VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

AVANI.A(1BM22CS059)

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
Feb-2025 to June-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 **Avani.A(1BM22CS059),** 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.

Megha J Assistant Professor Department of CSE, BMSCE	Dr. Kavitha Sooda Professor & HOD Department of CSE, BMSCE

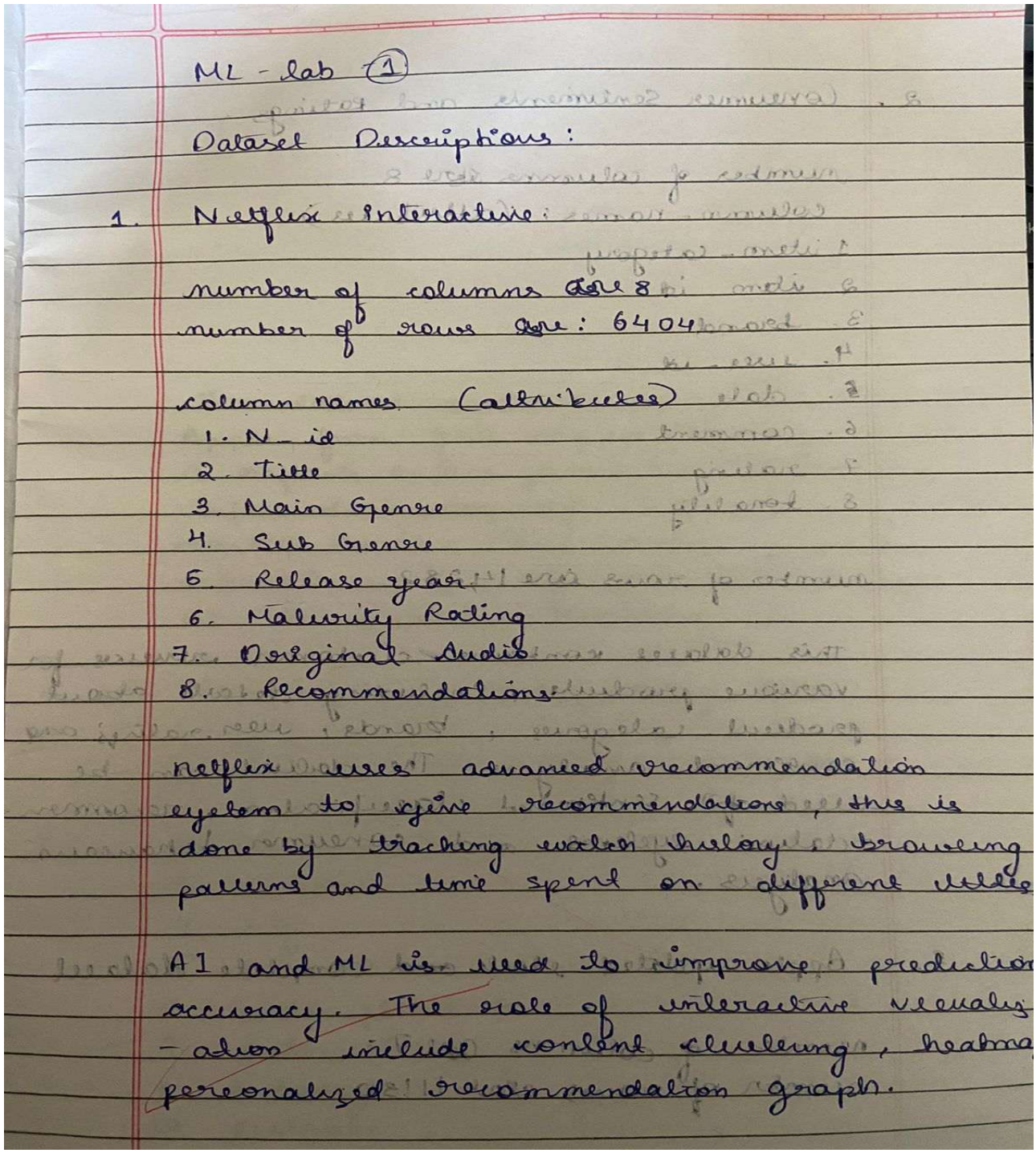
Index

Github Link: https://github.com/Avani-A/ML_LAB_CS059

Program 1

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

Screenshot



2. Consumer Sentiments and Ratings.

number of columns are 8
column-names: (attributes)

1. item_category
2. item_id
3. brand
4. user_id
5. date
6. comment
7. rating
8. tonality

number of rows are 14,282

This dataset contains customer reviews for various products, including details about product categories, brands, user ratings and sentiment analysis. The dataset can be used for sentiment classification, recommendation systems and consumer behaviour analysis.

3. Apple updated stock and complete dataset

number of columns are 7
number of rows are 1130

column names (attributes)

1. date
2. open
3. high
4. low
5. close
6. adj close
7. Volume.

This dataset contains daily stock data for Apple from first day of trading to Jan 23 it reflects apples journey as the most influential company. highlighting growth market trends. This dataset provides valuable insights for financial analysis ML & predictive modeling.

4. Food delivery Order History data

number of columns (attributes) : 10

1. Restaurant ID
2. Restaurant name
3. subzone
4. city
5. ~~order ID~~
6. Order placed At
7. order status
8. delivery
9. distance

number of rows is 21322

This dataset helps us to predict delivery times, analyzing customer behaviour, identifying top performing restaurants and pricing strategies.

5. Car Price dataset

number of columns are 10
(Attributes)

1. Brand
2. Model
3. Year
4. Engine_Size
5. ~~fuel_type~~
6. Transmission
7. Mileage
8. Doors
9. Owner_Count
10. Price

number of rows are : 21322

its used to analyze the prices of car for analysing market trends & fluctuat

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())
```

Sample data:

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

```
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())
```

```
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

    targ
      et
0      0
1      0
2      0
3      0
4      0
```

In [ ]:

```
# Load data from a CSV file (replace 'data.csv' with your file path)

file_path = '/content/industry.csv'
# Ensure the file exists in the same directory

df =

pd.read_csv(file_path)

print("Sample data:")
```

```
print(df.head())

print("\n")
```

Sample data:

	Industry
0	Accounting/Finance
1	Advertising/Public Relations
2	Aerospace/Aviation
3	Arts/Entertainment/Publishing
4	Automotive

In [ ]:

```
import pandas as pd
# Reading data from a CSV file
data = {

'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Evangline'],

'USN': ['1BM22CS025', '1BM22CS030', '1BM22CS035', '1BM22CS040',

'1BM22CS045'], 'Marks': [25, 30, 35, 40, 45]

}
df = pd.DataFrame(data)
print("Sample data:")

print(df.head())
```

Sample data:

	Name	USN	Marks
0	Alice	1BM22CS025	25
1	Bob	1BM22CS030	30
2	Charlie	1BM22CS035	35
3	David	1BM22CS040	40
4	Evangline		

1BM22CS045

45 In [ ]:

```
from sklearn.datasets import

load_diabetes dia =

load_diabetes()

df = pd.DataFrame(dia.data,

columns=dia.feature_names) df['target'] =

dia.target

print("Sample data:")
```

```
print(df.head())
```

Sample data:

```
      age     sex     bmi      bp      s1      s2      s3  \
0  0.038076  0.050680  0.061696  0.021872 -0.044223 -0.034821 -0.043401
1 -0.001882 -0.044642 -0.051474 -0.026328 -0.008449 -0.019163  0.074412
2  0.085299  0.050680  0.044451 -0.005670 -0.045599 -0.034194 -0.032356
3 -0.089063 -0.044642 -0.011595 -0.036656  0.012191  0.024991 -0.036038
4  0.005383 -0.044642 -0.036385  0.021872  0.003935  0.015596  0.008142

      s4      s5      s6  target
0 -0.002592  0.019907 -0.017646     151.0
1 -0.039493 -0.068332 -0.092204      75.0
2 -0.002592  0.002861 -0.025930     141.0
3  0.034309  0.022688 -0.009362     206.0
4 -0.002592 -0.031988 -0.046641     135.0
```

In [ ]:

```
# Load data from a CSV file (replace 'data.csv' with your file path)

file_path = '/content/sample_data/california_housing_train.csv' # Ensure the file exists in the same directory

df =

pd.read_csv(file_path)

print("Sample data:")

print(df.head())

print("\n")
```

Sample data:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms
0	-114.31	34.19	15.0	5612.0	1283.0
1	-114.47	34.40	19.0	7650.0	1901.0
2	-114.56	33.69	17.0	720.0	174.0
3	-114.57	33.64	14.0	1501.0	337.0
4	-114.57	33.57	20.0	1454.0	326.0

	population	households	median_income	median_house_value
0	1015.0	472.0	1.4936	66900.0
1	1129.0	463.0	1.8200	80100.0
2	333.0	117.0	1.6509	85700.0
3	515.0	226.0	3.1917	73400.0
4	624.0	262.0	1.9250	65500.0

In [ ]:

 # Load data from a CSV file (replace 'data.csv'
with your file path) # downloading and loading
file_path = '/content/Dataset of Diabetes .csv' *# Ensure the file exists in the same directory*

df =

pd.read_csv(file_path)

print("Sample data:")

print(df.head())

print("\n")

Sample data:

	ID	No_Pation	Gender	AGE	Urea	Cr	HbA1c	Chol	TG	HDL	LDL	VLDL	\
0	17975		F	50	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	
1	34221		M	26	4.5	62	4.9	3.7	1.4	2.1	1.1	0.6	
2	47975		F	50	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	
3	87656		F	50	4.7	46	4.9	4.2	0.9	2.4	1.4	0.5	
4	34223		M	33	7.1	46	4.9	4.9	1.0	2.0	0.8	0.4	

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

In [ ]:

```
import pandas as pd
# Reading data from a CSV file
df =pd.read_csv('/content/sample_data/california_housing_test.csv')
# Displaying the first few rowsof the DataFrame
print(df.head())

# Writing the DataFrame to a
CSV file
df.to_csv('output.csv',index=Fals
e) print("Data saved
tooutput.csv")
```

```
  longitude latitude housing_median_age total_rooms total_bedrooms \
0   -122.05     37.37                27.0      3885.0          661.0
1   -118.30     34.26                43.0      1510.0          310.0
2   -117.81     33.78                27.0      3589.0          507.0
```

```
3   -118.36            28.0     67.0       15.0
              33.82
4   -119.67            19.0   1241.0      244.
              36.33                           0
```

```
  population households median_income median_house_value
0   1537.0  606.0    6.6085     344700.0
1    809.0   277.0  3.5990        176500.
                                      0
2   1484.0  495.0    5.7934     270500.0
3     49.0    11.0    6.1359      330000.
                                      0
4    850.0   237.0  2.9375         81700.
                                      0
```

Data saved

tooutput.csv In [ ]:

```
# Reading sales data from a CSV file
california_df =pd.read_csv('/content/sample_data/california_housing_test.csv')
# Displaying the first fewrows of the dataset
print("First few rows of the
california_housing_test data:")
print(california_df.head())
```

```
First few rows of the california_housing_test data:
  longitude latitude housing_median_age total_rooms total_bedrooms \
0   -          37.37       27.0   3885.0      661.
122.05                                          0
1   -          34.26       43.0   1510.0      310.
118.30                                          0
2   -          33.78       27.0   3589.0      507.
117.81                                          0
3   -          33.82       28.0     67.0       15.0
118.36
4   -          36.33       19.0   1241.0      244.
119.67                                          0
```

```
  population households median_income median_house_value
```

0	1537.0	606.0	6.6085	344700.0
1	809.0	277.0	3.5990	176500.0
2	1484.0	495.0	5.7934	270500.0
3	49.0	11.0	6.1359	330000.0
4	850.0	237.0	2.9375	81700.0

In [ ]:

 # Grouping by Region and calculating total sales

```
california
=california_df.groupby('total_rooms')['total_bedrooms'].sum()
print("\nTotal housing by region:")
print(california)
```

```
Total housing by
region:
total_rooms
6.0      2.0
16.0     4.0
18.0     3.0
```

19.0 19.0

21.0 7.0

...

21988. 4055.
0 0

23915. 4135.
0 0

24121. 4522.
0 0

27870. 5027.
0 0

30450. 5033.
0 0

Name: total_bedrooms, Length: 2215,

dtype: float64 In [ ]:

 # Grouping by Product and calculating total quantity sold

best_selling_homes =

california_df.groupby('housing_median_age')['households'].sum().sort_values(ascending=

False) print("\nBest-selling products by quantity:")

print(best_selling_homes)

Best-selling products by

quantity:

housing_median_age

52.0 64943
.0

17.0 58184
.0

16.0 49321
.0

19.0 47612
.0

35.0 45376
.0

25.0 44133
.0

34.0 42328
.0

26.0 42320
.0

18.0 42040.0
24.0 41335.0
36.0 40843.0
15.0 40482.0
32.0 39534.0
29.0 38879.0
33.0 38627.0
27.0 38492.0
20.0 37554.0
5.0 37454.0
21.0 37112.0
4.0 35466.0
30.0 35027.0
22.0 34291.0
14.0 33256.0
37.0 31574.0
28.0 30872.0
12.0 28560.0
23.0 28165.0
11.0 25067.0

31.0 25032.0
13.0 24657.0
38.0 23639.0
39.0 22211.0
43.0 22042.0
6.0 20872.0
44.0 19610.0
42.0 19163.0
41.0 19140.0
45.0 17695.0
10.0 16622.0
46.0 16571.0
9.0 15913.0
40.0 14746.0
8.0 14511.0
48.0 12280.0
3.0 12250.0
7.0 12015.0
47.0 9384.0
49.0 6696.0
2.0 6085.0
50.0 5701.0
51.0 4037.

```
            0
1.0     17.0
Name: households, dtype:
float64 In [ ]:

# Saving the sales by region data to a CSV file
california.to_csv('california.csv')
# Saving the best-selling products data
to a CSV file
best_selling_homes.to_csv('best_selling
_homes.csv') print("\nAnalysis results
saved to CSV files.")

Analysis results saved to
CSV files. In [ ]:

import yfinance

as yf import

pandas as pd

import

matplotlib.pyplot as plt

In [ ]:

# Step 2: Downloading Stock Market Data
```

 # Define the ticker symbols for Indian companies

 # Example: Reliance Industries (RELIANCE.NS), TCS (TCS.NS), Infosys (INFY.NS)

tickers = ["RELIANCE.NS", "TCS.NS", "INFY.NS"]

 # Fetch historical data for the last 1 year

data = yf.download(tickers, start="2022-10-01",
end="2023-10-01", group_by='ticker')

 # Display the first 5 rows of the

dataset print("First 5 rows of the

dataset:") print(data.head())

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

Ticker	RELIANCE.NS				\
Price	Open	High	Low	Close	Volume
Date					
2022-10-03	1096.071886	1107.736072	1083.009806	1085.988892	11852723
2022-10-04	1098.959251	1108.217280	1095.453061	1106.017334	8948850
2022-10-06	1113.258819	1122.883445	1108.285998	1110.096313	13352162
2022-10-07	1106.681897	1120.087782	1106.681897	1114.794189	7714340
2022-10-10	1102.259136	1108.034009	1094.467737	1102.625854	6329527

Ticker	TCS.NS				\
Price	Open	High	Low	Close	

Volume	Date			
2022-10-03	2894.197635	2919.032606	2873.904430	2884.485840	1763331
2022-10-04	2927.970939	2993.730628	2921.254903	2987.111084	2145875
2022-10-06	3006.293304	3018.855764	2988.367592	2997.547852	1790816
2022-10-07	2993.150777	3000.495078	2955.173685	2961.744629	1939879
2022-10-10	2908.692292	3021.754418	2903.860578	3013.588867	3064063

Ticker	INFY.NS				
Price	Open	High	Low	Close	Volume
Date					
2022-10-03	1337.743240	1337.743240	1313.110574	1320.453003	4943169
2022-10-04	1345.038201	1356.928245	1339.638009	1354.228149	6631341
2022-10-06	1369.007786	1383.029504	1368.155094	1378.624023	6180672
2022-10-07	1370.286797	1381.182015	1364.412900	1374.881714	3994466
2022-10-10	1351.338576	1387.956005	1351.338576	1385.729614	5274677

In [ ]:

 # Step 3: Basic Data

Exploration # Check the

shape of the dataset

print("\nShape of the

dataset:")

print(data.shape)

 # Check column names

print("\nColumn names:")

print(data.columns)

 # Summary statistics for a specific stock

(e.g., Reliance) reliance_data =

data['RELIANCE.NS']

print("\nSummary statistics for Reliance

Industries:")

print(reliance_data.describe())

 # Calculate daily returns
 # Create a copy of the Reliance data to avoid modifying a slice of the original dataframe
reliance_data = data['RELIANCE.NS'].copy()

```
# Now, apply the calculation
reliance_data['Daily Return'] = reliance_data['Close'].pct_change()
```

```
Shape of the dataset:
(247, 15)

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

```
        (   'TCS.NS',   'Low'),
        (   'TCS.NS',  'Close'),
        (   'TCS.NS', 'Volume'),
        (   'INFY.NS',   'Open'),
        (   'INFY.NS',   'High'),
        (   'INFY.NS',   'Low'),
        (   'INFY.NS',  'Close'),
        (   'INFY.NS', 'Volume')],
       names=['Ticker', 'Price'])
```

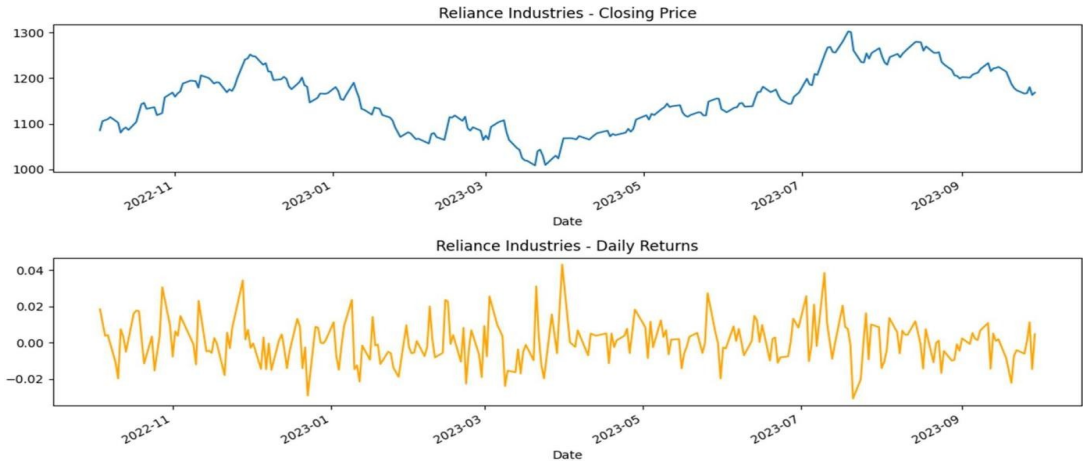
Summary statistics for Reliance Industries:

Price	Open	High	Low	Close	Volume
count	247.000000	247.000000	247.000000	247.000000	2.470000e+02
mean	1155.033899	1163.758985	1144.612976	1154.002433	1.316652e+07
std	65.890843	66.876907	65.755901	66.726021	6.754099e+06
min	1015.178443	1017.470038	999.137216	1008.876526	3.370033e+06
25%	1106.532938	1111.081861	1092.347974	1104.997559	8.717141e+06
50%	1155.424265	1163.078198	1146.716157	1155.240967	1.158959e+07
75%	1202.667031	1209.102783	1193.235594	1201.447937	1.530302e+07
max	1297.045129	1308.961472	1281.920577		

1302.476196 5.708188e+07 In [ ]:

```
# Plot the closing price and daily returns

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()



In [ ]:

 # Step 4: Saving the Processed Data to a

New CSV File # Save the Reliance data

to a CSV file

reliance_data.to_csv('reliance_stock_data.

csv')

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

Reliance stock data saved to
'reliance_stock_data.csv'. In [ ]:

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

