# VISVESVARAYA TECHNOLOGICAL UNIVERSITY

**“JnanaSangama”, Belgaum -590014, Karnataka.**

****

## LAB REPORT

### on

Machine Learning (23CS6PCMAL)

#### Submitted by

**Chahat Singh(1BM22CS075)**

#### in partial fulfillment for the award of the degree of

**BACHELOR OF ENGINEERING**

***in***

## COMPUTER SCIENCE AND ENGINEERING

****

**B.M.S. COLLEGE OF ENGINEERING**

**(Autonomous Institution under VTU)**

## BENGALURU-560019

### Sep-2024 to Jan-2025

**B.M.S. College of Engineering,**

**Bull Temple Road, Bangalore 560019**

(Affiliated To Visvesvaraya Technological University, Belgaum)

**Department of Computer Science and Engineering**

****

##### CERTIFICATE

This is to certify that the Lab work entitled “Machine Learning (23CS6PCMAL)” carried out by **Chahat Singh(1BM22CS075),** 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.

Dr. Kavitha Sooda Professor & HOD

Department of CSE, BMSCE

[Megha J](https://bmsce.ac.in/home/facultyProfile/649/Supreeth-S)

Assistant Professor Department of CSE, BMSCE

# Index

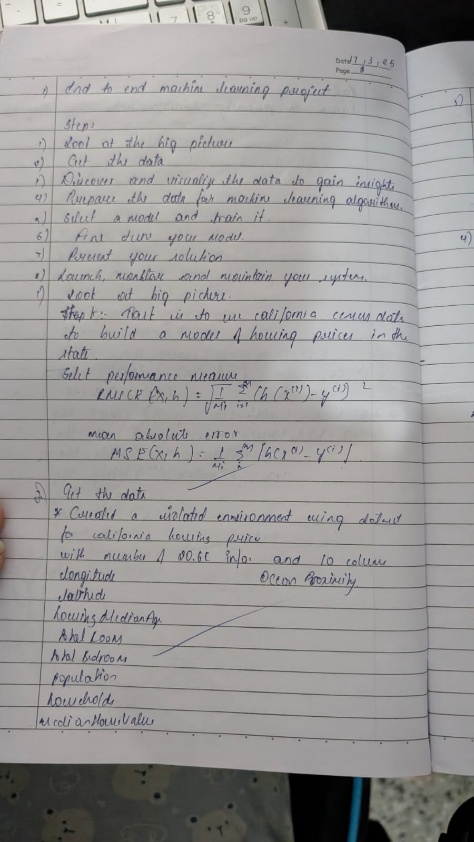
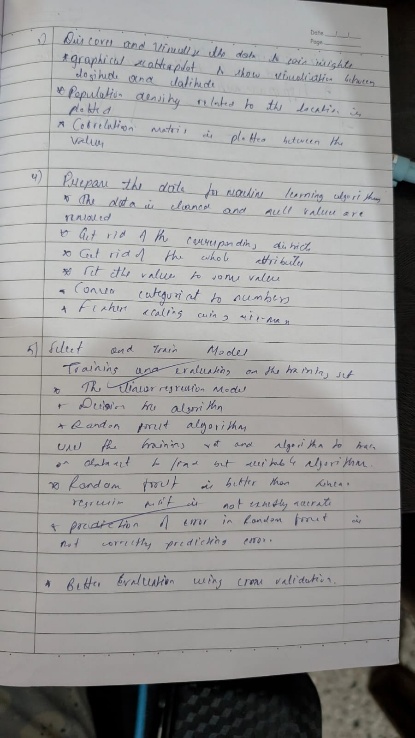
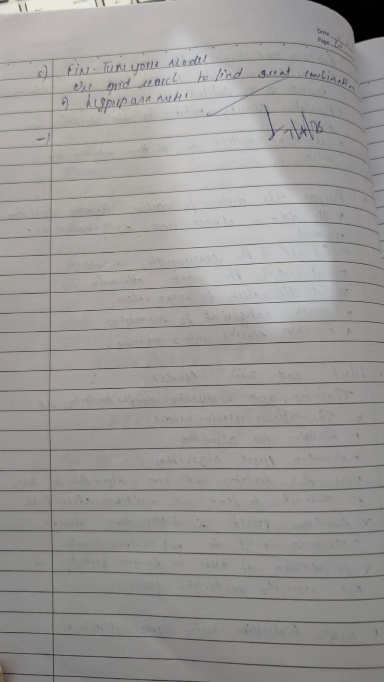
|  |  |  |  |
| --- | --- | --- | --- |
| **Sl.**  **No.** | **Date** | **Experiment Title** | **Page No.** |
| 1 | 4-3-2025 | Write a python program to import and export data using Pandas library functions | 1-10 |
| 2 | 11-3-2025 | Demonstrate various data pre-processing techniques for a given dataset | 1-15 |
| 3 | 18-3-2025 | Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample. | 15-17 |
| 4 | 1-4-2025 | Implement Linear and Multi-Linear Regression algorithm using appropriate dataset | 17-20 |
| 5 | 8-4-2025 | Build Logistic Regression Model for a given dataset | 20-25 |
| 6 | 15-4-2025 | Build KNN Classification model for a given dataset. | 25-28 |
| 7 | 15-4-2025 | Build Support vector machine model for a given dataset | 29-33 |
| 8 | 22-4-2025 | Implement Random forest ensemble method on a given dataset. | 33-38 |
| 9 | 22-4-2025 | Implement Boosting ensemble method on a given dataset. | 38-45 |
| 10 | 29-4-2025 | Build k-Means algorithm to cluster a set of data stored in a .CSV file. | 45-53 |
| 11 | 29-4-2025 | Implement Dimensionality reduction using Principal Component Analysis (PCA) method. | 53-58 |

Github Link: https://github.com/chahat20singh/ML-LAB

**Program 1**

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

**Screenshots**

**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

target

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 \

0 502 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

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=False)

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 33.82 28.0 67.0 15.0

4 -119.67 36.33 19.0 1241.0 244.0

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 -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 33.82 28.0 67.0 15.0

4 -119.67 36.33 19.0 1241.0 244.0

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

...

21988.0 4055.0

23915.0 4135.0

24121.0 4522.0

27870.0 5027.0

30450.0 5033.0

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)

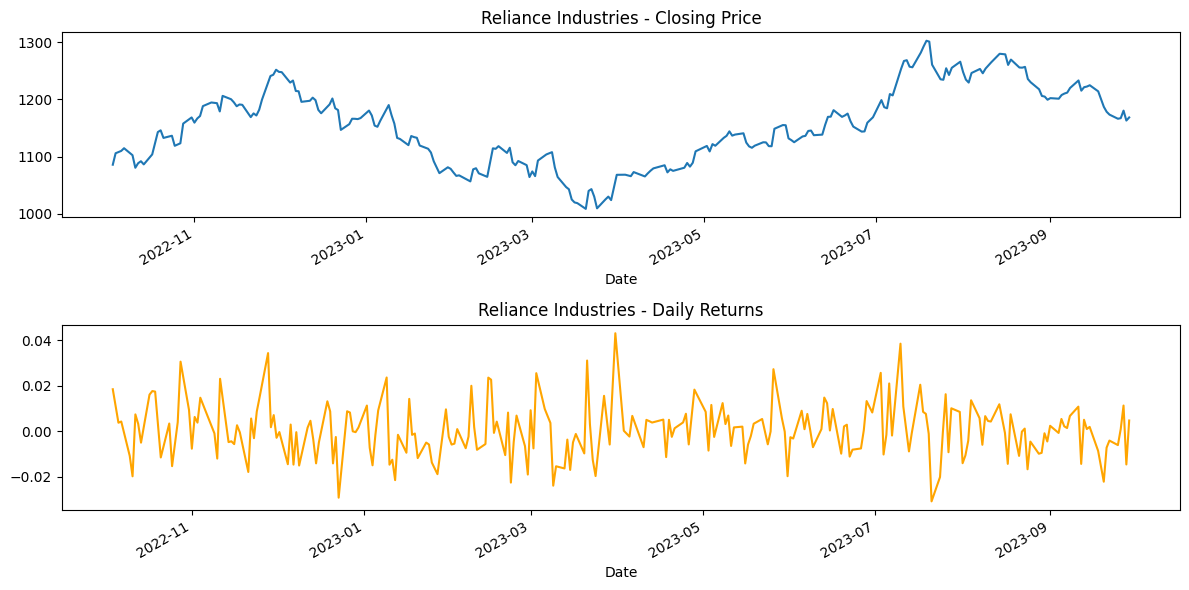
reliance\_data['Close'].plot(title="Reliance Industries - Closing Price")

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

2024-01-01 983.086778 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

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)

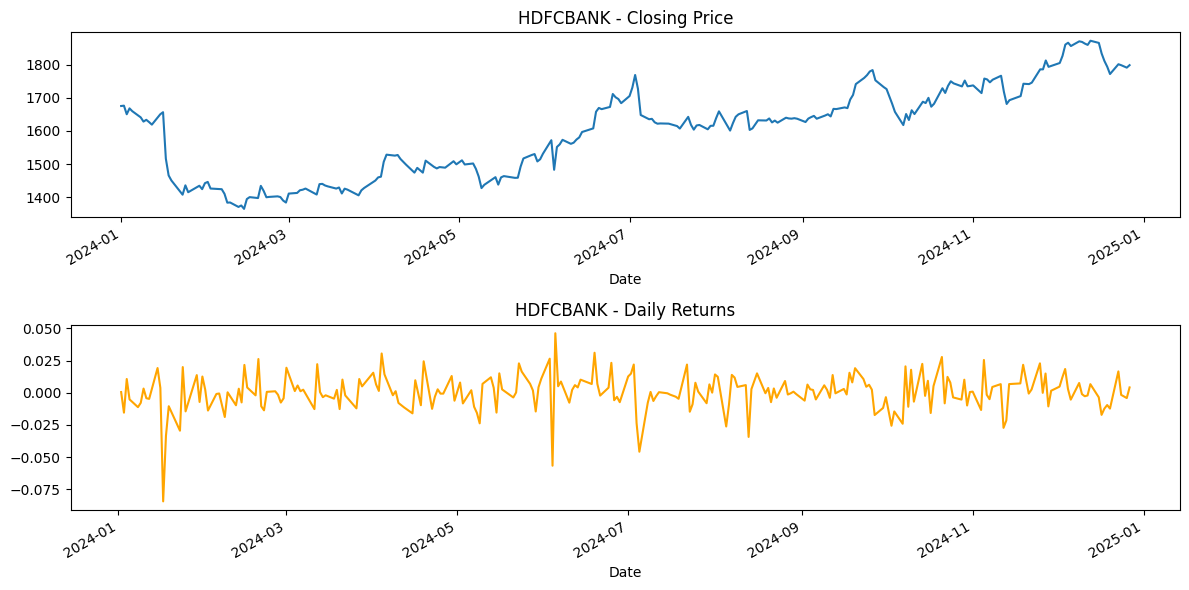
HDFC['Close'].plot(title="HDFCBANK - Closing Price")

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)

