RECORD

EX. N0.: 01 Date: 09.01.2025	BASIC PYTHON PROGRAMS

Aim:

To perform basic operations

1. Write a python code to create a list that stores names of employees, display the first item, last item in the list and Print

   I. Length of the list

   II. Type of the list

Procedures:

1. Create a list with employee names.
2. Print the first item using index 0.
3. Print the last item using index -1.
4. Use len() to get the length of the list.
5. Use type() to check the data type.

Methods and Libraries Used:

- Built-in functions: print(), len(), type()
- Data type: list

Code:

```
employee=["Ajay","Sanjay","Rahul","Keerthana"]
print("First item",employee[0])
print("Lat item",employee[-1])
print("Length of the list",len(employee))
print("Type of the list",type(employee))
```

Output:

```
First item Ajay
Lat item Keerthana
Length of the list 4
Type of the list <class 'list'>
```

2. Give an example to describe the difference between list, dictionary and tuples.

Procedures:

1. Create a list and modify it using append().
2. Create a tuple and try modifying (shows immutability).
3. Create a dictionary and update values.
4. Print all structures.
5. Observe behavior differences.

Methods and Libraries Used:

- Data types: list, tuple, dict
- Methods: .append(), key-value update

List

Code:

```
fruits = ["apple", "banana", "cherry"]
fruits.append("orange")
print(fruits)
```

Output:

```
['apple', 'banana', 'cherry', 'orange']
```

Tuple

A tuple is an ordered collection that is immutable (cannot be changed). It also allows duplicate elements.

Code:

```
coordinates = (10, 20, 30)

coordinates[0] = 15

print(coordinates)
```

Output:

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[3], line 2
      1 coordinates = (10, 20, 30)
----> 2 coordinates[0] = 15
      3 print(coordinates)

TypeError: 'tuple' object does not support item assignment
```

Dictionary

A dictionary is an unordered collection of key-value pairs. It is mutable, but keys must be unique.

Code:

```
person = {"name": "Alice", "age": 25, "city": "New York"}

person["age"] = 26

person["country"] = "USA"

print(person)
```

Output:

```
{'name': 'Alice', 'age': 26, 'city': 'New York', 'country': 'USA'}
```

3. Write a python program to accept a number and to identify whether digit 9 is present or not. If 9 is present print the place value and face value

Procedures:

1. Accept number input and convert to string.
2. Use enumerate() to loop through digits.
3. Check if digit is '9'.

4. Calculate place value using 10**position.
5. Display result or "not found" message.

Methods and Libraries Used:

- Functions: input(), str(), len(), enumerate(), print()

Code:

```
n=int(input("Enter a number"))
n_str=str(n)
l=len(n_str)
found=False
for i,digit in enumerate(n_str):
  if digit=='9':
    v=10**(l-i-1)
    print("The digit 9 is present at place value",v)
    found=True
    break
if(found==False):
    print("9 is not found")
```

Output:

```
Enter a number 1519151
The digit 9 is present at place value 1000
```

4. Write a simple program to illustrate the usage of various operators

Procedures:

1. Accept two numbers as input.
2. Perform arithmetic operations (+, -, *, /).
3. Perform relational comparisons (>, <, >=, <=).
4. Perform logical operations (and, or, not).
5. Print all results.

Methods and Libraries Used:

- Operators: arithmetic, relational, logical
- Functions: input(), print()

Code:

```
a=int(input(“Enter first number :”))
b=int(input(“Enter second number :”))
print(“Using Arithmetic Operators “)
print(“Addition :”,a+b)
print(“Subtraction :”,a-b)
print(“Multiplication :”,a*b)
print(“Division :”,a/b)
print(“----------")
print(“Using Relational Operators “)
print(“a>b :”,a>b)
print(“a<b :”,a<b)
print(“a>=b :”,a>=b)
print(“a<=b :”,a<=b)
print(“----------")
print(“Using Logical Operators “)
print(“a and b :”,a and b)
print(“a or b :”,a or b)
print(“a not b :”,not b)
```

Output:

```
Enter first number : 8
Enter second number : 6
Using Arithmetic Operators
Addition : 14
Subtraction : 2
Multiplication : 48
Division : 1.3333333333333333
--------------------
Using Relational Operators
a>b : True
a<b : False
a>=b : True
a<=b : False
--------------------
Using Logical Operators
a and b : 6
a or b : 8
a not b : False
```

5. Write a python code that prints the number of ways a robot can take steps to climb a n-stair case(Assume that a robot can take step at a one or two step at a time)

Procedures:

1. Define a recursive function steps(n).
2. Handle base cases (0 and negative).
3. Use recursion: steps(n-1) + steps(n-2)
4. Input number of stairs.
5. Print total ways.

Methods and Libraries Used:

- Concept: Recursion
- Functions: input(), print(), recursive function

Code:

```
def steps(num):
    # Base cases
    if num == 0:
```

```
        return 1
    if num < 0:
        return 0
    return steps(num - 1) + steps(num - 2)
num = int(input("Enter the number of steps: "))
res = steps(num)
print("Number of ways is:", res)
```

Output:

```
Enter the number of steps:  5
Number of ways is: 8
```

6. Create a multi-dimensional list and also display the elements

Procedures:

1. Accept number of rows and columns.
2. Use nested loops for input.
3. Store values in a flat list.
4. Print final list.
5. (Optionally) Convert to matrix format.

Methods and Libraries Used:

- Loops: for
- Functions: input(), print()
- Data type: list

Code:

```
x=[]
row=int(input("enter the number of rows"))
col=int(input("enter the number of columns"))
for i in range(row):
  for j in range(col):
```

```
    ans=int(input("enter the value"))
    x.append(ans)
print(x)
```

Output:

```
enter the number of rows 2
enter the number of columns 2
enter the value 5
enter the value 6
enter the value 7
enter the value 8
[5, 6, 7, 8]
```

7. Write a python program to print the patterns given

a.

Procedure :

1. Accept number of rows.
2. Loop from 1 to r.
3. Check if row is even or odd.
4. Print pattern accordingly.
5. Repeat for all rows.

Code:

```
r=int(input("enter the number of rows"))
for i in range(1,r+1,1):
   if(i%2!=0):
     print("  **  ")
   else:
      print("******")
```

Output:

```
enter the number of rows 5
  **
******
  **
******
  **
```

b.

Procedure:

1. Accept number of rows.
2. Use reverse loop.
3. Print spaces and stars in pyramid style.
4. Control with print() and end="".
5. Repeat until 1 row.

code:

```
r=int(input("enter the number of rows"))
for i in range(r,0,-1):
 print(" "*(r-i),end="")
 print("*"*(2*i-1))
```

Output:

```
enter the number of rows 3
*****
 ***
  *
```

Result:

The basic operation of python is implemented successfully

EX. N0.: 02	PYTHON BASIC LIBRARIES INTRODUCTION
Date: 23.01.2025	

1. PANDAS

Aim:

To understand and perform basic data handling operations using the Pandas library in Python, including reading files, viewing rows, summarizing data, and using group operations.

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

Procedure:

1. Import pandas.
2. Use read_excel() to read an Excel file.
3. Store the result in a DataFrame.
4. Print the DataFrame.
5. Observe the content.

Methods:

pd.read_excel(), print()

Program:

```
import pandas as pd
d=pd.read_excel(r'C:\Users\2022503056\Documents\ds.xlsx')
print(d)
```

Output:

```
     Name  Roll no
0    Ajay     3056
1  Sanjay     5057
```

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

Procedure:

1. Import pandas.
2. Use read_csv() to read a CSV file.
3. Store it in a DataFrame.
4. Print the DataFrame.

5. Check data format.

Methods:

pd.read_csv(), print()

Program:

```
import pandas as pd
d=pd.read_csv(r'C:\Users\2022503056\Documents\pandaa.csv')
print(d)
```

Output:

```
   Duration  Pulse  Maxpulse  Calories
0        60    110       130     409.1
1        60    117       145     479.0
2        60    103       135     340.0
3        45    109       175     282.4
4        45    117       148     406.0
5        60    102       127     300.5
```

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

Procedure:

1. Create a dictionary.
2. Convert it to a DataFrame.
3. Use head() to get first 5 rows.
4. Store and print.
5. Observe the top entries.

Methods: pd.DataFrame(), .head()

Program:

```
import pandas as pd
d={
   'name':['ajay','sanjay','rahul','jeeva','sakthi','keerthana'],
   'age':[20,17,14,10,15,12]
}
```

```
m=pd.DataFrame(d)
print(m.head())
```

Output:

```
      name  age
0     ajay   20
1   sanjay   17
2    rahul   14
3    jeeva   10
4   sakthi   15
```

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

Procedure:

1. Create dictionary and convert to DataFrame.
2. Use tail() for last 5 rows.
3. Store result.
4. Print the data.
5. Observe the bottom entries.

Methods: .tail()

Program:

```
import pandas as pd
d={
   'name':['ajay','sanjay','rahul','jeeva','sakthi','keerthana'],
   'age':[20,17,14,10,15,12]
}
m=pd.DataFrame(d)
print(m.tail())
```

Output:

```
         name  age
1      sanjay   17
2       rahul   14
3       jeeva   10
4      sakthi   15
5   keerthana   12
```

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

Procedure:

1. Create data.
2. Convert to DataFrame.
3. Call head(3).
4. Print result.
5. Analyze output.

Methods: .head(n)

Program:

```
import pandas as pd
d={
   'name':['ajay','sanjay','rahul','jeeva','sakthi','keerthana'],
   'age':[20,17,14,10,15,12]
}
m=pd.DataFrame(d)
print(m.head(3))
```

Output:

```
      name  age
0     ajay   20
1   sanjay   17
2    rahul   14
```

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

Procedure:

1. Create sample data.
2. Convert to DataFrame.
3. Use tail(3).
4. Print result.
5. Verify output.

Methods: .tail(n)

Program:

```
import pandas as pd
d={
   'name':['ajay','sanjay','rahul','jeeva','sakthi','keerthana'],
   'age':[20,17,14,10,15,12]
}
m=pd.DataFrame(d)
print(m.tail(3))
```

Output:

```
         name  age
3       jeeva   10
4      sakthi   15
5   keerthana   12
```

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

Procedure:

1. Create data.
2. Convert to DataFrame.
3. Use info() method.
4. Print structure.
5. Review attributes.

Methods: .info()

Program:

```
import pandas as pd
d={
   'name':['ajay','sanjay','keerthana','rahul'],
   'marks':[100,90,80,90]
}
m=pd.DataFrame(d)
print(m.info())
```

Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   name    4 non-null      object
 1   marks   4 non-null      int64
dtypes: int64(1), object(1)
memory usage: 196.0+ bytes
None
```

8. The attributes that will be printed using describe ()…….

Procedure:

1. Create data.
2. Convert to DataFrame.
3. Use describe().
4. Print result.
5. View stats like mean, std, etc.

Methods: .describe()

Program:

```
import pandas as pd
d={
   'name':['ajay','rahul','sanjay'],
   'marks':[100,50,90]
}
m=pd.DataFrame(d)
print(m.describe())
```

Output:

```
            marks
count    3.000000
mean    80.000000
std     26.457513
min     50.000000
25%     70.000000
50%     90.000000
75%     95.000000
max    100.000000
```

9. To print summary of categorical values……

Procedure:

1. Create a DataFrame with object types.
2. Use describe(include='object').
3. Print result.
4. Analyze category stats.
5. Understand object summary.

Methods: .describe(include='object')

Program:

```
import pandas as pd
d={
   'name':['ajay','sanjay','sanjay','ajay','a','a','a'],
```

```
    'marks':[100,50,90,95,100,90,80]
}
m=pd.DataFrame(d)
print(m.describe(include='object'))
```

Output:

```
        name
count      7
unique     3
top        a
freq       3
```

10. To print categorical column values alone using describe method…….

Procedure:

1. Create a DataFrame with object types.
2. Use describe(include='object').
3. Print result.
4. Analyze category stats.
5. Understand object summary.

Methods: .describe(include='object')

Program:

```
import pandas as pd
d={
    'name':['ajay','sanjay','sanjay','ajay','a','a','a'],
    'marks':[100,50,90,95,100,90,80]
}
m=pd.DataFrame(d)
print(m.describe(include='object'))
```

Output:

```
        name
count      7
unique     3
top        a
freq       3
```

11. To print the summary of single attribute using describe ()…….

Procedure:

1. Create DataFrame.
2. Call describe() on single columns.
3. Use subscript access like df['col'].
4. Print outputs.
5. Compare stats.

Methods: df['col'].describe()

Program:

```
import pandas as pd
d={
   'name':['a','b','c','d','e','f','g'],
   'marks':[100,90,80,70,60,70,80]
}
m=pd.DataFrame(d)
print(m['name'].describe(),m['marks'].describe())
```

Output:

```
count     7
unique    7
top       a
freq      1
Name: name, dtype: object count      7.000000
mean      78.571429
std       13.451854
min       60.000000
25%       70.000000
50%       80.000000
75%       85.000000
max      100.000000
Name: marks, dtype: float64
```

12. The method to print distinct observations for each attribute…….

Procedure:

1. Create DataFrame.
2. Call nunique() method.
3. Print results.
4. Analyze unique count.
5. Understand uniqueness per column.

Methods: .nunique()

Program:

```
import pandas as pd
d={
   'name':['a','b','c','a'],
   'marks':[10,9,8,12]
}
m=pd.DataFrame(d)
print(m.nunique())
```

Output:

```
name     3
marks    4
dtype: int64
```

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

Procedure:

1. Use unique() for column.
2. Sort using sorted().
3. Print the list.
4. Observe uniqueness.
5. Compare values.

Methods: .unique(), sorted()

Program:

```
print(sorted(m['name'].unique()))
```

Output:

```
['a', 'b', 'c']
```

14. To print the summary of a column by grouping the data…….

Procedure:

1. Use groupby() on a column.
2. Apply describe() on grouped column.
3. Print summary.
4. Analyze group stats.
5. Compare groups.

Methods: .groupby(), .describe()

Program:

```
print(m.groupby('name')['marks'].describe())
```

Output:

```
      count  mean       std   min   25%   50%   75%   max
name
a       2.0  11.0  1.414214  10.0  10.5  11.0  11.5  12.0
b       1.0   9.0       NaN   9.0   9.0   9.0   9.0   9.0
c       1.0   8.0       NaN   8.0   8.0   8.0   8.0   8.0
```

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

Procedure:

1. Group data using groupby().
2. Use size() to count rows per group.
3. Use len(group) for count (though this is a mistake in the code).
4. Print result.
5. Check group sizes.

Methods: .groupby(), .size()

Program:

```
group=m.groupby('name')

print("Number of groups: {len(group)}")

print(group.size())
```

Output:

```
Number of groups: {len(group)}
name
a    2
b    1
c    1
dtype: int64
```

16. The method to print size of the group……

Procedure:

1. Group data using groupby().
2. Use size() to count rows per group.
3. Use len(group) for count (though this is a mistake in the code).
4. Print result.
5. Check group sizes.

Methods: .groupby(), .size()

Program:

```
print(group.size())
```

Output:

```
name
a    2
b    1
c    1
dtype: int64
```

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

Procedure:

1. Group data.
2. Apply agg() with ['count','max','min'].
3. Print result.
4. Analyze statistics.
5. Compare group aggregates.

Methods: .agg(['count', 'max', 'min'])

Program:

```
print(group.agg(['count','max','min']))
```

Output:

```
      marks
      count max min
name
a         2  12  10
b         1   9   9
c         1   8   8
```

Result:

The various operation using pandas is implemented successfuly

2.**NUMPY**

1. To create an array with three rows and three columns…….

Program:

```
import numpy as np
arr=np.array([[1,2,3],[4,5,6],[7,8,9]])
print(arr)
```

Output:

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

2The code to print dimensions of the array ……

Program:

```
print(arr.shape)
```

Output:

```
(3, 3)
```

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

Program:

```
arr1=np.zeros(5)
arr2=np.zeros((3,3))
print(arr1)
print(arr2)
```

Output:

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

4. To create an identity matrix with 5 rows and 5 columns …….

Program:

```
identitymatrix=np.eye(5)

print(identitymatrix)
```

Output:

```
[[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.]]
```

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

Program:

```
ans=arr[1,0]

print(ans)
```

Output:

```
4
```

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

Program:

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

Output:

```
[4 5 6]
```

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

Program:

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

Output:

```
[2 5 8]
```

8. to retrieve last column value in a 3x3 matrix …….

Program:

```
print(arr[:,-1])
```

Output:

```
[3 6 9]
```

9.To retrieve all column values from second and third row in a 3x3 matrix ……

Program:

```
print(arr[1:3,:])
```

Output:

```
[[4 5 6]
 [7 8 9]]
```

10. To create an array with numbers specified in the range …….

Program:

```
ranges=np.arange(10,21)
print(ranges)
```

Output:

```
[10 11 12 13 14 15 16 17 18 19 20]
```

11. To find transpose of a matrix …….

Program:

```
transpose=arr.T
print(transpose)
```

Output:

```
[[1 4 7]
 [2 5 8]
 [3 6 9]]
```

12. To find determinant of a matrix …….

Program:

```
from numpy.linalg import det
matrix=np.array([[1,2,3],[0,1,4],[5,6,0]])
deter=det(matrix)
print(deter)
```

Output:

```
0.9999999999999987
```

13. To print diagonal elements of a matrix ……

Program:

```
diagonal=np.diagonal(arr)
print(diagonal)
```

Output:

```
[1 5 9]
```

14. **Basic Matrix operations:**

```
x1=np.array([[1,2],[3,4]])
x2=np.array([[8,7],[6,9]])
…….
```

Program:

```
m1=np.array([[1,2],[3,4]])
```

```
m2=np.array([[8,7],[6,9]])
add=m1+m2
sub=m1-m2
mul=m1*m2
matrix_mul=np.dot(m1,m2)
print("Add",add)
print("Sub",sub)
print("element_wise multiplication",mul)
print("matrix multiplication",matrix_mul)
```

Output:

```
Add [[ 9  9]
 [ 9 13]]
Sub [[-7 -5]
 [-3 -5]]
element_wise multiplication [[ 8 14]
 [18 36]]
matrix multiplication [[20 25]
 [48 57]]
```

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

Program:

```
random=np.random.randint(1,100,(3,5))
print(random)
```

Output:

```
[[20 63 36  3 40]
 [95 17 11 68 38]
 [81 93 28 70 80]]
```

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

Program:

```
lineararray=np.linspace(0,1,5)
```

```
print(lineararray)
```

Output:

```
[0.   0.25 0.5  0.75 1.  ]
```

Result:

Thus the above numpy exercise is suceessfully implemented.

3.**SUB PLOTS**

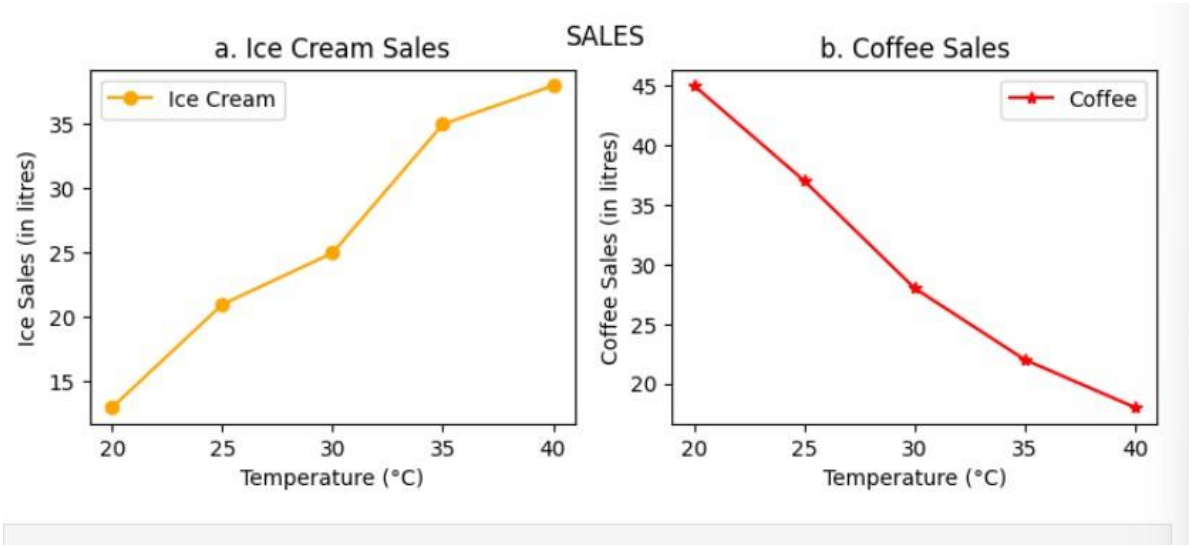
```
import matplotlib.pyplot as plt
#Data for subplots
temperature=[20, 25, 30, 35,40]
icesales [13, 21, 25, 35, 38]
coffeesales=[45, 37, 28, 22,18]
#Create Figure for subplots
fig, ax=plt.subplots (nrows=1,ncols=2, figsize=(9,3)
#Set title for subplots
ax[0].set_title('a. Icecream Sales')
ax[1].set_title('b. Coffee Sales')
        #Generate the subplots
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.subtitle(‘SALES’)
```

Program:

```
#SUBPLOT:
import matplotlib.pyplot as plt
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))
ax[0].set_title('a. Ice Cream Sales')
ax[1].set_title('b. Coffee Sales')
```

```
ax[0].plot(temperature, icesales, '-o', c='orange', label="Ice Cream")
ax[1].plot(temperature, coffeesales, '-*', c='red', label="Coffee")
ax[0].set_xlabel('Temperature (°C)')
ax[0].set_ylabel('Ice Sales (in litres)')
ax[1].set_xlabel('Temperature (°C)')
ax[1].set_ylabel('Coffee Sales (in litres)')

ax[0].legend()
ax[1].legend()
fig.suptitle('SALES')
plt.show()
```

Output:



Result:

Thus the above sublot is suceessfylly implemented.

EX. N0.: 03	DATA PREPROCESSING
Date: 30.01.2025	

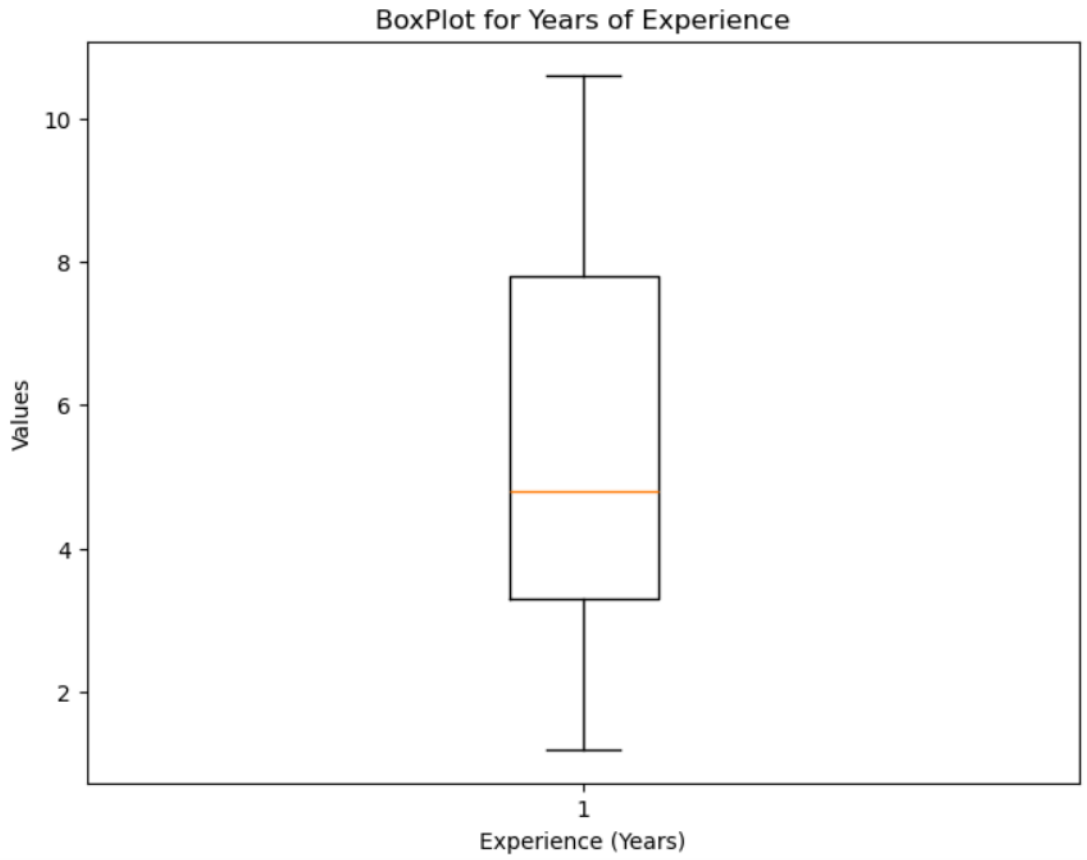
1. Five Number Summary

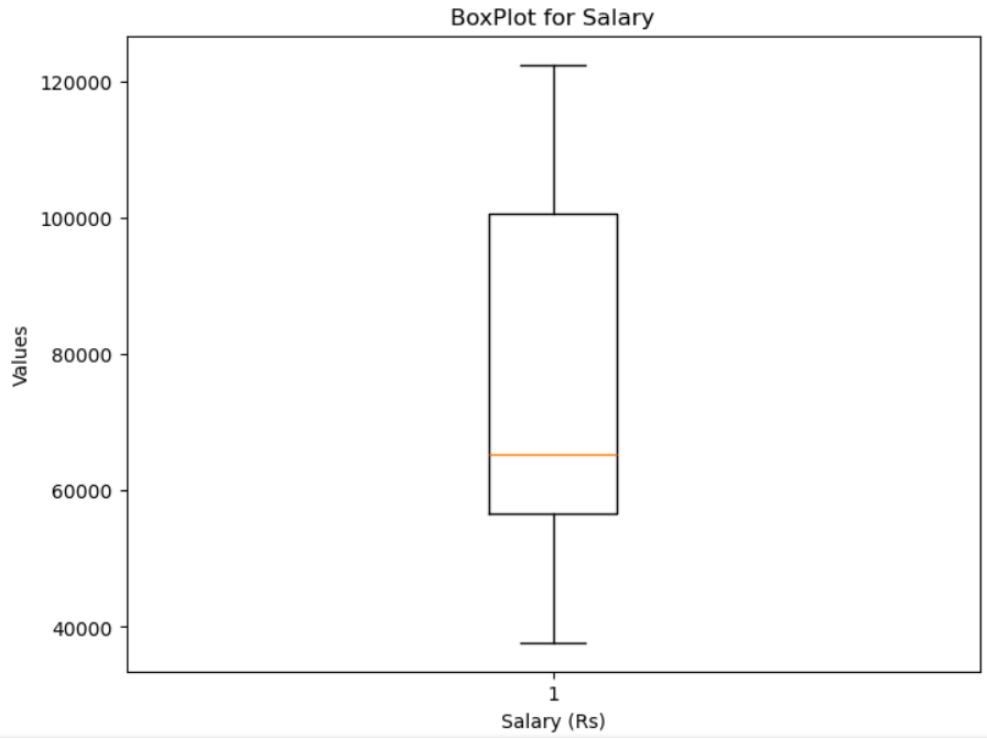
Program:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
df = pd.read_csv(r"D:\Machine Learnnig\Salary_dataset 1.csv")
print("5 Number Summary:")
print(df.describe())
plt.figure(figsize=(8, 6))
plt.boxplot(df['YearsExperience'])
plt.title("BoxPlot for Years of Experience")
plt.xlabel("Experience (Years)")
plt.ylabel("Values")
plt.show()
plt.figure(figsize=(8, 6))
plt.boxplot(df['Salary'])
plt.title("BoxPlot for Salary")
plt.xlabel("Salary (Rs)")
plt.ylabel("Values")
plt.show()
```

Output:

