VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT on

Machine Learning (23CS6PCMAL)

Submitted by

Shree Varna M (1BM23CS263)

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 **Shree Varna M (1BM23CS263),** 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.

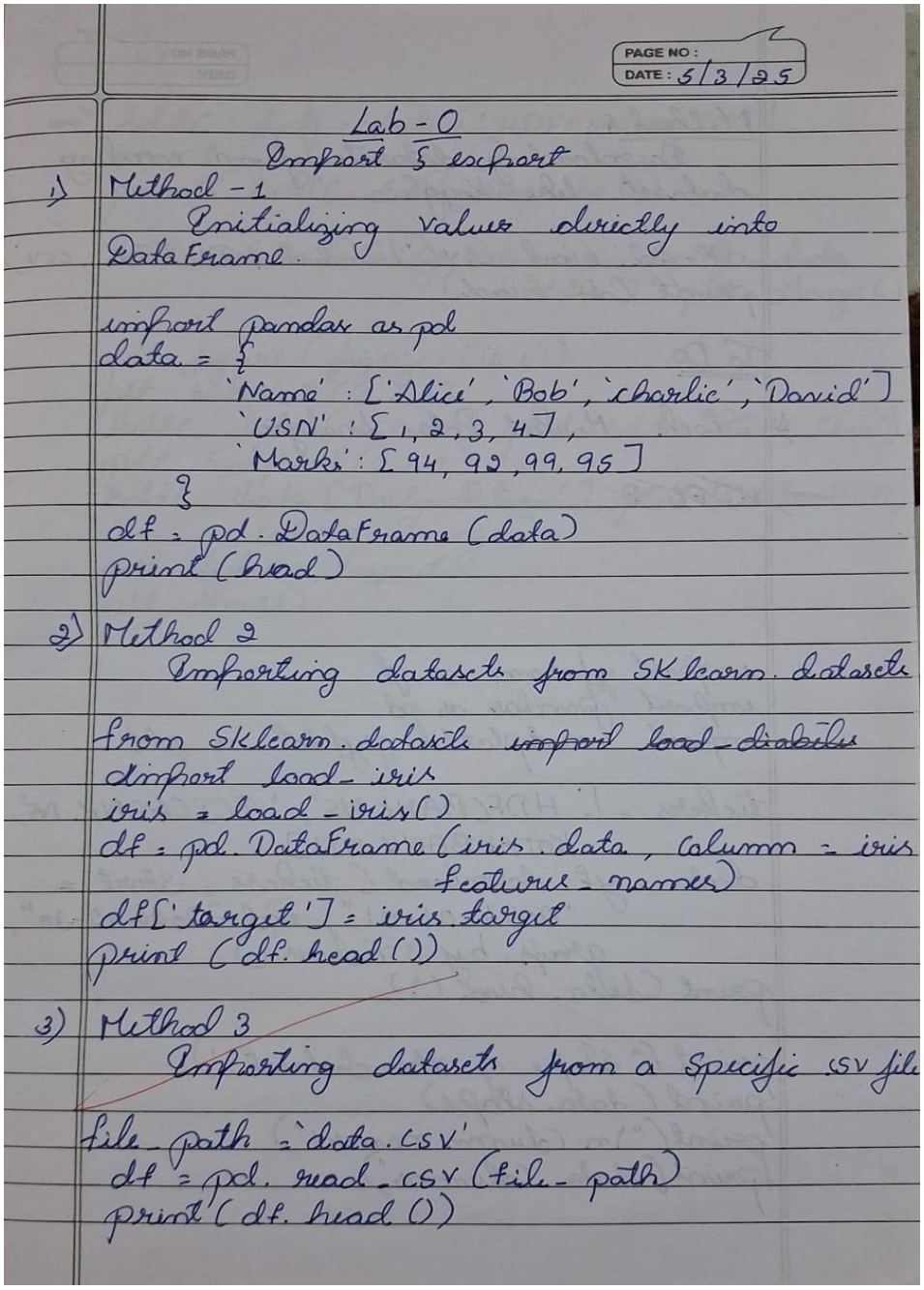
Lab Faculty Incharge Name: **Ms. Sheetal A V** Assistant Professor Department of CSE, BMSCE	 **Dr. Kavitha Sooda** Professor & HOD Department of CSE, BMSCE

Index

Github Link: https://github.com/Shree-varna/ML_lab.git

Program 1

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



PAGE NO :
DATE : 5/3/25

Lab - 0
Import & export

1) Method - 1
Initializing values directly into DataFrame.

```
import pandas as pd
data = {
    'Name' : ['Alice', 'Bob', 'charlie', 'David']
    'USN' : [1, 2, 3, 4],
    'Marks' : [94, 92, 99, 95]
}
df = pd.DataFrame(data)
print(head)
```

2) Method 2
Importing datasets from SKlearn.datasets

```
From SKlearn.datasets import load_diabetes
import load_iris
iris = load_iris()
df = pd.DataFrame(iris.data, column = iris
                  feature_names)
df['target'] = iris.target
print(df.head())
```

3) Method 3
Importing datasets from a specific csv file

```
file_path = 'data.csv'
df = pd.read_csv(file_path)
print(df.head())
```

PAGE NO :
DATE :

Method 4:
Downloading datasets from existing dataset like Kaggle.

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

To Do

1) Stock Market Data Analysis

→ ~~HDFC B~~

```
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 = 'tickers')
print(data.head())

print('Shape of the dataset:')
print(data.shape)
print("\n Column names:")
print("data.Columns")
```

PAGE NO :
DATE :

```
hdfc_data = data["HDFCBANK.NS"]
print("Summary")
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_Clos")
plt.subplot(2,1,2)
hdfc_data['Daily_Return'].plot(title=HDFC,
        color='orange')
plt.tight_layout
plt.show()
```

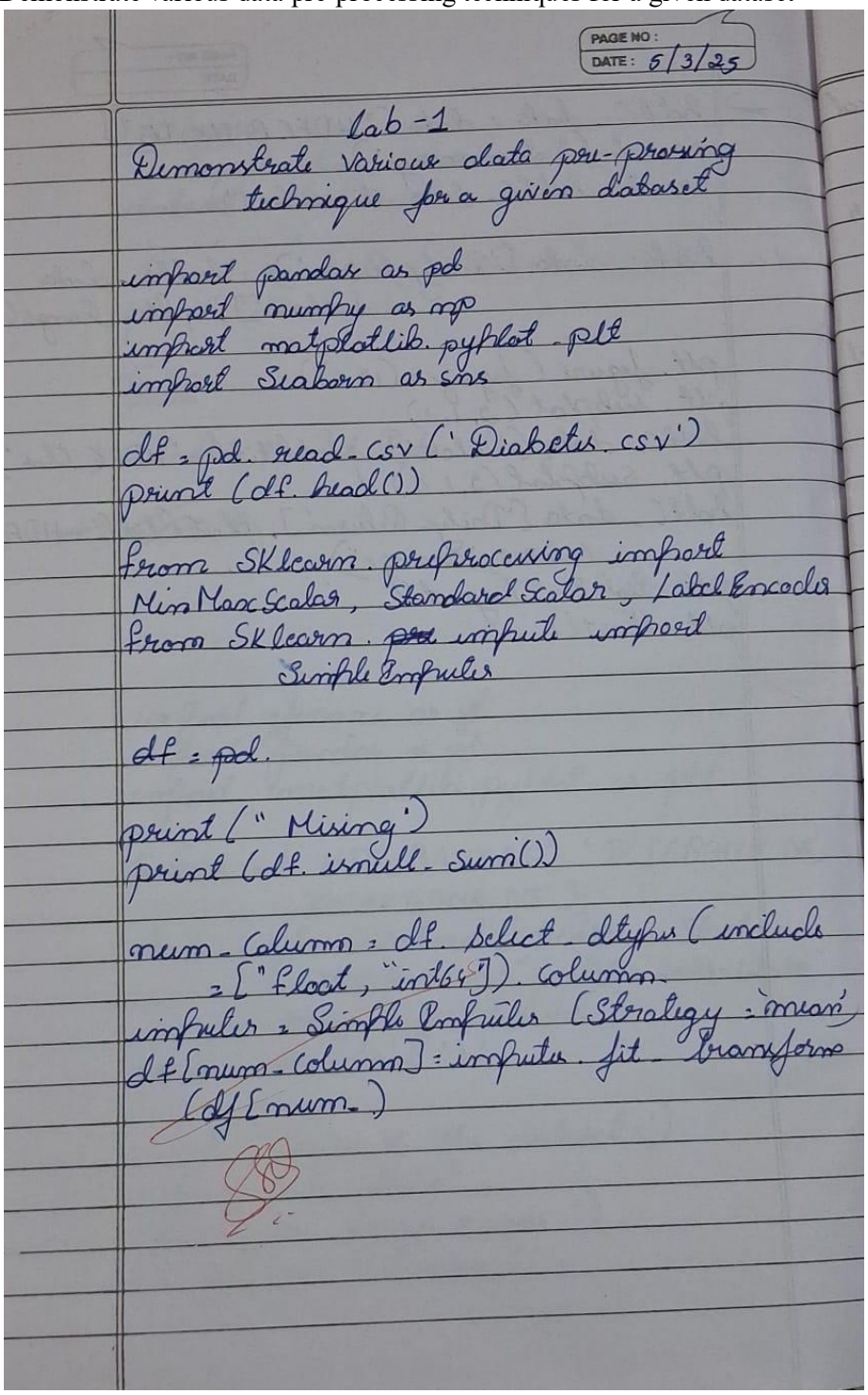
Code:

```
import pandas as pd data={
   'USN':['1','2','3','4','5'],
   'Name':['Arun', 'Soudarya','neha', 'suraj', 'Prajju'],
   'marks':[50, 80, 45, 99, 87]
}
df=pd.DataFrame(data) print(df)

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


file_path='/sample_sales_data (1).csv'
df=pd.read_csv(file_path) print("Sample
data:")
print(df)


file='Dataset of Diabetes .csv'
df=pd.read_csv(file) print("Sample
data:")
print(df.head())

# Step 1: Import required libraries
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
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())

# Calculate daily returns
hdfc=data['HDFCBANK.NS'] hdfc['Daily Return']
= hdfc['Close'].pct_change() print("\nSummary
statistics for HDFC:") print(hdfc.describe())
```

```
icic=data['ICICIBANK.NS'] icic['Daily Return'] =
icic['Close'].pct_change() print("\nSummary
statistics for ICICI:") print(icic.describe())

kotak=data['KOTAKBANK.NS'] kotak['Daily
Return'] = kotak['Close'].pct_change()
print("\nSummary statistics for Kotak:")
print(kotak.describe())


# Plot the closing price and daily returns
plt.figure(figsize=(12,  6))  plt.subplot(2,
1,  1)  hdfc['Close'].plot(title="    Closing
Price")  icic['Close'].plot(title="    Closing
Price") kotak['Close'].plot(title=" Closing
Price") plt.subplot(2, 1, 2)
hdfc['Daily Return'].plot(title="Daily Returns" ) icic['Daily
Return'].plot(title=" Daily Returns") kotak['Daily
Return'].plot(title="Daily Returns") plt.tight_layout()
plt.show()
```

