Informatics Institute of Technology

Department of Computing

BSc (Hons) Artificial Intelligence and Data Science

Module: CM2604: Machine Learning

Module Coordinator: Sahan Priyanayana

Coursework Report

Student Name	: Damitha Udara Weerasinghe
RGU ID number	: 2236765
IIT ID number	: 20210669

Contents Page

1. Introduction

This report investigates the use of machine learning models to predict whether an individual's income exceeds $50K per year based on the Adult Census Income dataset (https://archive.ics.uci.edu/dataset/2/adult). We compare the performance of two common models (Naive Bayes and Random Forest) to identify the most effective approach for this classification task.

2. Corpus Co-operation

2.1 Pre-Processing

Loading the dataset

The initial data consisted of two separate files: "adult.data" and "adult.test". The "adult.data" file contained 32,561 entries, while the "adult.test" file contained 16,281 entries. both datasets were combined using pandas' concatenation functionality, resulting in a single dataframe with a total of 48,842 entries.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48842 entries, 0 to 48841
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             48842 non-null  int64
 1   workclass       46043 non-null  object
 2   fnlwgt          48842 non-null  int64
 3   education       48842 non-null  object
 4   education-num   48842 non-null  int64
 5   marital-status  48842 non-null  object
 6   occupation      46033 non-null  object
 7   relationship    48842 non-null  object
 8   race            48842 non-null  object
 9   sex             48842 non-null  object
 10  capital-gain    48842 non-null  int64
 11  capital-loss    48842 non-null  int64
 12  hours-per-week  48842 non-null  int64
 13  native-country  47985 non-null  object
 14  income          48842 non-null  object
dtypes: int64(6), object(9)
memory usage: 5.6+ MB
```

Data inspection revealed the presence of missing values represented by "?" symbols. these symbols were replaced with the standard missing value indicator "NaN" using the na_values='?' argument during data loading. the "adult.test" dataset contained missing values represented by "." characters. These were replaced with empty strings using the replace() function.

Filling null values

All the ‘NaN’ indicators will be filled using Forward Fill method. This method replaces "NaN" values with the preceding non-missing value within each column, assuming a sequential trend in the data.

Checking for duplicates and dropping those entries

The ‘df.duplicated().values.any()’ function confirmed their presence, and subsequent application of ‘drop_duplicates()’ ensured their elimination. A final check verified that all duplicates were successfully addressed.

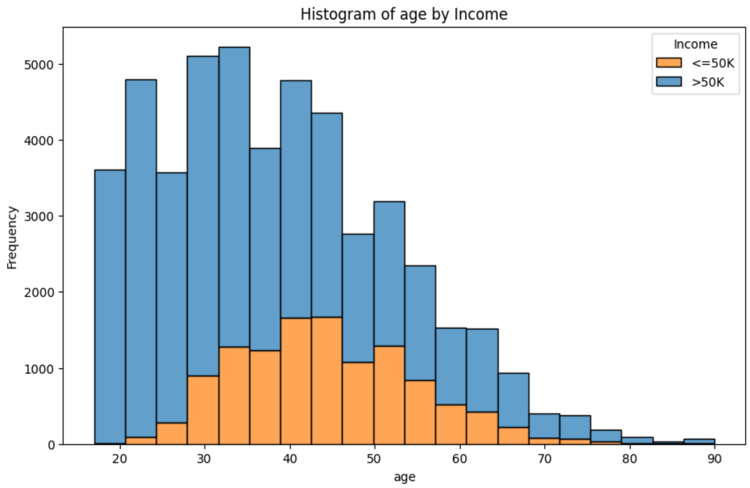
```
df.duplicated().values.any()
```

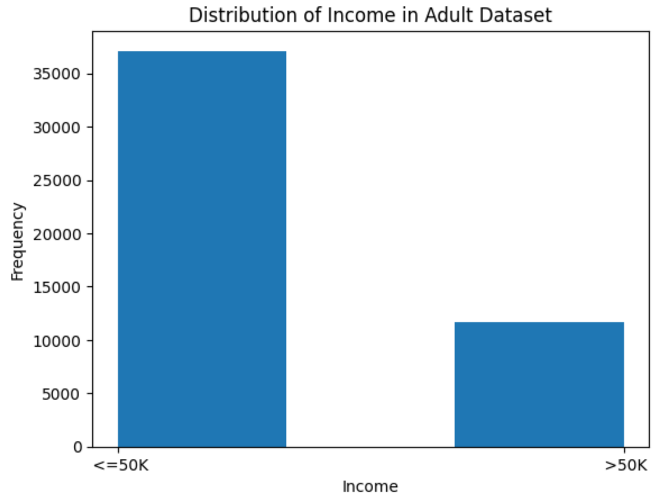
```
True
```

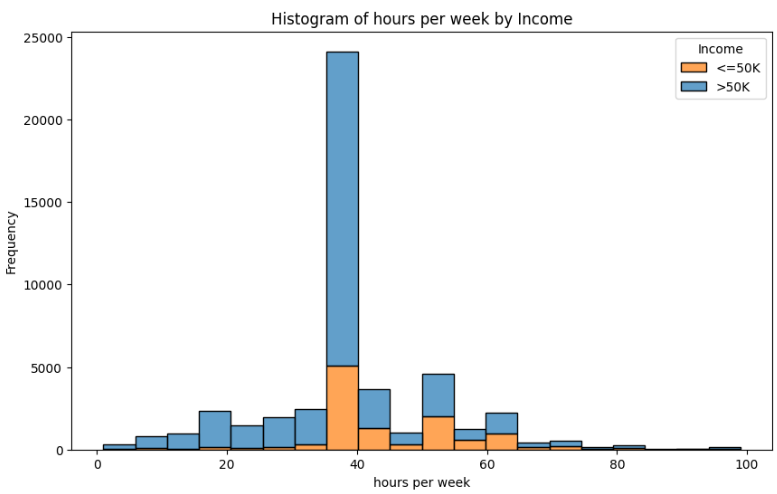
```
print("Before: ", df.duplicated().sum())