```
[**********100%**********
**] 3 of 3 completed First 5 rows of the dataset:
Ticker ICICIBANK.NS                          \
Price        Open    High     Low    Close
Volume Date
2024-01-    983.086778  996.273246  982.541485
  01                    990.869812  7683792
2024-01-    988.490253  989.134730  971.883221
  02                    973.866150  16263825
2024-01-    976.295294  979.567116  966.777197
  03                    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

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

In [ ]:

```
HDFC =
data['HDFCBANK.NS']

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

 # Calculate daily returns

```
# Create a copy of the Reliance data to avoid modifying a slice of the original dataframe
HDFC = data['HDFCBANK.NS'].copy()
# Now, apply the calculation
HDFC['Daily Return'] = HDFC['Close'].pct_change()
```

Summary statistics for HDFC:

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

max 1877.699951 1880.000000 1858.550049
1871.750000 2.226710e+08 In [ ]:

```
ICICI = data['ICICIBANK.NS']

print("\nSummary statistics for

ICICI:") print(ICICI.describe())

# Calculate daily returns
# Create a copy of the Reliance data to avoid modifying a slice of the original dataframe
ICICI = data['ICICIBANK.NS'].copy()
# Now, apply the calculation
ICICI['Daily Return'] = ICICI['Close'].pct_change()
```

Summary statistics for ICICI:
Price Open High Low Close Volume
count 244.000000 244.000000 244.000000 244.000000 2.440000e+02
mean 1161.723560 1173.687900 1151.318979 1162.751791 1.539172e+07
std 104.905646 105.668229 105.083015 105.520481 9.503609e+06
min 965.637027 979.567116 961.869473 971.387512 1.007022e+06
25% 1073.818215 1085.368782 1067.386038 1075.107086 1.014533e+07
50% 1169.443635 1178.450012 1157.361521 1165.470703 1.291768e+07
75% 1248.512512 1261.399994 1236.649963 1250.812531 1.755770e+07
max 1344.900024 1362.349976 1340.050049
1346.099976 7.325777e+07 In [ ]:

```
KOTAKBANK = data['KOTAKBANK.NS']

print("\nSummary statistics for

KOTAKBANK:")

print(KOTAKBANK.describe())
```

```
# Calculate daily returns
# Create a copy of the Reliance data to avoid modifying a slice of the original dataframe
KOTAKBANK = data['KOTAKBANK.NS'].copy()
# Now, apply the calculation
KOTAKBANK['Daily Return'] = KOTAKBANK['Close'].pct_change()
```

Summary statistics for KOTAKBANK:

Price	Open	High	Low	Close	Volume
count	244.000000	244.000000	244.000000	244.000000	2.440000e+02
mean	1771.245907	1787.548029	1754.395105	1770.792347	5.736598e+06
std	62.189675	61.978802	62.765980	62.594747	5.388927e+06
min	1581.266899	1586.161558	1542.159736	1545.006592	1.824890e+05

25% 1733.974927 1754.131905 1719.028421 1736.297058 3.300380e+06
50% 1769.500000 1789.450012 1758.099976 1773.681030 4.307680e+06
75% 1809.925018 1826.998164 1789.912506 1808.155670 6.159475e+06
max 1935.000000 1942.000000 1909.599976
1934.699951 6.617908e+07 In [ ]:

 # Plot the closing price and daily returns

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

plt.subplot(2, 1, 1)

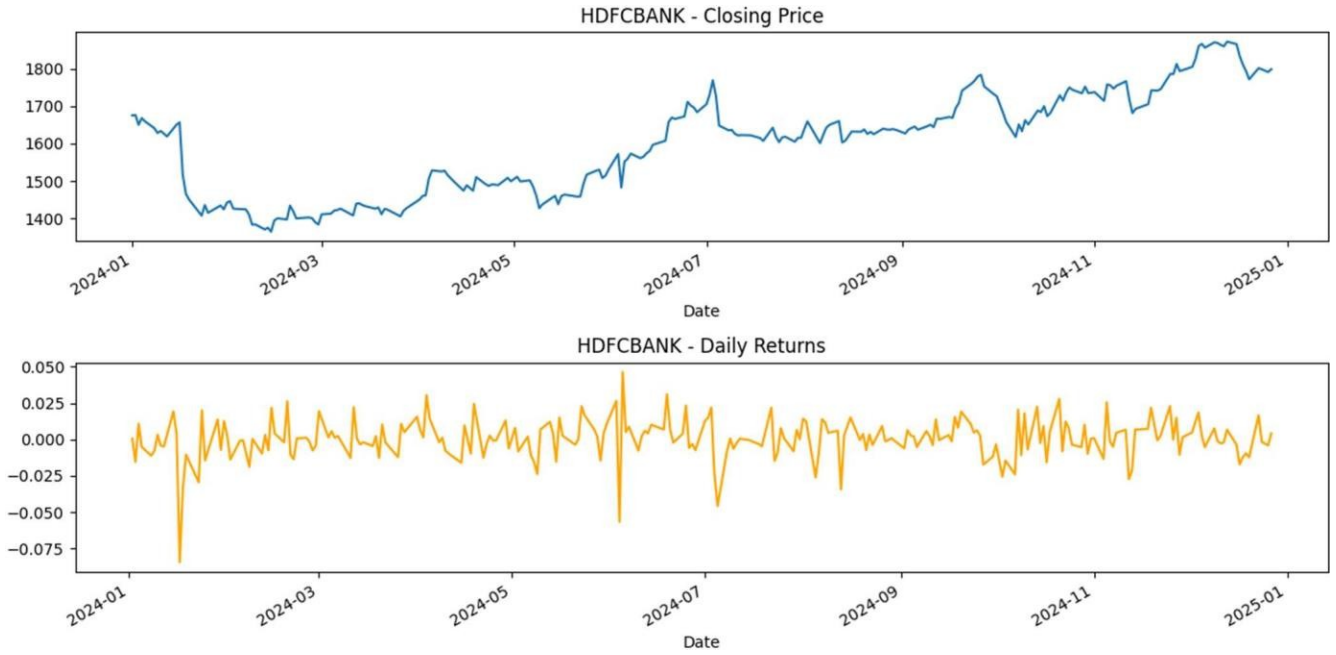
HDFC['Close'].plot(title="HDFCBANK -

Closing Price") plt.subplot(2, 1, 2)

HDFC['Daily Return'].plot(title="HDFCBANK - Daily

Returns", color='orange') plt.tight_layout()

plt.show()



In [ ]:

 # Plot the closing price and daily returns

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

plt.subplot(2, 1, 1)

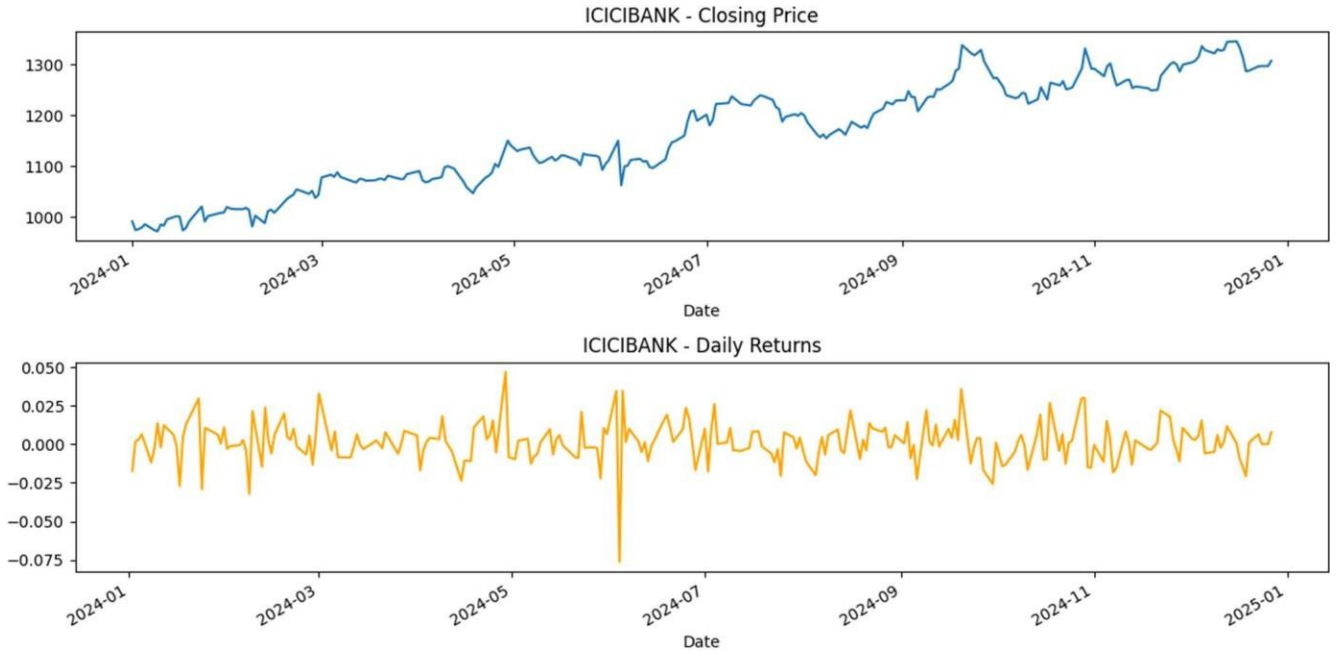
ICICI['Close'].plot(title="ICICIBANK -

Closing Price") plt.subplot(2, 1, 2)

ICICI['Daily Return'].plot(title="ICICIBANK - Daily

Returns", color='orange') plt.tight_layout()

plt.show()



In [ ]:

 # Plot the closing price and daily returns

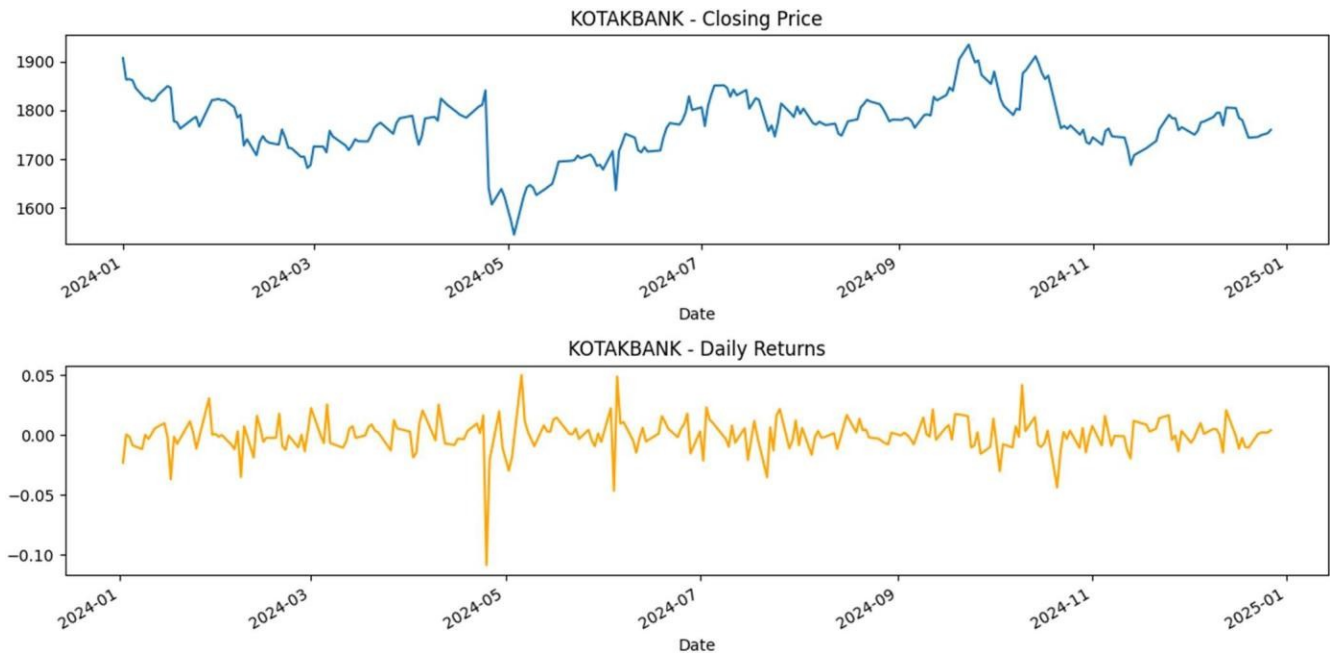
plt.figure(figsize=(12, 6))

plt.subplot(2, 1, 1)

KOTAKBANK['Close'].plot(title="KOTAKBANK

```
- Closing Price") plt.subplot(2, 1, 2)

KOTAKBANK['Daily Return'].plot(title="KOTAKBANK - Daily

Returns", color='orange') plt.tight_layout()
```

plt.show()



In [ ]:

 # Step 4: Saving the Processed Data to a

New CSV File # Save the Reliance data

to a CSV file HDFC.to_csv('HDFC.csv')
ICICI.to_csv('ICICI.csv')
KOTAKBANK.to_csv('KOTAK
BANK.csv')

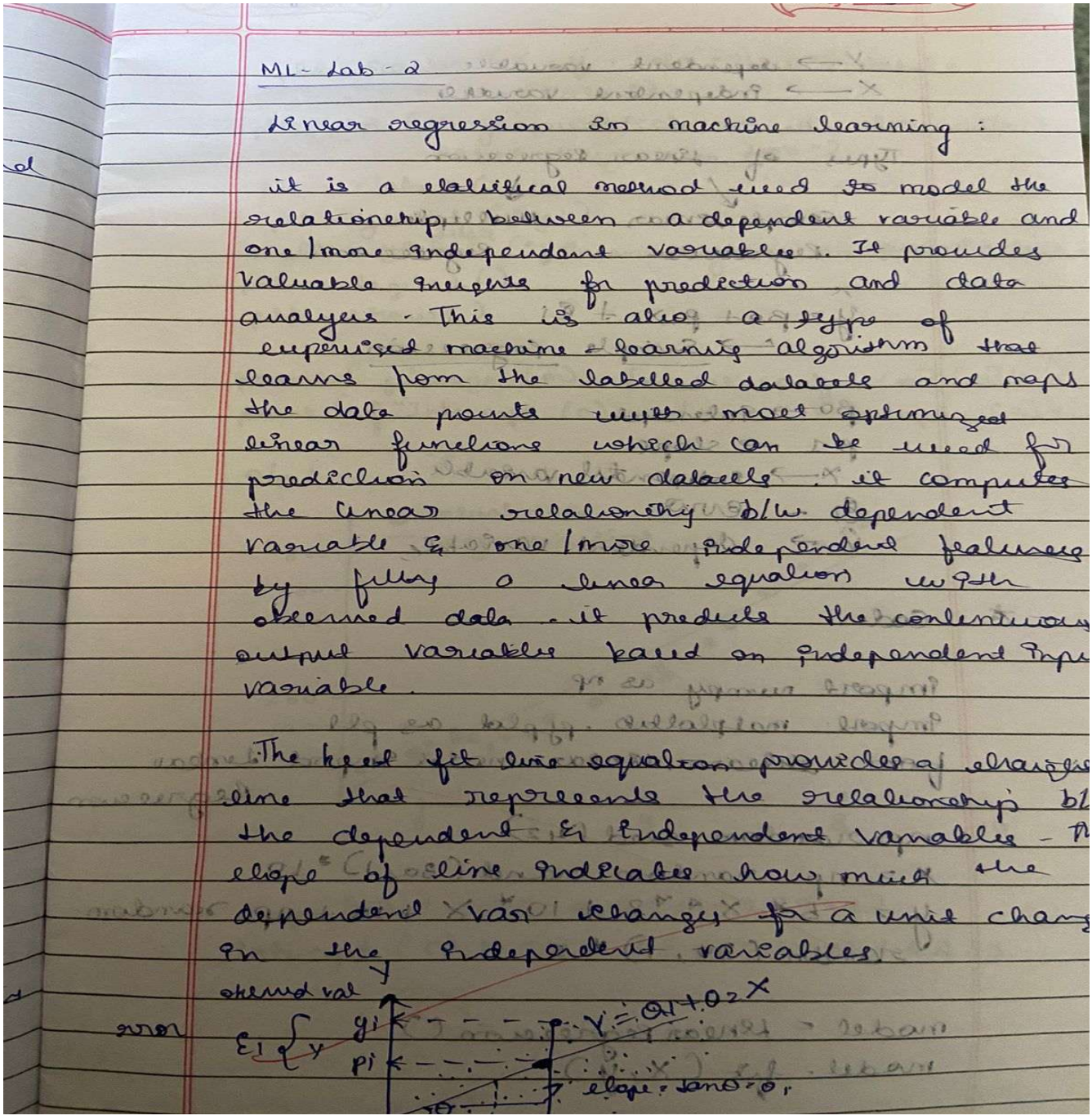
print("\nSAVED")

SAVED

Program 2

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Screenshots



Y → dependent variable
X → Independent variable

Types of linear Regression
- Simple Linear Regression
- Multiple Regression

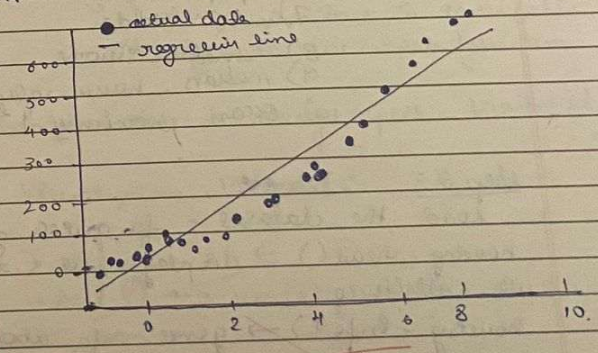
$\hat{y} = \beta_0 + \beta_1 x + \epsilon_i$
→ Simple Regression

β_0 → Intercept
β_1 → slope
x → Independent variable
ϵ_i → error
$\hat{y}$ → dependent variable

★ Code:

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

np.random.seed(42)
X = np.random.rand(30,1)*10
y = 5*X**2 + 10*X + 15 + np.random.randn(30,1)*5

model = LinearRegression()
model.fit(X,y)

y_pred = model.predict(X)

plt.scatter(X, y, color="blue", label="Actual Data")
plt.plot(X, y_pred, color="red", linewidth=2, label="Regression Line")
plt.xlabel("X")
plt.ylabel("y")
plt.title("Linear regression on quadratic data")
plt.legend()
plt.show()

print(f"slope (m): {model.coef_[0][0]}")
print(f"Intercept (c): {model.intercept_[0]}")
```

O/p:



