VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

Rushila V (1BM22CS226)

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
Mar-2025 to July-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 **Rushila V (1BM22CS226),** 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

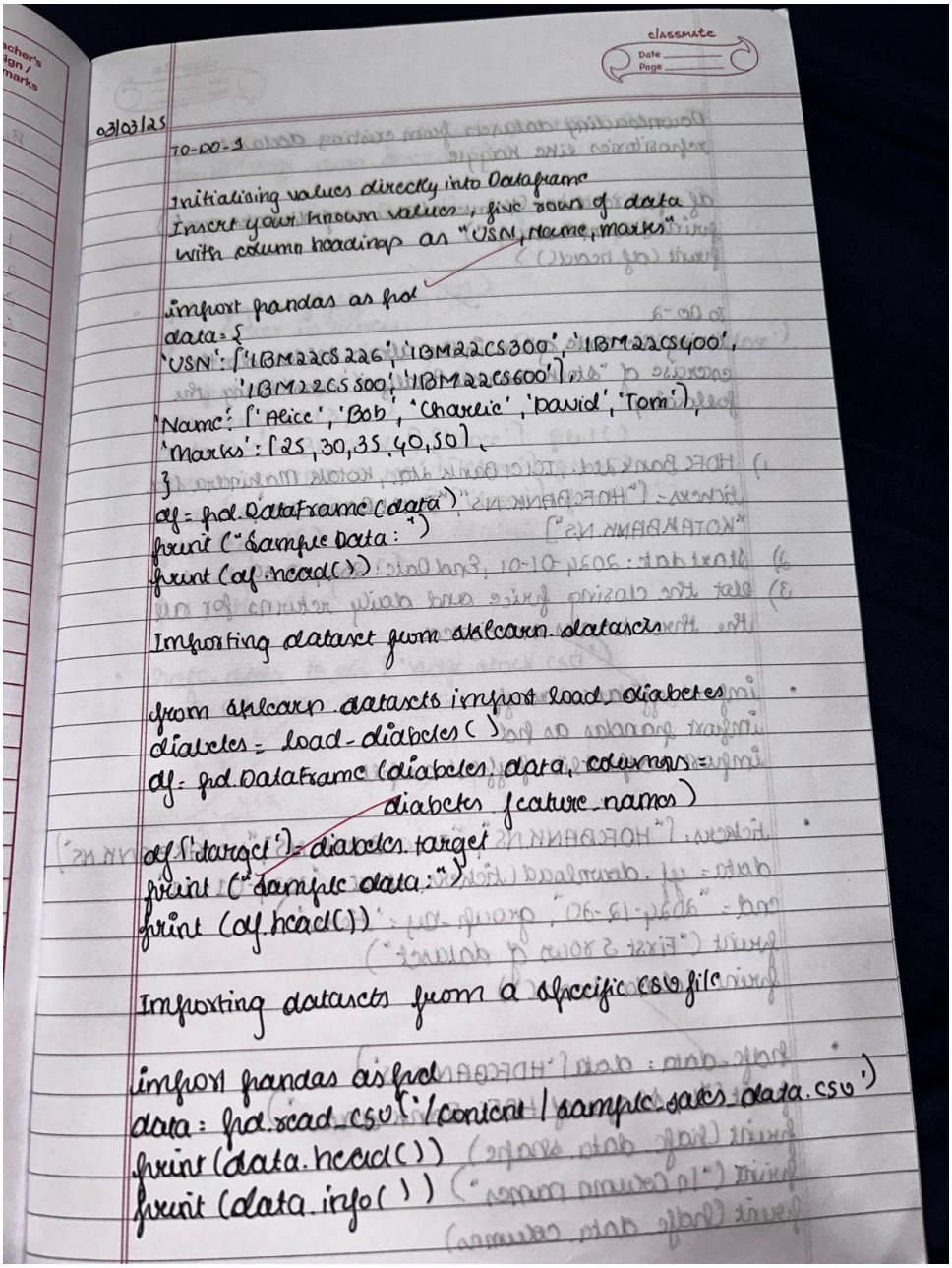
Index

Github Link:
https://github.com/Rushila226/ML

Program 1

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

Screenshot :



Downloading datasets from existing dataset repositories like Kaggle

```
df = pd.read_csv('/content/Dataset of diabetes.csv')
print('Sample data:')
print(df.head())
```

To Do - 2

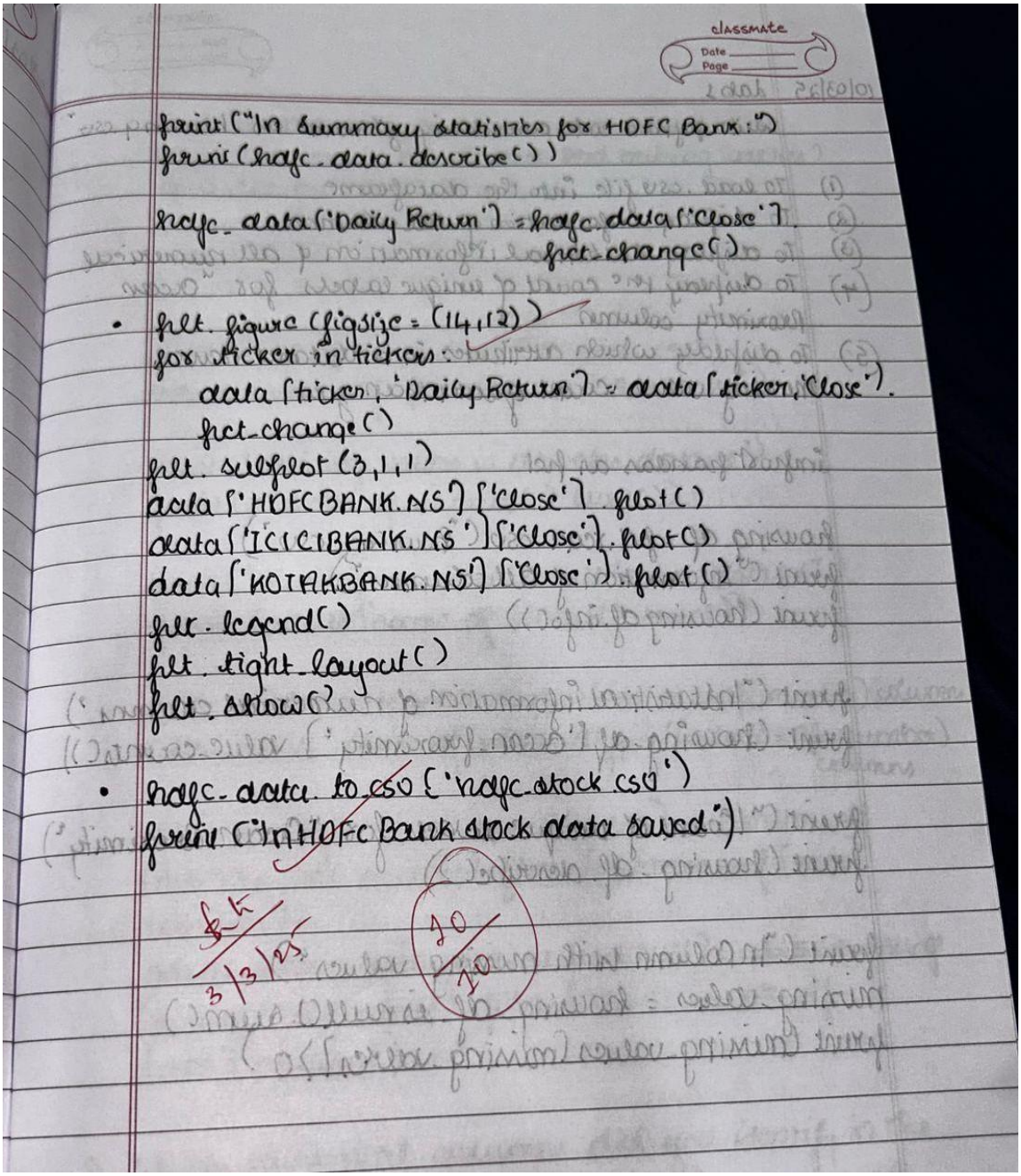
Using the code given in the above slides do the exercise of "stock market Analysis" considering the following

1) HDFC Bank Ltd., ICICI Bank Ltd., Kotak Mahindra 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 mentioned

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

-
```
tickers = ["HDFCBANK.NS", "ICICIBANK.NS", "KOTAKBANK
data = yf.download(tickers, start = "2024-01-01",
end = "2024-12-30", group_by = 'ticker')
print("First 5 rows of dataset")
print(data.head())
```

-
```
hdfc_data = data['HDFCBANK.NS']
print("In shape of HDFC Bank data:")
print(hdfc_data.shape)
print("In Column names")
print(hdfc_data.columns)
```



Code :

```
import pandas as pd
data = {
 'USN':['1BM22CS226','1BM22CS300','1BM22CS400','1BM22CS500','1BM22CS600'],
'Name': ['Alice', 'Bob', 'Charlie', 'David','Tom'],
'Marks': [25, 30, 35, 40,50],
}
df = pd.DataFrame(data)
print("Sample data:")
print(df.head())
```

```
import pandas as pd
# Load the dataset from a CSV file
data = pd.read_csv('/content/sample_sales_data.csv')
# Display the first few rows of the dataset
print(data.head())
# Get some basic information about the dataset
print(data.info())
```

```
tickers = ["HDFCBANK.NS", "ICICIBANK.NS", "KOTAKBANK.NS"]
data = yf.download(tickers, start="2024-01-01", end="2024-12-30",
group_by='ticker')
print("First 5 rows of the dataset:")
print(data.head())
```

```
tickers = ["HDFCBANK.NS", "ICICIBANK.NS", "KOTAKBANK.NS"]
data = yf.download(tickers, start="2024-01-01", end="2024-12-30",
group_by='ticker')
# Access HDFC Bank data using MultiIndex
hdfc_data = data[('HDFCBANK.NS',)]  # Accessing with a tuple for MultiIndex

print("\nShape of the HDFC Bank data:")
print(hdfc_data.shape)

print("\nColumn names:")
print(hdfc_data.columns)

print("\nSummary statistics for HDFC Bank:")
print(hdfc_data.describe())

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

```
plt.figure(figsize=(14, 12))

# Calculate Daily Returns for all tickers within the 'data' DataFrame
for ticker in tickers:
    data[ticker, 'Daily Return'] = data[ticker, 'Close'].pct_change()

# Plot Closing Prices
plt.subplot(3, 1, 1)
data['HDFCBANK.NS']['Close'].plot(title="HDFC Bank - Closing Price",
color='blue')
```

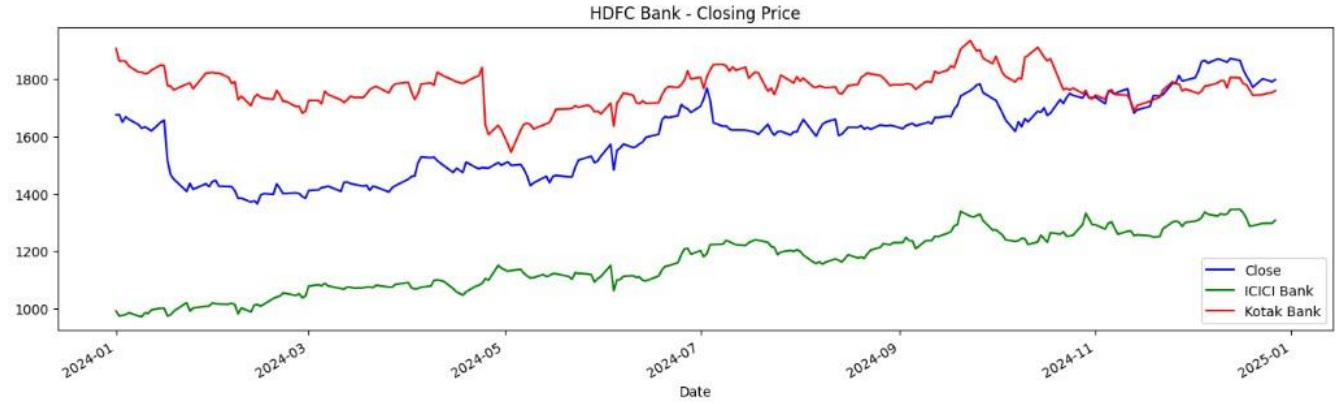
```
data['ICICIBANK.NS']['Close'].plot(label="ICICI Bank", color='green')
data['KOTAKBANK.NS']['Close'].plot(label="Kotak Bank", color='red')
plt.legend()

# Plot Daily Returns using the calculated column within 'data'
plt.subplot(3, 1, 2)
data['HDFCBANK.NS', 'Daily Return'].plot(title="HDFC Bank - Daily Returns",
color='blue')
data['ICICIBANK.NS', 'Daily Return'].plot(label="ICICI Bank", color='green')
data['KOTAKBANK.NS', 'Daily Return'].plot(label="Kotak Bank", color='red')
plt.legend()

# Adjust layout
plt.tight_layout()
plt.show()
```

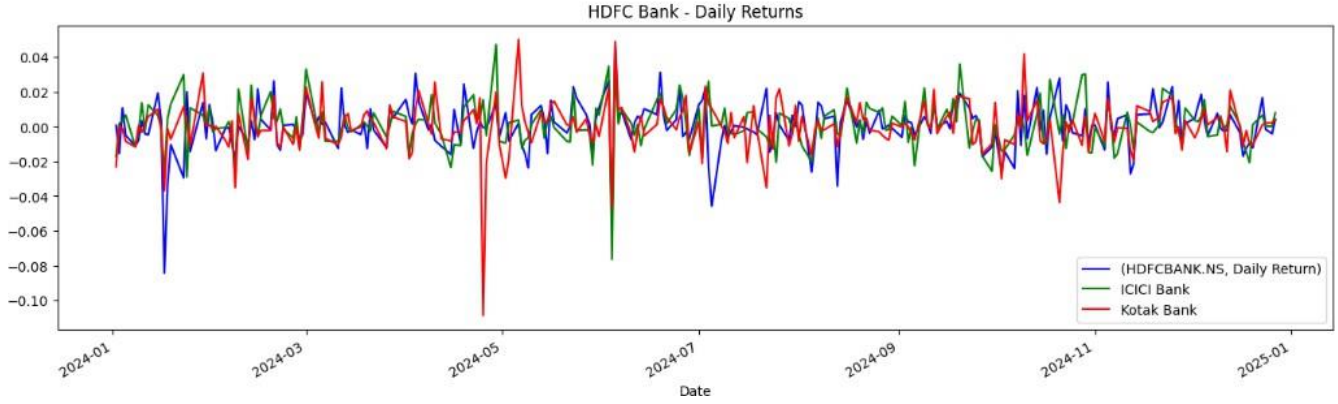
Output

```
Sample data:
          USN     Name  Marks
0  1BM22CS226    Alice     25
1  1BM22CS300      Bob     30
2  1BM22CS400  Charlie     35
3  1BM22CS500    David     40
4  1BM22CS600      Tom     50
```

```
     Product  Quantity  Price  Sales Region
0     Laptop         5   1000   5000  North
1      Mouse        15     20    300   west
2   keyboard        10     50    500   East
3    Monitor         8    200   1600  south
4     Laptop        12    950  11400  north
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8 entries, 0 to 7
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Product   8 non-null      object
 1   Quantity  8 non-null      int64
 2   Price     8 non-null      int64
 3   Sales     8 non-null      int64
 4   Region    8 non-null      object
dtypes: int64(3), object(2)
memory usage: 452.0+ bytes
None
```

```
YF.download() has changed argument auto_adjust default to True
[**********************100%**********************]  3 of 3 completed
First 5 rows of the dataset:
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

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
```

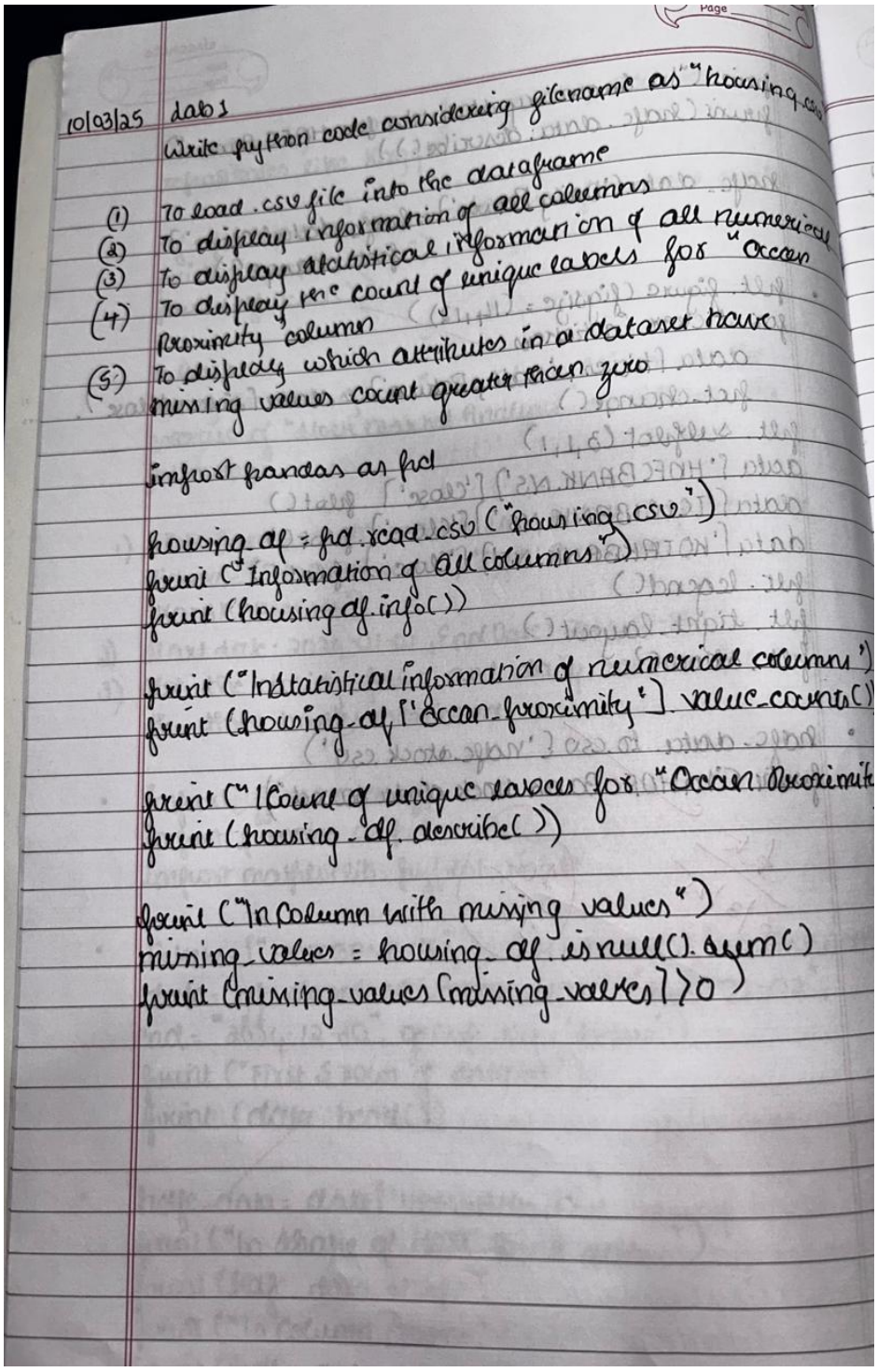

HDFC Bank - Closing Price
1800
1600
1400
1200
1000
2024-01
2024-03
2024-05
2024-07
2024-09
2024-11
2025-01
Date
Close
ICICI Bank
Kotak Bank


HDFC Bank - Daily Returns
0.04
0.02
0.00
−0.02
−0.04
−0.06
−0.08
−0.10
2024-01
2024-03
2024-05
2024-07
2024-09
2024-11
2025-01
Date
(HDFCBANK.NS, Daily Return)
ICICI Bank
Kotak Bank

Program 2

Demonstrate various data pre-processing techniques for a given dataset

Screenshot



For both the datasets 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("Diabetes.csv")
adult_income_df = pd.read_csv("Adult.csv")

print("missing values in diabetes dataset")
print(diabetes_df.isnull().sum())

print("missing values in adult income dataset")
print(adult_income_df.isnull().sum())

ncd = diabetes_df.select_dtypes(include=np.number).columns
nca = adult_income_df.select_dtypes(include=np.number).columns

diabetes_df[numeric_cols_diabetes] = diabetes_df[numeric_cols_diabetes].fillna(diabetes_df[ncd].median())
adult_income_df[numeric_cols_adult] = adult_income_df[numeric_cols_adult].fillna(adult_income_df[nca].median())
```

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

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

diabetes_df = pd.read_csv("Diabetes.csv")
print(diabetes_df.head())
```

```
bins = [0, 18.5,
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("Diabetes after encoding:")
print(diabetes_df.head())
```

3) What is the difference b/w min max scaling and standardization? When would you use one over the other?

Min-Max scaling vs Standardization

- Transforms: scales data to a fixed range usually [0, 1]
- $X_{scaled} = \frac{X - X_{min}}{X_{max} - X_{min}}$
- Best for models sensitive to feature ranges
- Sensitive to outliers

Standardization

- Transforms: Centers data to mean 0 and standard deviation 1
- $X_{standardized} = \frac{X - \mu}{\sigma}$
- Best for models assuming normal distribution
- Less sensitive to outliers

when to use:

- Min-max scaling: use when you need data between a fixed range
- Standardization: use for models that assume normal distribution or when your data has outliers

Sam
10.03.2025

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(housing_df['ocean_proximity'].value_counts())

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
# Changed pd.np.number to np.number
```

```
numeric_cols_adult = adult_income_df.select_dtypes(include=np.number).columns
# Changed pd.np.number to np.number

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

# Load the dataset

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

# Check the first few rows of the dataset

print("Diabetes dataset (first few rows):")

print(diabetes_df.head())

# Step 1: Categorize the 'BMI' column into bins (Low, Normal, High)

bins = [0, 18.5, 24.9, 40]  # BMI categories: Low (<18.5), Normal
(18.5-24.9), High (>24.9)

labels = ['Low', 'Normal', 'High']

diabetes_df['BMI_Category'] = pd.cut(diabetes_df['BMI'], bins=bins,
labels=labels, right=False)

# Step 2: One-Hot Encoding for 'BMI_Category'

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

# Step 3: Encoding 'Gender' column (if present)

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())
```

```
import pandas as pd

from sklearn.preprocessing import LabelEncoder

# Load the dataset

adult_df = pd.read_csv("/content/adult.csv")

# Check the first few rows of the dataset

print("Adult Income dataset (first few rows):")

print(adult_df.head())

# Step 1: Categorize the 'age' column into bins (Young, Middle-Aged, Old)

bins = [0, 25, 50, 100]  # Age categories: Young (0-25), Middle-Aged (25-50),
Old (50-100)

labels = ['Young', 'Middle-Aged', 'Old']

adult_df['Age_Category'] = pd.cut(adult_df['age'], bins=bins, labels=labels,
right=False)

adult_df = pd.get_dummies(adult_df, columns=['Age_Category'])

if 'sex' in adult_df.columns:

    label_encoder = LabelEncoder()

    adult_df['sex'] = label_encoder.fit_transform(adult_df['sex'])

print("\nAdult Income dataset after encoding:")
```

```
print(adult_df.head())
```

Output :

```
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
```

```
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

Missing values in Adult Income dataset:
age                0
workclass          0
fnlwgt             0
education          0
educational-num    0
marital-status     0
occupation         0
relationship       0
race               0
gender             0
capital-gain       0
capital-loss       0
hours-per-week     0
native-country     0
income             0
```

