Machine Learning

(IT 6080)

Assignment 2 - Income Classification using Machine Learning Algorithms

Submitted to

Sri Lanka Institute of Information Technology

In partial fulfilment of the requirements

for the MSc in Information Technology

28th of October 2024

DECLARATION

I hereby declare that this assignment is my own original work and that all sources used are fully acknowledged. I have not previously submitted this work, in part or in full, for any academic credit at any institution. I have cited all materials used, and to the best of my knowledge, this work does not contain any uncredited content from other authors or sources.

X _____

Wijesekera D.T.S.
MS23464582

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

In today's age of fast-paced technological progress, machine learning (ML) has become crucial for automating tasks and discovering patterns within complex data. This project focuses on predicting income levels, a classification problem that reflects broader socio-economic dynamics, using machine learning techniques. The choice of this topic lies in its applicability to practice since the correct identification of the income level is useful in other aspects, including public policy, marketing, and sociological research. Hence, this research seeks to develop a demographic model that factors age, education, and occupation to determine if the income earned by an individual exceeds a given benchmark.

To achieve this, writer utilized supervised learning algorithms, specifically Support Vector Machine (SVM) and Random Forest, to classify income levels based on different socio-economic features. Writer chose these models for their distinct strengths: SVM's ability to handle high-dimensional spaces and create robust decision boundaries, and Random Forest's ensemble-based resilience against overfitting. The dataset was pre-processed well where missing values were addressed and categorical and numerical data were transformed, making it well-structured for training. By employing both statistical methods (like ANOVA and Chi-Square tests) and model evaluation metrics (such as precision-recall and ROC curves), writer carefully assessed feature importance and model performance to achieve a balance between accuracy and interpretability in income classification.

The use of machine learning for income classification not only provides valuable insights into the contributing socio-economic factors but also addresses real-world challenges in predictive modelling. By building a model to classify income levels, this project aims to support data-driven decision-making, which can inform economic policies, identify high-potential demographic groups for financial services, and help organizations tailor their strategies to diverse income brackets. Furthermore, this analysis sheds light on underlying patterns and correlations between income and demographic factors, offering a deeper understanding of social disparities. The careful selection of algorithms and preprocessing techniques ensures that the model is not only accurate but also interpretable, making the outcomes more accessible for stakeholders who may leverage these insights to design inclusive and equitable policies.

2. DATASET PREPROCESSING

2.1. Data Exploration

The data set used in this problem is the publicly available by the name of "Adult" at the Irvine ML Database. The database which is used contains both categorical and integer values. It consists of 15 features and 31650 instances. The dataset is widely used for predicting if income of an individual exceeds fifty thousand dollars or not, some of the factors which contributes to annual income are factors such as age, level of education, marital status, gender, occupation etc. It provides a real-world scenario for classification tasks. Hence, this is suitable for testing Machine Learning algorithms. It is as result of the 1994 Census database. It also contains some missing values which will be handled accordingly.

2.2. Data Preparation

The dataset to be fed into Machine Learning algorithms should be of high quality. Hence the need to clean and format the data frame in such a way that it will be easier to analyze. The preprocessing steps involved handling missing values, converting qualitative features into quantitation using label encoding methods. Further, normalizing the numerical features for consistency. These steps ensured the data to be in structured format. The pre-processing steps are carried out in Jupyter Notebook. Jupyter notebook is used because it allows you to visualize the output at every step. The same is shown in the report as well.

2.2.1. Data loading & Visualization

The dataset used for this project is publicly available at UC Irvine ML Repository [1]. It was downloaded and then saved to the local drive. Later it was transferred to the Jupyter Notebook for additional data processing.

```
file_path = r"C:\Users\Thisura\Desktop\sliit\ML\assignment 2\MS23464582\adult.data"
df = pd.read_csv(file_path)
```

Figure 1: Data path

The dataset was initially viewed to inspect the labels of the columns and their respective values. It was observed that column names were not properly labeled. As a result, appropriate names were assigned to the column to ensure clarity. Below figure 2 shows the dataset it was done using df.head() command.

Figure 2: Reading the dataset

Similarly df.shape command was used to see the total instances of the data set and total features. It showed that dataset used has 32,560 instances and 15 features. The traditional ML models work well on large datasets. Therefore, the dataset used has amble data to train the model and classify the income

Figure 3: Tuple representation of dataset

The columns of the dataset had remained using the python commands. The results of this renaming process are shown in figure 4 below.

```
[5]: #adding Column Names in the data set.

      names_of_coulmns = ['age', 'employment', 'final_weight', 'education', 'education_number',
                          'marital_status', 'profession', 'relationship', 'race', 'gender',
                          'capital-gain', 'capital-loss', 'week_hours', 'country', 'income']

      # Renaming the columns in the dataset
      df.columns = names_of_coulmns

      #Displaying the column names of dataset after assigning names.
      df.head()
```

[5]:

age	employment	final_weight	education	education_number	marital_status	profession	relationship	race	gender	capital-gain	capital-loss	week_hours	country	income
50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K
37	Private	284582	Masters	14	Married-civ-spouse	Exec-managerial	Wife	White	Female	0	0	40	United-States	<=50K

Figure 4: Renaming the columns

To get a clear idea, wanted to check the datatypes of the columns in the dataset. Therefore, dtypes command was used to check the dataset. It is observed that all the datasets have correct datatypes. Like no numerical column had object datatype and vice versa. Figure 5 displays the command output.

```
[53]: df.dtypes

[53]: age                 int64
      employment          object
      final_weight        int64
      education           object
      education_number    int64
      marital_status      object
      profession          object
      relationship        object
      race                object
      gender              object
      capital-gain        int64
      capital-loss        int64
      week_hours          int64
      country             object
      income              object
      dtype: object
```

Figure 5: Verifying data types of columns

2.2.2. Checking Missing Values from Dataset

Accuracy and performance of models are affected by missing values in the dataset. Thus, resulting in either incorrect prediction of biased patterns. Furthermore, missing values can cause failure in model training as well. Addressing missing values is an essential step in the pre-processing phase. Figure 6 shows the total count of missing values in the dataset.

4

```
•[7]:  #making '?' consistent removing spaces in the column
       df = df.apply(lambda x: x.str.strip() if x.dtype == "object" else x)

       #identifying Missing Values in columns
       df.replace("?", np.nan, inplace=True)

       missing_data = df.isnull().sum()

       print(missing_data[missing_data > 0])

       employment    1836
       profession    1843
       country        583
       dtype: int64
```

Figure 6: Inspecting for missing values

```
[57]:  plt.figure(figsize=(10, 6))
       missing_data.plot(kind='bar', color='blue')

       plt.title('Missing Values in Each Column')
       plt.xlabel('Columns')
       plt.ylabel('Number of Missing Values')

       plt.show()
```

Figure 7: Representing the missing values

Missing values from the dataset were identified and inspected. It was observed that the following three columns having names 1) "Employment", 2) "Profession" and 3) "Country" contained missing values. Their respective numbers are shown in figure 8.
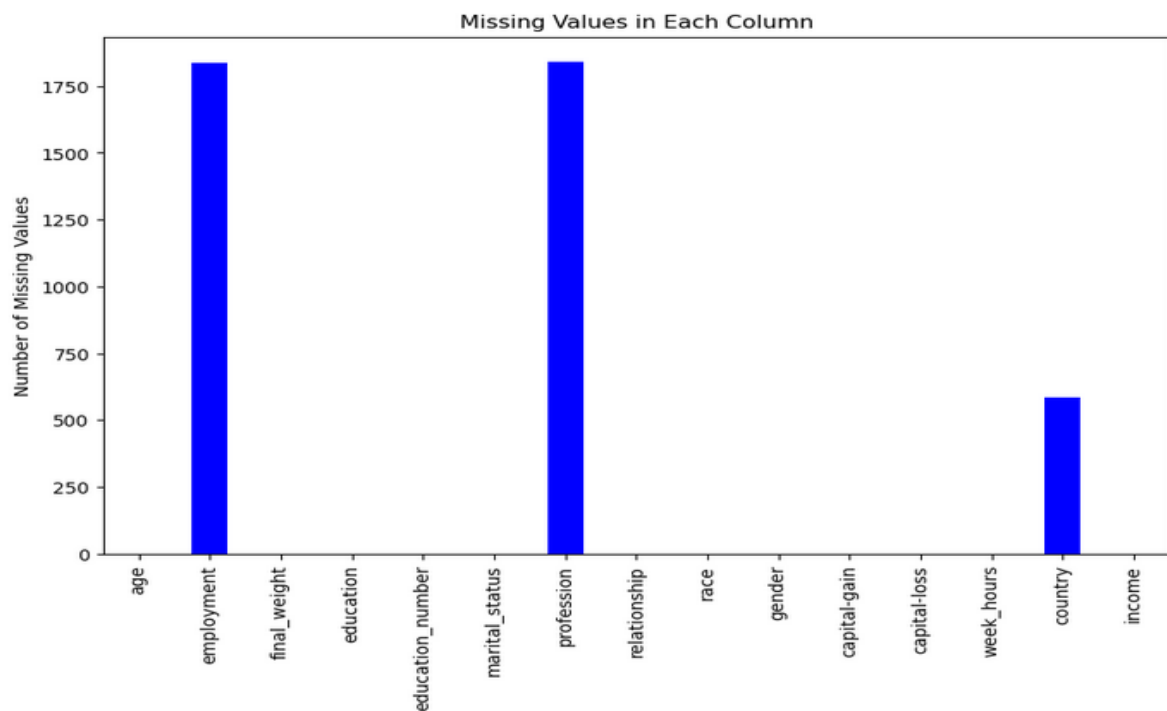
Figure 8: Representation of the missing values

Missing values for two columns that are "profession" and "employment" were changed with the mode of respective columns to ensure the consistency of the dataset. This helped retain the most frequent category for missing values.

```
[11]: df.replace("?", np.nan, inplace=True)

      # Replaceing the  missing values in 'employment' and 'profession' columns with their mode
      df['employment'].fillna(df['employment'].mode()[0], inplace=True)
      df['profession'].fillna(df['profession'].mode()[0], inplace=True)

      #Missing values are filled
      print(df[['employment', 'profession']].isnull().sum())
      print('Missing values are replaced by Mode of the columns of employement & profession')

      employment    0
      profession    0
      dtype: int64
      Missing values are replaced by Mode of the columns of employement & profession
```

Figure 9: Replacing the missing values with mode

Rows with missing values in the "country" column were dropped, as assigning nationalities can be sensitive and may introduce bias. This was done to ensure that the integrity of data frame remains intact. The sample dropped rows are shown in figure 10.

```
[13]: missing_country_rows = df[df['country'].isnull()]
      print(missing_country_rows)

      #Droping rows
      df.dropna(subset=['country'], inplace=True)

      print('Values with missing rows are deleted')
            age        employment  final_weight       education  education_number  \
      13     40           Private        121772        Assoc-voc                11
      37     31           Private         84154     Some-college                10
      50     18           Private        226956          HS-grad                 9
      60     32           Private        293936          7th-8th                 4
      92     30           Private        117747          HS-grad                 9
      ...    ...              ...           ...              ...               ...
      32448  44      Self-emp-inc         71556          Masters                14
      32468  58      Self-emp-inc        181974        Doctorate                16
      32491  42  Self-emp-not-inc        217597          HS-grad                 9
      32509  39           Private        107302          HS-grad                 9
      32524  81           Private        120478        Assoc-voc                11
```

Figure 10: Dropping the missing values

2.2.3. Verifying the unique values of categorical columns.

Values in categorical columns can sometimes have similar meanings but represented indifferent ways, this can cause inconsistencies in the dataset. To overcome this challenge, unique values from columns were identified. This step helped in reducing redundancy and ambiguity. Standardized values in categorical values can make dataset cleaner and reliable for further analysis.

6

```
[128]: #Checking unqiue values in the column to standarize it
       cat_columns = ['employment', 'education', 'marital_status', 'profession',
                      'relationship', 'race', 'gender', 'country', 'income']

       # Looping through each categorical column and print unique values
       for column in cat_columns:
           values = df[column].unique()
           print(f"Distinct entries in '{column}': {values}\n")
```

Figure 11: Identifying distinct entries in categorical columns

It is observed from seeing the output of the data that no spelling mistakes or variations in naming for the same values within the column of the dataset.

```
Distinct entries in 'employment': ['Self-emp-not-inc' 'Private' 'State-gov' 'Federal-gov' 'Local-gov'
 'Self-emp-inc' 'Without-pay' 'Never-worked']

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

Distinct entries in 'marital_status': ['Married-civ-spouse' 'Divorced' 'Married-spouse-absent' 'Never-married'
 'Separated' 'Married-AF-spouse' 'Widowed']

Distinct entries in 'profession': ['Exec-managerial' 'Handlers-cleaners' 'Prof-specialty' 'Other-service'
 'Adm-clerical' 'Sales' 'Transport-moving' 'Farming-fishing'
 'Machine-op-inspct' 'Tech-support' 'Craft-repair' 'Protective-serv'
 'Armed-Forces' 'Priv-house-serv']

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

Distinct entries in 'race': ['White' 'Black' 'Asian-Pac-Islander' 'Amer-Indian-Eskimo' 'Other']

Distinct entries in 'gender': ['Male' 'Female']

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

Distinct entries in 'income': ['<=50K' '>50K']
```

Figure 12: Identifying unique entries

2.2.4. Summary stats of Numerical Columns

Summary stats of numerical columns were visualized using describe command. It provides the count, average, standard deviation, minimum, maximum, and quartile range for each column. It Is observed from the stats that there is no outlier present in the dataset. However, further analysis must be done too to confirm it. Box Plot and Histogram were used to visualize the numerical columns for better understanding and to detect the outliers.

[29]:

	age	final_weight	education_number	capital-gain	capital-loss	week_hours
count	32560.000000	3.256000e+04	32560.000000	32560.000000	32560.000000	32560.000000
mean	38.581634	1.897818e+05	10.080590	1077.615172	87.306511	40.437469
std	13.640642	1.055498e+05	2.572709	7385.402999	402.966116	12.347618
min	17.000000	1.228500e+04	1.000000	0.000000	0.000000	1.000000
25%	28.000000	1.178315e+05	9.000000	0.000000	0.000000	40.000000
50%	37.000000	1.783630e+05	10.000000	0.000000	0.000000	40.000000
75%	48.000000	2.370545e+05	12.000000	0.000000	0.000000	45.000000
max	90.000000	1.484705e+06	16.000000	99999.000000	4356.000000	99.000000

Figure 13: Statistical summary of numerical columns

2.2.5. Addressing outliers from the numerical columns

Outliers can cause skewness in the data distribution. This may result in inaccurate predictions and impact the overall performance of the models. To identify outliers, visualization techniques were used. The Box-Wishker plot provides a representation of data's spread and highlights the points that can fall outside the normal range whereas, histogram offers insights into the frequency distribution of the data values. By employing these techniques, we can understand the presence of outliers. Functions were created to avoid reptation of code. Figure 14 shows the functions of Box plot and histogram.

```python
[18]: def plot_box(data, column_name):
          plt.figure(figsize=(10, 5))
          sns.boxplot(y=data[column_name])
          plt.ylabel(column_name)
          plt.title(f'{column_name} - Box Plot')
          plt.show()
```

```python
[20]: def plot_histogram(data, column_name):
          plt.figure(figsize=(10, 5))
          sns.histplot(data[column_name], bins=20, kde=True)
          plt.xlabel(column_name)
          plt.ylabel('Count')
          plt.title(f'{column_name} - Histogram')
          plt.show()
```

Figure 14: Functions of Box plot and histogram

2.2.6. Visualization of Numerical Columns

- **Age**

The visualization of the "age" column indicates that there are no outliers in the dataset. The maximum value for age is 90, which is normal. Furthermore, all datasets lie inside the normal age range. This shows that age columns are suitable for further analysis.

```
[132]: #checking outliers in the Age Column
       plot_box(df, 'age')
       plot_histogram(df, 'age')
```

Figure 15: Age boxplot and histogram functions.
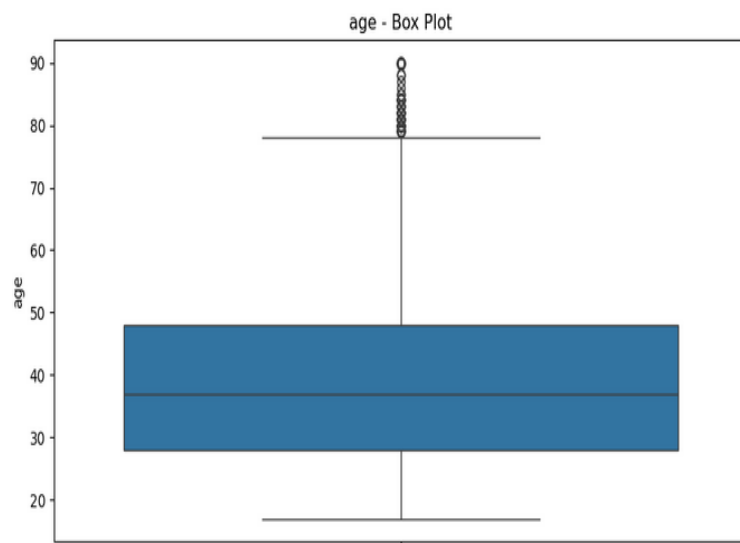
Figure 16: Age box plot
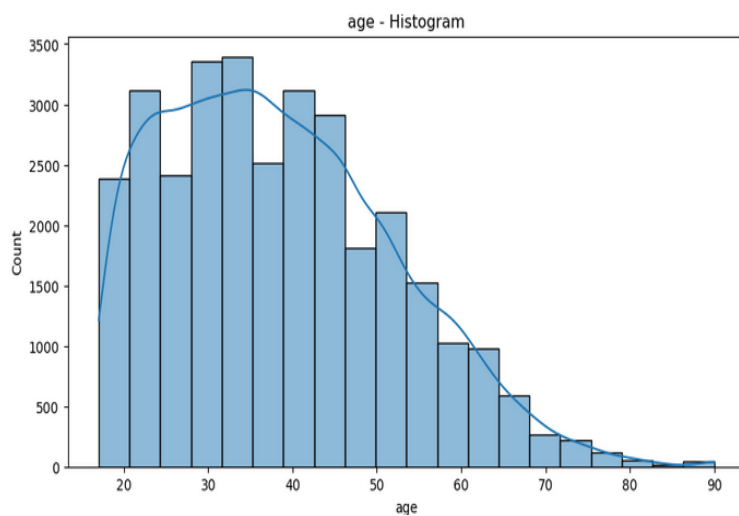
Figure 17: Age Histogram

- **Education_Number**

Similarly, visualization of Education-num reveals that all data points are inside the normal range. The max education number is 16. That means there is no outlier present in the education number column. Therefore, the education num data is ready for further processing.

```
[133]: #checking outliers in the Education Numbers
       plot_box(df, 'education_number')
       plot_histogram(df, 'education_number')
```

Figure 18: Education number boxplot and histogram functions
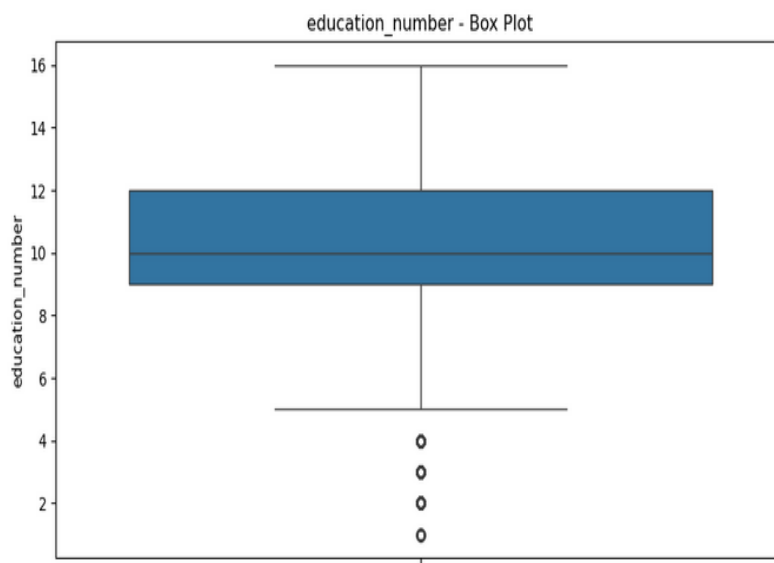
Figure 19: Education number boxplot
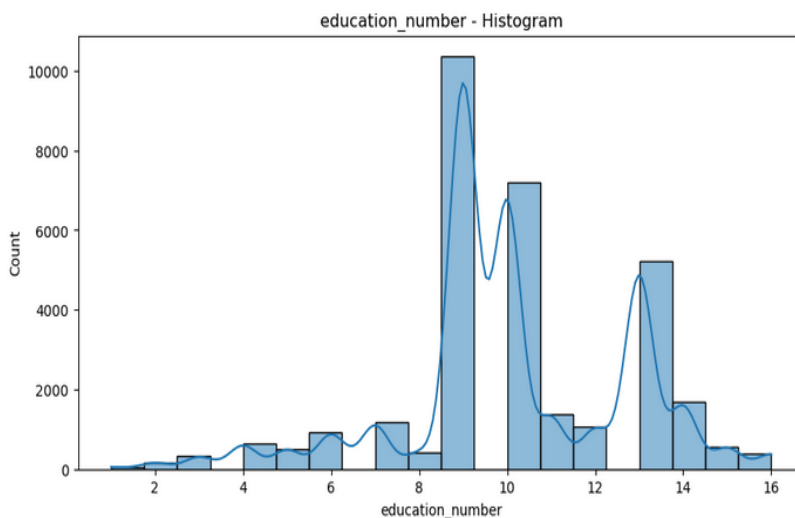
Figure 20: Education number histogram

