VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

Shraddha (1BM22CS357)

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 **Shraddha (1BM22CS357),** who is bonafide student of **B.M.S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum. The Lab report has been approved as it satisfies the academic requirements in respect of an Machine Learning (23CS6PCMAL) work prescribed for the said degree.

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

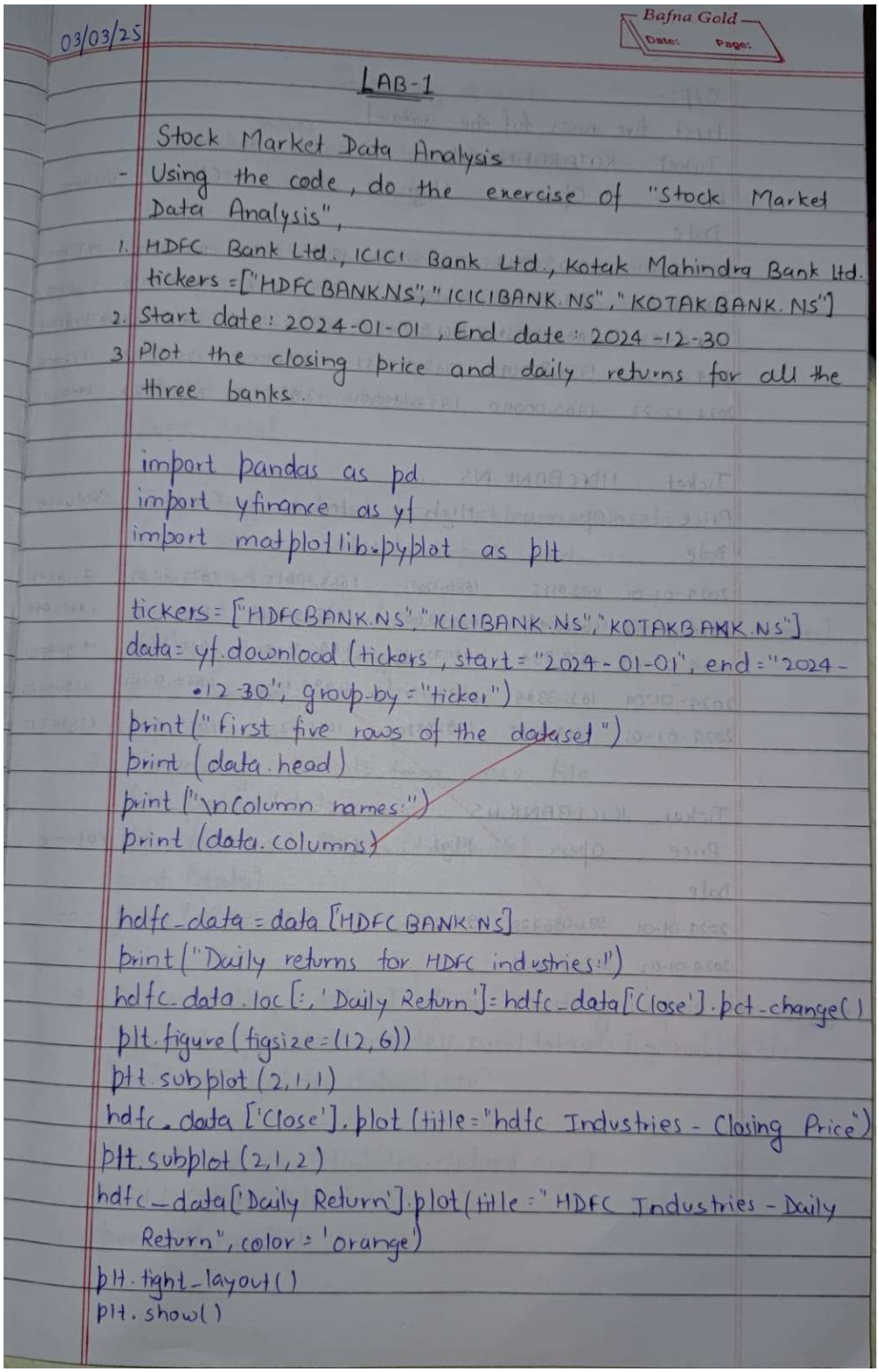
Index

Github Link:https://github.com/Shraddha-357/ML

Program 1

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

Screenshot



03/03/25

Bafna Gold
Date: Page:

LAB-1

Stock Market Data Analysis

- Using the code, do the exercise of "Stock Market Data Analysis",

1. HDFC Bank Ltd., ICICI Bank Ltd., Kotak Mahindra Bank Ltd.
   tickers = ["HDFCBANK.NS", "ICICIBANK.NS", "KOTAKBANK.NS"]
2. Start date: 2024-01-01, End date: 2024-12-30
3. Plot the closing price and daily returns for all the three banks.

```
import pandas as pd
import yfinance as yf
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 five rows of the dataset")
print(data.head)
print("\nColumn names:")
print(data.columns)

hdfc_data = data['HDFCBANK.NS']
print("Daily returns for HDFC industries:")
hdfc_data.loc[:,'Daily Return'] = hdfc_data['Close'].pct_change()
plt.figure(figsize=(12,6))
plt.subplot(2,1,1)
hdfc_data['Close'].plot(title="hdfc Industries - Closing Price")
plt.subplot(2,1,2)
hdfc_data['Daily Return'].plot(title="HDFC Industries - Daily
    Return", color='orange')
plt.tight_layout()
plt.show()
```

O/P:-

First five rows fof the dataset:

Ticker KOTAKBANK.NS

Price	Open	High	Low	Close	Volume
Date					
2024-01-01	1906.909914	1916.899006	1891.027338	1907.03814	1428902
2024-01-02	1903.911108	1905.91108	1858.063525	1863.008179	5120796
2024-01-03	1861.959234	1861.952665	1845.162158	1863.857178	3781815
2024-12-26	1756.800049	1769.449951	1742.050049	1752.800049	1132005
2024-12-27	1760.00000	1777.199951	1744.00000	1753.90024	2448221

Tickot HDFCBANK.NS

Price	Open	High	Low	Close	Volume
Dode					
2024-01-01	1683.8175	1686.1251	1669.20615	1675.2239	7119843
2024-01-02	167.9146	1679.8667	1665.9456	1676.2105	14612046
2024-01-03	1679.0514	1691.735069	1646.9180	1650.3635	14194481
2024-01-04	1655.3949	1672.116520	1648.1930	1668.07177	13367028
2024-01-05	1664.4215	1691.932477	1695.6281	1659.5382	15944735

Ticker ICICIBANK.NS

Price	Open	High	Low	Close	Volume
Date					
2024-01-01	983.086778	976.2734			---
2024-01-02	988.490253	---	---	---	---

Bafna Gold
Date: Page:

Importing datasets.

1. Initialise values directly into Dataframe

```
import pandas as pd
data = {
    'USN' : ['CS001', 'CS002', 'CS003', 'CS004', 'CS005'],
    'Name' : ['Alice', 'Bob', 'Charlie', 'David', 'Eve'],
    'Marks': ['96, 97, 94, 83, 59]
}
df = pd.DataFrame(data)
print(data)
```

2. Importing dataset from sklearn.datasets.

```
from sklearn.datasets import load_iris
iris = load_iris()
df = pd.DataFrame(iris.data, column = iris.feature_names)
df['target'] = iris.target
print("sample data:")
```

3. Importing datasets from .csv file

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

4. Downloading dataset from existing repos like Kaggle.

Kaggle: https://data.medely.com/datasets/wj9rwkp9c2/1
Download "diabetes-dataset.csv"

```
df = pd.read_csv("diabetes-dataset.csv")
print("sample data:")
print(df.head()).
```

Output:-

1.

	USN	Name	Marks
0	CS001	Alice	96
1	CS002	Bob	97
2	CS003	Charlie	94
3	CS004	David	83
4	CS005	Eve	84

2.

	sepal length	sepal width	peta length	petal width	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.8	1.4	a2	0
2	4.7	3.0	1.3	0.2	0

3.

	ID	Name	Age	City
0	1	Alice	25	New York
1	2	Bob	38	Los Angeles
2	3	Charlie	40	Chicago

10/10

3/3/25

Code:

```
import pandas as pd

import yfinance as yf

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 five rows of the dataset:")

print(data.head())


print("\nColumn names:")

print(data.columns)


# Summary statistics for a specific stock (e.g., HDFC)

hdfc_data = data['HDFCBANK.NS']

print("\nSummary statistics for hdfc Industries:")

print(hdfc_data.describe())


# Calculate daily returns

print("\ndaily returns for hdfc Industries:")

# Access the 'Close' column directly using its name

hdfc_data.loc[:, 'Daily Return'] = hdfc_data['Close'].pct_change()
```

Plot the closing price and daily returns

plt.figure(figsize=(12, 6))

plt.subplot(2, 1, 1)

hdfc_data['Close'].plot(title="hdfc Industries - Closing Price")

plt.subplot(2, 1, 2)

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

plt.tight_layout()

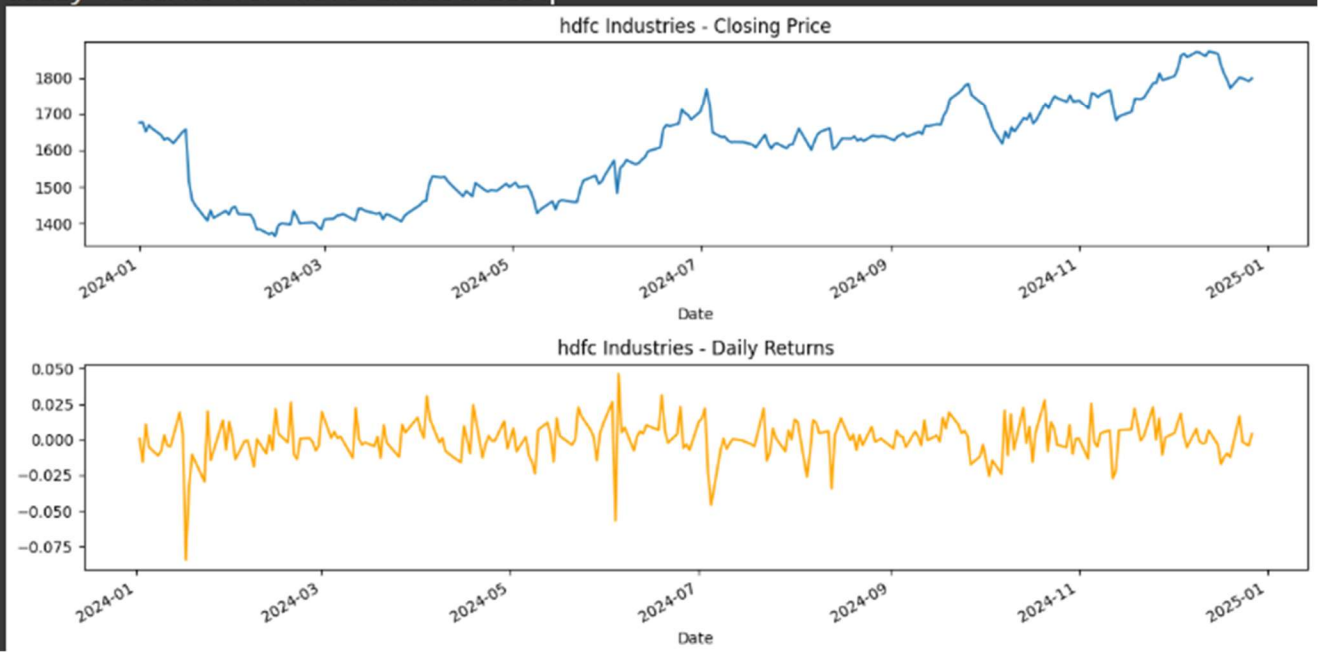
plt.show()

```
Ticker      HDFCBANK.NS                                                    \
Price             Open         High          Low        Close    Volume
Date
2024-01-01  1683.017598  1686.125187  1669.206199  1675.223999   7119843
2024-01-02  1675.914685  1679.860799  1665.950651  1676.210571  14621046
2024-01-03  1679.071480  1681.735059  1646.466666  1650.363525  14194881
2024-01-04  1655.394910  1672.116520  1648.193203  1668.071777  13367028
2024-01-05  1664.421596  1681.932477  1645.628180  1659.538208  15944735

Ticker     KOTAKBANK.NS                                                    \
Price             Open         High          Low        Close   Volume
Date
2024-01-01  1906.909954  1916.899006  1891.027338  1907.059814  1425902
2024-01-02  1905.911108  1905.911108  1858.063525  1863.008179  5120796
2024-01-03  1861.959234  1867.952665  1845.627158  1863.857178  3781515
2024-01-04  1869.451068  1869.451068  1858.513105  1861.559692  2865766
2024-01-05  1863.457575  1867.852782  1839.383985  1845.577148  7799341

Ticker     ICICIBANK.NS
Price             Open        High         Low       Close    Volume
Date
2024-01-01   983.086778  996.273246  982.541485  990.869812   7683792
2024-01-02   988.490253  989.134730  971.883221  973.866150  16263825
2024-01-03   976.295294  979.567116  966.777197  975.650818  16826752
2024-01-04   977.980767  980.707295  973.519176  978.724365  22789140
2024-01-05   979.567084  989.779158  975.402920  985.218445  14875499

Column names:
MultiIndex([( 'HDFCBANK.NS',   'Open'),
            ( 'HDFCBANK.NS',   'High'),
            ( 'HDFCBANK.NS',    'Low'),
            ( 'HDFCBANK.NS',  'Close'),
            ( 'HDFCBANK.NS', 'Volume'),
            ('KOTAKBANK.NS',   'Open'),
```

Summary statistics for hdfc Industries:

Price	Open	High	Low	Close	Volume
count	244.000000	244.000000	244.000000	244.000000	2.440000e+02
mean	1601.375295	1615.443664	1588.221245	1601.898968	2.119658e+07
std	134.648125	134.183203	132.796819	133.748372	2.133860e+07
min	1357.463183	1372.754374	1345.180951	1365.404785	8.798460e+05
25%	1475.316358	1494.072805	1460.259509	1474.564087	1.274850e+07
50%	1627.724976	1638.350037	1616.000000	1625.950012	1.686810e+07
75%	1696.474976	1711.425018	1679.250000	1697.062531	2.295014e+07
max	1877.699951	1880.000000	1858.550049	1871.750000	2.226710e+08

daily returns for hdfc Industries:



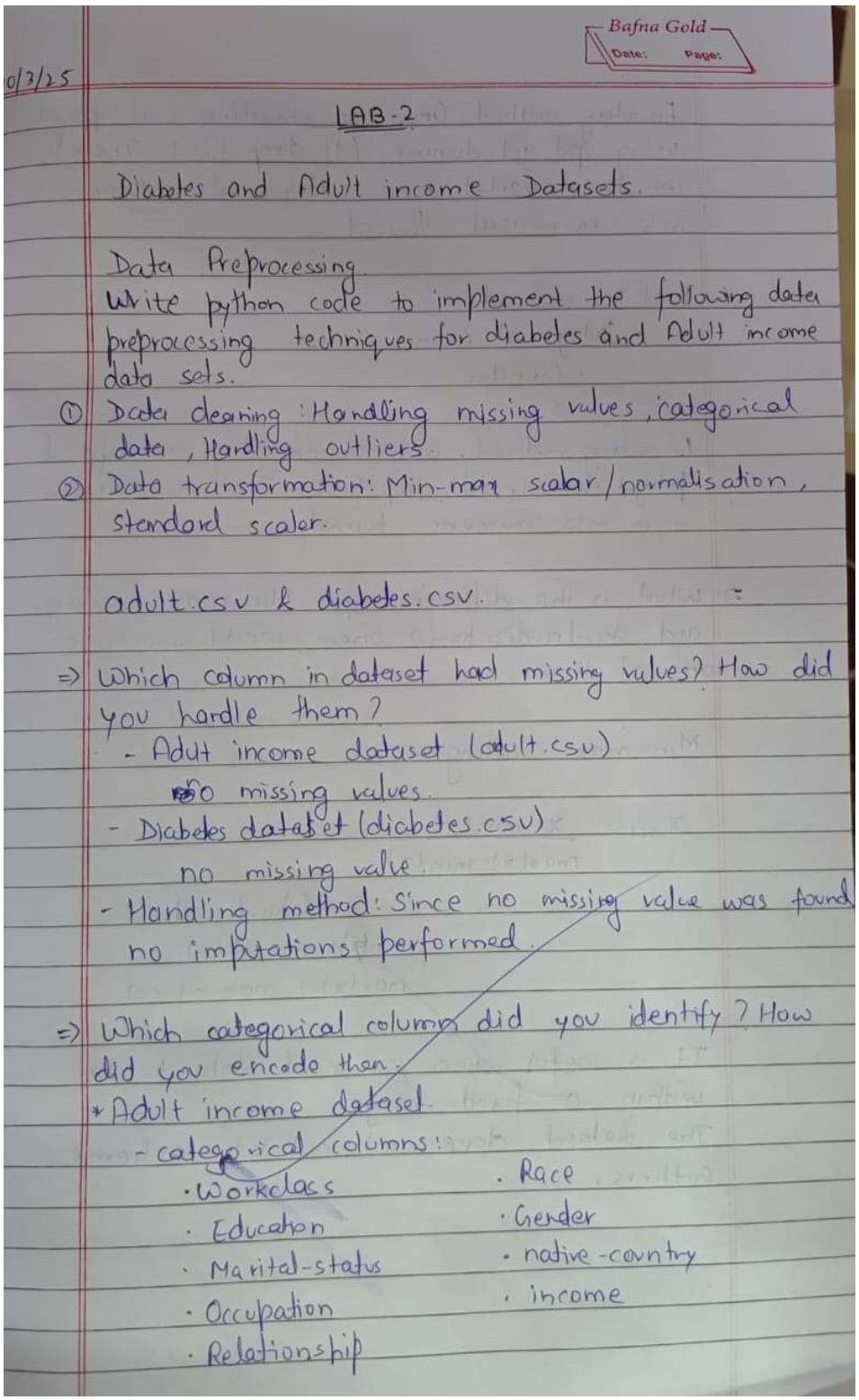

```
import pandas as pd
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David'],
    'Age': [25, 30, 35, 40],
    'City': ['New York', 'Los Angeles', 'Chicago', 'Houston']
}
df = pd.DataFrame(data)
print("Sample data:")
print(df.head())

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())

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 = '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())
```

Program 2

Demonstrate various data pre-processing techniques for a given dataset.

Screenshot



10/3/25

LAB-2

Diabetes and Adult income Datasets.

Data Preprocessing.
Write python code to implement the following data preprocessing techniques for diabetes and Adult income data sets.

① Data cleaning: Handling missing values, categorical data, Handling outliers.

② Data transformation: Min-max scalar / normalisation, standard scaler.

adult.csv & diabetes.csv.

⇒ Which column in dataset had missing values? How did you handle them?

- Adut income dataset (adult.csv)
  no missing values.
- Diabetes dataset (diabetes.csv)
  no missing value.
- Handling method: Since no missing value was found no imputations performed.

⇒ Which categorical column did you identify? How did you encode them.

* Adult income dataset.
- categorical columns:
  - Workclass
  - Education
  - Marital-status
  - Occupation
  - Relationship
  - Race
  - Gender
  - native-country
  - income

Encoding method: One hot encoding was applied using pd.get_dummies (df, drop_first = True), which converted these categorical columns. into numerical format.

Diabetes dataset:
- Categorical column:
  - Gender
  - SS

Encoding method.: one hot encoding was applied. pd.get dummies (df, drop-first = True) to transform them into numeric format.

⇒ What is the difference b/w Min-max scaling and standardization? Where would you use one of them.

Min-max Scaling :-

$$x_{std} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

x → dataset value
$\min(x)$ → min of col
$\max(x)$ → max of col.

It is useful when you need data to be within a fixed range.
The dataset does not contain extreme Outliers.

Bafna Gold
Date: Page:

Standardization:

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

$\mu \rightarrow$ mean
$\sigma \rightarrow$ std. deviation

It is useful when the dataset have varying units and a normal distribution.
There are significant outliers as standarisation is less sensitive to them.

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

try:
    diabetes_df = pd.read_csv('diabetes.csv')
    adult_df = pd.read_csv('adult.csv')
except FileNotFoundError:
    print("Error: Please upload 'diabetes.csv' and 'adult.csv' to your Google Colab environment.")
    exit()

diabetes_df.head(10)
adult_df.head(10)

diabetes_df.shape
adult_df.shape

#Handling Missing Values
diabetes_numeric_cols = diabetes_df.select_dtypes(include=[np.number]).columns
diabetes_categorical_cols = diabetes_df.select_dtypes(exclude=[np.number]).columns

adult_numeric_cols = adult_df.select_dtypes(include=[np.number]).columns
adult_categorical_cols = adult_df.select_dtypes(exclude=[np.number]).columns

diabetes_numeric_imputer = SimpleImputer(strategy='mean')
adult_numeric_imputer = SimpleImputer(strategy='mean')
diabetes_df[diabetes_numeric_cols] =
diabetes_numeric_imputer.fit_transform(diabetes_df[diabetes_numeric_cols])
adult_df[adult_numeric_cols] = adult_numeric_imputer.fit_transform(adult_df[adult_numeric_cols])
```

```
diabetes_categorical_imputer = SimpleImputer(strategy='most_frequent')
adult_categorical_imputer = SimpleImputer(strategy='most_frequent')
diabetes_df[diabetes_categorical_cols] =
diabetes_categorical_imputer.fit_transform(diabetes_df[diabetes_categorical_cols])
adult_df[adult_categorical_cols] =
adult_categorical_imputer.fit_transform(adult_df[adult_categorical_cols])

print("Missing values in Diabetes dataset after imputation:")
print(diabetes_df.isnull().sum())
print("Missing values in Adult Income dataset after imputation:")
print(adult_df.isnull().sum())

adult_df.replace("?", np.nan, inplace=True)
print("Missing values in Adult Income dataset after replacing '?':")
print(adult_df.isnull().sum())

from sklearn.impute import SimpleImputer

# Identify numeric and categorical columns
adult_numeric_cols = adult_df.select_dtypes(include=[np.number]).columns
adult_categorical_cols = adult_df.select_dtypes(exclude=[np.number]).columns

# Handle missing values in numeric columns using mean imputation
adult_numeric_imputer = SimpleImputer(strategy='mean')
adult_df[adult_numeric_cols] = adult_numeric_imputer.fit_transform(adult_df[adult_numeric_cols])

# Handle missing values in categorical columns using most frequent imputation
adult_categorical_imputer = SimpleImputer(strategy='most_frequent')
adult_df[adult_categorical_cols] =
adult_categorical_imputer.fit_transform(adult_df[adult_categorical_cols])
print("Missing values in Adult Income dataset after imputation:")
```

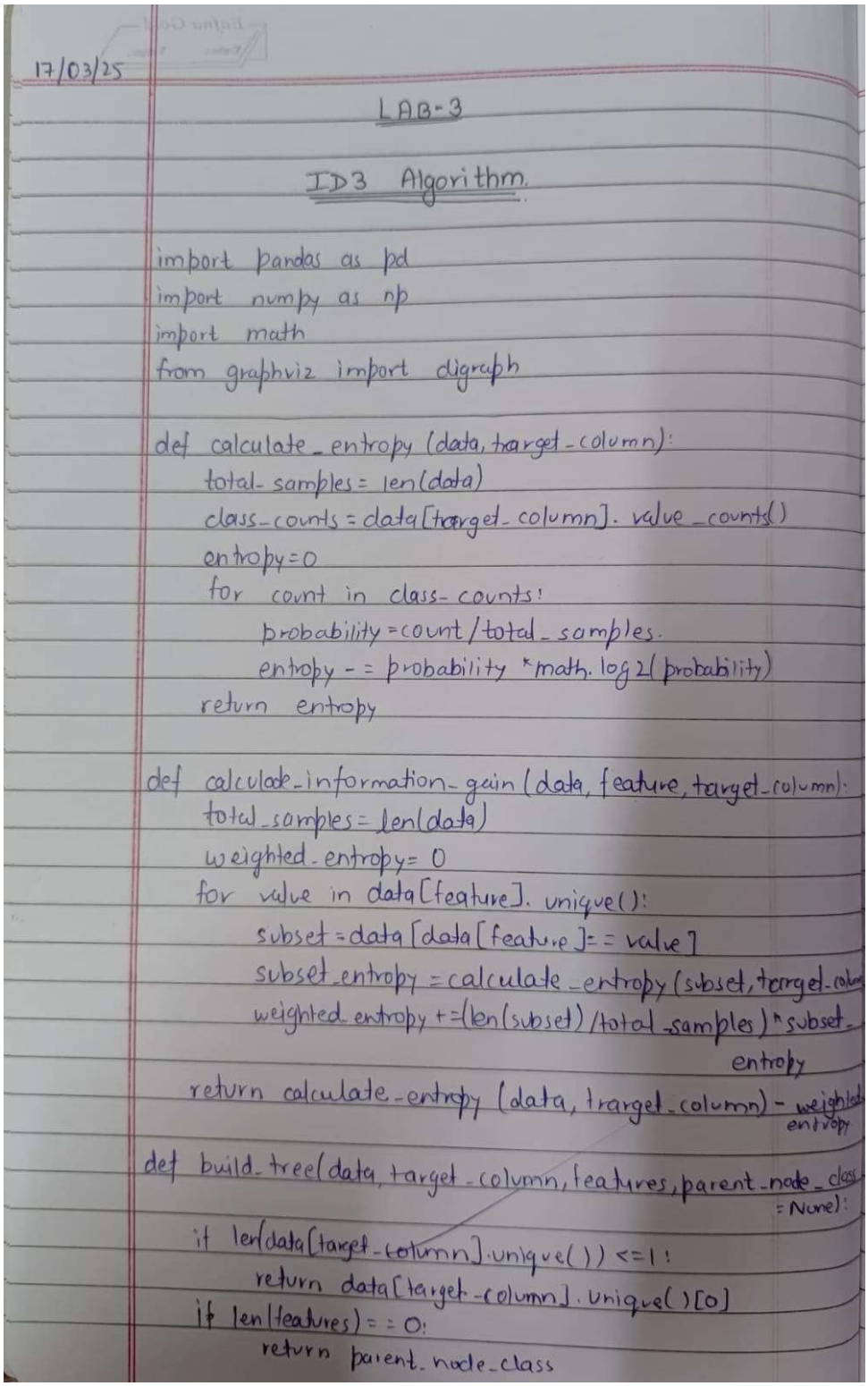
```
print(adult_df.isnull().sum())

from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()

# Encode categorical columns in Diabetes dataset
for col in diabetes_categorical_cols:
    diabetes_df[col] = label_encoder.fit_transform(diabetes_df[col])

# Encode categorical columns in Adult Income dataset
for col in adult_categorical_cols:
    adult_df[col] = label_encoder.fit_transform(adult_df[col])
print("Encoded columns in Diabetes dataset:")
print(diabetes_df.head())
print("Encoded columns in Adult Income dataset:")
print(adult_df.head())

#Handling outliers
def remove_outliers(df):
    Q1 = df.quantile(0.25)
    Q3 = df.quantile(0.75)
    IQR = Q3 - Q1
    df_no_outliers = df[~((df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR))).any(axis=1)]
    return df_no_outliers

diabetes_df_no_outliers = remove_outliers(diabetes_df)
adult_df_no_outliers = remove_outliers(adult_df)
print("Diabetes dataset shape after removing outliers:", diabetes_df_no_outliers.shape)
print("Adult Income dataset shape after removing outliers:", adult_df_no_outliers.shape)

#Min-max scaling
from sklearn.preprocessing import MinMaxScaler
```

```
min_max_scaler = MinMaxScaler()
diabetes_scaled_minmax = pd.DataFrame(min_max_scaler.fit_transform(diabetes_df_no_outliers),
columns=diabetes_df_no_outliers.columns)
adult_scaled_minmax = pd.DataFrame(min_max_scaler.fit_transform(adult_df_no_outliers),
columns=adult_df_no_outliers.columns)
print("Diabetes dataset after Min-Max scaling:")
print(diabetes_scaled_minmax.head())
print("Adult Income dataset after Min-Max scaling:")
print(adult_scaled_minmax.head())

# Initialize Standard Scaler
from sklearn.preprocessing import StandardScaler
standard_scaler = StandardScaler()
diabetes_scaled_standard = pd.DataFrame(standard_scaler.fit_transform(diabetes_df_no_outliers),
columns=diabetes_df_no_outliers.columns)
adult_scaled_standard = pd.DataFrame(standard_scaler.fit_transform(adult_df_no_outliers),
columns=adult_df_no_outliers.columns)
print("Diabetes dataset after Standard scaling:")
print(diabetes_scaled_standard.head())
print("Adult Income dataset after Standard scaling:")
print(adult_scaled_standard.head())
```

Program 3

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

Screenshot



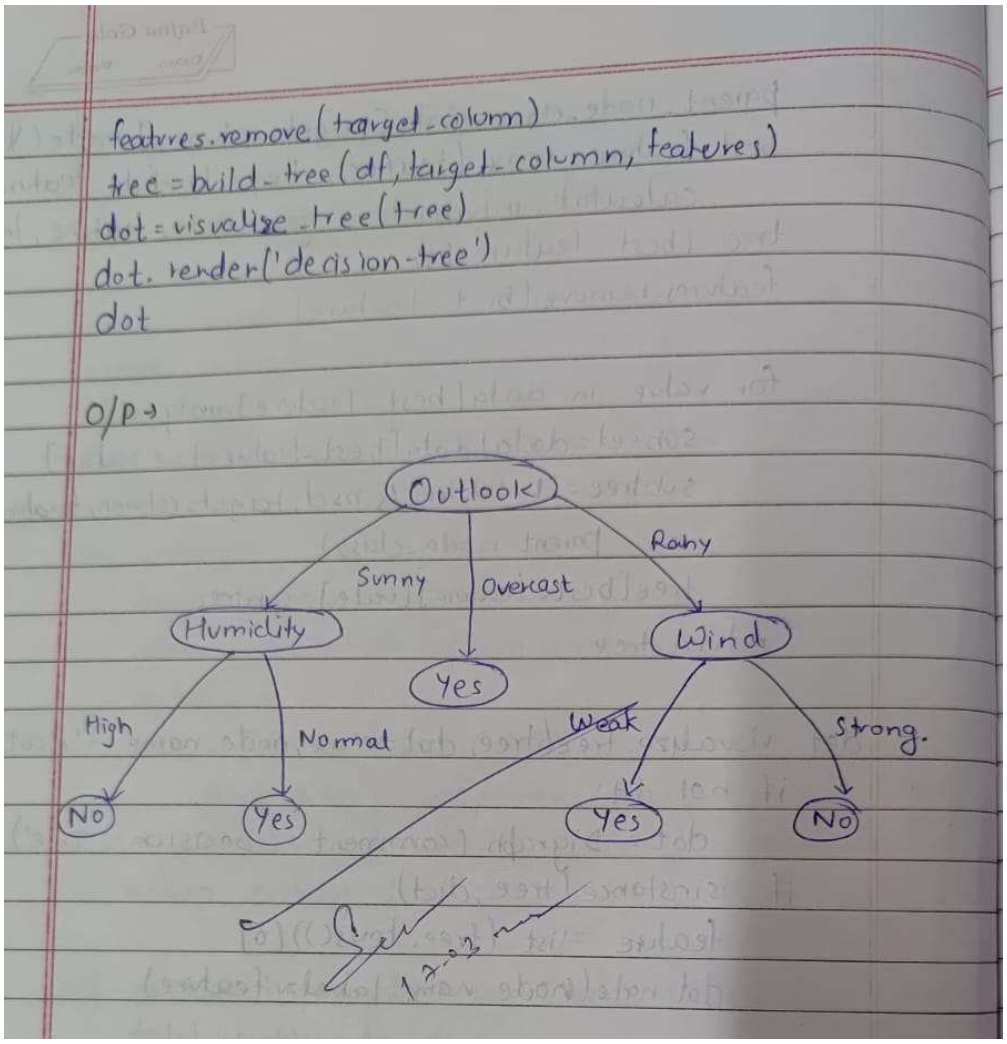
17/03/25

LAB-3

ID3 Algorithm.

```
import pandas as pd
import numpy as np
import math
from graphviz import digraph

def calculate_entropy(data, target_column):
    total_samples = len(data)
    class_counts = data[target_column].value_counts()
    entropy = 0
    for count in class_counts:
        probability = count / total_samples
        entropy -= probability * math.log2(probability)
    return entropy

def calculate_information_gain(data, feature, target_column):
    total_samples = len(data)
    weighted_entropy = 0
    for value in data[feature].unique():
        subset = data[data[feature] == value]
        subset_entropy = calculate_entropy(subset, target_colu
        weighted_entropy += (len(subset) / total_samples) * subset_
                                                        entropy
    return calculate_entropy(data, target_column) - weighted_
                                                        entropy

def build_tree(data, target_column, features, parent_node_class
                                                        = None):
    if len(data[target_column].unique()) <= 1:
        return data[target_column].unique()[0]
    if len(features) == 0:
        return parent_node_class
```

```
    parent_node_class = data[target_column].mode()[0]
    best_feature = max(features, key = lambda feature:
        calculate_information_gain(data, feature, target_col
    tree = {best_feature: {}}
    features.remove(best_feature)

    for value in data[best_feature].unique():
        subset = data[data[best_feature] == value]
        subtree = build_tree(subset, target_column, features.copy()
            parent_node_class)
        tree[best_feature][value] = subtree
    return tree

def visualize_tree(tree, dot = None, node_name = 'Root'):
    if not dot:
        dot = Digraph(comment = 'Decision Tree')
    if isinstance(tree, dict):
        feature = list(tree.keys())[0]
        dot.node(node_name, label = feature)

        for value, subtree in tree[feature].items():
            child_node_name = node_name + "_" + str(value)
            dot.edge(node_name, child_node_name, label = str(value))
            visualize_tree(subtree, dot, child_node_name)
    else:
        dot.node(node_name, label = str(tree))
    return dot

data =
df = pd.DataFrame(data)

target_column = 'Weather'
features = list(df.columns)
```



Code:

prompt: Decision tree is perfect but do not generate directly use entropy and informatin gain calculastion (ID3) to generate the exact tree now and visualize the output with graphviz in the form of tree

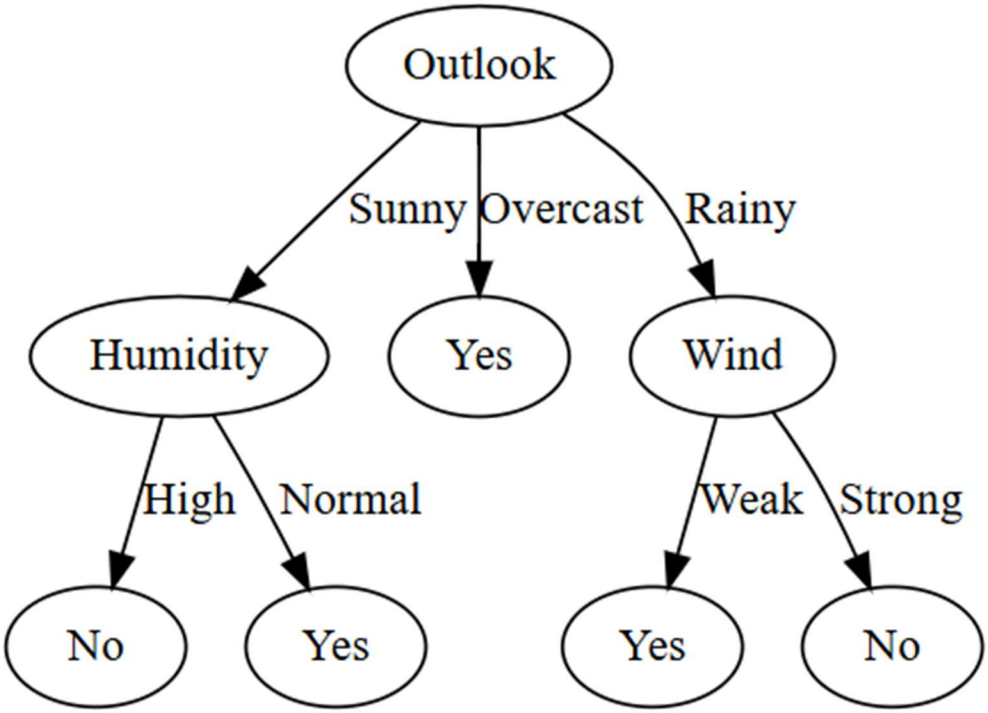
```
import pandas as pd

import numpy as np

import math

from graphviz import Digraph


def calculate_entropy(data, target_column):
```

```
    """Calculates the entropy of a dataset."""
    total_samples = len(data)
    class_counts = data[target_column].value_counts()
    entropy = 0
    for count in class_counts:
        probability = count / total_samples
        entropy -= probability * math.log2(probability)
    return entropy

def calculate_information_gain(data, feature, target_column):
    """Calculates the information gain of a feature."""
    total_samples = len(data)
    weighted_entropy = 0
    for value in data[feature].unique():
        subset = data[data[feature] == value]
        subset_entropy = calculate_entropy(subset, target_column)
        weighted_entropy += (len(subset) / total_samples) * subset_entropy
    return calculate_entropy(data, target_column) - weighted_entropy

def build_tree(data, target_column, features, parent_node_class=None):
    # Base cases
    if len(data[target_column].unique()) <= 1:
        return data[target_column].unique()[0]
    if len(features) == 0:
```

```
        return parent_node_class



    parent_node_class = data[target_column].mode()[0]  # Set mode of the target feature as parent node class



    # Find the best feature to split on

    best_feature = max(features, key=lambda feature: calculate_information_gain(data, feature, target_column))

    tree = {best_feature: {}}

    features.remove(best_feature) #Remove the feature that has been used already



    # Recursively build the tree

    for value in data[best_feature].unique():

        subset = data[data[best_feature] == value]

        subtree = build_tree(subset, target_column, features.copy(), parent_node_class) #Create subtree recursively with current subset of data

        tree[best_feature][value] = subtree



    return tree



def visualize_tree(tree, dot=None, node_name='Root'):

    if not dot:

        dot = Digraph(comment='Decision Tree')



    if isinstance(tree, dict):

        feature = list(tree.keys())[0]
```

```
        dot.node(node_name, label=feature)

        for value, subtree in tree[feature].items():
            child_node_name = node_name + "_" + str(value)
            dot.edge(node_name, child_node_name, label=str(value))
            visualize_tree(subtree, dot, child_node_name)

    else:
        dot.node(node_name, label=str(tree))  # Leaf node

    return dot

# Sample data (Replace with your actual data)
data = {'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rainy', 'Rainy', 'Rainy', 'Overcast', 'Sunny', 'Sunny',
'Rainy', 'Sunny', 'Overcast', 'Overcast', 'Rainy'],
        'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Cool', 'Mild', 'Cool', 'Mild', 'Mild', 'Mild',
'Hot', 'Mild'],
        'Humidity': ['High', 'High', 'High', 'High', 'Normal', 'Normal', 'Normal', 'High', 'Normal',
'Normal', 'Normal', 'High', 'Normal', 'High'],
        'Wind': ['Weak', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong', 'Strong', 'Weak', 'Weak', 'Weak',
'Strong', 'Strong', 'Weak', 'Strong'],
        'PlayTennis': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']}
df = pd.DataFrame(data)

target_column = 'PlayTennis'
features = list(df.columns)
```

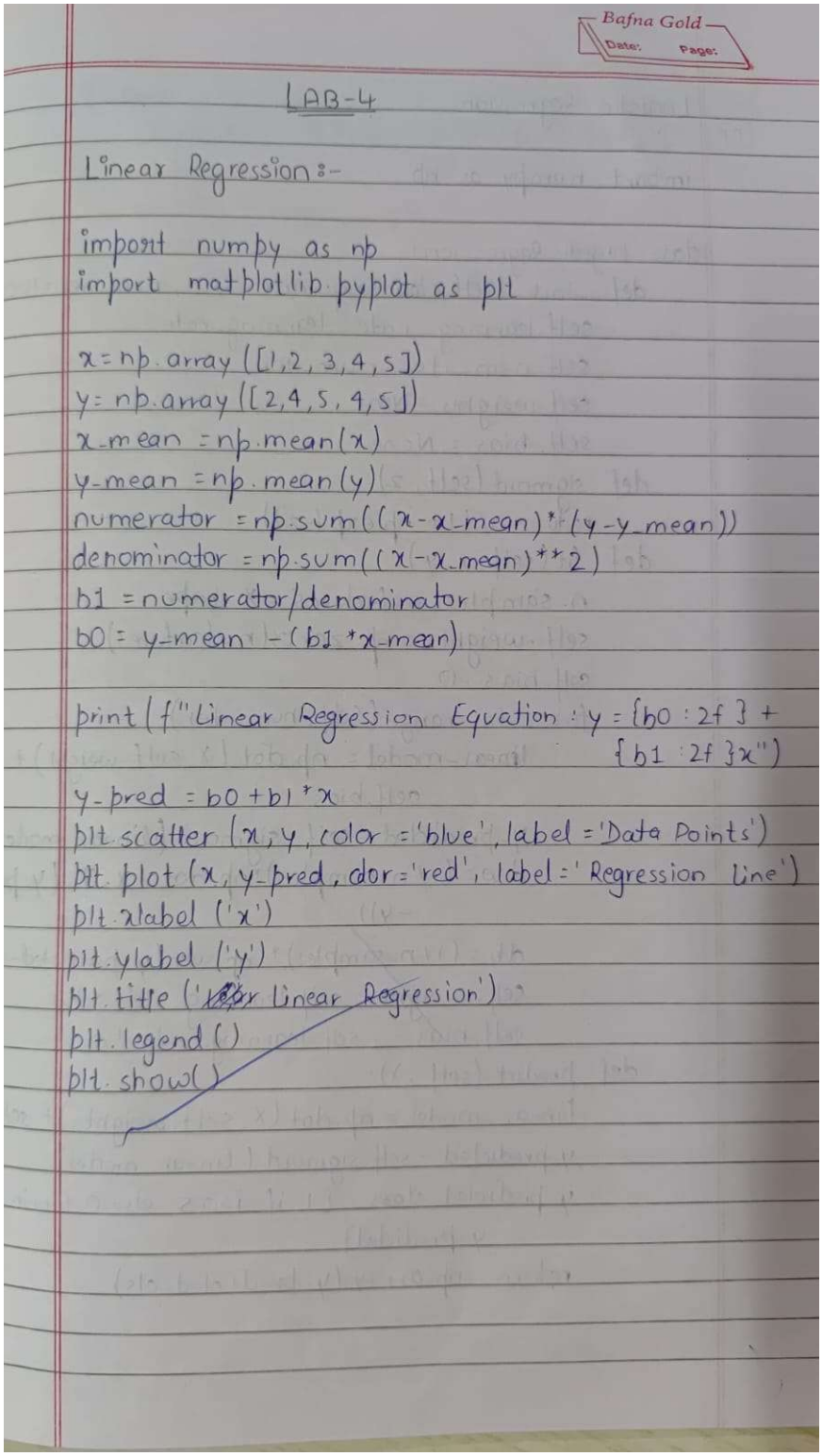
```
features.remove(target_column)

tree = build_tree(df, target_column, features)

dot = visualize_tree(tree)

dot.render('decision_tree')

dot
```



Program 4

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Screenshot



LAB-4

Linear Regression :-

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

x = np.array([1,2,3,4,5])
y = np.array([2,4,5,4,5])
x_mean = np.mean(x)
y_mean = np.mean(y)
numerator = np.sum((x-x_mean)*(y-y_mean))
denominator = np.sum((x-x_mean)**2)
b1 = numerator/denominator
b0 = y_mean - (b1*x_mean)

print(f"Linear Regression Equation : y = {b0:2f} +
                                    {b1:2f}x")
y_pred = b0 + b1*x
plt.scatter(x, y, color = 'blue', label = 'Data Points')
plt.plot(x, y_pred, dor = 'red', label = 'Regression Line')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Linear Regression')
plt.legend()
plt.show()
```

Code:

```
import pandas as pd
import numpy as np
from sklearn import linear_model
import matplotlib.pyplot as plt

df = pd.read_csv('housing_area_price.csv')
plt.xlabel('area')
plt.ylabel('price')
plt.scatter(df.area,df.price,color='red',marker='+')
new_df = df.drop('price',axis='columns')
new_df
price = df.price
reg = linear_model.LinearRegression()
reg.fit(new_df,price)

#(1) Predict price of a home with area = 3300 sqr ft
reg.predict([[3300]])
reg.coef_
reg.intercept_
3300*135.78767123 + 180616.43835616432

#(2) Predict price of a home with area = 5000 sqr ft
reg.predict([[5000]])

df = pd.read_csv('homeprices_Multiple_LR.csv')
df.bedrooms.median()
df.bedrooms = df.bedrooms.fillna(df.bedrooms.median())
reg = linear_model.LinearRegression()
reg.fit(df.drop('price',axis='columns'),df.price)
reg.coef_
reg.intercept_
```

```
#Find price of home with 3000 sqr ft area, 3 bedrooms, 40 year old
reg.predict([[3000, 3, 40]])
112.06244194*3000 + 23388.88007794*3 + -3231.71790863*40 + 221323.00186540384

df = pd.read_csv('canada_per_capita_income.csv')
print(df.head())
X = df[['year']]
y = df['per capita income (US$)']
reg = LinearRegression()
reg.fit(X, y)
predicted_income_2020 = reg.predict([[2020]])
print(f"Predicted per capita income for Canada in 2020: {predicted_income_2020[0]:.2f}")

plt.scatter(X, y, color='blue')
plt.plot(X, reg.predict(X), color='red')
plt.xlabel('Year')
plt.ylabel('Per Capita Income')
plt.title('Per Capita Income in Canada Over the Years')
plt.show()

df = pd.read_csv('salary.csv')
print(df.head())
print("Missing values in the dataset:")
print(df.isnull().sum())

df['YearsExperience'] = df['YearsExperience'].fillna(df['YearsExperience'].median())
print("\nMissing values after filling:")
print(df.isnull().sum())
X = df[['YearsExperience']]
y = df['Salary']
reg = LinearRegression()
```

```
reg.fit(X, y)
predicted_salary_12_years = reg.predict([[12]])
print(f"\nPredicted salary for an employee with 12 years of experience:
${predicted_salary_12_years[0]:,.2f}")

plt.scatter(X, y, color='blue')
plt.plot(X, reg.predict(X), color='red')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.title('Salary vs. Years of Experience')
plt.show()

def convert_to_numeric(value):
    word_to_num = {
        'zero': 0, '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,
        'fifteen': 15
    }
    return word_to_num.get(value.lower(), value) if isinstance(value, str) else value

df_hiring = pd.read_csv('hiring.csv')
print(df.head())
df_hiring['experience'] = df_hiring['experience'].apply(convert_to_numeric)

df_hiring['experience'].fillna(0, inplace=True)
df_hiring['test_score(out of 10)'].fillna(df_hiring['test_score(out of 10)'].median(), inplace=True)
df_hiring['interview_score(out of 10)'].fillna(df_hiring['interview_score(out of 10)'].median(),
inplace=True)
X_hiring = df_hiring[['experience', 'test_score(out of 10)', 'interview_score(out of 10)']]
y_hiring = df_hiring['salary($)']
reg_hiring = LinearRegression()
```

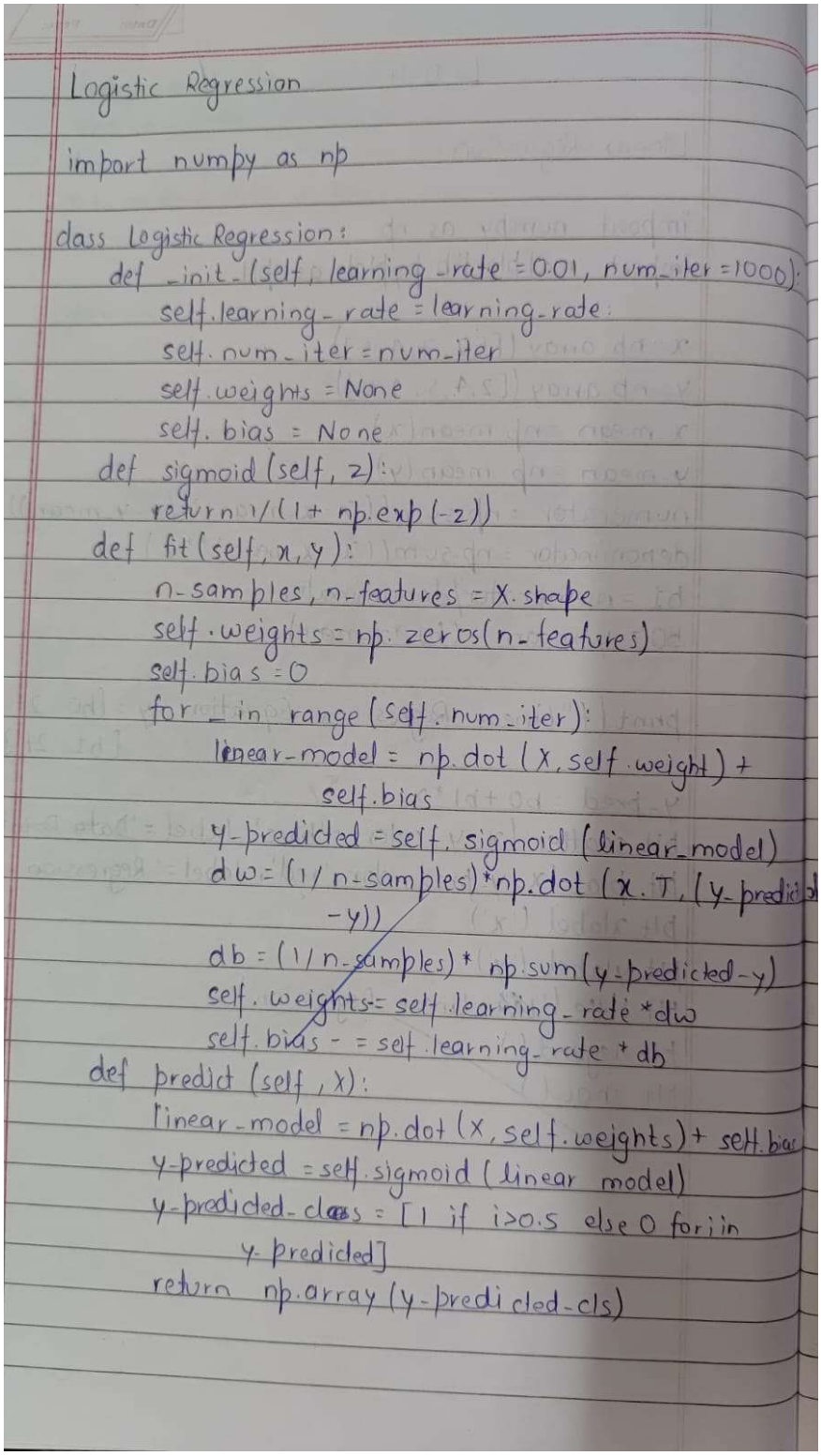
```
reg_hiring.fit(X_hiring, y_hiring)
candidates = np.array([[2, 9, 6], [12, 10, 10]])
predicted_salaries = reg_hiring.predict(candidates)

for i, candidate in enumerate(candidates):
    print(f"\nPredicted salary for candidate with {candidate[0]} yrs experience, {candidate[1]} test score,
{candidate[2]} interview score: {predicted_salaries[i]:.2f} USD")

plt.scatter(y_hiring, reg_hiring.predict(X_hiring), color='blue', label='Predicted vs Actual')
plt.xlabel("Actual Salary")
plt.ylabel("Predicted Salary")
plt.title("Actual vs Predicted Salary")
plt.legend()
plt.show()

df_companies = pd.read_csv('1000_Companies.csv')
print(df.head())
label_encoder = LabelEncoder()
df_companies['State'] = label_encoder.fit_transform(df_companies['State'])
X_companies = df_companies[['R&D Spend', 'Administration', 'Marketing Spend', 'State']]
y_companies = df_companies['Profit']
df_companies.fillna(df_companies.median(), inplace=True)
reg_companies = LinearRegression()
reg_companies.fit(X_companies, y_companies)

input_data = np.array([[91694.48, 515841.3, 11931.24, label_encoder.transform(['Florida'])[0]]])
predicted_profit = reg_companies.predict(input_data)
print(f"Predicted profit: {predicted_profit[0]:.2f} USD")

plt.scatter(y_companies, reg_companies.predict(X_companies), color='blue', label='Predicted vs
Actual')
plt.xlabel("Actual Profit")
```

```
plt.ylabel("Predicted Profit")
plt.title("Actual vs Predicted Profit")
plt.legend()
plt.show()
```

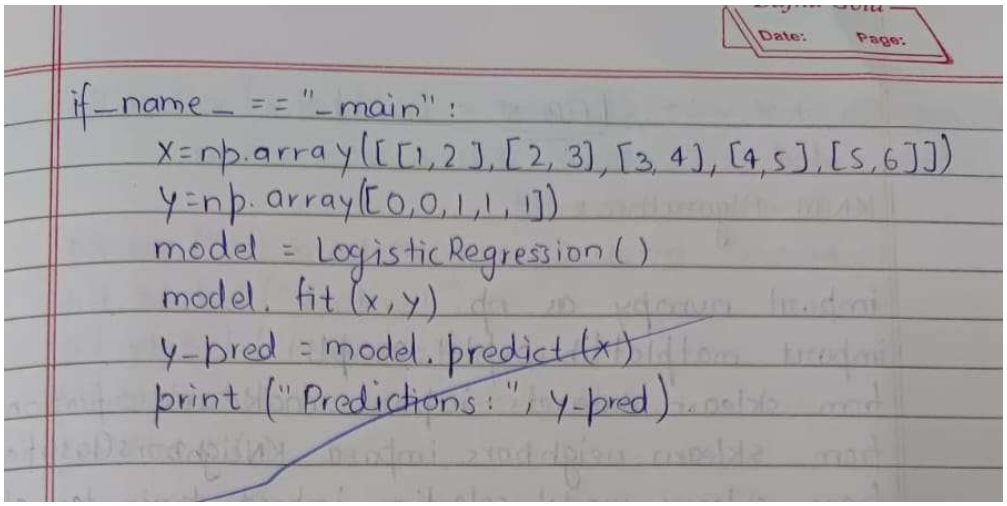
Program 5

Build Logistic Regression Model for a given dataset

Screenshot



```
Logistic Regression

import numpy as np

class LogisticRegression:
    def __init__(self, learning_rate=0.01, num_iter=1000):
        self.learning_rate = learning_rate
        self.num_iter = num_iter
        self.weights = None
        self.bias = None
    def sigmoid(self, z):
        return 1/(1+np.exp(-z))
    def fit(self, X, y):
        n_samples, n_features = X.shape
        self.weights = np.zeros(n_features)
        self.bias = 0
        for _ in range(self.num_iter):
            linear_model = np.dot(X, self.weight) +
                           self.bias
            y_predicted = self.sigmoid(linear_model)
            dw = (1/n_samples)*np.dot(X.T, (y_predic
                 -y))
            db = (1/n_samples)* np.sum(y_predicted-y)
            self.weights = self.learning_rate *dw
            self.bias -= self.learning_rate * db
    def predict(self, X):
        linear_model = np.dot(X, self.weights)+ self.bia
        y_predicted = self.sigmoid(linear model)
        y_predicted_class = [1 if i>0.5 else 0 for i in
            y_predicted]
        return np.array(y_predicted_cls)
```



Code:

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

df = pd.read_csv("HR_comma_sep.csv")
print(df.info())
numericCols = df.select_dtypes(include=['float64', 'int64']).columns
plt.figure(figsize=(10, 8))
sns.heatmap(df[numericCols].corr(), annot=True, cmap='coolwarm', fmt='.2f')
plt.title("Correlation Matrix (Numeric Features)")
plt.show()

plt.figure(figsize=(8, 6))
sns.countplot(x='salary', hue='left', data=df)
plt.title("Impact of Salary on Employee Retention")
plt.xlabel("Salary Level")
plt.ylabel("Employee Count")
plt.show()

import pandas as pd
df = pd.read_csv("zoo-data.csv")
print(df.info())
```

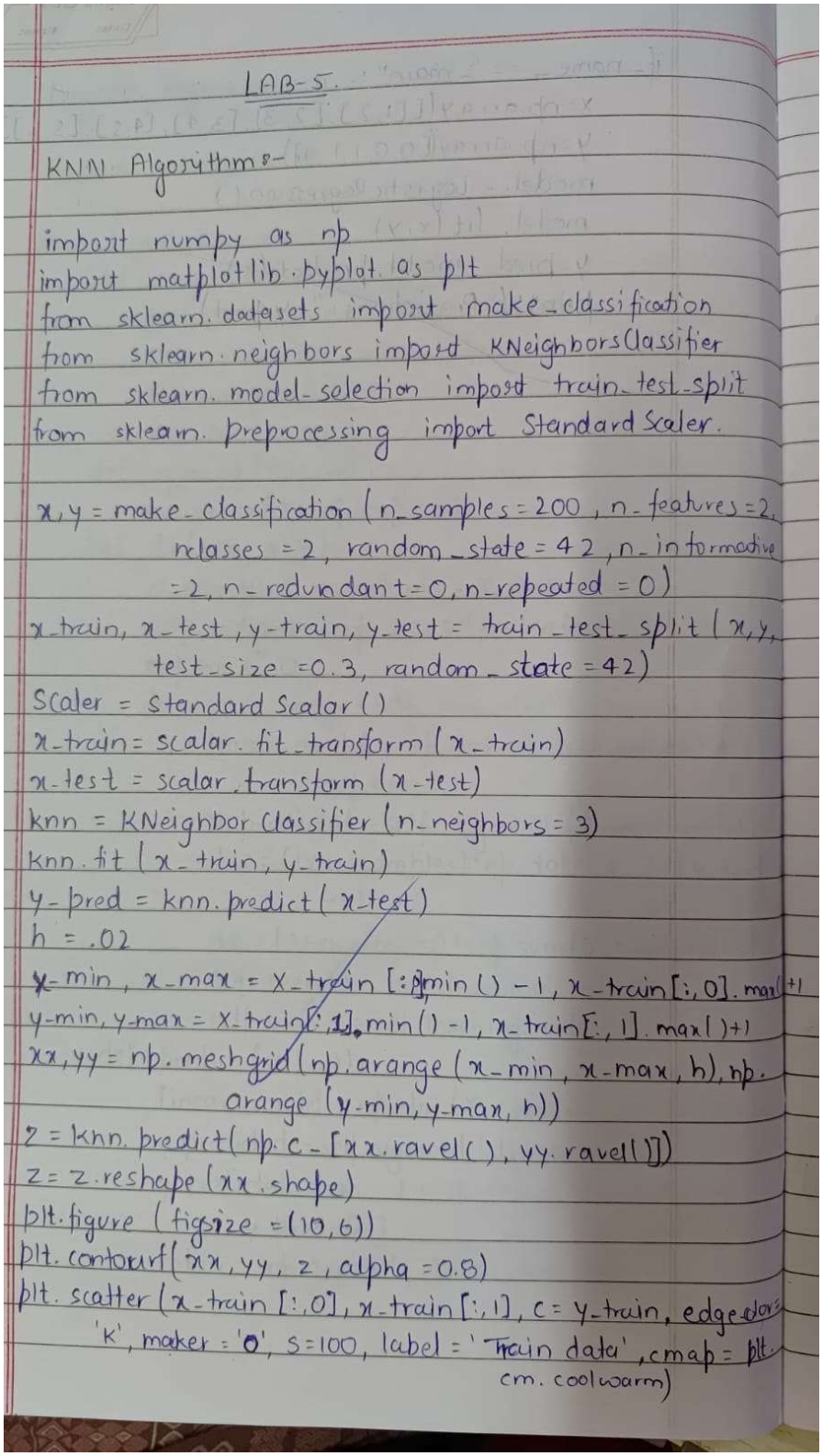
```
print(df.head())
print(df.isnull().sum())
df.drop(columns=['animal_name'], inplace=True)
X = df.drop(columns=['class_type'])
y = df['class_type']

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
logreg = LogisticRegression(max_iter=200, multi_class='multinomial', solver='lbfgs')
logreg.fit(X_train, y_train)

from sklearn.metrics import accuracy_score
y_pred = logreg.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy:.2f}")

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=logreg.classes_)
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix for Zoo Animal Classification")
plt.show()

y_pred = logreg.predict(X_test)
pred_classes = [class_mapping[pred] for pred in y_pred]
```

```
print("Predicted Classes:", pred_classes)

import seaborn as sns
import matplotlib.pyplot as plt
sns.countplot(x='class_type', data=df)
plt.title("Class Distribution of Animals in Zoo Dataset")
plt.xlabel("Class Type")
plt.ylabel("Count")
plt.show()

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
cm = confusion_matrix(y_test, y_pred)
class_labels = [class_mapping[num] for num in logreg.classes_]
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_labels)
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix with Class Names")
plt.show()
```

Program 6

Build KNN Classification model for a given dataset.

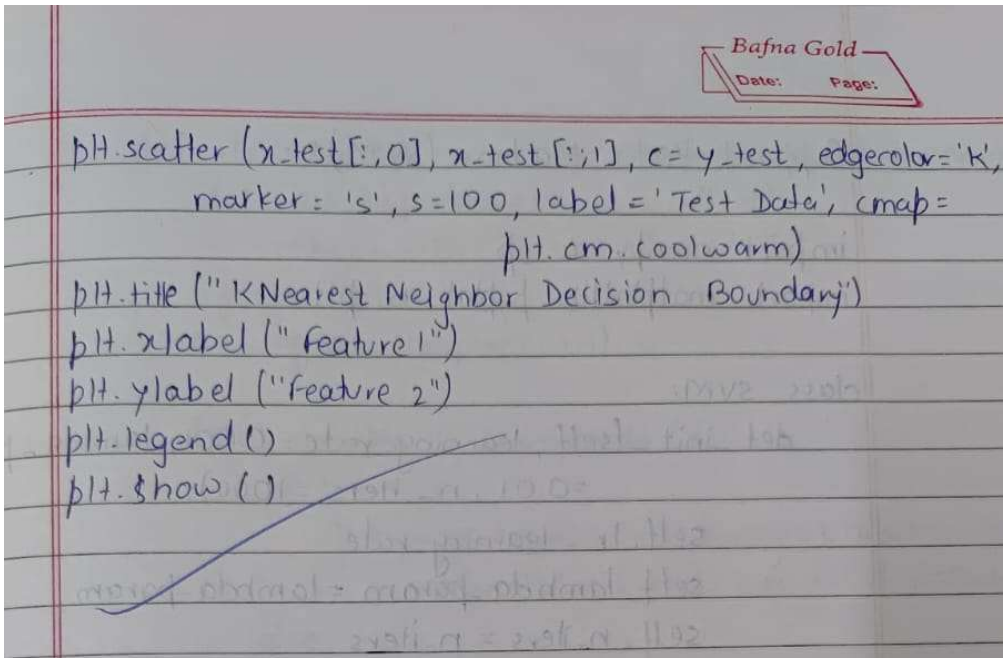
Screenshot



LAB-5.

KNN Algorithm:-

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler.

x,y = make_classification(n_samples=200, n_features=2,
        nclasses=2, random_state=42, n_informative
        =2, n_redundant=0, n_repeated=0)
x_train, x_test, y_train, y_test = train_test_split(x,y,
        test_size=0.3, random_state=42)
Scaler = StandardScaler()
x_train= scalar.fit_transform(x_train)
x_test = scalar.transform(x_test)
knn = KNeighborClassifier(n_neighbors=3)
knn.fit(x_train, y_train)
y_pred = knn.predict(x_test)
h = .02
x_min, x_max = X_train[:,0].min()-1, x_train[:,0].max()+1
y_min, y_max = X_train[:,1].min()-1, x_train[:,1].max()+1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.
        arange(y_min, y_max, h))
z = knn.predict(np.c_[xx.ravel(), yy.ravel()])
z = z.reshape(xx.shape)
plt.figure(figsize=(10,6))
plt.contourf(xx, yy, z, alpha=0.8)
plt.scatter(x_train[:,0], x_train[:,1], c=y_train, edgecolors
        'K', marker='o', s=100, label='Train data', cmap=plt.
        cm.coolwarm)
```

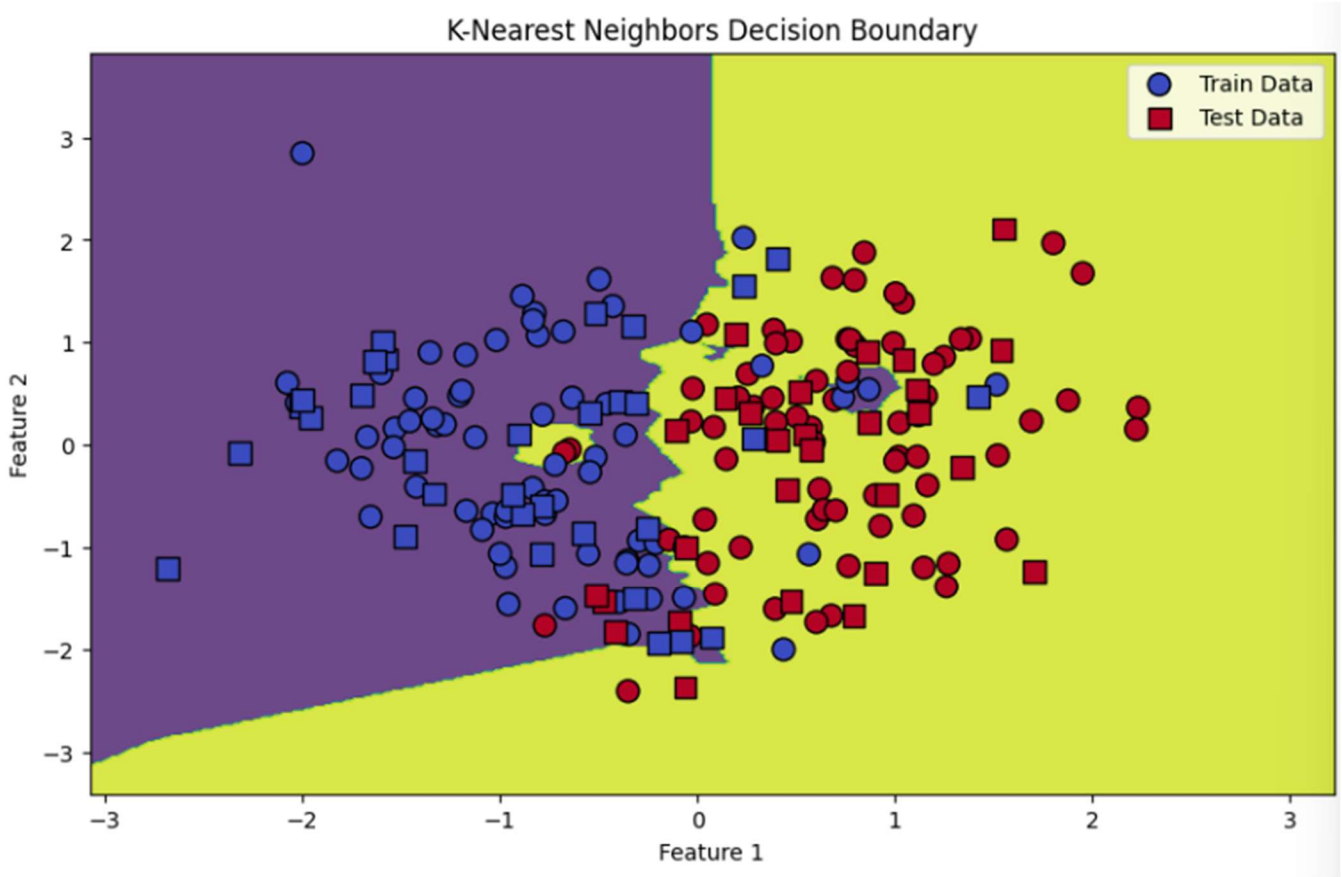


Code:

```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import make_classification

from sklearn.neighbors import KNeighborsClassifier

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler


# Generate a synthetic classification dataset

# Explicitly set n_informative, n_redundant, and n_repeated to be compatible with n_features=2

X, y = make_classification(n_samples=200, n_features=2, n_classes=2, random_state=42,

                           n_informative=2, n_redundant=0, n_repeated=0) #changed line


# Split data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
# Standardize the feature matrix
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# K-Nearest Neighbors Classifier (K=3 for this example)
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

# Make predictions
y_pred = knn.predict(X_test)

# Plotting decision boundary
h = .02  # Step size in the mesh grid
x_min, x_max = X_train[:, 0].min() - 1, X_train[:, 0].max() + 1
y_min, y_max = X_train[:, 1].min() - 1, X_train[:, 1].max() + 1

xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

# Predict for all points in the grid
Z = knn.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
```

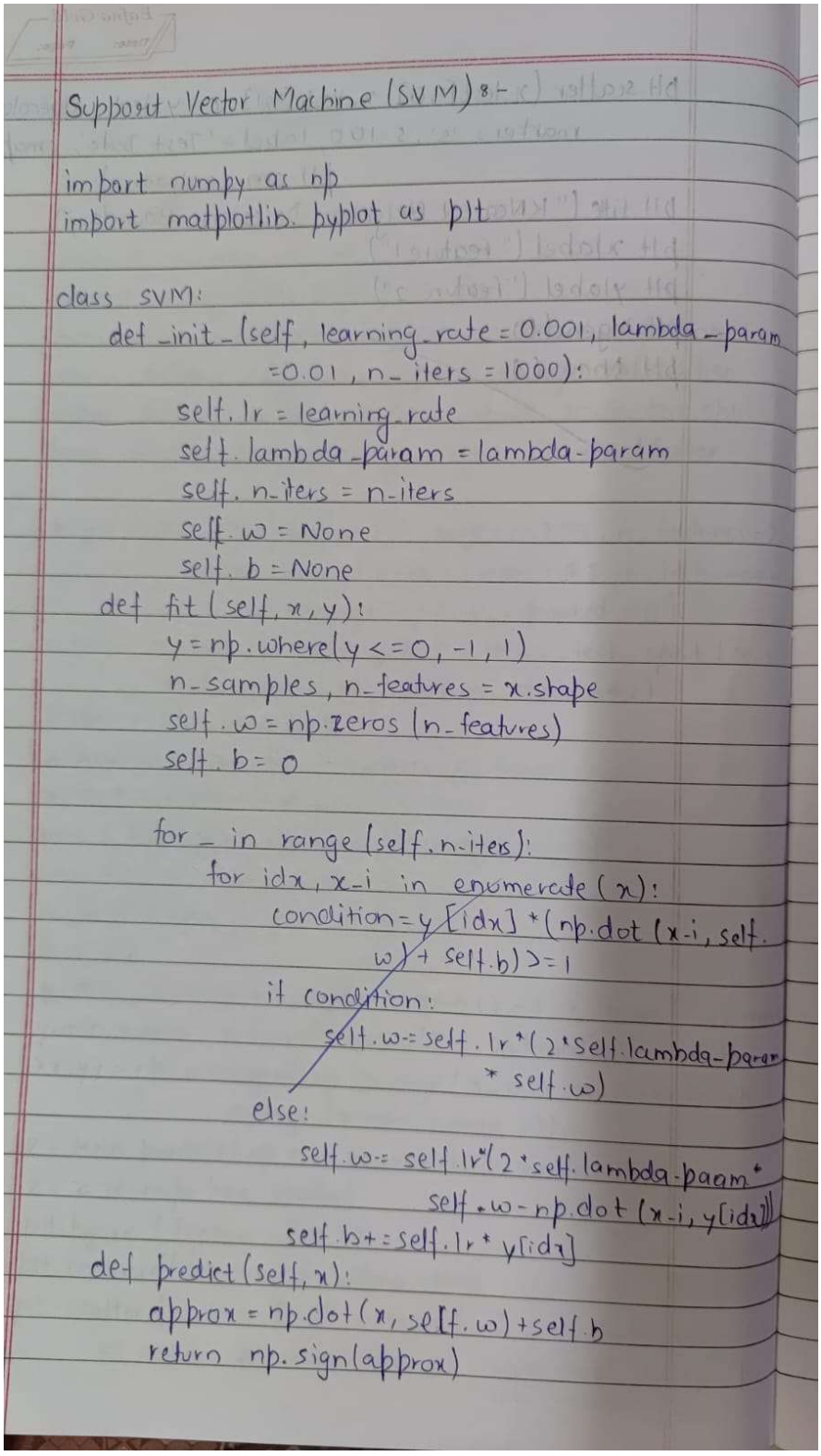
```
# Create a contour plot

plt.figure(figsize=(10, 6))

plt.contourf(xx, yy, Z, alpha=0.8)

plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, edgecolors='k', marker='o', s=100, label='Train Data', cmap=plt.cm.coolwarm)

plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, edgecolors='k', marker='s', s=100, label='Test Data', cmap=plt.cm.coolwarm)

plt.title("K-Nearest Neighbors Decision Boundary")

plt.xlabel("Feature 1")

plt.ylabel("Feature 2")

plt.legend()

plt.show()
```



Program 7

Build Support vector machine model for a given dataset

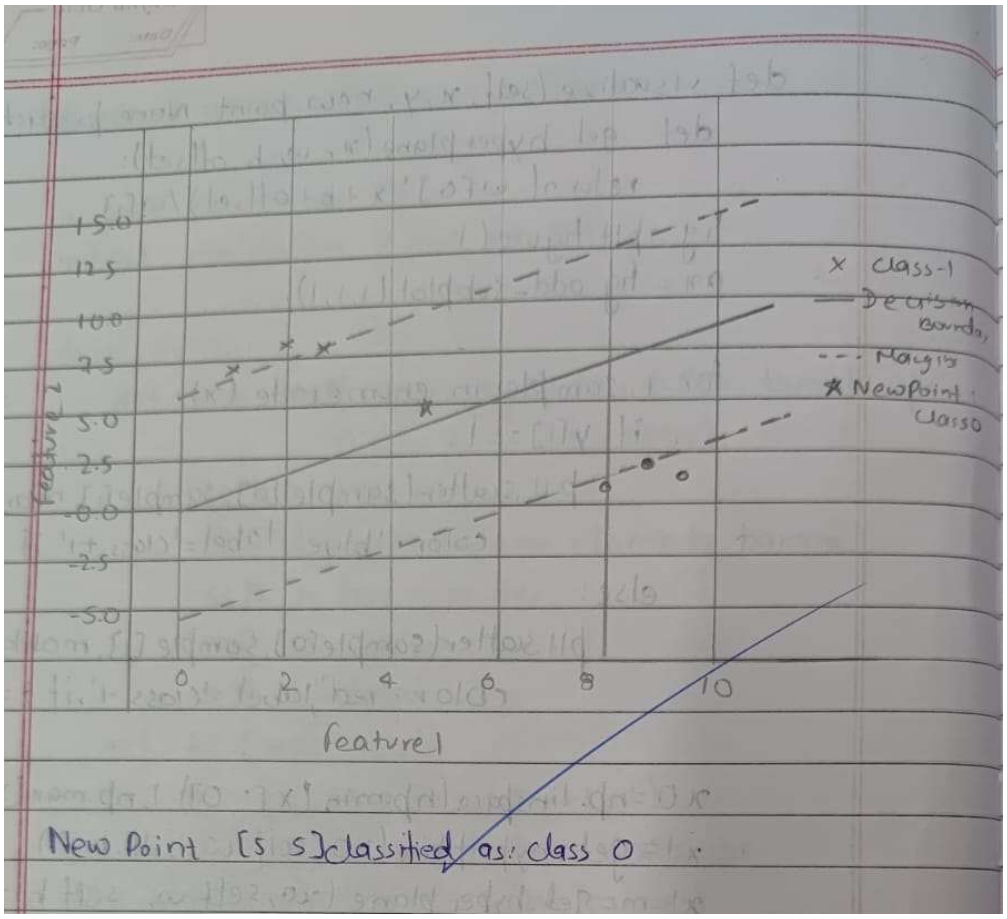
Screenshot



Support Vector Machine (SVM) :-

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

class SVM:
    def __init__(self, learning_rate=0.001, lambda_param
                 =0.01, n_iters=1000):
        self.lr = learning_rate
        self.lambda_param = lambda_param
        self.n_iters = n_iters
        self.w = None
        self.b = None
    def fit(self, x, y):
        y = np.where(y<=0, -1, 1)
        n_samples, n_features = x.shape
        self.w = np.zeros(n_features)
        self.b = 0

        for _ in range(self.n_iters):
            for idx, x_i in enumerate(x):
                condition = y[idx] * (np.dot(x_i, self.
                            w) + self.b) >= 1
                if condition:
                    self.w -= self.lr * (2 * self.lambda_param
                              * self.w)
                else:
                    self.w -= self.lr * (2 * self.lambda_param *
                              self.w - np.dot(x_i, y[idx]))
                    self.b += self.lr * y[idx]
    def predict(self, x):
        approx = np.dot(x, self.w) + self.b
        return np.sign(approx)
```

```
def visualize(self, x, y, new_point=None, prediction=None):
    def get_hyperplane(x, w, b, offset):
        return(-w[0]* x + b + offset)/w[1]
    fig = plt.figure()
    ax = fig.add_subplot(1,1,1)

    for i, sample in enumerate(x):
        if y[i]==1:
            plt.scatter(sample[0], sample[1], marker='o',
                color='blue', label='class +1' if i==0 else"")
        else:
            plt.scatter(sample[0], sample[1], marker='x',
                color='red', label='class -1', if i==0 else"")

    x0 = np.linspace(np.min(X[:,0])-1, np.max(X[:,0])+1,
    x1 = get_hyperplane(x0, self.w, self.b, 0)
    x1_m = get_hyperplane(x0, self.w, self.b, -1)
    x1_p = get_hyperplane(x0, self.w, self.b, 1)
    ax.plot(x0, x1, 'k-', label='Decision Boundary')
    ax.plot(x0, x1_m, 'k--', label='Margins')
    ax.plot(x0, x1_p, 'k--')

    if new_point is not None:
        color='green' if prediction==1 else 'orange'
        label=f'New Point: Class {"1" if prediction==1 else
                                  "0"}'

        plt.scatter(new_point[0], new_point[1], c=color,
            s=100, edgecolors='black', label=label,
                                  marker='*')
    ax.legend()
    plt.xlabel("Feature1")
    plt.ylabel("Feature2")
    plt.title("SVM with new Point Prediction")
    plt.grid(True)
    plt.show()
```



Code:

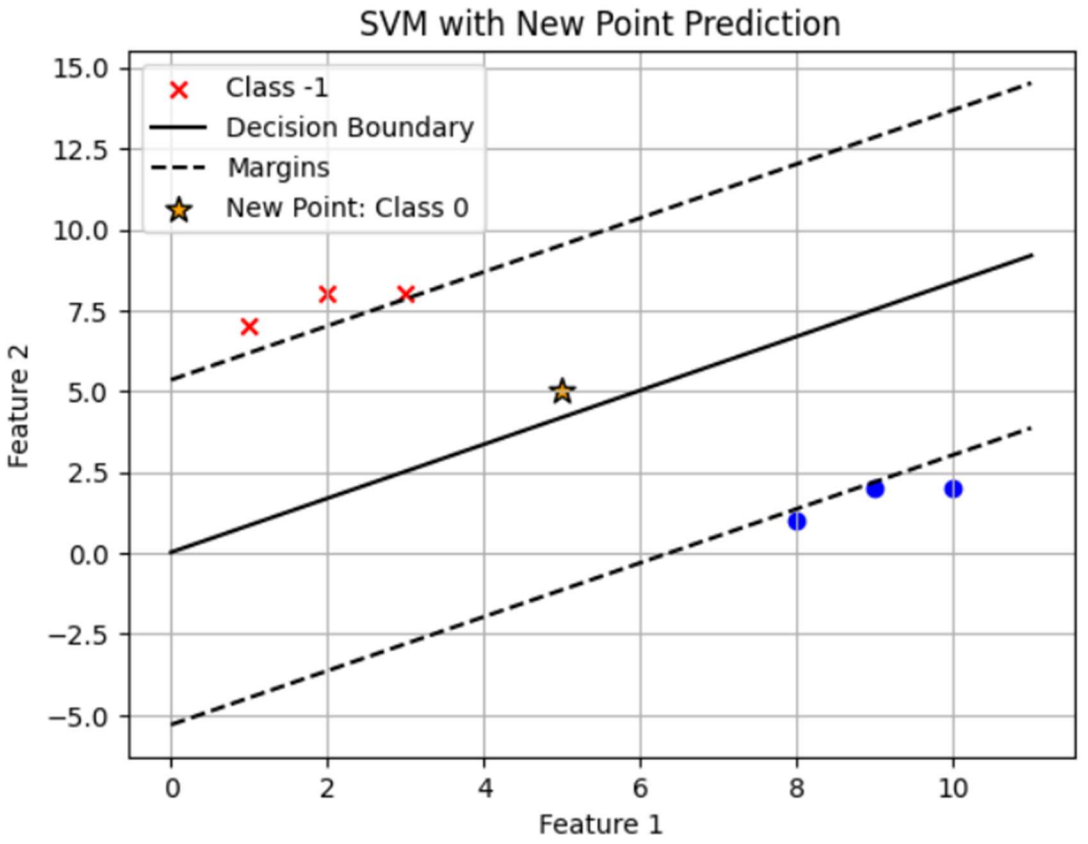
```
import numpy as np

import matplotlib.pyplot as plt


class SVM:

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

        self.lr = learning_rate

        self.lambda_param = lambda_param

        self.n_iters = n_iters

        self.w = None

        self.b = None
```

```
def fit(self, X, y):
    y = np.where(y <= 0, -1, 1)  # Convert labels to -1 and 1
    n_samples, n_features = X.shape
    self.w = np.zeros(n_features)
    self.b = 0

    for _ in range(self.n_iters):
        for idx, x_i in enumerate(X):
            condition = y[idx] * (np.dot(x_i, self.w) + self.b) >= 1
            if condition:
                self.w -= self.lr * (2 * self.lambda_param * self.w)
            else:
                self.w -= self.lr * (2 * self.lambda_param * self.w - np.dot(x_i, y[idx]))
                self.b += self.lr * y[idx]

def predict(self, X):
    approx = np.dot(X, self.w) + self.b
    return np.sign(approx)

def visualize(self, X, y, new_point=None, prediction=None):
    def get_hyperplane(x, w, b, offset):
        return (-w[0] * x + b + offset) / w[1]
```

```
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)

# Plot existing data points
for i, sample in enumerate(X):
    if y[i] == 1:
        plt.scatter(sample[0], sample[1], marker='o', color='blue', label='Class +1' if i == 0 else "")
    else:
        plt.scatter(sample[0], sample[1], marker='x', color='red', label='Class -1' if i == 0 else "")

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

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

# Plot the new point
if new_point is not None:
    color = 'green' if prediction == 1 else 'orange'
    label = f'New Point: Class {"1" if prediction == 1 else "0"}'
```

```
        plt.scatter(new_point[0], new_point[1], c=color, s=100, edgecolors='black', label=label, marker='*')


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


# 🚀 Example usage
if __name__ == "__main__":
    # Training data
    X = np.array([
        [1, 7],
        [2, 8],
        [3, 8],
        [8, 1],
        [9, 2],
        [10, 2]
    ])
    y = np.array([0, 0, 0, 1, 1, 1])  # 0 -> -1, 1 -> +1

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

```
# Train and predict
svm = SVM()
svm.fit(X, y)
prediction = svm.predict(new_point)[0]

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

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

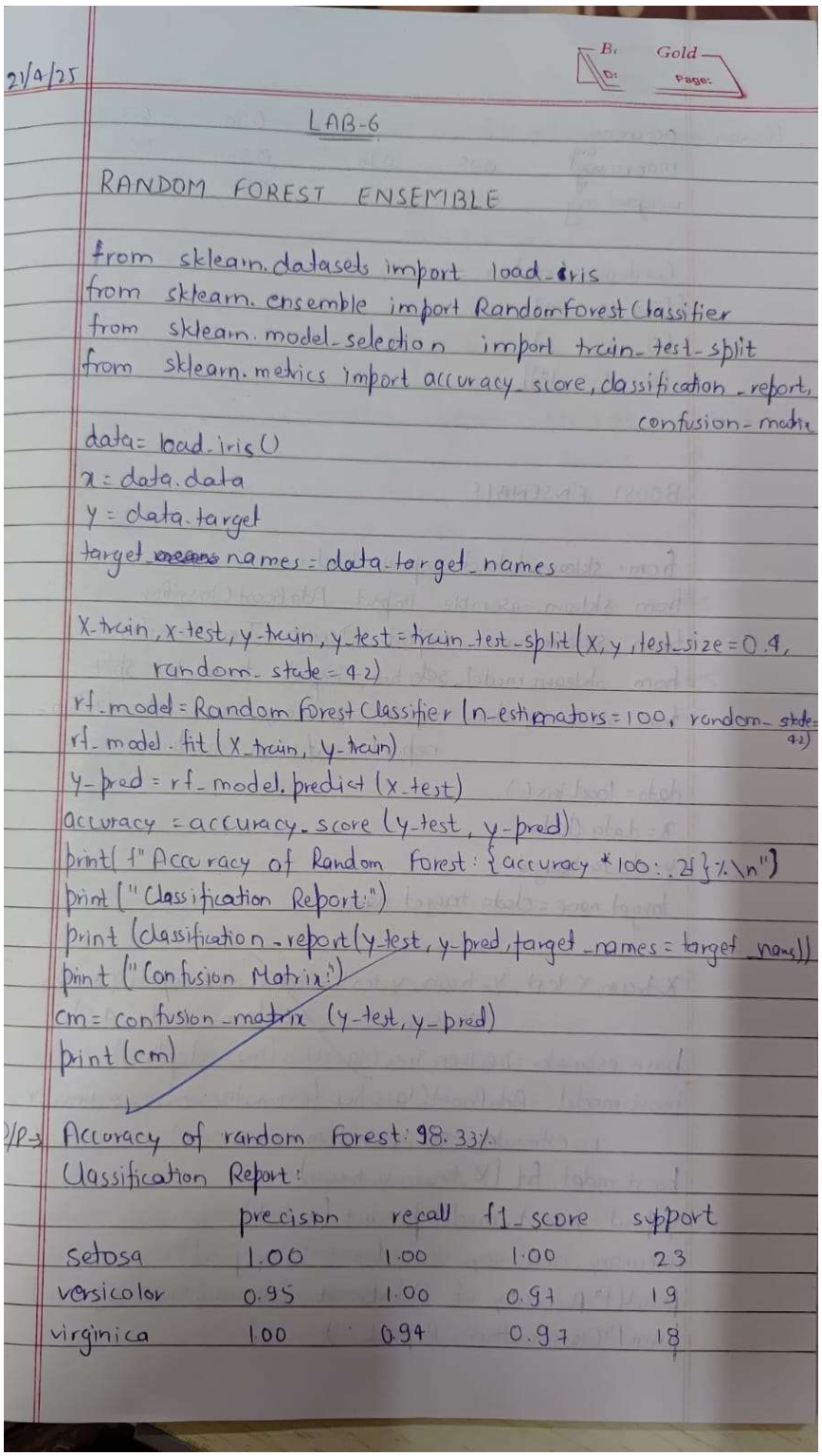



```
New point [5 5] classified as: Class 0
```

Program 8

Implement Random forest ensemble method on a given dataset.

Screenshot



21/4/25

LAB-6

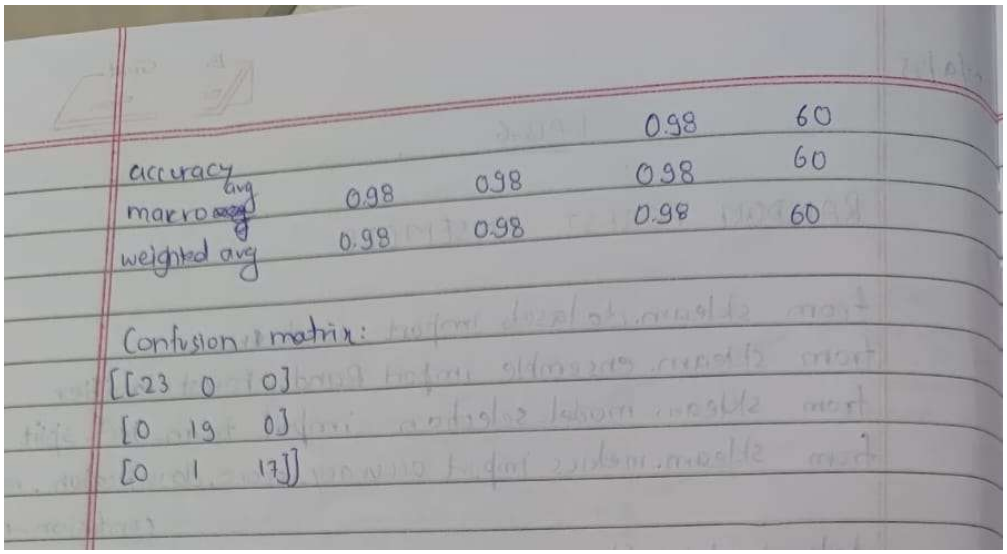
RANDOM FOREST ENSEMBLE

```
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report,
                                                    confusion_matrix
data= load_iris()
x = data.data
y = data.target
target_names = data.target_names

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,
        random_state=42)
rf_model = RandomForestClassifier(n_estimators=100, random_state=
rf_model.fit(X_train, y_train)                                    42)
y_pred = rf_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of Random Forest: {accuracy*100:.2f}%\n")
print("Classification Report:")
print(classification_report(y_test, y_pred, target_names=target_names))
print("Confusion Matrix:")
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

O/P→ Accuracy of random forest: 98.33%

Classification Report:

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	23
versicolor	0.95	1.00	0.97	19
virginica	1.00	0.94	0.97	18



Code:

```
# Import necessary libraries

from sklearn.datasets import load_iris

from sklearn.ensemble import RandomForestClassifier

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix


# Load the dataset

data = load_iris()

X = data.data

y = data.target

target_names = data.target_names


# Split the dataset

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=42)
```

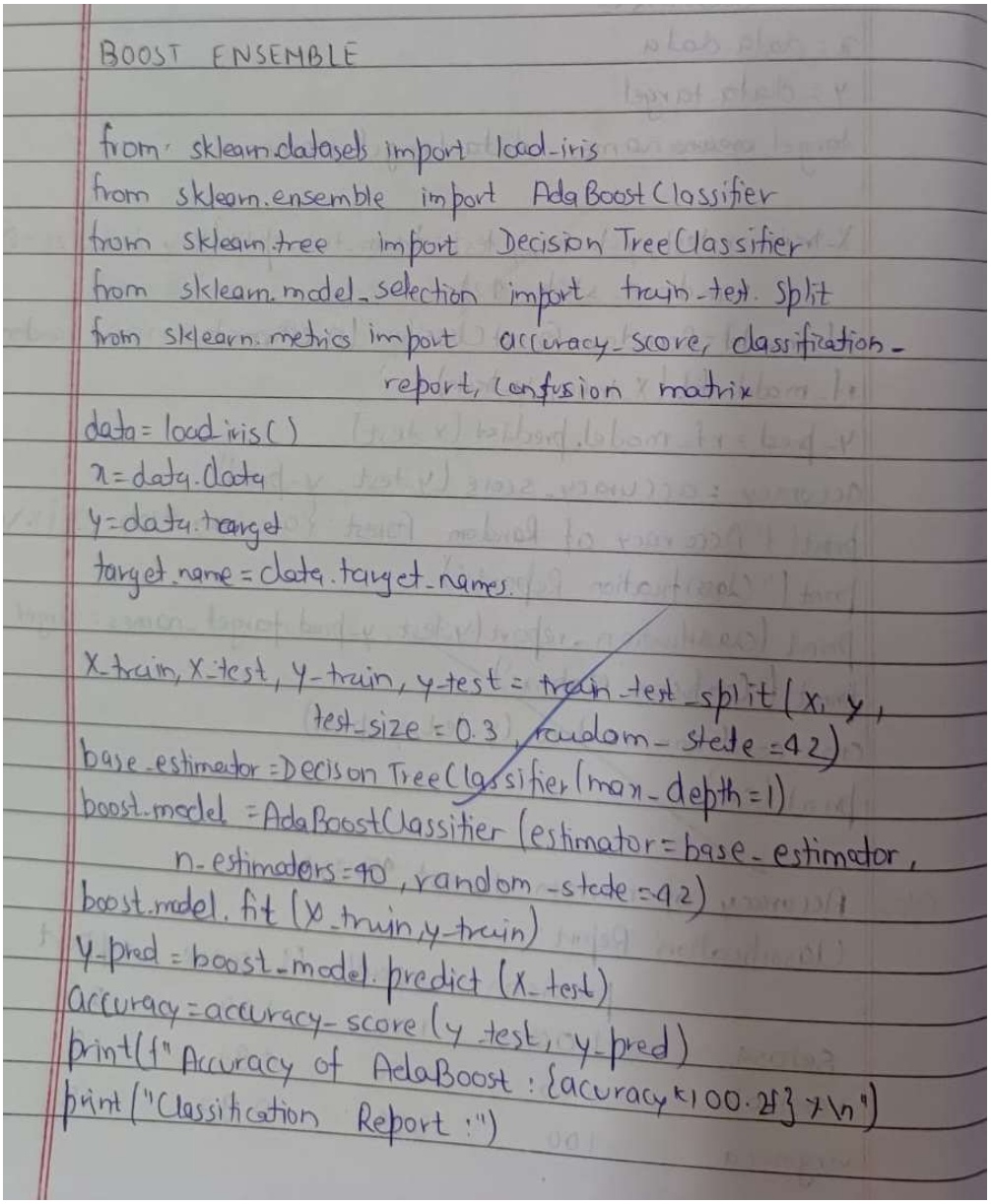
```
# Initialize and train the Random Forest model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Make predictions
y_pred = rf_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of Random Forest: {accuracy * 100:.2f}%\n")

print("Classification Report:")
print(classification_report(y_test, y_pred, target_names=target_names))

print("Confusion Matrix:")
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
Accuracy of Random Forest: 98.33%

Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        23
  versicolor       0.95      1.00      0.97        19
   virginica       1.00      0.94      0.97        18

    accuracy                           0.98        60
   macro avg       0.98      0.98      0.98        60
weighted avg       0.98      0.98      0.98        60

Confusion Matrix:
[[23  0  0]
 [ 0 19  0]
 [ 0  1 17]]
```

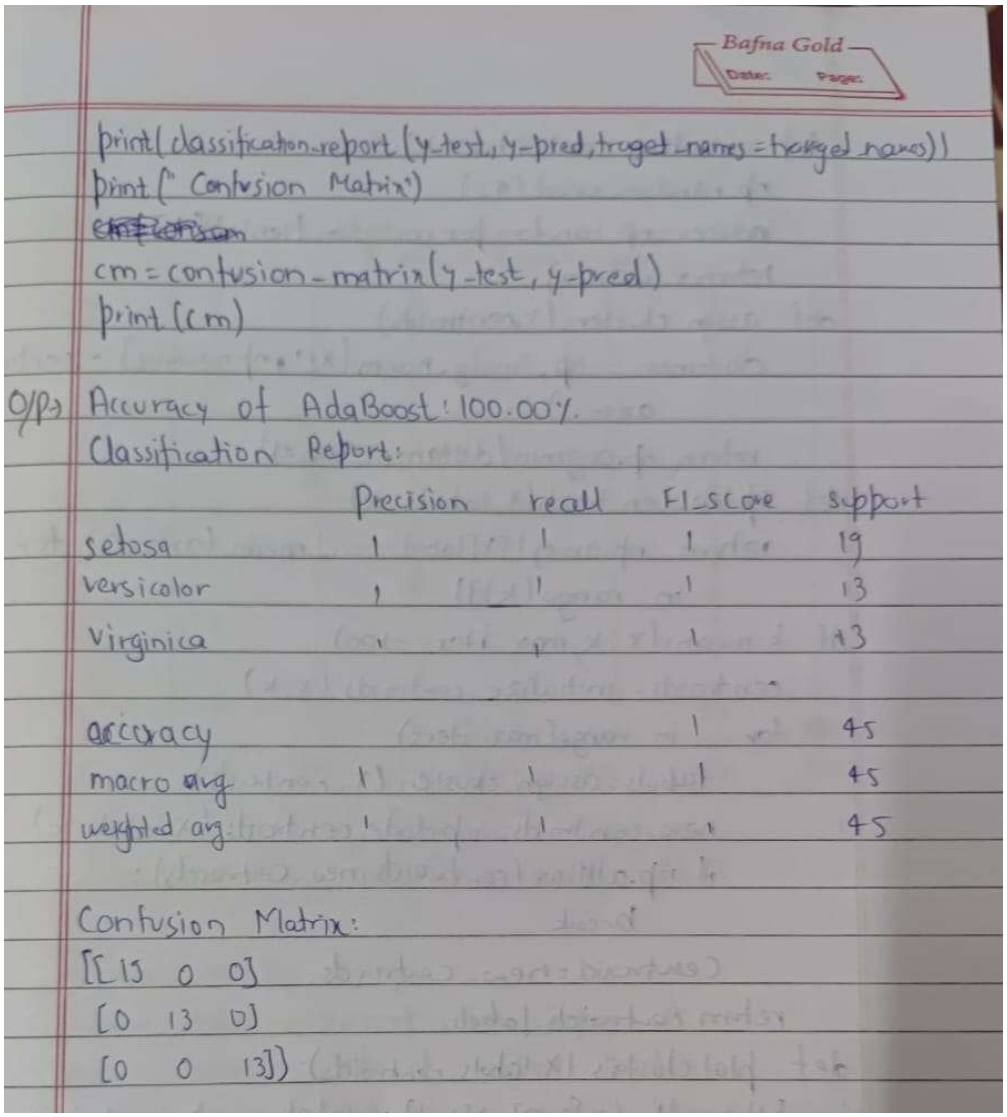
Program 9

Implement Boosting ensemble method on a given dataset.

Screenshot



```
BOOST ENSEMBLE

from sklearn.datasets import load_iris
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_
                            report, confusion_matrix
data = load_iris()
x = data.data
y = data.target
target_name = data.target_names

X_train, X_test, y_train, y_test = train_test_split(x, y,
                    test_size = 0.3, random_state = 42)
base_estimator = DecisionTreeClassifier(max_depth = 1)
boost_model = AdaBoostClassifier(estimator = base_estimator,
          n_estimators = 40, random_state = 42)
boost_model.fit(X_train, y_train)
y_pred = boost_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of AdaBoost : {accuracy * 100:.2f} %\n")
print("Classification Report :")
```



Code:

```
# Import necessary libraries
from sklearn.datasets import load_iris
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix


# Load the dataset
```

```
data = load_iris()

X = data.data

y = data.target

target_names = data.target_names


# Split the dataset

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)


# Initialize AdaBoost with DecisionTree as base estimator

base_estimator = DecisionTreeClassifier(max_depth=1)  # Use stumps (shallow trees)

# Changed 'base_estimator' to 'estimator'

boost_model = AdaBoostClassifier(estimator=base_estimator, n_estimators=40, random_state=42)


# Train the boosting model

boost_model.fit(X_train, y_train)


# Make predictions

y_pred = boost_model.predict(X_test)


# Evaluate the model

accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy of AdaBoost: {accuracy * 100:.2f}%\n")


print("Classification Report:")
```

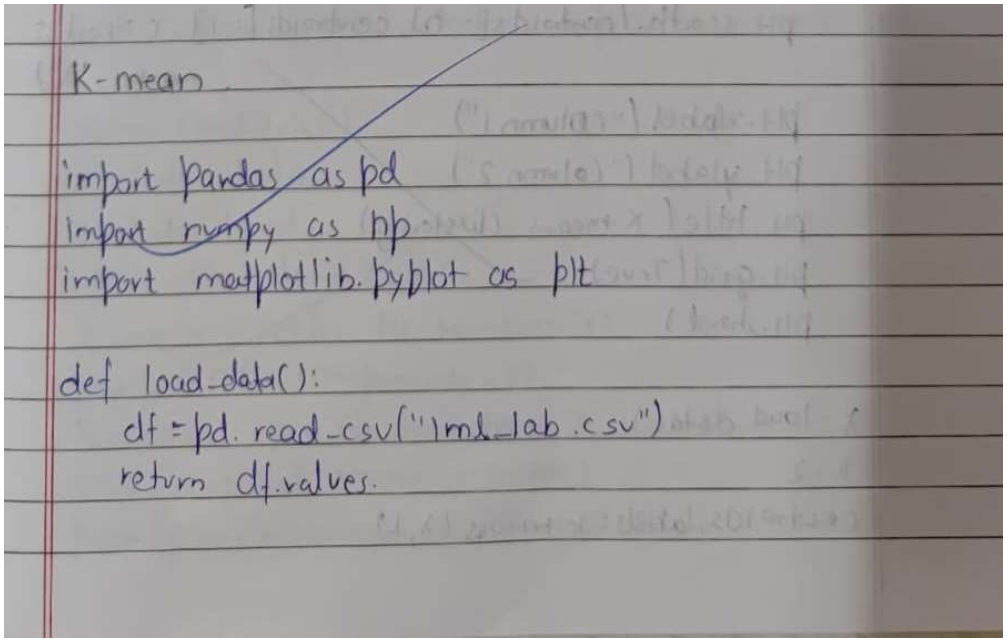
```
print(classification_report(y_test, y_pred, target_names=target_names))


print("Confusion Matrix:")
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
Accuracy of AdaBoost: 100.00%

Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        19
  versicolor       1.00      1.00      1.00        13
   virginica       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45

Confusion Matrix:
[[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
```

Program 10

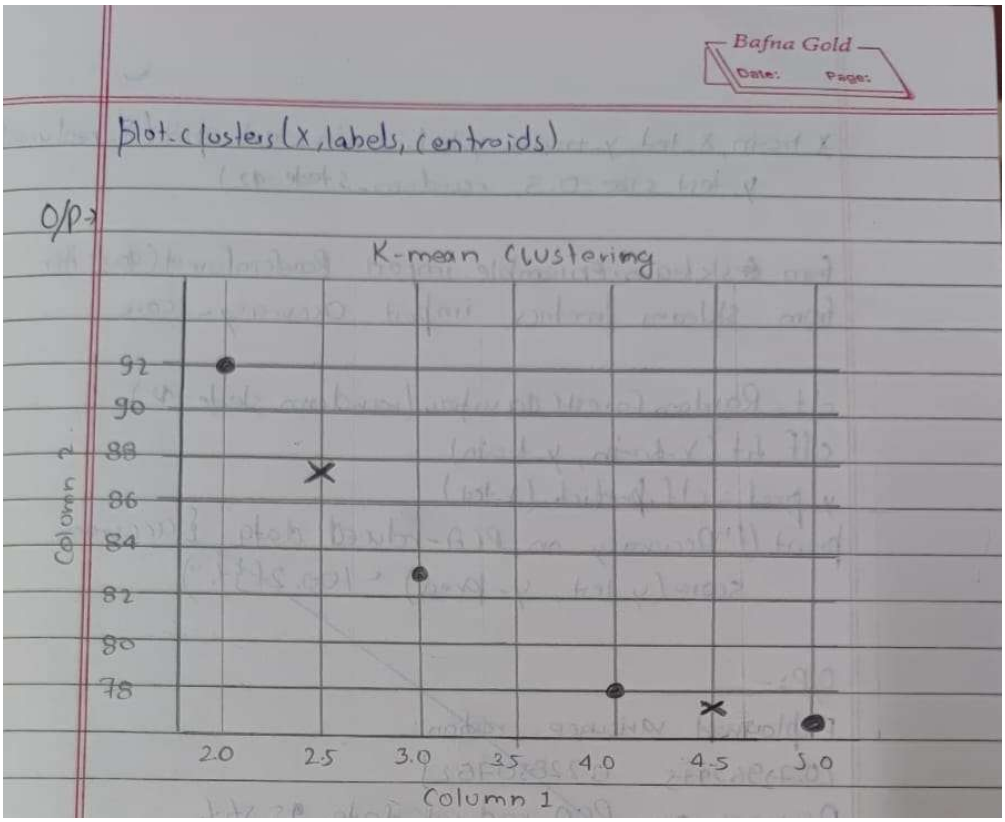
Build k-Means algorithm to cluster a set of data stored in a .CSV file.

Screenshot



```
K-mean

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

def load_data():
    df = pd.read_csv("ml_lab.csv")
    return df.values
```

```
def initialize_centroids(X,k):
    np.random.seed(42)
    indices=np.random.permutation(len(x))[:k]
    return x[indices]
def assign_clusters(X,centroids):
    distances = np.linalg.norm(X[:,np.newaxis] - centroids,
                axis=2)
    return np.argmin(distances,axis=1)
def update_centroids(X,labels,k):
    return np.array([X[labels==i].mean(axis=0) for i
                in range(k)])
df k-means(X,k,max_iters=100):
    centroids= initialize_centroids(X,k)
    for _ in range(max_iters):
        labels=assign_clusters(X,centroids)
        new_centroids=update_centroids(X,labels,k)
        if np.allclose(centroids,new_centroids):
            break
        Centroid=new_centroids.
    return centroids,labels.
def plot_clusters(X,labels,centroids):
    plt.scatter(X[:,0],X[:,1],c=labels,cmap='viridis',s=100)
    plt.scatter(centroids[:,0],centroids[:,1],c='red',s=200,
                                        marker='X')
    plt.xlabel("Column 1")
    plt.ylabel("Column 2")
    plt.title("K-means Clustering")
    plt.grid(True)
    plt.show()

X=load_data()
K=2
centroids,labels=k-means(X,k)
```

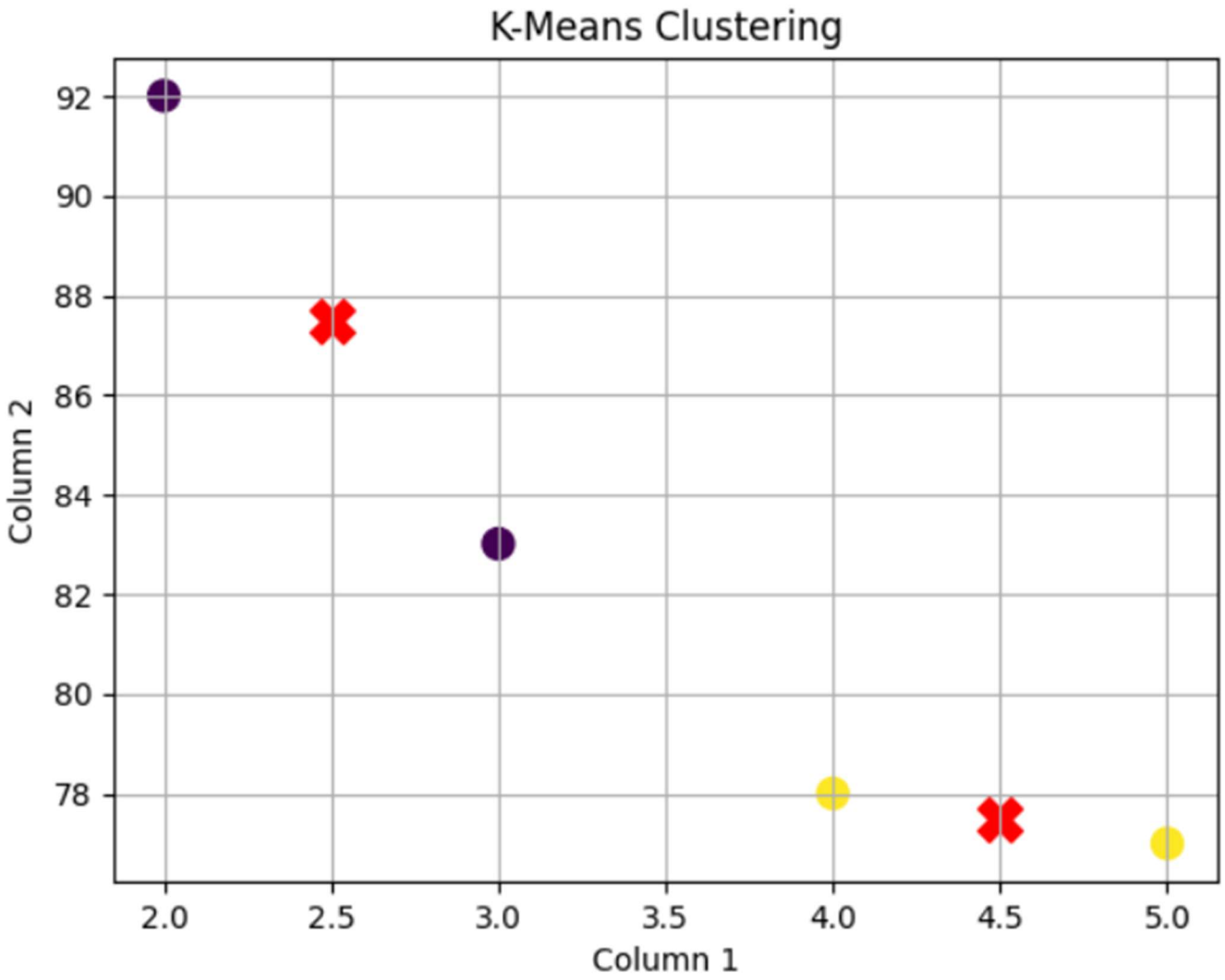


Code:

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt


def load_data():

    df = pd.read_csv("/ml_lab.csv")

    return df.values


def initialize_centroids(X, k):

    np.random.seed(42)

    indices = np.random.permutation(len(X))[:k]

    return X[indices]
```

```
def assign_clusters(X, centroids):
    distances = np.linalg.norm(X[:, np.newaxis] - centroids, axis=2)
    return np.argmin(distances, axis=1)

def update_centroids(X, labels, k):
    return np.array([X[labels == i].mean(axis=0) for i in range(k)])

def k_means(X, k, max_iters=100):
    centroids = initialize_centroids(X, k)
    for _ in range(max_iters):
        labels = assign_clusters(X, centroids)
        new_centroids = update_centroids(X, labels, k)
        if np.allclose(centroids, new_centroids):
            break
        centroids = new_centroids
    return centroids, labels

def plot_clusters(X, labels, centroids):
    plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', s=100)
    plt.scatter(centroids[:, 0], centroids[:, 1], c='red', s=200, marker='X')
    plt.xlabel("Column 1")
    plt.ylabel("Column 2")
    plt.title("K-Means Clustering")
```

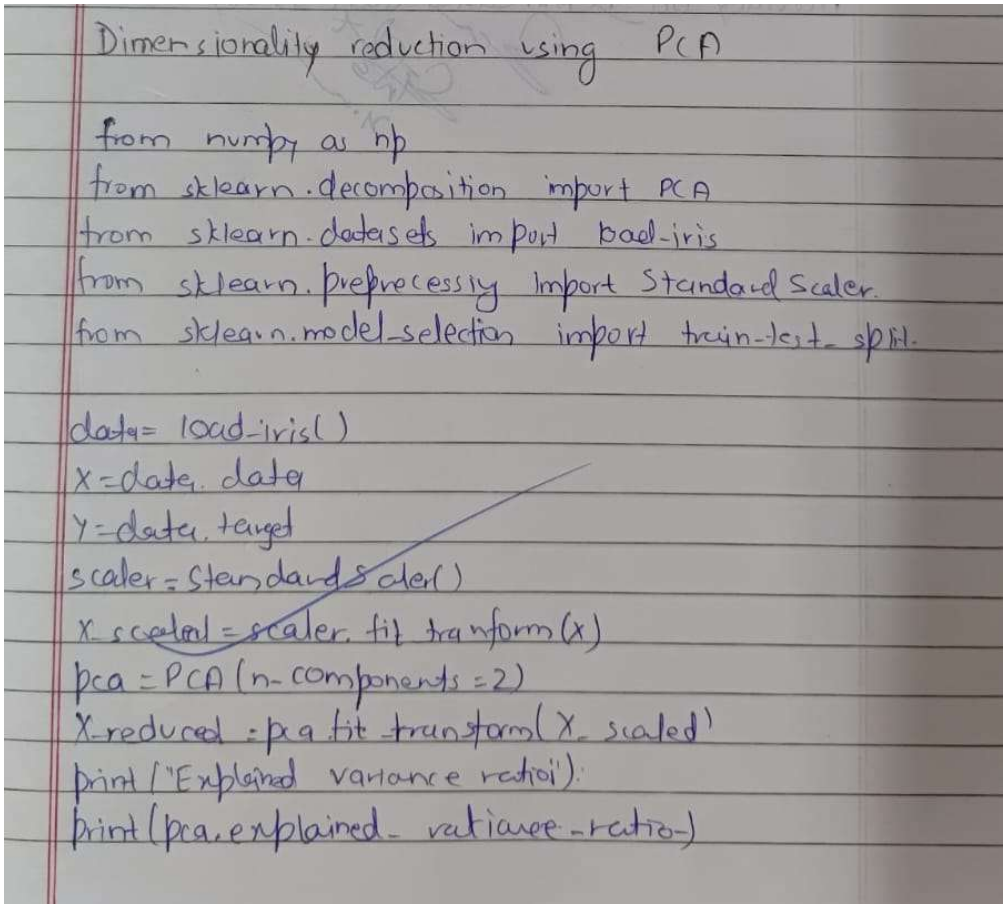
```
    plt.grid(True)
    plt.show()


# Run the clustering
X = load_data()
k = 2  # You can change this to try other numbers
centroids, labels = k_means(X, k)
plot_clusters(X, labels, centroids)
```



Program 11

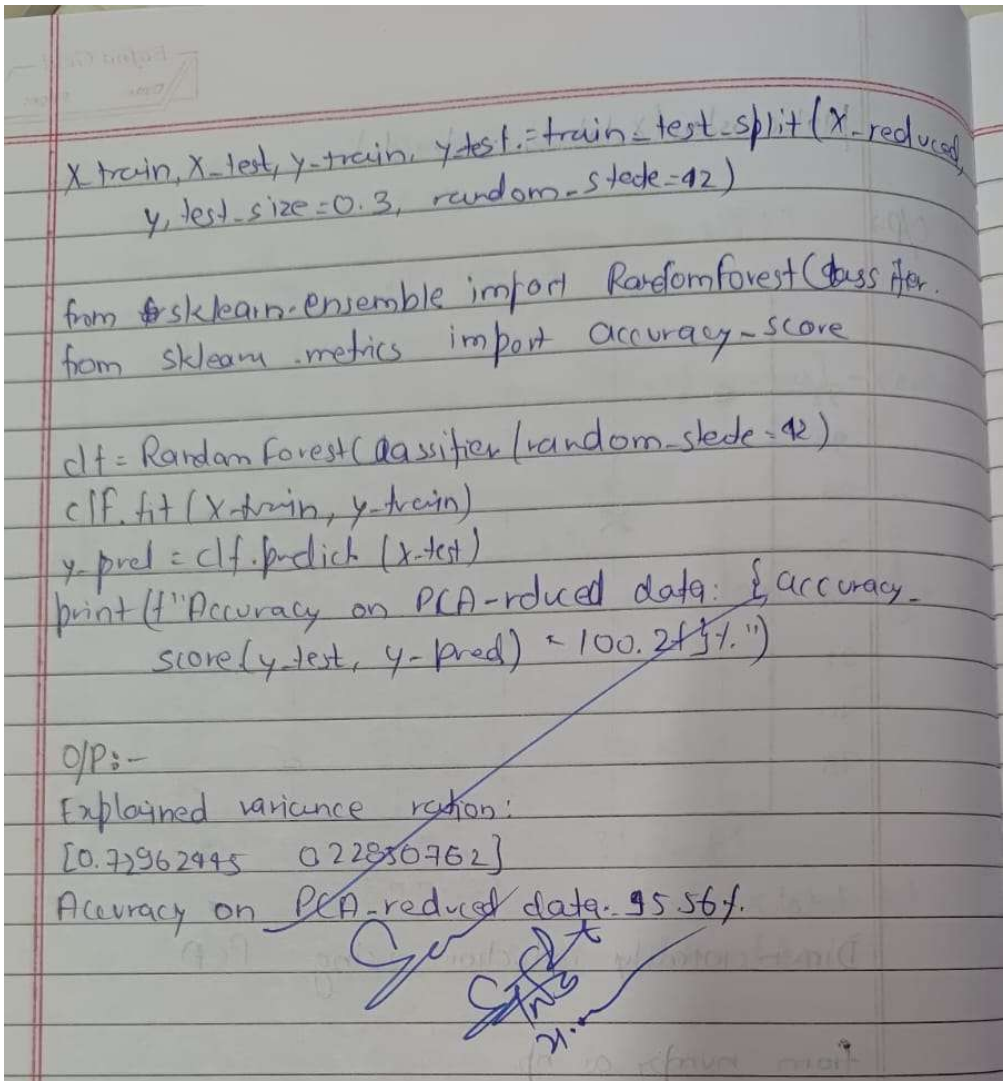
Implement Dimensionality reduction using Principle Component Analysis (PCA) method.

Screenshot



Dimensionality reduction using PCA

```
from numpy as np
from sklearn.decomposition import PCA
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

data= load_iris()
X=data.data
Y=data.target
scaler=StandardScaler()
X_scaled=scaler.fit_transform(X)
pca=PCA(n_components=2)
X_reduced=pca.fit_transform(X_scaled)
print("Explained variance ratio:")
print(pca.explained_variance_ratio_)
```



Code:

#PCA

Import required libraries

import numpy as np

from sklearn.decomposition import PCA

from sklearn.datasets import load_iris

from sklearn.preprocessing import StandardScaler

```
from sklearn.model_selection import train_test_split

# Load the dataset
data = load_iris()
X = data.data
y = data.target

# Step 1: Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 2: Apply PCA to reduce to 2 dimensions
pca = PCA(n_components=2)
X_reduced = pca.fit_transform(X_scaled)

# Print the explained variance ratio
print("Explained variance ratio of each principal component:")
print(pca.explained_variance_ratio_)

# Optional: split the reduced data for model training/testing
X_train, X_test, y_train, y_test = train_test_split(X_reduced, y, test_size=0.3, random_state=42)

# Example: Train a classifier on reduced data (Random Forest here)
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.metrics import accuracy_score


clf = RandomForestClassifier(random_state=42)

clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)


print(f"Accuracy on PCA-reduced data: {accuracy_score(y_test, y_pred) * 100:.2f}%")
```

```
Explained variance ratio of each principal component:
[0.72962445 0.22850762]
Accuracy on PCA-reduced data: 95.56%
```