VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

Supriya S (1BM22CS350)

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
Feb-2025 to June-2025

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

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled "Machine Learning (23CS6PCMAL)" carried out by **Supriya S (1BM22CS350),** 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: **Sunayana S** Assistant Professor Department of CSE, BMSCE	**Dr. Kavitha Sooda** Professor & HOD Department of CSE, BMSCE

Index

Sl. No.	Date	Experiment Title	Page No.
1	21-2-2025	Write a python program to import and export data using Pandas library functions	4
2	3-3-2025	Demonstrate various data pre-processing techniques for a given dataset	7
3	10-3-2025	Implement Linear and Multi-Linear Regression algorithm using appropriate dataset	10
4	17-3-2025	Build Logistic Regression Model for a given dataset	15
5	24-3-2025	Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample	20
6	7-4-2025	Build KNN Classification model for a given dataset	22
7	21-4-2025	Build Support vector machine model for a given dataset	25
8	5-5-2025	Implement Random forest ensemble method on a given dataset	28
9	5-5-2025	Implement Boosting ensemble method on a given dataset	32
10	12-5-2025	Build k-Means algorithm to cluster a set of data stored in a .CSV file	36
11	12-5-2025	Implement Dimensionality reduction using Principal Component Analysis (PCA) method	39

Github Link:

https://github.com/SupriyaS26/6THSEMMLLAB

Program 1

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

Screenshot

Lab - 0

3-25

1. To Do Exercise: Four different ways of importing datasets

1) Initializing values directly into DataFrame

```
df = pd.DataFrame({'USN': ['450', '451', '452',
                           '453', '454'],
    'Name': ['Sita', 'Rita', 'Meena', 'Manya',
             'Keerthi'],
    'Marks': [94, 89, 87, 86, 84]
  })
print(df)
```

2) Importing datasets from sklearn.datasets

```
from sklearn.datasets import load_diabetes
diabetes := load_diabetes()
df = pd.DataFrame(diabetesdata, columns=
                  diabetes.feature_names)
df['target'] = diabetes.target
print(df)
```

3) Importing datasets from a specific .csv file

```
file = 'sample_sales_data.csv'
df = pd.read_csv(file)
print(df.head())
```

4) Downloading datasets from existing dataset repositories like Kaggle, UCI, etc

```
file = 'Diabetes.csv'
df = pd.read_csv(file)
print(df.head())
```

2

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

2

```
tickers = ["HDFCBANK.NS", "ICICIBANK.NS",
           "KOTAKBANK.NS"]
data = yf.download(tickers, start="2024-01
                   end="2024-12-30",
                   group_by='ticker')

hdfc_data = data['HDFCBANK.NS']
hdfc_data['Daily Return'] = hdfc_data['Close
                            pct_change()
icici_data = data['ICICIBANK.NS']
icicidata['Daily Return'] = icici_data['Close
                            pct_change()
kotak_data = data['KOTAKBANK.NS']
kotakdata['Daily Return'] = kotak_data
                   ['Close'].pct_change()

plt.figure(figsize=(12,6))
plt.subplot(2,1,1)
hdfcdata['Close'].plot(title="Closing Price")
icicidata['Close'].plot(title="Closing Price")
kotakdata['Close'].plot(title="Closing
                             Price")
plt.subplot(2,1,2)
hdfc_data['Daily Return'].plot(title
                        ="Daily Returns")
icici_data['Daily Return'].plot(title
                        ="Daily Returns")
kotakdata['Daily Return'].plot(title="Daily
                                Returns")
plt.tight_layout()
plt.show()
```

Code:

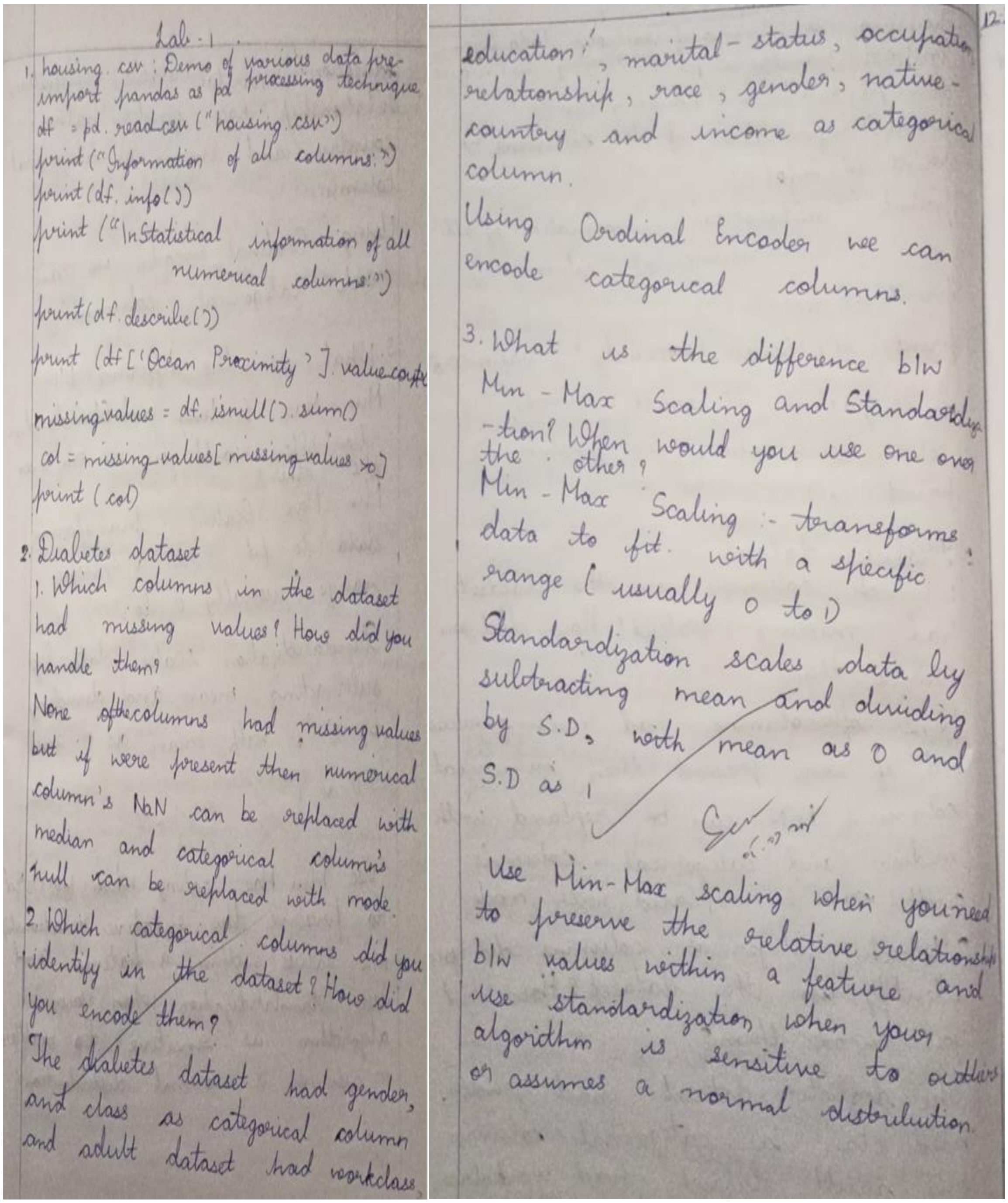
```
#1. Initializing values directly into DataFrame
data= {
'USN’: ['1BM22CS450','1BM22CS451','1BM22CS452','1BM22CS453','1BM22CS454'],
'Name’: ['Sita','Rita','Meena','Manya','Keerthi'],
'Marks’: [94,89,87,86,84]
}
df=pd.DataFrame(data)
print("Data:")
print(df)

#2. Importing datasets from sklearn.datasets
from sklearn.datasets import load_diabetes
db=load_diabetes()
df=pd.DataFrame(db.data)
print(df.head())

#3. Importing datsets from a specific .csv file
file_path='sample_sales_data.csv'
df=pd.read_csv(file_path)
print(df.head())

#4. Downloading datasets from existing dataset repositories like Kaggle,UCI,Mendely,KEEL,ETC
file_path='DatasetofDiabetes .csv'
df=pd.read_csv(file_path)
print(df.head())

import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt
tickers = ["HDFCBANK.NS", "ICICIBANK.NS", "KOTAKBANK.NS"]
# Fetch historical data for the last 1 year
```

```python
data = yf.download(tickers, start="2024-01-01", end="2024-12-30", group_by='ticker')
# Display the first 5 rows of the dataset
print("First 5 rows of the dataset:")
print(data.head())
hdfc_data = data['HDFCBANK.NS']
# Calculate daily returns
hdfc_data['Daily Return'] = hdfc_data['Close'].pct_change()
icici_data = data['ICICIBANK.NS']
# Calculate daily returns
icici_data['Daily Return'] = icici_data['Close'].pct_change()
kotak_data = data['KOTAKBANK.NS']
# Calculate daily returns
kotak_data['Daily Return'] = kotak_data['Close'].pct_change()
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
hdfc_data['Close'].plot(title="Hdfc Bank - Closing Price")
plt.subplot(2, 1, 2)
hdfc_data['Daily Return'].plot(title="Hdfc Bank - Daily Returns", color='orange')
plt.tight_layout()
plt.show()
```

Program 2

Demonstrate various data pre-processing techniques for a given dataset

Screenshot



Lab - 1

1. housing.csv : Demo of various data pre-processing technique

```
import pandas as pd
df = pd.read_csv("housing.csv")
print("Information of all columns:")
print(df.info())
print("\nStatistical information of all
                numerical columns:")
print(df.describe())
print(df['Ocean Proximity'].value_counts
missing_values = df.isnull().sum()
col = missing_values[missing_values > 0]
print(col)
```

2. Diabetes dataset

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

None of the columns had missing values but if were present then numerical column's NaN can be replaced with median and categorical column's null can be replaced with mode.

2. Which categorical columns did you identify in the dataset? How did you encode them?

The diabetes dataset had gender and class as categorical column and adult dataset had workclass, education, marital-status, occupation, relationship, race, gender, native-country and income as categorical column.

Using Ordinal Encoder we can encode categorical columns.

3. What is the difference b/w Min-Max Scaling and Standardization? When would you use one over the other?

Min-Max Scaling :- transforms data to fit with a specific range (usually 0 to 1)

Standardization scales data by subtracting mean and dividing by S.D, with mean as 0 and S.D as 1

Use Min-Max scaling when you need to preserve the relative relationship b/w values within a feature and use standardization when your algorithm is sensitive to outliers or assumes a normal distribution.

Code

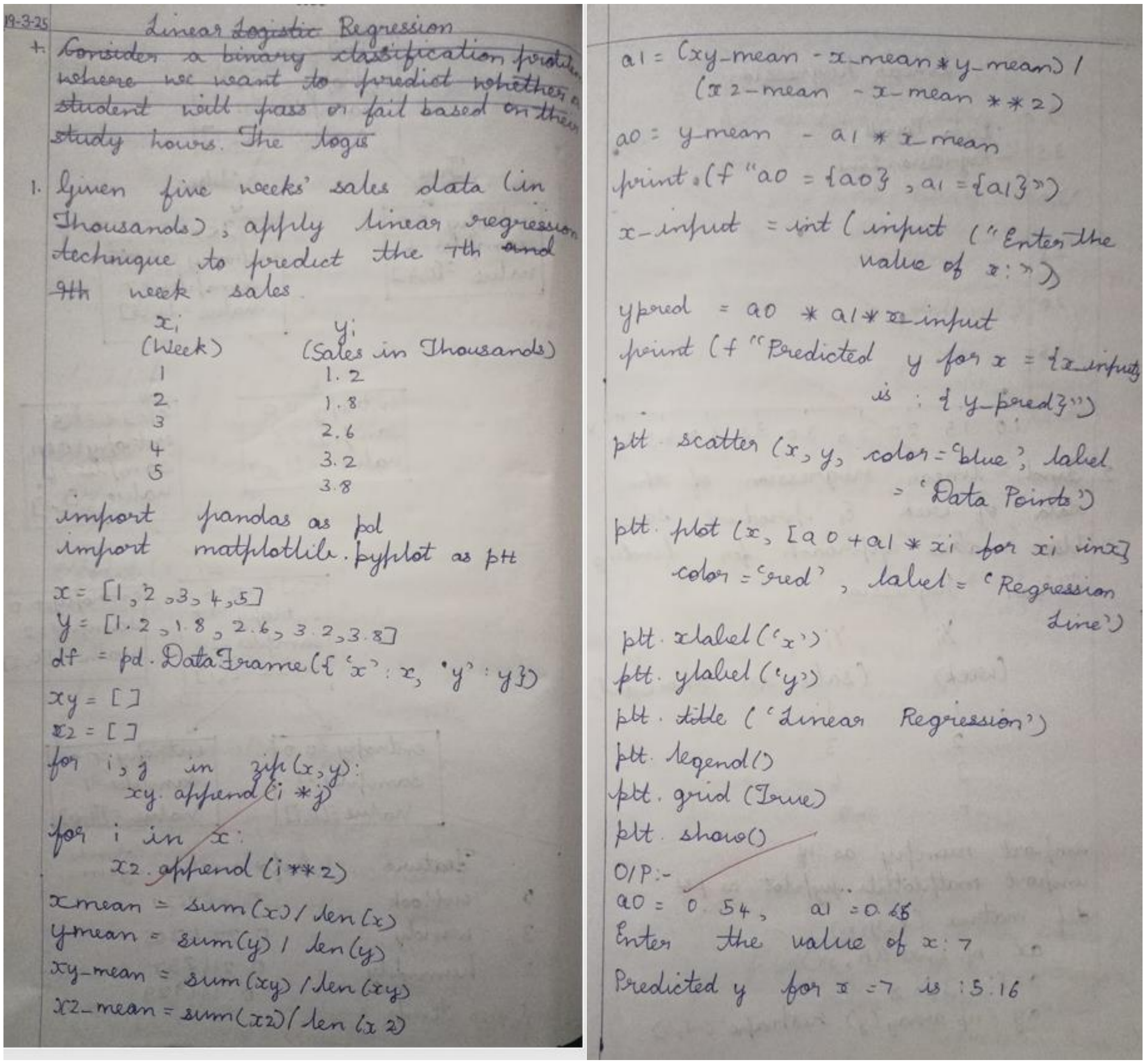
```
#Handling Missing Values, Handling categorical data, Handling Outliers

import pandas as pd

df=pd.read_csv('Dataset of Diabetes .csv')

missing_values=df.isnull().sum()

print(missing_values[missing_values > 0])

# Data Transformations: Min-max Scaler/Normalization , Standard Scaler

import pandas as pd

import numpy as np

from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder, StandardScaler

file_path = "Dataset of Diabetes .csv"

df = pd.read_csv(file_path)

df["Gender"] = df["Gender"].str.upper()

ordinal_encoder = OrdinalEncoder(categories=[["M", "F"]])

df["Gender_Encoded"] = ordinal_encoder.fit_transform(df[["Gender"]])

onehot_encoder = OneHotEncoder(sparse_output=False, drop="first")

encoded_class = onehot_encoder.fit_transform(df[["CLASS"]])

encoded_class_df = pd.DataFrame(encoded_class,
columns=onehot_encoder.get_feature_names_out(["CLASS"]))

df_encoded = pd.concat([df, encoded_class_df], axis=1)

df_encoded.drop(["Gender", "CLASS"], axis=1, inplace=True)

num_cols = ["AGE", "Urea", "Cr", "HbA1c", "Chol", "TG", "HDL", "LDL", "VLDL", "BMI"]

scaler = StandardScaler()

df_encoded[num_cols] = scaler.fit_transform(df_encoded[num_cols])
```

```
print(df_encoded.head())

df_encoded.to_csv("cleaned_diabetes_dataset.csv", index=False)

normalizer = MinMaxScaler()

df_encoded[['BMI']] = normalizer.fit_transform(df_encoded[['BMI']])

df_encoded.head()

df_encoded_copy1=df_encoded

df_encoded_copy2=df_encoded

df_encoded_copy3=df_encoded

Q1 = df_encoded_copy1['BMI'].quantile(0.25)

Q3 = df_encoded_copy1['BMI'].quantile(0.75)

IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR

upper_bound = Q3 + 1.5 * IQR

df_encoded_copy1['BMI'] = np.where(df_encoded_copy1['BMI'] > upper_bound, upper_bound,

np.where(df_encoded_copy1['BMI'] < lower_bound, lower_bound, df_encoded_copy1['BMI']))

print(df_encoded_copy1.head())
```

Program 3

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Screenshot



19-3-25 Linear ~~Logistic~~ Regression

+. ~~Consider a binary classification problem where we want to predict whether a student will pass or fail based on their study hours. The logis~~

1. Given five weeks' sales data (in Thousands), apply linear regression technique to predict the 7th ~~and~~ 9th week sales.

x_i (Week)	y_i (Sales in Thousands)
1	1.2
2	1.8
3	2.6
4	3.2
5	3.8

```
import pandas as pd
import matplotlib.pyplot as plt
x = [1,2,3,4,5]
y = [1.2,1.8,2.6,3.2,3.8]
df = pd.DataFrame({'x': x, 'y': y})
xy = []
x2 = []
for i, j in zip(x,y):
    xy.append(i*j)
for i in x:
    x2.append(i**2)
xmean = sum(x)/len(x)
ymean = sum(y)/len(y)
xy_mean = sum(xy)/len(xy)
x2_mean = sum(x2)/len(x2)
a1 = (xy_mean - x_mean*y_mean) /
     (x2_mean - x_mean**2)
a0 = y_mean - a1 * x_mean
print(f"a0 = {a0}, a1 = {a1}")
x_input = int(input("Enter the value of x:"))
ypred = a0 * a1 * x_input
print(f"Predicted y for x = {x_input} is : {y_pred}")
plt.scatter(x, y, color='blue', label='Data Points')
plt.plot(x, [a0+a1*xi for xi in x], color='red', label='Regression Line')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Linear Regression')
plt.legend()
plt.grid(True)
plt.show()
```

O/P:-

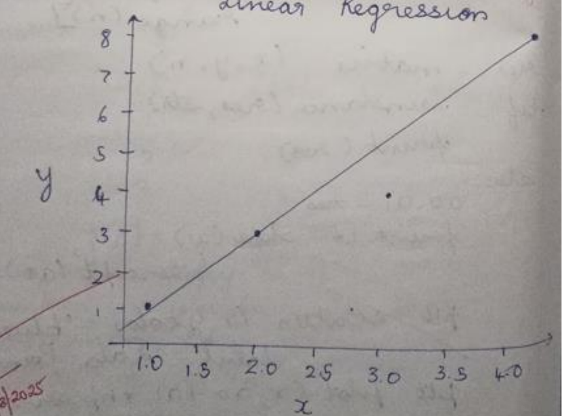
```
a0 = 0.54, a1 = 0.66
Enter the value of x: 7
Predicted y for x = 7 is : 5.16
```

```
xt = ax.T
A = np.dot(xt, ax)
det_A = np.linalg.det(A)
if det_A != 0:
    A_inv = np.linalg.inv(A)
    a = np.dot(A_inv, np.dot
                    (xt, ay))
    return a.flatten()

else:
    return "Inverse doesn't
                    exist"

n = int(input())
x = [int(input(~~f"Enter~~)) for
        i in range(n)]
y = [int(input()) for i in
                    range(n)]
res = matrix (x, y, n)
if isinstance(res, str):
    print(res)
else:
    a0, a1 = res
    print(f"Slope (a1) = {a1},
                    Intercept (a0) = {a0}")
    plt.scatter(x, y, color = 'blue',
                label = 'Data Points')
    plt.plot(x, a0 + a1 * np.array(x),
             color = 'red', label
             = 'Regression Line')
    plt.xlabel = ('x')
    plt.ylabel = ('y')
    plt.title('Linear Regression')
    plt.legend()
    plt.grid(True)
    plt.show()
```

O/P:-

```
4
1
2
3
4
1
3
4
8
Slope (a1) = 2.2 , Intercept (a0) = -1
```

Linear Regression



19/3/2025

Code

```
import pandas as pd

import matplotlib.pyplot as plt

# Data

x = [1, 2, 3, 4, 5]

y = [1.2, 1.8, 2.6, 3.2, 3.8]

df = pd.DataFrame({'x': x, 'y': y})

# Calculate required values

xy = []

x2 = []

for i, j in zip(x, y):

xy.append(i * j)

for i in x:

x2.append(i ** 2)

x_mean = sum(x) / len(x)

y_mean = sum(y) / len(y)

xy_mean = sum(xy) / len(xy)

x2_mean = sum(x2) / len(x2)

# Calculate coefficients a0 and a1

a1 = (xy_mean - x_mean * y_mean) / (x2_mean - x_mean ** 2)

a0 = y_mean - a1 * x_mean

# Output the coefficients

print(f"a0 = {a0}, a1 = {a1}")

# Predict y for a given x value
```

```
x_input = int(input("Enter the value of x: "))
y_pred = a0 + a1 * x_input
print(f"Predicted y for x = {x_input} is: {y_pred}")
# Plotting
plt.scatter(x, y, color='blue', label='Data Points')  # Scatter plot for the data points
plt.plot(x, [a0 + a1 * xi for xi in x], color='red', label='Regression Line')  # Regression line
plt.xlabel('x')
plt.ylabel('y')
plt.title('Linear Regression')
plt.legend()
plt.grid(True)
plt.show()
import numpy as np
import matplotlib.pyplot as plt
def matrix_operations(x, y, n):
ax = np.ones((n, 2))
ax[:, 1] = x
ay = np.array(y).reshape(-1, 1)
xt = ax.T
A = np.dot(xt, ax)
det_A = np.linalg.det(A)
if det_A != 0:
A_inv = np.linalg.inv(A)
a = np.dot(A_inv, np.dot(xt, ay))
```

```
return a.flatten()  # Returns [a0, a1]
else:
return "Inverse doesn't exist"
# Input
n = int(input("Enter the number of elements: "))
x = [int(input(f"Enter the value of x[{i+1}]: ")) for i in range(n)]
y = [int(input(f"Enter the value of y[{i+1}]: ")) for i in range(n)]
# Perform matrix operation
result = matrix_operations(x, y, n)
if isinstance(result, str):
print(result)
else:
a0, a1 = result
print(f"Slope (a1) = {a1}, Intercept (a0) = {a0}")
# Plotting
plt.scatter(x, y, color='blue', label='Data Points')
plt.plot(x, a0 + a1 * np.array(x), color='red', label='Regression Line')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Linear Regression')
plt.legend()
plt.grid(True)
plt.show()
```

Program 3

Build Logistic Regression Model for a given dataset

Screenshot

Logistic Regression

1. Consider a binary classification problem where we want to predict whether a student will pass or fail based on their study hours. The logistic regression model has been trained, and the learned parameters are $a_0 = -5$ (intercept) and $a_1 = 0.8$ (coefficient for study hours).

a. Write the logistic regression equation for this problem.

$$P(y=1|x) = \frac{1}{1+e^{-(a_0+a_1x)}} = \frac{1}{1+e^{-(5+0.8x)}}$$

b. Calculate the probability that a student who studies for 7 hours will pass

$$P=(y=1|7) = \frac{1}{1+e^{-(5+0.8(7))}} = \frac{1}{1+e^{-0.6}}$$

$$\simeq 0.6457 \text{ or } 64.57\%$$

c. Determine the predicted class (pass or fail) for this student based on a threshold of 0.5

Since $P=(y=1|7) = 64.57\%$. above 0.5 or 50%. predicted class is pass

2. Consider $z=[2,1,0]$ for three classes. Apply SoftMax function to find the probability values of three classes.

$$P(y=i|z) = \frac{e^{z_i}}{\sum_j e^{z_j}} \qquad z=[2,1,0]$$

is the SoftMax function.

$z_1 = 2 \quad z_2 = 1 \quad z_3 = 0$

$$P(y=1|z) = \frac{e^{z_1}}{e^{z_3}+e^{z_1}+e^{z_2}} = \frac{e^2}{e^{2}+e^{1}+e^{0}} \approx \frac{7.39}{11.10} \approx 0.665$$

$$P(y=2|z) = \frac{e^{z_2}}{e^{z_3}+e^{z_1}+e^{z_2}} = \frac{e^1}{e^{0}+e^{2}+e^{1}} = \frac{2.71}{11.107} \approx 0.245$$

$$P(y=3|z) = \frac{e^{z_3}}{e^{z_3}+e^{z_1}+e^{z_2}} = \frac{e^0}{e^{0}+e^{2}+e^{1}}$$

$$= \frac{1}{11.107} \approx 0.090$$

Code

```
import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import accuracy_score

# Step 1: Load the dataset

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

# Step 2: Exploratory Data Analysis

print(df.head())

print(df.info())

print(df['left'].value_counts())

# Correlation heatmap

# Only include numeric columns for correlation matrix

numeric_df = df.select_dtypes(include=['number'])

# Correlation heatmap

sns.heatmap(numeric_df.corr(), annot=True, cmap="coolwarm")

plt.title("Correlation Matrix")

plt.show()

# Salary vs Retention

sns.countplot(x='salary', hue='left', data=df)

plt.title("Impact of Salary on Retention")
```

```python
plt.show()

# Department vs Retention

sns.countplot(x='Department', hue='left', data=df)

plt.xticks(rotation=45)

plt.title("Department vs Retention")

plt.show()

# Step 3: Data Preprocessing

df_processed = df.copy()

le = LabelEncoder()

df_processed['salary'] = le.fit_transform(df_processed['salary'])

df_processed['Department'] = le.fit_transform(df_processed['Department'])

# Features selected based on correlation and EDA

features = ['satisfaction_level', 'last_evaluation', 'number_project',

'average_montly_hours', 'time_spend_company',

'Work_accident', 'promotion_last_5years',

'salary', 'Department']

X = df_processed[features]

y = df_processed['left']

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

model = LogisticRegression(max_iter=1000)

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

print(f"Model Accuracy: {accuracy:.2f}")
```

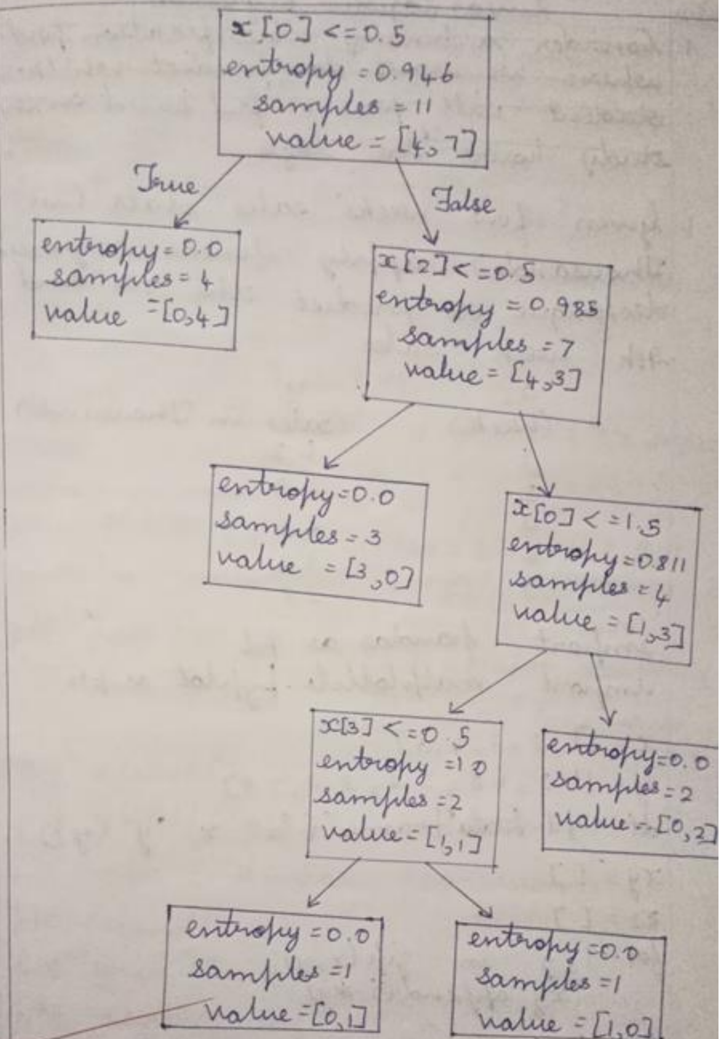
Program 5

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

Screenshot

Implement ID3 (Iterative Dichotomiser3) algorithm on Tennis Weather dataset

```
import pandas as pd
data = pd.readcsv('tennis.csv')
print (data)

from sklearn.preprocessing import LabelEncod
outlook = LabelEncoder()
temp = LabelEncoder()
humidity = LabelEncoder()
wind = LabelEncoder()
play = LabelEncoder()
data ['outlook'] = outlook.fit_transform
                   (data ['outlook'])
data ['temp'] = temp.fit_transform
                   (data ['temp'])
data ['humidity'] = humidity.fit_transfo
                   (data ['humidity'])
data ['windy'] = wind.fit_transform
                   (data ['windy'])
data ['play'] = play.fit_transform
                   (data ['play'])
print (data)

features_cols = ['outlook', 'temp', 'humidit
                 'windy']
x=data[features_cols]
y= data.play
print(x)
print(y)

from sklearn.model_selection impor
     train_test_split

x_train, x_test, y_train, y_test=
     train_test_split(x, y, test_size=0.3)

from sklearn.tree import
     DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion
                 ='entropy')
classifier.fit (x_train, y_train)
classifier.predict(x_test)
classifier. score (x_test, y_test)
clf = DecisionTreeClassifier(criterion
                 ='entropy')
clf.fit (x, y)
importances = clf.feature_importance
feature_importance_df = pd.DataFrame
  ({'Feature': x.columns,
   'Information Gain': importances
  })
feature_importance_df = feature_
  importance_df.sort_values (by=
         'Information Gain',
              ascending = False)
print (feature_importance_df)
from sklearn import tree
tree.plot_tree (classifier)
```



	Feature	Information Gain
0	outlook	0.362629
3	windy	0.274205
2	humidity	0.211237
	temp	0.151929

25/3/25

Code

```
import pandas as pd

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

data

from sklearn.preprocessing import LabelEncoder

outlook=LabelEncoder()

temp=LabelEncoder()

humidity=LabelEncoder()

wind=LabelEncoder()

play=LabelEncoder()

data['outlook']=outlook.fit_transform(data['outlook'])

data['temp']=temp.fit_transform(data['temp'])

data['humidity']=humidity.fit_transform(data['humidity'])

data['windy']=wind.fit_transform(data['windy'])

data['play']=play.fit_transform(data['play'])

data

features_cols=['outlook','temp','humidity','windy']

x=data[features_cols]

y=data.play

print(x)

print()

print(y)

from sklearn.model_selection import train_test_split
```

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

from sklearn.tree import DecisionTreeClassifier

classifier=DecisionTreeClassifier(criterion='entropy')

classifier.fit(x_train,y_train)

classifier.predict(x_test)

classifier.score(x_test,y_test)

clf = DecisionTreeClassifier(criterion='entropy')  # Using entropy to calculate information gain

clf.fit(x, y)

importances = clf.feature_importances_

# Create a DataFrame to see the feature importances

feature_importance_df = pd.DataFrame({

'Feature': x.columns,

'Information Gain': importances

})

# Sort by importance

feature_importance_df = feature_importance_df.sort_values(by='Information Gain', ascending=False)

print(feature_importance_df)
```

Program 6

Build KNN Classification model for a given dataset

Screenshot

Build KNN Classification model for ~~diabetes~~ a given dataset

1. Consider the following dataset, for k = 3 and test data (X, 35, 100) as (Person, Age, SalaryK) solve using Knn classifier model and predict the target.

Person	Age	Salaryk	Target	Euclidean Distance
A	18	50	N	$\sqrt{(35-18)^2+(100-50)^2}$ = 52.3
B	23	55	N	46.6
C	24	70	N	31.9
D	41	60	Y	40.5
E	43	70	Y	31.1
F	38	40	Y	60.1
X	35	100	?	

$d = \sqrt{(Age_1 - Age_2)^2 + (SalaryK_1 - SalaryK_2)^2}$

First 3 as k=3

(41,60) 31.1 Y
(24,70) 31.9 N
(41,60). 40.5 Y

Since majority is Y
(35, 100) Target is Y

2. For Iris dataset
How to choose the k value? Demonstrate using accuracy rate and error rate

1. Split the data into training and testing sets.
2. Train the model with various values of k (e.g. 1,3,5,7,9)
3. Evaluate accuracy and error rate for each k.

$\text{accuracy} = \dfrac{\text{Number of correct predictions}}{\text{Total predictions}}$

error rate = 1 - accuracy.

4. Plot accuracy vs k or error rate vs k and select k with the highest accuracy or lowest error rate.

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

- KNN uses distance, so features with larger values dominate. Scaling ensures fairness.
Using standardization (Z-score normalization).

2/4/2025

Code

```
import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

import seaborn as sns

import matplotlib.pyplot as plt

print("\n--- IRIS DATASET ---")


# Load iris dataset

iris = pd.read_csv("iris.csv")

print("Iris Dataset Shape:", iris.shape)


# Features and target

X_iris = iris.drop(columns='species')

y_iris = iris['species']


# Split data

X_train, X_test, y_train, y_test = train_test_split(X_iris, y_iris, test_size=0.2, random_state=42)


# Instantiate and train KNN

knn_iris = KNeighborsClassifier(n_neighbors=5)
```

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


# Predict and evaluate

y_pred = knn_iris.predict(X_test)

print("Accuracy Score (Iris):", accuracy_score(y_test, y_pred))


# Confusion matrix

cm_iris = confusion_matrix(y_test, y_pred)

sns.heatmap(cm_iris, annot=True, fmt='d', cmap='Blues', xticklabels=knn_iris.classes_, yticklabels=knn_iris.classes_)

plt.title("Confusion Matrix - IRIS")

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.show()


print("\nClassification Report (Iris):")

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

Program 7

Build Support vector machine model for a given dataset

Screenshot

Support Vector Machine (SVM)

Support Vector Machine (SVM) is a supervised learning algorithm used for classification and regression tasks, also called as Maximum Margin Classifier

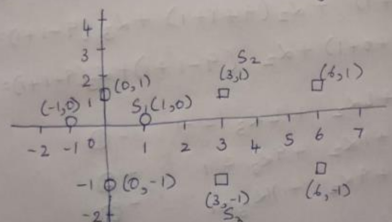
Given

Positively labeled data points

$\left\{\begin{pmatrix}3\\1\end{pmatrix}, \begin{pmatrix}3\\-1\end{pmatrix}, \begin{pmatrix}6\\1\end{pmatrix}, \begin{pmatrix}6\\-1\end{pmatrix}\right\}$

Negatively labeled data points

$\left\{\begin{pmatrix}1\\0\end{pmatrix}, \begin{pmatrix}0\\1\end{pmatrix}, \begin{pmatrix}0\\-1\end{pmatrix}, \begin{pmatrix}-1\\0\end{pmatrix}\right\}$



Support vectors are

$\left\{S_1 = \begin{pmatrix}1\\0\end{pmatrix}, S_2 = \begin{pmatrix}3\\1\end{pmatrix}, S_3 = \begin{pmatrix}3\\-1\end{pmatrix}\right\}$

Each vector is augmented with 1 as a bias input

$\tilde{S}_1 = \begin{pmatrix}1\\0\\1\end{pmatrix} \quad \tilde{S}_2 = \begin{pmatrix}3\\1\\1\end{pmatrix} \quad \tilde{S}_3 = \begin{pmatrix}3\\-1\\1\end{pmatrix}$

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

$$\alpha_1 \begin{pmatrix}1\\0\\1\end{pmatrix}\begin{pmatrix}1\\0\\1\end{pmatrix} + \alpha_2 \begin{pmatrix}3\\1\\1\end{pmatrix}\begin{pmatrix}1\\0\\1\end{pmatrix} + \alpha_3 \begin{pmatrix}3\\-1\\1\end{pmatrix}\begin{pmatrix}1\\0\\1\end{pmatrix} = -1$$
$$\alpha_1 \begin{pmatrix}1\\0\\1\end{pmatrix}\begin{pmatrix}3\\1\\1\end{pmatrix} + \alpha_2 \begin{pmatrix}3\\1\\1\end{pmatrix}\begin{pmatrix}3\\1\\1\end{pmatrix} + \alpha_3 \begin{pmatrix}3\\-1\\1\end{pmatrix}\begin{pmatrix}3\\1\\1\end{pmatrix} = 1$$
$$\alpha_1 \begin{pmatrix}1\\0\\1\end{pmatrix}\begin{pmatrix}3\\-1\\1\end{pmatrix} + \alpha_2 \begin{pmatrix}3\\1\\1\end{pmatrix}\begin{pmatrix}3\\-1\\1\end{pmatrix} + \alpha_3 \begin{pmatrix}3\\-1\\1\end{pmatrix}\begin{pmatrix}3\\-1\\1\end{pmatrix} = 1$$

$$\alpha_1 (1+0+1) + \alpha_2 (3+0+1) + \alpha_3 (3+0+1) = -1$$
$$\alpha_1 (3+0+1) + \alpha_2 (9+1+1) + \alpha_3 (9-1+1) = 1$$
$$\alpha_1 (3+0+1) + \alpha_2 (9-1+1) + \alpha_3 (9+1+1) = 1$$

$$2\alpha_1 + 4\alpha_2 + 4\alpha_3 = -1 \quad - ①$$
$$4\alpha_1 + 11\alpha_2 + 9\alpha_3 = 1 \quad - ②$$
$$4\alpha_1 + 9\alpha_2 + 11\alpha_3 = 1 \quad - ③$$

Solving we get $\alpha_1 = -3.5$, $\alpha_2 = 0.75$, $\alpha_3 = 0.75$

$$\tilde{w} = \sum_i \alpha_i \tilde{S}_i = -3.5 \begin{pmatrix}1\\0\\1\end{pmatrix} + 0.75 \begin{pmatrix}3\\1\\1\end{pmatrix} + 0.75 \begin{pmatrix}3\\-1\\1\end{pmatrix} = \begin{pmatrix}1\\0\\-2\end{pmatrix}$$

$w = \begin{pmatrix}1\\0\end{pmatrix} \quad b = -2$

Code

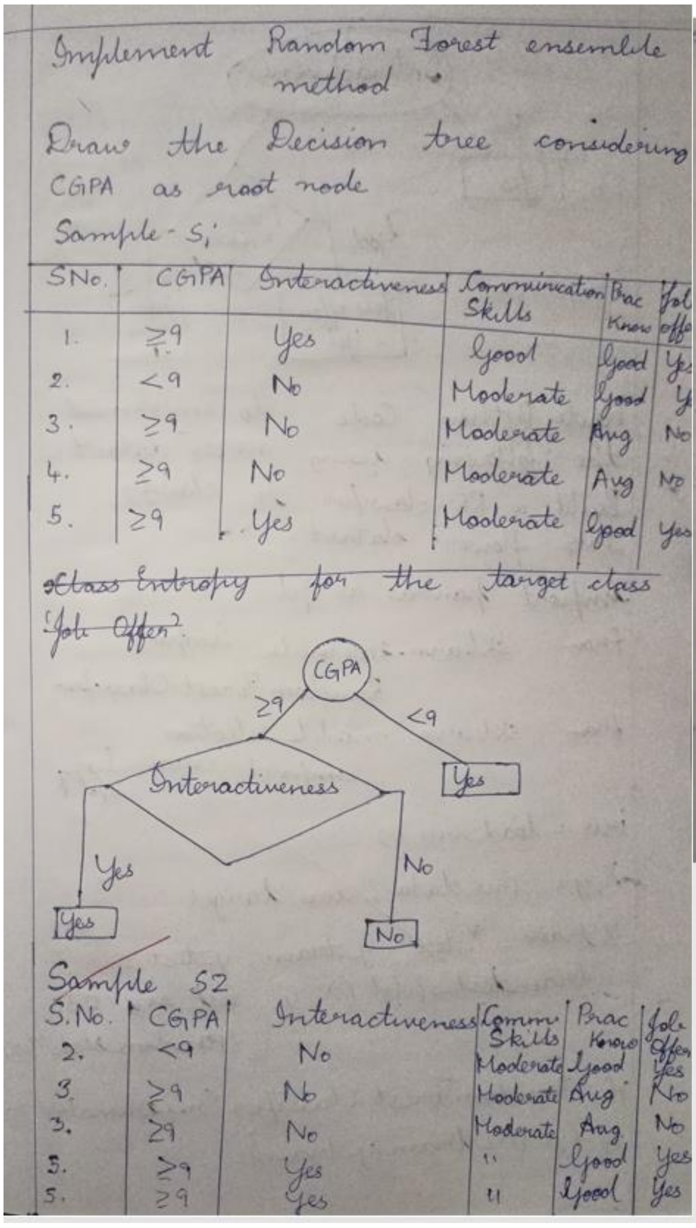
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

df = pd.read_csv("diabetes.csv")
X = df[["Glucose", "BMI"]]
y = df["Outcome"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

model = SVC(kernel='linear')
model.fit(X_train_scaled, y_train)
y_pred = model.predict(X_test_scaled)
print("Accuracy:", accuracy_score(y_test, y_pred))

# Plotting decision boundary
import numpy as np
```

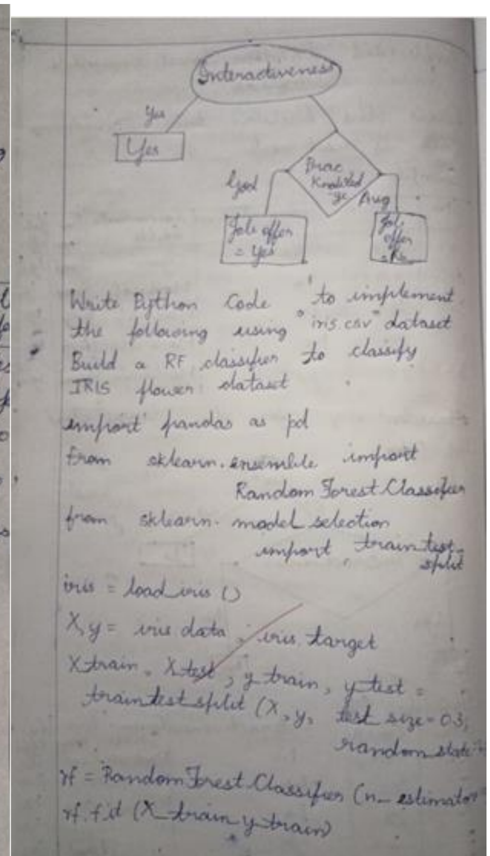
```
# Create meshgrid

x_min, x_max = X_train_scaled[:, 0].min() - 1, X_train_scaled[:, 0].max() + 1

y_min, y_max = X_train_scaled[:, 1].min() - 1, X_train_scaled[:, 1].max() + 1

xx, yy = np.meshgrid(np.linspace(x_min, x_max, 200),

np.linspace(y_min, y_max, 200))


# Predict for each point in the grid

Z = model.predict(np.c_[xx.ravel(), yy.ravel()])

Z = Z.reshape(xx.shape)


# Plot

plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.3)

plt.scatter(X_train_scaled[:, 0], X_train_scaled[:, 1], c=y_train, cmap=plt.cm.coolwarm,
edgecolors='k')

plt.xlabel("Glucose (scaled)")

plt.ylabel("BMI (scaled)")

plt.title("Linear SVM Decision Boundary")

plt.show()
```

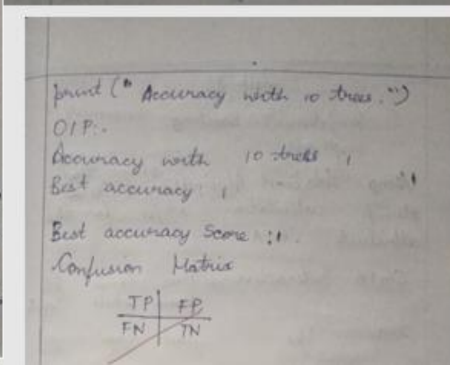
Program 8

Implement Random Forest ensemble method on a given dataset

Screenshot







Code

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

# Step 1: Load dataset
iris = pd.read_csv("iris.csv")

# Step 2: Features and labels
X = iris.drop(columns='species')
y = iris['species']

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

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

print("\nRandom Forest with 10 trees:")
```

```
print("Accuracy:", accuracy_score(y_test, y_pred))

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


# Step 5: Confusion Matrix

cm = confusion_matrix(y_test, y_pred)

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=rf_default.classes_,
yticklabels=rf_default.classes_)

plt.title("Confusion Matrix (10 Trees)")

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.show()


# Step 6: Fine tune number of trees

scores = []

trees_range = range(1, 51)


for n in trees_range:

    rf = RandomForestClassifier(n_estimators=n, random_state=42)

    rf.fit(X_train, y_train)

    score = rf.score(X_test, y_test)

    scores.append(score)


# Step 7: Plot results

plt.figure(figsize=(10, 5))

plt.plot(trees_range, scores, marker='o')
```

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

plt.xlabel("Number of Trees")

plt.ylabel("Accuracy")

plt.grid(True)

plt.show()


# Step 8: Best score and corresponding number of trees

best_n = trees_range[scores.index(max(scores))]

print(f"\nBest Accuracy: {max(scores):.2f} using {best_n} trees")
```

Program 9

Implement Boosting ensemble method on a given dataset

Screenshot

Lab-8

Implement Boosting ensemble method

Using AdaBoost Algorithm show the Decision stump calculation steps for the attribute CGPA.

CGPA	Interactiveness	Prac Know	Comm Skill	Job Profile
>=9	Yes	Good	Good	Yes
<9	No	Good	Moderate	Yes
>=9	No	Avg	"	No
<9	"	"	Good	"
>=9	Yes	Good	Moderate	Yes
>=9	"	"	"	"

α_c

1) Initial weight = 1/6

$$\varepsilon_i = \sum_{j=1}^{n} H_i(d_j) Wt(d_j)$$

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

$$\alpha_{CGPA} = \frac{1}{2} \ln \frac{(1 - \varepsilon_{CGPA})}{\varepsilon_{CGPA}} = 0.347$$

$$Z_{CGPA} = \frac{1}{6} \times 4 \times e^{-0.347} + \frac{1}{6} \times 2 \times e^{+0.347} = 0.9428$$

$$Wt(d_j)_{i+1} = \frac{\frac{1}{6} \times e^{-0.347}}{0.9428} = 0.1249$$

for incorrect instance

$$= \frac{\frac{1}{6} \times e^{+0.347}}{0.9423} = 0.2601$$

CGPA.	Initial Weight	Updated Weight
>=9	1/6	0.1249
<9	1/6	0.2501
>=9	1/6	0.1249
<9	1/6	0.2501
>=9	1/6	0.1249
>=9	1/6	0.1249

Best accuracy score = 1

Confusion matrix

TN FP

FN TP

Code

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

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

# Features and target
X = df.drop(columns='income_level')
y = df['income_level']

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

# Train AdaBoost with default n_estimators=10
ab_default = AdaBoostClassifier(n_estimators=10, random_state=42)
ab_default.fit(X_train, y_train)
y_pred = ab_default.predict(X_test)
```

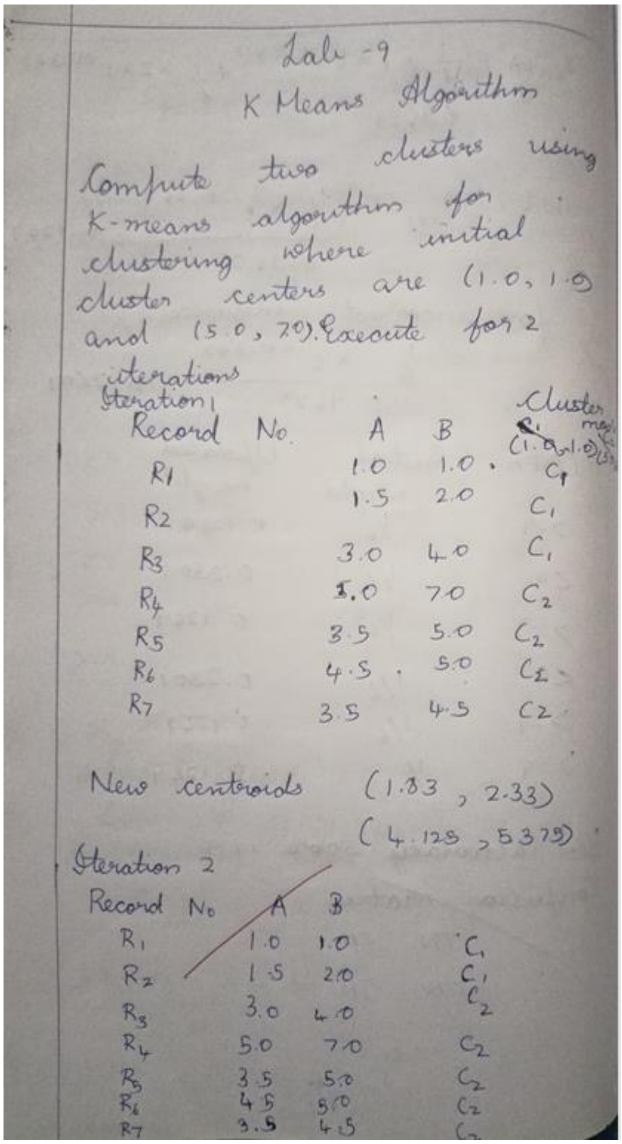
```python
print("\nAdaBoost Classifier with 10 Trees")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title("Confusion Matrix (10 Trees)")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

# Fine-tune number of estimators
scores = []
trees_range = range(1, 101)

for n in trees_range:
    ab = AdaBoostClassifier(n_estimators=n, random_state=42)
    ab.fit(X_train, y_train)
    score = ab.score(X_test, y_test)
    scores.append(score)

# Plot accuracy vs. number of trees
```

```
plt.figure(figsize=(10, 5))

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

plt.title("AdaBoost Accuracy vs Number of Trees")

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

plt.ylabel("Accuracy")

plt.grid(True)

plt.show()


# Best accuracy and corresponding number of trees

best_n = trees_range[np.argmax(scores)]

print(f"\nBest Accuracy: {max(scores):.4f} using {best_n} trees")
```

Program 10

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

Screenshot


Lab - 9

K Means Algorithm

Compute two clusters using K-means algorithm for clustering where initial cluster centers are (1.0, 1.0) and (5.0, 7.0). Execute for 2 iterations

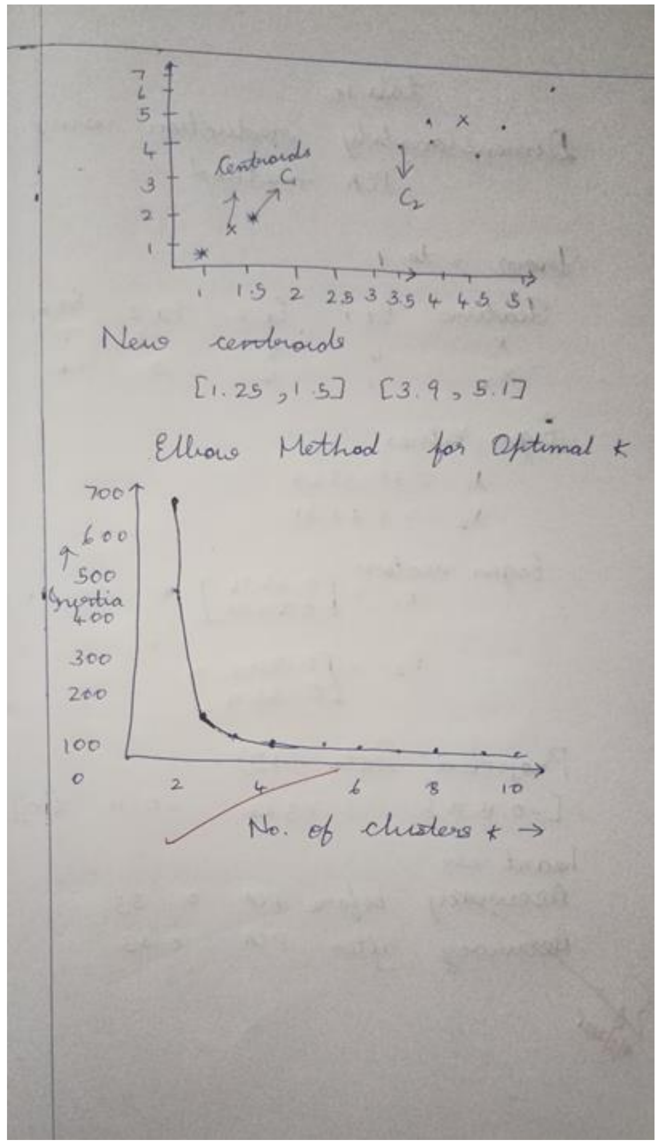
Iteration 1

Record No.	A	B	Cluster (1.0, 1.0)
R1	1.0	1.0	C1
R2	1.5	2.0	C1
R3	3.0	4.0	C1
R4	5.0	7.0	C2
R5	3.5	5.0	C2
R6	4.5	5.0	C2
R7	3.5	4.5	C2

New centroids (1.83, 2.33) (4.125, 5.375)

Iteration 2

Record No	A	B	
R1	1.0	1.0	C1
R2	1.5	2.0	C1
R3	3.0	4.0	C2
R4	5.0	7.0	C2
R5	3.5	5.0	C2
R6	4.5	5.0	C2
R7	3.5	4.5	[illegible]



New centroids

[1.25, 1.5] [3.9, 5.1]

Elbow Method for Optimal k

Code

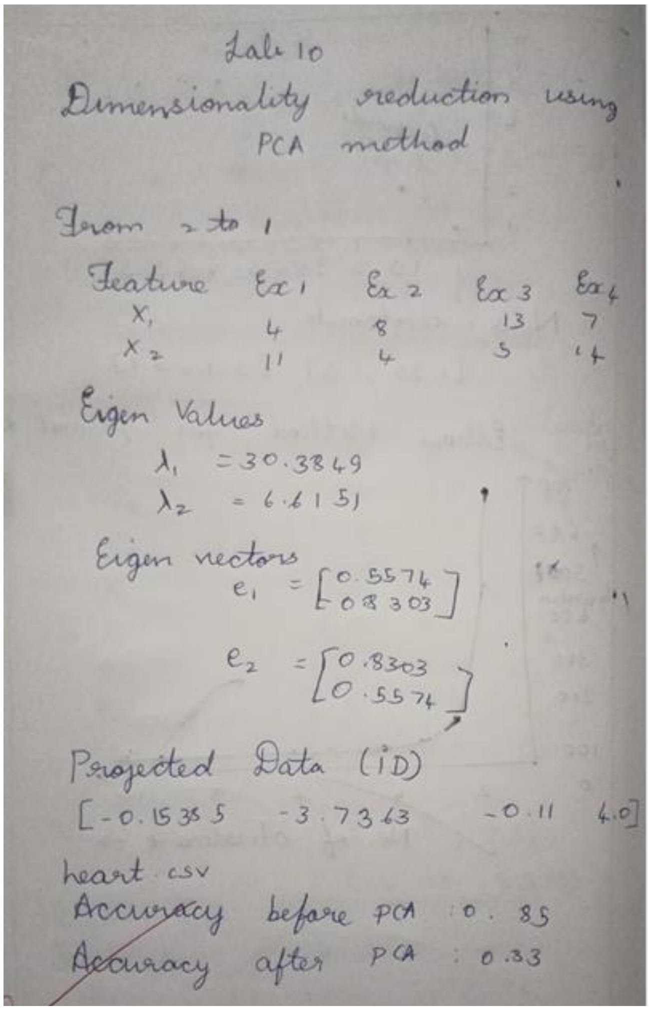
```
import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

iris = pd.read_csv("iris.csv")

X = iris[['petal_length', 'petal_width']]

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

sns.scatterplot(x='petal_length', y='petal_width', hue=iris['species'], data=iris)

plt.title("Original Iris Petal Data")

plt.show()

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)

inertia = []

k_range = range(1, 11)

for k in k_range:

km = KMeans(n_clusters=k, random_state=42, n_init=10)

km.fit(X_scaled)

inertia.append(km.inertia_)

plt.figure(figsize=(8, 5))

plt.plot(k_range, inertia, marker='o')

plt.title("Elbow Plot for Optimal k")

plt.xlabel("Number of Clusters (k)")
```

```
plt.ylabel("Inertia (Within-cluster Sum of Squares)")
plt.grid(True)
plt.show()
optimal_k = 3
kmeans = KMeans(n_clusters=optimal_k, random_state=42, n_init=10)
clusters = kmeans.fit_predict(X_scaled)
plt.figure(figsize=(6, 4))
sns.scatterplot(x=X['petal_length'], y=X['petal_width'], hue=clusters, palette='Set2')
plt.title(f"K-Means Clustering with k={optimal_k}")
plt.xlabel("Petal Length")
plt.ylabel("Petal Width")
plt.show()
```

Program 11

Implement Dimensionality reduction using Principal Component Analysis (PCA) method

Screenshot


Lab 10

Dimensionality reduction using PCA method

From 2 to 1

Feature	Ex 1	Ex 2	Ex 3	Ex 4
X_1	4	8	13	7
X_2	11	4	5	14

Eigen Values

$\lambda_1 = 30.3849$

$\lambda_2 = 6.6151$

Eigen vectors

$e_1 = \begin{bmatrix} 0.5574 \\ -0.8303 \end{bmatrix}$

$e_2 = \begin{bmatrix} 0.8303 \\ 0.5574 \end{bmatrix}$

Projected Data (1D)

[-0.15355 -3.7363 -0.11 4.0]

heart.csv

Accuracy before PCA : 0.85

Accuracy after PCA : 0.33

Code

import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.decomposition import PCA

from sklearn.metrics import accuracy_score

```
from sklearn.svm import SVC

from sklearn.linear_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier


# Load dataset

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


# Step 1: Encode categorical features

df_encoded = df.copy()

label_cols = ['Sex', 'ChestPainType', 'RestingECG', 'ExerciseAngina', 'ST_Slope']


# Apply label encoding for binary columns, one-hot encoding for others

for col in label_cols:

if df_encoded[col].nunique() == 2:

df_encoded[col] = LabelEncoder().fit_transform(df_encoded[col])

else:

df_encoded = pd.get_dummies(df_encoded, columns=[col], drop_first=True)


# Step 2: Split features and target

X = df_encoded.drop(columns='HeartDisease')

y = df_encoded['HeartDisease']


# Step 3: Feature Scaling

scaler = StandardScaler()
```

```
X_scaled = scaler.fit_transform(X)

# Step 4: Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Step 5: Train classifiers
models = {
'SVM': SVC(),
'Logistic Regression': LogisticRegression(max_iter=1000),
'Random Forest': RandomForestClassifier()
}

print("Accuracy Before PCA:")
for name, model in models.items():
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
acc = accuracy_score(y_test, y_pred)
print(f"{name}: {acc:.4f}")

# Step 6: Apply PCA to retain 95% variance
pca = PCA(n_components=0.95)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)
```

```python
print("\nAccuracy After PCA:")

for name, model in models.items():

model.fit(X_train_pca, y_train)

y_pred_pca = model.predict(X_test_pca)

acc_pca = accuracy_score(y_test, y_pred_pca)

print(f"{name} (PCA): {acc_pca:.4f}")


# Optional: Display PCA variance ratios

print("\nPCA Explained Variance Ratios:")

print(np.round(pca.explained_variance_ratio_ * 100, 2))
```