- **Capital – Gain**

The visualization of the "Capital-Gain" column shows that while one data point has a significant high value, whereas most of the data falls inside a normal range. Given the inherent nature of capital gains, this high value cannot be assumed as an outlier.

```
[134]:  #checking outliers in the capital Gain
        plot_box(df, 'capital-gain')
        plot_histogram(df, 'capital-gain')
```

Figure 21: Capital gain boxplot and histogram functions
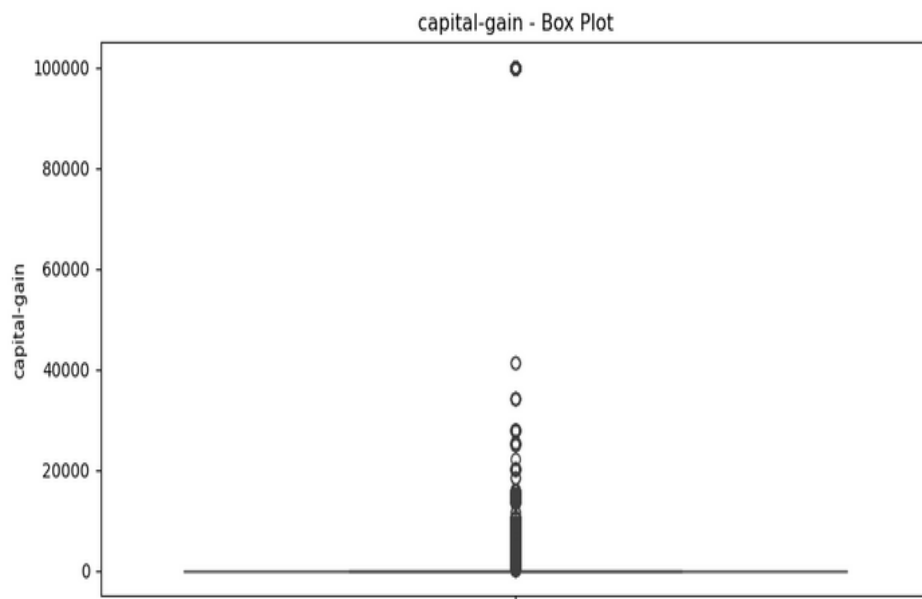
Figure 22: Capital gain boxplot
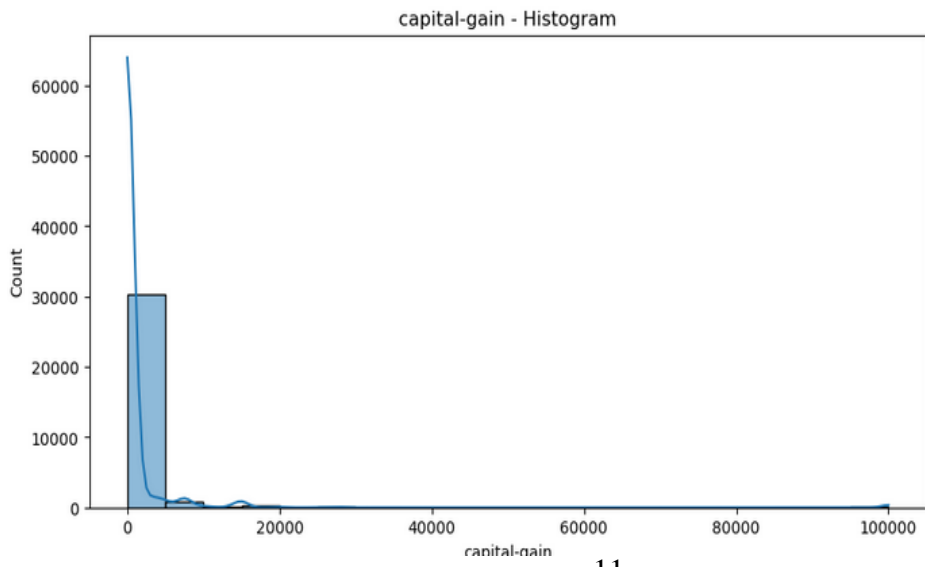
Figure 23: Capital gain histogram

- **Capital – Loss**

Similarly, capital loss column shows no presence of outlier values, indicating that all data points fall within normal range. Hence, further analysis can be done.

```
[135]:  #checking outliers in the Capital Loss
        plot_box(df, 'capital-loss')
        plot_histogram(df, 'capital-loss')
```

Figure 24: Capital loss boxplot and histogram functions
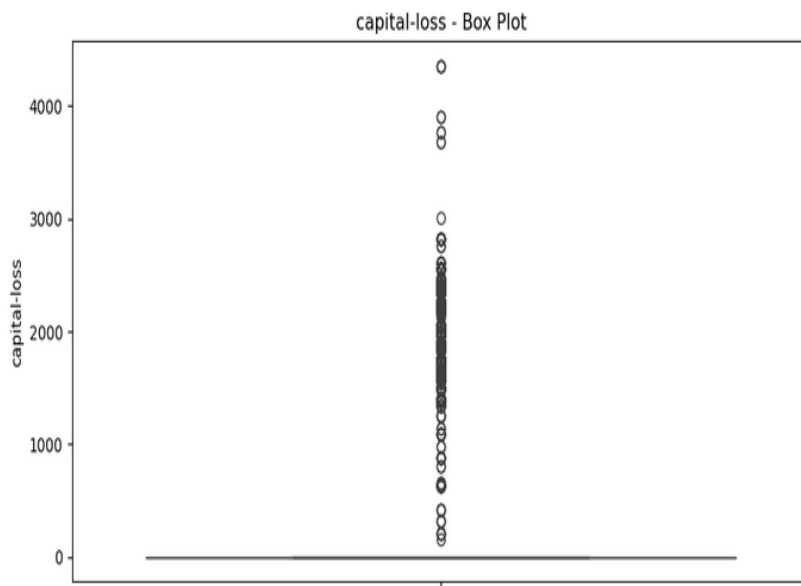
Figure 25: Capital loss boxplot
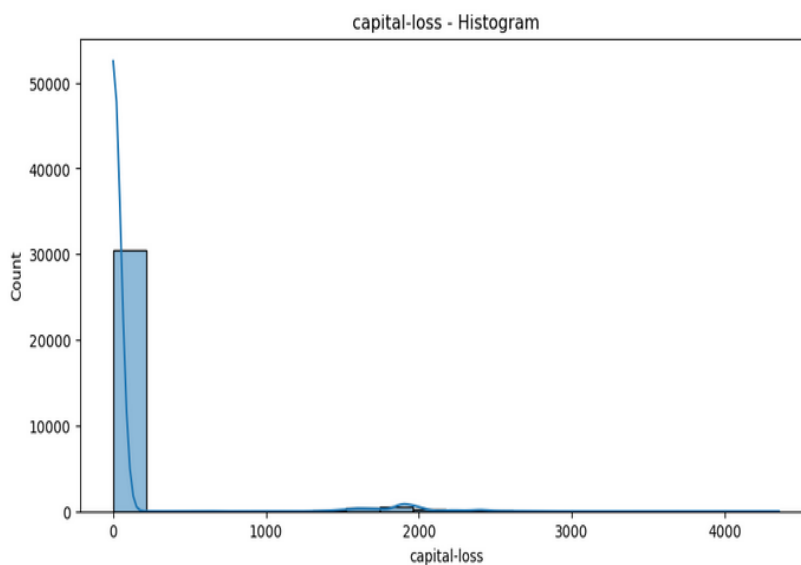
Figure 26: Capital gain histogram

- **Week_Hours**

Similarly, the "Week_Hours" column follows a normal distribution, with most individuals working within a typical range. However, there are few individuals who have high values, indicating that they are working more than 70 hours per week. While these values are notable, they do not significantly disrupt the overall distribution.

```
[136]: #checking outliers in the Weekly Hours
       plot_box(df, 'week_hours')
       plot_histogram(df, 'week_hours')
```

Figure 27: Week hours boxplot and histogram functions
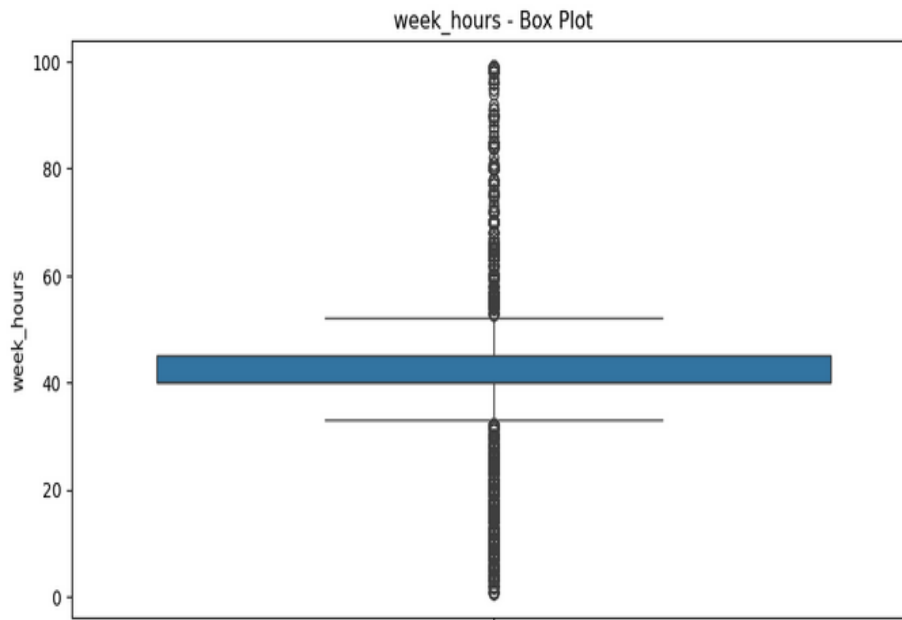
Figure 28: Week hours boxplot
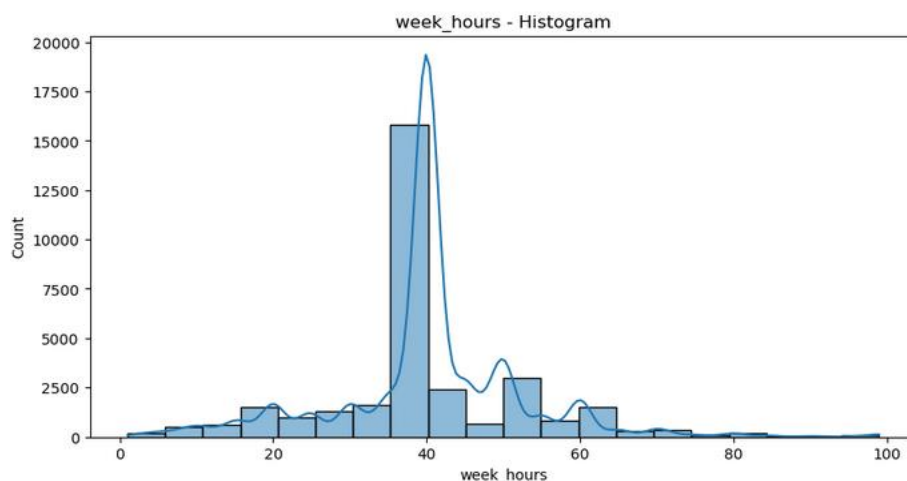
Figure 29: Week hours histogram

3. FEATURE CORRELATION WITH TARGET VARIABLE

To determine if the means of numerical features across several categories of a categorical variable differ significantly, the ANOVA (Analysis of Variance) test is frequently used. This can provide information about the variables that may affect income classification in the adult dataset by assisting in determining if a numerical feature and a categorical target variable, such as income, are related. Similarly, the Chi-Square test is used for categorical features. Both tests were used in datasets to find the correlation and the results are shown below.

3.1. ANOVA F-Statistic for Numerical Features

The Analysis of Variance (ANOVA) F-statistics is a widely used metric in statistical analysis, particularly in experimental and observational studies, to determine whether there are significant differences among group means. Introduced by Ronald A. Fisher, who established its mathematical foundation in the early 20th century, ANOVA serves as a statistical technique to assess whether there is an association between numerical features and a categorical variable [2]. ANOVA evaluates whether the differences in means are statistically significant or simply due to random variation by comparing the variance within groups to the variance between groups. In our case, the numerical features include age, final_weight, education_number, Capital-Gain, Capital-Loss, and week_hours, while the target variable is income, which is stored as object datatype, making it a categorical variable. The code and results are displayed in Figure 30 below.

```
[129]: anova_features = ['age', 'final_weight', 'education_number', 'capital-gain', 'capital-loss', 'week_hours']
       anova_results = {}

       for feature in anova_features:
           groups = [df[feature][df['income'] == category] for category in df['income'].unique()]
           anova_results[feature], p_value = stats.f_oneway(*groups)
           print(f"ANOVA result for {feature}: F-statistic = {anova_results[feature]}, P-value = {p_value}")

       ANOVA result for age: F-statistic = 1845.590672152451, P-value = 0.0
       ANOVA result for final_weight: F-statistic = 2.60941332046777763, P-value = 0.1062409224161227
       ANOVA result for education_number: F-statistic = 4066.8527089574845, P-value = 0.0
       ANOVA result for capital-gain: F-statistic = 1668.6628962523523, P-value = 0.0
       ANOVA result for capital-loss: F-statistic = 729.4876778136354, P-value = 7.054070014175733e-159
       ANOVA result for week_hours: F-statistic = 1793.3189217281883, P-value = 0.0
```

Figure 30: ANOVA analysis

```
[133]: plt.figure(figsize=(10, 6))

       # ANOVA bar plot
       plt.subplot(1, 2, 1)
       plt.bar(anova_results.keys(), anova_results.values(), color='skyblue')
       plt.title('ANOVA F-statistic for Numerical Features')
       plt.xlabel('Features')
       plt.ylabel('F-statistic')
       plt.xticks(rotation=45)
```

Figure 31: ANOVA F-statistic bar plot function
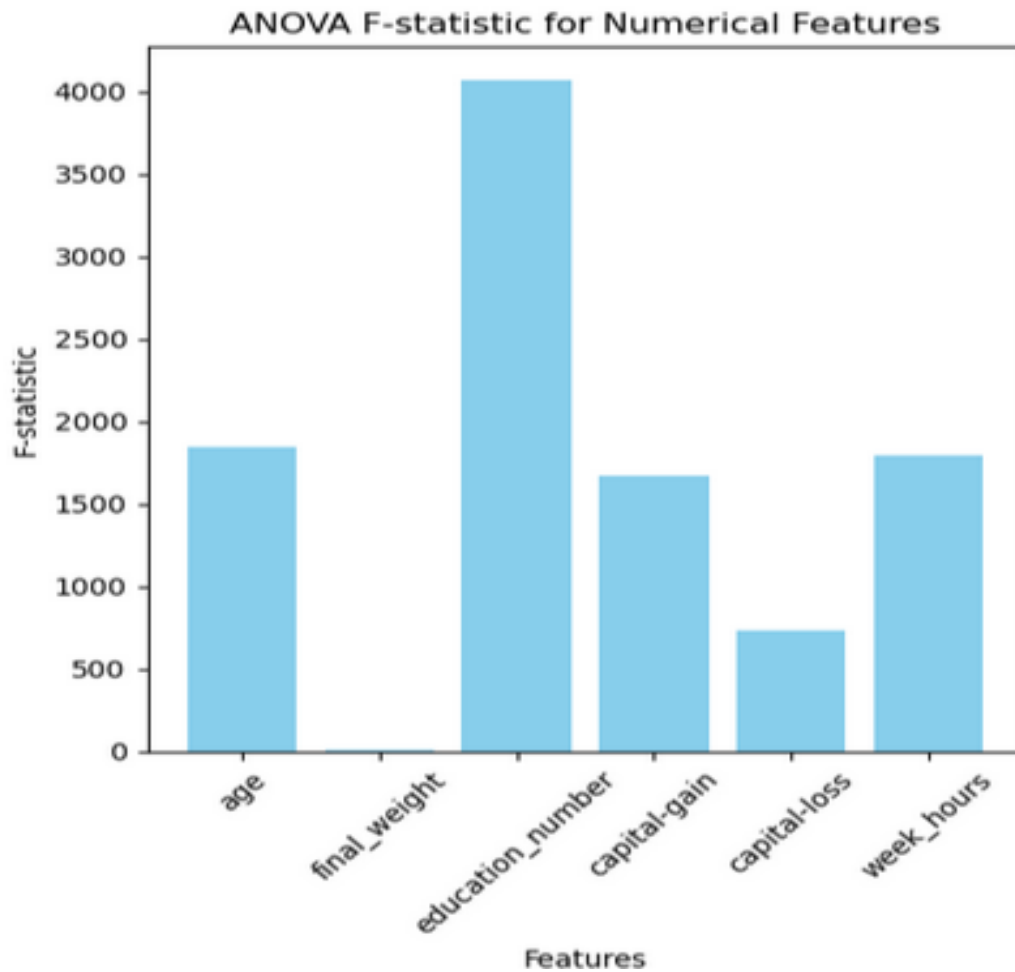
Figure 32: ANOVA F-statistic bar plot

3.1.1. Results Interpretation of ANOVA F-Statistic

The High F value indicates that there is a significant correlation with the target variable. In our case the education_number has the highest value, meaning that it has the strongest relationship with income among numerical features. The higher the education_number will be the more

likely it is that income of that person is greater than 50k. Similarly, age, capital gain, and weekly hours also have relatively high F-statistics, showing they may have a meaningful impact on income classification. Whereas, final weight show lower F-statistics, suggesting a weaker relationship with income.

3.2. Chi-Square P-Values for Categorical Features

The Chi-Square test was first introduced by the statistician Karl Pearson in 1900. The Chi-Square test and its associated p-values are important tools in statistics, primarily used to evaluate the independence of categorical features and test associations between them. A low p-value (typically < 0.05) indicates a strong association [3]. High P-Value indicates that the feature does not have a significant relationship with income. In our case employment, education, marital_status, profession, relationship, race, gender, country are categorical variables.

```
[183]:  categorical_features = ['employment', 'education', 'marital_status', 'profession', 'relationship', 'race', 'gender', 'country']
        chi2_results = {}
        chi2_p_values = {}

        for feature in categorical_features:
            # Applying Label Encoding for categorical data
            df[feature] = LabelEncoder().fit_transform(df[feature])

            # Chi-Square test
            chi2_stat, p_val = chi2(pd.get_dummies(df[feature]), df['income'])  # Use pd.get_dummies for correct shape
            chi2_results[feature] = chi2_stat[0]  # Getting the first value
            chi2_p_values[feature] = p_val[0]      # Getting the first value

            print(f"Chi-Square result for {feature}: Chi2-statistic = {chi2_stat[0]}, P-value = {p_val[0]}")
```

```
Chi-Square result for employment: Chi2-statistic = 110.63688126164794, P-value = 7.106506583092901e-26
Chi-Square result for education: Chi2-statistic = 153.3139021849762, P-value = 3.270924275311394e-35
Chi-Square result for marital_status: Chi2-statistic = 447.4312694149012, P-value = 2.613039359753383e-99
Chi-Square result for profession: Chi2-statistic = 232.0757852491291, P-value = 2.1021179659139007e-52
Chi-Square result for relationship: Chi2-statistic = 3059.501804805991, P-value = 0.0
Chi-Square result for race: Chi2-statistic = 26.544273357184906, P-value = 2.575670851397756e-07
Chi-Square result for gender: Chi2-statistic = 997.7753872236262, P-value = 5.467855961636479e-219
Chi-Square result for country: Chi2-statistic = 1.6976855283749277, P-value = 0.19259093787958279
```

Figure 33: Chi-Square test
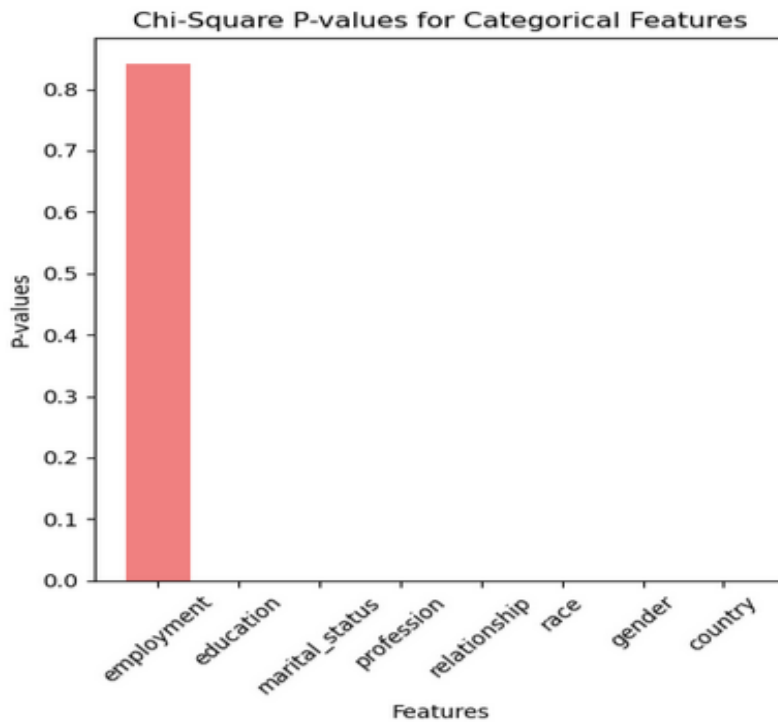
Figure 34: Chi-Square P-values for Categorical Features

3.2.1. Results Interpretation of Chi-Square

The chi-square test results show how each categorical feature relates to target variable.

Most of the categorical variables (education, marital status, profession, relationship, race, gender, and country) have very low p-values, indicating they are significantly associated with the dependent variable and may contribute to predicting it. Only employment has a high p-value, suggesting it does not have a significant association with the outcome and may not be a key predictor in your model.

3.2.2. Feature Selection based on the results

Choosing the right feature is crucial to good accuracy and predictions. Therefore, based on the test results of ANOVA and Chi – Square we would drop some feature that doesn't corresponds to the outcome results. These features are 1) final_weight 2) employment

- **Dropping final_weight Column**

```
#dropping of final_weight column
df.drop('final_weight', axis=1, inplace=True)
print('The final_weight column is deleted')

The final_weight column is deleted
```

Figure 35: Dropping the 'final_weight' column from the Data set

```
]: #data set columns after dropping final_weight column
print(df.columns)

Index(['age', 'employment', 'education', 'education_number', 'marital_status',
       'profession', 'relationship', 'race', 'gender', 'capital-gain',
       'capital-loss', 'week_hours', 'country', 'income'],
      dtype='object')
```

Figure 36: Verifying the structure of the data set after removing 'final_weight' column

- **Dropping Column of employment**

```
#dropping of employment  column
df.drop('employment', axis=1, inplace=True)
print('The employment  column is deleted')

The employment  column is deleted
```

Figure 37: Dropping the 'employment' column from the data set

```
#data set columns after dropping employment column
print(df.columns)

Index(['age', 'education', 'education_number', 'marital_status', 'profession',
       'relationship', 'race', 'gender', 'capital-gain', 'capital-loss',
       'week_hours', 'country', 'income'],
      dtype='object')
```

Figure 38: Verifying the structure of the data set after removing 'employment' column

4. DATASET PREPROCESSING (CONT...)

4.1. Income Mapping

The "income" column contained string values "<50k" and ">50", which needed to be converted for Boolean for processing as our model task is classification. Therefore, it is converted into Boolean. To achieve this, these values were mapped to 0 and 1. The results of this are mapped in 0 and 1 are shown in figure 41.

```
[138]: income_mapping = {"<=50K": 0, ">50K": 1}

       #Here income is being transformed into 0 and 1 value
       df['income'] = df['income'].map(income_mapping)

       income_counts = df['income'].value_counts()

       print(income_counts)
       print(df['income'].unique())

       print('income is mapped to 0 & 1')

       income
       0    24282
       1     7695
       Name: count, dtype: int64
       [0 1]
       income is mapped to 0 & 1
```

Figure 39: Mapping 'income' column to binary values for classification

The figure 41 below shows that income is mapped to 0 and 1 respectively.

```
[419]: df.head()
```

	age	employment	final_weight	education	education_number	marital_status	profession	relationship	race	gender	week_hours	income
0	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	13	0
1	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	40	0
2	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	40	0
3	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	40	0
4	37	Private	284582	Masters	14	Married-civ-spouse	Exec-managerial	Wife	White	Female	40	0

Figure 40: DataFrame after preprocessing

4.2. Label Encoding

Before feeding into algorithms, categorical columns must be converted into numerical columns. To accomplish this, label encoding methods were employed to transform categorical

19

columns into quantitative values. The process facilitates the effective processing of data by machine learning algorithms. Firstly, the categorical columns were identified and after that these columns were converted into numerical columns. The results of the label encoding are shown in figure 41.

```
[139]: label_encoders = {}
       for col in cat_columns:
           # Initializing the encoder for each column
           lbl_encoder = LabelEncoder()

           # Transforming the categorical column with label encoding
           df[col] = lbl_encoder.fit_transform(df[col])

           # Storeingthe encoder object for later use
           label_encoders[col] = lbl_encoder
```

```
[140]: df.head()
```

[140]:

	age	employment	final_weight	education	education_number	marital_status	profession	relationship	race	gender	capital-gain	capital-loss	week_hours	country	income
0	50	5	83311	9	13	2	3	0	4	1	0	0	13	38	0
1	38	3	215646	11	9	0	5	1	4	1	0	0	40	38	0
2	53	3	234721	1	7	2	5	0	2	1	0	0	40	38	0
3	28	3	338409	9	13	2	9	5	2	0	0	0	40	4	0
4	37	3	284582	12	14	2	3	5	4	0	0	0	40	38	0

Figure 41: Applying label encoding to categorical columns

4.3. Standardization of Values

Numerical values can exhibit a large spread or distribution. Due to this, potential challenges can arise during the training of algorithms. This spread can complicate the convergence of algorithms. By standardizing the features, we can stabilize the learning process, allowing algorithms to converge more efficiently. Therefore, standardization of numerical columns was introduced in the preprocessing. Post standardization results are shown in figure 42.

```
[443]: #transforming these columns to standard format
       numerical_cols = ['age', 'final_weight', 'education_number','week_hours']

       scaler = StandardScaler()

       df[numerical_cols] = scaler.fit_transform(df[numerical_cols])
       print(df.head())

               age  employment  final_weight  education  education_number  \
       0  0.835962           5     -1.006460          9          1.143809
       1 -0.042381           3      0.245246         11         -0.418316
       2  1.055548           3      0.425670          1         -1.199378
       3 -0.774334           3      1.406415          9          1.143809
       4 -0.115576           3      0.897286         12          1.534341

          marital_status  profession  relationship  race  gender  week_hours  income
       0               2           3             0     4       1   -2.220918     NaN
       1               0           5             1     4       1   -0.033848     NaN
       2               2           5             0     2       1   -0.033848     NaN
       3               2           9             5     2       0   -0.033848     NaN
       4               2           3             5     4       0   -0.033848     NaN
```

Figure 42: Standardizing numerical columns in the DataFrame

4.4. Target Value Dropping

The target variable, "income" was dropped from the original dataset and stored in separate variable to facilitate the model training process. It is done so to ensure that the features (X) used training do not include the target variable (y), allowing the algorithm to get trained and predict afterwards the output.

```
[26]: X = df.drop('income', axis=1)
      y = df['income']
```

Figure 43: Separating features and target variable for model training

```
[431]: X.head()
```

	age	employment	final_weight	education	education_number	marital_status	profession	relationship	race	gender	week_hours
0	50	5	83311	9	13	2	3	0	4	1	13
1	38	3	215646	11	9	0	5	1	4	1	40
2	53	3	234721	1	7	2	5	0	2	1	40
3	28	3	338409	9	13	2	9	5	2	0	40
4	37	3	284582	12	14	2	3	5	4	0	40

Figure 44: Data set after target variable is dropped

4.5. Data Splitting

The data was split in an 80:20 ratio, with 80% used for training and 20% set aside to test the model's performance. The 20% reserved for testing was not shown to the model during training, allowing it to assess the model's accuracy and generalization on unseen data.

```
[28]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Figure 45: Data splitting

```
[728]: X_train.shape
[728]: (25581, 11)

[730]: X_test.shape
[730]: (6396, 11)

[732]: y_train.shape
[732]: (25581,)

[736]: y_test.shape
[736]: (6396,)
```

Figure 46: Size of data after splitting

5. MACHINE LEARNING MODEL AND RESULTS

5.1. Supervised Learning

Supervised learning is ideal for tasks where labelled data is available, and the objective is to predict outcomes based on known features. In this project, where the goal is to classify individuals' income levels based on demographic information, supervised learning is effective because it allows models to learn relationships between input variables and the target income classification. Supervised models, like Support Vector Machine (SVM) and Random Forest, are particularly suitable for structured data with categorical outcomes, enabling them to identify patterns and distinguish classes in the dataset

Using supervised learning in income classification also allows for precise model evaluation on metrics like accuracy and recall, providing measurable insights into model performance [4]. This method's structured approach to learning and evaluation helps refine the model iteratively, ensuring better generalization to new data and supporting accurate decision boundaries for complex socio-economic features. Supervised learning's focus on labeled datasets makes it a robust choice for tasks requiring reliable prediction and interpretability [5].

5.1.1. Model Selection

Dataset become of high quality after thoroughly pre-processing. With the data organized and cleaned, it is ready for training. The next step was model selections and training algorithms on datasets. Two models of Machine learning were selected: 1) Support Vector Machine (Known as SVM) and 2) Random Forest were selected to take advantage of their respective strengths.

5.1.2. Support Vector Machine

SVM, developed by Vladimir Vapnik and Corinna Cortes at AT&T Bell Labs in the 1990s, is recognized for its capability to manage complex, high-dimensional datasets and to establish

decision boundaries (or hyperplanes) that separate classes with maximum margin. This makes it a favored option for binary classification tasks [6]. The working mechanism of this is, it finds the optimal hyperplane that differentiates classes of the problem. SVM is used because it can handle highly dimensional datasets easily and it can create decision-making boundaries [7]. SVM is effective in cases where the classes are not linearly separable. It is also robust against overfitting, especially where data is with many features. The model is trained on the training dataset, as shown in Figure 47.
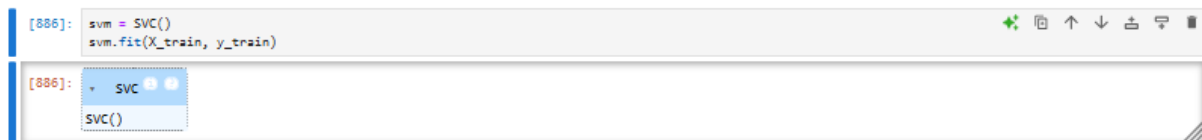
```
[886]: svm = SVC()
       svm.fit(X_train, y_train)

[886]:   ▾ SVC
         SVC()
```

Figure 47: Support Vector Classifier (SVC) on the training dataset

The accuracy of the model is checked on test dataset. The accuracy of model on test dataset is 81.83%.

```
# Predictions and accuracy
y_pred_svm = svm.predict(X_test)
print("SVM Accuracy on Test Set:", accuracy_score(y_test, y_pred_svm))

SVM Accuracy on Test Set: 0.8183239524702939
```

Figure 48: Accuracy of the Support Vector Machine (SVM) model

Precision indicates the accuracy of positive predictions. The model's precision on the test set is 0.8154.

```
precision = precision_score(y_test, y_pred_svm)
print(f"Precision: {precision:.4f}")

Precision: 0.8154
```
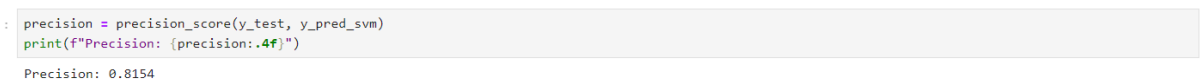
Figure 49: Precision of the Support Vector Machine (SVM) model

The F1-Score reflects the harmonic mean of precision and recall. For this dataset, the SVM model achieved an F1-Score of 0.4035.
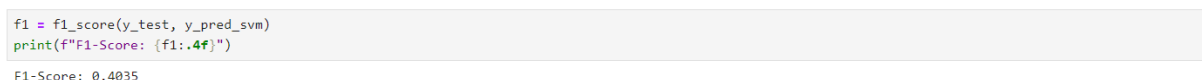
```
f1 = f1_score(y_test, y_pred_svm)
print(f"F1-Score: {f1:.4f}")

F1-Score: 0.4035
```

Figure 50: F1 score of the Support Vector Machine (SVM) model

24

The ROC-AUC score indicates the model's capability to differentiate between two classes. For this model, the ROC-AUC score is 0.6950.

```
auc = roc_auc_score(y_test, y_pred_svm)
print(f"AUC-ROC: {auc:.4f}")

AUC-ROC: 0.6250
```

Figure 51: ROC-AUC of the Support Vector Machine (SVM) model

The Precision-Recall Curve highlights how effectively the model manages the positive class by illustrating the balance between precision and recall. It is particularly valuable for imbalanced datasets where the positive class is uncommon.

```
[908]: precision, recall, thresholds = precision_recall_curve(y_test, y_pred_svm)

plt.figure(figsize=(8, 6))
plt.plot(recall, precision, label="Precision-Recall Curve", color="green")
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Precision-Recall Curve")
plt.legend(loc="best")
plt.show()
```

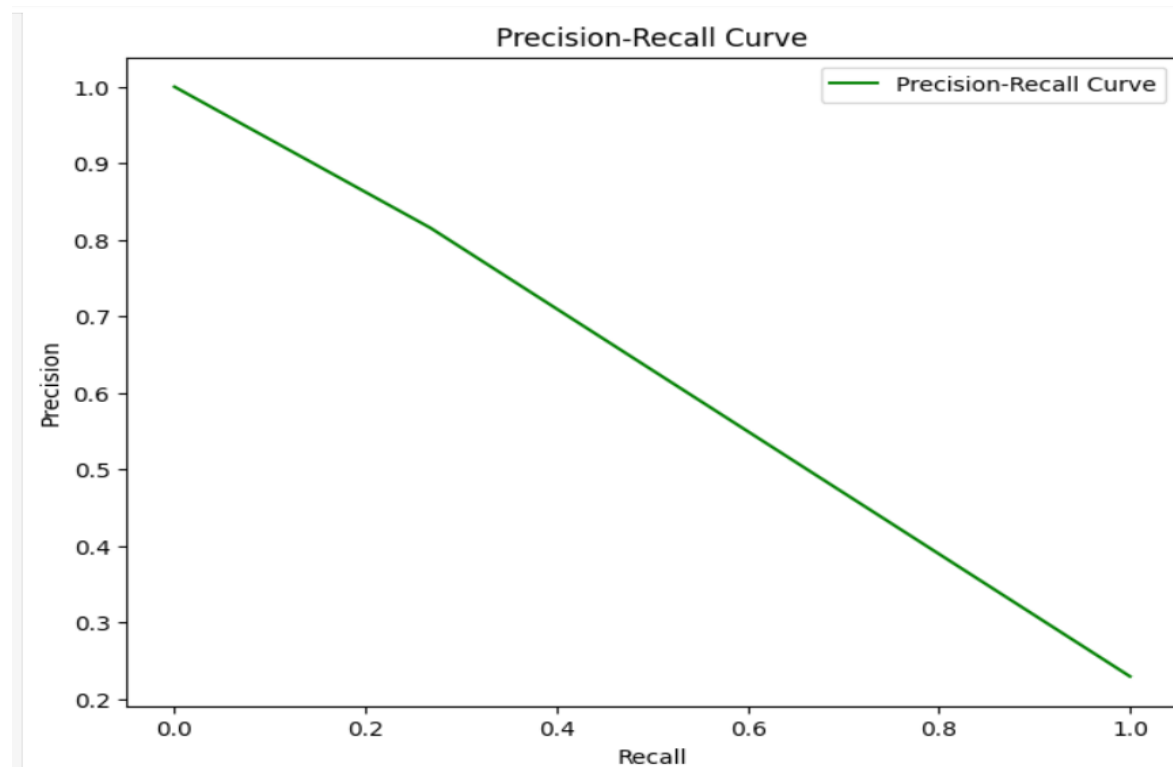*Figure 52: Precision-Recall Curve function of Support Vector Machine (SVM) model*

Figure 53: Precision-Recall Curve of Support Vector Machine (SVM)

25

The labelled confusion matrix visually demonstrates the performance of the SVM model, providing an understanding of how accurately the model predicted each class. It also offers insight into the types of errors, such as false positives and false negatives, made by the model.

```python
conf_matrix = confusion_matrix(y_test, y_pred_svm)
print("Confusion Matrix:\n", conf_matrix)

# Defining class labels
class_labels = ['True Negative Class (TN)', 'False Positive Class (FP)', 'False Negative Class (FN)', 'True Positive Class (TP)']

# Formatting group counts and percentages
count_values = [f"{int(val)}" for val in conf_matrix.ravel()]
percentage_values = [f"{val:.2%}" for val in conf_matrix.ravel() / np.sum(conf_matrix)]

# Creating labels combining names, counts, and percentages
combined_labels = [f"{label}\n{count}\n{percent}" for label, count, percent in zip(class_labels, count_values, percentage_values)]
combined_labels = np.array(combined_labels).reshape(2, 2)

# Plotting the confusion matrix with annotations
plt.figure(figsize=(9, 7))
sns.heatmap(conf_matrix, annot=combined_labels, fmt='', cmap="Blues",
            xticklabels=['<=50K', '>50K'], yticklabels=['<=50K', '>50K'])

plt.ylabel('Actual')
plt.xlabel('Predicted by Model')
plt.title('SVM Confusion Matrix with Labels')
plt.show()

Confusion Matrix:
 [[4841   89]
 [1073  393]]
```

Figure 54: Confusion matrix function of Support Vector Machine (SVM) model

The SVM Model achieved accuracy of 81.54% on test data set. It means that it correctly classified around 82% of the values. Model was also evaluated on the base of confusion matrix.

- True Positive (TP): 393 instances were predicted positive. It means that 658 individuals were earning more than $50k in test dataset.

- True Negatives (TN): 4841 instances were correctly predicted as negative. It means that 4642 individuals were earning less than $50K in dataset.

- False Positive (FP): 89 instances were wrongly classified. They belong to negative class, however model classified as positive.

- False Negative (FN): 1073 instances were misclassified. They belong to a positive class; however, models classify them as positive.

These results indicate that the model's overall performance is satisfactory. However, there are many false negatives, indicating that it struggled to predict some individuals with income above 50k. The SVM model shows a balanced trade-off between precision and recall.
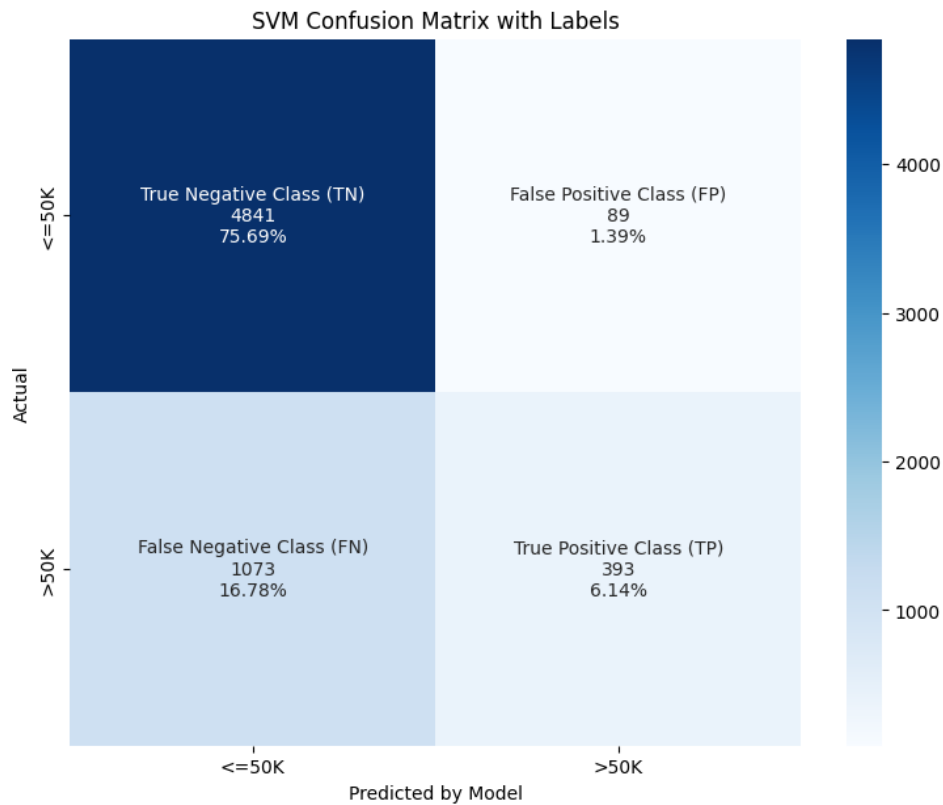
Figure 55: Confusion matrix for a Support Vector Machine (SVM)

5.1.3. Random Forest

Random Forest, developed by Leo Breiman in 2001, is an ensemble learning method designed to improve prediction accuracy by combining multiple decision trees [8]. Each tree in the Random Forest is constructed using a method called bootstrap aggregation, or "bagging," which trains each tree on a random subset of the original data [8]. This aggregation technique helps reduce variance and increase model robustness, making Random Forest well-suited for high-dimensional datasets.

The Random Forest model uses an ensemble learning approach by constructing multiple decision trees and then combining their outcomes to produce a final prediction. This approach helps to prevent overfitting, making the model to be more reliable and accurate [9]. It can also highlight which features are most important for making predictions. Overall, it is a robust model and combines the strengths of many trees to improve performance. In addition to the above Random Forest works very well with the large datasets [10]. It is also resilient to some

noise maintaining performance with noisy or imperfect data as well. The results of models are shown below as well.

```
[910]:  rf = RandomForestClassifier()
        rf.fit(X_train, y_train)

[910]:   ▾   RandomForestClassifier
        RandomForestClassifier()
```

Figure 56: Training Random Forest Classifier

The Random Forest model was trained on the same training dataset in which SVM was trained.

```
# Predictions and accuracy
y_pred_rf = rf.predict(X_test)
print("RF Accuracy on Test Set:", accuracy_score(y_test, y_pred_rf))

RF Accuracy on Test Set: 0.8520950594121326
```

Figure 57: Random Forest model accuracy

The Random Forest gives us 85.20% accuracy, which is around 3.5% greater than SVM model as shown in above figure 57.

The precision of Random Forest classifier is 0.6976.

```
precision = precision_score(y_test, y_pred_rf)
print(f"Precision: {precision:.4f}")

Precision: 0.6976
```

Figure 58: Random Forest model precision

Got 0.6262 as the recall score of the model.

```
recall = recall_score(y_test, y_pred_rf)
print(f"Recall: {recall:.4f}")

Recall: 0.6262
```

Figure 59: Random Forest model recall score

The F1 score of random forest model is 0.6600. Whereas the AUC-ROC score is 0.7727

```
f1 = f1_score(y_test, y_pred_rf)
print(f"F1-Score: {f1:.4f}")
```

F1-Score: 0.6600

```
auc = roc_auc_score(y_test, y_pred_rf)
print(f"AUC-ROC: {auc:.4f}")
```

AUC-ROC: 0.7727

Figure 60; F1 score and AUC-ROC score of Random Forest model.

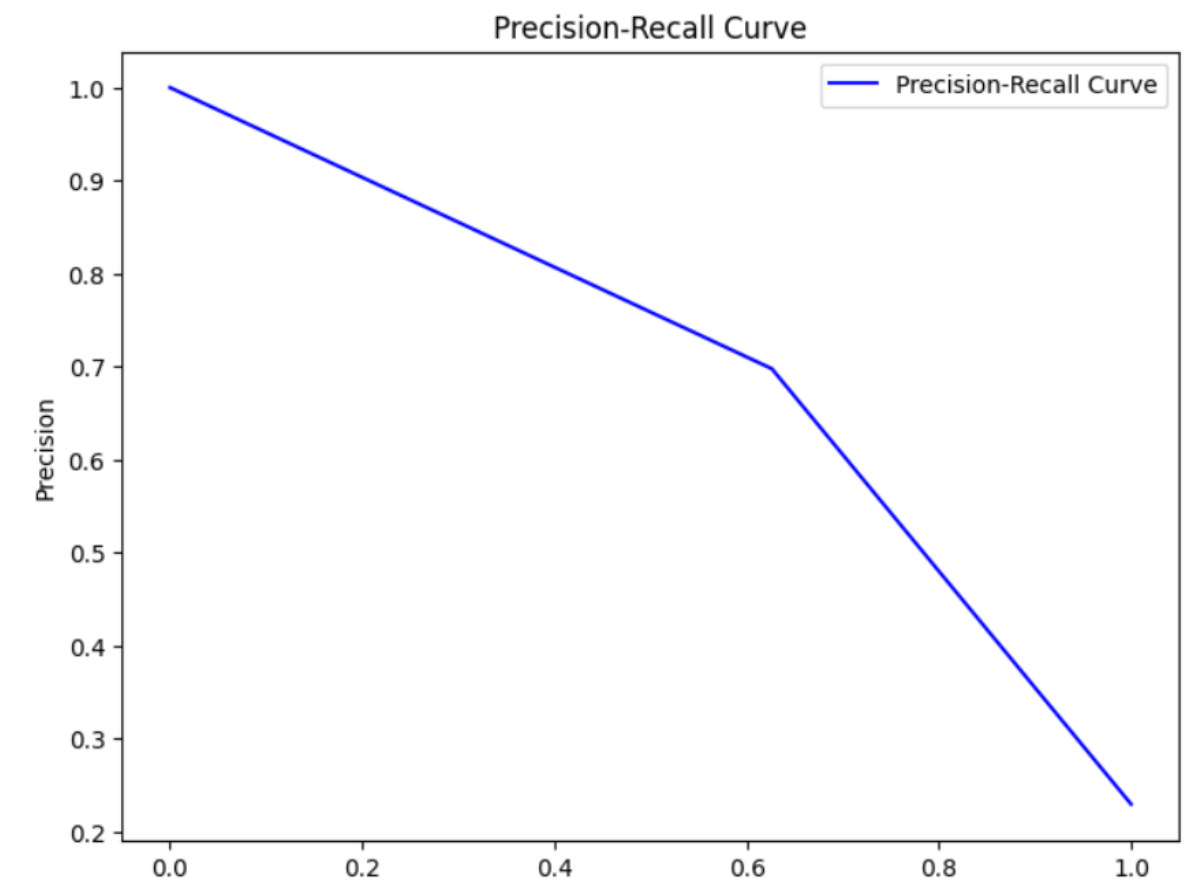Figure 61 shows the Precision-Recall curve of Random Forest model

Figure 61: Precision-Recall Curve of Random Forest model.

The Random Forest model attained an accuracy of 85.20%, surpassing the SVM model. Precision of model is 69.76% where recall of model is 62.62%. It shows the Random Forest classifier performs well, particularly for predicting negative classes.

 The confusion matrix summary is as follows.

- True Positive (TP): 918 instances were classified as positive. It means that 927 individuals were earning more than $50k in test dataset.

- True Negatives (TN): 4481 instances were correctly predicted as negative. It means that 4532 individuals were earning less than $50K in dataset.

- False Positive (FP): 398 instances were wrongly classified. They belong to negative class but classified as positive.

- False Negative (FN): 548 instances were wrongly classified. They belong to positive class but classified as negative.

```
conf_matrix = confusion_matrix(y_test, y_pred_rf)
print("Confusion Matrix:\n", conf_matrix)

# Defining class labels
class_labels = ['True Negative Class (TN)', 'False Positive Class (FP)', 'False Negative Class (FN)', 'True Positive Class (TP)']

# Formatting group counts and percentages
count_values = [f"{int(val)}" for val in conf_matrix.ravel()]
percentage_values = [f"{val:.2%}" for val in conf_matrix.ravel() / np.sum(conf_matrix)]

# Creating labels combining names, counts, and percentages
combined_labels = [f"{label}\n{count}\n{percent}" for label, count, percent in zip(class_labels, count_values, percentage_values)]
combined_labels = np.array(combined_labels).reshape(2, 2)

# Plotting the confusion matrix with annotations
plt.figure(figsize=(9, 7))
sns.heatmap(conf_matrix, annot=combined_labels, fmt='', cmap="Blues",
            xticklabels=['<=50K', '>50K'], yticklabels=['<=50K', '>50K'])

plt.ylabel('Actual')
plt.xlabel('Predicted by Model')
plt.title('Random Forest Confusion Matrix with Labels')
plt.show()

Confusion Matrix:
 [[4532  398]
 [ 548  918]]
```

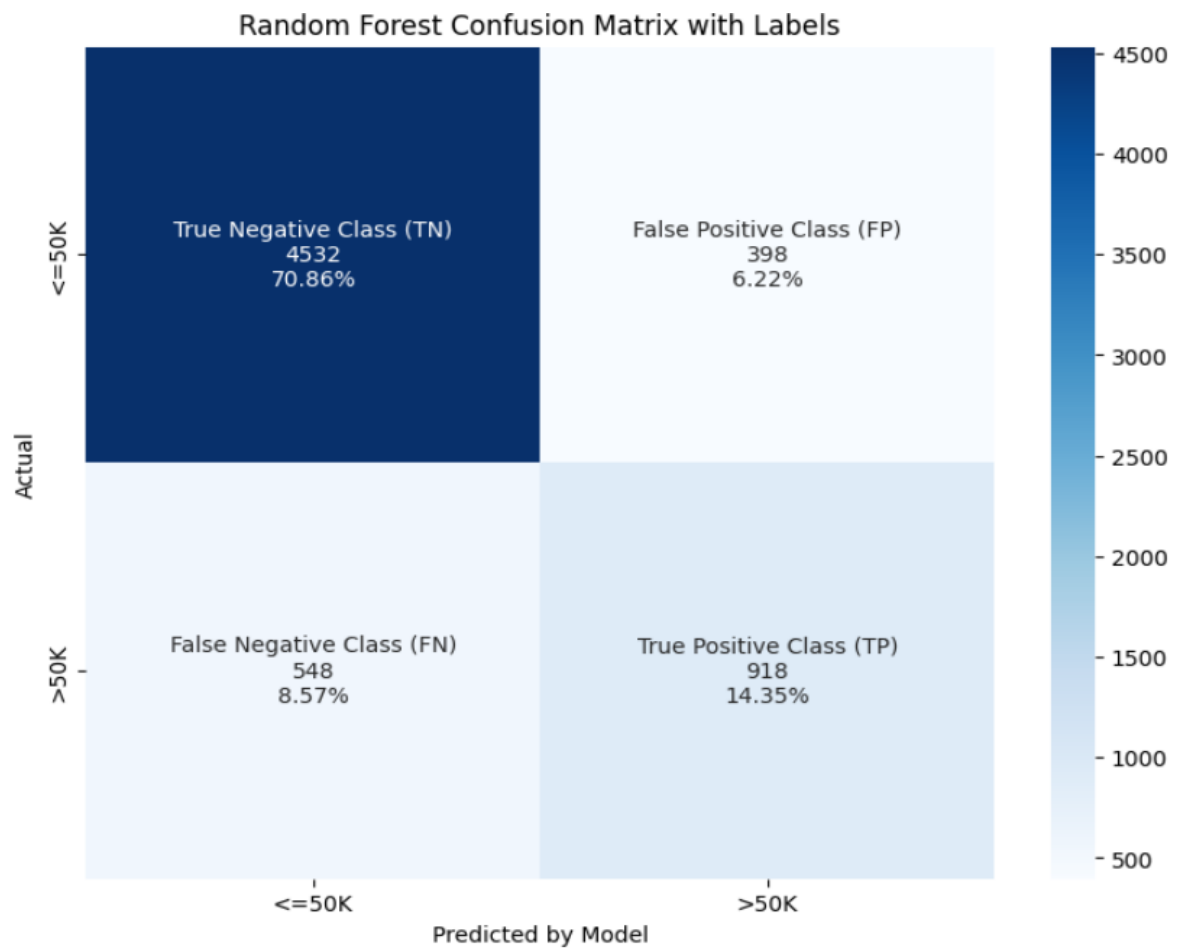*Figure 62: Confusion matrix function of Random Forest model*

Figure 63: Confusion matrix of Random Forest model

6. DISCUSSION

The results of both methods demonstrate the effectiveness of data preprocessing in getting the output. After addressing missing values, encoding categorical variables, and standardizing numerical variables, the dataset was prepared for training. The same metrics were used for evaluation of both models. The SVM achieved an accuracy of 81.83% on test set, showing reasonable performance. Whereas Random Forest got an accuracy of 85.20%. The results show that True Negative was predicted with high accuracy. However, both models struggle with false negative. The results also highlight the importance of feature engineering, preprocessing and evaluation in achieving reliable results.

To increase accuracy, various machine learning algorithms need to be explored. Furthermore, feature engineering should be done to mitigate the inherent bias in the dataset. Both models have their shortcomings and strengths. Therefore, it is crucial to choose the right model. Furthermore, quality and quantity of datasets play an important role in machine learning algorithms.

REFERENCES

[1] B. Becker and R. Kohavi, *Adult,* 1996.

[2] D. Montgomery, "Design and Analysis of Experiments," in *Design and Analysis of Experiments, 10th Edition*, Wiley, 2019, pp. 21-23.

[3] A. Agresti, "An Introduction to Categorical Data Analysis," in *An Introduction to Categorical Data Analysis, 3rd Edition*, Wiley, 2018, pp. 30-35.

[4] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science,* vol. 349, no. 6245, pp. 255-260, 2015.

[5] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science,* vol. 349, p. 255–260, 2015.

[6] C. Cortes and V. Vapnik, "Support-Vector Networks," *Machine Leaming,* no. 20, pp. 273-297, 1995.

[7] S. Suthaharan and S. Suthaharan, "Support vector machine," *Machine learning models and algorithms for big data classification: thinking with examples for effective learning,* p. 207–235, 2016.

[8] L. Breiman, "Random Forests," *Machine Learning,* vol. 1, no. 45, pp. 5-32., 2001.

[9] S. J. Rigatti, "Random forest," *Journal of Insurance Medicine,* vol. 47, p. 31–39, 2017.

[10] I. H. Sarker, "Machine learning: Algorithms, real-world applications and research directions," *SN computer science,* vol. 2, p. 160, 2021.

APPENDIX

```python
import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import LabelEncoder, StandardScaler

import matplotlib.pyplot as plt

import seaborn as sns

from scipy import stats

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import mean_squared_error, r2_score

from sklearn.feature_selection import chi2

from sklearn.metrics import accuracy_score

from sklearn.svm import SVC

from sklearn.metrics import accuracy_score, confusion_matrix

%matplotlib inline

from sklearn.metrics import precision_score

from sklearn.metrics import recall_score

from sklearn.metrics import f1_score

from sklearn.metrics import roc_auc_score

from sklearn.metrics import classification_report

from sklearn.metrics import precision_recall_curve
```

```
file_path = r"C:\Users\Thisura\Desktop\sliit\ML\assignment 2\MS23464582\adult.data"

df = pd.read_csv(file_path)


df.head()


#adding Column Names in the data set.


names_of_coulmns = ['age', 'employment', 'final_weight', 'education', 'education_number',

            'marital_status', 'profession', 'relationship', 'race', 'gender',

            'capital-gain', 'capital-loss', 'week_hours', 'country', 'income']


# Renaming the columns in the dataset

df.columns = names_of_coulmns


#Displaying the column names of dataset after assigning names.

df.head()


df.dtypes


#making '?' consistent removing spaces in the columnIde

df = df.apply(lambda x: x.str.strip() if x.dtype == "object" else x)
```

```
#identifying Missing Values in columns

df.replace("?", np.nan, inplace=True)


missing_data = df.isnull().sum()


print(missing_data[missing_data > 0])


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

missing_data.plot(kind='bar', color='blue')


plt.title('Missing Values in Each Column')

plt.xlabel('Columns')

plt.ylabel('Number of Missing Values')


plt.show()


df.replace("?", np.nan, inplace=True)


# Replaceing the  missing values in 'employment' and 'profession' columns with their mode

df['employment'].fillna(df['employment'].mode()[0], inplace=True)

df['profession'].fillna(df['profession'].mode()[0], inplace=True)
```

```python
#Missing values are filled

print(df[['employment', 'profession']].isnull().sum())

print('Missing values are replaced by Mode of the columns of employement & profession')


missing_country_rows = df[df['country'].isnull()]

print(missing_country_rows)


#Droping rows

df.dropna(subset=['country'], inplace=True)


print('Values with missing rows are deleted')


#Checking unqiue values in the column to standarize it

cat_columns = ['employment', 'education', 'marital_status', 'profession',

                'relationship', 'race', 'gender', 'country', 'income']


# Looping through each categorical column and print unique values

for column in cat_columns:

    values = df[column].unique()

    print(f"Distinct entries in '{column}': {values}\n")


df.describe()
```

```
def plot_box(data, column_name):

    plt.figure(figsize=(10, 5))

    sns.boxplot(y=data[column_name])

    plt.ylabel(column_name)

    plt.title(f'{column_name} - Box Plot')

    plt.show()


def plot_histogram(data, column_name):

    plt.figure(figsize=(10, 5))

    sns.histplot(data[column_name], bins=20, kde=True)

    plt.xlabel(column_name)

    plt.ylabel('Count')

    plt.title(f'{column_name} - Histogram')

    plt.show()


#checking outliers in the Age Column

plot_box(df, 'age')

plot_histogram(df, 'age')


#checking outliers in the Education Numbers

plot_box(df, 'education_number')

plot_histogram(df, 'education_number')
```

```
#checking outliers in the capital Gain

plot_box(df, 'capital-gain')

plot_histogram(df, 'capital-gain')


#checking outliers in the Capital Loss

plot_box(df, 'capital-loss')

plot_histogram(df, 'capital-loss')


#checking outliers in the Weekly Hours

plot_box(df, 'week_hours')

plot_histogram(df, 'week_hours')


anova_features = ['age', 'final_weight', 'education_number', 'capital-gain', 'capital-loss',
'week_hours']

anova_results = {}


for feature in anova_features:

    groups = [df[feature][df['income'] == category] for category in df['income'].unique()]

    anova_results[feature], p_value = stats.f_oneway(*groups)

    print(f"ANOVA result for {feature}: F-statistic = {anova_results[feature]}, P-value =
{p_value}")
```

```python
categorical_features = ['employment', 'education', 'marital_status', 'profession', 'relationship',
'race', 'gender', 'country']


chi2_results = {}

chi2_p_values = {}


df_encoded = df.copy()


# Loop over each categorical feature

for feature in categorical_features:

    # Apply Label Encoding for each categorical feature in the new DataFrame

    df_encoded[feature] = LabelEncoder().fit_transform(df_encoded[feature])


    # Perform the Chi-Square test

    chi2_stat, p_val = chi2(df_encoded[[feature]], df_encoded['income'])


    # Store the first P-value for each feature

    chi2_results[feature] = p_val[0]

    print(f"Chi-Square result for {feature}: P-value = {p_val[0]}")


# Plotting ANOVA and Chi-Square results

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

```
# ANOVA bar plot

plt.subplot(1, 2, 1)

plt.bar(anova_results.keys(), anova_results.values(), color='skyblue')

plt.title('ANOVA F-statistic for Numerical Features')

plt.xlabel('Features')

plt.ylabel('F-statistic')

plt.xticks(rotation=45)


# Chi-Square bar plot

plt.subplot(1, 2, 2)

plt.bar(chi2_results.keys(), chi2_results.values(), color='lightcoral')

plt.title('Chi-Square P-values for Categorical Features')

plt.xlabel('Features')

plt.ylabel('P-values')

plt.xticks(rotation=45)


plt.tight_layout()

plt.show()


#dropping of final_weight column

df.drop('final_weight', axis=1, inplace=True)

print('The final_weight column is deleted')
```

```
#data set columns after dropping final_weight column

print(df.columns)


#dropping of employment  column

df.drop('employment', axis=1, inplace=True)

print('The employment  column is deleted')


#data set columns after dropping employment column

print(df.columns)


df['income'] = df['income'].str.strip()


income_mapping = {"<=50K": 0, ">50K": 1}


df['income'] = df['income'].map(income_mapping)


income_counts = df['income'].value_counts()


print(income_counts)

print(df['income'].unique())

print('income is mapped to 0 & 1')
```

```
df.head()


cat_col = df.select_dtypes(include=['object']).columns

print(cat_col)


df.head()


label_encoders = {}

for col in cat_col:

    # Initializing the encoder for each column

    lbl_encoder = LabelEncoder()


    # Transforming the categorical column with label encoding

    df[col] = lbl_encoder.fit_transform(df[col])


    # Storeingthe encoder object for later use

    label_encoders[col] = lbl_encoder


df.head()


#transforming these columns to standard format

numerical_cols = ['age', 'capital-gain', 'capital-loss', 'education_number','week_hours']
```

```
scaler = StandardScaler()


df[numerical_cols] = scaler.fit_transform(df[numerical_cols])

print(df.head())


#Droping X Vecotr for feature space

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


#adding target in a seperate array

y = df['income']


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


X_train.shape


X_test.shape


y_train.shape


y_test.shape


svm = SVC()
```

```python
svm.fit(X_train, y_train)


# Predictions and accuracy

y_pred_svm = svm.predict(X_test)

print("SVM Accuracy on Test Set:", accuracy_score(y_test, y_pred_svm))


precision = precision_score(y_test, y_pred_svm)

print(f"Precision: {precision:.4f}")


f1 = f1_score(y_test, y_pred_svm)

print(f"F1-Score: {f1:.4f}")


auc = roc_auc_score(y_test, y_pred_svm)

print(f"AUC-ROC: {auc:.4f}")


precision, recall, thresholds = precision_recall_curve(y_test, y_pred_svm)


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

plt.plot(recall, precision, label="Precision-Recall Curve", color="green")

plt.xlabel("Recall")

plt.ylabel("Precision")

plt.title("Precision-Recall Curve")
```

```
plt.legend(loc="best")

plt.show()


conf_matrix = confusion_matrix(y_test, y_pred_svm)

print("Confusion Matrix:\n", conf_matrix)


# Defining class labels

class_labels = ['True Negative Class (TN)', 'False Positive Class (FP)', 'False Negative Class
(FN)', 'True Positive Class (TP)']


# Formatting group counts and percentages

count_values = [f"{int(val)}" for val in conf_matrix.ravel()]

percentage_values = [f"{val:.2%}" for val in conf_matrix.ravel() / np.sum(conf_matrix)]


# Creating labels combining names, counts, and percentages

combined_labels    =    [f"{label}\n{count}\n{percent}"    for    label,    count,    percent    in
zip(class_labels, count_values, percentage_values)]

combined_labels = np.array(combined_labels).reshape(2, 2)


# Plotting the confusion matrix with annotations

plt.figure(figsize=(9, 7))

sns.heatmap(conf_matrix, annot=combined_labels, fmt='', cmap="Blues",
```

```
 xticklabels=['<=50K', '>50K'], yticklabels=['<=50K', '>50K'])


plt.ylabel('Actual')

plt.xlabel('Predicted by Model')

plt.title('SVM Confusion Matrix with Labels')

plt.show()


rf = RandomForestClassifier()

rf.fit(X_train, y_train)


# Predictions and accuracy

y_pred_rf = rf.predict(X_test)

print("RF Accuracy on Test Set:", accuracy_score(y_test, y_pred_rf))


precision = precision_score(y_test, y_pred_rf)

print(f"Precision: {precision:.4f}")


recall = recall_score(y_test, y_pred_rf)

print(f"Recall: {recall:.4f}")


f1 = f1_score(y_test, y_pred_rf)

print(f"F1-Score: {f1:.4f}")
```

```python
auc = roc_auc_score(y_test, y_pred_rf)

print(f"AUC-ROC: {auc:.4f}")


precision, recall, thresholds = precision_recall_curve(y_test, y_pred_rf)


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

plt.plot(recall, precision, label="Precision-Recall Curve", color="Blue")

plt.xlabel("Recall")

plt.ylabel("Precision")

plt.title("Precision-Recall Curve")

plt.legend(loc="best")

plt.show()


conf_matrix = confusion_matrix(y_test, y_pred_rf)

print("Confusion Matrix:\n", conf_matrix)


# Defining class labels

class_labels = ['True Negative Class (TN)', 'False Positive Class (FP)', 'False Negative Class
(FN)', 'True Positive Class (TP)']


# Formatting group counts and percentages

count_values = [f"{int(val)}" for val in conf_matrix.ravel()]
```

```python
percentage_values = [f"{val:.2%}" for val in conf_matrix.ravel() / np.sum(conf_matrix)]


# Creating labels combining names, counts, and percentages

combined_labels = [f"{label}\n{count}\n{percent}" for label, count, percent in
zip(class_labels, count_values, percentage_values)]

combined_labels = np.array(combined_labels).reshape(2, 2)


# Plotting the confusion matrix with annotations

plt.figure(figsize=(9, 7))

sns.heatmap(conf_matrix, annot=combined_labels, fmt='', cmap="Blues",

        xticklabels=['<=50K', '>50K'], yticklabels=['<=50K', '>50K'])


plt.ylabel('Actual')

plt.xlabel('Predicted by Model')

plt.title('Random Forest Confusion Matrix with Labels')

plt.show()
```