VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

Aditya Dinesh Netrakar (1BM22CS017)

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
Sep-2024 to Jan-2025

B.M.S. College of Engineering,
Bull Temple Road, Bangalore 560019
(Affiliated To Visvesvaraya Technological University, Belgaum)
Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled "Machine Learning (23CS6PCMAL)" carried out by **Aditya Dinesh Netrakar (1BM22CS017),** who is bonafide student of **B.M.S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum. The Lab report has been approved as it satisfies the academic requirements in respect of an Machine Learning (23CS6PCMAL) work prescribed for the said degree.

Lakshmi Neelima Assistant Professor Department of CSE, BMSCE	Dr. Kavitha Sooda Professor & HOD Department of CSE, BMSCE

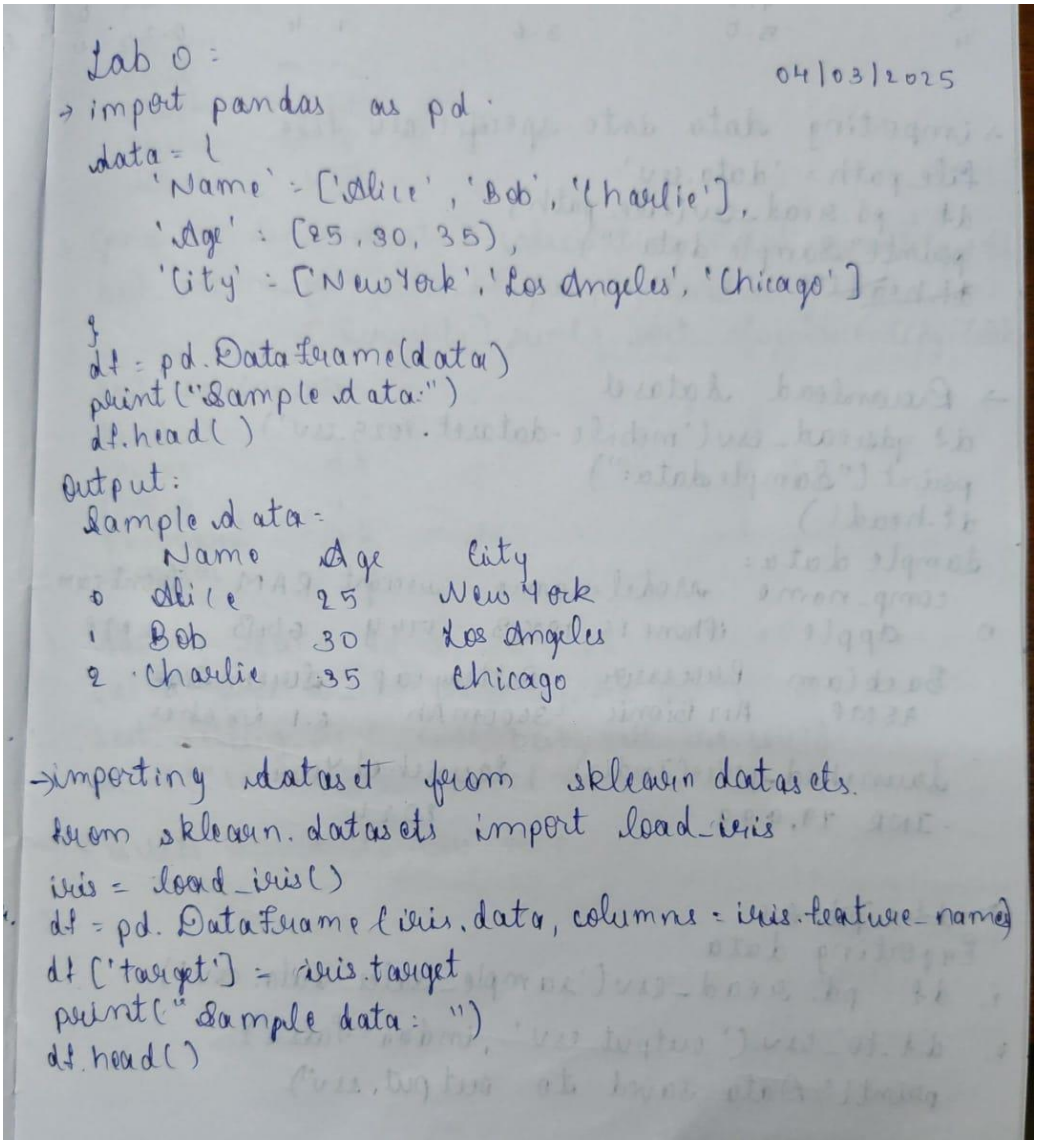
Index

Github Link: https://github.com/Aditya1BM22CS017/6A-ML-Lab-Batch2.git

Program 1

Write a python program to import and export data using Pandas library functions

Screenshot



Lab 0 :

04/03/2025

→ import pandas as pd.

```
data = {
    'Name' : ['Alice', 'Bob', 'Charlie'],
    'Age' : [25, 30, 35],
    'City' : ['New York', 'Los Angeles', 'Chicago']
}
df = pd.DataFrame(data)
print("Sample data:")
df.head()
```

Output:

Sample data:

	Name	Age	City
0	Alice	25	New York
1	Bob	30	Los Angeles
2	Charlie	35	Chicago

→ importing dataset from sklearn datasets.

```
from sklearn.datasets import load_iris
iris = load_iris()
df = pd.DataFrame(iris.data, columns = iris.feature_names)
df['target'] = iris.target
print("Sample data:")
df.head()
```

Output :
Sample data :

	sepal length(cm)	sepal width(cm)	petal len(cm)	petal width(cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

→ importing data into specific .csv file

```
file_path = 'data.csv'
df = pd.read_csv(file_path)
print("Sample data:")
df.head()
```

→ Download Dataset

```
df = pd.read_csv('mobile-dataset-2025.csv')
print("Sample data:")
df.head()
```

Sample data :

	comp_name	Model name	weight	RAM	front cam
0	Apple	iPhone 16	128GB	174 g	6GB	12MP

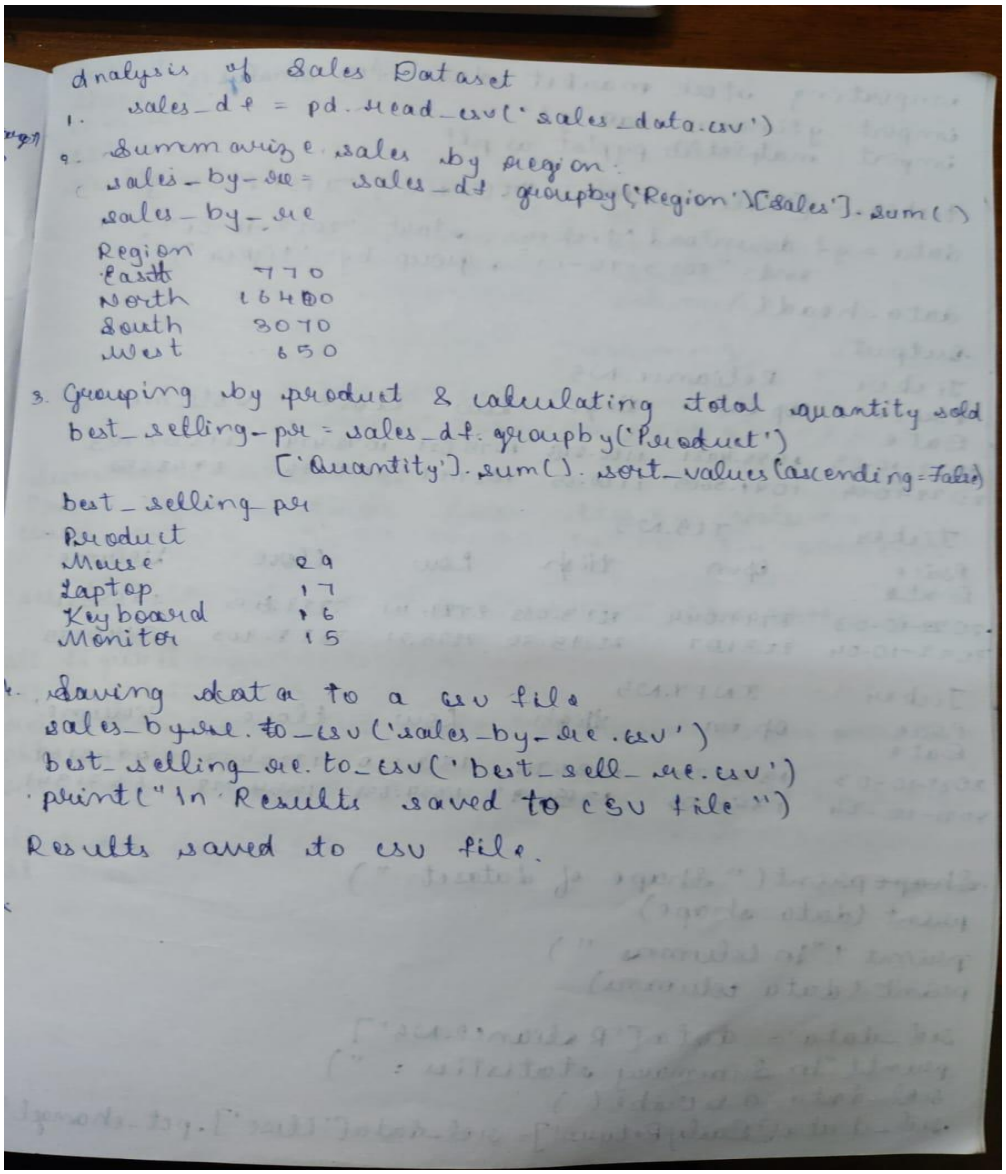
Back Cam	Processor	Battery cap	Screen Size
48MP	A17 Bionic	3600mAh	6.1 inches

launched-price(India)	Launched Year
INR 79,999	2024

→ ~~df = pd.re~~

Exporting data

1. df = pd.read_csv('sample_sales_data.csv')
2. df.to_csv('output.csv', index=False)
   print("Data saved to output.csv")



Analysis of Sales Dataset

1. sales_df = pd.read_csv('sales_data.csv')
2. Summarize sales by region:
   sales_by_re = sales_df.groupby('Region')['Sales'].sum()
   sales_by_re

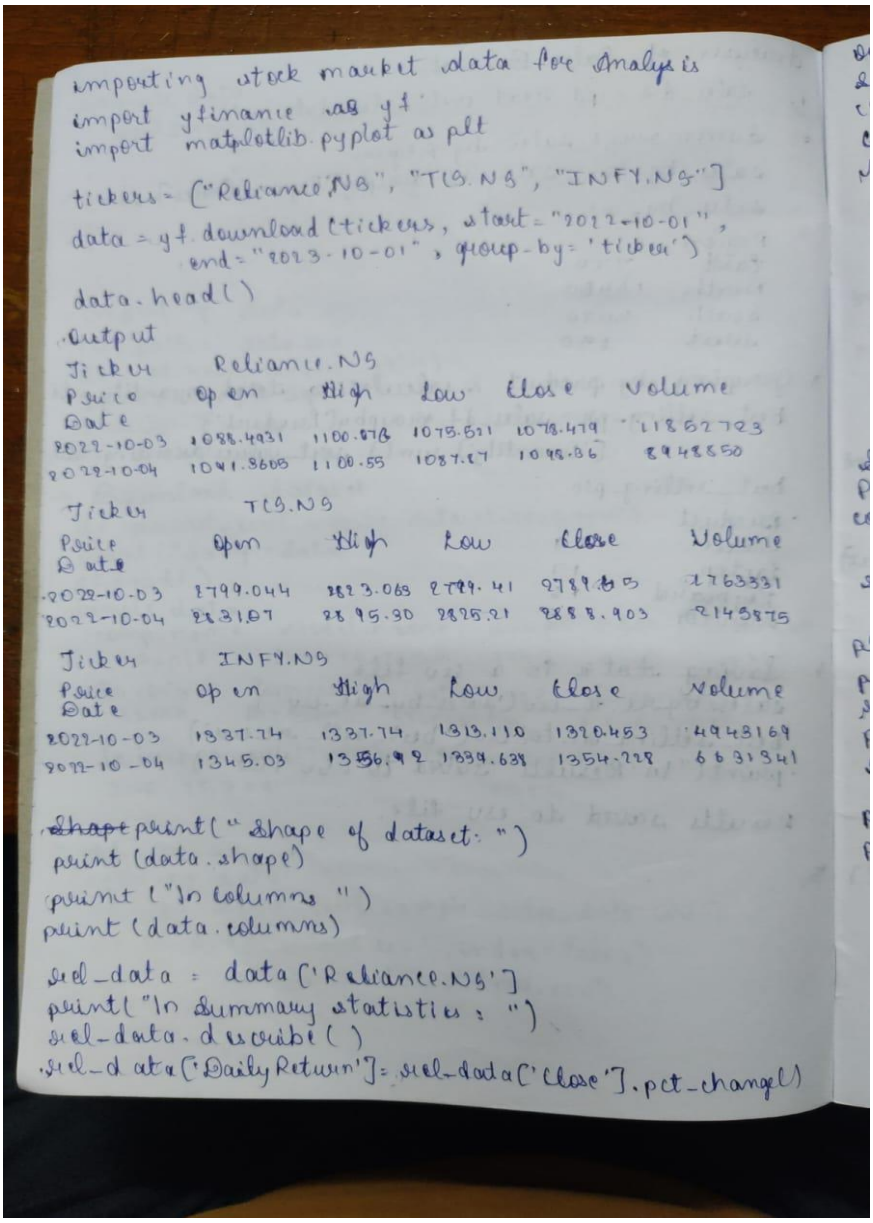
Region	
East	770
North	16400
South	3070
West	650

3. Grouping by product & calculating total quantity sold
   best_selling_pr = sales_df.groupby('Product')['Quantity'].sum().sort_values(ascending=False)
   best_selling_pr

Product	
Mouse	29
Laptop	17
Keyboard	16
Monitor	15

4. Saving data to a csv file
   sales_by_re.to_csv('sales_by_re.csv')
   best_selling_re.to_csv('best_sell_re.csv')
   print("\n Results saved to CSV file")

Results saved to csv file.



importing stock market data for Analysis

```
import yfinance as yf
import matplotlib.pyplot as plt

tickers = ["Reliance.NS", "TCS.NS", "INFY.NS"]
data = yf.download(tickers, start="2022-10-01",
                   end="2023-10-01", group_by='ticker')
data.head()
```

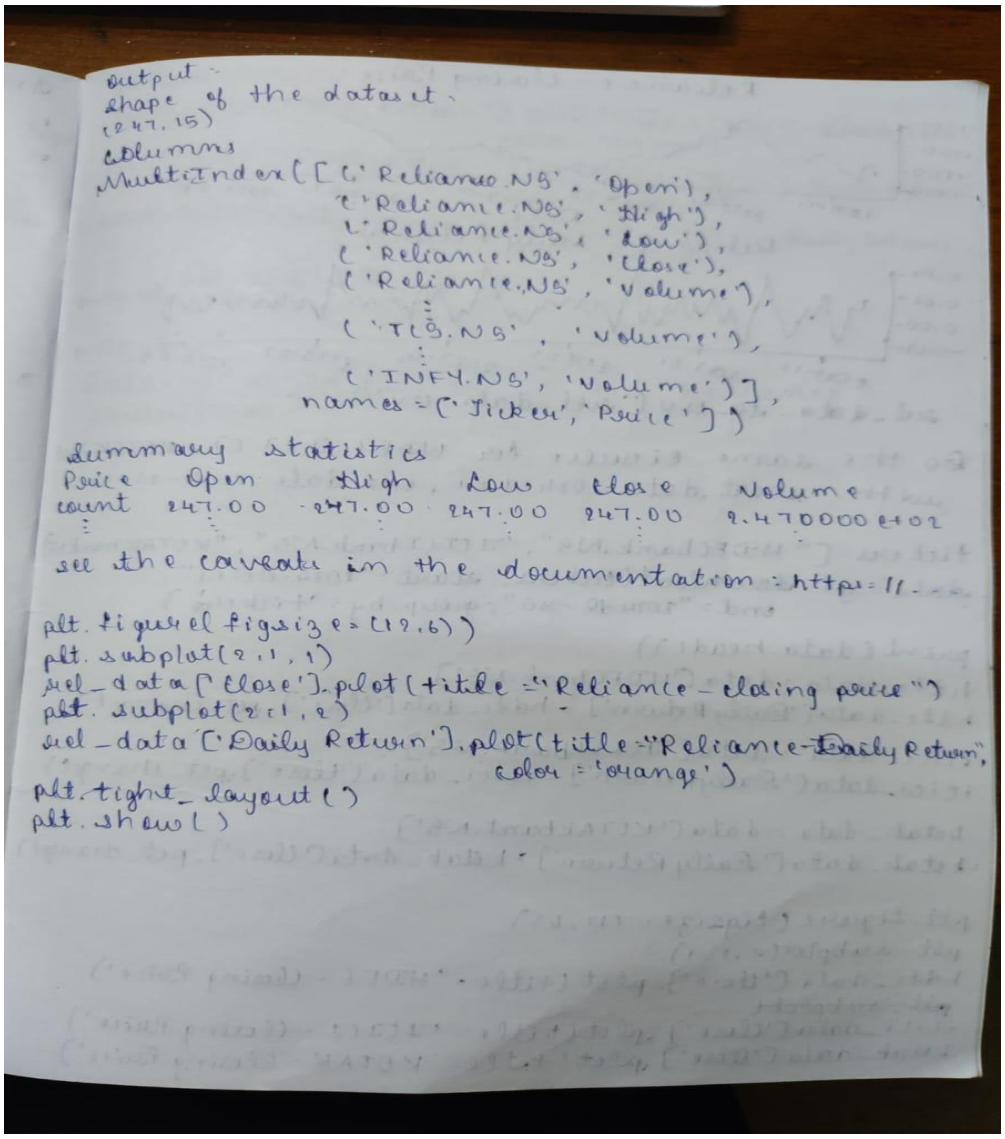
Output

Ticker	Reliance.NS				
Price	Open	High	Low	Close	Volume
Date					
2022-10-03	1088.4931	1100.676	1075.521	1078.479	11852723
2022-10-04	1091.8605	1100.55	1087.27	1098.36	8948850

Ticker	TCS.NS				
Price	Open	High	Low	Close	Volume
Date					
2022-10-03	2799.044	2823.069	2789.41	2789.85	1763331
2022-10-04	2831.07	2895.30	2875.21	2888.903	2143875

Ticker	INFY.NS				
Price	Open	High	Low	Close	Volume
Date					
2022-10-03	1337.74	1337.74	1313.110	1320.453	4943169
2022-10-04	1345.03	1356.92	1334.638	1354.228	6631341

```
print("Shape of dataset: ")
print(data.shape)
print("In Columns ")
print(data.columns)

sel_data = data['Reliance.NS']
print("In Summary statistics : ")
sel_data.describe()
sel_data['Daily Return'] = sel_data['Close'].pct_change()
```



Code:

import pandas as pd

data = {

'Name': ['Alice', 'Bob', 'Charlie', 'David'],

'Age': [25, 30, 35, 40],

```
'City': ['New York', 'Los Angeles', 'Chicago', 'Houston']

}

df = pd.DataFrame(data)

print("Sample data:")

print(df.head())
```

```
Sample data:
      Name  Age         City
0    Alice   25     New York
1      Bob   30  Los Angeles
2  Charlie   35      Chicago
3    David   40      Houston
```

In [ ]:

```
from sklearn.datasets import load_iris
iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target
print("Sample data:")
print(df.head())
```

Sample data:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	\
0	5.1	3.5	1.4	0.2	
1	4.9	3.0	1.4	0.2	
2	4.7	3.2	1.3	0.2	
3	4.6	3.1	1.5	0.2	
4	5.0	3.6	1.4	0.2	

	target
0	0
1	0
2	0
3	0
4	0

In [ ]:

```
from sklearn.datasets import load_iris
import pandas as pd
file_path= "data.csv"
df=pd.read_csv(file_path)
print("Sample data: ")
print(df.head())
print("\n")
```

Sample data:

```
   ID     Name  Age         City
0   1    Alice   25     New York
1   2      Bob   30  Los Angeles
2   3  Charlie   35      Chicago
3   4    David   40      Houston
4   5      Eva   28      Phoenix
```

In [ ]:

```
df = pd.read_csv("mobile.csv",encoding='latin-1')
print("Sample data:")
print(df.head())
```

Sample data:

```
  Company Name           Model Name Mobile Weight  RAM Front Camera  \
0        Apple       iPhone 16 128GB         174g  6GB        12MP
1        Apple       iPhone 16 256GB         174g  6GB        12MP
2        Apple       iPhone 16 512GB         174g  6GB        12MP
3        Apple  iPhone 16 Plus 128GB          203g  6GB        12MP
4        Apple  iPhone 16 Plus 256GB          203g  6GB        12MP
```

	Back Camera	Processor	Battery Capacity	Screen Size	\
0	48MP	A17 Bionic	3,600mAh	6.1 inches	
1	48MP	A17 Bionic	3,600mAh	6.1 inches	
2	48MP	A17 Bionic	3,600mAh	6.1 inches	
3	48MP	A17 Bionic	4,200mAh	6.7 inches	
4	48MP	A17 Bionic	4,200mAh	6.7 inches	

	Launched Price (Pakistan)	Launched Price (India)	Launched Price (China)	\
0	PKR 224,999	INR 79,999	CNY 5,799	
1	PKR 234,999	INR 84,999	CNY 6,099	
2	PKR 244,999	INR 89,999	CNY 6,499	
3	PKR 249,999	INR 89,999	CNY 6,199	
4	PKR 259,999	INR 94,999	CNY 6,499	

	Launched Price (USA)	Launched Price (Dubai)	Launched Year
0	USD 799	AED 2,799	2024
1	USD 849	AED 2,999	2024
2	USD 899	AED 3,199	2024
3	USD 899	AED 3,199	2024
4	USD 949	AED 3,399	2024

In [ ]:

 # Reading data from a CSV file

sales_df = pd.read_csv("sample_sales_data.csv")

In [ ]:

```
# Grouping by Region and calculating total sales
sales_by_region = sales_df.groupby('Region')['Sales'].sum()
print("\nTotal sales by region:")
print(sales_by_region)
```

```
Total sales by region:
Region
East       770
North    16400
South     3070
West       650
Name: Sales, dtype: int64
```

In [ ]:

```
# Grouping by Product and calculating total quantity sold
best_selling_products = sales_df.groupby('Product')['Quantity'].sum().sort_values(ascending=False)
print("\nBest-selling products by quantity:")
print(best_selling_products)
```

```
Best-selling products by quantity:
Product
```

Mouse 29

Laptop 17

Keyboard 16

Monitor 15

Name: Quantity, dtype: int64

In [ ]:

```
# Saving the sales by region data to a CSV file
sales_by_region.to_csv('sales_by_region.csv')
# Saving the best-selling products data to a CSV file
best_selling_products.to_csv('best_selling_products.csv')
print("\nAnalysis results saved to CSV files.")
```

Analysis results saved to CSV files.

In [ ]:

```
# Step 1: Import required libraries
import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt
tickers = ["RELIANCE.NS", "TCS.NS", "INFY.NS"]
data = yf.download(tickers, start="2022-10-01", end="2023-10-01", group_by='ticker')
print("First 5 rows of the dataset:")
print(data.head())
```

YF.download() has changed argument auto_adjust default to True

[**********100%**********] 3 of 3 completed

First 5 rows of the dataset:

Ticker RELIANCE.NS \

Price Open High Low Close Volume

Date

2022-10-03 1088.493134 1100.076669 1075.521371 1078.479858 11852723

2022-10-04 1091.360592 1100.554607 1087.878645 1098.369873 8948850

2022-10-06 1105.561415 1115.119493 1100.622978 1102.420776 13352162

2022-10-07 1099.029980 1112.343173 1099.029980 1107.086182 7714340

2022-10-10 1094.637648 1100.372591 1086.900123 1095.001831 6329527

Ticker TCS.NS \

Price Open High Low Close Volume

Date

2022-10-03 2799.044599 2823.063066 2779.418577 2789.652100 1763331

2022-10-04 2831.707297 2895.304988 2825.212065 2888.903076 2145875

2022-10-06 2907.454752 2919.604193 2890.118388 2898.996826 1790816

2022-10-07 2894.744206 2901.847047 2858.015696 2864.370605 1939879

2022-10-10 2813.062865 2922.407833 2808.390003 2914.510742 3064063

Ticker INFY.NS

Price Open High Low Close Volume

Date

2022-10-03 1337.743240 1337.743240 1313.110574 1320.453003 4943169

2022-10-04 1345.038201 1356.928245 1339.638009 1354.228149 6631341

2022-10-06 1369.007786 1383.029504 1368.155094 1378.624023 6180672

2022-10-07 1370.286676 1381.181893 1364.412779 1374.881592 3994466

2022-10-10 1351.338576 1387.956005 1351.338576 1385.729614 5274677

In [ ]:

```
# Basic Data Exploration
print("\nShape of the dataset:")
print(data.shape)

print("\nColumn names:")
print(data.columns)



reliance_data = data['RELIANCE.NS']
print("\nSummary statistics for Reliance Industries:")
print(reliance_data.describe())

# daily returns

reliance_data['Daily Return'] = reliance_data['Close'].pct_change()
```

Shape of the dataset:

(247, 15)

Column names:

```
MultiIndex([('RELIANCE.NS',   'Open'),
            ('RELIANCE.NS',   'High'),
            ('RELIANCE.NS',    'Low'),
            ('RELIANCE.NS',  'Close'),
            ('RELIANCE.NS', 'Volume'),
            (    'TCS.NS',   'Open'),
            (    'TCS.NS',   'High'),
            (    'TCS.NS',    'Low'),
            (    'TCS.NS',  'Close'),
            (    'TCS.NS', 'Volume'),
            (   'INFY.NS',   'Open'),
            (   'INFY.NS',   'High'),
            (   'INFY.NS',    'Low'),
            (   'INFY.NS',  'Close'),
            (   'INFY.NS', 'Volume')],
           names=['Ticker', 'Price'])
```

Summary statistics for Reliance Industries:

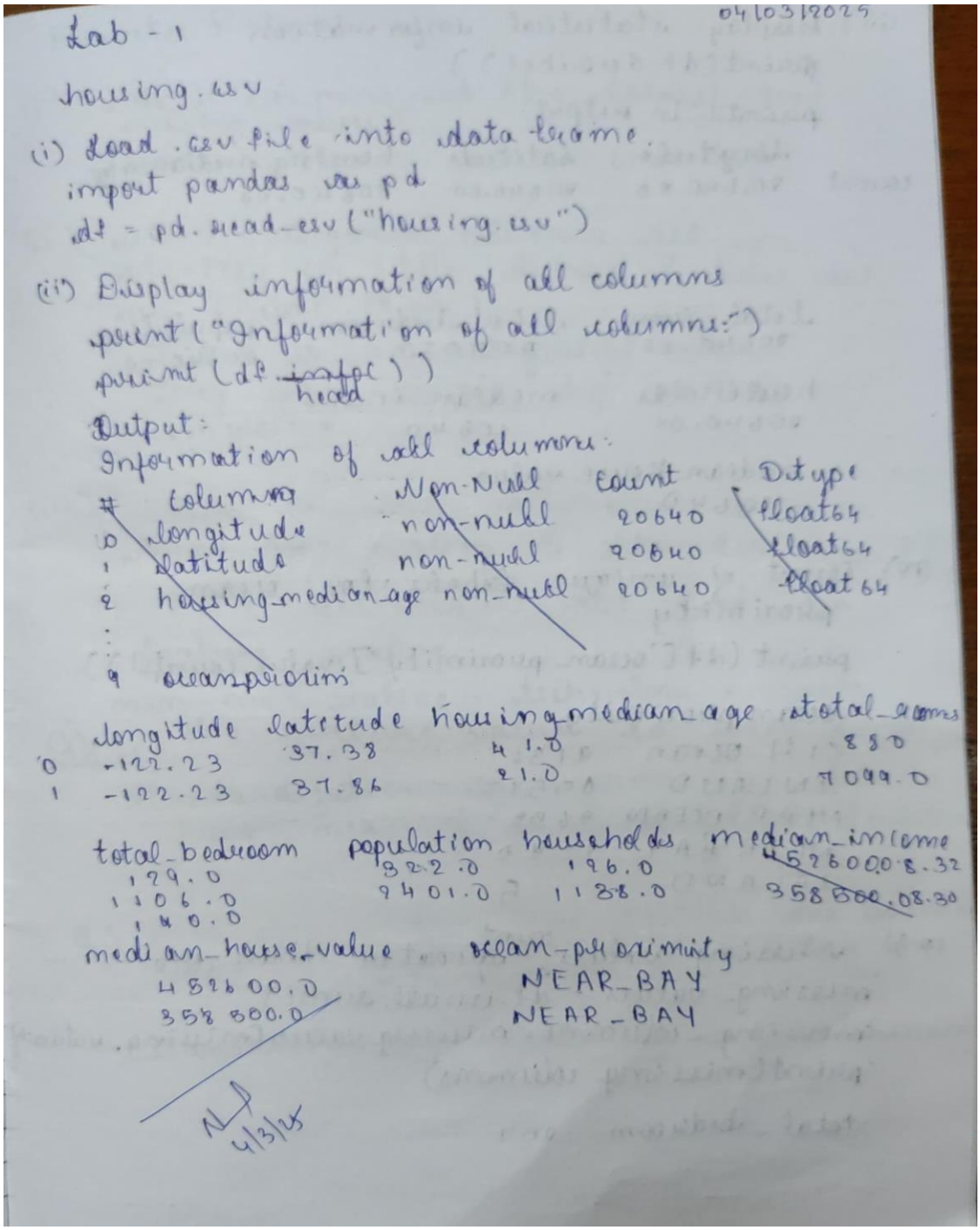
Price	Open	High	Low	Close	Volume
count	247.000000	247.000000	247.000000	247.000000	2.470000e+02

mean	1147.548566	1156.216094	1137.195378	1146.522438	1.316652e+07
std	65.890225	66.850133	65.752470	66.687258	6.754099e+06
min	1008.159038	1010.434788	992.228728	1001.900696	3.370033e+06
25%	1098.882005	1103.399363	1084.795044	1097.357117	8.717141e+06
50%	1147.435205	1155.036230	1138.787343	1147.253174	1.158959e+07
75%	1196.261444	1203.625226	1184.985050	1195.064575	1.530302e+07
max	1288.076701	1299.910648	1273.056728	1293.470215	5.708188e+07

PROGRAM 2

Demonstrate various data pre-processing techniques for a given dataset

Screenshot



(iii) Display statistical information

```
print(df.describe())
print("In output:
```

	longitude	latitude	housing_median_age
count	20,640.00	20640.00	20640.00

total_room	total_bedroom	population
20640.00	20433.0	20640.00

households	median_income
20640.00	20640

median_house_value
20640

(iv) Count of unique labels for ocean proximity

```
print(df['ocean_proximity'].value_counts())
```

ocean proximity	
<1H OCEAN	9136
INLAND	6551
NEAR OCEAN	2658
NEAR BAY	2290
ISLAND	5

v) Missing values count greater than zero

```
missing_values = df.isnull.sum()
missing_columns = missing_values[missing_values>0]
print(missing_columns)
```

total_bedroom 207

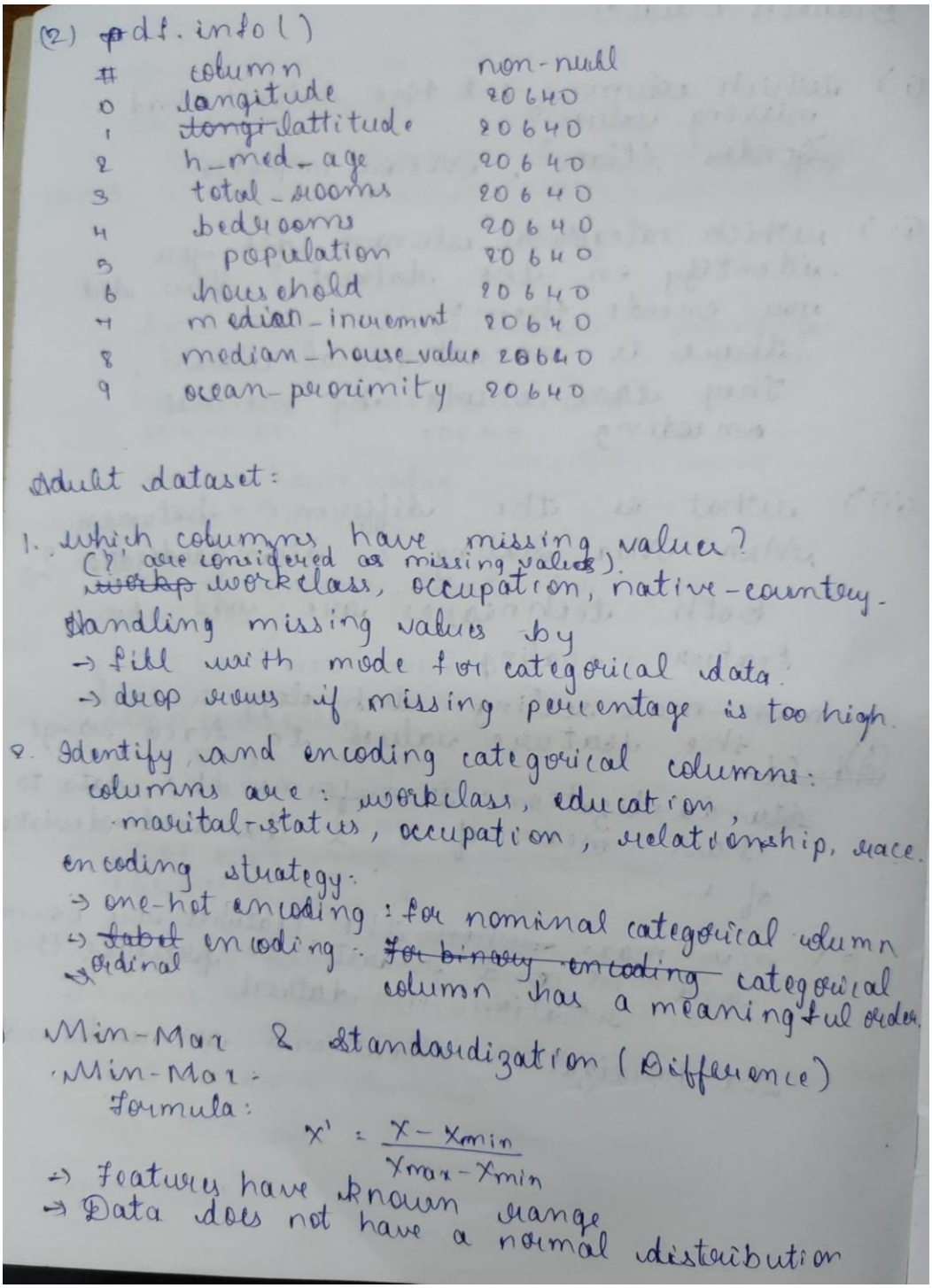
Diabetes Dataset

(i) which columns in the dataset had missing values?
'Gender', 'Class', Mean imputation.

(ii) which categorical columns did you identify in the dataset? How did you encode them?
There is no categorical column.
They can encode by one-hot encoding

(iii) What is the difference between Min-Max Scaling & standardization?
Both techniques are used for feature scaling
min-max scaling - technology rescales the feature values to fixed range
~~Adult~~
Standardization: transforms the data to have mean of 0 & standard deviation of 1.

(2) min-max: ~~when~~ the features are bounded and it ~~and~~ wants to preserve the relation in dataset.
standardization. features are unbounded.



(2) df.info()

#	column	non-null
0	longitude	20640
1	~~longi~~latitude	20640
2	h_med_age	20640
3	total_rooms	20640
4	bedrooms	20640
5	population	20640
6	household	20640
7	median_increment	20640
8	median_house_value	20640
9	ocean_proximity	20640

Adult dataset:

1. which columns have missing values?
('?' are considered as missing values).
~~workp~~ workclass, occupation, native-country.
Handling missing values by
→ fill with mode for categorical data.
→ drop rows if missing percentage is too high.

2. Identify and encoding categorical columns:
columns are: workclass, education, marital-status, occupation, relationship, race.
encoding strategy:
→ one-hot encoding: for nominal categorical column
→ ~~label~~ Ordinal encoding: ~~For binary encoding~~ categorical column has a meaningful order.

Min-Max & Standardization (Difference)
Min-Max:
formula:

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}}$$

→ features have known range
→ Data does not have a normal distribution

Code

 # Housing database

import pandas as pd

```
df = pd.read_csv("housing.csv")
print("Information of all columns:")
print(df.info())
```

```
Information of all columns:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object 
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
None
```

In [9]:

```
print("\nStatistical information of numerical columns:")
print(df.describe())

# count of unique labels for the Ocean Proximity column
print("\nCount of unique labels in 'Ocean Proximity' column:")
print(df['ocean_proximity'].value_counts())

# Display attributes with missing values count greater than zero
print("\nAttributes with missing values:")
missing_values = df.isnull().sum()
missing_columns = missing_values[missing_values > 0]
print(missing_columns)
```

```
Statistical information of numerical columns:
        longitude      latitude  housing_median_age   total_rooms  \
count  20640.000000  20640.000000       20640.000000  20640.000000
mean    -119.569704     35.631861          28.639486   2635.763081
std        2.003532      2.135952          12.585558   2181.615252
min     -124.350000     32.540000           1.000000      2.000000
25%     -121.800000     33.930000          18.000000   1447.750000
50%     -118.490000     34.260000          29.000000   2127.000000
75%     -118.010000     37.710000          37.000000   3148.000000
```

```
max     -114.310000     41.950000          52.000000  39320.000000


       total_bedrooms    population    households  median_income  \
count    20433.000000  20640.000000  20640.000000   20640.000000
mean       537.870553   1425.476744    499.539680       3.870671
std        421.385070   1132.462122    382.329753       1.899822
min          1.000000      3.000000      1.000000       0.499900
25%        296.000000    787.000000    280.000000       2.563400
50%        435.000000   1166.000000    409.000000       3.534800
75%        647.000000   1725.000000    605.000000       4.743250
max       6445.000000  35682.000000   6082.000000      15.000100


       median_house_value
count        20640.000000
mean        206855.816909
std         115395.615874
min          14999.000000
25%         119600.000000
50%         179700.000000
75%         264725.000000
max         500001.000000


Count of unique labels in 'Ocean Proximity' column:

ocean_proximity
```

<1H OCEAN 9136

INLAND 6551

NEAR OCEAN 2658

NEAR BAY 2290

ISLAND 5

Name: count, dtype: int64

Attributes with missing values:

total_bedrooms 207

dtype: int64

In [6]:

 # Adult database

```
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler, StandardScaler, LabelEncoder


file_path = "adult.csv"
df = pd.read_csv(file_path)
```

 # Handling Missing Values

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


num_imputer = SimpleImputer(strategy="mean")

df[df.select_dtypes(include=['number']).columns] =
num_imputer.fit_transform(df.select_dtypes(include=['number']))


cat_imputer = SimpleImputer(strategy="most_frequent")

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



# Step 4: Handling Categorical Data- encoding

label_encoders = {}

for col in df.select_dtypes(include=['object']).columns:

    le = LabelEncoder()

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

    label_encoders[col] = le


# Handling Outliers

Q1 = df.quantile(0.25)

Q3 = df.quantile(0.75)

IQR = Q3 - Q1

df = df[~((df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR))).any(axis=1)]


# Data Transformation
```

```
# Normalization
min_max_scaler = MinMaxScaler()
df_minmax = pd.DataFrame(min_max_scaler.fit_transform(df), columns=df.columns)

# Standardization: z-score
standard_scaler = StandardScaler()
df_standard = pd.DataFrame(standard_scaler.fit_transform(df), columns=df.columns)

# Display Processed Data
print("\nProcessed Adult Income Dataset (Min-Max Scaled):")
print(df_minmax.head())

print("\nProcessed Adult Income Dataset (Standard Scaled):")
print(df_standard.head())
```

```
Processed Adult Income Dataset (Min-Max Scaled):
        age  workclass    fnlwgt  education  educational-num  marital-status  \
0  0.344262        0.0  0.188277   0.555556         0.363636        0.333333
1  0.114754        0.0  0.881156   1.000000         0.454545        0.666667
2  0.147541        0.0  0.169156   0.555556         0.363636        0.666667
3  0.672131        0.0  0.708251   0.555556         0.363636        0.333333
4  0.131148        0.0  0.475807   0.333333         0.727273        0.333333
```

	occupation	relationship	race	gender	capital-gain	capital-loss	\
0	0.307692	0.0	0.0	1.0	0.0	0.0	
1	0.538462	0.8	0.0	0.0	0.0	0.0	
2	0.000000	0.2	0.0	0.0	0.0	0.0	
3	0.692308	0.0	0.0	1.0	0.0	0.0	
4	0.692308	0.0	0.0	1.0	0.0	0.0	

	hours-per-week	native-country	income
0	0.894737	0.0	0.0
1	0.368421	0.0	0.0
2	0.315789	0.0	0.0
3	0.105263	0.0	0.0
4	0.368421	0.0	0.0

Processed Adult Income Dataset (Standard Scaled):

	age	workclass	fnlwgt	education	educational-num	marital-status	\
0	0.220179	0.0	-1.022983	-0.151256	-0.654083	-0.398228	
1	-0.955630	0.0	2.234629	1.457372	-0.073261	0.828047	
2	-0.787657	0.0	-1.112882	-0.151256	-0.654083	0.828047	
3	1.899906	0.0	1.421707	-0.151256	-0.654083	-0.398228	
4	-0.871644	0.0	0.328856	-0.955571	1.669207	-0.398228	

	occupation	relationship	race	gender	capital-gain	capital-loss	\
0	-0.420679	-1.044582	0.0	0.770972	0.0	0.0	

1	0.305840	1.629927	0.0	-1.297064	0.0	0.0
2	-1.389371	-0.375955	0.0	-1.297064	0.0	0.0
3	0.790186	-1.044582	0.0	0.770972	0.0	0.0
4	0.790186	-1.044582	0.0	0.770972	0.0	0.0

	hours-per-week	native-country	income
0	2.312838	0.0	0.0
1	-0.329781	0.0	0.0
2	-0.594043	0.0	0.0
3	-1.651090	0.0	0.0
4	-0.329781	0.0	0.0

In [5]:

```
# Diabetes databse

from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler, StandardScaler

# 2: Load the Diabetes dataset
file_path = "Dataset of Diabetes .csv"
df = pd.read_csv(file_path)

# 3: Handling Missing Values (Replacing NaNs with mean)
```

```
df_numeric = df.select_dtypes(include=['number']).copy()
# 'mean' strategy
imputer = SimpleImputer(strategy="mean")
# Apply the imputer to fill missing values
df_numeric.iloc[:, :] = imputer.fit_transform(df_numeric)
# Merge
df[df_numeric.columns] = df_numeric



# Step 4: Handling Outliers
# Calculate quartiles and IQR for numeric columns
Q1 = df_numeric.quantile(0.25)
Q3 = df_numeric.quantile(0.75)
IQR = Q3 - Q1

# filter
df = df[~((df_numeric < (Q1 - 1.5 * IQR)) | (df_numeric > (Q3 + 1.5 * IQR))).any(axis=1)]

# Step 5: Data Transformation
# Min-Max Normalization
min_max_scaler = MinMaxScaler()
df_minmax = pd.DataFrame(min_max_scaler.fit_transform(df_numeric), columns=df_numeric.columns)

# Standardization (Z-score normalization)
```

```
standard_scaler = StandardScaler()
df_standard = pd.DataFrame(standard_scaler.fit_transform(df_numeric), columns=df_numeric.columns)

# Step 6: Display Processed Data
print("\nProcessed Diabetes Dataset (Min-Max Scaled):")
print(df_minmax.head())
print("\nProcessed Diabetes Dataset (Standard Scaled):")
print(df_standard.head())
```

```
Processed Diabetes Dataset (Min-Max Scaled):
         ID  No_Pation       AGE      Urea        Cr     HbA1c      Chol  \
0  0.627034   0.000237  0.508475  0.109375  0.050378  0.264901  0.407767
1  0.918648   0.000452  0.101695  0.104167  0.070529  0.264901  0.359223
2  0.524406   0.000634  0.508475  0.109375  0.050378  0.264901  0.407767
3  0.849812   0.001160  0.508475  0.109375  0.050378  0.264901  0.407767
4  0.629537   0.000452  0.220339  0.171875  0.050378  0.264901  0.475728

         TG       HDL       LDL      VLDL       BMI
0  0.044444  0.226804  0.114583  0.011461  0.173913
1  0.081481  0.092784  0.187500  0.014327  0.139130
2  0.044444  0.226804  0.114583  0.011461  0.173913
3  0.044444  0.226804  0.114583  0.011461  0.173913
4  0.051852  0.061856  0.177083  0.008596  0.069565
```

Processed Diabetes Dataset (Standard Scaled):

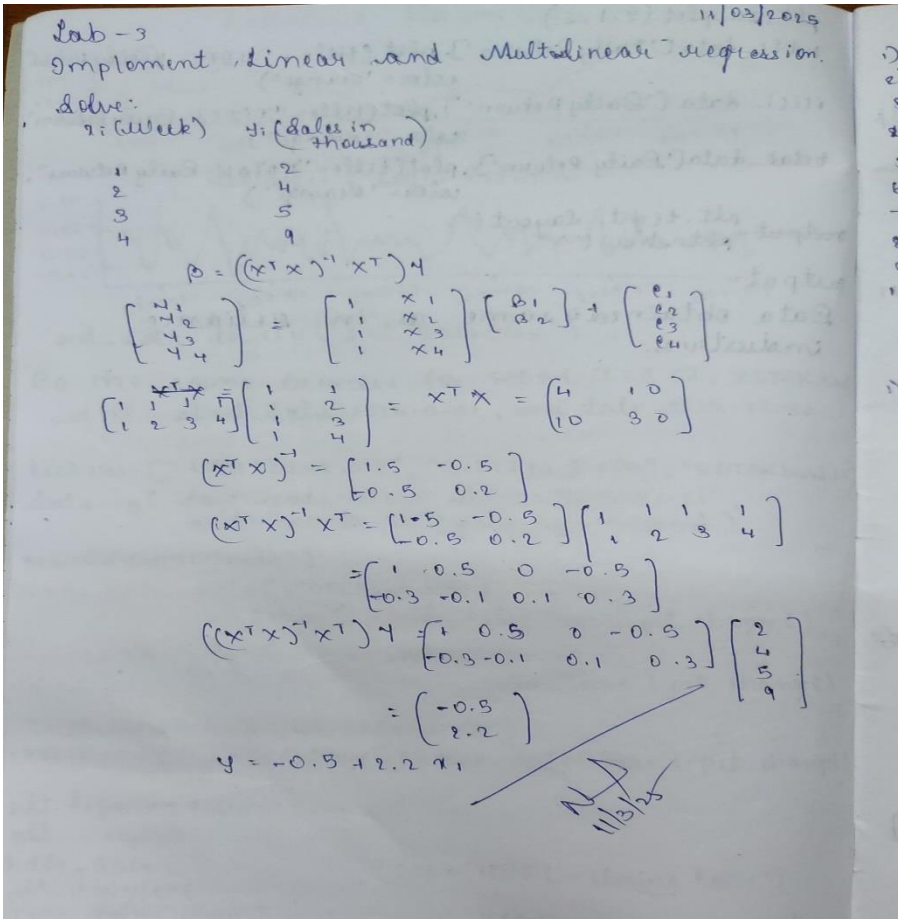
	ID	No_Pation	AGE	Urea	Cr	HbA1c	Chol	\
0	0.672140	-0.074747	-0.401144	-0.144781	-0.382672	-1.334983	-0.509436	
1	1.641852	-0.069940	-3.130017	-0.212954	-0.115804	-1.334983	-0.893730	
2	0.330868	-0.065869	-0.401144	-0.144781	-0.382672	-1.334983	-0.509436	
3	1.412950	-0.054126	-0.401144	-0.144781	-0.382672	-1.334983	-0.509436	
4	0.680463	-0.069939	-2.334096	0.673299	-0.382672	-1.334983	0.028576	

	TG	HDL	LDL	VLDL	BMI
0	-1.035084	1.810756	-1.085457	-0.369958	-1.124622
1	-0.678063	-0.158692	-0.457398	-0.342649	-1.326239
2	-1.035084	1.810756	-1.085457	-0.369958	-1.124622
3	-1.035084	1.810756	-1.085457	-0.369958	-1.124622
4	-0.963680	-0.613180	-0.547121	-0.397267	-1.729472

PROGRAM 3

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Screenshot



Decision Tree:

Instance	a_1	a_2	a_3	Classification
1	T	Heat	Hi	No
2	T	Heat	Hi	No
3	F	Heat	Hi	Y
4	F	Cool	Nor	Y
5	F	Cool	Nor	Y
6	T	Cool	Hi	No
7	T	Heat	Hi	No
8	T	Heat	Nor	Y
9	F	Cool	Nor	Y
10	F	Cool	Hi	Y

$S = [4+, 2-] = -\frac{4}{6}\log_2\frac{4}{6} - \frac{2}{6}\log_2\frac{2}{6}$

$= 0.9709$

attribute a_1 values $(a_1) = (T, F)$

$S_T = [4+, 1-] = -\frac{1}{5}\log_2\frac{1}{5} - \frac{4}{5}\log_2\frac{4}{5}$

$= 0.7219$

$S_F = [5+, 0-] = 0$

$\text{gain}(S, a_1) = \text{entropy}(S) - \frac{5}{10}(0.7219) - \frac{5}{10}(0)$

$= 0.9709 - 0.36095$

$= 0.60994$

attribute : a_2

$S_{Heat} = [2+, 3-] = 0.97095$

$S_{Cool} = [4+, 1-] = 0.72192$

$\text{Gain}(S, a_2) = 0.9709 - \frac{5}{10} \times 0.97095 - \frac{5}{10}(0.72192)$

$= 0.12464$

Decision Tree:

Instance	a_1	a_2	a_3	Classification
1	T	Heat	Hi	No
2	T	Heat	Hi	No
3	F	Heat	Hi	Y
4	F	Cool	Nor	N
5	F	Cool	Nor	N
6	T	Cool	Hi	No
7	T	Heat	Hi	No
8	T	Heat	Nor	Y
9	F	Cool	Nor	Y
10	F	Cool	Hi	N

$S = [4+, 2-] = -\frac{4}{6}\log_2\frac{4}{6} - \frac{2}{6}\log_2\frac{2}{6}$

$= 0.9709$

attribute a_1, values $(a_1) = (T, F)$

$S_T = [4+, 1-] = -\frac{1}{5}\log_2\frac{1}{5} - \frac{4}{5}\log_2\frac{4}{5}$

$= 0.7219$

$S_F = [5+, 0-] = 0$

$gain(S, a_1) = entropy(S) - \frac{5}{10}0.7219 - \frac{5}{10}(0)$

$= 0.9709 - 0.36095$

$= 0.6099$

attribute : a_2

$S_{Heat} = [2+, 3-] = 0.97095$

$S_{Cool} = [4+, 1-] = 0.72192$

$Gain(S, a_2) = 0.9709 - \frac{5}{10} \times 0.97095 - \frac{5}{10}0.72192$

$= 0.12449$

a_3 :

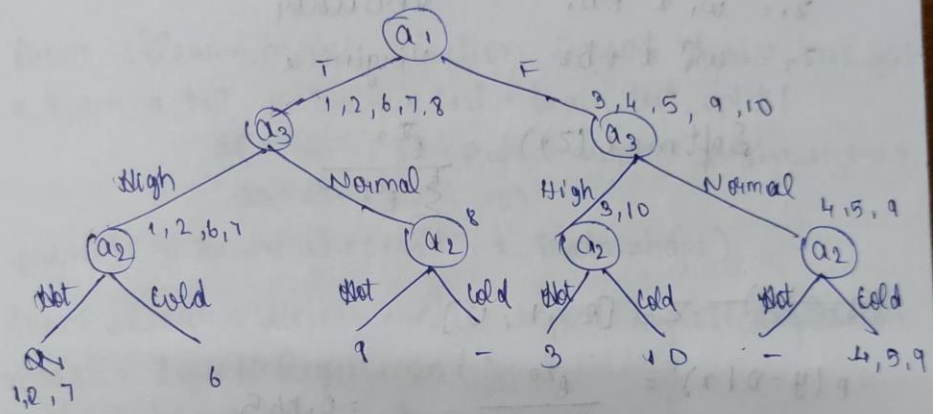
$S_{Hi} = [2+, 4-]$ Entropy$(S_{Hi}) = 0.9182$

$S_{Nor} = [4+, 0-]$ Entropy$(S_{Nor}) = 0$

Gain(S, a_3) $= 0.9709 - \frac{6}{10} \times 0.9182 - \frac{4}{10} \times 0$

$= 0.41998$

Max gain $= 0.6099 \rightarrow a_1$



Lab - 3

Binary logistic Regression

$a_0 = -5$ $a_1 = 0.8$ $x = 7$

$$y = \frac{1}{1 + e^{-(a_0 + a_1 x)}} = \frac{1}{1 + e^{-(-5 + 0.8(7))}}$$

$= 0.6457 > 0.5$

Pass

NLP

18/3/25

Output:

Coefficient: 828.4650

Intercept: -1632210.7578

Predicted per capita income in 2020: 41288.69 US$

2. Predict Salary of the employee with 12 years of experience

salary.csv

~~Predict~~ Output:

Predict salary for 12 years of experience: 13800561

Coefficient: 9275.46

Intercept: 26700.05

Mean squared error: 307288663.72

Multiple Regression:

```
from sklearn.metrics import mean_absolute_error,
                             r2_score

data = pd.read_csv("hiring.csv")
def conv_exp(val):
    word_to_num = {"zero":0, "one":1, "two":2,
                   "three":3, "four":4, "five":5, "six":6,
                   "seven":7, "eight":8, "nine":9, "ten":10,
                   "eleven":11, "twelve":12}
    return word_to_num
data['exp'] = data['exp'].apply(conv_exp)
data = data.dropna()
data = data.astype(float)

x = data[['exp','test_score','interview_score']]
y = data['salary($)']

model = LinearRegression()
model.fit(x,y)
```

Steps / Algorithm

1) import libraries
2) import data - distribution
3) analyze data - distribution
4) Distribute that - visualization
5) Relationship between variables
6) Split the data
7) Train the model
8) Predict the result
9) Visualize prediction
10) Check values of coefficient & intercept.

Linear Regression

1) Canada per capita

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
data = pd.read_csv("canada_per_capita_income.csv")

x = data[['year']]
y = data['per capita income (US$)']
x_train, x_test, y_train, y_test = train_test_split
                    (x, y, test_size = 0.2, random_state
                                                = 42)

model = LinearRegression()
model.fit(x, y)
y_pred = model.predict(x_test)
print("Coefficient : {model.coef_[0]}")
print("Intercept : {model.intercept_}")
y_2020 = model.predict([[2020]])
print("Predicted per capita income in 2020:
        {y_2020[0]:.2f} US$")
```

```
y_pred = model.predict(x)
print(f"Coefficients: {model.coef_}")
print(f"Intercept: {model.intercept_}")
candidates = np.array([[9.0, 6], [12, 10]])
salary_pr = model.predict(candidates)
print(f"Predicted salary for 9yrs: {salary_prediction
                                    [0]:.2f} US$")
print(f"Predicted salary for 12 yrs:
                {salary_prediction[1]:.2f} US$")
mae = mean_absolute_error(y_test, y_pred)
print(mae)
```

Output :

Coefficient = [2687.5 22125 1750]

Intercept = 21562.5

Predicted salary for 12 yrs : 92562.50 $

Mean absolute error : 687.5

11/3/25

2) Coefficients : [3.33043 1.13893 8.30755
-8.7449 -9.7133]

Intercept : -12439.155

Predict profit : 54066.30 US$

Mean Absolute error : 1404

(R2) Score : 0.98.

Code

```
import pandas as pd
import numpy as np
from sklearn import linear_model
import matplotlib.pyplot as plt

df = pd.read_csv('housing_area_price.csv')
print(df.head())

# Commented out IPython magic to ensure Python compatibility.
# %matplotlib inline
plt.xlabel('area')
plt.ylabel('price')
plt.scatter(df.area,df.price,color='red',marker='+')

new_df = df.drop('price',axis='columns')
new_df

price = df.price
price

# Create linear regression object
reg = linear_model.LinearRegression()
reg.fit(new_df,price)
```

"""(1) Predict price of a home with area = 3300 sqr ft"""

```
reg.predict([[3300]])
```

```
print(reg.coef_)
```

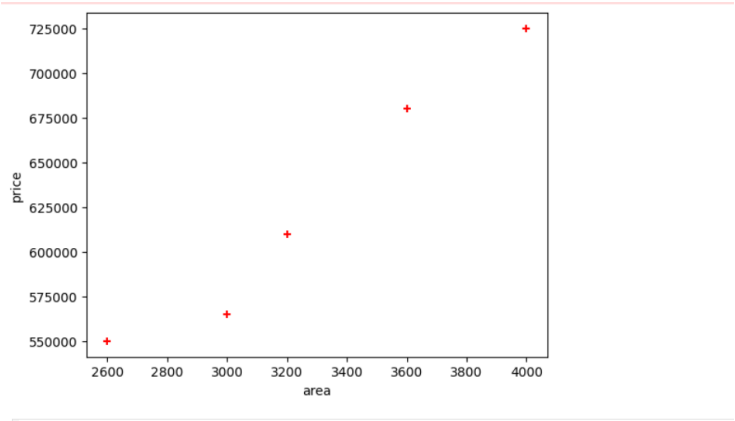
```
print(reg.intercept_)
```

"""Y = m * X + b (m is coefficient and b is intercept)"""

```
3300*135.78767123 + 180616.43835616432
```

"""(1) Predict price of a home with area = 5000 sqr ft"""

```
print(reg.predict([[5000]]))
```



```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# Load the dataset
data = pd.read_csv("canada_per_capita_income.csv")

# Analyze data distribution
print(data.describe())
print(data.info())

# Distribution plot visualization
plt.scatter(data['year'], data['per capita income (US$)'], color='blue', label='Actual Data')
plt.xlabel("Year")
plt.ylabel("Per Capita Income (US$)")
plt.title("Year vs Per Capita Income in Canada")

plt.show()

# Relationship between variables
X = data[['year']]
y = data['per capita income (US$)']
```

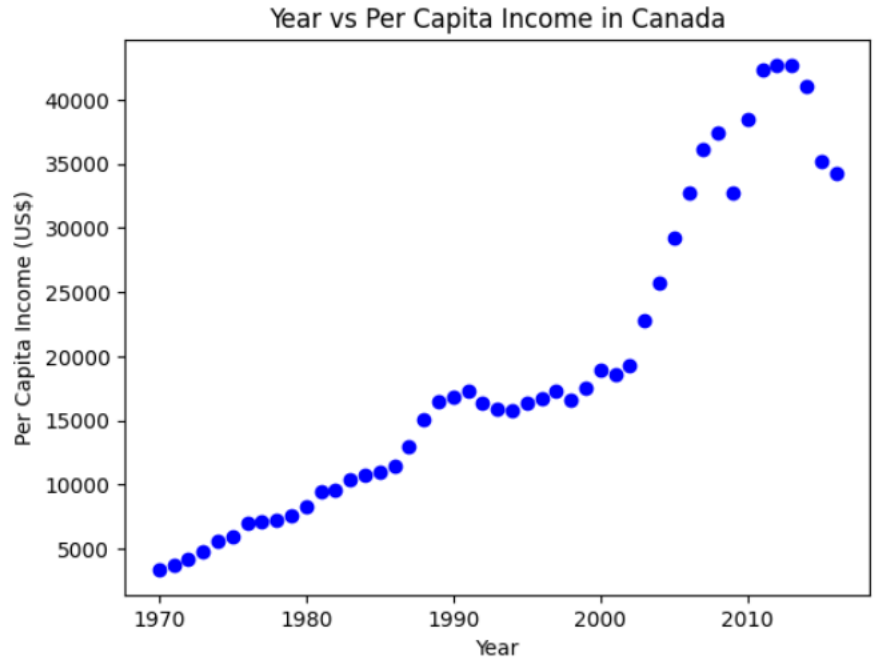
```
# Split the data (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train the model
model = LinearRegression()
model.fit(X, y)

# Predict the results
y_pred = model.predict(X_test)

# Visualize prediction
plt.scatter(X_test, y_test, color='blue', label='Actual')
plt.plot(X_test, y_pred, color='red', label='Predicted')
plt.xlabel("Year")
plt.ylabel("Per Capita Income (US$)")
plt.title("Prediction of Per Capita Income")

plt.show()

# Check values of coefficient and intercept
print(f"Coefficient: {model.coef_[0]}")
print(f"Intercept: {model.intercept_}")
```

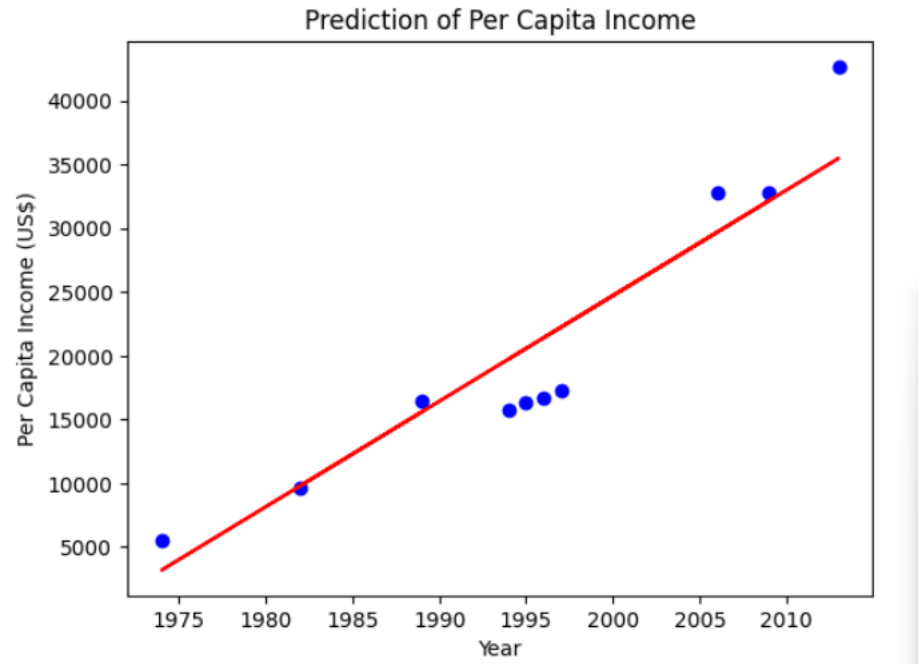
 # Predict per capita income for 2020

y_2020 = model.predict([[2020]])

print(f"Predicted per capita income in 2020: {y_2020[0]:.2f} US$")

```
             year  per capita income (US$)
count    47.000000                47.000000
mean   1993.000000             18920.137063
std      13.711309             12034.679438
min    1970.000000              3399.299037
25%    1981.500000              9526.914515
50%    1993.000000             16426.725480
75%    2004.500000             27458.601420
max    2016.000000             42676.468370
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 47 entries, 0 to 46
Data columns (total 2 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   year                    47 non-null     int64
 1   per capita income (US$)  47 non-null     float64
dtypes: float64(1), int64(1)
memory usage: 884.0 bytes
None
```





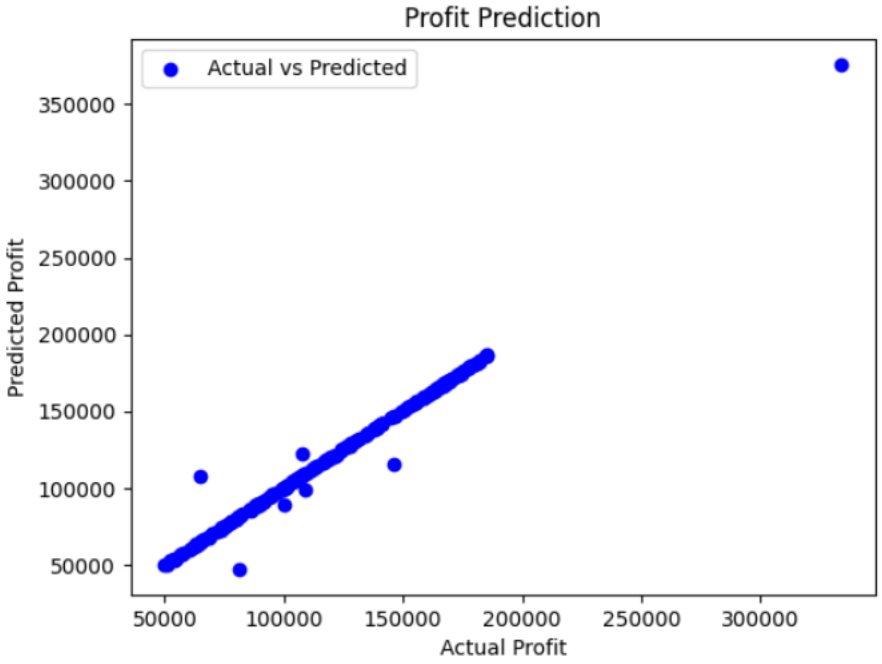

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.preprocessing import OneHotEncoder

from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score


# Load the dataset

data = pd.read_csv("1000_Companies.csv")


# Handle missing values by removing rows with NaN
```

```
data = data.dropna()

# Encode categorical variable (State) using OneHotEncoder
encoder = OneHotEncoder(drop='first', sparse_output=False)
state_encoded = encoder.fit_transform(data[['State']])
state_encoded_df = pd.DataFrame(state_encoded, columns=encoder.get_feature_names_out(['State']))

# Concatenate encoded state data with original dataset
data = pd.concat([data.drop(['State'], axis=1), state_encoded_df], axis=1)

# Analyze data distribution
print(data.describe())
print(data.info())

# Define independent (X) and dependent (y) variables
X = data[['R&D Spend', 'Administration', 'Marketing Spend'] + list(state_encoded_df.columns)]
y = data['Profit']

# Split the data (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train the model
model = LinearRegression()
model.fit(X_train, y_train)
```

```
# Predict the results
y_pred = model.predict(X_test)

# Visualize prediction
plt.scatter(y_test, y_pred, color='blue', label='Actual vs Predicted')
plt.xlabel("Actual Profit")
plt.ylabel("Predicted Profit")
plt.title("Profit Prediction")
plt.legend()
plt.show()

# Check values of coefficients and intercept
print(f"Coefficients: {model.coef_}")
print(f"Intercept: {model.intercept_}")

# Predict profit for given candidate dynamically
state_names = encoder.get_feature_names_out(['State'])
florida_encoded = (state_names == "State_Florida").astype(int)
candidate_features = np.array([91694.48, 515841.3, 11931.24] + list(florida_encoded)).reshape(1, -1)
profit_prediction = model.predict(candidate_features)
print(f"Predicted profit for given candidate: {profit_prediction[0]:.2f} US$")

# Calculate errors
```

```
mae = mean_absolute_error(y_test, y_pred)

mse = mean_squared_error(y_test, y_pred)

r2 = r2_score(y_test, y_pred)


print(f"Mean Absolute Error (MAE): {mae:.2f}")

print(f"Mean Squared Error (MSE): {mse:.2f}")

print(f"R-squared (R2) Score: {r2:.2f}")
```

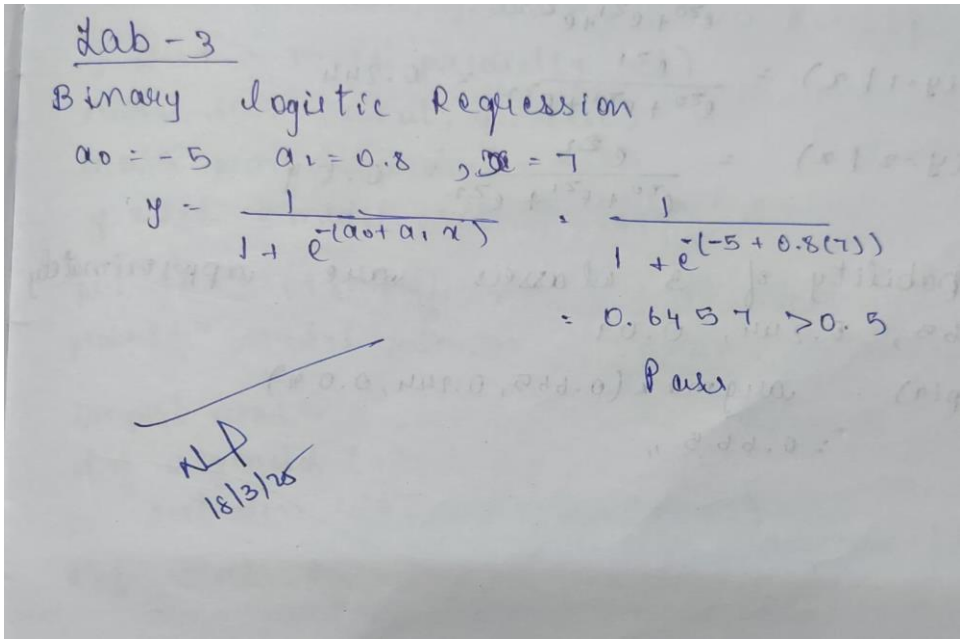
```
          R&D Spend  Administration  Marketing Spend         Profit  \
count    1000.000000     1000.000000      1000.000000    1000.000000
mean    81668.927200   122963.897612    226205.058419  119546.164656
std     46537.567891    12613.927535     91578.393542   42888.633848
min         0.000000    51283.140000         0.000000   14681.400000
25%     43084.500000   116640.684850    150969.584600   85943.198543
50%     79936.000000   122421.612150    224517.887350  117641.466300
75%    124565.500000   129139.118000    308189.808525  155577.107425
max    165349.200000   321652.140000    471784.100000  476485.430000

       State_Florida  State_New York
count    1000.000000     1000.000000
mean        0.322000        0.334000
std         0.467477        0.471876
min         0.000000        0.000000
25%         0.000000        0.000000
50%         0.000000        0.000000
75%         1.000000        1.000000
max         1.000000        1.000000
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 6 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   R&D Spend        1000 non-null   float64
 1   Administration   1000 non-null   float64
 2   Marketing Spend  1000 non-null   float64
 3   Profit           1000 non-null   float64
 4   State_Florida    1000 non-null   float64
 5   State_New York   1000 non-null   float64
dtypes: float64(6)
memory usage: 47.0 KB
None
```




```
Coefficients: [ 5.33045605e-01  1.13893831e+00  8.30755037e-02 -8.74491486e+02
 -9.71337988e+01]
Intercept: -82439.15560711118
Predicted profit for given candidate: 554066.30 US$
Mean Absolute Error (MAE): 1404.44
Mean Squared Error (MSE): 30775142.86
R-squared (R2) Score: 0.98
```

PROGRAM 4 Build Logistic Regression Model for a given dataset

Screenshot



Multi class logistic Regression

$x = [5.1, 3.5, 1.4, 0.2]$

$w_0 = [0.5, -0.2, 0.1, 0.3]$ bias, $b_0 = -1$

$w_1 = [-0.3, 0.4, -0.5, 0.2]$ $b_1 = 0.5$

$w_2 = [0.2, -0.1, 0.6, -0.4]$ $b_2 = 0$

$z_0 = w_0^T x + b_0$ Setosa

$z_1 = w_1^T x + b_1$ Versicolor

$z_2 = w_2^T x + b_2$ Virginica

$$\text{Softmax}(z_2) = \frac{e^{z_2}}{\sum_{j=1}^{k} e^{z_j}}$$

$z = [2, 1, 0]$

$$P(y=0|x) = \frac{e^{z_0}}{e^{z_0} + e^{z_1} + e^{z_2}} = 0.665$$

$$P(y=1|x) = \frac{e^{z_1}}{e^{z_0} + e^{z_1} + e^{z_2}} = 0.244$$

$$P(y=2|x) = \frac{e^{z_2}}{e^{z_0} + e^{z_1} + e^{z_2}} = 0.09$$

probability of 3 classes are approximately 0.665, 0.244, 0.09

$p(x) = \text{argmax}(0.665, 0.244, 0.09)$

$= 0.665$ //

Python code

Logical regression - binary

```
import pandas as pd
from matplotlib import pyplot as plt

df = pd.read_csv("insurance_data.csv")
df.head()

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(
                    df[['age']], df.bought_insurance, train_size=0.9,
                    random_state=10)
print("x train shape : ", x_train.shape)
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
model.score(x_test, y_test)
model.predict_proba(x_test)
y_pred = model.predict([[60]])
print("Model coffiicient : ", model.coef_)
print("Model intercept : ", model.intercept_)

import math
def sigmoid(x):
    return 1/(1 + math.exp(-x))

def pred_func(age):
    z = 0.127 * age - 4.973
    y = sigmoid(z)
    return y
```

```
age = 35
pred_func(age)
```

output :

Model x_train shape :- (24,1)
Model coefficient : [[0.12740657]]
Model intercept : [-4.97334]
~~0.3709x~~ Sigmoid function : 0.3709x3

Multi-class logistic Regression

```
from sklearn.datasets import load_iris
from sklearn ~~datasets~~ model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn import metrics
import matplotlib.pyplot as plt
iris = pd.read_csv("iris.csv")

x = iris.drop('species', axis='columns')
y = iris.species

x_train, x_test, y_train, y_test = train_test_split(x, y,
                    test_size = 0.2, random_state=42)
model = LogisticRegression(multi_class = 'multinomial')
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
acc = accuracy_score(y_test, y_pred)
print("Accuracy of model : {accuracy:.2f}")
conf_matrix = metrics.confusion_matrix(y_test, y_pred)
~~cm_display~~
print("Confusion matrix : ", conf_matrix)
```

Output :

Accuracy of the model : 1.00

Confusion matrix :

```
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

For HR dataset :

(i) which variable has a direct & clear impact on employee retention?

variables like satisfaction level, last_evaluation, promotion_last_5years have impact.

(ii) What is the accuracy of this model? Is it good?

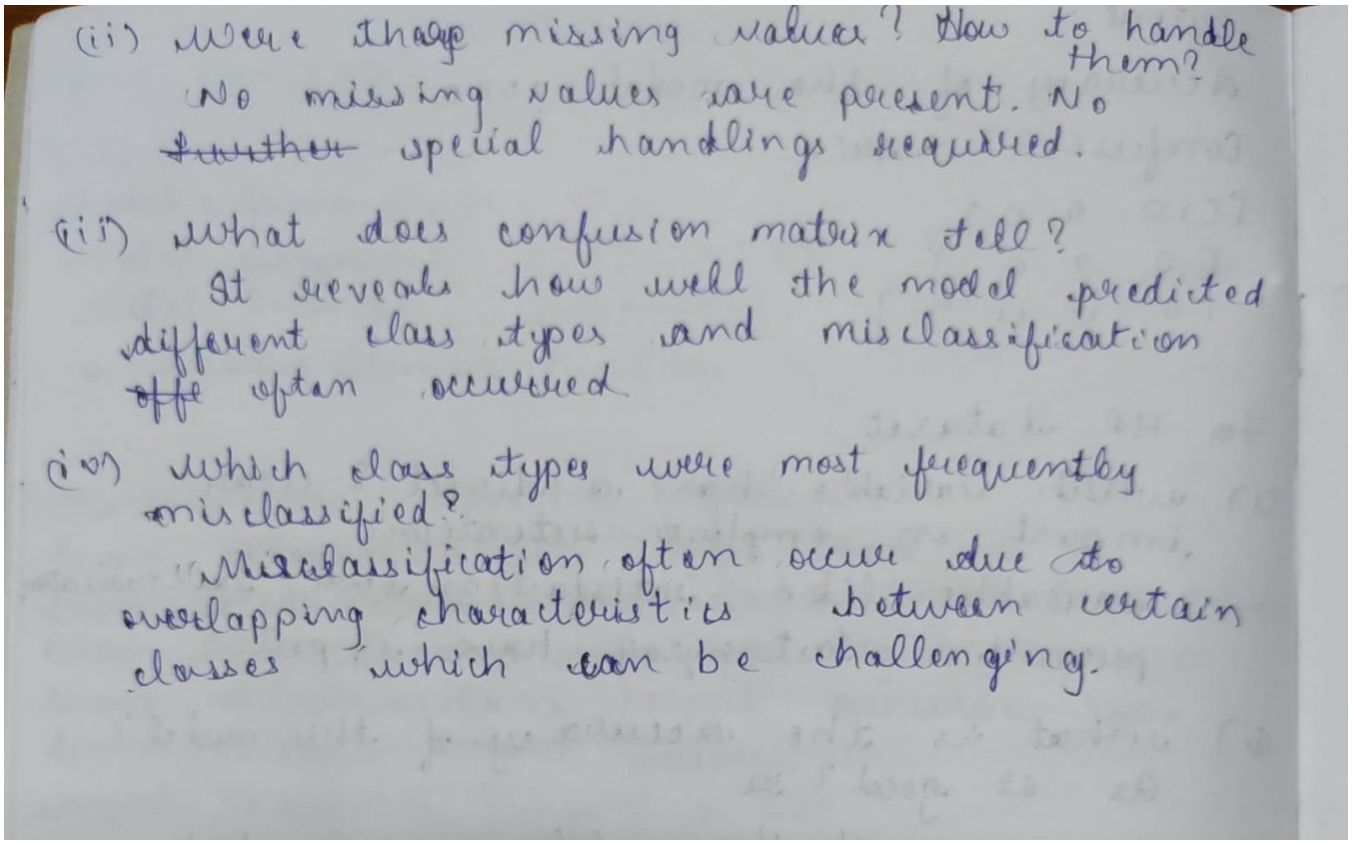
Accuracy of the model is 79% which is ~~unreasonable~~ manageable but need to be improved with more complex model.

For zoo :

(i) Did you perform data preprocessing steps? If yes, why?

Yes, it has steps like merging datasets, handling missing values & encoding categorical variables ~~to~~ prepare the data for logistic regression.

~~(ii)~~



Code

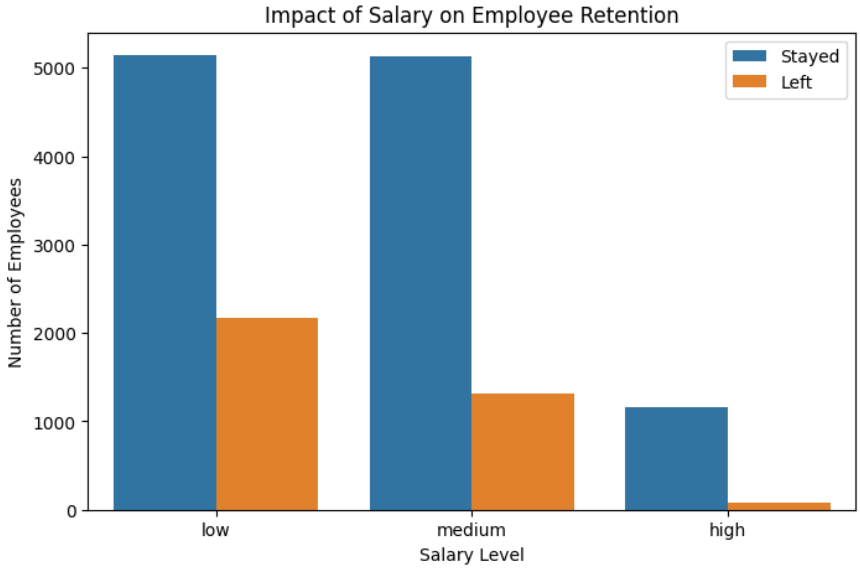
```
import pandas as pd

import numpy as np

df=pd.read_csv("/content/HR_comma_sep.csv")

df.head(3)

print(df.isnull().sum())

print(df.groupby('left').mean(numeric_only=True))

print(df.groupby('salary').mean(numeric_only=True))

import matplotlib.pyplot as plt

pd.crosstab(df.salary,df.left).plot(kind='bar')

plt.title('Employee Retention vs Salary')

plt.xlabel('Salary')
```

```
plt.ylabel('Number of Employees')

plt.show()

pd.crosstab(df.Department,df.left).plot(kind='bar')

plt.title('Employee Retention vs Department')

plt.xlabel('Department')

plt.ylabel('Number of Employees')

plt.show()

salary_dummies = pd.get_dummies(df.salary, prefix="salary")

dept_dummies = pd.get_dummies(df.Department, prefix="dept")


df_with_dummies = pd.concat([df, salary_dummies, dept_dummies], axis=1)


df_with_dummies = df_with_dummies.drop(['salary', 'Department'], axis=1)


X_features = ['satisfaction_level', 'last_evaluation', 'number_project', 'average_montly_hours',
'time_spend_company', 'Work_accident', 'promotion_last_5years'] + list(salary_dummies.columns) +
list(dept_dummies.columns)

X = df_with_dummies[X_features]

y = df_with_dummies.left



from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)


from sklearn.linear_model import LogisticRegression
```

```
model = LogisticRegression()

model.fit(X_train, y_train)


from sklearn.metrics import accuracy_score

y_pred = model.predict(X_test)


accuracy = accuracy_score(y_test, y_pred)

print("Accuracy of the model:", accuracy)
```

```
   satisfaction_level  last_evaluation  number_project  average_montly_hours  \
0                0.38             0.53               2                   157
1                0.80             0.86               5                   262
2                0.11             0.88               7                   272
3                0.72             0.87               5                   223
4                0.37             0.52               2                   159

   time_spend_company  Work_accident  left  promotion_last_5years Department  \
0                   3              0     1                      0      sales
1                   6              0     1                      0      sales
2                   4              0     1                      0      sales
3                   5              0     1                      0      sales
4                   3              0     1                      0      sales

   salary
0     low
1  medium
2  medium
3     low
4     low

Missing values:
 satisfaction_level       0
last_evaluation           0
number_project            0
average_montly_hours      0
time_spend_company        0
Work_accident             0
left                      0
promotion_last_5years     0
Department                0
salary                    0
dtype: int64
```

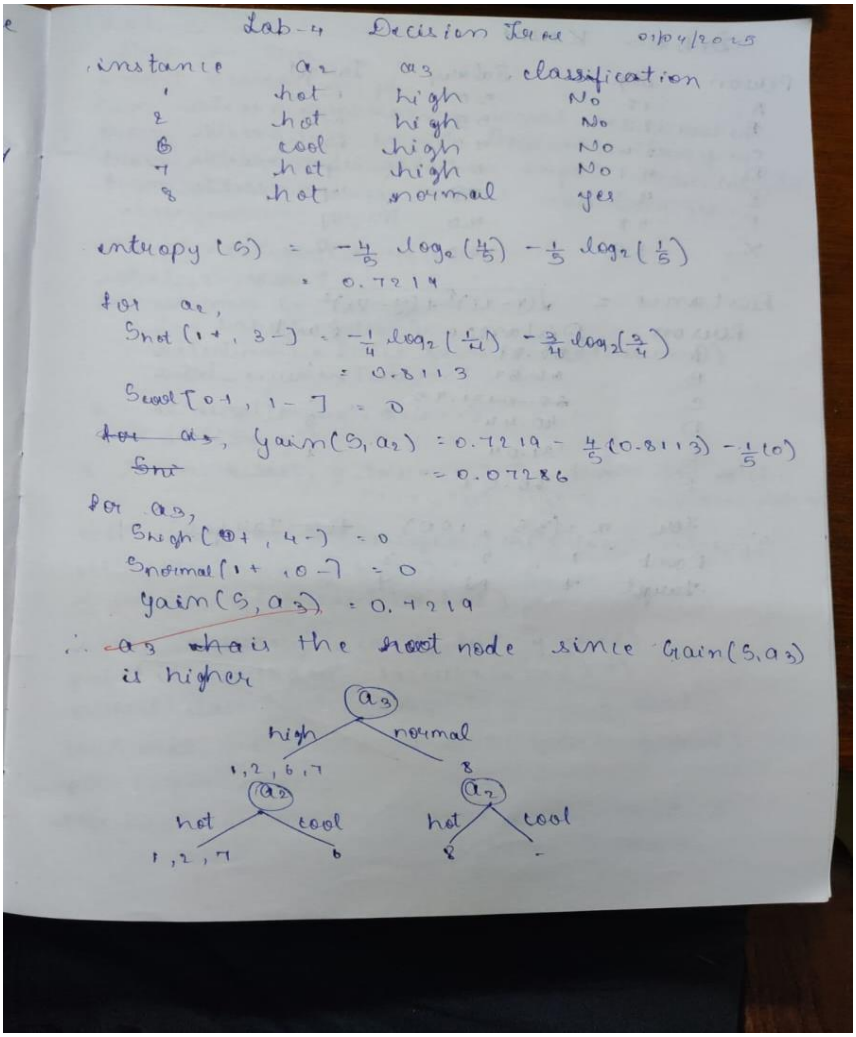
```
last_evaluation          float64
number_project             int64
average_montly_hours       int64
time_spend_company         int64
Work_accident              int64
left                       int64
promotion_last_5years      int64
Department                object
salary                    object
dtype: object

Unique values in categorical columns:
 Department    10
salary         3
dtype: int64
```



PROGRAM 5

Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample.

Screenshot



Lab - 5

Decision Tree

01/04/2025

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,
    classification_report

df = pd.read_csv('drug.csv')
label_encoder = {}
for column in df.columns:
    le = LabelEncoder()
    df[column] = le.fit_transform(df[column])
    label_encoder[column] = le

x = df.drop('Drug', axis = 1)
y = df['Drug']
x_train, x_test, y_train, y_test = train_test_split
                (x, y, test_size=0.2, random_state=42)

clf = DecisionTreeClassifier(criterion = 'entropy')
clf.fit(x_train, y_train)
y_pred = clf.predict(x_test)

acc = accuracy_score(y_test, y_pred)
print(f'Accuracy = {accuracy:.2f} ')
print(classification_report(y_test, y_pred))
cnf_mat = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(10,4))
sns.heatmap(cnf_mat, annot=True, fmt='d',
        cmap='Blues', xticklabels=le.classes_,
        yticklabels=le.classes_)
```

~~plt.rc('font')~~

plt.figure(figsize=(12,8))

.plot_tree(clf, filled=True, feature_name=X.colu

plt.show

output

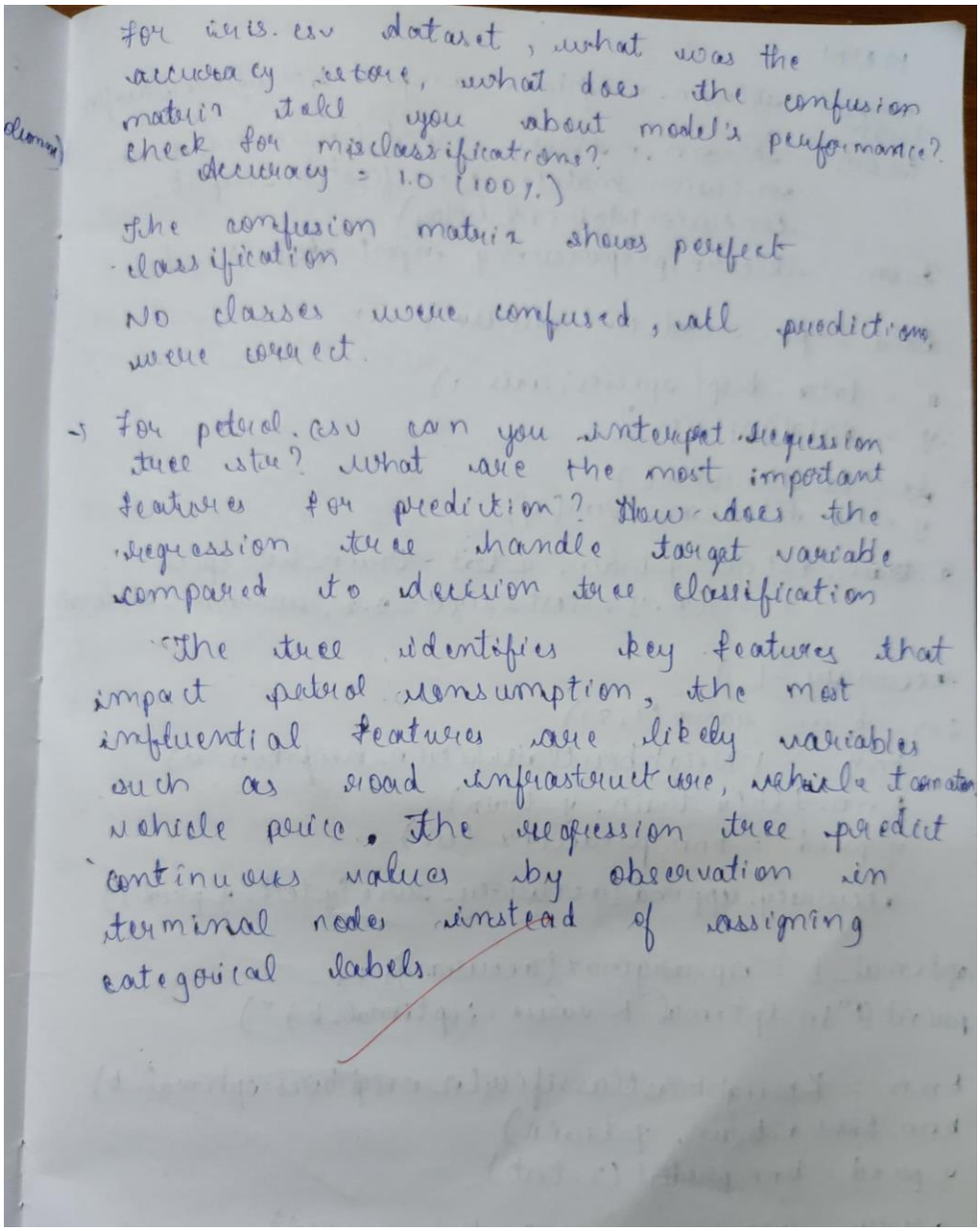
accuracy = 1.00

	precision	recall	f1-score	support
0	1	1	1	1
1	1	1	1	3
2	1	1	1	5
3	1	1	1	11
4	1	1	1	15
accuracy			1	40
macro avg	1	1	1	40
weighted avg	1	1	1	40

confusion matrix

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 11 & 0 \\ 0 & 0 & 0 & 0 & 15 \end{bmatrix}$$

11/4/25



for iris.csv dataset, what was the accuracy score, what does the confusion matrix tell you about model's performance? check for misclassifications?

Accuracy = 1.0 (100%)

The confusion matrix shows perfect classification

No classes were confused, all predictions were correct.

→ for petrol.csv can you interpret regression tree also? what are the most important features for prediction? How does the regression tree handle target variable compared to decision tree classification

The tree identifies key features that impact petrol consumption, the most influential features are likely variables such as road infrastructure, vehicle taxation, vehicle price. The regression tree predict continuous values by observation in terminal nodes instead of assigning categorical labels.

Code

```
from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy_score, confusion_matrix

from sklearn import tree
```

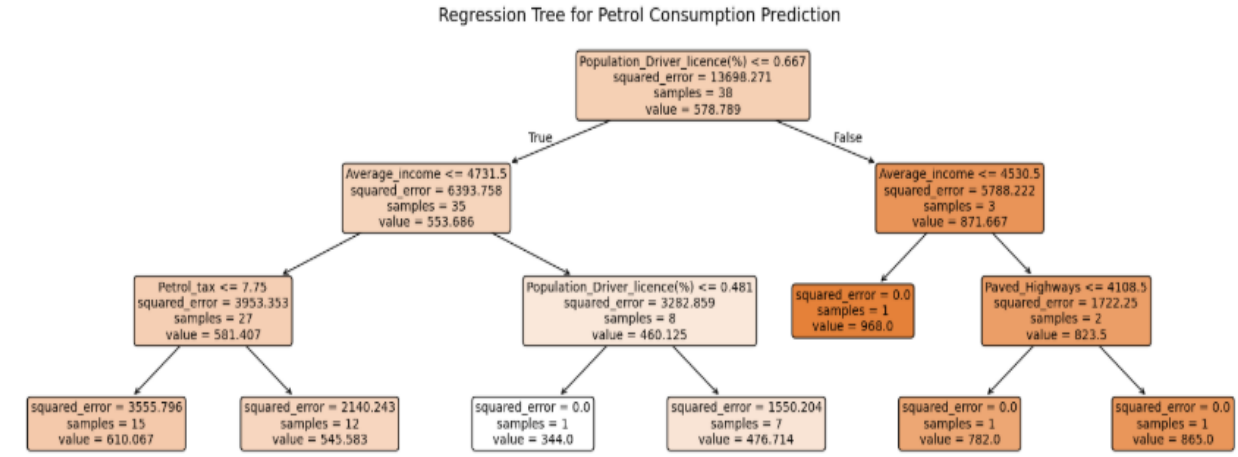
```
import matplotlib.pyplot as plt


iris = load_iris()

X = iris.data

y = iris.target


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


clf = DecisionTreeClassifier()


clf.fit(X_train, y_train)


y_pred = clf.predict(X_test)


accuracy = accuracy_score(y_test, y_pred)

conf_matrix = confusion_matrix(y_test, y_pred)


print("Accuracy:", accuracy)

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



plt.figure(figsize=(12, 8))

tree.plot_tree(clf, feature_names=iris.feature_names, class_names=iris.target_names, filled=True)

plt.show()
```

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn import tree
import matplotlib.pyplot as plt

iris = load_iris()
X = iris.data
y = iris.target

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

clf = DecisionTreeClassifier()

clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

print("Accuracy:", accuracy)
```

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



plt.figure(figsize=(12, 8))
tree.plot_tree(clf, feature_names=iris.feature_names, class_names=iris.target_names, filled=True)
plt.show()

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error
import numpy as np # import numpy

data = pd.read_csv("petrol_consumption.csv")

X = data[['Petrol_tax', 'Average_income', 'Paved_Highways',
        'Population_Driver_licence(%)']]
y = data['Petrol_Consumption']

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

regressor = DecisionTreeRegressor()
```

```
regressor.fit(X_train, y_train)


y_pred = regressor.predict(X_test)


mae = mean_absolute_error(y_test, y_pred)

mse = mean_squared_error(y_test, y_pred)

rmse = np.sqrt(mse)


print("Mean Absolute Error:", mae)

print("Mean Squared Error:", mse)

print("Root Mean Squared Error:", rmse)

from sklearn.tree import plot_tree

import matplotlib.pyplot as plt


plt.figure(figsize=(15, 10))

# Assuming 'data' is your original pandas DataFrame

plot_tree(regressor, feature_names=data[['Petrol_tax', 'Average_income', 'Paved_Highways',
'Population_Driver_licence(%)']].columns, filled=True, rounded=True)

plt.show()
```

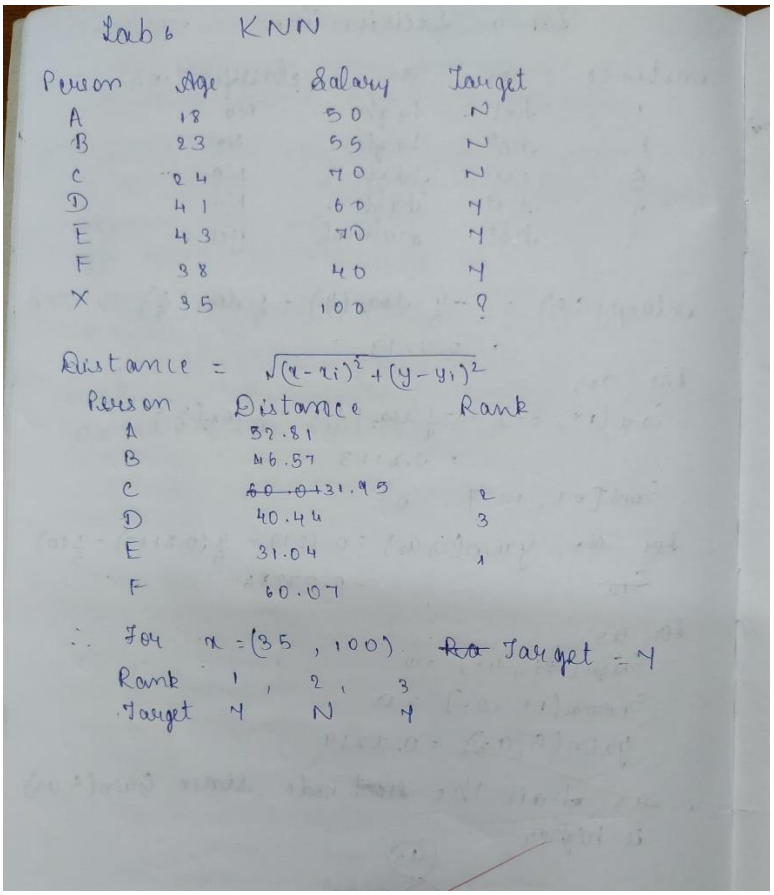
```
Regression Tree Evaluation Metrics:
Mean Absolute Error (MAE): 80.63
Mean Squared Error (MSE): 14718.40
Root Mean Squared Error (RMSE): 121.32
```



PROGRAM 6

Build KNN Classification model for a given dataset.

Screenshot



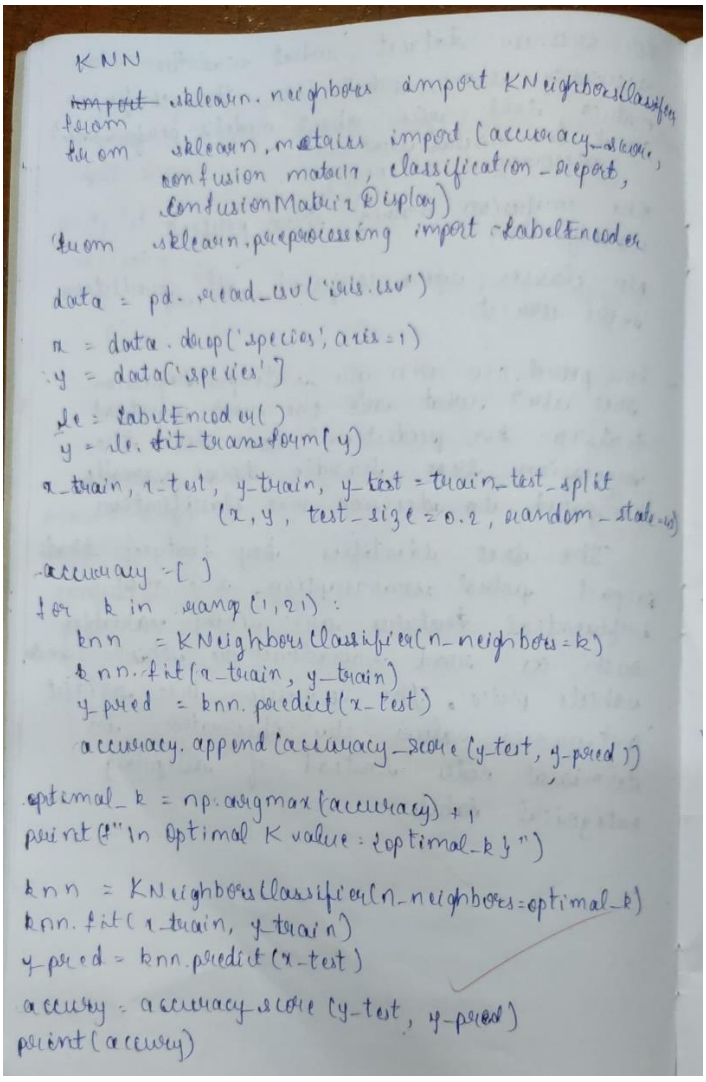
→ for iris data set how to choose k

k = 3 gives 100% accuracy but generalize data. k = 5, error rate = 1 - accuracy = 0 (no misclassification), to find best k, test multiple values & plot error rate.

→ for diabetes data what is the purpose of feature scaling? How to perform?

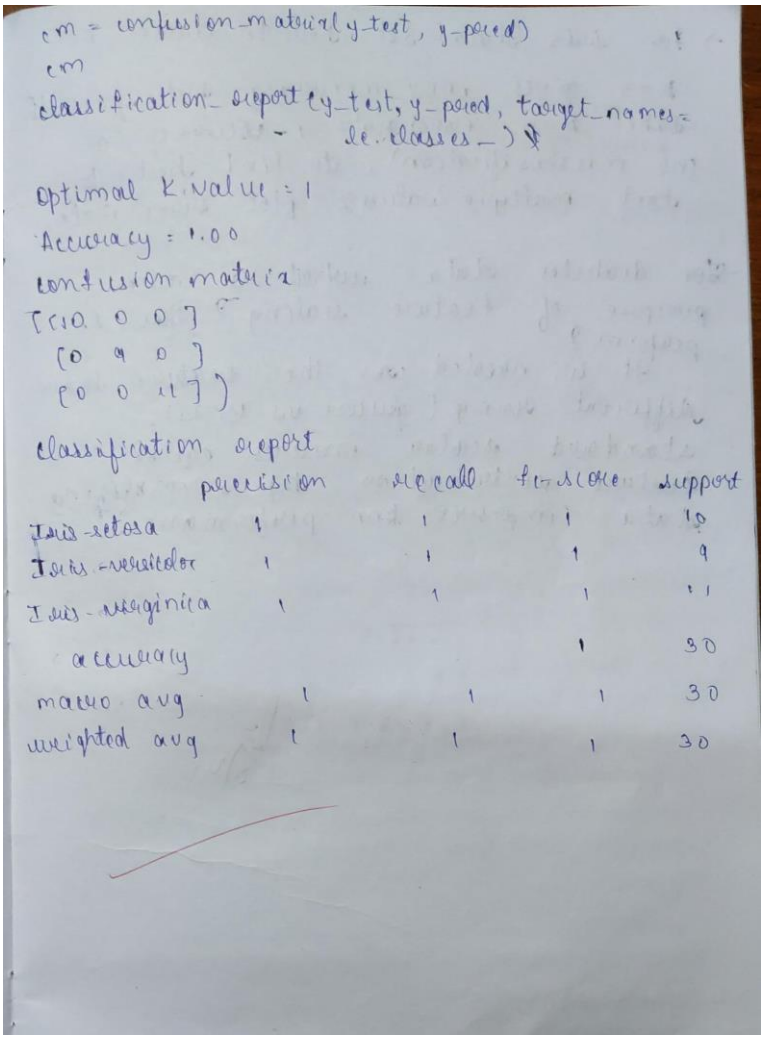
It is needed as the feature have different range (glucose vs BMI), standard scaler ensures equal feature contribution by normalizing data (improves knn performance)

8/4/1



KNN

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import (accuracy_score,
    confusion matrix, classification_report,
    ConfusionMatrixDisplay)
from sklearn.preprocessing import LabelEncoder

data = pd.read_csv('iris.csv')

x = data.drop('species', axis=1)
y = data['species']

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

x_train, x_test, y_train, y_test = train_test_split
                (x, y, test_size=0.2, random_state=0)

accuracy = []
for k in range(1,21):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(x_train, y_train)
    y_pred = knn.predict(x_test)
    accuracy.append(accuracy_score(y_test, y_pred))

optimal_k = np.argmax(accuracy) + 1
print(f"\n Optimal K value : {optimal_k}")

knn = KNeighborsClassifier(n_neighbors=optimal_k)
knn.fit(x_train, y_train)
y_pred = knn.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
print(accuracy)
```



```
cm = confusion_matrix(y_test, y_pred)
cm
classification_report(y_test, y_pred, target_names=
                      le.classes_)

Optimal K value = 1
Accuracy = 1.00
confusion matrix
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
classification report
                  precision    recall  f1-score   support

     Iris-setosa       1.00      1.00      1.00        10
 Iris-versicolor       1.00      1.00      1.00         9
  Iris-virginica       1.00      1.00      1.00        11

        accuracy                           1.00        30
       macro avg       1.00      1.00      1.00        30
    weighted avg       1.00      1.00      1.00        30
```

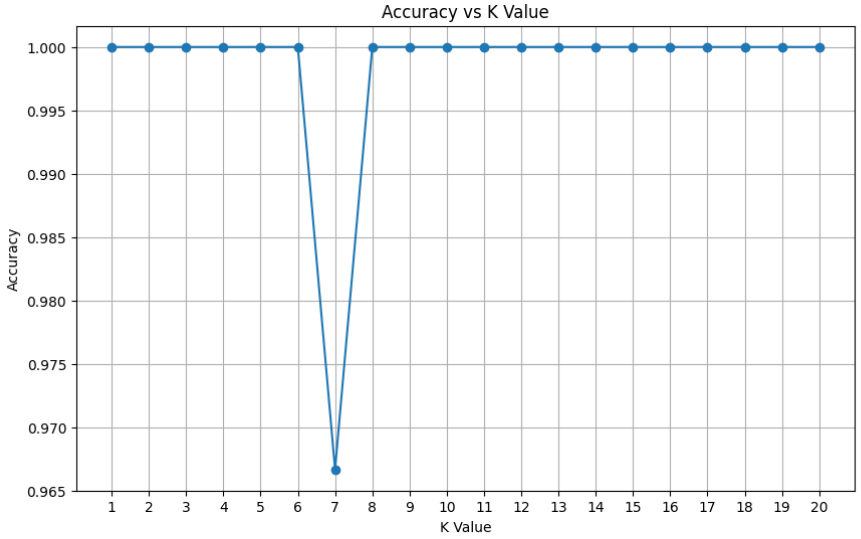
Code

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import seaborn as sns
import matplotlib.pyplot as plt


try:
```

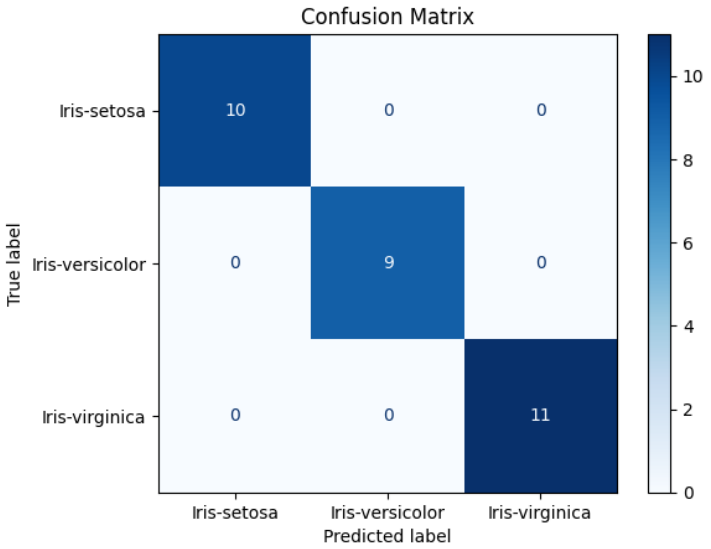
```
    data = pd.read_csv('/content/iris (1).csv')
except FileNotFoundError:
    print("Error: 'iris.csv' not found. Please upload the file to your Colab environment.")
    exit()

X = data.drop('species', axis=1)
y = data['species']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print("Accuracy Score:", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix:")
cm = confusion_matrix(y_test, y_pred)
print(cm)

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=knn.classes_, yticklabels=knn.classes_)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

print("\nClassification Report:")

print(classification_report(y_test, y_pred))



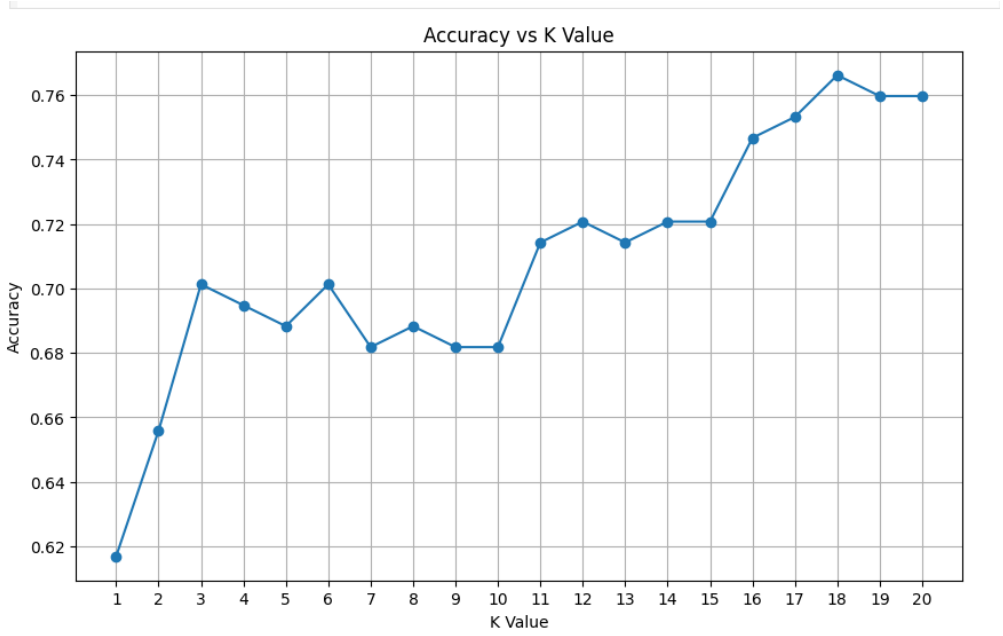

```
Optimal K value: 1

Accuracy: 1.0000

Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```




```
Classification Report:
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        10
Iris-versicolor       1.00      1.00      1.00         9
 Iris-virginica       1.00      1.00      1.00        11

       accuracy                           1.00        30
      macro avg       1.00      1.00      1.00        30
   weighted avg       1.00      1.00      1.00        30
```

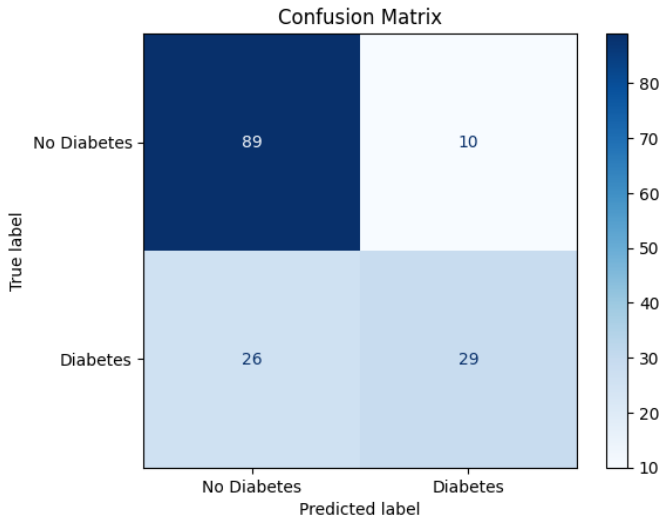
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.preprocessing import StandardScaler
import seaborn as sns
import matplotlib.pyplot as plt

try:
    diabetes = pd.read_csv('diabetes.csv')
except FileNotFoundError:
    print("Error: 'diabetes.csv' not found. Please ensure the file is in the current directory.")
    exit()

X = diabetes.drop('Outcome', axis=1)
y = diabetes['Outcome']

scaler = StandardScaler()
X = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
```

```
print(f"Accuracy: {accuracy}")

cm = confusion_matrix(y_test, y_pred)

print("Confusion Matrix:")

print(cm)

sns.heatmap(cm, annot=True, fmt="d")

plt.title('Confusion Matrix')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()


print("Classification Report:")

print(classification_report(y_test, y_pred))
```




```
Optimal K value: 18

Accuracy: 0.7662
```

```
Confusion Matrix:
[[89 10]
 [26 29]]
```

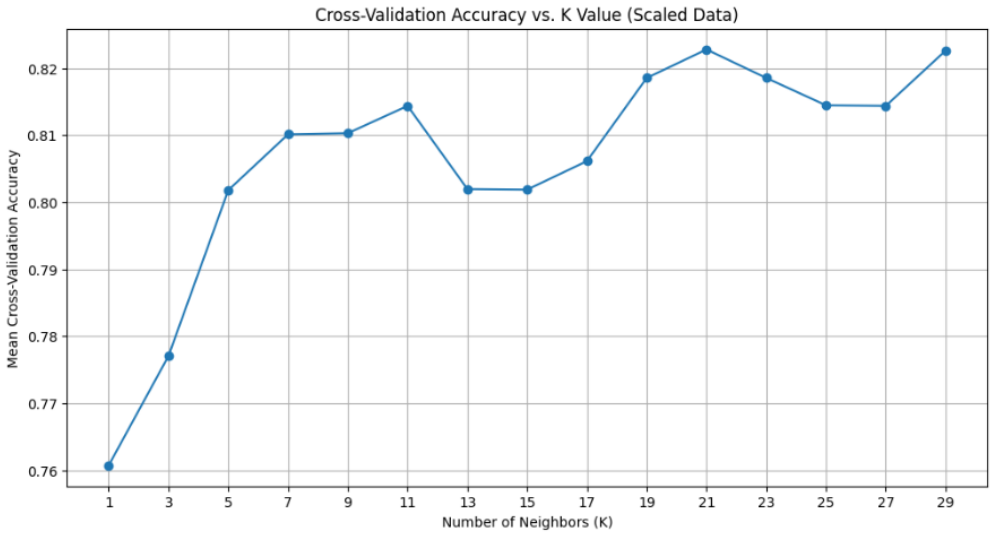



```
Model Evaluation:
- True Positives: 29
- True Negatives: 89
- False Positives: 10
- False Negatives: 26
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler


try:
    heart = pd.read_csv('heart.csv')
except FileNotFoundError:
    print("Error: 'heart.csv' not found. Please ensure the file is in the current directory.")
    exit()
```

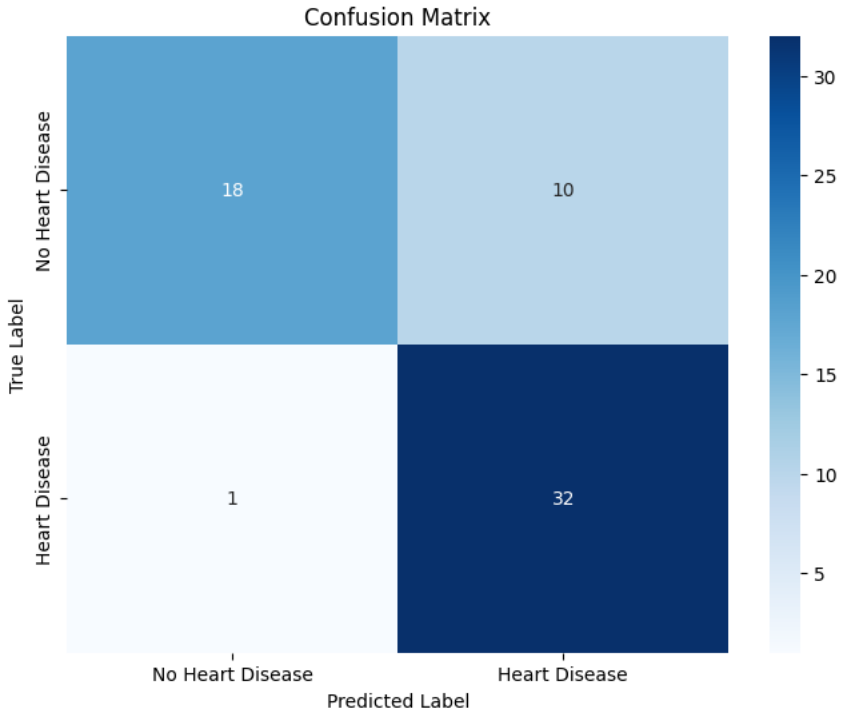
```
X = heart.drop('target', axis=1)
y = heart['target']

scaler = StandardScaler()
X = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
best_k = 1
best_accuracy = 0

for k in range(1, 21):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_k = k

print(f"Best k: {best_k} with accuracy {best_accuracy}")
knn = KNeighborsClassifier(n_neighbors=best_k)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
```

```
print(f"Accuracy: {accuracy}")


cm = confusion_matrix(y_test, y_pred)

print("Confusion Matrix:")

print(cm)


sns.heatmap(cm, annot=True, fmt="d")

plt.title('Confusion Matrix')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()


print("Classification Report:")

print(classification_report(y_test, y_pred))
```




```
The optimal K value found through cross-validation is: 21

K-Nearest Neighbors Classifier with K = 21 (Scaled Data)
Accuracy Score on Test Data: 0.819672131147541
```

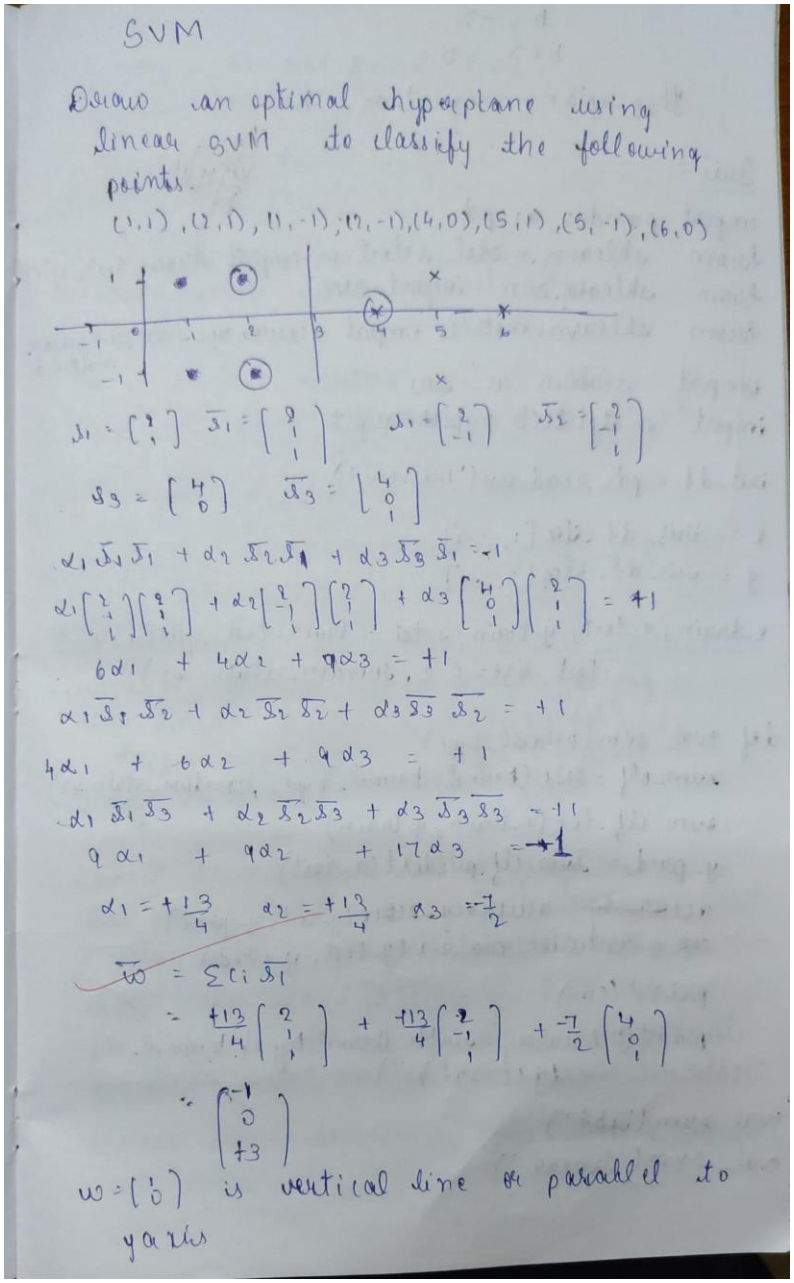
```
K-Nearest Neighbors Classifier with K = 21 (Scaled Data)
Accuracy Score on Test Data: 0.819672131147541
```




```
Classification Report on Test Data:
              precision    recall  f1-score   support

           0       0.95      0.64      0.77        28
           1       0.76      0.97      0.85        33

    accuracy                           0.82        61
   macro avg       0.85      0.81      0.81        61
```

PROGRAM 7 Build Support vector machine model for a given dataset

Screenshot



b = -3

b+3 = 0

Hyperplane equation b+3 = 0

Iris:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
iris_df = pd.read_csv('iris.csv')
x = iris_df.iloc[:, :-1]
y = iris_df.iloc[:, -1]
x_train, x_test, y_train, y_test = train_test_split(x, y,
        test_size=0.2, random_state=42)
def eval_svm(kernel_type):
    svm_clf = SVC(kernel=kernel_type, random_state=42)
    svm_clf.fit(x_train, y_train)
    y_pred = svm_clf.predict(x_test)
    accuracy = accuracy_score(y_test, y_pred)
    cm = confusion_matrix(y_test, y_pred)
    print(cm)
    print(f"Accuracy with {kernel_type} kernel:
          {acc:.4f}")
eval_svm('rbf')
eval_svm('linear')
```

Output:

Accuracy with rbf kernel: 1.000

confusion matrix

```
10  0  0
 0  9  0
 0  0 11
```

Accuracy with linear kernel: 1.00

confusion matrix

```
10  0  0   - setosa
 0  9  0   - versicolor
 0  0 11   - virginica
setosa versicolor virginica
```

Observations

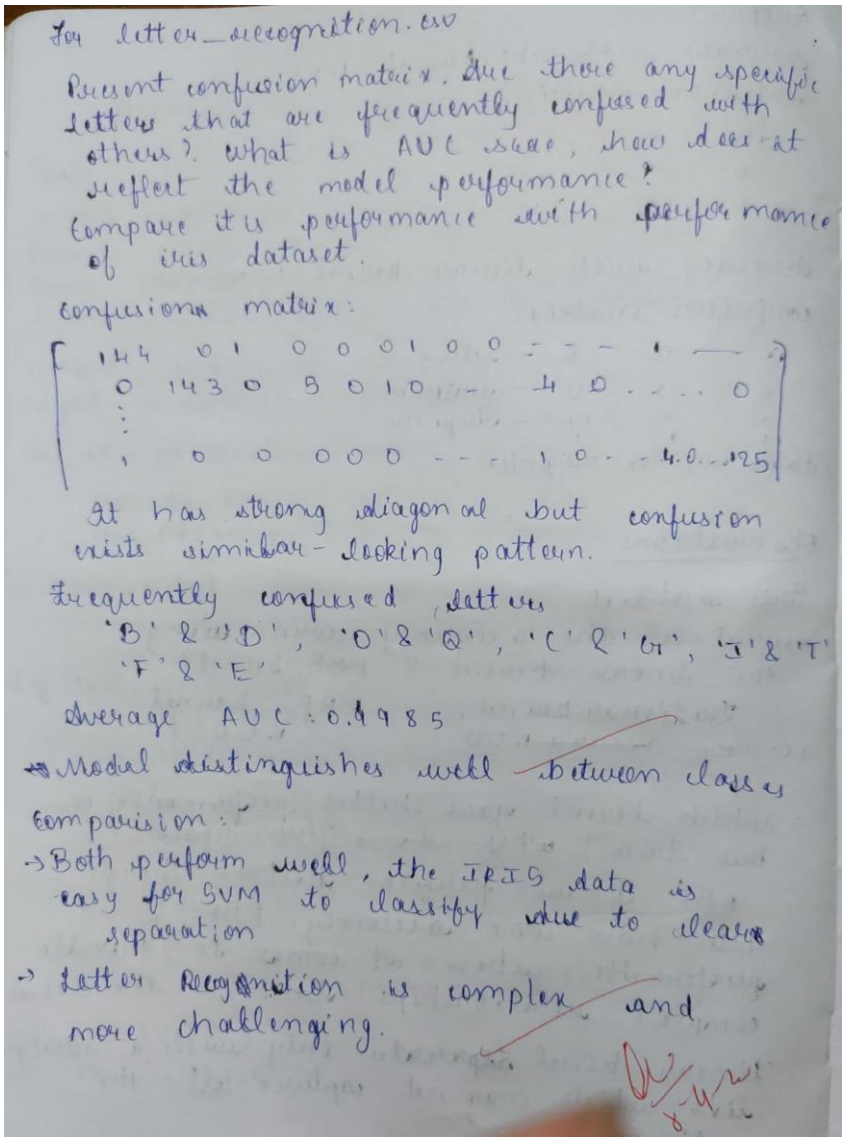
Iris dataset

→ what is the accuracy score using the linear kernel & RBF kernel?

	For linear kernel,	RBF kernel
accuracy	1.00	1.00

→ which kernal gave better performance on iris data? ~~why~~ give reason

RBF kernel perform better. Though both have 100% accuracy. RBF is preferable when it comes to handle complex relationships between the classes. Linear kernel separates only with a straight line which may not capture all the patterns.



For letter_recognition.csv

Present confusion matrix. Are there any specific letters that are frequently confused with others? What is AUC score, how does it reflect the model performance?
Compare its performance with performance of iris dataset.

confusion matrix:

[144 0 1 0 0 0 1 0 0 - - - 1 - ?
0 143 0 3 0 1 0 - 4 0 - - - 0
⋮
1 0 0 0 0 0 - - - 1 0 - - 4 0 - - 125]

It has strong diagonal but confusion exists similar-looking pattern.

frequently confused letters
'B' & 'D', 'O' & 'Q', 'C' & 'G', 'I' & 'T'
'F' & 'E'

Average AUC: 0.9985

→ Model distinguishes well between classes

comparision:

→ Both perform well, the IRIS data is easy for SVM to classify due to clear separation

→ Letter Recognition is complex and more challenging.

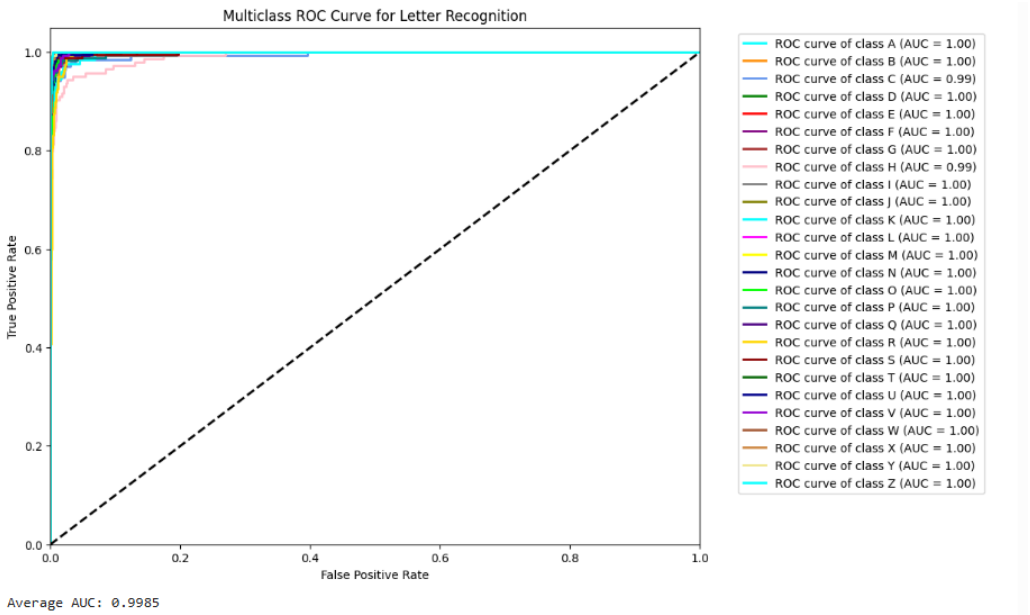
Code

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve, auc
from sklearn.preprocessing import label_binarize
import matplotlib.pyplot as plt
```

```
from itertools import cycle

data = pd.read_csv('letter-recognition.csv')
X = data.drop('letter', axis=1)
y = data['letter']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

svm = SVC(kernel='rbf', probability=True, random_state=42)
svm.fit(X_train, y_train)

y_pred = svm.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")

conf_mat = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:")
print(conf_mat)

y_test_bin = label_binarize(y_test, classes=np.unique(y))
n_classes = y_test_bin.shape[1]

fpr = dict()
tpr = dict()
```

```
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], svm.predict_proba(X_test)[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

plt.figure(figsize=(10, 8))
colors = cycle(['aqua', 'darkorange', 'cornflowerblue', 'green', 'red', 'purple',
                'brown', 'pink', 'gray', 'olive', 'cyan', 'magenta', 'yellow',
                'navy', 'lime', 'teal', 'indigo', 'gold', 'darkred', 'darkgreen',
                'darkblue', 'darkviolet', 'sienna', 'peru', 'khaki'])

for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=2,
             label='ROC curve of class {0} (AUC = {1:0.2f})'
             ''.format(np.unique(y)[i], roc_auc[i]))

plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Multiclass ROC Curve for Letter Recognition')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()
```

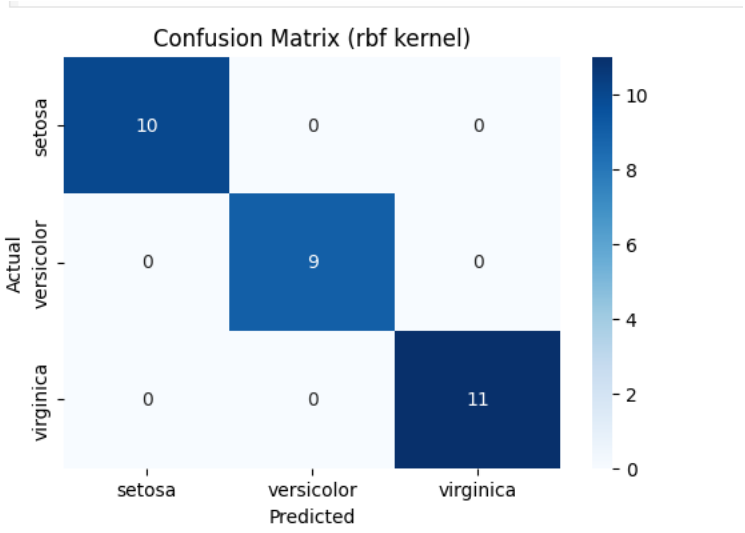
```
avg_auc = np.mean(list(roc_auc.values()))

print(f"\nAverage AUC: {avg_auc:.4f}")
```



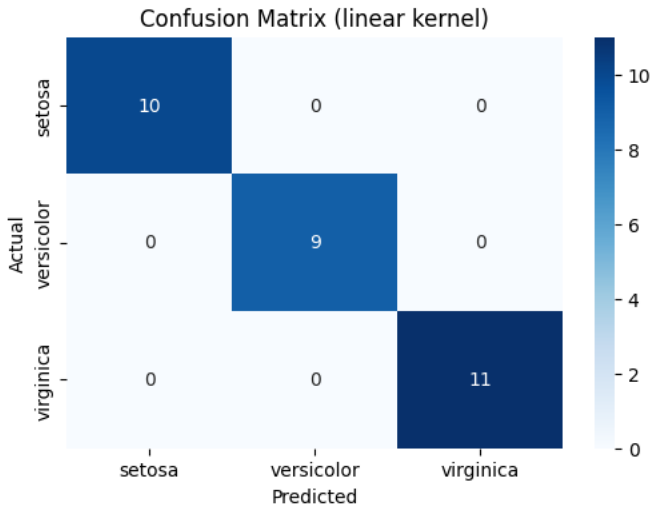

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.svm import SVC

from sklearn.metrics import accuracy_score, confusion_matrix

import seaborn as sns

import matplotlib.pyplot as plt



iris_df = pd.read_csv('iris.csv')

X = iris_df.iloc[:, :-1]

y = iris_df.iloc[:, -1]

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

```
def evaluate_svm(kernel_type):
    # Create SVM classifier
    svm_clf = SVC(kernel=kernel_type, random_state=42)

    # Train the model
    svm_clf.fit(X_train, y_train)

    # Make predictions
    y_pred = svm_clf.predict(X_test)

    # Calculate accuracy
    accuracy = accuracy_score(y_test, y_pred)

    # Generate confusion matrix
    cm = confusion_matrix(y_test, y_pred)

    # Plot confusion matrix
    plt.figure(figsize=(6, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=svm_clf.classes_,
                yticklabels=svm_clf.classes_)
    plt.title(f'Confusion Matrix ({kernel_type} kernel)')
    plt.ylabel('Actual')
```

```
    plt.xlabel('Predicted')
    plt.show()

    print(f"Accuracy with {kernel_type} kernel: {accuracy:.4f}")
    print("\n")

evaluate_svm('rbf')
evaluate_svm('linear')
```




```
Accuracy with rbf kernel: 1.0000
```

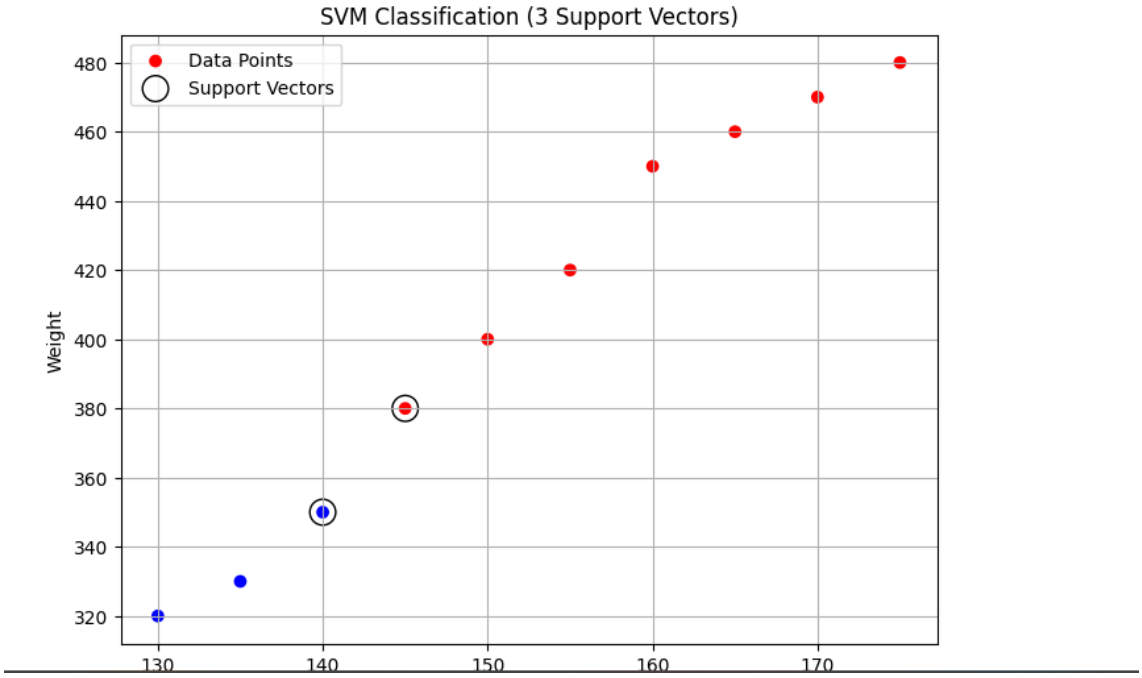



```
Accuracy with linear kernel: 1.0000
```

```
import pandas as pd

import matplotlib.pyplot as plt

from sklearn.svm import SVC

from sklearn.preprocessing import LabelEncoder


df = pd.read_csv("horse_mule_data.csv")


# Encode 'Horse'=0, 'Mule'=1

df['Label'] = LabelEncoder().fit_transform(df['Label'])



X = df[['Height', 'Weight']]

y = df['Label']
```

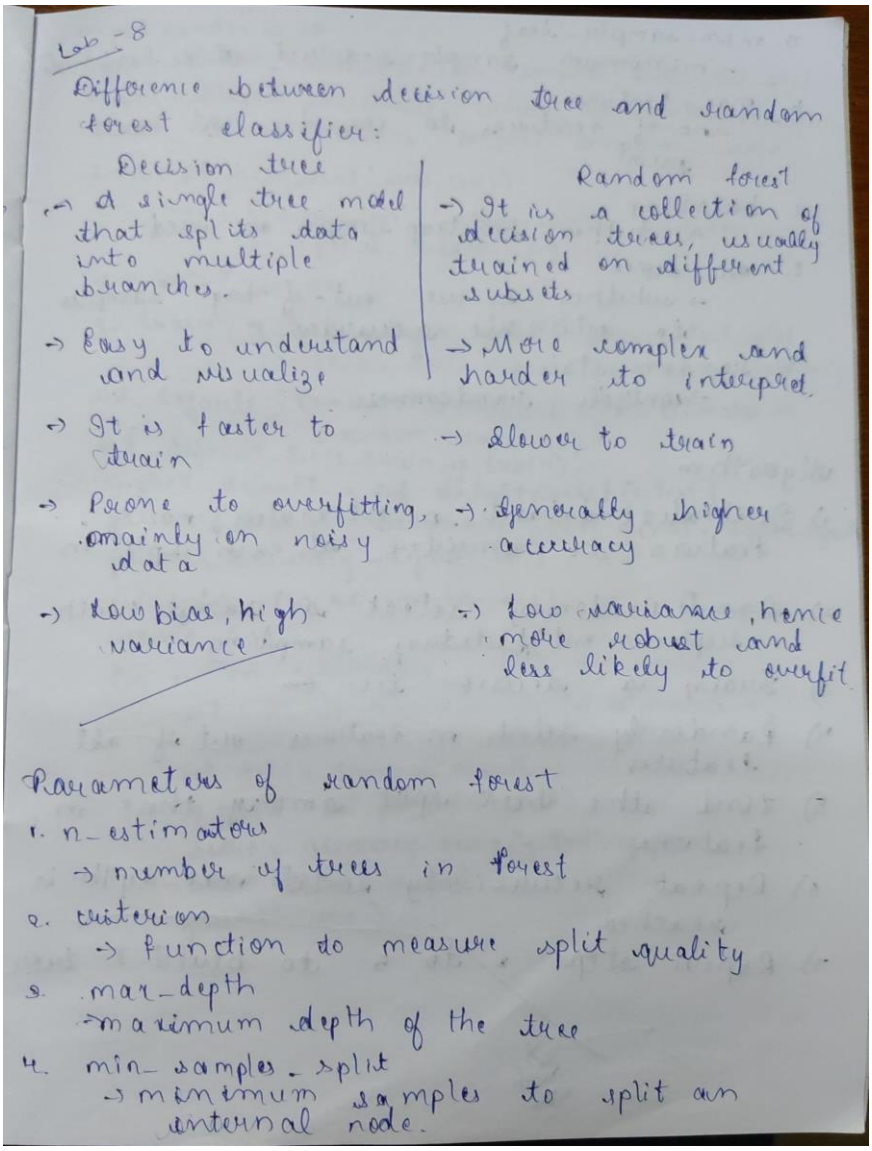
```
model = SVC(kernel='linear', C=1000)  # High C -> fewer support vectors
model.fit(X, y)

support_vectors = model.support_vectors_
accuracy = model.score(X, y)

print("Accuracy:", accuracy)
print("Support Vectors:\n", support_vectors)
print("Number of Support Vectors:", len(support_vectors))

colors = ['red' if label == 0 else 'blue' for label in y]
plt.figure(figsize=(8,6))
plt.scatter(X['Height'], X['Weight'], c=colors, label='Data Points')

plt.scatter(support_vectors[:, 0], support_vectors[:, 1],
            s=200, facecolors='none', edgecolors='black', label='Support Vectors')

plt.xlabel("Height")
plt.ylabel("Weight")
plt.title("SVM Classification (3 Support Vectors)")
plt.legend()
plt.grid(True)
plt.show()
```

```
Accuracy: 1.0
Support Vectors:
 [[145. 380.]
 [140. 350.]]
Number of Support Vectors: 2
```



PROGRAM 8 Implement Random forest ensemble method on a given dataset.

Screenshot



Lab - 8

Difference between decision tree and random forest classifier:

Decision tree	Random forest
→ A single tree model that splits data into multiple branches.	→ It is a collection of decision trees, usually trained on different subsets
→ Easy to understand and visualize	→ More complex and harder to interpret.
→ It is faster to train	→ Slower to train
→ Prone to overfitting, mainly on noisy data	→ Generally higher accuracy
→ Low bias, high variance	→ Low variance, hence more robust and less likely to overfit

Parameters of random forest

1. n_estimators
   → number of trees in forest
2. criterion
   → function to measure split quality
3. max_depth
   → maximum depth of the tree
4. min_samples_split
   → minimum samples to split an internal node.

5. min_samples_leaf
   → minimum samples required at a leaf node
6. max_features
   → no. of features to consider at each split
7. bootstrap
   → whether bootstrap samples are used
8. oob_score
   → whether to use out-of-bag samples to estimate accuracy.
9. random_state
   → controls randomness

Algorithm

1) Initialize dataset, n_estimators, no. of features to consider at each split m
2) ~~Rando~~ Randomly select a sample (with replacement) bootstrap sampling
3) Train a decision tree ~~on~~
4) Randomly select m features out of all features
5) Find the best split among those m features
6) Repeat recursively until max_depth is reached
7) Repeat steps 2 to 5 to build T trees

19/2/25

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
iris = pd.read_csv('iris.csv')
x = iris [['sepal_len', 'sepal_width', 'petal_len',
           'petal_width']]
y = iris ['species']
x_train, x_test, y_train, y_test = train_test_split
                    (x, y, test_size = 0.2, random_state=42)
rf_default = RandomForestClassifier(n_estimators=10,
rf_default.fit(x_train, y_train)
                                    random_state=42)
y_pred_default = rf_default.predict(x_test)
accuracy_default = accuracy_score(y_test, y_pred_default)
print(accuracy_default:.4f)
estimators = [10, 50, 100, 200, 500]
score = [ ]
for n in estimators
    rf = RandomForestClassifier(n_estimators=n,
                                random_state=42)
    rf.fit(x_train, y_train)
    y_pred = rf.predict(x_test)
    score = accuracy_score(y_test, y_pred)
    score.append(score)
    print(score)
```

```
best_n = estimators[score.index(max(score))]
best_score = accuracy_score(y_test, y_pred)
print(best_n)
print(best_score)
```

o/p

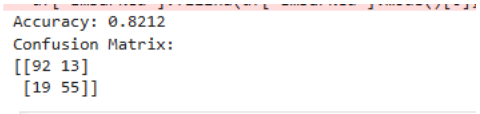
accuracy with n estimators = 10 : 1.000
= 50 : 1
= 100 : 1
= 200 : 1
= 500 : 1

Best n_estimator = 10
Best accuracy score = 1

Code

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.preprocessing import LabelEncoder

df = pd.read_csv("train.csv")

df = df.drop(columns=['PassengerId', 'Name', 'Ticket', 'Cabin'])

df['Age'].fillna(df['Age'].median(), inplace=True)
df['Embarked'].fillna(df['Embarked'].mode()[0], inplace=True)

label_encoders = {}
for col in ['Sex', 'Embarked']:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le

X = df.drop(columns=['Survived'])
y = df['Survived']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

print(f"Accuracy: {accuracy:.4f}")
print("Confusion Matrix:")
print(conf_matrix)
```



```
Accuracy: 0.8212
Confusion Matrix:
[[92 13]
 [19 55]]
```

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

iris = pd.read_csv("iris.csv")
X = iris.iloc[:,:-1]
y = iris.iloc[:,-1]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

rf_classifier = RandomForestClassifier(random_state=42)
```

```
rf_classifier.fit(X_train, y_train)

y_pred = rf_classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy with default n_estimators (10): {accuracy:.4f}")

best_accuracy = 0
best_n_estimators = 0

for n_estimators in range(10, 201, 10):
    rf_classifier = RandomForestClassifier(n_estimators=n_estimators, random_state=42)
    rf_classifier.fit(X_train, y_train)
    y_pred = rf_classifier.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)

    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_n_estimators = n_estimators

print(f"Best accuracy: {best_accuracy:.4f} achieved with n_estimators = {best_n_estimators}")
```

```
Accuracy with default n_estimators (10): 1.0000
```

Best accuracy: 1.0000 achieved with n_estimators = 10

In [12]:

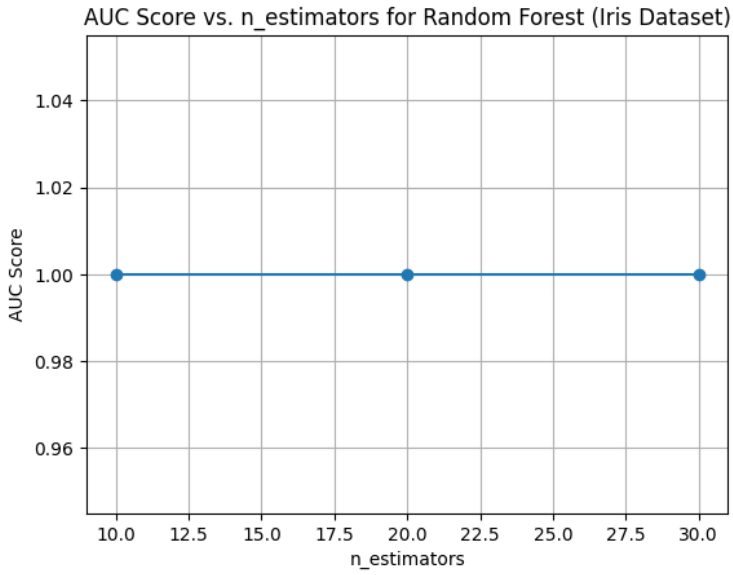
```
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
from sklearn.preprocessing import label_binarize
from sklearn.multiclass import OneVsRestClassifier

iris = load_iris()
X, y = iris.data, iris.target

y = label_binarize(y, classes=[0, 1, 2])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

n_estimators_values = [10, 20, 30]

auc_scores = []

for n_estimators in n_estimators_values:

    rf_classifier = OneVsRestClassifier(RandomForestClassifier(n_estimators=n_estimators,
random_state=42))
```

```
    rf_classifier.fit(X_train, y_train)
    y_pred_proba = rf_classifier.predict_proba(X_test)

    auc_scores.append(roc_auc_score(y_test, y_pred_proba, average='weighted', multi_class='ovr'))
    print(f"AUC Score for n_estimators = {n_estimators}: {auc_scores[-1]}")
plt.plot(n_estimators_values, auc_scores, marker='o')
plt.xlabel('n_estimators')
plt.ylabel('AUC Score')
plt.title('AUC Score vs. n_estimators for Random Forest (Iris Dataset)')
plt.grid(True)
plt.show()
```

AUC Score for n_estimators = 10: 1.0

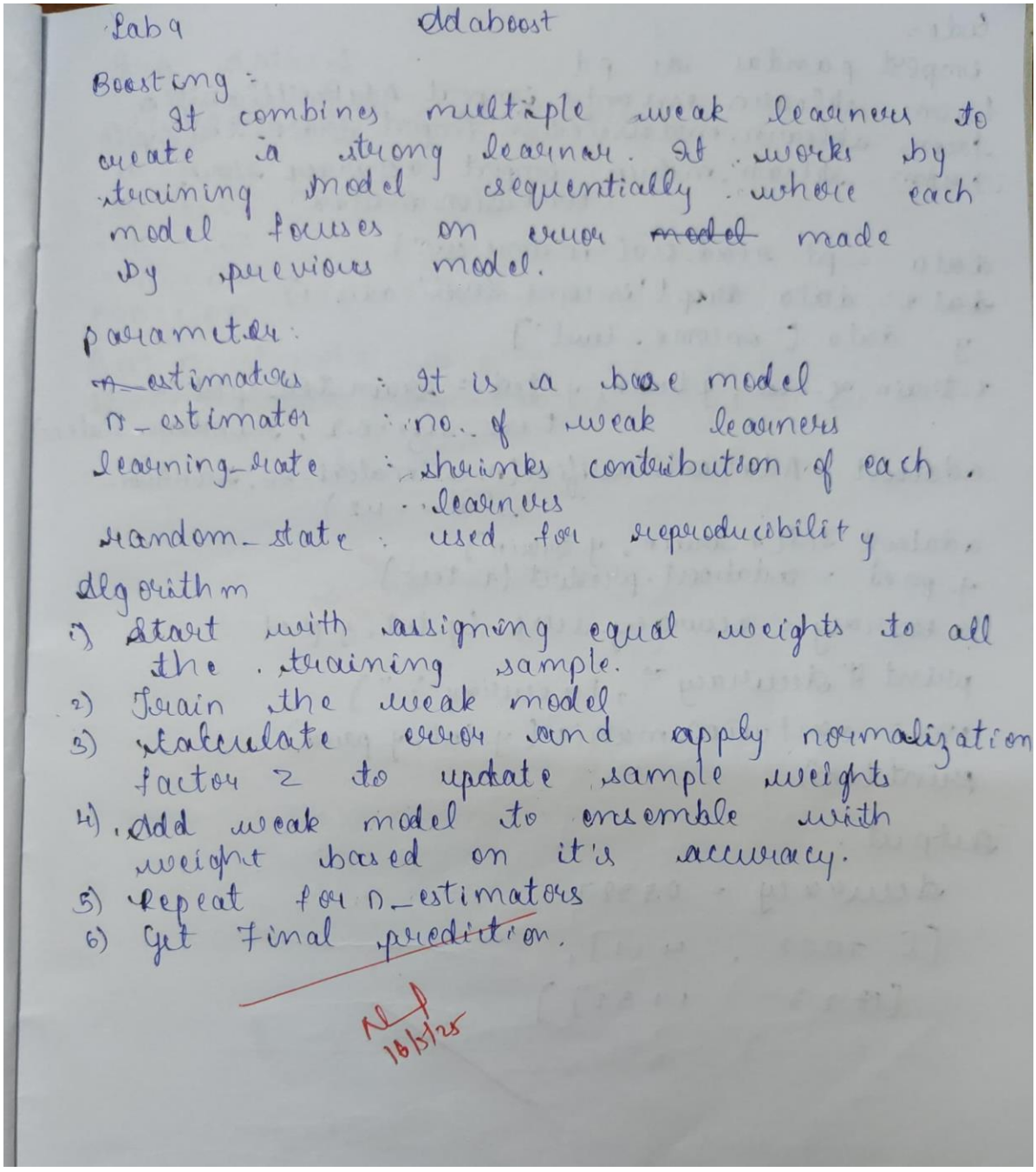
AUC Score for n_estimators = 20: 1.0

AUC Score for n_estimators = 30: 1.0



PROGRAM 9 Implement Boosting ensemble method on a given dataset.

Screenshot



Lab 9 adaboost

Boosting :
It combines multiple weak learners to create a strong learner. It works by training model sequentially where each model focuses on error ~~model~~ made by previous model.

parameters :
~~A~~ estimators : It is a base model
n_estimator : no. of weak learners
learning_rate : shrinks contribution of each learners
random_state : used for reproducibility

Algorithm
1) Start with assigning equal weights to all the training sample.
2) Train the weak model
3) Calculate error and apply normalization factor z to update sample weights
4) Add weak model to ensemble with weight based on it's accuracy.
5) Repeat for n_estimators
6) Get final prediction.

NL 16/5/25

Code :-

```
import pandas as pd
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,
                            confusion_matrix

data = pd.read_csv('income.csv')
x = data.drop('income_level', axis=1)
y = data['income_level']
x_train, x_test, y_train, y_test = train_test_split(x, y,
                    test_size=0.2, random_state=42)
adaboost = AdaBoostClassifier(n_estimators=50, random_
                    state=42)
adaboost.fit(x_train, y_train)
y_pred = adaboost.predict(x_test)
accuracy = accuracy_score(y_test, y_pred
print("Accuracy =", {accuracy})
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

Output :-

```
Accuracy = 0.8327
[[7003   411]
 [1423   1132]]
```

Iris dataset

accuracy with n estimators :-

10 : 1.00
50 : 1.00
100 : 1.00
200 : 1.00
500 : 1.00

Best n_estimator : 10
Best accuracy_score : 1.00

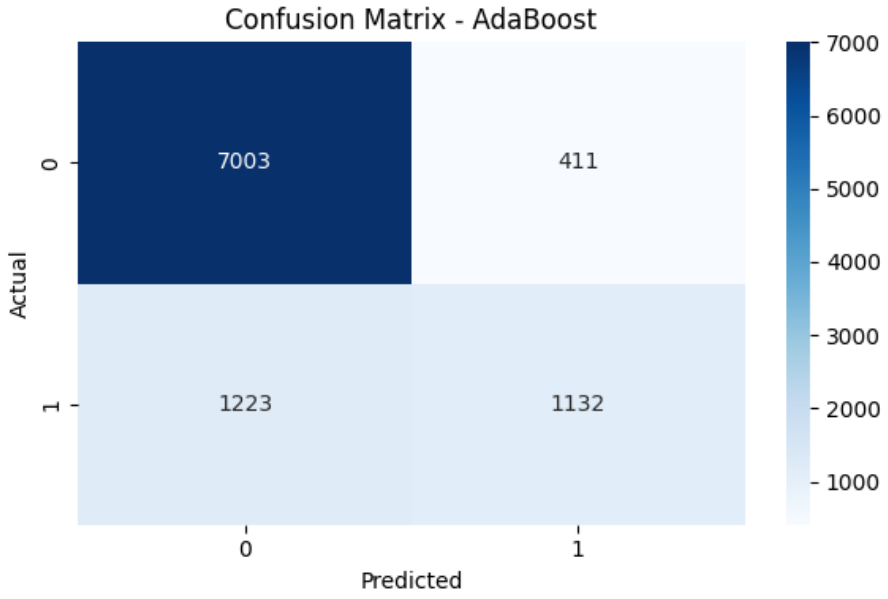
16/5/25

Code

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

# Load data
data = pd.read_csv('income.csv')

# Preprocess the data (placeholder)
# Example: data = pd.get_dummies(data)
# Make sure all features are numeric and handle any missing values.

# Split features and target
X = data.drop('income_level', axis=1)
y = data['income_level']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train AdaBoost classifier
ada_boost = AdaBoostClassifier(n_estimators=50, random_state=42)
ada_boost.fit(X_train, y_train)
```

```
# Predictions and evaluation
y_pred = ada_boost.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")

# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print(f"Confusion Matrix:\n{conf_matrix}")

# Plotting the confusion matrix
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
        xticklabels=ada_boost.classes_, yticklabels=ada_boost.classes_)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - AdaBoost')
plt.tight_layout()
plt.show()
```

```
Accuracy: 0.8327362063670796
Confusion Matrix:
[[7003  411]
 [1223 1132]]
```



In [4]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load Iris dataset
iris = load_iris()
X = iris.data
```

```
y = iris.target


# Split data into training (80%) and test (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# --- Try different base classifiers and parameters ---
results = []


# Parameters to experiment with
n_estimators_list = [10, 50, 100]
learning_rates = [0.01, 0.1, 1]


# DecisionTreeClassifier as base estimator
for n in n_estimators_list:
    for lr in learning_rates:
        tree_base = DecisionTreeClassifier(max_depth=1)
        model = AdaBoostClassifier(estimator=tree_base, n_estimators=n, learning_rate=lr,
random_state=42)
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        acc = accuracy_score(y_test, y_pred)
        results.append({
            'Base': 'DecisionTree',
            'n_estimators': n,
            'learning_rate': lr,
```

```
            'Accuracy': acc
        })

# LogisticRegression as base estimator
for n in n_estimators_list:
    for lr in learning_rates:
        log_reg_base = LogisticRegression(max_iter=1000)
        model = AdaBoostClassifier(estimator=log_reg_base, n_estimators=n, learning_rate=lr,
random_state=42)
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        acc = accuracy_score(y_test, y_pred)
        results.append({
            'Base': 'LogisticRegression',
            'n_estimators': n,
            'learning_rate': lr,
            'Accuracy': acc
        })

# Convert results to DataFrame for easy viewing
results_df = pd.DataFrame(results)
print(results_df)

# Optional: Visualize the result
import seaborn as sns
```

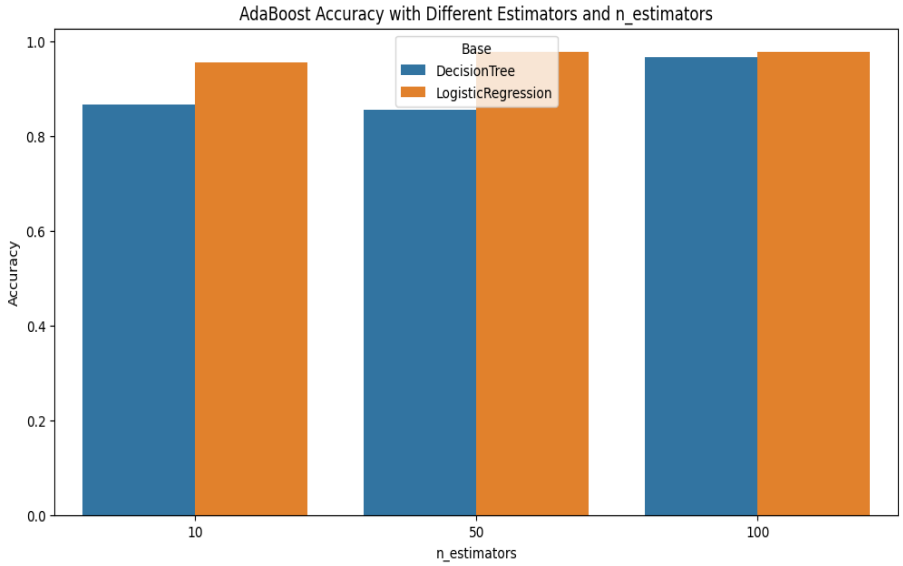
```
plt.figure(figsize=(12, 6))
sns.barplot(x='n_estimators', y='Accuracy', hue='Base', data=results_df, ci=None)
plt.title('AdaBoost Accuracy with Different Estimators and n_estimators')
plt.show()
```

	Base	n_estimators	learning_rate	Accuracy
0	DecisionTree	10	0.01	0.633333
1	DecisionTree	10	0.10	0.966667
2	DecisionTree	10	1.00	1.000000
3	DecisionTree	50	0.01	0.633333
4	DecisionTree	50	0.10	1.000000
5	DecisionTree	50	1.00	0.933333
6	DecisionTree	100	0.01	0.966667
7	DecisionTree	100	0.10	1.000000
8	DecisionTree	100	1.00	0.933333
9	LogisticRegression	10	0.01	0.933333
10	LogisticRegression	10	0.10	1.000000
11	LogisticRegression	10	1.00	0.933333
12	LogisticRegression	50	0.01	1.000000
13	LogisticRegression	50	0.10	1.000000
14	LogisticRegression	50	1.00	0.933333
15	LogisticRegression	100	0.01	1.000000
16	LogisticRegression	100	0.10	1.000000
17	LogisticRegression	100	1.00	0.933333

<ipython-input-4-897755f76214>:63: FutureWarning:

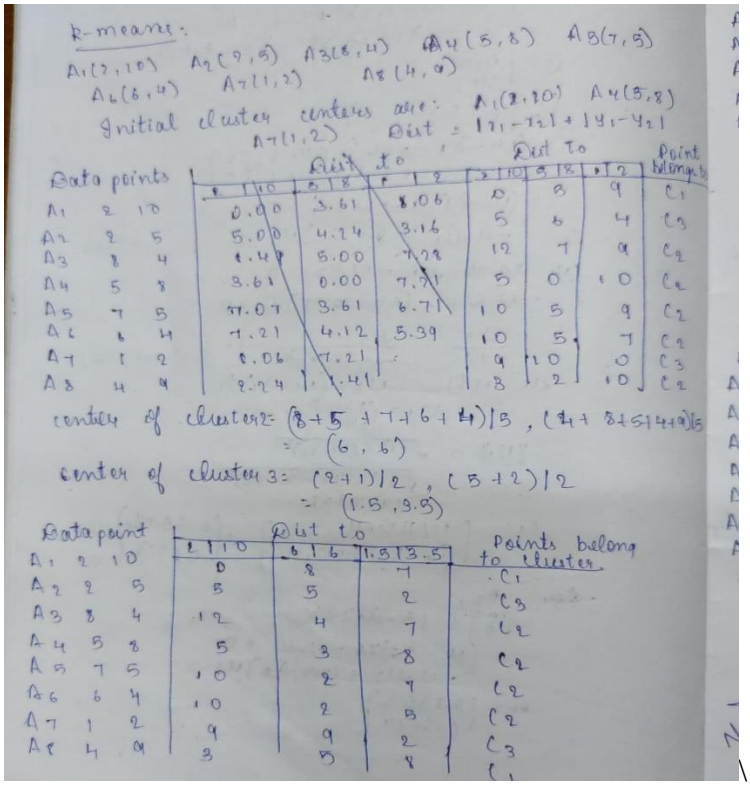
The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

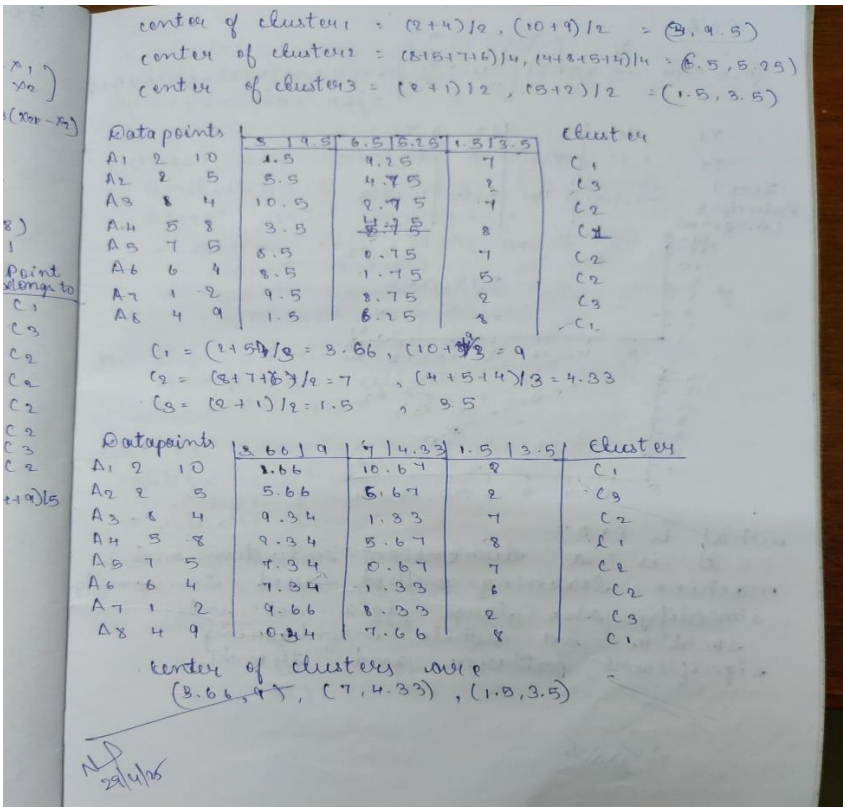
```
sns.barplot(x='n_estimators', y='Accuracy', hue='Base', data=results_df, ci=None)
```



PROGRAM 10 Build k-Means algorithm to cluster a set of data stored in a .CSV file.

Screenshot





k-means

1. choose the number of clusters K
2. Initialize k centroids randomly
3. Repeat until convergence
   - -> assign each data point to the nearest centroid
   - -> update centroids as the mean of points in each cluster
4. Stop when there is no longer change in collection of clusters or until max iteration.

How to determine no. of clusters?

- elbow method
  - -> ~~run~~ k-means
- -> silhotte score

Sum of square errors

$$SSE = \sum_{i=1}^{n} (d(x_i, u_i))^2$$

where i is the index of data point
x_i is the data point
u_i is the centroid.
d() is the distance function

SSE vs No. of clusters plot

- -> As no. of clusters increases, SSE generally decreases (more clusters allow for a closer fit of points)
- -> Decrease in SSE slows down after a certain point leading to elbow shape

Elbow Technique

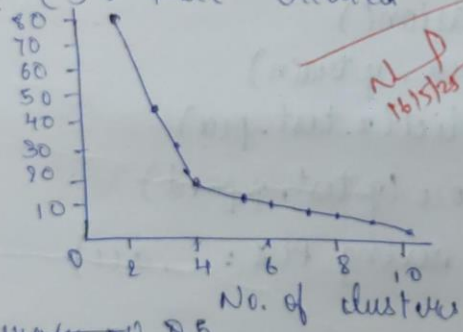
This method involves plotting of SSE against no. of clusters. The

Code:

```
import pandas as pd
import numpy as np
import random
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import adjusted_rand_score

names = ["Person {i}" for i in range(1, 51)]
ages = np.random.randint(20, 60, size=50)
incomes = np.random.randint(20000, 100000, size=50)
df = pd.DataFrame({
    'Name': names,
    'Age': ages,
    'Income': incomes
})
df.to_csv("income.csv", index=False)
data = read_csv("income.csv")
x = data[['Age', 'Income']]
x_train, x_test = train_test_split(x, test_size=0.2, random_state=
```

```
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)

sse = []
cluster_range = (1, 11)
for k in cluster_range:
    kmeans = KMeans(n_clusters=k, random_state=42,
                    n_init=10)
    kmeans.fit(x_train_scaled)
    sse.append(kmeans.inertia_)

kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
kmeans.fit(x_train_scaled)
y_train_pred = kmeans.predict(x_train_scaled)
y_test_pred = kmeans.predict(x_test_scaled)
score = adjusted_rand_score(y_train_pred[:len(y_test_pred)],
                            y_test_pred)
print("Adjusted Rand Index score (approx accuracy):",
      score)
```

output: CSV file created



~~accuracy = 0.85~~

~~Ex~~ Adjusted Rand Index score (proxy accuracy)
: 0.47

Code

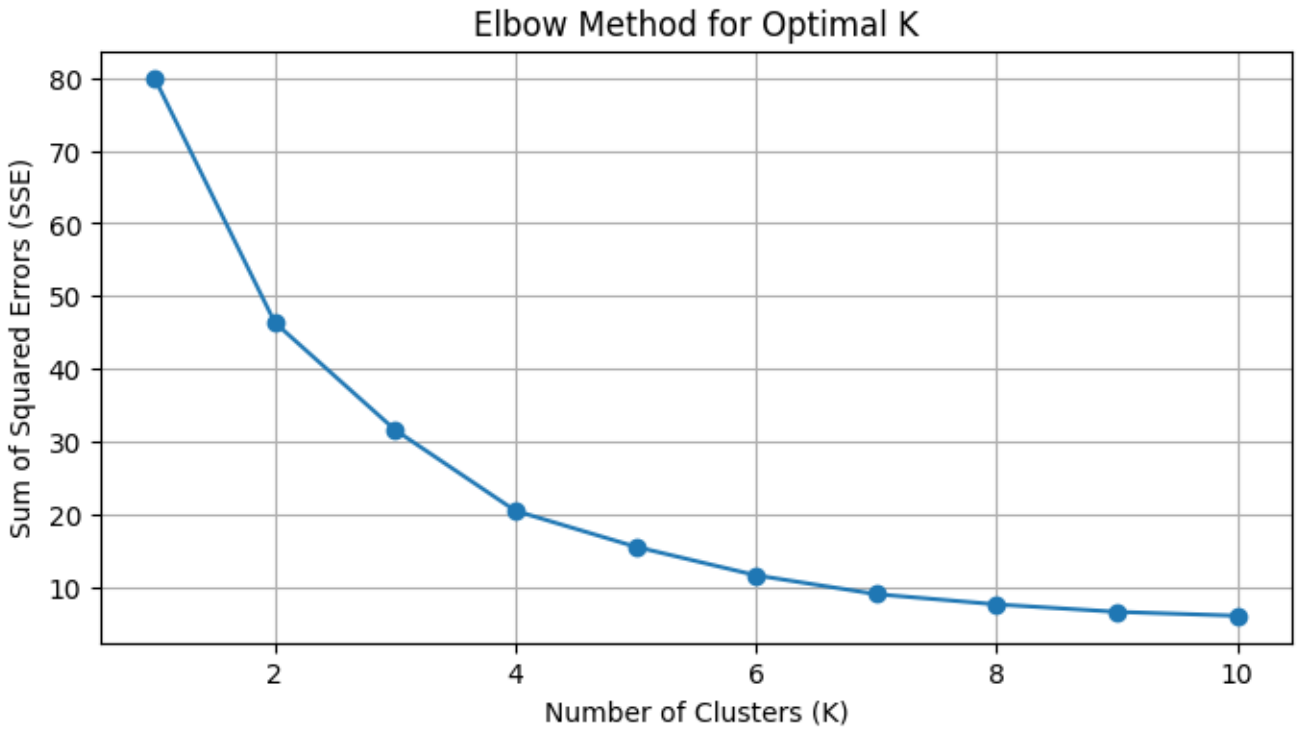
```
import pandas as pd
import numpy as np
import random
import string
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import adjusted_rand_score
import matplotlib.pyplot as plt

# Step 1: Generate data
def random_name():
    return ''.join(random.choices(string.ascii_uppercase, k=5))

np.random.seed(42)
data = {
    'Name': [random_name() for _ in range(50)],
    'Age': np.random.randint(20, 60, size=50),
    'Income': np.random.randint(30000, 100000, size=50)
}

df = pd.DataFrame(data)
df.to_csv("income.csv", index=False)
```

```
print("CSV file 'income.csv' created.")


# Step 2: Load and preprocess
df = pd.read_csv("income.csv")
X = df[['Age', 'Income']]  # Drop Name for modeling


# Step 3: Train-test split
X_train, X_test = train_test_split(X, test_size=0.2, random_state=42)


# Step 4: Scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)


# Step 5: Elbow Method - Plot SSE vs K
sse = []
K_range = range(1, 11)
for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init='auto')
    kmeans.fit(X_train_scaled)
    sse.append(kmeans.inertia_)


plt.figure(figsize=(8, 4))
plt.plot(K_range, sse, marker='o')
```

```
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Sum of Squared Errors (SSE)')
plt.title('Elbow Method for Optimal K')
plt.grid(True)
plt.show()

# Step 6: Fit and Predict using optimal K (e.g., choose K=3)
kmeans = KMeans(n_clusters=3, random_state=42, n_init='auto')
kmeans.fit(X_train_scaled)
train_labels = kmeans.labels_
test_labels = kmeans.predict(X_test_scaled)

# Since we don't have true labels, simulate labels for "accuracy" proxy
# We'll re-cluster the full dataset and use Adjusted Rand Index as proxy
kmeans_full = KMeans(n_clusters=3, random_state=42, n_init='auto')
X_scaled = scaler.fit_transform(X)
pred_full = kmeans_full.fit_predict(X_scaled)

# Evaluate clustering consistency (proxy to accuracy)
X_train_idx = X_train.index
X_test_idx = X_test.index

true_test_labels = pred_full[X_test_idx]
ari_score = adjusted_rand_score(true_test_labels, test_labels)
```

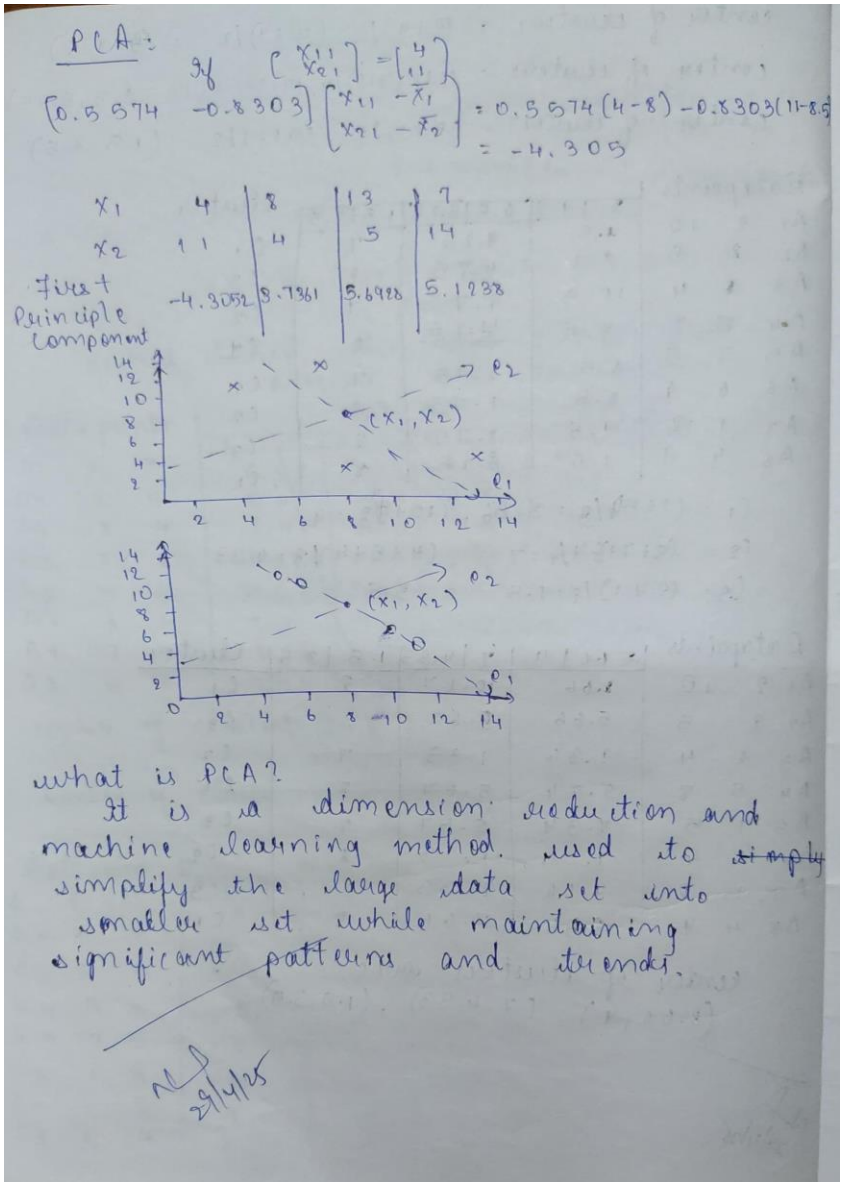
print(f"Adjusted Rand Index (proxy accuracy): {ari_score:.2f}")

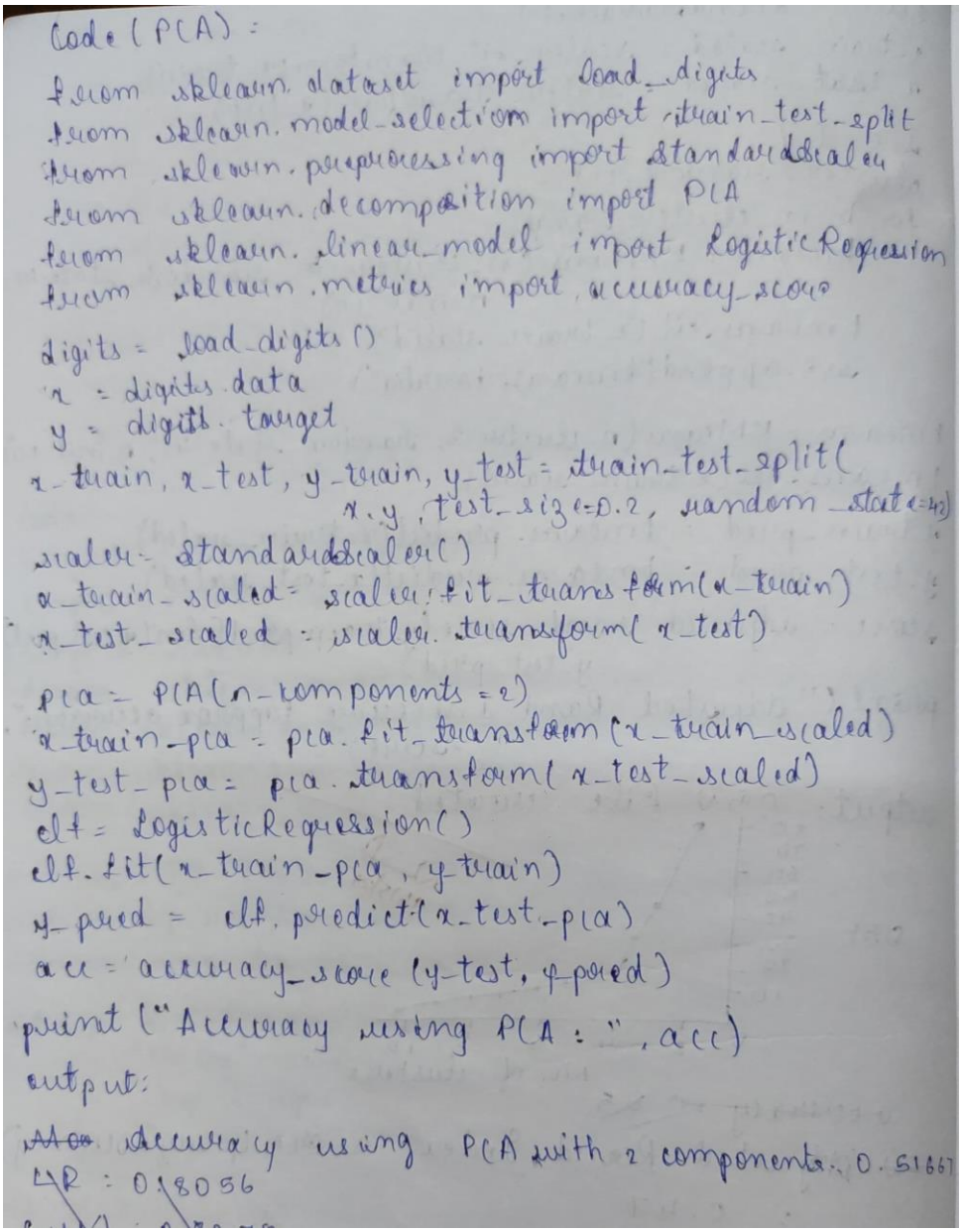
CSV file 'income.csv' created.



PROGRAM 11 Implement Dimensionality reduction using Principal Component Analysis (PCA) method.

Screenshot





Code

```
from sklearn.datasets import load_digits
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
digits = load_digits()
x = digits.data
y = digits.target
x_train, x_test, y_train, y_test = train_test_split(
x, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)
pca = PCA(n_components = 2)
x_train_pca = pca.fit_transform(x_train_scaled)
y_test_pca = pca.transform(x_test_scaled)
clf = LogisticRegression()
clf.fit(x_train_pca, y_train)
y_pred = clf.predict(x_test_pca)
acc = accuracy_score(y_test, y_pred)
print("Accuracy using PCA :", acc)
```

```
digits = load_digits()

X = digits.data

y = digits.target


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


scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)


pca = PCA(n_components=2)

X_train_pca = pca.fit_transform(X_train_scaled)

X_test_pca = pca.transform(X_test_scaled)


clf = LogisticRegression()

clf.fit(X_train_pca, y_train)


y_pred = clf.predict(X_test_pca)

accuracy = accuracy_score(y_test, y_pred)


print("Accuracy using PCA with 2 components:", accuracy)
```

Accuracy using PCA with 2 components: 0.5166666666666667

In [1]:

```
import numpy as np
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

def apply_pca(data, n_components=None):
    """
    Applies Principal Component Analysis (PCA) to the input data.

    Args:
        data (numpy.ndarray): The input data matrix (n_samples x n_features).
        n_components (int, optional): The number of principal components to keep.
                                      If None, all components are kept. Defaults to None.

    Returns:
        tuple: A tuple containing:
            - reduced_data (numpy.ndarray): The data projected onto the principal components.
            - pca (sklearn.decomposition.PCA): The fitted PCA object.
    """
    pca = PCA(n_components=n_components)
    reduced_data = pca.fit_transform(data)
    return reduced_data, pca
```

```
def visualize_pca(original_data, reduced_data, n_components):
    """
    Visualizes the original and reduced data (only for 2D or 3D original data).

    Args:
        original_data (numpy.ndarray): The original data matrix.
        reduced_data (numpy.ndarray): The reduced data matrix (should have 2 or 3 components).
        n_components (int): The number of principal components used.
    """
    original_n_features = original_data.shape[1]
    reduced_n_features = reduced_data.shape[1]

    if original_n_features == 2 and reduced_n_features == 2:
        plt.figure(figsize=(10, 5))
        plt.subplot(1, 2, 1)
        plt.scatter(original_data[:, 0], original_data[:, 1])
        plt.title("Original Data (2D)")
        plt.xlabel("Feature 1")
        plt.ylabel("Feature 2")

        plt.subplot(1, 2, 2)
        plt.scatter(reduced_data[:, 0], reduced_data[:, 1])
        plt.title(f"Reduced Data ({n_components} Components)")
```

```
    plt.xlabel("Principal Component 1")
    plt.ylabel("Principal Component 2")
    plt.tight_layout()
    plt.show()
elif original_n_features == 3 and reduced_n_features in [2, 3]:
    from mpl_toolkits.mplot3d import Axes3D
    fig = plt.figure(figsize=(12, 6))

    ax1 = fig.add_subplot(121, projection='3d')
    ax1.scatter(original_data[:, 0], original_data[:, 1], original_data[:, 2])
    ax1.set_title("Original Data (3D)")
    ax1.set_xlabel("Feature 1")
    ax1.set_ylabel("Feature 2")
    ax1.set_zlabel("Feature 3")

    ax2 = fig.add_subplot(122, projection='3d' if reduced_n_features == 3 else None)
    if reduced_n_features == 3:
        ax2.scatter(reduced_data[:, 0], reduced_data[:, 1], reduced_data[:, 2])
        ax2.set_zlabel("Principal Component 3")
    else:
        ax2.scatter(reduced_data[:, 0], reduced_data[:, 1])
    ax2.set_title(f"Reduced Data ({n_components} Components)")
    ax2.set_xlabel("Principal Component 1")
    ax2.set_ylabel("Principal Component 2")
```

```
        plt.tight_layout()
        plt.show()
    else:
        print(f"Visualization is only supported for original data with 2 or 3 features. Original data has {original_n_features} features.")

if __name__ == "__main__":
    # Generate some sample data
    np.random.seed(42)
    mean = [0, 0]
    cov = [[2, 1], [1, 2]]
    data = np.random.multivariate_normal(mean, cov, 100)

    # Apply PCA to reduce to 1 component
    n_components_to_keep = 1
    reduced_data_1, pca_1 = apply_pca(data, n_components=n_components_to_keep)
    print(f"Original data shape: {data.shape}")
    print(f"Reduced data shape (with {n_components_to_keep} component): {reduced_data_1.shape}")
    print("\nExplained variance ratio (1 component):", pca_1.explained_variance_ratio_)

    # Apply PCA to reduce to 2 components (which is the original dimension here)
    n_components_to_keep = 2
    reduced_data_2, pca_2 = apply_pca(data, n_components=n_components_to_keep)
    print(f"\nReduced data shape (with {n_components_to_keep} components): {reduced_data_2.shape}")
```

```
print("Explained variance ratio (2 components):", pca_2.explained_variance_ratio_)

# Visualize the results for 2D original data
visualize_pca(data, reduced_data_1, n_components=1)  # Changed to 'n_components'
visualize_pca(data, reduced_data_2, n_components=2)  # Changed to 'n_components'

# Example with 3D data
np.random.seed(123)
mean_3d = [1, 2, 3]
cov_3d = [[1, 0.5, 0.2], [0.5, 1.5, 0.1], [0.2, 0.1, 1]]
data_3d = np.random.multivariate_normal(mean_3d, cov_3d, 100)

reduced_data_3d_2, pca_3d_2 = apply_pca(data_3d, n_components=2)
print(f"\nOriginal 3D data shape: {data_3d.shape}")
print(f"Reduced 3D data shape (to 2 components): {reduced_data_3d_2.shape}")
print("Explained variance ratio (2 components for 3D data):", pca_3d_2.explained_variance_ratio_)
visualize_pca(data_3d, reduced_data_3d_2, n_components=2) # Changed to 'n_components'

reduced_data_3d_3, pca_3d_3 = apply_pca(data_3d, n_components=3)
print(f"\nReduced 3D data shape (to 3 components): {reduced_data_3d_3.shape}")
print("Explained variance ratio (3 components for 3D data):", pca_3d_3.explained_variance_ratio_)
visualize_pca(data_3d, reduced_data_3d_3, n_components=3)
```

```
Original data shape: (100, 2)
```

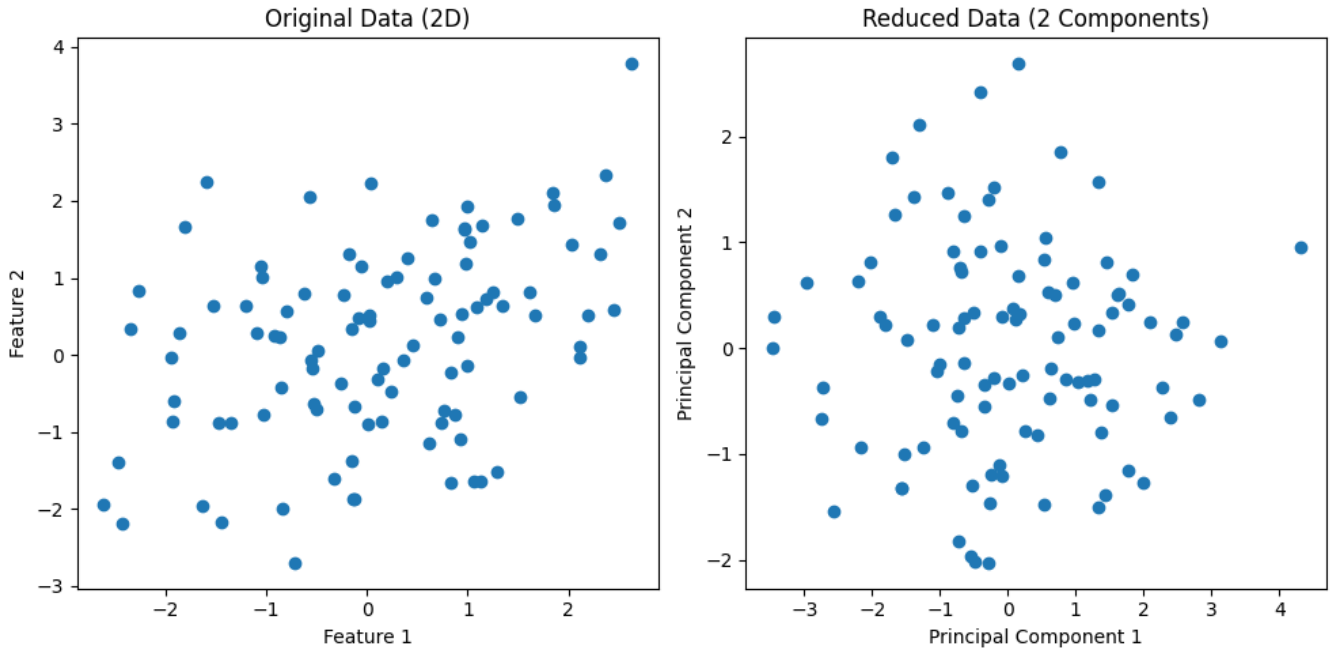
Reduced data shape (with 1 component): (100, 1)

Explained variance ratio (1 component): [0.68856826]

Reduced data shape (with 2 components): (100, 2)

Explained variance ratio (2 components): [0.68856826 0.31143174]

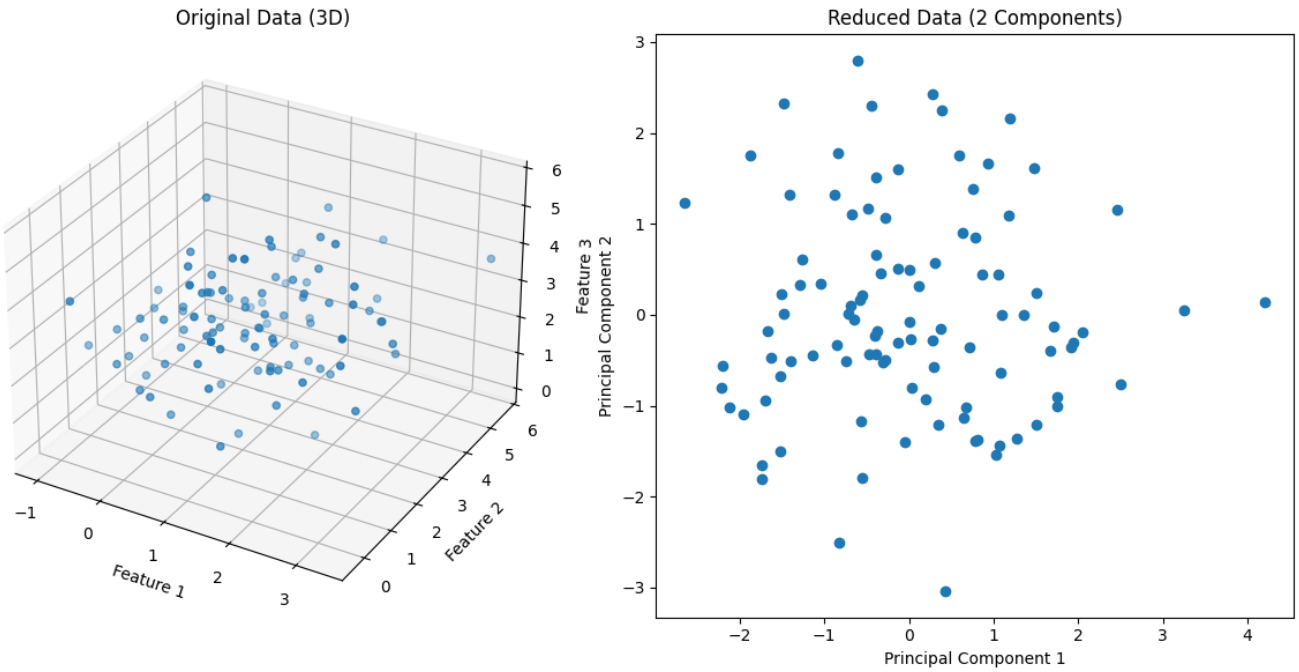
Visualization is only supported for original data with 2 or 3 features. Original data has 2 features.



Original 3D data shape: (100, 3)

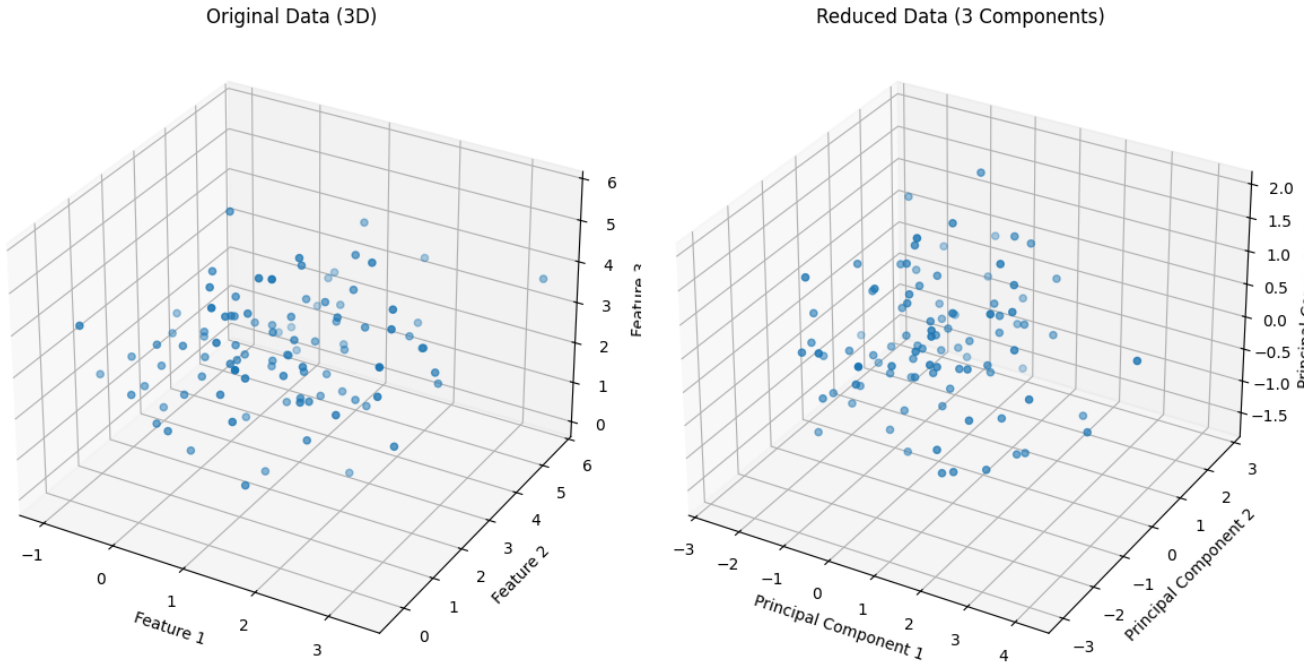
Reduced 3D data shape (to 2 components): (100, 2)

Explained variance ratio (2 components for 3D data): [0.46031414 0.36523668]



Reduced 3D data shape (to 3 components): (100, 3)

Explained variance ratio (3 components for 3D data): [0.46031414 0.36523668 0.17444918]



In [2]:

```
import numpy as np
from sklearn.decomposition import PCA

def reduce_dimensions(data, n_components):
    """Applies PCA to reduce data dimensionality.

    Args:
        data (numpy.ndarray): Input data (n_samples x n_features).
        n_components (int): Number of principal components to keep.

    Returns:
        numpy.ndarray: Reduced data (n_samples x n_components).
    """
    pca = PCA(n_components=n_components)
    reduced_data = pca.fit_transform(data)
    return reduced_data

if __name__ == "__main__":
    # Example usage
    np.random.seed(42)
    data = np.random.rand(100, 5)  # 100 samples, 5 features
    n_components_to_keep = 2
    reduced_data = reduce_dimensions(data, n_components_to_keep)
    print("Original data shape:", data.shape)
```

```
    print("Reduced data shape:", reduced_data.shape)
```

```
Original data shape: (100, 5)
Reduced data shape: (100, 2)
```

In [3]:

```
import numpy as np
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# 1. Download Load_digits from sklearn dataset
digits = load_digits()
X = digits.data
y = digits.target

# 2. Split data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 3. Perform scaling
```

```
scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)


# 4. Build a PCA model with no_of_components=2

n_components = 2

pca = PCA(n_components=n_components)

X_train_pca = pca.fit_transform(X_train_scaled)

X_test_pca = pca.transform(X_test_scaled)


# 5. Predict score using logistic regression on PCA-transformed data

logistic_regression = LogisticRegression(random_state=42, solver='liblinear', multi_class='ovr')

logistic_regression.fit(X_train_pca, y_train)

y_pred_pca = logistic_regression.predict(X_test_pca)


# 6. Predict accuracy score using PCA

accuracy_pca = accuracy_score(y_test, y_pred_pca)

print(f"Accuracy score with {n_components} principal components: {accuracy_pca:.4f}")


# Optional: Predict score using logistic regression on original scaled data for comparison

logistic_regression_original = LogisticRegression(random_state=42, solver='liblinear',
multi_class='ovr')

logistic_regression_original.fit(X_train_scaled, y_train)

y_pred_original = logistic_regression_original.predict(X_test_scaled)

accuracy_original = accuracy_score(y_test, y_pred_original)
```

```
print(f"Accuracy score with original scaled data: {accuracy_original:.4f}")
```

```
Accuracy score with 2 principal components: 0.5083
Accuracy score with original scaled data: 0.9667
```

```
import pandas as pd
import numpy as np
from scipy.stats import zscore
from sklearn.preprocessing import StandardScaler, OneHotEncoder, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.decomposition import PCA
import kagglehub
import os

# --- Download dataset using kagglehub ---
try:
    path = kagglehub.dataset_download("fedesoriano/heart-failure-prediction", path=".")
    print("Path to downloaded dataset files:", path)

    # Assuming the CSV file is directly in the downloaded folder or a subfolder
```

```
    # Let's try to find the 'heart.csv' file
    heart_csv_path = ""
    for root, _, files in os.walk(path):
        if 'heart.csv' in files:
            heart_csv_path = os.path.join(root, 'heart.csv')
            break

    if not heart_csv_path:
        raise FileNotFoundError("heart.csv not found in the downloaded dataset.")

    # Load the dataset
    df = pd.read_csv(heart_csv_path)

except ImportError:
    print("Error: kagglehub library not found. Please install it using: pip install kagglehub")
    exit()
except FileNotFoundError as e:
    print(f"Error: {e}")
    print("Please ensure you are authenticated with Kaggle and the dataset exists.")
    exit()
except Exception as e:
    print(f"An error occurred during download: {e}")
    exit()
```

```
# Display initial info
print("\nFirst few rows of the dataset:")
print(df.head())
print("\nDataset information:")
df.info()

# --- 2. Remove Outliers using Z-score ---
numerical_cols = df.select_dtypes(include=np.number).columns
z_scores = df[numerical_cols].apply(zscore)
df_filtered = df[(z_scores > -3).all(axis=1) & (z_scores < 3).all(axis=1)]
print(f"\nOriginal DataFrame shape: {df.shape}")
print(f"DataFrame shape after outlier removal: {df_filtered.shape}")

# --- 3. Convert Text Columns to Numbers ---
categorical_cols = df_filtered.select_dtypes(include='object').columns
binary_cols = []
multi_category_cols = []
for col in categorical_cols:
    if df_filtered[col].nunique() == 2:
        binary_cols.append(col)
    else:
        multi_category_cols.append(col)

label_encoders = {}
```

```
for col in binary_cols:
    label_encoders[col] = LabelEncoder()
    df_filtered[col] = label_encoders[col].fit_transform(df_filtered[col])

one_hot_encoder = OneHotEncoder(handle_unknown='ignore', sparse_output=False)
encoded_cols = one_hot_encoder.fit_transform(df_filtered[multi_category_cols])
encoded_df = pd.DataFrame(encoded_cols,
columns=one_hot_encoder.get_feature_names_out(multi_category_cols))

df_filtered = df_filtered.reset_index(drop=True)
encoded_df = encoded_df.reset_index(drop=True)
df_processed = pd.concat([df_filtered.drop(multi_category_cols, axis=1), encoded_df], axis=1)

print("\nDataFrame after encoding categorical features:")
print(df_processed.head())

# --- 4. Apply Scaling ---
numerical_cols_scaled = df_processed.select_dtypes(include=np.number).drop('HeartDisease',
axis=1).columns
scaler = StandardScaler()
df_processed[numerical_cols_scaled] = scaler.fit_transform(df_processed[numerical_cols_scaled])

print("\nDataFrame after scaling numerical features:")
print(df_processed.head())
```

```
# --- 5. Build and Evaluate Classification Models (Without PCA) ---
X = df_processed.drop('HeartDisease', axis=1)
y = df_processed['HeartDisease']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

svm_model = SVC(random_state=42)
logistic_model = LogisticRegression(random_state=42, solver='liblinear')
rf_model = RandomForestClassifier(random_state=42)

models = {'SVM': svm_model, 'Logistic Regression': logistic_model, 'Random Forest': rf_model}
baseline_accuracies = {}

print("\n--- Model Performance without PCA ---")
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    baseline_accuracies[name] = accuracy
    print(f"{name} Accuracy: {accuracy:.4f}")

# --- 6. Now use PCA to reduce dimensions, retrain your model and see what impact it has ---
pca = PCA(n_components=0.95) # Keep components that explain 95% of the variance
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)
```

```
print(f"\nNumber of components before PCA: {X_train.shape[1]}")
print(f"Number of components after PCA: {X_train_pca.shape[1]}")

pca_accuracies = {}

print("\n--- Model Performance with PCA ---")
for name, model in models.items():
    model.fit(X_train_pca, y_train)
    y_pred_pca = model.predict(X_test_pca)
    accuracy_pca = accuracy_score(y_test, y_pred_pca)
    pca_accuracies[name] = accuracy_pca
    print(f"{name} Accuracy (with PCA): {accuracy_pca:.4f}")

# --- Comparison of Results ---
print("\n--- Comparison of Accuracies ---")
for name in models.keys():
    print(f"{name}:")
    print(f"  Without PCA: {baseline_accuracies[name]:.4f}")
    print(f"  With PCA:    {pca_accuracies[name]:.4f}")
```

```
Path to downloaded dataset files: /kaggle/input/heart-failure-prediction/.

First few rows of the dataset:
```

```
   Age Sex ChestPainType  RestingBP  Cholesterol  FastingBS RestingECG  MaxHR  \
0   40   M           ATA        140          289          0     Normal    172
1   49   F           NAP        160          180          0     Normal    156
2   37   M           ATA        130          283          0         ST     98
3   48   F           ASY        138          214          0     Normal    108
4   54   M           NAP        150          195          0     Normal    122

  ExerciseAngina  Oldpeak ST_Slope  HeartDisease
0              N      0.0       Up             0
1              N      1.0     Flat             1
2              N      0.0       Up             0
3              Y      1.5     Flat             1
4              N      0.0       Up             0

Dataset information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 918 entries, 0 to 917
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Age             918 non-null    int64
 1   Sex             918 non-null    object
 2   ChestPainType   918 non-null    object
 3   RestingBP       918 non-null    int64
```

```
 4   Cholesterol     918 non-null    int64
 5   FastingBS       918 non-null    int64
 6   RestingECG      918 non-null    object
 7   MaxHR           918 non-null    int64
 8   ExerciseAngina  918 non-null    object
 9   Oldpeak         918 non-null    float64
 10  ST_Slope        918 non-null    object
 11  HeartDisease    918 non-null    int64
dtypes: float64(1), int64(6), object(5)
memory usage: 86.2+ KB

Original DataFrame shape: (918, 12)
DataFrame shape after outlier removal: (899, 12)

DataFrame after encoding categorical features:
   Age  Sex  RestingBP  Cholesterol  FastingBS  MaxHR  ExerciseAngina  \
0   40    1        140          289          0    172               0
1   49    0        160          180          0    156               0
2   37    1        130          283          0     98               0
3   48    0        138          214          0    108               1
4   54    1        150          195          0    122               0

   Oldpeak  HeartDisease  ChestPainType_ASY  ChestPainType_ATA  \
0      0.0             0                0.0                1.0
```

1	1.0	1	0.0	0.0
2	0.0	0	0.0	1.0
3	1.5	1	1.0	0.0
4	0.0	0	0.0	0.0

	ChestPainType_NAP	ChestPainType_TA	RestingECG_LVH	RestingECG_Normal \
0	0.0	0.0	0.0	1.0
1	1.0	0.0	0.0	1.0
2	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	1.0
4	1.0	0.0	0.0	1.0

	RestingECG_ST	ST_Slope_Down	ST_Slope_Flat	ST_Slope_Up
0	0.0	0.0	0.0	1.0
1	0.0	0.0	1.0	0.0
2	1.0	0.0	0.0	1.0
3	0.0	0.0	1.0	0.0
4	0.0	0.0	0.0	1.0

	Age	Sex	RestingBP	Cholesterol	FastingBS	MaxHR \
0	-1.428154	0.515943	0.465900	0.849636	-0.550362	1.384320
1	-0.475855	-1.938199	1.634714	-0.168122	-0.550362	0.752973
2	-1.745588	0.515943	-0.118507	0.793612	-0.550362	-1.535661
3	-0.581666	-1.938199	0.349019	0.149344	-0.550362	-1.141069

4 0.053200 0.515943 1.050307 -0.028064 -0.550362 -0.588640

	ExerciseAngina	Oldpeak	HeartDisease	ChestPainType_ASY
0	-0.822945	-0.855469	0	-1.077524
1	-0.822945	0.137516	1	-1.077524
2	-0.822945	-0.855469	0	-1.077524
3	1.215148	0.634008	1	0.928054
4	-0.822945	-0.855469	0	-1.077524

	ChestPainType_ATA	ChestPainType_NAP	ChestPainType_TA	RestingECG_LVH
0	2.063325	-0.534905	-0.22955	-0.503821
1	-0.484655	1.869492	-0.22955	-0.503821
2	2.063325	-0.534905	-0.22955	-0.503821
3	-0.484655	-0.534905	-0.22955	-0.503821
4	-0.484655	1.869492	-0.22955	-0.503821

	RestingECG_Normal	RestingECG_ST	ST_Slope_Down	ST_Slope_Flat	ST_Slope_Up
0	0.809702	-0.489898	-0.260184	-0.998888	1.134695
1	0.809702	-0.489898	-0.260184	1.001113	-0.881294
2	-1.235023	2.041241	-0.260184	-0.998888	1.134695
3	0.809702	-0.489898	-0.260184	1.001113	-0.881294
4	0.809702	-0.489898	-0.260184	-0.998888	1.134695

--- Model Performance without PCA ---

SVM Accuracy: 0.8444

Logistic Regression Accuracy: 0.8481

Random Forest Accuracy: 0.8778

Number of components before PCA: 18

Number of components after PCA: 13

--- Model Performance with PCA ---

SVM Accuracy (with PCA): 0.8370

Logistic Regression Accuracy (with PCA): 0.8593

Random Forest Accuracy (with PCA): 0.8259

--- Comparison of Accuracies ---

SVM:

Without PCA: 0.8444

With PCA: 0.8370

Logistic Regression:

Without PCA: 0.8481

With PCA: 0.8593

Random Forest:

Without PCA: 0.8778

With PCA: 0.8259

In [5]:

```
import numpy as np
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# Given data from the table
data = np.array([[4, 11],
                 [8, 4],
                 [13, 5],
                 [7, 14]])

# 1. Center the data by subtracting the mean of each feature
mean = np.mean(data, axis=0)
centered_data = data - mean
print("Centered Data:\n", centered_data)

# 2. Calculate the covariance matrix
covariance_matrix = np.cov(centered_data, rowvar=False)
print("\nCovariance Matrix:\n", covariance_matrix)

# 3. Calculate the eigenvalues and eigenvectors of the covariance matrix
eigenvalues, eigenvectors = np.linalg.eig(covariance_matrix)
print("\nEigenvalues:\n", eigenvalues)
print("\nEigenvectors:\n", eigenvectors)
```

```
# 4. Sort the eigenvalues and eigenvectors in descending order of eigenvalues

sorted_indices = np.argsort(eigenvalues)[::-1]

sorted_eigenvalues = eigenvalues[sorted_indices]

sorted_eigenvectors = eigenvectors[:, sorted_indices]

print("\nSorted Eigenvalues:\n", sorted_eigenvalues)

print("\nSorted Eigenvectors:\n", sorted_eigenvectors)


# 5. Select the principal component(s). We want to reduce to 1 dimension, so we take the eigenvector corresponding to the largest eigenvalue.

principal_component = sorted_eigenvectors[:, 0]

print("\nPrincipal Component (Eigenvector with largest eigenvalue):\n", principal_component)


# 6. Project the original (centered) data onto the principal component

reduced_data = np.dot(centered_data, principal_component)

print("\nReduced Data (1 dimension):\n", reduced_data)


# --- Using scikit-learn's PCA for verification ---

print("\n--- Using scikit-learn's PCA for Verification ---")

pca = PCA(n_components=1)

pca.fit(data)

sklearn_reduced_data = pca.transform(data)

print("Scikit-learn Principal Component:\n", pca.components_)

print("Scikit-learn Reduced Data (1 dimension):\n", sklearn_reduced_data)
```

Centered Data:

[[-4. 2.5]

[0. -4.5]

[5. -3.5]

[-1. 5.5]]

Covariance Matrix:

[[14. -11.]

[-11. 23.]]

Eigenvalues:

[6.61513568 30.38486432]

Eigenvectors:

[[-0.83025082 0.55738997]

[-0.55738997 -0.83025082]]

Sorted Eigenvalues:

[30.38486432 6.61513568]

Sorted Eigenvectors:

[[0.55738997 -0.83025082]

[-0.83025082 -0.55738997]]

Principal Component (Eigenvector with largest eigenvalue):

[0.55738997 -0.83025082]

Reduced Data (1 dimension):

[-4.30518692 3.73612869 5.69282771 -5.12376947]

--- Using scikit-learn's PCA for Verification ---

Scikit-learn Principal Component:

[[-0.55738997 0.83025082]]

Scikit-learn Reduced Data (1 dimension):

[[4.30518692]

[-3.73612869]

[-5.69282771]

[5.12376947]]

In [ ]: