BSc. Machine Learning – CM 2604

Level 05

Module	Machine Learning
Module Code	CM 2604
Stage	Year 02 (2^{nd} Semester)
Name	B.G.C. GOMES
IIT ID	20220578
RGU ID	2313082

Table of Contents

Introduction

This coursework involves developing and evaluating classification models to predict individual income levels using census data. It encompasses preprocessing steps such as data cleansing and encoding categorical variables. Naïve Bayes and Random Forest algorithms are trained and assessed for predictive accuracy using key metrics like confusion matrices and classification reports. Visual representations like heatmaps and ROC curves enhance interpretation. Comparative analysis aids in model selection. Through this coursework, participants gain practical machine learning skills applicable to real-world classification tasks.

Data Set

The spam-non spam dataset, which has over 4601 rows and 57 characteristics, was used totrain the model.

Source of the Dataset	**UCI Machine Learning Repository.**
Number of instances	**48,842 instances**
Number of attributes	**14 Attributes**
Missing values	**Replaced '?' with NaN, then dropped rows with missing data.**
Number of classes	**02 (>50K and <=50K)**
Relatable Tasks	**Classification to predict income levels.**

Corpus Preparation

Pre – processing techniquesData Cleaning

Data Cleaning Step	Description
Handling Missing Values	Replaced '?' with NaN to denote missing values
Removal of Duplicate Rows	Removed duplicate rows from the data set
Final Dataset size before cleaning	48,842 instances
Final Dataset size after cleaning	42'010 instances

```
Number of rows before removing duplicate rows: 48848
Number of rows after removing duplicate rows: 48813
```

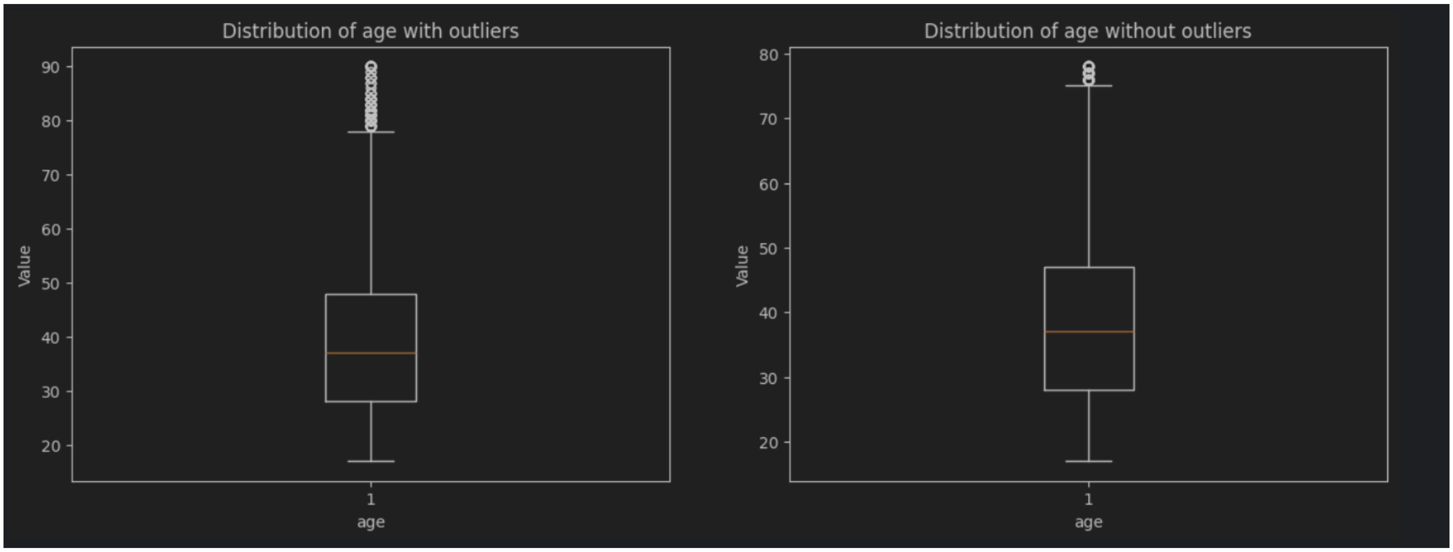
Data transformation

Data transformation involved outlier removal and standardization using StandardScaler. Detected outliers were nullified and removed, resulting in enhanced data quality and suitability for analysis.

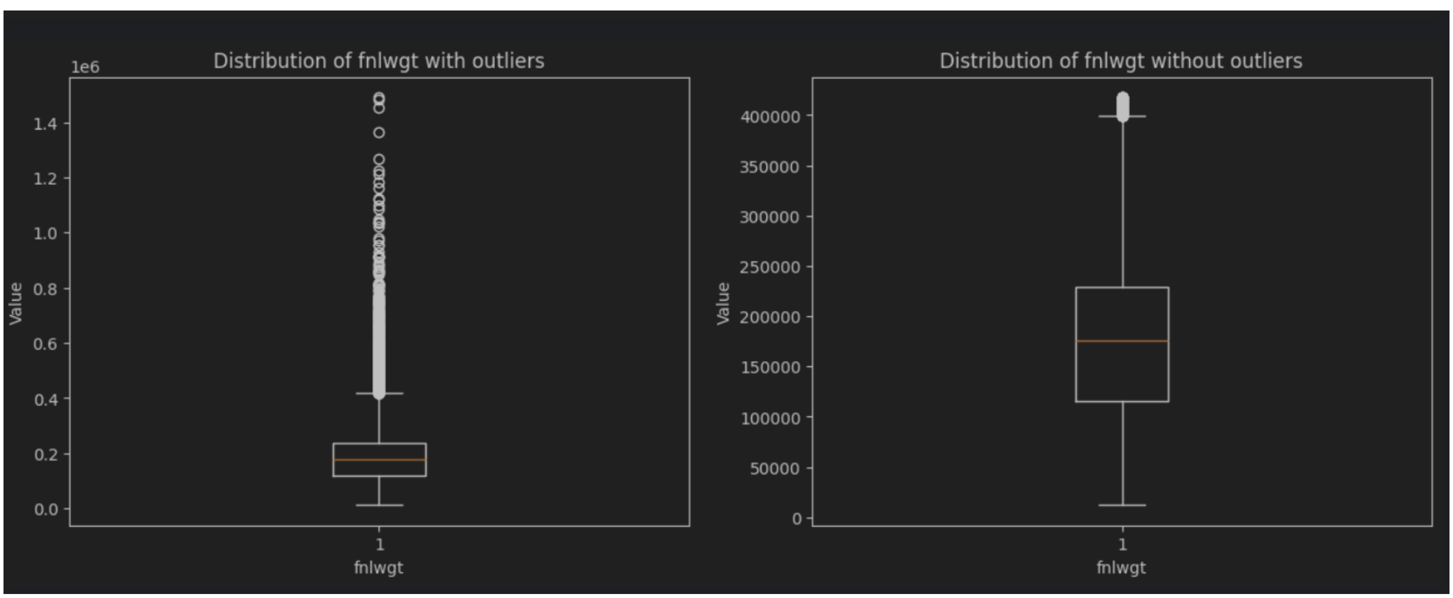
Data Transformation Stem	Description
Handling Outliers	Detected and replaced outliers with NaN values.
Removal of rows with null values	Removed rows containing null values from the dataset
Final dataset size before cleanup	42,010 instances
Final dataset size after cleanup	**34,466 instances**

Outlier Handling

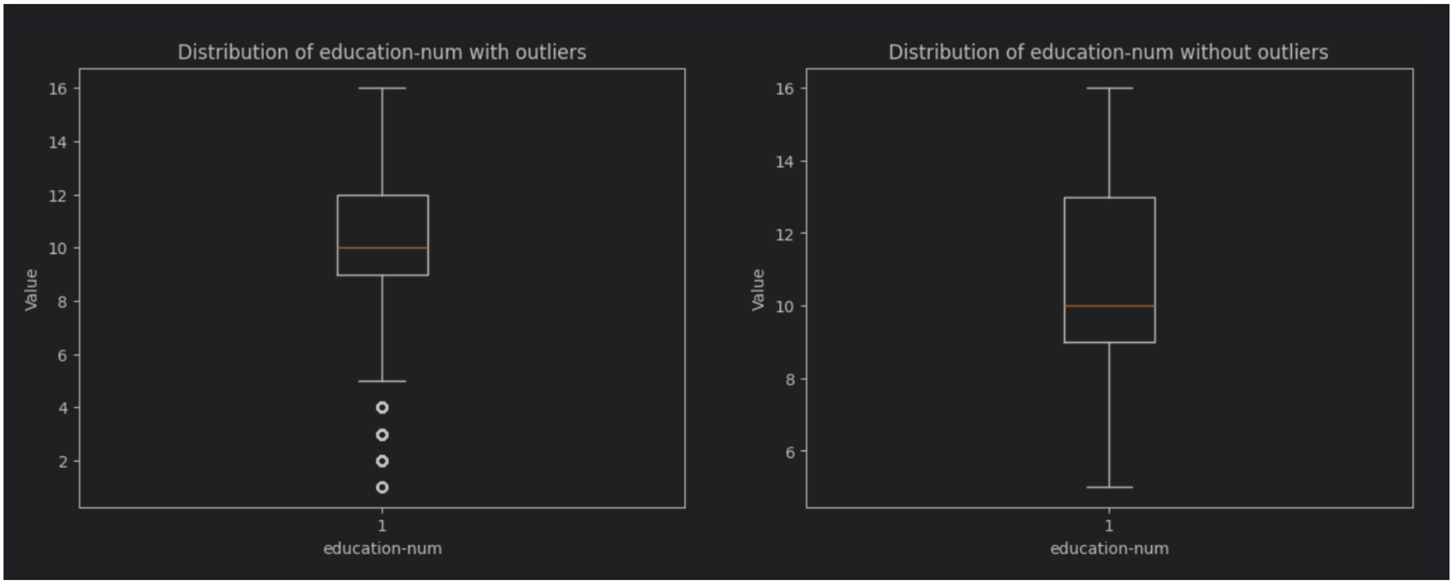
- Boxplot of the “age” column before handling outliers and after handling outliers.



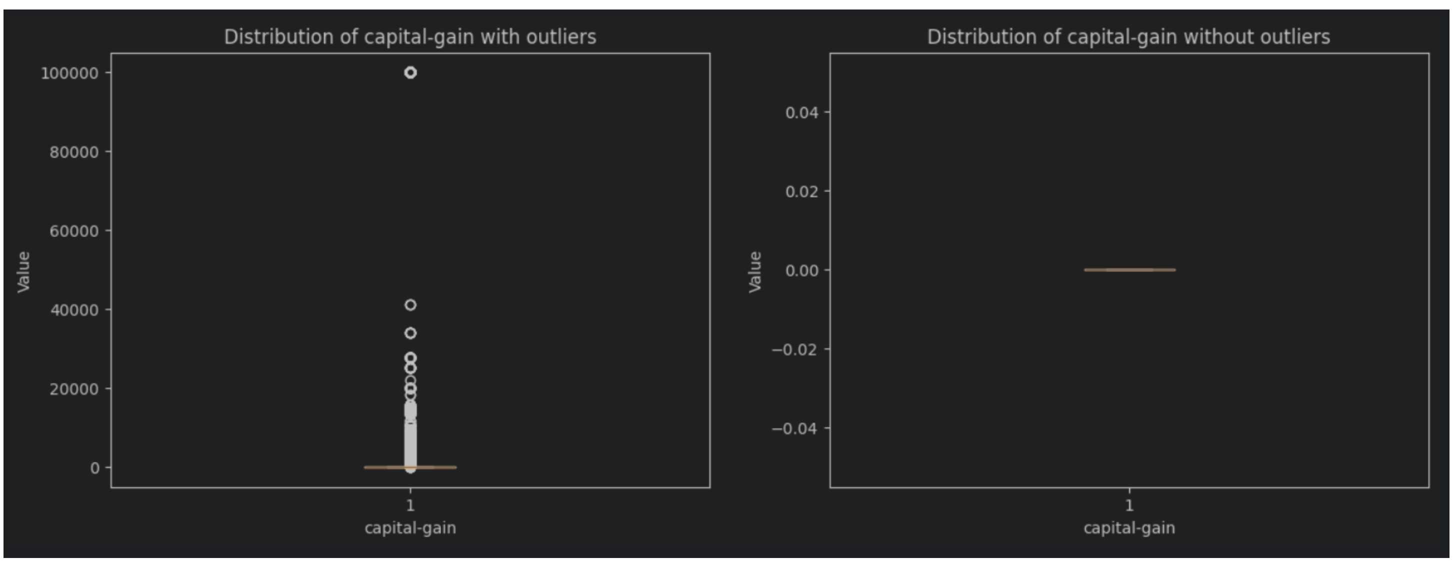
- Boxplot of the “fnlwgt” (Final Weight) column before handling outliers and after handling outliers



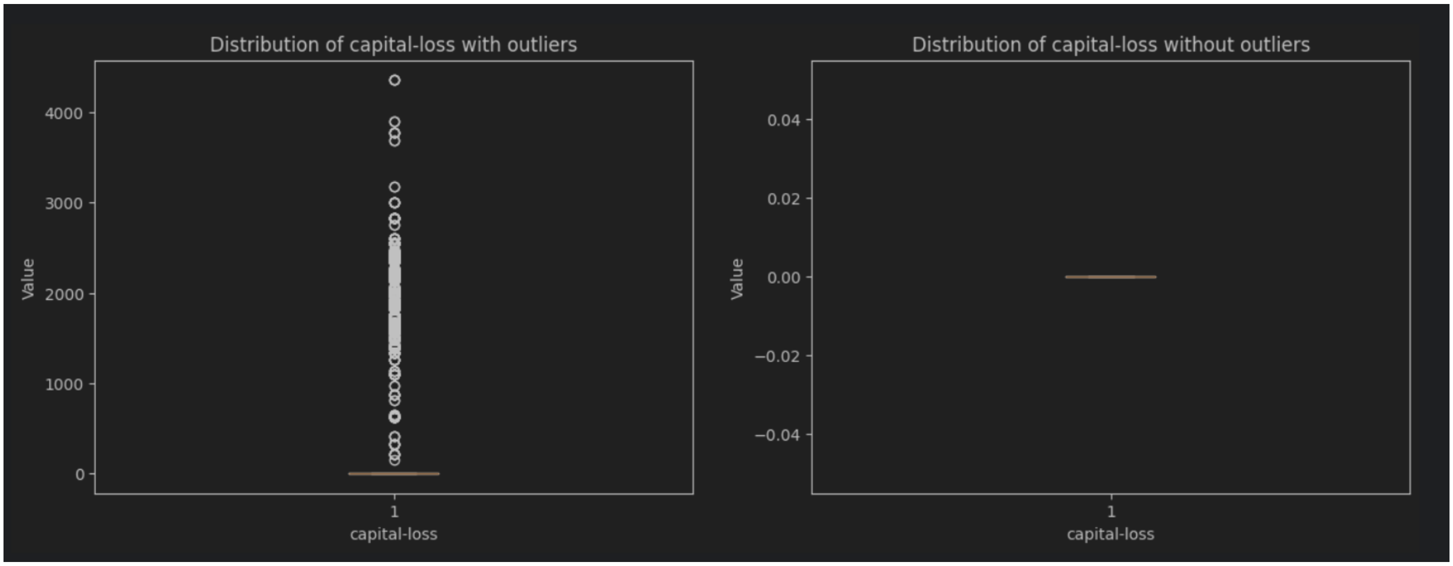
- Boxplot of the “education-num” column before handling outliers and after handling outliers.



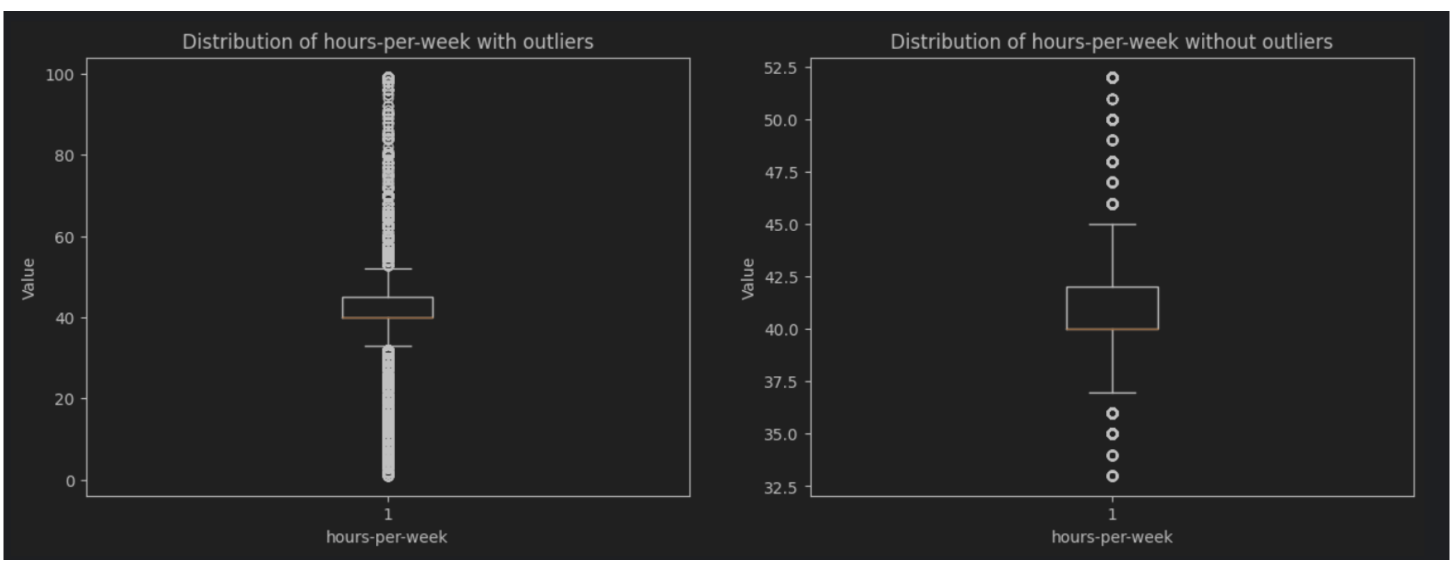
- Boxplot of the "capital-gain” column before handling outliers and after handling outliers.



- Boxplot of the "capital-loss" column before handling outliers and after handling outliers.



- Boxplot of the "hours-per-week" column before handling outliers and after handling outliers.



Handling Class Imbalance

Addressing the class imbalance challenge in analyzing the census income dataset, I utilized an up-sampling technique to ensure equal representation of instances between the two income categories (>50K and <=50K). Specifically, I augmented the '<=50K' subset by replicating instances through random draws with replacement while keeping the random state fixed at 42 for reproducibility. This strategic augmentation boosted the size of the '<=50K' subset to match that of the '>50K' subset, effectively establishing class equilibrium within the dataset. To counter the disparity in class distribution within the census income dataset, I implemented an up-sampling approach aimed at achieving parity between the '>50K' and '<=50K' income categories. Through this technique, I artificially inflated the size of the '<=50K' subset by performing random selections with replacement, maintaining a constant random state of 42 to ensure consistency in results. By elevating the number of instances in the '<=50K' subset to mirror that of the '>50K' subset, I successfully rectified the imbalance, fostering a more balanced foundation for subsequent modeling endeavors.

```
Greater than $50K Subset Summary:
count    11685.000000
mean      4042.931365
std      14757.939193
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max      99999.000000

Column Name: capital-gain
Data Type: int64
```

```
Less than or equal to $50K Subset Summary:
count    37128.000000
mean       147.117216
std        937.085843
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max      41310.000000

Column Name: capital-gain
Data Type: int64
```

```
Balanced Data Summary:
count    23370.000000
mean      2090.375310
std      10637.035352
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max      99999.000000

Column Name: capital-gain
Data Type: int64
```

```
Income with below 50k count: 37128
Income with above 50k count: 11685
```

Data Validation

Dataset doesn't contain any missing values.

```
cleaned_adult_df.isna().sum().any()
cleaned_adult_df.isna().sum()
```

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

Feature Encoding

Feature encoding is vital in machine learning as it transforms categorical data into a numerical format, enabling algorithms to interpret and process them effectively. It ensures compatibility between data types and facilitates model training, enhancing prediction accuracy and performance. Proper encoding preserves essential information while minimizing dimensionality, crucial for robust model development and interpretation.

```
categorical_features = ['workclass', 'education', 'marital-status', 'occupation', 'relationship', 'race', 'sex', 'native-country']
```

```
       age  workclass  fnlwgt  education  education-num  marital-status  \
0       39          5   77516          9             13               4
1       50          4   83311          9             13               2
2       38          2  215646         11              9               0
3       53          2  234721          1              7               2
4       28          2  338409          9             13               2
...    ...        ...     ...        ...            ...             ...
48836   33          2  245211          9             13               4
48837   39          2  215419          9             13               0
48839   38          2  374983          9             13               2
48840   44          2   83891          9             13               0
```

```
       occupation  relationship  race  sex  capital-gain  capital-loss  \
0               0             1     4    1          2174             0
1               3             0     4    1             0             0
2               5             1     4    1             0             0
3               5             0     2    1             0             0
4               9             5     2    0             0             0
...           ...           ...   ...  ...           ...           ...
48836           9             3     4    1             0             0
48837           9             1     4    0             0             0
48839           9             0     4    1             0             0
48840           0             3     1    1          5455             0
48841           3             0     4    1             0             0

       hours-per-week  native-country  income
0                  40              38   <=50K
1                  13              38   <=50K
2                  40              38   <=50K
3                  40              38   <=50K
4                  40               4   <=50K
...               ...             ...     ...
48836              40              38  <=50K.
48837              36              38  <=50K.
48839              50              38  <=50K.
48840              40              38  <=50K.
48841              60              38   >50K.
```

Train/ Test split

```
# Split 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)
```

The training and test datasets are essential components in machine learning model development. The training dataset is used to train the model by fitting it to the data, while the test dataset evaluates the model's performance on unseen data to assess its generalization ability. Splitting data into these sets aids in measuring model accuracy and avoiding overfitting.

splitting the dataset into training and testing sets is vital for evaluating model performance. This separation allows us to train the model on one subset and validate it on another, ensuring unbiased assessment. Utilizing 'scikit-learn's train_test_split' function facilitates this process, enabling random partitioning based on specified ratios, such as 80% for training and 20% for testing. Additionally, setting a random state ensures reproducibility across multiple runs.

Solution Methodology

Model Selection

In this project, we're tasked with employing Naïve Bayes and Random Forest classifiers to predict income levels categorized as either greater than or less than $50K. The objective is to determine the most accurate algorithm for this classification task and identify the optimal approach to achieve our goal.

Naïve base classifier

The Naive Bayes model is based on Bayes' theorem and assumes that features are independent of each other given the class label. It calculates the probability of a class label given the input features using conditional probability. Naive Bayes is suitable for tasks with categorical or continuous features and works well with large datasets, text classification, and spam filtering, among others. Its simplicity, efficiency, and ability to handle high-dimensional data make it suitable for various classification tasks.

Evaluation Criteria

- Naïve Base **Classifier**

```
GaussianNB
GaussianNB()
```

The Naïve Bayes classifier, built on census data, predicts income levels by applying Bayes' Theorem. It simplifies by assuming independence among characteristics. It calculates probabilities for different combinations of features to predict incomes above $50,000. Updating with new data, it selects the most probable outcome. Naïve Bayes excels due to its simplicity, efficiency, and adaptability to various data types. It's particularly suited for income prediction tasks, especially with smaller datasets.

Evaluation criteria (Naïve Base)

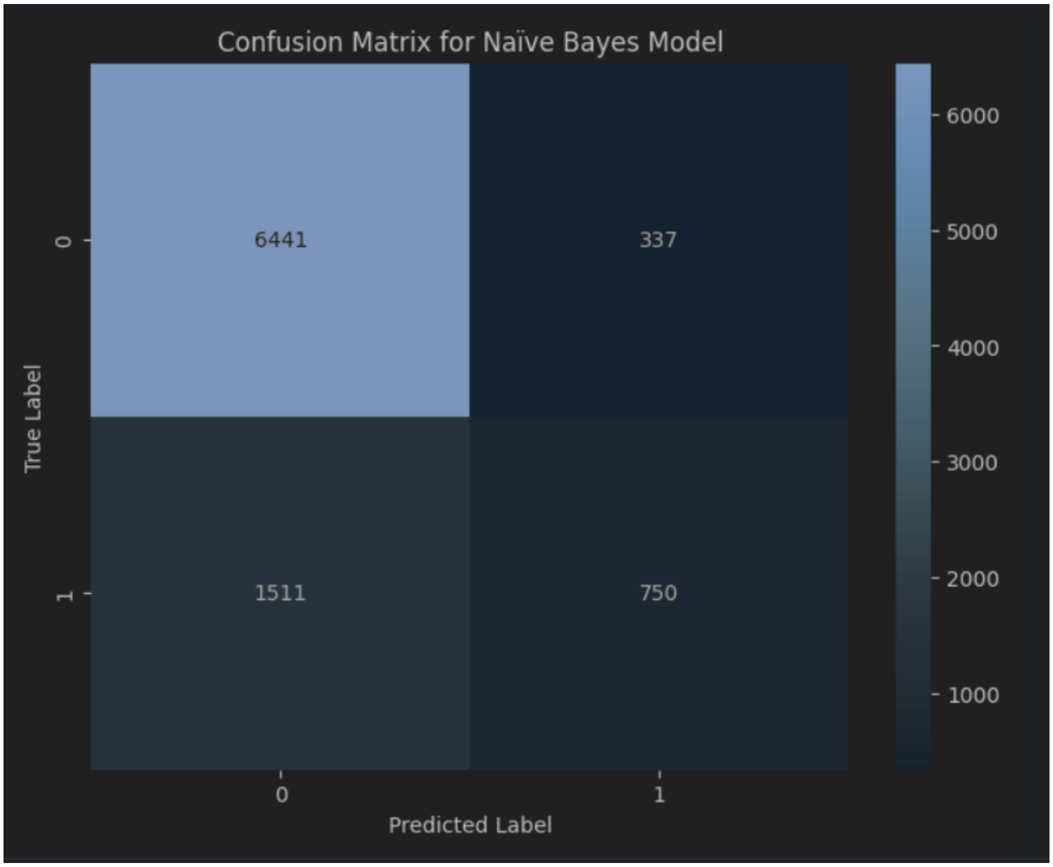
Classification Report

```
Classification Report
Naïve Bayes Model Accuracy: 0.7955526053767009

              precision    recall  f1-score   support

           0       0.81      0.95      0.87      6778
           1       0.69      0.33      0.45      2261

    accuracy                           0.80      9039
   macro avg       0.75      0.64      0.66      9039
weighted avg       0.78      0.80      0.77      9039
```

Testing Dataset Accuracy → 0.7955526053767009

Training Dataset Accuracy → 0.7972894482090997

Confusion Matrix



True Negative: 6441 - The model accurately predicted 6441 individuals whose income does not exceed SSOK/year

False Positive: 337 - The model predicted that 337 individuals had an income exceeding SSOK/year when, in fact, they did got.

False Negative: 1511 -1511 individuals actually had an income exceeding S50K/year, but the model failed to identify them as such.

True Positive: 750- The model correctly identified 750 individuals as having an income that exceeds SS0K/year.

ROC Curve _ Naïve Base

5.2.3 ROC Curve

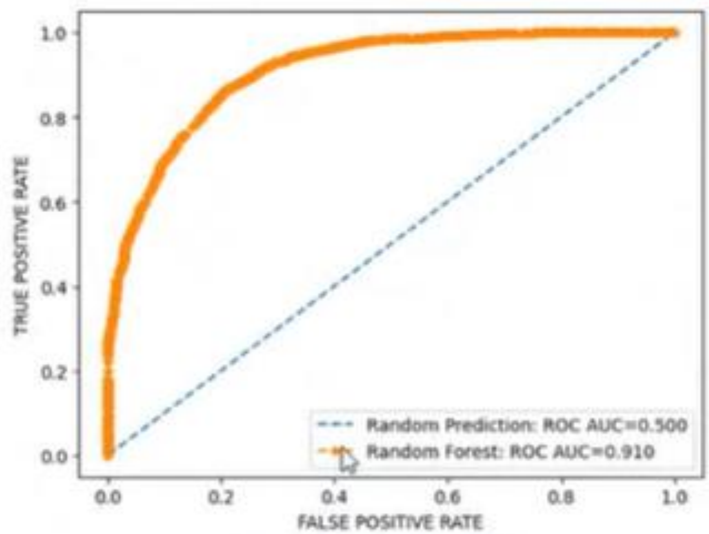


Figure 27 : Random Forest ROC Curve

```
Receiver Operating Characteristic (ROC) curve value for Naïve Bayes Model : 0.8583660172770634
```

Random Forest Classifier

```
▾ RandomForestClassifier
RandomForestClassifier()
```

The Random Forest Classifier, utilizing decision trees and census data, excels as an ensemble learning method for predicting income levels. By accommodating both numerical and categorical data seamlessly, it captures complex interactions effectively. Its inherent feature importance assessment and reduced overfitting make it invaluable for income prediction tasks, while its scalability ensures efficiency even with large datasets.

Evaluation criteria (Random Forest)

Testing Accuracy → 0.8643655271600841

Classification Report (RF - Testing Data)

```
Classification Report for Random Forest Model (Testing Data):
              precision    recall  f1-score   support

           0       0.88      0.92      0.90      6778
           1       0.73      0.63      0.68      2261

    accuracy                           0.85      9039
   macro avg       0.81      0.78      0.79      9039
weighted avg       0.85      0.85      0.85      9039
```

Confusion Matrix (RF – Testing Data)

True Positive(TP) – 6263 - Actually Less than 50K and model predicts as Less than 50k

True Negative(TN) – 1428 - Actually, Greater than 50K and mode predicts as greater than 50k

False Positive(FP) – 515 - Actually Less than 50K but model predicts as Greater than 50k

False Negative(FN) – 833 - Actually Greater than 50k but model predicts as less than 50k

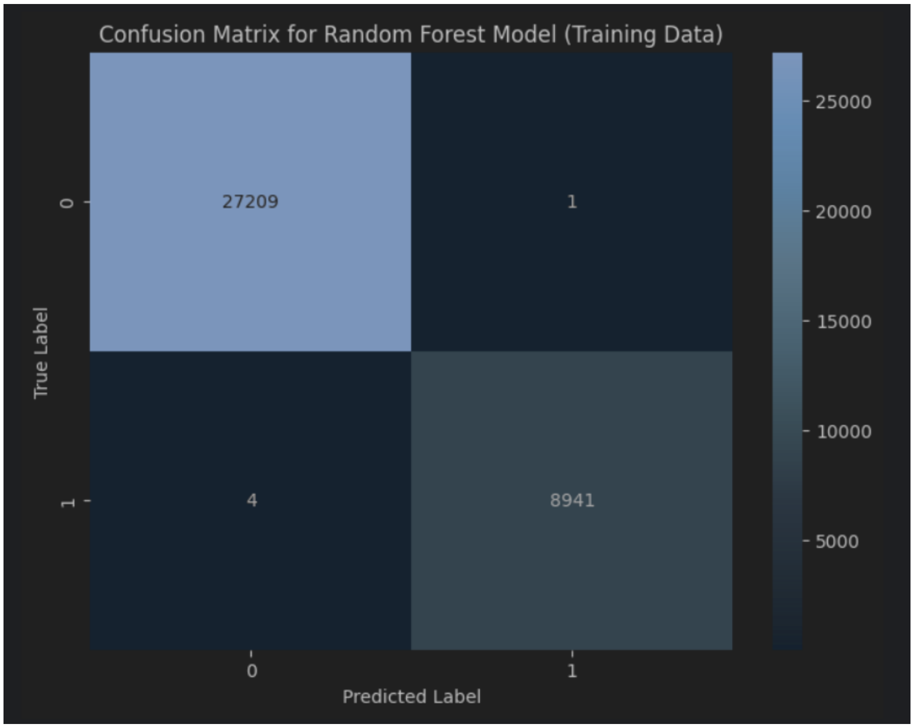
Training Accuracy → 0.9113262342691191

Classification Report (RF – Training Data)

Classification Report for Random Forest Model (Training Data):

	precision	recall	f1-score	support
0	1.00	1.00	1.00	27210
1	1.00	1.00	1.00	8945
accuracy			1.00	36155
macro avg	1.00	1.00	1.00	36155
weighted avg	1.00	1.00	1.00	36155

Confusion Matrix (RF – Training Data)



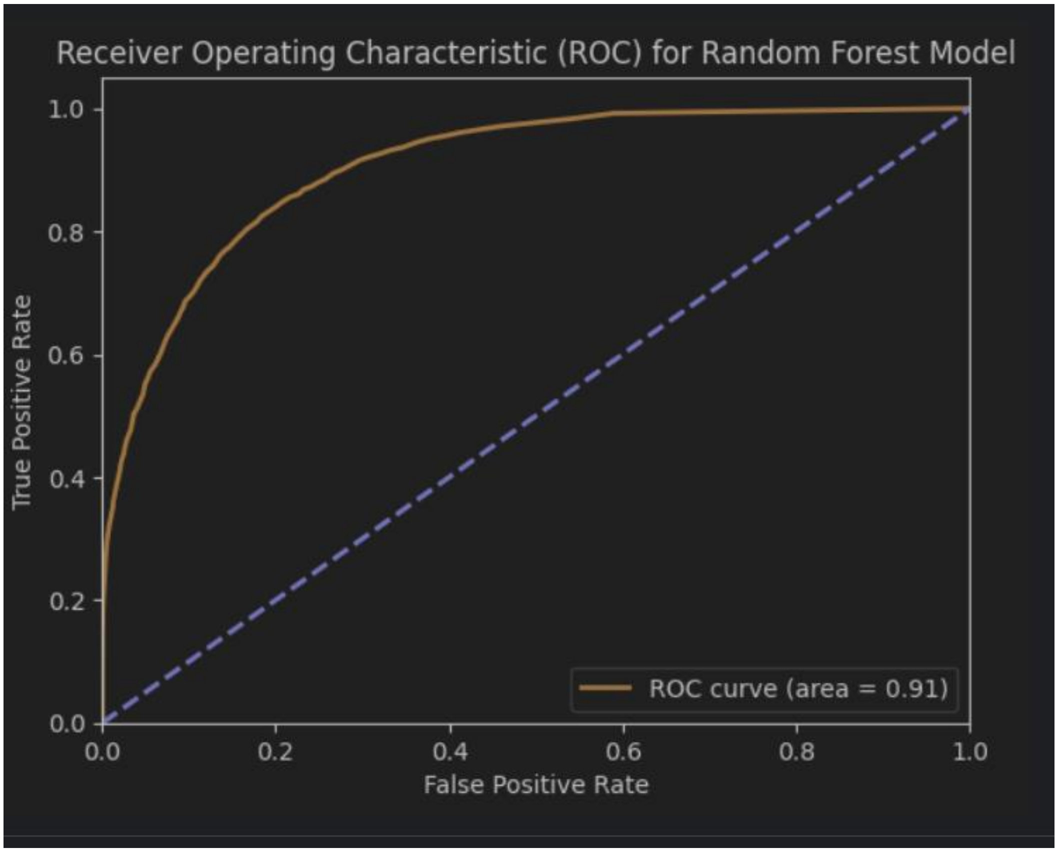
True Positive(TP) – 27209 - Actually Less than 50K and model predicts as Less than 50k

True Negative(TN) – 04 - Actually, Greater than 50K and mode predicts as greater than 50k

False Positive(FP) – 1 - Actually Less than 50K but model predicts as Greater than 50k

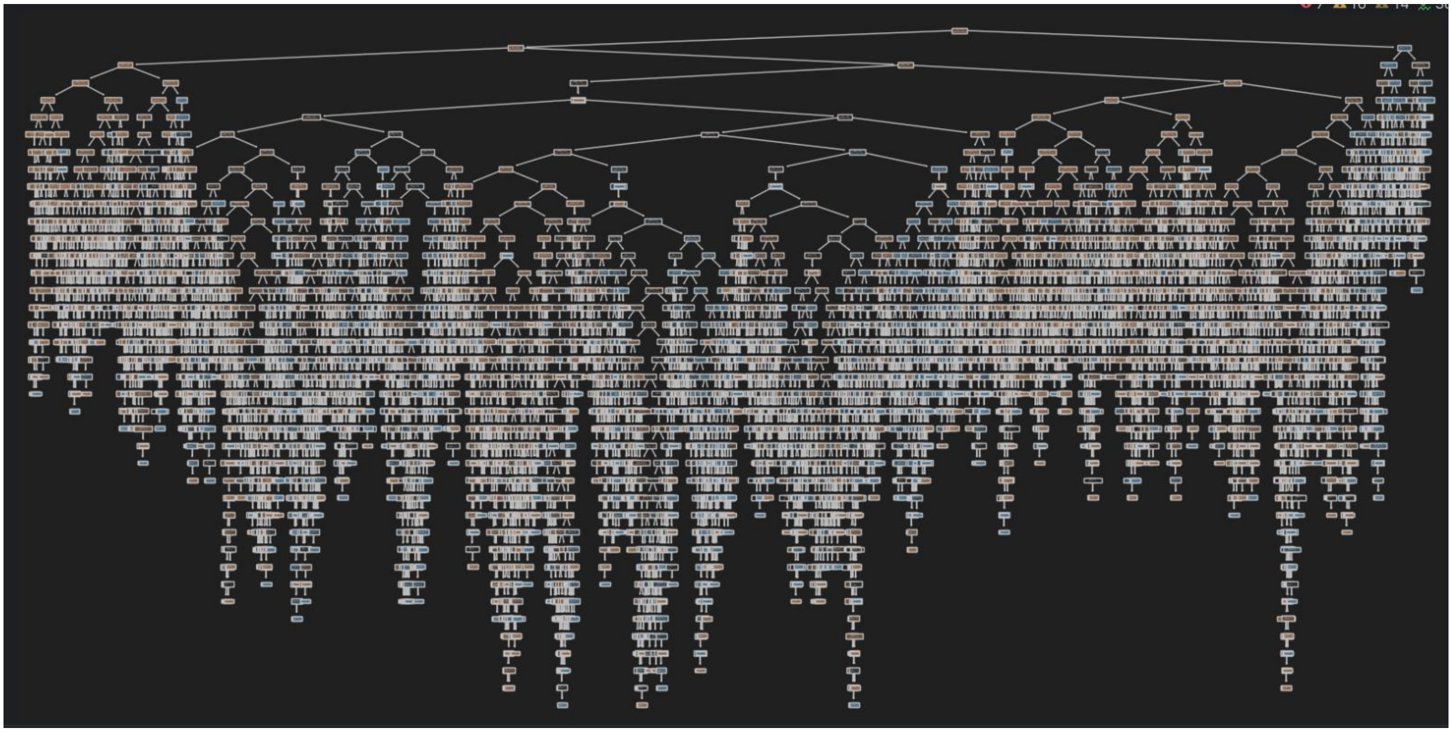
False Negative(FN) – 8941 - 2303 Actually Greater than 50k but model predicts as less than 50k

ROC curve _ Random Forest



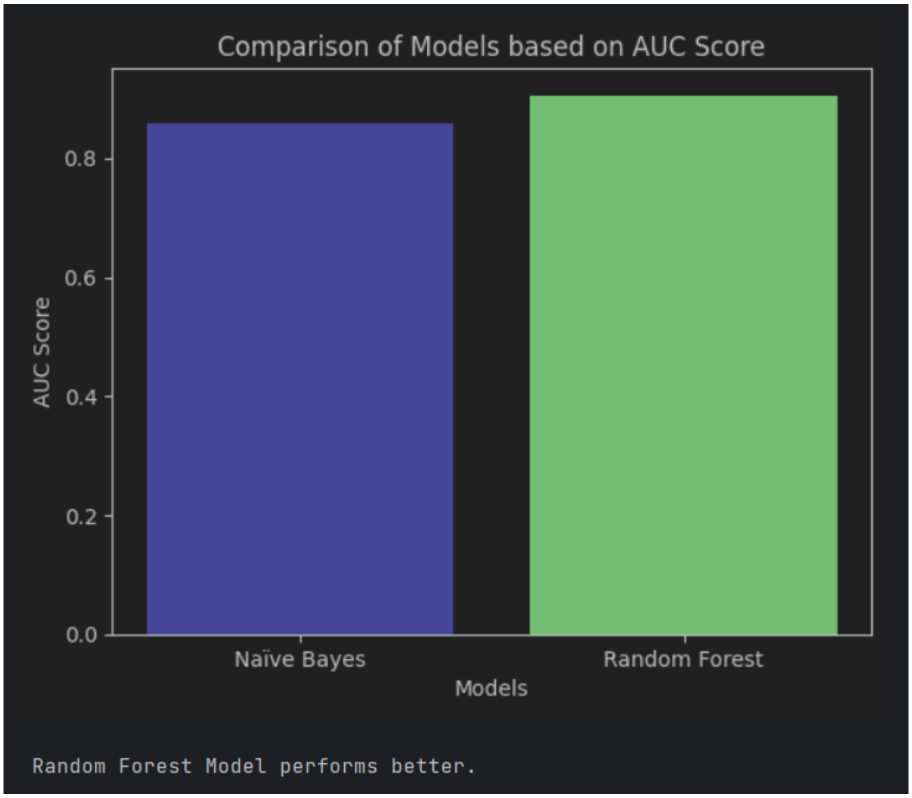

```
Receiver Operating Characteristic (ROC) curve value for Random Forest Model : 0.905651711073459
```

Visualization – Random Forest



Experimental Results

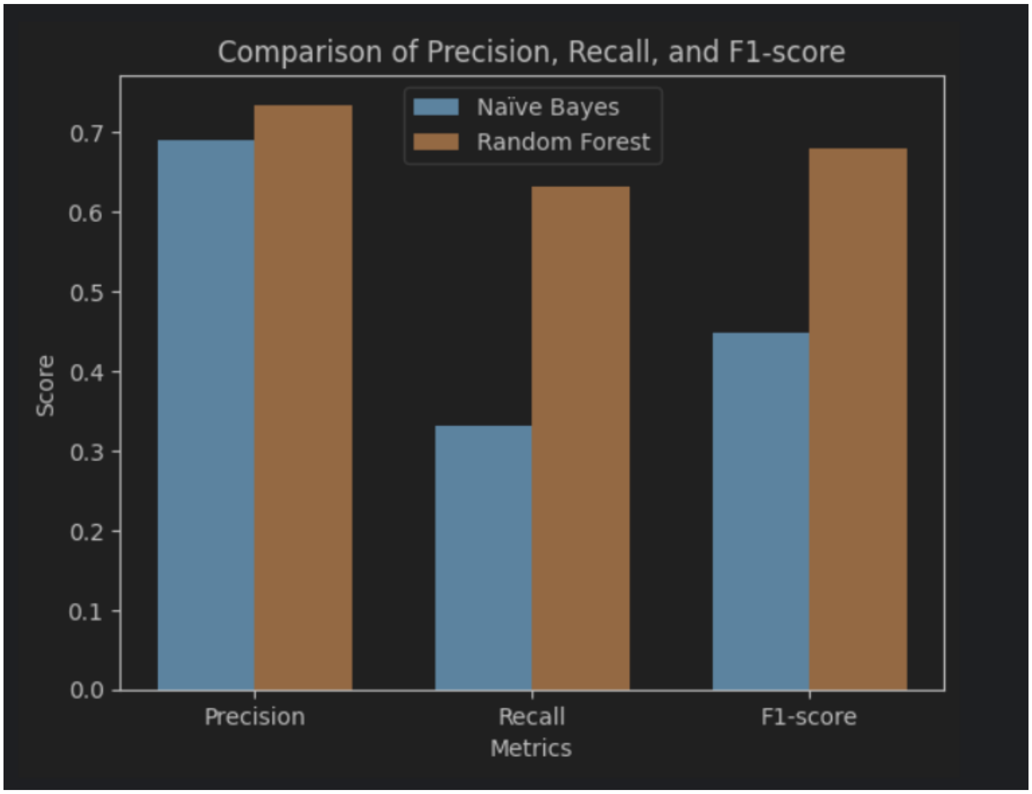
Accuracy



Based on the comparison of models, Random Forest demonstrated superior performance with higher accuracy compared to Naïve Bayes. This suggests that Random Forest is better suited for the given task, as it achieves higher accuracy in predicting the target variable.

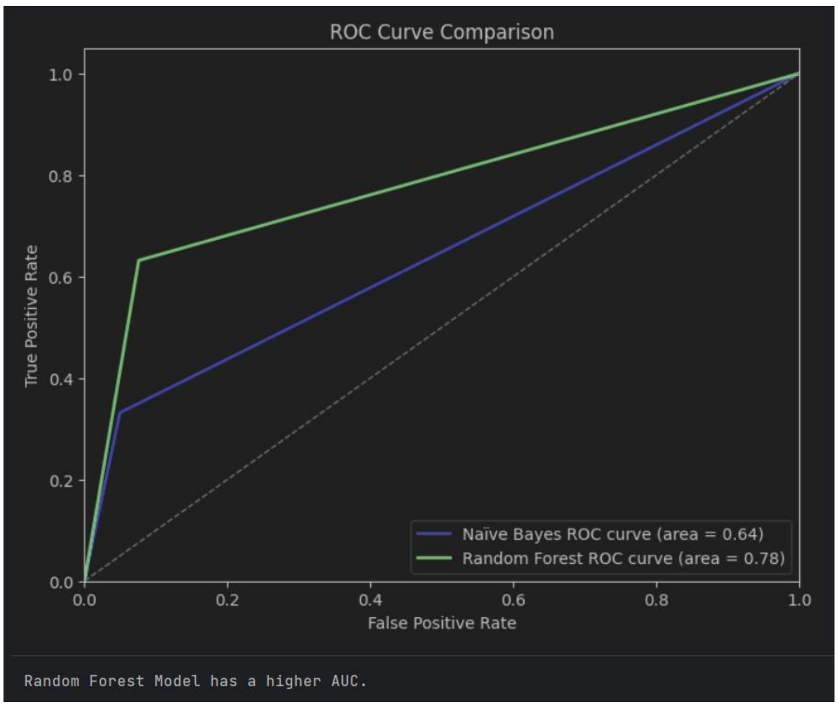
Recall, precision and F1 Score

Based on the comparison of models, Random Forest demonstrated superior performance with higher accuracy compared to Naïve Bayes. This suggests that Random Forest is better suited for the given task, as it achieves higher accuracy in predicting the target variable.



ROC Curve

Based on the comparison of ROC curve values, the Random Forest model outperforms the Naïve Bayes model. The Random Forest model exhibits a higher area under the ROC curve (AUC), indicating better discriminatory power and overall performance in binary classification tasks. Therefore, we can conclude that the Random Forest model is superior in this regard.



- Overall Random Forest model performs better that Naïve Base Model

Limitations

- Dependency on review Metrics: The conclusion is based only on the values of the ROC curve, ignoring other significant metrics that could offer a more thorough review, such as precision, recall, and F1-score.

- Data Imbalance Handling: Although class imbalance was addressed by the up-sampling technique, real-world circumstances where imbalance fluctuates between features and classes may be more complex than the technique can fully represent.

- Assumptions for Feature Encoding: The Label Encoding method may create biases that could potentially influence model performance, particularly for categorical features with ordinality assumptions.

- Overfitting Risk: While test data performance is the main focus of the evaluation, overfitting risks on training data are not specifically addressed, which may have an impact on the models' capacity to generalize.

Further Enhancements

- Feature Engineering: Explore additional features or transformations to improve model performance.
- Hyperparameter Tuning: Fine-tune model parameters to optimize predictive accuracy.
- Ensemble Methods: Experiment with ensemble techniques like stacking or boosting to enhance model robustness.
- Cross-Validation: Implement cross-validation techniques to assess model generalization on unseen data.
- Advanced Algorithms: Investigate advanced machine learning algorithms beyond Naïve Bayes and Random Forest for potentially better performance.

Reference

- *Adult* (no date) *UCI Machine Learning Repository*. Available at: https://archive.ics.uci.edu/dataset/2/adult (Accessed: 29 March 2024).

- *Data science - intro to statistics* (no date) *Data Science Statistics Intro*. Available at: https://www.w3schools.com/datascience/ds_stat_intro.asp (Accessed: 29 March 2024).

- GeeksforGeeks. (2020b). Interquartile Range to Detect Outliers in Data. [online] Available at: https://www.geeksforgeeks.org/interquartile-range-to-detect-outliers-in-data/ [Accessed 25 Mar. 2024].

- Scikit-learn (2018). 3.2.4.3.2. sklearn.ensemble.RandomForestRegressor — scikit-learn 0.20.3 documentation. [online] Scikit-learn.org. Available at: https://scikit learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html [Accessed 22 Mar. 2024].

GitHub Repository

Click here

Appendix – Code

```
from ucimlrepo import fetch_ucirepo  # Importing function to fetch dataset
import pandas as pd  # Importing pandas for data manipulation

# Fetch dataset
adult = fetch_ucirepo(id=2)  # Fetching the dataset with id 2 from UCI repository

# Extract features and target variable from dataset
X = adult.data.features
y = adult.data.targets

# Print metadata information
print(adult.metadata)  # Print metadata of the dataset
```

```
# Print variable information
print(adult.variables)  # Print information about variables (features and target)

import warnings
warnings.filterwarnings("ignore")


#%%


adult_X=pd.DataFrame(X) # Convert features to a pandas DataFrame

adult_X



#%%
adult_Y= pd.DataFrame(y) # Convert target variable to a pandas DataFrame
adult_Y

#%%
# Combine features and target variable into a single DataFrame
adult_data = pd.concat([adult_X,adult_Y], axis=1)
adult_data



#%% md
Finding Duplicates
#%%
adult_data.duplicated()
#%%

#%% md
Remove Duplicates
#%%
adult_data.drop_duplicates(inplace=True)
adult_data.drop_duplicates(inplace=True)
print("No of rows after droping duplicated rows: ", len(adult_data))




#%%
#find outliers
import pandas as pd
from scipy.stats import iqr
```

```
import numpy as np

def find_outliers_iqr(data):
    """
    Identify outliers in a pandas DataFrame using Interquartile Range (IQR).

    Args:
        data: pandas DataFrame containing the data.

    Returns:
        A dictionary containing:
            - inliers: DataFrame containing rows without outliers.
            - outliers: DataFrame containing rows with outliers (one column for each numeric
feature).
    """
    outliers = {}  # Dictionary to store outliers for each numeric column
    inliers = data.copy()  # Working on a copy for inliers

    for col in data.select_dtypes(include=[np.number]):
        Q1 = data[col].quantile(0.25)
        Q3 = data[col].quantile(0.75)
        IQR = Q3 - Q1  # Calculate IQR directly from Q1 and Q3
        lower_bound = Q1 - (1.5 * IQR)
        upper_bound = Q3 + (1.5 * IQR)

        # Identify outliers in this column
        outliers[col] = data[(data[col] < lower_bound) | (data[col] > upper_bound)]

        # Remove outliers from inliers dataframe
        inliers = inliers.loc[(inliers[col] >= lower_bound) & (inliers[col] <= upper_bound)]

    return inliers, outliers

# Example usage
adult_data_filtered = adult_data.copy()  # Working on a copy to avoid modifying original data
inliers, outliers_df = find_outliers_iqr(adult_data_filtered)

print("Number of inliers:", len(inliers))
print("Outlier Values.")

outliers_df



#%%
import matplotlib.pyplot as plt

def plot_outliers_comparison(data, col_name):
    # Plot with outliers
```

```
    plt.figure(figsize=(15, 5))
    plt.subplot(1, 2, 1)
    plt.boxplot([data[col_name]])
    plt.title(f"Distribution of {col_name} with outliers")
    plt.xlabel(col_name)
    plt.ylabel("Value")

    # Plot without outliers
    plt.subplot(1, 2, 2)
    Q1 = data[col_name].quantile(0.25)
    Q3 = data[col_name].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - (1.5 * IQR)
    upper_bound = Q3 + (1.5 * IQR)
    inliers = data.loc[(data[col_name] >= lower_bound) & (data[col_name] <= upper_bound)]
    plt.boxplot([inliers[col_name]])
    plt.title(f"Distribution of {col_name} without outliers")
    plt.xlabel(col_name)
    plt.ylabel("Value")

    plt.show()

# Example usage
col_to_plot = 'age'  # Choose the column you want to plot
plot_outliers_comparison(adult_data, col_to_plot)

col_to_plot = 'fnlwgt'  # Choose the column you want to plot
plot_outliers_comparison(adult_data, col_to_plot)

col_to_plot = 'education-num'  # Choose the column you want to plot
plot_outliers_comparison(adult_data, col_to_plot)

col_to_plot = 'capital-gain'  # Choose the column you want to plot
plot_outliers_comparison(adult_data, col_to_plot)

col_to_plot = 'capital-loss'  # Choose the column you want to plot
plot_outliers_comparison(adult_data, col_to_plot)

col_to_plot = 'hours-per-week'  # Choose the column you want to plot
plot_outliers_comparison(adult_data, col_to_plot)


#%%
import pandas as pd
from scipy.stats import iqr

def remove_outliers_iqr(data, cols):
  """
```

```
  Remove outliers from a pandas DataFrame using Interquartile Range (IQR).

  Args:
      data: pandas DataFrame containing the data.
      cols: List of column names to check for outliers.

  Returns:
      A new pandas DataFrame with outliers removed.
  """
  outliers = data.copy()  # Working on a copy for inliers

  for col in cols:
    if col in data.columns:  # Check if column exists before processing
      Q1 = data[col].quantile(0.25)
      Q3 = data[col].quantile(0.75)
      IQR = Q3 - Q1
      lower_bound = Q1 - (1.5 * IQR)
      upper_bound = Q3 + (1.5 * IQR)
      outliers = outliers.loc[(outliers[col] >= lower_bound) & (outliers[col] <=
upper_bound)]

  return outliers

# Check column names from downloaded data (replace with your actual check)
# if 'fnlwgt' not in adult.data.feature_names:
#   numeric_columns.remove('fnlwgt')  # Remove if 'fnlwgt' is missing
numeric_features = ['age', 'fnlwgt', 'education-num', 'capital-gain', 'capital-loss', 'hours-
per-week']
# Apply outlier removal to numeric columns
adult_data_filtered = adult_data.copy()  # Working on a copy to avoid modifying original data
adult_data_filtered = remove_outliers_iqr(adult_data_filtered, numeric_features)

print("Number of rows after removing outliers:", len(adult_data_filtered))
adult_data_filtered
#%%
# Replace '?' with missing values (NaN)
adult_data.replace('?',pd.NA,inplace=True)
adult_data




#%%
# Drop rows with missing values
cleaned_adult_df = adult_data.dropna()
cleaned_adult_df

#%%
```

```
cleaned_adult_df.isna().sum().any()
cleaned_adult_df.isna().sum()
#%%
# Import LabelEncoder to encode categorical variables
from sklearn.preprocessing import LabelEncoder


#%%
# Define categorical features to be encoded
categorical_features = ['workclass', 'education', 'marital-status', 'occupation',
'relationship', 'race', 'sex', 'native-country']

Label_Encoder = {}
# Initialize a dictionary to store LabelEncoder objects
#%%
# Encode categorical features using LabelEncoder
for col in categorical_features:
    Label_Encoder[col] = LabelEncoder()
    cleaned_adult_df[col] = Label_Encoder[col].fit_transform(cleaned_adult_df[col])





#%%
# Import pandas library
import pandas as pd

# Assuming cleaned_adult_df contains the dataset after cleaning and encoding

# Print the encoded dataset
print(cleaned_adult_df)




#%%
from sklearn.utils import resample

# Separate data into '>50K' and '<=50K' income categories
income_greater_than_50k = cleaned_adult_df[cleaned_adult_df['income'] == 1]
income_less_than_or_equal_50k = cleaned_adult_df[cleaned_adult_df['income'] == 0]

# Determine the number of samples needed to balance the dataset
num_samples_needed = len(income_greater_than_50k) - len(income_less_than_or_equal_50k)

# Check if upsampling is required
if num_samples_needed > 0:
    # Upsample the '<=50K' subset to match the size of the '>50K' subset
```

```
    income_less_than_or_equal_50k_upsampled = resample(income_less_than_or_equal_50k,
                                                       replace=True,
                                                       n_samples=num_samples_needed,
                                                       random_state=42)

    # Combine the upsampled '<=50K' subset with the '>50K' subset
    balanced_data = pd.concat([income_greater_than_50k,
income_less_than_or_equal_50k_upsampled])

    # Summary statistics for the balanced data
    greater_than_50k_summary = income_greater_than_50k.describe()
    less_than_or_equal_50k_summary = income_less_than_or_equal_50k.describe()
    balanced_data_summary = balanced_data.describe()

    # Print summaries
    print("Summary statistics for income greater than $50k:")
    print(greater_than_50k_summary)

    print("\nSummary statistics for income less than or equal to $50k:")
    print(less_than_or_equal_50k_summary)

    print("\nSummary statistics for balanced data after addressing class imbalance:")
    print(balanced_data_summary)
else:
    print("The dataset is already balanced. No upsampling is needed.")



#%%
# Get counts for income below 50k and above 50k
below_50k_count = cleaned_adult_df[cleaned_adult_df['income'] == 0].shape[0]
above_50k_count = cleaned_adult_df[cleaned_adult_df['income'] == 1].shape[0]

# Display the counts
print("Income with below 50k count:", below_50k_count)
print("Income with above 50k count:", above_50k_count)

#%%
cleaned_adult_df



#%%
# Import StandardScaler to scale numerical features
from sklearn.preprocessing import StandardScaler

# Define numerical features to be scaled
numeric_features = ['age', 'fnlwgt', 'education-num', 'capital-gain', 'capital-loss', 'hours-
per-week']
```

```
scaler = StandardScaler()
cleaned_adult_df[numeric_features]= scaler.fit_transform(cleaned_adult_df[numeric_features])

#%%
cleaned_adult_df
#%%


from sklearn.model_selection import train_test_split

X = cleaned_adult_df.drop(columns=['income']) # all columns without income

y = cleaned_adult_df['income'] # target variable column

# Split 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)

cleaned_adult_df
#%%
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

#%%

# Split the dataset 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)

#%%
# Initialize and train Naïve Bayes model
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)

# Evaluate Naïve Bayes model
nb_predictions = nb_model.predict(X_test)
nb_accuracy = accuracy_score(y_test, nb_predictions)
print("Classification Report")
print("Naïve Bayes Model Accuracy:", nb_accuracy)
print(" ")
print(classification_report(y_test, nb_predictions))


#%%
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
```

```
# Generate confusion matrix for Naïve Bayes model
nb_confusion_matrix = confusion_matrix(y_test, nb_predictions)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(nb_confusion_matrix, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix for Naïve Bayes Model')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
#%%

#%%
from sklearn.metrics import accuracy_score

# Calculate accuracy on test data
test_accuracy = accuracy_score(y_test, nb_predictions)
print("Accuracy on Testing Data:", test_accuracy)
#%%
train_accuracy = accuracy_score(y_train,nb_model.predict(X_train))
print("Accuracy of training data :", train_accuracy)
#%%
from sklearn.metrics import roc_curve, auc, roc_auc_score
from sklearn.metrics import classification_report,
confusion_matrix,roc_curve,accuracy_score

#get probabilities for positive class
y_prob_rf = nb_model.predict_proba(X_test)[:,1]

# Compute ROC curve and ROC area for Naïve Bayes model
fpr_nb, tpr_nb, thresholds_nb = roc_curve(y_test, nb_predictions)
roc_auc_nb = auc(fpr_nb, tpr_nb)

AUC_NB = roc_auc_score(y_test, y_prob_rf)

# Plot ROC curve for Naïve Bayes model
plt.figure()
plt.plot(fpr_nb, tpr_nb, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' %
roc_auc_nb)
plt.plot([0, 1], [0, 1], color='navy', 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) for Naïve Bayes Model')
plt.legend(loc="lower right")
plt.show()
print("Receiver Operating Characteristic (ROC) curve value for Naïve Bayes Model
```

```
:",AUC_NB) from sklearn.metrics import roc_curve, auc, roc_auc_score
from sklearn.metrics import classification_report, confusion_matrix,roc_curve,accuracy_score

#get probabilities for positive class
y_prob_rf = nb_model.predict_proba(X_test)[:,1]

# Compute ROC curve and ROC area for Naïve Bayes model
fpr_nb, tpr_nb, thresholds_nb = roc_curve(y_test, nb_predictions)
roc_auc_nb = auc(fpr_nb, tpr_nb)

AUC_NB = roc_auc_score(y_test, y_prob_rf)

# Plot ROC curve for Naïve Bayes model
plt.figure()
plt.plot(fpr_nb, tpr_nb, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' %
roc_auc_nb)
plt.plot([0, 1], [0, 1], color='navy', 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) for Naïve Bayes Model')
plt.legend(loc="lower right")
plt.show()
print("Receiver Operating Characteristic (ROC) curve value for Naïve Bayes Model :",AUC_NB)

#%%


# Initialize and train Random Forest model
rf_model = RandomForestClassifier()
rf_model.fit(X_train, y_train)




#%%
# Make predictions on the test data
y_pred = rf_model.predict(X_test)


#%%
# Make predictions on the training data
y_pred_train = rf_model.predict(X_train)
#%%

#hyperparameter tuning for random forest
from sklearn.model_selection import RandomizedSearchCV
```

```
param_grid_rf = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10]
}
rf_classifier_tuned = RandomizedSearchCV(RandomForestClassifier(random_state=42),
param_distributions=param_grid_rf, n_iter=10, cv=5, scoring='accuracy')
rf_classifier_tuned.fit(X_train, y_train)

print("Best hyperparameters:", rf_classifier_tuned.best_params_)
print("Best cross-validation score:", rf_classifier_tuned.best_score_)

#%%
y_pred = rf_classifier_tuned.best_estimator_.predict(X_test)
#%%
from sklearn.metrics import accuracy_score


# Calculate testing data accuracy
rf_accuracy_test = accuracy_score(y_test, y_pred)
print("Random Forest Model Testing Accuracy:", rf_accuracy_test)



#%%
from sklearn.metrics import classification_report

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

# Generate classification report for testing data
rf_classification_report_test = classification_report(y_test, rf_predictions_test)

# Print the classification report
print("Classification Report for Random Forest Model (Testing Data):\n",
rf_classification_report_test)



#%%
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Generate confusion matrix for testing data in Random Forest model
rf_confusion_matrix_test = confusion_matrix(y_test, rf_predictions_test)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(rf_confusion_matrix_test, annot=True, fmt='d', cmap='Blues')
```

```python
plt.title('Confusion Matrix for Random Forest Model (Testing Data)')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()

#%%
#Accuracy of training data
from sklearn.metrics import accuracy_score

# Make predictions on the training data
y_pred_train_rf = rf_classifier_tuned.best_estimator_.predict(X_train)

# Calculate training data accuracy
rf_accuracy_train = accuracy_score(y_train, y_pred_train_rf)
print("Random Forest Model Training Accuracy:", rf_accuracy_train)


#%%
from sklearn.metrics import classification_report

# Generate classification report for the training data set in Random Forest model
rf_classification_report_train = classification_report(y_train, y_pred_train)

# Print the classification report
print("Classification Report for Random Forest Model (Training Data):\n",
rf_classification_report_train)






#%%
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Generate confusion matrix for training data in Random Forest model
rf_confusion_matrix_train = confusion_matrix(y_train, y_pred_train)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(rf_confusion_matrix_train, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix for Random Forest Model (Training Data)')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

```
#%%

# Get a single decision tree from the Random Forest
tree = rf_model.estimators_[0]

# Visualize the tree (you may need to install graphviz and pydotplus)
# Note: Visualizing a single tree from Random Forest is optional and can be resource-
intensive
from sklearn.tree import plot_tree
plt.figure(figsize=(20, 10))
plot_tree(tree, filled=True, feature_names=X.columns)
plt.show()



#%%
from sklearn.metrics import roc_curve, auc, roc_auc_score

# Get probabilities for positive class
y_prob_rf = rf_model.predict_proba(X_test)[:, 1]

# Compute ROC curve and ROC area for Random Forest model
fpr_rf, tpr_rf, thresholds_rf = roc_curve(y_test, y_prob_rf)
AUC_RF = auc(fpr_rf, tpr_rf)

# Plot ROC curve for Random Forest model
plt.figure()
plt.plot(fpr_rf, tpr_rf, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % AUC_RF)
plt.plot([0, 1], [0, 1], color='navy', 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) for Random Forest Model')
plt.legend(loc="lower right")
plt.show()

print("Receiver Operating Characteristic (ROC) curve value for Random Forest Model :",
AUC_RF)

#%%
# Compare models based on accuracy
import matplotlib.pyplot as plt

# AUC scores
AUC_scores = [AUC_NB, AUC_RF]
models = ['Naïve Bayes', 'Random Forest']
```

```python
plt.bar(models, AUC_scores, color=['blue', 'green'])
plt.xlabel('Models')
plt.ylabel('AUC Score')
plt.title('Comparison of Models based on AUC Score')
plt.show()



if AUC_NB > AUC_RF :
    print("Naïve Bayes Model performs better.")
elif AUC_RF > AUC_NB:
    print("Random Forest Model performs better.")
else:
    print("Both models perform equally.")


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

# Compute ROC curve and ROC area for Naïve Bayes model
fpr_nb, tpr_nb, _ = roc_curve(y_test, nb_predictions)
roc_auc_nb = auc(fpr_nb, tpr_nb)

# Compute ROC curve and ROC area for Random Forest model
fpr_rf, tpr_rf, _ = roc_curve(y_test, rf_predictions)
roc_auc_rf = auc(fpr_rf, tpr_rf)

# Plot ROC curves
plt.figure(figsize=(8, 6))
plt.plot(fpr_nb, tpr_nb, color='blue', lw=2, label='Naïve Bayes ROC curve (area = %0.2f)' %
roc_auc_nb)
plt.plot(fpr_rf, tpr_rf, color='green', lw=2, label='Random Forest ROC curve (area = %0.2f)'
% roc_auc_rf)
plt.plot([0, 1], [0, 1], color='gray', lw=1, 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('ROC Curve Comparison')
plt.legend(loc="lower right")
plt.show()

# Determine the best model based on AUC
if roc_auc_nb > roc_auc_rf:
    print("Naïve Bayes Model has a higher AUC.")
elif roc_auc_nb < roc_auc_rf:
    print("Random Forest Model has a higher AUC.")
```

```
else:
    print("Both models have the same AUC.")
```