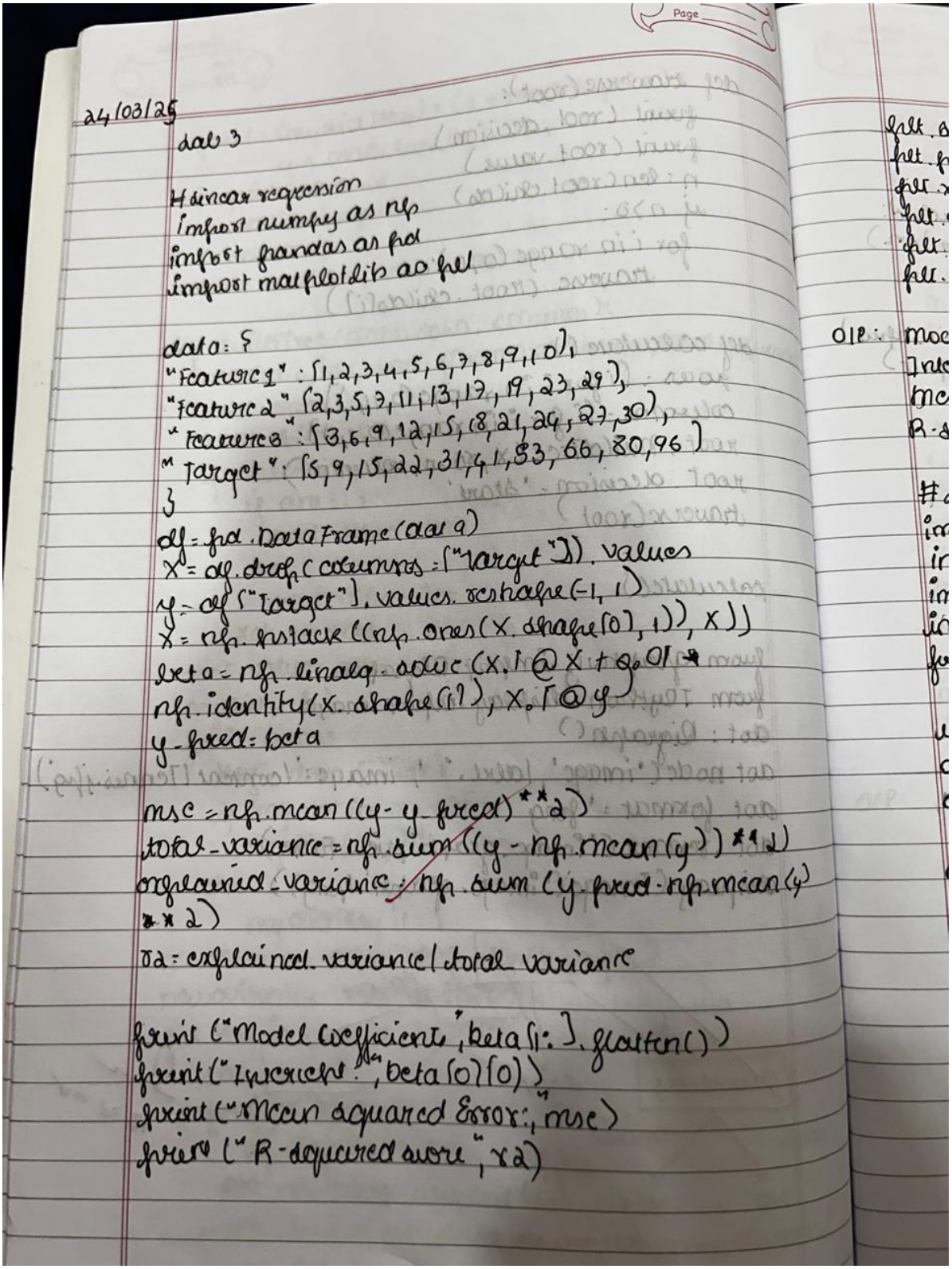
```
Diabetes dataset (first few rows):
    ID  No_Pation Gender  AGE  Urea  Cr  HbA1c  Chol   TG  HDL  LDL  VLDL  \
0  502      17975      F   50   4.7  46    4.9   4.2  0.9  2.4  1.4   0.5
1  735      34221      M   26   4.5  62    4.9   3.7  1.4  1.1  2.1   0.6
2  420      47975      F   50   4.7 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

```
Adult Income dataset (first few rows):
   age  workclass  fnlwgt     education  educational-num      marital-status
0   25    Private  226802          11th                7       Never-married
1   38    Private   89814       HS-grad                9  Married-civ-spouse
2   28  Local-gov  336951    Assoc-acdm               12  Married-civ-spouse
3   44    Private  160323  Some-college               10  Married-civ-spouse
4   18          ?  103497  Some-college               10       Never-married

          occupation relationship   race  gender  capital-gain  capital-loss
0  Machine-op-inspct    Own-child  Black    Male             0             0
1    Farming-fishing      Husband  White    Male             0             0
2    Protective-serv      Husband  White    Male             0             0
3  Machine-op-inspct      Husband  Black    Male          7688             0
4                  ?    Own-child  White  Female             0             0

   hours-per-week native-country income
0              40  United-States  <=50K
1              50  United-States  <=50K
2              40  United-States   >50K
3              40  United-States   >50K
4              30  United-States  <=50K

Adult Income dataset after encoding:
   age  workclass  fnlwgt     education  educational-num      marital-status
0   25    Private  226802          11th                7       Never-married
1   38    Private   89814       HS-grad                9  Married-civ-spouse
2   28  Local-gov  336951    Assoc-acdm               12  Married-civ-spouse
3   44    Private  160323  Some-college               10  Married-civ-spouse
4   18          ?  103497  Some-college               10       Never-married
```

Program 3

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Screenshot:



```
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()
```

```
O/P: Model Coefficients: [0.09026032, 3.31855093, 0.1207809]
Intercept: -3.159950949 2879
Mean squared error: 5.362912814290877
R-squared score: 0.9935261949 15287
```

```
# 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://www.geeksforgeeks.com"
data = pd.read_csv(url)
data
```

```
data = data.dropna()
train_input = np.array(data.x[0:5000]).reshape(500,1)
train_output = np.array(data.y[500:7000]).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 cost_function(self, predictions, train_output):
        cost = np.mean((train_output - prediction) ** 2)
        return cost
    def update(frame):
        predictions = self.forward_propogation(train_input)
        cost = self.cost_function(predictions, train_output)
        line.set_ydata(self.parameters['m'] *
        x_vals + self.parameters['c'])
        return line
ani = FuncAnimation(fig, update, frames = iters,
interval = 200, blit = True)
plt.xlabel('Input')
plt.ylabel('Output')
plt.title('Linear Regression')
plt.legend()
plt.show()

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

O/P:

```
Iteration = 1, Loss = 9232.536559
Iteration = 1, Loss = 1131.59129
Iteration = 1, Loss = 143.2881
```

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

# Drop the missing values

data = data.dropna()

# training dataset and labels

train_input = np.array(data.x[0:500]).reshape(500, 1)

train_output = np.array(data.y[0:500]).reshape(500, 1)

# valid dataset and labels

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/n * mean of (predictions-actual) * input

        dm = 2 * np.mean(np.multiply(train_input, df))

        # dc = 2/n * mean of (predictions-actual)

        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):

        # Initialize random parameters

        self.parameters['m'] = np.random.uniform(0, 1) * -1

        self.parameters['c'] = np.random.uniform(0, 1) * -1

        # Initialize loss

        self.loss = []

        # Initialize figure and axis for animation

        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,

        # Create animation

        ani = FuncAnimation(fig, update, frames=iters, interval=200, blit=True)

        # Save the animation as a video file (e.g., MP4)

        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
```

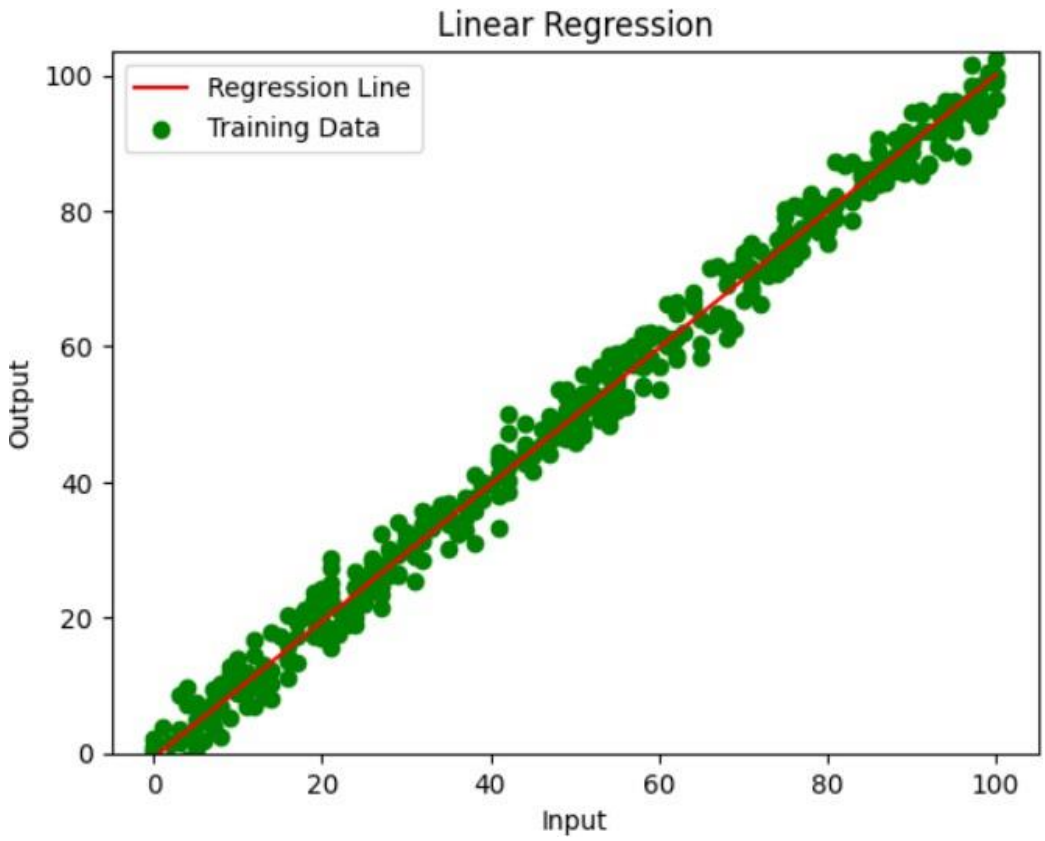
```
#Example usage

linear_reg = LinearRegression()

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

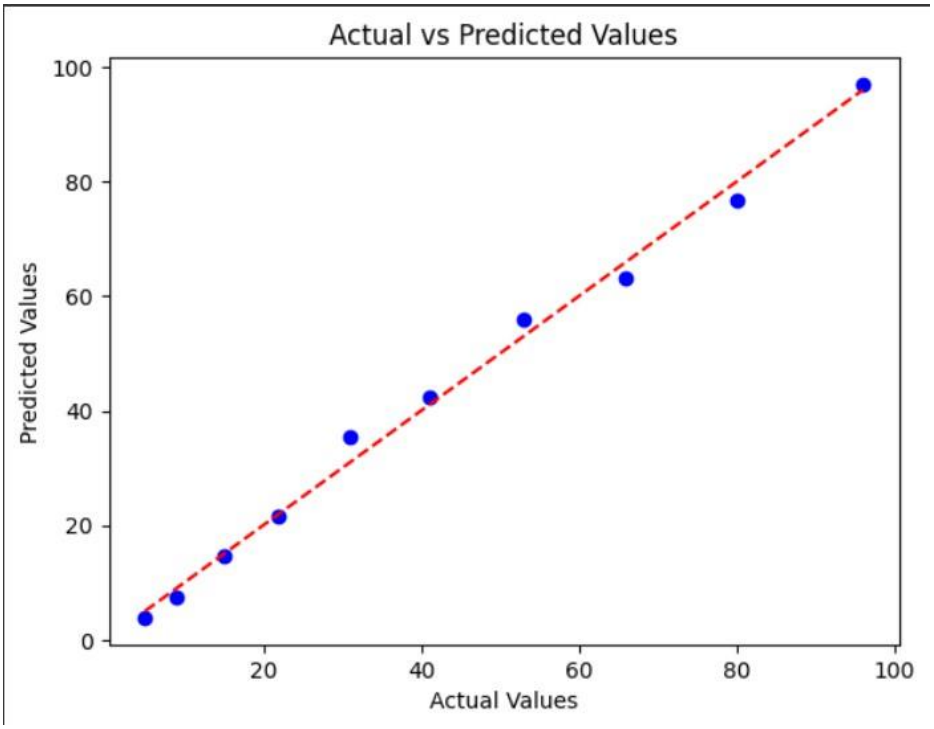
```
#multiple linear regression

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt
```

```
# Sample dataset

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)

# Add intercept column (bias term)

X = np.hstack((np.ones((X.shape[0], 1)), X))

# Compute the coefficients using the Normal Equation

beta = np.linalg.solve(X.T @ X + 0.01 * np.identity(X.shape[1]), X.T @ y)

# Make predictions

y_pred = X @ beta
```

```
# Evaluate the model

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

# Display results

print("Model Coefficients:", beta[1:].flatten())

print("Intercept:", beta[0][0])

print("Mean Squared Error:", mse)

print("R-squared Score:", r2)

# Plot actual vs predicted values

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 :

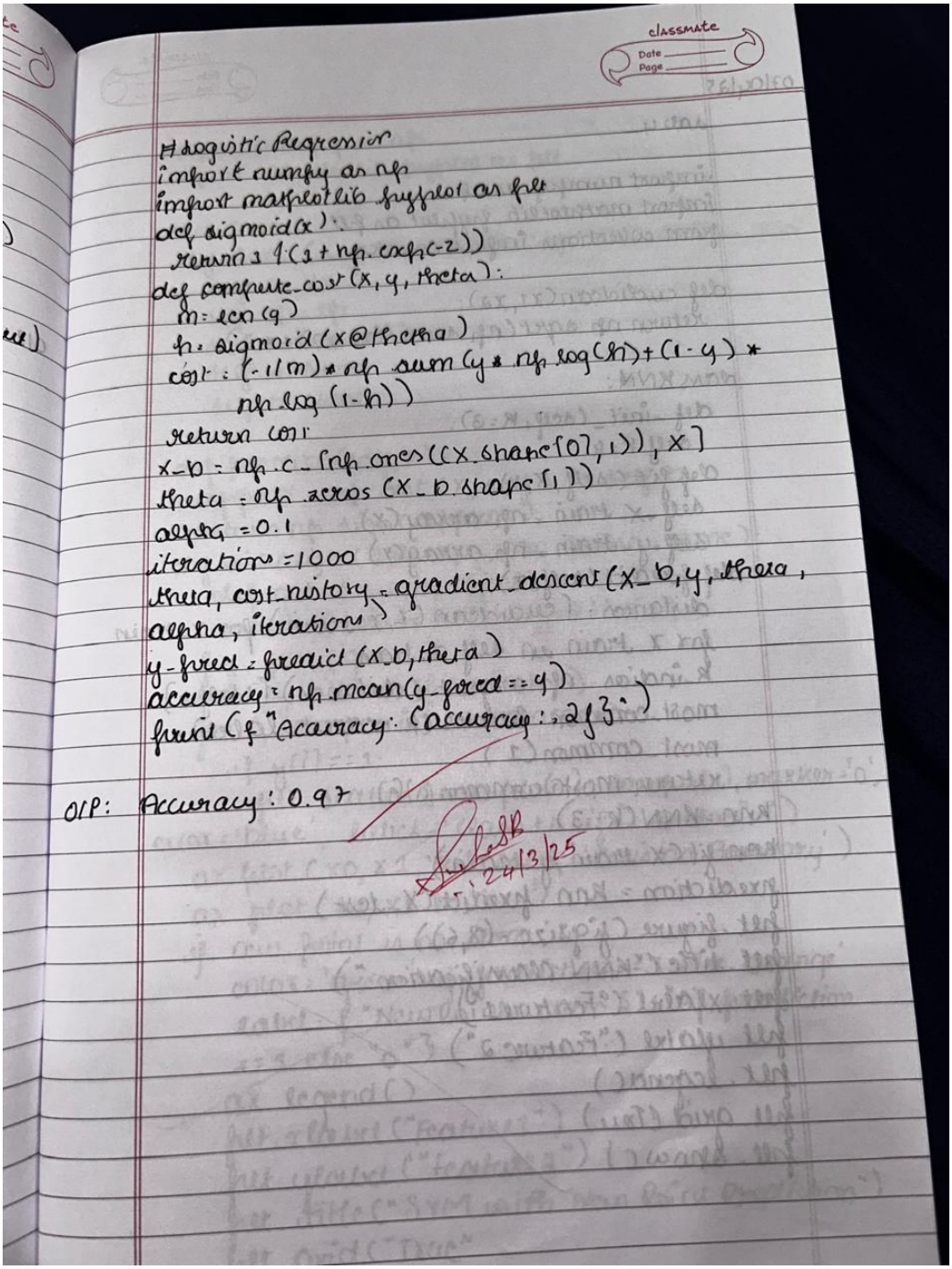

Linear Regression
Regression Line
Training Data
Output
Input
0
20
40
60
80
100


Actual vs Predicted Values
Predicted Values
Actual Values
0
20
40
60
80
100

Program 4

Build Logistic Regression Model for a given dataset

Screenshot:



```
# Logistic Regression
import numpy as np
import matplotlib.pyplot as plt
def sigmoid(x):
    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
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("Accuracy: {accuracy:.2f}")
```

O/P: Accuracy: 0.97

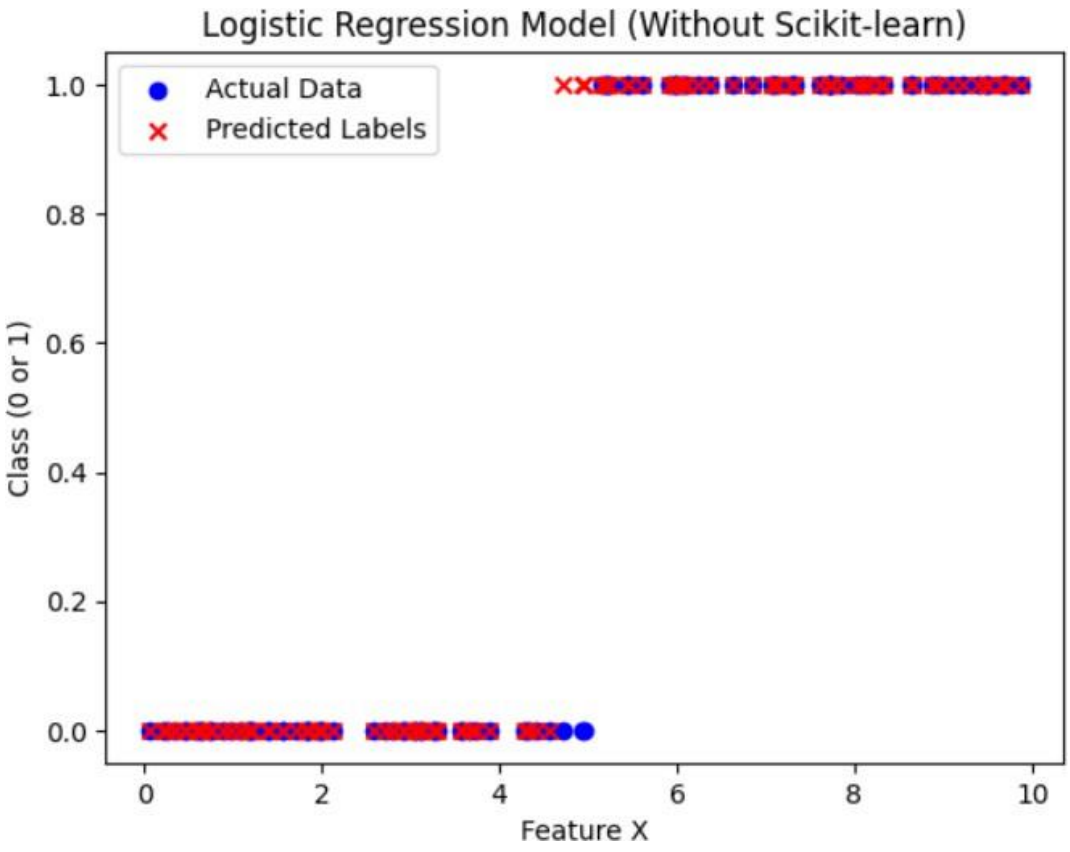
24/3/25

Code:

```
#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  # Feature values between 0 and 10
y = (X > 5).astype(int).ravel()  # Label: 1 if X > 5, else 0
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()
```

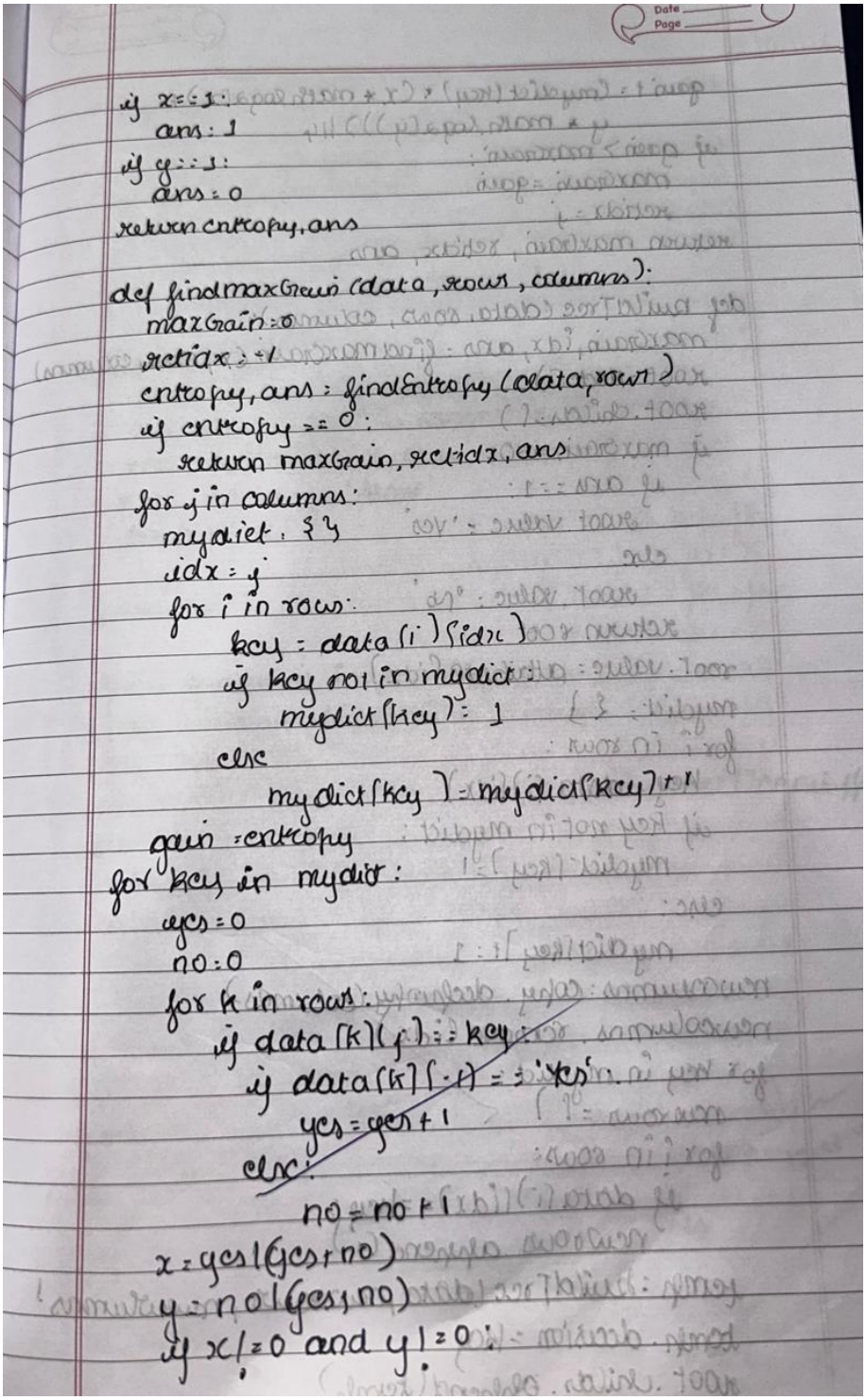
Output



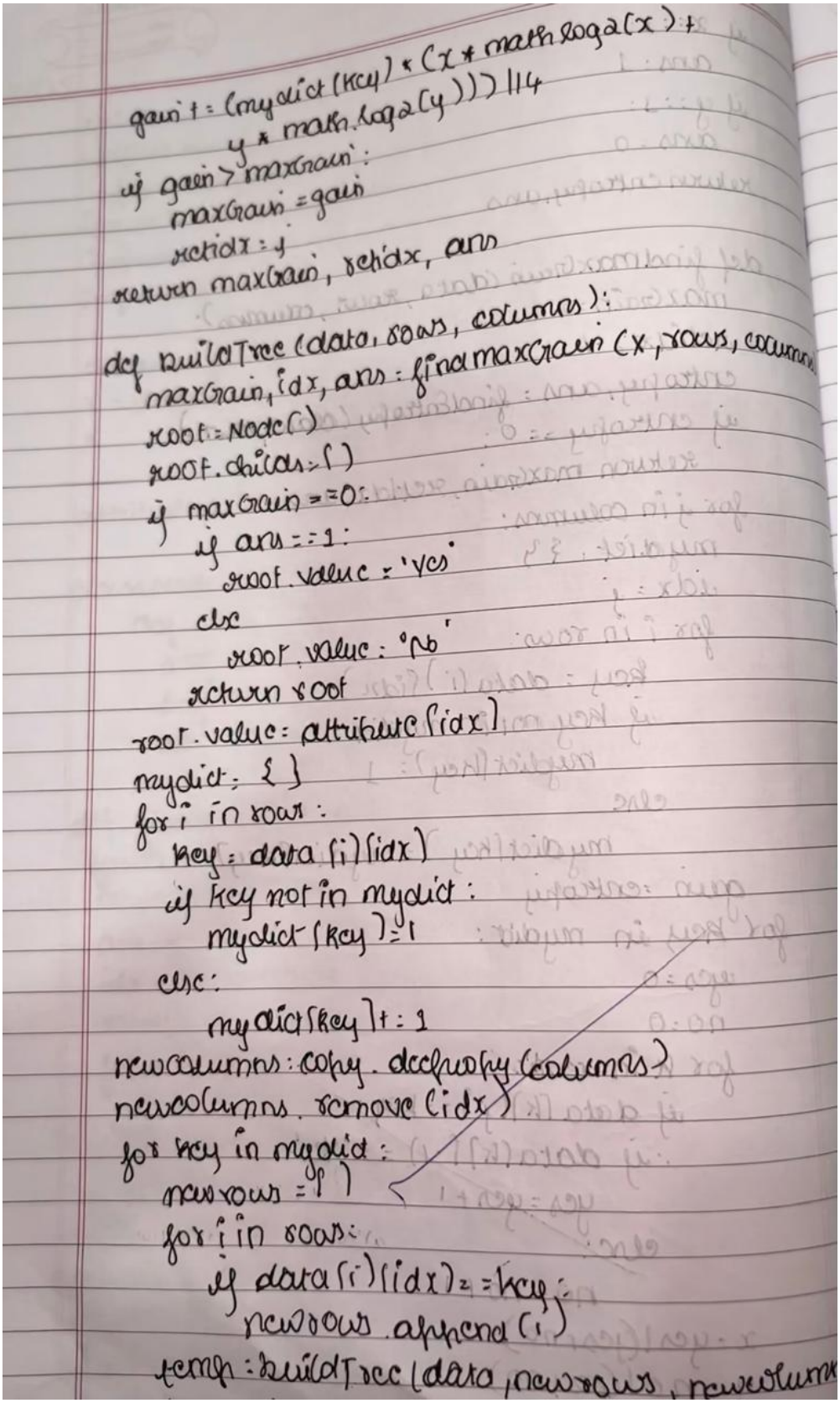
Program 5

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

Screenshot:



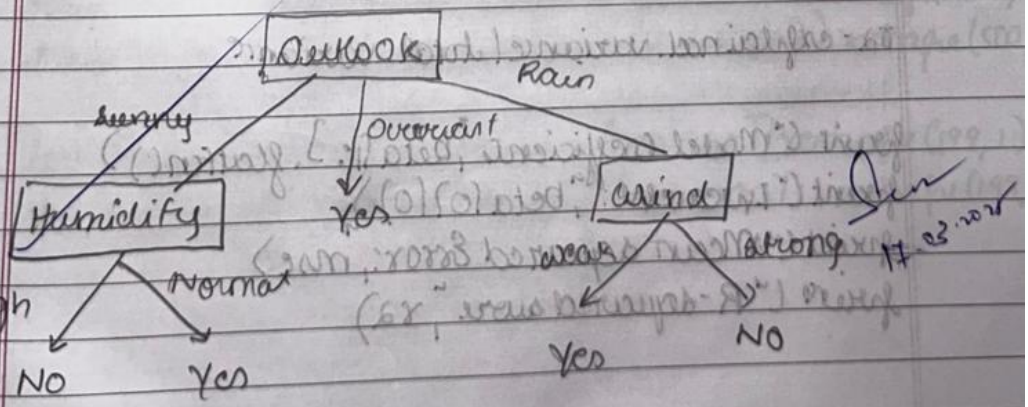
```
if x==1:
    ans=1
if y==1:
    ans=0
return entropy,ans

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 mydir:
        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=yes1(yes+no)
        y=no1(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 = i
return maxGain, retidx, ans

def buildTree(data, rows, columns):
    maxGain, idx, ans = findmaxGain(x, rows, columns)
    root = Node()
    root.child = []
    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
```

```
gain = (mydict[key] * (x * math.log2(x) +
        y * math.log2(y))) / 14
if gain > maxGain:
    maxGain = gain
    rchidx = j
return maxGain, rchidx, ans

def buildTree(data, rows, columns):
    maxGain, idx, ans = findmaxGain(x, rows, columns)
    root = Node()
    root.child = {}
    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
```

```
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\Tennis.jpg')
dot.format = 'png'
dot.render('image_graph', view=False)
display(Image('image_graph.png'))
```



17.03.2021

Code:

```
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
```

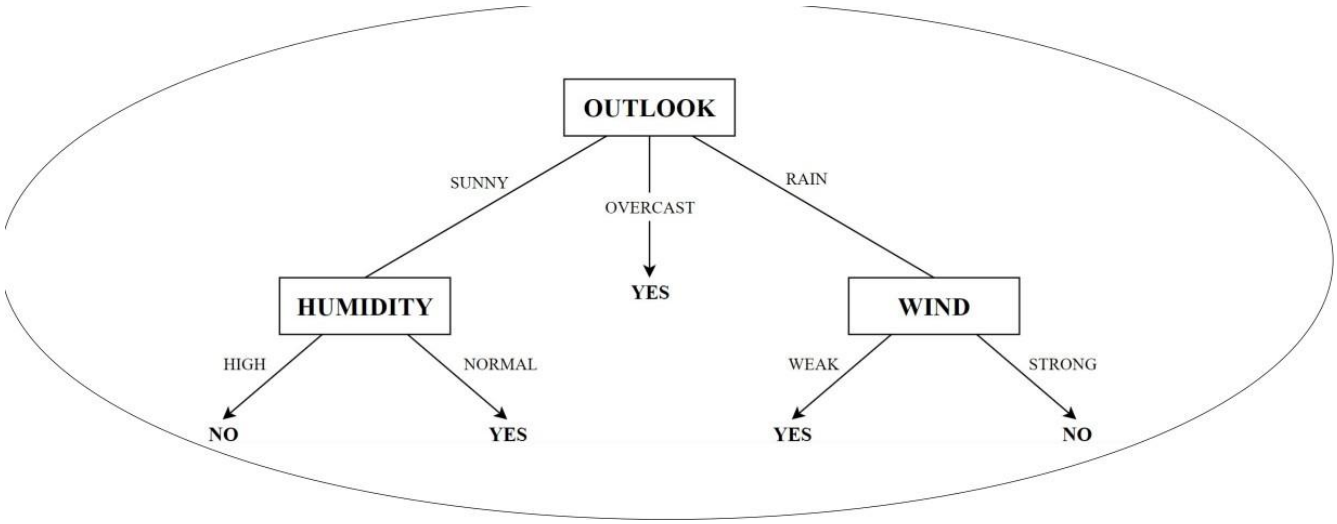
```
def findMaxGain(data, rows, columns):
    maxGain = 0
    retidx = -1
    entropy, ans = findEntropy(data, rows)
    if entropy == 0:
        return maxGain, retidx, ans
    for j in columns:
        mydict = {}
        idx = j
        for i in rows:
            key = data[i][idx]
            if key not in mydict:
                mydict[key] = 1
```

```
            else:
                mydict[key] = mydict[key] + 1
        gain = entropy
        # print(mydict)
        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
            # print(yes, no)
            x = yes/(yes+no)
            y = no/(yes+no)
            # print(x, y)
            if x != 0 and y != 0:
                gain += (mydict[key] * (x*math.log2(x) + y*math.log2(y)))/14
        # print(gain)
        if gain > maxGain:
            # print("hello")
            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)
        # print(newrows)
        temp = buildTree(data, newrows, newcolumns)
        temp.decision = key
        root.childs.append(temp)
    return root
```

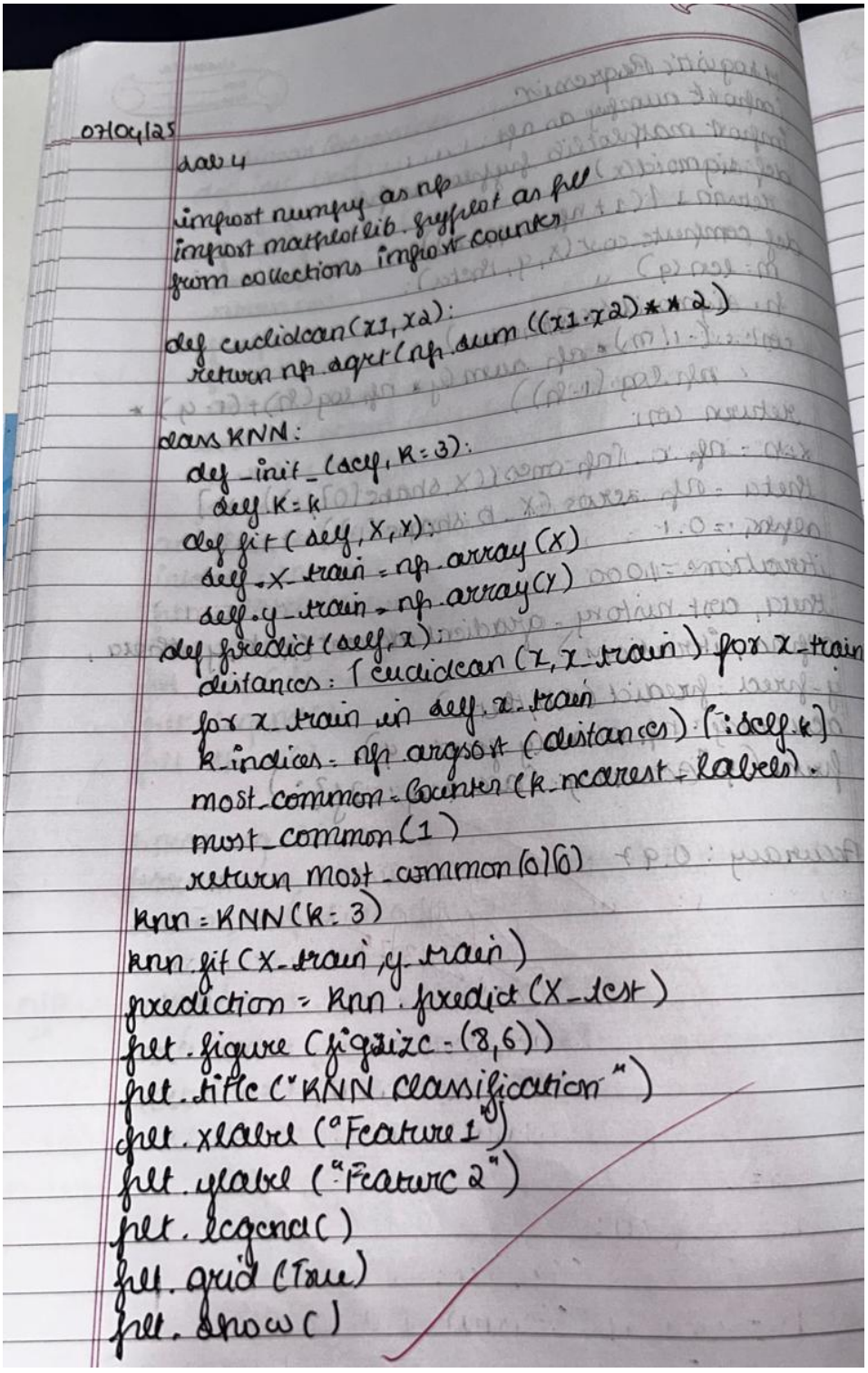
Output:



Program 6

Build KNN Classification model for a given dataset

Screenshot:



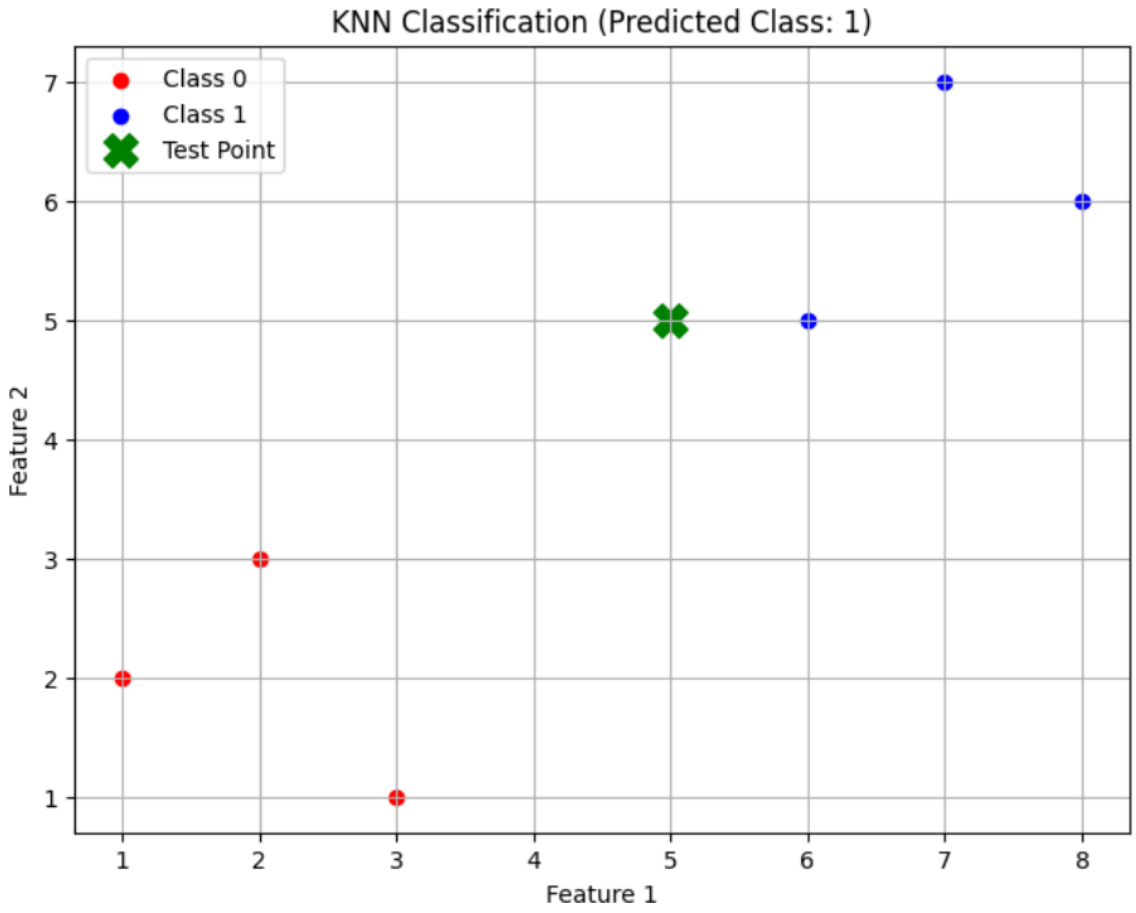
```
07/04/25
Lab 4

import numpy as np
import matplotlib.pyplot as plt
from collections import Counter

def euclidean(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):
        distances = [euclidean(x, x_train) for x_train
        for x_train in self.X_train]
        k_indices = np.argsort(distances)[:self.k]
        most_common = Counter(k_nearest_labels).
        most_common(1)
        return most_common[0][0]
knn = KNN(K=3)
knn.fit(X_train, y_train)
prediction = knn.predict(X_test)
plt.figure(figsize=(8,6))
plt.title("KNN classification")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
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):
    # This line should be indented to be part of the function
    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)
```

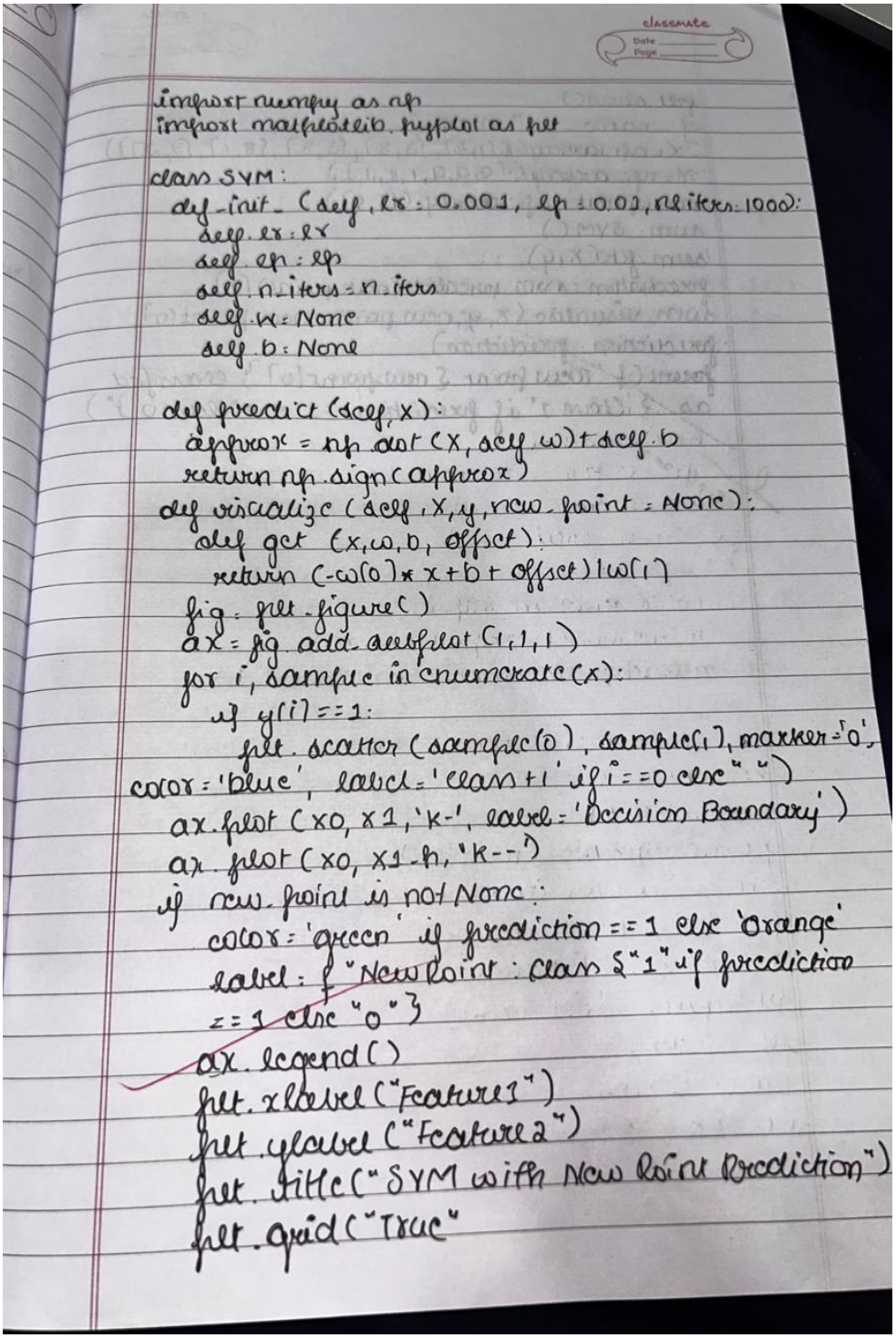
Output:



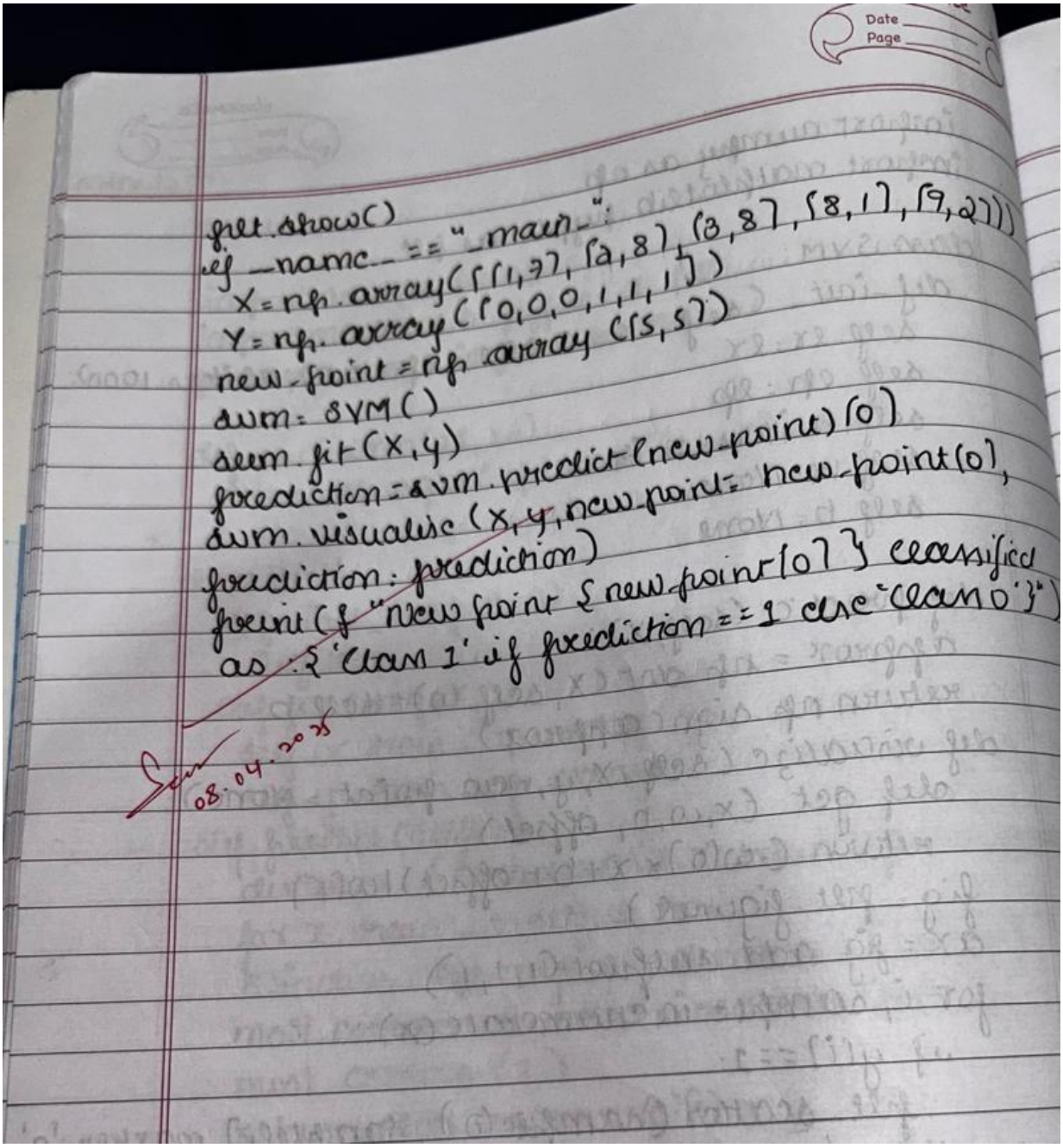
Program 7

Build Support vector machine model for a given dataset

Screenshot:



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

class SVM:
    def __init__(self, lr=0.001, lp=0.01, n_iters=1000):
        self.lr = lr
        self.lp = lp
        self.n_iters = n_iters
        self.w = None
        self.b = None

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

    def visualize(self, X, y, new_point=None):
        def get(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 "")
        ax.plot(x0, x1, 'k-', label='Decision Boundary')
        ax.plot(x0, x1-h, '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"}
        ax.legend()
        plt.xlabel("Feature1")
        plt.ylabel("Feature2")
        plt.title("SVM with New Point Prediction")
        plt.grid("True"
```



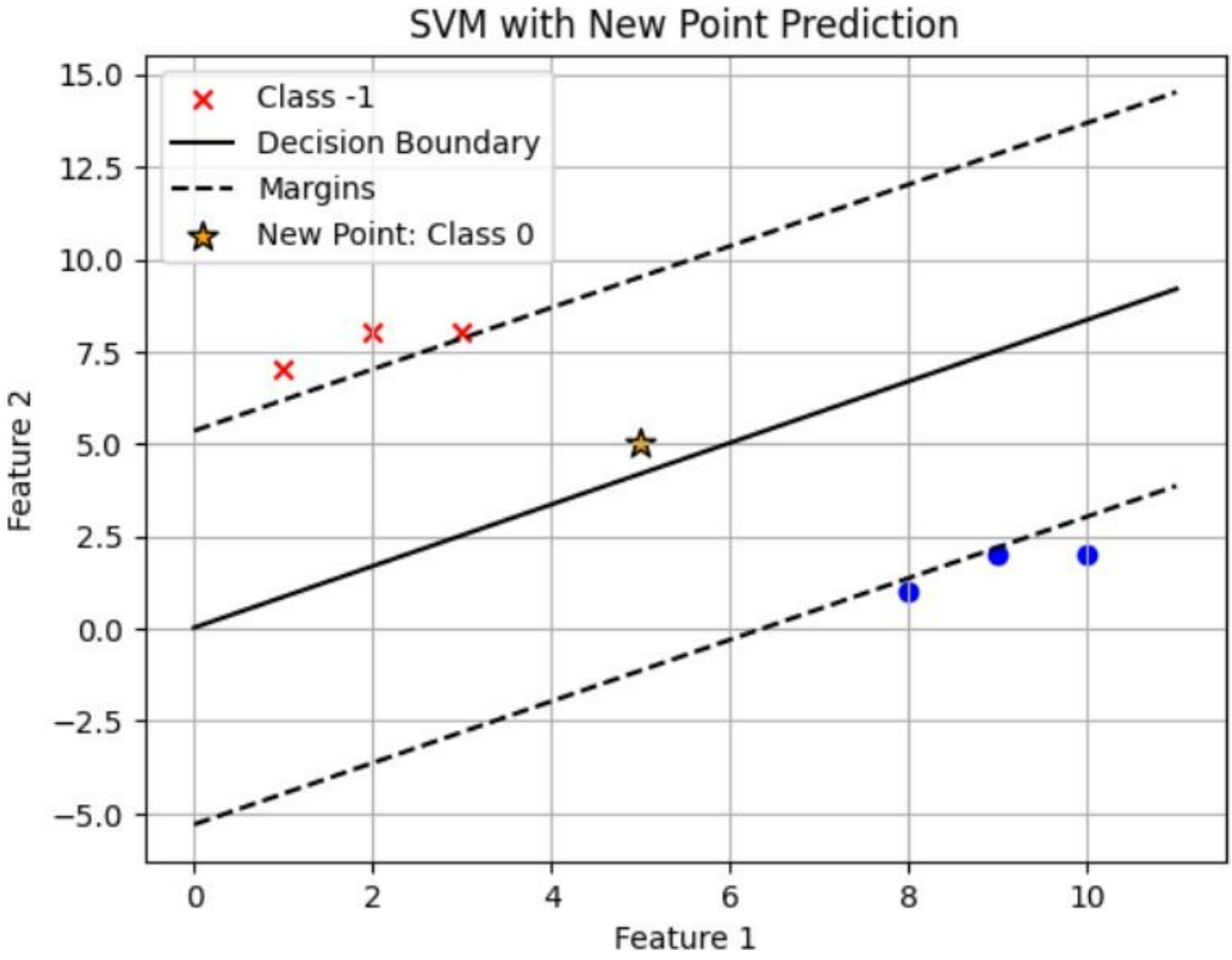
Code:

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

class SVM:
    def __init__(self, learning_rate=0.001, lambda_param=0.01, n_iters=1000):
        self.lr = learning_rate
        self.lambda_param = lambda_param
        self.n_iters = n_iters
```

```
        self.w = None
        self.b = None

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

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

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

        fig = plt.figure()
        ax = fig.add_subplot(1, 1, 1)
        for i, sample in enumerate(X):
            if y[i] == 1:
                    plt.scatter(sample[0], sample[1], marker='o', color='blue',
label='Class +1' if i == 0 else "")
            else:
                     plt.scatter(sample[0], sample[1], marker='x', color='red',
label='Class -1' if i == 0 else "")
        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'}")
```

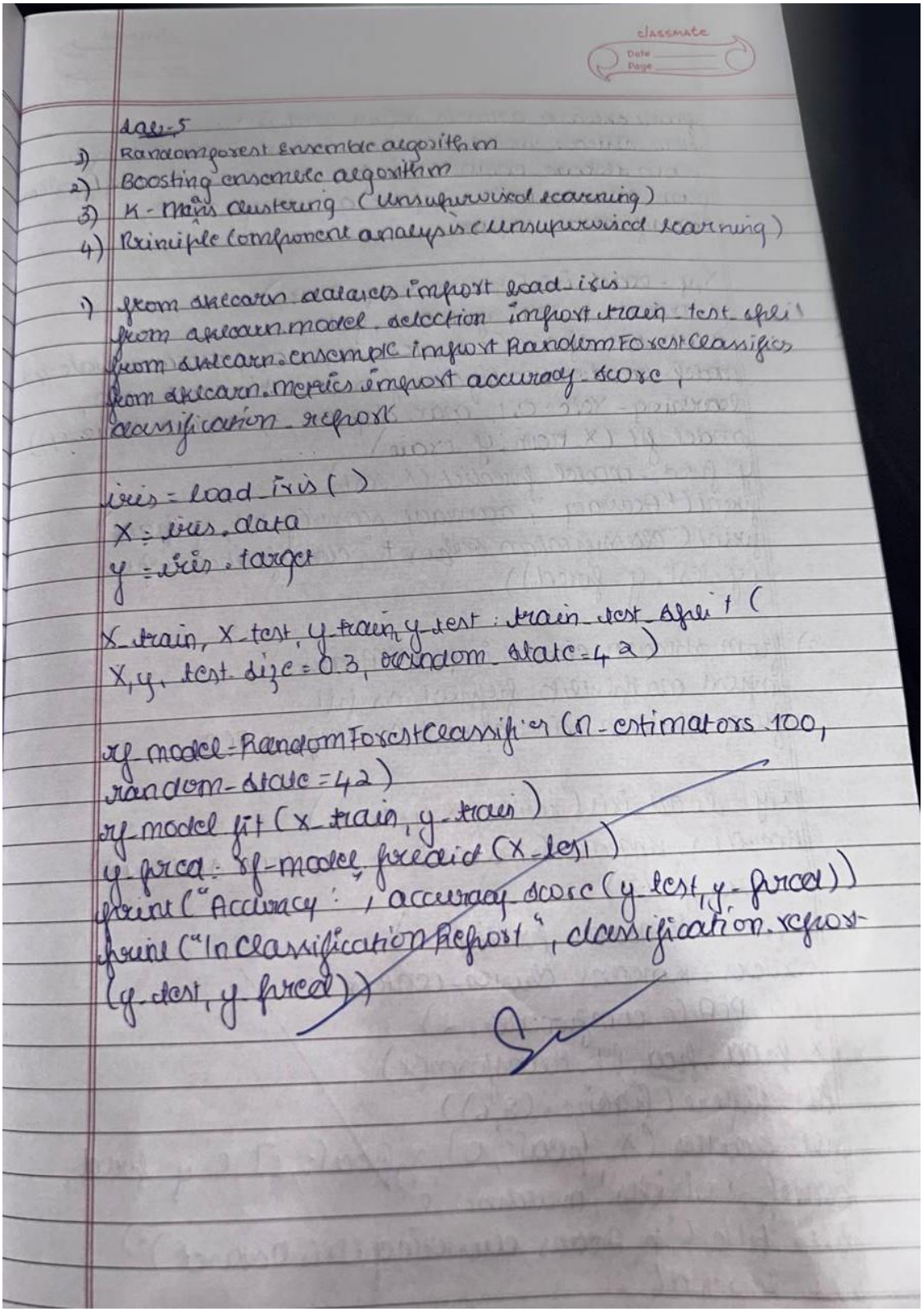
Output :


SVM with New Point Prediction
Class -1
Decision Boundary
Margins
New Point: Class 0
Feature 2
Feature 1
15.0
12.5
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
0
2
4
6
8
10

Program 8

Implement Random forest ensemble method on a given dataset

Screenshot:



Day-5

1) Randomforest ensemble algorithm
2) Boosting ensemble algorithm
3) K-means clustering (Unsupervised learning)
4) Principle Component analysis (Unsupervised learning)

1) from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

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

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, 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("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report", classification_report(y_test, y_pred))

Code:

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
# Load the Iris dataset
iris = load_iris()
X = iris.data      # Features
y = iris.target    # Labels
X_train, X_test, y train, y test = train test split(X, y, test size=0.3,
random_state=42)
# Create the Random Forest model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
# Train the model
rf_model.fit(X_train, y_train)
y_pred = rf_model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

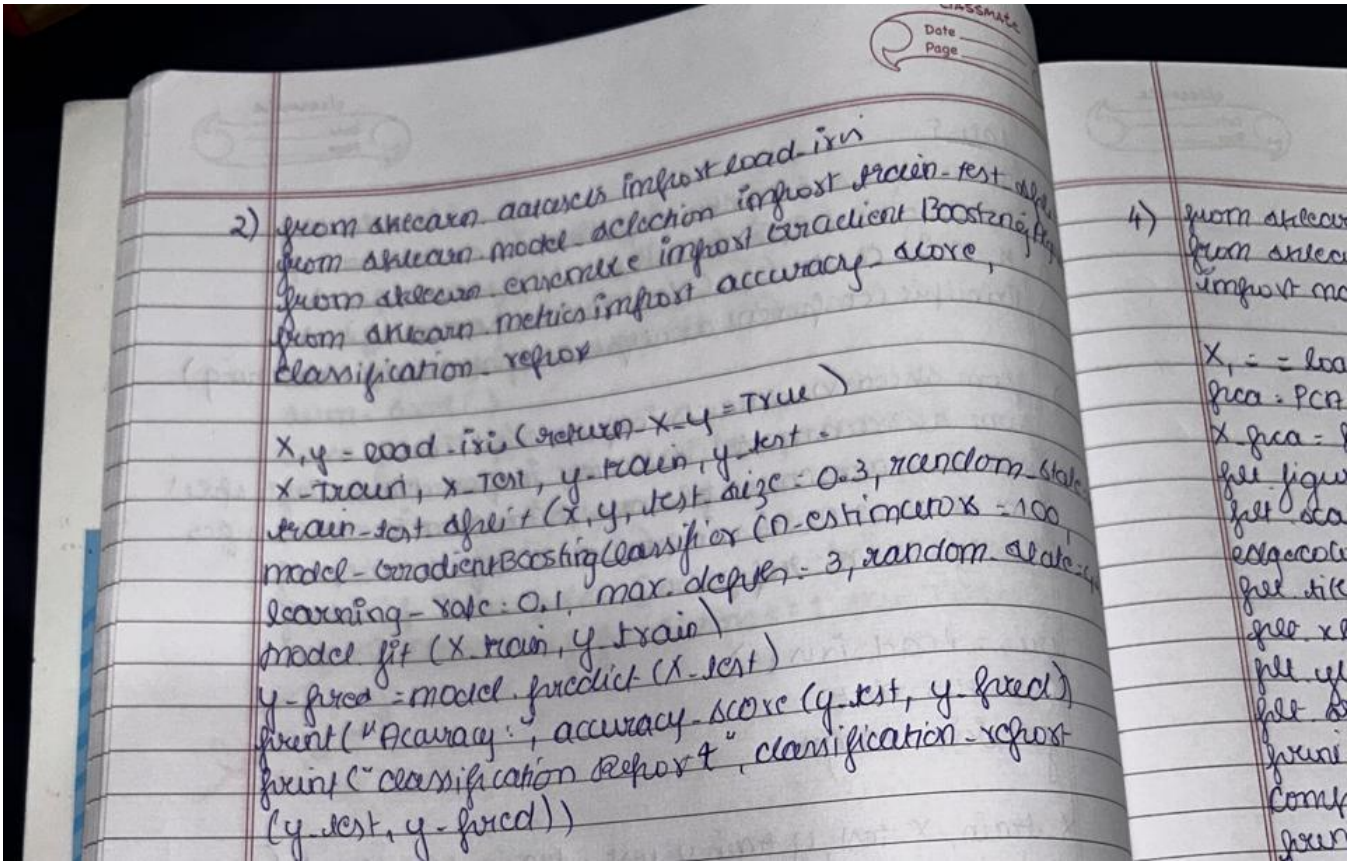
Output:

```
Accuracy: 1.0

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        13
           2       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
```

Program 9

Implement Boosting ensemble method on a given dataset

Screenshot:



Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score, classification_report
url =
"https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabe
tes.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))
```

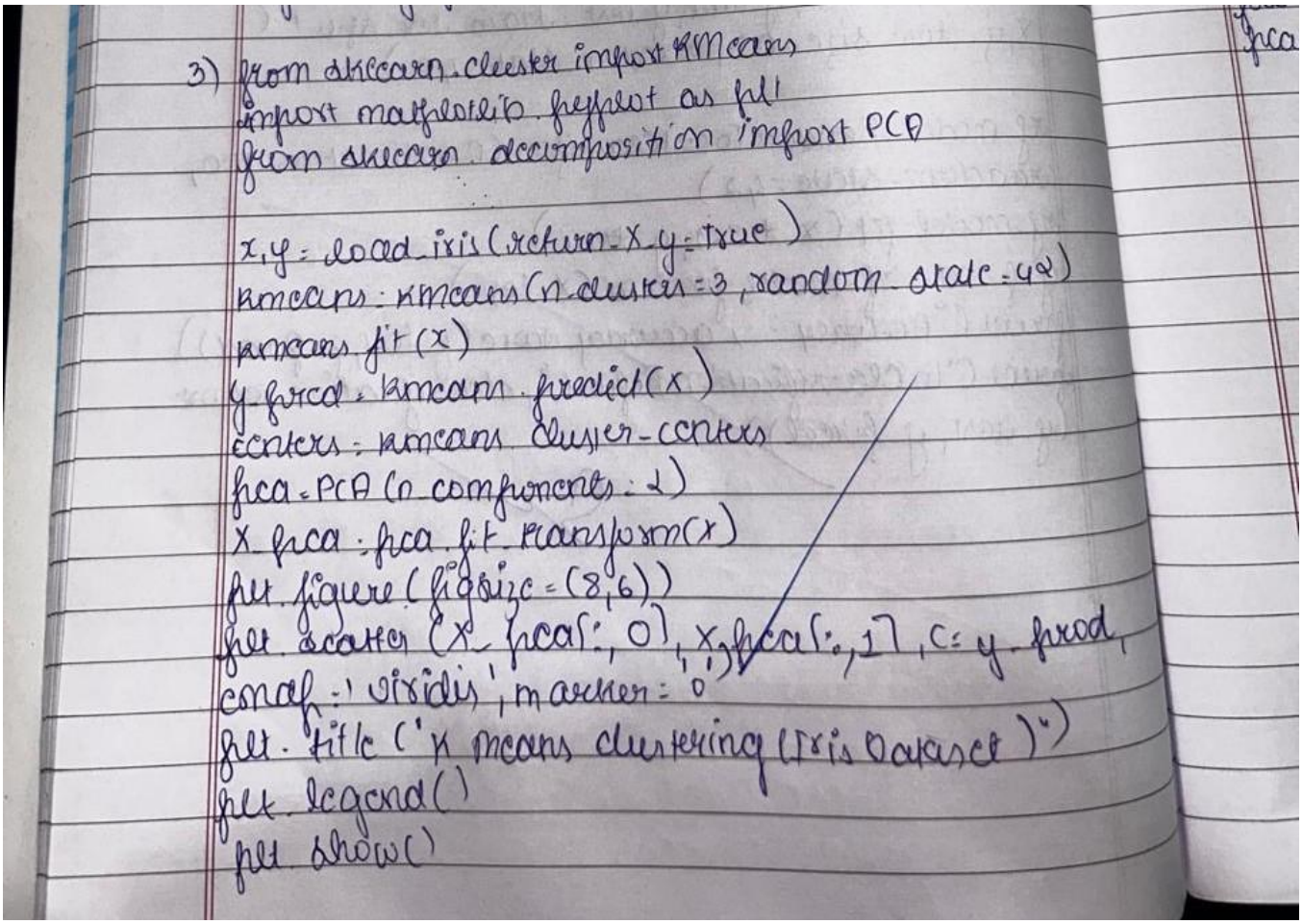
Output:

```
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:

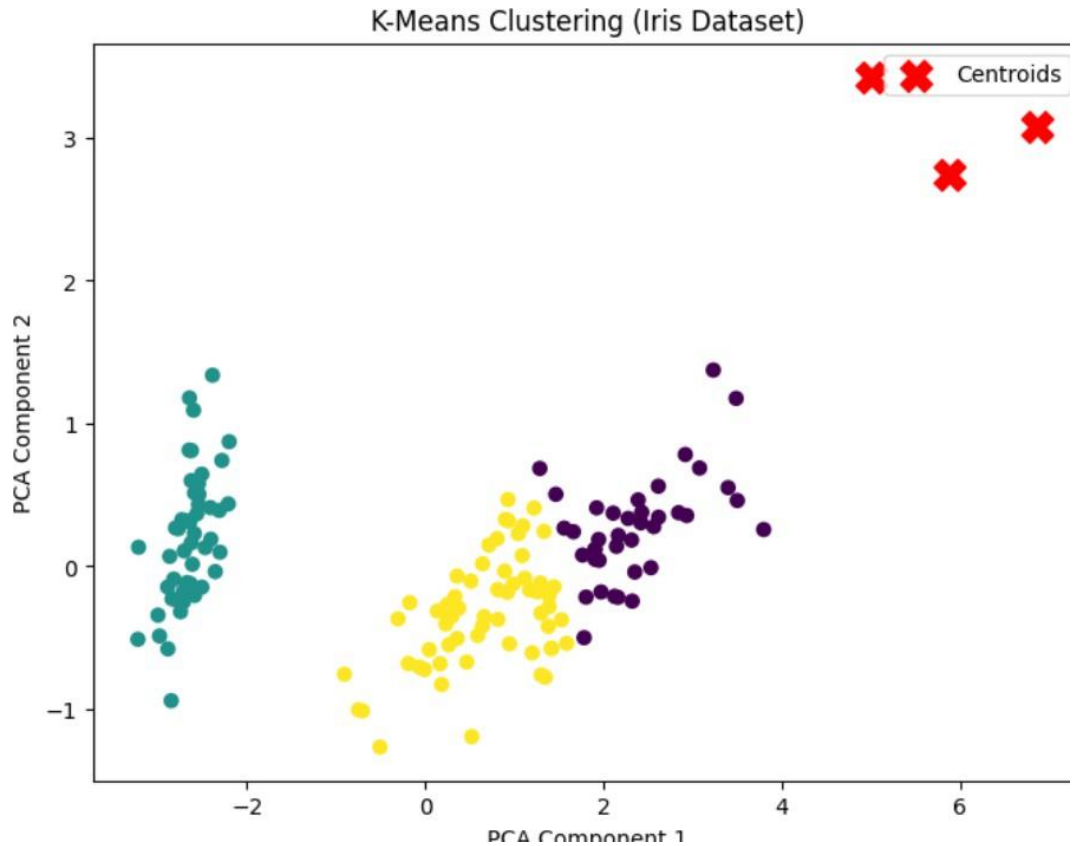


Code:

```
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
X, y = load_iris(return_X_y=True)
# Initialize the KMeans model with the number of clusters (e.g., 3 for Iris
dataset)
kmeans = KMeans(n_clusters=3, random_state=42)
# Fit the model to the data
kmeans.fit(X)
y_pred = kmeans.predict(X)
```

```
centers = kmeans.cluster_centers_
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)
plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y_pred, cmap='viridis', marker='o')
plt.scatter(centers[:,   0],   centers[:,   1],   c='red',   marker='X',   s=200,
label="Centroids")
plt.title("K-Means Clustering (Iris Dataset)")
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.legend()
plt.show()
print("Cluster Centers:\n", centers)
```

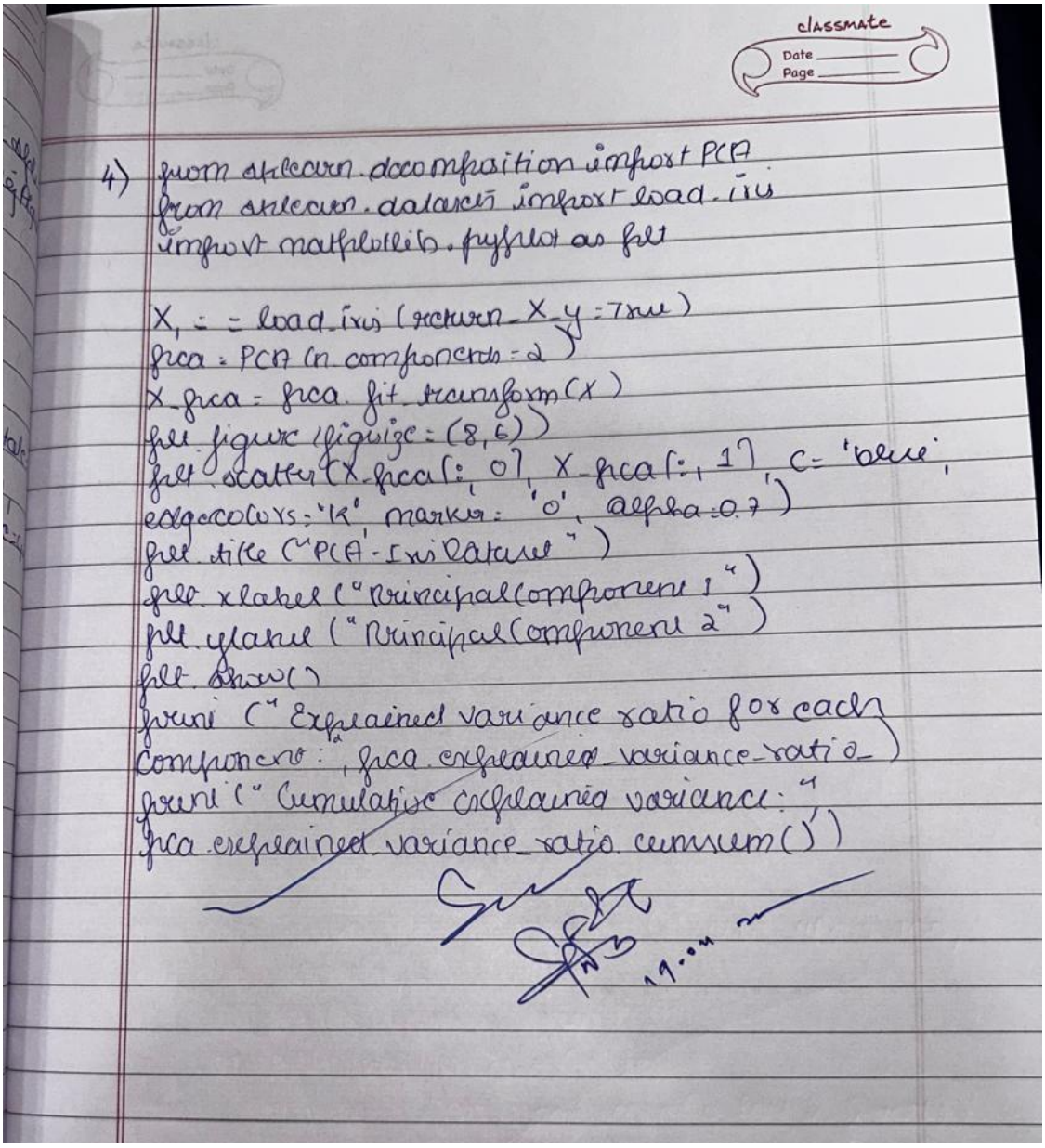
Output:



Program 11

Implement Dimensionality reduction using Principal Component Analysis (PCA) method

Screenshot:



Code:

```
from sklearn.decomposition import PCA
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
X, _ = load_iris(return_X_y=True)
pca = PCA(n_components=2)
```

```
X_pca = pca.fit_transform(X)
plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c='blue', edgecolors='k', marker='o',
alpha=0.7)
plt.title("PCA - Iris Dataset (Unsupervised Learning)")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.grid(True)
plt.show()
print("Explained variance ratio for each component:",
pca.explained_variance_ratio_)
print("Cumulative explained variance:", pca.explained_variance_ratio_.cumsum())
```

Output: