VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

Prakruthi B S (1BM23CS414)

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 **Prakruthi B S (1BM23CS414),** 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/prakruthi23/ML.git

Program 1

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

Screenshot:

DATE: PAGE:

07/02/2025

Lab 1

Method 1: Initializing values directly into Dataframe.
Insert your know values, five rows of data with column headings as "USN, Name, Marks"

```
imports pandas as pd.
data= { 'USN': ['101', '102', '103', '104'],
'Name': ['Alice', 'Bob', 'Charlie', 'David'],
'Marks': [25, 30, 35, 40] }
df = pd.DataFrame(data)
print("Sample data:")
print(df.head()).
```

Method 2: Importing datasets from sklearn.datasets.
Loading diabetes datasets sklearn.datasets.load_diabetes.

```
from sklearn.datasets import load_diabetes.
diabetes =load_diabetes().
df = pd.Dataframe(diabetes.data, columns=diabetes.
        feature_names).
df ['target'] = diabetes.target
print("Sample data:")
print(df.head()).
```

Method 3: Importing datasets from a specific CSV file
sample_sales_data.csv.

```
file_path= 'sample_sales_data.csv'
df = pd.read_csv(file_path)
```

```
print("Sample data")
print(df.head())
print("\n")
```

Method4: Downloading datasets from existing dataset repositories like Kaggle, UCI, Mendely, KEEL, etc.

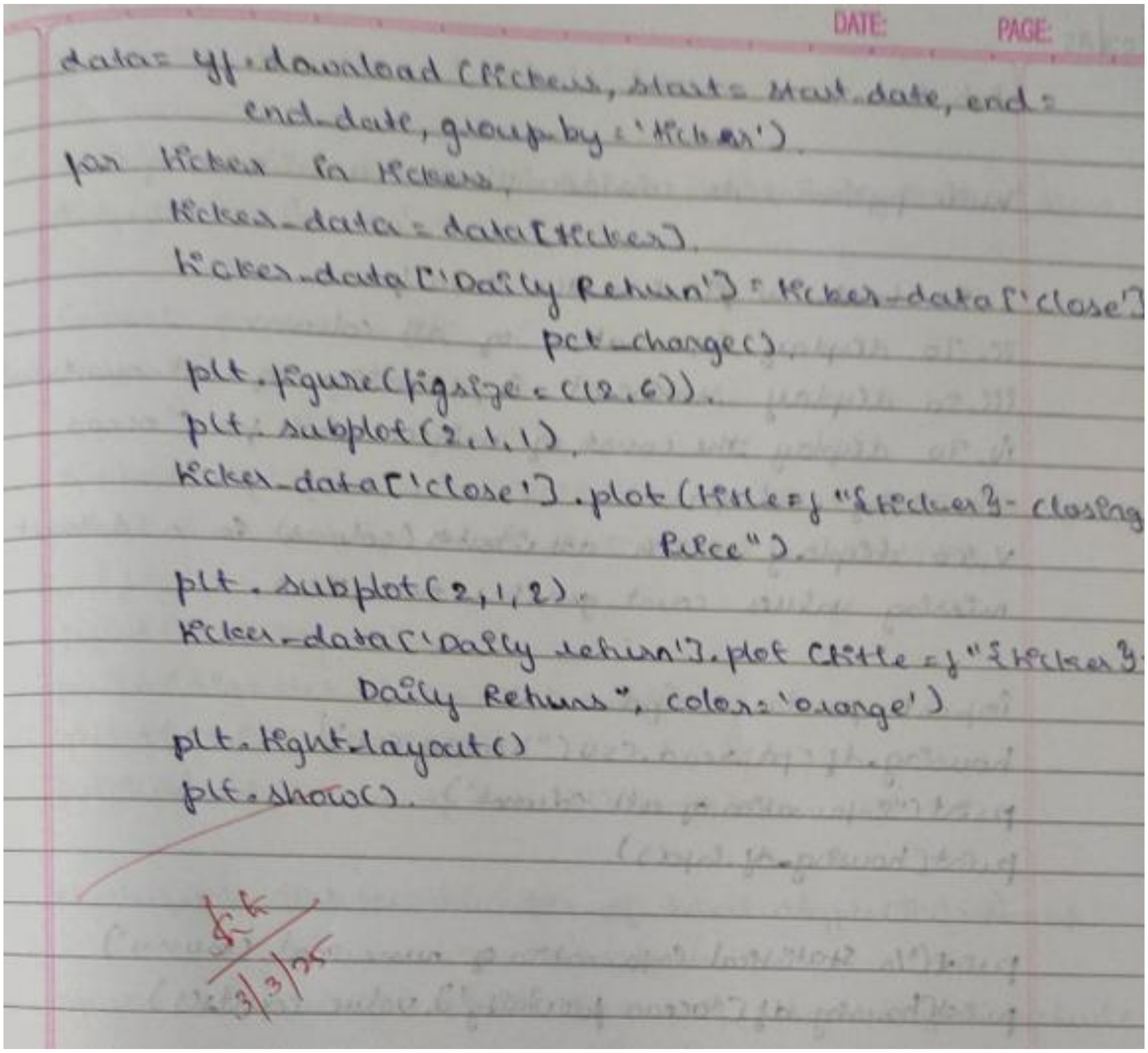
```
df = pd.read_csv('/content/Dataset of Diabetes.csv',
                 encoding='latin-1').
print("Sample data")
print(df.head()).
```

<u>To Do 2</u>

Using the code given in the above slides, do the exercise of the "Stock Market Data Analysis", considering the following.

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

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

tickers = ['HDFCBANK.NS", "ICICIBANK.NS", "KOTAKBANK.NS"]
start_date = "2024-01-01"
end_date = "2024-12-30"
```



```
data= yf.download(tickers, start= start_date, end=
        end_date, group_by='ticker')
for ticker in tickers.
    ticker_data = data[ticker].
    ticker_data['Daily Return'] = ticker_data['close']
                    pct_change().
    plt.figure(figsize=(12,6)).
    plt.subplot(2,1,1).
    ticker_data['close'].plot(title=f"{ticker}- closing
                    Price").
    plt.subplot(2,1,2).
    ticker_data['Daily return'].plot(title=f"{ticker}-
            Daily Returns", color='orange')
    plt.tight_layout()
    plt.show().
```

3/3/25

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' # Ensure the file exists in the same directory
df = pd.read_csv(file_path)
print("Sample data:")
print(df.head())
print("\n")
```

```
df = pd.read_csv('/content/Mobiles Dataset (2025).csv', encoding='latin-1')
print("Sample data:")
print(df.head())
```

```
import pandas as pd
data = {
    'USN': ['101', '102', '103', '104'],
'Name': ['Alice', 'Bob', 'Charlie', 'David'],
'Marks': [25, 30, 35, 40],
}
df = pd.DataFrame(data)
print("Sample data:")
print(df.head())
```

```python
from sklearn.datasets import load_diabetes
diabetes = load_diabetes()
df = pd.DataFrame(diabetes.data, columns=diabetes.feature_names)
df['target'] = diabetes.target
print("Sample data:")
print(df.head())
```

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

```python
df = pd.read_csv('/content/Dataset of Diabetes .csv', encoding='latin-1')
print("Sample data:")
print(df.head())
```

```python
import yfinance as yf
import pandas as pd
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)
reliance_data = data['RELIANCE.NS']
print("\nSummary statistics for Reliance Industries:")
print(reliance_data.describe())
reliance_data['Daily Return'] = reliance_data['Close'].pct_change()
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')
```

```
print("\nReliance stock data saved to 'reliance_stock_data.csv'.")
```

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

    ticker_data['Daily Return'] = ticker_data['Close'].pct_change()
    plt.figure(figsize=(12, 6))
    plt.subplot(2, 1, 1) ticker_data['Close'].plot(title=f"{ticker} - Closing Price") plt.subplot(2,
    1, 2)
    ticker_data['Daily Return'].plot(title=f"{ticker} - Daily Returns", color='orange')
    plt.tight_layout() plt.show()
```

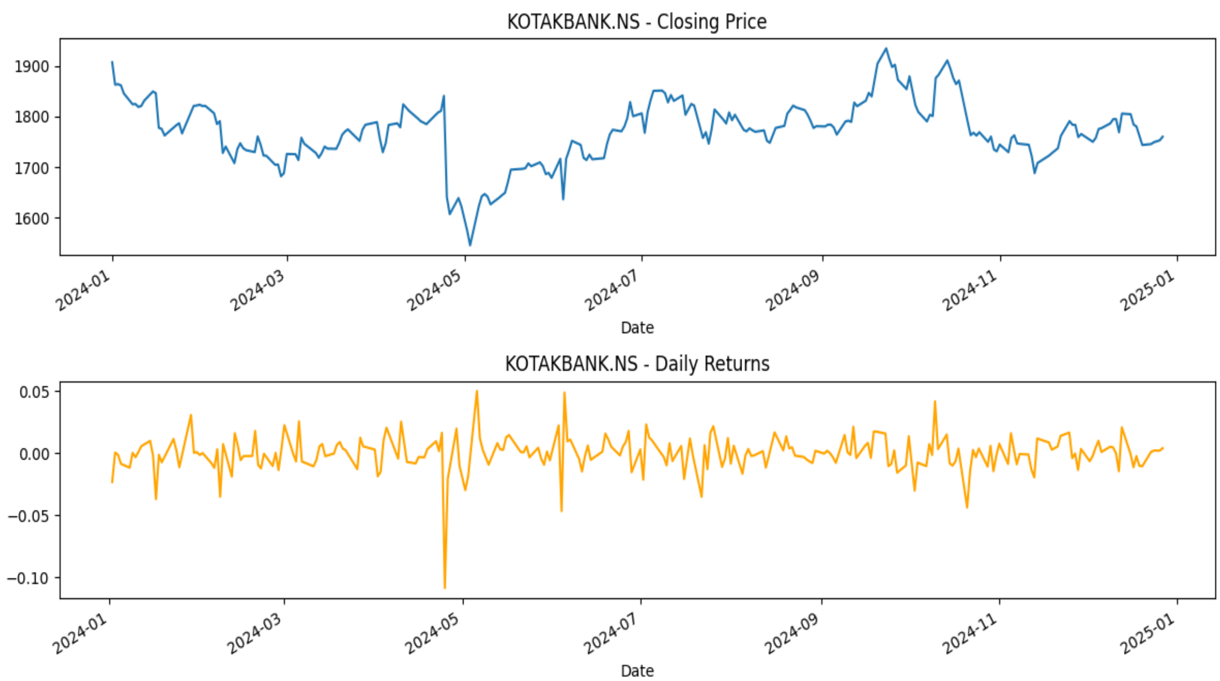
Output:

```
Sample data:
      Name  Age         City
0    Alice   25     New York
1      Bob   30  Los Angeles
2  Charlie   35      Chicago
3    David   40      Houston
```

```
Sample data:
   ID     Name  Age         City
0   1    Alice   25     New York
1   2      Bob   30  Los Angeles
2   3  Charlie   35      Chicago
3   4    David   40      Houston
4   5      Eva   28      Phoenix
```

```
Sample data:
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

   target
0       0
1       0
2       0
3       0
4       0
```

```
Sample data:
  Company Name            Model Name Mobile Weight  RAM Front Camera  \
0        Apple       iPhone 16 128GB          174g  6GB         12MP
1        Apple       iPhone 16 256GB          174g  6GB         12MP
2        Apple       iPhone 16 512GB          174g  6GB         12MP
3        Apple  iPhone 16 Plus 128GB          203g  6GB         12MP
4        Apple  iPhone 16 Plus 256GB          203g  6GB         12MP

  Back Camera   Processor Battery Capacity Screen Size  \
0        48MP  A17 Bionic         3,600mAh  6.1 inches
1        48MP  A17 Bionic         3,600mAh  6.1 inches
2        48MP  A17 Bionic         3,600mAh  6.1 inches
3        48MP  A17 Bionic         4,200mAh  6.7 inches
4        48MP  A17 Bionic         4,200mAh  6.7 inches

  Launched Price (Pakistan) Launched Price (India) Launched Price (China)  \
0               PKR 224,999             INR 79,999              CNY 5,799
1               PKR 234,999             INR 84,999              CNY 6,099
2               PKR 244,999             INR 89,999              CNY 6,499
3               PKR 249,999             INR 89,999              CNY 6,199
4               PKR 259,999             INR 94,999              CNY 6,499

  Launched Price (USA) Launched Price (Dubai)  Launched Year
0              USD 799              AED 2,799           2024
1              USD 849              AED 2,999           2024
2              USD 899              AED 3,199           2024
3              USD 899              AED 3,199           2024
4              USD 949              AED 3,399           2024
```



Program 2

Demonstrate various data pre-processing techniques for a given dataset

Screenshot:

10/02/25

Lab-2

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

i. To load.csv file into the data frame.
ii. To display information of all columns
iii. To display statistical information of all numerical
iv. To display the count of unique labels for "ocean Proximity" column.
v. To display which attributes (colums) in a dataset ha missing values count greater than zero.

```
import pandas as pd.
housing.df=pd.read.csv("housing.csv").
print("information of all colums").
print(housing-df.info()).

print("\n statistical information of numerical colums")
print(housing.df['ocean proximity'].value_counts())

print("/ count of unique labels for "ocean Proximity")
print(housing-df.describe())

print("\n column with missing values")
missing-values=housing-df.isnull().sum()
print(missing_values[missing-values]>0).
```

For both diabetes and adult income.

1) which columns in the dataset had missing values? How did you handle them?

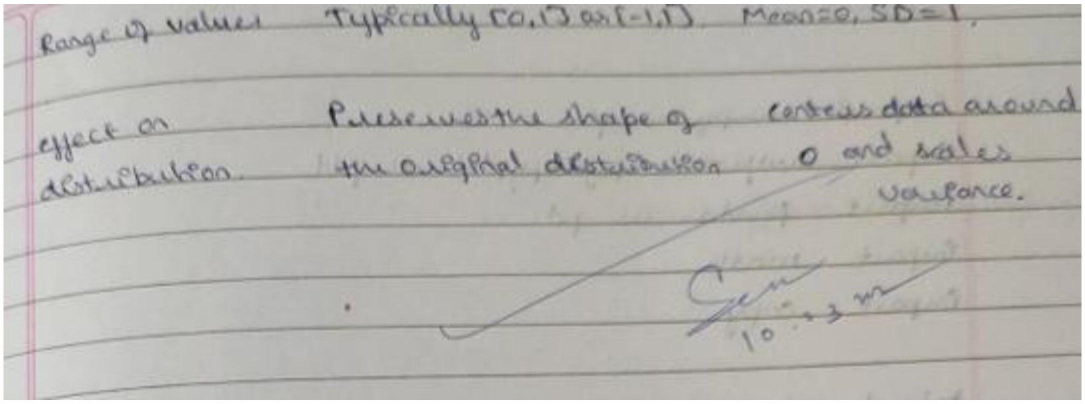
```
import pandas as pd.
import numpy as np

diabetes_df = pd.read_csv("/content/Dataset of Diabetes(1).csv")
adult_income_df = pd.read_csv("/content/adult.csv").

print("Missing values")
print(diabetes_df.isnull().sum())
print("Missing values")
print(adult_income_df.isnull().sum())

number_cols_diabetes = diabetes_df.select_dtypes(include=np.
                    number).columns
number_cols_adult = adult_income_df.select_dtypes(include=
                    np.number).colums

diabetes_df[numeric_cols_diabetes] = diabetes_df[numeric_
cols_diabetes].fillna(diabetes_df[numeric_cols_diabetes].
    median()].
adult_income_df[numeric_cols_adult] = adult_income_df
[numeric_cols_adult].fillna(adult_income_df[numeric_co
    adult].median()]
```

2) which categorical columns did you identify in the dataset? How did you encode them?

```
imports pandas as pd.
from sklearn.preprocessing import LabelEncoder.

diabetes_df = pd.read_csv("/content/dataset of Diabetes()).

print("Diabetes dataset:")
print(diabetes_df.head())

bins=[0,18.5,24.9,40].
labels=['Low','Normal','High']
diabetes_df['BMI_category']= pd.cut (diabetes_df['BMI'],
    bins=bins, labels=labels, right=False).

diabetes_df = pd.get_dummies(diabetes_df, columns=
    ['BMI_category'])

if 'gender' in diabetes_df.columns:
    label_encoder = LabelEncoder()
    diabetes_df['gender']= label_encoder.fit_transform(
diabetes_df['gender'])

print("\nDiabetes dataset after encoding:")
print(diabetes_df.head()).
```

What is the difference between min-max scaling and standardization? When would you use one over the other

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



Range of values	Typically [0,1] or [-1,1]	Mean=0, SD=1.
effect on distribution.	Preserves the shape of the original distribution	centers data around 0 and scales variance.

Code:

```
import pandas as pd
housing_df = pd.read_csv("housing.csv")
print("Information of all columns:")
print(housing_df.info())
print("\nStatistical information of numerical columns:")
print(housing_df.describe())
print("\nCount of unique labels for 'Ocean Proximity' column:")
print("\nColumns with missing values:")
missing_values = housing_df.isnull().sum()
print(missing_values[missing_values > 0])
```

```
import pandas as pd
import numpy as np
diabetes_df = pd.read_csv("/content/Dataset of Diabetes  (1).csv")
adult_income_df = pd.read_csv("/content/adult.csv")
print("Missing values in Diabetes dataset:")
print(diabetes_df.isnull().sum())
print("\nMissing values in Adult Income dataset:")
print(adult_income_df.isnull().sum())
numeric_cols_diabetes = diabetes_df.select_dtypes(include=np.number).columns
numeric_cols_adult = adult_income_df.select_dtypes(include=np.number).columns
diabetes_df[numeric_cols_diabetes] = diabetes_df[numeric_cols_diabetes].fillna(diabetes_df[numeric_cols_diabetes].median())
adult_income_df[numeric_cols_adult] = adult_income_df[numeric_cols_adult].fillna(adult_income_df[numeric_cols_adult].median())
```

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
diabetes_df = pd.read_csv("/content/Dataset of Diabetes  (1).csv")
print("Diabetes dataset (first few rows):")
```

```python
print(diabetes_df.head())
bins = [0, 18.5, 24.9, 40]
labels = ['Low', 'Normal', 'High']
diabetes_df['BMI_Category'] = pd.cut(diabetes_df['BMI'], bins=bins, labels=labels, right=False)
diabetes_df = pd.get_dummies(diabetes_df, columns=['BMI_Category'])
if 'Gender' in diabetes_df.columns:
    label_encoder = LabelEncoder()
    diabetes_df['Gender'] = label_encoder.fit_transform(diabetes_df['Gender'])
print("\nDiabetes dataset after encoding:")
print(diabetes_df.head())
```

Output:

```
Diabetes dataset (first few rows):
    ID  No_Pation Gender  AGE  Urea  Cr  HbA1c  Chol   TG  HDL  LDL  VLDL  \
0  502      17975      F   50   4.7  46    4.9   4.2  0.9  2.4  1.4   0.5
1  735      34221      M   26   4.5  62    4.9   3.7  1.4  1.1  2.1   0.6
2  420      47975      F   50   4.7  46    4.9   4.2  0.9  2.4  1.4   0.5
3  680      87656      F   50   4.7  46    4.9   4.2  0.9  2.4  1.4   0.5
4  504      34223      M   33   7.1  46    4.9   4.9  1.0  0.8  2.0   0.4

    BMI CLASS
0  24.0     N
1  23.0     N
2  24.0     N
3  24.0     N
4  21.0     N

Diabetes dataset after encoding:
    ID  No_Pation  Gender  AGE  Urea  Cr  HbA1c  Chol   TG  HDL  LDL  VLDL  \
0  502      17975       0   50   4.7  46    4.9   4.2  0.9  2.4  1.4   0.5
1  735      34221       1   26   4.5  62    4.9   3.7  1.4  1.1  2.1   0.6
2  420      47975       0   50   4.7  46    4.9   4.2  0.9  2.4  1.4   0.5
3  680      87656       0   50   4.7  46    4.9   4.2  0.9  2.4  1.4   0.5
4  504      34223       1   33   7.1  46    4.9   4.9  1.0  0.8  2.0   0.4

    BMI CLASS  BMI_Category_Low  BMI_Category_Normal  BMI_Category_High
0  24.0     N             False                 True              False
1  23.0     N             False                 True              False
2  24.0     N             False                 True              False
3  24.0     N             False                 True              False
4  21.0     N             False                 True              False
```

```
Information of all columns:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 7 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   Avg. Area Income              5000 non-null   float64
 1   Avg. Area House Age           5000 non-null   float64
 2   Avg. Area Number of Rooms     5000 non-null   float64
 3   Avg. Area Number of Bedrooms  5000 non-null   float64
 4   Area Population               5000 non-null   float64
 5   Price                         5000 non-null   float64
 6   Address                       5000 non-null   object
dtypes: float64(6), object(1)
memory usage: 273.6+ KB
None

Statistical information of numerical columns:
       Avg. Area Income  Avg. Area House Age  Avg. Area Number of Rooms  \
count       5000.000000          5000.000000                5000.000000
mean       68583.108984             5.977222                   6.987792
std        10657.991214             0.991456                   1.005833
min        17796.631190             2.644304                   3.236194
25%        61480.562390             5.322283                   6.299250
50%        68804.286405             5.970429                   7.002902
75%        75783.338665             6.650808                   7.665871
max       107701.748400             9.519088                  10.759588

       Avg. Area Number of Bedrooms  Area Population         Price
count                   5000.000000      5000.000000  5.000000e+03
mean                       3.981330     36163.516039  1.232073e+06
std                        1.234137      9925.650114  3.531176e+05
min                        2.000000       172.610686  1.593866e+04
25%                        3.140000     29403.928700  9.975771e+05
50%                        4.050000     36199.406690  1.232669e+06
75%                        4.490000     42861.290770  1.471210e+06
max                        6.500000     69621.713380  2.469066e+06

Count of unique labels for 'Ocean Proximity' column:

Columns with missing values:
Series([], dtype: int64)
```

```
Missing values in Diabetes dataset:
ID           0
No_Pation    0
Gender       0
AGE          0
Urea         0
Cr           0
HbA1c        0
Chol         0
TG           0
HDL          0
LDL          0
VLDL         0
BMI          0
CLASS        0
dtype: int64
```

Program 3

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Screenshot:

Lab - 4.

DATE: PAGE:

Linear regression

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.axes as ax
from matplotlib.animation import FuncAnimation

url = 'https://media.geeksforgeeks.org/wp-content/uploads/
   20240320114716/data_for_lr.csv'
data = pd.read_csv(url)
data
data = data.dropna()
train_input = np.array(data.x[0:500]).reshape(500,1)
train_output = np.array(data.y[0:500]).reshape(500,1)
test_input = np.array(data.x[500:700]).reshape(199,1)
test_output = np.array(data.y[500:700]).reshape(199,1)

class LinearRegression:
    def __init__(self):
        self.parameters = {}
    def forward_propogation(self, train_input):
        m = self.parameters['m']
        c = self.parameters['c']
        predictions = np.multiply(m, train_input) + c
        return predictions
    def cost_function(self, predictions, train_output):
        cost = np.mean((train_output - predictions) ** 2)
        return cost
```

```
def backward_propagation(self, train_input, train_output, predictions):
    derivates = {}
    df = (predictions - train_output).
    dm = 2 * np.mean(np.multiply(train_input, df))
    dc = 2 * np.mean(df).
    derivate['dm'] = dm
    derivates['dc'] = dc
    return derivates.

def update_parameters(self, derivates, learning_rate):
    self.parameters['m'] = self.parameters['m'] - learning_
        rate * derivates['dm']
    self.parameters['c'] = self.parameters['c'] - learning_
        rate * derivates['dc'].

def train(self, train_input, train_output, learning_rate,
        iters):
    self.parameters['m'] = np.random.uniform(0,1) * -1
    self.parameters['c'] = np.random.uniform(0,1) * -1
    self.loss = [].
    fig, ax = plt.subplots()
    x_vals = np.linspace(min(train_input), max(train_input),
    100).
    line = ax.plot(x_vals, self.parameters['m'] * x_vals +
        self.parameters['c'], color='red', label=
            'Regression line').
    ax.scatter(train_input, train_output, marker='o',
        color='green', label='Training data').
    ax.set_ylim(0, max(train_output) + 1).
```

```
def update (frame):
    predictions= self. forward_propagation(train_input).
    cost= self. cost_function(predictions, train_output).
    derivatives = self. backward_propagation(
        train_input, train_output, predictions).
    self. update_parameters (derivatives, learning_rate).
    line. set_ydata(self. parameters['m'] * x_vals +
        self. parameters['c'])
    self. loss.append(cost).
    print("Iteration - {}, loss={}".format(frame+1, cost)).
    return line,
ani = ~~FuncAni~~ FuncAnimation(fig, update, frames=iters, interval=
                    200, blit= true).
ani. save('linear_regression_A.gif', writer='ffmpeg').
  plt. xlabel('Input')
  plt . ylabel('output')
  plt. title('Linear Regression')
  plt. legend()
  plt. show()
  return self. parameters, self. loss.

linear_reg = LinearRegression().
parameters, loss= linear_reg. train(train_input, train_output,
    0.0001, 20).
```

Multiple linear regression.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
data={"Feature1":[1,2,3,4,5,6,7,8,9,10],
  "Feature2":[2,3,5,7,11,13,17,19,23,29],
  "Feature3":[3,6,9,12,15,18,21,24,27,30],
  "Target": [5,9,15,22,31,41,53,66,80,96]}
df=pd.DataFrame(data)
x=df.drop(columns=["Target"]).values
y=df["Target"].values.reshape(-1,1)
x=np.hstack((np.ones((x.shape[0],1)),x)).
beta=np.linalg.solve(x.T@x+0.01*np.identity(x.sh
          x.T@y)
y_pred= x@beta.
mse=np.mean((y-(y_pred)**2)
total_variance=np.sum((y-np.mean(y))**2)
explained_variance=np.sum((y_pred-np.mean(y))**2)
r2=explained_variance/total_variance.
print("model coefficients:",beta[1:].flatten())
print("Intercept:",beta[0][0])
print("mean squared Error:",mse).
print("R-squared :",r2).
plt.scatter(y,y_pred,color='blue')
plt.plot(y,y,color='red',linestyle='--')
plt.xlabel("Actual values")
plt.ylabel("Predicted values").
plt.title("Actual vs predicted values")
plt.show()
```

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.axes as ax
from matplotlib.animation import FuncAnimation
```

```
url = 'https://media.geeksforgeeks.org/wp-content/uploads/20240320114716/data_for_lr.csv'
data = pd.read_csv(url)
data
data = data.dropna()
train_input = np.array(data.x[0:500]).reshape(500, 1)
train_output = np.array(data.y[0:500]).reshape(500, 1)
test_input = np.array(data.x[500:700]).reshape(199, 1)
test_output = np.array(data.y[500:700]).reshape(199, 1)
```

```
class LinearRegression:
    def __init__(self):
        self.parameters = {}
    def forward_propagation(self, train_input):
        m = self.parameters['m']
        c = self.parameters['c']
        predictions = np.multiply(m, train_input) + c
        return predictions
    def cost_function(self, predictions, train_output):
        cost = np.mean((train_output - predictions) ** 2)
        return cost
    def backward_propagation(self, train_input, train_output, predictions):
        derivatives = {}
        df = (predictions-train_output)
        dm = 2 * np.mean(np.multiply(train_input, df))
        dc = 2 * np.mean(df)
        derivatives['dm'] = dm
        derivatives['dc'] = dc
        return derivatives
    def update_parameters(self, derivatives, learning_rate):
        self.parameters['m'] = self.parameters['m'] - learning_rate * derivatives['dm']
        self.parameters['c'] = self.parameters['c'] - learning_rate * derivatives['dc']
    def train(self, train_input, train_output, learning_rate, iters):
        self.parameters['m'] = np.random.uniform(0, 1) * -1
        self.parameters['c'] = np.random.uniform(0, 1) * -1
        self.loss = []
        fig, ax = plt.subplots()
```

```
        x_vals = np.linspace(min(train_input), max(train_input), 100)
        line, = ax.plot(x_vals, self.parameters['m'] * x_vals +
                        self.parameters['c'], color='red', label='Regression Line')
        ax.scatter(train_input, train_output, marker='o',
                   color='green', label='Training Data')
        ax.set_ylim(0, max(train_output) + 1)
        def update(frame):
            predictions = self.forward_propagation(train_input)
            cost = self.cost_function(predictions, train_output)
            derivatives = self.backward_propagation(
                train_input, train_output, predictions)
            self.update_parameters(derivatives, learning_rate)
            line.set_ydata(self.parameters['m']
                           * x_vals + self.parameters['c'])
            self.loss.append(cost)
            print("Iteration = {}, Loss = {}".format(frame + 1, cost))
            return line,
        ani = FuncAnimation(fig, update, frames=iters, interval=200, blit=True)
        ani.save('linear_regression_A.gif', writer='ffmpeg')
        plt.xlabel('Input')
        plt.ylabel('Output')
        plt.title('Linear Regression')
        plt.legend()
        plt.show()
        return self.parameters, self.loss
```

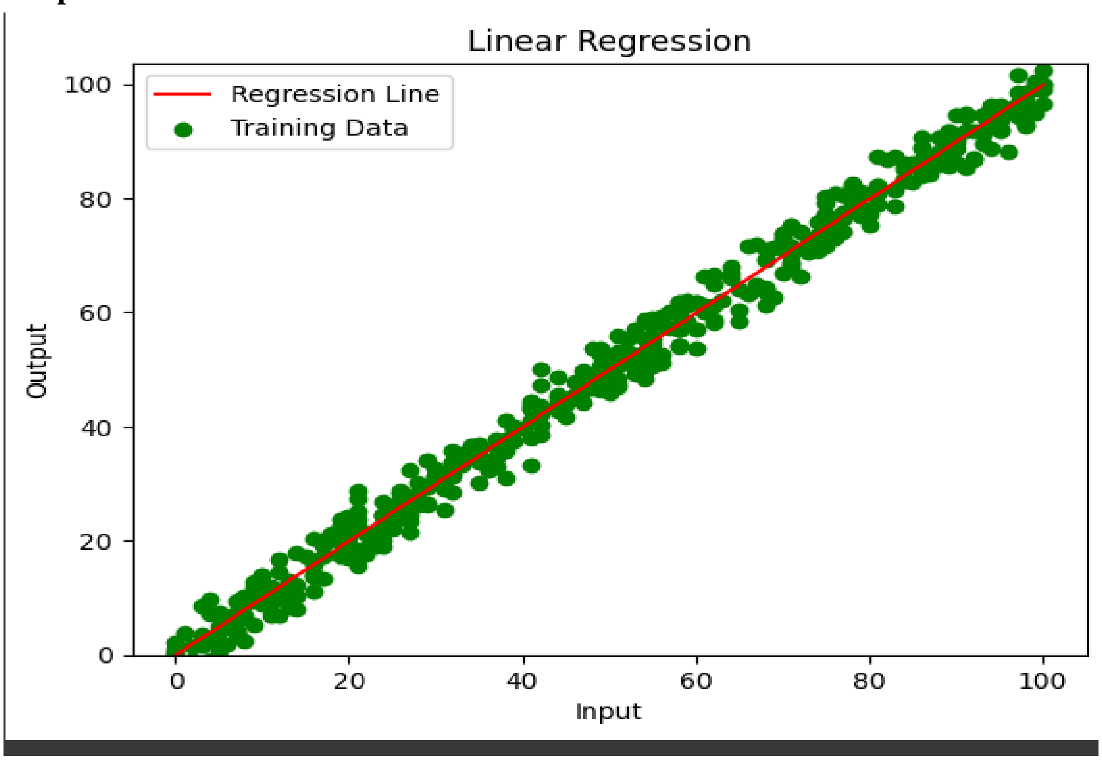
```
linear_reg = LinearRegression()
parameters, loss = linear_reg.train(train_input, train_output, 0.0001, 20)
```

Multi Linear Regression

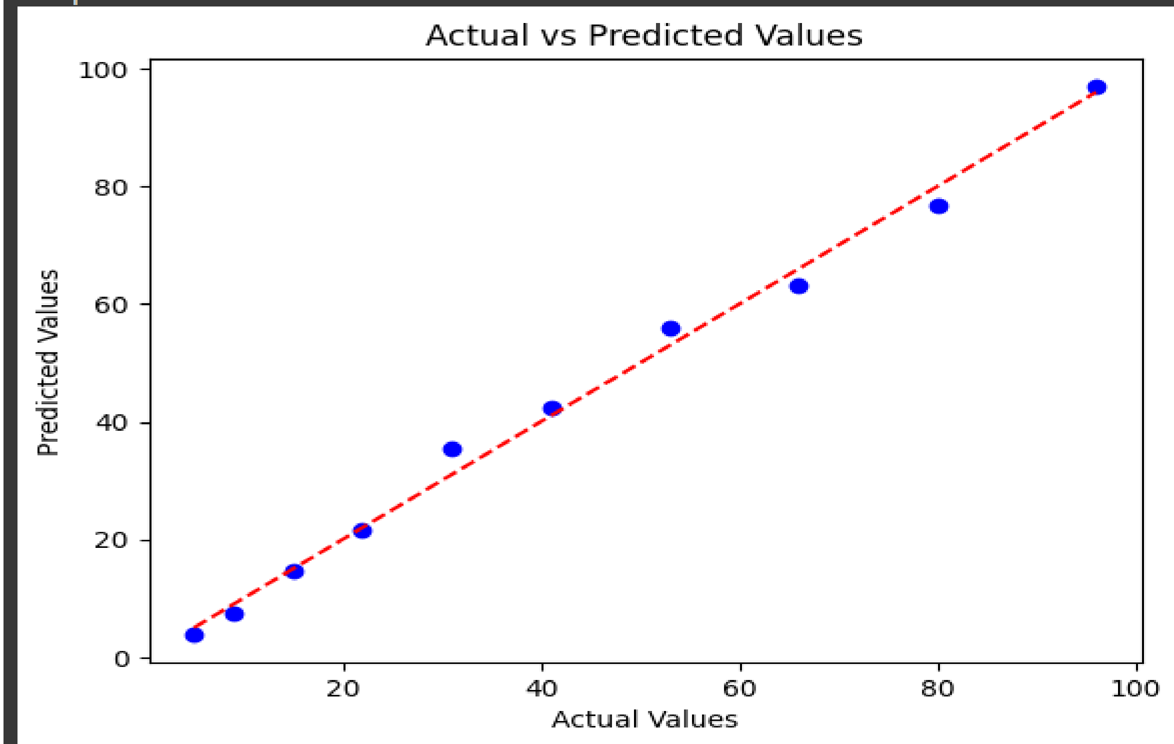
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
data = {
    "Feature1": [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    "Feature2": [2, 3, 5, 7, 11, 13, 17, 19, 23, 29],
    "Feature3": [3, 6, 9, 12, 15, 18, 21, 24, 27, 30],
    "Target": [5, 9, 15, 22, 31, 41, 53, 66, 80, 96]
}
df = pd.DataFrame(data)
X = df.drop(columns=["Target"]).values
y = df["Target"].values.reshape(-1, 1)
```

```
X = np.hstack((np.ones((X.shape[0], 1)), X))
beta = np.linalg.solve(X.T @ X + 0.01 * np.identity(X.shape[1]), X.T @ y)
y_pred = X @ beta
mse = np.mean((y - y_pred) ** 2)
total_variance = np.sum((y - np.mean(y)) ** 2)
explained_variance = np.sum((y_pred - np.mean(y)) ** 2)
r2 = explained_variance / total_variance
print("Model Coefficients:", beta[1:].flatten())
print("Intercept:", beta[0][0])
print("Mean Squared Error:", mse)
print("R-squared Score:", r2)
plt.scatter(y, y_pred, color='blue')
plt.plot(y, y, color='red', linestyle='--')
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title("Actual vs Predicted Values")
plt.show()
```

Output:



```
Model Coefficients: [0.04026032 3.31385993 0.12078097]
Intercept: -3.1599509492879228
Mean Squared Error: 5.362912814290877
R-squared Score: 0.9935326149415827
```



Program 4

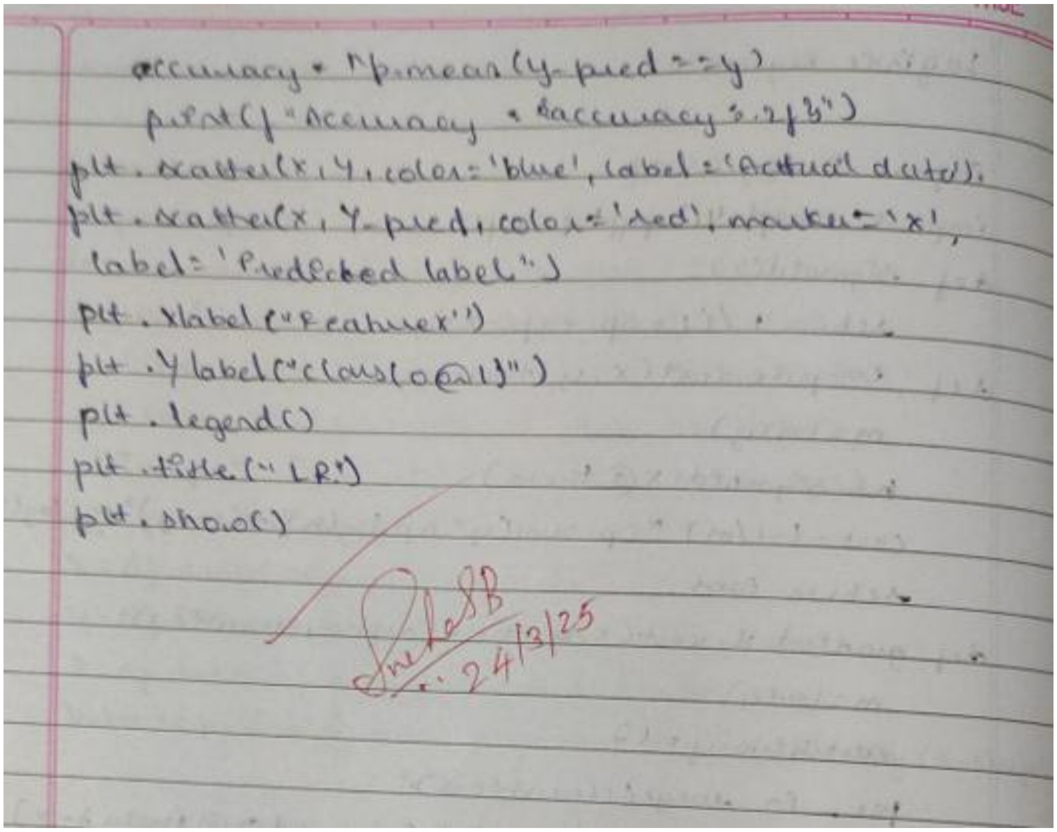
Build Logistic Regression Model for a given dataset

Screenshot:

DATE: PAGE:

Logistic Regression.

```
import numpy as np
import matplotlib.pyplot as plt
def sigmoid(z):
    return 1 /(1 + np.exp(-z))
def compute_cost(X, y, theta):
    m = len(y)
    h = sigmoid(X @ theta)
    cost = (-1/m) * np.sum(y * np.log(h) + (1-y) * np.log(1-h))
    return cost
def gradient_descent(X, y, theta, alpha, iterations):
    m = len(y)
    cost_history = []
    for _ in range(iterations):
        gradient = (1/m) * X.T @ (sigmoid(X @ theta) - Y)
        theta - = alpha * gradient
        cost_history.append(compute_cost(X, Y, theta))
    return theta, cost_history
def predict(X, theta):
    return(sigmoid (X @ theta) >= 0.5).astype(int)
    np.random.seed(42)
    X = np.random.rand(100, 1) * 10
    y = (X > 5 ).astype (int).ravel()
    X_b = np.c_[np.ones((X.shape[0], 1)), X]
    theta = np.zeros (X_b.shape[1])
    alpha = 0.1
    iterations = 1000
    theta, cost_history = gradient_descent(X_b, y, theta,
                    alpha, iterations)
    y_pred = predict(X_b, theta)
```



```
accuracy = np.mean(y_pred == y)
print(f"Accuracy = {accuracy:.2f}")
plt.scatter(X, Y, color='blue', label='Actual data').
plt.scatter(X, Y_pred, color='red', marker='x',
label='Predicted label")
plt.xlabel("Feature")
plt.ylabel("class(0@1)")
plt.legend()
plt.title("LR")
plt.show()
```

24/3/25

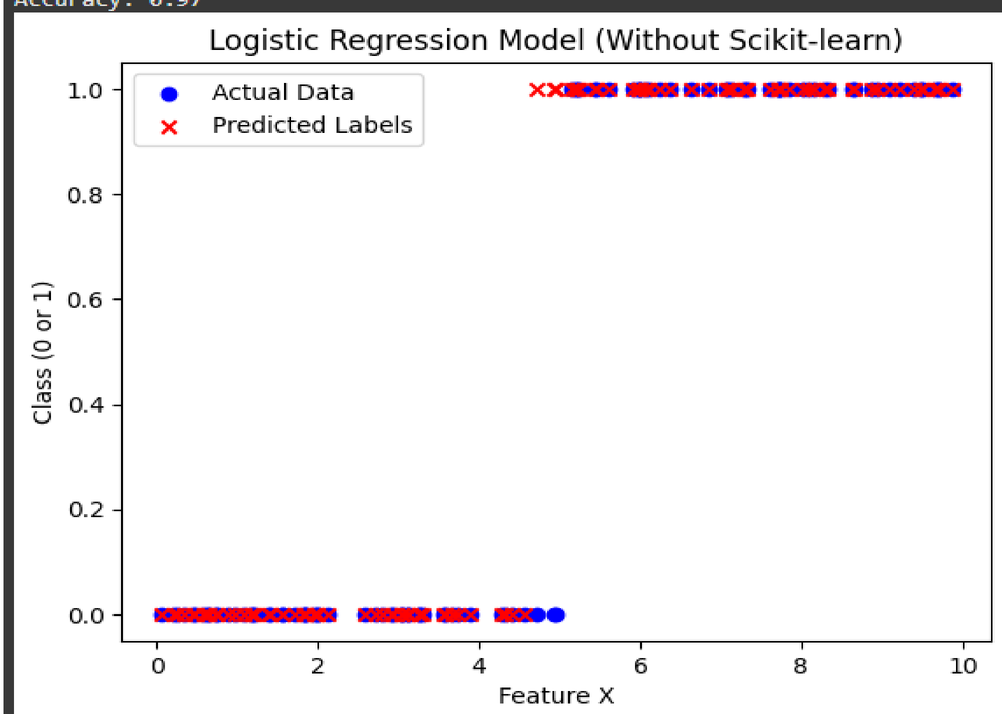
Code:

```
import numpy as np
import matplotlib.pyplot as plt
def sigmoid(z):
    return 1 / (1 + np.exp(-z))
def compute_cost(X, y, theta):
    m = len(y)
    h = sigmoid(X @ theta)
    cost = (-1/m) * np.sum(y * np.log(h) + (1 - y) * np.log(1 - h))
    return cost
def gradient_descent(X, y, theta, alpha, iterations):
    m = len(y)
    cost_history = []
    for _ in range(iterations):
        gradient = (1/m) * X.T @ (sigmoid(X @ theta) - y)
        theta -= alpha * gradient
        cost_history.append(compute_cost(X, y, theta))
    return theta, cost_history
def predict(X, theta):
    return (sigmoid(X @ theta) >= 0.5).astype(int)
np.random.seed(42)
X = np.random.rand(100, 1) * 10
```

```
y = (X > 5).astype(int).ravel()
X_b = np.c_[np.ones((X.shape[0], 1)), X]
theta = np.zeros(X_b.shape[1])
alpha = 0.1
iterations = 1000
theta, cost_history = gradient_descent(X_b, y, theta, alpha, iterations)
y_pred = predict(X_b, theta)
accuracy = np.mean(y_pred == y)
print(f"Accuracy: {accuracy:.2f}")
plt.scatter(X, y, color='blue', label='Actual Data')
plt.scatter(X, y_pred, color='red', marker='x', label='Predicted Labels')
plt.xlabel("Feature X")
plt.ylabel("Class (0 or 1)")
plt.legend()
plt.title("Logistic Regression Model (Without Scikit-learn)")
plt.show()
```

Output:

```
Accuracy: 0.97
```



Program 5

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

Screenshot:

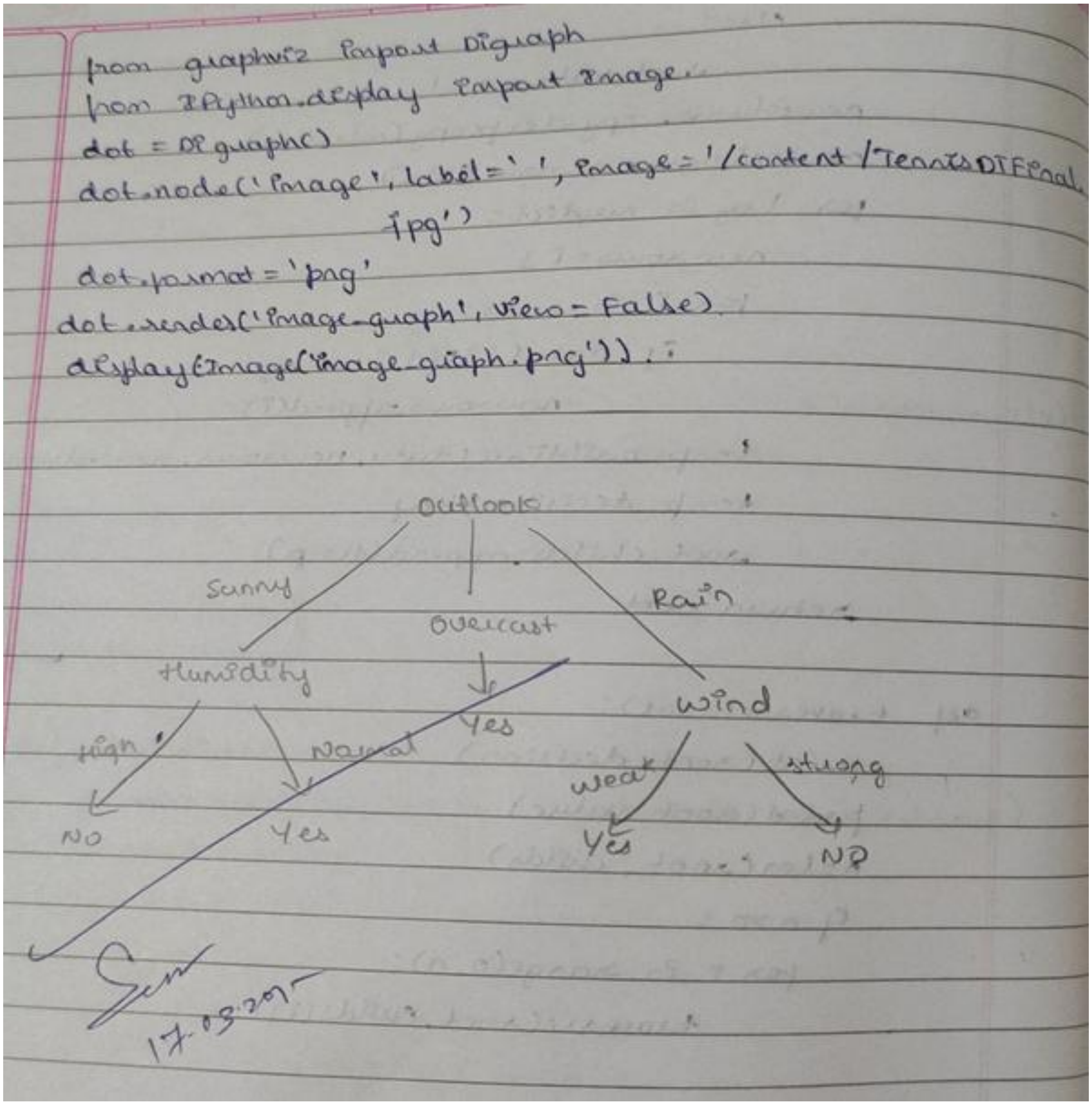
```
Lab-3.

import numpy as np
import matplotlib.pylotpyplot as plt
import pandas as pd
import math
import copy.

dataset = pd.read_csv('/content/Tennis.csv').
X = dataset.iloc[:,:].values.
X

attributes = ['outlook', 'Temp', 'Humidity', 'wind'].

class node(object):
        def __init__(self):
              self.value = None
              self.decision = None
              self.child = None

def findEntropy(data, rows):
      yes = 0
      no = 0
      ans = -1
      idx = len(data[0]) - 1.
      entrophy = 0

      for i in rows:
            if data[i][idx] == 'Yes':
                  Yes = Yes+1.
            else:
```

```
            no=no+1.
    x=yes/(yes+no)
    y=no/(yes+no)
    if x!=0 and y!=0:
        entropy=-1*(x*math.log2(x)+y*math.log2(y))
    if x==1:
        ans=1
    if y==1:
        ans=0
    return entropy,ans

def findMaxGain(data,rows,columns):
    maxgain=0
    retidx=-1
    entropy,ans=findEntropy(data,rows)
    if entropy==0:
        return maxgain,retidx,ans
    for j in columns:
        mydict={}
        idx=j
        for i in rows:
            key=data[i][idx]
            if key not in mydict:
                mydict[key]=1
            else:
                mydict[key]=mydict[key]+1
        gain=entropy
        for key in mydict:
            yes=0
            no=0
            for k in rows:
```

```
                if data[k][j] == key:
                    if data[k][-1] == 'Yes':
                        yes = yes + 1
                    else:
                        no = no + 1
            x = yes/(yes+no)
            y = no/(yes+no)
            if x != 0 and y != 0:
                gain += (mydict[key] * (x * math.log2(x) +
                    y * math.log2(y)))/14.
        if gain > maxgain:
            maxgain = gain
            retidx = j
    return maxgain, retidx, ans.

def buildTree(data, rows, columns):
    maxgain, idx, ans = findMaxgain(X, rows, columns)
    root = Node()
    root.childs = []
    if maxgain == 0:
        if ans == 1:
            root.value = 'Yes'
        else:
            root.value = 'NO'
        return root
    root.value = attribute[idx]
    mydict = {}
    for i in rows:
        key = data[i][idx]
        if key not in mydict:
            mydict[key] = 1.
```

DATE: PAGE:

```
        else:
            root.value+=1.
    newcolumns=copy.deepcopy(columns)
    newcolumns.remove(idx)
    for key in mydict:
        newrows=[]
        for i in rows:
            if data[i][idx]==key:
                newrows.append(i)
        temp=buildTree(data,newrows,newcolumns)
        temp.decision=key
        root.childs.append(temp)
    return root

def traverse(root):
    print(root.decision)
    print(root.value)
    n=len(root.childs)
    if n>0:
        for i in range(0,n):
            traverse(root.childs[i]).

def calculate():
    rows=[i for i in range(0,w)]
    columns=[i for i in range(0,w)].
    root=buildTree(x,rows,columns)
    root.decision='Start'
    traverse(root)

calculate().
```



```
from graphviz import Digraph
from IPython.display import Image
dot = Digraph()
dot.node('image', label=' ', image='/content/TennisDTFinal.
                              jpg')
dot.format = 'png'
dot.render('image_graph', view=False)
display(Image('image_graph.png'))
```


Code:

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

```
dataset = pd.read_csv('/content/Tennis.csv')
X = dataset.iloc[:,:].values
X
```

```
attribute = ['Outlook', 'Temp', 'Humidity', 'Wind']
```

```
class Node(object):
    def __init__(self):
        self.value = None
        self.decision = None
        self.child = None
```

```
def findEntropy(data, rows):
    yes=0
    no=0
    ans=-1
    idx=len(data[0])-1
    entropy=0
    for i in rows:
        if data[i][idx]=='Yes':
            yes=yes+1
        else:
            no=no+1
    x=yes/(yes+no)
    y=no/(yes+no)
    if x!=0 and y!=0:
        entropy= -1*(x*math.log2(x)+y*math.log2(y))
    if x==1:
        ans = 1
    if y==1:
        ans = 0
    return entropy, ans
```

```
def findMaxGain(data, rows, columns):
    maxGain = 0
    retidx = -1
    entropy, ans = findEntropy(data, rows)
    if entropy == 0:
        """if ans == 1:
            print("Yes")
        else:
            print("No")"""
        return maxGain, retidx, ans
    for j in columns:
        mydict = {}
        idx = j
        for i in rows:
            key = data[i][idx]
            if key not in mydict:
```

```
            mydict[key] = 1
        else:
            mydict[key] = mydict[key] + 1
    gain = entropy
    for key in mydict:
        yes = 0
        no = 0
        for k in rows:
            if data[k][j] == key:
                if data[k][-1] == 'Yes':
                    yes = yes + 1
                else:
                    no = no + 1
        x = yes/(yes+no)
        y = no/(yes+no)
        if x != 0 and y != 0:
            gain += (mydict[key] * (x*math.log2(x) + y*math.log2(y)))/14
    if gain > maxGain:
        maxGain = gain
        retidx = j
return maxGain, retidx, ans
```

```
def buildTree(data, rows, columns):
    maxGain, idx, ans = findMaxGain(X, rows, columns)
    root = Node()
    root.childs = []
    if maxGain == 0:
        if ans == 1:
            root.value = 'Yes'
        else:
            root.value = 'No'
        return root
    root.value = attribute[idx]
    mydict = {}
    for i in rows:
        key = data[i][idx]
        if key not in mydict:
            mydict[key] = 1
        else:
            mydict[key] += 1
    newcolumns = copy.deepcopy(columns)
    newcolumns.remove(idx)
    for key in mydict:
        newrows = []
```

```
        for i in rows:
            if data[i][idx] == key:
                newrows.append(i)
        temp = buildTree(data, newrows, newcolumns)
        temp.decision = key
        root.childs.append(temp)
    return root
```

```
def traverse(root):
    print(root.decision)
    print(root.value)
    n = len(root.childs)
    if n > 0:
        for i in range(0, n):
            traverse(root.childs[i])
```

```
def calculate():
    rows = [i for i in range(0, 14)]
    columns = [i for i in range(0, 4)]
    root = buildTree(X, rows, columns)
    root.decision = 'Start'
    traverse(root)
```

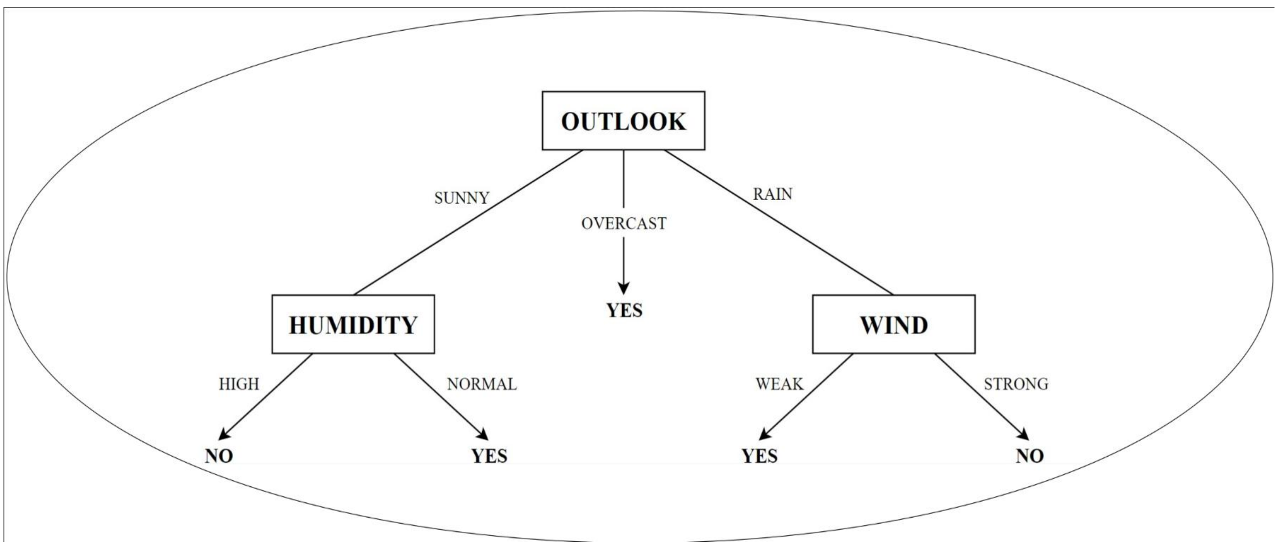
```
calculate()
```

```
from graphviz import Digraph
from IPython.display import Image
dot = Digraph()
dot.node('image', label='', image='/content/TennisDTFinal.jpg')
dot.format = 'png'
dot.render('image_graph', view=False)
display(Image('image_graph.png'))
```

Output:

```
array([['Sunny', 'Hot', 'High', 'Weak', 'No'],
       ['Sunny', 'Hot', 'High', 'Strong', 'No'],
       ['Overcast', 'Hot', 'High', 'Weak', 'Yes'],
       ['Rain', 'Mild', 'High', 'Weak', 'Yes'],
       ['Rain', 'Cool', 'Normal', 'Weak', 'Yes'],
       ['Rain', 'Cool', 'Normal', 'Strong', 'No'],
       ['Overcast', 'Cool', 'Normal', 'Strong', 'Yes'],
       ['Sunny', 'Mild', 'High', 'Weak', 'No'],
       ['Sunny', 'Cool', 'Normal', 'Weak', 'Yes'],
       ['Rain', 'Mild', 'Normal', 'Weak', 'Yes'],
       ['Sunny', 'Mild', 'Normal', 'Strong', 'Yes'],
       ['Overcast', 'Mild', 'High', 'Strong', 'Yes'],
       ['Overcast', 'Hot', 'Normal', 'Weak', 'Yes'],
       ['Rain', 'Mild', 'High', 'Strong', 'No']], dtype=object)
```

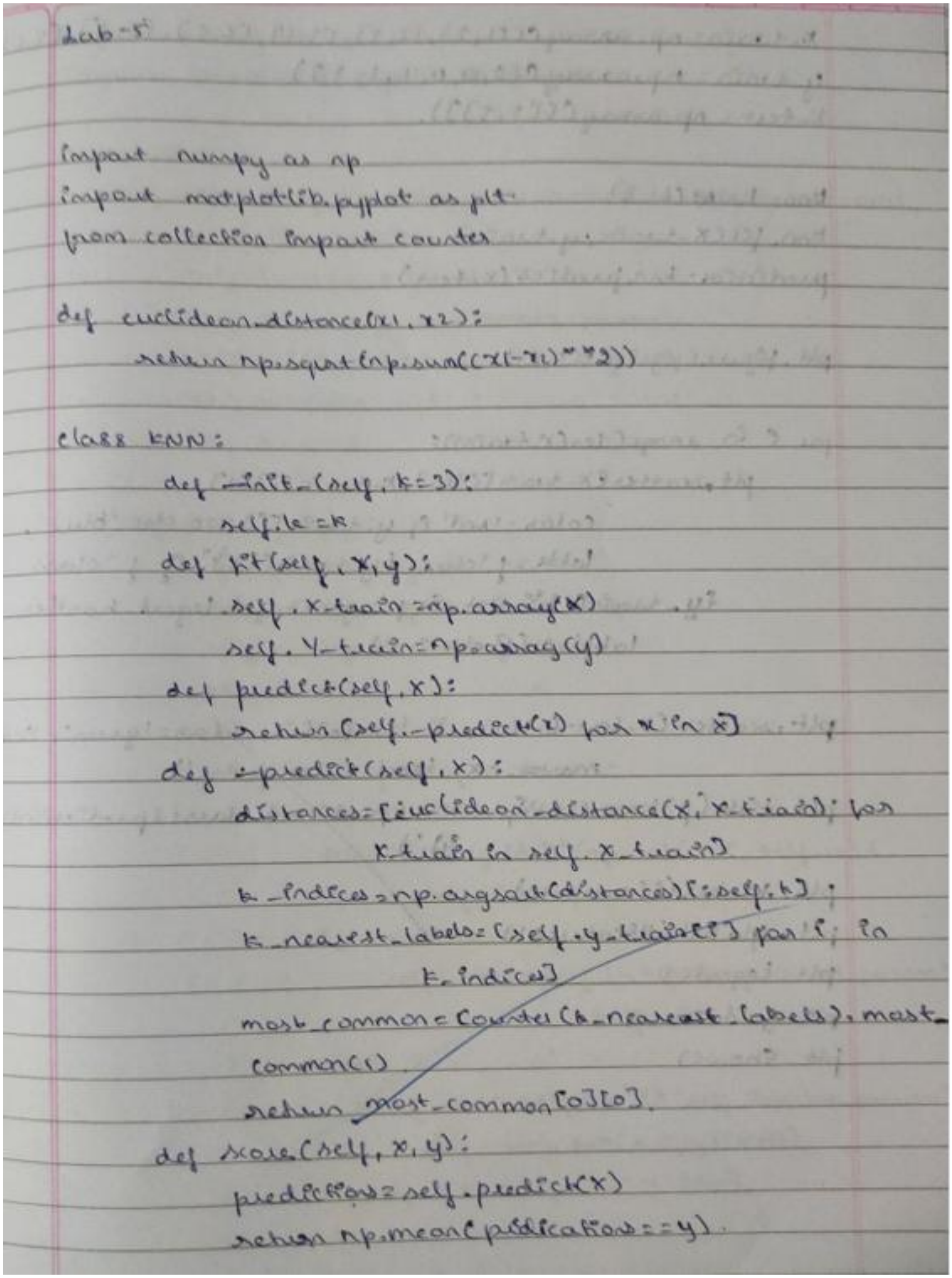
```
Start
Outlook
Sunny
Humidity
High
No
Normal
Yes
Overcast
Yes
Rain
Wind
Weak
Yes
Strong
No
```



Program 6

Build KNN Classification model for a given dataset

Screenshot:



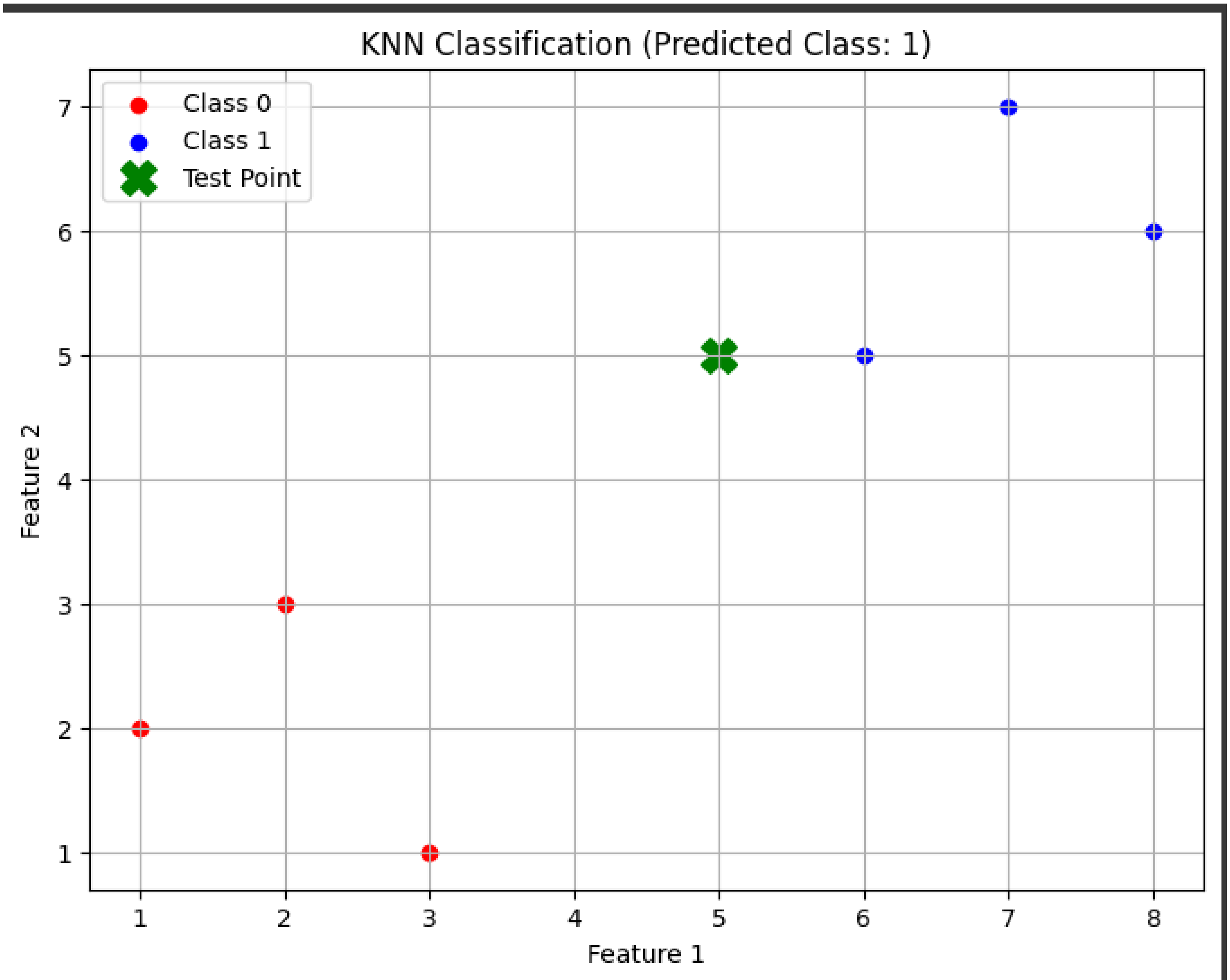
```
Lab-5

import numpy as np
import matplotlib.pyplot as plt
from collection import counter

def euclidean_distance(x1, x2):
    return np.sqrt(np.sum((x1-x2)**2))

class KNN:
    def __init__(self, k=3):
        self.k = k
    def fit(self, X, y):
        self.X_train = np.array(X)
        self.Y_train = np.array(y)
    def predict(self, X):
        return [self._predict(x) for x in X]
    def _predict(self, x):
        distances = [euclidean_distance(x, x_train) for
                x_train in self.X_train]
        k_indices = np.argsort(distances)[:self.k]
        k_nearest_labels = [self.y_train[i] for i in
                k_indices]
        most_common = Counter(k_nearest_labels).most_
        common(1)
        return most_common[0][0]
    def score(self, X, y):
        predictions = self.predict(X)
        return np.mean(predications == y)
```

```
x_train = np.array([[1,2],[2,2],[3,1],[6,5],[7,7],[8,6]])
y_train = np.array([0,0,0,1,1,1])
x_test = np.array([[5,5]]).

knn = KNN(k=3)
knn.fit(x_train, y_train)
prediction = knn.predict(x_test).

plt.figure(figsize=(8,6))

for i in range(len(x_train)):
        plt.scatter(x_train[i][0], x_train[i][1],
                    color='red' if y_train[i]==0 else 'blue',
                    lable = f"class {y_train[i]}" if f"class
            {y_train[i]}" not in pt.gca().get_legend_handles_
                labels()[1] else "")

plt.scatter(x_test[0][0], x_test[0][1], color='green', s=200,
                    marker='x', label='Test point').
plt.title(f"KNN Classification (predicted class: {prediction[0]
                    }) ").
plt.xlabel("Feature1")
plt.ylabel("Feature2")
plt.legend()
plt.grid(True)
plt.show()
```

Code:

```
import numpy as np
import matplotlib.pyplot as plt
from collections import Counter
def euclidean_distance(x1, x2):
    return np.sqrt(np.sum((x1 - x2) ** 2))
class KNN:
    def __init__(self, k=3):
        self.k = k
    def fit(self, X, y):
        self.X_train = np.array(X)
        self.y_train = np.array(y)
    def predict(self, X):
        return [self._predict(x) for x in X]
    def _predict(self, x):
        distances = [euclidean_distance(x, x_train) for x_train in self.X_train]
        k_indices = np.argsort(distances)[:self.k]
        k_nearest_labels = [self.y_train[i] for i in k_indices]
        most_common = Counter(k_nearest_labels).most_common(1)
        return most_common[0][0]
    def score(self, X, y):
        predictions = self.predict(X)
        return np.mean(predictions == y)
X_train = np.array([[1, 2], [2, 3], [3, 1], [6, 5], [7, 7], [8, 6]])
y_train = np.array([0, 0, 0, 1, 1, 1])
X_test = np.array([[5, 5]])
knn = KNN(k=3)
knn.fit(X_train, y_train)
prediction = knn.predict(X_test)
plt.figure(figsize=(8, 6))
for i in range(len(X_train)):
    plt.scatter(X_train[i][0], X_train[i][1],
                color='red' if y_train[i] == 0 else 'blue',
                label=f"Class {y_train[i]}" if f"Class {y_train[i]}" not in plt.gca().get_legend_handles_labels()[1] else "")
plt.scatter(X_test[0][0], X_test[0][1], color='green', s=200, marker='X', label='Test Point')
plt.title(f"KNN Classification (Predicted Class: {prediction[0]})")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.legend()
plt.grid(True)
plt.show()
```

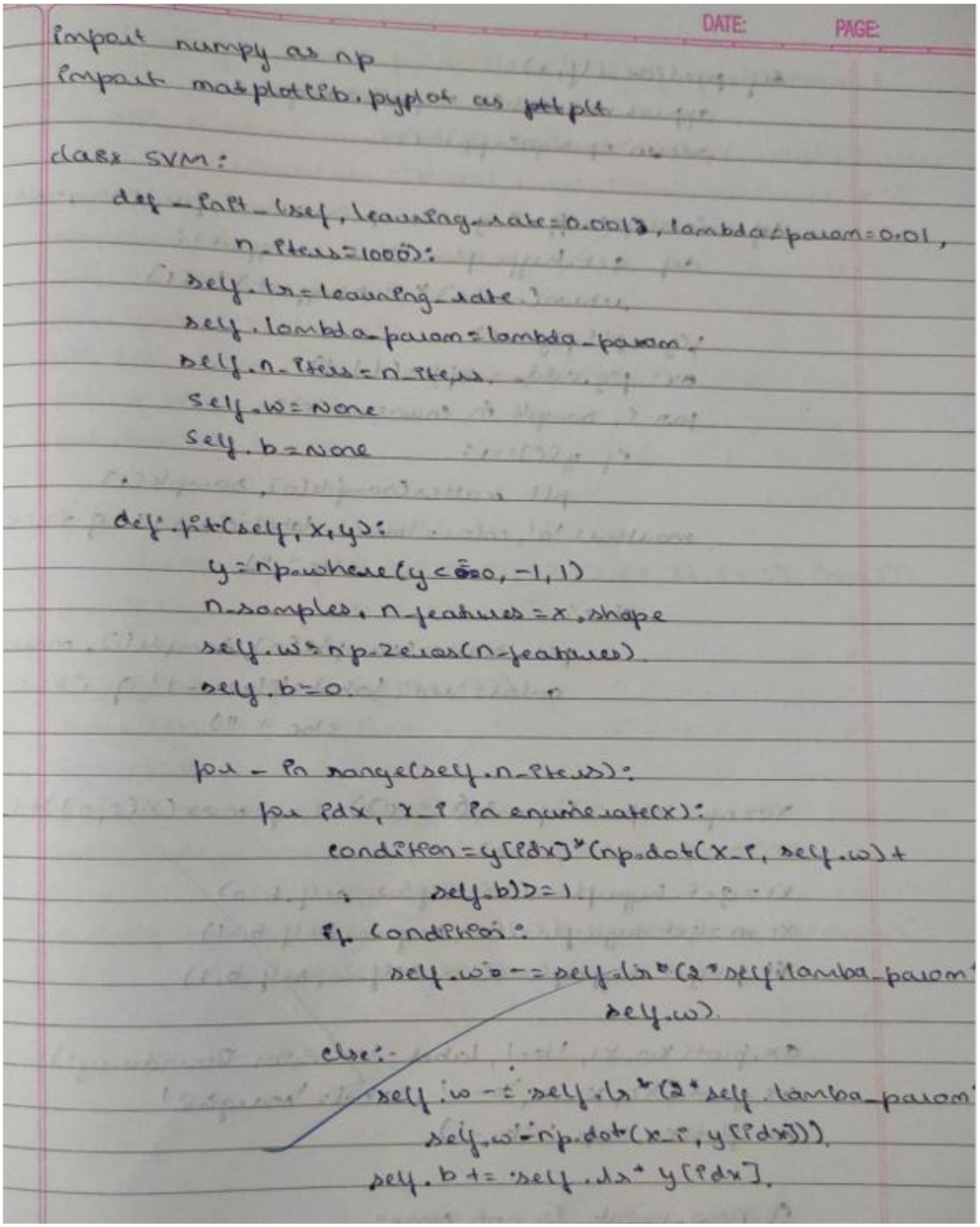
Output:


KNN Classification (Predicted Class: 1)
Class 0
Class 1
Test Point
Feature 2
Feature 1
1
2
3
4
5
6
7
8

Program 7

Build Support vector machine model for a given dataset

Screenshot:



```
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.lamba_param *
                            self.w)
                else:
                    self.w -= self.lr * (2 * self.lamba_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=
            color='red', label='class -1' if i==0
                    else " ")

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

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

if new_point is not None:
    color='green' if prediction==1 else 'orange'
```

```
label= f'New point: class {"1" if prediction==1
        else "0"}'
plt.scatter(new_point[0], new_point[1], c=color, s=100,
        edgecolors='black', label=label, marker='x')

ax.legend()
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.title("SVM with new point prediction")
plt.grid(True)
plt.show()

if __name__ == "__main__":
    X=np.array([[1,7],[2,8],[3,8],[8,7],[9,2],[10,2]])
    y=np.array([0,0,0,1,1,1])

new_point = np.array([[5,5]])
svm= SVM()
svm.fit(X,y)
prediction=svm.predict(new_point)[0]
svm.visualize(X,y,new_point=new_point[0], prediction=
        prediction)
print(f"New point {new_point[0]} classified as: {'class 1'
        if prediction==1 else 'class 0'}")
```

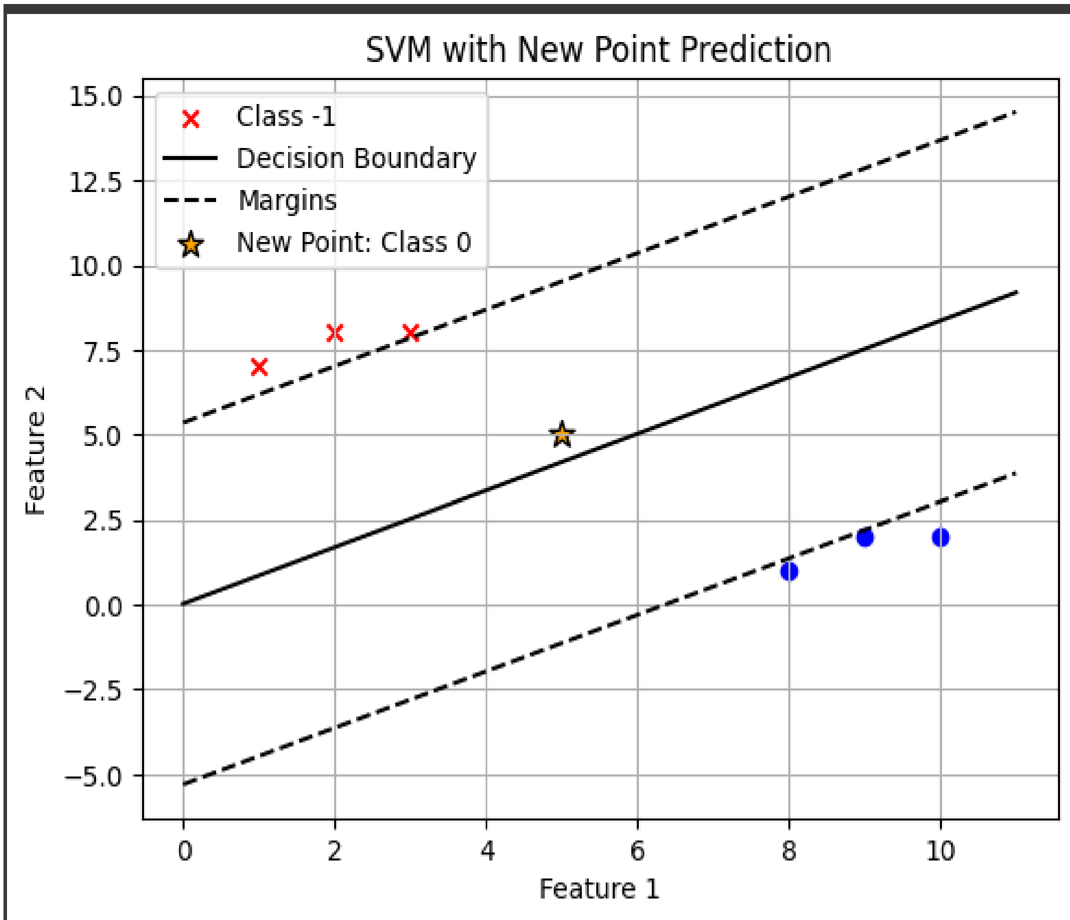
Code:

```
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)
        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, 100)
        x1 = get_hyperplane(x0, self.w, self.b, 0)
        x1_m = get_hyperplane(x0, self.w, self.b, -1)
        x1_p = get_hyperplane(x0, self.w, self.b, 1)
        ax.plot(x0, x1, 'k-', label='Decision Boundary')
        ax.plot(x0, x1_m, 'k--', label='Margins')
        ax.plot(x0, x1_p, 'k--')
        if new_point is not None:
            color = 'green' if prediction == 1 else 'orange'
            label = f'New Point: Class {"1" if prediction == 1 else "0"}'
            plt.scatter(new_point[0], new_point[1], c=color, s=100, edgecolors='black', label=label, marker='*')
        ax.legend()
        plt.xlabel("Feature 1")
        plt.ylabel("Feature 2")
```

```
        plt.title("SVM with New Point Prediction")
        plt.grid(True)
        plt.show()
if __name__ == "__main__":
    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 = np.array([[5, 5]])
    svm = SVM()
    svm.fit(X, y)
    prediction = svm.predict(new_point)[0]
    svm.visualize(X, y, new_point=new_point[0], prediction=prediction)
    print(f"New point {new_point[0]} classified as: {'Class 1' if prediction == 1 else 'Class 0'}")
```

Output:




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

Program 8

Implement Random forest ensemble method on a given dataset

Screenshot:

21/04/25 Lab-6

1. Random forest ensemble algorithm.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score,
classification_report.

url= "https://raw.githubusercontent.com/jbrownlee/
Datasets/master/pima-indians-diabetes.data.csv"
columns= ["Pregnancies", "Glucose", "BloodPressure",
"SkinThickness", "Insulin", "BMI", "DiabetesPedigreeFunction",
        "Age", "target"].
df = pd.read_csv(url, names= columns).
print("Dataset Preview:\n", df.head()).
x = df.drop("target", axis=1)
y = df["target"].
x_train, x_test, y_train, y_test = train_test_split
      (x, y, test_size =0.3, random_state =42)
rf_model = RandomForestClassifier (n_estimators =100,
          random_state =42)
rf_model.fit (x_train, y_train)
y_pred = rf_model.predict (x_test);
print("\nAccuracy:", accuracy_score(y_test, y_pred))
print("\n classification Report:\n", classification_report
          (y_test, y_pred)).
```

Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
columns = [
    "Pregnancies", "Glucose", "BloodPressure", "SkinThickness",
    "Insulin", "BMI", "DiabetesPedigreeFunction", "Age", "target"
]
df = pd.read_csv(url, names=columns)
print("Dataset Preview:\n", df.head())
X = df.drop('target', axis=1)
y = df['target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
y_pred = rf_model.predict(X_test)
print("\nAccuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

Output:

```
Dataset Preview:
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  target
0                     0.627   50       1
1                     0.351   31       0
2                     0.672   32       1
3                     0.167   21       0
4                     2.288   33       1

Accuracy: 0.7532467532467533

Classification Report:
               precision    recall  f1-score   support

           0       0.82      0.80      0.81       151
           1       0.64      0.66      0.65        80

    accuracy                           0.75       231
   macro avg       0.73      0.73      0.73       231
weighted avg       0.76      0.75      0.75       231
```

Program 9

Implement Boosting ensemble method on a given dataset

Screenshot:

DATE: PAGE:

```
9. Boosting.

import pandas as pd.
from sklearn.model_selection import train_test_split.
from sklearn.ensemble import AdaBoostclassifier,
from sklearn.metrics import accuracy_score,
      classification_report.

url="https://raw.githubusercontent.com/jbrownlee/Datasets/
master/pima-indians-diabetes.data.csv".
columns=["Pregnancies", "glucose", "BloodPressure", "SkinThick
ness", "Insulin", "BMI", "DiabetesPedigreeFunction", "age",
      "target"]
df=pd.read_csv(url,names=columns).
print("Dataset Preview:\n", df.head()).
X=df.drop('target', axis=1)
y=df['target']
X_train, X_test, y_train, y_test=train_test_split(X,y,
      test_size=0.2, random_state=40)
boost_model=AdaBoostclassifier(n_estimators=100, learning_rate=
                              1.0, random_state=40)
boost_model.fit(X_train, y_train).
y_pred=boost_model.predict(X_test).
print("Accuracy:", accuracy_score(y_test, y_pred))
print("classification Report:\n", classification_report(y_test,
              y_pred)).
```

Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score, classification_report

url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
columns = [
    "Pregnancies", "Glucose", "BloodPressure", "SkinThickness",
    "Insulin", "BMI", "DiabetesPedigreeFunction", "Age", "target"
]
df = pd.read_csv(url, names=columns)
print("Dataset Head:\n", df.head())
X = df.drop('target', axis=1)
y = df['target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=40)
boost_model = AdaBoostClassifier(n_estimators=100, learning_rate=1.0, random_state=40)
boost_model.fit(X_train, y_train)
y_pred = boost_model.predict(X_test)
print("\n--- AdaBoost Results ---")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

Output:

```
Dataset Head:
    Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  target
0                     0.627   50       1
1                     0.351   31       0
2                     0.672   32       1
3                     0.167   21       0
4                     2.288   33       1

--- AdaBoost Results ---
Accuracy: 0.7489177489177489
Classification Report:
               precision    recall  f1-score   support

           0       0.75      0.88      0.81       142
           1       0.74      0.54      0.62        89

    accuracy                           0.75       231
   macro avg       0.75      0.71      0.72       231
weighted avg       0.75      0.75      0.74       231
```

Program 10

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

Screenshot:

3. k means clustering.

```
import pandas as pd
import matplotlib.pyplot as plt.
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn. decomposition import PCA.

url= "https://raw.githubusercontent.com/jbrownlee/Datasets/
 master/pima-indians-diabetes.data.csv".
columns = ["Pregnancies", "glucose", "BloodPressure",
 "SkinThickness", "Insulin", "BMI", "DiabetesPedigreeFunction",
       "Age", "target"].
df = pd.read_csv(url, names=columns)
print("Dataset Preview:\n", df.head())
x=df.drop('target',axis=1)
scaler=StandardScaler()
scaled_data = scaler.fit_transform(x)
k=3
kmeans= KMeans(n_clusters=k, random_state=42)
kmeans.fit(scaled_data).
df['cluster']= kmeans.labels_
print("clustered Data Preview:\n", df.head())
pca = PCA(n_components=2)
reduced_data=pca.fit_transform(scaled_data)
plt.figure(figsize=(8,6)).
plt.scatter(reduced_data[:,0], reduced_data[:,1],
   c=kmeans.labels_, cmap='viridis').
plt.scatter(kmeans.cluster_centers_[:,0], kmeans.cluster_
        centers_[:,1], s=300, c='red', marker='x',
       label='centroids').
```

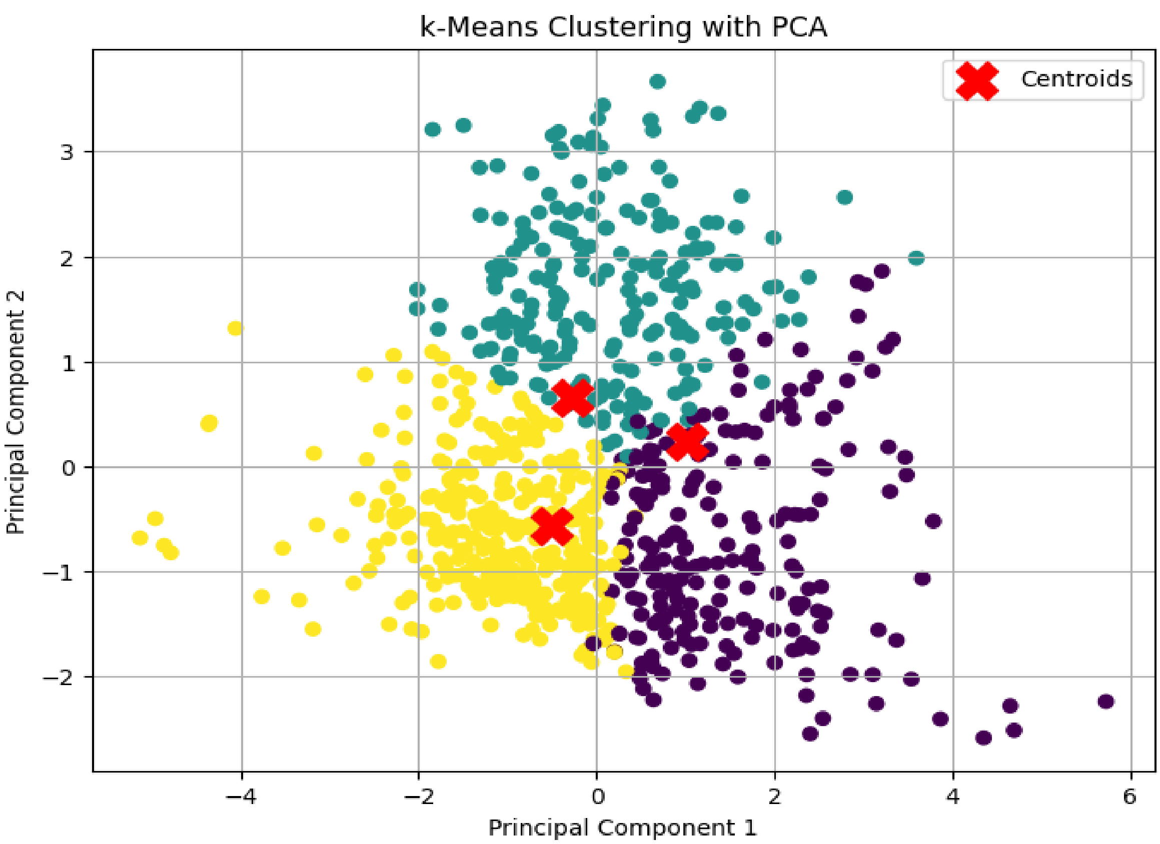
```
plt.title("K-means clustering with PCA")
plt.xlabel('Principal component 1")
plt.ylabel("Principal component 2")
plt.legend()
plt.grid(True)
plt.show()
```

Code:

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
columns = [
    "Pregnancies", "Glucose", "BloodPressure", "SkinThickness",
    "Insulin", "BMI", "DiabetesPedigreeFunction", "Age", "target"
]
df = pd.read_csv(url, names=columns)
print("Dataset Preview:\n", df.head())
X = df.drop('target', axis=1)
scaler = StandardScaler()
scaled_data = scaler.fit_transform(X)
k = 3
kmeans = KMeans(n_clusters=k, random_state=42)
kmeans.fit(scaled_data)
df['Cluster'] = kmeans.labels_
print("\nClustered Data Preview:\n", df.head())
pca = PCA(n_components=2)
reduced_data = pca.fit_transform(scaled_data)
plt.figure(figsize=(8, 6))
plt.scatter(reduced_data[:, 0], reduced_data[:, 1], c=kmeans.labels_, cmap='viridis')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=300, c='red', marker='X',
label='Centroids')
plt.title("k-Means Clustering with PCA")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.legend()
plt.grid(True)
plt.show()
```

Output:

```
Dataset Preview:
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  target
0                     0.627   50       1
1                     0.351   31       0
2                     0.672   32       1
3                     0.167   21       0
4                     2.288   33       1

Clustered Data Preview:
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  target  Cluster
0                     0.627   50       1        1
1                     0.351   31       0        2
2                     0.672   32       1        1
3                     0.167   21       0        2
4                     2.288   33       1        0
```



Program 11

Implement Dimensionality reduction using Principal Component Analysis (PCA) method

Screenshot:

u. PCA.

```
import pandas as pd.
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler,
import matplotlib.pyplot as plt.

url = "https://raw.githubusercontent.com/jbrownlee/Datasets/
master/pima-indians-diabetes.data.csv".
columns = ["Pregnancies", "glucose", "BloodPressure", "SkinThickne
-ss', "Insulin", "BMI", "DiabetesPedigreeFunction", "Age",
        "target"]
df = pd.read_csv(url, names=columns).
print("Original Data:\n", df.head()).
scaler = StandardScaler().
scaled_data = scaler.fit_transform(scaled_data)
pca_df = pd.DataFrame(data=pca_result, columns=['PC1', 'PC2'])
print("\n PCA Reduced Data:\n", pca_df.head()).

plt.figure(figsize=(8,6)).
plt.scatter(pca_df['PC1'], pca_df['PC2'], alpha=0.7).
plt.xlabel('Principal component1')
plt.ylabel('Principal component2).
plt.title('PCA - Dimensionality Reduction').
```

DATE:

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

Code:

```
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
columns = [
    "Pregnancies", "Glucose", "BloodPressure", "SkinThickness",
    "Insulin", "BMI", "DiabetesPedigreeFunction", "Age", "target"
]
df = pd.read_csv(url, names=columns)
print("Original Data:\n", df.head())
scaler = StandardScaler()
scaled_data = scaler.fit_transform(df)
pca = PCA(n_components=2)
pca_result = pca.fit_transform(scaled_data)
pca_df = pd.DataFrame(data=pca_result, columns=['PC1', 'PC2'])
print("\nPCA Reduced Data:\n", pca_df.head())
plt.figure(figsize=(8, 6))
plt.scatter(pca_df['PC1'], pca_df['PC2'], alpha=0.7)
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA - Dimensionality Reduction')
plt.grid(True)
plt.show()
```

Output:

```
Original Data:
    Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0             6      148             72             35        0  33.6
1             1       85             66             29        0  26.6
2             8      183             64              0        0  23.3
3             1       89             66             23       94  28.1
4             0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  target
0                     0.627   50       1
1                     0.351   31       0
2                     0.672   32       1
3                     0.167   21       0
4                     2.288   33       1

PCA Reduced Data:
        PC1       PC2
0  1.756947  1.111743
1 -1.507421 -0.559406
2  0.650822  1.929576
3 -1.587398 -1.065075
4  2.483374 -2.359563
```