VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

Sanjana Suresh (1BM22CS239)

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
Feb-2025 to June-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 **Sanjana Suresh(1BM22CS239),** who is a 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.

Lab Faculty Incharge Name: Ms. Saritha A N Assistant Professor Department of CSE, BMSCE	Dr. Kavitha Sooda Professor & HOD Department of CSE, BMSCE

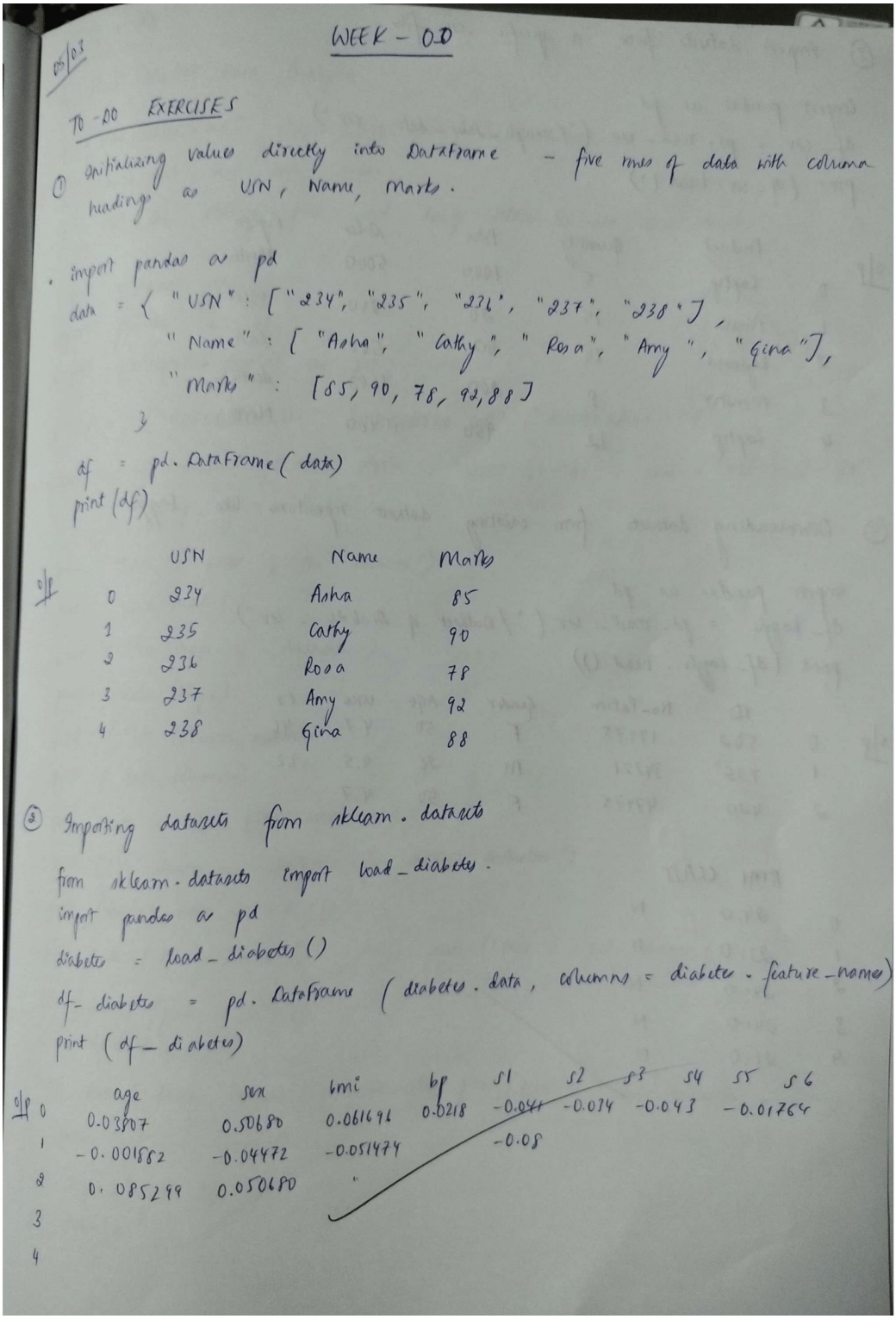
Index

Github Link: https://github.com/SanjanaSuresh30/ML_LAB_1BM22CS239

Program 1

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

Screenshot:



WEEK - 00

TO-DO EXERCISES

① Initializing values directly into DataFrame - five rows of data with column headings as USN, Name, Marks.

```
import pandas as pd
data = { "USN" : ["234", "235", "236", "237", "238"],
         "Name" : ["Asha", "Cathy", "Rosa", "Amy", "Gina"],
         "Marks" : [85, 90, 78, 92, 88]
       }
df = pd.DataFrame(data)
print(df)
```

o/p

	USN	Name	Marks
0	234	Asha	85
1	235	Cathy	90
2	236	Rosa	78
3	237	Amy	92
4	238	Gina	88

② Importing datasets from sklearn.datasets

```
from sklearn.datasets import load_diabetes
import pandas as pd
diabetes = load_diabetes()
df_diabetes = pd.DataFrame(diabetes.data, columns = diabetes.feature_names)
print(df_diabetes)
```

o/p

	age	sex	bmi	bp	s1	s2	s3	s4	s5	s6
0	0.03807	0.50680	0.061696	0.0218	-0.044	-0.034	-0.043	-0.01764		
1	-0.001882	-0.04472	-0.051474		-0.08					
2	0.085299	0.050680	"							
3										
4										

③ Import datasets from a specific .csv file

```
import pandas as pd
df_csv = pd.read_csv("/sample_sales_data.csv")
print(df_csv.head())
```

o/p

	Product	Quantity	Price	Sales	Region
0	Laptop	5	1000	5000	North
1	Mouse	15	20	300	West
2	Keyboard	10	50	500	East
3	Monitor	8	200	1600	South
4	Laptop	12	950	11400	North

④ Downloading datasets from existing dataset repository like Kaggle..

```
import pandas as pd
df_kaggle = pd.read_csv("/Dataset of Diabetes.csv")
print(df_kaggle.head())
```

o/p

	ID	No_Patient	Gender	AGE	Urea	Cr	BMI	CLASS
0	502	17975	F	50	4.7	46	24.0	N
1	735	34221	M	26	4.5	62	23.0	N
2	420	47975	F	50	4.7		24.0	N
3							24.0	N
4							21.0	N

TO - DO

Stock Market Data Analysis

1. ["HDFCBANK.NS", "ICICIBANK.NS", "KOTAKBANK.NS"]
2. start date and end date.
3. Plot the closing price and daily return for all three banks.

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

tickers = ["HDFCBANK.NS", "ICICIBANK.NS", "KOTAKBANK.NS"]
data = yf.download(tickers, start = "2024-01-01", end = "2024-12-30",
    group_by = 'ticker')
print("First 5 rows of the dataset")
print(data.head())
print("\n Shape of dataset")
print(data.shape)
print("\n Column names")
print(data.columns)
hdfc_data = data['HDFCBANK.NS']
print("\n Summary statistics for HDFC Industries")
print(hdfc_data.describe())
hdfc_data['Daily Return'] = hdfc_data['Close'].pct_change()
plt.figure(figsize = (12, 6))
plt.subplot(2, 1, 1)
hdfc_data['Daily_Return'] = hdfc_data.plot(title = "HDFC Industries - Daily
  Returns", color = orange)
plt.tight_layout()
plt.show()
```

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())

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())

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())

file_path = 'mobiles-dataset-2025.csv'
df = pd.read_csv(file_path, encoding='latin-1') # or 'cp1252' or other suitable encoding
print("Sample data:")
print(df.head())

import pandas as pd
data = {
   'USN': ['IS001','IS002','IS003','IS004','IS005'],
```

```
    'Name': ['Alice', 'Bob', 'Charlie', 'David','Eve'],
    'Marks': [25, 30, 35, 40,45]
}
df = pd.DataFrame(data)
print("Sample data:")
print(df.head())

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

df = pd.read_csv("/content/dataset-of-diabetes .csv",encoding='latin-1')
print("Sample data:")
print(df.head())
print("\n")

df =pd.read_csv('sample_sales_data.csv')
print("Sample data:")
print(df.head())
df.to_csv('output.csv',index=False)
print("Data saved to output.csv")

sales_df =pd.read_csv('sample_sales_data.csv')
print("Sample data:")
print(sales_df.head())
sales_by_region =sales_df.groupby('Region')['Sales'].sum()
print("\nTotal sales by region:")
print(sales_by_region)
best_selling_products
=sales_df.groupby('Product')['Quantity'].sum().sort_values(ascending=False)
print("\nBest-selling products by quantity:")
print(best_selling_products)
```

```
sales_by_region.to_csv('sales_by_region.csv')
best_selling_products.to_csv('best_selling_products.csv')
print("Data saved to sales_by_region.csv and best_selling_products.csv")

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')
print("First 5 rows of the dataset:")
print(data.head())
print("\nShape of the dataset:")
print(data.shape)
print("\nColumn names:")
print(data.columns)
print("\n")
reliance_data = data['RELIANCE.NS']
print("\nSummary statistics for Reliance Industries:")
print(reliance_data.describe())
reliance_data['Daily Return'] = reliance_data['Close'].pct_change()
print("\n")
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
reliance_data['Close'].plot(title="Reliance Industries - Closing Price")
plt.subplot(2, 1, 2)
reliance_data['Daily Return'].plot(title="Reliance Industries - Daily Returns", color='orange')
plt.tight_layout()
plt.show()
reliance_data.to_csv('reliance_stock_data.csv')

tickers = ["HDFCBANK.NS", "ICICI.NS", "KOTAKBANK.NS"]
data = yf.download(tickers, start="2024-01-01", end="2024-12-30",
                   group_by='ticker')
print("First 5 rows of the dataset:")
```

```
print(data.head())
print("\nShape of the dataset:")
print(data.shape)
print("\nColumn names:")
print(data.columns)
print("\n")
reliance_data = data['HDFCBANK.NS']
print("\nSummary statistics for Reliance Industries:")
print(reliance_data.describe())
reliance_data['Daily Return'] = reliance_data['Close'].pct_change()
print("\n")
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
reliance_data['Close'].plot(title="HDFC Industries - Closing Price")
plt.subplot(2, 1, 2)
reliance_data['Daily Return'].plot(title="HDFCIndustries - Daily Returns", color='red')
plt.tight_layout()
plt.show()
reliance_data.to_csv('hdfc_stock_data.csv')
print("\nhdfc stock data saved to 'hdfc_stock_data.csv'.")
```

Method1-Initializing values directly into DataFrame

```
[ ]  import pandas as pd
     # Create a DataFrame directly from a dictionary
     data = {
         'Name': ['Alice', 'Bob', 'Charlie', 'David','Dexter'],
         'USN': ['1BM22CS260', '1BM22CS261', '1BM22CS262', '1BM22CS263','1BM22CS264'],
         'MARKS': [25, 30, 35, 40,50]

     }
     df = pd.DataFrame(data)
     print("Sample data:")
     print(df.head())
```

Method-2: Importing datasets 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:")
     print(df.head())
```

Method-3: Importing datasets from a specific .csv file

```
[ ]  # Load data from a CSV file (replace 'data.csv' with your file path)
     file_path = '/content/drive/MyDrive/SEM6/ML/ML_LAB/ML_LAB0/ml_Lab-0/Lab-0/data.csv'  # Ensure the file exists in the same directory
     df = pd.read_csv(file_path)
     print("Sample data:")
     print(df.head())
     print("\n")
```

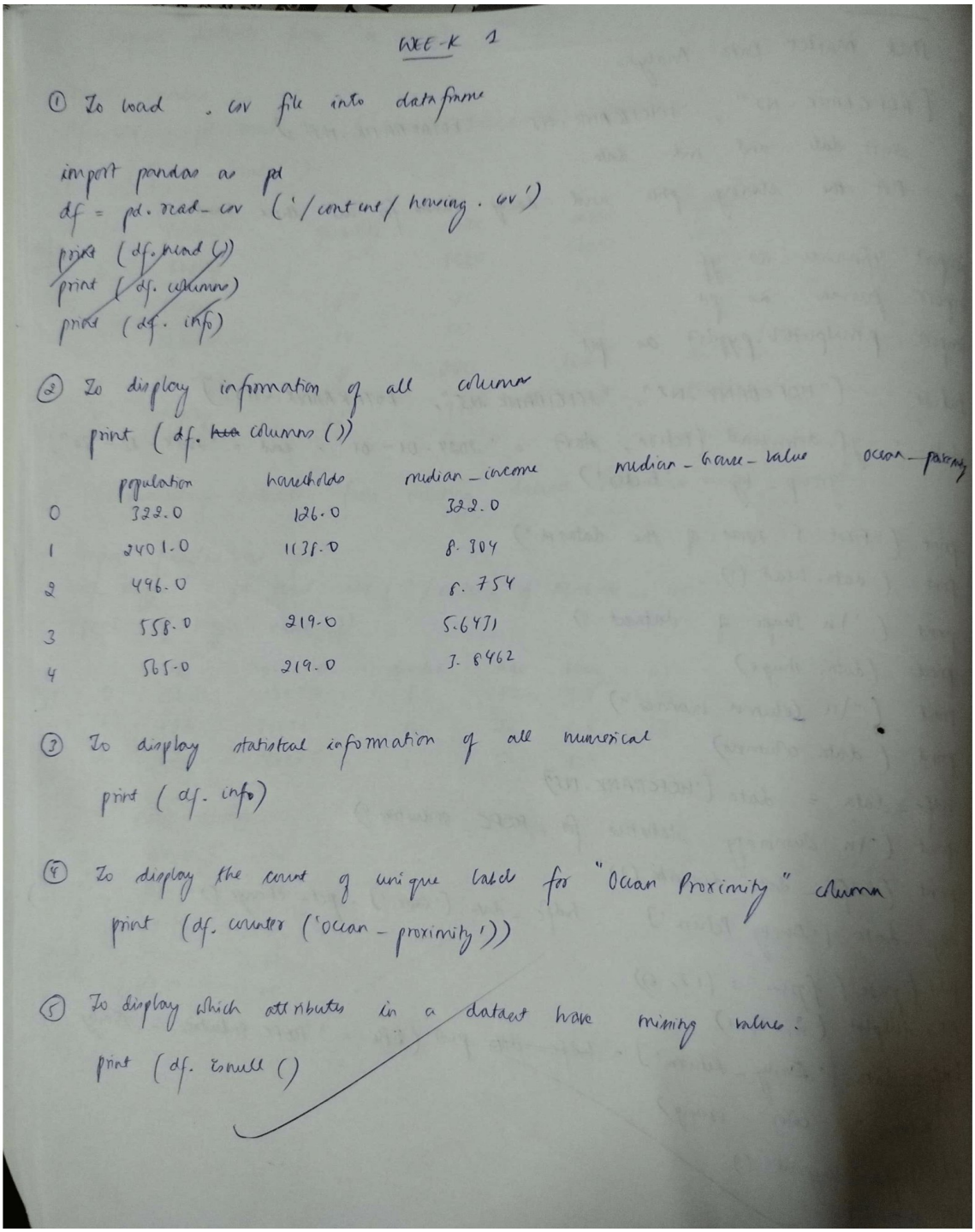
```
[ ]  # Load data from a CSV file (replace 'data.csv' with your file path)
     file_path = '/content/drive/MyDrive/SEM6/ML/ML_LAB/ML_LAB0/ml_Lab-0/Lab-0/sample_sales_data.csv'  # Ensure the file exists in the same directory
     df = pd.read_csv(file_path)
     print("Sample data:")
     print(df.head())
     print("\n")
```

```
[ ]  # Load data from a CSV file (replace 'data.csv' with your file path)
     file_path = '/content/drive/MyDrive/SEM6/ML/ML_LAB/ML_LAB0/ml_Lab-0/Lab-0/Dataset of Diabetes .csv'  # Ensure the file exists in the same directory
     df = pd.read_csv(file_path)
     print("Sample data:")
     print(df.head())
     print("\n")
```

Program 2

Demonstrate various data pre-processing techniques for a given dataset.

Screenshot:



o/p

	ocean_proximity
0	NEAR BAY
1	NEAR BAY
2	INLAND
.	
.	
.	

Sem
05-03-24

For Diabetes dataset, apply data preprocessing technique —

```
import pandas as pd
import numpy as np
import os
import sklearn.preprocessing import LabelEncoder, MinMaxScaler, StandardScaler

diabetes_df = pd.read_csv("/content/Dataset_of_Diabetes.csv")

# Check for missing value
print("Missing values in Diabetes dataset : \n", diabetes_df.isnull().sum())

# Handle missing values with mean
numeric_cols = diabetes_df.select_dtypes(include=np.number).columns
diabetes_df[numeric_cols] = diabetes_df[numeric_cols].fillna(diabetes_df[numeric_cols].
                                                             mean())

# To fill missing categorical values with mode
for col in diabetes_df.select_dtypes(include=['object']).columns:
    diabetes_df[col].fillna(diabetes_df[col].mode()[0], inplace=True)

# Identify categorical values and encode them.
categorical_cols = ['Gender', 'CLASS']
label_enc = LabelEncoder()
for col in categorical_cols:
    diabetes_df[col] = label_enc.fit_transform(diabetes_df[col])

# Apply min max scaling
min_max_scaler = MinMaxScaler()
diabetes_df_scaled = pd.DataFrame(min_max_scaler.fit_transform(diabetes_df),
                                  columns=diabetes_df.columns)
```

Apply standardization

① std_scaler = StandardScaler()
diabetes_df_standardized = pd.DataFrame(std_scaler.fit_transform(diabetes_df), columns = diabetes_df.columns)

i # Save the preprocessed datasets
d diabetes_df_scaled.to_csv("/content/diabetes_preprocessed.csv", index = False)
p os.makedirs("content", exist_ok = True)
p diabetes_df_standardized.to_csv("/content/diabetes_standardized.csv", index = False)
l print("Diabetes dataset preprocessing completed. Preprocessed datasets saved.")

②

Q1. Which columns in the dataset had missing values? How did you handle them?

- Numeric columns such as glucose level, blood pressure, insulin -
- Categorical columns such as Gender and CLASS

0
1 To handle numeric data, mean of the respective column was used -
2 categorical data, mode
3
4 Q2. Which categorical columns did you identify in the dataset? How did you encode?

Gender and CLASS are the two categorical columns.

③ Encoding method - label encoding Male → 0
Female → 1

Q3. What is the difference between min-max scaling and standardization?

④ • Min Max Scaling - transforms data to fixed range [0,1]

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

⑤ • Standardization - transforms data so that it has a mean of 0 and standard deviation of 1

$$x' = \frac{x - \mu}{\sigma}$$

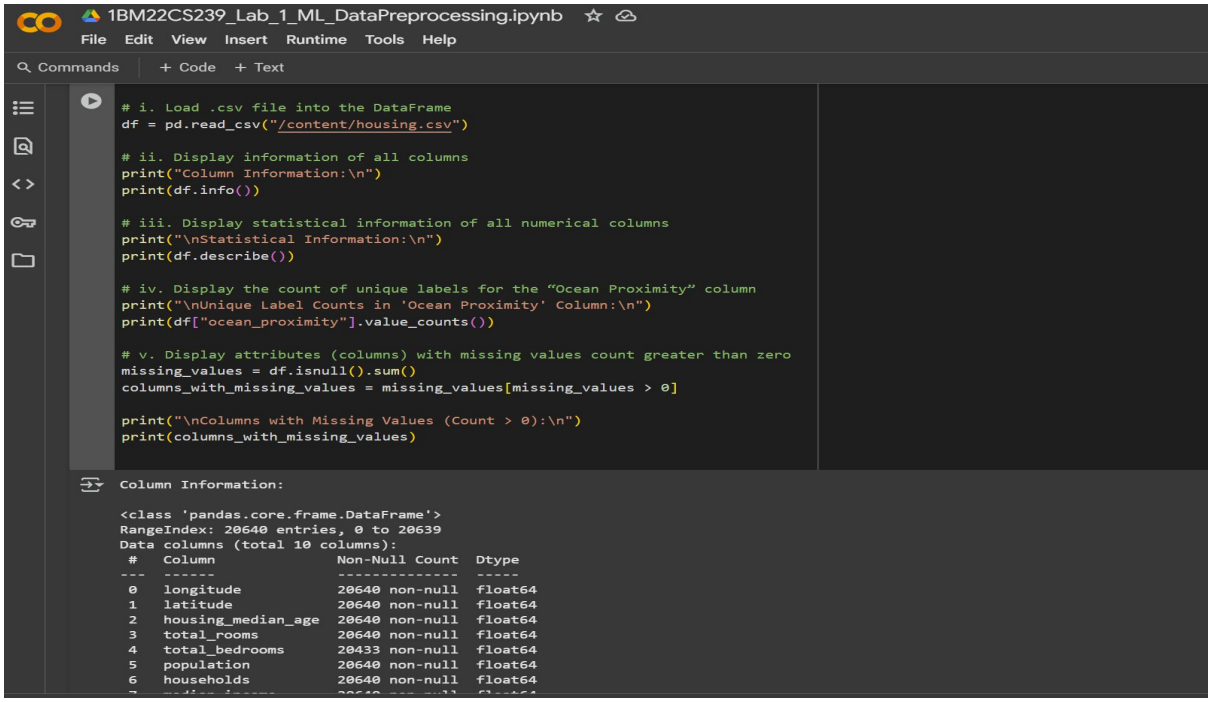
Code:

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

try:
    diabetes_df = pd.read_csv('diabetes.csv')
    adult_df = pd.read_csv('adult.csv')
except FileNotFoundError:
    print("Error: Please upload 'diabetes.csv' and 'adult.csv' to your Google Colab
environment.")
    exit()

diabetes_df.head(10)
adult_df.head(10)

diabetes_df.shape
adult_df.shape

#Handling Missing Values
diabetes_numeric_cols = diabetes_df.select_dtypes(include=[np.number]).columns
diabetes_categorical_cols = diabetes_df.select_dtypes(exclude=[np.number]).columns

adult_numeric_cols = adult_df.select_dtypes(include=[np.number]).columns
adult_categorical_cols = adult_df.select_dtypes(exclude=[np.number]).columns

diabetes_numeric_imputer = SimpleImputer(strategy='mean')
adult_numeric_imputer = SimpleImputer(strategy='mean')
diabetes_df[diabetes_numeric_cols] =
diabetes_numeric_imputer.fit_transform(diabetes_df[diabetes_numeric_cols])
adult_df[adult_numeric_cols] =
```

```
adult_numeric_imputer.fit_transform(adult_df[adult_numeric_cols])

diabetes_categorical_imputer = SimpleImputer(strategy='most_frequent')
adult_categorical_imputer = SimpleImputer(strategy='most_frequent')
diabetes_df[diabetes_categorical_cols] =
diabetes_categorical_imputer.fit_transform(diabetes_df[diabetes_categorical_cols])
adult_df[adult_categorical_cols] =
adult_categorical_imputer.fit_transform(adult_df[adult_categorical_cols])

print("Missing values in Diabetes dataset after imputation:")
print(diabetes_df.isnull().sum())
print("Missing values in Adult Income dataset after imputation:")
print(adult_df.isnull().sum())

adult_df.replace("?", np.nan, inplace=True)
print("Missing values in Adult Income dataset after replacing '?':")
print(adult_df.isnull().sum())

from sklearn.impute import SimpleImputer

# Identify numeric and categorical columns
adult_numeric_cols = adult_df.select_dtypes(include=[np.number]).columns
adult_categorical_cols = adult_df.select_dtypes(exclude=[np.number]).columns

# Handle missing values in numeric columns using mean imputation
adult_numeric_imputer = SimpleImputer(strategy='mean')
adult_df[adult_numeric_cols] =
adult_numeric_imputer.fit_transform(adult_df[adult_numeric_cols])

# Handle missing values in categorical columns using most frequent imputation
adult_categorical_imputer = SimpleImputer(strategy='most_frequent')
adult_df[adult_categorical_cols] =
adult_categorical_imputer.fit_transform(adult_df[adult_categorical_cols])
print("Missing values in Adult Income dataset after imputation:")
```

```
print(adult_df.isnull().sum())

from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()

# Encode categorical columns in Diabetes dataset
for col in diabetes_categorical_cols:
    diabetes_df[col] = label_encoder.fit_transform(diabetes_df[col])

# Encode categorical columns in Adult Income dataset
for col in adult_categorical_cols:
    adult_df[col] = label_encoder.fit_transform(adult_df[col])
print("Encoded columns in Diabetes dataset:")
print(diabetes_df.head())
print("Encoded columns in Adult Income dataset:")
print(adult_df.head())

#Handling outliers
def remove_outliers(df):
    Q1 = df.quantile(0.25)
    Q3 = df.quantile(0.75)
    IQR = Q3 - Q1
    df_no_outliers = df[~((df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR))).any(axis=1)]
    return df_no_outliers

diabetes_df_no_outliers = remove_outliers(diabetes_df)
adult_df_no_outliers = remove_outliers(adult_df)
print("Diabetes dataset shape after removing outliers:", diabetes_df_no_outliers.shape)
print("Adult Income dataset shape after removing outliers:", adult_df_no_outliers.shape)

#Min-max scaling
from sklearn.preprocessing import MinMaxScaler
min_max_scaler = MinMaxScaler()
diabetes_scaled_minmax =
```

```
pd.DataFrame(min_max_scaler.fit_transform(diabetes_df_no_outliers),
columns=diabetes_df_no_outliers.columns)
adult_scaled_minmax = pd.DataFrame(min_max_scaler.fit_transform(adult_df_no_outliers),
columns=adult_df_no_outliers.columns)
print("Diabetes dataset after Min-Max scaling:")
print(diabetes_scaled_minmax.head())
print("Adult Income dataset after Min-Max scaling:")
print(adult_scaled_minmax.head())

# Initialize Standard Scaler
from sklearn.preprocessing import StandardScaler
standard_scaler = StandardScaler()
diabetes_scaled_standard =
pd.DataFrame(standard_scaler.fit_transform(diabetes_df_no_outliers),
columns=diabetes_df_no_outliers.columns)
adult_scaled_standard = pd.DataFrame(standard_scaler.fit_transform(adult_df_no_outliers),
columns=adult_df_no_outliers.columns)
print("Diabetes dataset after Standard scaling:")
print(diabetes_scaled_standard.head())
print("Adult Income dataset after Standard scaling:")
print(adult_scaled_standard.head())
```




```
None

Missing values in each column:
 ID           0
No_Pation     0
Gender        0
AGE           0
Urea          0
Cr            0
HbA1c         0
Chol          0
TG            0
HDL           0
LDL           0
VLDL          0
BMI           0
CLASS         0
dtype: int64

Categorical columns:
 Index(['Gender', 'CLASS'], dtype='object')

First few rows of the dataset:
     ID  No_Pation Gender  AGE  Urea  Cr  HbA1c  Chol   TG  HDL  LDL  VLDL  \
0  502      17975      F   50   4.7  46    4.9   4.2  0.9  2.4  1.4   0.5
1  735      34221      M   26   4.5  62    4.9   3.7  1.4  1.1  2.1   0.6
2  420      47975      F   50   4.7  46    4.9   4.2  0.9  2.4  1.4   0.5
3  680      87656      F   50   4.7  46    4.9   4.2  0.9  2.4  1.4   0.5
4  504      34223      M   33   7.1  46    4.9   4.9  1.0  0.8  2.0   0.4

    BMI CLASS
0  24.0     N
1  23.0     N
2  24.0     N
3  24.0     N
4  21.0     N
```

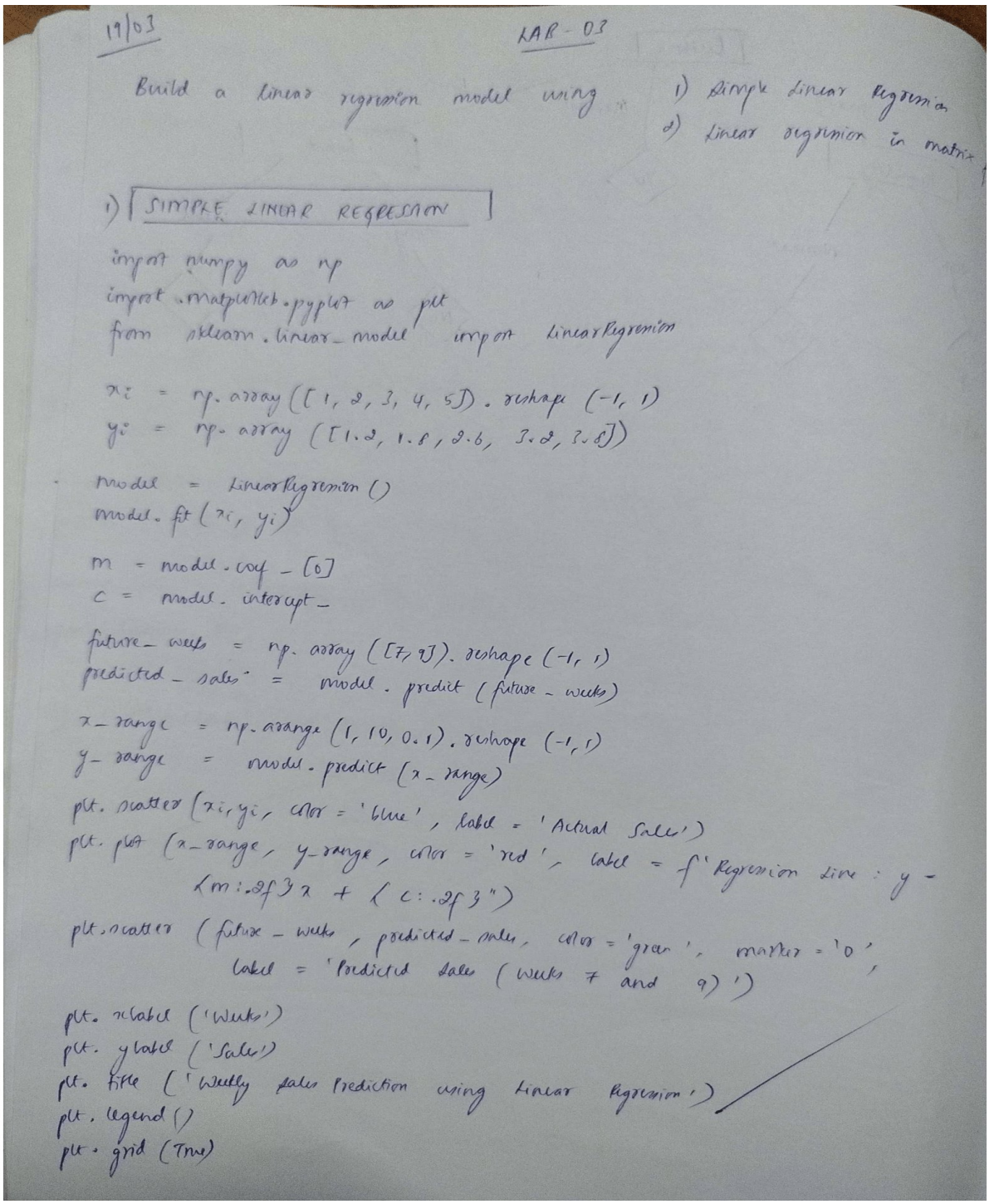
```python
# Step 6: Apply Min-Max Scaling
min_max_scaler = MinMaxScaler()
adult_df_scaled = pd.DataFrame(min_max_scaler.fit_transform(adult_df), columns=adult_df.columns)

# Step 7: Apply Standardization
std_scaler = StandardScaler()
adult_df_standardized = pd.DataFrame(std_scaler.fit_transform(adult_df), columns=adult_df.columns)

# Step 8: Save the preprocessed datasets
adult_df_scaled.to_csv("/content/adult_preprocessed.csv", index=False)

# Create the 'content' directory if it doesn't exist
os.makedirs("content", exist_ok=True)

adult_df_standardized.to_csv("content/adult_standardized.csv", index=False)

print("Adult Income dataset preprocessing completed. Preprocessed datasets saved.")
```

```
Missing values in Adult Income dataset:
 age                0
workclass           0
fnlwgt              0
education           0
educational-num     0
marital-status      0
occupation          0
relationship        0
race                0
gender              0
capital-gain        0
capital-loss        0
hours-per-week      0
native-country      0
income              0
dtype: int64
Adult Income dataset preprocessing completed. Preprocessed datasets saved.
```

Program 3

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset.

Screenshot:



17/03

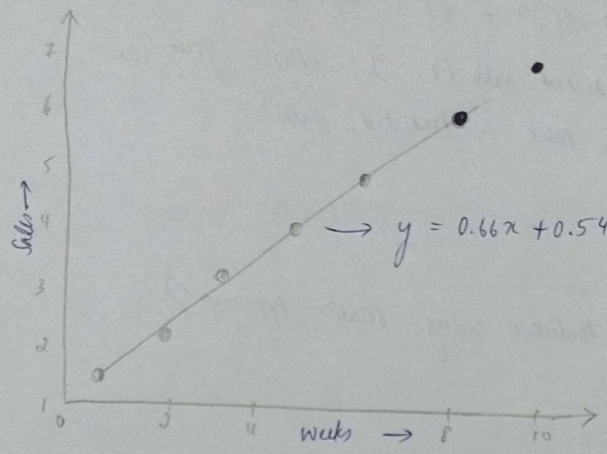
LAB - 03

Build a linear regression model using : 1) Simple Linear Regression
2) Linear regression in matrix

1) SIMPLE LINEAR REGRESSION

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

xi = np.array([1, 2, 3, 4, 5]).reshape(-1, 1)
yi = np.array([1.2, 1.8, 2.6, 3.2, 3.8])

model = LinearRegression()
model.fit(xi, yi)

m = model.coef_[0]
c = model.intercept_

future_weeks = np.array([7, 9]).reshape(-1, 1)
predicted_sales = model.predict(future_weeks)

x_range = np.arange(1, 10, 0.1).reshape(-1, 1)
y_range = model.predict(x_range)

plt.scatter(xi, yi, color = 'blue', label = 'Actual Sales')
plt.plot(x_range, y_range, color = 'red', label = f'Regression line : y =
        {m:.2f}x + {c:.2f}')
plt.scatter(future_weeks, predicted_sales, color = 'green', marker = 'o',
        label = 'Predicted Sales (Weeks 7 and 9)')

plt.xlabel('Weeks')
plt.ylabel('Sales')
plt.title('Weekly Sales Prediction using Linear Regression')
plt.legend()
plt.grid(True)
```

```
plt.show()
print(f"Equation of the regression line : y = {m:.2f}x + {c:.2f}")
print(f"Predicted sales for week 7 : {predicted_sales[0]:.2f}")
print(f"Predicted sales for week 9 : {predicted_sales[1]:.2f}")
```



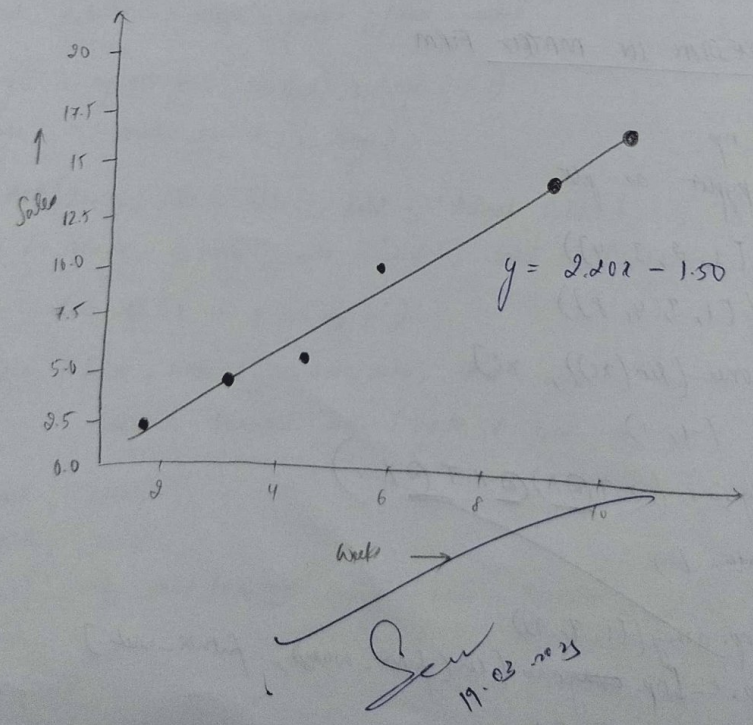
Equation of regression line : $y = 0.66x + 0.54$
Predicted sales for week 7 : 5.16
week 9 : 6.48

(2) LINEAR REGRESSION IN MATRIX FORM

```
import numpy as np
import matplotlib.pyplot as plt
xi = np.array([1, 2, 3, 4])
yi = np.array([1, 3, 4, 8])
X = np.c_[np.ones(len(xi)), xi]
Y = yi.reshape(-1, 1)
theta = np.linalg.inv(X.T@X)@X.T@Y
c, m = theta.flatten()
future_week = np.array([1, 7, 9])
X_future = np.c_[np.ones(len(future_week)), future_week]
predicted_sales = X_future@theta
```

```
x_range = np.linspace(1, 10, 100)
y_range = c + m * x_range

plt.scatter(xi, yi, color = 'blue', label = 'Actual Sales')
plt.plot(x_range, y_range, color = 'red', label = f'Regression Line:
        y = {m:.2f}x + {c:.2f}')
plt.scatter([7,9], predicted_sales[1:3], color = 'green',
        marker = 'o', label = 'Predicted_Sales')

plt.xlabel('Weeks')
plt.ylabel('Sales')
plt.title('Weekly Sale Prediction using Matrix Approach')
plt.legend()
plt.grid(True)
plt.show()
```

O/P Equation of the regression line $y = 2.20x - 1.50$.



19.03.2025

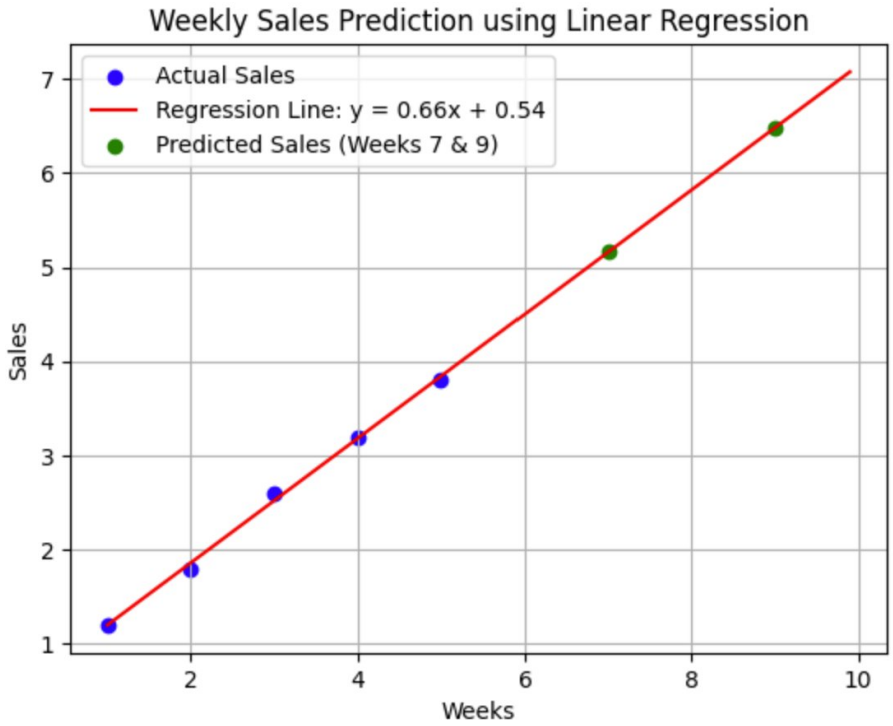
Code:

```
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')
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
reg = linear_model.LinearRegression()
reg.fit(new_df,price)

#(1) Predict price of a home with area = 3300 sqr ft
reg.predict([[3300]])
reg.coef_
reg.intercept_
3300*135.78767123 + 180616.43835616432

#(2) Predict price of a home with area = 5000 sqr ft
reg.predict([[5000]])

df = pd.read_csv('homeprices_Multiple_LR.csv')
df.bedrooms.median()
df.bedrooms = df.bedrooms.fillna(df.bedrooms.median())
reg = linear_model.LinearRegression()
reg.fit(df.drop('price',axis='columns'),df.price)
reg.coef_
reg.intercept_
```

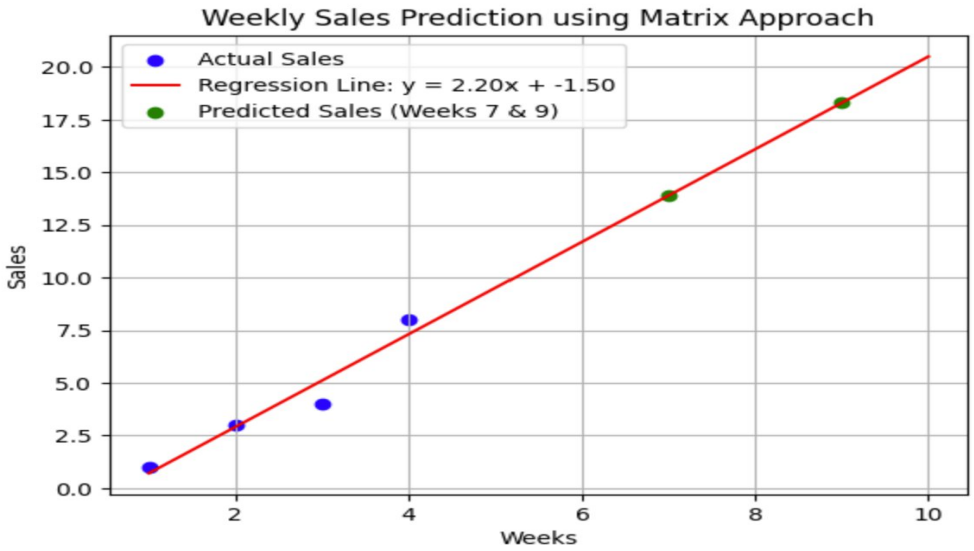
```
#Find price of home with 3000 sqr ft area, 3 bedrooms, 40 year old
reg.predict([[3000, 3, 40]])
112.06244194*3000 + 23388.88007794*3 + -3231.71790863*40 + 221323.00186540384

df = pd.read_csv('canada_per_capita_income.csv')
print(df.head())
X = df[['year']]
y = df['per capita income (US$)']
reg = LinearRegression()
reg.fit(X, y)
predicted_income_2020 = reg.predict([[2020]])
print(f"Predicted per capita income for Canada in 2020: {predicted_income_2020[0]:.2f}")

plt.scatter(X, y, color='blue')
plt.plot(X, reg.predict(X), color='red')
plt.xlabel('Year')
plt.ylabel('Per Capita Income')
plt.title('Per Capita Income in Canada Over the Years')
plt.show()

df = pd.read_csv('salary.csv')
print(df.head())
print("Missing values in the dataset:")
print(df.isnull().sum())

df['YearsExperience'] = df['YearsExperience'].fillna(df['YearsExperience'].median())
print("\nMissing values after filling:")
print(df.isnull().sum())
X = df[['YearsExperience']]
y = df['Salary']
reg = LinearRegression()
reg.fit(X, y)
predicted_salary_12_years = reg.predict([[12]])
```

```
print(f"\nPredicted salary for an employee with 12 years of experience:
${predicted_salary_12_years[0]:,.2f}")

plt.scatter(X, y, color='blue')
plt.plot(X, reg.predict(X), color='red')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.title('Salary vs. Years of Experience')
plt.show()

def convert_to_numeric(value):
    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, 'thirteen': 13, 'fourteen': 14,
        'fifteen': 15
    }
    return word_to_num.get(value.lower(), value) if isinstance(value, str) else value

df_hiring = pd.read_csv('hiring.csv')
print(df.head())
df_hiring['experience'] = df_hiring['experience'].apply(convert_to_numeric)

df_hiring['experience'].fillna(0, inplace=True)
df_hiring['test_score(out of 10)'].fillna(df_hiring['test_score(out of 10)'].median(),
inplace=True)
df_hiring['interview_score(out of 10)'].fillna(df_hiring['interview_score(out of
10)'].median(), inplace=True)
X_hiring = df_hiring[['experience', 'test_score(out of 10)', 'interview_score(out of 10)']]
y_hiring = df_hiring['salary($)']
reg_hiring = LinearRegression()
reg_hiring.fit(X_hiring, y_hiring)
candidates = np.array([[2, 9, 6], [12, 10, 10]])
predicted_salaries = reg_hiring.predict(candidates)
```

```
for i, candidate in enumerate(candidates):
    print(f"\nPredicted salary for candidate with {candidate[0]} yrs experience, {candidate[1]} test score, {candidate[2]} interview score: {predicted_salaries[i]:.2f} USD")

plt.scatter(y_hiring, reg_hiring.predict(X_hiring), color='blue', label='Predicted vs Actual')
plt.xlabel("Actual Salary")
plt.ylabel("Predicted Salary")
plt.title("Actual vs Predicted Salary")
plt.legend()
plt.show()

df_companies = pd.read_csv('1000_Companies.csv')
print(df.head())
label_encoder = LabelEncoder()
df_companies['State'] = label_encoder.fit_transform(df_companies['State'])
X_companies = df_companies[['R&D Spend', 'Administration', 'Marketing Spend', 'State']]
y_companies = df_companies['Profit']
df_companies.fillna(df_companies.median(), inplace=True)
reg_companies = LinearRegression()
reg_companies.fit(X_companies, y_companies)

input_data = np.array([[91694.48, 515841.3, 11931.24,
label_encoder.transform(['Florida'])[0]]])
predicted_profit = reg_companies.predict(input_data)
print(f"Predicted profit: {predicted_profit[0]:.2f} USD")

plt.scatter(y_companies, reg_companies.predict(X_companies), color='blue', label='Predicted vs Actual')
plt.xlabel("Actual Profit")
plt.ylabel("Predicted Profit")
plt.title("Actual vs Predicted Profit")
plt.legend()
plt.show()
```




```
Equation of the regression line: y = 0.66x + 0.54
Predicted sales for week 7: 5.16
Predicted sales for week 9: 6.48
```

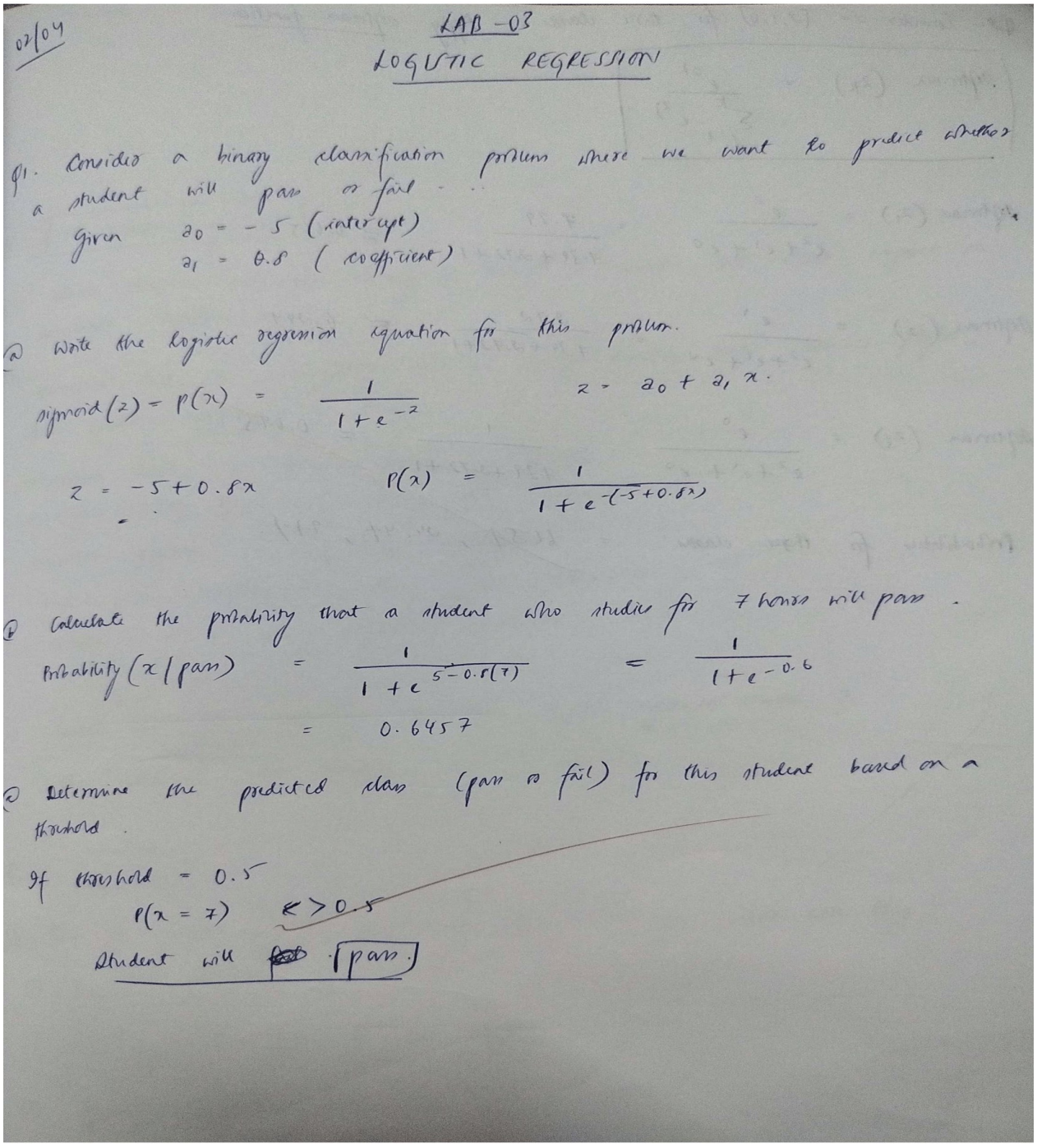



```
Equation of the regression line: y = 2.20x + -1.50
Predicted sales for week 7: 13.90
Predicted sales for week 9: 18.30
```

Program 4

Build Logistic Regression Model for a given dataset.

Screenshot:



Q9. Consider $z = [2, 1, 0]$ for three classes. Apply softmax function

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

$$\text{softmax}(z_1) = \frac{e^2}{e^2 + e^1 + e^0} = \frac{7.39}{7.39 + 2.72 + 1} \approx 0.665$$

$$\text{softmax}(z_2) = \frac{e^1}{e^2 + e^1 + e^0} = \frac{2.72}{7.39 + 2.72 + 1} \approx 0.244$$

$$\text{softmax}(z_3) = \frac{e^0}{e^2 + e^1 + e^0} = \frac{1}{7.39 + 2.72 + 1} \approx 0.091$$

Probabilities for three classes = 66.5%, 24.4%, 9.1%.

Q. HR_comma_sep.csv

(i) Which variable did you identify as having a direct and clear impact on employee retention? Why?

Variables such as satisfaction_level, average_monthly_hours, number_project, time_spend_company, salary.

- satisfaction level : strong negative correlation
- average_monthly_hours : strong positive correlation
- number_project : positive correlation
- time_spend_company : strong positive correlation
- salary : strong negative correlation

(ii) What was the accuracy of your logistic regression model?

The accuracy was around (0.79) (79%.

Q. Zoo.csv

(i) Did you perform any data preprocessing steps? If yes, what were they?

① Handling categorical data.

class_type → class_mapping

② Splitting dataset

80% training, 20% testing

③ Drop irrelevant columns

Drop animal_name as it is irrelevant.

④ Feature scaling.

standardScaler() to standardize numerical features mean = 0, variance = 1

(ii) Were there any missing or inconsistent values in the dataset?

Ans No missing values.

(iii) What does the confusion matrix tell you about the performance of your

The confusion matrix represents how well the model classifies different classes.

Diagonal values: Correct classification

Other values: Incorrect classification

(iv) Which class type were most frequently misclassified?

Ans Class 3, 5, and 6 were misclassified.

Possible reasons ① Feature similarity

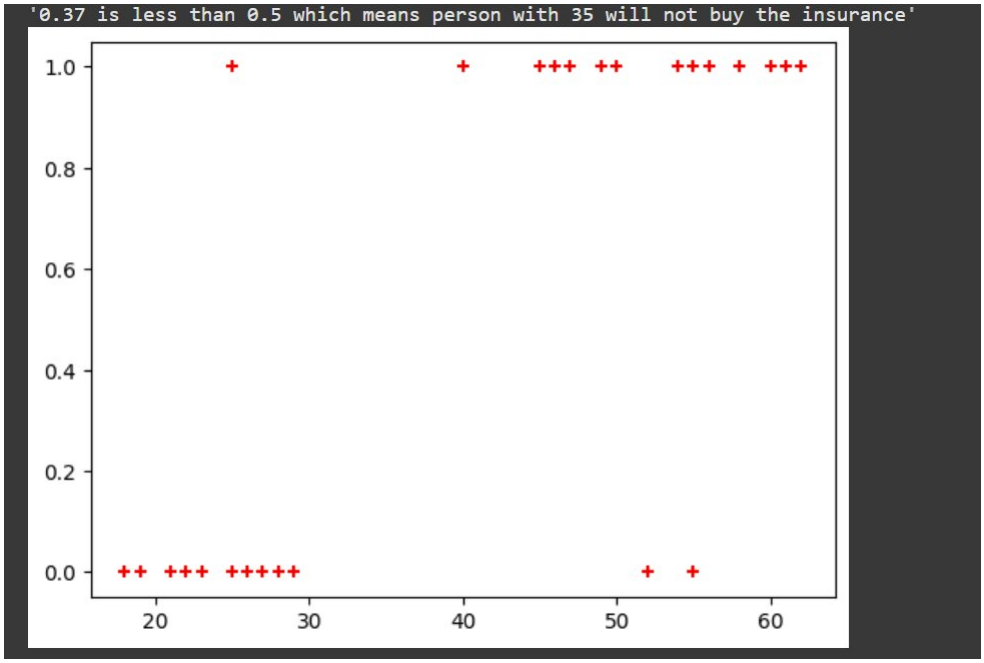
② Small sample size

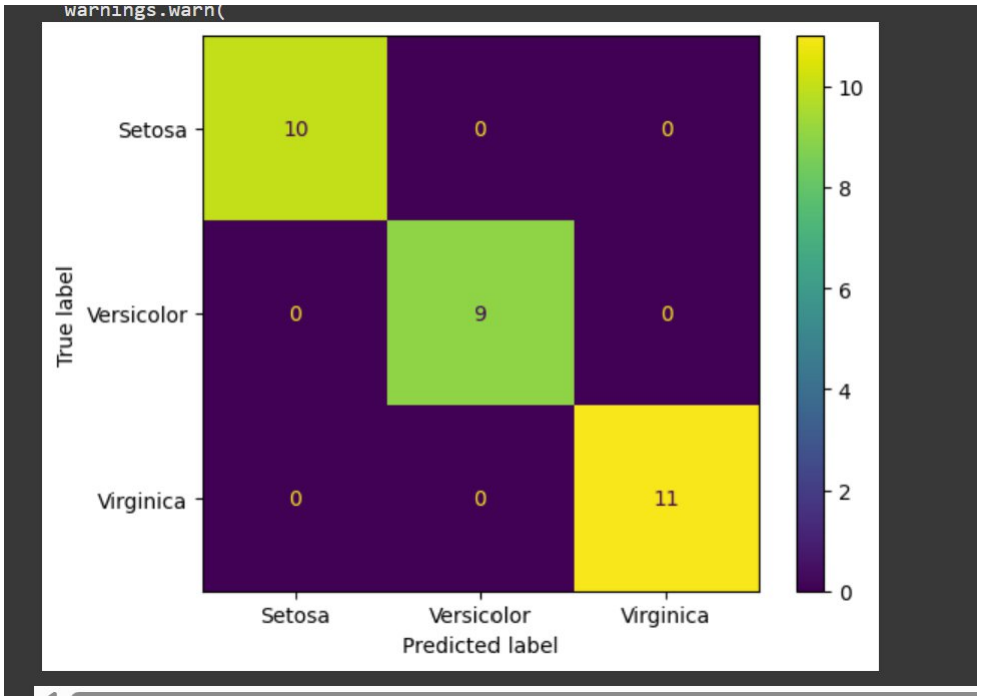
Code:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

df = pd.read_csv("HR_comma_sep.csv")
print(df.info())
numericCols = df.select_dtypes(include=['float64', 'int64']).columns
plt.figure(figsize=(10, 8))
sns.heatmap(df[numericCols].corr(), annot=True, cmap='coolwarm', fmt='.2f')
plt.title("Correlation Matrix (Numeric Features)")
plt.show()

plt.figure(figsize=(8, 6))
sns.countplot(x='salary', hue='left', data=df)
plt.title("Impact of Salary on Employee Retention")
plt.xlabel("Salary Level")
plt.ylabel("Employee Count")
plt.show()

import pandas as pd
df = pd.read_csv("zoo-data.csv")
print(df.info())
print(df.head())
print(df.isnull().sum())
df.drop(columns=['animal_name'], inplace=True)
X = df.drop(columns=['class_type'])
y = df['class_type']

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
X_train, X_test, y_train, y_test = train_test_split(
```

```
    X, y, test_size=0.2, random_state=42, stratify=y)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
logreg = LogisticRegression(max_iter=200, multi_class='multinomial', solver='lbfgs')
logreg.fit(X_train, y_train)

from sklearn.metrics import accuracy_score
y_pred = logreg.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy:.2f}")

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=logreg.classes_)
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix for Zoo Animal Classification")
plt.show()

y_pred = logreg.predict(X_test)
pred_classes = [class_mapping[pred] for pred in y_pred]
print("Predicted Classes:", pred_classes)

import seaborn as sns
import matplotlib.pyplot as plt
sns.countplot(x='class_type', data=df)
plt.title("Class Distribution of Animals in Zoo Dataset")
plt.xlabel("Class Type")
plt.ylabel("Count")
plt.show()

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
cm = confusion_matrix(y_test, y_pred)
```

```
class_labels = [class_mapping[num] for num in logreg.classes_]
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_labels)
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix with Class Names")
plt.show()
```

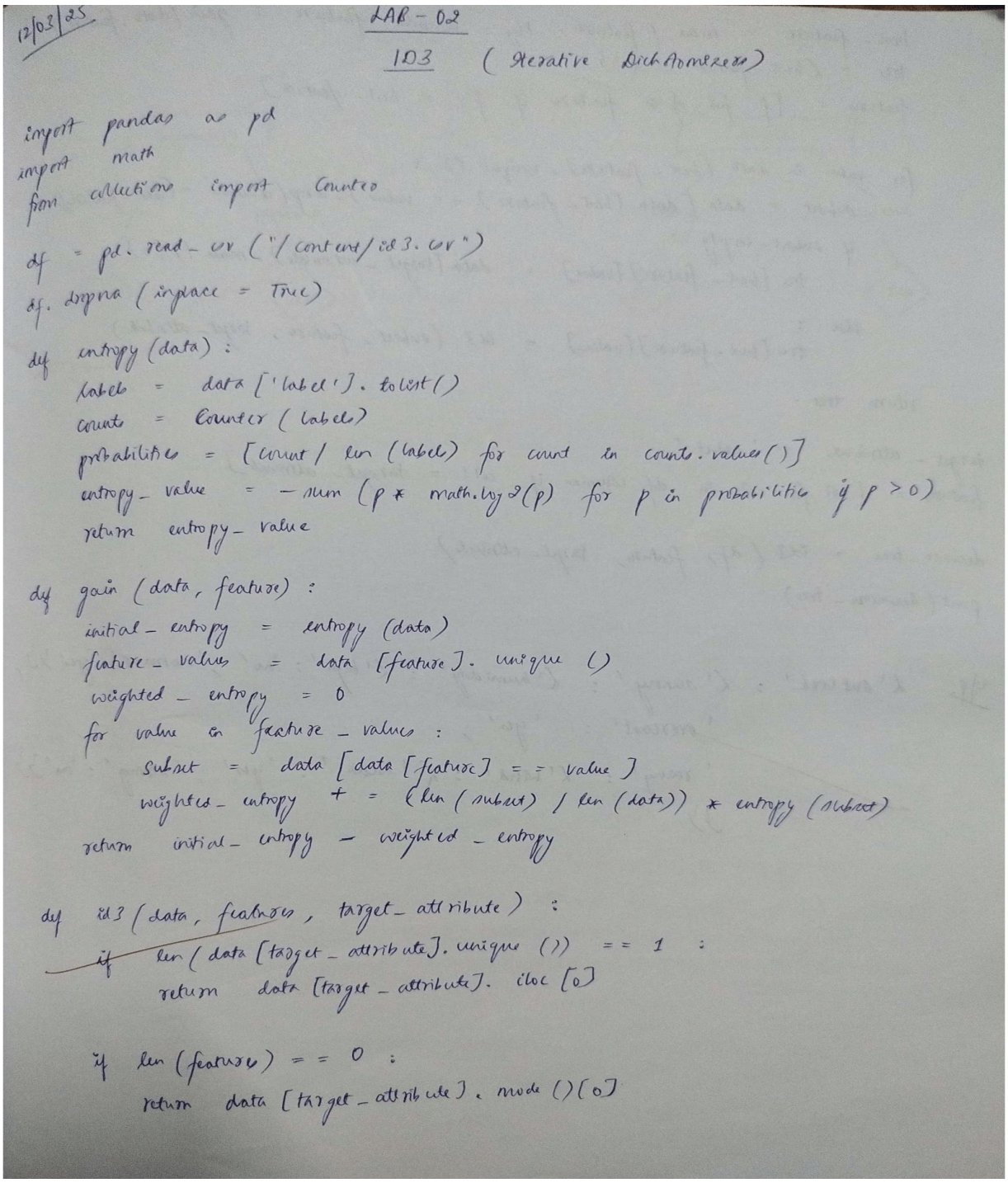




Program 5

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

Screenshot:



12/03/25

LAB - 02

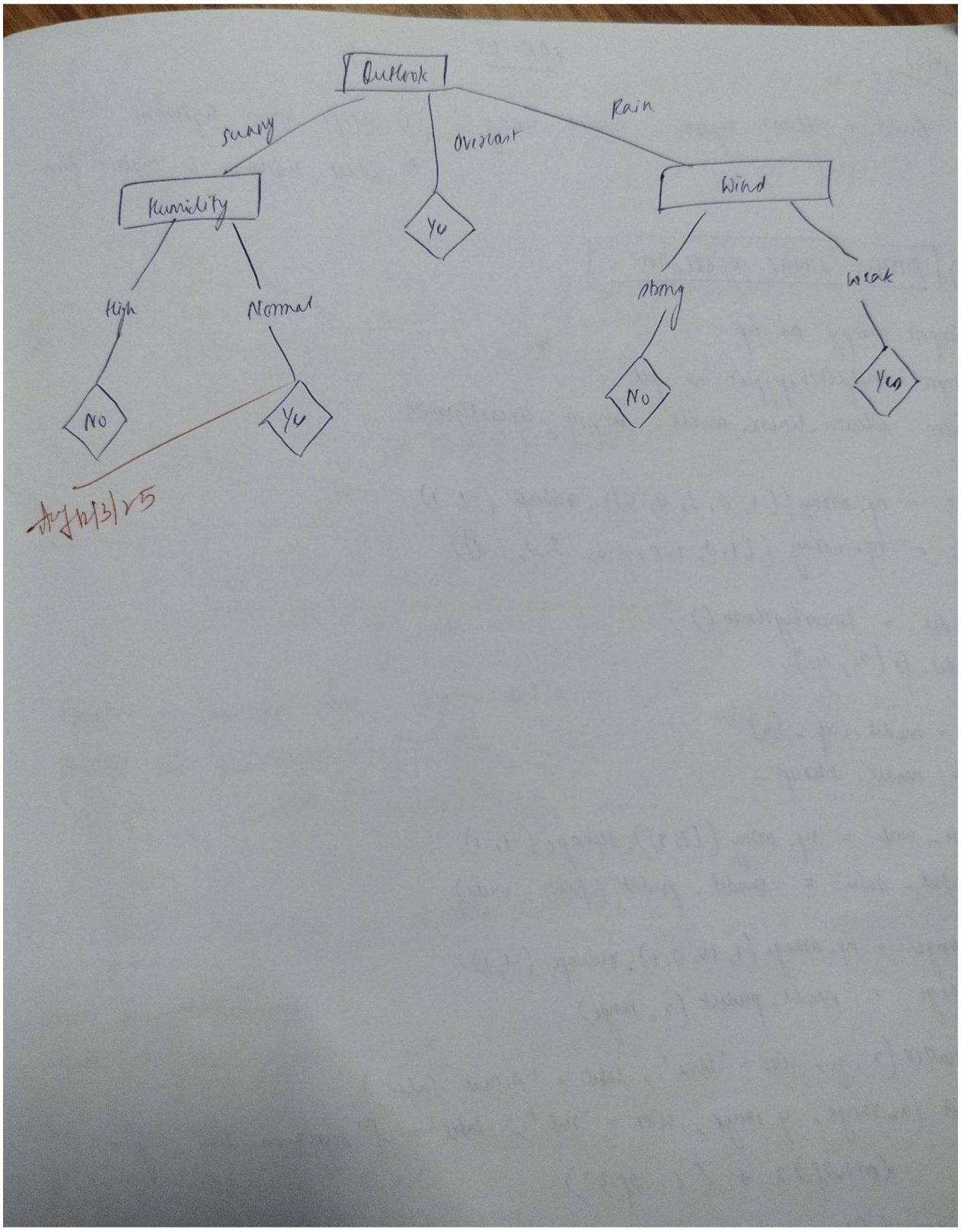
ID3 (Iterative Dichotomizer)

```
import pandas as pd
import math
from collections import Counter

df = pd.read_csv("/content/id3.csv")
df.dropna(inplace = True)

def entropy(data):
    labels = data['label'].tolist()
    counts = Counter(labels)
    probabilities = [count / len(labels) for count in counts.values()]
    entropy_value = -sum(p * math.log2(p) for p in probabilities if p > 0)
    return entropy_value

def gain(data, feature):
    initial_entropy = entropy(data)
    feature_values = data[feature].unique()
    weighted_entropy = 0
    for value in feature_values:
        subset = data[data[feature] == value]
        weighted_entropy += (len(subset) / len(data)) * entropy(subset)
    return initial_entropy - weighted_entropy

def id3(data, features, target_attribute):
    if len(data[target_attribute].unique()) == 1:
        return data[target_attribute].iloc[0]

    if len(features) == 0:
        return data[target_attribute].mode()[0]
```

```
best_feature = max(features, key = lambda feature : gain(data, feature))
tree = {best_feature : {}}
features = [f for f in features if f != best_feature]

for value in data[best_feature].unique():
    subset = data[data[best_feature] == value].drop(columns=[best_feature])
    if subset.empty:
        tree[best_feature][value] = data[target_attribute].mode()[0]
    else:
        tree[best_feature][value] = id3(subset, feature, target_attribute)

return tree.

target_attribute = 'Label'
features = [col for col in df.columns if col != target_attribute]

decision_tree = id3(df, features, target_attribute)
print(decision_tree)
```

O/P

```
{'outlook': {'sunny': {'humidity': {'high': 'no', 'normal': 'yes'}},
             'overcast': 'yes',
             'rainy': {'wind': {'weak': 'yes', 'strong': 'no'}}}}
```


Outlook
Sunny
Overcast
Rain
Humidity
Yes
Wind
High
Normal
Strong
Weak
No
Yes
No
Yes

Code:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score,
f1_score
from sklearn.preprocessing import LabelEncoder

def train_and_evaluate_iris():
    iris_df = pd.read_csv("iris.csv")
    X = iris_df.drop(columns=["species"])
    y = iris_df["species"]
    y_le = LabelEncoder()
    y = y_le.fit_transform(y)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
    model = DecisionTreeClassifier(random_state=42)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    # Evaluating the model
    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred, average='weighted')
    rec = recall_score(y_test, y_pred, average='weighted')
    f1 = f1_score(y_test, y_pred, average='weighted')
    cm = confusion_matrix(y_test, y_pred)

    print("IRIS Dataset Classification:")
    print(f"Accuracy Score: {acc:.4f}")
    print(f"Precision Score: {prec:.4f}")
```

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

    plt.figure(figsize=(6, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=y_le.classes_,
yticklabels=y_le.classes_)
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.title("Confusion Matrix: iris.csv")
    plt.show()

train_and_evaluate_iris()

def train_and_evaluate_drug():
    drug_df = pd.read_csv("drug.csv")
    categorical_features = ["Sex", "BP", "Cholesterol"]
    label_encoders = {}
    for col in categorical_features:
        le = LabelEncoder()
        drug_df[col] = le.fit_transform(drug_df[col])
        label_encoders[col] = le
    X = drug_df.drop(columns=["Drug"])
    y = drug_df["Drug"]
    y_le = LabelEncoder()
    y = y_le.fit_transform(y)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    model = DecisionTreeClassifier(random_state=42)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred, average='weighted')
    rec = recall_score(y_test, y_pred, average='weighted')
```

```
    f1 = f1_score(y_test, y_pred, average='weighted')
    cm = confusion_matrix(y_test, y_pred)

    print("Drug Dataset Classification:")
    print(f"Accuracy Score: {acc:.4f}")
    print(f"Precision Score: {prec:.4f}")
    print(f"Recall Score: {rec:.4f}")
    print(f"F1 Score: {f1:.4f}")

    plt.figure(figsize=(6, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=y_le.classes_,
yticklabels=y_le.classes_)
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.title("Confusion Matrix: drug.csv")
    plt.show()

train_and_evaluate_drug()

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor, plot_tree
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error

petrol_df = pd.read_csv("petrol_consumption.csv")
X = petrol_df.drop(columns=["Petrol_Consumption"])
y = petrol_df["Petrol_Consumption"]
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,
```

random_state=42)

model = DecisionTreeRegressor(max_depth=5, random_state=42)

model.fit(X_train, y_train)

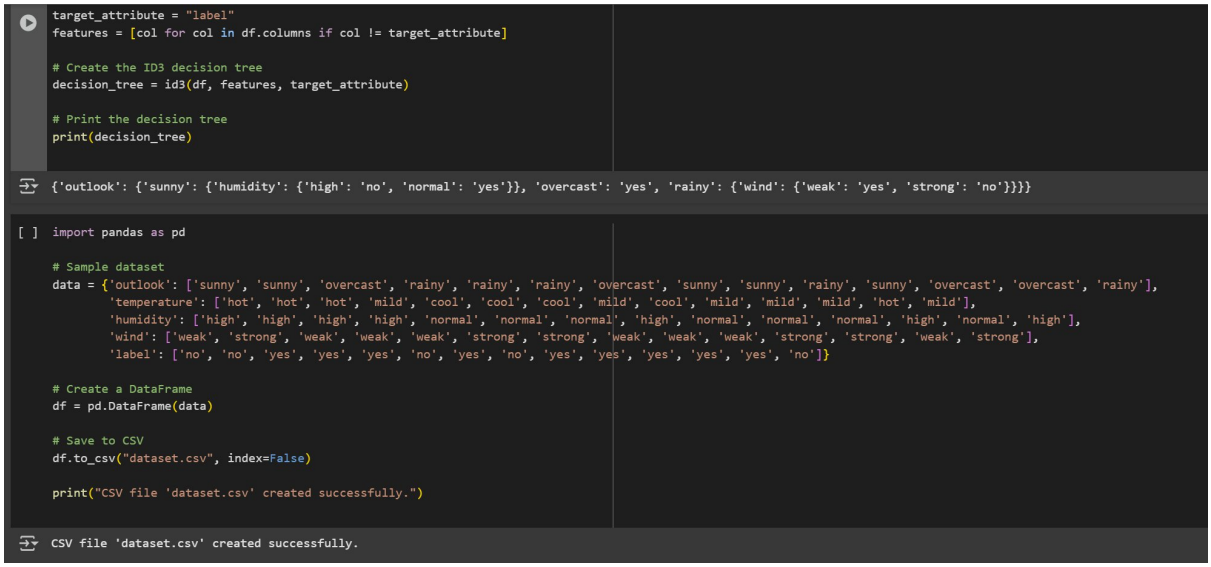
y_pred = model.predict(X_test)

print("Petrol Consumption Regression:")

print("Mean Absolute Error (MAE):", mean_absolute_error(y_test, y_pred))

print("Mean Squared Error (MSE):", mean_squared_error(y_test, y_pred))

print("Root Mean Squared Error (RMSE):", np.sqrt(mean_squared_error(y_test, y_pred)))



```
target_attribute = "label"
features = [col for col in df.columns if col != target_attribute]

# Create the ID3 decision tree
decision_tree = id3(df, features, target_attribute)

# Print the decision tree
print(decision_tree)
```

```
{'outlook': {'sunny': {'humidity': {'high': 'no', 'normal': 'yes'}}, 'overcast': 'yes', 'rainy': {'wind': {'weak': 'yes', 'strong': 'no'}}}}
```

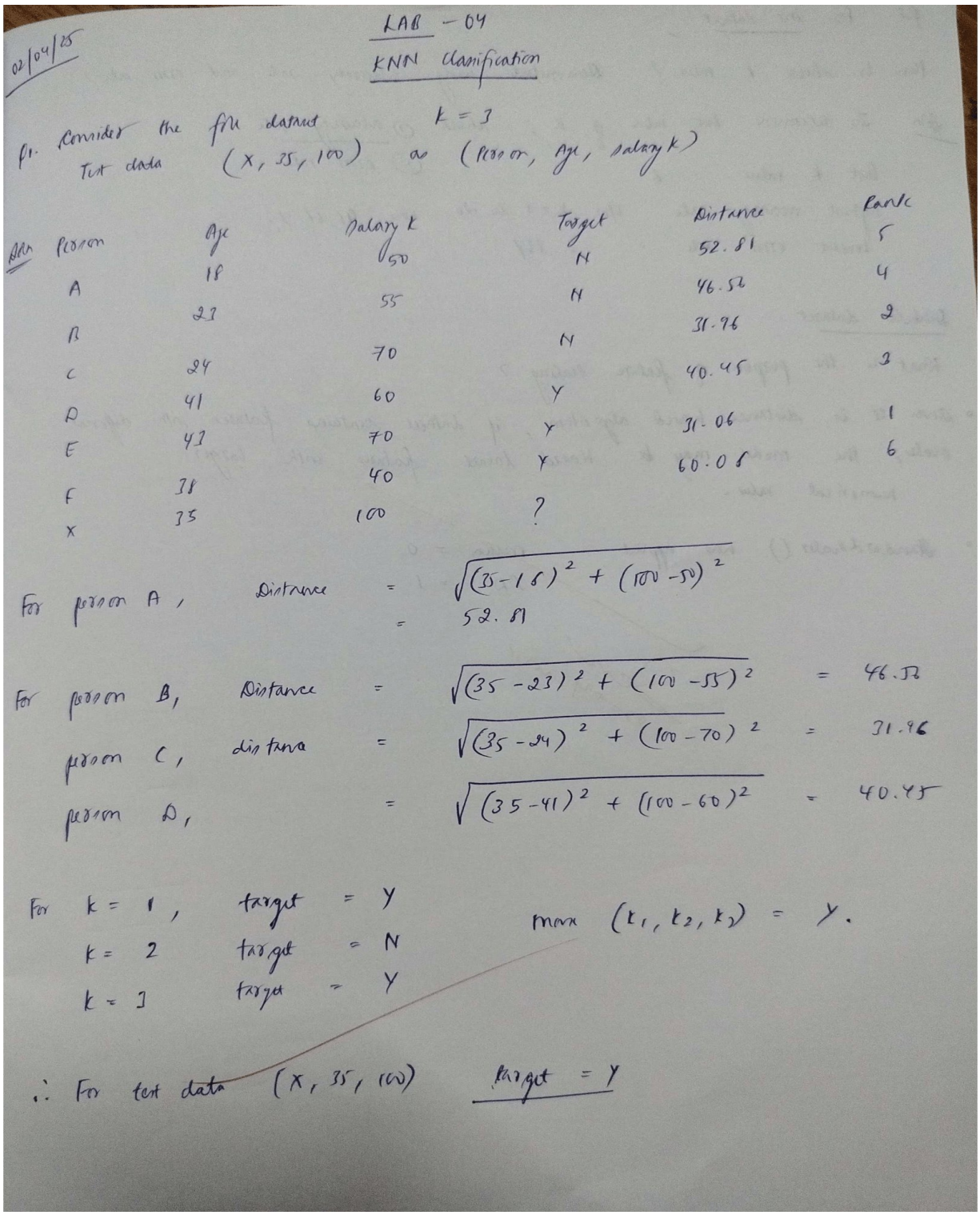
```
[ ] import pandas as pd

    # Sample dataset
    data = {'outlook': ['sunny', 'sunny', 'overcast', 'rainy', 'rainy', 'rainy', 'overcast', 'sunny', 'sunny', 'rainy', 'sunny', 'overcast', 'overcast', 'rainy'],
            'temperature': ['hot', 'hot', 'hot', 'mild', 'cool', 'cool', 'cool', 'mild', 'cool', 'mild', 'mild', 'mild', 'hot', 'mild'],
            'humidity': ['high', 'high', 'high', 'high', 'normal', 'normal', 'normal', 'high', 'normal', 'normal', 'normal', 'high', 'normal', 'high'],
            'wind': ['weak', 'strong', 'weak', 'weak', 'weak', 'strong', 'strong', 'weak', 'weak', 'weak', 'strong', 'strong', 'weak', 'strong'],
            'label': ['no', 'no', 'yes', 'yes', 'yes', 'no', 'yes', 'no', 'yes', 'yes', 'yes', 'yes', 'yes', 'no']}

    # Create a DataFrame
    df = pd.DataFrame(data)

    # Save to CSV
    df.to_csv("dataset.csv", index=False)

    print("CSV file 'dataset.csv' created successfully.")
```

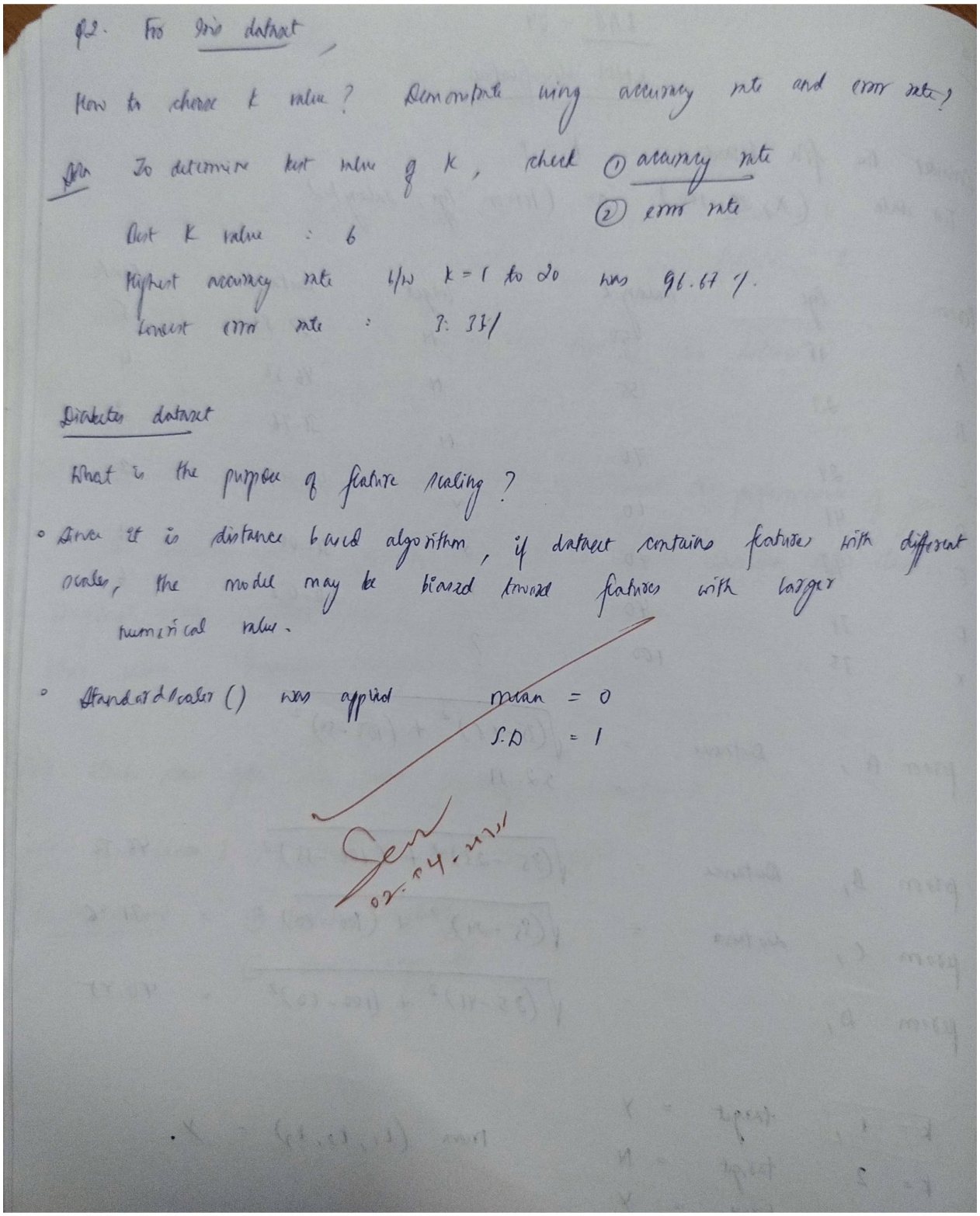
```
CSV file 'dataset.csv' created successfully.
```

Program 6

Build KNN Classification model for a given dataset.

Screenshot:




Q2. For Iris dataset
How to choose k value? Demonstrate using accuracy rate and error rate?
Soln To determine best value of k, check ① accuracy rate
② error rate
Best K value : 6
Highest accuracy rate i/p k = 1 to 20 was 96.67 %.
Lowest error rate : 3.33 %
Diabetes dataset
What is the purpose of feature scaling?
Since it is distance based algorithm, if dataset contains features with different scales, the model may be biased toward features with larger numerical value.
StandardScaler() was applied mean = 0
S.D = 1
02.04.2024

Code:

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

iris_df = pd.read_csv('iris.csv')
le = LabelEncoder()
iris_df['species'] = le.fit_transform(iris_df['species'])
X = iris_df.drop('species', axis=1)
y = iris_df['species']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

error_rates = []
accuracies = []
k_values = range(1, 10)
for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred_k = knn.predict(X_test)
    error = 1 - accuracy_score(y_test, y_pred_k)
    error_rates.append(error)
    accuracies.append(accuracy_score(y_test, y_pred_k))

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(k_values, accuracies, marker='o', color='blue')
plt.title("Accuracy vs K")
plt.xlabel("K Value")
plt.ylabel("Accuracy")
```

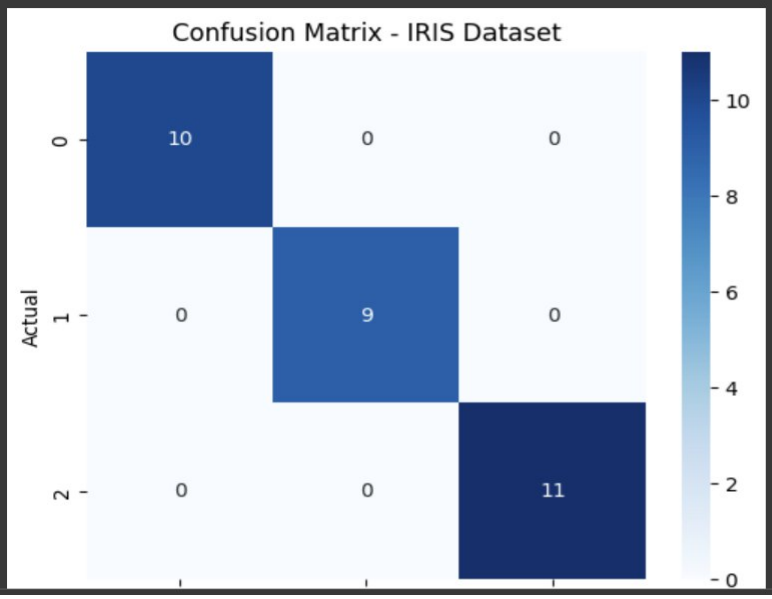
```
plt.subplot(1, 2, 2)
plt.plot(k_values, error_rates, marker='o', color='red')
plt.title("Error Rate vs K")
plt.xlabel("K Value")
plt.ylabel("Error Rate")
plt.tight_layout()
plt.show()
best_k = k_values[accuracies.index(max(accuracies))]
print(f"Best K: {best_k} with Accuracy: {max(accuracies):.2f}")

knn = KNeighborsClassifier(n_neighbors=best_k)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

# Evaluation
print("\n=== Final Evaluation on IRIS Dataset ===")
print("Accuracy Score:", accuracy_score(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred, labels=[0, 1, 2], target_names=le.classes_))

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=le.classes_, yticklabels=le.classes_)
plt.title("Confusion Matrix - IRIS")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

df = pd.read_csv('diabetes.csv')
X = df.drop('Outcome', axis=1)
y = df['Outcome']
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42, stratify=y
)
accuracy_scores = []
k_range = range(1, 21)

for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred_k = knn.predict(X_test)
    acc = accuracy_score(y_test, y_pred_k)
    accuracy_scores.append(acc)

plt.figure(figsize=(8, 5))
plt.plot(k_range, accuracy_scores, marker='o', color='purple')
plt.title("Accuracy vs K (Diabetes Dataset)")
plt.xlabel("K Value")
plt.ylabel("Accuracy")
plt.xticks(k_range)
plt.grid()
plt.show()
best_k = k_range[accuracy_scores.index(max(accuracy_scores))]
print(f"Best K: {best_k} with Accuracy: {max(accuracy_scores):.2f}")

knn = KNeighborsClassifier(n_neighbors=best_k)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

print("=== Final Evaluation (Diabetes Dataset) ===")
print("Accuracy Score:", accuracy_score(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

cm = confusion_matrix(y_test, y_pred)
```

```
sns.heatmap(cm, annot=True, fmt='d', cmap='Purples', xticklabels=['No Diabetes', 'Diabetes'],
yticklabels=['No Diabetes', 'Diabetes'])
plt.title("Confusion Matrix - Diabetes")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

heart_df = pd.read_csv('heart.csv')
X = heart_df.drop('target', axis=1)
y = heart_df['target']
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42, stratify=y
)
accuracy_scores = []
k_range = range(1, 21)

for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred_k = knn.predict(X_test)
    acc = accuracy_score(y_test, y_pred_k)
    accuracy_scores.append(acc)

plt.figure(figsize=(8, 5))
plt.plot(k_range, accuracy_scores, marker='o', color='red')
plt.title("Accuracy vs K (Heart Dataset)")
plt.xlabel("K Value")
plt.ylabel("Accuracy")
plt.xticks(k_range)
plt.grid()
plt.show()
```

```
best_k = k_range[accuracy_scores.index(max(accuracy_scores))]
print(f"Best K: {best_k} with Accuracy: {max(accuracy_scores):.2f}")

knn = KNeighborsClassifier(n_neighbors=best_k)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

print("=== Final Evaluation (Heart Dataset) ===")
print("\nAccuracy Score:", accuracy_score(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=['No Disease', 'Disease']))
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Reds', xticklabels=['No Disease', 'Disease'],
yticklabels=['No Disease', 'Disease'])
plt.title("Confusion Matrix - Heart Disease")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

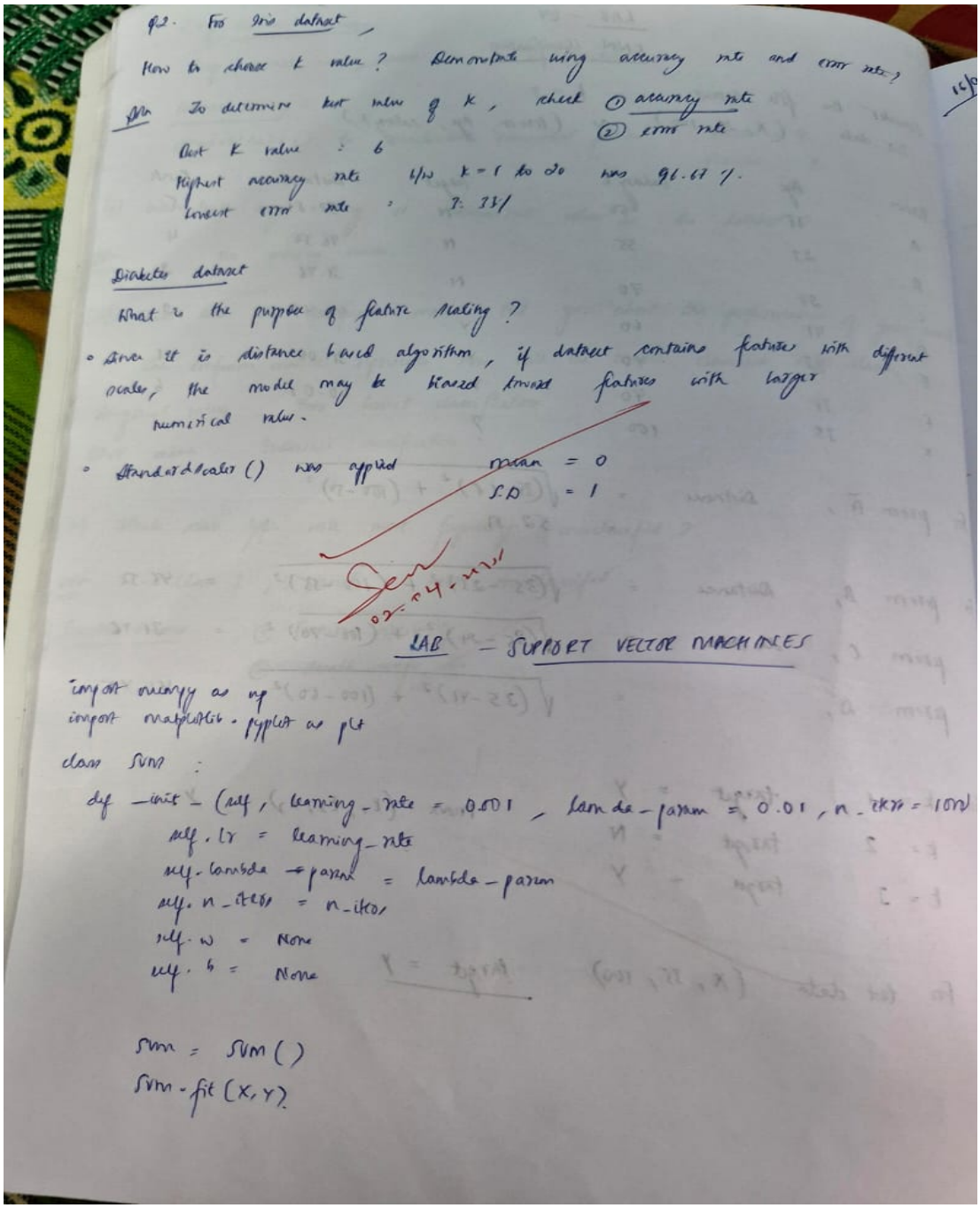
```
IRIS Dataset Accuracy: 1.0
Confusion Matrix for IRIS:
 [[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
Classification Report for IRIS:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      1.00      1.00         9
   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
```



Program 7

Build Support vector machine model for a given dataset.

Screenshot:



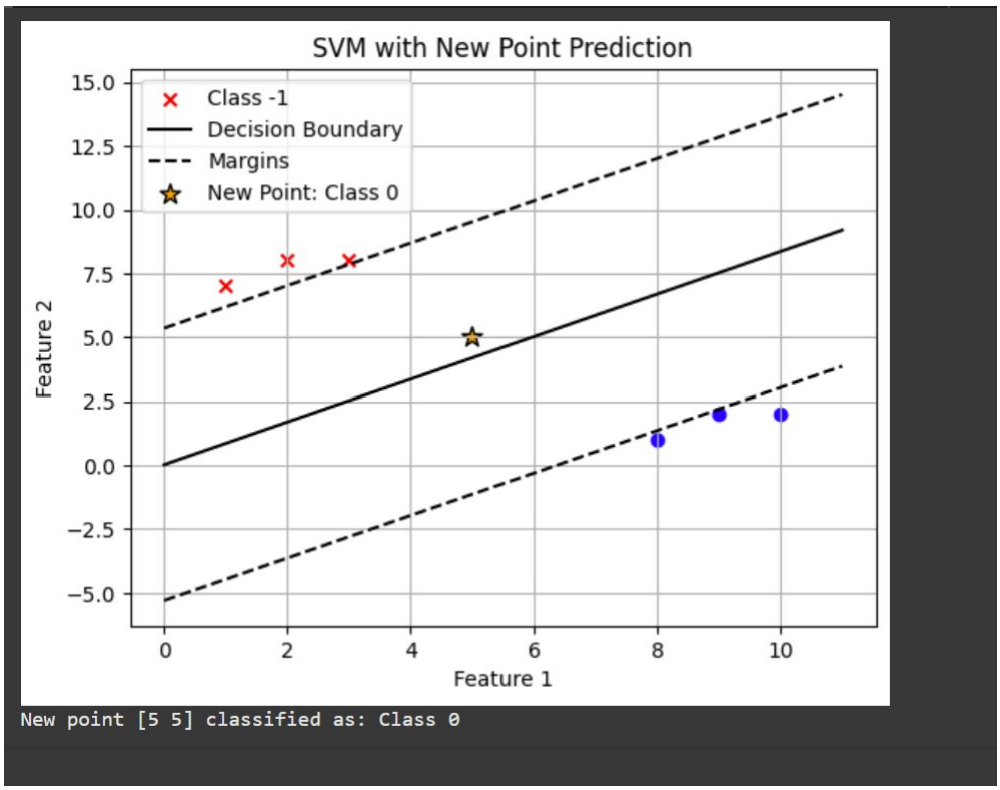
Code:

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

iris = pd.read_csv("iris.csv")
label_encoder = LabelEncoder()
iris['species'] = label_encoder.fit_transform(iris['species'])
class_names_iris = label_encoder.classes_
X_iris = iris.drop('species', axis=1)
y_iris = iris['species']
X_train_iris, X_test_iris, y_train_iris, y_test_iris = train_test_split(X_iris, y_iris,
test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train_iris = scaler.fit_transform(X_train_iris)
X_test_iris = scaler.transform(X_test_iris)
svm_linear = SVC(kernel='linear')
svm_linear.fit(X_train_iris, y_train_iris)
y_pred_linear = svm_linear.predict(X_test_iris)
acc_linear = accuracy_score(y_test_iris, y_pred_linear)
cm_linear = confusion_matrix(y_test_iris, y_pred_linear)

plt.figure(figsize=(6,4))
sns.heatmap(cm_linear, annot=True, fmt='d', cmap='Blues', xticklabels=class_names_iris,
yticklabels=class_names_iris)
plt.title(f'IRIS SVM Linear Kernel\nAccuracy: {acc_linear:.2f}')
```

```
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.tight_layout()
plt.show()

svm_rbf = SVC(kernel='rbf')
svm_rbf.fit(X_train_iris, y_train_iris)
y_pred_rbf = svm_rbf.predict(X_test_iris)
acc_rbf = accuracy_score(y_test_iris, y_pred_rbf)
cm_rbf = confusion_matrix(y_test_iris, y_pred_rbf)

plt.figure(figsize=(6,4))
sns.heatmap(cm_rbf, annot=True, fmt='d', cmap='Greens', xticklabels=class_names_iris,
yticklabels=class_names_iris)
plt.title(f'IRIS SVM RBF Kernel\nAccuracy: {acc_rbf:.2f}')
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.tight_layout()
plt.show()

letters = pd.read_csv("letter-recognition.csv")
X_letters = letters.drop('letter', axis=1)
y_letters = letters['letter']
label_encoder_letters = LabelEncoder()
y_letters_encoded = label_encoder_letters.fit_transform(y_letters)
class_names_letters = label_encoder_letters.classes_

X_train_letters, X_test_letters, y_train_letters, y_test_letters = train_test_split(
    X_letters, y_letters_encoded, test_size=0.2, random_state=42)

scaler_letters = StandardScaler()
X_train_letters = scaler_letters.fit_transform(X_train_letters)
X_test_letters = scaler_letters.transform(X_test_letters)
svm_letters = SVC(kernel='rbf', probability=True)
```

```
svm_letters.fit(X_train_letters, y_train_letters)

y_pred_letters = svm_letters.predict(X_test_letters)
acc_letters = accuracy_score(y_test_letters, y_pred_letters)
cm_letters = confusion_matrix(y_test_letters, y_pred_letters)

plt.figure(figsize=(14, 12))
sns.heatmap(cm_letters, annot=True, fmt='d', cmap='Purples',
            xticklabels=class_names_letters,
            yticklabels=class_names_letters,
            annot_kws={"size": 8},
            cbar=True)
plt.title(f'Letter Recognition - SVM RBF Kernel\nAccuracy: {acc_letters*100:.2f}%',
fontsize=16)
plt.xlabel("Predicted Label", fontsize=14)
plt.ylabel("True Label", fontsize=14)
plt.xticks(rotation=45)
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()

y_test_binarized = label_binarize(y_test_letters, classes=np.arange(len(class_names_letters)))
y_score = svm_letters.predict_proba(X_test_letters)
auc_score = roc_auc_score(y_test_binarized, y_score, average='macro')

fpr = dict()
tpr = dict()
for i in range(len(class_names_letters)):
    fpr[i], tpr[i], _ = roc_curve(y_test_binarized[:, i], y_score[:, i])

plt.figure(figsize=(8, 6))
for i in range(0, len(class_names_letters), 4): # Plot every 4th class
    plt.plot(fpr[i], tpr[i], lw=1.5, label=f'Class {class_names_letters[i]}')
```

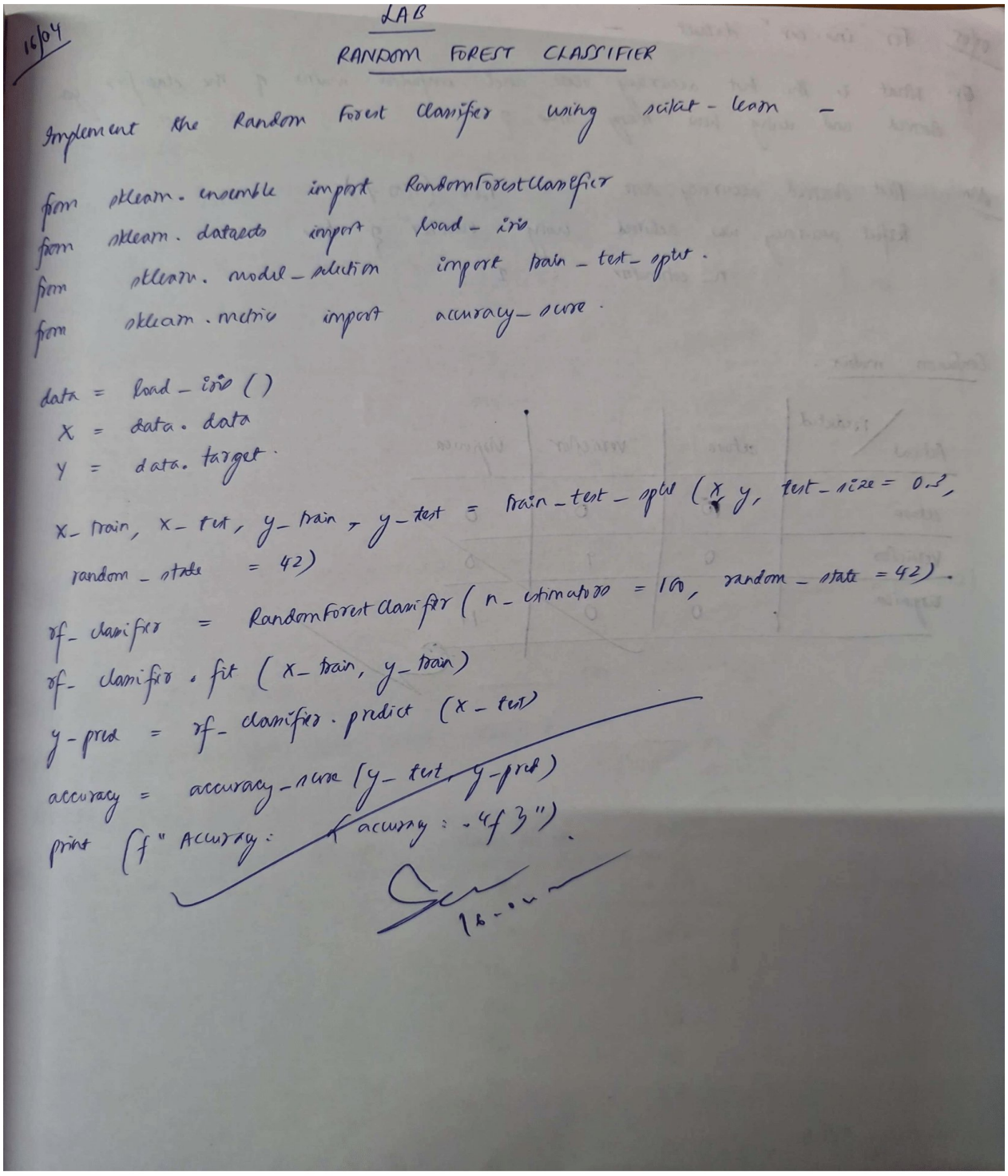
```
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title(f"Multi-Class ROC Curve (Macro AUC = {auc_score:.6f})")
plt.legend(loc="lower right", fontsize='small')
plt.grid()
plt.tight_layout()
plt.show()

print(f"Exact AUC Score = {auc_score}")
```

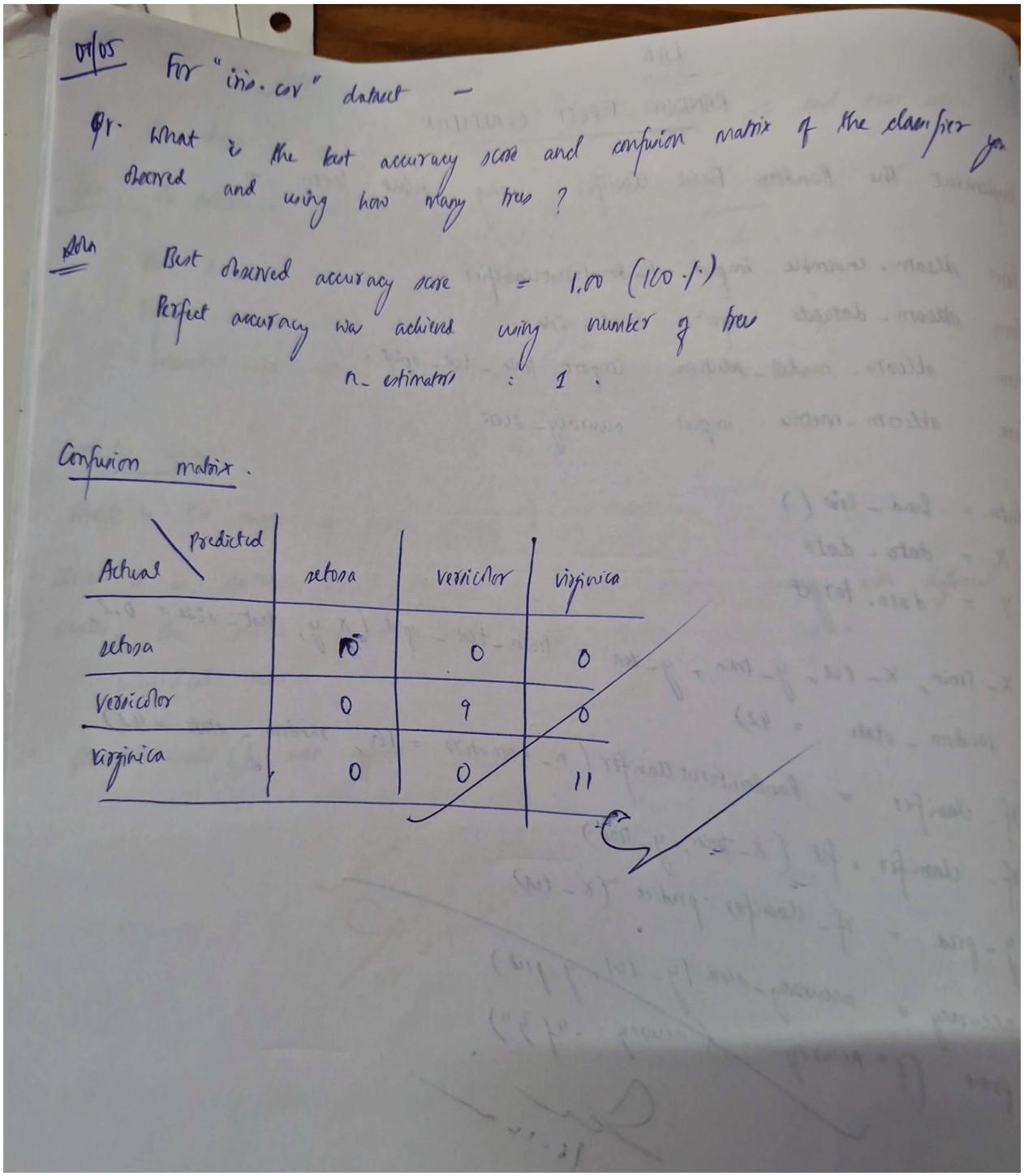


Program 8

Implement Random forest ensemble method on a given dataset.

Screenshot:





07/05 For "iris.csv" dataset –

Qr. What is the best accuracy score and confusion matrix of the classifier you observed and using how many trees ?

Ans Best observed accuracy score = 1.00 (100 %)

Perfect accuracy was achieved using number of trees

n_estimators : 1 .

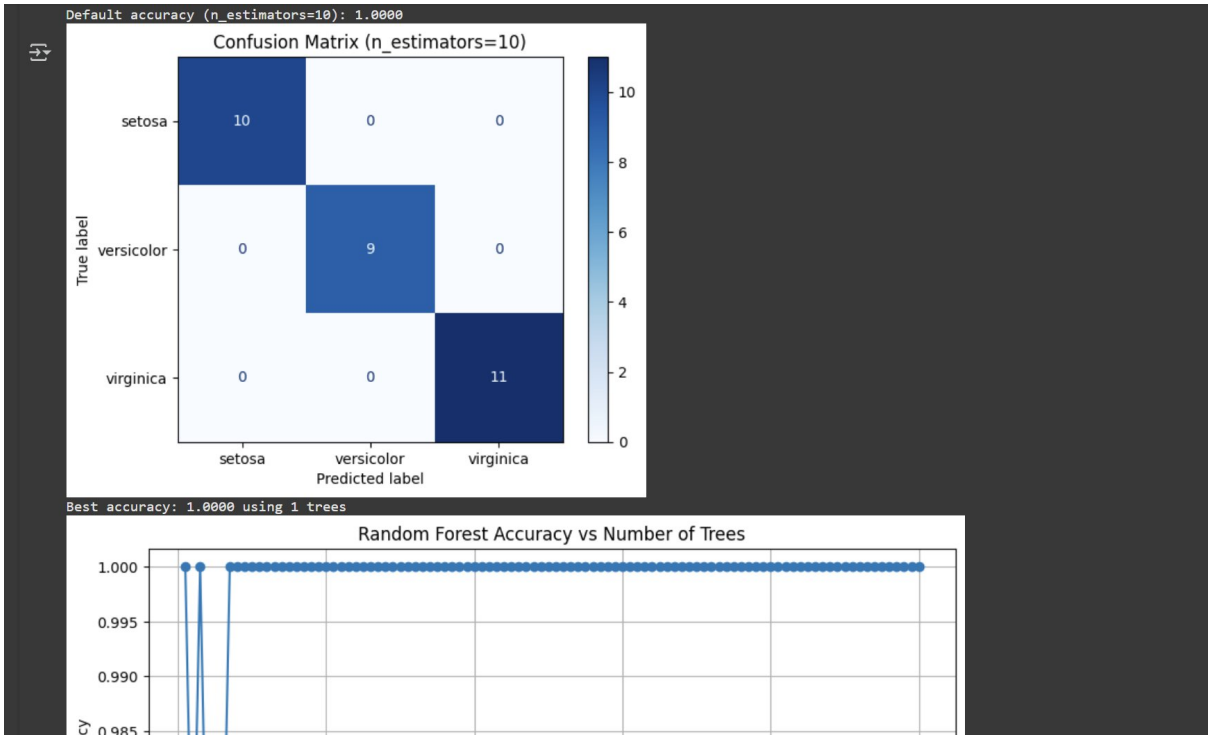
Confusion matrix.

Actual \ Predicted	setosa	versicolor	virginica
setosa	10	0	0
versicolor	0	9	0
virginica	0	0	11

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix

iris_df = pd.read_csv("iris.csv")
X = iris_df.drop('species', axis=1)
y = iris_df['species']
le = LabelEncoder()
y_encoded = le.fit_transform(y)

X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.3,
random_state=42)
rf_model = RandomForestClassifier(n_estimators=10, random_state=42)
rf_model.fit(X_train, y_train)
y_pred = rf_model.predict(X_test)
print("Random Forest Accuracy with 10 trees:", accuracy_score(y_test, y_pred))

scores = []
n_range = range(1, 101)
best_model = None
best_preds = None

for n in n_range:
    model = RandomForestClassifier(n_estimators=n, random_state=42)
    model.fit(X_train, y_train)
    preds = model.predict(X_test)
    acc = accuracy_score(y_test, preds)
    scores.append(acc)
```

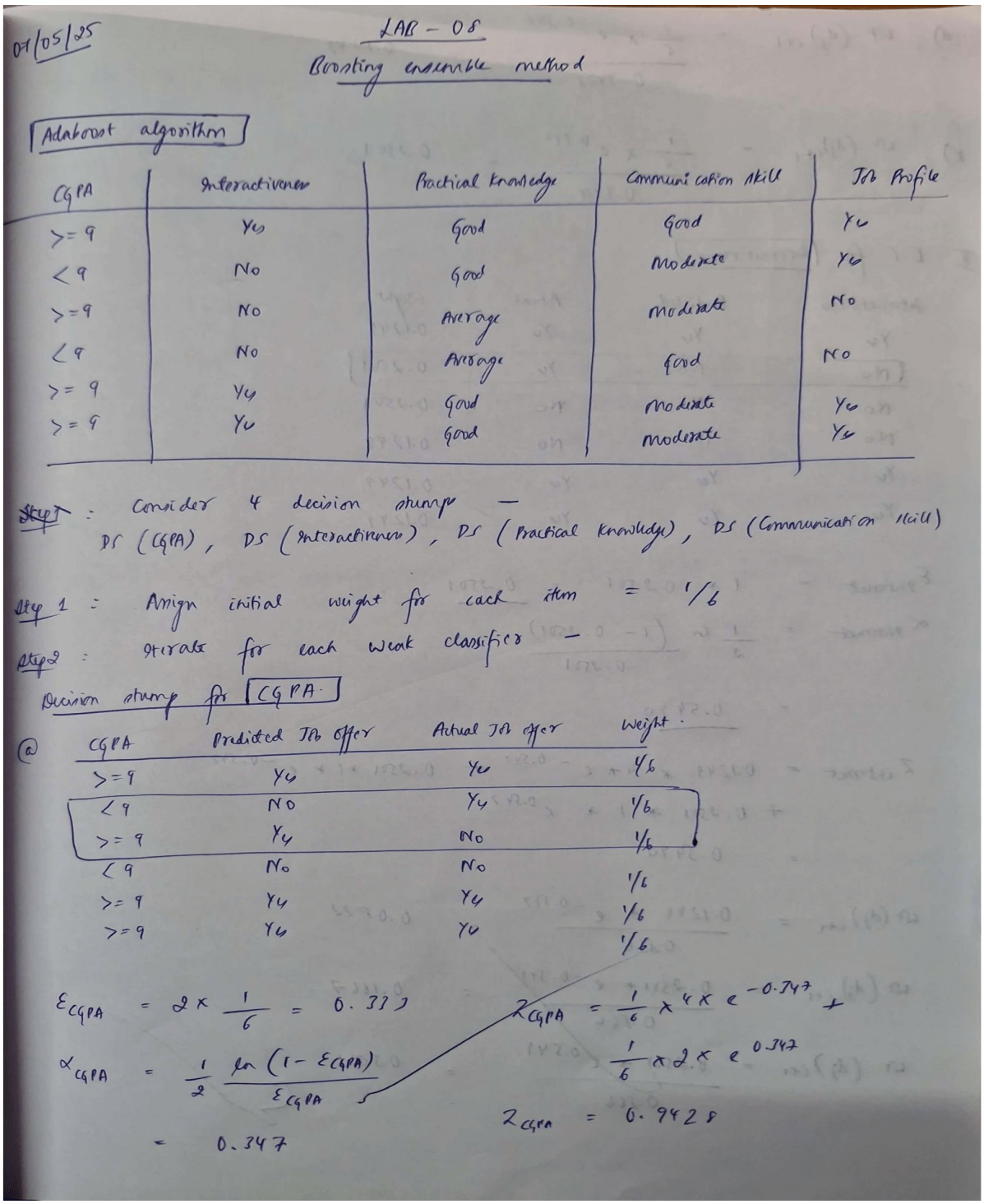
```
    if acc == max(scores):
        best_model = model
        best_preds = preds

best_score = max(scores)
best_n = n_range[scores.index(best_score)]
print(f"Best Random Forest Accuracy: {best_score:.4f} with {best_n} trees")

plt.figure(figsize=(10, 5))
plt.plot(n_range, scores, marker='o', linestyle='-', color='blue')
plt.title('Random Forest Accuracy vs Number of Trees (Iris Dataset)')
plt.xlabel('Number of Trees')
plt.ylabel('Accuracy')
plt.grid(True)
plt.show()

cm = confusion_matrix(y_test, best_preds)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=le.classes_,
yticklabels=le.classes_)
plt.title(f"Confusion Matrix for Best Random Forest Model ({best_n} Trees)")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```


Default accuracy (n_estimators=10): 1.0000
Confusion Matrix (n_estimators=10)
setosa
versicolor
virginica
True label
Predicted label
10
0
0
0
9
0
0
0
11
Best accuracy: 1.0000 using 1 trees
Random Forest Accuracy vs Number of Trees
1.000
0.995
0.990

Program 9

Implement Boosting ensemble method on a given dataset.

Screenshot:



07/05/25

LAB - 08

Boosting ensemble method

Adaboost algorithm

CGPA	Interactiveness	Practical knowledge	Communication skill	Job Profile
>= 9	Yes	Good	Good	Yes
< 9	No	Good	Moderate	Yes
>= 9	No	Average	Moderate	No
< 9	No	Average	Good	No
>= 9	Yes	Good	Moderate	Yes
>= 9	Yes	Good	Moderate	Yes

Step 1 : Consider 4 decision stumps —
DS (CGPA), DS (Interactiveness), DS (Practical knowledge), DS (Communication skill)

Step 1 : Assign initial weight for each item = 1/6

Step 2 : Iterate for each weak classifier —

Decision stump for CGPA

	CGPA	Predicted Job Offer	Actual Job Offer	Weight
①	>= 9	Yes	Yes	1/6
	< 9	No	Yes	1/6
	>= 9	Yes	No	1/6
	< 9	No	No	1/6
	>= 9	Yes	Yes	1/6
	>= 9	Yes	Yes	1/6

$\varepsilon_{CGPA} = 2 \times \frac{1}{6} = 0.333$

$\alpha_{CGPA} = \frac{1}{2} \frac{\ln(1 - \varepsilon_{CGPA})}{\varepsilon_{CGPA}}$

$= 0.347$

$Z_{CGPA} = \frac{1}{6} \times 4 \times e^{-0.347} + \frac{1}{6} \times 2 \times e^{0.347}$

$Z_{CGPA} = 0.9428$

d) $wt\ (d_j)_{i+1} = \frac{\frac{1}{6} \times e^{-0.347}}{0.9425} = 0.1249$

x) $wt\ (d_j)_{i+1} = \frac{\frac{1}{6} \times e^{0.347}}{0.9425} = 0.2501$

I D.S for Interactiveness

Interactiveness	Predicted	Actual	weight
Yes	Yes	Yes	0.1249
No	No	Yes	0.2501
No	No	No	0.2501
No	No	No	0.1249
Yes	Yes	Yes	0.1249
Yes	Yes	Yes	0.1249

$\varepsilon_{interact} = 1 * 0.2501 = 0.2501$

$\alpha_{interact} = \frac{1}{2} \ln \frac{(1 - 0.2501)}{0.2501}$

$= 0.5490$

$Z_{interact} = 0.1249 * 4 * e^{-0.549} + 0.2501 * 1 * e^{-0.549} + 0.2501 * 1 * e^{0.549}$

$= 0.5490$

$wt\ (d_j)_{i+1} = \frac{0.1249 * e^{-0.549}}{0.816} = 0.0832$

$wt\ (d_j)_{i+1} = \frac{0.2501 * e^{-0.549}}{0.816} = 0.1667$

$wt\ (d_j)_{i+1} = \frac{0.2501 * e^{0.549}}{0.816} = 0.5001$

III DS for practical knowledge

- No misclarification

IV DS for communication skills

V

$\alpha_{CGPA} = 0.547$	$\alpha_{interact} = 0.5470$	$\alpha_{comm} = -0.5455$	Final prediction
Yes	Yes	Yes	Yes
No	No	No	No
Yes	No	No	Yes
No	No	Yes	No
Yes	Yes	No	Yes
Yes	Yes	No	Yes.

Income.csv

Q1. What is the best accuracy score and confusion matrix of the classifier?

Ans Best accuracy score = 0.8140

No. of estimators = 73

Confusion matrix (n_estimators = 10)

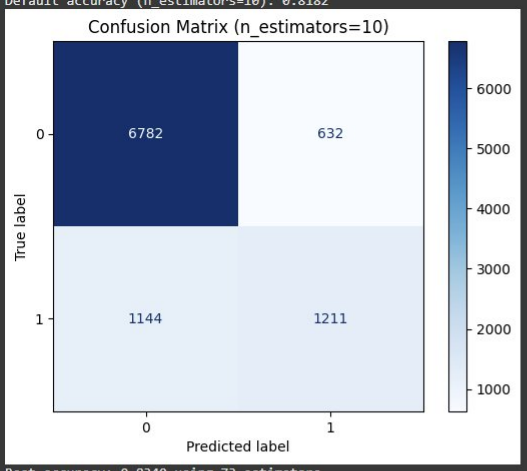
	0	1
0	6782	632
1	1144	1211

Code:

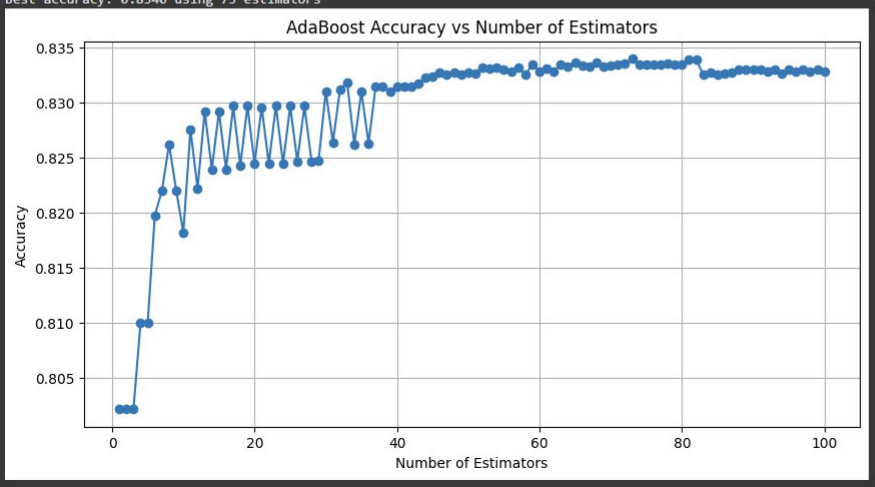
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics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

income_df = pd.read_csv("income.csv")
X_income = income_df.drop('income_level', axis=1)
y_income = income_df['income_level']
X_train_i, X_test_i, y_train_i, y_test_i = train_test_split(X_income, y_income, test_size=0.3,
random_state=42)

ada_model = AdaBoostClassifier(n_estimators=10, random_state=42)
ada_model.fit(X_train_i, y_train_i)
y_pred_i = ada_model.predict(X_test_i)
print("AdaBoost Accuracy with 10 estimators:", accuracy_score(y_test_i, y_pred_i))

scores_ada = []
n_range_ada = range(1, 51)
best_model_ada = None
best_preds_ada = None

for n in n_range_ada:
    model = AdaBoostClassifier(n_estimators=n, random_state=42)
    model.fit(X_train_i, y_train_i)
    preds = model.predict(X_test_i)
    acc = accuracy_score(y_test_i, preds)
    scores_ada.append(acc)
```

```
    if acc == max(scores_ada):
        best_model_ada = model
        best_preds_ada = preds

best_score_ada = max(scores_ada)
best_n_ada = n_range_ada[scores_ada.index(best_score_ada)]
print(f"Best AdaBoost Accuracy: {best_score_ada:.4f} with {best_n_ada} estimators")

plt.figure(figsize=(10, 5))
plt.plot(n_range_ada, scores_ada, marker='o', linestyle='-', color='orange')
plt.title('AdaBoost Accuracy vs Number of Estimators (Income Dataset)')
plt.xlabel('Number of Estimators')
plt.ylabel('Accuracy')
plt.grid(True)
plt.show()

cm_ada = confusion_matrix(y_test_i, best_preds_ada)
plt.figure(figsize=(6, 5))
sns.heatmap(cm_ada, annot=True, fmt='d', cmap='Oranges', xticklabels=[0, 1],
yticklabels=[0, 1])
plt.title(f"Confusion Matrix for Best AdaBoost Model ({best_n_ada} Estimators)")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

```
Default accuracy (n_estimators=10): 0.8182
```



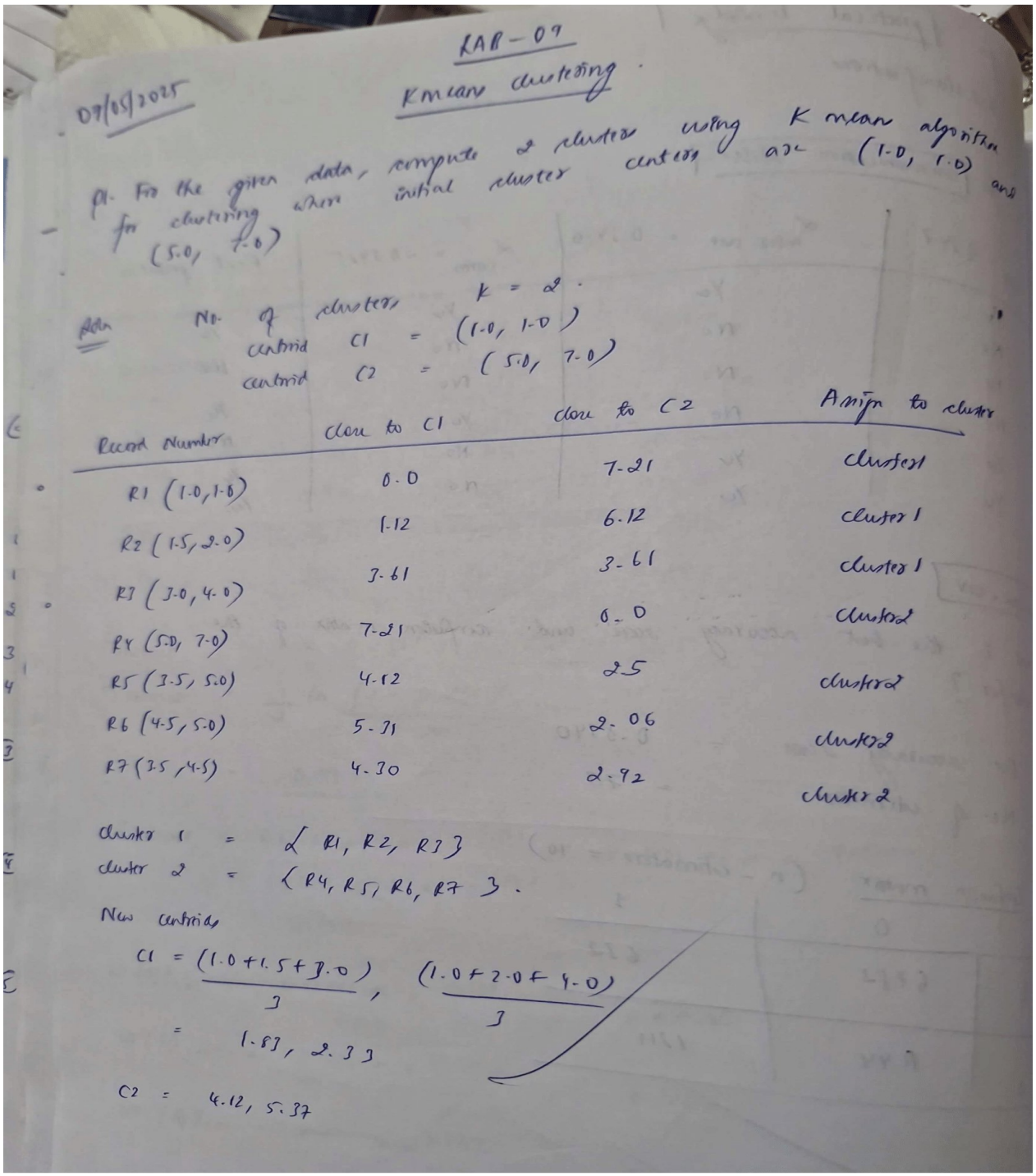

```
Best accuracy: 0.8340 using 73 estimators
```



Program 10

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

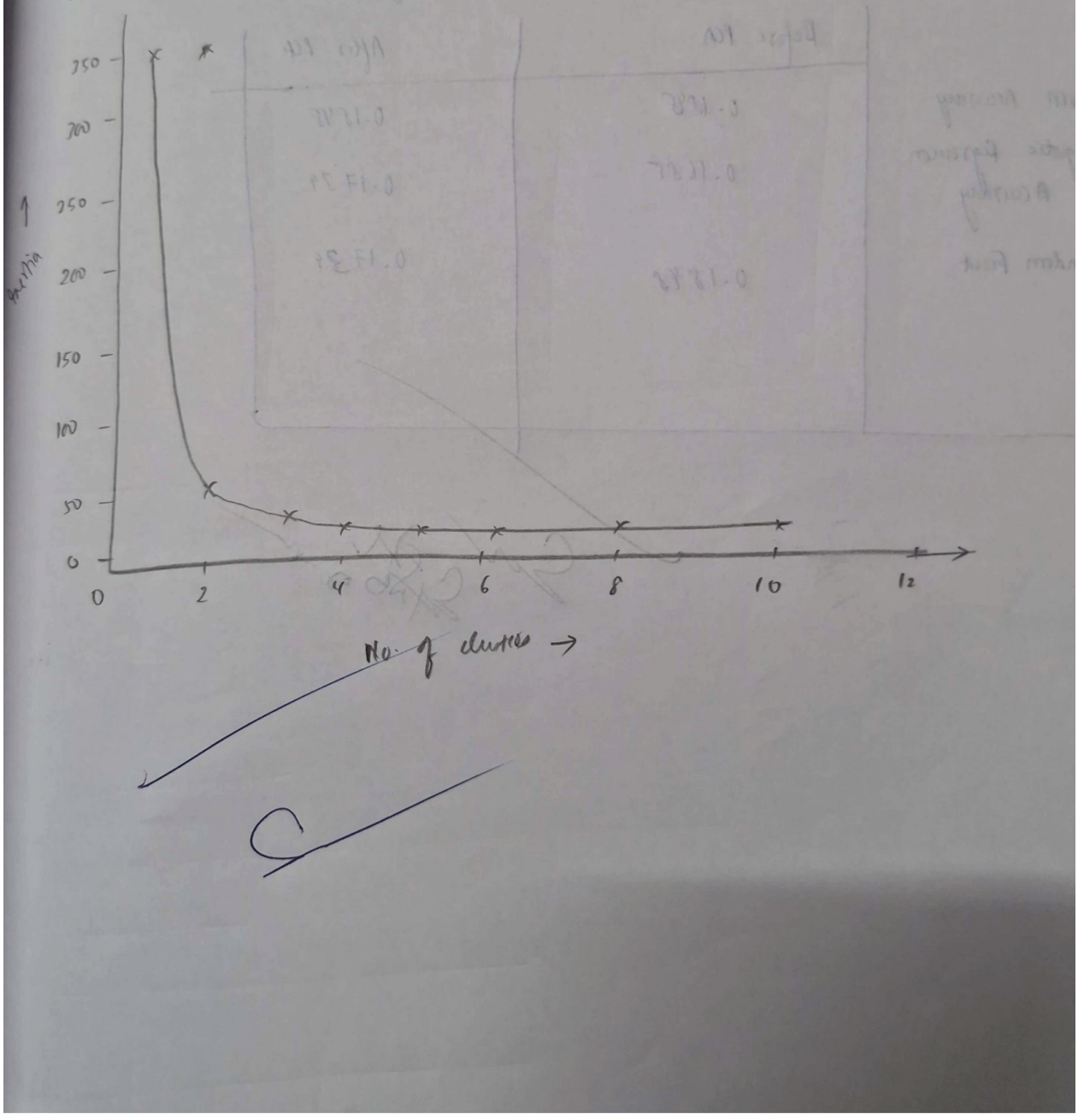
Screenshot:



Q.no. 10

Draw the elbow plot. What was the optimal k value obtained.

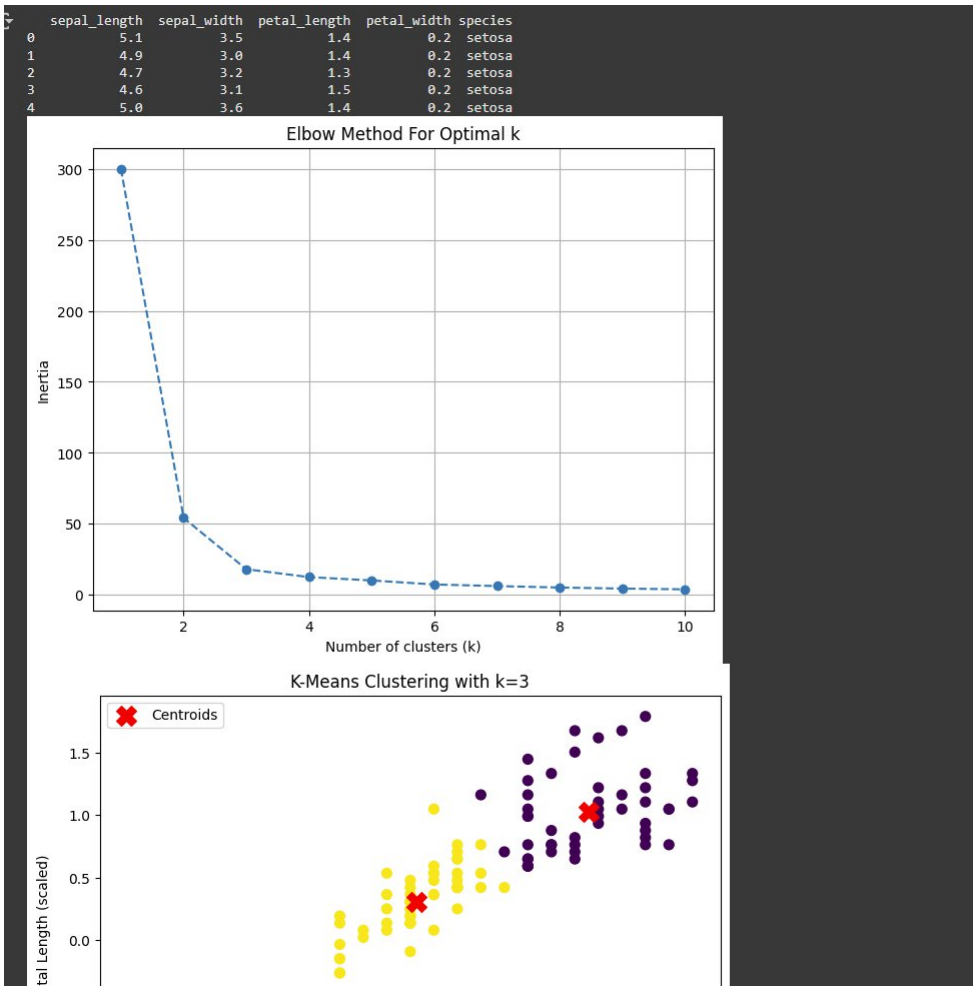
Optimal value of k = 3



Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from scipy import stats
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

df1=pd.read_csv("iris.csv")
df1.head()
df = df1.drop(['sepal_length','sepal_width','species'],axis=1)
scaler = StandardScaler()
scaled_df = scaler.fit_transform(df)
wcss = []
for i in range(1, 11):
kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10,
random_state=0)
kmeans.fit(scaled_df)
wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()

kmeans = KMeans(n_clusters=3, init='k-means++', max_iter=300, n_init=10,
random_state=0)
pred_y = kmeans.fit_predict(scaled_df)
df['cluster'] = pred_y
```

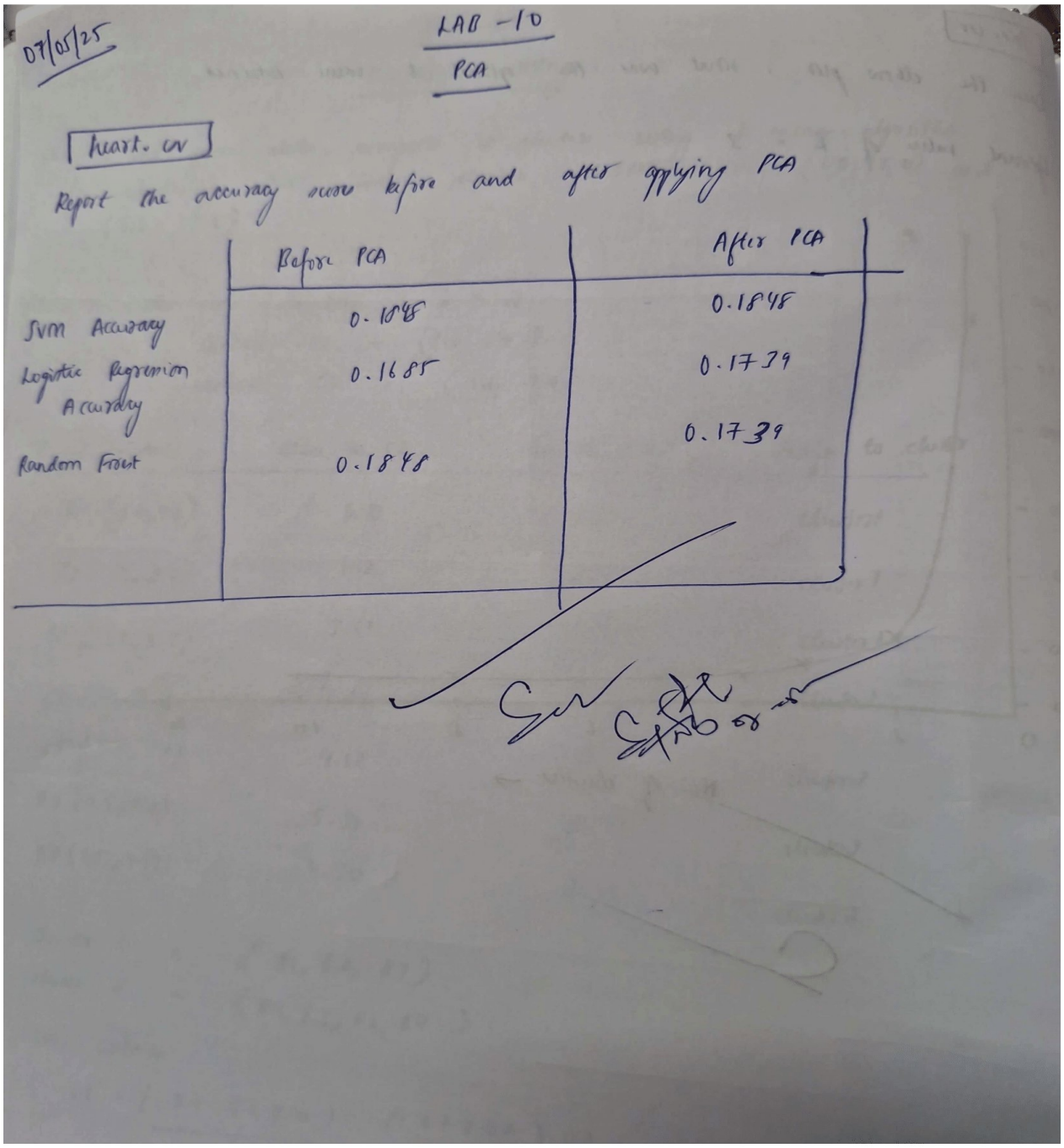
```
plt.scatter(df['petal_length'], df['petal_width'], c=df['cluster'])
plt.title('Clusters of Iris Flowers')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.show()
```



Program 11

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

Screenshot:



Code:

```
from google.colab import files
heart=files.upload()
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from scipy import stats
import seaborn as sns
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.decomposition import PCA

df1=pd.read_csv("heart.csv")
df1.head()
text_cols = df1.select_dtypes(include=['object']).columns
label_encoder = LabelEncoder()
for col in text_cols:
df1[col] = label_encoder.fit_transform(df1[col])
print(df1.head())
X = df1.drop('HeartDisease', axis=1)
y = df1['HeartDisease']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
# Support Vector Machine
svm_model = SVC(kernel='linear', random_state=42)
svm_model.fit(X_train, y_train)
svm_predictions = svm_model.predict(X_test)
svm_accuracy = accuracy_score(y_test, svm_predictions)
print(f"SVM Accuracy: {svm_accuracy}")

# Logistic Regression
lr_model = LogisticRegression(random_state=42)
lr_model.fit(X_train, y_train)
lr_predictions = lr_model.predict(X_test)
lr_accuracy = accuracy_score(y_test, lr_predictions)
print(f"Logistic Regression Accuracy: {lr_accuracy}")

# Random Forest
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)
rf_predictions = rf_model.predict(X_test)
rf_accuracy = accuracy_score(y_test, rf_predictions)
print(f"Random Forest Accuracy: {rf_accuracy}")

models = {
"SVM": svm_accuracy,
"Logistic Regression": lr_accuracy,
"Random Forest": rf_accuracy}
best_model = max(models, key=models.get)
print(f"\nBest Model: {best_model} with accuracy {models[best_model]}")

pca = PCA(n_components=0.95)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)
svm_model_pca = SVC(kernel='linear', random_state=42)
svm_model_pca.fit(X_train_pca, y_train)
svm_predictions_pca = svm_model_pca.predict(X_test_pca)
```

```
svm_accuracy_pca = accuracy_score(y_test, svm_predictions_pca)
print(f"SVM Accuracy (with PCA): {svm_accuracy_pca}")

lr_model_pca = LogisticRegression(random_state=42)
lr_model_pca.fit(X_train_pca, y_train)
lr_predictions_pca = lr_model_pca.predict(X_test_pca)
lr_accuracy_pca = accuracy_score(y_test, lr_predictions_pca)
print(f"Logistic Regression Accuracy (with PCA): {lr_accuracy_pca}")

rf_model_pca = RandomForestClassifier(random_state=42)
rf_model_pca.fit(X_train_pca, y_train)
rf_predictions_pca = rf_model_pca.predict(X_test_pca)
rf_accuracy_pca = accuracy_score(y_test, rf_predictions_pca)
print(f"Random Forest Accuracy (with PCA): {rf_accuracy_pca}")

models_pca = {
"SVM": svm_accuracy_pca,
"Logistic Regression": lr_accuracy_pca,
"Random Forest": rf_accuracy_pca}
best_model_pca = max(models_pca, key=models_pca.get)
print(f"\nBest Model (with PCA): {best_model_pca} with accuracy
{models_pca[best_model_pca]}")
```

```
   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
SVM Accuracy: 0.1848
Logistic Regression Accuracy: 0.1685
Random Forest Accuracy: 0.1848
SVM Accuracy with PCA: 0.1848
Logistic Regression Accuracy with PCA: 0.1739
Random Forest Accuracy with PCA: 0.1739

--- Model Performance Comparison ---
SVM Accuracy without PCA: 0.1848
SVM Accuracy with PCA: 0.1848
Logistic Regression Accuracy without PCA: 0.1685
Logistic Regression Accuracy with PCA: 0.1739
Random Forest Accuracy without PCA: 0.1848
Random Forest Accuracy with PCA: 0.1739
```