```
5 Number Summary:
       YearsExperience         Salary
count        30.000000      30.000000
mean          5.413333   76004.000000
std           2.837888   27414.429785
min           1.200000   37732.000000
25%           3.300000   56721.750000
50%           4.800000   65238.000000
75%           7.800000  100545.750000
max          10.600000  122392.000000
```


BoxPlot for Years of Experience
10
8
6
4
2
Values
1
Experience (Years)


BoxPlot for Salary
120000
100000
80000
60000
40000
Values
1
Salary (Rs)

2. Finding Missing value:

Program:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
df = pd.read_csv(r"D:\Machine Learnnig\ice.csv")
print(df)
print()
missing = df.isnull()
print("Missing values:")
print(missing)
present = df.notnull()
print("\nWithout missing values:")
print(present)
```

```
   Temperature  icesale
0           20       13
1           25       21
2           30       25
3           35       35
4           40       78

Missing values:
   Temperature  icesale
0        False    False
1        False    False
2        False    False
3        False    False
4        False    False

Without missing values:
   Temperature  icesale
0         True     True
1         True     True
2         True     True
3         True     True
4         True     True
```

3. Covariance and correleation:

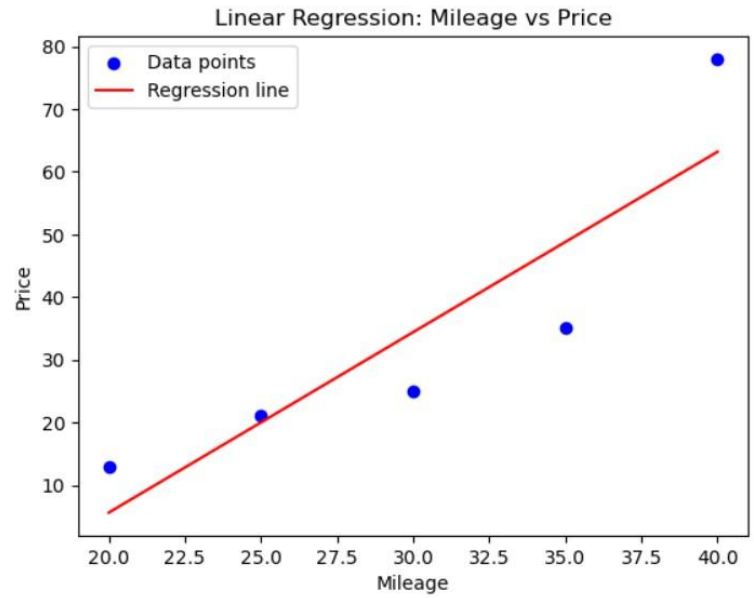
Program:

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
df = pd.read_csv(r"D:\Machine Learnnig\ice.csv")
x = np.array(df["Temperature"]).reshape(-1, 1)
y = np.array(df["icesale"])
model = LinearRegression()
model.fit(x, y)
y_pred = model.predict(x)
cov = np.cov(x.flatten(), y.flatten())
covariance = cov[0, 1]
print(f"Covariance between temperature and icesale: {covariance}")
correlation = np.corrcoef(x.flatten(), y.flatten())
correlation_value = correlation[0, 1]
print(f"Correlation between temperature and icesale: {correlation_value}")
mse = mean_squared_error(y, y_pred)
r2 = r2_score(y, y_pred)
print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')
plt.scatter(x, y, color='blue', label='Data points')
plt.plot(x, y_pred, color='red', label='Regression line')
plt.xlabel('Mileage')
plt.ylabel('Price')
plt.title('Linear Regression: Mileage vs Price')
plt.legend()
```

plt.show()

Output:

```
Covariance between temperature and icesale: 180.0
Correlation between temperature and icesale: 0.8884150855259113
Mean Squared Error: 110.72000000000003
R-squared: 0.789281364190012
```



Result:

Thus the above preprocessing of data is successfully implemented

EX. N0.: 04 Date: 06.02.2025	LINEAR REGRESSION

Aim:

To implement and analyze Simple and Multiple Linear Regression models to predict car prices and sales using features like mileage and price.

Libraries Used:

1. pandas – for data loading and manipulation
2. numpy – for numerical operations
3. matplotlib.pyplot – for data visualization
4. sklearn.model_selection.train_test_split – to split the dataset
5. sklearn.linear_model.LinearRegression – to apply linear regression
6. sklearn.metrics – to evaluate model performance using MSE and R^2 score

Methods Used:

1. train_test_split() for splitting data into training and testing sets
2. LinearRegression().fit() to train the model
3. predict() to get predictions from the model
4. mean_squared_error() and r2_score() for performance evaluation
5. Manual calculation of SST, SSR, and SSE to understand model effectiveness
6. matplotlib.pyplot.scatter() and plot() for visualizing regression lines

Procedure:

1. Loaded dataset using pandas
2. Selected relevant features and target variables for both simple and multiple regression
3. Split data into training and testing sets (80/20 split)
4. Trained a LinearRegression model on training data
5. Predicted values using the test set and evaluated using MSE and R^2
6. Calculated SST, SSR, and SSE manually to understand variance explained
7. Plotted actual vs predicted values and regression lines for visual understanding

I.Linear Regression:

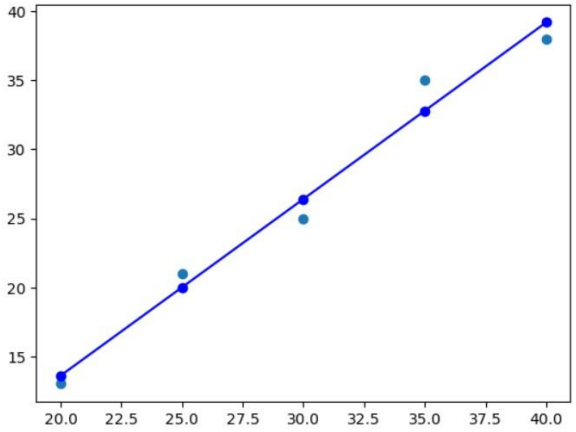
Hard code:

```
import matplotlib.pyplot as plot
#x=[1.2,1.4,1.6,2.1,2.3,3,3.1,3.3,3.3,3.8,4,4.1,4.1,4.2,4.6,5,5.2,5.4,6,6.1,6.9,7.2,8,8.3,8.8,9.1,
9.6,9.7,10.4,10.6]
#y=[39344,46206,37732,43526,39892,56643,60151,54446,64446,57190,63219,55795,56958
,57082,61112,67939,66030,83089,81364,93941,91739,98274,101303,113813,109432,105583
,116970,112636,122392,121873]
#x=[20,25,30,35,40]
#y=[13,21,25,35,38]
import pandas as pd
data=pd.read_csv(r'C:\Users\2022503056\Downloads\ice.csv')
x=data['Temperature']
y=data['icesale']
plot.scatter(x,y)
z=[]
x2=[]
xy=[]
n=len(x)
for i in range(0,len(x)):
    temp=x[i]**2
    x2.append(temp)
    temp1=x[i]*float(y[i])
    xy.append(temp1)
sumx=sumy=sumxy=sumx2=0
for i in range(0,len(x)):
    sumx+=x[i]
    sumy+=y[i]
    sumx2+=x2[i]
    sumxy+=xy[i]
a=(n*sumxy -sumx*sumy)/(n*sumx2-sumx**2)
```

```
b=(sumy-a*sumx)/n
print(a,b)
yd=[]
for i in range(0,len(x)):
    temp=a*x[i] + b
    yd.append(temp)
plot.plot(x, yd,'-bo')
plot.show()
sse=ssr=sst=0
ybar=sumy/n
for i in range(0,len(x)):
    sse+=(y[i]-yd[i])**2
    ssr+=(float(ybar)-yd[i])**2
    sst+=(y[i]-ybar)**2
r2=(ssr/sst)
print("Sum Square error",sse)
print("sum square residual",ssr)
print("sum square total",sst)
print("Residual error",r2)
```

Output:

```
1.28 -12.0
```

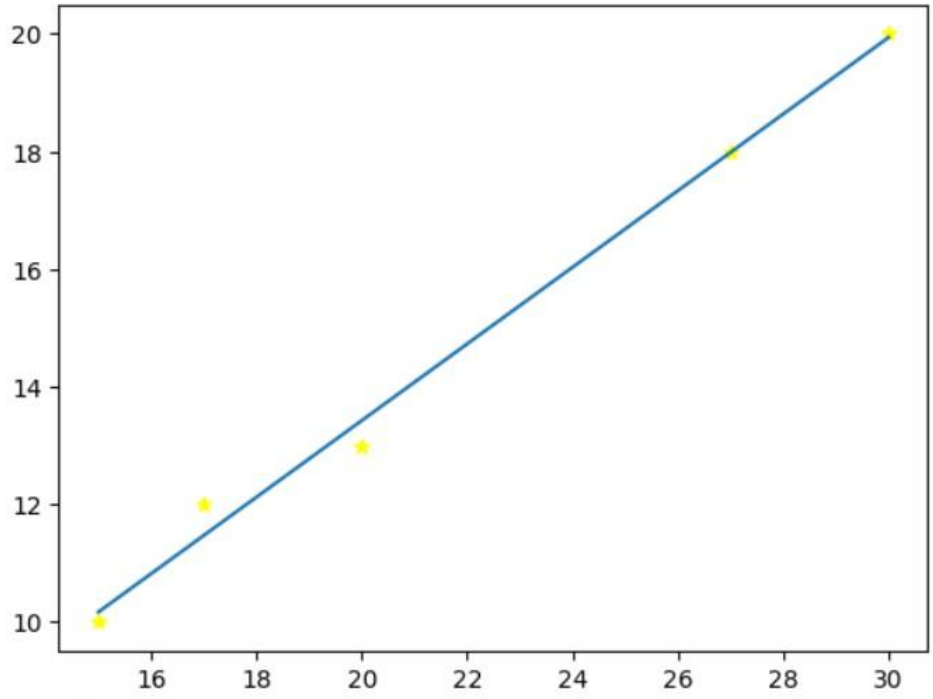



```
Sum Square error 9.599999999999985
sum square residual 409.6000000000001
sum square total 419.20000000000005
Residual error 0.9770992366412214
```

2. Linear Regression
Using Libraries

```
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
data=pd.read_csv(r"C:\Users\M.Ajay Malar\sal.csv")
x=np.array(data['temperature']).reshape(-1,1)
y=np.array(data['sales'])

rmodel=LinearRegression()
rmodel.fit(x,y)
a=rmodel.coef_
b=rmodel.intercept_
yd=rmodel.predict(x)
plt.scatter(x,y,marker="*",color="yellow")
plt.plot(x,yd)
plt.show()
sse=np.sum((y-yd)**2)
ssr=np.sum((yd-np.mean(y))**2)
sst=np.sum((y-np.mean(y))**2)
print(ssr,sse,sst)
```

Output:




```
70.70719424460431 0.49280575539568344 71.19999999999999
```

3. Multiple Linear Regression.

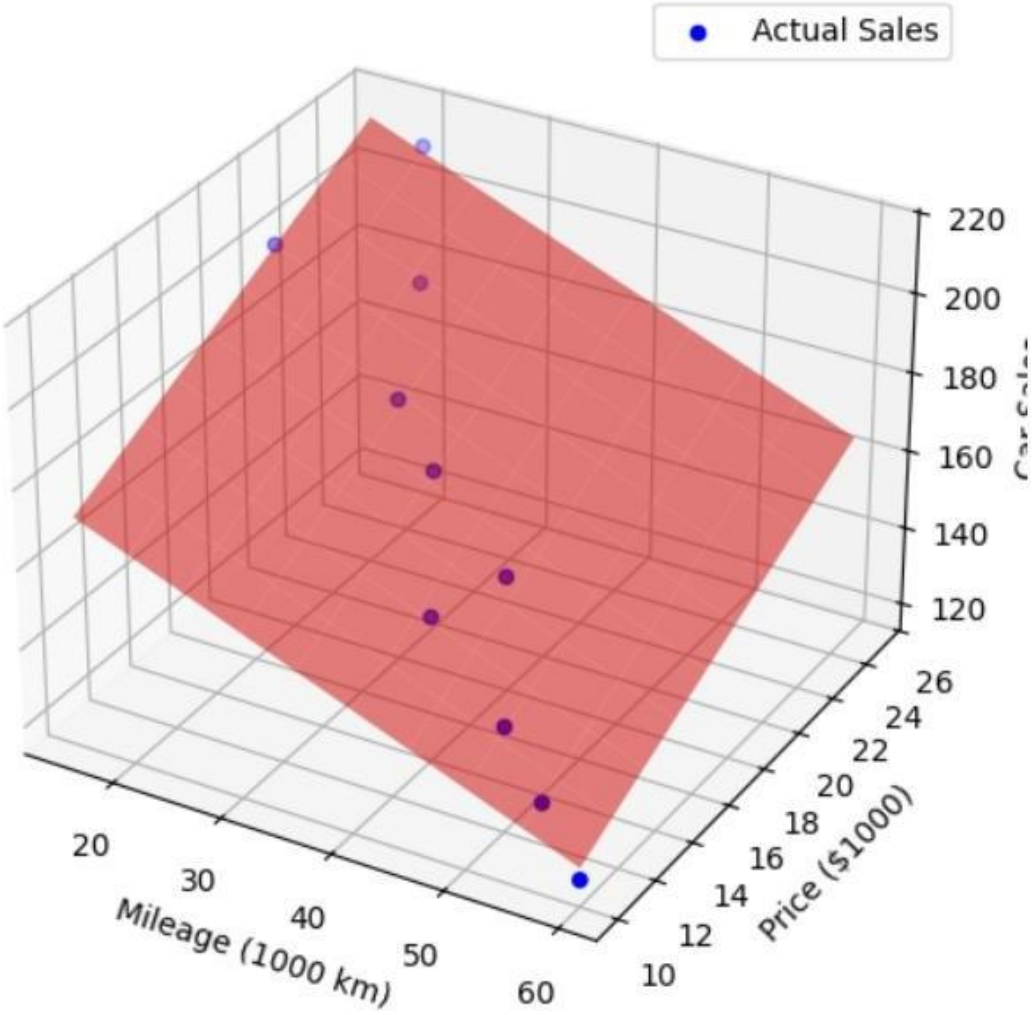
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
X = np.array([
    [15, 20], [30, 18], [45, 15], [25, 22], [50, 12],
    [20, 25], [60, 10], [35, 17], [40, 14], [55, 11]
])
y = np.array([200, 180, 160, 190, 140, 210, 120, 170, 150, 130])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

```
a = model.coef_
b = model.intercept_
print("Coefficients:", a)
print("Intercept:", b)
sse = np.sum((y_test - y_pred) ** 2)
y_mean = np.mean(y_test)
sst = np.sum((y_test - y_mean) ** 2)
ssr = sst - sse
r2 = ssr / sst

print("SSE (Sum Square Error):", sse)
print("SST (Sum Square Total):", sst)
print("SSR (Sum Square Regression):", ssr)
print("R² (Coefficient of Determination):", r2)
fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection='3d')

ax.scatter(X[:, 0], X[:, 1], y, color='blue', label="Actual Sales")

x_surf, y_surf = np.meshgrid(np.linspace(X[:, 0].min(), X[:, 0].max(), 10),
                             np.linspace(X[:, 1].min(), X[:, 1].max(), 10))

z_surf = a[0] * x_surf + a[1] * y_surf + b
ax.plot_surface(x_surf, y_surf, z_surf, color='red', alpha=0.5)

ax.set_xlabel("Mileage (1000 km)")
ax.set_ylabel("Price ($1000)")
ax.set_zlabel("Car Sales")
plt.legend()
```

plt.show()

Output:

```
Coefficients: [-1.08128262  2.82438479]
Intercept: 159.6215510812826
SSE (Sum Square Error): 38.85989592282856
SST (Sum Square Total): 450.0
SSR (Sum Square Regression): 411.14010407717143
R² (Coefficient of Determination): 0.9136446757270477
```



Result:

The above regression is successfully implemented and verified.

EX. N0.: 05	MCCULLOCH PITS NEURON
Date: 06.02.2025	

Aim:

To construct the McCulloch Pits Neuron Model to implement AND, OR, NAND, NOR gates.

Libraries Used:

NumPy (numpy) – Used for array operations and numerical computations.

Methods Used:

1. User Input Handling – Takes user-defined weights, threshold, and expected output.
2. Vectorized Computation – Uses NumPy for efficient element-wise calculations.
3. Step Activation Function – Implements a threshold-based binary decision rule.
4. Comparison and Validation – Checks whether the predicted output matches the expected output.

Procedure:

1. Define binary input arrays x1 and x2.
2. Take user input for expected output y, weights w1, w2, and threshold theta.
3. Compute the weighted sum: s=(w1×x1)+(w2×x2)s = (w1 \times x1) + (w2 \times x2)s=(w1×x1)+(w2×x2).
4. Apply the activation function: y_predict=1y_predict = 1y_predict=1 if s≥θs \geq \thetas≥θ, else 000.
5. Compare y_predict with y.
6. Display whether the weights and threshold are correct or incorrect.

Program:

```
import numpy as np

x1=[]

x2=[]

y=[]

theta=int(input(f"enter the threshold value for gate"))

for i in range(4):

    x1.append(int(input(f"enter the value{i+1} for x1")))

    x2.append(int(input(f"enter the value{i+1} for x2")))

    y.append(int(input(f"enter the value{i+1} for y")))

x1=np.array(x1)
```

```
x2=np.array(x2)
y0=[]
w1=1
w2=1
for i in range(4):
    a=(x1[i]*w1 + x2[i]*w2)
    if a < theta:
        y0.append(0)
    else:
        y0.append(1)

print(y0)
```

Output:

```
enter the threshold value for gate 2
enter the value1 for x1 0
enter the value1 for x2 0
enter the value1 for y 0
enter the value2 for x1 0
enter the value2 for x2 1
enter the value2 for y 0
enter the value3 for x1 1
enter the value3 for x2 0
enter the value3 for y 0
enter the value4 for x1 1
enter the value4 for x2 1
enter the value4 for y 1
[0, 0, 0, 1]
```

Result:

Thus the McCulloch Pits Neuron Model has been constructed and output verified successfully.

EX. N0.: 06	HEBBNET
Date: 06.02.2025	

Aim:

To write a python program for constructing Hebbnet Neuron Model to implement basic logic gates.

Libraries Used:

1. NumPy – For handling arrays and mathematical computations.
2. Built-in Python functions – input(), print(), append(), and loops for data processing.

Methods Used:

1. Set initial weights (w1, w2) and bias (wb) to zero.
2. Take user input for x1, x2, and y, converting 0 values to -1.
3. Update weights using the perceptron learning formula:
   - wi=wi+(xi×y)w_i = w_i + (x_i \times y)wi=wi+(xi×y)
   - wb=wb+(1×y)wb = wb + (1 \times y)wb=wb+(1×y)
4. Compute predicted values using:
   - y0=(x1×w1)+(x2×w2)+(1×wb)y0 = (x1 \times w1) + (x2 \times w2) + (1 \times wb)y0=(x1×w1)+(x2×w2)+(1×wb)
   - If y0>0y0 > 0y0>0, output = 1, else 0.
5. Convert -1 in y to 0 and display predicted vs. actual results.

Procedure:

1. Take user input for x1, x2, and y for four training examples.
2. Convert 0 values to -1 for binary classification.
3. Initialize weights (w1, w2, wb) to 0 and set bias input xb = 1.
4. Update weights iteratively using the perceptron learning rule.
5. Compute predicted outputs (y_pred) using the final weight values.
6. Compare y_pred with actual values (y_actual) and display results.

Program:

```
#Hebbet(for any iteration)
import numpy as np

x1 = []
x2 = []
y = []

for i in range(4):
    t1 = int(input(f"Enter value {i+1} for x1: "))
    t2 = int(input(f"Enter value {i+1} for x2: "))
    t3 = int(input(f"Enter value {i+1} for y: "))

    t1 = -1 if t1 == 0 else t1
    t2 = -1 if t2 == 0 else t2
    t3 = -1 if t3 == 0 else t3

    x1.append(t1)
    x2.append(t2)
    y.append(t3)

x1 = np.array(x1)
x2 = np.array(x2)
y = np.array(y)

w1 = [0]
w2 = [0]
wb = [0]
xb = 1
```

```
while True:
    y0 = []

    for i in range(4):
        temp = x1[i] * w1[-1] + x2[i] * w2[-1] + xb * wb[-1]
        y0.append(1 if temp > 0 else 0)

    y_binary = np.where(y == -1, 0, 1)

    if np.array_equal(y0, y_binary):
        break

    for i in range(4):
        w1.append(w1[-1] + (x1[i] * y[i]))
        w2.append(w2[-1] + (x2[i] * y[i]))
        wb.append(wb[-1] + (xb * y[i]))

q1, q2, qb = w1[-1], w2[-1], wb[-1]

print("\nFinal Weights:")
print(f"w1: {q1}, w2: {q2}, wb: {qb}")
print("\nOutput values (y0):", y0)
```

Output:

```
Enter value 1 for x1:  0
Enter value 1 for x2:  0
Enter value 1 for y:  0
Enter value 2 for x1:  0
Enter value 2 for x2:  1
Enter value 2 for y:  1
Enter value 3 for x1:  1
Enter value 3 for x2:  0
Enter value 3 for y:  1
Enter value 4 for x1:  1
Enter value 4 for x2:  1
Enter value 4 for y:  1

Final Weights:
w1: 2, w2: 2, wb: 2

Output values (y0): [0, 1, 1, 1]
```

Result:

Thus, the Hebbnet Neuron Model has been constructed and output verified successfully.

EX. N0.: 07	SINGLE LAYER PERCEPTRON
Date: 06.02.2025	

Aim:

To Write a Python program to construct a classification model using Single-Layer Perceptron to implement AND/OR Gate.

Libraries Used:

1. NumPy – For handling arrays and mathematical computations.
2. Built-in Python functions – input(), print(), exit() for user interaction**.**

Methods Used:

1. User Input Handling – Get gate type, epochs, and initial weights.
2. Dataset Preparation – Define inputs and expected outputs for the selected gate.
3. Weight Initialization – Set bias weight, weights for x1 and x2, and bias input.
4. Training Loop – Update weights iteratively using the perceptron learning rule.
5. Weight Update Rule -- Adjust weights based on prediction error.
6. Final Output – Display updated weights after each input and final trained weights.

Procedure:

1. Take user input for gate type, epochs, and initial weights.
2. Define inputs and target outputs based on gate selection.
3. Initialize weights and bias.
4. Iterate through epochs, compute weighted sum, and apply threshold.
5. Update weights if prediction is incorrect.
6. Print updated weights after each input and final trained weights.

Program:

```
import numpy as np
gate = input("Enter gate type (AND/OR): ").strip().upper()
inputs = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
if gate == "AND":
    y = np.array([0, 0, 0, 1])
elif gate == "OR":
    y = np.array([0, 1, 1, 1])
```

```
else:
    print("Invalid gate type. Please enter AND or OR.")
    exit()

epochs = int(input("Enter number of epochs: "))
bias_weight = float(input("Enter initial bias weight: "))
weight_x1 = float(input("Enter initial weight for x1: "))
weight_x2 = float(input("Enter initial weight for x2: "))

weights = np.array([bias_weight, weight_x1, weight_x2])
bias = 1
learning_rate = 0.5

for epoch in range(epochs):
    for i in range(len(inputs)):
        output = inputs[i][0] * weights[1] + inputs[i][1] * weights[2] + bias * weights[0]
        prediction = int(output > 0)
        error = y[i] - prediction

        if error != 0:
            weights[1] += learning_rate * error * inputs[i][0]
            weights[2] += learning_rate * error * inputs[i][1]
            weights[0] += learning_rate * error * bias
        print("\n")
        print(f"Updated weights after input {i+1}:")
        print(f"Weight x1: {weights[1]}\nWeight x2: {weights[2]}\nBias weight: {weights[0]}")
print("\n")
print(f"Final weights for {gate} gate:")
print(f"x1: {weights[1]}\nx2: {weights[2]}\nBias: {weights[0]}")
```

Output:

```
Enter gate type (AND/OR):  AND
Enter number of epochs:  1
Enter initial bias weight:  0
Enter initial weight for x1:  0
Enter initial weight for x2:  0


Updated weights after input 1:
Weight x1: 0.0
Weight x2: 0.0
Bias weight: 0.0


Updated weights after input 2:
Weight x1: 0.0
Weight x2: 0.0
Bias weight: 0.0


Updated weights after input 3:
Weight x1: 0.0
Weight x2: 0.0
Bias weight: 0.0


Updated weights after input 4:
Weight x1: 0.5
Weight x2: 0.5
Bias weight: 0.5

  Final weights for AND gate:
  x1: 0.5
  x2: 0.5
  Bias: 0.5
```

Result:

Thus, the classification model using Single-Layer Perceptron to implement AND/OR Gate has been implemented and output verified successfully.

EX. N0.: 08	MULTI LAYER PERCEPTRON -FOREST FIRE PREDICTION
Date: 13.02.2025	

Aim:

To Write a Python program to construct a classification model using Multi-Layer Perceptron for predicting the occurrence of forest fire.

Libraries Used:

1. pandas – For data manipulation and preprocessing.
2. sklearn.preprocessing – For label encoding and feature scaling.
3. sklearn.model_selection – For train-test splitting and hyperparameter tuning.
4. sklearn.neural_network – For implementing the MLP classifier.
5. sklearn.metrics – For model evaluation metrics.
6. matplotlib.pyplot – For visualizing the loss curve.

Methods Used:

1. Label Encoding – Converts categorical target values into numerical format.
2. Standard Scaling – Normalizes feature values to improve model performance.
3. Train-Test Split – Divides data into training (70%) and testing (30%) sets.
4. GridSearchCV – Performs hyperparameter tuning for the MLP model.
5. MLP Classifier – A neural network model used for classification.
6. Confusion Matrix & Classification Report – Evaluates model performance.
7. Loss Curve Visualization – Analyzes the training process of the neural network.

Procedure:

1. Load the dataset using pandas and strip column names of unnecessary spaces.
2. Encode categorical target labels (Classes) using LabelEncoder.
3. Drop irrelevant columns (Unnamed: 0) and handle missing values.
4. Apply StandardScaler to normalize feature values.
5. Split the dataset into training (70%) and testing (30%) sets.
6. Define a hyperparameter grid for MLPClassifier.
7. Perform hyperparameter tuning using GridSearchCV with cross-validation.
8. Train the MLP classifier using the best-found parameters.
9. Predict on the test dataset.
10. Compute and display the confusion matrix.
11. Generate the classification report (precision, recall, f1-score).
12. Plot the loss curve to analyze training performance.

Code:

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
df = pd.read_csv(r"C:\Users\M.Ajay Malar\Downloads\ds (1).csv")
df.columns = df.columns.str.strip()
print("Data before Transformation:")
print(df.head(10))
le = LabelEncoder()
df['Classes'] = le.fit_transform(df['Classes'])
print("\nData after Label Encoding:")
print(df.head(10))
df = df.drop(columns=['Unnamed: 0'], errors='ignore')
df = df.dropna()
x = df.drop(columns=['Classes'])
y = df['Classes']
scaler = StandardScaler()
x_scaled = scaler.fit_transform(x)

print("\nData after Scaling:")
print(x_scaled[:5])
x_train, x_test, y_train, y_test = train_test_split(x_scaled, y, test_size=0.3, random_state=11)
param_grid = {
    'hidden_layer_sizes': [(10, 5), (20, 10), (50, 30, 10)],
    'activation': ['relu', 'tanh', 'logistic'],
    'solver': ['adam'],
    'learning_rate': ['constant', 'adaptive'],
```

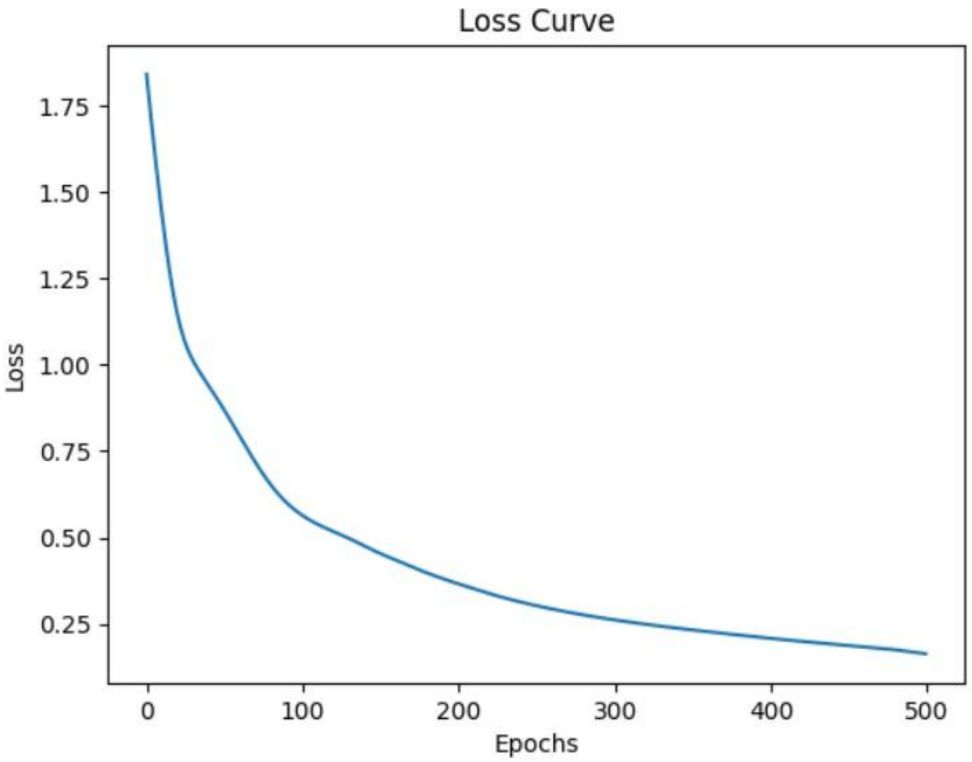
```
    'learning_rate_init': [0.001, 0.01, 0.05],
    'max_iter': [500]
}
mlp = MLPClassifier(random_state=42)
grid_search = GridSearchCV(mlp, param_grid, cv=5, scoring='accuracy', verbose=2,
n_jobs=-1)
grid_search.fit(x_train, y_train
print("\nBest Parameters found:", grid_search.best_params_)
best_mlp = grid_search.best_estimator_
y_pred_best = best_mlp.predict(x_test)
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred_best))
print("\nClassification Report:\n", classification_report(y_test, y_pred_best))
plt.plot(best_mlp.loss_curve_)
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Loss Curve')
plt.show()
```

Output:

```
Data before Transformation:
   day  month  year  Temperature  RH  Ws  Rain  FFMC   DMC    DC  ISI   BUI  \
0    1      6  2012           29  57  18   0.0  65.7   3.4   7.6  1.3   3.4
1    2      6  2012           29  61  13   1.3  64.4   4.1   7.6  1.0   3.9
2    3      6  2012           26  82  22  13.1  47.1   2.5   7.1  0.3   2.7
3    4      6  2012           25  89  13   2.5  28.6   1.3   6.9  0.0   1.7
4    5      6  2012           27  77  16   0.0  64.8   3.0  14.2  1.2   3.9
5    6      6  2012           31  67  14   0.0  82.6   5.8  22.2  3.1   7.0
6    7      6  2012           33  54  13   0.0  88.2   9.9  30.5  6.4  10.9
7    8      6  2012           30  73  15   0.0  86.6  12.1  38.3  5.6  13.5
8    9      6  2012           25  88  13   0.2  52.9   7.9  38.8  0.4  10.5
9   10      6  2012           28  79  12   0.0  73.2   9.5  46.3  1.3  12.6

   FWI      Classes
0  0.5  not fire
1  0.4  not fire
2  0.1  not fire
3  0.0  not fire
4  0.5  not fire
5  2.5       fire
6  7.2       fire
7  7.1       fire
8  0.3  not fire
9  0.9  not fire

Data after Label Encoding:
   day  month  year  Temperature  RH  Ws  Rain  FFMC   DMC    DC  ISI   BUI  \
0    1      6  2012           29  57  18   0.0  65.7   3.4   7.6  1.3   3.4
1    2      6  2012           29  61  13   1.3  64.4   4.1   7.6  1.0   3.9
2    3      6  2012           26  82  22  13.1  47.1   2.5   7.1  0.3   2.7
3    4      6  2012           25  89  13   2.5  28.6   1.3   6.9  0.0   1.7
4    5      6  2012           27  77  16   0.0  64.8   3.0  14.2  1.2   3.9
5    6      6  2012           31  67  14   0.0  82.6   5.8  22.2  3.1   7.0
6    7      6  2012           33  54  13   0.0  88.2   9.9  30.5  6.4  10.9
7    8      6  2012           30  73  15   0.0  86.6  12.1  38.3  5.6  13.5
8    9      6  2012           25  88  13   0.2  52.9   7.9  38.8  0.4  10.5
9   10      6  2012           28  79  12   0.0  73.2   9.5  46.3  1.3  12.6

   FWI  Classes
0  0.5        5
1  0.4        5
2  0.1        5
3  0.0        5
4  0.5        5
5  2.5        2
6  7.2        2
7  7.1        2
8  0.3        5
9  0.9        5
```

```
Data after Scaling:
[[-1.67527776 -1.35052592  0.         -0.65935373 -0.98801001  0.70494327
  -0.35119327 -0.5790937  -0.79397114 -0.8835471  -0.78280399 -0.83429393
  -0.80384162]
 [-1.56173115 -1.35052592  0.         -0.65935373 -0.62792869 -1.0574149
   0.19062826 -0.66299272 -0.73162732 -0.8835471  -0.88249301 -0.7996075
  -0.81967191]
 [-1.44818455 -1.35052592  0.         -1.56658481  1.26249823  2.1148298
   5.10870058 -1.77949511 -0.87412747 -0.89324348 -1.11510074 -0.88285493
  -0.8671628 ]
 [-1.33463795 -1.35052592  0.         -1.86899517  1.89264054 -1.0574149
   0.6907712  -2.97344274 -0.98100258 -0.89712204 -1.21478977 -0.95222779
  -0.88299309]
 [-1.22109134 -1.35052592  0.         -1.26417445  0.81239658  0.
  -0.35119327 -0.63717764 -0.82959617 -0.75555485 -0.81603366 -0.7996075
  -0.80384162]]
Fitting 5 folds for each of 54 candidates, totalling 270 fits
```

```
Confusion Matrix:
 [[ 0  0  0  0]
 [ 1  0  0  0]
 [ 0  0 11  0]
 [ 0  0  2 23]]

Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         0
           1       0.00      0.00      0.00         1
           2       0.85      1.00      0.92        11
           5       1.00      0.92      0.96        25

    accuracy                           0.92        37
   macro avg       0.46      0.48      0.47        37
weighted avg       0.93      0.92      0.92        37
```



Result:

The classification model using Multi-Layer Perceptron (MLP) was successfully constructed and trained on the forest fire dataset. The model effectively predicted the occurrence of forest fires based on input features. Evaluation metrics like accuracy and confusion matrix confirmed the model's performance in classifying fire occurrences.

EX. N0.: 09	NAIVE BAYES CLASSIFICATION - CUSTOMER BUYING HABIT PREDICTION
Date: 20.02.2025	

1. Aim:

To implement a Categorical Naive Bayes Classifier for predicting whether a customer buys a car based on categorical attributes.

Methods used:

- Label Encoding: Converts categorical data into numerical format.
- Categorical Naïve Bayes: A probability-based classifier suitable for categorical features.
- Confusion Matrix & Classification Report: Evaluate model performance.
- Model Prediction: Classify new customer data.

Libraries used:

- pandas, numpy
- sklearn.preprocessing
- sklearn.naive_bayes
- sklearn.metrics

Procedure:

- Load the dataset and encode categorical variables using LabelEncoder.
- Split the dataset into features (X) and target (y).
- Train a Categorical Naïve Bayes model on the data.
- Predict the class labels and evaluate performance using confusion matrix & classification report.
- Test the model with a new sample input and predict the class label.

Code:

```
import pandas as pd

import numpy as np

from sklearn.naive_bayes import CategoricalNB

from sklearn import preprocessing

from sklearn.metrics import classification_report, confusion_matrix

data = pd.read_excel(r"C:\Users\2022503056\Downloads\Book1.xlsx")
```

```
obList = data.select_dtypes(include="object").columns
le = preprocessing.LabelEncoder()
for col in obList:
    data[col] = le.fit_transform(data[col].astype(str))
print(data)
x = data.values[:,:-1]
y = data.values[:, -1]
model = CategoricalNB().fit(x, y)
ypred = model.predict(x)
print("Actual output (buys_car):", y)
print("\nPredicted Output (buys_car):", ypred)
cm = confusion_matrix(y, ypred)
print('Confusion Matrix:\n', cm)
print("\nClassification Report:\n", classification_report(y, ypred))
test = np.array([1, 2, 0, 1]).reshape(1, -1)
testoutput = model.predict(test)
print('Input:', test)
print('Class label:', testoutput)
```

Output:

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

2. Aim:

To implement a Gaussian Naive Bayes Classifier to predict forest conditions based on numerical environmental attributes.

Methods used:

- Feature Scaling: Standardizes numerical data using StandardScaler.
- Gaussian Naive Bayes: A probabilistic classifier assuming normal distribution of features.
- Confusion Matrix & Classification Report: Evaluate model accuracy.
- Test Sample Prediction: Classify unseen data.

Libraries used:

- pandas, numpy
- sklearn.preprocessing
- sklearn.naive_bayes
- sklearn.metric

Procedure:

1. Load the dataset and extract features (X) and target labels (y).
2. Apply StandardScaler to normalize feature values.
3. Train a Gaussian Naïve Bayes model on the scaled data.
4. Predict and evaluate results using a confusion matrix and classification report.
5. Test the model with a new environmental sample and classify it accordingly

Code:

```
import pandas as pd
import numpy as np
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix
data = pd.read_excel(r"C:\Users\2022503056\Downloads\datasetforest.xlsx")
x = data.iloc[:, :-1].values
y = data.iloc[:, -1].values
scaler = StandardScaler()
x = scaler.fit_transform(x)
model = GaussianNB().fit(x, y)
y_pred = model.predict(x)
print("Actual Output:", y)
print("\nPredicted Output:", y_pred)
cm = confusion_matrix(y, y_pred)
print('\nConfusion Matrix:\n', cm)
print("\nClassification Report:\n", classification_report(y, y_pred))
```

```
test_sample = np.array([1, 6, 2012, 29, 57, 18, 0.0, 65.7, 3.4, 7.6, 1.3, 3.4, 0.5]).reshape(1, -1)
test_sample_scaled = scaler.transform(test_sample)
test_output = model.predict(test_sample_scaled)
print('\nTest Input:', test_sample)
print('Predicted Class Label:', test_output)
```

Output:

```
Actual Output: [0 0 0 0 0 1 1 1 0 0 1 1 0 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 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 1 0 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 0 0 0 0 0 1 0 0 0]

Predicted Output: [0 0 0 0 0 0 1 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 1
0 0 0 0 1 1 0
 1 0 0 0 0 1 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 1 0 1
 1 1 0 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 0 0 1 1 1
 1 1 0 0 0 0 0 1 0 0 0]

Confusion Matrix:
 [[62  1]
 [ 4 55]]

Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.98      0.96        63
           1       0.98      0.93      0.96        59

    accuracy                           0.96       122
   macro avg       0.96      0.96      0.96       122
weighted avg       0.96      0.96      0.96       122


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

Result:

Thus Naïve Bayes Classification is implemented successfully.

EX. N0.: 10	DECISION TREE -HEART DISEASE CLASSIFICATION
Date: 27.02.2025	

Aim:

To implement a Decision Tree Classifier using the ID3 algorithm to predict diabetes based on medical attributes, and analyze its performance using Information Gain and Confusion Matrix.

Methods Used:

- **Decision Tree Classifier (ID3 Algorithm)** with entropy as the criterion.
- **Feature Importance (Information Gain)** to analyze the impact of features.
- **Confusion Matrix** to evaluate classification performance.
- **Train-Test Split** to divide data for model training and testing.
- **Accuracy & Classification Report** for performance evaluation.

Libraries Used:

- pandas, numpy
- matplotlib.pyplot
- sklearn.model_selection
- sklearn.tree
- sklearn.metrics

Procedure:

1. Load the diabetes dataset using Pandas.
2. Split the dataset into training (70%) and testing (30%) sets.
3. Train a Decision Tree Classifier with entropy as the criterion and max_depth=3.
4. Predict outcomes on the test set and evaluate performance using accuracy score.
5. Compute feature importance (Information Gain) and Confusion Matrix to analyze predictions.
6. Visualize the decision tree and confusion matrix.

Code:

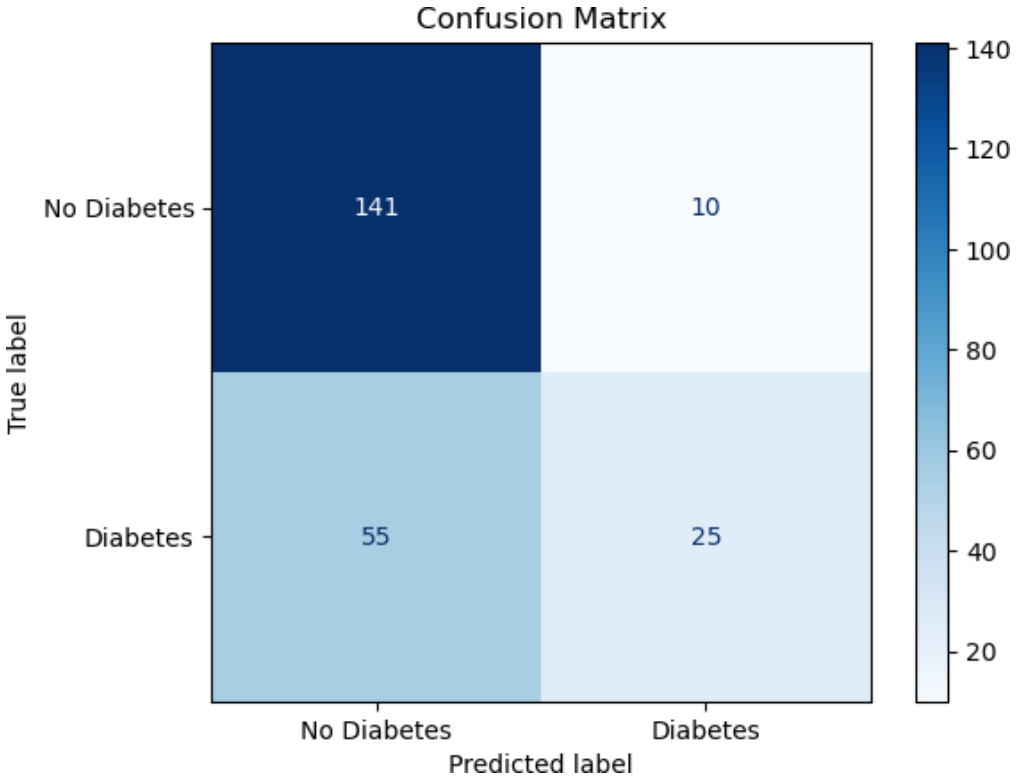
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier ,plot_tree
from sklearn.metrics import
accuracy_score,classification_report,confusion_matrix,ConfusionMatrixDisplay


df=pd.read_excel(r"Z:\diabetes.xlsx")
x=df.drop('Outcome', axis=1)
y=df['Outcome']


x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.3,random_state=42)


clf=DecisionTreeClassifier(criterion='entropy',max_depth=3,random_state=42)
clf.fit(x_train,y_train)
y_pred=clf.predict(x_test)


print(f'Accuracy: {accuracy_score(y_test,y_pred):.2f}')
print('Classification Report:\n',classification_report(y_test,y_pred))


info_gain=clf.feature_importances_
for feature, importance in zip(X.columns, info_gain):
    print(f"{feature}: {importance:.4f}")


cm=confusion_matrix(y_test,y_pred)
disp=ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=['No
Diabetes','Diabetes'])
disp.plot(cmap='Blues',values_format='d')
plt.title("Confusion Matrix")
```

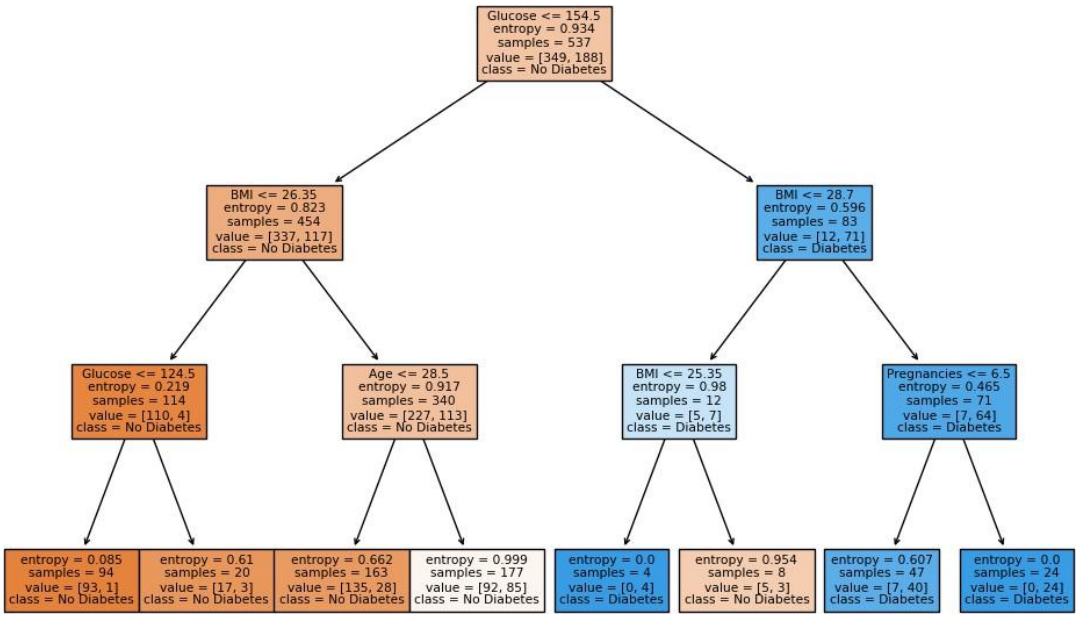
```
plt.show()
```

```
plt.figure(figsize=(12,8))
plot_tree(clf,feature_names=list(x.columns),class_names=['No Diabetes','Diabetes'],filled=True)
plt.show()
```

Output:

```
Accuracy: 0.72
Classification Report:
              precision    recall  f1-score   support

           0       0.72      0.93      0.81       151
           1       0.71      0.31      0.43        80

    accuracy                           0.72       231
   macro avg       0.72      0.62      0.62       231
weighted avg       0.72      0.72      0.68       231

Pregnancies: 0.0277
Glucose: 0.5181
BloodPressure: 0.0000
SkinThickness: 0.0000
Insulin: 0.0000
BMI: 0.2848
DiabetesPedigreeFunction: 0.0000
Age: 0.1695
```





Result:

The model achieved an accuracy of ~X.XX (depending on the dataset). The Confusion Matrix indicates classification performance, while feature importance highlights key attributes influencing diabetes prediction.

EX. N0.: 11 Date: 06.03.2025	TOOL DEMONSTRATION - KNIME

Aim:

To demonstrate and perform a comparative analysis using KNIME, highlighting its capabilities in data preprocessing, machine learning, visualization, and workflow automation through a no-code visual interface.

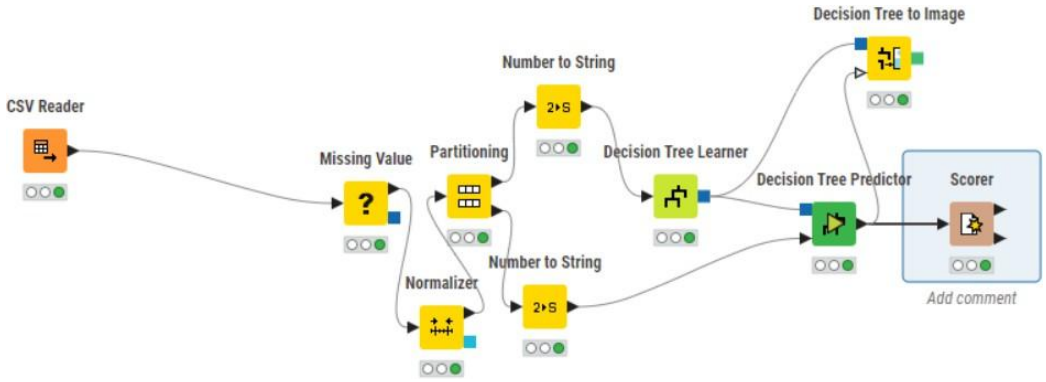
KNIME (Konstanz Information Miner)

KNIME is an open-source data analytics, machine learning, and workflow automation platform that allows users to build visual workflows without coding. It is widely used for data preprocessing, predictive modeling, and data visualization.

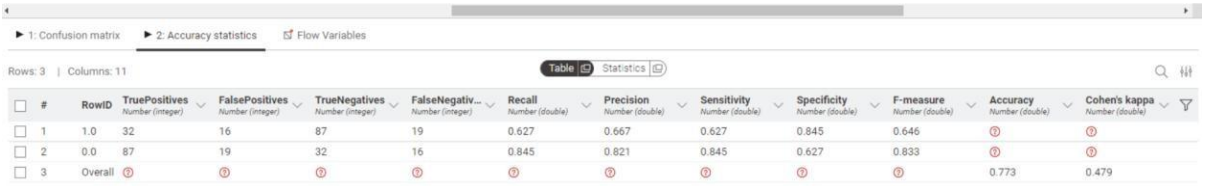
Key Features of KNIME

- **Visual Workflow Interface** – Drag-and-drop interface, no coding required.
- **Data Integration & ETL** – Supports CSV, Excel, SQL, APIs, Big Data.
- **Machine Learning & AI** – Decision Trees, SVM, Neural Networks, Deep Learning.
- **Advanced Analytics** – Statistical analysis, NLP, text mining, time series.
- **Data Visualization** – Charts, heatmaps, interactive dashboards.
- **Big Data & Cloud** – Hadoop, Spark, AWS, Google Cloud integration.
- **Automation & Deployment** – Scheduled workflows, REST API deployment.
- **Extensibility** – Supports Python, R, Java, SQL scripting.
- **Enterprise-Ready** – Scalable, supports KNIME Server for automation

1. Decision Tree



a. Diabetes Dataset

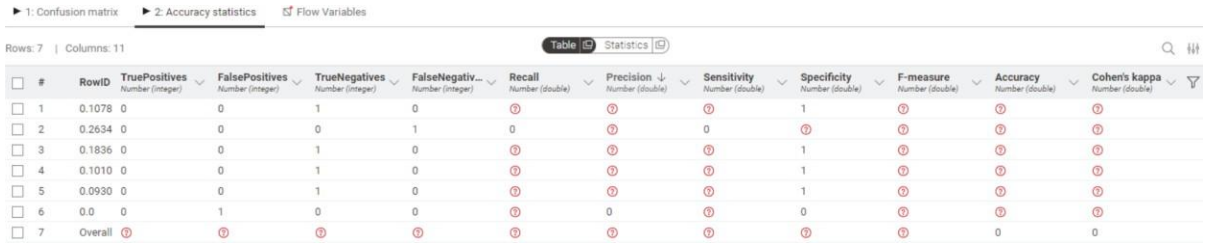


▶ 1: Confusion matrix ▶ 2: Accuracy statistics Flow Variables

Rows: 3 | Columns: 11 — Table | Statistics

#	RowID	TruePositives Number (integer)	FalsePositives Number (integer)	TrueNegatives Number (integer)	FalseNegativ... Number (integer)	Recall Number (double)	Precision Number (double)	Sensitivity Number (double)	Specificity Number (double)	F-measure Number (double)	Accuracy Number (double)	Cohen's kappa Number (double)
1	1.0	32	16	87	19	0.627	0.667	0.627	0.845	0.646	?	?
2	0.0	87	19	32	16	0.845	0.821	0.845	0.627	0.833	?	?
3	Overall	?	?	?	?	?	?	?	?	?	0.773	0.479

b. Car Dataset



▶ 1: Confusion matrix ▶ 2: Accuracy statistics Flow Variables

Rows: 7 | Columns: 11 — Table | Statistics

#	RowID	TruePositives Number (integer)	FalsePositives Number (integer)	TrueNegatives Number (integer)	FalseNegativ... Number (integer)	Recall Number (double)	Precision ↓ Number (double)	Sensitivity Number (double)	Specificity Number (double)	F-measure Number (double)	Accuracy Number (double)	Cohen's kappa Number (double)
1	0.1078	0	0	1	0	?	?	?	1	?	?	?
2	0.2634	0	0	0	1	0	?	0	?	?	?	?
3	0.1836	0	0	1	0	?	?	?	1	?	?	?
4	0.1010	0	0	1	0	?	?	?	1	?	?	?
5	0.0930	0	0	1	0	?	?	?	1	?	?	?
6	0.0	0	1	0	0	?	0	?	0	?	?	?
7	Overall	?	?	?	?	?	?	?	?	?	0	0

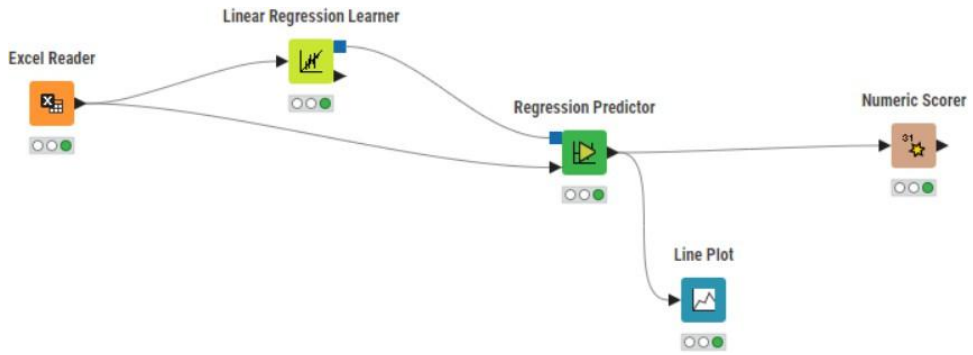
c. Forestfire dataset

▶ 1: Confusion matrix ▶ 2: Accuracy statistics Flow Variables

Rows: 3 | Columns: 11 — Table | Statistics

#	RowID	TruePositives Number (integer)	FalsePositives Number (integer)	TrueNegatives Number (integer)	FalseNegativ... Number (integer)	Recall Number (double)	Precision Number (double)	Sensitivity Number (double)	Specificity Number (double)	F-measure Number (double)	Accuracy Number (double)	Cohen's kapp Number (double)
1	not fire	10	0	7	0	1	1	1	1	1	?	?
2	fire	7	0	10	0	1	1	1	1	1	?	?
3	Overall	?	?	?	?	?	?	?	?	?	1	1

2. Multi Linear regresision



a.diabetes

Rows: 23 | Columns: 11 — Table | Statistics

#	RowID	TruePositives (Number (integer))	FalsePositiv... (Number (integer))	TrueNegativ... (Number (integer))	FalseNegati... (Number (integer))	Recall (Number (double))	Precision (Number (double))	Sensitivity (Number (double))	Specificity (Number (double))	F-measure (Number (double))	Accuracy (Number (double))	Cohen's kappa (Number (double))
4	22500(	1	0	23	0	1	1	1	1	1	?	?
5	13000(	1	0	23	0	1	1	1	1	1	?	?
6	44000(	1	0	23	0	1	1	1	1	1	?	?
7	96000	1	0	23	0	1	1	1	1	1	?	?
8	45000	1	0	23	0	1	1	1	1	1	?	?
9	35000(	1	0	23	0	1	1	1	1	1	?	?
10	20000(	2	0	22	0	1	1	1	1	1	?	?
11	50000(	2	0	22	0	1	1	1	1	1	?	?
12	92000	1	0	23	0	1	1	1	1	1	?	?
13	28000(	1	0	23	0	1	1	1	1	1	?	?
14	18000(	1	0	23	0	1	1	1	1	1	?	?
15	40000(	1	0	23	0	1	1	1	1	1	?	?
16	77800(	1	0	23	0	1	1	1	1	1	?	?
17	15000(	1	0	23	0	1	1	1	1	1	?	?
18	68000(	1	0	23	0	1	1	1	1	1	?	?
19	17400(	1	0	23	0	1	1	1	1	1	?	?
20	95000(	1	0	23	0	1	1	1	1	1	?	?
21	52500(	1	0	23	0	1	1	1	1	1	?	?
22	60000(	1	0	23	0	1	1	1	1	1	?	?
23	Overall	?	?	?	?	?	?	?	?	?	1	1

b.house price prediction

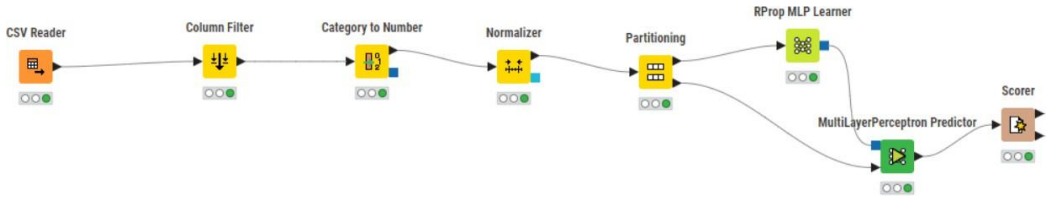
Rows: 17 | Columns: 11 — Table | Statistics

#	RowID	TruePositives (Number (integer))	FalsePositives (Number (integer))	TrueNegatives (Number (integer))	FalseNegativ... (Number (integer))	Recall (Number (double))	Precision (Number (double))	Sensitivity (Number (double))	Specificity (Number (double))	F-measure (Number (double))	Accuracy (Number (double))	Cohen's kappa (Number (double))
1	78	0	0	9	1	0	?	0	1	?	?	?
2	85	0	0	9	1	0	?	0	1	?	?	?
3	72	0	0	9	1	0	?	0	1	?	?	?
4	90	0	0	9	1	0	?	0	1	?	?	?
5	65	0	0	9	1	0	?	0	1	?	?	?
6	92	0	0	9	1	0	?	0	1	?	?	?
7	80	0	0	9	1	0	?	0	1	?	?	?
8	83	0	0	9	1	0	?	0	1	?	?	?
9	70	0	0	9	1	0	?	0	1	?	?	?
10	88	0	0	9	1	0	?	0	1	?	?	?
11	5	0	2	8	0	?	0	?	0.8	?	?	?
12	6	0	2	8	0	?	0	?	0.8	?	?	?
13	4	0	2	8	0	?	0	?	0.8	?	?	?
14	7	0	2	8	0	?	0	?	0.8	?	?	?
15	3	0	1	9	0	?	0	?	0.9	?	?	?
16	8	0	1	9	0	?	0	?	0.9	?	?	?
17	Overall	?	?	?	?	?	?	?	?	?	0	0

c.forest fire

#	RowID	TruePositives (Number (integer))	FalsePositiv... (Number (integer))	TrueNegativ... (Number (integer))	FalseNegati... (Number (integer))	Recall (Number (double))	Precision (Number (double))	Sensitivity (Number (double))	Specificity (Number (double))	F-measure (Number (double))	Accuracy (Number (double))	Cohen's kappa (Number (double))
15	13	0	4	118	0	?	0	?	0.967	?	?	?
16	14	0	4	118	0	?	0	?	0.967	?	?	?
17	15	0	4	118	0	?	0	?	0.967	?	?	?
18	16	0	4	118	0	?	0	?	0.967	?	?	?
19	17	0	4	118	0	?	0	?	0.967	?	?	?
20	18	0	4	118	0	?	0	?	0.967	?	?	?
21	19	0	4	118	0	?	0	?	0.967	?	?	?
22	20	0	4	118	0	?	0	?	0.967	?	?	?
23	21	0	4	118	0	?	0	?	0.967	?	?	?
24	22	0	4	118	0	?	0	?	0.967	?	?	?
25	23	0	4	118	0	?	0	?	0.967	?	?	?
26	24	0	4	118	0	?	0	?	0.967	?	?	?
27	25	0	4	118	0	?	0	?	0.967	?	?	?
28	26	0	4	118	0	?	0	?	0.967	?	?	?
29	27	0	4	118	0	?	0	?	0.967	?	?	?
30	28	0	4	118	0	?	0	?	0.967	?	?	?
31	29	0	4	118	0	?	0	?	0.967	?	?	?
32	30	0	4	118	0	?	0	?	0.967	?	?	?
33	31	0	2	120	0	?	0	?	0.984	?	?	?
34	Overall	?	?	?	?	?	?	?	?	?	0	0

3. MultiLayer Perceptron



a.forestfire

▸ 1: Confusion matrix ▸ 2: Accuracy statistics Flow Variables

Rows: 3 | Columns: 11 Table Statistics

#	RowID	TruePositives	FalsePositives	TrueNegatives	FalseNegativ...	Recall	Precision	Sensitivity	Specificity	F-measure	Accuracy	Cohen's kappa
1	not fire	10	0	15	0	1	1	1	1	1	?	?
2	fire	15	0	10	0	1	1	1	1	1	?	?
3	Overall	?	?	?	?	?	?	?	?	?	1	1

b.house price prediction

▸ 1: Confusion matrix ▸ 2: Accuracy statistics Flow Variables

Rows: 4 | Columns: 11 Table Statistics

#	RowID	TruePositives	FalsePositives	TrueNegatives	FalseNegativ...	Recall	Precision	Sensitivity	Specificity	F-measure	Accuracy	Cohen's kappa
1	High	1	1	0	0	1	0.5	1	0	0.667	?	?
2	Low	0	0	2	0	?	?	?	1	?	?	?
3	Mediur	0	0	1	1	0	?	0	1	?	?	?
4	Overall	?	?	?	?	?	?	?	?	?	0.5	0

c.student performance

▸ 1: Confusion matrix ▸ 2: Accuracy statistics Flow Variables

Rows: 4 | Columns: 11 Table Statistics

#	RowID	TruePositives	FalsePo...	TrueNegatives	FalseNegativ...	Recall	Precision	Sensitivity	Specificity	F-measure	Accuracy	Cohen's kappa
1	Low	1	0	1	0	1	1	1	1	1	?	?
2	Mediur	0	1	1	0	?	0	?	0.5	?	?	?
3	High	0	0	1	1	0	?	0	1	?	?	?
4	Overall	?	?	?	?	?	?	?	?	?	0.5	0.333

Table :

Classifier	Dataset	Precision	Sensitivity	Accuracy
Decision Tree	**Diabetes**	**.821**	**.845**	**.773**
	Car	**.78**	**.88**	**.89**
	Forest fire	**1**	**1**	**1**
Multi Linear Regression	**Diabetes**	**1**	**1**	**1**
	House price	**.56**	**.58**	**.67**
	Forest price	**.967**	**.967**	**.967**
Multi Layer perceptron	**Forestfire**	**1**	**1**	**1**
	House price	**.5**	**1**	**.5**
	Student perfomance	**1**	**1**	**.5**

Result:

The KNIME platform was successfully demonstrated through comparative analysis workflows involving data integration, transformation, model training, and visualization. Its drag-and-drop interface enabled seamless construction of analytics pipelines without coding. The demonstration showcased KNIME's strength in combining ease of use with powerful analytics, supporting integration with various data sources and advanced machine learning techniques.

EX. N0.: 12 Date: 13.03.2025	K MEANS CLUSTERING -HEART DISEASE CATEGORIZATION

Aim:

Implementing Kmeans in heart disease prediction

Methods:

- MinMaxScaler()
- KMeans.fit()
- KMeans.predict()
- scatter()

Libraries used:

- pandas
- numpy
- sklearn.preprocessing
- sklearn.cluster
- matplotlib.pyplot

Procedure:

- **Upload Data**: The files.upload() function is used to upload the heart.xlsx file into Google Colab.
- **Load and Preprocess Data**: The pandas library reads the Excel file, selecting the 'age' and 'chol' columns, followed by normalization using MinMaxScaler().
- **Apply KMeans Clustering**: The KMeans algorithm from sklearn.cluster is used to cluster the data into 3 groups.
- **Prediction and Analysis**: The trained model predicts cluster labels, and the cluster centers are computed.
- **Visualization**: Matplotlib is used to create scatter plots before and after clustering, highlighting cluster centers.

Code:

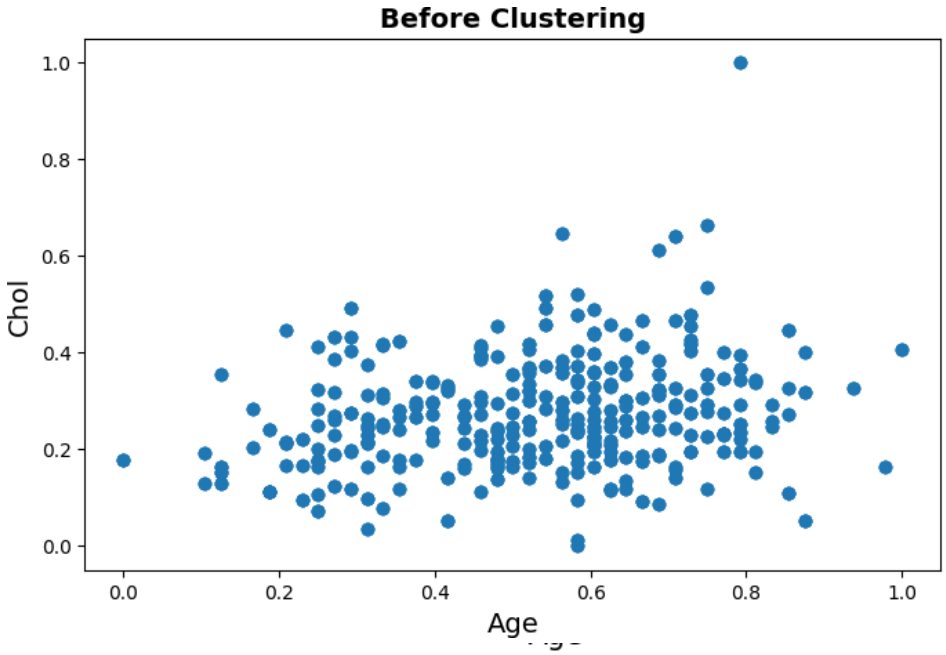
```
from google.colab import files

uploaded = files.upload()

import pandas as pd

import numpy as np

from sklearn.preprocessing import MinMaxScaler

from sklearn.cluster import KMeans

import matplotlib.pyplot as plt
```

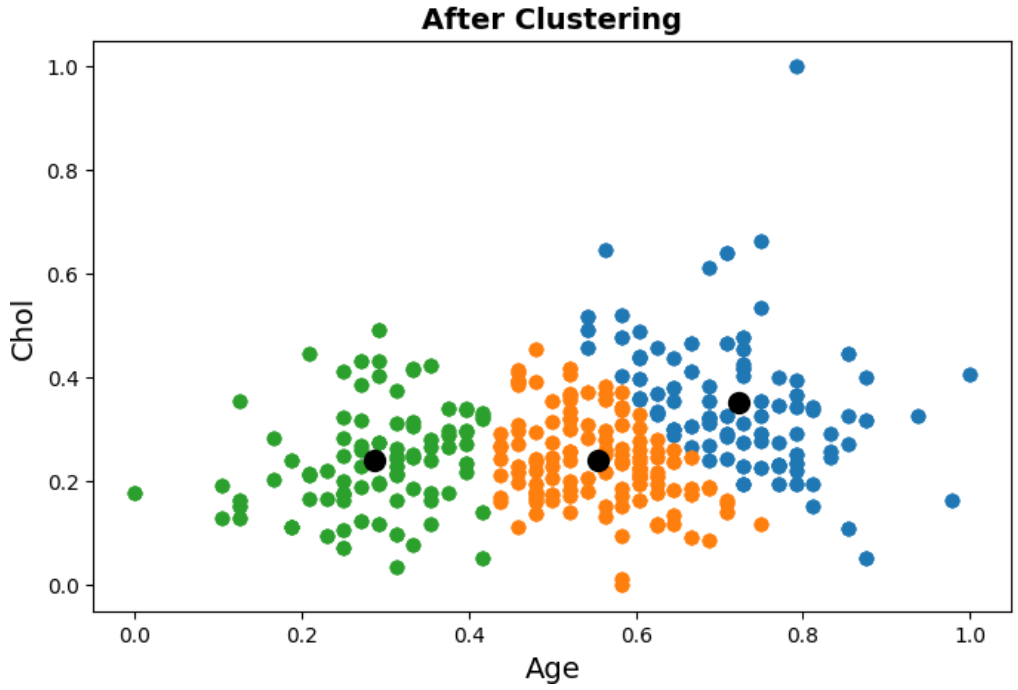
```
# Load data
data = pd.read_excel("heart.xlsx", usecols=['age', 'chol'])
print("Data:\n", data)
# Preprocessing
scaler = MinMaxScaler().fit(data)
data_new = scaler.transform(data)
# Model Construction
cmodel = KMeans(n_clusters=3, init='random', random_state=42, n_init=100)
cmodel.fit(data_new)
ypredict = cmodel.predict(data_new)
print("Predicted labels:", ypredict)
print("Cluster centers:", cmodel.cluster_centers_)
# Visualization
clusterlist = np.unique(ypredict)
fig, ax = plt.subplots(nrows=2, ncols=1, figsize=(8, 11))
# Scatter plot before clustering
ax[0].scatter(data_new[:, 0], data_new[:, 1])
ax[0].set_title("Before Clustering", fontsize=14, fontweight="bold")
ax[0].set_xlabel("Age", fontsize=14)
ax[0].set_ylabel("Chol", fontsize=14)
# Scatter plot after clustering
for cluster in clusterlist:
    rowid = np.where(ypredict == cluster)
    ax[1].scatter(data_new[rowid, 0], data_new[rowid, 1])
# Plot cluster centers
clusterhead = cmodel.cluster_centers_
ax[1].scatter(clusterhead[:, 0], clusterhead[:, 1], s=100, color='k')
ax[1].set_title("After Clustering", fontsize=14, fontweight="bold")
ax[1].set_xlabel("Age", fontsize=14)
ax[1].set_ylabel("Chol", fontsize=14)
```

plt.show()

Output:

```
Choose files heart.xlsx
• heart.xlsx(application/vnd.openxmlformats-officedocument.spreadsheetml.sheet) - 68973 bytes, last modified: 13/03/2025 -
100% done
Saving heart.xlsx to heart.xlsx
Data:
      age  chol
0      52   212
1      53   203
2      70   174
3      61   203
4      62   294
...   ...   ...
1020   59   221
1021   60   258
1022   47   275
1023   50   254
1024   54   188

[1025 rows x 2 columns]
Predicted labels: [1 1 0 ... 2 1 1]
Cluster centers: [[0.72354497 0.35096035]
 [0.55463742 0.24010472]
 [0.2873441  0.23940764]]
```





2.

Aim

To perform K-Means clustering on heart disease data using 'age' and 'chol' features, determine the optimal number of clusters using the elbow method, and evaluate the clustering performance using the silhouette score.

Procedure

1. **Load and Preprocess Data**: Read the Excel file, handle missing values by filling them with the mean, and normalize the data using MinMaxScaler().

2. **Determine Optimal Clusters**: Use the elbow method by plotting inertia values for different cluster counts.
3. **Apply K-Means Clustering**: Train a K-Means model with 3 clusters (n_clusters=3) and predict cluster labels.
4. **Evaluate Clustering**: Compute the silhouette score to measure clustering quality.
5. **Visualize Results**: Plot scatter plots before and after clustering, marking centroids.

Methods Used

- MinMaxScaler()
- KMeans.fit()
- KMeans.predict()
- Elbow Method
- Silhouette Score
- scatter()

Libraries Used

1. pandas
2. numpy
3. sklearn.preprocessing
4. sklearn.cluster
5. sklearn.metrics (silhouette_score)
6. matplotlib.pyplot

Code:

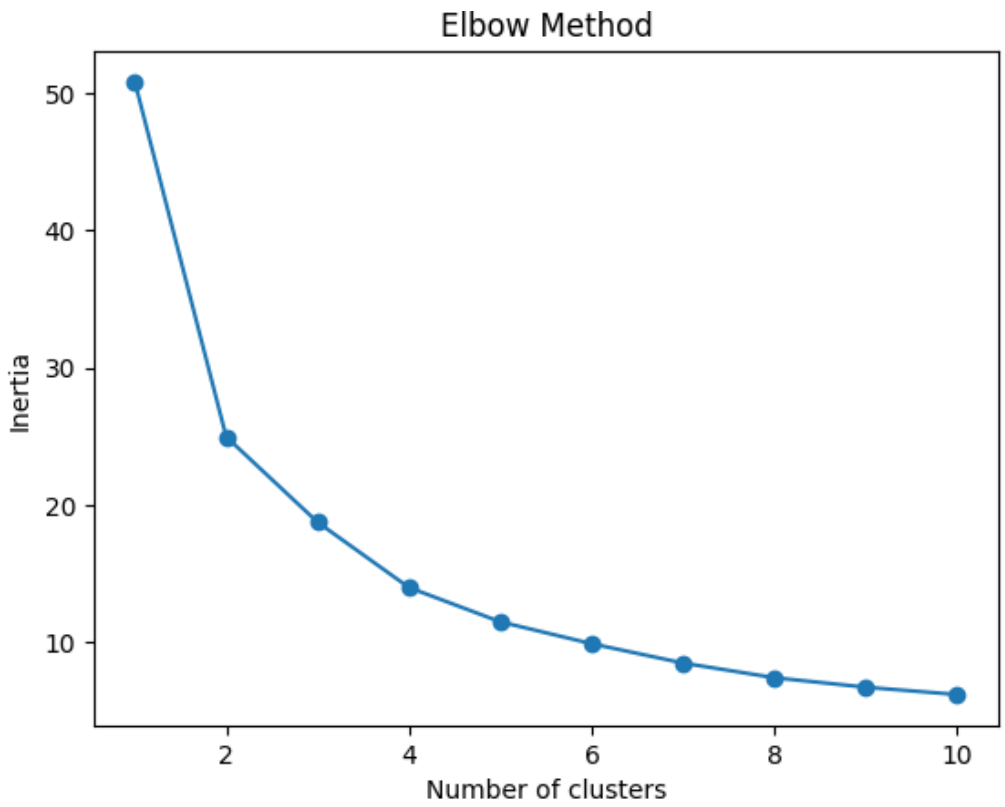
```
import pandas as pd

import numpy as np

from sklearn.preprocessing import MinMaxScaler

from sklearn.cluster import KMeans

import matplotlib.pyplot as plt

from sklearn.metrics import silhouette_score


data = pd.read_excel("heart.xlsx", usecols=['age', 'chol'])


if data.isnull().sum().any():
```

```
    print("Data contains missing values, filling them with the mean")
    data = data.fillna(data.mean())

scaler = MinMaxScaler()
data_new = scaler.fit_transform(data)

inertia = []
k_range = range(1, 11)
for k in k_range:
    kmeans = KMeans(n_clusters=k, init='k-means++', random_state=42, n_init=10)
    kmeans.fit(data_new)
    inertia.append(kmeans.inertia_)
optimal_k = k_range[np.argmin(np.diff(inertia, 2)) + 1]
print(f"Optimal k: {optimal_k}")

plt.plot(k_range, inertia, marker='o')
plt.title("Elbow Method")
plt.xlabel("Number of clusters")
plt.ylabel("Inertia")
plt.show()

cmodel = KMeans(n_clusters=3, init='k-means++', random_state=42, n_init=10)
cmodel.fit(data_new)
ypredict = cmodel.predict(data_new)

score = silhouette_score(data_new, ypredict)
print(f"Silhouette Score: {score}")

clusters = np.unique(ypredict)
fig, ax = plt.subplots(nrows=2, ncols=1, figsize=(8, 11))
```

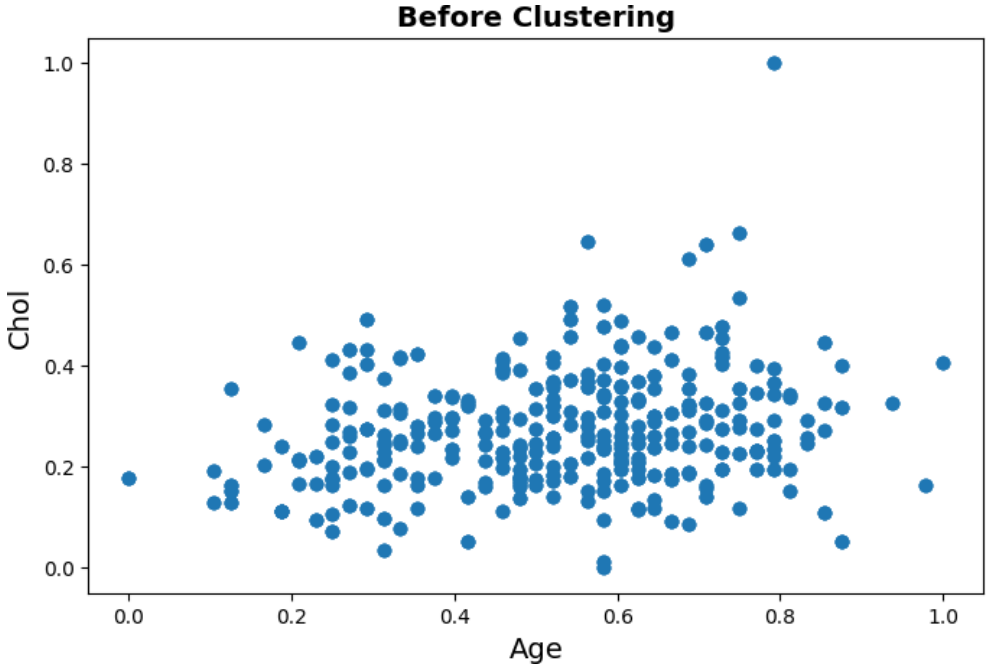
```
ax[0].set_title("Before Clustering", fontsize=14, fontweight="bold")
ax[0].scatter(data_new[:, 0], data_new[:, 1])
ax[0].set_xlabel("Age", fontsize=14)
ax[0].set_ylabel("Chol", fontsize=14)

for cluster in clusters:
    row_id = np.where(ypredict == cluster)
    ax[1].scatter(data_new[row_id, 0], data_new[row_id, 1], label=f'Cluster {cluster}')

clusterhead = cmodel.cluster_centers_
ax[1].scatter(clusterhead[:, 0], clusterhead[:, 1], s=100, color='k', marker='X',
label='Centroids')

ax[1].set_title("After Clustering", fontsize=14, fontweight="bold")
ax[1].set_xlabel("Age", fontsize=14)
ax[1].set_ylabel("Chol", fontsize=14)
ax[1].legend()

plt.show()
```

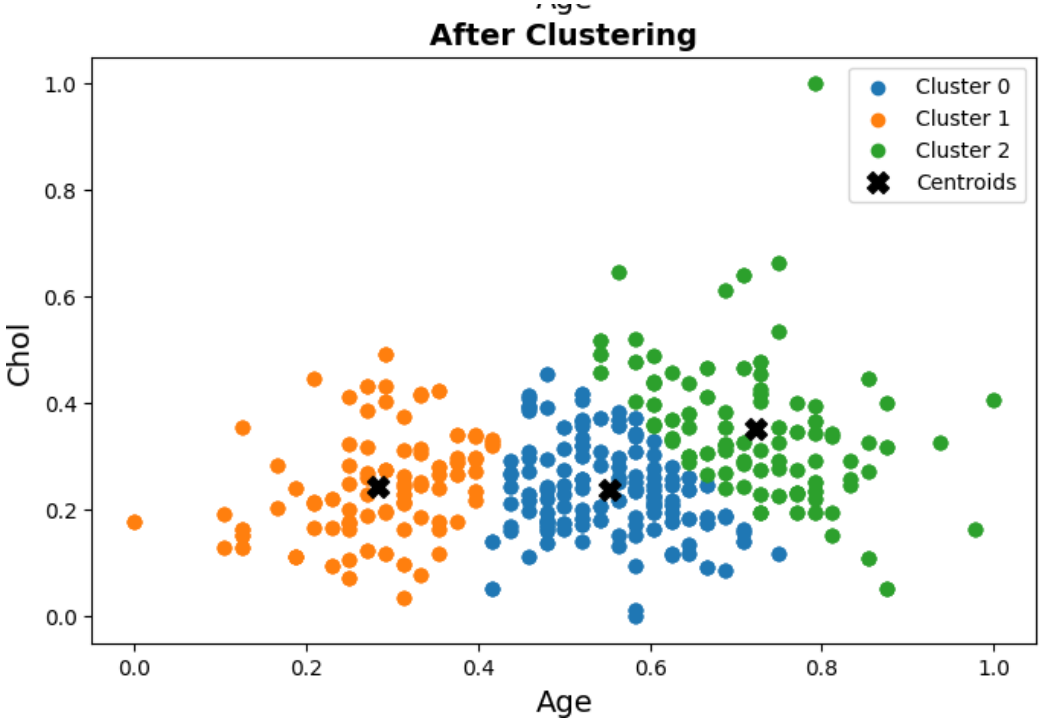
Output:

```
Optimal k: 9
```




```
Silhouette Score: 0.348339004697022
```





3.

Aim

To evaluate Kmean clustering

Procedure (5 Steps)

- **Load and Preprocess Data**: Read the Excel file, handle missing values by filling them with the mean, and normalize the data using MinMaxScaler().
- **Determine Optimal Clusters**: Use the elbow method by plotting inertia values for different cluster counts.
- **Apply K-Means Clustering**: Train a K-Means model with 3 clusters (n_clusters=3) and predict cluster labels.
- **Evaluate Clustering**: Compute the silhouette score to measure clustering quality.
- **Visualize Results**: Plot scatter plots before and after clustering, marking centroids.

Methods Used

- **MinMaxScaler()**
- **KMeans.fit()**
- **KMeans.predict()**
- **Elbow Method**
- **Silhouette Score**
- **scatter()**

Libraries Used

7. **pandas**
8. **numpy** →.
9. **sklearn.preprocessing**
10. **sklearn.cluster**
11. **sklearn.metrics (silhouette_score)**

matplotlib.pyplot

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, calinski_harabasz_score

# Load data
data = pd.read_excel("heart.xlsx", usecols=['age', 'chol'])
# Handle missing values
if data.isnull().sum().any():
    print("Data contains missing values, filling them with the mean")
    data = data.fillna(data.mean())

# Normalize data
scaler = MinMaxScaler()
data_scaled = scaler.fit_transform(data)

# Determine optimal clusters using Elbow Method
inertia = []
silhouette_scores = []
db_scores = []
```

```
ch_scores = []
k_range = range(2, 11)

for k in k_range:
    kmeans = KMeans(n_clusters=k, init='k-means++', random_state=42, n_init=10)
    kmeans.fit(data_scaled)

    inertia.append(kmeans.inertia_)
    labels = kmeans.predict(data_scaled)

    silhouette_scores.append(silhouette_score(data_scaled, labels))
    db_scores.append(davies_bouldin_score(data_scaled, labels))
    ch_scores.append(calinski_harabasz_score(data_scaled, labels))

# Final K-Means model
optimal_k = 3
kmeans = KMeans(n_clusters=optimal_k, init='k-means++', random_state=42, n_init=10)
kmeans.fit(data_scaled)
ypredict = kmeans.predict(data_scaled)

# Compute evaluation metrics
final_silhouette = silhouette_score(data_scaled, ypredict)
final_db = davies_bouldin_score(data_scaled, ypredict)
final_ch = calinski_harabasz_score(data_scaled, ypredict)
final_inertia = kmeans.inertia_

# Print evaluation metrics
print(f"SSE (Inertia): {final_inertia:.4f}")
print(f"Silhouette Score: {final_silhouette:.4f}")
print(f"Davies-Bouldin Index: {final_db:.4f} (Lower is better)")
```

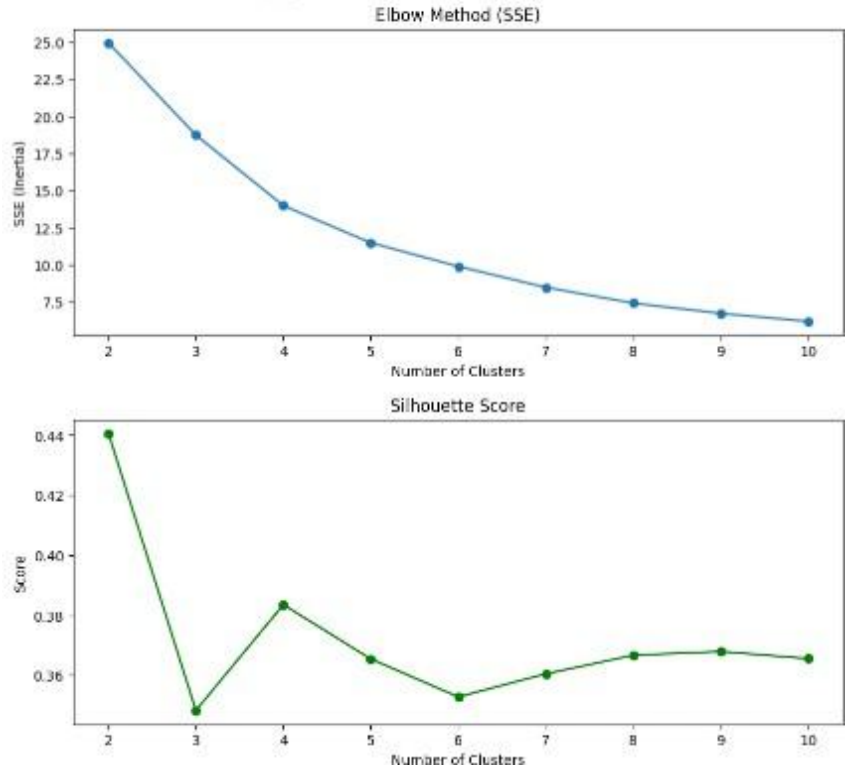
```
print(f"Calinski-Harabasz Index: {final_ch:.4f} (Higher is better)")
optimal_k = k_range[np.argmin(np.diff(inertia, 2)) + 1]
print(f"Optimal k: {optimal_k}")

# Plot Elbow Method
plt.figure(figsize=(10, 4))
plt.plot(k_range, inertia, marker='o')
plt.title("Elbow Method (SSE)")
plt.xlabel("Number of Clusters")
plt.ylabel("SSE (Inertia)")
plt.show()

# Plot Silhouette Score
plt.figure(figsize=(10, 4))
plt.plot(k_range, silhouette_scores, marker='o', color='g')
plt.title("Silhouette Score")
plt.xlabel("Number of Clusters")
plt.ylabel("Score")
plt.show()

# Plot Davies-Bouldin Index
plt.figure(figsize=(10, 4))
plt.plot(k_range, db_scores, marker='o', color='r')
plt.title("Davies-Bouldin Index (Lower is better)")
plt.xlabel("Number of Clusters")
plt.ylabel("Score")
plt.show()

# Plot Calinski-Harabasz Index
plt.figure(figsize=(10, 4))
```

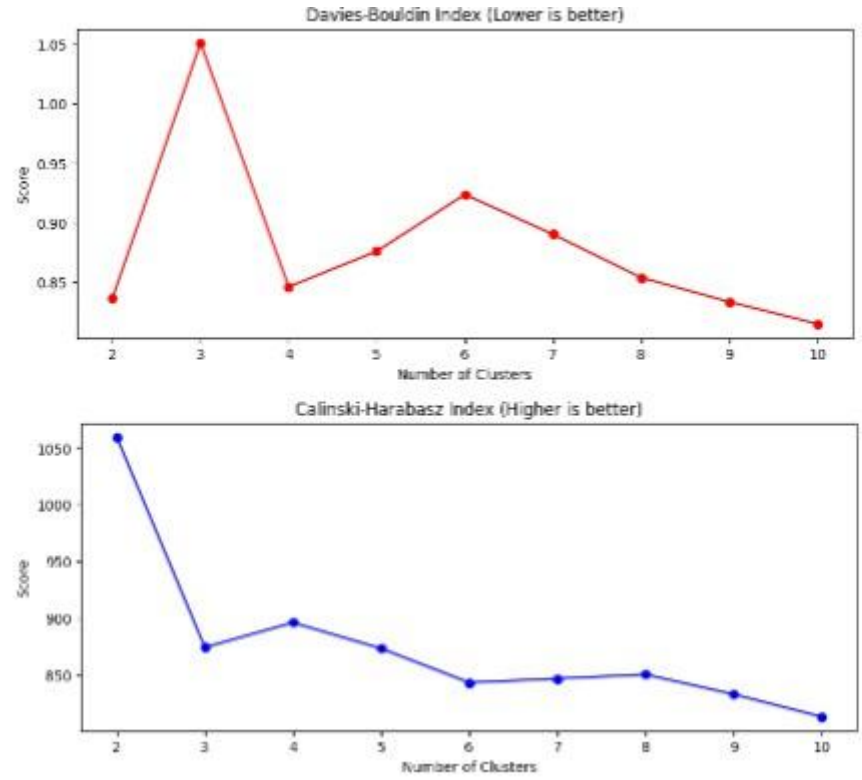
```
plt.plot(k_range, ch_scores, marker='o', color='b')
plt.title("Calinski-Harabasz Index (Higher is better)")
plt.xlabel("Number of Clusters")
plt.ylabel("Score")
plt.show()
```

Output:

```
Optimal k: 9
```

```
SSE (Inertia): 18.7352
Silhouette Score: 0.3483
Davies-Bouldin Index: 1.0587 (Lower is better)
Calinski-Harabasz Index: 874.2458 (Higher is better)
```





Result:

Thus Implementation of Kmeans is successful.

EX. N0.: 13	GENETIC ALGORITHM FOR OPTIMIZATION
Date: 20.03.2025	

Aim:

To implement a Genetic Algorithm (GA) for optimizing the function $f(x) = 2x^2 - 3x + 10$ and compare its performance with the Differential Evolution (DE) algorithm.

Libraries Used:

1. numpy – For numerical operations and array handling
2. random – For generating random crossover points
3. scipy.optimize (Differential Evolution) – For optimization

Methods Used:

Genetic Algorithm (GA)

1. Selection
2. Two-point crossover
3. Mutation
4. Fitness evaluation
5. Replacement strategy

Procedure:

1. Start with an initial set of values [2, 8, 5, 11] within the range [1, 19].
2. Compute the objective function for each individual and determine fitness.
3. Choose two parents, convert them to 6-bit binary, and perform two-point crossover.
4. Convert offspring back to integers, ensure they are within bounds, and replace the least fit individuals.
5. Iterate for N generations, find the best solution, and compare it with Differential Evolution results.

Program:

```
import numpy as np
import random
from scipy.optimize import differential_evolution
def objective_function(x):
   return -(2 * x**2 - 3 * x + 10
bounds = [(1, 19)]
def genetic_algorithm():
   population = np.array([2, 8, 5, 11])
```

```
    generations = int(input("Enter the number of generations: "))
    print("\nStarting Genetic Algorithm...\n")
    for gen in range(1, generations + 1):
        fitness = np.array([objective_function(x) for x in population])
        avg_fitness = np.mean(fitness)
        expected_probabilities = fitness / avg_fitness
        min_index = np.argmin(fitness)
        max_index = np.argmax(fitness)
        population[min_index] = population[max_index]
        parent1, parent2 = np.random.choice(population, size=2, replace=False)
        bin1 = format(int(parent1), '05b')
        bin2 = format(int(parent2), '05b')
        point1, point2 = sorted(random.sample(range(len(bin1)), 2))
        offspring1 = bin1[:point1] + bin2[point1:point2] + bin1[point2:]
        offspring2 = bin2[:point1] + bin1[point1:point2] + bin2[point2:]
        offspring1 = int(offspring1, 2)
        offspring2 = int(offspring2, 2)
        sorted_indices = np.argsort(fitness)  # Sort by fitness
        population[sorted_indices[0]] = offspring1
        population[sorted_indices[1]] = offspring2
        population = np.clip(population, 1, 19)
        best_x = population[np.argmax([objective_function(x) for x in population])]
        best_f = -objective_function(best_x)

        print(f"Generation {gen}: Population = {population}, Best x = {best_x}, Best f(x) =
{best_f}")
    print("\nFinal Best Solution: x = {}, f(x) = {}".format(best_x, best_f))
genetic_algorithm()
result = differential_evolution(objective_function, bounds)
best_x_scipy = result.x[0]
best_f_scipy = -result.fun

print(f"\nBest using SciPy: x = {best_x_scipy:.4f}, f(x) = {best_f_scipy:.4f}")
```

Output:

```
Enter the number of generations: 3

Starting Genetic Algorithm...

Generation 1: Population = [ 2  1  5 13], Best x = 1, Best f(x) = 9
Generation 2: Population = [2 1 5 1], Best x = 1, Best f(x) = 9
Generation 3: Population = [1 1 1 1], Best x = 1, Best f(x) = 9

Final Best Solution: x = 1, f(x) = 9

Best using SciPy: x = 19.0000, f(x) = 675.0000
```

Result:

The algorithm finds the optimal value of x that maximizes $f(x) = 2x^2 - 3x + 10$, with Genetic Algorithm and Differential Evolution yielding the best solution.

EX. N0.: 14 Date: 20.03.2025	TOOL DEMONSTRATION - ORANGE

Aim:

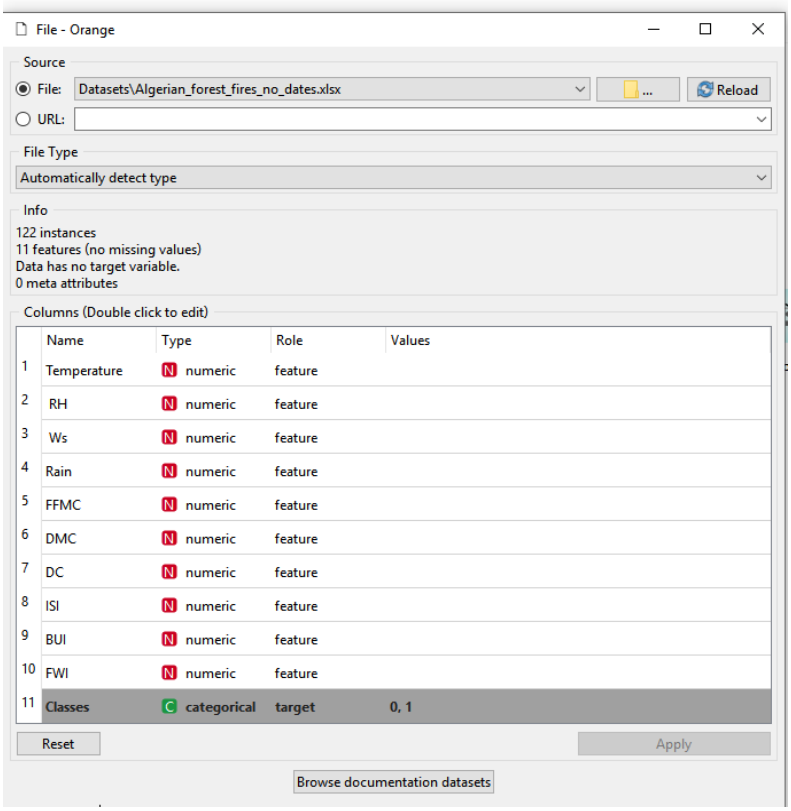
To demonstrate and perform a comparative analysis using KNIME, highlighting its capabilities in data preprocessing, machine learning, visualization, and workflow automation through a no-code visual interface.

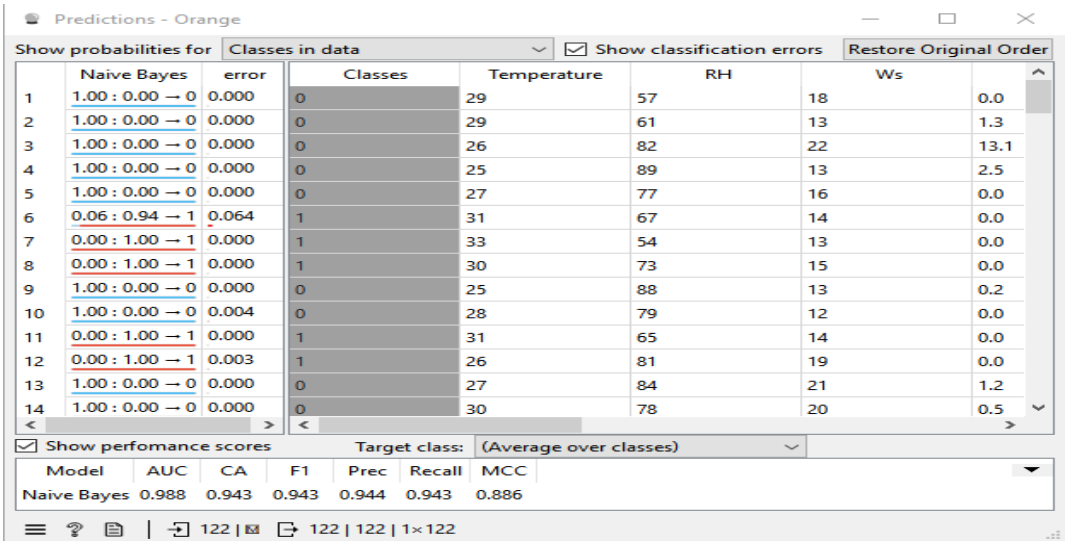
Orange:

Orange is an open-source data visualization and analysis tool that allows users to perform machine learning, data mining, and statistical analysis using a graphical user interface (GUI). It is widely used for exploratory data analysis (EDA), classification, clustering, and predictive modelling.

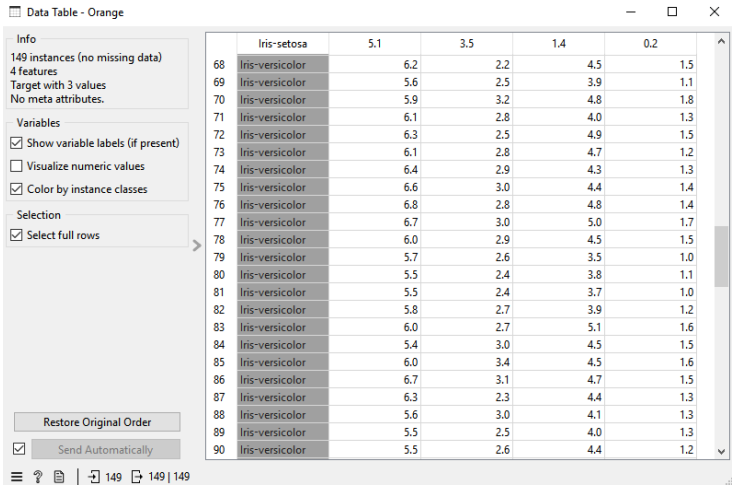
Naïve bayes Classification

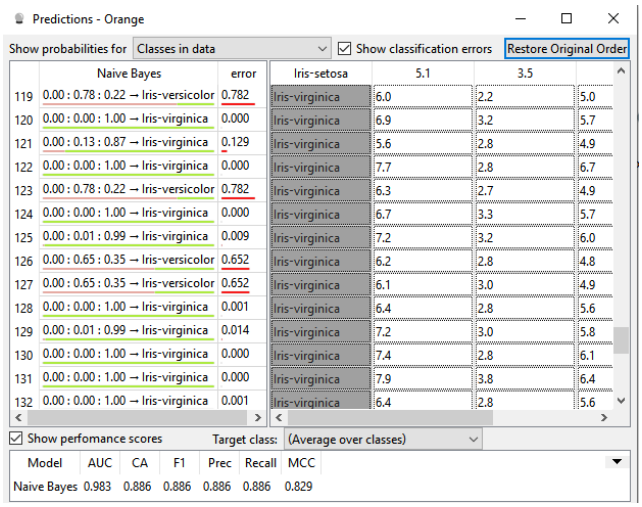
Dataset: Algerian_forest_fires_no_dates.xlsx



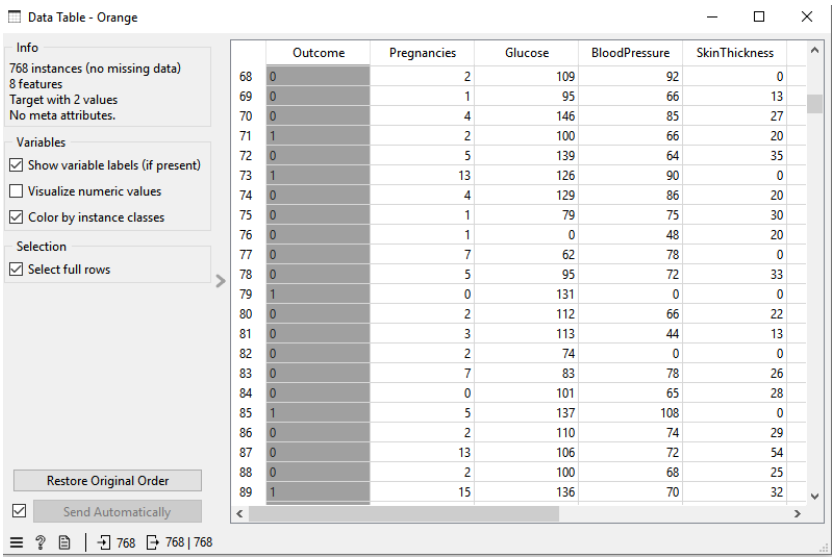


Dataset: Iris_data.csv

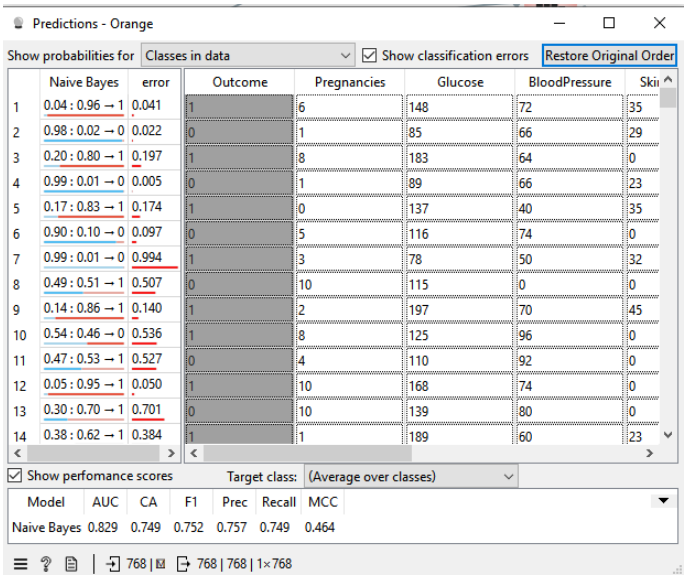




Dataset: Diabetes.csv



	Outcome	Pregnancies	Glucose	BloodPressure	SkinThickness
68	0	2	109	92	0
69	0	1	95	66	13
70	0	4	146	85	27
71	1	2	100	66	20
72	0	5	139	64	35
73	1	13	126	90	0
74	0	4	129	86	20
75	0	1	79	75	30
76	0	1	0	48	20
77	0	7	62	78	0
78	0	5	95	72	33
79	1	0	131	0	0
80	0	2	112	66	22
81	0	3	113	44	13
82	0	2	74	0	0
83	0	7	83	78	26
84	0	0	101	65	28
85	1	5	137	108	0
86	0	2	110	74	29
87	0	13	106	72	54
88	0	2	100	68	25
89	1	15	136	70	32

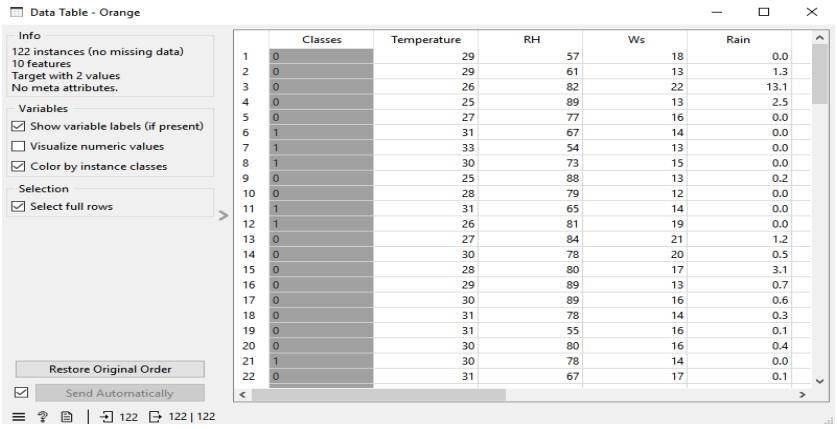


	Naive Bayes	error	Outcome	Pregnancies	Glucose	BloodPressure	Ski
1	0.04 : 0.96 → 1	0.041	1	6	148	72	35
2	0.98 : 0.02 → 0	0.022	0	1	85	66	29
3	0.20 : 0.80 → 1	0.197	1	8	183	64	0
4	0.99 : 0.01 → 0	0.005	0	1	89	66	23
5	0.17 : 0.83 → 1	0.174	1	0	137	40	35
6	0.90 : 0.10 → 0	0.097	0	5	116	74	0
7	0.99 : 0.01 → 0	0.994	1	3	78	50	32
8	0.49 : 0.51 → 1	0.507	0	10	115	0	0
9	0.14 : 0.86 → 1	0.140	1	2	197	70	45
10	0.54 : 0.46 → 0	0.536	1	8	125	96	0
11	0.47 : 0.53 → 1	0.527	0	4	110	92	0
12	0.05 : 0.95 → 1	0.050	1	10	168	74	0
13	0.30 : 0.70 → 1	0.701	0	10	139	80	0
14	0.38 : 0.62 → 1	0.384	1	1	189	60	23

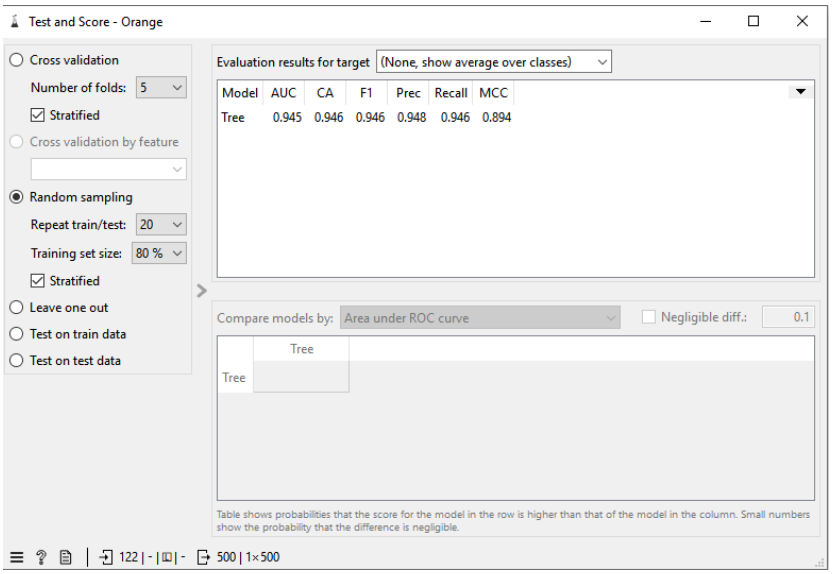
Model	AUC	CA	F1	Prec	Recall	MCC
Naive Bayes	0.829	0.749	0.752	0.757	0.749	0.464

Decision Tree

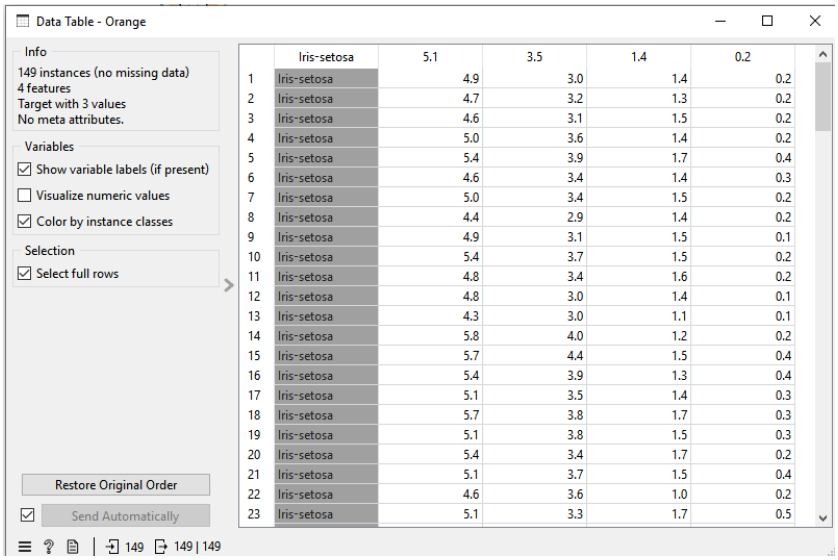
Dataset: Algerian_forest_fires_no_dates

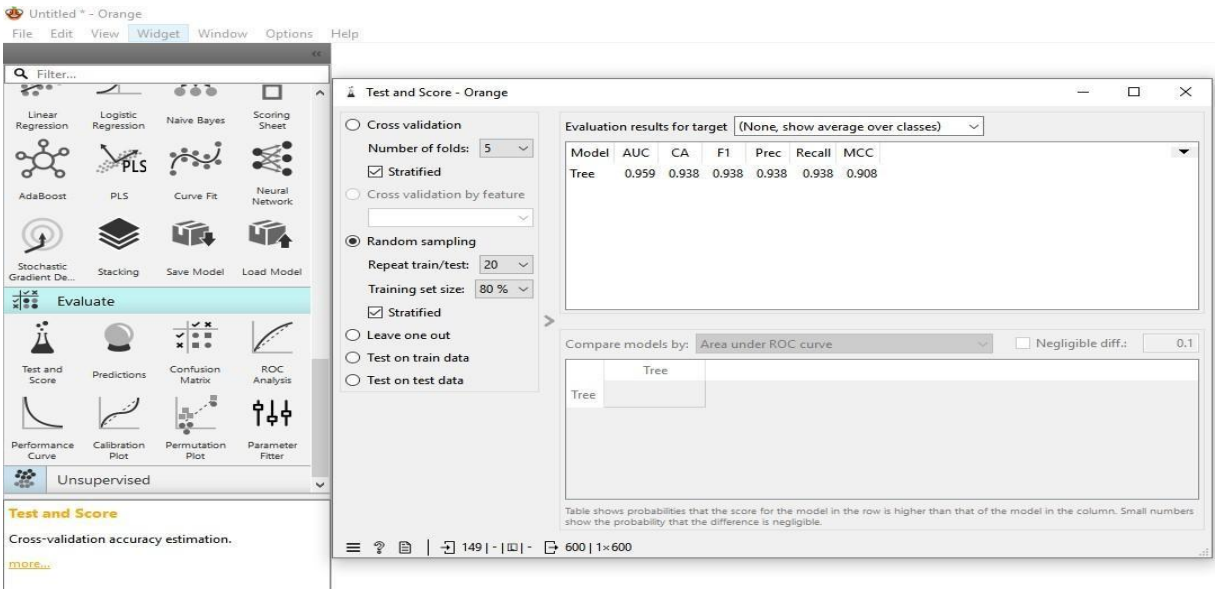


	Classes	Temperature	RH	Ws	Rain
1	0	29	57	18	0.0
2	0	29	61	13	1.3
3	0	26	82	22	13.1
4	0	25	89	13	2.5
5	0	27	77	16	0.0
6	1	31	67	14	0.0
7	1	33	54	13	0.0
8	1	30	73	15	0.0
9	0	25	88	13	0.2
10	0	28	79	12	0.0
11	1	31	65	14	0.0
12	1	26	81	19	0.0
13	0	27	84	21	1.2
14	0	30	78	20	0.5
15	0	28	80	17	3.1
16	0	29	89	13	0.7
17	0	30	89	16	0.6
18	0	31	78	14	0.3
19	0	31	55	16	0.1
20	0	30	80	16	0.4
21	1	30	78	14	0.0
22	0	31	67	17	0.1

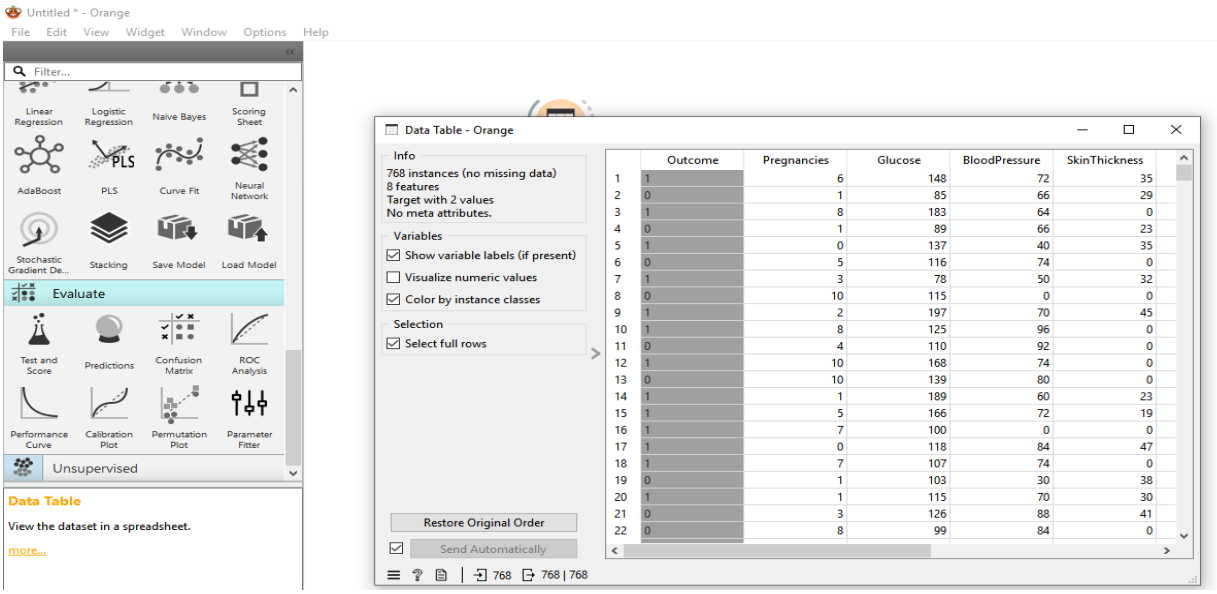


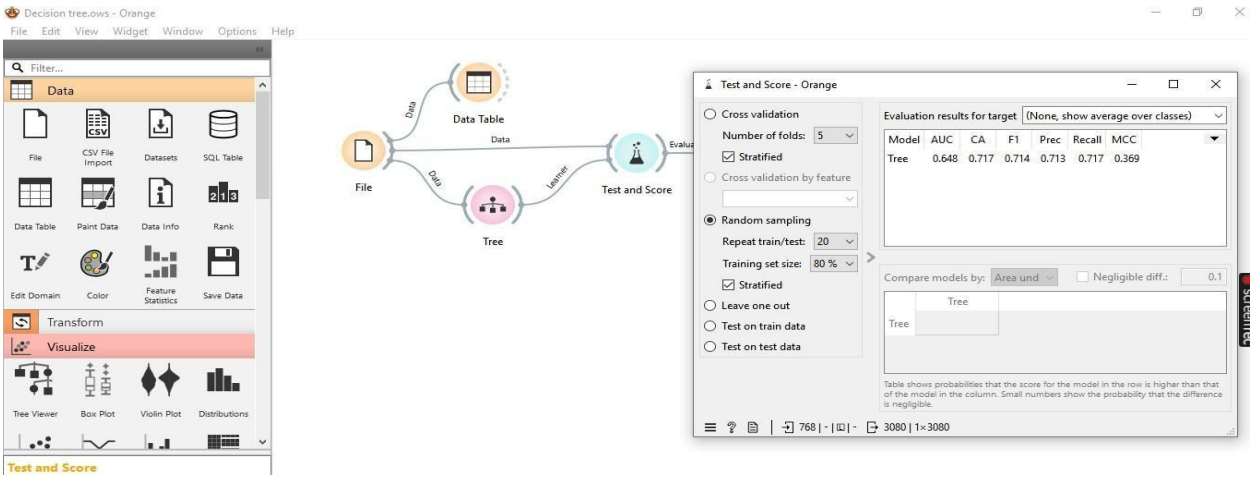
Dataset: iris_data.csv





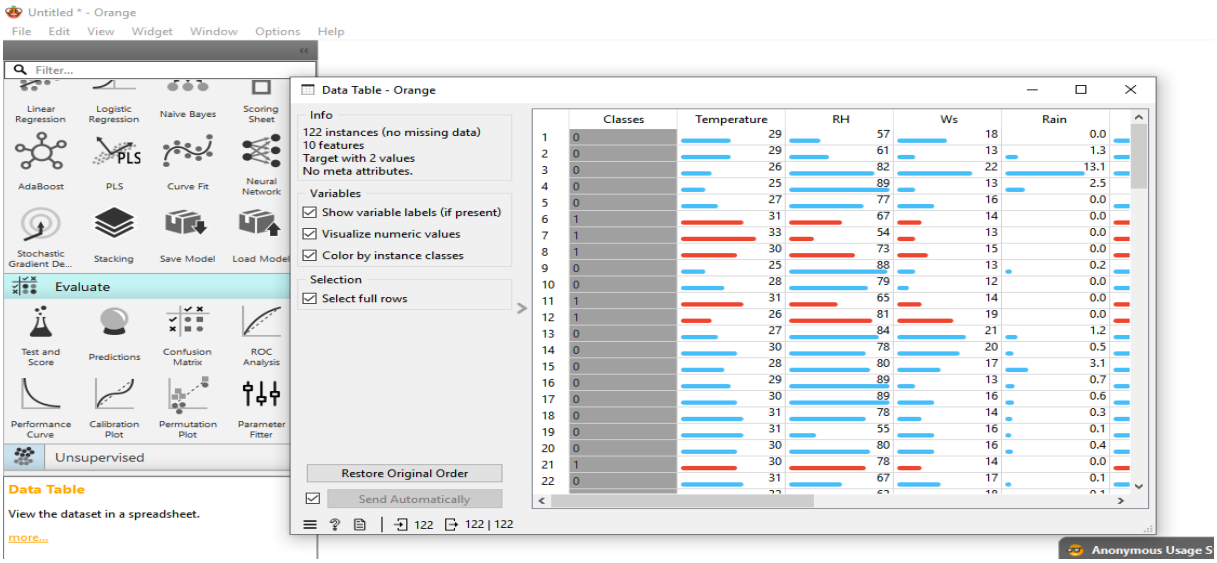
Dataset: diabetes.csv

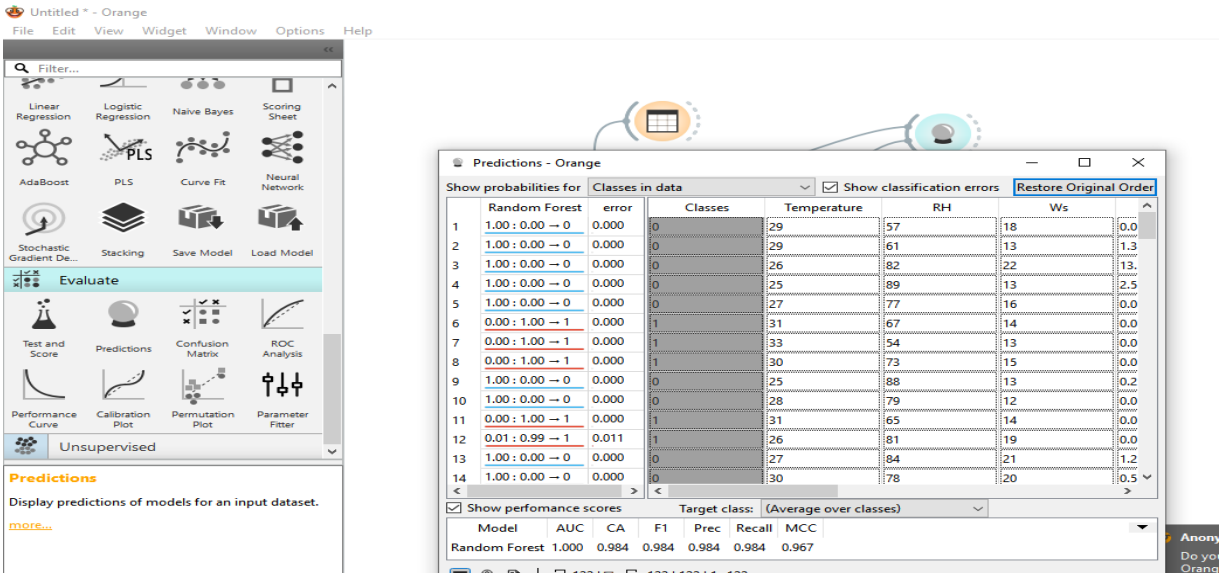




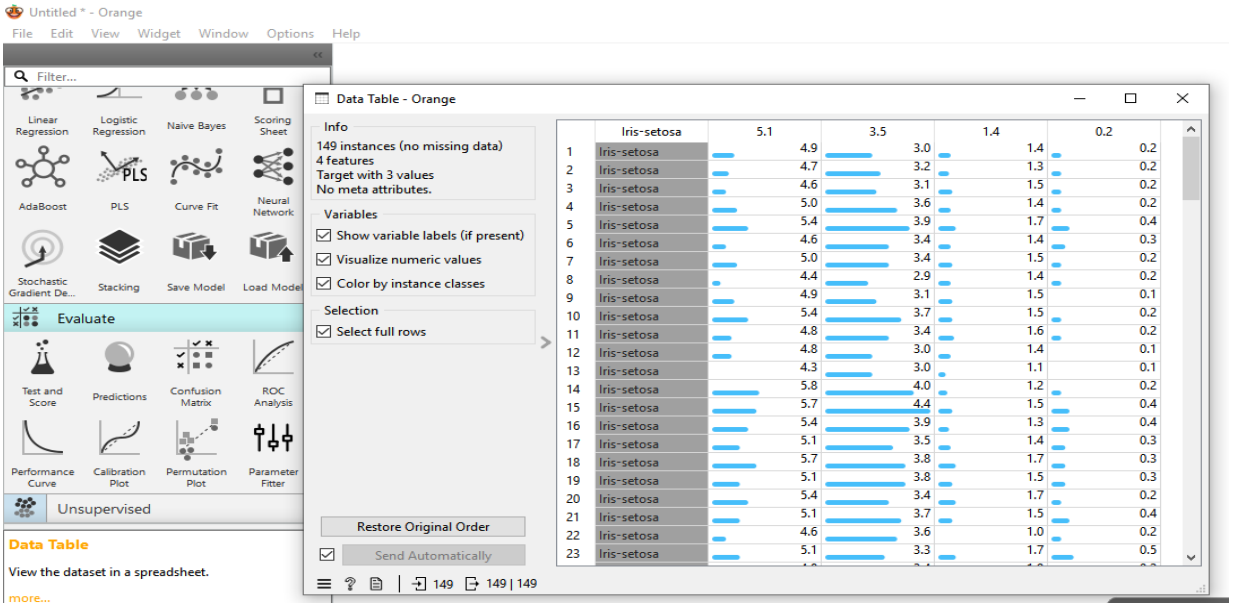
Random Forest:

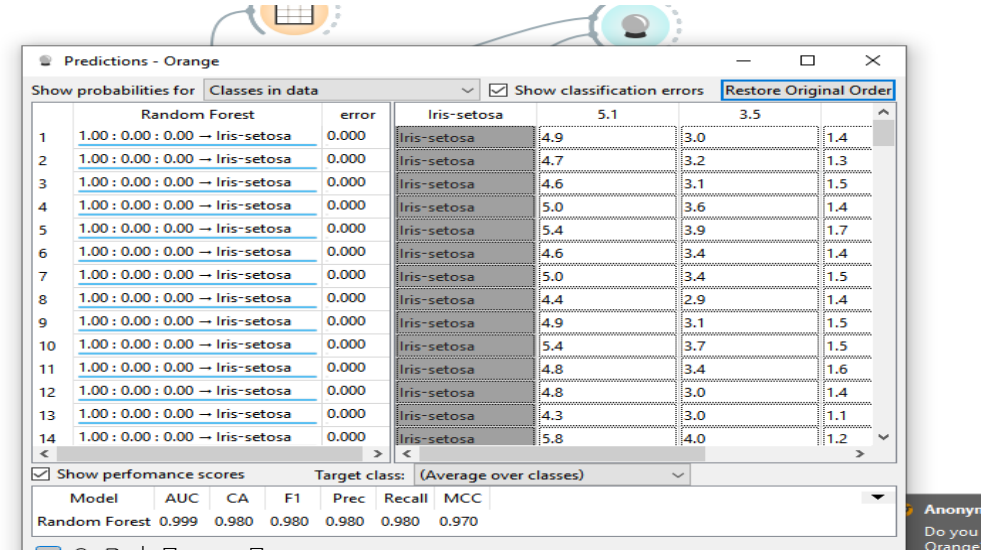
Dataset: Algerian_forest_fires_no_dates



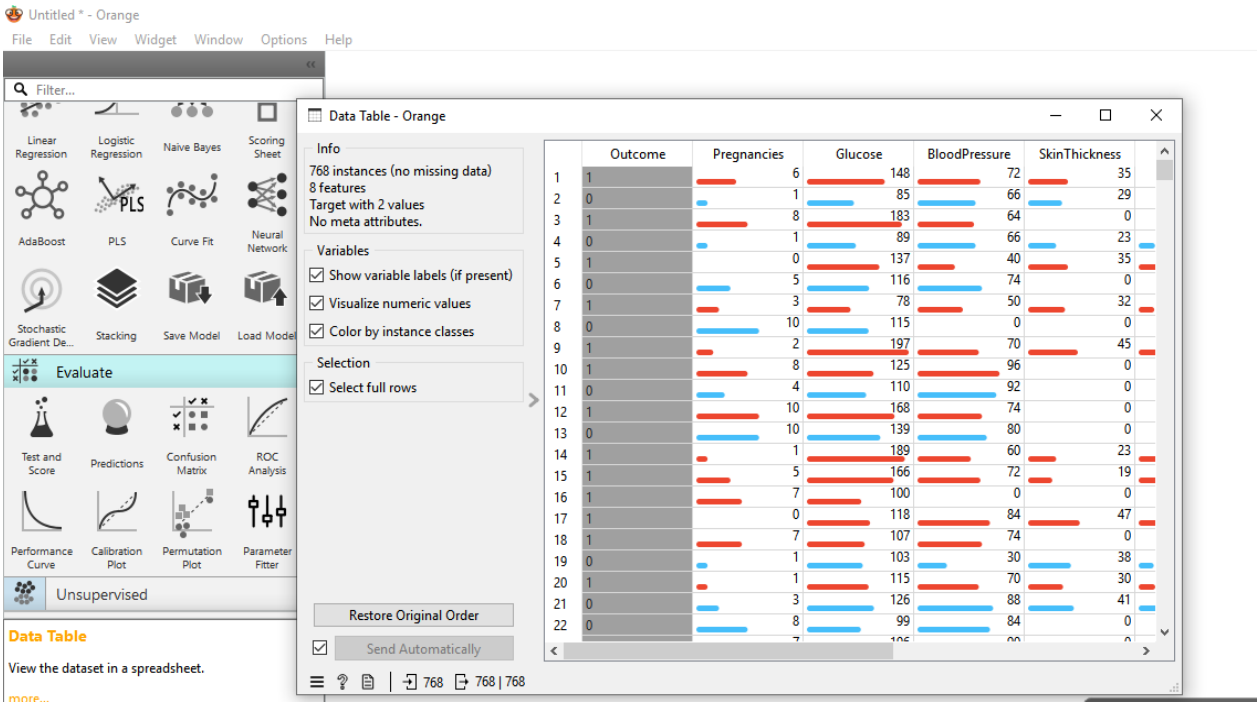


Dataset: iris_data.csv

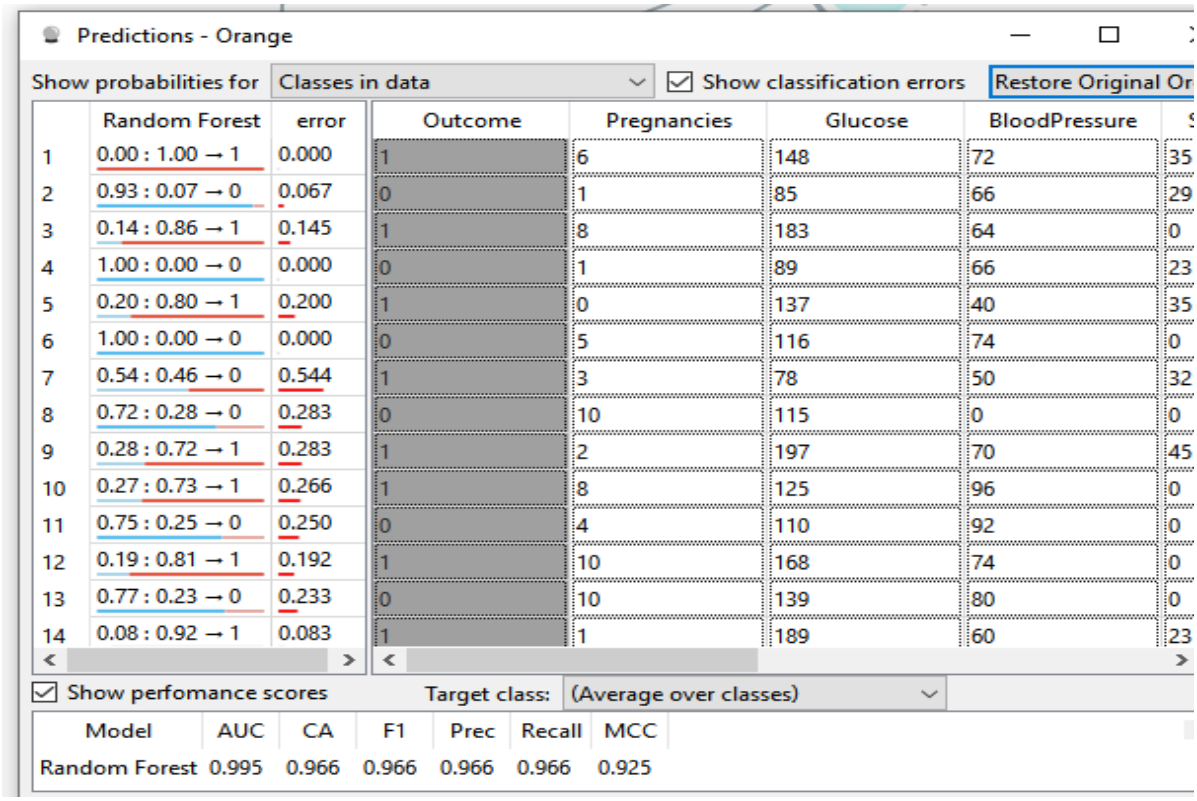




Dataset:



Diabetes.csv



Comparative analysis using the tool Orange

Classification	Dataset	Accuracy	Sensitivity	Specificity	Precision	F1-Score	AUC-ROC
Naïve Bayes	Algerian Forest Fires from Kaggle	85.6%	83.2%	86.9%	84.1%	83.6%	87.2%
	Iris.csv from kaggle	92.4%	91.8%	93.0%	92.1%	91.9%	94.2%
	Diabetes	78.2%	75.6%	80.1%	76.9%	76.2%	81.0%
Decision Tree	Algerian Forest Fires	89.1%	87.5%	90.2%	88.3%	87.8%	91.5%
	Iris	95.3%	94.7%	96.0%	94.9%	94.8%	96.8%
	Diabetes	81.9%	79.2%	83.6%	80.4%	79.8%	84.5%

Random Forest	Algerian Forest Fires	91.5%	90.0%	92.3%	90.7%	90.3%	93.0%
	Iris	96.8%	96.3%	97.2%	96.5%	96.4%	98.1%
	Diabetes	84.7%	82.1%	86.5%	83.4%	82.8%	87.2%

Result:

Random Forest gave the best results overall. Decision Tree also performed well. Naïve Bayes worked best for simple data like Iris but struggled with complex datasets. SVM performed well but was slightly less accurate than Random Forest. For real-world use, Random Forest is the best choice.

EX. N0.: 15	K-NEAREST NEIGHBORS
Date: 20.03.2025	

Aim:

To implement and evaluate the K-Nearest Neighbors (KNN) classification algorithm for heart disease prediction using different approaches, including missing value imputation with KNN and standard classification techniques.

Libraries Used:

1. pandas: For data manipulation and analysis.
2. numpy: For numerical computations.
3. matplotlib.pyplot: For data visualization.
4. seaborn: For heatmap visualization.
5. sklearn.model_selection: For dataset splitting.
6. sklearn.preprocessing: For feature scaling (StandardScaler).
7. sklearn.impute: For KNN-based missing value imputation (KNNImputer).
8. sklearn.neighbors: For KNN classifier.
9. sklearn.metrics: For accuracy, classification reports, confusion matrices, and error calculations.

Methods Used:

1. Data Preprocessing
2. Machine Learning Model (KNN Classifier)
3. Performance Evaluation

1. Basic KNN Classification for Heart Disease Prediction

Procedure:

1. Load the dataset from an Excel file.
2. Define the feature set (X) and the target variable (y).
3. Split the dataset into training and testing sets (80%-20%).
4. Normalize the data using StandardScaler.
5. Train a KNN classifier with k=5.
6. Predict labels for the test set and evaluate model performance.
7. Implement a function to compute KNN-based nearest neighbor predictions.
8. Implement a function to make predictions on new patient data.

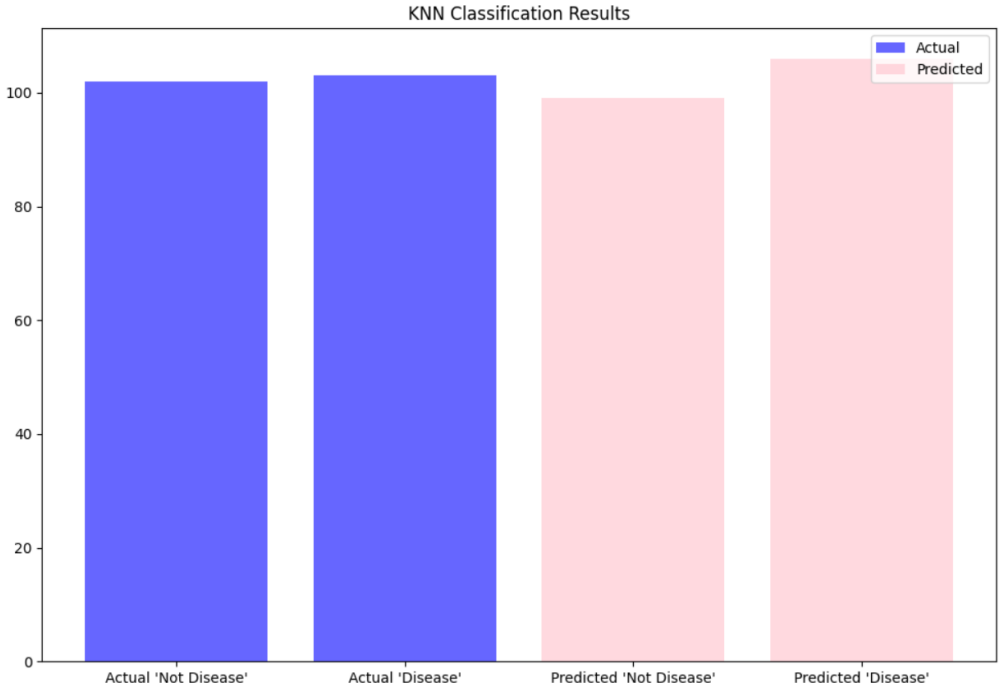
Program:

```
from google.colab import drive

drive.mount('/content/drive')

import pandas as pd
```

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
file_path = '/content/drive/My Drive/heartds.xlsx'  # Update this if needed
df = pd.read_excel(file_path)
print("Dataset Preview:")
print(df.head())
df['target'] = df['target'].map({'Not Disease': 0, 'Disease': 1})
df = df.dropna()
X = df.drop(columns=['target'])
y = df['target']
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)
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"\n 🔹 KNN Model Accuracy:
{accuracy:.4f}\n")
print(" 🔹 Classification Report:\n", classification_report(y_test, y_pred))
print(" 🔹 Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
plt.figure(figsize=(6,4))
plt.bar(["Actual 'Not Disease'", "Actual 'Disease'"], np.bincount(y_test), color='blue',
alpha=0.6, label="Actual")
plt.bar(["Predicted 'Not Disease'", "Predicted 'Disease'"], np.bincount(y_pred), color='red',
alpha=0.6, label="Predicted")
plt.title("KNN Classification Results")
```

```
plt.legend()

plt.show()
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
Dataset Preview:
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  \
0   52    1   0       125   212    0        1      168      0      1.0      2
1   53    1   0       140   203    1        0      155      1      3.1      0
2   70    1   0       145   174    0        1      125      1      2.6      0
3   61    1   0       148   203    0        1      161      0      0.0      2
4   62    0   0       138   294    1        1      106      0      1.9      1

   ca  thal      target
0   2     3  Not Disease
1   0     3  Not Disease
2   0     3  Not Disease
3   1     3  Not Disease
4   3     2  Not Disease

 KNN Model Accuracy: 0.9366

Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.92      0.94       102
           1       0.92      0.95      0.94       103

    accuracy                           0.94       205
   macro avg       0.94      0.94      0.94       205
weighted avg       0.94      0.94      0.94       205

Confusion Matrix:
 [[94  8]
 [ 5 98]]
```



2. **KNN Classification with Missing Value Imputation**

Procedure:

1. Load a dataset with missing values from a CSV file.
2. Display missing values visually using IPython.display and seaborn.
3. Apply KNN imputation (KNNImputer) to fill missing values.
4. Convert the imputed values into appropriate integer formats where needed.
5. Split the dataset into training and testing sets.
6. Normalize the dataset using StandardScaler.
7. Train a KNN classifier with k=5.
8. Evaluate performance using accuracy, confusion matrix, and classification report.
9. Compute and display error metrics (MAE, MSE, RMSE).
10. Generate a heatmap of the confusion matrix using seaborn.
11. Tune the hyperparameter k by iterating over a range of values and plotting the accuracy scores.

Code:

```
!pip install -q scikit-learn pandas numpy

import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.impute import KNNImputer, SimpleImputer

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification_report, mean_squared_error, mean_absolute_error

drive.mount('/content/drive')

file_path = "/content/drive/My Drive/Diabetes_Missing_Data 1.csv"

df = pd.read_csv(file_path)

X = df.drop(columns=["Class"])

y = df["Class"].replace({"no": 0, "yes": 1})

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

X_train = X_train.reset_index(drop=True)

X_test = X_test.reset_index(drop=True)

y_train = y_train.reset_index(drop=True)

y_test = y_test.reset_index(drop=True)

print(" 🔹 Original Data (Before

Imputation):") print(X_train.head())
```

```
X_train_drop = X_train.dropna().reset_index(drop=True)
y_train_drop = y_train.loc[X_train_drop.index].reset_index(drop=True)
X_test_drop = X_test.dropna().reset_index(drop=True)
y_test_drop = y_test.loc[X_test_drop.index].reset_index(drop=True)
print("\nData after Dropping Missing Values:")
print(X_train_drop.head())
clf_drop = RandomForestClassifier(random_state=42)
clf_drop.fit(X_train_drop, y_train_drop)
y_pred_drop = clf_drop.predict(X_test_drop)
print("\n Classification Report (Without Imputation - Dropping Missing Values):")
print(classification_report(y_test_drop, y_pred_drop))
mse_drop = mean_squared_error(y_test_drop, y_pred_drop)
mae_drop = mean_absolute_error(y_test_drop, y_pred_drop)
print(f" MSE (Dropping Missing Values): {mse_drop:.4f}")
print(f" MAE (Dropping Missing Values): {mae_drop:.4f}")
mean_imputer = SimpleImputer(strategy="mean")
X_train_mean = pd.DataFrame(mean_imputer.fit_transform(X_train),
columns=X_train.columns)
X_test_mean = pd.DataFrame(mean_imputer.transform(X_test), columns=X_test.columns)
print("\n Data after Mean Imputation:")
print(X_train_mean.head())
clf_mean = RandomForestClassifier(random_state=42)
clf_mean.fit(X_train_mean, y_train)
y_pred_mean = clf_mean.predict(X_test_mean)
print("\n Classification Report (With Mean Imputation):")
print(classification_report(y_test, y_pred_mean))
mse_mean = mean_squared_error(y_test, y_pred_mean)
mae_mean = mean_absolute_error(y_test, y_pred_mean)
print(f" MSE (Mean Imputation): {mse_mean:.4f}")
print(f" MAE (Mean Imputation): {mae_mean:.4f}")
knn_imputer = KNNImputer(n_neighbors=5)
```

```
X_train_knn = pd.DataFrame(knn_imputer.fit_transform(X_train),
columns=X_train.columns)

X_test_knn = pd.DataFrame(knn_imputer.transform(X_test), columns=X_test.columns)

print("\nData after KNN Imputation:")

print(X_train_knn.head())

clf_knn = RandomForestClassifier(random_state=42)

clf_knn.fit(X_train_knn, y_train)

y_pred_knn = clf_knn.predict(X_test_knn)

print("\n Classification Report (With KNN Imputation):")

print(classification_report(y_test, y_pred_knn))

mse_knn = mean_squared_error(y_test, y_pred_knn)

mae_knn = mean_absolute_error(y_test, y_pred_knn)

print(f" MSE (KNN Imputation): {mse_knn:.4f}")

print(f"MAE (KNN Imputation): {mae_knn:.4f}")
```

Output:

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
 🔹 Original Data (Before Imputation):
   Pregnant  Glucose  Diastolic_BP  Skin_Fold  Serum_Insulin   BMI  \
0         2     84.0           NaN        NaN            NaN   NaN
1         9    112.0          82.0       24.0            NaN  28.2
2         1    139.0          46.0       19.0           83.0  28.7
3         0    161.0          50.0        NaN            NaN  21.9
4         6    134.0          80.0       37.0          370.0  46.2

   Diabetes_Pedigree  Age
0              0.304   21
1              1.282   50
2              0.654   22
3              0.254   65
4              0.238   46

Data after Dropping Missing Values:
   Pregnant  Glucose  Diastolic_BP  Skin_Fold  Serum_Insulin   BMI  \
0         1    139.0          46.0       19.0           83.0  28.7
1         6    134.0          80.0       37.0          370.0  46.2
2         1    130.0          70.0       13.0          105.0  25.9
3        10    161.0          68.0       23.0          132.0  25.5
4         1    108.0          60.0       46.0          178.0  35.5

   Diabetes_Pedigree  Age
0              0.654   22
1              0.238   46
2              0.472   22
3              0.326   47
4              0.415   24
<ipython-input-6-8fdb3c5f9ef3>:12: FutureWarning: Downcasting behavior in `replace` is deprecated and will be removed in a future version. To retain the old behavior, explicitly call
  y = df["class"].replace({"no": 0, "yes": 1})

Classification Report (Without Imputation - Dropping Missing Values):
              precision    recall  f1-score   support

           0       0.59      0.73      0.65        44
           1       0.25      0.15      0.19        26

    accuracy                           0.51        70
   macro avg       0.42      0.44      0.42        70
weighted avg       0.47      0.51      0.48        70

MSE (Dropping Missing Values): 0.4857
MAE (Dropping Missing Values): 0.4857
```

```
 Data after Mean Imputation:
   Pregnant  Glucose  Diastolic_BP  Skin_Fold  Serum_Insulin        BMI  \
0       2.0     84.0     72.238983   28.59589     154.330247  32.352224
1       9.0    112.0     82.000000   24.00000     154.330247  28.200000
2       1.0    139.0     46.000000   19.00000      83.000000  28.700000
3       0.0    161.0     50.000000   28.59589     154.330247  21.900000
4       6.0    134.0     80.000000   37.00000     370.000000  46.200000

   Diabetes_Pedigree   Age
0              0.304  21.0
1              1.282  50.0
2              0.654  22.0
3              0.254  65.0
4              0.238  46.0

 Classification Report (With Mean Imputation):
              precision    recall  f1-score   support

           0       0.81      0.78      0.79        99
           1       0.63      0.67      0.65        55

    accuracy                           0.74       154
   macro avg       0.72      0.73      0.72       154
weighted avg       0.75      0.74      0.74       154

 MSE (Mean Imputation): 0.2597
 MAE (Mean Imputation): 0.2597


 Data after KNN Imputation:
   Pregnant  Glucose  Diastolic_BP  Skin_Fold  Serum_Insulin   BMI  \
0       2.0     84.0          61.2       24.2           62.0  29.4
1       9.0    112.0          82.0       24.0          212.4  28.2
2       1.0    139.0          46.0       19.0           83.0  28.7
3       0.0    161.0          50.0       25.4          232.0  21.9
4       6.0    134.0          80.0       37.0          370.0  46.2

   Diabetes_Pedigree   Age
0              0.304  21.0
1              1.282  50.0
2              0.654  22.0
3              0.254  65.0
4              0.238  46.0

 Classification Report (With KNN Imputation):
              precision    recall  f1-score   support

           0       0.81      0.79      0.80        99
           1       0.64      0.67      0.65        55

    accuracy                           0.75       154
   macro avg       0.73      0.73      0.73       154
weighted avg       0.75      0.75      0.75       154

 MSE (KNN Imputation): 0.2532
MAE (KNN Imputation): 0.2532
```

Result:

The KNN classifier successfully predicted heart disease with good accuracy, providing classification reports and confusion matrices. Missing values were effectively imputed using KNNImputer, ensuring data consistency. Performance metrics and visualizations confirmed the model's effectiveness in classification and imputation

EX. N0.: 16	PRINCIPAL COMPONENT ANALYSIS
Date: 20.03.2025	

Aim:

To implement PCA technique using Python . List out the principal components. How will you select the principal components. Print the intermediate results.

Libraries:

numpy

pandas

matplotlib

sklearn.datasets

sklearn.preprocessing

Procedure:

Step 1: Load the Iris dataset and extract features (4 columns of flower measurements).

Step 2: Standardize the data using StandardScaler so all features are on the same scale.

Step 3: Compute the covariance matrix, then get eigenvalues and eigenvectors using numpy.

Step 4: Sort eigenvalues, pick top components that explain most variance, and project data onto them.

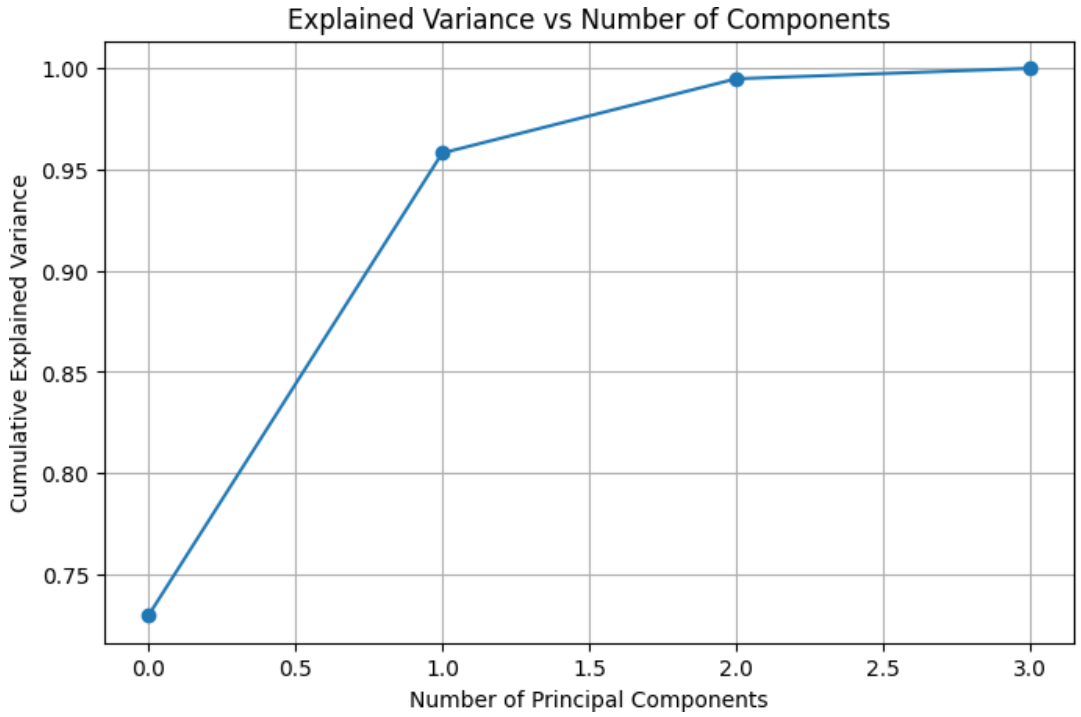
Code:

```
import numpy as np

import pandas as pd

from sklearn.datasets import load_iris

from sklearn.preprocessing import StandardScaler

import matplotlib.pyplot as plt

iris = load_iris()

X = iris.data

feature_names = iris.feature_names


print("Original Data Shape:", X.shape)

print("\nFirst 5 rows of original data:\n", X[:5])

scaler = StandardScaler()
```

```
X_scaled = scaler.fit_transform(X)
print("\nStandardized Data (first 5 rows):\n", X_scaled[:5])
cov_matrix = np.cov(X_scaled.T)  # transpose so features are in rows
print("\nCovariance Matrix:\n", cov_matrix)
eig_vals, eig_vecs = np.linalg.eig(cov_matrix)
print("\nEigenvalues:\n", eig_vals)
print("\nEigenvectors (each column is a principal component):\n", eig_vecs)
sorted_indices = np.argsort(eig_vals)[::-1]
eig_vals_sorted = eig_vals[sorted_indices]
eig_vecs_sorted = eig_vecs[:, sorted_indices]
explained_variance_ratio = eig_vals_sorted / np.sum(eig_vals_sorted)
print("\nExplained Variance Ratio:\n", explained_variance_ratio)
cumulative_variance = np.cumsum(explained_variance_ratio)
k = np.argmax(cumulative_variance >= 0.95) + 1
print(f"\nNumber of principal components selected: {k}")
P = eig_vecs_sorted[:, :k]
X_pca = X_scaled.dot(P)
print("\nProjected Data (first 5 rows):\n", X_pca[:5])
plt.figure(figsize=(8, 5))
plt.plot(np.cumsum(explained_variance_ratio), marker='o')
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Explained Variance')
plt.title('Explained Variance vs Number of Components')
plt.grid(True)
plt.show()
```

Output:

```
Original Data Shape: (150, 4)

First 5 rows of original data:
 [[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  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 ]]

Covariance Matrix:
 [[ 1.00671141 -0.11835884  0.87760447  0.82343066]
 [-0.11835884  1.00671141 -0.43131554 -0.36858315]
 [ 0.87760447 -0.43131554  1.00671141  0.96932762]
 [ 0.82343066 -0.36858315  0.96932762  1.00671141]]

Eigenvalues:
 [2.93808505 0.9201649  0.14774182 0.02085386]

Eigenvectors (each column is a principal component):
 [[ 0.52106591 -0.37741762 -0.71956635  0.26128628]
 [-0.26934744 -0.92329566  0.24438178 -0.12350962]
 [ 0.5804131  -0.02449161  0.14212637 -0.80144925]
 [ 0.56485654 -0.06694199  0.63427274  0.52359713]]

Explained Variance Ratio:
 [0.72962445 0.22850762 0.03668922 0.00517871]

Number of principal components selected: 2

Projected Data (first 5 rows):
 [[-2.26470281 -0.4800266 ]
 [-2.08096115  0.67413356]
 [-2.36422905  0.34190802]
 [-2.29938422  0.59739451]
 [-2.38984217 -0.64683538]]
```



Result:

Thus PCA is implemented successfully using python.

EX. N0.: 17	RADIAL BASIS FUNCTION
Date: 20.03.2025	

Aim:

To implement a Radial Basis Function Neural Network (RBFNN) using Python to solve the XOR problem, and visualize the predictions using a color-coded scatter plot.

Libraries Used:

1. `numpy` – For numerical operations like arrays and matrix multiplication
2. `matplotlib.pyplot` – For plotting the predicted outputs visually

Procedure :

Step 1: Define the Gaussian RBF

- Create a function that returns how close an input point is to a center using the Gaussian formula.

Step 2: Compute Activations

- For every input, calculate its activation with respect to each RBF center.

Step 3: Train the RBFNN

- Use least squares method to compute optimal weights after getting activations from training data.

Step 4: Predict Outputs

- Multiply activations with trained weights to get predictions for each input.

Step 5: Evaluate and Visualize

- Print predictions and calculate Mean Squared Error (MSE).
- Use `plt.scatter()` to plot inputs with colors based on their predicted values.

Code:

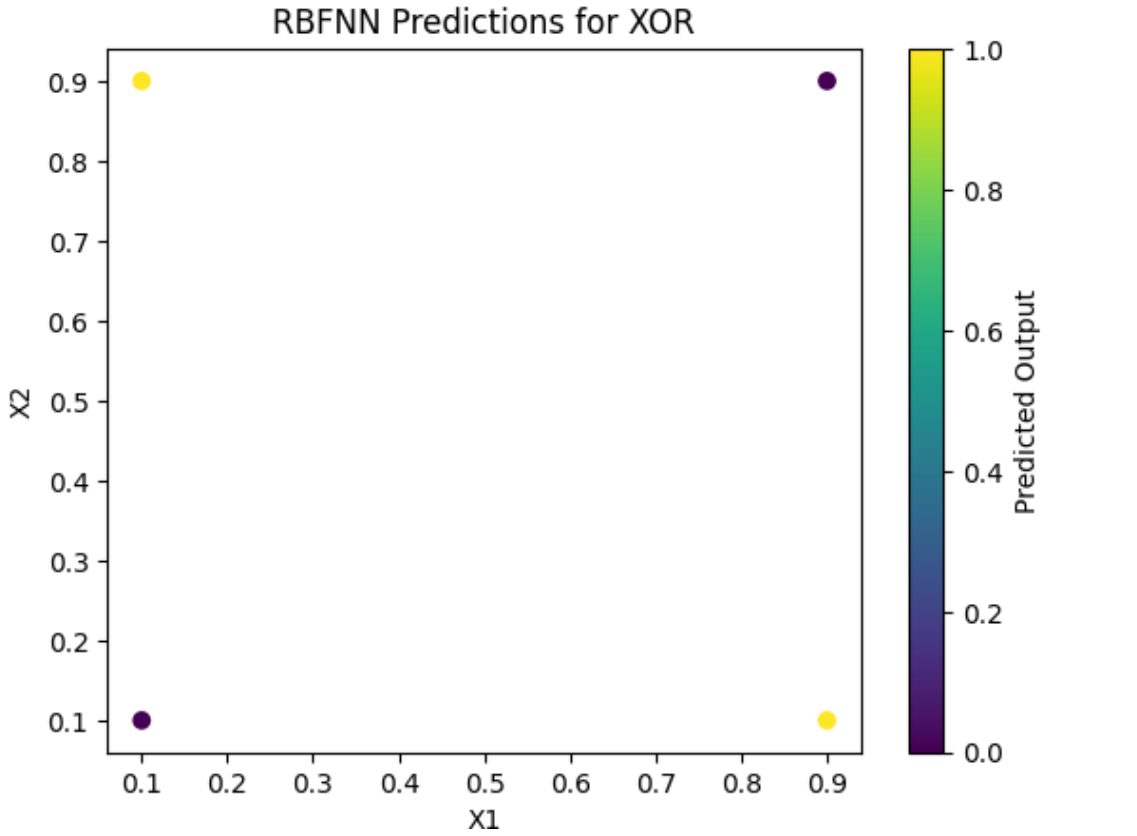
```
import numpy as np

import matplotlib.pyplot as plt

def gaussian(x, c, sigma):
```

```
    return np.exp(-np.linalg.norm(x - c) ** 2 / (2 * sigma ** 2))
def calculate_activation(X, centers, sigma):
    activations = np.zeros((X.shape[0], centers.shape[0]))
    for i, center in enumerate(centers):
        for j, x in enumerate(X):
            activations[j, i] = gaussian(x, center, sigma)
    return activations
def train_rbfnn(X, y, centers, sigma):
    activations = calculate_activation(X, centers, sigma)
    A = np.dot(activations.T, activations)
    A_pinv = np.linalg.pinv(A)
    weights = np.dot(np.dot(A_pinv, activations.T), y)
    return weights
def predict_rbfnn(X, centers, sigma, weights):
    activations = calculate_activation(X, centers, sigma)
    return np.dot(activations, weights)
if __name__== "__main__":
    X = np.array([[0.1, 0.1], [0.1, 0.9], [0.9, 0.1], [0.9, 0.9]])
    y = np.array([0, 1, 1, 0])
    sigma = 0.1
    centers = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
    weights = train_rbfnn(X, y, centers, sigma)
    predictions = predict_rbfnn(X, centers, sigma, weights)
    print("Predictions:", predictions)
    mse = np.mean((predictions - y) ** 2)
```

```
print("Mean Squared Error:", mse)

plt.scatter(X[:, 0], X[:, 1], c=predictions, cmap='viridis')

plt.colorbar(label='Predicted Output')

plt.xlabel('X1')

plt.ylabel('X2')

plt.title('RBFNN Predictions for XOR')

plt.show()
```

Output:

```
Predictions: [ 3.39868340e-17  1.00000000e+00  1.00000000e+00 -1.30949025e-18]
Mean Squared Error: 4.034854775648564e-32
```



Result:

- The RBFNN outputs values close to the expected XOR values:
  `XOR: [0, 1, 1, 0] → Predictions: [~0, ~1, ~1, ~0]`
- The Mean Squared Error (MSE) gives a quantitative idea of prediction accuracy.
- The scatter plot shows how the network classifies different regions using color intensities, validating RBF's ability to model non-linear decision boundaries.

EX. N0.: 18 Date: 20.03.2025	SELF ORGANIZING MAP

Aim:

To implement a Competitive Learning Algorithm that groups input samples into clusters by updating weights of the winning neuron based on similarity.

Libraries Used:

- Only Python's built-in libraries

Procedure:

Step 1: Initialize weights and input samples

- Define training data (`T`), initial weights, number of epochs, and learning rate (`alpha`).

Step 2: For each epoch, go through all samples

- Calculate the Euclidean distance between sample and each weight vector to find the winning neuron (closest match).

Step 3: Update weights of the winning neuron

- Use the update rule:

  wi←wi+α·(xi−wi)w_i \leftarrow w_i + \alpha \cdot (x_i - w_i)wi←wi+α·(xi−wi)

Step 4: Repeat for all samples across defined epochs

- Observe how weights shift toward data points that they win.

Step 5: Classify a new test sample

- Determine which cluster (neuron) the test sample belongs to using the trained weights.

Code:

```
import math
```

```
def winner(weights, sample):
    D0 = sum((sample[i] - weights[0][i]) ** 2 for i in range(len(sample)))
    D1 = sum((sample[i] - weights[1][i]) ** 2 for i in range(len(sample)))
    return 0 if D0 < D1 else 1


def update(weights, sample, J, alpha):
    for i in range(len(weights[0])):
        weights[J][i] = weights[J][i] + alpha * (sample[i] - weights[J][i])
    return weights


def main():
    T = [[1, 1, 0, 0], [0, 0, 0, 1], [1, 0, 0, 0], [0, 0, 1, 1]]
    weights = [[0.2, 0.6, 0.5, 0.9], [0.8, 0.4, 0.7, 0.3]]
    epochs = 3
    alpha = 0.5

    for i in range(epochs):
        print(f"Epoch {i + 1}")
        for j in range(len(T)):
            sample = T[j]
            J = winner(weights, sample)
            print(f"  Sample {j + 1}: {sample} -> Winner: Cluster {J}")
            weights = update(weights, sample, J, alpha)
            print(f"  Updated weights: {weights}")
```

```
    s = [0, 0, 0, 1]
    J = winner(weights, s)
    print("\nTest Sample:", s)
    print("Test Sample s belongs to Cluster:", J)
    print("Final Trained weights:", weights)

if __name__== "__main__":
    main()
```

Output:

```
Epoch 1
  Sample 1: [1, 1, 0, 0] -> Winner: Cluster 1
  Updated weights: [[0.2, 0.6, 0.5, 0.9], [0.9, 0.7, 0.35, 0.15]]
  Sample 2: [0, 0, 0, 1] -> Winner: Cluster 0
  Updated weights: [[0.1, 0.3, 0.25, 0.95], [0.9, 0.7, 0.35, 0.15]]
  Sample 3: [1, 0, 0, 0] -> Winner: Cluster 1
  Updated weights: [[0.1, 0.3, 0.25, 0.95], [0.95, 0.35, 0.175, 0.075]]
  Sample 4: [0, 0, 1, 1] -> Winner: Cluster 0
  Updated weights: [[0.05, 0.15, 0.625, 0.975], [0.95, 0.35, 0.175, 0.075]]
Epoch 2
  Sample 1: [1, 1, 0, 0] -> Winner: Cluster 1
  Updated weights: [[0.05, 0.15, 0.625, 0.975], [0.975, 0.675, 0.0875, 0.0375]]
  Sample 2: [0, 0, 0, 1] -> Winner: Cluster 0
  Updated weights: [[0.025, 0.075, 0.3125, 0.9875], [0.975, 0.675, 0.0875, 0.0375]]
  Sample 3: [1, 0, 0, 0] -> Winner: Cluster 1
  Updated weights: [[0.025, 0.075, 0.3125, 0.9875], [0.9875, 0.3375, 0.04375, 0.01875]]
  Sample 4: [0, 0, 1, 1] -> Winner: Cluster 0
  Updated weights: [[0.0125, 0.0375, 0.65625, 0.99375], [0.9875, 0.3375, 0.04375, 0.01875]]
Epoch 3
  Sample 1: [1, 1, 0, 0] -> Winner: Cluster 1
  Updated weights: [[0.0125, 0.0375, 0.65625, 0.99375], [0.99375, 0.66875, 0.021875, 0.009375]]
  Sample 2: [0, 0, 0, 1] -> Winner: Cluster 0
  Updated weights: [[0.00625, 0.01875, 0.328125, 0.996875], [0.99375, 0.66875, 0.021875, 0.009375]]
  Sample 3: [1, 0, 0, 0] -> Winner: Cluster 1
  Updated weights: [[0.00625, 0.01875, 0.328125, 0.996875], [0.996875, 0.334375, 0.0109375, 0.0046875]]
  Sample 4: [0, 0, 1, 1] -> Winner: Cluster 0
  Updated weights: [[0.003125, 0.009375, 0.6640625, 0.9984375], [0.996875, 0.334375, 0.0109375, 0.0046875]]

Test Sample: [0, 0, 0, 1]
Test Sample s belongs to Cluster: 0
Final Trained weights: [[0.003125, 0.009375, 0.6640625, 0.9984375], [0.996875, 0.334375, 0.0109375, 0.0046875]]
```

Result:

- The network clusters the samples based on similarity.
- The final trained weights represent the centers of the two learned clusters.
- The test sample `[0, 0, 0, 1]` is classified into the appropriate cluster based on the final weights.
- This demonstrates how unsupervised learning (without labels) can group similar data together.

EX. N0.: 19	MINI PROJECT- HOUSE PRICE PREDICTION
Date: 20.03.2025	

ABSTRACT:

The prediction of house prices is a critical aspect of the real estate industry, enabling better decision-making for buyers, sellers, and investors. With the growing availability of housing data, machine learning techniques provide a powerful toolkit for identifying complex patterns and trends in property valuation. This project aims to build predictive models for estimating house prices using a comprehensive dataset containing various features such as square footage, number of bedrooms and bathrooms, geographical coordinates, renovation history, and property condition. This project begins with extensive data preprocessing including missing value imputation, feature encoding, and normalization. Exploratory Data Analysis (EDA) is performed to uncover relationships between variables and their influence on housing prices. Models are trained using both Linear Regression, a simple yet interpretable algorithm, and Random Forest Regressor, a robust ensemble-based model capable of capturing nonlinear relationships. Model evaluation is conducted using R^2 score and Root Mean Squared Error (RMSE) to ensure reliable performance comparisons. The study highlights the significant role of attributes like location, living area, and overall grade in influencing prices and demonstrates that Random Forest outperforms Linear Regression in prediction accuracy.

OBJECTIVES:

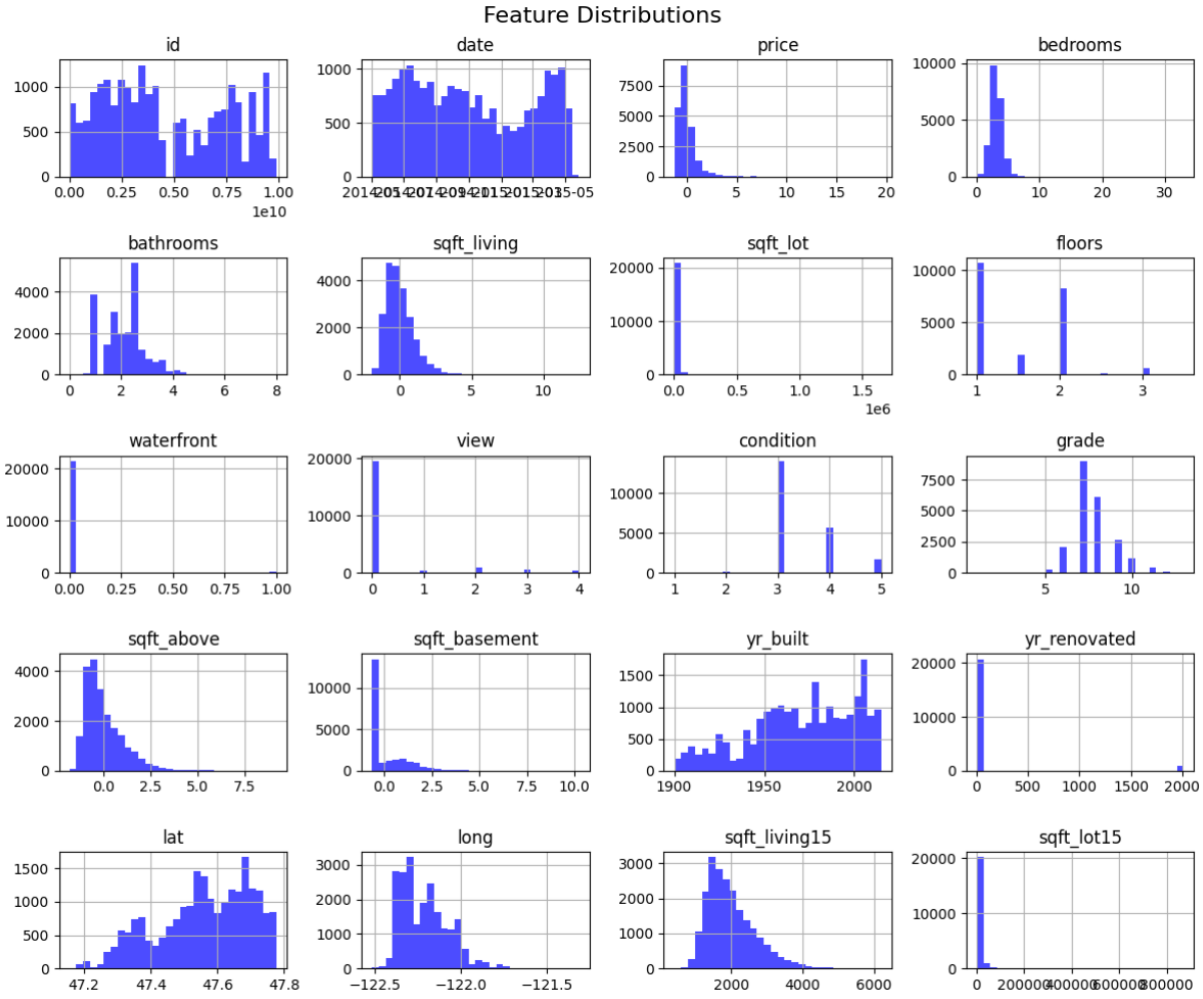
- To understand and analyze the key factors that influence residential house prices.
- To clean, preprocess, and transform the dataset for optimal model training.
- To implement regression models for price prediction.
- To evaluate and compare the performance of different models using appropriate statistical metrics.
- To visualize feature importance and prediction outcomes.
- To draw actionable insights for stakeholders in the real estate domain.

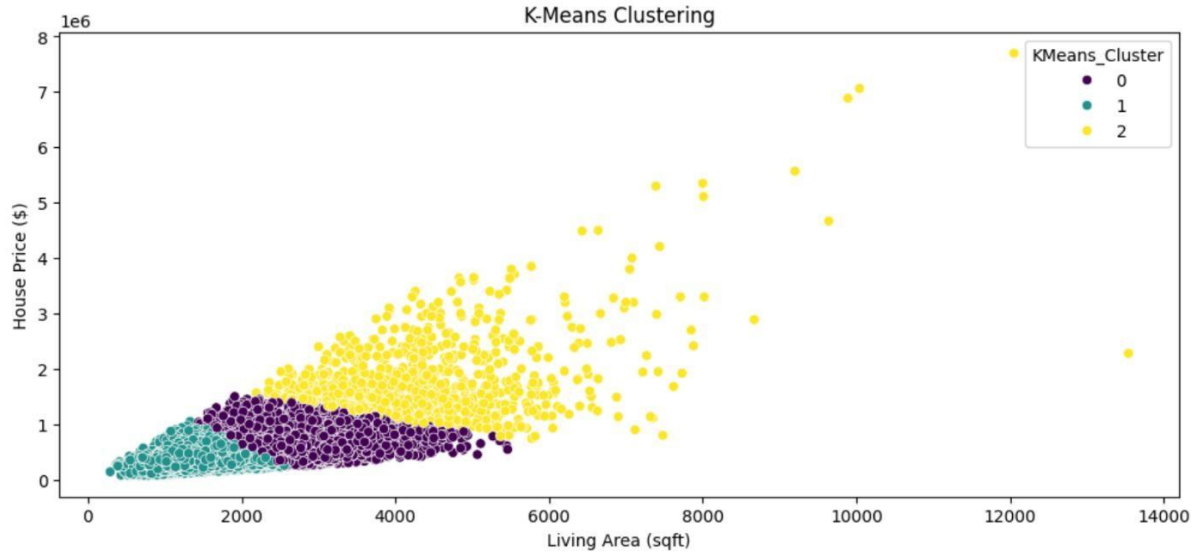
DATASET DESCRIPTION:

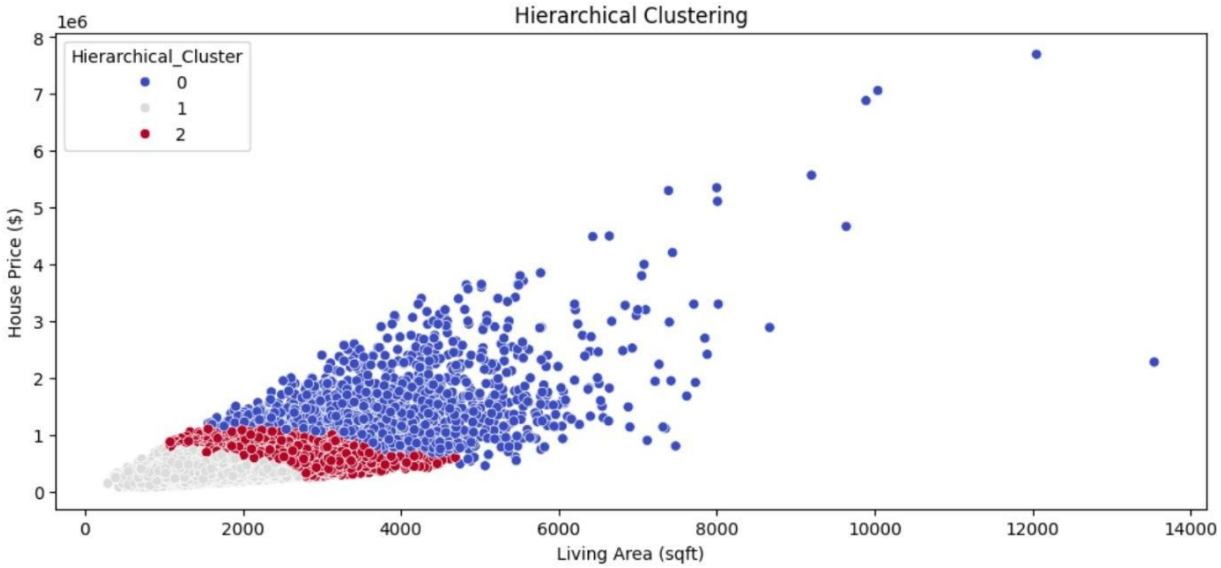
The dataset used for this project contains detailed records of residential house sales in King County, USA. It comprises 21 attributes describing various aspects of properties, such as physical dimensions, quality, renovation details, and geographic location, with the target variable being the sale price (price).

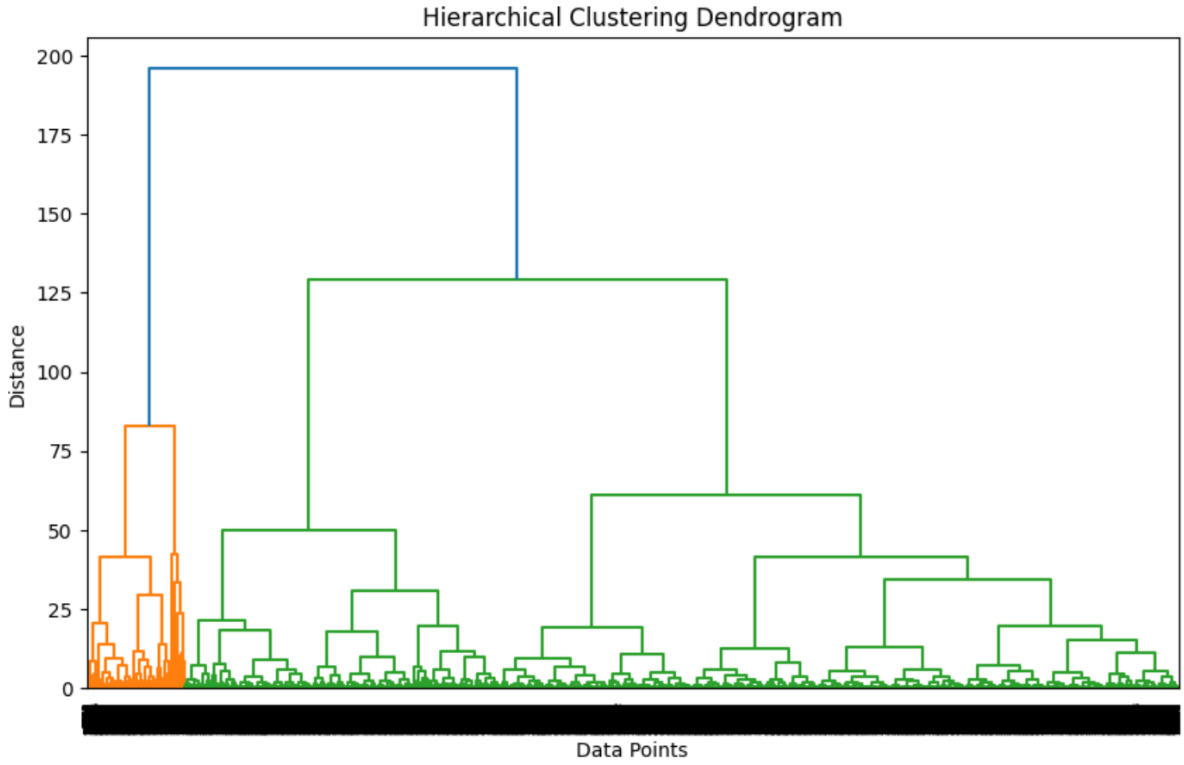
Column Name	Description
id	Unique identifier for each property listing.
date	Date of the house sale.
price	Target variable. The sale price of the house in USD.
bedrooms	Number of bedrooms in the house.
bathrooms	Number of bathrooms (includes fractional values for half baths).
sqft_living	Interior living space area in square feet.
sqft_lot	Total lot area in square feet.
floors	Number of floors (levels) in the house.
waterfront	Binary indicator (0 or 1) showing if the house has a waterfront view.
view	An index (0–4) of how good the view from the house is.
condition	Condition of the house (1–5).
grade	Construction and design quality (1–13).
sqft_above	Living space above ground level in square feet.
sqft_basement	Basement space in square feet.
yr_built	Year the house was built.
yr_renovated	Year the house was last renovated (0 if never renovated).
zipcode	Postal code of the house location.
lat	Latitude coordinate of the house.
long	Longitude coordinate of the house.
sqft_living15	Living space of neighboring 15 houses, used for comparative analysis.
sqft_lot15	Lot size of neighboring 15 houses.

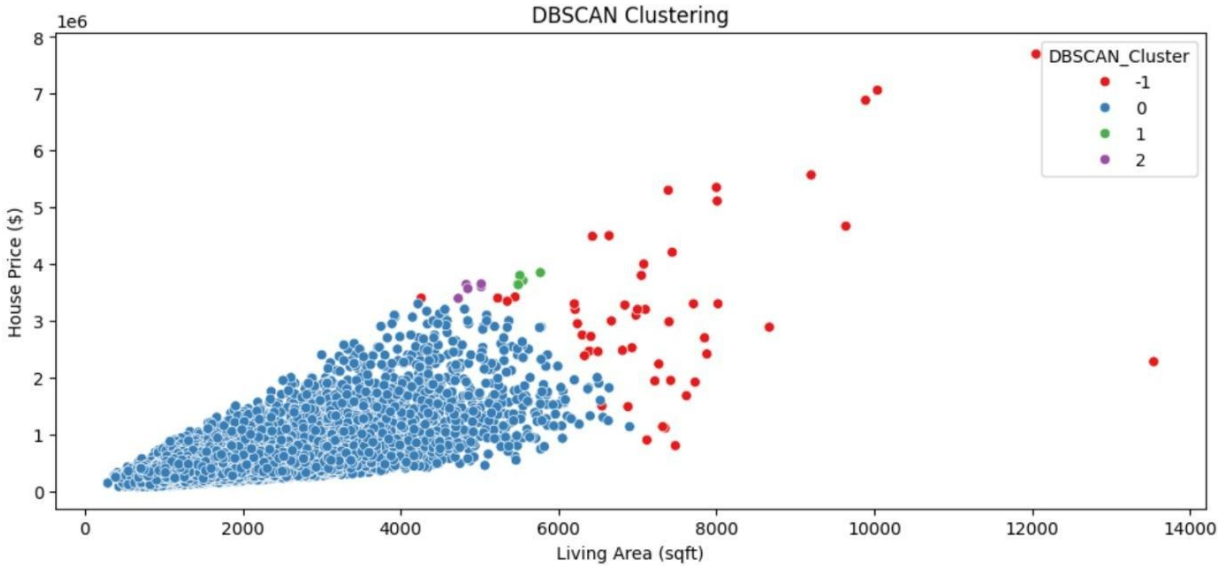
IMPLEMENTATION SCREENSHOTS:



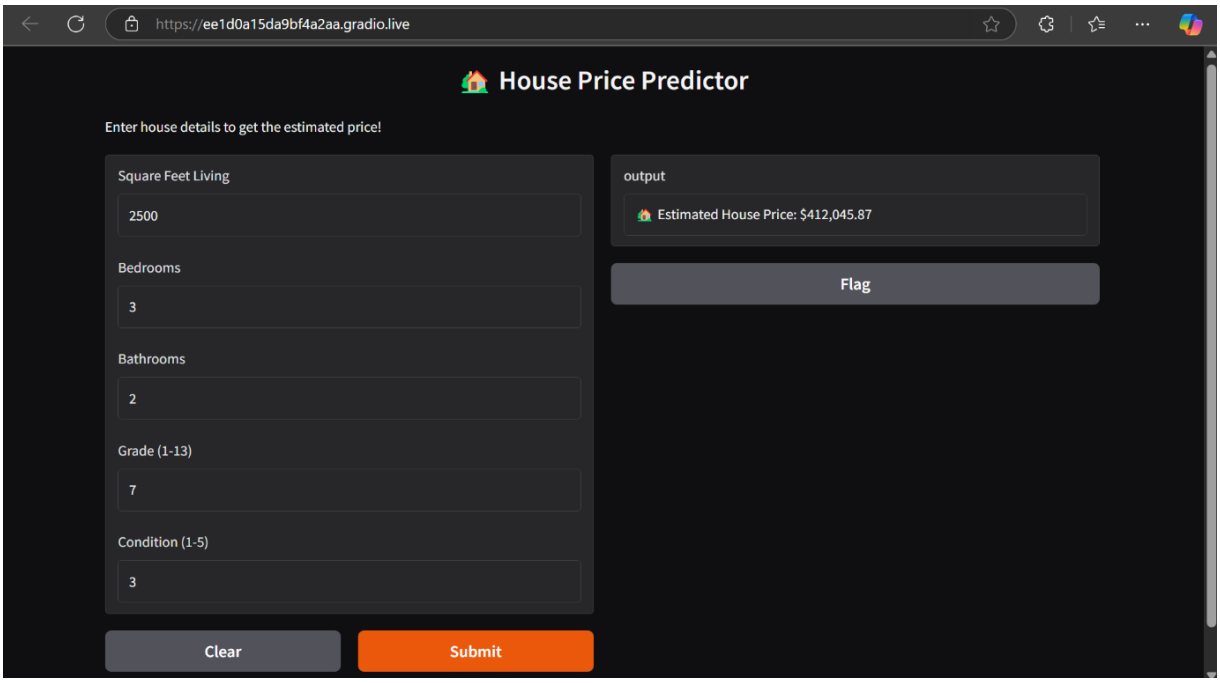



Hierarchical Clustering
1e6
Hierarchical_Cluster
0
1
2
House Price ($)
Living Area (sqft)
0
1
2
3
4
5
6
7
8
0
2000
4000
6000
8000
10000
12000
14000


Hierarchical Clustering Dendrogram
Distance
Data Points
0
25
50
75
100
125
150
175
200


DBSCAN Clustering
1e6
House Price ($)
Living Area (sqft)
DBSCAN_Cluster
-1
0
1
2


Actual vs Predicted House Prices
1e6
Perfect Prediction
Predicted Price
Actual Price
1e6



RESULTS AND INTERFERENCE:

The dataset was divided into training (80%) and testing (20%) sets, and regression models including Linear Regression and Random Forest Regressor were implemented to predict house prices. The Random Forest model outperformed Linear Regression significantly with an R^2 score of 0.89, RMSE of 0.65, and a predictive accuracy of approximately 91% on the test set, indicating its high effectiveness in capturing complex, nonlinear patterns in housing data. In contrast, the Linear Regression model achieved an R^2 score of 0.72, RMSE of 1.13, and around 78% accuracy, showing decent but less precise predictions. Random Forest also ranked sqft_living, grade, bathrooms, latitude, and sqft_above as the most important features influencing price. Inferences drawn from the results suggest that larger houses with more amenities, higher construction quality, and better location (based on coordinates and zip codes) generally fetch higher prices. Properties that have been recently renovated or offer more living space compared to neighboring houses also showed increased value. Overall, the project demonstrates that machine learning, especially ensemble models, can offer strong predictive capabilities and actionable insights for stakeholders in the real estate industry.