Date :

# 1. INTRODUCTION TO NUMPY

NumPy is a Python package. It stands for 'Numerical Python'. It is a library consisting of Multidimensional array of objects and a collection of routines for processing an array.

**Numeric**, the ancestor of NumPy, was developed by Jim Hugunin. Another package Numarray was also developed, having some additional functionalities. In 2005, Travis Oliphant created NumPy package by incorporating the features of Numarray into Numeric package. There are many contributors to this open source project.

Using NumPy, a developer can perform the following operations −

- Mathematical and logical operations on arrays.
- Fourier transforms and routines for shape manipulation.
- Operations related to linear algebra. NumPy has in-built functions for linear algebra and random number generation.

**importnumpyas np**

**OUTPUT:**

1)

a) array([ True,  True,  True, False,  True,  True, False, False,  True])

b) array([ True, False, False,  True, False,  True, False, False])

c) array([False,  True,  True, False,  True, False,  True,  True])

d) array([ True,  True, False,  True, False,  True,  True, False])

e) array([False,  True,  True,  True,  True, False,  True,  True])

1) Write a numpy program to create a element vise comparison of 2 given arrays

import numpy as np

a) a=np.array([1,2,32,3,4,5,6,76,34])

b=np.array([1,2,32,4,4,5,5,65,34])

np.equal(a,b)

b) c = np.array([34,20,11,4,2,100,24,5])

d = np.array([20,55,13,1,17,90,50,8])

np.greater(c,d)

c) c = np.array([34,20,11,4,2,100,24,5])

d = np.array([20,55,13,1,17,90,50,8])

np.less(c,d)

d) c = np.array([34,20,11,4,2,100,24,5])

d = np.array([14,20,13,4,17,90,24,8])

np.greater_equal(c,d)

e) c = np.array([34,20,11,4,2,100,24,5])

d = np.array([14,20,13,4,17,90,24,8])

np.less_equal(c,d)

**OUTPUT:**

2)

```
 [30 32 34 36 38 40 42 44 46 48]
```

3)

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

4)

```
[  0   1   2   3   4   5   6   7   8  -9 -10 -11 -12 -13 -14 -15  16  17  18  19  20]
```

5)

```
 [[1 0 0 0 0]
 [0 2 0 0 0]
 [0 0 3 0 0]
 [0 0 0 4 0]
 [0 0 0 0 5]]
```

6)

```
45
[12 15 18]
[ 6 15 24]
```

7)

```
[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11.]
```

2) Write a NumPy program to create an array of all the even integers from 30 to 70.

```
a = np.arange(30,50,2)

print(a)
```

3) Write a NumPy program to create a 3x3 identity matrix.

```
x = np.identity(3)

print(x)
```

4) Write a NumPy program to create a vector with values from 0 to 20 and change the signof the numbers in the range from 9 to 15. x = np.arange(21)

```
x[ (x>=9) & (x<=15)] *=-1

print(x)
```

5) Write a NumPy program to create a 5x5 zero matrix with elements on the main diagonal equal to 1, 2, 3, 4, 5.x = np.diag([1, 2, 3, 4, 5])

```
print(x)
```

6) Write a NumPy program to compute sum of all elements, sum of each column and sum of each row of a given array.

```
x = np.array([[1,2,3],[4,5,6],[7,8,9]])

print(np.sum(x))

print(np.sum(x, axis=0))

print(np.sum(x, axis=1))
```

7)Write a NumPy program to save a given array to a text file and load it

```
x = np.arange(12)

np.savetxt('temp.txt', x, fmt="%d")

result = np.loadtxt('temp.txt')

print(result)
```

**OUTPUT:**

8)

```
array([ True,  True,  True, False,  True,  True, False, False,  True])
```

9)

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]
[[12 13 14 15]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [ 0  1  2  3]]
```

10)

```
Array1:
[[2 5 2]
 [1 5 5]]
Array2:
[[5 3 4]
 [3 2 5]]

Multiply said arrays of same size element-by-element:
[[10 15  8]
 [ 3 10 25]]
```

8) Write a NumPy program to check whether two arrays are equal (element wise) or not.

```
a=np.array([1,2,32,3,4,5,6,76,34])

b=np.array([1,2,32,4,4,5,5,65,34])

np.equal(a,b)
```

9) Write a NumPy program to create a 4x4 array with random values, now create a new array from the said array swapping first and last rows.

```
nums = np.arange(16).reshape(4,4)

print(nums)

# nums[0],nums[-1]=nums[-1],nums[0]

nums[[0,-1]]=nums[[-1,0]]

print(nums)
```

10) Write a NumPy program to multiply two given arrays of same size element-by-element.

```
nums1 = np.array([[2, 5, 2],

            [1, 5, 5]])

nums2 = np.array([[5, 3, 4],

            [3, 2, 5]])

print("Array1:")

print(nums1)

print("Array2:")

print(nums2)

print("\nMultiply said arrays of same size element-by-element:")

print(np.multiply(nums1, nums2))
```

**RESULT**: The program is executed and output is verified

**OUTPUT:**

```
enter elements of first matrix:
m1:
 [[1 2]
 [3 4]]
enter elements of second matrix:
m2:
 [[1 2]
 [3 4]]
dot product:
 [[ 7 10]
 [15 22]]
transpose:
 [[1 3]
 [2 4]]
trace:
 5
rank:
 2
determinant:
 -2.0000000000000004
inverse:
 [[-2.   1. ]
 [ 1.5 -0.5]]

E-value :
[-0.37228132  5.37228132]

E-vector :
[[-0.82456484 -0.41597356]
 [ 0.56576746 -0.90937671]]
```

Date :

# 2. MATRIX OPERATIONS AND TRANSFORMATIONS

**AIM:** Write python program to create to matrices and find the following:

- Dot product
- Transpose
- Trace
- Rank
- Determinant
- Inverse
- Eigen Values and eigen vectors

**PROGRAM:**

```
import numpy as np

r1=int(input("enter the number of rows"))

c1=int(input("enter the number of columns"))

print("enter elements of first matrix:")

m1=list(map(int,input().split()))

m1=np.array(m1).reshape(r1,c1)

print("m1:\n",m1)

r2=int(input("enter the number of rows"))

c2=int(input("enter the number of columns"))

print("enter elements of second matrix:")

m2=list(map(int,input().split()))

m2=np.array(m1).reshape(r1,c1)

print("m2:\n",m2)

print("dot product:\n",np.dot(m1,m2))

print("transpose:\n",m1.transpose())

print("trace:\n",m1.trace())

print("rank:\n",np.linalg.matrix_rank(m1))

print("determinant:\n",np.linalg.det(m1))

print("inverse:\n",np.linalg.inv(m1))

x,y = np.linalg.eig(m1)
```

```
print("\nE-value : ")
print(x)
print("\nE-vector : ")
print(y)
```

**RESULT**: The program is executed and output is verified

Date :

# 3. PROGRAMS USING MATPLOTLIB

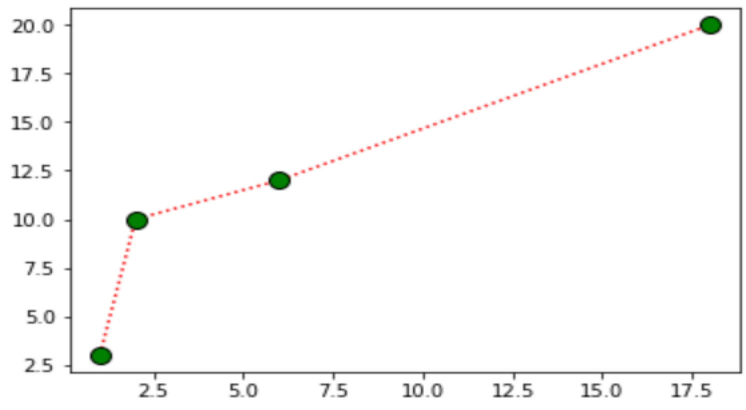
Matplotlib is a plotting library for Python. It is used along with NumPy to provide an environment that is an effective open source alternative for MatLab. It can also be used with graphics toolkits like PyQt and wxPython. Matplotlib module was first written by John D. Hunter. Since 2012, Michael Droettboom is the principal developer. Currently, Matplotlib ver. 1.5.1 is the stable version available. The package is available in binary distribution as well as in the source code form on www.matplotlib.org.

Conventionally, the package is imported into the Python script by adding the following statement –

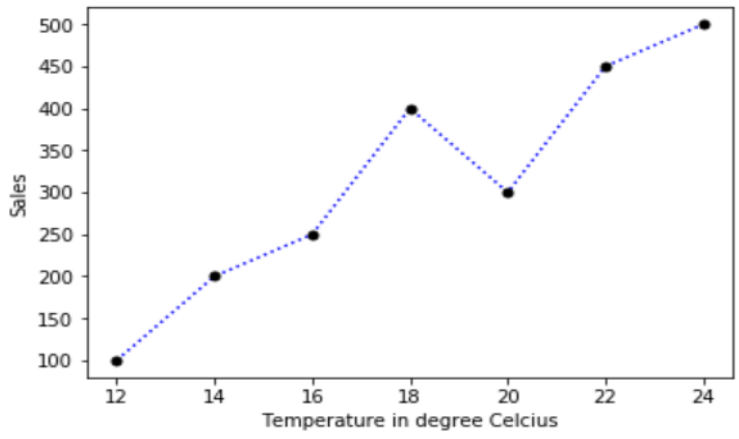
```
from matplotlib import pyplot as plt
```

**OUTPUT:**

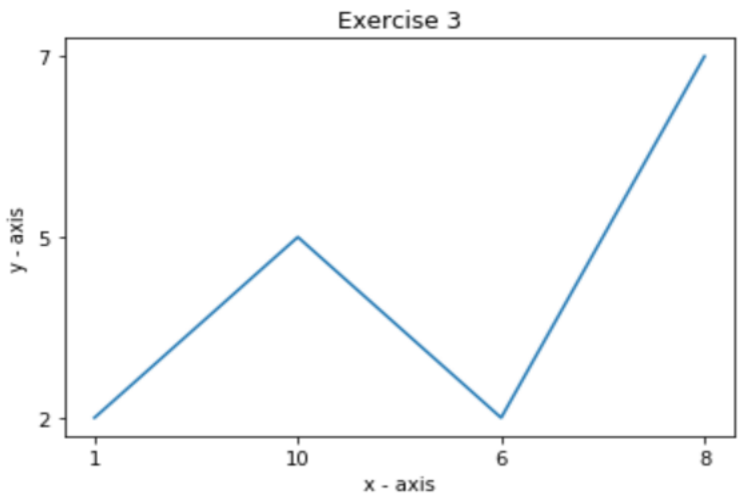
1)


20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
2.5
5.0
7.5
10.0
12.5
15.0
17.5

2)


500
450
400
350
300
250
200
150
100
Sales
12
14
16
18
20
22
24
Temperature in degree Celcius

3)


Exercise 3
7
5
2
y - axis
1
10
6
8
x - axis

**AIM:**

1. Draw a line in a diagram from position (1, 3) to (2, 10) then to (6, 12) and finally to position (18, 20). (Mark each point with a beautiful green colour and set line colour to red and line style dotted)

