**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

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

****

**LAB REPORT**

**on**

**Machine Learning (23CS6PCMAL)**

***Submitted by***

**Anurag Singh (1BM22CS048)**

***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 **Anurag Singh (1BM22CS048)** who is bonafide student of **B.M.S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum. The Lab report has been approved as it satisfies the academic requirements in respect of an Machine Learning (23CS6PCMAL) work prescribed for the said degree.

|  |  |
| --- | --- |
| Lab faculty Incharge Name: Prof. Saritha A. N  Assistant Professor  Department of CSE, BMSCE | Dr. Kavitha Sooda  Professor & HOD  Department of CSE, BMSCE |

**Index**

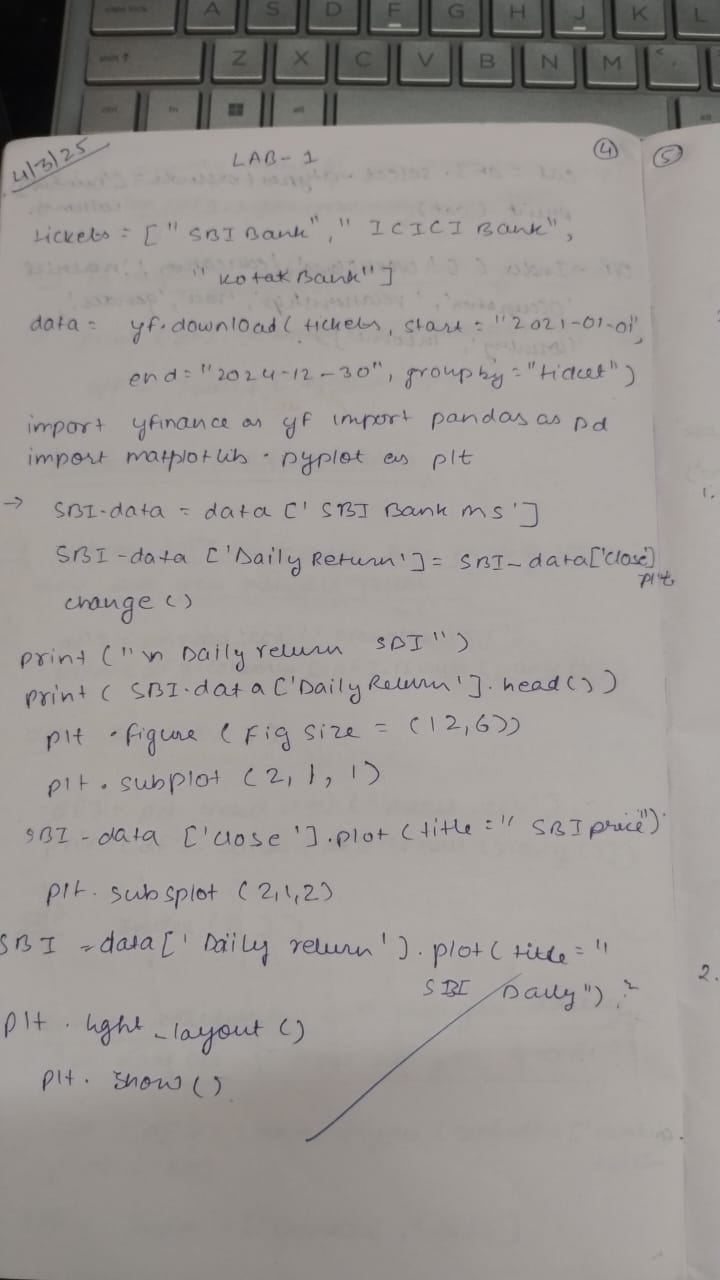
|  |  |  |  |
| --- | --- | --- | --- |
| **Sl.**  **No.** | **Date** | **Experiment Title** | **Page No.** |
| 1 | 4-3-2025 | Write a python program to  import and export data using Pandas library functions | 4 |
| 2 | 11-3-2025 | Demonstrate various data pre-processing techniques for a given dataset | 10 |
| 3 | 18-3-2025 | Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample. | 24 |
| 4 | 1-4-2025 | Implement Linear and Multi-Linear Regression algorithm using appropriate dataset | 35 |
| 5 | 8-4-2025 | Build Logistic Regression Model for a given dataset | 47 |
| 6 | 15-4-2025 | Build KNN Classification model for a given dataset. | 65 |
| 7 | 15-4-2025 | Build Support vector machine model for a given dataset | 76 |
| 8 | 22-4-2025 | Implement Random forest ensemble method on a given dataset. | 88 |
| 9 | 22-4-2025 | Implement Boosting ensemble method on a given dataset. | 94 |
| 10 | 29-4-2025 | Build k-Means algorithm to cluster a set of data stored in a .CSV file. | 100 |
| 11 | 29-4-2025 | Implement Dimensionality reduction using Principal Component Analysis (PCA) method. | 109 |

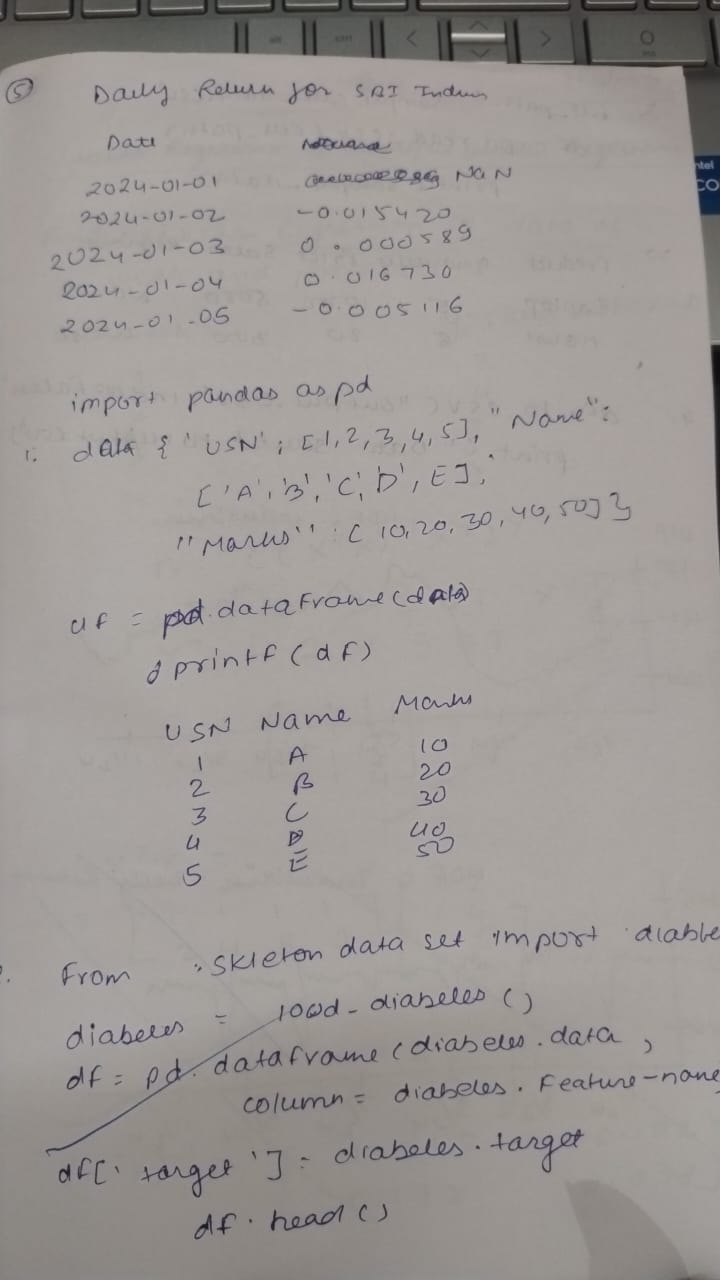
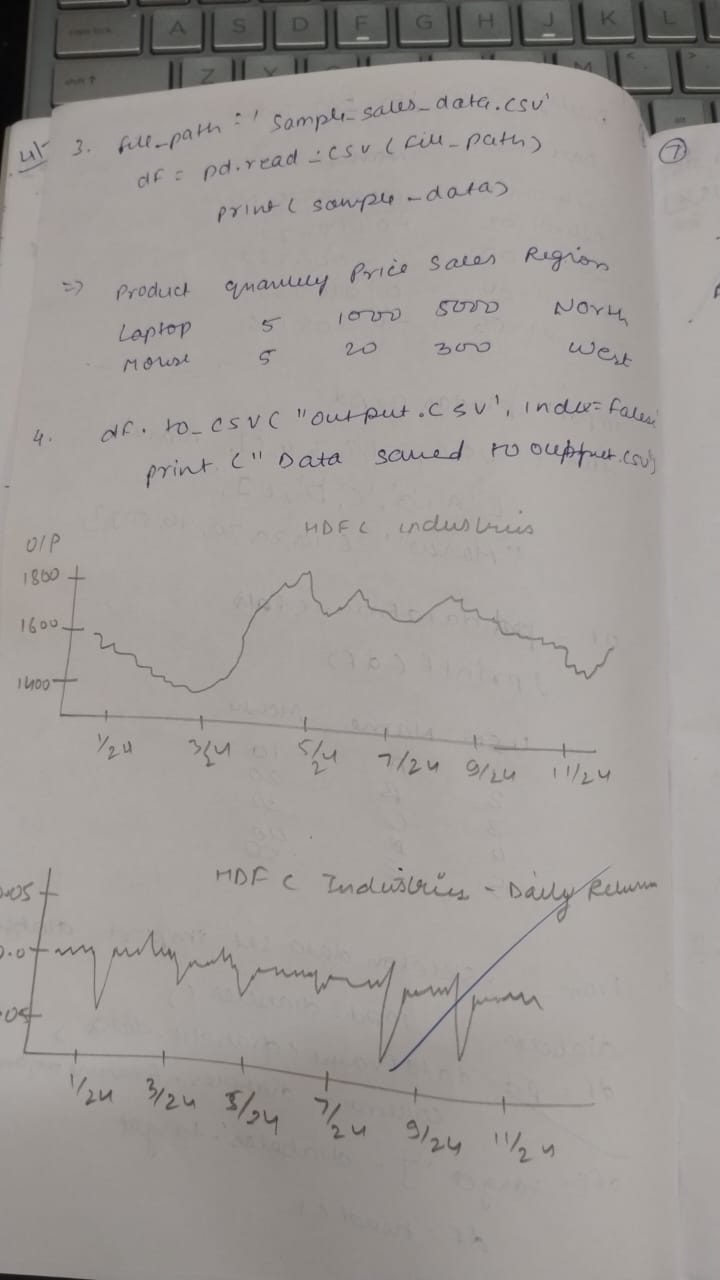
Github Link:

<https://github.com/anuragsingh472002/6thSem-ML-Lab>

**Program 1**

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





Code:

*#1*

import pandas as pd

In [32]:

*#2*

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

In [33]:

*#3*

df.columns

Out[33]:

Index(['longitude', 'latitude', 'housing\_median\_age', 'total\_rooms',

'total\_bedrooms', 'population', 'households', 'median\_income',

'median\_house\_value', 'ocean\_proximity'],

dtype='object')

In [34]:

*#4*

df.describe()

Out[34]:

|  | **longitude** | **latitude** | **housing\_median\_age** | **total\_rooms** | **total\_bedrooms** | **population** | **households** | **median\_income** | **median\_house\_value** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **count** | 20640.000000 | 20640.000000 | 20640.000000 | 20640.000000 | 20433.000000 | 20640.000000 | 20640.000000 | 20640.000000 | 20640.000000 |
| **mean** | -119.569704 | 35.631861 | 28.639486 | 2635.763081 | 537.870553 | 1425.476744 | 499.539680 | 3.870671 | 206855.816909 |
| **std** | 2.003532 | 2.135952 | 12.585558 | 2181.615252 | 421.385070 | 1132.462122 | 382.329753 | 1.899822 | 115395.615874 |
| **min** | -124.350000 | 32.540000 | 1.000000 | 2.000000 | 1.000000 | 3.000000 | 1.000000 | 0.499900 | 14999.000000 |
| **25%** | -121.800000 | 33.930000 | 18.000000 | 1447.750000 | 296.000000 | 787.000000 | 280.000000 | 2.563400 | 119600.000000 |
| **50%** | -118.490000 | 34.260000 | 29.000000 | 2127.000000 | 435.000000 | 1166.000000 | 409.000000 | 3.534800 | 179700.000000 |
| **75%** | -118.010000 | 37.710000 | 37.000000 | 3148.000000 | 647.000000 | 1725.000000 | 605.000000 | 4.743250 | 264725.000000 |
| **max** | -114.310000 | 41.950000 | 52.000000 | 39320.000000 | 6445.000000 | 35682.000000 | 6082.000000 | 15.000100 | 500001.000000 |

In [36]:

*#5*

l=df.ocean\_proximity.unique()

print(l)

print(len(l))

['NEAR BAY' '<1H OCEAN' 'INLAND' 'NEAR OCEAN' 'ISLAND']

5

In [39]:

*#6*

df.isna().sum()

print(df.columns[df.isnull().any()])

Index(['total\_bedrooms'], dtype='object')

In [42]:

df=pd.read\_csv('/content/Dataset of Diabetes .csv')

print("Sample Data")

print(df.head())

print(df.isna().sum())

print(df.columns[df.isnull().any()])

categorical\_cols = df.select\_dtypes(exclude=['number']).columns

categorical\_cols

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

ID 0

No\_Pation 0

Gender 0

AGE 0

Urea 0

Cr 0

HbA1c 0

Chol 0

TG 0

HDL 0

LDL 0

VLDL 0

BMI 0

CLASS 0

dtype: int64

Index([], dtype='object')

Out[42]:

Index(['Gender', 'CLASS'], dtype='object')

In [44]:

df=pd.read\_csv('/content/adult.csv')

print("Sample Data")

print(df.head())

print(df.isna().sum())

print(df.columns[df.isnull().any()])

categorical\_col = df.select\_dtypes(exclude=['number']).columns

categorical\_col

Sample Data

age workclass fnlwgt education educational-num marital-status \

0 25 Private 226802 11th 7 Never-married

1 38 Private 89814 HS-grad 9 Married-civ-spouse

2 28 Local-gov 336951 Assoc-acdm 12 Married-civ-spouse

3 44 Private 160323 Some-college 10 Married-civ-spouse

4 18 ? 103497 Some-college 10 Never-married

occupation relationship race gender capital-gain capital-loss \

0 Machine-op-inspct Own-child Black Male 0 0

1 Farming-fishing Husband White Male 0 0

2 Protective-serv Husband White Male 0 0

3 Machine-op-inspct Husband Black Male 7688 0

4 ? Own-child White Female 0 0

hours-per-week native-country income

0 40 United-States <=50K

1 50 United-States <=50K

2 40 United-States >50K

3 40 United-States >50K

4 30 United-States <=50K

age 0

workclass 0

fnlwgt 0

education 0

educational-num 0

marital-status 0

occupation 0

relationship 0

race 0

gender 0

capital-gain 0

capital-loss 0

hours-per-week 0

native-country 0

income 0

dtype: int64

Index([], dtype='object')

Out[44]:

Index(['workclass', 'education', 'marital-status', 'occupation',

'relationship', 'race', 'gender', 'native-country', 'income'],

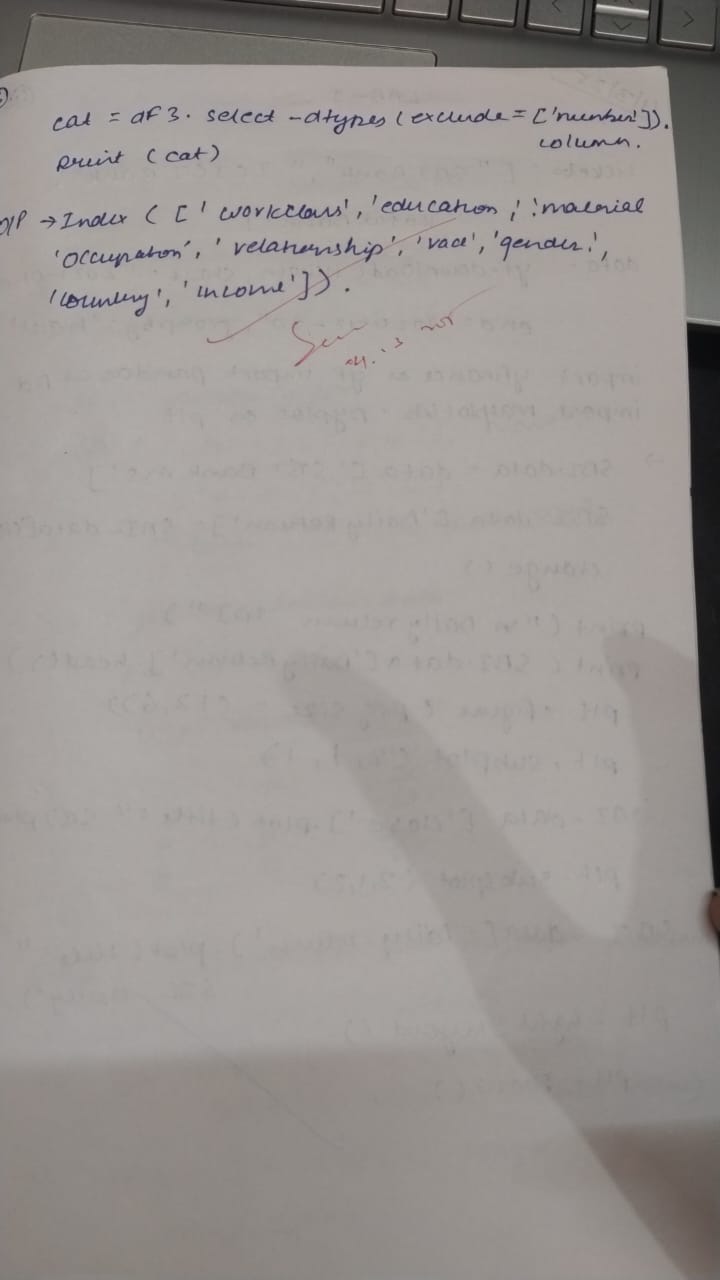
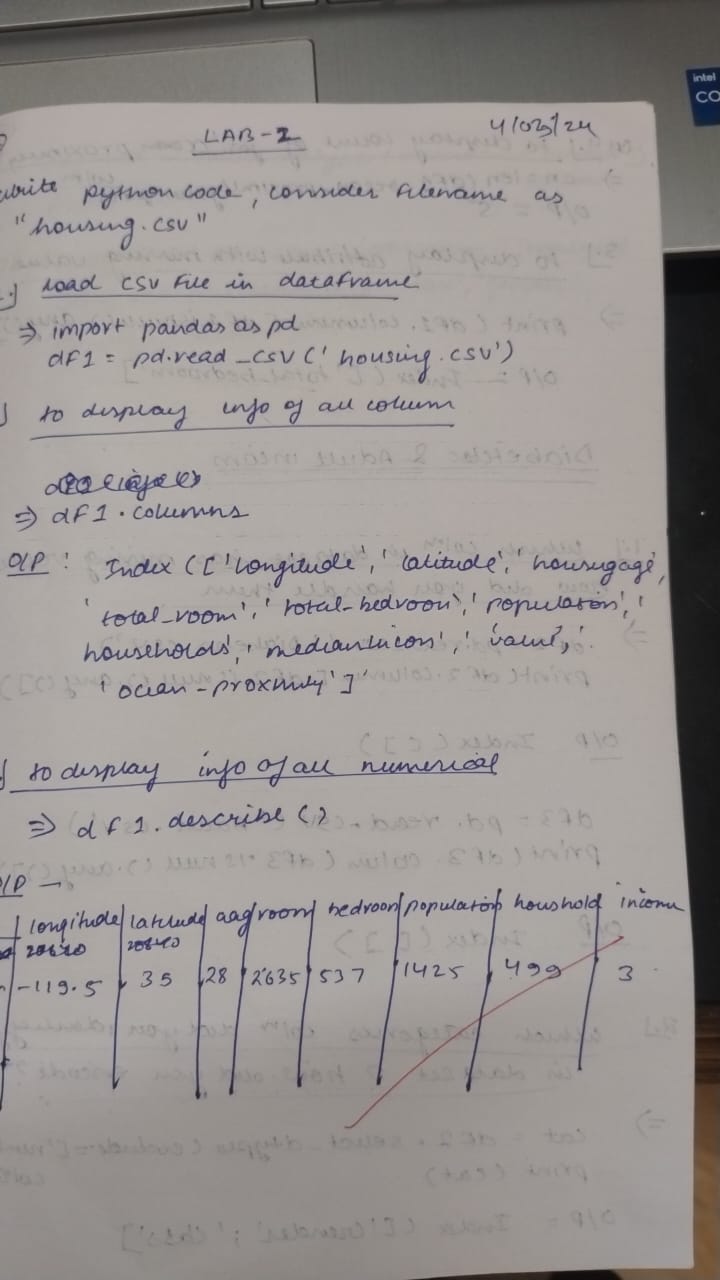
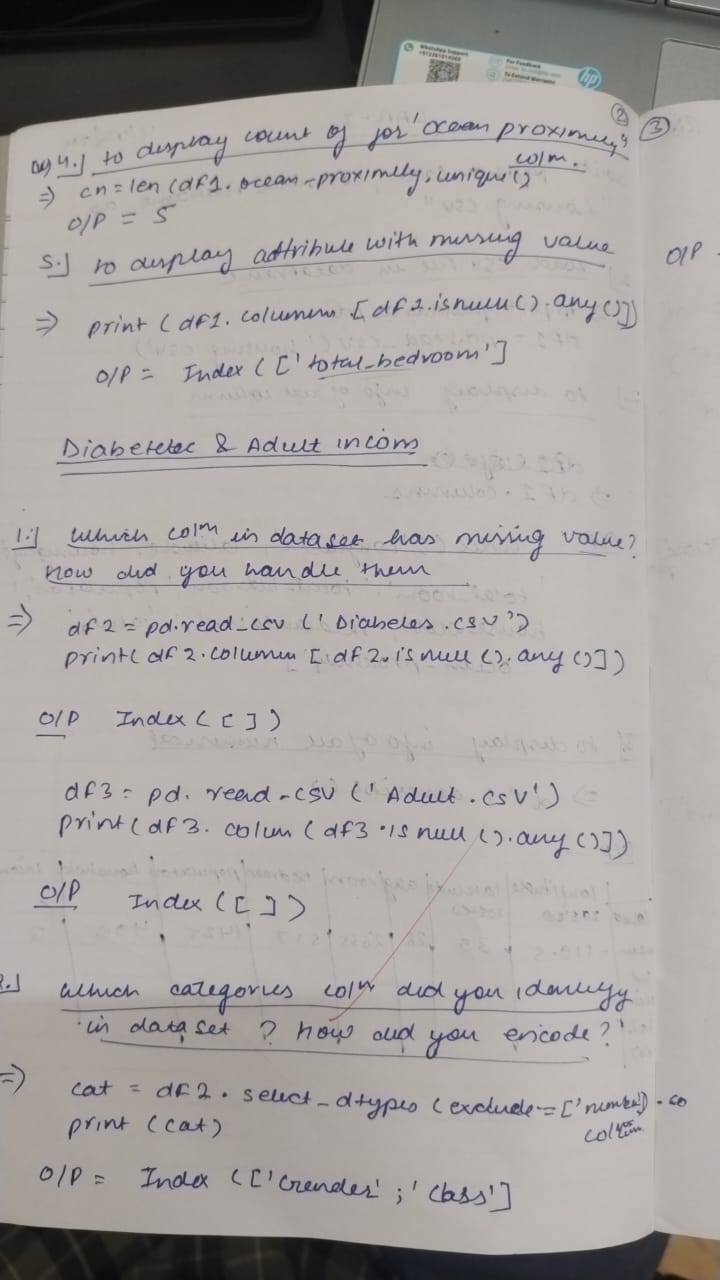
dtype='object')

In [ ]:

**Program 2**

Demonstrate various data pre-processing techniques for a given dataset

Algorithm:



Code:

import pandas as pd

data ={

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

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

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

}

data = pd.DataFrame(data)

print(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

df.head()

print(df)

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

.. ... ... ... ...

145 6.7 3.0 5.2 2.3

146 6.3 2.5 5.0 1.9

147 6.5 3.0 5.2 2.0

148 6.2 3.4 5.4 2.3

149 5.9 3.0 5.1 1.8

target

0 0

1 0

2 0

3 0

4 0

.. ...

145 2

146 2

147 2

148 2

149 2

[150 rows x 5 columns]

In [ ]:

file\_path='data.csv'

df=pd.read\_csv(file\_path)

print("Sample Data")

print(df.head())

Sample Data

ID Name Age City

0 1 Alice 25 New York

1 2 Bob 30 Los Angeles

2 3 Charlie 35 Chicago

3 4 David 40 Houston

4 5 Eva 28 Phoenix

In [20]:

df=pd.read\_csv('mobiles.csv',encoding='unicode\_escape')

print("Sample Data")

print(df.head())

Sample Data

Company Name Model Name Mobile Weight RAM Front Camera \

0 Apple iPhone 16 128GB 174g 6GB 12MP

1 Apple iPhone 16 256GB 174g 6GB 12MP

2 Apple iPhone 16 512GB 174g 6GB 12MP

3 Apple iPhone 16 Plus 128GB 203g 6GB 12MP

4 Apple iPhone 16 Plus 256GB 203g 6GB 12MP

Back Camera Processor Battery Capacity Screen Size \

0 48MP A17 Bionic 3,600mAh 6.1 inches

1 48MP A17 Bionic 3,600mAh 6.1 inches

2 48MP A17 Bionic 3,600mAh 6.1 inches

3 48MP A17 Bionic 4,200mAh 6.7 inches

4 48MP A17 Bionic 4,200mAh 6.7 inches

Launched Price (Pakistan) Launched Price (India) Launched Price (China) \

0 PKR 224,999 INR 79,999 CNY 5,799

1 PKR 234,999 INR 84,999 CNY 6,099

2 PKR 244,999 INR 89,999 CNY 6,499

3 PKR 249,999 INR 89,999 CNY 6,199

4 PKR 259,999 INR 94,999 CNY 6,499

Launched Price (USA) Launched Price (Dubai) Launched Year

0 USD 799 AED 2,799 2024

1 USD 849 AED 2,999 2024

2 USD 899 AED 3,199 2024

3 USD 899 AED 3,199 2024

4 USD 949 AED 3,399 2024

In [21]:

*#1*

df={

'USN':[1,2,3,4,5],

'Name':['A','B','C','D','E'],

'Marks':[10,20,30,40,50]

}

df = pd.DataFrame(df)

print(df)

USN Name Marks

0 1 A 10

1 2 B 20

2 3 C 30

3 4 D 40

4 5 E 50

In [24]:

*#2*

from sklearn.datasets import load\_diabetes

diabetes= load\_diabetes()

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

df['target']=diabetes.target

df.head()

print(df)

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

.. ... ... ... ... ... ... ...

437 0.041708 0.050680 0.019662 0.059744 -0.005697 -0.002566 -0.028674

438 -0.005515 0.050680 -0.015906 -0.067642 0.049341 0.079165 -0.028674

439 0.041708 0.050680 -0.015906 0.017293 -0.037344 -0.013840 -0.024993

440 -0.045472 -0.044642 0.039062 0.001215 0.016318 0.015283 -0.028674

441 -0.045472 -0.044642 -0.073030 -0.081413 0.083740 0.027809 0.173816

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

.. ... ... ... ...

437 -0.002592 0.031193 0.007207 178.0

438 0.034309 -0.018114 0.044485 104.0

439 -0.011080 -0.046883 0.015491 132.0

440 0.026560 0.044529 -0.025930 220.0

441 -0.039493 -0.004222 0.003064 57.0

[442 rows x 11 columns]

In [29]:

*#3*

file\_path='sample\_sales\_data.csv'

df=pd.read\_csv(file\_path)

print("Sample Data")

print(df.head())

Sample Data

Product Quantity Price Sales Region

0 Laptop 5 1000 5000 North

1 Mouse 15 20 300 West

2 Keyboard 10 50 500 East

3 Monitor 8 200 1600 South

4 Laptop 12 950 11400 North

In [30]:

*#4*

df=pd.read\_csv('/content/Dataset of Diabetes .csv')

print("Sample Data")

print(df.head())

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 [28]:

df.to\_csv('output.csv', index=False)

print("Data saved to output.csv")

Data saved to output.csv

In [34]:

sales\_df = pd.read\_csv('sample\_sales\_data.csv')

sales\_by\_region = sales\_df.groupby('Region')['Sales'].sum()

print("\nTotal sales by region:")

print(sales\_by\_region)

best\_selling\_products = sales\_df.groupby('Product')['Quantity'].sum().sort\_values(ascending=False)

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

print(best\_selling\_products)

*# Saving the sales by region data to a CSV file*

sales\_by\_region.to\_csv('sales\_by\_region.csv')

*# Saving the best-selling products data to a CSV file*

best\_selling\_products.to\_csv('best\_selling\_products.csv')

print("\nAnalysis results saved to CSV files.")

Total sales by region:

Region

East 770

North 16400

South 3070

West 650

Name: Sales, dtype: int64

Best-selling products by quantity:

Product

Mouse 29

Laptop 17

Keyboard 16

Monitor 15

Name: Quantity, dtype: int64

Analysis results saved to CSV files.

In [35]:

import yfinance as yf

import pandas as pd

import matplotlib.pyplot as plt

In [36]:

*# 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 1088.493134 1100.076669 1075.521371 1078.479858 11852723

2022-10-04 1091.360592 1100.554607 1087.878645 1098.369873 8948850

2022-10-06 1105.561293 1115.119370 1100.622856 1102.420654 13352162

2022-10-07 1099.029980 1112.343173 1099.029980 1107.086182 7714340

2022-10-10 1094.637648 1100.372591 1086.900123 1095.001831 6329527

Ticker TCS.NS \

Price Open High Low Close Volume

Date

2022-10-03 2799.043864 2823.062325 2779.417847 2789.651367 1763331

2022-10-04 2831.707536 2895.305232 2825.212304 2888.903320 2145875

2022-10-06 2907.454507 2919.603947 2890.118145 2898.996582 1790816

2022-10-07 2894.744206 2901.847047 2858.015696 2864.370605 1939879

2022-10-10 2813.062865 2922.407833 2808.390003 2914.510742 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 1370.286676 1381.181893 1364.412779 1374.881592 3994466

2022-10-10 1351.338576 1387.956005 1351.338576 1385.729614 5274677

In [38]:

*# 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())

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 1147.548565 1156.216094 1137.195377 1146.522438 1.316652e+07

std 65.890232 66.850140 65.752477 66.687266 6.754099e+06

min 1008.159038 1010.434788 992.228728 1001.900696 3.370033e+06

25% 1098.881943 1103.399363 1084.795104 1097.357178 8.717141e+06

50% 1147.435126 1155.036230 1138.787343 1147.253052 1.158959e+07

75% 1196.261444 1203.625226 1184.985050 1195.064575 1.530302e+07

max 1288.076823 1299.910771 1273.056848 1293.470337 5.708188e+07

In [39]:

*# Calculate daily returns*

reliance\_data['Daily Return'] = reliance\_data['Close'].pct\_change()

print("\nDaily returns for Reliance Industries:")

print(reliance\_data['Daily Return'].head())

*#*

Daily returns for Reliance Industries:

Date

2022-10-03 NaN

2022-10-04 0.018443

2022-10-06 0.003688

2022-10-07 0.004232

2022-10-10 -0.010915

Name: Daily Return, dtype: float64

<ipython-input-39-2bd40fbe735c>:2: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame.

Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy

reliance\_data['Daily Return'] = reliance\_data['Close'].pct\_change()

In [40]:

*# Plot the closing price and daily returns*

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

plt.subplot(2, 1, 1)

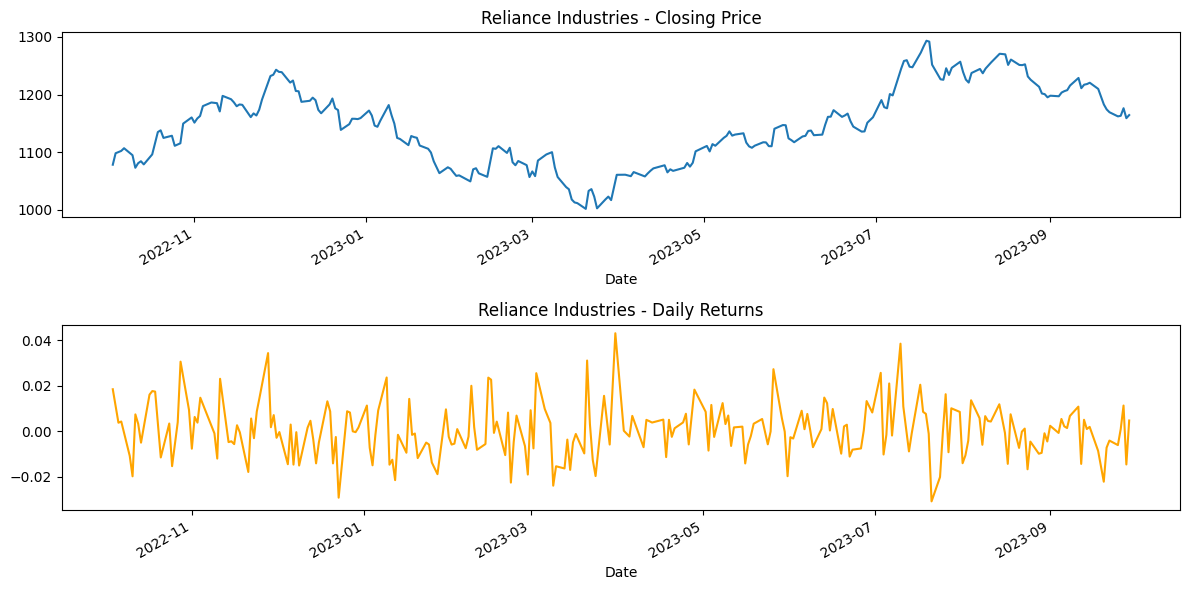
reliance\_data['Close'].plot(title="Reliance Industries - Closing Price")

plt.subplot(2, 1, 2)

reliance\_data['Daily Return'].plot(title="Reliance Industries - Daily Returns", color='orange')

plt.tight\_layout()

plt.show()



In [41]:

*# 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 [43]:

tickers = ["HDFCBANK.NS", "ICICIBANK.NS", "KOTAKBANK.NS"]

data = yf.download(tickers, start="2024-01-01", end="2024-12-30", group\_by='ticker')

[\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*100%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*] 3 of 3 completed

In [44]:

hdfc\_data = data['HDFCBANK.NS']

hdfc\_data['Daily Return'] = hdfc\_data['Close'].pct\_change()

print("\nDaily returns for HDFC Industries:")

print(hdfc\_data['Daily Return'].head())

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

plt.subplot(2, 1, 1)

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

plt.subplot(2, 1, 2)

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

plt.tight\_layout()

plt.show()

<ipython-input-44-f8f2798a87f5>:2: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame.

Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy

hdfc\_data['Daily Return'] = hdfc\_data['Close'].pct\_change()

Daily returns for HDFC Industries:

Date

2024-01-01 NaN

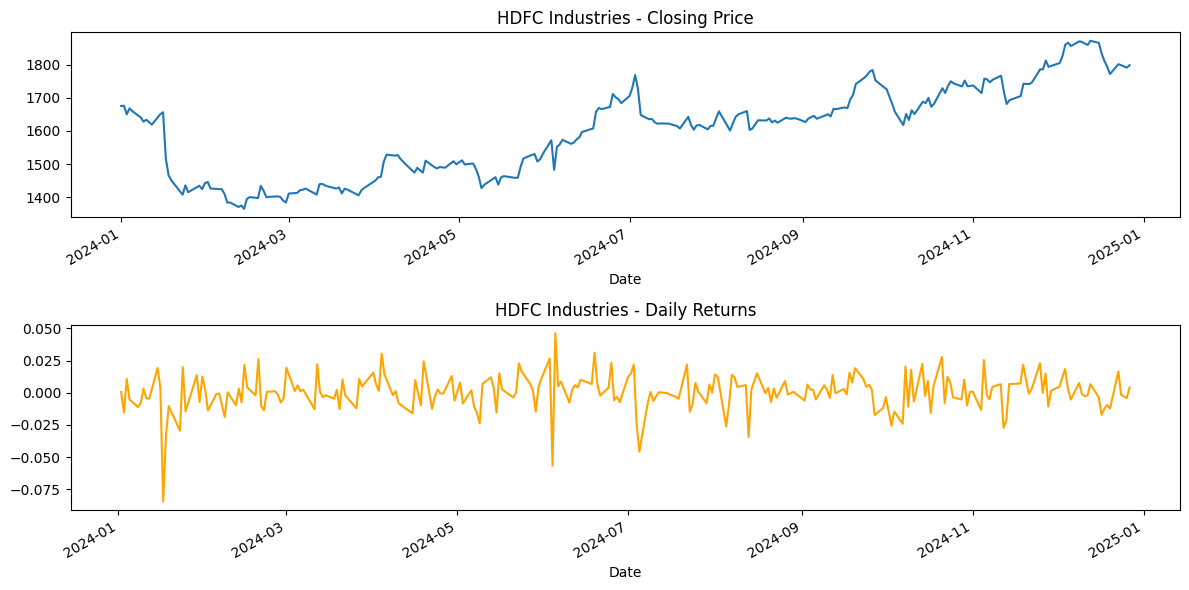
2024-01-02 0.000589

2024-01-03 -0.015420

2024-01-04 0.010730

2024-01-05 -0.005116

Name: Daily Return, dtype: float64



In [45]:

icici\_data = data['ICICIBANK.NS']

icici\_data['Daily Return'] = icici\_data['Close'].pct\_change()

print("\nDaily returns for ICICI Industries:")

print(icici\_data['Daily Return'].head())

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

plt.subplot(2, 1, 1)

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

plt.subplot(2, 1, 2)

icici\_data['Daily Return'].plot(title="ICICI Industries - Daily Returns", color='orange')

plt.tight\_layout()

plt.show()

<ipython-input-45-6f81270ab07d>:2: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame.

Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy

icici\_data['Daily Return'] = icici\_data['Close'].pct\_change()

Daily returns for ICICI Industries:

Date

2024-01-01 NaN

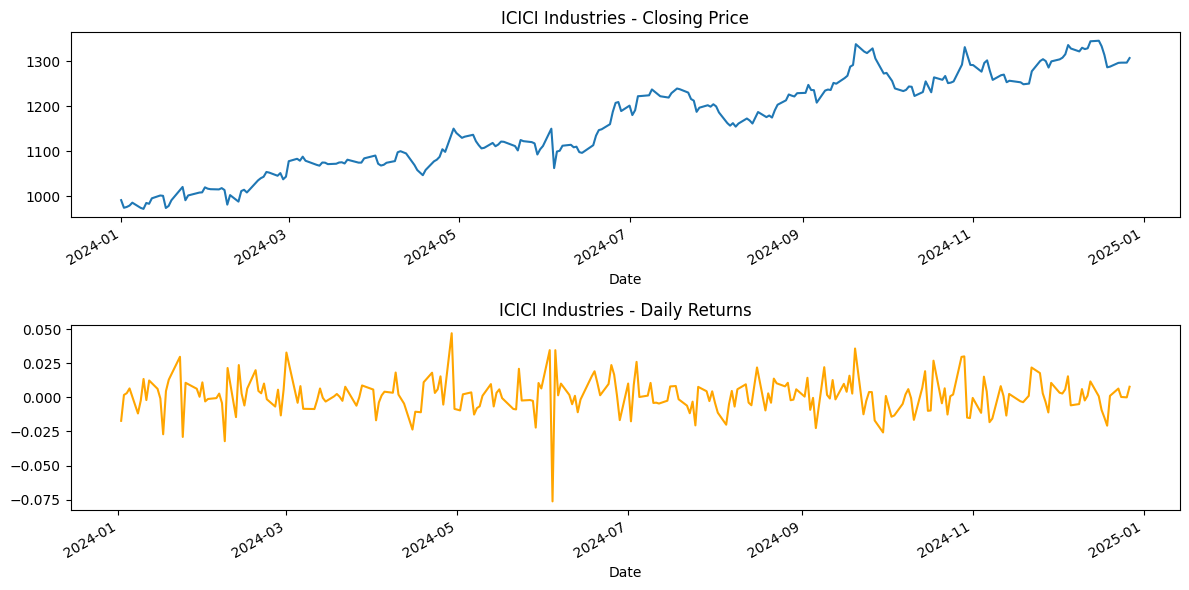
2024-01-02 -0.017160

2024-01-03 0.001833

2024-01-04 0.003150

2024-01-05 0.006635

Name: Daily Return, dtype: float64



In [46]:

kotak\_data = data['KOTAKBANK.NS']

kotak\_data['Daily Return'] = kotak\_data['Close'].pct\_change()

print("\nDaily returns for Kotak Industries:")

print(kotak\_data['Daily Return'].head())

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

plt.subplot(2, 1, 1)

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

plt.subplot(2, 1, 2)

kotak\_data['Daily Return'].plot(title="Kotak Industries - Daily Returns", color='orange')

plt.tight\_layout()

plt.show()

<ipython-input-46-098de9be386a>:2: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame.

Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy

kotak\_data['Daily Return'] = kotak\_data['Close'].pct\_change()

Daily returns for Kotak Industries:

Date

2024-01-01 NaN

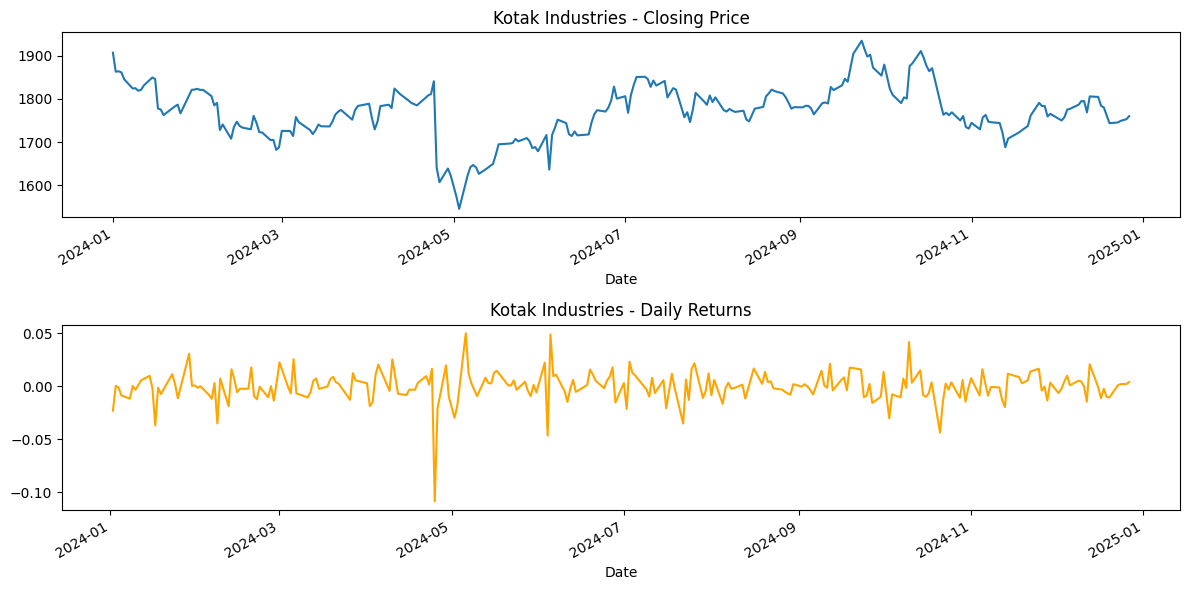
2024-01-02 -0.023099

2024-01-03 0.000456

2024-01-04 -0.001233

2024-01-05 -0.008586

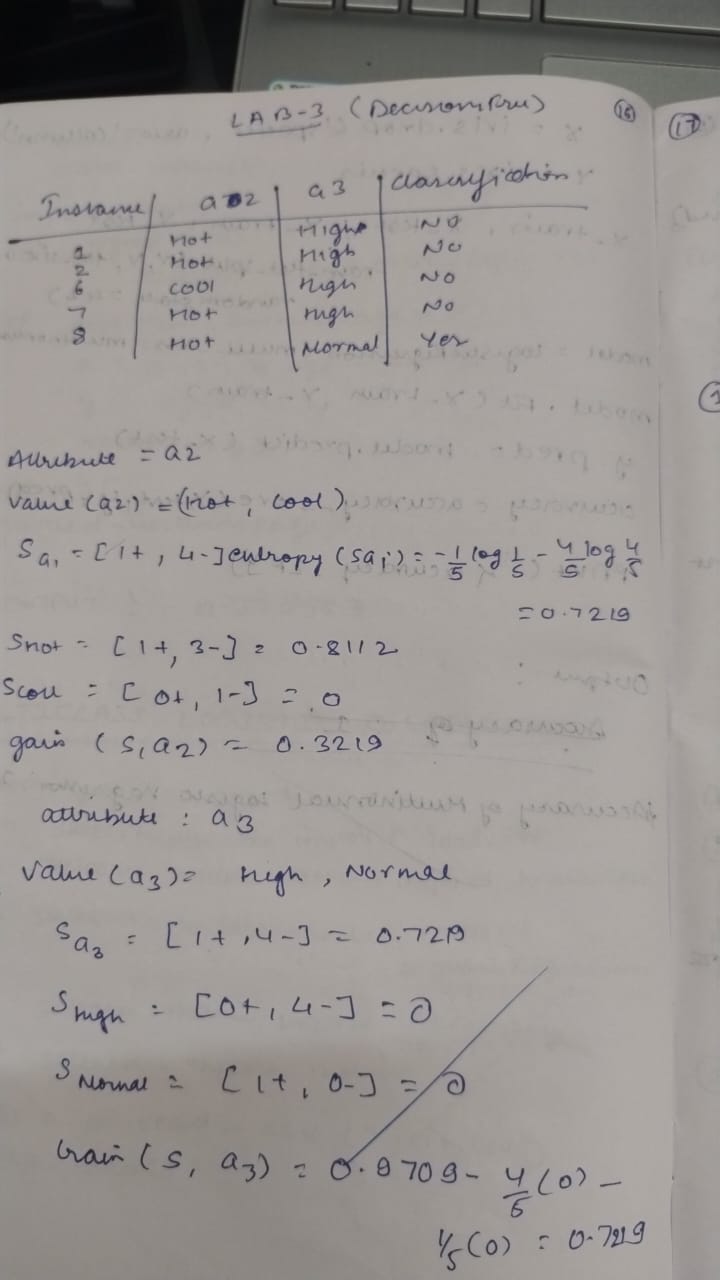
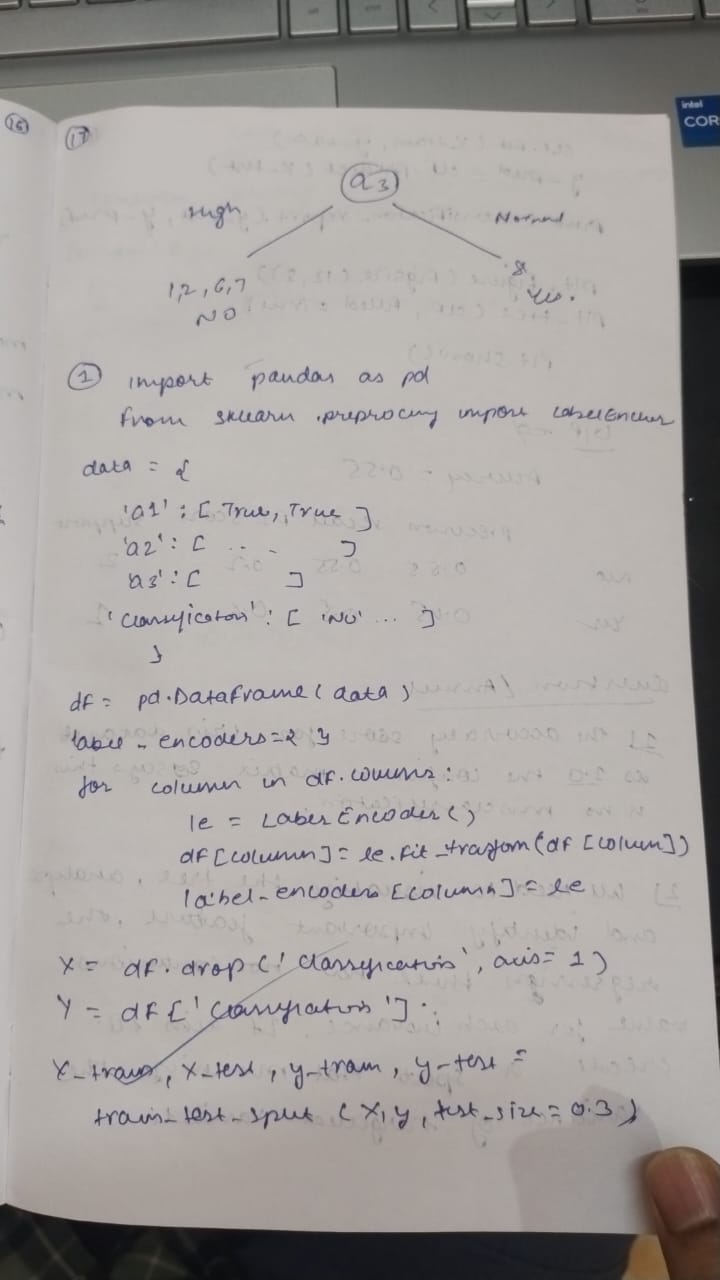
Name: Daily Return, dtype: float64

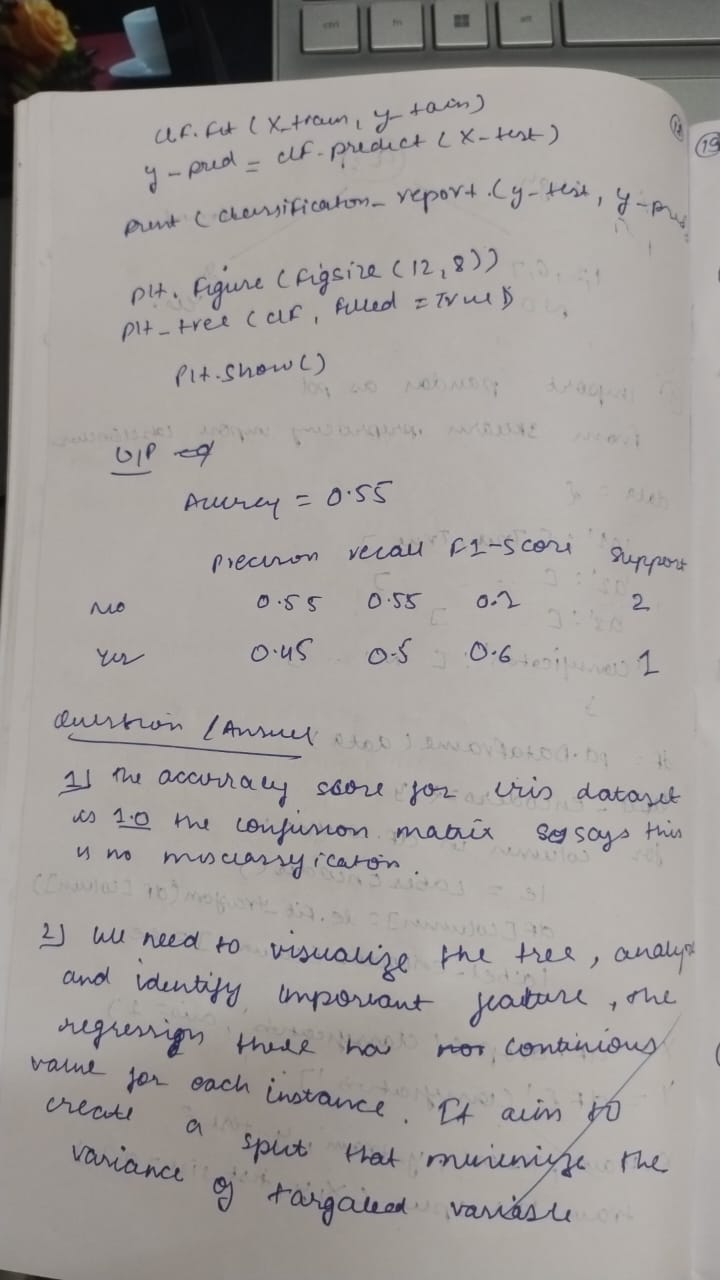


**Program 3**

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

Algorithm:

****

Code:

In [ ]:

*# -\*- coding: utf-8 -\*-*

"""Decision\_Tree.ipynb

Automatically generated by Colab.

Original file is located at

https://colab.research.google.com/drive/1RXDK8CR1doVCMHgkaXpJsNLAvzOIaXdd

"""

import pandas as pd

from sklearn.preprocessing import LabelEncoder

from sklearn.tree import DecisionTreeClassifier

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

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

*# Create the dataset*

data = {

'a1': [True, True, False, False, False, True, True, True, False, False],

'a2': ['Hot', 'Hot', 'Hot', 'Cool', 'Cool', 'Cool', 'Hot', 'Hot', 'Cool', 'Cool'],

'a3': ['High', 'High', 'High', 'Normal', 'Normal', 'High', 'High', 'Normal', 'Normal', 'High'],

'Classification': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'No', 'Yes', 'Yes', 'Yes']

}

data

*# Convert to DataFrame*

df = pd.DataFrame(data)

*# Convert categorical data to numerical data*

label\_encoders = {}

for column in df.columns:

le = LabelEncoder()

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

label\_encoders[column] = le

*# Split the dataset into features and target*

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

y = df['Classification']

*# Split the data into training and testing sets*

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

*# Initialize the Decision Tree Classifier with entropy as the criterion*

clf = DecisionTreeClassifier(criterion='entropy')

*# Train the classifier*

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

*# Make predictions*

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

*# Evaluate the classifier*

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

print(f'Accuracy: {accuracy:.2f}')

print(classification\_report(y\_test, y\_pred, target\_names=['No', 'Yes']))

*# Optionally, visualize the decision tree*

from sklearn.tree import plot\_tree

import matplotlib.pyplot as plt

plt.figure(figsize=(12,8))

plot\_tree(clf, filled=True, feature\_names=X.columns, class\_names=['No', 'Yes'])

plt.show()

Accuracy: 1.00

precision recall f1-score support

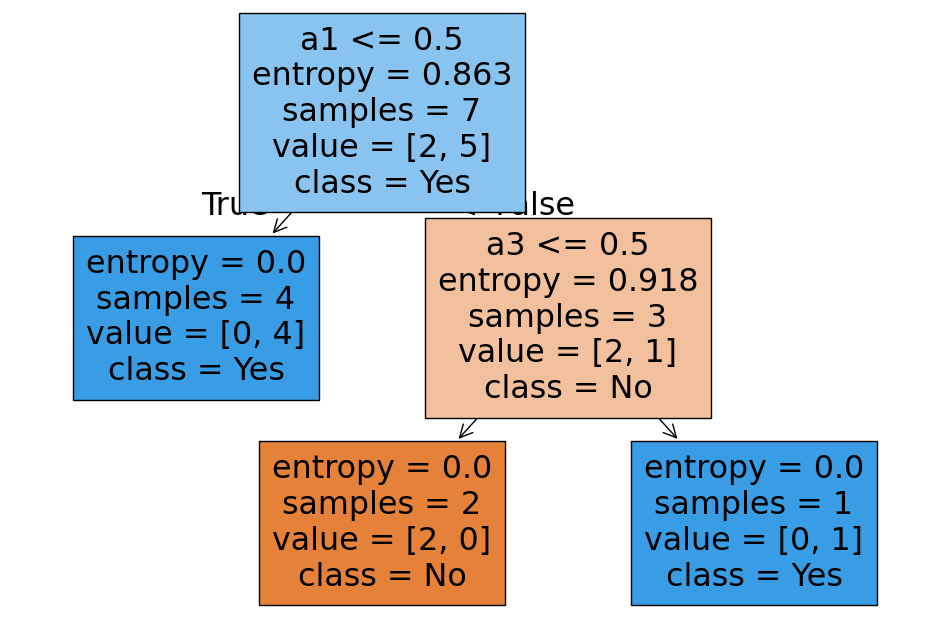
No 1.00 1.00 1.00 2

Yes 1.00 1.00 1.00 1

accuracy 1.00 3

macro avg 1.00 1.00 1.00 3

weighted avg 1.00 1.00 1.00 3



In [ ]:

from sklearn.datasets import load\_iris

from sklearn.tree import DecisionTreeClassifier

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

from sklearn.metrics import accuracy\_score, confusion\_matrix

import seaborn as sns

import matplotlib.pyplot as plt

*# Load the iris dataset*

iris = load\_iris()

X = iris.data

y = iris.target

*# Split 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)

*# Initialize and train the DecisionTreeClassifier*

clf = DecisionTreeClassifier()

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

*# Make predictions on the test set*

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

*# Evaluate the model*

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

print(f"Accuracy: {accuracy}")

*# Create and display the confusion matrix*

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

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

sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",

xticklabels=iris.target\_names, yticklabels=iris.target\_names)

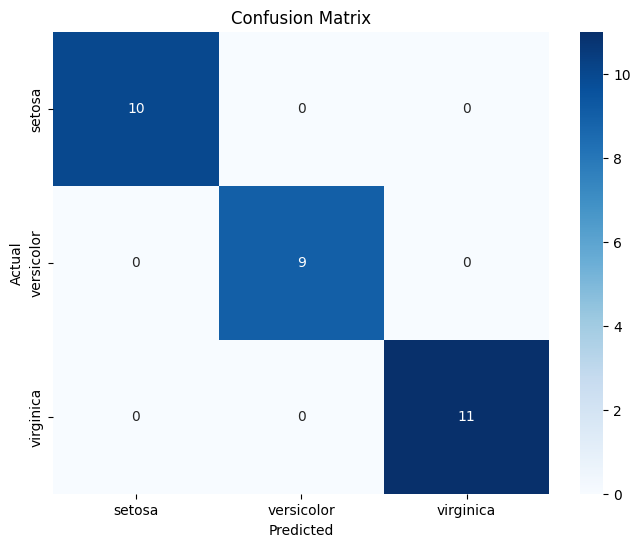
plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.title("Confusion Matrix")

plt.show()

Accuracy: 1.0



In [ ]:

import pandas as pd

from sklearn.tree import DecisionTreeClassifier

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

from sklearn.metrics import accuracy\_score, confusion\_matrix

import seaborn as sns

import matplotlib.pyplot as plt

try:

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

except FileNotFoundError:

print("Error: 'drugdataset.csv' not found. Please upload the file or provide the correct path.")

exit()

*# Identify categorical columns (replace with your actual column names)*

categorical\_cols = ['Sex', 'BP', 'Cholesterol', 'Na\_to\_K']

*# Use one-hot encoding for categorical features*

df = pd.get\_dummies(df, columns=categorical\_cols, drop\_first=True)

*# Split data into features (X) and target (y)*

X = df.drop('Drug', axis=1) *# Assuming 'Drug' is the target variable*

y = df['Drug']

*# Split data into training and testing sets (80% train, 20% test)*

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

*# Initialize the DecisionTreeClassifier*

clf = DecisionTreeClassifier()

*# Train the classifier*

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

*# Make predictions on the test data*

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

*# Calculate the accuracy score*

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

print(f"Accuracy: {accuracy}")

*# Create and display the confusion matrix*

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

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

sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")

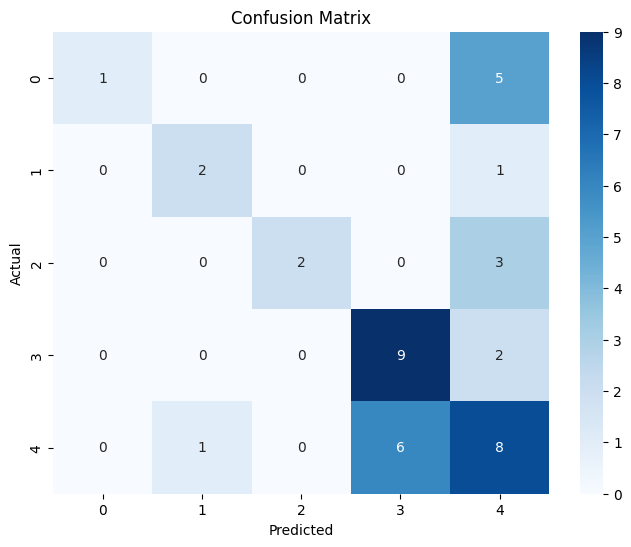
plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.title("Confusion Matrix")

plt.show()

Accuracy: 0.55



In [ ]:

import pandas as pd

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

from sklearn.tree import DecisionTreeRegressor

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error

df = pd.read\_csv('petrol\_consumption.csv')

*# Split data into features (X) and target (y)*

X = df.drop('Petrol\_Consumption', axis=1)

y = df['Petrol\_Consumption']

*# Split data into training and testing sets (80% train, 20% test)*

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

*# Initialize the DecisionTreeRegressor*

regressor = DecisionTreeRegressor()

*# Train the regressor*

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

*# Make predictions on the test data*

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

*# Evaluate the model*

mae = mean\_absolute\_error(y\_test, y\_pred)

mse = mean\_squared\_error(y\_test, y\_pred)

rmse = mse\*\*0.5

print(f"Mean Absolute Error: {mae}")

print(f"Mean Squared Error: {mse}")

print(f"Root Mean Squared Error: {rmse}")

Mean Absolute Error: 96.2

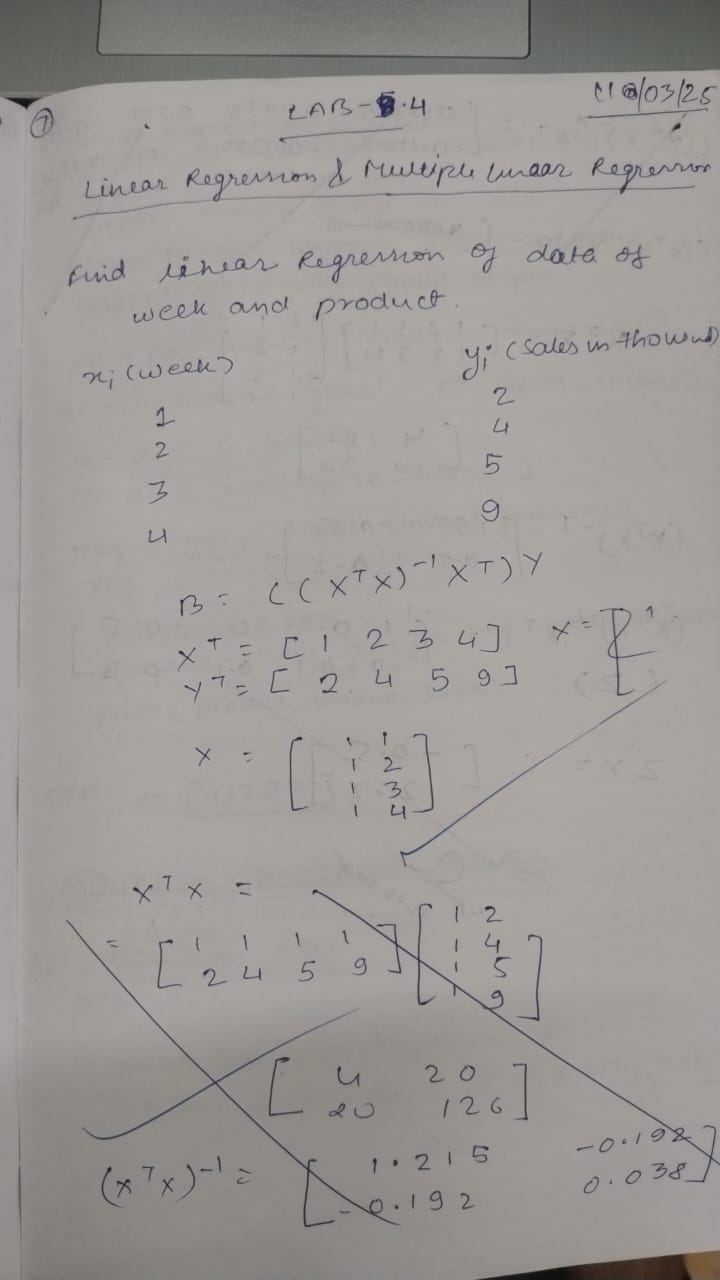
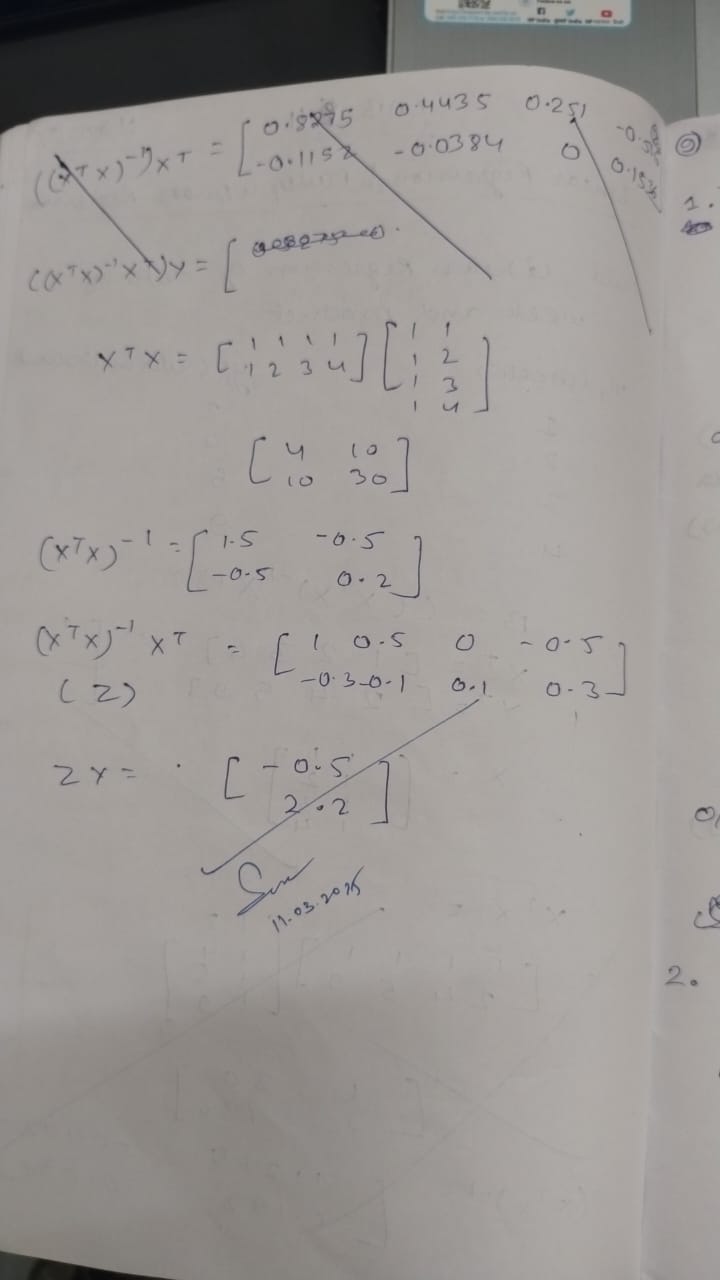
Mean Squared Error: 17858.2

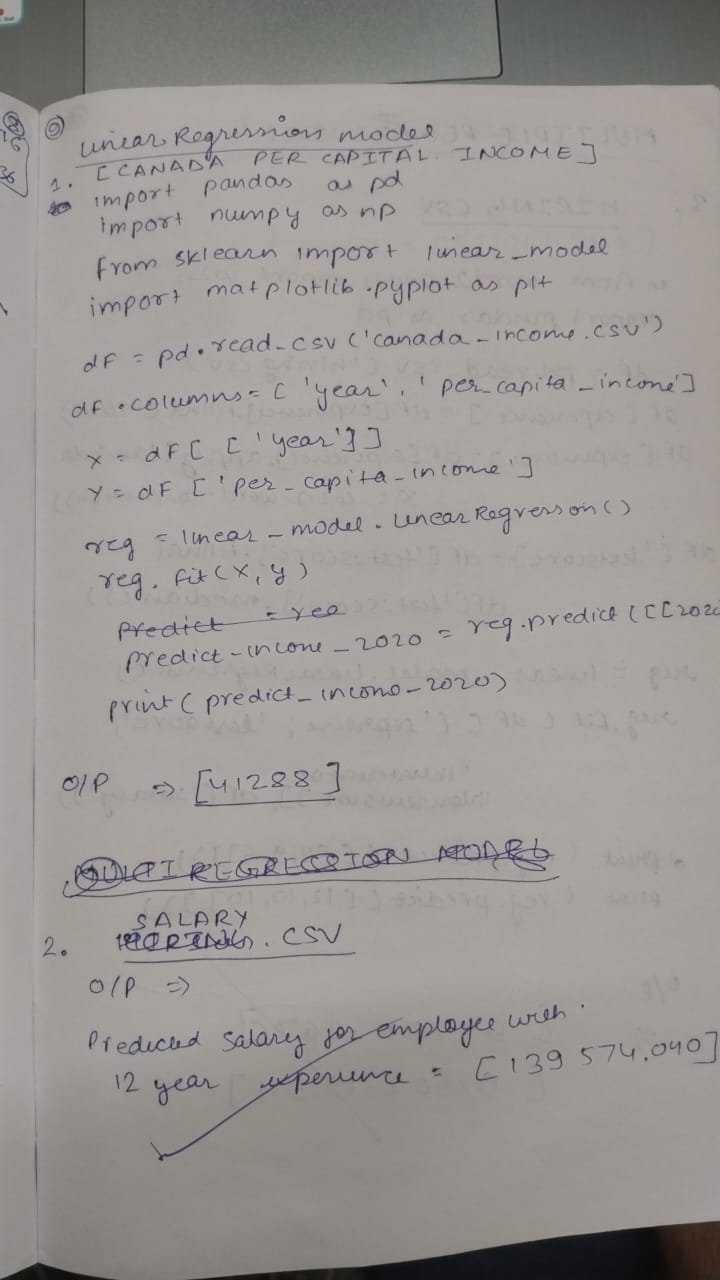
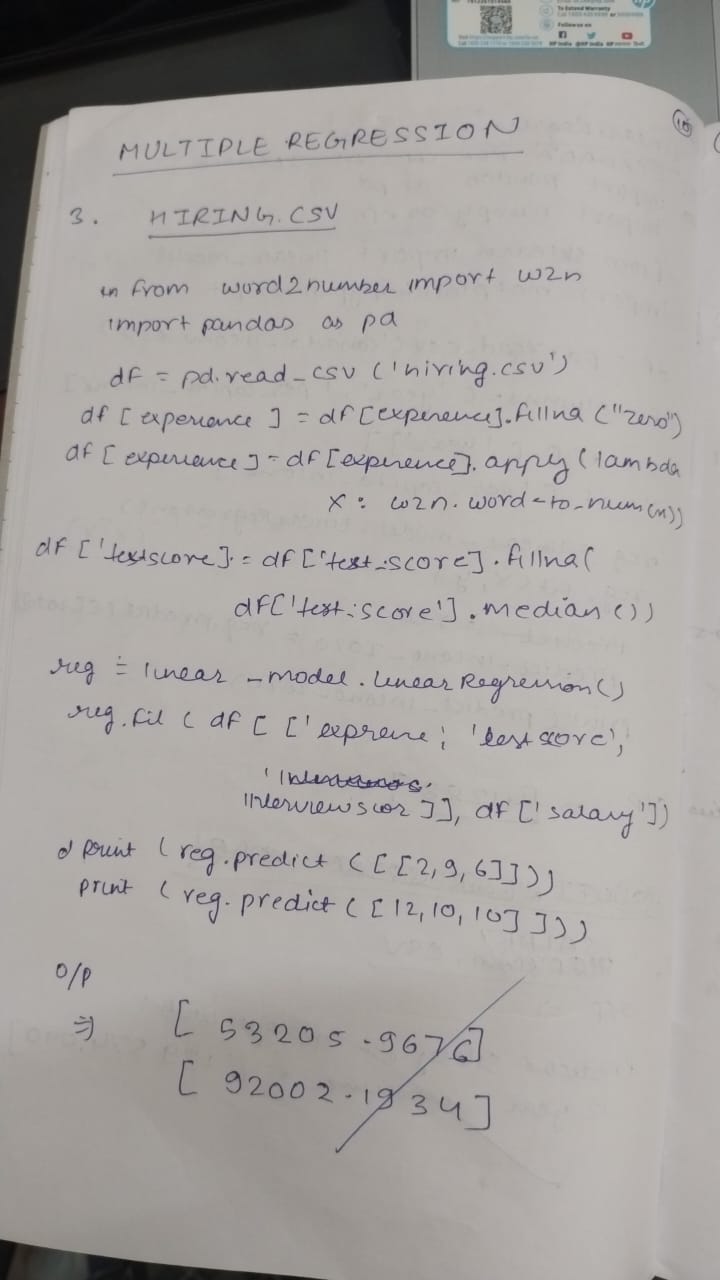
Root Mean Squared Error: 133.63457636405332

**Program 4**

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Algorithm



Code:

*# -\*- coding: utf-8 -\*-*

*#1. LINEAR\_LR*

import pandas as pd

import numpy as np

from sklearn import linear\_model

import matplotlib.pyplot as plt

df = pd.read\_csv('canada\_per\_capita\_income.csv')

df

*# Commented out IPython magic to ensure Python compatibility.*

*# %matplotlib inline*

plt.xlabel('year')

plt.ylabel('per capita income (US$)')

plt.scatter(df['year'],df['per capita income (US$)'],color='red',marker='+')

new\_df = df.drop('per capita income (US$)',axis='columns')

new\_df

price = df['per capita income (US$)']

*# Create linear regression object*

reg = linear\_model.LinearRegression()

reg.fit(new\_df,price)

print(reg.predict([[2020]]))

print(reg.coef\_)

print(reg.intercept\_)

"""Y = m \* X + b (m is coefficient and b is intercept)"""

[41288.69409442]

[828.46507522]

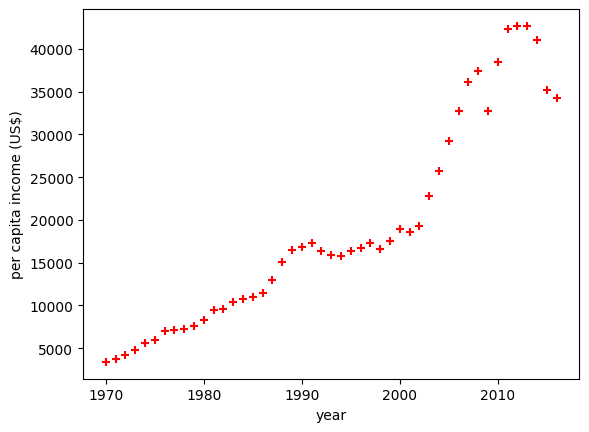
-1632210.7578554575

/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but LinearRegression was fitted with feature names

warnings.warn(

Out[31]:

'Y = m \* X + b (m is coefficient and b is intercept)'



In [15]:

*# -\*- coding: utf-8 -\*-*

*#2. LINEAR\_LR*

import pandas as pd

import numpy as np

from sklearn import linear\_model

import matplotlib.pyplot as plt

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

df

*# Commented out IPython magic to ensure Python compatibility.*

*# %matplotlib inline*

plt.xlabel('YearsExperience')

plt.ylabel('Salary')

plt.scatter(df['YearsExperience'],df['Salary'],color='red',marker='+')

df['YearsExperience']=df['YearsExperience'].fillna(df['YearsExperience'].mean())

new\_df = df.drop('Salary',axis='columns')

new\_df

salary = df['Salary']

*# Create linear regression object*

reg = linear\_model.LinearRegression()

reg.fit(new\_df,salary)

reg.predict([[12]])

print(reg.coef\_)

print(reg.intercept\_)

"""Y = m \* X + b (m is coefficient and b is intercept)"""

print(reg.predict([[12]]))

[9398.64060184]

27197.2020175696

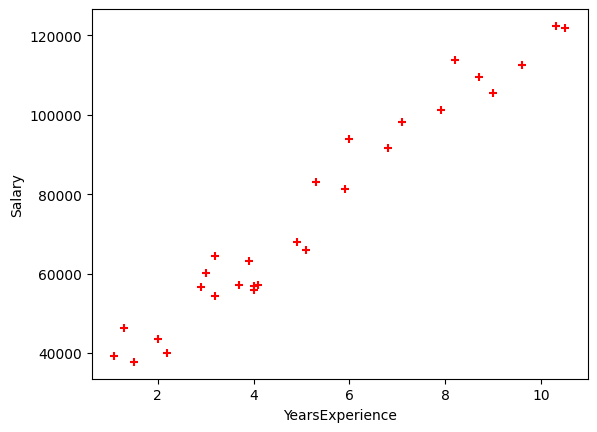
[139980.88923969]

/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but LinearRegression was fitted with feature names

warnings.warn(

/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but LinearRegression was fitted with feature names

warnings.warn(



In [17]:

!pip install word2number

Collecting word2number

Downloading word2number-1.1.zip (9.7 kB)

Preparing metadata (setup.py) ... done

Building wheels for collected packages: word2number

Building wheel for word2number (setup.py) ... done

Created wheel for word2number: filename=word2number-1.1-py3-none-any.whl size=5568 sha256=88e1bcbd080599c701a9fbaf1d8af7e1c4b020a078bd8566ebf1efa7804baf89

Stored in directory: /root/.cache/pip/wheels/cd/ef/ae/073b491b14d25e2efafcffca9e16b2ee6d114ec5c643ba4f06

Successfully built word2number

Installing collected packages: word2number

Successfully installed word2number-1.1

In [20]:

*#1 MULTIPLE\_LR*

*# -\*- coding: utf-8 -\*-*

from word2number import w2n

import pandas as pd

import numpy as np

from sklearn import linear\_model

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

df

"""Data Preprocessing: Fill NA values with median value of a column"""

df.experience = df.experience.bfill()

*# df.experience = df.experience.fillna("zero")*

*# df*

df.experience = df.experience.apply(w2n.word\_to\_num)

*# df*

df['test\_score(out of 10)'] = df['test\_score(out of 10)'].fillna(df['test\_score(out of 10)'].mean())

*# df*

reg = linear\_model.LinearRegression()

reg.fit(df.drop('salary($)',axis='columns'),df['salary($)'])

reg.coef\_

reg.intercept\_

print(reg.predict([[2, 9, 6]]))

print(reg.predict([[12, 10, 10]]))

[45968.51077949]

[92712.4732389]

/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but LinearRegression was fitted with feature names

warnings.warn(

/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but LinearRegression was fitted with feature names

warnings.warn(

In [32]:

*#2. MULTIPLE\_LR*

import pandas as pd

import numpy as np

from sklearn import linear\_model

df = pd.read\_csv('1000\_Companies.csv')

from sklearn.preprocessing import OrdinalEncoder

ord\_enc = OrdinalEncoder()

df["State"] = ord\_enc.fit\_transform(df[["State"]])

reg = linear\_model.LinearRegression()

reg.fit(df.drop('Profit',axis='columns'),df['Profit'])

print(reg.coef\_)

print(reg.intercept\_)

print(reg.predict([[91694.48, 515841.3, 11931.24,1.0]]))

[ 0.55389023 1.02672443 0.08058525 46.62387015]

-70214.44175560221

[511209.20332292]

/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but LinearRegression was fitted with feature names

warnings.warn(

In [28]:

pd.get\_dummies(df, columns=["State"]).head()

from sklearn.preprocessing import OrdinalEncoder

ord\_enc = OrdinalEncoder()

df["State"] = ord\_enc.fit\_transform(df[["State"]])

df.head()

Out[28]:

|  | **R&D Spend** | **Administration** | **Marketing Spend** | **State** | **Profit** | **state\_code** |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | 165349.20 | 136897.80 | 471784.10 | 2.0 | 192261.83 | 2.0 |
| **1** | 162597.70 | 151377.59 | 443898.53 | 0.0 | 191792.06 | 0.0 |
| **2** | 153441.51 | 101145.55 | 407934.54 | 1.0 | 191050.39 | 1.0 |
| **3** | 144372.41 | 118671.85 | 383199.62 | 2.0 | 182901.99 | 2.0 |
| **4** | 142107.34 | 91391.77 | 366168.42 | 1.0 | 166187.94 | 1.0 |

In [3]:

*# -\*- coding: utf-8 -\*-*

"""Multiple\_LR\_HomePrice.ipynb

Automatically generated by Colab.

Original file is located at

https://colab.research.google.com/drive/1fK78C8TPV44HdvT6lsMhaau2wMtKXquQ

"""

import pandas as pd

import numpy as np

from sklearn import linear\_model

df = pd.read\_csv('homeprices\_Multiple\_LR.csv')

df

"""Data Preprocessing: Fill NA values with median value of a column"""

df.bedrooms.median()

df.bedrooms = df.bedrooms.fillna(df.bedrooms.median())

df

reg = linear\_model.LinearRegression()

reg.fit(df.drop('price',axis='columns'),df.price)

reg.coef\_

reg.intercept\_

"""Find price of home with 3000 sqr ft area, 3 bedrooms, 40 year old"""

reg.predict([[3000, 3, 40]])

112.06244194\*3000 + 23388.88007794\*3 + -3231.71790863\*40 + 221323.00186540384

/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but LinearRegression was fitted with feature names

warnings.warn(

Out[3]:

498408.25157402386

In [4]:

*# -\*- coding: utf-8 -\*-*

"""Linear-Regression-Housing\_Area\_Price.ipynb

Automatically generated by Colab.

Original file is located at

https://colab.research.google.com/drive/1CAlZml-P6V2V1RIrodgMfF8L3Ux4V9FT

"""

import pandas as pd

import numpy as np

from sklearn import linear\_model

import matplotlib.pyplot as plt

df = pd.read\_csv('housing\_area\_price.csv')

df

*# Commented out IPython magic to ensure Python compatibility.*

*# %matplotlib inline*

plt.xlabel('area')

plt.ylabel('price')

plt.scatter(df.area,df.price,color='red',marker='+')

new\_df = df.drop('price',axis='columns')

new\_df

price = df.price

price

*# Create linear regression object*

reg = linear\_model.LinearRegression()

reg.fit(new\_df,price)

"""(1) Predict price of a home with area = 3300 sqr ft"""

reg.predict([[3300]])

reg.coef\_

reg.intercept\_

"""Y = m \* X + b (m is coefficient and b is intercept)"""

3300\*135.78767123 + 180616.43835616432

"""(1) Predict price of a home with area = 5000 sqr ft"""

reg.predict([[5000]])

/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but LinearRegression was fitted with feature names

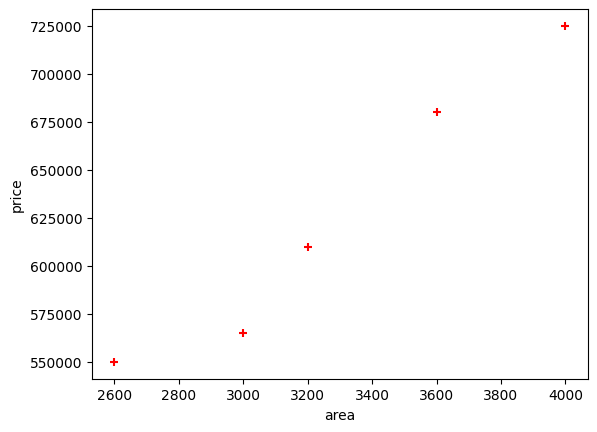
warnings.warn(

/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but LinearRegression was fitted with feature names

warnings.warn(

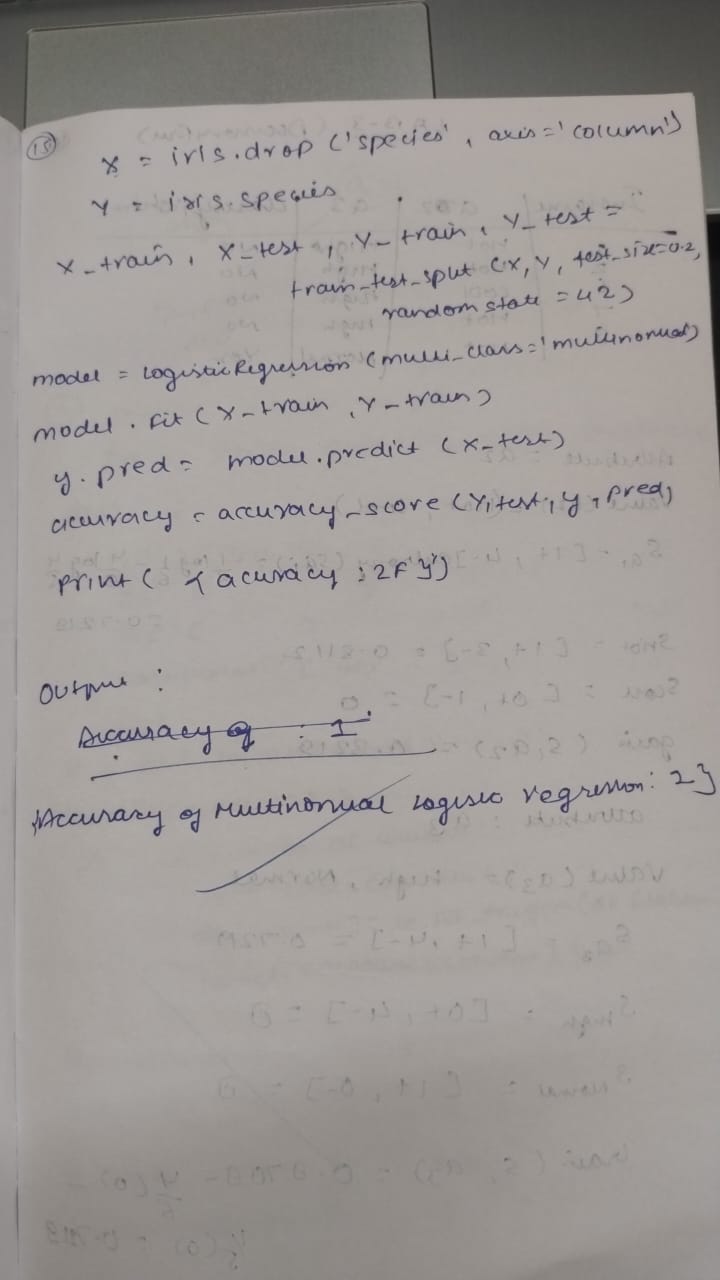
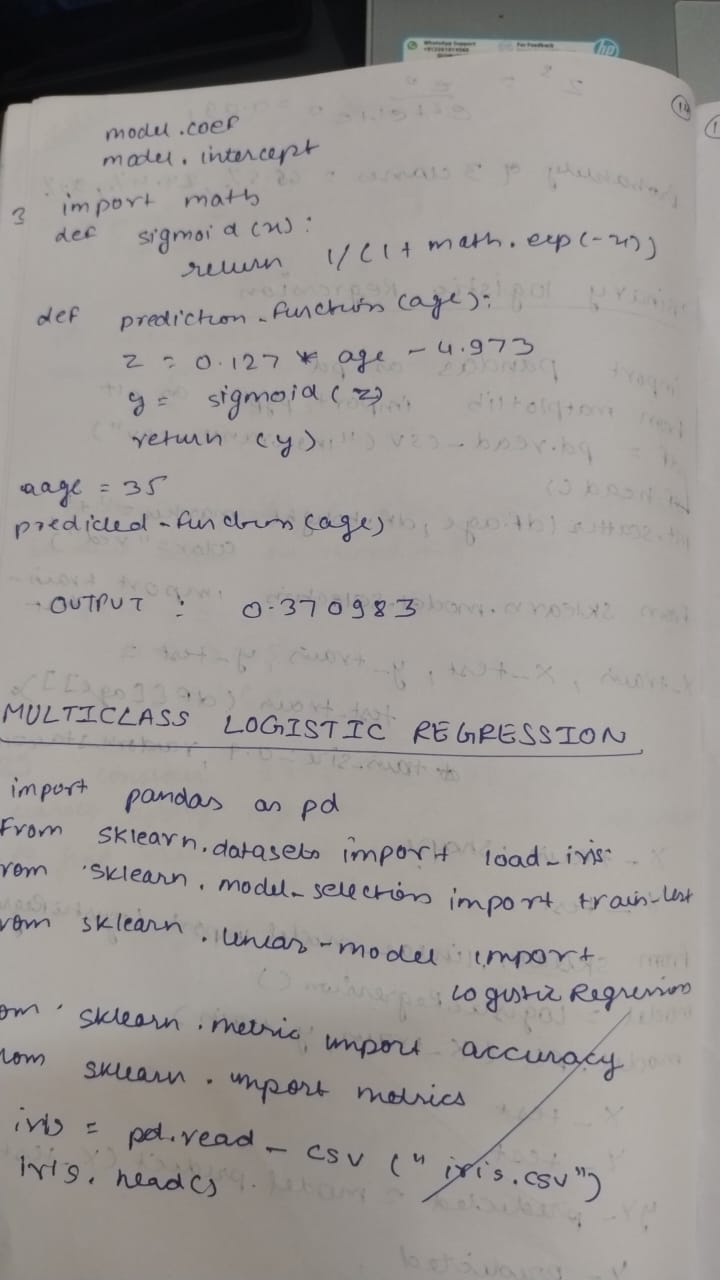
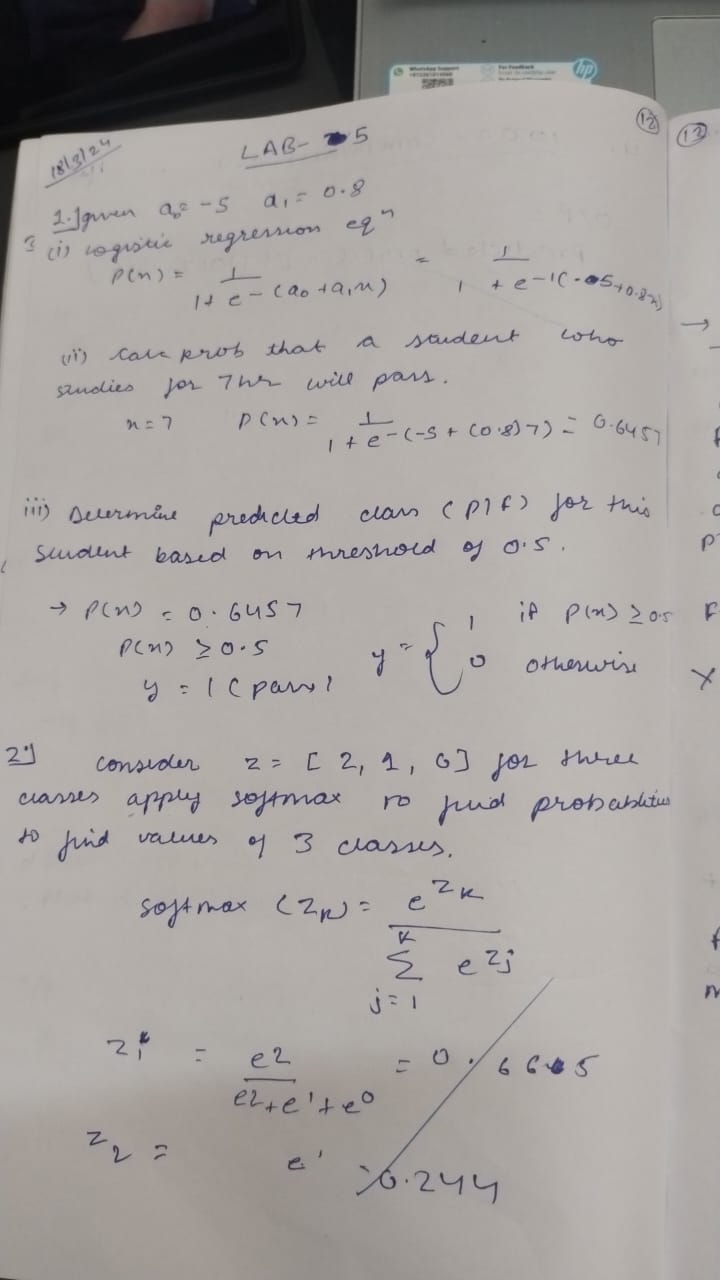
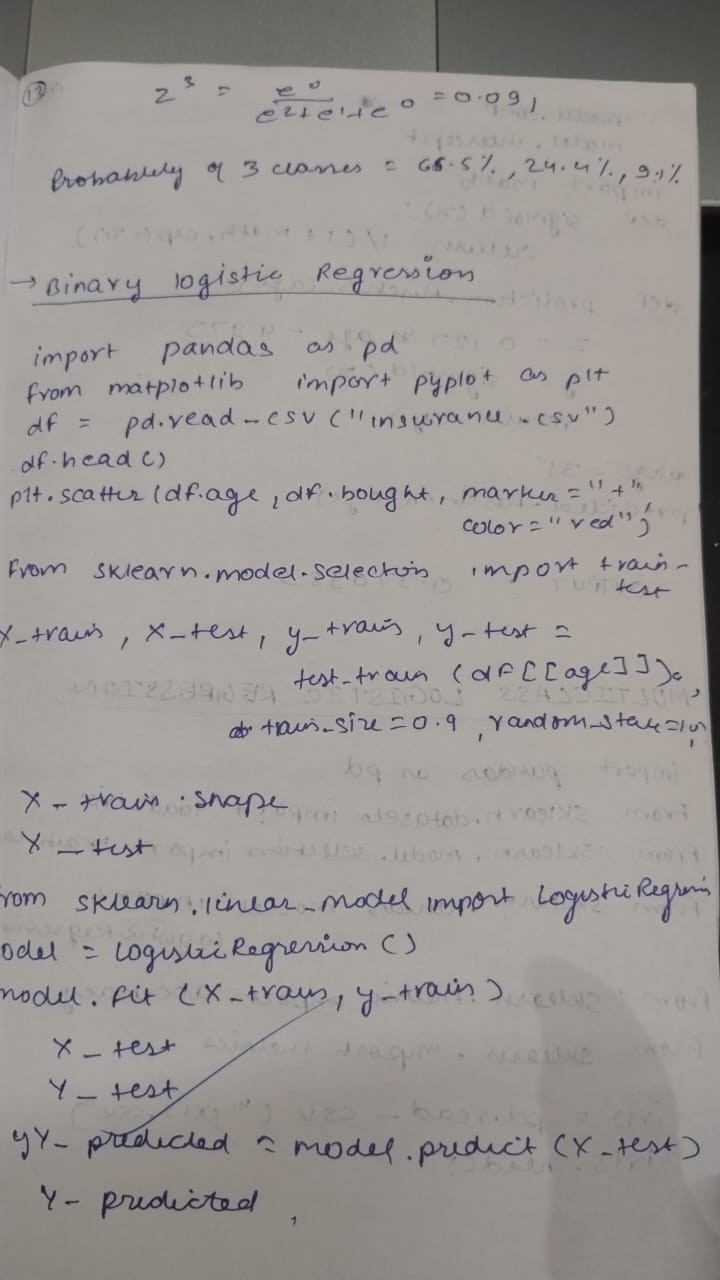
Out[4]:

array([859554.79452055])



In [ ]:

**Program 5**

Build Logistic Regression Model for a given dataset  
Algorithm  


Code:

In [ ]:

*# -\*- coding: utf-8 -\*-*

"""LogisticRegression\_Binary.ipynb

Automatically generated by Colab.

Original file is located at

https://colab.research.google.com/drive/1M8PXdcmPsrQtqyVXpET3sgghAMr\_MCg5

"""

*# Commented out IPython magic to ensure Python compatibility.*

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)

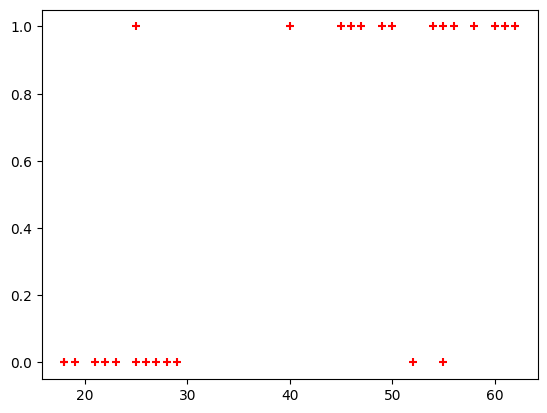
"""0.37 is less than 0.5 which means person with 35 will not buy the insurance"""

/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(

Out[ ]:

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



In [ ]:

*# -\*- coding: utf-8 -\*-*

"""LogisticRegression\_Multiclass.ipynb

Automatically generated by Colab.

Original file is located at

https://colab.research.google.com/drive/1anBybVXILenh0a\_R4aM\_ZemLrEqYWnJl

"""

*# Import necessary libraries*

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"])

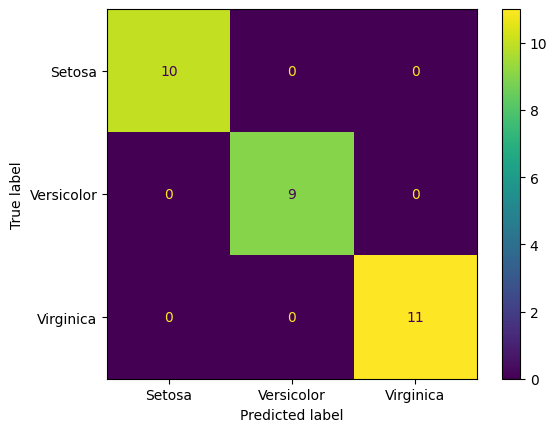
cm\_display.plot()

plt.show()

Accuracy of the Multinomial Logistic Regression model on the test set: 1.00

/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(



In [ ]:

*# Importing necessary libraries*

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

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

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, confusion\_matrix

*# Load the dataset*

df = pd.read\_csv('HR\_comma\_sep.csv')

*# 1. Exploratory Data Analysis (EDA)*

*# Check the first few rows of the dataset*

print(df.head())

*# Check for missing values*

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

*# Describe the dataset (summary statistics)*

print(df.describe())

*# Check the distribution of 'left' column (Employee Retention: 1 means left, 0 means stayed)*

sns.countplot(x='left', data=df)

plt.title("Employee Retention (Stayed vs Left)")

plt.show()

*# 2. Impact of Employee Salaries on Retention*

*# Plotting bar chart showing impact of employee salaries on retention*

sns.countplot(x='salary', hue='left', data=df)

plt.title("Impact of Employee Salaries on Retention")

plt.show()

*# 3. Correlation between Department and Employee Retention*

*# Plotting bar chart showing correlation between department and retention*

sns.countplot(x='Department', hue='left', data=df)

plt.title("Correlation Between Department and Employee Retention")

plt.xticks(rotation=45) *# Rotate x-axis labels for better readability*

plt.show()

*# 4. Build Logistic Regression Model*

*# Convert categorical columns to numeric using one-hot encoding*

df\_encoded = pd.get\_dummies(df, drop\_first=True)

*# Split the dataset into features (X) and target (y)*

X = df\_encoded.drop('left', axis=1) *# Features (drop the 'left' column as it's the target)*

y = df\_encoded['left'] *# Target: whether the employee left (1) or stayed (0)*

*# Split the data into training and testing sets (80% training, 20% testing)*

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

*# Initialize the Logistic Regression model*

logreg\_model = LogisticRegression()

*# Fit the model to the training data*

logreg\_model.fit(X\_train, y\_train)

*# 5. Measure Model Accuracy*

*# Predict on the test set*

y\_pred = logreg\_model.predict(X\_test)

*# Calculate accuracy*

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

print(f"Accuracy of Logistic Regression Model: {accuracy \* 100:.2f}%")

*# Plot confusion matrix*

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

sns.heatmap(conf\_matrix, annot=True, fmt="d", cmap="Blues", cbar=False)

plt.title("Confusion Matrix")

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.show()

satisfaction\_level last\_evaluation number\_project average\_montly\_hours \

0 0.38 0.53 2 157

1 0.80 0.86 5 262

2 0.11 0.88 7 272

3 0.72 0.87 5 223

4 0.37 0.52 2 159

time\_spend\_company Work\_accident left promotion\_last\_5years Department \

0 3 0 1 0 sales

1 6 0 1 0 sales

2 4 0 1 0 sales

3 5 0 1 0 sales

4 3 0 1 0 sales

salary

0 low

1 medium

2 medium

3 low

4 low

satisfaction\_level 0

last\_evaluation 0

number\_project 0

average\_montly\_hours 0

time\_spend\_company 0

Work\_accident 0

left 0

promotion\_last\_5years 0

Department 0

salary 0

dtype: int64

satisfaction\_level last\_evaluation number\_project \

count 14999.000000 14999.000000 14999.000000

mean 0.612834 0.716102 3.803054

std 0.248631 0.171169 1.232592

min 0.090000 0.360000 2.000000

25% 0.440000 0.560000 3.000000

50% 0.640000 0.720000 4.000000

75% 0.820000 0.870000 5.000000

max 1.000000 1.000000 7.000000

average\_montly\_hours time\_spend\_company Work\_accident left \

count 14999.000000 14999.000000 14999.000000 14999.000000

mean 201.050337 3.498233 0.144610 0.238083

std 49.943099 1.460136 0.351719 0.425924

min 96.000000 2.000000 0.000000 0.000000

25% 156.000000 3.000000 0.000000 0.000000

50% 200.000000 3.000000 0.000000 0.000000

75% 245.000000 4.000000 0.000000 0.000000

max 310.000000 10.000000 1.000000 1.000000

promotion\_last\_5years

count 14999.000000

mean 0.021268

std 0.144281

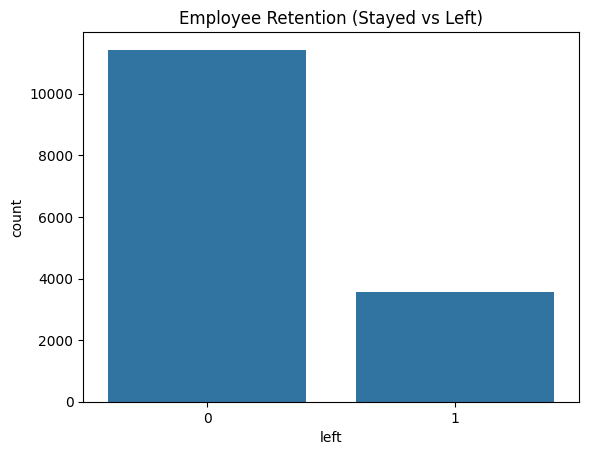
min 0.000000

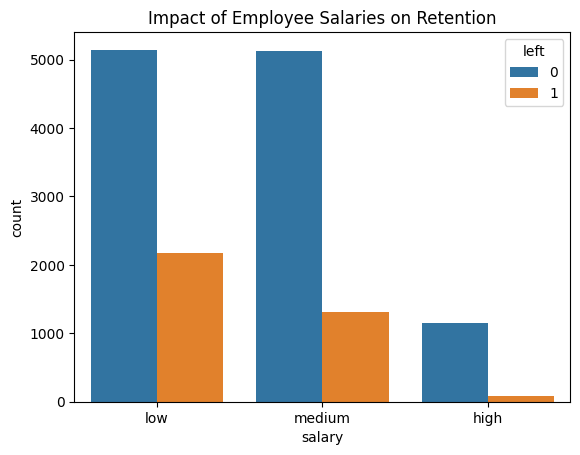
25% 0.000000

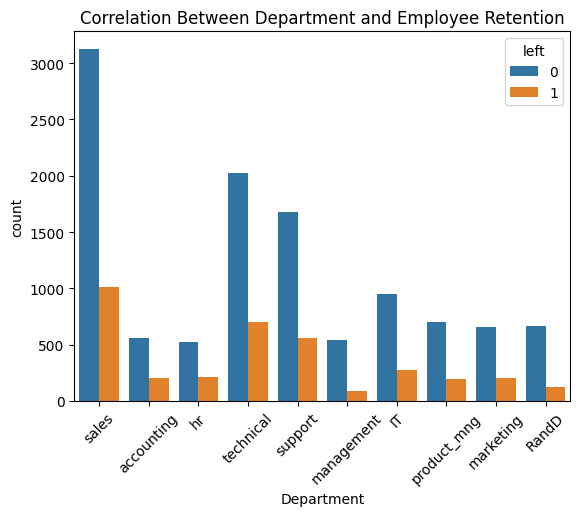
50% 0.000000

75% 0.000000

max 1.000000







Accuracy of Logistic Regression Model: 79.63%

/usr/local/lib/python3.11/dist-packages/sklearn/linear\_model/\_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=1):

STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

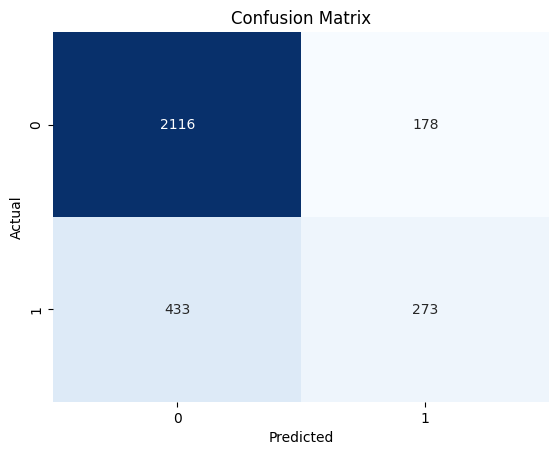
Increase the number of iterations (max\_iter) or scale the data as shown in:

https://scikit-learn.org/stable/modules/preprocessing.html

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear\_model.html#logistic-regression

n\_iter\_i = \_check\_optimize\_result(



In [ ]:

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

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

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, confusion\_matrix

import numpy as np

*# Load the datasets*

zoo\_data = pd.read\_csv('zoo-data.csv')

class\_type = pd.read\_csv('zoo-class-type.csv')

*# 1. Data Preprocessing: Merge datasets to get class\_type*

zoo\_data = zoo\_data.merge(class\_type[['Class\_Number', 'Class\_Type']], how='left', left\_on='class\_type', right\_on='Class\_Number')

*# Drop the unnecessary 'Class\_Number' column and keep 'Class\_Type' as the target variable*

zoo\_data.drop(columns=['Class\_Number'], inplace=True)

*# Check for missing values*

print(zoo\_data.isnull().sum())

*# Since we don't have missing values, we can proceed to encode categorical variables and model building.*

*# 2. Feature Engineering: Encode categorical variables*

*# Here, we'll encode 'class\_type' which is now 'Class\_Type' into numeric labels using LabelEncoder*

from sklearn.preprocessing import LabelEncoder

label\_encoder = LabelEncoder()

*# Convert 'Class\_Type' (target) into numerical labels*

zoo\_data['Class\_Type'] = label\_encoder.fit\_transform(zoo\_data['Class\_Type'])

*# 3. Split the data into features (X) and target (y)*

X = zoo\_data.drop(['animal\_name', 'class\_type', 'Class\_Type'], axis=1) *# Features*

y = zoo\_data['Class\_Type'] *# Target*

*# 4. Train-Test Split*

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

*# 5. Build Logistic Regression Model*

logreg\_model = LogisticRegression(max\_iter=1000, multi\_class='ovr') *# Using 'ovr' for multiclass classification*

logreg\_model.fit(X\_train, y\_train)

*# 6. Predictions and Accuracy*

y\_pred = logreg\_model.predict(X\_test)

*# Accuracy*

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

print(f"Accuracy of Logistic Regression Model: {accuracy \* 100:.2f}%")

*# 7. Confusion Matrix*

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

*# Plotting Confusion Matrix*

sns.heatmap(conf\_matrix, annot=True, fmt="d", cmap="Blues", cbar=False, xticklabels=label\_encoder.classes\_, yticklabels=label\_encoder.classes\_)

plt.title("Confusion Matrix for Logistic Regression Model")

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.show()

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

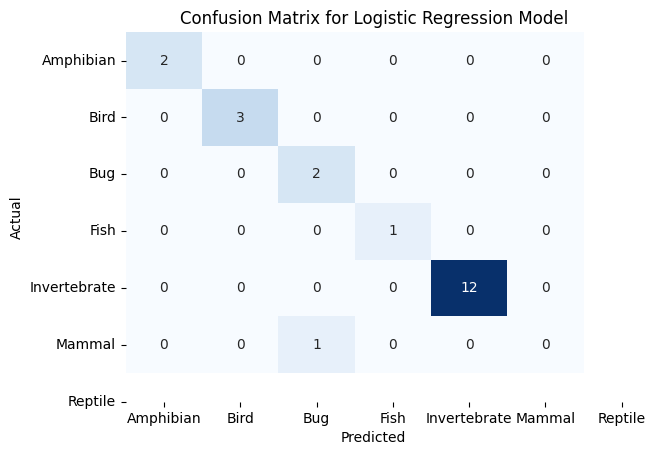
Class\_Type 0

dtype: int64

Accuracy of Logistic Regression Model: 95.24%

/usr/local/lib/python3.11/dist-packages/sklearn/linear\_model/\_logistic.py:1256: FutureWarning: 'multi\_class' was deprecated in version 1.5 and will be removed in 1.7. Use OneVsRestClassifier(LogisticRegression(..)) instead. Leave it to its default value to avoid this warning.

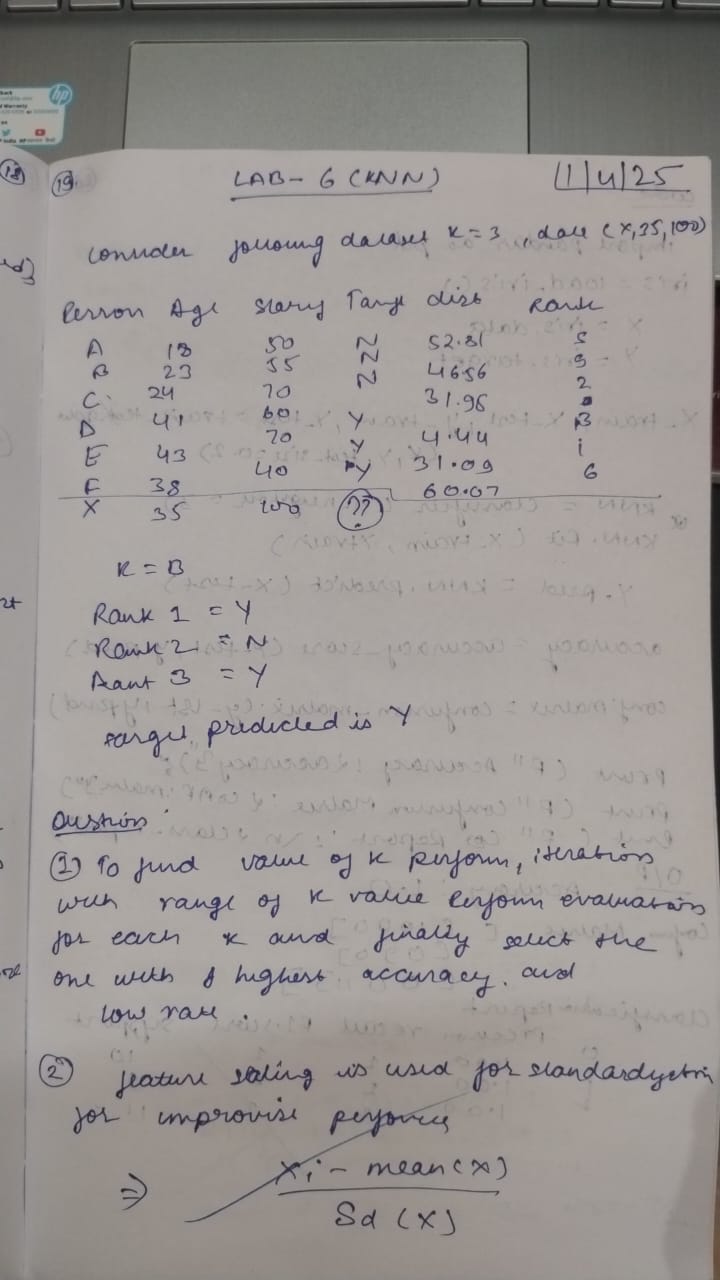
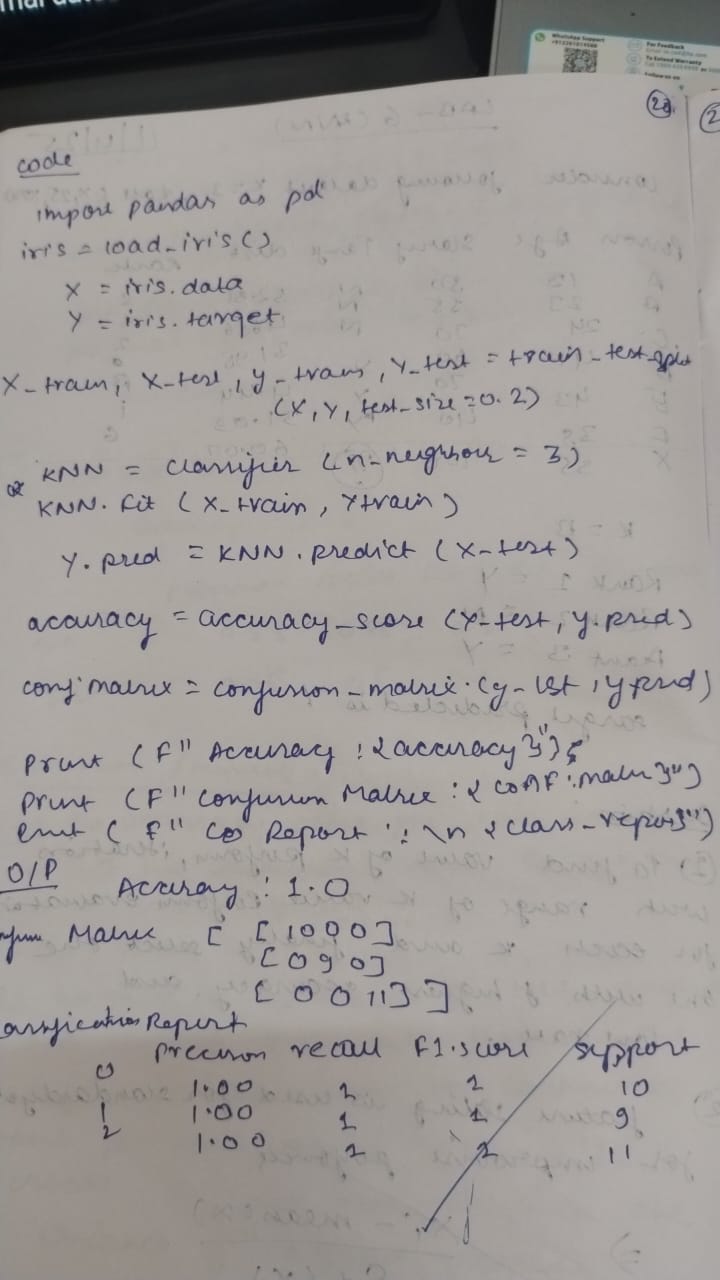
warnings.warn(



In [ ]:

**Program 6**

Build KNN Classification model for a given dataset.

Algorithm  


Code

IRIS DATASET

In [3]:

import pandas as pd

*# Assuming the uploaded file is named 'iris.csv'*

file\_path = 'iris.csv' *# The path to your uploaded file*

*# Load the dataset into a pandas DataFrame*

df = pd.read\_csv(file\_path)

*# Display the first few rows to ensure the dataset loaded correctly*

print(df.head())

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

from sklearn.neighbors import KNeighborsClassifier

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

from sklearn.preprocessing import LabelEncoder

*# Assuming the dataset has columns: sepal\_length, sepal\_width, petal\_length, petal\_width, species*

X = df[['sepal\_length', 'sepal\_width', 'petal\_length', 'petal\_width']] *# Features*

y = df['species'] *# Target label*

*# Encode the target labels (species) if needed*

le = LabelEncoder()

y = le.fit\_transform(y)

*# Split the data 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)

*# Create and train the KNN model*

knn = KNeighborsClassifier(n\_neighbors=5)

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

*# Make predictions on the test set*

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

*# Evaluate the model*

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

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

class\_report = classification\_report(y\_test, y\_pred)

*# Print results*

print("Accuracy:", accuracy)

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

print("Classification Report:\n", class\_report)

sepal\_length sepal\_width petal\_length petal\_width species

0 5.1 3.5 1.4 0.2 setosa

1 4.9 3.0 1.4 0.2 setosa

2 4.7 3.2 1.3 0.2 setosa

3 4.6 3.1 1.5 0.2 setosa

4 5.0 3.6 1.4 0.2 setosa

Accuracy: 1.0

Confusion Matrix:

[[10 0 0]

[ 0 9 0]

[ 0 0 11]]

Classification Report:

precision recall f1-score support

0 1.00 1.00 1.00 10

1 1.00 1.00 1.00 9

2 1.00 1.00 1.00 11

accuracy 1.00 30

macro avg 1.00 1.00 1.00 30

weighted avg 1.00 1.00 1.00 30

DIABETES PREDICTION

In [4]:

import pandas as pd

import numpy as np

In [5]:

data = pd.read\_csv("diabetes.csv")

data.head()

Out[5]:

|  | **Pregnancies** | **Glucose** | **BloodPressure** | **SkinThickness** | **Insulin** | **BMI** | **DiabetesPedigreeFunction** | **Age** | **Outcome** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 6 | 148 | 72 | 35 | 0 | 33.6 | 0.627 | 50 | 1 |
| **1** | 1 | 85 | 66 | 29 | 0 | 26.6 | 0.351 | 31 | 0 |
| **2** | 8 | 183 | 64 | 0 | 0 | 23.3 | 0.672 | 32 | 1 |
| **3** | 1 | 89 | 66 | 23 | 94 | 28.1 | 0.167 | 21 | 0 |
| **4** | 0 | 137 | 40 | 35 | 168 | 43.1 | 2.288 | 33 | 1 |

In [20]:

from sklearn.preprocessing import StandardScaler

sc\_X = StandardScaler()

X = pd.DataFrame(sc\_X.fit\_transform(data\_copy.drop(["Outcome"], axis =1),),columns=['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin','BMI', 'DiabetesPedigreeFunction', 'Age'])

In [21]:

X.head()

Out[21]:

|  | **Pregnancies** | **Glucose** | **BloodPressure** | **SkinThickness** | **Insulin** | **BMI** | **DiabetesPedigreeFunction** | **Age** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 0.639947 | 0.865108 | -0.033518 | 0.670643 | -0.181541 | 0.166619 | 0.468492 | 1.425995 |
| **1** | -0.844885 | -1.206162 | -0.529859 | -0.012301 | -0.181541 | -0.852200 | -0.365061 | -0.190672 |
| **2** | 1.233880 | 2.015813 | -0.695306 | -0.012301 | -0.181541 | -1.332500 | 0.604397 | -0.105584 |
| **3** | -0.844885 | -1.074652 | -0.529859 | -0.695245 | -0.540642 | -0.633881 | -0.920763 | -1.041549 |
| **4** | -1.141852 | 0.503458 | -2.680669 | 0.670643 | 0.316566 | 1.549303 | 5.484909 | -0.020496 |

In [22]:

y =data\_copy.Outcome

In [23]:

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 1/3, random\_state = 42, stratify=y)

In [24]:

from sklearn.neighbors import KNeighborsClassifier

train\_scores = []

test\_scores = []

for i in range(1,15):

knn = KNeighborsClassifier(i)

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

train\_scores.append(knn.score(X\_train, y\_train))

test\_scores.append(knn.score(X\_test, y\_test))

In [25]:

max\_test\_score =max(test\_scores)

In [26]:

test\_score\_index = [i for i, v in enumerate(test\_scores) if v== max\_test\_score]

print('Max test score {} % and k = {}'.format(max\_test\_score\*100,list(map(lambda x: x+1, test\_score\_index))))

Max test score 76.5625 % and k = [11]

In [28]:

*# K=11*

*#Setup a knn classifier with k neighbors*

knn = KNeighborsClassifier(11)

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

knn.score(X\_test,y\_test)

Out[28]:

0.765625

In [29]:

from mlxtend.plotting import plot\_decision\_regions

value = 20000

width =20000

plot\_decision\_regions(X.values, y.values, clf = knn, legend =2,filler\_feature\_values={2: value, 3: value, 4: value, 5: value, 6: value, 7: value},

filler\_feature\_ranges={2: width, 3: width, 4: width, 5: width, 6: width, 7: width},

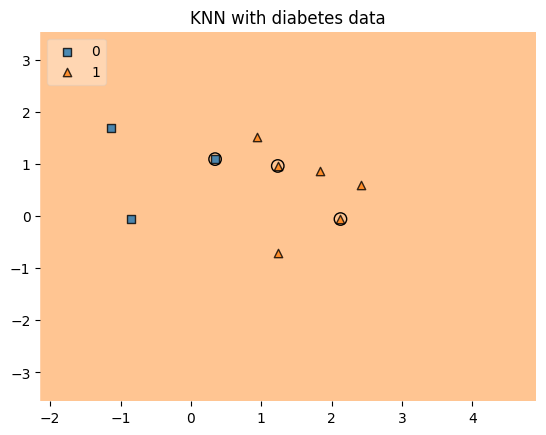
X\_highlight=X\_test.values)

plt.title("KNN with diabetes data")

plt.show()

/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but KNeighborsClassifier was fitted with feature names

warnings.warn(



In [30]:

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, fbeta\_score

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

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

In [31]:

p = sns.heatmap(pd.DataFrame(cnf\_matrix), annot=True, cmap="YlGnBu" ,fmt='g')

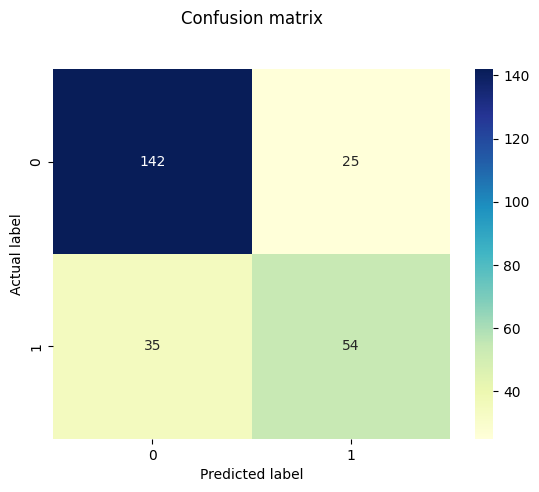
plt.title('Confusion matrix', y=1.1)

plt.ylabel('Actual label')

plt.xlabel('Predicted label')

Out[31]:

Text(0.5, 23.52222222222222, 'Predicted label')



In [32]:

def model\_evaluation(y\_test, y\_pred, model\_name):

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

prec = precision\_score(y\_test, y\_pred)

rec = recall\_score(y\_test, y\_pred)

f1 = f1\_score(y\_test, y\_pred)

f2 = fbeta\_score(y\_test, y\_pred, beta = 2.0)

results = pd.DataFrame([[model\_name, acc, prec, rec, f1, f2]],

columns = ["Model", "Accuracy", "Precision", "Recall",

"F1 SCore", "F2 Score"])

results = results.sort\_values(["Precision", "Recall", "F2 Score"], ascending = False)

return results

model\_evaluation(y\_test, y\_pred, "KNN")

Out[32]:

|  | **Model** | **Accuracy** | **Precision** | **Recall** | **F1 SCore** | **F2 Score** |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | KNN | 0.765625 | 0.683544 | 0.606742 | 0.642857 | 0.62069 |

HEART DISEASE PREDICTION

In [62]:

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

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

from sklearn.neighbors import KNeighborsClassifier

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

from sklearn.preprocessing import StandardScaler

*# Load the dataset*

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

*# Define features (X) and target (y)*

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

y = df['target']

*# Split 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)

*# Feature scaling*

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

*# Find the best k value*

best\_k = 1

best\_accuracy = 0

for k in range(1, 31): *# Test k values from 1 to 30*

knn = KNeighborsClassifier(n\_neighbors=k)

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

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

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

if accuracy > best\_accuracy:

best\_accuracy = accuracy

best\_k = k

print(f"Best k: {best\_k}")

*# Train the KNN classifier with the best k value*

knn = KNeighborsClassifier(n\_neighbors=best\_k)

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

*# Make predictions*

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

*# Evaluate the model*

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

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

class\_report = classification\_report(y\_test, y\_pred)

*# Display results*

print(f"Accuracy: {accuracy}")

print(f"Confusion Matrix:\n{conf\_matrix}")

print(f"Classification Report:\n{class\_report}")

*# Plot confusion matrix*

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

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues',

xticklabels=['No Disease', 'Disease'],

yticklabels=['No Disease', 'Disease'])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()

*# Plot classification report (text-based, can't be plotted directly)*

print("Classification Report:\n", class\_report)

Best k: 7

Accuracy: 0.9180327868852459

Confusion Matrix:

[[27 2]

[ 3 29]]

Classification Report:

precision recall f1-score support

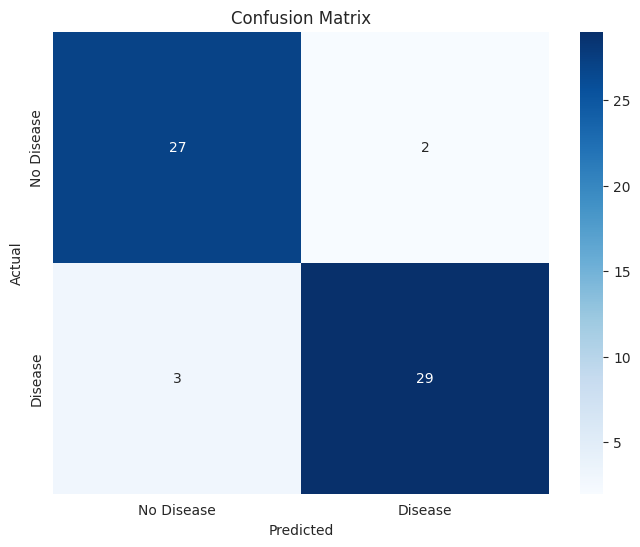
0 0.90 0.93 0.92 29

1 0.94 0.91 0.92 32

accuracy 0.92 61

macro avg 0.92 0.92 0.92 61

weighted avg 0.92 0.92 0.92 61



Classification Report:

precision recall f1-score support

0 0.90 0.93 0.92 29

1 0.94 0.91 0.92 32

accuracy 0.92 61

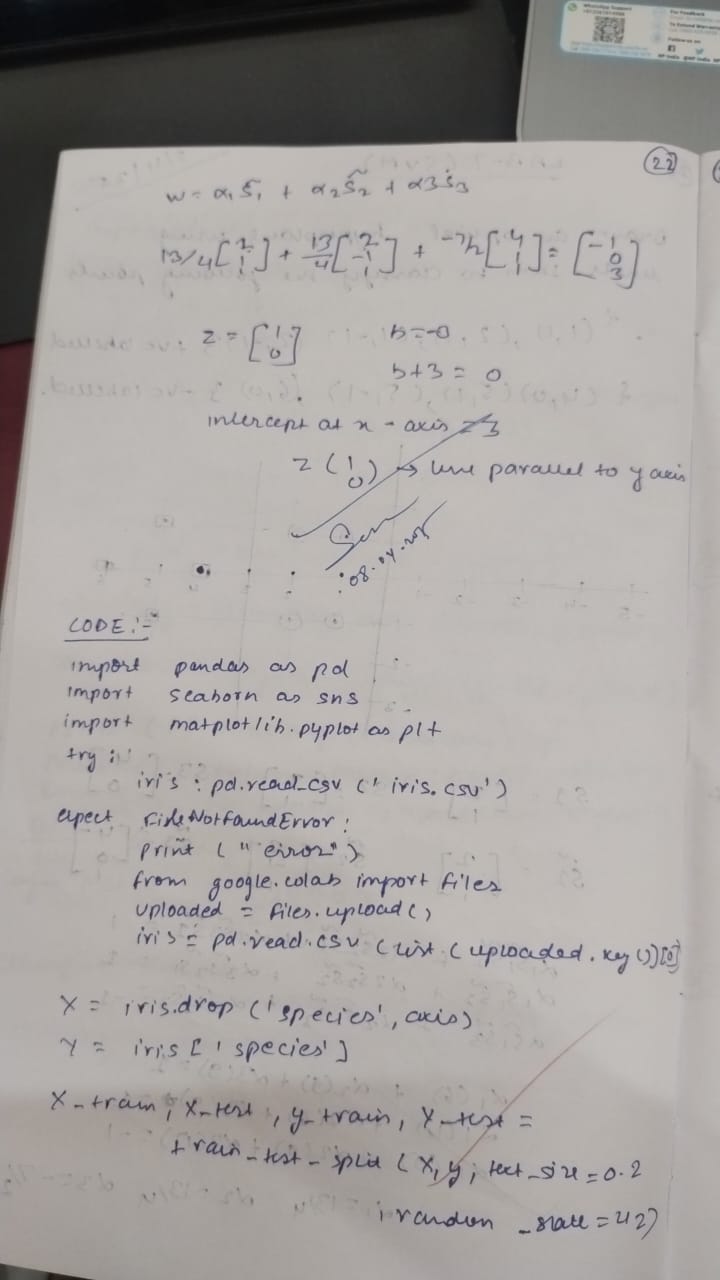
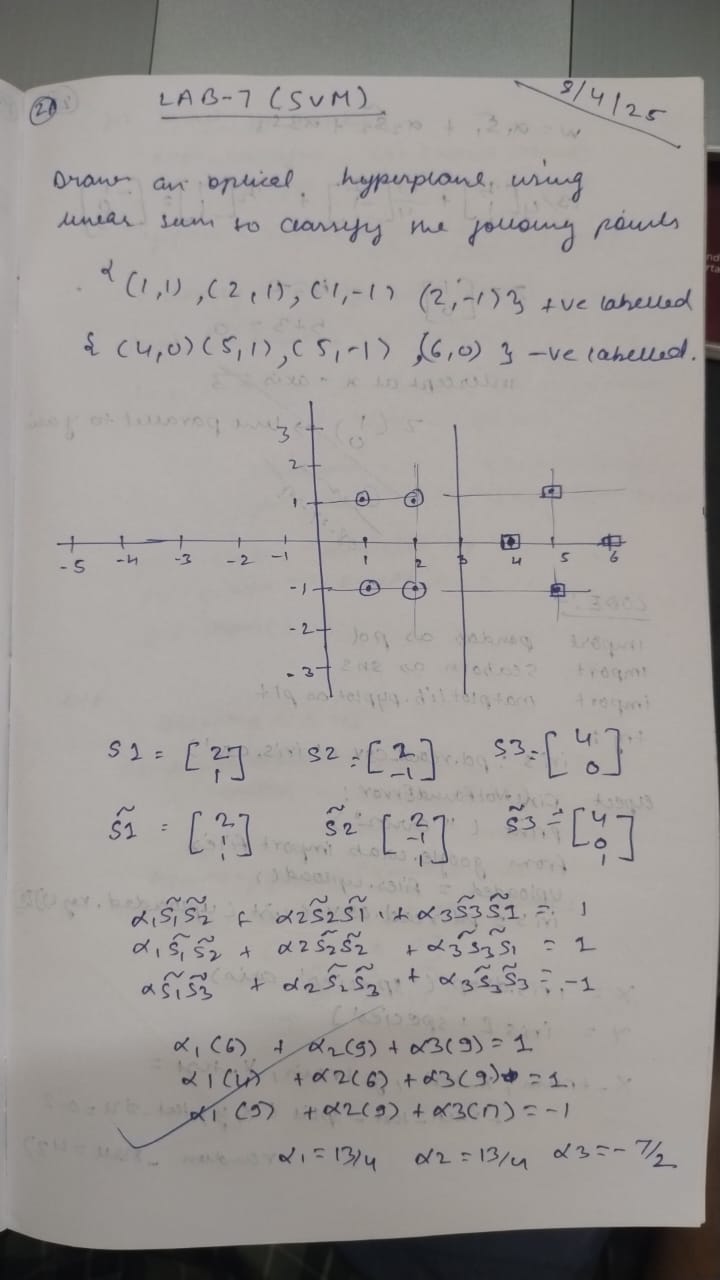
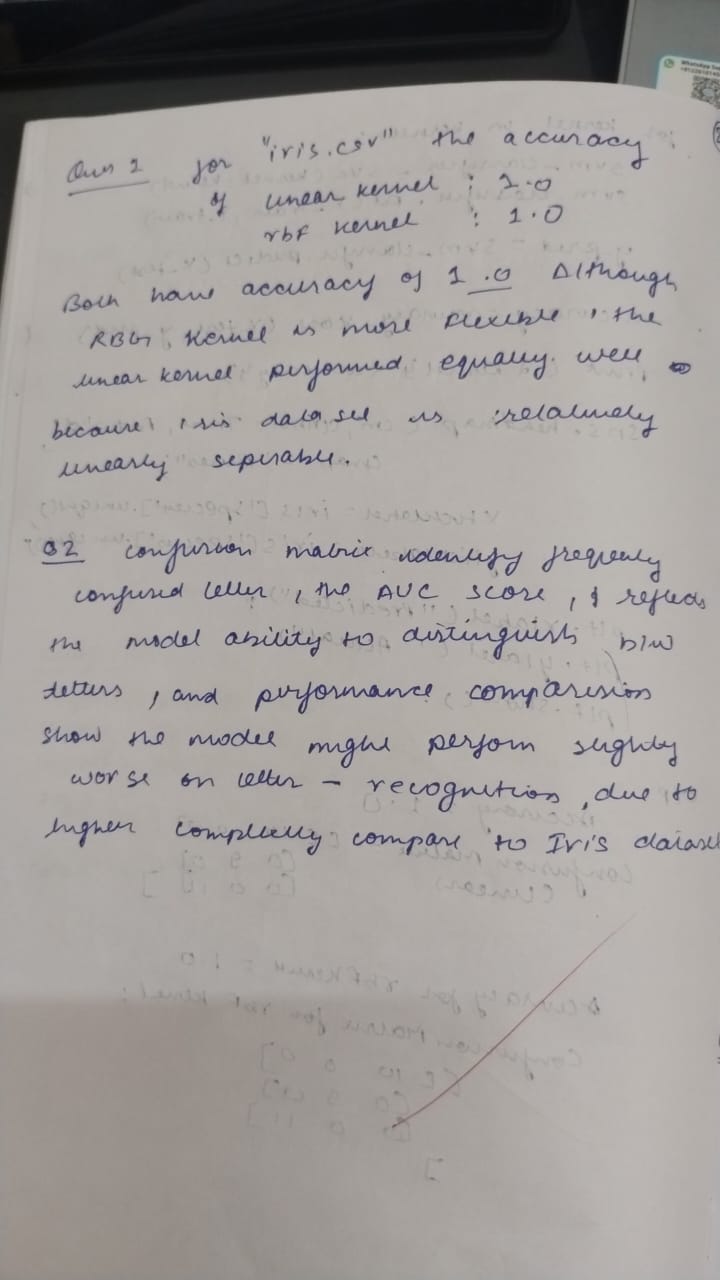
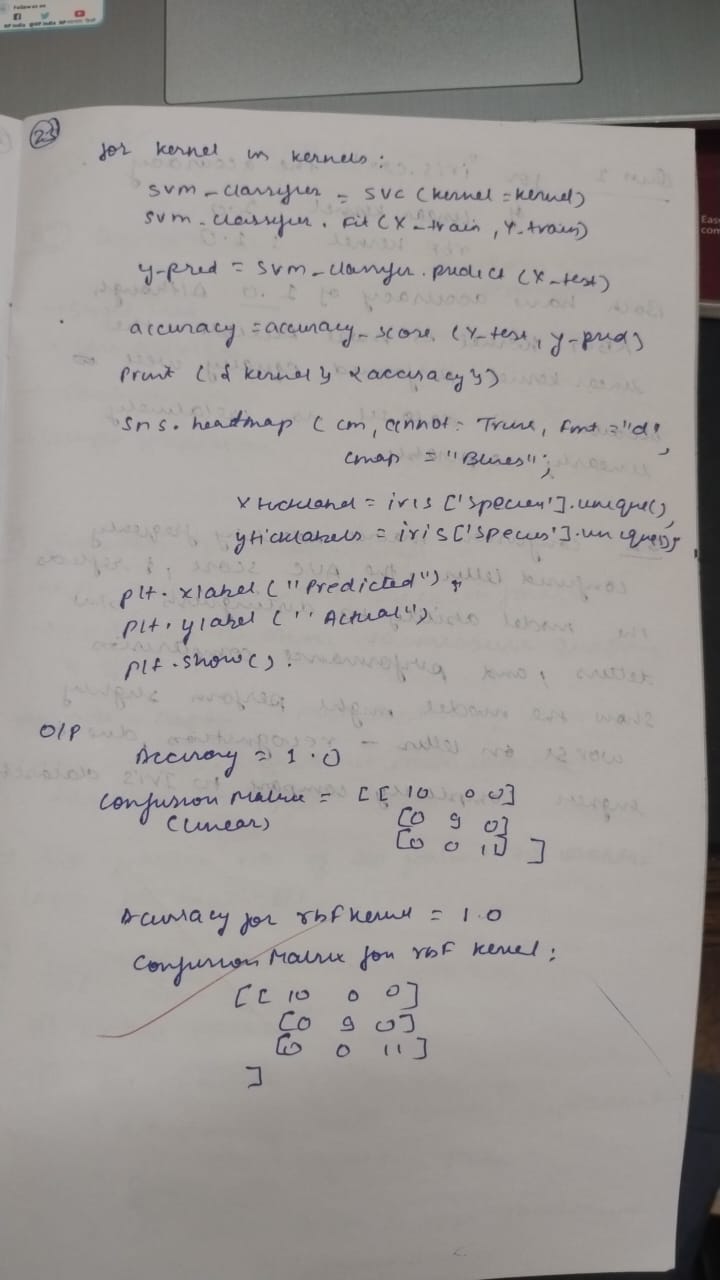
macro avg 0.92 0.92 0.92 61

weighted avg 0.92 0.92 0.92 61

In [ ]:

**Program 7**

Build Support vector machine model for a given dataset

Algorithm****Code:

In [23]:

*# Import necessary libraries*

import pandas as pd

from sklearn.svm import SVC

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

from sklearn.metrics import accuracy\_score, confusion\_matrix

import seaborn as sns

import matplotlib.pyplot as plt

*# URL of the Iris dataset (adjust this URL if needed)*

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

*# Column names as the dataset contains headers now*

columns = ['sepal\_length', 'sepal\_width', 'petal\_length', 'petal\_width', 'species']

*# Load the dataset from the URL into a pandas DataFrame*

iris = pd.read\_csv(url, header=None, names=columns)

*# Map the species names to numeric values (for classification)*

iris['species'] = iris['species'].map({'Iris-setosa': 0, 'Iris-versicolor': 1, 'Iris-virginica': 2})

*# Split features and target*

X = iris.drop('species', axis=1)

y = iris['species']

*# Split data into training and testing sets (80% train, 20% test)*

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

*# Initialize and train SVM with a linear kernel*

svm\_linear = SVC(kernel='linear', random\_state=42)

svm\_linear.fit(X\_train, y\_train)

*# Predict and calculate accuracy for the linear kernel model*

y\_pred\_linear = svm\_linear.predict(X\_test)

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

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

*# Initialize and train SVM with RBF kernel*

svm\_rbf = SVC(kernel='rbf', random\_state=42)

svm\_rbf.fit(X\_train, y\_train)

*# Predict and calculate accuracy for the RBF kernel model*

y\_pred\_rbf = svm\_rbf.predict(X\_test)

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

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

*# Display accuracy scores*

print(f"Accuracy of SVM with Linear Kernel: {accuracy\_linear:.2f}")

print(f"Accuracy of SVM with RBF Kernel: {accuracy\_rbf:.2f}")

*# Plot confusion matrices for both models*

fig, axes = plt.subplots(1, 2, figsize=(12, 5))

*# Linear Kernel Confusion Matrix*

sns.heatmap(conf\_matrix\_linear, annot=True, fmt="d", cmap="Blues", xticklabels=iris['species'].unique(),

yticklabels=iris['species'].unique(), ax=axes[0])

axes[0].set\_title('Confusion Matrix - Linear Kernel')

*# RBF Kernel Confusion Matrix*

sns.heatmap(conf\_matrix\_rbf, annot=True, fmt="d", cmap="Blues", xticklabels=iris['species'].unique(),

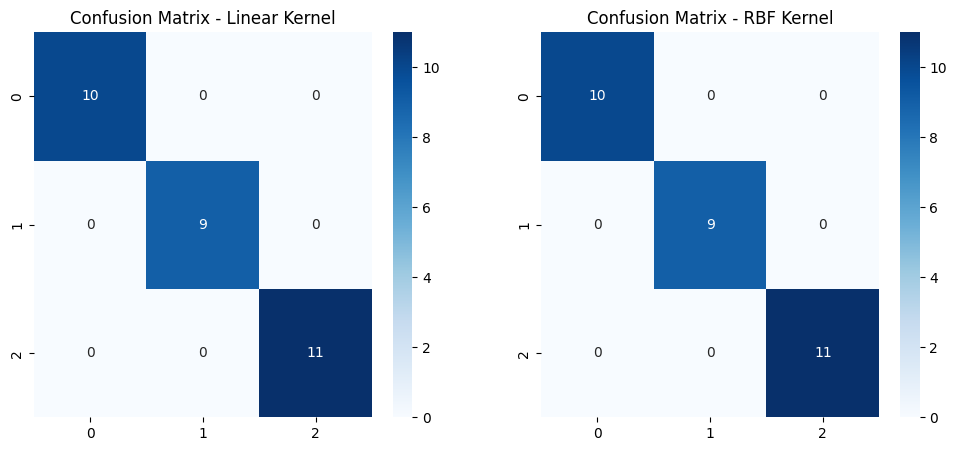
yticklabels=iris['species'].unique(), ax=axes[1])

axes[1].set\_title('Confusion Matrix - RBF Kernel')

plt.show()

Accuracy of SVM with Linear Kernel: 1.00

Accuracy of SVM with RBF Kernel: 1.00



In [21]:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

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

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score, confusion\_matrix, roc\_curve, auc

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import ConfusionMatrixDisplay

from sklearn.preprocessing import label\_binarize

*# Load the Letter-recognition dataset*

letter\_data = pd.read\_csv('letter-recognition.csv')

*# Prepare the dataset*

X\_letter = letter\_data.drop('letter', axis=1)

y\_letter = letter\_data['letter']

*# Encode the letter labels to numerical values*

le = LabelEncoder()

y\_letter\_encoded = le.fit\_transform(y\_letter)

*# Split the data into training and testing sets (80% training, 20% testing)*

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

*# Train the SVM model with a linear kernel*

svm\_linear = SVC(kernel='linear')

svm\_linear.fit(X\_train\_letter, y\_train\_letter)

*# Make predictions*

y\_pred\_letter = svm\_linear.predict(X\_test\_letter)

*# Calculate accuracy score*

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

print(f"Accuracy Score with Linear Kernel: {accuracy:.4f}")

*# Confusion Matrix*

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

*# Plot Confusion Matrix (without printing the elements)*

plt.figure(figsize=(10, 8)) *# Make the plot larger*

ConfusionMatrixDisplay(conf\_matrix, display\_labels=le.classes\_).plot(cmap='Blues')

plt.title('Confusion Matrix for Letter-recognition Dataset')

plt.show()

*# Binarize the labels for multi-class classification*

y\_test\_letter\_binarized = label\_binarize(y\_test\_letter, classes=np.arange(26)) *# 26 classes*

y\_pred\_prob\_letter = svm\_linear.decision\_function(X\_test\_letter)

*# Compute ROC curve and AUC score for each class*

fpr = {}

tpr = {}

roc\_auc = {}

for i in range(26): *# 26 classes*

fpr[i], tpr[i], \_ = roc\_curve(y\_test\_letter\_binarized[:, i], y\_pred\_prob\_letter[:, i])

roc\_auc[i] = auc(fpr[i], tpr[i])

*# Plot ROC curve for each class*

plt.figure(figsize=(10, 8))

for i in range(26):

plt.plot(fpr[i], tpr[i], lw=2, label=f'Class {i} (AUC = {roc\_auc[i]:.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic for Letter-recognition Dataset (Multi-class)')

plt.legend(loc="lower right")

plt.show()

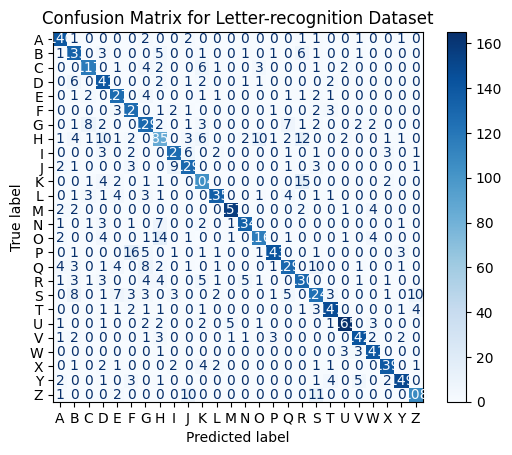
*# Display AUC scores for each class*

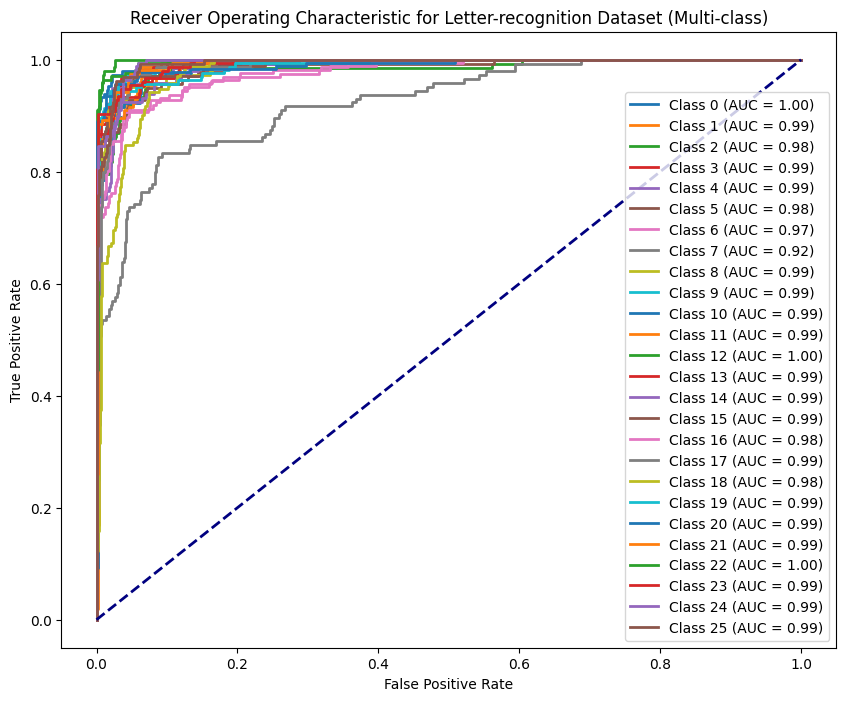
for i in range(26):

print(f"AUC score for Class {i}: {roc\_auc[i]:.4f}")

Accuracy Score with Linear Kernel: 0.8545

<Figure size 1000x800 with 0 Axes>





AUC score for Class 0: 0.9955

AUC score for Class 1: 0.9896

AUC score for Class 2: 0.9825

AUC score for Class 3: 0.9938

AUC score for Class 4: 0.9915

AUC score for Class 5: 0.9825

AUC score for Class 6: 0.9742

AUC score for Class 7: 0.9238

AUC score for Class 8: 0.9888

AUC score for Class 9: 0.9925

AUC score for Class 10: 0.9864

AUC score for Class 11: 0.9908

AUC score for Class 12: 0.9973

AUC score for Class 13: 0.9918

AUC score for Class 14: 0.9855

AUC score for Class 15: 0.9909

AUC score for Class 16: 0.9766

AUC score for Class 17: 0.9882

AUC score for Class 18: 0.9779

AUC score for Class 19: 0.9893

AUC score for Class 20: 0.9906

AUC score for Class 21: 0.9934

AUC score for Class 22: 0.9979

AUC score for Class 23: 0.9931

AUC score for Class 24: 0.9936

AUC score for Class 25: 0.9934

In [13]:

import pandas as pd

*# Data*

data = {

"Height": [44, 52.1, 57.1, 33, 27.8, 27.2, 32, 45.1, 56.7, 56.9, 122.1],

"Weight": [126.3, 136.9, 109.2, 148.3, 110.4, 107.8, 128.4, 120.2, 140.2, 139.2, 154.1]

}

*# Create a DataFrame*

df = pd.DataFrame(data)

*# Save to CSV*

df.to\_csv('height\_weight\_data.csv', index=False)

import pandas as pd

import matplotlib.pyplot as plt

*# Load the data from CSV*

df = pd.read\_csv('height\_weight\_data.csv')

*# Plotting the data*

plt.scatter(df['Height'], df['Weight'], color='blue', label='Data points')

plt.xlabel('Height')

plt.ylabel('Weight')

plt.title('Height vs Weight')

plt.grid(True)

plt.show()

from sklearn.svm import SVR

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

from sklearn.metrics import mean\_squared\_error

*# Define features and target*

X = df[['Height']] *# Feature: Height*

y = df['Weight'] *# Target: Weight*

*# Split 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 the model*

svr = SVR(kernel='linear')

*# Fit the model*

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

*# Predict on the test set*

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

*# Calculate accuracy using Mean Squared Error (since it's regression)*

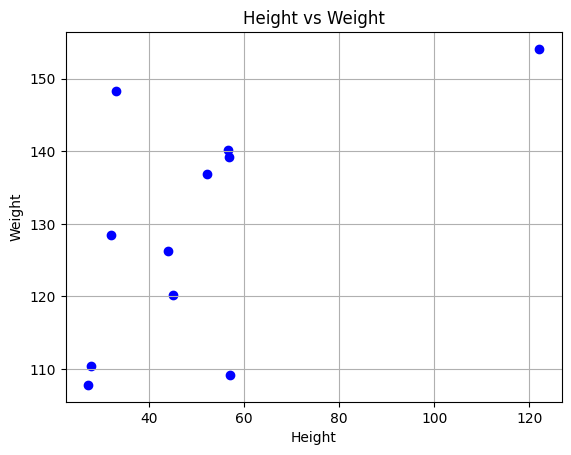
mse = mean\_squared\_error(y\_test, y\_pred)

rmse = mse \*\* 0.5 *# Root Mean Squared Error*

*# Print the support vectors and accuracy*

print(f"Support Vectors: {svr.support\_}")

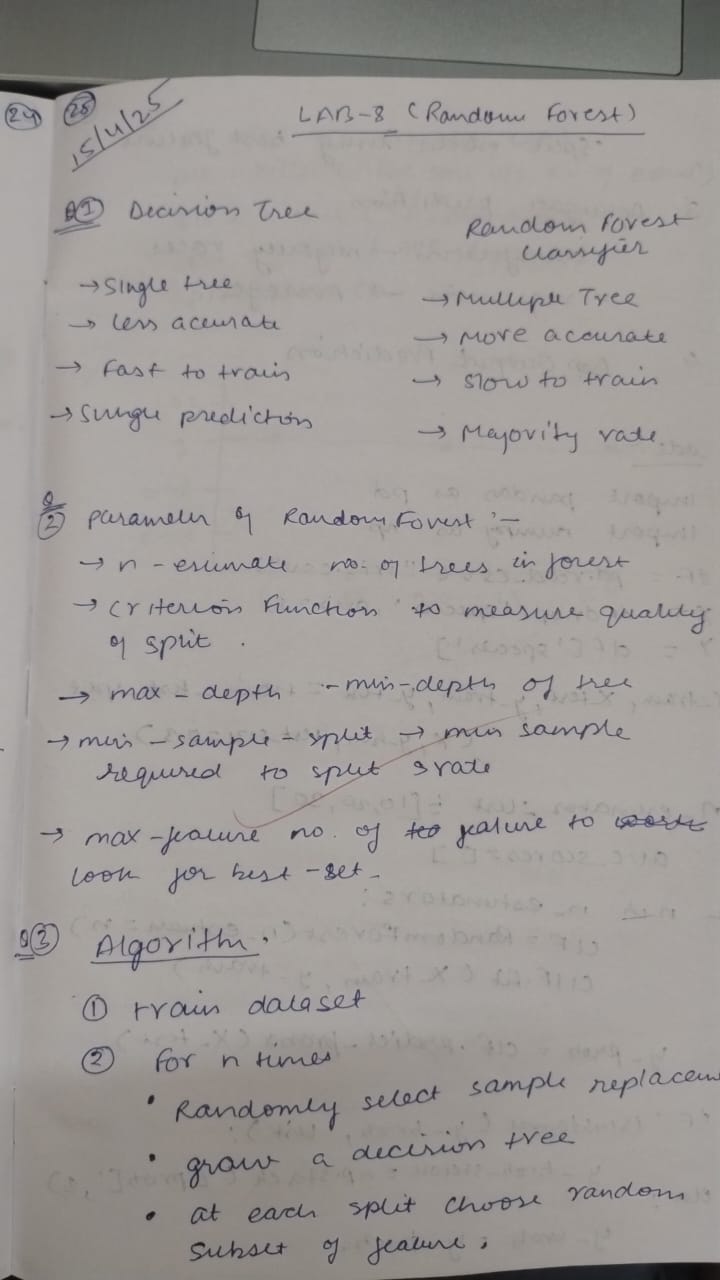
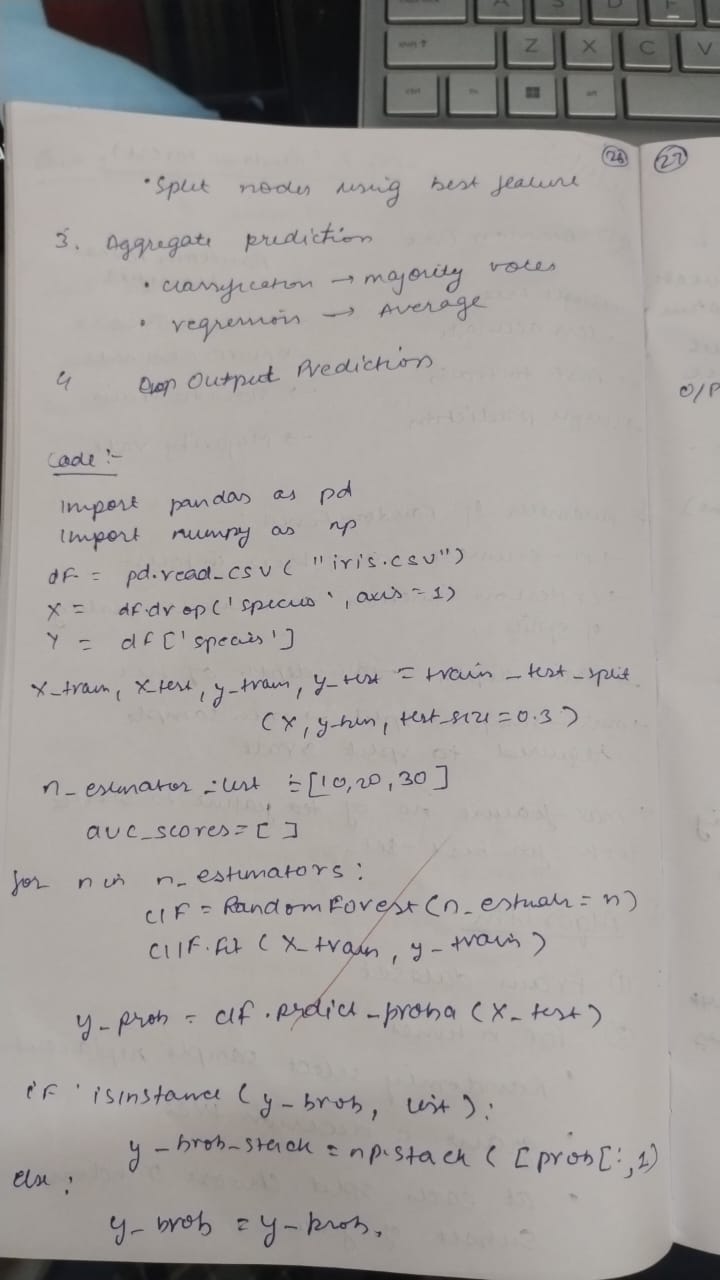
print(f"Root Mean Squared Error: {rmse}")

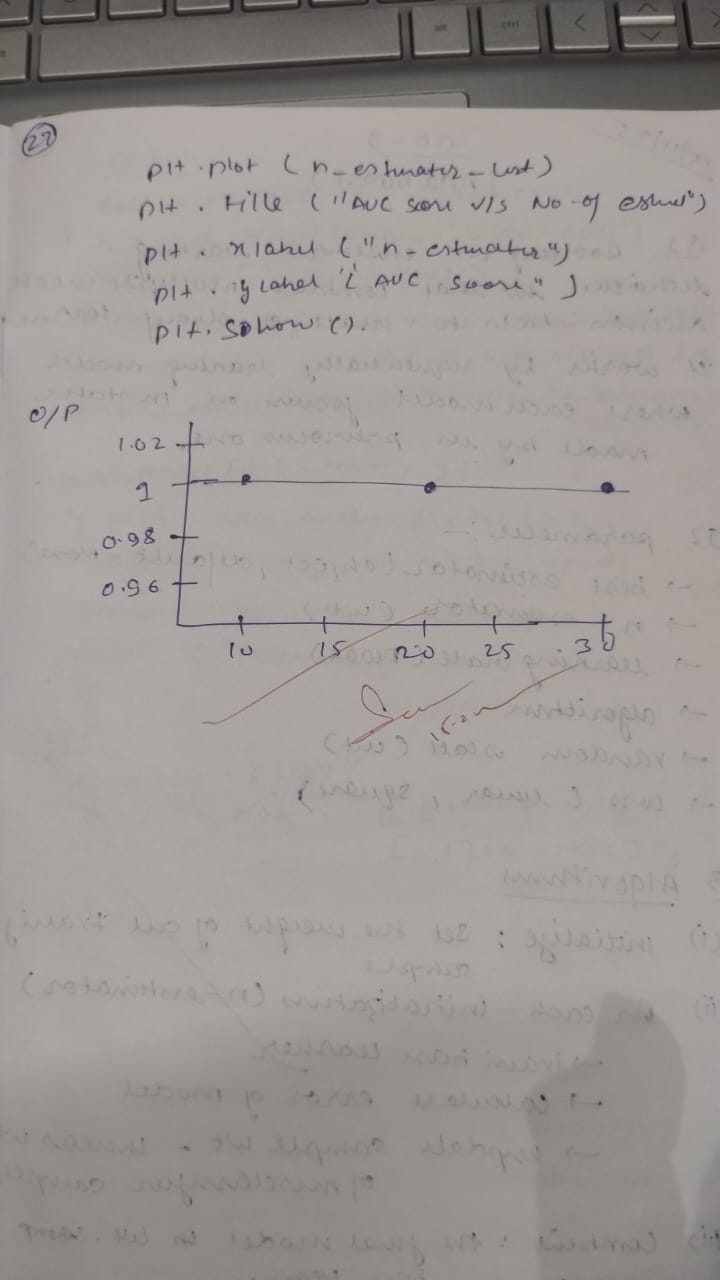


Support Vectors: [0 1 2 3 4 5 6 7]

Root Mean Squared Error: 11.680171349283054

**Program 8**

Implement Random forest ensemble method on a given dataset.  
****



Code:

import pandas as pd

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

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, confusion\_matrix

from sklearn.preprocessing import LabelEncoder

*# Load the dataset*

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

*# Preprocess the dataset*

*# Convert categorical columns into numeric*

*# For simplicity, we'll drop columns that are not necessary for modeling (e.g., 'Name', 'Ticket', 'Cabin')*

df = df.drop(columns=['Name', 'Ticket', 'Cabin'])

*# Convert categorical features into numeric values (Sex, Embarked)*

label\_encoder = LabelEncoder()

df['Sex'] = label\_encoder.fit\_transform(df['Sex'])

df['Embarked'] = df['Embarked'].map({'S': 0, 'C': 1, 'Q': 2}) *# 'S', 'C', 'Q' to numeric*

*# Handle missing values (impute with the median for simplicity)*

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

df['Embarked'].fillna(df['Embarked'].mode()[0], inplace=True) *# Mode is used for Embarked*

*# Features and target variable*

X = df.drop(columns=['Survived', 'PassengerId'])

y = df['Survived']

*# Split the data into training (80%) and testing (20%) sets*

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

*# Initialize the Random Forest Classifier*

rf\_classifier = RandomForestClassifier(n\_estimators=100, random\_state=42)

*# Train the model*

rf\_classifier.fit(X\_train, y\_train)

*# Predict on the test set*

y\_pred = rf\_classifier.predict(X\_test)

*# Calculate accuracy*

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

print(f'Accuracy: {accuracy:.4f}')

*# Display confusion matrix*

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

print('Confusion Matrix:')

print(conf\_matrix)

<ipython-input-11-ca7102c51cf1>:21: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.

The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.

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

<ipython-input-11-ca7102c51cf1>:22: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.

The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.

df['Embarked'].fillna(df['Embarked'].mode()[0], inplace=True) # Mode is used for Embarked

Accuracy: 0.8212

Confusion Matrix:

[[91 14]

[18 56]]

In [12]:

*# Import necessary libraries*

from sklearn.datasets import load\_iris

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

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score

import matplotlib.pyplot as plt

*# Load the Iris dataset*

iris = load\_iris()

X = iris.data

y = iris.target

*# Split the dataset into training and testing sets*

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

*# Initialize RandomForestClassifier with default n\_estimators=10*

clf = RandomForestClassifier(n\_estimators=10, random\_state=42)

*# Train the model*

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

*# Predict on the test set*

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

*# Evaluate the performance*

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

print(f"Accuracy with n\_estimators=10: {score\_default}")

*# Fine-tune the model by changing the value of n\_estimators*

n\_estimators\_values = [10, 50, 100, 200, 500]

scores = []

for n in n\_estimators\_values:

clf = RandomForestClassifier(n\_estimators=n, random\_state=42)

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

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

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

scores.append(score)

print(f"Accuracy with n\_estimators={n}: {score}")

*# Plot the results*

plt.plot(n\_estimators\_values, scores, marker='o')

plt.xlabel('Number of estimators')

plt.ylabel('Accuracy')

plt.title('Effect of n\_estimators on Random Forest Classifier Accuracy')

plt.grid(True)

plt.show()

Accuracy with n\_estimators=10: 1.0

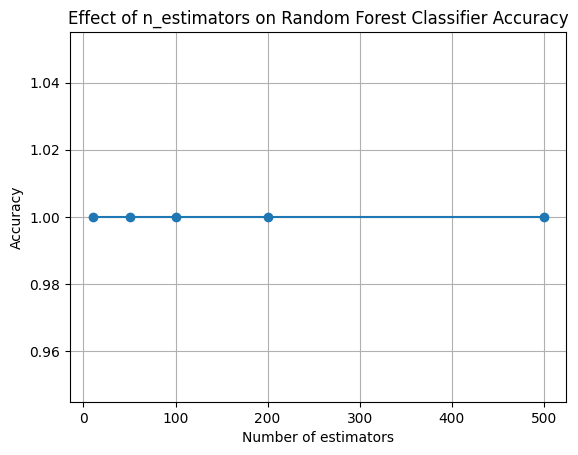
Accuracy with n\_estimators=10: 1.0

Accuracy with n\_estimators=50: 1.0

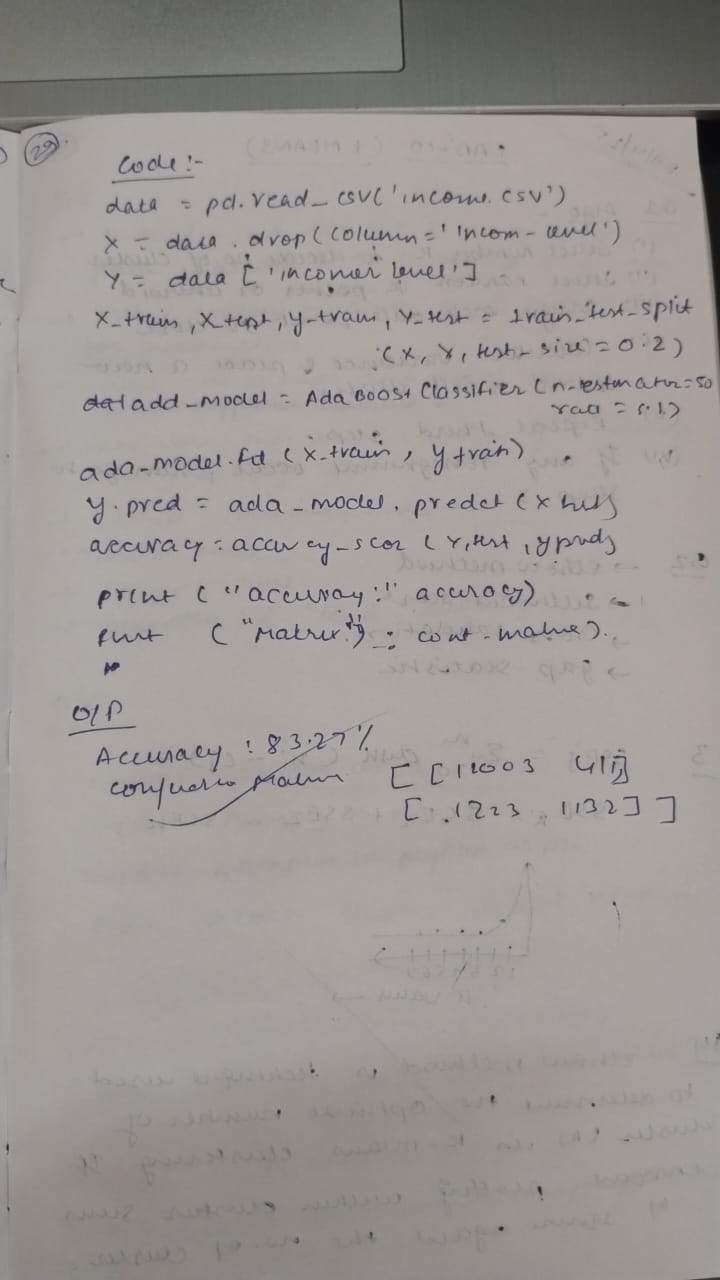
Accuracy with n\_estimators=100: 1.0

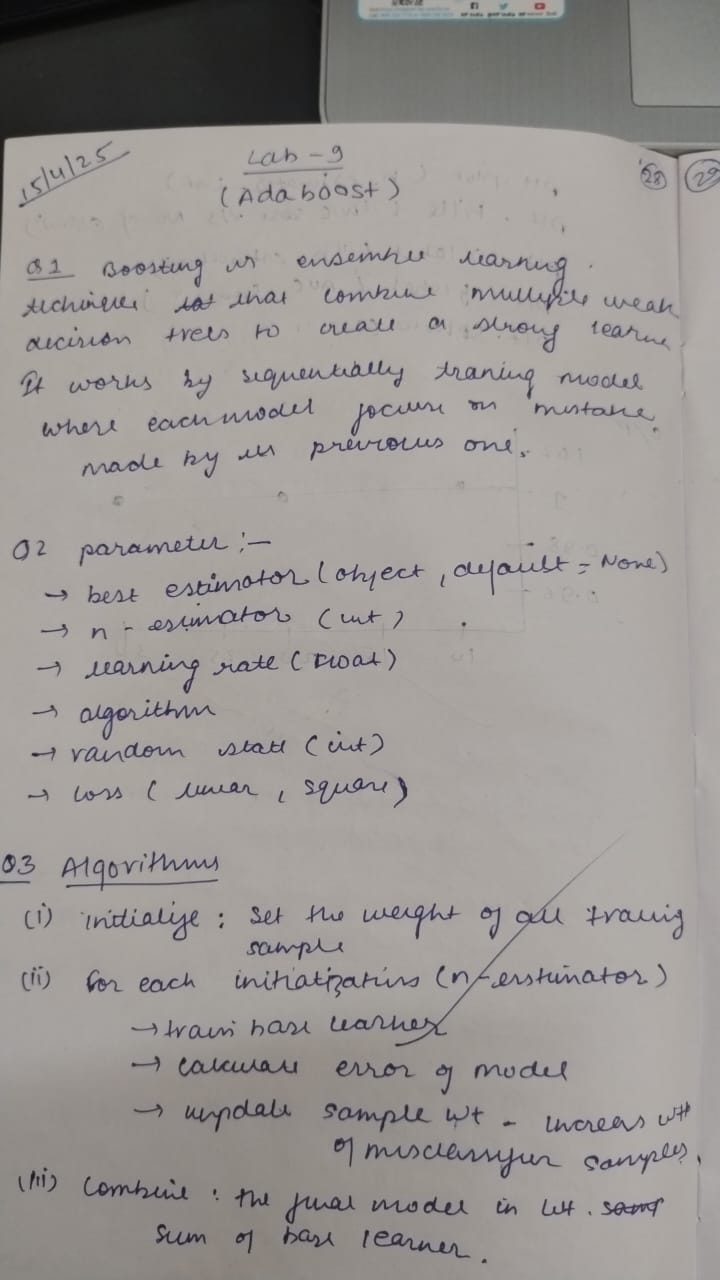
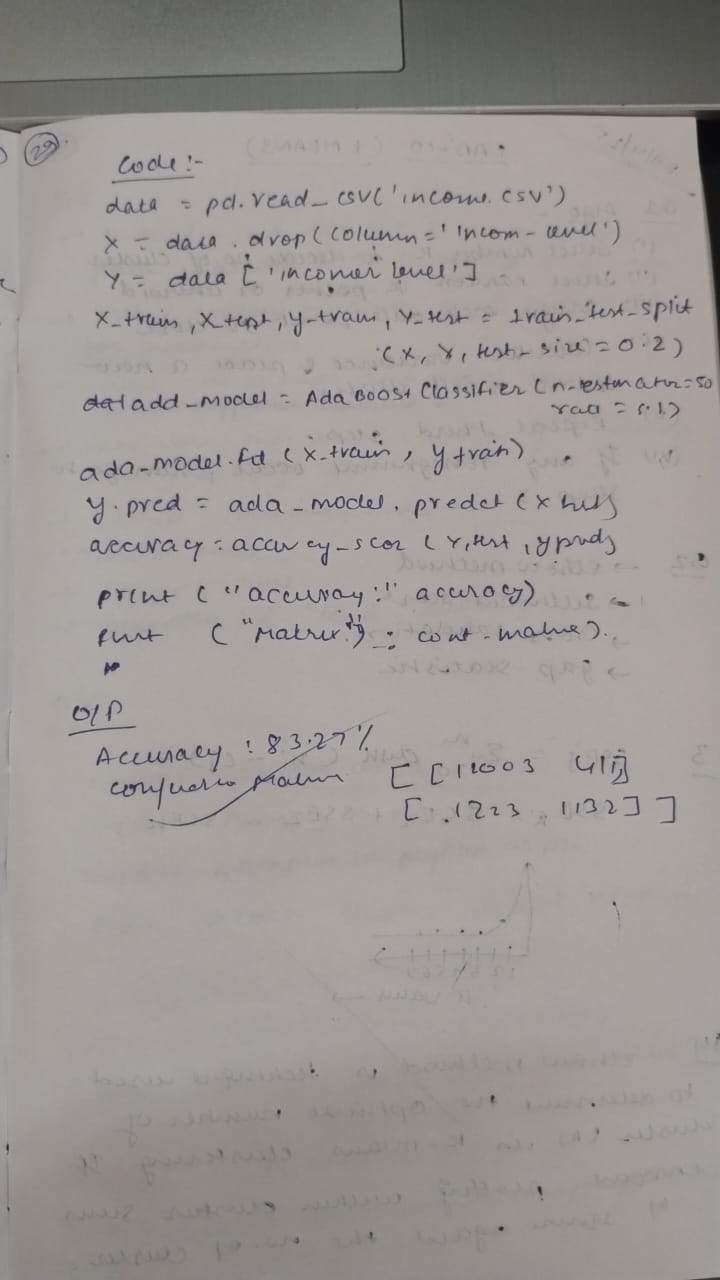
Accuracy with n\_estimators=200: 1.0

Accuracy with n\_estimators=500: 1.0



In [ ]:

**Program 9**Implement Boosting ensemble method on a given datasetAlgorithm  


****

Code

In [32]:

import pandas as pd

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

from sklearn.preprocessing import LabelEncoder

from sklearn.ensemble import AdaBoostClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, confusion\_matrix

from sklearn.ensemble import RandomForestClassifier

*# Load the data (assume "income.csv" is already downloaded)*

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

*# Let's check the columns to see what we're dealing with*

print(df.head())

*# Preprocessing: Encode categorical data if necessary (in this case 'income\_level' is categorical)*

encoder = LabelEncoder()

df['income\_level'] = encoder.fit\_transform(df['income\_level'])

*# Splitting the data into features (X) and target (y)*

X = df.drop('income\_level', axis=1)

y = df['income\_level']

*# Split the data 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)

age fnlwgt education\_num capital\_gain capital\_loss hours\_per\_week \

0 39 77516 13 2174 0 40

1 50 83311 13 0 0 13

2 38 215646 9 0 0 40

3 53 234721 7 0 0 40

4 28 338409 13 0 0 40

income\_level

0 0

1 0

2 0

3 0

4 0

In [33]:

*# Initialize the AdaBoost model with Random Forest as the base classifier*

ada\_boost = AdaBoostClassifier(RandomForestClassifier(n\_estimators=10, max\_depth=5, random\_state=42), n\_estimators=50, learning\_rate=1.0)

*# Train the AdaBoost model*

ada\_boost.fit(X\_train, y\_train)

*# Predictions*

y\_pred = ada\_boost.predict(X\_test)

*# Evaluate accuracy and confusion matrix*

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

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

print(f"Accuracy: {accuracy}")

print(f"Confusion Matrix:\n{conf\_matrix}")

Accuracy: 0.8368307912785341

Confusion Matrix:

[[7039 375]

[1219 1136]]

In [34]:

from sklearn.datasets import load\_iris

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

from sklearn.ensemble import AdaBoostClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score

*# Load the Iris dataset*

iris = load\_iris()

X = iris.data

y = iris.target

*# 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)

base\_classifier = RandomForestClassifier(n\_estimators=10, max\_depth=5, random\_state=42)

*# Initialize AdaBoost with Decision Tree as base classifier*

ada\_boost = AdaBoostClassifier(base\_classifier, n\_estimators=50)

*# Train the model*

ada\_boost.fit(X\_train, y\_train)

*# Evaluate accuracy on the test set*

y\_pred = ada\_boost.predict(X\_test)

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

print(f"Accuracy of AdaBoost on Iris dataset: {accuracy}")

Accuracy of AdaBoost on Iris dataset: 1.0

In [35]:

*# Varying n\_estimators and measuring the accuracy*

estimators = [10, 50, 100, 200, 500]

for n in estimators:

ada\_boost = AdaBoostClassifier(base\_classifier, n\_estimators=n)

ada\_boost.fit(X\_train, y\_train)

y\_pred = ada\_boost.predict(X\_test)

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

print(f"Accuracy with n\_estimators={n}: {accuracy}")

Accuracy with n\_estimators=10: 1.0

Accuracy with n\_estimators=50: 1.0

Accuracy with n\_estimators=100: 1.0

Accuracy with n\_estimators=200: 1.0

Accuracy with n\_estimators=500: 1.0

In [36]:

*# Varying n\_estimators and learning\_rate*

learning\_rates = [0.1, 0.5, 1.0, 1.5]

n\_estimators\_values = [50, 100, 200]

for lr in learning\_rates:

for n in n\_estimators\_values:

ada\_boost = AdaBoostClassifier(base\_classifier,

n\_estimators=n, learning\_rate=lr)

ada\_boost.fit(X\_train, y\_train)

y\_pred = ada\_boost.predict(X\_test)

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

print(f"Accuracy with n\_estimators={n} and learning\_rate={lr}: {accuracy}")

Accuracy with n\_estimators=50 and learning\_rate=0.1: 1.0

Accuracy with n\_estimators=100 and learning\_rate=0.1: 1.0

Accuracy with n\_estimators=200 and learning\_rate=0.1: 1.0

Accuracy with n\_estimators=50 and learning\_rate=0.5: 1.0

Accuracy with n\_estimators=100 and learning\_rate=0.5: 1.0

Accuracy with n\_estimators=200 and learning\_rate=0.5: 1.0

Accuracy with n\_estimators=50 and learning\_rate=1.0: 1.0

Accuracy with n\_estimators=100 and learning\_rate=1.0: 1.0

Accuracy with n\_estimators=200 and learning\_rate=1.0: 1.0

Accuracy with n\_estimators=50 and learning\_rate=1.5: 1.0

Accuracy with n\_estimators=100 and learning\_rate=1.5: 1.0

Accuracy with n\_estimators=200 and learning\_rate=1.5: 1.0

In [37]:

from sklearn.linear\_model import LogisticRegression

*# Using Logistic Regression as base estimator in AdaBoost*

ada\_boost\_lr = AdaBoostClassifier(LogisticRegression(max\_iter=1000), n\_estimators=50)

ada\_boost\_lr.fit(X\_train, y\_train)

y\_pred\_lr = ada\_boost\_lr.predict(X\_test)

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

print(f"Accuracy with Logistic Regression base estimator: {accuracy\_lr}")

Accuracy with Logistic Regression base estimator: 0.9333333333333333

In [38]:

*# Using Decision Tree as base estimator in AdaBoost*

ada\_boost\_tree = AdaBoostClassifier(DecisionTreeClassifier(max\_depth=1), n\_estimators=50)

ada\_boost\_tree.fit(X\_train, y\_train)

y\_pred\_tree = ada\_boost\_tree.predict(X\_test)

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

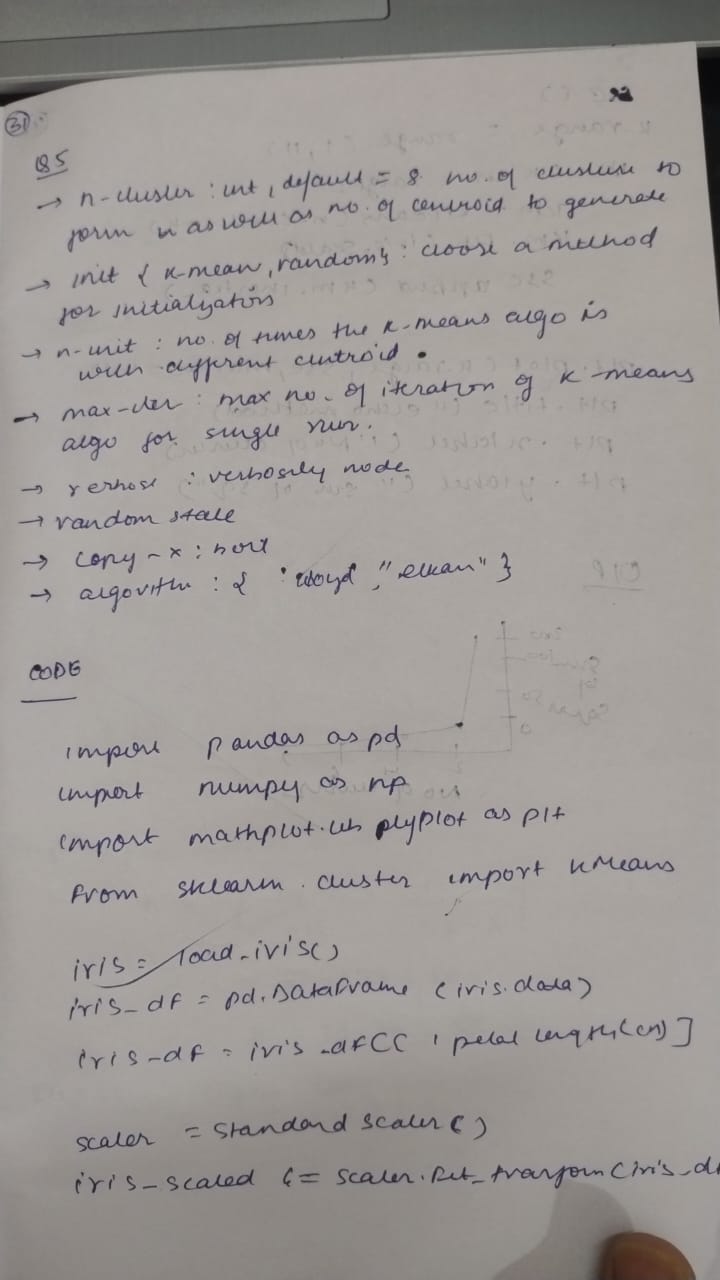
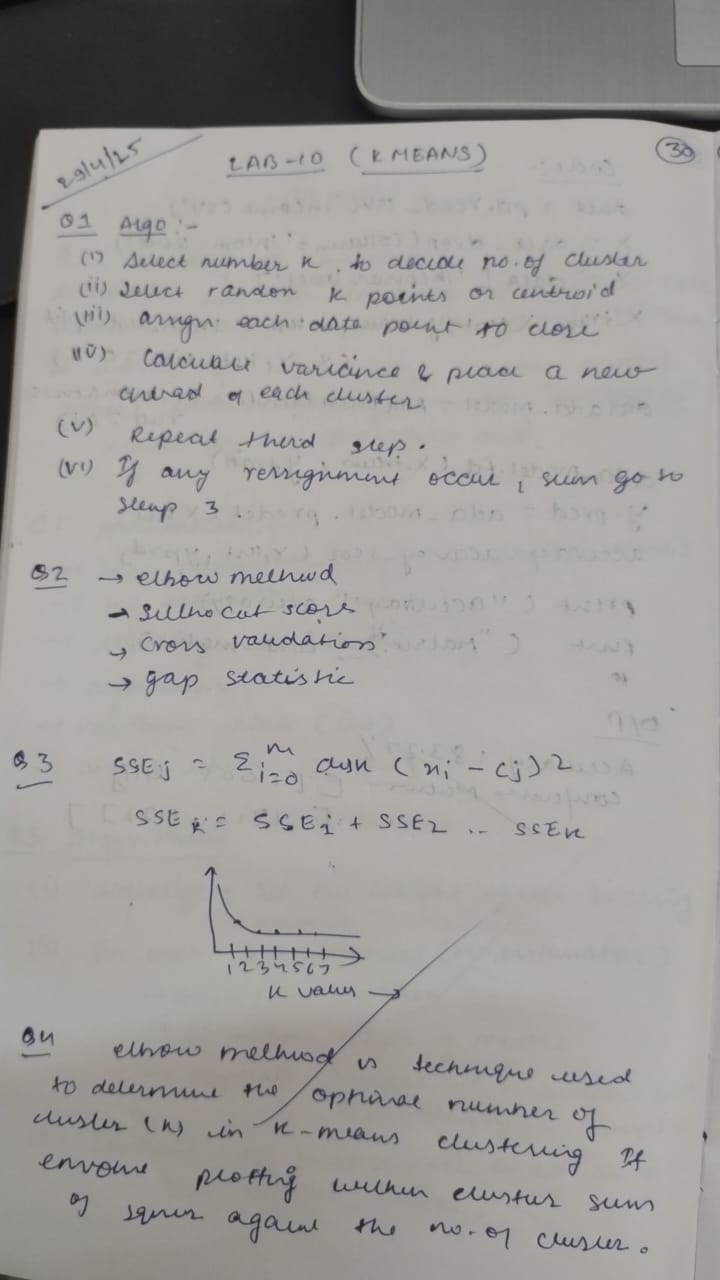
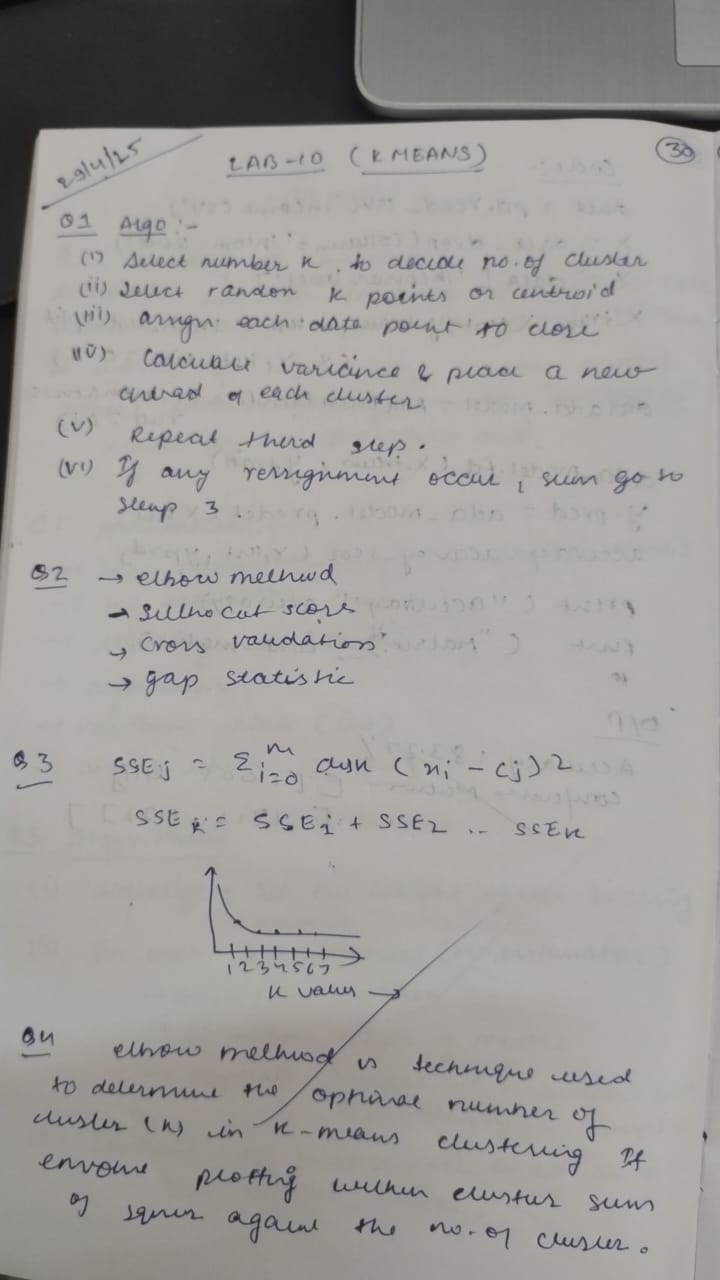
print(f"Accuracy with Decision Tree base estimator: {accuracy\_tree}")

Accuracy with Decision Tree base estimator: 0.9333333333333333

In [ ]:

**Program 10**

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

Algorithm  


Code

In [13]:

import pandas as pd

import numpy as np

from sklearn.cluster import KMeans

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

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy\_score

import matplotlib.pyplot as plt

import random

*# Step 1: Create synthetic income dataset*

np.random.seed(42)

names = [f'Person\_{i}' for i in range(1, 51)]

ages = np.random.randint(20, 60, size=50)

incomes = np.random.randint(20000, 120000, size=50)

income\_df = pd.DataFrame({

'Name': names,

'Age': ages,

'Income': incomes

})

*# Save to CSV (optional)*

income\_df.to\_csv("income.csv", index=False)

*# Step 2: Load and preprocess*

df = pd.read\_csv("income.csv")

*# Drop name (non-numeric)*

df\_numeric = df.drop('Name', axis=1)

*# Step 3: Scaling*

scaler = StandardScaler()

df\_scaled = scaler.fit\_transform(df\_numeric)

*# Step 4: Train-test split*

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

*# Step 5: SSE vs number of clusters (Elbow method)*

sse = []

k\_range = range(1, 11)

for k in k\_range:

km = KMeans(n\_clusters=k, random\_state=42)

km.fit(X\_train)

sse.append(km.inertia\_)

*# Step 6: Plot SSE*

plt.figure(figsize=(8, 5))

plt.plot(k\_range, sse, marker='o')

plt.title('SSE vs Number of Clusters')

plt.xlabel('Number of Clusters (k)')

plt.ylabel('Sum of Squared Errors (SSE)')

plt.grid(True)

plt.show()

*# Step 7: Fit KMeans with optimal k (e.g., 3)*

optimal\_k = 3

kmeans = KMeans(n\_clusters=optimal\_k, random\_state=42)

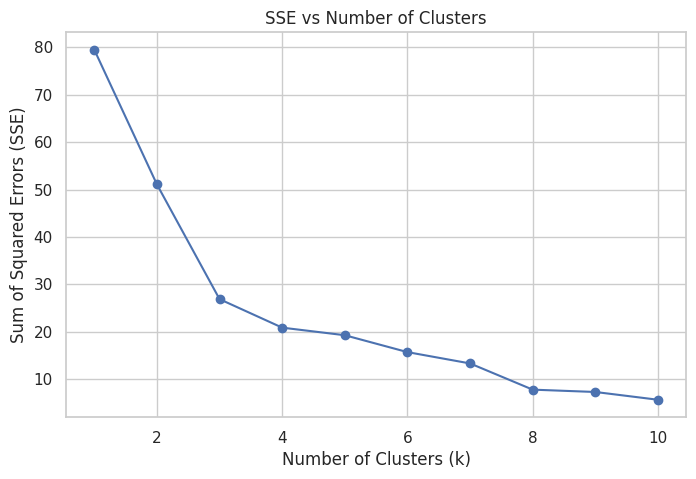
kmeans.fit(X\_train)

*# Step 8: Predict clusters on test set*

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

*# Step 9: "Accuracy" – not meaningful in unsupervised, but just a check*

print("Cluster labels for test data:", y\_pred)



Cluster labels for test data: [2 1 1 1 2 0 2 2 1 0]

In [14]:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

from sklearn.datasets import load\_iris

*# Step 1: Load iris dataset and use only petal length & width*

iris = load\_iris()

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

iris\_df = iris\_df[['petal length (cm)', 'petal width (cm)']] *# drop other features*

*# Step 2: Scaling*

scaler = StandardScaler()

iris\_scaled = scaler.fit\_transform(iris\_df)

*# Step 3: Elbow method to find optimal k*

sse = []

k\_range = range(1, 11)

for k in k\_range:

km = KMeans(n\_clusters=k, random\_state=42)

km.fit(iris\_scaled)

sse.append(km.inertia\_)

*# Step 4: Plot SSE vs k*

plt.figure(figsize=(8, 5))

plt.plot(k\_range, sse, marker='o')

plt.title("Elbow Method: SSE vs Number of Clusters")

plt.xlabel("Number of Clusters (k)")

plt.ylabel("Sum of Squared Errors (SSE)")

plt.grid(True)

plt.show()

*# Optional: Fit with optimal k = 3 and visualize clusters*

optimal\_k = 3

kmeans = KMeans(n\_clusters=optimal\_k, random\_state=42)

labels = kmeans.fit\_predict(iris\_scaled)

plt.figure(figsize=(8, 5))

plt.scatter(iris\_scaled[:, 0], iris\_scaled[:, 1], c=labels, cmap='viridis', s=50)

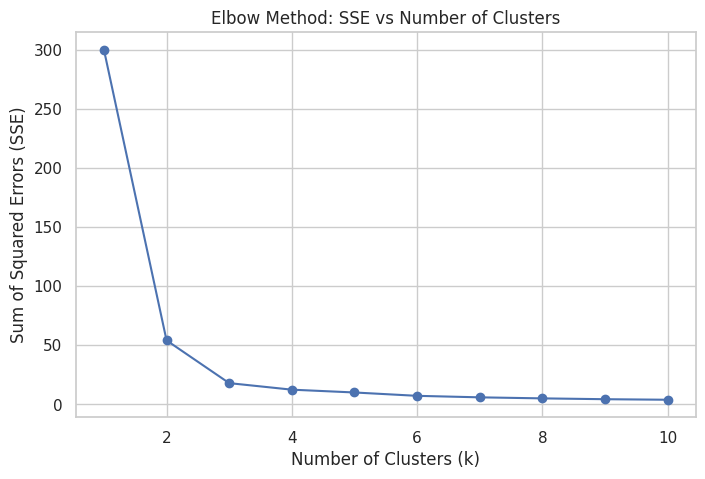
plt.title("KMeans Clustering (k=3) on Iris Petal Features")

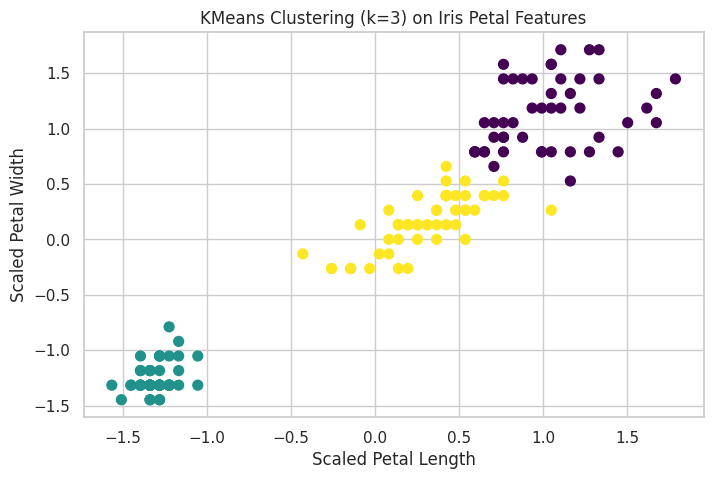
plt.xlabel("Scaled Petal Length")

plt.ylabel("Scaled Petal Width")

plt.grid(True)

plt.show()

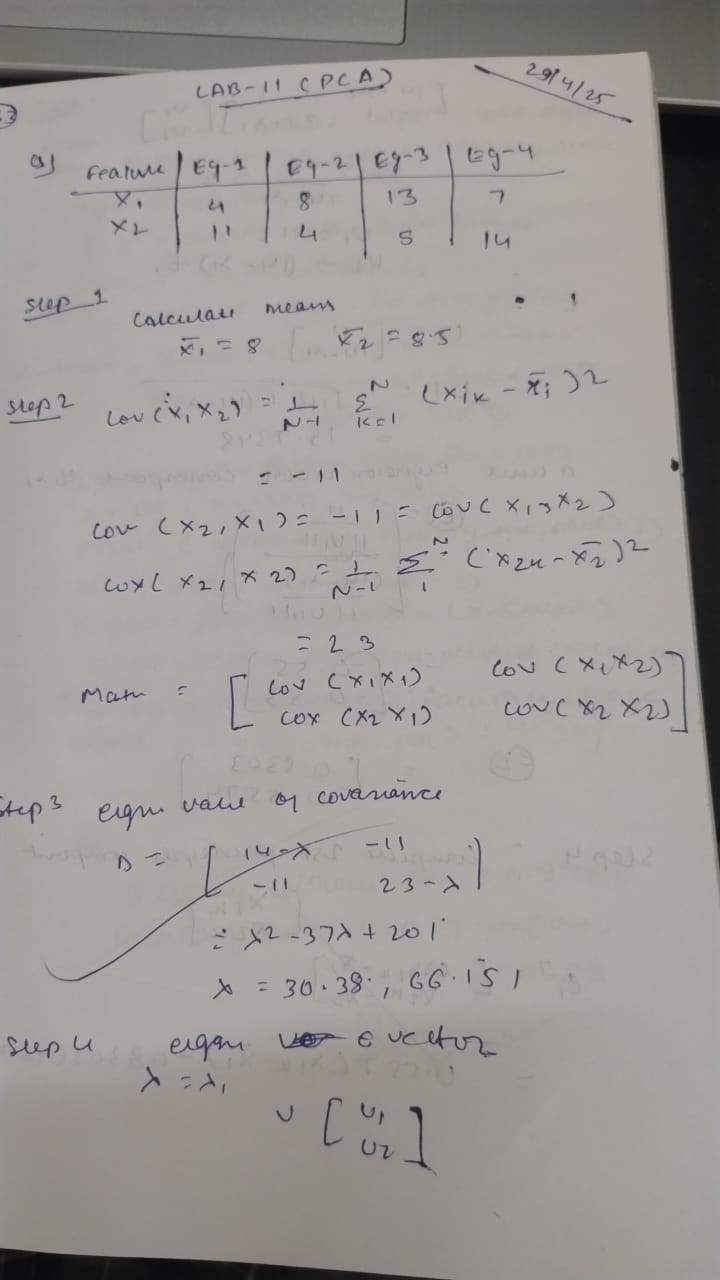


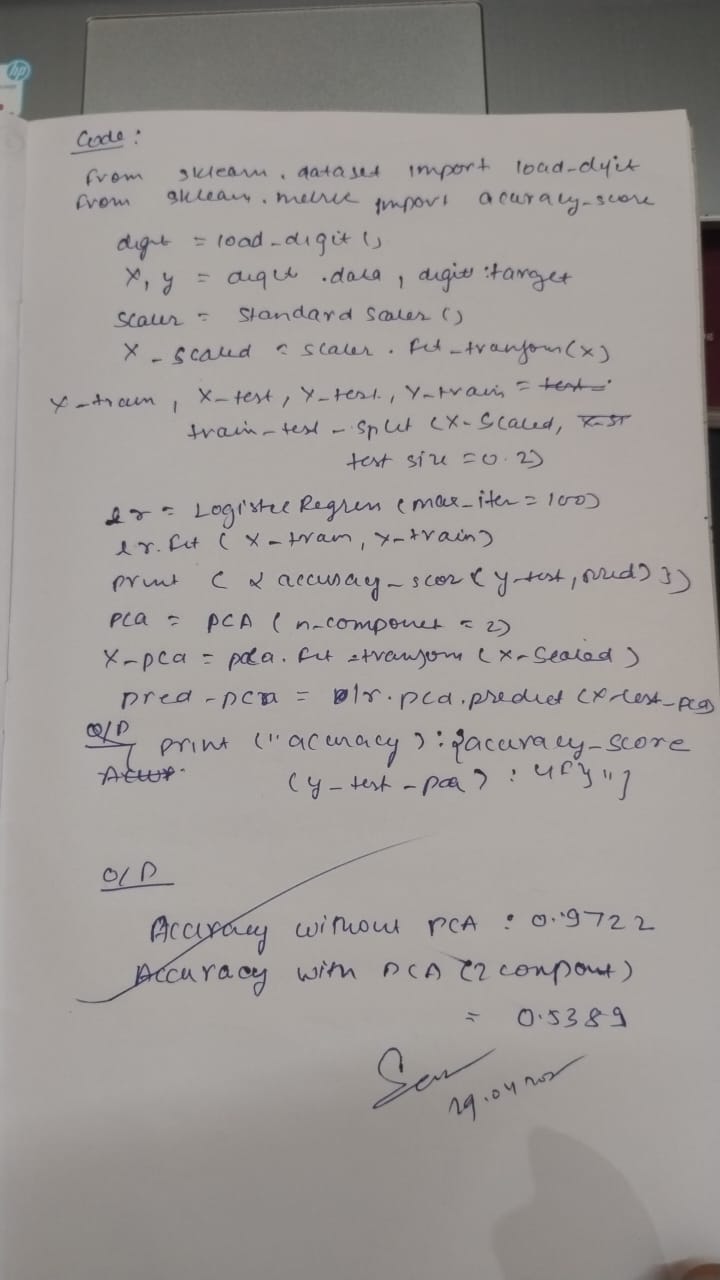


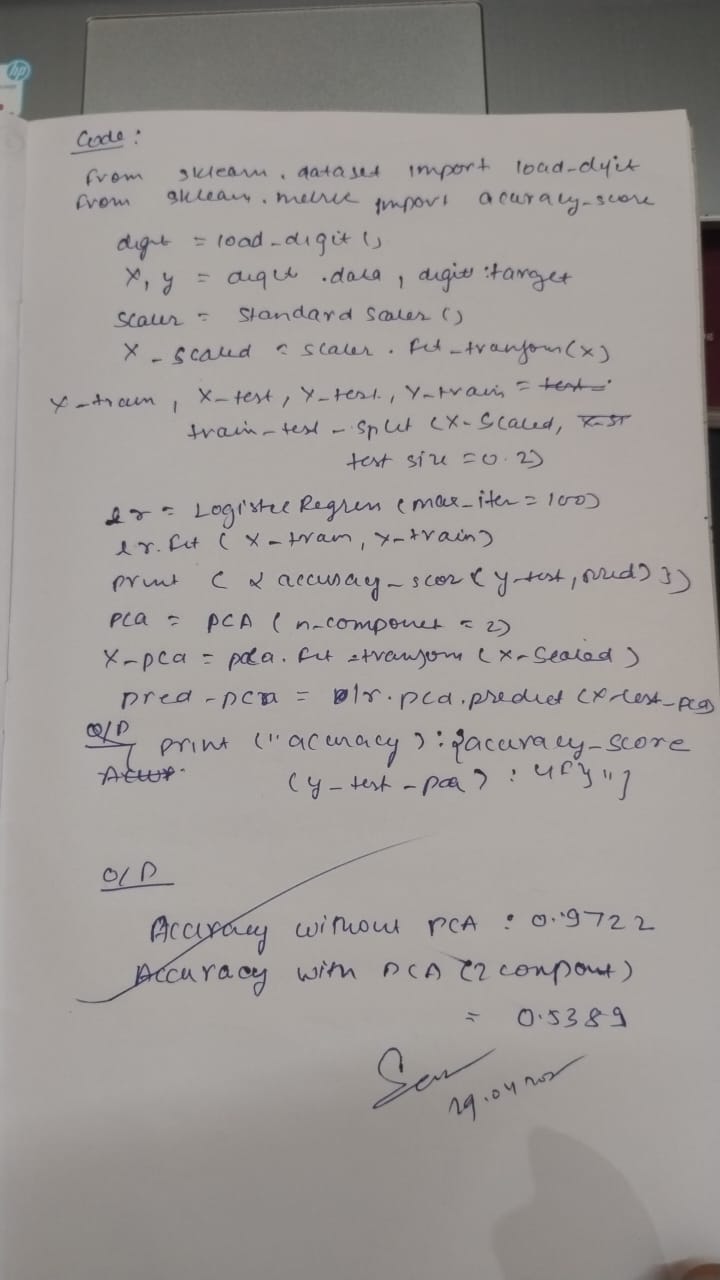
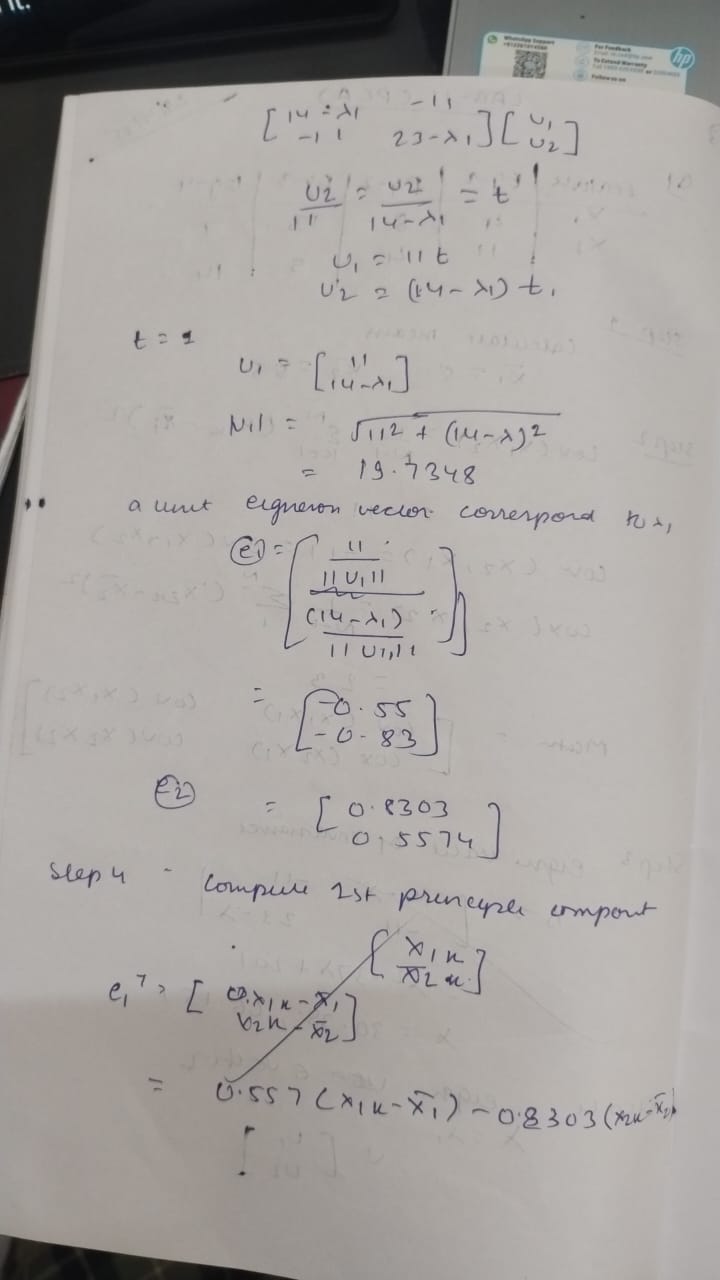
In [ ]:

**Program 11**

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







Code

In [12]:

from sklearn.datasets import load\_digits

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

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LogisticRegression

from sklearn.decomposition import PCA

from sklearn.metrics import accuracy\_score

*# Load dataset*

digits = load\_digits()

X, y = digits.data, digits.target

*# Scaling*

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)

*# Logistic Regression without PCA*

lr = LogisticRegression(max\_iter=1000)

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

preds = lr.predict(X\_test)

print(f"Accuracy without PCA: {accuracy\_score(y\_test, preds):.4f}")

*# Apply PCA with 2 components*

pca = PCA(n\_components=2)

X\_pca = pca.fit\_transform(X\_scaled)

*# Split PCA-reduced data*

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)

*# Logistic Regression with PCA-reduced data*

lr\_pca = LogisticRegression(max\_iter=1000)

lr\_pca.fit(X\_train\_pca, y\_train\_pca)

preds\_pca = lr\_pca.predict(X\_test\_pca)

print(f"Accuracy with PCA (2 components): {accuracy\_score(y\_test\_pca, preds\_pca):.4f}")

Accuracy without PCA: 0.9722

Accuracy with PCA (2 components): 0.5389

In [13]:

import pandas as pd

import numpy as np

from sklearn.preprocessing import StandardScaler

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

from sklearn.ensemble import RandomForestClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score

from sklearn.decomposition import PCA

from scipy.stats import zscore

*# Load dataset*

heart\_df = pd.read\_csv("heart.csv")

*# Print column names to verify*

print("Column names in dataset:", heart\_df.columns.tolist())

*# Optional: rename columns if necessary (based on your dataset)*

*# Example only – skip this if your columns are already clean*

heart\_df.columns = [col.strip().lower() for col in heart\_df.columns]

*# Reprint cleaned column names*

print("Cleaned column names:", heart\_df.columns.tolist())

*# Remove outliers using Z-score*

z\_scores = np.abs(zscore(heart\_df.select\_dtypes(include=[np.number])))

heart\_df = heart\_df[(z\_scores < 3).all(axis=1)]

*# Determine categorical columns that need encoding*

categorical\_cols = heart\_df.select\_dtypes(include=['object', 'category']).columns.tolist()

*# Encode categorical columns using one-hot encoding*

if categorical\_cols:

heart\_df = pd.get\_dummies(heart\_df, columns=categorical\_cols, drop\_first=True)

*# Split features and target*

target\_col = 'target' if 'target' in heart\_df.columns else heart\_df.columns[-1]

X = heart\_df.drop(target\_col, axis=1)

y = heart\_df[target\_col]

*# Feature scaling*

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()

}

print("\nModel Accuracy WITHOUT PCA:")

for name, model in models.items():

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

preds = model.predict(X\_test)

print(f"{name}: {accuracy\_score(y\_test, preds):.4f}")

*# Apply PCA*

pca = PCA(n\_components=5)

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)

print("\nModel Accuracy WITH PCA:")

for name, model in models.items():

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

preds = model.predict(X\_test\_pca)

print(f"{name}: {accuracy\_score(y\_test\_pca, preds):.4f}")

Column names in dataset: ['Age', 'Sex', 'ChestPainType', 'RestingBP', 'Cholesterol', 'FastingBS', 'RestingECG', 'MaxHR', 'ExerciseAngina', 'Oldpeak', 'ST\_Slope', 'HeartDisease']

Cleaned column names: ['age', 'sex', 'chestpaintype', 'restingbp', 'cholesterol', 'fastingbs', 'restingecg', 'maxhr', 'exerciseangina', 'oldpeak', 'st\_slope', 'heartdisease']

Model Accuracy WITHOUT PCA:

Logistic Regression: 0.9444

SVM: 0.9333

Random Forest: 0.9222

Model Accuracy WITH PCA:

Logistic Regression: 0.8611

SVM: 0.8778

Random Forest: 0.8833

In [ ]: