hcextgy9u

March 31, 2025

0.1 ——————————————- DATA MINING AND INFORMATICS ——————————————

0.2 PROJECT OBJECTIVE :

The objective of this project is to implement differnt data mining technique. This is part of an the assessment of the course data mining and informatics. This project is sub divided into 6 different task or workshop:

1. Workshop 1 : Write a function to capitalize all words in an optional text input and print the result. Prompt the user for a circle's radius, compute its area, and display it. Load the 'adult' dataset, split its columns into two equal halves, swap them, and return the modified dataset. Create a function that compares values in two numerical columns, returning True if the first column's value is greater and False otherwise. Append a new column with these results and apply the function to the 'age' and 'hours-per-week' columns, then print the updated dataset.
2. Workshop 2 : This workshop contributes 15% to the final grade and requires applying data preprocessing techniques from Lecture 2, with additional research where needed. The adult dataset must be downloaded, imported, and fully preprocessed in Jupyter Notebook. Steps include: changing display settings and printing the first 100 rows, printing dataset information, identifying and handling null values with justification, ensuring no missing values remain, encoding data for income prediction, and normalizing the dataset. Comments must be included in the code to avoid mark deductions.
3. Workshop 3 :This workshop contributes 25% to the final grade and requires applying four classifiers from Lecture #3—Support Vector Machine (SVM), Decision Tree (DT), Random Forest (RF), and K-Nearest Neighbors (KNN)—to the adult_WS#3 dataset for income prediction. Tasks include computing the confusion matrix and evaluation metrics for all classifiers, plotting feature importance values, and identifying the three most important features impacting income, which must be reported in the report cell. An ordinal encoder should be used for both nominal and ordinal columns to optimize processing time. Model accuracy should be improved where possible. Comments must be added to the code, and even incomplete but well-commented code will receive partial credit.
4. Workshop 4 :This workshop contributes 15% to the final grade and requires applying K-Means and Hierarchical Clustering to three optional columns from the adult_WS4 dataset. The optimal number of clusters must be determined for both methods (10%). Additionally, Principal Component Analysis (PCA) must be applied to extract the first two principal components (n_components=2), followed by a scatter plot visualizing the dataset's two classes based on the income column (5%). Before applying PCA, categorical columns should be encoded, the

dataset should be normalized, and the income column should be removed. Code must be well-commented with brief explanations to avoid mark deductions.

5. Workshop 5 :This task contributes 10% to the final grade and requires applying the Apriori algorithm to the 'Portugal_online_retail', 'Sweden_online_retail', and 'UK_online_retail' datasets. The algorithm should be tested with three different confidence levels, and the best confidence level should be selected for each dataset. The top three most important association rules for each dataset must be determined and reported in the report cell, along with an explanation of what each rule means. Completing the report cell is required, and comments should be added to the code for clarity.

6. Workshop 6 : This task contributes 10% to the final grade and requires classifying the sentiments in the 'Tweets' dataset using six classifiers. You must calculate all relevant evaluation metrics for the classifiers. If the processing time is too long, reducing the number of samples is allowed. The code should be well-commented, with clear explanations of what each part is doing. Additionally, efforts should be made to improve the model's performance as much as possible.

[ ]:
```

```

0.2.1 LIBRARIES :

[7]:
```
!pip install nltk
```

```
Requirement already satisfied: nltk in c:\users\user\anaconda3\lib\site-packages
(3.9.1)
Requirement already satisfied: click in c:\users\user\anaconda3\lib\site-
packages (from nltk) (8.1.7)
Requirement already satisfied: joblib in c:\users\user\anaconda3\lib\site-
packages (from nltk) (1.4.2)
Requirement already satisfied: regex>=2021.8.3 in
c:\users\user\anaconda3\lib\site-packages (from nltk) (2023.10.3)
```

[8]:
```
!pip install TextBlob
```

```
Requirement already satisfied: TextBlob in c:\users\user\anaconda3\lib\site-
packages (0.19.0)
Requirement already satisfied: nltk>=3.9 in c:\users\user\anaconda3\lib\site-
packages (from TextBlob) (3.9.1)
Requirement already satisfied: click in c:\users\user\anaconda3\lib\site-
packages (from nltk>=3.9->TextBlob) (8.1.7)
Requirement already satisfied: joblib in c:\users\user\anaconda3\lib\site-
packages (from nltk>=3.9->TextBlob) (1.4.2)
```

[4]:
```
import math
import pandas as pd
import numpy as np
import seaborn as sns
from collections import Counter
```

```
import matplotlib
import matplotlib.pyplot as plt
import warnings

from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
import nltk
from textblob import TextBlob
from sklearn.preprocessing import normalize ## use to normalize and put our␣
    ↪data on same scale
from sklearn.preprocessing import OneHotEncoder, LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.inspection import permutation_importance
from sklearn.metrics import accuracy_score, classification_report,␣
    ↪confusion_matrix
from sklearn.cluster import KMeans
from scipy.cluster.hierarchy import linkage, dendrogram
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import silhouette_score
from mpl_toolkits.mplot3d import Axes3D
from sklearn.decomposition import PCA
import string
from mlxtend.frequent_patterns import apriori, association_rules
from io import StringIO
matplotlib.use('TkAgg')


warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.filterwarnings("ignore", category=DeprecationWarning)
```

```
[ ]:
```

0.3 WORKSHOP 1 :

TASK 1 :

Complete the following tasks.

NOTE: You should comment on your code. Failing to add comments may result in a mark deduction

TASK 1.1:

Write a code to receive an optional text, capitalise all words in the text and print them (2%) :

[12]:
```
def convertcasetext(text):
    '''
    Capitalize each word in the input text.

    parameters :
    text (string) : any text

    returns :
    string : capitalize text

    '''
    capitalize_text = ''
    all_text = text.split(' ') ## Creates a list from the optional text

    for i in range(0, len(all_text)):
        capitalize_text = capitalize_text + all_text[i].capitalize() + ' ' ##␣
 ↪loops to get every string in text and print them.

    ## remove space around the string
    return capitalize_text.strip()


#TESTING CODE :
intput = 'My name is njinju zilefac fogap and i am a student.'
word = convertcasetext(intput) # testing the method above

print(word)
```

```
My Name Is Njinju Zilefac Fogap And I Am A Student.
```

The above code is a method that capitalize each word in a string. The method create a list call all_text from the optional text and loops through it and capitalizing every text and adding it to an empty string. The mothod finally returns a string which is strip to remove wide space.

[ ]:
```

```

TASK 1.2 :

Write a program that prompts the user to input the radius of a circle and calculates its area (2%) :

```
[18]: ############# WRITE THE CODE IN THIS CELL ###################
def calculatecirclearea():
    '''
     Calculates the area of a circle.

     parameters :
     radius (float) : radius of circle

     returns :
     float : the area of a circle
    '''
    radius = float(input('Enter the radius of a circle: ')) ## prompting the␣
 ↪user to enter an input as a float.
    radiussquare = math.pow(radius, 2) # since the area of a circle is pir2. i␣
 ↪compute the value of r2.
    area = math.pi * radiussquare ## computing the area using math.pi value
    return area

## TESTING CODE
area1 = calculatecirclearea() ### testing the method above.

print(f'The area of the circle is: {area1}')
```

```
Enter the radius of a circle:  10

The area of the circle is: 314.1592653589793
```

The above methods calculate the area of circle after prompting the user to enter the radius of a given circle. The method uses the math library to get the power and the value of pi, since the area of a circe is pir2 and returns the value of the are of the circle.

```
[ ]:
```

TASK 1.3 : Download and import the 'adult' dataset from Canvas. Write a function to split the dataset column-wise into two halves and swap the first half with the second half (3%)

```
[24]: def swap_df(dataset):
    '''
    This method split a dataframe into two halves (column-wse)

    parameter :
    dataset : csv file

    returns :
    dataframe
    '''
    data_df = pd.read_csv(dataset) ## creating a dataframe using the name of␣
 ↪our data set.
```

```
    width = data_df.shape[1] ## getting the width of the the data set.

    half_number_of_col = math.ceil(width/2) ## getting an approximate value of␣
  ↪the half the width of the dataset.

    first_df = data_df.iloc[: , :half_number_of_col] ## getting part of the␣
  ↪dataframe using the half value of the data set.
    second_df = data_df.iloc[: , half_number_of_col:] ## getting part of the␣
  ↪dataframe using the half value of the data set.

    swap_df = pd.concat([second_df, first_df], axis=1) ## using concat method␣
  ↪to swap the fist and the second part.

    return swap_df

## TESTING CODE

df = swap_df('adult.csv') ## testing the above method using a data set, must be␣
  ↪a csv.
df
```

```
<IPython.core.display.HTML object>
```

[24]:

The above method is to split and create a new dataframe by swaping columns of another dataframe. The method takes the name of the dataframe as a csv and read it as a dataframe. It thens calculate the width of the dataframe to i can get an approximate value of half of its value. it now splits the data frame into two halves using half of the value of the width of the dataframe. it now using a concat method to join the second part and the fist part and then returns the new dataframe.

[ ]:

0.3.1 Task 1.4:

Write a function that takes the names of two numerical columns as input and compares their values for all rows. If the value in the first column is greater than the value in the second column, the function should return True; otherwise, it should return False. The function should append a new column to the dataset to store the comparison results for all rows. Apply the function to the 'age' and 'hours-per-week' columns in the adult dataset and print the resulting dataset (3%)

[30]:
```
## Method 1
def createmewcol(col1, col2 , dataframe):
    '''
    create a new column in a dataframe by comparing two columns
    parameters :
    col1 (string) : column name 1
    col2 (string) : column name 2
    dataframe : the dataframe u want to use

    returns :
    dataframe (object)

    '''
    j = 0  ## j is use to get values from both columns of the dataframe

    result = [] ## a list to hold values of gotten from comparinng values of␣
 ↪both columns

    while j < len(dataframe[col1]):
            ## comparing the values of the first columns and the second columns␣
 ↪and adding to the list
            if dataframe[col1][j] > dataframe[col2][j]:
                result.append('TRUE')
            else:
                result.append('FALSE')

            j += 1


    dataframe['compared_result'] = result ## creating a new columns to hold the␣
 ↪results from the list.

    return dataframe

## TESTING THE CODE :
data_df = pd.read_csv('adult.csv') ## reading in the data.

new_data_df = createmewcol('age', 'hours-per-week' , data_df) ## calling the␣
 ↪method to test with age and hours-per-week values
new_data_df
```

```
<IPython.core.display.HTML object>
```

[30]:

[31]:
```
## Method 2
def createmewcol(col1, col2, dataframe):
    '''
    Create a new column in a dataframe by comparing two columns.

    Parameters:
```

```
    col1 (str): Name of the first column
    col2 (str): Name of the second column
    dataframe (pandas.DataFrame): The DataFrame to modify

    Returns:
    pandas.DataFrame: DataFrame with a new column 'compared_result'
    '''
    # Validate column names
    if col1 not in dataframe.columns or col2 not in dataframe.columns:
        raise ValueError("One or both column names not found in the DataFrame")

    dataframe['compared_result'] = dataframe[col1] > dataframe[col2]

    return dataframe

# TESTING THE CODE:
data_df = pd.read_csv('adult.csv')  # Reading in the data
new_data_df = createmewcol('age', 'hours-per-week', data_df)  # Applying the␣
 ↪function

new_data_df
```

```
<IPython.core.display.HTML object>
```

[31]:

The above method is use to compare values of two columns and create a new columns based on the result gotten from the comparison. In other to access values of both column i created a variable call j. Then i use a while loop to interate and get this values. The while loop stops when the value of j is greater than lenght of one of the columns. In the while loop and if statement is use to compare and append result to the list and after this is done a new column is form using the values from the list.

[ ]:

```

```

0.4 WORKSHOP 2 :

TASK 2 :

- This workshop includes marked tasks that contribute 15% to your final grade in this module.
- To complete the tasks, you need to apply the data preprocessing methods discussed in Lecture 2. However, some tasks may require additional research to find appropriate solutions.
- You may duplicate code and report cells if you need more than one for your solution.

0.4.1 TASK 2.1:

Download the adult dataset from Canvas, import it into Jupyter Notebook, and perform a complete data preprocessing. Your solution should include at least the following steps (completing the report cell is required):

a. Change the display setting and print the first 100 rows.

b. Print the dataset infromation.

c. Write a Python script to display the number of Null values in each column. Handle the Null values using an appropriate method, and explain the rationale behind your choice in the report cell. Print the number of null values again at the end to ensure that no null values remain in the processed dataset.

d. Assume we want to predict the income column. Encode the data using the appropriate method.

e. Normalise the data. NOTE: You must add comments to your code. Failure to do so will result in a mark deduction.

[42]:
```
class dataprocessing:
    '''
        This class is for preprocesssing, it does the following :
            - Change dispaly setting to display first 100 rows.
            - Print data information.
            - Check for missing values amd display.
            - Fill in misssing values.
            - Encoding the y.
            - Normalizing the data.
    '''
    def __init__(self, data):

        self.data = data

    def dispaly_first_100(self):
        '''
            This method is use to display the first 100 rows of our data set␣
↪and returns a dataframe.
        '''
        pd.set_option('display.max_rows', 100)
        data = self.data.head(100)
        return data

    def dispaly_data_info(self):
        '''
            This method is used to display information about the data set.
        '''
        return self.data.info()

    def display_missing_values(self):
        '''
            This method is used to display all missing values in the data set.
        '''
        return self.data.isnull().sum()

    def filling_in_values(self, col):
        '''
            This method is use to fill the data set missing values the␣
↪variables with the mode.
        '''
        return self.data[col].fillna(self.data[col].mode()[0], inplace=True)

    def encoding_predictive_value(self, col):
        '''
            This method selects the target variable, y then encodes using lable␣
↪encoder.
        '''
        y = self.data[col]

        le = LabelEncoder()
        y = le.fit_transform(y)

        return y


    def one_hot_encode(self, columns):
        '''
            This method drops the target varaible encodes all norminal␣
↪variables in x using one hot encoder.
        '''
        x = self.data.drop('income', axis = 1)

        encoder = OneHotEncoder(sparse_output=False, drop='first')  # Drop␣
↪first to avoid multicollinearity

        encoded = encoder.fit_transform(x[columns])
        encoded_df = pd.DataFrame(encoded, columns=encoder.
↪get_feature_names_out(columns))
        x = x.drop(columns, axis=1).reset_index(drop=True)
        return pd.concat([x, encoded_df], axis=1)

    def label_encode(self,x_one_hot_encode, columns):
        '''
            This method drops the target varaible encodes all ordinal/binary␣
↪variables in x using label encoder.
        '''
        encoder = LabelEncoder()
        for col in columns:
            x_one_hot_encode[col] = encoder.fit_transform(x_one_hot_encode[col])
        return x_one_hot_encode

    def normalizaing_data_set(self, scalled_data):
        '''
             This method is use to scale our data set using minmaxcaler.
        '''
        scaler = MinMaxScaler()
        numeric_columns = scaler.fit_transform(scalled_data)

        return  numeric_columns
```

[ ]:

```
```

Reading in the dataset :

[46]:

```
data_df = pd.read_csv('adult.csv') ## reading the data and converting it to a␣
  ↪dataframe using pandas.
```

[48]:

```
data_copy = data_df.copy() ## creating a copy of the data set.
```

[ ]:

```
```

Since I have to preprocess the data set, i have to create a copy of the data set so i don't alter the original data set and if there's any changes done, it will not alter the original data set and i can also have a source where i can revert if there is any need to.

[ ]:

```
```

Creating an object of the class :

[54]:

```
## Since i am coding in OO, i created an object of the class called data1.
data1 = dataprocessing(data_copy)
```

[ ]:

```
```

A. Display first 100 rows :

[58]:

```
print("First 100 rows of the dataset:")
data1.dispaly_first_100()
```

```
First 100 rows of the dataset:

<IPython.core.display.HTML object>
```

[58]:

The above method is use to set the dsplay method so we can see a maximum of 100 rows when we are accesssing the data and we have an idea of the overall structure of the data set.

```
[ ]:
```

B. Data set information :

```
[63]: print("Dataset Information:")
      data1.dispaly_data_info()
```

```
Dataset Information:
<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       47879 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      47876 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  48568 non-null  object
 14  income          48842 non-null  object
dtypes: int64(6), object(9)
memory usage: 5.6+ MB
```

Above, we have the information of the data set. we noticed the following about the adult data set: - They are 48842 entries or observations and 15 variables. - Age, fnlwgt,education-num ,capital-gain ,capital-loss ,hours-per-week are of type int64. - Workclass, education, marital-status, occupation, relationship, race, sex, native-country and income are of type object. - The data has has some missing values , we would further see it.

```
[ ]:
```

C. Checking for missing values :

```
[68]: print("Number of Null Values Before Handling:")
      data1.display_missing_values()
```

```
Number of Null Values Before Handling:
```

```
[68]: age                 0
      workclass         963
      fnlwgt              0
      education           0
      education-num       0
      marital-status      0
      occupation        966
      relationship        0
      race                0
      sex                 0
      capital-gain        0
      capital-loss        0
      hours-per-week      0
      native-country    274
      income              0
      dtype: int64
```

From above we see that the columns workclass , occupation and native-country have missing values (963 , 966 and 274) respectively. We need to fix this using appropraite method so we can have a complete data so as to have a highly efficient machine learning model.

D. Fixing null and missing values :

```
[72]: # Replace '?' with NaN for consistency
      data_copy.replace('?', np.nan, inplace=True)
```

```
[74]: data1.filling_in_values('workclass')
      data1.filling_in_values('occupation')
      data1.filling_in_values('native-country')
```

From above I use the method, 'filling_ing_values' to fill in the missings values in workclass , occupation and native-country. The use the mode, which is the value with the highest occurence.This is use because we are dealing with cartegorical data. For numerically data we can use the median or the mean to fill in the missing values. Ideally it is adviceable to reaplace these values with 'unknown' so as to reduce bais when training our machine learning model or use efficient model to predict this value to fill in the gap.

```
[77]: print("Number of Null Values After Handling:")
      data1.display_missing_values()
```

```
Number of Null Values After Handling:
```

```
[77]: 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
dtype: int64
```

```
[ ]:
```

E1. Encoding the predictive value (INCOME) :

```
[81]: print("Encoded 'income' column (first 10 values):")
      y = data1.encoding_predictive_value('income')
      y[:10]
```

```
Encoded 'income' column (first 10 values):
```

```
[81]: array([0, 0, 0, 0, 0, 0, 0, 2, 2, 2])
```

```
[ ]:
```

E2. Encoding the norminal columns :

```
[85]: categorical_nominal = ['workclass', 'education', 'occupation', 'relationship',␣
        ↪'race', 'native-country']
      x_one_hot_encode = data1.one_hot_encode(categorical_nominal)
      x_one_hot_encode
```

```
<IPython.core.display.HTML object>
```

```
[85]:
```

The above method of the class encodes the norminal varaibles (workclass', 'education', 'occupation', 'relationship', 'race', 'native-country') using one hot encoder, this is because it ensures no arbitrary order is imposed on categories , avoids imposing an arbitrary order and is suitable when there is no inherent ranking.

```
[ ]:
```

E2. Encoding the ordinal columns :

```
[89]: categorical_binary_ordinal = ['sex', 'marital-status']
      x_encoded = data1.label_encode(x_one_hot_encode , categorical_binary_ordinal)
      x_encoded
```

```
<IPython.core.display.HTML object>
```

[89]:

The above method of the class encodes the ordinal varaibles ('sex', 'marital-status') using one label encoder, this is because Label Encoding is efficient for binary variables and when order matters, like marital status, which can influence income potential.

[ ]:

```

```

0.4.2 F. Normalize the data :

[95]:

```
data1.normalizaing_data_set(x_encoded)
```

[95]:

```
array([[0.30136986, 0.04413121, 0.8       , …, 1.        , 0.        ,
         0.        ],
        [0.45205479, 0.04805174, 0.8       , …, 1.        , 0.        ,
         0.        ],
        [0.28767123, 0.13758131, 0.53333333, …, 1.        , 0.        ,
         0.        ],
        …,
        [0.28767123, 0.24537874, 0.8       , …, 1.        , 0.        ,
         0.        ],
        [0.36986301, 0.04844413, 0.8       , …, 1.        , 0.        ,
         0.        ],
        [0.24657534, 0.11491866, 0.8       , …, 1.        , 0.        ,
         0.        ]])
```

[97]:

```
print("In the adult dataset, missing values are often represented by '?' in␣
 ↪columns like 'workclass', 'occupation', and 'native-country'. "
      "These were replaced with NaN and then filled with the mode of each␣
 ↪column. The mode was chosen because these are categorical variables, "
      "and using the most frequent category preserves the dataset's␣
 ↪distribution better than arbitrary imputation or dropping rows, especially "
      "given the dataset size. This approach minimizes data loss and maintains␣
 ↪representativeness.")
```

```
In the adult dataset, missing values are often represented by '?' in columns
like 'workclass', 'occupation', and 'native-country'. These were replaced with
NaN and then filled with the mode of each column. The mode was chosen because
these are categorical variables, and using the most frequent category preserves
the dataset's distribution better than arbitrary imputation or dropping rows,
especially given the dataset size. This approach minimizes data loss and
maintains representativeness.
```

[ ]:

```

```

0.5 WORKSHOP 3 :

TASK 3

- This workshop includes marked tasks that comprise 25% of your final mark in this module.
- You need to read the examples in the 'Lecture #3 - examples' notebook to complete the tasks.

TASK 3.1 : Apply four classifiers discussed in Lecture #3, i.e. Support Vector Machine (SVM), Decision Tree (DT), Random Forest (RF), and K-nearest neighbours (KNN) classifiers to the adult_WS#3 dataset available on Canvas to predict the income column. Calculate the confusion matrix and evaluation metrics for all classifiers. Plot the features' importance values, determine the three most important features (i.e. columns) which have the highest impact on the income and report them in the report cell (25%).

NOTE1: To decrease the processing time, use an ordinal encoder for both nominal and ordinal input columns. You don't need to apply the one hot encoder to nominal columns.

NOTE2 You are expected to improve your models in any way possible to get as high accuracy as possible.

NOTE3: You should add comments on your code wherever necessary and briefly explain what the code is doing

NOTE4: Completing the report cell is required only for reporting the three most important features. Other explanations are optional.

NOTE5: You will still get some marks if your code doesn't run, but you have written some codes and have added comments on the code.

[ ]:
```

```

Reading in the dataset

[108]:
```
adult_df = pd.read_csv('adult_WS#3.csv')
```

[ ]:
```

```

EDA (Exploratory data analysis) : In this section I am going to perform a summarize exploratory data analysis on our data set so as to get insight of our data set, check on irregularities and understand pattern between inputs and output if any so as to prepare us for preprocessing and model building phase for our project.

[113]:
```
adult_df.sample(20)
```

```
<IPython.core.display.HTML object>
```

[113]:

[115]:
```
adult_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   age             10000 non-null  int64 
 1   workclass       9825 non-null   object
 2   fnlwgt          10000 non-null  int64 
 3   education       10000 non-null  object
 4   education-num   10000 non-null  int64 
 5   marital-status  10000 non-null  object
 6   occupation      9825 non-null   object
 7   relationship    10000 non-null  object
 8   race            10000 non-null  object
 9   sex             10000 non-null  object
 10  capital-gain    10000 non-null  int64 
 11  capital-loss    10000 non-null  int64 
 12  hours-per-week  10000 non-null  int64 
 13  native-country  9939 non-null   object
 14  income          10000 non-null  object
dtypes: int64(6), object(9)
memory usage: 1.1+ MB
```

[117]:
```
adult_df.describe()
```

```
<IPython.core.display.HTML object>
```

[117]:

[118]:
```
adult_df['workclass'].unique()
```

[118]:
```
array(['Private', '?', 'Local-gov', 'Self-emp-inc', 'Federal-gov',
       'Self-emp-not-inc', 'State-gov', nan, 'Without-pay',
       'Never-worked'], dtype=object)
```

[121]:
```
adult_df['occupation'].unique()
```

[121]:
```
array(['Exec-managerial', 'Craft-repair', '?', 'Handlers-cleaners',
       'Sales', 'Other-service', 'Prof-specialty', 'Farming-fishing',
       'Transport-moving', 'Protective-serv', 'Adm-clerical',
       'Tech-support', 'Machine-op-inspct', nan, 'Priv-house-serv',
       'Armed-Forces'], dtype=object)
```

[123]:
```
adult_df['native-country'].unique()
```

[123]:
```
array(['United-States', nan, '?', 'Scotland', 'Columbia', 'Haiti',
       'Canada', 'El-Salvador', 'China', 'Cuba', 'Philippines', 'Germany',
       'Hong', 'Mexico', 'Puerto-Rico', 'Dominican-Republic', 'South',
       'India', 'Jamaica', 'Vietnam', 'England', 'Cambodia', 'Portugal',
       'Japan', 'Iran', 'Italy', 'Greece', 'Taiwan', 'Thailand',
       'Ecuador', 'Poland', 'Outlying-US(Guam-USVI-etc)', 'Hungary',
       'Guatemala', 'France', 'Nicaragua', 'Honduras', 'Laos', 'Ireland',
       'Peru', 'Yugoslavia', 'Trinadad&Tobago'], dtype=object)
```

[ ]:

CONCLUDSION FROM EXPLORATORY DATA ANALYSIS From the above exploratory data analysis, we note the following :

- Our data set has missing values.
- The variable 'workclass' , 'occupation' and 'native-country' has inconsistent values (that is '?') which needs to be taken care of.
- From the describe statictics, base on the value of standard deviation we can see the presence of outliers in columns (capital-loss , capital-gain and fnlwgt).
- Ideally we need to use graphs to have more details about our data set but i don't want to go out of context.

We need to investigate this further and preprocess it so as to build efficient and accurate model.

[ ]:

Preprocessing the data : In this section we will be dealing with preprocesing of our data set. This is a very important step in our machine learning project because anything we train or feed our model with can affect the outcome. We will be check for missing values, duplicates rows and outliers.

[131]:
```python
class Preprocessing:
    """
     This class is use to preprocessing of the data, initial preprocess with␣
 ↪methods for preprocessing.
    """

    def __init__(self , data):
        self.data = data

    def checkforduplicatesrows(self):
        return self.data.duplicated().sum()

    def dealingwithduplicatedrows(self):
        return self.data.drop_duplicates(inplace=True)

    def checkformissingvalues(self):
        return self.data.isnull().sum()

    def filling_in_values(self, col):
        return self.data[col].fillna(self.data[col].mode()[0], inplace=True)
    def dealingwithincorrectcol(self):
        return self.data.replace('?', pd.NA, inplace=True)

    def dealingwithoutliers(self):
        cleandata = self.dealingwithincorrectcol()
        for column in cleandata:
            m1=cleandata[column].quantile(0.50)
            p1=cleandata[column].quantile(0.95)
            cleandata[column] = np.where(cleandata[column] > p1, m1,␣
↪cleandata[column])

    def norminal_encoding(self):
        # Define categorical columns (both nominal and ordinal)
        categorical_columns = [
            'workclass', 'education', 'marital-status', 'occupation',
            'relationship', 'race', 'sex', 'native-country', 'income'
        ]

        encoder = OrdinalEncoder()

        # Apply encoding to categorical columns
        self.data[categorical_columns] = encoder.fit_transform(self.
↪data[categorical_columns])

        return self.data

    def standardizing_data(self):

        X = self.data.drop('income' , axis = 1)

        # Initialize the MinMaxScaler
        scaler = MinMaxScaler()

        return scaler.fit_transform(X)
```

[ ]:

Creating an object of our class Preprocessing :

[135]:
```
adult_df_copy = adult_df.copy()
```

[137]:
```
preprocessingobject = Preprocessing(adult_df_copy)
```

[ ]:

Checking for duplicated rows

```
[141]: preprocessingobject.checkforduplicatesrows()
```

```
[141]: 1
```

Dealing with duplicated rows

```
[144]: preprocessingobject.dealingwithduplicatedrows()
```

```
[146]: preprocessingobject.checkforduplicatesrows()
```

```
[146]: 0
```

From above we can see that there is a duplicated row, so i dropped it.

```
[ ]:
```

Dealing with incorrect data in each columns and dealing with them Just as seen in the above exploratory data analysis we discovered that the columns workclass ,occupation and native-country had incorrect values (?). Therefore we need to fix this ;

```
[152]: preprocessingobject.dealingwithincorrectcol()
```

The above methods replace '?' with NaN , so we will treat as a missing value and treat is accordingly.

```
[ ]:
```

Checking for missing values :

```
[157]: preprocessingobject.checkformissingvalues()
```

```
[157]: age                 0
       workclass         582
       fnlwgt              0
       education           0
       education-num       0
       marital-status      0
       occupation        583
       relationship        0
       race                0
       sex                 0
       capital-gain        0
       capital-loss        0
       hours-per-week      0
       native-country    197
       income              0
       dtype: int64
```

From above just as seen in the exploratory data analysis phase (workclass,occupation and native-country) has missing values. Since they are cartegorical variable we would replace them with the mode.

Dealing with missing values :

```
[161]: preprocessingobject.filling_in_values('workclass')
       preprocessingobject.filling_in_values('occupation')
       preprocessingobject.filling_in_values('native-country')
```

```
[163]: preprocessingobject.checkformissingvalues()
```

```
[163]: 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
       dtype: int64
```

From above we have complete data set, i filled in the missing values using the mode of each cartegorical values.

```
[ ]:
```

Encoding the data set :

```
[168]: preprocessingobject.norminal_encoding()
```

```
<IPython.core.display.HTML object>
```

```
[168]:
```

Since many machine learning algorithms requires numerical input, i converted the cartegorical input into numerical values using OrdinalEncoder. Ideally we normally use different encoders depending on the type of cartegorical value but because of processing time we use just the ordinal encoder.

```
[ ]:
```

NORMALIZATION OF THE DATA

```
[173]: X = preprocessingobject.standardizing_data()
```

```
[175]: X
```

```
[175]: array([[0.16438356, 0.42857143, 0.15082783, …, 0.        , 0.39795918,
         0.94871795],
        [0.26027397, 0.42857143, 0.19816669, …, 0.        , 0.44897959,
         0.94871795],
        [0.10958904, 0.42857143, 0.06545702, …, 0.        , 0.24489796,
         0.94871795],
        …,
        [0.17808219, 0.42857143, 0.24406519, …, 0.        , 0.39795918,
         0.94871795],
        [0.32876712, 0.71428571, 0.09043864, …, 0.        , 0.39795918,
         0.94871795],
        [0.05479452, 0.42857143, 0.17725055, …, 0.        , 0.39795918,
         0.94871795]])
```

In other to improve the performance of our machine learning agloriithms, prevent dominance of feature and reduce computational complexity. I use Min-Max scalling which transform

```
[ ]:
```

CONCLUDSION ON PREPROCCESSING : From above we can see that we have successfully dealt with missing values , duplicated rows, outliers , data encoding and normalizing the data set. Ideally we needed to split our data set into train and test data set before handling missig values and outliers so as to aviod data leakage so as to aviod poor performance of our model on deployment.

```
[ ]:
```

Machine Learning Model Buidling (Support Vector Machine (SVM), Decision Tree (DT), Random Forest (RF), and K-nearest neighbours (KNN) classifiers)

```
[183]: y = preprocessingobject.data['income'].values
       y
```

```
[183]: array([1., 0., 0., …, 0., 1., 0.])
```

```
[185]: counts = Counter(y)
       counts
```

```
[185]: Counter({0.0: 7576, 1.0: 2423})
```

The above class distribution, 7,576 instances of class 0.0 and 2,423 instances of class 1.0. This confirms a significant imbalance in the above dataset, with class 0.0 being much more frequent than class 1.0. Specifically, the ratio of class 0.0 to class 1.0 is approximately 7,576 / 2,423 3.13. This imbalance might affect the machine learning model's performance, often causing it to favor the majority class (0.0) over the minority class (1.0). So we would use this to improve the performance of our model.

[188]:
```
# Split 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)
```

[ ]:
```

```

0.5.1 A. RANDOM FOREST CLASSIFIER

[192]:
```
# Define Random Forest Classifier
rf = RandomForestClassifier(random_state=42)
```

[194]:
```
# Define hyperparameter grid
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [5, 10, 15, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 5]
}
```

[196]:
```
# Perform GridSearchCV to select the best hyperparameter for our model.
grid_search = GridSearchCV(rf, param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)
```

[196]:
```
GridSearchCV(cv=5, estimator=RandomForestClassifier(random_state=42), n_jobs=-1,
             param_grid={'max_depth': [5, 10, 15, None],
                         'min_samples_leaf': [1, 2, 5],
                         'min_samples_split': [2, 5, 10],
                         'n_estimators': [50, 100, 200]},
             scoring='accuracy')
```

[198]:
```
# Best parameters
best_params = grid_search.best_params_
print("Best Parameters:", best_params)
```

```
Best Parameters: {'max_depth': None, 'min_samples_leaf': 2, 'min_samples_split':
2, 'n_estimators': 100}
```

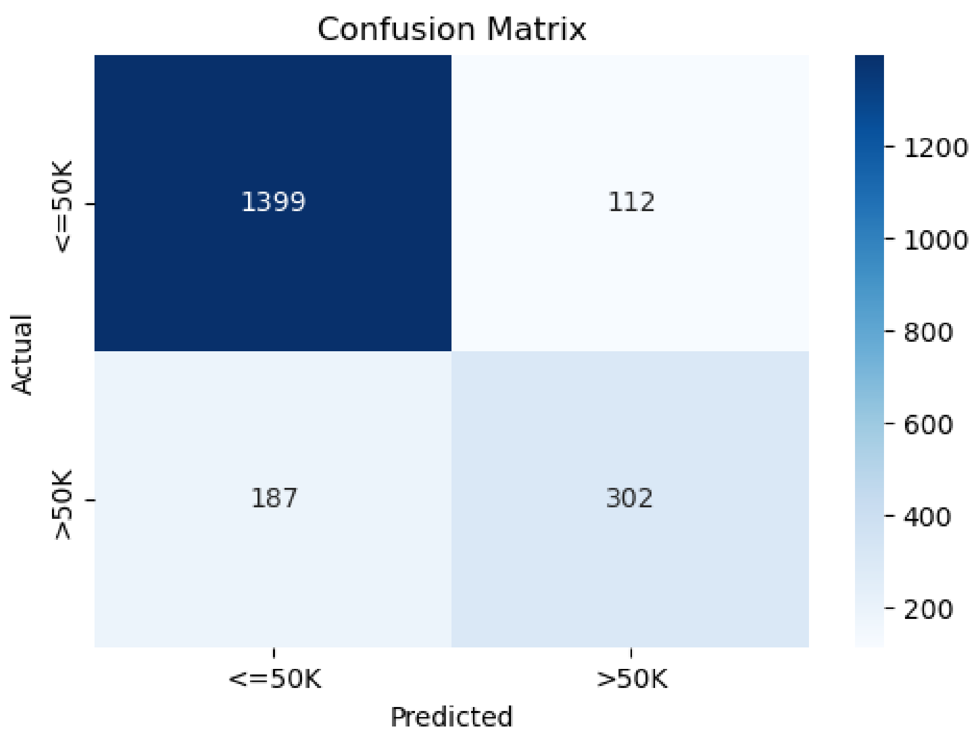
[200]:
```
# Train best model
best_model = grid_search.best_estimator_
```

[202]:
```
# Predictions
y_pred = best_model.predict(X_test)
```

[204]:
```python
# Evaluate Model
accuracy = accuracy_score(y_test, y_pred)
print("\nAccuracy:", accuracy)
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 0.8505

Classification Report:
               precision    recall  f1-score   support

         0.0       0.88      0.93      0.90      1511
         1.0       0.73      0.62      0.67       489

    accuracy                           0.85      2000
   macro avg       0.81      0.77      0.79      2000
weighted avg       0.84      0.85      0.85      2000
```

