VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

Mohammed Zeeshan Umar (1BM22CS160)

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 Zeeshan Umar (1BM22CS160),** 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.

Lab faculty Incharge Name Assistant Professor Department of CSE, BMSCE	Dr. Kavitha Sooda Professor & HOD Department of CSE, BMSCE

Index

Github Link: https://github.com/ZeeshanBMSCE/1BM22CS160_ML_LAB-.git

Program 1

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

Screenshot

3/3/25

Lab-1

Write python code, consider filename as "housing.csv"

(i). To load .csv file into the data frame.
ii. To display information of all columns
iii). To display statistical information of all numerical.
iv). To display the count of unique labels for "Ocean Proximity" column.
v). To display which attributes (columns) in a dataset have missing values count greater than zero.

(i). ~~pd.~~ df = pd.read_csv ("housing.csv")

(ii) print (df)

(iii) print (df.describe ()).

(iv). print (df ["Ocean Proximity"].value_count ())

(v). missing_value = df.isnull().sum()
missing_column = ~~df~~ missing_values[missing_values > 0]
print (missing_column).

1. Which column in the dataset had missing values? How did you handlee them?

Dropped the missing values with the help of ~~drop na~~ dropna since the number of rows are less with missing value is less compatibility

2. What were the categorial ~~data~~ columns did you identify in the dataset? How did you encode them?

diabetes.csv : ["Gender", "Class"]
adult.csv : ["workclass", "education", "mar status", "occupation", "relationship", "gender", "native_country", "income"].

and we used LabelEncounter to encode it.

3. What is the difference between Min-Max Scaling and standardization? When would you use one over the other?

Both MinMax Scaling and standardization are feature scaling techniques.

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

we use min max when we do not have significant outlier.
and we use standardization when we have outliers.

Code:

```
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

#**Diabetes Dataset**
df=pd.read_csv('/content/Dataset of Diabetes .csv')
df.head()
df.shape
print(df.info())
# Summary statistics
print(df.describe())
missing_values=df.isnull().sum()
print(missing_values[missing_values > 0])
categorical_cols = df.select_dtypes(include=['object']).columns
print("Categorical columns identified:", categorical_cols)
if len(categorical_cols) > 0:
    df = pd.get_dummies(df, columns=categorical_cols, drop_first=True)
    print("\nDataFrame after one-hot encoding:")
    print(df.head())
else:
    print("\nNo categorical columns found in the dataset.")
from sklearn.preprocessing import MinMaxScaler, StandardScaler
import pandas as pd

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

scaler = MinMaxScaler()
df_minmax = df.copy()  # Create a copy to avoid modifying the original
df_minmax[numerical_cols] = scaler.fit_transform(df[numerical_cols])

scaler = StandardScaler()
df_standard = df.copy()
df_standard[numerical_cols] = scaler.fit_transform(df[numerical_cols])
print("\nDataFrame after Min-Max Scaling:")
print(df_minmax.head())
print("\nDataFrame after Standardization:")
print(df_standard.head())

#**Adult Income Dataset**
df1=pd.read_csv('/content/adult.csv')
df1.head()
df1.shape
```

```
print(df1.info())
# Summary statistics
print(df.describe())
missing_values=df1.isnull().sum()
print(missing_values[missing_values > 0])
categorical_cols = df1.select_dtypes(include=['object']).columns
print("Categorical columns identified:", categorical_cols)
if len(categorical_cols) > 0:
    df1 = pd.get_dummies(df1, columns=categorical_cols, drop_first=True)
    print("\nDataFrame after one-hot encoding:")
    print(df.head())
else:
    print("\nNo categorical columns found in the dataset.")
from sklearn.preprocessing import MinMaxScaler, StandardScaler
import pandas as pd

numerical_cols = df1.select_dtypes(include=['number']).columns

scaler = MinMaxScaler()
df_minmax = df1.copy()  # Create a copy to avoid modifying the original
df_minmax[numerical_cols] = scaler.fit_transform(df1[numerical_cols])

scaler = StandardScaler()
df_standard = df1.copy()
df_standard[numerical_cols] = scaler.fit_transform(df1[numerical_cols])
print("\nDataFrame after Min-Max Scaling:")
print(df_minmax.head())
print("\nDataFrame after Standardization:")
print(df_standard.head())
```

PROGRAM 2 Demonstrate various data pre-processing techniques for a given dataset

Screenshot

10/03/25

LAB - 2

Date :
Page No. :

Demonstrate the steps to build a machine-Learning model that predicts the median housing price using the California housing price dataset.

1. Perform the describe and info steps:

```
import pandas as pd
housing = pd.read_csv("content/drive/...").
print(housing.info())
print(housing.describe()).
```

Output : column = 9
rows = 20640.

2. Plot histogram for each feature

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 ages of houses are distributed.

Date :
Page No. :

3. Demonstrate process of creating set & differences b/w random & stratified test.

Random	Stratified.
Splits data randomly into training & testing	Both are representative of the overall distribution of a feature.
May not preserve the distribution of key feature in training & testing set.	Preserves the distribution of key features across training & testing.

4. list geographical feature:

This graph visualizes housing prices in relation to their geographic location, with the color representing medium house value and size represents population

5. Which feature correlates to maximum:

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

Date :
Page No. :

6. list the feature that could improve correlation.

By creating rooms per household & bedrooms per room.

The median house value is Improved.

7. ~~total bedrooms~~ list the features that ~~leads~~ needs data to be cleaned & demonstrate cleaning process.

total bedrooms.

8. Categorial data to numerical data.

Yes, ocean-proximity is categorial feature and method used to convert is OneHotEncoder().

9. Importante Scaling feature :-

- ensure all feature contribute equally to result.

Technique used StandardScaler & MinmaxScaler.

Code

```
import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

df=pd.read_csv('housing.csv')

df.head(2)

df.describe()

df.info()

sns.histplot(df['median_income'], kde=True, color='green')

sns.histplot(df['housing_median_age'])

from sklearn.model_selection import train_test_split


X = df.drop("median_house_value", axis=1)

y = df["median_house_value"]


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

X = df.drop("median_house_value", axis=1)

y = df["median_house_value"]

df["income_cat"] = pd.cut(df["median_house_value"],

bins=[0, 100000, 200000, 300000, 400000, np.inf],

labels=[1, 2, 3, 4, 5])

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

stratify=df["income_cat"])
```

```
train_set = X_train.copy()

train_set["median_house_value"] = y_train

train_set.plot(kind="scatter", x="longitude", y="latitude", alpha=0.4,s=train_set["population"]/100,
label="population",figsize=(10,7), c="median_house_value", cmap=plt.get_cmap("jet"),

colorbar=True)

plt.legend()

numerical_columns = df.select_dtypes(include=['float64', 'int64'])

correlation_matrix = numerical_columns.corr()

print(correlation_matrix["median_house_value"].sort_values(ascending=False))

df.plot(kind="scatter", x="median_income", y="median_house_value", alpha=0.1)

# Combine 'median_income' and 'households'

df["income_households"] = df["median_income"] * df["households"]


numerical_columns = df.select_dtypes(include=['float64', 'int64'])

correlation_matrix = numerical_columns.corr()

print(correlation_matrix["median_house_value"].sort_values(ascending=False))

df.plot(kind="scatter", x="income_households", y="median_house_value", alpha=0.1)

plt.show()

missing_values = df.isnull().sum()

print(missing_values[missing_values > 0])

h=df

h.dropna(subset=["total_bedrooms"])

from sklearn.preprocessing import OneHotEncoder

df1=pd.read_csv('housing.csv')

hc=df1[["ocean_proximity"]]
```

encoder=OneHotEncoder()

hc_encoded=encoder.fit_transform(hc).toarray()

hc_1hot_df = pd.DataFrame(hc_encoded, columns=encoder.get_feature_names_out(hc.columns))

hc_1hot_df.head()

Feature scaling is crucial in machine learning for several reasons, particularly when using algorithms that are sensitive to the scale of features. Here's a breakdown of its importance:

1. **Improved Performance of Distance-Based Algorithms:**

2. **Faster Convergence of Gradient Descent:**

3. **Improved Regularization:**

4. **Better Interpretation of Coefficients:**

5. **Numerical Stability:**

```
from sklearn.base import BaseEstimator, TransformerMixin

from sklearn.pipeline import Pipeline

from sklearn.compose import ColumnTransformer

from sklearn.preprocessing import StandardScaler


# Custom transformer to add engineered attributes

class CombinedAttributesAdder(BaseEstimator, TransformerMixin):

    def __init__(self, add_bedrooms_per_room=True):
```

```
        self.add_bedrooms_per_room = add_bedrooms_per_room
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        # Assumes X is a NumPy array with the following columns:
        # total_rooms (index 3), total_bedrooms (index 2), population (index 4), households (index 5)
        rooms_per_household = X[:, 3] / X[:, 5]
        population_per_household = X[:, 4] / X[:, 5]
        if self.add_bedrooms_per_room:
            bedrooms_per_room = X[:, 2] / X[:, 3]
            return np.c_[X, rooms_per_household, population_per_household, bedrooms_per_room]
        else:
            return np.c_[X, rooms_per_household, population_per_household]

# Identify numerical and categorical columns
num_attribs = df1.drop("ocean_proximity", axis=1).columns  # All numeric columns
cat_attribs = ["ocean_proximity"]

# Build numerical pipeline: impute missing values, add new attributes, then scale
num_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy="median")),
    ('attribs_adder', CombinedAttributesAdder()),
    ('std_scaler', StandardScaler()),
])
```

```
# Build the full pipeline combining numerical and categorical processing
full_pipeline = ColumnTransformer([
    ("num", num_pipeline, num_attribs),
    ("cat", OneHotEncoder(), cat_attribs),
])

# Process the dataset using the pipeline
housing_prepared = full_pipeline.fit_transform(housing)
print("Shape of processed data:", housing_prepared.shape)
```

