VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

Sagar Bangari (1BM22CS231)

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 **,** who is bonafide student Sagar bangari(1BM22CS231) of **B.M.S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum. The Lab report has been approved as it satisfies the academic requirements in respect of an Machine Learning (23CS6PCMAL) work prescribed for the said degree.

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

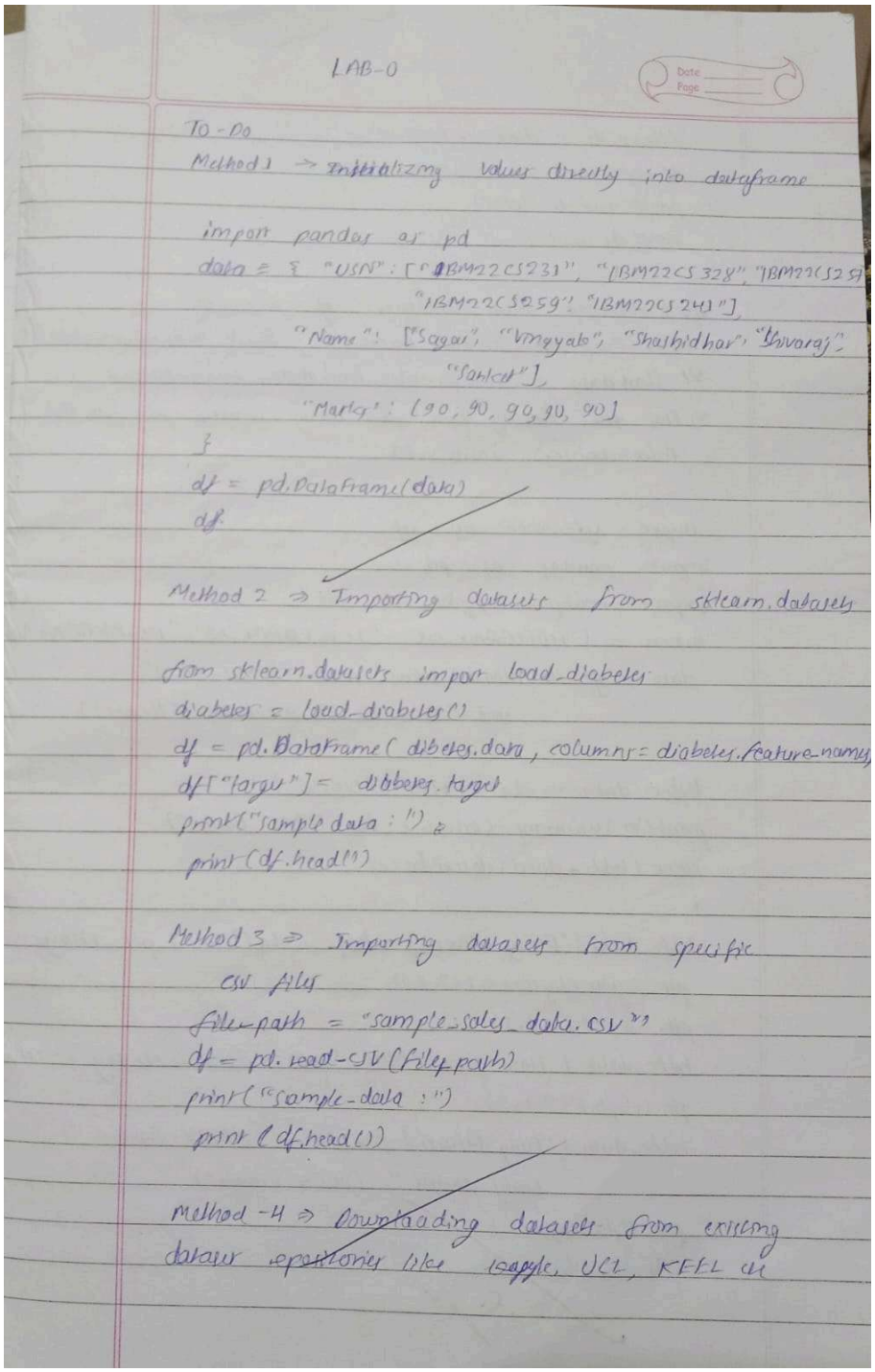
Index

Github Link:
https://github.com/Sagar-bangari/6thsemML_LAb

Program 1

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

Screenshot:



```
file_path = "Diabetes.csv"
df = pd.read_csv(file_path)
print("sample data")
print(df.head())
```

Stock Market Data Analysis of following
1) HDFC Bank LTD, ICICI Bank Ltd, Kotak mahindra Bank Ltd
2) Start date = 2024-01-01, End date: 2024-12-30
3) Plot the closing price and daily returns for all the three banks mentioned

```
import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt
tickers = ["HDFCBANK.NS", "ICICIBANK.NS", "KOTAKBANK.NS"]
data = yf.download(tickers, start="2024-01-01",
                   end="2024-12-30", group_by="ticker")

hdfc_data = data['HDFCBANK.NS']
print("\n Summary Statistics for HDFC Bank:")
print(hdfc_data.describe())

hdfc_data['Daily return'] = hdfc_data['Close'].pct_change
plt.figure(figsize=(12,6))
plt.subplot(2,1,1)
hdfc_data['Close'].plot(title="HDFC Bank - Closing price")
plt.subplot(2,1,2)
hdfc_data['Daily Return'].plot(title="KOTAK Bank -
        Daily returns", color='orange')
plt.tight_layout()
plt.show()
```

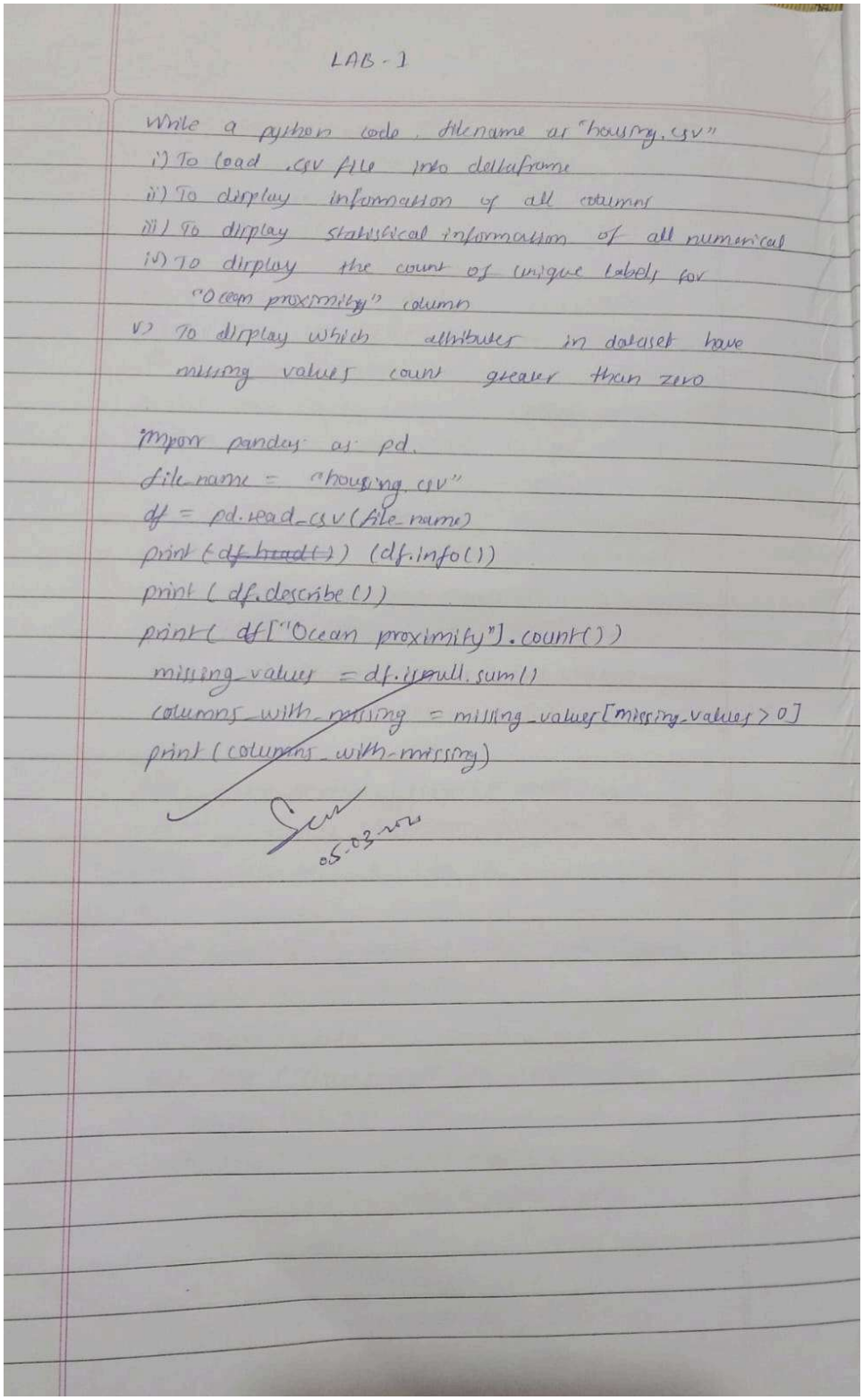
Code:

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

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

      data = yf.download(tickers, start="2024-01-01", end="2024-12-30",
group_by='ticker')

      hdfc_data = data['HDFCBANK.NS']
      print("\nSummary statistics for HDFC bank :")
      print(hdfc_data.describe())

      hdfc_data['Daily Return'] = hdfc_data['Close'].pct_change()
      plt.figure(figsize=(12, 6))
      plt.subplot(2, 1, 1)
      hdfc_data['Close'].plot(title="HDFC bank - Closing Price")
      plt.subplot(2, 1, 2)
      hdfc_data['Daily Return'].plot(title="Reliance Industries - Daily
Returns", color='orange')
      plt.tight_layout()
      plt.show()


      icici_data = data['ICICIBANK.NS']
      print("\nSummary statistics for ICICI bank :")
      print(icici_data.describe())

      icici_data['Daily Return'] = icici_data['Close'].pct_change()
      plt.figure(figsize=(12, 6))
      plt.subplot(2, 1, 1)
      icici_data['Close'].plot(title="ICICI bank - Closing Price")
      plt.subplot(2, 1, 2)
      icici_data['Daily Return'].plot(title="ICICI Bank - Daily Returns",
color='orange')
      plt.tight_layout()
      plt.show()
```

```
      kotak_data = data['KOTAKBANK.NS']
      print("\nSummary statistics for KOTAK bank :")
      print(kotak_data.describe())

      kotak_data['Daily Return'] = kotak_data['Close'].pct_change()
      plt.figure(figsize=(12, 6))
      plt.subplot(2, 1, 1)
      kotak_data['Close'].plot(title="KOTAK bank - Closing Price")
      plt.subplot(2, 1, 2)
      kotak_data['Daily Return'].plot(title="KOTAK Bank  - Daily Returns",
color='orange')
      plt.tight_layout()
      plt.show()
```

Program 2

Screenshot:



LAB - 1

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

i) To load .csv file into dataframe

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 in dataset have missing values count greater than zero

```
import pandas as pd.
file_name = "housing.csv"
df = pd.read_csv(file_name)
print (df.head()) (df.info())
print ( df.describe())
print ( df["Ocean proximity"].count())
missing_values = df.isnull.sum()
columns_with_missing = missing_values[missing_values > 0]
print (columns_with_missing)
```

05.03.2025

Write python code implement following data processing techniques for diabetes dataset

1) Data cleaning :- Handling missing values, Handling categorical data, Handling outliers
2) Data transformations :- Min-max Scaler / Normalized, Standard scaler

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler, StandardScaler,
        LabelEncoder
diabetes = pd.read_csv('diabetes.csv')
diabetes.drop(columns=['ID', 'NO_Patient'], inplace=True)
diabetes.dropna(inplace=True)
diabetes[['Gender', 'CLASS']] = diabetes[['Gender', 'CLASS']].
        apply(LabelEncoder().fit_transform)

diabetes = diabetes[(np.abs((diabetes.select_dtypes(include=
    [np.number]) - diabetes.mean()) / diabetes.std()) < 3).all(axis=1)]

scaler = MinMaxScaler()
diabetes_scaled = pd.DataFrame(scaler.fit_transform(diabetes.drop
    (columns=['CLASS'])), columns=diabetes.columns[:-1])
diabetes_scaled['CLASS'] = diabetes['CLASS'].values
print(diabetes_scaled.head())
```

Sev
06.03.25

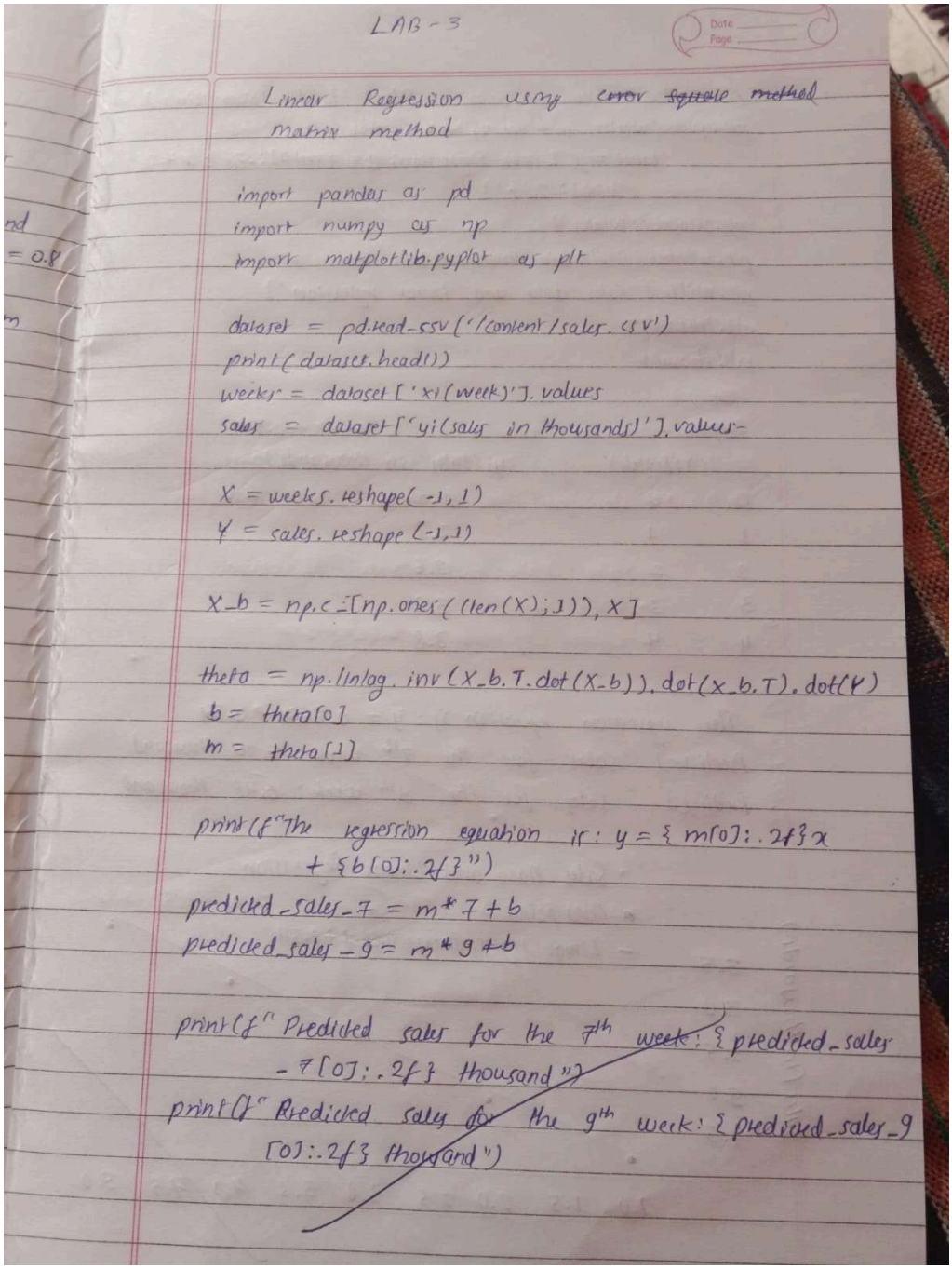
Code:

```
import pandas as pd

import numpy as np

from sklearn.preprocessing import MinMaxScaler, StandardScaler, LabelEncoder



# Load dataset

diabetes = pd.read_csv('diabetes.csv')  # Replace with actual file path



diabetes.drop(columns=['ID', 'No_Pation'], inplace=True)  # Drop unnecessary
columns



diabetes.dropna(inplace=True)  # Handle missing values



diabetes[['Gender', 'CLASS']] = diabetes[['Gender',
'CLASS']].apply(LabelEncoder().fit_transform)  # Encode categorical data



diabetes = diabetes[(np.abs((diabetes.select_dtypes(include=[np.number]) -
diabetes.mean()) / diabetes.std()) < 3).all(axis=1)]  # Remove outliers



scaler = MinMaxScaler()

diabetes_scaled =
pd.DataFrame(scaler.fit_transform(diabetes.drop(columns=['CLASS'])),
columns=diabetes.columns[:-1])

diabetes_scaled['CLASS'] = diabetes['CLASS'].values  # Add target column back
```

```
print(diabetes_scaled.head())
```

Program 3

Screenshot:



```
plt.scatter(weeks, sales, color='blue', label='Data points')
plt.plot(weeks, m*weeks+b, color='red',
         label=f'Linear Regression: y = {m[0]:.2f}x +
         {b[0]:.2f}')
plt.xlabel('Weeks')
plt.ylabel('Sales (in thousands)')
plt.title('Sales data and Linear regression')
plt.legend()
plt.show()
```

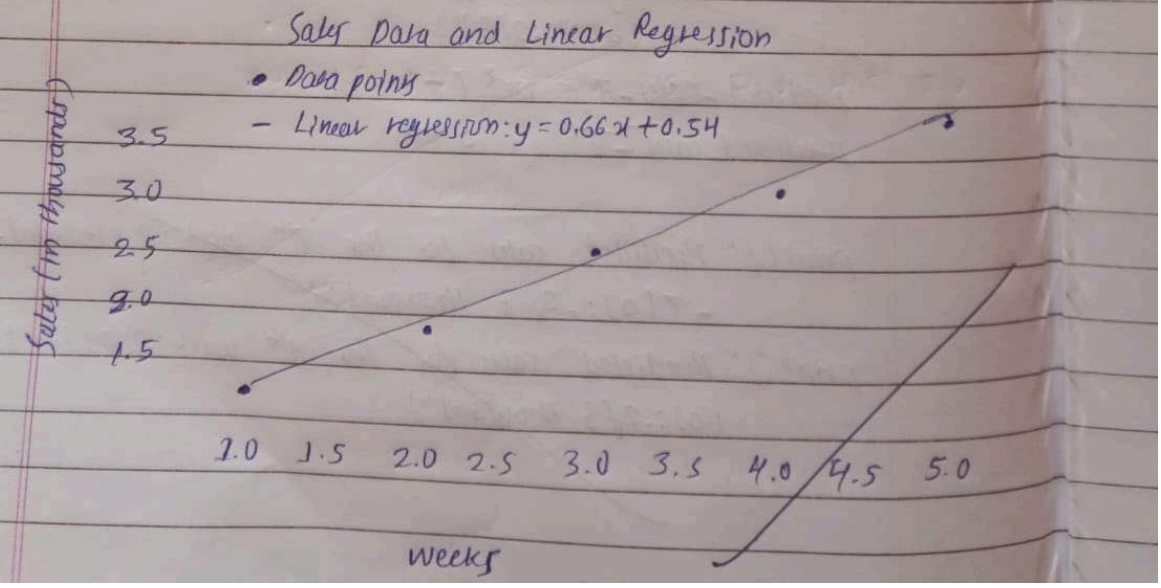
Output:-

	Xi (week)	yi (Sales in thousands)
0	0	1.2
1	1	1.8
2	2	2.6
3	3	3.2
4	4	3.8

The regression equation is: $y = 0.66x + 0.54$

Predicted Sales for the 7th week: 5.16 thousand

Predicted Sales for the 9th week: 6.48 thousand



Linear Regression using error squared method

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

dataset = pd.read_csv('content/sales.csv')
print(dataset.head())

weeks = dataset['x(1 weeks)'].values
Sales = dataset['y(Sales in thousands)'].values
n = len(weeks)

sum_x = np.sum(weeks)
sum_y = np.sum(sales)
sum_x2 = np.sum(weeks**2)
sum_xy = np.sum(weeks * sales)

m = (n * sum_xy - sum_x * sum_y) / (n * sum_x2
    - sum_x**2)
b = (sum_y - m * sum_x) / n

print(f"The regression equation is: y = {m:.2f}x +
    {b:.2f}")

print(f"Predicted sales for 7th week: {predicted_sales_7:1.2f}
    thousand")
print(f"Predicted sales for 9th week: {predicted_sales:.2f}
    thousand")
```

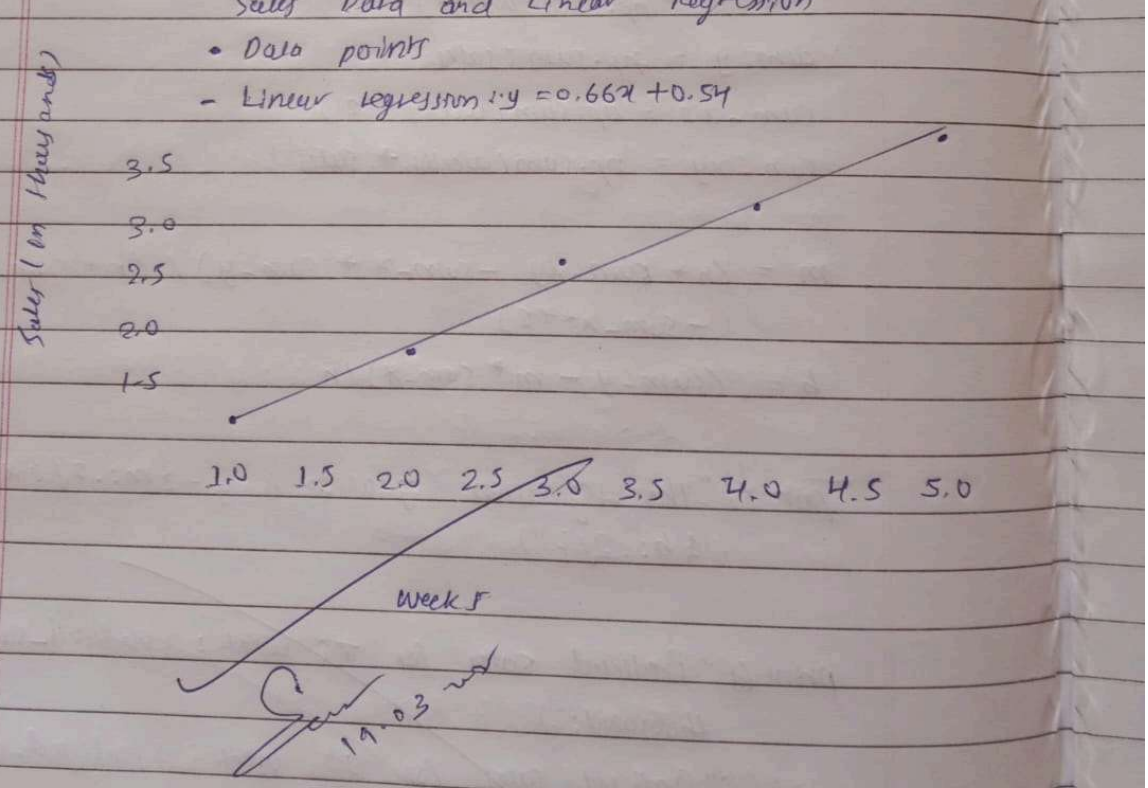
```
plt.scatter(weeks, sales, color='blue', label='Data points')
plt.plot(weeks, m*weeks+b, color='red',
         label=f'Linear Regression: y={m:.2f}x+{b:.2f}')
plt.xlabel('Weeks')
plt.ylabel('Sales (in thousands)')
plt.title('Sales Data and Linear Regression')
plt.legend()
plt.show()
```

Output :-

The regression equation is: $y = 0.66x + 0.54$

Predicted sales for the 7th week : 5.16 thousand

Predicted sales for the 9th week : 6.48 thousand



Code:

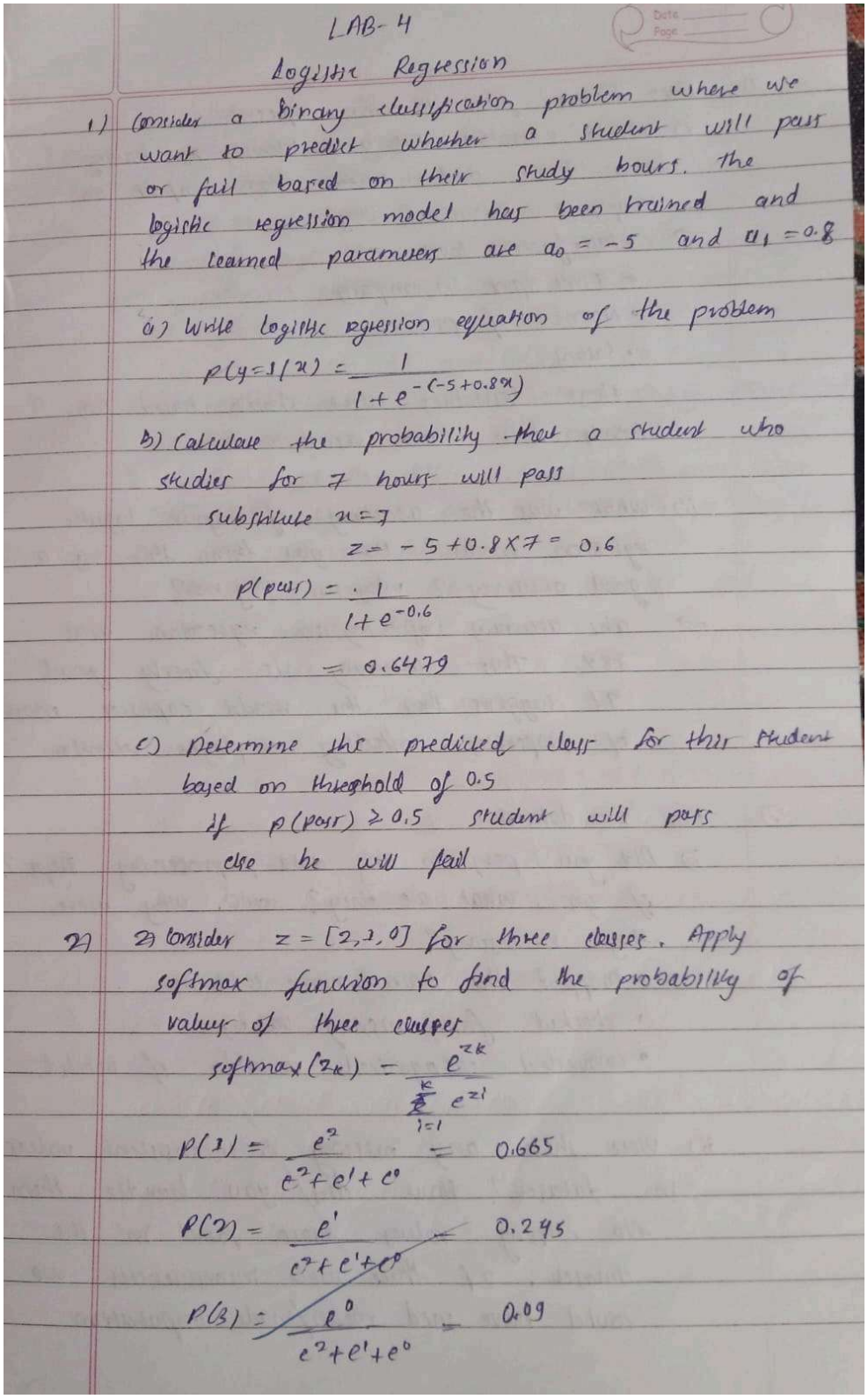
```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt



# Load the dataset from CSV file

dataset = pd.read_csv('/content/sales.csv')



# Display the first few rows of the dataset

print(dataset.head())



# Assuming the CSV has columns 'Week' and 'Sales' (adjust based on your
actual column names)

weeks = dataset['xi(week)'].values

sales = dataset['yi(Sales in thousands)'].values



# Reshaping weeks for matrix operations (make it a column vector)

X = weeks.reshape(-1, 1)

y = sales.reshape(-1, 1)



# Add a column of ones to X to account for the intercept (b)

X_b = np.c_[np.ones((len(X), 1)), X]  # The "1" is for the bias term
(intercept)
```

```
# Compute theta using the normal equation: θ = (X^T X)^(-1) X^T y
theta = np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(y)

# Extract slope (m) and intercept (b) from theta
b = theta[0]
m = theta[1]

# Print the regression line equation
print(f"The regression equation is: y = {m[0]:.2f}x + {b[0]:.2f}")

# Predict the sales for 7th and 9th weeks
week_7 = 7
week_9 = 9

predicted_sales_7 = m * week_7 + b
predicted_sales_9 = m * week_9 + b

print(f"Predicted sales for the 7th week: {predicted_sales_7[0]:.2f}
thousand")
print(f"Predicted sales for the 9th week: {predicted_sales_9[0]:.2f}
thousand")
```

```
# Plot the data points and the regression line

plt.scatter(weeks, sales, color='blue', label='Data Points')

plt.plot(weeks, m * weeks + b, color='red', label=f'Linear Regression: y =
{m[0]:.2f}x + {b[0]:.2f}')

plt.xlabel('Weeks')

plt.ylabel('Sales (in thousands)')

plt.title('Sales Data and Linear Regression')

plt.legend()

plt.show()
```

Program 4

Screenshot:



LAB- 4

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 $a_0 = -5$ and $a_1 = 0.8$

a) Write logistic regression equation of the problem

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

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

substitute $x=7$

$$z = -5 + 0.8 \times 7 = 0.6$$

$$P(pass) = \frac{1}{1+e^{-0.6}}$$

$$= 0.6479$$

c) Determine the predicted class for this student based on threshold of 0.5

if $P(pass) \geq 0.5$ student will pass

else he will fail

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

$$softmax(z_k) = \frac{e^{z_k}}{\sum_{i=1}^{k} e^{z_i}}$$

$$P(1) = \frac{e^2}{e^2+e^1+e^0} = 0.665$$

$$P(2) = \frac{e^1}{e^2+e^1+e^0} = 0.245$$

$$P(3) = \frac{e^0}{e^2+e^1+e^0} = 0.09$$

1) For dataset file "HR_comma_sep.csv"

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

→ • Satisfaction level
• Time spent in company
• Number of projects
• Salary.

These variables were chosen based on trends in data visualization

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

→ The accuracy of logistic regression was 78%. This accuracy is fairly good. It suggests that the model captures most of properties affecting employee retention

2) For zoo dataset

i) Did you perform any data preprocessing steps? if yes, what are they? and why were they necessary?

• Dropped the animal_name column
• checked for missing values
• converted categorical variables if needed.

ii) Were there any missing or inconsistent values in dataset? How did you handle them

No missing values were found in the dataset. If there were inconsistencies we could have used mean/mode imputation

iii) what does the cofusion matrix tell you about the performance of model

→ Confusion matrix showed how well the model predicted different class types

→ A high number of correct predictions along the diagonal of the matrix indicates good performance

iv) Which class types were most frequently misclassified why do you think this happened?

→ The most frequently misclassified classes were likely amphibians, birds or reptiles as they share similar features

→ possible reasons are feature overlap and simplified model

Code:

```
import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score



# Load dataset

file_path = "/content/HR_comma_sep"

df = pd.read_csv(file_path)



# Exploratory Data Analysis

# Plot bar chart showing impact of salaries on retention

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

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

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

plt.xlabel("Salary Level")

plt.ylabel("Number of Employees")

plt.legend(["Stayed", "Left"])

plt.show()
```

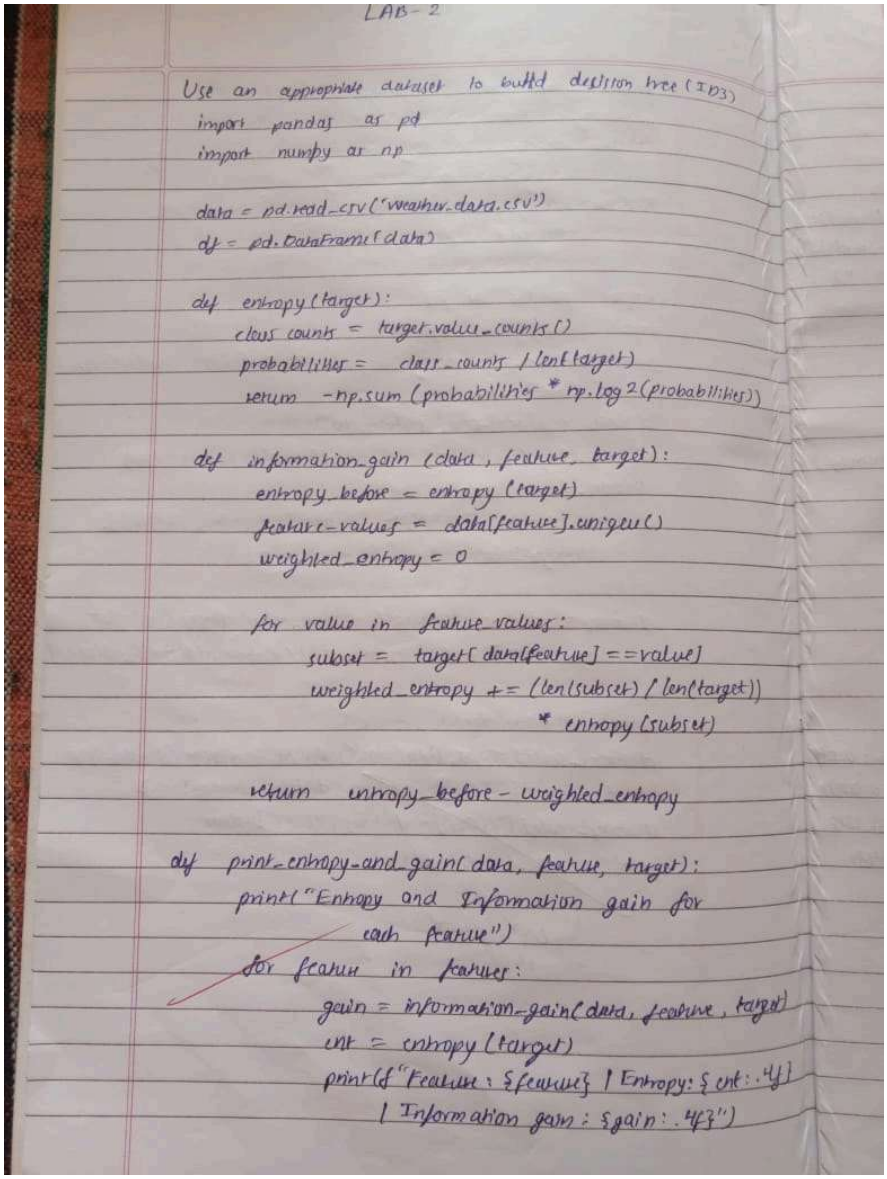
```
# Plot bar chart showing correlation between department and employee
retention

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

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

plt.title("Department-wise Employee Retention")

plt.xlabel("Department")

plt.ylabel("Number of Employees")

plt.xticks(rotation=45)

plt.legend(["Stayed", "Left"])

plt.show()



# Selecting key features based on analysis

X = df[['satisfaction_level', 'time_spend_company', 'Work_accident',
'salary']]

X = pd.get_dummies(X, columns=['salary'], drop_first=True)  # Convert
categorical 'salary' to numerical

y = df['left']



# Splitting data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.9,
random_state=10)



# Train logistic regression model

model = LogisticRegression(max_iter=1000)
```

```
model.fit(X_train, y_train)



# Predictions

y_predicted = model.predict(X_test)



# Model accuracy

accuracy = accuracy_score(y_test, y_predicted)

print(f"Model Accuracy: {accuracy:.4f}")



# Plotting actual vs predicted values

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

sns.heatmap(pd.crosstab(y_test, y_predicted), annot=True, fmt='d',
cmap='Blues')

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.title("Confusion Matrix")

plt.show()



# Predict probability of an employee leaving

def predict_leave_probability(features):

    return model.predict_proba([features])[0][1]  # Probability of leaving



# Example prediction
```

```
example_employee = [[0.5, 3, 0, 1, 0]]  # Sample feature values with one-hot
encoded salary

probability = predict_leave_probability(example_employee[0])

print(f"Probability of leaving: {probability:.4f}")
```

Program 5

Screenshot:



LAB-2

Use an appropriate dataset to build decision tree (ID3)

```
import pandas as pd
import numpy as np

data = pd.read_csv("weather_data.csv")
df = pd.DataFrame(data)

def entropy(target):
    class_counts = target.value_counts()
    probabilities = class_counts / len(target)
    return -np.sum(probabilities * np.log2(probabilities))

def information_gain(data, feature, target):
    entropy_before = entropy(target)
    feature_values = data[feature].unique()
    weighted_entropy = 0

    for value in feature_values:
        subset = target[data[feature] == value]
        weighted_entropy += (len(subset) / len(target))
                            * entropy(subset)

    return entropy_before - weighted_entropy

def print_entropy_and_gain(data, feature, target):
    print("Entropy and Information gain for
          each feature")
    for feature in features:
        gain = information_gain(data, feature, target)
        ent = entropy(target)
        print(f"Feature: {feature} | Entropy: {ent:.4f}
              | Information gain: {gain:.4f}")
```

```
def build_tree (data, target, features):
    if len(target.unique()) == 1:
        return target.iloc[0]

    if len(features) == 0:
        return target.mode()[0]

    gains = {feature: information_gain(data, feature,
                target) for feature in features}
    best_feature = max(gains, key=gains.get)

    tree = {best_feature: {}}
    feature_values = data[best_feature].unique()

    for value in feature_values:
        subset_data = data[data[best_feature] == value]
        subset_target = target[data[best_feature] == value]

        remaining_features = [f for f in
                features if f != best_feature
        subtree = build_tree(subset_data, subset_target,
                remaining_features)
        tree[best_feature][value] = subtree

    return tree
```

```
def print_tree(tree, indent=""):
    if isinstance(tree, dict):
        for feature, branches in tree.items():
            print(f"{indent}{feature}:")
            for value, subtree in branches.items():
                print(f"{indent} {value} ->", end="")
                print_tree(subtree, indent + "  ")

    else:
        print(f"{indent}{tree}")

target = df['Play Tennis']
features = ['Outlook', 'Temperature', 'Humidity', 'Windy']

print_entropy_and_gain(df, features, target)
tree = build_tree(df, target, features)
print("\nDecision tree")
print_tree(tree, indent="  ")
```

output:-

Entropy and Information gain for each feature

Feature: Outlook | Entropy: 0.9403 | Information gain: 0.241

Feature: Temperature | Entropy: 0.9403 | Information gain: 0.029

Feature: Humidity | Entropy: 0.9403 | Information gain: 0.151

Feature: Windy | Entropy: 0.9403 | Information gain: 0.048

Decision tree

Outlook:

Sunny → Humidity:

High → No

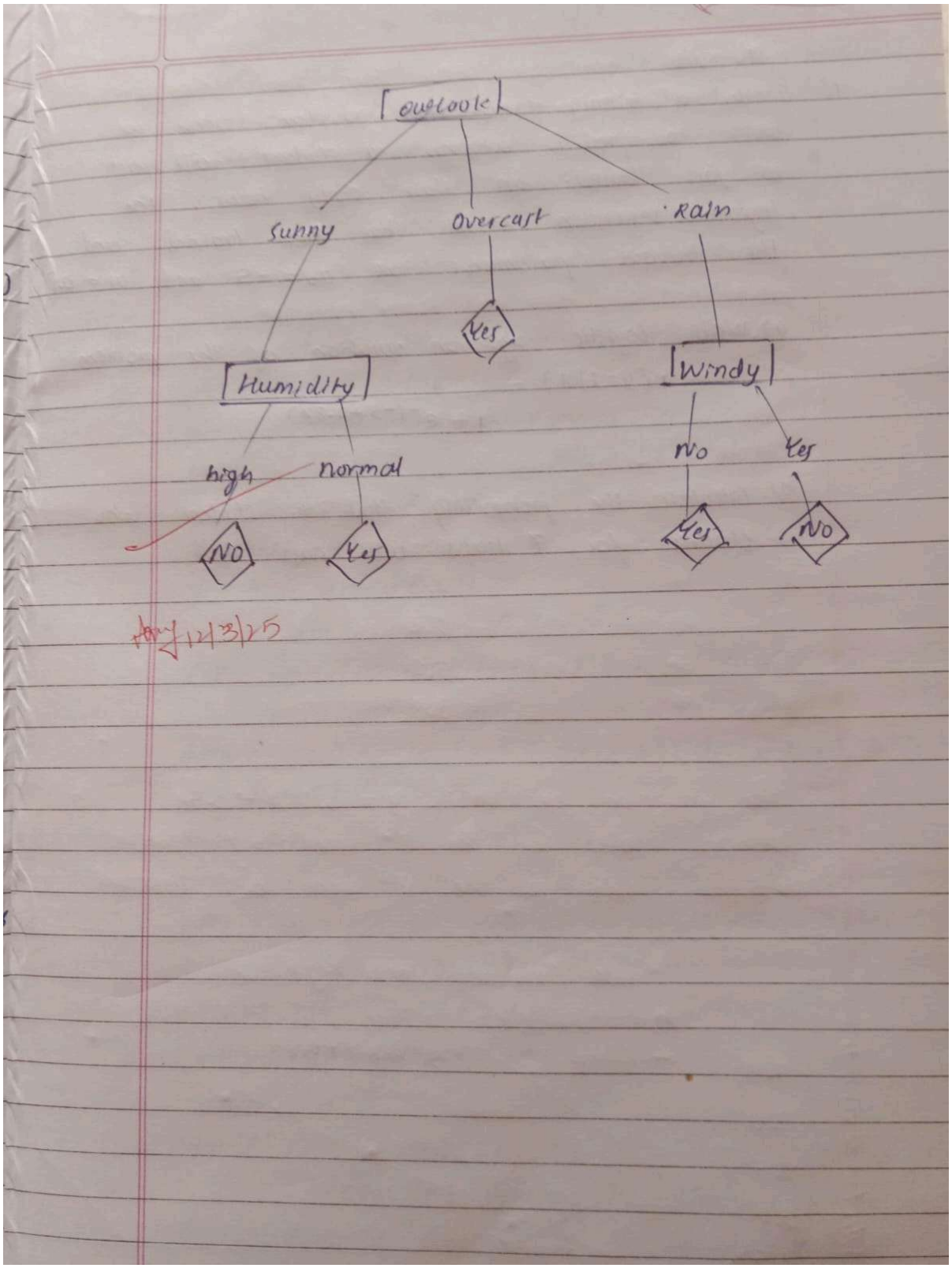
Normal → Yes

Overcast → Yes

Rain → Windy:

No → Yes

Yes → No



Code:

```
import pandas as pd
```

```
import numpy as np



# Sample weather dataset

data = {

    'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rain', 'Rain', 'Rain',
'Overcast', 'Sunny', 'Sunny', 'Rain', 'Sunny', 'Overcast', 'Overcast',
'Rain'],

    'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Cool',
'Mild', 'Cool', 'Mild', 'Mild', 'Mild', 'Hot', 'Mild'],

    'Humidity': ['High', 'High', 'High', 'High', 'Normal', 'Normal',
'Normal', 'High', 'Normal', 'Normal', 'Normal', 'High', 'Normal', 'High'],

    'Windy': ['No', 'Yes', 'No', 'No', 'No', 'Yes', 'Yes', 'No', 'No', 'No',
'Yes', 'Yes', 'No', 'Yes'],

    'Play Tennis?': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No',
'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']

}



# Convert to DataFrame

df = pd.DataFrame(data)



# Function to calculate entropy

def entropy(target):

    # Get the counts of each class

    class_counts = target.value_counts()

    # Calculate the entropy using the formula
```

```
    probabilities = class_counts / len(target)
    return -np.sum(probabilities * np.log2(probabilities))

# Function to calculate information gain
def information_gain(data, feature, target):
    # Calculate the entropy of the whole dataset
    entropy_before = entropy(target)

    # Get the unique values of the feature
    feature_values = data[feature].unique()

    # Calculate the weighted entropy after the split
    weighted_entropy = 0
    for value in feature_values:
        subset = target[data[feature] == value]
        weighted_entropy += (len(subset) / len(target)) * entropy(subset)

    # Information gain is the reduction in entropy
    return entropy_before - weighted_entropy

# Function to print entropy and information gain for each feature
def print_entropy_and_gain(data, features, target):
```

```
    print("\nEntropy and Information Gain for each feature:")

    for feature in features:

        gain = information_gain(data, feature, target)

        ent = entropy(target)

        print(f"Feature: {feature} | Entropy: {ent:.4f} | Information Gain:
{gain:.4f}")



# Function to build the decision tree recursively

def build_tree(data, target, features):

    # Base case: If all target values are the same, return a leaf node

    if len(target.unique()) == 1:

        return target.iloc[0]



    # Base case: If no features left to split, return the majority class

    if len(features) == 0:

        return target.mode()[0]



    # Calculate information gain for each feature

    gains = {feature: information_gain(data, feature, target) for feature in
features}



    # Find the feature with the highest information gain

    best_feature = max(gains, key=gains.get)
```

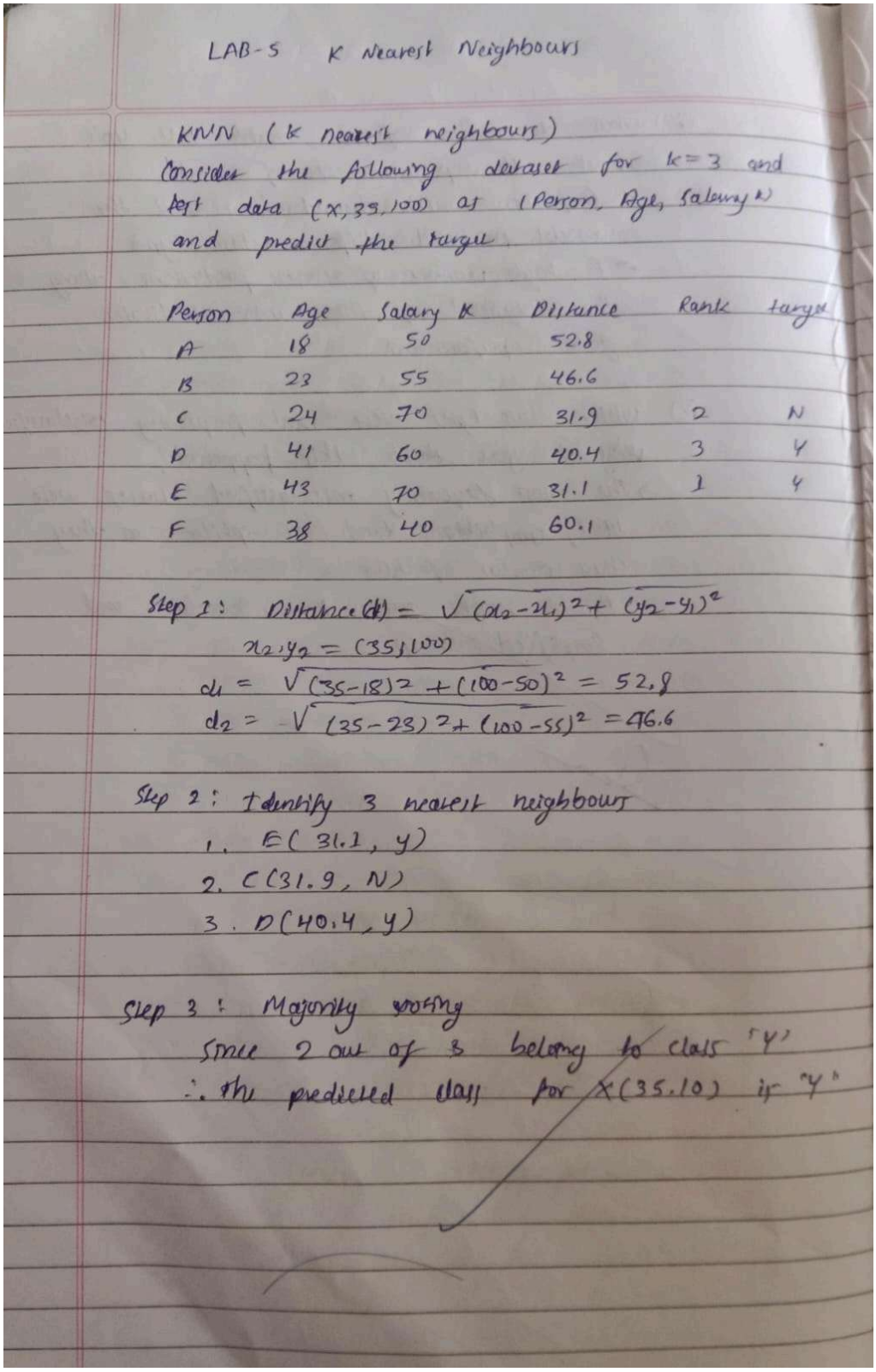
```
    # Create the tree node with the best feature
    tree = {best_feature: {}}

    # Get the unique values of the best feature
    feature_values = data[best_feature].unique()

    # Recursively build the tree for each subset of the data
    for value in feature_values:
        subset_data = data[data[best_feature] == value]
        subset_target = target[data[best_feature] == value]

        # Remove the best feature from the list of features for the next
level
        remaining_features = [f for f in features if f != best_feature]

        # Build the subtree for the subset
        subtree = build_tree(subset_data, subset_target, remaining_features)

        # Add the subtree to the tree
        tree[best_feature][value] = subtree
```

```
    return tree


# Function to print the tree in a visually structured way
def print_tree(tree, indent=""):
    if isinstance(tree, dict):
        for feature, branches in tree.items():
            print(f"{indent}{feature}:")
            for value, subtree in branches.items():
                print(f"{indent}  {value} ->", end=" ")
                print_tree(subtree, indent + "    ")
    else:
        print(f"{indent}{tree}")


# Target variable
target = df['Play Tennis?']


# Features
features = ['Outlook', 'Temperature', 'Humidity', 'Windy']


# Step 1: Print entropy and information gain for each feature
print_entropy_and_gain(df, features, target)
```

```
# Step 2: Build the decision tree

tree = build_tree(df, target, features)



# Step 3: Print the decision tree (formatted)

print("\nDecision Tree:")

print_tree(tree, indent="    ")
```

Program 6

Screenshot:



LAB-5 K Nearest Neighbours

KNN (K nearest neighbours)
Consider the following dataset for k=3 and test data (x, 35, 100) as (Person, Age, Salary)
and predict the target.

Person	Age	Salary	K	Distance	Rank	target
A	18	50		52.8		
B	23	55		46.6		
C	24	70		31.9	2	N
D	41	60		40.4	3	Y
E	43	70		31.1	1	Y
F	38	40		60.1		

Step 1: Distance(d) = $\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$

$x_2, y_2 = (35, 100)$

$d_1 = \sqrt{(35-18)^2+(100-50)^2} = 52.8$

$d_2 = \sqrt{(35-23)^2+(100-55)^2} = 46.6$

Step 2: Identify 3 nearest neighbours

1. E(31.1, Y)
2. C(31.9, N)
3. D(40.4, Y)

Step 3: Majority voting

Since 2 out of 3 belong to class 'Y'

∴ the predicted class for X(35,10) is 'Y'

For iris dataset
How to choose the k value? Demonstrate using accuracy rate and error rate

steps to choose k using accuracy rate & error rate

1) Split the dataset (training (70%), testing (30%))
2) Train KNN with different k value $k = \{1, 3, 5 \ldots\}$
3) Calculate accuracy
4) $\text{Accuracy} = \frac{\text{Correct predictions}}{\text{Total predictions}} \times 100$
4) Calculate error rate = Error rate = 1 - accuracy
5) Plot accuracy vs k

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

→. Purpose of feature scaling

1) Diabetes dataset has features like glucose level, BMI, and age which have different ranges
2) Machine learning algorithms perform better when features are on similar scale
3) It improves convergence speed
4) Prevent dominant features from biasing the model

→ ~~Method~~.

→ Methods to perform feature scaling

1) Min-Max scaling

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

2) Standardization

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

09.04.2024

Code:

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report



# Load the Iris dataset

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

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

y_iris = iris['species']



# Split the dataset into training and testing sets

X_train_iris, X_test_iris, y_train_iris, y_test_iris =
train_test_split(X_iris, y_iris, test_size=0.2, random_state=42)



# Choose the best k value (testing k from 1 to 20)

k_values = range(1, 21)

accuracy_list = []
```

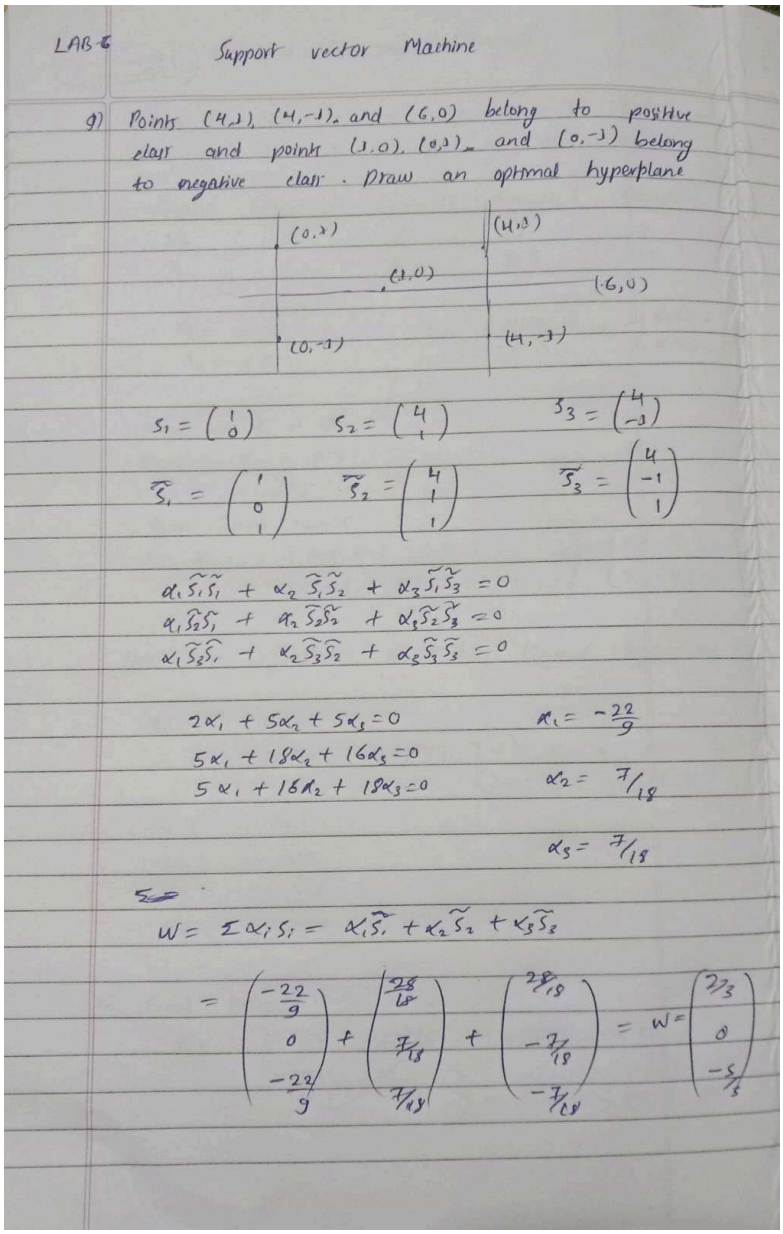
```
for k in k_values:

    knn = KNeighborsClassifier(n_neighbors=k)

    knn.fit(X_train_iris, y_train_iris)

    y_pred = knn.predict(X_test_iris)

    accuracy_list.append(accuracy_score(y_test_iris, y_pred))



best_k_iris = k_values[np.argmax(accuracy_list)]

print(f"Optimal k value for Iris dataset: {best_k_iris}")



# Train KNN with the best k value

knn_iris = KNeighborsClassifier(n_neighbors=best_k_iris)

knn_iris.fit(X_train_iris, y_train_iris)

y_pred_iris = knn_iris.predict(X_test_iris)



# Display Accuracy and Confusion Matrix for Iris dataset

print("Iris Dataset Accuracy:", accuracy_score(y_test_iris, y_pred_iris))

print("Confusion Matrix for Iris:")

print(confusion_matrix(y_test_iris, y_pred_iris))

print("Classification Report for Iris:")

print(classification_report(y_test_iris, y_pred_iris))



# Load the Diabetes dataset
```

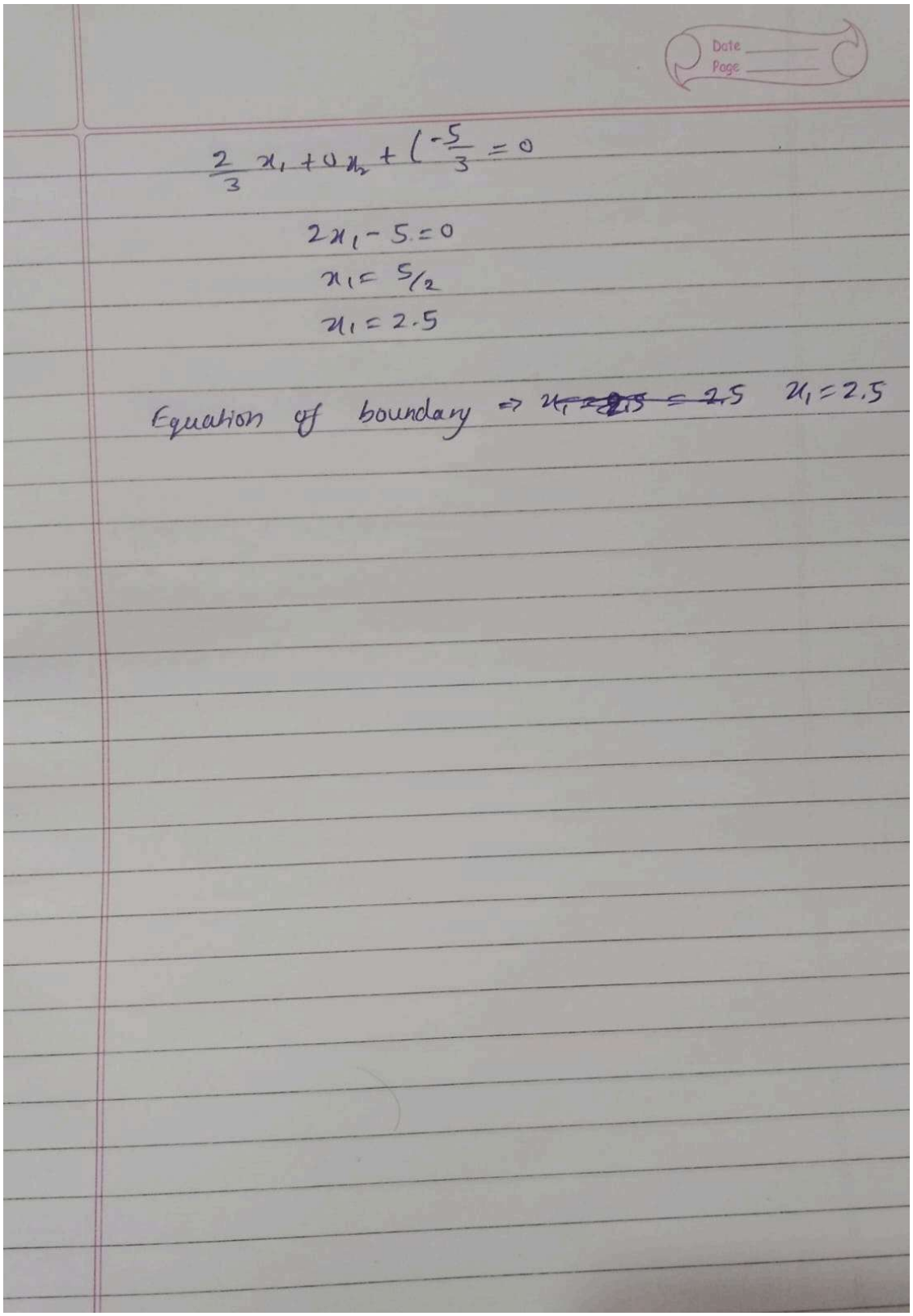
```
diabetes = pd.read_csv("/content/diabetes.csv")

X_diabetes = diabetes.drop(columns=['Outcome'])

y_diabetes = diabetes['Outcome']



# Split into training and testing

X_train_diabetes, X_test_diabetes, y_train_diabetes, y_test_diabetes =
train_test_split(X_diabetes, y_diabetes, test_size=0.2, random_state=42)



# Feature Scaling

scaler = StandardScaler()

X_train_diabetes = scaler.fit_transform(X_train_diabetes)

X_test_diabetes = scaler.transform(X_test_diabetes)



# Train KNN Classifier for Diabetes dataset

best_k_diabetes = 7  # Assume 7 as a reasonable k value (can be tuned
further)

knn_diabetes = KNeighborsClassifier(n_neighbors=best_k_diabetes)

knn_diabetes.fit(X_train_diabetes, y_train_diabetes)

y_pred_diabetes = knn_diabetes.predict(X_test_diabetes)



# Display Accuracy and Confusion Matrix for Diabetes dataset

print("Diabetes Dataset Accuracy:", accuracy_score(y_test_diabetes,
y_pred_diabetes))

print("Confusion Matrix for Diabetes:")
```

```
print(confusion_matrix(y_test_diabetes, y_pred_diabetes))

print("Classification Report for Diabetes:")

print(classification_report(y_test_diabetes, y_pred_diabetes))
```

Program 7

Screenshot:





Code:

```
import numpy as np
```

```
import matplotlib.pyplot as plt



class SVM:

    def __init__(self, learning_rate=0.001, lambda_param=0.01, n_iters=1000):

        self.lr = learning_rate

        self.lambda_param = lambda_param

        self.n_iters = n_iters

        self.w = None

        self.b = None



    def fit(self, X, y):

        y = np.where(y <= 0, -1, 1) # Convert labels to -1 and 1

        n_samples, n_features = X.shape

        self.w = np.zeros(n_features)

        self.b = 0



        for _ in range(self.n_iters):

            for idx, x_i in enumerate(X):

                condition = y[idx] * (np.dot(x_i, self.w) + self.b) >= 1

                if condition:

                    self.w -= self.lr * (2 * self.lambda_param * self.w)

                else:
```

```
                    self.w -= self.lr * (2 * self.lambda_param * self.w -
np.dot(x_i, y[idx]))

                    self.b += self.lr * y[idx]



    def predict(self, X):

        approx = np.dot(X, self.w) + self.b

        return np.sign(approx)



    def visualize(self, X, y, new_point=None, prediction=None):

        def get_hyperplane(x, w, b, offset):

            return (-w[0] * x + b + offset) / w[1]



        fig = plt.figure()

        ax = fig.add_subplot(1, 1, 1)



        # Plot existing data points

        for i, sample in enumerate(X):

            if y[i] == 1:

                plt.scatter(sample[0], sample[1], marker='o', color='blue',
label='Class +1' if i == 0 else "")

            else:

                plt.scatter(sample[0], sample[1], marker='x', color='red',
label='Class -1' if i == 0 else "")
```

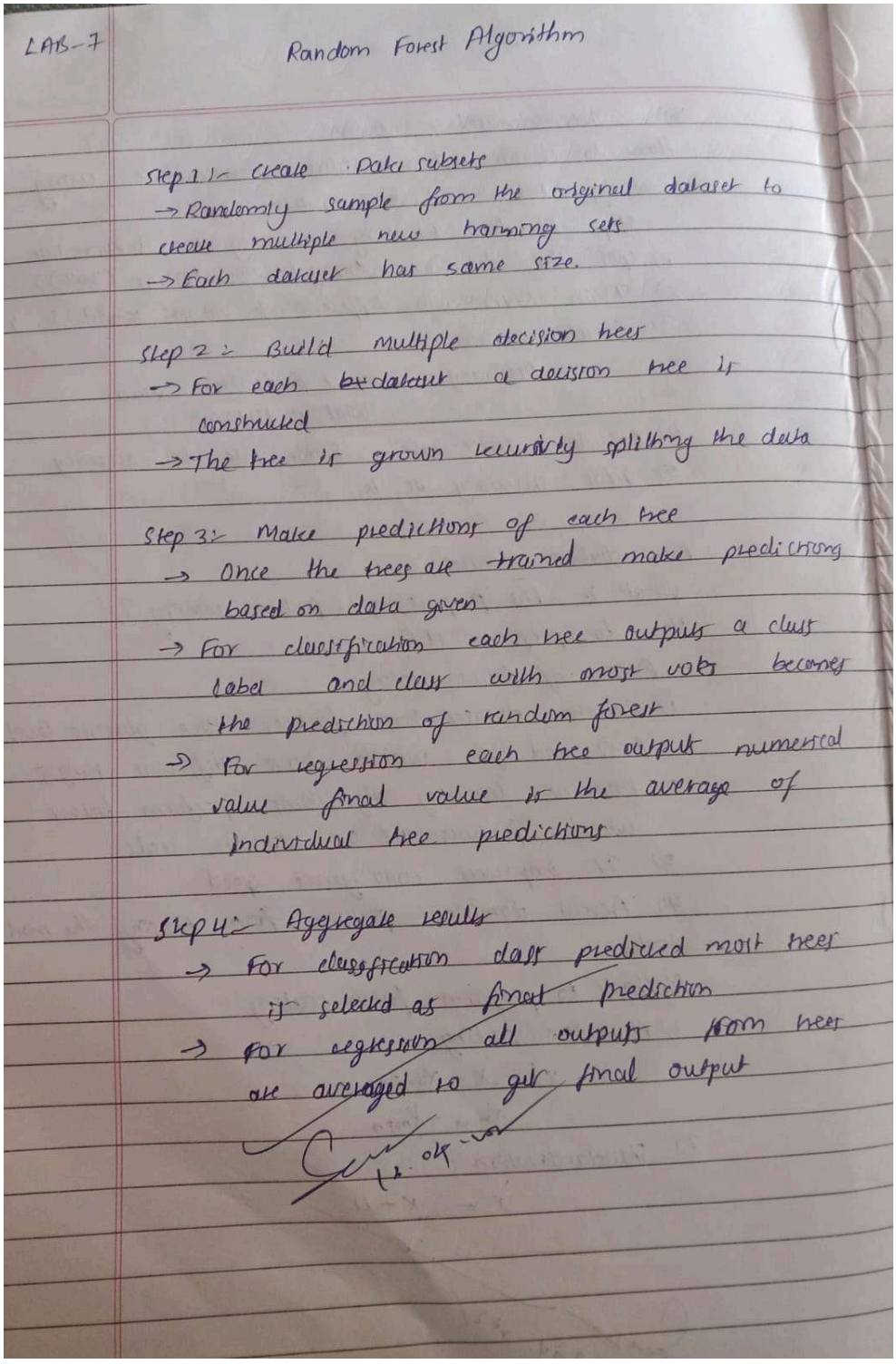
```
        # Plot decision boundary
        x0 = np.linspace(np.min(X[:, 0])-1, np.max(X[:, 0])+1, 100)
        x1 = get_hyperplane(x0, self.w, self.b, 0)
        x1_m = get_hyperplane(x0, self.w, self.b, -1)
        x1_p = get_hyperplane(x0, self.w, self.b, 1)

        ax.plot(x0, x1, 'k-', label='Decision Boundary')
        ax.plot(x0, x1_m, 'k--', label='Margins')
        ax.plot(x0, x1_p, 'k--')

        # Plot the new point
        if new_point is not None:
            color = 'green' if prediction == 1 else 'orange'
            label = f'New Point: Class {"1" if prediction == 1 else "0"}'
            plt.scatter(new_point[0], new_point[1], c=color, s=100,
edgecolors='black', label=label, marker='*')

        ax.legend()
        plt.xlabel("Feature 1")
        plt.ylabel("Feature 2")
        plt.title("SVM with New Point Prediction")
        plt.grid(True)
```

```
        plt.show()




if __name__ == "__main__":
    # Training data
    X = np.array([
        [1, 0],
        [0, 1],
        [0, -1],
        [4, -1],
        [4, 1],
        [6, 0]
    ])
    y = np.array([0, 0, 0, 1, 1, 1]) # 0 -> -1, 1 -> +1


    # New point to classify
    new_point = np.array([[5, 5]])


    # Train and predict
    svm = SVM()
    svm.fit(X, y)
```

```
    prediction = svm.predict(new_point)[0]



    # Visualize

    svm.visualize(X, y, new_point=new_point[0], prediction=prediction)



    # Print prediction

    print(f"New point {new_point[0]} classified as: {'Class 1' if prediction
== 1 else 'Class 0'}")
```

Program 8

Screenshot:



LAB-7

Random Forest Algorithm

Step 1:- Create Data subsets
→ Randomly sample from the original dataset to create multiple new training sets
→ Each dataset has same size.

Step 2:- Build multiple decision trees
→ For each ~~be~~ dataset a decision tree is constructed
→ The tree is grown recursively splitting the data

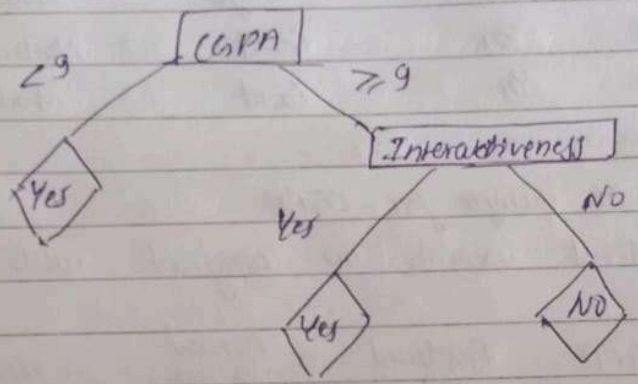
Step 3:- Make predictions of each tree
→ Once the trees are trained make predictions based on data given
→ For classification each tree outputs a class label and class with most vote becomes the prediction of random forest
→ For regression each tree output numerical value final value is the average of individual tree predictions

Step 4:- Aggregate results
→ For classification class predicted most trees is selected as final prediction
→ For regression all outputs from trees are averaged to get final output

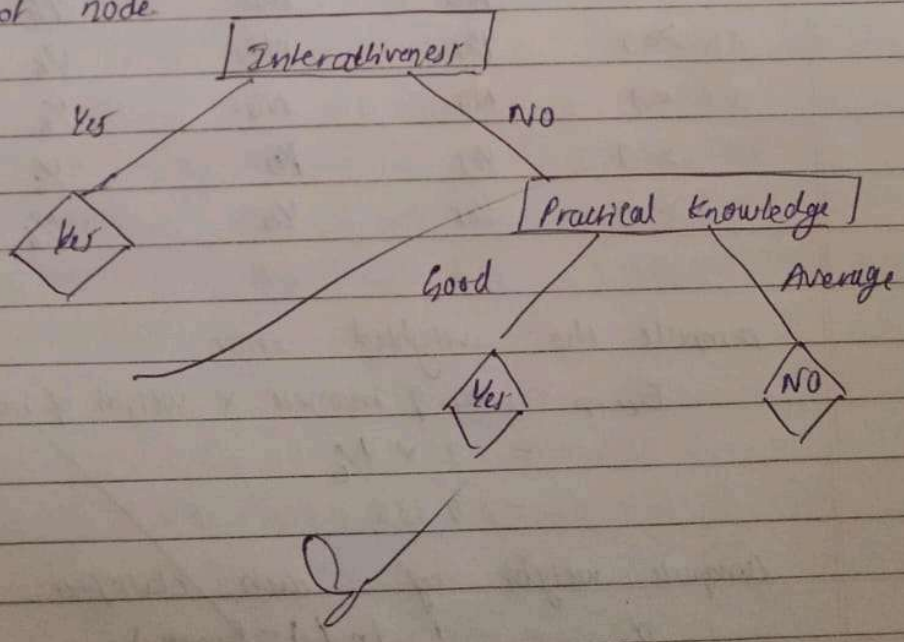
ok

Draw decision tree considering CGPA as root node

SNo	CGPA	Interactiveness	Communication	Practical knowledge	Job offer
1	≥9	Yes	Good	Good	Yes
2	<9	No	Moderate	Good	Yes
3	≥9	No	Moderate	Average	No
4	≥9	No	Moderate	Average	No
5	≥9	Yes	Moderate	Good	Yes



Draw decision tree considering interactiveness as root node.



Code:

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score, confusion_matrix

import matplotlib.pyplot as plt

import seaborn as sns



# Load dataset

df = pd.read_csv("/content/iris (4).csv")



# Features and target

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=0.2,
random_state=42)



# ------------------------------------------

# 1. Default Random Forest with 10 estimators

# ------------------------------------------

clf_default = RandomForestClassifier(n_estimators=10, random_state=42)
```

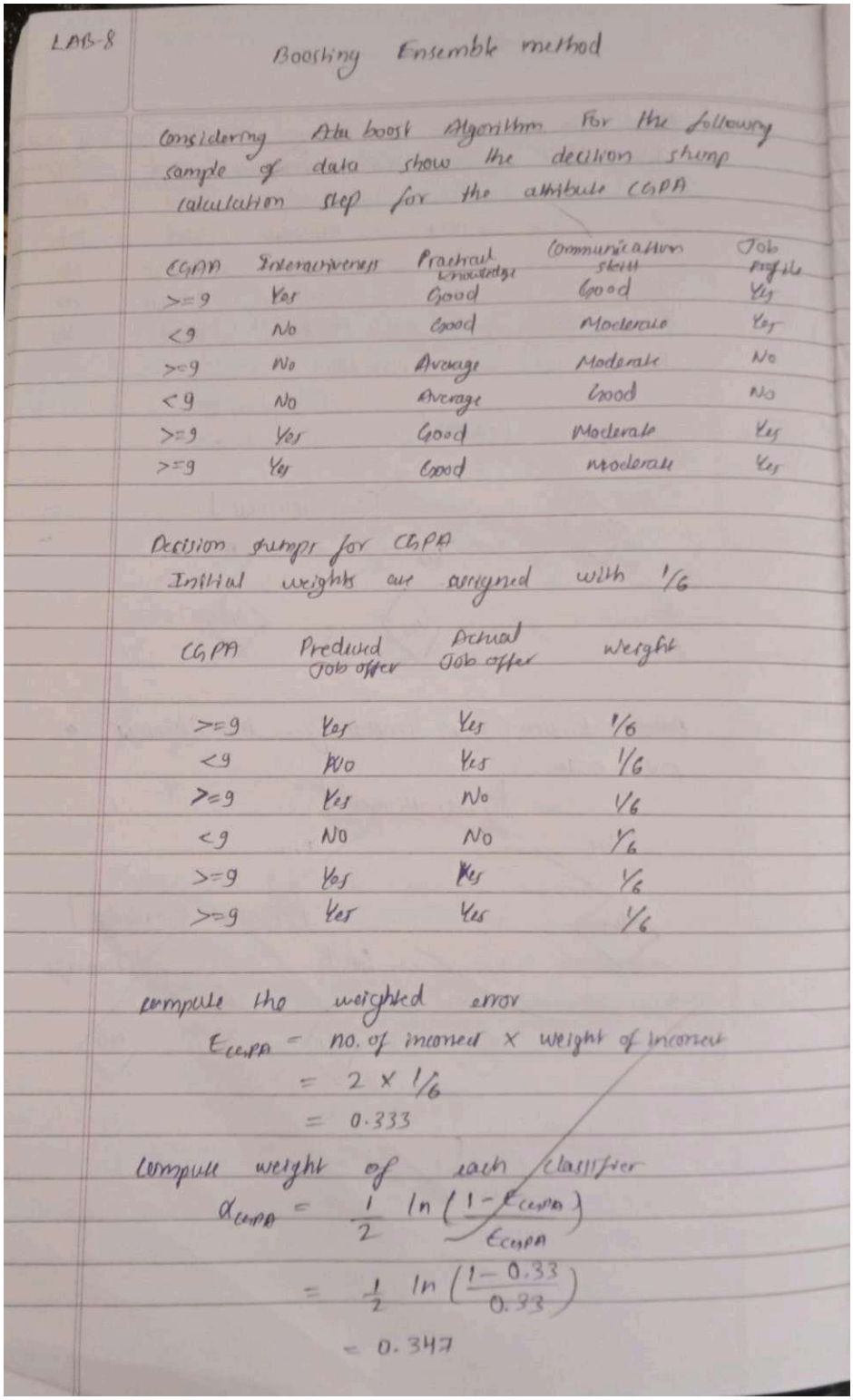
```
clf_default.fit(X_train, y_train)

y_pred_default = clf_default.predict(X_test)

default_accuracy = accuracy_score(y_test, y_pred_default)

default_cm = confusion_matrix(y_test, y_pred_default)



print("=== Default Model (10 trees) ===")

print(f"Accuracy: {default_accuracy:.4f}")

print("Confusion Matrix:")

print(default_cm)



# ------------------------------------------

# 2. Fine-tune number of trees for best accuracy

# ------------------------------------------

best_score = 0

best_n = 0

best_model = None

best_cm = None



scores = []

n_estimators_range = range(1, 101)



for n in n_estimators_range:
```

```
    clf = RandomForestClassifier(n_estimators=n, random_state=42)
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    score = accuracy_score(y_test, y_pred)
    scores.append(score)


    if score > best_score:
        best_score = score
        best_n = n
        best_model = clf
        best_cm = confusion_matrix(y_test, y_pred)


# Print best result
print("\n=== Tuned Model ===")
print(f"Best Accuracy: {best_score:.4f}")
print(f"Best Number of Trees: {best_n}")
print("Confusion Matrix:")
print(best_cm)


# Plot accuracy vs number of trees
plt.figure(figsize=(8, 5))
plt.plot(n_estimators_range, scores, marker='o')
```

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

plt.ylabel("Accuracy")

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

plt.grid(True)

plt.show()



# Plot best confusion matrix

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

sns.heatmap(best_cm, annot=True, fmt="d", cmap="Blues",
xticklabels=y.unique(), yticklabels=y.unique())

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.title("Best Confusion Matrix")

plt.show()
```

Program 9

Screenshot:



LAB-8

Boosting Ensemble method

Considering Ada boost Algorithm for the following sample of data show the decision stump calculation step for the attribute CGPA

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

Decision stumps for CGPA

Initial weights are assigned with 1/6

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

compute the weighted error

E_{CGPA} = no. of incorrect x weight of incorrect

= 2 x 1/6

= 0.333

compute weight of each classifier

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

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

= 0.347

Calculate the normalizing factor

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

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

$Z_{CGPA} = 0.9428$

Update the weight of all data

$Wt(d_j)_{i+1} = \frac{Wt(d_i)_{CGPA}\ of\ correct\ instance \times e^{-\alpha_{CGPA}}}{Z_{CGPA}}$

$Wt(d_i)_{i+1} = \frac{\frac{1}{6} \times e^{-0.340}}{0.9428} = 0.1249$

$Wt(d_i)_{i+1} = \frac{Wt(d_i)_{CGPA}\ of\ incorrect\ instance \times e^{\alpha_{CGPA}}}{Z_{CGPA}}$

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

$= 0.2501$

CGPA	Predicted Job offer	Actual Job offer	Weight
>=9	Yes	Yes	0.1249
<9	No	Yes	0.2501
>=9	Yes	No	0.2501
<9	No	No	0.1249
>=9	Yes	Yes	0.1249
>=9	Yes	Yes	0.1249

Code:

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.ensemble import AdaBoostClassifier

from sklearn.metrics import accuracy_score, confusion_matrix

import matplotlib.pyplot as plt

import seaborn as sns



# Load dataset

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



# Feature and target split

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

y = df.iloc[:, -1]



# Handle categorical variables (if any)

X = pd.get_dummies(X)

y = pd.factorize(y)[0]



# Train/test split

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

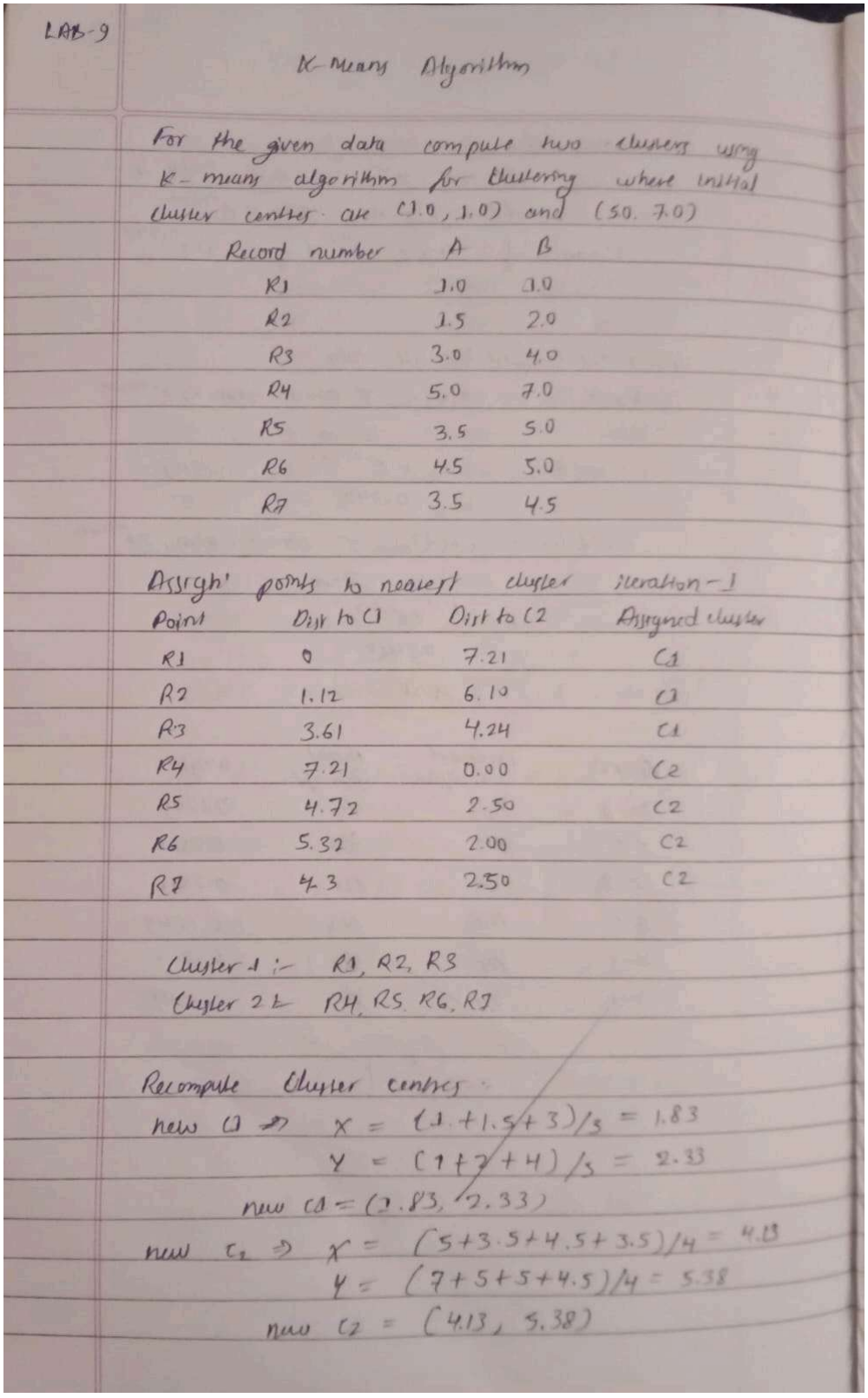
```
# ------------------------

# 1. Default AdaBoost model (10 estimators)

# ------------------------

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

clf_default.fit(X_train, y_train)

y_pred_default = clf_default.predict(X_test)

default_accuracy = accuracy_score(y_test, y_pred_default)

default_cm = confusion_matrix(y_test, y_pred_default)



print("=== Default AdaBoost Model (10 estimators) ===")

print(f"Accuracy: {default_accuracy:.4f}")

print("Confusion Matrix:")

print(default_cm)



# ------------------------

# 2. Fine-tune n_estimators

# ------------------------

best_score = 0

best_n = 0

best_cm = None

scores = []
```

```
for n in range(1, 101):
    clf = AdaBoostClassifier(n_estimators=n, random_state=42)
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    score = accuracy_score(y_test, y_pred)
    scores.append(score)

    if score > best_score:
        best_score = score
        best_n = n
        best_cm = confusion_matrix(y_test, y_pred)

print("\n=== Best AdaBoost Model ===")
print(f"Best Accuracy: {best_score:.4f}")
print(f"Best Number of Estimators: {best_n}")
print("Confusion Matrix:")
print(best_cm)

# Plot accuracy vs number of estimators
plt.figure(figsize=(8, 5))
plt.plot(range(1, 101), scores, marker='o')
```

```
plt.xlabel("Number of Estimators")

plt.ylabel("Accuracy")

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

plt.grid(True)

plt.show()



# Plot best confusion matrix

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

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

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.title("Best Confusion Matrix")

plt.show()
```

Program 10

Screenshot:



LAB-9

K-Means Algorithm

For the given data compute two clusters using K-means algorithm for clustering where initial cluster centres are (1.0, 1.0) and (5.0, 7.0)

Record number	A	B
R1	1.0	1.0
R2	1.5	2.0
R3	3.0	4.0
R4	5.0	7.0
R5	3.5	5.0
R6	4.5	5.0
R7	3.5	4.5

Assign points to nearest cluster iteration-1

Point	Dist to C1	Dist to C2	Assigned cluster
R1	0	7.21	C1
R2	1.12	6.10	C1
R3	3.61	4.24	C1
R4	7.21	0.00	C2
R5	4.72	2.50	C2
R6	5.32	2.00	C2
R7	4.3	2.50	C2

Cluster 1 :- R1, R2, R3

Cluster 2 :- R4, R5, R6, R7

Recompute cluster centres:

new C1 → $x = (1 + 1.5 + 3)/3 = 1.83$

$y = (1 + 2 + 4)/3 = 2.33$

new C1 = (1.83, 2.33)

new C_2 → $x = (5 + 3.5 + 4.5 + 3.5)/4 = 4.13$

$y = (7 + 5 + 5 + 4.5)/4 = 5.38$

new C2 = (4.13, 5.38)

Assign points to nearest clusters Iteration - 2

Point	Dist to C1	Dist to C2	Assigned cluster
R1	1.87	5.62	C1
R2	0.47	4.53	C1
R3	2.03	1.92	C2
R4	5.67	1.89	C2
R5	2.63	0.71	C2
R6	3.25	0.47	C2
R7	2.73	0.94	C2

Cluster 1 :- R1, R2

Cluster 2 :- R3, R4, R5, R6, R7

Final cluster assignments

C1 centre = (1.25, 1.5)

C2 centre = (3.9, 5.1)

Code:

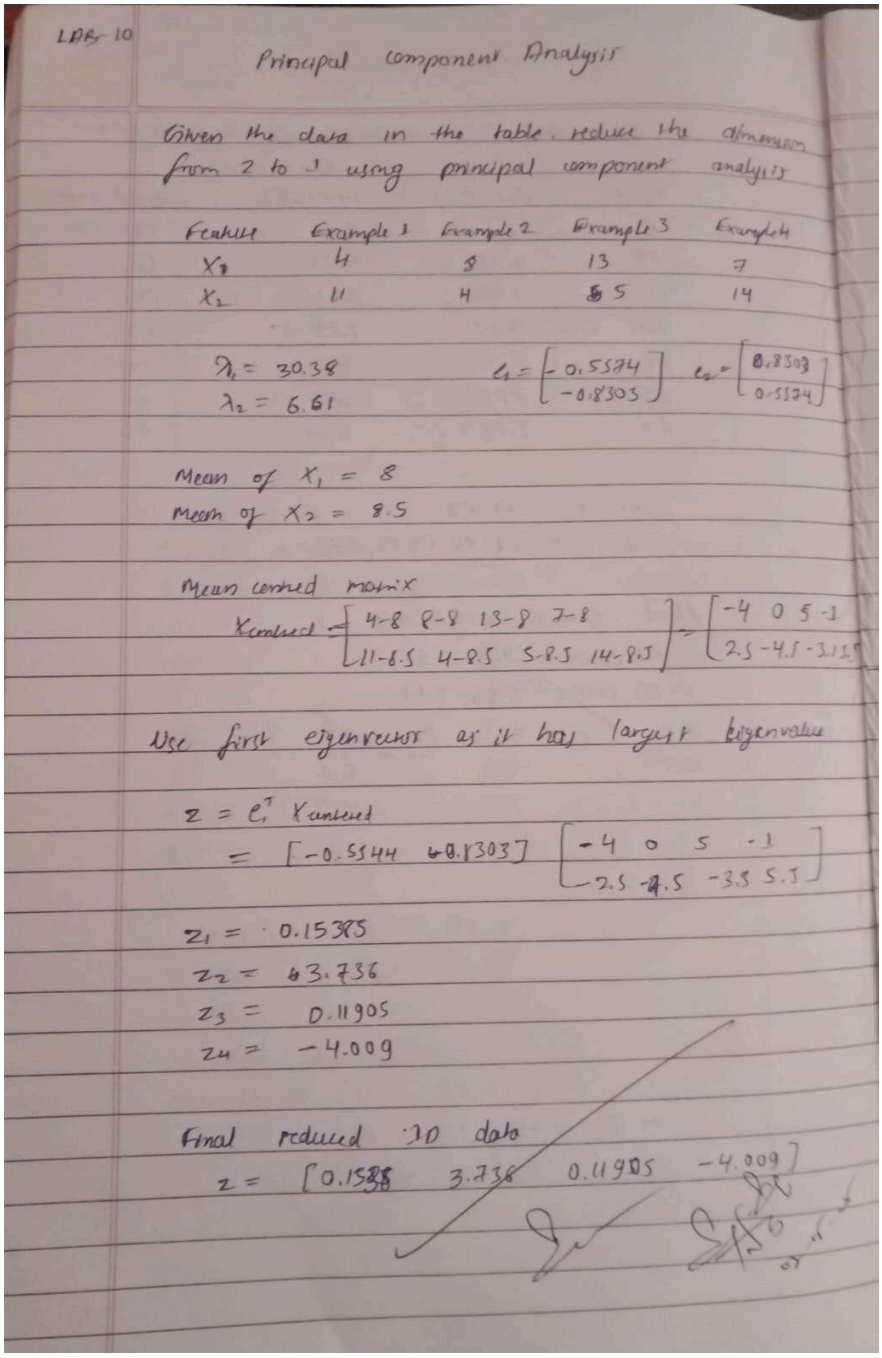
```
import pandas as pd

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

import numpy as np



# Load dataset

df = pd.read_csv("iris (4).csv")  # Replace with your actual filename if
needed



# Use only petal length and petal width

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



# Feature scaling

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)



# Elbow method to find optimal K

inertia = []

k_range = range(1, 11)

for k in k_range:

    kmeans = KMeans(n_clusters=k, random_state=42)
```

```
    kmeans.fit(X_scaled)

    inertia.append(kmeans.inertia_)



# Automatically determine the elbow point (optional)

diff = np.diff(inertia)

diff_r = np.diff(diff)

optimal_k = np.argwhere(diff_r > -0.1)[0][0] + 2  # add 2 due to second
derivative shift



# Plot elbow graph with optimal K marked

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

plt.plot(k_range, inertia, 'bo-')

plt.axvline(x=optimal_k, color='red', linestyle='--', label=f'Optimal k =
{optimal_k}')

plt.xlabel('Number of Clusters (k)')

plt.ylabel('Inertia')

plt.title('Elbow Method For Optimal k')

plt.legend()

plt.grid(True)

plt.tight_layout()

plt.show()



# Output the optimal k
```

```
print(f"Optimal number of clusters (k): {optimal_k}")
```

Program 11

Screenshot:



Code:

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

from sklearn.svm import SVC

from sklearn.linear_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score




df = pd.read_csv("heart (1).csv")




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

df = pd.get_dummies(df, columns=categorical_cols, drop_first=True)




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

y = df["HeartDisease"]
```

```
scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)



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



models = {

    "SVM": SVC(),

    "Logistic Regression": LogisticRegression(max_iter=1000),

    "Random Forest": RandomForestClassifier()

}



accuracy_before_pca = {}

for name, model in models.items():

    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)

    accuracy_before_pca[name] = accuracy_score(y_test, y_pred)



pca = PCA(n_components=0.95)

X_pca = pca.fit_transform(X_scaled)



X_train_pca, X_test_pca, y_train_pca, y_test_pca = train_test_split(X_pca, y,
```

```
test_size=0.2, random_state=42)



accuracy_after_pca = {}

for name, model in models.items():

    model.fit(X_train_pca, y_train_pca)

    y_pred_pca = model.predict(X_test_pca)

    accuracy_after_pca[name] = accuracy_score(y_test_pca, y_pred_pca)



print(" Accuracy BEFORE PCA:")

for name, acc in accuracy_before_pca.items():

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



print("\nAccuracy AFTER PCA:")

for name, acc in accuracy_after_pca.items():

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



print(f"\nOriginal features: {X.shape[1]}")

print(f"Features after PCA: {X_pca.shape[1]}")
```