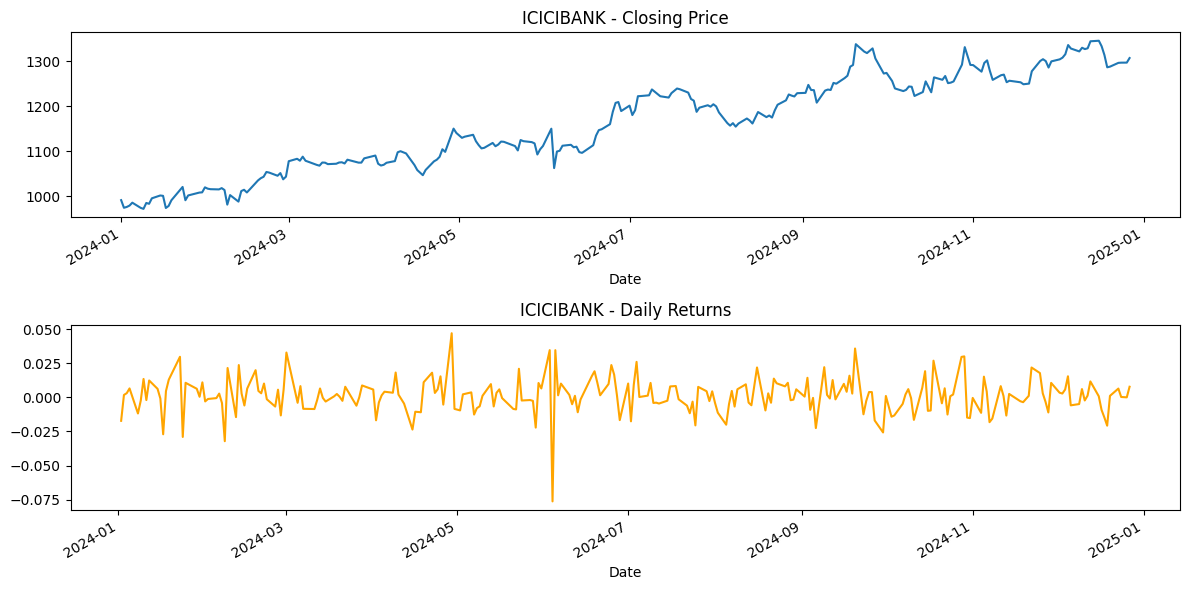
ICICI['Close'].plot(title="ICICIBANK - Closing Price")

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)

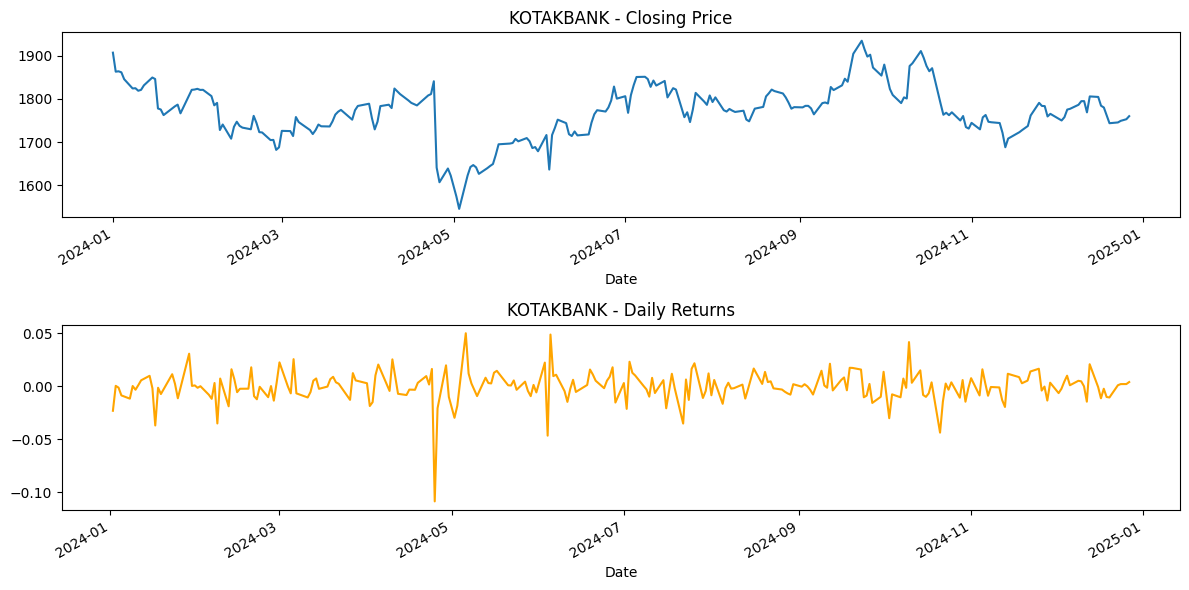
KOTAKBANK['Close'].plot(title="KOTAKBANK - Closing Price")

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('KOTAKBANK.csv')

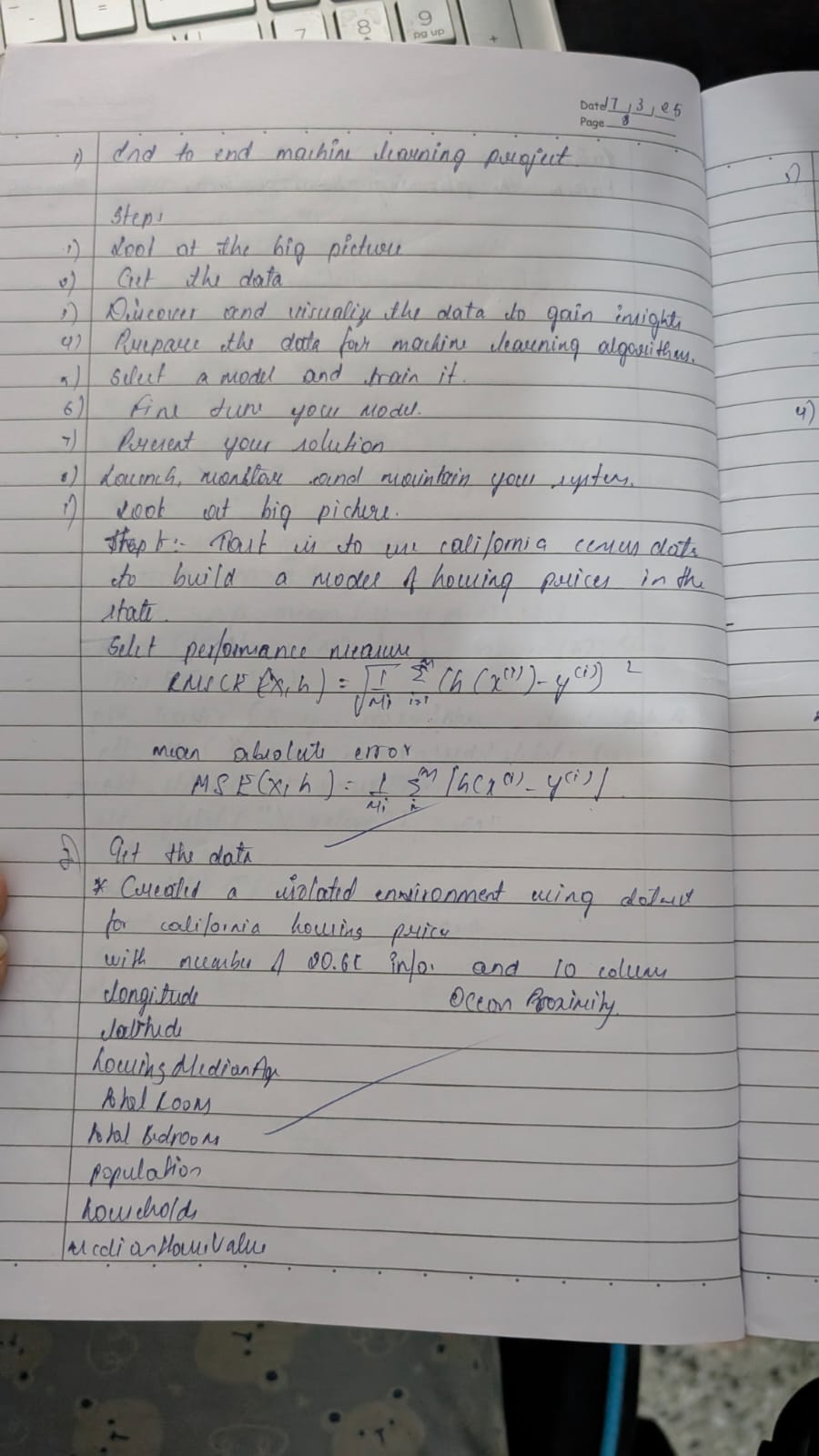
print("\nSAVED")

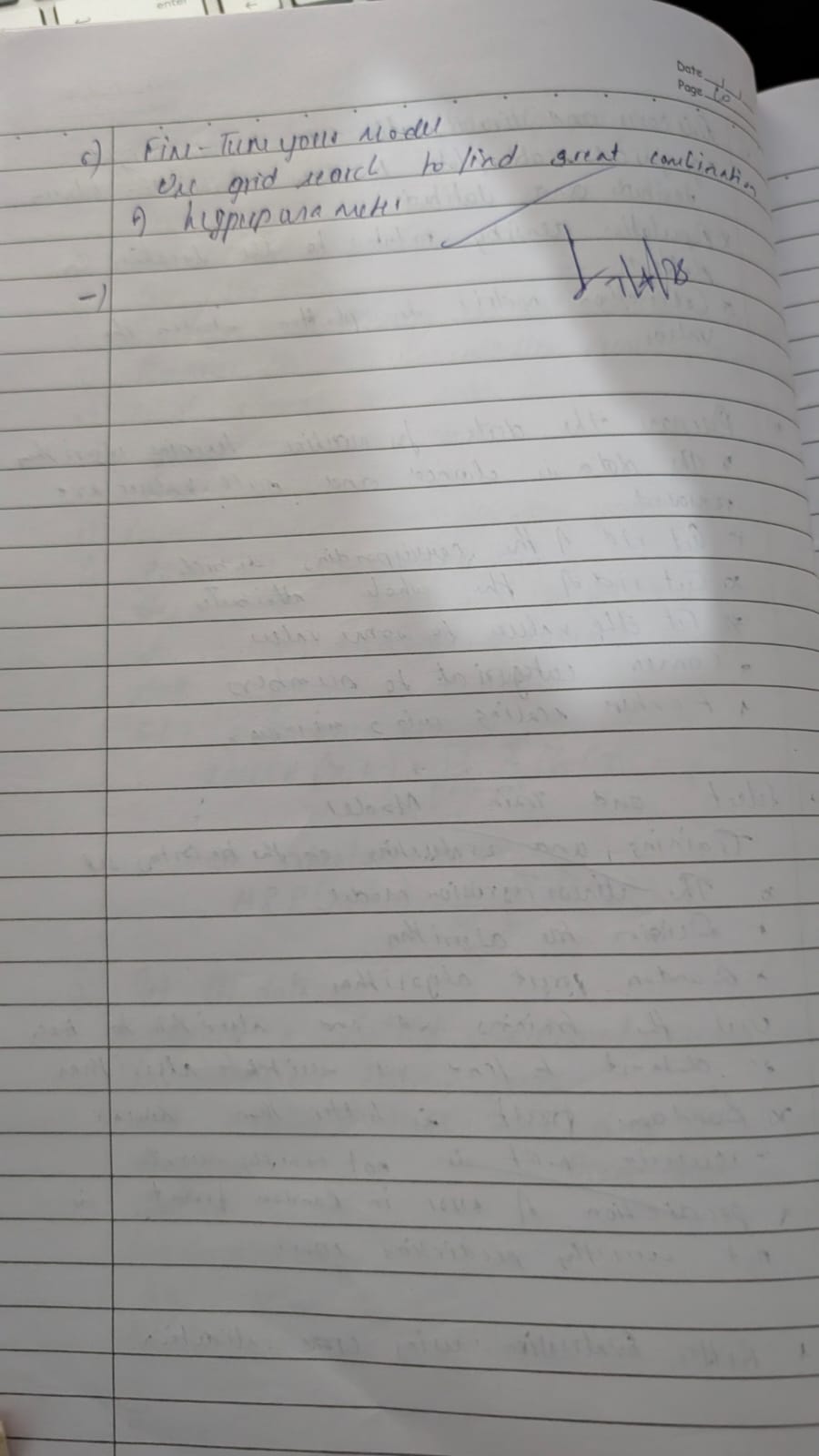
SAVED

**Program 2**

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

**Screenshots**





**Code**:

import pandas as pd # Create a DataFrame directly from a dictionary

data = { 'Name': ['Alice', 'Bob', 'Charlie', 'David'],

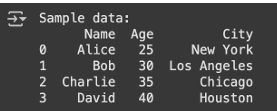
'Age': [25, 30, 35, 40],

'City': ['New York', 'Los Angeles', 'Chicago', 'Houston'] }

df = pd.DataFrame(data)

print("Sample data:")

print(df.head())

****

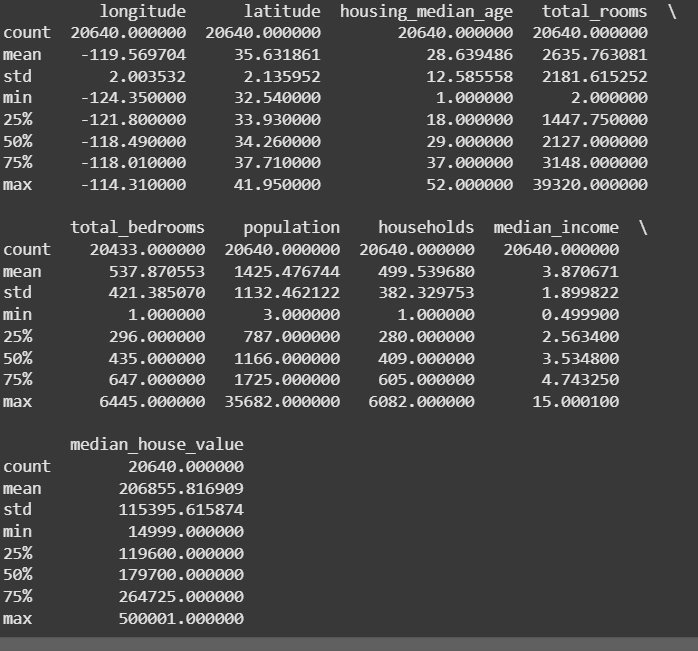
#To display information of all columns

print(df.info)



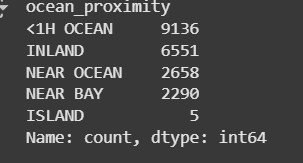
#To display statistical information of all numerical

print(df.describe())



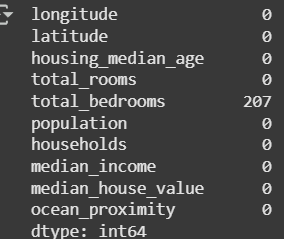
#To display the count of unique labels for “Ocean Proximity” column

print(df['ocean\_proximity'].value\_counts())



#To display which attributes (columns) in a dataset have missing values count greater than zero

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

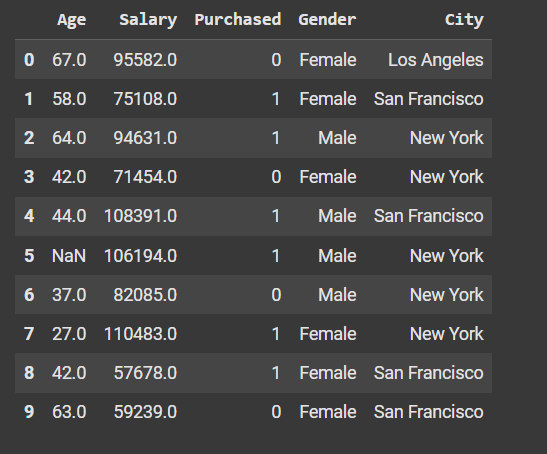


# Introduce some missing values for demonstration

df.loc[5, 'Age'] = np.nan

df.loc[10, 'Salary'] = np.nan

df.head(10)



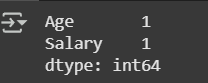
#Code to Find Missing Values

# Check for missing values in each column

missing\_values = df.isnull().sum()

# Display columns with missing values

print(missing\_values[missing\_values > 0])



#Set the values to some value (zero, the mean, the median, etc.).

# Step 1: Create an instance of SimpleImputer with the median strategy for Age and mean stratergy for Salary

imputer1 = SimpleImputer(strategy="median")

imputer2 = SimpleImputer(strategy="mean")

df\_copy=df

# Step 2: Fit the imputer on the "Age" and "Salary"column

# Note: SimpleImputer expects a 2D array, so we reshape the column

imputer1.fit(df\_copy[["Age"]])

imputer2.fit(df\_copy[["Salary"]])

# Step 3: Transform (fill) the missing values in the "Age" and "Salary"c column

df\_copy["Age"] = imputer1.transform(df[["Age"]])

df\_copy["Salary"] = imputer2.transform(df[["Salary"]])

# Verify that there are no missing values left

print(df\_copy["Age"].isnull().sum())

print(df\_copy["Salary"].isnull().sum())

#Handling Categorical Attributes

#Using Ordinal Encoding for gender COlumn and One-Hot Encoding for City Column

# Initialize OrdinalEncoder

ordinal\_encoder = OrdinalEncoder(categories=[["Male", "Female"]])

# Fit and transform the data

df\_copy["Gender\_Encoded"] = ordinal\_encoder.fit\_transform(df\_copy[["Gender"]])

# Initialize OneHotEncoder

onehot\_encoder = OneHotEncoder()

# Fit and transform the "City" column

encoded\_data = onehot\_encoder.fit\_transform(df[["City"]])

# Convert the sparse matrix to a dense array

encoded\_array = encoded\_data.toarray()

# Convert to DataFrame for better visualization

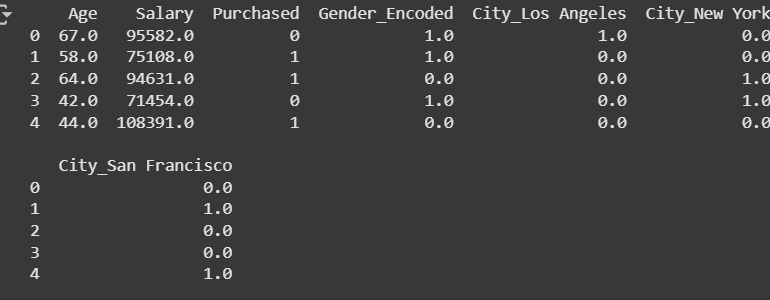
encoded\_df = pd.DataFrame(encoded\_array, columns=onehot\_encoder.get\_feature\_names\_out(["City"]))

df\_encoded = pd.concat([df\_copy, encoded\_df], axis=1)

df\_encoded.drop("Gender", axis=1, inplace=True)

df\_encoded.drop("City", axis=1, inplace=True)

print(df\_encoded. head())



#Removing Outliers

# Z-score method

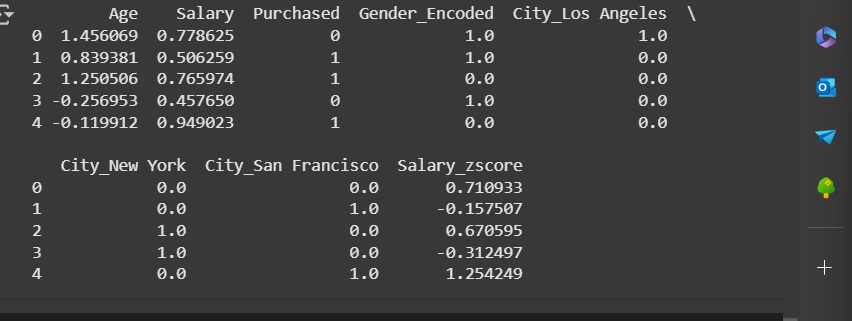
#Pros: Good for normally distributed data.

#Cons: Not suitable for non-normal data; may miss outliers in skewed distributions.

df\_encoded\_copy2['Salary\_zscore'] = stats.zscore(df\_encoded\_copy2['Salary'])

df\_encoded\_copy2['Salary'] = np.where(df\_encoded\_copy2['Salary\_zscore'].abs() > 3, np.nan, df\_encoded\_copy2['Salary'])  # Replace outliers with NaN

print(df\_encoded\_copy2.head())



#Removing Outliers

# Median replacement for outliers

#Pros: Keeps distribution shape intact, useful when capping isn’t feasible.

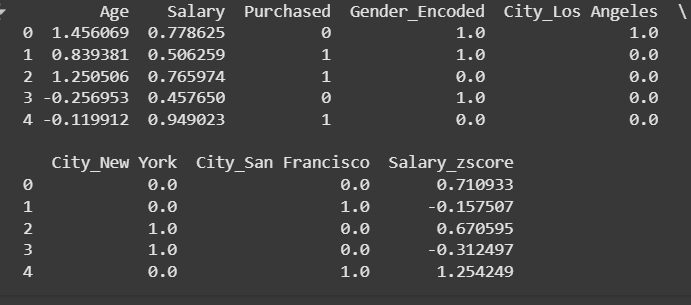
#Cons: May distort data if outliers represent real phenomena.

df\_encoded\_copy3['Salary\_zscore'] = stats.zscore(df\_encoded\_copy3['Salary'])

median\_salary = df\_encoded\_copy3['Salary'].median()

df\_encoded\_copy3['Salary'] = np.where(df\_encoded\_copy3['Salary\_zscore'].abs() > 3, median\_salary, df\_encoded\_copy3['Salary'])

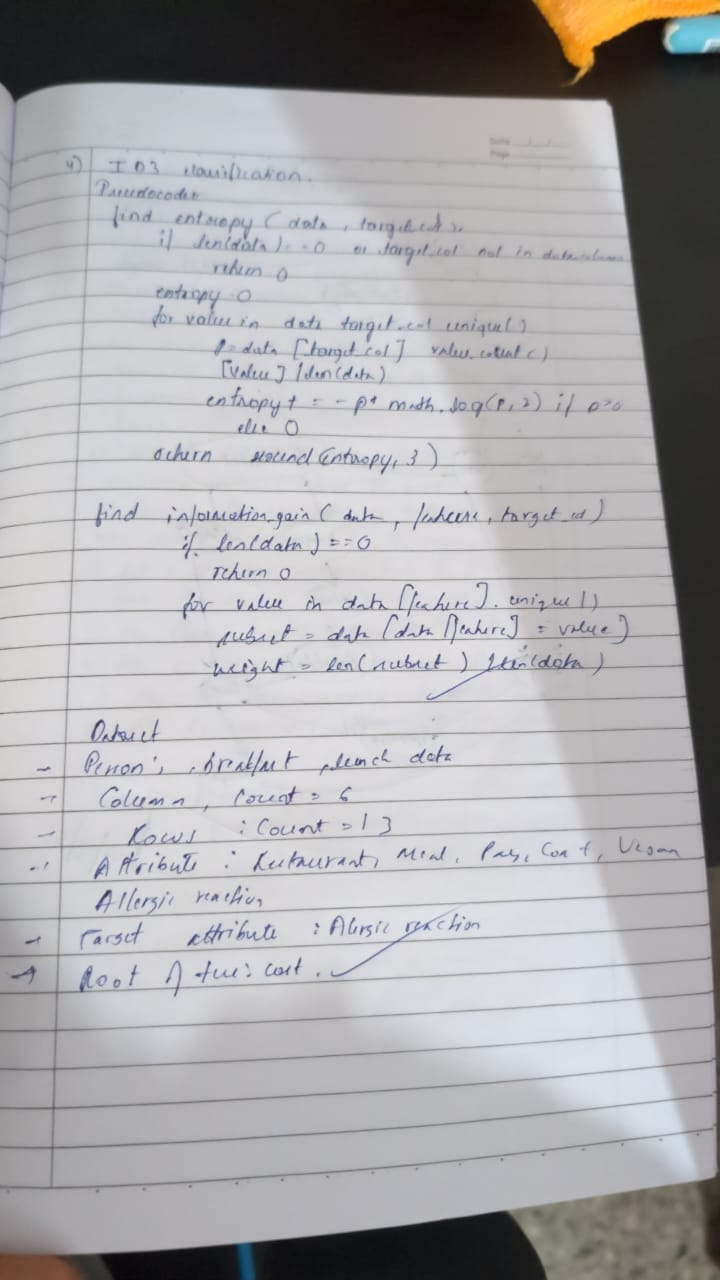
print(df\_encoded\_copy3.head())

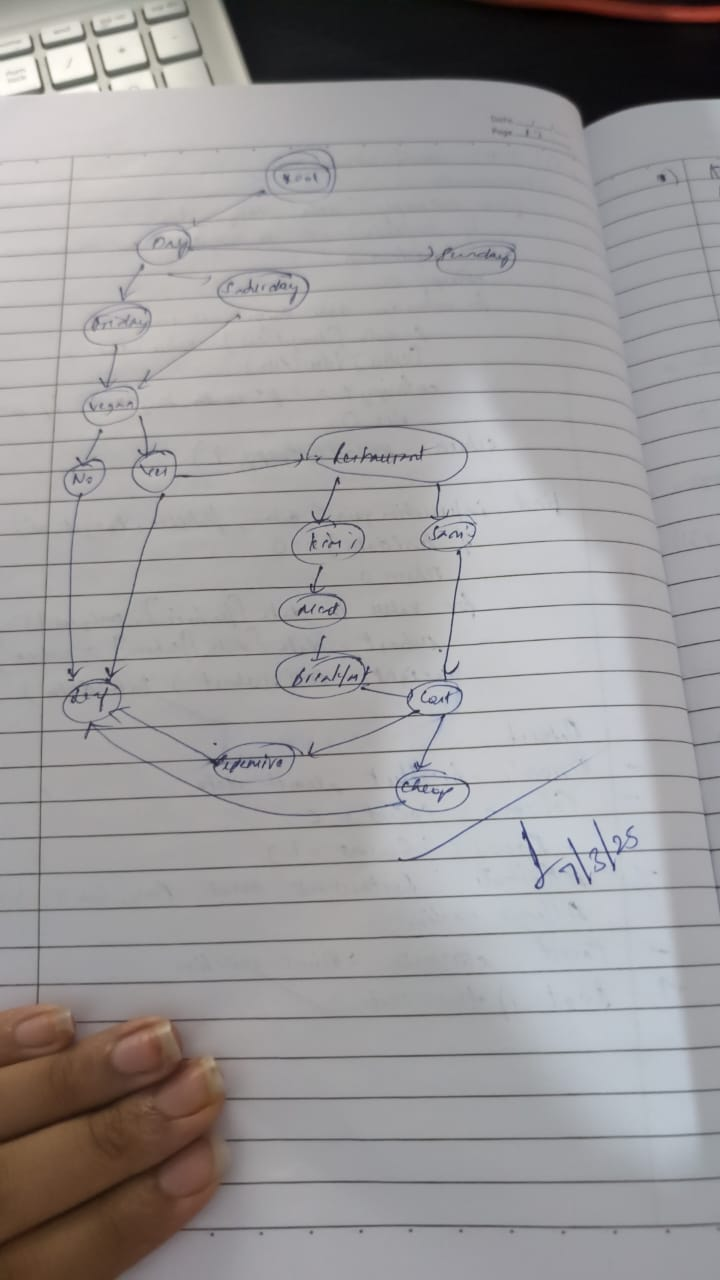


**Program 5**

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

**Screenshots**





**Code**

import pandas as pd  # for manipulating the csv data

import numpy as np    # for mathematical calculation

# Load dataset

train\_data\_m = pd.read\_csv("/content/PlayTennis.csv")

# Function to calculate total entropy of the dataset

def calc\_total\_entropy(train\_data, label, class\_list):

    total\_row = train\_data.shape[0]

    total\_entr = 0

    for c in class\_list:

        total\_class\_count = train\_data[train\_data[label] == c].shape[0]

        if total\_class\_count != 0:

            probability = total\_class\_count / total\_row

            total\_entr -= probability \* np.log2(probability)

    return total\_entr

# Function to calculate entropy for a subset of the data

def calc\_entropy(feature\_value\_data, label, class\_list):

    class\_count = feature\_value\_data.shape[0]

    entropy = 0

    for c in class\_list:

        label\_class\_count = feature\_value\_data[feature\_value\_data[label] == c].shape[0]

        if label\_class\_count != 0:

            probability\_class = label\_class\_count / class\_count

            entropy -= probability\_class \* np.log2(probability\_class)

    return entropy

# Function to calculate information gain for a feature

def calc\_info\_gain(feature\_name, train\_data, label, class\_list):

    feature\_value\_list = train\_data[feature\_name].unique()

    total\_row = train\_data.shape[0]

    feature\_info = 0.0

    for feature\_value in feature\_value\_list:

        feature\_value\_data = train\_data[train\_data[feature\_name] == feature\_value]

        feature\_value\_count = feature\_value\_data.shape[0]

        feature\_value\_entropy = calc\_entropy(feature\_value\_data, label, class\_list)

        feature\_info += (feature\_value\_count / total\_row) \* feature\_value\_entropy

    return calc\_total\_entropy(train\_data, label, class\_list) - feature\_info

# Find the most informative feature

def find\_most\_informative\_feature(train\_data, label, class\_list):

    feature\_list = train\_data.columns.drop(label)

    max\_info\_gain = -1

    max\_info\_feature = None

    for feature in feature\_list:

        feature\_info\_gain = calc\_info\_gain(feature, train\_data, label, class\_list)

        if max\_info\_gain < feature\_info\_gain:

            max\_info\_gain = feature\_info\_gain

            max\_info\_feature = feature

    return max\_info\_feature

# Generate subtree for a feature

def generate\_sub\_tree(feature\_name, train\_data, label, class\_list):

    feature\_value\_count\_dict = train\_data[feature\_name].value\_counts(sort=False)

    tree = {}

    rows\_to\_remove = []

    for feature\_value, count in feature\_value\_count\_dict.items():

        feature\_value\_data = train\_data[train\_data[feature\_name] == feature\_value]

        assigned\_to\_node = False

        for c in class\_list:

            class\_count = feature\_value\_data[feature\_value\_data[label] == c].shape[0]

            if class\_count == count:

                tree[feature\_value] = c

                rows\_to\_remove.append(feature\_value\_data.index)

                assigned\_to\_node = True

                break

        if not assigned\_to\_node:

            tree[feature\_value] = "?"

    train\_data = train\_data.drop(index=np.concatenate(rows\_to\_remove)) if rows\_to\_remove else train\_data

    return tree, train\_data

# Recursive tree-building function

def make\_tree(root, prev\_feature\_value, train\_data, label, class\_list):

    if train\_data.shape[0] != 0:

        max\_info\_feature = find\_most\_informative\_feature(train\_data, label, class\_list)

        if max\_info\_feature is None:

            return

        tree, updated\_train\_data = generate\_sub\_tree(max\_info\_feature, train\_data, label, class\_list)

        next\_root = None

        if prev\_feature\_value is not None:

            root[prev\_feature\_value] = {max\_info\_feature: tree}

            next\_root = root[prev\_feature\_value][max\_info\_feature]

        else:

            root[max\_info\_feature] = tree

            next\_root = root[max\_info\_feature]

        for node, branch in list(next\_root.items()):

            if branch == "?":

                feature\_value\_data = updated\_train\_data[updated\_train\_data[max\_info\_feature] == node]

                make\_tree(next\_root, node, feature\_value\_data, label, class\_list)

# ID3 entry point

def id3(train\_data\_m, label):

    train\_data = train\_data\_m.copy()

    tree = {}

    class\_list = train\_data[label].unique()

    make\_tree(tree, None, train\_data, label, class\_list)

    return tree

# Prediction function for a single instance

def predict(tree, instance):

    if not isinstance(tree, dict):

        return tree

    root\_node = next(iter(tree))

    feature\_value = instance[root\_node]

    if feature\_value in tree[root\_node]:

        return predict(tree[root\_node][feature\_value], instance)

    else:

        return None

# Evaluate the decision tree

def evaluate(tree, test\_data\_m, label):

    correct\_predict = 0

    wrong\_predict = 0

    for index, row in test\_data\_m.iterrows():

        result = predict(tree, row)

        actual = row[label]

        if result == actual:

            correct\_predict += 1

        else:

            wrong\_predict += 1

    total = correct\_predict + wrong\_predict

    accuracy = correct\_predict / total if total != 0 else 0

    return accuracy

# Build and evaluate tree

tree = id3(train\_data\_m, 'Play Tennis')

test\_data\_m = pd.read\_csv("/content/PlayTennis.csv")

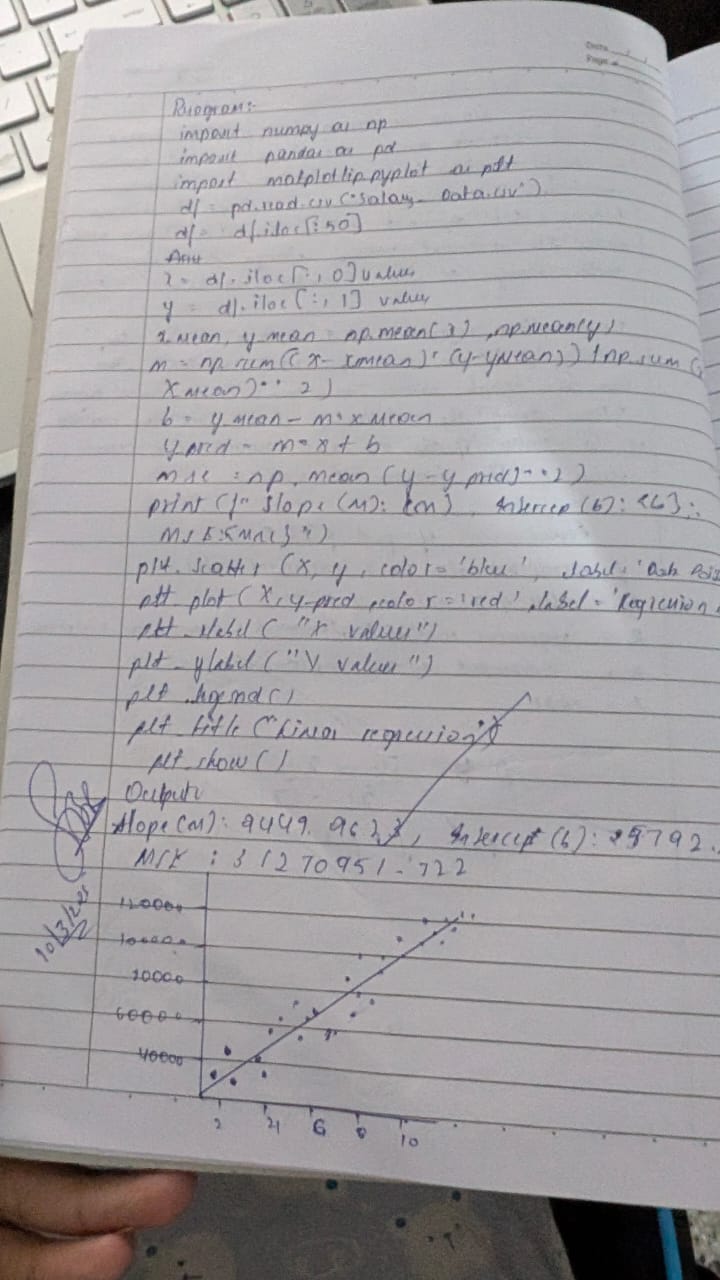
accuracy = evaluate(tree, test\_data\_m, 'Play Tennis')

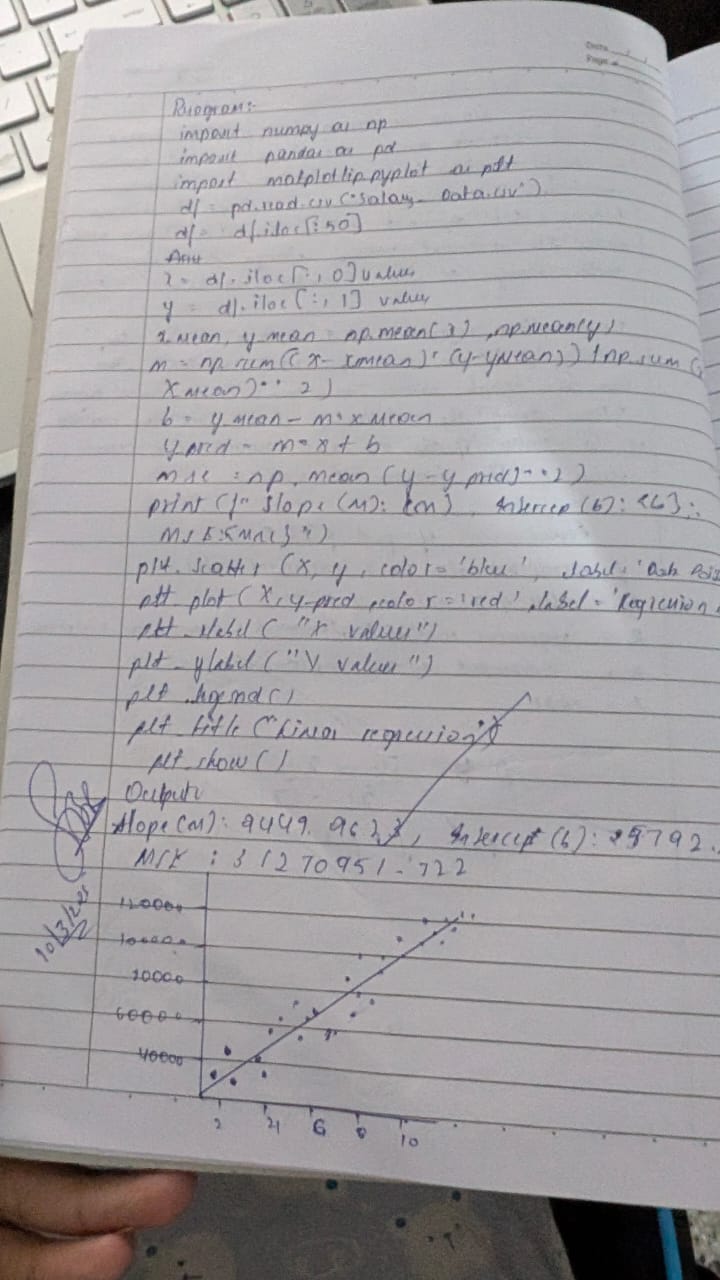
print("Accuracy:", accuracy)

print(tree)

**Program 4**

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset **Screenshots**





**Code**:

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.datasets import fetch\_california\_housing

*# Load the California Housing dataset*

california\_housing = fetch\_california\_housing()

*# Assign the data (features) and target (house prices)*

X = pd.DataFrame(california\_housing.data, columns=california\_housing.feature\_names)

y = pd.Series(california\_housing.target)

*# Select features for Linear Regression*

X = X[['MedInc', 'AveRooms']]

*# Split the data into training and testing sets*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

*# Create and train the Linear Regression model*

model = LinearRegression()

model.fit(X\_train, y\_train)

*# Make predictions*

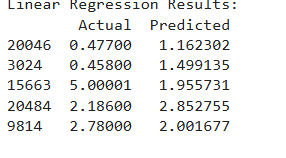
y\_pred = model.predict(X\_test)

*# Print the actual vs predicted values*

results = pd.DataFrame({'Actual': y\_test, 'Predicted': y\_pred})

print("Linear Regression Results:")

print(results.head())



import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.datasets import fetch\_california\_housing

california\_housing = fetch\_california\_housing()

X = pd.DataFrame(california\_housing.data, columns=california\_housing.feature\_names)

y = pd.Series(california\_housing.target)

X = X[['MedInc', 'AveRooms']]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

model = LinearRegression()

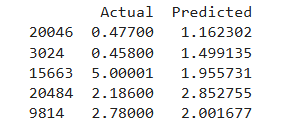
model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

*# Print the actual vs predicted values*

results = pd.DataFrame({'Actual': y\_test, 'Predicted': y\_pred})

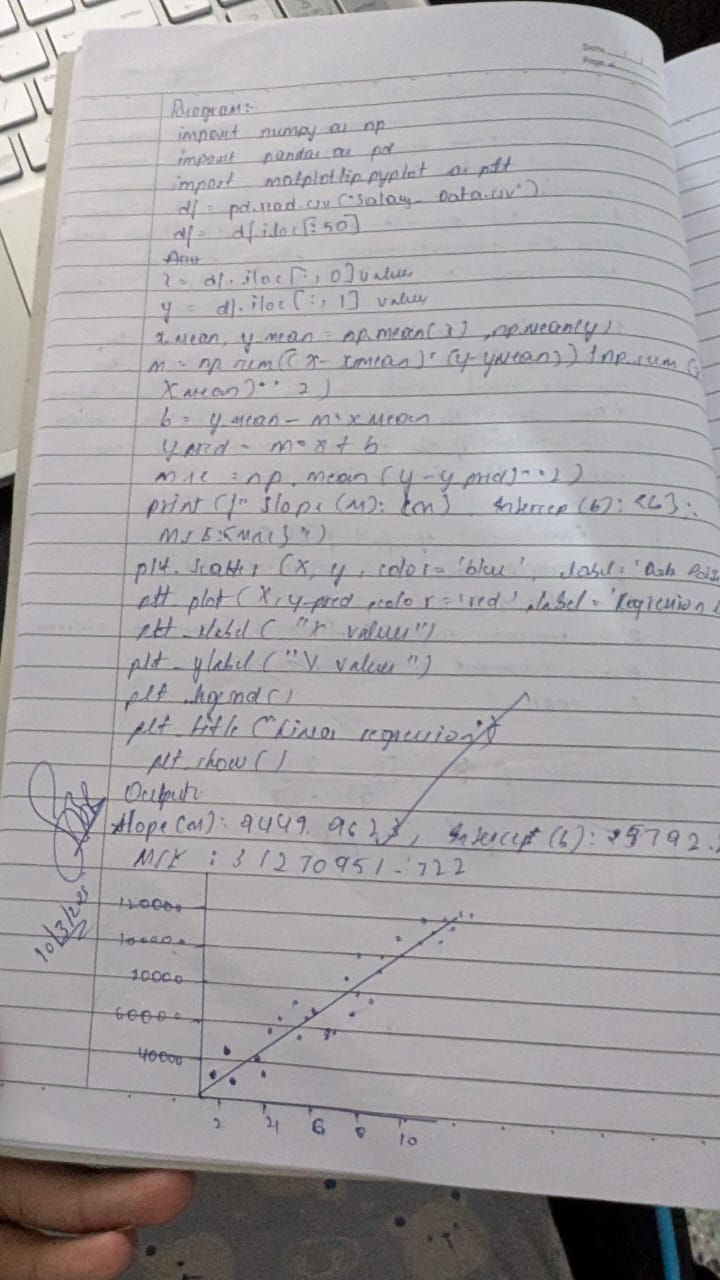
print(results.head())



**Program 6**

Build Logistic Regression Model for a given dataset

**Screenshots**



**Code**

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.datasets import load\_iris

*# Load the Iris dataset*

iris = load\_iris()

*# Assign the data (features) and target (species)*

X = pd.DataFrame(iris.data, columns=iris.feature\_names)

y = pd.Series(iris.target)

*# For simplicity, we will classify only two classes (0 and 1)*

X = X[y.isin([0, 1])] *# Select only classes 0 and 1*

y = y[y.isin([0, 1])]

*# Split the data into training and testing sets*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

*# Create and train the Logistic Regression model*

model = LogisticRegression()

model.fit(X\_train, y\_train)

*# Make predictions*

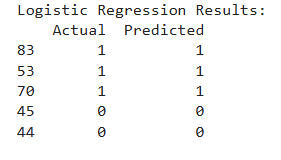
y\_pred = model.predict(X\_test)

*# Print the actual vs predicted values*

results = pd.DataFrame({'Actual': y\_test, 'Predicted': y\_pred})

print("Logistic Regression Results:")

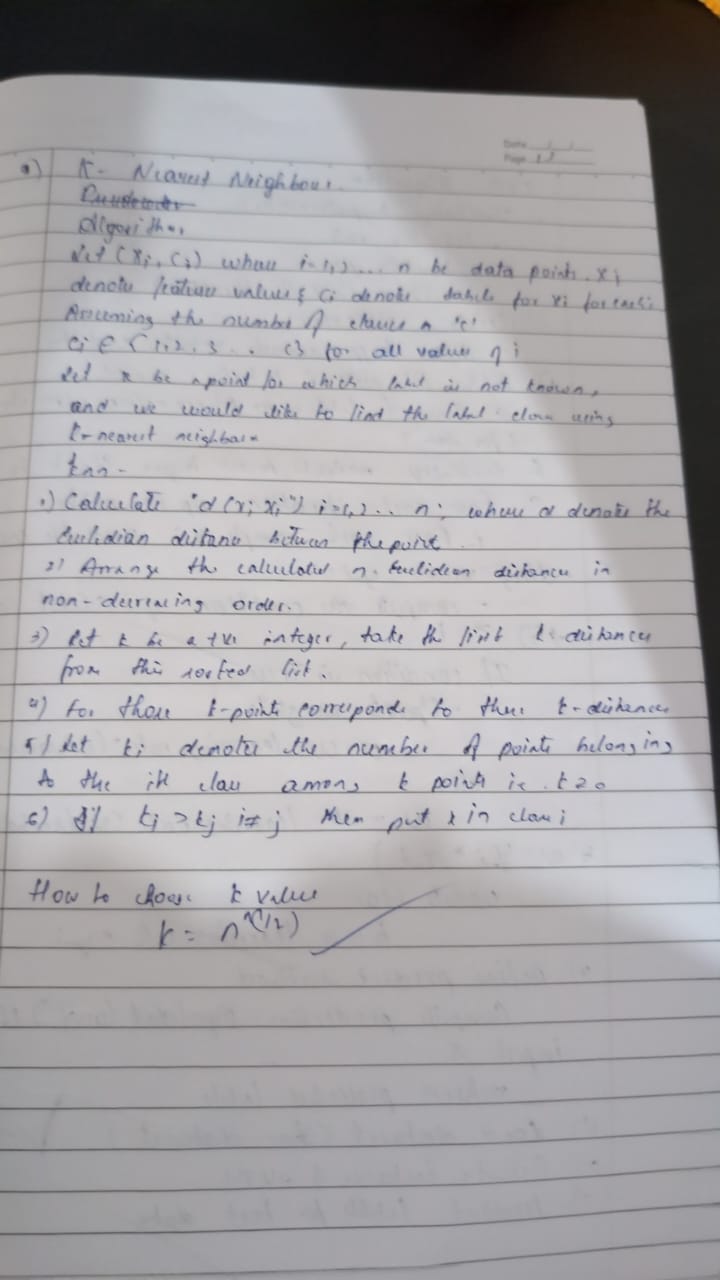
print(results.head())



**Program 6**

Build KNN Classification model for a given dataset

**Screenshots**



**Code**

import math

from collections import Counter

import pandas as pd

from sklearn.model\_selection import train\_test\_split

def euclidean\_distance(point1, point2):

return math.sqrt(sum((x - y) \*\* 2 for x, y in zip(point1, point2)))

class KNN:

def \_\_init\_\_(self, k=3):

self.k = k

self.X\_train = []

self.y\_train = []

def fit(self, X, y):

self.X\_train = X

self.y\_train = 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 = sorted(range(len(distances)), key=lambda i: distances[i])[: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 sum(pred == true for pred, true in zip(predictions, y)) / len(y)

if \_\_name\_\_ == "\_\_main\_\_":

from sklearn.datasets import load\_iris

iris = load\_iris()

df = pd.DataFrame(data=iris.data, columns=iris.feature\_names)

df['target'] = iris.target

X = df.iloc[:, :-1].values.tolist()

y = df['target'].tolist()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

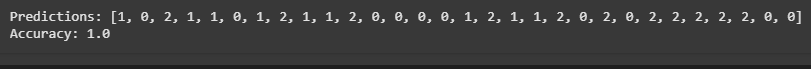
model = KNN(k=3)

model.fit(X\_train, y\_train)

predictions = model.predict(X\_test)

print("Predictions:", predictions)

print("Accuracy:", model.score(X\_test, y\_test))



**Program 7**

Build Support vector machine model for a given dataset

**Screenshots**

**Code**

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

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):

        n\_samples, n\_features = X.shape

        y\_ = np.where(y <= 0, -1, 1)

        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)

if \_\_name\_\_ == "\_\_main\_\_":

    from sklearn.datasets import load\_iris

    iris = load\_iris()

    df = pd.DataFrame(data=iris.data, columns=iris.feature\_names)

    df['target'] = iris.target

    df = df[df['target'] != 2]

    X = df.iloc[:, :2].values

    y = df['target'].values

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

    model = SVM()

    model.fit(X\_train, y\_train)

    predictions = model.predict(X\_test)

    acc = np.mean(predictions == np.where(y\_test == 0, -1, 1))

    print("Predictions:", predictions)

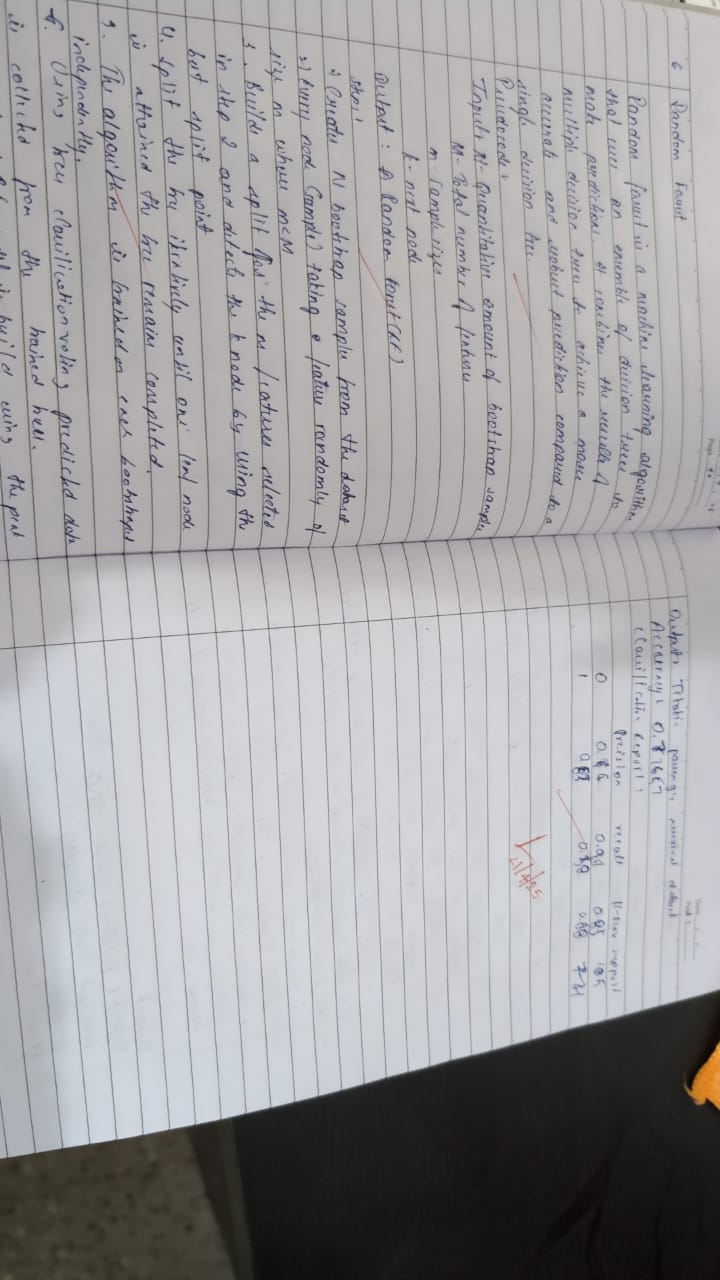
    print("Accuracy:", acc)



**Program 8**

Implement Random forest ensemble method on a given dataset.

**Screenshots**



**Code**

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score, classification\_report

from sklearn.preprocessing import LabelEncoder

from collections import Counter

import random

# -------------------------

# Random Forest From Scratch

# -------------------------

class CustomRandomForest:

    def \_\_init\_\_(self, n\_estimators=10, max\_features='sqrt', max\_depth=None):

        self.n\_estimators = n\_estimators

        self.max\_features = max\_features

        self.max\_depth = max\_depth

        self.trees = []

    def \_bootstrap\_sample(self, X, y):

        indices = np.random.choice(len(X), size=len(X), replace=True)

        return X.iloc[indices], y.iloc[indices]

    def \_get\_max\_features(self, n\_features):

        if self.max\_features == 'sqrt':

            return int(np.sqrt(n\_features))

        elif self.max\_features == 'log2':

            return int(np.log2(n\_features))

        elif isinstance(self.max\_features, int):

            return self.max\_features

        else:

            return n\_features

    def fit(self, X, y):

        self.trees = []

        for \_ in range(self.n\_estimators):

            X\_sample, y\_sample = self.\_bootstrap\_sample(X, y)

            max\_feats = self.\_get\_max\_features(X.shape[1])

            features = random.sample(list(X.columns), max\_feats)

            tree = DecisionTreeClassifier(max\_depth=self.max\_depth)

            tree.fit(X\_sample[features], y\_sample)

            self.trees.append((tree, features))

    def predict(self, X):

        tree\_preds = []

        for tree, features in self.trees:

            preds = tree.predict(X[features])

            tree\_preds.append(preds)

        tree\_preds = np.array(tree\_preds).T

        final\_preds = [Counter(row).most\_common(1)[0][0] for row in tree\_preds]

        return np.array(final\_preds)

# -------------------------

# Load and Preprocess Titanic Dataset

# -------------------------

df = pd.read\_csv('train.csv')

# Feature engineering

df['Title'] = df['Name'].str.extract(' ([A-Za-z]+)\.', expand=False)

df['Title'] = df['Title'].replace(['Lady', 'Countess','Capt','Col','Don',

                                   'Dr','Major','Rev','Sir','Jonkheer','Dona'], 'Rare')

df['Title'] = df['Title'].replace('Mlle', 'Miss')

df['Title'] = df['Title'].replace('Ms', 'Miss')

df['Title'] = df['Title'].replace('Mme', 'Mrs')

# Fill missing values

df['Age'].fillna(df['Age'].median(), inplace=True)

df['Embarked'].fillna(df['Embarked'].mode()[0], inplace=True)

df['Fare'].fillna(df['Fare'].median(), inplace=True)

# Drop unused columns

df.drop(['PassengerId', 'Name', 'Ticket', 'Cabin'], axis=1, inplace=True)

# Encode categoricals

for col in ['Sex', 'Embarked', 'Title']:

    le = LabelEncoder()

    df[col] = le.fit\_transform(df[col])

# Features and target

X = df.drop('Survived', axis=1)

y = df['Survived']

# Train/test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# -------------------------

# Train and Evaluate Custom Random Forest

# -------------------------

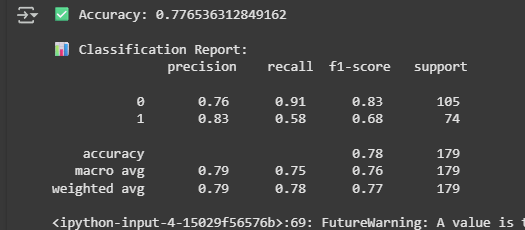
model = CustomRandomForest(n\_estimators=20, max\_features='sqrt', max\_depth=7)

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

print("✅ Accuracy:", accuracy\_score(y\_test, y\_pred))

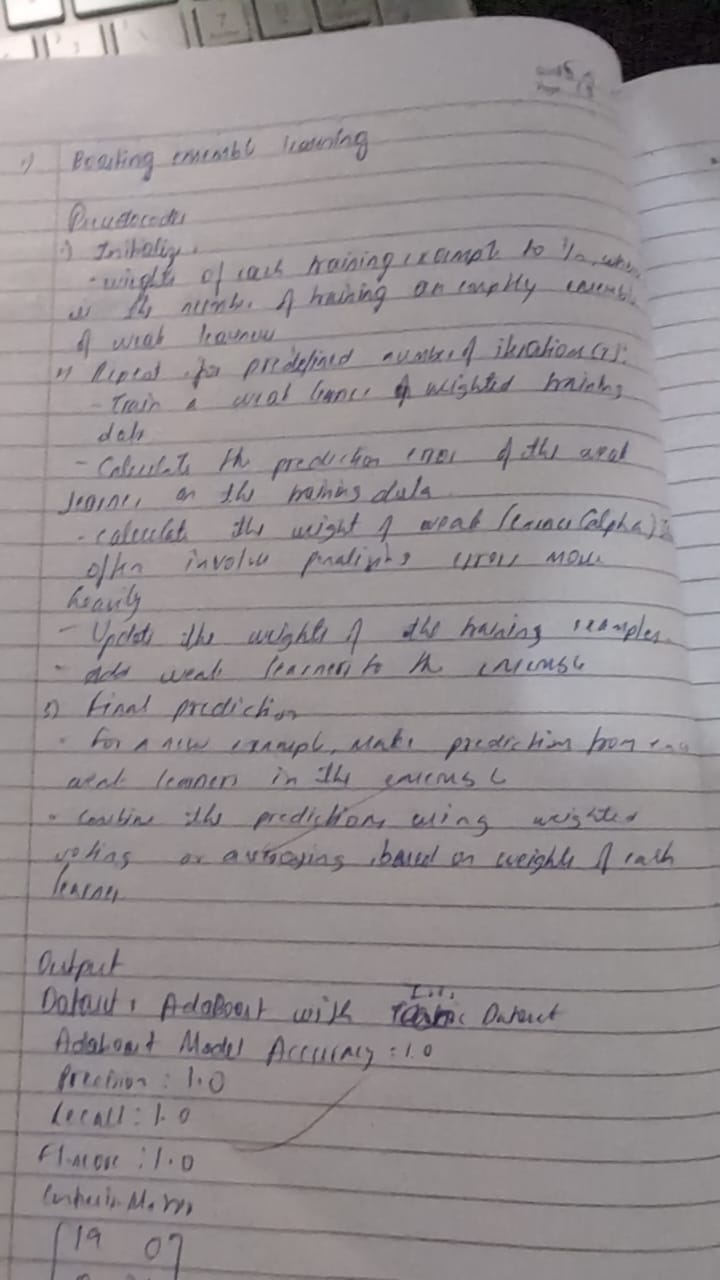
print("\n📊 Classification Report:\n", classification\_report(y\_test, y\_pred))



**Program 9**

Implement Boosting ensemble method on a given dataset.

**Screenshots**



**Code**

import numpy as np

import pandas as pd

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, confusion\_matrix, classification\_report

# Function to calculate weighted accuracy

def calculate\_weighted\_error(y\_true, y\_pred, weights):

    return np.sum(weights \* (y\_true != y\_pred)) / np.sum(weights)

# Function to update weights

def update\_weights(weights, alpha, y\_true, y\_pred):

    return weights \* np.exp(alpha \* (y\_true != y\_pred).astype(float))

# AdaBoost implementation

def adaboost(X, y, n\_estimators):

    n\_samples, n\_features = X.shape

    weights = np.ones(n\_samples) / n\_samples

    estimators = []

    alphas = []

    for \_ in range(n\_estimators):

        # Train a weak learner (Decision Stump)

        best\_feature, best\_threshold, best\_polarity, best\_error = None, None, None, float('inf')

        for feature in range(n\_features):

            thresholds = np.unique(X[:, feature])

            for threshold in thresholds:

                for polarity in [1, -1]:

                    y\_pred = np.ones(n\_samples)

                    y\_pred[polarity \* X[:, feature] < polarity \* threshold] = -1

                    error = calculate\_weighted\_error(y, y\_pred, weights)

                    if error < best\_error:

                        best\_feature = feature

                        best\_threshold = threshold

                        best\_polarity = polarity

                        best\_error = error

        # Calculate alpha (model weight)

        alpha = 0.5 \* np.log((1 - best\_error) / (best\_error + 1e-10))

        # Update weights

        y\_pred = np.ones(n\_samples)

        y\_pred[best\_polarity \* X[:, best\_feature] < best\_polarity \* best\_threshold] = -1

        weights = update\_weights(weights, alpha, y, y\_pred)

        estimators.append((best\_feature, best\_threshold, best\_polarity))

        alphas.append(alpha)

    return estimators, alphas

# Prediction function

def predict(X, estimators, alphas):

    n\_samples = X.shape[0]

    final\_prediction = np.zeros(n\_samples)

    for (feature, threshold, polarity), alpha in zip(estimators, alphas):

        prediction = np.ones(n\_samples)

        prediction[polarity \* X[:, feature] < polarity \* threshold] = -1

        final\_prediction += alpha \* prediction

    return np.sign(final\_prediction)

# Load dataset

iris = pd.read\_csv('/Iris.csv')

# Prepare features and target

X = iris[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']].values

y = iris['Species']

# Convert target to numerical labels

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

y = le.fit\_transform(y)

y = np.where(y == 0, -1, 1)  # Convert labels to -1 and 1

# Train-test split

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Train AdaBoost

n\_estimators = 50

estimators, alphas = adaboost(X\_train, y\_train, n\_estimators)

# Make predictions

y\_pred = predict(X\_test, estimators, alphas)

# Evaluate

accuracy = accuracy\_score(y\_test, y\_pred)

precision = precision\_score(y\_test, y\_pred, average='weighted')

recall = recall\_score(y\_test, y\_pred, average='weighted')

f1 = f1\_score(y\_test, y\_pred, average='weighted')

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

print("AdaBoost Model Accuracy:", accuracy)

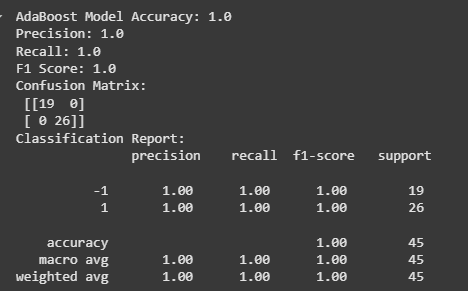
print("Precision:", precision)

print("Recall:", recall)

print("F1 Score:", f1)

print("Confusion Matrix:\n", conf\_matrix)

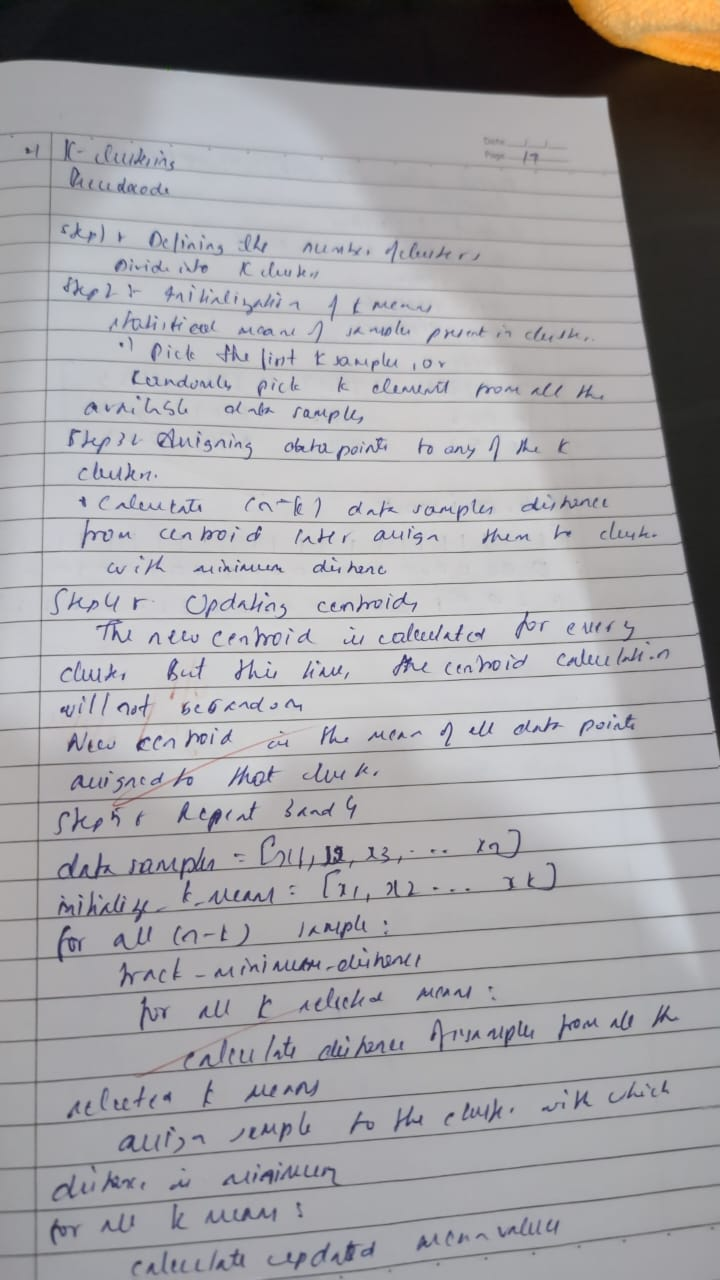
print("Classification Report:\n", classification\_report(y\_test, y\_pred))

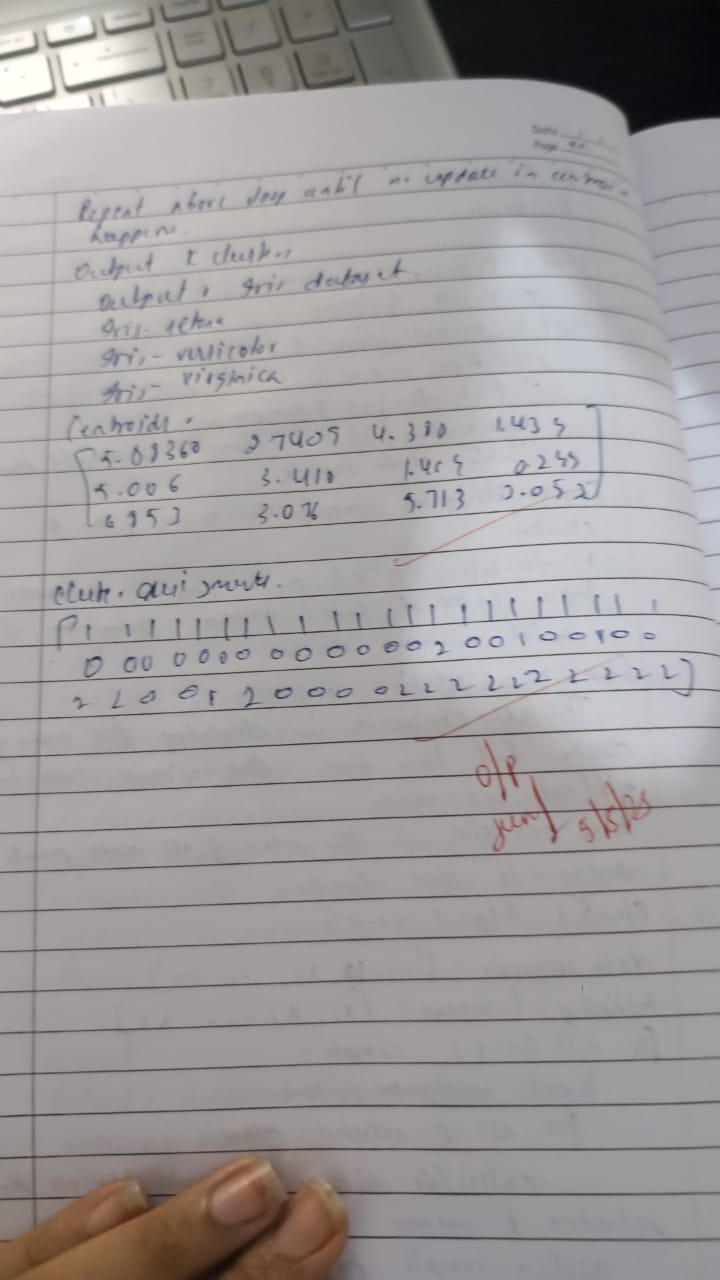


**Program 10**

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

**Screenshots**





**Code**

import numpy as np

import pandas as pd

from sklearn.metrics import silhouette\_score

# K-Means Clustering Implementation

def kmeans(X, n\_clusters, max\_iters=300, tol=1e-4):

    n\_samples, n\_features = X.shape

    # Randomly initialize cluster centers

    rng = np.random.default\_rng(seed=42)

    centroids = X[rng.choice(n\_samples, n\_clusters, replace=False)]

    for \_ in range(max\_iters):

        # Assign samples to nearest centroid

        distances = np.linalg.norm(X[:, np.newaxis] - centroids, axis=2)

        cluster\_assignments = np.argmin(distances, axis=1)

        # Calculate new centroids

        new\_centroids = np.array([X[cluster\_assignments == k].mean(axis=0) for k in range(n\_clusters)])

        # Check for convergence

        if np.linalg.norm(new\_centroids - centroids) < tol:

            break

        centroids = new\_centroids

    return centroids, cluster\_assignments

# Load dataset

iris = pd.read\_csv('/Iris.csv')

# Prepare features

X = iris[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']].values

# Number of clusters

n\_clusters = 3

# Apply K-Means

centroids, cluster\_assignments = kmeans(X, n\_clusters)

# Evaluate clustering using silhouette score

silhouette\_avg = silhouette\_score(X, cluster\_assignments)

# Print results

print("Centroids:")

print(centroids)

print("\nCluster Assignments:")

print(cluster\_assignments)

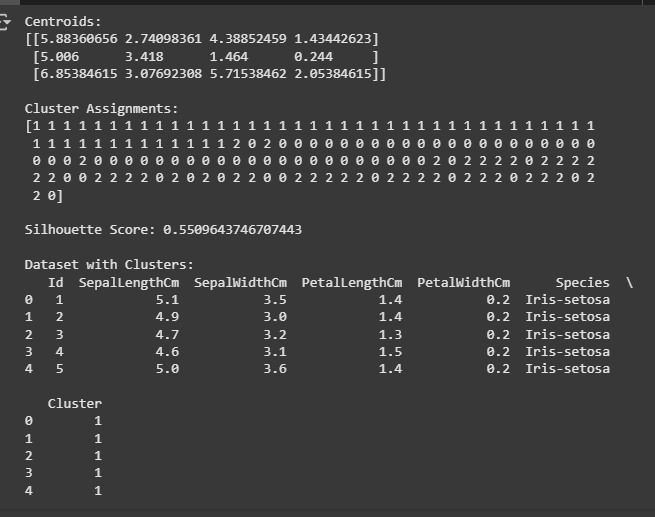
print("\nSilhouette Score:", silhouette\_avg)

# Add cluster assignments to the original dataset

iris['Cluster'] = cluster\_assignments

print("\nDataset with Clusters:")

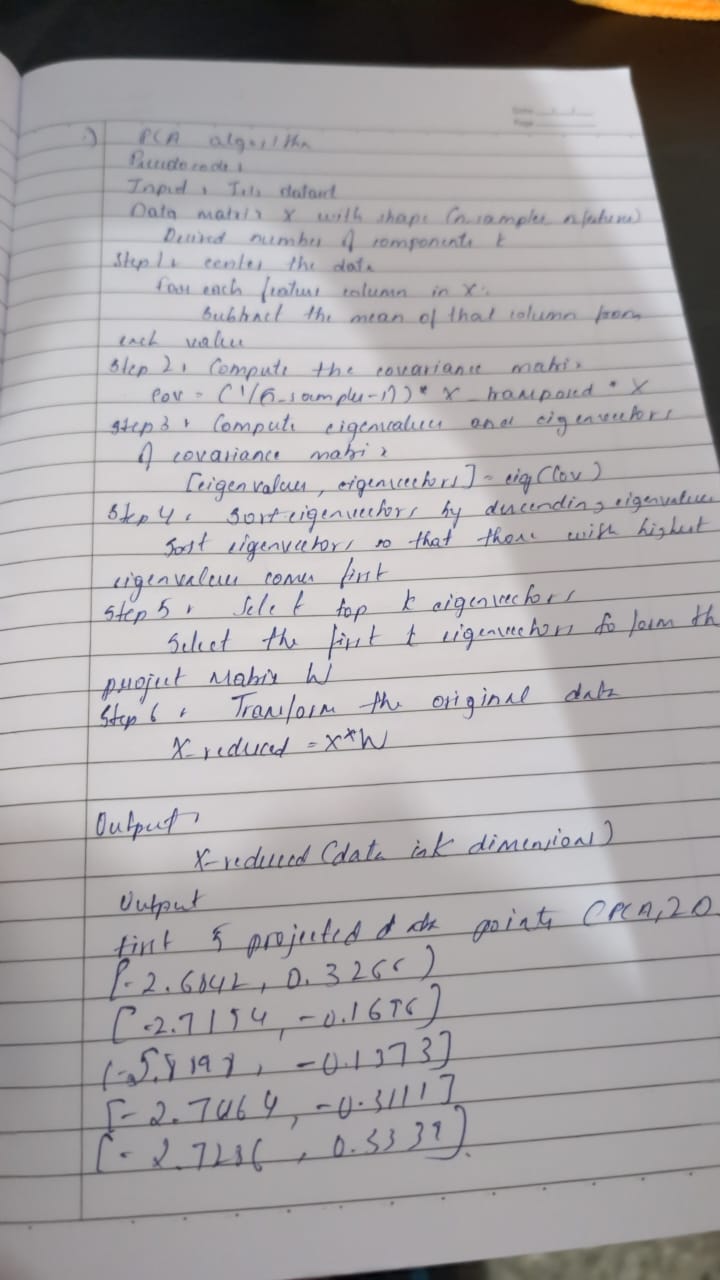
print(iris.head())



**Program 11**

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

**Screenshots**



**Code**

# PCA implementation from scratch using only built-in Python functions

# Step 1: Load the dataset

import csv

file\_path = "Iris.csv"  # Change path if needed

with open(file\_path, "r") as file:

    reader = csv.reader(file)

    data = list(reader)

header = data[0]

rows = data[1:]

# Step 2: Extract only numerical features (columns 1 to 4)

features = []

for row in rows:

    features.append([float(row[1]), float(row[2]), float(row[3]), float(row[4])])

# Step 3: Mean centering

def mean\_center(data):

    n = len(data)

    d = len(data[0])

    mean = [0.0] \* d

    for i in range(n):

        for j in range(d):

            mean[j] += data[i][j]

    for j in range(d):

        mean[j] /= n

    centered = []

    for i in range(n):

        centered.append([data[i][j] - mean[j] for j in range(d)])

    return centered, mean

centered\_data, mean\_vector = mean\_center(features)

# Step 4: Compute covariance matrix

def compute\_covariance\_matrix(data):

    n = len(data)

    d = len(data[0])

    cov\_matrix = [[0.0 for \_ in range(d)] for \_ in range(d)]

    for i in range(d):

        for j in range(d):

            for k in range(n):

                cov\_matrix[i][j] += data[k][i] \* data[k][j]

            cov\_matrix[i][j] /= (n - 1)

    return cov\_matrix

cov\_matrix = compute\_covariance\_matrix(centered\_data)

# Step 5: Eigenvalue and eigenvector using power iteration

def dot(v1, v2):

    return sum(x \* y for x, y in zip(v1, v2))

def mat\_vec\_mult(mat, vec):

    return [sum(mat[i][j] \* vec[j] for j in range(len(vec))) for i in range(len(mat))]

def norm(vec):

    return sum(x \* x for x in vec) \*\* 0.5

def normalize(vec):

    n = norm(vec)

    return [x / n for x in vec]

def power\_iteration(mat, num\_iter=1000):

    b\_k = [1.0 for \_ in range(len(mat))]

    for \_ in range(num\_iter):

        b\_k1 = mat\_vec\_mult(mat, b\_k)

        b\_k = normalize(b\_k1)

    eigenvalue = dot(b\_k, mat\_vec\_mult(mat, b\_k)) / dot(b\_k, b\_k)

    return eigenvalue, b\_k

def deflate\_matrix(mat, eigenvalue, eigenvector):

    d = len(mat)

    for i in range(d):

        for j in range(d):

            mat[i][j] -= eigenvalue \* eigenvector[i] \* eigenvector[j]

    return mat

# Step 6: Get top 2 eigenvectors (principal components)

eigvals = []

eigvecs = []

cov\_copy = [row[:] for row in cov\_matrix]

for \_ in range(2):

    val, vec = power\_iteration(cov\_copy)

    eigvals.append(val)

    eigvecs.append(vec)

    cov\_copy = deflate\_matrix(cov\_copy, val, vec)

# Step 7: Project data

def project\_data(data, components):

    projected = []

    for row in data:

        proj = [dot(row, comp) for comp in components]

        projected.append(proj)

    return projected

projected\_data = project\_data(centered\_data, eigvecs)

# Step 8: Print first 5 projected 2D data points

print("First 5 projected data points (PCA 2D):")

for point in projected\_data[:5]:

    print([round(x, 4) for x in point])

