Experiment-1

Aim:Introduction to required python libraries such as numpy,pandas,Matplotlib,Scipy,sklearn

Numpy:

In [3]:
```
import numpy as np
a1=np.array([1,2,3])
print(a1)
print(type(a1))
```

```
[1 2 3]
<class 'numpy.ndarray'>
```

In [6]:
```
import numpy as np
a2=np.array((1,2,3))
print(a2)
print(type(a2))
```

```
[1 2 3]
<class 'numpy.ndarray'>
```

In [19]:
```
import numpy as np
arr=np.array([[10,20],[30,40],[50,60]])
print(arr)
print("row1:",arr[0])
print("row2:",arr[1])
print("row3:",arr[2])
print("first two rows:/n",arr[0,:1:2])
```

```
[[10 20]
 [30 40]
 [50 60]]
row1: [10 20]
row2: [30 40]
row3: [50 60]
first two rows:/n [10]
```

In [10]:
```
import numpy as np
arr1=np.array([1,3,5])
print(arr1.ndim)
```

```
1
```

In [12]:
```
print(arr1.shape)
```

```
(3,)
```

In [13]:
```
print(arr1.size)
```

```
3
```

In [14]:
```
print(arr1.dtype)
```

```
int32
```

In [15]:
```
print(arr.itemsize)
```

```
4
```

In [24]:
```
import numpy as np
z1=np.zeros(5)
print(z1)
```

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

In [26]:
```
import numpy as np
z2=np.zeros((2,3))
print(z2)
```

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

In [28]:
```
import numpy as np
z3=np.zeros((2,3),dtype=np.uint8)
print(z3)
```

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

In [37]:
```
import numpy as np
a=np.array([1,2,3])
b=np.array([[1,2,3],[4,5,6]])
c=np.array([[[1,2,3],[4,5,6]],[[7,8,9],[2,8,4]]])
print(c)
print(a.ndim)
print(b.ndim)
print(c.ndim)
```

```
[[[1 2 3]
  [4 5 6]]

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

In [42]:
```
import numpy as np
arr=np.array([[1,2,3,4,5],[6,7,8,9,10]])
print('last element from 2nd dim:',arr[1,-3])
```

```
last element from 2nd dim: 8
```

In [49]:
```
import numpy as np
arr3=np.array([1,2,3,4,5,6,7,8])
print(arr3[1:4])
print(arr3[-3:-1])
print(arr3[1:5:2])
```

```
[2 3 4]
[6 7]
[2 4]
```

In [54]:
```
import numpy as np
a4=np.array([[1,2,3,4,5],[6,7,8,9,10]])
print(a4[1,1:4])
```

```
[7 8 9]
```

In [56]:
```
print(a4[1,-3:-1])
```

```
[8 9]
```

In [59]:
```
print(a4[0:2,1:4])
```

```
[[2 3 4]
 [7 8 9]]
```

In [60]:
```
print(a4[::,2:5])
```

```
[[ 3  4  5]
 [ 8  9 10]]
```

In [66]:
```
import numpy as np
a=np.array([[1,2,3,4,5],[6,7,8,9,10]])
a1=a.copy()
print(a1)
print(a.shape)
```

```
[[ 1  2  3  4  5]
 [ 6  7  8  9 10]]
(2, 5)
```

In [63]:
```
a=a1.view()
print(a1)
```

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

In [64]:
```
a=np.array(['a','b'])
print(a.dtype)
```

```
<U1
```

In [77]:
```
import numpy as np
arr1=np.array([1,2,3,4,5,6,7,8,9,10,11,12])
arr2=arr1.reshape(4,3)
arr3=arr1.reshape(2,3,2)
```

```
print(arr2)
print(arr3)
```

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

 [[ 7  8]
  [ 9 10]
  [11 12]]]
```

In [81]:
```
import numpy as np
a=np.array([[1,2,3],[6,7,8]])
for x in a:
    for y in x:
        print(y)
```

```
1
2
3
6
7
8
```

In [118…
```
import numpy as np

a1 = np.array([[1, 2, 3], [4, 5, 6]])
a2 = np.array([[7, 8, 9], [11, 12, 13]])


a3 = np.concatenate((a1, a2), axis=0)
print(a3)

a4 = np.concatenate((a1, a2), axis=1)
print(a4)
```

```
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [11 12 13]]
[[ 1  2  3  7  8  9]
 [ 4  5  6 11 12 13]]
```

In [116…
```
s1=np.array([1,2,3,4,5])
s2=np.array([6,7,8,9,10])
s3=np.concatenate((s1,s2))
print(s3)
```

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

In [96]:
```
import numpy as np
a1=np.array([[1,2,3],[4,5,6]])
a2=np.array([[7,8,9],[11,12,13]])
a3=np.concatenate((a1,a2),axis=0)
print(a3)
```

```
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [11 12 13]]
```

In [100…
```
import numpy as np
s1=np.array([1,2,3,4,5])
s2=np.array([6,7,8,9,10])
s3=np.stack((s1,s2),axis=1)
print(s3)
```

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

In [102…
```
import numpy as np
s1=np.array([1,2,3,4,5])
s2=np.array([6,7,8,9,10])
s3=np.hstack((s1,s2))
print(s3)
```

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

In [104…
```
import numpy as np
s1=np.array([1,2,3,4,5])
s2=np.array([6,7,8,9,10])
s3=np.dstack((s1,s2))
print(s3)
```

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

In [106…
```
import numpy as np
s1=np.array([1,2,3,4,5])
s2=np.array([6,7,8,9,10])
s3=np.vstack((s1,s2))
print(s3)
```

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

In [110…
```
import numpy as np
arr1=np.array([1,2,3,4,5,6,7,8,9,10,11,12])
arr2=np.split(arr1,4)
print(arr2)
```

```
[array([1, 2, 3]), array([4, 5, 6]), array([7, 8, 9]), array([10, 11, 12])]
```

In [111…
```
import numpy as np
arr1=np.array([1,2,3,4,5,6,7,8,9,10,11,12])
arr2=np.hsplit(arr1,4)
print(arr2)
```

```
[array([1, 2, 3]), array([4, 5, 6]), array([7, 8, 9]), array([10, 11, 12])]
```

In [121…
```
import numpy as np
s1=np.array([[1,2,3],[4,5,6],[7,8,9],[10,11,12]])
```

```
s2=np.vsplit(s1,2)
print(s2)
```

```
[array([[1, 2, 3],
       [4, 5, 6]]), array([[ 7,  8,  9],
       [10, 11, 12]])]
```

In [122…

```
import numpy as np
s1=np.array([[1,2,3],[4,5,6],[7,8,9],[10,11,12]])
s2=np.split(s1,3,axis=1)
print(s2)
```

```
[array([[ 1],
       [ 4],
       [ 7],
       [10]]), array([[ 2],
       [ 5],
       [ 8],
       [11]]), array([[ 3],
       [ 6],
       [ 9],
       [12]])]
```

In [125…

```
import numpy as np
a1=np.arange(20).reshape(4,5)
print(a1)
print(np.vsplit(a1,4))
```

```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]]
[array([[0, 1, 2, 3, 4]]), array([[5, 6, 7, 8, 9]]), array([[10, 11, 12, 13, 1
4]]), array([[15, 16, 17, 18, 19]])]
```

In [120…

```
import numpy as np

a = np.array([4, 5, 2, 7, 8])

# Correct usage of np.split
a1 = np.split(a, [2, 4])  # Split at indices 2 and 4

print(a1)

a2 = np.split(a, 5) # split into 5 equal parts

print(a2)
```

```
[array([4, 5]), array([2, 7]), array([8])]
[array([4]), array([5]), array([2]), array([7]), array([8])]
```

In [ ]:

Pandas:

In [ ]:
```
AIM:Introduction to required python libraries such as Numpy,Pandas,Scipy,Matplot
```

In [ ]:
```
Description:Pandas is a python library used for working with datasets.
It has functions for analysis,cleaning,exploration and manipulating the data.
```

In [2]:
```
import pandas as pd
```

In [6]:
```
a=[1,7,2]
x=pd.Series(a)
print(x)
```

```
0    1
1    7
2    2
dtype: int64
```

In [7]:
```
print(x[0])
```

```
1
```

In [9]:
```
import pandas as pd
a=[1,7,2]
x=pd.Series(a,index=["x","y","z"])
print(x)
```

```
x    1
y    7
z    2
dtype: int64
```

In [10]:
```
import pandas as pd
calories={"day1":320,"day2":230,"day3":430}
x=pd.Series(calories)
print(x)
```

```
day1    320
day2    230
day3    430
dtype: int64
```

In [11]:
```
import pandas as pd
calories={"day1":320,"day2":230,"day3":430}
x=pd.Series(calories,index=["day1","day2"])
print(x)
```

```
day1    320
day2    230
dtype: int64
```

In [12]:
```
sl1 = pd.Series([10, 20, 30, 40])
print(sl1)
print(type(sl1))
```

```
0    10
1    20
2    30
3    40
dtype: int64
```

In [15]:
```
import numpy as np
x=np.array([1,2,3,4,5])
p=pd.Series(x)
print(p)
```

```
0    1
1    2
2    3
3    4
4    5
dtype: int32
```

In [16]:
```
y={1:'a',2:'b',3:'c'}
x=pd.Series(y)
print(y)
```

```
{1: 'a', 2: 'b', 3: 'c'}
```

In [19]:
```
import pandas as pd
x=pd.Series(data=[1,2,3,4],index=['i','ii','iii','iv'])
y=pd.Series(data=['a','b','c','d','e'])
print(x)
print(x.index)
print(x.values)
print(x.shape)
print(x.dtype)
print(x.size)
print(x.ndim)
print(x.nbytes)
print(y)
```

```
i       1
ii      2
iii     3
iv      4
dtype: int64
Index(['i', 'ii', 'iii', 'iv'], dtype='object')
[1 2 3 4]
(4,)
int64
4
1
32
0    a
1    b
2    c
3    d
4    e
dtype: object
```

In [48]:
```
import numpy as np
import pandas as pd
```

In [47]:
```
a=pd.Series(['java','c','c++',np.nan])
a.map({'java':'core'})
```

Out[47]:
```
0    core
1     NaN
2     NaN
3     NaN
dtype: object
```

In [26]:
```
import pandas as pd
import numpy as np
a=pd.Series(['java','c','c++',np.nan])
a.map({'java':'core'})
a.map('i like {}'.format,na_action='ignore')
```

Out[26]:
```
0    i like java
1       i like c
2     i like c++
3            NaN
dtype: object
```

In [28]:
```
a=pd.Series(['java','c','c++',np.nan])
a.map({'java':'core','c':'ANSII  c'})
```

Out[28]:
```
0       core
1    ANSII c
2        NaN
3        NaN
dtype: object
```

In [29]:
```
a.map('i like {}'.format,na_action='ignore')
```

Out[29]:
```
0    i like java
1       i like c
2     i like c++
3            NaN
dtype: object
```

In [37]:
```
x=np.array(['ram','hari','sita','krishna','radha'])
y=pd.Series(x)
print("Sorting the string array\n",y.sort_values())
print("In Desending order\n",y.sort_values(ascending=False))
```

```
Sorting the string array
 1       hari
3    krishna
4      radha
0        ram
2       sita
dtype: object
In Desending order
 2       sita
0        ram
4      radha
3    krishna
1       hari
dtype: object
```

In [41]:
```
import pandas as pd
data=[1,2,3,4,5]
df=pd.DataFrame(data)
print(df)
```

```
   0
0  1
1  2
2  3
3  4
4  5
```

In [53]:
```
data=[['Alex',10],['Bob',12],['clarke',13]]
df=pd.DataFrame(data,columns=['Name','Age'])
print(df)
```

```
     Name  Age
0    Alex   10
1     Bob   12
2  clarke   13
```

In [54]:
```
df=pd.DataFrame(data,columns=['Name','Age'],
                dtype=float)
print(df)
```

```
     Name   Age
0    Alex  10.0
1     Bob  12.0
2  clarke  13.0
C:\Users\HP\AppData\Local\Temp\ipykernel_3688\4138737743.py:1:  FutureWarning:  Cou
ld not cast to float64, falling back to object. This behavior is deprecated. In a
future version, when a dtype is passed to 'DataFrame', either all columns will be
cast to that dtype, or a TypeError will be raised.
  df=pd.DataFrame(data,columns=['Name','Age'],
```

In [45]:
```
#dict of  lists
data={'Name':['Tom','jack','steve','Ricky'],'Age':[12,23,13,24]}
df=pd.DataFrame(data,index=['Rank1','Rank2','Rank3','Rank4'])
print(df)
```

```
        Name  Age
Rank1    Tom   12
Rank2   jack   23
Rank3  steve   13
```

In [49]:
```
#multiple dict
data=[{'a':1,'b':2},{'a':10,'b':20,'c':30}]
df=pd.DataFrame(data,index=['first','second'])
print(df)
```

```
          a   b     c
first     1   2   NaN
second   10  20  30.0
```

In [60]:
```
df=pd.DataFrame(data,columns=['Name','Age'])
df['Age']=df['Age'].astype(float)
print(df)
```

```
     Name   Age
0    Alex  10.0
1     Bob  12.0
2  clarke  13.0
```

In [76]:
```
#create a dataframedict of series
import pandas as pd

d = {
    'one': pd.Series([1, 2, 3], index=['a', 'b', 'c']),
    'two': pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])
}

df = pd.DataFrame(d)

print(df)
```

```
   one  two
a  1.0    1
b  2.0    2
c  3.0    3
d  NaN    4
```

In [77]:
```
df['three']=pd.Series([10,20,30,40],index=['a','b','c','d'])
print(df)
df['four']=df['one']+df['three']
print(df)
df['sub']=df['one']+df['three']
print(df)
#deleting col three
del df['three']
print(df)
```

```
   one  two  three
a  1.0    1     10
b  2.0    2     20
c  3.0    3     30
d  NaN    4     40
   one  two  three  four
a  1.0    1     10  11.0
b  2.0    2     20  22.0
c  3.0    3     30  33.0
d  NaN    4     40   NaN
   one two three four       sub
a  1.0    1     10  11.0  11.0
b  2.0    2     20  22.0  22.0
c  3.0    3     30  33.0  33.0
d NaN     4     40   NaN   NaN
   one two four      sub
a  1.0    1  11.0   11.0
b  2.0    2  22.0   22.0
c  3.0    3  33.0   33.0
d  NaN    4   NaN    NaN
```

In [78]:
```
df1=pd.DataFrame([[1,2],[3,4]],columns=['a','b'])
df1.drop(0,inplace=False)#rowise deletion
print(df1)
```

```
   a  b
0  1  2
1  3  4
```

In [80]:
```
df1.drop(columns=['a'],inplace=True)
print(df1)
```

```
   b
0  2
1  4
```

In [81]:
```
df2=pd.DataFrame([[5,6],[7,8]],columns=['a','b'])
print(df2)
df2['c']=pd.Series([10,20])
print(df2)
df2.drop(columns=['a','b'],inplace=True)#colwisedeletion
print(df2)
```

```
   a  b
0  5  6
1  7  8
   a  b   c
0  5  6  10
1  7  8  20
    c
0  10
1  20
```

In [11]:
```
#create a dataframe with columns Name,Age,University,Percentage,for 5 students
data={'Name':['Tom','jack','steve','Ricky','jio'],'Age':[12,23,13,24,23],'univer
df_students = pd.DataFrame(data,index=['s1','s2','s3','s4','s5'])
print(df_students)
```

```
     Name  Age university  Percentage
s1    Tom   12     aditya        85.5
s2   jack   23        vst        90.0
s3  steve   13      JNTUK        78.5
s4  Ricky   24       srkr        88.0
s5    jio   23       svec        92.5
```

In [12]:
```
print(df_students[0:3])
```

```
     Name  Age university  Percentage
s1    Tom   12     aditya        85.5
s2   jack   23        vst        90.0
s3  steve   13      JNTUK        78.5
```

In [13]:
```
print(df_students[['Name','Age']])
```

```
     Name  Age
s1    Tom   12
s2   jack   23
s3  steve   13
s4  Ricky   24
s5    jio   23
```

In [15]:
```
print(df_students.loc['s1':'s3', ['Name', 'university']])
print(df_students.iloc[0,0:3])
```

```
     Name university
s1    Tom     aditya
s2   jack        vst
s3  steve      JNTUK
Name             Tom
Age               12
university    aditya
Name: s1, dtype: object
```

In [134…
```
print(df_students.loc[(df_students['Name']=='Tom')&(df_students['university']=='
```

```
  Name  Age university  Percentage
s1  Tom   12     aditya        85.5
```

In [ ]:
```

```

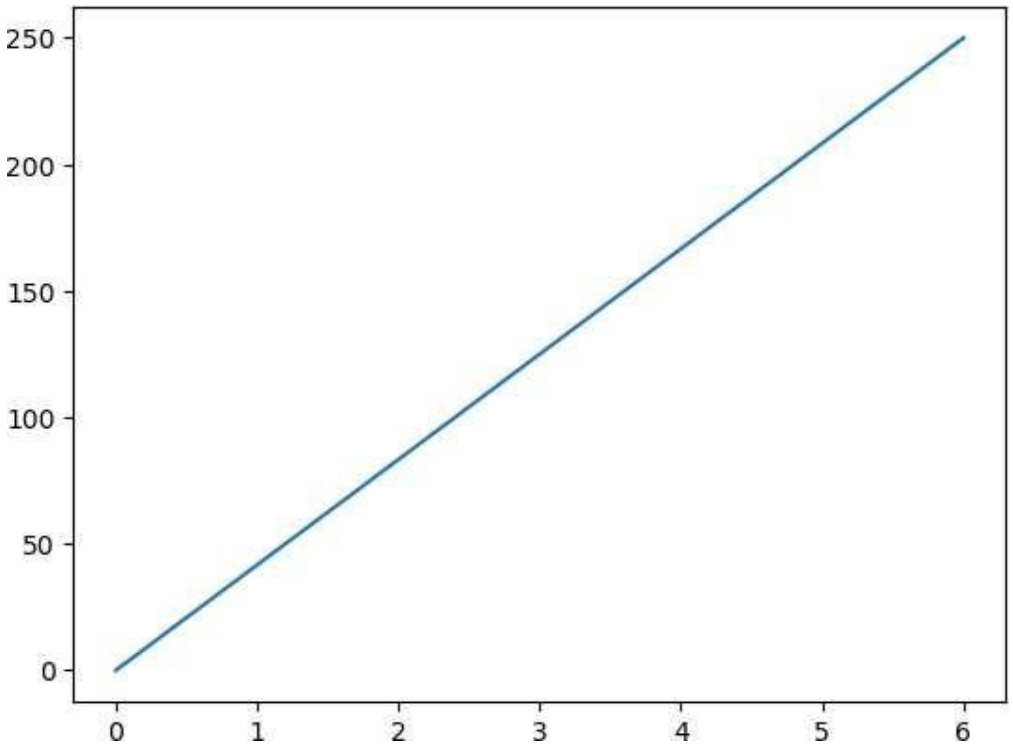
Matplot:

In [ ]:
```
AIM:Introduction to required python libraries such as Numpy,Pandas,Scipy,Matplot
```

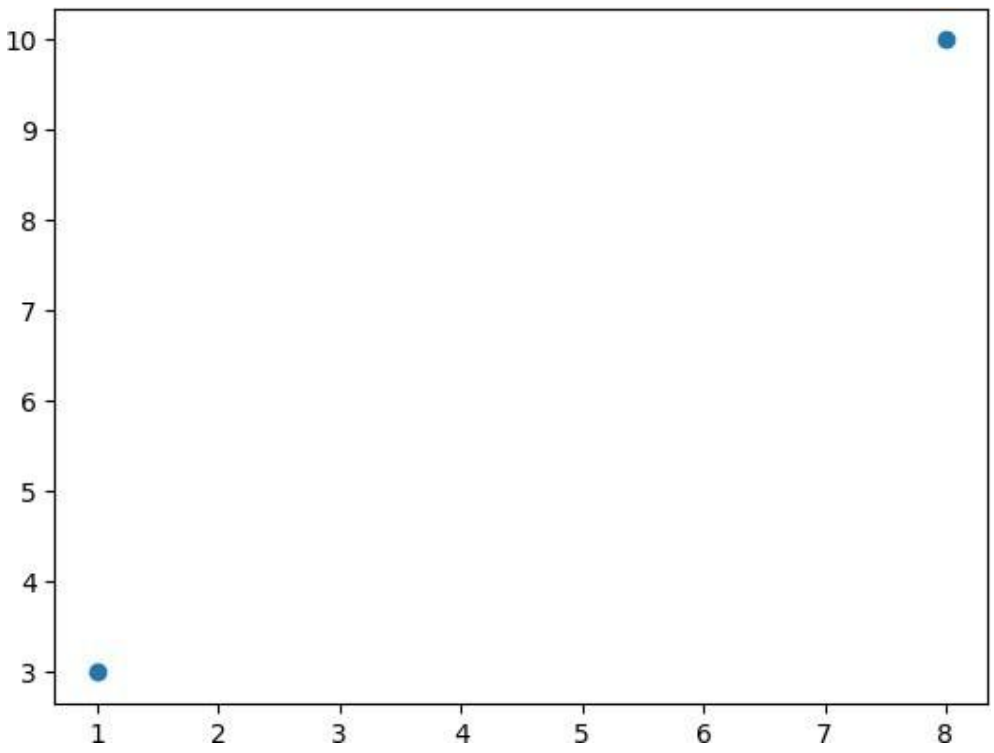
In [ ]:
```
Description:Matplotlib is a lowlevel graph plotting library in python serves as
Matplotlib library is open source.
```

In [3]:
```
import matplotlib.pyplot as plt
import numpy as np
```

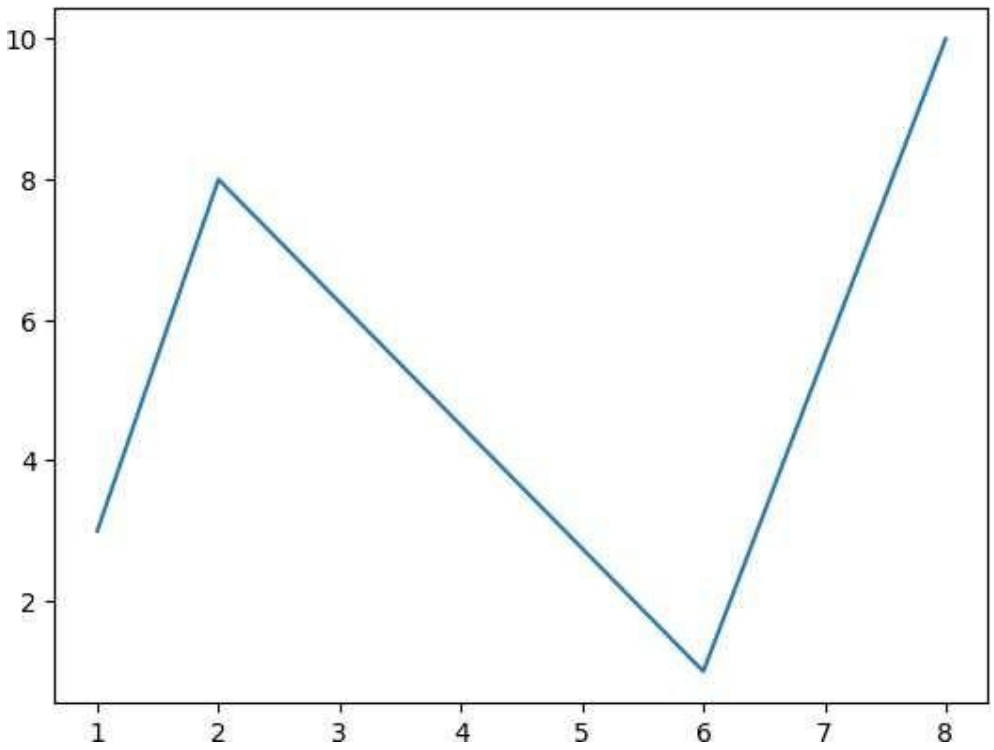
In [5]:
```
xpoints =np.array([0,6])
ypoints =np.array([0,250])
plt.plot(xpoints,ypoints)
plt.show()
```



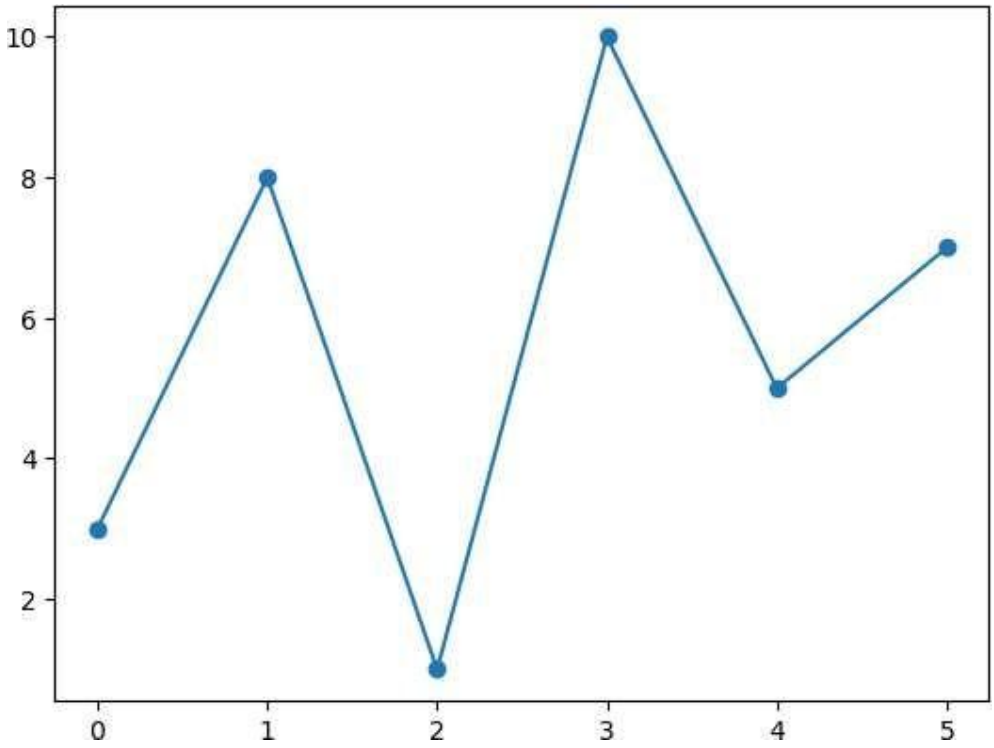
In [11]:
```
xpoints =np.array([1,8])
ypoints =np.array([3,10])
plt.plot(xpoints,ypoints,'o')
plt.show()
```



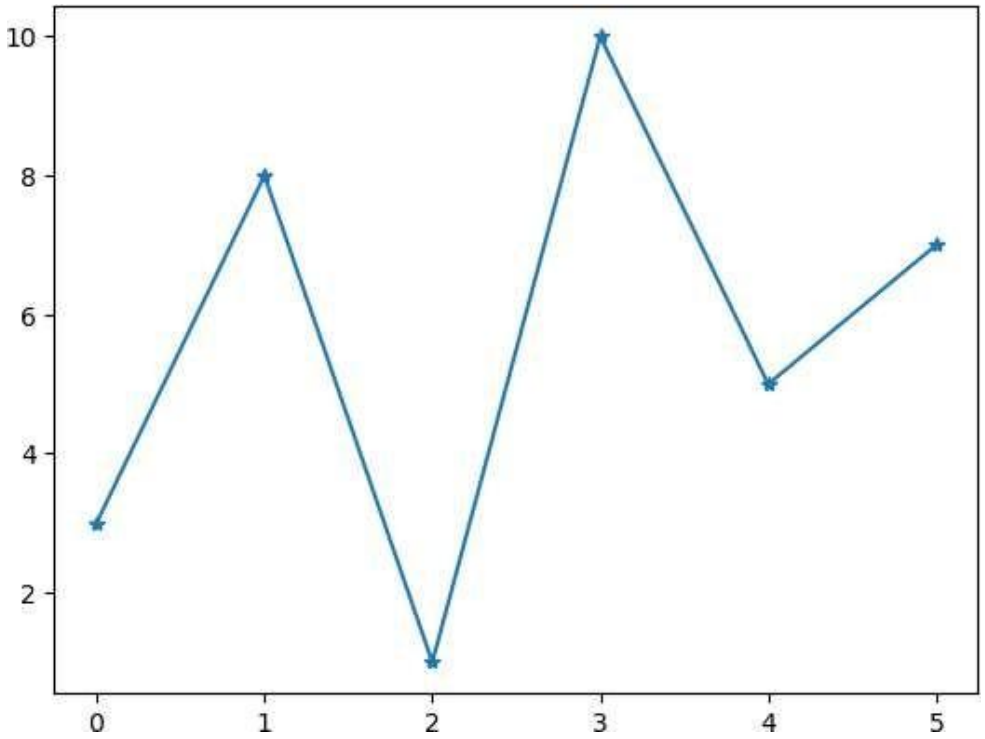
In [13]:
```
xpoints =np.array([1,2,6,8])
ypoints =np.array([3,8,1,10])
plt.plot(xpoints,ypoints)
plt.show()
```



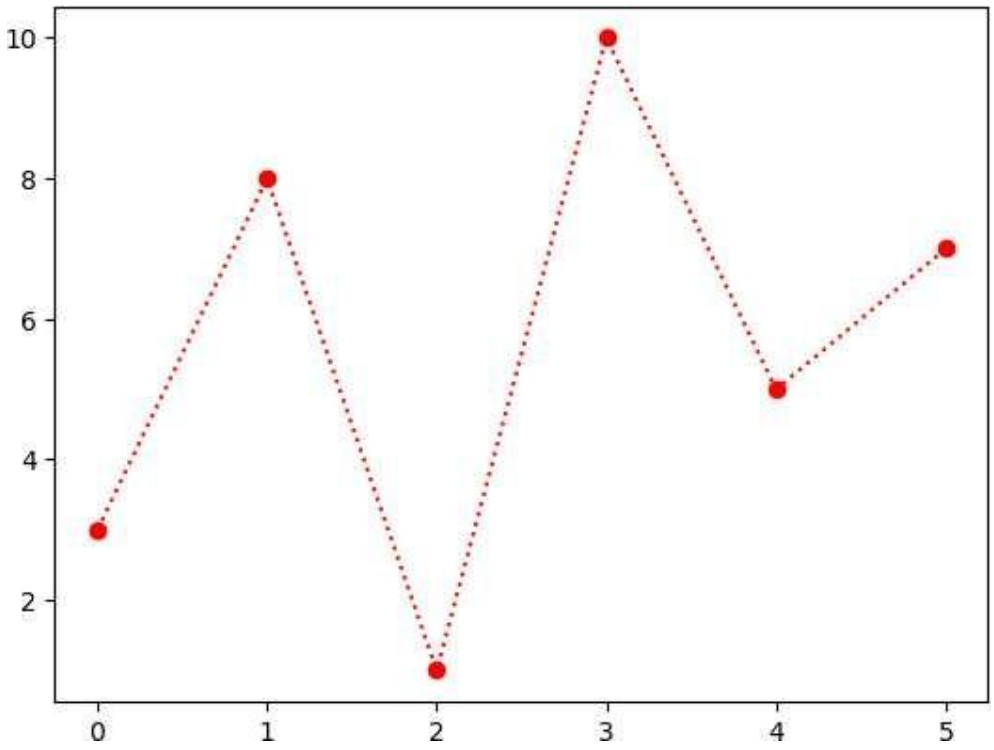
In [21]:
```
import matplotlib.pyplot as plt
import numpy as np
ypoints =np.array([3,8,1,10,5,7])
plt.plot(ypoints,marker='o')
plt.show()
```



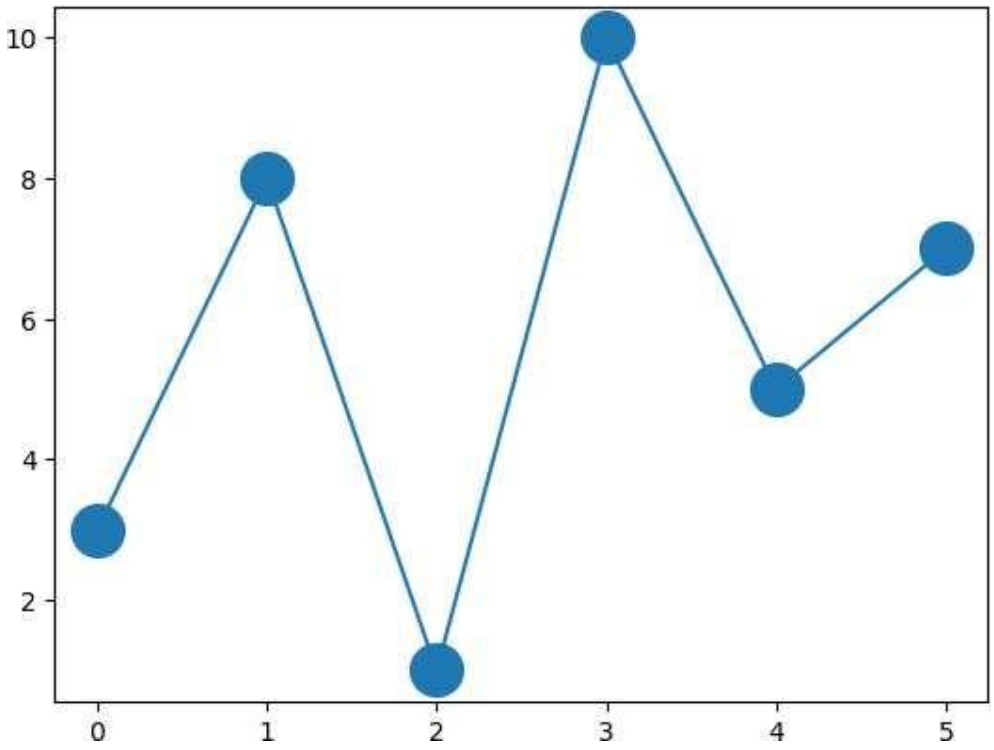
In [23]:
```
import matplotlib.pyplot as plt
import numpy as np
ypoints =np.array([3,8,1,10,5,7])
plt.plot(ypoints,marker='*')
plt.show()
```



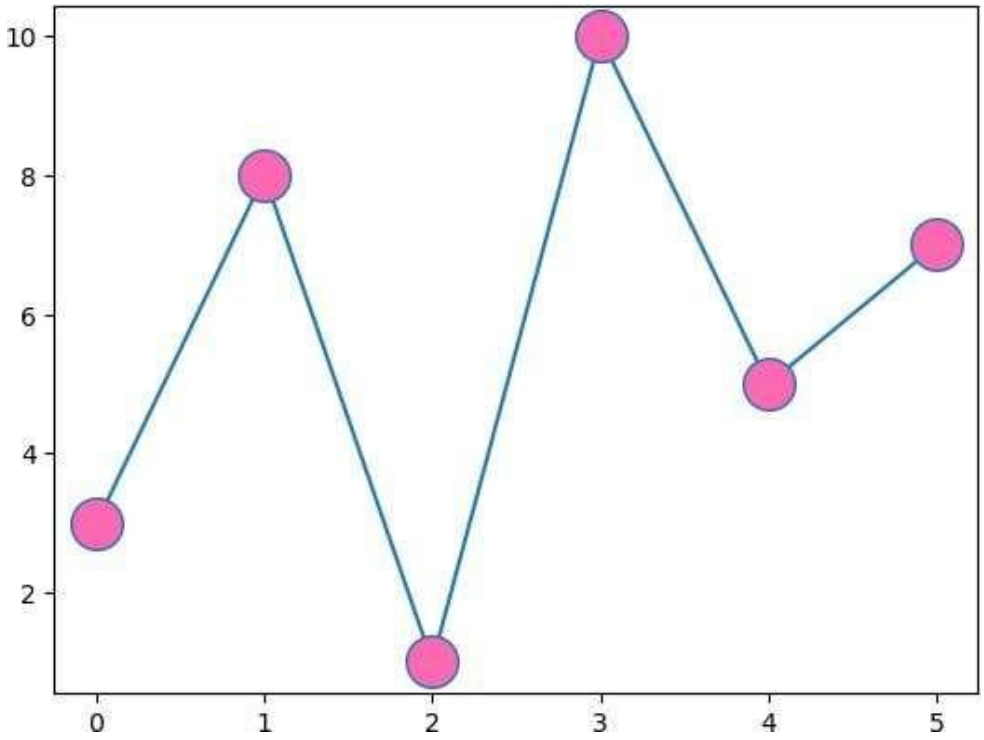
In [27]:
```
ypoints =np.array([3,8,1,10,5,7])
plt.plot(ypoints,'o:r')
plt.show()
```



In [61]:
```
ypoints  =np.array([3,8,1,10,5,7])
plt.plot(ypoints,marker='o',ms=20)
plt.show()
```

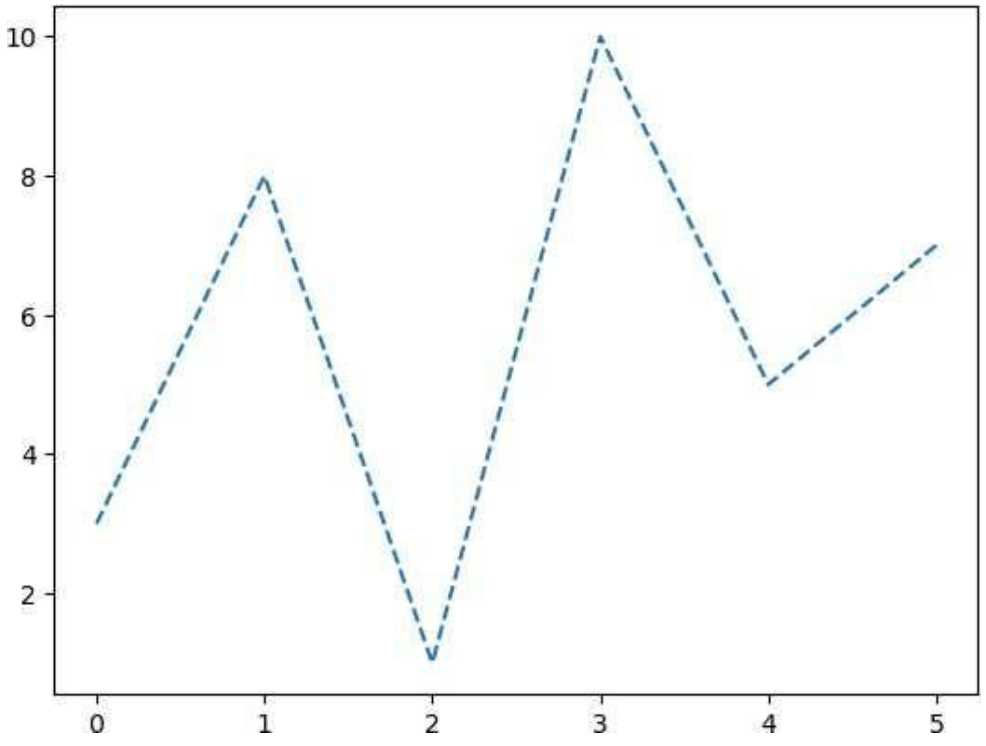


In [63]:
```
ypoints =np.array([3,8,1,10,5,7])
plt.plot(ypoints,marker='o',ms=20,mfc='hotpink')
plt.show()
```

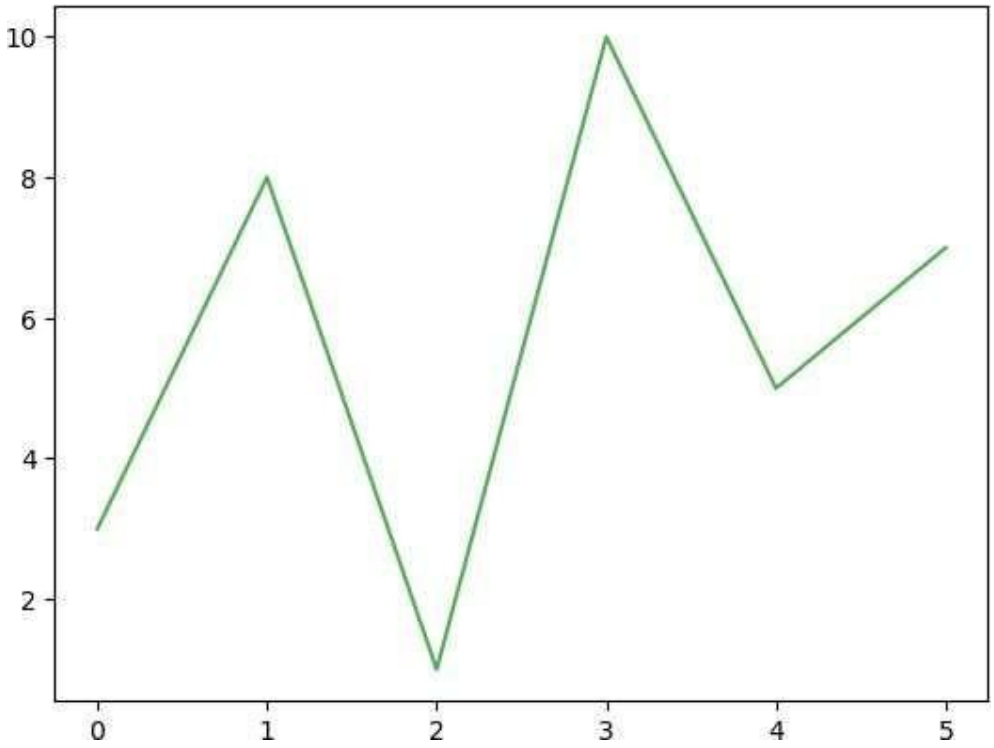


In [73]:
```
import matplotlib.pyplot as plt
import numpy as np
ypoints =np.array([3,8,1,10,5,7])
```

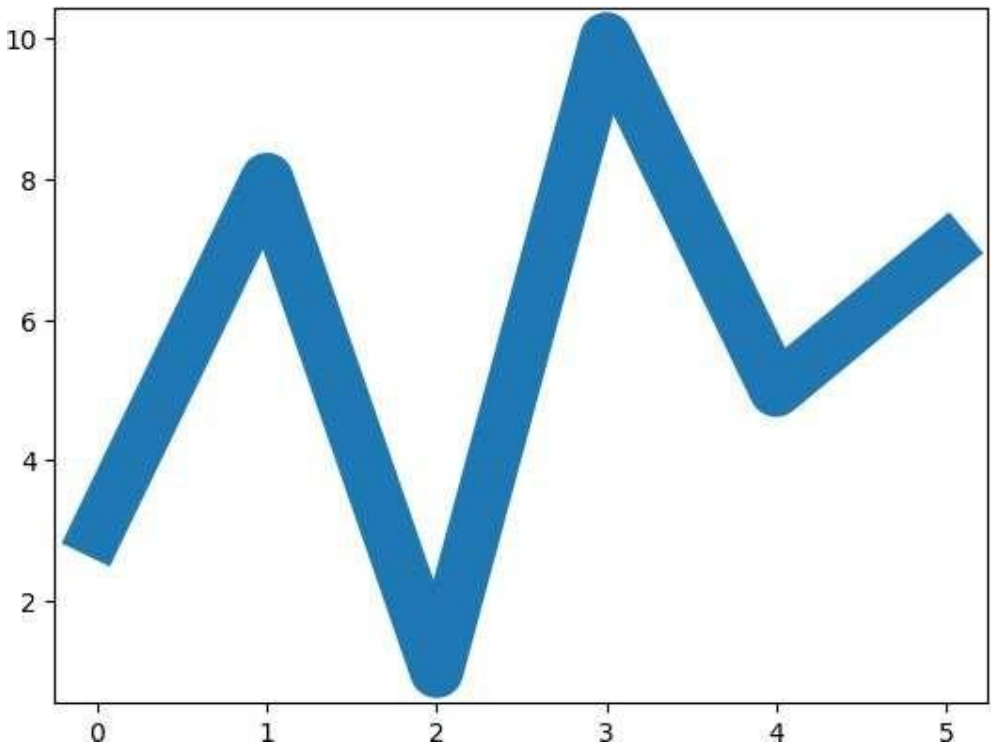
```
plt.plot(ypoints,linestyle='dashed')
plt.show()
```



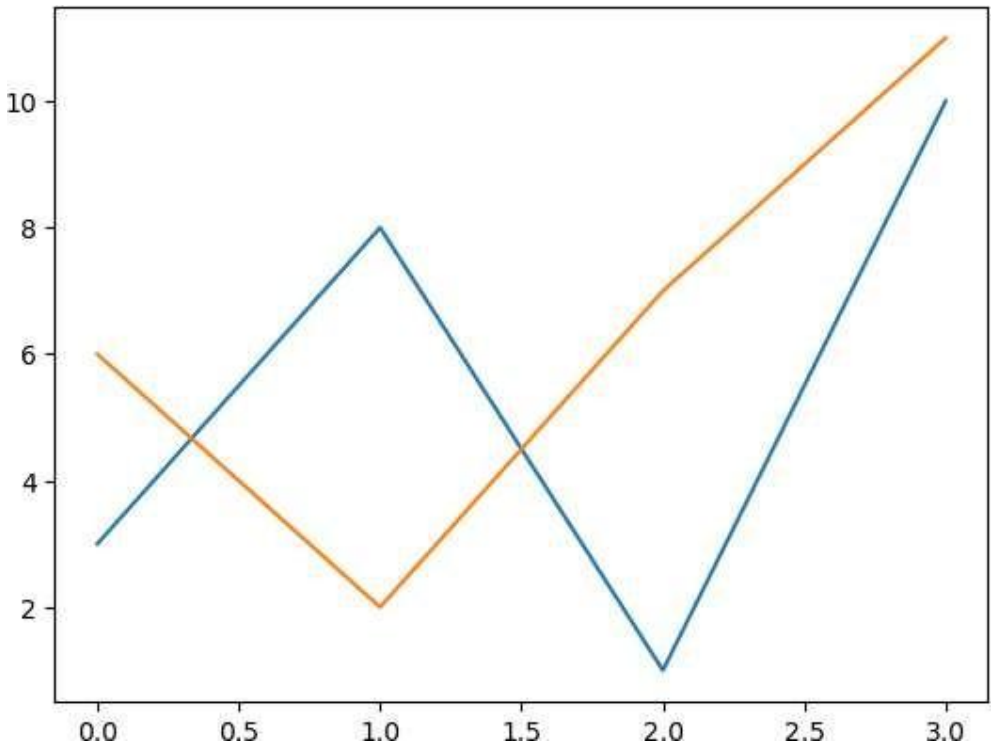
In [77]:
```
import matplotlib.pyplot as plt
import numpy as np
ypoints =np.array([3,8,1,10,5,7])
plt.plot(ypoints,c = '#4CAF50')
plt.show()
```



In [79]:
```
import matplotlib.pyplot as plt
import numpy as np
ypoints  =np.array([3,8,1,10,5,7])
plt.plot(ypoints,linewidth='20.5')
plt.show()
```

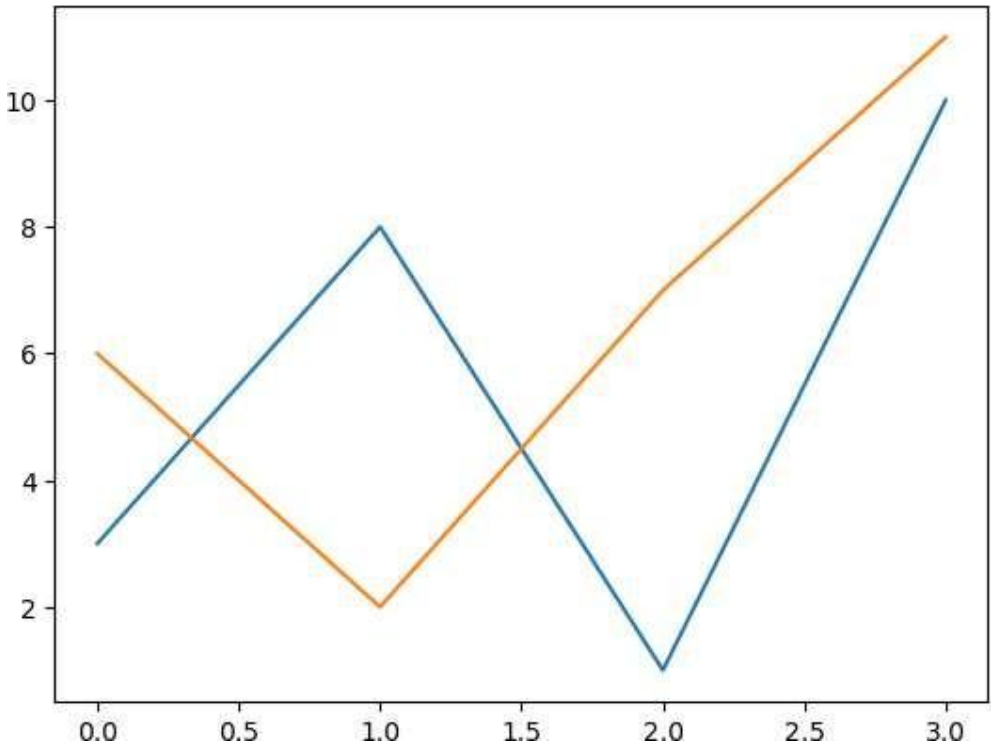


In [83]:
```
y1=np.array([3,8,1,10])
y2=np.array([6,2,7,11])
plt.plot(y1)
plt.plot(y2)
plt.show()
```

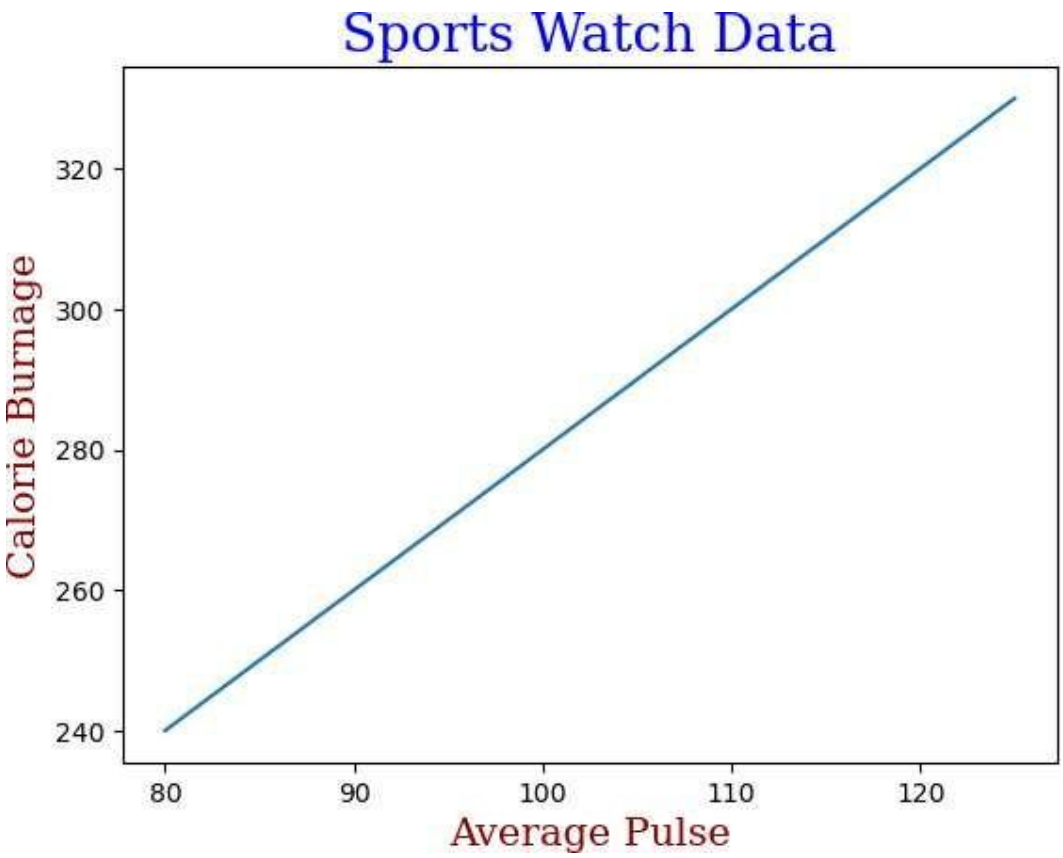


In [85]:
```
import matplotlib.pyplot as plt
import numpy as np
```

```
x1 = np.array([0, 1, 2, 3])
y1 = np.array([3, 8, 1, 10])
x2 = np.array([0, 1, 2, 3])
y2 = np.array([6, 2, 7, 11])

plt.plot(x1, y1, x2, y2)
plt.show()
```

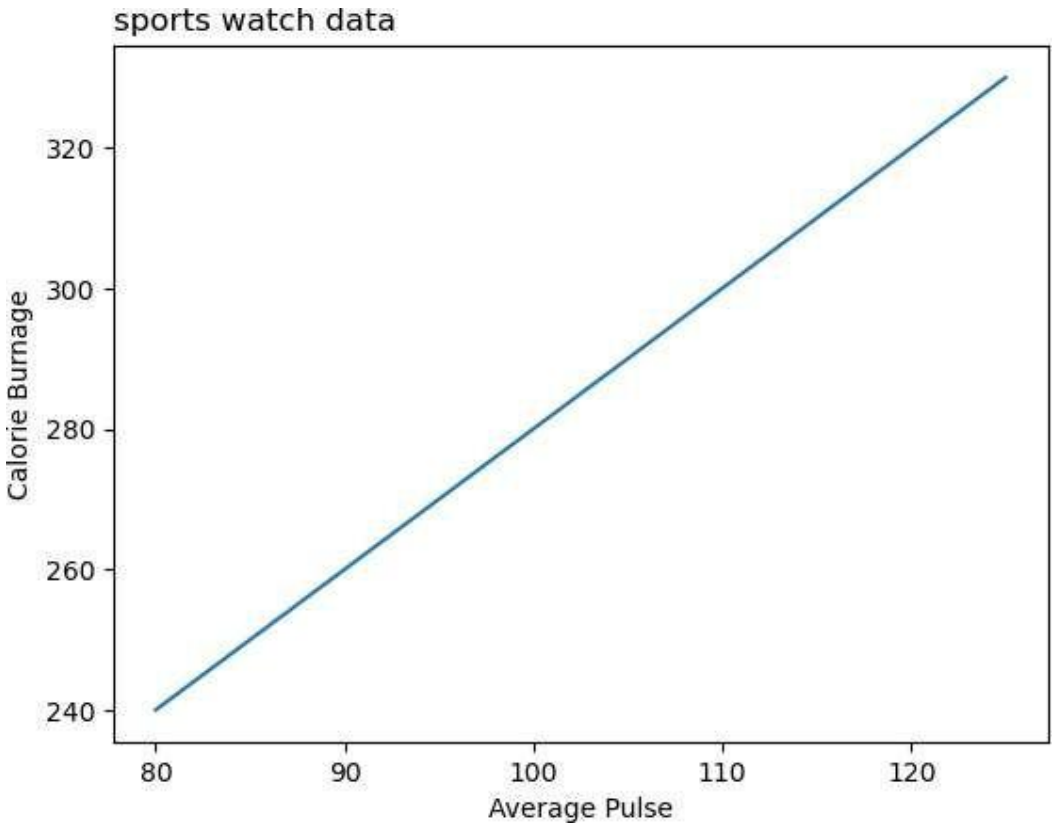


In [99]:
```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

font1 = {'family':'serif','color':'blue','size':20}
font2 = {'family':'serif','color':'darkred','size':15}

plt.title("Sports Watch Data", fontdict = font1)
plt.xlabel("Average Pulse", fontdict = font2)
plt.ylabel("Calorie Burnage", fontdict = font2)

plt.plot(x, y)
plt.show()
```



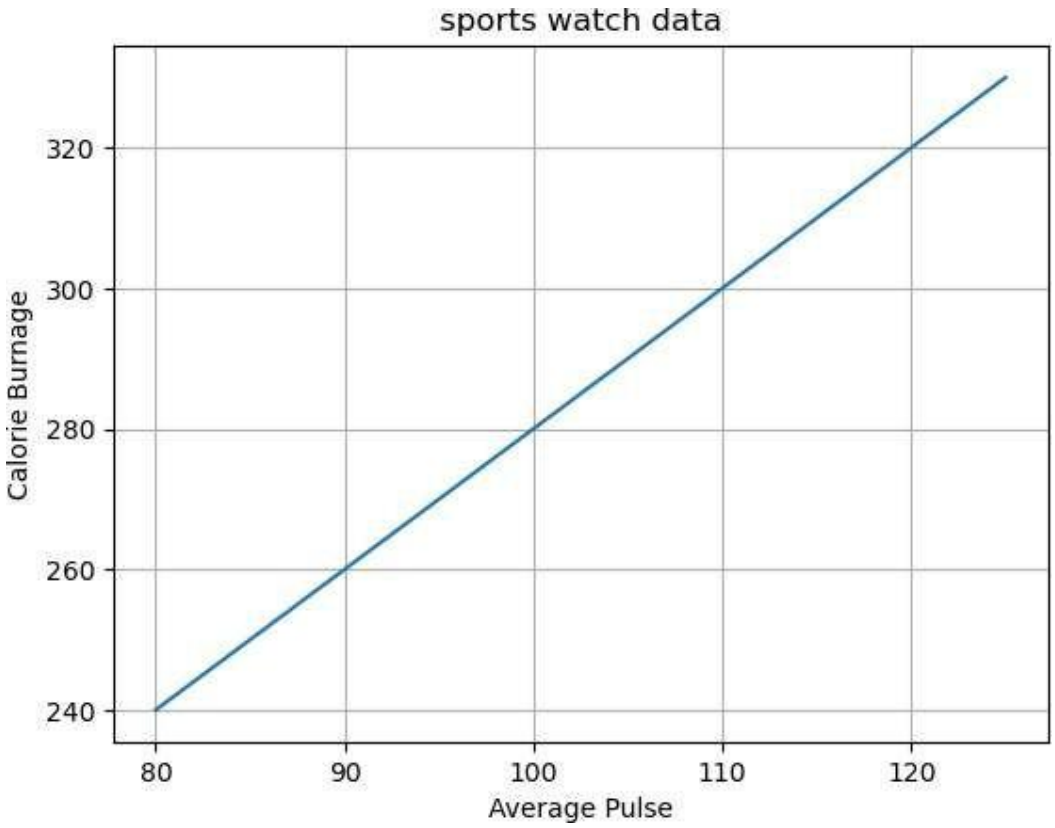
In [101…

```python
import numpy as np
import matplotlib.pyplot as plt

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.plot(x, y)
plt.title("sports watch data",loc='left')
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.show()
```



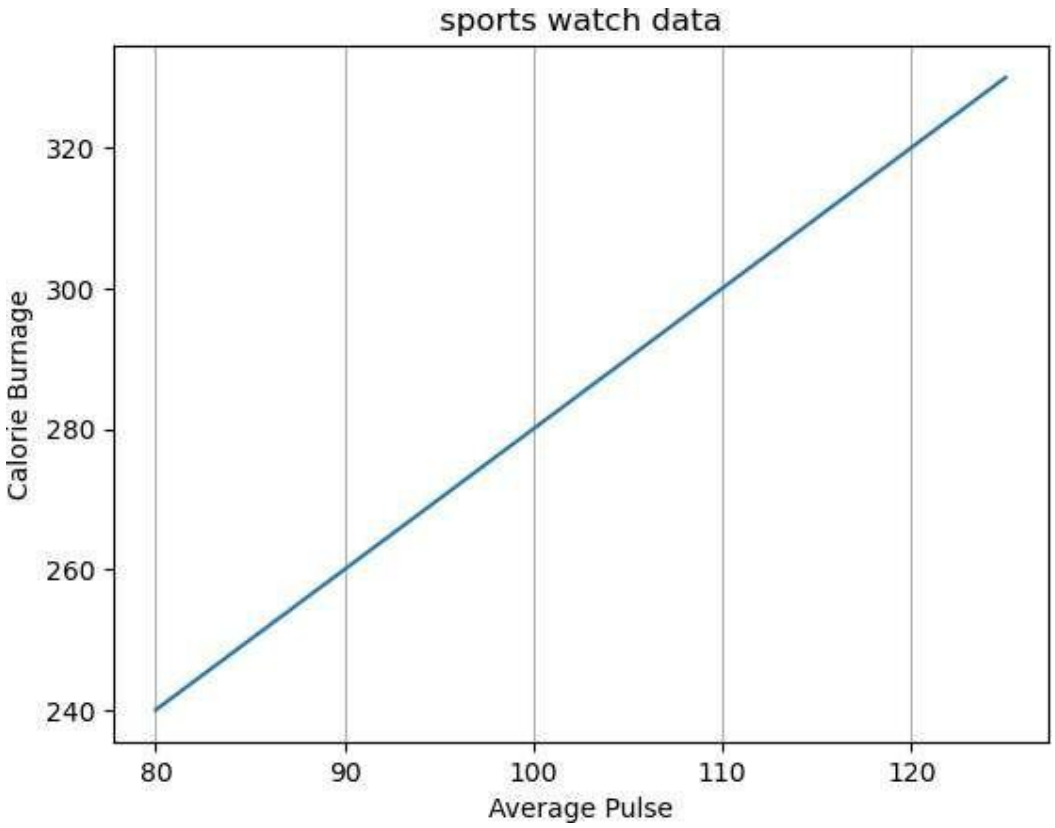
In [103…

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

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.plot(x, y)
plt.title("sports watch data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")
plt.grid()

plt.show()
```



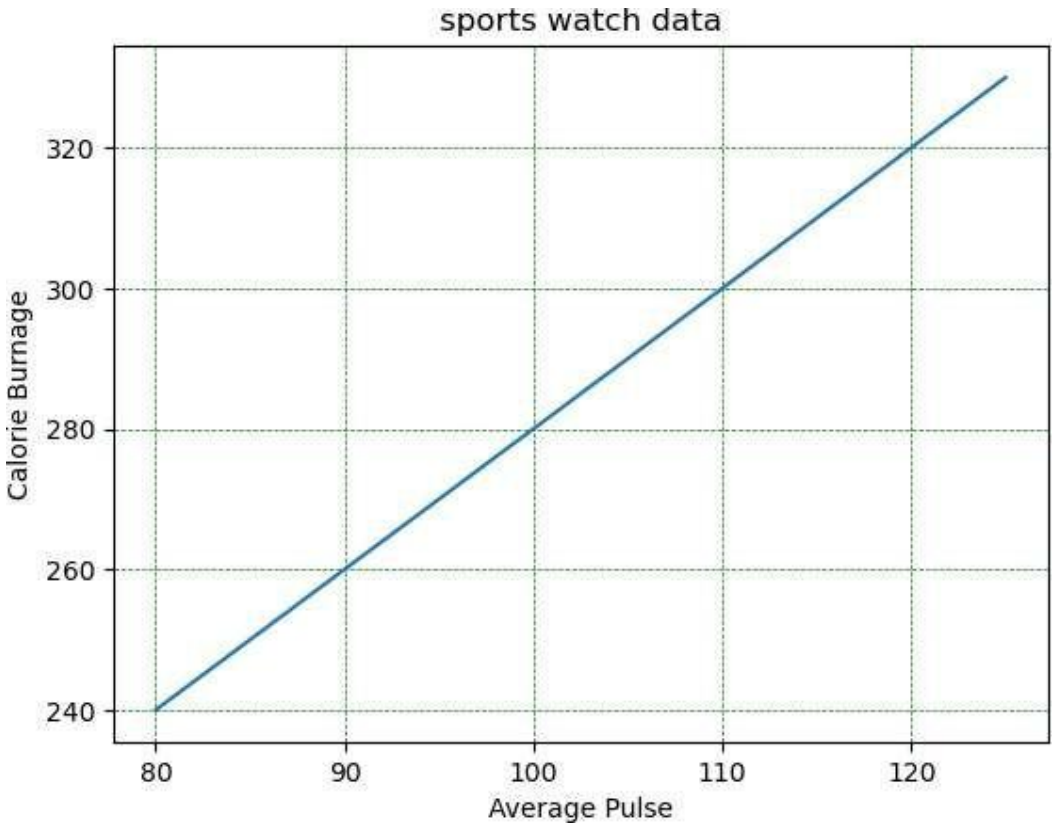
In [107…

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

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.plot(x, y)
plt.title("sports watch data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")
plt.grid(axis = 'x')

plt.show()
```



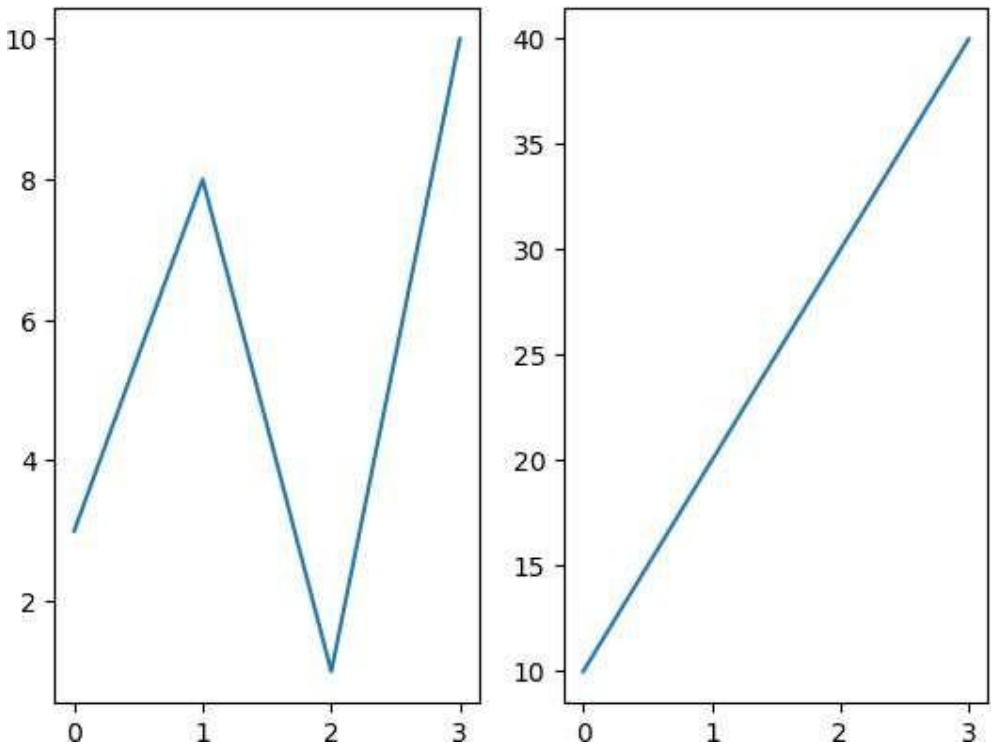
In [113…

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

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.plot(x, y)
plt.title("sports watch data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")
plt.grid(color = 'green', linestyle = '--', linewidth = 0.5)
plt.show()
```



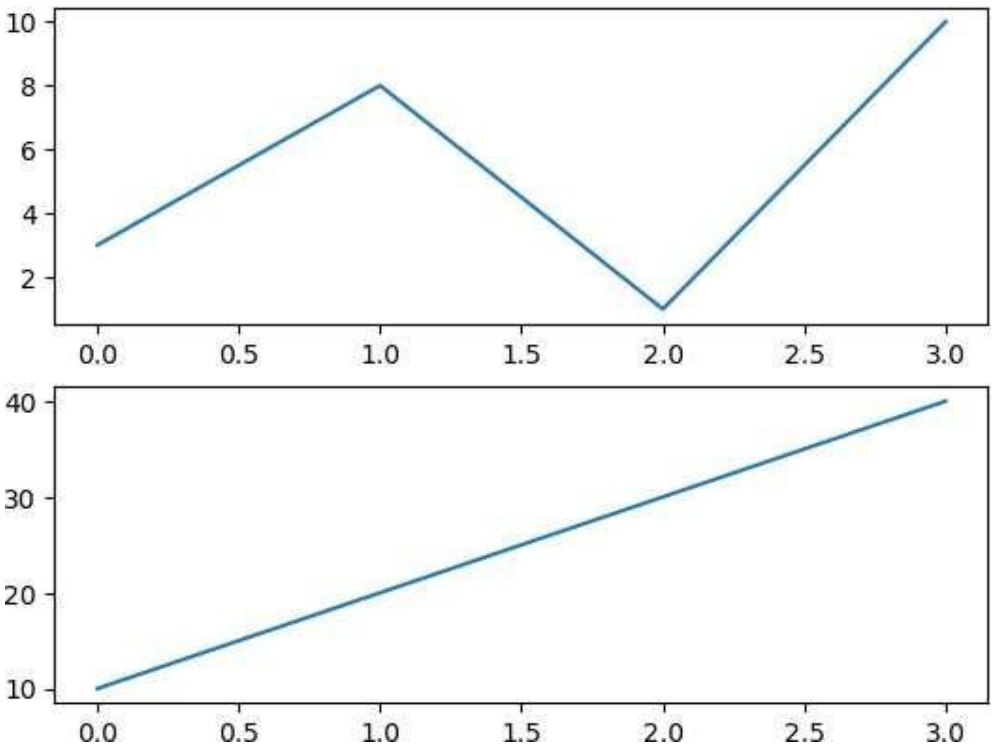
In [115…

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

#plot 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(1, 2, 1)
plt.plot(x,y)

#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(1, 2, 2)
plt.plot(x,y)

plt.show()
```



In [117…

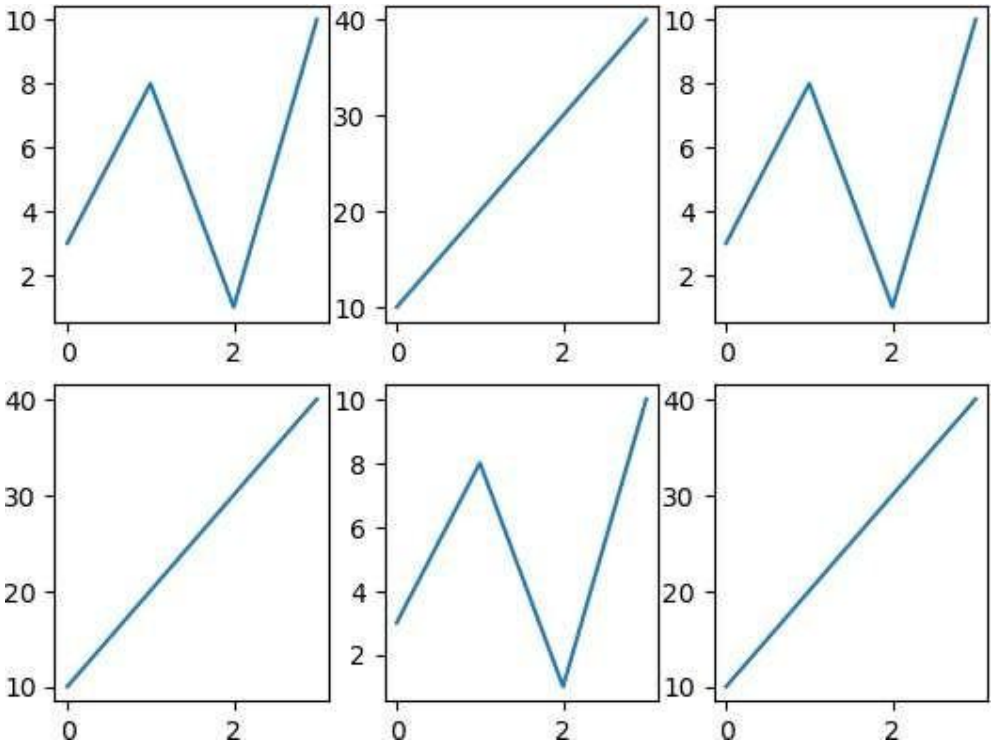
```
import matplotlib.pyplot as plt
import numpy as np

#plot 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 1, 1)
plt.plot(x,y)

#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 1, 2)
plt.plot(x,y)

plt.show()
```



In [119…

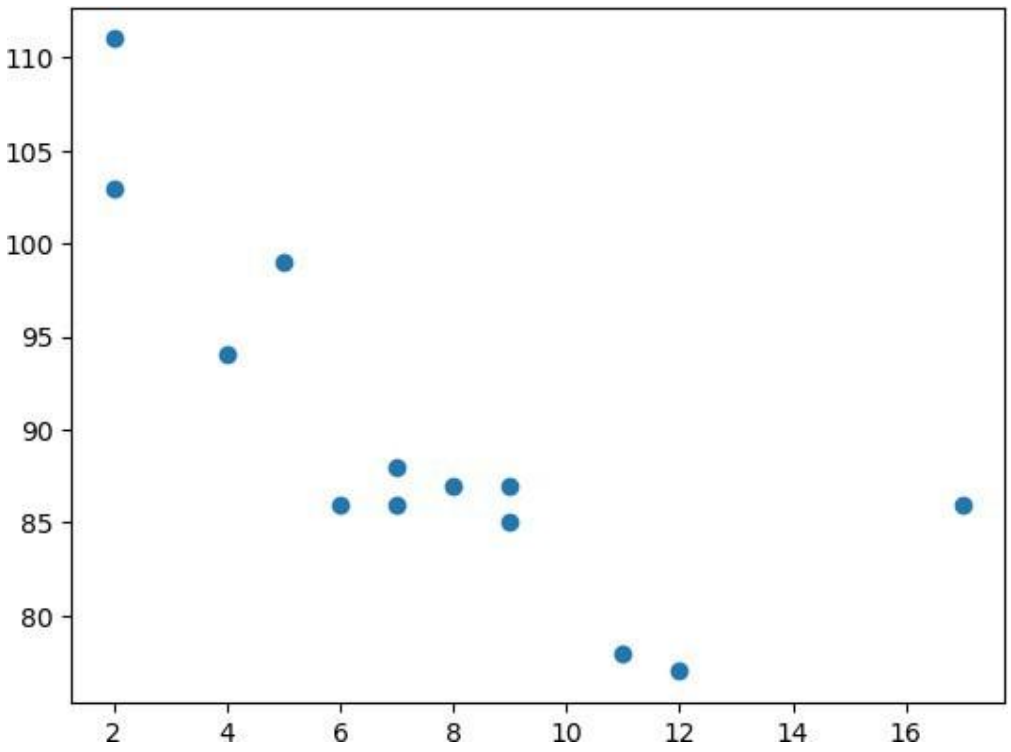
```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 3, 1)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 3, 2)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 3, 3)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 3, 4)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 3, 5)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
```

```
y = np.array([10,  20,  30,  40])

plt.subplot(2, 3,  6)
plt.plot(x,y)

plt.show()
```



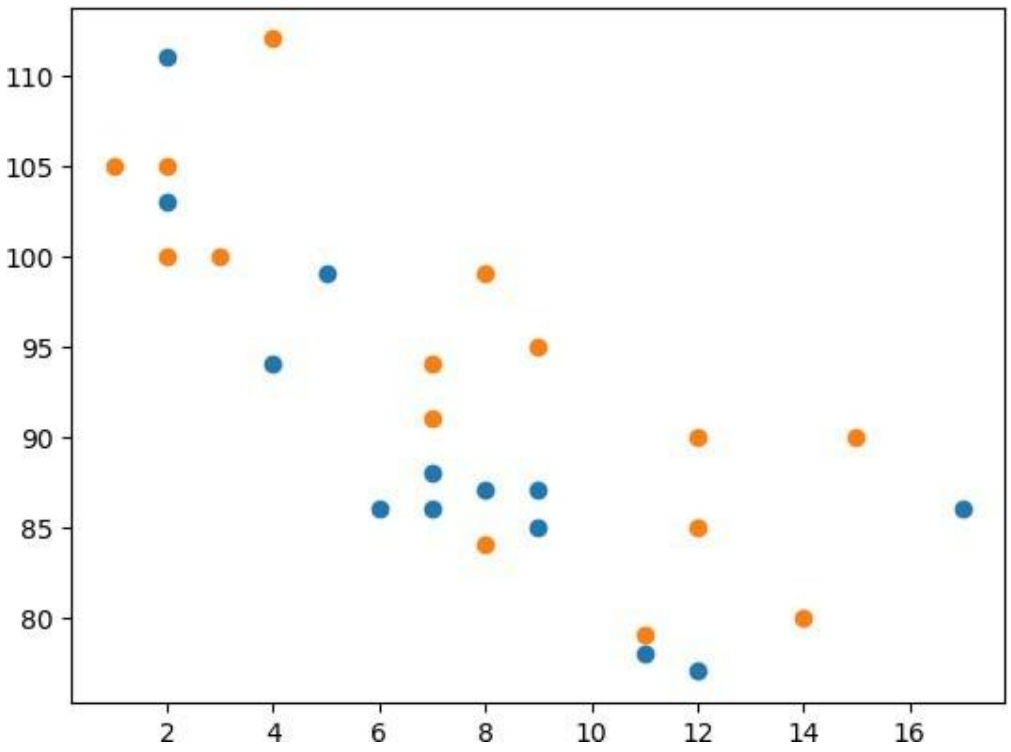
In [123…

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

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y =  np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])

plt.scatter(x, y)
plt.show()
```



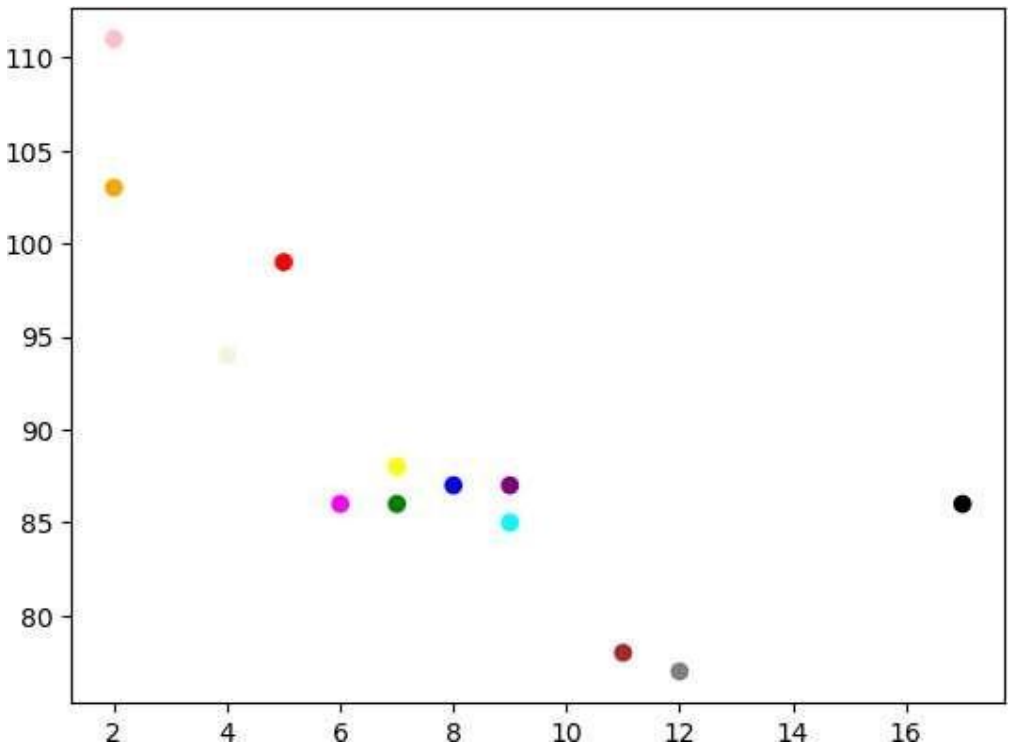
In [125…

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

#day one, the age and speed of 13 cars:
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
plt.scatter(x, y)

#day two, the age and speed of 15 cars:
x = np.array([2,2,8,1,15,8,12,9,7,3,11,4,7,14,12])
y = np.array([100,105,84,105,90,99,90,95,94,100,79,112,91,80,85])
plt.scatter(x, y)

plt.show()
```



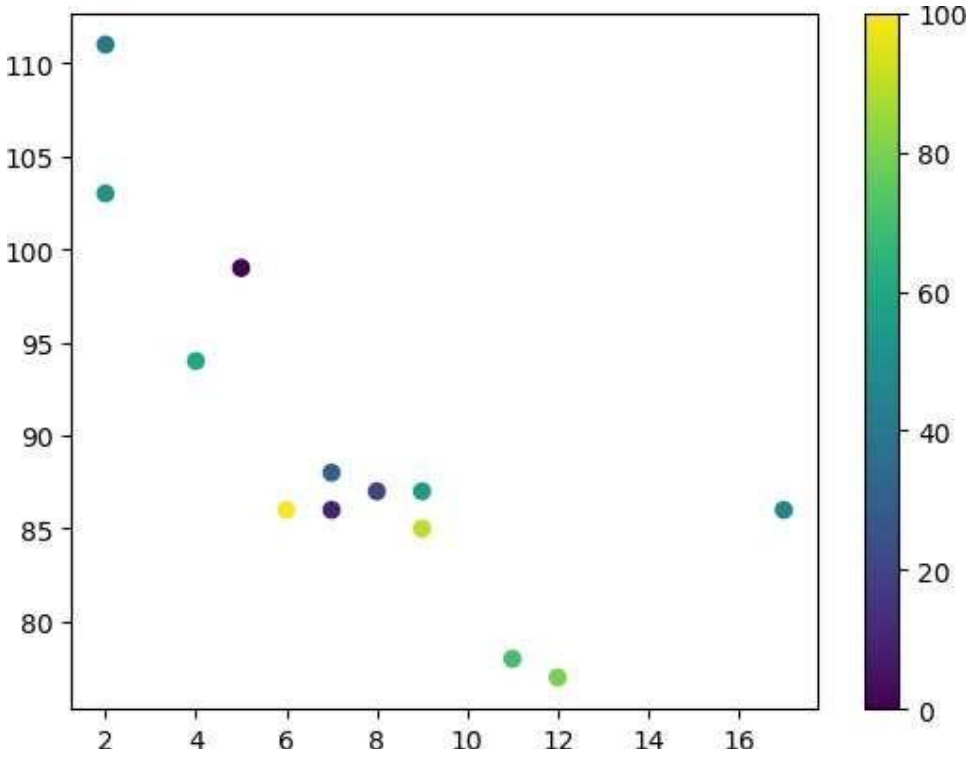
In [129…

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

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
colors = np.array(["red","green","blue","yellow","pink","black","orange","purple

plt.scatter(x, y, c=colors)

plt.show()
```



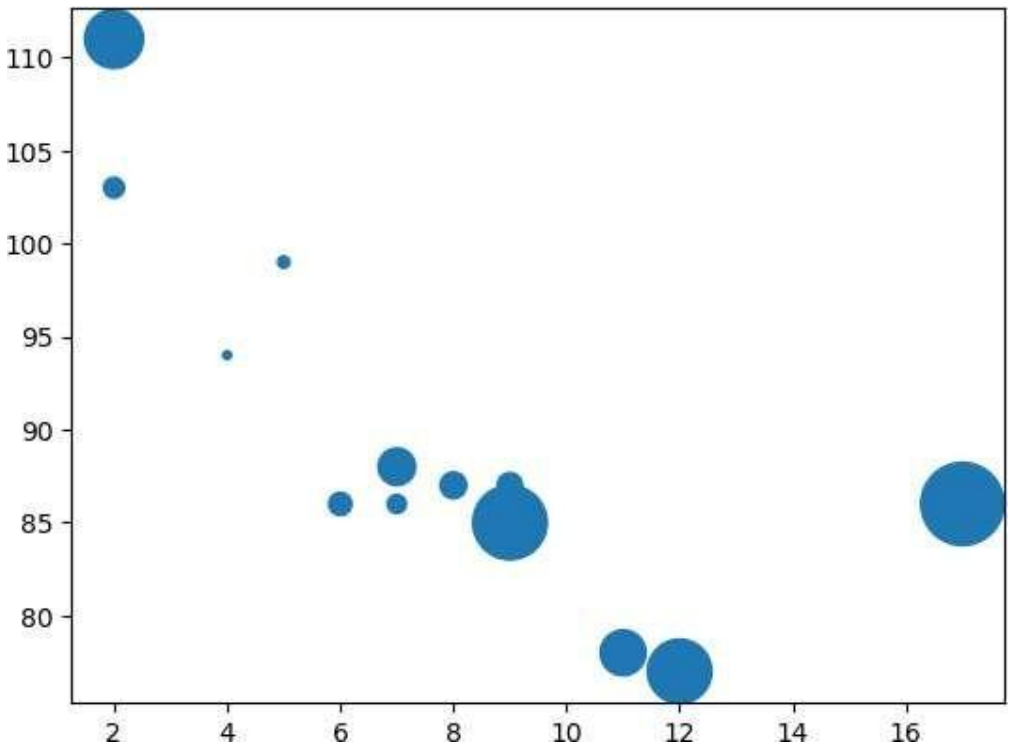
In [139…

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

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
colors = np.array([0, 10, 20, 30, 40, 45, 50, 55, 60, 70, 80, 90, 100])

plt.scatter(x, y, c=colors, cmap='viridis')
plt.colorbar()
plt.show()
```



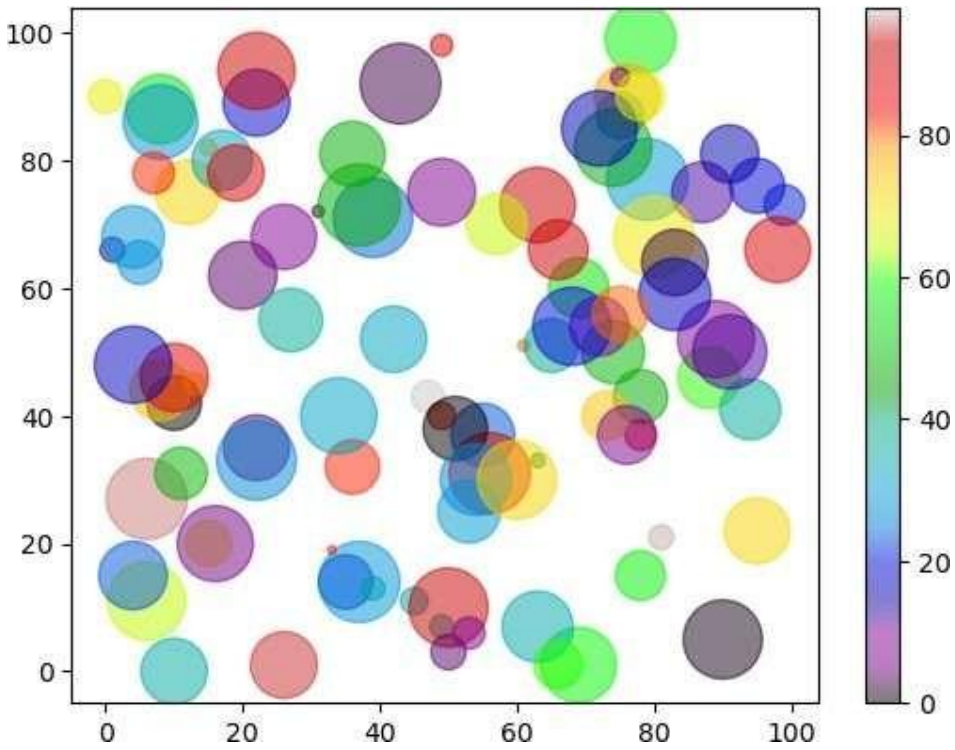
In [141…

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

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
sizes = np.array([20,50,100,200,500,1000,60,90,10,300,600,800,75])

plt.scatter(x, y, s=sizes)

plt.show()
```



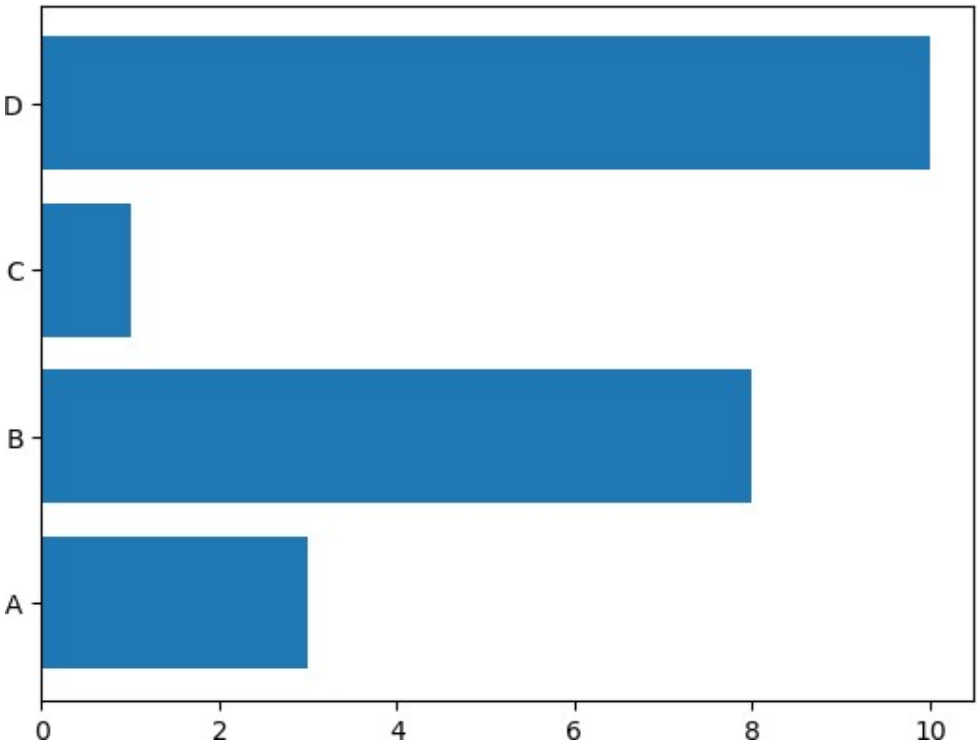
In [143…

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

x = np.random.randint(100, size=(100))
y = np.random.randint(100, size=(100))
colors = np.random.randint(100, size=(100))
sizes = 10 * np.random.randint(100, size=(100))

plt.scatter(x, y, c=colors, s=sizes, alpha=0.5, cmap='nipy_spectral')

plt.colorbar()

plt.show()
```



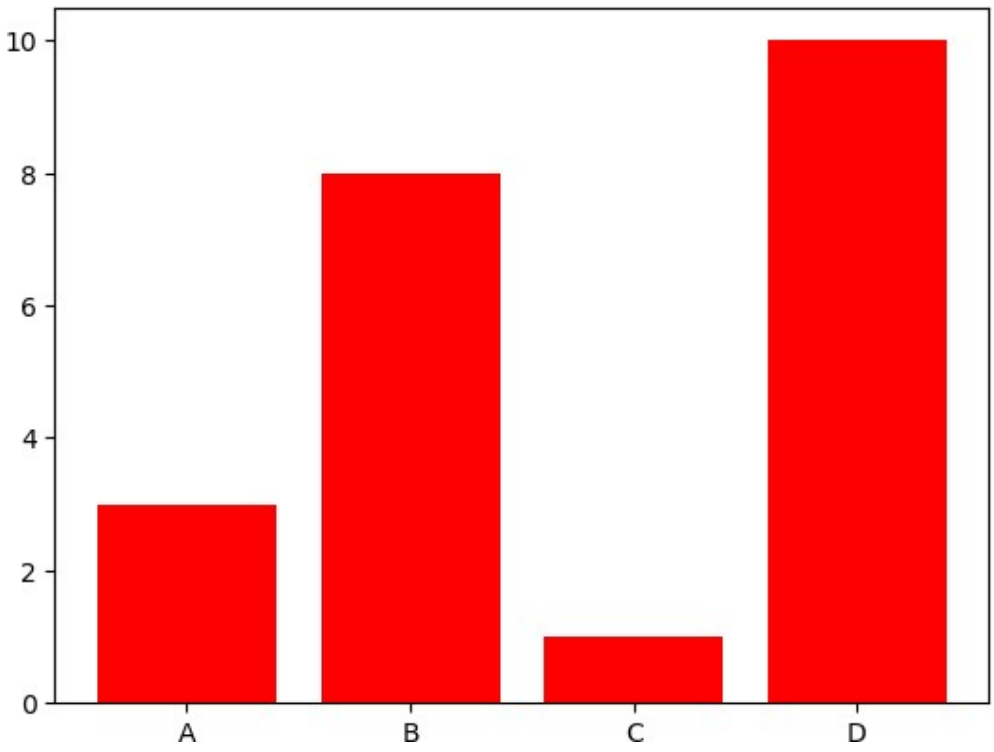
In [147…

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

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.barh(x, y)
plt.show()
```



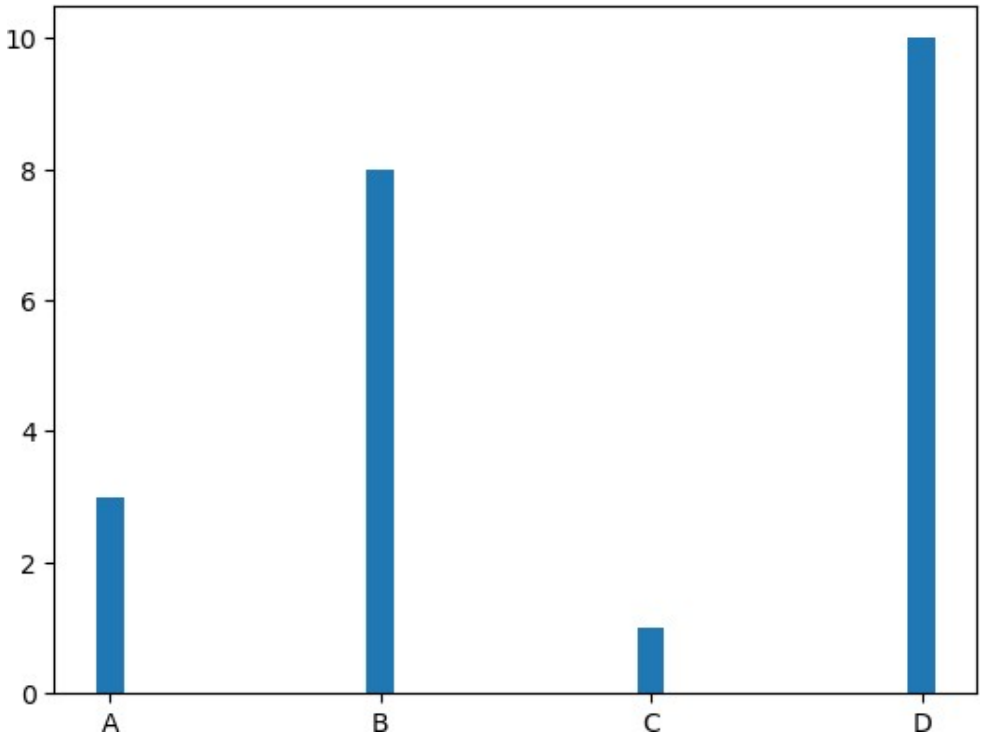
In [149…

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

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.bar(x, y, color = "red")
plt.show()
```

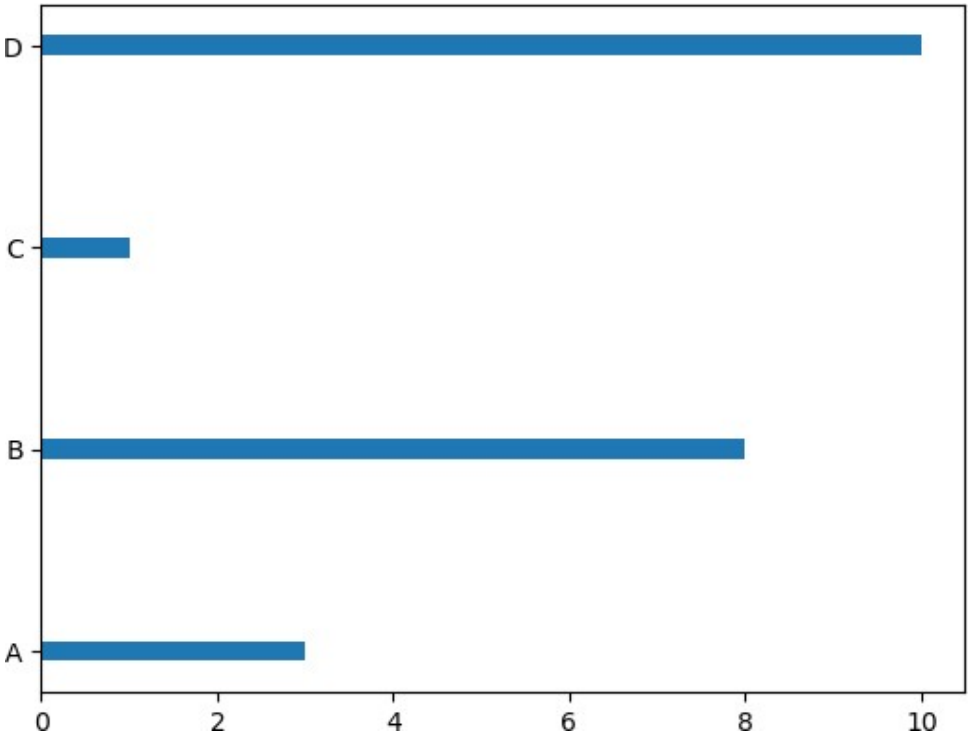


In [151…

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

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.bar(x, y, width = 0.1)
plt.show()
```



In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.barh(x, y, height = 0.1)
plt.show()
```



Scipy:

In [1]:
```
from scipy import constants
print(constants.pi)
```

```
3.141592653589793
```

In [3]:
```
from scipy import constants
print(dir(constants))
```

```
['Avogadro', 'Boltzmann', 'Btu', 'Btu_IT', 'Btu_th', 'ConstantWarning', 'G', 'Jul
ian_year', 'N_A', 'Planck', 'R', 'Rydberg', 'Stefan_Boltzmann', 'Wien', '__all_
_', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name_
_', '__package__', '__path__', '__spec__', '_codata', '_constants', '_obsolete_co
nstants', 'acre', 'alpha', 'angstrom', 'arcmin', 'arcminute', 'arcsec', 'arcsecon
d', 'astronomical_unit', 'atm', 'atmosphere', 'atomic_mass', 'atto', 'au', 'bar',
'barrel', 'bbl', 'blob', 'c', 'calorie', 'calorie_IT', 'calorie_th', 'carat', 'ce
nti', 'codata', 'constants', 'convert_temperature', 'day', 'deci', 'degree', 'deg
ree_Fahrenheit', 'deka', 'dyn', 'dyne', 'e', 'eV', 'electron_mass', 'electron_vol
t', 'elementary_charge', 'epsilon_0', 'erg', 'exa', 'exbi', 'femto', 'fermi', 'fi
nd', 'fine_structure', 'fluid_ounce', 'fluid_ounce_US', 'fluid_ounce_imp', 'foo
t', 'g', 'gallon', 'gallon_US', 'gallon_imp', 'gas_constant', 'gibi', 'giga', 'go
lden', 'golden_ratio', 'grain', 'gram', 'gravitational_constant', 'h', 'hbar', 'h
ectare', 'hecto', 'horsepower', 'hour', 'hp', 'inch', 'k', 'kgf', 'kibi', 'kilo',
'kilogram_force', 'kmh', 'knot', 'lambda2nu', 'lb', 'lbf', 'light_year', 'liter',
'litre', 'long_ton', 'm_e', 'm_n', 'm_p', 'm_u', 'mach', 'mebi', 'mega', 'metric_
ton', 'micro', 'micron', 'mil', 'mile', 'milli', 'minute', 'mmHg', 'mph', 'mu_0',
'nano', 'nautical_mile', 'neutron_mass', 'nu2lambda', 'ounce', 'oz', 'parsec', 'p
ebi', 'peta', 'physical_constants', 'pi', 'pico', 'point', 'pound', 'pound_forc
e', 'precision', 'proton_mass', 'psi', 'pt', 'quecto', 'quetta', 'ronna', 'ront
o', 'short_ton', 'sigma', 'slinch', 'slug', 'speed_of_light', 'speed_of_sound',
'stone', 'survey_foot', 'survey_mile', 'tebi', 'tera', 'test', 'ton_TNT', 'torr',
'troy_ounce', 'troy_pound', 'u', 'unit', 'value', 'week', 'yard', 'year', 'yobi',
'yocto', 'yotta', 'zebi', 'zepto', 'zero_Celsius', 'zetta']
```

In [7]:
```
print(constants.micro)
print(constants.degree)
```

```
1e-06
0.017453292519943295
```

In [9]:
```
import numpy as np
from scipy.sparse import csr_matrix
arr = np.array([0, 0, 0, 0, 0, 1, 1, 0, 2])
print(csr_matrix(arr))
```

```
  (0, 5)        1
  (0, 6)        1
  (0, 8)        2
```

In [11]:
```
import numpy as np
from scipy.sparse import csr_matrix
arr = np.array([[0,0,0],[0,0,1],[1,0,2]])
print(csr_matrix(arr).data)
```

```
[1 1 2]
```

In [13]:
```
arr = np.array([[0,0,0],[0,0,1],[1,0,2]])
print(csr_matrix(arr).count_nonzero())
```

```
3
```

In [15]:
```
arr = np.array([0, 0, 0, 0, 0, 1, 1, 0, 2])
mat=csr_matrix(arr)
mat.eliminate_zeros()
print(mat)
```

```
  (0, 5)        1
  (0, 6)        1
  (0, 8)        2
```

In [17]:
```
arr = np.array([[0, 0, 0], [0, 0, 1], [1, 0, 2]])
mat = csr_matrix(arr)
mat.sum_duplicates()
print(mat)
```

```
  (1, 2)        1
  (2, 0)        1
  (2, 2)        2
```

In [19]:
```
arr = np.array([[0, 0, 0], [0, 0, 1], [1, 0, 2]])
newarr = csr_matrix(arr).tocsc()
print(newarr)
```

```
  (2, 0)        1
  (1, 2)        1
  (2, 2)        2
```

In [21]:
```
import numpy as np
from scipy.sparse.csgraph import connected_components
from scipy.sparse import csr_matrix
arr = np.array([
  [0, 1, 2],
  [1, 0, 0],
  [2, 0, 0]
])
newarr = csr_matrix(arr)
print(connected_components(newarr))
```

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

In [ ]:

Scikit-learn:

In [ ]:
```
AIM:Introduction to required python libraries such as Numpy,Pandas,Scipy,Matplot
```

In [ ]:
```
Description:Scikit-learn (sklearn) is a powerful, open-source Python library
            widely used for machine learning,.Offering a comprehensive suite
            of algorithms and tools for tasks like classification, regression,
             clustering, and dimensionality reduction.
```

In [52]:
```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
```

In [53]:
```
x=np.arange(16).reshape(8,2)
y=range(8)
print(x,"",y)
```

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

In [54]:
```
#training
x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=0.8,random_state=4
print(x_train)
print(x_test)
```

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

In [55]:
```
#testing
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42
print(x_test)
print(y_test)
```

```
[[ 2  3]
 [10 11]]
[1, 5]
```

In [56]:
```
#validation  train
x_train,x_combine,y_train,y_combine=train_test_split(x,y,test_size=0.5,random_st
print(x_train)
print(y_train)
```

```
[[ 4  5]
 [ 8  9]
 [ 6  7]
 [12 13]]
[2, 4, 3, 6]
```

In [57]:
```
#validation test
x_val,x_test,y_val,y_test=train_test_split(x,y,test_size=0.5,random_state=42)
print(x_test)
print(y_test)
```

```
[[ 2  3]
 [10 11]
 [ 0  1]
 [14 15]]
[1, 5, 0, 7]
```

In [ ]:
```
AIM:Import,Preprocess and split the datasets using scikit learn
```

In [ ]:
```
Description:Scikit-learn (sklearn) is a powerful, open-source Python library
        widely used for machine learning, offering a comprehensive suite of
        algorithms and tools for tasks like classification, regression,
        clustering, and dimensionality reduction.
```

In [58]:
```
import numpy as np
from sklearn.datasets import load_iris
iris=load_iris()
print(iris.data)
```

```
[[6.9 3.2  5.7  2.3]
[5.6   2.8 4.9  2. ]
[7.7   2.8 6.7  2. ]
[6.3   2.7 4.9  1.8]
[6.7   3.3 5.7  2.1]
[7.2   3.2 6.   1.8]
[6.2   2.8 4.8  1.8]
[6.1   3.  4.9  1.8]
[6.4   2.8 5.6  2.1]
[7.2   3.  5.8  1.6]
[7.4   2.8 6.1  1.9]
[7.9   3.8 6.4  2. ]
[6.4   2.8 5.6  2.2]
[6.3   2.8 5.1  1.5]
[6.1   2.6 5.6  1.4]
[7.7   3.  6.1  2.3]
[6.3   3.4 5.6  2.4]
[6.4   3.1 5.5  1.8]
[6.    3.  4.8  1.8]
[6.9   3.1 5.4  2.1]
[6.7   3.1 5.6  2.4]
[6.9   3.1 5.1  2.3]
[5.8   2.7 5.1  1.9]
[6.8   3.2 5.9  2.3]
[6.7   3.3 5.7  2.5]
[6.7   3.  5.2  2.3]
[6.3   2.5 5.   1.9]
[6.5   3.  5.2  2. ]
[6.2   3.4 5.4  2.3]
[5.9   3.  5.1  1.8]
[5.1   3.5 1.4  0.2]
 [4.9 3.  1.4  0.2]
 [4.7 3.2 1.3  0.2]
 [4.6 3.1 1.5  0.2]
 [5.  3.6 1.4  0.2]
 [5.4 3.9 1.7  0.4]
 [4.6 3.4 1.4  0.3]
 [5.  3.4 1.5  0.2]
 [4.4 2.9 1.4  0.2]
 [4.9 3.1 1.5  0.1]
 [5.4 3.7 1.5  0.2]
 [4.8 3.4 1.6  0.2]
 [4.8 3.  1.4  0.1]
 [4.3 3.  1.1  0.1]
 [5.8 4.  1.2  0.2]
 [5.7 4.4 1.5  0.4]
 [5.4 3.9 1.3  0.4]
```

```
[5.4 3.4 1.7 0.2]
[5.1 3.7 1.5 0.4]
[4.6 3.6 1.  0.2]
[5.1 3.3 1.7 0.5]
[4.8 3.4 1.9 0.2]
[5.  3.  1.6 0.2]
[5.  3.4 1.6 0.4]
[5.2 3.5 1.5 0.2]
[5.2 3.4 1.4 0.2]
[4.7 3.2 1.6 0.2]
[4.8 3.1 1.6 0.2]
[5.4 3.4 1.5 0.4]
[5.2 4.1 1.5 0.1]
[5.5 4.2 1.4 0.2]
[4.9 3.1 1.5 0.2]
[5.  3.2 1.2 0.2]
[5.5 3.5 1.3 0.2]
[4.9 3.6 1.4 0.1]
[4.4 3.  1.3 0.2]
[5.1 3.4 1.5 0.2]
[5.  3.5 1.3 0.3]
[4.5 2.3 1.3 0.3]
[4.4 3.2 1.3 0.2]
[5.  3.5 1.6 0.6]
[5.1 3.8 1.9 0.4]
[4.8 3.  1.4 0.3]
[5.1 3.8 1.6 0.2]
[4.6 3.2 1.4 0.2]
[5.3 3.7 1.5 0.2]
[5.  3.3 1.4 0.2]
[7.  3.2 4.7 1.4]
[6.4 3.2 4.5 1.5]
[6.9 3.1 4.9 1.5]
[5.5 2.3 4.  1.3]
[6.5 2.8 4.6 1.5]
[5.7 2.8 4.5 1.3]
[6.3 3.3 4.7 1.6]
[4.9 2.4 3.3 1. ]
[6.6 2.9 4.6 1.3]
[5.2 2.7 3.9 1.4]]
```

In [59]:
```
featurenames=iris.feature_names
print(featurenames)
targetnames=iris.target_names
print(targetnames)
```

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (c
m)']
['setosa' 'versicolor' 'virginica']
```

In [60]:
```
y=iris.target
print(y)
```

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

In [1]:
```
from sklearn.datasets import load_iris, load_digits
dig = load_digits()
print(dig.data)
print(load_iris().data)
```

```
[[ 0.  0.  5. ...  0.  0.  0.]
 [ 0.  0.  0. ... 10.  0.  0.]
 [ 0.  0.  0. ... 16.  9.  0.]
 ...
 [ 0.  0.  1. ...  6.  0.  0.]
 [ 0.  0.  2. ... 12.  0.  0.]
 [ 0.  0. 10. ... 12.  1.  0.]]
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]
 [4.8 3.4 1.6 0.2]
 [4.8 3.  1.4 0.1]
 [4.3 3.  1.1 0.1]
 [5.8 4.  1.2 0.2]
 [5.7 4.4 1.5 0.4]
 [5.4 3.9 1.3 0.4]
 [5.1 3.5 1.4 0.3]
 [5.7 3.8 1.7 0.3]
 [5.1 3.8 1.5 0.3]
 [5.4 3.4 1.7 0.2]
 [5.1 3.7 1.5 0.4]
 [4.6 3.6 1.  0.2]
 [5.1 3.3 1.7 0.5]
 [4.8 3.4 1.9 0.2]
 [5.  3.  1.6 0.2]
 [5.  3.4 1.6 0.4]
 [5.2 3.5 1.5 0.2]
 [5.2 3.4 1.4 0.2]
 [4.7 3.2 1.6 0.2]
 [4.8 3.1 1.6 0.2]
 [5.4 3.4 1.5 0.4]
 [5.2 4.1 1.5 0.1]
 [5.5 4.2 1.4 0.2]
 [4.9 3.1 1.5 0.2]
 [5.  3.2 1.2 0.2]
 [5.5 3.5 1.3 0.2]
 [4.9 3.6 1.4 0.1]
 [4.4 3.  1.3 0.2]
 [5.1 3.4 1.5 0.2]
 [5.  3.5 1.3 0.3]
 [4.5 2.3 1.3 0.3]
 [4.4 3.2 1.3 0.2]
 [5.  3.5 1.6 0.6]
 [5.1 3.8 1.9 0.4]
 [4.8 3.  1.4 0.3]
 [5.1 3.8 1.6 0.2]
 [4.6 3.2 1.4 0.2]
 [5.3 3.7 1.5 0.2]
 [5.  3.3 1.4 0.2]
 [7.  3.2 4.7 1.4]
 [6.4 3.2 4.5 1.5]
 [6.9 3.1 4.9 1.5]
```