```
    importmatplotlib.pyplotasplt
        importnumpyas np
        x=[1,2,6,18]
        y=[3,10,12,20]
    plt.plot(x,y,'o:r',mec='k',mfc='g', ms=10)
plt.show()
```

2. Draw a plot for the following data:

| Temperature in degree Celsius | Sales |
|---|---|
| 12 | 100 |
| 14 | 200 |
| 16 | 250 |
| 18 | 400 |
| 20 | 300 |
| 22 | 450 |
| 24 | 500 |

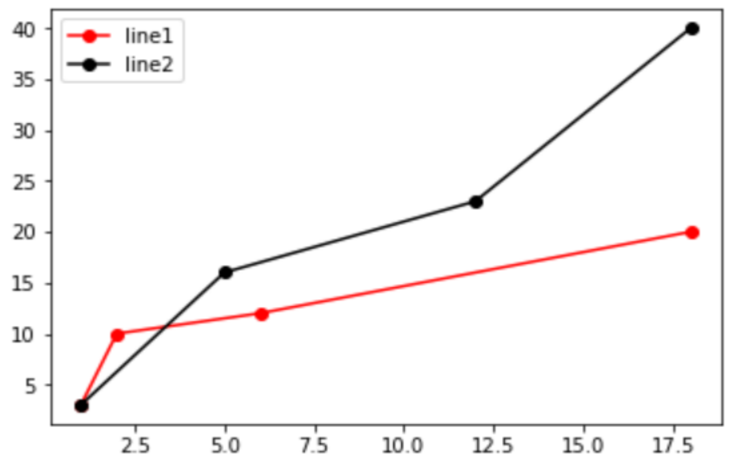
```
x  =[12,14,16,18,20,22,24]
y=[100,200,250,400,300,450,500]
plt.plot(x,y,'o:b',mec='k',mfc='k', ms=5)
plt.xlabel('Temperature in degree Celcius')
plt.ylabel('Sales')
plt.show()
```

3. Write a Python program to draw a line using given axis values taken from a text file, with suitable label in the x axis, y axis and a title.

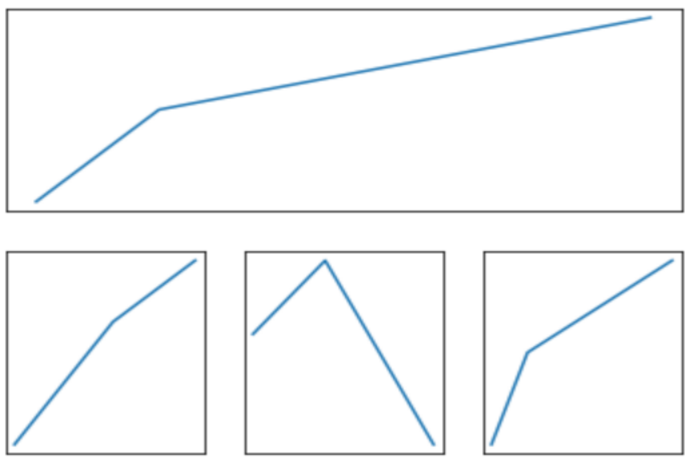
```
with open("test.txt") as f:
data =f.read()
data =data.split('\n')
x = [row.split(' ')[0] for row in data]
y = [row.split(' ')[1] for row in data]
plt.plot(x, y)
print(type(x[0]))
plt.xlabel('x - axis')
plt.ylabel('y - axis')
plt.title('Exercise 3')
plt.show()
```

**OUTPUT:**

4)



5)



4. Write a Python program to plot two or more lines on same plot with suitable legends of each line

```
x1=[1,2,6,18]
y1=[3,10,12,20]
plt.plot(x1,y1,'o-r',label="line1")
x2=[1,5,12,18]
y2=[3,16,23,40]
plt.plot(x2,y2,'o-k',label="line2")
plt.legend()
plt.show()
```

5. Write a Python program to create multiple plots.

```
fig =plt.figure()
        fig.subplots_adjust(bottom=0.020, left=0.020, top = 0.900, right=0.800)
        y=[1,4,7]
        x=[5,8,20]
        y1=[4,6,1]
        x1=[5,11,20]
        y2=[1,5,7]
        x2=[9,15,20]
        plt.subplot(2, 1, 1)
        plt.plot(x,y)
        plt.xticks(()), plt.yticks(())
        plt.subplot(2, 3, 4)
        plt.plot(x2,y2)
        plt.xticks(())
        plt.yticks(())
        plt.subplot(2, 3, 5)
        plt.plot(x1,y1)
        plt.xticks(())
        plt.yticks(())
        plt.subplot(2, 3, 6)
        plt.plot(x,y)
        plt.xticks(())
        plt.yticks(())
        plt.show()
```

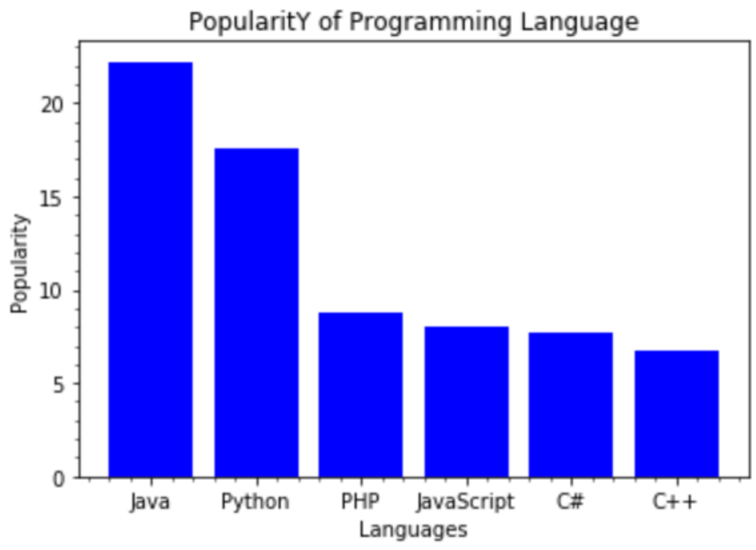
6. Consider the following data.

| Programming languages: | Java | Python | PHP | JavaScript | C# | C++ |
|---|---|---|---|---|---|---|
| Popularity | 22.2 | 17.6 | 8.8 | 8 | 77 | 6.7 |

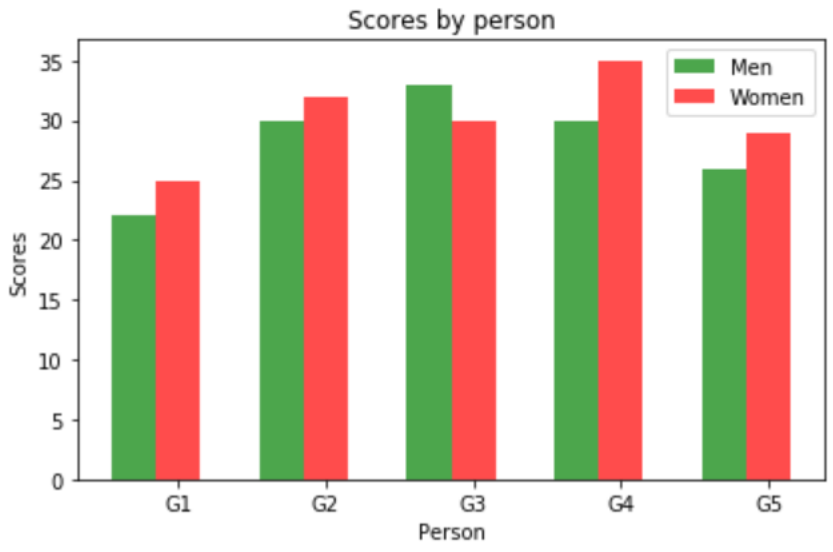
i. Write a Python programming to display a bar chart of the popularity of programming Languages.
ii. Write a Python programming to display a horizontal bar chart of the popularity of programming Languages (Give Red colour to the bar chart).
iii. Write a Python programming to display a bar chart of the popularity of programming Languages. Use different colour for each bar.

**OUTPUT:**

6)

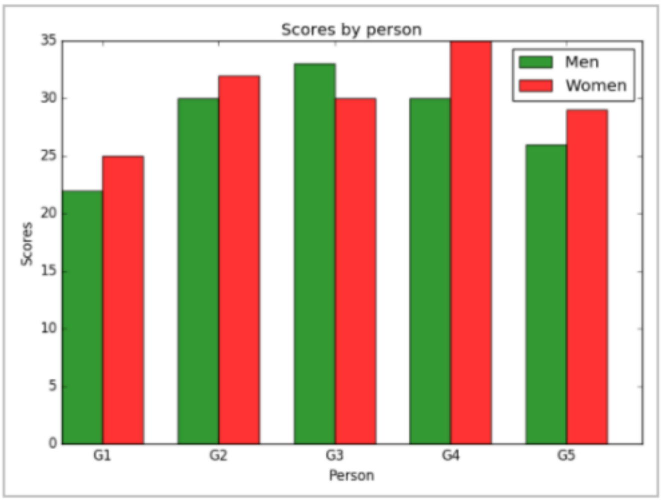

PopularitY of Programming Language
Popularity
0
5
10
15
20
Java
Python
PHP
JavaScript
C#
C++
Languages

7)


Scores by person
Scores
0
5
10
15
20
25
30
35
Men
Women
G1
G2
G3
G4
G5
Person

```
x = ['Java', 'Python', 'PHP', 'JavaScript', 'C#', 'C++']
popularity = [22.2, 17.6, 8.8, 8, 7.7, 6.7]
x_pos= [ifori, _ in enumerate(x)]
plt.bar(x_pos, popularity, color='blue')
plt.xlabel("Languages")
plt.ylabel("Popularity")
plt.title("PopularitY of Programming Language")
plt.xticks(x_pos, x)
# Turn on the grid
plt.minorticks_on()
plt.show()
```

7. Write a Python program to create bar plot of scores by group and gender. Use multiple X values onthe same chart for men and women.
   Sample Data:
   Means (men) = (22, 30, 35, 35, 26)
   Means (women) = (25, 32, 30, 35, 29)

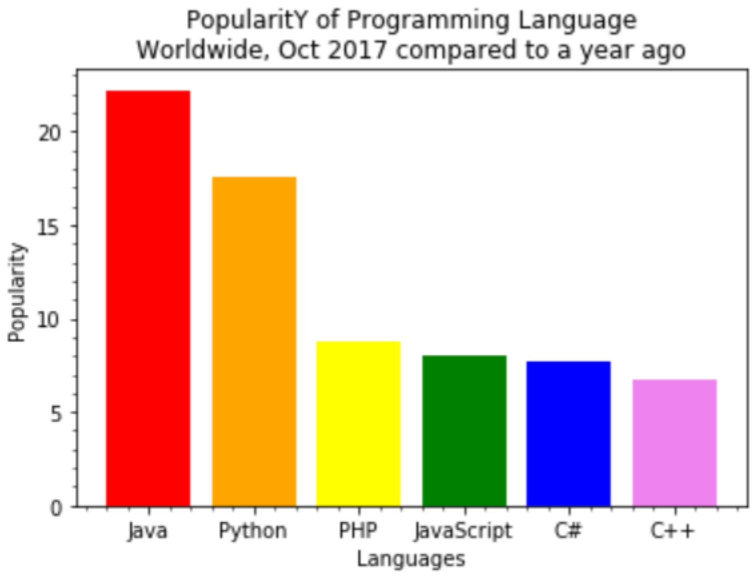


**PROGRAM:**

```
importnumpyas np

men_means= (22, 30, 33, 30, 26)
women_means= (25, 32, 30, 35, 29)

fig, ax=plt.subplots()
index =np.arange(5)
bar_width= 0.30
opacity = 0.7

rects1 =plt.bar(index, men_means, bar_width,alpha=opacity,color='g',label='Men')
```

**OUTPUT:**

8)


PopularitY of Programming Language
Worldwide, Oct 2017 compared to a year ago
Popularity
0
5
10
15
20
Java
Python
PHP
JavaScript
C#
C++
Languages

```
rects2 =plt.bar(index +bar_width, women_means,
bar_width,alpha=opacity,color='r',label='Women')
plt.xlabel('Person')
plt.ylabel('Scores')
plt.title('Scores by person')
plt.xticks(index +bar_width, ('G1', 'G2', 'G3', 'G4', 'G5'))
plt.legend()

plt.tight_layout()
plt.show()
```

8. Write a Python programming to create a pie chart of the popularity of programming Languages.

| Programming languages: | Java | Python | PHP | JavaScript | C# | C++ |
|---|---|---|---|---|---|---|
| Popularity | 22.2 | 17.6 | 8.8 | 8 | 77 | 6.7 |

**PROGRAM:**

```
x = ['Java', 'Python', 'PHP', 'JavaScript', 'C#', 'C++']
popularity = [22.2, 17.6, 8.8, 8, 7.7, 6.7]
x_pos= [ifori, _ in enumerate(x)]

plt.bar(x_pos, popularity, color=['red', 'orange','yellow', 'green', 'blue', 'violet'])

plt.xlabel("Languages")
plt.ylabel("Popularity")
plt.title("PopularitY of Programming Language\n" + "Worldwide, Oct 2017 compared
to a year ago")
plt.xticks(x_pos, x)
# Turn on the grid
plt.minorticks_on()
plt.show()
```

9. Write a Python programming to create a pie chart of gold medal achievements of five most successful countries in 2016 Summer Olympics. Read the data from a csv file.

Sample data:
**medal.csv**

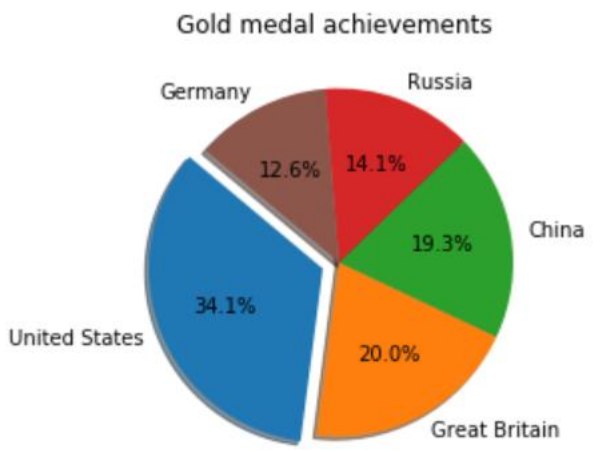
```
country,gold_medal
United States,46
Great Britain,27
China,26
Russia,19
Germany,17
```

**PROGRAM:**

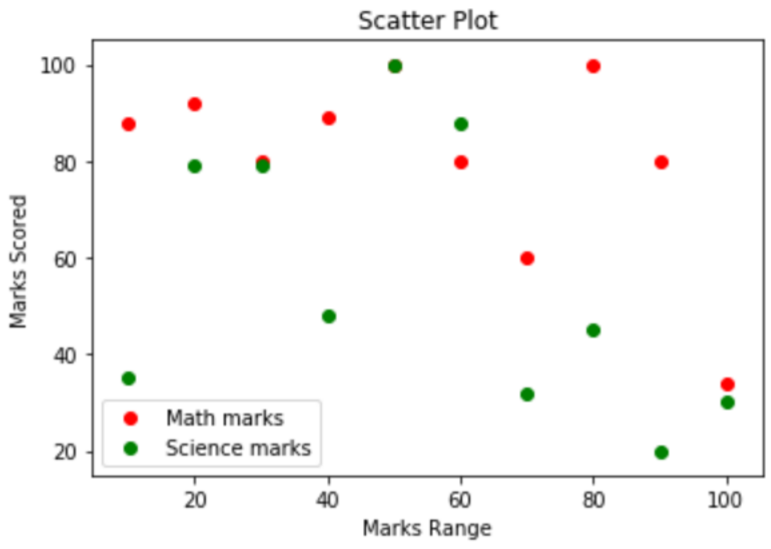
```
import pandas as pd
    df=pd.read_csv('medal.csv')
    country_data=df["country"]
```

**OUTPUT:**

9)


Gold medal achievements
Germany
Russia
China
United States
Great Britain
12.6%
14.1%
19.3%
34.1%
20.0%

10)

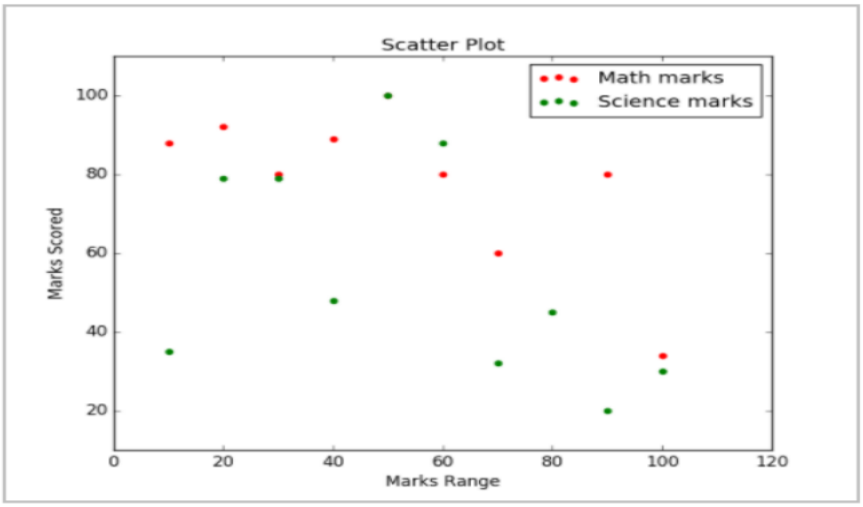

Scatter Plot
Marks Scored
Marks Range
● Math marks
● Science marks

```
medal_data=df["gold_medal"]
colors= ["#1f77b4", "#ff7f0e", "#2ca02c", "#d62728", "#8c564b"]
explode = (0.1, 0, 0, 0, 0)
plt.pie(medal_data, labels=country_data, explode=explode, colors=colors,
autopct='%1.1f%%', shadow=True, startangle=140)
plt.title("Gold medal achievements ")
plt.show()
```

10. Write a Python program to draw a scatter plot comparing two subject marks of Mathematics andScience. Use marks of 10 students.

Sample data:

Test Data:
math_marks = [88, 92, 80, 89, 100, 80, 60, 100, 80, 34]
science_marks = [35, 79, 79, 48, 100, 88, 32, 45, 20, 30]
marks_range = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]



**PROGRAM:**

```
import pandas as pd
math_marks= [88, 92, 80, 89, 100, 80, 60, 100, 80, 34]
science_marks= [35, 79, 79, 48, 100, 88, 32, 45, 20, 30]
marks_range= [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
plt.scatter(marks_range, math_marks, label='Math marks', color='r')
plt.scatter(marks_range, science_marks, label='Science marks', color='g')
plt.title('Scatter Plot')
plt.xlabel('Marks Range')
plt.ylabel('Marks Scored')
plt.legend()
plt.show()
```

**RESULT**: The program is executed and output is verified

Date :

# 4. PROGRAMS USING PANDAS

Pandas is a Python library used for working with data sets. It has functions for analyzing, cleaning, exploring, and manipulating data.

**Why Use Pandas**

Pandas allows us to analyze big data and make conclusions based on statistical theories.

Pandas can clean messy data sets, and make them readable and relevant.

Relevant data is very important in data science.

**What Can Pandas Do?**

Pandas gives you answers about the data. Like:

- Is there a correlation between two or more columns?
- What is average value?
- Max value?
- Min value?

Pandas are also able to delete rows that are not relevant, or contains wrong values, like empty or NULL values. This is called cleaning the data.

**OUTPUT:**

1)
```
0    1
dtype: int64
```

2)
```
DatetimeIndex(['2021-05-01', '2021-05-02', '2021-05-03', '2021-05-04',
               '2021-05-05', '2021-05-06', '2021-05-07', '2021-05-08',
               '2021-05-09', '2021-05-10', '2021-05-11', '2021-05-12'],
dtype='datetime64[ns]', freq='D')
```

3)
```
name  ageocc
0   Vinay   22    engineer
1  Kushal   25      doctor
2    Aman   24  accountant


id    name  age
0   2  Vishal   22
1   1  Kushal   25
2   1    Aman   24
```

4)
```
id    name  ageocc  salary
0   3   Vinay   22    engineer   60000
1   1  Kushal   25      doctor   70000
2   3    Aman   24  accountant   50000
3   2   Rahul   27      doctor   60000
4   1  Ramesh   31      doctor   65000
```

5)
```
id    name  ageocc
1   1  Kushal   25      doctor
0   3   Vinay   22    engineer
2   3    Aman   24  accountant
```

6)
```
id  ageocc
name
Vinay    3   22    engineer
Kushal   1   25      doctor
Aman     3   24  accountant

name  id  age         occ
0   Vinay   3   22    engineer
1  Kushal   1   25      doctor
2    Aman   3   24  accountant
```

7)
```
name  id  age       occ
0   Vinay   3   22  engineer
1    Kushal   1   25    doctor
```

1. Write a python program to implement List-to-Series Conversion.

```
import pandas as pd
l1 = [int(i) foriininput("Enter the list:").split()]
x =pd.Series(l1)
print(x)
```

2. Write a python program to Generate the series of dates from 1st May, 2021 to 12th May, 2021 (both inclusive).

```
dt =pd.date_range(start = '05-01-2021', end = '05-12-2021')
print(dt)
```

3. Given a dictionary, convert it into corresponding dataframe and display it.

```
dictionary = {'name': ['Vinay', 'Kushal', 'Aman'],
            'age' : [22, 25, 24],
            'occ' : ['engineer', 'doctor', 'accountant']}
df=pd.DataFrame(dictionary)
print(df)
```

4. a 2D List, convert it into corresponding dataframe and display it.

```
lists = [[2, 'Vishal', 22],
        [1, 'Kushal', 25],
        [1, 'Aman' Given, 24]]
df=pd.DataFrame(lists, columns = ['id', 'name', 'age'])
print(df)
```

5. Given a CSV file, read it into a dataframe and display it.

```
df=pd.read_csv('data.csv')
print(df)
```

6. Given a dataframe, sort it by multiple columns.

```
print(df)
df_sorted=df.sort_values(by = ['id', 'age'])
print()
print(df_sorted)
```

7. Given a dataframe with custom indexing, convert and it to default indexing and display it.

```
print(df)
df_customindex=df.set_index('name')
print(df_customindex)
df=df_customindex.reset_index()
print(df)
```

8)
```
occ
accountant    50000.0
doctor        65000.0
engineer      60000.0
Name: salary, dtype: float64
```

9)
```
    name   age         occ   salary
0   Vinay   NaNengineer  65000.0
1  Kushal  25.0      doctor      NaN
2    Aman  24.0  accountant  50000.0

  name   age         occ   salary
0   Vinay   0.0    engineer  65000.0
1  Kushal  25.0      doctor      0.0
2    Aman  24.0  accountant  50000.0
```

10)
```
cname  profit
0          Shyam& Co.  -10000
1       Ramlal& Bros.   10000
2  Sharma Enterprises   -5000
3    Verma Furnitures   15000
4        Rahul Stores   20000
cname  profit
0          Shyam& Co.   False
1       Ramlal& Bros.    True
2  Sharma Enterprises   False
3    Verma Furnitures    True
4        Rahul Stores    True
```

11)
```
eidename  stipend
0    1     Sid    10000
1    2  Ramesh    10000
2    3     Ron     5000
3    4   Harry    15000

   eid         position
0    1         employee
1    2         employee
2    3           intern
   2    4  senior employee
```

12)
```
eidename  stipend         position
0    1     Sid    10000         employee
1    2  Ramesh    10000         employee
2    3     Ron     5000           intern
3    4   Harry    15000  senior employee
```

8.Given a dataframe, select first 2 rows and output them.

```
o =df.iloc[[0,1], :]
```

```
print()
print(o)
```

9. Given is a dataframe showing name, occupation, salary of people. Find the average salary per occupation.

```
avg=df.groupby('occ')['salary'].mean()
print(avg)
```

10. Given a dataframe with NaN Values, fill the NaN values with 0.

```
dictionary = {'name': ['Vinay', 'Kushal', 'Aman'],
            'age' : [None, 25, 24],
            'occ' : ['engineer', 'doctor', 'accountant'],
            'salary' : [65000,None,50000]}
df=pd.DataFrame(dictionary)
print(df)
print()
df_nullfill=df.fillna(0)
print(df_nullfill)
```

11. Given is a dataframe showing Company Names (cname) and corresponding Profits (profit). Convert the values of Profit column such that values in it greater than 0 are set to True and the rest are set to False.

```
data = {'cname' : ['Shyam&
Co.','Ramlal&Bros.','SharmaEnterprises','VermaFurnitures','Rahul Stores'] ,
             'profit' : [-10000,10000,-5000,15000,20000]}
company_data=pd.DataFrame(data)
print(company_data)
print()
company_data['profit'] =company_data['profit'].apply(lambda x:x>0)
print(company_data)
```

12. Given are 2 dataframes, with one dataframe containing Employee ID (eid), Employee Name (ename) and Stipend (stipend) and the other dataframe containing Employee ID (eid) and designation of the employee (designation). Output the Dataframe containing Employee ID (eid), Employee Name (ename), Stipend (stipend) and Position (position).

```
emp = {'eid' : [1,2,3,4],
    'ename' : ['Sid','Ramesh','Ron','Harry'],
    'stipend' : [10000,10000,5000,15000]}

company = {'eid':[1,2,3,4],
        'position' : ['employee','employee','intern','senior employee']}
emp_data=pd.DataFrame(emp)
company_data=pd.DataFrame(company)
print(emp_data)
print()
print(company_data)
print()
```

```
df=pd.merge(emp_data,company_data,how='inner',on='eid')
print(df)
```

**RESULT**: The program is executed and Output is verified

Date :

## 5. KNN ALGORITHM

For a given data point in the set, the algorithms find the distances between this and all other K numbers of datapoint in the dataset close to the initial point and votes for that category that has the most frequency.

Usually, Euclidean distance is taking as a measure of distance. Thus the end resultant model is just the labelled data placed in a space. This algorithm is popularly known for various applications like genetics, forecasting, etc. The algorithm is best when more features are present and out shows SVM in this case.

KNN reducing overfitting is a fact. On the other hand, there is a need to choose the best value for K. So now how do we choose K? Generally we use the Square root of the number of samples in the dataset as value for K. An optimal value has to be found out since lower value may lead to overfitting and higher value may require high computational complication in distance. So using an error plot may help. Another method is the elbow method. You can prefer to take root else can also follow the elbow method.

Let’s dive deep into the different steps of K-NN for classifying a new data point

Step 1: Select the value of K neighbours(say k=5)

Step 2: Find the K (5) nearest data point for our new data point based on Euclidean distance

Step 3: Among these K data points count the data points in each category

Step 4: Assign the new data point to the category that has the most neighbours of the new datapoint

**OUTPUT:**

```
Accuracy of the model :  0.9777777777777777

['versicolor']
```

**AIM:** Program to implement k-NN classification using any standard dataset available in the public domain and find the  accuracy of the algorithm.

**PROGRAM:**

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics

iris = load_iris()
x = iris.data
y = iris.target

my_knn = KNeighborsClassifier(n_neighbors = 3)
my_knn.fit(x_train,y_train)
y_pred = my_knn.predict(x_test)
print("Accuracy of the model : ,metrics.accuracy_score(y_test,y_pred)


sample = [[1,2,1,9]]
pred = my_knn.predict(sample)
pred_v = [iris.target_names[p] for p in pred]
print(pred_v)
```

**RESULT**: The program is executed and Output is verified

Date :

# 6. NAÏVE BAYES ALGORITHM

Naïve Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems.It is mainly used in *text classification* that includes a high-dimensional training dataset.Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions.

It is a probabilistic classifier, which means it predicts on the basis of the probability of an object. Some popular examples of Naïve Bayes Algorithm are spam filtration, Sentimental analysis, and classifying articles.

Bayes' Theorem:

Bayes' theorem is also known as **Bayes' Rule** or **Bayes' law**, which is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability.

The formula for Bayes' theorem is given as:

$$P(A|B)=\frac{P(B|A)P(A)}{P(B)}$$

**Where,P(A|B) is Posterior probability**: Probability of hypothesis A on the observed event B.

**P(B|A) is Likelihood probability**: Probability of the evidence given that the probability of a hypothesis is true.

**P(A) is Prior Probability**: Probability of hypothesis before observing the evidence.

**P(B) is Marginal Probability**: Probability of Evidence.

Working of Naïve Bayes' Classifier:

1. Convert the given dataset into frequency tables.
2. Generate Likelihood table by finding the probabilities of given features.
3. Now, use Bayes theorem to calculate the posterior probability.

**OUTPUT:**

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

Predicted output for [[5,5,4,4]]: [2]
Naive bayes score : 0.9466666666666667
```

**AIM:** Program to implement Naive Bayes algorithm using any standard dataset available in the public domain and find the accuracy of the algorithm.

**PROGRAM:**

```
fromsklearn.datasetsimportload_iris
fromsklearn.model_selectionimporttrain_test_split
fromsklearn.naive_bayesimportGaussianNB


X,y=load_iris(return_X_y=True)
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.5,random_state=0)


gnb=GaussianNB()


y_pred=gnb.fit(X_train,y_train).predict(X_test)
print(y_pred)
x_new=[[5,5,4,4]]
y_new=gnb.fit(X_train,y_train).predict(x_new)



print("Predicted output for [[5,5,4,4]]:",y_new)
print("Naive bayes score :",gnb.score(X_test,y_test))
```

**RESULT**: The program is executed and output is verified

Date:

## 7. LINEAR AND MULTIPLE LINEAR REGRESSION

**AIM:** Program to implement linear and multiple regression techniques using any standard dataset available in the public domain and evaluate its performance.

*Linear regression* assumes a linear relationship between the input variable (X) and a single output variable (Y). When there is a single input variable, the method is referred to as a simple linear regression.

In a simple linear regression, we can estimate the coefficients required by the model to make predictions on new data analytically. That is, the line for a simple linear regression model can be written as:

$y = B0 + B1 * x + \in$

where B0 and B1 are the coefficients we must estimate from the training data and ∈ is an error term. Once the coefficients are estimated, we can use this equation to predict output values for y conditional on new input examples of x. We use LinearRegression() from sklearn.linear_model.

*Multiple Linear Regression*: When one variable/column in a dataset is not sufficient to create a good model and make more accurate predictions, we'll use a multiple linear regression model instead of a simple linear regression model. The line equation for the multiple linear regression model is:

$y = \beta0 + \beta1X1 + \beta2X2 + \beta3X3 + .... + \beta pXp + e$

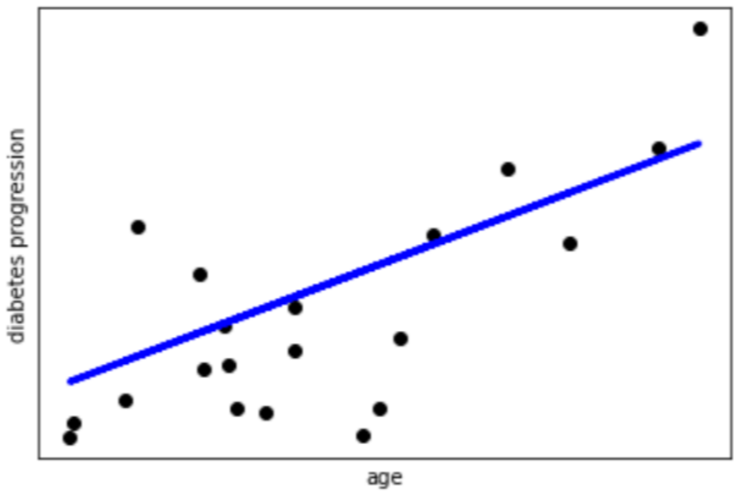
We use datasets.load_diabetes dataset for this program.

**About the dataset.**

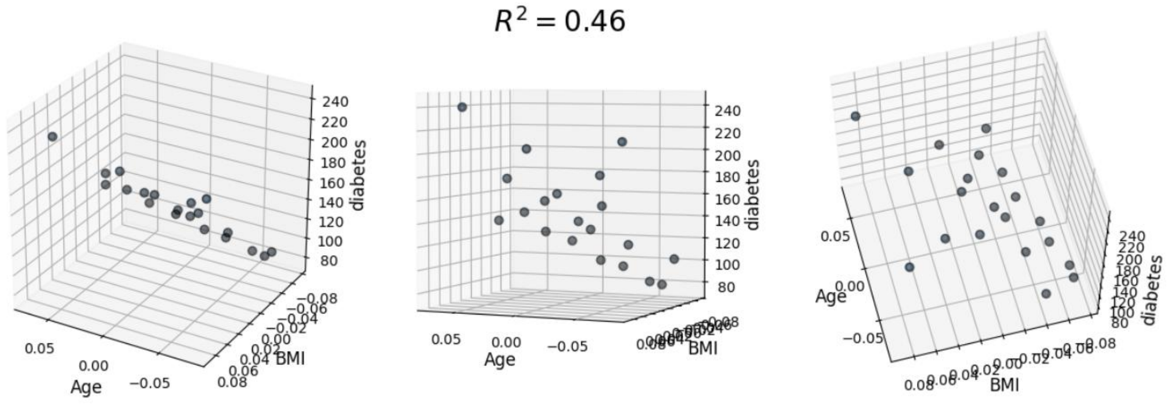
The diabetes dataset consists of 10 physiological variables (such as age, sex, weight, blood pressure) measure on 442 patients, and an indication of disease progression after one year. Note that the input variables have all been standardised to each have mean 0 and squared length = 1.

**OUTPUT:**

```
['age', 'sex', 'bmi', 'bp', 's1', 's2', 's3', 's4', 's5', 's6']
(442, 10)
(442,)
(442, 1)
LinearRegression()
Coefficients:
 [938.23786125]
Mean squared error: 2548.07
Coefficient of determination: 0.47
```



```
(442, 2)
LinearRegression()
Coefficients:
 [139.20420118 912.45355549]
Intercept:
 152.8767000140564
Mean squared error: 2596.60
Coefficient of determination: 0.46
```



**PROGRAM:**

```
importmatplotlib.pyplotasplt
importnumpyas np
fromsklearnimport datasets, linear_model
fromsklearn.metricsimportmean_squared_error, r2_score
df=datasets.load_diabetes()
df['feature_names']

diabetes_X, diabetes_y=datasets.load_diabetes(return_X_y=True)
diabetes_X.shape

diabetes_y.shape

diabetes_X=diabetes_X[:,np.newaxis, 2]
diabetes_X.shape

# Split the data into training/testing sets
diabetes_X_train=diabetes_X[:-20]
diabetes_X_test=diabetes_X[-20:]

# Split the targets into training/testing sets
diabetes_y_train=diabetes_y[:-20]
diabetes_y_test=diabetes_y[-20:]

# Create linear regression object
regr=linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(diabetes_X_train, diabetes_y_train)

diabetes_y_pred=regr.predict(diabetes_X_test)

# The coefficients
print("Coefficients: \n", regr.coef_)
# The mean squared error
print("Mean squared error: %.2f" %mean_squared_error(diabetes_y_test, diabetes_y_pred))
# The coefficient of determination: 1 is perfect prediction
print("Coefficient of determination: %.2f" % r2_score(diabetes_y_test, diabetes_y_pred))

# Plot outputs
plt.scatter(diabetes_X_test, diabetes_y_test, color="black")
plt.plot(diabetes_X_test, diabetes_y_pred, color="blue", linewidth=3)
plt.xlabel("age")
plt.ylabel("diabetes progression")
plt.xticks(())
plt.yticks(())
```

```
plt.show()
diabetes_X, diabetes_y=datasets.load_diabetes(return_X_y=True)
diabetes_X.shape
diabetes_X=diabetes_X[:,[0,2]]
diabetes_X.shape

# Split the data into training/testing sets
diabetes_X_train=diabetes_X[:-20]
diabetes_X_test=diabetes_X[-20:]

# Split the targets into training/testing sets
diabetes_y_train=diabetes_y[:-20]
diabetes_y_test=diabetes_y[-20:]

# Create linear regression object
regr=linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(diabetes_X_train, diabetes_y_train)

# Make predictions using the testing set
diabetes_y_pred=regr.predict(diabetes_X_test)
# The coefficients
print("Coefficients: \n", regr.coef_)
print("Intercept: \n", regr.intercept_)
# The mean squared error
print("Mean squared error: %.2f" %mean_squared_error(diabetes_y_test, diabetes_y_pred))
# The coefficient of determination: 1 is perfect prediction
print("Coefficient of determination: %.2f" % r2_score(diabetes_y_test, diabetes_y_pred))

x =diabetes_X_test[:, 0]
y =diabetes_X_test[:, 1]
#z = diabetes_X_test[:, 2]
plt.style.use('default')

fig =plt.figure(figsize=(12, 4))

ax1 =fig.add_subplot(131, projection='3d')
ax2 =fig.add_subplot(132, projection='3d')
ax3 =fig.add_subplot(133, projection='3d')

axes = [ax1, ax2, ax3]

foraxin axes:
ax.plot(x, y, diabetes_y_pred, color='k', zorder=15, linestyle='none', marker='o', alpha=0.5)
ax.scatter(x.flatten(), y.flatten(), diabetes_y_pred, facecolor=(0,0,0,0), s=20,
edgecolor='#70b3f0')
ax.set_xlabel('Age', fontsize=12)
ax.set_ylabel('BMI', fontsize=12)
ax.set_zlabel('diabetes', fontsize=12)
```

```
ax.locator_params(nbins=4, axis='x')
ax.locator_params(nbins=5, axis='x')

ax1.view_init(elev=28, azim=120)
ax2.view_init(elev=4, azim=114)
ax3.view_init(elev=60, azim=165)

fig.suptitle('$R^2 = %.2f$' % r2_score(diabetes_y_test, diabetes_y_pred), fontsize=20)

fig.tight_layout()
```

**RESULT**: The program is executed and output is verified

Date:

## 8. SUPPORT VECTOR MACHINE

**AIM:** Program to implement text classification using Support vector machine.

A **Support Vector Machine model** is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. In addition to performing linear classification, SVMs can efficiently perform a non-linear classification, implicitly mapping their inputs into high-dimensional feature spaces.

A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples.

**Concept of hyperplane:**

A hyperplane is a decision boundary that differentiates the two classes in SVM. A data point falling on either side of the hyperplane can be attributed to different classes. The dimension of the hyperplane depends on the number of input features in the dataset.

For two dimensions, the separating line was the hyperplane. Similarly, for three dimensions a plane with two dimensions divides the 3d space into two parts and thus act as a hyperplane. Thus for a space of *n* dimensions we have a hyperplane of n-1 dimensions separating it into two parts.

**Text classification using SVM:**

Text Classification is an automated process of classification of text into predefined categories. We can classify Emails into spam or non-spam, news articles into different categories like Politics, Stock Market, Sports, etc. SVM can be applied to any kind of vectors which encode any kind of data. This means that in order to leverage the power of svm text classification, texts have to be transformed into vectors.

Vectors are (sometimes huge) lists of numbers which represent a set of coordinates in some space. when SVM determines the decision boundary we mentioned above, SVM decides where to draw the best “line” (or the best hyperplane) that divides the space into two subspaces: one for the vectors which belong to the given category and one for the vectors which do not belong to it. So, provided we can find vector representations which encode as much information from our texts as possible, we will be able to apply the SVM algorithm to text classification problems and obtain very good results.

**Algorithm:**

1. Add the Required Libraries
2. Set random seed [This is used to reproduce the same result every time if the script is kept consistent otherwise each run will produce different results. The seed can be set to any number.]
3. Add the Corpus [The data set can be easily added as a pandas Data Frame with the help of ‘read_csv’ function. I have set the encoding to ‘latin-1’ as the text had many special characters.]

4. Data pre-processing [This basically involves transforming raw data into an understandable format for NLP models.]

(a) Remove Blank rows in Data, if any

(b) Change all the text to lower case

(c) Word Tokenization

(d) Remove Stop words

(e) Remove Non-alpha text

(f) Word Lemmatization

5. Prepare Train and Test Data sets [This can be done through the *train_test_split* from the sklearn library. The Training Data will have 70% of the corpus and Test data will have the remaining 30% as we have set the parameter test_size=0.3]
6. Encoding [Label encode the target variable — This is done to transform Categorical data of string type in the data set into numerical values which the model can understand.]
7. Word Vectorization [It is a general process of turning a collection of text documents into numerical feature vectors. Most popular method is called TF-IDF ("Term Frequency — Inverse Document" Frequency) which are the components of the resulting scores assigned to each word.]

(a) Term Frequency: This summarizes how often a given word appears within a document.

(b) Inverse Document Frequency: This down scales words that appear a lot across documents.

8. Use the SVM to Predict the outcome and display the accuracy of prediction.

**Data set**: Amazon Review Data set which has 10,000 rows of Text data which is classified into "Label 1" and "Label 2". The Data set has two columns "Text" and "Label". Data can be found https://github.com/Gunjitbedi/Text-Classification/blob/master/corpus.csv.

**OUTPUT:**

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.

(0, 4502)	0.3763188267807246
  (0, 4501)	0.15031494427382475
  (0, 3974)	0.35868777245753825
  (0, 3890)	0.2515140235472667
  (0, 3858)	0.2690675584422277
  (0, 3748)	0.34695623926050195
  (0, 3658)	0.2896999547088821
  (0, 3561)	0.29449641491430995
  (0, 2922)	0.229683025366997
  (0, 1940)	0.13406125327954532
  (0, 1536)	0.17761496997588844
  (0, 517)	0.321056290554803
  (0, 488)	0.12303572865008613
  (0, 238)	0.2448559358109696
  (1, 4687)	0.21384275526442909
  (1, 4069)	0.3566872275481094
  (1, 3434)	0.21279175847748263
  (1, 3319)	0.8157357261127677
  (1, 2595)	0.2173336717856602
  (1, 1252)	0.2074693534878867
  (1, 598)	0.1614401835472762
  (2, 4734)	0.21251405574612364
  (2, 4621)	0.17383471522304228
  (2, 4464)	0.11898591577849023
  (2, 4197)	0.13515537469996092
  :	:
  (6998, 2522)	0.11512409752599596
  (6998, 2130)	0.13650214385741868
  (6998, 1976)	0.07126908030410523
  (6998, 1788)	0.22013355385880556
  (6998, 1755)	0.19935027840675415
  (6998, 1719)	0.13508979239544552
  (6998, 1595)	0.13521530232348064
  (6998, 1579)	0.1852028677205329
  (6998, 1545)	0.13386451534221278
  (6998, 1540)	0.09899700620855782
  (6998, 1299)	0.2957222991515454
  (6998, 1186)	0.2328691080708402
```

**PROGRAM:**

```
import pandas as pd
importnumpyas np
fromnltk.tokenizeimportword_tokenize
fromnltkimportpos_tag
importnltk
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')
nltk.download('stopwords')
fromnltk.stemimportWordNetLemmatizer
fromsklearn.preprocessingimportLabelEncoder
from collections importdefaultdict
fromnltk.corpusimport wordnet aswn
fromsklearn.feature_extraction.textimportTfidfVectorizer
fromsklearnimportmodel_selection, naive_bayes, svm
fromsklearn.metricsimportaccuracy_score


np.random.seed(500)


Corpus =pd.read_csv(r"corpus.csv",encoding='latin-1')

# Step - a : Remove blank rows if any.
Corpus['text'].dropna(inplace=True)
# Step - b : Change all the text to lower case. This is required as python interprets 'dog' and
'DOG' differently
Corpus['text'] = [entry.lower() for entry in Corpus['text']]
# Step - c : Tokenization : In this each entry in the corpus will be broken into set of words
Corpus['text']= [word_tokenize(entry) for entry in Corpus['text']]
# Step - d : Remove Stop words, Non-Numeric and perfom Word Stemming/Lemmenting.
# WordNetLemmatizer requires Pos tags to understand if the word is noun or verb or
adjective etc. By default it is set to Noun
tag_map=defaultdict(lambda : wn.NOUN)
tag_map['J'] =wn.ADJ
tag_map['V'] =wn.VERB
tag_map['R'] =wn.ADV
forindex,entryin enumerate(Corpus['text']):
# Declaring Empty List to store the words that follow the rules for this step
Final_words= []
# Initializing WordNetLemmatizer()
word_Lemmatized=WordNetLemmatizer()
# pos_tag function below will provide the 'tag' i.e if the word is Noun(N) or Verb(V) or
something else.
for word, tag inpos_tag(entry):
# Below condition is to check for Stop words and consider only alphabets
if word notinstopwords.words('english') andword.isalpha():
word_Final=word_Lemmatized.lemmatize(word,tag_map[tag[0]])
Final_words.append(word_Final)
```

```
  (6998, 488)	0.36187164406935557
  (6999, 4866)	0.16162053232828144
  (6999, 4149)	0.5217385422290042
  (6999, 3823)	0.3649735868975402
  (6999, 2902)	0.16568268411504228
  (6999, 2699)	0.14615260074275177
  (6999, 1976)	0.14155242939037596
  (6999, 1672)	0.28173977593289695
  (6999, 1536)	0.17292898689607455
  (6999, 1406)	0.2877416804485293
  (6999, 1238)	0.314256617088249
  (6999, 319)	0.28499442233774624
  (6999, 50)	0.3571519291530235

SVM Accuracy Score ->  84.7
```

```
# The final processed set of words for each iteration will be stored in 'text_final'
Corpus.loc[index,'text_final'] =str(Final_words)
Train_X,Test_X,Train_Y,Test_Y=model_selection.train_test_split(Corpus['text_final'],Corpu
s['label'],test_size=0.3)
Encoder =LabelEncoder()
Train_Y=Encoder.fit_transform(Train_Y)
Test_Y=Encoder.fit_transform(Test_Y)
Tfidf_vect=TfidfVectorizer(max_features=5000)
Tfidf_vect.fit(Corpus['text_final'])
Train_X_Tfidf=Tfidf_vect.transform(Train_X)
Test_X_Tfidf=Tfidf_vect.transform(Test_X)

print(Tfidf_vect.vocabulary_)
print(Train_X_Tfidf)

# Classifier - Algorithm - SVM
# fit the training dataset on the classifier
SVM =svm.SVC(C=1.0, kernel='linear', degree=3, gamma='auto')
SVM.fit(Train_X_Tfidf,Train_Y)
# predict the labels on validation dataset
predictions_SVM=SVM.predict(Test_X_Tfidf)
# Use accuracy_score function to get the accuracy
print("SVM Accuracy Score -> ",accuracy_score(predictions_SVM, Test_Y)*100)
```

**RESULT**: The program is executed and output is verified

Date:

## 9. DECISION TREE

Decision tree is a type of supervised learning algorithm (having a predefined target variable) that is mostly used in classification problems. It works for both categorical and continuous input and output variables. In this technique, we split the population or sample into two or more homogeneous sets (or sub-populations) based on most significant splitter / differentiator in input variables.

**Used Python Packages:**

**sklearn** : In python, sklearn is a machine learning package which include a lot of ML algorithms. Here, we are using some of its modules like train_test_split, DecisionTreeClassifier and accuracy_score.

**NumPy** : It is a numeric python module which provides fast maths functions for calculations. It is used to read data in numpy arrays and for manipulation purpose.

**Pandas** : Used to read and write different files. Data manipulation can be done easily with dataframes.

Important Terminology related to Tree based Algorithms:

**Root Node**: It represents entire population or sample and this further gets divided into two or more homogeneous sets.

**Splitting**: It is a process of dividing a node into two or more sub-nodes.

**Decision Node**: When a sub-node splits into further sub-nodes, then it is called decision node.

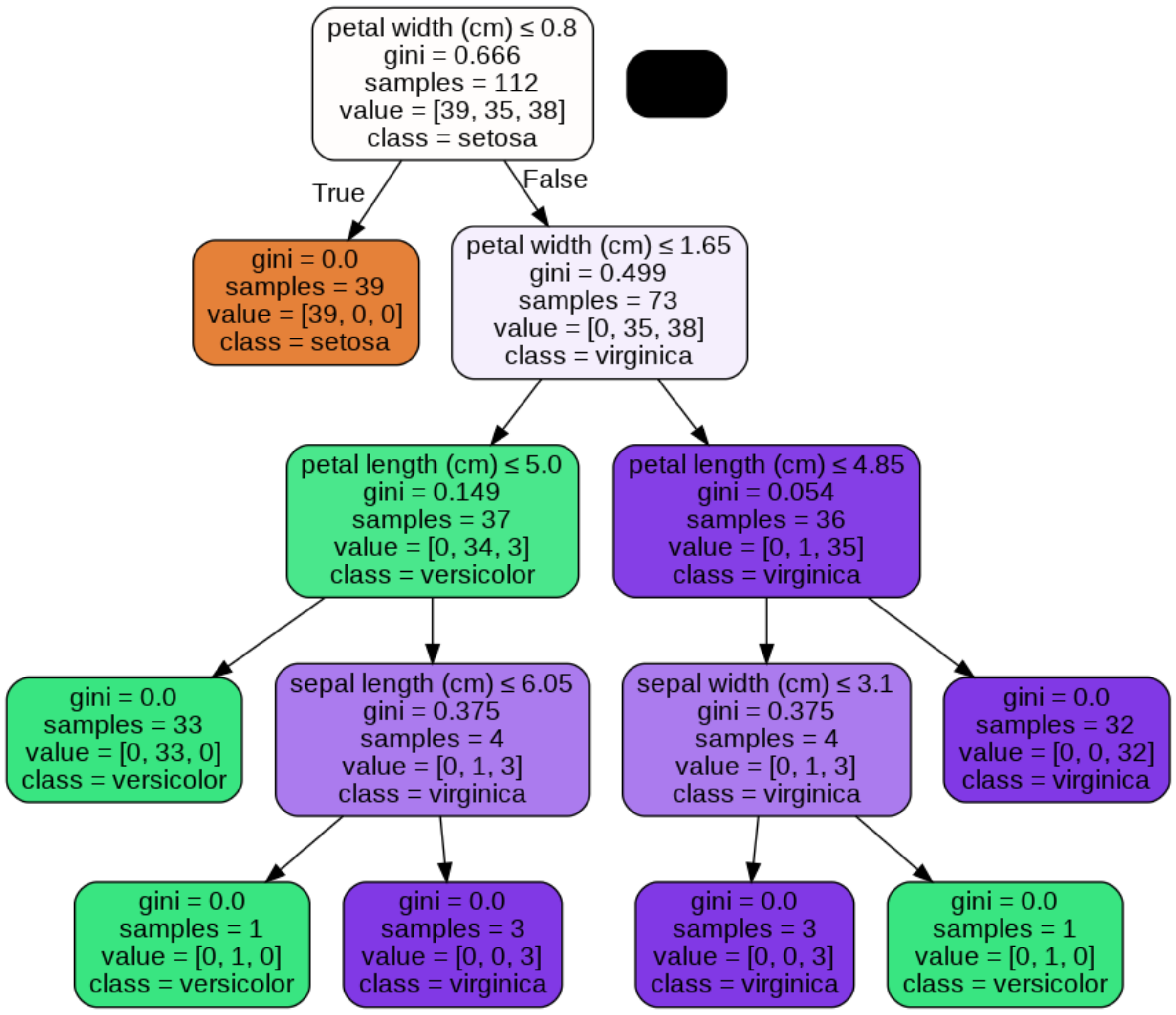
**Leaf/ Terminal Node**: Nodes do not split is called Leaf or Terminal node.

**Pruning**: When we remove sub-nodes of a decision node, this process is called pruning. You can say opposite process of splitting.

**Branch / Sub-Tree**: A sub section of entire tree is called branch or sub-tree.

**Parent and Child Node:** A node, which is divided into sub-nodes is called parent node of sub-nodes whereas sub-nodes are the child of parent node.

**OUTPUT:**


petal width (cm) ≤ 0.8
gini = 0.666
samples = 112
value = [39, 35, 38]
class = setosa
True
False
gini = 0.0
samples = 39
value = [39, 0, 0]
class = setosa
petal width (cm) ≤ 1.65
gini = 0.499
samples = 73
value = [0, 35, 38]
class = virginica
petal length (cm) ≤ 5.0
gini = 0.149
samples = 37
value = [0, 34, 3]
class = versicolor
petal length (cm) ≤ 4.85
gini = 0.054
samples = 36
value = [0, 1, 35]
class = virginica
gini = 0.0
samples = 33
value = [0, 33, 0]
class = versicolor
sepal length (cm) ≤ 6.05
gini = 0.375
samples = 4
value = [0, 1, 3]
class = virginica
sepal width (cm) ≤ 3.1
gini = 0.375
samples = 4
value = [0, 1, 3]
class = virginica
gini = 0.0
samples = 32
value = [0, 0, 32]
class = virginica
gini = 0.0
samples = 1
value = [0, 1, 0]
class = versicolor
gini = 0.0
samples = 3
value = [0, 0, 3]
class = virginica
gini = 0.0
samples = 3
value = [0, 0, 3]
class = virginica
gini = 0.0
samples = 1
value = [0, 1, 0]
class = versicolor

**AIM :**Program to implement decision trees using any standard dataset available in the public domain and find the accuracy of the algorithm.

**PROGRAM:**

```
import pandas as pd

import numpy as np

from sklearn.datasets import load_iris

data = load_iris()

X = data.data

y = data.target

display (X.shape, y.shape)

from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier

X_train, X_test, y_train, y_test = train_test_split(X, y,random_state = 50, test_size = 0.25)

#default criterion is Gini

classifier = DecisionTreeClassifier()

classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import accuracy_score

print('Accuracy on train data using Gini: ',accuracy_score(y_true = y_train, y_pred =
classifier.predict(X_train)))

print('Accuracy on test data using Gini: ',accuracy_score(y_true = y_test, y_pred = y_pred))

#change criterion to entropy

classifier_entropy = DecisionTreeClassifier(criterion='entropy')

classifier_entropy.fit(X_train, y_train)

y_pred_entropy = classifier_entropy.predict(X_test)

print('Accuracy on train data using entropy', accuracy_score(y_true=y_train, y_pred =
classifier_entropy.predict(X_train)))

print('Accuracy on test data using entropy', accuracy_score(y_true=y_test, y_pred =
y_pred_entropy))

#change criterion to entropy with min_samples_split to 50. Default value is 2

classifier_entropy1 = DecisionTreeClassifier(criterion='entropy', min_samples_split=50)
```

```
classifier_entropy1.fit(X_train, y_train)

y_pred_entropy1 = classifier_entropy1.predict(X_test)

print('Accuracy on train data using entropy', accuracy_score(y_true=y_train, y_pred =
classifier_entropy1.predict(X_train)))

print('Accuracy on test data using entropy', accuracy_score(y_true=y_test, y_pred =
y_pred_entropy1))

from sklearn.tree import export_graphviz

from six import StringIO

from IPython.display import Image

import pydotplus

dot_data = StringIO()

#the students can try using classifier, classifier_entropy and classifier_entropy1

#as first parameter below.

export_graphviz(classifier, out_file = dot_data,filled = True, rounded =
True,special_characters = True, feature_names = data.feature_names, class_names =
data.target_names)

graph = pydotplus.graph_from_dot_data(dot_data.getvalue())

Image(graph.create_png())
```

**RESULT**: The program is obtained and output is verified.

Date:

## 10. K MEANS CLUSTERING

**AIM:**Program to implement k- means clustering technique using any standard dataset available in the public domain.

K-Means clustering is an unsupervised learning algorithm. There is no labeled data for this clustering, unlike in supervised learning. K-Means performs the division of objects into clusters that share similarities and are dissimilar to the objects belonging to another cluster.

The term ‘K’ is a number. You need to tell the system how many clusters you need to create. For example, K = 2 refers to two clusters. There is a way of finding out what is the best or optimum value of K for a given data.

**ALGORITHM:**

**Step 1** − First, we need to specify the number of clusters, K, need to be generated by this algorithm.

**Step 2** − Next, randomly select K data points and assign each data point to a cluster. In simple words, classify the data based on the number of data points.

**Step 3** − Now it will compute the cluster centroids.

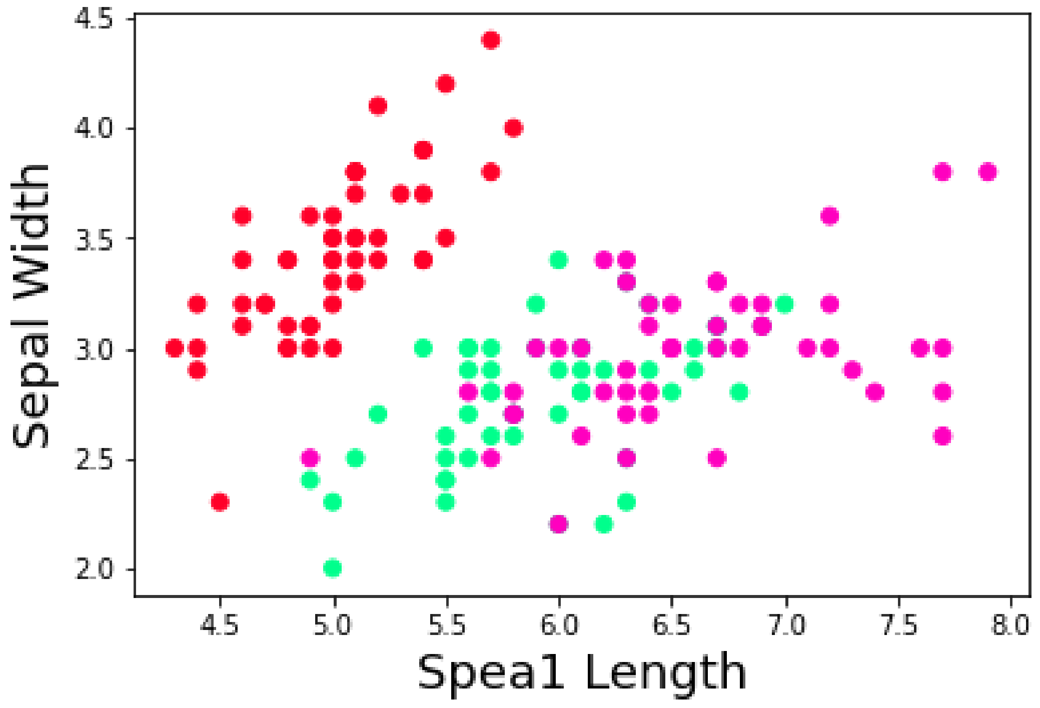
**Step 4** − Next, keep iterating the following until we find optimal centroid which is the assignment of data points to the clusters that are not changing any more −

**4.1** − First, the sum of squared distance between data points and centroids would be computed.

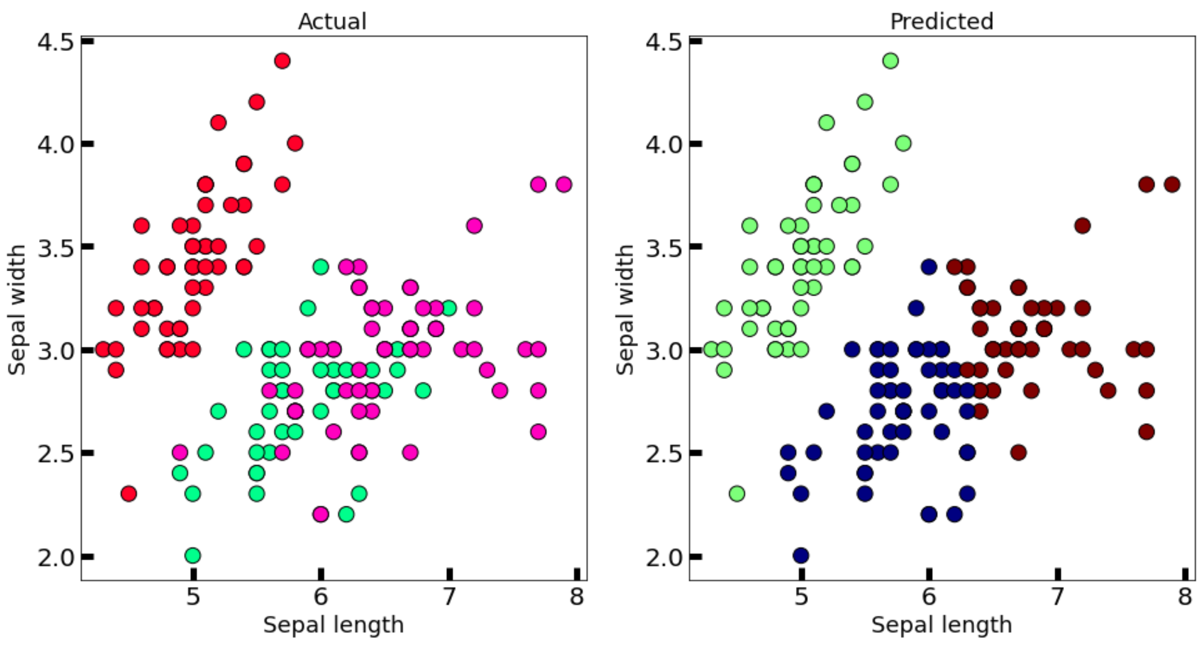
**4.2** − Now, we have to assign each data point to the cluster that is closer than other cluster (centroid).

**4.3** − At last compute the centroids for the clusters by taking the average of all data points of that cluster.

**OUTPUT:**



```
KMeans(n_clusters=3, random_state=21)
[[5.77358491 2.69245283]
 [5.006      3.428     ]
 [6.81276596 3.07446809]]
Text(0.5, 1.0, 'Predicted')
```



**PROGRAM:**

```
from sklearn import datasets

import matplotlib.pyplot as plt

import pandas as pd

from sklearn.cluster import KMeans

iris = datasets.load_iris()

X = iris.data[:, :2]

y = iris.target

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

plt.xlabel('Spea1 Length', fontsize=18)

plt.ylabel('Sepal Width', fontsize=18)

km=KMeans(n_clusters = 3, init='k-
means++', n_init=10, max_iter=300, tol=0.0001, verbose=0, random_state=21, copy_x=True,
 algorithm="auto")

km.fit(X)

centers = km.cluster_centers_

print(centers)

#this will tell us to which cluster does the data observations belong.

new_labels = km.labels_

# Plot the identified clusters and compare with the answers

fig, axes = plt.subplots(1, 2, figsize=(16,8))

axes[0].scatter(X[:, 0], X[:, 1], c=y, cmap='gist_rainbow',

edgecolor='k', s=150)

axes[1].scatter(X[:, 0], X[:, 1], c=new_labels, cmap='jet',

edgecolor='k', s=150)

axes[0].set_xlabel('Sepal length', fontsize=18)

axes[0].set_ylabel('Sepal width', fontsize=18)

axes[1].set_xlabel('Sepal length', fontsize=18)

axes[1].set_ylabel('Sepal width', fontsize=18)

axes[0].tick_params(direction='in', length=10, width=5, colors='k', labelsize=20)

axes[1].tick_params(direction='in', length=10, width=5, colors='k', labelsize=20)
```

```
axes[0].set_title('Actual', fontsize=18)
axes[1].set_title('Predicted', fontsize=18)
```

**RESULT**: The program is executed and output is verified

Date:

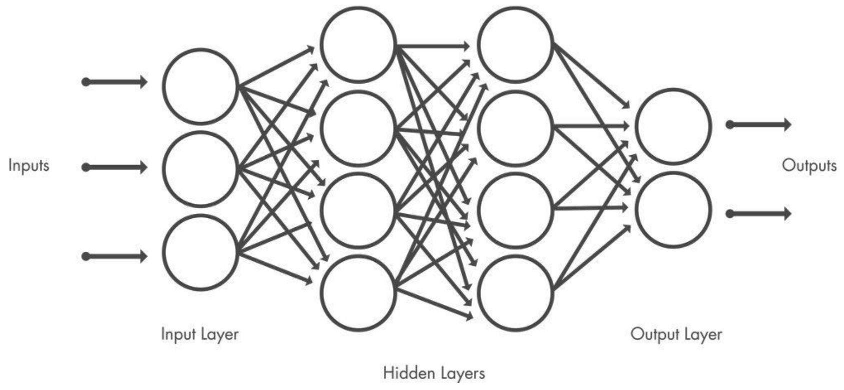
## 11. CONVOLUTIONAL NEURAL NETWORK

**AIM:** Program on convolutional neural network to classify images from any standard dataset in the public domain.

A convolutional neural network (CNN or ConvNet), is a network architecture for deep learning which learns directly from data, eliminating the need for manual feature extraction.

CNNs are particularly useful for finding patterns in images to recognize objects, faces, and scenes. They can also be quite effective for classifying non-image data such as audio, time series, and signal data.

Like other neural networks, a CNN is composed of an input layer, an output layer, and many hidden layers in between.



These layers perform operations that alter the data with the intent of learning features specific to the data. Three of the most common layers are: convolution, activation or ReLU, and pooling.

- **Convolution** puts the input images through a set of convolutional filters, each of which activates certain features from the images.
- **Rectified linear unit (ReLU)** allows for faster and more effective training by mapping negative values to zero and maintaining positive values. This is sometimes referred to as *activation*, because only the activated features are carried forward into the next layer.
- **Pooling** simplifies the output by performing nonlinear downsampling, reducing the number of parameters that the network needs to learn.

These operations are repeated over tens or hundreds of layers, with each layer learning to identify different features.

**OUTPUT:**

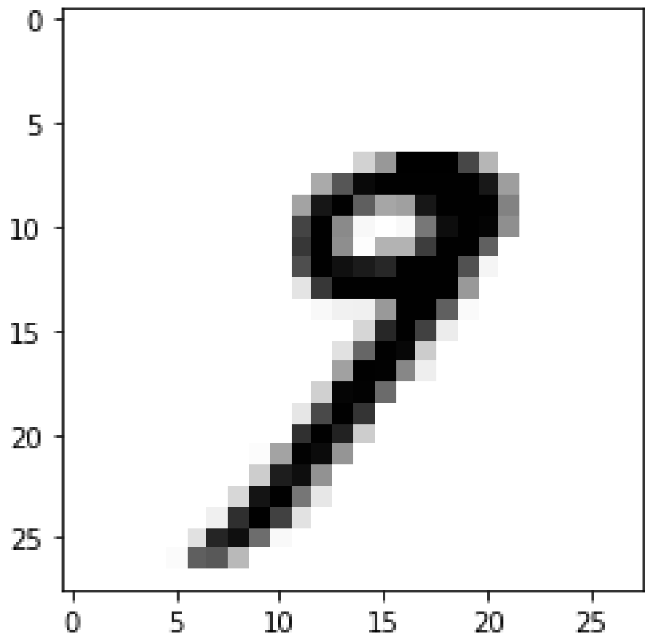
```
Shape of x_train: (60000, 28, 28)
Shape of x_test: (10000, 28, 28)
Shape of y_train: (60000,)
Shape of y_test: (10000,)
Shape of x_train: (60000, 28, 28, 1)
Shape of x_test: (10000, 28, 28, 1)
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_7 (Conv2D)           (None, 26, 26, 32)        320

 conv2d_8 (Conv2D)           (None, 24, 24, 32)        9248

 max_pooling2d_3 (MaxPooling  (None, 12, 12, 32)       0
 2D)

 dropout_5 (Dropout)         (None, 12, 12, 32)        0

 flatten_3 (Flatten)         (None, 4608)              0

 dense_5 (Dense)             (None, 128)               589952

 dropout_6 (Dropout)         (None, 128)               0

 dense_6 (Dense)             (None, 10)                1290

=================================================================
Total params: 600,810
```

**PROGRAM:**

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt


import tensorflow as tf

from tensorflow import keras

mnistDB = keras.datasets.mnist

(x_train,y_train),(x_test,y_test) = mnistDB.load_data()


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

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

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

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


plt.imshow(x_train[33],cmap='binary')

x_train = x_train.reshape((60000,28,28,1))

x_test = x_test.reshape((10000,28,28,1))


x_train = x_train.astype('float32')/255

x_test = x_test.astype('float32')/255


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

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


cnnModel_MNIST = keras.models.Sequential()


cnnModel_MNIST.add(keras.layers.Conv2D(32,(3,3),activation="relu",input_shape=x_train.
shape[1:]))

cnnModel_MNIST.add(keras.layers.Conv2D(32,(3,3),activation="relu"))

cnnModel_MNIST.add(keras.layers.MaxPooling2D((2,2)))
```

**OUTPUT:**

```
Trainable params: 600,810

Non-trainable params: 0

3750/3750 [==============================] - 145s 38ms/step - loss: 0.1386 -
accuracy: 0.9566

313/313 [==============================] - 6s 17ms/step - loss: 0.0434 - accuracy:
0.9859
```



```
cnnModel_MNIST.add(keras.layers.Dropout(0.25))


cnnModel_MNIST.add(keras.layers.Flatten())


cnnModel_MNIST.add(keras.layers.Dense(128,activation="relu"))
cnnModel_MNIST.add(keras.layers.Dropout(0.25))
cnnModel_MNIST.add(keras.layers.Dense(10,activation="softmax"))
cnnModel_MNIST.summary()
cnnModel_MNIST.compile(loss = 'sparse_categorical_crossentropy',optimizer='adam',metric
s=['accuracy'])
cnnModel_MNIST.fit(x_train,y_train,epochs=1,batch_size=16)
```

**RESULT**: The program is executed and output is verified.