INDEX

EX. NO.	DATE	TITLE	PAGE NO.	MARKS	SIGN.
1		BASIC PYTHON PROGRAMS			
2		PYTHON BASIC LIBRARIES INTRODUCTION A. DATA ACQUISTION & DESCRIPTION: PANDAS B. DATA HANDLING & COMPUTATION: NUMPY C. DATA VISUALIZATION: MATPLOTLIB			
3		DATA PREPROCESSING A. FIVE NUMBER SUMMARY B. MISSING VALUE IMPUTATION C. COVARIANCE & CORRELATION			
4		LINEAR REGRESSION A. SIMPLE LINEAR REGRESSION MULTIPLE LINEAR REGRESSION			
5		MCCULLOCH PITTS NEURON			
6		HEBB NET			
7		SINGLE LAYER PERCEPTRON			
8		MULTI LAYER PERCEPTRON – FOREST FIRE PREDICTION			
9		NAIVE BAYES CLASSIFICATION - CUSTOMER BUYING HABIT PREDICTION			
10		DECISION TREE – HEART DISEASE CLASSIFICATION			
11		TOOL DEMONSTRATION – KNIME			
12		K MEANS CLUSTERING – HEART DISEASE CATEGORIZATION			
13		GENETIC ALGORITHM FOR OPTIMIZATION			
14		TOOL DEMONSTRATION – ORANGE			
15		K- NEAREST NEIGHBOR A. CLASSIFICATION B. MISSING VALUE IMPUTATION			
16		PRINCIPAL COMPONENT ANALYSIS (PCA)			
17		RADIAL BASIS FUNCTION (RBF)			
18		SELF ORGANIZING MAP (SOM)			
19		MINI PROJECT – DIET RECOMMENDATION SYSTEM			

BASIC PYTHON PROGRAMS

Ex.No.:

Date:

AIM

To implement and learn basic python program

1. **Write a python program that illustrates the usage of various operators**

```
a=5
b=15
print(a+b)
print(a-b)
print(a*b)
print(a/b)
print(a**b)
print(a//b)
print(~a)
print(++a)
print(a%b)
```

OUTPUT

```
20
-10
75
0.3333333333333333
30517578125
0
-6
5
5
```

2. **Control Statements**

```
age=15
if(age>18):
   print("eligible")
elif(age<18):
   print("not eligible")
else:
   print("invalid")

n=10
```

```
for i in range(n):
  print(i)

n=4
while n>0:
   print(f"n is {n}")
   n=n-1
```

OUTPUT

```
not eligible
0
1
2
3
n is 4
n is 3
n is 2
n is 1
```

3. **Write a python code to create a list of names of employees and display first item and last item of the list. Print length and type of the list.**

```
emp=["monday","tuesday","wednesday","thursday","friday"]
print(emp[0])
print(emp[-1])
print(len(emp))
print(type(emp))
print(emp[0:4])
```

OUTPUT

```
monday
friday
5
<class 'list'>
['monday', 'tuesday', 'wednesday', 'thursday']
```

4. **Give an example to display the difference between list, dictionary and tuple.**

```
list=["john",4]
set={1,2,3}
tup=("john",40)
dict={"name":"john","age":40}
print(list)
print(type(list))
```

```
print(set)
print(type(set))
print(tup)
print(type(tup))
print(dict)
print(type(dict))
```

OUTPUT

```
['john', 4]
<class 'list'>
{1, 2, 3}
<class 'set'>
('john', 40)
<class 'tuple'>
{'name': 'john', 'age': 40}
<class 'dict'>
```

5. **Create a multi-dimensional list and display list elements**

```
matrix=[
   [1,2],
   [3,5]
]
for i in matrix:
   for el in i:
      print(el,end=" ")
   print()
```

OUTPUT

```
1 2
3 5
```

6. **Find face value, position, place value**

```
a=int(input("Enter a number "))
b=int(input("Enter digit "))
temp=a
i=1
t=0
print(f"face value {b}")
while(a>0):
   t2=a%10
   if(t2==b):
      print(f"place value {t2*i}")
   i*=10
   t+=1
   a//=10
```

```
print(f"place is {t}")
```

OUTPUT

```
Enter a number 234
Enter digit 2
face value 2
place value 200
place is 3
```

7. **Pattern**

```
n=int(input("Enter a number "))
def pat(n):
    temp=n
    t1=0
    while temp>0:
        for i in range(t1):
            print(' ',end=' ')
        for i in range(temp):
            print('*',end=' ')
        print()
        temp-=2
        t1+=1
    for i in range(t1):
        print(' ',end=' ')
    print('*')
pat(n)
```

OUTPUT

```
Enter a number 6
* * * * * *
  * * * *
    * *
      *
```

8. **Pattern**

```
n=int(input("Enter a number "))
temp=(n*(n-1))-1
#print(temp)
while temp>0:
    if temp%2!=0:
        for i in range(n-1):
            print(' ',end=' ')
        for i in range(n-1):
```

```
            print('*',end=' ')
        for i in range(n-1):
            print(' ',end=' ')
        print()
    else:
        for i in range(n-1):
            print('*',end=' ')
        for i in range(n-1):
            print('*',end=' ')
        for i in range(n-1):
            print('*',end=' ')
        print()
    temp-=1
```

OUTPUT

```
Enter a number 3
    * *
* * * * * *
    * *
* * * * * *
    * *
```

9. List methods

```
a=[1,2,3,2]
print(f"{a} - {type(a)}")
a.append(4)
b=a.copy()
print(b)
a.clear()
print(a)
print(b.count(2))
a.extend(b)
print(a.index(2))
a.insert(1,8)
print(a)
a.pop()
print(a)
a.remove(2)
print(a)
a.reverse()
print(a)
a.sort()
print(a)
```

OUTPUT

```
[1, 2, 3, 2] - <class 'list'>
[1, 2, 3, 2, 4]
[]
2
1
[1, 8, 2, 3, 2, 4]
[1, 8, 2, 3, 2]
[1, 8, 3, 2]
[2, 3, 8, 1]
[1, 2, 3, 8]
```

10. **Write a python program that prints the number of ways a robot can take steps to climb n staircase. Assume that the robot can take 1 or 2 steps a time using function.**
n=int(input("Enter number of steps "))

```
def func(n):
    if (n==0 or n==1):
        return 1
    prev=1
    cur=1
    for _ in range(2,n+1):
        prev=cur
        cur=prev+cur
    return cur
```

OUTPUT

```
Enter number of steps  3
The robot can climb 3 steps in 3 ways.
```

RESULT

Thus, the above program was executed and implemented successfully.

PYTHON BASIC LIBRARIES INTRODUCTION

Ex.No.:02

Date:

A. DATA ACQUISITION AND DESCRIPTION : PANDAS

1. To read the content in excel file to the data frame

```
df=pd.read_excel('Book1.xlsx')
```

2. To read the content in CSV file to the data frame

```
df2=pd.read_csv('fileCSV.csv')
```

3. The code to print first five rows in the dataframe

```
print(df.head())
print(df2.head())
```

OUTPUT

```
   NO.           Name  Levels
0    1  Sarah Johnson       1
1    2  Michael Brown       3
2    3    Emma Wilson       1
3    4      James Lee       3
4    5  Olivia Harris       2
```

4. The code to print last five rows in the dataframe

```
print(df.tail())
print(df2.tail())
```

OUTPUT

```
    NO.            Name  Levels
45   46    Jacob Carter       1
46   47   Ruby Mitchell       3
47   48    Thomas Davis       1
48   49  Stella Jackson       3
49   50             Law       1
```

5. The code to print first three rows in the dataframe

```
print(df.head(3))
print(df2.head(3))
```

OUTPUT

```
   NO.            Name  Levels
0    1  Sarah Johnson       1
1    2  Michael Brown       3
2    3    Emma Wilson       1
```

6. The code to print last three rows in the dataframe

```
print(df.tail(3))
print(df2.tail(3))
```

OUTPUT

```
     NO.             Name  Levels
47    48    Thomas Davis       1
48    49  Stella Jackson       3
49    50             Law       1
```

7. The method to print the summary of data and what attributes will be displayed

```
print(df.info())
print()
print(df2.info())
print()
print()
#Attributes printed using describe():
print(df.describe())
print()
print(df2.describe())
```

OUTPUT

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   NO.     50 non-null     int64
 1   Name    50 non-null     object
 2   Levels  50 non-null     int64
dtypes: int64(2), object(1)
memory usage: 1.3+ KB
None
```

```
              NO.     Levels
count   50.00000  50.000000
mean    25.50000   1.980000
std     14.57738   0.820403
min      1.00000   1.000000
25%     13.25000   1.000000
50%     25.50000   2.000000
75%     37.75000   3.000000
max     50.00000   3.000000
```

8. To print summary of categorical values

```
df2['Industry_aggregation_NZSIOC']=df2['Industry_aggregation_NZSIOC'].astype('category')
print(df2.describe(include=['category']))
```

OUTPUT

```
        Industry_aggregation_NZSIOC
count                            39
unique                            1
top                         Level 1
freq                             39
```

9. To print categorical column values alone using describe method

```
print(df2["Industry_aggregation_NZSIOC"].describe())
```

OUTPUT

```
count           39
unique           1
top       Level 1
freq            39
Name: Industry_aggregation_NZSIOC, dtype: object
```

10. To print the summary of single attribute using describe

```
print(df["Name"].describe())
```

OUTPUT

```
count                50
unique               50
top       Sarah Johnson
freq                  1
Name: Name, dtype: object
```

11. The method to print distinct observations for each attribute

```
print(df.nunique())
print(df2.nunique())

for c in df2.columns:
    print(f"{c} : {df2[c].unique()}")
```

OUTPUT

```
Units                          3
Variable_code                 32
Variable_name                 33
Variable_category              3
Value                         39
dtype: int64
Year : [2023.   nan]
Industry_aggregation_NZSIOC : ['Level 1', NaN]
Categories (1, object): ['Level 1']
Industry_name_NZSIOC : ['All industries' 'Agriculture, Forestry and Fishing' nan]
Units : ['Dollars (millions)' 'Dollars' 'Percentage' nan]
```

12. The method to print unique values of a column in ascending order

```
print(df["Name"].sort_values().unique())
print(df2["Year"].sort_values().unique())
```

OUTPUT

```
'Ryan Anderson' 'Samuel Moore' 'Sarah Johnson' 'Scarlett Allen'
'Sophia Scott' 'Stella Jackson' 'Thomas Davis' 'Victoria Martinez'
'William White' 'Zoey Hall']
2023.   nan]
```

13. To print the summary of a column by grouping the data

```
print(df2.groupby("Units").describe())
```

OUTPUT

	Year							
	count	mean	std	min	25%	50%	75%	max
Units								
Dollars	2.0	2023.0	0.0	2023.0	2023.0	2023.0	2023.0	2023.0
Dollars (millions)	32.0	2023.0	0.0	2023.0	2023.0	2023.0	2023.0	2023.0
Percentage	5.0	2023.0	0.0	2023.0	2023.0	2023.0	2023.0	2023.0

14. The code to print the number of groups created along with row number

```
grouped=df2.groupby("Units")
print(len(grouped))
print(grouped.size())

for group_number, (group_name, group_data) in enumerate(grouped, 1):
    print(f"Group {group_number} ({group_name}):")
    print("Row indices:", group_data.index.tolist())
    print()


print("Total number of groups:", len(grouped))
```

OUTPUT

```
3
Units
Dollars                2
Dollars (millions)    32
Percentage             5
dtype: int64
Group 1 (Dollars):
Row indices: [24, 25]

Group 2 (Dollars (millions)
Row indices: [0, 1, 2, 3, 4

Group 3 (Percentage):
Row indices: [26, 27, 28, 2

Total number of groups: 3
```

15. The method to print size of the group

```
print(grouped.size())
```

OUTPUT

```
Units
Dollars                2
Dollars (millions)    32
Percentage             5
dtype: int64
```

16. The code to print particular group values

```
print(grouped.get_group('Dollars'))
```

OUTPUT

```
       Year Industry_aggregation_NZSIOC Industry_name_NZSIOC    Units
24   2023.0                     Level 1       All industries  Dollars
25   2023.0                     Level 1       All industries  Dollars
```

17. The code to print count, max and min values of a group

```
gr=df.groupby("Levels")
print(gr.count())
print(gr.max())
print(gr.min())
```

OUTPUT

```
        NO.  Name
Levels
1        17    17
2        17    17
3        16    16
        NO.                Name
Levels
1        50      William White
2        45  Victoria Martinez
3        49          Zoey Hall
        NO.               Name
Levels
1         1  Alexander Carter
2         5    Benjamin Young
3         2       David Clark
```

B. DATA ACQUISITION AND DESCRIPTION : NUMPY

18. To create an array with three rows and three columns

```
arr=np.array([[1,2,3],[2,34,4],[45,7,5]])
arr
OUTPUT
```

```
array([[ 1,  2,  3],
       [ 2, 34,  4],
       [45,  7,  5]])
```

19. The code to print dimensions of the array

```
arr.shape
```

OUTPUT

```
(3, 3)
```

20. Create a 1D and 2D array with default initialization of zeros

```
a=np.zeros(2)
ar=np.zeros((2,2))
print(a)
print(ar)
```

OUTPUT

```
[0. 0.]
```

```
array([[0., 0.],
       [0., 0.]])
```

21. To create an identity matrix with 5 rows and 5 columns

```
id=np.eye(5)
print(id)
```

OUTPUT

```
array([[1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1.]])
```

22. The code to retrieve second row first column value in a 3x3 matrix

```
print(arr)
arr[1,0]
```

OUTPUT

```
[[ 1  2  3]
 [ 2 34  4]
 [45  7  5]]
2
```

23. The code to print all column values of second row in a 3x3 matrix

```
arr[1,:]
```

OUTPUT

```
array([ 2, 34,  4])
```

24. The code to retrieve all the values from second column in a 3x3 matrix

```
arr[:,1]
```

OUTPUT

```
array([ 2, 34,  7])
```

25. To retrieve last column value in a 3x3 matrix

```
arr[1:3,:]
```

OUTPUT

```
array([[ 2, 34,  4],
       [45,  7,  5]])
```

26. To create an array with numbers specified in the range

```
ar=np.arange(10,20)
print(ar)
```

OUTPUT

```
[10 11 12 13 14 15 16 17 18 19]

[[ 1  2  3]
 [ 2 34  4]
 [45  7  5]]
[[ 1  2 45]
 [ 2 34  7]
 [ 3  4  5]]
```

27. To find transpose of a matrix

```
print(arr)
print(arr.T)
```

OUTPUT

```
[[ 1  2  3]
 [ 2 34  4]
 [45  7  5]]
[[ 1  2 45]
 [ 2 34  7]
 [ 3  4  5]]
```

28. To find determinant of a matrix

```
ap=np.array([[10,2],[2,3]])
print(det(ap))
```

OUTPUT

```
26.000000000000004
```

29. To print diagonal elements of a matrix

```
print(np.diag(arr))
```

OUTPUT

```
[ 1 34  5]
```

30. Basic Matrix operations

```
x1=np.array([[1,2],[3,4]])
x2=np.array([[8,7],[6,9]])
print("add")
print(x1+x2)
print("sub")
print(x1-x2)
print("mul")
print(x1*x2)
print("division")
print(x1/x2)
```

OUTPUT

```
add
[[ 9  9]
 [ 9 13]]
sub
[[-7 -5]
 [-3 -5]]
mul
[[ 8 14]
 [18 36]]
division
[[0.125      0.28571429]
 [0.5        0.44444444]]
```

31. To create an array with random integer values of size (3,5)

```
ar=np.random.randint(0,100,(3,5))
print(ar)
```

OUTPUT

```
array([[23, 91, 49, 21, 75],
       [55,  9, 20, 34, 17],
       [82, 45, 21, 95, 58]])
```

32. To create 1D array with 5 values of equal step size

```
ar=np.linspace(0,4,5)
print(ar)
```

OUTPUT

```
array([0., 1., 2., 3., 4.])
```

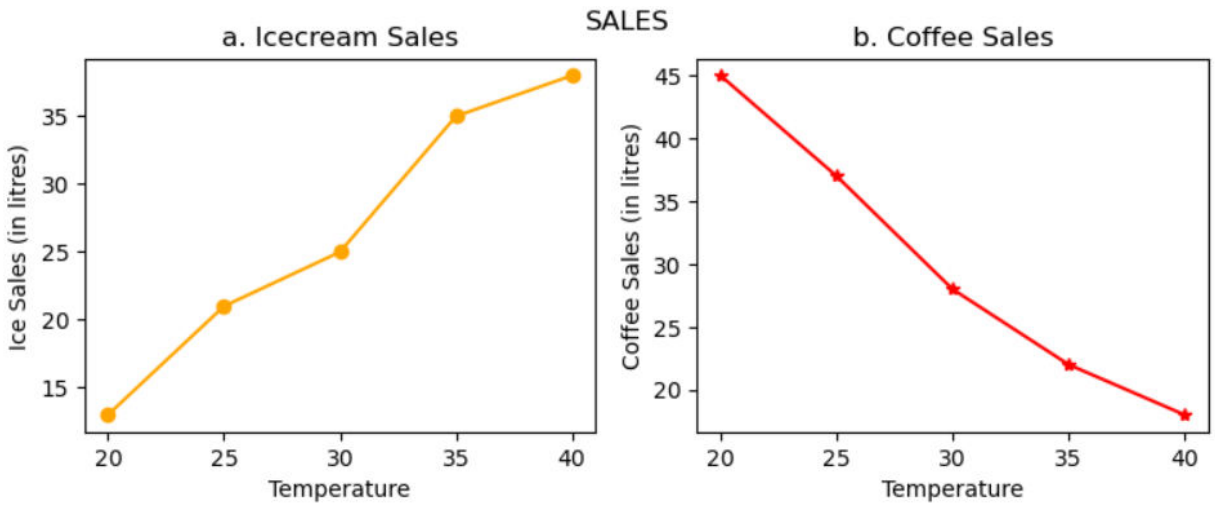
C. DATA ACQUISITION AND DESCRIPTION : MATPLOTLIB

33. Mapping the plot diagram using matplotlib

```
import matplotlib.pyplot as plt
```

```
#Data for subplots
temperature=[20, 25, 30, 35,40]
icesales=[13, 21, 25, 35, 38]
coffeesales=[45, 37, 28, 22,18]

fig, ax=plt.subplots(nrows=1,ncols=2, figsize=(9,3))
#Set title for subplots
ax[0].set_title('a. Icecream Sales')
ax[1].set_title('b. Coffee Sales')

ax[0].plot(temperature, icesales, '-o',c='orange')
ax[1].plot(temperature, coffeesales, '-*',c='red')

#Set the Xlabel and Ylabel
ax[0].set_xlabel('Temperature')
ax[0].set_ylabel('Ice Sales (in litres)')
ax[1].set_xlabel('Temperature')
ax[1].set_ylabel('Coffee Sales (in litres)')
fig.suptitle('SALES')
plt.show()
```

OUTPUT



RESULT

Thus, the above program was executed and implemented successfully

DATA PREPROCESSING

Ex.No.:

Date:

A. FIVE NUMBER SUMMARY

AIM

To learn and implement data preprocessing

METHODS USED

1. **Five-Number Summary Calculation**
    a. Using describe() method in Pandas.
    b. Manual calculation using min(), quantile(), and max().
2. **Box Plot for Data Visualization**
    a. Using Matplotlib's boxplot() function.
3. **Covariance and Correlation Calculation**
    a. Manual calculation using mean and summation formula.
    b. Using Pandas' built-in cov() and corr() methods.
4. **Handling Missing Values**
    a. Checking missing values using isnull().sum().
    b. Removing missing values using dropna().
    c. Filling missing values with median, mean, constant (0), and forward fill.
5. **Plotting Covariance and Correlation Matrices**
    a. Using Seaborn's heatmap() function.

LIBRARIES USED

- **Pandas**: Data manipulation (read_csv, describe, fillna, dropna, cov, corr).
- **NumPy**: Statistical calculations (mean, std, sum).
- **Matplotlib**: Data visualization (boxplot, figure, show).
- **Seaborn**: Heatmaps for covariance and correlation matrices (heatmap).

PROCEDURE

1. **Load Dataset**
    a. Read CSV files using pd.read_csv().
2. **Compute Five-Number Summary**
    a. Use df.describe() or manually compute min, Q1, median, Q3, and max.
3. **Visualize Data with Box Plot**
    a. Use plt.boxplot() to display distributions.
4. **Compute Covariance and Correlation**
    a. Manually calculate using formulas or use Pandas' cov() and corr().

5. **Handle Missing Values**
    a. Identify missing values using isnull().sum().
    b. Use dropna() to remove missing values.
    c. Fill missing values with median(), mean(), a constant (0), or ffill().
6. **Plot Covariance and Correlation Matrices**
    a. Compute matrices using df.cov() and df.corr().
    b. Visualize using sns.heatmap().

1. **Five number summary**

```
import pandas as pd
df = pd.read_excel('carM.xlsx',sheet_name="carM")
five_num_summary = df.describe().loc[['min', '25%', '50%', '75%', 'max']]
#location of specific rows and columns
five_num_summary=five_num_summary.applymap(lambda x:f"{x:.1f}")
print(five_num_summary)

#hardcode
min_value = df['Sales'].min()

q1 = df['Sales'].quantile(0.25)

median = df['Sales'].median()

q3 = df['Sales'].quantile(0.75)

max_value = df['Sales'].max()

print("For sales: ", f"Min: {min_value:.2f}, Q1: {q1:.2f}, Median: {median}, Q3: {q3}, Max:
{max_value}")
```

OUTPUT

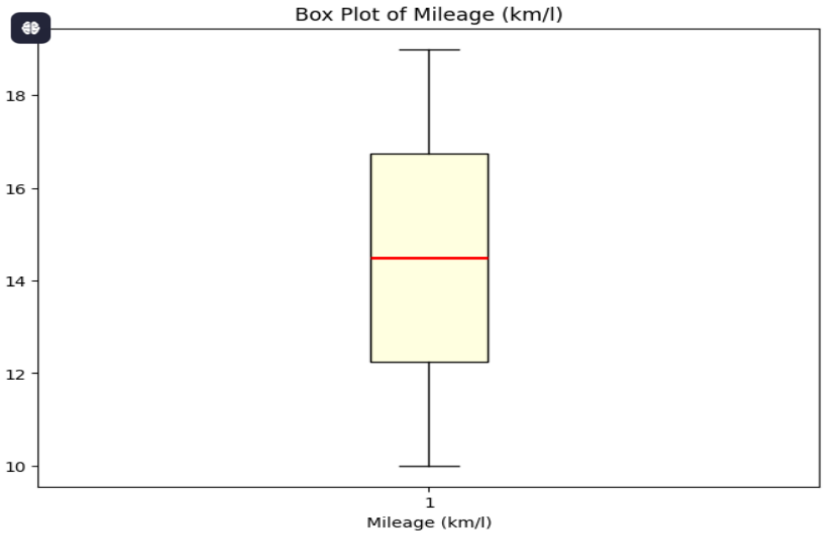
```
     Mileage (km/l)  Price (lakhs)  Sales (units)
min           10.00          4.000         150.0
25%           12.25          5.125         192.5
50%           14.50          6.250         215.0
75%           16.75          7.375         265.0
max           19.00          9.500         320.0
For sales:  Min: 150, Q1: 192.5, Median: 215.0, Q3: 265.0, Max: 320
```

2. **Implementation of boxplot**

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

```
df = pd.read_excel('carM.xlsx',sheet_name="carM")


plt.figure(figsize=(8, 6))
plt.boxplot(df['Mileage'], patch_artist=True, boxprops=dict(facecolor='lightyellow',
color='black'),
            medianprops=dict(color='red', linewidth=2))
plt.title('Box Plot of Mileage ')
plt.xlabel('Mileage')
plt.show()

five_num_mileage = df['Mileage'].describe()[['min', '25%', '50%', '75%', 'max']]

print("\nFive Number Summary for 'Mileage':")
print(five_num_mileage)

plt.figure(figsize=(4,4))
plt.boxplot(df['Sales'],patch_artist=True,boxprops=dict(facecolor='lightgreen',color='black'),
          medianprops=dict(color='red',linewidth=3))
```

OUTPUT



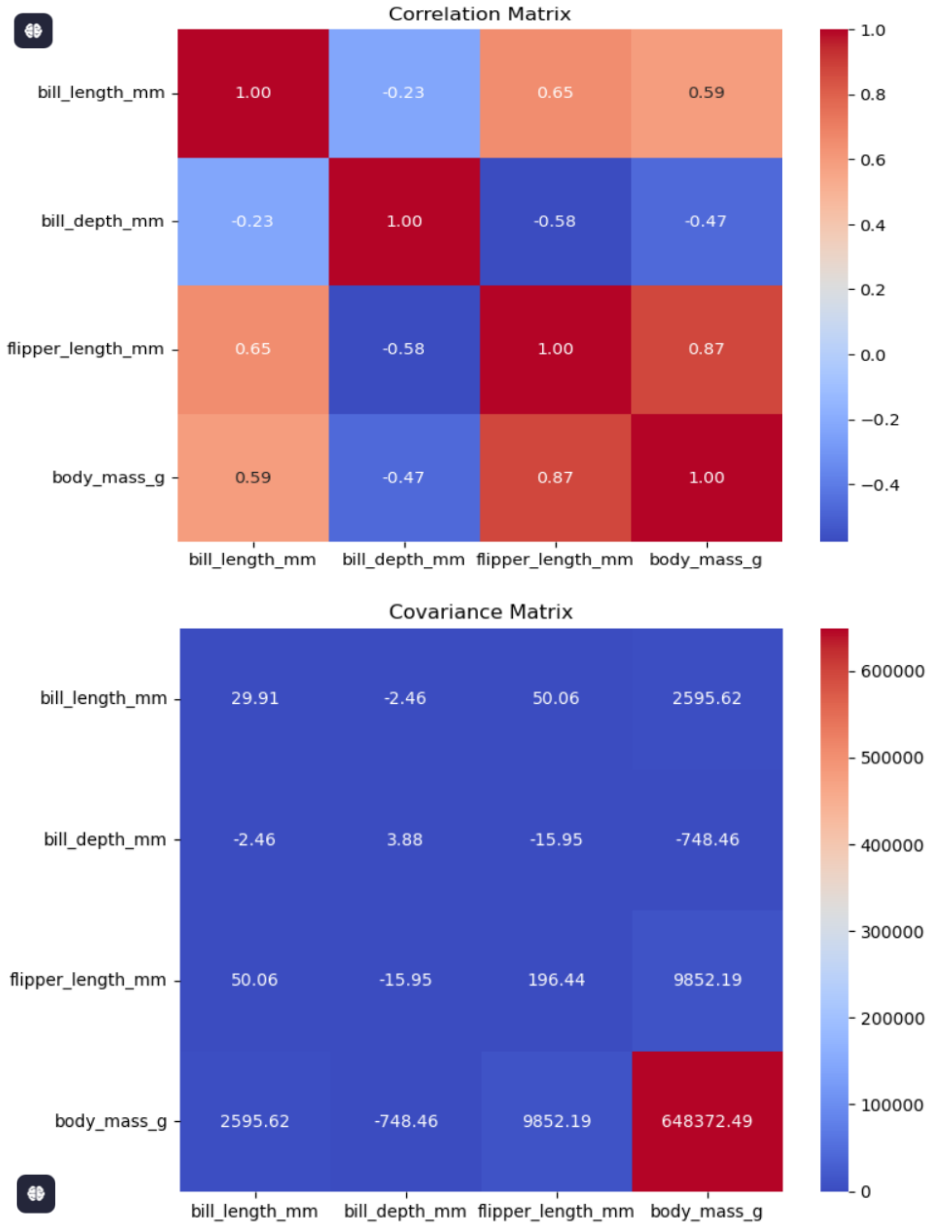

```
Five Number Summary for 'Mileage (km/l)':
min    10.00
25%    12.25
50%    14.50
75%    16.75
max    19.00
Name: Mileage (km/l), dtype: float64
```

3. Co-variance and Co-relation using seaborn

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

df = sns.load_dataset("penguins").dropna()

cov_matrix = df.cov(numeric_only=True)
corr_matrix = df.corr(numeric_only=True)

plt.figure(figsize=(8, 6))
sns.heatmap(cov_matrix, annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Covariance Matrix")
plt.show()

plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Correlation Matrix")
```

```
plt.show()
```

OUTPUT



B. MISSING VALUE IMPUTATION

```
import pandas as pd

# Load the dataset from 'miss.csv'
df = pd.read_excel('carM.xlsx',sheet_name="miss")

# Print the original dataset
print("Original Dataset:")
```

```
print(df)
print("\n")

# 1. Checking for missing values
print("Checking for Missing Values:")
print(df.isnull().sum())
print("\n")

# 2. Dropping missing values
df_dropped = df.dropna()
print("Dataset After Dropping Missing Values:")
print(df_dropped)
print("\n")

# 3. Filling with Median
df_filled_median = df.fillna(df.median(numeric_only=True))
print("Dataset After Filling Missing Values with Median:")
print(df_filled_median)
print("\n")

# 4. Filling with Mean
df_filled_mean = df.fillna(df.mean(numeric_only=True))
df_filled_mean = df_filled_mean.applymap(lambda x: f"{x:.2f}")
print("Dataset After Filling Missing Values with Mean:")
print(df_filled_mean)
print("\n")

# 5. with  constant (e.g., 0)
df_filled_constant = df.fillna(0)
print("Dataset After Filling Missing Values with Constant (0):")
print(df_filled_constant)
print("\n")

# 6. Forward Fill
df_forward_fill = df.ffill()
print("Dataset After Forward Filling Missing Values:")
print(df_forward_fill)
print("\n")
```

OUTPUT

Original Dataset:

	Car_Model	Sales (units)	Price (lakhs)	Mileage (km/l)
0	Model A	200.0	6.5	15.0
1	Model B	NaN	5.5	13.5
2	Model C	250.0	NaN	14.0
3	Model D	150.0	7.0	NaN
4	Model E	300.0	6.0	16.0
5	Model F	NaN	NaN	12.0

Checking for Missing Values:

```
Car_Model         0
Sales (units)     2
Price (lakhs)     2
Mileage (km/l)    1
dtype: int64
```

Dataset After Dropping Missing Values:

	Car_Model	Sales (units)	Price (lakhs)	Mileage (km/l)
0	Model A	200.0	6.5	15.0
4	Model E	300.0	6.0	16.0

Dataset After Filling Missing Values with Median:

	Car_Model	Sales (units)	Price (lakhs)	Mileage (km/l)
0	Model A	200.0	6.50	15.0
1	Model B	225.0	5.50	13.5
2	Model C	250.0	6.25	14.0
3	Model D	150.0	7.00	14.0
4	Model E	300.0	6.00	16.0
5	Model F	225.0	6.25	12.0

Dataset After Filling Missing Values with Mean:

	Car_Model	Sales (units)	Price (lakhs)	Mileage (km/l)
0	Model A	200.0	6.50	15.0
1	Model B	225.0	5.50	13.5
2	Model C	250.0	6.25	14.0
3	Model D	150.0	7.00	14.1
4	Model E	300.0	6.00	16.0
5	Model F	225.0	6.25	12.0

Dataset After Filling Missing Values with Mode:

	Car_Model	Sales (units)	Price (lakhs)	Mileage (km/l)
0	Model A	200.0	6.5	15.0
1	Model B	NaN	5.5	13.5
2	Model C	250.0	NaN	14.0
3	Model D	150.0	7.0	NaN
4	Model E	300.0	6.0	16.0
5	Model F	NaN	NaN	12.0

Dataset After Filling Missing Values with Constant (0):

	Car_Model	Sales (units)	Price (lakhs)	Mileage (km/l)
0	Model A	200.0	6.5	15.0
1	Model B	0.0	5.5	13.5
2	Model C	250.0	0.0	14.0
3	Model D	150.0	7.0	0.0
4	Model E	300.0	6.0	16.0
5	Model F	0.0	0.0	12.0

Dataset After Forward Filling Missing Values:

	Car_Model	Sales (units)	Price (lakhs)	Mileage (km/l)
0	Model A	200.0	6.5	15.0
1	Model B	200.0	5.5	13.5
2	Model C	250.0	5.5	14.0
4	Model E	300.0	6.0	16.0
5	Model F	300.0	6.0	12.0

Dataset After Backward Filling Missing Values:

	Car_Model	Sales (units)	Price (lakhs)	Mileage (km/l)
0	Model A	200.0	6.5	15.0
1	Model B	250.0	5.5	13.5
2	Model C	250.0	7.0	14.0
3	Model D	150.0	7.0	16.0
4	Model E	300.0	6.0	16.0
5	Model F	NaN	NaN	12.0

C. COVARIANCE AND CORRELATION

AIM

To calculate and compare the **covariance** and **correlation** between **Price** and **Mileage** of cars using **manual formulas** and **Pandas library functions**.

METHODS USED

- **Manual Calculation**:
  - Covariance calculated using the statistical formula.
  - Correlation calculated based on covariance and standard deviations.
- **Library Functions**:
  - Pandas `.cov()` for covariance.
  - Pandas `.corr()` for correlation

PROCEDURE

- **Import necessary libraries**:
  - `pandas` for data handling.
  - `numpy` for numerical operations.
- **Load the dataset**:
  - Read the Excel file `CarM.xlsx` from the sheet named `"carM"`.
- **Extract the required columns**:
  - Select **Price** and **Mileage** columns for analysis.
- **Define functions for manual calculations**:
  - `calculate_covariance(x, y)` : Computes covariance using the formula:

$$\text{Cov}(X,Y) = \frac{\sum(X_i - \bar{X})(Y_i - \bar{Y})}{n-1}$$

  - `calculate_correlation(x, y)` : Computes correlation using:

$$\text{Corr}(X,Y) = \frac{\text{Cov}(X,Y)}{\sigma_X \sigma_Y}$$

CODE

```
import pandas as pd
import numpy as np

df = pd.read_excel('CarM.xlsx',sheet_name="carM")

print(df)
print("\n")

x_price = df['Price']
y_mileage = df['Mileage']

def calculate_covariance(x, y):
    x_mean = np.mean(x)
    y_mean = np.mean(y)
    covariance = sum((x - x_mean) * (y - y_mean)) / (len(x) - 1)
    return covariance

def calculate_correlation(x, y):
    covariance = calculate_covariance(x, y)
    std_x = np.std(x, ddof=1)
    std_y = np.std(y, ddof=1)
    correlation = covariance / (std_x * std_y)
    return correlation

cov_price_mileage_manual = calculate_covariance(x_price, y_mileage)
corr_price_mileage_manual = calculate_correlation(x_price, y_mileage)

cov_matrix = df.cov(numeric_only=True)
corr_matrix = df.corr(numeric_only=True)

cov_price_mileage_pandas = cov_matrix.loc['Price', 'Mileage']
corr_price_mileage_pandas = corr_matrix.loc['Price', 'Mileage']

print("Covariance Results:")
print(f"Covariance between Price and Mileage (Manual): {cov_price_mileage_manual:.2f}")
print(f"Covariance between Price and Mileage (Pandas): {cov_price_mileage_pandas:.2f}")
print("\n")

print("Correlation Results:")
print(f"Correlation between Price and Mileage (Manual): {corr_price_mileage_manual:.2f}")
print(f"Correlation between Price and Mileage (Pandas): {corr_price_mileage_pandas:.2f}")
print("\n")
```

OUTPUT

```
Dataset:
   Mileage (km/l)  Price (lakhs)  Sales (units)
0              15            5.0            250
1              12            7.5            200
2              18            4.5            300
3              14            6.0            180
4              16            8.0            220
5              10            9.5            150
6              13            7.0            210
7              17            5.5            270
8              11            6.5            190
9              19            4.0            320


Covariance Results:
Covariance between Price and Mileage (Manual): -3.97
Covariance between Price and Mileage (Pandas): -3.97


Correlation Results:
Correlation between Price and Mileage (Manual): -0.77
Correlation between Price and Mileage (Pandas): -0.77
```

RESULT

Thus, the above program was executed and implemented successfully.

LINEAR REGRESSION

Ex.No.:

Date:

A. SIMPLE LINEAR REGRESSION

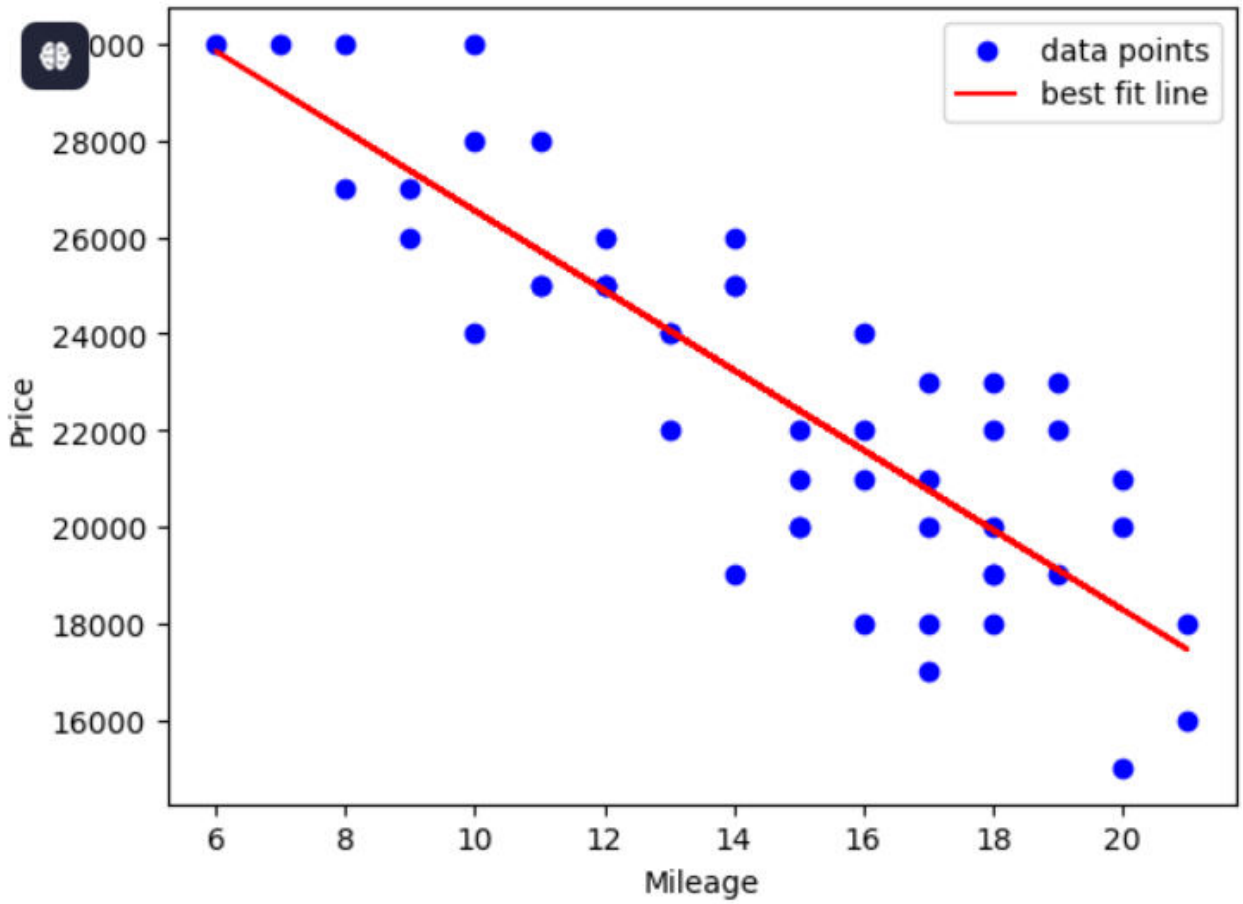
AIM

To implement linear regression with and without using libraries

1. **Simple linear regression**

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

df=pd.read_excel('carM.xlsx',sheet_name='carS')
x=df['Mileage'].values.reshape(-1,1)
y=df['Price'].values.reshape(-1,1)

rmodel=LinearRegression()     #linear regression
rmodel=rmodel.fit(x,y)

rslope=rmodel.coef_     #slope
rintercept=rmodel.intercept_  #intercept

##evaluating
y_predict=rmodel.predict(x)
#predicting y cap from giving a,x,b

rmse=np.sqrt(mean_squared_error(y,y_predict))

r2=rmodel.score(x,y)
plt.scatter(x,y,color="blue",label="data points")
plt.plot(x,y_predict,color="red",label="best fit line")
plt.xlabel("Mileage")
plt.ylabel("Price")
plt.legend()

plt.show()
```

OUTPUT



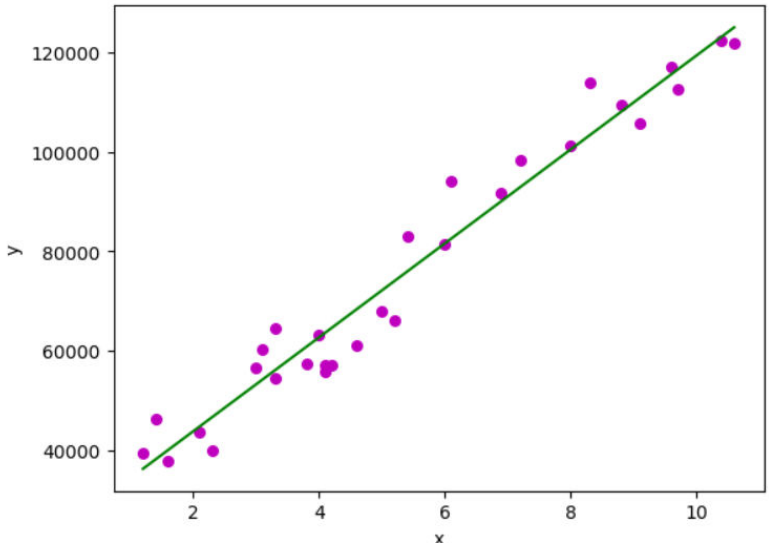
2. **Simple linear regression without using libraries**

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

df=pd.read_csv('Salary_dataset.csv')
x=df['YearsExperience'].values.T
y=df['Salary'].values.T

n=np.size(x)
meanx=np.mean(x)
meany=np.mean(y)
Sxy=np.sum(y*x)-n*meanx*meany
Sxx=np.sum(x*x)-n*meanx*meanx
a=Sxy/Sxx#slope=ax+b
b=meany-meanx*a#intercept

print(f"Co-efficient is {a,b}")
plt.scatter(x,y,color="m",marker="o",s=30)
ypred=(a*x)+b
```

```
plt.plot(x,ypred,color="g")
plt.xlabel('x')
plt.ylabel('y')

sse = np.sum((y - ypred) ** 2)  # Sum of Squared Errors
mse = sse / len(y)  # Mean Squared Error
rmse = np.sqrt(mse)  # Root Mean Squared Error
mae= np.mean(np.abs(y - ypred))  # Mean Absolute Error
sst = np.sum((y - meany) ** 2)  # Total Sum of Squares
ssr = sst - sse  # Sum of Squares for Regression

print(f"sse {sse},mse {mse},rmse {rmse}\
mae {mae} , sse {sst}, ssr {ssr}");
```

OUTPUT

```
Co-efficient is (9449.962321455083, 24848.203966523164)
sse 938128551.6684282,mse 31270951.72228094,rmse 5592.0436087606595mae 4644.201289443537 , sse 21794977852.0, ssr 20856849300.331573
```



B. **MULTIPLE LINEAR REGRESSION**

AIM

To implement multiple linear regression using librarie

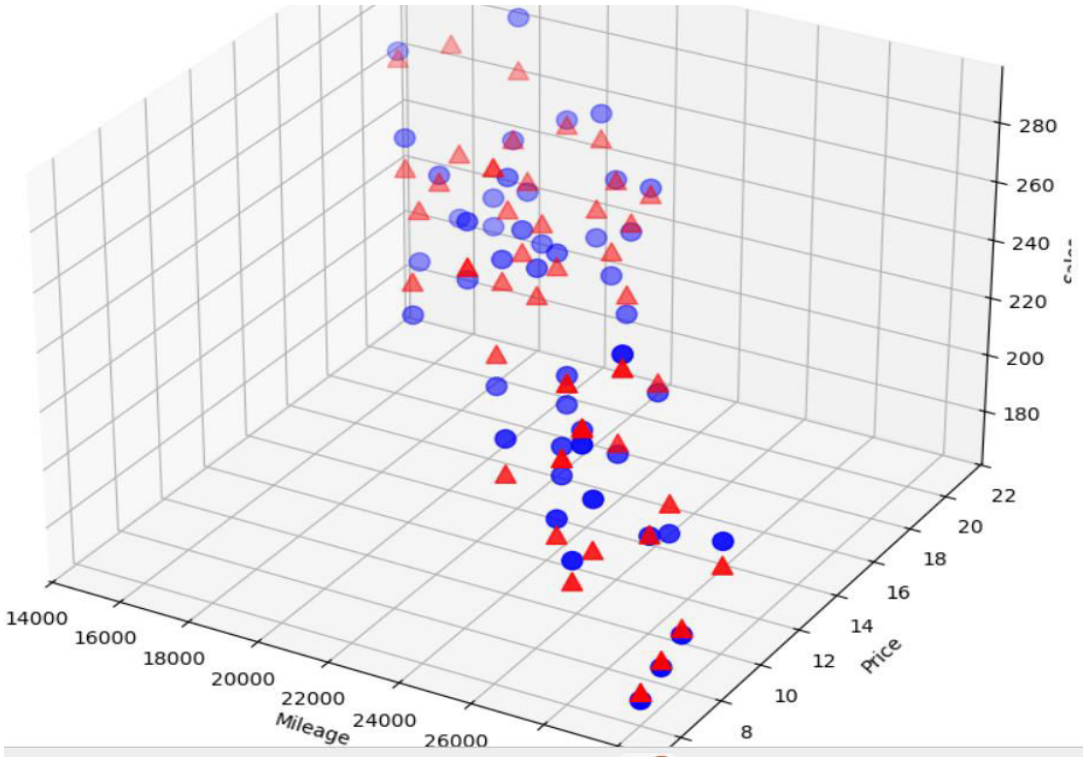
CODE

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

df = pd.read_csv('/content/CAR_SALES.csv')
X = df.iloc[:, [0, 1]].values
y = df.iloc[:, 2].values

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

a_mileage, a_price = rmodel.coef_
b = rmodel.intercept_

print(f"slope (Mileage): {a_mileage:.2f}")
print(f"slope (Price): {a_price:.2f}")
print(f"intercept: {b:.2f}")

y_pred = rmodel.predict(X)

mse = mean_squared_error(y, y_pred)
y_mean = np.mean(y)
sst = np.sum((y - y_mean) ** 2)
sse = np.sum((y - y_pred) ** 2)
r2 = 1 - (sse / sst)

result_df = pd.DataFrame({
   'Mileage': X[:, 0],
   'Price': X[:, 1],
   'Sales': y,
   'y_pred': y_pred
})

print(result_df.round(2))
```

```
print(f"MSE: {mse:.2f}")
print(f"SST: {sst:.2f}")
print(f"SSE: {sse:.2f}")
print(f"R2: {r2:.2f}")

fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(X[:, 0], X[:, 1], y, color='blue', label='Actual data', s=100)
ax.scatter(X[:, 0], X[:, 1], y_pred, color='red', label='Predicted data', s=100, marker='^')
ax.set_xlabel('Mileage')
ax.set_ylabel('Price')
ax.set_zlabel('Sales')
ax.set_title('Sales vs. Price & Mileage (Multiple Regression)', fontsize=16, fontweight='bold')
ax.legend()
plt.show()
```

OUTPUT

```
slope (Mileage): 12.11
slope (Price): -10.51
intercept: 120.06
   Mileage  Price  Sales  y_pred
0     15.0    5.0    250  249.24
1     12.0    7.5    200  186.63
2     18.0    4.5    300  290.84
3     14.0    6.0    180  226.62
4     16.0    8.0    220  229.84
5     10.0    9.5    150  141.39
6     13.0    7.0    210  204.00
7     17.0    5.5    270  268.22
8     11.0    6.5    190  185.03
9     19.0    4.0    320  308.20
MSE: 281.06
SST: 26890.00
SSE: 2810.63
R²: 0.90
```



RESULT

Thus, the above program was executed and implemented successfully.

MCCULLOCH PITTS NEURON

Ex.No.:

Date:

AIM

To implement mc-culloch pitts neuron model using user weights and threshold implementing AND, OR, NAND, NOR gate

METHODS USED

- User-defined function for computing logic gates
- Threshold-based activation function
- Looping through all binary combinations of inputs

LIBRARIES USED

numpy – For handling binary input arrays

PROCEDURE

- Define Binary Inputs as `np.array([0,1])`.
- Create a Function (`mc()`) that:
- Takes input values (`i`, `j`), weights (`w1`, `w2`), and a threshold.
- Computes weighted sum: `tem = (i*w1) + (j*w2)`.
- Returns `1` if `tem >= threshold`, otherwise `0`.
- Accept User Input for weights and threshold using `input()`.
- Validate Inputs using `try-except` to handle invalid entries.
- Iterate Over Binary Inputs using nested loops and apply the `mc()` function to print logic gate outputs.

PROGRAM

```
import numpy as np


a=np.array([0,1])
```

```
def mc(i,j,w1,w2,threshold):
    tem=(i*w1) + (j*w2)
    if(tem>=threshold):
        return 1
    else:
        return 0



print("Logical gates")
try:
    w1=int(input("Enter weight1: "))
    w2=int(input("Enter weight2: "))
    threshold=int(input("Enter threshold: "))
except ValueError:
    print("Invalid inputs")
    exit()

for i in a:
    for j in a:
        print(f"{i} {j} {mc(i,j,w1,w2,threshold)}")

import numpy as np

a=np.array([0,1])

def mc(i,j,w1,w2,threshold):
    tem=(i*w1) + (j*w2)
    if(tem>=threshold):
        return 1
```

```
    else:
        return 0

def and_gate(i,j):
    w1=1
    w2=1
    threshold=2
    return mc(i,j,w1,w2,threshold)

print("AND gate")
for i in a:
    for j in a:
        print(f"{i} {j} {and_gate(i,j)}")


def or_gate(i,j):
    w1=1
    w2=1
    threshold=1
    return mc(i,j,w1,w2,threshold)

print("OR gate")
for i in a:
    for j in a:
        print(f"{i} {j} {or_gate(i,j)}")


def nand_gate(i,j):
    w1=-1
```

```
    w2=-1
    threshold=-1
    return mc(i,j,w1,w2,threshold)

print("NAND gate")
for i in a:
    for j in a:
        print(f"{i} {j} {nand_gate(i,j)}")

def nor_gate(i,j):
    w1=-1
    w2=-1
    threshold=0
    return mc(i,j,w1,w2,threshold)

print("NOR gate")
for i in a:
    for j in a:
        print(f"{i} {j} {nor_gate(i,j)}")
```

OUTPUT

```
Logical gates
Enter weight1: 1
Enter weight2: 1
Enter threshold: 1
0 0 0
0 1 1
1 0 1
1 1 1
```

```
AND gate
0 0 0
0 1 0
1 0 0
1 1 1
OR gate
0 0 0
0 1 1
1 0 1
1 1 1
NAND gate
0 0 1
0 1 1
1 0 1
1 1 0
NOR gate
0 0 1
0 1 0
1 0 0
1 1 0
```

RESULT

Thus, the above program was implemented successfully.

HEBB NET

Ex.No.:

Date:

AIM

To construct hebb net to implement AND, OR logical gates

METHODS USED

- Hebbian Learning Rule:
- Update weights using the formula:

$$w_1 = w_1 + x_1 \cdot y$$

$$w_2 = w_2 + x_2 \cdot y$$

$$\text{bias} = \text{bias} + y$$

- Iterative Weight Update
- Final Weight Calculation and Testing

LIBRARIES USED

numpy – For array operations

PROCEDURE

- Define Input and Output Pairs for AND and OR gates.
- x1, x2 (Inputs)
- y (Expected output)
- Initialize Weights & Bias from user input.
- Iterate Over Training Samples
- Update weights (w1, w2) and bias using the Hebbian rule.
- Print updated weights and bias after each iteration.
- Evaluate Final Weights
- Compute output for each input pair using the final weights and bias.
- Run for AND & OR Gates
- Use different expected outputs (y = [0,0,0,1] for AND, y = [0,1,1,1] for OR).

PROGRAM

#hebnet for and , or gate implementation

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

```
import numpy as np

def hebnet(x1, x2, y):
    w1 = float(input("Enter initial weight for x1: "))
    w2 = float(input("Enter initial weight for x2: "))
    bias = float(input("Enter initial bias: "))

    n = len(x1)

    for i in range(n):
        w1 += x1[i] * y[i]
        w2 += x2[i] * y[i]
        bias += y[i]

        print('---------------------------------')
        print(f'Updated weights after input {i+1}:')
        print(f'x1: {w1:.4f}')
        print(f'x2: {w2:.4f}')
        print(f'bias: {bias:.4f}')

    print("----------------------------")
    print(f'Final weights:')
    print(f'x1: {w1:.4f}')
    print(f'x2: {w2:.4f}')
    print(f'bias: {bias:.4f}')

    print("\nEvaluating test inputs:")
    for i in range(n):
```

```
    output = w1 * x1[i] + w2 * x2[i] + bias
    print(f'Input ({x1[i]}, {x2[i]}) -> Output: {output:.4f}')



x1 = np.array([0, 0, 1, 1])
x2 = np.array([0, 1, 0, 1])
y = np.array([0, 0, 0, 1])
print('AND gate ')
hebnet(x1, x2, y)
y=np.array([0,1,1,1])
print('OR gate implementation')
hebnet(x1,x2,y)
```

OUTPUT

```
AND gate
Enter initial weight for x1:  0.5
Enter initial weight for x2:  0.5
Enter initial bias:  1
--------------------------------
Updated weights after input 1:
x1: 0.5000
x2: 0.5000
bias: 1.0000
--------------------------------
Updated weights after input 2:
x1: 0.5000
x2: 0.5000
bias: 1.0000
--------------------------------
Updated weights after input 3:
x1: 0.5000
x2: 0.5000
bias: 1.0000
--------------------------------
Updated weights after input 4:
x1: 1.5000
x2: 1.5000
bias: 2.0000
----------------------------
Final weights:
x1: 1.5000
x2: 1.5000
bias: 2.0000
```

```
Evaluating test inputs:
Input (0, 0) -> Output: 2.0000
Input (0, 1) -> Output: 3.5000
Input (1, 0) -> Output: 3.5000
Input (1, 1) -> Output: 5.0000
OR gate implementation
Enter initial weight for x1:  0.4
Enter initial weight for x2:  0.4
Enter initial bias:  1
--------------------------------
Updated weights after input 1:
x1: 0.4000
x2: 0.4000
bias: 1.0000
--------------------------------
Updated weights after input 2:
x1: 0.4000
x2: 1.4000
bias: 2.0000
--------------------------------
Updated weights after input 3:
x1: 1.4000
x2: 1.4000
bias: 3.0000
--------------------------------
Updated weights after input 4:
x1: 2.4000
x2: 2.4000
bias: 4.0000
---------------------------
Final weights:
x1: 2.4000
x2: 2.4000
bias: 4.0000
```

RESULT

Thus, the above program was implemented successfully.

SINGLE LAYER PERCEPTRON

Ex.No.:

Date:

AIM

To implement a single layer perceptron to implement AND, OR logical gates without using libraries

METHODS USED

Perceptron Learning Rule:

- Compute weighted sum:

$$f = x1 \cdot w1 + x2 \cdot w2 + bias \cdot w0$$

- Apply activation function:

$$y_{out} = \begin{cases} 1, & \text{if } f > 0 \\ 0, & \text{otherwise} \end{cases}$$

- Compute error:

$$error = y - y_{out}$$

- Update weights:

$$w1 = w1 + \text{learning rate} \times error \times x1$$

$$w2 = w2 + \text{learning rate} \times error \times x2$$

$$w0 = w0 + \text{learning rate} \times error \times bias$$

LIBRARIES USED

numpy – For numerical operations

PROCEDURE

- Define Input and Output Pairs
- x1, x2 (Inputs)
- y (Expected Output)
- Initialize Weights and Bias
- User inputs initial values.

- Set Parameters
- Epochs – Number of iterations over the dataset.
- Learning Rate – Step size for weight updates.
- Train the Perceptron
- Compute weighted sum (f).
- Apply threshold activation function.
- Compute error and adjust weights.
- Print Weights After Each Iteration
- Display weight updates.
- Final Weights Display
- Show weights after training for AND and OR gates.

PROGRAM

```
import numpy as np

def imp(x1, x2, y):
    epochs = int(input("Enter the epochs: "))
    bias = 1
    learning_rate = 0.5

    a = []
    a.append(float(input("Enter the weight for bias: ")))
    a.append(float(input("Enter the weight for x1: ")))
    a.append(float(input("Enter the weight for x2: ")))
    w = np.array(a)

    n = x1.shape[0]

    for k in range(epochs):
        for i in range(n):
            f = x1[i] * w[1] + x2[i] * w[2] + bias * w[0]
            y_out = (f > 0).astype(int)
            error = y[i] - y_out

            if error != 0:
                w[1] += learning_rate * error * x1[i]
                w[2] += learning_rate * error * x2[i]
                w[0] += learning_rate * error * bias

            print('--------------------------------')
            print(f'Updated weights after input {i+1}:')
            print(f'x1: {w[1]:.4f}')
```

```
            print(f'x2: {w[2]:.4f}')
            print(f'bias: {w[0]:.4f}')

    print("----------------------------")
    print(f'Final weights after {epochs} epochs:')
    print(f'x1: {w[1]:.4f}')
    print(f'x2: {w[2]:.4f}')
    print(f'bias: {w[0]:.4f}')


x1 = np.array([0, 0, 1, 1])
x2 = np.array([0, 1, 0, 1])

y = np.array([0, 0, 0, 1])
print('AND gate implementation')
imp(x1, x2, y)

y = np.array([0, 1, 1, 1])
print('OR gate implementation')
imp(x1, x2, y)
```

OUTPUT

```
AND gate implementation
Enter the epochs:  1
Enter the weight for bias:  -0.5
Enter the weight for x1:  0.2
Enter the weight for x2:  0.3
--------------------------------
Updated weights after input 1:
x1: 0.2000
x2: 0.3000
bias: -0.5000
--------------------------------
Updated weights after input 2:
x1: 0.2000
x2: 0.3000
bias: -0.5000
--------------------------------
Updated weights after input 3:
x1: 0.2000
x2: 0.3000
bias: -0.5000
--------------------------------
Updated weights after input 4:
x1: 0.7000
x2: 0.8000
bias: 0.0000
---------------------------
Final weights after 1 epochs:
x1: 0.7000
x2: 0.8000
bias: 0.0000
```

RESULT

Thus, the program was implemented successfully.

MULTILAYER PERCEPTRON - FOREST FIRE PREDICTION

Ex.No.:

Date:

Construct a classification model using multilayer perceptron for predicting the occurrence of forest fire. Assume that dataset is holding two class classification based on that sensor data. Print weight matrix

AIM

The aim is to **predict the occurrence of forest fires** based on given input features using a **Multi-Layer Perceptron (MLP) classifier**. The model is trained on preprocessed data and evaluated using classification metrics.

LIBRARIES USED

1. **pandas** – For loading and manipulating the dataset.
2. **sklearn.preprocessing.LabelEncoder** – To convert categorical labels into numerical form.
3. **sklearn.preprocessing.StandardScaler** – To standardize input features.
4. **sklearn.model_selection.train_test_split** – To split the dataset into training and testing sets.
5. **sklearn.neural_network.MLPClassifier** – To implement a Multi-Layer Perceptron (MLP) neural network.
6. **sklearn.metrics** – For evaluating model performance using confusion matrix and classification report.
7. **matplotlib.pyplot** – To visualize the loss curve of the MLP model.

METHODS USED

1. **Label Encoding:**
    a. The target variable ('Fire') is converted into numerical values using LabelEncoder().
2. **Feature Scaling:**
    a. The input features are scaled using StandardScaler() to ensure uniform data distribution.
3. **Train-Test Split:**
    a. The dataset is divided into **70% training** and **30% testing** for model evaluation.
4. **Multi-Layer Perceptron (MLP) Classifier:**
    a. A neural network is built with:
        i. **One hidden layer** containing **3 neurons**.
        ii. **Logistic activation function** (sigmoid).
        iii. **Adam optimizer** for weight updates.
        iv. **Learning rate of 0.9** and **100 iterations**.
5. **Model Training and Prediction:**
    a. The MLP model is trained using fit() and tested using predict().
6. **Performance Evaluation:**
    a. **Confusion matrix** and **classification report** are used to assess model accuracy, precision, recall, and F1-score.
7. **Loss Curve Visualization:**
    a. The **loss curve** is plotted to analyze the model's convergence over epochs.

Procedure:

1. Load the dataset from Forest_fire.csv into a DataFrame.
2. Display the dataset before transformation.
3. Convert the categorical target variable ('Fire') into numerical values using LabelEncoder().
4. Standardize the input features using StandardScaler().
5. Split the dataset into training (70%) and testing (30%) sets.
6. Define an **MLPClassifier** with a **3-neuron hidden layer**, logistic activation function, and Adam optimizer.
7. Train the model using fit().
8. Predict the test set labels using predict().
9. Compute the **confusion matrix** and **classification report** to evaluate performance.
10. Print the model's weight matrices and biases.
11. Plot the **loss curve** to observe model convergence.

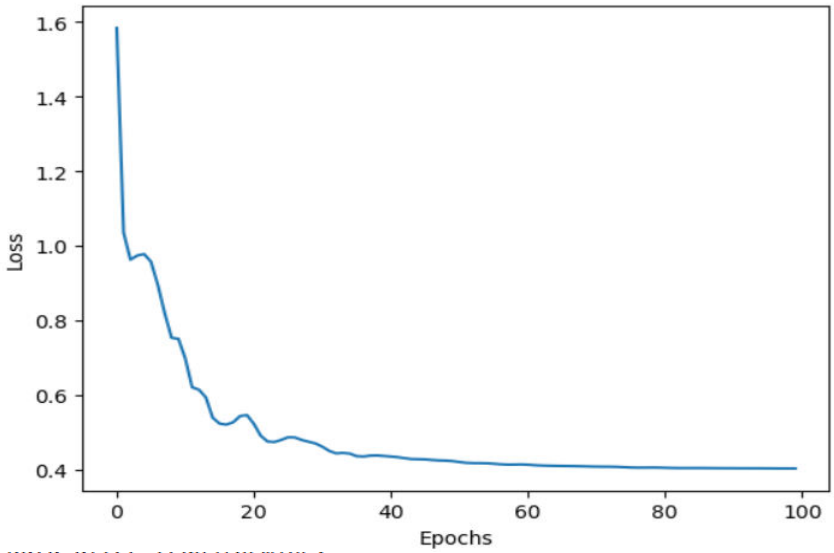
CODE

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt

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

#data preprocessing
le=LabelEncoder()
scale=StandardScaler()
print("Data before transforming:\n",df)
df['Fire']=le.fit_transform(df['Fire'])
print("Data after transforming:\n",df)

inplist=df.columns[:-1]
print("Data before scaling:\n",df)
df[inplist]=scale.fit_transform(df[inplist])
print("Data after scaling:\n",df)

x=df.values[:,:-1]
y=df.values[:,-1]

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=11)

cf=MLPClassifier(hidden_layer_sizes=(3),activation="logistic",max_iter=100,solver="adam",learning_ra
te="constant",learning_rate_init=0.9)

cf.fit(x_train,y_train)

y_pred=cf.predict(x_test)

con=confusion_matrix(y_test,y_pred)
print("confusion matrix:\n",con)
cla=classification_report(y_test,y_pred)
```

```
print("classification report:\n",cla)

print("coefficient:",cf.coefs_)
print("intercept: ",cf.intercepts_)

# Print weight matrices
print("\nWeight Matrices:")
for i, weight_matrix in enumerate(clf.coefs_):
    print(f"\nLayer {i + 1} Weights (Shape {weight_matrix.shape}):\n", weight_matrix)

loss_values=cf.loss_curve_

plt.plot(loss_values)
plt.xlabel('Epochs')
plt.ylabel('Loss')

plt.show()
```

OUTPUT

```
Data before transforming:
      day  month  year  Temperature   RH   Ws  Rain   FFMC  DMC    DC  ISI  \
0       1      6  2012           29   57   18   0.0  65.7  3.4   7.6  1.3
1       2      6  2012           29   61   13   1.3  64.4  4.1   7.6  1.0
2       3      6  2012           26   82   22  13.1  47.1  2.5   7.1  0.3
3       4      6  2012           25   89   13   2.5  28.6  1.3   6.9  0.0
4       5      6  2012           27   77   16   0.0  64.8  3.0  14.2  1.2
..    ...    ...   ...          ...  ...  ...   ...   ...  ...   ...  ...
117    26      9  2012           31   54   11   0.0  82.0  6.0  16.3  2.5
118    27      9  2012           31   66   11   0.0  85.7  8.3  24.9  4.0
119    28      9  2012           32   47   14   0.7  77.5  7.1   8.8  1.8
120    29      9  2012           26   80   16   1.8  47.4  2.9   7.7  0.3
121    30      9  2012           25   78   14   1.4  45.0  1.9   7.5  0.2

     BUI  FWI  Fire
0    3.4  0.5     y
1    3.9  0.4  n
2    2.7  0.1  n
3    1.7  0.0  n
4    3.9  0.5  n
..   ...  ...   ...
117  6.2  1.7  n
118  9.0  4.1  y
119  6.8  0.9  n
120  3.0  0.1  n
121  2.4  0.1  n

[122 rows x 14 columns]
Data after transforming:
```



Data after transforming:

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	\
0	1	6	2012	29	57	18	0.0	65.7	3.4	7.6	1.3	
1	2	6	2012	29	61	13	1.3	64.4	4.1	7.6	1.0	
2	3	6	2012	26	82	22	13.1	47.1	2.5	7.1	0.3	
3	4	6	2012	25	89	13	2.5	28.6	1.3	6.9	0.0	
4	5	6	2012	27	77	16	0.0	64.8	3.0	14.2	1.2	
..	...	...	...	...	...	...	...	...	...	...	...	
117	26	9	2012	31	54	11	0.0	82.0	6.0	16.3	2.5	
118	27	9	2012	31	66	11	0.0	85.7	8.3	24.9	4.0	
119	28	9	2012	32	47	14	0.7	77.5	7.1	8.8	1.8	
120	29	9	2012	26	80	16	1.8	47.4	2.9	7.7	0.3	
121	30	9	2012	25	78	14	1.4	45.0	1.9	7.5	0.2	

	BUI	FWI	Fire
0	3.4	0.5	3
1	3.9	0.4	2
2	2.7	0.1	2
3	1.7	0.0	2
4	3.9	0.5	2
..	...	...	...
117	6.2	1.7	2
118	9.0	4.1	5
119	6.8	0.9	2
120	3.0	0.1	2
121	2.4	0.1	2

[122 rows x 14 columns]

Data before scaling:

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	\
0	1	6	2012	29	57	18	0.0	65.7	3.4	7.6	1.3	
1	2	6	2012	29	61	13	1.3	64.4	4.1	7.6	1.0	
2	3	6	2012	26	82	22	13.1	47.1	2.5	7.1	0.3	
3	4	6	2012	25	89	13	2.5	28.6	1.3	6.9	0.0	
4	5	6	2012	27	77	16	0.0	64.8	3.0	14.2	1.2	
..	...	...	...	...	...	...	...	...	...	...	...	
117	26	9	2012	31	54	11	0.0	82.0	6.0	16.3	2.5	
118	27	9	2012	31	66	11	0.0	85.7	8.3	24.9	4.0	
119	28	9	2012	32	47	14	0.7	77.5	7.1	8.8	1.8	
120	29	9	2012	26	80	16	1.8	47.4	2.9	7.7	0.3	
121	30	9	2012	25	78	14	1.4	45.0	1.9	7.5	0.2	

```
     BUI  FWI  Fire
0    3.4  0.5     3
1    3.9  0.4     2
2    2.7  0.1     2
3    1.7  0.0     2
4    3.9  0.5     2
..   ...  ...   ...
117  6.2  1.7     2
118  9.0  4.1     5
119  6.8  0.9     2
120  3.0  0.1     2
121  2.4  0.1     2

[122 rows x 14 columns]
Data after scaling:
           day     month  year  Temperature        RH        Ws     Rain   \
0   -1.675278 -1.350526   0.0    -0.659354 -0.988010  0.704943 -0.351193
1   -1.561731 -1.350526   0.0    -0.659354 -0.627929 -1.057415  0.190628
2   -1.448185 -1.350526   0.0    -1.566585  1.262498  2.114830  5.108701
3   -1.334638 -1.350526   0.0    -1.868995  1.892641 -1.057415  0.690771
4   -1.221091 -1.350526   0.0    -1.264174  0.812397  0.000000 -0.351193
..        ...       ...   ...          ...       ...       ...       ...
117  1.163387  1.350526   0.0    -0.054533 -1.258071 -1.762358 -0.351193
118  1.276934  1.350526   0.0    -0.054533 -0.177827 -1.762358 -0.351193
119  1.390481  1.350526   0.0     0.247877 -1.888213 -0.704943 -0.059443
120  1.504027  1.350526   0.0    -1.566585  1.082458  0.000000  0.399021
121  1.617574  1.350526   0.0    -1.868995  0.902417 -0.704943  0.232307

         FFMC       DMC        DC       ISI       BUI       FWI  Fire
0   -0.579094 -0.793971 -0.883547 -0.782804 -0.834294 -0.803842     3
1   -0.662993 -0.731627 -0.883547 -0.882493 -0.799608 -0.819672     2
2   -1.779495 -0.874127 -0.893243 -1.115101 -0.882855 -0.867163     2
3   -2.973443 -0.981003 -0.897122 -1.214790 -0.952228 -0.882993     2
4   -0.637178 -0.829596 -0.755555 -0.816034 -0.799608 -0.803842     2
..        ...       ...       ...       ...       ...       ...   ...
117  0.472871 -0.562408 -0.714830 -0.384048 -0.640050 -0.613878     2
118  0.711661 -0.357564 -0.548052  0.114397 -0.445806 -0.233951     5
119  0.182451 -0.464440 -0.860276 -0.616656 -0.598426 -0.740520     2
120 -1.760134 -0.838502 -0.881608 -1.115101 -0.862043 -0.867163     2
121 -1.915024 -0.927565 -0.885486 -1.148330 -0.903667 -0.867163     2

[122 rows x 14 columns]
```

```
confusion matrix:
 [[19  1  0  4]
 [ 1  0  0  0]
 [ 0  0  0  1]
 [ 0  0  0 11]]
classification report:
              precision    recall  f1-score   support

         2.0       0.95      0.79      0.86        24
         3.0       0.00      0.00      0.00         1
         4.0       0.00      0.00      0.00         1
         5.0       0.69      1.00      0.81        11

    accuracy                           0.81        37
   macro avg       0.41      0.45      0.42        37
weighted avg       0.82      0.81      0.80        37

coefficient: [array([[-3.17711682e-01,  5.40779598e-01, -4.87083183e-01],
       [-2.69776243e+01,  6.86462229e+00, -8.26967398e+00],
       [-9.33014882e-04,  1.64317297e-04,  1.41351392e-03],
       [-1.11275990e+00,  2.77984009e+00,  3.84701840e+00],
       [-9.58059088e-01,  6.51914711e-01, -3.62143221e+00],
       [-5.20253752e+00,  3.63267282e+00, -8.96242298e+00],
       [ 8.84774304e+00,  5.98727658e+00, -4.27763121e+00],
       [-4.88624808e+00, -1.08025141e+01,  6.78688564e+00],
       [-3.37966116e+00, -4.17953834e+00,  5.00994545e+00],
       [-4.33926921e+00, -8.13652787e+00,  8.86406981e+00],
       [-4.94862122e+00, -1.23938925e+01,  9.81594501e+00],
       [-3.65127382e+00, -6.66367832e+00,  6.68565804e+00],
       [-4.28480131e+00, -1.10528143e+01,  8.42173640e+00]]), array([[ 4.38849036,  3.87863501, -0.07419192, -6.86474773, -5.59488782,
        -0.88262956],
       [ 0.81261202,  0.87592617,  4.3929074 , -8.14921923, -6.1845317 ,
        -0.04771804],
       [-6.47271712, -6.27691176,  1.59602455, -1.17460936,  3.86109355,
         4.37887794]])]
intercept:  [array([-8.8664454 , -4.87779514,  4.36182584]), array([-2.93988955, -3.22148587,  0.08026693,  4.68477994, -1.4797912 ,
        1.33405889])]
```

```
Weight Matrices:

Layer 1 Weights (Shape (13, 3)):
 [[ 1.30685802e+00  4.51909195e-01  1.88322850e+00]
 [ 1.19695426e-02 -3.07904006e+00 -2.73845174e+00]
 [ 3.71396699e-04 -2.35689464e-05 -1.51460811e-04]
 [ 5.22866345e+00  1.68973407e+00 -6.49953537e-01]
 [-1.16620184e+00 -1.40980107e+00 -2.64447858e+00]
 [ 7.01665247e-01 -3.56570040e+00 -4.19999071e+00]
 [ 1.85539292e+00 -3.14952283e+00  7.62618793e-01]
 [-6.37666244e+00  3.14356878e+00 -1.73596205e+00]
 [ 1.62531525e-01  1.59485309e+00 -1.85250012e+00]
 [-7.51610914e-01  5.19198577e+00 -3.14557630e+00]
 [-5.73085418e+00  3.54773147e+00 -2.20724899e+00]
 [-3.55643095e-01  2.46585649e+00 -1.85934788e+00]
 [-5.07214101e+00  3.45906420e+00 -2.02875890e+00]]

Layer 2 Weights (Shape (3, 6)):
 [[ 1.95239765  1.79171447  3.83426205 -3.6531391  -1.94842418 -2.81064578]
 [-3.4945323  -4.13237882 -1.7177979   0.20765469  1.36501794  3.09642555]
 [ 1.61928119  1.40825643  1.42634763 -4.20894784 -2.73635095  0.93208301]]
```

RESULT

Thus the program was executed successfully

NAIVE BAYES CLASSIFICATION - CUSTOMER BUYING HABIT PREDICTION

Ex.No.:

Date:

1. Categorical Naive Bayes

AIM

The aim of this project is to apply a Categorical Naive Bayes model to a dataset to predict whether a person buys a car or not based on various features.

LIBRARIES USED

- pandas: For data manipulation and reading the dataset.
- numpy: For array manipulation.
- sklearn.preprocessing.LabelEncoder: For transforming categorical features into numerical labels.
- sklearn.naive_bayes.CategoricalNB: For implementing the Categorical Naive Bayes classifier model, suitable for categorical features.
- sklearn.metrics.confusion_matrix: For evaluating the model's performance using a confusion matrix.
- sklearn.metrics.classification_report: For generating performance metrics such as precision, recall, and F1-score.

METHODS USED

- Label Encoding: Since the dataset may contain categorical variables (such as strings or labels), these features are converted into numerical values using `LabelEncoder`.
- Model Training: This version of Naive Bayes works well with categorical data and computes the probabilities based on the frequency of features within each class.
- Prediction: After training the model, predictions are made on the entire dataset, and the predicted values are compared to the actual values (target variable).
- Performance Evaluation:
- Confusion Matrix: This is a matrix used to evaluate the performance of the classification algorithm by showing the actual versus predicted values.
- Classification Report: This provides essential metrics such as precision, recall, and F1-score for each class in the dataset.

- Prediction on Custom Test Input: A custom test input is provided to the model, and the model predicts the class label for that input.

PROCEDURE

1. Data Preprocessing:
    a. Load the dataset using `pandas.read_excel()` and inspect the data.
    b. Identify categorical columns using `select_dtypes(include="object")`.
    c. Encode the categorical columns into numerical values using `LabelEncoder` from `sklearn.preprocessing`.

2. Splitting Data into Features and Target:
    a. Extract the feature columns (`X`) and target column (`y`). The feature columns are all the columns except the last one, and the target column is the last one.

3. Model Training:
    a. Initialize the Categorical Naive Bayes classifier using `CategoricalNB()`.
    b. Fit the model on the features (`X`) and the target variable (`y`) using `model.fit(X, y)`.

4. Making Predictions:
    a. Use `model.predict(X)` to generate predictions for the training data.
    b. Print the predicted output alongside the actual values for comparison.

5. Model Evaluation:
    a. Confusion Matrix: Evaluate the model's performance using the confusion matrix (`confusion_matrix(y, ypred)`).
    b. Classification Report: Generate the classification report (`classification_report(y, ypred)`) to evaluate the model's accuracy in terms of precision, recall, and F1-score.

6. Testing the Model:
    a. Provide a custom test input (an array) and use `model.predict()` to predict the class label for that test input. Print the result.

CODE

```
import pandas as pd

import numpy as np

from sklearn import preprocessing
```

```
from sklearn.naive_bayes import CategoricalNB
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
 df=pd.read_excel('trainingDataNaiveBayes.xlsx')
objlist=df.select_dtypes(include="object").columns
print(objlist)
print(df)
 len_encoder=preprocessing.LabelEncoder()
for col in objlist:
    df[col]=len_encoder.fit_transform(df[col].astype(str))
print(df)
 x=df.values[:,:-1]
y=df.values[:,-1]
model=CategoricalNB()
model.fit(x,y)
ypred=model.predict(x)
print('Actual Output(buys_car):',y)
print('Predicted Output(buys_car):',ypred)
cm=confusion_matrix(y,ypred)
print('Confusion Matrix\n',cm)
print('Classification report\n',classification_report(y,ypred))
test=np.array([2,0,0,1])
testoutput=model.predict([test])
print('Input\n',test)
print('Class label\n',testoutput)
```

OUTPUT

```
Index(['Age', 'Income', 'Marital Status', 'Credit rating', 'buys_car'], dtype='object')
           Age  Income Marital Status Credit rating buys_car
0        Youth    High             No          Fair       No
1        Youth    High             No     Excellent       No
2   Middle Aged    High             No          Fair      Yes
3       Senior  Medium             No          Fair      Yes
4       Senior     Low            Yes          Fair      Yes
5       Senior     Low            Yes     Excellent       No
6   Middle Aged     Low            Yes     Excellent      Yes
7        Youth  Medium             No          Fair       No
8        Youth     Low            Yes          Fair      Yes
9       Senior  Medium            Yes          Fair      Yes
10       Youth  Medium            Yes     Excellent      Yes
11  Middle Aged  Medium             No     Excellent      Yes
12  Middle Aged    High            Yes          Fair      Yes
13      Senior  Medium             No     Excellent       No
    Age  Income  Marital Status  Credit rating  buys_car
0     2       0               0              1         0
1     2       0               0              0         0
2     0       0               0              1         1
3     1       2               0              1         1
4     1       1               1              1         1
5     1       1               1              0         0
6     0       1               1              0         1
7     2       2               0              1         0
8     2       1               1              1         1
9     1       2               1              1         1
10    2       2               1              0         1
11    0       2               0              0         1
12    0       0               1              1         1
13    1       2               0              0         0
Actual Output(buys_car): [0 0 1 1 1 0 1 0 1 1 1 1 1 0]
Predicted Output(buys_car): [0 0 1 1 1 1 1 0 1 1 1 1 1 0]
Confusion Matrix
 [[4 1]
 [0 9]]
Classification report
              precision    recall  f1-score   support

           0       1.00      0.80      0.89         5
           1       0.90      1.00      0.95         9

    accuracy                           0.93        14
   macro avg       0.95      0.90      0.92        14
weighted avg       0.94      0.93      0.93        14

Input
 [2 0 0 1]
Class label
 [0]
```

RESULT

Thus, the above categorical Naive Bayes classification is implemented successfully.

2. Gaussian Naive Bayes

AIM

The aim of this project is to apply a Gaussian Naive Bayes (GaussianNB) model to a dataset (forestFire.csv) to predict a target variable (possibly related to fire occurrence or fire intensity) based on various features.

LIBRARIES USED

1. pandas
2. sklearn.naive_bayes.GaussianNB
3. Sklearn.preprocessing.LabelEncoder
4. sklearn.metrics.classification_report
   Sklearn.metrics.confusion_ matrix
5. Numpy

METHODS USED

1. Label Encoding: Since the dataset may contain categorical features (e.g., string labels), these features are transformed into numerical values using LabelEncoder.
2. Model Training: The Gaussian Naive Bayes (GaussianNB) classifier is used to train the model.
3. Prediction: After training the model, predictions are made on the training data (
4. Performance Evaluation: After predicting the target values, the model's performance is evaluated using:
    a. Confusion Matrix: A table that is used to evaluate the performance of a classification algorithm.
    b. Classification Report: Provides important metrics such as precision, recall, and F1-score to assess model accuracy.
5. Prediction on Custom Test Input: A custom input sample is tested to predict the class label based on the trained model.

PROCEDURE

1. Data Preprocessing:
    a. Load the dataset using pandas.read_csv().
    b. Identify categorical columns using select_dtypes(include="object").
    c. Use LabelEncoder from sklearn.preprocessing to encode categorical features into numerical values. This is needed as Naive Bayes models, especially GaussianNB, require numerical inputs.

2. Splitting Data into Features and Target:
    a. Split the data into feature variables (X) and target variable (y), where X contains all columns except the last one, and y contains the last column (the target class label).
3. Model Training:
    a. Initialize a Gaussian Naive Bayes classifier using GaussianNB().
    b. Train the model using model.fit(X, y) where X is the features and y is the target.
4. Making Predictions:
    a. Use model.predict(X) to generate predictions on the training data.
    b. Print the predicted and actual outputs for comparison.
5. Model Evaluation:
    a. Generate the Confusion Matrix using confusion_matrix(y, ypred) to check the model's performance.
    b. Generate the Classification Report using classification_report(y, ypred) to calculate precision, recall, and F1-score.
6. Testing the Model:
    a. A custom test input is reshaped and passed through the trained model to predict the class label. The prediction is printed.

CODE

```
import pandas as pd
from sklearn.naive_bayes import GaussianNB
from sklearn import preprocessing
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
import numpy as np

data = pd.read_csv('forestFire.csv')
objlist = data.select_dtypes(include="object").columns
print('Categorical columns:\n', objlist)
print(data.head(10))

len_encoder = preprocessing.LabelEncoder()
for col in objlist:
   data[col] = len_encoder.fit_transform(data[col].astype(str))

print(data)

x = data.values[:, :-1]
y = data.values[:, -1]
```

```
model = GaussianNB()
model.fit(x, y)
ypred = model.predict(x)

print('Actual Output(fire):', y)
print('Predicted Output(fire):', ypred)

cm = confusion_matrix(y, ypred)
print('Confusion Matrix\n', cm)
print('Classification report\n', classification_report(y, ypred))

test = np.array([[1, 6, 2012, 29.57, 18.0, 0.65, 7.3, 4.7, 6.1, 1.3, 3.4, 0.5]])
testoutput = model.predict(test)
print('Input', test)
print('Class label', testoutput)
```

OUTPUT

```
[122 rows x 14 columns]
Actual Output(fire): [1. 0. 0. 0. 1. 1. 1. 1. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 1.
 1. 1. 1. 1. 0. 1. 0. 0. 0. 0. 1. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 1.
 1. 1. 1. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 1. 1. 1. 1. 0. 1. 1. 1.
 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0.
 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 1.
 0. 0.]
Predicted Output(fire): [0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1.
 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 1. 1. 1. 1. 0. 0. 0. 1. 0. 0. 0. 1. 1.
 1. 1. 1. 1. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 1. 1. 1. 1. 0. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 1. 0.
 0. 0.]
Confusion Matrix
 [[53  7]
 [ 7 55]]
Classification report
              precision    recall  f1-score   support

         0.0       0.88      0.88      0.88        60
         1.0       0.89      0.89      0.89        62

    accuracy                           0.89       122
   macro avg       0.89      0.89      0.89       122
weighted avg       0.89      0.89      0.89       122

Input [1.000e+00 6.000e+00 2.012e+03 2.900e+01 5.700e+01 1.800e+01 0.000e+00
 6.570e+01 3.400e+00 7.600e+00 1.300e+00 3.400e+00 5.000e-01]
Class label [0.]
```

```
Categorical columns:
 Index(['Fire'], dtype='object')
   day  month  year  Temperature   RH   Ws  Rain   FFMC   DMC    DC  ISI  \
0    1      6  2012           29   57   18   0.0  65.7   3.4   7.6  1.3
1    2      6  2012           29   61   13   1.3  64.4   4.1   7.6  1.0
2    3      6  2012           26   82   22  13.1  47.1   2.5   7.1  0.3
3    4      6  2012           25   89   13   2.5  28.6   1.3   6.9  0.0
4    5      6  2012           27   77   16   0.0  64.8   3.0  14.2  1.2
5    6      6  2012           31   67   14   0.0  82.6   5.8  22.2  3.1
6    7      6  2012           33   54   13   0.0  88.2   9.9  30.5  6.4
7    8      6  2012           30   73   15   0.0  86.6  12.1  38.3  5.6
8    9      6  2012           25   88   13   0.2  52.9   7.9  38.8  0.4
9   10      6  2012           28   79   12   0.0  73.2   9.5  46.3  1.3

    BUI  FWI Fire
0   3.4  0.5    y
1   3.9  0.4    n
2   2.7  0.1    n
3   1.7  0.0    n
4   3.9  0.5    y
5   7.0  2.5    y
6  10.9  7.2    y
7  13.5  7.1    y
8  10.5  0.3    n
9  12.6  0.9    n
     day  month  year  Temperature   RH   Ws  Rain   FFMC  DMC    DC  ISI  \
0      1      6  2012           29   57   18   0.0  65.7  3.4   7.6  1.3
1      2      6  2012           29   61   13   1.3  64.4  4.1   7.6  1.0
2      3      6  2012           26   82   22  13.1  47.1  2.5   7.1  0.3
3      4      6  2012           25   89   13   2.5  28.6  1.3   6.9  0.0
4      5      6  2012           27   77   16   0.0  64.8  3.0  14.2  1.2
..   ...    ...   ...          ...  ...  ...   ...   ...  ...   ...  ...
117   26      9  2012           31   54   11   0.0  82.0  6.0  16.3  2.5
118   27      9  2012           31   66   11   0.0  85.7  8.3  24.9  4.0
119   28      9  2012           32   47   14   0.7  77.5  7.1   8.8  1.8
120   29      9  2012           26   80   16   1.8  47.4  2.9   7.7  0.3
121   30      9  2012           25   78   14   1.4  45.0  1.9   7.5  0.2

    BUI  FWI  Fire
0   3.4  0.5     1
1   3.9  0.4     0
2   2.7  0.1     0
3   1.7  0.0     0
4   3.9  0.5     1
..  ...  ...   ...
117 6.2  1.7     0
118 9.0  4.1     0
119 6.8  0.9     1
120 3.0  0.1     0
121 2.4  0.1     0
```

RESULT

Thus, the above Gaussian naive bayes classification is implemented successfully.

DECISION TREE

Ex.No.:

Date:

AIM

To build a **Decision Tree Classifier** using the **ID3 algorithm** to predict whether to **diabetes** based on given weather conditions.

DATASET DESCRIPTION

This dataset is a small categorical dataset used to determine whether to play tennis based on weather conditions. It consists of **14 instances (days) and 5 features**, including the target variable **Play** (Yes/No).

Attributes

1. **Day** – Identifier for the day (D1 to D14). Not used as a predictor.
2. **Outlook** – The weather condition:
    a. **Sunny**
    b. **Overcast**
    c. **Rain**
3. **Temperature** – The temperature condition:
    a. **Hot**
    b. **Mild**
    c. **Cool**
4. **Humidity** – The humidity level:
    a. **High**
    b. **Normal**
5. **Wind** – Wind strength:
    a. **Weak**
    b. **Strong**
6. **Play (Target Variable)** – Whether to play tennis:
    a. **Yes (1)**
    b. **No (0)**

Dataset Characteristics

Categorical Data: All features are categorical, requiring **Label Encoding** before training a model.

- **Decision-Based Data**: This dataset is useful for **Decision Tree** algorithms since it involves rule-based classification.
- **Entropy-Based Splitting**: The ID3 algorithm calculates **information gain** using entropy to determine feature importance.
- **Small and Balanced**: There are 14 samples, with both "Yes" and "No" decisions fairly distributed.

METHODS USED

Data Preprocessing

Read dataset using `pandas` (`pd.read_csv`).
Separated features (`X`) and target (`y`).
Encoded categorical target variable using `LabelEncoder`.
Encoded categorical features using `LabelEncoder` for numerical representation.

Data Splitting

Split dataset into training (80%) and testing (20%) using `train_test_split`.

Model Training

Trained a Decision Tree Classifier (`DecisionTreeClassifier`) using the **ID3 algorithm** (`criterion='entropy'`).

Model Evaluation

Predicted on test data using `clf.predict`.
Measured **accuracy** using `accuracy_score`.
Generated a **confusion matrix** using `confusion_matrix` and visualized it using `seaborn`.
Generated a **classification report** (`precision, recall, f1-score`) using `classification_report`.

Decision Tree Visualization

Plotted the trained Decision Tree using `plot_tree` from `sklearn.tree`.

LIBRARIES USED

- **pandas** – Data handling (`read_csv`, `drop`, `apply`).
- **numpy** – Numerical operations (though not directly used).
- **matplotlib.pyplot** – Plotting graphs (`plt.show()`).
- **seaborn** – Visualizing confusion matrix (`sns.heatmap`).
- **sklearn.model_selection** – Splitting dataset (`train_test_split`).
- **sklearn.tree** – Decision Tree (`DecisionTreeClassifier`, `plot_tree`).
- **sklearn.metrics** – Accuracy, confusion matrix, and classification report.
- **scipy.stats** – Entropy calculation (not directly used).
- **sklearn.preprocessing** – Label Encoding (`LabelEncoder`).

PROCEDURE

Load Dataset: Import libraries and load `diabetes.csv`.
Preprocess Data: Encode categorical features and target variable.
Split Data: Divide into 80% training and 20% testing sets.
Train Model: Use `DecisionTreeClassifier(criterion='entropy')` and fit on training data.
Make Predictions: Predict on test data using `clf.predict(X_test)`.
Evaluate Model: Compute accuracy, confusion matrix, and classification report.
Visualize Results: Plot confusion matrix and decision tree.
Conclusion: Analyze results and suggest improvements (e.g., hyperparameter tuning).

CODE

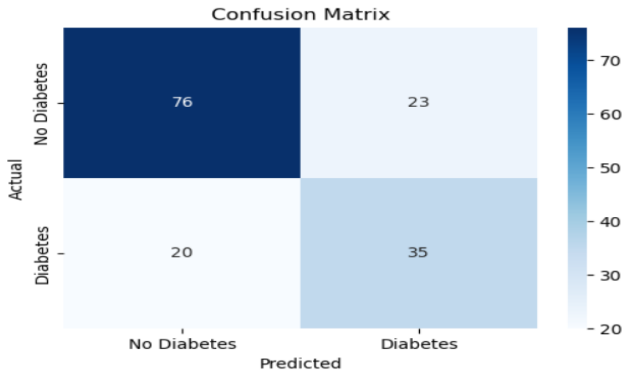
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

```
from scipy.stats import entropy

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

X = df.drop(columns=['Outcome'])
y = df['Outcome']


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


def calculate_entropy(y):
    class_counts = np.bincount(y)
    probabilities = class_counts / len(y)
    return entropy(probabilities, base=2)

root_entropy = calculate_entropy(y_train)

clf = DecisionTreeClassifier(criterion='entropy', random_state=42)
clf.fit(X_train, y_train)


feature_index = clf.tree_.feature[0]
split_feature = X.columns[feature_index]
threshold = clf.tree_.threshold[0]

left_mask = X_train.iloc[:, feature_index] <= threshold
right_mask = ~left_mask

left_entropy = calculate_entropy(y_train[left_mask])
right_entropy = calculate_entropy(y_train[right_mask])

n_total = len(y_train)
n_left = sum(left_mask)
n_right = sum(right_mask)
```

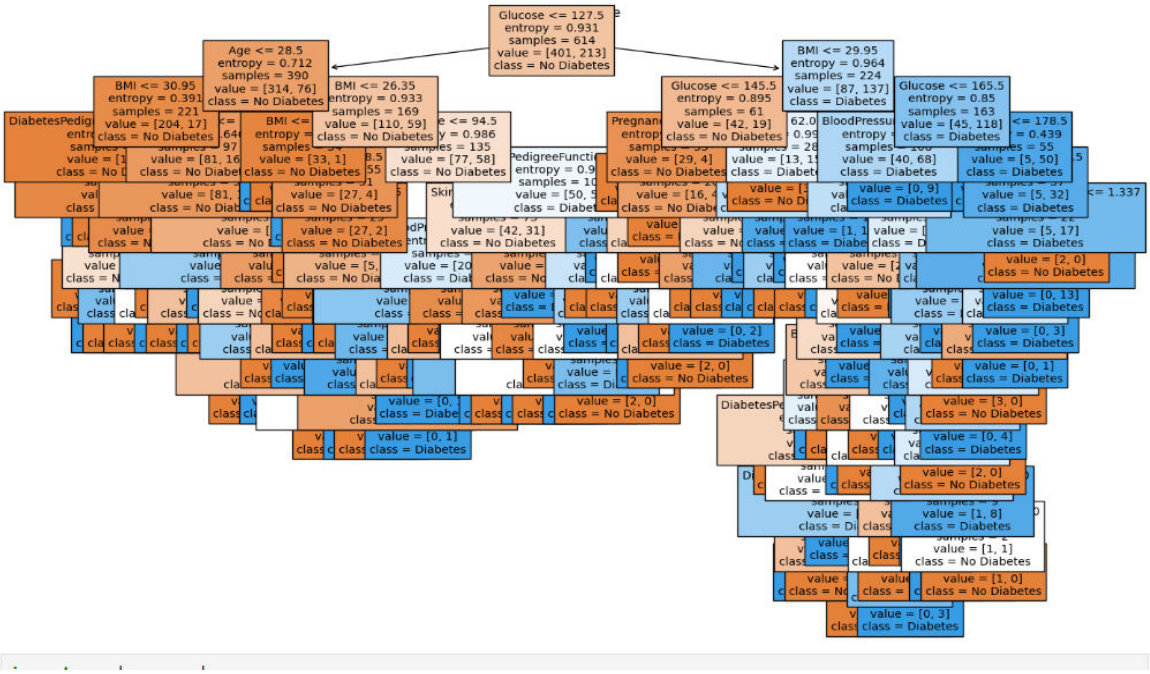
```
weighted_entropy_after_split = (n_left / n_total) * left_entropy + (n_right / n_total) *
right_entropy

info_gain = root_entropy - weighted_entropy_after_split

print(f"Entropy before split (Root Node Entropy): {root_entropy:.4f}")
print(f"Entropy of Left Child: {left_entropy:.4f}")
print(f"Entropy of Right Child: {right_entropy:.4f}")
print(f"Weighted Entropy after Split: {weighted_entropy_after_split:.4f}")
print(f"Information Gain for feature '{split_feature}': {info_gain:.4f}")

y_pred = clf.predict(X_test)

print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")

cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)

plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['No Diabetes', 'Diabetes'],
yticklabels=['No Diabetes', 'Diabetes'])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()

print("Classification Report:")
print(classification_report(y_test, y_pred))

plt.figure(figsize=(15, 10))
plot_tree(clf, feature_names=X.columns.tolist(), class_names=['No Diabetes', 'Diabetes'],
filled=True, fontsize=10)
plt.title("Decision Tree")
plt.show()
```

OUTPUT

```
Entropy before split (Root Node Entropy): 0.9313
Entropy of Left Child: 0.7116
Entropy of Right Child: 0.9638
Weighted Entropy after Split: 0.8036
Information Gain for feature 'Glucose': 0.1277
Accuracy: 0.7208
Confusion Matrix:
[[76 23]
 [20 35]]
```




```
Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.77      0.78        99
           1       0.60      0.64      0.62        55

    accuracy                           0.72       154
   macro avg       0.70      0.70      0.70       154
weighted avg       0.72      0.72      0.72       154
```



RESULT

Thus, the above program was implemented successfully.

TOOL STUDY KNIME

Ex.No.:

Date:

AIM

To study and implement Machine learning models in knime

KNIME

- KNIME (Konstanz Information Miner) is an open-source platform for data analytics, reporting, and integration.
- Enables users to create data flows (pipelines), execute analysis, and model data without deep programming skills.
- Supports visual programming with an intuitive drag-and-drop interface.

KNIME FEATURES

- **Data Integration**: Connects to databases, flat files, and cloud services.
- **Data Preprocessing**: Cleaning, filtering, and transforming data.
- **Analytics**: Machine learning, statistical, and deep learning algorithms.
- **Reporting & Visualization**: Data visualization and interactive reports.
- **Extension Support**: Community-contributed extensions for specialized tasks.

ADVANTAGES OF KNIME

- **Open Source**: Free to use.
- **User-Friendly**: Intuitive interface.
- **Flexibility**: Integrates with Python, R, Java.
- **Scalability**: Supports enterprise-level analysis.
- **Extensibility**: Thousands of extensions.
- **Strong Community**: Active support and documentation.

KNIME VS SIMILAR TOOLS

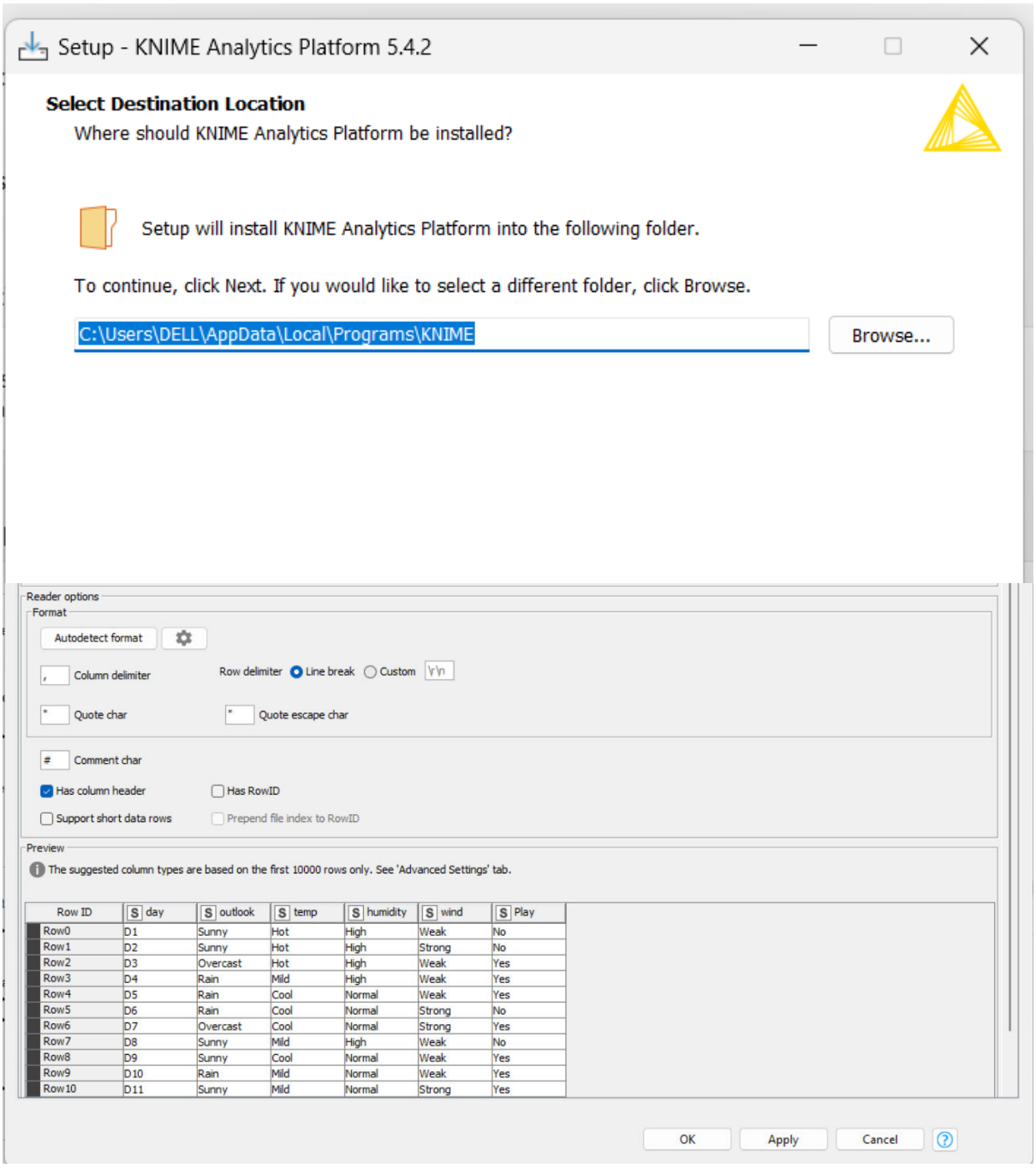
- **KNIME vs. RapidMiner**: KNIME offers more control, while RapidMiner is more polished for business users.
- **KNIME vs. SAS**: KNIME is free; SAS is commercial and used for statistical analysis.
- **KNIME vs. Python/R**: KNIME is no-code; Python/R offers more customization.

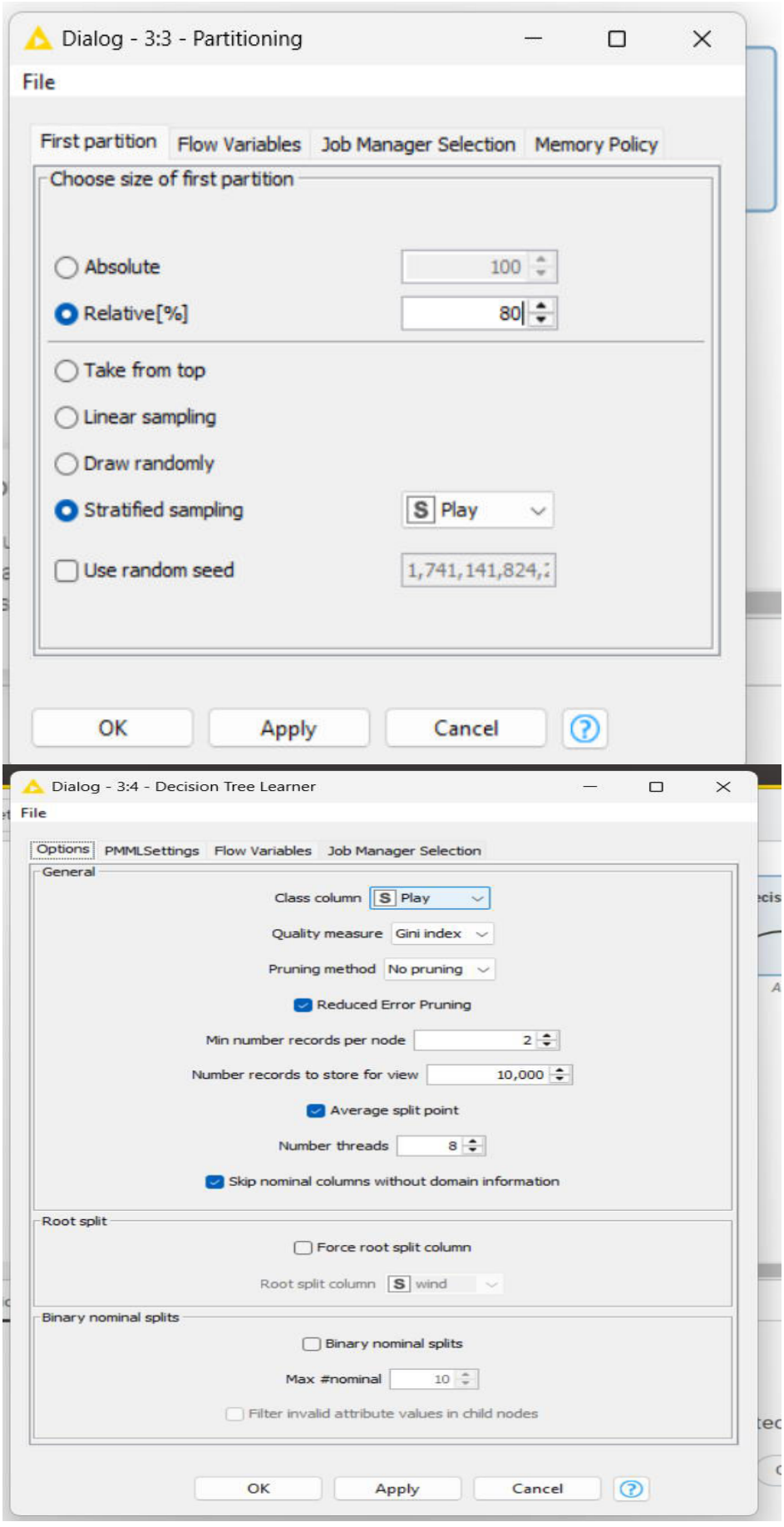
CHALLENGES AND LIMITATION

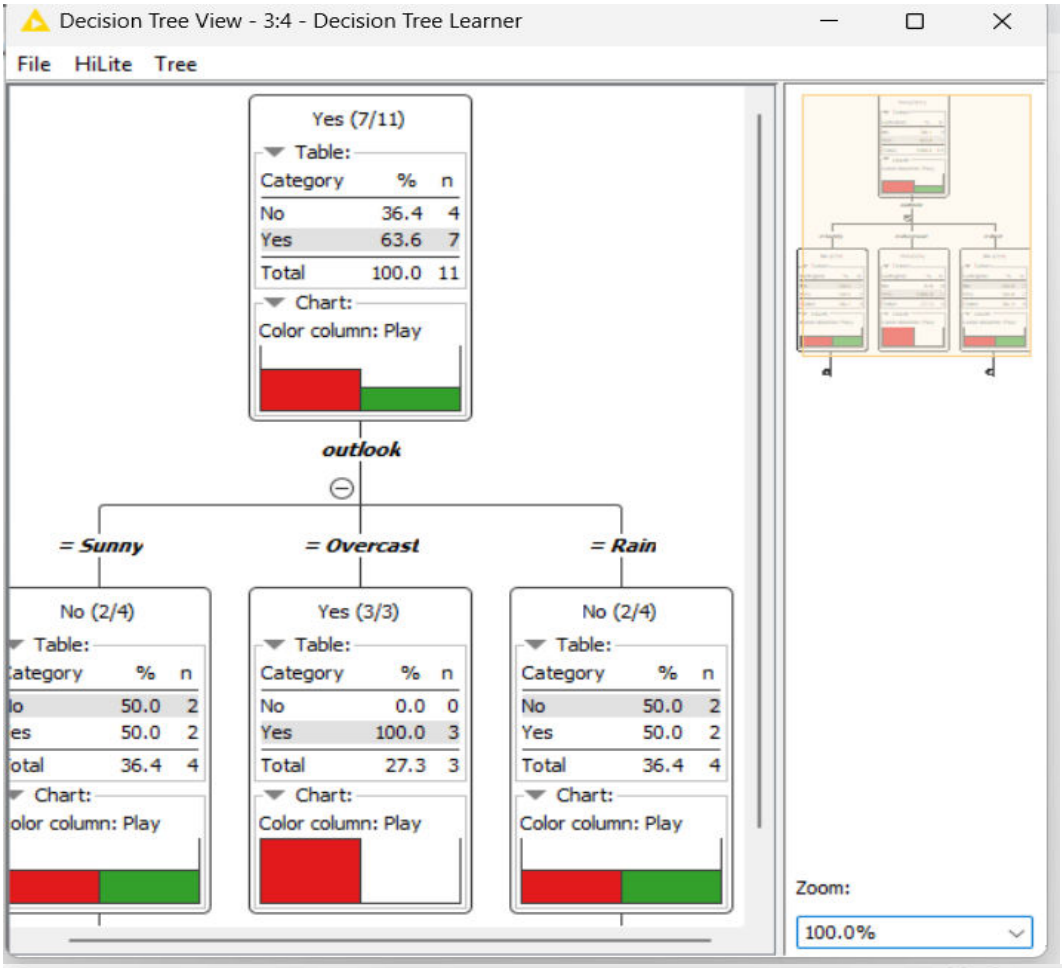
- **Learning Curve: Large number of nodes and options can be overwhelming.**
- **Performance: Can be slow with large datasets.**

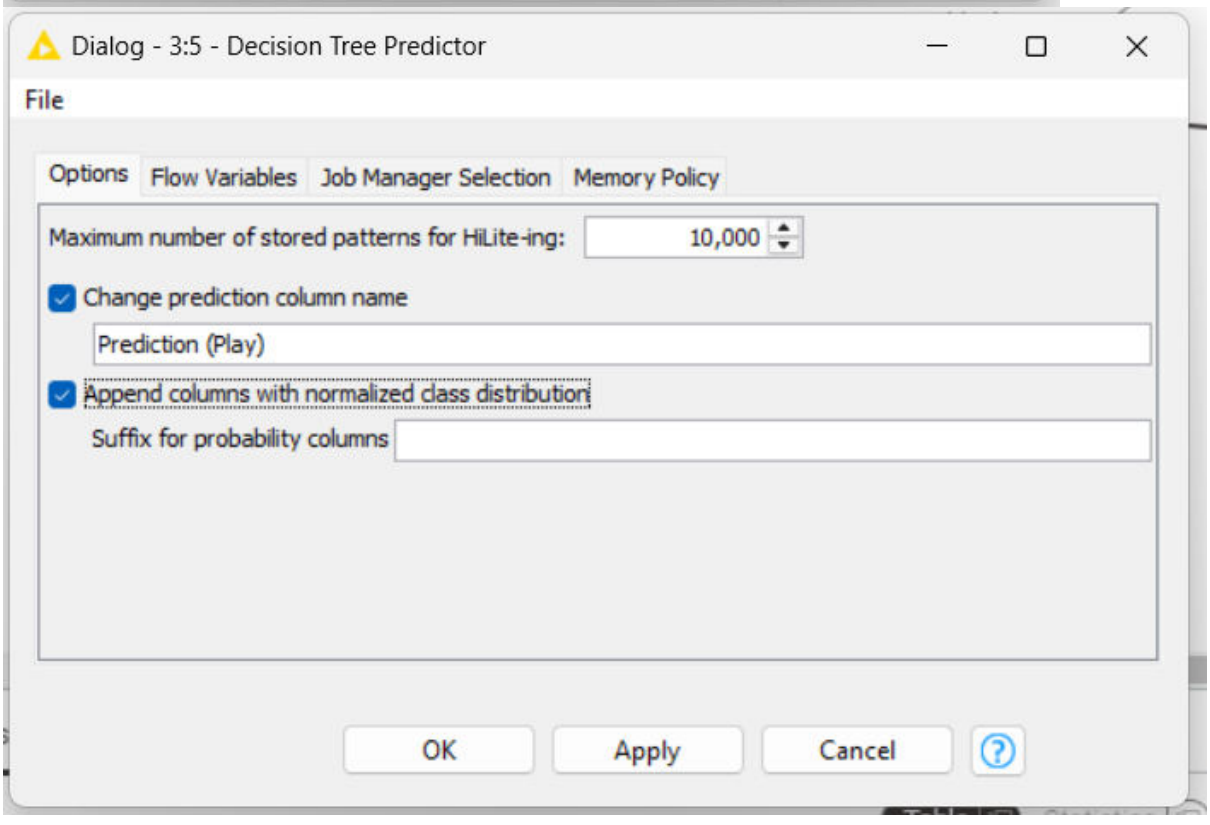
- **Dependency on Extensions: Some advanced features require additional plugins.**

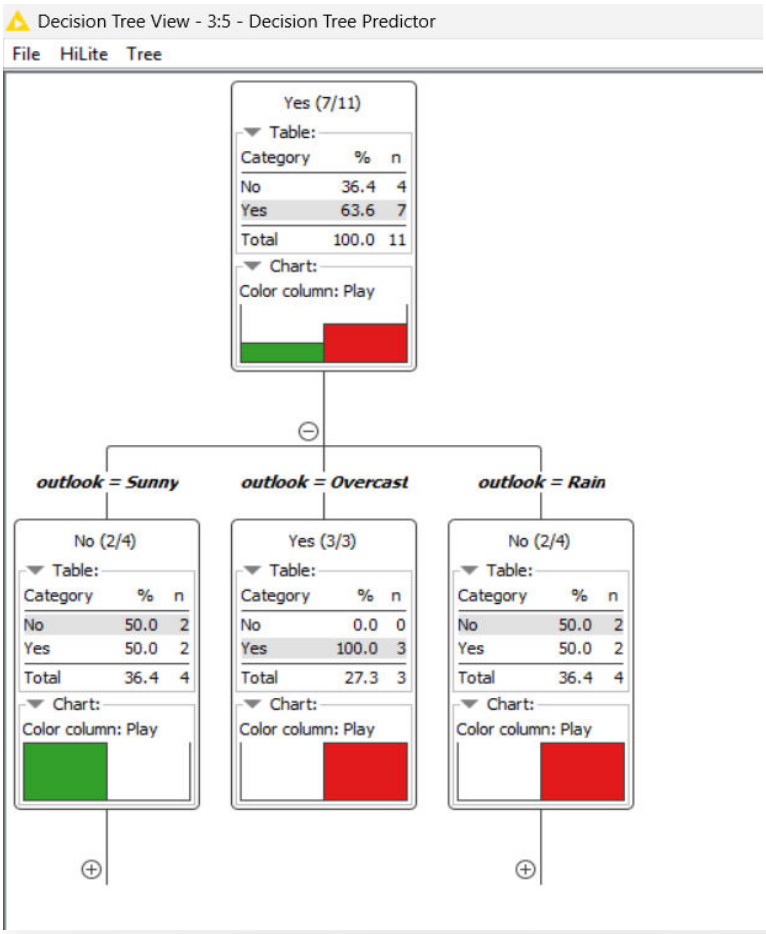
OUTPUT

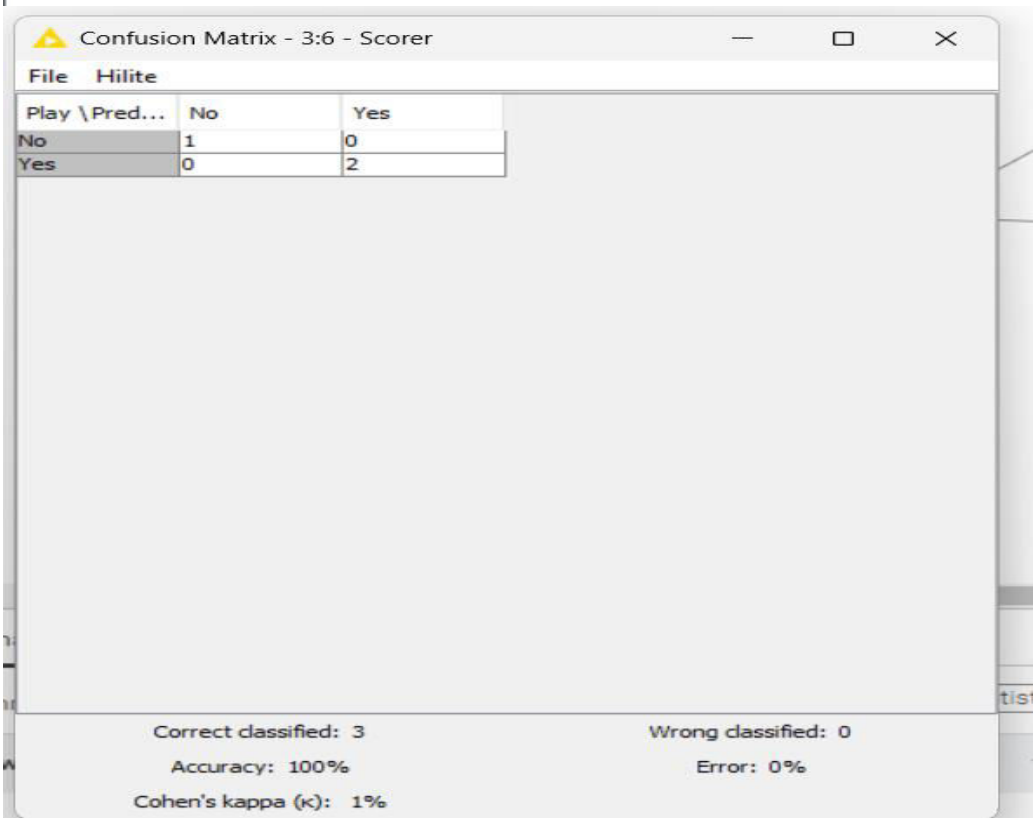


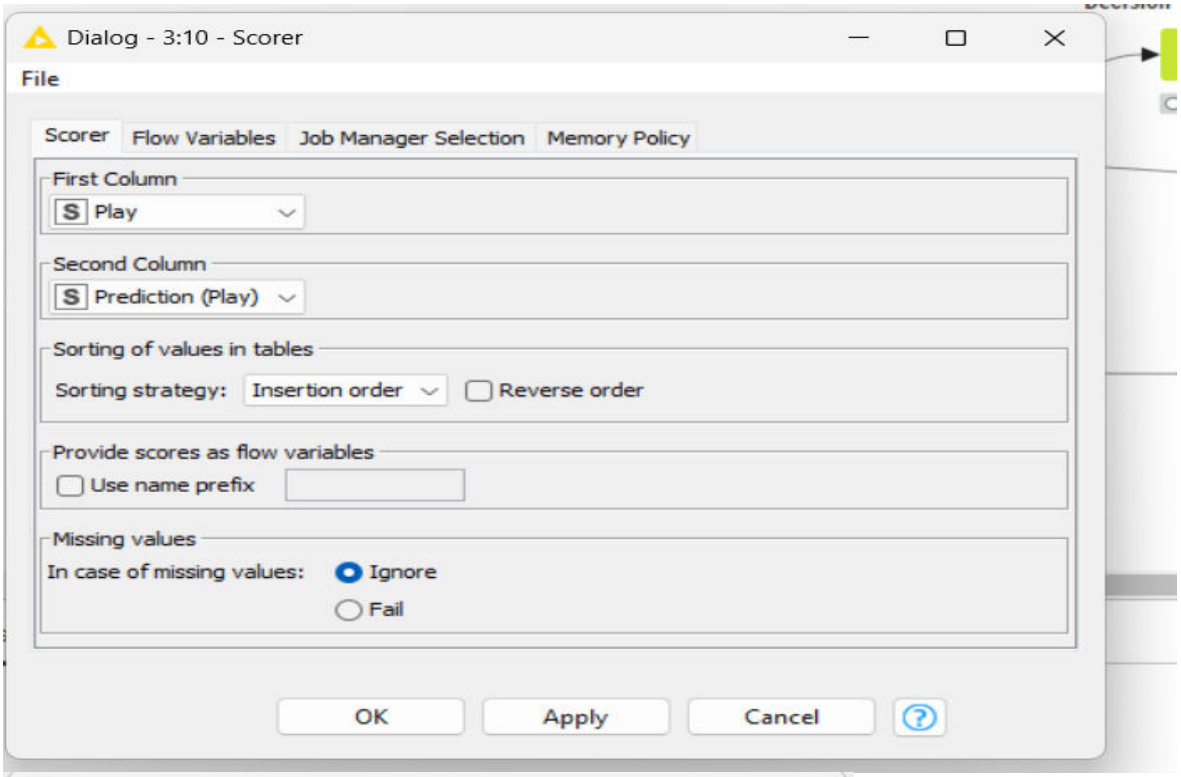

Dialog - 3:3 - Partitioning
File
First partition
Flow Variables
Job Manager Selection
Memory Policy
Choose size of first partition
Absolute
100
Relative[%]
80
Take from top
Linear sampling
Draw randomly
Stratified sampling
Play
Use random seed
1,741,141,824,
OK
Apply
Cancel
Dialog - 3:4 - Decision Tree Learner
File
Options
PMMLSettings
Flow Variables
Job Manager Selection
General
Class column
Play
Quality measure
Gini index
Pruning method
No pruning
Reduced Error Pruning
Min number records per node
2
Number records to store for view
10,000
Average split point
Number threads
8
Skip nominal columns without domain information
Root split
Force root split column
Root split column
wind
Binary nominal splits
Binary nominal splits
Max #nominal
10
Filter invalid attribute values in child nodes
OK
Apply
Cancel

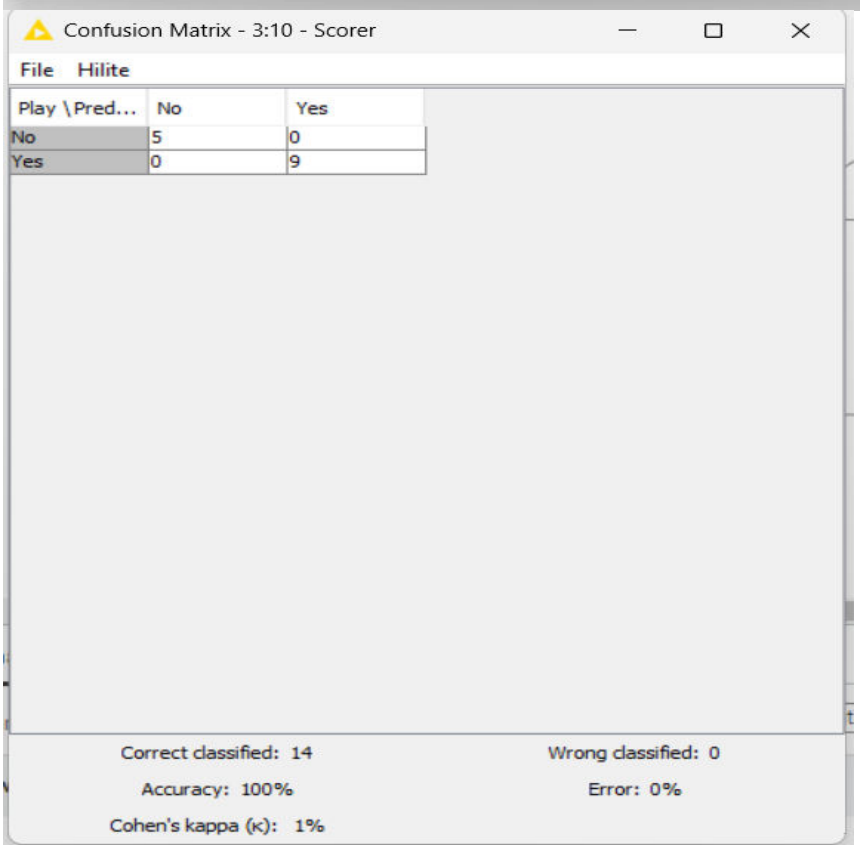

Decision Tree View - 3:4 - Decision Tree Learner
File HiLite Tree
Yes (7/11)
Table:
Category % n
No 36.4 4
Yes 63.6 7
Total 100.0 11
Chart:
Color column: Play
outlook
= Sunny
= Overcast
= Rain
No (2/4)
Table:
Category % n
No 50.0 2
Yes 50.0 2
Total 36.4 4
Chart:
Color column: Play
Yes (3/3)
Table:
Category % n
No 0.0 0
Yes 100.0 3
Total 27.3 3
Chart:
Color column: Play
No (2/4)
Table:
Category % n
No 50.0 2
Yes 50.0 2
Total 36.4 4
Chart:
Color column: Play
Zoom:
100.0%

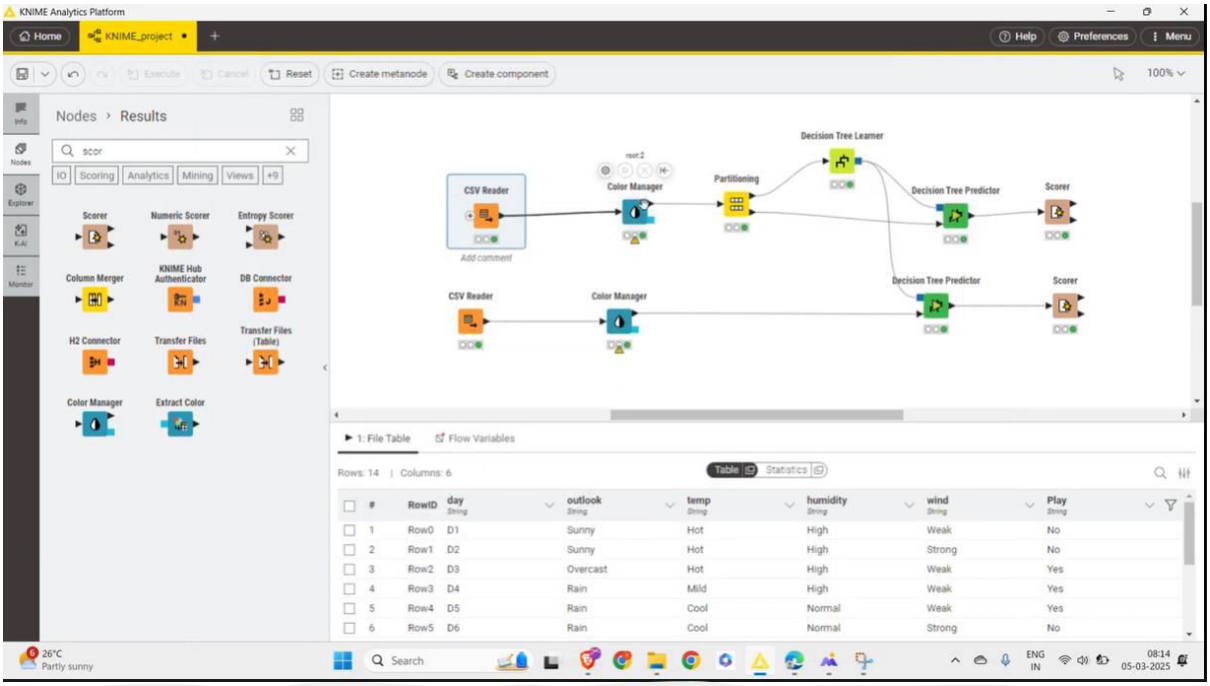

Dialog - 3:5 - Decision Tree Predictor
File
Options
Flow Variables
Job Manager Selection
Memory Policy
Maximum number of stored patterns for HiLite-ing:
10,000
Change prediction column name
Prediction (Play)
Append columns with normalized class distribution
Suffix for probability columns
OK
Apply
Cancel


Decision Tree View - 3:5 - Decision Tree Predictor
File HiLite Tree
Yes (7/11)
Table:
Category % n
No 36.4 4
Yes 63.6 7
Total 100.0 11
Chart:
Color column: Play
outlook = Sunny
outlook = Overcast
outlook = Rain
No (2/4)
Table:
Category % n
No 50.0 2
Yes 50.0 2
Total 36.4 4
Chart:
Color column: Play
Yes (3/3)
Table:
Category % n
No 0.0 0
Yes 100.0 3
Total 27.3 3
Chart:
Color column: Play
No (2/4)
Table:
Category % n
No 50.0 2
Yes 50.0 2
Total 36.4 4
Chart:
Color column: Play


Confusion Matrix - 3:6 - Scorer
File Hilite
Play \ Pred... No Yes
No 1 0
Yes 0 2
Correct classified: 3
Wrong classified: 0
Accuracy: 100%
Error: 0%
Cohen's kappa (κ): 1%


Dialog - 3:10 - Scorer
File
Scorer
Flow Variables
Job Manager Selection
Memory Policy
First Column
Play
Second Column
Prediction (Play)
Sorting of values in tables
Sorting strategy:
Insertion order
Reverse order
Provide scores as flow variables
Use name prefix
Missing values
In case of missing values:
Ignore
Fail
OK
Apply
Cancel


Confusion Matrix - 3:10 - Scorer
File
Hilite
Play \Pred...
No
Yes
No
5
0
Yes
0
9
Correct classified: 14
Wrong classified: 0
Accuracy: 100%
Error: 0%
Cohen's kappa (κ): 1%



Reader options

Format

Autodetect format

Column delimiter ,

Row delimiter Line break Custom \r\n

Quote char "

Quote escape char "

Comment char #

Has column header

Has RowID

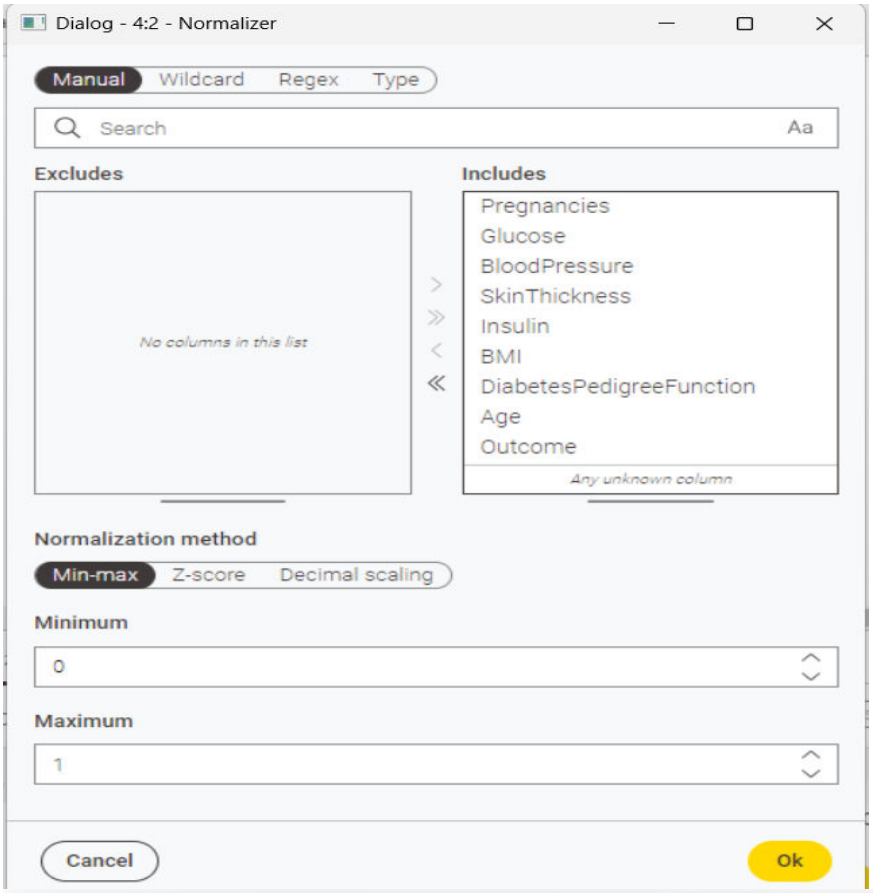
Support short data rows

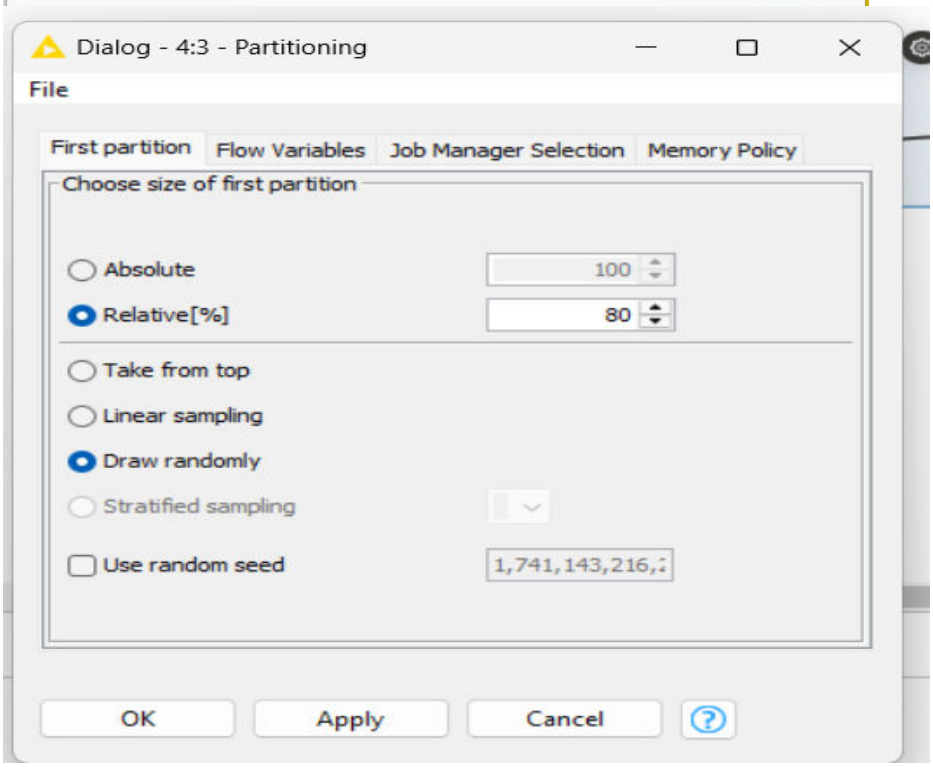
Prepend file index to RowID

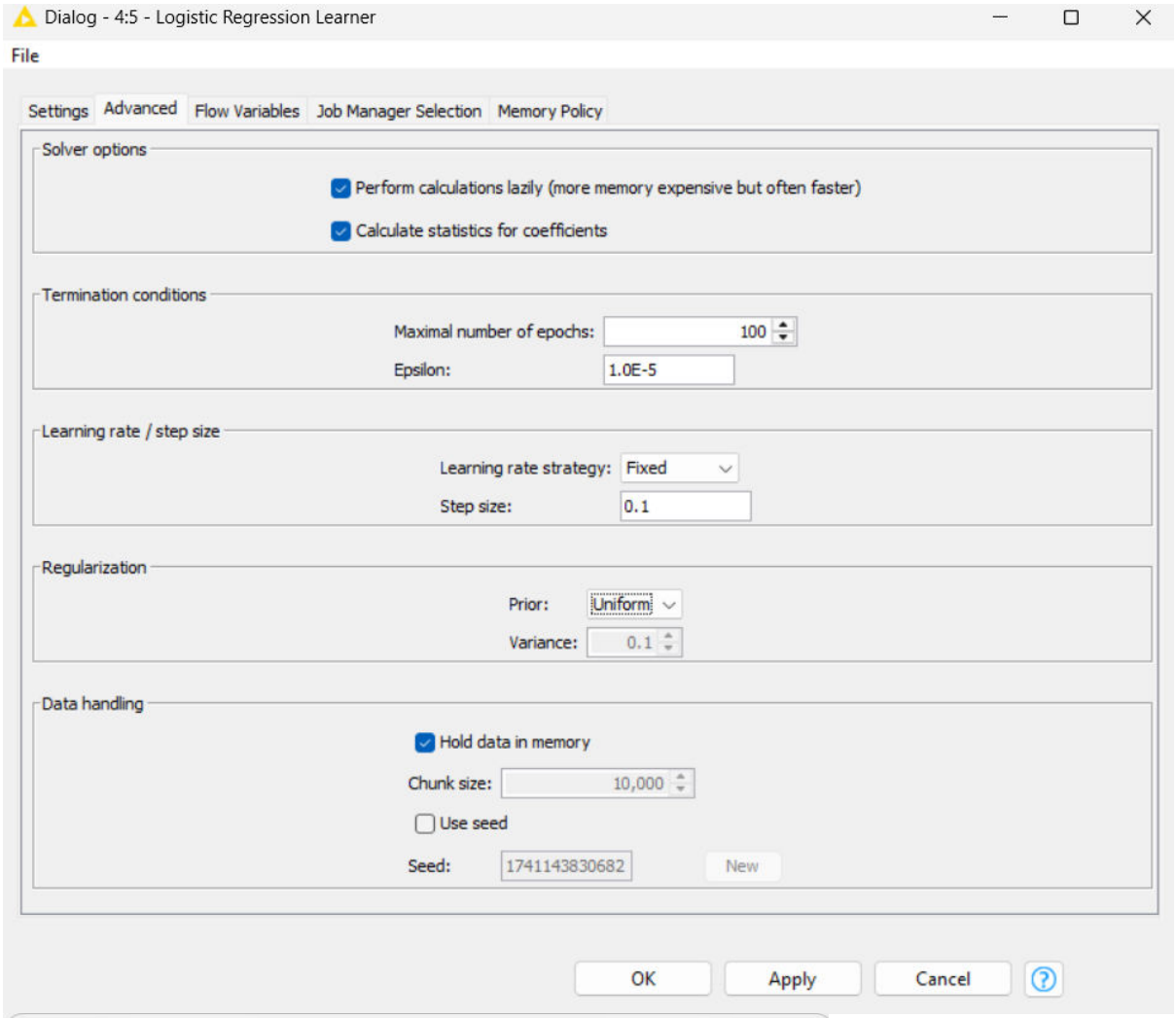
Preview

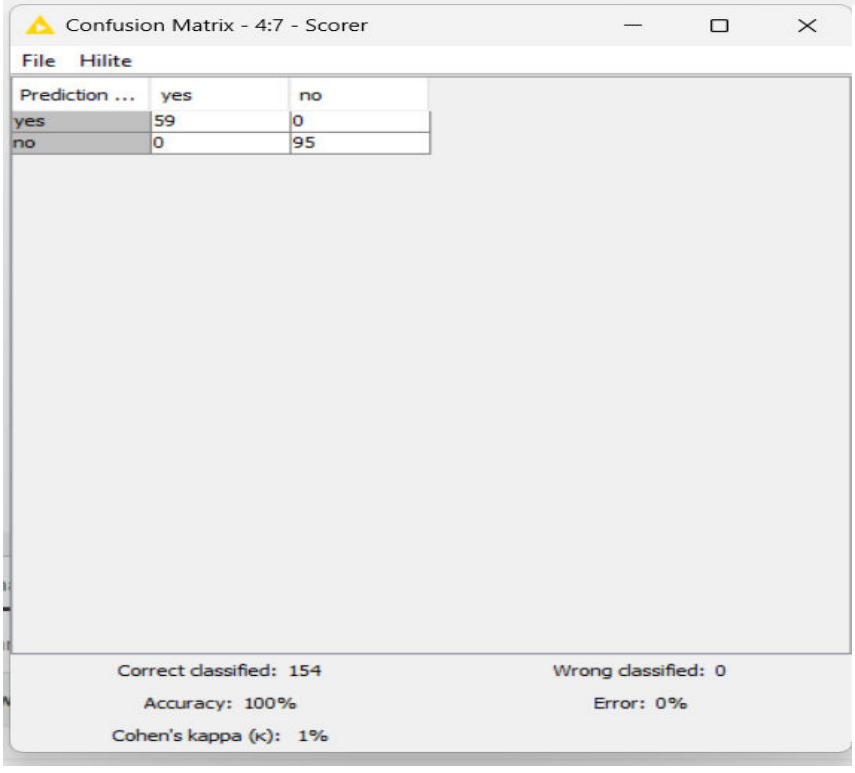
The suggested column types are based on the first 10000 rows only. See 'Advanced Settings' tab.

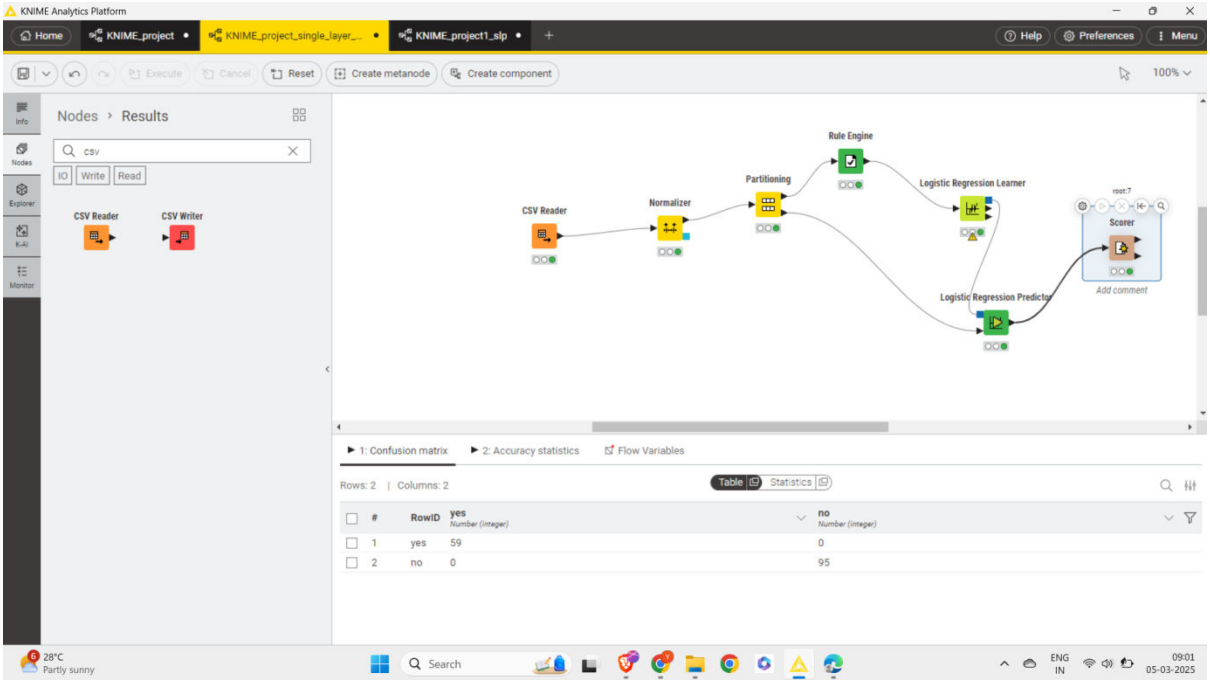
Row ID	Pregna...	Glucose	BloodPr...	SkinThi...	Insulin	BMI	Diabete...	Age	Outcome
Row0	6	148	72	35	0	33.6	0.627	50	1
Row1	1	85	66	29	0	26.6	0.351	31	0
Row2	8	183	64	0	0	23.3	0.672	32	1
Row3	1	89	66	23	94	28.1	0.167	21	0
Row4	0	137	40	35	168	43.1	2.288	33	1
Row5	5	116	74	0	0	25.6	0.201	30	0
Row6	3	78	50	32	88	31	0.248	26	1
Row7	10	115	0	0	0	35.3	0.134	29	0
Row8	2	197	70	45	543	30.5	0.158	53	1
Row9	8	125	96	0	0	0	0.232	54	1


Dialog - 4:2 - Normalizer
Manual
Wildcard
Regex
Type
Search
Aa
Excludes
No columns in this list
Includes
Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI
DiabetesPedigreeFunction
Age
Outcome
Any unknown column
Normalization method
Min-max
Z-score
Decimal scaling
Minimum
0
Maximum
1
Cancel
Ok


Dialog - 4:3 - Partitioning
File
First partition
Flow Variables
Job Manager Selection
Memory Policy
Choose size of first partition
Absolute
100
Relative[%]
80
Take from top
Linear sampling
Draw randomly
Stratified sampling
Use random seed
1,741,143,216,
OK
Apply
Cancel


Dialog - 4:5 - Logistic Regression Learner
File
Settings
Advanced
Flow Variables
Job Manager Selection
Memory Policy
Solver options
Perform calculations lazily (more memory expensive but often faster)
Calculate statistics for coefficients
Termination conditions
Maximal number of epochs:
100
Epsilon:
1.0E-5
Learning rate / step size
Learning rate strategy:
Fixed
Step size:
0.1
Regularization
Prior:
Uniform
Variance:
0.1
Data handling
Hold data in memory
Chunk size:
10,000
Use seed
Seed:
1741143830682
New
OK
Apply
Cancel


Confusion Matrix - 4:7 - Scorer
File
Hilite
Prediction ...
yes
no
yes
59
0
no
0
95
Correct classified: 154
Wrong classified: 0
Accuracy: 100%
Error: 0%
Cohen's kappa (κ): 1%



RESULT

Thus, the above tool Knime was implemented successfully.

KMEANS CLUSTERING – HEART DISEASE CLASSIFICATION

Ex.No.:

Date:

AIM

The aim of this code is to perform clustering on the Heart_Disease_Prediction.csv dataset, using the Age and Cholesterol features to identify patterns and group similar data points using the KMeans clustering algorithm

LIBRARIES USED

- pandas: To handle the dataset and perform data manipulation.
- numpy: For numerical operations.
- sklearn.preprocessing.MinMaxScaler: To scale the data to a range of [0, 1].
- sklearn.cluster.KMeans: For performing the KMeans clustering algorithm.
- matplotlib.pyplot: For generating scatter plots to visualize the data.

METHODS USED

- **MinMaxScaler:** Normalizes the data to scale the features between 0 and 1.
- **KMeans Clustering:**
    - Number of clusters set to 3.
    - init='random' initializes centroids randomly.
    - n_init=100 ensures the algorithm runs 100 times with different initializations and picks the best solution.
- **Scatter Plot Visualization:** To visually compare the data before and after clustering.

PROCEDURE

- **Data Loading:** The dataset is loaded using pandas, focusing on the Age and Cholesterol columns.
- **Preprocessing:** The data is normalized using MinMaxScaler to scale the features to a range of [0, 1].
- **KMeans Clustering:** The KMeans algorithm is applied to the normalized data with 3 clusters.
- **Clustering Prediction:** The clusters are predicted for the dataset, and the cluster centers are identified.
- **Visualization:** Two scatter plots are generated:
- One showing the data before clustering.
- One showing the data after clustering with different colors representing each cluster and cluster centroids.

CODE

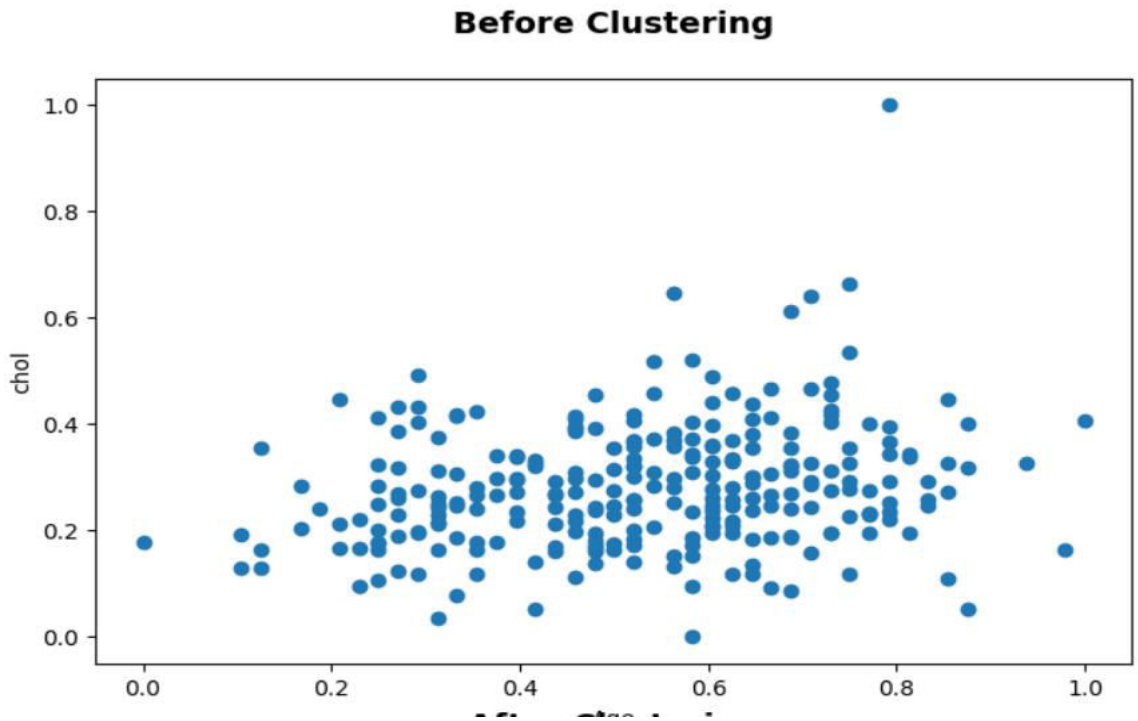
```
import pandas as pd

import numpy as np
```

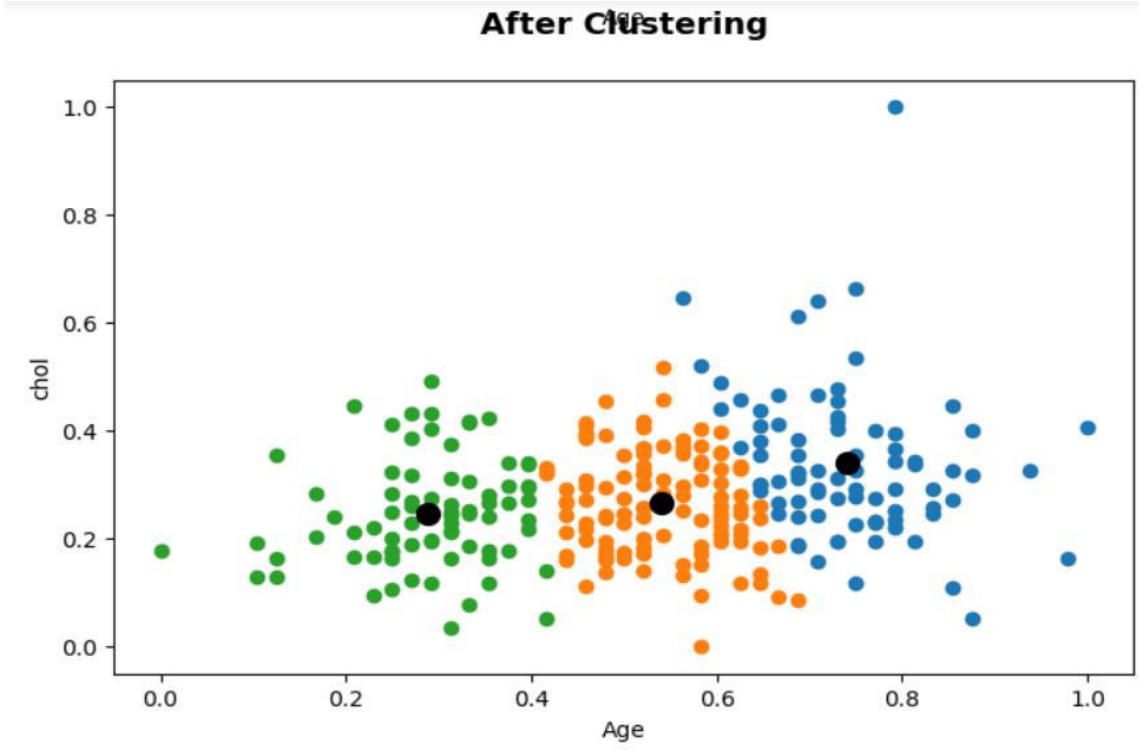
```
from sklearn.preprocessing import MinMaxScaler
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

data=pd.read_csv("Heart_Disease_Prediction.csv",usecols=['Age','Cholesterol'])
print("Data\n",data)

scaler=MinMaxScaler().fit(data)
data_new=scaler.transform(data)

cmodel=KMeans(n_clusters=3,init='random',random_state=42,n_init=100)
cmodel.fit(data_new)
ypredict=cmodel.predict(data_new)
print("Predicted Cluster\n",ypredict)
print("Cluster centers:\n",cmodel.cluster_centers_)
clusterlist=np.unique(ypredict)
fig,ax=plt.subplots(nrows=2,ncols=1,figsize=(8,11))
ax[0].scatter(data_new[:,0],data_new[:,1])
ax[0].set_title("Before Clustering\n",fontsize=14,fontweight="bold")
ax[0].set_xlabel("Age")
ax[0].set_ylabel("chol")
for cluster in clusterlist:
    row_id=np.where(ypredict==cluster)
    ax[1].scatter(data_new[row_id,0],data_new[row_id,1])
clusterhead=cmodel.cluster_centers_
ax[1].scatter(clusterhead[:,0],clusterhead[:,1],s=100,color='k')
```

```
ax[1].set_title("After Clustering\n",fontsize=14,fontweight="bold")

ax[1].set_xlabel("Age")

ax[1].set_ylabel("chol")
```

OUTPUT

```
Data
      Age  Cholesterol
0      70          322
1      67          564
2      57          261
3      64          263
4      74          269
..    ...          ...
265    52          199
266    44          263
267    56          294
268    57          192
269    67          286

[270 rows x 2 columns]

Predicted Cluster
 [0 0 1 0 0 0 1 1 0 0 1 1 2 0 1 0 2 1 0 2 0 2 2 2 1 2 2 1 1 0 1 0 2 0 1 2 1
 1 2 2 2 0 2 2 1 1 2 2 0 0 2 1 0 0 2 2 1 1 1 0 1 1 2 1 0 1 1 0 2 2 0 0 0 0
 0 2 2 2 2 1 1 2 1 2 1 2 0 1 2 1 1 0 1 1 1 2 0 1 0 1 2 0 1 1 0 2 1 0 1 2 1
 2 0 1 2 1 2 1 0 1 2 1 1 0 1 1 0 1 1 1 0 0 2 0 1 2 0 1 2 1 0 1 1 1 1 1 1 2
 1 2 2 1 1 0 1 2 1 2 1 0 2 0 1 1 2 0 1 1 2 0 0 0 0 0 2 0 1 2 0 1 2 0 2 2 1
 2 1 1 0 0 1 0 1 2 2 1 1 1 0 0 0 1 0 1 1 1 0 1 0 2 0 1 2 1 2 2 0 1 1 2 1 0
 1 0 2 2 0 2 1 1 0 2 2 1 0 1 1 2 2 1 0 2 1 0 1 0 0 0 1 0 1 2 2 1 1 0 0 1 0
 2 0 1 1 1 2 1 2 1 1 0]
Cluster centers:
 [[0.74034553 0.33937521]
 [0.53962054 0.26492172]
 [0.28837719 0.24642514]]

Text(0, 0.5, 'chol')
```





RESULT

The result will display the predicted clusters and visualization before and after clustering.

2. HOW WILL YOU CHOOSE K VALUE

AIM

The aim of this code is to determine the optimal number of clusters (K) for KMeans clustering using the Elbow Method.

LIBRARIES USED

- pandas: To handle the dataset and perform data manipulation.
- numpy: For numerical operations.
- sklearn.preprocessing.MinMaxScaler: To scale the data to a range of [0, 1].
- sklearn.cluster.KMeans: For performing the KMeans clustering algorithm.
- matplotlib.pyplot: For visualizing the Elbow Method.

METHODS USED

- **Elbow Method**: Evaluates the sum of squared distances (inertia) within clusters to determine the optimal number of clusters.

PROCEDURE

1. Load the dataset focusing on the 'Age' and 'Cholesterol' columns.
2. Normalize the data using MinMaxScaler.
3. Run KMeans clustering for different values of K (1 to 10) and calculate inertia values.
4. Plot the inertia values against K to observe the Elbow Point.

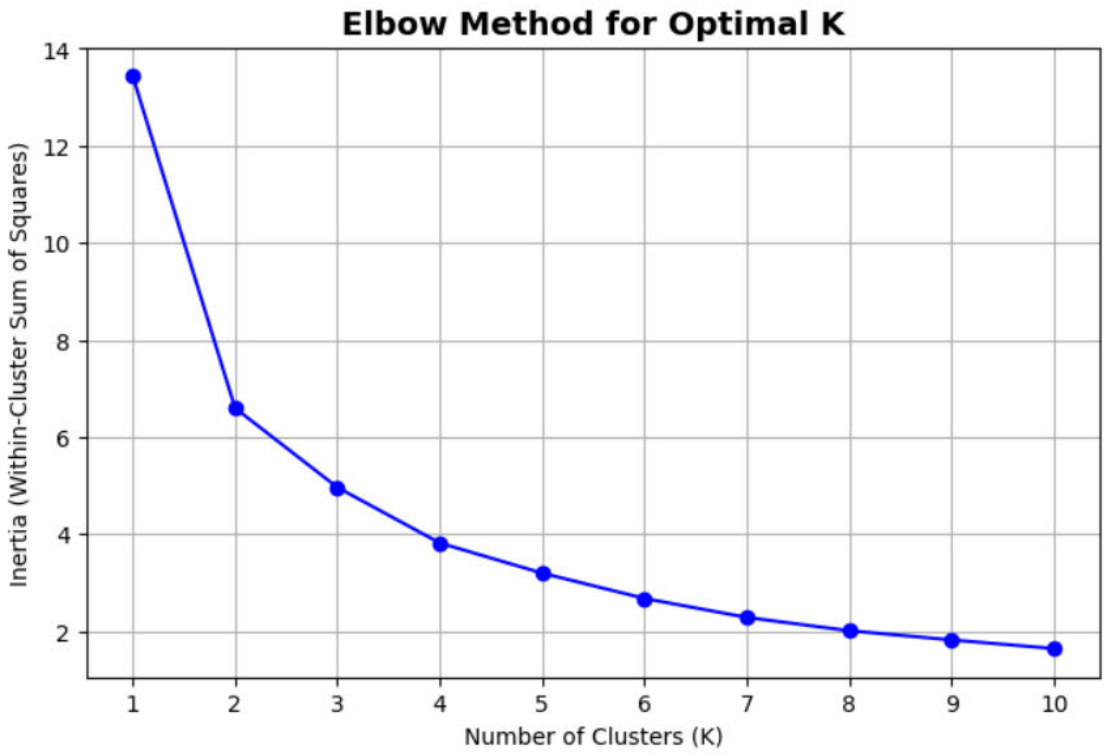
CODE

```
inertia_values = []

K_range = range(1, 11)


for k in K_range:
    km = KMeans(n_clusters=k, random_state=42, n_init=10)
    km.fit(data_new)
    inertia_values.append(km.inertia_)


print("KValue:",k)
```

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

plt.plot(K_range, inertia_values, marker='o', linestyle='-', color='b')

plt.title("Elbow Method for Optimal K", fontsize=14, fontweight="bold")

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

plt.ylabel("Inertia (Within-Cluster Sum of Squares)")

plt.xticks(K_range)

plt.grid(True)

plt.show()
```

OUTPUT

```
KValue: 10
```



RESULT

The optimal number of clusters (K) is determined using the Elbow Method by identifying the point where the inertia stops decreasing significantly.

3.HOW WILL YOU EVALUATE K VALUE

AIM

The aim of this code is to evaluate the quality of clustering for different values of K using the Silhouette Score and Davies-Bouldin Index.

LIBRARIES USED

- pandas: To handle the dataset and perform data manipulation.
- numpy: For numerical operations.
- matplotlib.pyplot: For visualizing evaluation metrics.
- sklearn.preprocessing.MinMaxScaler: To scale the data.
- sklearn.cluster.KMeans: For clustering the data.
- sklearn.metrics: For evaluating clustering performance using Silhouette Score and Davies-Bouldin Index.

METHODS USED

- **Silhouette Score**: Measures how similar a data point is to its own cluster compared to other clusters.
- **Davies-Bouldin Index**: Evaluates clustering compactness and separation.

PROCEDURE

1. Load the dataset focusing on 'Age' and 'Cholesterol'.
2. Normalize the data using MinMaxScaler.
3. Perform KMeans clustering for different values of K (2 to 10).
4. Calculate the Silhouette Score for each K.
5. Plot the Silhouette Score against K to determine the best K.
6. Compute additional evaluation metrics (Davies-Bouldin Index and Inertia) for the optimal K.

CODE

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler

from sklearn.cluster import KMeans

from sklearn.metrics import silhouette_score, davies_bouldin_score
```

```
# file_path = r'C:\Users\2022503550\Downloads\Heart_Disease_Prediction.csv'
data=pd.read_csv("Heart_Disease_Prediction.csv",usecols=['Age','Cholesterol'])
# data = pd.read_csv(file_path, usecols=['Age', 'Cholesterol'])

print("Dataset:\n", data.head())

scaler = MinMaxScaler().fit(data)
data_scaled = scaler.transform(data)

K_range = range(2, 11)
silhouette_scores = []

for k in K_range:
    kmeans = KMeans(n_clusters=k, init='random', random_state=42, n_init=100)
    kmeans.fit(data_scaled)
    silhouette_scores.append(silhouette_score(data_scaled, kmeans.labels_))

plt.figure(figsize=(8, 5))
plt.plot(K_range, silhouette_scores, marker='s', linestyle='-', color='r')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Silhouette Score')
plt.title('Silhouette Score for Optimal K')
plt.show()

optimal_k = 3
```

```
cmodel = KMeans(n_clusters=optimal_k, init='random', random_state=42, n_init=100)

cmodel.fit(data_scaled)

ypredict = cmodel.predict(data_scaled)


silhouette_avg = silhouette_score(data_scaled, ypredict)

print(f"Silhouette Score for k={optimal_k}: {silhouette_avg}")


db_index = davies_bouldin_score(data_scaled, ypredict)

print(f"Davies-Bouldin Index for k={optimal_k}: {db_index}")


inertia = cmodel.inertia_

print(f"Intra-cluster similarity (inertia) for k={optimal_k}: {inertia}")


centroids = cmodel.cluster_centers_

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

print(f"Inter-cluster similarity (pairwise distances between centroids) for k={optimal_k}: \n{distances}"
```

OUTPUT

- Graph showing the Silhouette Score for different K values.
- Evaluation metrics (Silhouette Score, Davies-Bouldin Index, Inertia, and Inter-cluster distances) displayed for the optimal K.

```
Dataset:
    Age  Cholesterol
0   70           322
1   67           564
2   57           261
3   64           263
4   74           269
```


```
Silhouette Score for k=3: 0.3404712294581846
Davies-Bouldin Index for k=3: 1.0382108011858866
Intra-cluster similarity (inertia) for k=3: 4.959281348727481
Inter-cluster similarity (pairwise distances between centroids) for k=3:
[[0.         0.2140884  0.46142723]
 [0.2140884  0.         0.25192328]
 [0.46142723 0.25192328 0.        ]]
```

RESULT

The optimal K value is determined based on the highest Silhouette Score, along with Davies-Bouldin Index and Inertia to assess clustering quality.

GENETIC ALGORITHM FOR OPTIMIZATION

Ex.No.:

Date:

AIM

The aim of this code is to minimize the function f(x)=2x2−3x+10 using a genetic algorithm. The algorithm will perform selection, crossover, and mutation operations over two generations with a fixed population size of 4. The chromosomes are represented using 6-bit binary encoding, and the initial population consists of the values [2, 8, 5, 11].

PROCEDURE

1. Convert the initial population values [2, 8, 5, 11] into 6-bit binary chromosomes.
2. For each generation:
    - Calculate the fitness of each individual.
    - Perform roulette wheel selection to select parents.
    - Perform two-point crossover to generate offspring.
    - Apply mutation to the offspring.
    - Convert the binary chromosomes back to decimal values to form the new population.
3. Output the final population after two generations.

LIBRARIES USED

- numpy
- random

CODE

```
import numpy as np
import random
def f(x):
    return 2 * x**2 - 3 * x + 10
def decimal_to_binary(n, bits=6):
    return format(n, f'0{bits}b')

def binary_to_decimal(b):
    return int(b, 2)
def roulette_wheel_selection(population, fitness_values):
```

```
    max_fitness = max(fitness_values) + 1  # Avoid division by zero
    inverted_fitness = [max_fitness - f for f in fitness_values]  # Higher is better for selection
    total_fitness = sum(inverted_fitness)
    probabilities = [f/total_fitness for f in inverted_fitness]
    cum_prob = np.cumsum(probabilities)
    r = random.random()
    for i, cp in enumerate(cum_prob):
        if r < cp:
            return population[i]
    return population[-1]
def two_point_crossover(parent1, parent2):
    point1, point2 = sorted(random.sample(range(1, len(parent1)), 2))
    child1 = parent1[:point1] + parent2[point1:point2] + parent1[point2:]
    child2 = parent2[:point1] + parent1[point1:point2] + parent2[point2:]
    return child1, child2
def mutate(chromosome, mutation_rate=0.1):
    mutated = list(chromosome)
    for i in range(len(mutated)):
        if random.random() < mutation_rate:
            mutated[i] = '1' if mutated[i] == '0' else '0'
    return ''.join(mutated)

population = [2, 8, 5, 11]
chromosomes = [decimal_to_binary(x, bits=6) for x in population]
generations = 2
for gen in range(generations):
    print(f"\nGeneration {gen + 1}:")
    fitness_values = [f(x) for x in population]
    print("Fitness:", fitness_values)
    selected_chromosomes = [roulette_wheel_selection(chromosomes, fitness_values) for _ in
range(len(population))]
    new_chromosomes = []
    for i in range(0, len(selected_chromosomes), 2):
        c1, c2 = two_point_crossover(selected_chromosomes[i], selected_chromosomes[i+1])
        new_chromosomes.extend([c1, c2])
    mutated_chromosomes = [mutate(c) for c in new_chromosomes]
    population = [binary_to_decimal(c) for c in mutated_chromosomes]

print("\nFinal Population:", population)
```

OUTPUT

```
Generation 1:
Fitness: [12, 114, 45, 219]

Generation 2:
Fitness: [12, 12, 45, 45]

Final Population: [8, 8, 0, 2]
```

RESULT

The genetic algorithm successfully minimized the function f(x)=2x2−3x+10 over two generations, with the final population showing improved fitness values.

AIM

The aim of this code is to maximize the function f(x)=x2 using a genetic algorithm. The algorithm will perform selection, crossover, and mutation operations over 20 generations with a population size of 10. The individuals are represented as integers, and the initial population is randomly generated within the range [0, 31].

PROCEDURE

1. Initialize a population of 10 individuals with random values between 0 and 31.
2. For each generation:
    - Calculate the fitness of each individual.
    - Perform roulette wheel selection to select parents.
    - Perform single-point crossover to generate offspring.
    - Apply mutation to the offspring.
3. Output the best individual found after 20 generations.

LIBRARIES USED

- random

CODE

```
import random
def fitness_function(x):
    return x ** 2
def initialize_population(size, lower, upper):
    return [random.randint(lower, upper) for _ in range(size)]
def select_parent(population, fitness):
    total_fitness = sum(fitness)
    selection_probs = [f / total_fitness for f in fitness]
    rand_val = random.uniform(0, 1)
    cumulative = 0
    for i, prob in enumerate(selection_probs):
        cumulative += prob
        if rand_val <= cumulative:
            return population[i]
    return population[-1]
def crossover(parent1, parent2):
    point = random.randint(1, len(bin(parent1)[2:]) - 1)
    mask = (1 << point) - 1
    offspring1 = (parent1 & mask) | (parent2 & ~mask)
    offspring2 = (parent2 & mask) | (parent1 & ~mask)
    return offspring1, offspring2
def mutate(individual, mutation_rate=0.1):
    if random.random() < mutation_rate:
        bit = 1 << random.randint(0, len(bin(individual)[2:]) - 1)
        individual ^= bit
    return individual
def genetic_algorithm(pop_size=10, generations=20, lower=0, upper=31):
    population = initialize_population(pop_size, lower, upper)

    for gen in range(generations):
        fitness = [fitness_function(ind) for ind in population]
        new_population = []
        for _ in range(pop_size // 2):
            parent1, parent2 = select_parent(population, fitness), select_parent(population, fitness)
            child1, child2 = crossover(parent1, parent2)
            new_population.extend([mutate(child1), mutate(child2)])

        population = new_population
```

```
        best_individual = max(population, key=fitness_function)
        print(f"Gen {gen+1}: Best = {best_individual}, Fitness =
{fitness_function(best_individual)}")

    return max(population, key=fitness_function)
best_solution = genetic_algorithm()
print(f"Best solution found: {best_solution}")
```

OUTPUT

```
Gen 1: Best = 31, Fitness = 961
Gen 2: Best = 31, Fitness = 961
Gen 3: Best = 31, Fitness = 961
Gen 4: Best = 31, Fitness = 961
Gen 5: Best = 31, Fitness = 961
Gen 6: Best = 31, Fitness = 961
Gen 7: Best = 31, Fitness = 961
Gen 8: Best = 31, Fitness = 961
Gen 9: Best = 31, Fitness = 961
Gen 10: Best = 31, Fitness = 961
Gen 11: Best = 31, Fitness = 961
Gen 12: Best = 31, Fitness = 961
Gen 13: Best = 31, Fitness = 961
Gen 14: Best = 31, Fitness = 961
Gen 15: Best = 31, Fitness = 961
Gen 16: Best = 31, Fitness = 961
Gen 17: Best = 31, Fitness = 961
Gen 18: Best = 31, Fitness = 961
Gen 19: Best = 30, Fitness = 900
Gen 20: Best = 30, Fitness = 900
Best solution found: 30
```

RESULT

The genetic algorithm successfully maximized the function f(x)=x2 over 20 generations, with the best solution found being the individual with the highest fitness value.

TOOL STUDY ORANGE

ORANGE

Orange is an open-source data visualization, data mining, and machine learning tool that provides an easy-to-use graphical user interface (GUI) for analyzing data without requiring extensive programming knowledge.

KEY FEATURES:

1. **Drag-and-Drop Workflow** – No coding required!
2. **Data Preprocessing** – Handles missing values, normalization, feature selection, etc
3. **Machine Learning** – Supports classification, regression, clustering, and deep learning.
4. **Visualization** – Interactive graphs, scatter plots, decision trees, and heatmaps.

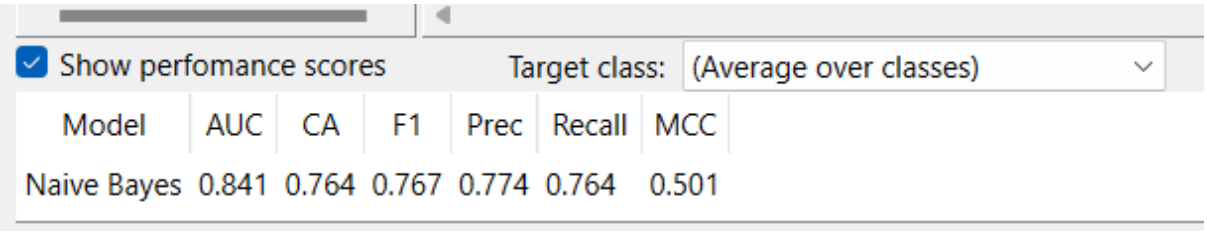
TABULATION:

CLASSIFIER	DATASET USED	AREA UNDER CURVE	ACCURACY	F1 SCORE	PRECISION	RECALL	MCC
Naïve Bayes	**PIMA INDIAN**	0.789	81.6	0.75	0.76	0.746	0.467
	PLAY TENNIS	0.64	65	0.649	0.652	0.65	0.302
	FOREST FIRE	0.973	93.6	0.936	0.939	0.939	0.874
Decision Tree	**PIMA INDIAN**	0.638	68.6	0.684	0.683	0.686	0.304
	PLAY TENNIS	0.938	90	0.897	0.914	0.9	0.802
	FOREST FIRE	0.97	95.7	0.957	0.958	0.957	0.914
Multilayer Perceptron	**PIMA INDIAN**	0.86	79.6	0.793	0.792	0.796	0.542
	PLAY TENNIS	0.979	0.9	0.897	0.914	0.9	0.802
	FOREST FIRE	0.972	90	0.9	0.911	0.9	0.81
Random Forest	**PIMA INDIAN**	0.994	96.5	0.965	0.965	0.965	0.962
	PLAY TENNIS	0.854	80	0.8	0.8	0.8	0.583
	FOREST FIRE	1.0	99.4	0.994	0.944	0.944	0.988

COMPARISON

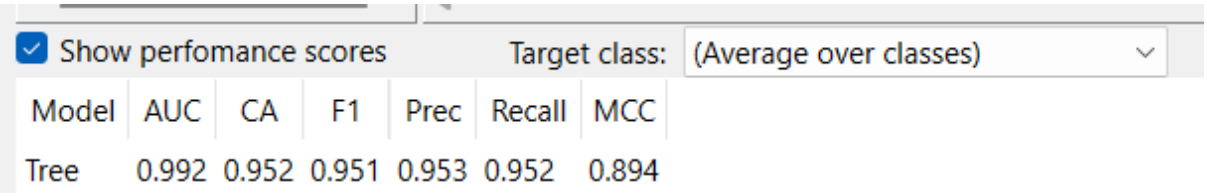
1.PIMA INDIAN DATSET:

Naïve Bayes



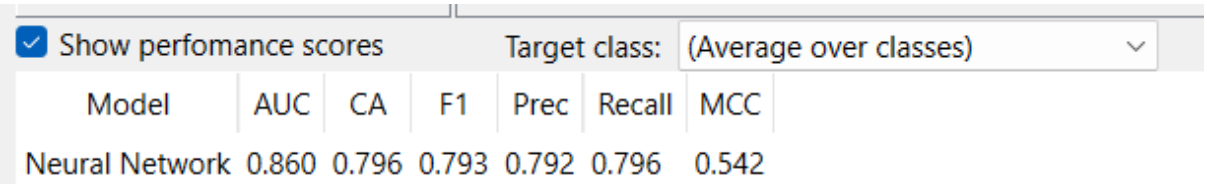
Model	AUC	CA	F1	Prec	Recall	MCC
Naive Bayes	0.841	0.764	0.767	0.774	0.764	0.501

Decision Tree



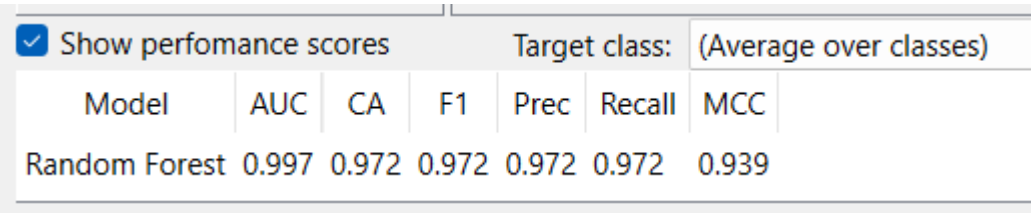
Model	AUC	CA	F1	Prec	Recall	MCC
Tree	0.992	0.952	0.951	0.953	0.952	0.894

MultiLayer Perceptron



Model	AUC	CA	F1	Prec	Recall	MCC
Neural Network	0.860	0.796	0.793	0.792	0.796	0.542

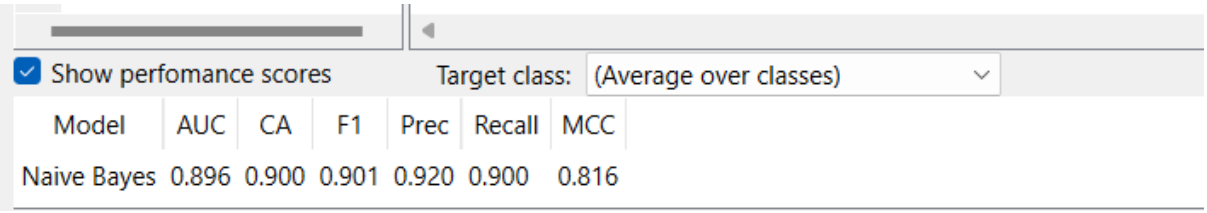
Random Forest



Model	AUC	CA	F1	Prec	Recall	MCC
Random Forest	0.997	0.972	0.972	0.972	0.972	0.939

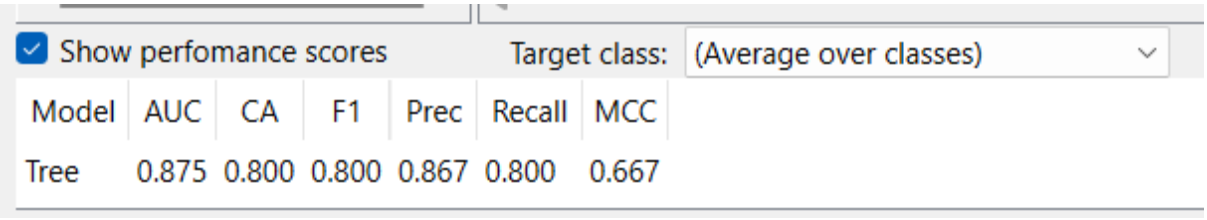
2.PLAY TENNIS:

Naïve Bayes



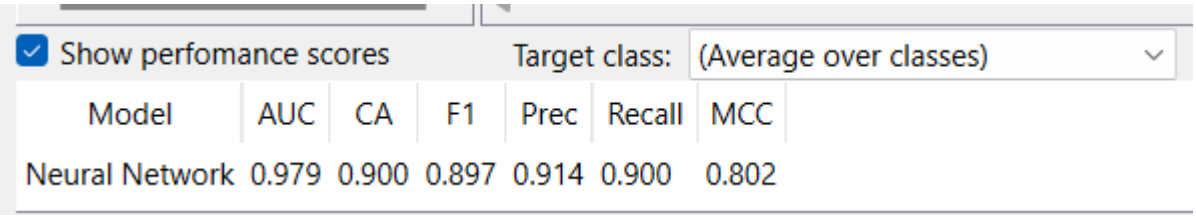
Model	AUC	CA	F1	Prec	Recall	MCC
Naive Bayes	0.896	0.900	0.901	0.920	0.900	0.816

Decision Tree


Show perfomance scores Target class: (Average over classes)

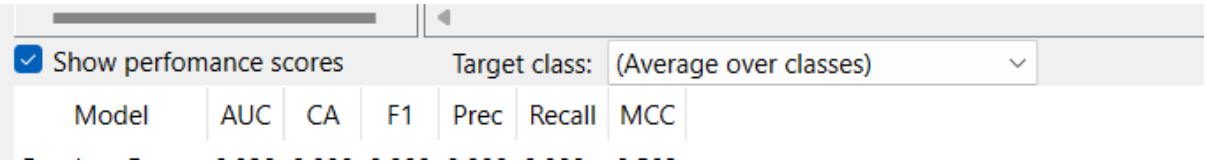
Model	AUC	CA	F1	Prec	Recall	MCC
Tree	0.875	0.800	0.800	0.867	0.800	0.667

MultiLayer Perceptron


Show perfomance scores Target class: (Average over classes)

Model	AUC	CA	F1	Prec	Recall	MCC
Neural Network	0.979	0.900	0.897	0.914	0.900	0.802

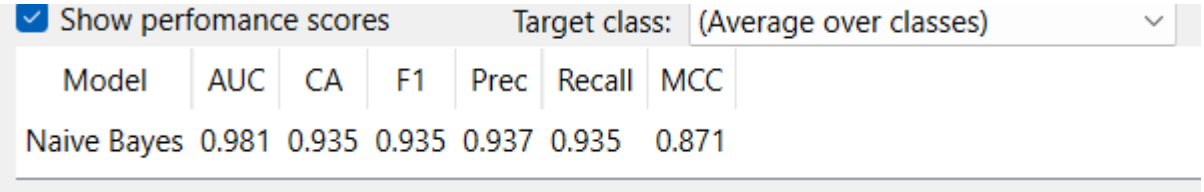
Random Forest


Show perfomance scores Target class: (Average over classes)

Model	AUC	CA	F1	Prec	Recall	MCC

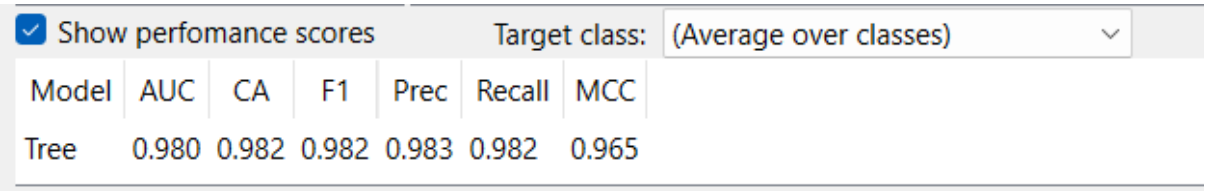
3.FOREST FIRE:

Naïve Bayes


Show perfomance scores Target class: (Average over classes)

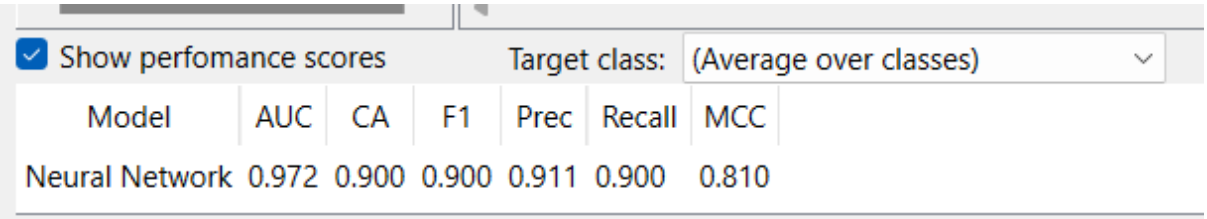
Model	AUC	CA	F1	Prec	Recall	MCC
Naive Bayes	0.981	0.935	0.935	0.937	0.935	0.871

Decision Tree


Show perfomance scores Target class: (Average over classes)

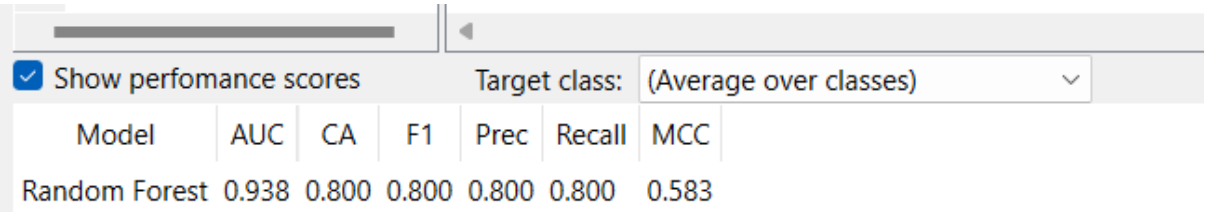
Model	AUC	CA	F1	Prec	Recall	MCC
Tree	0.980	0.982	0.982	0.983	0.982	0.965

MultiLayer Perceptron


Show perfomance scores Target class: (Average over classes)

Model	AUC	CA	F1	Prec	Recall	MCC
Neural Network	0.972	0.900	0.900	0.911	0.900	0.810

Random Forest



Model	AUC	CA	F1	Prec	Recall	MCC
Random Forest	0.938	0.800	0.800	0.800	0.800	0.583

RESULT

Thus, the above tool “Orange” was implemented successfully.

K-NEAREST NEIGHBORS

Ex.No.:

Date:

A. CLASSIFICATION

AIM

A **K-Nearest Neighbors (KNN) classification model** to predict heart disease based on patient health data.

DATASET DESCRIPTION

- The dataset used is **Heart_Disease_Prediction.csv**.
- It consists of multiple patient health parameters as features (**X**) and a target variable (**y**) indicating whether a patient has heart disease.
- The last column in the dataset is considered the target variable.

METHODS USED

- The dataset is loaded and split into features (**X**) and target (**y**).
- Data is standardized using **StandardScaler** to ensure uniform feature scaling.
- The dataset is split into **training (80%)** and **testing (20%)** sets.
- The **KNN Classifier (k=5)** is trained using the training set.
- The model is evaluated using the **accuracy_score** metric.

LIBRARIES USED

- **pandas** – for handling and manipulating dataset.
- **numpy** – for numerical operations.
- **sklearn.model_selection** – for splitting the dataset into training and testing sets.
- **sklearn.preprocessing** – for feature scaling.
- **sklearn.neighbors** – for implementing KNN classification and KNN imputation.
- **sklearn.metrics** – for evaluating the performance of the models.

PROCEDURE

1. Load the dataset (**Heart_Disease_Prediction.csv**).
2. Define the features (**X**) and target variable (**y**).
3. Split the dataset into training and testing sets (80-20 split).
4. Standardize the feature values using **StandardScaler**.
5. Train the **KNN Classifier** with **k=5** on the training data.
6. Predict the test data outcomes.

7. Evaluate the model using **accuracy_score**.
8. Display the accuracy result.

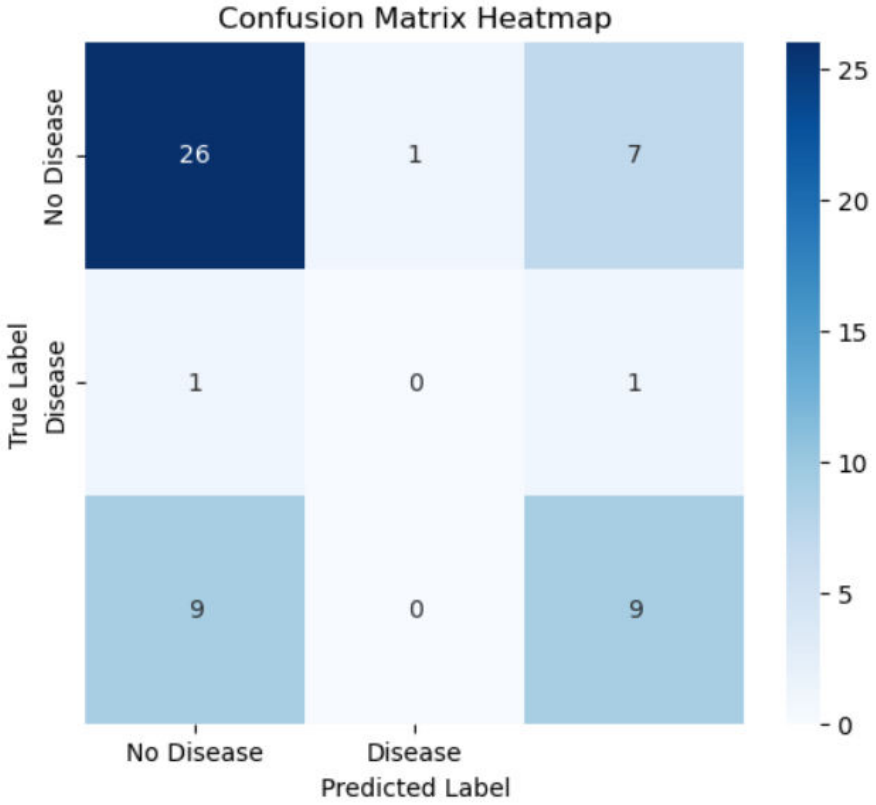
CODE

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report


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


X = df.iloc[:, :-1]

y = df.iloc[:, -1]


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


scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)


k = 5

knn = KNeighborsClassifier(n_neighbors=k)

knn.fit(X_train, y_train)


y_pred = knn.predict(X_test)
```

```
accuracy = accuracy_score(y_test, y_pred)

print(f"KNN Accuracy: {accuracy:.2f}")


cm = confusion_matrix(y_test, y_pred)

print("\nConfusion Matrix:\n", cm)


report = classification_report(y_test, y_pred)

print("\nClassification Report:\n", report)


plt.figure(figsize=(6, 5))

sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["No Disease", "Disease"],
yticklabels=["No Disease", "Disease"])

plt.xlabel("Predicted Label")

plt.ylabel("True Label")

plt.title("Confusion Matrix Heatmap")

plt.show()


plt.figure(figsize=(5, 4))

plt.bar(["Test Accuracy"], [accuracy], color="green")

plt.ylabel("Accuracy")

plt.ylim(0, 1)

plt.title("KNN Model Accuracy")

plt.show()
```

OUTPUT

```
KNN Accuracy: 0.65

Confusion Matrix:
 [[26  1  7]
 [ 1  0  1]
 [ 9  0  9]]

Classification Report:
              precision    recall  f1-score   support

           3       0.72      0.76      0.74        34
           6       0.00      0.00      0.00         2
           7       0.53      0.50      0.51        18

    accuracy                           0.65        54
   macro avg       0.42      0.42      0.42        54
weighted avg       0.63      0.65      0.64        54
```



RESULT

Thus, the code was implemented and executed successfully.

B. MISSING VALUE IMPUTATION

AIM

A **KNN Imputation method** to handle missing values in a dataset related to horse colic disease prediction.

DATASET DESCRIPTION

- The dataset used is **horseColic.csv**.
- It contains numerical and categorical attributes related to horse health conditions.
- The dataset may contain missing values, which will be handled using KNN Imputation.

METHODS USED

- The dataset is loaded with missing values represented as '?'.
- All columns are converted to numeric format to handle mixed data types.
- 10% missing values are introduced artificially for evaluation.
- The dataset is normalized using **StandardScaler** before applying imputation.
- **KNNImputer (k=3)** is used to replace missing values based on the nearest neighbors.
- The imputed data is then **denormalized** back to its original scale.
- Evaluation metrics such as **Mean Absolute Deviation (MAD)**, **Mean Squared Error (MSE)**, **Root Mean Squared Error (RMSE)**, and **Mean Absolute Error (MAE)** are computed.
- The imputed dataset is saved as **horseColicKnnImputation.csv**.

LIBRARIES USED

- **pandas** – for handling and manipulating dataset.
- **numpy** – for numerical operations.
- **sklearn.model_selection** – for splitting the dataset into training and testing sets.
- **sklearn.preprocessing** – for feature scaling.
- **sklearn.neighbors** – for implementing KNN classification and KNN imputation.
- **sklearn.metrics** – for evaluating the performance of the models.

PROCEDURE

1. Load the dataset (**horseColic.csv**) and convert all columns to numeric.
2. Select only numerical columns.
3. Create a copy of the dataset for evaluation purposes.
4. Introduce 10% missing values at random positions.
5. Normalize the dataset using **StandardScaler**.
6. Apply **KNN Imputation (k=3)** to fill missing values.
7. Denormalize the dataset to its original scale.
8. Extract original and imputed values for comparison.
9. Calculate error metrics: **MAD, MSE, RMSE, MAE**.

10. Compute accuracy within ±5% tolerance and exact match accuracy.
11. Save the imputed dataset as **horseColicKnnImputation.csv**.

CODE

```
import pandas as pd
import numpy as np
from sklearn.impute import KNNImputer
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error

df = pd.read_csv("horseColic.csv", na_values="?")
df = df.apply(pd.to_numeric, errors='coerce')

num_cols = df.select_dtypes(include=[np.number]).columns
df = df[num_cols]

df_original = df.copy()

missing_fraction = 0.1
np.random.seed(42)
missing_indices = np.random.choice(df.size, int(missing_fraction * df.size), replace=False)

df_missing = df_original.copy()
df_missing.values.ravel()[missing_indices] = np.nan

scaler = StandardScaler()
df_scaled = scaler.fit_transform(df_missing)

knn_imputer = KNNImputer(n_neighbors=3)
df_imputed_scaled = knn_imputer.fit_transform(df_scaled)

df_imputed = scaler.inverse_transform(df_imputed_scaled)
df_imputed = pd.DataFrame(df_imputed, columns=df.columns)

true_values = df_original.values.ravel()[missing_indices]
imputed_values = df_imputed.values.ravel()[missing_indices]

valid_indices = ~np.isnan(true_values)
true_values = true_values[valid_indices]
imputed_values = imputed_values[valid_indices]

mad = np.mean(np.abs(true_values - imputed_values))
mse = mean_squared_error(true_values, imputed_values)
rmse = np.sqrt(mse)
```

```
mae = mean_absolute_error(true_values, imputed_values)

tolerance = 0.05 * np.abs(true_values)
correct_imputations = np.abs(true_values - imputed_values) <= tolerance
accuracy_tolerance = np.mean(correct_imputations) * 100

exact_matches = np.sum(true_values == imputed_values)
accuracy_exact_match = (exact_matches / len(true_values)) * 100

df_imputed.to_csv("horseColicKnnImputation.csv", index=False)

print(f"Mean Absolute Deviation (MAD): {mad:.4f}")
print(f"Mean Squared Error (MSE): {mse:.4f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.4f}")
print(f"Mean Absolute Error (MAE): {mae:.4f}")
print(f"Imputation Accuracy (within ±5% tolerance): {accuracy_tolerance:.2f}%")
print(f"Exact Match Accuracy: {accuracy_exact_match:.2f}%")
print("Total missing values introduced:", np.isnan(df_missing.values).sum())
print("Total valid true_values:", len(true_values))
print("Total valid imputed_values:", len(imputed_values))
print("Sample true values:", true_values[:10])
print("Sample imputed values:", imputed_values[:10])
print("Imputed dataset saved as 'horseColicKnnImputation.csv'.")
```

OUTPUT

```
Mean Absolute Deviation (MAD): 0.0000
Mean Squared Error (MSE): 0.0000
Root Mean Squared Error (RMSE): 0.0000
Mean Absolute Error (MAE): 0.0000
Imputation Accuracy (within ±5% tolerance): 99.57%
Exact Match Accuracy: 97.98%
Total missing values introduced: 1605
Total valid true_values: 693
Total valid imputed_values: 693
Sample true values: [2.000e+01 3.000e+00 3.205e+03 1.000e+00 1.000e+00 2.000e+00 1.000e+00
 1.000e+00 0.000e+00 2.000e+00]
Sample imputed values: [2.000e+01 3.000e+00 3.205e+03 1.000e+00 1.000e+00 2.000e+00 1.000e+00
 1.000e+00 0.000e+00 2.000e+00]
Imputed dataset saved as 'horseColicKnnImputation.csv'.
```

RESULT

Thus, the code was implemented and executed successfully.

PRINCIPAL COMPONENT ANALYSIS

Ex.No.:

Date:

AIM

To implement the Principal Component Analysis (PCA) technique using Python, list the principal components, explain the selection process, and print intermediate results.

METHODS USED

- Data Standardization
- Principal Component Analysis (PCA)
- Dimensionality Reduction
- Data Visualization

LIBRARIES USED

- numpy
- pandas
- matplotlib
- seaborn
- sklearn.datasets
- sklearn.preprocessing
- sklearn.decomposition

DATASET DESCRIPTION

Iris Dataset
Contains 150 samples of iris flowers with 4 numerical features:

- Sepal Length
- Sepal Width
- Petal Length
- Petal Width
  Target: 3 classes (Setosa, Versicolor, Virginica)

PROCEDURE

1. Load the dataset and convert it to a DataFrame.
2. Standardize the features.

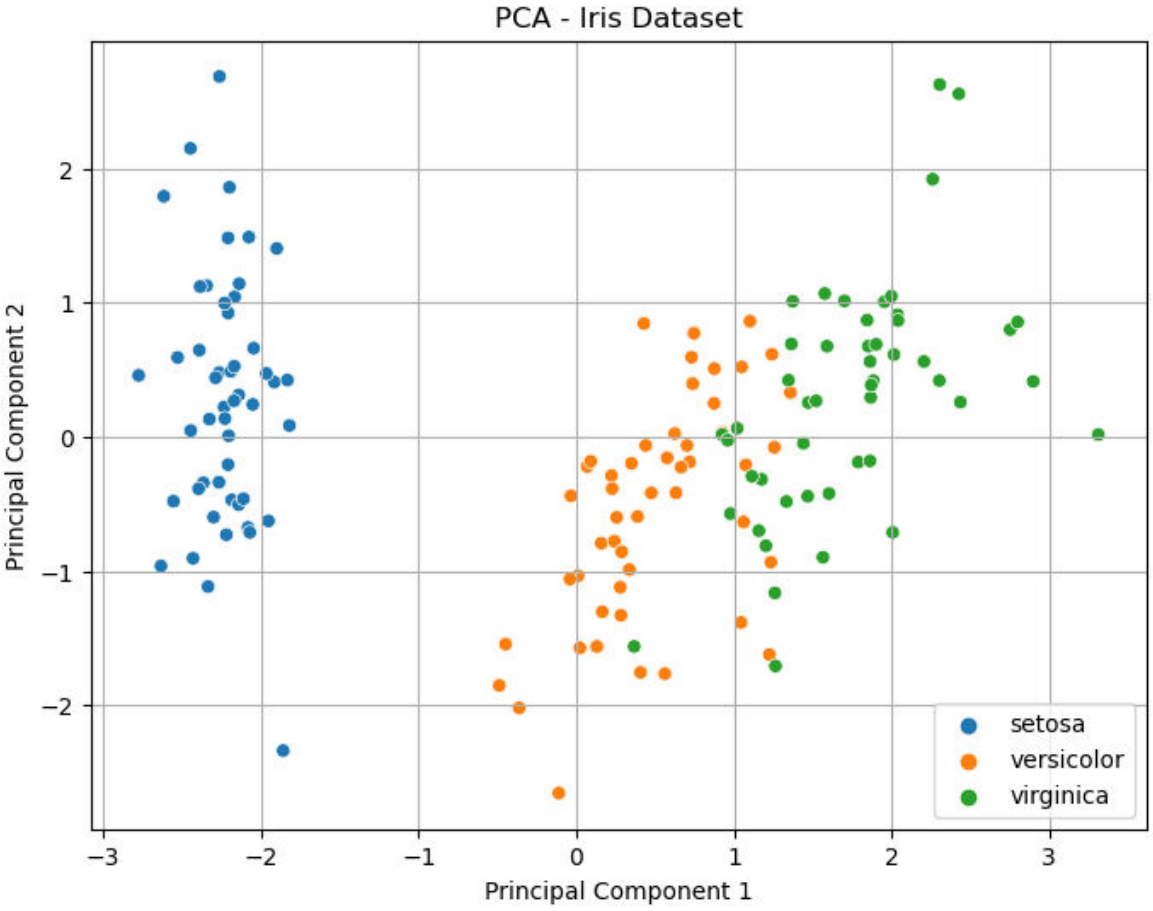
3. Apply PCA to reduce dimensions.
4. Extract and display principal components and explained variance.
5. Select components based on cumulative variance.
6. Visualize data in reduced space.

CODE

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.datasets import load_iris

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA


iris = load_iris()

X = iris.data

y = iris.target

feature_names = iris.feature_names


df = pd.DataFrame(X, columns=feature_names)

print("Original Data:\n", df.head())


scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)

print("\nStandardized Data (first 5 rows):\n", X_scaled[:5])


pca = PCA()

X_pca = pca.fit_transform(X_scaled)
```

```
print("\nExplained Variance Ratio:")
for i, var in enumerate(pca.explained_variance_ratio_):
    print(f"PC{i+1}: {var:.4f}")

cumulative_variance = np.cumsum(pca.explained_variance_ratio_)
print("\nCumulative Explained Variance:")
for i, var in enumerate(cumulative_variance):
    print(f"PC1 to PC{i+1}: {var:.4f}")

print("\nPrincipal Components (Eigenvectors):\n", pca.components_)

n_components = np.argmax(cumulative_variance >= 0.95) + 1
print(f"\nNumber of components selected to retain 95% variance: {n_components}")

pca_final = PCA(n_components=n_components)
X_reduced = pca_final.fit_transform(X_scaled)

plt.figure(figsize=(8, 6))
sns.scatterplot(x=X_reduced[:, 0], y=X_reduced[:, 1], hue=iris.target_names[y])
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.title("PCA - Iris Dataset")
plt.grid(True)
plt.show()
```

OUTPUT


PCA - Iris Dataset
Principal Component 2
Principal Component 1
2
1
0
−1
−2
−3
−2
−1
0
1
2
3
setosa
versicolor
virginica

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

Standardized Data (first 5 rows):
 [[-0.90068117  1.01900435 -1.34022653 -1.3154443 ]
 [-1.14301691 -0.13197948 -1.34022653 -1.3154443 ]
 [-1.38535265  0.32841405 -1.39706395 -1.3154443 ]
 [-1.50652052  0.09821729 -1.2833891  -1.3154443 ]
 [-1.02184904  1.24920112 -1.34022653 -1.3154443 ]]

Explained Variance Ratio:
PC1: 0.7296
PC2: 0.2285
PC3: 0.0367
PC4: 0.0052

Cumulative Explained Variance:
PC1 to PC1: 0.7296
PC1 to PC2: 0.9581
PC1 to PC3: 0.9948
PC1 to PC4: 1.0000

Principal Components (Eigenvectors):
 [[ 0.52106591 -0.26934744  0.5804131   0.56485654]
 [ 0.37741762  0.92329566  0.02449161  0.06694199]
 [-0.71956635  0.24438178  0.14212637  0.63427274]
 [-0.26128628  0.12350962  0.80144925 -0.52359713]]

Number of components selected to retain 95% variance: 2
```

RESULT

PCA was successfully applied to the Iris dataset. The first **2 principal components** retain approximately **95.8%** of the variance, making them suitable for dimensionality reduction. The principal components were visualized and showed clear class separation.

RADIAL BASIS FUNCTION (RBF)

Ex.No.:

Date:

AIM

To implement a Radial Basis Function (RBF) Neural Network with Gaussian kernels to learn the non-linearly separable XOR function, and demonstrate how center-based transformations enable a linear output layer to solve complex classification tasks.

LIBRARIES USED

numpy – array operations and linear algebra

METHODS USED

 Gaussian RBF Transformation: Map inputs into a higher-dimensional space via radial basis functions.

 Center Selection: Use predefined centers (corners of the XOR input) as RBF kernel centers.

 Linear Regression (Pseudoinverse): Compute output weights by solving a linear system via the Moore-Penrose pseudoinverse.

PROCEDURE

 Define Gaussian RBF: Function computing

$$\exp(-\|x - c\|^2/(2\sigma^2))$$

 Prepare XOR Data: Four 2-D input vectors and their binary labels.

 Select Centers: Use the four possible input vectors as RBF centers.

 Compute Φ (Phi) Matrix: For each sample and each center, compute the RBF response.

 Compute Weights: Use the pseudoinverse of Φ to solve for weights mapping RBF outputs to labels.

 Predict: For any input, compute its RBF responses and apply learned weights.

 Evaluate: Print continuous predictions and rounded binary outputs.

CODE

```
import numpy as np


# Gaussian RBF function

def rbf(x, c, s):

    return np.exp(-np.linalg.norm(x - c)**2 / (2 * s**2))


# Training data for XOR

X = np.array([

    [0, 0],

    [0, 1],

    [1, 0],

    [1, 1]

])

y = np.array([0, 1, 1, 0])


# Choose centers (can use k-means for real problems)

centers = np.array([

    [0, 0],

    [0, 1],
```

```
    [1, 0],
    [1, 1]
])
spread = 1.0  # σ value


# Compute the RBF layer (Phi matrix)
phi = np.zeros((len(X), len(centers)))
for i in range(len(X)):
    for j in range(len(centers)):
        phi[i, j] = rbf(X[i], centers[j], spread)
weights = np.dot(np.linalg.pinv(phi), y)
def predict(x):
    phi_x = np.array([rbf(x, c, spread) for c in centers])
    return np.dot(phi_x, weights)
for x_input in X:
    pred = predict(x_input)
    print(f"Input: {x_input}, Predicted: {pred:.3f}, Binary Output: {round(pred)}")
```

OUTPUT

```
Input: [0 0], Predicted: 0.000, Binary Output: 0
Input: [0 1], Predicted: 1.000, Binary Output: 1
Input: [1 0], Predicted: 1.000, Binary Output: 1
Input: [1 1], Predicted: -0.000, Binary Output: 0
```

RESULT

The above python program is implemented successfully.

SELF ORGANISING MAP (SOM)

Ex.No.:

Date:

AIM

To implement and visualize a **Self-Organizing Map (SOM)** for the **Iris dataset** using Python, in order to demonstrate unsupervised learning and clustering.

DATASET DESCRIPTION

- **Name**: Iris Dataset
- **Features**: 4 (sepal length, sepal width, petal length, petal width)
- **Target classes**:
- 0 = Setosa
- 1 = Versicolor
- 2 = Virginica
- **Samples**: 150 flower records (50 per class)

METHODS USED

- **Min-Max Normalization**: To scale features between 0 and 1
- **Self-Organizing Map (SOM)**:
  - Type of unsupervised neural network
  - Projects high-dimensional data into a lower-dimensional (2D) grid
  - Preserves topological relationships

LIBRARIES USED

- minisom – for implementing the Self-Organizing Map
- scikit-learn – for dataset loading and preprocessing
- matplotlib – for visualization
- numpy – for numerical operations

PROCEDURE

 Load the **Iris dataset**

 Scale the data using **MinMaxScaler**

 Initialize and train a **7x7 SOM grid**

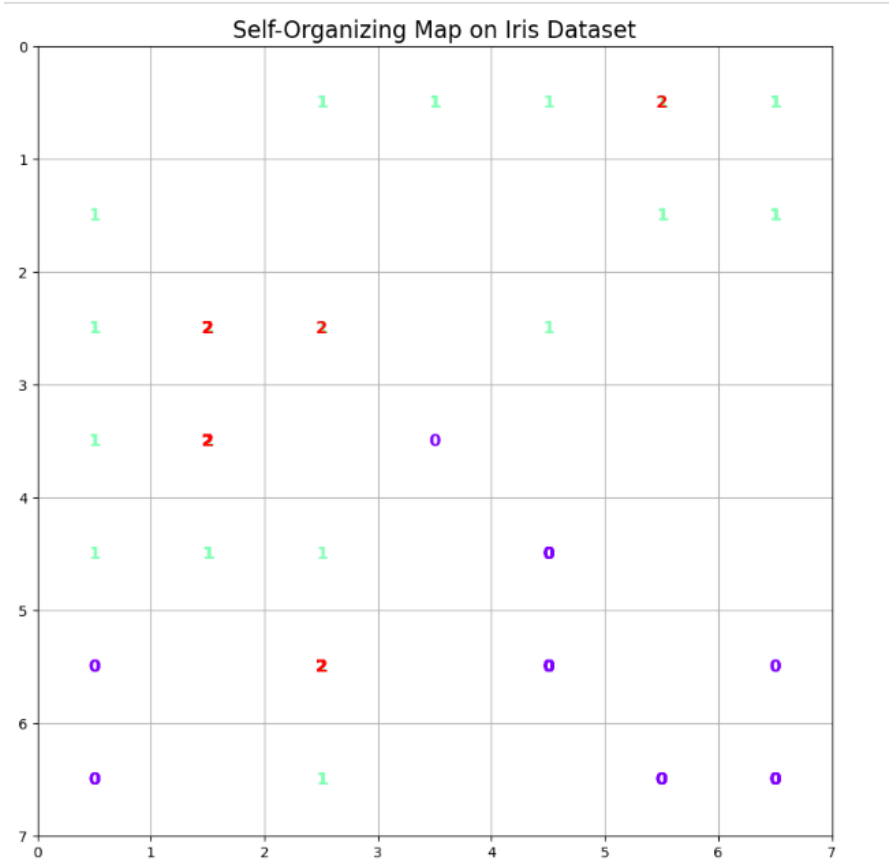
 Visualize the cluster map with label overlays

CODE

```
from minisom import MiniSom
from sklearn.preprocessing import MinMaxScaler
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
import numpy as np

# Load and scale data
data = load_iris()
X = data.data
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)

# Initialize SOM
som = MiniSom(x=7, y=7, input_len=4, sigma=1.0, learning_rate=0.5)
som.random_weights_init(X_scaled)
som.train_random(X_scaled, num_iteration=100)

# Visualize
plt.figure(figsize=(10, 10))
for i, x in enumerate(X_scaled):
    w = som.winner(x)
    plt.text(w[0] + 0.5, w[1] + 0.5, str(data.target[i]),
             color=plt.cm.rainbow(data.target[i] / 2),
```

```
        fontdict={'weight': 'bold', 'size': 12},
        ha='center', va='center')
plt.xlim([0, som.get_weights().shape[0]])
plt.ylim([0, som.get_weights().shape[1]])
plt.gca().invert_yaxis()
plt.title("Self-Organizing Map on Iris Dataset", fontsize=16)
plt.grid(True)
plt.show()
```

OUTPUT



RESULT

The SOM displays a 2D grid where similar Iris flower types are clustered together. Distinct classes (Setosa, Versicolor, Virginica) appear in separate regions, validating unsupervised learning.

DIET RECOMMENDATION SYSTEM

Ex.No.:

Date:

AIM

To implement Diet Recommendation System with UI design.

PROBLEM STATEMENT

Individuals with specific health conditions and dietary needs often struggle to identify suitable food choices that align with their medical, nutritional, and lifestyle requirements. There is a lack of personalized, easily accessible solutions that consider multiple health factors, dietary restrictions, and cuisine preferences to generate optimal diet plans.

OBJECTIVES

- To develop a personalized diet recommendation system based on user inputs such as age, gender, medical conditions, activity level, and dietary restrictions.
- To suggest nutrient-balanced food items and highlight special considerations (e.g., low sodium for hypertension).
- To generate a daily sample meal plan (breakfast, lunch, dinner) tailored to the user's chosen cuisine and health profile.

DATA ACQUISITION

Data acquisition involves collecting and organizing relevant data to train and support the diet recommendation system. The dataset used includes user health parameters, medical conditions, dietary preferences, and sample food recommendations.

KEY FEATURES COLLECTED:

Demographics: Age, Gender, Weight, Height

Medical Metrics: Blood Pressure, Cholesterol, Glucose, Condition Severity

Lifestyle & Diet: Physical Activity Level, Weekly Exercise Hours, Caloric Intake, Dietary Restrictions, Allergies, Preferred Cuisine

Output Labels: Recommended Foods, Special Considerations (e.g., low sodium), Sample Meal Plans (Breakfast, Lunch, Dinner)

PURPOSE:

To enable the model/system to learn patterns between health/lifestyle factors and ideal dietary recommendations, ensuring personalized and medically appropriate meal suggestions.

MACHINE LEARNING TECHNIQUES

Voting Classifier (Ensemble Model)

- Combines multiple models to improve prediction accuracy using soft voting

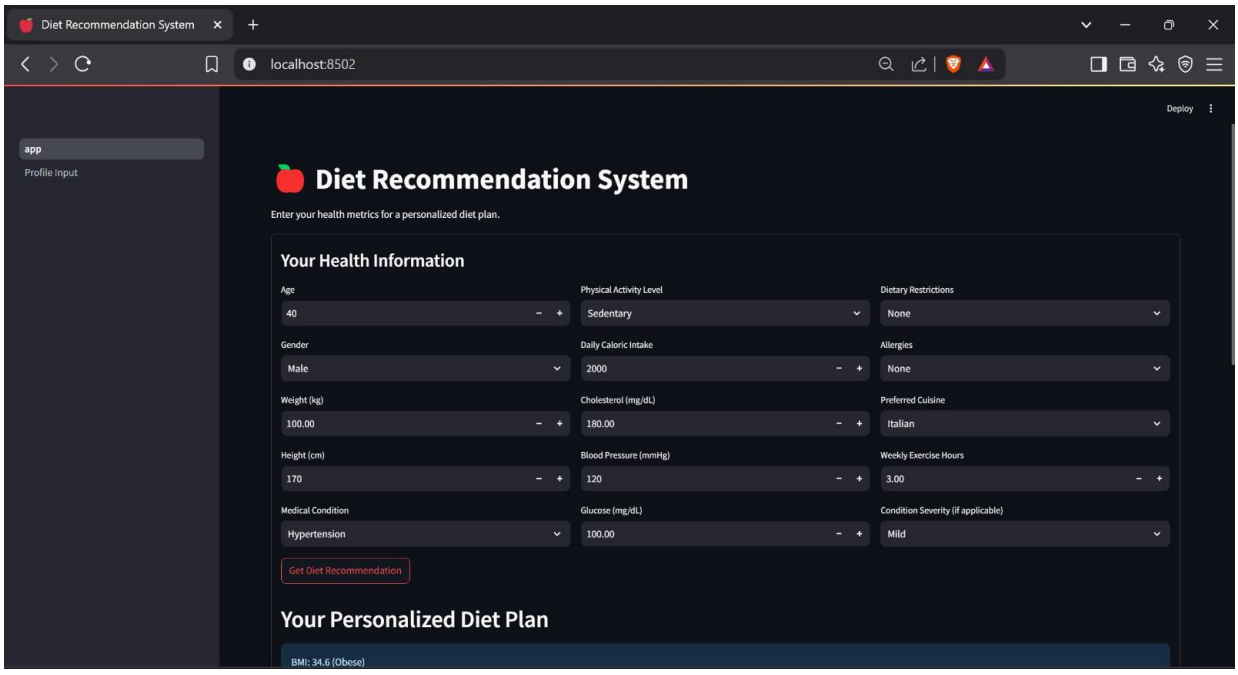
Models used:

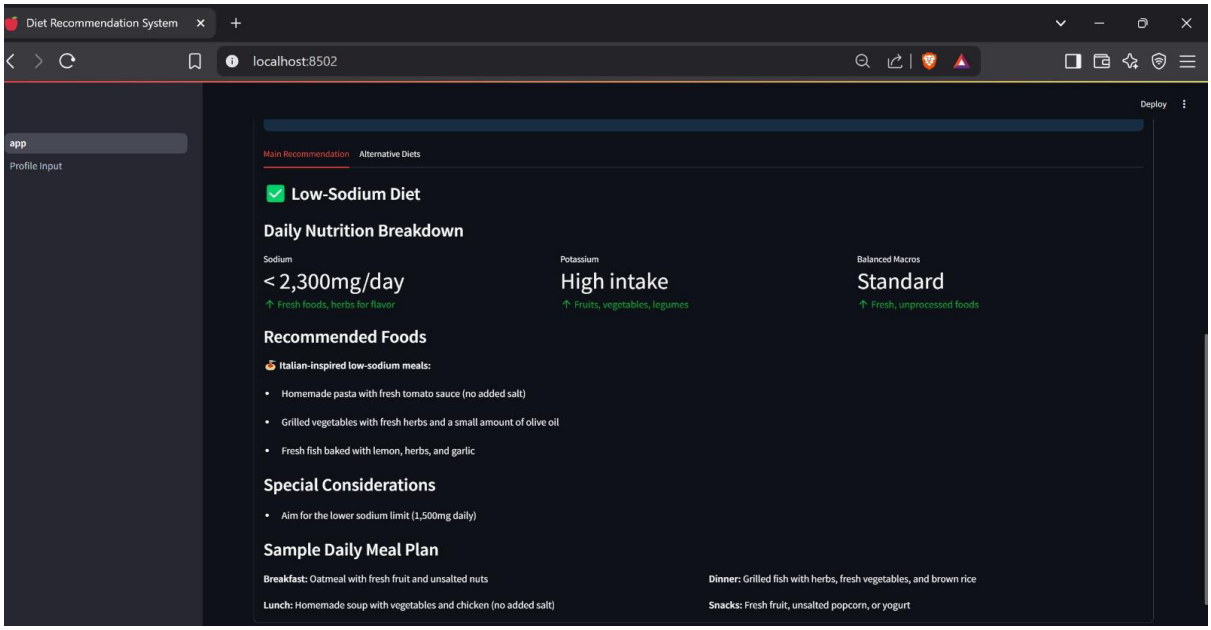
- **Voting Classifier**: Combines the predictions of the XGBoost, RandomForest, and GradientBoosting models using a "soft" voting strategy to improve prediction accuracy.
- **XGBoost Classifier**:A gradient boosting model optimized for speed and performance, used in the ensemble.
- **Random Forest Classifier:** An ensemble learning model that uses multiple decision trees for classification.
- **Gradient Boosting Classifier**: Another boosting model that focuses on optimizing weak learners to create a strong model.

Train-Test Split

- Dataset split into 80% training and 20% testing
- Stratified sampling used to maintain label distribution

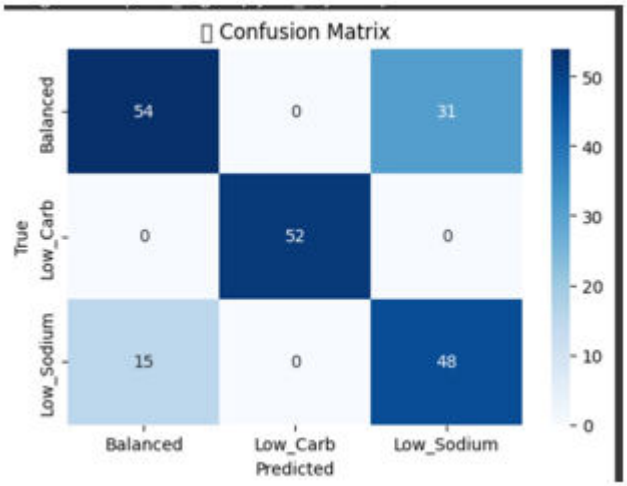
OUTPUT

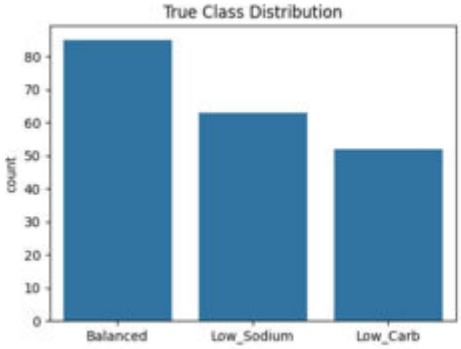


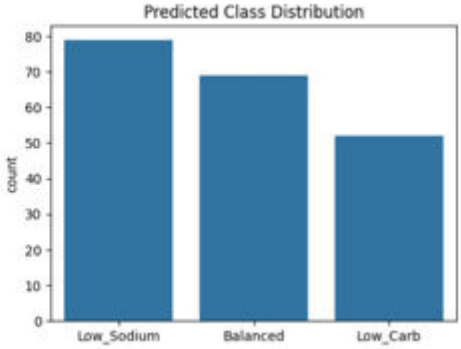


EVALUATION

```
  warnings.warn(smsg, UserWarning)
✅ Accuracy: 77.00%

📋 Classification Report:
              precision    recall  f1-score   support

    Balanced       0.78      0.64      0.70        85
    Low_Carb       1.00      1.00      1.00        52
  Low_Sodium       0.61      0.76      0.68        63

    accuracy                           0.77       200
   macro avg       0.80      0.80      0.79       200
weighted avg       0.78      0.77      0.77       200
```







Inference

The model achieved 77% accuracy, performing best on the Low_Carb class with perfect scores, while struggling slightly with Balanced and Low_Sodium due to misclassifications. Predicted class distribution skews toward Low_Sodium, as reflected in the confusion matrix.

TOOLS USED

- **Streamlit:** A Python library for creating interactive web apps for data science and machine learning projects with minimal code.
- **Pandas:** A data manipulation and analysis library in Python, offering powerful data structures like DataFrames.
- **Scikit-Learn:** A machine learning library in Python that provides simple and efficient tools for data mining and modeling.
- **XGBoost:** An optimized gradient boosting library designed for speed and performance in supervised learning tasks.
- **Joblib:** A Python library for efficient serialization and parallel computing, often used to save and load machine learning models.
- **OS:** A Python module that provides functions to interact with the operating system, such as file and directory operations.

RESULT

Thus the Diet Recommendation System was implemented successfully.