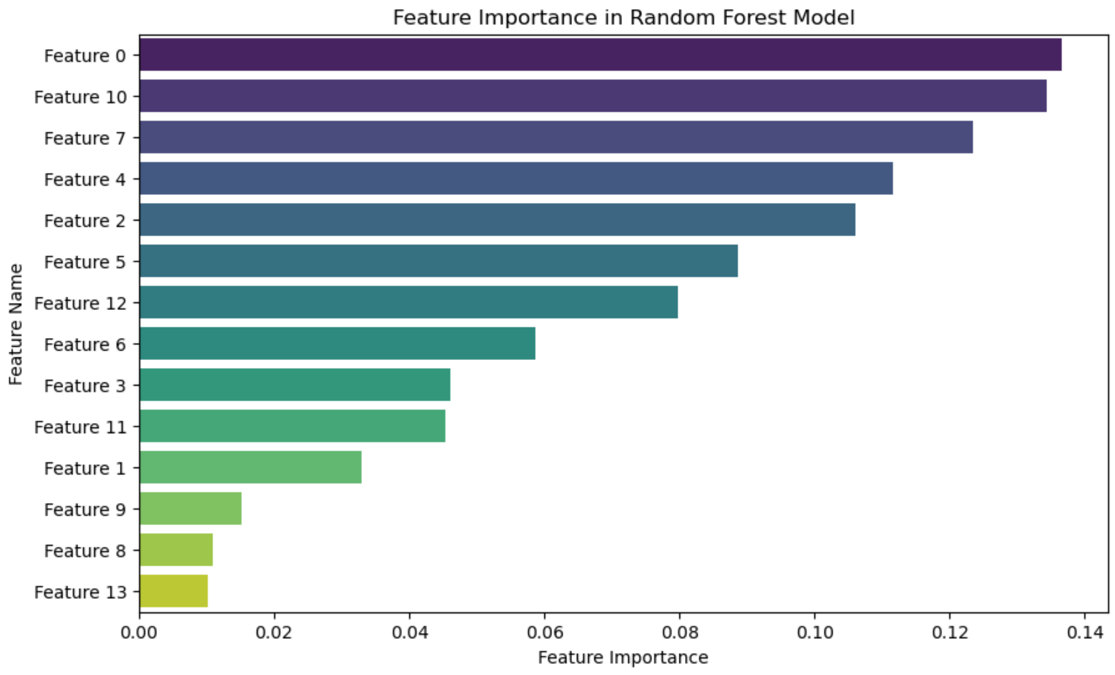
[206]:
```python
# Enable inline plotting in Jupyter
%matplotlib inline

# Assuming y_test and y_pred are defined
conf_matrix = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(6,4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap="Blues",
            xticklabels=["<=50K", ">50K"], yticklabels=["<=50K", ">50K"])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```



[208]:
```
# Enable inline plotting in Jupyter
%matplotlib inline

# Feature Importance Analysis
# Ensure X_train is a DataFrame and extract feature names
if isinstance(X_train, pd.DataFrame):
    feature_names = X_train.columns  # Use actual column names from DataFrame
else:
    # Fallback: if X_train is not a DataFrame (e.g., NumPy array), generate␣
  ↪placeholder names
    feature_names = [f'Feature {i}' for i in range(X_train.shape[1])]
    print("Warning: X_train is not a DataFrame. Using placeholder feature names.
  ↪")

# Get feature importances from the best model
feature_importance = best_model.feature_importances_

# Convert to a Pandas DataFrame for better readability
importance_df = pd.DataFrame({'Feature': feature_names, 'Importance':␣
  ↪feature_importance})

# Sort features by importance (descending order)
importance_df = importance_df.sort_values(by="Importance", ascending=False)
```

```
# Display the top 3 most important features
top_3_features = importance_df.head(3)
print("Top 3 Most Important Features:\n", top_3_features)

# Plot feature importance
plt.figure(figsize=(10,6))
sns.barplot(x=importance_df['Importance'], y=importance_df['Feature'],␣
 ↪palette="viridis")
plt.xlabel("Feature Importance")
plt.ylabel("Feature Name")
plt.title("Feature Importance in Random Forest Model")
plt.show()
```

```
Warning: X_train is not a DataFrame. Using placeholder feature names.
Top 3 Most Important Features:
        Feature  Importance
0    Feature 0    0.136612
10  Feature 10    0.134410
7    Feature 7    0.123471
```



Concludsion on Random Forest Classifier The target output y has the following distribution ({0.0: 7576, 1.0: 2423}) , since it is inbalance we won't use the accuracy to evaluate the performance of our random forest classifier since it proportion of correct predictions (for both classes) out of the total 2000 instances, so it will be misleading to use it. However , the model performs strongly on class 0.0, with high precision (88%) and recall (93%), yielding an f1-score of 0.90. This suggests it's reliable at both predicting 0.0 correctly and capturing most actual 0.0 instances. For class 1.0, however, performance drops: precision is 73%, and recall is only 62%, resulting in an f1-score of 0.67. This indicates more errors—both false positives (27% of 1.0 predictions are wrong) and false negatives (38% of actual 1.0 instances are missed).

Feature importance indicates how much each feature contributes to the model's predictive power. Higher values suggest greater influence.

The top 3 most important feature for this random forest classifier is : - Feature 0 -> 0.136612 = Age , this is the most influential feature in the model. - Feature 10 -> 0.134410 = Sex , it is the second most important feature, very close in value to Feature 0. - Feature 7 -> 0.123471 = Occupation , and this is the 3rd.

The importance scores decrease gradually from Feature 0 down to Feature 11, with Feature 13 having the lowest score among the displayed features. Features 0, 10, and 7 stand out significantly, suggesting they play a critical role in the model's decision-making process. Other features (Feature 3, 11, 1, 9, 8 and 13) have progressively lower importance, indicating they contribute less to the model's predictions.

[ ]:
```

```

0.5.2 DECISION TREE CLASSIFIER

[214]:
```
dt = DecisionTreeClassifier(random_state=42)
```

[216]:
```
dt_param_grid = {
    'max_depth': [3, 5, 10, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'criterion': ['gini', 'entropy']
}
```

[218]:
```
dt_grid_search = GridSearchCV(dt, dt_param_grid, cv=5, scoring='accuracy',␣
 ↪n_jobs=-1)
```

[220]:
```
dt_grid_search.fit(X_train, y_train)
```

[220]:
```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(random_state=42), n_jobs=-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [3, 5, 10, None],
                         'min_samples_leaf': [1, 2, 4],
                         'min_samples_split': [2, 5, 10]},
             scoring='accuracy')
```

[222]:
```
dt_best_params = dt_grid_search.best_params_
print("Best Parameters for DT:", dt_best_params)
```

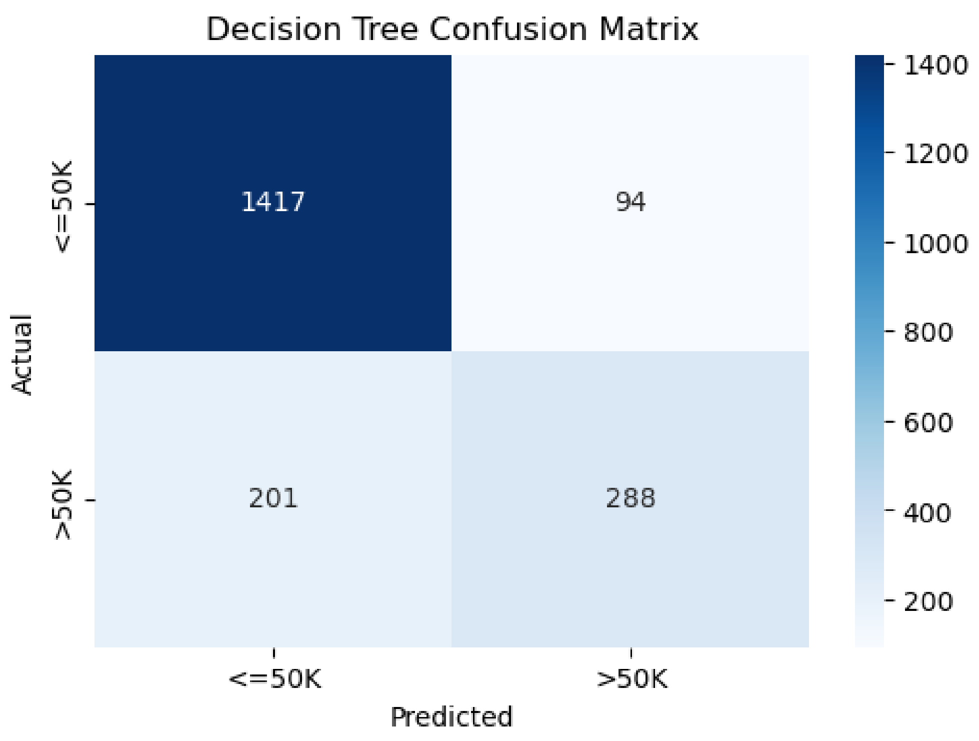
```
Best Parameters for DT: {'criterion': 'gini', 'max_depth': 5,
'min_samples_leaf': 1, 'min_samples_split': 2}
```

[224]:
```
dt_best_model = dt_grid_search.best_estimator_
dt_y_pred = dt_best_model.predict(X_test)
dt_accuracy = accuracy_score(y_test, dt_y_pred)
```

[226]:
```
print("\nDT Accuracy:", dt_accuracy)
print("\nDT Classification Report:\n", classification_report(y_test, dt_y_pred))
```

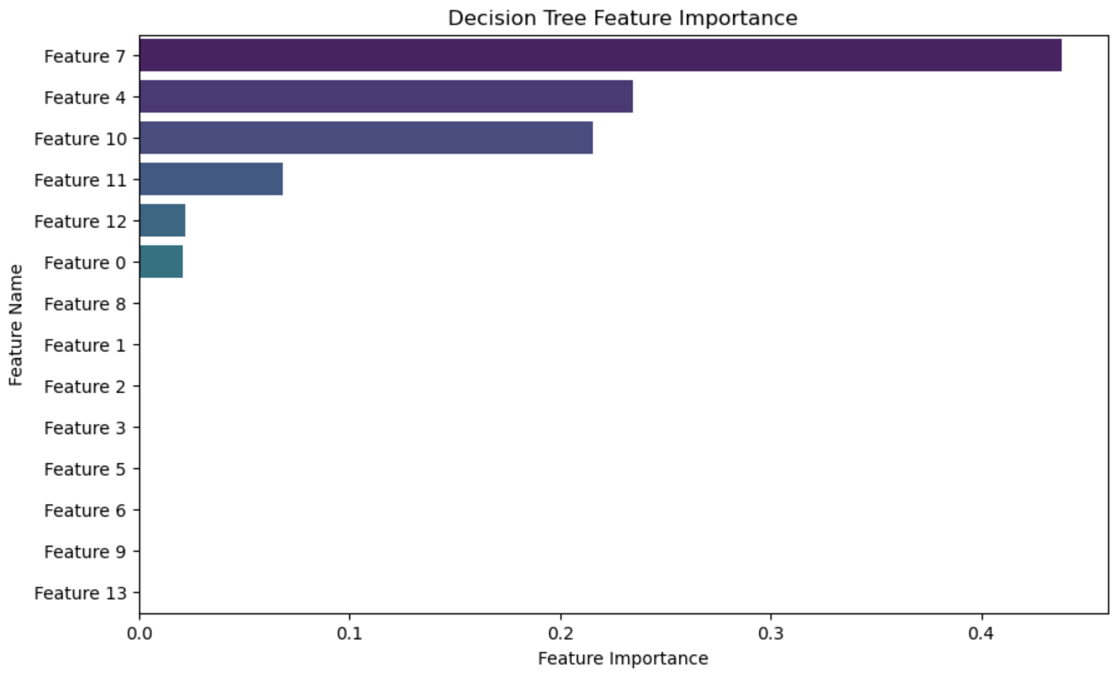
```

DT Accuracy: 0.8525

DT Classification Report:
               precision    recall  f1-score   support

         0.0       0.88      0.94      0.91      1511
         1.0       0.75      0.59      0.66       489

    accuracy                           0.85      2000
   macro avg       0.81      0.76      0.78      2000
weighted avg       0.85      0.85      0.85      2000
```

[228]:
```
# DT Confusion Matrix
dt_conf_matrix = confusion_matrix(y_test, dt_y_pred)
plt.figure(figsize=(6,4))
sns.heatmap(dt_conf_matrix, annot=True, fmt='d', cmap="Blues",
            xticklabels=["<=50K", ">50K"], yticklabels=["<=50K", ">50K"])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Decision Tree Confusion Matrix")
plt.show()
```

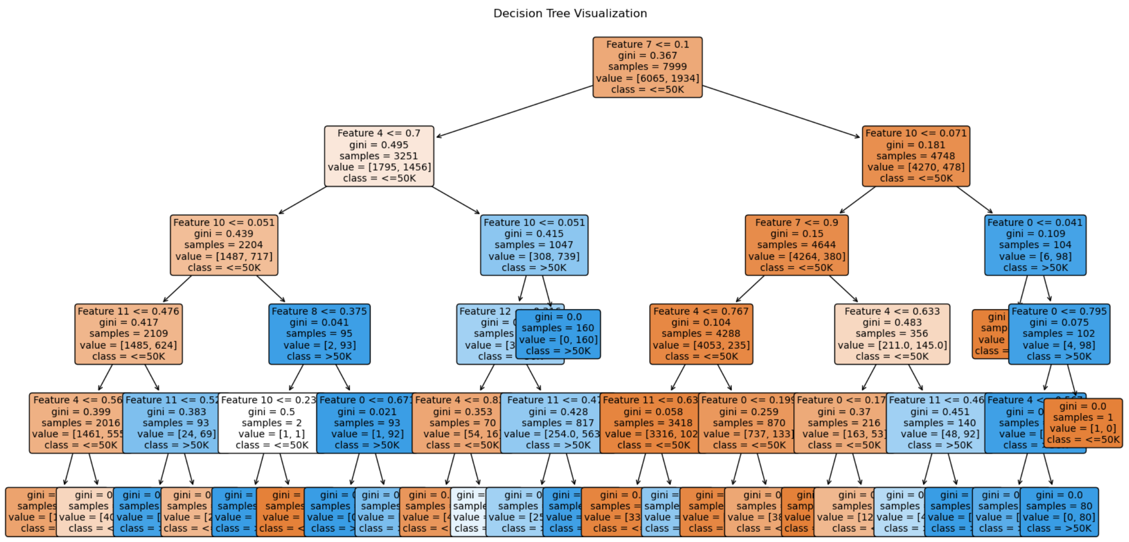


[230]:
```
# DT Feature Importance
dt_importance_df = pd.DataFrame({'Feature': feature_names,
                               'Importance': dt_best_model.
 ↪feature_importances_})
dt_importance_df = dt_importance_df.sort_values(by="Importance",␣
 ↪ascending=False)
print("Top 3 Most Important Features (DT):\n", dt_importance_df.head(3))
plt.figure(figsize=(10,6))
sns.barplot(x=dt_importance_df['Importance'], y=dt_importance_df['Feature'],␣
 ↪palette="viridis")
plt.xlabel("Feature Importance")
plt.ylabel("Feature Name")
plt.title("Decision Tree Feature Importance")
plt.show()
```

```
Top 3 Most Important Features (DT):
        Feature  Importance
7     Feature 7    0.438288
4     Feature 4    0.234289
10   Feature 10    0.215700
```



[232]:
```
# Decision Tree Visualization
plt.figure(figsize=(20,10))
plot_tree(dt_best_model,
          feature_names=feature_names,
          class_names=["<=50K", ">50K"],
          filled=True,
          rounded=True,
          fontsize=10)
plt.title("Decision Tree Visualization")
plt.show()
```



0.5.3 Concludsion on Decision Tree Classifier :

The Decision Tree classifier performs well overall, with an accuracy of 85%, driven by its strong performance on the majority class (0.0: f1-score 0.91). However, it struggles with the minority class (1.0: f1-score 0.66), particularly in recall (59%), due to the class imbalance (1511 vs. 489 instances). While the weighted average f1-score (0.85) aligns with the accuracy, the macro average f1-score (0.78) provides a more balanced view, highlighting the model's limitations on class 1.0. The model performs exceptionally well on the majority class (0.0), with high precision (88%), recall (94%), and f1-score (91%). However, its performance on the minority class (1.0) is noticeably weaker, with a lower recall (59%) and f1-score (66%). This disparity suggests that the model is biased toward the majority class, likely due to the imbalance in the dataset.

Feature importance in a Decision Tree typically reflects how much each feature contributes to reducing impurity (Gini impurity or entropy) across the splits in the tree. Higher values indicate greater influence on the model's decisions. The top 3 feature importance are :

- Feature 7 -> 0.438288 = Sex , This feature has the highest importance score to predicting the income feature.
- Feature 4 -> 0.234289 = Education
- Feature 10 -> 0.215700 = Occupation

Feature 7 is likely a critical variable in this dataset for the Decision Tree model.The low importance of many features (Features 1, 2, 3, 5, 6, 8, 9 and 13) suggests they might not be useful for this particular Decision Tree model, though they could still have some value in other models or contexts.

```
[ ]:
```

0.5.4 SUPPORT VECTOR CLASSIFIER (SVC)

[238]:
```
svc = SVC(random_state=42, kernel='linear')  # Force linear kernel
```

[240]:
```
svc_param_grid = {
    'C': [0.1, 1, 10],
    'gamma': ['scale', 'auto', 0.1]  # Still included but only affects linear␣
  ↪kernel minimally
}
```

[242]:
```
svc_grid_search = GridSearchCV(svc, svc_param_grid, cv=5, scoring='accuracy',␣
  ↪n_jobs=-1)
svc_grid_search.fit(X_train, y_train)
```

[242]:
```
GridSearchCV(cv=5, estimator=SVC(kernel='linear', random_state=42), n_jobs=-1,
             param_grid={'C': [0.1, 1, 10], 'gamma': ['scale', 'auto', 0.1]},
             scoring='accuracy')
```

[243]:
```
svc_best_params = svc_grid_search.best_params_
print("Best Parameters for SVC (Linear):", svc_best_params)
```

```
Best Parameters for SVC (Linear): {'C': 10, 'gamma': 'scale'}
```

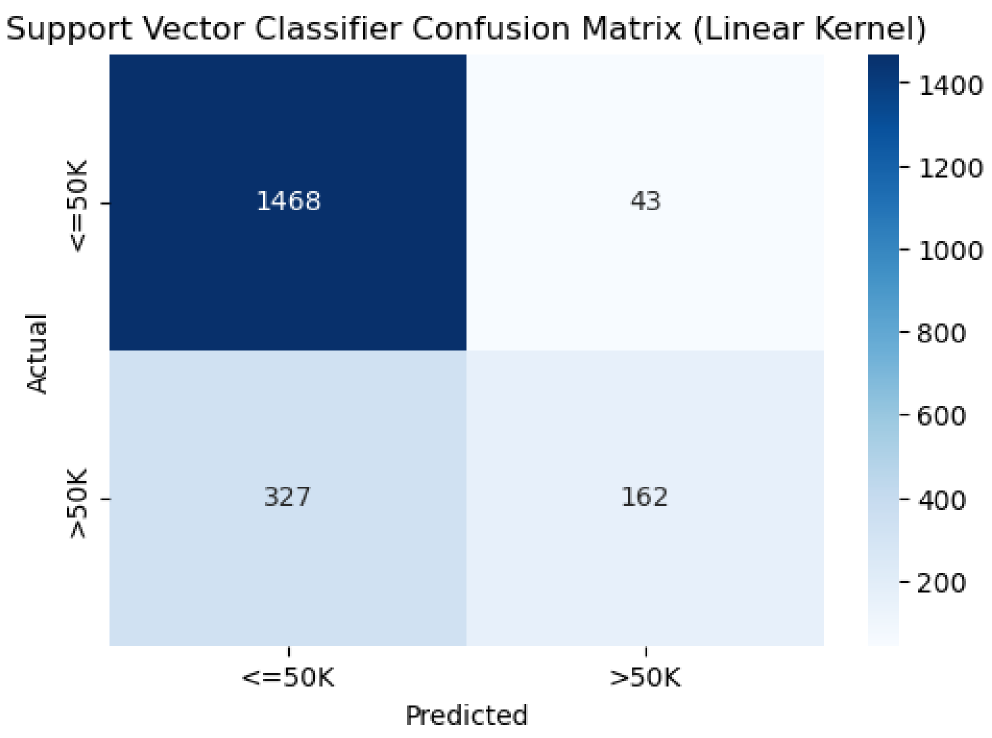
[244]:
```
svc_best_model = svc_grid_search.best_estimator_
svc_y_pred = svc_best_model.predict(X_test)
svc_accuracy = accuracy_score(y_test, svc_y_pred)
```

[245]:
```
print("\nSVC Accuracy:", svc_accuracy)
print("\nSVC Classification Report:\n", classification_report(y_test,␣
  ↪svc_y_pred))
```

```

SVC Accuracy: 0.815

SVC Classification Report:
               precision    recall  f1-score   support

         0.0       0.82      0.97      0.89      1511
         1.0       0.79      0.33      0.47       489

    accuracy                           0.81      2000
   macro avg       0.80      0.65      0.68      2000
weighted avg       0.81      0.81      0.79      2000

```

[246]:
```
svc_conf_matrix = confusion_matrix(y_test, svc_y_pred)
plt.figure(figsize=(6,4))
```

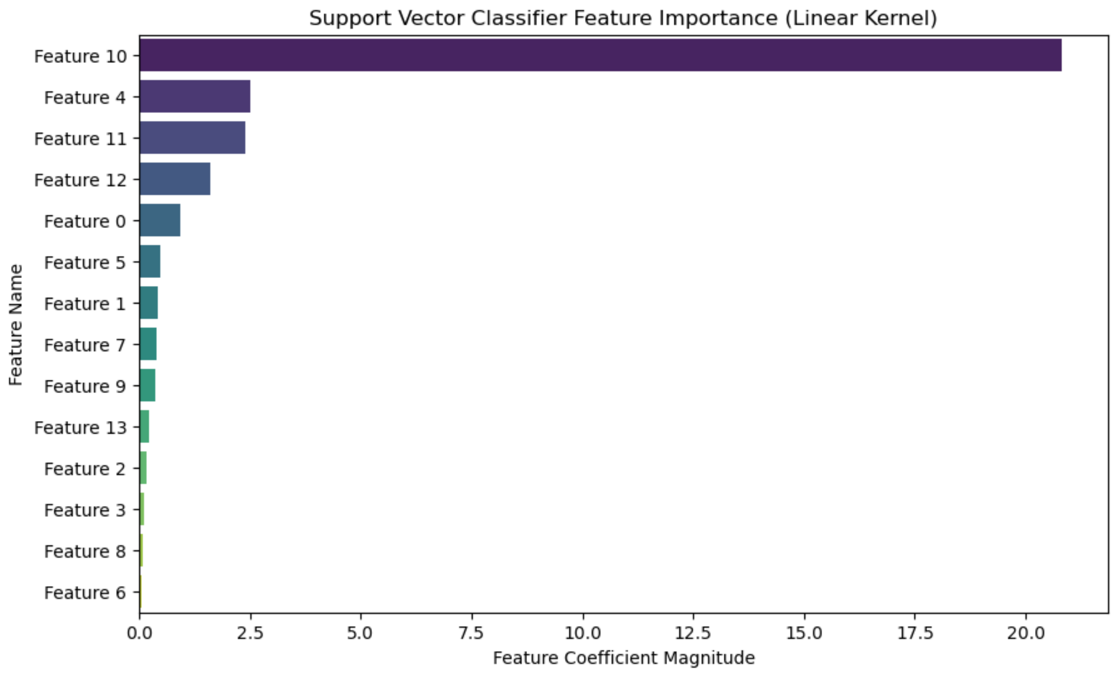
```
sns.heatmap(svc_conf_matrix, annot=True, fmt='d', cmap="Blues",
            xticklabels=["<=50K", ">50K"], yticklabels=["<=50K", ">50K"])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Support Vector Classifier Confusion Matrix (Linear Kernel)")
plt.show()
```



[247]:
```
# Feature importance (always available since we're using linear kernel)
svc_coefficients = pd.DataFrame({'Feature': feature_names,
                                'Coefficient': np.abs(svc_best_model.coef_[0])})
svc_coefficients = svc_coefficients.sort_values(by="Coefficient",␣
 ↪ascending=False)
print("Top 3 Most Influential Features (SVC - Linear Kernel):\n",␣
 ↪svc_coefficients.head(3))
plt.figure(figsize=(10,6))
sns.barplot(x=svc_coefficients['Coefficient'], y=svc_coefficients['Feature'],␣
 ↪palette="viridis")
plt.xlabel("Feature Coefficient Magnitude")
plt.ylabel("Feature Name")
plt.title("Support Vector Classifier Feature Importance (Linear Kernel)")
plt.show()
```

```
Top 3 Most Influential Features (SVC - Linear Kernel):
```

```
         Feature  Coefficient
10   Feature 10    20.819386
4     Feature 4     2.509731
11   Feature 11     2.391031
```



0.5.5 Conludsion on Support Vector Machine :

The SVC performs well on the majority class (0.0), with high recall (97%) and a strong f1-score (0.89), but it struggles with the minority class (1.0), where recall is only 33% and the f1-score is 0.47. The class imbalance (1511 vs. 489) biases the model toward class 0.0, leading to an overall accuracy of 81% that masks the poor performance on class 1.0. The macro average f1-score of 0.68 provides a more balanced view of the model's limitations. If detecting class 1.0 is important, improvements like resampling or class weight adjustments are needed to make the model more effective.

In SVC with a linear kernel, feature importance is typically derived from the absolute values of the weights assigned to each feature in the decision function. Higher absolute weights indicate greater influence on the classification boundary.

- Feature 10 -> 20.819386 = Occupation , This feature has the highest importance score by a significant margin, making it the most influential feature in the SVC model.
- Feature 4 -> 2.509731 = Education
- Feature 11 -> 2.391031 = Capital-Gain

The linear kernel in SVC assumes a linear relationship between features and the target variable. Feature 10's (Occupation) dominance suggests it has a strong linear correlation with the classification outcome. The rapid drop-off in importance after the top few features (Feature 4, 11, 12, 0) indicates that the model relies heavily on a small subset of features, which is typical for linear SVC when the data is separable with a few key variables.

```
[ ]:
```

0.6 K-NEAREST NEIGBORS CLASSIFIERS

```
[258]: knn = KNeighborsClassifier()
```

```
[260]: knn_param_grid = {
           'n_neighbors': [3, 5, 7, 11],
           'weights': ['uniform', 'distance'],
           'p': [1, 2]
       }
```

```
[262]: knn_grid_search = GridSearchCV(knn, knn_param_grid, cv=5, scoring='accuracy',␣
         ↪n_jobs=-1)
       knn_grid_search.fit(X_train, y_train)
```

```
[262]: GridSearchCV(cv=5, estimator=KNeighborsClassifier(), n_jobs=-1,
                    param_grid={'n_neighbors': [3, 5, 7, 11], 'p': [1, 2],
                                'weights': ['uniform', 'distance']},
                    scoring='accuracy')
```

```
[263]: knn_best_params = knn_grid_search.best_params_
       print("Best Parameters for KNN:", knn_best_params)
```

```
Best Parameters for KNN: {'n_neighbors': 11, 'p': 1, 'weights': 'distance'}
```

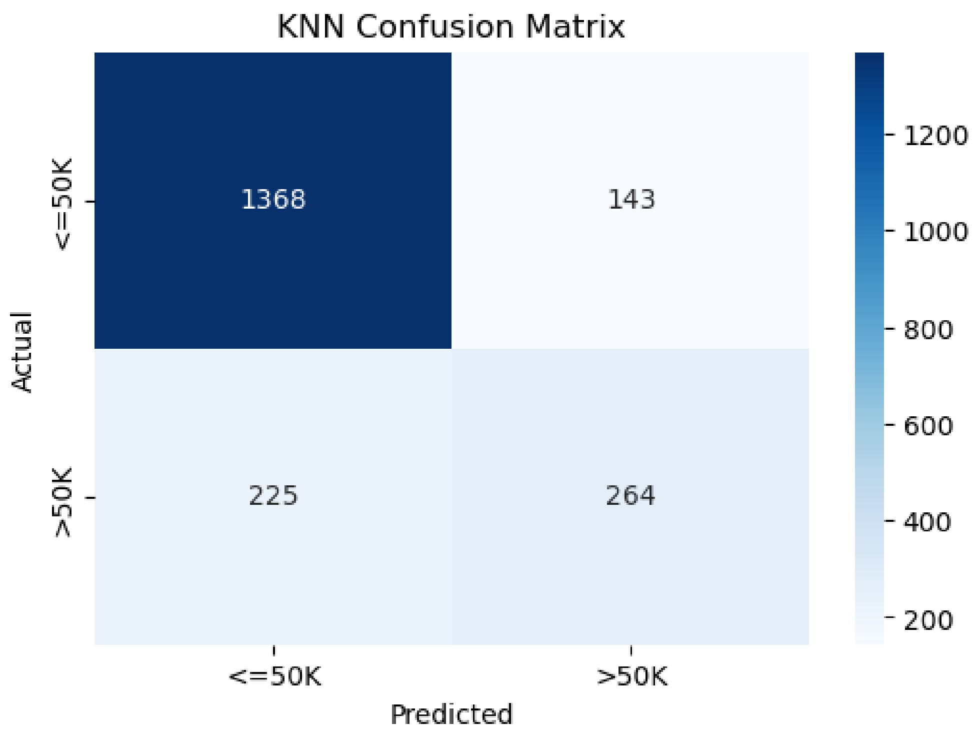
```
[266]: knn_best_model = knn_grid_search.best_estimator_
       knn_y_pred = knn_best_model.predict(X_test)
       knn_accuracy = accuracy_score(y_test, knn_y_pred)
```

```
[268]: print("\nKNN Accuracy:", knn_accuracy)
       print("\nKNN Classification Report:\n", classification_report(y_test,␣
         ↪knn_y_pred))
```

```

KNN Accuracy: 0.816

KNN Classification Report:
               precision    recall  f1-score   support

         0.0       0.86      0.91      0.88      1511
         1.0       0.65      0.54      0.59       489

    accuracy                           0.82      2000
   macro avg       0.75      0.72      0.74      2000
```

```
weighted avg       0.81      0.82      0.81      2000
```

[270]:
```
knn_conf_matrix = confusion_matrix(y_test, knn_y_pred)
plt.figure(figsize=(6,4))
sns.heatmap(knn_conf_matrix, annot=True, fmt='d', cmap="Blues",
            xticklabels=["<=50K", ">50K"], yticklabels=["<=50K", ">50K"])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("KNN Confusion Matrix")
plt.show()
```



[272]:
```
def plot_all_features(importances, feature_names, model_name):
    indices = np.argsort(importances)[::-1]  # Sort indices by importance in␣
↪descending order
    sorted_features = [feature_names[i] for i in indices]
    sorted_importances = importances[indices]

    plt.figure(figsize=(8, 6))
    sns.barplot(x=sorted_importances, y=sorted_features)
    plt.title(f"Feature Importances for {model_name}")
    plt.xlabel("Importance")
    plt.ylabel("Feature")
    plt.tight_layout()
    plt.show()
```

[274]:
```
# If X_train is a NumPy array, generate generic feature names
feature_names = [f"Feature {i}" for i in range(X_train.shape[1])]
```

[276]:
```
# Plot feature importances
plot_all_features(knn_best_model.feature_importances_, feature_names, "KNN")
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[276], line 2
      1 # Plot feature importances
----> 2 plot_all_features(knn_best_model.feature_importances_, feature_names,␣
  ↪"KNN")

AttributeError: 'KNeighborsClassifier' object has no attribute␣
  ↪'feature_importances_'
```

0.6.1 Concludsion on KNN :

The model performs better on Class 0.0 (higher precision, recall, and F1-score) than on Class 1.0. This could be due to the imbalance in the dataset (1511 instances of 0.0 vs. 489 instances of 1.0), meaning Class 0.0 dominates the training data. The lower recall for Class 1.0 (0.54) suggests the model struggles to identify all instances of this class, missing 46% of them. The overall accuracy of 82% is decent, but the weighted average being higher than the macro average reflects the influence of the larger Class 0.0 support.

82% of all 2000 predictions were correct. At first glance, this seems solid, but accuracy can be misleading in imbalanced datasets.

Feature importance : The K-Nearest Neighbors (KNN) algorithm does not natively provide feature importance in the same way that tree-based models (e.g., Random Forest, Decision Tree) or linear models (e.g., Support Vector Classifier with a linear kernel) do. This is due to fundamental differences in how KNN operates and makes predictions.

[ ]:
```

```

0.7 WORKSHOP 4 :

Tasks 4

- This workshop includes marked tasks that comprise 15% of your final mark in this module.
- You need to read the examples in Lecture #4 and Lecture #4 exercise to complete the tasks.

TASK 4.1: Download the adult_WS4 dataset. Apply K-Means and Hierarchical clustering to three optional columns in the dataset. Find the optimum number of clusters for both clustering methods (10%).

NOTE: You should comment on your code wherever necessary and briefly explain what the code is doing

[ ]:

```

```

Reading in the data

[288]:

```
adult_WS4_df = pd.read_csv('adult_WS4.csv')

adult_WS4_df.head(40)
```

```
<IPython.core.display.HTML object>
```

[288]:

[ ]:

```

```

Select optional columns

[292]:

```
data_copy1 = adult_WS4_df.copy() #creating a copy of the data set.

optional_columns = ["age", "fnlwgt", "hours-per-week"] ## selecting optional␣
 ↪features.

optional_df = data_copy1[optional_columns].dropna()  ## creating a dataframe␣
 ↪that has only the optional columns.

optional_df
```

```
<IPython.core.display.HTML object>
```

[292]:

[ ]:

```

```

0.7.1 K-MEANS

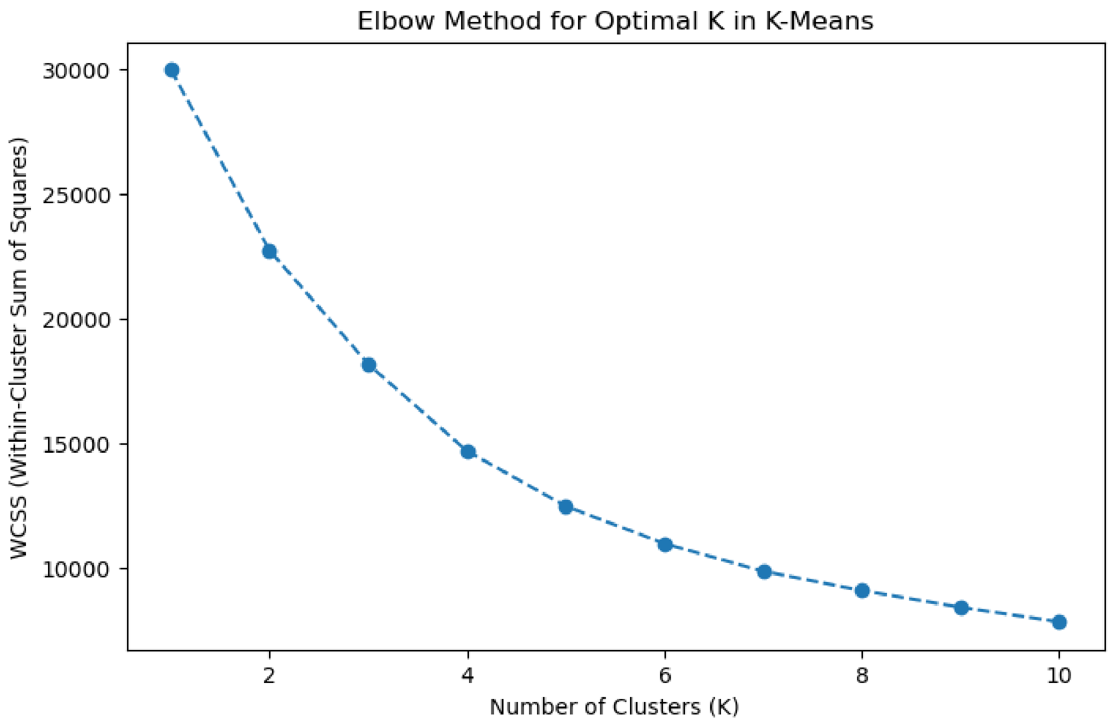
[296]:

```
# Standardize the data
scaler = StandardScaler()
df_scaled = scaler.fit_transform(optional_df)
```

[298]:

```
# Find the optimal number of clusters using the Elbow Method , this is help us␣
 ↪to plot the elbow chart.
wcss = []  # Within-cluster sum of squares
K_range = range(1, 11)

for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(df_scaled)
    wcss.append(kmeans.inertia_)
```

[299]:
```
# Plot the Elbow Method
plt.figure(figsize=(8, 5))
plt.plot(K_range, wcss, marker='o', linestyle='--')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('WCSS (Within-Cluster Sum of Squares)')
plt.title('Elbow Method for Optimal K in K-Means')
plt.show()
```

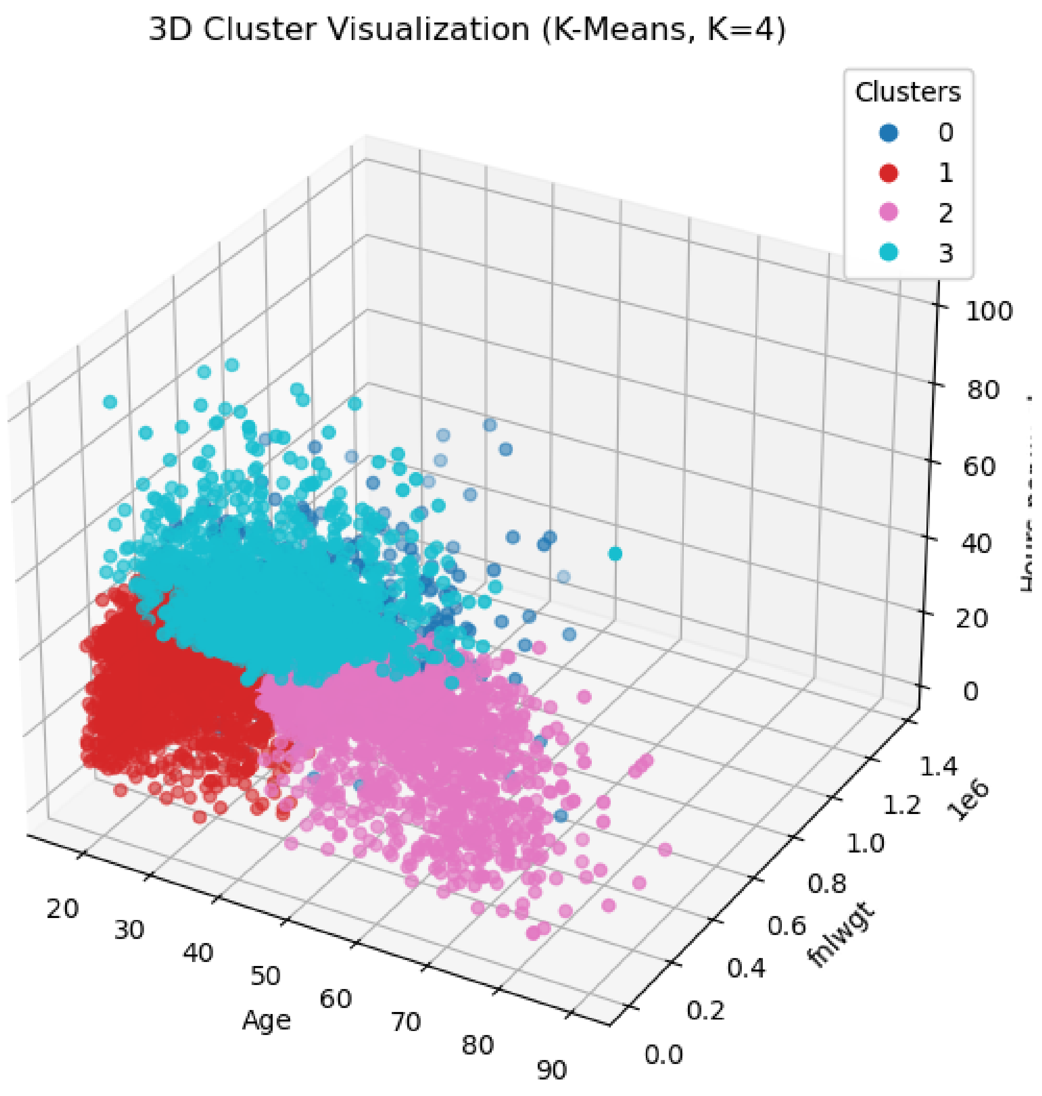


The above elbow plot helps us to determine the number of clusters that would work perfectly with this unsupervised machine aglo. From above k=4 is the best as that's where the plot start decreasing.

[301]:
```
# Apply K-Means with K = 4
kmeans = KMeans(n_clusters=4, random_state=42, n_init=10)
optional_df["Cluster"] = kmeans.fit_predict(df_scaled)
```

[302]:
```
# Create a 3D scatter plot
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')

# Scatter plot with clusters
```

```python
scatter = ax.scatter(
    optional_df["age"],
    optional_df["fnlwgt"],
    optional_df["hours-per-week"],
    c=optional_df["Cluster"], cmap="tab10", marker="o"
)

# Labels and title
ax.set_xlabel("Age")
ax.set_ylabel("fnlwgt")
ax.set_zlabel("Hours-per-week")
ax.set_title("3D Cluster Visualization (K-Means, K=4)")

# Add legend
legend1 = ax.legend(*scatter.legend_elements(), title="Clusters")
ax.add_artist(legend1)

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



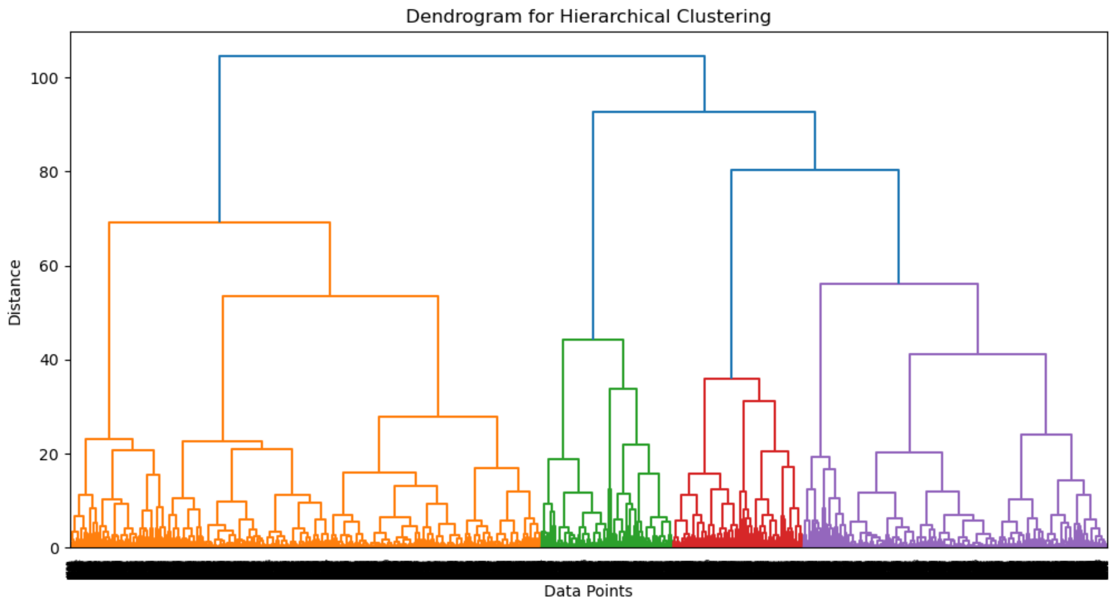
0.7.2 Concldusion on K-Mean :

The clusters are relatively well-separated in this 3D space, indicating that the clustering algorithm has done a decent job of grouping similar data points together. Cluster 3 and Cluster 2 are more spread out and overlap to some extent, particularly along the "fnlwgt" and "age" axes. Cluster 0 and Cluster 1 are more densely packed and seem to occupy distinct regions, with less overlap compared to Clusters 2 and 3.

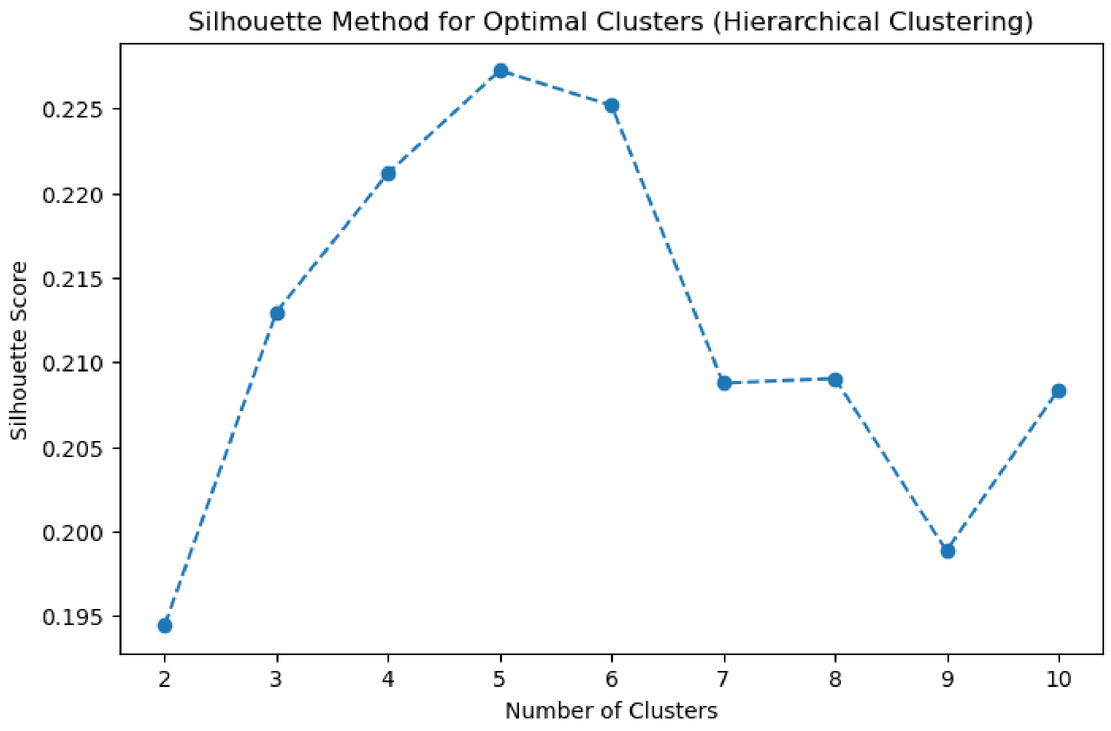
```
[ ]:
```

0.7.3 Hierarchical Clustering

[311]:
```
plt.figure(figsize=(12, 6))
linkage_matrix = linkage(df_scaled, method="ward")
dendrogram(linkage_matrix)
plt.xlabel("Data Points")
plt.ylabel("Distance")
plt.title("Dendrogram for Hierarchical Clustering")
plt.show()
```

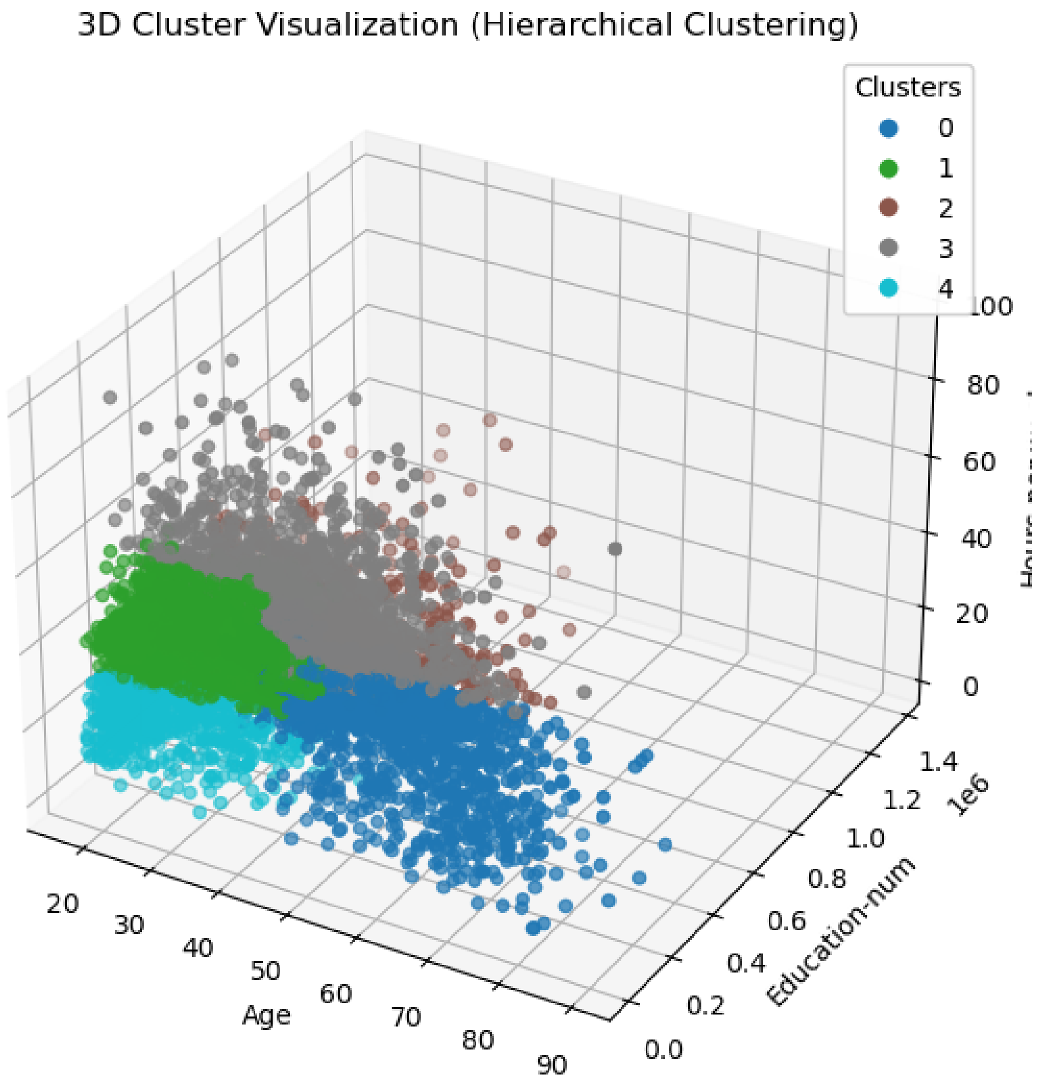


[312]:
```
# Define the range of clusters to test
cluster_range = range(2, 11)  # We start from 2 because silhouette score is␣
 ↪undefined for 1 cluster
silhouette_scores = []

# Compute silhouette scores for each number of clusters
for k in cluster_range:
    hierarchical = AgglomerativeClustering(n_clusters=k, linkage="ward")
    cluster_labels = hierarchical.fit_predict(df_scaled)
    silhouette_avg = silhouette_score(df_scaled, cluster_labels)
    silhouette_scores.append(silhouette_avg)

# Plot Silhouette Scores
plt.figure(figsize=(8, 5))
plt.plot(cluster_range, silhouette_scores, marker="o", linestyle="--")
plt.xlabel("Number of Clusters")
plt.ylabel("Silhouette Score")
plt.title("Silhouette Method for Optimal Clusters (Hierarchical Clustering)")
plt.show()
```



[313]:
```
# Find the optimal K (highest silhouette score)
optimal_k = cluster_range[silhouette_scores.index(max(silhouette_scores))]
print(f"Optimal number of clusters based on Silhouette Score: {optimal_k}")

### --- Step 3: Apply Hierarchical Clustering ---
hierarchical = AgglomerativeClustering(n_clusters=optimal_k, linkage="ward")
optional_df["Hierarchical_Cluster"] = hierarchical.fit_predict(df_scaled)

### --- Step 4: 3D Scatter Plot of Clusters ---
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection="3d")

# Scatter plot with cluster colors
scatter = ax.scatter(
    optional_df["age"],
    optional_df["fnlwgt"],
    optional_df["hours-per-week"],
    c=optional_df["Hierarchical_Cluster"],
    cmap="tab10",
    marker="o",
)

# Labels and title
ax.set_xlabel("Age")
ax.set_ylabel("Education-num")
ax.set_zlabel("Hours-per-week")
ax.set_title("3D Cluster Visualization (Hierarchical Clustering)")

# Add legend
legend1 = ax.legend(*scatter.legend_elements(), title="Clusters")
ax.add_artist(legend1)

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

```
Optimal number of clusters based on Silhouette Score: 5
```



[ ]:

```
```

0.7.4 Task 4.2 :

Apply the PCA method to the dataset and extract the first two principal components (n_components=2). Plot the scatter plot of the dataset's first two components for the two classes of the income column (5%).

NOTE 1: You should comment on your code wherever necessary and briefly explain what the code is doing.

NOTE 2: You need to encode the categorical columns, normalise the dataset, and remove the income column before applying the PCA method.

HINT: See the examples in the last three slides in Lecture #4 or the Lecture #4 exercise notebook

[320]:

```
Data_copy2 = adult_WS4_df.copy()
```

[322]:

```
X = Data_copy2.drop('income' , axis = 1)
X.head(20)
```

```
<IPython.core.display.HTML object>
```

[322]:

[323]:

```
class Preprocessing:
    """
     This class is use to preprocessing of the data, initial preprocess with␣
 ↪methods for preprocessing.
    """

    def __init__(self , data):
        self.data = data

    def filling_in_values(self, col):
        return self.data[col].fillna(self.data[col].mode()[0], inplace=True)

    def norminal_encoding(self):
        # Define categorical columns (both nominal and ordinal)
        categorical_columns = [
            'workclass', 'education', 'marital-status', 'occupation',
            'relationship', 'race', 'sex', 'native-country'
        ]

        encoder = OrdinalEncoder()
        # Apply encoding to categorical columns
        self.data[categorical_columns] = encoder.fit_transform(self.
↪data[categorical_columns])

        return self.data

    def standardizing_data(self):

        X = self.data

        # Initialize the MinMaxScaler
        scaler = MinMaxScaler()

        return scaler.fit_transform(X)
```

```
[ ]:
```

Create an object of the class

```
[328]: preprocessingobject = Preprocessing(X)
```

Filling in missing values

```
[331]: preprocessingobject.filling_in_values('workclass')
       preprocessingobject.filling_in_values('occupation')
       preprocessingobject.filling_in_values('native-country')
```

```
[ ]:
```

```
[334]: preprocessingobject.norminal_encoding()
```

```
<IPython.core.display.HTML object>

[334]:
```

Normalising the dataset

```
[337]: x_scaled = preprocessingobject.standardizing_data()
       x_scaled
```

```
[337]: array([[0.16438356, 0.5       , 0.15082783, …, 0.        , 0.39795918,
                0.95      ],
               [0.26027397, 0.5       , 0.19816669, …, 0.        , 0.44897959,
                0.95      ],
               [0.10958904, 0.        , 0.06545702, …, 0.        , 0.24489796,
                0.95      ],
               …,
               [0.17808219, 0.5       , 0.24406519, …, 0.        , 0.39795918,
 0.95      ],
[0.32876712, 0.75      , 0.09043864, …, 0.        , 0.39795918,
 0.95      ],
[0.05479452, 0.5       , 0.17725055, …, 0.        , 0.39795918,
 0.95      ]])
```

[ ]:

0.7.5 Principal Component Analysis

[341]:
```python
data = preprocessingobject.norminal_encoding()
```

[343]:
```python
pca = PCA(n_components=2)

X_pca = pca.fit_transform(x_scaled)

X_df = pd.DataFrame(data = X_pca, columns = ['Principal-Component-1',␣
↪'Principal-Component-2'])
```

[345]:
```python
X_df
```

```
<IPython.core.display.HTML object>
```

[345]:

[ ]:

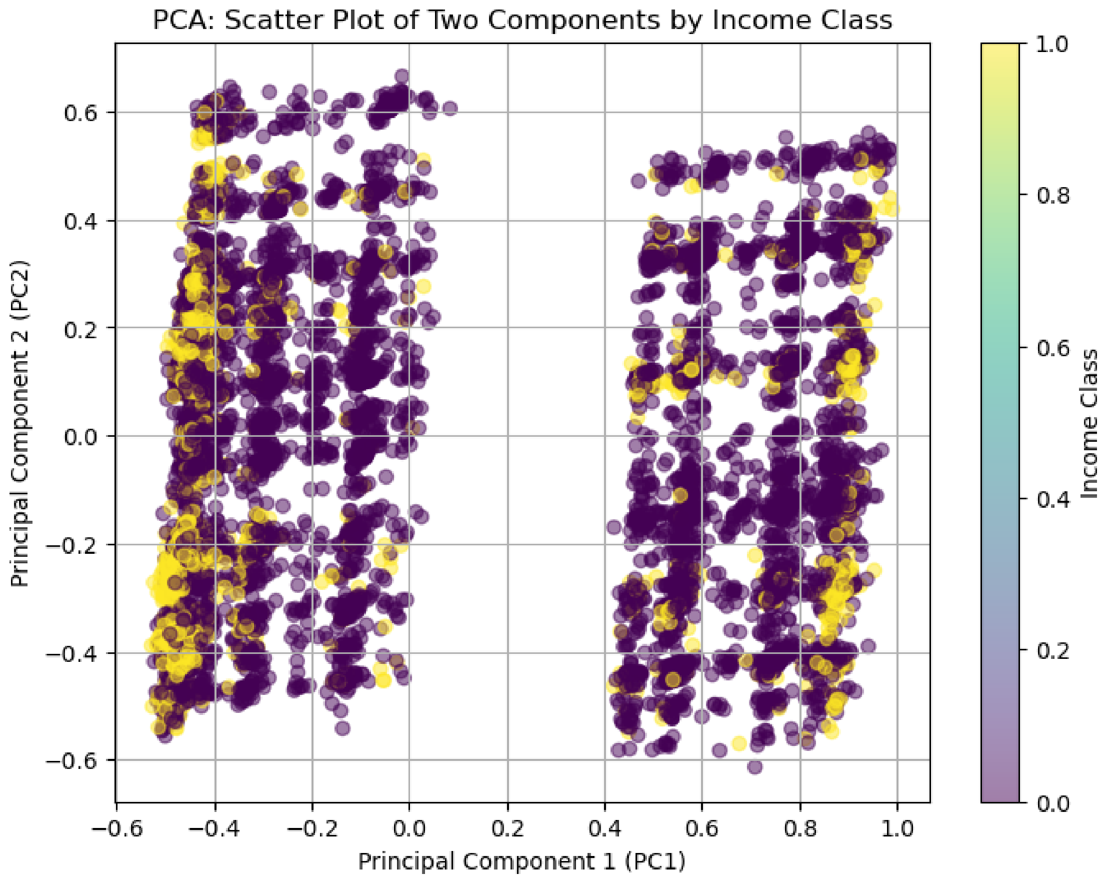
Scatter Plot

[349]:
```python
y = Data_copy2['income']
```

[351]:
```python
le = LabelEncoder()
y_encoded = le.fit_transform(y)
```

[353]:
```python
# Create a scatter plot
plt.figure(figsize=(8, 6))
scatter = plt.scatter(X_df['Principal-Component-1'],␣
↪X_df['Principal-Component-2'], c=y_encoded, cmap='viridis', alpha=0.5)

# Add labels and title
plt.xlabel('Principal Component 1 (PC1)')
plt.ylabel('Principal Component 2 (PC2)')
plt.title('PCA: Scatter Plot of Two Components by Income Class')

# Add a colorbar to show the class mapping
plt.colorbar(scatter, label='Income Class')
```

```
# Add a grid for better readability
plt.grid(True)
plt.show()
```



```
[ ]:
```

0.7.6 Concludsion on PCA

The above scatter plots shows the first two principal components (PC1 and PC2) from a Principal Component Analysis (PCA). There data points shows a clear two distinct vertical clusters, this indicates that the variance could create these groupings. The color gradient (from purple to yellow) represents different income classes. The distribution suggests that the income classes are spread across both clusters rather than being concentrated in a single group.

```
[ ]:
```

0.8 WORKSHOP 5 :

TASK 5 : 5.1: Download the 'Portugal_online_retail', 'Sweden_online_retail, and 'UK_online_retail' datasets from Canvas. Apply the apriori algorithm to all datasets using three different confidence levels. Select one confidence level for each dataset that you think works better. Determine the first three most important rules for each dataset using the selected confidence level and report them in the report cell. Explain what each rule means (Completing the report cell is required) (10%).

NOTE: You should comment on your code

```
[ ]:
```

Reading in the data :

```
[364]: Portugal_online_retail_df = pd.read_csv('Portugal_online_retail.csv')
```

```
[366]: Sweden_online_retail_df = pd.read_csv('Sweden_online_retail.csv')
```

```
[368]: UK_online_retail_df = pd.read_csv('UK_online_retail.csv', dtype={'InvoiceNo':␣
        ↪str})
```

```
[369]: UK_online_retail_df.columns
```

```
[369]: Index(['InvoiceNo', '*Boombox Ipod Classic', '*USB Office Mirror Ball',
              '10 COLOUR SPACEBOY PEN', '12 COLOURED PARTY BALLOONS',
              '12 DAISY PEGS IN WOOD BOX', '12 EGG HOUSE PAINTED WOOD',
              '12 HANGING EGGS HAND PAINTED', '12 IVORY ROSE PEG PLACE SETTINGS',
              '12 MESSAGE CARDS WITH ENVELOPES',
              …
              'wrongly coded 20713', 'wrongly coded 23343', 'wrongly coded-23343',
              'wrongly marked', 'wrongly marked 23343', 'wrongly marked carton 22804',
              'wrongly marked. 23343 in box', 'wrongly sold (22719) barcode',
              'wrongly sold as sets', 'wrongly sold sets'],
             dtype='object', length=4176)
```

```
[370]: # Function to preprocess dataset into binary format
       def preprocess_retail_data(filepath, sample_size=None):
           """
           Preprocess retail dataset into a binary transaction matrix with optional␣
        ↪sampling.

           Parameters:
               filepath (str): Path to the CSV file
               sample_size (int, optional): Number of rows to sample (None for full␣
        ↪dataset)

           Returns:
               pd.DataFrame: Binary DataFrame suitable for Apriori
           """
           df = pd.read_csv(filepath,  low_memory=False)
           print(f"Original Data Shape: {df.shape}")
```

```
    if sample_size and sample_size < df.shape[0]:
        df = df.sample(sample_size, random_state=42)
        print(f"Sampled Data Shape: {df.shape}")

    binary_df = df.drop('InvoiceNo', axis=1).astype(bool).astype(int)
    print(f"Binary Data Shape: {binary_df.shape}")
    return binary_df

# Function to apply Apriori and generate rules
def apply_apriori(df, min_support, confidence_levels):
    """
    Apply Apriori algorithm and generate association rules with low_memory␣
  ↪option.

    Parameters:
        df (pd.DataFrame): Binary transaction DataFrame
        min_support (float): Minimum support threshold
        confidence_levels (list): List of confidence levels to test

    Returns:
        tuple: Frequent itemsets and rules dictionary
    """
    print(f"Applying Apriori with min_support={min_support}")
    frequent_itemsets = apriori(df, min_support=min_support, use_colnames=True,␣
  ↪low_memory=True)
    print(f"Number of Frequent Itemsets: {len(frequent_itemsets)}")
    print("Frequent Itemsets Sample:")
    print(frequent_itemsets.sort_values(by='support', ascending=False).head())

    rules_dict = {}
    for conf in confidence_levels:
        rules = association_rules(frequent_itemsets, metric="confidence",␣
  ↪min_threshold=conf)
        print(f"Number of Rules at {conf*100}% Confidence: {len(rules)}")
        rules_dict[conf] = rules

    return frequent_itemsets, rules_dict

# Function to filter top 3 rules for reporting
def get_top_rules(rules, top_n=3):
    """
    Filter top N rules based on confidence and lift.

    Parameters:
        rules (pd.DataFrame): Association rules DataFrame
        top_n (int): Number of top rules to return
    Returns:
        pd.DataFrame: Top N rules
    """
    if len(rules) == 0:
        print("No rules found for this confidence level.")
        return pd.DataFrame()
    return rules.sort_values(by=['confidence', 'lift'], ascending=False).
↪head(top_n)

# Main processing function
def process_retail_dataset(dataset_name, filepath, min_support,␣
↪confidence_levels, selected_conf, sample_size=None):
    """
    Process a retail dataset and generate association rules report.

    Parameters:
        dataset_name (str): Name of the dataset
        filepath (str): Path to the CSV file
        min_support (float): Minimum support threshold
        confidence_levels (list): List of confidence levels to test
        selected_conf (float): Selected confidence level for reporting
        sample_size (int, optional): Number of rows to sample

    Returns:
        list: Report entries for this dataset
    """
    binary_df = preprocess_retail_data(filepath, sample_size)
    frequent_itemsets, rules_dict = apply_apriori(binary_df, min_support,␣
↪confidence_levels)

    rules = rules_dict[selected_conf]
    top_rules = get_top_rules(rules)

    report = []
    if not top_rules.empty:
        for _, row in top_rules.iterrows():
            antecedents = ', '.join(list(row['antecedents']))
            consequents = ', '.join(list(row['consequents']))
            support = row['support']
            confidence = row['confidence']

            if dataset_name == 'Portugal_online_retail':
                explanation = f"In Portugal, if {antecedents} is bought,␣
↪{consequents} is likely, suggesting a preference for coordinated purchases."
            elif dataset_name == 'Sweden_online_retail':
                explanation = f"In Sweden, buying {antecedents} implies␣
↪{consequents}, indicating a cultural or seasonal buying pattern."
            else:  # UK_online_retail
                explanation = f"In the UK, purchasing {antecedents} leads to␣
↪{consequents}, hinting at popular item combinations."

            report.append({
                'Dataset': dataset_name,
                'Selected Confidence Level': f"{selected_conf*100}%",
                'Rule': f"{{{antecedents}}} → {{{consequents}}}",
                'Support': f"{support:.2%}",
                'Confidence': f"{confidence:.2%}",
                'Explanation': explanation
            })
    else:
        report.append({
            'Dataset': dataset_name,
            'Selected Confidence Level': f"{selected_conf*100}%",
            'Rule': "No rules found",
            'Support': "N/A",
            'Confidence': "N/A",
            'Explanation': f"No rules met the {selected_conf*100}% confidence␣
↪threshold with min_support={min_support}."
        })

    return report

# Define datasets and parameters
datasets = {
    'Portugal_online_retail': 'Portugal_online_retail.csv',
    'Sweden_online_retail': 'Sweden_online_retail.csv',
    'UK_online_retail': 'UK_online_retail.csv'
}

# Adjusted parameters for each dataset
params = {
    'Portugal_online_retail': {
        'min_support': 0.1,  # 10% of 58   6 transactions
        'confidence_levels': [0.2, 0.3, 0.4],
        'selected_conf': 0.3,  # Moderate for small dataset
        'sample_size': None  # Full dataset (small enough)
    },
    'Sweden_online_retail': {
        'min_support': 0.05,  # Assuming moderate size (e.g., 500 rows, 5%   25␣
↪transactions)
        'confidence_levels': [0.2, 0.3, 0.4],
        'selected_conf': 0.3,  # Moderate threshold
        'sample_size': None  # Adjust if Sweden is large
    },
    'UK_online_retail': {
        'min_support': 0.03,  # 3% of 5,000 = 150 transactions
        'confidence_levels': [0.2, 0.3, 0.4],
        'selected_conf': 0.3,  # Lowered to ensure rules
        'sample_size': 5000  # Sampled to avoid memory issues
    }
}

# Process all datasets and collect report
report = []
for dataset_name, filepath in datasets.items():
    dataset_params = params[dataset_name]
    dataset_report = process_retail_dataset(
        dataset_name,
        filepath,
        dataset_params['min_support'],
        dataset_params['confidence_levels'],
        dataset_params['selected_conf'],
        dataset_params['sample_size']
    )
    report.extend(dataset_report)

# Display report as a DataFrame
report_df = pd.DataFrame(report)
print("Association Rules Report:")
print(report_df.to_string(index=False))
```

```
Original Data Shape: (58, 714)
Binary Data Shape: (58, 713)
Applying Apriori with min_support=0.1
Number of Frequent Itemsets: 110
Frequent Itemsets Sample:
     support                            itemsets
25  0.517241                           (POSTAGE)
30  0.241379  (RETROSPOT TEA SET CERAMIC 11 PC)
14  0.241379          (LUNCH BAG RED RETROSPOT)
1   0.206897      (BAKING SET 9 PIECE RETROSPOT)
11  0.206897               (LUNCH BAG CARS BLUE)
Number of Rules at 20.0% Confidence: 319
Number of Rules at 30.0% Confidence: 317
Number of Rules at 40.0% Confidence: 317
Original Data Shape: (36, 262)
Binary Data Shape: (36, 261)
Applying Apriori with min_support=0.05
Number of Frequent Itemsets: 2856
```

```
Frequent Itemsets Sample:
      support                                  itemsets
50   0.611111                                 (POSTAGE)
63   0.194444        (SET OF 3 CAKE TINS PANTRY DESIGN)
33   0.166667                  (MINI PAINT SET VINTAGE)
53   0.138889           (RED TOADSTOOL LED NIGHT LIGHT)
332  0.138889  (RED TOADSTOOL LED NIGHT LIGHT, POSTAGE)
Number of Rules at 20.0% Confidence: 104538
Number of Rules at 30.0% Confidence: 103458
Number of Rules at 40.0% Confidence: 102926
Original Data Shape: (18667, 4176)
Sampled Data Shape: (5000, 4176)
Binary Data Shape: (5000, 4175)
Applying Apriori with min_support=0.03
Number of Frequent Itemsets: 138
Frequent Itemsets Sample:
     support                             itemsets
122   0.1184  (WHITE HANGING HEART T-LIGHT HOLDER)
52    0.1004             (JUMBO BAG RED RETROSPOT)
98    0.0934            (REGENCY CAKESTAND 3 TIER)
87    0.0850                       (PARTY BUNTING)
70    0.0762             (LUNCH BAG RED RETROSPOT)
Number of Rules at 20.0% Confidence: 22
Number of Rules at 30.0% Confidence: 22
Number of Rules at 40.0% Confidence: 18
Association Rules Report:
               Dataset Selected Confidence Level
Rule Support Confidence
Explanation
Portugal_online_retail                    30.0%
{SCANDINAVIAN PAISLEY PICNIC BAG} → {PINK VINTAGE PAISLEY PICNIC BAG}  12.07%
100.00%                                 In Portugal, if SCANDINAVIAN PAISLEY
PICNIC BAG is bought, PINK VINTAGE PAISLEY PICNIC BAG is likely, suggesting a
preference for coordinated purchases.
Portugal_online_retail                    30.0%        {LUNCH BAG RED
RETROSPOT, SCANDINAVIAN PAISLEY PICNIC BAG} → {PINK VINTAGE PAISLEY PICNIC BAG}
10.34%    100.00%        In Portugal, if LUNCH BAG RED RETROSPOT, SCANDINAVIAN
PAISLEY PICNIC BAG is bought, PINK VINTAGE PAISLEY PICNIC BAG is likely,
suggesting a preference for coordinated purchases.
Portugal_online_retail                    30.0% {PLASTERS IN TIN CIRCUS PARADE,
PLASTERS IN TIN VINTAGE PAISLEY} → {PLASTERS IN TIN WOODLAND ANIMALS}  10.34%
100.00% In Portugal, if PLASTERS IN TIN CIRCUS PARADE, PLASTERS IN TIN VINTAGE
PAISLEY is bought, PLASTERS IN TIN WOODLAND ANIMALS is likely, suggesting a
preference for coordinated purchases.
  Sweden_online_retail                    30.0%
{PACK OF 72 SKULL CAKE CASES} → {12 PENCILS SMALL TUBE SKULL}   5.56%    100.00%
In Sweden, buying PACK OF 72 SKULL CAKE CASES implies 12 PENCILS SMALL TUBE
SKULL, indicating a cultural or seasonal buying pattern.
```

```
  Sweden_online_retail                      30.0%
{12 PENCILS SMALL TUBE SKULL} → {PACK OF 72 SKULL CAKE CASES}   5.56%    100.00%
In Sweden, buying 12 PENCILS SMALL TUBE SKULL implies PACK OF 72 SKULL CAKE
CASES, indicating a cultural or seasonal buying pattern.
  Sweden_online_retail                      30.0%
{ASSORTED BOTTLE TOP  MAGNETS} → {36 DOILIES DOLLY GIRL}   5.56%    100.00%
In Sweden, buying ASSORTED BOTTLE TOP  MAGNETS implies 36 DOILIES DOLLY GIRL,
indicating a cultural or seasonal buying pattern.
      UK_online_retail                      30.0%
{PINK REGENCY TEACUP AND SAUCER} → {GREEN REGENCY TEACUP AND SAUCER}   3.26%
85.79%                                                     In the UK, purchasing
PINK REGENCY TEACUP AND SAUCER leads to GREEN REGENCY TEACUP AND SAUCER, hinting
at popular item combinations.
      UK_online_retail                      30.0%
{PINK REGENCY TEACUP AND SAUCER} → {ROSES REGENCY TEACUP AND SAUCER}   3.02%
79.47%                                                     In the UK, purchasing
PINK REGENCY TEACUP AND SAUCER leads to ROSES REGENCY TEACUP AND SAUCER, hinting
at popular item combinations.
      UK_online_retail                      30.0%
{GREEN REGENCY TEACUP AND SAUCER} → {ROSES REGENCY TEACUP AND SAUCER}   3.80%
76.31%                                                    In the UK, purchasing
GREEN REGENCY TEACUP AND SAUCER leads to ROSES REGENCY TEACUP AND SAUCER,
hinting at popular item combinations.
```

[ ]:

```

```

0.9 WORKSHOP 6

Task 6.1: Download the 'Tweets' dataset from Canvas. Classify the sentiments in the dataset using six classifiers and calculate all evaluation metrics (10%). - NOTE: If the running time is too long, you can reduce the number of samples. - NOTE: You should comment on your code and explain what each part is doing - NOTE: You should improve the model's performance as much as possible

Reading in the data :

[378]:

```
tweets_df = pd.read_csv('Tweets.csv')
```

[379]:

```
tweets_df
```

```
<IPython.core.display.HTML object>
```

[379]:

EXPLORATING AND PREPROCESSING TEXT

1. LOWERCASING

[385]:
```
tweets_df['lower'] = tweets_df['tweet'].apply(lambda x: " ".join(x.lower() for␣
↪x in x.split()))
```

[386]:
```
tweets_df
```

```
<IPython.core.display.HTML object>
```

[386]:

2. Punctuation Removal

[388]:
```
tweets_df['PunctuationRemoval1'] = tweets_df.lower.apply(lambda x:''.join(i for␣
↪i in x if i not in string.punctuation))
```

[389]:
```
tweets_df
```

```
<IPython.core.display.HTML object>
```

[389]:

3. Stop Words Removal

[395]:
```
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\User\AppData\Roaming\nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

[395]:
```
True
```

[396]:
```
allstopwords = stopwords.words('english')
```

[397]:
```
tweets_df['StopWordsRemoval'] = tweets_df.PunctuationRemoval1.apply(lambda x: "␣
↪".join(i for i in x.split() if i not in allstopwords))
```

[398]:
```
tweets_df
```

```
<IPython.core.display.HTML object>
```

[398]:

4. Spelling Correction

[404]:
```
# tweets_df['SpellingCorrection'] = tweets_df['StopWordsRemoval'].apply(lambda␣
↪x: str(TextBlob(x).correct()))
```

[ ]:
```

```

5. Stemming

[408]:
```
st = PorterStemmer()
```

[410]:
```
tweets_df['stem'] = tweets_df['StopWordsRemoval'].apply(lambda x:" ".join([st.
↪stem(word) for word in x.split()]))
```

[411]:
```
tweets_df
```

```
<IPython.core.display.HTML object>
```

[411]:

Lemmatization

[415]:
```
import textblob
from textblob import Word
```

[416]:
```
tweets_df['cleanText'] = tweets_df['stem'].apply(lambda x: " ".join([Word(word).
↪lemmatize() for word in x.split()]))
```

[417]:
```
tweets_df
```

```
<IPython.core.display.HTML object>
```

[417]:

[ ]:
```

```

0.9.1 Word cloud for both + and -

[423]:
```
# split df - positive and negative sentiment:
positive = tweets_df[tweets_df['sentiment'] == 1]
negative = tweets_df[tweets_df['sentiment'] == -1]
```

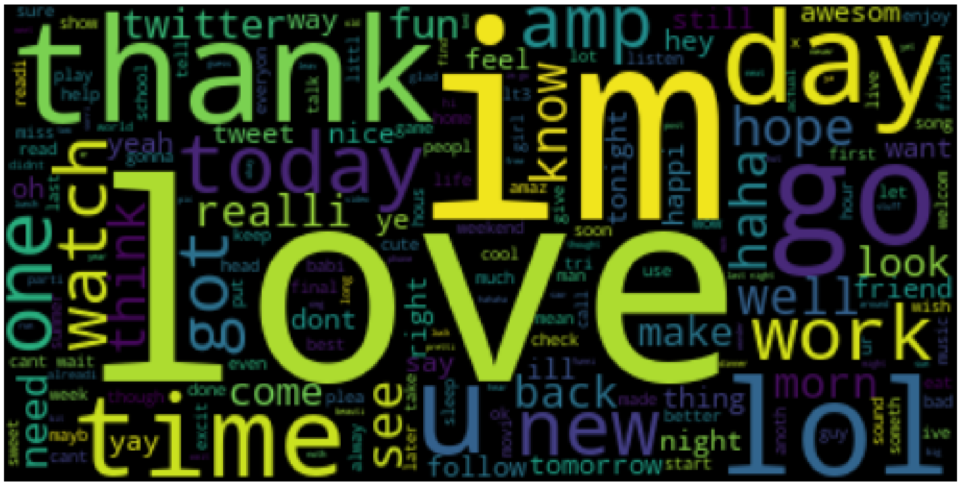
[424]:
```
pip install wordcloud
```

```
Requirement already satisfied: wordcloud in c:\users\user\anaconda3\lib\site-
packages (1.9.4)
Requirement already satisfied: numpy>=1.6.1 in c:\users\user\anaconda3\lib\site-
packages (from wordcloud) (1.26.4)
Requirement already satisfied: pillow in c:\users\user\anaconda3\lib\site-
packages (from wordcloud) (10.4.0)
Requirement already satisfied: matplotlib in c:\users\user\anaconda3\lib\site-
packages (from wordcloud) (3.8.4)Note: you may need to restart the kernel to use
updated packages.

Requirement already satisfied: contourpy>=1.0.1 in
c:\users\user\anaconda3\lib\site-packages (from matplotlib->wordcloud) (1.2.0)
```

```
Requirement already satisfied: cycler>=0.10 in c:\users\user\anaconda3\lib\site-
packages (from matplotlib->wordcloud) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in
c:\users\user\anaconda3\lib\site-packages (from matplotlib->wordcloud) (4.51.0)
Requirement already satisfied: kiwisolver>=1.3.1 in
c:\users\user\anaconda3\lib\site-packages (from matplotlib->wordcloud) (1.4.4)
```

[426]:
```python
from wordcloud import WordCloud
from wordcloud import STOPWORDS
```

[427]:
```python
# Word cloud positive
stopwords = set(STOPWORDS)
stopwords.update(["br", "href","good","great"])
## good and great removed because they were included in negative sentiment
pos = " ".join(review for review in positive.cleanText)
wordcloud2 = WordCloud(stopwords=stopwords).generate(pos)
plt.imshow(wordcloud2, interpolation='bilinear')
plt.axis("off")
plt.show()
```



[431]:
```python
# word cloud negative

# neg = " ".join(str(review) for review in negative.cleanText)
# wordcloud3 = WordCloud(stopwords=stopwords).generate(neg)
# plt.imshow(wordcloud3, interpolation='bilinear')
# plt.axis("off")
# plt.savefig('wordcloud33.png')
# plt.show()
```

Model building

[434]:
```
# Extracting input and output
X = tweets_df['cleanText']
y = tweets_df['sentiment']
```

[436]:
```
# count vectorizer:
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(token_pattern=r'\b\w+\b')
```

[438]:
```
X = vectorizer.fit_transform(X)
```

[440]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,␣
 ↪random_state=42)
```

[442]:
```
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
SVM = svm.SVC()
RF = RandomForestClassifier()
KNN = KNeighborsClassifier()
DT=DecisionTreeClassifier()
NB = GaussianNB()
LR = LogisticRegression()
```

[444]:
```
# Step 2: training the models
SVM.fit(X_train, y_train)
RF.fit(X_train, y_train)
KNN.fit(X_train, y_train)
DT.fit(X_train, y_train)
LR.fit(X_train,y_train)
NB.fit(X_train.toarray(),y_train)
```

[444]:
```
GaussianNB()
```

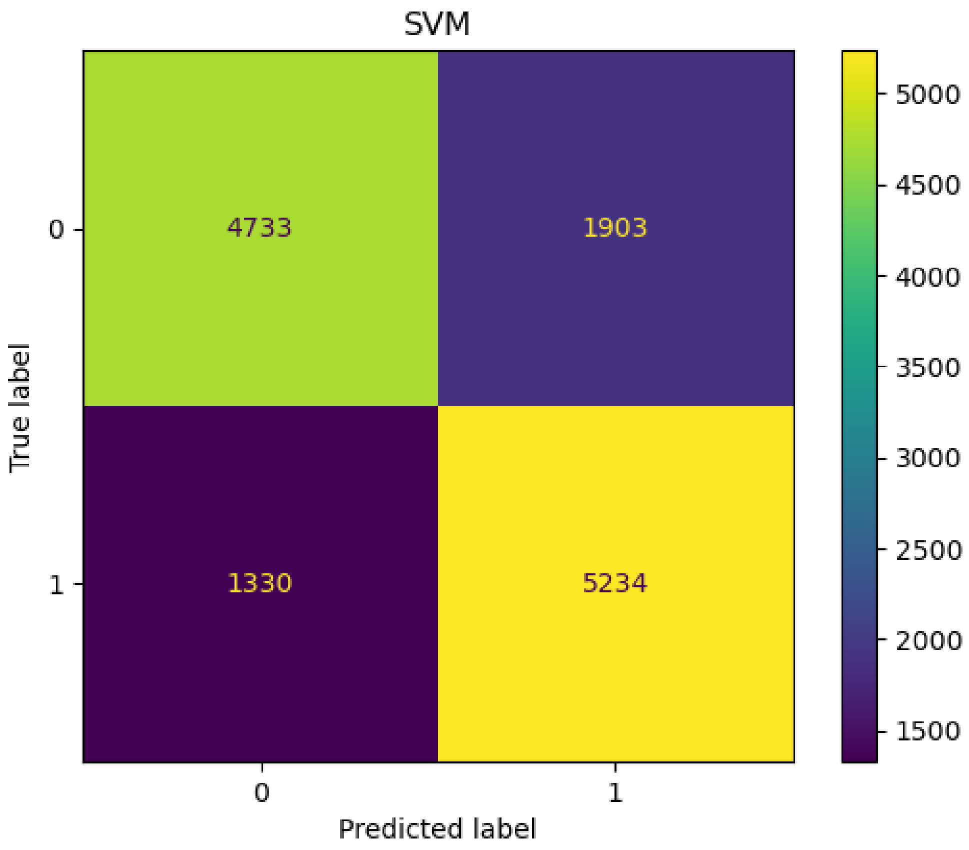
[445]:
```
#Step 3: prediction
y_pred1=SVM.predict(X_test)
y_pred2=RF.predict(X_test)
y_pred3=KNN.predict(X_test)
y_pred4=DT.predict(X_test)
y_pred5=LR.predict(X_test)
y_pred6=NB.predict(X_test.toarray())
```

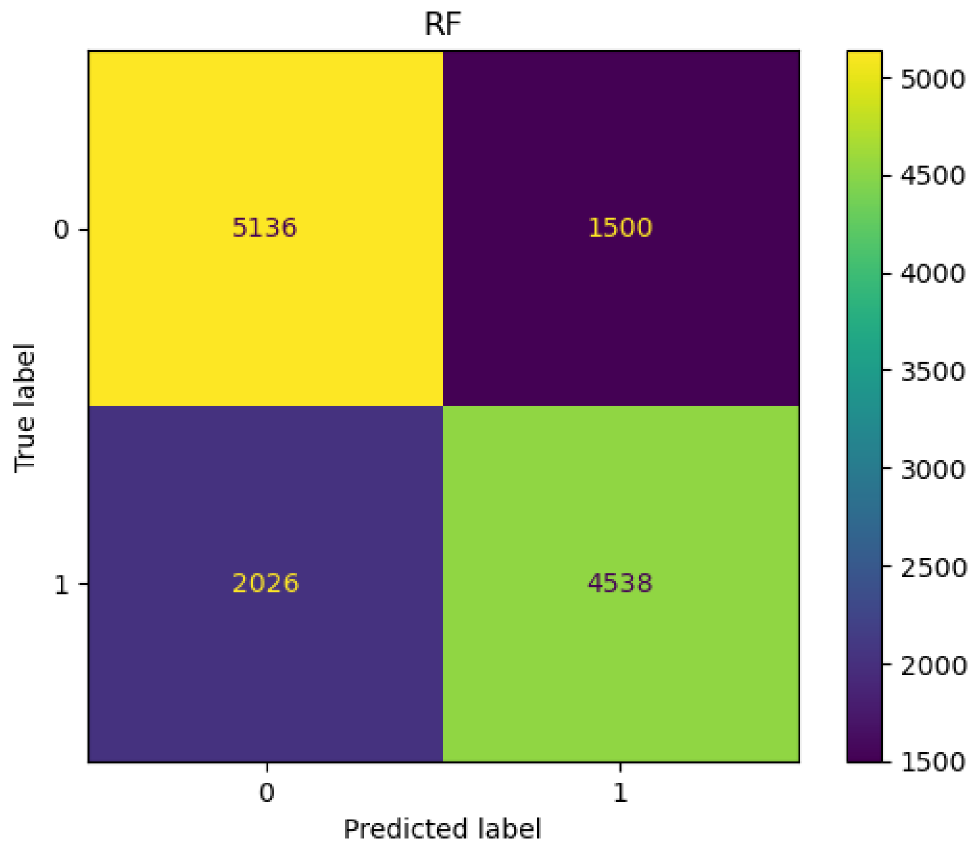
[446]:
```
# This function takes the confusion matrix (cm) from the cell above and␣
 ↪produces all evaluation matrix
def confusion_metrics (conf_matrix):
    TP = conf_matrix[1][1]
    TN = conf_matrix[0][0]
    FP = conf_matrix[0][1]
    FN = conf_matrix[1][0]
    print('True Positives:', TP)
    print('True Negatives:', TN)
    print('False Positives:', FP)
    print('False Negatives:', FN)

    # calculate accuracy
    conf_accuracy = (float (TP+TN) / float(TP + TN + FP + FN))

    # calculate mis-classification
    conf_misclassification = 1- conf_accuracy

    # calculate the sensitivity
    conf_sensitivity = (TP / float(TP + FN))
    # calculate the specificity
    conf_specificity = (TN / float(TN + FP))

    # calculate precision
    conf_precision = (TN / float(TN + FP))
    # calculate f_1 score
    conf_f1 = 2 * ((conf_precision * conf_sensitivity) / (conf_precision +␣
↪conf_sensitivity))
    print('-'*50)
    print(f'Accuracy: {round(conf_accuracy,2)}')
    print(f'Mis-Classification: {round(conf_misclassification,2)}')
    print(f'Sensitivity: {round(conf_sensitivity,2)}')
    print(f'Specificity: {round(conf_specificity,2)}')
    print(f'Precision: {round(conf_precision,2)}')
    print(f'f_1 Score: {round(conf_f1,2)}')
```

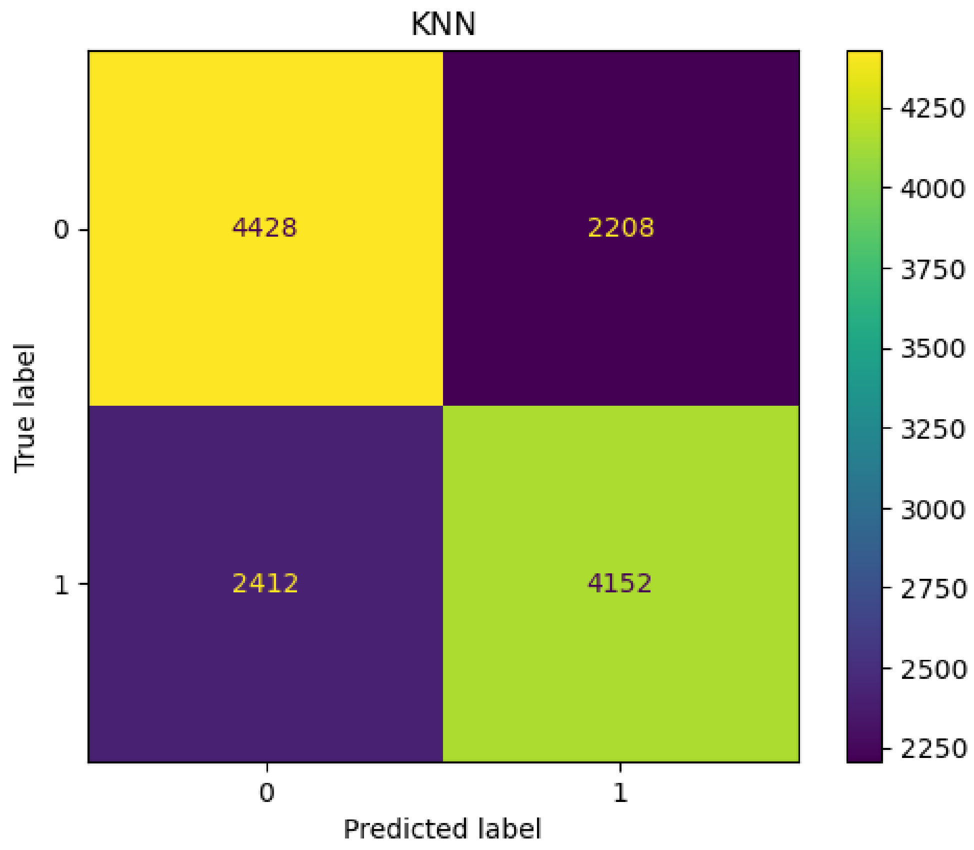
```
[447]: # Creating the confusion matrics for all classifiers' predictions
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

cm1 = confusion_matrix(y_test, y_pred1, labels=SVM.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm1,display_labels=SVM.classes_)
disp.plot()
plt.title("SVM")

cm2 = confusion_matrix(y_test, y_pred2, labels=RF.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm2,display_labels=RF.classes_)
disp.plot()
```

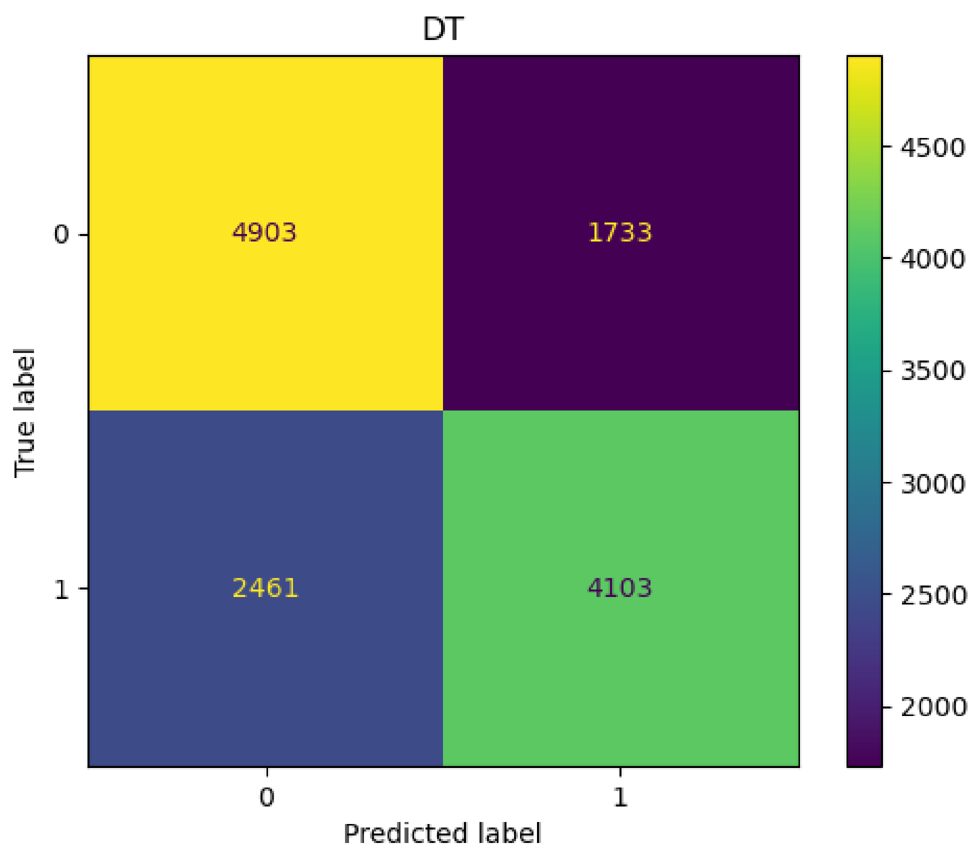
```
plt.title("RF")


cm3 = confusion_matrix(y_test, y_pred3, labels=KNN.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm3,display_labels=KNN.classes_)
disp.plot()
plt.title("KNN")

cm4 = confusion_matrix(y_test, y_pred4, labels=DT.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm4,display_labels=DT.classes_)
disp.plot()
plt.title("DT")

cm5 = confusion_matrix(y_test, y_pred5, labels=DT.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm5,display_labels=DT.classes_)
disp.plot()
plt.title("LR")

cm6 = confusion_matrix(y_test, y_pred6, labels=DT.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm6,display_labels=DT.classes_)
disp.plot()
plt.title("NB")
```

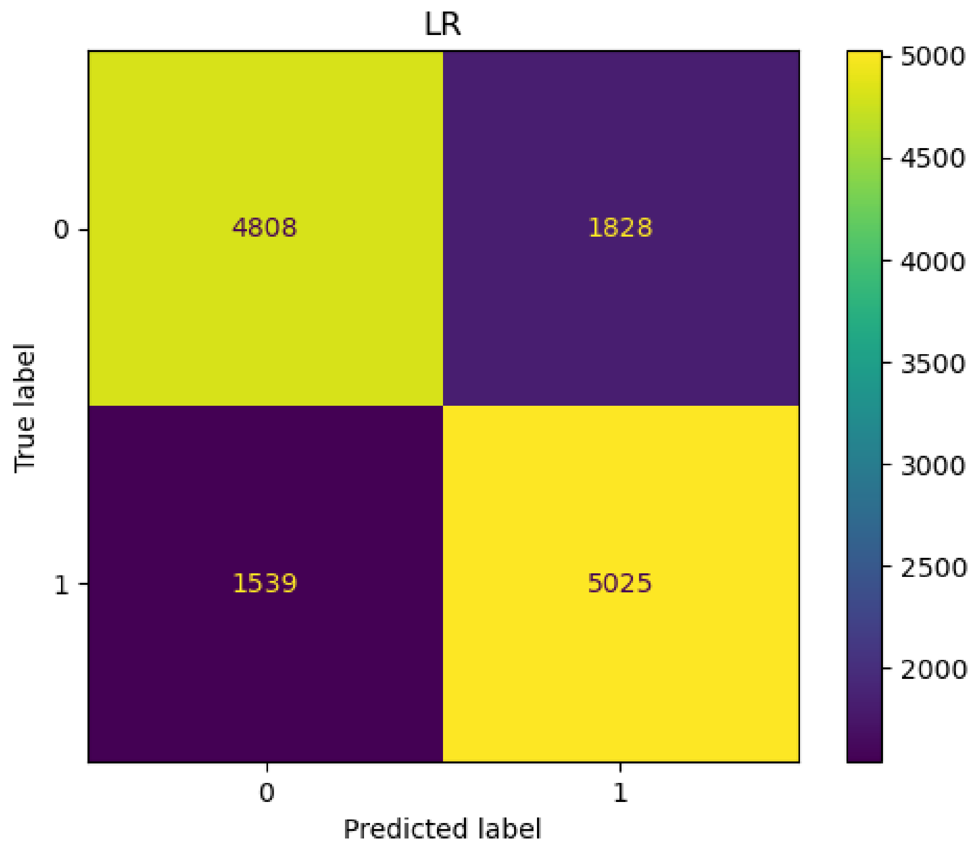
```
[447]: Text(0.5, 1.0, 'NB')
```

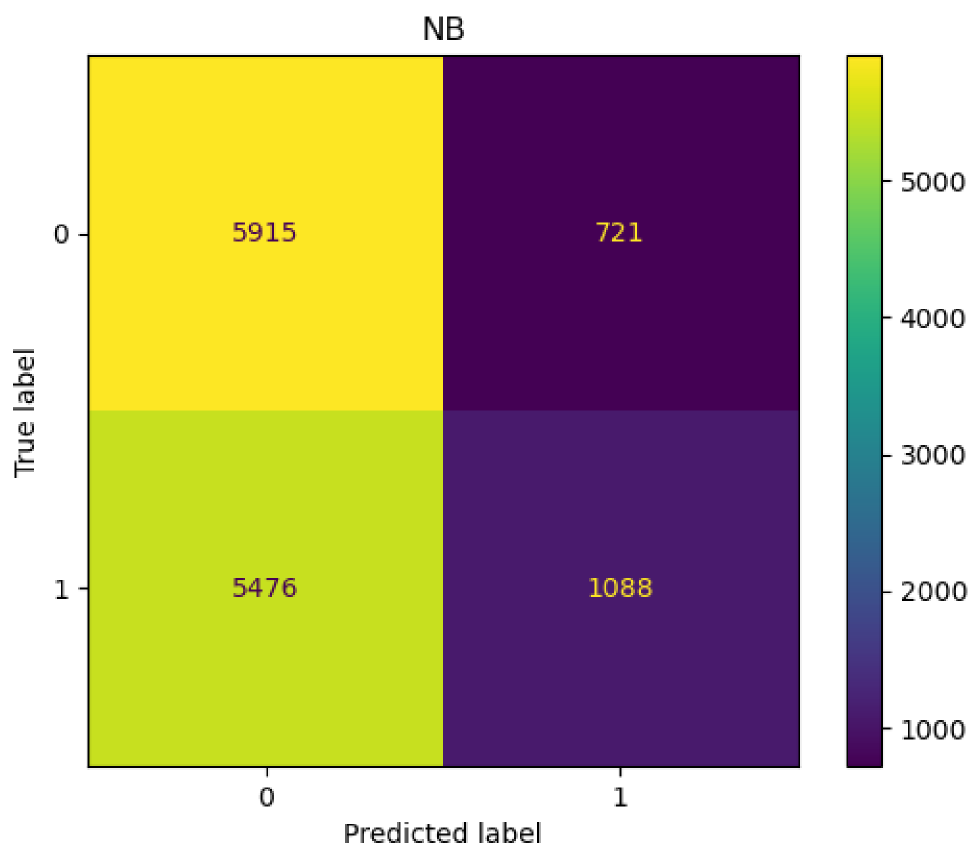

SVM
True label
Predicted label
0
1
4733
1903
1330
5234
5000
4500
4000
3500
3000
2500
2000
1500


RF
True label
Predicted label
0
1
0
1
5136
1500
2026
4538
5000
4500
4000
3500
3000
2500
2000
1500


KNN
True label
Predicted label
0
1
0
1
4428
2208
2412
4152
4250
4000
3750
3500
3250
3000
2750
2500
2250


DT
4903
1733
2461
4103
True label
Predicted label
0
1
0
1
4500
4000
3500
3000
2500
2000


LR
True label
Predicted label
0
1
0
1
4808
1828
1539
5025
5000
4500
4000
3500
3000
2500
2000



[448]:
```
#printing the evaluation metrics for all classifiers
print('SVM metrics\n')
confusion_metrics(cm1)
print('\n\n')
print('RF metrics\n')
confusion_metrics(cm2)
print('\n\n')
print('KNN metrics\n')
confusion_metrics(cm3)
print('\n\n')
print('DT metrics\n')
confusion_metrics(cm4)
print('\n\n')
print('LR metrics\n')
confusion_metrics(cm5)
print('\n\n')
print('NB metrics\n')
confusion_metrics(cm6)
print('\n\n')
```

```
SVM metrics

True Positives: 5234
True Negatives: 4733
False Positives: 1903
False Negatives: 1330
--------------------------------------------------
Accuracy: 0.76
Mis-Classification: 0.24
Sensitivity: 0.8
Specificity: 0.71
Precision: 0.71
f_1 Score: 0.75



RF metrics

True Positives: 4538
True Negatives: 5136
False Positives: 1500
False Negatives: 2026
--------------------------------------------------
Accuracy: 0.73
Mis-Classification: 0.27
Sensitivity: 0.69
Specificity: 0.77
Precision: 0.77
f_1 Score: 0.73



KNN metrics

True Positives: 4152
True Negatives: 4428
False Positives: 2208
False Negatives: 2412
--------------------------------------------------
Accuracy: 0.65
Mis-Classification: 0.35
Sensitivity: 0.63
Specificity: 0.67
Precision: 0.67
f_1 Score: 0.65
```

```
DT metrics

True Positives: 4103
True Negatives: 4903
False Positives: 1733
False Negatives: 2461
--------------------------------------------------
Accuracy: 0.68
Mis-Classification: 0.32
Sensitivity: 0.63
Specificity: 0.74
Precision: 0.74
f_1 Score: 0.68



LR metrics

True Positives: 5025
True Negatives: 4808
False Positives: 1828
False Negatives: 1539
--------------------------------------------------
Accuracy: 0.74
Mis-Classification: 0.26
Sensitivity: 0.77
Specificity: 0.72
Precision: 0.72
f_1 Score: 0.74



NB metrics

True Positives: 1088
True Negatives: 5915
False Positives: 721
False Negatives: 5476
--------------------------------------------------
Accuracy: 0.53
Mis-Classification: 0.47
Sensitivity: 0.17
Specificity: 0.89
Precision: 0.89
f_1 Score: 0.28
```

[ ]:

```
```

0.10 Task 6.2 :

Assume six data points with two binary attributes, 1 and 2, are given as listed in Table 1. These data points belong to three classes, {1,2,3}, and our purpose is to classify these data points using a decision tree classifier with only one split.

Calculate the information gain values when the data points are split using 1 and 2.
Explain which split is better and why. Draw the decision tree using the best split, label the branches, and determine what the predicted class label in each leaf is.
[10%]

Creating the dataframe :

[454]:
```
data = {
    'X1': [1, 1, 1, 1, 0, 0],
    'X2': [1, 1, 1, 0, 0, 0],
    'Y':  [1, 1, 2, 3, 2, 3]
}
```

[455]:
```
df = pd.DataFrame(data)

df
```

```
<IPython.core.display.HTML object>
```

[455]:

[456]:
```
# Function to calculate entropy
def entropy(y):
    proportions = y.value_counts(normalize=True)
    return -sum(p * np.log2(p) for p in proportions if p > 0)
```

[457]:
```
# Calculate initial entropy
initial_entropy = entropy(df['Y'])
print(f"Initial Entropy: {initial_entropy:.4f} bits")
```

```
Initial Entropy: 1.5850 bits
```

[458]:
```
# Split on X1
x1_left = df[df['X1'] == 0]['Y']
x1_right = df[df['X1'] == 1]['Y']
entropy_x1_left = entropy(x1_left)
entropy_x1_right = entropy(x1_right)
weighted_entropy_x1 = (len(x1_left)/len(df)) * entropy_x1_left + (len(x1_right)/
 ↪len(df)) * entropy_x1_right
gain_x1 = initial_entropy - weighted_entropy_x1
```

```
print(f"Entropy X1=0: {entropy_x1_left:.4f}, X1=1: {entropy_x1_right:.4f}")
print(f"Information Gain (X1): {gain_x1:.4f} bits")
```

```
Entropy X1=0: 1.0000, X1=1: 1.5000
Information Gain (X1): 0.2516 bits
```

[459]:
```
# Split on X2
x2_left = df[df['X2'] == 0]['Y']
x2_right = df[df['X2'] == 1]['Y']
entropy_x2_left = entropy(x2_left)
entropy_x2_right = entropy(x2_right)
weighted_entropy_x2 = (len(x2_left)/len(df)) * entropy_x2_left + (len(x2_right)/
 ↪len(df)) * entropy_x2_right
gain_x2 = initial_entropy - weighted_entropy_x2
print(f"Entropy X2=0: {entropy_x2_left:.4f}, X2=1: {entropy_x2_right:.4f}")
print(f"Information Gain (X2): {gain_x2:.4f} bits")
```

```
Entropy X2=0: 0.9183, X2=1: 0.9183
Information Gain (X2): 0.6667 bits
```

[460]:
```
# Train decision tree with max_depth=1
X = df[['X1', 'X2']]
y = df['Y']
clf = DecisionTreeClassifier(max_depth=1, criterion='entropy', random_state=42)
clf.fit(X, y)
```

[460]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=1, random_state=42)
```

[461]:
```
# Display the tree
print("\nDecision Tree Structure:")
print(tree.export_text(clf, feature_names=['X1', 'X2']))
```

```

Decision Tree Structure:
|--- X2 <= 0.50
|   |--- class: 3
|--- X2 >  0.50
|   |--- class: 1

```

[462]:
```
# Predictions
predictions = clf.predict(X)
print("Predicted Class Labels:", predictions)
print("Actual Class Labels:", y.values)
```

```
Predicted Class Labels: [1 1 1 3 3 3]
Actual Class Labels: [1 1 2 3 2 3]
```

- X2 has higher information gain (0.6667 vs. 0.2517), matching our manual calculation.
- The tree splits on X2 <= 0.50, with predicted classes 3 (left) and 1 (right), as expected.

[ ]:

```
```

0.10.1 Task 6.3 :

Use the K-means clustering to find two different clusters in the following sequence of three-dimensional points:
X=[(1,9,14),(2,18,23),(3,30,30),(4,21,9),(5,9,17),(6,25,32),(7,36,25),(8,10,12),(9,38,45),(10,1,2)]
Choose two random centres for your clusters to start the algorithm. Include the centres of clusters and calculations for each iteration in your answer. You can use the Euclidean distance to calculate the distance between points [5%

[476]:

```python
# Define the data
X = np.array([
    (1,9,14), (2,18,23), (3,30,30), (4,21,9), (5,9,17),
    (6,25,32), (7,36,25), (8,10,12), (9,38,45), (10,1,2)
])
```

[477]:

```python
# Initialize K-means with 2 clusters and specific initial centroids
initial_centers = np.array([[1,9,14], [10,1,2]])
kmeans = KMeans(n_clusters=2, init=initial_centers, n_init=1, random_state=42)
kmeans.fit(X)
```

```
C:\Users\User\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1446:
UserWarning: KMeans is known to have a memory leak on Windows with MKL, when
there are less chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=1.
  warnings.warn(
```

[477]:

```
KMeans(init=array([[ 1,  9, 14],
       [10,  1,  2]]), n_clusters=2, n_init=1,
       random_state=42)
```

[478]:

```python
# Results
print("Cluster Assignments (0 or 1):", kmeans.labels_)
print("Final Cluster Centers:")
print(kmeans.cluster_centers_)
```

```
Cluster Assignments (0 or 1): [1 0 0 1 1 0 0 1 0 1]
Final Cluster Centers:
[[ 5.4 29.4 31. ]
 [ 5.6 10.  10.8]]
```

[479]:

```python
# Detailed output
for i, point in enumerate(X):
    print(f"Point {point}: Cluster {kmeans.labels_[i]}")
```

```
Point [ 1  9 14]: Cluster 1
Point [ 2 18 23]: Cluster 0
Point [ 3 30 30]: Cluster 0
Point [ 4 21  9]: Cluster 1
Point [ 5  9 17]: Cluster 1
Point [ 6 25 32]: Cluster 0
Point [ 7 36 25]: Cluster 0
Point [ 8 10 12]: Cluster 1
Point [ 9 38 45]: Cluster 0
Point [10  1  2]: Cluster 1
```

[ ]:

[ ]: