VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

AKANKSHA SINGA (1BM22CS027)

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 **Akanksha Singa(1BM22CS027),** 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.

M Lakshmi Neelima Assistant Professor Department of CSE, BMSCE	Dr. Kavitha Sooda Professor & HOD Department of CSE, BMSCE

Index

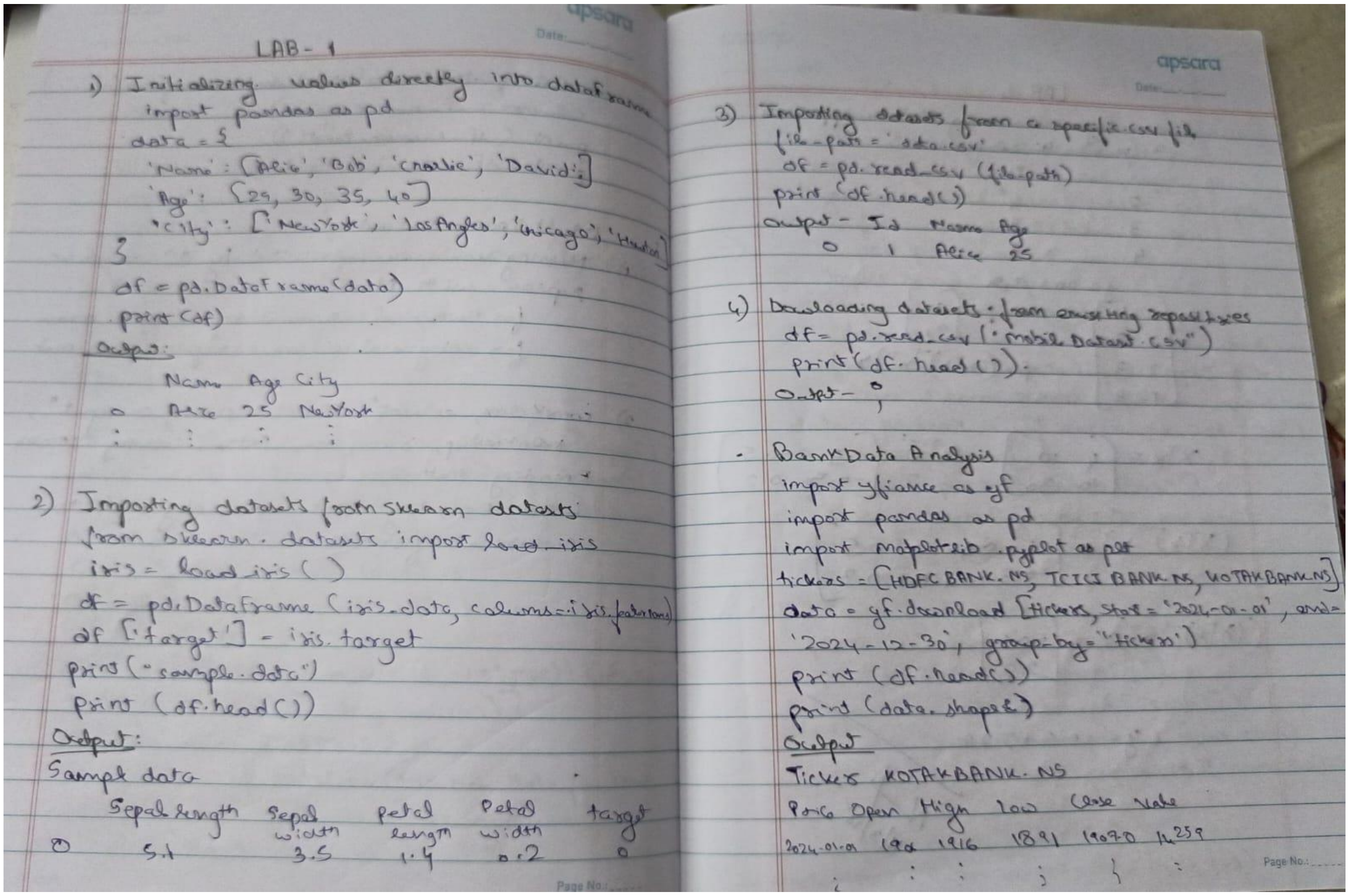
Github Link:

https://github.com/Akanksha-singa/6A_ML_LAB_B2

Program 1

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

Screenshot:



Code:

```
import pandas as pd
data = {
'Name': ['Alice', 'Bob', 'Charlie', 'David'],
'Age': [25, 30, 35, 40],
'City': ['New York', 'Los Angeles', 'Chicago', 'Houston']
 }
df = pd.DataFrame(data)
print("Sample data:")
print(df.head())
from sklearn.datasets import load_iris
iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target
print("Sample data:")
```

```
print(df.head())
file_path = 'data.csv'
df = pd.read_csv(file_path)
print("Sample data:")
print(df.head())
print("\n")
file_path = 'mobiles-dataset-2025.csv'
df = pd.read_csv(file_path, encoding='latin-1') # or 'cp1252' or other suitable
encoding  print("Sample data:")
print(df.head())
import pandas as pd

data = {
'USN': ['IS001','IS002','IS003','IS004','IS005'],
'Name': ['Alice', 'Bob', 'Charlie', 'David','Eve'],
'Marks': [25, 30, 35, 40,45]
 }

df = pd.DataFrame(data)
print("Sample data:")
print(df.head())
from sklearn.datasets import load_diabetes
iris = load_diabetes()
df = pd.DataFrame(iris.data, columns=iris.feature_names)

print("Sample data:")
print(df.head())
file_path = 'sample_sales_data.csv'
df = pd.read_csv(file_path)
print("Sample data:")
print(df.head())
print("\n")

df = pd.read_csv("/content/dataset-of-diabetes .csv",encoding='latin-1')
print("Sample data:")
print(df.head())
print("\n")


df =pd.read_csv('sample_sales_data.csv')
print("Sample data:")
print(df.head())

df.to_csv('output.csv',index=False)
print("Data saved to output.csv")
sales_df =pd.read_csv('sample_sales_data.csv')
print("Sample data:")
print(sales_df.head())
sales_by_region =sales_df.groupby('Region')['Sales'].sum()
print("\nTotal sales by region:")
print(sales_by_region)
best_selling_products
```

```
=sales_df.groupby('Product')['Quantity'].sum().sort_values(ascending=False)
print("\nBest-selling  products by quantity:")
print(best_selling_products)
sales_by_region.to_csv('sales_by_region.csv')
best_selling_products.to_csv('best_selling_products.csv')
print("Data saved to sales_by_region.csv and best_selling_products.csv")

import yfinance as yf
import matplotlib.pyplot as plt
tickers = ["RELIANCE.NS", "TCS.NS", "INFY.NS"]
data = yf.download(tickers, start="2022-10-01", end="2023-10-01",
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)
print("\n")
reliance_data = data['RELIANCE.NS']
print("\nSummary statistics for Reliance Industries:")
print(reliance_data.describe())
reliance_data['Daily Return'] = reliance_data['Close'].pct_change()
print("\n")
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
reliance_data['Close'].plot(title="Reliance Industries - Closing Price")
plt.subplot(2, 1, 2)
reliance_data['Daily Return'].plot(title="Reliance Industries - Daily Returns",
color='orange')  plt.tight_layout()
plt.show()
reliance_data.to_csv('reliance_stock_data.csv')

tickers = ["HDFCBANK.NS", "ICICI.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)
print("\n")
reliance_data = data['HDFCBANK.NS']
print("\nSummary statistics for Reliance Industries:")
print(reliance_data.describe())
reliance_data['Daily Return'] = reliance_data['Close'].pct_change()
print("\n")
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
reliance_data['Close'].plot(title="HDFC Industries - Closing Price")
plt.subplot(2, 1, 2)
```

```
reliance_data['Daily Return'].plot(title="HDFCIndustries - Daily Returns",
color='red')  plt.tight_layout()
plt.show()
reliance_data.to_csv('hdfc_stock_data.csv')
print("\nhdfc stock data saved to 'hdfc_stock_data.csv'.")

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)
print("\n")
reliance_data = data['ICICIBANK.NS']
print("\nSummary statistics for ICICI Industries:")
print(reliance_data.describe())
reliance_data['Daily Return'] = reliance_data['Close'].pct_change()
print("\n")
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
reliance_data['Close'].plot(title="ICICI Industries - Closing Price")
plt.subplot(2, 1, 2)
reliance_data['Daily Return'].plot(title="ICICI Industries - Daily Returns",
color='BLACK')  plt.tight_layout()



plt.show()
reliance_data.to_csv('icici_stock_data.csv')
print("\nicici stock data saved to 'icici_stock_data.csv'.")


tickers = ["HDFCBANK.NS", "ICICI.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)
print("\n")
reliance_data = data['KOTAKBANK.NS']
print("\nSummary statistics for Reliance Industries:")
print(reliance_data.describe())
reliance_data['Daily Return'] =
reliance_data['Close'].pct_change()  print("\n")
plt.figure(figsize=(12, 6))
```

```
plt.subplot(2, 1, 1)
reliance_data['Close'].plot(title="KOTAK Industries -
Closing Price")  plt.subplot(2, 1, 2)
reliance_data['Daily Return'].plot(title="kotak Industries - Daily Returns",
color='red')  plt.tight_layout()
plt.show()
reliance_data.to_csv('kotak_stock_data.csv')
print("\nkotak stock data saved to 'kotak_stock_data.csv'.")
```

Program 2

Demonstrate various data pre-processing techniques for a given dataset

Screenshot:

apsara

4/3/25 LAB-2 Date:

- Write python code, consider filename as 'housing.csv'

1) To load .csv file into the dataframe

```
import pandas as pd
df = pd.read_csv(file_path)
```

2) To display information of all colums

Code:

```
print ("Dataset Information:")
print (df.info())
```

Output:

#	Column	Non-Null Count	Dtype
0	longitude	20640 non-null	float64

3) To display statistical information of all numerical

Code:

```
print ("Statistical Information:")
print (df.describe())
```

Output:

	longitude	latitude	housing_median_age	total_rooms
mean	-119.569	35.63	28.63	2635.76

total_bedrooms	population	households	median_income
537.87	1425.47	499.53	3.87

median_house_value

206855.816

4/3/25

Page No.:

iv) To display the count of unique labels for 'OceanProximity' column

→

Code:
```
print(" Unique labels count for 'ocean Proximity'")
print(df['ocean-proximity']. value_counts())
```

Output: Unique Labels count for 'Ocean Proximity' column

```
<1H Ocean 9136
Inland 6551
Near Ocean 2658
```

5) To display which attributes in a dataset having missing values count greater than 0

→ Code:
```
missing_values = df.isnull().sum()
Columns_with_missing = missing_vals[missing_vals >0]
print (columns_with_missing)
```

Output: Attributes with Missing Values:

```
total_bedrooms 207
dtype: int64
```

Date:

- For Diabetes and Adult income:

1) Which columns in dataset had missing values? How did you handle them
2) Which categorical column did you identify in the dataset? How did you encode them?
3) What is the difference between min max scaling and standardization? When would you use one over other?

1) Missing values are present in numerical columns if present which are replaced by mean of the respective column
2) No categorical columns, hence no encoding
3) Min Max Scaling transform data to a fixed range [0, 1] using

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

It is used when dataset doesn't follow a normal distribution, have different ranges and need to be bound

Standardization transforms data to have zero mean and unit variance

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

It is used when dataset follows a Gaussian distribution, many ML algorithms assume normality.

Page No.:

Diabities csv

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxscale, stdscale
import seaborn as sns
import matplotlib.pyplot as plt
Import sklearn.input
file_path = "diabetes csv"
df = pd.read_csv(file_path)
df.num = df.select_dtypes(include=['numeric']).copy()
inputs = df select simple imputer(strategy = "mean")
df[num.iloc[:, :] = imputer.fit_transform(df_numeric)
df[df_num. columns] = df. numeric
Q1 = df_num. quantile (0.25)
Q3 = df_num. Quantile (0.75)
IQR = Q3 - Q1
df = df [~((df_numeric < (Q1-1.5* IQR)) | (df_numeric > (Q3+1.5* IQR))). any (axis = 1)]
min_max_scale = MinMaxScaler( )
df_minmax = pd.DataFrame(min_max_scaler.fit_transform (df_num . columns = df_num.(columns)
std_scale = StandardScaler( )
df_std = pd.DataFrame (std_scaler.fit_transform(df_ ),
Columns = df_num columns)
print (df_minmax. head())
print (df_standard. head())
```

- Adult Income Dataset

1) Missing values are represented by "?" which we replace with Nan
   - for numerical columns = replace with mean
   - for categorical columns - replace with mode

2) workclass, education, marital-status, occupation, relationship, race, sex, native_country, income
   Encoding_method:
   Label Encoding to convert categorical to numerical formats

Code:

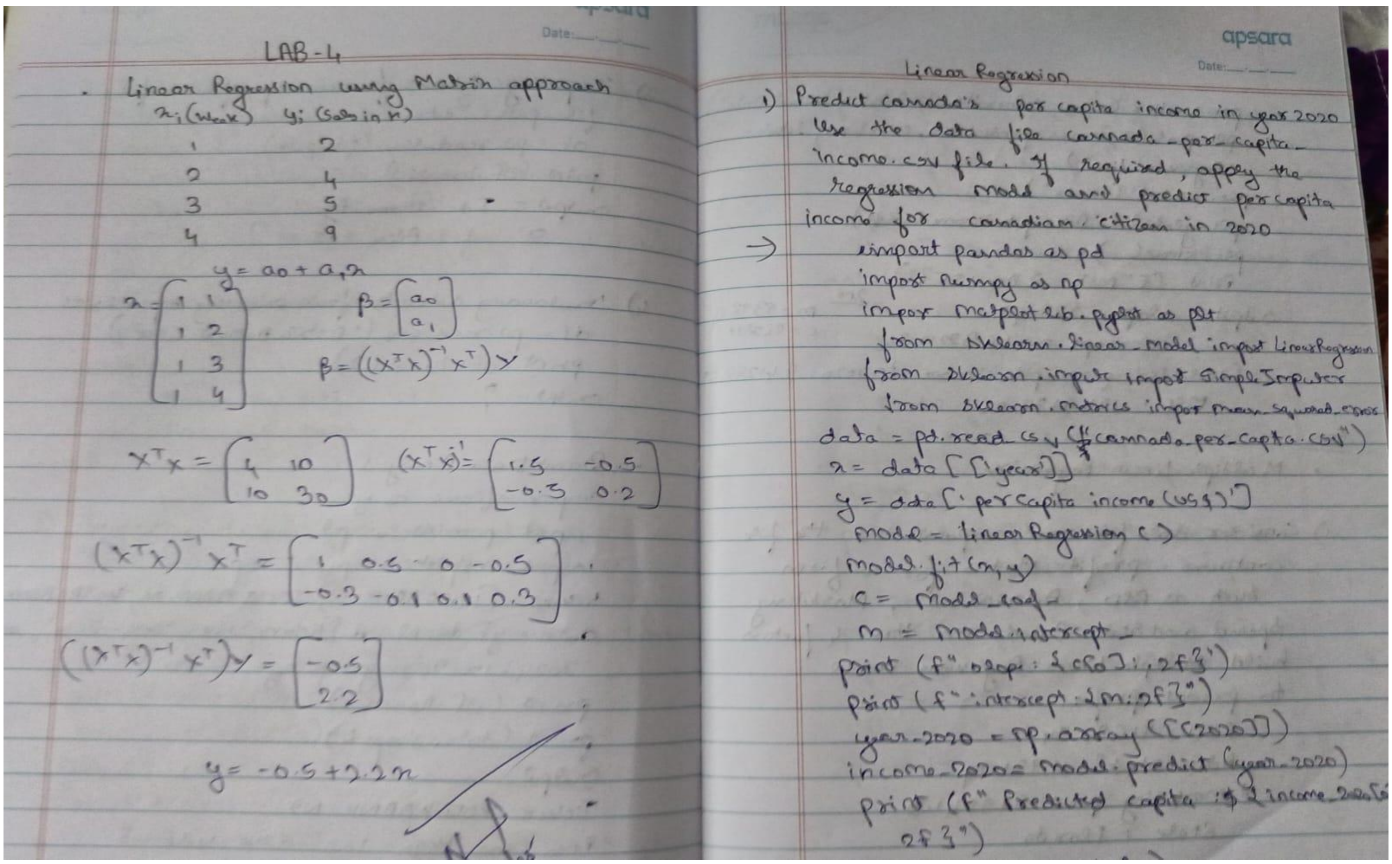
```
from google.colab import files
diabetes=files.upload()

from google.colab import files
adult_income=files.upload()

df1=pd.read_csv("Dataset of Diabetes .csv")
df1.head()

df2=pd.read_csv("adult.csv")
df2.head()

df1.info()
df2.info()
df1.describe()
df2.describe()

missing_values1 = df1.isnull().sum()
print(missing_values1)
missing_values2 = df2.isnull().sum()
print(missing_values2)

df1['Gender'] = df1['Gender'].replace('f', 'F')
ordinal_encoder = OrdinalEncoder(categories=[["F", M"]])
df1["Gender_Encoded"] =
ordinal_encoder.fit_transform(df1[["Gender"]]) onehot_encoder =
OneHotEncoder()
encoded_data =
onehot_encoder.fit_transform(df1[["CLASS"]]) encoded_array
= encoded_data.toarray()
encoded_df = pd.DataFrame(encoded_array,
columns=onehot_encoder.get_feature_names_out(["CLASS"])) df_encoded = pd.concat([df1,
encoded_df],  axis=1)
df1 = pd.concat([df1, encoded_df], axis=1)
df1.drop("CLASS", axis=1, inplace=True)
df1.drop("Gender", axis=1, inplace=True)
print(df2.head())
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder
df_copy2 = df2
ordinal_encoder = OrdinalEncoder(categories=[["Male","Female"]])
df_copy2["Gender_Encoded"] =
ordinal_encoder.fit_transform(df_copy2[["gender"]])
print(df_copy2[["gender","Gender_Encoded"]])

onehot_encoder = OneHotEncoder()
encoded_data =
onehot_encoder.fit_transform(df2[["occupation","workclass","education
","marital status","relationship","race","n ative-country","income"]])
encoded_array = encoded_data.toarray()
encoded_df =
pd.DataFrame(encoded_array,
```

```
columns=onehot_encoder.get_feature_names_out(["occupation","workclass","education
","marital status","relatio nship","race","native-country","income"]))
df_encoded = pd.concat([df_copy2, encoded_df], axis=1)

df_encoded.drop("gender", axis=1, inplace=True)
df_encoded.drop("occupation", axis=1, inplace=True)
df_encoded.drop("workclass", axis=1, inplace=True)
df_encoded.drop("education", axis=1, inplace=True)
df_encoded.drop("marital-status", axis=1, inplace=True)
df_encoded.drop("relationship", axis=1, inplace=True)
df_encoded.drop("race", axis=1, inplace=True)
df_encoded.drop("native-country", axis=1, nplace=True)
df_encoded.drop("income", axis=1, inplace=True)
print(df_encoded. head())

normalizer = MinMaxScaler()
df_encoded[["fnlwgt","educational-num","capital-gain","capital-loss","hours-per-week"]] =
normalizer.fit_transform(df_encoded[["fnlwgt","educational-num","capital-gain","capital-
loss","hours-per week"]
])
df_encoded.head()
normalizer = MinMaxScaler()
df1[["No_Pation","AGE","Urea","Cr" , "HbA1c" ,
"Chol","TG","HDL","LDL","VLDL","BMI"]] =
normalizer.fit_transform(df1[["No_Pation","AGE","Urea","Cr" , "HbA1c" ,
"Chol","TG","HDL","LDL","VLDL","BMI"]])
df1.head()
```

<u>**Program 3**</u>

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Screenshot:



LAB-4

Linear Regression using Matrix approach

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

$$y = a_0 + a_1 x$$

$$x = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{bmatrix} \qquad \beta = \begin{bmatrix} a_0 \\ a_1 \end{bmatrix}$$

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

$$X^T X = \begin{bmatrix} 4 & 10 \\ 10 & 30 \end{bmatrix} \qquad (X^T X)^{-1} = \begin{bmatrix} 1.5 & -0.5 \\ -0.3 & 0.2 \end{bmatrix}$$

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

$$((X^T X)^{-1} X^T) y = \begin{bmatrix} -0.5 \\ 2.2 \end{bmatrix}$$

$$y = -0.5 + 2.2x$$

Linear Regression

1) Predict canada's per capita income in year 2020. Use the data file cannada_per_capita_income.csv file. If required, apply the regression model and predict per capita income for canadian citizen in 2020

```
→ import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.impute import SimpleImputer
from sklearn.metrics import mean_squared_error
data = pd.read_csv(f"cannada_per_capita.csv")
x = data[['year']]
y = data['per capita income (US$)']
model = LinearRegression()
model.fit(x, y)
c = model.coef_
m = model.intercept_
print(f"slope: {c[0]:.2f}")
print(f"intercept: {m:.2f}")
year_2020 = np.array([[2020]])
income_2020 = model.predict(year_2020)
print(f"Predicted capita: $ {income_2020[0]:
  .2f}")
```

apsara

Date:

```
plt.scatter (x, y, color='blue', label='actual Data')
plt.plot (x, model.predict(x), color=red, label=
"Regression line")
plt.xlabel ('Year')
plt.ylabel ('Per capita income (US$)')
plt.show( )
Print (f"MSE: {mse:2f}")
```

Output: slope = 828.47

intercept = -1632210.76

2nd: m = 9398.4, c = 2626591

Predicted per capita income in 2020: $ 41288.6

MSE: 15467439.06

Code:

- Multiple Linear Regression

1) Consider data file 1000_companies.csv the file contains profit statics for a firm such as R&D, Administratn, Marketing Spend and State. Based on these 4 factors build Multiple Linear Regression Model to predict profit.

R&D = 91694.94

Administratn: 11931.24

Marketing Spend: 515841.3

State : Florida

apsara

Date:

→

```
from sklearn.preprocessing import LabelEncoder
data= pd.read_csv ("1000_companies.csv")
data.fillna (data.mean (numeric_only=True),
inplace=True)
label_Encoder = LabelEncoder()
data['state'] = label_encoder.fit_transform(
data['state'])
x = data.iloc [:, :-1].
y = data.iloc [:, -1]
model = LinearRegression( )
model.fit (x, y)
test_data = np.array ([[91694.48, 158413,
11931.24, label_encoder.transform(['Florida'])
[0]]])
predicted_profit = model.predict (test_data)
print (f"Predicted profit: $ {predict_profit[0]
:.2f}")
print (f"slope : {model.coef_[0]:.2f}")
print (f"Intercept (c): {model.intercept_:.2f}")
y_pred = model.predict (x)
mse = mean_squared_error (y, y_pred)
print (f"mse: {mse:.2f}")
```

Output: Predicted profit: $ -253.01

slope: 0.55

Intercept = -70214.44

MSE : 92128865.28

③ Candidate 1 salary: 4773289

Candidate 2 salary: 86424.67

Intercept 14910.23

Slope = 2633.05

11/3/25

Code:

```
from google.colab import files
per_capita_income=files.upload()

from google.colab import files
salary=files.upload()

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import
train_test_split  from sklearn.impute
import SimpleImputer
from sklearn.preprocessing import OrdinalEncoder,
OneHotEncoder  from sklearn.preprocessing import
StandardScaler, MinMaxScaler  from scipy import stats
from sklearn import linear_model

df1=pd.read_csv("canada_per_capita_inc
ome.csv")  df1.head()

df2=pd.read_csv("salary.csv")
df2.head()
df2.YearsExperience.median()
df2.YearsExperience =
df2.YearsExperience.fillna(df2.YearsExperience.median()) df2

plt.xlabel("year")
plt.ylabel("per capita income (US$)")
plt.scatter(df1.year, df1['per capita income (US$)'])

plt.xlabel("YearsExperience")
plt.ylabel("Salary")
plt.scatter(df2.YearsExperience, df2.Salary)

reg1 = linear_model.LinearRegression()
reg1.intercept_
reg1.predict([[2020]])

reg2 = linear_model.LinearRegression()
reg2.fit(df2.drop('Salary', axis='columns'),
df2['Salary'])  reg2.coef_
reg2.intercept_
reg2.predict([[12]])

from google.colab import files
hiring=files.upload()

from google.colab import files
companies=files.upload()

df3=pd.read_csv("hiring.csv")
```

```
df3.head()

df4=pd.read_csv("1000_Companies.csv")
df4.head()

df3.isnull().sum()
df4.isnull().sum()

df3_copy = df3.copy()
experience_mapping = {'two': 2, 'three': 3, 'five': 5, 'seven': 7, 'ten': 10,
'eleven': 11}  df3_copy['experience'] =
df3_copy['experience'].map(experience_mapping)
median_experience = df3_copy['experience'].median()
df3_copy['experience'] = df3_copy['experience'].fillna(median_experience)
df3_copy
df3_copy['test_score(out of 10)'] = df3_copy['test_score(out of
 10)'].fillna(df3_copy['test_score(out of  10)'].mean())
reg3 = linear_model.LinearRegression()
reg3.fit(df3_copy.drop('salary($)', axis='columns'),
df3_copy['salary($)']) reg3.coef_
reg3.intercept_
reg3.predict([[2,9,6]])
reg3.predict([[12,10,10]])
ohe = OneHotEncoder(sparse_output=False, handle_unknown='ignore')
state_encoded =  ohe.fit_transform(df4[['State']])
state_encoded_df = pd.DataFrame(state_encoded, columns=ohe.get_feature_names_out(['State']))

df4 = pd.concat([df4, state_encoded_df], axis=1).drop(columns=['State'])
print(df4)
reg4 = linear_model.LinearRegression()
reg4.fit(df4.drop('Profit',axis='columns'),df4.Profit)
print(reg4.coef_)
print(reg4.intercept_)
reg4.predict([[91694.48, 515841.3, 11931.24,0,1,0]])
```

Program 4

Build Logistic Regression Model for a given dataset

Screenshot:

apsara

Date:

LAB - 3

i) Given $a_0 = -5$ $a_1 = 0.8$

i) Logistic regression equation

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

ii) Calculate probablity that a student who studies for 7 hrs will pass

→ $x = 7$ $p(x) = \frac{1}{1+e^{-(-5+0.8(7))}}$

$= 0.6457$

iii) Determine the predicted class (P/F) for this Student based on threshold of 0.5

→ $P(x) = 0.6457$

$P(x) \geq 0.5$ $y = \begin{cases} 1 & \text{if } P(x) \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$

Thus $y = 1$ (Pass)

N 18/3/25

2) Consider $z = [2, 1, 0]$ for three classes Apply softmax func^n to find probabilities values of 3 classes

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

$$\text{softmax}(z_1) = \frac{e^2}{e^2+e^1+e^0} = 0.665$$

Page No.:

$$\text{Softmax}(z_2) = \frac{e^1}{e^2+e^1+e^0} = 0.244$$

$$\text{Softmax}(z_3) = \frac{e^0}{e^2+e^1+e^0} = 0.091$$

Probabilities of the 3 classes are approximately 66.5%, 24.4%, 9.1%.

- Logistic Regression - Binary

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, marker='+', color="red")
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(df[['age']], df.bought, train_size=0.9, random_state=10)
x_train.shape
x_test
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(x_train, y_train)
x_test
y_test
y_predicted = model.predict(x_test)
model.score(x_test, y_test)
model.predict(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_func(age):
    z = 0.127 * age - 4.973
    y = sigmoid(z)
    return y
age = 35
prediction_func(age)

Output:
0.35 is less than 0.5 which means person with age 35 will not buy insurance

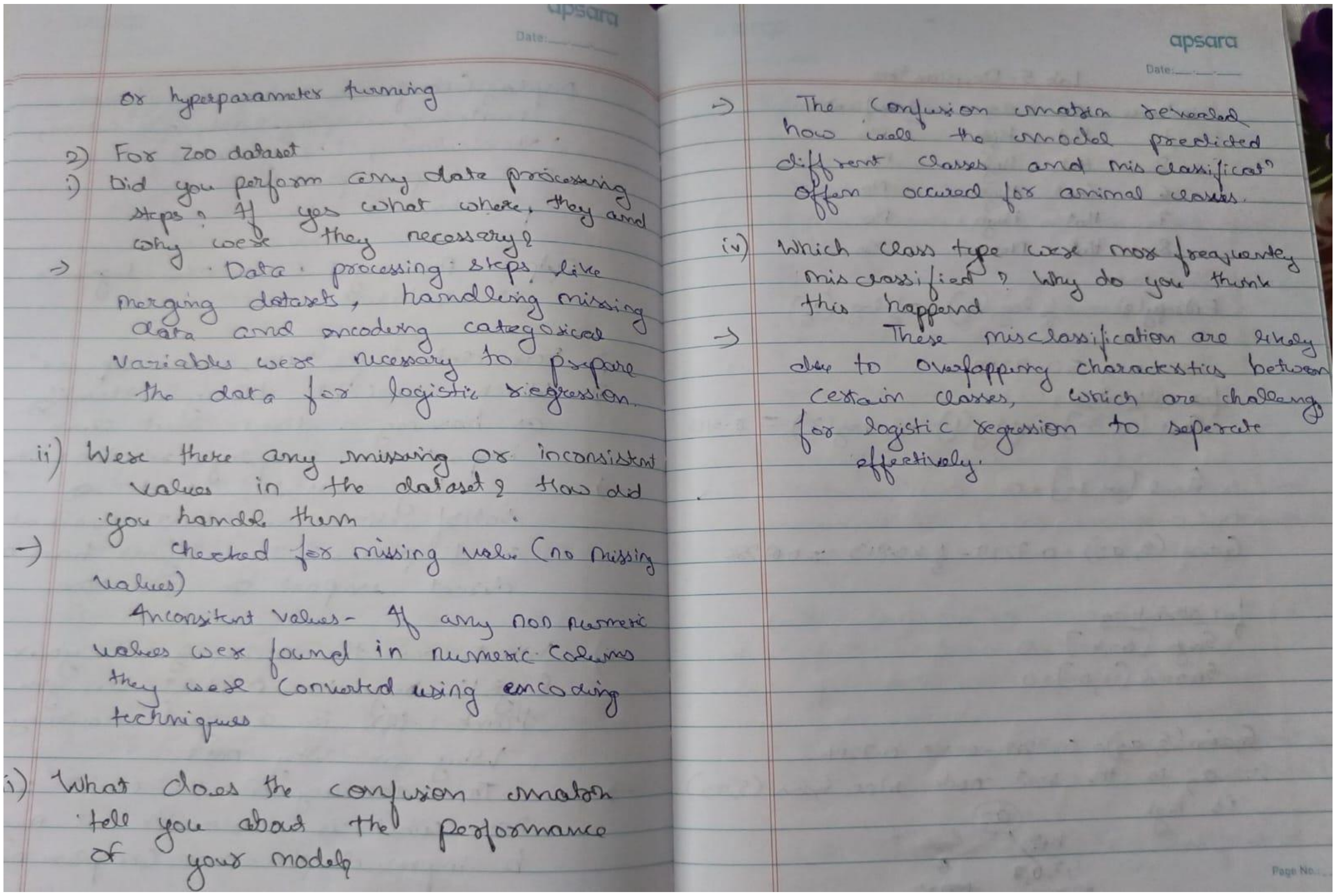
- LogisticRegression - Multiclass

import pandas as pd
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn import metrics
import matplotlib.pyplot as plt
iris = pd.read_csv("iris.csv")
iris.head()
x = iris.drop('species', axis='columns')
y = iris.species
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
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(f"accuracy of multinomial regression {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()

Output: Accuracy of Multinomial Regression is set to 1.00

i) For dataset file "HR_comma_sep.csv"
i) Which variable did you identify as having a direct and clear impact on employee retention? Why?
→ Key variables such as satisfaction_level, last_evaluation and promotion_last_5years had a direct impact on retention

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 the model was 79% which is reasonable but might be improved with more complex mod


or hyperparameter tuning

2) For zoo dataset
i) Did you perform any data processing steps? If yes what where they and why were they necessary?
→ Data processing steps like merging datasets, handling missing data and encoding categorical variables were necessary to prepare the data for logistic regression.

ii) Were there any missing or inconsistent values in the dataset? How did you handle them
→ Checked for missing value (no missing values)
Inconsitent values - If any non numeric values were found in numeric columns they were converted using encoding techniques

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

→ The confusion matrix revealed how well the model predicted different classes and misclassification often occured for animal classes.

iv) Which class type were most frequently misclassified? Why do you think this happend
→ These misclassification are likely due to overlapping characteristics between certain classes, which are challeng for logistic regression to seperate effectively.

Code:

```
from google.colab import files
hr=files.upload()

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OrdinalEncoder,
OneHotEncoder  from sklearn.preprocessing import
StandardScaler, MinMaxScaler
from scipy import stats
from sklearn import linear_model
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

df1=pd.read_csv("HR_comma_sep.csv")
df1.head()
df1.isnull().sum()
plt.figure(figsize=(12, 6))
```

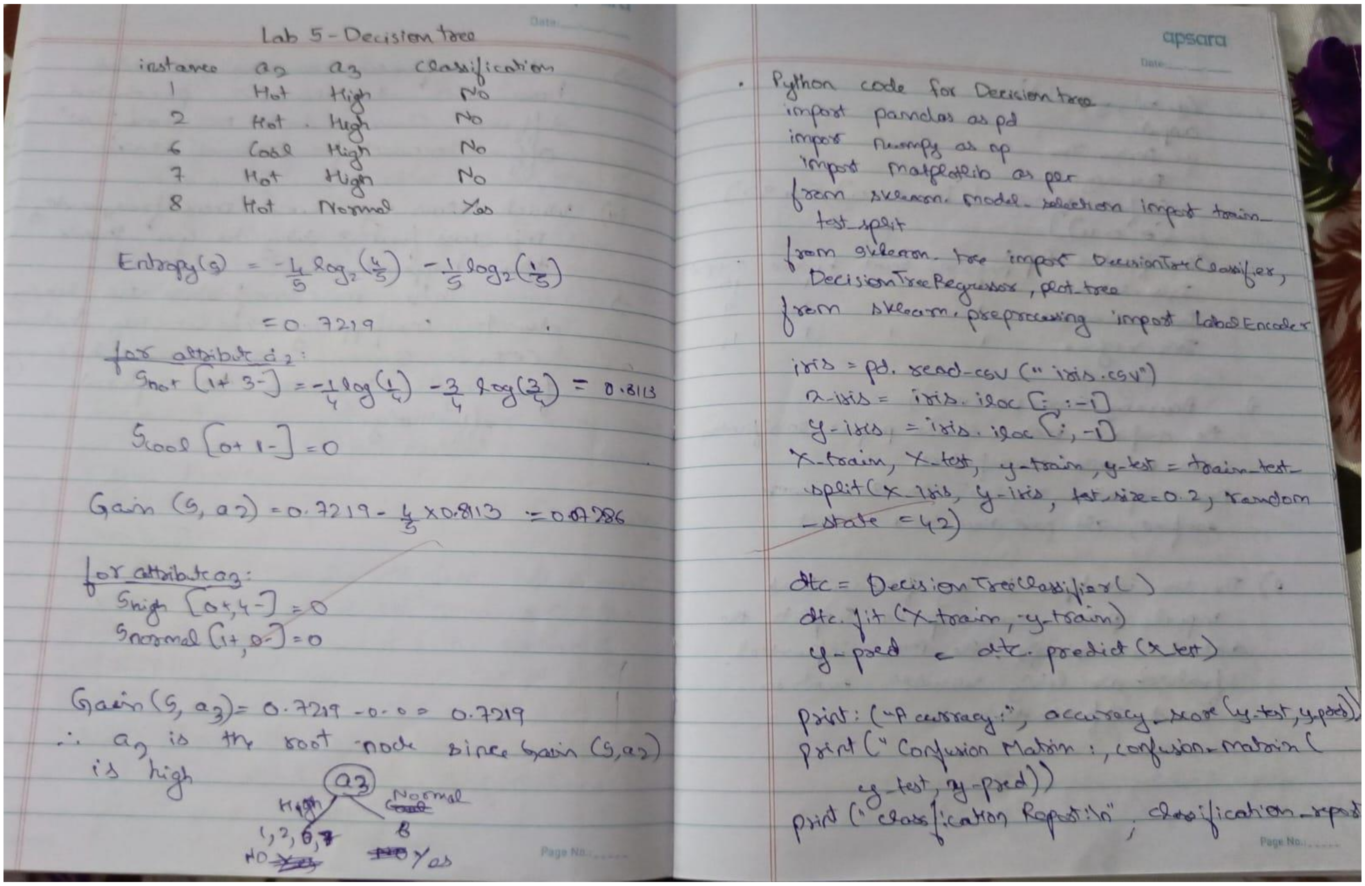
```
sns.barplot(x='Department', y='left', data=df1)
plt.title('Employee Retention Rate by Department')
plt.xlabel('Department')
plt.ylabel('Proportion of Employees Left')
plt.xticks(rotation=45, ha='right')
plt.show()

ohe = OneHotEncoder(handle_unknown='ignore',
sparse_output=False)  department_encoded =
ohe.fit_transform(df1[['Department']])
department_encoded_df = pd.DataFrame(department_encoded,
columns=ohe.get_feature_names_out(['Department']))
df1 = pd.concat([df1, department_encoded_df], axis=1)
df1 = df1.drop('Department', axis=1)
ordinal_encoder = OrdinalEncoder(categories=[['low', 'medium', 'high']],
dtype=np.int64)  salary_encoded =
ordinal_encoder.fit_transform(df1[['salary']])
df1['salary_encoded'] = salary_encoded
df1 = df1.drop('salary', axis=1)
df1.head()

correlation_matrix = df1.corr()
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm',
fmt=".2f")  plt.title('Correlation Matrix of Features')
plt.show()
plt.figure(figsize=(8, 6))
sns.barplot(x='salary_encoded', y='left', data=df1)
plt.title('Impact of Employee Salary on Retention')
plt.xlabel('Salary Level (Encoded)')
plt.ylabel('Proportion of Employees Left')
plt.show()



df_copy = df1[['number_project', 'average_montly_hours', 'time_spend_company',
'left','salary_encoded',  'satisfaction_level','Work_accident']]
df_copy.head()
X = df_copy.drop('left', axis=1)
y = df_copy['left']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)  model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of the Logistic Regression model: {accuracy}")

from google.colab import files
zoodata=files.upload()
zootype=files.upload()

zoo_data = pd.read_csv('zoo-data.csv')
zoo_class = pd.read_csv('zoo-class-type.csv')
```

```
merged_data = pd.merge(zoo_data, zoo_class, left_on='class_type',
right_on='Class_Number') merged_data = merged_data.drop(['Animal_Names',
'Number_Of_Animal_Species_In_Class',
'Class_Number','class_type','animal_name'], axis=1)
X = merged_data.drop('Class_Type', axis=1)
y = merged_data['Class_Type']
print(merged_data.head())
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42) model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=np.unique(y_test)) disp.plot(cmap="Blues", values_format="d")
plt.title("Confusion Matrix")
plt.show()
```

Program 5

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

Screenshot:



Lab 5 - Decision tree

instance	a_2	a_3	Classification
1	Hot	High	No
2	Hot	High	No
6	Cool	High	No
7	Hot	High	No
8	Hot	Normal	Yes

$Entropy(S) = -\frac{4}{5}\log_2\left(\frac{4}{5}\right) - \frac{1}{5}\log_2\left(\frac{1}{5}\right)$

$= 0.7219$

for attribute a_2:

$S_{hot}\,[1+, 3-] = -\frac{1}{4}\log\left(\frac{1}{4}\right) - \frac{3}{4}\log\left(\frac{3}{4}\right) = 0.8113$

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

$Gain(S, a_2) = 0.7219 - \frac{4}{5} \times 0.8113 = 0.0786$

for attribute a_3:

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

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

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

$\therefore a_3$ is the root node since $Gain(S, a_3)$ is high

a3

High ~~Cool~~ Normal

1, 2, 6, 7 — 8

No ~~Yes~~ — ~~No~~ Yes

• Python code for Decision tree

```
import pandas as pd
import numpy as np
import matplotlib as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, plot_tree
from sklearn.preprocessing import LabelEncoder

iris = pd.read_csv("iris.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("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report
```

(y_test, y_pred))

Output

Decision Tree

Accuracy: 1.0

Confusion Matrix: [[10, 0 0], [0 9 0]
[0 0 11]]

Classification Report:

	precision	recall	f1-score	support
Iris-Setosa	1	1	1	10
Iris-Versicolor	1	1	1	9
Iris-Virginica	1	1	1	11

1) Accuracy score = 1.0 (100%)
the confusion matrix shows perfect classification with no miss classifications
Miss classification analysis: No classes were confused, all predictions were correct

2) The tree identifies the key features that impact petrol consumption & splits the dataset accordingly, the most influential features are likely variables such as road infrastructure

Code:

```
from google.colab import files
iris=files.upload()
df1=pd.read_csv("iris.csv")
df1.head()

df1.isnull().sum()

X = df1.drop('species', axis=1)
y = df1['species']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)  clf = DecisionTreeClassifier(criterion='entropy')
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
print(classification_report(y_test, y_pred))
plt.figure(figsize=(12, 8))
plot_tree(clf, filled=True, feature_names=X.columns,
class_names=y.unique()) plt.show()

cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=clf.classes_)  cmap = plt.cm.get_cmap('PuBuGn')
disp.plot(cmap=cmap)
plt.show()

drug=files.upload()
df2=pd.read_csv("drug.csv")
df2.head()
df2.isnull().sum()

label_encoders = {}
for column in df2.columns:
    le = LabelEncoder()
    df2[column] = le.fit_transform(df2[column])
    label_encoders[column] = le
X = df2.drop('Drug', axis=1)
y = df2['Drug']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)  clf = DecisionTreeClassifier(criterion='entropy')
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
print(classification_report(y_test, y_pred))
plt.figure(figsize=(12, 8))
plot_tree(clf, filled=True, feature_names=X.columns, class_names=[str(c) for c in
y.unique()])  plt.show()

cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=clf.classes_)
cmap = plt.cm.Blues
disp.plot(cmap=cmap)
plt.show()

pc=files.upload()
df3=pd.read_csv("petrol_consumption.csv")
df3.head()
df3.isnull().sum()
X = df3.drop('Petrol_Consumption', axis=1)
y = df3['Petrol_Consumption']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)  regressor = DecisionTreeRegressor(random_state=42)
regressor.fit(X_train, y_train)
y_pred =
regressor.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
rmse = sqrt(mse)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f'Mean Squared Error: {mse:.2f}')
print(f'Root Mean Squared Error:
{rmse:.2f}') print(f'Mean Absolute Error:
{mae:.2f}') print(f'R-squared: {r2:.2f}')
plt.figure(figsize=(30, 30))
plot_tree(regressor, filled=True, feature_names=X.columns,
fontsize=10)  plt.show()
```

Program 6

Build KNN Classification model for a given dataset.

Screenshot :

LAB-6 KNN

Person	Age	Salary(K)	Target	Distance	Rank
A	18	50	N	52.8	
B	23	55	N	46.57	
C	24	70	N	31.95	
D	41	60	Y	40.44	2
E	43	70	Y	31.04	3
F	38	40	Y	60.07	1
X	35	100	?		

$$\text{Distance} = \sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$$

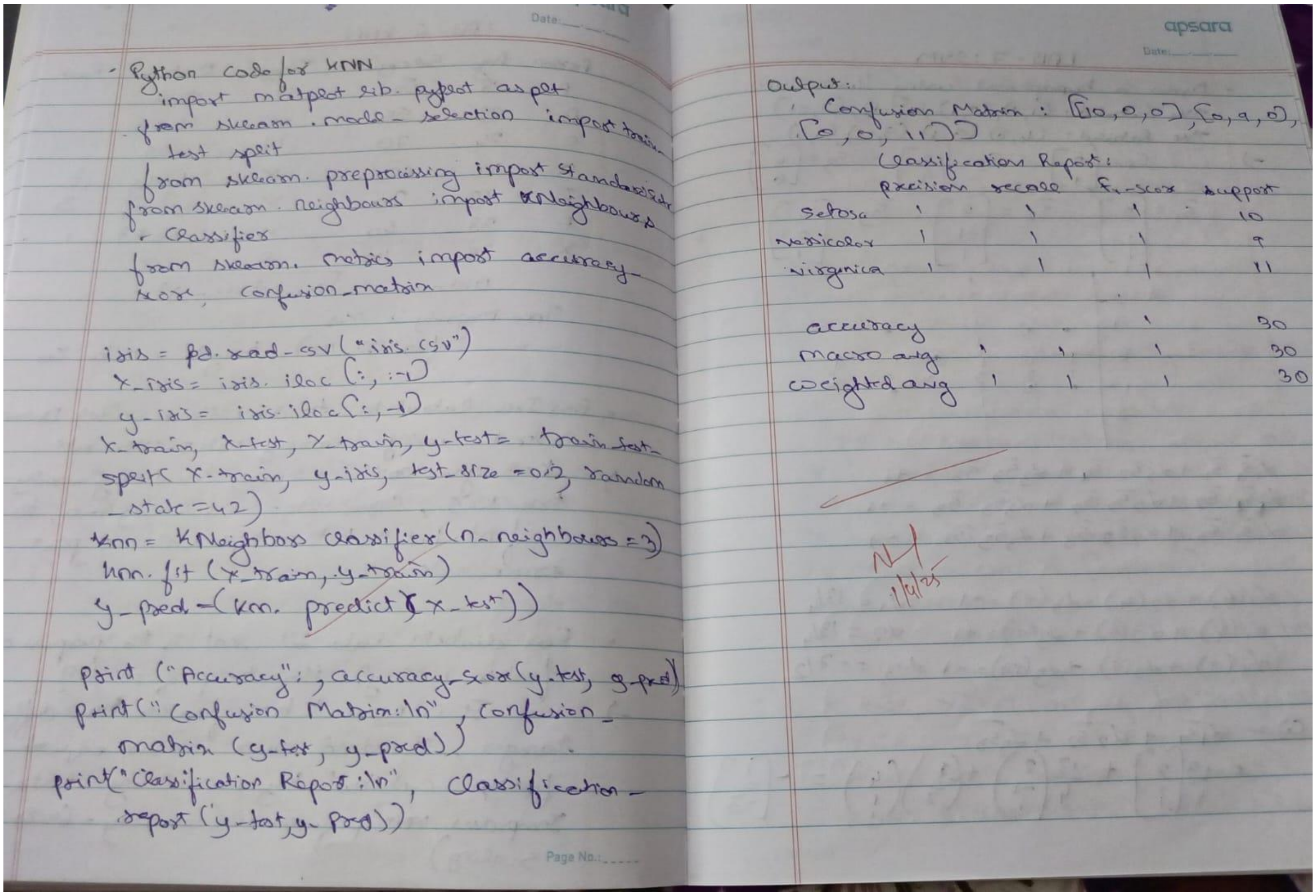
Since Majority is Yes
for (35, 100) target will be Yes

- For Iris dataset how to choose the k value? Demonstrate using accuracy rate & error rate

→ K=3 gives 100% accuracy here but generally take k=5, error rate = 1 - accuracy = 0, to find best k, test multiple values & plot error rate

- For diabetes data set what is the purpose of feature scaling? How to perform it?

→ It is needed bcz features have different ranges, Standard scaler ensures equal feature contribution by normalizing data. Improves KNN performance (accuracy = 69.84% after scaling)


Python code for KNN
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbours import KNeighboursClassifier
from sklearn.metrics import accuracy_score, confusion_matrix

iris = pd.read_csv("iris.csv")
x_iris = iris.iloc[:, :-1]
y_iris = iris.iloc[:, -1]
x_train, x_test, x_train, y_test = train_test_split(x_train, y_iris, test_size = 0.2, random_state = 42)
knn = KNeighbors Classifier(n_neighbours = 3)
knn.fit(x_train, y_train)
y_pred = (knn.predict(x_test))

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))

Output:
Confusion Matrix: [[10, 0, 0], [0, 9, 0], [0, 0, 11]]
Classification Report:

	precision	recall	f1-score	support
setosa	1	1	1	10
versicolor	1	1	1	9
virginica	1	1	1	11
accuracy			1	30
macro avg	1	1	1	30
weighted avg	1	1	1	30

1/4/25

Code:

```
from google.colab import files
iris=files.upload()
df1=pd.read_csv("iris (2).csv")
df1.head()
df1.isnull().sum()
X = df1.drop('species', axis=1)
y = df1['species']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)  best_k = 1
best_accuracy = 0
for k in range(1,
 11):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Accuracy for k={k}: {accuracy}, Error Rate for k={k}: {1-accuracy}")if accuracy >
    best_accuracy:  best_accuracy = accuracy best_k = k
print(f"Best k value: {best_k}")
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print("Accuracy Score:", accuracy_score(y_test, y_pred))
```

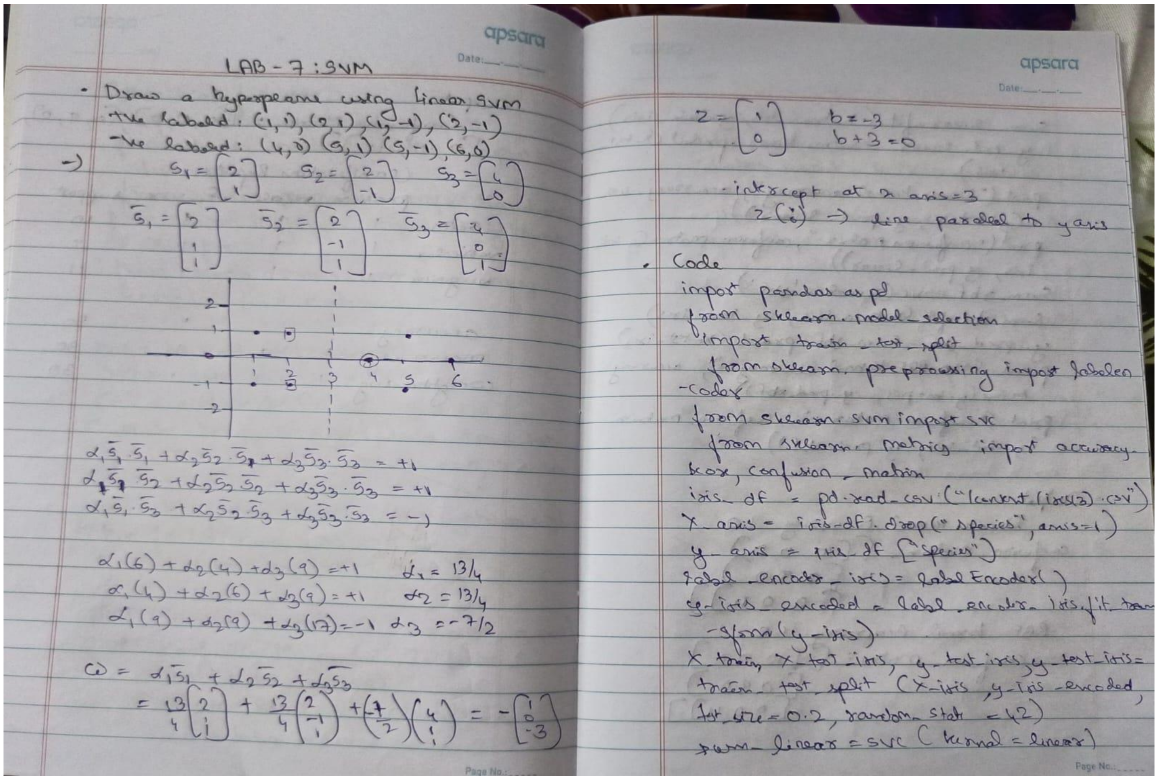
```
print("\nConfusion Matrix:")
cm = confusion_matrix(y_test, y_pred)
print(cm)
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
        xticklabels=knn.classes_, yticklabels=knn.classes_)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

diabetes=files.upload()
df2=pd.read_csv("diabetes.csv")
df2.head()
df2.isnull().sum()
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df2.drop('Outcome', axis=1))
X_train, X_test, y_train, y_test = train_test_split(X_scaled, df2['Outcome'], test_size=0.2,
random_state=42)  best_k = 1
best_accuracy = 0
for k in range(1,
11):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Accuracy for k={k}: {accuracy}")
    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_k = k
print(f"Best k value: {best_k}")
knn = KNeighborsClassifier(n_neighbors=best_k)
knn.fit(X_train, y_train) y_pred = knn.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel("Predicted") plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

heart=files.upload()
df3=pd.read_csv("heart.csv")
df3.head()
```

```
df3.isnull().sum()
X = df3.drop('target', axis=1)
y = df3['target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)  best_k = 1
best_accuracy = 0
for k in range(1,
 11):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Accuracy for k={k}: {accuracy}, Error Rate for k={k}: {1-
    accuracy}")  if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_k = k
print(f"Best k value: {best_k}")
knn = KNeighborsClassifier(n_neighbors=optimal_k)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print("Accuracy Score:", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix:")
cm = confusion_matrix(y_test, y_pred)
print(cm)
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=knn.classes_, yticklabels=knn.classes_)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

Program7

Build Support vector machine model for a given dataset

Screenshot:



LAB - 7 : SVM

- Draw a hyperplane using linear SVM
  +ve labeled: (1, 1), (2, 1), (1, -1), (2, -1)
  -ve labeled: (4, 0) (5, 1) (5, -1), (6, 0)

→ $S_1 = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$ $S_2 = \begin{bmatrix} 2 \\ -1 \end{bmatrix}$ $S_3 = \begin{bmatrix} 4 \\ 0 \end{bmatrix}$

$\bar{S}_1 = \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix}$ $\bar{S}_2 = \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix}$ $\bar{S}_3 = \begin{bmatrix} 4 \\ 0 \\ 1 \end{bmatrix}$

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

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

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

$\alpha_1(6) + \alpha_2(4) + \alpha_3(9) = +1$ $\alpha_1 = 13/4$

$\alpha_1(4) + \alpha_2(6) + \alpha_3(9) = +1$ $\alpha_2 = 13/4$

$\alpha_1(9) + \alpha_2(9) + \alpha_3(17) = -1$ $\alpha_3 = -7/2$

$\omega = \alpha_1 \bar{S}_1 + \alpha_2 \bar{S}_2 + \alpha_3 \bar{S}_3$

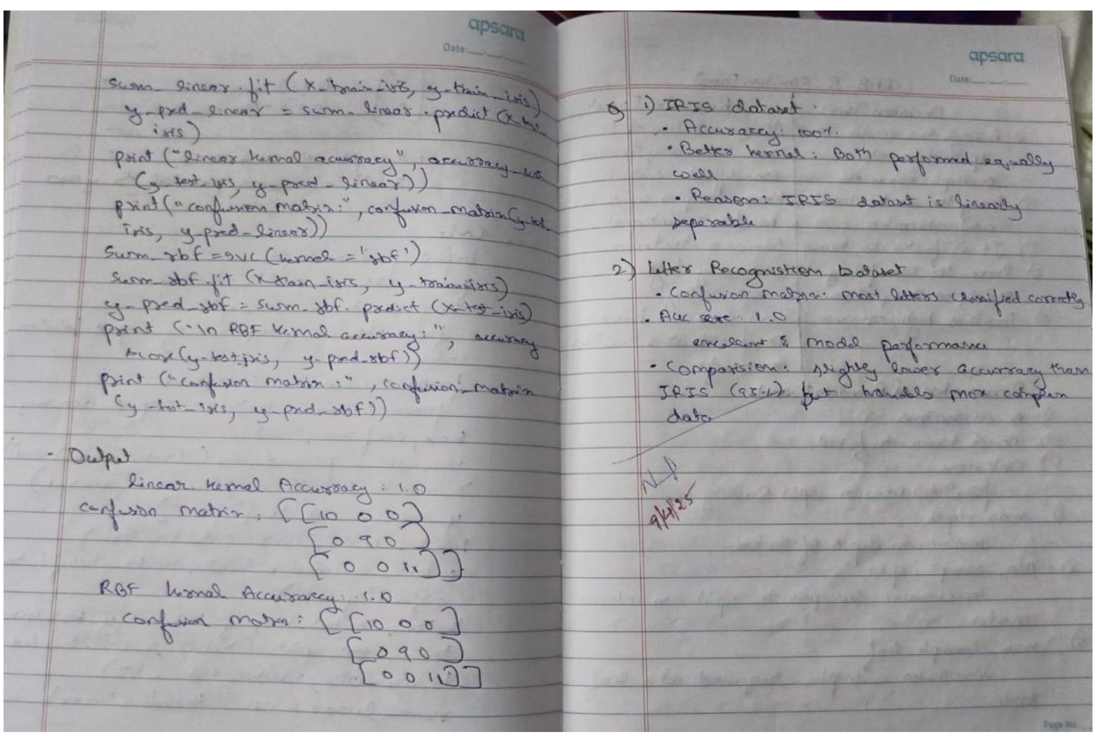
$= \frac{13}{4}\begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix} + \frac{13}{4}\begin{pmatrix} 2 \\ -1 \\ 1 \end{pmatrix} + \left(\frac{-7}{2}\right)\begin{pmatrix} 4 \\ 1 \\ 1 \end{pmatrix} = -\begin{bmatrix} 1 \\ 0 \\ -3 \end{bmatrix}$

$z = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ $b = -3$, $b + 3 = 0$

- intercept at x axis = 3
  $z\binom{1}{0}$ → line parallel to y axis

- Code

```
import pandas as pd
from sklearn.model_selection
  import train_test_split
  from sklearn.preprocessing import labelen
-coder
 from sklearn.svm import svc
  from sklearn.metrics import accuracy
score, confusion_matrix
iris_df = pd.read_csv("/content/(iris(3).csv")
X_axis = iris_df.drop("species", axis=1)
y_axis = iris_df ["species"]
label_encoder_iris = label Encoder( )
y-iris_encoded = label_encoder_iris.fit_tran
  -sform(y-iris)
X_train, X_test_iris, y_test_iris, y_test_iris=
 train_test_split (X-iris, y-iris-encoded,
 test_size=0.2, random_state =42)
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 kernal accuracy", accuracy_sco
  (y_test_iris, y_pred_linear))
print ("confusion matrix:", confusion_matrix(y_test_
 iris, y_pred_linear))
Svm_rbf = svc (kernel = 'rbf')
 svm_rbf.fit (x_train_iris, y_train_iris)
y_pred_rbf = svm_rbf.predict (x_test_iris)
print ("\n RBF kernal accuracy:", accuracy
  score(y_test_iris, y_pred_rbf))
print ("confusion matrix :", confusion_matrix
 (y_test_iris, y_pred_rbf))
```

- Output

```
 linear kernal Accuracy : 1.0
confusion matrix : [[10 0 0]
                    [0 9 0]
                    [0 0 11]]
RBF kernal Accuracy : 1.0
 confusion matrix : [[10 0 0]
                     [0 9 0]
                     [0 0 11]]
```

Q) 1) IRIS dataset.
- Accuracy: 100%.
- Better kernal: Both performed equally well
- Reason: IRIS dataset is linearly seperable

2) Letter Recognistion Dataset
- Confusion matrix: most letters classified correctly
- Auc score 1.0
  excellent & model performance
- Comparision: slightly lower accurracy than IRIS (95.1%) but handles more complex data

9/4/25

Code:

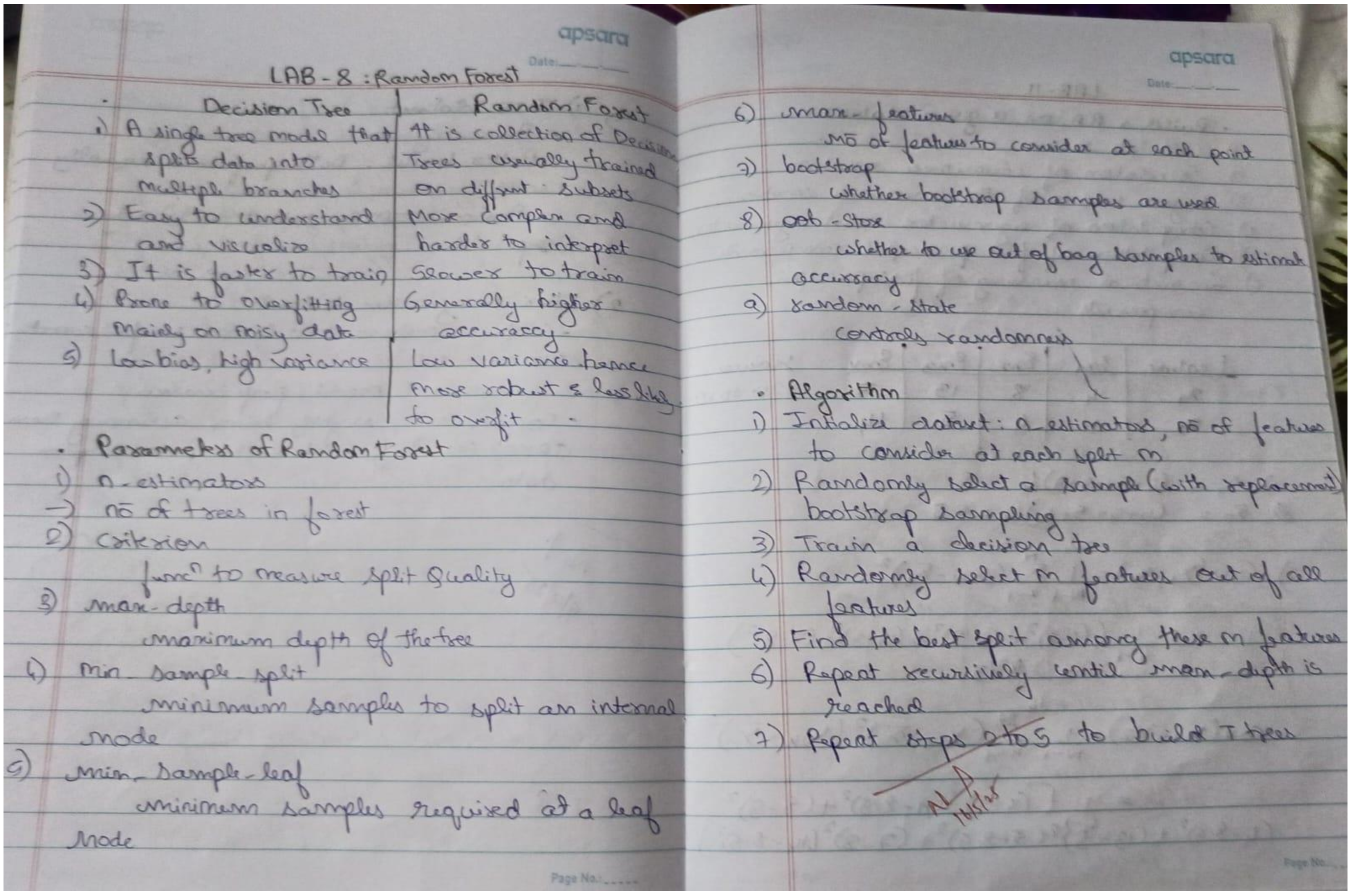
```
from google.colab import files
iris=files.upload()
df1=pd.read_csv("iris (1).csv")
df1.head()
X = df1.drop('species', axis=1)
y = df1['species']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)  rbf_svm = SVC(kernel='rbf')
rbf_svm.fit(X_train, y_train)
rbf_y_pred = rbf_svm.predict(X_test)
print("RBF Kernel SVM:")
print("Accuracy:", accuracy_score(y_test, rbf_y_pred))
cm = confusion_matrix(y_test, rbf_y_pred)
sns.heatmap(cm, annot=True, fmt='d',cmap="Blues")
plt.title('Confusion Matrix for RBF Kernel SVM')
plt.xlabel('Predicted')
plt.ylabel('True') plt.show()
print(classification_report(y_test, rbf_y_pred))
linear_svm = SVC(kernel='linear')
linear_svm.fit(X_train, y_train)
linear_y_pred = linear_svm.predict(X_test)
print("\nLinear Kernel SVM:")
print("Accuracy:", accuracy_score(y_test, linear_y_pred))
cm = confusion_matrix(y_test, linear_y_pred)
sns.heatmap(cm, annot=True, fmt='d',cmap="Blues")
plt.title('Confusion Matrix for Linear Kernel SVM')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
print(classification_report(y_test, linear_y_pred))
letter=files.upload()
df2=pd.read_csv("letter-recognition.csv")
df2.head()
X = df2.drop('letter', axis=1)
y = df2['letter']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)  svm_classifier = SVC(kernel='linear', probability=True)
svm_classifier.fit(X_train, y_train)
y_pred =
svm_classifier.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(10,10))
sns.heatmap(cm, annot=True, fmt='d', cmap="Blues")
plt.title('Confusion Matrix for SVM')
plt.xlabel('Predicted')
plt.ylabel('True')
```

```
plt.show()
lb = LabelBinarizer()
lb.fit(y_test)




y_test_lb = lb.transform(y_test)
y_pred_prob =
svm_classifier.predict_proba(X_test) fpr = {}
tpr = {}
thresh ={}
roc_auc = dict()
n_class = y_test_lb.shape[1]
for i in range(n_class):
    fpr[i], tpr[i], thresh[i] = roc_curve(y_test_lb[:,i], y_pred_prob[:,i])
    roc_auc[i] = auc(fpr[i], tpr[i])
plt.plot(fpr[0], tpr[0], linestyle='--',color='orange', label='SVM (AUC = %0.2f)' %
roc_auc[0])  plt.title('ROC Curve for Class 0')
plt.xlabel('False Positive
Rate') plt.ylabel('True Positive
rate') plt.legend(loc='best')
plt.show()
print(f"AUC score for class 0: {roc_auc[0]}")
```

Program 8

Implement Random forest ensemble method on a given dataset

Screenshot:


LAB - 8 : Random Forest

Decision Tree	Random Forest
1) A single tree model that splits data into multiple branches	It is collection of Decision Trees usually trained on different subsets
2) Easy to understand and visualize	More Complex and harder to interpret
3) It is faster to train	Slower to train
4) Prone to overfitting mainly on noisy data	Generally higher accuracy
5) Low bias, high variance	Low variance hence more robust & less likely to overfit

- Parameters of Random Forest

1) n-estimators
→ no of trees in forest
2) criterion
func^n to measure split Quality
3) max-depth
maximum depth of the tree
4) min-sample-split
minimum samples to split an internal node
5) min-sample-leaf
minimum samples required at a leaf node
6) max-features
no of features to consider at each point
7) bootstrap
whether bootstrap samples are used
8) oob-score
whether to use out of bag samples to estimate accuracy
9) random-state
controls randomness

- Algorithm

1) Initialize dataset: n estimators, no of features to consider at each split m
2) Randomly select a sample (with replacement) bootstrap sampling
3) Train a decision tree
4) Randomly select m features out of all features
5) Find the best split among these m features
6) Repeat recursively until max-depth is reached
7) Repeat steps 2 to 5 to build T trees