Program 2

Demonstrate various data pre-processing techniques for a given dataset



PAGE NO :
DATE : 5/3/25

lab -1
Demonstrate various data pre-processing technique for a given dataset

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

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

from SKlearn.preprocessing import
MinMaxScalar, StandardScalar, LabelEncoder
from SKlearn.impute import
        SimpleImputer

df = pd.

print("Mising")
print(df.isnull.sum())

num_column = df.select_dtypes(include
    =["float","int64"]).column.
imputer = SimpleImputer(Strategy='mean')
df[num_column] = imputer.fit_transform
    (df[num_])
```

PAGE NO :
DATE :

```
cat_col = df.select_dtypes(include=['object']).
                columns
imputer_cat = SimpleImputer(Strategy='most_fri')
df[cat_col] = imputer_cat.fit_transform
                (df[cat_columns])
```

(iii) Handling outliers

```
Q1 = df[num_col].quantile(0.25)
Q3 = df[num_col].quantile[0.75)
IQR = Q3 - Q1

df_clean = df[(df[num_col] < (Q1 - 1.5 * IQR)) |
               (df[num_col] > (Q3 + 1.5 * IQR))

df_scaled = pd.Dataframe(scaler.fit_transform
             (df_clean[num_col]), columns = num_col)

df_final = pd.concat([df_clean[cat_col],
             df_scaled], axis = 1)

print("\cleaned & Scaled data:")
print(df_final.head()).
```

① load .csv file into dataframe

```
df = pd.Data read_csv("housing_csv")
```

② display informat° of all column

```
print(df.info())
```

PAGE NO :
DATE :

(iii) To display Statistical info of all numerical
print(df.describe())

(iv) To display the count of unique labels for "Ocean proxi" column
print(df["ocean proxi"].count())

(v) To display which attributes (col) in a dataset have missing values count greater than zero
Miss = df.isnull().Sum()
col_miss = miss[miss>0]
print(col_miss)

Diabetes

① which column in the Dataset had missing values? how did you handle them?
None, but if had then Sum column's NAN can be replaced with median &, Categorial col null can be replaced by mode

② which categorial col did you identify in the dataset? how did you encode them?
gender, class
using ordinal Encoder we can encode categorical Column → low-0, medium-1, High-2.

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

PAGE NO :
DATE :

Min-Max Scaling is also called normalizat° transforms data to fit within a specific range [0 to 1] by scaling based on min-max values in dataset

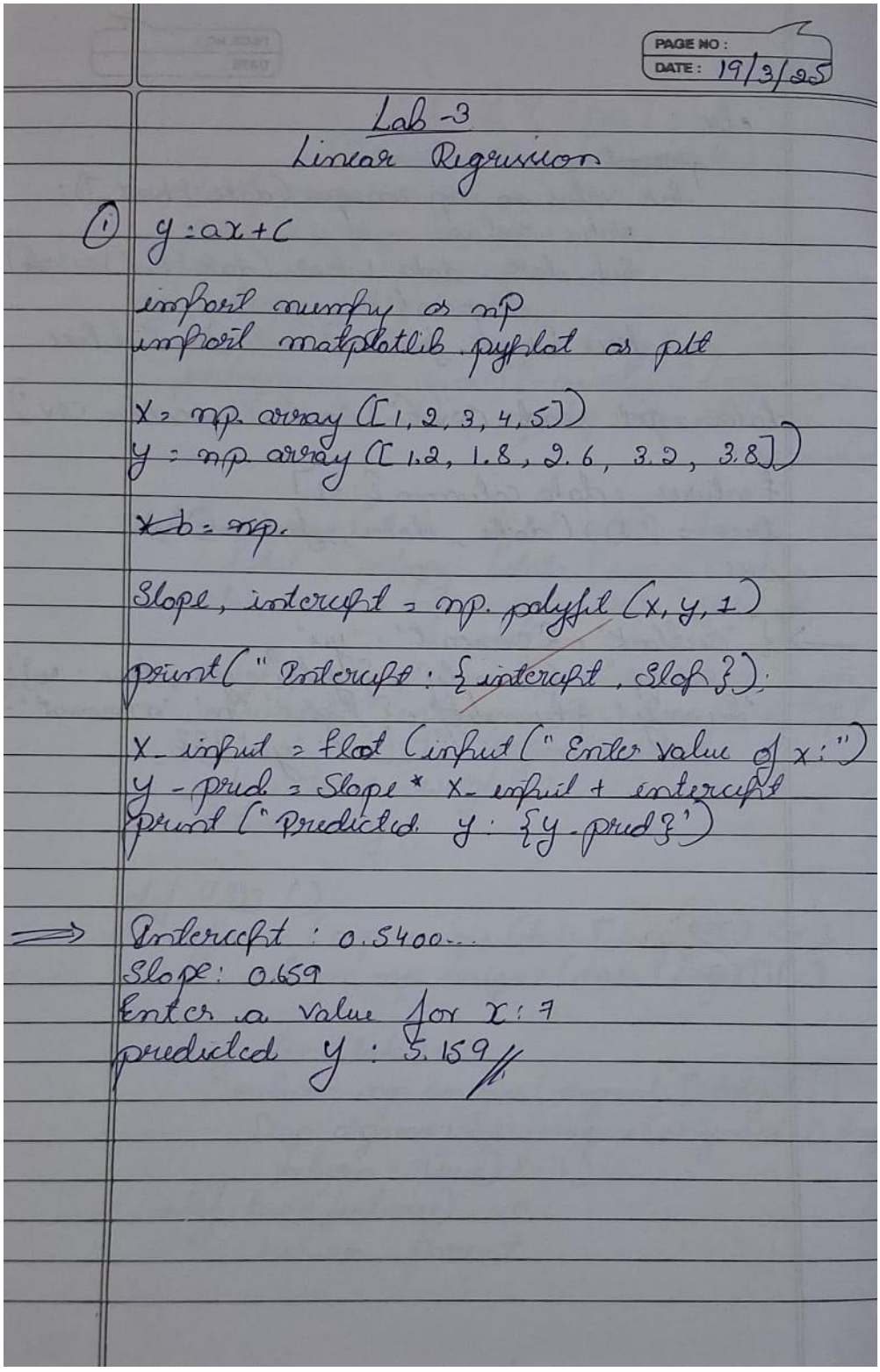
Standardizat° scales data by substracting mean and dividing.

Code:

```
#Handling Missing Values, Handling categorical data, Handling Outliers
import pandas as pd df=pd.read_csv('/content/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)
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



PAGE NO :
DATE : 19/3/25

Lab -3
Linear Regression

① $y = ax + c$

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

x = np.array([1, 2, 3, 4, 5])
y = np.array([1.2, 1.8, 2.6, 3.2, 3.8])

~~x_b = np.~~

slope, intercept = np.polyfit(x, y, 1)

print("Intercept: {intercept}, Slope:{}")

x_input = float(input("Enter value of x:"))
y_pred = slope * x_input + intercept
print("Predicted y: {y_pred}")
```

⟹
```
Intercept : 0.5400...
Slope: 0.659
Enter a value for x: 7
predicted y : 5.159
```

PAGE NO :
DATE :

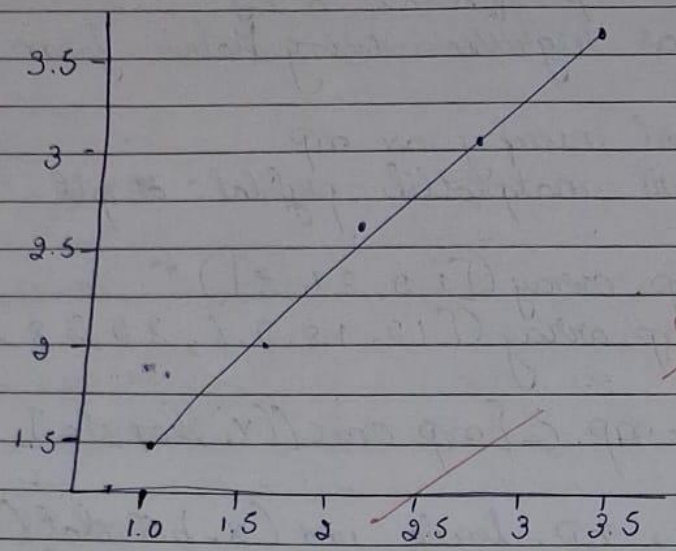
theta
$\beta = [(X^T \cdot X)^{-1} X^T] y.$

=> Linear regression using Matrix form.

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

x = np.array([1,2,3,4,5])
y = np.array([1.2, 1.8, 2.6, 3.2, 3.8])

x_b = np.c_[np.ones((x.shape[0], 1)), x]

theta = np.linalg.inv(x_b.T.dot(x_b)).
        dot(x_b.T).dot(y)

intercept, slope = theta

print(intercept);
print(slope);

x_input = float(input("Enter a value of x"))
x_input_b = np.array([1, x_input])
y_pred = x_input_b.dot(theta)
                    (p.7)

print("Predicted y for x =
    {x_input} : {y_pred}).
```

=> Intercept : 0.540
Slope : 0.66 [theta 1)
Enter a value of x to predict y : 7
predicted y for X : 7.0 : 5.160.

PAGE NO :
DATE :



Code:

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

# Data
x = [1, 2, 3, 4, 5] y =
[12, 18, 22, 28, 35]
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()
```

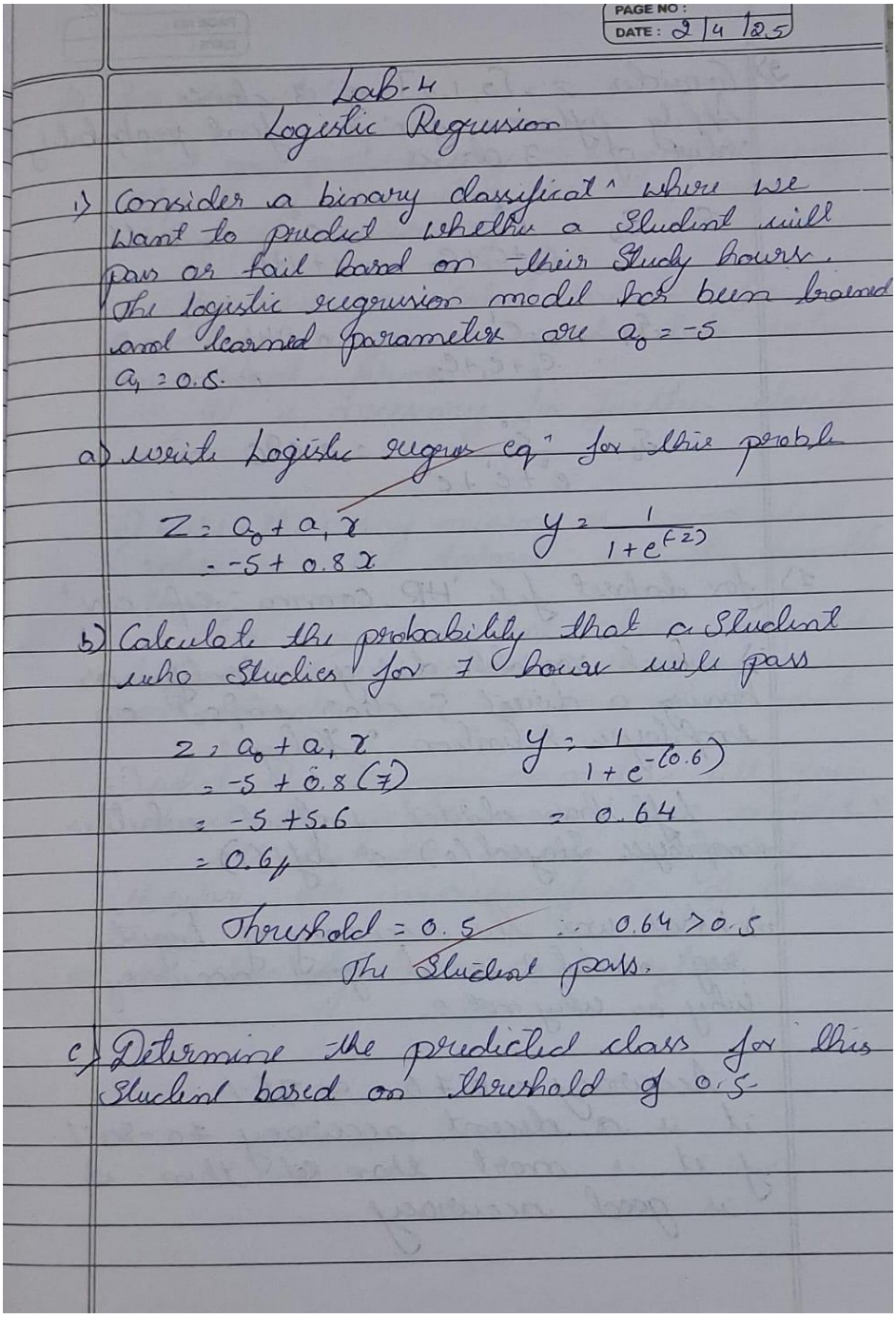
Code:

```
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 4

Build Logistic Regression Model for a given dataset



PAGE NO :
DATE : 2/4/25

Lab-4
Logistic Regression

1) Consider a binary classificat^ 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 learned parameter are $a_0 = -5$ $a_1 = 0.8$.

a) write Logistic regres eq^ for this probl

$z = a_0 + a_1 x$
$= -5 + 0.8x$

$y = \frac{1}{1+e^{(-z)}}$

b) Calculate the probability that a Student who Studies for 7 hours will pass

$z = a_0 + a_1 x$
$= -5 + 0.8(7)$
$= -5 + 5.6$
$= 0.6$

$y = \frac{1}{1+e^{-(0.6)}}$
$= 0.64$

Threshold = 0.5 $\therefore 0.64 > 0.5$
The Student pass.

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

DATE :

9) Consider $z = [2, 1, 0]$ of 3 classes. Apply softmax fun to find probability value of 3 classes

$$S = S_2 = \frac{e^2}{e^2 + e^1 + e^0} = \frac{e^2}{2 + 1 + 0} = 0.66$$

$$S_1 = \frac{e^1}{e^2 + e^1 + e^2} = 0.244$$

$$S_0 = \frac{e^0}{e^2 + e^1 + e^0} = 0.090$$

1) for dataset file 'HR_comma_sep.csv"

i) which variable did you identify as having a direct & clear impact on employee retention? Why?

Left has direct impact whether employee Stayed (0) or left (1)

ii) What was the accuracy of Logistic reg model? is it good accuracy? why or why not?

Accuracy = 0.76 ≈ 76%
it is a decent accuracy 70-80%
if it is more than 80% then it is good accuracy.

PAGE NO :
DATE :

2) for Zoo dataset

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

Yes, we use label encoder for converting categorical ~~encoder~~ data into numerical form.

It is necessary for further classification

ii) Were there any missing or inconsistent value in the dataset: how do handle them?

There are inconsistent values in the Animal names column as There is not Standardizat" of formats & units

iii) what does the confusion matrix tell you about the performance of your model?

all the No are in diagonal shows No misidentificat" of classes.

iv) No class types are misclassified

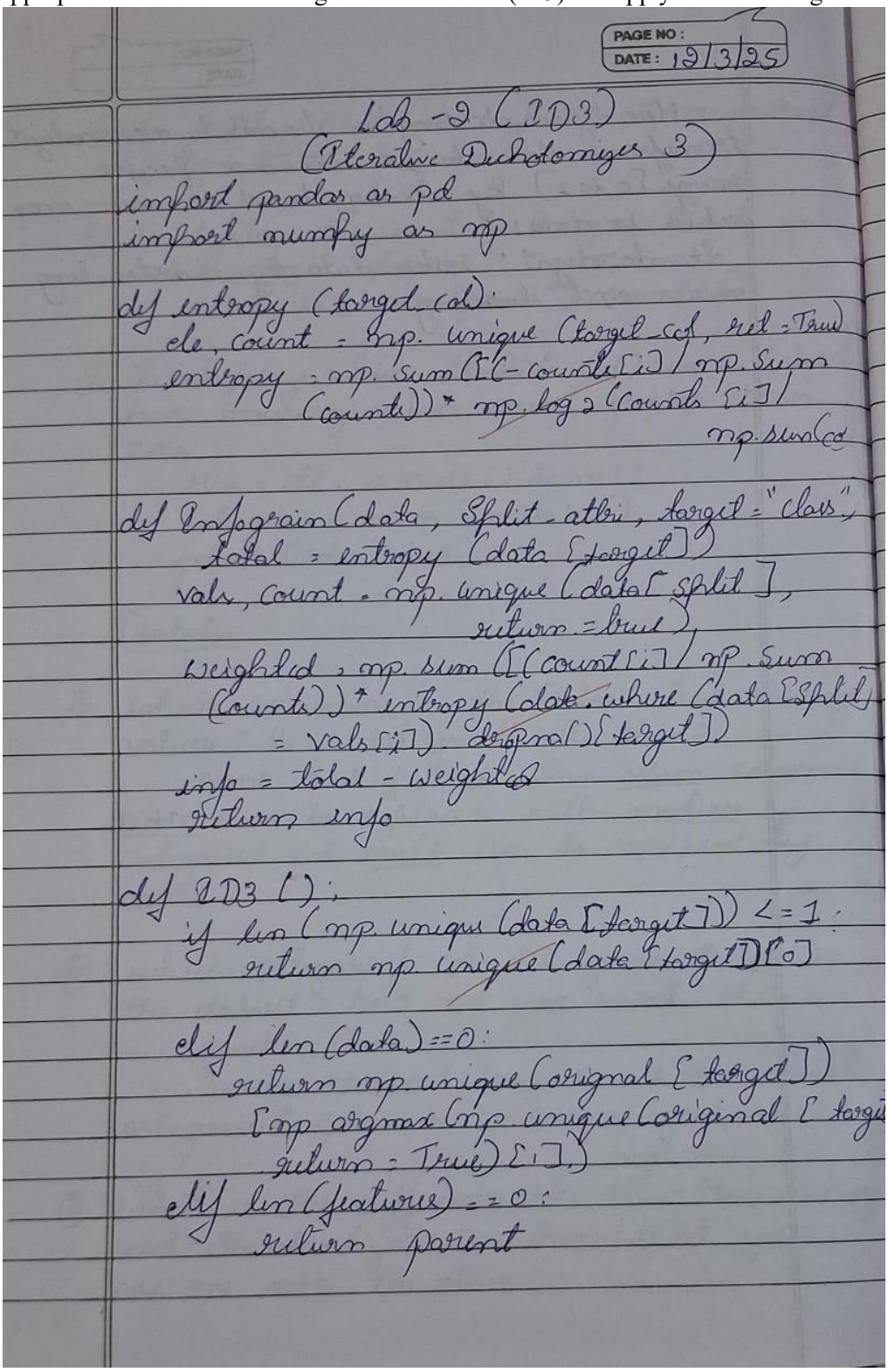
Code:

```
import numpy as np 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, classification_report, confusion_matrix
# Load dataset
file_path = "/content/zoo-data.csv" df
= pd.read_csv(file_path)
# Drop the animal_name column as it is not a feature
df = df.drop(columns=["animal_name"]) # Define
features and target X =
df.drop(columns=["class_type"]) y = df["class_type"]
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Train multinomial logistic regression model
model = LogisticRegression(multi_class='multinomial', solver='lbfgs', max_iter=500)
model.fit(X_train, y_train) # Make predictions
y_pred = model.predict(X_test)
# Evaluate model
accuracy = accuracy_score(y_test, y_pred) print("Accuracy:",
accuracy)
print("Classification Report:\n", classification_report(y_test, y_pred))
# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
labels = np.unique(y) plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=labels, yticklabels=labels)
plt.xlabel('Predicted') plt.ylabel('Actual') plt.title('Confusion Matrix') plt.show() import pandas as pd
from matplotlib import pyplot as plt
import math
from sklearn.model_selection import train_test_split from
sklearn.linear_model import LogisticRegression

# Load HR dataset
df = pd.read_csv("/content/HR_comma_sep.csv") df.head()

plt.scatter(df.satisfaction_level, df.left, marker='+', color='red')

# Splitting dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(df[['satisfaction_level']], df.left, train_size=0.9,
random_state=10)
```

```
X_train.shape
X_test

# Training logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train) X_test
y_test
y_predicted = model.predict(X_test) y_predicted
model.score(X_test, y_test) model.predict_proba(X_test)
y_predicted = model.predict([[0.5]])

# model.coef_ indicates value of m in y=m*x + b equation model.coef_

# model.intercept_ indicates value of b in y=m*x + b equation
model.intercept_

# Define sigmoid function and do the math manually def
sigmoid(x):
   return 1 / (1 + math.exp(-x))

def prediction_function(satisfaction_level):
   z = model.coef_[0][0] * satisfaction_level + model.intercept_[0]
y = sigmoid(z)     return y satisfaction_level = 0.35
probability = prediction_function(satisfaction_level) print(f"Probability
of leaving: {probability:.2f}")

if probability >= 0.5:
   print("The employee will leave.") else:
   print("The employee will stay.")
```

Program 5

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



PAGE NO :
DATE : 12/3/25

Lab - 2 (ID3)
(Iterative Dichotomizer 3)

```
import pandas as pd
import numpy as np

def entropy(target_col):
    ele, count = np.unique(target_col, ret=True)
    entropy = np.sum([(-counts[i]/np.sum
        (counts))* np.log2(counts[i]/
                           np.sum(co
def Infogain(data, Split_attri, target="class",
    total = entropy(data[target])
    vals, count = np.unique(data[Split],
                            return=True)
    weighted = np.sum([(count[i]/np.sum
        (counts))* entropy(data.where(data[Split]
            = vals[i]).dropna()[target])
    info = total - weighted
    return info

def ID3():
    if len(np.unique(data[target])) <= 1:
        return np.unique(data[target])[0]

    elif len(data)==0:
        return np.unique(original[target])
            [np.argmax(np.unique(original[targ
                return=True)[1]]
    elif len(features)==0:
        return parent
```

PAGE NO :
DATE :

```
clx :
    parent :
    for value in np unique (data [best_]):
        value : value
        Sub_data = data. where (data[best] == val).
                   dropna ()
        tree [best_feature][value] = Subtree

data = pd. read_csv ("\Content\ weather.csv");

features = data. columns [:-1]
tree = ID3 (data, data, features)
tree
```

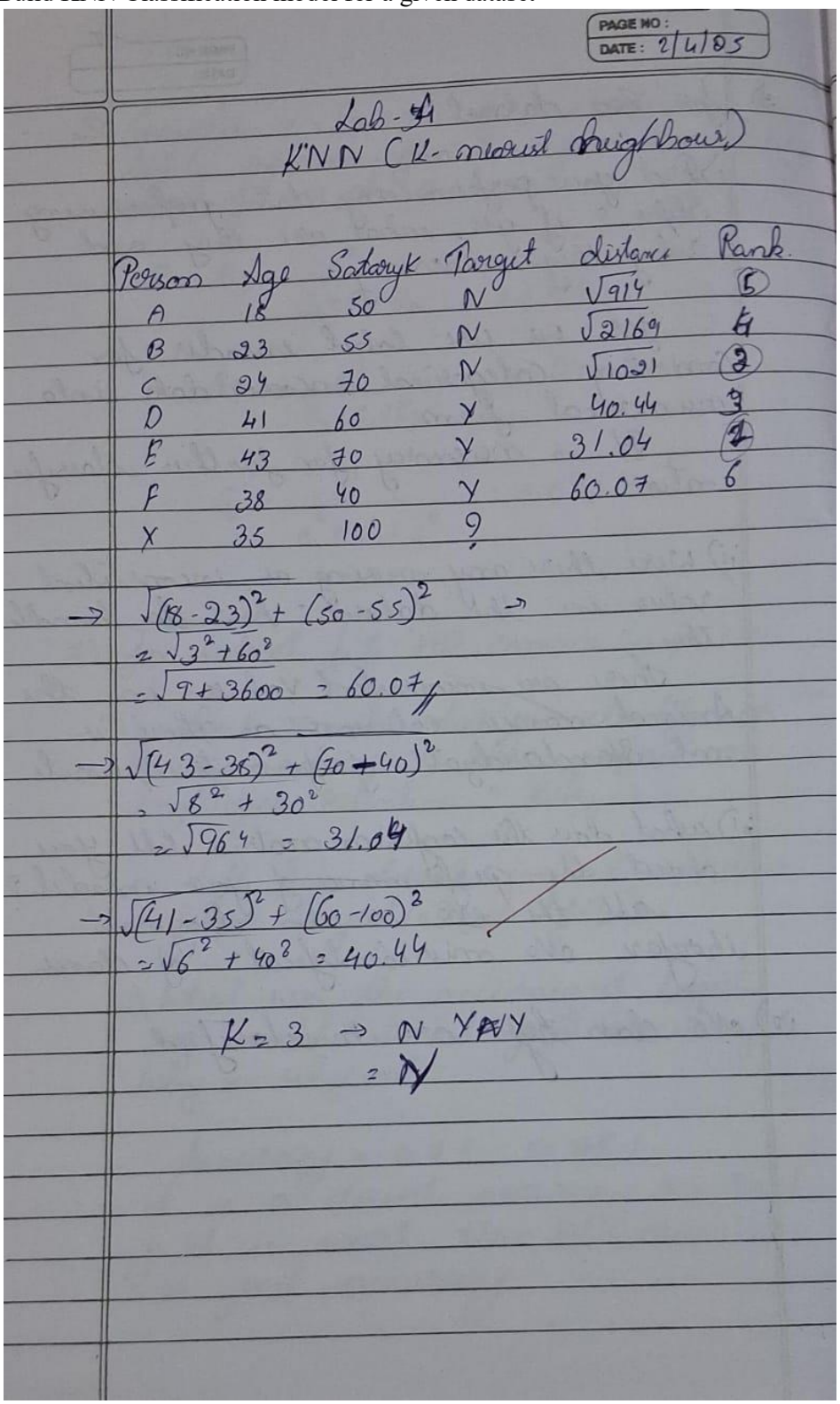
→ { "outlook" : { "overcast" : "yes",
"rainy" : { "windy" : { false : yes, true : no}},
"Sunny" : { "humidity" : { "high" : "no", "normal" : "yes" }}}

12-3-25

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']=outlook.fit_transform(data['temp'])
data['humidity']=outlook.fit_transform(data['humidity'])
data['windy']=outlook.fit_transform(data['windy'])
data['play']=outlook.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)
x_test y_test
classifier.score(x_test,y_test) from
sklearn import tree
tree.plot_tree(classifier)
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



Lab - 4
KNN (K- nearest neighbour)

Person	Age	Salary	Target	distance	Rank
A	18	50	N	$\sqrt{914}$	5
B	23	55	N	$\sqrt{2169}$	4
C	24	70	N	$\sqrt{1021}$	2
D	41	60	Y	40.44	3
E	43	70	Y	31.04	1
F	38	40	Y	60.07	6
X	35	100	?		

→ $\sqrt{(18-23)^2 + (50-55)^2}$ →
$= \sqrt{3^2+60^2}$
$= \sqrt{9+3600} = 60.07$

→ $\sqrt{(43-35)^2 + (70+40)^2}$
$= \sqrt{8^2+30^2}$
$= \sqrt{964} = 31.04$

→ $\sqrt{(41-35)^2 + (60-100)^2}$
$= \sqrt{6^2+40^2} = 40.44$

K = 3 → N Y Y
= Y

PAGE NO :
DATE :

For the Iris dataset:

→ how to choose K value?

Testing multiple K values & comparing their accuracy. The accuracy & error rate for K are compared and the most optimal K is selected

Here K = 3

Diabetes dataset

⇒ what is the purpose of feature Scaling? how to perform it?

Feature Scaling ensures all features contribute equally to the nearest neighbours Scaling is done so that features like glucose / age don't dominate the one with the smaller ranges.

2-4-25

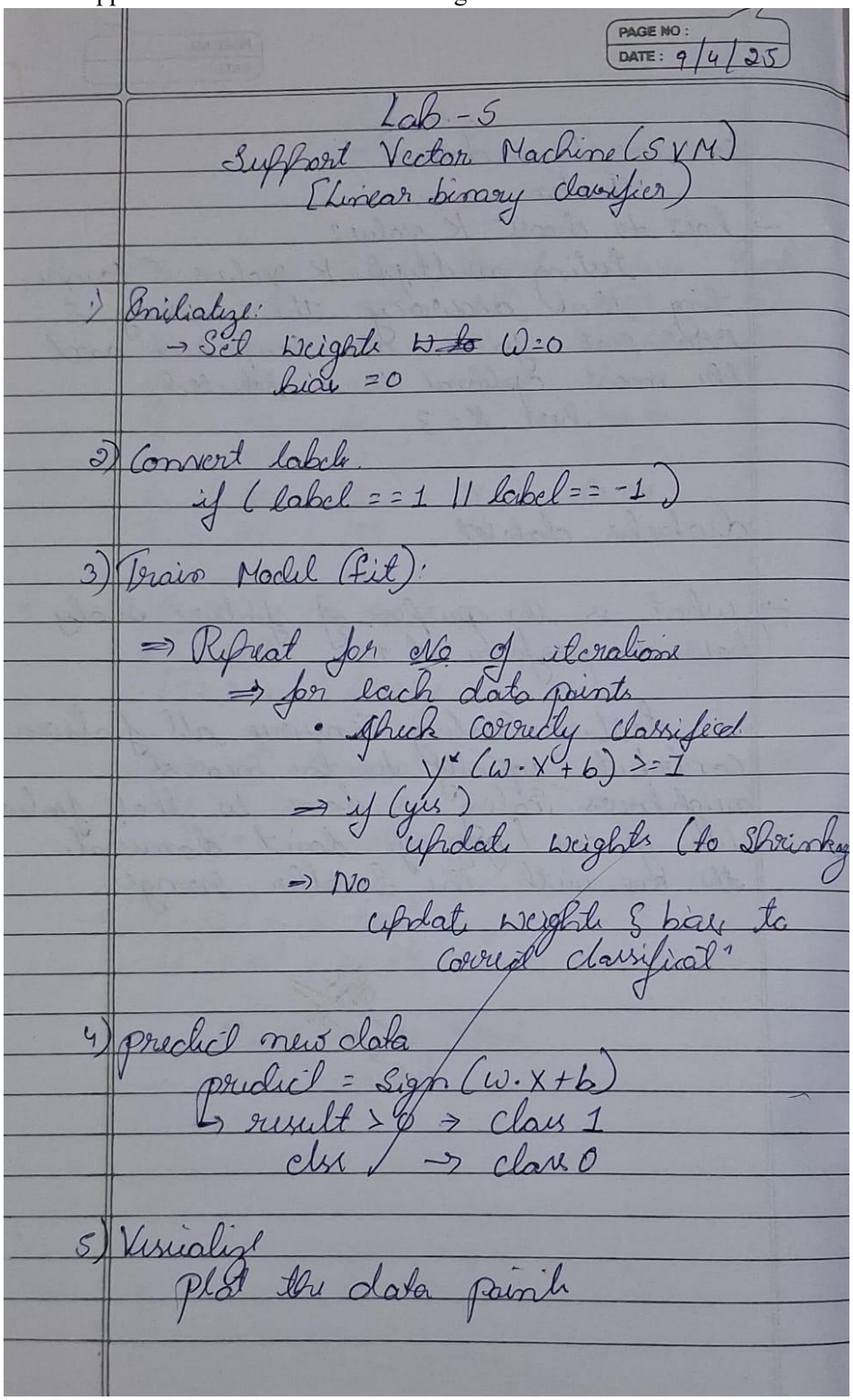
Code:

```
import numpy as np import
pandas as pd import
matplotlib.pyplot as plt import
seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler from
sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matri
# Load dataset
file_path = "/content/diabetes.csv" df
= pd.read_csv(file_path)
# Define features and target X =
df.drop(columns=["Outcome"]) y =
df["Outcome"]
# Split data into training (80%) and testing (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Feature scaling (Standardization) scaler
= StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
# Initialize and train KNN classifier with k=5 k
= 5
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train, y_train) #
Make predictions y_pred =
knn.predict(X_test)
# Evaluate model
accuracy = accuracy_score(y_test, y_pred) print("Accuracy:",
accuracy)
print("Classification Report:\n", classification_report(y_test, y_pred))
conf_matrix = confusion_matrix(y_test, y_pred)
labels = ["Non-Diabetic", "Diabetic"] plt.figure(figsize=(8,
6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=labels, yticklabels=labels)
plt.xlabel('Predicted') plt.ylabel('Actual') plt.title('Confusion Matrix') plt.show() import pandas as pd
import numpy as np import matplotlib.pyplot as plt import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler from
sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_repor df
= pd.read_csv('/content/heart.csv')
# Define features and target
```

```
X = df.drop(columns=['target'])  # Assuming 'target' is the classification column y
= df['target']
# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Feature scaling scaler
= StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
# Find the best K value k_values
= range(1, 21) accuracy_scores =
[] for k in k_values:
   model = KNeighborsClassifier(n_neighbors=k)
   model.fit(X_train, y_train)
   y_pred = model.predict(X_test)accuracy_scores.append(accuracy_score(y_test, y_pred)) best_k
= k_values[np.argmax(accuracy_scores)]
print(f'Best K value: {best_k}') #
Train model with best K
best_model = KNeighborsClassifier(n_neighbors=best_k)
best_model.fit(X_train, y_train) y_pred
= best_model.predict(X_test)
# Evaluate model
accuracy = accuracy_score(y_test, y_pred) print(f'Accuracy
with best K ({best_k}): {accuracy:.4f}') print("Classification
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.xlabel('Predicted') plt.ylabel('Actual')
plt.title(f'Confusion Matrix - KNN (K={best_k})') plt.show()
# Plot K values vs. Accuracy
plt.plot(k_values, accuracy_scores, marker='o')
plt.xlabel('K Value')
plt.ylabel('Accuracy') plt.title('K
Value vs Accuracy')
plt.show()
```

Program 7

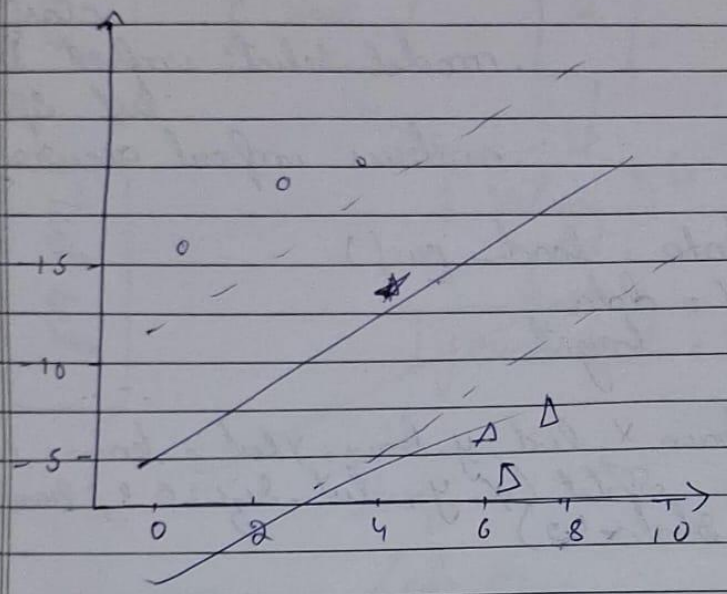
Build Support vector machine model for a given dataset



PAGE NO :
DATE :

, decision boundary & margin

→ plot new point & show its predicted class.



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)

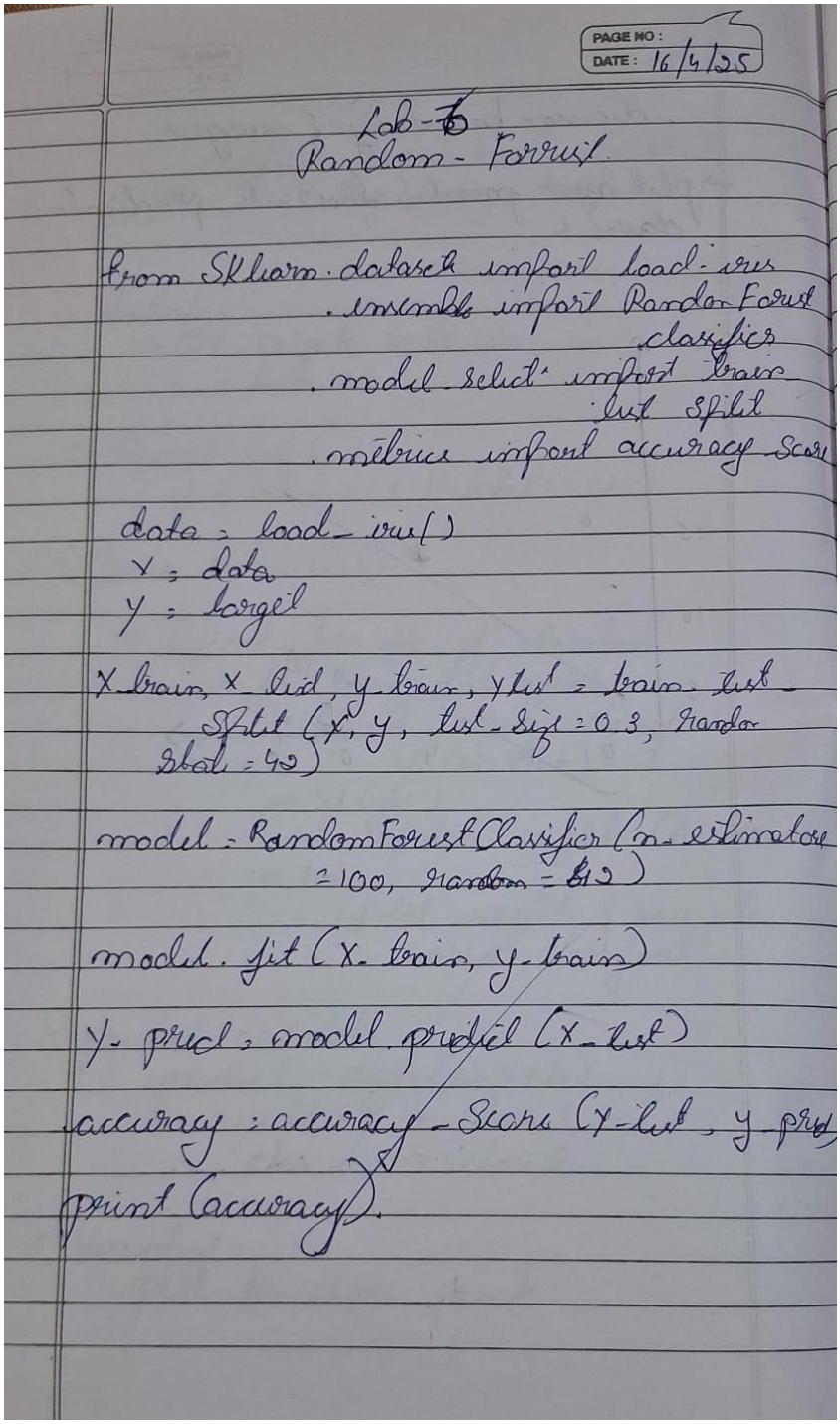
def visualize(self, X, y, new_point=None, prediction=None):
  def get_hyperplane(x, w, b, offset):
return (-w[0] * x + b + offset) / w[1]
  fig = plt.figure() ax =
fig.add_subplot(1, 1, 1) for i,
sample in enumerate(X): if
y[i] == 1:
  plt.scatter(sample[0], sample[1], marker='o', color='blue', label='Class +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--')

        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, 7],
        [2, 8],
        [3, 8],
        [8, 1],
        [9, 2],
        [10, 2]
    ])
    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'}")
```

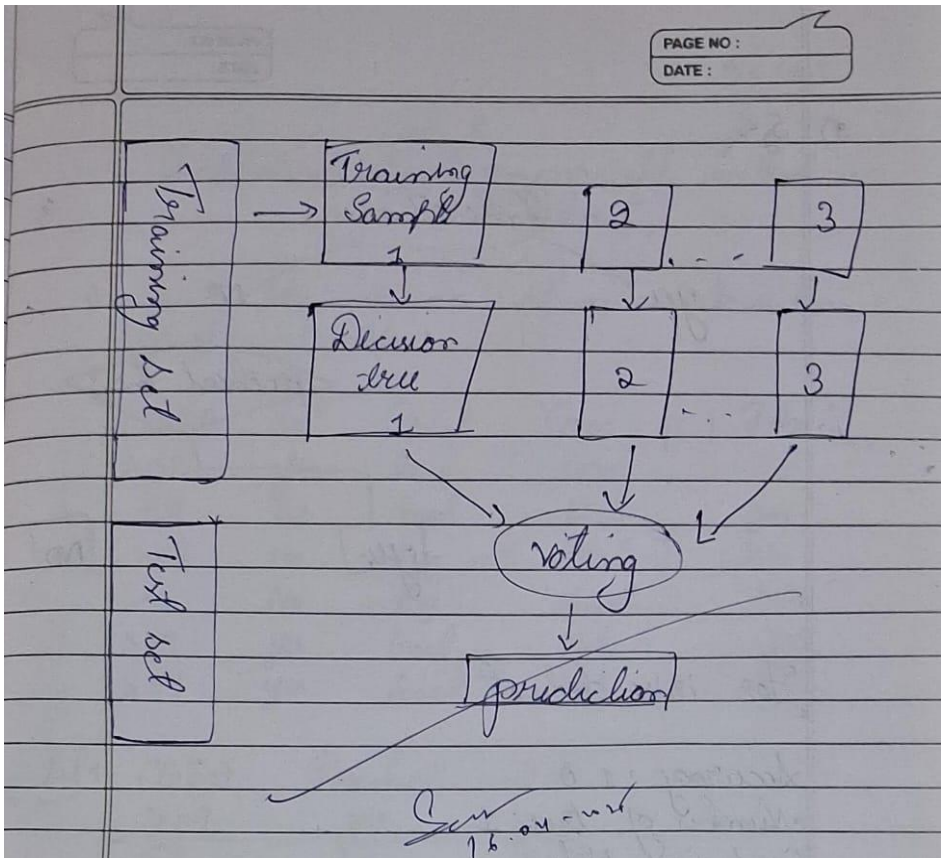
Program 8

Implement Random forest ensemble method on a given dataset



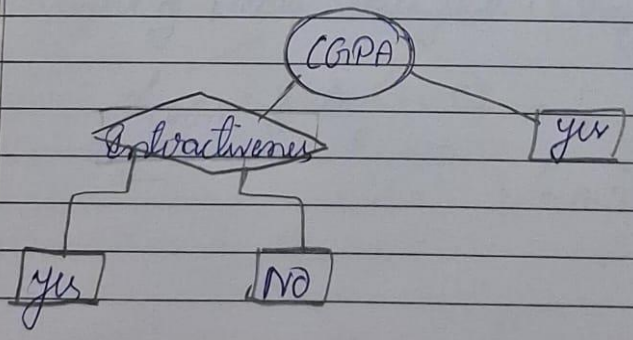
Lab-6
Random - Forrest

```
from Sklearn.dataset import load_iris
    .ensemble import Random Forest
                    classifier
    .model_select import train
                    test split
    .metrics import accuracy_score

data = load_iris()
X = data
y = target

X_train, X_test, y_train, y_test = train_test_
    split(X, y, test_size=0.3, random
    state=42)

model = RandomForestClassifier(n_estimators
    =100, random = 42)

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

print(accuracy).
```



Lab-7

S1.	CGPA	Interac	Skills	Knows	Job
1	≥9	yes	Good	Good	Yes
2	<9	No	Moder	Good	Yes
3	≥9	No	Moder	Aver	No
3	≥9	No	Moder	Ave	No
5	≥9	yes	Mode	Good	Yes

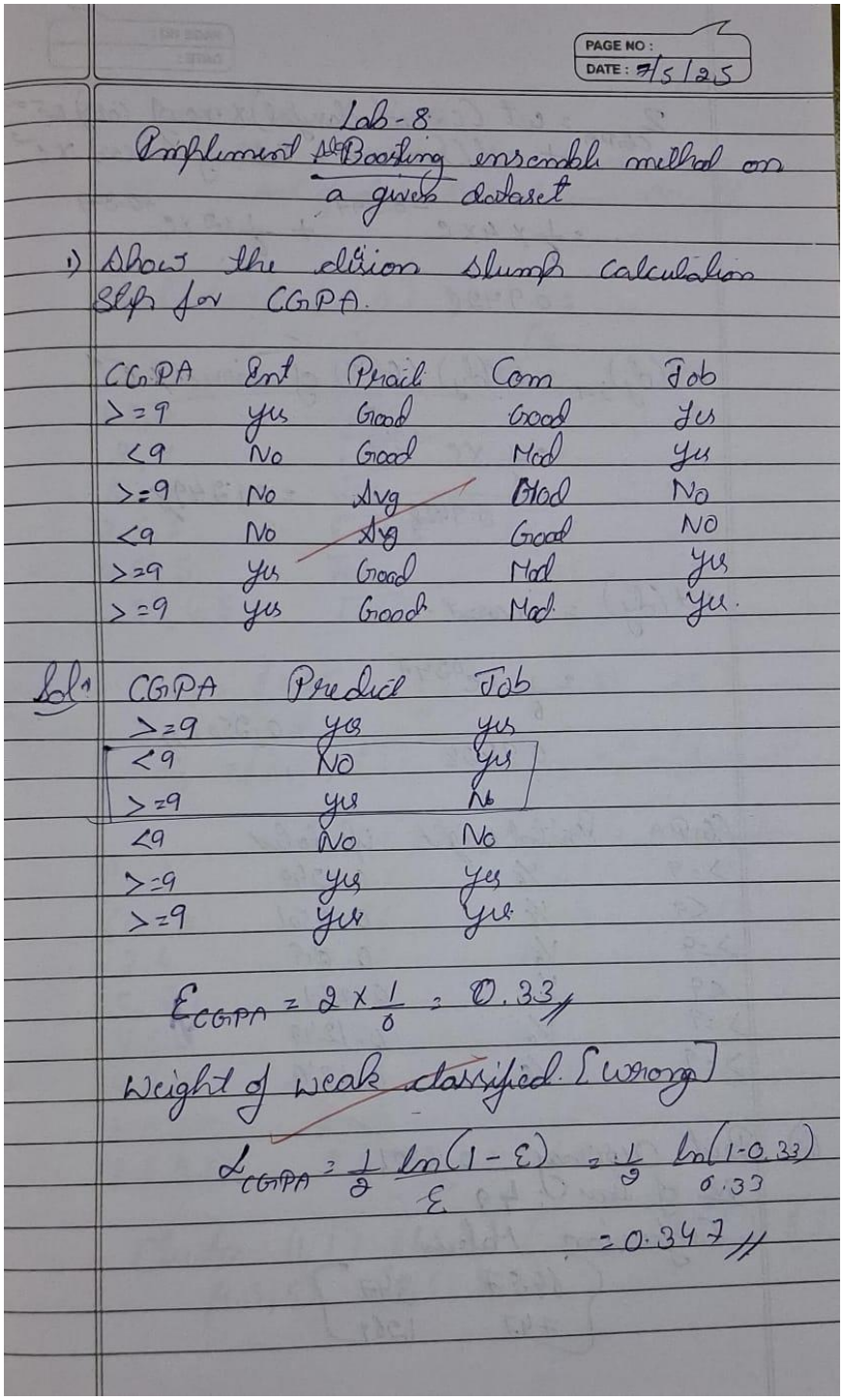


Code:

```
import pandas as pd
from sklearn.datasets import load_iris from
sklearn.model_selection import train_test_split from
sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix

# Load the iris dataset
iris = load_iris()

# Convert to DataFrame
df = pd.DataFrame(data=iris.data, columns=iris.feature_names) df['species']
= iris.target

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

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

best_score = 0 best_n
= 0
best_cm = None

# Try different numbers of trees for n in range(1, 101):     clf =
RandomForestClassifier(n_estimators=n, random_state=42)
clf.fit(X_train, y_train)     y_pred = clf.predict(X_test)     acc =
accuracy_score(y_test, y_pred)     if acc > best_score:
best_score = acc
      best_n = n
      best_cm = confusion_matrix(y_test, y_pred)

print(f"Best Accuracy: {best_score}")
print(f"Number of Trees: {best_n}")
print("Confusion Matrix:") print(best_cm)
```

Program 9

Implement Boosting ensemble method on a given dataset



PAGE NO :
DATE : 7/5/25

Lab-8

Implement Boosting ensemble method on a given dataset

1) Show the decision stump calculation steps for CGPA.

CGPA	Int	Practical	Com	Job
>=9	yes	Good	Good	Yes
<9	No	Good	Mod	yes
>=9	No	Avg	Mod	No
<9	No	Avg	Good	NO
>=9	yes	Good	Mod	yes
>=9	yes	Good	Mod	yes.

Sol:

CGPA	Predict	Job
>=9	yes	yes
<9	NO	yes
>=9	yes	No
<9	No	No
>=9	yes	yes
>=9	yes	yes

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

Weight of weak classified [wrong]

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

$= 0.347$ //

DATE :

Z_{CGPA} = wt (Correct classified) × no of corr) $e^{-\alpha}$
+ wt (Incorrect) × No of incor) × e^{α}

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

$= 0.9428$

$wt(d_j)_{i+1}$ = wt(d_j) (GPA) of correct $* e^{-\alpha}$ / Z

$= \frac{\frac{1}{6} \times e^{-0.347}}{0.9428} = 0.1249$

$wt(d_j)$ = incorrect

$= \frac{\frac{1}{6} \times e^{0.347}}{0.9428} = 0.2501$

CGPA	Initial weight	Updated
>=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

(d) But average : 0.8501
No of Iter 0.49
Confusion Matrix:

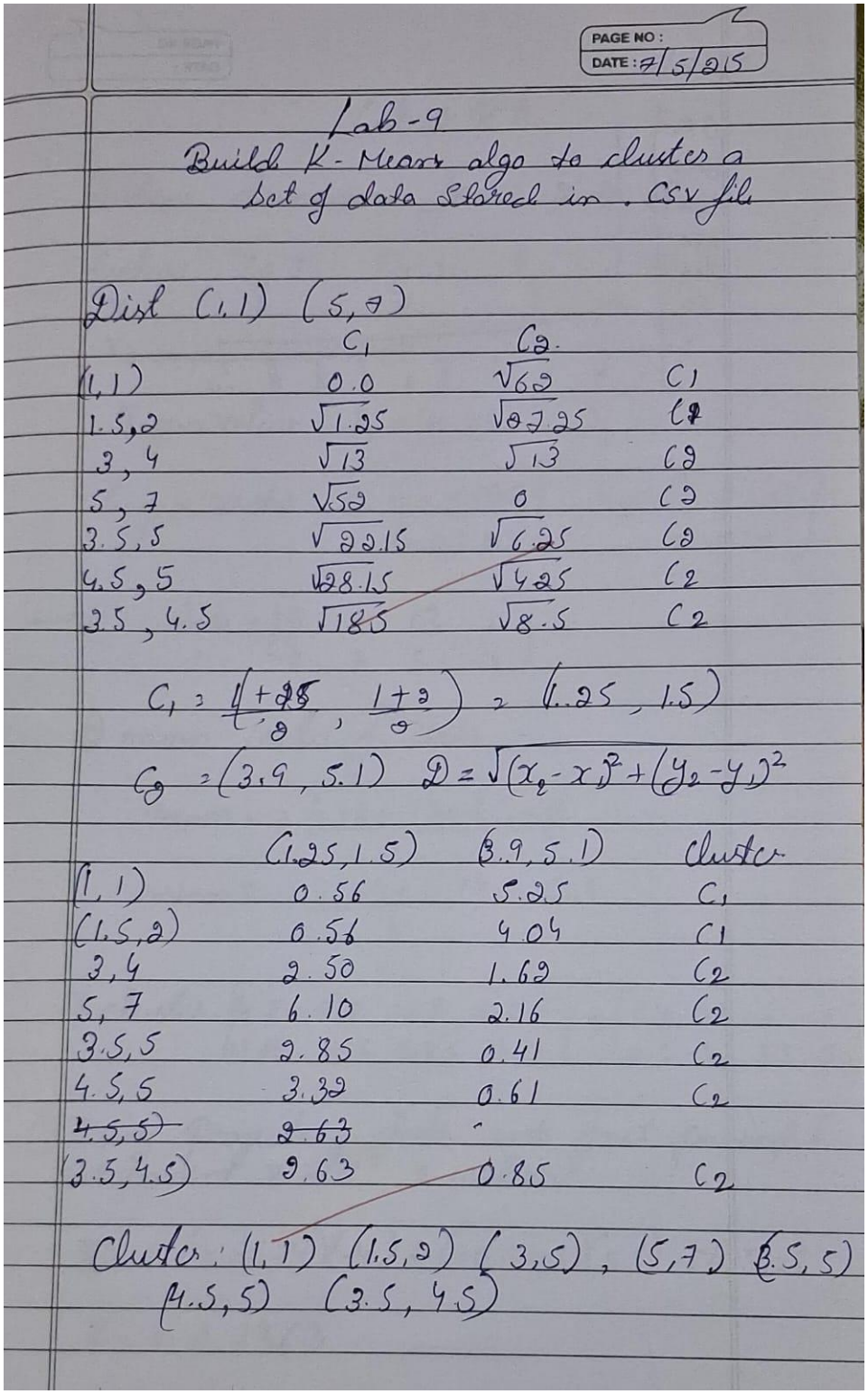
$$\begin{bmatrix} 6457 & 347 \\ 747 & 1269 \end{bmatrix}$$

Code:

```
import pandas as pd import
numpy as np
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
# Load dataset
df = pd.read_csv("/content/income.csv")
# Define features and target X =
df.drop("income_level", axis=1) y =
df["income_level"]
# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 1: AdaBoost with 10 estimators
model = AdaBoostClassifier(n_estimators=10, random_state=42)
model.fit(X_train, y_train) y_pred = model.predict(X_test)
base_accuracy = accuracy_score(y_test, y_pred) conf_matrix =
confusion_matrix(y_test, y_pred) print(f"Accuracy with 10
estimators: {base_accuracy:.4f}") print("Confusion Matrix:\n",
conf_matrix) # Step 2: Fine-tuning n_estimators
estimator_range = range(10, 201, 10)
scores = [] for n in estimator_range:     model =
AdaBoostClassifier(n_estimators=n, random_state=42)
model.fit(X_train, y_train)     y_pred = model.predict(X_test)
acc = accuracy_score(y_test, y_pred)     scores.append(acc)
   print(f"n_estimators={n}, Accuracy={acc:.4f}") # Plot
accuracy vs. number of estimators
plt.plot(estimator_range, scores, marker='o')
plt.title("Accuracy vs. Number of Estimators (AdaBoost)")
plt.xlabel("Number of Estimators") plt.ylabel("Accuracy")
plt.grid(True) plt.show()
# Best score and configuration best_n =
estimator_range[np.argmax(scores)]
best_score = max(scores)
print(f"Best accuracy: {best_score:.4f} achieved with {best_n} estimators.")
```

Program 10

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



PAGE NO :
DATE : 7/5/25

Lab-9

Build K-Means algo to cluster a set of data stored in .csv file

Dist (1,1) (5,7)

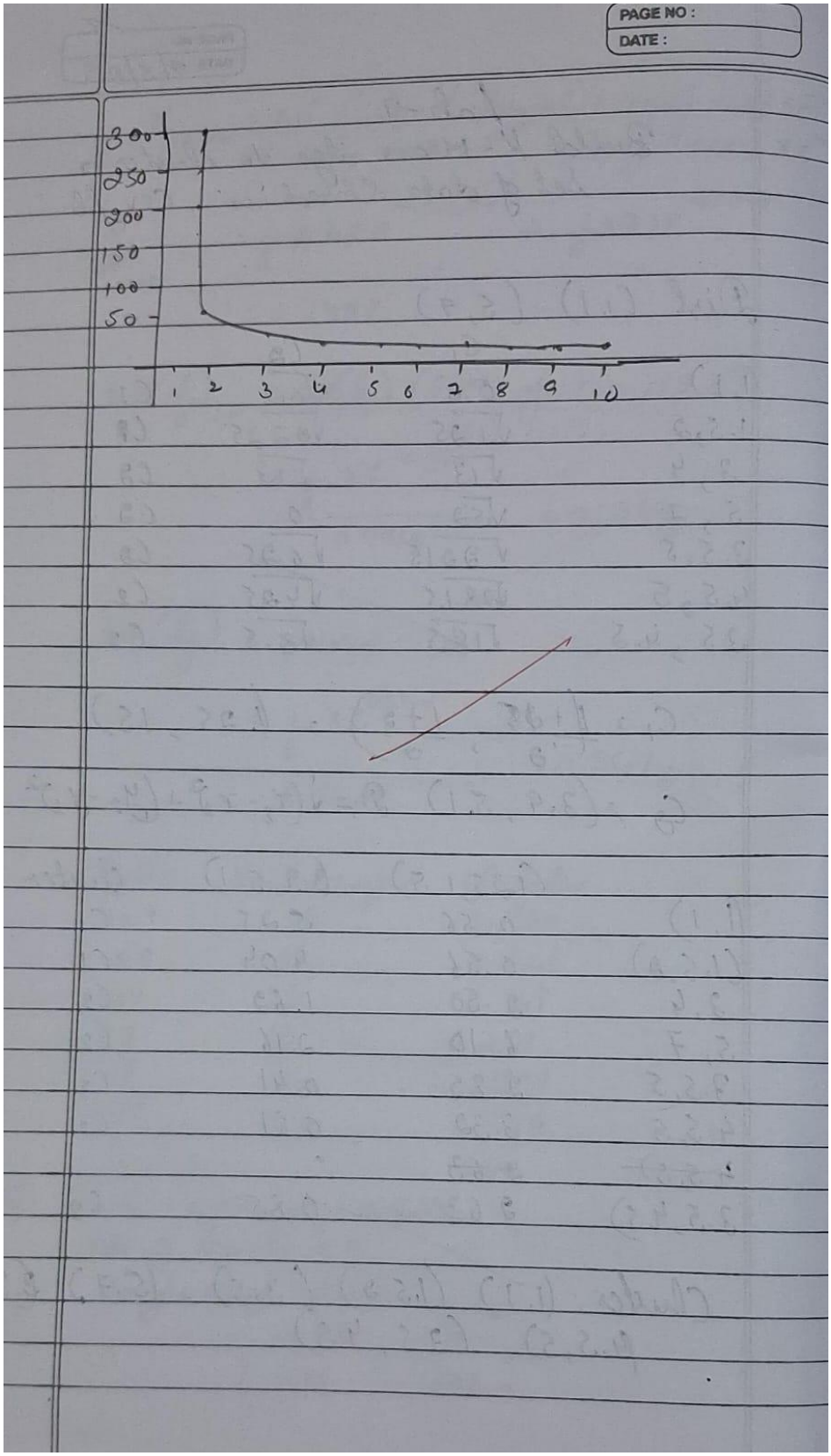
	C_1	C_2	
(1,1)	0.0	$\sqrt{62}$	C_1
1.5,2	$\sqrt{1.25}$	$\sqrt{87.25}$	C_1
3, 4	$\sqrt{13}$	$\sqrt{13}$	C_2
5, 7	$\sqrt{52}$	0	C_2
3.5, 5	$\sqrt{20.15}$	$\sqrt{6.25}$	C_2
4.5, 5	$\sqrt{28.15}$	$\sqrt{4.25}$	C_2
3.5, 4.5	$\sqrt{185}$	$\sqrt{8.5}$	C_2

$C_1 = \left(\frac{1+2}{2}, \frac{1+2}{2}\right) = (1.25, 1.5)$

$C_2 = (3.9, 5.1)$ $\quad d = \sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$

	(1.25, 1.5)	(3.9, 5.1)	cluster
(1, 1)	0.56	5.25	C_1
(1.5, 2)	0.56	4.04	C_1
3, 4	2.50	1.62	C_2
5, 7	6.10	2.16	C_2
3.5, 5	2.85	0.41	C_2
4.5, 5	3.32	0.61	C_2
~~4.5, 5~~	~~2.63~~		
(3.5, 4.5)	2.63	0.85	C_2

Cluster: (1,1) (1.5,2), (3,5), (5,7), (3.5,5) (4.5,5) (3.5,4.5)

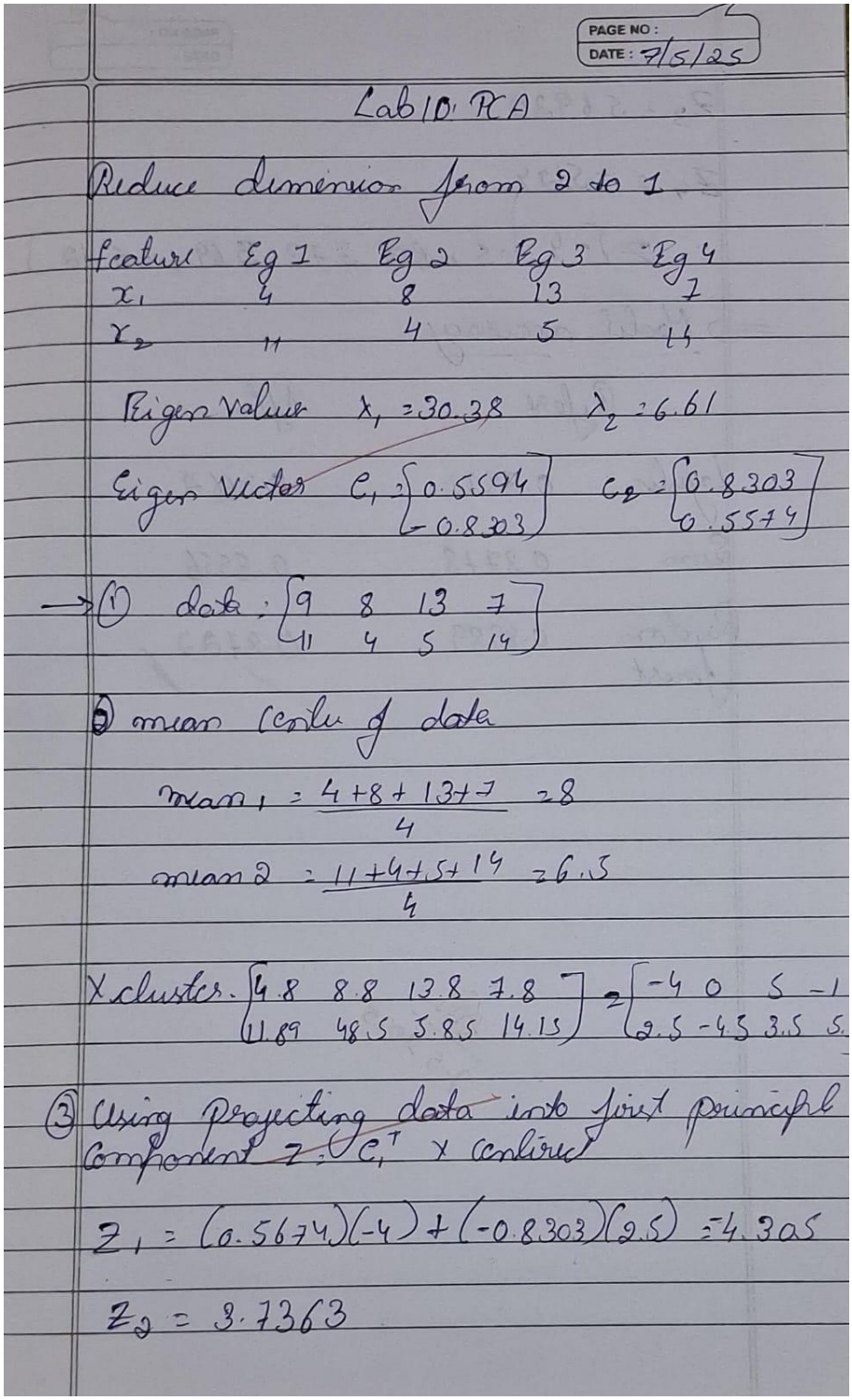

PAGE NO :
DATE :
300
250
200
150
100
50
1 2 3 4 5 6 7 8 9 10

Code:

```
import pandas as pd from sklearn.preprocessing
import StandardScaler from sklearn.cluster
import KMeans import matplotlib.pyplot as plt
df = pd.read_csv("/content/iris (6).csv")  # Replace with actual file path if needed
# Use only petal_length and petal_width
X = df[['petal_length', 'petal_width']]
scaler = StandardScaler() X_scaled =
scaler.fit_transform(X) # Elbow method
to find optimal k wcss = [] k_values =
range(1, 11) for k in k_values:
   kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
kmeans.fit(X_scaled)     wcss.append(kmeans.inertia_)
plt.figure(figsize=(8, 5)) plt.plot(k_values, wcss, marker='o')
plt.title('Elbow Method for Optimal k') plt.xlabel('Number of
clusters (k)') plt.ylabel('WCSS') plt.grid(True) plt.tight_layout()
plt.show()
# Apply K-Means with optimal k = 3
kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
clusters = kmeans.fit_predict(X_scaled) #
Add cluster labels to original dataframe
df['cluster'] = clusters
plt.figure(figsize=(8, 5)) for cluster in
range(3):
   cluster_points = X_scaled[clusters == cluster]     plt.scatter(cluster_points[:,
0], cluster_points[:, 1], label=f'Cluster {cluster}')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
s=200, c='black', marker='X', label='Centroids') plt.title('K-Means Clustering
(k=3)') plt.xlabel('Petal Length (scaled)') plt.ylabel('Petal Width (scaled)')
plt.legend() plt.grid(True) plt.tight_layout()
plt.show()
```

Program 11

Implement Dimensionality reduction using Principal Component Analysis (PCA) method



PAGE NO :
DATE : 7/5/25

Lab 10: PCA

Reduce dimension from 2 to 1

feature	Eg 1	Eg 2	Eg 3	Eg 4
x_1	4	8	13	7
x_2	11	4	5	14

Eigen value $\lambda_1 = 30.38$ $\lambda_2 = 6.61$

Eigen vector $C_1 = \begin{bmatrix} 0.5594 \\ -0.8303 \end{bmatrix}$ $C_2 = \begin{bmatrix} 0.8303 \\ 0.5574 \end{bmatrix}$

→ (1) data: $\begin{bmatrix} 9 & 8 & 13 & 7 \\ 11 & 4 & 5 & 14 \end{bmatrix}$

(2) mean centre of data

$\text{mean}_1 = \frac{4+8+13+7}{4} = 8$

$\text{mean}_2 = \frac{11+4+5+14}{4} = 6.5$

$X_{\text{cluster}} = \begin{bmatrix} 4.8 & 8.8 & 13.8 & 7.8 \\ 11.89 & 48.5 & 5.85 & 14.15 \end{bmatrix} = \begin{bmatrix} -4 & 0 & 5 & -1 \\ 2.5 & -4.5 & 3.5 & 5. \end{bmatrix}$

(3) Using Projecting data into first principal component $Z = C_1^T \times \text{centered}$

$Z_1 = (0.5674)(-4) + (-0.8303)(2.5) = -4.305$

$Z_2 = 3.7363$

DATE :

$Z_3 = 5.6930$

$Z_4 = -5.124$

$\therefore Z[-4.305, 3.73, 5.69, -5.12]$

⇒ Model accuracy

	Before	After
Logistic	0.8833	0.8667
Svm	0.8778	0.8556
Random forest	0.8989	0.8722

7-5-2025

Code:

```
import pandas as pd from sklearn.preprocessing import
LabelEncoder, StandardScaler 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

# Load dataset df = pd.read_csv("/content/heart (1).csv")  # Use
actual path if needed

# Label encode binary categorical columns le
= LabelEncoder()
df["Sex"] = le.fit_transform(df["Sex"])  # M/F -> 1/0
df["ExerciseAngina"] = le.fit_transform(df["ExerciseAngina"])  # Y/N -> 1/0

# One-hot encode categorical columns with >2 categories
df = pd.get_dummies(df, columns=["ChestPainType", "RestingECG", "ST_Slope"],
drop_first=True)

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

# Scale features scaler =
StandardScaler()
X_scaled = scaler.fit_transform(X)

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

# Define models models
= {
   "SVM": SVC(),
   "Logistic Regression": LogisticRegression(max_iter=1000),
   "Random Forest": RandomForestClassifier()
}
```

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

# Apply PCA to reduce dimensions to 2
pca = PCA(n_components=2) X_pca =
pca.fit_transform(X_scaled)

# Split again for PCA-transformed data
X_train_pca, X_test_pca, y_train, y_test = train_test_split(X_pca, y, test_size=0.2,
random_state=42)

# Train and evaluate models (after PCA) print("\n===
Accuracy After PCA (2 components) ===") for name,
model in models.items():     model.fit(X_train_pca,
y_train)     acc = accuracy_score(y_test,
model.predict(X_test_pca))     print(f"{name}: {acc:.4f}")
```