FACULTY OF ENGINEERING

SCHOOL OF COMPUTING

SEMESTER 2/20222023

SECP2753-01 DATA MINING

FINAL PROJECT

Youtube Link: https://youtu.be/qIz4obOzSCY

LECTURER'S NAME:

DR. ROZILAWATI BINTI DOLLAH

NAME	MATRIC ID
ALYA BALQISS BINTI AZAHAR	A21EC0158
NIK AMIRUL ARIFF BIN AMRAN	A21EC0214
MUHAMMAD IQMAL BIN SIS	A21EC0080
MUHAMMAD HARITH HAKIM BIN OTHMAN	A21EC0205

Table of Content

INTRODUCTION..... **2**
DATA PREPROCESSING..... **3**
DATA MINING TASKS..... **20**
CLASSIFICATION..... 20
Classification Algorithm..... 20
REGRESSION..... 27
Regression Algorithm..... 28
Simple Linear Regression..... 29
Multiple Linear Regression..... 34
CLUSTERING..... 40
Elbow Method..... 42
Silhouette Analysis Method..... 44
Hierarchical Clustering..... 47
PREDICTION..... 50
Prediction Algorithm..... 51
Logistic Regression..... 52
Decision Tree..... 57
CONCLUSION..... **63**

INTRODUCTION

Large datasets may be mined for useful facts and insights through a technique called data mining. It entails using various computer tools to find data patterns, connections, and trends. The discovery and execution of specified activities to produce desired results is one of the essential components of data mining. The team concentrates on four main data mining tasks in this context: classification, clustering, regression, and prediction.

The practice of placing data instances into predetermined classes or categories according to their features is known as classification. This job is quite helpful when working with datasets where the target variable is categorical and building a model to categorise new occurrences reliably.

On the other hand, clustering entails assembling related data instances based on their intrinsic commonalities. Because it is an unsupervised learning problem, the algorithm recognizes patterns and structures in the data without knowing the class labels beforehand.

A data mining process called regression uses the correlations between independent variables to forecast a continuous numerical result. It is frequently utilized when the target variable is numerical and aims to create a model to estimate values for new occurrences.

Lastly, prediction entails predicting future events using previous data. Predictive models may be created to calculate the chance of specific occurrences or values occurring by examining patterns and trends in the available data.

The dataset for the data mining project the team is working on focuses on analyzing the link between sex, age, income, etc. The main goal is to establish if these variables may be used to predict whether a person's income will surpass $50,000 per year. Additionally, the team has also utilized another dataset related to forest fires, indicating the exploration of different domains and diverse data sources.

DATA PREPROCESSING

Dataset 1: Adult.csv (**https://archive.ics.uci.edu/dataset/2/adult)**

Step 1: Importing the libraries

```
#Load packages
import numpy as np
import pandas as pd
import missingno as msno
```

The code line "import numpy as np, pandas as pd, missingno as msno" imports the NumPy, Pandas, and missingno libraries in Python. These libraries are commonly used for numerical computations, data manipulation and analysis, and visualizing missing data in datasets, respectively.

Step 2: Import the dataset and set the header for each column

```
# Load data
df = pd.read_csv('adult.data', header=None)

# Rename columns
column_names = [
     "age", "workclass", "fnlwgt", "education", "education-num", "marital-status",
     "occupation", "relationship", "race", "sex", "capital-gain", "capital-loss",
     "hours-per-week", "native-country", "class"
]

df.columns = column_names
```

The code snippet begins with the line "df = pd.read_csv('adult.data', header=None)" which reads a CSV file named 'adult.data' into a Pandas DataFrame called "df". The "header=None" argument indicates that the CSV file does not have a header row.

The code assigns a list of column names to the variable "column_names". This list contains the names for various attributes in the dataset, such as age, workclass, fnlwgt, education, education-num, marital-status, occupation, relationship, race, sex, capital-gain, capital-loss, hours-per-week, native-country, and class.

The line "df.columns = column_names" assigns the list of column names to the columns of the DataFrame "df". This step ensures that the DataFrame's columns are labeled with the provided column names.

```
# Data types
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             32561 non-null  int64
 1   workclass       32561 non-null  object
 2   fnlwgt          32561 non-null  int64
 3   education       32561 non-null  object
 4   education-num   32561 non-null  int64
 5   marital-status  32561 non-null  object
 6   occupation      32561 non-null  object
 7   relationship    32561 non-null  object
 8   race            32561 non-null  object
 9   sex             32561 non-null  object
 10  capital-gain    32561 non-null  int64
 11  capital-loss    32561 non-null  int64
 12  hours-per-week  32561 non-null  int64
 13  native-country  32561 non-null  object
 14  class           32561 non-null  object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
None
```

Step 3: Taking care of the missing data

The presence of missing data is a common challenge in survey-based data collection, where some observations in the dataset may lack values for certain features.

There are several techniques to handle the missing data which include:

- Deleting the observation with the missing value(s).

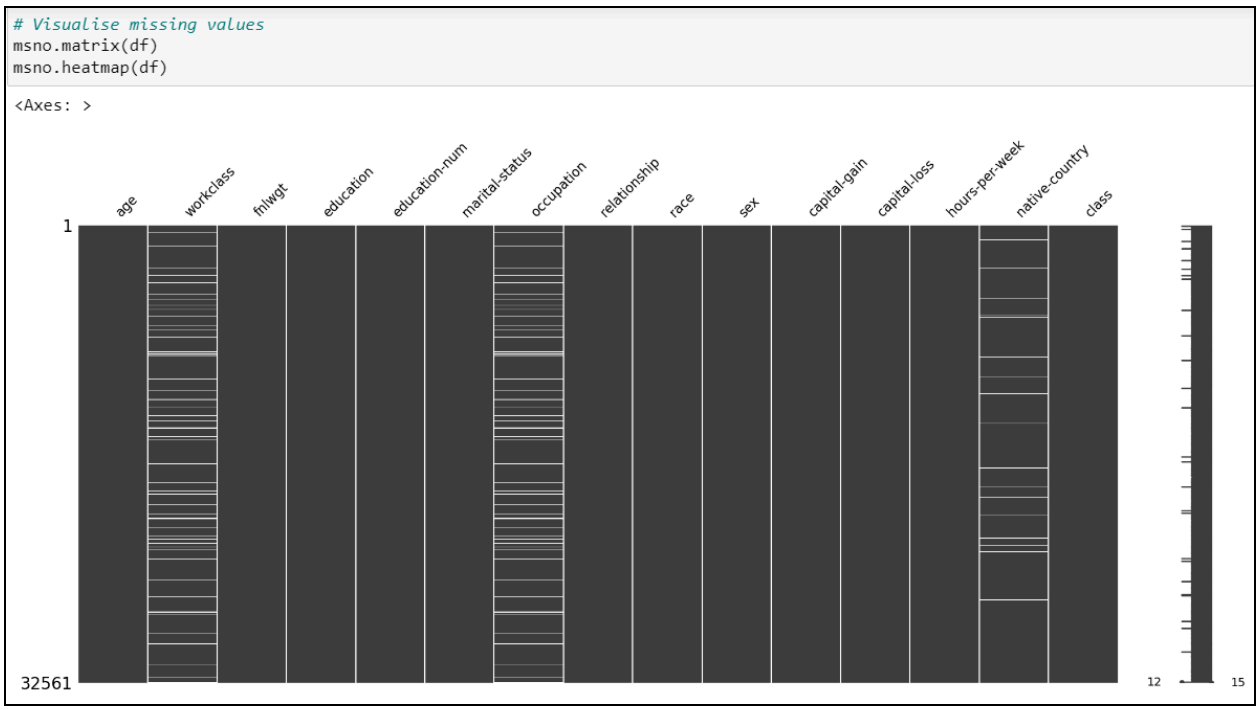
```
# Treat '?' as NA
data = {'Count': df[df==' ?'].count(),
        'Percentage': (df[df==' ?'].count() / df.shape[0] * 100)}
data = pd.DataFrame(data)

df = df.replace(' ?', np.nan)

print(data)
```

```
                Count  Percentage
age                 0    0.000000
workclass        1836    5.638647
fnlwgt              0    0.000000
education           0    0.000000
education-num       0    0.000000
marital-status      0    0.000000
occupation       1843    5.660146
relationship        0    0.000000
race                0    0.000000
sex                 0    0.000000
capital-gain        0    0.000000
capital-loss        0    0.000000
hours-per-week      0    0.000000
native-country    583    1.790486
class               0    0.000000
```

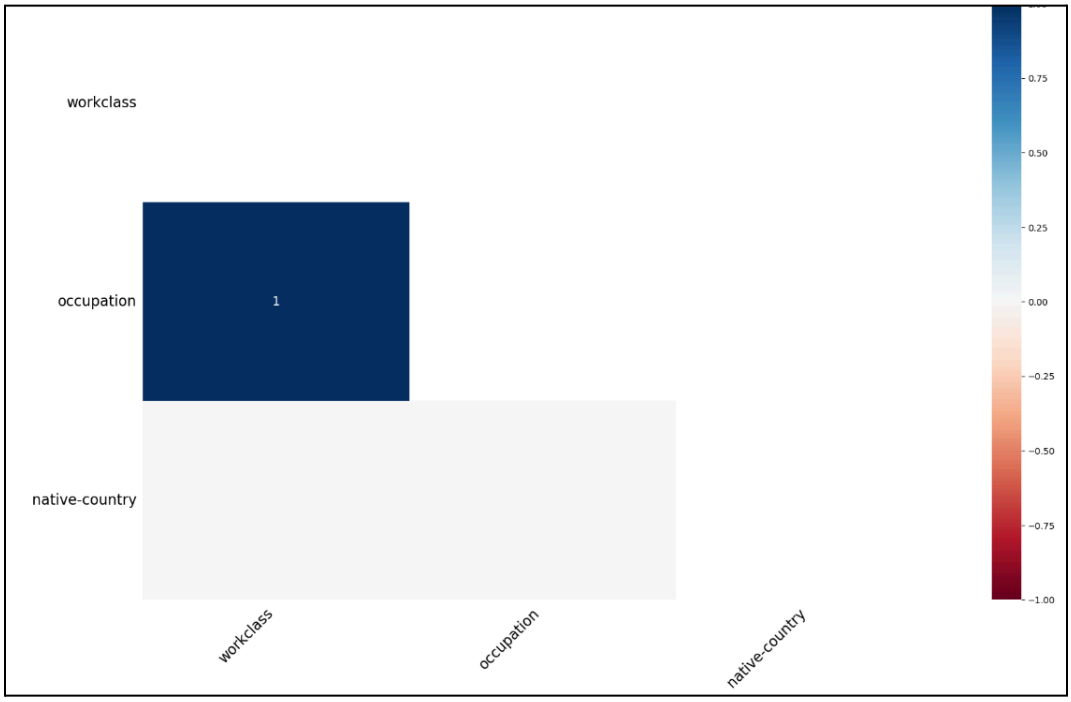
The code treats '?' values as missing values by replacing them with NaN. It calculates the count and percentage of '?' values in each column, stores them in a dictionary called "data," and converts it into a DataFrame. Then, it replaces '?' values with NaN in the DataFrame. Finally, it prints the resulting DataFrame.

```
# Visualise missing values
msno.matrix(df)
msno.heatmap(df)
```

```
<Axes: >
```



The line "msno.matrix(df)" generates a matrix visualization that displays missing values as white lines in the DataFrame. This visualization provides an overview of the distribution of missing values across different columns.



The next line, "msno.heatmap(df)", generates a heatmap visualization that represents the correlation between missing values in different columns. The heatmap displays a color-coded grid where darker squares indicate a higher correlation of missing values between columns. This visualization helps to identify patterns or clusters of missing values in the dataset

```
# Remove missing values
df = df.dropna()
```

The code "df = df.dropna()" removes missing values from the DataFrame by using the dropna() function. After executing "df = df.dropna()", the DataFrame "df" will only contain complete rows without any missing values.

- Deleting the observation with both the values of ‘capital-gains’ and ‘hours-per-week’ equal to 0.

```
# Count capital-gain & capital-loss != 0
print(df[(df['capital-gain'] != 0) | (df['capital-loss'] != 0)].count())
```

```
age               3965
workclass         3965
fnlwgt            3965
education         3965
education-num     3965
marital-status    3965
occupation        3965
relationship      3965
race              3965
sex               3965
capital-gain      3965
capital-loss      3965
hours-per-week    3965
native-country    3965
class             3965
dtype: int64
```

This code counts the number of rows in the DataFrame where either the 'capital-gain' or 'capital-loss' column is not equal to 0. It uses the expression "df[(df['capital-gain'] != 0) | (df['capital-loss'] != 0)].count()" to filter the DataFrame and count the matching rows.

```
# Create subset data of capital-gain & capital-loss != 0
data = df[(df['capital-gain'] != 0) | (df['capital-loss'] != 0)]

# Remove the "fnlwgt" column
df = df.drop("fnlwgt", axis=1)
```

The code snippet creates a subset of data from the DataFrame where either the 'capital-gain' or 'capital-loss' column is not equal to 0, and assigns it to a variable named "data".

the code removes the "fnlwgt" column from the DataFrame using the drop() function. The line "df = df.drop("fnlwgt", axis=1)" modifies the DataFrame by dropping the specified column along the axis 1, which represents the columns.

Step 4: Encoding categorical data

Encoding is a process where textual data is converted into a numerical representation. This transformation is necessary when dealing with categorical data, as it involves converting data that belong to specific categories into a numeric format.

```
df["sex"] = df["sex"].map({"Male": 0, "Female": 1})
```

```
exclude_columns = ["age", "fnlwgt", "capital-gain", "capital-loss", "education-num", "hours-per-week"]

# Iterate over each column (excluding columns in exclude_columns)
for column in df.columns:
    if column in exclude_columns:
        continue  # Skip this column and move to the next one

    # Get unique values in the column
    unique_values = df[column].unique()

    # Display the column name and the number of unique values
    num_unique_values = len(unique_values)
    print(f"Column: {column}")
    print(f"Number of unique values: {num_unique_values}")

    # Display the unique values
    print("Unique values:")
    for value in unique_values:
        print(value)

    print()  # Add a newline for readability
```

The code snippet excludes specific columns from analysis using the "exclude_columns" list. It iterates over each column in the DataFrame, retrieves the unique values, and displays the column name, number of unique values, and the unique values themselves. The snippet includes a newline for better readability.

```
 Assoc-voc
 Prof-school
 5th-6th
 10th
 Preschool
 12th
 1st-4th

Column: marital-status
Number of unique values: 7
Unique values:
 Never-married
 Married-civ-spouse
 Divorced
 Married-spouse-absent
 Separated
 Married-AF-spouse
 Widowed
```

Step 4: Write the cleaned data to a new .csv file

```
# Write cleaned data
data.to_csv('adult_cleaned.csv', index=False)
```

The code snippet writes the cleaned data, stored in the DataFrame "data", to a CSV file named 'adult_cleaned.csv'. It uses the to_csv() function with the parameters 'adult_cleaned.csv' for the file name and index=False to exclude the index column from being written to the file. By executing this line of code, the cleaned data will be saved as a CSV file for further use or analysis.

Dataset 2: forestfires.csv (**https://archive.ics.uci.edu/dataset/162/forest+fires**)

Step 1: Importing all the necessary packages

```
import numpy as np
import pandas as pd
from scipy import stats
import matplotlib.pyplot as plt
```

This code shows how to import all the necessary packages for further preprocessing of the data, below is the meaning of each packages:-

- *numpy* is a library for numerical computing in Python, providing functions and data structures for handling large arrays and matrices.
- *pandas* is a library for data manipulation and analysis, providing data structures like DataFrame to work with structured data.
- *scipy.stats* is a module within the SciPy library that provides a wide range of statistical functions and distributions.
- *matplotlib.pyplot* is a plotting library that provides a MATLAB-like interface for creating various types of plots and visualizations.

Step 2: Importing dataset

```
df = pd.read_csv("forestfires.csv")
```

Using the pd.read_csv function to import the forestfires.csv file and store into the *'**df**'* variables for further analysis of the data.

Step 3: Creating new column named ‘size_category’

```
df['size_category'] = np.where(df['area']>6, '1', '0')
df['size_category']= pd.to_numeric(df['size_category'])
df.tail(10)
```

Creating the new column named ‘size_category, in which the content of it will store the value of small fire (0) and large fire (1), based on area column. If the area value is greater than 6, the size category will be 1, and if not, 0. Then the ‘size_category’ column type will be set to numerical. The code then shows the last 10 rows of the new data.

	X	Y	month	day	FFMC	DMC	DC	ISI	temp	RH	wind	rain	area	size_category
507	2	4	aug	fri	91.0	166.9	752.6	7.1	25.9	41	3.6	0.0	0.00	0
508	1	2	aug	fri	91.0	166.9	752.6	7.1	25.9	41	3.6	0.0	0.00	0
509	5	4	aug	fri	91.0	166.9	752.6	7.1	21.1	71	7.6	1.4	2.17	0
510	6	5	aug	fri	91.0	166.9	752.6	7.1	18.2	62	5.4	0.0	0.43	0
511	8	6	aug	sun	81.6	56.7	665.6	1.9	27.8	35	2.7	0.0	0.00	0
512	4	3	aug	sun	81.6	56.7	665.6	1.9	27.8	32	2.7	0.0	6.44	1
513	2	4	aug	sun	81.6	56.7	665.6	1.9	21.9	71	5.8	0.0	54.29	1
514	7	4	aug	sun	81.6	56.7	665.6	1.9	21.2	70	6.7	0.0	11.16	1
515	1	4	aug	sat	94.4	146.0	614.7	11.3	25.6	42	4.0	0.0	0.00	0
516	6	3	nov	tue	79.5	3.0	106.7	1.1	11.8	31	4.5	0.0	0.00	0

Step 4: finding column data type

```
df.info()
```

The code shows data type of each column using the .info() function. The data type is used to identify whether the column needs some adjustments or not. In this case, all the data type is suitable or good for further analysis.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 517 entries, 0 to 516
Data columns (total 14 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   X              517 non-null    int64
 1   Y              517 non-null    int64
 2   month          517 non-null    object
 3   day            517 non-null    object
 4   FFMC           517 non-null    float64
 5   DMC            517 non-null    float64
 6   DC             517 non-null    float64
 7   ISI            517 non-null    float64
 8   temp           517 non-null    float64
 9   RH             517 non-null    int64
 10  wind           517 non-null    float64
 11  rain           517 non-null    float64
 12  area           517 non-null    float64
 13  size_category  517 non-null    int64
dtypes: float64(8), int64(4), object(2)
memory usage: 56.7+ KB
```

Step 5: finding missing value

Next step is to find if the dataset has any missing values. To find this, The function '.isnull()' is used to detect whether any value has a null value. Then to sum all of the missing values for each column is by using the '.sum()' function. Based on the result, there are no missing values for every column.

```
df.isnull().sum()
```

```
X              0
Y              0
month          0
day            0
FFMC           0
DMC            0
DC             0
ISI            0
temp           0
RH             0
wind           0
rain           0
area           0
size_category  0
dtype: int64
```

Step 6: Changing the ‘month’ column from name to number

After that, the ‘month’ column changed from words to numbers based on real-life situations. The map function is used to change every row of the ‘month’ column.

```
df['month'] = df['month'].map({
    'jan':1,
    'feb':2,
    'mar':3,
    'apr':4,
    'may':5,
    'jun':6,
    'jul':7,
    'aug':8,
    'sep':9,
    'oct':10,
    'nov':11,
    'dec':12,
})
```

	X	Y	month	day	FFMC	DMC	DC	ISI	temp	RH	wind	rain	area	\
0	7	5	3	fri	86.2	26.2	94.3	5.1	8.2	51	6.7	0.0	0.00	
1	7	4	10	tue	90.6	35.4	669.1	6.7	18.0	33	0.9	0.0	0.00	
2	7	4	10	sat	90.6	43.7	686.9	6.7	14.6	33	1.3	0.0	0.00	
3	8	6	3	fri	91.7	33.3	77.5	9.0	8.3	97	4.0	0.2	0.00	
4	8	6	3	sun	89.3	51.3	102.2	9.6	11.4	99	1.8	0.0	0.00	
..	..	..	...	...	...	...	...	...	...	..	...	...	...	
512	4	3	8	sun	81.6	56.7	665.6	1.9	27.8	32	2.7	0.0	6.44	
513	2	4	8	sun	81.6	56.7	665.6	1.9	21.9	71	5.8	0.0	54.29	
514	7	4	8	sun	81.6	56.7	665.6	1.9	21.2	70	6.7	0.0	11.16	
515	1	4	8	sat	94.4	146.0	614.7	11.3	25.6	42	4.0	0.0	0.00	
516	6	3	11	tue	79.5	3.0	106.7	1.1	11.8	31	4.5	0.0	0.00	

	size_category
0	0
1	0
2	0
3	0
4	0
..	...
512	1
513	1
514	1
515	0
516	0

Step 7: Detecting outliers on rain and area

The probability of outliers happening in ‘area’ and ‘rain’ is quite big. This is because it is one of the most important values needed for further analysis. That is why our group decided to choose these 2 column as our priority to find if there exist outliers in it. The column is then stored in a variable named ‘columns_of_outliers’ so that it can be used to perform further processes if needed.

```
columns_of_outliers = [12, 13]
```

Calculating the Z scores, so that it can be used to find the outliers of the data.

```
# Calculate z-scores for the selected columns
z_scores = np.abs(stats.zscore(df.iloc[:,columns_of_outliers]))
```

```
# Define a threshold for considering values as outliers (e.g., z-score > 3)
threshold = 3

# Find outliers based on the threshold
outliers = np.where(z_scores > threshold)
```

Printing existing outliers

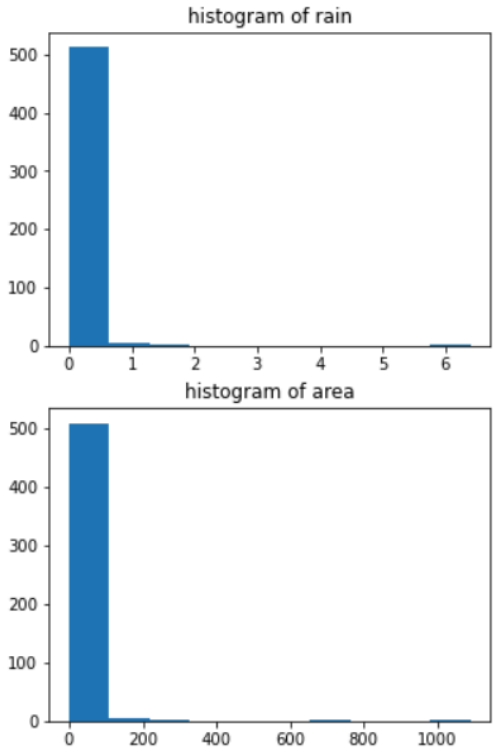
```
print("Indices of outliers:")
for i, j in zip(*outliers):
    print("Row:", i, "Column:",columns_of_outliers[j])
```

```
Indices of outliers:
Row: 237 Column: 12
Row: 238 Column: 12
Row: 415 Column: 12
Row: 479 Column: 12
```

Step 8: Visualizing outliers and data for area and rain (histogram)

The outliers are then visualized using the histogram graph. Based on the graph, it shows that the data is quite ‘flat’ and it is not ‘pretty’ at all.

```
# visualizing
fig, ax = plt.subplots(2, figsize = (5, 8))
ax[0].hist(df['rain'])
ax[0].title.set_text('histogram of rain')
ax[1].hist(df['area'])
ax[1].title.set_text('histogram of area')
```

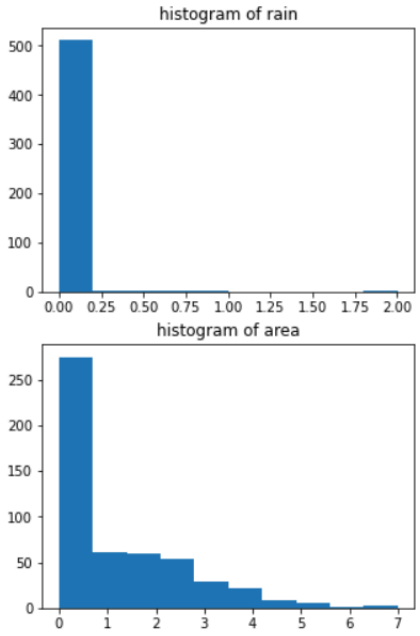


Step 9: Scaling the data using natural logarithm

The scaling of the data is to make sure the data looks 'prettier' and to eliminate the outliers. Natural logarithm scaling has been used to eliminate this problem. Then the data is visualized again using a histogram graph. The data is now more compact compared to before. Thus it can be considered successful.

```
# natural logarithm scaling (+1 to prevent errors at 0)
df.loc[:, ['rain', 'area']] = df.loc[:, ['rain', 'area']].apply(lambda x: np.log(x + 1), axis = 1)
```

```
# visualizing
fig, ax = plt.subplots(2, figsize = (5, 8))
ax[0].hist(df['rain'])
ax[0].title.set_text('histogram of rain')
ax[1].hist(df['area'])
ax[1].title.set_text('histogram of area')
```



Step 10: Creating new file 'forestfires_cleaned.csv' for further analysis

Lastly, the new .csv file has been created to separate which file has been cleaned or not. This is also to make sure that our group has the backup for raw data (uncleaned data) and the new data. The 'df.to_csv' function is to generated a new file for further data mining tasks.

