SREE NARAYANA GURUKULAM COLLEGE OF ENGINEERING

KADAYIRUPPU, KOLENCHERY 682311

20MCA241 DATA SCIENCE LAB

LABORATORY RECORD

YEAR: 2023 TO 2024

NAME: ADARSH P K
SEMESTER: 3 **ROLL NO: 05**
BRANCH: COMPUTER APPLICATIONS

Certified that this is a Bonafide Record of Practical work done in partial fulfillment of the requirements for the award of the Degree in Master of Computer Applications of Sree Narayana Gurukulam College of Engineering.

Kadayiruppu
Date:

Prof.Dr. Sandhya R
Head of the Department Course Instructor

Submitted for University Practical Examination

Reg. No: SNG22MCA-2005 **on**

External Examiner Internal Examiner

INDEX PAGE

Program No. 1
Aim: Review of python programming.

1. Write code to:
    a) Create different numeric data print its value and type
    b) Perform addition subtraction multiplication and division on numbers in Python

```
#write your code here
# a. Create different numeric data, print its value and type
a= 10 #integer
b= 10.23 #float
c = 10+2j #complex

#printing types
print("type(a):",type(a))
print("type(b):", type(b))
print("type(c):",type(c))


print("value of a: ", a)
print("value of b: ", b)
print("value of c: ", c)


#b. Perform addition subtraction multiplication and division on numbers in Python.
print("Addition : ",a+b);
print("Subtraction   : ",a-b);
print("Multiplication : ",a*b);
print("Division : ",a/b);
print("Floor Division : ",a//b);
```

```
type(a): <class 'int'>
type(b): <class 'float'>
type(c): <class 'complex'>
```

```
value of a: 10
value of b: 10.23
value of c: (10+2j)
Addition: 20.23
Subtraction: -0.23000000000000043
Multiplication: 102.30000000000001
Division : 0.9775171065493645 Floor
Division : 0.0
```

2. Write code to

   1. Create a list contain Reg No., Name and Mark of 2 subjects(out of 100) of a student and Append one more mark to the
   2. Print the list

```
#write your code here
# Create a list for the student's information
student info = []

# Add registration number, name, and two initial subject marks
reg no = "12345"
name = "John Doe"
marks_subject1 = 85
marks_subject2 = 92
student_info.append( [reg_no, name, marks_subject1, marks_subject2])

# Append one more mark to the list
new mark = 78
student info[0].append (new mark)
# Print the updated student information
```

```
print("Student Information:")
print(f"Registration Number: {student info[0][0]}")
print (f"Name: {student info[0][1]}")
print(f"Subject 1 Marks: {student info[0][2]}")
print(f"Subject 2 Marks: {student info[0][3]}")
print(f"Additional Mark (Appended) : {student info[0][4]}")
```

```
Student Information:
Registration Number: 12345
Name: John Doe
Subject 1 Marks: 85
Subject 2 Marks: 92
Additional Mark (Appended): 78
```

** 3. Update the value of the second mark **

```
#write your code here
# Update the value of the second subject mark
new_subject2_mark = 95
student info[0][3] = new_subject2_mark

#print the updated student information
Print(“updated student information:”)
print(f"Registration Number: {student info[0][0]}")
print(f"Name: {student info[0][1]}")
print (f"Subject 1 Marks: {student info[0][2]}")
print(f"Updated Subject 2 Marks: {student info[0][3]}")
print(f"Additional Mark: {student info[0][4]}")
```

```
Student Information:
Registration Number: 12345
```

```
Name: John Doe

Subject 1 Marks: 85

Subject 2 Marks: 92

Additional Mark (Appended): 78
```

** 3. Update the value of the second mark**

```
#write your code here

# Update the value of the second subject mark

new_subject2_mark = 95

student_info[0][3] = new_subject2_mark


# print the updated student information

Print("updated student information:")

print(f"Registration Number: |{student_info[0][0]}")

print(f"Name: {student_info[0][1]}") print(f"Subject 1 Marks: {student_info[0][2]}")

print(f"Updated Subject 2 Marks: {student_info[0][3]}")

print(f"Additional Mark: {student_info[0][4]}")
```

```
        Updated Student Information:

        Registration Number : 12345

        Name: John Doe

        Subject 1 Marks: 85 updated

        Subject 2 Marks: 95

        Additional Mark: 78
```

** 4. Create a random list contain 10 elements with values between 40 and 80**

```
#write your code here

import random

# Create an empty list to store the random elements

random_list = []

# Generate 10 random elements between 40 and 80 (inclusive)
```

```
for _ in range(10):
    random_element = random.randint(40, 80)
    random_list.append(random_element)
# Print the random list
print("Random List:")
print(random_list)
```

```
    Random List:
    [65, 65, 77, 69, 42, 53, 63, 68, 55, 62]
```

** 5. Decrement the value of each element in the above list**

```
#write your code here
# Define the value to decrement by
decrement_value = 2 # You can change this to any value you want

# Decrement the value of each element in the list
decremented_list = [element - decrement_value for element in random_list]

# Print the decremented list
print("Decremented List:")
print(decremented_list)
```

```
      Decremented List:
      [63, 63, 75, 67, 40, 51, 61, 66, 53, 60]
```

Create a dictionary store the Name, City and Mobile number of a person Print the Name and Mobile number

```
#wite your code here
# Create a dictionary to store person information
person_info = {
    "Name": "John Doe",
    "City": "New York",
    "Mobile Number": "123-456-7890" }
# Print the Name and Mobile Number
print("Name:", person_info["Name"])
print("Mobile Number:", person_info["Mobile Number"])
```

```
    Name: John Doe
    Mobile Number: 123-456-7890
```

Program No. 2
Aim: Matrix operations.

This colab is designed for you to practice and solve the activities that are based on the following concepts:

1. NumPy Arrays
2. Matrix operations (using vectorization) and Trasformation
3. Python Lists
4. SVD using Python.

Import Library

```
# import libraries
import numpy as np
```

Create 1D Array contain 9 elements print its shape

```
# write code here
# Create a 1D array with 9 elements
array1d = np.arange(9)
print("1D Array:")
print(array1d)
print("Shape:",array1d.shape)
```

```
1D Array:
[0 1 2 3 4 5 6 7 8]
Shape: (9,)
```

Reshape the array to a 3x3 matrix

```
# Write your code here
# Reshape the array to a 3x3 matrix
array3x3 = array1d.reshape(3, 3)
print("\n3x3 Matrix:")
print(array3x3)
```

```
3x3 Matrix:
[[0 1 2]
[3 4 5]
[6 7 8]]
```

Add two 3x2 matrices and print the sum

```
# Write your code here
# Create two 3x2 matrices and add them
matrix1= np.array([[1, 2], [3, 4], [5, 6]])
matrix2 = np.array([[7, 8], [9, 10], [11, 12]])
sum_matrix = matrix1 + matrix2
print("\nSum of 3x2 Matrices:")
print(sum_matrix)
```

```
Sum of 3x2 Matrices:
[[ 8 10]
[12 14]
```

```
        [16 18]]
```

Multiply Two matrices of order 3x3 and 3x4

```
# Write your code here
# Multiply two matrices of order 3x3 and 3x4
matrix3x3 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
matrix3x4 = np.array([[9, 10, 11, 12], [13, 14, 15, 16], [17, 18, 19, 20]])
product_matrix = np.dot(matrix3x3, matrix3x4)
print("\nMatrix Multiplication (3x3 * 3x4):")
print(product_matrix)
        Matrix Multiplication (3x3 * 3x4):
        [[ 86 92 98 104]
        [203 218 233 248]
        [320 344 368 392]]
```

perform Elementwise multiplication of two 2x2 matrices

```
# Write your code here
# Perform element-wise multiplication of two 2x2 matrices
matrix2x2_1 = np.array([[1, 2], [3, 4]])
matrix2x2_2 = np.array([[5, 6], [7, 8]])
elementwise_product= matrix2x2_1 * matrix2x2_2
print("\nElement-wise Multiplication of 2x2 Matrices:")
print(elementwise_product)
        Element-wise Multiplication of 2x2 Matrices:
        [[ 5 12]
        [21 32]]
```

Transpose and Invers of a 3x3 matrix

```
# Write your code here
# Transpose and Inverse of a 3x3 matrix
matrix3x3 = np.array([[1, 2, 3], [0, 1, 4], [5, 6, 0]])
matrix3x3_transpose = matrix3x3. T
matrix3x3_inverse = np.linalg.inv(matrix3x3)
print("\nTranspose of 3x3 Matrix:")
print(matrix3x3_transpose)
print("\nInverse of 3x3 Matrix:")
print(matrix3x3_inverse)
        Transpose of 3x3 Matrix:
        [[1 0 5]
        [2 1 6]
        [3 4 0]]


        Inverse of 3x3 Matrix:
        [[-24. 18. 5.]
        [ 20. -15.-4.]
        [-5. 4. 1.]]
```

Mathematical Operations on a 3x3 matrix increment all the elements of a matrix by 5 multiply all the the elements of a matrix by 5 calculate the mean and SD of elements of a matrix

```
# Write your code here
# Mathematical operations on a 3x3 matrix
matrix3x3 += 5 # Increment all elements by 5
matrix3x3 *= 5 # Multiply all elements by 5

mean = np.mean(matrix3x3)
std_dev = np.std(matrix3x3)
print("\nMatrix After Operations:")
print(matrix3x3)
print("\nMean of Elements:", mean)
print("Standard Deviation:", std_dev)
```

```
    Matrix After Operations:
    [ [30 35 40]
    [45 50 55]
    [60 65 70]]

      Mean of Elements: 50.0
      Standard Deviation: 12.909944487358056
```

Perform SVD on a 3x2 matrix (use np.linalg.svd) print the values of U, S V

```
# Write your code here
# Perform SVD on a 3x2 matrix
matrix3x2 = np.array([[1, 2], [3, 4], [5, 6]])
U, S, V = np.linalg.svd(matrix3x2) print("\nSVD- U matrix:")
print(U)
print("\nSVD - S matrix (singular values):")
print(S)
print("\nSVD - V matrix:")
print(V)
```

```
    SVD - U matrix:
    [[-0.2298477 0.88346102 0.40824829]
    [-0.52474482 0.24078249 -0.81649658]
    [-0.81964194 -0.40189603 0.40824829]]

    SVD - S matrix (singular values):
    [9.52551809 0.51430058]

    SVD - V matrix:
    [[-0.61962948 -0.78489445]
```

```
[-0.78489445 0.61962948]]
```

Indexing and Slicing Create 3x4 array

1. Print last row elements
2. Print last column elements
3. Print all elements in the array except rst row rst column

```
# write code here
# Create a 3x4 array
array3x4 = np.arange(1, 13).reshape(3, 4)
print("\n3x4 Array:")
print(array3x4)

#Indexing and slicing
print("\nLast Row Elements:")
print(array3x4[-1, :])

print("\nLast Column Elements:")
print(array3x4[:, -1])

print("\nAll Elements Except First Row and First Column:")
print(array3x4[1:, 1:])
```

```
3x4 Array:
[[ 1 2 3 4]
[ 5 6 7 8]
[ 9 10 11 12]]

Last Row Elements:
[ 9 10 11 12]

Last Column Elements:
[ 4 8 12]

All Elements Except First Row and First Column:
[[ 6 7 8]
[10 11 12]]
```

Convert the array to a list

```
#write your code here
# Convert the array to a list
array_as_list = array3x4. tolist()
print("\nArray Converted to List:")
print(array_as_list)
```

Create a 3 X 3 Matrix with values ranging from 2 to 10.
For Example:

```
[[ 2 3 4]
[5 6 7]
[8 9 10]]
```

Step 1: Import numpy module.
Step 2: Use arange() function to create array of numbers from 2 to 10
Step 3: Use reshape() function to reshape your array having 3 rows and 3 columns. Store this reshaped array in a variable x ..

```
# Convert the array to a list
array_as_list =array3x4. tolist()
print("\nArray Converted to List:")
print(array_as_list)
```

```
Array Converted to List:
[[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]
```

Create and Update a Null NumPy Array
Create a null NumPy array of size 9 and update the sixth value to 11.

A null array is basically an array with all elements as 0 .

Step 1: Import the Numpy module as np .

Step 2: Create a null array by passing the size i.e. 10 inside the np.zeros() function and store it in a variable null_arr .

Step 3: Print the null array.

Step 4: Now update the 6th element of the array by using **list indexing** method.

Step 5: Print the updated array in the output

```
# Write your code here
# Create a 3x3 Matrix with values ranging from 2 to 10
matrix3x3_range np.arange(2, 11).reshape(3, 3)
print("\n3x3 Matrix with Values Ranging from 2 to 10:")
print(matrix3x3_range)

# Create and Update a Null NumPy Array
null_arr = np.zeros(9)
null_arr[5] =11
print("\nNull NumPy Array:")
print(null_arr)
```

```
3x3 Matrix with Values Ranging from 2 to 10:

[[ 2 3 4]

[ 5 6 7]

[8 9 10]]
```

```
Null NumPy Array:
[ 0. 0. 0. 0. 0. 11. 0. 0. 0.]
```

In the above program we have created a null array by using the np.zeros() function of the numpy module

Populate a Number List

Write a program that populates a list by numbers that lies in the range of 0 - 50 and also divisible by 5. Use List Comprehension method

```
# Write a program to populate a number list divisible by 5 in a range 0 - 50 #
Populate a number list divisible by 5 in a range 0 - 50
divisible_by_5 = [x for x in range(0, 51) if x % 5 == 0]
print("\nNumbers Divisible by 5 in Range 0-50:”)
print(divisible_by_5)
```

```
Numbers Divisible by 5 in Range 0-50:
[0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50]
```

Program No. 3

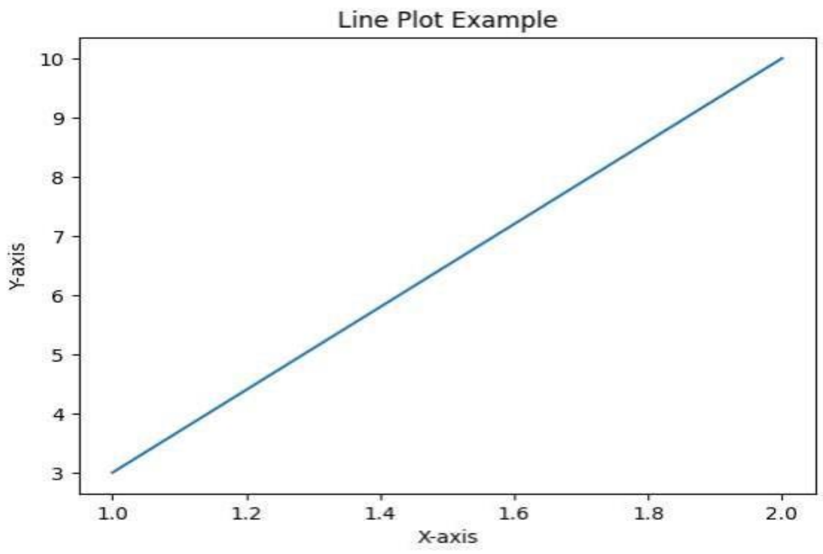
Aim: Programs using matplotlib, seaborn for data visualisation (Data Visualization 1).

Line, Bar,Scatter & Pie plots

```
#import library
import matplotlib.pyplot as plt
import numpy as np
```

Draw a line in a diagram from position (1, 3) to position (2, 10)

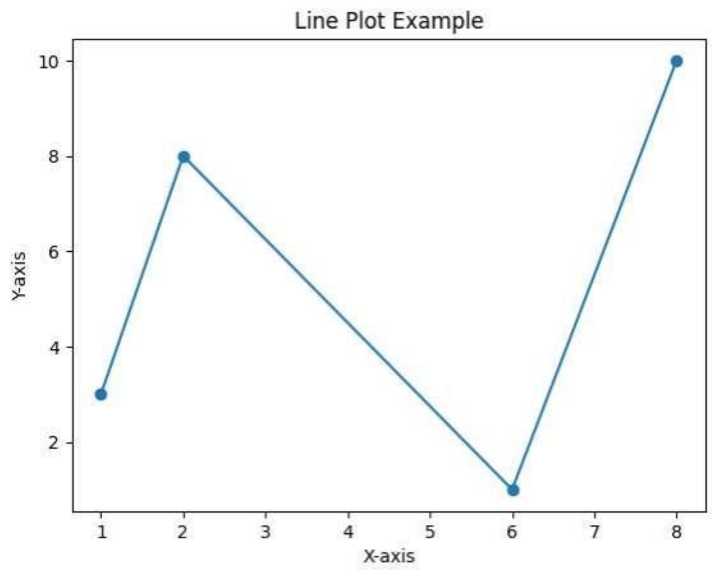
```
# Define the coordinates of the two points x
= np.array([1, 2])
y = np.array([3, 10])

# Create a line plot
plt.plot(x, y)

# Add labels and a title
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Line Plot Example')

# Show the plot
plt. show()
```



Draw a line in a diagram from position (1, 3) to (2, 8) then to (6, 1) and nally to position (8, 10)

```
# Define the coordinates of the four points
```

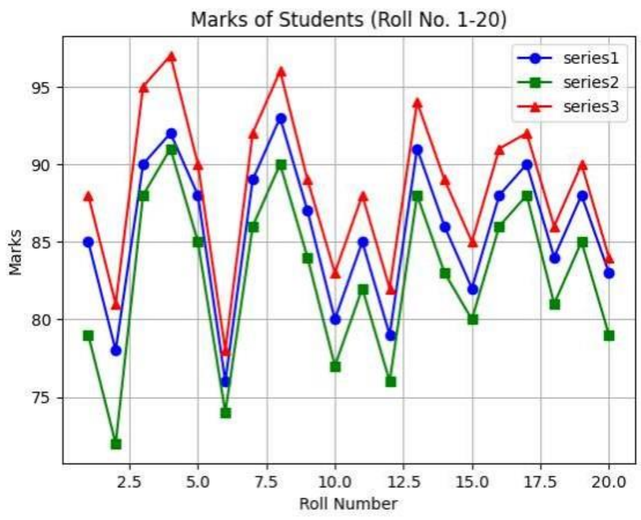
```
x= [1, 2, 6, 8]

y = [3, 8, 1, 10]


# Create a line plot

plt.plot(x, y, marker='o', linestyle='-')


# Add labels and a title

plt.xlabel('x-axis')

plt.ylabel('Y-axis')

plt.title('Line Plot Example')


# Show the plot

plt. show()
```



*Multiple Plots - plot series1, series2 and series3 marks of students with roll no 1 to 20 ** Give appropriate labels to graph, give color to lines

```
# Roll numbers

roll_numbers = list(range(1, 21))


# Series1, Series2, and Series3 marks for the students

series1_marks = [85, 78, 90, 92, 88, 76, 89, 93, 87, 80, 85, 79, 91, 86, 82, 88, 90, 84, 88, 83]
```

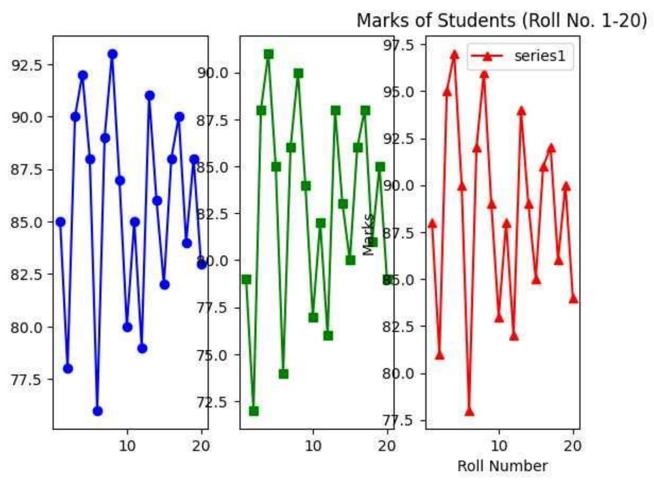
```
series2_marks = [79, 72, 88, 91, 85, 74, 86, 90, 84, 77, 82, 76, 88, 83, 80, 86, 88, 81, 85, 79]

series3_marks = [88, 81, 95, 97, 90, 78, 92, 96, 89, 83, 88, 82, 94, 89, 85, 91, 92, 86, 90, 84]

# Plot series1 marks in blue

plt.plot(roll_numbers, series1_marks, marker='o', linestyle='-', color='blue', label='Series1 Marks')

# Plot series2 marks in green

plt.plot(roll_numbers, series2_marks, marker='s', linestyle='-', color='green', label='Series2 Marks')

# Plot series3 marks in red

plt.plot(roll_numbers, series3_marks, marker='^', linestyle='-', color='red', label='Series3 Marks')

# Add labels and a title

plt.xlabel("Roll Number")

plt.ylabel("Marks")

plt.title('Marks of Students (Roll No. 1-20)')

plt.legend(["series1","series2", "series3"])


# Show the plot

plt.grid()

plt.show()
```



Subplots - plot the series1 and series2 and series3 marks as subplots in different rows

```
# Roll numbers
```

```
roll_numbers = list(range(1, 21))


# Series1, Series2, and Series3 marks for the students

series1_marks = [85, 78, 90, 92, 88, 76, 89, 93, 87, 80, 85, 79, 91, 86, 82, 88, 90, 84, 88, 83]

series2_marks = [79, 72, 88, 91, 85, 74, 86, 90, 84, 77, 82, 76, 88, 83, 80, 86, 88, 81, 85, 79]

series3_marks = [88, 81, 95, 97, 90, 78, 92, 96, 89, 83, 88, 82, 94, 89, 85, 91, 92, 86, 90, 84]

# Plot series1 marks in blue

plt.subplot(1,3,1)

plt.plot(roll_numbers, series1_marks, marker='o', linestyle='-', color='blue', label='Series1 
Marks')


# Plot series2 marks in green

plt.subplot(1,3,2)

plt.plot(roll_numbers, series2_marks, marker='s', linestyle='-', color='green', label='Series2 
Marks')


# Plot series3 marks in red

plt.subplot(1,3,3)

plt.plot(roll_numbers, series3_marks, marker='^', linestyle='-', color='red', label='Series3 
Marks')


# Add labels and a title

plt.xlabel("Roll Number")

plt.ylabel("Marks")

plt.title('Marks of Students (Roll No. 1-20)')

plt.legend(["series1","series2", "series3"])


# Show the plot

plt.show()
```



Draw a bar chart of the popularity of programming Languages.

Programming languages:(Java, Python ,PHP ,JavaScript,C#,C++) Popularity : (22.2, 17.6, 8.8 , 8, 7,76.7)

# Programming languages and their popularity

languages = np.array(['Java', 'Python', 'PHP', 'JavaScript', 'C#', 'C++'])

popularity = np.array([22.2, 17.6, 8.8, 8, 7, 76.7])

# Create a bar chart

plt.bar(languages, popularity, color='skyblue')

#Add labels and a title

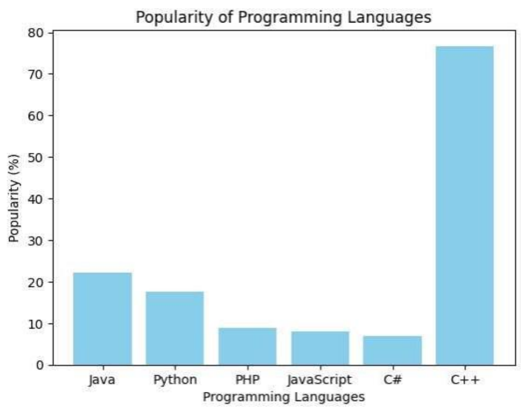
plt.xlabel('Programming Languages')

plt.ylabel('Popularity (%)')

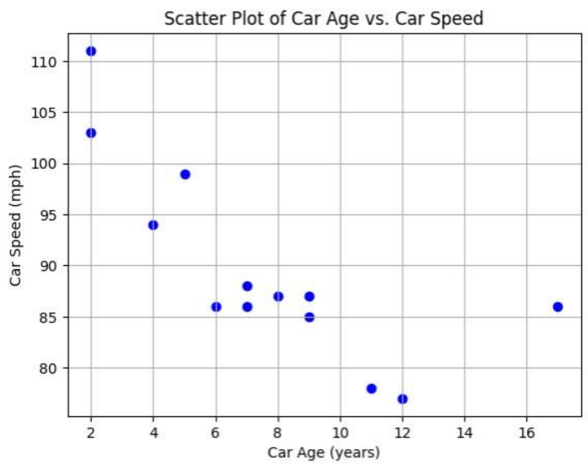
plt.title('Popularity of Programming Languages')

#Show the plot

plt.show()

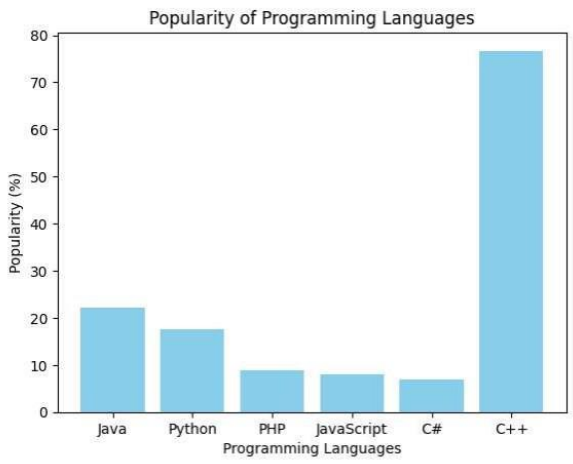


```
# Car ages and car speeds data
car_age = [5, 7, 8, 7, 2, 17, 2, 9, 4, 11, 12, 9, 6]
car_speed = [99, 86, 87, 88, 111, 86, 103, 87, 94, 78, 77, 85, 86]
#Create a scatter plot
plt.scatter(car_age, car_speed, color='blue')
#Add labels and a title
plt.xlabel('Car Age (years)')
plt.ylabel('Car Speed (mph)')
plt.title('Scatter Plot of Car Age vs. Car Speed')
#Show the plot
plt.grid()
plt.show()
```



Draw a bar chart of the popularity of programming Languages. Programming languages: (Java, Python ,PHP ,JavaScript,C#,C++) Popularity : (22.2, 17.6, 8.8 , 8, 7,76.7)

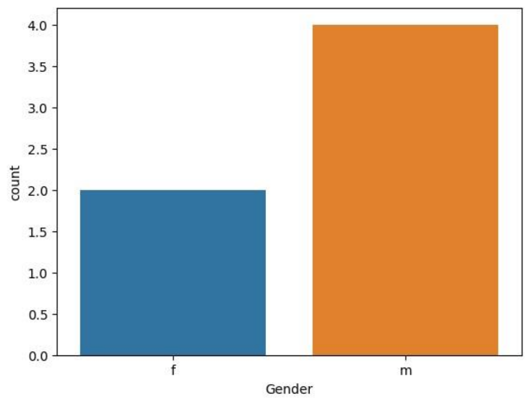
```
# Programming languages and their popularity
languages = np.array(['Java', 'Python', 'PHP', 'JavaScript', 'C#', 'C++'])
popularity = np.array([22.2, 17.6, 8.8, 8, 7, 76.7])
# Create a bar chart
plt.bar(languages, popularity, color='skyblue')
#Add labels and a title
plt.xlabel('Programming Languages' )
plt.ylabel('Popularity (%)')
plt.title('Popularity of Programming Languages')
#Show the plot
plt.show()
```



Pie Chart Draw pie char using the data of students Placed Sectors--- IT-80, Marketing-34, Banking-5, BPO-40,Marine-37

```
# Data for students placed in sectors
sectors = ['IT', 'Marketing', 'Banking', 'BPO', 'Marine']
students_placed = [80, 34, 5, 40, 37]


# Create a pie chart
plt.pie(students_placed, labels=sectors, colors=['skyblue', 'lightgreen', 'lightcoral',
'lightyellow', 'lightpink'], autopct="%2.1f%%", shadow=True)
```

```
# Add a title
plt. title('Students Placed in Different Sectors')


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


stud=np. array(['f','m', 'm', 'm', 'f', 'm'])
df=pd.DataFrame(stud, columns=['Gender'])
print(stud)
sns.countplot(x='Gender',data=df)
plt. show()
```



Program No. 4

Aim : Programs to handle data using pandas (data visualization using student dataset).

Data Visualisation

Instructions

Do the following tasks:

1. Create a DataFrame for a **students.csv** le which is available in yourclassoom repository:
2. Display the rst 5 rows of the DataFrame.
3. Display the last 5 rows of the DataFrame.
4. Find the number of rows and columns.
5. Check for the missing values in the DataFrame.
6. Create a scatter plot and a line plot between:
   'Finalmark and Attendance`
   FinalMark and Series1
   Finalmark and Series2
   Finalmark and Attendace/10+Series1+series2

   such that the FinalMark values are plotted on the Vertical axis and other values are plotted on the Horizontal axis.

1. Create a DataFrame

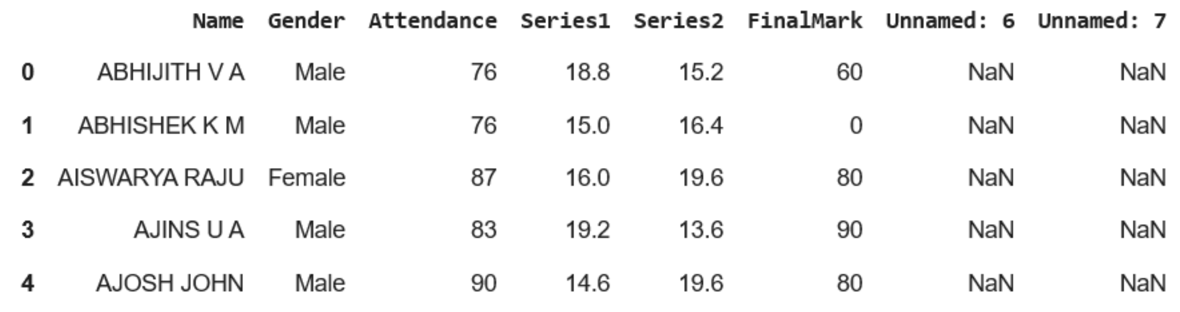
Import the required modules. Create a DataFrame for the dataset and store it in the df variable.

```
#Create a DataFrame for the dataset and store it in the df variable.
import pandas as pd
import matplotlib.plt as plt
stud_df=pd.read_csv("/content/students.csv")
```

2. Display The First 5 Rows

Here you have to display the rst ve rows of the df DataFrame

```
# Display the first 5 rows using the 'head()' function
stud_df.head()
```



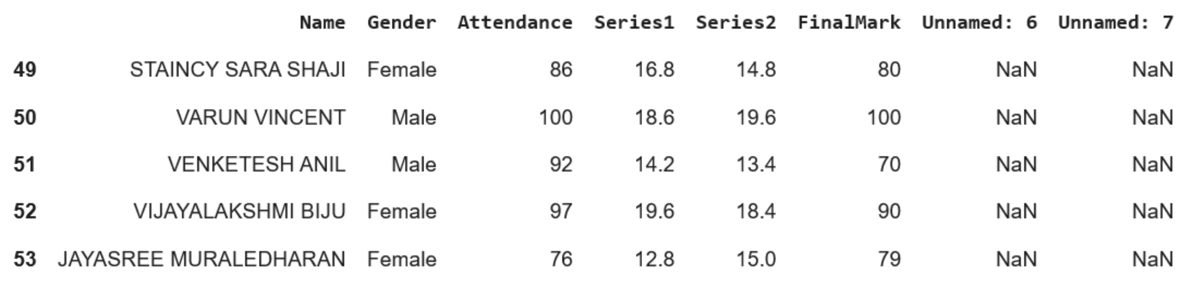
	Name	Gender	Attendance	Series1	Series2	FinalMark	Unnamed: 6	Unnamed: 7
0	ABHIJITH V A	Male	76	18.8	15.2	60	NaN	NaN
1	ABHISHEK K M	Male	76	15.0	16.4	0	NaN	NaN
2	AISWARYA RAJU	Female	87	16.0	19.6	80	NaN	NaN
3	AJINS U A	Male	83	19.2	13.6	90	NaN	NaN
4	AJOSH JOHN	Male	90	14.6	19.6	80	NaN	NaN

3. Display The Last 5 Rows

Here you have to display the last ve rows of the df DataFrame

# Display the last 5 rows using the 'tail()' function

stud_df.tail()



	Name	Gender	Attendance	Series1	Series2	FinalMark	Unnamed: 6	Unnamed: 7
49	STAINCY SARA SHAJI	Female	86	16.8	14.8	80	NaN	NaN
50	VARUN VINCENT	Male	100	18.6	19.6	100	NaN	NaN
51	VENKETESH ANIL	Male	92	14.2	13.4	70	NaN	NaN
52	VIJAYALAKSHMI BIJU	Female	97	19.6	18.4	90	NaN	NaN
53	JAYASREE MURALEDHARAN	Female	76	12.8	15.0	79	NaN	NaN

4. Display Number Of Rows & Columns

Now, display the number of rows and columns that are present in the df DataFrame

# Display the number of rows and columns using the 'shape' keyword stud_df.shape

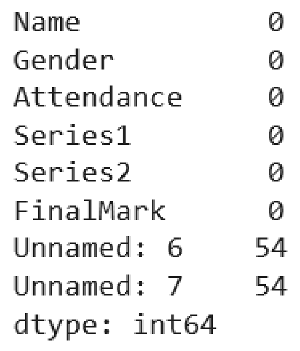
```
(54, 8)
```

5. Check For The Missing Values

Check whether the df DataFrame contains the missing values.

# Check for the missing values using the 'isnull()' function.

stud_df.isnull().sum()



```
Name           0
Gender         0
Attendance     0
Series1        0
Series2        0
FinalMark      0
Unnamed: 6    54
Unnamed: 7    54
dtype: int64
```

Hint: Use the sum() function on top of the isnull() to nd the total number of True values in each column

6. Scatter & Line Plots

Now you have to create the scatter plots and line plots between the following columns:

1. 'FinalmarkandAttendance`
2. 'FinalmarkandSeries1`
3. 'FinalmarkandSeries2`
4. 'FinalmarkandAttendance/10+Series1+Series2`

# Scatter plot between Finalmark and Attendance

plt.scatter(stud_df['FinalMark'], stud_df['Attendance'])

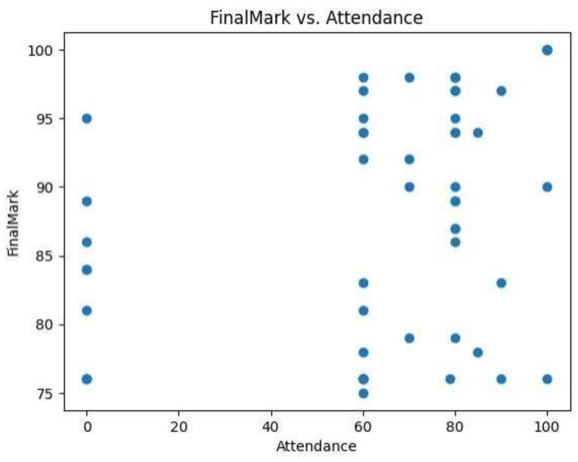
plt.title('FinalMarkvs.Attendance')

plt.xlabel('Attendance')

plt.ylabel('FinalMark')

plt.show()

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



# Line plot between Finalmark and Attendance

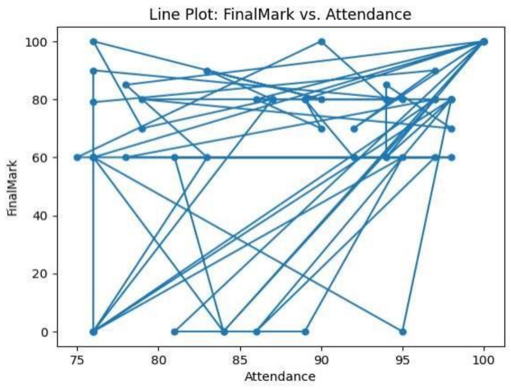
plt.plot(stud_df['Attendance'],stud_df['FinalMark'], linestyle='-', marker='o', ms=5)

plt.title('Line Plot: FinalMark vs. Attendance')

plt.xlabel('Attendance')

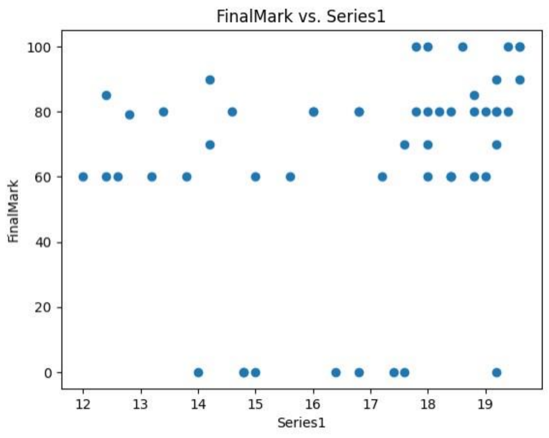
plt.ylabel('FinalMark')

plt.show()



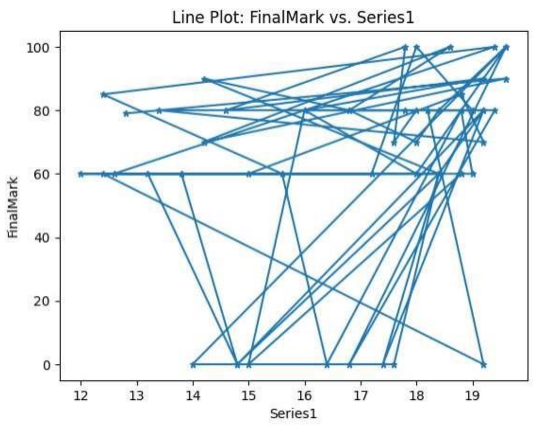
```
# Scatter plot between Finalmark and Series1
plt.scatter(stud_df['Series1'], stud_df['FinalMark'])
plt.title('FinalMarkvs.Series1')
plt.xlabel('Series1')
plt.ylabel('FinalMark')
plt.show()
```

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



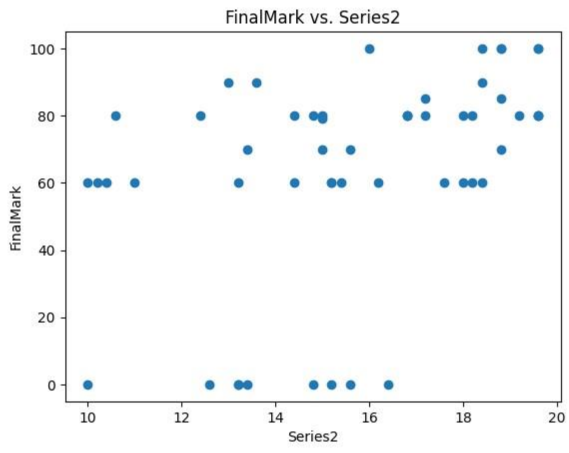
```
# Line plot between Finalmark and Series1
plt.plot(stud_df['Series1'], stud_df['FinalMark'], linestyle='-', marker='*', ms=5)
plt.title('Line Plot: FinalMark vs. Series1')
plt.xlabel('Series1')
plt.ylabel('FinalMark')
plt.show()
```

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



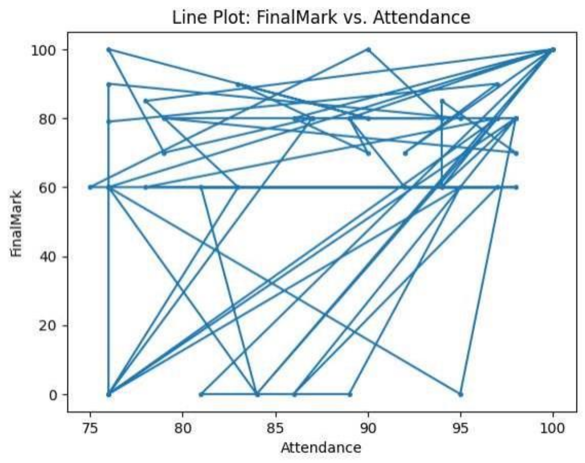
```
# Scatter plot between Finalmark and Series2
plt.scatter(stud_df['Series2'], stud_df['FinalMark'])
plt.title('FinalMark vs. Series2')
plt.xlabel('Series2')
plt.ylabel('FinalMark')
plt.show()
```

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



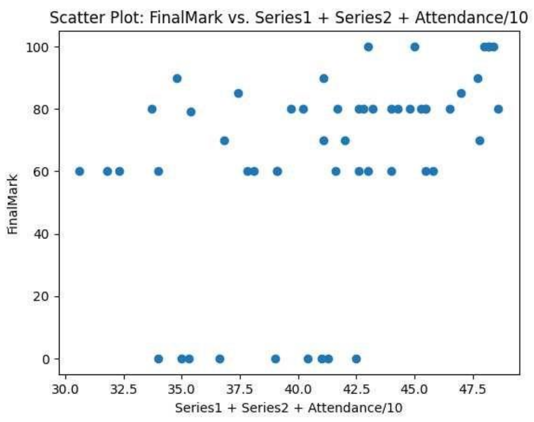
```
# Line plot between Finalmark and Series2
plt.plot(stud_df['Attendance'],stud_df['FinalMark'],linestyle='',marker='.',ms=5)
plt.title('Line Plot: FinalMark vs. Attendance')
plt.xlabel('Attendance')
plt.ylabel('FinalMark')
plt.show()
```

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

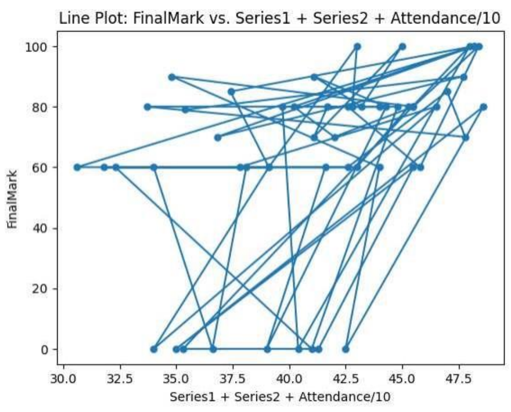


```
# Scatter plot between Finalmark and Series1+Series2+Attendance/10

stud_df['Series1_Series2_Attendance'] = stud_df['Series1'] + stud_df['Series2'] +
stud_df['Attendance'] / 10

plt.scatter(stud_df['Series1_Series2_Attendance'], stud_df['FinalMark'])

plt.title('Scatter Plot: FinalMark vs. Series1 + Series2 + Attendance/10')

plt.xlabel('Series1 + Series2 + Attendance/10')

plt.ylabel('FinalMark')

plt.show()
```

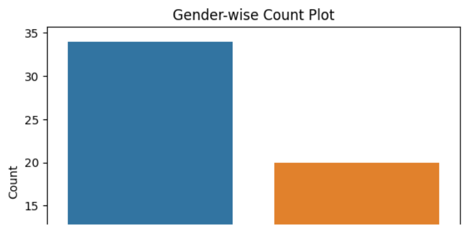
```
    Text(0, 0.5, 'FinalMark')
```



```
# Line plot between Finalmark and Series1+Series2+Attendance/10

plt.plot(stud_df['Series1_Series2_Attendance'], stud_df['FinalMark'], linestyle='-', marker='o',
markersize=5)

plt.title('Line Plot: FinalMark vs. Series1 + Series2 + Attendance/10')

plt.xlabel('Series1 + Series2 + Attendance/10')

plt.ylabel('FinalMark')

plt.show()
```



```
#Plot Gerderwise count

import seaborn as sns

sns.countplot(data=stud_df, x='Gender') plt.xlabel('Gender')

plt.ylabel('Count')

plt.title('Gender-wise Count Plot')

plt.show()
```



Program No. 5

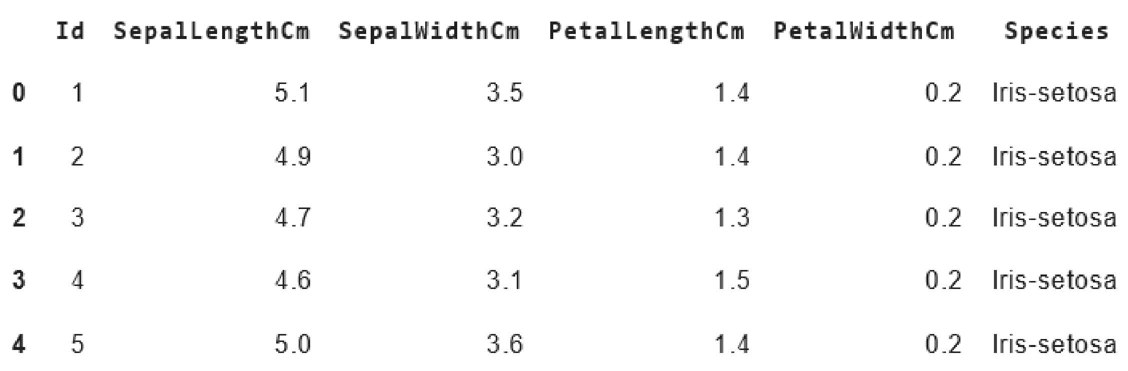
Aim: Program to implement k-NN classi cation using any standard dataset available in the public domain and nd the accuracy of the algorithm Use Iris dataset

Algorithm:

The class of an unknown instance is computed using the following steps:

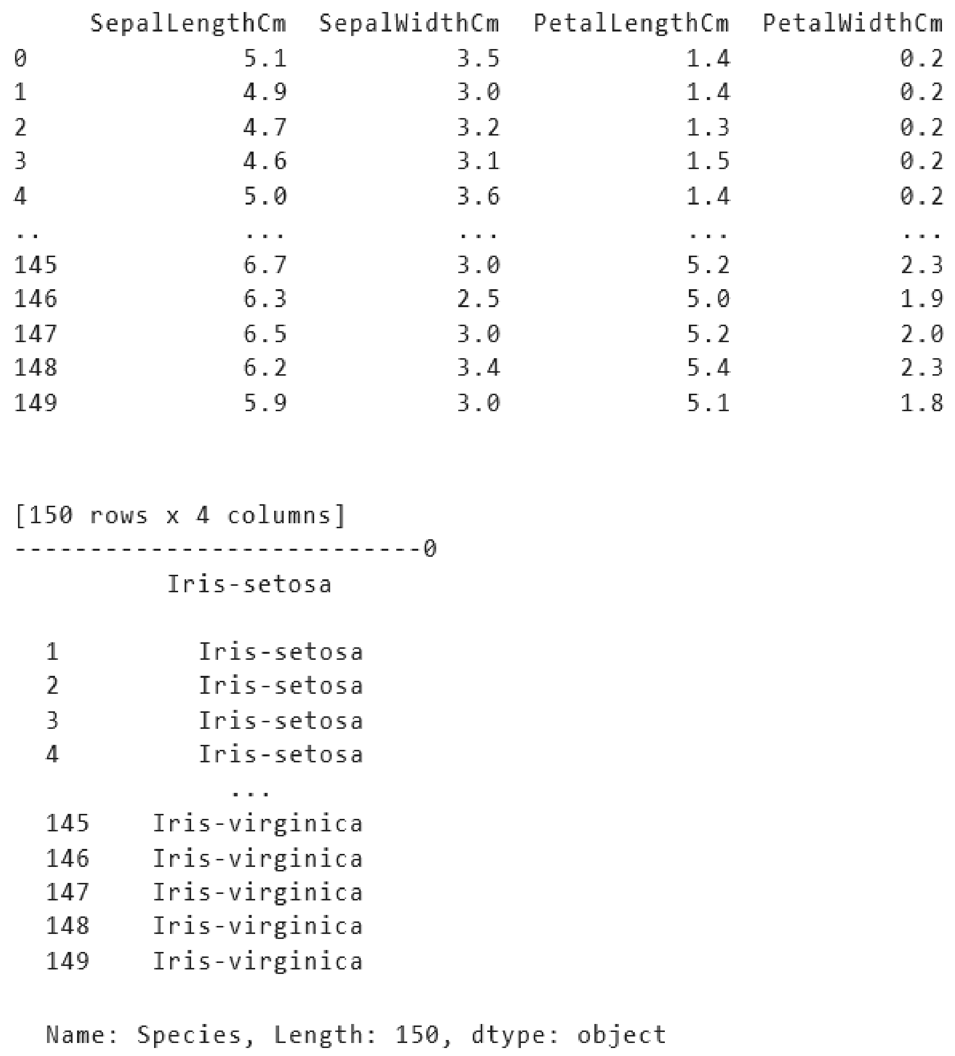
1. The distance between the unknown instance and all other training instances is computed.
2. The k nearest neighbors are identi ed.
3. The class labels of the k nearest neighbors are used to determine the class label of the unknown instance by using techniques like majority voting.

```
#import libraries
from sklearn import neighbors
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report


#load dataset import pandas as pd
df=pd.read_csv("/content/Iris.csv")
#print first 5 records
df.head()
```



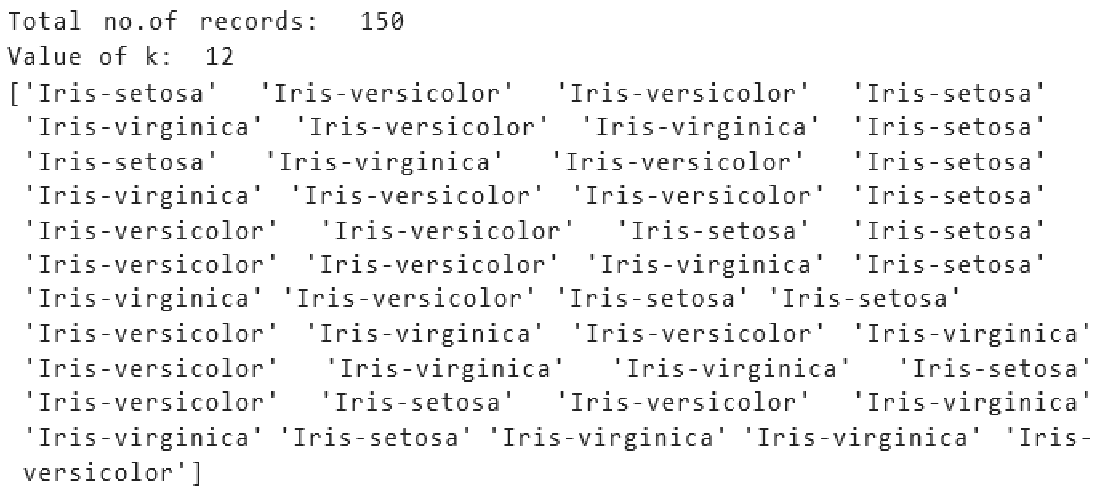
```
   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species
0   1            5.1           3.5            1.4           0.2  Iris-setosa
1   2            4.9           3.0            1.4           0.2  Iris-setosa
2   3            4.7           3.2            1.3           0.2  Iris-setosa
3   4            4.6           3.1            1.5           0.2  Iris-setosa
4   5            5.0           3.6            1.4           0.2  Iris-setosa
```

```
#separate features and target

x=df.iloc[:,1:5]

y=df.iloc[:,-1]

print(x)

print("--------------------------")

print(y)
```



```
     SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
0              5.1           3.5            1.4           0.2
1              4.9           3.0            1.4           0.2
2              4.7           3.2            1.3           0.2
3              4.6           3.1            1.5           0.2
4              5.0           3.6            1.4           0.2
..             ...           ...            ...           ...
145            6.7           3.0            5.2           2.3
146            6.3           2.5            5.0           1.9
147            6.5           3.0            5.2           2.0
148            6.2           3.4            5.4           2.3
149            5.9           3.0            5.1           1.8


[150 rows x 4 columns]
--------------------------0
          Iris-setosa

  1         Iris-setosa
  2         Iris-setosa
  3         Iris-setosa
  4         Iris-setosa
                ...
  145    Iris-virginica
  146    Iris-virginica
  147    Iris-virginica
  148    Iris-virginica
  149    Iris-virginica

  Name: Species, Length: 150, dtype: object
```

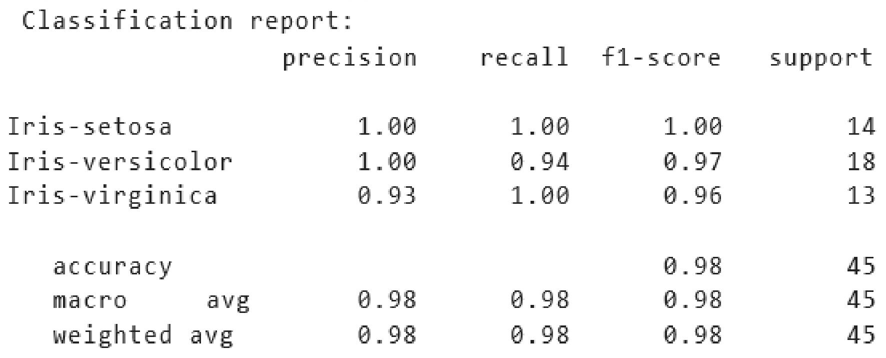
```
#split test and training set as x_train,x_test,y_train,y_test

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=1)


#print the shape of trainig and test inputs

print("Input data set :",x.shape)
print("Input training data set :",x_train.shape)

#x_train.head()

print("Input test data set :",x_test.shape)

#x_test.head()

            Input data set : (150, 4)

            Input training data set : (105, 4)

            Input test data set : (45, 4)


# Train kNN model for 'k = '.

#find k value

import math

n=x.shape[0]

print("Total no.of records: ",n)

k=round(math.sqrt(n))

print("Value of k: ",k)


#train knn model with k=12

knn=neighbors.KNeighborsClassifier(n_neighbors=k)

knn.fit(x_train,y_train)

y_predict=knn.predict(x_test)

print(y_predict)
```



```
Total no.of records:   150
Value of k:  12
['Iris-setosa'   'Iris-versicolor'   'Iris-versicolor'   'Iris-setosa'
 'Iris-virginica'  'Iris-versicolor'  'Iris-virginica'  'Iris-setosa'
 'Iris-setosa'   'Iris-virginica'   'Iris-versicolor'   'Iris-setosa'
 'Iris-virginica'  'Iris-versicolor'  'Iris-versicolor'  'Iris-setosa'
 'Iris-versicolor'   'Iris-versicolor'   'Iris-setosa'   'Iris-setosa'
 'Iris-versicolor'  'Iris-versicolor'  'Iris-virginica'  'Iris-setosa'
 'Iris-virginica' 'Iris-versicolor' 'Iris-setosa' 'Iris-setosa'
 'Iris-versicolor'  'Iris-virginica'  'Iris-versicolor'  'Iris-virginica'
 'Iris-versicolor'   'Iris-virginica'   'Iris-virginica'   'Iris-setosa'
 'Iris-versicolor'   'Iris-setosa'   'Iris-versicolor'   'Iris-virginica'
 'Iris-virginica' 'Iris-setosa' 'Iris-virginica' 'Iris-virginica' 'Iris-
 versicolor']
```

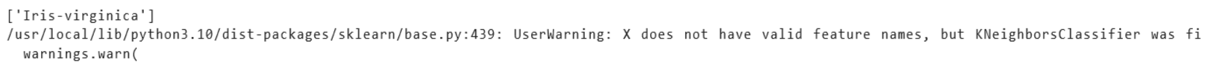
```
# Call the 'score()' function to check the accuracy score of the train set and test set -Accuracy, confusion matrix, classification report.

print("Accuracy :",accuracy_score(y_test,y_predict))

print("Confusion matrix:")

print(confusion_matrix(y_test,y_predict))

print("Classification report:")

print(classification_report(y_test,y_predict))
```

```
        Accuracy : 0.9777777777777777

        Confusion matrix:

                [[14 0 0]

                [ 0 17 1]

                [ 0 0 13]]
```



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

 Iris-setosa           1.00      1.00      1.00        14
 Iris-versicolor       1.00      0.94      0.97        18
 Iris-virginica        0.93      1.00      0.96        13

    accuracy                               0.98        45
    macro     avg      0.98      0.98      0.98        45
    weighted avg       0.98      0.98      0.98        45
```

```
#predict the Species of an unkown input

unknown=[[4,2,4.8,5.5]] ypred=knn.predict(unknown)

print(ypred)
```



```
['Iris-virginica']
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but KNeighborsClassifier was fi
  warnings.warn(
```

Program No. 6

Aim: Program to implement Naïve Bayes Algorithm using any standard dataset available in the public domain and the accuracy of the algorithm

Algorithm:

Step 1: Separate By Class.

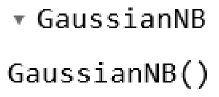
Step 2: Summarize Dataset.

Step 3: Summarize Data By Class.

Step 4: Gaussian Probability Density Function.

Step 5: Class Probabilities.

```
#Import Modules
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix

#Load iris dataset & do train_test_split
X,Y=load_iris(return_X_y=True)
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.4,random_state=2)

#Implement Naive Bayes
gnb=GaussianNB()
gnb.fit(X_train,Y_train)
```



```
▾ GaussianNB
GaussianNB()
```

```
from sklearn.metrics import classification_report
#Predict the values for test data
```

y_pred=gnb.predict(X_test)

print(y_pred)

# Display accuracy score & display confusion matrix & classification report cm=confusion_matrix(Y_test,y_pred)

print("Confusion Matrix \n",cm) print("\n")

print("\nAccuracy score:\n",accuracy_score(Y_test,y_pred))

print("\nClassification report:\n",classification_report(Y_test,y_pred))

#Print the score: the mean accuracy of the method used.

print("Gaussian Naive Bayes score:",gnb.score(X_test,Y_test))

```
 [0 0 2 0 0 2 0 2 2 0 0 0 0 0 1 1 0 1 2 1 2 1 2 1 1 0 0 2 0 2 2 0 1 2 1 0 2
  1 1 2 1 1 2 1 0 1 0 1 0 0 0 1 2 2 0 2 2 2 1 0]
 Confusion  Matrix
  [[23  0  0]
  [ 0 15  1] [
  0  3 18]]



 Accuracy      score:
  0.9333333333333333

 Classification report:
                precision    recall  f1-score   support

           0       1.00      1.00      1.00        23
           1       0.83      0.94      0.88        16
           2       0.95      0.86      0.90        21

accuracy                           0.93        60
macro     avg       0.93      0.93      0.93        60
weighted avg       0.94      0.93      0.93        60


 Gaussian Naive Bayes score: 0.9333333333333333
```

#Predict the values for unknown data [5,5,4,4]

#write code here

x_new=[[5,5,4,4]]

y_new=gnb.predict(x_new)

```
match y_new:
case 0:
        print("Class : Iris setosa")
case 1:
        print("Class : Iris Virginica")
case 2:
        print("Class : Iris Versicolor")
case _:
        print("Invalid input!!")
```

```
        Class : Iris Versicolor
```

Program No. 7

Aim: Program to implement linear regression techniques using any standard dataset (Insurance Dataset) available in the public domain and evaluate its performance.

Algorithm:

1. Collect a dataset with paired observations of the independent variable (X) and the dependent variable (Y).

2. Assume a linear relationship between X and Y in the form Y=mX+b, where m is the slope (coe cient) and b is the y-intercept.

3. Use the method of least squares to nd the values of m and b that minimize the sum of squared differences between the observed Y and the predicted Y.

4. Fit the model by adjusting the parameters m and b based on the training dataset.

5. Prediction:

6. Given a new value of X, predict the corresponding Y using the learned parameters m and b:

   Ypredicted=mXnew +b.

7. Assess the model's performance using metrics such as Mean Squared Error (MSE) or R-squared, comparing predicted values to actual values.

Activity 1: Analysing the Dataset

```
# Import modules import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as snb

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error


# Load the dataset

df=pd.read_csv('/content/insurance_data.csv')

# Print first five rows using head() function

df.head()
```

	age	sex	bmi	children	region	charges
0	18	male	33.770	1	southeast	1725.55230
1	28	male	33.000	3	southeast	4449.46200
2	33	male	22.705	0	northwest	21984.47061
3	32	male	28.880	0	northwest	3866.85520
4	31	female	25.740	0	southeast	3756.62160

# Check if there are any null values. If any column has null values, treat them accordingly
df.isnull().sum()

```
age         0
sex         0
bmi         0
children    0
region      0
charges     0
dtype: int64
```

df.shape

(1064, 6)

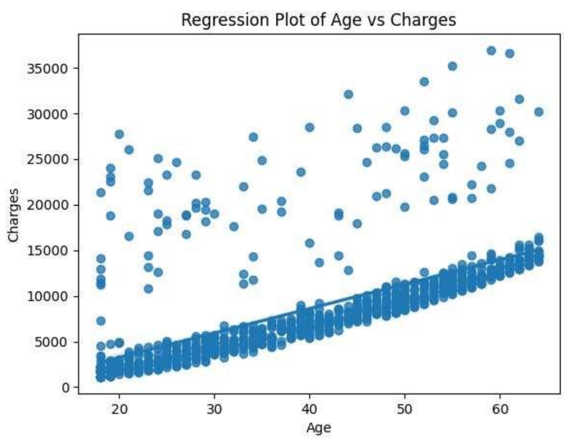
#Create a regression plot between 'age' and 'charges' using regplot in seaborn
snb.regplot(x=df['age'], y=df['charges'], data=df)

plt.title('Regression Plot of Age vs Charges')

plt.xlabel('Age')

plt.ylabel('Charges')

plt.show()

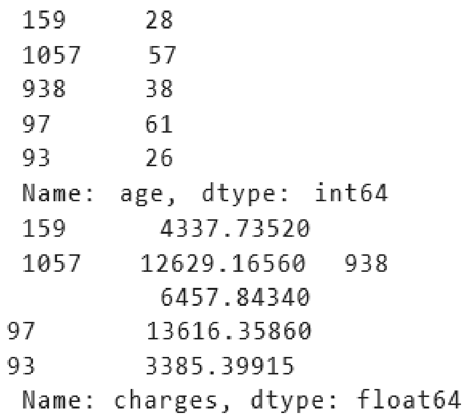


Activity 2: Train-Test Split

We have to determine the effect of age on insurance charges. Thus, age is the feature variable and charges is the target variable.

Split the dataset into training set and test set such that the training set contains 67% of the instances and the remaining instances will become the test set.

```
# Split the DataFrame into the training and test sets. #separate features and target

x=df.iloc[:,0] y=df.iloc[:,-1]

# Split the DataFrame into the training and test sets.

X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.33, random_state=3)
print(X_train.head())

y_train.head()
```



```
159     28
1057     57
938     38
97      61
93      26
Name:  age,  dtype:  int64
159      4337.73520
1057    12629.16560   938
         6457.84340
97        13616.35860
93        3385.39915
Name: charges, dtype: float64
```

```
# 1. Create two-dimensional NumPy arrays for the feature and target variables.

X_train_array = X_train.values.reshape(-1, 1) # Reshape to a 2D array (assuming X_train is a single feature)

y_train_array = y_train.values.reshape(-1, 1)

X_test_array = X_test.values.reshape(-1, 1)

y_test_array = y_test.values.reshape(-1, 1)

# Print the shape or dimensions of these arrays

print("Shape of X_train_array:", X_train_array.shape)

print("Shape of y_train_array:", y_train_array.shape)
```

```
        Shape of X_train_array: (712, 1)
```

```
Shape of y_train_array: (712, 1)
```

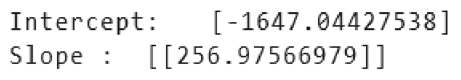
Activity 3: Model Training

Implement simple linear regression using sklearn module in the following way:

1.Deploy the model by importing the LinearRegression class and create an object of this class.

2. Call the fit() function on the LinearRegression object and print the slope and intercept values of the best t line

```
# 2. Deploy linear regression model using the 'sklearn.linear_model' module.

# Create an object of the 'LinearRegression' class.

# 3. Call the 'fit()' function


# Print the slope and intercept values

lr = LinearRegression()

lr.fit(X_train_array, y_train_array)

print("Intercept: ",lr.intercept_)

print("Slope : ", lr.coef_)
```



```
Intercept:   [-1647.04427538]
Slope :  [[256.97566979]]
```

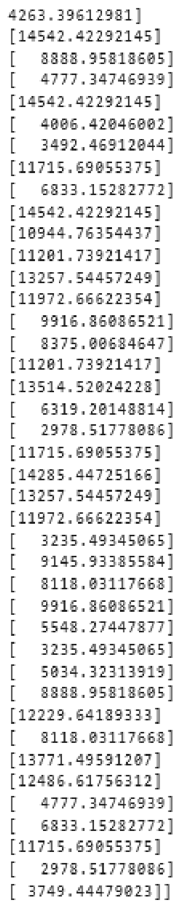
Activity 4: Model Prediction

Predict the values for both training and test sets by calling the predict() function on the LinearRegression object.

```
# Predict the target variable values for both training set and test set

#Testing set

ypred_test = lr.predict(X_test_array)

#Training set

ypred_train = lr.predict(X_train_array)
```

print("Predictions by model post build with training data: ",ypred_train)

print("Predictions by model post build with testing data: ",ypred_test)



```
 4263.39612981]
[14542.42292145]
[  8888.95818605]
[  4777.34746939]
[14542.42292145]
[  4006.42046002]
[  3492.46912044]
[11715.69055375]
[  6833.15282772]
[14542.42292145]
[10944.76354437]
[11201.73921417]
[13257.54457249]
[11972.66622354]
[  9916.86086521]
[  8375.00684647]
[11201.73921417]
[13514.52024228]
[  6319.20148814]
[  2978.51778086]
[11715.69055375]
[14285.44725166]
[13257.54457249]
[11972.66622354]
[  3235.49345065]
[  9145.93385584]
[  8118.03117668]
[  9916.86086521]
[  5548.27447877]
[  3235.49345065]
[  5034.32313919]
[  8888.95818605]
[12229.64189333]
[  8118.03117668]
[13771.49591207]
[12486.61756312]
[  4777.34746939]
[  6833.15282772]
[11715.69055375]
[  2978.51778086]
[ 3749.44479023]]
```

Activity 5: Model Evaluation

Calculate the R2, MSE, RMSE and MAE values to evaluate the accuracy of your model

#print test Performance measure

#Evaluate the model

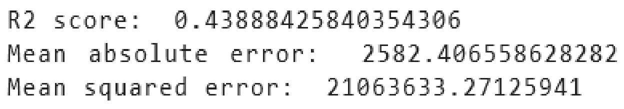
mae = mean_absolute_error(y_test_array, ypred_test)

mse = mean_squared_error(y_test_array, ypred_test)

r2 = r2_score(y_test_array, ypred_test)

print("R2 score: ",r2) print("Mean absolute error: ",mae)

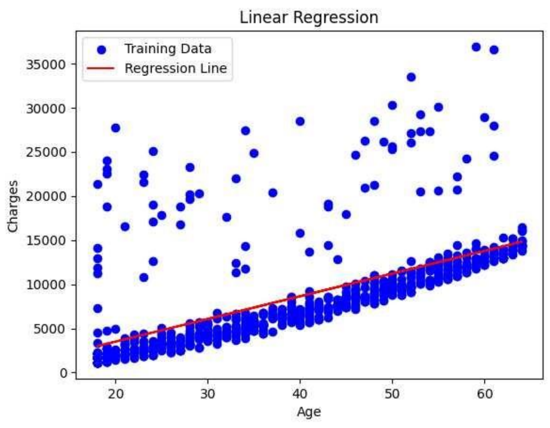
print("Mean squared error: ",mse)



```
R2 score:  0.43888425840354306
Mean absolute error:   2582.406558628282
Mean squared error:  21063633.27125941
```

#Plot the regression line

```
# Plotting the scatter plot of the training data
plt.scatter(X_train_array, y_train_array, color='blue', label='Training Data')


# Plotting the regression line
plt.plot(X_train_array, ypred_train, color='red', label='Regression Line')


# Labeling axes and title
plt.title('Linear Regression')
plt.xlabel('Age') plt.ylabel('Charges')
plt.legend()


# Show plot
plt.show()
```



Program No. 8

Aim : Program to implement multiple regression techniques using Advertising dataset available in the public domain and evaluate its performance (Forecast Sales).

Algorithm :

1. Multiple Linear Regression extends Simple Linear Regression to model the relationship between a dependent variable Y and multiple independent variables X1,X2,...,Xn . Here's a concise algorithm for Multiple Linear Regression:

2. Collect a dataset with paired observations of the dependent variable (Y) and multiple independent variables (X1,X2,...,Xn).

3. Assume a linear relationship between Y and the independent variables: Y=b0+b1X1+b2X2+...+bn Xn, where b0 is the intercept and b1,b2 ,...,bn are the coe cients.

4. Use Ordinary Least Squares (OLS) method to nd the values of b0,b1,b2,...,bn that minimize the sum of squared differences between the observed Y and the predicted Y (model's predictions).

5. Fit the model by adjusting the parameters b0,b1,b2,...,bn based on the training dataset.

6. Given a new set of values for X1,X2,...,Xn , predict the corresponding Y using the learned parameters:

   newY predicted =vb0+b1X1 new +b2X2 new +...+bn Xn new .

7. Assess the model's performance using metrics such as Mean Squared Error (MSE) or R-squared, comparing predicted values to actual values.

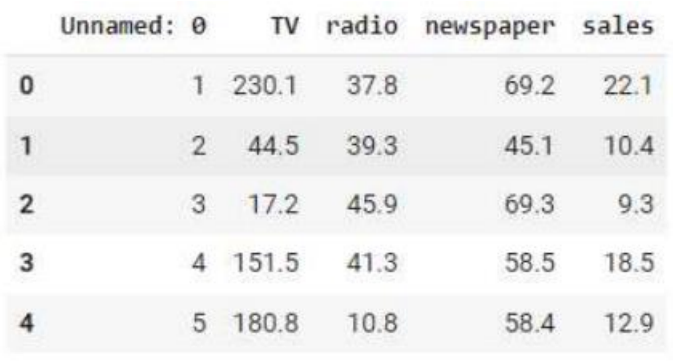
Activity 1: Analysing the Dataset

Create a Pandas DataFrame for Advertising-Sales dataset using the below link. This dataset contains information about the money spent on the TV, radio and newspaper advertisement (in thousand dollars) and their generated sales (in thousand units). The dataset consists of examples that are divided by 1000.

DatasetLink:https://raw.githubusercontent.com/smithaks/Notebooks/Datasets/Advertising.sv

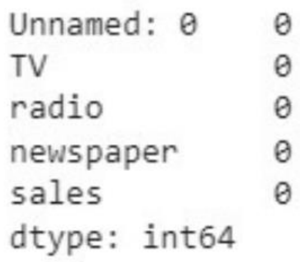
Also, print the first five rows of the dataset. Check for null values and treat them accordingly.

```
# Import modules

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt
import seaborn as snb
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
# Load the dataset
df=pd.read_csv("https://raw.githubusercontent.com/smithaks/Notebooks/Datasets/Advertisin
g.csv")
# Print first five rows using head() function
df.head()
```



	Unnamed: 0	TV	radio	newspaper	sales
0	1	230.1	37.8	69.2	22.1
1	2	44.5	39.3	45.1	10.4
2	3	17.2	45.9	69.3	9.3
3	4	151.5	41.3	58.5	18.5
4	5	180.8	10.8	58.4	12.9

```
# Check if there are any null values. If any column has null value
print(df.isnull().sum())
```



```
Unnamed: 0    0
TV            0
radio         0
newspaper     0
sales         0
dtype: int64
```

Activity 2: Train-Test Split

For Multiple linear regression, consider the effect of TV,Radio and Newspaper ads on sales. Thus, TV,Radio and Newspaper are the feature variable and Sales is the target variable.

Split the dataset into training set and test set such that the training set contains 70% of the instances and the remaining instances will become the test set.# Split the DataFrame into the training and test sets.

#separate features and target

x=df.iloc[:,1:4]

y=df.iloc[:,-1]

# Split the DataFrame into the training and test sets.

X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=3) print(X_train.head())

y_train.head()

```
           TV radio   newspaper
      77   120.528.5         14.2
      73   129.45.7          31.3
      71   109.814.3         31.7
      78     5.4 29.9        9.4

      42  293.6   27.7         1.8
      77    14.2
      73    11.0
      71    12.4
      78     5.3
      42    20.7
      Name: sales, dtype: float64
```

#Build the Model

lr = LinearRegression()

lr.fit(X_train, y_train)

y_pred = lr.predict(X_test)

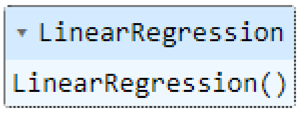
#Print the slope and intercept values

print("Intercept: ",lr.intercept_) print("Slope : ", lr.coef_)

print("Predictions by model post build with training data: ",y_pred)

```
Intercept:  3.5238550353772666
Slope :  [ 0.04294217  0.1879147  -0.00541318]
Predictions by model post build with training data:  [16.23908522  9.61974227 19.70411458 12.83758096  7.75117142 10.39614031
 23.56568814  9.04287249 17.61434029 13.61264994 12.41895295 14.63059421
15.36549523 12.91196831 12.44967293 12.02022165 16.18805828 17.47379948
17.19147955 21.69255609 18.22173873  8.90050617 10.7849561   12.02942207
6.84754722 13.66447529 22.17451249 13.50945016 22.52702726 11.88138574
17.03988139  21.52399711  10.71565512 7.85644348 10.20198234   8.44594245
13.001148   10.77141583 12.17524762  9.8569604  15.45704842 13.04055175

  5.84887331 20.59192848 22.47250008 24.37729901 14.37760834 10.94824196
 16.36148492 18.13598497 11.43279278 14.72792516 16.94299022   8.98448136
 19.51641741 10.89477891 22.74714758 21.18575245 15.70986415 14.876838  ]
```

```
lr=LinearRegression()

lr.fit(x_train,y_train)
```



```
LinearRegression
LinearRegression()
```

```
#print test Performance measure

train_pred=lr.predict(x_train)

test_pred=lr.predict(x_test)

print("Training performance\n")

print("R.squared:\t",r2_score(y_train,train_pred))

print("Mean squared error:\t",mean_squared_error(y_train,train_pred))

print("Root mean squared error:\t",np.sqrt(mean_squared_error(y_train,train_pred)))

print("Mean absolute erroe:\t",mean_absolute_error(y_train,train_pred))
```

```
Training performance

R.squared:       0.8850053786777522

Mean squared error:     3.2031174449376514

Root mean squared error:        1.7897255222345272

Mean absolute erroe:    1.3746543626017602
```

```
# Calculate the slope and intercept values for the best fit line.

print(lr.intercept_)

print(lr.coef_)
```

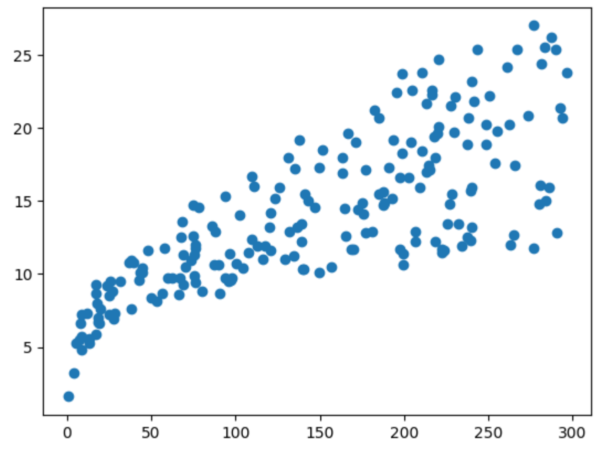
```
2.937215734690609

[0.04695205 0.17658644 0.00185115]
```

Q: What is the equation obtained for the best fit line of this model?

A:y=0.04695205x1+0.17658644x2+0.00185115x3+2.937215734690609

```
#  Plot the regression line in the scatter plot between Sales and TV advertisment values

plt.scatter(df["TV"],df["sales"])
```

```
<matplotlib.collections.PathCollection at 0x7c7b1afa9ba0>
```



Q: If you are planning to invest $50,000 ,dollars in TV , 1000 dollars in newspaper and 50000 dollars in Radio advertising, how many unit of sales can be predicted according to this multiple linear regression model?

```
# Calculating sales value against ads

#y=0.04695205x1+0.17658644x2+0.00185115x3+2.937215734690609

x1=50000

x2=1000

x3=50000

y=0.04695205*50000+0.17658644*1000+0.00185115*50000+2.937215734690609

print(y)

        2619.6836557346905
```

Program No. 9

Aim: Program to implement classi cation using Support vector machine use Iris dataset.

Algorithm:

1. Data Preparation: Collect a labeled dataset with input features and corresponding class labels.

2. Model Initialization: Assume a hyperplane that best separates the data into different classes. The hyperplane is de ned by a set of weights (w) and a bias term (b).

3. Feature Scaling (Optional): Standardize or normalize the input features to ensure they have similar scales. This step can enhance the performance of SVM.

4. Kernel Selection (Optional): Choose a kernel function (linear, polynomial, radial basis function, etc.) based on the nature of the data. The kernel function determines the transformation of input features.

5. Parameter Selection: Choose parameters such as the regularization parameter (C) and kernel parameters. These parameters in uence the exibility of the decision boundary.

6. Training: Optimize the hyperplane to maximize the margin between different classes while minimizing classi cation errors.

7. Support Vectors: Identify the support vectors, which are the data points that lie closest to the decision boundary.

8. Decision Function: The decision function is de ned as $f(x)=w \cdot x+b$, where x is the input feature vector. The sign of $f(x)$ determines the predicted class.

9. Prediction: Given a new data point, use the trained model to predict its class based on the decision function.

10. Evaluation: Assess the performance of the SVM using metrics such as accuracy, precision, recall, or F1 score

```
#Import the necessary libraries from sklearn

import datasets from sklearn.model_selection

import train_test_split from sklearn.svm

import SVC from sklearn.metrics

import accuracy_score from sklearn.metrics

import confusion_matrix

#Load a dataset(Iris dataset is used in this example.)

iris_df=datasets.load_iris()
```

X=iris_df.data[:, :2]

#Take the 1st 2 features.

y=iris_df.target

#Split the data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)

#Create an SVM Classifier (we can choose different kernel)

svm_classifier= SVC(kernel='linear', C=1.0, gamma=0.5)

#Train the SVM classifier on the training data

svm_classifier.fit(X_train,y_train)

```
    ▾              SVC
    SVC(gamma=0.5, kernel='linear')
```

#Make predictions on the test data

y_pred=svm_classifier.predict(X_test)

print(y_pred)

```
[0 1 2 0 2 2 2 0 0 2 1 0 2 2 1 0 1 1 0 0 1 1 2 0 2 1 0 0 1 1 2 2 1 2 1 0 1
 0 2 2 2 0 1 2 2]
```

#Calculate accuracy

accuracy=accuracy_score(y_test,y_pred)

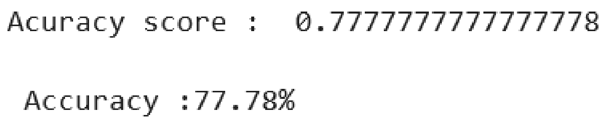
print("Acuracy score : ", accuracy)

print(f'\n Accuracy :{accuracy*100:.2f}%')

```
Acuracy score :  0.7777777777777778

 Accuracy :77.78%
```

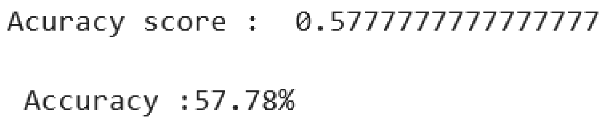
#Kernel = 'rbf' with gamma value 0.7, then 100 and 500

```
#Create an SVM Classifier (we can choose different kernel, here we choose rbf with gamma 0.7)
svm_classifier1= SVC(kernel='rbf', C=1.0, gamma=0.7)
#Train the SVM classifier on the training data
svm_classifier1.fit(X_train,y_train)
#Make predictions on the test data
ypred1=svm_classifier1.predict(X_test)
#Calculate accuracy
accuracy=accuracy_score(y_test,ypred1)
print("Acuracy score : ", accuracy)
print(f'\n Accuracy :{accuracy*100:.2f}%')
```



```
Acuracy score :  0.7777777777777778

 Accuracy :77.78%
```

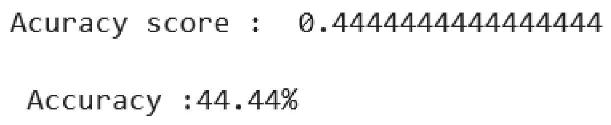
```
#Create an SVM Classifier (we can choose different kernel, here we choose rbf with gamma 100)
svm_classifier2= SVC(kernel='rbf', C=1.0, gamma=100)
#Train the SVM classifier on the training data
svm_classifier2.fit(X_train,y_train)
#Make predictions on the test data
ypred2=svm_classifier2.predict(X_test)
#Calculate accuracy
accuracy=accuracy_score(y_test,ypred2)
print("Acuracy score : ", accuracy)
print(f'\n Accuracy :{accuracy*100:.2f}%')
```



```
Acuracy score :  0.5777777777777777

 Accuracy :57.78%
```

```
#Create an SVM Classifier (we can choose different kernel, here we choose rbf with gamma 500)
```

```
svm_classifier3= SVC(kernel='rbf', C=1.0, gamma=500)
#Train the SVM classifier on the training data
svm_classifier3.fit(X_train,y_train)
#Make predictions on the test data
ypred3=svm_classifier3.predict(X_test)
#Calculate accuracy
accuracy=accuracy_score(y_test,ypred3)
print("Acuracy score : ", accuracy)
print(f'\n Accuracy :{accuracy*100:.2f}%')
```



```
Acuracy score :  0.4444444444444444

 Accuracy :44.44%
```

Program No. 10

Aim : Program to implement decision trees using Iris dataset available in the public domain and find the accuracy of the algorithm.

Algorithm

1. Data Preparation: Collect a labeled dataset with input features and corresponding class labels for classification.

2. Tree Initialization: Start with the root node containing the entire dataset.

3. Feature Selection: Choose the best feature to split the data. This is based on a criterion such as Gini impurity.

4. Splitting: Divide the dataset into subsets based on the chosen feature. Each subset corresponds to a branch from the current node.

5. Recursive Splitting: Repeat the splitting process for each subset, creating child nodes. Continue this recursive process until a stopping criterion is met, such as reaching a maximum depth or a minimum number of samples per node.

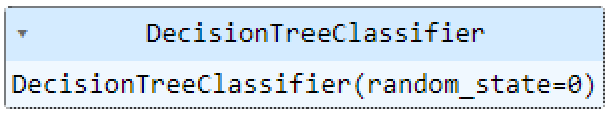
6. Leaf Node Assignment: Assign a class label (for classification) or a predicted value (for regression) to each leaf node. This is typically the majority class for classification or the mean value for regression.

7. Tree Pruning (Optional): To avoid overfitting, prune the tree by removing branches that do not contribute significantly to improving predictive accuracy.

8. Prediction: Given a new data point, traverse the decision tree from the root to a leaf node, and assign the corresponding class label or predicted value.

9. Visualization (Optional): For interpretability, visualize the decision tree structure to understand how decisions are made at each node.

```
# Import the necessary libraries

from sklearn import datasets

from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy_score

from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

# Load a dataset -Iris dataset

iris=datasets.load_iris()

x=iris.data

y=iris.target

# Split the dataset into a training set and a testing set(trainig 70% records)

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=2)

# Create a Decision Tree Classifier

DT_clf=DecisionTreeClassifier(random_state=0)

# Train the classifier on the training data

DT_clf.fit(x_train,y_train)
```

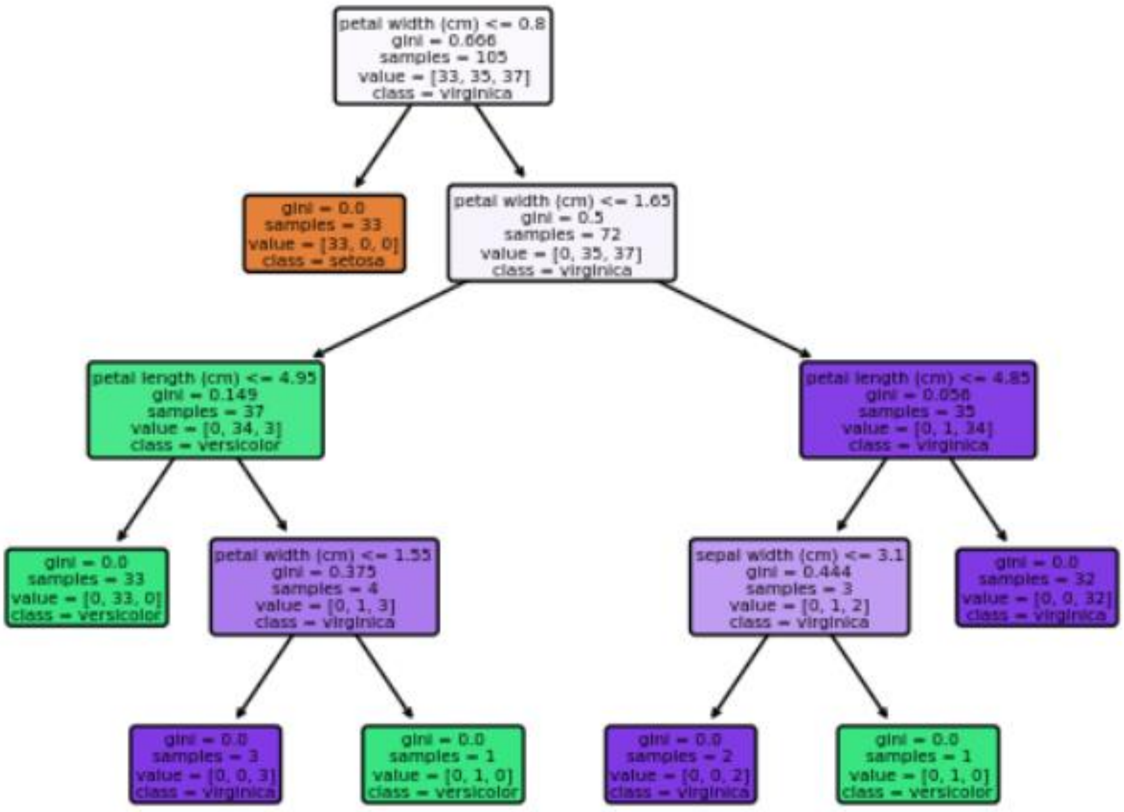


```
DecisionTreeClassifier
DecisionTreeClassifier(random_state=0)
```

```
# Make predictions on the test data

y_pred=DT_clf.predict(x_test)

# Calculate accuracy and confusion matrix

accuracy=accuracy_score(y_test,y_pred)

print("Accuracy:",accuracy)

cm=confusion_matrix(y_test,y_pred)

print("Confusion matric\n",cm)

classification_report(y_test,y_pred)

Accuracy: 0.9555555555555556

        Confusion matric

         [[17  0  0]
```

```
 [ 0 14  1]

 [ 0  1 12]]
```

```
import matplotlib.pyplot as plt

from sklearn.tree import plot_tree

plot_tree(DT_clf, filled=True, feature_names=iris.feature_names,
class_names=iris.target_names, rounded=True)

plt.show()
```



Program No. 11

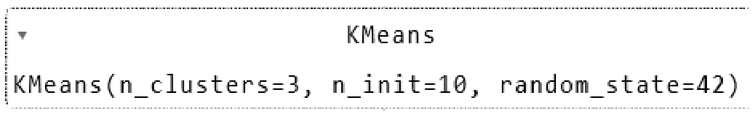
Aim : Program to implement k-means clustering technique using Iris dataset available in the public domain.

Algorithm:

1. K-Means clustering is a popular unsupervised machine learning algorithm used for partitioning a dataset into K distinct, non-overlapping subsets (clusters).
2. Initialization: Choose the number of clusters K that you want to identify in the dataset.
3. Randomly initialize the centroids of the K clusters by selecting K data points from the dataset.
4. Assignment: Assign each data point to the nearest centroid, forming K clusters.
5. Update Centroids: Recalculate the centroids of the K clusters by taking the mean of all data points assigned to each cluster.
6. Repeat: Repeat the assignment and centroid update steps until convergence. Convergence occurs when the centroids no longer change signi cantly or after a prede ned number of iterations.
7. Result: The nal centroids represent the centers of the K clusters, and each data point is assigned to the cluster with the nearest centroid.
8. Evaluation (Optional): Assess the quality of the clustering using metrics such as the sum of squared distances between data points and their assigned centroids(inertia).
9. Visualization (Optional): Visualize the clusters by plotting the data points and centroids

```
#Import required libraries
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from sklearn import metrics
# Load the Iris dataset iris = load_iris()
X = iris.data # Features


# Implementing K-means with 3 clusters and explicitly setting n_init
```

```
kmeans = KMeans(n_clusters=3, n_init=10, random_state=42)
kmeans.fit(X)
```



```
KMeans
KMeans(n_clusters=3, n_init=10, random_state=42)
```

```
# Getting the inertia value (lower is better)
inertia = kmeans.inertia_
print("Inertia:", inertia)
        Inertia: 78.851441426146
# Getting the cluster centers and labels
cluster_centers = kmeans.cluster_centers_
labels = kmeans.labels_
# Silhouette score (higher is better)
silhouette_score = metrics.silhouette_score(X, kmeans.labels_)
print("Silhouette Score:", silhouette_score)
        Silhouette Score: 0.5528190123564095

# Implementing K-means with 3 clusters and explicitly setting n_init
kmeans = KMeans(n_clusters=3, n_init=10, random_state=42)
kmeans.fit(X)
# New, unknown data points (replace these values with your own data)
new_data = [
        [5.1, 3.5, 1.4, 0.2], # Example 1
        [6.2, 2.8, 4.8, 1.8], # Example 2
        [7.3, 3.1, 6.3, 2.1]] # Example 3

# Predicting the clusters for the new data
new_predictions = kmeans.predict(new_data)
print("Predicted clusters for the new data points:", new_predictions)
        Predicted clusters for the new data points: [1 0 2]
```

# Visualizing the clusters (Considering only the first two features for simplicity)
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', marker='o', edgecolor='black')

plt.scatter(cluster_centers[:, 0], cluster_centers[:, 1], marker='X', s=200, c='red', label='Centroids')

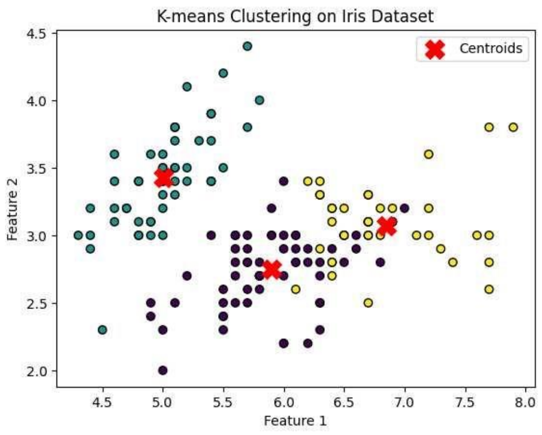
plt.title('K-means Clustering on Iris Dataset')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.legend()

plt.show()



from sklearn.metrics import accuracy_score

y_true = iris.target # True labels (only for demonstration purposes)

# Get the predicted labels

d k l b l

y_pred = kmeans.labels_

# Accuracy score (only for demonstration with Iris dataset where true labels are available)
accuracy = accuracy_score(y_true, y_pred)

print("Accuracy score (only for demonstration with Iris dataset):", accuracy)

```
Accuracy score (only for demonstration with Iris dataset): 0.24
```

Program No. 12

Aim: Program on convolutional neural network to classify images using MNIST dataset in the public domain using Keras framework

Algorithm:

1. Data Preparation: Collect a labelled dataset of images with corresponding class labels for training and, optionally, separate datasets for validation and testing.

2. Network Architecture: Design the CNN architecture, which typically includes convolutional layers for feature extraction, pooling layers for down-sampling, fully connected layers for classi cation, and activation functions like ReLU.

3. Convolutional Layers: Apply convolutional operations to the input images using lters or kernels. This helps extract relevant features from the images.

4. Activation Function: Introduce non-linearity to the model using activation functions like ReLU (Recti ed Linear Unit) after convolutional and fully connected layers.

5. Pooling Layers: Use pooling layers (e.g., max pooling) to reduce spatial dimensions, focusing on the most essential features and enhancing computational e ciency.

6. Flattening: Flatten the output from the last convolutional or pooling layer into a 1D vector to feed into the fully connected layers.

7. Fully Connected Layers: Add one or more fully connected layers to perform classi cation based on the extracted features. The last layer typically has softmax activation for multi-class classi cation.

8. Loss Function: Choose a suitable loss function, such as categorical cross-entropy,to measure the difference between predicted and actual class probabilities.

9. Optimizer: Select an optimizer (e.g., Adam, SGD) to minimize the loss function during training.

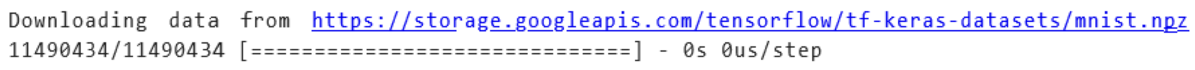
10. Training: Train the CNN on the training dataset using backpropagation and optimization. Adjust the network weights to minimize the loss. 11. Validation: Validate the trained model on a separate dataset to ensure it generalizes well to unseen data and adjust hyperparameters if needed.

12. Testing: Evaluate the nal model on the test dataset to assess its performance on completely unseen data.

13. Fine-Tuning (Optional): Fine-tune the model by adjusting hyperparameters, modifying the architecture, or incorporating techniques like regularization to improve performance.

14. Prediction: Use the trained CNN to make predictions on new images for classi cation.

```
#Import necessary libraries

from keras.datasets import mnist

from keras.models import Sequential

from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

from keras.utils import to_categorical

# Load the MIST dataset

(x_train, y_train), (x_test, y_test)=mnist.load_data()
```



```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
```

```
y_train

        array([5, 0, 4, ..., 5, 6, 8], dtype=uint8)

x_train.shape

        (60000, 28, 28)

x_test.shape

        (10000, 28, 28)

#Preprocess the data

x_train=x_train.reshape((x_train.shape[0], 28, 28, 1))

x_test=x_test.reshape((x_test.shape[0], 28, 28, 1 ))

x_train=x_train/255.0

x_test=x_test/255.0

x_train.shape

        (60000, 28, 28, 1)

x_test.shape

        (10000, 28, 28, 1)

#Convert to labels in the training set into one-hot encoding vectors
y_train=to_categorical(y_train)

y_test=to_categorical(y_test)

y_train
```

```
array([[0., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 1., 0.]], dtype=float32)
```

#Create a sequential model . ie. layer by layer

model= Sequential()

#Add a convolutional layer with 32 filters, a 3*3 kernel and 'relu' activation

#add() adds each layer to the n/w

model.add(Conv2D(32, (3, 3),activation='relu', input_shape=(28, 28, 1)))

#Add a max pooling layer with 2*2 pool size

model.add(MaxPooling2D(pool_size=(2, 2)))

#Flatten the ouput before feeding it into densely conected layers

model.add(Flatten())

#Add a dense layer with 128 units and 'relu' activation

model.add(Dense(128, activation='relu'))

#Add the output layer with 10 units (for 10 classes) and 'softmax' activation
model.add(Dense(10,activation='softmax'))

#Compile the models
model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=["accuracy"])

#Train the model

model.fit(x_train, y_train, epochs=5, batch_size=64, validation_data=(x_test, y_test))

```
Epoch 1/5
938/938 [==============================] - 20s 21ms/step - loss: 0.1730 - accuracy: 0.9494 - val_loss: 0.0661 - val_accuracy: 0.9795
Epoch 2/5
938/938 [==============================] - 18s 19ms/step - loss: 0.0551 - accuracy: 0.9834 - val_loss: 0.0539 - val_accuracy: 0.9838
Epoch 3/5
938/938 [==============================] - 19s 20ms/step - loss: 0.0375 - accuracy: 0.9885 - val_loss: 0.0421 - val_accuracy: 0.9862
Epoch 4/5
938/938 [==============================] - 19s 20ms/step - loss: 0.0264 - accuracy: 0.9919 - val_loss: 0.0398 - val_accuracy: 0.9870
Epoch 5/5
938/938 [==============================] - 18s 19ms/step - loss: 0.0181 - accuracy: 0.9941 - val_loss: 0.0433 - val_accuracy: 0.9851
<keras.src.callbacks.History at 0x7fd3f7f43760>
```

#Evaluate the model on the test set

loss, accuracy = model.evaluate(x_test, y_test) print(f'\nTest loss:{loss:.4f},Test Accuracy:{accuracy:.4f}')

```
313/313 [==============================] - 2s 7ms/step - loss: 0.0433 - accuracy: 0.9851

Test loss:0.0433,Test Accuracy:0.9851
```

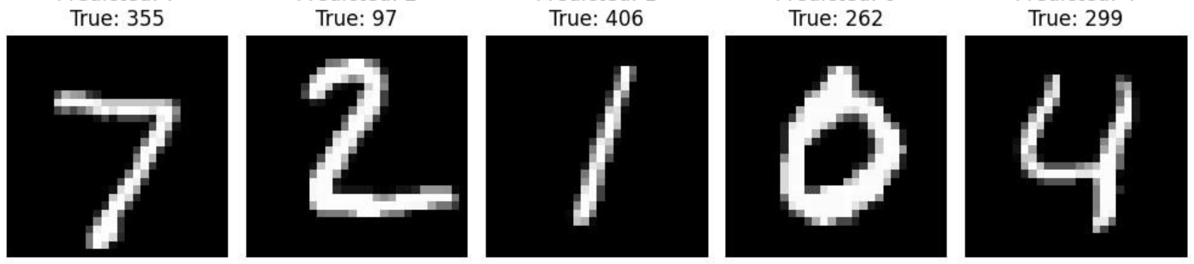
```
# Make predictions on test data
predictions = model.predict(x_test)
        313/313 [==============================] - 1s 4ms/step
print(predictions)
```

```
[[2.97589775e-09 1.84179267e-08 3.41827729e-07 ... 9.99998987e-01
  1.97926608e-09 3.97527430e-07]
 [7.67650761e-07 3.63180629e-06 9.99994576e-01 ... 4.17919838e-10
  1.37880685e-08 2.03839542e-10]
 [2.02734839e-07 9.99977648e-01 1.65136782e-07 ... 1.04900582e-05
  1.09738721e-06 3.46519080e-08]
 ...
 [1.51214517e-13 1.72952416e-10 7.81245121e-12 ... 1.24356561e-08
  1.59861369e-08 8.12365002e-08]
 [9.20821863e-10 6.06356718e-12 5.02318076e-13 ... 8.69199879e-10
  7.57513277e-04 1.07631832e-08]
 [4.90774699e-10 8.77211715e-13 3.17746274e-09 ... 2.09131635e-13
  5.06760589e-09 4.11113591e-12]]
```

```
print(predictions[0])
```

```
[2.9758978e-09 1.8417927e-08 3.4182773e-07 2.4179579e-07 1.4495514e-09
 3.1526354e-10 2.5520897e-13 9.9999899e-01 1.9792661e-09 3.9752743e-07]
```

```
# Plot a few test images along with their predicted labels
import matplotlib.pyplot as plt
import numpy as np
num_images_to_plot = 5
plt.figure(figsize=(10, 5))
for i in range(num_images_to_plot):
        plt.subplot(1, num_images_to_plot, i + 1)
        plt.imshow(x_test[i].reshape(28, 28), cmap='gray')
        plt.title(f"Predicted: {np.argmax(predictions[i])}\nTrue: {np.argmax(x_test[i])}")
        plt.axis('off')
plt.tight_layout()
plt.show()
```



Program No. 13

Aim: Program to implement a simple web crawler and scrapping web pages.

Algorithm:

A simple web crawler is a program that systematically navigates through web pages, extracts information, and may follow links to discover more pages.

1. Seed URL: Start with an initial URL (seed URL) that you want to crawl.

2. Initialize Queue: Create a queue to manage the URLs to be crawled. Initially,enqueue the seed URL.

3. Crawl Loop

4. Start a loop that continues until the queue is empty or a speci ed limit is reached.

5. Dequeue a URL from the queue.

6. Send an HTTP request to fetch the HTML content of the page corresponding to the dequeued URL.

7. Parse the HTML content to extract relevant information or links. Libraries like BeautifulSoup or Scrapy in Python are commonly used for HTML parsing.

8. Process the extracted information or store it for further analysis.

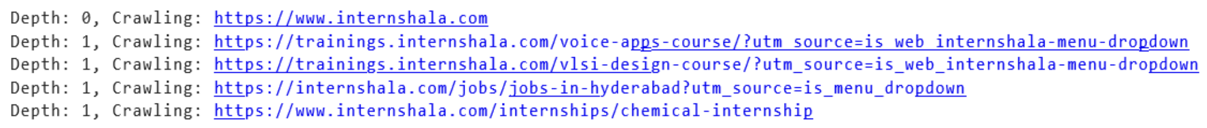
9. Enqueue any new URLs found on the page, ensuring they haven't been visited before to avoid duplicate crawling. 10. Repeat: Repeat the crawl loop until the queue is empty or a speci ed limit is reached.

11. Data Storage (Optional): Optionally, store the extracted data in a database or le for later analysis.

12. Respect Robots.txt: Follow ethical practices by respecting the rules speci ed in the "robots.txt" le on websites, which can de ne which parts of a site are off-limits for crawling.

13. Error Handling: Implement error handling to manage issues like connection errors or unexpected content during the crawling process

```
import requests

from bs4 import BeautifulSoup from urllib.parse import urljoin

def get_links(url):

    response = requests.get(url)

    soup = BeautifulSoup(response.text, 'html.parser')
    links = set()
    for anchor in soup.find_all('a'):
        href = anchor.get('href')
        if href and href.startswith('http'):
            links.add(href)
        else:
            full_url = urljoin(url, href) links.add(full_url)
    return links

def crawl(start_url, max_depth=3, max_pages=10):
    visited = set()
    queue = [(start_url, 0)]
    pages_visited = 0
    while queue and pages_visited < max_pages:
        current_url, depth = queue.pop(0)
        if current_url in visited or depth > max_depth:
            continue
        print(f"Depth: {depth}, Crawling: {current_url}")
        try:
            links = get_links(current_url)
            visited.add(current_url)
            pages_visited += 1
            queue.extend((link, depth + 1) for link in links if link not in visited)
        except Exception as e:
            print(f"Error crawling {current_url}: {e}")

if __name__ == "__main__":
    seed_url = "https://www.internshala.com"
    crawl(seed_url, max_pages=5) # Set max_pages to limit the number of pages crawled
```



```
Depth: 0, Crawling: https://www.internshala.com
Depth: 1, Crawling: https://trainings.internshala.com/voice-apps-course/?utm_source=is_web_internshala-menu-dropdown
Depth: 1, Crawling: https://trainings.internshala.com/vlsi-design-course/?utm_source=is_web_internshala-menu-dropdown
Depth: 1, Crawling: https://internshala.com/jobs/jobs-in-hyderabad?utm_source=is_menu_dropdown
Depth: 1, Crawling: https://www.internshala.com/internships/chemical-internship
```