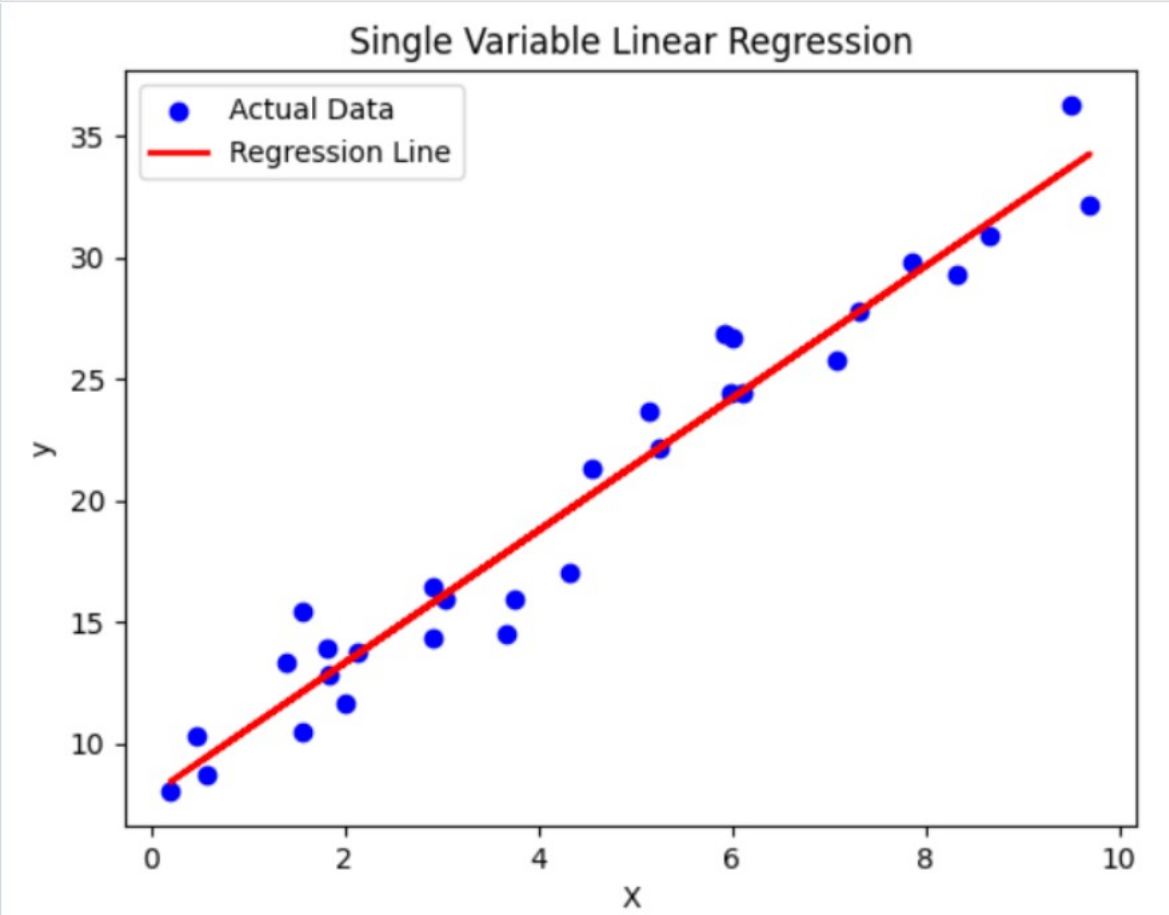
slope (m): 57.48279660799547
Intercept (c): -59.43684554532285

10/8/24

Code:

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

# Generate dataset
np.random.seed(42)
X = np.random.rand(30, 1) * 10  # 30 random values between 0 and 10
y = 3 * X + 7 + np.random.randn(30, 1) * 2  # y = 3x + 7 + noise

# Train Linear Regression model
model = LinearRegression()
model.fit(X, y)

# Predict values
y_pred = model.predict(X)

# Plot results
plt.scatter(X, y, color="blue", label="Actual Data")
plt.plot(X, y_pred, color="red", linewidth=2, label="Regression Line")
plt.xlabel("X")
plt.ylabel("y")
plt.title("Single Variable Linear Regression")
plt.legend()
plt.show()
# Print Model Parameters
print(f"Slope (m): {model.coef_[0][0]}")
print(f"Intercept (c): {model.intercept_[0]}")
```

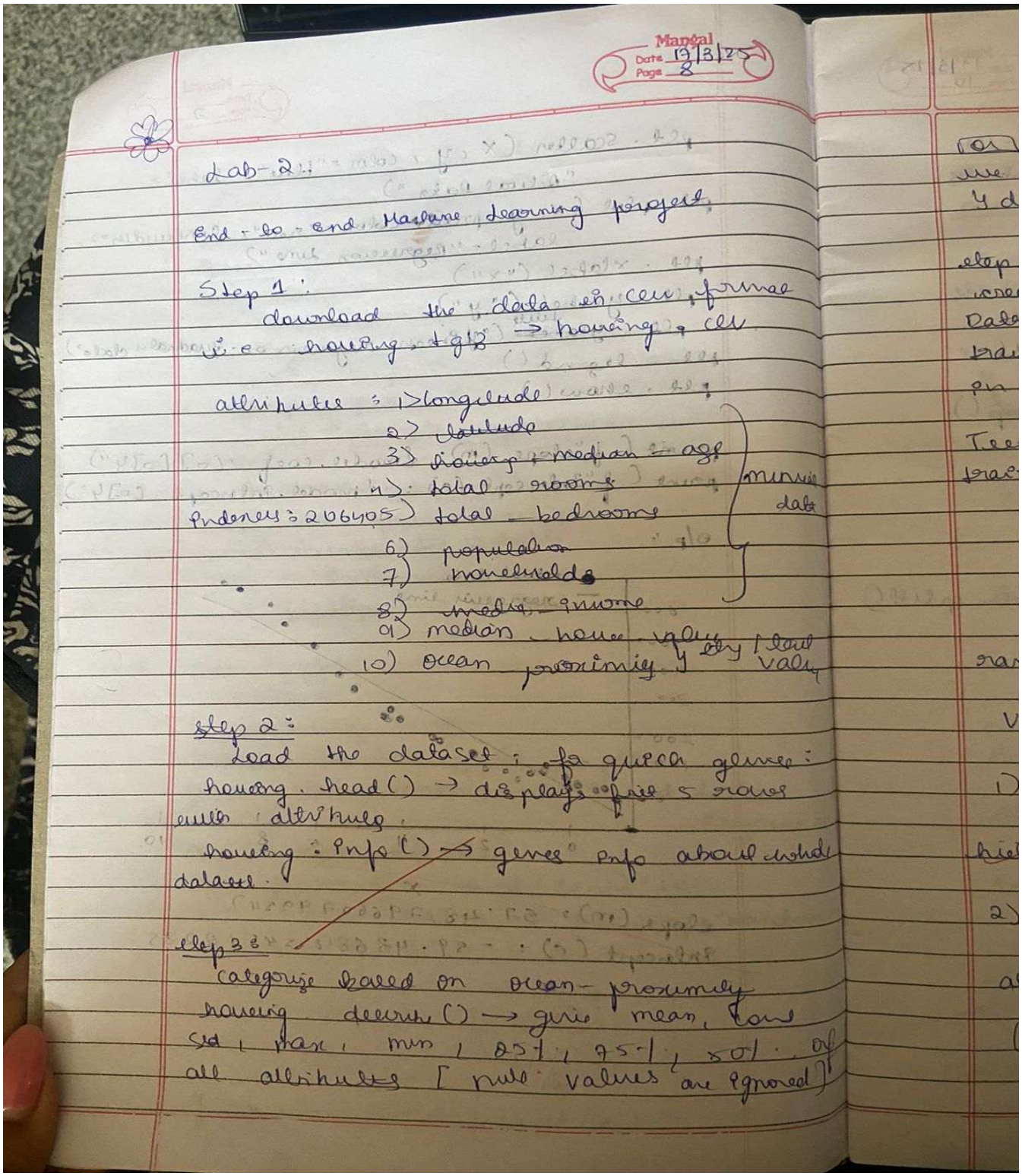
Output:




```
Slope (m): 2.7213842056621584
Intercept (c): 7.857942555802442
```

Program 3:
Demonstrate various data pre-processing techniques for a given dataset

Screenshots



Or

use histogram for each attribute, this also divides the data into 50 equal width

step 4
create a test set
Data Snooping: it occurs when model is trained or tested on data that was used in preprocessing / decisions making process

Test-data : test size
training data = Total dataset - Test data [16512]
also [ ] → specify rows
(np.random.permutation())
using unique ID's & permutable identifier or use hash function

random_state & using split_train_test()

Visualization methodes:

1) Histogram:
• hist() → visualize the data based on histograms

2) using scatter plots
• using scatter plots b/w diff types of attributes

(alpha = 0.1) → density based scatter plotting for visualization

Page 10

3) correlation:
using corr() for attributes

Step-5
data cleaning:
1) remove corresponding labels
2) get rid of whole attribute
3) set values to some value

use dropna(), fillna(), drop()

Step-6
select and train a model
use regression models

for validation use: train_test_split()

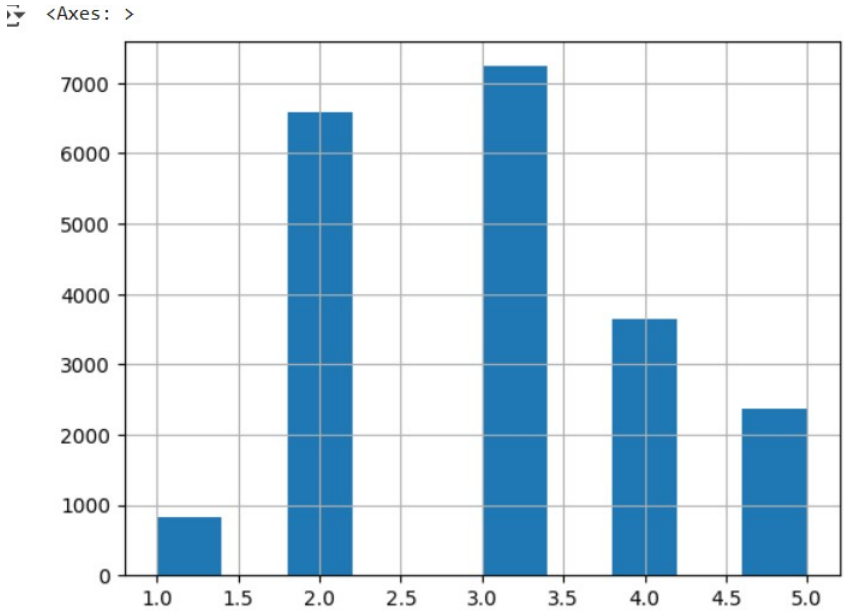
Code

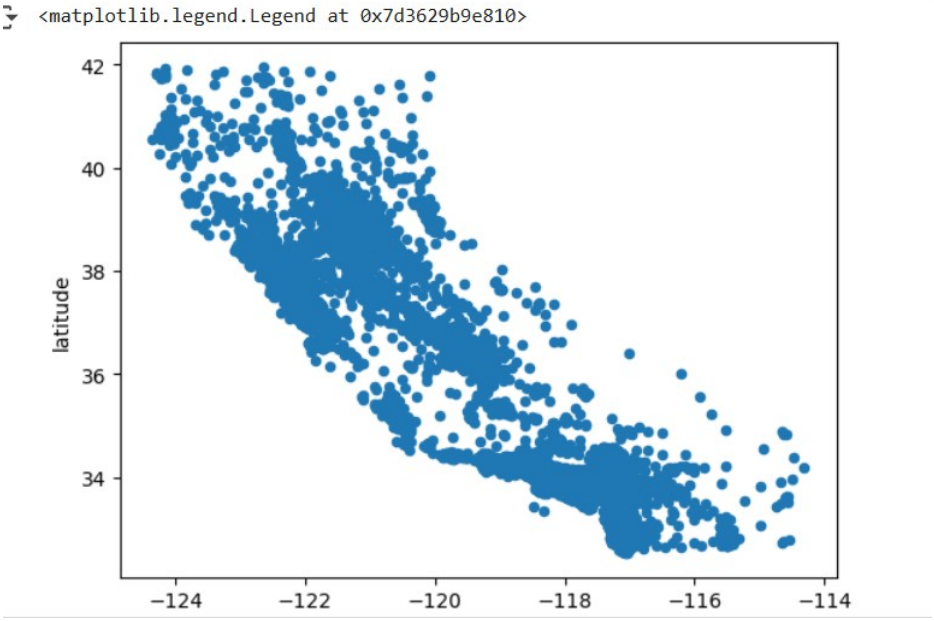
```
import os
import tarfile
import urllib.request

DOWNLOAD_ROOT = "https://raw.githubusercontent.com/ageron/handson-ml2/master/"
HOUSING_PATH = os.path.join("datasets", "housing")
HOUSING_URL = DOWNLOAD_ROOT + "datasets/housing/housing.tgz"

def fetch_housing_data(housing_url=HOUSING_URL, housing_path=HOUSING_PATH):
    os.makedirs(housing_path, exist_ok=True)  # Ensure the directory exists
    tgz_path = os.path.join(housing_path, "housing.tgz")

    try:
        urllib.request.urlretrieve(housing_url, tgz_path)
        housing_tgz = tarfile.open(tgz_path)
        housing_tgz.extractall(path=housing_path)
        housing_tgz.close()
        print("Housing data successfully downloaded and extracted.")
    except urllib.error.URLError as e:
        print(f"Error downloading data: {e}")

# Call the function
fetch_housing_data()
import pandas as pd
def load_housing_data(housing_path=HOUSING_PATH):
  csv_path = os.path.join(housing_path, "housing.csv")
  return pd.read_csv(csv_path)
housing = load_housing_data()
housing.head()
>>> housing["ocean_proximity"].value_counts()

  housing.describe()
  import numpy as np
def split_train_test(data, test_ratio):
  shuffled_indices = np.random.permutation(len(data))
  test_set_size = int(len(data) * test_ratio)
  test_indices = shuffled_indices[:test_set_size]
  train_indices = shuffled_indices[test_set_size:]
  return data.iloc[train_indices], data.iloc[test_indices]
```

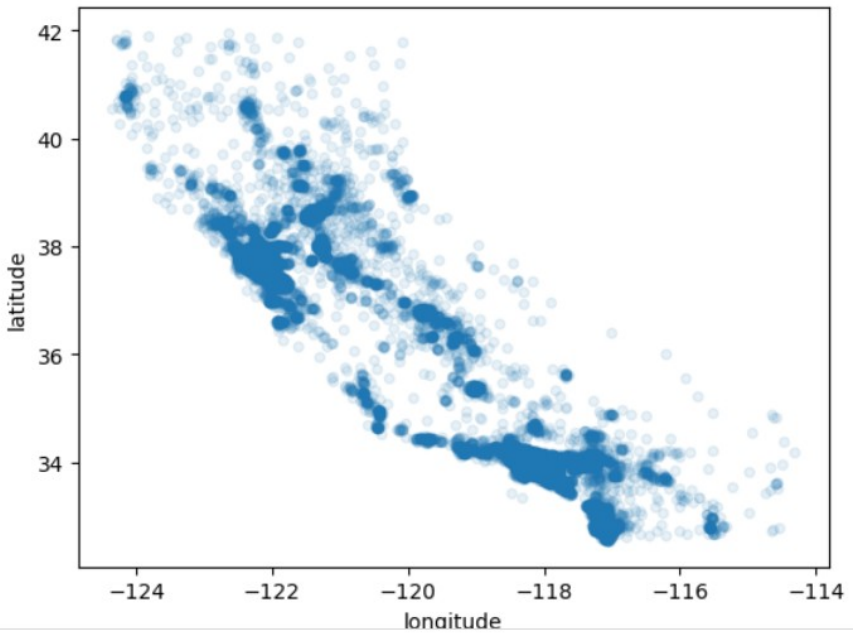
```
>>> train_set, test_set = split_train_test(housing, 0.2)
>>> len(train_set)
  >>> len(test_set)

from zlib import crc32
def test_set_check(identifier, test_ratio):
  return crc32(np.int64(identifier)) & 0xffffffff < test_ratio * 2**32

def split_train_test_by_id(data, test_ratio, id_column):
  ids = data[id_column]
  in_test_set = ids.apply(lambda id_: test_set_check(id_, test_ratio))
  return data.loc[~in_test_set], data.loc[in_test_set]
housing_with_id = housing.reset_index() # adds an `index` column
train_set, test_set = split_train_test_by_id(housing_with_id, 0.2, "index")

from sklearn.model_selection import train_test_split
train_set, test_set = train_test_split(housing, test_size=0.2, random_state=42)

housing["income_cat"] = pd.cut(housing["median_income"],
                               bins=[0., 1.5, 3.0, 4.5, 6., np.inf],
                               labels=[1, 2, 3, 4, 5])
housing["income_cat"].hist()

from sklearn.model_selection import StratifiedShuffleSplit
split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_index, test_index in split.split(housing, housing["income_cat"]):
  strat_train_set = housing.loc[train_index]
  strat_test_set = housing.loc[test_index]
for set_ in (strat_train_set, strat_test_set):
  set_.drop("income_cat", axis=1, inplace=True)

