VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

Tamanna Rukhaya (1BM22CS301)

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 **Tamanna Rukhaya (1BM22CS301),** 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.

Spoorthi D M Assistant Professor Department of CSE, BMSCE	Dr. Kavitha Sooda Professor & HOD Department of CSE, BMSCE

Index

Github Link:

https://github.com/TamannaRukhayaa/ml_lab_6_sem

Program 1

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

Code:

```
import pandas as pd

# Method-1: Initializing values directly into DataFrame

data_method1 = {'USN': ['1JS17CS001', '1JS17CS002', '1JS17CS003', '1JS17CS004',
'1JS17CS005'],

'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve'],  'Marks': [90, 85, 92, 78, 88]}

df_method1 = pd.DataFrame(data_method1)

print("Method-1:")

print(df_method1)

print("-" * 20)



# Method-2: Importing datasets from sklearn.datasets

from sklearn.datasets import load_diabetes

diabetes_data = load_diabetes()

df_method2 = pd.DataFrame(data=diabetes_data.data,
columns=diabetes_data.feature_names)

df_method2['target'] = diabetes_data.target


print("Method-2:")

print(df_method2.head())

print("-" * 20)
```

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

try:

df_method3 = pd.read_csv('sample_sales_data.csv')

 print("Method-3:")

 print(df_method3.head())

 print("-" * 20)

 except FileNotFoundError:

 print("sample_sales_data.csv not found. Please upload the  file.")  print("-" * 20)


import yfinance as yf


import matplotlib.pyplot as plt



tickers = ["HDFCBANK.NS", "ICICIBANK.NS", "KOTAKBANK.NS"]

start_date = "2024-01-01"

end_date = "2024-12-30"


data = yf.download(tickers, start=start_date, end=end_date) closing_prices =

data['Close']

daily_returns = closing_prices.pct_change().dropna()


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

closing_prices.plot()
```

```python
plt.title('Closing Prices (2024)')

plt.xlabel('Date')

plt.ylabel('Price (INR)')

plt.grid(True)

plt.show()


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

daily_returns.plot()

plt.title('Daily Returns (2024)')

plt.xlabel('Date')

plt.ylabel('Daily Return')

plt.grid(True)

plt.show()
```

Lab-1 03.03.2(

import pandas as pd

1. df = Pd. read_csv (housing.csv)

Print(df)

2. df2 = df. info()

Print(df2)

3. df3 = df. describe()

Print (df3)

4. df4 = df. value_counts()

Print (df4)

5. df5 = df.isnull(). sum()

df6 = df5 [df5 >0]

Print(df6)

RG 5/3/25

Ans 3 Min Max Scaling

$$\frac{x_i - \min(x_i)}{\max(x_i) - \min(x_i)}$$

[0 . . . 1] [-1 — 1]

It is used for Bounded Algorithms.

Standardization Scaling

$$\frac{x_i - mean(x_i)}{std(x_i)}$$

x_i → actual value w.r.t scaling

$\hat{x}_i$ → Predicted value w.r.t Scaling

Ans 1 In diabetes dataset, none of the columns had missing values.

In adult dataset we had missing values

workclass	2799
occupation	2809
native-country	857

Ans 2

Diabetes Dataset has 'gender' and 'class'

Adult Dataset had almost all categori-cal columns using label encoding these categorical columns were encoded to numeric values or one hot encoding

RY 5/3/25

10/10

Program 2

Demonstrate various data pre-processing techniques for a given dataset

Screenshots:

Code:

```
#diabetes dataset
import pandas as pd
import io

df = pd.read_csv("diabetes.csv")
print(df.head())  # Display first 5 rows
print('-----------------------------')
#Handling missing values
df.dropna(inplace = True)
df.drop_duplicates(inplace = True)

#Handling categorical data
from sklearn.preprocessing import LabelEncoder

# Encode Gender column (Male = 0, Female = 1)
label_encoder = LabelEncoder()
df['Gender'] = label_encoder.fit_transform(df['Gender'])

# Check the unique values after encoding
print(df['Gender'].unique())

#Handling outliers
from scipy import stats

# Calculate Z-scores for the numerical columns
z_scores = stats.z_score(df[['AGE', 'Urea', 'Cr', 'HbA1c', 'Chol', 'TG', 'HDL', 'LDL', 'VLDL', 'BMI']])

# Set a threshold for Z-scores (e.g., 3 standard deviations)
df_no_outliers = df[(z_scores < 3).all(axis=1)]

# Calculate IQR for each numerical column
Q1 = df[['AGE', 'Urea', 'Cr', 'HbA1c', 'Chol', 'TG', 'HDL', 'LDL', 'VLDL', 'BMI']].quantile(0.25)
Q3 = df[['AGE', 'Urea', 'Cr', 'HbA1c', 'Chol', 'TG', 'HDL', 'LDL', 'VLDL', 'BMI']].quantile(0.75)
IQR = Q3 - Q1
```

```python
# Remove rows with outliers
df_no_outliers = df[~((df[['AGE', 'Urea', 'Cr', 'HbA1c', 'Chol', 'TG', 'HDL', 'LDL', 'VLDL', 'BMI']] < (Q1 - 1.5 * IQR))
| (df[['AGE', 'Urea', 'Cr', 'HbA1c', 'Chol', 'TG', 'HDL', 'LDL', 'VLDL', 'BMI']] > (Q3 + 1.5 * IQR))).any(axis=1)]

#min max scaler
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

# Apply Min-Max scaling to the numerical columns
df[['AGE', 'Urea', 'Cr', 'HbA1c', 'Chol', 'TG', 'HDL', 'LDL', 'VLDL', 'BMI']] = scaler.fit_transform(
   df[['AGE', 'Urea', 'Cr', 'HbA1c', 'Chol', 'TG', 'HDL', 'LDL', 'VLDL', 'BMI']])

#standard scaler
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

# Apply Standard scaling to the numerical columns
df[['AGE', 'Urea', 'Cr', 'HbA1c', 'Chol', 'TG', 'HDL', 'LDL', 'VLDL', 'BMI']] = scaler.fit_transform(
   df[['AGE', 'Urea', 'Cr', 'HbA1c', 'Chol', 'TG', 'HDL', 'LDL', 'VLDL', 'BMI']])

print(df)
```

Output:

```
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
-----------------------------
[0 1 2]
     ID  No_Pation  Gender       AGE      Urea        Cr     HbA1c      Chol  \
0    502    17975       0 -0.401144 -0.144781 -0.382672 -1.334983 -0.509436
1    735    34221       1 -3.130017 -0.212954 -0.115804 -1.334983 -0.893730
2    420    47975       0 -0.401144 -0.144781 -0.382672 -1.334983 -0.509436
3    680    87656       0 -0.401144 -0.144781 -0.382672 -1.334983 -0.509436
4    504    34223       1 -2.334096  0.673299 -0.382672 -1.334983  0.028576
..   ...      ...     ...       ...       ...       ...       ...       ...
995  200   454317       1  1.986619  2.002680  0.467970 -0.505840  2.026906
```

```
996 671    876534      1 -2.561502 -0.724254 -0.149162  1.586758 -0.586295
997 669     87654      1 -2.675205  0.673299  0.201102 -0.624289 -0.586295
998  99     24004      1 -1.765581  0.230173 -0.165842 -0.624289  0.336011
999 248     24054      1  0.053668 -0.042521 -0.032408 -0.545323 -0.816871

          TG       HDL       LDL      VLDL       BMI CLASS
0   -1.035084  1.810756 -1.085457 -0.369958 -1.124622     N
1   -0.678063 -0.158692 -0.457398 -0.342649 -1.326239     N
2   -1.035084  1.810756 -1.085457 -0.369958 -1.124622     N
3   -1.035084  1.810756 -1.085457 -0.369958 -1.124622     N
4   -0.963680 -0.613180 -0.547121 -0.397267 -1.729472     N
..        ...       ...       ...       ...       ...   ...
995 -0.463850 -0.007196 -0.726566 -0.342649  0.085078     Y
996 -0.106828 -0.764676 -0.188229  3.699116  1.536719     Y
997 -0.892276 -0.007196 -0.188229  1.705543 -0.439125     Y
998 -0.249637  0.598788  0.260385  3.316787  2.202054     Y
999 -0.463850 -0.158692  0.350107 -0.315340  0.689928     Y

[1000 rows x 14 columns]
```

```python
#adult dataset
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from scipy import stats

adult = pd.read_csv("adult.csv")
print(adult.head())  # Display first 5 rows
print('------------------------------')

adult.replace('?',np.nan,inplace = True)
print(adult.isnull().sum())

missing_values = adult.isnull().sum()

# Display columns with missing values
print(missing_values[missing_values > 0])

#handling missing values
# Initialize OrdinalEncoder
ordinal_encoder = OrdinalEncoder(categories=[["Male", "Female"]])
# Fit and transform the data
df_copy["Gender_Encoded"] = ordinal_encoder.fit_transform(df_copy[["Gender"]])
```

```
# Initialize OneHotEncoder
onehot_encoder = OneHotEncoder()

# Fit and transform the "City" column
encoded_data = onehot_encoder.fit_transform(df[["City"]])

# Convert the sparse matrix to a dense array
encoded_array = encoded_data.toarray()

# Convert to DataFrame for better visualization
encoded_df = pd.DataFrame(encoded_array, columns=onehot_encoder.get_feature_names_out(["City"]))
df_encoded = pd.concat([df_copy, encoded_df], axis=1)

df_encoded.drop("Gender", axis=1, inplace=True)
df_encoded.drop("City", axis=1, inplace=True)

print(df_encoded. head())
```

Output:

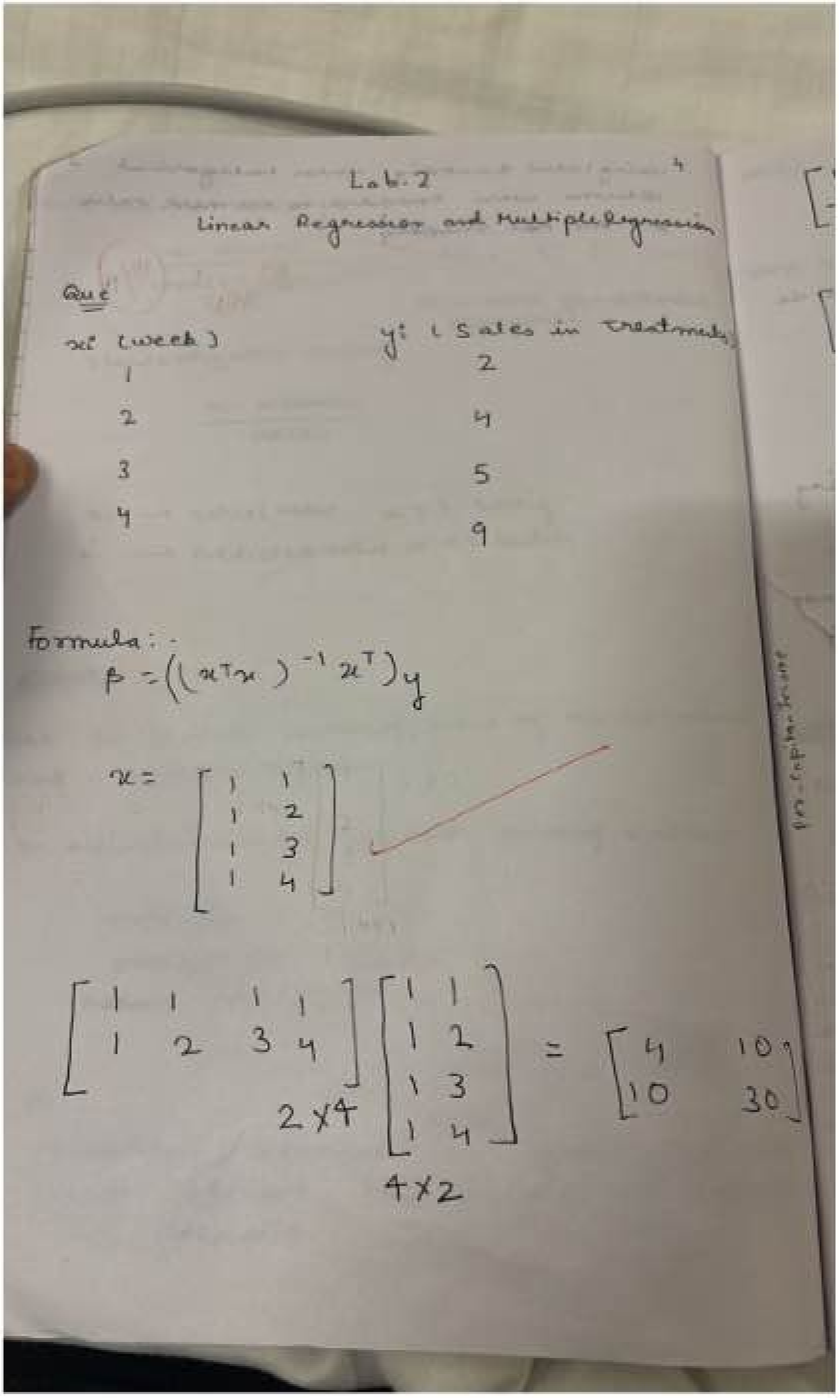
```
age  workclass  fnlwgt    education  educational-num     marital-status  \
0  25    Private  226802         11th                7      Never-married
1  38    Private   89814      HS-grad                9  Married-civ-spouse
2  28  Local-gov  336951   Assoc-acdm               12  Married-civ-spouse
3  44    Private  160323  Some-college              10  Married-civ-spouse
4  18          ?  103497  Some-college              10      Never-married

          occupation relationship   race  gender  capital-gain  capital-loss  \
0  Machine-op-inspct    Own-child  Black    Male             0             0
1    Farming-fishing      Husband  White    Male             0             0
2    Protective-serv      Husband  White    Male             0             0
3  Machine-op-inspct      Husband  Black    Male          7688             0
4                  ?    Own-child  White  Female             0             0

   hours-per-week native-country income
0              40  United-States  <=50K
1              50  United-States  <=50K
2              40  United-States   >50K
3              40  United-States   >50K
4              30  United-States  <=50K
-----------------------------
age                 0
workclass        2799
fnlwgt              0
education           0
educational-num     0
marital-status      0
occupation       2809
```

