VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

MOHAMMED SHURAIM (1BM22CS158)

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 **MOHAMMED SHURAIM (1BM22CS158),** 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.

Amruta B Assistant Professor Department of CSE, BMSCE	Dr. Kavitha Sooda Professor & HOD Department of CSE, BMSCE

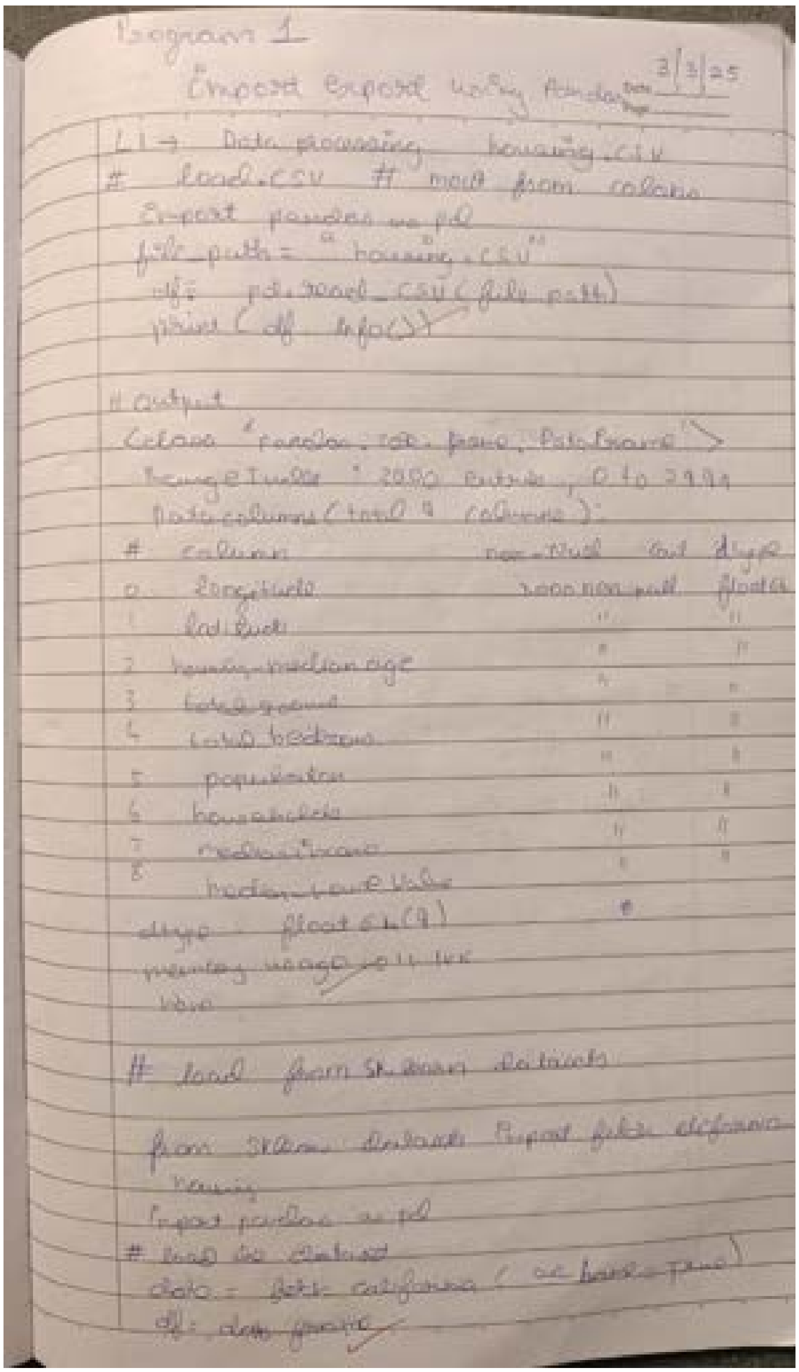
Index

Github Link:
https://github.com/Mohammed-Shuraim/6thSem-ML-Lab_1BM22CS158

Program 1

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

Screenshot



Program 1
Import export using Pandas 3/3/25

L1 → Data processing housing.csv
# load.csv # read from colab
import pandas as pd
file_path = "housing.csv"
df = pd.read_csv(file_path)
print(df.info())

output
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 2999
Data columns (total 9 columns):

#	column	non-null count	dtype
0	longitude	2000 non-null	float64
1	latitude	"	"
2	housing_median_age	"	"
3	total_rooms	"	"
4	total_bedrooms	"	"
5	population	"	"
6	households	"	"
7	median_income	"	"
8	median_house_value	"	

dtype: float64(9)
memory usage: [illegible] KB
None

load from sklearn datasets

from sklearn.datasets import fetch_california_housing
import pandas as pd
# load the dataset
data = fetch_california(as_frame=True)
df = data.frame

# Display information of all columns
print(df.info())

# display statistical info of all numerical columns
print(df.describe())

[illegible] check if ocean proximity [illegible] (converting [illegible])

column_name = "ocean proximity" if "ocean proximity" in df.columns else "ocean proximity"

Display the count of unique labels for ocean proximity column

print("\n Ocean proximity value counts:")
print(df[column_name].value_counts())

display which attributes has missing values
missing value count greater than 0

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

print("\n Missing values detected")
if missing_values.empty:
    print("No missing")
else:
    print(missing_values)

(3) What is the difference between min-max scaling and standardization? When would you use 1 over the other?

Min-max scaling scales the data to a specific range (usually 0 to 1). It is sensitive to outliers.

Min-max scaling is preferred when the algorithm is sensitive to the magnitude of the features (eg KNN, k-means) and when you know the data's distribution doesn't have significant outliers.

Standardization is generally preferred if the data has a Gaussian like distribution or when the algorithm is not sensitive to the feature scales (eg linear regression, logistic regression). Use it when you're concerned about outliers affecting the scaling. Standardization is often more [illegible]

Code:

```
import pandas as pd

import numpy as np

from sklearn.preprocessing import MinMaxScaler, StandardScaler, LabelEncoder,
OneHotEncoder

import seaborn as sns

import matplotlib.pyplot as plt

diabetes = pd.read_csv("/content/drive/MyDrive/MLlab dataset/Dataset of Diabetes
.csv")

# Update filename accordingly

# Load Adult Income dataset

adult_income = pd.read_csv("/content/drive/MyDrive/MLlab dataset/housing.csv")

# Update filename accordingly

# 1. Data Cleaning

## Identify numeric and categorical columns

numeric_cols = diabetes.select_dtypes(include=['number']).columns

categorical_cols = diabetes.select_dtypes(include=['object']).columns

## Handling Missing Values

diabetes[numeric_cols] =
diabetes[numeric_cols].fillna(diabetes[numeric_cols].mean())

diabetes[categorical_cols] =
diabetes[categorical_cols].fillna(diabetes[categorical_cols].mode().iloc[0])

adult_income.dropna(inplace=True)  # Drop missing values in Adult dataset

## Handling Categorical Data

categorical_columns = adult_income.select_dtypes(include=['object']).columns

label_encoders = {}

for col in categorical_columns:
    le = LabelEncoder()
    adult_income[col] = le.fit_transform(adult_income[col])
    label_encoders[col] = le  # Store encoder for inverse transform if needed

## Handling Outliers using IQR method

def remove_outliers(df, column):
    Q1 = df[column].quantile(0.25)
```

```
        Q3 = df[column].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR
        df[column] = np.where(df[column] < lower_bound, lower_bound, df[column])
        df[column] = np.where(df[column] > upper_bound, upper_bound, df[column])
        return df[column]  # Return only the processed column
    # Apply outlier removal only to numeric columns
    for col in numeric_cols:
        diabetes[col] = remove_outliers(diabetes, col)
    # 2. Data Transformations
    ## Min-Max Scaling
    minmax_scaler = MinMaxScaler()
    diabetes_scaled =
    pd.DataFrame(minmax_scaler.fit_transform(diabetes[numeric_cols]),
    columns=numeric_cols)
    adult_income_scaled = pd.DataFrame(minmax_scaler.fit_transform(adult_income),
    columns=adult_income.columns)
    ## Standardization
    standard_scaler = StandardScaler()
    diabetes_standardized =
    pd.DataFrame(standard_scaler.fit_transform(diabetes[numeric_cols]),
    columns=numeric_cols)
    adult_income_standardized =
    pd.DataFrame(standard_scaler.fit_transform(adult_income),
    columns=adult_income.columns)
    # Save processed datasets
    diabetes_scaled.to_csv("diabetes_preprocessed.csv", index=False)
    adult_income_scaled.to_csv("adult_income_preprocessed.csv", index=False)
    print("Data preprocessing completed.")
```

Show Missing Values Before & After Handling

```
# Before handling missing values
```

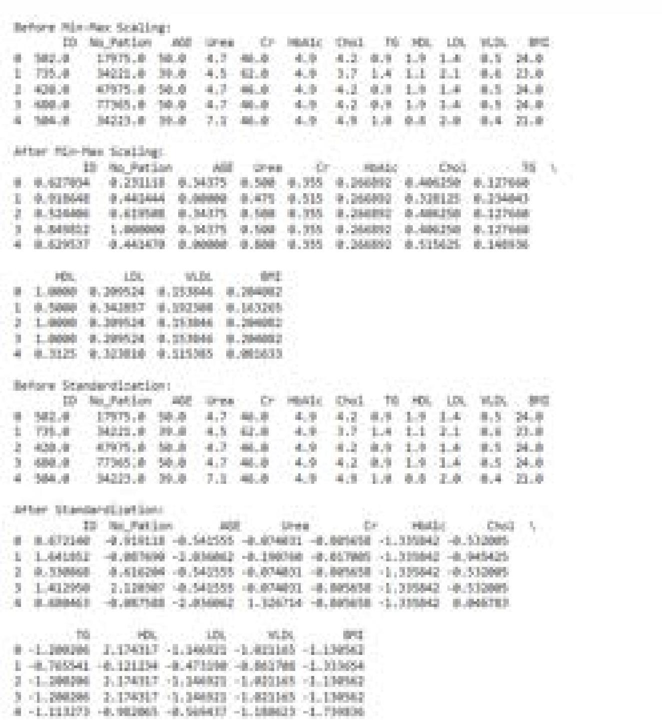
```python
print("Missing values before handling:")
print("Diabetes Dataset:\n", diabetes.isnull().sum())
print("\nAdult Income Dataset:\n", adult_income.isnull().sum())
# After handling missing values
print("\nMissing values after handling:")
print("Diabetes Dataset:\n", diabetes.isnull().sum())
print("\nAdult Income Dataset:\n", adult_income.isnull().sum())
```

Show Categorical Encoding

```python
print("Categorical Columns in Adult Income Dataset:\n", categorical_columns)
print("\nEncoded Example:\n", adult_income.head())
```

Before & After Scaling (Min-Max & Standardization)

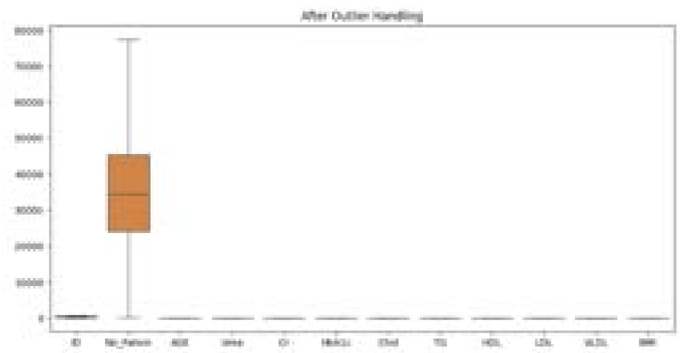
```python
print("Before Min-Max Scaling:\n", diabetes[numeric_cols].head())
print("\nAfter Min-Max Scaling:\n", diabetes_scaled.head())
print("\nBefore Standardization:\n", diabetes[numeric_cols].head())
print("\nAfter Standardization:\n", diabetes_standardized.head())
```



Plot Outliers Before & After Removal

```python
plt.figure(figsize=(12,6))
sns.boxplot(data=diabetes[numeric_cols])
```

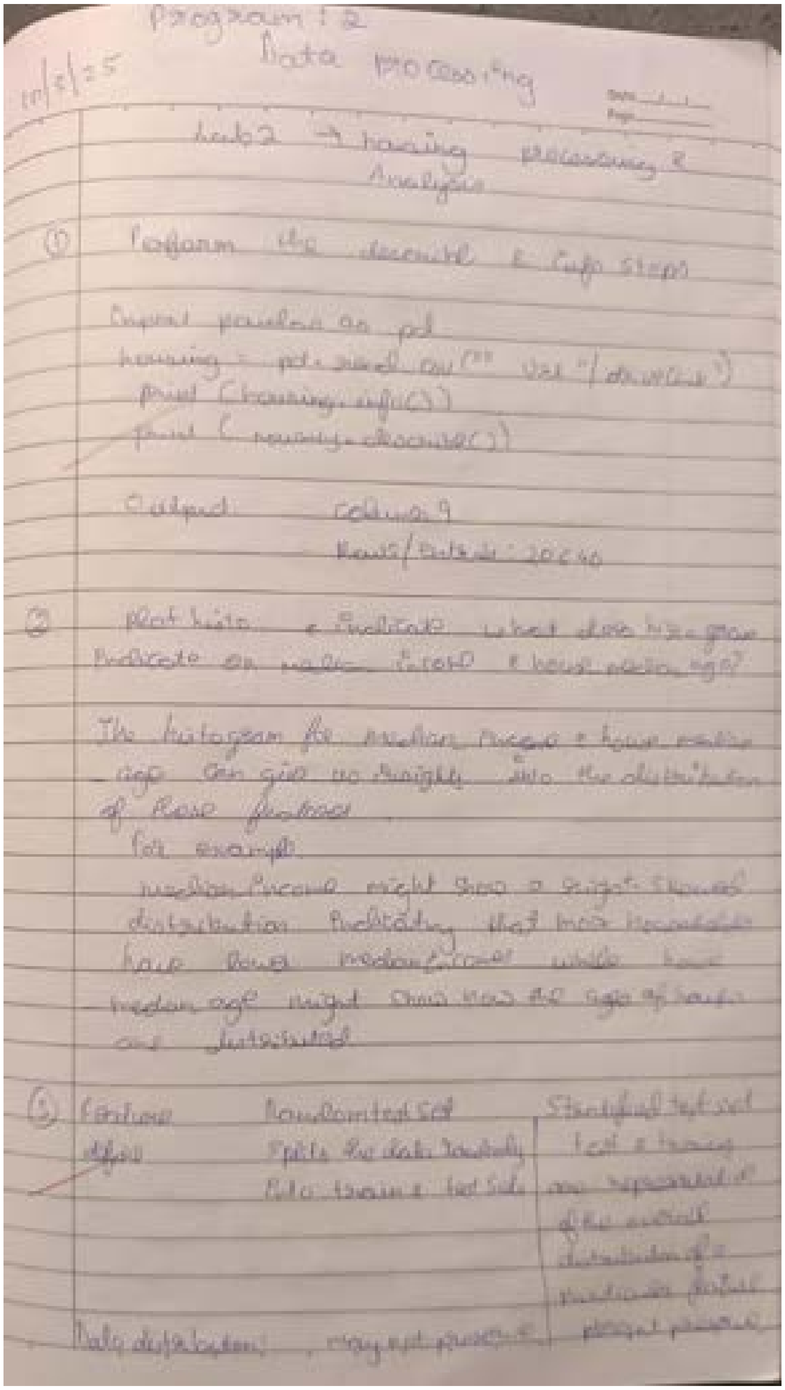
```python
plt.title("Before Outlier Handling")

plt.show()

# Boxplot after outlier handling

plt.figure(figsize=(12,6))

sns.boxplot(data=diabetes[numeric_cols])

plt.title("After Outlier Handling")

plt.show()
```



Program 2

Demonstrate various data pre-processing techniques for a given dataset

Screenshot



Program : 2
Data processing

10/2/25

Lab2 → housing processing & Analysis

(1) Perform the describe & info steps

```
import pandas as pd
housing = pd.read_csv("[illegible]")
print(housing.info())
print(housing.describe())
```

Output: columns 9
Rows/Entries: 20640

(2) Plot histo & indicate what does histogram indicate on median income & house median age?

The histogram for median income & house median age can give us insights into the distribution of these features.
For example
median income might show a right skewed distribution indicating that most households have lower median income while house median age might show how the age of houses are distributed

(3) Feature diff	Randomized set	Stratified test set
	Splits the data randomly into train & test sets	test & training are representative of the overall distribution of a particular feature
Data distribution	may not preserve	[illegible]

(4) The graph shows the variation in their geographic location
[illegible] [illegible]
size of [illegible] [illegible]

(6) Which feature correlates to the maximum?

median income usually shows the highest correlation with the median house value

(5) List the features that could be combined to improve correlation?

(rooms per household & bedrooms per room
& [illegible] [illegible]
[illegible] [illegible]
feature the median house value is [illegible]

(7) Data for the feature [illegible] needs to be cleaned & [illegible] cleaning [illegible]
total bedrooms

(8) Is there any categorical data, if so what is the method to be used
yes, ocean proximity is a categorical feature

(9) [illegible] method to convert to numerical

(10) Important scaling feature
to ensure all features contribute equally to the result

Technique: Standard Scaler & min max scaler

Code:

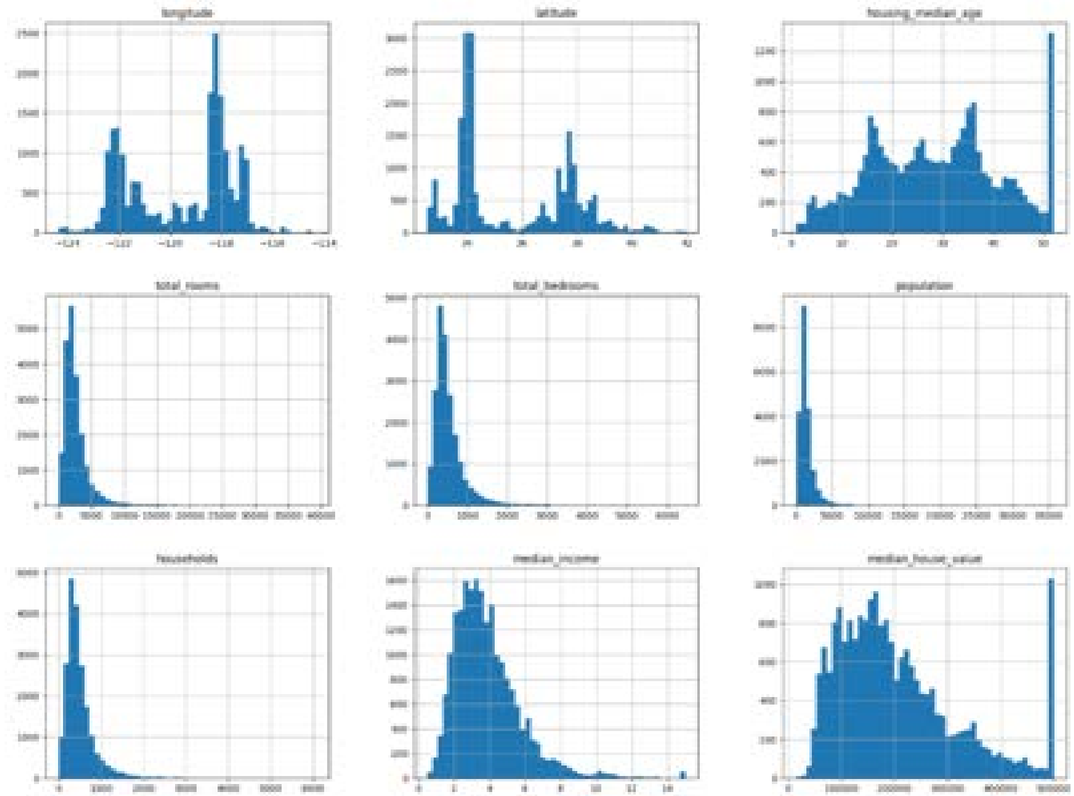
Perform the describe and info steps

```python
import pandas as pd
housing = pd.read_csv("/content/drive/MyDrive/MLlab dataset/housing10112025.csv")
   print(housing.info())
   print(housing.describe())
```

```
<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
          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
```

2)Plot the histogram of each feature(Indicate what does histogram indicate on median_income and house_median_age)

```python
import matplotlib.pyplot as plt
# Plot histogram for each feature
housing.hist(bins=50, figsize=(20, 15))
plt.show()
```

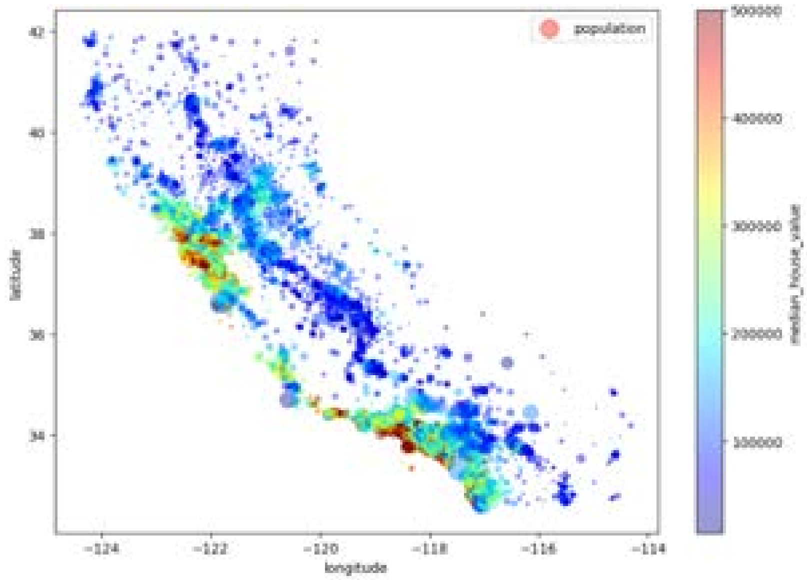


Demonstrate the process of creating a test set & random sampling

```
from sklearn.model_selection import train_test_split
train_set, test_set = train_test_split(housing, test_size=0.2, random_state=42)
print(len(train_set))
print(len(test_set))
16512 4128
```

List the geographical features from the dataset and plot a graph to Visualize Geographical Data(what does the graph indicate w.r.t housing prices and location)

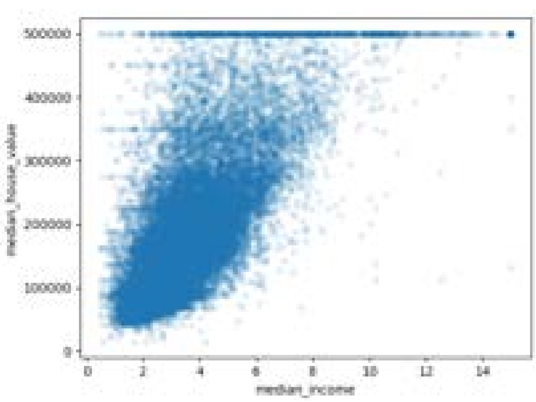
```
housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.4,
             s=housing["population"]/100, label="population", figsize=(10,7),
             c="median_house_value", cmap=plt.get_cmap("jet"), colorbar=True,
             sharex=False)
plt.legend()
plt.show()
```



This graph visualizes housing prices in relation to their geographic locations, with the color representing the median house value and the size of the circle representing the population.