duplicates = df.duplicated()
df = df.loc[~duplicates]

print("After: ", df.duplicated().sum())

display(df)
```

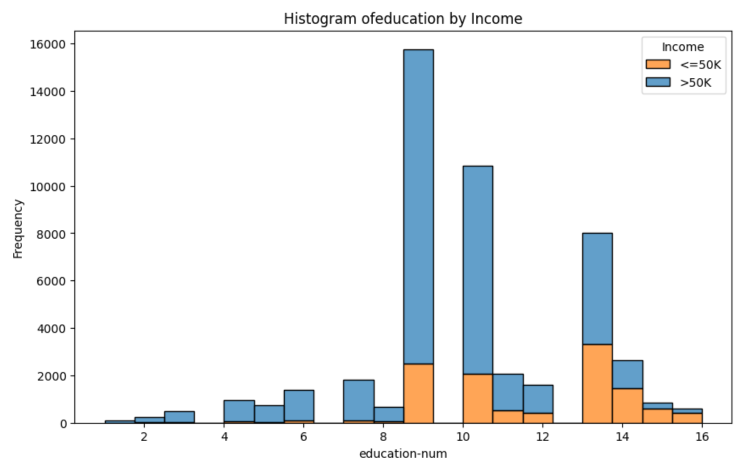
```
Before:  52
After:  0
```

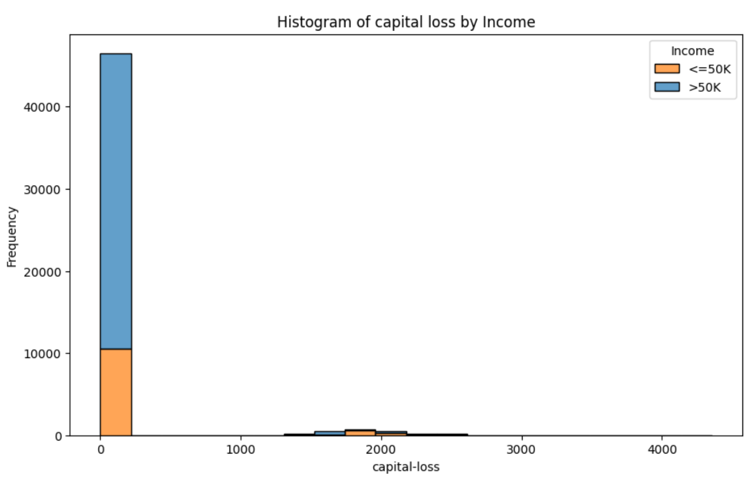
Data visualization

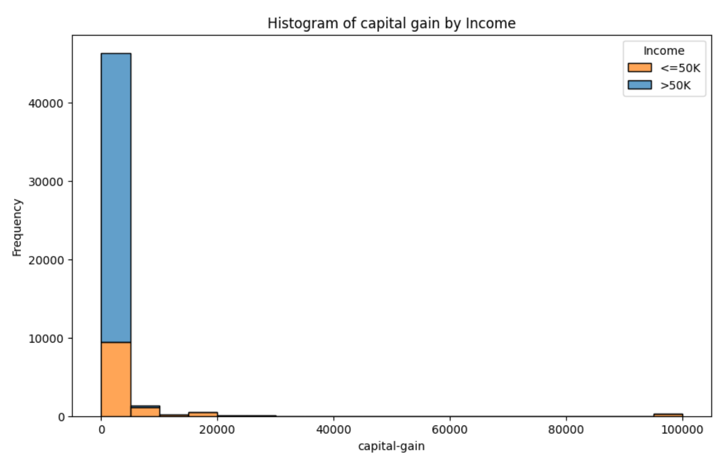






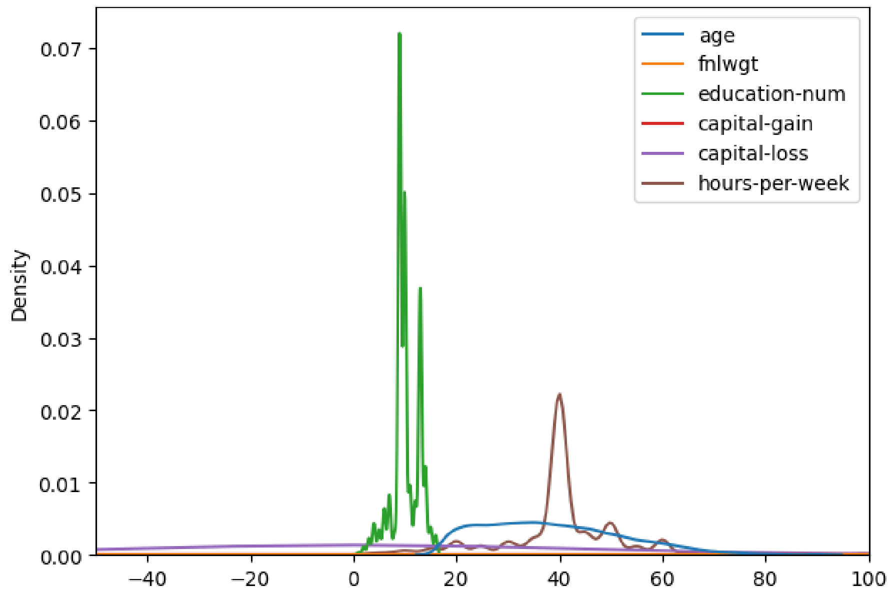


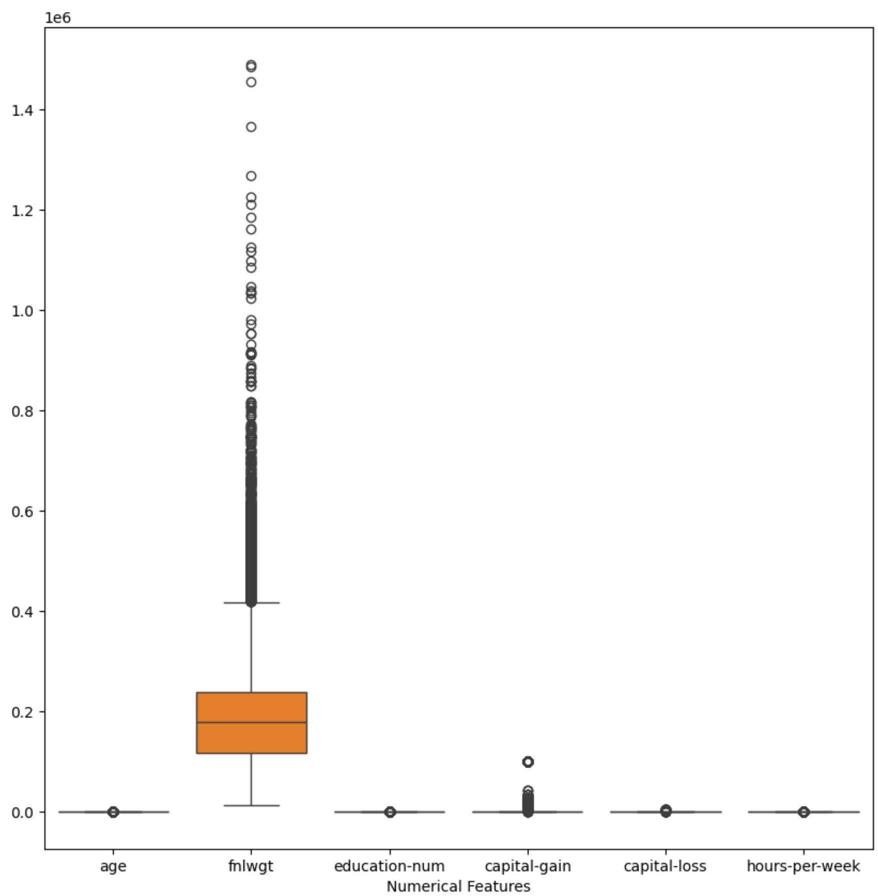




Feature Scaling

Numeric columns (using StandardScaler) were standardized to have a mean of 0 and standard deviation of 1.





Handling outliers

Tackled outliers in specific features ("age", "hours-per-week", "capital-gain", and "capital-loss") employing the Interquartile Range (IQR) method to identify potential outliers.

```
Feature: age
lower bound: -2.0
upper bound: 78.0
```

```
Feature: hours-per-week
lower bound: 32.5
upper bound: 52.5
```

```
Feature: capital-gain
lower bound: 0.0
upper bound: 0.0
```

```
Feature: capital-loss
lower bound: 0.0
upper bound: 0.0
```

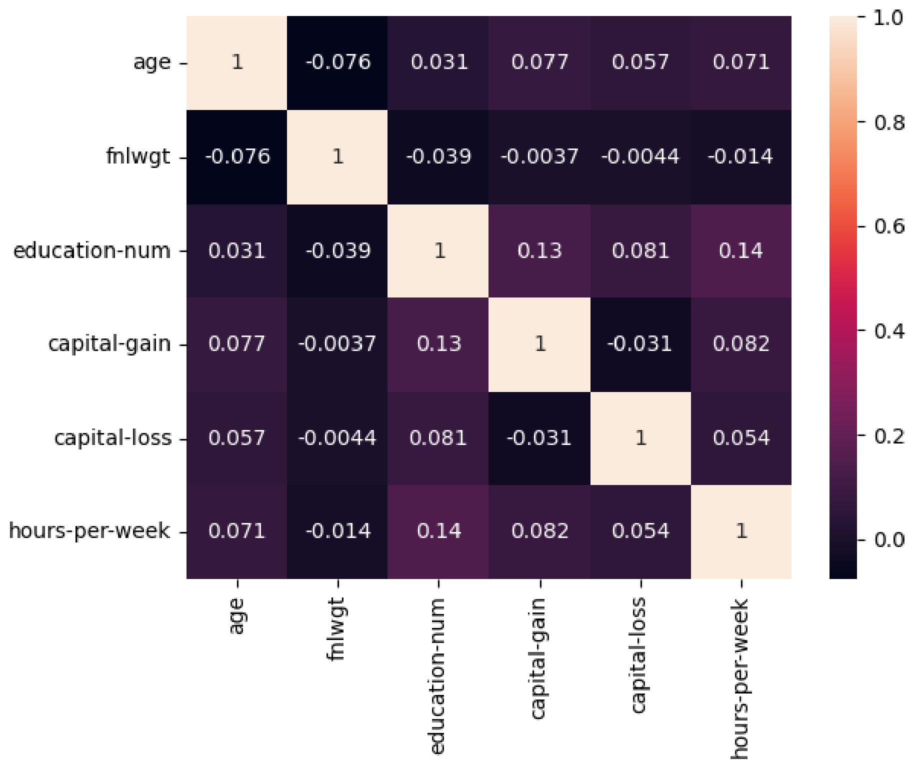
For "capital gain" and "capital loss" resulted in zero as both the lower and upper bounds. This likely indicates that the data for these features doesn't exhibit significant outliers. Since the IQR method identifies outliers based on deviations from the quartiles, values in these features might be naturally clustered around zero with minimal extreme values.

Feature Scaling

numeric features were standardized using 'StandardScaler'. This transforms them to have a mean of 0 and standard deviation of 1, creating a level playing field for all features and preventing bias from features with larger scales. The result is a preprocessed dataset ready for further exploration.

Correlation Analysis

Investigating potential multicollinearity in the data by calculating the correlation between all features and visualizing them using a heatmap. A correlation threshold is set (0.7) to identify features with very strong linear relationships.



X,y split

Scaled data is split into inputs(X), and target output(y) data sets.

3. Solution Methodology

3.1 Random Forest Classifier

A Random Forest model was trained to learn from the data and make predictions. To evaluate its success, the data was split, the model was trained on one part, and its performance was assessed on the other part using accuracy metrics and a confusion matrix. This helps us understand how well the model generalizes to unseen data.

3.2 Gaussian Naive Bayes Classifier

The Logistic Regression model's performance was evaluated using accuracy metrics, a classification report, and a confusion matrix visualization. These metrics assess both overall accuracy and class-specific performance. Additionally, the model's ability to handle continuous variables and its probabilistic nature make it suitable for various classification tasks.

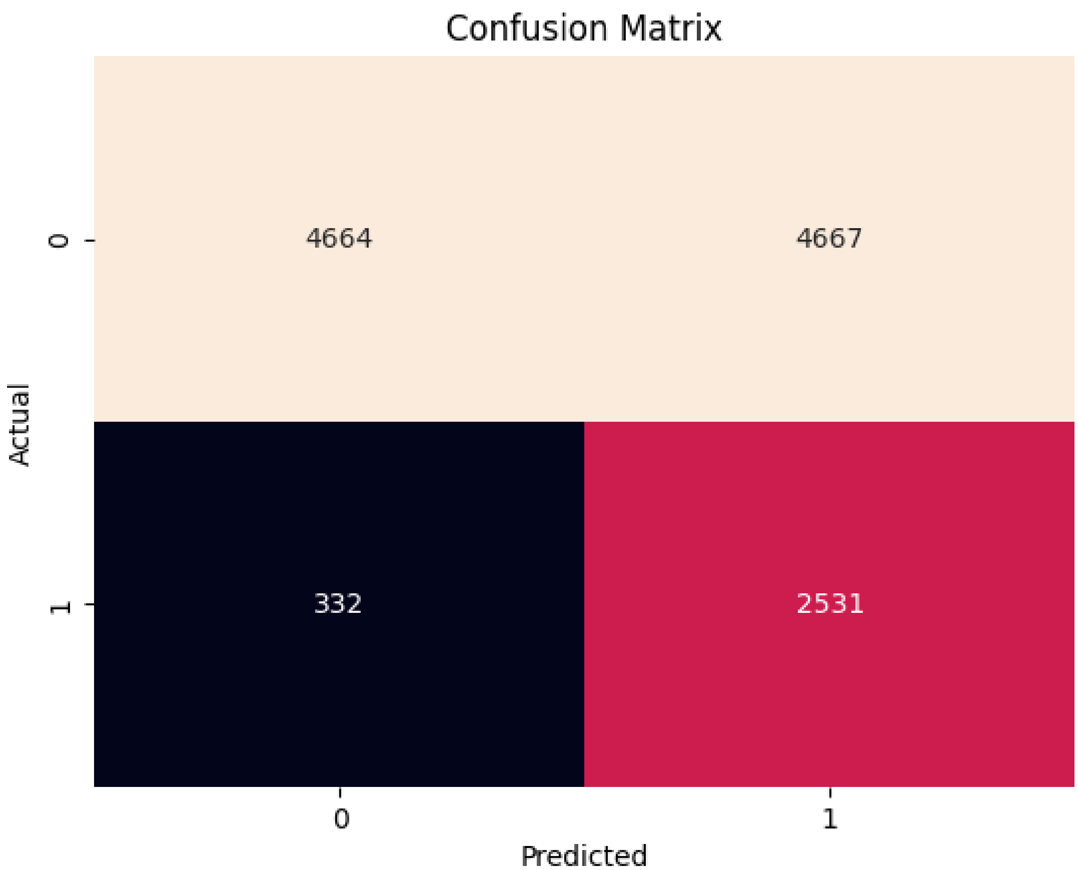
4. Model Evaluation

4.1 Test data outputs

Gaussian Naïve Bayes Classifier

Classification report:

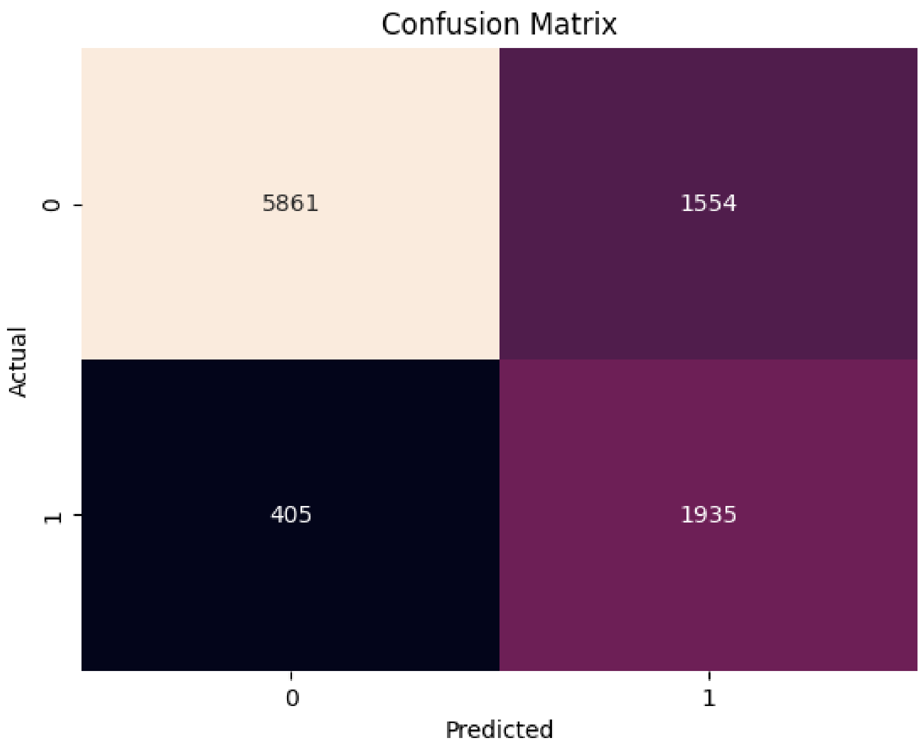
	precision	recall	f1-score	support
<=50K	0.93	0.50	0.65	9331
>50K	0.35	0.88	0.50	2863
accuracy			0.59	12194
macro avg	0.64	0.69	0.58	12194
weighted avg	0.80	0.59	0.62	12194



Random Forest Classifier

Classification report:

	precision	recall	f1-score	support
<=50K	0.94	0.79	0.86	7415
>50K	0.55	0.83	0.66	2340
accuracy			0.80	9755
macro avg	0.74	0.81	0.76	9755
weighted avg	0.84	0.80	0.81	9755

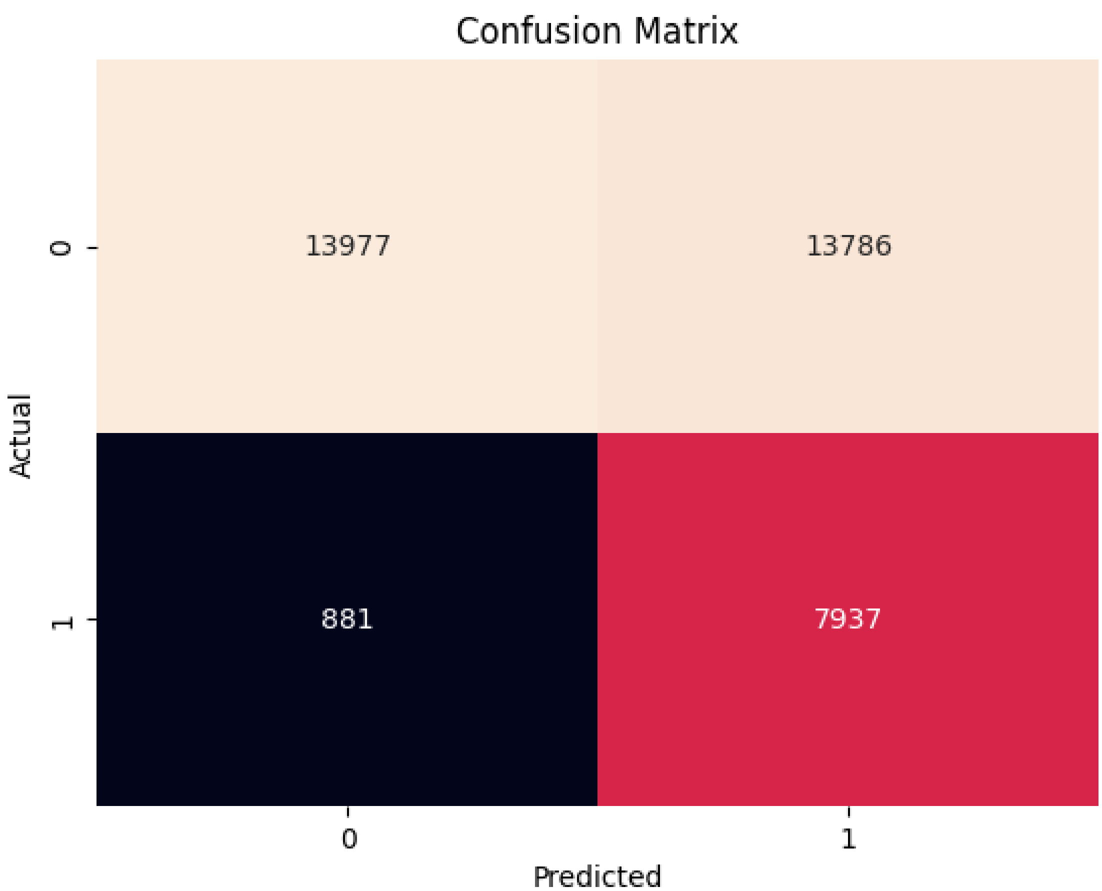


4.2 Train data outputs

Gaussian Naïve Bayes Classifier

Classification report:

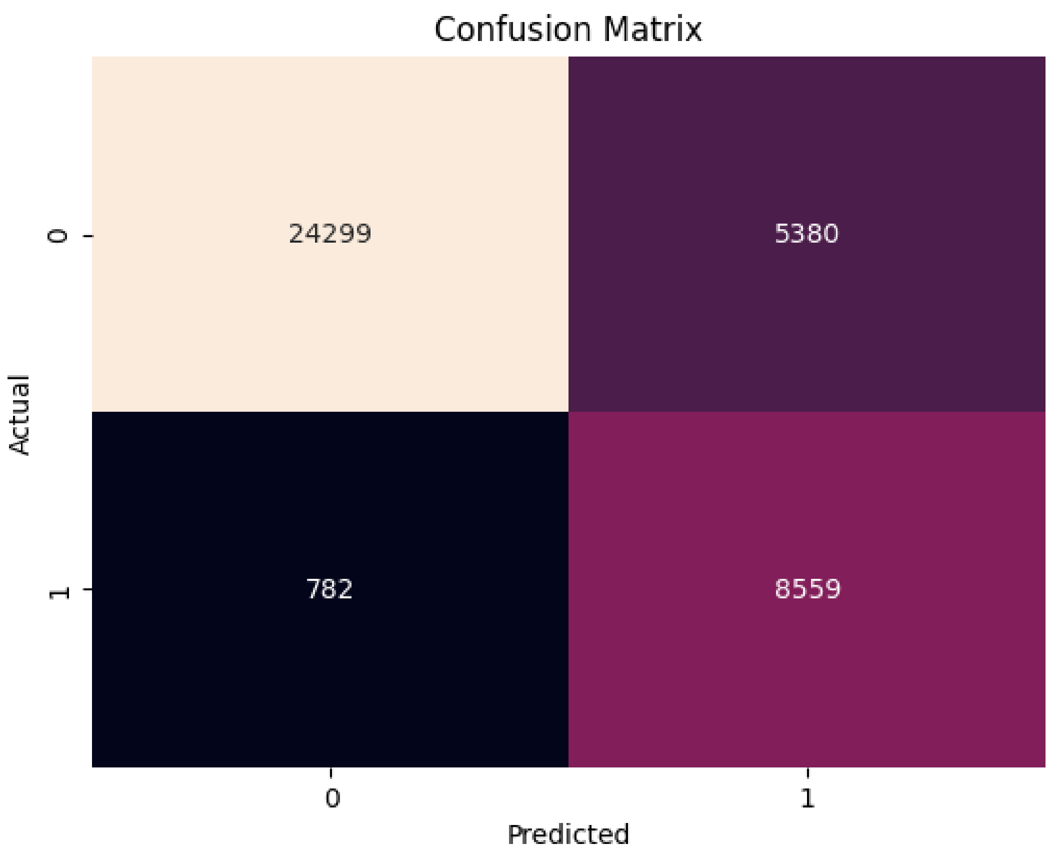
	precision	recall	f1-score	support
<=50K	0.94	0.50	0.66	27763
>50K	0.37	0.90	0.52	8818
accuracy			0.60	36581
macro avg	0.65	0.70	0.59	36581
weighted avg	0.80	0.60	0.62	36581



Random Forest Classifier

Classification report:

	precision	recall	f1-score	support
<=50K	0.97	0.82	0.89	29679
>50K	0.61	0.92	0.74	9341
accuracy			0.84	39020
macro avg	0.79	0.87	0.81	39020
weighted avg	0.88	0.84	0.85	39020



Summary

	Naïve Bayes	**Random Forest**
Test Data		
Accuracy	59%	80%
Train Data		
Accuracy	60%	84%

Random Forest achieves a significantly higher overall accuracy (80%) compared to Naive Bayes (59%). This indicates that Random Forest correctly predicts income labels for a much larger proportion of data points.

5. Limitations

This analysis has some limitations:

- **Inconsistent Data:** Random errors or inconsistencies within the data can be misleading for the model. This can lead the model to develop inaccurate predictions.
- **Missing Information:** Incomplete data, where some values are missing, can also hinder the model's ability to learn effectively. The model may struggle to identify the underlying relationships within the data and ultimately deliver less accurate results.

6. Future Enhancements

- Model Ensembling: Combining predictions from multiple models for more accurate results.

7. Final Code

GitHub REPO LINK - https://github.com/DamithaWee/Census-Income-prediction.git

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sb
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder, LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from imblearn.combine import SMOTEENN
from imblearn.pipeline import Pipeline
from imblearn.over_sampling import RandomOverSampler, SMOTE
from imblearn.under_sampling import RandomUnderSampler
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix, roc_curve, roc_auc_score
```

LOAD DATASET

```
# defind column names
columns = ['age', 'workclass', 'fnlwgt', 'education', 'education-num', 'marital-
status', 'occupation', 'relationship',
           'race', 'sex', 'capital-gain', 'capital-loss', 'hours-per-week',
'native-country', 'income']

# load data while replacing ? with NaN
data = pd.read_csv('drive/MyDrive/IIT/ML/adult.data', names=columns, na_values='
?')
print(data.shape)
test = pd.read_csv('drive/MyDrive/IIT/ML/adult.test', names=columns, na_values='
?', skiprows=1)
print(test.shape)

# remove dots at the end the line in audlt.test file
test.loc[:, 'income'] = test['income'].replace(regex=True, to_replace=r'\.',
value=r'')

# Combine two datasets
df = pd.concat([data, test], ignore_index=True)
df.reset_index(drop=True,inplace=True)

display(df)
# total record and columns
```

```
df.shape
# variations of income values
df['income'].value_counts()
# dataset information
df.info()
```

#FILLING NULL VALUES

```
df.replace('?', np.NaN,inplace = True)
# replacing with foward fill method
df.fillna(method = 'ffill', inplace = True)

display(df)
```

CHECK FOR DUPLICATES

```
# Check for duplications
df.duplicated().values.any()
# Display number of duplicated entries before dropping
print("Before: ", df.duplicated().sum())

duplicates = df.duplicated()
df = df.loc[~duplicates]

# Display number of duplicated entries after dropping
print("After: ", df.duplicated().sum())

display(df)
# Check for duplications
df.duplicated().values.any()
```

#DATA VISUALIZATION

```
# distribution of income
plt.hist(df["income"], bins=3)
plt.xlabel("Income")
plt.ylabel("Frequency")
plt.title("Distribution of Income in Adult Dataset")
plt.show()
```

```
# age/ income
plt.figure(figsize=(10, 6))
sb.histplot(data=df, x='age', hue='income', bins=20, alpha=0.7, multiple='stack')
plt.title('Histogram of age by Income')
plt.xlabel('age')
plt.ylabel('Frequency')
```

```
plt.legend(title='Income', labels=['<=50K', '>50K'])
plt.show()
```

```
# hours per week/ income
plt.figure(figsize=(10, 6))
sb.histplot(data=df, x='hours-per-week', hue='income', bins=20, alpha=0.7,
multiple='stack')
plt.title('Histogram of hours per week by Income')
plt.xlabel('hours per week')
plt.ylabel('Frequency')
plt.legend(title='Income', labels=['<=50K', '>50K'])
plt.show()
```

```
# education num/ income
plt.figure(figsize=(10, 6))
sb.histplot(data=df, x='education-num', hue='income', bins=20, alpha=0.7,
multiple='stack')
plt.title('Histogram ofeducation by Income')
plt.xlabel('education-num')
plt.ylabel('Frequency')
plt.legend(title='Income', labels=['<=50K', '>50K'])
plt.show()
```

```
# capital gain/ income
plt.figure(figsize=(10, 6))
sb.histplot(data=df, x='capital-gain', hue='income', bins=20, alpha=0.7,
multiple='stack')
plt.title('Histogram of capital gain by Income')
plt.xlabel('capital-gain')
plt.ylabel('Frequency')
plt.legend(title='Income', labels=['<=50K', '>50K'])
plt.show()
```

```
# capital loss/ income
plt.figure(figsize=(10, 6))
sb.histplot(data=df, x='capital-loss', hue='income', bins=20, alpha=0.7,
multiple='stack')
plt.title('Histogram of capital loss by Income')
plt.xlabel('capital-loss')
plt.ylabel('Frequency')
plt.legend(title='Income', labels=['<=50K', '>50K'])
plt.show()
```

```
# Density plot
fig, ax = plt.subplots(figsize=(7,5))
```

```
sb.kdeplot(data=df, ax=ax)
ax.set_xlim(-50,100)

plt.show()
```

BOXPLOT FOR NUMERICAL FEATURES

```
plt.figure(figsize=(10, 10))
sb.boxplot(data=df)
plt.xlabel("Numerical Features")
plt.show()
```

HANDLING OUTLIERS

```
# function to find outliers
def calIqr(feature):
  q1 = df[feature].quantile(0.25)
  q3 = df[feature].quantile(0.75)
  iqr = q3 - q1
  lower_bound = q1 - 1.5 * iqr
  upper_bound = q3 + 1.5 * iqr
  print(f"Feature: {feature}\nlower bound: {lower_bound}\nupper bound:
{upper_bound}")
  return lower_bound, upper_bound
```

```
age_lower, age_upper = calIqr("age")
df["age"] = np.clip(df["age"], age_lower, age_upper)
```

```
hpw_lower, hpw_upper = calIqr("hours-per-week")
df["hours-per-week"] = np.clip(df["hours-per-week"], hpw_lower, hpw_upper)
```

```
calIqr("capital-gain")
df["capital-gain"] = np.clip(df["capital-gain"], hpw_lower, hpw_upper)
```

```
calIqr("capital-loss")
df["capital-loss"] = np.clip(df["capital-loss"], hpw_lower, hpw_upper)
```

CHECK FOR DUPLICATES

```
df.duplicated().values.any()
print("Before: ", df.duplicated().sum())

duplicates = df.duplicated()
df = df.loc[~duplicates]
```

```
print("After: ", df.duplicated().sum())

display(df)
df.duplicated().values.any()
```

#SCALE DATASET

```
# Create a StandardScaler object
scaler = StandardScaler()

# Select numeric features
scaledVals = scaler.fit_transform(df.select_dtypes(include=['int64', 'float64']))
# Standardize the numeric data
dfScaled = pd.DataFrame(scaledVals, columns=df.select_dtypes(include=['int64',
'float64']).columns)
# Reset index
dfScaled.reset_index(drop=True, inplace=True)
# Combine scaled numeric data with non-numeric features
dfScaled = pd.concat([dfScaled, df.select_dtypes(exclude=['int64',
'float64']).reset_index(drop=True)], axis=1)

display(dfScaled)


# numeric_cols = df.select_dtypes(include=['int64', 'float64']).columns
# scaler = StandardScaler()
# scaled_data = scaler.fit_transform(df[numeric_cols])
# df_scaled = pd.DataFrame(scaled_data, columns=numeric_cols)
# dfScaled = pd.concat([df_scaled, df.select_dtypes(exclude=['int64',
'float64'])], axis=1)
# dfScaled.reset_index(drop=True, inplace=True)

# display(dfScaled)
```

#CHECK FOR CORRELATION

```
# Calculate correlation matrix
correlation = df.corr()
sb.heatmap(correlation, annot=True)
plt.show()
```

```
# correlation threshold
correlation_threshold = 0.7
```

```
highly_correlated = set()

# Iterate through correlation matrix
for col in correlation.columns:
    for other_col in correlation.columns:
      # Check for absolute correlation exceeding threshold
        if col != other_col and abs(correlation[col][other_col]) >
correlation_threshold:
            highly_correlated.add(tuple(sorted([col, other_col])))

if highly_correlated:
    print("Highly correlated features (absolute correlation > ",
correlation_threshold, "):")
    for item in highly_correlated:
        print(item)
else:
    print("No highly correlated features found after oversampling.")
```

#X,y split

```
X = dfScaled.drop('income', axis=1)
y = dfScaled['income']

print(f"X: {len(X)} | y: {len(y)}")
```

TRAIN AND TEST THE NAIVE BAYES CLASSIFICATION

```
# Label encoding
label_categorical = ['workclass', 'education', 'marital-status', 'occupation',
'relationship', 'race', 'sex', 'native-country']
X_categorical = pd.get_dummies(X[label_categorical], drop_first=True)
encoded_features = pd.concat([X.drop(label_categorical, axis=1), X_categorical],
axis=1)

# train test split
X_train, X_test, y_train, y_test = train_test_split(encoded_features, y,
test_size=0.25)
```

```
# create naive bayes model and train
# nbModel = GaussianNB()
# nbModel.fit(X_train, y_train)

pipeline_nb = Pipeline([
    ('sampling', SMOTEENN()),  # Over- and undersampling
```

```
    ('classifier', GaussianNB())  # Classifier
])

pipeline_nb.fit(X_train, y_train)
```

```
# test data predict
y_pred_nb = pipeline_nb.predict(X_test)
print(classification_report(y_test,y_pred_nb))

conf_matrix = confusion_matrix(y_test, y_pred_nb)
sb.heatmap(conf_matrix, annot=True, fmt="d", cbar=False)
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
# training data predict
y_pred_nb = pipeline_nb.predict(X_train)
print(classification_report(y_train,y_pred_nb))

conf_matrix = confusion_matrix(y_train, y_pred_nb)
sb.heatmap(conf_matrix, annot=True, fmt="d", cbar=False)
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

RANDOM FOREST CLASSIFICATION MODEL

```
# Label encoding
label_categorical = ['workclass', 'education', 'marital-status', 'occupation',
'relationship', 'race', 'sex', 'native-country']
X_categorical = pd.get_dummies(X[label_categorical], drop_first=True)
encoded_features = pd.concat([X.drop(label_categorical, axis=1), X_categorical],
axis=1)

# test train split
X_train, X_test, y_train, y_test = train_test_split(encoded_features, y,
test_size=0.2, random_state=42)

# random forest classifier
pipeline_rf = Pipeline([
    ('sampling', SMOTEENN(random_state=42)),  # Over- and undersampling
```

```
    ('classifier', RandomForestClassifier(random_state=42, n_estimators=100))  # Classifier
])

pipeline_rf.fit(X_train, y_train)
```

```
# test data
y_pred_rf = pipeline_rf.predict(X_test)
print(classification_report(y_test,y_pred_rf))

conf_matrix = confusion_matrix(y_test, y_pred_rf)
sb.heatmap(conf_matrix, annot=True, fmt="d", cbar=False)
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
# train data
y_pred_rf = pipeline_rf.predict(X_train)
print(classification_report(y_train,y_pred_rf))

conf_matrix = confusion_matrix(y_train, y_pred_rf)
sb.heatmap(conf_matrix, annot=True, fmt="d", cbar=False)
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

8. References

- www.youtube.com. (n.d.). Adult Sensus Income Kaggle Dataset Analysis | Kaggle | Who earns more than $50K/year ?? [online] Available at:. https://youtu.be/reVAGcwOxH8?si=aDL9SXw7NcoFXrmY

- www.youtube.com. (n.d.). Machine Learning for Everybody – Full Course. [online] Available at: https://www.youtube.com/watch?v=i_LwzRVP7bg&t=6309s&pp=ygULbWwgbGVhcm5pbmc%3D [Accessed 24 Mar. 2024].

- Rajavelu, S. (2022). Adult-Income-Analysis. [online] GitHub. Available at:
- https://github.com/saravrajavelu/Adult-Income-Analysis.