```
# Write cleaned data
df.to_csv('forestfire_cleaned.csv', index=False)
```

DATA MINING TASKS

CLASSIFICATION

Classification is the process of developing a model capable of comprehending and discerning distinct groups or concepts within data. It entails organizing items into specific categories. For instance, given a collection of animal pictures, the objective is for the model to accurately identify whether each picture depicts a cat, dog, or bird. To accomplish this, a training set consisting of animal pictures is required, where the category of each picture is already known. The model utilizes this training set to acquire knowledge and determine how to classify new pictures that it has not encountered previously.

Classification Algorithm

Sample dataset with adults with different characteristics and total income that will obtain in the future. The dataset also consists of multiple categories of characteristics such as sex, age, occupation, capital gain, capital loss, etc.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	age	workclass	education	education	marital-st	occupatio	relationsh	race	sex	capital-ga	capital-los	hours-per	native-co	class
2	39	State-gov	Bachelors	13	Never-ma	Adm-cler	Not-in-fa	White	0	2174	0	40	United-St	<=50K
3	31	Private	Masters	14	Never-ma	Prof-spec	Not-in-fa	White	1	14084	0	50	United-St	>50K
4	42	Private	Bachelors	13	Married-c	Exec-man	Husband	White	0	5178	0	40	United-St	>50K
5	43	Private	11th	7	Married-c	Transport	Husband	White	0	0	2042	40	United-St	<=50K
6	45	Private	Bachelors	13	Divorced	Exec-man	Own-chil	White	0	0	1408	40	United-St	<=50K
7	47	Private	Prof-scho	15	Married-c	Prof-spec	Wife	White	1	0	1902	60	Honduras	>50K
8	30	Private	HS-grad	9	Married-c	Machine-	Husband	White	0	5013	0	40	United-St	<=50K
9	30	Private	Bachelors	13	Married-c	Sales	Husband	White	0	2407	0	40	United-St	<=50K
10	44	Private	HS-grad	9	Divorced	Craft-repa	Not-in-fa	White	1	14344	0	40	United-St	>50K
11	48	Self-emp	Doctorate	16	Married-c	Prof-spec	Husband	White	0	0	1902	60	United-St	>50K
12	44	Private	Bachelors	13	Married-c	Exec-man	Husband	White	0	15024	0	60	United-St	>50K
13	32	Self-emp	HS-grad	9	Married-c	Craft-repa	Husband	White	0	7688	0	40	United-St	>50K
14	56	Self-emp	HS-grad	9	Married-c	Other-ser	Husband	White	0	0	1887	50	Canada	>50K
15	28	Private	Some-col	10	Married-c	Sales	Wife	White	1	4064	0	25	United-St	<=50K
16	20	Private	Some-col	10	Never-ma	Adm-cler	Own-chil	White	1	0	1719	28	United-St	<=50K
17	24	Private	HS-grad	9	Never-ma	Other-ser	Not-in-fa	White	1	0	1762	40	United-St	<=50K
18	38	Self-emp	HS-grad	9	Married-c	Craft-repa	Husband	White	0	4386	0	35	United-St	<=50K
19	45	Private	Assoc-voc	11	Never-ma	Prof-spec	Not-in-fa	White	1	0	1564	40	United-St	>50K
20	64	Private	11th	7	Married-c	Craft-repa	Husband	White	0	0	2179	40	United-St	<=50K
21	71	Self-emp	Some-col	10	Separated	Sales	Unmarrie	Black	0	0	1816	2	United-St	<=50K
22	27	Private	HS-grad	9	Never-ma	Other-ser	Not-in-fa	White	0	0	1980	40	United-St	<=50K
23	51	Private	Some-col	10	Married-c	Sales	Husband	White	0	0	1977	40	United-St	>50K
24	40	Federal-g	Masters	14	Never-ma	Exec-man	Not-in-fa	White	1	14084	0	55	United-St	>50K
25	35	Private	Masters	14	Married-c	Prof-spec	Other-rel	White	0	7298	0	40	United-St	>50K
26	26	Private	Masters	14	Never-ma	Prof-spec	Not-in-fa	White	1	0	1876	40	United-St	<=50K
27	42	Local-gov	Some-col	10	Never-ma	Prof-spec	Not-in-fa	White	1	0	1340	40	United-St	<=50K
28	36	Private	Bachelors	13	Married-c	Other-ser	Husband	Black	0	7298	0	36	United-St	>50K
29	58	Self-emp	HS-grad	9	Married-c	Sales	Wife	White	1	15024	0	35	United-St	>50K
30	90	Private	HS-grad	9	Never-ma	Other-ser	Not-in-fa	Black	0	0	2206	40	United-St	<=50K
31	66	Self-emp	HS-grad	9	Married-c	Farming-f	Husband	White	0	1409	0	50	United-St	<=50K
32	35	Private	11th	7	Separated	Transport	Not-in-fa	Black	0	3674	0	40	United-St	<=50K
33	38	Private	Some-col	10	Divorced	Craft-repa	Not-in-fa	White	0	0	1741	40	United-St	<=50K
34	40	Private	Assoc-acd	12	Married-c	Tech-sup	Husband	White	0	0	1977	60	United-St	>50K
35	59	Private	Some-col	10	Married-c	Sales	Husband	White	0	4064	0	40	United-St	<=50K
36	17	Private	9th	5	Never-ma	Other-ser	Own-chil	White	0	1055	0	24	United-St	<=50K

1. Import the necessary packages and classes required for the classification task.

```
import numpy as np
import pandas as pd
import matplotlib as plt
%matplotlib notebook
```

- `numpy` (imported as `np`) for numerical operations
- `matplotlib` (imported as `plt`) for data visualisation
- `pandas` (imported as `pd`) for data manipulation
- `matplotlib notebook` for right format plotting

2. Provide the relevant data and apply any necessary transformations to prepare it for analysis.
   - `Dataset` is set as a variable for the data of the table that we will read afterwards.
   - The dataset is read from a CSV file called 'adult_cleaned.csv' using `pd.read_csv()`.

```
dataset = pd.read_csv('adult_cleaned.csv')
```

```
dataset.head()
```

- We displayed the first five of the data using `head()`.

	age	workclass	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	class
0	39	State-gov	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	0	2174	0	40	United-States	<=50K
1	31	Private	Masters	14	Never-married	Prof-specialty	Not-in-family	White	1	14084	0	50	United-States	>50K
2	42	Private	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	0	5178	0	40	United-States	>50K
3	43	Private	11th	7	Married-civ-spouse	Transport-moving	Husband	White	0	0	2042	40	United-States	<=50K
4	45	Private	Bachelors	13	Divorced	Exec-managerial	Own-child	White	0	0	1408	40	United-States	<=50K

```
X=dataset.iloc[:, [8,0]]

Y=dataset.iloc[:, -1]
```

The code selects three specific columns for analysis: the 8th column (indexed as 9), 0th column (indexed as 1) and the -1st column (indexed as the last column).

The selected columns are stored in X and y, where

- `X` represents the independent variable (Sex and Age)
- `Y` represents the dependent variable (Class).

```
X.head()
```

	sex	age
0	0	39
1	1	31
2	0	42
3	0	43
4	0	45

```
Y.head()
```

```
0     <=50K
1      >50K
2      >50K
3     <=50K
4     <=50K
Name: class, dtype: object
```

3. Split the dataset into **training sets** and **testing sets**

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, Y , test_size=0.25, random_state=0)
```

75% of data used for training and 25% for testing. The `train_test_split` function from `sklearn.model_selection` is used to split X and Y into **training** and **test** sets. The purpose of setting the `random_state` to 0 is to guarantee that the identical random split is produced whenever the code is executed, ensuring reproducibility.

4. Fitting a logistic regression model to the training data

```
from sklearn.linear_model import LogisticRegression

LR = LogisticRegression()

LR.fit(X_train, y_train)
```

Logistic regression is a widely used statistical method that predicts binary or categorical outcomes by modeling the connection between predictor variables and the probability of a specific outcome. When we fit a logistic regression model to the training data, we estimate the coefficients and build a model that can be utilized to predict the class labels of new data points.

5. Make predictions on the testing data

```
pred = LR.predict(X_test)
```

With the model trained, the model is use to predict targets based on the test data.

6. Calculate the accuracy score by comparing the actual values and predicted values.

```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, pred)

TN, FP, FN, TP = confusion_matrix(y_test, pred).ravel()

print('True Positive(TP)  = ', TP)
print('False Positive(FP) = ', FP)
print('True Negative(TN)  = ', TN)
print('False Negative(FN) = ', FN)
```

```
True Positive(TP)  =  498
False Positive(FP) =  272
True Negative(TN)  =  138
False Negative(FN) =  84
```

```
accuracy =  (TP + TN) / (TP + FP + TN + FN)

print('Accuracy of the binary classifier = {:0.3f}'.format(accuracy))
```

```
Accuracy of the binary classifier = 0.641
```

- An result where the model properly predicted the positive class is referred to as a true positive. Similar to a genuine positive, a true negative is a result for which the model accurately foresees the negative class.

- A false positive is a result when the model forecasts the positive class inaccurately.

- A false negative is a result when the model forecasts the negative class inaccurately.

7. Using other models and comparing the models accuracy, precision and recall.

```
models = {}

# Logistic Regression
from sklearn.linear_model import LogisticRegression
models['Logistic Regression'] = LogisticRegression()

# Support Vector Machines
from sklearn.svm import LinearSVC
models['Support Vector Machines'] = LinearSVC()

# Decision Trees
from sklearn.tree import DecisionTreeClassifier
models['Decision Trees'] = DecisionTreeClassifier()

# Random Forest
from sklearn.ensemble import RandomForestClassifier
models['Random Forest'] = RandomForestClassifier()

# Naive Bayes
from sklearn.naive_bayes import GaussianNB
models['Naive Bayes'] = GaussianNB()

# K-Nearest Neighbors
from sklearn.neighbors import KNeighborsClassifier
models['K-Nearest Neighbor'] = KNeighborsClassifier()
```

Looping over each one, train it by calling `.fit()`, make predictions, calculate metrics, and store each result in a dictionary.

```
from sklearn.metrics import accuracy_score, precision_score, recall_score

accuracy, precision, recall = {}, {}, {}

for key in models.keys():

    # Fit the classifier
    models[key].fit(X_train, y_train)

    # Make predictions
    predictions = models[key].predict(X_test)

    # Calculate metrics
    accuracy[key] = accuracy_score(predictions, y_test)
    precision[key] = precision_score(predictions, y_test, pos_label=' >50K')
    recall[key] = recall_score(predictions, y_test, pos_label=' >50K')
```

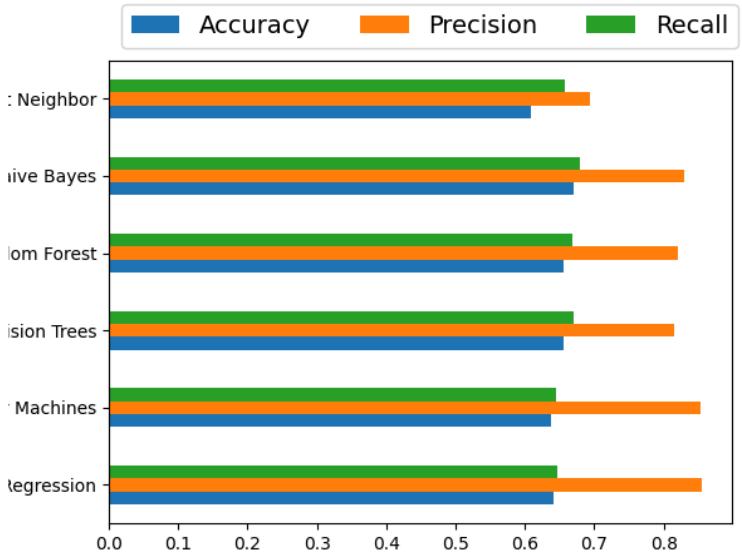
Output:

```
import pandas as pd

df_model = pd.DataFrame(index=models.keys(), columns=['Accuracy', 'Precision', 'Recall'])
df_model['Accuracy'] = accuracy.values()
df_model['Precision'] = precision.values()
df_model['Recall'] = recall.values()

df_model
```

	Accuracy	Precision	Recall
Logistic Regression	0.641129	0.855670	0.646753
Support Vector Machines	0.638105	0.853952	0.644617
Decision Trees	0.655242	0.814433	0.669492
Random Forest	0.655242	0.821306	0.667598
Naive Bayes	0.669355	0.829897	0.678371
K-Nearest Neighbor	0.608871	0.694158	0.657980

Bar Chart:

```
ax = df_model.plot.barh()
ax.legend(
    ncol=len(models.keys()),
    bbox_to_anchor=(0, 1),
    loc='lower left',
    prop={'size': 14}
)
plt.tight_layout()
```



REGRESSION

Regression analysis is a statistical technique used to investigate the association between a dependent variable and one or more independent variables. This technique is utilised in both machine learning and statistical modelling to make predictions about future events or outcomes based on the relationship established through the analysis.

Two types of regression include:

- **Simple Linear Regression**

  Simple linear regression is a statistical technique used to model the relationship between a dependent variable and a single independent variable. The relationship between the data points is used to fit a straight line that best reflects the relationship between the variables. The dependent variable's values can therefore be predicted using the line based on the values of the independent variable.

- **Multiple Linear Regression**

  Multiple regression extends the concept of linear regression that incorporates two or more independent variables to predict a value. In contrast to simple linear regression, multiple regression considers multiple predictors to create a more comprehensive predictive model. This technique aims to deepen the understanding of the relationship between the predictors and the dependent variable to produce a more insightful analysis.

Regression Algorithm

Implementing linear regression typically involves following several fundamental steps:

1. Import the necessary packages and classes required for the analysis.
2. Provide the relevant data and apply any necessary transformations to prepare it for analysis.
3. Split the dataset into **training sets** and **testing sets**.
4. Train the regression model and fit it using the available data.
5. Utilise the fitted model for making predictions.
6. Calculate the accuracy.
7. Print the outputs.

These steps are generally applicable across various regression approaches and implementations.

In this example, Simple Linear Algorithm and Multiple Linear Algorithm will be compared using the "forestfire_cleaned.csv" dataset.

X	Y	month	day	FFMC	DMC	DC	ISI	temp	RH	wind	rain	area	size_category
7	5	3	fri	86.2	26.2	94.3	5.1	8.2	51	6.7	0	0	0
7	4	10	tue	90.6	35.4	669.1	6.7	18	33	0.9	0	0	0
7	4	10	sat	90.6	43.7	686.9	6.7	14.6	33	1.3	0	0	0
8	6	3	fri	91.7	33.3	77.5	9	8.3	97	4	0.182322	0	0
8	6	3	sun	89.3	51.3	102.2	9.6	11.4	99	1.8	0	0	0
8	6	8	sun	92.3	85.3	488	14.7	22.2	29	5.4	0	0	0
8	6	8	mon	92.3	88.9	495.6	8.5	24.1	27	3.1	0	0	0
8	6	8	mon	91.5	145.4	608.2	10.7	8	86	2.2	0	0	0
8	6	9	tue	91	129.5	692.6	7	13.1	63	5.4	0	0	0
7	5	9	sat	92.5	88	698.6	7.1	22.8	40	4	0	0	0
7	5	9	sat	92.5	88	698.6	7.1	17.8	51	7.2	0	0	0
7	5	9	sat	92.8	73.2	713	22.6	19.3	38	4	0	0	0
6	5	8	fri	63.5	70.8	665.3	0.8	17	72	6.7	0	0	0
6	5	9	mon	90.9	126.5	686.5	7	21.3	42	2.2	0	0	0
6	5	9	wed	92.9	133.3	699.6	9.2	26.4	21	4.5	0	0	0
6	5	9	fri	93.3	141.2	713.9	13.9	22.9	44	5.4	0	0	0
5	5	3	sat	91.7	35.8	80.8	7.8	15.1	27	5.4	0	0	0
8	5	10	mon	84.9	32.8	664.2	3	16.7	47	4.9	0	0	0
6	4	3	wed	89.2	27.9	70.8	6.3	15.9	35	4	0	0	0
6	4	4	sat	86.3	27.4	97.1	5.1	9.3	44	4.5	0	0	0
6	4	9	tue	91	129.5	692.6	7	18.3	40	2.7	0	0	0
5	4	9	mon	91.8	78.5	724.3	9.2	19.1	38	2.7	0	0	0
7	4	6	sun	94.3	96.3	200	56.1	21	44	4.5	0	0	0
7	4	8	sat	90.2	110.9	537.4	6.2	19.5	43	5.8	0	0	0
7	4	8	sat	93.5	139.4	594.2	20.3	23.7	32	5.8	0	0	0
7	4	8	sun	91.4	142.4	601.4	10.6	16.3	60	5.4	0	0	0

Simple Linear Regression

1. Import the necessary packages and classes required for the analysis.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

   - `numpy` (imported as `np`) for numerical operations
   - `matplotlib.pyplot` (imported as `plt`) for data visualisation
   - `pandas` (imported as `pd`) for data manipulation

2. Provide the relevant data and apply any necessary transformations to prepare it for analysis.

```
# Load the dataset
dataset = pd.read_csv('forestfire_cleaned.csv')
X = dataset.iloc[:, 9].values.reshape(-1, 1) #Reshape to 2D array
y = dataset.iloc[:, 12].values
```

   The dataset is read from a CSV file called 'forestfire_cleaned.csv' using pd.read_csv().

   The code selects two specific columns for analysis: the 10th column (indexed as 9) and the 13th column (indexed as 12).

   The selected columns are stored in X and y, where

   - `X` represents the independent variable (RH - Relative Humidity)
   - `y` represents the dependent variable (Area - forest fire area).

3. Split the dataset into **training sets** and **testing sets**.

```
# Split the dataset into training and test sets
from sklearn.model_selection import train_test_split
X_train_simple, X_test_simple, y_train_simple, y_test_simple =
train_test_split(X, y, test_size=1/3, random_state=0)
```

- The `train_test_split` function from `sklearn.model_selection` is used to split X and y into **training** and **test** sets.
- The **test** set size is set to 1/3 of the total dataset using the `test_size parameter.`
    - It specifies that the **testing** dataset should be one-third (33.33%) of the entire dataset. This means that 2/3 of the data will be used for **training**.
- The random state is set to 0 to ensure that the same random split is generated each time the code is run and reproducible.

4. Train the regression model and fit it using the available data.

```
# Simple Linear Regression
from sklearn.linear_model import LinearRegression
simple_regressor = LinearRegression()
simple_regressor.fit(X_train_simple, y_train_simple)
```

The `LinearRegression` class from `sklearn.linear_model` is used to create an instance of the simple linear regression model.

The model is initialised as `simple_regressor`.

The `fit` method is called on `simple_regressor` with the training data (`X_train_simple` and `y_train_simple`) to train the model.

5. Utilise the fitted model for making predictions.

```
# Predict the values for Simple Linear Regression
y_pred_simple = simple_regressor.predict(X_test_simple)
```

The `predict` method is used to predict the values of `y` for the **test** set (`X_test_simple`), and the predictions are stored in `y_pred_simple`.

6. Calculate the accuracy.

```
# Calculate the accuracy metrics for Simple Linear Regression
from sklearn.metrics import mean_squared_error, r2_score
mse_simple = mean_squared_error(y_test_simple, y_pred_simple)
r2_simple = r2_score(y_test_simple, y_pred_simple)
```

The mean squared error (MSE) and the R-squared score are calculated using the `mean_squared_error` and `r2_score` functions from `sklearn.metrics`, respectively.

- Calculates the mean squared error with `y_test_simple` (actual target values) and `y_pred_simple` (predicted target values) as arguments.
  The resulting MSE is assigned to the variable `mse_simple`.

- Calculates the R-squared score with `y_test_simple` (actual target values) and `y_pred_simple` (predicted target values) as arguments.
  The resulting R-squared score is assigned to the variable `r2_simple`.

7. Print the outputs.

```
print("\nPredicted values for Simple Linear Regression:")
for i in range(len(y_pred_simple)):
print("Sample", i+1, "- Predicted:", y_pred_simple[i], "-
Actual:", y_test_simple[i])

plt.figure(figsize=(12, 8))
plt.subplots_adjust(hspace=0.5)

# Plot the predicted values and actual values
plt.scatter(X_test_simple, y_test_simple, color='blue',
label='Actual')
plt.plot(X_test_simple, y_pred_simple, color='red',
linewidth=2, label='Predicted')
plt.title(f'Simple Linear Regression: RH vs Area\nPredicted
probabilities')
plt.xlabel('RH (Relative Humidity)')
plt.ylabel('Area')
plt.legend()
plt.show()
plt.tight_layout()

# Print the accuracy metrics
print("\nSimple Linear Regression:")
print("Mean Squared Error:", mse_simple)
print("R-squared Score:", r2_simple)
```

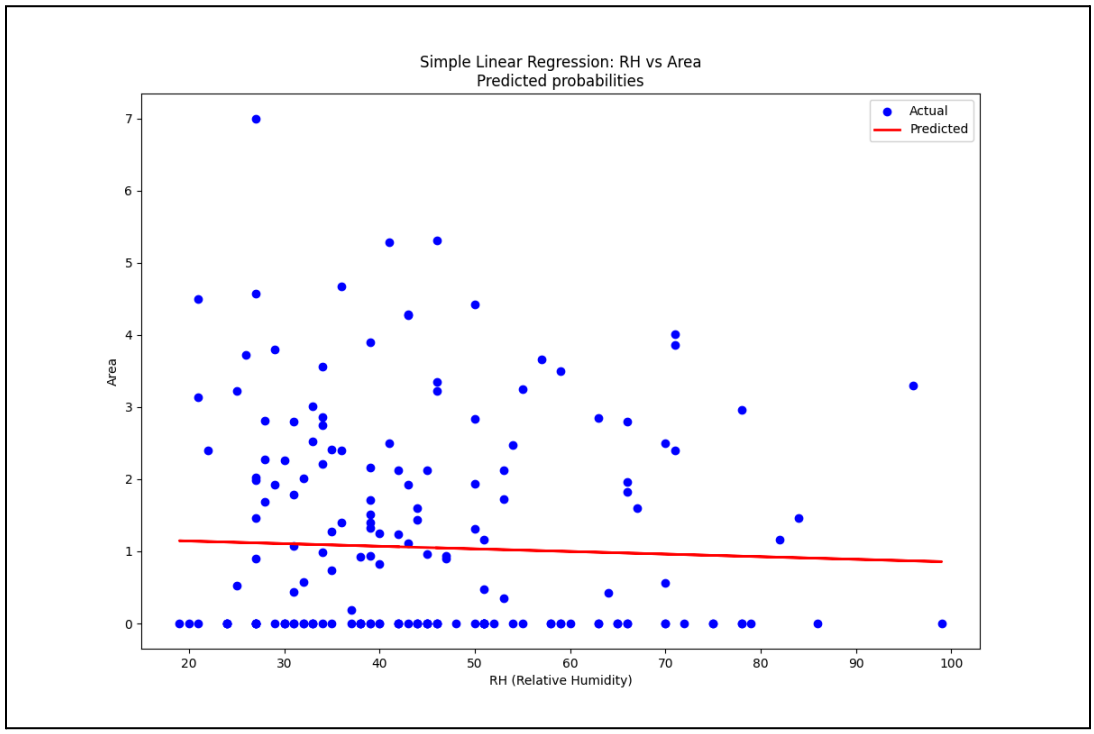
- Displaying the predicted values:
    - The code prints the predicted values and the actual values for each sample in the test set using a loop.

- Visualising the results:
    - A scatter plot is created with the actual values (blue dots) and the predicted values (red line) using `plt.scatter` and `plt.plot`, respectively.
    - The plot is titled 'Simple Linear Regression: RH vs Area' and labelled with appropriate axes.
    - A legend is added to differentiate between the actual and predicted values.
    - The plot is displayed using `plt.show()`.

- `plt.tight_layout()` function is used to automatically adjust the padding and spacing between subplots in a figure created

- Printing the accuracy metrics:
  - The mean squared error (MSE) and R-squared score are printed to evaluate the performance of the simple linear regression model.

Scatter plot:



Output:

```
Sample 168 - Predicted: 1.1389479050324665 - Actual: 0.0
Sample 169 - Predicted: 1.1063738726693166 - Actual: 0.0
Sample 170 - Predicted: 0.9435037108535684 - Actual: 0.0
Sample 171 - Predicted: 1.1136125465277944 - Actual: 2.2700619012884857
Sample 172 - Predicted: 1.1027545357400779 - Actual: 0.0
Sample 173 - Predicted: 1.0629418295184505 - Actual: 2.128231705849268

Simple Linear Regression:
Mean Squared Error: 2.2105447807404968
R-squared Score: -0.008475512084447079
```

Multiple Linear Regression

1. Import the necessary packages and classes required for the analysis.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

- `numpy` (imported as `np`) for numerical operations
- `matplotlib.pyplot` (imported as `plt`) for data visualisation
- `pandas` (imported as `pd`) for data manipulation

2. Provide the relevant data and apply any necessary transformations to prepare it for analysis.

```
# Load the dataset
dataset = pd.read_csv('forestfire_cleaned.csv')
X = dataset.iloc[:, [2, 4, 5, 6, 7, 9]].values
y = dataset.iloc[:, 12].values
```

The dataset is read from a CSV file called 'forestfire_cleaned.csv' using pd.read_csv().

The code selects specific columns (2, 4, 5, 6, 7, 9) for the independent variables (X) and column 12 as the dependent variable (y).

The selected columns are stored in X and y, where

- `X` represents the independent variable (Month, FFMC, DMC, DC, ISI, RH)
- `y` represents the dependent variable (Area - forest fire area).

3. Split the dataset into **training sets** and **testing sets**.

```
# Split the dataset into training and test sets for Multiple
Linear Regression
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

- The `train_test_split` function from `sklearn.model_selection` is used to split X and y into **training** and **test** sets.
- The random state is set to 42 to ensure reproducibility.
- The **test** set size is set to 0.2 of the total dataset using the `test_size parameter.`
    - It specifies that the **testing** dataset should be 20% of the entire dataset. This means that 80% of the data will be used for **training**.
- The random state is set to 42 to specify a random seed for reproducibility and ensure that the random splitting of the dataset is consistent across different runs of the code.

4. Train the regression model and fit it using the available data.

```
# Create and fit the Multiple Linear Regression model
from sklearn.linear_model import LinearRegression
multiple_regressor = LinearRegression()
multiple_regressor.fit(X_train, y_train)
```

The `LinearRegression` class from `sklearn.linear_model` is used to create an instance of the multiple linear regression model.

The model is initialised as `multiple_regressor`.

The `fit` method is called on `multiple_regressor` with the training data (`X_train` and `y_train`) to train the model.

5. Utilise the fitted model for making predictions.

```
# Predict the values for the test set
y_pred = multiple_regressor.predict(X_test)
```

The `predict` method is used to predict the values of `y` for the **test** set (`X_test`), and the predictions are stored in `y_pred`.

6. Calculate the accuracy.

```
# Calculate the accuracy metrics for Multiple Linear
Regression
from sklearn.metrics import mean_squared_error, r2_score
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

The mean squared error (MSE) and the R-squared score are calculated using the `mean_squared_error` and `r2_score` functions from `sklearn.metrics`, respectively.

- Calculates the mean squared error with `y_test` (actual target values) and `y_pred` (predicted target values) as arguments.
  The resulting MSE is assigned to the variable `mse`.

- Calculates the R-squared score with `y_test` (actual target values) and `y_pred` (predicted target values) as arguments.
  The resulting R-squared score is assigned to the variable `r2`.

7. Print the outputs.

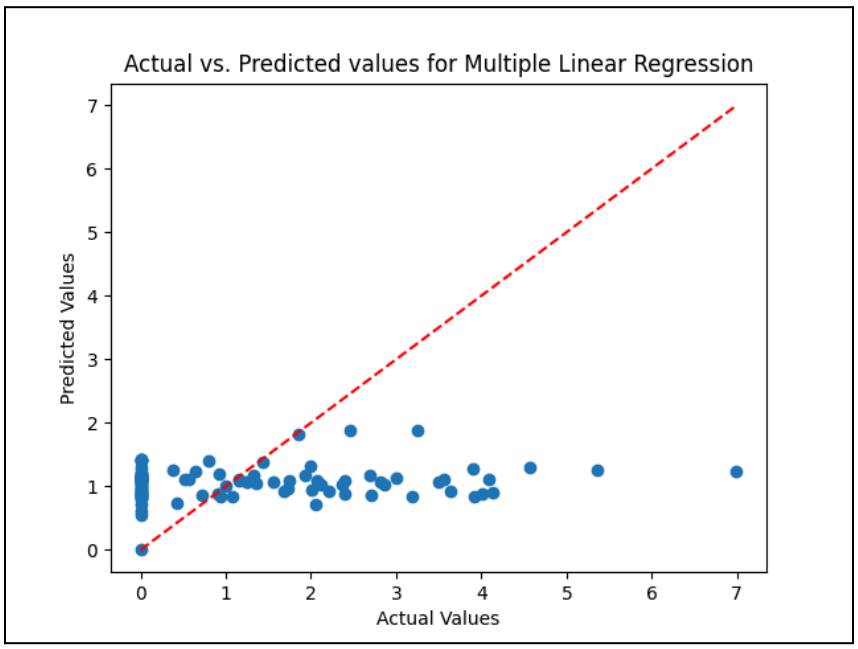
```
# Show the predicted values for Multiple Linear Regression
print("\nPredicted values for Multiple Linear Regression:")
for i in range(len(y_pred)):
print("Sample", i+1, "- Predicted:", y_pred[i], "- Actual:",
y_test[i])

# Plot the predicted values against the actual values
plt.scatter(y_test, y_pred)
plt.plot([min(y_test), max(y_test)], [min(y_test),
max(y_test)], color='red', linestyle='--')
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title("Actual vs. Predicted values for Multiple Linear
Regression")
plt.show()

# Print the accuracy metrics
print("\nMultiple Linear Regression:")
print("Mean Squared Error:", mse)
print("R-squared Score:", r2)
```

- Displaying the predicted values:
    - The code prints the predicted values and the actual values for each sample in the test set using a loop.

- Visualising the results:
    - A scatter plot is created with the actual values (y_test) on the x-axis and the predicted values (y_pred) on the y-axis using `plt.scatter`.
    - A diagonal line is plotted using `plt.plot` to represent the ideal case where the predicted and actual values are the same.
    - The plot is labelled with appropriate axes and given a title.
    - The plot is displayed using `plt.show()`.

- Printing the accuracy metrics:
    - The mean squared error (MSE) and R-squared score are printed to evaluate the performance of the simple linear regression model.

Scatter plot:



Output:

```
Sample 99 - Predicted: 1.428802144050388 - Actual: 0.0
Sample 100 - Predicted: 1.0657351371049748 - Actual: 1.2556160374777745
Sample 101 - Predicted: 1.8208092819184105 - Actual: 1.8531680973566984
Sample 102 - Predicted: 1.0946072360835666 - Actual: 1.1537315878891892
Sample 103 - Predicted: 0.8781436208793298 - Actual: 4.012592060349841
Sample 104 - Predicted: 1.1097840214783825 - Actual: 3.5655812377694427

Multiple Linear Regression:
Mean Squared Error: 2.1555901994408333
R-squared Score: 0.01923295511561174
```

Conclusion

Regression Model/Accuracy	Mean Squared Error (MSE)	R-squared Score
Simple Linear Regression	2.2105447807404968	-0.008475512084447079
Multiple Linear Regression	2.1555901994408333	0.01923295511561174

It can be seen that multiple linear regression is more accurate with a lower MSE value (2.1555901994408333) compared to simple linear regression's MSE value (2.2105447807404968).

The multiple linear regression is more suitable for this data as its R-squared value is 0.01923295511561174 compared to the simple linear regression which has a negative R-squared value (-0.008475512084447079). A negative R-squared value suggests that the chosen model is not a good fit for the data and is performing worse than a baseline model.

CLUSTERING

Dataset is taken from a preprocessed CSV file named ‘cleaned.CSV’ with the following example. Clustering tasks will be conducted between the data of the value of “hours-per-week” and the value of “capital-gains”.

	B	C	D	E	F	G	H	I	J	K	L	M	N
1	workclass	education	ucation-nu	arital-stat	ccupation	elationshi	race	sex	capital-gain	capital-loss	urs-per-w	native-country	class
2	State-gov	Bachelors	13	Never-ma	Adm-cleri	Not-in-fa	White	0	2174	0	40	United-States	<=50K
3	Private	Masters	14	Never-ma	Prof-spec	Not-in-fa	White	1	14084	0	50	United-States	>50K
4	Private	Bachelors	13	Married-c	Exec-man	Husband	White	0	5178	0	40	United-States	>50K
5	Private	11th	7	Married-c	Transport	Husband	White	0	0	2042	40	United-States	<=50K
6	Private	Bachelors	13	Divorced	Exec-man	Own-chil	White	0	0	1408	40	United-States	<=50K
7	Private	Prof-school	15	Married-c	Prof-spec	Wife	White	1	0	1902	60	Honduras	>50K
8	Private	HS-grad	9	Married-c	Machine-	Husband	White	0	5013	0	40	United-States	<=50K
9	Private	Bachelors	13	Married-c	Sales	Husband	White	0	2407	0	40	United-States	<=50K
10	Private	HS-grad	9	Divorced	Craft-repa	Not-in-fa	White	1	14344	0	40	United-States	>50K
11	Self-emp-nc	Doctorate	16	Married-c	Prof-spec	Husband	White	0	0	1902	60	United-States	>50K
12	Private	Bachelors	13	Married-c	Exec-man	Husband	White	0	15024	0	60	United-States	>50K
13	Self-emp-in	HS-grad	9	Married-c	Craft-repa	Husband	White	0	7688	0	40	United-States	>50K
14	Self-emp-nc	HS-grad	9	Married-c	Other-ser	Husband	White	0	0	1887	50	Canada	>50K
15	Private	Some-colleg	10	Married-c	Sales	Wife	White	1	4064	0	25	United-States	<=50K
16	Private	Some-colleg	10	Never-ma	Adm-cleri	Own-chil	White	1	0	1719	28	United-States	<=50K
17	Private	HS-grad	9	Never-ma	Other-ser	Not-in-fa	White	1	0	1762	40	United-States	<=50K
18	Self-emp-nc	HS-grad	9	Married-c	Craft-repa	Husband	White	0	4386	0	35	United-States	<=50K
19	Private	Assoc-voc	11	Never-ma	Prof-spec	Not-in-fa	White	1	0	1564	40	United-States	>50K
20	Private	11th	7	Married-c	Craft-repa	Husband	White	0	0	2179	40	United-States	<=50K
21	Self-emp-nc	Some-colleg	10	Separated	Sales	Unmarrie	Black	0	0	1816	2	United-States	<=50K

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
151	54	Self-emp-in	Prof-school	15	Married-c	Prof-spec	Husband	White	0	99999	0	60	United-States	>50K
152	51	Self-emp-in	Masters	14	Married-c	Exec-man	Husband	White	0	15024	0	50	United-States	>50K
153	28	Self-emp-nc	Some-colleg	10	Never-ma	Transport	Not-in-fa	White	0	0	1762	40	United-States	<=50K
154	38	Private	Bachelors	13	Married-c	Prof-spec	Husband	White	0	0	1887	40	United-States	>50K
155	46	Private	Bachelors	13	Married-c	Sales	Husband	White	0	0	1977	40	United-States	>50K
156	67	Private	Bachelors	13	Married-c	Protective	Husband	White	0	6514	0	7	United-States	>50K
157	52	Private	HS-grad	9	Never-ma	Sales	Not-in-fa	White	0	0	1876	50	United-States	<=50K
158	50	Private	HS-grad	9	Married-c	Adm-cleri	Wife	White	1	5178	0	40	United-States	>50K
159	47	Self-emp-nc	Bachelors	13	Married-c	Sales	Husband	White	0	15024	0	50	United-States	>50K
160	26	Private	5th-6th	3	Married-c	Craft-repa	Husband	White	0	0	1628	50	United-States	<=50K
161	35	Private	Doctorate	16	Married-c	Prof-spec	Husband	White	0	7298	0	40	United-States	>50K
162	61	Local-gov	HS-grad	9	Widowed	Adm-cleri	Unmarrie	White	1	1471	0	35	United-States	<=50K
163	33	Private	Some-colleg	10	Never-ma	Prof-spec	Not-in-fa	White	1	3674	0	16	United-States	<=50K
164	71	Private	Bachelors	13	Divorced	Tech-supp	Own-chil	White	1	2329	0	16	United-States	<=50K
165	32	Self-emp-in	Bachelors	13	Married-c	Craft-repa	Husband	White	0	7688	0	50	United-States	>50K
166	61	Self-emp-nc	HS-grad	9	Married-c	Farming-f	Husband	White	0	0	1848	60	United-States	>50K
167	52	Private	HS-grad	9	Married-c	Exec-man	Husband	Asian-Pac	0	99999	0	40	Japan	>50K
168	65	Private	Masters	14	Married-c	Sales	Husband	White	0	20051	0	40	United-States	>50K
169	40	Self-emp-nc	Some-colleg	10	Divorced	Exec-man	Other-rel	White	0	0	1564	70	Iran	>50K
170	59	Private	HS-grad	9	Divorced	Sales	Unmarrie	White	1	0	1762	30	United-States	<=50K
171	54	Private	HS-grad	9	Married-c	Transport	Husband	Black	0	5013	0	40	United-States	<=50K

First, import the .csv files into the python by creating a new python file and the following code is typed.

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

# Load the CSV file into a DataFrame
df = pd.read_csv("adult_cleaned.csv")
```

Next, only two columns are read for the clustering purpose which are the value of "hours-per-week" and the value of "capital-gains" by retrieving the column by their name in the csv file and assigning it to a variable.

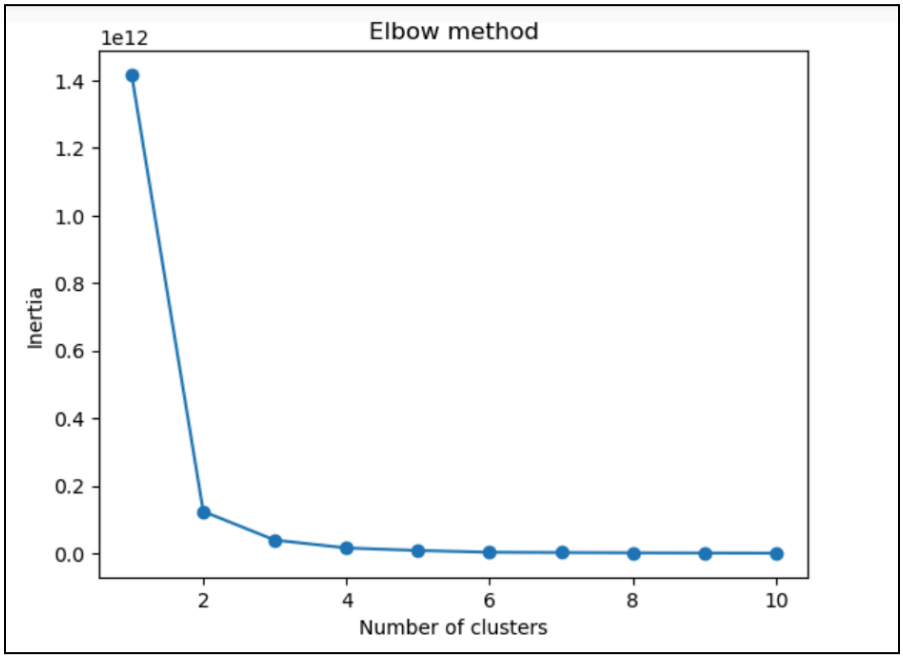
```
data = pd.DataFrame()
data['hours-per-week'] = df['hours-per-week']
data['capital-gain'] = df['capital-gain']
```

Elbow Method

The Elbow method graphs the inertia (a distance-based metric) and visualises the point at which **it starts decreasing linearly**. This point is referred to as the "elbow" and is a good estimate for the best value for K based on our data. Choose the optimum number for each value K in range(1,11) by using this code.

```
inertias = []

for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, n_init=10)
    kmeans.fit(data)
    inertias.append(kmeans.inertia_)

plt.plot(range(1, 11), inertias, marker='o')
plt.title('Elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.show()
```

So this will generate a graph like the below.

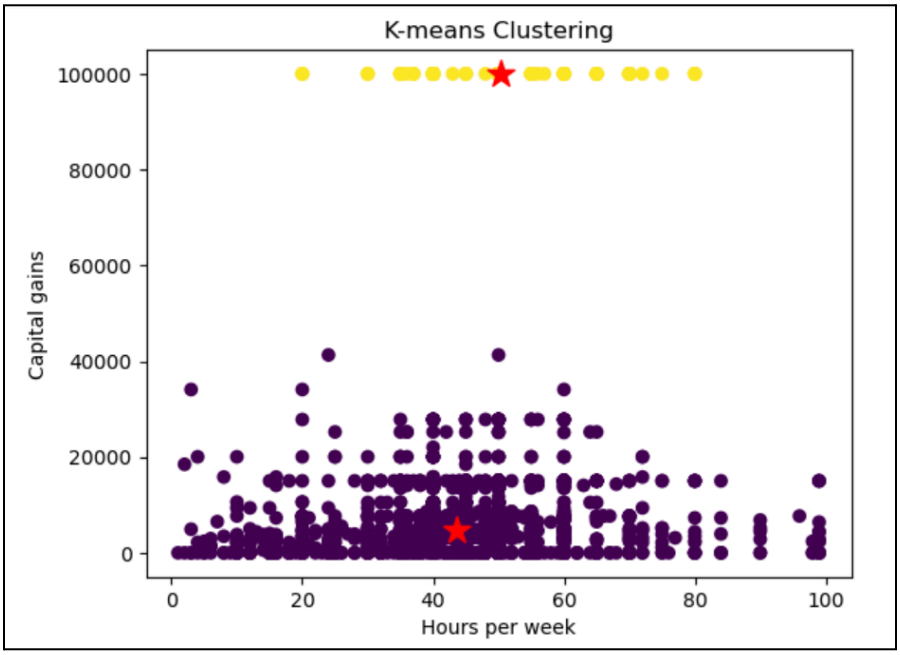


In the above plot, there is a sharp fall of average distance at **k=2.** So, this value of K to be used in the clustering like the following code

```
# Perform clustering using K-means
k = 2  # Number of clusters
kmeans = KMeans(n_clusters=k, n_init = 10)
kmeans.fit(data)
```

Next, get the value of centroids and the label for the graph of the new clustering. Lastly, visualise the graph with a complete label for each axis and colour of every cluster by using the following code

```
# Get the cluster labels and centroids
labels = kmeans.labels_
centroids = kmeans.cluster_centers_


# Visualize the clusters
plt.scatter(x, y, c=labels, cmap='viridis')
plt.scatter(centroids[:, 0], centroids[:, 1], marker='*', s=200, color='red')
plt.xlabel('Hours per week')
plt.ylabel('Capital gains')
plt.title('K-means Clustering')
plt.show()
```



Silhouette Analysis Method

First, the value inertia[] is created to make an empty list to store the inertia values for different values of k and the value of range for k from 2 until 11.

```
k_values = range(2, 11)
silhouette_scores = []
```

Next, every k value is iterated to find its silhouette score or coefficient and the values for each k will be printed using the following code.

```
for k in k_values:
    # Perform K-means clustering
    kmeans = KMeans(n_clusters=k, n_init=10)
    labels = kmeans.fit_predict(data)

    # Compute the silhouette score
    silhouette_avg = silhouette_score(data, labels)
    silhouette_scores.append(silhouette_avg)
    print(f"Silhouette score for k={k}: {silhouette_avg}")
```

```
Silhouette score for k=2: 0.9392769585096182
Silhouette score for k=3: 0.7265297677515004
Silhouette score for k=4: 0.707363132664571
Silhouette score for k=5: 0.7218567145533205
Silhouette score for k=6: 0.8070474145478245
Silhouette score for k=7: 0.813545405308744
Silhouette score for k=8: 0.8219625329194801
Silhouette score for k=9: 0.8351844009138792
Silhouette score for k=10: 0.836783396179824
```

After that, value k with the most coefficient score will be printed along with the coefficient value

```
best_k = k_values[silhouette_scores.index(max(silhouette_scores))]
best_score = max(silhouette_scores)

print("Silhouette scores for all k values:")
for k, score in zip(k_values, silhouette_scores):
    print(f"k={k}: {score}")

print("Best value of k:", best_k)
print("Best silhouette score:", best_score)
```

```
Silhouette scores for all k values:
k=2: 0.9392769585096182
k=3: 0.7265297677515004
k=4: 0.707363132664571
k=5: 0.7218567145533205
k=6: 0.8070474145478245
k=7: 0.813545405308744
k=8: 0.8219625329194801
k=9: 0.8351844009138792
k=10: 0.836783396179824
Best value of k: 2
Best silhouette score: 0.9392769585096182
```

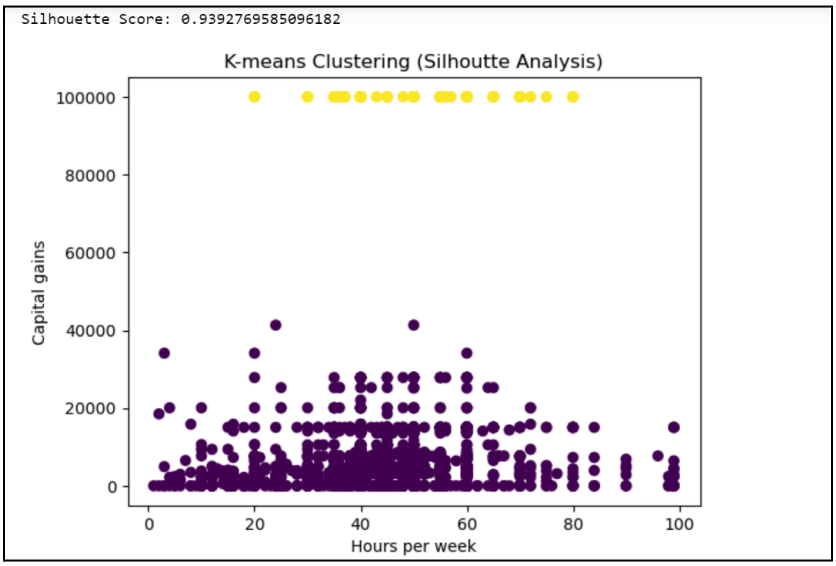
Following output terminal shows the value of the coefficient for all k with the highest k value. So, the highest coefficient value is 2 and it will be chosen as the k value for the clustering format by using the following code.

```
# Perform clustering using K-means
k = 2  # Number of clusters
kmeans = KMeans(n_clusters=k, n_init=10)
kmeans.fit(data)
```

Lastly, the graph is visualised with a complete label for each axis and colour of every cluster by using the following code.

```
# Get the cluster labels
labels = kmeans.labels_

# Compute the silhouette score
silhouette_avg = silhouette_score(data, labels)

print(f"Silhouette Score: {silhouette_avg}")

# Visualize the clusters
plt.scatter(data['hours-per-week'] , data['capital-gain'], c=labels, cmap='viridis')
plt.xlabel('Hours per week')
plt.ylabel('Capital gains')
plt.title('K-means Clustering (Silhoutte Analysis)')
plt.show()
```

```
Silhouette Score: 0.9392769585096182
```



Hierarchical Clustering

Dataset is taken from a preprocessed CSV file named ‘cleaned.CSV’ with the following example. Clustering tasks will be conducted between the data of the value of “hours-per-week” and the value of “capital-gains”.

	B	C	D	E	F	G	H	I	J	K	L	M	N
1	workclass	education	ucation-nu	arital-stat	occupation	elationshi	race	sex	capital-gain	capital-loss	urs-per-w	native-country	class
2	State-gov	Bachelors	13	Never-ma	Adm-cleri	Not-in-fa	White	0	2174	0	40	United-States	<=50K
3	Private	Masters	14	Never-ma	Prof-spec	Not-in-fa	White	1	14084	0	50	United-States	>50K
4	Private	Bachelors	13	Married-c	Exec-man	Husband	White	0	5178	0	40	United-States	>50K
5	Private	11th	7	Married-c	Transport	Husband	White	0	0	2042	40	United-States	<=50K
6	Private	Bachelors	13	Divorced	Exec-man	Own-chil	White	0	0	1408	40	United-States	<=50K
7	Private	Prof-school	15	Married-c	Prof-spec	Wife	White	1	0	1902	60	Honduras	>50K
8	Private	HS-grad	9	Married-c	Machine-	Husband	White	0	5013	0	40	United-States	<=50K
9	Private	Bachelors	13	Married-c	Sales	Husband	White	0	2407	0	40	United-States	<=50K
10	Private	HS-grad	9	Divorced	Craft-repa	Not-in-fa	White	1	14344	0	40	United-States	>50K
11	Self-emp-nc	Doctorate	16	Married-c	Prof-spec	Husband	White	0	0	1902	60	United-States	>50K
12	Private	Bachelors	13	Married-c	Exec-man	Husband	White	0	15024	0	60	United-States	>50K
13	Self-emp-in	HS-grad	9	Married-c	Craft-repa	Husband	White	0	7688	0	40	United-States	>50K
14	Self-emp-nc	HS-grad	9	Married-c	Other-ser	Husband	White	0	0	1887	50	Canada	>50K
15	Private	Some-colleg	10	Married-c	Sales	Wife	White	1	4064	0	25	United-States	<=50K
16	Private	Some-colleg	10	Never-ma	Adm-cleri	Own-chil	White	1	0	1719	28	United-States	<=50K
17	Private	HS-grad	9	Never-ma	Other-ser	Not-in-fa	White	1	0	1762	40	United-States	<=50K
18	Self-emp-nc	HS-grad	9	Married-c	Craft-repa	Husband	White	0	4386	0	35	United-States	<=50K
19	Private	Assoc-voc	11	Never-ma	Prof-spec	Not-in-fa	White	1	0	1564	40	United-States	>50K
20	Private	11th	7	Married-c	Craft-repa	Husband	White	0	0	2179	40	United-States	<=50K
21	Self-emp-nc	Some-colleg	10	Separatec	Sales	Unmarrie	Black	0	0	1816	2	United-States	<=50K

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
151	54	Self-emp-in	Prof-school	15	Married-c	Prof-spec	Husband	White	0	99999	0	60	United-States	>50K
152	51	Self-emp-in	Masters	14	Married-c	Exec-man	Husband	White	0	15024	0	50	United-States	>50K
153	28	Self-emp-nc	Some-colleg	10	Never-ma	Transport	Not-in-fa	White	0	0	1762	40	United-States	<=50K
154	38	Private	Bachelors	13	Married-c	Prof-spec	Husband	White	0	0	1887	40	United-States	>50K
155	46	Private	Bachelors	13	Married-c	Sales	Husband	White	0	0	1977	40	United-States	>50K
156	67	Private	Bachelors	13	Married-c	Protective	Husband	White	0	6514	0	7	United-States	>50K
157	52	Private	HS-grad	9	Never-ma	Sales	Not-in-fa	White	0	0	1876	50	United-States	<=50K
158	50	Private	HS-grad	9	Married-c	Adm-cleri	Wife	White	1	5178	0	40	United-States	>50K
159	47	Self-emp-nc	Bachelors	13	Married-c	Sales	Husband	White	0	15024	0	50	United-States	>50K
160	26	Private	5th-6th	3	Married-c	Craft-repa	Husband	White	0	0	1628	50	United-States	<=50K
161	35	Private	Doctorate	16	Married-c	Prof-spec	Husband	White	0	7298	0	40	United-States	>50K
162	61	Local-gov	HS-grad	9	Widowed	Adm-cleri	Unmarrie	White	1	1471	0	35	United-States	<=50K
163	33	Private	Some-colleg	10	Never-ma	Prof-spec	Not-in-fa	White	1	3674	0	16	United-States	<=50K
164	71	Private	Bachelors	13	Divorced	Tech-supp	Own-chil	White	1	2329	0	16	United-States	<=50K
165	32	Self-emp-in	Bachelors	13	Married-c	Craft-repa	Husband	White	0	7688	0	50	United-States	>50K
166	61	Self-emp-nc	HS-grad	9	Married-c	Farming-f	Husband	White	0	0	1848	60	United-States	>50K
167	52	Private	HS-grad	9	Married-c	Exec-man	Husband	Asian-Pac	0	99999	0	40	Japan	>50K
168	65	Private	Masters	14	Married-c	Sales	Husband	White	0	20051	0	40	United-States	>50K
169	40	Self-emp-nc	Some-colleg	10	Divorced	Exec-man	Other-rel	White	0	0	1564	70	Iran	>50K
170	59	Private	HS-grad	9	Divorced	Sales	Unmarrie	White	1	0	1762	30	United-States	<=50K
171	54	Private	HS-grad	9	Married-c	Transport	Husband	Black	0	5013	0	40	United-States	<=50K

First, the .csv files will be imported into the python by creating a new python file and type the following code.

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

# Load the CSV file into a DataFrame
df = pd.read_csv("adult_cleaned.csv")
```

Next, only two columns will be read for the clustering purpose which are the value of "hours-per-week" and the value of "capital-gains" by retrieving the column by their name in the csv file and assigning it to a variable.

```
data = pd.DataFrame()
data['hours-per-week'] = df['hours-per-week']
data['capital-gain'] = df['capital-gain']
```

Next, make a list of tuples that combine from both data variables or elements.

```
data_tuple = list(zip(data['hours-per-week'],data['capital-gain']))
```

After that, hierarchical clustering will be used by using ward linkage functions like the following code. The ward linkage is computed among the various points using a basic Euclidean distance metric and Ward's linkage method, which aims to reduce the variance between clusters.

```
# Perform hierarchical clustering
dendrogram = sch.dendrogram(sch.linkage(data_tuple, method='ward'))
```

Lastly, the graph will be visualised with a complete label for each axis and colour of every cluster by using the following code. The result is plotted in a dendrogram. This plot will show the hierarchy of clusters from the bottom (individual points) to the top (a single cluster consisting of all data points).

```
plt.xlabel('Data Points')
plt.ylabel('Euclidean Distances')
plt.title('Hierarchical Clustering Dendrogram')
plt.show()
```

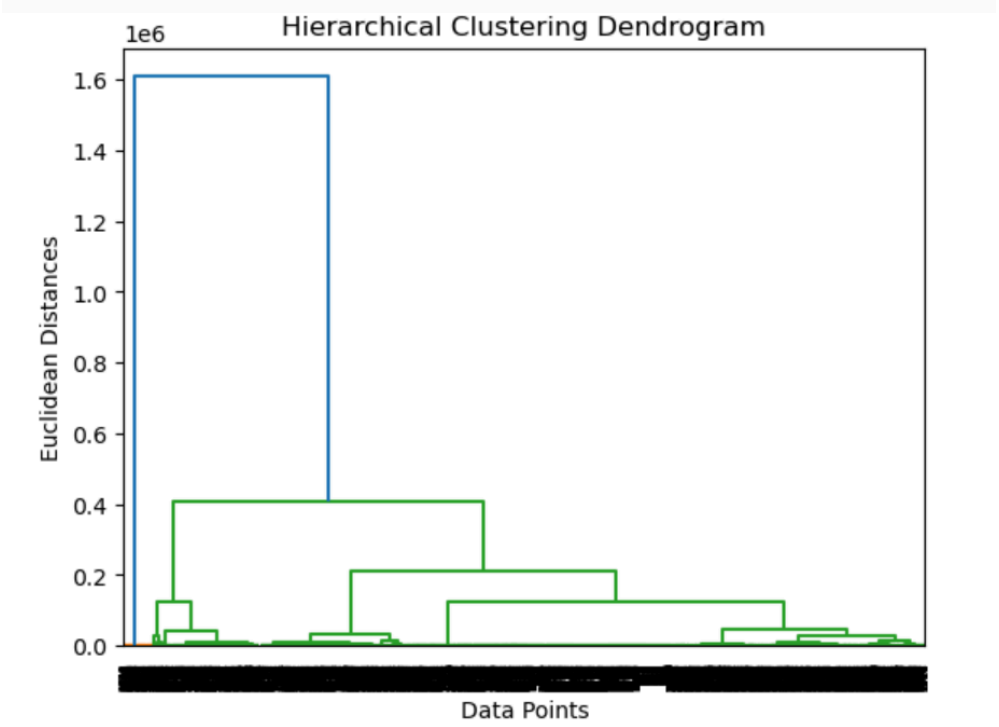


Figure 4.2.1.2.3 Hierarchical Plot

PREDICTION

Prediction is a way to guess what will happen in the future based on patterns in current data. It uses machine learning and statistical techniques to identify patterns and then make a conclusion about what the future value will be. Prediction is a powerful tool that can be used in a variety of settings. It can be used to forecast sales, predict customer behavior, and even make medical diagnoses. The accuracy of the prediction depends on the quality of the data that is used. The more data that is available, the more accurate the predictions will be.

There are two algorithms that will be performed for this prediction:-

- **Logistic Regression**

  Logistic Regression is an algorithm to predict the binary value of the data. It is a type of regression analysis that models the relationship between a dependent variable (the binary outcome) and one or more independent variables (also known as predictor or explanatory variables). In this case, it will predict the value of whether the forest experiencing large fire or a small fire.

- **Decision Tree**

  Decision Tree is an algorithm that uses the tree structure to make decisions based on multiple conditions or features. It used supervised learning algorithm such as regression and classification. It will take dependent variable (features) and create the conditions of it to predict the dependent variable based on different features given.

Prediction Algorithm

Implementing the prediction algorithm requires several repeated steps for both algorithms:

1. Import the necessary packages and classes required for the analysis.
2. Provide the relevant data and apply any necessary transformations to prepare it for analysis.
3. Split the dataset into **training sets** and **testing sets**.
4. Train the prediction model and fit it using the available data.
5. Utilise the fitted model for making predictions.
6. Calculate the accuracy of the predicted value.
7. Create confusion matrix
8. Print and visualize the outputs of confusion matrix and predicted value.

These steps are generally applicable across various prediction algorithms.

In this example, Logistic Regression and Decision Tree will be compared using the "forestfire_cleaned.csv" dataset.

	X	Y	month	day	FFMC	DMC	DC	ISI	temp	RH	wind	rain	area	size_category
2	7	5	3	fri	86.2	26.2	94.3	5.1	8.2	51	6.7	0	0	0
3	7	4	10	tue	90.6	35.4	669.1	6.7	18	33	0.9	0	0	0
4	7	4	10	sat	90.6	43.7	686.9	6.7	14.6	33	1.3	0	0	0
5	8	6	3	fri	91.7	33.3	77.5	9	8.3	97	4	0.182322	0	0
6	8	6	3	sun	89.3	51.3	102.2	9.6	11.4	99	1.8	0	0	0
7	8	6	8	sun	92.3	85.3	488	14.7	22.2	29	5.4	0	0	0
8	8	6	8	mon	92.3	88.9	495.6	8.5	24.1	27	3.1	0	0	0
9	8	6	8	mon	91.5	145.4	608.2	10.7	8	86	2.2	0	0	0
10	8	6	9	tue	91	129.5	692.6	7	13.1	63	5.4	0	0	0
11	7	5	9	sat	92.5	88	698.6	7.1	22.8	40	4	0	0	0
12	7	5	9	sat	92.5	88	698.6	7.1	17.8	51	7.2	0	0	0
13	7	5	9	sat	92.8	73.2	713	22.6	19.3	38	4	0	0	0
14	6	5	8	fri	63.5	70.8	665.3	0.8	17	72	6.7	0	0	0
15	6	5	9	mon	90.9	126.5	686.5	7	21.3	42	2.2	0	0	0
16	6	5	9	wed	92.9	133.3	699.6	9.2	26.4	21	4.5	0	0	0
17	6	5	9	fri	93.3	141.2	713.9	13.9	22.9	44	5.4	0	0	0
18	5	5	3	sat	91.7	35.8	80.8	7.8	15.1	27	5.4	0	0	0
19	8	5	10	mon	84.9	32.8	664.2	3	16.7	47	4.9	0	0	0
20	6	4	3	wed	89.2	27.9	70.8	6.3	15.9	35	4	0	0	0
21	6	4	4	sat	86.3	27.4	97.1	5.1	9.3	44	4.5	0	0	0
22	6	4	9	tue	91	129.5	692.6	7	18.3	40	2.7	0	0	0
23	5	4	9	mon	91.8	78.5	724.3	9.2	19.1	38	2.7	0	0	0
24	7	4	6	sun	94.3	96.3	200	56.1	21	44	4.5	0	0	0
25	7	4	8	sat	90.2	110.9	537.4	6.2	19.5	43	5.8	0	0	0
26	7	4	8	sat	93.5	139.4	594.2	20.3	23.7	32	5.8	0	0	0
27	7	4	8	sun	91.4	142.4	601.4	10.6	16.3	60	5.4	0	0	0

Logistic Regression

1. Import the necessary packages and classes required for the analysis.

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn import metrics
```

- import pandas as pd: Imports the Pandas library, which provides data manipulation and analysis tools.
- import matplotlib.pyplot as plt: Imports the Matplotlib library, which allows for data visualization.
- from sklearn.model_selection import train_test_split: Imports the train_test_split function from the model_selection module of Scikit-learn. It is used to split the data into training and testing sets.
- from sklearn.linear_model import LogisticRegression: Imports the Logistic Regression model from the linear_model module of Scikit-learn. Logistic Regression is a popular classification algorithm used for binary classification problems.
- from sklearn.metrics import accuracy_score, confusion_matrix: Imports the accuracy_score and confusion_matrix functions from the metrics module of Scikit-learn. These functions are used to evaluate the performance of the classification model.
- from sklearn import metrics: Imports the metrics module from Scikit-learn, which provides various metrics for evaluating the performance of machine learning models.

2. Provide the relevant data and apply any necessary transformations to prepare it for analysis.

```
# Load the dataset
data = pd.read_csv("forestfire_cleaned.csv") # Replace
"your_dataset.csv" with the actual file name

# Select the independent variables (X) and the dependent variable (Y)
X = data[['FFMC', 'DMC', 'DC', 'ISI', 'temp', 'RH', 'wind', 'rain']]
Y = data['size_category']
```

The dataset will be read from csv file named ‘forestfire_cleaned.csv’ using the pd.read_csv function. Then it will store into data variables.

- X = data[['FFMC', 'DMC', 'DC', 'ISI', 'temp', 'RH', 'wind', 'rain']]: Selects the independent variables (features) from the dataset and assigns them to the variable X. The selected features are 'FFMC', 'DMC', 'DC', 'ISI', 'temp', 'RH', 'wind', and 'rain'.
- Y = data['size_category']: Selects the dependent variable (target) from the dataset and assigns it to the variable Y. The dependent variable in this case is 'size_category'.

3. Split the dataset into **training sets** and **testing sets**.

```
# Split the data into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,
random_state=42)
```

The data will be split into data training and testing to make sure the model is successful.

- X_train: The training set of independent variables.
- X_test: The testing set of independent variables.
- Y_train: The training set of dependent variable values.
- Y_test: The testing set of dependent variable values.
- test_size = 0.2: it will take 20% of the data as test set
- random_state =42: ensure the reproducibility when split the data

4. Train the prediction model and fit it using the available data.

```
# Create and fit the logistic regression model
model = LogisticRegression(max_iter=1000000)
model.fit(X_train, Y_train)
```

- model = LogisticRegression(max_iter=1000000): Creates an instance of the logistic regression model using the LogisticRegression class from Scikit-learn. The max_iter parameter is set to a large value (1000000) to ensure that the optimization algorithm converges.
- model.fit(X_train, Y_train): Fits the logistic regression model to the training data. The X_train parameter represents the independent variables of the training set, and Y_train represents the corresponding dependent variable values. The fit method trains the model using the provided data.

5. Utilise the fitted model for making predictions

```
# Make predictions on the test set
Y_pred = model.predict(X_test)
```

- Y_pred = model.predict(X_test): Uses the predict method of the logistic regression model to generate predictions for the independent variables X_test. The predicted values are stored in the Y_pred variable.

6. Calculate the accuracy of the predicted value

```
# Evaluate accuracy
accuracy = accuracy_score(Y_test, Y_pred)
print("Accuracy:", accuracy)
```

- accuracy = accuracy_score(Y_test, Y_pred): Uses the accuracy_score function from Scikit-learn's metrics module to calculate the accuracy of the model's predictions. The Y_test parameter represents the actual values of the dependent variable in the test set, and Y_pred represents the predicted values.
- print("Accuracy:", accuracy): Prints the calculated accuracy score to the console.

7. Create confusion matrix

```
# Create a confusion matrix
confusion_mat = confusion_matrix(Y_test, Y_pred)
print("Confusion Matrix:")
print(confusion_mat)
```

- confusion_mat = confusion_matrix(Y_test, Y_pred): Uses the confusion_matrix function from Scikit-learn's metrics module to compute the confusion matrix. The Y_test parameter represents the actual values of the dependent variable in the test set, and Y_pred represents the predicted values.
- print("Confusion Matrix:"): Prints a header indicating that the following output is the confusion matrix.
- print(confusion_mat): Prints the confusion matrix to the console.

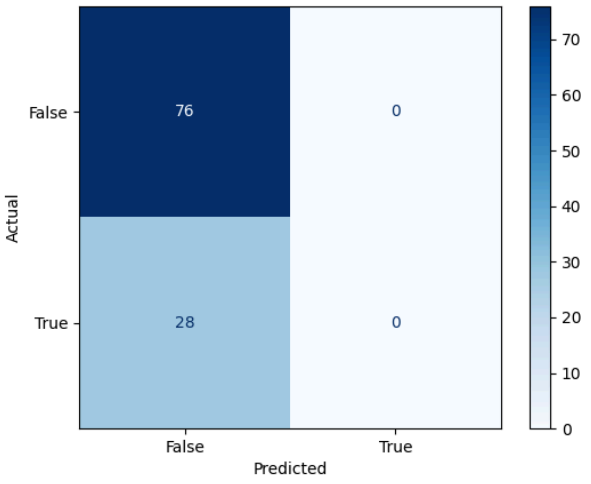
8. Print and visualize the outputs of confusion matrix and predicted value

```
# Visualize the confusion matrix with blue color and labeled axes
cm_display = metrics.ConfusionMatrixDisplay(confusion_mat,
display_labels=[False, True])
cm_display.plot(cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

- cm_display=metrics.ConfusionMatrixDisplay(confusion_mat, display_labels=[False, True]): Creates a ConfusionMatrixDisplay object from Scikit-learn's metrics module. The confusion_mat parameter represents the confusion matrix computed earlier, and the display_labels parameter specifies the labels to be shown on the x-axis and y-axis of the plot.
- cm_display.plot(cmap='Blues'): Plots the confusion matrix using the plot method of the ConfusionMatrixDisplay object. The cmap='Blues' parameter sets the color scheme to shades of blue.
- plt.xlabel('Predicted'): Sets the label for the x-axis of the plot as "Predicted".
- plt.ylabel('Actual'): Sets the label for the y-axis of the plot as "Actual".
- plt.show(): Displays the plot.

Output

a) Confusion Matrix



b) Result

```
Accuracy: 0.7307692307692307
Confusion Matrix:
[[76  0]
 [28  0]]
```

Decision Tree

1. Import the necessary packages and classes required for the analysis.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
```

- import pandas as pd: Imports the Pandas library, which provides data manipulation and analysis tools.
- from sklearn.model_selection import train_test_split: Imports the train_test_split function from the model_selection module of Scikit-learn. It is used to split the data into training and testing sets.
- from sklearn.tree import DecisionTreeClassifier: Imports the Decision Tree Classifier from the tree module of Scikit-learn. Decision trees are supervised machine learning models used for classification and regression tasks.
- from sklearn.metrics import accuracy_score, confusion_matrix: Imports the accuracy_score and confusion_matrix functions from the metrics module of Scikit-learn. These functions are used to evaluate the performance of the classification model.
- import seaborn as sns: Imports the Seaborn library, which provides enhanced data visualization capabilities.
- import matplotlib.pyplot as plt: Imports the Matplotlib library for data visualization.

2. Provide the relevant data and apply any necessary transformations to prepare it for analysis.

```
# Load the dataset
data = pd.read_csv("forestfire_cleaned.csv") # Replace
"your_dataset.csv" with the actual file name

# Select the independent variables (X) and the dependent variable (Y)
X = data[['FFMC', 'DMC', 'DC', 'ISI', 'temp', 'RH', 'wind', 'rain']]
Y = data['size_category']
```

- data = pd.read_csv("forestfire_cleaned.csv"): Loads the dataset from a CSV file named "forestfire_cleaned.csv" using the Pandas library. The dataset is stored in the data variable.
- X = data[['FFMC', 'DMC', 'DC', 'ISI', 'temp', 'RH', 'wind', 'rain']]: Selects the independent variables (features) from the dataset and assigns them to the variable X. The selected features are 'FFMC', 'DMC', 'DC', 'ISI', 'temp', 'RH', 'wind', and 'rain'.
- Y = data['size_category']: Selects the dependent variable (target) from the dataset and assigns it to the variable Y. The dependent variable in this case is 'size_category'.

3. Split the dataset into **training sets** and **testing sets**.

```
# Split the data into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,
random_state=42)
```

- X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42): Splits the dataset into training and testing sets using the train_test_split function from Scikit-learn. The X and Y variables represent the independent and dependent variables, respectively. The test_size parameter specifies the proportion of the dataset that should be allocated for testing (in this case, 20% of the data). The random_state parameter is used to ensure reproducibility of the split, so the same split can be obtained each time the code is executed.

4. Train the prediction model and fit it using the available data.

```
# Create and fit the decision tree model
model = DecisionTreeClassifier()
model.fit(X_train, Y_train)
```

- model = DecisionTreeClassifier(): Creates an instance of the decision tree classifier using the DecisionTreeClassifier class from Scikit-learn. By default, the decision tree classifier uses the Gini impurity as the criterion for splitting.
- model.fit(X_train, Y_train): Fits the decision tree model to the training data. The X_train parameter represents the independent variables of the training set, and Y_train represents the corresponding dependent variable values. The fit method trains the model using the provided data.

5. Utilise the fitted model for making predictions.

```
# Make predictions on the test set
Y_pred = model.predict(X_test)
```

- Y_pred = model.predict(X_test): Uses the predict method of the decision tree model to generate predictions for the independent variables X_test. The predicted values are stored in the Y_pred variable.

6. Calculate the accuracy of the predicted value.

```
# Evaluate accuracy
accuracy = accuracy_score(Y_test, Y_pred)
print("Accuracy:", accuracy)
```

- accuracy = accuracy_score(Y_test, Y_pred): Uses the accuracy_score function from Scikit-learn's metrics module to calculate the accuracy of the model's predictions. The Y_test parameter represents the actual values of the dependent variable in the test set, and Y_pred represents the predicted values.
- print("Accuracy:", accuracy): Prints the calculated accuracy score to the console.

7. Create confusion matrix

```
# Create a confusion matrix
confusion_mat = confusion_matrix(Y_test, Y_pred)
print("Confusion Matrix:")
print(confusion_mat)
```

- confusion_mat = confusion_matrix(Y_test, Y_pred): Uses the confusion_matrix function from Scikit-learn's metrics module to compute the confusion matrix. The Y_test parameter represents the actual values of the dependent variable in the test set, and Y_pred represents the predicted values.
- print("Confusion Matrix:"): Prints a header indicating that the following output is the confusion matrix.
- print(confusion_mat): Prints the confusion matrix to the console.

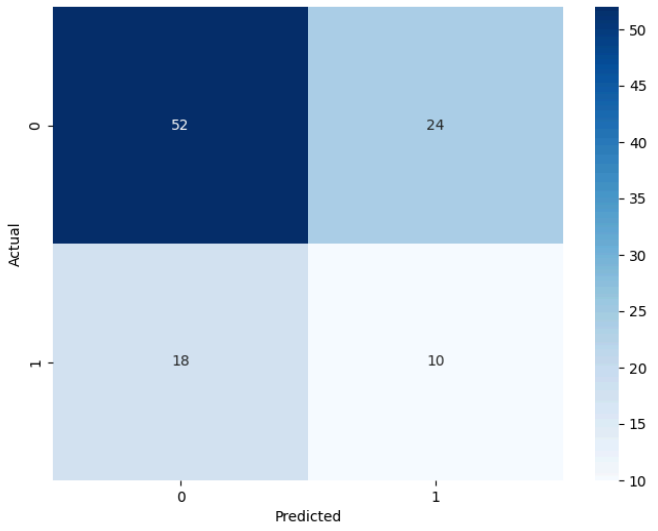
8. Print and visualize the outputs of confusion matrix and predicted value.

```
# Visualize the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_mat, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

- plt.figure(figsize=(8, 6)): Creates a new figure with a specific size for the plot.
- sns.heatmap(confusion_mat, annot=True, fmt='d', cmap='Blues'): Uses the heatmap function from the Seaborn library to create a heatmap plot of the confusion matrix. The confusion_mat parameter represents the confusion matrix computed earlier. The annot=True parameter displays the numerical values in each cell. The fmt='d' parameter formats the cell values as integers. The cmap='Blues' parameter sets the color scheme to shades of blue.
- plt.xlabel('Predicted'): Sets the label for the x-axis of the plot as "Predicted".
- plt.ylabel('Actual'): Sets the label for the y-axis of the plot as "Actual".
- plt.show(): Displays the plot.

Output

b) Confusion Matrix



b) Result

```
Accuracy: 0.5961538461538461
Confusion Matrix:
[[52 24]
 [18 10]]
```

Conclusion

Prediction Model	Accuracy	TP	FP	FN	FT
Logistic Regression	0.7307692307692307	0	0	28	76
Decision Tree	0.5961538461538461	10	24	18	52

Based on the result given, it can be seen that Logistic Regression model is more suitable for this dataset. This is because the accuracy for Logistic Regression is higher than Decision Tree (0.7307692307692307 > 0.5961538461538461). This might be due to the nature of the dependent variable (size_category) which is binary which makes it very suitable for Logistic regression that requires binary results.

CONCLUSION

In conclusion, Python proves to be an exceptional tool for performing data mining tasks such as classification, regression, clustering, and prediction. Its versatility and robustness enable efficient and effective analysis of large datasets. The simplicity of Python's syntax and its extensive ecosystem of libraries provide powerful capabilities for implementing sophisticated data mining algorithms.

For classification tasks, Python offers various algorithms such as decision trees, random forests, and support vector machines, allowing accurate and reliable prediction of categorical outcomes. In terms of regression, Python's libraries offer linear regression, polynomial regression, and other advanced techniques, enabling the modelling and prediction of continuous variables. Python's clustering algorithms, including k-means, hierarchical clustering, and DBSCAN, facilitate grouping similar data points together, uncovering patterns and structures within datasets. Lastly, Python's capabilities for prediction, encompassing time series forecasting, neural networks, and ensemble methods, enable accurate predictions based on historical data patterns.

The availability of extensive documentation, online resources, and a vibrant community contribute to Python's accessibility and continuous development in the field of data mining. Overall, Python's flexibility, ease of use, and comprehensive range of libraries make it an ideal choice for conducting diverse data mining tasks, empowering data scientists and analysts to derive valuable insights and make informed decisions.