Plot a graph to show features correlation with housing price. Which feature corelates to the maximum. Plot the graph for that with housing price and analyze what the graph indicate

```
# Drop the 'ocean_proximity' column
housing_num = housing.drop("ocean_proximity", axis=1)
# Calculate and print correlation matrix
corr_matrix = housing_num.corr()
print(corr_matrix["median_house_value"].sort_values(ascending=False))
# Plot the correlation between 'median_income' and 'median_house_value'
housing.plot(kind="scatter", x="median_income", y="median_house_value", alpha=0.1)
plt.show()
```



median_income usually shows the highest correlation with the median house value.

List the features that needs to be cleaned and demonstrate the process of cleaning

```
print(housing.columns)
```

```python
if "total_bedrooms" in housing.columns:
    # Option 1: Drop rows with missing values in 'total_bedrooms'
    housing.dropna(subset=["total_bedrooms"], inplace=True)
    # Option 2: Drop the 'total_bedrooms' column entirely
    housing.drop("total_bedrooms", axis=1, inplace=True)
    # Option 3: Fill missing values in 'total_bedrooms' with the median value
    housing["total_bedrooms"].fillna(housing["total_bedrooms"].median(),
inplace=True)
else:
    print("Column 'total_bedrooms' not found in the dataset.")
    Index(['longitude', 'latitude', 'housing_median_age', 'total_rooms',
       'population', 'households', 'median_income', 'median_house_value',
       'ocean_proximity', 'income_cat', 'rooms_per_household',
       'bedrooms_per_room'],
      dtype='object')
Column 'total_bedrooms' not found in the dataset.
```

Output the Corrected Data

```python
print(housing.head())
```

```
   longitude  latitude  housing_median_age  total_rooms  population  \
0    -122.23     37.88                41.0        880.0       322.0
1    -122.22     37.86                21.0       7099.0      2401.0
2    -122.24     37.85                52.0       1467.0       496.0
3    -122.25     37.85                52.0       1274.0       558.0
4    -122.25     37.85                52.0       1627.0       565.0

   households  median_income  median_house_value ocean_proximity income_cat  \
0       126.0         8.3252            452600.0        NEAR BAY          5
1      1138.0         8.3014            358500.0        NEAR BAY          5
2       177.0         7.2574            352100.0        NEAR BAY          5
3       219.0         5.6431            341300.0        NEAR BAY          4
4       259.0         3.8462            342200.0        NEAR BAY          3

   rooms_per_household  bedrooms_per_room
0             6.984127           0.146591
1             6.238137           0.155797
2             8.288136           0.129516
3             5.817352           0.184458
4             6.281853           0.172096
```

Convert Categorical Data to Numerical

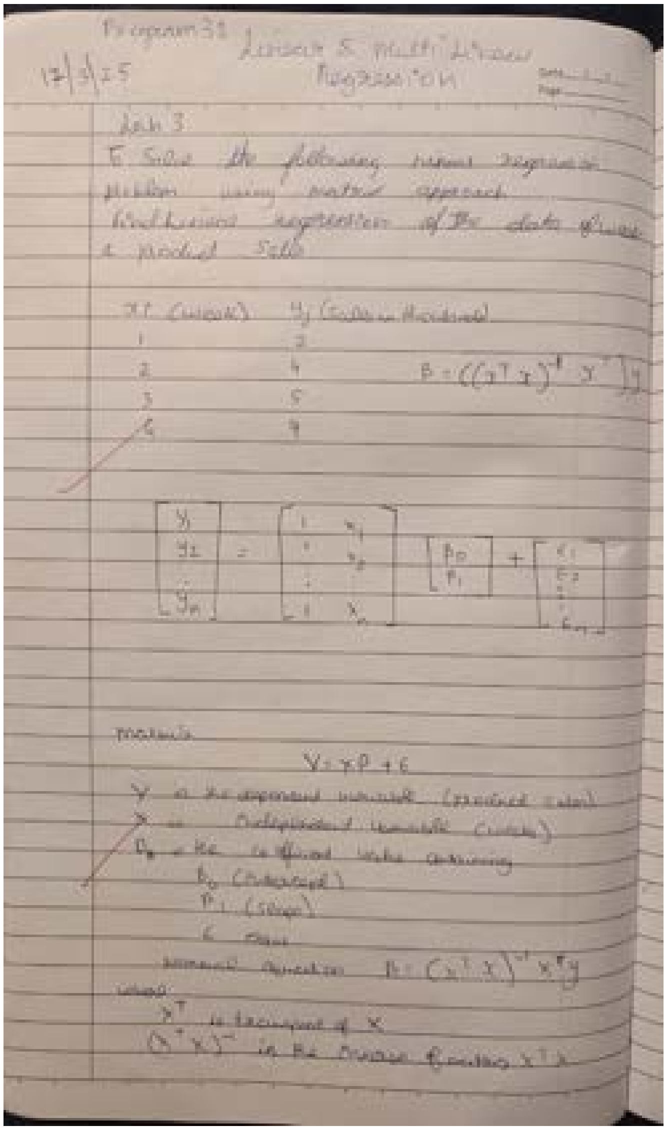
```python
from sklearn.preprocessing import OneHotEncoder
housing_cat = housing[["ocean_proximity"]]
encoder = OneHotEncoder()
housing_cat_1hot = encoder.fit_transform(housing_cat)
print(housing_cat_1hot.toarray())
```

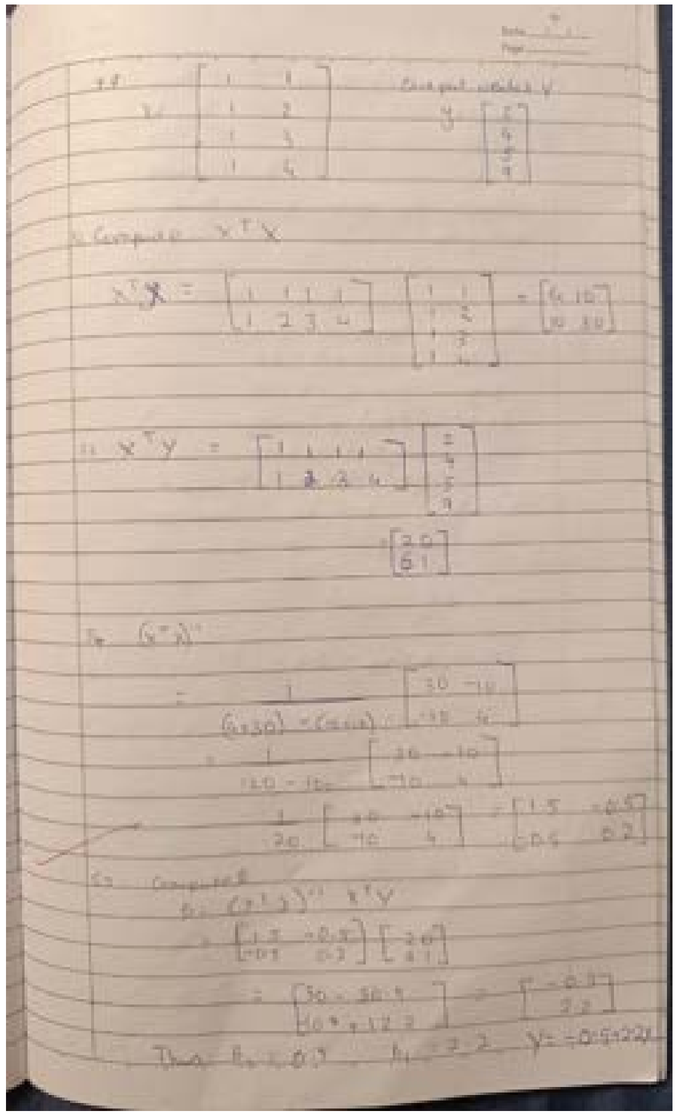
```
[[0. 0. 0. 1. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 1. 0.]
 ...
 [0. 1. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 1. 0. 0. 0.]]
```

Program 3

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Screenshot





Diameter (x) inches | Price (y) in dollars

8	10
10	13
12	16

predict the price of 20 inch pizza using the data given

We need price when $x = 20$

compute required sums

$\Sigma x = 8 + 10 + 12 = 30$

$\Sigma y = 10 + 13 + 16 = 39$

$\Sigma xy = (8 \times 10) + (10 \times 13) + (12 \times 16)$

$= 80 + 130 + 192 = 402$

$\Sigma x^2 = (8)^2 + (10)^2 + (12)^2$

$= 64 + 100 + 144 = 308$

$n = 3$

Compute B_1 (slope)

$B_1 = (n \Sigma xy - \Sigma x \Sigma y) / n(\Sigma x^2 - (\Sigma x)^2)$

$= (3 \times 402) - (30 \times 39) / (3 \times 308) - 30^2$

$1206 - 1170 / 924 - 900$

$= 36/24 = 1.5$

Compute B_0 (intercept)

$B_0 = \Sigma y - B_1 \Sigma x / n$

$39 - (1.5 \times 30) / 3$

$39 - 45 / 3 = -6/3 = -2$

Date: 15
Page:

Formulate the Regression Equation

$y = B_0 + B_1 x$

$= -2 + 1.5x$

predict price for $x = 20$

$y = -2 + (1.5 \times 20)$

$y = -2 + 30$

$y = 28$

Price for 20 inch pizza \$28

Observation

① Considering the 3 data csv files (companies.csv, Salary.csv, [illegible].csv) did you perform any data preprocessing steps?

yes, I handled missing values by filling them with the column mean. I also applied label encoding to categorical columns (like "State") & scaled numerical features for [illegible].csv to normalize the data.

② Did you visualize the regression for [illegible].csv?

yes, the regression line was plotted. The plot shows a strong linear relationship between [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]

③ predicted salary for [illegible] years experience [illegible] [illegible] [illegible]?

[illegible] [illegible]

The predicted salary is [illegible] with the target salary with [illegible] [illegible]

④ Did you create categorical variables for [illegible] companies.csv?

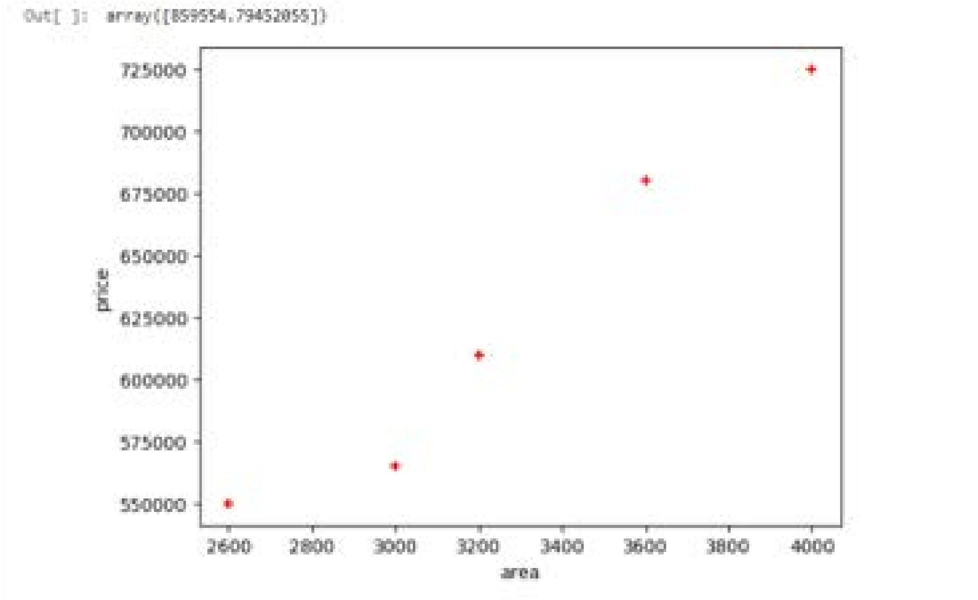
yes the "State" column was [illegible] using label encoder.

Did you scale the features?

yes [illegible] R&D spend, [illegible] & marketing spend [illegible] [illegible] [illegible], feature scaling [illegible] [illegible] applied to improve performance.

Linear Regression

```python
import pandas as pd

import numpy as np

from sklearn import linear_model

import matplotlib.pyplot as plt

df = pd.read_csv('/content/drive/MyDrive/ML Lab 3/housing_area_price.csv')

df

# 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]])

reg.coef_

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"""

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




```
import pandas as pd

import numpy as np

from sklearn import linear_model

df = pd.read_csv('/content/drive/MyDrive/ML Lab 3/homeprices_Multiple_LR.csv')

df

"""Data Preprocessing: Fill NA values with median value of a column"""

df.bedrooms.median()

df.bedrooms = df.bedrooms.fillna(df.bedrooms.median())

df

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


498408.25157402386
```

Write Python code to implement the following Considering the data file hiring.csv. The file contains hiring statics for a firm such as experience of candidate, his written test score and personal interview score. Based on these 3 factors, HR will decide the salary. Given this data, you need to build a Multiple Linear Regression model for HR department that can help them decide salaries for future candidates. Using this predict salaries for following candidates, 2 yr experience, 9 test score, 6 interview score 12 yr experience, 10 test score, 10 interview score Considering the data file 1000_companies.csv. The file contains profit statics for a firm such as R&D Spend, Administration, Marketing Spend and State. Based on these four factors build a Multiple Linear Regression model to predict the profit. Using this predict profit for following, 91694.48 R&D Spend, 515841.3 Administration, 11931.24 Marketing Spend, Florida State

Note: If required, apply the necessary data processing steps to data files.

```
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 LabelEncoder, StandardScaler

# Function to process data and train model

def train_model(df, target_column, categorical_columns=None, scale_features=False):

    # Convert numeric columns to proper type

    for col in df.columns:

        df[col] = pd.to_numeric(df[col], errors='coerce')  # Converts non-numeric text to NaN

 # Handle missing values

    df.fillna(df.mean(), inplace=True)



    # Encode categorical data

    label_encoders = {}

    if categorical_columns:

        for col in categorical_columns:

            le = LabelEncoder()

            df[col] = le.fit_transform(df[col].astype(str))  # Ensure categorical data is string

            label_encoders[col] = le

  # Splitting features and target
```

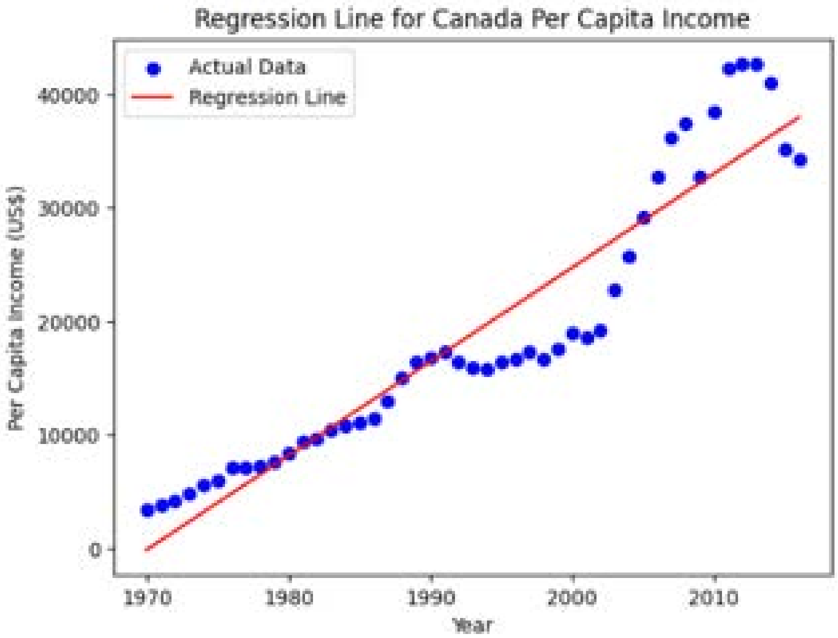
```python
    X = df.drop(columns=[target_column])

    y = df[target_column]

# Feature scaling if required

    scaler = None

    if scale_features:

        scaler = StandardScaler()

        X = scaler.fit_transform(X)

# Train-test split

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

 # Train model

    model = LinearRegression()

    model.fit(X_train, y_train)

  return model, label_encoders, scaler

# Load datasets

hiring_df = pd.read_csv("/content/drive/MyDrive/ML Lab 3/hiring.csv")

companies_df = pd.read_csv("/content/drive/MyDrive/ML Lab 3/1000_Companies.csv")

canada_df = pd.read_csv("/content/drive/MyDrive/ML Lab 
3/canada_per_capita_income.csv")

# Print column names for debugging

print("Hiring CSV Columns:", hiring_df.columns)

print("Companies CSV Columns:", companies_df.columns)

print("Canada CSV Columns:", canada_df.columns)

# Fix column name mismatches

salary_col = "salary($)"

hiring_df.rename(columns={"test_score(out of 10)": "test_score", 
"interview_score(out of 10)": "interview_score"}, inplace=True)

# Fix experience column in hiring dataset (convert text to numbers)

if 'experience' in hiring_df.columns:

    hiring_df['experience'] = hiring_df['experience'].replace({'zero': 0, 'one': 1,
```

```
'two': 2, 'three': 3,'four': 4, 'five': 5, 'six': 6, 'seven': 7,'eight': 8, 'nine':
9, 'ten': 10})

# 1. Train Hiring dataset model

if salary_col in hiring_df.columns:

    hiring_model, _, _ = train_model(hiring_df, salary_col)

    # Predict salary

    candidates_df = pd.DataFrame([[12, 10, 10]],
columns=hiring_df.drop(columns=[salary_col]).columns)

    salary_prediction = hiring_model.predict(candidates_df)

    print(" Predicted Salary for (12 yrs, 10 test, 10 interview):",
salary_prediction[0])

else:

    print(" Error: Salary column is missing in hiring.csv!")

# 2. Train Companies dataset model

if "Profit" in companies_df.columns:

    companies_model, label_encoders, scaler = train_model(companies_df, "Profit",
categorical_columns=["State"], scale_features=True)

    # Encoding Florida state

    if "State" in label_encoders:

        try:

            florida_state_encoded =
label_encoders["State"].transform(["Florida"])[0]

        except ValueError:

            print(" Warning: 'Florida' was not in training data. Assigning default
category.")

            florida_state_encoded = 0  # Assign a default category

    # Predicting profit

    company_features_df = pd.DataFrame(

        [[91694.48, 515841.3, 11931.24, florida_state_encoded]],

        columns=companies_df.drop(columns=["Profit"]).columns)

    # Apply the same scaling transformation
```

```
    company_features_scaled = scaler.transform(company_features_df)

    profit_prediction = companies_model.predict(company_features_scaled)

    print("Predicted Profit:", profit_prediction[0])

else:

    print(" Error: Profit column is missing in 1000_companies.csv!")