```
relationship         0
race              0
gender            0
capital-gain         0
capital-loss         0
hours-per-week        0
native-country      857
income            0
dtype: int64
workclass        2799
occupation       2809
native-country     857
dtype: int64
```

Program 3

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Screenshots:



Lab. 2

Linear Regression and Multiple Regression

Que

x_i (week)	y_i (Sales in treatment)
1	2
2	4
3	5
4	9

Formula:

$$\beta = \left((x^T x)^{-1} x^T \right) y$$

$$x = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{bmatrix}$$

$$\underset{2 \times 4}{\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \end{bmatrix}} \underset{4 \times 2}{\begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{bmatrix}} = \begin{bmatrix} 4 & 10 \\ 10 & 30 \end{bmatrix}$$

$$\begin{bmatrix} 1.5 & -0.5 \\ -0.5 & 0.2 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0.5 & 0 & -0.5 \\ -0.3 & -0.1 & 0.1 & 0.3 \end{bmatrix} \times \begin{bmatrix} 2 \\ 4 \\ 5 \\ 9 \end{bmatrix}$$

$$= \begin{bmatrix} -0.5 \\ 2.2 \end{bmatrix}$$

$\beta_0 = -0.5$

$\beta_1 = 2.2$

$y = \beta_0 + \beta_1 x$

$$\boxed{y = -0.5 + 2.2x}$$

10/3/25

Observation Book

Ans 1

For canada file, we don't need to do data processing as there are no missing values

For Salary file we did data processing as In the years-experience column we got 2 missing values

For Salary file we did data proces as In experience column, two rows no missing values and test-score we g 1 missing value

For 1000-companies we don't nee data processing as there are no m Values

Ans 2 The plot show the relations of linear regression In that per capita Income is dependent Variable and year is the Independent variable as the year will ~~inc~~, after every year there is increase in Per capita Income.

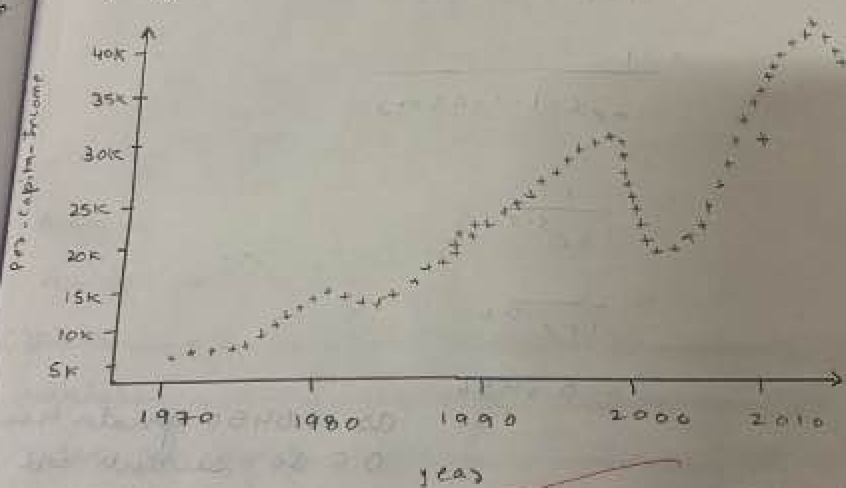
Ans 3 P

Ans 4

Ans3 Predicted salary => 90438.68025
262

Ans 4 ~~Yes I, encoded the categorical variables using one hot encoding~~
Encoding is applied for the state model using label encoder. Florida is encoded to 0 using label_encoder.transform.

graph:-



$y = mx+c$

10/10

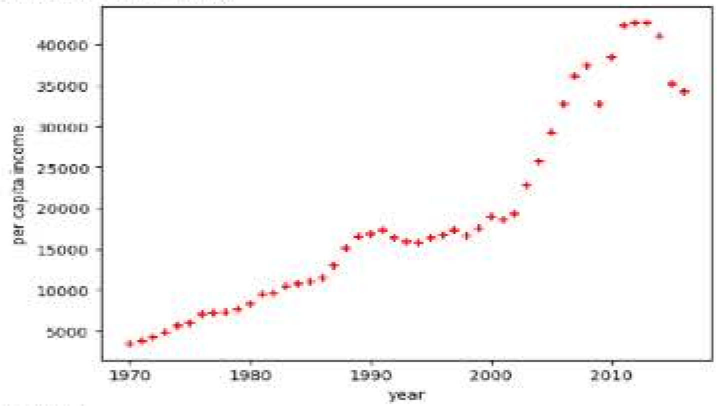
10/3/25

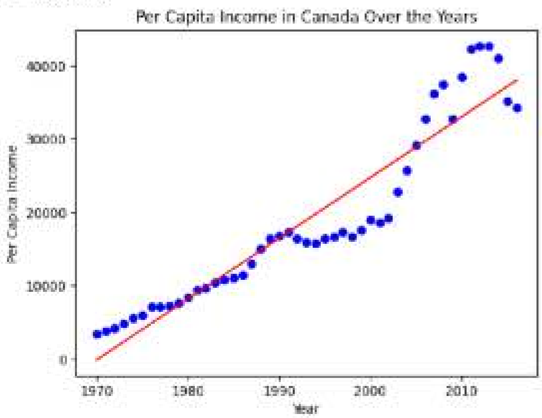
Code for linear regression:

```
#canada dataset
import pandas as pd
import io
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression

df = pd.read_csv("canada.csv")
print(df.head())
missing_values = df.isnull().sum()
# Display columns with missing values
print(missing_values[missing_values > 0])

df.drop_duplicates(inplace = True)

plt.xlabel('year')
plt.ylabel('per capita income')
plt.scatter(df['year'], df['per capita income (US$)'], color='red', marker='+')
plt.show()

X = df[['year']] #independent variable (predictor)
y = df['per capita income (US$)'] #dependent variable (target)

reg = LinearRegression()#req 2 parameters
reg.fit(X,y)
predicted_income = reg.predict([[2025]])
print(predicted_income)

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()
```

Output:

```
   year  per capita income (US$)
0  1970              3399.299037
1  1971              3768.297935
2  1972              4251.175484
3  1973              4804.463248
4  1974              5576.514583
Series([], dtype: int64)
```



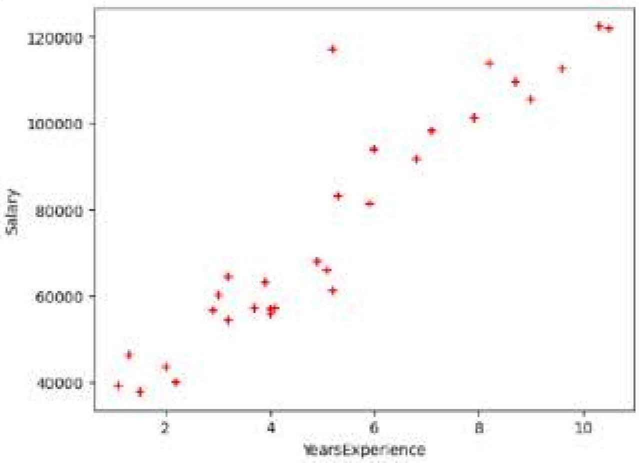


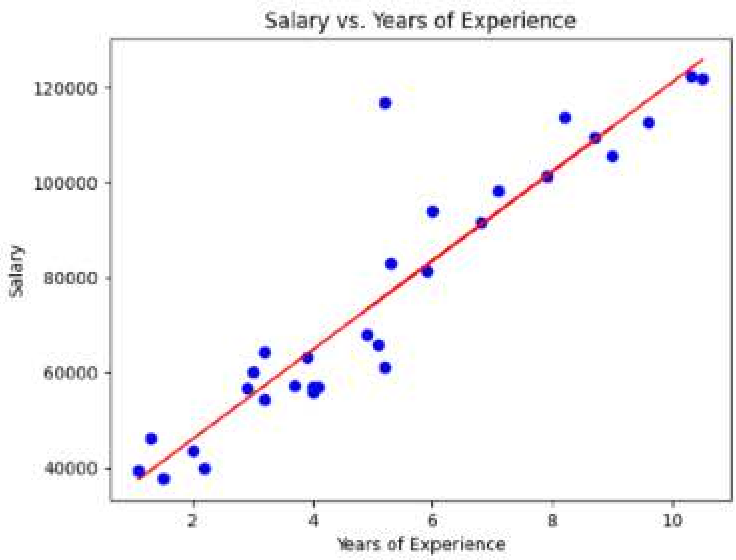

```
#salart dataset
import pandas as pd
import io
from sklearn import linear_model
import numpy as np


df = pd.read_csv("salary.csv")
print(df.head())

df.replace(' ',np.nan,inplace = True)
print(df.head())

missing_values = df.isnull().sum()
# Display columns with missing values
print(missing_values[missing_values > 0])

#handle missing values
from sklearn.impute import SimpleImputer
```

```
imputer2 = SimpleImputer(strategy="mean")

df_copy=df

# Step 2: Fit the imputer on the "Age" and "Salary"column
# Note: SimpleImputer expects a 2D array, so we reshape the column

imputer2.fit(df_copy[["YearsExperience"]])

# Step 3: Transform (fill) the missing values in the "Age" and "Salary"c column

df_copy["YearsExperience"] = imputer2.transform(df[["YearsExperience"]])

# Verify that there are no missing values left

print(df_copy["YearsExperience"].isnull().sum())

plt.xlabel('YearsExperience')
plt.ylabel('Salary')
plt.scatter(df_copy['YearsExperience'], df_copy['Salary'], color='red', marker='+')
plt.show()

X = df_copy[['YearsExperience']] #independent
y = df_copy['Salary'] #dependent

reg = linear_model.LinearRegression()
reg.fit(X,y)
predicted_salary = reg.predict([[12]])
print(predicted_salary)

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()
```

Output:

```
   YearsExperience  Salary
0              1.1   39343
1              1.3   46205
2              1.5   37731
3              2.0   43525
4              2.2   39891
   YearsExperience  Salary
0              1.1   39343
1              1.3   46205
2              1.5   37731
3              2.0   43525
4              2.2   39891
YearsExperience    2
dtype: int64
0
```





Code for multiple regression:

```
#hiring dataset
import pandas as pd
import io
from sklearn import linear_model
import numpy as np
from sklearn.preprocessing import OrdinalEncoder
from sklearn.impute import SimpleImputer
```

```
df = pd.read_csv("hiring.csv")
print(df.head())

df.replace(' ',np.nan,inplace = True)
print(df.head())

missing_values = df.isnull().sum()
# Display columns with missing values
print(missing_values[missing_values > 0])


df['experience'].fillna("unknown", inplace=True)
print(df.head())



#handle missing values
ordinal_encoder = OrdinalEncoder(categories=[["unknown","one",
"two","three","four","five","six","seven","eight","nine","ten","eleven"]])
# Fit and transform the data
df['experience_encoded'] = ordinal_encoder.fit_transform(df[['experience']])

print(df.head())

df.drop('experience',axis = 1,inplace = True)
print(df.head())

from sklearn.impute import SimpleImputer
imputer2 = SimpleImputer(strategy="mean")

df_copy=df

# Step 2: Fit the imputer on the "Age" and "Salary"column
# Note: SimpleImputer expects a 2D array, so we reshape the column

imputer2.fit(df_copy[["test_score(out of 10)"]])

# Step 3: Transform (fill) the missing values in the "Age" and "Salary"c column

df_copy["test_score(out of 10)"] = imputer2.transform(df[["test_score(out of 10)"]])

# Verify that there are no missing values left

print(df_copy["test_score(out of 10)"].isnull().sum())
```

```
X = df_copy[['test_score(out of 10)','interview_score(out of 10)','experience_encoded']]
y = df_copy[['salary($)']]

reg = linear_model.LinearRegression()
reg.fit(X,y)
predicted_salary = reg.predict([[2,9,6]])
print(predicted_salary)

predicted_salary = reg.predict([[12,10,10]])
print(predicted_salary)
```

Output:

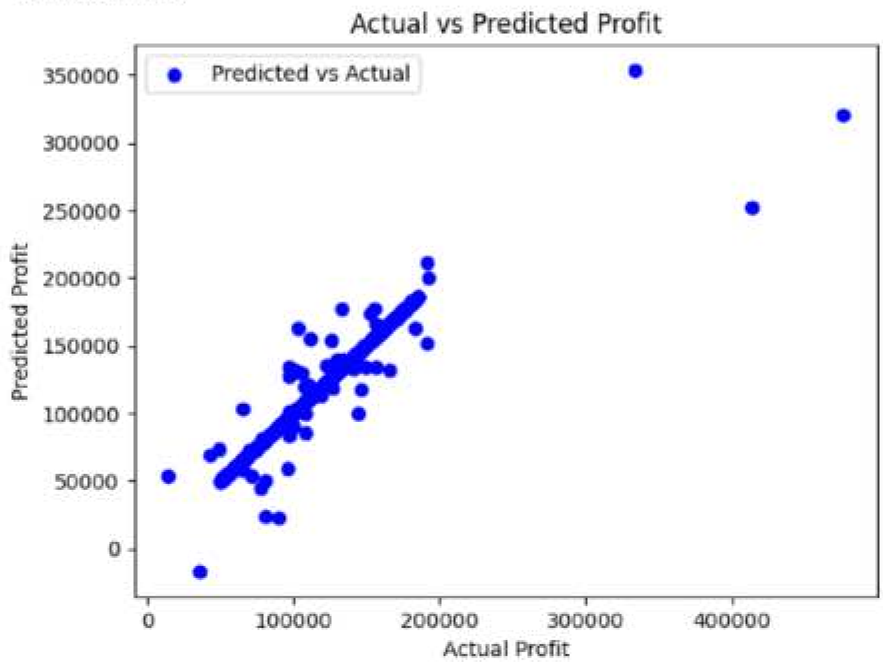
```
experience  test_score(out of 10)  interview_score(out of 10)  salary($)
0       NaN                  8.0                          9      50000
1       NaN                  8.0                          6      45000
2      five                  6.0                          7      60000
3       two                 10.0                         10      65000
4     seven                  9.0                          6      70000
experience              2
test_score(out of 10)   1
dtype: int64
  experience  test_score(out of 10)  interview_score(out of 10)  salary($)
0    unknown                  8.0                          9      50000
1    unknown                  8.0                          6      45000
2       five                  6.0                          7      60000
3        two                 10.0                         10      65000
4      seven                  9.0                          6      70000
  experience  test_score(out of 10)  interview_score(out of 10)  salary($)  \
0    unknown                  8.0                          9      50000
1    unknown                  8.0                          6      45000
2       five                  6.0                          7      60000
3        two                 10.0                         10      65000
4      seven                  9.0                          6      70000

   experience_encoded
0                 0.0
1                 0.0
2                 5.0
3                 2.0
4                 7.0
   test_score(out of 10)  interview_score(out of 10)  salary($)  \
0                    8.0                          9      50000
1                    8.0                          6      45000
2                    6.0                          7      60000
3                   10.0                         10      65000
4                    9.0                          6      70000

   experience_encoded
0                 0.0
1                 0.0
2                 5.0
3                 2.0
4                 7.0
```

```
0
[[57801.7884606]]
[[90438.68025262]]
```

```
#company dataset
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import LabelEncoder

df_companies = pd.read_csv('company.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()
```

Output:

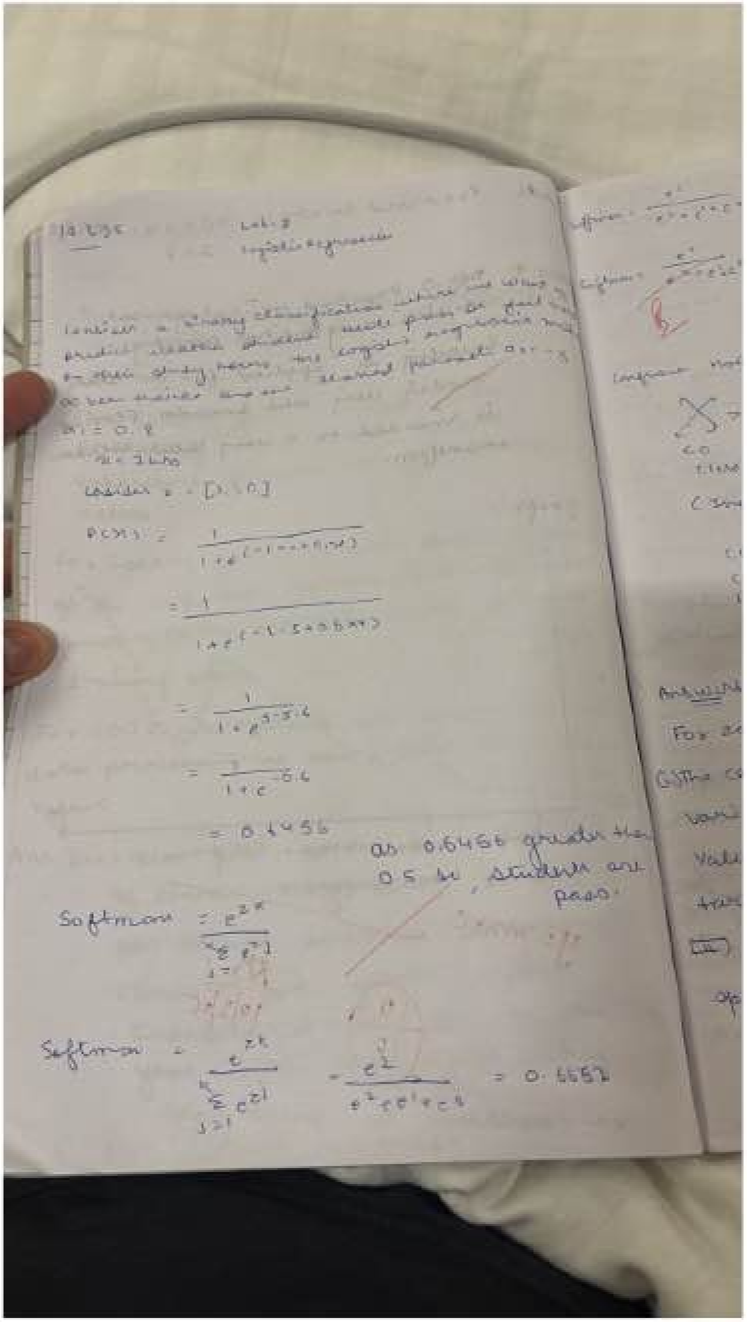
```
   R&D Spend  Administration  Marketing Spend       State     Profit
0  165349.20       136897.80        471784.10    New York  192261.83
1  162597.70       151377.59        443898.53  California  191792.06
2  153441.51       101145.55        407934.54     Florida  191050.39
3  144372.41       118671.85        383199.62    New York  182901.99
4  142107.34        91391.77        366168.42     Florida  166187.94
Predicted profit: 511209.20 USD
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X doe
  warnings.warn(
```



Program 4

Build Logistic Regression Model for a given dataset

Screenshots:



Softmax $= \dfrac{e^1}{e^2+e^1+e^0} = 0.2447$

Softmax $= \dfrac{e^0}{e^2+e^1+e^0} = 0.0900$

Confusion Matrix

Binary classification

+ve ↙ ↘ -ve

T Cat   F Dog.

> 0 (more) (correct classification)

< 0 (less) (Incorrect classification)

CC → CC

CD → IC

DC → IC

DD → CC

Answers

For zoo dataset

(i) The class_type col have categorical ~~numerical~~ variables and is converted into numerical value as Logistic regression require numerical targets

(ii) For prediction we are using number_of_animal species_in class.

(ii) Dataset does not contain any missing values, so no inconsistencies are observed

(iii) The confusion matrix showed 100% misclassification, leading to an accuracy of 0.0.
The model failed to predict any class, meaning at dataset lacks meaningful diversity.
so, number-of-animal-species-in-class also is not a good predictor for class type.

(iv) All class types were misclassified likely due to insufficient features, small dataset size and logistic regression limitation

For HR_comma_sep.csv Dataset

(i) Variables with direct and clear impact on employee Retention from exploratory Data Analysis and logistic regression, the following variable had a significant impact on employee retention:

Satisfaction level

Employee with low satisfaction levels are more likely to leave the company. strong correlation with employee retention

$\boxed{\text{Salary}}$

Employee with low salaries have a higher turnover rate. those with higher salary are more likely to stay $\boxed{\text{time spent at the company}}$

Employee who have spent more years at the company have higher chance of leaving. This suggest possible burnout or lack of growth opp. $\boxed{\text{No of Projects}}$

Employee with too few or too many projects are more likely to leave. A balanced workload contributes to retention. $\boxed{\text{Avg monthly hrs}}$

employee working extremely high or low hrs tend to leave. overwork or underutilization can lead to dissatisfaction. $\boxed{\text{Promoted in last 5 years}}$

employee who have not been promoted in the last 5 yrs are more likely to leave. Lack of career growth opportunities aff retention

(ii) Accuracy: 75.8%

The model correctly predicts employee attrition about 76% of the time. This is acceptable but not best

75.8% is decent but could be improved.

Code for logistic regression for binary classification:

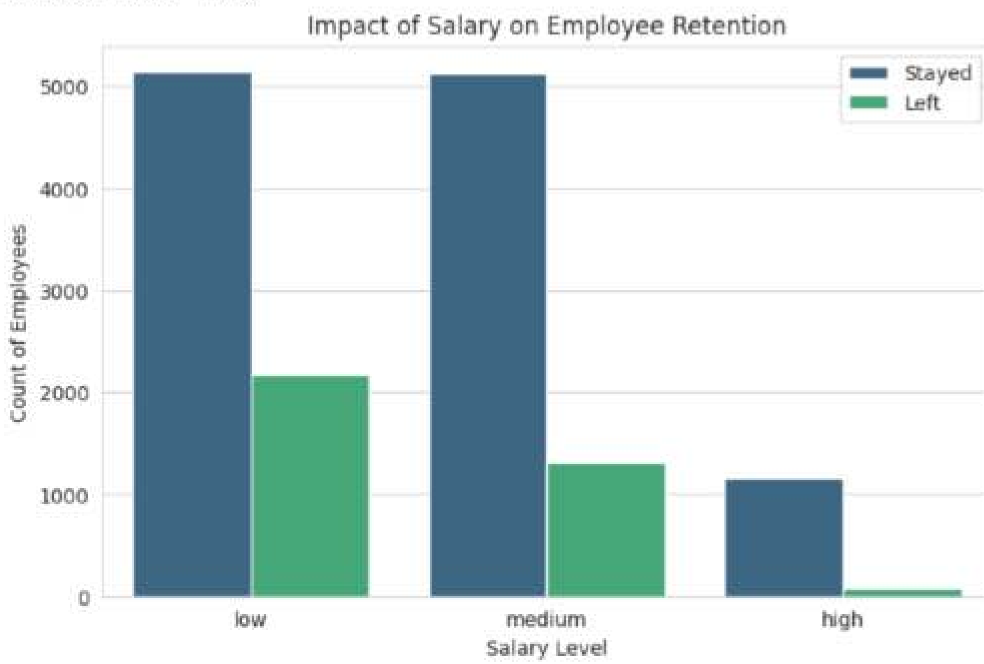
```
#HR dataset
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

df = pd.read_csv("HR.csv")
print(df.head())

missing_values = df.isnull().sum()
# Display columns with missing values
print(missing_values[missing_values > 0])

# Set seaborn style
sns.set_style("whitegrid")

# Plot bar chart for salary vs retention
plt.figure(figsize=(8, 5))
sns.countplot(x="salary", hue="left", data=df, palette="viridis")
plt.xlabel("Salary Level")
plt.ylabel("Count of Employees")
plt.title("Impact of Salary on Employee Retention")
plt.legend(["Stayed", "Left"])
plt.show()

# Plot bar chart for department vs retention
plt.figure(figsize=(12, 5))
sns.countplot(y="Department",        hue="left",        data=df,        palette="coolwarm",
```

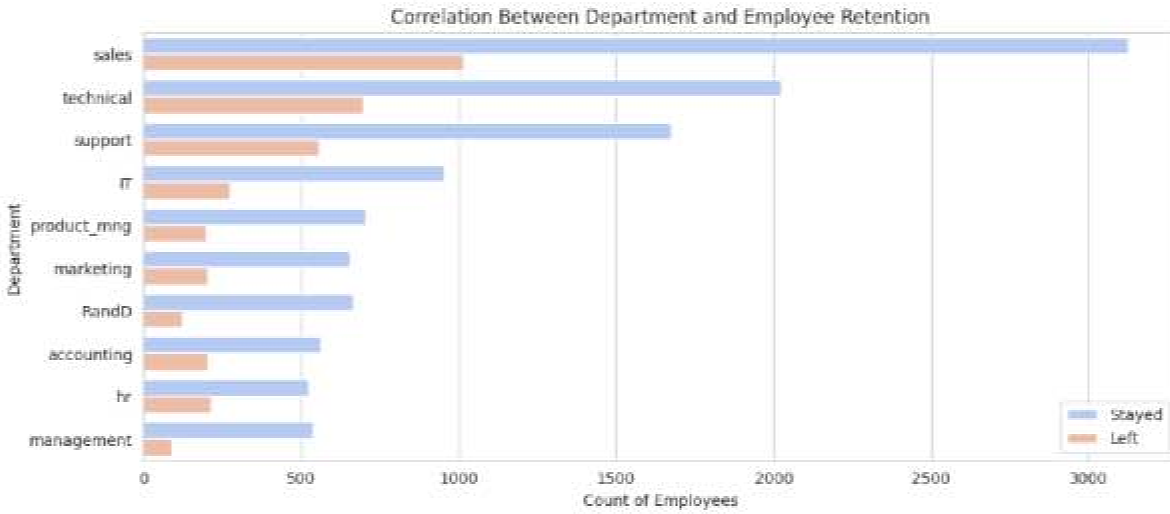
```
order=df["Department"].value_counts().index)
plt.xlabel("Count of Employees")
plt.ylabel("Department")
plt.title("Correlation Between Department and Employee Retention")
plt.legend(["Stayed", "Left"])
plt.show()

# Encode categorical variables
label_encoders = {}
for col in ["salary", "Department"]:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le

# Select relevant features
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 train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the numerical features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train logistic regression model
log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)

# Predict on test set
```

```
y_pred = log_reg.predict(X_test)


# Measure accuracy
accuracy = accuracy_score(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)


# Print results
print(f"Model Accuracy: {accuracy:.4f}")
print("Classification Report:")
print(classification_rep)
```

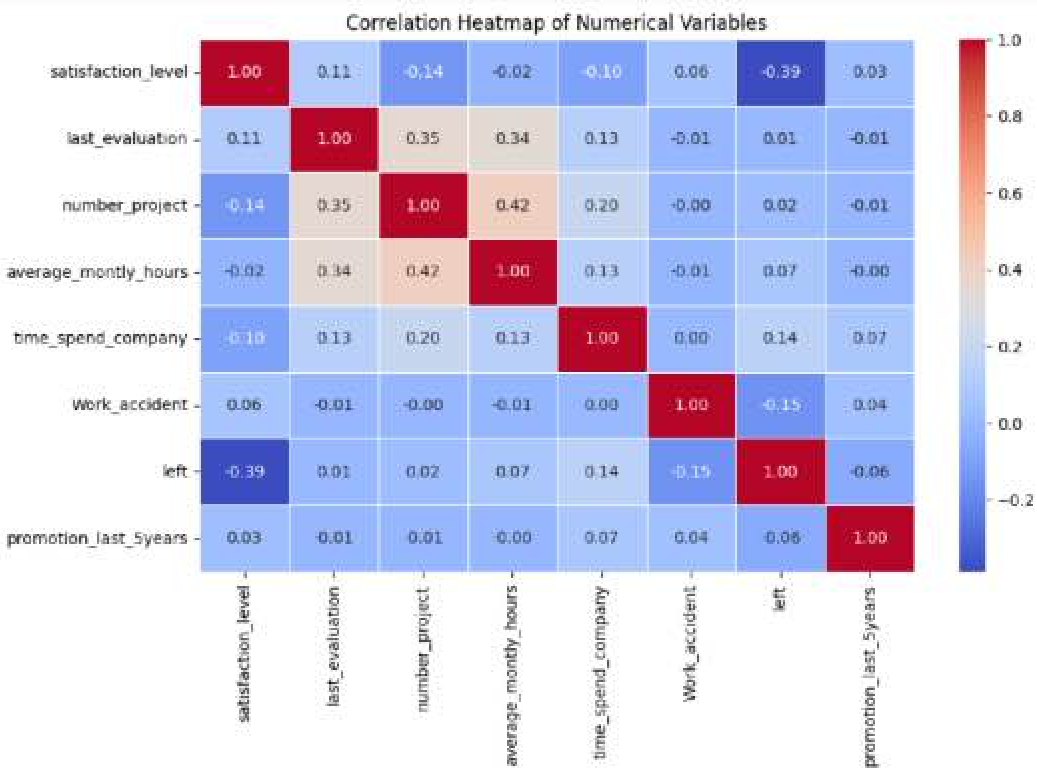
Output:

```
     salary
0       low
1    medium
2    medium
3       low
4       low
Series([], dtype: int64)
```






```
Model Accuracy: 0.7577
Classification Report:
              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
```



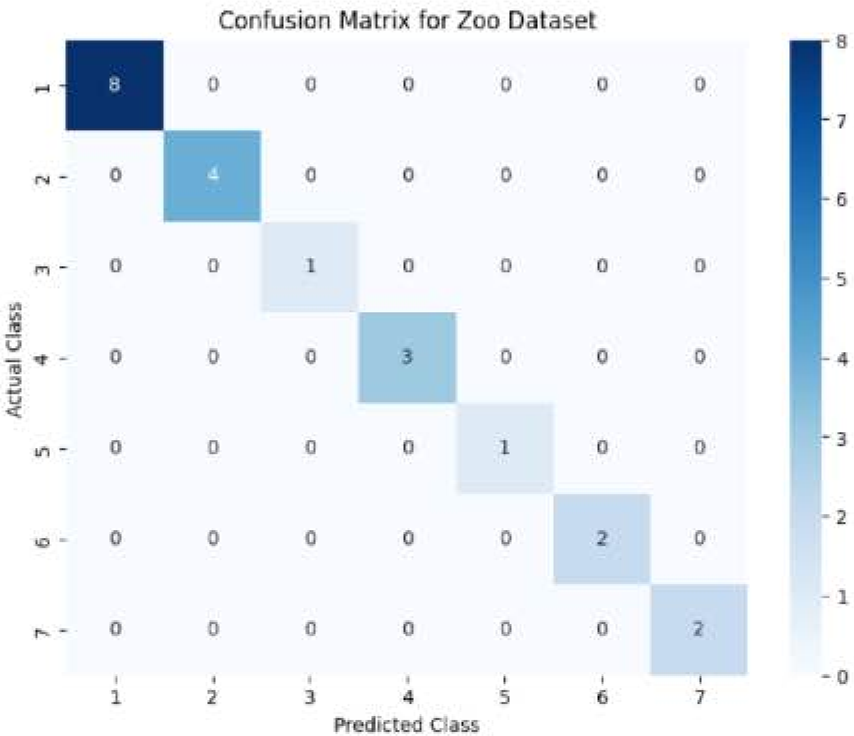
Code for logistic regression for multi classification:

```python
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.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Load the dataset
df_zoo = pd.read_csv("zoo.csv")

# Drop 'animal_name' as it's not useful for classification
df_zoo = df_zoo.drop(columns=["animal_name"])

# Separate features and target variable
X = df_zoo.drop(columns=["class_type"])  # Features
y = df_zoo["class_type"]  # Target (class type)

# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Standardize numerical features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train multinomial logistic regression model
model = LogisticRegression(multi_class="multinomial", max_iter=1000)
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)
```

```python
# Measure accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy:.4f}")

# Print classification report
print("Classification Report:\n", classification_report(y_test, y_pred))

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

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=np.unique(y), yticklabels=np.unique(y))
plt.xlabel("Predicted Class")
plt.ylabel("Actual Class")
plt.title("Confusion Matrix for Zoo Dataset")
plt.show()
```

Output:

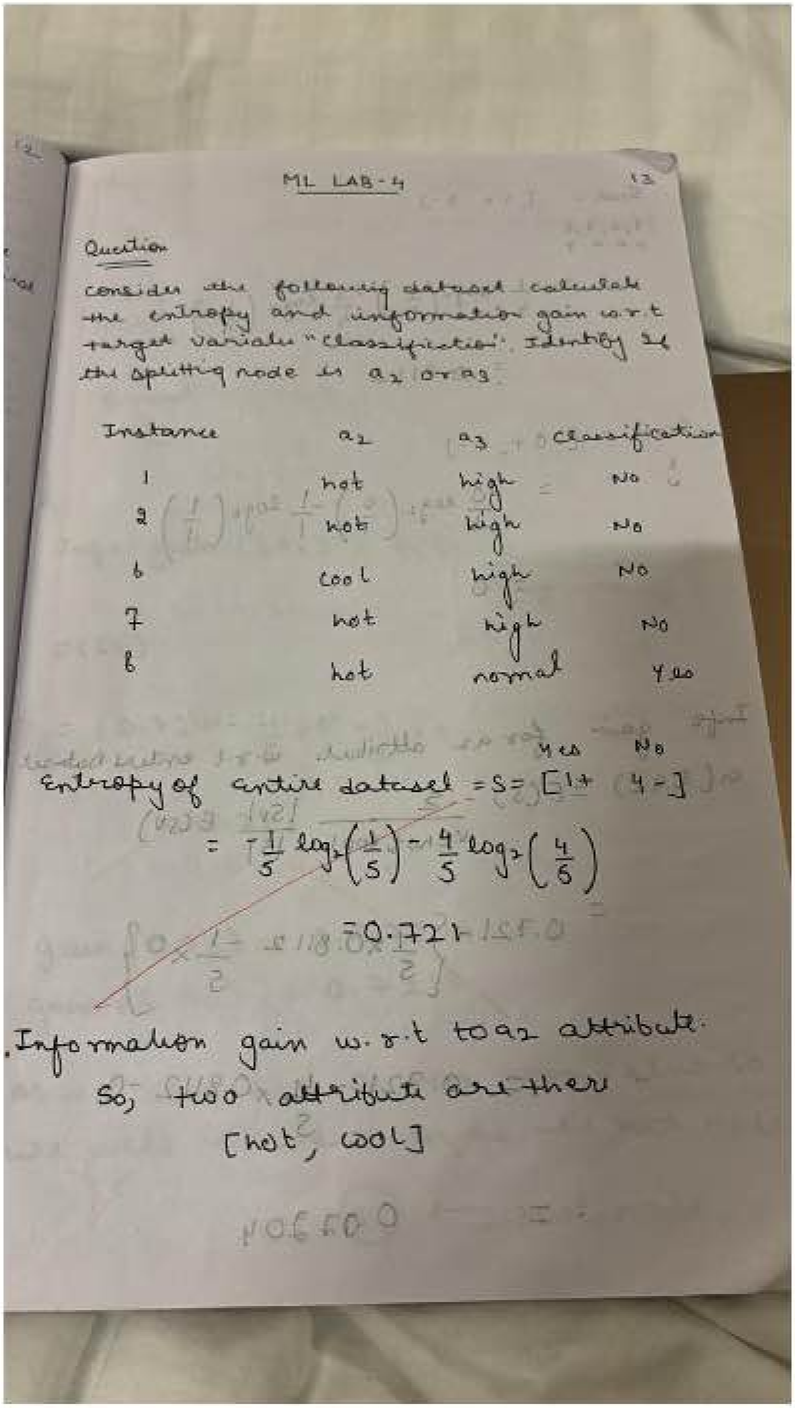
```
Model Accuracy: 1.0000
Classification Report:
               precision    recall  f1-score   support

           1       1.00      1.00      1.00         8
           2       1.00      1.00      1.00         4
           3       1.00      1.00      1.00         1
           4       1.00      1.00      1.00         3
           5       1.00      1.00      1.00         1
           6       1.00      1.00      1.00         2
           7       1.00      1.00      1.00         2

    accuracy                           1.00        21
   macro avg       1.00      1.00      1.00        21
weighted avg       1.00      1.00      1.00        21
```


Confusion Matrix for Zoo Dataset
Actual Class
Predicted Class
1
2
3
4
5
6
7
8
0

Program 5

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

Screenshots:



ML LAB-4 13

Question

consider the following dataset calculate the entropy and information gain w.r.t target variable "classification". Identify if the splitting node is a_2 or a_3

Instance	a_2	a_3	Classification
1	hot	high	No
2	hot	high	No
6	cool	high	No
7	hot	high	No
8	hot	normal	Yes

Entropy of entire dataset $= S = [1+ \; 4-]$ (Yes, No)

$$= -\frac{1}{5}\log_2\left(\frac{1}{5}\right) - \frac{4}{5}\log_2\left(\frac{4}{5}\right)$$

$$= 0.721$$

Information gain w.r.t to a_2 attribute.

So, two attribute are there

[hot, cool]

$S_{not} = [1+ \; 3-]$

1,2,7,8
N N N Y

$$= -\frac{1}{4}\log_2\left(\frac{1}{4}\right) - \frac{3}{4}\log_2\left(\frac{3}{4}\right)$$

$$= 0.8112$$

$S_{cool} = [0+, 1-]$

$$= -\frac{0}{1}\log_2\left(\frac{0}{1}\right) - \frac{1}{1}\log_2\left(\frac{1}{1}\right)$$

$$= 0$$

Info gain for a2 attribute w.r.t entire Data

$$G(S, a_2) = E(S) - \sum_{v \in not, cool} \frac{|S_v|}{|S|} E(S_v)$$

$$= 0.721 - \left\{\frac{4}{5} \times 0.8112 - \frac{1}{5} \times 0\right\}$$

$$= 0.721 - \frac{4}{5} \times 0.8112 - 0$$

$$= 0.07204$$

Info gain w.r.t to a_3 attribute 15

So, two attribute are there {high, normal}

$S_{high} = [0+, 4-]$

1, 2, 4, 7

$= 0$

$S_{normal} = [1+, 0-]$

8

yes

$= 0$

$$\text{Info gain}(S, a_3) = E(S) - \sum_{v \in \{high,\, normal\}} \frac{|S_v|}{|S|} E(S_v)$$

$$= 0.721 - \frac{4}{5} \times 0 - \frac{1}{5} \times 0$$

$$= 0.721$$

$gain(S, a_2) = 0.07204$

$gain(S, a_3) = 0.7219$

As we can see a_3 has max value so we will consider a_3 as the root node.

a_3 ← root node.

Accuracy score = 100%

$= \frac{\text{no. of correct prediction}}{\text{Total prediction}}$

since, all prediction are correct so, 1.0.

Diagonal value (10, 9, 11) show the correctly classified instance

non Diagonal → misclassification

$$\begin{array}{cc} & \text{Predicted} \\ \text{Actual} & \begin{bmatrix} TP & FN \\ FP & TN \end{bmatrix} \end{array}$$

intersection of that

TP → Actual True Predicted True

TN → Actual False Predicted False

FN → Actual True Predicted False

FP → Actual False Predicted True

The pattern it has learn sepal length, sepal width, petal length, petal width

Based on this it will predict which flower it is.

Questions

1. For "Iris.csv" dataset

Accuracy score =1

confusion matrix

Actual	Predicted: Setosa	Predicted: Versicolor	Predicted: virginica
iris setosa	10	0	0
iris versicolor	0	9	0
iris virginica	0	0	11

There is no misclassification as all the diagonal values are True positive. so no misclassification.

2. For "Petroleum.csv" dataset.

The regression tree works by recursively splitting dataset into smaller region based on the feature value.
The mean Squared error value is used

The Driving license is most imp feature for predicting petrol consumption

Decision tree is used for categorical values where as regression tree, we use for numerical continuous value.

Splitting is based on min MSE.

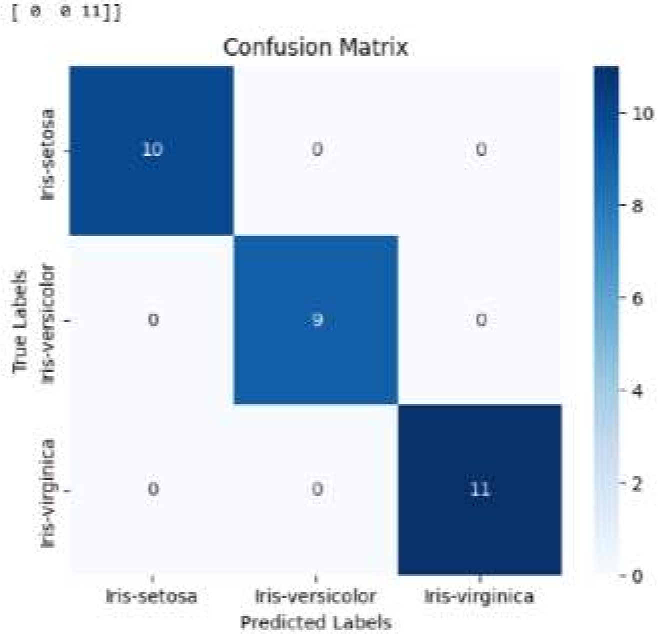
Code:

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

df = pd.read_csv("iris.csv")
print(df.head())

missing_values = df.isnull().sum()
# Display columns with missing values
print(missing_values[missing_values > 0])

X = df.iloc[:, :-1]  # All columns except the last one (features)
y = df.iloc[:, -1]   # Last column (target)

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

# Create and train the Decision Tree model
model = DecisionTreeClassifier()
model.fit(X_train, y_train)

# Predict on test data
y_pred = model.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy Score: {accuracy:.4f}')
```

```
# Display confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:')
print(conf_matrix)

plt.figure(figsize=(6,5))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=df.iloc[:, -1].unique(),
yticklabels=df.iloc[:, -1].unique())
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```

Output:

```
   sepal_length  sepal_width  petal_length  petal_width      species
0           5.1          3.5           1.4          0.2  Iris-setosa
1           4.9          3.0           1.4          0.2  Iris-setosa
2           4.7          3.2           1.3          0.2  Iris-setosa
3           4.6          3.1           1.5          0.2  Iris-setosa
4           5.0          3.6           1.4          0.2  Iris-setosa
Series([], dtype: int64)
Accuracy Score: 1.0000
Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

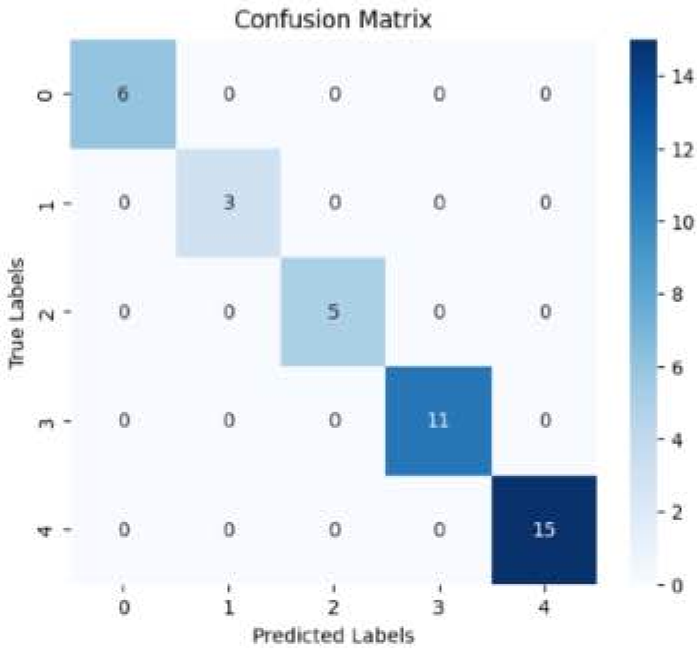


```python
#drug dataset
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.preprocessing import LabelEncoder

# Load the dataset
df = pd.read_csv("drug.csv")

# Check the first few rows
print(df.head())

missing_values = df.isnull().sum()
# Display columns with missing values
print(missing_values[missing_values > 0])



# Encode categorical columns if any
label_encoders = {}
for column in df.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    df[column] = le.fit_transform(df[column])
    label_encoders[column] = le  # Store label encoders for later decoding if needed

# Separate features and target variable
X = df.iloc[:, :-1]  # All columns except the last one (features)
y = df.iloc[:, -1]   # Last column (target)

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

```
# Create and train the Decision Tree model
model = DecisionTreeClassifier()
model.fit(X_train, y_train)

# Predict on test data
y_pred = model.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy Score: {accuracy:.4f}')

# Compute confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Plot the confusion matrix
plt.figure(figsize=(6,5))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```

Output:

```
   Age Sex      BP Cholesterol  Na_to_K   Drug
0   23   F    HIGH        HIGH   25.355  drugY
1   47   M     LOW        HIGH   13.093  drugC
2   47   M     LOW        HIGH   10.114  drugC
3   28   F  NORMAL        HIGH    7.798  drugX
4   61   F     LOW        HIGH   18.043  drugY
Series([], dtype: int64)
Accuracy Score: 1.0000
```




```
#petrol dataset
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error

# Load the dataset
df = pd.read_csv('petrol.csv')

# Display first few rows
```

```
print(df.head())

# Separate features and target variable
X = df.iloc[:, :-1]  # All columns except the last one (features)
y = df.iloc[:, -1]   # Last column (target - Petrol Consumption)

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

# Create and train the Regression Tree model
model = DecisionTreeRegressor()
model.fit(X_train, y_train)

# Predict on test data
y_pred = model.predict(X_test)

# Compute error metrics
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)

# Display results
print(f'Mean Absolute Error (MAE): {mae:.4f}')
print(f'Mean Squared Error (MSE): {mse:.4f}')
print(f'Root Mean Squared Error (RMSE): {rmse:.4f}')
```

Output:

```
Petrol_tax  Average_income  Paved_Highways  Population_Driver_licence(%) \
0     9.0          3571          1976                  0.525
1     9.0          4092          1250                  0.572
2     9.0          3865          1586                  0.580
3     7.5          4870          2351                  0.529
4     8.0          4399           431                  0.544

   Petrol_Consumption
```

0	541
1	524
2	561
3	414
4	410

Mean Absolute Error (MAE): 84.5000

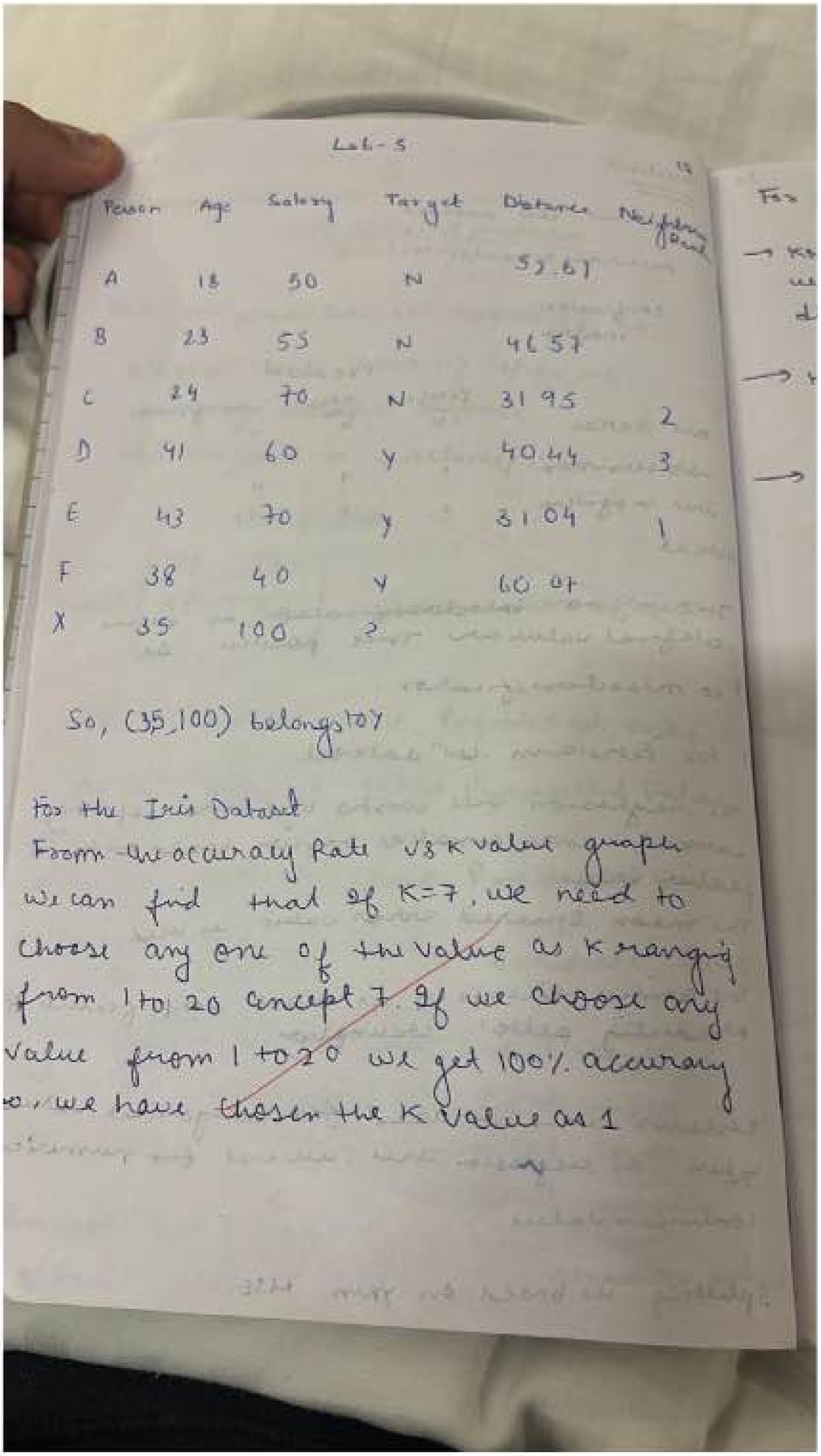
Mean Squared Error (MSE): 15672.9000

Root Mean Squared Error (RMSE): 125.1915

Program 6

Build KNN Classification model for a given dataset.

Screenshots:



Lab - 5

Person	Age	Salary	Target	Distance	Neighbour (Rank)
A	18	50	N	53.67	
B	23	55	N	46.57	
C	24	70	N	31.95	2
D	41	60	Y	40.44	3
E	43	70	Y	31.04	1
F	38	40	Y	60.07	
X	35	100	?		

So, (35,100) belongs to Y

For the Iris Dataset
From the accuracy Rate vs K value graph we can find that if K=7, we need to choose any one of the value as K ranging from 1 to 20 except 7. If we choose any value from 1 to 20 we get 100% accuracy so, we have chosen the K value as 1

19

For diabetes dataset

→ KNN is a distance based algo so features with larger values will dominate the distance metric.

→ Here, we use standardization to ensure features scaling

→ Now, all features contribute equally to distance calculation.

