VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

Anish Budavi (1BM23CS401)

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 **Anish Budavi (1BM23CS401),** 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.

Sarala D V Assistant Professor Department of CSE, BMSCE	Dr. Kavitha Sooda Professor & HOD Department of CSE, BMSCE

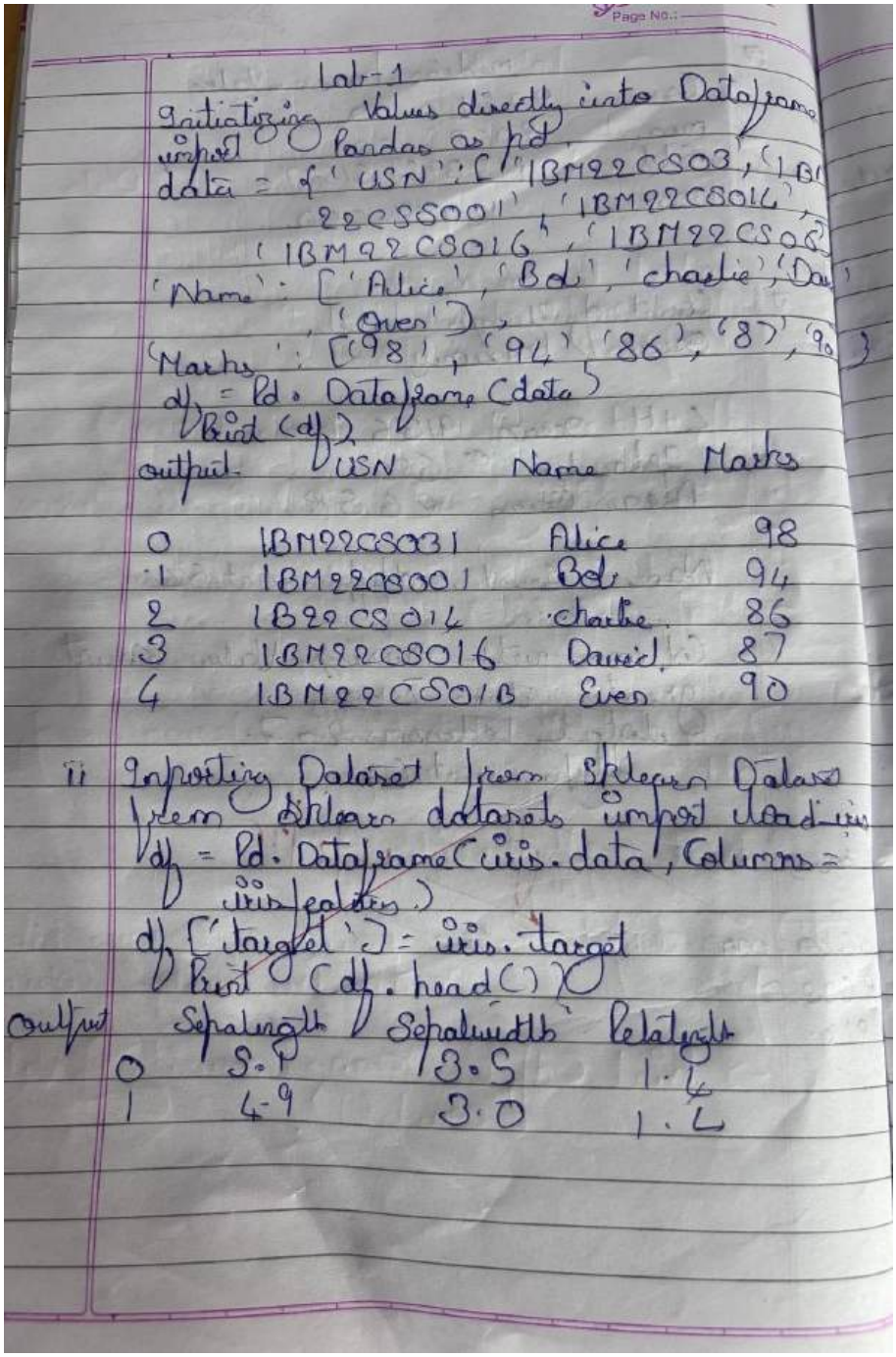
Index

Github Link: https://github.com/Anishbudavi/6thSem-ML-Lab-1BM23CS401.git

Program 1

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

Screenshot



iv) df = pd.read_csv('Mobile Dataset.csv')
print(df.head())
output 0
1

iii) Importing datasets from specific csv file
filePath = "data.csv"
df = pd.read_csv(filePath)
print(df.head())
output

	ID	Name	Age
0	1	Alice	30
1	2	Bob	25

v) Bank Data Analysis
import finance as yf
import pandas as pd
import matplotlib.pyplot as plt
a) tickets = ["HDFCBANK", "ICICIBANK", "KOTAKBANK"]
b) data = yf.download(tickers, start="2021-01-01", end="2024-12-30", group_by="ticker")
print(data.head())
print(data.shape())
print(data.columns)
plt.figure(figsize=(14,10))
for i, ticker in enumerate(tick
bank_data['Daily_Return'] = bank.data['close'].pct_change()
plt.subplot(3,2,2*i+1)
bank_data['close'].plot(title="ticker daily_Return", color="blue")
plt.ylabel('Daily Return')
plt.tight_layout()

Date :
Page No. :

Plt.show()
for ticker in tickers:
    bank_data = data(ticker)
    bank_data.csv(<ticker>) = Stock_d

Output:
Ticker KOTAKBANK.NS
Price Open High low close Volu
2024-01-01 1906 1916 1891 1904. 1425

Ticker HDFCBANK.NS
Price Open High low close Volume
Date
2024-01-01 983.056 996.273 989.541 990.869 768379

Ticker ICICIBANK.NS.
Price Open High low close Volume
Date
2024-01-01 983.056 996.273 989.541 990.869 768

Code:

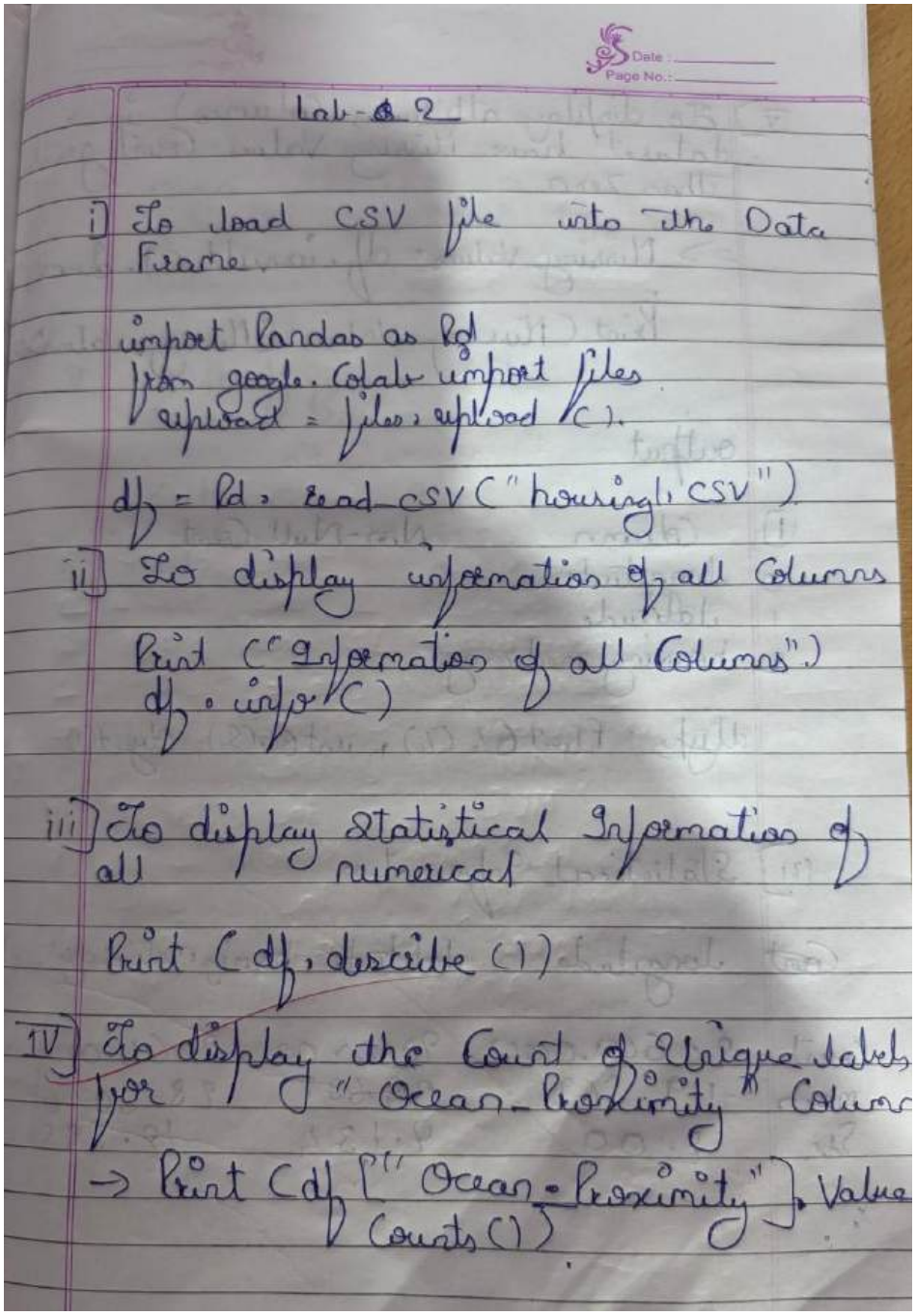
```
import yfinance as yf

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')

print("First 5 rows of the dataset:")

print(data.head())

print("\nShape of the dataset:")

print(data.shape)

print("\nColumn names:")

print(data.columns)

hdfc_data = data['HDFCBANK.NS']

print("\nSummary statistics for HDFC Bank:")

print(hdfc_data.describe())

hdfc_data['Daily Return'] = hdfc_data['Close'].pct_change()

icici_data = data['ICICIBANK.NS']

print("\nSummary statistics for ICICI Bank:")

print(icici_data.describe())

icici_data['Daily Return'] = icici_data['Close'].pct_change()

kotak_data = data['KOTAKBANK.NS']

print("\nSummary statistics for Kotak Mahindra Bank:")

print(kotak_data.describe())
```

```
kotak_data['Daily Return'] = kotak_data['Close'].pct_change()

plt.figure(figsize=(14, 10))

plt.subplot(3, 2, 1)

hdfc_data['Close'].plot(title="HDFC Bank - Closing Price")

plt.subplot(3, 2, 2)

hdfc_data['Daily Return'].plot(title="HDFC Bank - Daily Returns", color='orange')

plt.subplot(3, 2, 3)

icici_data['Close'].plot(title="ICICI Bank - Closing Price")

plt.subplot(3, 2, 4)

icici_data['Daily Return'].plot(title="ICICI Bank - Daily Returns", color='orange')

plt.subplot(3, 2, 5)

kotak_data['Close'].plot(title="Kotak Mahindra Bank - Closing Price")

plt.subplot(3, 2, 6)

kotak_data['Daily Return'].plot(title="Kotak Mahindra Bank - Daily Returns", color='orange')

plt.tight_layout()

plt.show()


hdfc_data.to_csv('hdfc_bank_data.csv')

icici_data.to_csv('icici_bank_data.csv')

kotak_data.to_csv('kotak_bank_data.csv')


print("\nHDFC Bank data saved to 'hdfc_bank_data.csv'.")

print("ICICI Bank data saved to 'icici_bank_data.csv'.")

print("Kotak Bank data saved to 'kotak_bank_data.csv'.")
```

Program 2

Demonstrate various data pre-processing techniques for a given dataset

Screenshot



Lab - 2

i] To load CSV file into the Data Frame.

```
import pandas as pd
from google.colab import files
upload = files.upload()
df = pd.read_csv("housing.csv")
```

ii] To display information of all Columns

```
Print ("Information of all Columns")
df.info()
```

iii] To display Statistical Information of all numerical

```
Print (df.describe())
```

IV] To display the Count of Unique labels for "Ocean_Proximity" Column

→ `Print (df["Ocean_Proximity"].value_counts())`

v) To display attributes (columns) in a dataset have Missing Values Count greater than zero

=> Missing_Values = df.isnull().sum()

Print (Missing_Values [Missing_Values > 0

output

[1] Column Non-Null Count

0 longitude.
1 latitude
2 housing_median_age.

dtypes: float64(4), int64(3), object(1)

[2] Statistical Information.

	longitude	latitude	housing_median_age
count	20640.000	20640.00	20640.000
mean	-119.569	35.631	28.6394
Std.	2.00	2.134	12.585

Lab-2.