# 3. Train Canada per capita income model

if "year" in canada_df.columns and "per capita income (US$)" in canada_df.columns:

    X_canada = canada_df["year"].values.reshape(-1, 1)

    y_canada = canada_df["per capita income (US$)"]

# Train model

    canada_model = LinearRegression()

    canada_model.fit(X_canada, y_canada)

    # Plot regression line

    plt.scatter(X_canada, y_canada, color='blue', label='Actual Data')

    plt.plot(X_canada, canada_model.predict(X_canada), color='red',
label='Regression Line')

    plt.xlabel("Year")

    plt.ylabel("Per Capita Income (US$)")

    plt.title("Regression Line for Canada Per Capita Income")

    plt.legend()

    plt.show()

else:

    print(" Error: Columns missing in canada_per_capita_income.csv!")

print("\n Observations:")

print("1. Data preprocessing was performed to handle missing values, label encoding
for categorical variables (State), and feature scaling for numerical variables in
1000_companies.csv.")

print("2. The regression plot for Canada's per capita income shows a linear increase
over time, indicating a strong positive correlation between year and per capita
income.")

if salary_col in hiring_df.columns:
```

```python
    print(f"3. The predicted salary for a candidate with 12 years of experience, 10 test score, and 10 interview score is {salary_prediction[0]:.2f}.")

print("4. The 'State' column in 1000_companies.csv was label encoded using sklearn's LabelEncoder. Feature scaling was applied because of different unit scales in R&D Spend, Administration, and Marketing Spend.")
```

```
Hiring CSV Columns: Index(['experience', 'test_score(out of 10)', 'interview_score(out of 10)',
       'salary($)'],
      dtype='object')
Companies CSV Columns: Index(['R&D Spend', 'Administration', 'Marketing Spend', 'State', 'Profit'], dtype='object')
Canada CSV Columns: Index(['year', 'per capita income (US$)'], dtype='object')
 Predicted Salary for (12 yrs, 10 test, 10 interview): 65567.47584311156
 Warning: 'Florida' was not in training data. Assigning default category.
Predicted Profit: 554894.6415885255
```

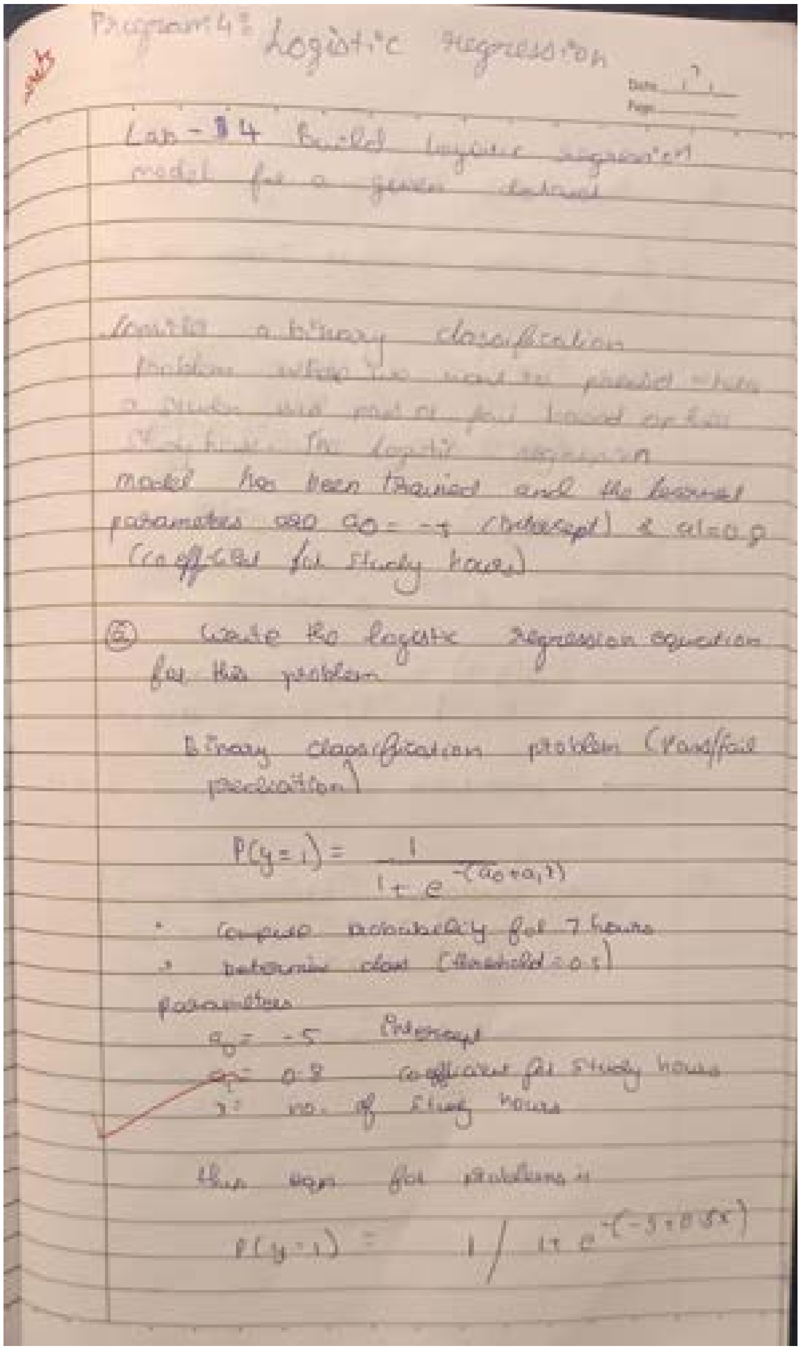



```
 Observations:
1. Data preprocessing was performed to handle missing values, label encoding for categorical variables (State), and feature sc
aling for numerical variables in 1000_companies.csv.
2. The regression plot for Canada's per capita income shows a linear increase over time, indicating a strong positive correlat
ion between year and per capita income.
3. The predicted salary for a candidate with 12 years of experience, 10 test score, and 10 interview score is 65567.48.
4. The 'State' column in 1000_companies.csv was label encoded using sklearn's LabelEncoder. Feature scaling was applied becaus
e of different unit scales in R&D Spend, Administration, and Marketing Spend.
```

Program 4

Build Logistic Regression Model for a given dataset

Screenshot


Program 4: Logistic Regression

Lab - 4 Build logistic regression model for a given dataset

model has been trained and the learned parameters are $a_0 = -5$ (intercept) & $a_1 = 0.8$ (coefficient for study hours)

② Write the logistic regression equation for the problem

Binary classification problem (pass/fail prediction)

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

- compute probability for 7 hours
- determine class (threshold = 0.5)

parameters

$a_0 = -5$ intercept

$a_1 = 0.8$ coefficient for study hours

x: no. of study hours

the eqn for problem is

$$P(y=1) = 1 / 1 + e^{-(-5+0.8x)}$$

(b) Probability calculation for 7 hours

For a student studying 7 hours we calculate

z = ?

$P(y=1) = 1 / 1 + e^{-(-5 + 0.8(7))}$

$P(y=1) = 1 / 1 + e^{-(-5 + 5.6)}$

$P(y=1) = 1 / 1 + e^{-0.6}$

Approx $e^{-0.6} = 0.5488$

$P(y=1) = 1 / 1 + 0.5488$

$P(y=1) = 1 / 1.5488 = 0.645$

→ probability student passes is 0.645
64.5%

(c) class prediction (threshold = 0.5)

if $P(y=1) \geq 0.5$ predict pass

if $P(y=1) < 0.5$ predict fail

Since $0.645 > 0.5$ the student is predicted to pass

(d) Softmax function calculation

$$P_i = \frac{e^{z_i}}{\sum e^{z_j}}$$

given z = [2, 1, 0]

Exponentials $e^2 = 7.389$

$e^1 = 2.718$

$e^0 = 1$

Compute sum of exponentials

$7.389 + 2.718 + 1 = 11.107$

Step 3: compute probabilities

$P_1 = 7.389 / 11.107 = 0.665$

$P_2 = 2.718 / 11.107 = 0.245$

$P_3 = 1 / 11.107 = 0.090$

66.5%

24.5%

9.0%

HR Dataset Observation?

(1) Identified key predictors for employee retention today?

Satisfaction level: low satisfaction may lead to higher employee turnover.

Time spent with company: employees have been with the company for more likely to stay.

Work accident: employees who have been had accidents maybe more likely to leave.

Salary: employees with low salaries may have a higher turnover rate.

Promotion in last 5 years: employees who were not promoted may be more likely to leave.

Ex: missing value

(2) What was the accuracy of your logistic regression model? Do you think this is a good accuracy? Why or why not?

(3) Model accuracy 78.67%

this accuracy is decent but not perfect given the complexity of employee retention. Other factors might influence the outcome such as external job opportunities and company policies.

Improvements

use advanced models like

+ random forest

+ neural network

(4) Confusion matrix analysis

$$\begin{bmatrix} 2117 & 117 \\ 457 & 249 \end{bmatrix}$$

2117 TN = Stay

117 FP – [illegible]

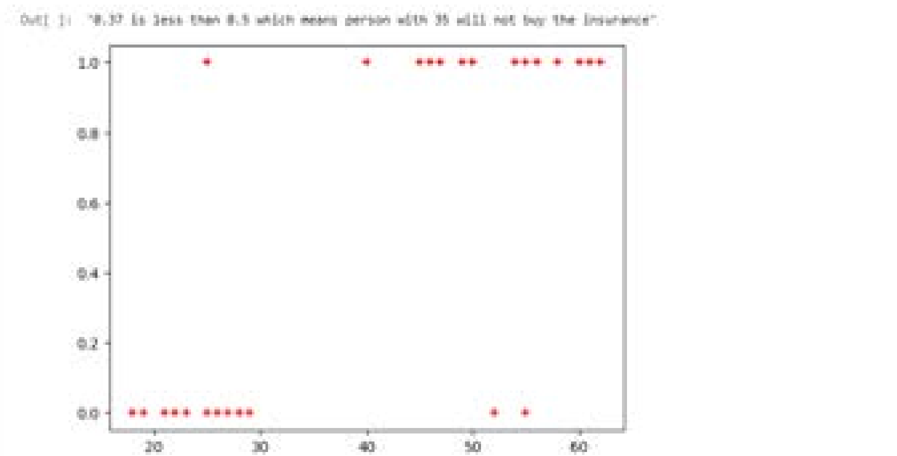
457 FN – [illegible]

249 TP – [illegible]

Code:

```
import pandas as pd
from matplotlib import pyplot as plt
# %matplotlib inline
df = pd.read_csv("insurance_data.csv")
df.head()
plt.scatter(df.age,df.bought_insurance,marker='+',color='red')
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)
X_train.shape
X_test
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train, y_train)
X_test
y_test
y_predicted = model.predict(X_test)
y_predicted
model.score(X_test,y_test)
model.predict_proba(X_test)
y_predicted = model.predict([[60]])
y_predicted
#model.coef_ indicates value of m in y=m*x + b equation
model.coef_
#model.intercept_ indicates value of b in y=m*x + b equation
model.intercept_

#Lets defined sigmoid function now and do the math with hand
import math
def sigmoid(x):
  return 1 / (1 + math.exp(-x))
def prediction_function(age):
   z = 0.127 * age - 4.973 # 0.12740563 ~ 0.0127 and -4.97335111 ~ -4.97
```

```
    y = sigmoid(z)
    return y
age = 35
prediction_function(age)
"""0.37 is less than 0.5 which means person with 35 will not buy the insurance"""
```

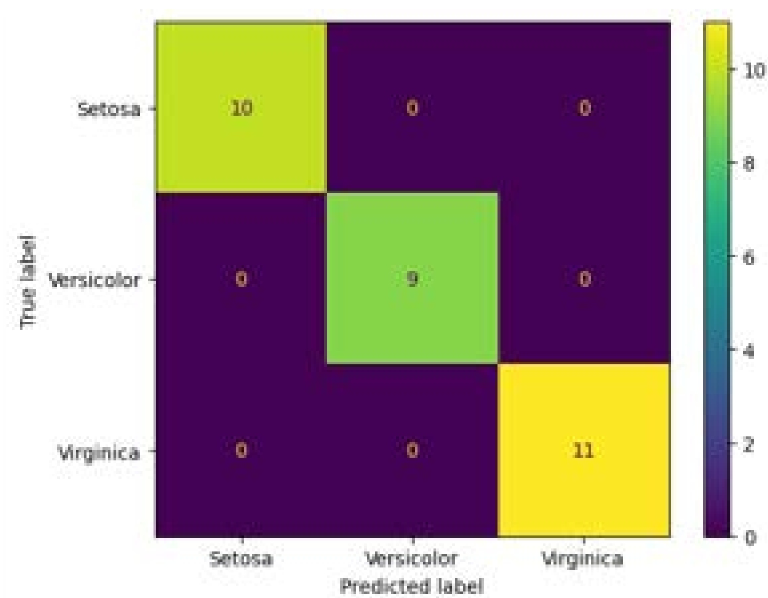


LogisticRegression_Multiclass.ipynb

```
# Import necessary libraries
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.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
# Load the Iris dataset
iris =  pd.read_csv("iris.csv")
iris.head()
X=iris.drop('species',axis='columns')# Features (sepal length, sepal width, petal
length, petal width)
y = iris.species # Target labels (0: Setosa, 1: Versicolor, 2: Virginica)
# Split the dataset into 80% training and 20% testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
# Initialize the Multinomial Logistic Regression model
# Use 'multinomial' for multi-class classification and 'lbfgs' solver
```

```
model = LogisticRegression(multi_class='multinomial')
# Train the model on the training data
model.fit(X_train, y_train)
# Make predictions on the test data
y_pred = model.predict(X_test)
# Calculate the accuracy of the model on the test data
accuracy = accuracy_score(y_test, y_pred)
# Display the accuracy
print(f"Accuracy of the Multinomial Logistic Regression model on the test set:
{accuracy:.2f}")
confusion_matrix = metrics.confusion_matrix(y_test, y_pred)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix,
display_labels = ["Setosa", "Versicolor", "Virginica"])
cm_display.plot()
plt.show()
```

Accuracy of the Multinomial Logistic Regression model on the test set: 1.00



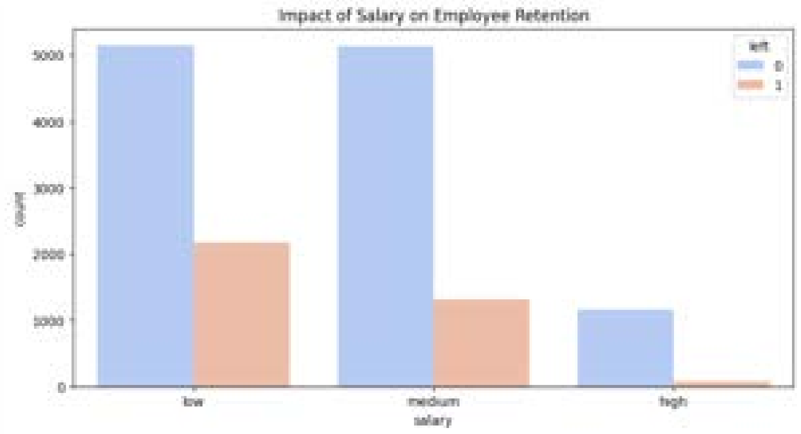

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
```

```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
df = pd.read_csv("HR_comma_sep.csv")
df.head()
 print(df.info())
print(df.isnull().sum())
```

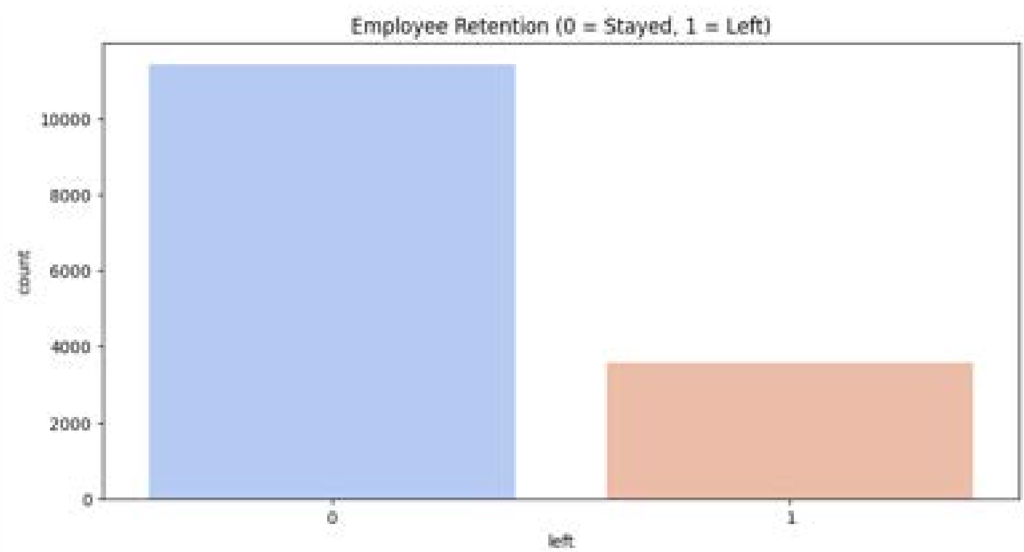
	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Work_accident	left	promotion_las
0	0.38	0.53	2	157	3	0	1	
1	0.80	0.86	5	262	6	0	1	
2	0.11	0.88	7	272	4	0	1	
3	0.72	0.87	5	223	5	0	1	
4	0.37	0.52	2	159	3	0	1	

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14999 entries, 0 to 14998
Data columns (total 10 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   satisfaction_level     14999 non-null  float64
 1   last_evaluation        14999 non-null  float64
 2   number_project         14999 non-null  int64
 3   average_montly_hours   14999 non-null  int64
 4   time_spend_company     14999 non-null  int64
 5   Work_accident          14999 non-null  int64
 6   left                   14999 non-null  int64
 7   promotion_last_5years  14999 non-null  int64
 8   Department             14999 non-null  object
 9   salary                 14999 non-null  object
dtypes: float64(2), int64(6), object(2)
memory usage: 1.1+ MB
None
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
```

```python
import matplotlib.pyplot as plt
plt.figure(figsize=(10,5))
sns.countplot(x='salary', hue='left', data=df, palette='coolwarm')
plt.title('Impact of Salary on Employee Retention')
plt.show()
```




```
plt.figure(figsize=(10,5))
sns.countplot(x='left', data=df, palette='coolwarm')
plt.title('Employee Retention (0 = Stayed, 1 = Left)')
plt.show()
```




```
# Convert categorical variables into numeric
le_salary = LabelEncoder()
df['salary'] = le_salary.fit_transform(df['salary'])
le_dept = LabelEncoder()
df['Department'] = le_dept.fit_transform(df['Department'])
# Select relevant features based on EDA
features = ['satisfaction_level', 'last_evaluation', 'number_project',
'average_montly_hours', 'time_spend_company', 'Work_accident',
'promotion_last_5years', 'salary', 'Department']
X = df[features]
```

```
y = df['left']

#Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
# Train Logistic Regression model
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)
Out[ ]:
LogisticRegression(max_iter=1000)

# Predictions
y_pred = model.predict(X_test)
# Model evaluation
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy of Logistic Regression Model: {accuracy:.2f}')
print(classification_report(y_test, y_pred))
```

```
Accuracy of Logistic Regression Model: 0.76
              precision    recall  f1-score   support

           0       0.79      0.92      0.85      2294
           1       0.47      0.23      0.31       706

    accuracy                           0.76      3000
   macro avg       0.63      0.57      0.58      3000
weighted avg       0.72      0.76      0.72      3000
```

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.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
print("zoo-class-type.csv")
zoo_class =  pd.read_csv("/content/zoo-class-type.csv")
zoo_class.head()
```

zoo-class-type.csv

Out[ ]:

	Class_Number	Number_Of_Animal_Species_In_Class	Class_Type	Animal_Names
0	1	41	Mammal	aardvark, antelope, bear, boar, buffalo, calf,...
1	2	20	Bird	chicken, crow, dove, duck, flamingo, gull, haw...
2	3	5	Reptile	pitviper, seasnake, slowworm, tortoise, tuatara
3	4	13	Fish	bass, carp, catfish, chub, dogfish, haddock, h...
4	5	4	Amphibian	frog, frog, newt, toad

```python
print("zoo-data.csv")
zoo = pd.read_csv("/content/zoo-data.csv")
zoo.head()
```

zoo-data.csv

Out[ ]:

	animal_name	hair	feathers	eggs	milk	airborne	aquatic	predator	toothed	backbone	breathes	venomous	fins	legs	tail	c
0	aardvark	1	0	0	1	0	0	1	1	1	1	0	0	4	0	
1	antelope	1	0	0	1	0	0	0	1	1	1	0	0	4	1	
2	bass	0	0	1	0	0	1	1	1	1	0	0	1	0	1	
3	bear	1	0	0	1	0	0	1	1	1	1	0	0	4	0	
4	boar	1	0	0	1	0	0	1	1	1	1	0	0	4	1	

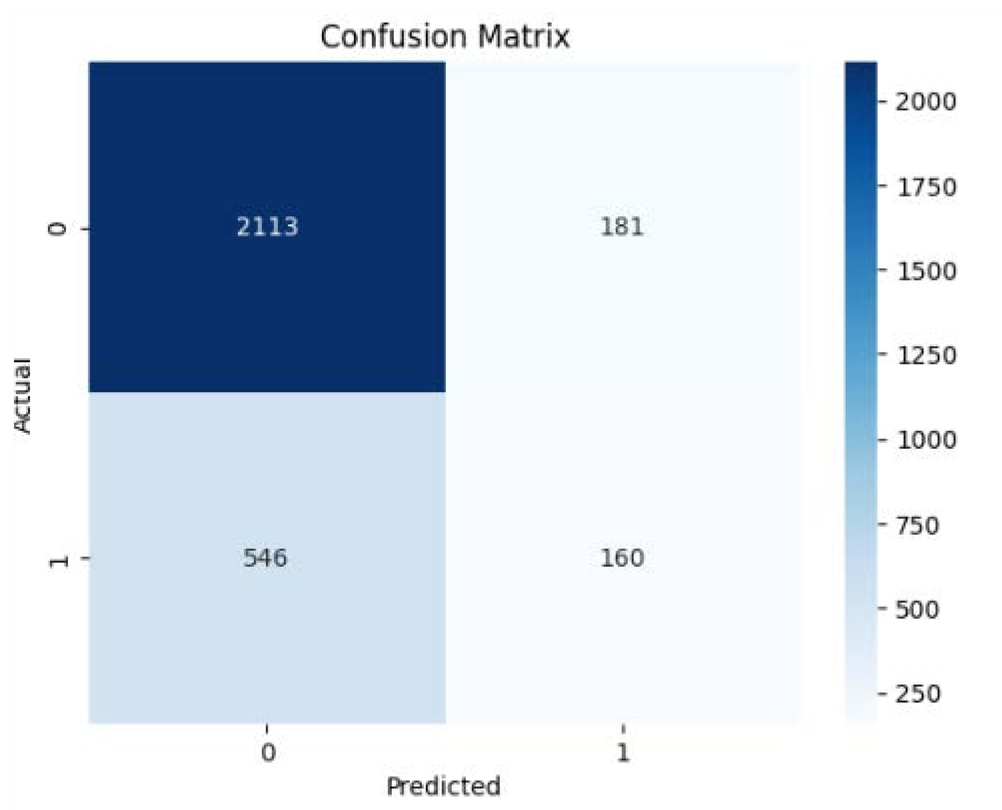
```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import numpy as np
# Load datasets
zoo_data = pd.read_csv("zoo-data.csv")
zoo_class = pd.read_csv("zoo-class-type.csv")
# Display basic info
print(zoo_data.info())
print(zoo_data.isnull().sum())   # Check for missing values
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 101 entries, 0 to 100
Data columns (total 18 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   animal_name  101 non-null    object
 1   hair         101 non-null    int64
 2   feathers     101 non-null    int64
 3   eggs         101 non-null    int64
 4   milk         101 non-null    int64
 5   airborne     101 non-null    int64
 6   aquatic      101 non-null    int64
 7   predator     101 non-null    int64
 8   toothed      101 non-null    int64
 9   backbone     101 non-null    int64
 10  breathes     101 non-null    int64
 11  venomous     101 non-null    int64
 12  fins         101 non-null    int64
 13  legs         101 non-null    int64
 14  tail         101 non-null    int64
 15  domestic     101 non-null    int64
 16  catsize      101 non-null    int64
 17  class_type   101 non-null    int64
dtypes: int64(17), object(1)
memory usage: 14.3+ KB
None
animal_name    0
hair           0
feathers       0
eggs           0
milk           0
airborne       0
aquatic        0
predator       0
toothed        0
backbone       0
breathes       0
venomous       0
```

```python
# Drop 'animal_name' as it's not a useful feature for classification
zoo_data = zoo_data.drop(columns=['animal_name'])
    # Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,
random_state=42)

# Train Logistic Regression model
model = LogisticRegression(max_iter=1000, multi_class='ovr')
model.fit(X_train, y_train)
# Predictions
y_pred = model.predict(X_test)
# Model evaluation
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy of Logistic Regression Model: {accuracy:.2f}')
print(classification_report(y_test, y_pred))
```

```
Accuracy of Logistic Regression Model: 0.76
              precision    recall  f1-score   support

           0       0.79      0.92      0.85      2294
           1       0.47      0.23      0.31       706

    accuracy                           0.76      3000
   macro avg       0.63      0.57      0.58      3000
weighted avg       0.72      0.76      0.72      3000
```

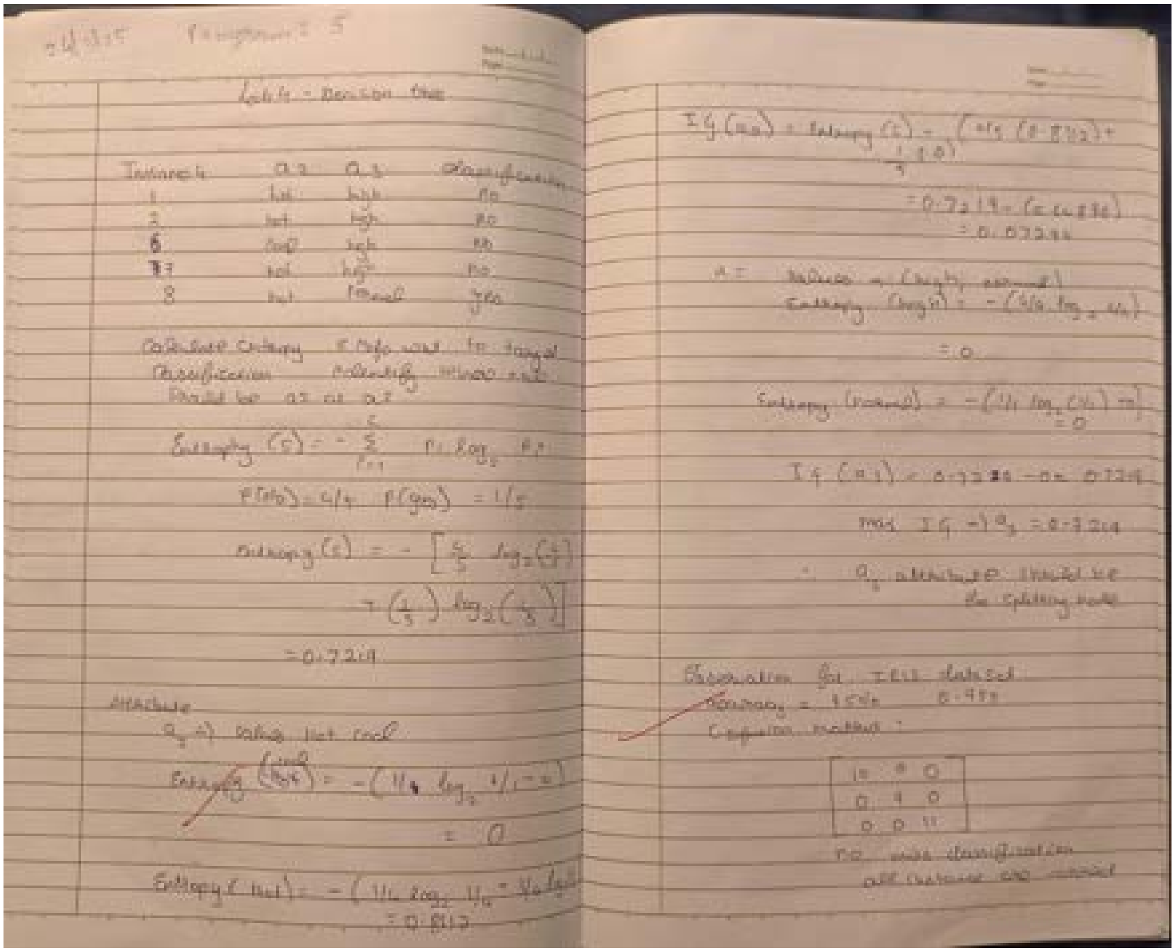
```
# Plot confusion matrix

conf_matrix = confusion_matrix(y_test, y_pred)

sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='d',

xticklabels=np.unique(y), yticklabels=np.unique(y))

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()
```

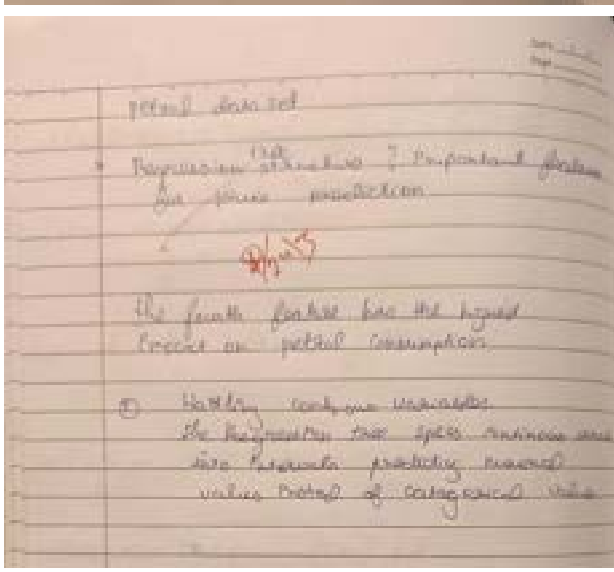


Program 5

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

Screenshot



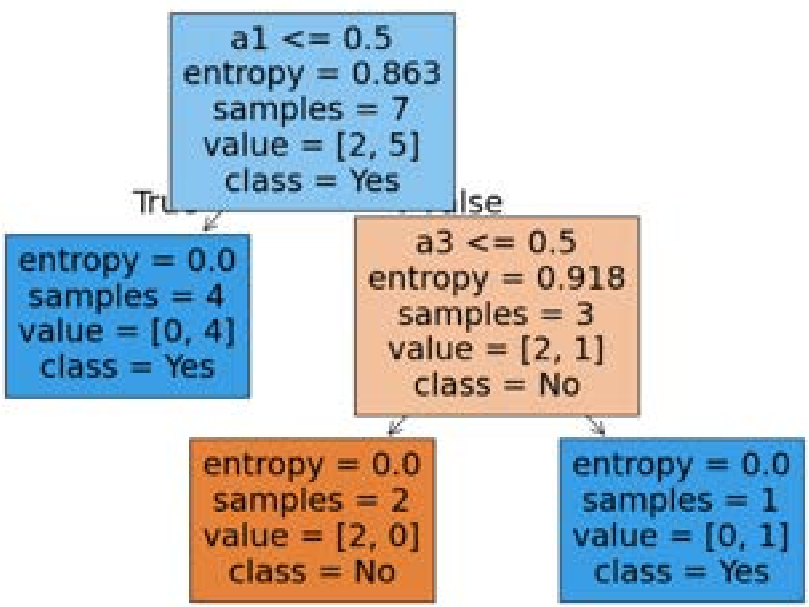


Code:

```
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
# Create the dataset
data = {
    'a1': [True, True, False, False, False, True, True, True, False, False],
    'a2': ['Hot', 'Hot', 'Hot', 'Cool', 'Cool', 'Cool', 'Hot', 'Hot',
'Cool','Cool'],
    'a3': ['High', 'High', 'High', 'Normal', 'Normal', 'High', 'High', 'Normal',
'Normal', 'High'],
    'Classification': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'No', 'Yes', 'Yes',
'Yes']
}
data
# Convert to DataFrame
df = pd.DataFrame(data)
# Convert categorical data to numerical data
label_encoders = {}
for column in df.columns:
    le = LabelEncoder()
    df[column] = le.fit_transform(df[column])
    label_encoders[column] = le
# Split the dataset into features and target
X = df.drop('Classification', axis=1)
y = df['Classification']
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
# Initialize the Decision Tree Classifier with entropy as the criterion
clf = DecisionTreeClassifier(criterion='entropy')
# Train the classifier
```

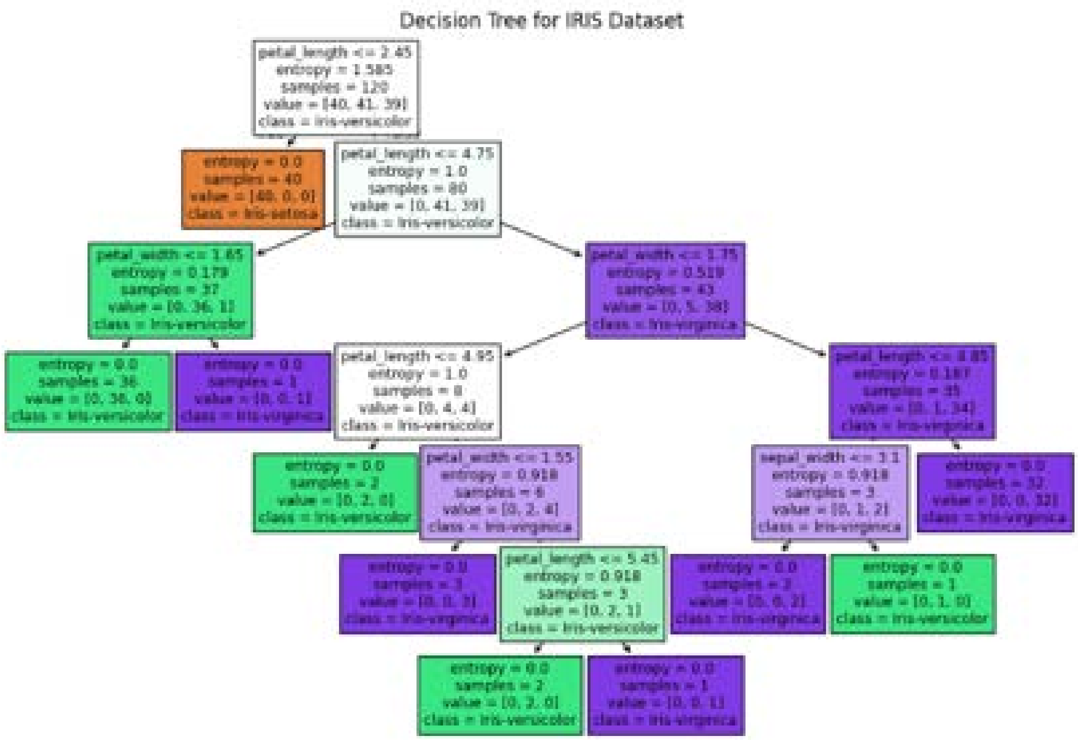
```python
clf.fit(X_train, y_train)

# Make predictions

y_pred = clf.predict(X_test)

# Evaluate the classifier

accuracy = accuracy_score(y_test, y_pred)

print(f'Accuracy: {accuracy:.2f}')

print(classification_report(y_test, y_pred, target_names=['No', 'Yes']))

# Optionally, visualize the decision tree

from sklearn.tree import plot_tree

import matplotlib.pyplot as plt

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

plot_tree(clf, filled=True, feature_names=X.columns, class_names=['No', 'Yes'])

plt.show()
```

```
Accuracy: 1.00
              precision    recall  f1-score   support

          No       1.00      1.00      1.00         2
         Yes       1.00      1.00      1.00         1

    accuracy                           1.00         3
   macro avg       1.00      1.00      1.00         3
weighted avg       1.00      1.00      1.00         3
```



```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree
# Load the iris dataset (make sure iris.csv is in the working directory)
iris = pd.read_csv("iris.csv")
# Assuming the last column is the target (species) and the rest are features.
X = iris.iloc[:, :-1]
y = iris.iloc[:, -1]
# Split data into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
# Initialize and train the Decision Tree classifier
clf_iris = DecisionTreeClassifier(criterion='entropy', random_state=42)
clf_iris.fit(X_train, y_train)
# Make predictions and evaluate the model
y_pred_iris = clf_iris.predict(X_test)
accuracy_iris = accuracy_score(y_test, y_pred_iris)
conf_matrix_iris = confusion_matrix(y_test, y_pred_iris)
print("IRIS Dataset Decision Tree Classifier")
print("Accuracy:", accuracy_iris)
print("Confusion Matrix:\n", conf_matrix_iris)
print("Classification Report:\n", classification_report(y_test, y_pred_iris))
# Visualize the decision tree
plt.figure(figsize=(12, 8))
plot_tree(clf_iris, filled=True, feature_names=X.columns,
class_names=clf_iris.classes_)
plt.title("Decision Tree for IRIS Dataset")
plt.show()
```

```
IRIS Dataset Decision Tree Classifier
Accuracy: 1.0
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
```

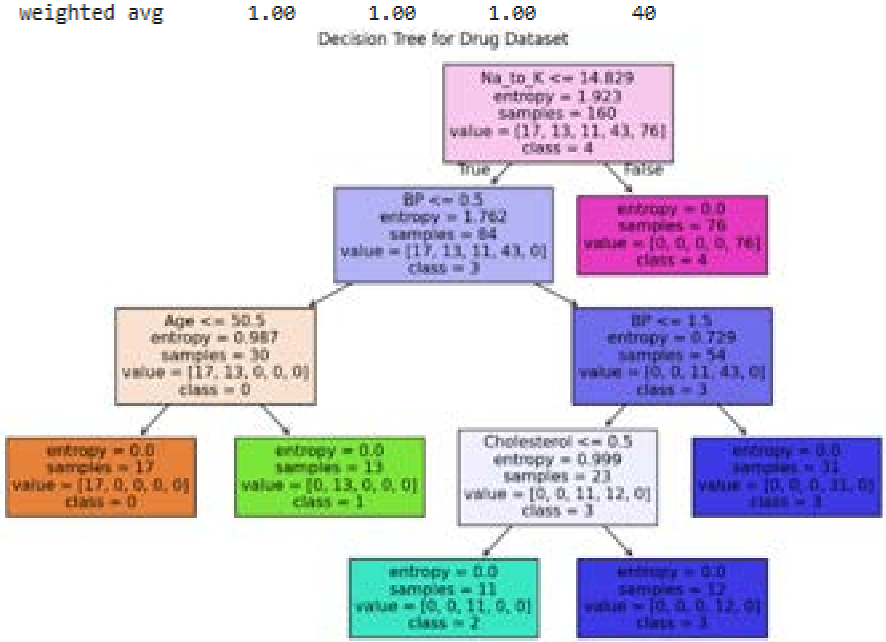



```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
import matplotlib.pyplot as plt
```

```python
from sklearn.tree import plot_tree

# Load the drug dataset (make sure drug.csv is in the working directory)
drug = pd.read_csv("drug.csv")

# Since the target column is 'Drug', drop it from the features
X_drug = drug.drop('Drug', axis=1)
y_drug = drug['Drug']

# If there are categorical features, perform necessary encoding
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
# Encode features that are categorical
for col in X_drug.select_dtypes(include='object').columns:
    X_drug[col] = le.fit_transform(X_drug[col])
# Also encode the target variable if necessary
y_drug = le.fit_transform(y_drug)

# Split the data (80% training, 20% testing)
X_train_d, X_test_d, y_train_d, y_test_d = train_test_split(X_drug, y_drug,
test_size=0.2, random_state=42)

# Initialize and train the Decision Tree classifier using entropy criterion
clf_drug = DecisionTreeClassifier(criterion='entropy', random_state=42)
clf_drug.fit(X_train_d, y_train_d)

# Make predictions and evaluate the model
y_pred_drug = clf_drug.predict(X_test_d)
accuracy_drug = accuracy_score(y_test_d, y_pred_drug)
conf_matrix_drug = confusion_matrix(y_test_d, y_pred_drug)

print("Drug Dataset Decision Tree Classifier")
print("Accuracy:", accuracy_drug)
print("Confusion Matrix:\n", conf_matrix_drug)
```

```python
print("Classification Report:\n", classification_report(y_test_d, y_pred_drug))


# Visualize the decision tree
plt.figure(figsize=(12, 8))
plot_tree(clf_drug, filled=True, feature_names=X_drug.columns,
          class_names=[str(cls) for cls in clf_drug.classes_])
plt.title("Decision Tree for Drug Dataset")
plt.show()
```

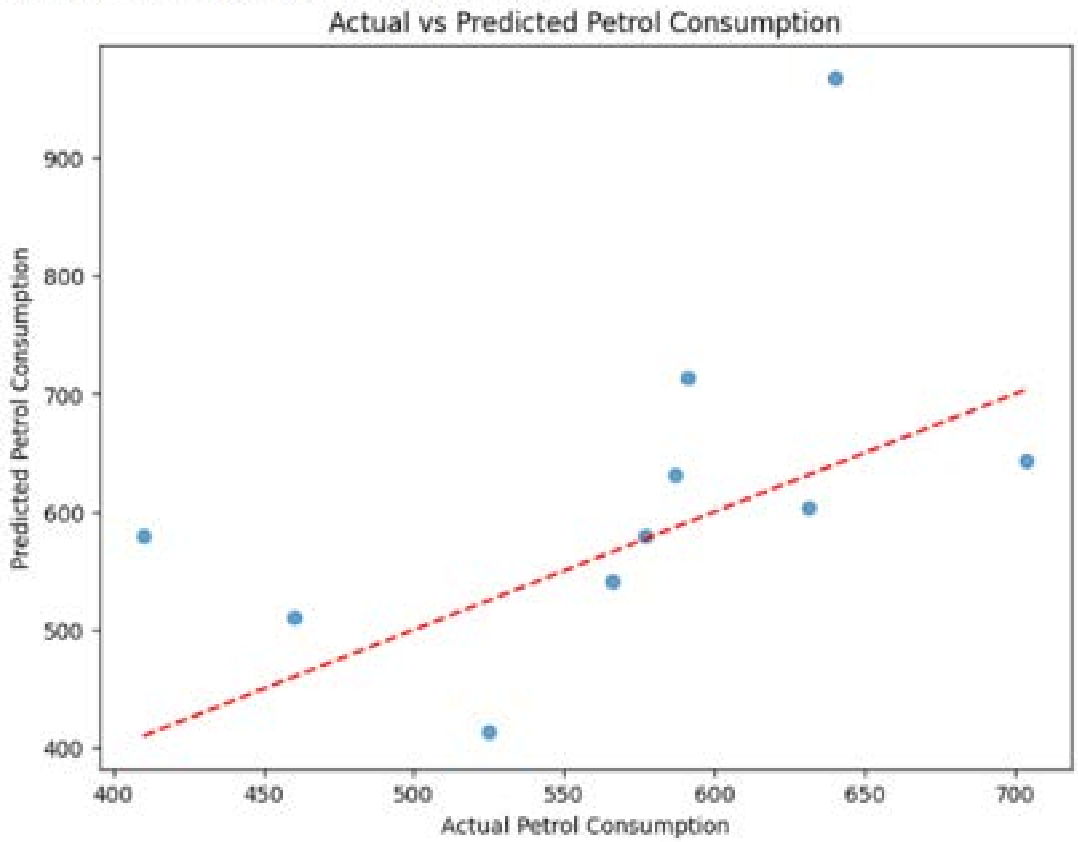
```
Drug Dataset Decision Tree Classifier
Accuracy: 1.0
Confusion Matrix:
 [[ 6  0  0  0  0]
 [ 0  3  0  0  0]
 [ 0  0  5  0  0]
 [ 0  0  0 11  0]
 [ 0  0  0  0 15]]
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00         3
           2       1.00      1.00      1.00         5
           3       1.00      1.00      1.00        11
           4       1.00      1.00      1.00        15

    accuracy                           1.00        40
   macro avg       1.00      1.00      1.00        40
weighted avg       1.00      1.00      1.00        40
```



```python
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 matplotlib.pyplot as plt
# Load the petrol consumption dataset (make sure petrol_consumption.csv is in the
working directory)
petrol = pd.read_csv("petrol_consumption.csv")
# The dataset has a target variable named 'Petrol_Consumption'
# and the remaining columns are features.
X_petrol = petrol.drop('Petrol_Consumption', axis=1)
y_petrol = petrol['Petrol_Consumption']
# If necessary, handle categorical variables here (e.g., using dummy encoding)
# X_petrol = pd.get_dummies(X_petrol)
# Split the data (80% training, 20% testing)
X_train_p, X_test_p, y_train_p, y_test_p = train_test_split(X_petrol, y_petrol,
test_size=0.2, random_state=42)
# Initialize and train the Decision Tree Regressor
regressor = DecisionTreeRegressor(random_state=42)
regressor.fit(X_train_p, y_train_p)
# Make predictions
y_pred_p = regressor.predict(X_test_p)
# Evaluate the regression model
mae = mean_absolute_error(y_test_p, y_pred_p)
mse = mean_squared_error(y_test_p, y_pred_p)
rmse = np.sqrt(mse)
print("Petrol Consumption Regression Tree")
print("Mean Absolute Error (MAE):", mae)
print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
# Optional: Plotting predicted vs actual values-*/
plt.figure(figsize=(8, 6))
plt.scatter(y_test_p, y_pred_p, alpha=0.7)
```

```
plt.xlabel("Actual Petrol Consumption")
plt.ylabel("Predicted Petrol Consumption")
plt.title("Actual vs Predicted Petrol Consumption")
plt.plot([y_test_p.min(), y_test_p.max()], [y_test_p.min(), y_test_p.max()], 'r--')
plt.show()
```

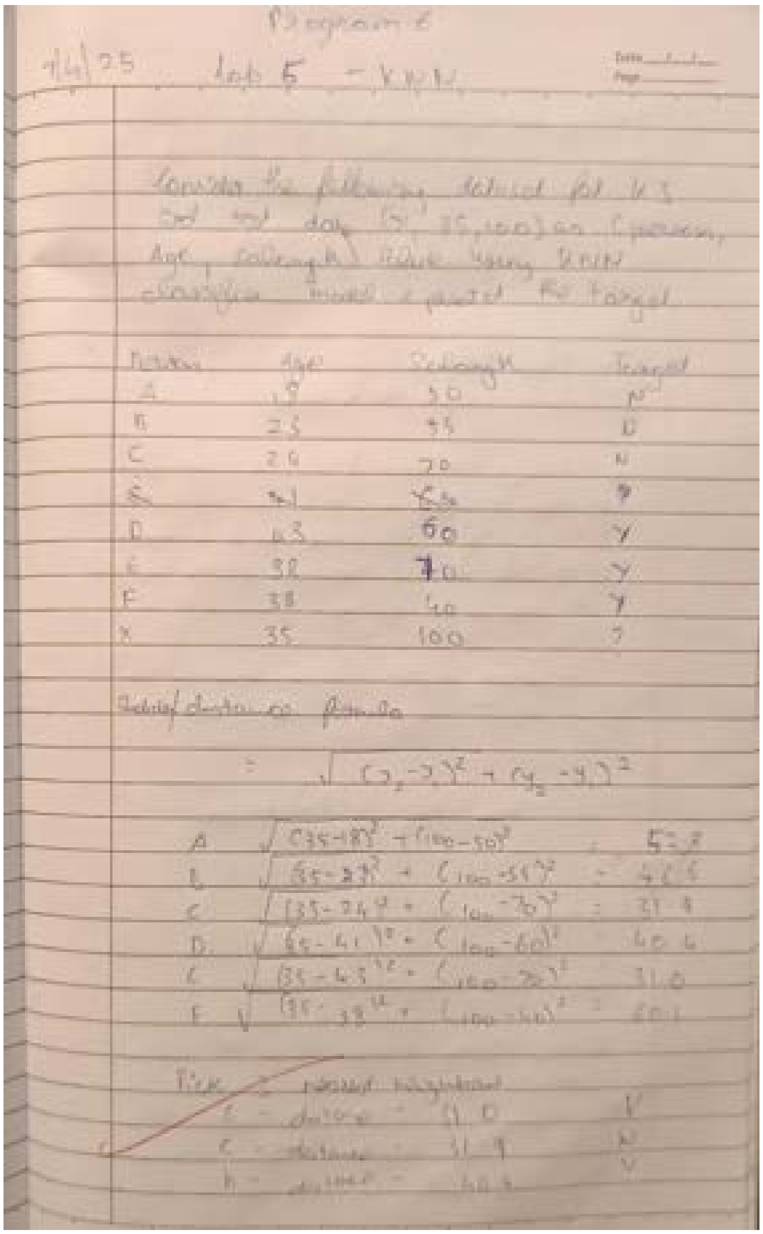
```
Petrol Consumption Regression Tree
Mean Absolute Error (MAE): 94.3
Mean Squared Error (MSE): 17347.7
Root Mean Squared Error (RMSE): 131.7106677532234
```



Program 6

Build KNN Classification model for a given dataset.

Screenshot



Program 6

Lab 5 - KNN

Person	Age	[illegible]	Target
A	18	50	N
B	23	55	[illegible]
C	26	70	N
[illegible]	41	[illegible]	[illegible]
D	43	60	Y
E	32	70	Y
F	38	40	Y
X	35	100	?

$$\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$$

Date / /
Page

Majority vote from 3 neighbour

(5,0)
2 votes   N: 2 votes (c)

predicted target for (3, 5, 10)

for Iris Dataset
Q) How to chose the K value?

To choose the best K value, we plot the error & accuracy ratio for different values of K (1 to 20). The value of K that gives the highest accuracy on the test set is selected as optimal.

Accuracy rate (correct prediction) / (total predictions)

Error rate 1 − accuracy

Diabetes Data Set
What is the purpose of feature scaling? how to perform it

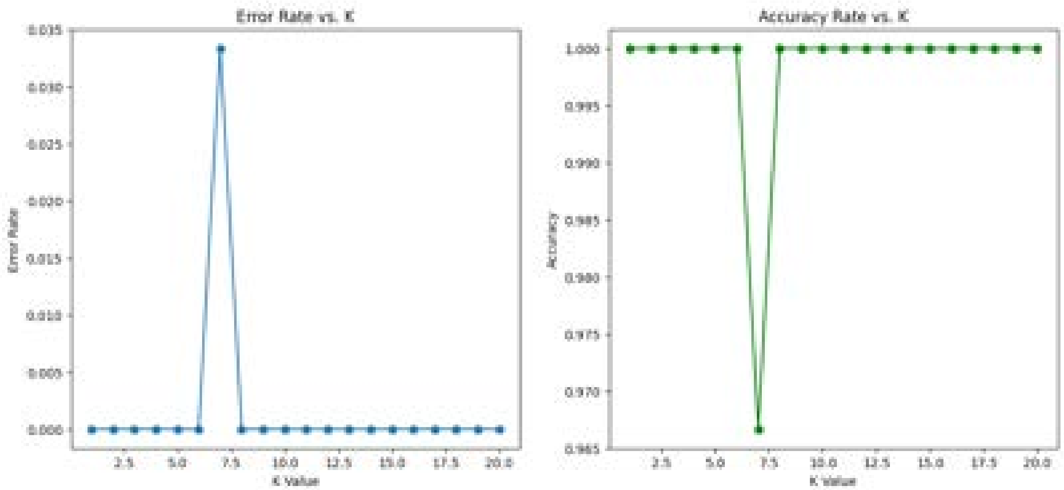
Feature scaling ensures that all features contribute equally to the distance computation in KNN. Without scaling features with larger ranges dominate. It is performed using StandardScaler in Sklearn which transforms features to have mean and unit variance

45

code

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
# Load iris dataset
iris_df = pd.read_csv("/content/drive/MyDrive/MLlab dataset/iris (1).csv")
 # change path
# Features and labels
X = iris_df.drop(columns=['species'])
y = iris_df['species']
# Train test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Try different k values
error_rate = []
accuracy_rate = []
k_range = range(1, 21)

for k in k_range:
    model = KNeighborsClassifier(n_neighbors=k)
    model.fit(X_train, y_train)
```

```python
    preds = model.predict(X_test)
    error_rate.append(np.mean(preds != y_test))
    accuracy_rate.append(accuracy_score(y_test, preds))

# Plot error rate and accuracy
plt.figure(figsize=(14,6))
plt.subplot(1,2,1)
plt.plot(k_range, error_rate, marker='o')
plt.title("Error Rate vs. K")
plt.xlabel("K Value")
plt.ylabel("Error Rate")

plt.subplot(1,2,2)
plt.plot(k_range, accuracy_rate, marker='o', color='green')
plt.title("Accuracy Rate vs. K")
plt.xlabel("K Value")
plt.ylabel("Accuracy")
plt.show()

# Final Model with best k
best_k = accuracy_rate.index(max(accuracy_rate)) + 1
knn = KNeighborsClassifier(n_neighbors=best_k)
knn.fit(X_train, y_train)
predictions = knn.predict(X_test)

print("Best K:", best_k)
print("Accuracy:", accuracy_score(y_test, predictions))
print("Confusion Matrix:\n", confusion_matrix(y_test, predictions))
print("Classification Report:\n", classification_report(y_test, predictions))
```




```
Best K: 1
Accuracy: 1.0
Confusion Matrix:
 [[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
Classification Report:
               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
```

Diabetes Dataset - KNN with Feature Scaling

```
# Load dataset
diabetes_df = pd.read_csv('/content/drive/MyDrive/MLlab dataset/diabetes.csv')
# change path
X = diabetes_df.drop(columns=['Outcome'])
y = diabetes_df['Outcome']
# Feature scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Train test split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,
random_state=42)
```

```python
# Try different k values
   accuracy_scores = []
   for k in range(1, 21):
       model = KNeighborsClassifier(n_neighbors=k)
       model.fit(X_train, y_train)
       y_pred = model.predict(X_test)
       accuracy_scores.append(accuracy_score(y_test, y_pred))
# Best K
   best_k = accuracy_scores.index(max(accuracy_scores)) + 1
   # Final model
   knn = KNeighborsClassifier(n_neighbors=best_k)
   knn.fit(X_train, y_train)
   predictions = knn.predict(X_test)
   print("Best K:", best_k)
   print("Accuracy:", accuracy_score(y_test, predictions))
   print("Confusion Matrix:\n", confusion_matrix(y_test, predictions))
print("Classification Report:\n", classification_report(y_test, predictions))
```

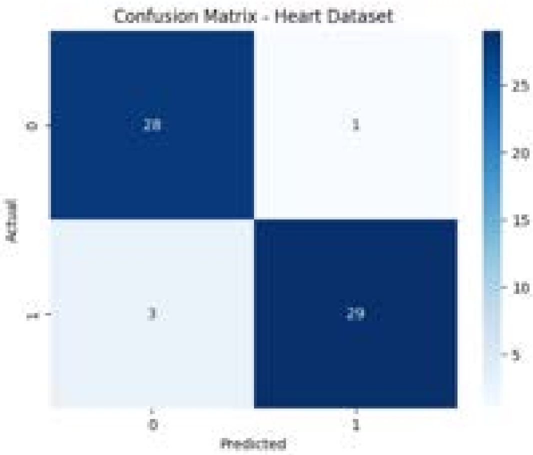
```
Best K: 18
Accuracy: 0.7662337662337663
Confusion Matrix:
 [[89 10]
 [26 29]]
Classification Report:
              precision    recall  f1-score   support

           0       0.77      0.90      0.83        99
           1       0.74      0.53      0.62        55

    accuracy                           0.77       154
   macro avg       0.76      0.71      0.72       154
weighted avg       0.76      0.77      0.76       154
```

Heart Dataset - KNN Classification

```python
# Load dataset
heart_df = pd.read_csv('/content/drive/MyDrive/MLlab dataset/heart.csv')
# change path
X = heart_df.drop(columns=['target'])
y = heart_df['target']
# Scaling
X_scaled = StandardScaler().fit_transform(X)
# Train test split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,
random_state=42)
```

```python
# Test different k values
accuracy_scores = []
for k in range(1, 21):
    model = KNeighborsClassifier(n_neighbors=k)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy_scores.append(accuracy_score(y_test, y_pred))
best_k = accuracy_scores.index(max(accuracy_scores)) + 1
# Final model
knn = KNeighborsClassifier(n_neighbors=best_k)
knn.fit(X_train, y_train)
predictions = knn.predict(X_test)
# Metrics and plots
print("Best K:", best_k)
print("Accuracy:", accuracy_score(y_test, predictions))
print("Confusion Matrix:\n", confusion_matrix(y_test, predictions))
print("Classification Report:\n", classification_report(y_test, predictions))
# Plot confusion matrix
sns.heatmap(confusion_matrix(y_test, predictions), annot=True, cmap='Blues', fmt='d')
plt.title("Confusion Matrix - Heart Dataset")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

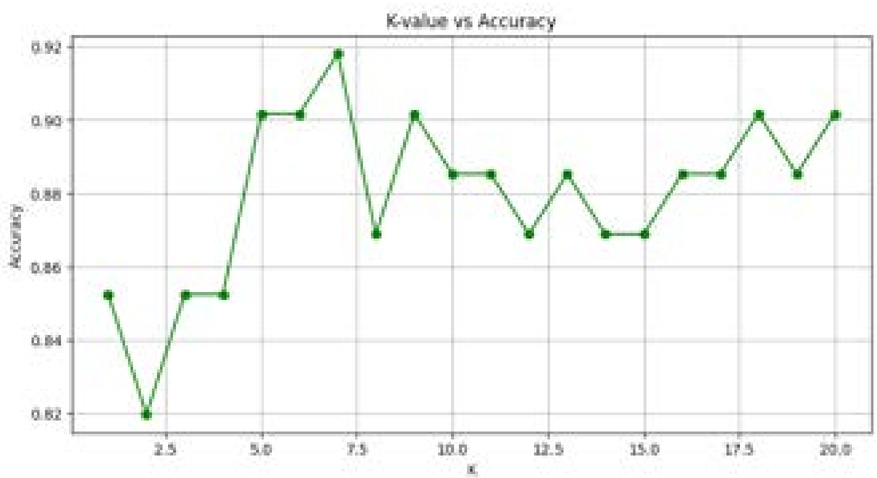
```
Best K: 6
Accuracy: 0.9344262295081968
Confusion Matrix:
 [[28  1]
 [ 3 29]]
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.97      0.93        29
           1       0.97      0.91      0.94        32

    accuracy                           0.93        61
   macro avg       0.93      0.94      0.93        61
weighted avg       0.94      0.93      0.93        61
```



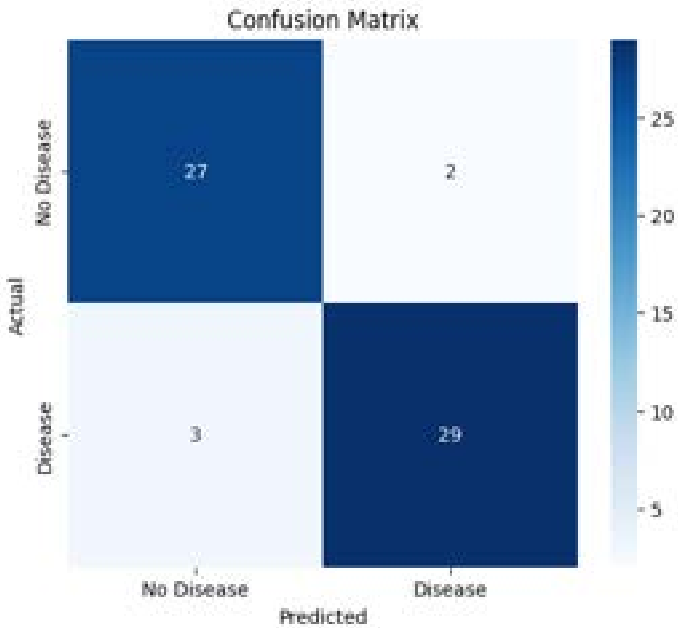

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
# Load dataset
heart = pd.read_csv('/content/drive/MyDrive/MLlab dataset/heart.csv')
# Features and target
X = heart.drop(columns=['target'])
y = heart['target']
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

```python
# Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
# Try different values of k and choose the best one
accuracy_scores = []
k_values = range(1, 21)
for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train_scaled, y_train)
    y_pred = knn.predict(X_test_scaled)
    acc = accuracy_score(y_test, y_pred)
    accuracy_scores.append(acc)
# Plot accuracy vs. k
plt.figure(figsize=(10, 5))
plt.plot(k_values, accuracy_scores, marker='o', color='green')
plt.title('K-value vs Accuracy')
plt.xlabel('K')
plt.ylabel('Accuracy')
plt.grid(True)
plt.show()
# Get best K
best_k = k_values[np.argmax(accuracy_scores)]
print(f"Best K: {best_k} with Accuracy: {max(accuracy_scores):.4f}")
# Train final model with best K
knn = KNeighborsClassifier(n_neighbors=best_k)
knn.fit(X_train_scaled, y_train)
y_pred = knn.predict(X_test_scaled)
# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 5))
```

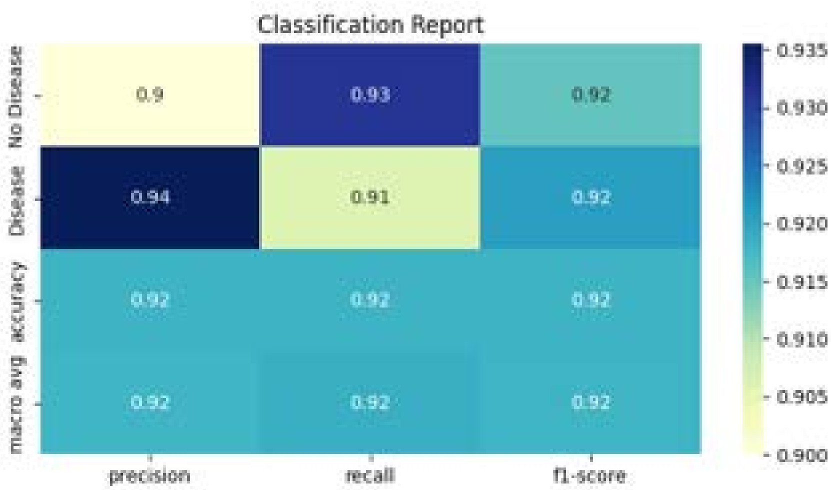
```
sns.heatmap(cm, annot=True, fmt="d", cmap='Blues', xticklabels=['No Disease',
 'Disease'], yticklabels=['No Disease', 'Disease'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
# Classification Report
report = classification_report(y_test, y_pred, target_names=['No Disease',
'Disease'], output_dict=True)
df_report = pd.DataFrame(report).transpose()
plt.figure(figsize=(8, 4))
sns.heatmap(df_report.iloc[:-1, :-1], annot=True, cmap="YlGnBu")
plt.title("Classification Report")
plt.show()
```




```
Best K: 7 with Accuracy: 0.9180
```



Confusion Matrix
No Disease
Disease
Actual
27
2
3
29
No Disease
Disease
Predicted
25
20
15
10
5

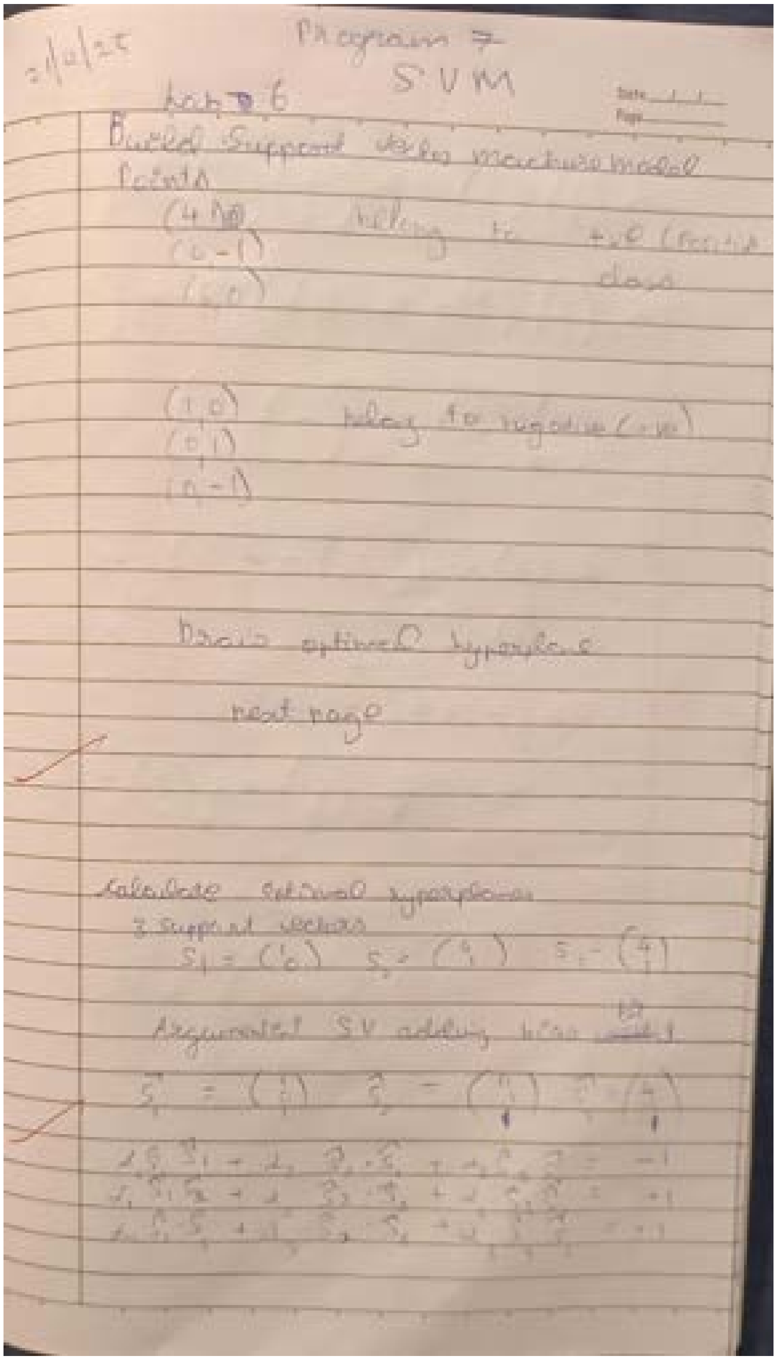


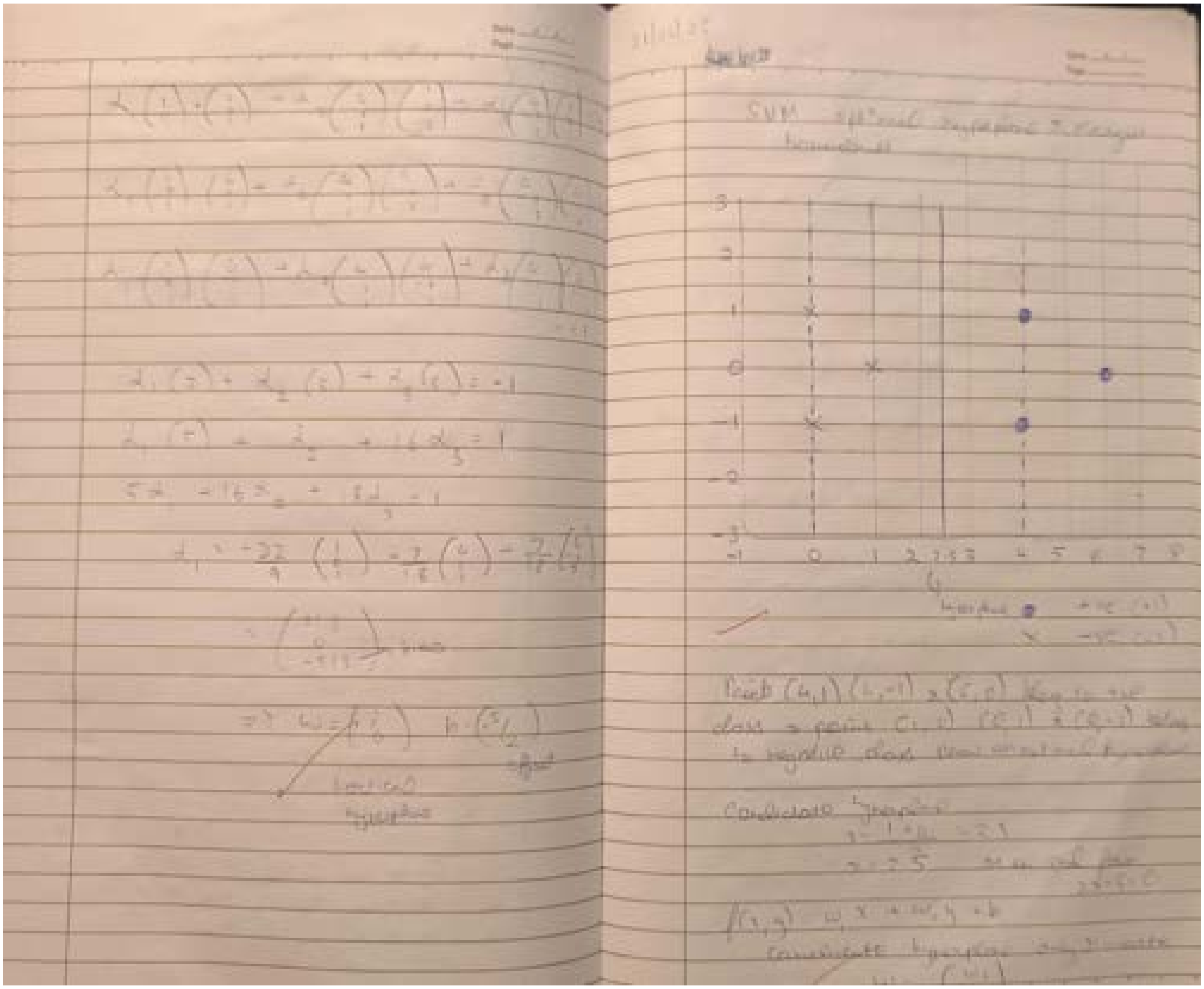
Classification Report
No Disease
Disease
accuracy
macro avg
0.9
0.93
0.92
0.94
0.91
0.92
0.92
0.92
0.92
0.92
0.92
0.92
precision
recall
f1-score
0.935
0.930
0.925
0.920
0.915
0.910
0.905
0.900

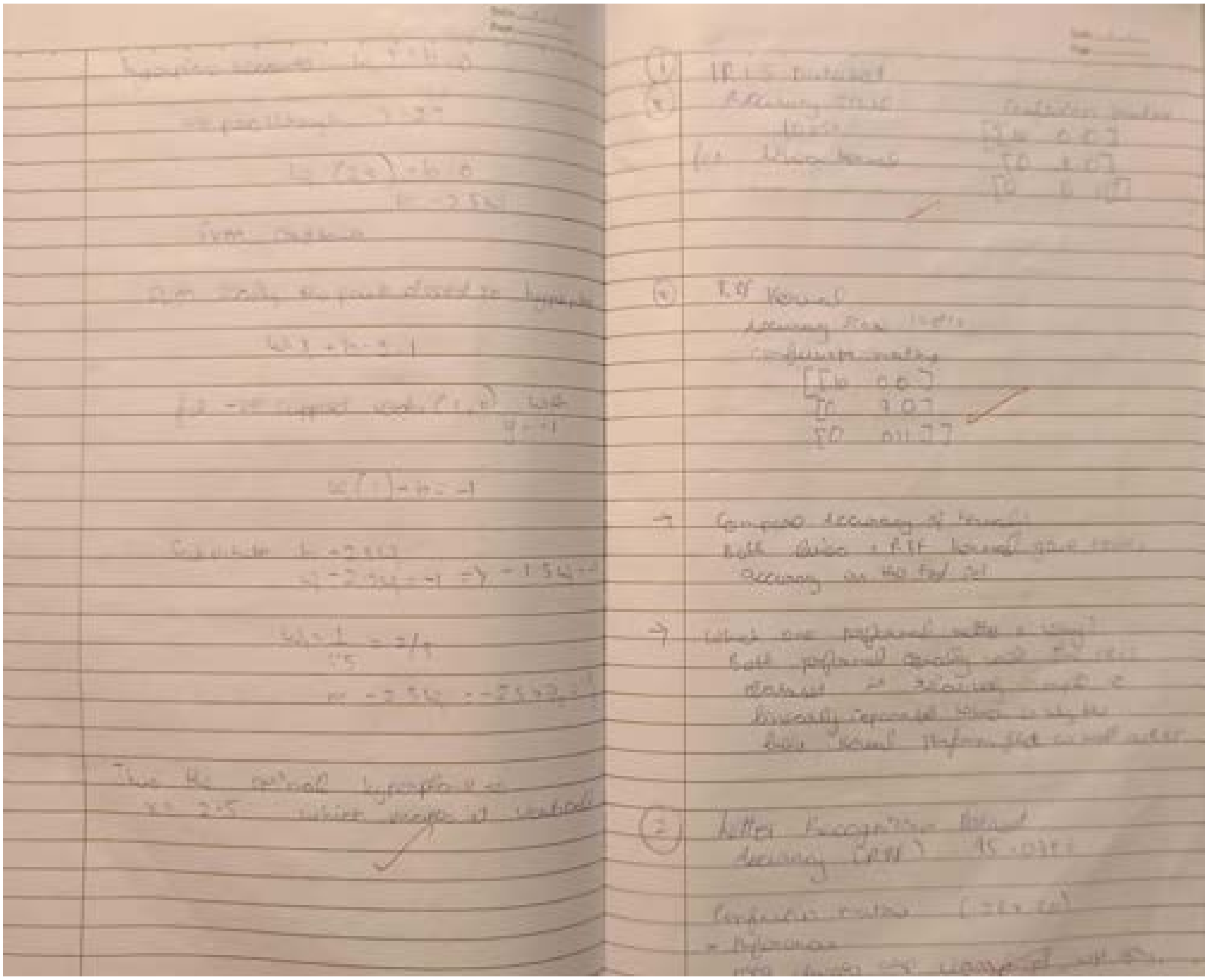
Program 7

Build Support vector machine model for a given dataset

Screenshots:




SVM



Page

high accuracy.

minor confusion.
class B is sometimes misclassified as D &
class H is confused as L & R

B ↔ D
H ↔ L & R

AUC score : too computationally
heavy to full compute due to
26 classes but generally a high ROC
AUC is expected given the high
accuracy

Comparison with IRIS:
100% accuracy since simpler
linearly separable dataset

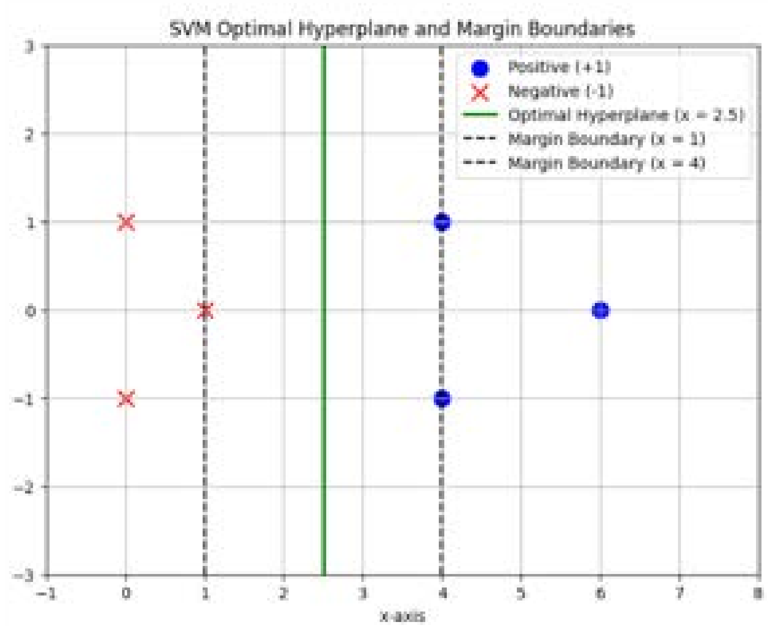
letter dataset is a 94% due class
imbalance high complexity

R
3/1/4/25

Code:

```
import matplotlib.pyplot as plt
import numpy as np
# Define the data points for each class
positive_points = np.array([[4, 1],
                            [4, -1],
                            [6, 0]])
negative_points = np.array([[1, 0],
                            [0, 1],
                            [0, -1]])
# Extract the x and y coordinates for each class
pos_x, pos_y = positive_points[:, 0], positive_points[:, 1]
neg_x, neg_y = negative_points[:, 0], negative_points[:, 1]
# Create a new figure for the plot
plt.figure(figsize=(8, 6))
# Plot the points
plt.scatter(pos_x, pos_y, color="blue", marker="o", s=100,
label="Positive (+1)")
plt.scatter(neg_x, neg_y, color="red", marker="x", s=100,
label="Negative (-1)")
# Draw the optimal hyperplane (vertical line at x = 2.5)
plt.axvline(x=2.5, color="green", linestyle="solid", linewidth=2,
label="Optimal Hyperplane (x = 2.5)")
# Draw the margin boundaries (vertical lines at x = 1 and x = 4)
plt.axvline(x=1.0, color="black", linestyle="dashed", linewidth=1.5,
label="Margin Boundary (x = 1)")
plt.axvline(x=4.0, color="black", linestyle="dashed", linewidth=1.5,
label="Margin Boundary (x = 4)")
# Set the plot labels and title
plt.xlabel("x-axis")
plt.ylabel("y-axis")
plt.title("SVM Optimal Hyperplane and Margin Boundaries")
# Set the axis limits for clarity
plt.xlim(-1, 8)
plt.ylim(-3, 3)
# Turn on the grid
plt.grid(True)
# Display legend
plt.legend()
# Show the plot
```

```
plt.show()
```



Build a SVM classifier to classify IRIS flower dataset using the kernels RBF and linear.Use 80% of data for training and 20% for testing.Display accuracy score and confusion matrix of the trained model on test data.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix
# Load the iris dataset
iris_df = pd.read_csv("/content/drive/MyDrive/MLlab dataset/iris (1)
(1).csv")
# Prepare features and labels
X = iris_df.iloc[:, :-1]  # all columns except last
y = iris_df.iloc[:, -1]   # last column is label
# Encode target labels (if needed)
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)
# Train-test split (80-20)
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded,
test_size=0.2, random_state=42)
# Train SVM models
svm_linear = SVC(kernel='linear')
svm_rbf = SVC(kernel='rbf')
```

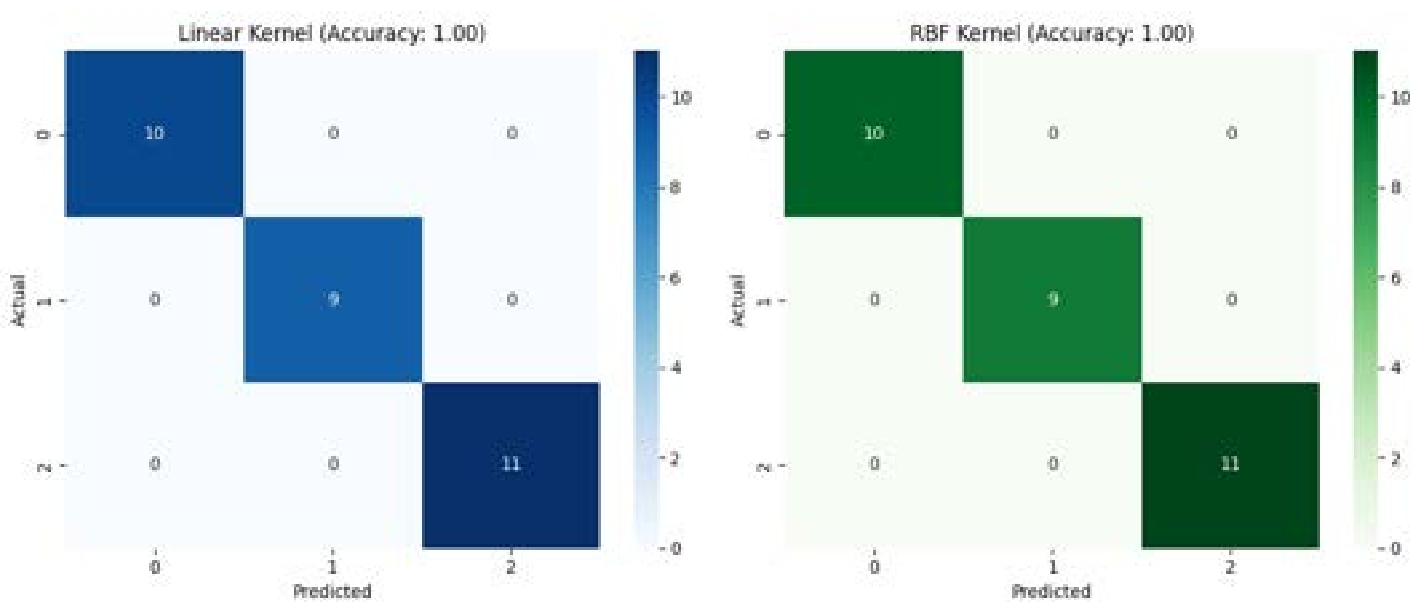
```python
svm_linear.fit(X_train, y_train)
svm_rbf.fit(X_train, y_train)
# Predictions
y_pred_linear = svm_linear.predict(X_test)
y_pred_rbf = svm_rbf.predict(X_test)
# Accuracy and Confusion Matrix
acc_linear = accuracy_score(y_test, y_pred_linear)
acc_rbf = accuracy_score(y_test, y_pred_rbf)

cm_linear = confusion_matrix(y_test, y_pred_linear)
cm_rbf = confusion_matrix(y_test, y_pred_rbf)

acc_linear, cm_linear, acc_rbf, cm_rbf
```

```
Out[ ]: (1.0,
         array([[10,  0,  0],
                [ 0,  9,  0],
                [ 0,  0, 11]]),
         1.0,
         array([[10,  0,  0],
                [ 0,  9,  0],
                [ 0,  0, 11]]))
```

IRIS Dataset - SVM Classification Results • Linear Kernel Accuracy: 100%

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_csv("/content/drive/MyDrive/MLlab dataset/iris (1) (1).csv")
X = df.drop('species', axis=1)
y = LabelEncoder().fit_transform(df['species'])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
svm_linear = SVC(kernel='linear')
svm_rbf = SVC(kernel='rbf')
svm_linear.fit(X_train, y_train)
svm_rbf.fit(X_train, y_train)
y_pred_linear = svm_linear.predict(X_test)
y_pred_rbf = svm_rbf.predict(X_test)
```

```
acc_linear = accuracy_score(y_test, y_pred_linear)
acc_rbf = accuracy_score(y_test, y_pred_rbf)
cm_linear = confusion_matrix(y_test, y_pred_linear)
cm_rbf = confusion_matrix(y_test, y_pred_rbf)
fig, axes = plt.subplots(1, 2, figsize=(12, 5))
sns.heatmap(cm_linear, annot=True, fmt='d', cmap='Blues', ax=axes[0])
axes[0].set_title(f"Linear Kernel (Accuracy: {acc_linear:.2f})")
axes[0].set_xlabel("Predicted")
axes[0].set_ylabel("Actual")
sns.heatmap(cm_rbf, annot=True, fmt='d', cmap='Greens', ax=axes[1])
axes[1].set_title(f"RBF Kernel (Accuracy: {acc_rbf:.2f})")
axes[1].set_xlabel("Predicted")
axes[1].set_ylabel("Actual")
plt.tight_layout()
plt.show()
print(f"Linear Kernel Accuracy: {acc_linear:.4f}")
print(f"RBF Kernel Accuracy: {acc_rbf:.4f}")
```

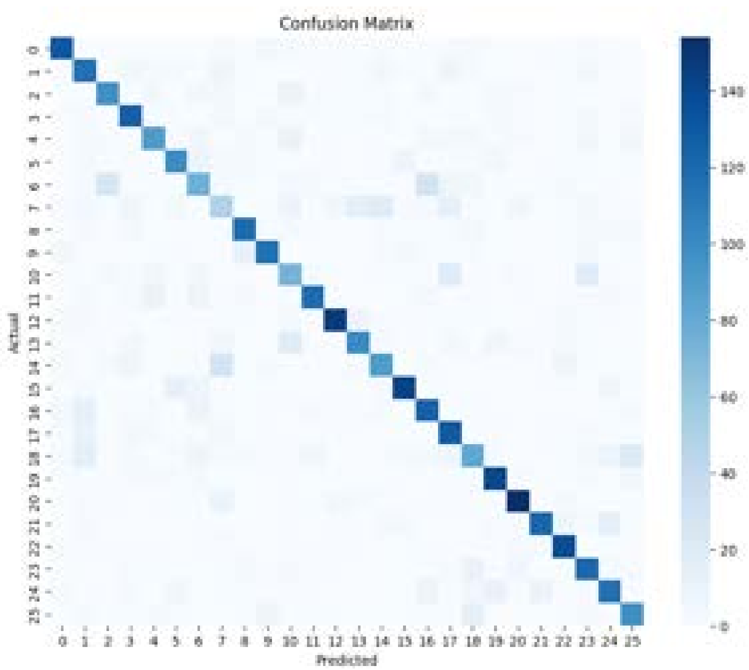


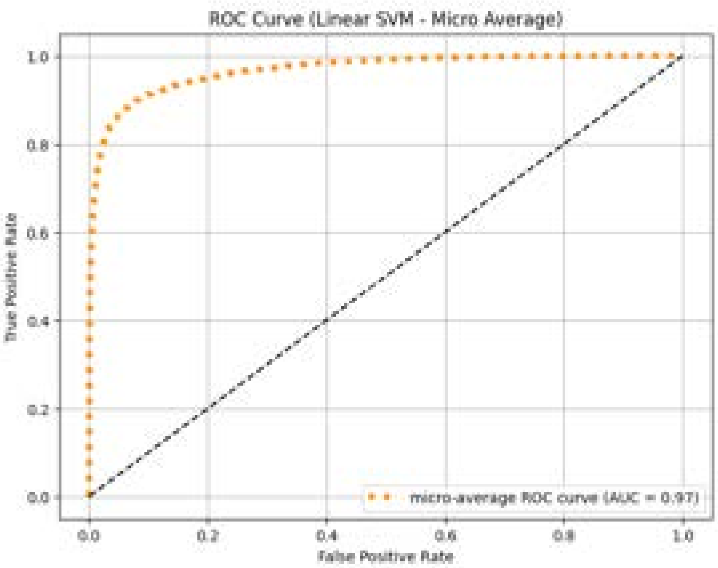

```
Linear Kernel Accuracy: 1.0000
RBF Kernel Accuracy: 1.0000
```

Letter recognition

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, label_binarize
from sklearn.svm import LinearSVC
from sklearn.calibration import CalibratedClassifierCV
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve,
auc
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_csv("/content/drive/MyDrive/MLlab
dataset/letter-recognition.csv")
X = df.drop("letter", axis=1)
y = df["letter"]
classes = sorted(y.unique())
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
y_bin = label_binarize(y, classes=classes)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,
test_size=0.2, random_state=42)
svc = LinearSVC(max_iter=10000)
calibrated_svc = CalibratedClassifierCV(svc, cv=3)
calibrated_svc.fit(X_train, y_train)
y_pred = calibrated_svc.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=False, cmap="Blues")
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
y_test_bin = label_binarize(y_test, classes=classes)
y_score = calibrated_svc.predict_proba(X_test)
from sklearn.metrics import roc_curve, auc
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(len(classes)):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[:, i])
```

```
    roc_auc[i] = auc(fpr[i], tpr[i])
fpr["micro"], tpr["micro"], _ = roc_curve(y_test_bin.ravel(),
y_score.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])
plt.figure(figsize=(8, 6))
plt.plot(fpr["micro"], tpr["micro"],
         label=f'micro-average ROC curve (AUC = {roc_auc["micro"]:.2f})',
         color='darkorange', linestyle=':', linewidth=4)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve (Linear SVM - Micro Average)")
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
print(f"Accuracy: {accuracy:.4f}")
```






```
Accuracy: 0.7288
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import roc_auc_score, roc_curve
from sklearn.multiclass import OneVsRestClassifier
from sklearn.preprocessing import label_binarize
import numpy as np
# Load letter-recognition dataset
letter_df = pd.read_csv("/content/drive/MyDrive/MLlab
dataset/letter-recognition.csv")
# First column is the label (letter), rest are features
X_letter = letter_df.iloc[:, 1:]
y_letter = letter_df.iloc[:, 0]
# Encode letter labels
y_letter_encoded = label_encoder.fit_transform(y_letter)
# Standardize features
scaler = StandardScaler()
X_letter_scaled = scaler.fit_transform(X_letter)
# Binarize labels for ROC/AUC
y_letter_binarized = label_binarize(y_letter_encoded,
classes=np.unique(y_letter_encoded))
# Train-test split
X_train_letter, X_test_letter, y_train_letter, y_test_letter =
train_test_split(
    X_letter_scaled, y_letter_encoded, test_size=0.2, random_state=42
)
# Use RBF kernel for classification
svm_letter = SVC(kernel='rbf', probability=True)
```

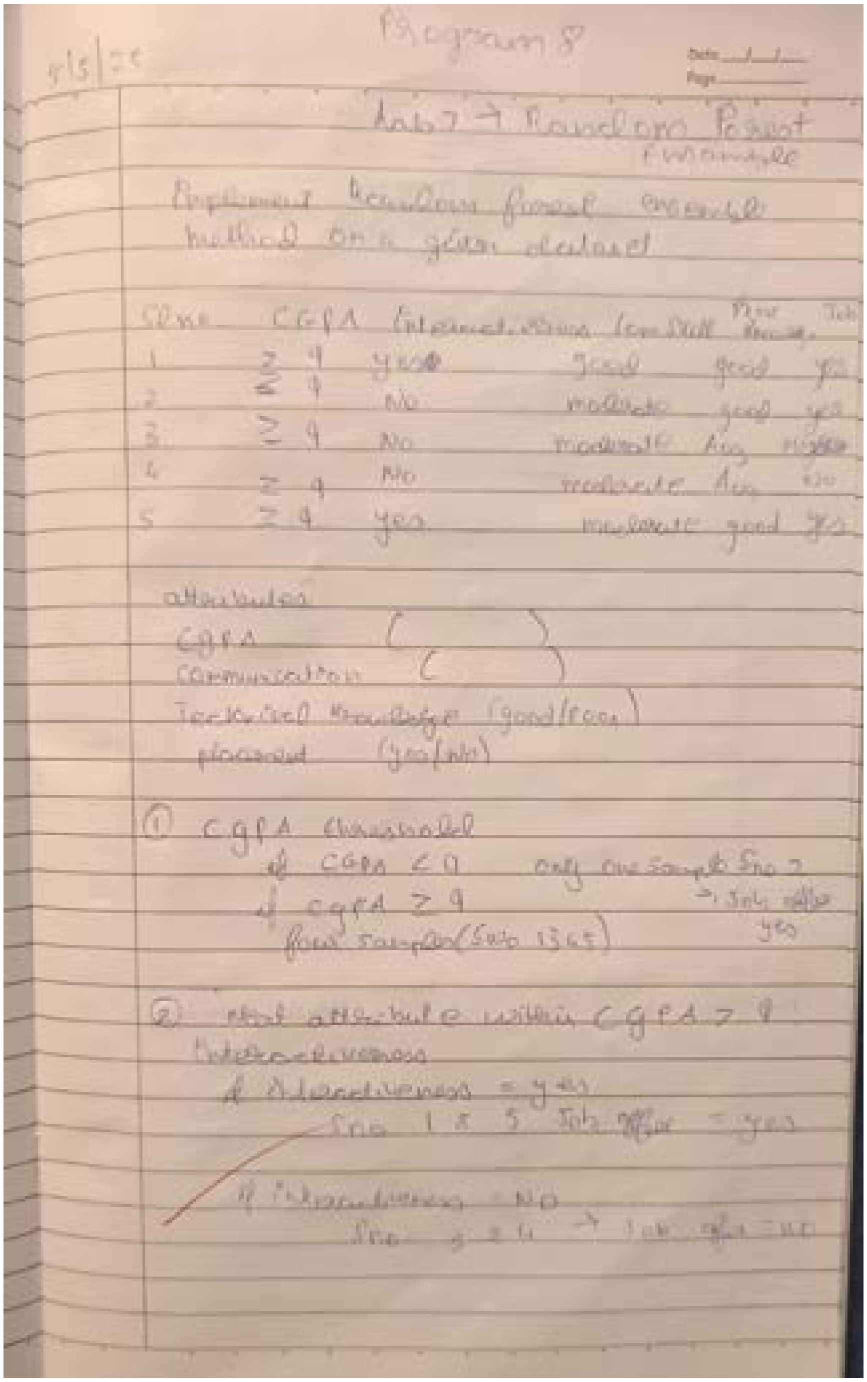
```python
svm_letter.fit(X_train_letter, y_train_letter)
y_pred_letter = svm_letter.predict(X_test_letter)
y_pred_proba = svm_letter.predict_proba(X_test_letter)
# Accuracy and Confusion Matrix
acc_letter = accuracy_score(y_test_letter, y_pred_letter)
cm_letter = confusion_matrix(y_test_letter, y_pred_letter)
# One-vs-Rest AUC score
classifier_ovr = OneVsRestClassifier(SVC(kernel='rbf', probability=True))
classifier_ovr.fit(X_train_letter, label_binarize(y_train_letter,
classes=np.unique(y_letter_encoded)))
y_score = classifier_ovr.decision_function(X_test_letter)
auc_score = roc_auc_score(label_binarize(y_test_letter,
classes=np.unique(y_letter_encoded)), y_score, average="macro")
acc_letter, cm_letter, auc_score
```

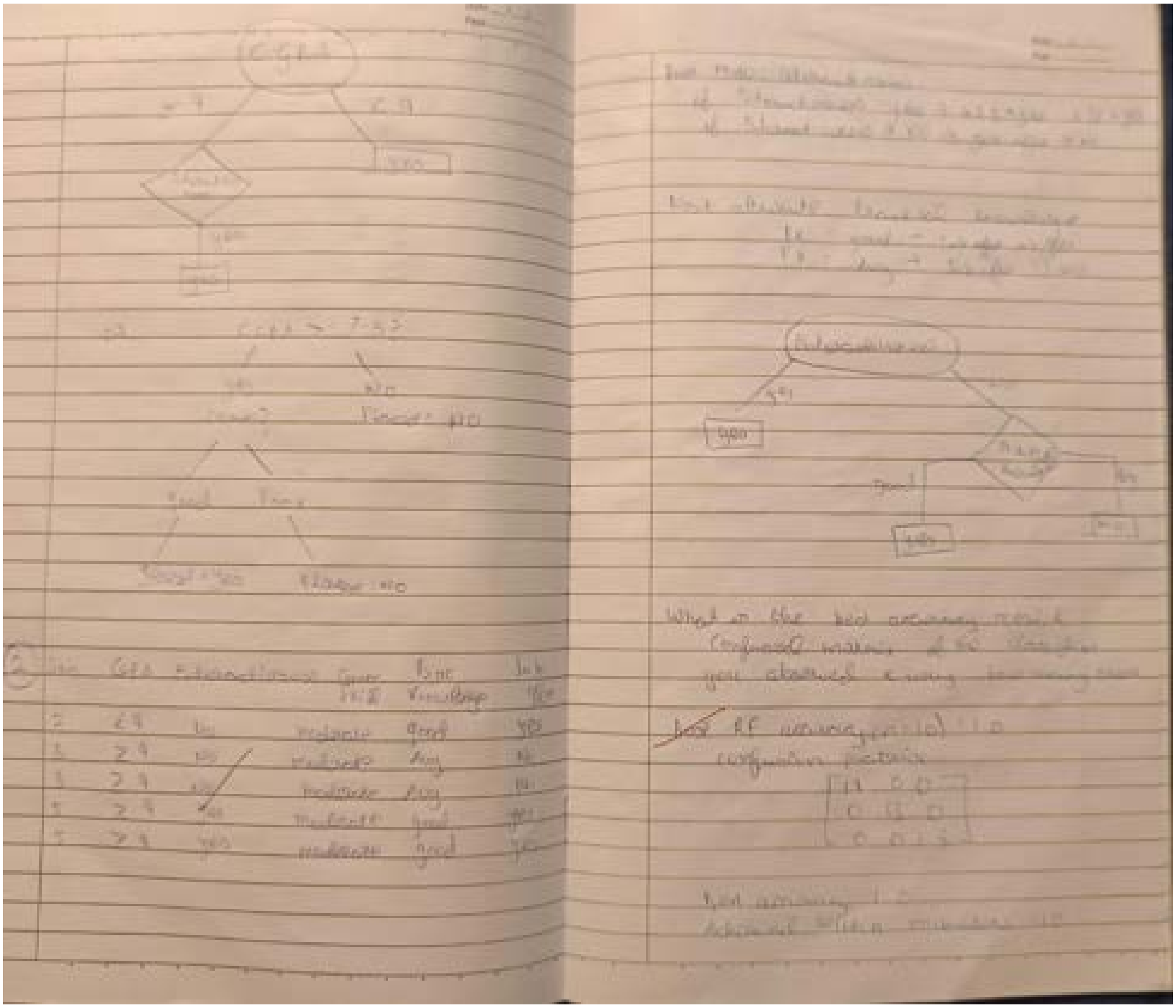
```
(0.95025,
 array([[148,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   1,
           0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
        [  1, 147,   0,   3,   0,   0,   0,   0,   0,   0,   0,   0,   0,
           0,   0,   0,   0,   2,   0,   0,   0,   0,   0,   0,   0,   0],
        [  0,   0, 126,   0,   2,   0,   2,   1,   0,   0,   2,   0,   0,
           0,   4,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
        [  0,   1,   0, 153,   0,   0,   0,   1,   0,   0,   0,   0,   0,
           1,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
        [  0,   0,   1,   0, 135,   0,   3,   0,   0,   0,   0,   0,   0,
           0,   0,   0,   1,   0,   0,   0,   0,   0,   0,   0,   0,   1],
        [  0,   1,   0,   0,   1, 134,   0,   0,   1,   0,   0,   0,   0,
           0,   0,   1,   0,   0,   0,   2,   0,   0,   0,   0,   0,   0],
        [  0,   0,   1,   3,   0,   0, 153,   0,   0,   0,   0,   0,   0,
           0,   0,   0,   0,   2,   0,   0,   0,   1,   0,   0,   0,   0],
        [  0,   3,   0,   4,   0,   0,   1, 118,   0,   0,   4,   0,   0,
           1,   2,   0,   0,   8,   0,   0,   1,   0,   0,   1,   1,   0],
        [  0,   0,   0,   0,   0,   2,   0,   0, 136,   7,   0,   0,   0,
           0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   1,   0,   0],
        [  1,   0,   0,   0,   1,   1,   0,   0,   2, 143,   0,   0,   0,
           0,   1,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   1,   0,   0, 117,   0,   0,
           0,   0,   0,   0,  10,   0,   0,   0,   0,   0,   2,   0,   0],
        [  0,   0,   1,   0,   4,   0,   3,   0,   0,   0,   0, 144,   0,
           0,   0,   0,   0,   1,   1,   0,   0,   0,   0,   0,   0,   1],
        [  0,   2,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 166,
           0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
        [  0,   0,   0,   2,   0,   0,   0,   1,   0,   0,   0,   0,   0,
         140,   4,   0,   0,   3,   0,   0,   0,   0,   1,   0,   0,   0],
        [  0,   0,   0,   4,   0,   0,   1,   0,   0,   0,   0,   0,   0,
           0, 136,   0,   1,   0,   0,   0,   1,   0,   2,   0,   0,   0],
        [  0,   1,   0,   0,   2,  11,   3,   0,   0,   0,   0,   1,   0,
           0,   1, 154,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
        [  0,   1,   0,   1,   1,   0,   0,   0,   0,   0,   0,   0,   0,
           0,   0,   0, 163,   0,   0,   0,   0,   0,   0,   0,   0,   0],
        [  0,   5,   0,   2,   0,   0,   0,   0,   0,   0,   2,   0,   0,
           1,   0,   0,   0, 150,   0,   0,   0,   0,   0,   0,   0,   0],
        [  0,   2,   0,   0,   1,   0,   0,   0,   0,   0,   0,   0,   0,
           0,   0,   0,   0,   0, 167,   0,   0,   0,   0,   0,   0,   1],
        [  0,   1,   0,   0,   0,   0,   1,   0,   0,   0,   1,   0,   0,
           0,   0,   0,   0,   1,   0, 155,   1,   0,   0,   2,   1,   0],
        [  0,   1,   0,   0,   0,   0,   0,   0,   0,   0,   2,   0,   1,
           0,   0,   0,   0,   0,   0,   0, 175,   1,   3,   0,   0,   0],
        [  0,   4,   0,   0,   0,   1,   0,   0,   0,   0,   0,   0,   1,
           0,   0,   0,   0,   2,   0,   0,   0, 150,   0,   0,   0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
           0,   0,   0,   0,   0,   0,   0,   2,   0, 146,   0,   0,   0],
        [  0,   0,   0,   2,   0,   0,   0,   0,   0,   0,   0,   0,   0,
           0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 152,   0,   0],
        [  1,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
           0,   0,   0,   0,   0,   0,   1,   0,   0,   0,   0, 166,   0],
        [  0,   0,   0,   0,   1,   0,   0,   0,   0,   2,   0,   0,   0,
           0,   0,   0,   0,   1,   1,   0,   0,   0,   0,   0,   0, 127]]),
 np.float64(0.9967299152393889))
```

Program 8

Implement Random forest ensemble method on a given dataset.

Screenshots:





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
import matplotlib.pyplot as plt

# Step 1: Load dataset
df = pd.read_csv("/content/drive/MyDrive/MLlab dataset/iris (1).csv")

# Step 2: Prepare features and labels
X = df.drop("species", axis=1)
y = df["species"]

# Step 3: Split into training and test data
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

# Step 4: Train with default n_estimators=10
clf_default = RandomForestClassifier(n_estimators=10, random_state=42)
clf_default.fit(X_train, y_train)
y_pred_default = clf_default.predict(X_test)

# Evaluation
default_accuracy = accuracy_score(y_test, y_pred_default)
default_cm = confusion_matrix(y_test, y_pred_default)

print("Default RF Accuracy (n=10):", default_accuracy)
print("Confusion Matrix:\n", default_cm)

# Step 5: Fine-tune number of estimators
accuracies = []
tree_range = range(10, 101, 10)

for n in tree_range:
    clf = RandomForestClassifier(n_estimators=n, random_state=42)
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
```

```python
    accuracies.append(acc)

# Step 6: Find best accuracy
best_accuracy = max(accuracies)
best_n = tree_range[accuracies.index(best_accuracy)]

print("\nBest Accuracy:", best_accuracy)
print("Achieved with n_estimators =", best_n)

# Step 7: Plot accuracy vs number of trees
plt.figure(figsize=(10, 5))
plt.plot(tree_range, accuracies, marker='o')
plt.title("Accuracy vs Number of Trees in Random Forest")
plt.xlabel("Number of Trees")
plt.ylabel("Accuracy")
plt.grid(True)
plt.show()
```

Default RF Accuracy (n=10): 1.0

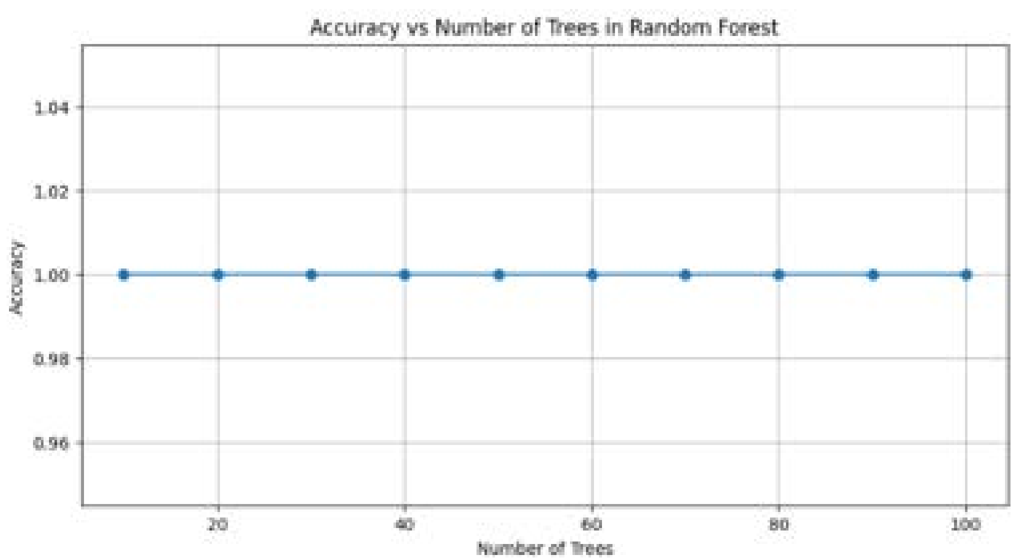
Confusion Matrix:

[[19 0 0]

[0 13 0]

[0 0 13]]

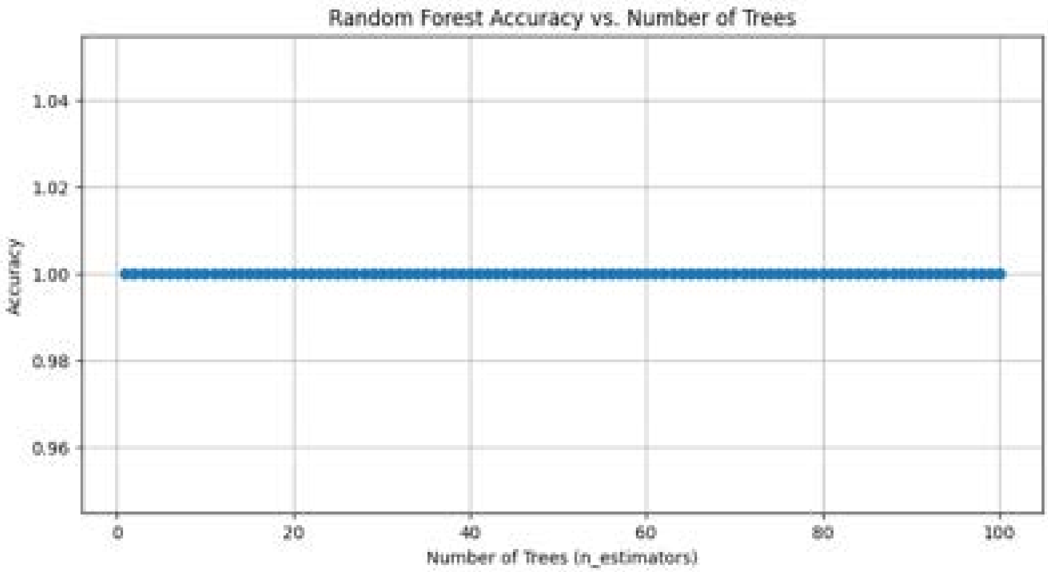
Best Accuracy: 1.0
Achieved with n_estimators = 10



different code with estimator changed

```
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
iris = load_iris()
X, y = iris.data, iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
rf_default = RandomForestClassifier(random_state=42,n_estimators=10)
rf_default.fit(X_train, y_train)
y_pred_default = rf_default.predict(X_test)
default_score = accuracy_score(y_test, y_pred_default)
print(f"Default (n_estimators=10) Accuracy: {default_score:.4f}")
scores = []
tree_range = range(1, 101)
for n in tree_range:
    rf = RandomForestClassifier(n_estimators=n, random_state=42)
    rf.fit(X_train, y_train)
    y_pred = rf.predict(X_test)
    scores.append(accuracy_score(y_test, y_pred))
best_score = max(scores)
best_n = tree_range[scores.index(best_score)]
print(f"Best Accuracy: {best_score:.4f} with n_estimators={best_n}")
plt.figure(figsize=(10, 5))
plt.plot(tree_range, scores, marker='o')
plt.xlabel('Number of Trees (n_estimators)')
plt.ylabel('Accuracy')
plt.title('Random Forest Accuracy vs. Number of Trees')
plt.grid(True)
plt.show()
```

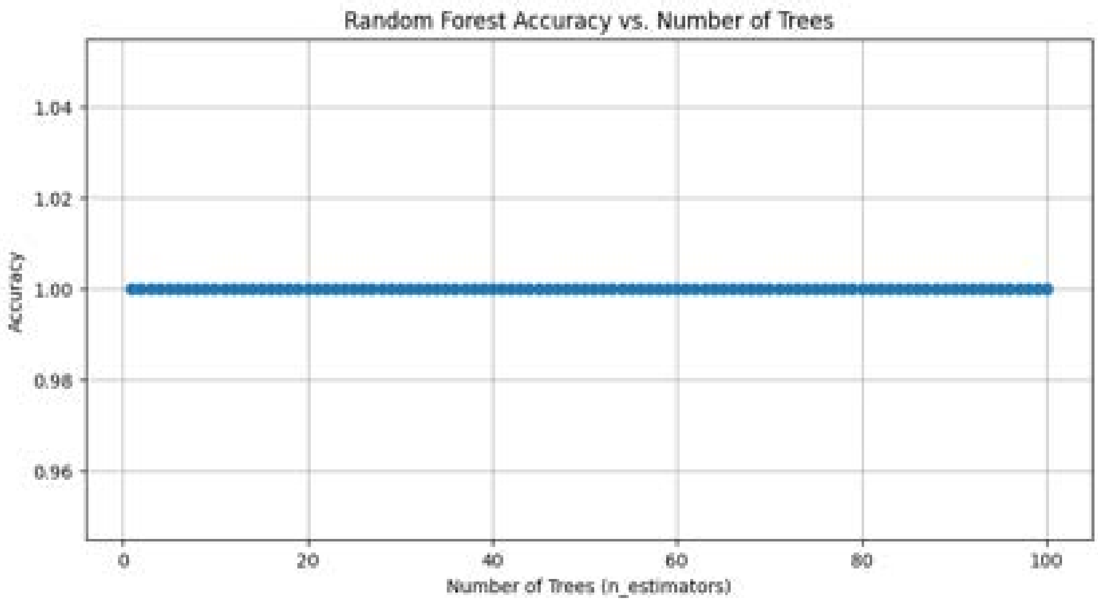
```
Default (n_estimators=10) Accuracy: 1.0000
Best Accuracy: 1.0000 with n_estimators=1
```




```
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
iris = load_iris()
X, y = iris.data, iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
rf_default = RandomForestClassifier(random_state=42,n_estimators=100)
rf_default.fit(X_train, y_train)
y_pred_default = rf_default.predict(X_test)
default_score = accuracy_score(y_test, y_pred_default)
print(f" (n_estimators=100) Accuracy: {default_score:.4f}")
scores = []
tree_range = range(1, 101)
for n in tree_range:
    rf = RandomForestClassifier(n_estimators=n, random_state=42)
    rf.fit(X_train, y_train)
    y_pred = rf.predict(X_test)
    scores.append(accuracy_score(y_test, y_pred))
best_score = max(scores)
best_n = tree_range[scores.index(best_score)]
print(f"Best Accuracy: {best_score:.4f} with n_estimators={best_n}")
```

```python
plt.figure(figsize=(10, 5))
plt.plot(tree_range, scores, marker='o')
plt.xlabel('Number of Trees (n_estimators)')
plt.ylabel('Accuracy')
plt.title('Random Forest Accuracy vs. Number of Trees')
plt.grid(True)
plt.show()
```

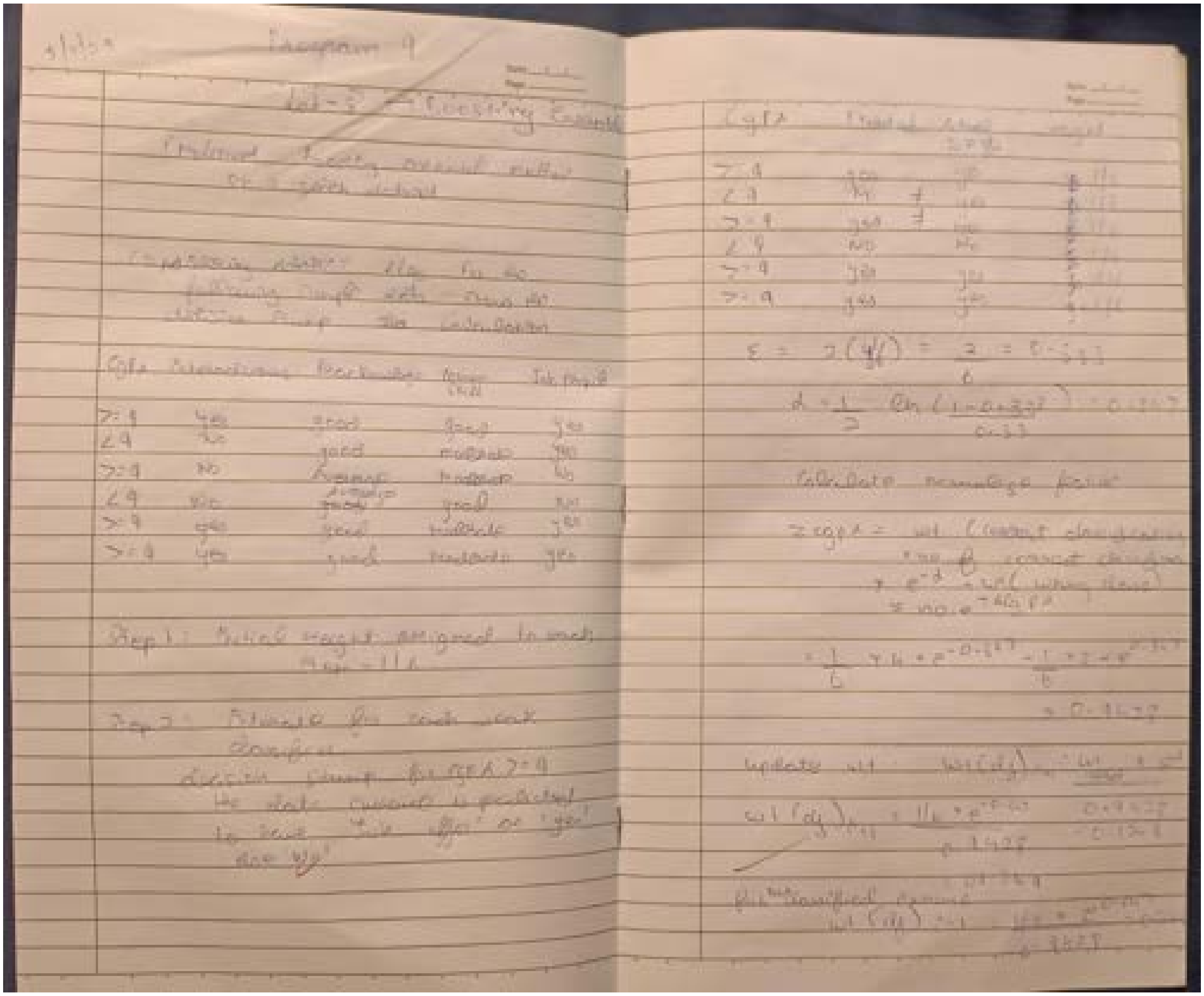
```
(n_estimators=100) Accuracy: 1.0000
Best Accuracy: 1.0000 with n_estimators=1
```

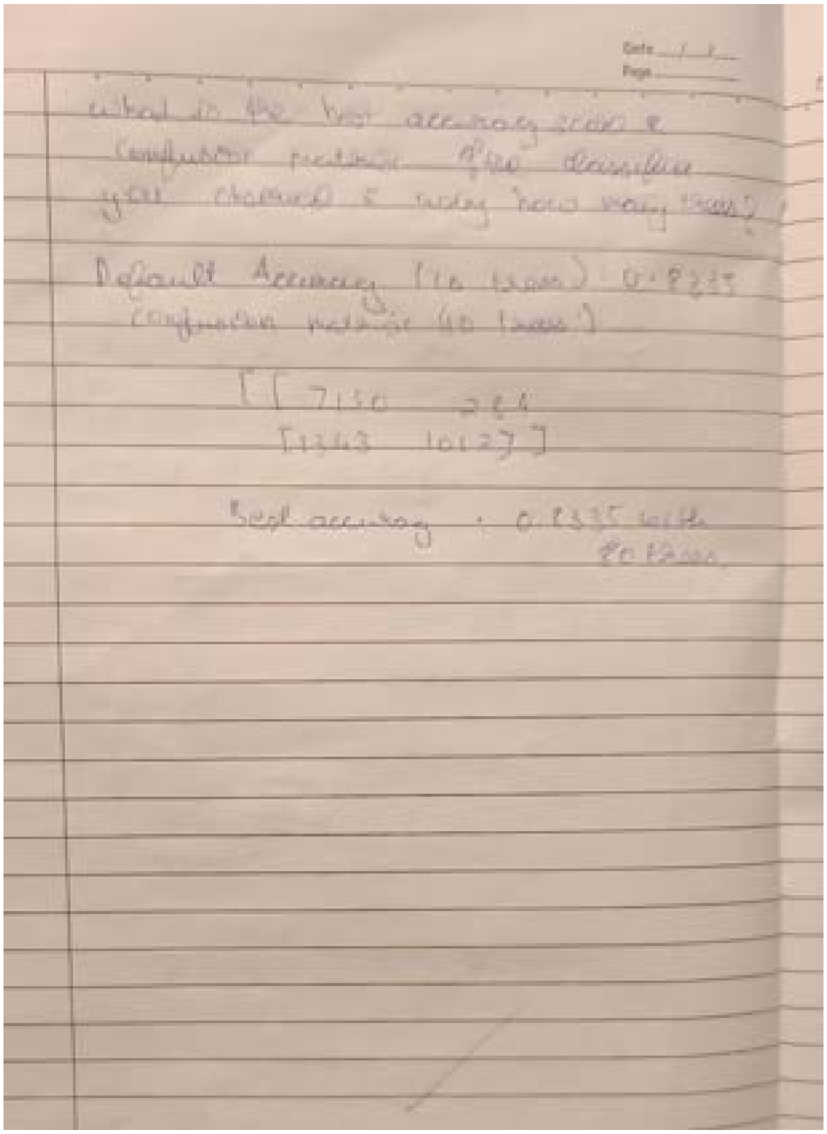


Program 9

Implement Boosting ensemble method on a given dataset.

Screenshots:





What is the best accuracy score & confusion matrix of the classifier you choose & using how many trees?

Default Accuracy (10 trees) : 0.8235
Confusion matrix (40 trees)

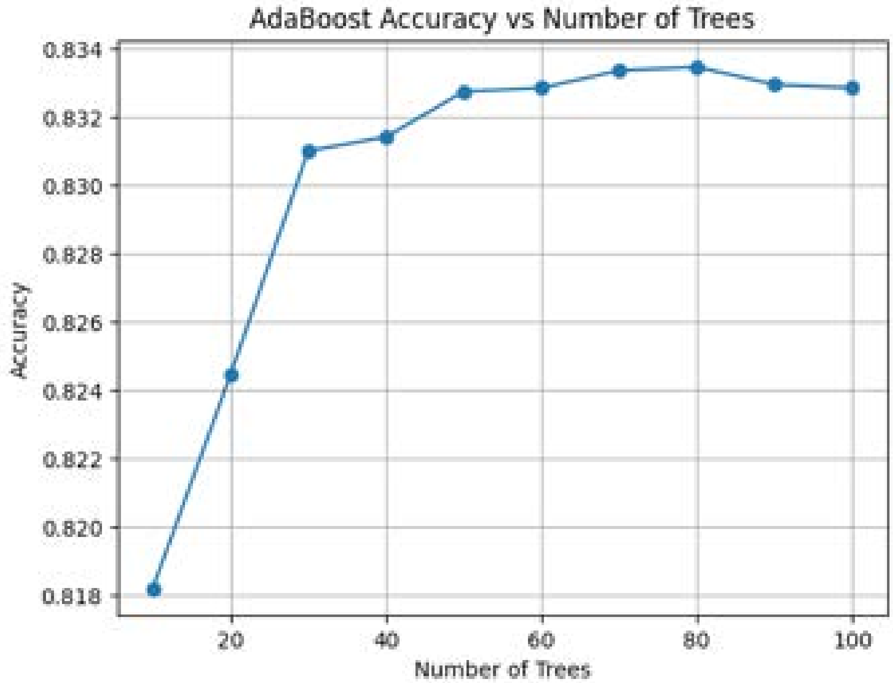
[[7150 288]
[1343 10127]]

Best accuracy : 0.8335 with 80 trees

```python
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
df = pd.read_csv('/content/drive/MyDrive/MLlab dataset/income.csv')
df = df.dropna()
label_encoders = {}
for column in df.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    df[column] = le.fit_transform(df[column])
    label_encoders[column] = le
```

```
X = df.drop('income_level', axis=1)
y = df['income_level']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
model_default = AdaBoostClassifier(n_estimators=80, random_state=42)
model_default.fit(X_train, y_train)
y_pred_default = model_default.predict(X_test)
accuracy_default = accuracy_score(y_test, y_pred_default)
conf_matrix_default = confusion_matrix(y_test, y_pred_default)
print(f"Default Accuracy (10 trees): {accuracy_default:.4f}")
print("Confusion Matrix (10 trees):")
print(conf_matrix_default)
best_accuracy = 0
best_n = 0
accuracy_scores = []
for n in range(10, 101, 10):
    model = AdaBoostClassifier(n_estimators=n, random_state=42)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    accuracy_scores.append(acc)
    if acc > best_accuracy:
        best_accuracy = acc
        best_n = n
print(f"\nBest Accuracy: {best_accuracy:.4f} with {best_n} trees")
plt.plot(range(10, 101, 10), accuracy_scores, marker='o')
plt.title('AdaBoost Accuracy vs Number of Trees')
plt.xlabel('Number of Trees')
plt.ylabel('Accuracy')
plt.grid(True)
plt.show()
```

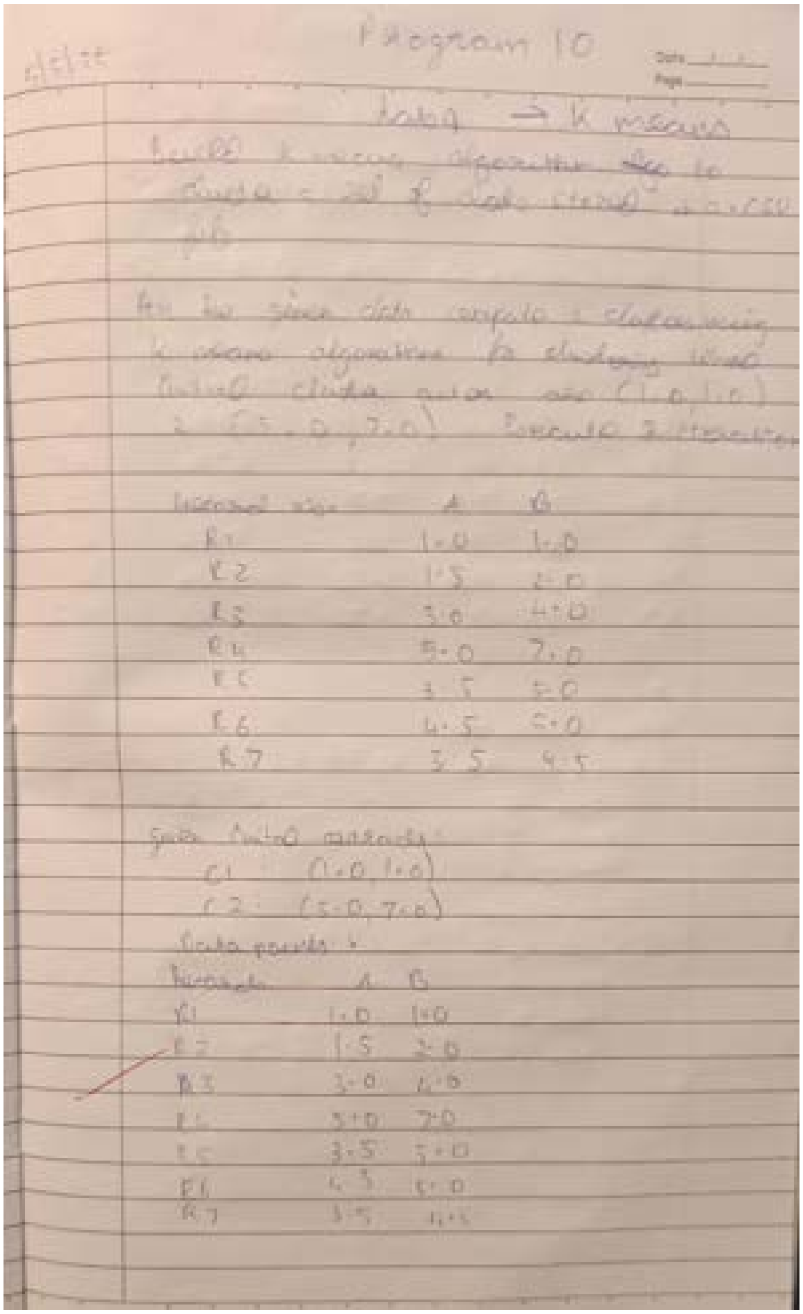
```
Default Accuracy (10 trees): 0.8335
Confusion Matrix (10 trees):
[[7130  284]
 [1343 1012]]
Best Accuracy: 0.8335 with 80 trees
```

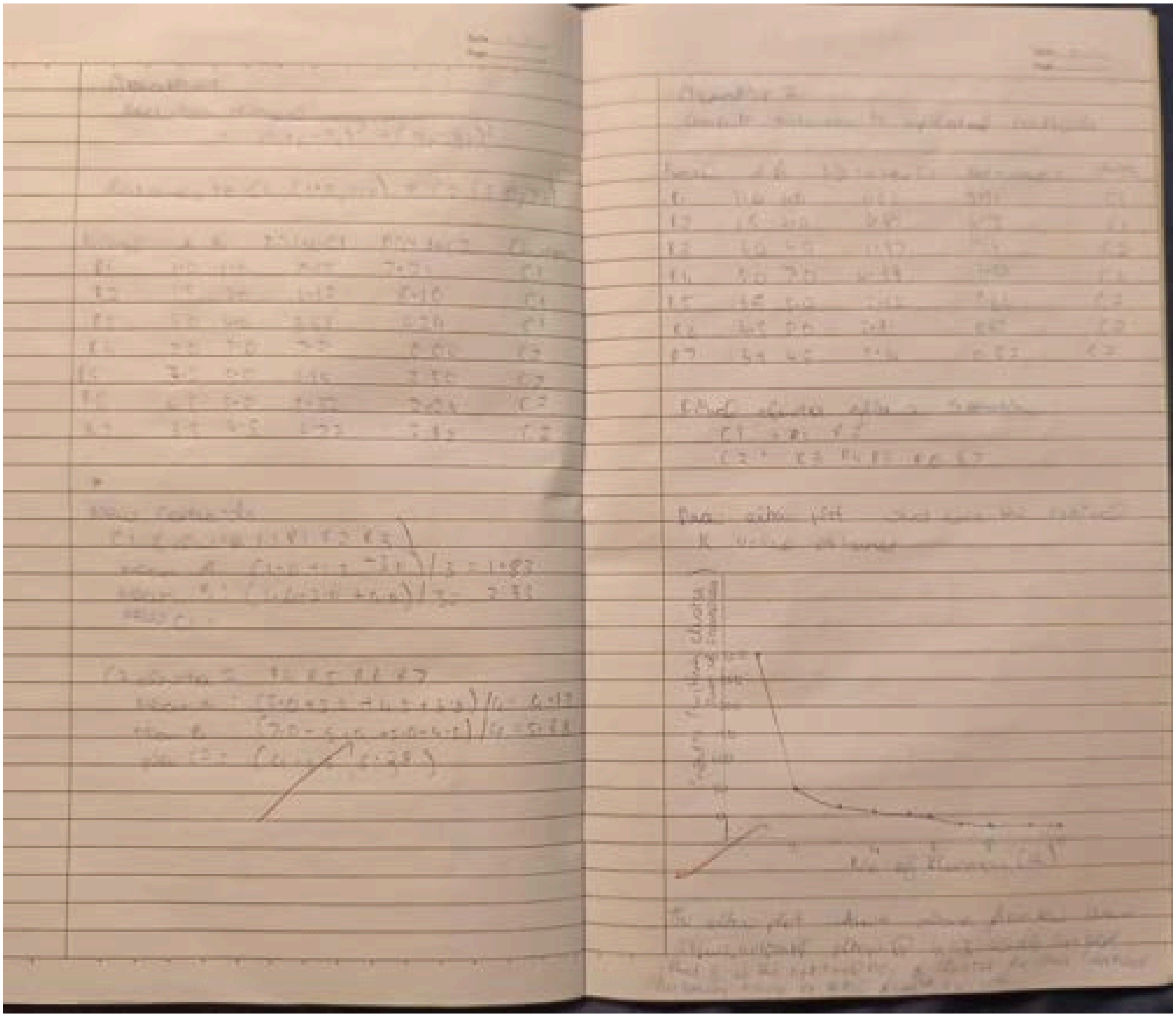

AdaBoost Accuracy vs Number of Trees
0.834
0.832
0.830
0.828
0.826
0.824
0.822
0.820
0.818
Accuracy
20
40
60
80
100
Number of Trees

Program 10

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

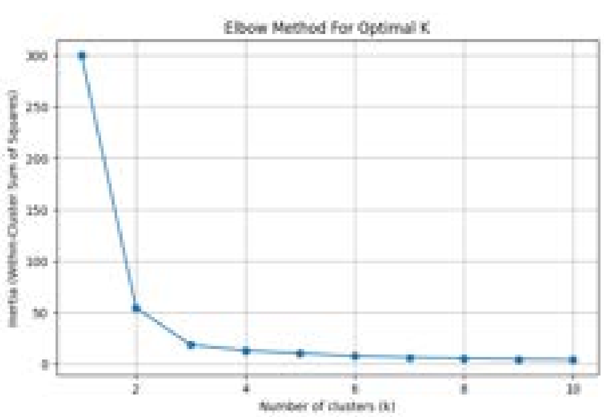
Screenshots:





Code:

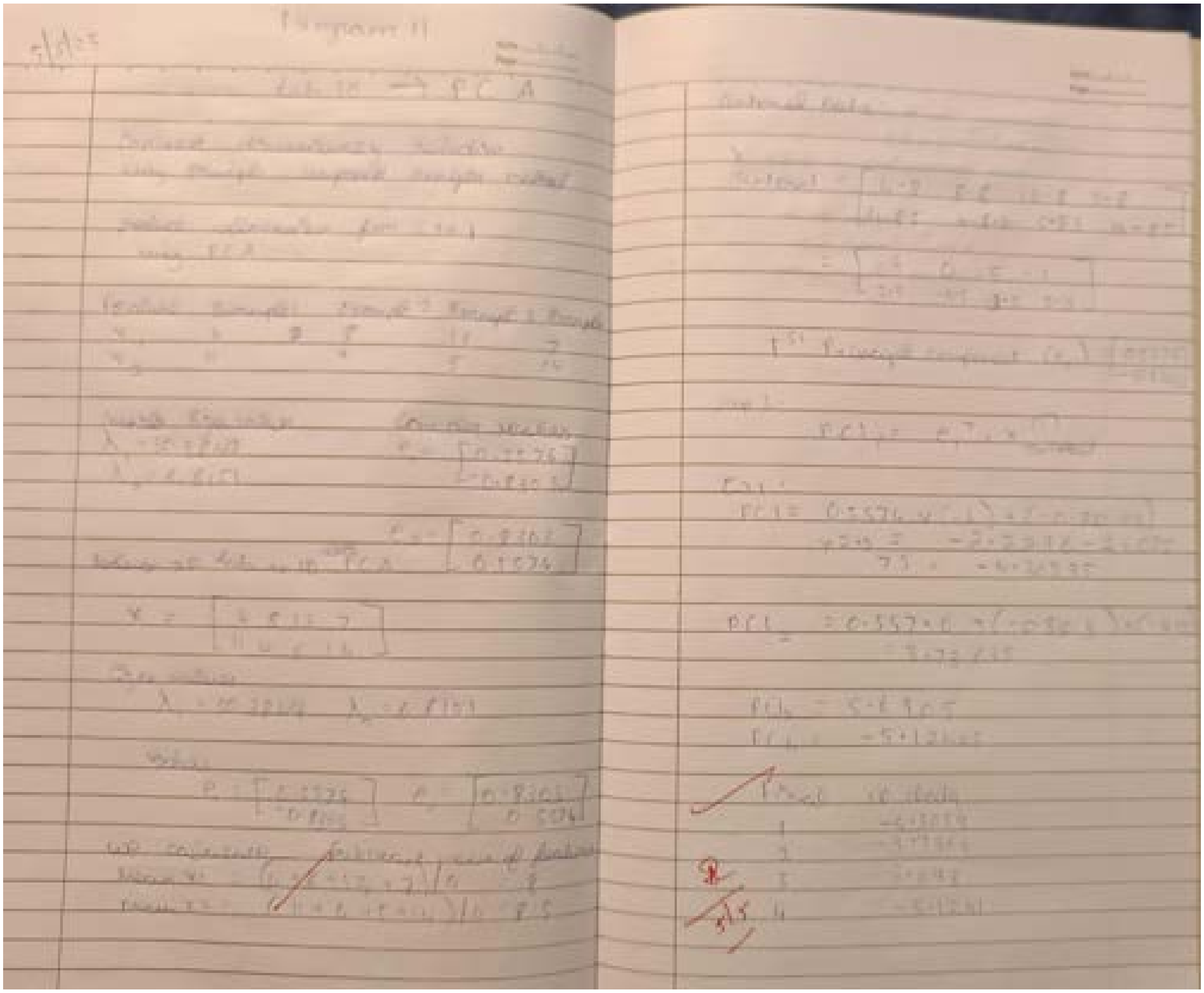
```
# Import necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
# Load the dataset
df = pd.read_csv('/content/drive/MyDrive/MLlab dataset/iris (1) (1).csv')
# Select only petal length and width
X = df[['petal_length', 'petal_width']]
# Optional: Scaling the data helps K-Means perform better
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Elbow Method to find optimal k
inertia = []
K = range(1, 11)
for k in K:
    model = KMeans(n_clusters=k, random_state=42)
    model.fit(X_scaled)
    inertia.append(model.inertia_)
# Plot elbow graph
plt.figure(figsize=(8, 5))
plt.plot(K, inertia, marker='o')
plt.title('Elbow Method For Optimal K')
plt.xlabel('Number of clusters (k)')
plt.ylabel('Inertia (Within-Cluster Sum of Squares)')
plt.grid(True)
plt.show()
```

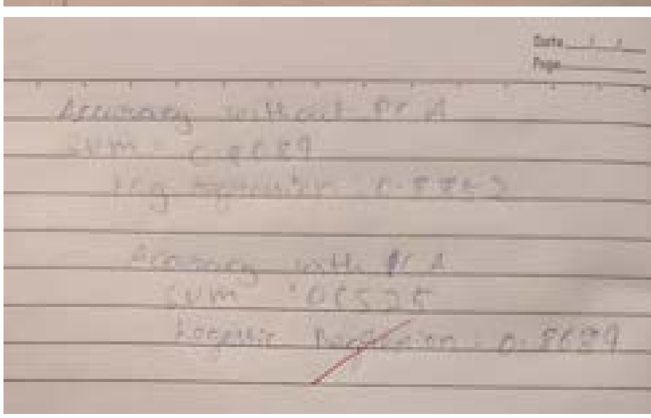


Program 11

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

Screenshots:





Code:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, OneHotEncoder,
StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
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
# Step 1: Load data
df = pd.read_csv("/content/drive/MyDrive/MLlab dataset/heart.csv")
# Step 2: Identify categorical columns
categorical_cols = df.select_dtypes(include=['object']).columns.tolist()
# Step 3: Encode categorical columns using Label Encoding (for binary)
or OneHotEncoding (for >2 categories)
df_encoded = df.copy()
label_enc = LabelEncoder()
for col in categorical_cols:
    if df_encoded[col].nunique() == 2:
        df_encoded[col] = label_enc.fit_transform(df_encoded[col])
    else:
        df_encoded = pd.get_dummies(df_encoded, columns=[col])
# Step 4: Separate features and target
X = df_encoded.drop('target', axis=1)
y = df_encoded['target']
# Step 5: Scale the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Step 6: Split into train/test sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,
test_size=0.2, random_state=42)
# Step 7: Train and evaluate models
models = {
    "Logistic Regression": LogisticRegression(),
    "SVM": SVC(),
```

```python
    "Random Forest": RandomForestClassifier()
}
print("=== Accuracy Without PCA ===")
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    print(f"{name}: {accuracy_score(y_test, y_pred):.4f}")
# Step 8: Apply PCA to reduce dimensionality
pca = PCA(n_components=0.95)  # Retain 95% variance
X_pca = pca.fit_transform(X_scaled)
# Step 9: Split PCA-transformed data
X_train_pca, X_test_pca, y_train, y_test = train_test_split(X_pca, y,
test_size=0.2, random_state=42)
print("\n=== Accuracy With PCA ===")
for name, model in models.items():
    model.fit(X_train_pca, y_train)
    y_pred = model.predict(X_test_pca)
    print(f"{name}: {accuracy_score(y_test, y_pred):.4f}")
```

```
=== Accuracy Without PCA ===
Logistic Regression: 0.8525
SVM: 0.8689
Random Forest: 0.8689

=== Accuracy With PCA ===
Logistic Regression: 0.8525
SVM: 0.8361
Random Forest: 0.8525
```