Code:

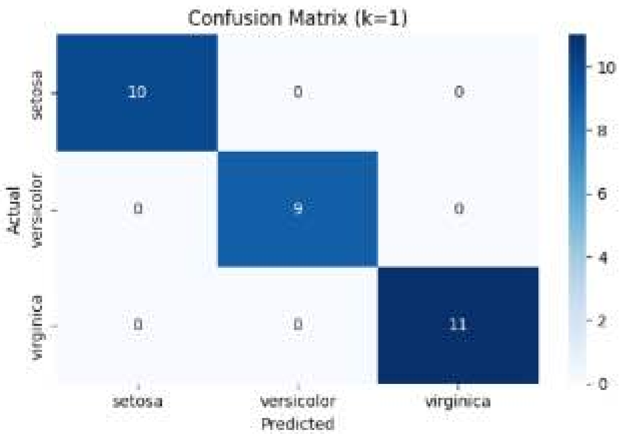
```
#iris dataset
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot as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

# Load dataset
df = pd.read_csv("iris.csv")

# Features and target
X = df.drop('species', axis=1)
y = df['species']

# Encode target labels
le = LabelEncoder()
y_encoded = le.fit_transform(y)

# Train-test split (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.2, random_state=42)

# Find best k (1 to 20)
scores = []
k_range = range(1, 21)
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    scores.append(knn.score(X_test, y_test))

best_k = k_range[scores.index(max(scores))]
```

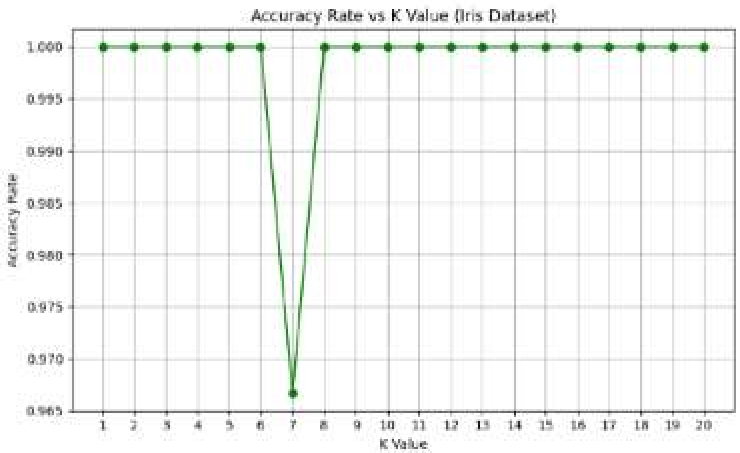
```
# Train with best k
knn_final = KNeighborsClassifier(n_neighbors=best_k)
knn_final.fit(X_train, y_train)

# Predictions
y_pred = knn_final.predict(X_test)

# Accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Best k: {best_k}")
print(f"Accuracy: {accuracy:.2f}")

# Classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=le.classes_))

# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Plot confusion matrix
plt.figure(figsize=(6,4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
        xticklabels=le.classes_, yticklabels=le.classes_)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title(f'Confusion Matrix (k={best_k})')
plt.tight_layout()
plt.show()

# Plot Accuracy vs K
plt.figure(figsize=(8, 5))
plt.plot(k_range, scores, marker='o', linestyle='-', color='green')
plt.title('Accuracy Rate vs K Value (Iris Dataset)')
```

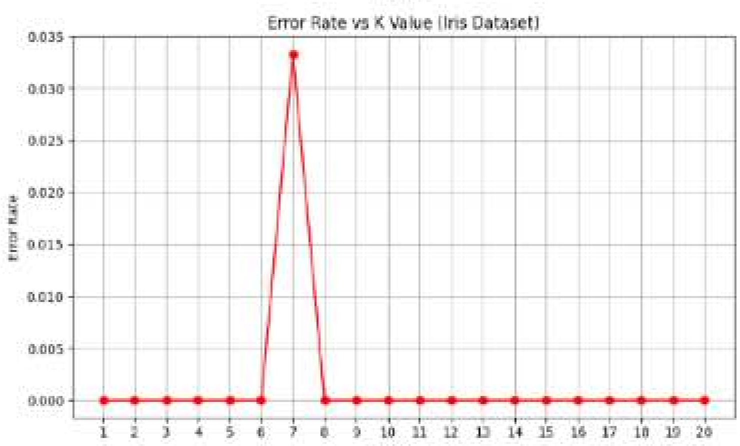
```python
plt.xlabel('K Value')
plt.ylabel('Accuracy Rate')
plt.xticks(k_range)
plt.grid(True)
plt.tight_layout()
plt.show()

# Calculate error rates
errors = [1 - acc for acc in scores]

# Plot Error Rate vs K
plt.figure(figsize=(8, 5))
plt.plot(k_range, errors, marker='o', linestyle='-', color='red')
plt.title('Error Rate vs K Value (Iris Dataset)')
plt.xlabel('K Value')
plt.ylabel('Error Rate')
plt.xticks(k_range)
plt.grid(True)
plt.tight_layout()
plt.show()
```

Output:

```
Best k: 1
Accuracy: 1.00

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
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

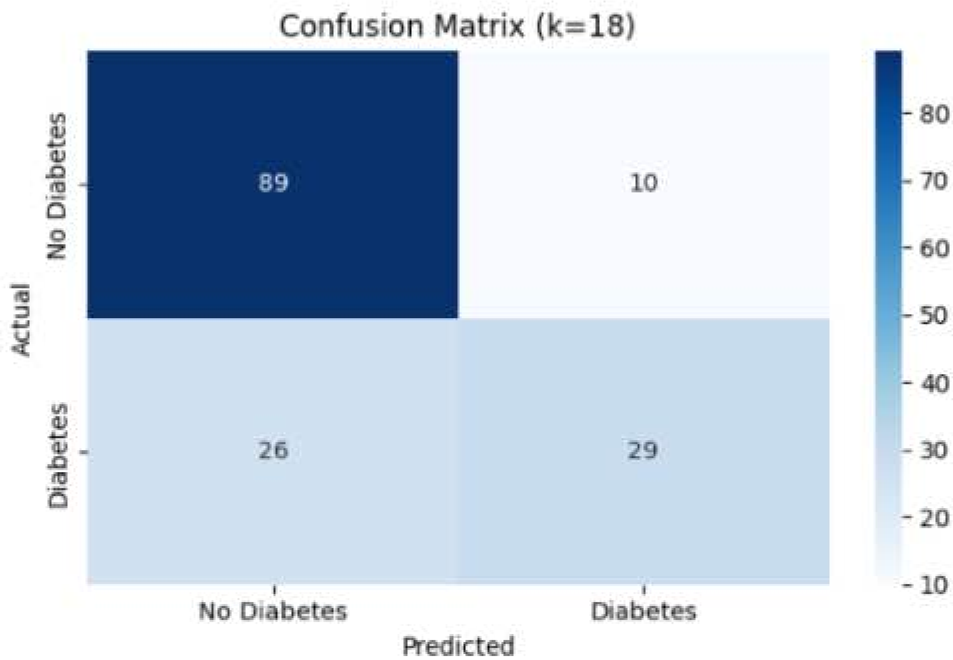
```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
df = pd.read_csv("diabetes.csv")

# Separate features and target
X = df.drop("Outcome", axis=1)
y = df["Outcome"]

# Feature scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Train-test split (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Find the best k from range 1 to 20
k_scores = []
k_range = range(1, 21)
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    k_scores.append(knn.score(X_test, y_test))

# Best k value
best_k = k_range[k_scores.index(max(k_scores))]
print(f"Best k value: {best_k}")

# Train final model
knn_final = KNeighborsClassifier(n_neighbors=best_k)
knn_final.fit(X_train, y_train)
```

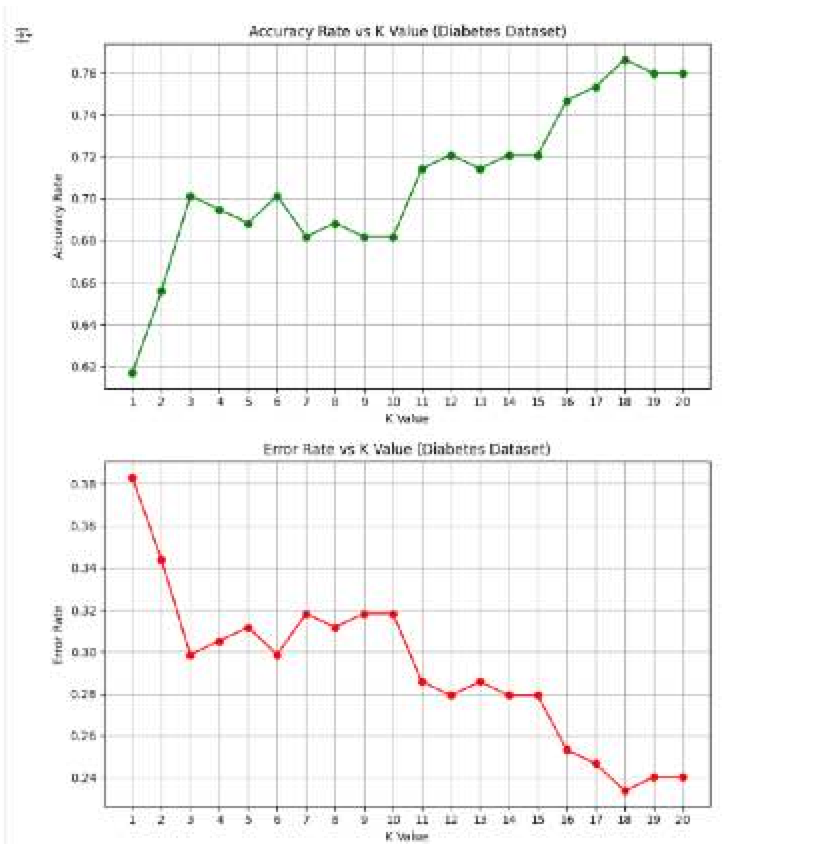
```
# Predictions
y_pred = knn_final.predict(X_test)

# Accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# Classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Plotting confusion matrix
plt.figure(figsize=(6,4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
        xticklabels=["No Diabetes", "Diabetes"],
        yticklabels=["No Diabetes", "Diabetes"])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title(f'Confusion Matrix (k={best_k})')
plt.tight_layout()
plt.show()

# Plot Accuracy Rate vs K
plt.figure(figsize=(8, 5))
plt.plot(k_range, k_scores, marker='o', linestyle='-', color='green')
plt.title('Accuracy Rate vs K Value (Diabetes Dataset)')
plt.xlabel('K Value')
plt.ylabel('Accuracy Rate')
plt.xticks(k_range)
plt.grid(True)
```

```python
plt.tight_layout()
plt.show()



# Calculate error rate
error_rates = [1 - acc for acc in k_scores]

# Plot Error Rate vs K
plt.figure(figsize=(8, 5))
plt.plot(k_range, error_rates, marker='o', linestyle='-', color='red')
plt.title('Error Rate vs K Value (Diabetes Dataset)')
plt.xlabel('K Value')
plt.ylabel('Error Rate')
plt.xticks(k_range)
plt.grid(True)
plt.tight_layout()
plt.show()
```

Output:

```
Best k value: 18
Accuracy: 0.77

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
import pandas as pd
import numpy as np
```

```python
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
import matplotlib.pyplot as plt
import seaborn as sns

# Load dataset
df = pd.read_csv("heart.csv")

# Features and target
X = df.drop("target", axis=1)
y = df["target"]

# Feature scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Train-test split (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Find best k value (1 to 20)
k_range = range(1, 21)
k_scores = []

for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    k_scores.append(knn.score(X_test, y_test))

best_k = k_range[k_scores.index(max(k_scores))]
print(f"Best k value: {best_k}")

# Train final model with best k
```

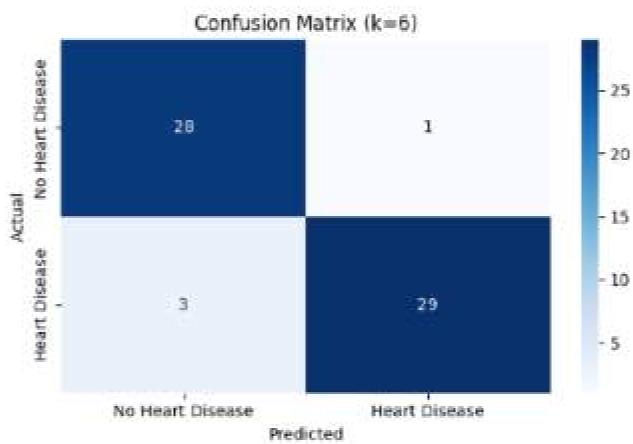
```
knn_final = KNeighborsClassifier(n_neighbors=best_k)
knn_final.fit(X_train, y_train)

# Predictions
y_pred = knn_final.predict(X_test)

# Accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# Classification report
report = classification_report(y_test, y_pred, output_dict=True)
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Plot confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6,4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
        xticklabels=["No Heart Disease", "Heart Disease"],
        yticklabels=["No Heart Disease", "Heart Disease"])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title(f"Confusion Matrix (k={best_k})")
plt.tight_layout()
plt.show()

# Plot classification report as heatmap
plt.figure(figsize=(6,4))
sns.heatmap(pd.DataFrame(report).iloc[:-1, :].T, annot=True, cmap="YlGnBu")
plt.title("Classification Report")
plt.tight_layout()
plt.show()
```

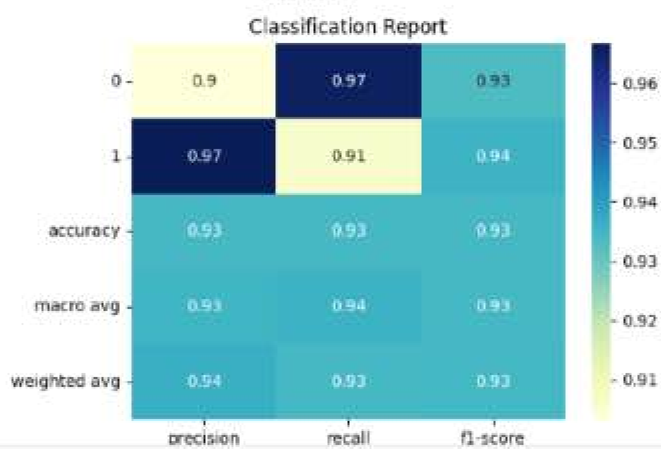
Output:

Accuracy: 0.93

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

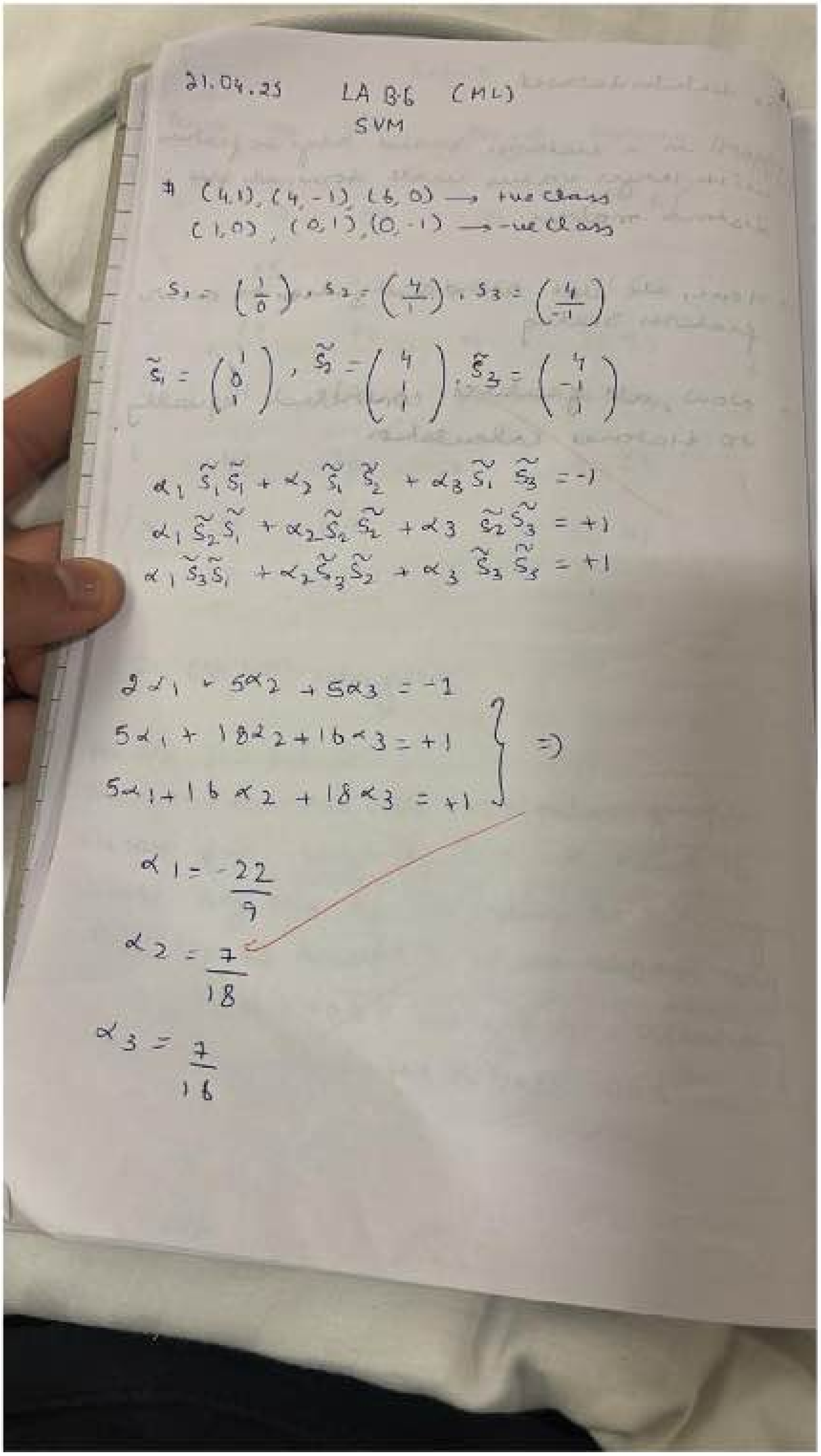




Program 7

Build Support vector machine model for a given dataset

Screenshots:


21.04.25 LAB 6 (ML)
SVM

(4,1), (4,-1), (6,0) → +ve class
(1,0), (0,1), (0,-1) → -ve class

$s_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, s_2 = \begin{pmatrix} 4 \\ 1 \end{pmatrix}, s_3 = \begin{pmatrix} 4 \\ -1 \end{pmatrix}$

$\tilde{s}_1 = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}, \tilde{s}_2 = \begin{pmatrix} 4 \\ 1 \\ 1 \end{pmatrix}, \tilde{s}_3 = \begin{pmatrix} 4 \\ -1 \\ 1 \end{pmatrix}$

$\alpha_1 \tilde{s}_1 \tilde{s}_1 + \alpha_2 \tilde{s}_1 \tilde{s}_2 + \alpha_3 \tilde{s}_1 \tilde{s}_3 = -1$

$\alpha_1 \tilde{s}_2 \tilde{s}_1 + \alpha_2 \tilde{s}_2 \tilde{s}_2 + \alpha_3 \tilde{s}_2 \tilde{s}_3 = +1$

$\alpha_1 \tilde{s}_3 \tilde{s}_1 + \alpha_2 \tilde{s}_3 \tilde{s}_2 + \alpha_3 \tilde{s}_3 \tilde{s}_3 = +1$

$\left.\begin{aligned} 2\alpha_1 + 5\alpha_2 + 5\alpha_3 &= -1 \\ 5\alpha_1 + 18\alpha_2 + 16\alpha_3 &= +1 \\ 5\alpha_1 + 16\alpha_2 + 18\alpha_3 &= +1 \end{aligned}\right\} \Rightarrow$

$\alpha_1 = \frac{-22}{9}$

$\alpha_2 = \frac{7}{18}$

$\alpha_3 = \frac{7}{18}$

$$w = \sum_{i=1}^{3} \alpha_i \tilde{s}_i$$

$$= \alpha_1 \tilde{s}_1 + \alpha_2 \tilde{s}_2 + \alpha_3 \tilde{s}_3$$

$$= -\frac{22}{9}\begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} + \frac{7}{18}\begin{pmatrix} 4 \\ 1 \\ 1 \end{pmatrix} + \frac{7}{18}\begin{pmatrix} 4 \\ -1 \\ 1 \end{pmatrix}$$

$$= \begin{pmatrix} -22/9 \\ 0 \\ -22/9 \end{pmatrix} + \begin{pmatrix} 28/98 \\ 7/18 \\ 7/18 \end{pmatrix} + \begin{pmatrix} 28/18 \\ -7/18 \\ 7/18 \end{pmatrix}$$

$$= \begin{pmatrix} 2/3 \\ 0 \\ -5/3 \end{pmatrix}$$

$$w = \begin{pmatrix} 2/3 \\ 0 \end{pmatrix}, \; b = \left(-5/3 \right)$$

now,

$$w^T x + b = 0 \Rightarrow 2/3\, x_1 + 0 x_2 + (-5/3) = 0$$

$$\Rightarrow \frac{2}{3} x_1 , -\frac{5}{3} = 0$$

$$\Rightarrow x_1 = 2.5 \text{ or } x = 5/2.$$

21/04/[illegible]

1 → For "iris.csv" Dataset

→ Linear Kernel Accuracy = 97%
RBF Kernel Accuracy = 100%

→ Clearly RBF Kernel has given a better perfor-mance.

→ RBF can handle more complex boundaries between classes, since it is non-linear which would have contributed to the increased Accuracy

2. → For "letter-recognition.csv" Dataset

The letters that are most frequently confused are 'p' with 'f', 'k' with 'r'

→ The AUC score is 1, reflecting accuracy and excellent separability

→ It performs well on letter dataset considering

1. This Dataset is more complex.
2. Iris Dataset is simpler with less classes / features.

This demonstrates SVM's strength in handling high dimensional data.

Code:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
df1=pd.read_csv("/content/iris.csv")
df2=pd.read_csv("/content/letter.csv")
print("Iris\n",df1.head())
print("Letter recognition\n",df2.head())
X_iris = df1.drop('species', axis=1)
y_iris = df1['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)

# Linear Kernel SVM
svm_linear = SVC(kernel='linear', random_state=42)
svm_linear.fit(X_train_iris, y_train_iris)

# RBF Kernel SVM
svm_rbf = SVC(kernel='rbf', random_state=42)
svm_rbf.fit(X_train_iris, y_train_iris)
y_pred_linear = svm_linear.predict(X_test_iris)
y_pred_rbf = svm_rbf.predict(X_test_iris)

# Accuracy and Confusion Matrix for Linear Kernel
accuracy_linear = accuracy_score(y_test_iris, y_pred_linear)
conf_matrix_linear = confusion_matrix(y_test_iris, y_pred_linear)

# Accuracy and Confusion Matrix for RBF Kernel
```

```
accuracy_rbf = accuracy_score(y_test_iris, y_pred_rbf)
conf_matrix_rbf = confusion_matrix(y_test_iris, y_pred_rbf)

# Display Results
print(f"Linear Kernel Accuracy: {accuracy_linear}")
print(f"RBF Kernel Accuracy: {accuracy_rbf}")
# Confusion Matrices
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

sns.heatmap(conf_matrix_linear, annot=True, fmt='d', cmap='Blues', ax=ax1)
ax1.set_title("Linear Kernel Confusion Matrix")
ax1.set_xlabel('Predicted')
ax1.set_ylabel('Actual')

sns.heatmap(conf_matrix_rbf, annot=True, fmt='d', cmap='Blues', ax=ax2)
ax2.set_title("RBF Kernel Confusion Matrix")
ax2.set_xlabel('Predicted')
ax2.set_ylabel('Actual')

plt.show()
X_letter = df2.drop('letter', axis=1)
y_letter = df2['letter']

y_letter = y_letter.astype('category').cat.codes

X_train_letter, X_test_letter, y_train_letter, y_test_letter = train_test_split(X_letter,
y_letter, test_size=0.2, random_state=42)

# Linear Kernel SVM for Letter Recognition
svm_linear_letter = SVC(kernel='linear', random_state=42, probability=True)
svm_linear_letter.fit(X_train_letter, y_train_letter)
y_pred_linear_letter = svm_linear_letter.predict(X_test_letter)
y_pred_rbf_letter = svm_rbf_letter.predict(X_test_letter)
```

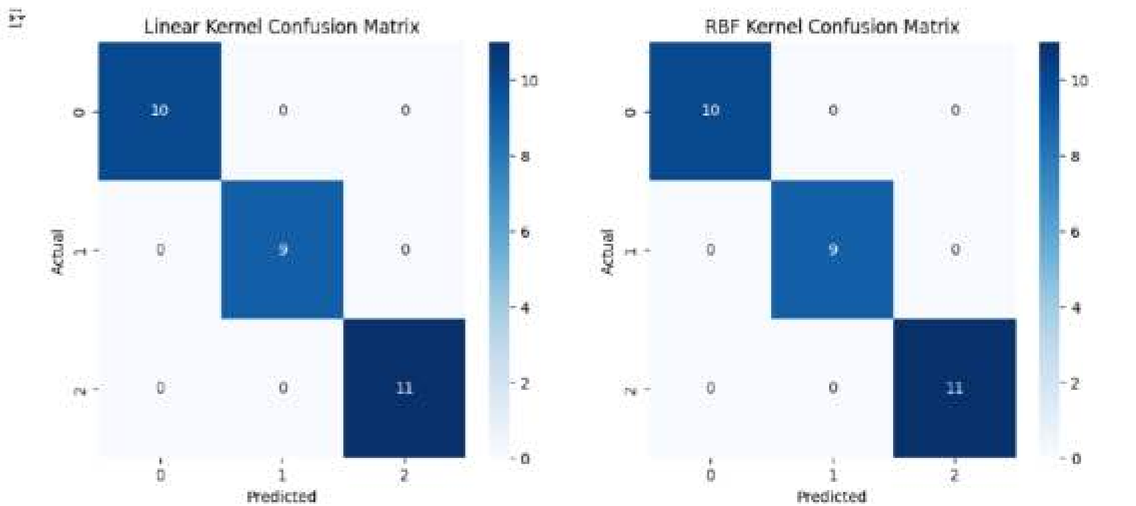
```
accuracy_linear_letter = accuracy_score(y_test_letter, y_pred_linear_letter)
conf_matrix_linear_letter = confusion_matrix(y_test_letter, y_pred_linear_letter)

accuracy_rbf_letter = accuracy_score(y_test_letter, y_pred_rbf_letter)
conf_matrix_rbf_letter = confusion_matrix(y_test_letter, y_pred_rbf_letter)

print(f"Linear Kernel Accuracy (Letter-recognition): {accuracy_linear_letter}")
print(f"RBF Kernel Accuracy (Letter-recognition): {accuracy_rbf_letter}")

# RBF Kernel SVM for Letter Recognition
svm_rbf_letter = SVC(kernel='rbf', random_state=42, probability=True)
svm_rbf_letter.fit(X_train_letter, y_train_letter)
# Confusion Matrices
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(25, 12))

sns.heatmap(conf_matrix_linear_letter, annot=True, fmt='d', cmap='Blues', ax=ax1)
ax1.set_title("Linear Kernel Confusion Matrix")
ax1.set_xlabel('Predicted')
ax1.set_ylabel('Actual')

sns.heatmap(conf_matrix_rbf_letter, annot=True, fmt='d', cmap='Blues', ax=ax2)
ax2.set_title("RBF Kernel Confusion Matrix")
ax2.set_xlabel('Predicted')
ax2.set_ylabel('Actual')

plt.show()
# Plotting ROC curve for Linear Kernel
fpr, tpr, thresholds = roc_curve(y_test_letter,
svm_linear_letter.predict_proba(X_test_letter)[:, 1], pos_label=1)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
```

```
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

Output:

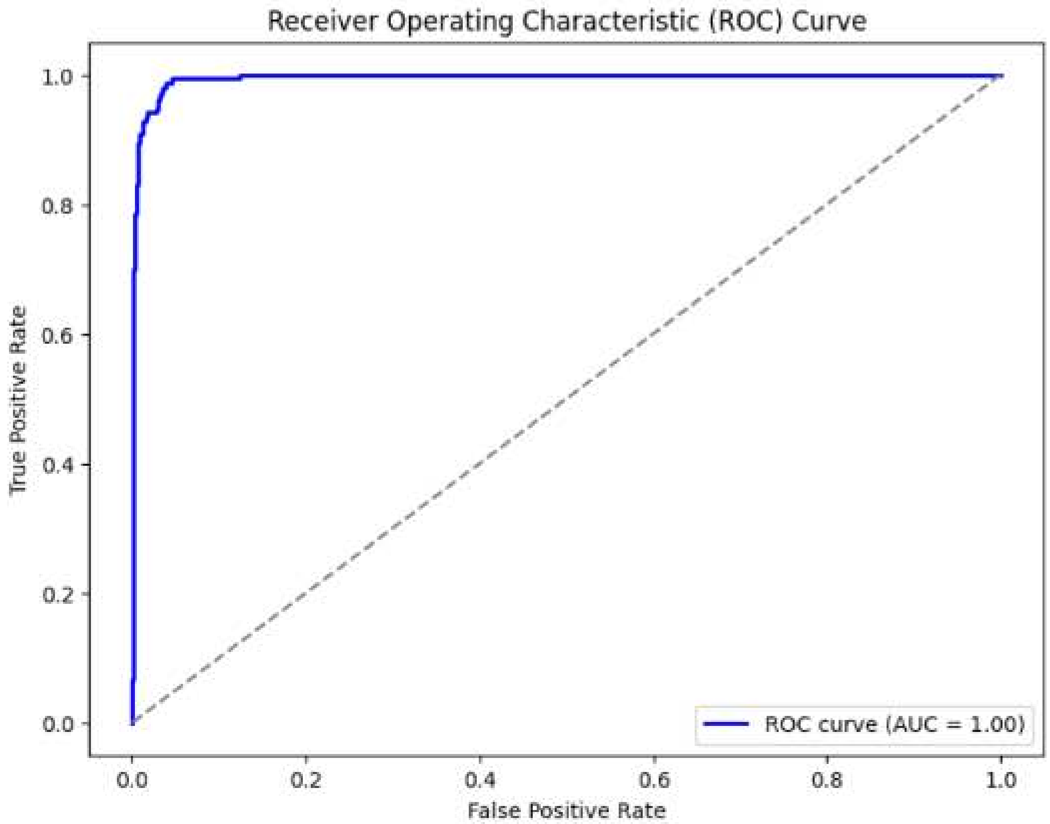
Linear Kernel Accuracy: 1.0

RBF Kernel Accuracy: 1.0



Linear Kernel Accuracy (Letter-recognition): 0.8545

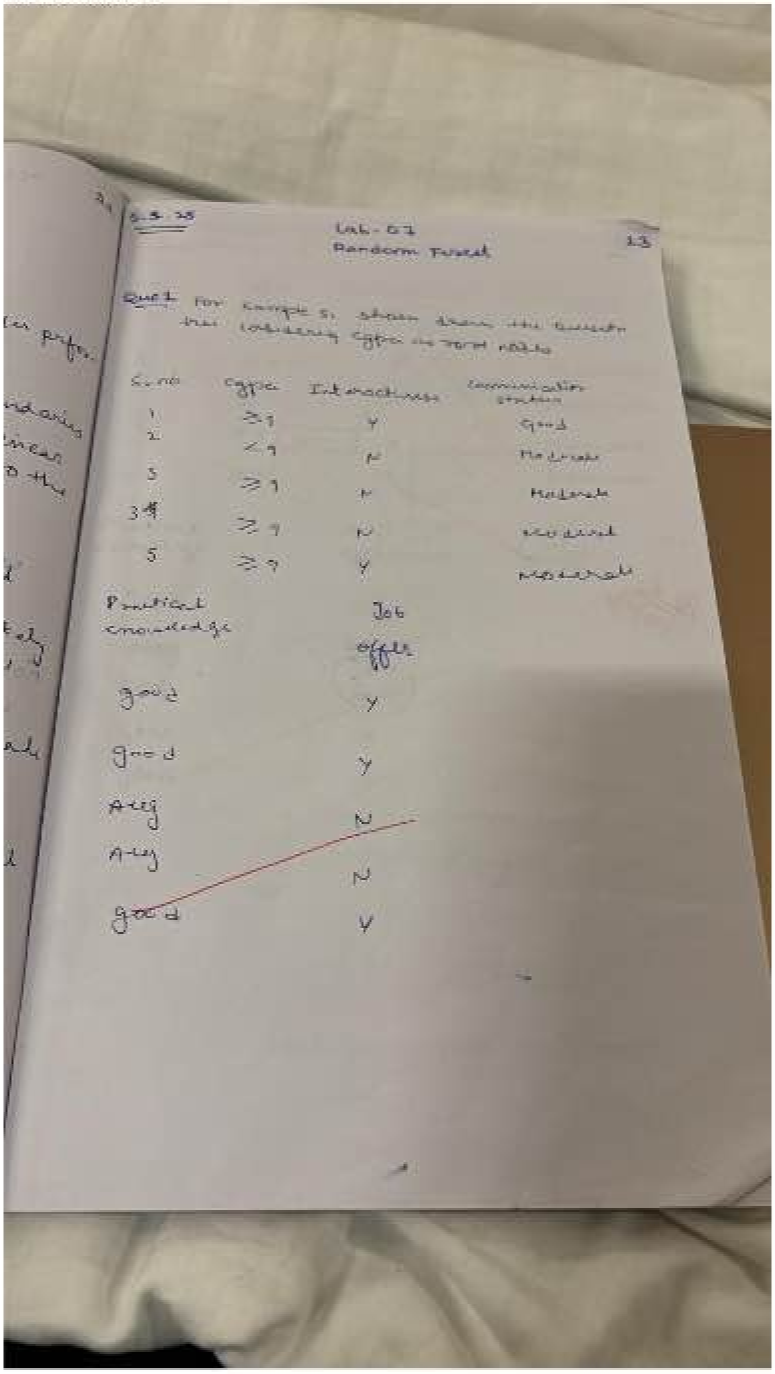
RBF Kernel Accuracy (Letter-recognition): 0.9305

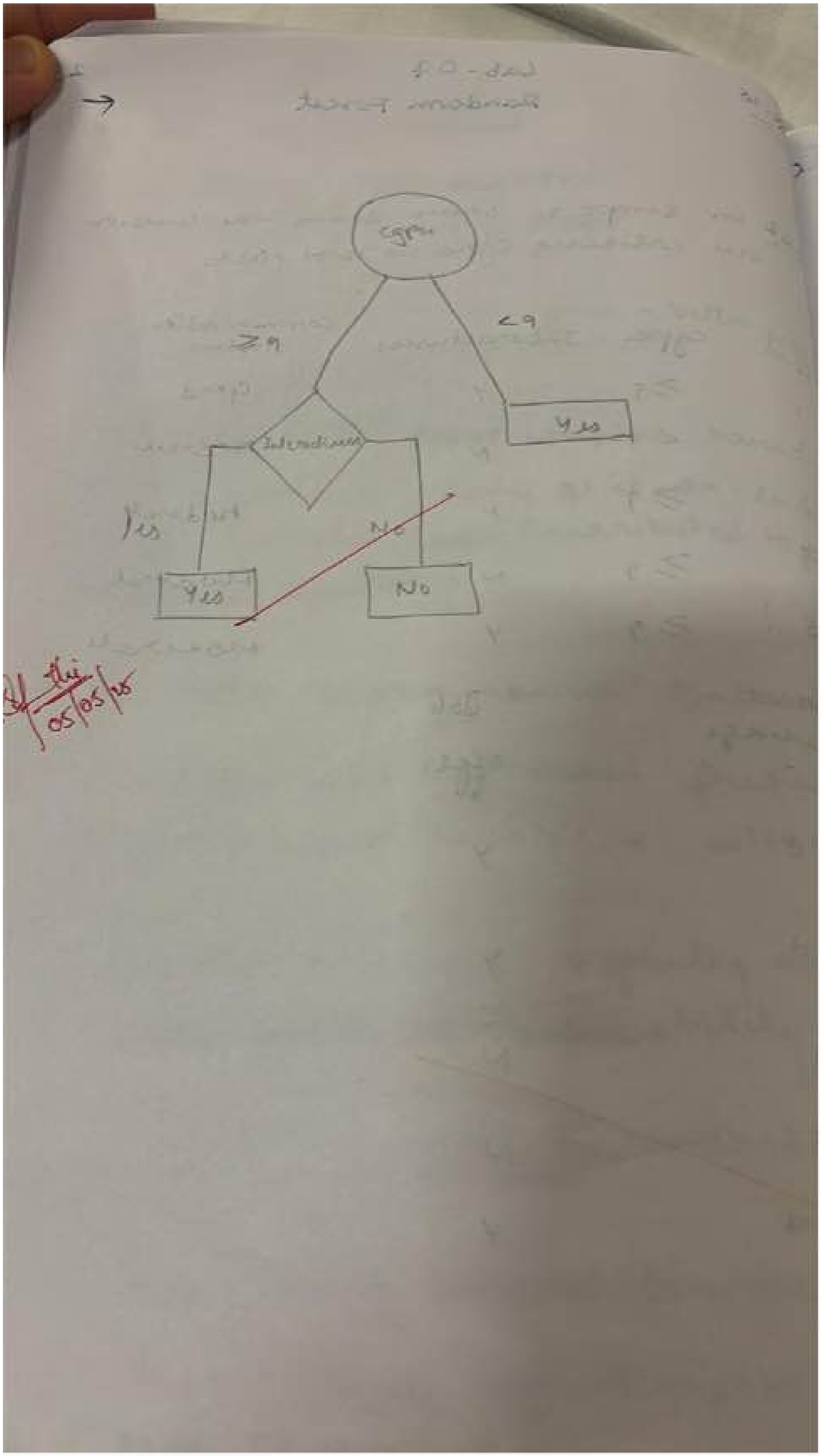

Receiver Operating Characteristic (ROC) Curve
True Positive Rate
False Positive Rate
ROC curve (AUC = 1.00)
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

Program 8

Implement Random forest ensemble method on a given dataset.

Screenshots:




CGPA
≥9
<9
Interactive
Yes
No
Yes
Yes
No
05/05/25

2. For sample shown 3, draw the decision tree considering interactions as root

S. no.	cgpa	interactiveness	communication skills	Practical knowledge	Job offer.
1	<9	N	Moderate	good	Y
2.	≥ 9	N	Moderate	Avg	N
3.	≥ 9	N	Moderate	Avg	N
5.	≥ 9	Y	Moderate	good	Y
6.	≥ 9	Y	Moderate	good	Y

Observation for iris Dataset :

⇒ The Best Accuracy score is 100% and confusion Matrix is

$$\begin{bmatrix} [19 & 0 & 0] \\ [0 & 13 & 0] \\ [0 & 0 & 13] \end{bmatrix}$$

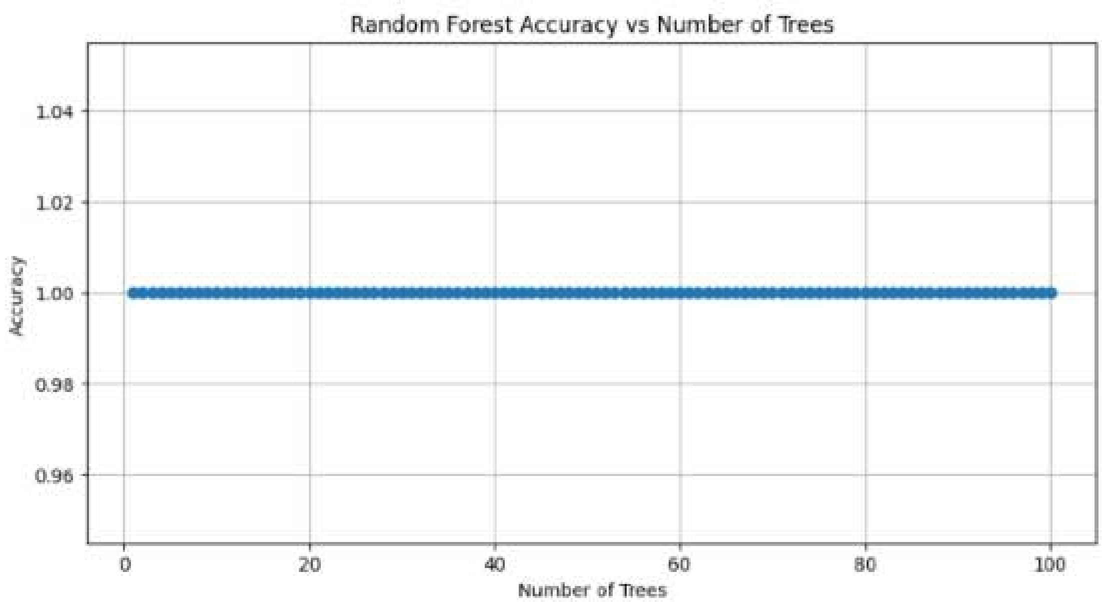
using 100 trees or 1 tree.

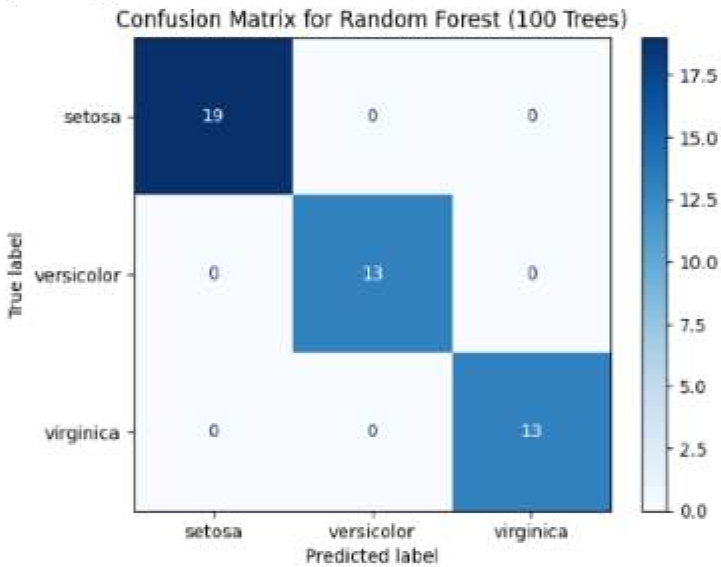
Code:
```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Default Random Forest with n_estimators = 10
rf_default = RandomForestClassifier(n_estimators=10, random_state=42)
rf_default.fit(X_train, y_train)
y_pred_default = rf_default.predict(X_test)
default_accuracy = accuracy_score(y_test, y_pred_default)
print(f"Default RF accuracy (10 trees): {default_accuracy:.4f}")

# Fine-tune n_estimators
accuracies = []
tree_counts = range(1, 101)  # try from 1 to 100 trees

for n in tree_counts:
    rf = RandomForestClassifier(n_estimators=n, random_state=42)
    rf.fit(X_train, y_train)
    y_pred = rf.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    accuracies.append(acc)

# Find best accuracy and corresponding number of trees
best_accuracy = max(accuracies)
best_n = tree_counts[accuracies.index(best_accuracy)]
print(f"Best RF accuracy: {best_accuracy:.4f} using {best_n} trees")

# Plot accuracy vs number of trees
plt.figure(figsize=(10, 5))
plt.plot(tree_counts, accuracies, marker='o')
plt.title("Random Forest Accuracy vs Number of Trees")
plt.xlabel("Number of Trees")
plt.ylabel("Accuracy")
```

```
plt.grid(True)
plt.show()
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)

# Predict on the test set
y_pred = clf.predict(X_test)

# Compute the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Print the confusion matrix
print("Confusion Matrix:")
print(cm)

# Optional: Plot the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=iris.target_names)
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix for Random Forest (100 Trees)")
plt.show()
```

Output:

Default RF accuracy (10 trees): 1.0000
Best RF accuracy: 1.0000 using 1 trees

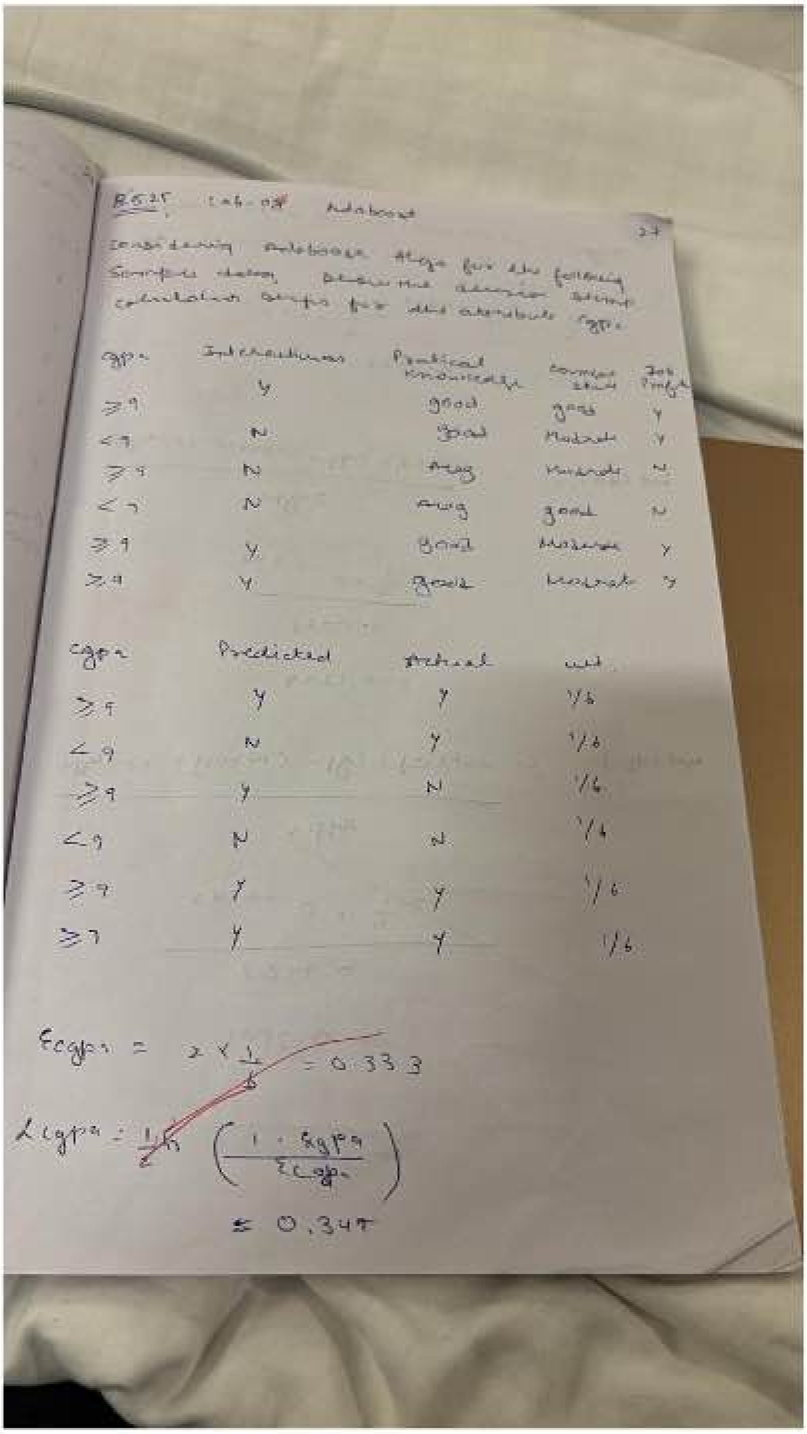



Confusion Matrix for Random Forest (100 Trees)
setosa
versicolor
virginica
True label
19
0
0
0
13
0
0
0
13
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
setosa
versicolor
virginica
Predicted label

Program 9

Implement Boosting ensemble method on a given dataset.

Screenshots:



Z_{gpc} = wt (correct) × no. of correct × $e^{-\alpha_{gpc}}$
+ wt (wrong) × no. of wrong × $e^{+\alpha_{gpc}}$

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

$$= 0.9428$$

$$wt(d_i)_{i+1} = \frac{wt(d_i)\ (gpc\ (correct)) \times e^{-\alpha_{gpc}}}{Z_{gpc}}$$

$$= \frac{\frac{1}{6} \times e^{-0.347}}{0.9428}$$

$$= 0.1249$$

$$wt(d_i)_{i+1} = \frac{wt(d_i)\ (gpc\ (wrong)) \times e^{+\alpha_{gpc}}}{Z_{gpc}}$$

$$= \frac{\frac{1}{6} \times 0^{+0.347}}{0.9428}$$

$$= 0.2501$$

[illegible]

[illegible] dataset"

=> The best Accuracy score is 0.8812.

confusion matrix

$$\begin{bmatrix} 10406 & 613 \\ 1864 & 1680 \end{bmatrix}$$

using 42 trees.

Code:

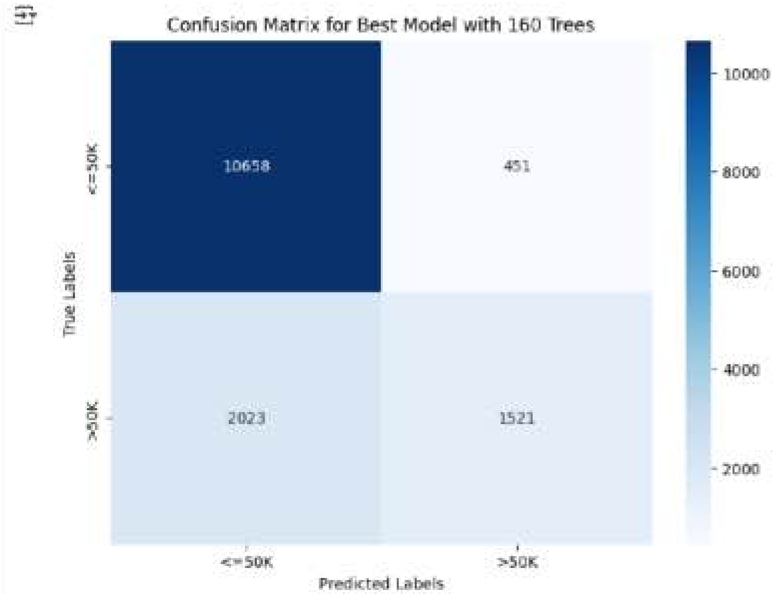
```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
X = df.drop('income_level', axis=1)
y = df['income_level']

# Split the dataset into training and testing sets (70% training, 30% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# 1. Build AdaBoost Classifier with default n_estimators (10)
ada_default = AdaBoostClassifier(n_estimators=10, random_state=42)
ada_default.fit(X_train, y_train)
y_pred_default = ada_default.predict(X_test)
accuracy_default = accuracy_score(y_test, y_pred_default)

print(f"Accuracy with default n_estimators (10): {accuracy_default:.4f}")
best_accuracy = 0
best_n_estimators = 10

for n in range(10, 201, 10):
    ada_tuned = AdaBoostClassifier(n_estimators=n, random_state=42)
    ada_tuned.fit(X_train, y_train)

    # Predict and calculate accuracy
    y_pred_tuned = ada_tuned.predict(X_test)
    accuracy_tuned = accuracy_score(y_test, y_pred_tuned)

    # Track the best accuracy and corresponding n_estimators
    if accuracy_tuned > best_accuracy:
        best_accuracy = accuracy_tuned
```

```
        best_n_estimators = n

print(f"Best accuracy: {best_accuracy:.4f} with n_estimators = {best_n_estimators}")
ada_best = AdaBoostClassifier(n_estimators=best_n_estimators, random_state=42)
ada_best.fit(X_train, y_train)

y_pred_best = ada_best.predict(X_test)

cm = confusion_matrix(y_test, y_pred_best)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['<=50K', '>50K'],
        yticklabels=['<=50K', '>50K'])
plt.title(f"Confusion Matrix for Best Model with {best_n_estimators} Trees")
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```

Output:

Accuracy with default n_estimators (10): 0.8277

Best accuracy: 0.8312 with n_estimators = 160



Accuracy: 0.8312

Precision (for >50K): 0.7713

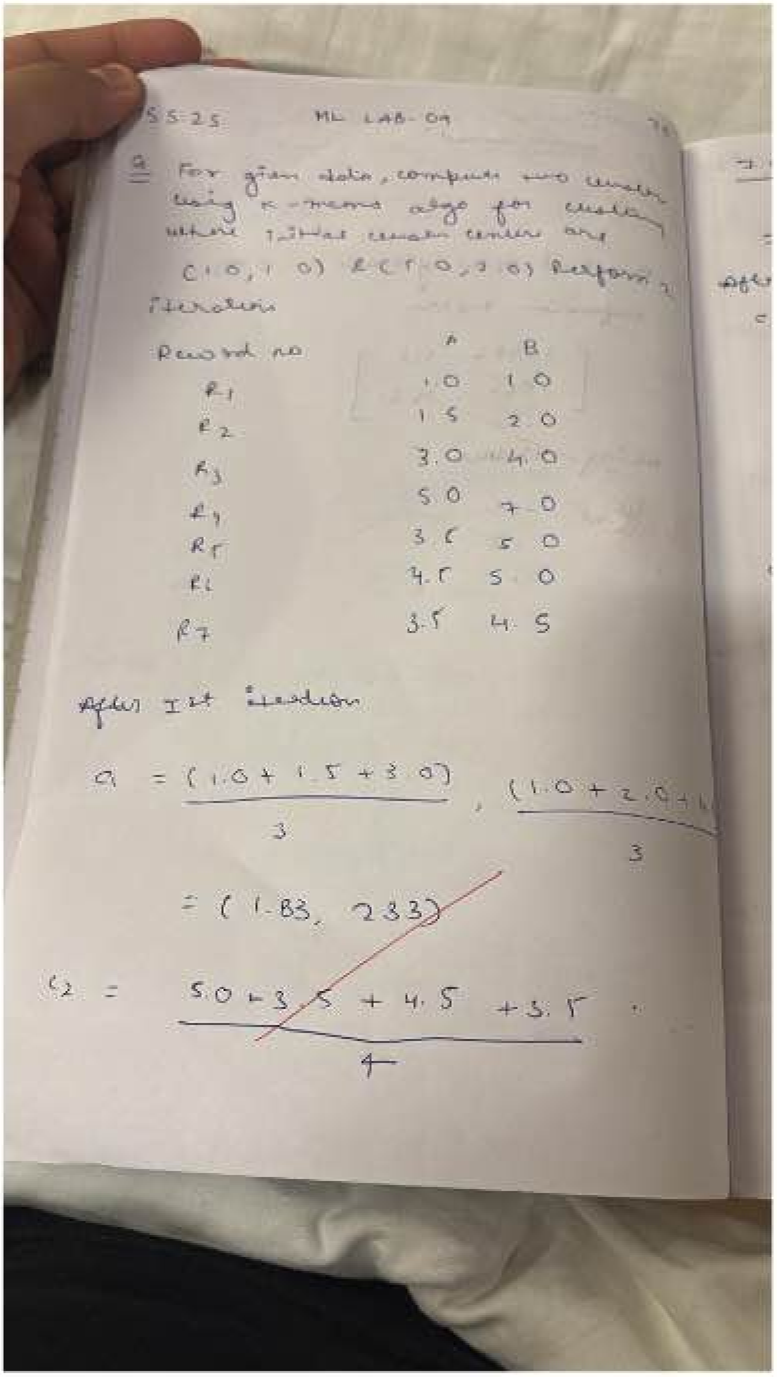
Recall (for >50K): 0.4292

F1-Score (for >50K): 0.5515

Program 10

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

Screenshots:



5-5-25 ML LAB-09

Q For given data, compute two clusters using k-means algo for clustering where initial cluster centers are (1.0, 1.0) & (5.0, 7.0) Perform 2 iterations

Record no	A	B
R1	1.0	1.0
R2	1.5	2.0
R3	3.0	4.0
R4	5.0	7.0
R5	3.5	5.0
R6	4.5	5.0
R7	3.5	4.5

After Ist iteration

$c_1 = \frac{(1.0 + 1.5 + 3.0)}{3}, \frac{(1.0 + 2.0 + 4 \ldots}{3}$

$= (1.83, 2.33)$

$c_2 = \frac{5.0 + 3.5 + 4.5 + 3.5}{4}$

$$\frac{7.0+5.0+5.0+4.5}{4}$$

91

$= (4.12, 5.37)$

After 2nd iteration

$$c_1 = \frac{1.0+1.5}{2}, \frac{1+2}{2}$$

$$\frac{2.5}{2} = 1.25 \quad = \frac{3}{2} = 1.5$$

$(1.25, 1.5)$

$C_2 = \{R3, R4, R5, R6, R7\}$

$$= \frac{3.0+5.0+3.5+4.5+3.5}{5}$$

$$= \frac{4.0+7.0+5.0+5.0+4.5}{5}$$

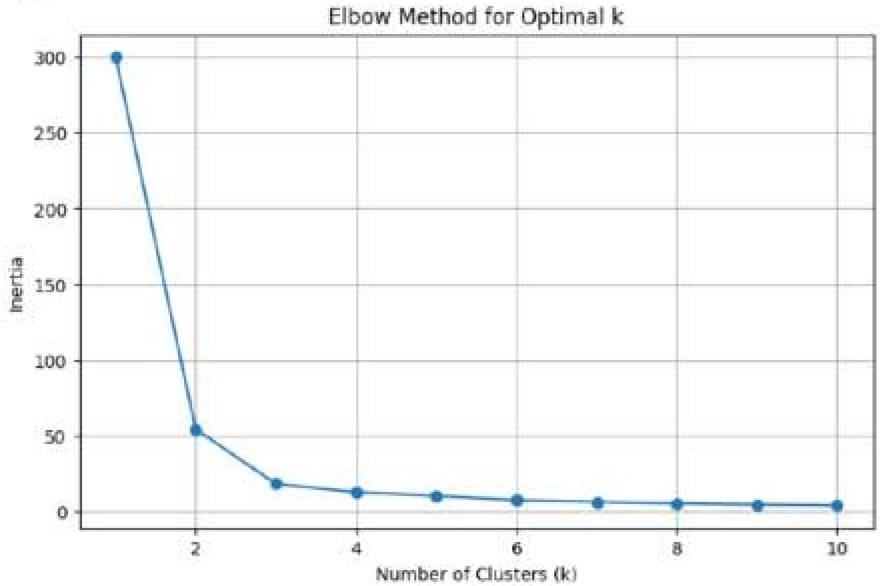
$= (3.9, 5.1)$

for iris Dataset, optimal k-value obtained =3

16/05/25

```python
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

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

# Select only petal length and width
X = data[["petal_length", "petal_width"]]

# Check if scaling helps (KMeans is sensitive to scale)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Elbow method: Try k from 1 to 10 and compute inertia
inertias = []
k_range = range(1, 11)
for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    inertias.append(kmeans.inertia_)

# Plot elbow graph
plt.figure(figsize=(8, 5))
plt.plot(k_range, inertias, marker='o')
plt.title("Elbow Method for Optimal k")
plt.xlabel("Number of Clusters (k)")
plt.ylabel("Inertia")
plt.grid(True)
plt.show()
```

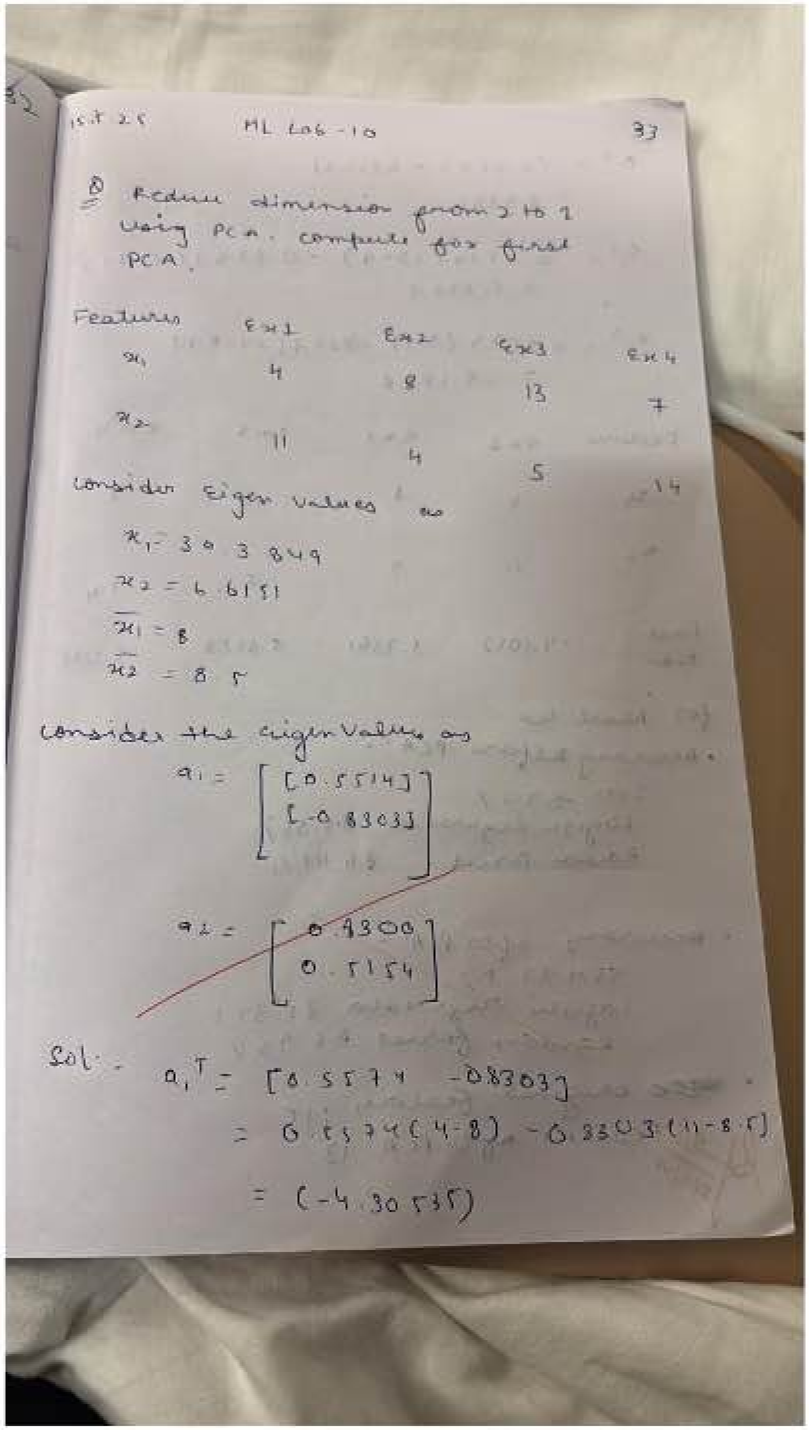
Output:


Elbow Method for Optimal k
300
250
200
150
100
50
0
Inertia
2
4
6
8
10
Number of Clusters (k)

Program 11

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

Screenshots:



15.7.25 ML Lab-10 33

Q Reduce dimension from 2 to 1 using PCA. compute for first PCA.

Features	Ex1	Ex2	Ex3	Ex4
x_1	4	8	13	7
x_2	11	4	5	14

consider eigen values as

$\lambda_1 = 30.3849$

$\lambda_2 = 6.6151$

$\bar{x}_1 = 8$

$\bar{x}_2 = 8.5$

consider the eigen values as

$$a_1 = \begin{bmatrix} [0.5514] \\ [-0.8303] \end{bmatrix}$$

$$a_2 = \begin{bmatrix} 0.8300 \\ 0.5154 \end{bmatrix}$$

Sol: $a_1^T = [0.5574 \quad -0.8303]$

$= 0.5574(4-8) - 0.8303(11-8.5)$

$= (-4.30535)$

$e_1^T = [0.5574 \; -0.8303]$
$= 3.7361$

$e_1^T = 0.5574(13-8) - 0.8303(5-8.5)$
$= 5.6928$

$e_1^T = 0.5574(7-8) - 0.8303[14-8.5]$
$= -5.1238$

Features	ex 1	ex 2	ex 3	ex 4
x_1	4	8	13	7
x_2	11	4	5	14
First PCA	-4.3052	3.7361	5.6928	-5.1238

for heart csv

- Accuracy Before PCA:-
  - SVM 87.5%
  - Logistic Regression 85.33%
  - Random forest 86.41%

- Accuracy after PCA:-
  - SVM 87.5%
  - Logistic regression 85.34%
  - Random forest 86.96%

original features : 15
Features after PCA: 13

16/05/25

Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

# Load the dataset (replace with your own file path)
data = pd.read_csv("heart.csv")

# Display first few rows to understand the dataset structure
print(data.head())

# Encode categorical columns using Label Encoding
label_encoder = LabelEncoder()

# Label Encoding for 'Sex', 'RestingECG', 'ExerciseAngina', and 'ST_Slope'
data['Sex'] = label_encoder.fit_transform(data['Sex'])
data['RestingECG'] = label_encoder.fit_transform(data['RestingECG'])
data['ExerciseAngina'] = label_encoder.fit_transform(data['ExerciseAngina'])
data['ST_Slope'] = label_encoder.fit_transform(data['ST_Slope'])

# One Hot Encoding for 'ChestPainType' (if necessary, based on dataset)
data = pd.get_dummies(data, columns=['ChestPainType'], drop_first=True)

# Split data into features and target
X = data.drop("HeartDisease", axis=1)  # Features
y = data["HeartDisease"]  # Target
```

```python
# Train-test split (80-20 split)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Apply scaling using StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
# Build and evaluate the models: SVM, Logistic Regression, and Random Forest
models = {
    "SVM": SVC(),
    "Logistic Regression": LogisticRegression(),
    "Random Forest": RandomForestClassifier()
}

# Train and evaluate models without PCA
for model_name, model in models.items():
    model.fit(X_train_scaled, y_train)
    y_pred = model.predict(X_test_scaled)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"{model_name} Accuracy without PCA: {accuracy:.4f}")

# Apply PCA for dimensionality reduction
pca = PCA(n_components=0.95)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)

# Train and evaluate models with PCA
for model_name, model in models.items():
    model.fit(X_train_pca, y_train)
    y_pred = model.predict(X_test_pca)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"{model_name} Accuracy with PCA: {accuracy:.4f}")
```

```python
# Plotting the accuracy comparison (without PCA vs with PCA)
accuracies_without_pca = []
accuracies_with_pca = []
for model_name, model in models.items():
    model.fit(X_train_scaled, y_train)
    y_pred = model.predict(X_test_scaled)
    accuracies_without_pca.append(accuracy_score(y_test, y_pred))

    model.fit(X_train_pca, y_train)
    y_pred = model.predict(X_test_pca)
    accuracies_with_pca.append(accuracy_score(y_test, y_pred))

# Bar plot comparison
labels = list(models.keys())
x = range(len(models))

plt.figure(figsize=(10, 5))
plt.bar(x, accuracies_without_pca, width=0.4, label='Without PCA', align='center')
plt.bar(x, accuracies_with_pca, width=0.4, label='With PCA', align='edge')
plt.xlabel("Model")
plt.ylabel("Accuracy")
plt.title("Model Accuracy Comparison (With and Without PCA)")
plt.xticks(x, labels)
plt.legend()
plt.show()
```

Output:

```
 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 without PCA: 0.8587

Logistic Regression Accuracy without PCA: 0.8424

Random Forest Accuracy without PCA: 0.8641

SVM Accuracy with PCA: 0.8750

Logistic Regression Accuracy with PCA: 0.8478

Random Forest Accuracy with PCA: 0.8424