import matplotlib.pyplot as plt

housing = strat_train_set.copy()
housing.plot(kind="scatter", x="longitude", y="latitude")
housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.1)
housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.4,
            s=housing["population"]/100, label="population", figsize=(10,7),
            c="median_house_value", cmap=plt.get_cmap("jet"), colorbar=True,
            )
plt.legend()
```

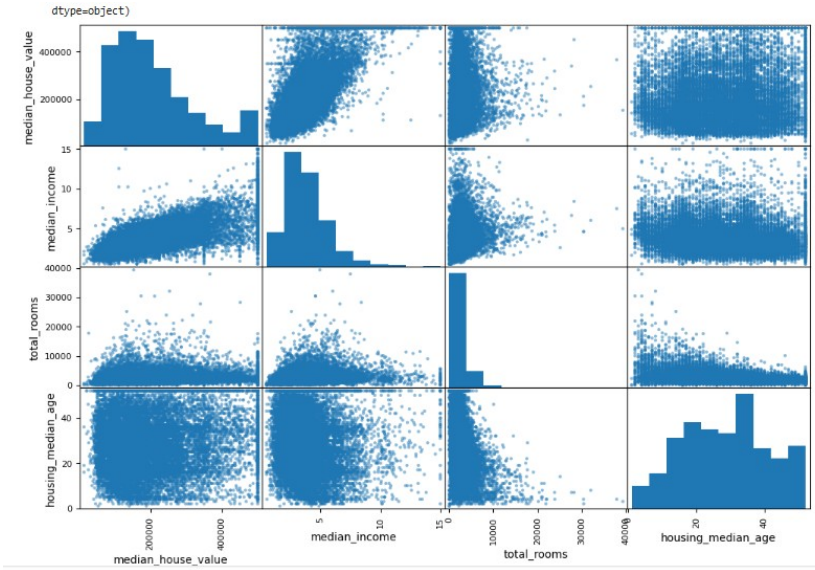
```
corr_matrix = housing.drop(columns=["ocean_proximity"]).corr()
print(corr_matrix)
from pandas.plotting import scatter_matrix
attributes = ["median_house_value", "median_income", "total_rooms",
              "housing_median_age"]
scatter_matrix(housing[attributes], figsize=(12, 8))
```

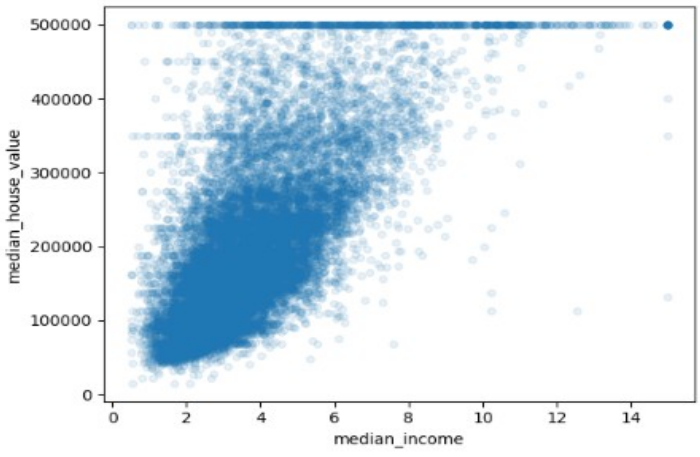
Output:






42
40
38
36
34
latitude
−124
−122
−120
−118
−116
−114
longitude

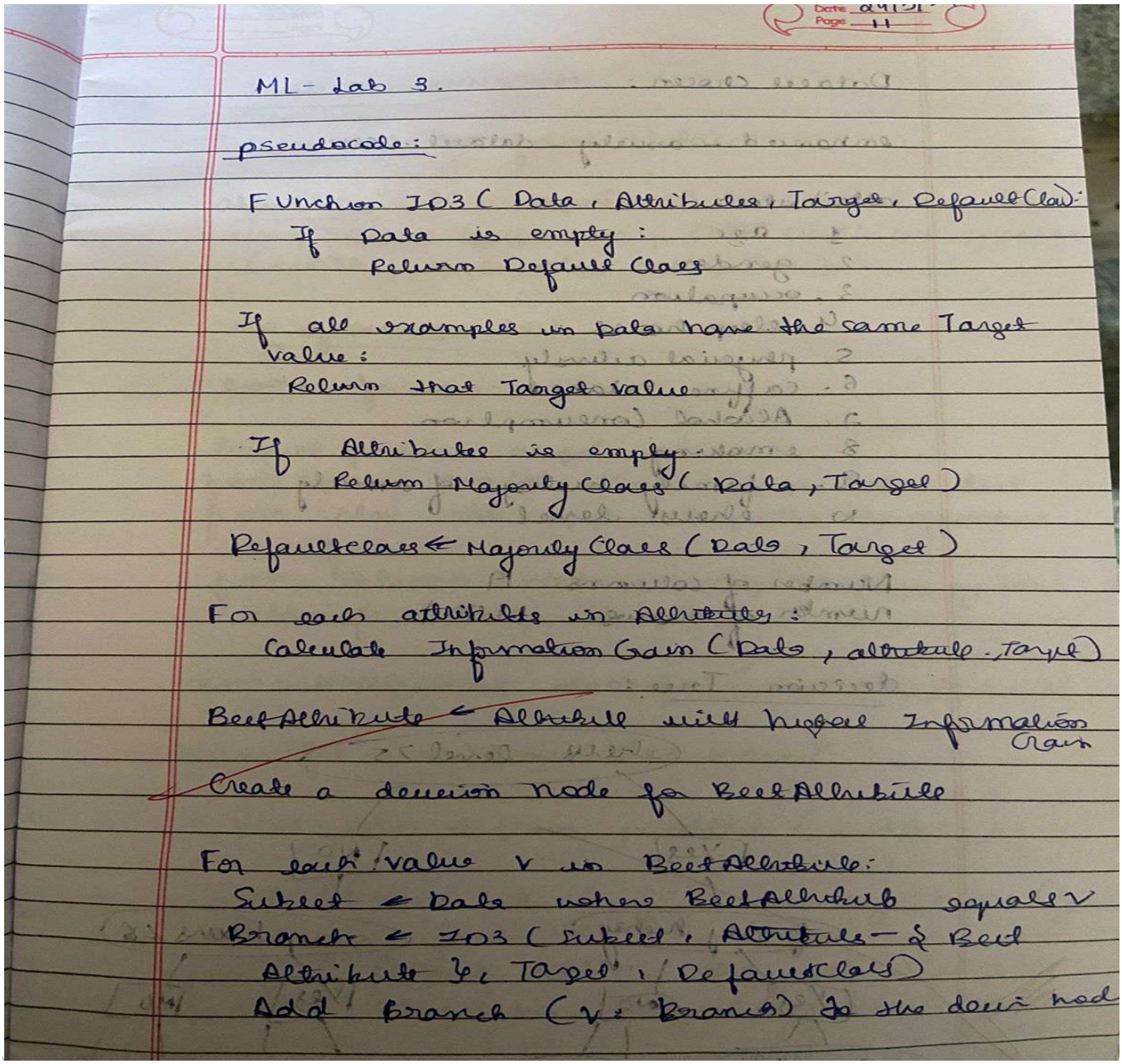

dtype=object)
median_house_value
median_income
total_rooms
housing_median_age


500000
400000
300000
200000
100000
0
median_house_value
0
2
4
6
8
10
12
14
median_income

Program 4

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

Screenshots



ML - Lab 3.

pseudocode:

Function ID3 (Data, Attributes, Target, Default Class):
If Data is empty:
Return Default Class

If all examples in Data have the same Target value:
Return that Target value

If Attributes is empty:
Return Majority Class (Data, Target)

DefaultClass ← Majority Class (Data, Target)

For each attribute in Attributes:
Calculate Information Gain (Data, attribute, Target)

BestAttribute ← Attribute with highest Information Gain

Create a decision node for BestAttribute

For each value v in BestAttribute:
Subset ← Data where BestAttribute equals v
Branch ← ID3 (Subset, Attributes - { BestAttribute }, Target, DefaultClass)
Add Branch (v, Branch) to the decision node

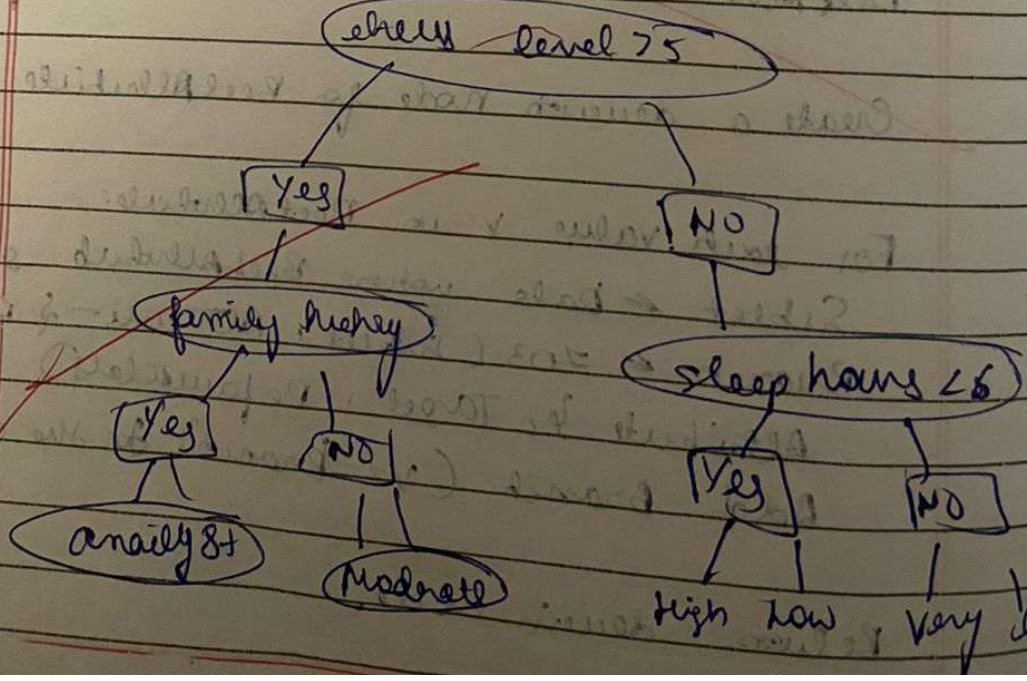
enhanced_anxiety_dataset.csv

attributes:

1. age
2. gender
3. occupation
4. sleep hours
5. physical activity
6. caffine intake
7. Alcohol Consumption
8. smoking
9. family history of anxiety
10. stress level

Number of columns: 19
number of rows: 11001.

decision Tree:



Code:

1. IRIS DATASET

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy_score, confusion_matrix


# Load the IRIS dataset

iris = pd.read_csv("iris (1).csv")

X_iris = iris.iloc[:, :-1]  # Features

y_iris = iris.iloc[:, -1]   # Target


# Split into train (80%) and test (20%)

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

```
# Train DecisionTree classifier

clf_iris = DecisionTreeClassifier()

clf_iris.fit(X_train_iris, y_train_iris)


# Predict and evaluate

y_pred_iris = clf_iris.predict(X_test_iris)

print("IRIS Dataset:")

print("Accuracy Score:", accuracy_score(y_test_iris, y_pred_iris))


# Generate and plot confusion matrix

cm = confusion_matrix(y_test_iris, y_pred_iris)

plt.figure(figsize=(6, 4))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=clf_iris.classes_,
yticklabels=clf_iris.classes_)

plt.xlabel("Predicted Value")

plt.ylabel("Actual Value")

plt.title("Confusion Matrix - IRIS Dataset")

plt.show()
```

IRIS Dataset:

Accuracy Score: 1.0

2. DRUG DATASET

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.preprocessing import LabelEncoder

# Load the Drug dataset
drug = pd.read_csv("drug.csv")
X_drug = drug.iloc[:, :-1]  # Features
y_drug = drug.iloc[:, -1]   # Target

# Encode categorical variables
label_encoders = {}
for col in ['Sex', 'BP', 'Cholesterol']:
    label_encoders[col] = LabelEncoder()
    X_drug[col] = label_encoders[col].fit_transform(X_drug[col])
```

```
# Split into train (80%) and test (20%)

X_train_drug, X_test_drug, y_train_drug, y_test_drug = train_test_split(X_drug, y_drug, test_size=0.2, random_state=42)



# Train DecisionTree classifier

clf_drug = DecisionTreeClassifier()

clf_drug.fit(X_train_drug, y_train_drug)



# Predict and evaluate

y_pred_drug = clf_drug.predict(X_test_drug)

print("Drug Dataset:")

print("Accuracy Score:", accuracy_score(y_test_drug, y_pred_drug))



# Generate and plot confusion matrix

cm = confusion_matrix(y_test_drug, y_pred_drug)

plt.figure(figsize=(6, 4))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=clf_drug.classes_, yticklabels=clf_drug.classes_)

plt.xlabel("Predicted Label")

plt.ylabel("Actual Label")

plt.title("Confusion Matrix - Drug Dataset")

plt.show()
```

Drug Dataset:

Accuracy Score: 1.0

3. PETROL CONSUMPTION DATASET

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor, plot_tree
from sklearn.metrics import mean_absolute_error, mean_squared_error

# Load the Petrol Consumption dataset
petrol = pd.read_csv("petrol_consumption.csv")
X_petrol = petrol.iloc[:, :-1]  # Features
y_petrol = petrol.iloc[:, -1]   # Target

# Split into train (80%) and test (20%)
```

```
X_train_petrol, X_test_petrol, y_train_petrol, y_test_petrol = train_test_split(X_petrol, y_petrol, test_size=0.2, random_state=42)


# Train RegressionTree model

regressor = DecisionTreeRegressor()

regressor.fit(X_train_petrol, y_train_petrol)


# Predict on test data

y_pred_petrol = regressor.predict(X_test_petrol)


# Evaluate the model

mae = mean_absolute_error(y_test_petrol, y_pred_petrol)

mse = mean_squared_error(y_test_petrol, y_pred_petrol)

rmse = np.sqrt(mse)


print("Petrol Consumption Prediction:")

print("Mean Absolute Error:", mae)

print("Mean Squared Error:", mse)

print("Root Mean Squared Error:", rmse)


# Visualizing the Regression Tree

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

```
plot_tree(regressor, feature_names=X_petrol.columns, filled=True)

plt.title("Regression Tree Petrol Consumption Prediction")

plt.show()


# Display feature importance

feature_importance = regressor.feature_importances_

feature_names = X_petrol.columns

sorted_features = sorted(zip(feature_names, feature_importance), key=lambda x: x[1],
reverse=True)

print("Feature Importances:", sorted_features)
```

Petrol Consumption Prediction:

Mean Absolute Error: 96.2

Mean Squared Error: 17858.2

Root Mean Squared Error: 133.63457636405332

```
Feature Importances: [('Population_Driver_licence(%)',
np.float64(0.6657971008323302)), ('Average_income',
np.float64(0.2325126418438352 4)), ('Petrol_tax', np.float64(0.057828019412832596)),
('Paved_Highways', np.float64(0.043862237911001 91))]


import pandas as pd
```

```
import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeRegressor, plot_tree

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import mean_absolute_error, mean_squared_error


petrol_df = pd.read_csv("petrol_consumption.csv")


X = petrol_df.drop(columns=["Petrol_Consumption"])

y = petrol_df["Petrol_Consumption"]


scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)


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


model = DecisionTreeRegressor(max_depth=5, random_state=42)

model.fit(X_train, y_train)
```

```
y_pred = model.predict(X_test)

mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)

print("Petrol Consumption Regression:")
print(f"Mean Absolute Error (MAE): {mae:.4f}")
print(f"Mean Squared Error (MSE): {mse:.4f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.4f}")

plt.figure(figsize=(8, 5))
plt.barh(petrol_df.columns[:-1], model.feature_importances_)
plt.xlabel("Feature Importance")
plt.ylabel("Features")
plt.title("Feature Importance in Petrol Consumption Prediction")
plt.show()

plt.figure(figsize=(12, 8))
plot_tree(model, feature_names=petrol_df.columns[:-1], filled=True, rounded=True)
plt.title("Regression Tree Structure")
```

```
plt.show()
```

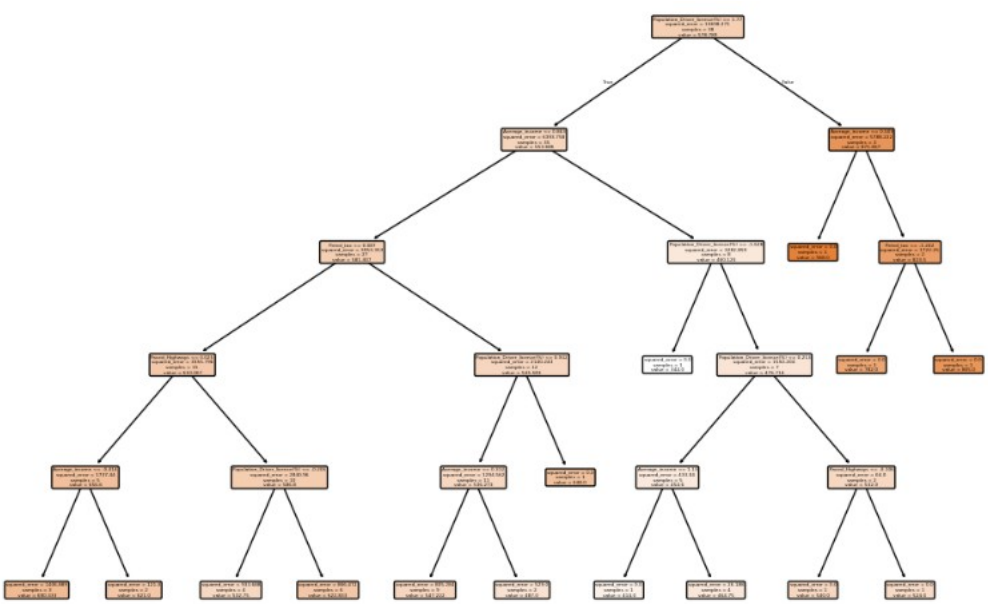
Petrol Consumption Regression:

Mean Absolute Error (MAE): 86.4444

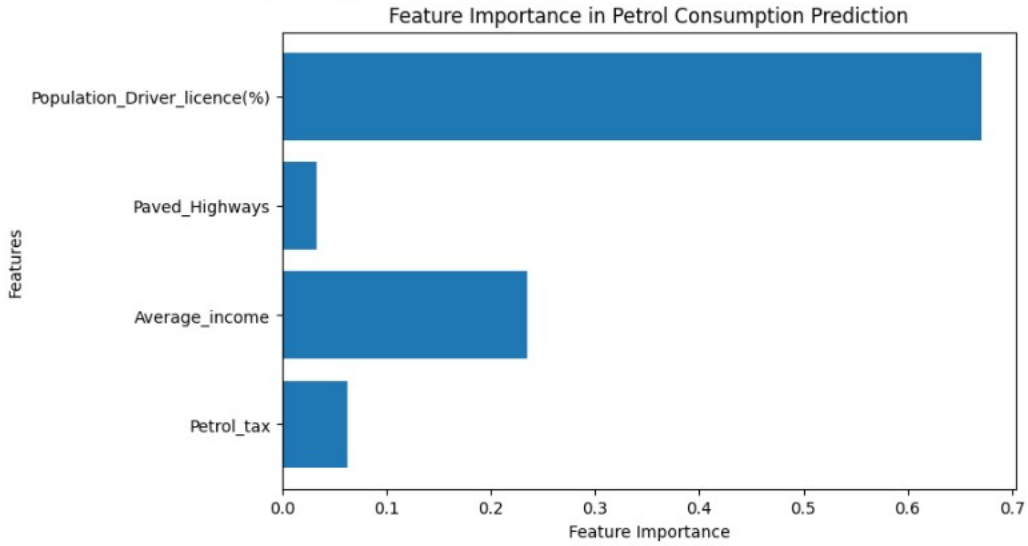
Mean Squared Error (MSE): 15649.4426

Root Mean Squared Error (RMSE): 125.0977

Output:



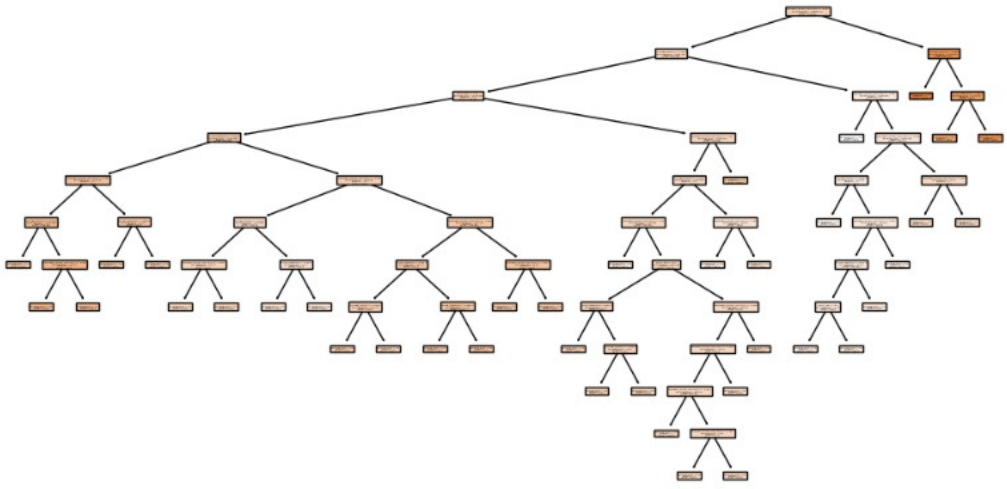
```
Petrol Consumption Regression:
Mean Absolute Error (MAE): 86.4444
Mean Squared Error (MSE): 15649.4426
Root Mean Squared Error (RMSE): 125.0977
```



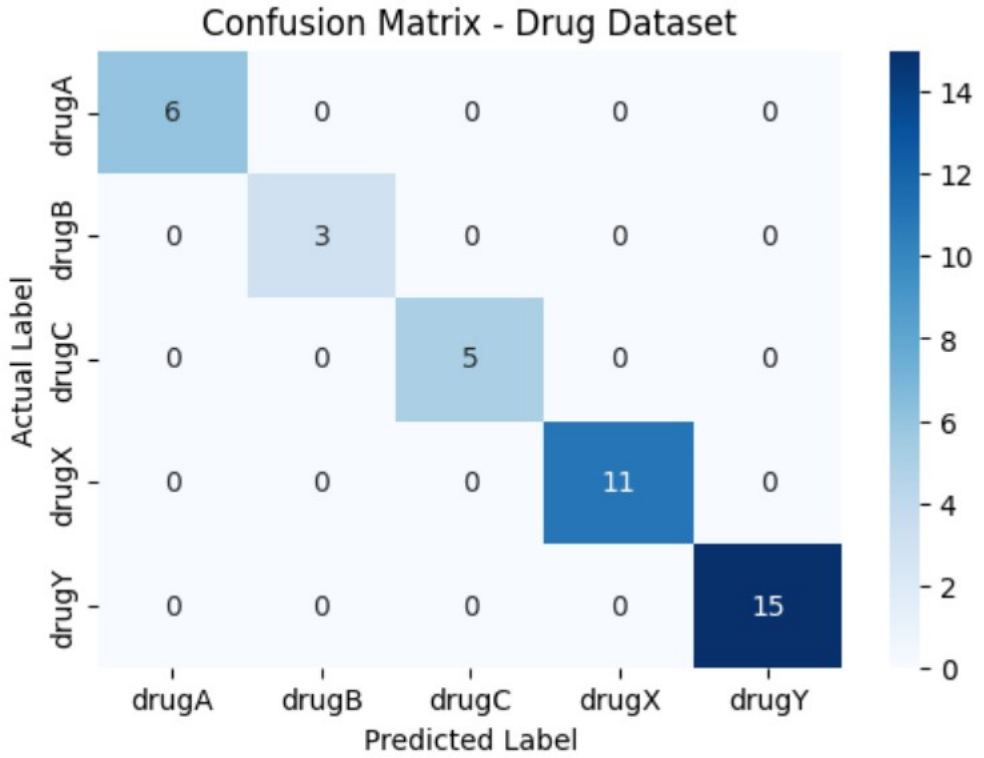
Regression Tree Structure

```
Petrol Consumption Prediction:
Mean Absolute Error: 96.2
Mean Squared Error: 17858.2
Root Mean Squared Error: 133.63457636405332
```

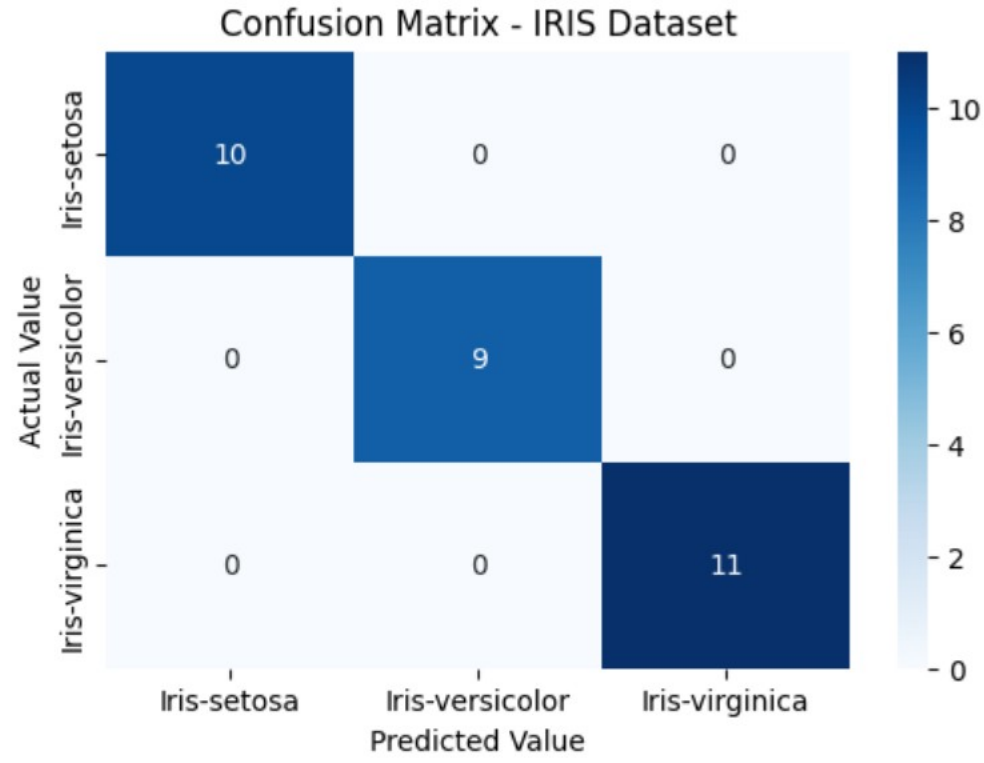
Regression Tree Petrol Consumption Prediction



```
Drug Dataset:
Accuracy Score: 1.0
```



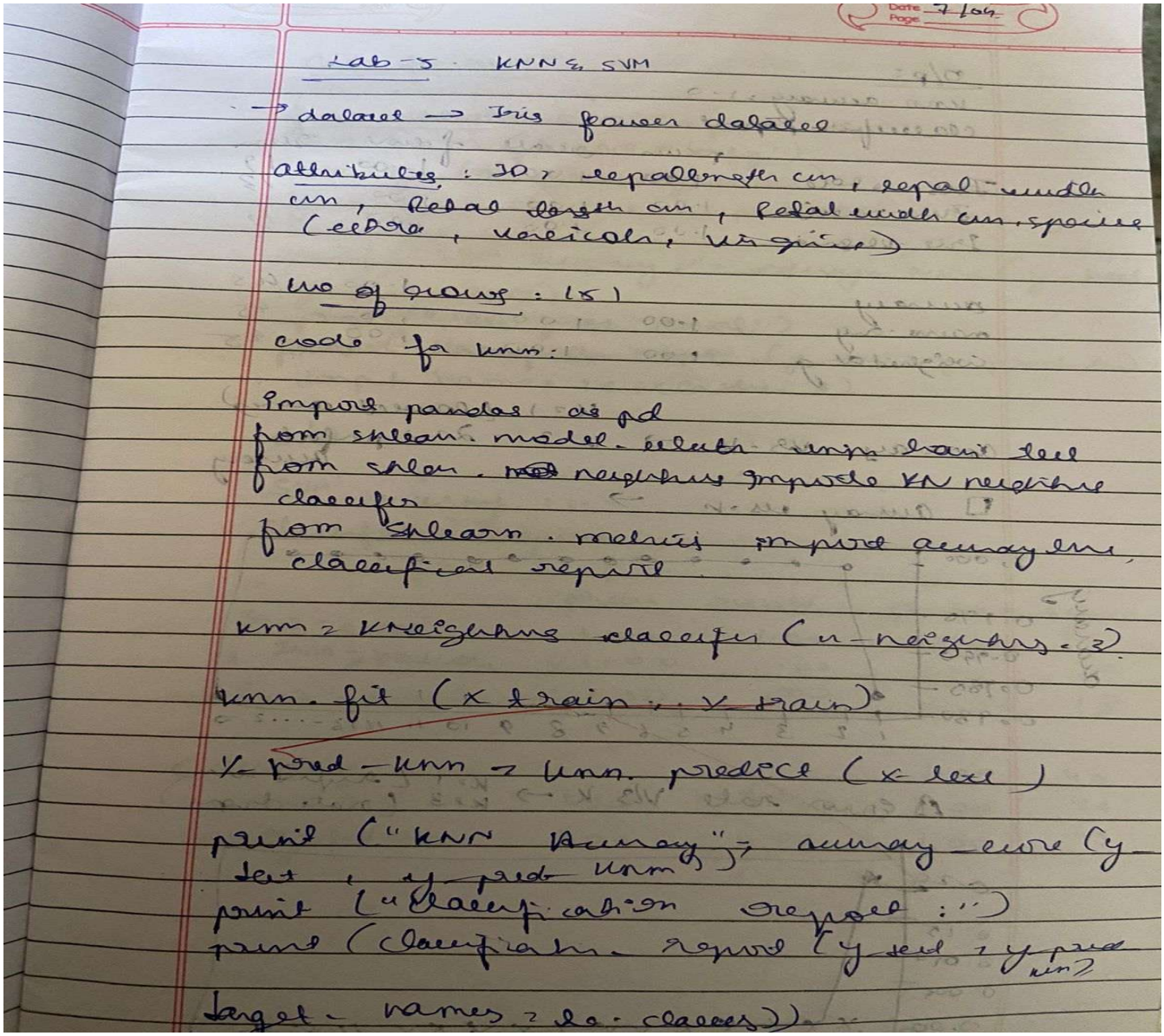

```
IRIS Dataset:
Accuracy Score: 1.0
```



Program 5:

Build Logistic Regression Model for a given dataset

Screenshots



Lab-5 KNN & SVM

→ dataset → Iris flower dataset

Attributes : ID, sepallength cm, sepal width cm, Petal length cm, Petal width cm, species (setosa, versicolor, virginica)

No of Rows : 150

code for knn:

```
import pandas as pd
from sklearn.model_selection import train_test
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report

knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
y_pred_knn = knn.predict(X_test)
print("KNN Accuracy:", accuracy_score(y_test, y_pred_knn))
print("Classification report:")
print(classification_report(y_test, y_pred_knn,
target_names=iris.target_names))
```

O/p:

knn accuracy: 1.00

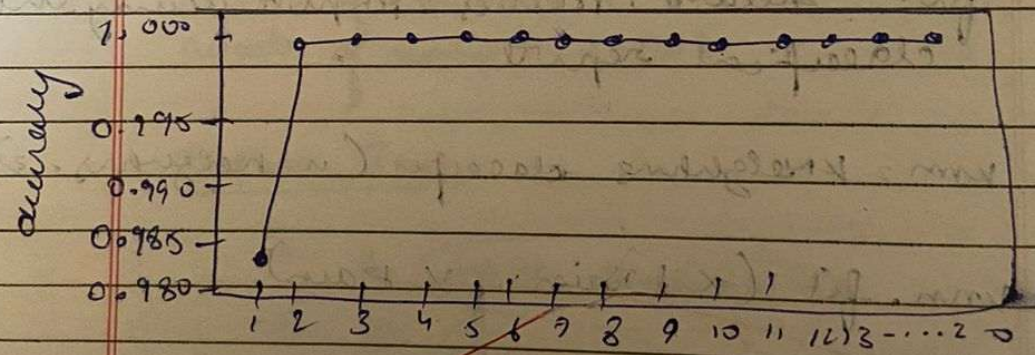
Classification Report

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	19
Iris-versicolor	1.00	1.00	1.00	13
Iris-virginica	1.00	1.00	1.00	13
accuracy			1.00	45
macro avg	1.00	1.00	1.00	45
weighted avg	1.00	1.00	1.00	45

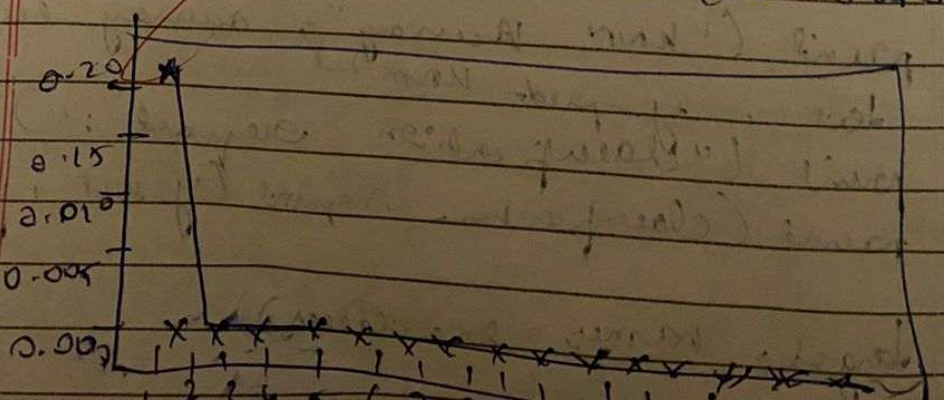
K=1 (lowest accuracy)

Accuracy vs K → K=3 - 20 → perfect accuracy

□ Accuracy vs K →



Δ Error rate V/S K → K=1 (error rate)
K=3 (rate drops)



Mangal
Date
Page

→ Code for SVM:

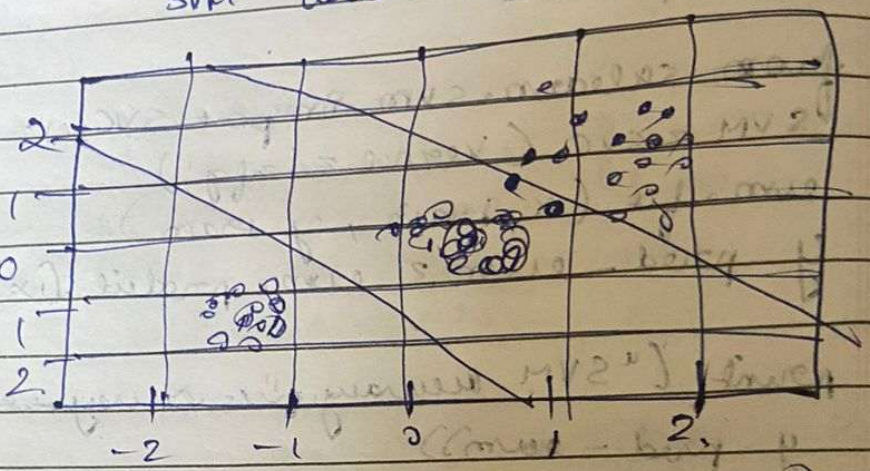
```
from sklearn.svm import SVC
svm = SVC(kernel='rbf')
svm.fit(x_train, y_train)
y_pred_svm = svm.predict(x_test)
print("SVM Accuracy:", accuracy_score(y_test, y_pred_svm))
print("Classification report:")
print(classification_report(y_test, y_pred_svm, target_names=iris.target_names))
```

o/p:

SVM accuracy : 1.0
Classification report:

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	19
Iris-versicolor	1.00	1.00	1.00	13
Iris-virginica	1.00	1.00	1.00	13
accuracy			1.00	45
macro avg	1.00	1.00	1.00	45
weighted avg	1.00	1.00	1.00	45

SVM dessin Boundary

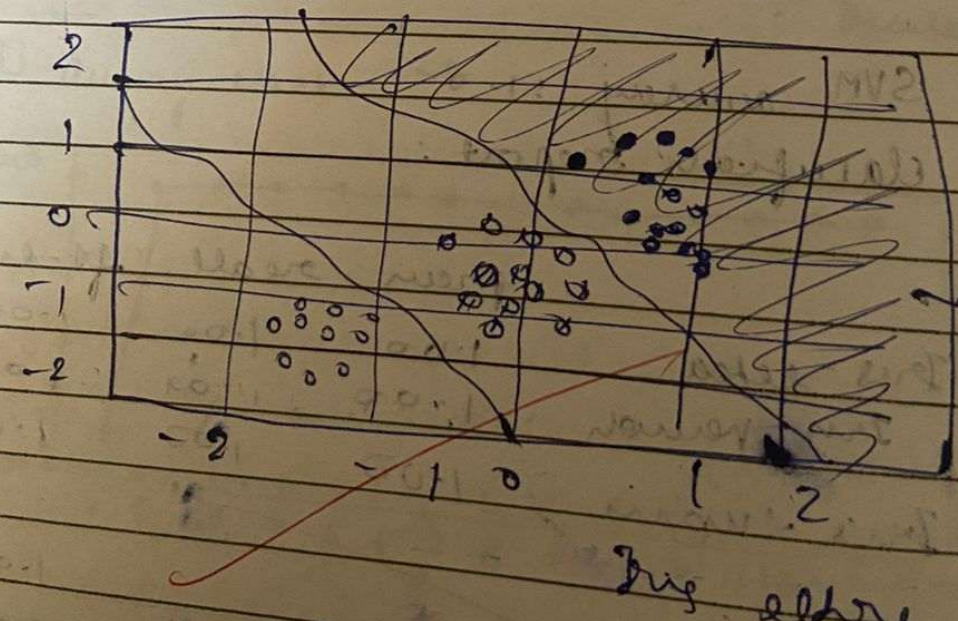


petal length (standardized)

0 - Iris-setosa ① - Iris-versicolor

● - Iris virginica

KNN dessin Boundary



Iris virginica

Iris setosa

Code

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn import metrics
import matplotlib.pyplot as plt

# Load the Iris dataset
iris =  pd.read_csv("iris.csv")
iris.head()

X=iris.drop('species',axis='columns')# Features (sepal length, sepal width, petal length, petal width)
y = iris.species # Target labels (0: Setosa, 1: Versicolor, 2: Virginica)

# Split the dataset into 80% training and 20% testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the Multinomial Logistic Regression model
# Use 'multinomial' for multi-class classification and 'lbfgs' solver
model = LogisticRegression(multi_class='multinomial')
```

```
# Train the model on the training data
model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = model.predict(X_test)

# Calculate the accuracy of the model on the test data
accuracy = accuracy_score(y_test, y_pred)

# Display the accuracy
print(f"Accuracy of the Multinomial Logistic Regression model on the test set: {accuracy:.2f}")

confusion_matrix = metrics.confusion_matrix(y_test, y_pred)

cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = ["Setosa", "Versicolor", "Virginica"])

cm_display.plot()
plt.show()
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:1247: FutureWarning: 'multi_class' was deprecated in version 1.5 and will be removed in 1.7. From then on, it will always use 'multinomial'. Leave it to its default value to avoid this warning.
  warnings.warn(
Accuracy of the Multinomial Logistic Regression model on the test set: 1.00
```

```
import pandas as pd

from matplotlib import pyplot as plt

# %matplotlib inline

#"%matplotlib inline" will make your plot outputs appear and be stored within the notebook.


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

df.head()


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


from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test =
train_test_split(df[['age']],df.bought_insurance,train_size=0.9,random_state=10)

X_train.shape


X_test


from sklearn.linear_model import LogisticRegression

model = LogisticRegression()


model.fit(X_train, y_train)
```

```
X_test

y_test

y_predicted = model.predict(X_test)
y_predicted

model.score(X_test,y_test)

model.predict_proba(X_test)

y_predicted = model.predict([[60]])
y_predicted

#model.coef_ indicates value of m in y=m*x + b equation
model.coef_

#model.intercept_ indicates value of b in y=m*x + b equation
model.intercept_

#Lets defined sigmoid function now and do the math with hand
import math
def sigmoid(x):
  return 1 / (1 + math.exp(-x))

def prediction_function(age):
```

```
    z = 0.127 * age - 4.973 # 0.12740563 ~ 0.0127 and -4.97335111 ~ -4.97
    y = sigmoid(z)
    return y

age = 35
prediction_function(age)

"""0.37 is less than 0.5 which means person with 35 will not buy the insurance"""

/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but LogisticRegression was fitted with feature names
  warnings.warn(
'0.37 is less than 0.5 which means person with 35 will not buy the insurance'



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

# Load dataset
file_path = "HR_comma_sep.csv"  # Update this if needed
df = pd.read_csv(file_path)

# Display basic info
display(df.info())
display(df.head())
```

```
# Correlation heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(df.corr(numeric_only=True), annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5)
plt.title("Correlation Heatmap of Numerical Variables")
plt.show()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14999 entries, 0 to 14998
Data columns (total 10 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   satisfaction_level     14999 non-null  float64
 1   last_evaluation        14999 non-null  float64
 2   number_project         14999 non-null  int64  
 3   average_montly_hours   14999 non-null  int64  
 4   time_spend_company     14999 non-null  int64  
 5   Work_accident          14999 non-null  int64  
 6   left                   14999 non-null  int64  
 7   promotion_last_5years  14999 non-null  int64  
 8   Department             14999 non-null  object 
 9   salary                 14999 non-null  object 
dtypes: float64(2), int64(6), object(2)
memory usage: 1.1+ MB
None
```

satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Work_accident	left	promotion_last_5years	Department	salary
0	0.38	0.53	2	157	3	0	1	0	sales	low
1	0.80	0.86	5	262	6	0	1	0	sales	medium
2	0.11	0.88	7	272	4	0	1	0	sales	medium
3	0.72	0.87	5	223	5	0	1	0	sales	low
4	0.37	0.52	2	159	3	0	1	0	sales	low

```
# Salary vs. Retention
plt.figure(figsize=(6, 4))
sns.countplot(data=df, x="salary", hue="left", order=["low", "medium", "high"])
plt.title("Salary vs. Employee Retention")
plt.ylabel("Count")
plt.show()
```

```
plt.figure(figsize=(10, 6))
sns.countplot(x='Department', hue='left', data=df)
plt.title("Impact of Department on Employee Retention")
plt.xlabel("Department")
plt.ylabel("Employee Count")
plt.xticks(rotation=45)
plt.show()
```

```
df_encoded = pd.get_dummies(df, columns=['salary', 'Department'], drop_first=True)
print(df_encoded.columns)

features = ['average_montly_hours','time_spend_company']

salary_columns = [col for col in df_encoded.columns if 'salary_' in col]
features.extend(salary_columns)

department_columns = [col for col in df_encoded.columns if 'Department_' in col]
features.extend(department_columns)

X = df_encoded[features]
y = df_encoded['left']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()
```

```
logreg.fit(X_train, y_train)

LogisticRegression()
from sklearn.metrics import accuracy_score, classification_report

y_pred = logreg.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy:.2f}\n")
print("Classification Report:")
print(classification_report(y_test, y_pred))

import pandas as pd

df = pd.read_csv("zoo-data.csv")

print(df.info())

print(df.head())

print(df.isnull().sum())

df.drop(columns=['animal_name'], inplace=True)

# Separate features and target variable
X = df.drop(columns=['class_type'])  # Features
```

```
y = df['class_type']            # Target variable
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 101 entries, 0 to 100
Data columns (total 18 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   animal_name  101 non-null    object
 1   hair         101 non-null    int64
 2   feathers     101 non-null    int64
 3   eggs         101 non-null    int64
 4   milk         101 non-null    int64
 5   airborne     101 non-null    int64
 6   aquatic      101 non-null    int64
 7   predator     101 non-null    int64
 8   toothed      101 non-null    int64
 9   backbone     101 non-null    int64
 10  breathes     101 non-null    int64
 11  venomous     101 non-null    int64
 12  fins         101 non-null    int64
 13  legs         101 non-null    int64
 14  tail         101 non-null    int64
 15  domestic     101 non-null    int64
 16  catsize      101 non-null    int64
 17  class_type   101 non-null    int64
```

dtypes: int64(17), object(1)

memory usage: 14.3+ KB

None

	animal_name	hair	feathers	eggs	milk	airborne	aquatic	predator	\
0	aardvark	1	0	0	1	0	0	1	
1	antelope	1	0	0	1	0	0	0	
2	bass	0	0	1	0	0	1	1	
3	bear	1	0	0	1	0	0	1	
4	boar	1	0	0	1	0	0	1	

	toothed	backbone	breathes	venomous	fins	legs	tail	domestic	catsize	\
0	1	1	1	0	0	4	0	0	1	
1	1	1	1	0	0	4	1	0	1	
2	1	1	0	0	1	0	1	0	0	
3	1	1	1	0	0	4	0	0	1	
4	1	1	1	0	0	4	1	0	1	

	class_type
0	1
1	1
2	4
3	1
4	1

animal_name 0

hair 0

feathers 0

```
eggs          0
milk          0
airborne      0
aquatic       0
predator      0
toothed       0
backbone      0
breathes      0
venomous      0
fins          0
legs          0
tail          0
domestic      0
catsize       0
class_type    0
dtype: int64
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression

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

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
```

```
X_test = scaler.transform(X_test)


logreg = LogisticRegression(max_iter=200, multi_class='multinomial', solver='lbfgs')
logreg.fit(X_train, y_train)


/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:1247:
FutureWarning: 'multi_class' was deprecated in version 1.5 and will be removed in 1.7.
From then on, it will always use 'multinomial'. Leave it to its default value to avoid this
warning.
  warnings.warn(
LogisticRegression(max_iter=200, multi_class='multinomial')
In a Jupyter environment, please rerun this cell to show the HTML representation or trust
the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page
with nbviewer.org.


from sklearn.metrics import accuracy_score


y_pred = logreg.predict(X_test)


accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy:.2f}")


class_info = pd.read_csv("zoo-class-type.csv")


class_mapping = dict(zip(class_info['Class_Number'], class_info['Class_Type']))


print(class_mapping)
```

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

print("Predicted Classes:", pred_classes)

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

cm = confusion_matrix(y_test, y_pred)

class_labels = [class_mapping[num] for num in logreg.classes_]

disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_labels)
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix with Class Names")
plt.show()

from sklearn.metrics import classification_report

accuracy = accuracy_score(y_test, y_pred)
print(f"Overall Accuracy: {accuracy:.2f}\n")

# Class-wise precision, recall, and F1-score
print("Classification Report:\n")
print(classification_report(y_test, y_pred, target_names=class_labels))
```

Output:

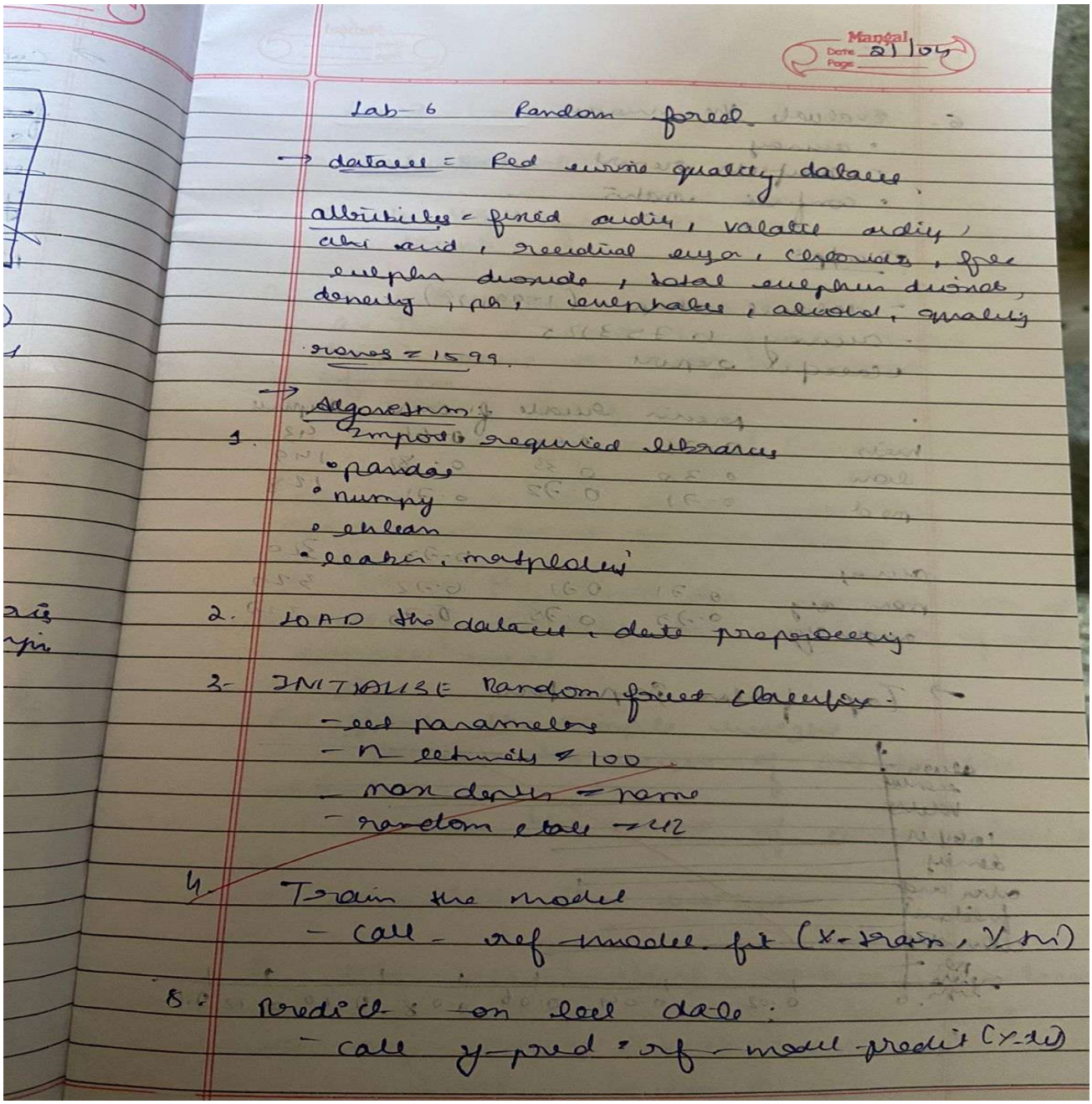
```
Model Accuracy: 0.75

Classification Report:
              precision    recall  f1-score   support

           0       0.76      0.98      0.86      2294
           1       0.08      0.01      0.01       706

    accuracy                           0.75      3000
   macro avg       0.42      0.49      0.43      3000
weighted avg       0.60      0.75      0.66      3000
```

```
Overall Accuracy: 1.00

Classification Report:

              precision    recall  f1-score   support

      Mammal       1.00      1.00      1.00         8
        Bird       1.00      1.00      1.00         4
     Reptile       1.00      1.00      1.00         1
        Fish       1.00      1.00      1.00         3
   Amphibian       1.00      1.00      1.00         1
         Bug       1.00      1.00      1.00         2
Invertebrate       1.00      1.00      1.00         2

    accuracy                           1.00        21
   macro avg       1.00      1.00      1.00        21
weighted avg       1.00      1.00      1.00        21
```

Program 6

Build KNN Classification model for a given dataset

Screenshots



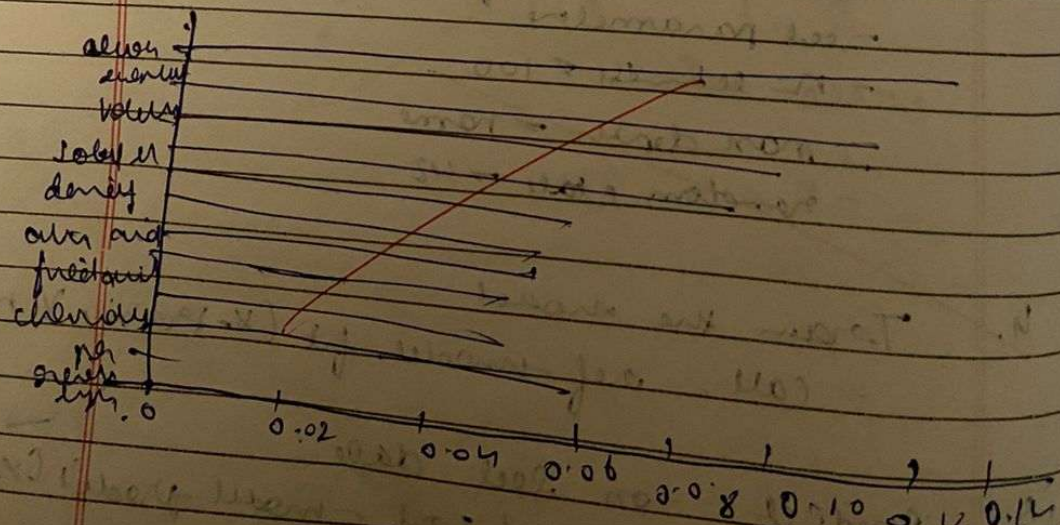
6. Evaluate the model
- accuracy
- classification report
- confusion matrix

→ o/p:

- dataset shape - (1599, 12)
- accuracy 0.753125
classification report

	precision	recall	f1-score	support
high	0.71	0.58	0.64	43
low	0.30	0.33	0.31	149
med	0.71	0.72	0.71	128
accuracy			0.53	320
macro avg	0.57	0.54	0.52	320
weighted avg	0.57	0.53	0.55	320

→ Top feature importance



Code

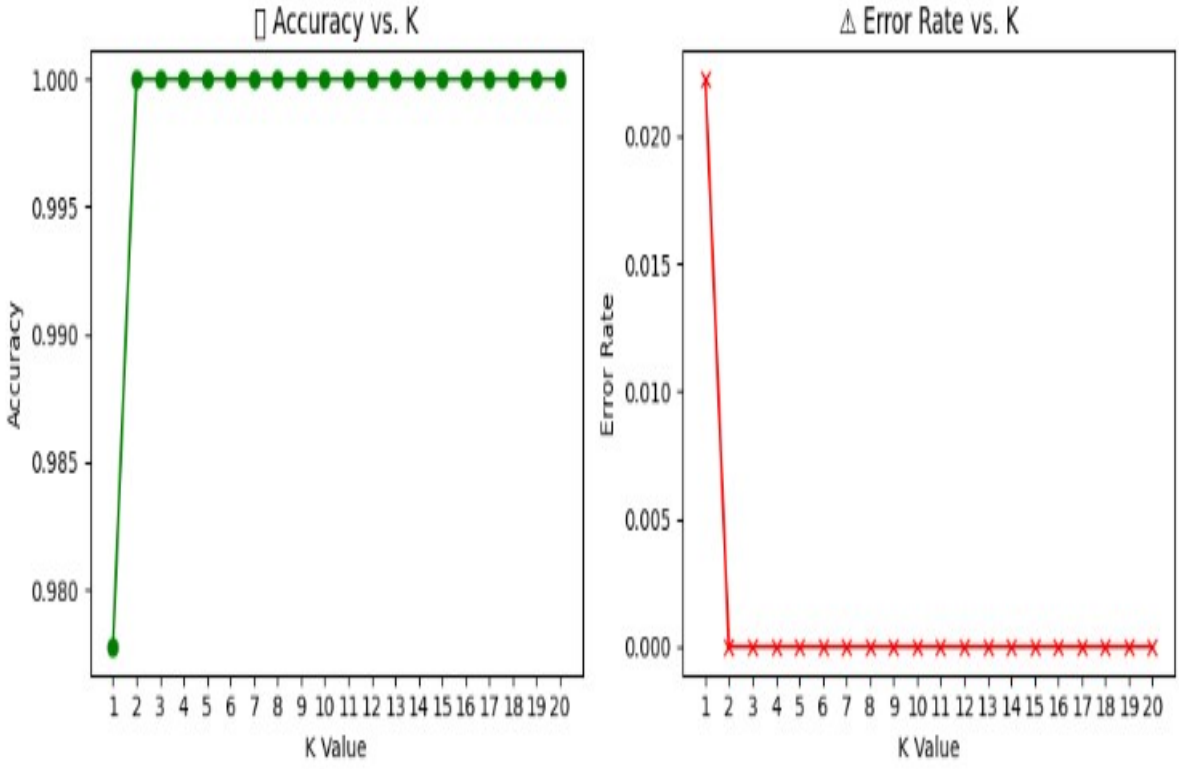
```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Try different values of k
k_range = range(1, 21)
accuracies = []
error_rates = []

for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred_k = knn.predict(X_test)

    acc = accuracy_score(y_test, y_pred_k)
    accuracies.append(acc)
    error_rates.append(1 - acc)  # Error = 1 - Accuracy

# Plot Accuracy
plt.figure(figsize=(10, 4))

plt.subplot(1, 2, 1)
plt.plot(k_range, accuracies, color='green', marker='o')
plt.title(' Accuracy vs. K')
plt.xlabel('K Value')
plt.ylabel('Accuracy')
plt.xticks(k_range)

# Plot Error Rate
plt.subplot(1, 2, 2)
plt.plot(k_range, error_rates, color='red', marker='x')
plt.title('Error Rate vs. K')
plt.xlabel('K Value')
plt.ylabel('Error Rate')
plt.xticks(k_range)

plt.tight_layout()
plt.show()
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report

# Load the dataset
df = pd.read_csv("Iris.csv")

# Drop ID column if present
df.drop(columns=["Id"], inplace=True)

# Split features and labels
X = df.drop("Species", axis=1)
y = df["Species"]

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# KNN Classifier
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

# Predictions
y_pred_knn = knn.predict(X_test)

# Evaluation
print(" KNN Accuracy:", accuracy_score(y_test, y_pred_knn))
print(" Classification Report:\n", classification_report(y_test, y_pred_knn))
```

Output:

```
 KNN Accuracy: 1.0
 Classification Report:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        19
Iris-versicolor       1.00      1.00      1.00        13
 Iris-virginica       1.00      1.00      1.00        13

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

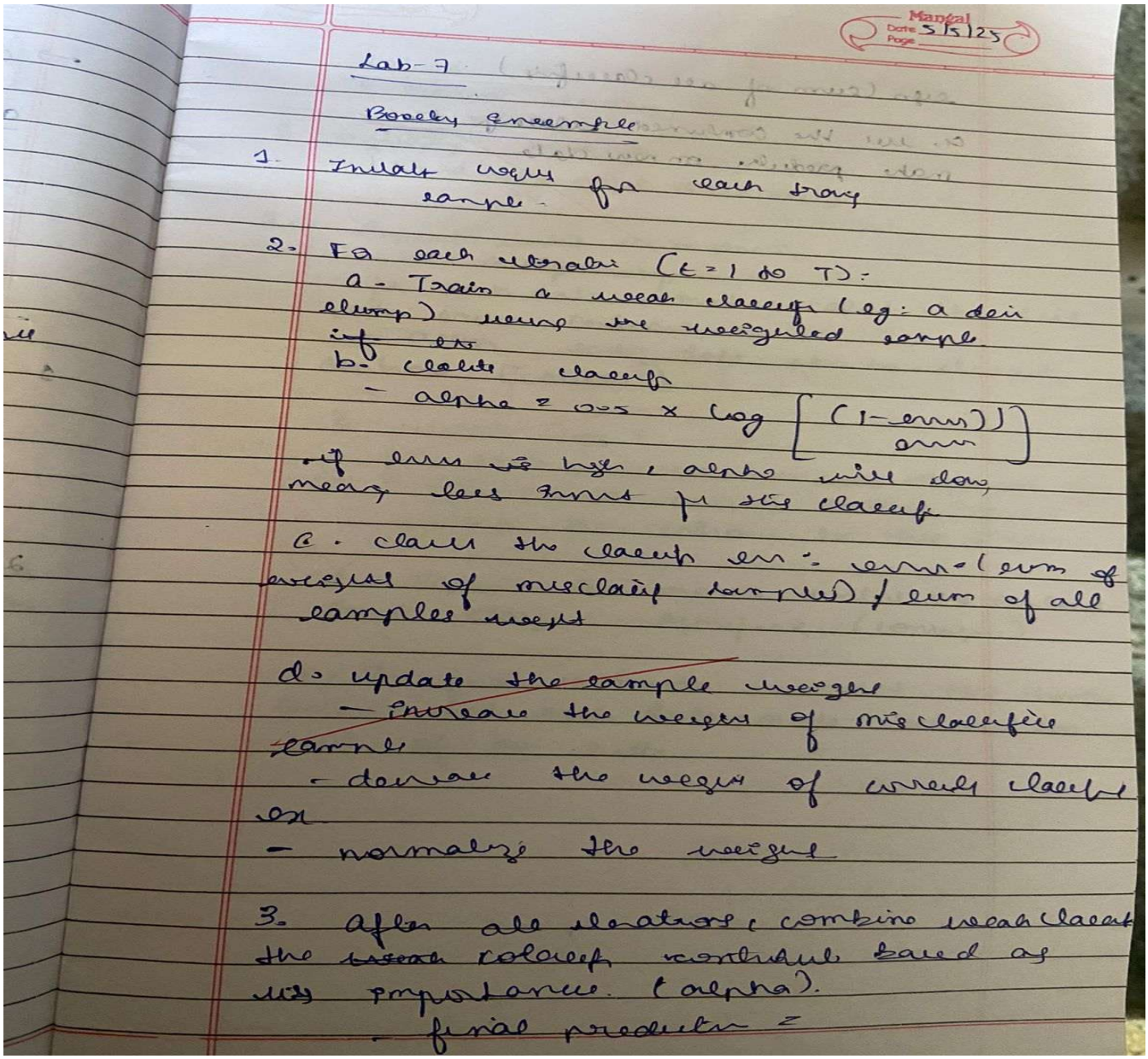
```
fig.canvas.print_figure(bytes_io, **kw)
```

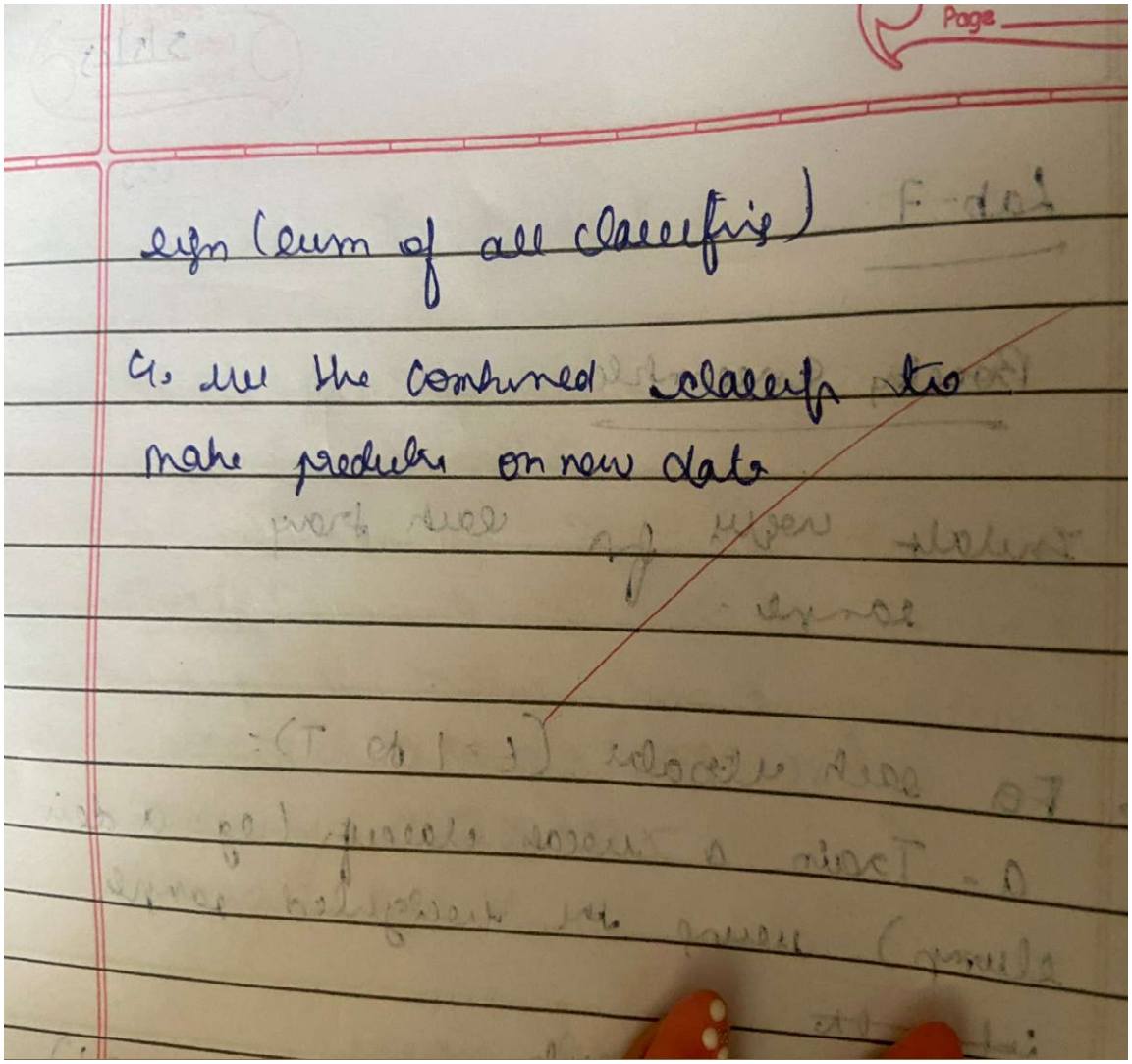


Program 7

Build Support vector machine model for a given dataset

Screenshots




sign (sum of all classifiers)
Q. use the combined classifier to
make predictions on new data

Code

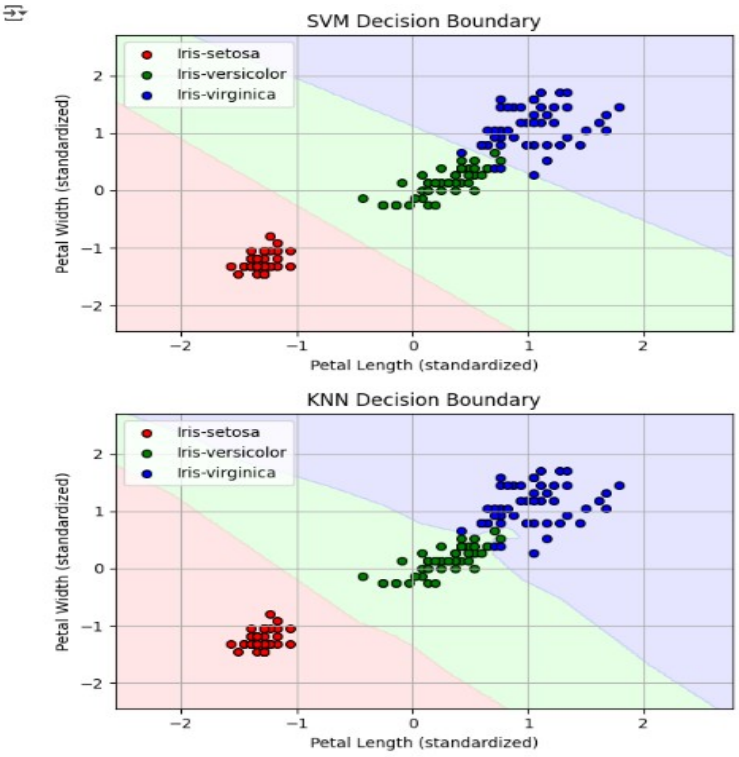
```
# Import Libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report

#Load the Dataset
df = pd.read_csv("Iris.csv")

# Drop 'Id' column if it exists
df.drop(columns=["Id"], inplace=True)

# Encode Target Labels
le = LabelEncoder()
y = le.fit_transform(df["Species"])  # Converts to 0, 1, 2
X = df.drop("Species", axis=1)

#  Feature Scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

#  Split Dataset
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3,
random_state=42)

Train SVM Model (Linear Kernel)
svm = SVC(kernel='rbf')
svm.fit(X_train, y_train)

#  Predictions & Evaluation
y_pred = svm.predict(X_test)
```

```
print(" SVM Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred,
target_names=le.classes_))


import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

# Simple plot function
def plot_decision_boundary_simple(clf, model_name):
    # Create a mesh grid
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
                         np.arange(y_min, y_max, 0.02))

    # Predict on mesh grid
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    # Plot
    plt.figure(figsize=(6, 4))
    cmap_light = ListedColormap(['#FFCCCC', '#CCFFCC', '#CCCCFF'])
    cmap_bold = ['red', 'green', 'blue']

    plt.contourf(xx, yy, Z, alpha=0.5, cmap=cmap_light)
    for i, color in zip(np.unique(y), cmap_bold):
        plt.scatter(X[y == i, 0], X[y == i, 1], c=color, label=le.classes_[i], edgecolor='k', s=30)

    plt.xlabel("Petal Length (standardized)")
    plt.ylabel("Petal Width (standardized)")
    plt.title(f"{model_name} Decision Boundary")
    plt.legend()
    plt.grid(True)
```

```
    plt.tight_layout()
    plt.show()

# Plot SVM
plot_decision_boundary_simple(svm, "SVM")

# Plot KNN
plot_decision_boundary_simple(knn, "KNN")
```

Output:

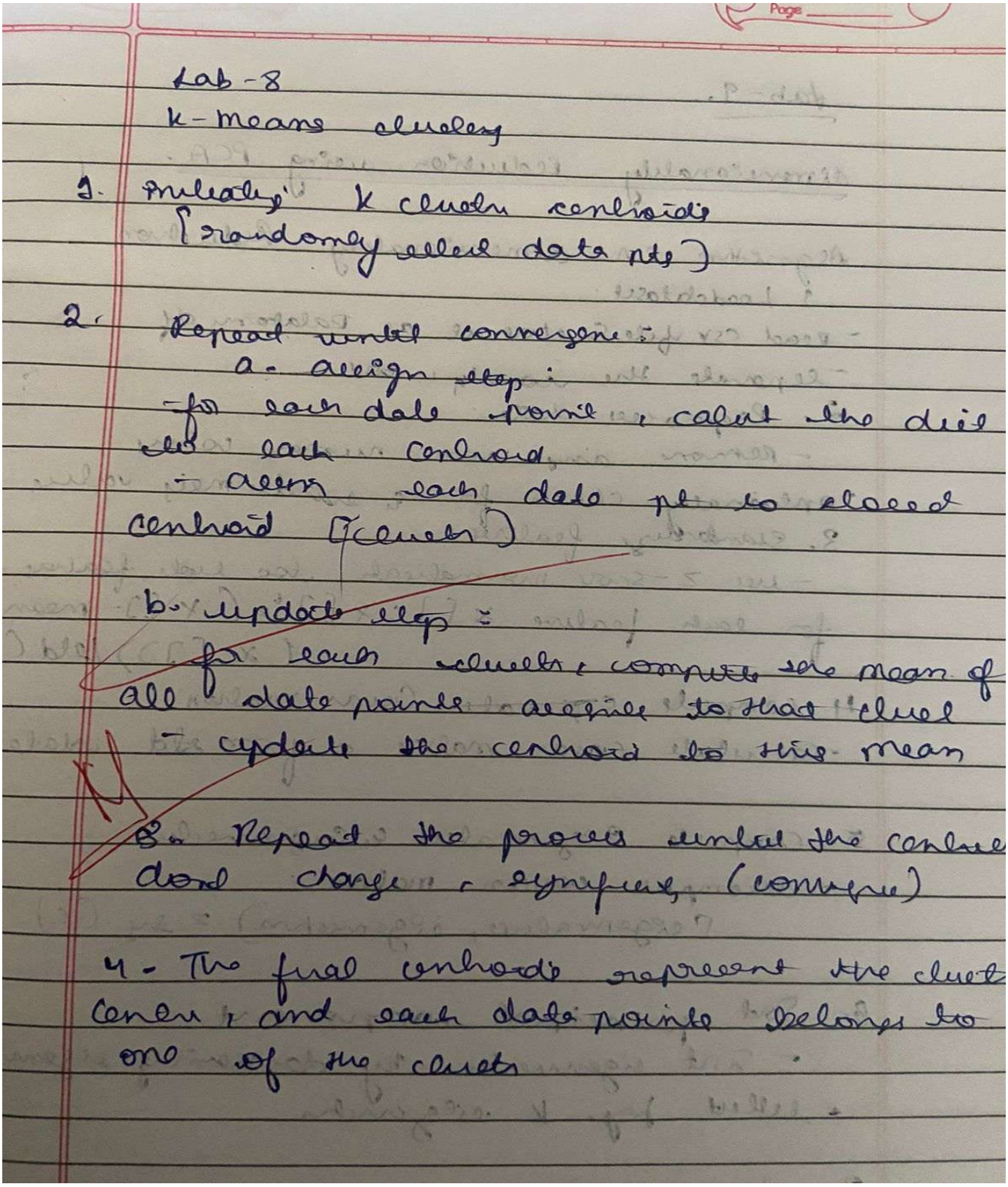
```
SVM Accuracy: 1.0

Classification Report:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        19
Iris-versicolor       1.00      1.00      1.00        13
 Iris-virginica       1.00      1.00      1.00        13

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



Program 8

Implement Random forest ensemble method on a given dataset.

Screenshots



Code

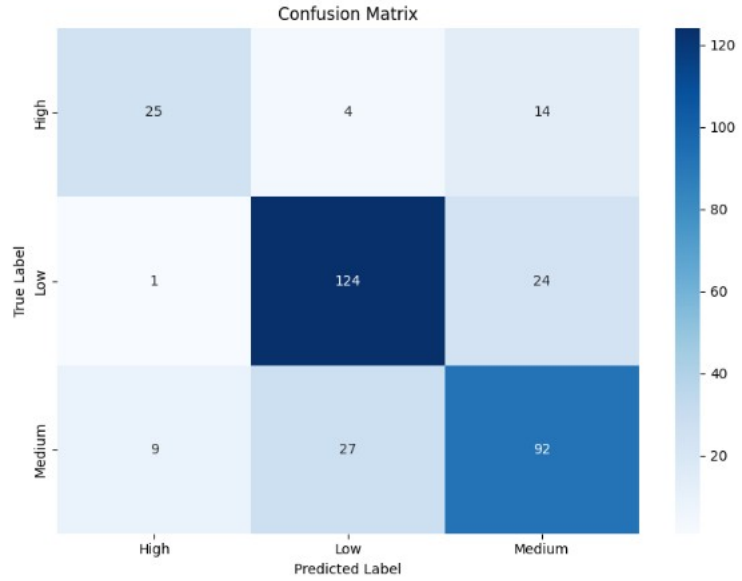
```
import pandas as pd
# Load the dataset directly from the URL
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv"
df = pd.read_csv(url, sep=';')  # Use ';' as separator
# Preview the data
print(df.head())

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Load the dataset
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv"
df = pd.read_csv(url, sep=';')  # Note: separator is a semicolon

# Check basic info
print(" 📄 Dataset shape:", df.shape)
print(df.head())

# Convert the quality score to categories (optional: 3 levels for better balance)
# You can change this logic based on how granular you want the classification
def quality_label(q):
    if q <= 5:
        return "Low"
    elif q == 6:
        return "Medium"
```

```
    else:
        return "High"

df['Quality_Label'] = df['quality'].apply(quality_label)

# Separate features and target
X = df.drop(columns=['quality', 'Quality_Label'])
y = df['Quality_Label']

# Encode the target labels
le = LabelEncoder()
y_encoded = le.fit_transform(y)

# Split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y_encoded, test_size=0.2, random_state=42, stratify=y_encoded
)

# Train the Random Forest classifier
rf_model = RandomForestClassifier(n_estimators=100, max_depth=None, random_state=42)
rf_model.fit(X_train, y_train)

# Predict on test data
y_pred = rf_model.predict(X_test)

# Evaluate the model
print("☑ Accuracy:", accuracy_score(y_test, y_pred))
print("\n📊 Classification Report:\n")
print(classification_report(y_test, y_pred, target_names=le.classes_))

# Plot the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
```

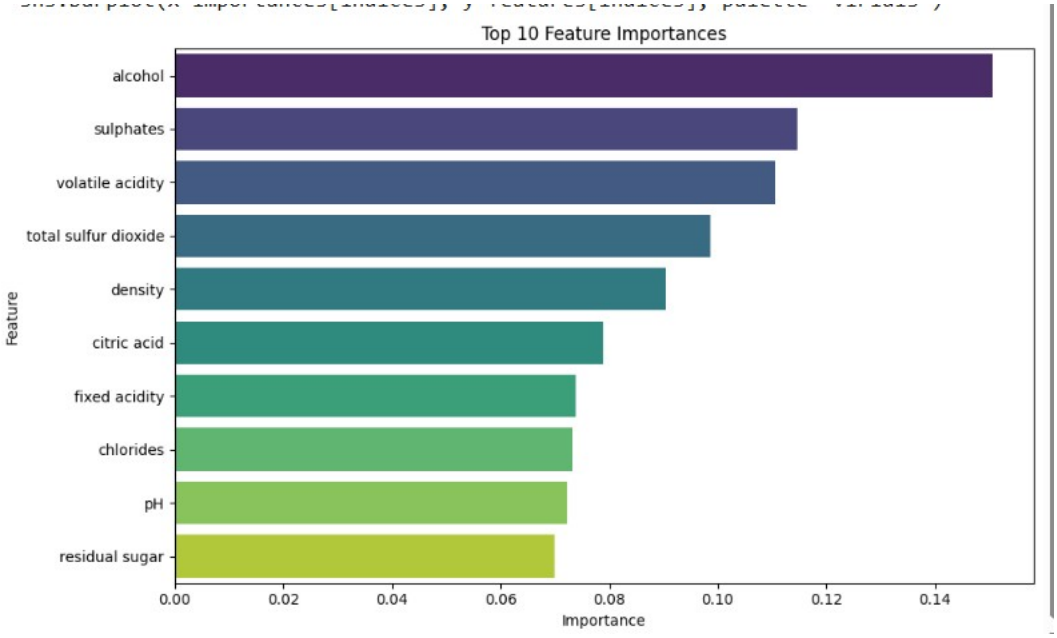
```
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=le.classes_, yticklabels=le.classes_)
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.tight_layout()
plt.show()

# Plot top 10 feature importances
importances = rf_model.feature_importances_
features = X.columns
indices = importances.argsort()[::-1][:10]  # Top 10

plt.figure(figsize=(10, 6))
sns.barplot(x=importances[indices], y=features[indices], palette="viridis")
plt.title("Top 10 Feature Importances")
plt.xlabel("Importance")
plt.ylabel("Feature")
plt.tight_layout()
plt.show()
```

Output:

```
Dataset shape: (1599, 12)
   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0            7.4              0.70         0.00             1.9      0.076
1            7.8              0.88         0.00             2.6      0.098
2            7.8              0.76         0.04             2.3      0.092
3           11.2              0.28         0.56             1.9      0.075
4            7.4              0.70         0.00             1.9      0.076

   free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                 11.0                  34.0   0.9978  3.51       0.56
1                 25.0                  67.0   0.9968  3.20       0.68
2                 15.0                  54.0   0.9970  3.26       0.65
3                 17.0                  60.0   0.9980  3.16       0.58
4                 11.0                  34.0   0.9978  3.51       0.56

   alcohol  quality
0      9.4        5
1      9.8        5
2      9.8        5
3      9.8        6
4      9.4        5
Accuracy: 0.753125

Classification Report:

              precision    recall  f1-score   support

        High       0.71      0.58      0.64        43
         Low       0.80      0.83      0.82       149
      Medium       0.71      0.72      0.71       128

    accuracy                           0.75       320
   macro avg       0.74      0.71      0.72       320
weighted avg       0.75      0.75      0.75       320
```



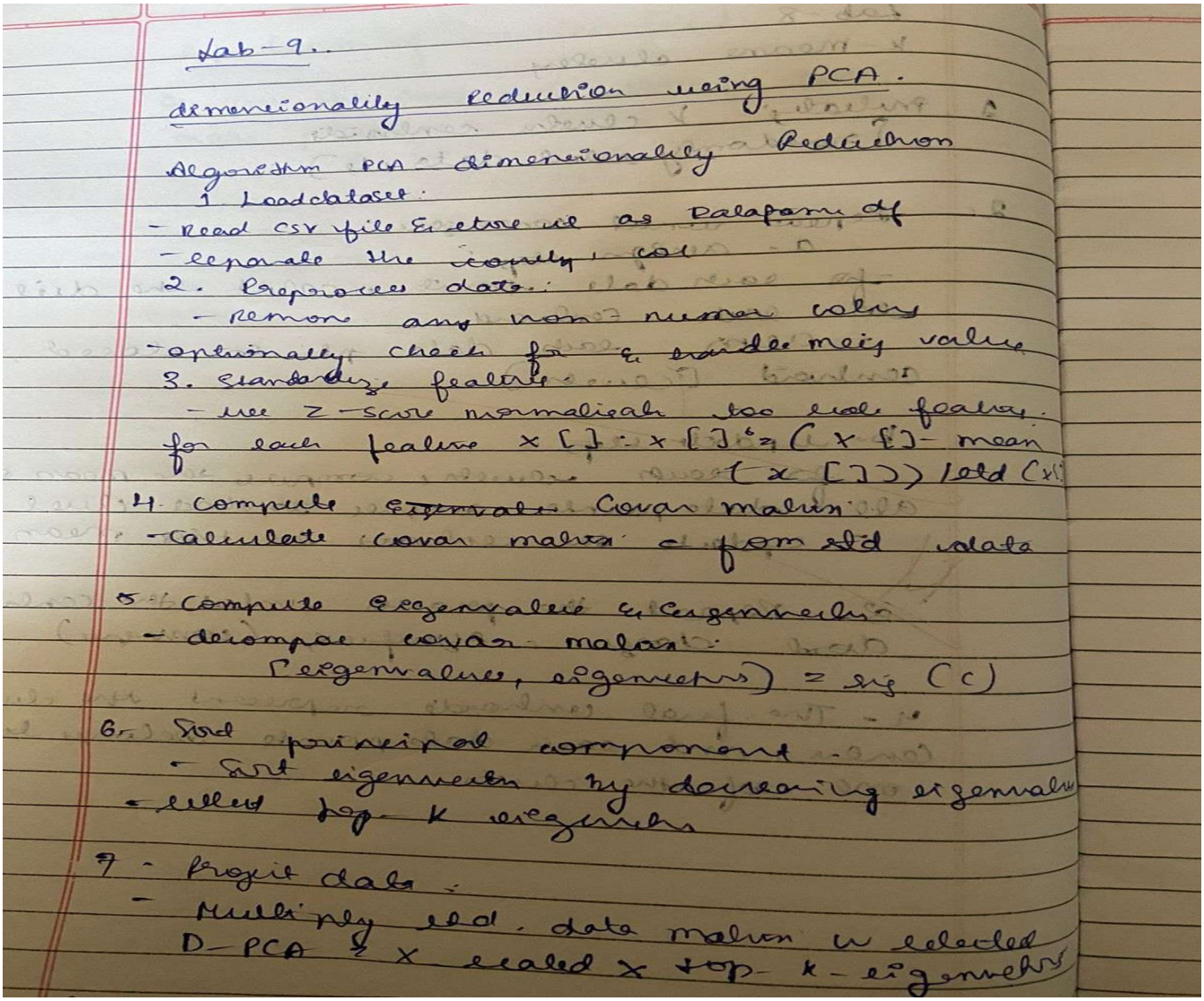

```
<ipython-input-3-e78bc36e6dec>:71: FutureWarning:
```



Program 9

Implement Boosting ensemble method on a given dataset.

Screenshots



Code

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


import zipfile
import os
# Use the uploaded filename (replace if different)
zip_path = "hmboost.zip"
extract_path = "/content/hmboost"
# Extract
with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_path)
# List files
os.listdir(extract_path)



import pandas as pd
import os
# List files inside the extracted folder
extracted_path = "/content/hmboost"
files = os.listdir(extracted_path)
print("Extracted files:", files)



import pandas as pd
import os
# Define the path
extracted_path = "/content/hmboost"
csv_file = os.path.join(extracted_path, 'train.csv')
# Load the training data
```

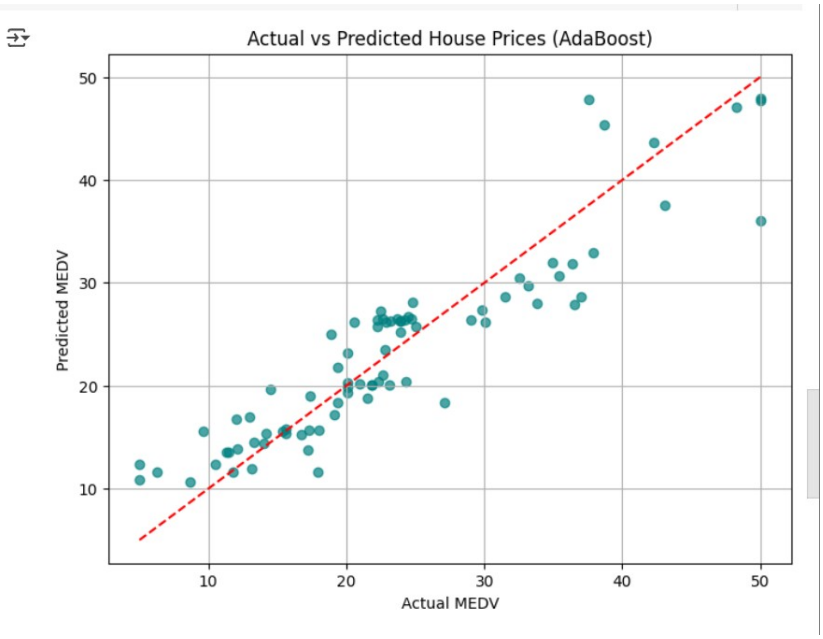
```
df = pd.read_csv(csv_file)
# Show the first few rows
df.head()


from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostRegressor
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np
# Drop unnecessary index column
df = df.drop(columns=['Unnamed: 0'])
# Separate features and target
X = df.drop('MEDV', axis=1)
y = df['MEDV']
# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Train AdaBoost Regressor
model = AdaBoostRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
# Predict
y_pred = model.predict(X_test)
# Evaluation
print("R² Score:", r2_score(y_test, y_pred))
print("RMSE:", np.sqrt(mean_squared_error(y_test, y_pred)))


import matplotlib.pyplot as plt
plt.figure(figsize=(8,6))
plt.scatter(y_test, y_pred, alpha=0.7, color='teal')
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'r--')  # ideal prediction line
plt.xlabel("Actual MEDV")
plt.ylabel("Predicted MEDV")
plt.title("Actual vs Predicted House Prices (AdaBoost)")
plt.grid(True)
```

plt.show()

Output:

R^2 Score: 0.8469189116196101

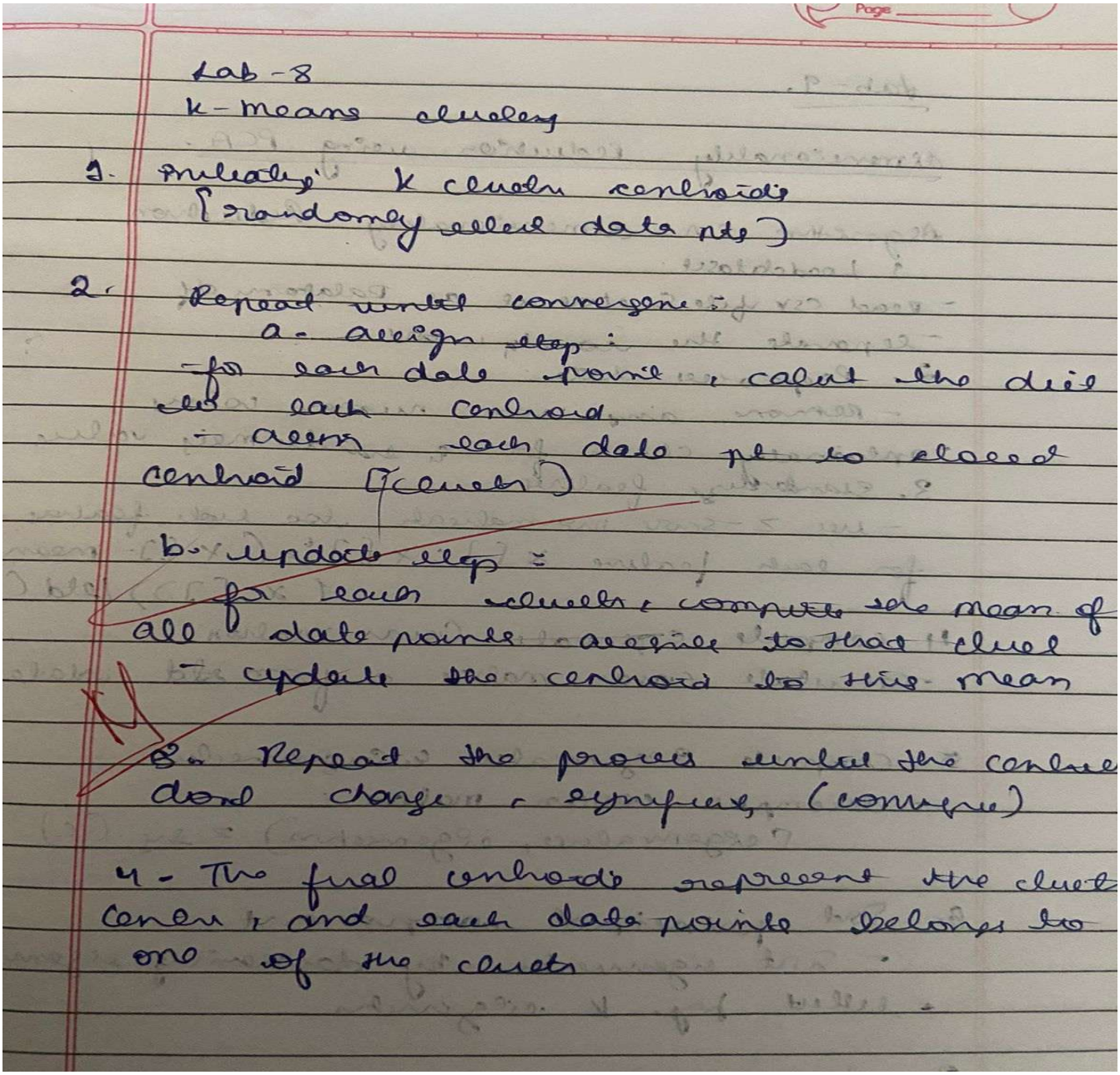
RMSE: 4.070084114697759



Program 10

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

Screenshots



Code

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


# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler


import zipfile
import os
# Path to the uploaded zip file
zip_file_path = 'archive.zip'
# Extract the zip file
with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    zip_ref.extractall()
# List the files in the extracted folder
extracted_files = os.listdir()
extracted_files


import pandas as pd
# Load the dataset
df = pd.read_csv('Wholesale customers data.csv')
# Display the first few rows to check if the data is loaded correctly
df.head()
```

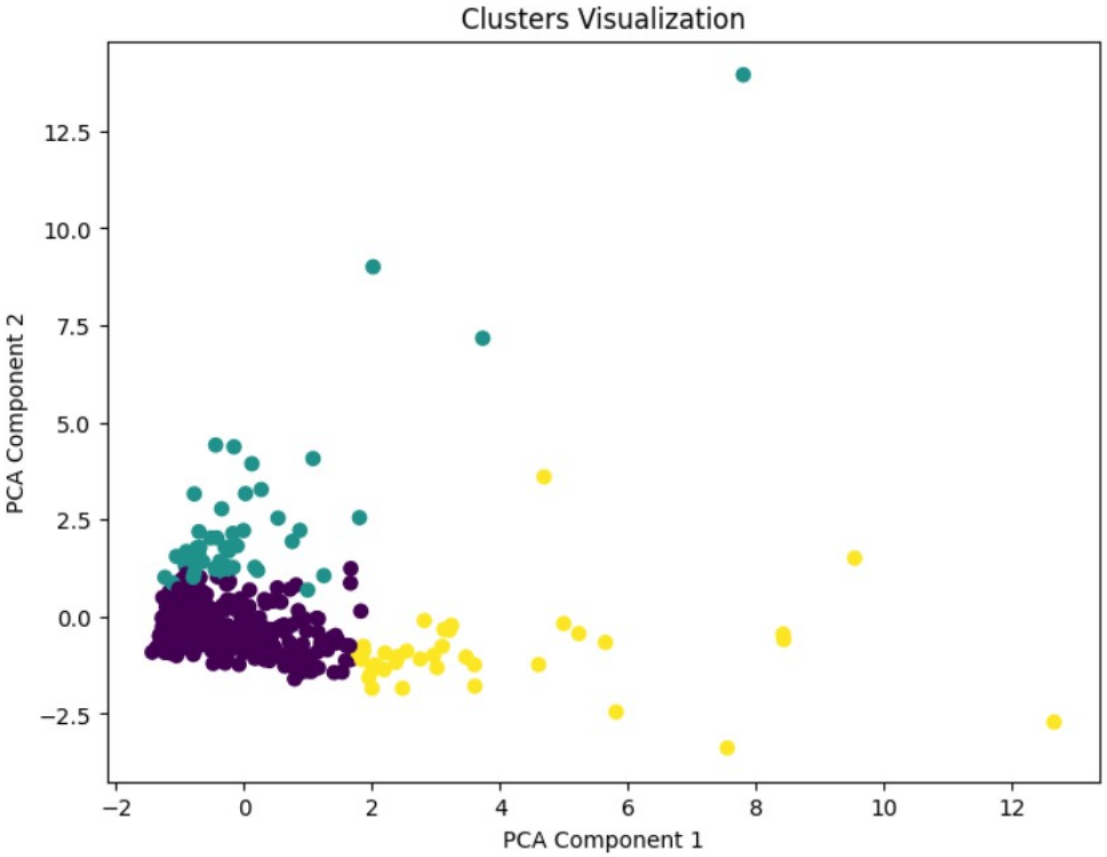
```
# Select the numerical features (drop 'Channel' and 'Region')
X = df.drop(['Channel', 'Region'], axis=1)
# Standardize the features
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Check the scaled data
X_scaled[:5]


from sklearn.cluster import KMeans
# Apply KMeans clustering
kmeans = KMeans(n_clusters=3, random_state=42)
y_kmeans = kmeans.fit_predict(X_scaled)
# Add the cluster labels to the original dataframe
df['Cluster'] = y_kmeans
# Display the first few rows with the cluster labels
df.head()

# Display the cluster centers (in the scaled space)
cluster_centers = kmeans.cluster_centers_
print('Cluster Centers (Scaled Data):\n', cluster_centers)
# Group by cluster and calculate the mean of each cluster
cluster_summary = df.groupby('Cluster').mean()
print(cluster_summary)



from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
# Perform PCA to reduce the data to 2D
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
```

```
# Plot the clusters
plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=df['Cluster'], cmap='viridis')
plt.title('Clusters Visualization')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.show()
```

```
# Centroids of each cluster (in scaled space)
centroids = kmeans.cluster_centers_
# Display the centroids
centroids_df = pd.DataFrame(centroids, columns=X.columns)
print(centroids_df)

# Reverse the scaling to get the centroids in the original units
centroids_original = scaler.inverse_transform(centroids)
# Create a DataFrame with original units
centroids_original_df = pd.DataFrame(centroids_original, columns=X.columns)
print(centroids_original_df)
```

Output:

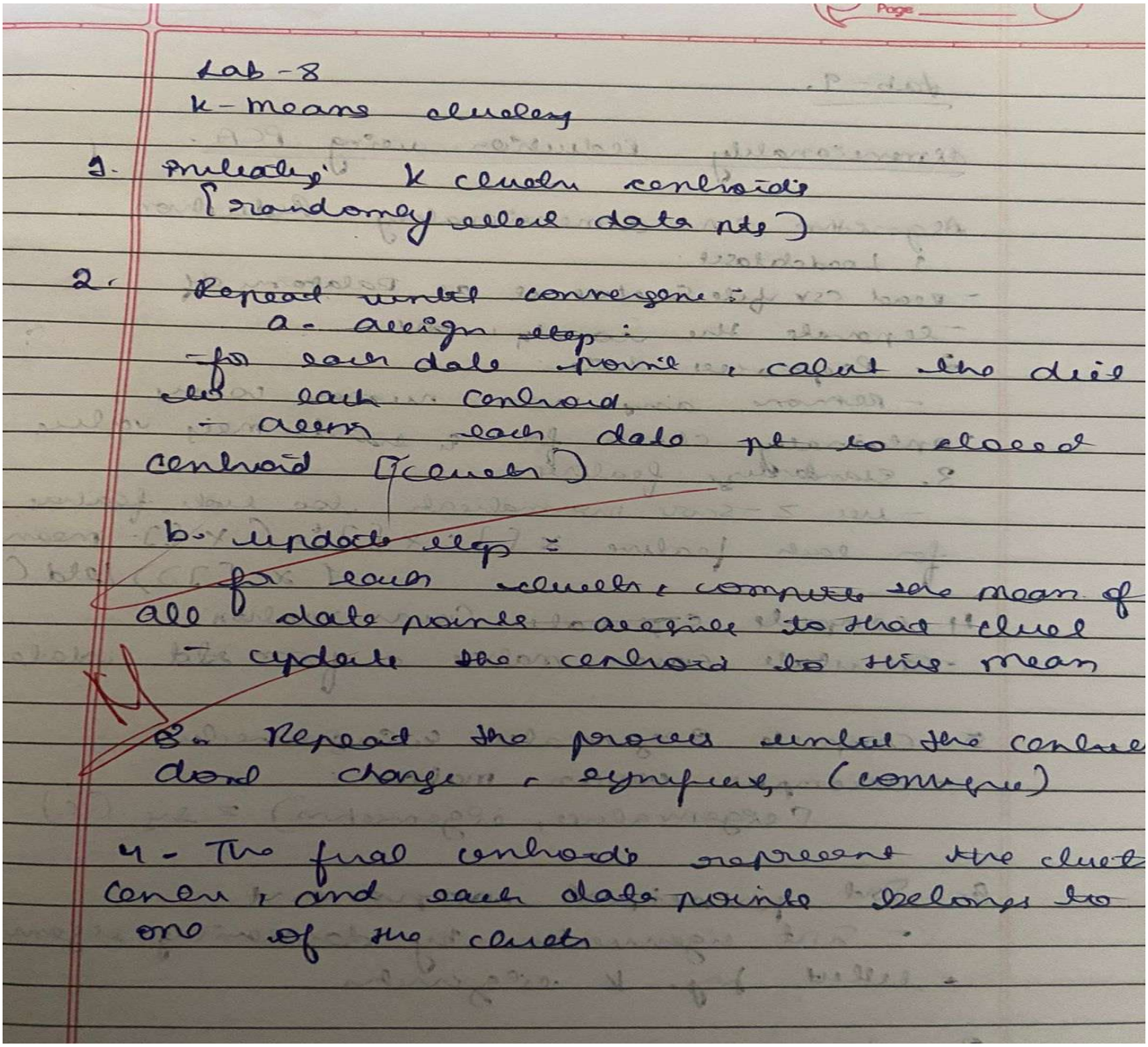
```
Cluster Centers (Scaled Data):
 [[-0.24260351 -0.21266148 -0.22157224 -0.18656872 -0.20323381 -0.15007652]
 [ 1.78420858  0.00869407 -0.19264494  1.39607954 -0.39896023  0.75947859]
 [-0.26086015  1.99920893  2.37190452 -0.23495033  2.49396554  0.33174095]]
          Channel    Region         Fresh          Milk       Grocery  \
Cluster
0        1.282857  2.534286   8935.500000   4228.528571   5848.034286
1        1.113208  2.698113  34540.113208   5860.358491   6122.622642
2        2.000000  2.405405   8704.864865  20534.405405  30466.243243

              Frozen  Detergents_Paper   Delicassen
Cluster
0        2167.231429       1913.605714  1102.120000
1        9841.735849        981.471698  3664.245283
2        1932.621622      14758.837838  2459.351351
```


Clusters Visualization
PCA Component 2
PCA Component 1
12.5
10.0
7.5
5.0
2.5
0.0
−2.5
−2
0
2
4
6
8
10
12

Program 11

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

Screenshots



Code

Heart Dataset

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

# Load dataset
df = pd.read_csv('heart.csv')
print("Columns:", df.columns)

# Check if 'target' is named differently
target_col = 'target' if 'target' in df.columns else df.columns[-1]

# Separate features and target
X = df.drop(target_col, axis=1)
y = df[target_col]

# One-hot encode categorical columns if any
cat_cols = X.select_dtypes(include=['object', 'category']).columns
X = pd.get_dummies(X, columns=cat_cols, drop_first=True)

# Scale features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Models
models = {
    'Logistic Regression': LogisticRegression(max_iter=1000),
    'SVM': SVC(),
```

```
    'Random Forest': RandomForestClassifier()
}

# Evaluate models
print("\nOriginal Accuracy:")
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    print(f"{name}: {accuracy_score(y_test, y_pred):.4f}")

Columns: Index(['Age', 'Sex', 'ChestPainType', 'RestingBP', 'Cholesterol', 'FastingBS',
       'RestingECG', 'MaxHR', 'ExerciseAngina', 'Oldpeak', 'ST_Slope',
       'HeartDisease'],
      dtype='object')
pca = PCA(n_components=0.95)
X_pca = pca.fit_transform(X_scaled)

X_train_pca, X_test_pca, y_train_pca, y_test_pca = train_test_split(X_pca, y, test_size=0.2,
random_state=42)

# Evaluate after PCA
print("\nAccuracy After PCA:")
for name, model in models.items():
    model.fit(X_train_pca, y_train_pca)
    y_pred_pca = model.predict(X_test_pca)
    print(f"{name}: {accuracy_score(y_test_pca, y_pred_pca):.4f}")
```

Output:

```
Columns: Index(['Age', 'Sex', 'ChestPainType', 'RestingBP', 'Cholesterol',
'FastingBS',
       'RestingECG', 'MaxHR', 'ExerciseAngina', 'Oldpeak', 'ST_Slope',
       'HeartDisease'],
      dtype='object')

Original Accuracy:
Logistic Regression: 0.8533
SVM: 0.8750
Random Forest: 0.8696

Accuracy After PCA:
Logistic Regression: 0.8533
SVM: 0.8750
Random Forest: 0.8478
```