Output :

```
Missing values in Diabetes dataset after removal:
 ID           0
No_Pation    0
Gender       0
AGE          0
Urea         0
Cr           0
HbA1c        0
Chol         0
TG           0
HDL          0
LDL          0
VLDL         0
BMI          0
CLASS        0
dtype: int64

Missing values in Adult dataset after removal:
 age                0
workclass          0
fnlwgt             0
education          0
educational-num    0
marital-status     0
occupation         0
relationship       0
race               0
gender             0
capital-gain       0
capital-loss       0
hours-per-week     0
native-country     0
income             0
dtype: int64
```

```
Missing values in Diabetes dataset before removal:
 ID           60
No_Pation    44
Gender       56
AGE          39
Urea         58
Cr           48
HbA1c        42
Chol         47
TG           49
HDL          54
LDL          40
VLDL         46
BMI          51
CLASS        48
dtype: int64

Missing values in Adult dataset before removal:
 age                2304
workclass          2324
fnlwgt             2325
education          2403
educational-num    2392
marital-status     2446
occupation         2480
relationship       2402
race               2379
gender             2463
capital-gain       2369
capital-loss       2376
hours-per-week     2367
native-country     2382
income             2324
dtype: int64
```

```
Categorical Columns in Diabetes Dataset: Index(['Gender', 'CLASS'], dtype='object')
Categorical Columns in Adult Dataset: Index(['workclass', 'education', 'marital-status', 'occupation',
       'relationship', 'race', 'gender', 'native-country', 'income'],
      dtype='object')

Encoded Diabetes Dataset:
      ID  No_Pation  Gender   AGE  Urea    Cr  HbA1c  Chol   TG  HDL  LDL  \
0  502.0    17975.0       0  50.0   4.7  46.0    4.9   4.2  0.9  2.4  1.4   
1  735.0        NaN       1  26.0   4.5  62.0    4.9   3.7  NaN  NaN  2.1   
2  420.0    47975.0       0  50.0   4.7  46.0    4.9   4.2  0.9  2.4  1.4   
3  680.0    87656.0       0  50.0   4.7  46.0    4.9   4.2  0.9  2.4  1.4   
4  504.0    34223.0       1  33.0   NaN  46.0    4.9   4.9  1.0  0.8  2.0   

   VLDL   BMI  CLASS  
0   0.5  24.0      0  
1   0.6  23.0      0  
2   0.5  24.0      0  
3   0.5  24.0      0  
4   NaN  21.0      0  

Encoded Adult Dataset:
    age  workclass    fnlwgt  education  educational-num  marital-status  \
0  25.0          4  226802.0          1              7.0               4   
1  38.0          4   89814.0         11              9.0               2   
2  28.0          2  336951.0          7             12.0               2   
3  44.0          4  160323.0         15             10.0               2   
4  18.0          0  103497.0         15             10.0               4   

   occupation  relationship  race  gender  capital-gain  capital-loss  \
0           7             3     2       1           0.0           0.0   
1           5             0     4       1           0.0           0.0   
2          11             0     4       2           0.0           NaN   
3           7             0     2       1        7688.0           0.0   
4           0             3     4       0           0.0           0.0   

   hours-per-week  native-country  income  
0            40.0              39       0  
1            50.0              39       0  
2            40.0              39       1  
3            40.0              39       1  
4            30.0              39       0  
```

```
   Feature  MinMax_Scaled  Standardized
0       10       0.000000     -0.502219
1       20       0.010101     -0.474574
2       30       0.020202     -0.446929
3       40       0.030303     -0.419284
4       50       0.040404     -0.391639
5     1000       1.000000      2.234643
```

PROGRAM 3 Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Screenshot

17/03/25

LAB-3

Date :

Page No.:

Q. Solving the following Linear Regression Problem using Matrix approach. Find linear Regression of the data of week and product sales.

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

$$\beta = ((X^TX)^{-1}X^T)Y$$

$$X^TX = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \end{pmatrix} \times \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{bmatrix} = \begin{bmatrix} 4 & 10 \\ 10 & 30 \end{bmatrix}$$

$$(X^TX)^{-1} = \begin{bmatrix} 4 & 10 \\ 10 & 30 \end{bmatrix}^{-1} = \begin{bmatrix} 1.5 & -0.5 \\ -0.5 & 0.2 \end{bmatrix}$$

$$((X^TX)^{-1}X^T) = \begin{bmatrix} 1.5 & -0.5 \\ -0.5 & 0.2 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0.5 & 0 & -0.5 \\ -0.3 & -0.1 & 0.1 & 0.3 \end{bmatrix}$$

Page No.:

Finally,

$$((X^TX)^{-1}X^T)Y = \begin{bmatrix} 1 & 0.5 & 0 & -0.5 \\ -0.3 & -0.1 & 0.1 & 0.3 \end{bmatrix} \times \begin{bmatrix} 2 \\ 4 \\ 5 \\ 9 \end{bmatrix}$$

$$= \begin{bmatrix} -0.5 \\ -0.6 - 0.4 + 0.5 + 2.7 \end{bmatrix}$$

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

$$y = -0.5 + 2.2x.$$

→ Normal:

$\Sigma x = 10$ $\Sigma x^2 = 30$.

$\Sigma y = 20$. $\Sigma xy = 61$.

$\bar{x} = 2.5$ $\bar{y} = 5$, $\bar{x_i^2} = 7.5$, $\bar{xy} = 15.25$.

$$\frac{\bar{xy} - \bar{x}\bar{y}}{\bar{x_i^2} - (\bar{x})^2}$$

$$= \frac{15.25 - 5 \times 2.5}{7.5 - 6.25} = 2.2$$

$$x_0 = \bar{y} - \bar{x}_1\bar{x} = 5 - 2.2 \times 2.5 = -0.5$$

$$y = 2.2x - 0.5.$$

1. Did you perform any data processing steps?
Yes, handled missing value by filling them with the column mean. Also applied label encoding to categorical (like "State") ~~ca~~ and scaled numerical features for 1000-companies.csv to normalize the data.

2. Did you vizualize the regression line for cannada-per-capita-income.csv?
~~Yes~~.
Yes, the regression line was plotted. The plot shows a strong linear relationship between year & per-capita income, meaning that as the year increases, per capita income also rises.

3. Predicted salary for (12 years experience, 10 testscore, 10 interview score)? ₹
Sol
The predicted salary is printed in the script and depends on the trained model's coefficient.

4. Did you encode categorical variables for 1000-companies.csv?
Yes, the "State" column was encoded using LabelEncoder().

Did you scale the feature? if yes, why?
Yes, because of R & D Spend, Administration, & Marketing Spend have different units, feature scaling (using Standard Scaler()) was applied to improve model performance.

Code

```
# -*- coding: utf-8 -*-

import pandas as pd

import numpy as np

from sklearn import linear_model

import matplotlib.pyplot as plt


df = pd.read_csv('/content/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]])

# -*- coding: utf-8 -*-

import pandas as pd

import numpy as np

from sklearn import linear_model


df = pd.read_csv('/content/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
import pandas as pd
from sklearn.linear_model import LinearRegression

# Load the dataset
df1 = pd.read_csv('/content/canada_per_capita_income.csv')

# Prepare the data
X = df1.year.values.reshape(-1, 1)  # Features (year)
y = df1['per capita income (US$)']   # Target (per capita income)

# Create and train the linear regression model
model = LinearRegression()
model.fit(X, y)

# Predict per capita income for 2020
year_2020 = [[2020]]
predicted_income = model.predict(year_2020)

print(f"Predicted per capita income for Canada in 2020: {predicted_income[0]:.2f}")
```

```python
import pandas as pd

from sklearn.linear_model import LinearRegression

import matplotlib.pyplot as plt


# Load the dataset (canada_per_capita_income.csv)

df1 = pd.read_csv('/content/canada_per_capita_income.csv')


# Prepare the data

X = df1.year.values.reshape(-1, 1)  # Features (year)

y = df1['per capita income (US$)']   # Target (per capita income)


# Create and train the linear regression model

model = LinearRegression()

model.fit(X, y)


# Create the plot

plt.figure(figsize=(8, 6))

plt.scatter(X, y, color='blue', label='Data Points') # Now using the correct X and y

plt.plot(X, model.predict(X), color='red', label='Regression Line')

plt.xlabel('Year')

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

plt.title('Per Capita Income in Canada over Time')

plt.legend()
```

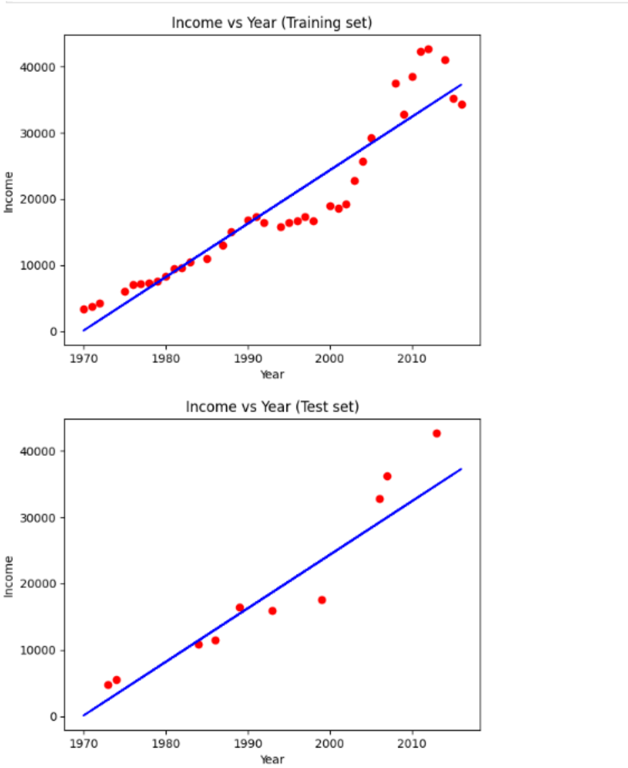
```
plt.grid(True)

plt.show()

import pandas as pd

from sklearn.linear_model import LinearRegression

from sklearn.impute import SimpleImputer


# Load the dataset

df = pd.read_csv('/content/salary.csv')


# Prepare the data

X = df.iloc[:, :-1].values  # Features (years of experience)

y = df.iloc[:, 1].values    # Target (salary)


# Impute missing values with the mean

imputer = SimpleImputer(strategy='mean') # Create an imputer object with strategy as mean

X = imputer.fit_transform(X) # Fit and transform the imputer on feature data 'X'


# Create and train the linear regression model

model = LinearRegression()

model.fit(X, y)


# Predict salary for 12 years of experience

years_experience = [[12]]

predicted_salary = model.predict(years_experience)
```

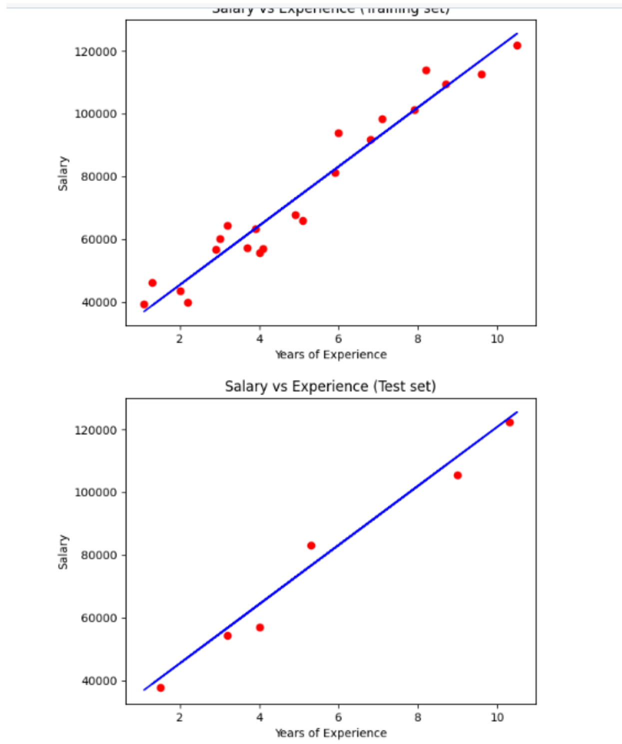
```
print(f"Predicted salary for 12 years of experience: {predicted_salary[0]:.2f}")
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.impute import SimpleImputer

# Load the dataset
df = pd.read_csv('/content/hiring.csv')

# Handle missing values
# Convert 'experience' column to numeric, replacing non-numeric with NaN
df['experience'] = pd.to_numeric(df['experience'], errors='coerce')

imputer = SimpleImputer(strategy='mean')
df['experience'] = imputer.fit_transform(df[['experience']])
df['test_score(out of 10)'] = imputer.fit_transform(df[['test_score(out of 10)']])

# Prepare the data
X = df.drop('salary($)', axis='columns')
y = df['salary($)']

# Create and train the linear regression model
model = LinearRegression()
model.fit(X, y)
```

```
# Predict salaries for the given candidates
candidate1 = [[2, 9, 6]]
candidate2 = [[12, 10, 10]]

predicted_salary1 = model.predict(candidate1)
predicted_salary2 = model.predict(candidate2)

print(f"Predicted salary for candidate 1: ${predicted_salary1[0]:.2f}")
print(f"Predicted salary for candidate 2: ${predicted_salary2[0]:.2f}")
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.compose import ColumnTransformer

# Load the dataset
df = pd.read_csv('/content/1000_Companies.csv')

# Separate features (X) and target (y)
X = df.iloc[:, :-1].values
y = df.iloc[:, 4].values

# Encode categorical data (State)
```

```
labelencoder = LabelEncoder()

X[:, 3] = labelencoder.fit_transform(X[:, 3])


ct = ColumnTransformer(

    transformers=[('encoder', OneHotEncoder(), [3])],

    remainder='passthrough'

)

X = ct.fit_transform(X)


# Avoid dummy variable trap (remove one encoded column)

X = X[:, 1:]


# 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=0)


# Create and train the multiple linear regression model

regressor = LinearRegression()

regressor.fit(X_train, y_train)


# Predict profit for the given values

new_prediction = regressor.predict([[1, 0, 91694.48, 515841.3, 11931.24]])

print(f"Predicted Profit: {new_prediction[0]:.2f}")
```

Output :


Income vs Year (Training set)
Income
Year
0
10000
20000
30000
40000
1970
1980
1990
2000
2010
Income vs Year (Test set)
Income
Year
0
10000
20000
30000
40000
1970
1980
1990
2000
2010


Salary
Years of Experience
40000
60000
80000
100000
120000
2
4
6
8
10
Salary vs Experience (Test set)
Salary
Years of Experience
40000
60000
80000
100000
120000
2
4
6
8
10

PROGRAM 4 Build Logistic Regression Model for a given dataset

Screenshot

24/03/25 Lab-4

Date: Page No.:

Logistic Regression :-

(1). Student - passed or fail based on the study
Intercept $a_1 = -5$ coefficient : $a_1 = 0.8$.

(a). $P(z) = \frac{1}{1+e^{-z}}$ $z = a_0 + a_1 \cdot x$
$z = -5 + 0.8x$.

$\frac{1}{1+e^{5-0.8x}}$

(b). Probability that student who studies for 7 hours will pass :

$P(7) = \frac{1}{1+e^{5-5.6}} = \frac{1}{1+e^{-0.6}} = 0.6456$

(c). Determine predicted class (P/F) for student based on threshold 0.5

As $0.6456 > 0.5$ this statement will Pass.

Q2. Consider $z = [2, 1, 0]$ for 3 classes. Apply softmax function to find probability value of 3 classes.

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

$P(2) = \frac{e^2}{e^0+e^1+e^2} = 0.6652$

Date: Page No.:

$P(1) = \frac{e^1}{e^0+e^1+e^2} = 0.0900$.

$P(0) = \frac{e^0}{e^0+e^1+e^2}$, 0.2448

For HR_comma_sep.csv.

Q.1 Which variable had clear and direct impact on employee retention? Why?

Sol : satisfaction_level, time_spend_company, number_project, average_monthly_hours.

Reason: They showed strong correlation with retention based on EDA.

Q.2 Accuracy of logistic regression model? Good?

Accuracy (eg ~ 79%)

Yes it has good accuracy, it is above baseline, through improvement may be possible.

For Zoo dataset.

i) Did you perform data processing steps?

Dropped animal-name as its not useful for classification.

ii) Where there any missing values?

No missing values detected.

iii) What does the confusion matrix tell you about the performance.

Shows ~~class-wise performance.~~
It shows how many instance of each class were correctly or incorrectly classified. It helps assess accuracy and identify area of miscalculation.

iv) Which class type were most frequently misclassified? Why?

Class types with similar features (eg mammal & humans) were most frequently misclassified. This happened due to overlapping features and insufficient distinction between classes.

Code

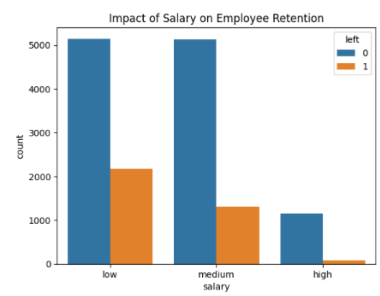
```
import pandas as pd

import numpy as np

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

df.head(3)

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

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

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

import matplotlib.pyplot as plt

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

plt.title('Employee Retention vs Salary')
```

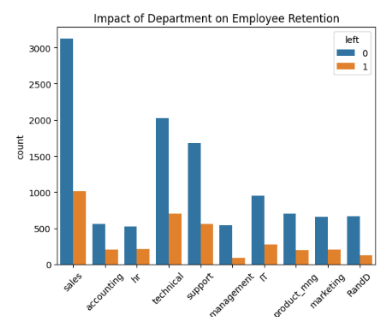
```
plt.xlabel('Salary')

plt.ylabel('Number of Employees')

plt.show()

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

plt.title('Employee Retention vs Department')

plt.xlabel('Department')

plt.ylabel('Number of Employees')

plt.show()

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

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


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


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


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

X = df_with_dummies[X_features]

y = df_with_dummies.left


from sklearn.model_selection import train_test_split

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

```python
from sklearn.linear_model import LogisticRegression

model = LogisticRegression()

model.fit(X_train, y_train)


from sklearn.metrics import accuracy_score

y_pred = model.predict(X_test)


accuracy = accuracy_score(y_test, y_pred)

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

Output :

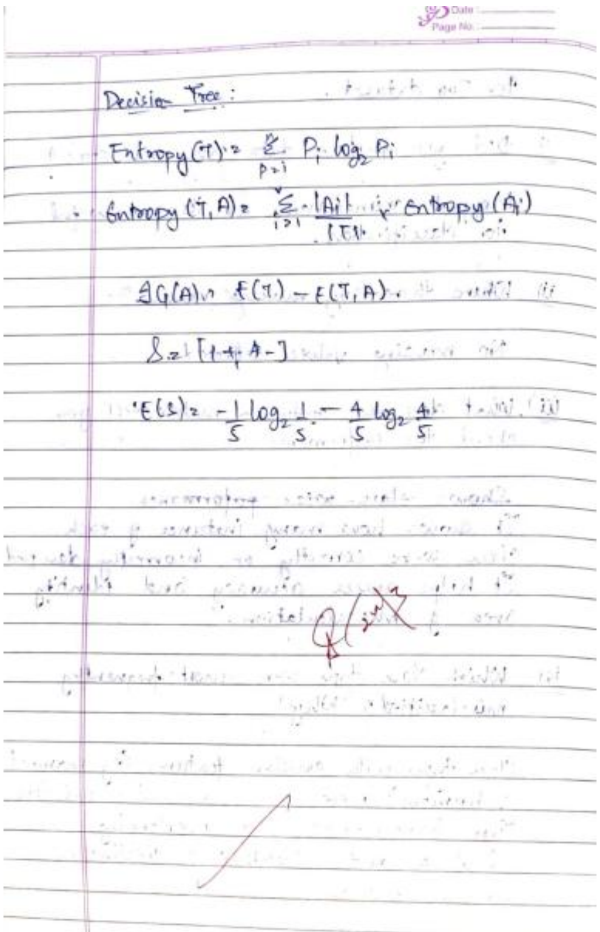
```
   satisfaction_level  last_evaluation  number_project  average_montly_hours  \
0                0.38             0.53               2                   157
1                0.80             0.86               5                   262
2                0.11             0.88               7        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

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

   salary
0     low
1  medium
2  medium
3     low
4     low
<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  last_evaluation  number_project  \
count        14999.000000     14999.000000    14999.000000
mean             0.612834         0.716102        3.803054
std              0.248631         0.171169        1.232592
min              0.090000         0.360000        2.000000
25%              0.440000         0.560000        3.000000
50%              0.640000         0.720000        4.000000
75%              0.820000         0.870000        5.000000
max              1.000000         1.000000        7.000000

       average_montly_hours  time_spend_company  Work_accident          left  \
count          14999.000000        14999.000000   14999.000000  14999.000000
mean             201.050337            3.498233       0.144610      0.238083
std               49.943099            1.460136       0.351719      0.425924
min               96.000000            2.000000       0.000000      0.000000
25%              156.000000            3.000000       0.000000      0.000000
50%              200.000000            3.000000       0.000000      0.000000
75%              245.000000            4.000000       0.000000      0.000000
max              310.000000           10.000000       1.000000      1.000000

       promotion_last_5years
```





PROGRAM 5 Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample.

Screenshot



Code

```
from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy_score, confusion_matrix

from sklearn import tree

import matplotlib.pyplot as plt
```

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

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

clf = DecisionTreeClassifier()

clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

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

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


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

from sklearn.datasets import load_iris
```

```
from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy_score, confusion_matrix

from sklearn import tree

import matplotlib.pyplot as plt


iris = load_iris()

X = iris.data

y = iris.target


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


clf = DecisionTreeClassifier()


clf.fit(X_train, y_train)


y_pred = clf.predict(X_test)


accuracy = accuracy_score(y_test, y_pred)

conf_matrix = confusion_matrix(y_test, y_pred)


print("Accuracy:", accuracy)

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

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

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

plt.show()


import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeRegressor

from sklearn.metrics import mean_absolute_error, mean_squared_error

import numpy as np # import numpy


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


X = data[['Petrol_tax', 'Average_income', 'Paved_Highways',

          'Population_Driver_licence(%)']]

y = data['Petrol_Consumption']


X_train, X_test, y_train, y_test = train_test_split(

    X, y, test_size=0.2, random_state=42)


regressor = DecisionTreeRegressor()


regressor.fit(X_train, y_train)
```

```
y_pred = regressor.predict(X_test)


mae = mean_absolute_error(y_test, y_pred)

mse = mean_squared_error(y_test, y_pred)

rmse = np.sqrt(mse)


print("Mean Absolute Error:", mae)

print("Mean Squared Error:", mse)

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

from sklearn.tree import plot_tree

import matplotlib.pyplot as plt


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

# Assuming 'data' is your original pandas DataFrame

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

plt.show()
```

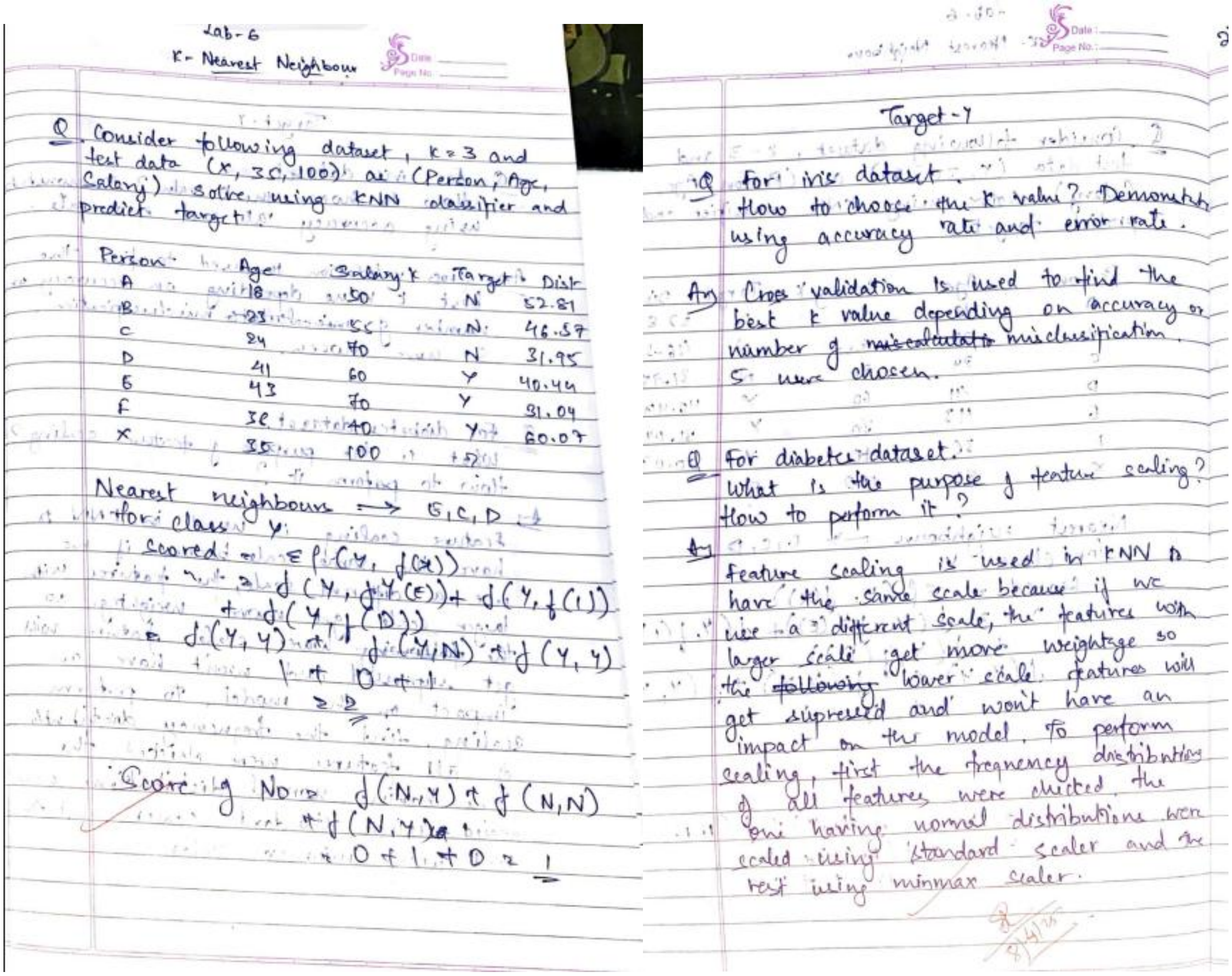
Output :

```
Iris Dataset Results:
Accuracy: 1.00
Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
Classification Report:
                 precision    recall  f1-score   support

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

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

```
Drug Dataset Results:
Accuracy: 1.00
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
```

PROGRAM 6 Build KNN Classification model for a given dataset.

Screenshot


Lab-6

K- Nearest Neighbour

Q. Consider following dataset, k=3 and test data (x, 35, 100) as (Person, Age, Salary). Solve using KNN classifier and predict target.

Person	Age	Salary k	Target	Dist
A	18	50	N	52.81
B	23	55	N	46.57
C	24	70	N	31.95
D	41	60	Y	40.44
E	43	70	Y	31.04
F	36	40	Y	60.07
x	35	100	+ ?	

Nearest neighbours → E, C, D

For class Y:

Score d: = Σ f(Y, f(x))
= d(Y, f(E)) + d(Y, f(C)) + d(Y, f(D))
= d(Y, Y) + d(Y, N) + d(Y, Y)
= 1 + 0 + 1
= 2

Score N = d(N, Y) + d(N, N) + d(N, Y)
= 0 + 1 + 0 = 1

Target-Y

Q For iris dataset.
How to choose the K value? Demonstrate using accuracy rate and error rate.

Ans Cross validation is used to find the best k value depending on accuracy or number of ~~miscalculation~~ misclassification. 5 were chosen.

Q For diabetes dataset.
What is the purpose of feature scaling? How to perform it?

Ans Feature scaling is used in KNN to have the same scale because if we use a different scale, the features with larger scale get more weightage so the ~~following~~ lower scale features will get supressed and won't have an impact on the model. To perform scaling, first the frequency distributions of all features were checked, the one having normal distributions were scaled using standard scaler and the rest using minmax scaler.

Code

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

import seaborn as sns

import matplotlib.pyplot as plt

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

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

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

```python
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

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

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

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

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

```
knn = KNeighborsClassifier(n_neighbors=5)

knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy}")

cm = confusion_matrix(y_test, y_pred)

print("Confusion Matrix:")

print(cm)

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

plt.title('Confusion Matrix')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()


print("Classification Report:")

print(classification_report(y_test, y_pred))


import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler
```

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

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

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

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

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

cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)

sns.heatmap(cm, annot=True, fmt="d")
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

print("Classification Report:")
print(classification_report(y_test, y_pred))

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import classification_report, confusion_matrix
```

```
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(8, 6))

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

plt.title("Confusion Matrix")

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.show()

print(classification_report(y_test, y_pred))


# prompt: For Iris dataset

# How to choose the k value? Demonstrate using accuracy rate and error

# rate. Give theory


import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler


# Load the Iris dataset

try:
```

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

# Prepare the data
X = data.drop('species', axis=1)
y = data['species']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Scale the data (important for KNN)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Find the optimal k value
error_rates = []
for k in range(1, 31):  # Test k values from 1 to 30
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    error_rates.append(1 - accuracy_score(y_test, y_pred)) # Error rate = 1 - accuracy

# Plot error rates
```

```
plt.figure(figsize=(10, 6))
plt.plot(range(1, 31), error_rates, color='blue', linestyle='dashed', marker='o',
         markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
plt.show()



# Theory for choosing k:
# The optimal 'k' value minimizes the error rate.
# Very small k (e.g., 1) can lead to overfitting, being too sensitive to noise.
# Very large k (e.g., 30) can lead to underfitting, smoothing out the decision boundaries too much.
# We seek a k that balances these extremes, as shown by the error rate plot.

#Select k based on the minimum error rate observed in the plot
best_k = error_rates.index(min(error_rates)) + 1  #Add 1 as the index starts from 0



# Train and evaluate the model with the best k
knn = KNeighborsClassifier(n_neighbors=best_k)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
```

```
# Evaluate the model
print("Accuracy Score:", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix:")
cm = confusion_matrix(y_test, y_pred)
print(cm)
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

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

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

# Load data
df = pd.read_csv('/content/iris (1).csv')
```

```
X = df.iloc[:, :-1]
y = df.iloc[:, -1]

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)

# Store accuracy and error rate
accuracy = []
error_rate = []

# Try k from 1 to 20
for k in range(1, 21):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    preds = knn.predict(X_test)
    acc = accuracy_score(y_test, preds)
    accuracy.append(acc)
    error_rate.append(1 - acc)

# Plot
plt.figure(figsize=(10,5))
plt.plot(range(1, 21), accuracy, label='Accuracy')
plt.plot(range(1, 21), error_rate, label='Error Rate')
plt.xlabel('K Value')
```

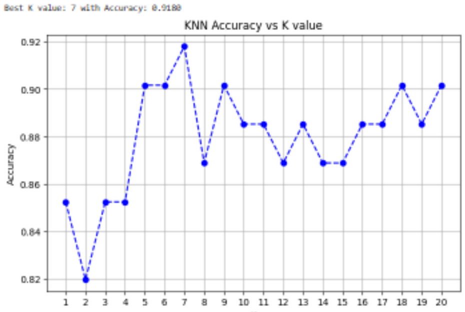
```
plt.ylabel('Rate')

plt.title('K vs Accuracy and Error Rate')

plt.legend()

plt.show()


import pandas as pd

from sklearn.preprocessing import StandardScaler


# Load data

df = pd.read_csv('/content/diabetes.csv')

X = df.drop('Outcome', axis=1)  # Features

y = df['Outcome']               # Target


# Perform scaling

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)


# Convert back to DataFrame (optional)

X_scaled_df = pd.DataFrame(X_scaled, columns=X.columns)
```

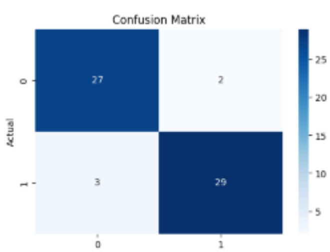
Output :

```
IRIS Dataset - Actual vs Predicted:

        Actual   Predicted
0   versicolor  versicolor
1       setosa      setosa
2    virginica   virginica
3   versicolor  versicolor
4   versicolor  versicolor
5       setosa      setosa
6   versicolor  versicolor
7    virginica   virginica
8   versicolor  versicolor
9   versicolor  versicolor
10   virginica   virginica
11      setosa      setosa
12      setosa      setosa
13      setosa      setosa
14      setosa      setosa
15  versicolor  versicolor
16   virginica   virginica
17  versicolor  versicolor
18  versicolor  versicolor
19   virginica   virginica
20      setosa      setosa
21   virginica   virginica
22      setosa      setosa
23   virginica   virginica
24   virginica   virginica
25   virginica   virginica
26   virginica   virginica
27   virginica   virginica
28      setosa      setosa
29      setosa      setosa

Accuracy Score: 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 - Actual vs Predicted:

     Actual  Predicted
0         0          0
1         0          0
2         0          0
3         0          0
4         0          1
..      ...        ...
149       1          1
150       0          0
151       0          0
152       1          0
153       0          0

[154 rows x 2 columns]

Accuracy Score: 0.6948051948051948

Confusion Matrix:
 [[79 20]
 [27 28]]
```

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





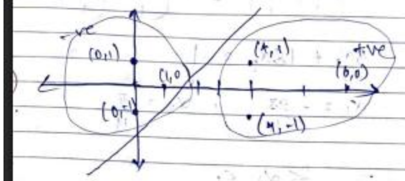

```
Classification Report:

              precision    recall  f1-score   support

           0       0.90      0.93      0.92        29
           1       0.94      0.91      0.92        32

    accuracy                           0.92        61
   macro avg       0.92      0.92      0.92        61
weighted avg       0.92      0.92      0.92        61
```

PROGRAM 7 Build Support vector machine model for a given dataset

Screenshot

Lab-6 - Support Vector Machine

Points (4,1), (4,-1) and (6,0) belong to positive class and points (1,0), (0,1) and (0,-1) belong to negative class. Draw an optimal hyperplane.



S1 $\begin{pmatrix}1\\0\end{pmatrix}$ S2 $\begin{pmatrix}4\\1\end{pmatrix}$

$\alpha_1 \Rightarrow \underset{S_1}{\begin{pmatrix}1\\0\\1\end{pmatrix}} \quad \underset{S_2}{\begin{pmatrix}4\\1\\1\end{pmatrix}} \Leftarrow \alpha_2$

$\cancel{\alpha_1 S_1 \cdot S_1 + \alpha_2 S_2 \cdot S_2 = -1}$

$\cancel{\alpha_1 \begin{pmatrix}1\\0\end{pmatrix}\cdot\begin{pmatrix}1\\0\end{pmatrix} + \alpha_2 \begin{pmatrix}4\\1\end{pmatrix}\cdot\begin{pmatrix}1\\0\end{pmatrix} = -1}$

$\alpha_1 \begin{pmatrix}1\\0\\1\end{pmatrix}\begin{pmatrix}1\\0\\1\end{pmatrix} + \alpha_2 \begin{pmatrix}4\\1\\1\end{pmatrix}\begin{pmatrix}1\\0\\1\end{pmatrix} =$

$2\alpha_1 + 5\alpha_2 = -1$

$\alpha_1 S_1 \cdot S_2 + \alpha_2 S_2 \cdot S_2 = 1$

$\alpha_1 \begin{pmatrix}1\\0\\1\end{pmatrix}\begin{pmatrix}4\\1\\1\end{pmatrix} + \alpha_2 \begin{pmatrix}4\\1\\1\end{pmatrix}\begin{pmatrix}4\\1\\1\end{pmatrix} = 1$

$5\alpha_1 + 18\alpha_2 = 1$

$\alpha_1 = -\frac{23}{11}, \quad \alpha_2 = \frac{7}{11}$

$\bar{w} = \sum_{i=1}^{n} \alpha_i \cdot S_i$

$= \alpha_1 S_1 + \alpha_2 S_2$

$= -\frac{23}{11}\begin{pmatrix}1\\0\\1\end{pmatrix} + \frac{7}{11}\begin{pmatrix}4\\1\\1\end{pmatrix}$

$= \begin{pmatrix}5/11\\7/11\\-16/11\end{pmatrix}$ $\frac{5}{11}, \frac{7}{11} \rightarrow$ orientation line

$b = -\frac{16}{11}$ $\boxed{b + \frac{16}{11} = 0}$

$m = \tan\theta$

$\theta = \tan^{-1}(m)$

~~orientation~~

orientation line $= \tan^{-1}\left(\frac{5}{7}\right)$

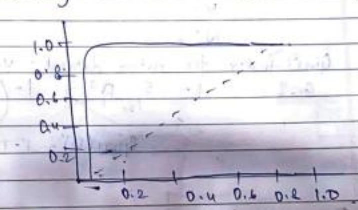
compare accuracy of kernel. both linear RBF kernel gave 100% accuracy on the test set.

Which one preferred & why?

Both preferred equally. The IRIS dataset is relatively simple & linearly separated that is why the linear kernel performs just as well as RBF.

Letter Recognition Dataset

Accuracy : 0.9055 AUC : 1.00



Code

```
import numpy as np

import matplotlib.pyplot as plt



positive_class = np.array([[4, 1], [4, -1], [6, 0]])

negative_class = np.array([[1, 0], [0, 1], [0, -1]])



plt.figure(figsize=(8, 6))

plt.scatter(positive_class[:, 0], positive_class[:, 1], color='red', label='Positive Class', s=100,
edgecolors='black')

plt.scatter(negative_class[:, 0], negative_class[:, 1], color='blue', label='Negative Class', s=100,
edgecolors='black')



all_points = np.concatenate([positive_class, negative_class])

labels = ["(4,1)", "(4,-1)", "(6,0)", "(1,0)", "(0,1)", "(0,-1)"]



for i, txt in enumerate(labels):

    plt.annotate(txt, (all_points[i][0], all_points[i][1]), textcoords="offset points", xytext=(0,5),
ha='center', fontsize=10)



x_values = np.linspace(-1, 7, 100)

y_values = np.zeros_like(x_values)



plt.plot(x_values, y_values, color='black', linestyle='--', label='Optimal Hyperplane (y = 0)')
```

```
plt.plot(x_values, y_values + 1, color='gray', linestyle=':', label='Margin at y = 1')
plt.plot(x_values, y_values - 1, color='gray', linestyle=':', label='Margin at y = -1')

plt.title('Optimal Hyperplane for SVM (Visual Approximation)', fontsize=14)
plt.xlabel('x1')
plt.ylabel('x2')
plt.xlim(-1, 7)
plt.ylim(-2, 2)
plt.axhline(0, color='black',linewidth=0.5)
plt.axvline(0, color='black',linewidth=0.5)
plt.legend()

plt.grid(True)
plt.show()
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

data = pd.read_csv('/content/iris (1) (1).csv')
```

```
X = data.drop('species', axis=1)

y = data['species']

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


svm_rbf = SVC(kernel='rbf')

svm_rbf.fit(X_train, y_train)

y_pred_rbf = svm_rbf.predict(X_test)

accuracy_rbf = accuracy_score(y_test, y_pred_rbf)

cm_rbf = confusion_matrix(y_test, y_pred_rbf)


print("SVM with RBF Kernel:")

print("Accuracy:", accuracy_rbf)

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


plt.figure(figsize=(6, 4))

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

            xticklabels=data['species'].unique(),

            yticklabels=data['species'].unique())

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix (RBF Kernel)')

plt.show()


svm_linear = SVC(kernel='linear')
```

```
svm_linear.fit(X_train, y_train)

y_pred_linear = svm_linear.predict(X_test)

accuracy_linear = accuracy_score(y_test, y_pred_linear)

cm_linear = confusion_matrix(y_test, y_pred_linear)


print("\nSVM with Linear Kernel:")

print("Accuracy:", accuracy_linear)

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


plt.figure(figsize=(6, 4))

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

            xticklabels=data['species'].unique(),

            yticklabels=data['species'].unique())

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix (Linear Kernel)')

plt.show()

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.svm import SVC

from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve, auc

import seaborn as sns
```

```
from sklearn.preprocessing import label_binarize
from sklearn.multiclass import OneVsRestClassifier


data = pd.read_csv('/content/letter-recognition.csv')  # Replace with the correct path if necessary


X = data.drop('letter', axis=1)
y = data['letter']


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


svm_classifier = SVC(kernel='rbf', probability=True) # probability=True is needed for ROC curve
svm_classifier.fit(X_train, y_train)


y_pred = svm_classifier.predict(X_test)


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


print("SVM Classifier:")
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", cm)


plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=np.unique(y),
yticklabels=np.unique(y))
```

```
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

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

classifier = OneVsRestClassifier(SVC(kernel='rbf', probability=True))
classifier.fit(X_train, y_train)
y_score = classifier.predict_proba(X_test)

fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_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='micro-average ROC curve (area = {0:0.2f})'
```

".format(roc_auc["micro"]))

plt.plot([0, 1], [0, 1], 'k--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Micro-averaged ROC Curve')

plt.legend(loc="lower right")

plt.show()

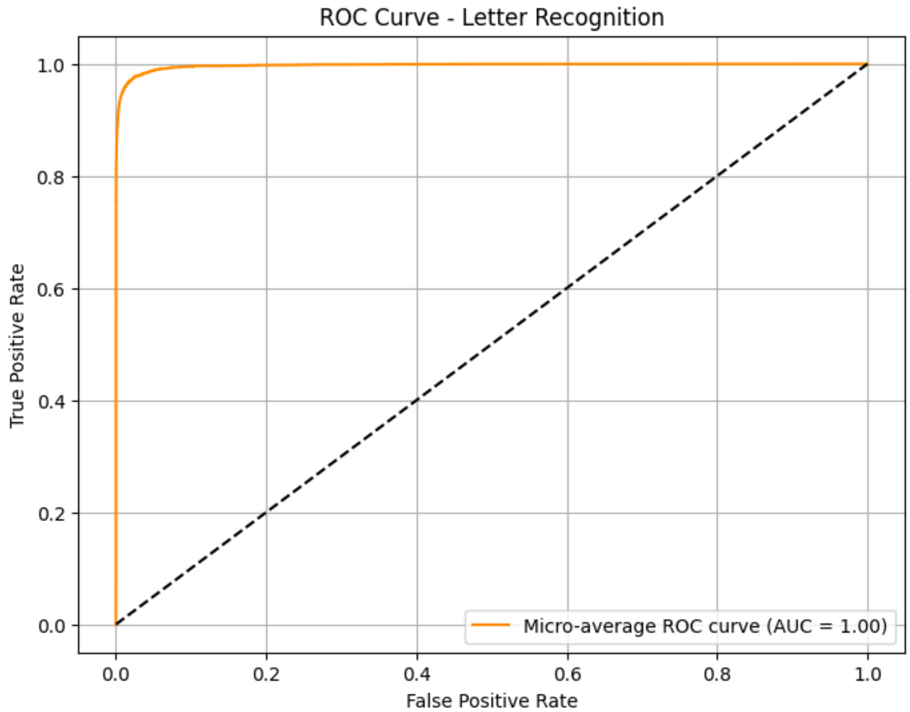
print(f"Micro-averaged AUC: {roc_auc['micro']}")

Output :

```
---- Letter Recognition Dataset ----
Accuracy: 0.95025
Confusion Matrix:
[[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
```

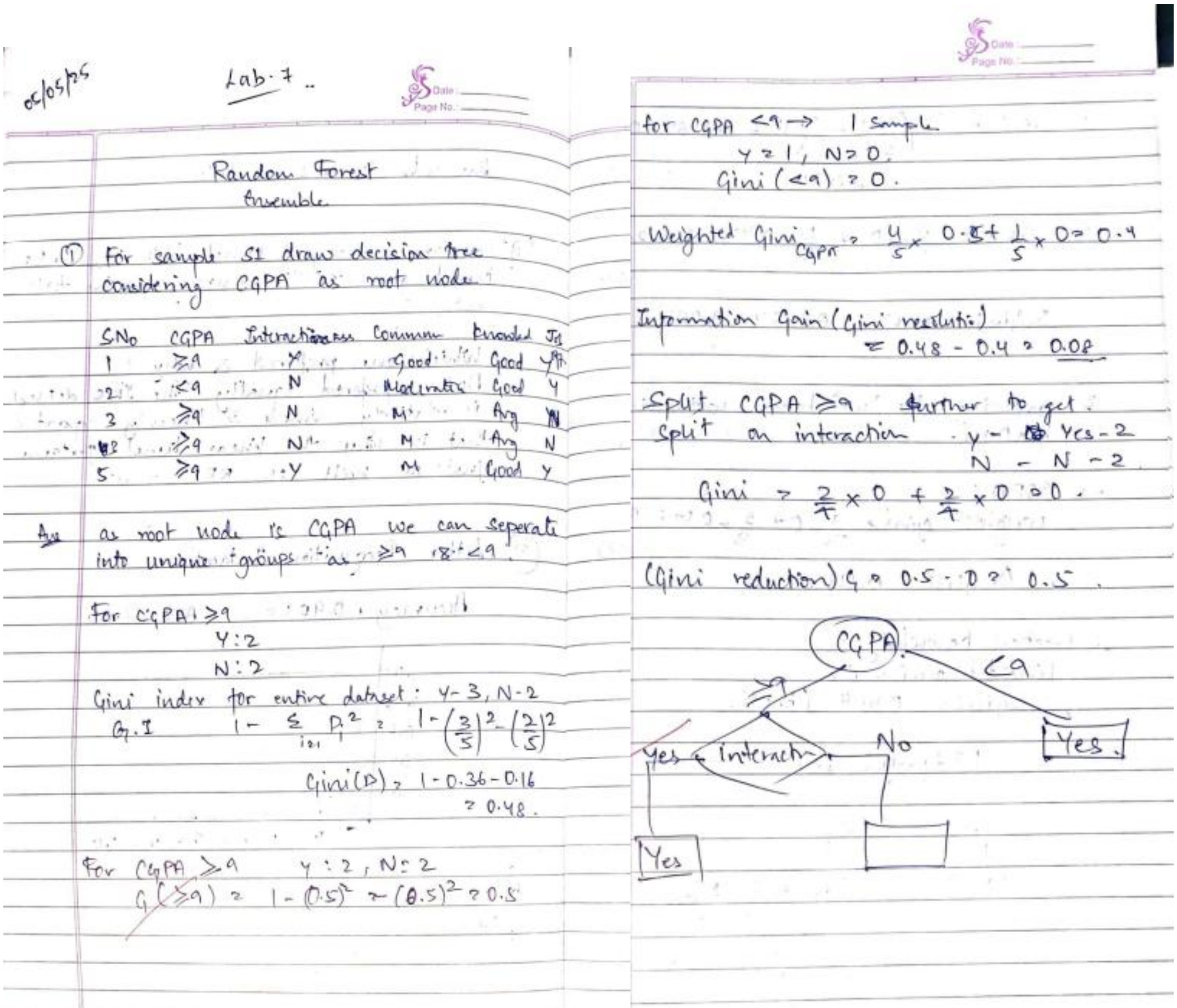
```
---- IRIS Dataset ----

Linear Kernel:
Accuracy: 1.0
Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]

RBF Kernel:
Accuracy: 1.0
Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

```
[   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]]
```



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

Screenshot



06/05/25 Lab·7

Random Forest Ensemble

① For sample S1 draw decision tree considering CGPA as root node

SNo	CGPA	Interactiveness	Commun	Knowled	Job
1	≥9	Y	Good	Good	Y
2	<9	N	Moderate	Good	Y
3	≥9	N	M	Avg	N
4	≥9	N	M	Avg	N
5	≥9	Y	M	Good	Y

Ans as root node is CGPA we can seperate into unique groups as ≥9 & <9

For CGPA ≥9
Y : 2
N : 2

Gini index for entire dataset : Y-3, N-2

G.I $1 - \sum_{i=1} P_i^2 = 1 - \left(\frac{3}{5}\right)^2 - \left(\frac{2}{5}\right)^2$

$Gini(D) = 1 - 0.36 - 0.16 = 0.48$.

For CGPA ≥9 Y : 2, N : 2
$G(\geq 9) = 1 - (0.5)^2 - (0.5)^2 = 0.5$

for CGPA <9 → 1 sample
Y = 1, N = 0.
Gini (<9) = 0.

Weighted $Gini_{CGPA} = \frac{4}{5} \times 0.5 + \frac{1}{5} \times 0 = 0.4$

Information Gain (Gini reduction)
$= 0.48 - 0.4 = 0.08$

Split CGPA ≥9 further to get split on interaction Y - Yes - 2, N - N - 2

$Gini = \frac{2}{4} \times 0 + \frac{2}{4} \times 0 = 0$.

(Gini reduction) G = 0.5 - 0 = 0.5.

CGPA
≥9
<9
Yes
Interaction
No
Yes
Yes

1. Gini = $1 - \left(\frac{4}{5}\right)^2 - \left(\frac{1}{5}\right)^2 = 1 - 0.64 - 0.04$
$= 0.32$.

2. Gini = $\frac{2}{5} \times 0 + \frac{3}{5} \times 0.444 = 0.2664$.

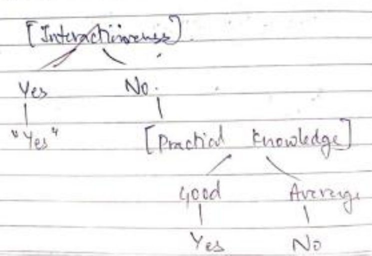
Δ Gini = $0.32 - 0.2664 = 0.0536$

3. growing the "no" beach with other features.

→ CGPA:
weighted gini = $\frac{1}{3} \times 0 + \frac{2}{3} \times 0.50 = 0.333$

Δ Gini = 0.111.

→ Practical knowledge:
Weighted gini = 0.
ΔGini = 0.444 (best).



RF	n-estimates	mean accuracy.
1	10	0.9667
2	50	0.9667
3	100	0.9667
4	150	0.9667
5	200	0.9667.

Accuracy = 1.0
Best number of tree = 20

confusion matrix = $\begin{bmatrix} 10 & 0 & 0 \\ 0 & 9 & 0 \\ 0 & 0 & 11 \end{bmatrix}$.

Code

```
import pandas as pd

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 dataset

df = pd.read_csv('/content/iris (1).csv')


# Prepare features and target

X = df.drop(columns=['species'])  # Assuming 'species' is the target column
```

```
y = df['species']


# Split 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)


# Build Random Forest with default 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)


# Measure accuracy
default_score = accuracy_score(y_test, y_pred_default)
print(f"Default RF accuracy (n_estimators=10): {default_score:.4f}")


# Fine-tune the number of trees
scores = []
n_range = range(1, 101)


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

Find the best score and number of trees

best_score = max(scores)

best_n = n_range[scores.index(best_score)]

print(f"Best RF accuracy: {best_score:.4f} with n_estimators={best_n}")

Optional: Plot accuracy vs number of estimators

plt.figure(figsize=(10, 6))

plt.plot(n_range, scores, marker='o')

plt.title('Random Forest Accuracy vs Number of Trees')

plt.xlabel('Number of Trees (n_estimators)')

plt.ylabel('Accuracy')

plt.grid(True)

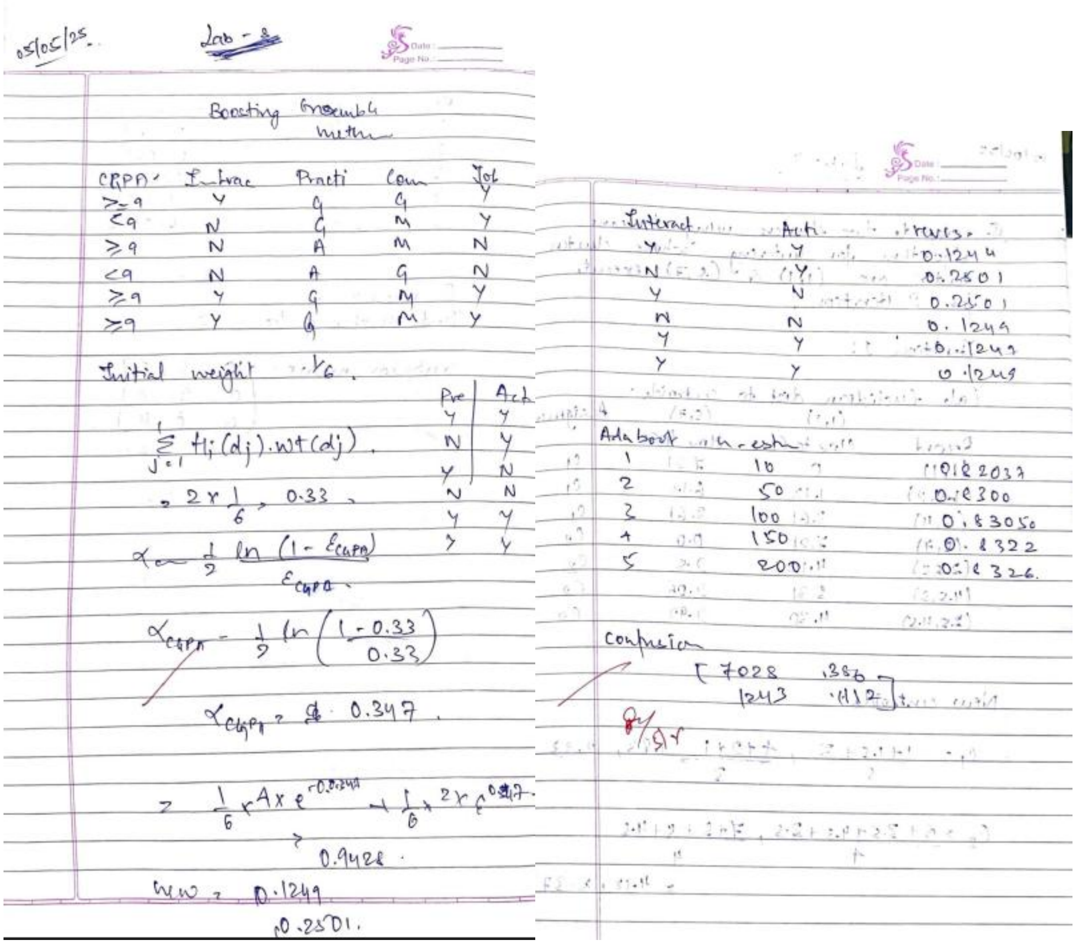
plt.show()

Output :

```
Default n_estimators=10 → Mean CV accuracy: 0.9667

Random Forest Tuning Results    n_estimators  mean_accuracy
0           10       0.966667
1           50       0.966667
2          100       0.966667
3          150       0.966667
4          200       0.966667
Best number of trees: 10 → Mean CV accuracy: 0.9667
```

PROGRAM 9 Implement Boosting ensemble method on a given dataset.

Screenshot


05/05/25 Lab - 8

Boosting Ensemble method

CGPA	Interac	Practi	Com	Job
≥9	Y	G	G	Y
<9	N	G	M	Y
≥9	N	A	M	N
<9	N	A	G	N
≥9	Y	G	M	Y
≥9	Y	G	M	Y

Initial weight = 1/6

Pre	Act
Y	Y
N	Y
Y	N
N	N
Y	Y
Y	Y

$\sum_{j=1} t_i(d_j) \cdot wt(d_j)$

$= 2 \times \frac{1}{6} = 0.33$

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

$\alpha_{CGPA} = \frac{1}{2} \ln \left(\frac{1 - 0.33}{0.33}\right)$

$\alpha_{CGPA} = 0.347$

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

$= 0.9428$

NW = 0.1249

0.2501

Interact	Acti	Weights
Y	Y	0.1249
N	Y	0.2501
Y	N	0.2501
N	N	0.1249
Y	Y	0.1249
Y	Y	0.1249

Adaboost with estimator

1	10	0.82037
2	50	0.8300
3	100	0.83050
4	150	0.8322
5	200	0.8326

Confusion

[7028 1386
1243 1412]

8/8

```python
# Load dataset

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

# Drop rows with missing values

df.dropna(inplace=True)

# Encode categorical columns

label_encoders = {}

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

    le = LabelEncoder()

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

    label_encoders[column] = le

# Separate features and target

X = df.drop(columns=['income_level'], errors='ignore', axis=1)

y = df['income_level']

# Split 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)

# AdaBoost with 10 estimators

model_10 = AdaBoostClassifier(n_estimators=10, random_state=42)

model_10.fit(X_train, y_train)

y_pred_10 = model_10.predict(X_test)

score_10 = accuracy_score(y_test, y_pred_10)

print(f"Accuracy with 10 estimators: {score_10:.4f}")

# Fine-tune number of estimators

best_score = 0

best_n = 0
```

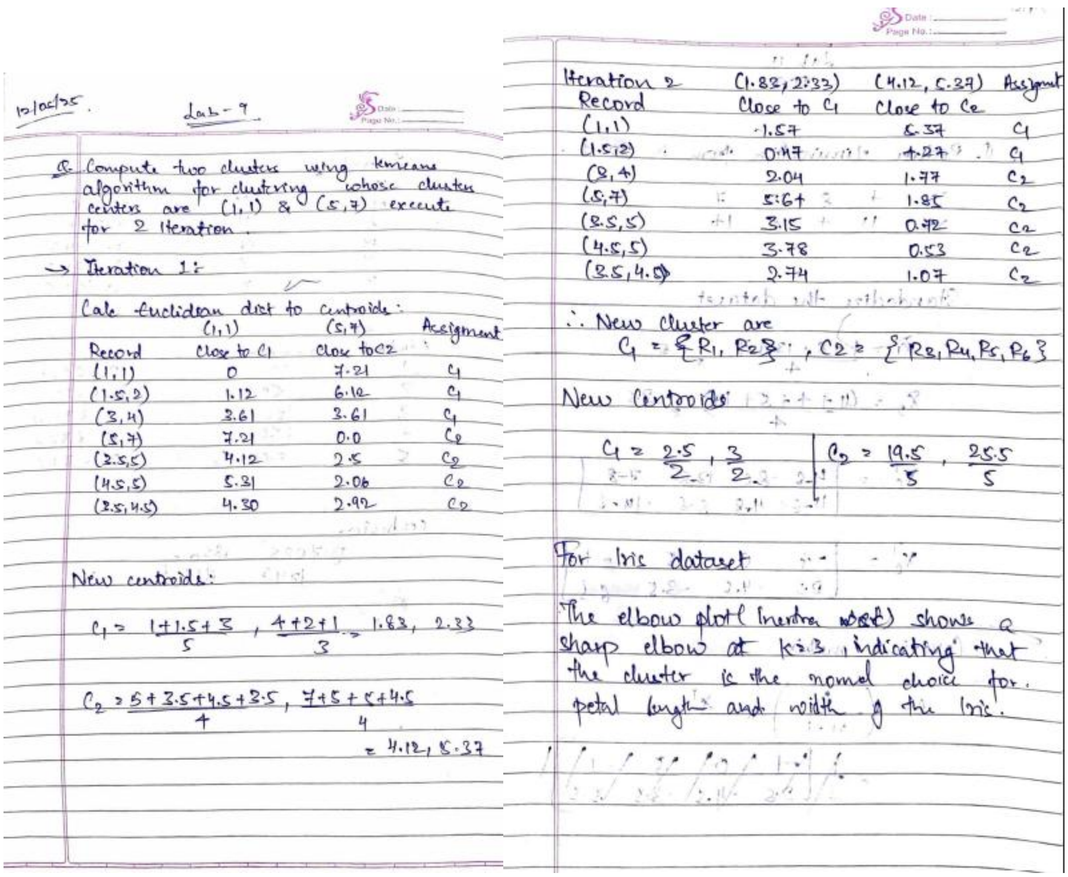
```
estimators_range = list(range(10, 201, 10))
scores = []
for n in estimators_range:
    model = AdaBoostClassifier(n_estimators=n, random_state=42)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    score = accuracy_score(y_test, y_pred)
    scores.append(score)
    print(f"n_estimators={n}, Accuracy={score:.4f}")
    if score > best_score:
        best_score = score
        best_n = n
print(f"\nBest Accuracy: {best_score:.4f} using {best_n} estimators")
# Plot accuracy vs number of estimators
plt.figure(figsize=(7, 4))
plt.plot(estimators_range, scores, marker='o', linestyle='-', color='blue')
plt.title("Accuracy vs Number of Estimators (AdaBoost)")
plt.xlabel("Number of Estimators (Trees)")
plt.ylabel("Accuracy")
plt.grid(True)
plt.xticks(estimators_range)
plt.tight_layout()
plt.show()
```

Output :

```
Default AdaBoost (n_estimators=10) ⇒ Mean 5-fold CV accuracy: 0.8202

Tuning results:
   n_estimators  mean_accuracy
0            10       0.820237
1            50       0.830023
2           100       0.832050
3           150       0.832296
4           200       0.832624

Best performance: 0.8326 accuracy using 200 trees
```

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

Screenshot


10/06/25 Lab - 9

Q. Compute two clusters using kmeans algorithm for clustering whose clusters centers are (1,1) & (5,7) execute for 2 Iteration.

→ Iteration 1:

Calc Euclidean dist to centroids:

Record	(1,1) Close to C1	(5,7) Close to C2	Assignment
(1,1)	0	7.21	C1
(1.5,2)	1.12	6.10	C1
(3,4)	3.61	3.61	C1
(5,7)	7.21	0.0	C2
(3.5,5)	4.12	2.5	C2
(4.5,5)	5.31	2.06	C2
(3.5,4.5)	4.30	2.92	C2

New centroids:

$C_1 = \frac{1+1.5+3}{3}, \frac{4+2+1}{3} = 1.83, 2.33$

$C_2 = \frac{5+3.5+4.5+3.5}{4}, \frac{7+5+5+4.5}{4} = 4.12, 5.37$

Iteration 2 Record	(1.83, 2.33) Close to C1	(4.12, 5.37) Close to C2	Assignment
(1,1)	1.57	5.37	C1
(1.5,2)	0.47	4.27	C1
(3,4)	2.04	1.77	C2
(5,7)	5.64	1.85	C2
(3.5,5)	3.15	0.72	C2
(4.5,5)	3.78	0.53	C2
(3.5,4.5)	2.74	1.07	C2

∴ New Cluster are

$C_1 = \{R_1, R_2\}$, $C_2 = \{R_3, R_4, R_5, R_6\}$

New Centroids

$C_1 = \frac{2.5}{2}, \frac{3}{2}$ | $C_2 = \frac{19.5}{5}, \frac{25.5}{5}$

For Iris dataset

The elbow plot (Inertia vs k) shows a sharp elbow at k=3, indicating that the cluster is the normal choice for petal length and width of the Iris.

Code

import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

```
from sklearn.cluster import KMeans
from sklearn.metrics import accuracy_score
from scipy.stats import mode
import matplotlib.pyplot as plt

# Step 1: Generate sample data and save to CSV
np.random.seed(42)
names = [f"Person_{i}" for i in range(50)]
ages = np.random.randint(20, 60, 50)
income = np.random.randint(30000, 120000, 50)

df = pd.DataFrame({'Name': names, 'Age': ages, 'Income': income})
df.to_csv("income.csv", index=False)

# Step 2: Load the data
data = pd.read_csv("income.csv")

# Drop 'Name' and extract features
X = data[['Age', 'Income']]

# Step 3: Split the data
X_train, X_test = train_test_split(X, test_size=0.2, random_state=42)

# Step 4: Perform scaling
```

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Step 5: Plot SSE vs number of clusters (Elbow method)
sse = []
k_range = range(1, 11)
for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_train_scaled)
    sse.append(kmeans.inertia_)

plt.figure(figsize=(8, 4))
plt.plot(k_range, sse, marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('SSE (Inertia)')
plt.title('Elbow Method For Optimal k')
plt.grid(True)
plt.show()

# Step 6: Choose optimal number of clusters (say 3) and fit model
optimal_k = 3
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
kmeans.fit(X_train_scaled)
```

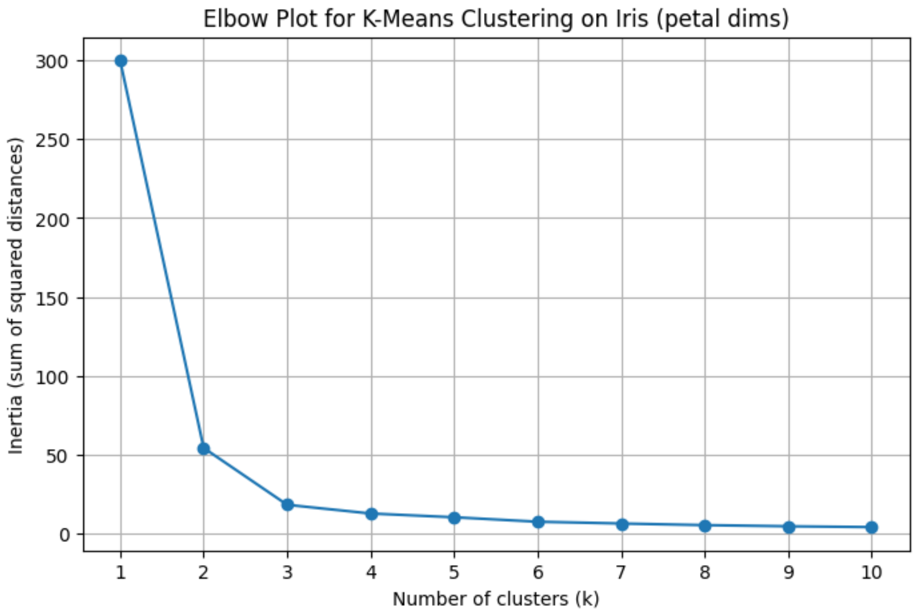
```
# Predict on test data
predictions = kmeans.predict(X_test_scaled)

# Note: There's no ground truth labels, but for demonstration,
# we can try assigning true clusters (via KMeans on full data)
# and see if predicted clusters align

# Fit on full data to assign pseudo-labels
full_kmeans = KMeans(n_clusters=optimal_k, random_state=42)
true_clusters = full_kmeans.fit_predict(scaler.fit_transform(X))

# Align predicted clusters using majority voting (only for demonstration)
# Match predicted labels to closest true labels
def map_clusters(true_labels, pred_labels):
    labels = np.zeros_like(pred_labels)
    for i in range(optimal_k):
        mask = (pred_labels == i)
        if np.sum(mask) == 0:
            continue
        labels[mask] = mode(true_labels[mask])[0]
    return labels

mapped_preds = map_clusters(true_clusters[X_test.index], predictions)
```

```
accuracy = accuracy_score(true_clusters[X_test.index], mapped_preds)
print(f"Approximate Clustering Accuracy: {accuracy:.2f}")

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score

# Step 1: Load Iris dataset
iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target

# Keep only petal length and petal width
X = df[['petal length (cm)', 'petal width (cm)']].values

# Step 2: Check impact of scaling
# Try without scaling
sse_unscaled = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
```

```
    kmeans.fit(X)
    sse_unscaled.append(kmeans.inertia_)

# Now scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

sse_scaled = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    sse_scaled.append(kmeans.inertia_)

# Step 3: Plot Elbow Comparison (Scaled vs Unscaled)
plt.figure(figsize=(10, 5))

plt.plot(range(1, 11), sse_unscaled, marker='o', label='Unscaled')
plt.plot(range(1, 11), sse_scaled, marker='s', label='Scaled')
plt.title('Elbow Method (Petal Features Only)')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('SSE (Inertia)')
plt.legend()
plt.grid(True)
plt.show()
```

Output :

```
k=1, inertia=300.00
k=2, inertia=54.15
k=3, inertia=18.05
k=4, inertia=12.52
k=5, inertia=10.14
k=6, inertia=7.31
k=7, inertia=6.19
k=8, inertia=5.16
k=9, inertia=4.41
k=10, inertia=3.90
```



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

Screenshot

13/02/25 Lab 11

Lab 11

PCA

Q. Reduce elimination from 2 to 1.

x_1	4	8	13	7
x_2	11	4	5	14

Standardize the dataset

$\bar{x}_1 = \frac{4+8+13+7}{4} = 8$

$\bar{x}_2 = \frac{(11+4+5+14)}{4} = 8.5$

$$X_c^T = \begin{bmatrix} 4-8 & 8-8 & 13-8 & 7-8 \\ 11-8.5 & 4-8.5 & 5-8.5 & 14-8.5 \end{bmatrix}$$

$$X_c^T = \begin{bmatrix} -4 & 0 & 5 & -1 \\ 2.5 & -4.5 & -3.5 & 5.5 \end{bmatrix}$$

Covariance matrix

$$C = \left(\frac{1}{n-1}\right) \times X^T$$

$$= \frac{1}{3}\begin{bmatrix} -4 & 0 & 5 & -1 \\ 2.5 & -4.5 & -3.5 & 5.5 \end{bmatrix}$$

$$= \frac{1}{3}\begin{bmatrix} -4 & 0 & 6 & -1 \\ 2.5 & -4.5 & -3.5 & 5.5 \end{bmatrix}\begin{bmatrix} -4 & 2.5 \\ 0 & -4.5 \\ 5 & -3.5 \\ -1 & 5.5 \end{bmatrix}$$

$$C = \frac{1}{3}\begin{bmatrix} 42 & -33 \\ -33 & 69 \end{bmatrix}$$

$$C = \begin{bmatrix} 14 & -11 \\ -11 & 23 \end{bmatrix}$$

Let $(C - \lambda I) = 0$

$$\begin{vmatrix} 14-\lambda & -11 \\ -11 & 23-\lambda \end{vmatrix} = 0$$

$\lambda^2 - 37\lambda + 201 = 0$

$\lambda = 30.3849$

Consider larger λ

$$\begin{bmatrix} 14-30.3849 & -11 \\ -11 & 23-30.3849 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} -16.3849x - 11y \\ -11x - 7.3849y \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$-16.3849x - 11y = 0$$
$$x = -0.6713\,y$$

taking $y = 1$

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -0.6713 \\ 1 \end{bmatrix}$$

Normalize

$$x = \frac{x}{\sqrt{x^2+y^2}}, \quad y = \frac{y}{\sqrt{x^2+y^2}}$$

$$C_1 = \begin{bmatrix} -0.5574 \\ 0.8303 \end{bmatrix}$$

Calculate principle component.

$$Z = X_c C_1 = \begin{bmatrix} -4 & 2.5 \\ 0 & -4.5 \\ 5 & -3.5 \\ -1 & 5.5 \end{bmatrix} \begin{bmatrix} -0.5574 \\ 0.8303 \end{bmatrix}$$

$$Z = \begin{bmatrix} 4.305 \\ -3.736 \\ -5.693 \\ 5.124 \end{bmatrix}$$

Accuracy before PCA

logistic regression : 0.9016
SVM : 0.8525
Random forest : 0.8361

Accuracy After PCA
logistic : 0.8689
SVM : 0.8689
Random forest : 0.88

Code


```
import pandas as pd

from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardScaler

from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline

from sklearn.model_selection import train_test_split

from sklearn.svm import SVC

from sklearn.linear_model import LogisticRegression
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score

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

# 2. Label-encode binary text columns
le = LabelEncoder()
for col in ["Sex", "ExerciseAngina"]:
    df[col] = le.fit_transform(df[col])

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

# 4. Build preprocessing pipeline:
#    - One-hot for multi-category columns (using sparse_output=False)
#    - passthrough the rest
#    - then scale everything
cat_cols = ["ChestPainType", "RestingECG", "ST_Slope"]
preprocessor = Pipeline([
    ("onehot", ColumnTransformer([
        ("ohe", OneHotEncoder(sparse_output=False, drop="first"), cat_cols)
```

```
    ], remainder="passthrough")),
    ("scaler", StandardScaler())
])

# 5. Apply preprocessing
X_proc = preprocessor.fit_transform(X)

# 6. Train/test split
X_train, X_test, y_train, y_test = train_test_split(
    X_proc, y, test_size=0.2, random_state=42
)

# 7. Define models
models = {
    "SVM": SVC(random_state=42),
    "LogisticRegression": LogisticRegression(max_iter=1000, random_state=42),
    "RandomForest": RandomForestClassifier(random_state=42)
}

# 8. Train & evaluate before PCA
print("=== Accuracies BEFORE PCA ===")
scores_before = {}
for name, clf in models.items():
    clf.fit(X_train, y_train)
```

```
    preds = clf.predict(X_test)

    acc = accuracy_score(y_test, preds)

    scores_before[name] = acc

    print(f"{name:17s}: {acc:.4f}")


# 9. Apply PCA (retain 95% variance)

pca = PCA(n_components=0.95, random_state=42)

X_train_pca = pca.fit_transform(X_train)

X_test_pca  = pca.transform(X_test)

print(f"\nPCA retained {pca.n_components_} components, "

      f"explained variance = {pca.explained_variance_ratio_.sum():.4f}\n")

# 10. Train & evaluate after PCA

print("=== Accuracies AFTER PCA ===")

scores_after = {}

for name, clf in models.items():

    clf.fit(X_train_pca, y_train)

    preds = clf.predict(X_test_pca)

    acc = accuracy_score(y_test, preds)

    scores_after[name] = acc

    print(f"{name:17s}: {acc:.4f}")
```

Output :

```
=== Accuracies BEFORE PCA ===
SVM              : 0.8750
LogisticRegression: 0.8533
RandomForest     : 0.8641

PCA retained 13 components, explained variance = 0.9719

=== Accuracies AFTER PCA ===
SVM              : 0.8750
LogisticRegression: 0.8533
RandomForest     : 0.8641
```