Data Preprocessing

a) Housing.csv

b) import Pandas as Pd
file_Path = "housing.csv"
df = Pd.read_csv(file_Path)
Print(df.info())

c) Print(df.describe())

d) Print(df['Ocean_Proximity'].Value_Counts())

e) Missing_values = df.isnull().sum()
Cols_with_missing_vals = missing_vals[missing_vals > 0]
Print(Cols_with_missing_vals)

Output b) Data Columns (total 10 cols):
Colums Non-Null Count Dtype
0 Constint 20640 nonull float.

c) 8 numerical
Count 20640.00
Mean 3.87

d) OceanProximity -
Near Ocean 2658
Near Bay 2290

e) Attributes with Missing Values.
total bedrooms 207
dtype: int 64

1) Diabetes.csv
a) which columns in the Dataset missing Values? How to Handle them?
→ Missing Values are present in numerical Columns of Pregnant which are replaced by the Mean of the respective Columns

b) which Categorical Columns did you identify? How did you encode them?
=> No Categorical data, no encoding

c) what is the difference b/w MinMaxScaler & Standardization?
when would you one over the other?
→ Minmax Scaler transform data to a fixed range [0, 1] Range.

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

Standardization transform to have Zero Mean & Unit Variance

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

ii) Adult_income.csv.

@ which columns in the dataset has missing Values?
How to Handle them?

=> Missing Values are represented by "?" replacing with "NAN" for numerical Colums.
replace with mode

4) which Categorical Columns did you identify & How did you encode them?

-> Workclass, education, marital_status, Occupation, relationship, race, sex, native Country, income encoding method: label Encoding to Convert Categorical.

Code:

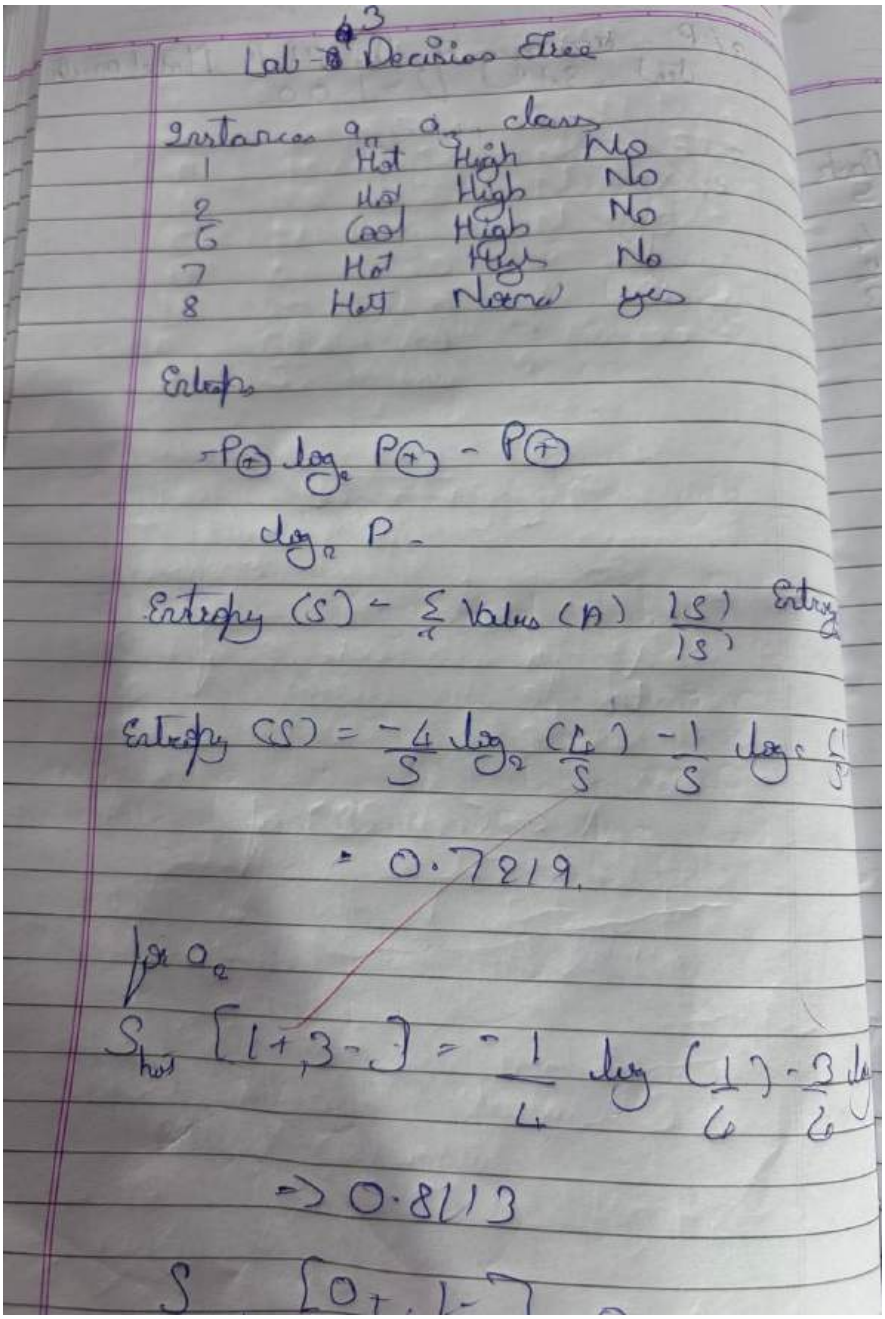
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder
import seaborn as sns
import matplotlib.pyplot as plt

diabetes_data = pd.read_csv('/content/Dataset of Diabetes .csv')
adult_income_data = pd.read_csv('/content/adult.csv')

print("Diabetes Dataset:")
print(diabetes_data.head())

print("\nAdult Income Dataset:")
print(adult_income_data.head())

diabetes_numerical_cols = diabetes_data.select_dtypes(include=[np.number]).columns
diabetes_categorical_cols = diabetes_data.select_dtypes(include=[object]).columns

diabetes_imputer_num = SimpleImputer(strategy='median')
diabetes_data[diabetes_numerical_cols] =
diabetes_imputer_num.fit_transform(diabetes_data[diabetes_numerical_cols])

diabetes_imputer_cat = SimpleImputer(strategy='most_frequent')
diabetes_data[diabetes_categorical_cols] =
diabetes_imputer_cat.fit_transform(diabetes_data[diabetes_categorical_cols])

adult_income_numerical_cols = adult_income_data.select_dtypes(include=[np.number]).columns
adult_income_categorical_cols = adult_income_data.select_dtypes(include=[object]).columns

adult_income_imputer_num = SimpleImputer(strategy='median')
adult_income_data[adult_income_numerical_cols] =
adult_income_imputer_num.fit_transform(adult_income_data[adult_income_numerical_cols])

adult_income_imputer_cat = SimpleImputer(strategy='most_frequent')
adult_income_data[adult_income_categorical_cols] =
adult_income_imputer_cat.fit_transform(adult_income_data[adult_income_categorical_cols])

categorical_columns_adult = adult_income_data.select_dtypes(include=['object']).columns
label_encoder = LabelEncoder()

for col in categorical_columns_adult:
    adult_income_data[col] = label_encoder.fit_transform(adult_income_data[col])
```

```
def detect_and_remove_outliers(df):
    numerical_df = df.select_dtypes(include=[np.number])
    Q1 = numerical_df.quantile(0.25)
    Q3 = numerical_df.quantile(0.75)
    IQR = Q3 - Q1
    return df[~((numerical_df < (Q1 - 1.5 * IQR)) | (numerical_df > (Q3 + 1.5 * IQR))).any(axis=1)]

diabetes_data_cleaned = detect_and_remove_outliers(diabetes_data)
adult_income_data_cleaned = detect_and_remove_outliers(adult_income_data)

min_max_scaler = MinMaxScaler()

diabetes_numerical_cols = diabetes_data_cleaned.select_dtypes(include=[np.number]).columns
diabetes_data_normalized = diabetes_data_cleaned.copy()

diabetes_data_normalized[diabetes_numerical_cols] =
min_max_scaler.fit_transform(diabetes_data_cleaned[diabetes_numerical_cols])

adult_income_numerical_cols =
adult_income_data_cleaned.select_dtypes(include=[np.number]).columns
adult_income_data_normalized = adult_income_data_cleaned.copy()

adult_income_data_normalized[adult_income_numerical_cols] =
min_max_scaler.fit_transform(adult_income_data_cleaned[adult_income_numerical_cols])

standard_scaler = StandardScaler()

diabetes_data_standardized = diabetes_data_cleaned.copy()
diabetes_data_standardized[diabetes_numerical_cols] =
standard_scaler.fit_transform(diabetes_data_cleaned[diabetes_numerical_cols])

adult_income_data_standardized = adult_income_data_cleaned.copy()
adult_income_data_standardized[adult_income_numerical_cols] =
standard_scaler.fit_transform(adult_income_data_cleaned[adult_income_numerical_cols])
```

Program 3

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

Screenshot:



Gain $(S, a_2) = 0.7219 - \frac{4}{5}(0.8113)$

$\Rightarrow 0.07286$

for a_3

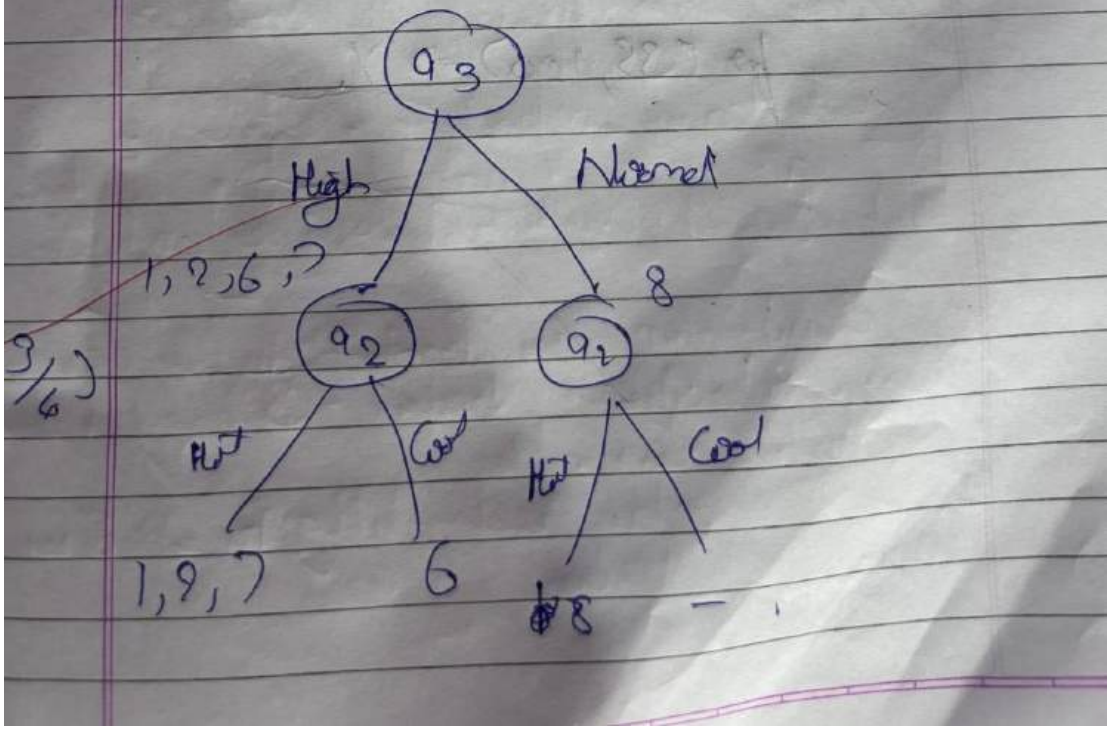
$S_{high}\ [0+, 4-] = 0$

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

Gain $(S, a_3) = 0.7219 - 0 - 0 \Rightarrow$

0.7219

a_3 is the root node. Since $IG(S, a_3)$ is highest



Code:

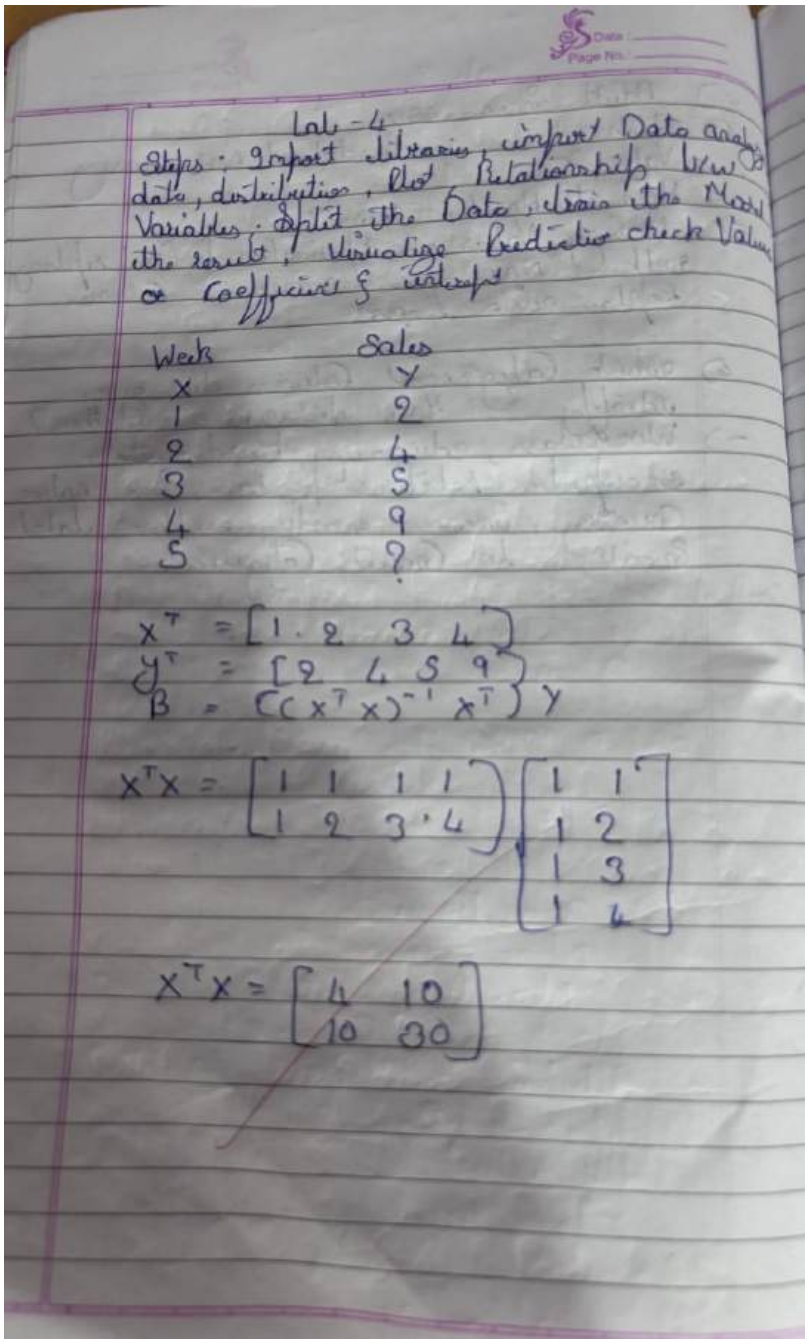
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, plot_tree
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report,
mean_absolute_error, mean_squared_error
from sklearn.preprocessing import LabelEncoder

iris = pd.read_csv("/content/iris (4).csv")
drug = pd.read_csv("/content/drug.csv")
petrol = pd.read_csv("/content/petrol_consumption.csv")

X_iris = iris.iloc[:, :-1]
y_iris = iris.iloc[:, -1]
X_train, X_test, y_train, y_test = train_test_split(X_iris, y_iris, test_size=0.2, random_state=42)

dtc = DecisionTreeClassifier()
dtc.fit(X_train, y_train)
y_pred = dtc.predict(X_test)

print("Decision Tree Classification for IRIS Dataset:")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))

X_drug = drug.iloc[:, :-1]
y_drug = drug.iloc[:, -1]

le = LabelEncoder()

for col in X_drug.select_dtypes(include=['object']).columns:
    X_drug[col] = le.fit_transform(X_drug[col])

X_train, X_test, y_train, y_test = train_test_split(X_drug, y_drug, test_size=0.2, random_state=42)

dtc = DecisionTreeClassifier()
dtc.fit(X_train, y_train)
y_pred = dtc.predict(X_test)

print("\nDecision Tree Classification for Drug Dataset:")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))

X_petrol = petrol.iloc[:, :-1]
```

```
y_petrol = petrol.iloc[:, -1]
X_train, X_test, y_train, y_test = train_test_split(X_petrol, y_petrol, test_size=0.2, random_state=42)

dtr = DecisionTreeRegressor()
dtr.fit(X_train, y_train)
y_pred = dtr.predict(X_test)

print("\nDecision Tree Regression for Petrol Consumption:")
print("Mean Absolute Error:", mean_absolute_error(y_test, y_pred))
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("Root Mean Squared Error:", np.sqrt(mean_squared_error(y_test, y_pred)))
```

Program 4

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Screenshot



$$(X^T X)^{-1} = \frac{1}{20}\begin{bmatrix} 30 & -10 \\ -10 & 4 \end{bmatrix}$$

$$(X^T X)^{-1} = \begin{bmatrix} 1.5 & -0.5 \\ -0.5 & 0.2 \end{bmatrix}$$

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

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

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

$$= \begin{bmatrix} -0.5 \\ 2.2 \end{bmatrix} \Rightarrow \begin{bmatrix} \text{Intercept} \\ \text{Slope} \end{bmatrix}$$

$y = \beta_0 + \beta_1 x + e \quad x = 5$

$y = (-0.5) + (2.2)(5) + e$

$y = -0.5 + 11 + e$

$Y = 10.5 + e$

18/9/25

Lab-04

Q)

Instance	a_1	a_2	a_3	Classification
1	T	Hot	High	No
2	T	H	H	N
3	F	H	H	Yes
4	F	Cool	Normal	Y
5	F	C	N	Y
6	T	C	H	N
7	T	H	H	N
8	T	H	N	Y
9	F	C	N	Y
10	F	C	H	Y

Solut. Attribute : a_1, Values = T, F

$S = [6+, 4-] = -\frac{6}{10}\log_2(6/10) - 4/10\log_2(4/10)$

$S_T = [1+, 4-] = -1/6\log_2(1/6) - 5/6\log_2(5/6)$

$= 0.72192$

Gain $(S, a_1) = S - \sum_{V \in (T,F)} \frac{|S_V|}{|S|}$ Entropy

$= 0.971 - (0.7192)(\frac{5}{10}) - 0$

$= 0.60994.$

a_2

$S = 0.9709$

$S_T = [2+ \; 3-] = 0.9709$

$S_T = [4+, 1-] = 0.72192$

$G(S, a_2) = 0.9709 - \frac{5}{10} * 0.9709 - S$

$a_3 = S$

$S = 0.9709$

$S_T = [2+, 4-] = 0.9182$

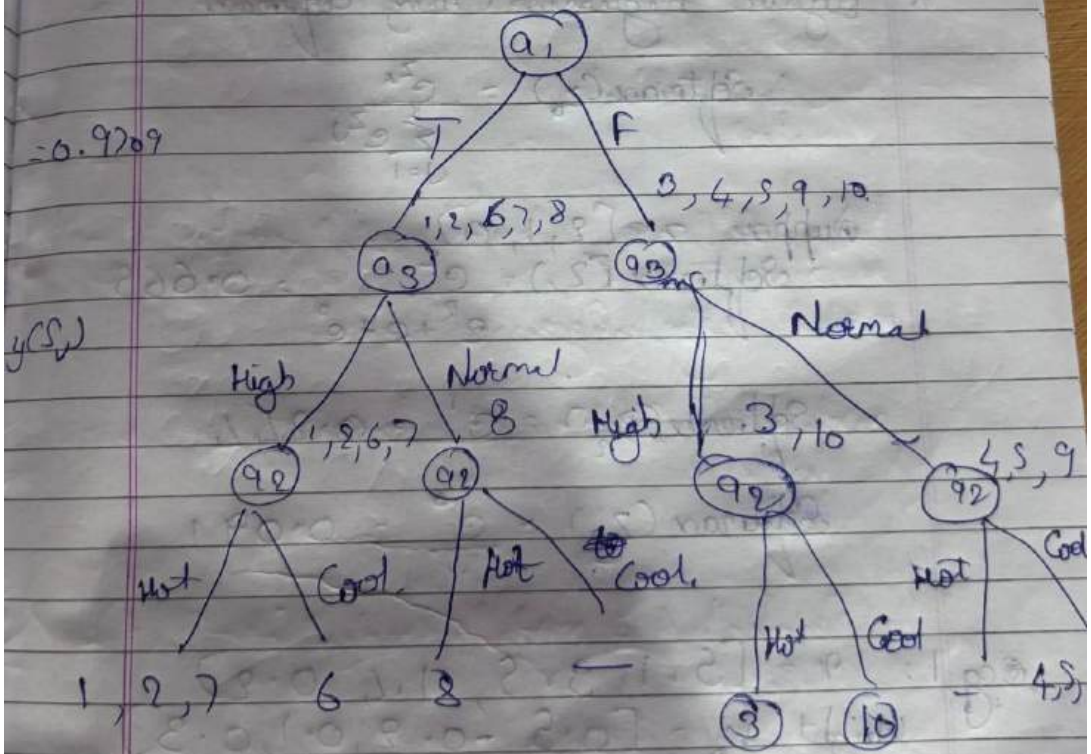
$S_F = [4+, 0-] = 0$

Gain $(S, a_3) = 0.9709 - 0.9182 \times 0.6 - 0$

$= 0.41998$

Max gain $\Rightarrow a_1$

Decision tree.



Code:

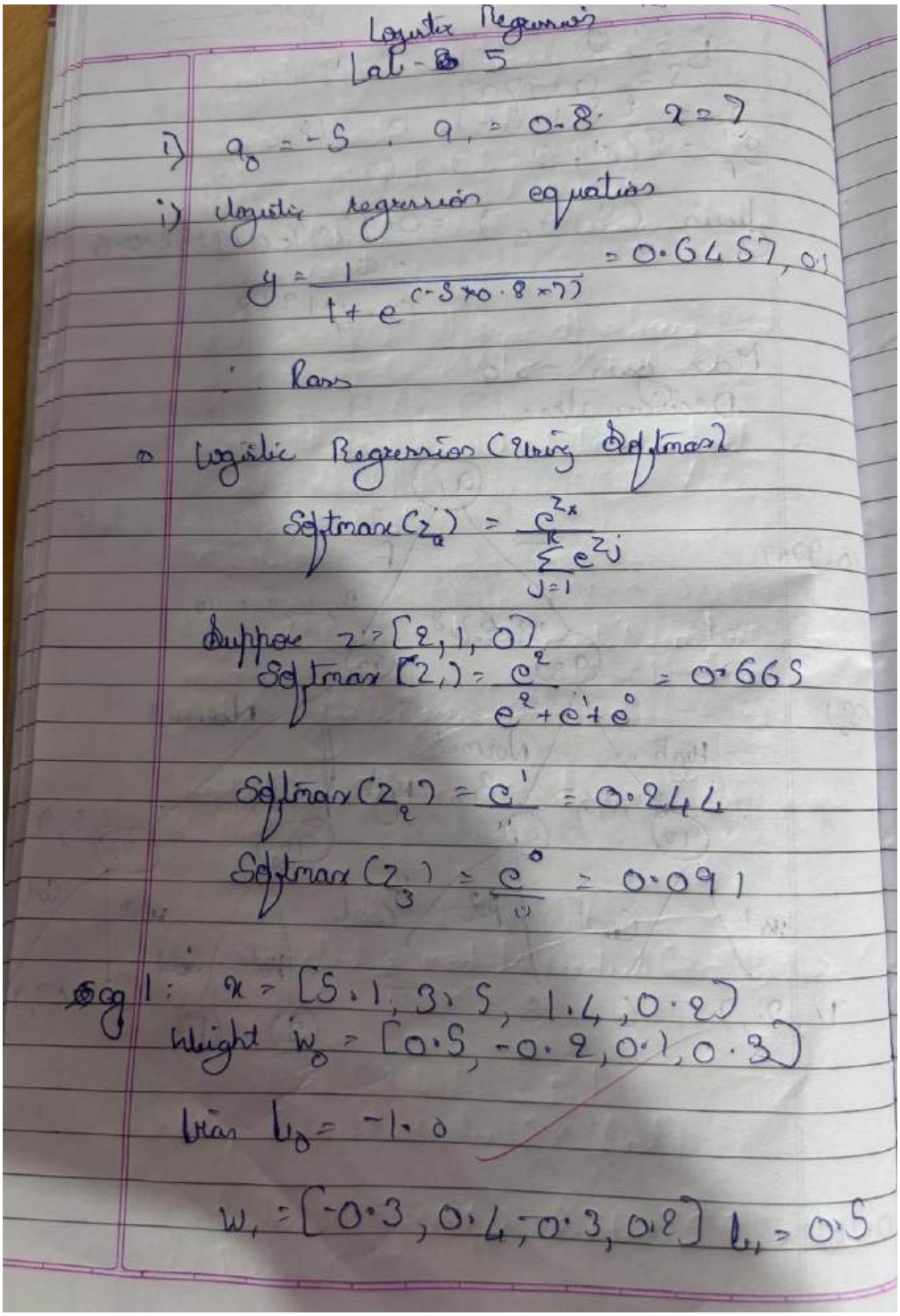
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_absolute_error
import matplotlib.pyplot as plt

hiring_data = pd.read_csv('hiring.csv')
print(hiring_data.head())
hiring_data = hiring_data.dropna()

experience_mapping = {
    'one': 1, 'two': 2, 'three': 3, 'four': 4, 'five': 5, 'six': 6, 'seven': 7, 'eight': 8,
    'nine': 9, 'ten': 10, 'eleven': 11, 'twelve': 12, 'thirteen': 13, 'fourteen': 14,
}

hiring_data['experience'] = hiring_data['experience'].replace(experience_mapping)
hiring_data['experience'] = pd.to_numeric(hiring_data['experience'], errors='coerce')

if hiring_data['experience'].isnull().any():
    print("Warning: There are still non-numeric values in the 'experience' column.")
    hiring_data = hiring_data.dropna(subset=['experience'])

X_hiring = hiring_data[['experience', 'test_score(out of 10)', 'interview_score(out of 10)']]
y_hiring = hiring_data['salary($)']

X_train_hiring, X_test_hiring, y_train_hiring, y_test_hiring = train_test_split(X_hiring, y_hiring,
test_size=0.2, random_state=42)

regressor_hiring = LinearRegression()
regressor_hiring.fit(X_train_hiring, y_train_hiring)

candidate_1 = np.array([[2, 9, 6]])
candidate_2 = np.array([[12, 10, 10]])

salary_1 = regressor_hiring.predict(candidate_1)
salary_2 = regressor_hiring.predict(candidate_2)

print(f"Predicted salary for candidate 1 (2 yr experience, 9 test score, 6 interview score):
{salary_1[0]}")
print(f"Predicted salary for candidate 2 (12 yr experience, 10 test score, 10 interview score):
{salary_2[0]}")
```

```
companies_data = pd.read_csv('/content/1000_Companies.csv')
print(companies_data.head())
companies_data = companies_data.dropna()

label_encoder = LabelEncoder()
companies_data['State'] = label_encoder.fit_transform(companies_data['State'])

X_companies = companies_data[['R&D Spend', 'Administration', 'Marketing Spend', 'State']]
y_companies = companies_data['Profit']

X_train_companies, X_test_companies, y_train_companies, y_test_companies =
train_test_split(X_companies, y_companies, test_size=0.2, random_state=42)

regressor_companies = LinearRegression()
regressor_companies.fit(X_train_companies, y_train_companies)

input_data = np.array([[91694.48, 515841.3, 11931.24, label_encoder.transform(['Florida'])[0]]])
predicted_profit = regressor_companies.predict(input_data)

print(f"Predicted profit for the given inputs (Florida State): {predicted_profit[0]}")

y_pred_hiring = regressor_hiring.predict(X_test_hiring)
mae_hiring = mean_absolute_error(y_test_hiring, y_pred_hiring)
print(f"Mean Absolute Error for Salary Prediction: {mae_hiring}")

y_pred_companies = regressor_companies.predict(X_test_companies)
mae_companies = mean_absolute_error(y_test_companies, y_pred_companies)
print(f"Mean Absolute Error for Profit Prediction: {mae_companies}")
```

Program 5

Build Logistic Regression Model for a given dataset

Screenshot



$w_2 = [0.2, -0.1, 0.6, -0.4]$, $b_2 = 0.2$

$z_1 = [(5.1 \times -0.3) + (3.5 \times 0.4) + (1.4 \times -0.5) + (0.2 \times 0.2)] + 0.5 =$
$= -0.29.$

$z_2 = 1.43$

$z_0 = w_0^T x + b_0$

$z_1 = w_1^T x + b_1$

$z_2 = w_2^T x + b_2.$

$P(y = 0/x) = \frac{e^{z_0}}{e^{z_0} + e^{z_1} + e^{z_2}} = 0.3671.$

$P(y = 1/x) = \frac{e^{z_1}}{\quad} = 0.0961$

$P(y = 2/x) = \frac{e^{z_2}}{\quad} = 0.5368$

Predict class

$\hat{y} = \text{arg max} \{ P(y = 0/x), P(y = 1/x), P(y = 2/x) \}$
(class 2)

$= 0.5368 \Rightarrow$ Virginica

Page No.:

Logistic Regression - Binary - Lab - 5

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

df = pd.read_csv("insurance_data.csv")
df.head()

plt.scatter(df.age, df.bought_insurance
, marker='+', color='red');

from sklearn.model_selection import train_
test_split
X_train.shape
X_test

from sklearn.linear_model import Logistic
Regression
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([[60]])
Y_predicted
```

model.coef_
model.intercept_

import math
def Sigmoid(x):
    return 1 / (1 + math.exp(-x))

def Prediction_function(age):
    z = 0.127 * age - 4.973 # 0.12740S
    63 ~ 0.0127 - 4.97335111 ~ -4.97

Y = Sigmoid(z)
return Y

age = 35
Prediction_function(age).

O/P

0.37 is less than 0.5 which means
Person with 35 will not buy the insure

Logistic Regression Multiclass

⇒ import Pandas as Pd.
from Sklearn.dataset import load_iris

iris = Pd.read.csv ("iris.csv")
iris.head()

X = iris.drop('Species', axis = 'Columns')
y = iris['Species']

X

Model = LogisticRegression(Multi_class = 'Multinomial')
model.fit(x_train, y_train)
y_pred = Model.Predict(X_test)
Accuracy = accuracy_score(y_test, y_pred)
Print("Accuracy of the MLR Model on the Set: {:.2f}".format(accuracy: .2f"))

Confusion_matrix = metrics.confusion_matrix(y_test, y_pred)

cm_display = Metrics.ConfusionMatrixDisplay(
Confusion_matrix = Confusion_Matrix, display_labels = ['Setosa', 'Versicolor', 'Virginica'])
cm_display.Plot()
Plt.show()

Code:

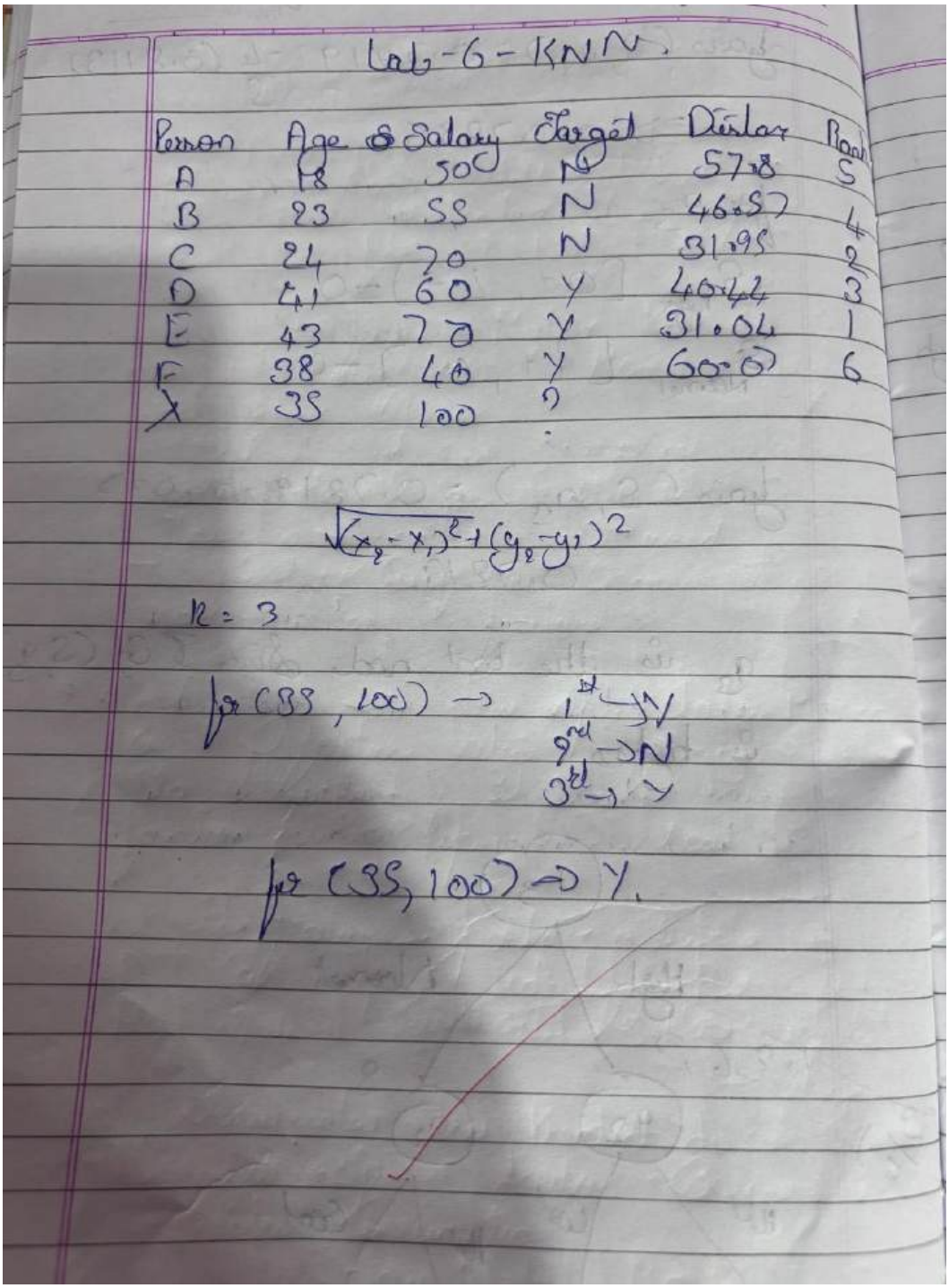
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix

file_path = 'HR_comma_sep.csv'
data = pd.read_csv(file_path)

print(data.info())

print(data.head())

print(data.describe())

plt.figure(figsize=(8, 5))
sns.countplot(x='salary', hue='left', data=data)
plt.title('Impact of Salary on Employee Retention')
plt.xlabel('Salary')
plt.ylabel('Count')
plt.legend(title='Employee Retention', labels=['Stayed', 'Left'])
plt.show()

plt.figure(figsize=(10, 6))
sns.countplot(x='Department', hue='left', data=data)
plt.title('Impact of Department on Employee Retention')
plt.xlabel('Department')
plt.ylabel('Count')
plt.legend(title='Employee Retention', labels=['Stayed', 'Left'])
plt.xticks(rotation=45)
plt.show()

data_encoded = pd.get_dummies(data, columns=['salary', 'Department'], drop_first=True)

print(data_encoded.info())

X = data_encoded.drop('left', axis=1)
y = data_encoded['left']

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)

logreg = LogisticRegression(max_iter=1000)

logreg.fit(X_train_scaled, y_train)

y_pred = logreg.predict(X_test_scaled)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of the Logistic Regression Model: {accuracy * 100:.2f}%")

cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False, xticklabels=['Stayed', 'Left'],
yticklabels=['Stayed', 'Left'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

Program 6

Build KNN Classification model for a given dataset.

Screenshot



For iris Dataset

1) How to choose the k value ? Demonstrate using accuracy rate & error rate.

∴ k value choosing by the rate which accuracy rate is high & error rate is low.

2) For Diabetes Data Set what is the purpose of feature scaling? How to perform it ?

=> To Prevent Biased & Ensures all features

=) By standardizing values to make it have 0 mean & unit variance from Sklearn. Preprocessing import Standard Scale = StandardScale ();
X Scale = Scale. fit_transform (X)

Lab-6 KNN code

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

# load
iris = pd.read_csv('iris.csv')
X_iris = iris.iloc[:, 1:-1]
y_iris = iris.iloc[:, -1]
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)

# train
knn_iris = KNeighborsClassifier(n_neighbors=5)
knn_iris.fit(X_train_iris, y_train_iris)
y_pred_iris = knn_iris.predict(X_test_iris)

# Result:
print("IRIS Dataset")
print("Accuracy Score:", accuracy_score(y_
test_iris, y_pred_iris))
("Confusion matrix:\n", confusion_matrix
y_test_iris, y_pred_iris))
print("classification:\n", classification_
report(y_test_iris, y_pred_iris))
```

Code:

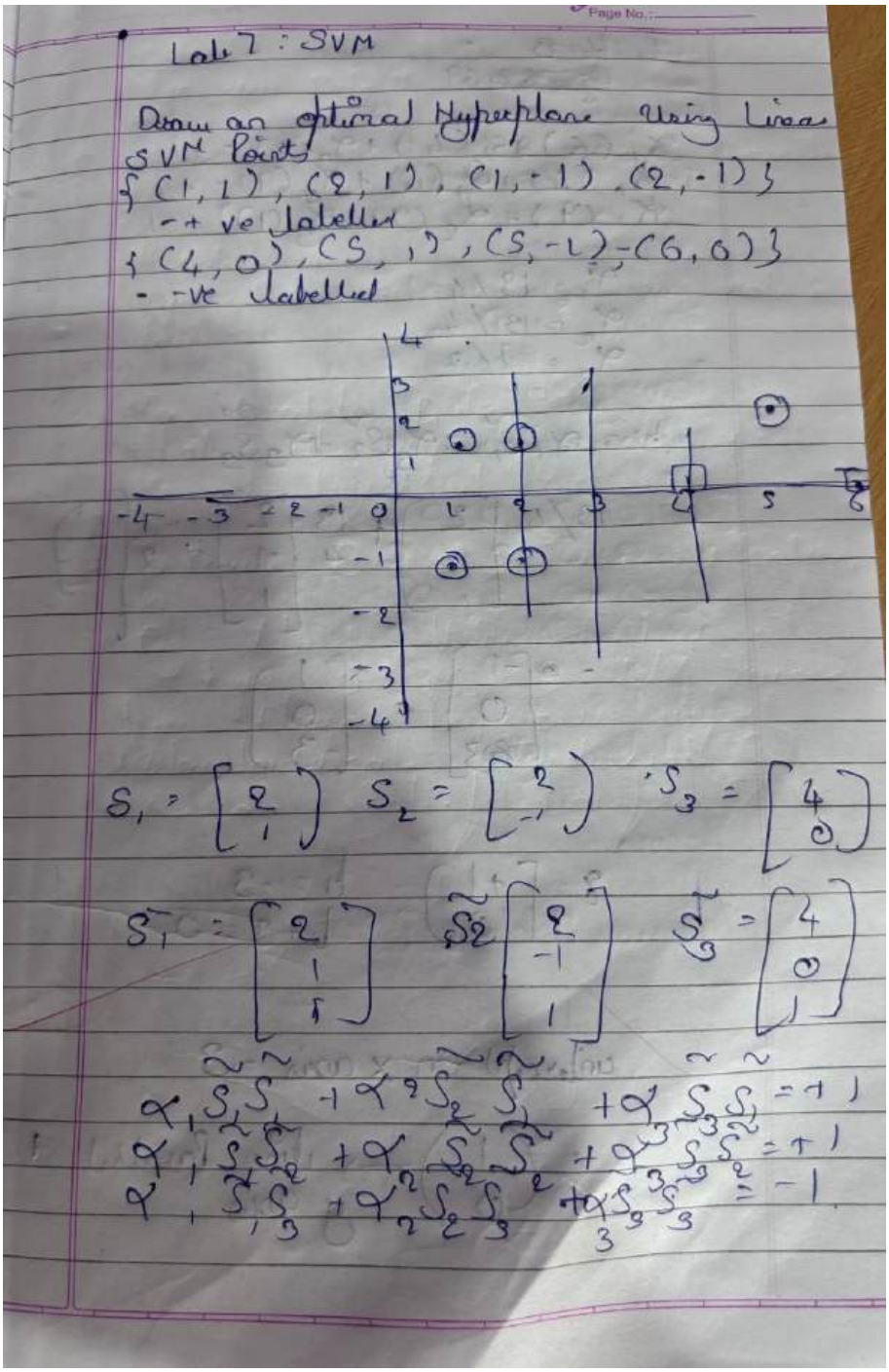
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot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

iris_df = pd.read_csv('/content/iris (3).csv')

print(iris_df.head())

X_iris = iris_df.drop(columns=['species'])
y_iris = iris_df['species']

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

scaler = StandardScaler()
X_train_iris = scaler.fit_transform(X_train_iris)
X_test_iris = scaler.transform(X_test_iris)

knn_iris = KNeighborsClassifier(n_neighbors=3)

knn_iris.fit(X_train_iris, y_train_iris)

y_pred_iris = knn_iris.predict(X_test_iris)

accuracy_iris = accuracy_score(y_test_iris, y_pred_iris)
print(f"Accuracy on Iris test data: {accuracy_iris * 100:.2f}%")

cm_iris = confusion_matrix(y_test_iris, y_pred_iris)
sns.heatmap(cm_iris, annot=True, fmt="d", cmap="Blues", xticklabels=knn_iris.classes_,
yticklabels=knn_iris.classes_)
plt.title("Confusion Matrix for Iris Dataset")
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

print("Classification Report for Iris Dataset:")
print(classification_report(y_test_iris, y_pred_iris))

diabetes_df = pd.read_csv('diabetes.csv')
print(diabetes_df.head())
```

```
X_diabetes = diabetes_df.drop(columns=['Outcome'])
y_diabetes = diabetes_df['Outcome']

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)

scaler = StandardScaler()
X_train_diabetes = scaler.fit_transform(X_train_diabetes)
X_test_diabetes = scaler.transform(X_test_diabetes)

knn_diabetes = KNeighborsClassifier(n_neighbors=5)

knn_diabetes.fit(X_train_diabetes, y_train_diabetes)

y_pred_diabetes = knn_diabetes.predict(X_test_diabetes)

accuracy_diabetes = accuracy_score(y_test_diabetes, y_pred_diabetes)
print(f"Accuracy on Diabetes test data: {accuracy_diabetes * 100:.2f}%")

cm_diabetes = confusion_matrix(y_test_diabetes, y_pred_diabetes)
sns.heatmap(cm_diabetes, annot=True, fmt="d", cmap="Blues", xticklabels=knn_diabetes.classes_,
yticklabels=knn_diabetes.classes_)
plt.title("Confusion Matrix for Diabetes Dataset")
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

print("Classification Report for Diabetes Dataset:")
print(classification_report(y_test_diabetes, y_pred_diabetes))
```

Program 7

Build Support vector machine model for a given dataset

Screenshot



Page No.:

69

$\alpha_1(6) + \alpha_2(4) + \alpha_3(9) = +1$
$\alpha_1(4) + \alpha_2(6) + \alpha_3(9) = +1$
$\alpha_1(9) + \alpha_2(9) + \alpha_3(17) = -1$

$\alpha_1 = 13/4$
$\alpha_2 = 13/4$
$\alpha_3 = -7/2$

$w = \alpha_1 \tilde{s_1} + \alpha_3 \tilde{s_2} + \alpha_3 \tilde{s_3}$

$$= 13/4 \begin{bmatrix} 2 \\ 1 \end{bmatrix} + \frac{13}{4} \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix} + \frac{-7}{2} \begin{bmatrix} 4 \\ 1 \end{bmatrix}$$

$$= -\begin{bmatrix} -1 \\ 0 \\ 3 \end{bmatrix} = -\begin{bmatrix} 1 \\ 0 \\ -3 \end{bmatrix}$$

$w = \begin{bmatrix} +1 \\ 0 \end{bmatrix}$ $\quad b = -3$
$\quad b + 3 = 0$

intercept on x axis $= 3$

$= \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ → line parallel to y axis.

Lab 7 Code

Iris Dataset

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix

# Load Iris Dataset
iris_df = pd.read_csv("iris.csv")

X_iris = iris_df.drop("Species", axis=1)
y_iris = iris_df["Species"]

# Encode target labels
label_encoder_iris = LabelEncoder()
y_iris_encoded = label_encoder_iris.fit_transform(y_iris)

# Split
X_train_iris, X_test_iris, y_train_iris, y_test_iris = train_test_split(X_iris, y_iris_encoded, test_size=0.2, random_state=42)

# train and evaluate linear SVM
svm_linear = SVC(kernel='linear')
svm_linear.fit(X_train_iris, y_train_iris)
y_pred_linear = svm_linear.predict(X_test_iris)
```

```
Print ("Linear Kernel Accuracy:", accuracy_score
    (y_test_iris, y_pred_linear))
Print ("Confusion Matrix:\n", Confusion_Matrix)

Train & Evaluate RBF SVM

SVM_rbf = SVC(kernel='rbf')
SVM_rbf.fit(X_train_iris, y_train_iris)
y_pred_rbf = SVM_rbf.predict(X_test_iris)

Print(accuracy_score(y_test_iris, y_pred_rbf))
Print(Confusion_matrix(y_test_iris, y_pred_rbf))
```

Output

Linear Kernel Accuracy: 1.0
Confusion Matrix:

```
[[10  0  0]
 [ 0  9  0
  0  11  0]]
```

RBF Kernel accuracy 1.0
Confusion Matrix:

```
[[10  0  0]
 [ 0  9  0
  0  0  11]]
```

Code:

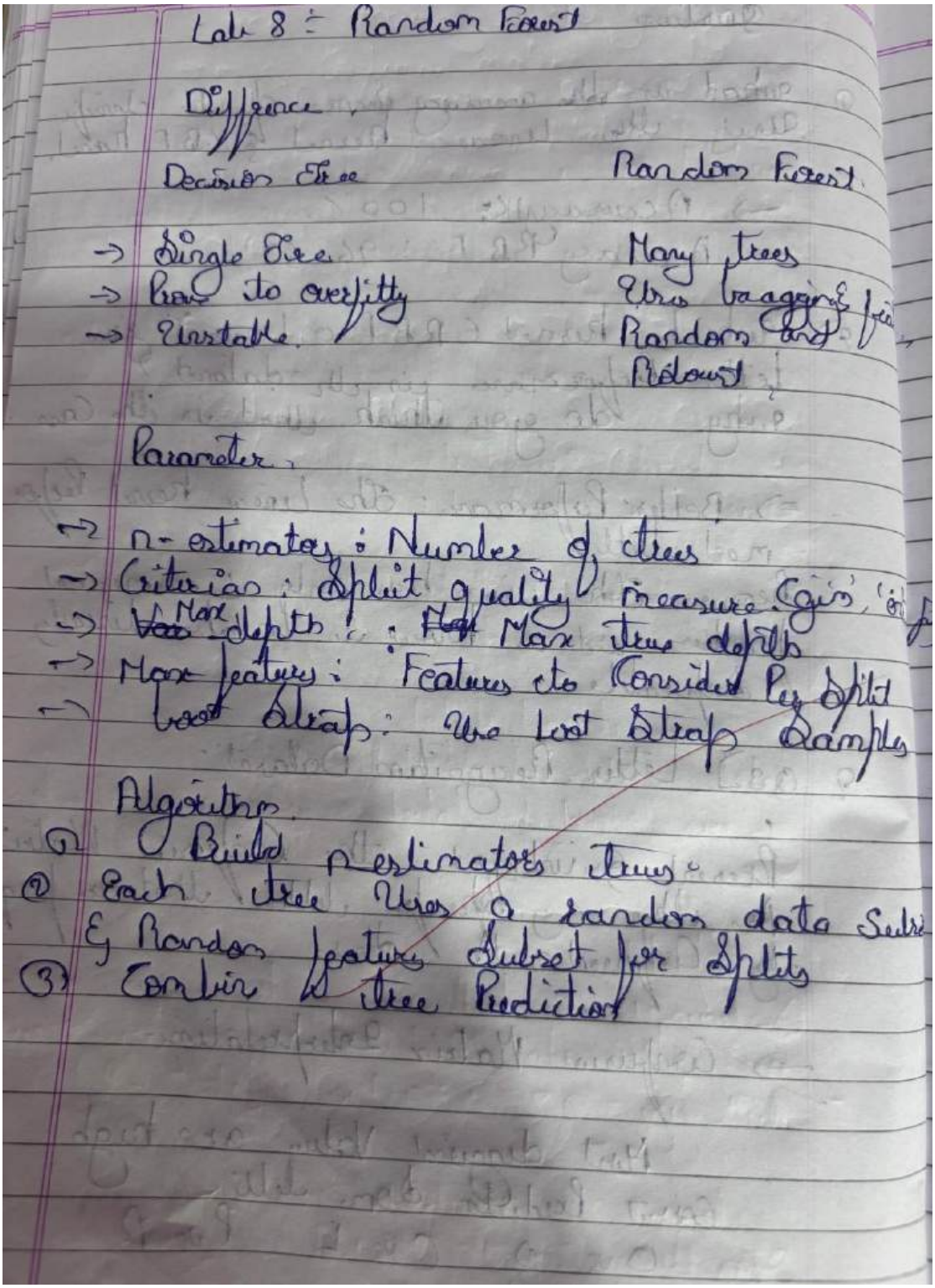
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, roc_auc_score, roc_curve
from sklearn.preprocessing import LabelEncoder, label_binarize
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

df = pd.read_csv("/content/letter-recognition.csv")

top_classes = df['letter'].value_counts().head(5).index.tolist()
df = df[df['letter'].isin(top_classes)]

X = df.iloc[:, 1:]
y = df.iloc[:, 0]

label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

y_bin = label_binarize(y_encoded, classes=np.unique(y_encoded))
n_classes = y_bin.shape[1]

X_train, X_test, y_train, y_test_bin = train_test_split(X, y_bin, test_size=0.2, random_state=42)

svm_model = SVC(kernel='linear', probability=True)
svm_model.fit(X_train, y_train.argmax(axis=1))
y_score = svm_model.predict_proba(X_test)

y_pred = svm_model.predict(X_test)
y_true = y_test_bin.argmax(axis=1)

print("Accuracy:", accuracy_score(y_true, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_true, y_pred))

plt.figure()
for i in range(n_classes):
    fpr, tpr, _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    auc = roc_auc_score(y_test_bin[:, i], y_score[:, i])
    plt.plot(fpr, tpr, label=f"{label_encoder.inverse_transform([i])[0]} AUC={auc:.2f}")
plt.plot([0, 1], [0, 1], 'k--')
plt.title("ROC Curve (Top 5 Classes)")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend(loc="lower right")
plt.tight_layout()
```

```
plt.show()

macro_auc = roc_auc_score(y_test_bin, y_score, average="macro")
print("Macro AUC Score:", macro_auc)
```

Program 8

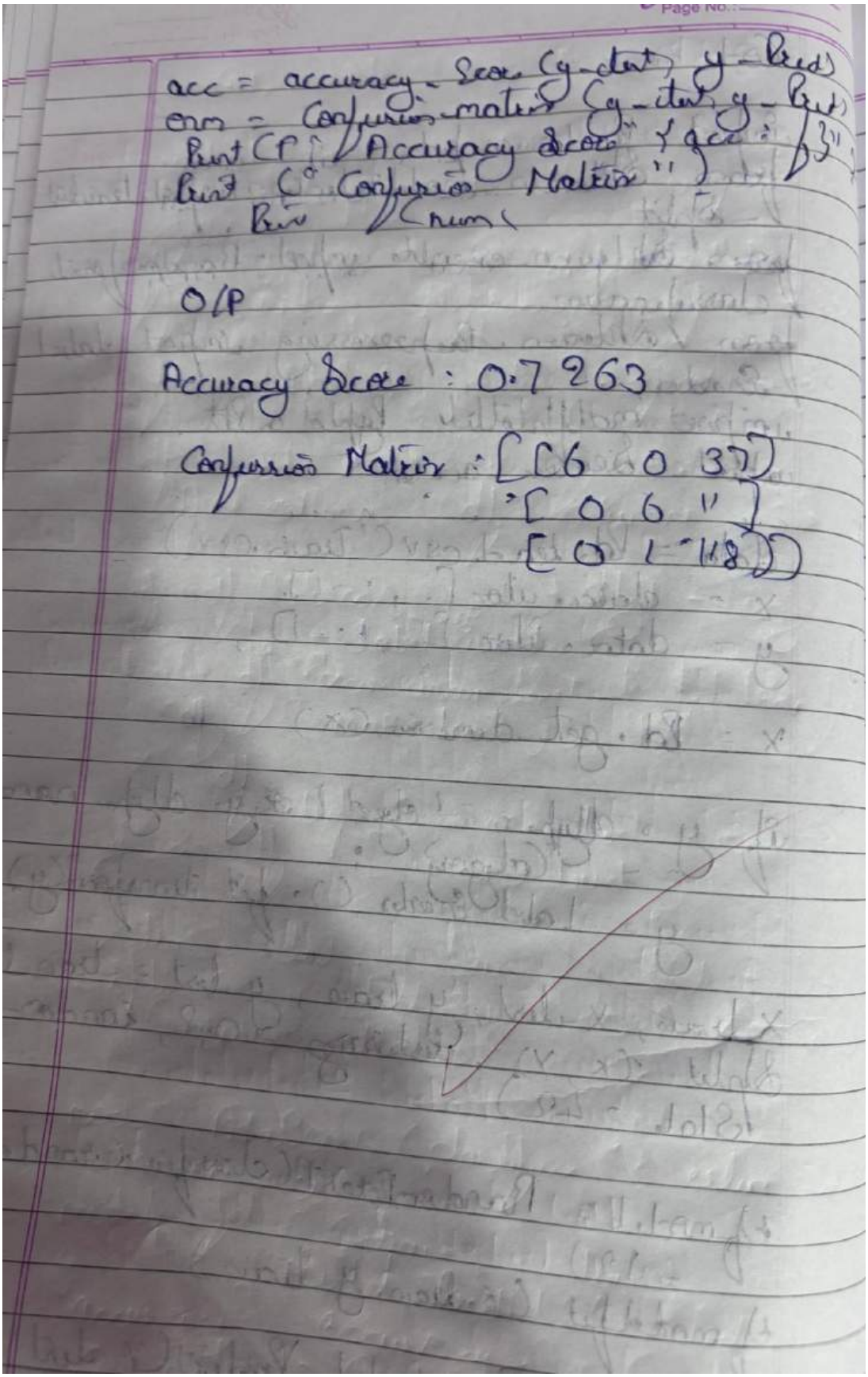
Implement Random forest ensemble method on a given dataset.

Screenshot



Code :-

```
import Pandas as pd.
from Sklearn.model_selection import train_test
_split
from Sklearn.ensemble import RandomForest
classifier.
from Sklearn.Preprocessing import label
Encoder.
import matplotlib.pyplot as plt.
import Seaborn as sn.

data = Pd.read_csv('Train.csv')
x = data.iloc[:, :-1].
y = data.iloc[:, :-1]

x = Pd.get_dummies(x)

if y.dtype == 'object' or y.dtype.name
== 'category':
    y = LabelEncoder().fit_transform(y)

x_train, x_test, y_train, y_test = train_test
_split(x, y), test_size = 0.2, random_
state = 42)

clf_model = RandomForest(classifier.(random_s
= 17)
clf_model.fit(x_train, y_train)

y_pred = clf_model.Predict(x_test)
```



```
acc = accuracy_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)
print("Accuracy Score :", acc)
print("Confusion Matrix :")
print(cm)
```

O/P

```
Accuracy Score : 0.7263
Confusion Matrix : [[ 6  0 37]
                    [ 0  6  1]
                    [ 0  1 118]]
```

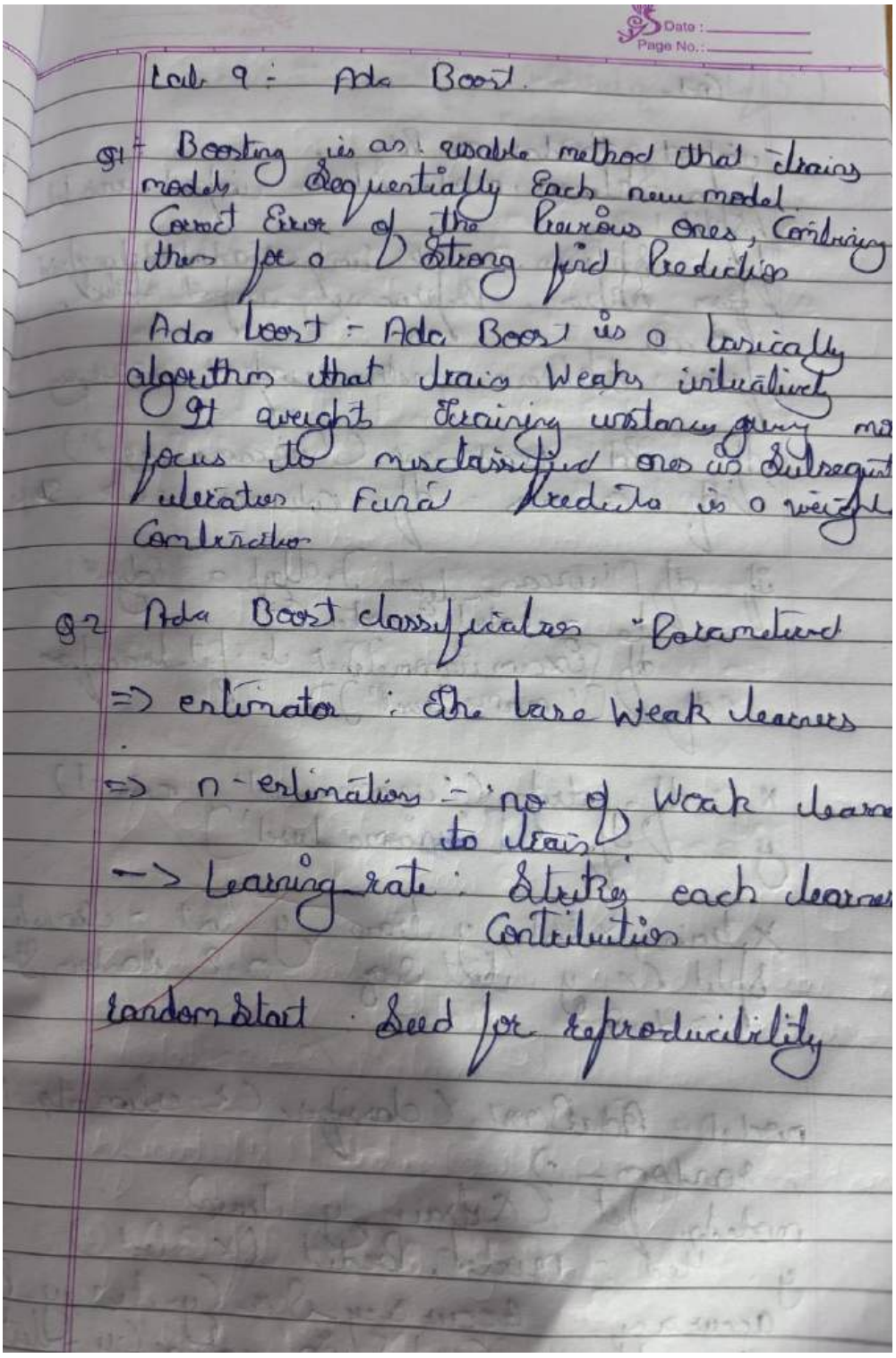
Code:

```
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
from sklearn import preprocessing

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

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

for column in X.columns:
    if X[column].dtype == 'object':
      le = preprocessing.LabelEncoder()
      X[column] = le.fit_transform(X[column])

if y.dtype == 'object':
  le = preprocessing.LabelEncoder()
  y = le.fit_transform(y)

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

rf_classifier = RandomForestClassifier(random_state=42)
rf_classifier.fit(X_train, y_train)

y_pred = rf_classifier.predict(X_test)

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

print(f"Accuracy: {accuracy}")
print(f"Confusion Matrix:\n{conf_matrix}")
```

Program 9

Implement Boosting ensemble method on a given dataset.

Screenshot



Code :

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, confusion_matrix

df = pd.read_csv("income.csv")
df.columns = df.columns.str.strip()

if df['income_level'].dtype == 'object':
    le = LabelEncoder()
    df['Exam income label'] = le.fit_transform(df['income_level'])

X = df.drop('income_level', axis=1)
y = df['income_level']

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

model = AdaBoostClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
```

Print ("Accuracy = ", accuracy : 4/5").
Print ("Confusion matrix").
Print (Conf_matrix).

O/P.

Accuracy : 0.8328

Confusion Matrix :

[[217 297
 [336 1019]]

Code:

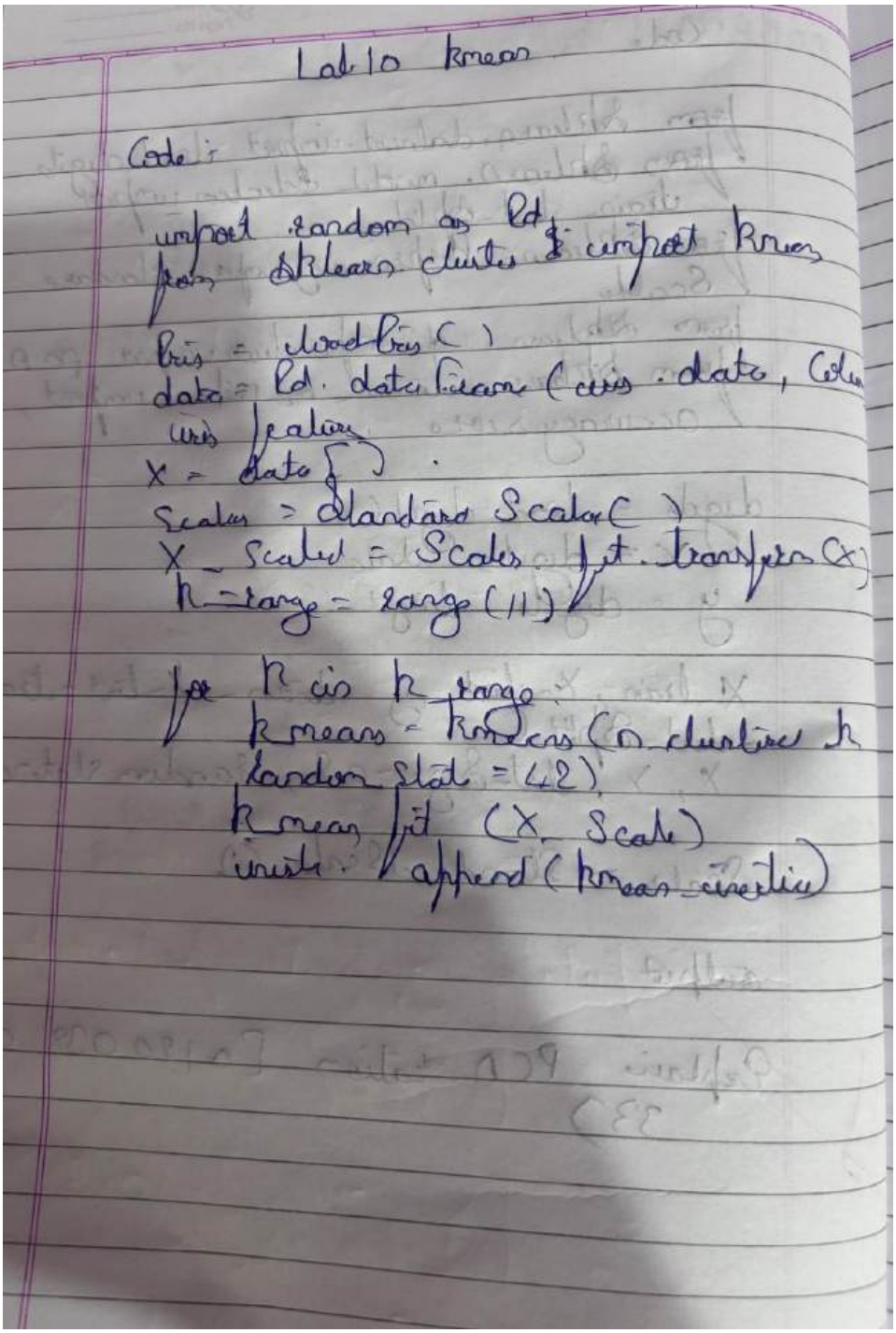
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

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

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

results = []

n_estimators_list = [10, 50, 100]
learning_rates = [0.01, 0.1, 1]

for n in n_estimators_list:
    for lr in learning_rates:
        tree_base = DecisionTreeClassifier(max_depth=1)
        model = AdaBoostClassifier(estimator=tree_base, n_estimators=n, learning_rate=lr,
random_state=42)
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        acc = accuracy_score(y_test, y_pred)
        results.append({
            'Base': 'DecisionTree',
            'n_estimators': n,
            'learning_rate': lr,
            'Accuracy': acc
        })

for n in n_estimators_list:
    for lr in learning_rates:
        log_reg_base = LogisticRegression(max_iter=1000)
        model = AdaBoostClassifier(estimator=log_reg_base, n_estimators=n, learning_rate=lr,
random_state=42)
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        acc = accuracy_score(y_test, y_pred)
        results.append({
            'Base': 'LogisticRegression',
```

```
            'n_estimators': n,
            'learning_rate': lr,
            'Accuracy': acc
        })

results_df = pd.DataFrame(results)
print(results_df)

import seaborn as sns
plt.figure(figsize=(12, 6))
sns.barplot(x='n_estimators', y='Accuracy', hue='Base', data=results_df, ci=None)
plt.title('AdaBoost Accuracy with Different Estimators and n_estimators')
plt.show()
```

Program 10

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

Screenshot



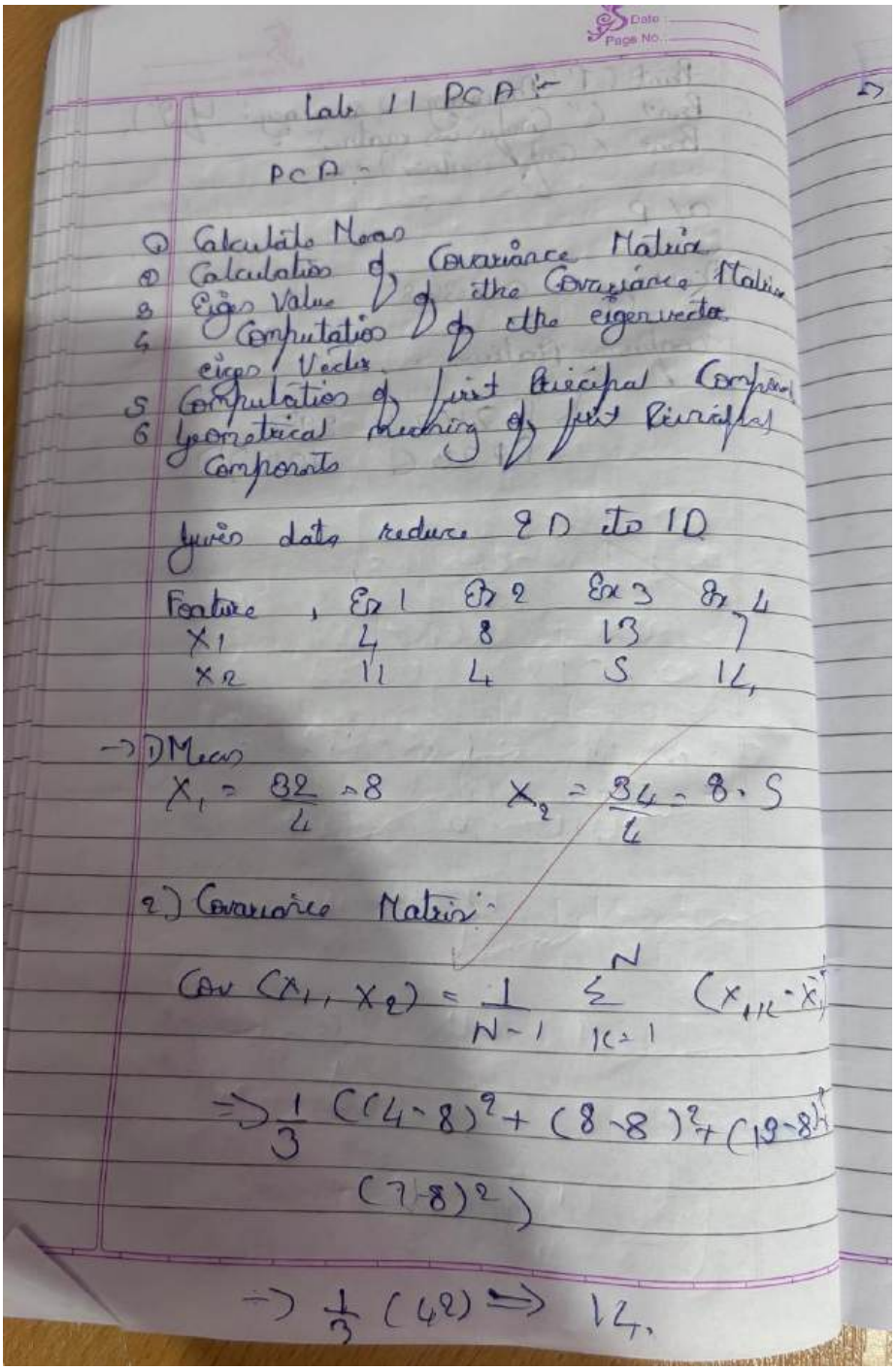
Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

data = {
    'Name': [f'Person_{i+1}' for i in range(50)],
    'Age': np.random.randint(18, 70, size=50),
    'Income': np.random.randint(20000, 120000, size=50)
}

df = pd.DataFrame(data)

df.to_csv('income.csv', index=False)

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

X = df[['Age', 'Income']]

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

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

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

plt.plot(k_range, sse, marker='o')
plt.title('SSE vs Number of Clusters')
plt.xlabel('Number of Clusters')
plt.ylabel('Sum of Squared Errors (SSE)')
plt.show()

optimal_k = 3
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
kmeans.fit(X_train)
y_pred = kmeans.predict(X_test)

print(f'Predicted Clusters for Test Data: {y_pred}')
```

Program 11

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

Screenshot



Date :
Page No. :

$$\Rightarrow \text{Cov}(X_1, X_2) = \frac{1}{N-1} \sum_{k=1}^{N} (X_{1k} - \bar{X}_1)(X_{2k} - \bar{X}_2)$$

$$= \frac{1}{3}((4-8)(11-8.5) + (8-8)(6-8.5) + (13-8)(5-8.5) + (7-8)(14-8.5))$$

$$\Rightarrow -11$$

$$\rightarrow \text{Cov}(X_2, X_1) = -11$$

$$\rightarrow \text{Cov}(X_2, X_2) = \frac{1}{N-1} \sum_{k=1}^{N} (X_{2k} - \bar{X}_2)^2$$

$$\Rightarrow \frac{1}{3}(11-8)^2 + (6-8.5)^2 + (5-8.5)^2 + (14-8.5)^2$$

$$\Rightarrow 23$$

3) Eigen Value of Covariance Matrix

$$S = \begin{bmatrix} \text{Cov}(X_1, X_1) & \text{Cov}(X_1, X_2) \\ \text{Cov}(X_2, X_1) & \text{Cov}(X_2, X_2) \end{bmatrix}$$

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

$$0 = \det |S - \lambda I|$$

$$|S - \lambda I| \Rightarrow \begin{vmatrix} 14-\lambda & -11 \\ -11 & 23-\lambda \end{vmatrix}$$

$\Rightarrow (14-\lambda)\times(23-\lambda) - (-11)\times(-11)$

$= 322 - 14\lambda - 23\lambda + \lambda^2 - 121$

$\Rightarrow \lambda^2 - 37\lambda + 201$

$\lambda_1 = 30.38, \quad \lambda_2 = 6.616$

4) $0 = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$

$\begin{bmatrix} 0 \\ 0 \end{bmatrix} = (S - \lambda I)$

$\Rightarrow \begin{bmatrix} 14-\lambda_1 & -11 \\ -11 & 23-\lambda_1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$

$\Rightarrow \begin{bmatrix} (14-\lambda_1)v_1 - 11v_2 \\ -11v_1 + (23-\lambda_1)v_2 \end{bmatrix}$

$\Rightarrow (14-\lambda_2)v_1 - 11v_2 = 0$

$(14-\lambda_1)v_1 = 11v_2$

$\frac{v_1}{11} = \frac{v_2}{14-\lambda_1} \Rightarrow 1$

$v_1 = 11t \quad v_2 = (14-\lambda_2)t$

Taking $t = 1$

$v_1 = \begin{bmatrix} 11 \\ 14-\lambda_1 \end{bmatrix}$

Eigen Vector

$$\|V_1\| = \sqrt{11^2 + (14 - \lambda_1)^2}$$

$$= \sqrt{11^2 + (14 - 30.38)^2}$$

$$\Rightarrow 19.7348$$

Unit Eigenvector corresponding to λ_1 is

$$e_1 = \begin{bmatrix} 11/\|V_1\| \\ (14 - \lambda_1)/\|V_1\| \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} 11/19.7348 \\ (14 - 30.38)/19.734 \end{bmatrix}$$

$$e_1 \Rightarrow \begin{bmatrix} 0.5574 \\ 0.8300 \end{bmatrix}$$

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

Computation of 1st Principal Component

$$\begin{bmatrix} X_1K \\ X_2K \end{bmatrix}$$

$$e_1^T \begin{bmatrix} X_1K - \bar{X}_1 \\ X_2K - \bar{X}_2 \end{bmatrix} = [0.5574 -$$

$$0.8303 \Big] \begin{bmatrix} X_1K - \bar{X}_1 \\ X_2K - \bar{X}_2 \end{bmatrix}$$

$\Rightarrow 0.8574 (x_{1k} - \bar{x}_1) - 0.8303$

$(x_{2k} - \bar{x}_2)$

Step 5: $\begin{bmatrix} x_{11} \\ x_{21} \end{bmatrix} = \begin{bmatrix} 4 \\ 11 \end{bmatrix}$

$\begin{bmatrix} 0.8574 & -0.8303 \end{bmatrix} \begin{bmatrix} x_{11} - \bar{x}_1 \\ x_{21} - \bar{x}_2 \end{bmatrix}$

$= 0.8574 (x_{11} - \bar{x}_1)$

$- 0.8303 (x_{21} - \bar{x}_2)$

$\Rightarrow 0.8574 (4 - 8) - 0.8303 (11 - 8)$

$\Rightarrow -4.30585$

x_1	4	8	13	7
x_2	11	4	5	14
Fun	-4.3058	3.73	5.68	-5.123

PC

Code

```
from sklearn.datasets import load_digits
from sklearn.model_selection import
    train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.metrics import
    accuracy_score

digits = load_digits()
X = digits.data
y = digits.target

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

Scaler = StandardScaler()
```

output

```
Explain PCA ratios [0.12038 0.0883
    337)
```

Code:

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.decomposition import PCA
from scipy import stats

df = pd.read_csv('heart (2).csv')

z_scores = np.abs(stats.zscore(df.select_dtypes(include=[np.number])))
df_no_outliers = df[(z_scores < 3).all(axis=1)]

df_cleaned = df_no_outliers.copy()
for col in df_cleaned.select_dtypes(include='object').columns:
    df_cleaned[col] = LabelEncoder().fit_transform(df_cleaned[col])

X = df_cleaned.drop('HeartDisease', axis=1)
y = df_cleaned['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,
stratify=y)

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

print("Accuracy without 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}")

pca = PCA(n_components=5)
X_pca = pca.fit_transform(X_scaled)
X_train_pca, X_test_pca, y_train, y_test = train_test_split(X_pca, y, test_size=0.2, random_state=42,
stratify=y)
```

```
print("\nAccuracy with PCA:")
for name, model in models.items():
    model.fit(X_train_pca, y_train)
    y_pred = model.predict(X_test_pca)
    acc = accuracy_score(y_test, y_pred)
    print(f"{name}: {acc:.4f}")
```