Code:

```
from google.colab import files
iris=files.upload()
df1=pd.read_csv("iris (4).csv")
df1.head()
X = df1.drop('species', axis=1)
y = df1['species']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=0)  rf_classifier = RandomForestClassifier(random_state=0)
rf_classifier.fit(X_train, y_train)
y_pred =
rf_classifier.predict(X_test)
default_accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy with default n_estimators: {default_accuracy}")
best_accuracy = 0
best_n_estimators = 0
for n_estimators in range(1, 101):
    rf_classifier = RandomForestClassifier(n_estimators=n_estimators, random_state=0)
    rf_classifier.fit(X_train, y_train)
```

```
    y_pred = rf_classifier.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    if accuracy > best_accuracy:best_accuracy
    = accuracy best_n_estimators =
    n_estimators
print(f"\nBest accuracy: {best_accuracy} achieved with n_estimators =
{best_n_estimators}")  cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=np.unique(y_test), yticklabels=np.unique(y_test))
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```

Program 9

Implement Boosting ensemble method on a given dataset

Screenshot:

LAB - 9 : AdaBoost

Boosting - Combines multiple weak learners to create a strong learner. It works by training models sequentially where each model focuses on errors model by previous one.

Parameters

- Estimator - The base model
- n_estimator - no of weak learners
- learning rate - shrinks contribution of each learner
- Algorithm - 'SAMME.R'
- Random_state - for reproducibility

- Algorithm

1) Start with equal wts for all training sample
2) Train a weak model
~~3) learning rate shrinks contri~~
3) Calculate error and update sample weights
4) Add weak model to ensemble with a wt based on its accuracy
5) repeat n_estimator
6) Final prediction.

10/5/25

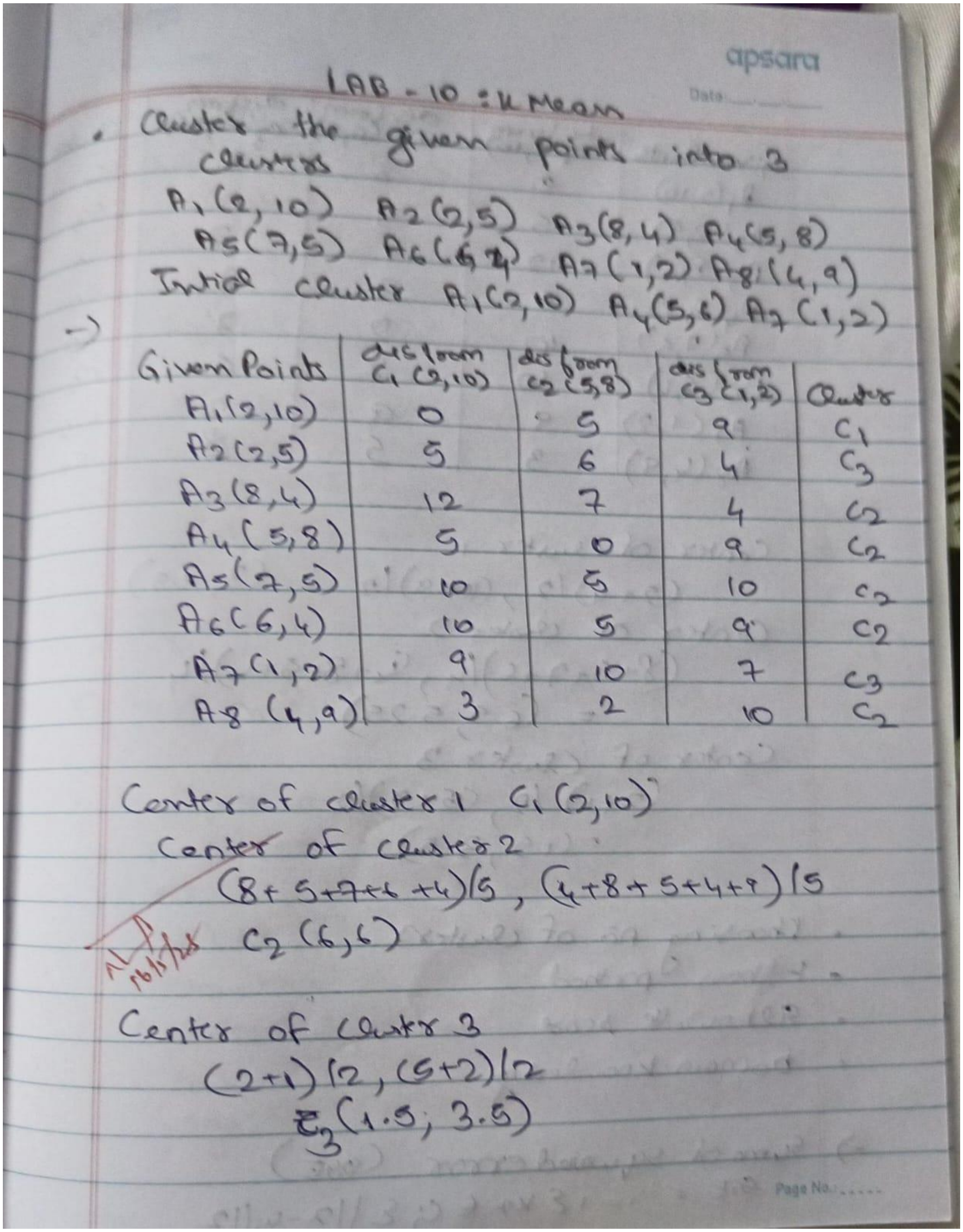
Code:

```
from google.colab import files
income=files.upload()
df1=pd.read_csv("income.csv")
df1.head()
X = df1.drop('income_level', axis=1)
y = df1['income_level'] X = pd.get_dummies(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)  abc = AdaBoostClassifier(n_estimators=10,
random_state=42)
abc.fit(X_train, y_train)
y_pred = abc.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Initial AdaBoost accuracy (10 trees): {accuracy}")
param_grid = {'n_estimators': [50, 100, 150, 200]}
grid_search = GridSearchCV(AdaBoostClassifier(random_state=42), param_grid, cv=5,
scoring='accuracy')  grid_search.fit(X_train, y_train)
print(f"Best parameters: {grid_search.best_params_}")
print(f"Best cross-validation score: {grid_search.best_score_}")
best_abc = grid_search.best_estimator_
y_pred_best = best_abc.predict(X_test)
best_accuracy = accuracy_score(y_test,
y_pred_best)
print(f"Accuracy of the best model on the test set: {best_accuracy}")
cm = confusion_matrix(y_test, y_pred_best)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['<=50K', '>50K'], yticklabels=['<=50K', '>50K'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

Program 10

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

Screenshot:



LAB - 10 : k Mean

- Cluster the given points into 3 clusters

$A_1(2,10)$ $A_2(2,5)$ $A_3(8,4)$ $A_4(5,8)$
$A_5(7,5)$ $A_6(6,4)$ $A_7(1,2)$ $A_8(4,9)$

Initial cluster $A_1(2,10)$ $A_4(5,8)$ $A_7(1,2)$

→

Given Points	dis from $C_1(2,10)$	dis from $C_2(5,8)$	dis from $C_3(1,2)$	Cluster
$A_1(2,10)$	0	5	9	C_1
$A_2(2,5)$	5	6	4	C_3
$A_3(8,4)$	12	7	4	C_2
$A_4(5,8)$	5	0	9	C_2
$A_5(7,5)$	10	5	10	C_2
$A_6(6,4)$	10	5	9	C_2
$A_7(1,2)$	9	10	7	C_3
$A_8(4,9)$	3	2	10	C_2

Center of cluster 1 $C_1(2,10)$

Center of cluster 2

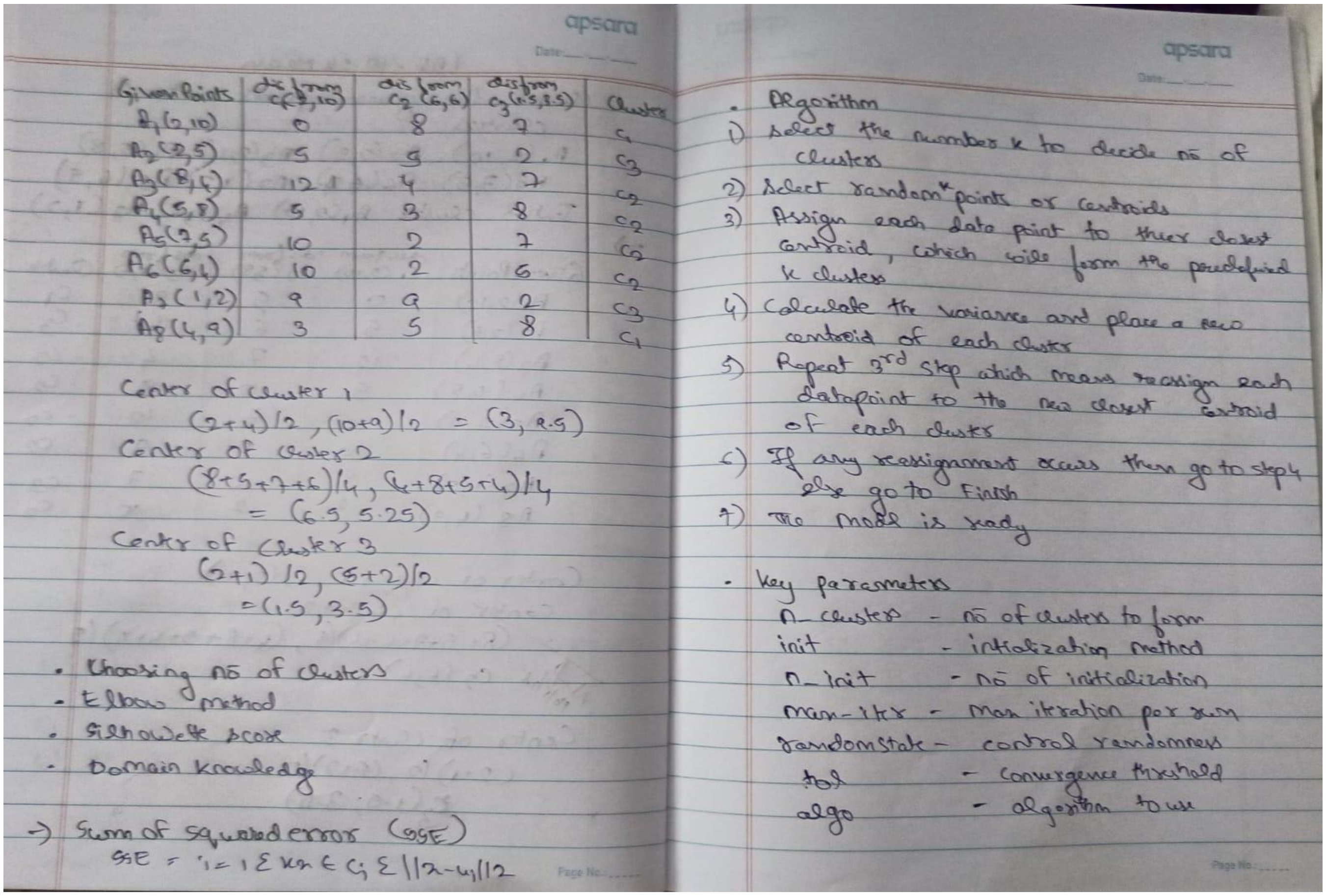
$(8+5+7+6+4)/5, (4+8+5+4+9)/5$

$C_2(6,6)$

Center of cluster 3

$(2+1)/2, (5+2)/2$

$C_3(1.5, 3.5)$



Given Points	dis from $c_1(2,10)$	dis from $c_2(6,6)$	dis from $c_3(1.5,3.5)$	Cluster
$A_1(2,10)$	0	8	7	c_1
$A_2(2,5)$	5	5	2	c_3
$A_3(8,4)$	12	4	7	c_2
$A_4(5,8)$	5	3	8	c_2
$A_5(7,5)$	10	2	7	c_2
$A_6(6,4)$	10	2	6	c_2
$A_7(1,2)$	9	9	2	c_3
$A_8(4,9)$	3	5	8	c_1

Center of cluster 1
$(2+4)/2, (10+9)/2 = (3, 9.5)$
Center of cluster 2
$(8+5+7+6)/4, (4+8+5+4)/4$
$= (6.5, 5.25)$
Center of cluster 3
$(2+1)/2, (5+2)/2$
$= (1.5, 3.5)$

- Choosing no of clusters
- Elbow method
- Silhouette score
- Domain knowledge

→ Sum of squared error (SSE)
$SSE = \sum_{i=1} \sum_{x_n \in C_i} ||x - \mu_i||^2$

- Algorithm
1) select the number k to decide no of clusters
2) Select random k points or centroids
3) Assign each data point to their closest centroid, which will form the predefined k clusters
4) Calculate the variance and place a new centroid of each cluster
5) Repeat 3rd step which means reassign each datapoint to the new closest centroid of each cluster
6) If any reassignment occurs then go to step 4 else go to Finish
7) The model is ready

- Key parameters
n_clusters - no of clusters to form
init - initialization method
n_init - no of initialization
max-iter - max iteration per run
randomstate - control randomness
tol - convergence threshold
algo - algorithm to use

Code:

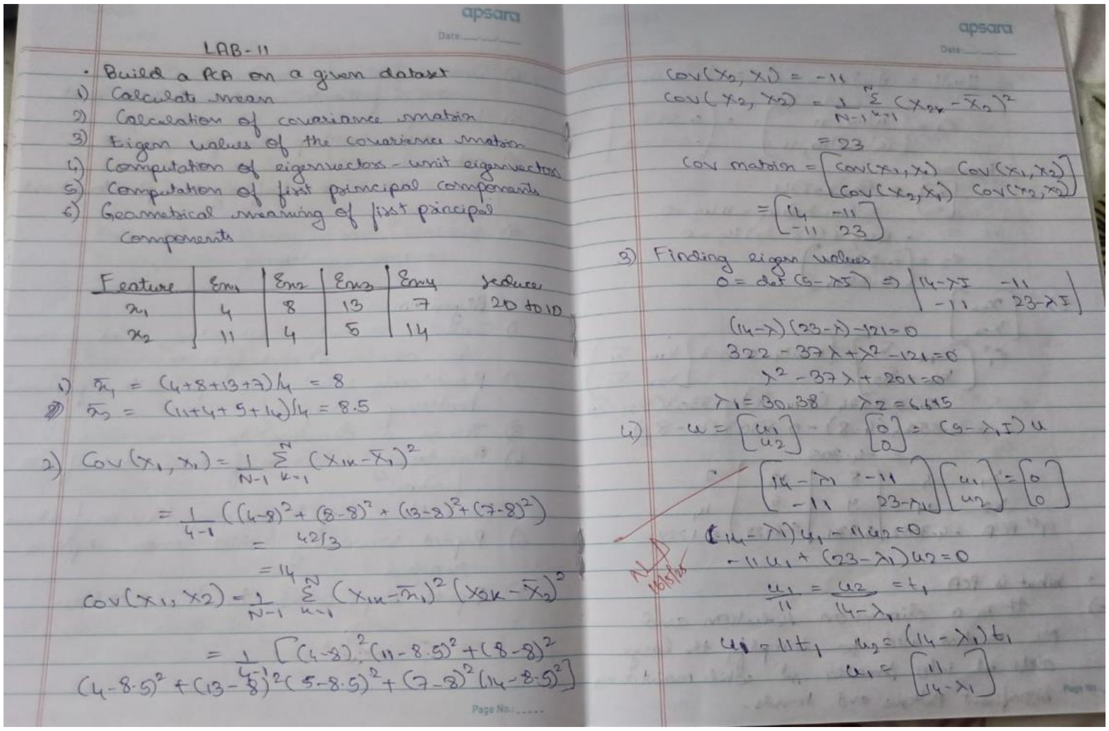
```
from google.colab import files
iris=files.upload()
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from scipy import stats
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score  from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

df1=pd.read_csv("iris (4).csv")
df1.head()
df = df1.drop(['sepal_length','sepal_width','species'],axis=1)
scaler = StandardScaler()
scaled_df = scaler.fit_transform(df) wcss = []
for i in range(1, 11):
   kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10,
   random_state=0)  kmeans.fit(scaled_df)
```

```
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
kmeans = KMeans(n_clusters=3, init='k-means++', max_iter=300, n_init=10,
random_state=0)  pred_y = kmeans.fit_predict(scaled_df)
df['cluster'] = pred_y
plt.scatter(df['petal_length'], df['petal_width'], c=df['cluster'])
plt.title('Clusters of Iris Flowers')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.show()
```

Program 11

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

Screenshot:



LAB-11

- Build a PCA on a given dataset
1) Calculate mean
2) Calculation of covariance matrix
3) Eigen values of the covariance matrix
4) Computation of eigenvectors - unit eigenvectors
5) Computation of first principal components
6) Geometrical meaning of first principal components

Feature	Ex_1	Ex_2	Ex_3	Ex_4	reduce
x_1	4	8	13	7	20 to 10
x_2	11	4	5	14	

1) $\bar{x}_1 = (4+8+13+7)/4 = 8$

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

2) $Cov(x_1, x_1) = \frac{1}{N-1} \sum_{k=1}^{N} (x_{1k} - \bar{x}_1)^2$

$= \frac{1}{4-1} ((4-8)^2 + (8-8)^2 + (13-8)^2 + (7-8)^2)$

$= 42/3$

$= 14$

$cov(x_1, x_2) = \frac{1}{N-1} \sum_{k=1}^{N} (x_{1k} - \bar{x}_1)^2 (x_{2k} - \bar{x}_2)^2$

$= \frac{1}{4-1} [(4-8)^2 (11-8.5)^2 + (8-8)^2 (4-8.5)^2 + (13-8)^2 (5-8.5)^2 + (7-8)^2 (14-8.5)^2]$

$cov(x_2, x_1) = -11$

$cov(x_2, x_2) = \frac{1}{N-1} \sum_{k=1}^{N} (x_{2k} - \bar{x}_2)^2$

$= 23$

$\text{cov matrix} = \begin{bmatrix} cov(x_1,x_1) & cov(x_1,x_2) \\ cov(x_2,x_1) & cov(x_2,x_2) \end{bmatrix}$

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

3) Finding eigen values

$0 = \det(S - \lambda I) \Rightarrow \begin{vmatrix} 14-\lambda I & -11 \\ -11 & 23-\lambda I \end{vmatrix}$

$(14-\lambda)(23-\lambda) - 121 = 0$

$322 - 37\lambda + \lambda^2 - 121 = 0$

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

$\lambda_1 = 30.38 \quad \lambda_2 = 6.615$

4) $u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \quad \begin{bmatrix} 0 \\ 0 \end{bmatrix} = (S - \lambda_1 I) u$

$\begin{bmatrix} 14-\lambda_1 & -11 \\ -11 & 23-\lambda_1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

$(14-\lambda_1)u_1 - 11u_2 = 0$

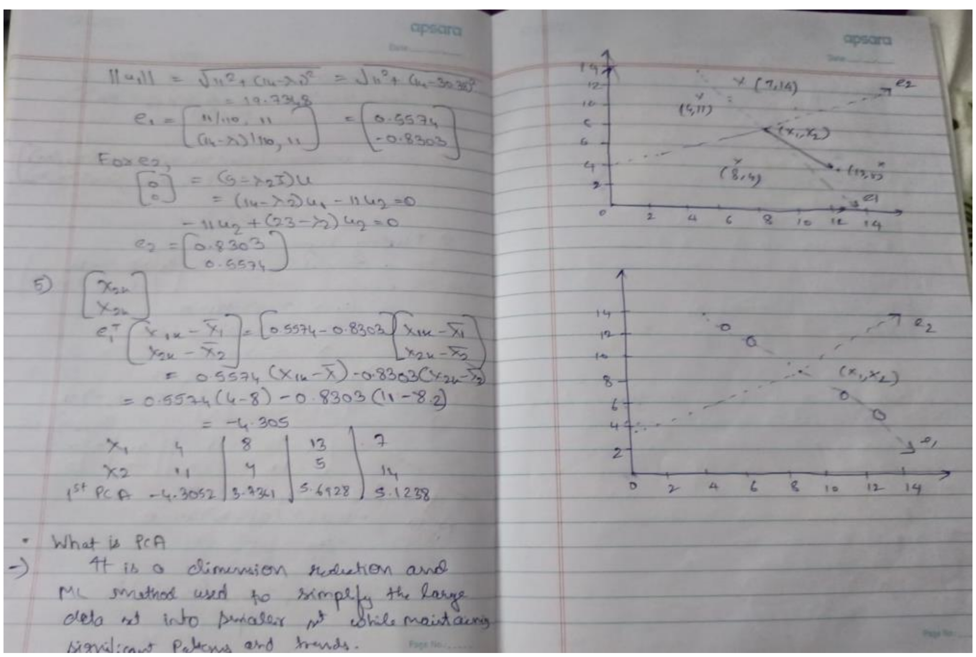
$-11u_1 + (23-\lambda_1)u_2 = 0$

$\frac{u_1}{11} = \frac{u_2}{14-\lambda_1} = t_1$

$u_1 = 11t_1 \quad u_2 = (14-\lambda_1)t_1$

$u_1 = \begin{bmatrix} 11 \\ 14-\lambda_1 \end{bmatrix}$

N 16/5/25



$\|u_1\| = \sqrt{11^2 + (14-\lambda_1)^2} = \sqrt{11^2 + (14-30.38)^2}$

$= 19.7348$

$e_1 = \begin{bmatrix} 11/19.73 \\ (14-\lambda_1)/19.73 \end{bmatrix} = \begin{bmatrix} 0.5574 \\ -0.8303 \end{bmatrix}$

For e_2,

$\begin{bmatrix} 0 \\ 0 \end{bmatrix} = (S - \lambda_2 I) u$

$= (14-\lambda_2)u_1 - 11u_2 = 0$

$-11u_2 + (23-\lambda_2)u_2 = 0$

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

5) $\begin{bmatrix} x_{1k} \\ x_{2k} \end{bmatrix}$

$e_1^T \begin{bmatrix} x_{1k} - \bar{x}_1 \\ x_{2k} - \bar{x}_2 \end{bmatrix} = [0.5574 \; -0.8303] \begin{bmatrix} x_{1k} - \bar{x}_1 \\ x_{2k} - \bar{x}_2 \end{bmatrix}$

$= 0.5574(x_{1k} - \bar{x}) - 0.8303(x_{2k} - \bar{x}_2)$

$= 0.5574(4-8) - 0.8303(11-8.5)$

$= -4.305$

x_1	4	8	13	7
x_2	11	4	5	14
1st PCA	-4.3052	3.7361	5.6928	3.1238

- What is PCA

-> It is a dimension reduction and ML method used to simplify the large data set into smaller set while maintaining significant patterns and trends.

Code:

```
from google.colab import files
heart=files.upload()

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from scipy import stats
import seaborn as sns
from sklearn.preprocessing import LabelEncoder,
OneHotEncoder  from sklearn.model_selection import
train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score  from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.decomposition import PCA

df1=pd.read_csv("heart (1).csv")
df1.head()
text_cols = df1.select_dtypes(include=['object']).columns
label_encoder = LabelEncoder()
for col in text_cols:
   df1[col] =
label_encoder.fit_transform(df1[col])
print(df1.head())
X = df1.drop('HeartDisease', axis=1)
y = df1['HeartDisease']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)  scaler = StandardScaler()
X_train =
scaler.fit_transform(X_train) X_test =
scaler.transform(X_test)
# Support Vector Machine
svm_model = SVC(kernel='linear', random_state=42)
svm_model.fit(X_train, y_train)
svm_predictions = svm_model.predict(X_test)
svm_accuracy = accuracy_score(y_test, svm_predictions)
print(f"SVM Accuracy: {svm_accuracy}")

# Logistic Regression
lr_model = LogisticRegression(random_state=42)
lr_model.fit(X_train, y_train) lr_predictions =
lr_model.predict(X_test) lr_accuracy =
accuracy_score(y_test, lr_predictions)
print(f"Logistic Regression Accuracy: {lr_accuracy}")
```

```
# Random Forest
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)
rf_predictions = rf_model.predict(X_test)
rf_accuracy = accuracy_score(y_test, rf_predictions)
print(f"Random Forest Accuracy: {rf_accuracy}")


models = {
    "SVM": svm_accuracy,
    "Logistic Regression":
    lr_accuracy, "Random Forest":
    rf_accuracy
}

best_model = max(models, key=models.get)
print(f"\nBest Model: {best_model} with accuracy {models[best_model]}")
pca = PCA(n_components=0.95)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)

svm_model_pca = SVC(kernel='linear', random_state=42)
svm_model_pca.fit(X_train_pca, y_train)
svm_predictions_pca = svm_model_pca.predict(X_test_pca)
svm_accuracy_pca = accuracy_score(y_test, svm_predictions_pca)
print(f"SVM Accuracy (with PCA): {svm_accuracy_pca}")

lr_model_pca = LogisticRegression(random_state=42)
lr_model_pca.fit(X_train_pca, y_train)
lr_predictions_pca = lr_model_pca.predict(X_test_pca)
lr_accuracy_pca = accuracy_score(y_test, lr_predictions_pca)
print(f"Logistic Regression Accuracy (with PCA): {lr_accuracy_pca}")

rf_model_pca = RandomForestClassifier(random_state=42)
rf_model_pca.fit(X_train_pca, y_train)
rf_predictions_pca = rf_model_pca.predict(X_test_pca)
rf_accuracy_pca = accuracy_score(y_test, rf_predictions_pca)
print(f"Random Forest Accuracy (with PCA): {rf_accuracy_pca}")

models_pca = {
    "SVM": svm_accuracy_pca,
    "Logistic Regression": lr_accuracy_pca,
    "Random Forest": rf_accuracy_pca
}

best_model_pca = max(models_pca, key=models_pca.get)
print(f"\nBest Model (with PCA): {best_model_pca} with accuracy {models_pca[best_model_pca]}")
```