_within\_threshold

# Create a box plot for the "capital-loss" column after handling outliers

plt.figure(figsize=(8, 5))

sns.boxplot(x=df["capital-loss"])

plt.title("capital-loss Box Plot After Outlier Handling")

plt.show()

df

df = df[(df["capital-loss"] < upper\_bound) & (df["capital-loss"] > lower\_bound)]

df

le=LabelEncoder()

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

for col in encoding\_columns:

    df[col] = le.fit\_transform(df[col])

df

# Define the directory to save the CSV file

directory = "Dataset"

# Create the directory

if not os.path.exists(directory):

    os.makedirs(directory)

# Define the path to save the CSV file

csv\_path = os.path.join(directory, "Preprocessed.csv")

# Save the encoded DataFrame to a CSV file

df.to\_csv(csv\_path, index=False)

print(f"CSV file saved: {csv\_path}")

plt.figure(figsize=(18, 15))

plt.title("Correlation between different features of the dataset", fontsize=18)

sns.heatmap(df.corr(), cmap="YlOrBr", annot=True)

plt.xticks(fontsize=12, rotation=90)

plt.yticks(fontsize=12)

plt.show()

import pandas as pd

from sklearn.naive\_bayes import GaussianNB

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import train\_test\_split, GridSearchCV

from sklearn.metrics import accuracy\_score

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

import seaborn as sns

from sklearn.metrics import classification\_report

from sklearn.metrics import confusion\_matrix

import matplotlib.pyplot as plt

from sklearn.metrics import roc\_curve, roc\_auc\_score

df = pd.read\_csv("Dataset/Preprocessed.csv")

print(df)

df

x = df.drop('income', axis=1)

y = df['income']

x

y

X\_train, X\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.2, random\_state=42)

len(X\_test), len(X\_train), len(y\_test), len(y\_train)

# Random Forest

# Define the hyperparameter grid for GridSearchCV

param\_grid = {

    'n\_estimators': [50, 75, 100],

    'max\_depth': [5, 10, 15],

    'min\_samples\_split': [2, 3, 5],

    'min\_samples\_leaf': [1, 2, 3],

    'max\_features': ['sqrt', 'log2']

}

# Random Forest Classifier

rf\_model = RandomForestClassifier(random\_state=42)

# Instantiate GridSearchCV

grid\_search = GridSearchCV(estimator=rf\_model, param\_grid=param\_grid, scoring='accuracy', cv=5, n\_jobs=-1)

# Perform grid search on the training data

grid\_search.fit(X\_train, y\_train)

# Print the best hyperparameters

print("Best Hyperparameters:", grid\_search.best\_params\_)

# Model from grid search

best\_rf\_model = grid\_search.best\_estimator\_

# Predictions on the training set

best\_rf\_train\_predictions = best\_rf\_model.predict(X\_train)

# Calculate accuracy for the training set

best\_rf\_train\_accuracy = accuracy\_score(y\_train, best\_rf\_train\_predictions)

print("Best Random Forest Training Accuracy:", best\_rf\_train\_accuracy)

# Predictions on the test set

best\_rf\_predictions = best\_rf\_model.predict(X\_test)

# Calculate accuracy for the test set

best\_rf\_test\_accuracy = accuracy\_score(y\_test, best\_rf\_predictions)

print("Best Random Forest Test Accuracy:", best\_rf\_test\_accuracy)

# Get predicted probabilities

rf\_probabilities = best\_rf\_model.predict\_proba(X\_test)[:, 1]

# Calculate the ROC curve

fpr, tpr, thresholds = roc\_curve(y\_test, rf\_probabilities)

# Plot the ROC curve

plt.figure(figsize=(8, 6))

plt.plot(fpr, tpr, label=f'AUC = {roc\_auc\_score(y\_test, rf\_probabilities):.2f}')

plt.plot([0, 1], [0, 1], linestyle='--', color='gray', label='Random')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Curve for Best Random Forest Model')

plt.legend()

plt.show()

# Get feature importances from the trained Random Forest model

feature\_importances = best\_rf\_model.feature\_importances\_

# Create a DataFrame to display feature importances

feature\_importance\_df = pd.DataFrame({'Feature': x.columns, 'Importance': feature\_importances})

feature\_importance\_df = feature\_importance\_df.sort\_values(by='Importance', ascending=False)

# Plot the top N most important features

top\_features = 10

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

sns.barplot(x='Importance', y='Feature', data=feature\_importance\_df.head(top\_features))

plt.title(f'Top {top\_features} Feature Importances for Best Random Forest Model')

plt.show()

# Get the confusion matrix

cm = confusion\_matrix(y\_test, best\_rf\_predictions)

# Display the confusion matrix using a heatmap

plt.figure(figsize=(8, 6))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)

plt.xlabel('Predicted Labels')

plt.ylabel('True Labels')

plt.title('Confusion Matrix for Best Random Forest Model')

plt.show()

# Generate classification report

report = classification\_report(y\_test, best\_rf\_predictions)

# Print classification report

print("Classification Report:")

print(report)

# Naïve Bayes

nb\_model = GaussianNB()

nb\_model.fit(X\_train, y\_train)

nb\_train\_predictions = nb\_model.predict(X\_train)

nb\_test\_predictions = nb\_model.predict(X\_test)

nb\_train\_accuracy = accuracy\_score(y\_train, nb\_train\_predictions)

print("Naïve Bayes Training Accuracy:", nb\_train\_accuracy)

nb\_test\_accuracy = accuracy\_score(y\_test, nb\_test\_predictions)

print("Naïve Bayes Test Accuracy:", nb\_test\_accuracy)

# Get predicted probabilities

nb\_probabilities = nb\_model.predict\_proba(X\_test)[:, 1]

# Calculate the ROC curve

fpr, tpr, thresholds = roc\_curve(y\_test, nb\_probabilities)

# Plot the ROC curve

plt.figure(figsize=(8, 6))

plt.plot(fpr, tpr, label=f'AUC = {roc\_auc\_score(y\_test, nb\_probabilities):.2f}')

plt.plot([0, 1], [0, 1], linestyle='--', color='gray', label='Random')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Curve for Naïve Bayes')

plt.legend()

plt.show()

from sklearn.metrics import confusion\_matrix

import seaborn as sns

import matplotlib.pyplot as plt

# Get the confusion matrix

cm\_nb = confusion\_matrix(y\_test, nb\_test\_predictions)

# Display the confusion matrix using a heatmap

plt.figure(figsize=(8, 6))

sns.heatmap(cm\_nb, annot=True, fmt='d', cmap='Blues', cbar=False)

plt.xlabel('Predicted Labels')

plt.ylabel('True Labels')

plt.title('Confusion Matrix for Naïve Bayes')

plt.show()

# Comparing models

# Calculate accuracy for Naïve Bayes

nb\_accuracy = accuracy\_score(y\_test, nb\_test\_predictions)

print("Naïve Bayes Accuracy:", nb\_accuracy)

# Calculate accuracy for Random Forest

rf\_accuracy = accuracy\_score(y\_test, best\_rf\_predictions)

print("Random Forest Accuracy:", rf\_accuracy)

# Naïve Bayes

nb\_classification\_report = classification\_report(y\_test, nb\_test\_predictions)

print("Naïve Bayes Classification Report:\n", nb\_classification\_report)

# Random Forest

rf\_classification\_report = classification\_report(y\_test, best\_rf\_predictions)

print("Random Forest Classification Report:\n", rf\_classification\_report)

# Train the Random Forest model

rf\_model.fit(X\_train, y\_train)

# Get predicted probabilities for Random Forest

rf\_probabilities = rf\_model.predict\_proba(X\_test)[:, 1]

# Calculate ROC curve for Random Forest

fpr\_rf, tpr\_rf, \_ = roc\_curve(y\_test, rf\_probabilities)

# Calculate AUC for Random Forest

auc\_rf = roc\_auc\_score(y\_test, rf\_probabilities)

# Calculate AUC for Naïve Bayes

auc\_nb = roc\_auc\_score(y\_test, nb\_probabilities)

# Plot ROC curve for Random Forest

plt.figure(figsize=(8, 6))

plt.plot(fpr\_nb, tpr\_nb, label=f'Naïve Bayes AUC = {auc\_nb:.2f}')

plt.plot(fpr\_rf, tpr\_rf, label=f'Random Forest AUC = {auc\_rf:.2f}')

plt.plot([0, 1], [0, 1], linestyle='--', color='gray', label='Random')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

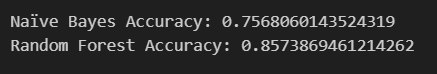
plt.title('ROC Curves for Naïve Bayes and Random Forest')

plt.legend()

plt.show()

# Comparing models

## Accuracy



## Classification Report

A screenshot of a computer screen

Description automatically generated

## ROC Curves for Naïve Bayes and Random Forest

A line graph with blue and orange lines

Description automatically generated

The Naïve Bayes classifier accuracy is 75% and the Random Forest classifier accuracy is 85%. Naïve Bayes, despite its simplicity, may struggle with capturing complex relationships, leading to lower accuracy. In contrast, Random Forest, with its ensemble approach and ability to handle non-linearities, achieves higher accuracy by leveraging multiple decision trees.

# Reference

1. www.youtube.com. (n.d.). *Random Forest Algorithm | Sinhala*. [online] Available at: https://youtu.be/-546xQKbG90?si=YTe00tub3aS2Cuq1 [Accessed 29 Mar. 2024].
2. www.youtube.com. (n.d.). *Naive Bayes Algorithm | Sinhala*. [online] Available at: https://youtu.be/VMvj9x1HtDY?si=ufRW2dLTysZaTEyQ [Accessed 29 Mar. 2024].
3. www.youtube.com. (n.d.). *Outlier Detection using Z-Score | Sinhala*. [online] Available at: https://youtu.be/h4zsu3\_\_uzQ?si=STEIQO8xCfpjPZiX [Accessed 29 Mar. 2024].

‌