```
[5.7 2.5 5.  2. ]
[5.8 2.8 5.1 2.4]
[6.4 3.2 5.3 2.3]
[6.5 3.  5.5 1.8]
[7.7 3.8 6.7 2.2]
[7.7 2.6 6.9 2.3]
[6.  2.2 5.  1.5]
[6.9 3.2 5.7 2.3]
[5.6 2.8 4.9 2. ]
[7.7 2.8 6.7 2. ]
[6.3 2.7 4.9 1.8]
[6.7 3.3 5.7 2.1]
[7.2 3.2 6.  1.8]
[6.2 2.8 4.8 1.8]
[6.1 3.  4.9 1.8]
[6.4 2.8 5.6 2.1]
[7.2 3.  5.8 1.6]
[7.4 2.8 6.1 1.9]
[7.9 3.8 6.4 2. ]
[6.4 2.8 5.6 2.2]
[6.3 2.8 5.1 1.5]
[6.1 2.6 5.6 1.4]
[7.7 3.  6.1 2.3]
[6.3 3.4 5.6 2.4]
[6.4 3.1 5.5 1.8]
[6.  3.  4.8 1.8]
[6.9 3.1 5.4 2.1]
[6.7 3.1 5.6 2.4]
[6.9 3.1 5.1 2.3]
[5.8 2.7 5.1 1.9]
[6.8 3.2 5.9 2.3]
[6.7 3.3 5.7 2.5]
[6.7 3.  5.2 2.3]
[6.3 2.5 5.  1.9]
[6.5 3.  5.2 2. ]
[6.2 3.4 5.4 2.3]
[5.9 3.  5.1 1.8]]
```

In [ ]:
```
print(x[1:10])
print(type(x))
```

In [ ]:
```
#train data
x1=iris.data
x=x1[1:20]
y1=iris.target
y=y1[1:20]
print(x)
print(y)
x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=0.8,random_state=4
print(x_train)
print(x_test)
```

In [ ]:
```
x1=iris.data
y1=iris.target
x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=0.7,random_state=4
print(x.shape)
print(x_train)
print(x_train.shape)
```

```
print(y_train)
print(y_train.shape)
```

In [ ]:
```
#testing data
x1=iris.data
y1=iris.target
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42
print(x_test)
print(x_test.shape)
print(y_test)
print(y_test.shape)
```

In [ ]:
```
# datset on diabetes
import numpy as np
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
```

In [ ]:
```
x=np.arange(16).reshape(8,2)
y=range(8)
print(x,"",y)
```

In [ ]:
```
#training
x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=0.8,random_state=4
print(x_train)
print(x_test)
```

In [ ]:
```
#testing
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42
print(x_test)
print(y_test)
```

In [ ]:
```
#validation  train
x_train,x_combine,y_train,y_combine=train_test_split(x,y,test_size=0.5,random_st
print(x_train)
print(y_train)
```

In [ ]:
```
#validation test
x_val,x_test,y_val,y_test=train_test_split(x,y,test_size=0.5,random_state=42)
print(x_test)
print(y_test)
```

In [ ]:
```
from sklearn.datasets import load_diabetes
diabetes=load_diabetes()
print(diabetes.data)
```

In [ ]:
```
feature_names = diabetes.feature_names
print("Feature Names:", feature_names)

# Access target name
target_name = 'disease_progression'
print("Target Name:", target_name)
```

In [ ]:
```
y=diabetes.target
print(y)
```

In [ ]:
```
print(x[1:10])
```

```
print(type(x))
```

In [ ]:
```
#train data
x1=diabetes.data
x=x1[1:20]
y1=diabetes.target
y=y1[1:20]
print(x)
print(y)
x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=0.8,random_state=4
print(x_train)
print(x_test)
```

In [ ]:
```
x1=diabetes.data
y1=diabetes.target
x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=0.7,random_state=4
print(x.shape)
print(x_train)
print(x_train.shape)
print(y_train)
print(y_train.shape)
```

In [ ]:
```
#testing data
x1=iris.data
y1=iris.target
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42
print(x_test)
print(x_test.shape)
print(y_test)
print(y_test.shape)
```

In [77]:
```
#splitting our own data set
import pandas as pd
df=pd.read_csv('diabetes.csv')
```

In [79]:
```
x=df['Glucose']
y=df['Outcome']
print(x)
print(y)
```

```
0      148
1       85
2      183
3       89
4      137
      ...
763    101
764    122
765    121
766    126
767     93
Name: Glucose,  Length: 768, dtype: int64
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome,  Length: 768, dtype: int64
```

In [81]:
```
#train data
x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=0.7,random_state=4
print(x.shape)
print(x_train)
print(x_train.shape)
print(y_train)
print(y_train.shape)
```

```
(768,)
334     95
139    105
485    135
547    131
18     103
      ...
71     139
106     96
270    101
435    141
102    125
Name: Glucose,  Length: 537, dtype: int64
(537,)
334    0
139    0
485    1
547    0
18     0
      ..
71     0
106    0
270    1
435    1
102    0
Name: Outcome,  Length: 537, dtype: int64
(537,)
```

Experiment 3

AIM: Construct a classification model using the Bayes classifier using Python Programming

Description:The Bayes classifier is a theoretical concept in machine learning that represents the best possible classifier for a given problem. It is based on Bayes' theorem, which describes how to update probabilities based on new evidence.

The Naive Bayes classifier is a simple probabilistic classifier based on applying Bayes' theorem with a strong (naive) independence assumption between the features. It is widely used for text classification, spam filtering, and other tasks involving high-dimensional data.

In [1]:
```
import pandas as pd
df=pd.read_csv('Titanic-Dataset.csv')
```

In [3]:
```
df.head(10)
```

Out[3]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71
2	3	1	3	Heikkinen, Miss. ina	female	26.0	0	0	STON/O2. 3101282	7
3	4	1	1	Fut elle, Mrs. Jac ues Heath (Lily May Peel)	female	35.0	1	0	113803	53
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742	11
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736	30

In [7]:
```
#droping unrelavent columns
df.drop(['PassengerId','Name','SibSp','Parch','Ticket','Cabin','Embarked'],axis=
```

In [17]:
```
df.head()
```

Out[17]:

	Survived	Pclass	Sex	Age	Fare
0	0	3	male	22.0	7.2500
1	1	1	female	38.0	71.2833
2	1	3	female	26.0	7.9250
3	1	1	female	35.0	53.1000
4	0	3	male	35.0	8.0500

In [30]:
```
inputs=df.drop('Survived',axis='columns')
target=df.Survived
```

In [32]:
```
dummies=pd.get_dummies(inputs.Sex)
dummies.head()
```

Out[32]:

	female	male
0	False	True
1	True	False
2	True	False
3	True	False
4	False	True

In [36]:
```
inputs=pd.concat([inputs,dummies],axis='columns')
inputs.head()
```

Out[36]:

	Pclass	Sex	Age	Fare	female	male
0	3	male	22.0	7.2500	False	True
1	1	female	38.0	71.2833	True	False
2	3	female	26.0	7.9250	True	False
3	1	female	35.0	53.1000	True	False
4	3	male	35.0	8.0500	False	True

In [40]:
```
inputs.drop(['Sex','male'],axis='columns',inplace=True)
inputs.head(3)
```

Out[40]:

	Pclass	Age	Fare	female
0	3	22.0	7.2500	False
1	1	38.0	71.2833	True
2	3	26.0	7.9250	True

In [42]:
```
inputs.columns[inputs.isna().any()]
```

Out[42]:
```
Index(['Age'], dtype='object')
```

In [44]:
```
inputs.Age[:10]
```

Out[44]:
```
0    22.0
1    38.0
2    26.0
3    35.0
4    35.0
5     NaN
6    54.0
7     2.0
8    27.0
9    14.0
Name: Age, dtype: float64
```

In [46]:
```
inputs.Age=inputs.Age.fillna(inputs.Age.mean())
inputs.head(10)
```

Out[46]:

	Pclass	Age	Fare	female
0	3	22.000000	7.2500	False
1	1	38.000000	71.2833	True
2	3	26.000000	7.9250	True
3	1	35.000000	53.1000	True
4	3	35.000000	8.0500	False
5	3	29.699118	8.4583	False
6	1	54.000000	51.8625	False
7	3	2.000000	21.0750	False
8	3	27.000000	11.1333	True
9	2	14.000000	30.0708	True

In [48]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(inputs,target,test_size=0.2)
```

In [52]:
```
from sklearn.naive_bayes import GaussianNB
model=GaussianNB()
```

In [54]:
```
model.fit(x_train,y_train)
```

Out[54]:
```
GaussianNB
GaussianNB()
```

In [56]:
```
model.score(x_test,y_test)
```

Out[56]:
```
0.8044692737430168
```

In [58]:
```
x_test[0:10]
```

Out[58]:

	Pclass	Age	Fare	female
283	3	19.0	8.0500	False
641	1	24.0	69.3000	True
515	1	47.0	34.0208	False
230	1	35.0	83.4750	True
405	2	34.0	21.0000	False
153	3	40.5	14.5000	False
480	3	9.0	46.9000	False
8	3	27.0	11.1333	True
608	2	22.0	41.5792	True
85	3	33.0	15.8500	True

In [60]:
```
y_test[0:10]
```

Out[60]:
```
283    1
641    1
515    0
230    1
405    0
153    0
480    0
8      1
608    1
85     1
Name: Survived, dtype: int64
```

In [62]:
```
model.predict(x_test[0:10])
```

Out[62]:
```
array([0, 1, 0, 1, 0, 0, 0, 1, 1, 1], dtype=int64)
```

In [64]:
```
model.predict_proba(x_test[:10])
```

Out[64]:
```
array([[0.95918239, 0.04081761],
       [0.03543916, 0.96456084],
       [0.75443285, 0.24556715],
       [0.01868306, 0.98131694],
       [0.92868154, 0.07131846],
       [0.9655783 , 0.0344217 ],
       [0.92231314, 0.07768686],
       [0.48324095, 0.51675905],
       [0.23244707, 0.76755293],
       [0.49582998, 0.50417002]])
```

In [68]:
```
test=[[1,25.000000,15.2750,0]]
a=model.predict(test)
if a[0] == 0:
    print("not Servives")
else:
    print("servived")
```

```
not Servives
```

```
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\base.py:493: UserWarning: X do
es not have valid feature names, but GaussianNB was fitted with feature names
  warnings.warn(
```

In [70]:
```
from sklearn.model_selection import cross_val_score
cross_val_score(GaussianNB(),x_train,y_train,cv=5)
```

Out[70]:
```
array([0.72727273, 0.81818182, 0.77464789, 0.74647887, 0.78169014])
```

In [ ]:
```

```

In [ ]:
```
                                Experiment 4
```

In [ ]:
```
AIM:-Implement a Logistic Regression algorithm for binary classification using P
```

In [ ]:
```
Description: Logistic regression is a supervised machine learning algorithm that
    accomplishes binary classification tasks by predicting the probability of an
    outcome, event, or observation.
```

In [1]:
```
import pandas as pd
df=pd.read_csv('diabetes (1).csv')
df.head()
```

Out[1]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigree
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	

In [3]:
```
df.isnull().sum()
df.columns[df.isna().any()]
```

Out[3]:
```
Index([], dtype='object')
```

In [5]:
```
ind=df[['Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedig
dep=df[['Outcome']]
```

In [9]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(ind,dep,test_size=0.2)
```

In [13]:
```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(random_state=0)
clf.fit(x_train, y_train)
```

```
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\utils\validation.py:1300:  Data
ConversionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Out[13]:
```
LogisticRegression
LogisticRegression(random_state=0)
```

In [15]:
```
clf.score(x_test,y_test)
```

Out[15]:
```
0.7142857142857143
```

In [21]:
```
clf.predict_proba(x_test)
```

Out[21]:
```
array([[0.71622907, 0.28377093],
       [0.81009205, 0.18990795],
       [0.59463224, 0.40536776],
       [0.98544092, 0.01455908],
       [0.64584897, 0.35415103],
       [0.71330441, 0.28669559],
       [0.10329224, 0.89670776],
       [0.72650187, 0.27349813],
       [0.49776377, 0.50223623],
       [0.37667973, 0.62332027],
       [0.72052599, 0.27947401],
       [0.86191889, 0.13808111],
       [0.47654928, 0.52345072],
       [0.67908081, 0.32091919],
       [0.7666752 , 0.2333248 ],
       [0.05401174, 0.94598826],
       [0.73264195, 0.26735805],
       [0.93483705, 0.06516295],
       [0.70590862, 0.29409138],
       [0.22365303, 0.77634697],
       [0.80656196, 0.19343804],
       [0.53267443, 0.46732557],
       [0.88919897, 0.11080103],
       [0.92835281, 0.07164719],
       [0.78139908, 0.21860092],
       [0.53611385, 0.46388615],
       [0.8158336 , 0.1841664 ],
       [0.81093023, 0.18906977],
       [0.47306878, 0.52693122],
       [0.46783077, 0.53216923],
       [0.74614346, 0.25385654],
       [0.71426116, 0.28573884],
       [0.91194273, 0.08805727],
       [0.79221653, 0.20778347],
       [0.77244967, 0.22755033],
       [0.97517836, 0.02482164],
       [0.620061  , 0.379939  ],
       [0.11019756, 0.88980244],
       [0.20571949, 0.79428051],
       [0.70194865, 0.29805135],
       [0.79533069, 0.20466931],
       [0.09510267, 0.90489733],
       [0.84485236, 0.15514764],
       [0.47921408, 0.52078592],
       [0.77907517, 0.22092483],
       [0.5673333 , 0.4326667 ],
       [0.97345384, 0.02654616],
       [0.65932125, 0.34067875],
       [0.8988344 , 0.1011656 ],
       [0.89010266, 0.10989734],
       [0.57967007, 0.42032993],
       [0.26490145, 0.73509855],
       [0.65296159, 0.34703841],
       [0.54123515, 0.45876485],
       [0.22409965, 0.77590035],
       [0.28705577, 0.71294423],
       [0.94183493, 0.05816507],
       [0.78226737, 0.21773263],
       [0.9780263 , 0.0219737 ],
       [0.78154232, 0.21845768],
```

```
       [0.58746367,  0.41253633],
       [0.91140108,  0.08859892],
       [0.3236746 ,  0.6763254 ],
       [0.37067956,  0.62932044],
       [0.9391977 ,  0.0608023 ],
       [0.30413127,  0.69586873],
       [0.60897691,  0.39102309],
       [0.99563158,  0.00436842],
       [0.14398838,  0.85601162],
       [0.57015964,  0.42984036],
       [0.24278779,  0.75721221],
       [0.80713986,  0.19286014],
       [0.71411178,  0.28588822],
       [0.8566079 ,  0.1433921 ],
       [0.72162031,  0.27837969],
       [0.80262964,  0.19737036],
       [0.77257635,  0.22742365],
       [0.50992607,  0.49007393],
       [0.97243056,  0.02756944],
       [0.5410379 ,  0.4589621 ],
       [0.84895895,  0.15104105],
       [0.76476342,  0.23523658],
       [0.8364076 ,  0.1635924 ],
       [0.87885388,  0.12114612],
       [0.61023124,  0.38976876],
       [0.91964223,  0.08035777],
       [0.96083667,  0.03916333],
       [0.21784905,  0.78215095],
       [0.81524858,  0.18475142],
       [0.90029458,  0.09970542],
       [0.92420832,  0.07579168],
       [0.98722146,  0.01277854],
       [0.92482487,  0.07517513],
       [0.89143193,  0.10856807]])
```

In [23]:
```
clf.score(x_test,y_test)
```

Out[23]:
```
0.7142857142857143
```

In [25]:
```
x_test[0:10]
```

Out[25]:

	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	A
652	123	74	40	77	34.1	0.269	
214	112	82	32	175	34.2	0.260	
646	167	74	17	144	23.4	0.447	
680	56	56	28	45	24.2	0.332	
312	155	74	17	96	26.6	0.433	
189	139	80	35	160	31.6	0.361	
489	194	80	0	0	26.1	0.551	
162	114	80	34	285	44.2	0.167	
153	153	82	42	485	40.6	0.687	
54	150	66	42	342	34.7	0.718	

In [27]:
```
y_test[0:10]
```

Out[27]:

	Outcome
652	0
214	1
646	1
680	0
312	1
189	1
489	0
162	0
153	0
54	0

In [39]:
```
test=[[150,47,40,1,29,1,30]]
a=clf.predict(test)
if a[0]==0:
    print("no")
else:
    print("yes")
```

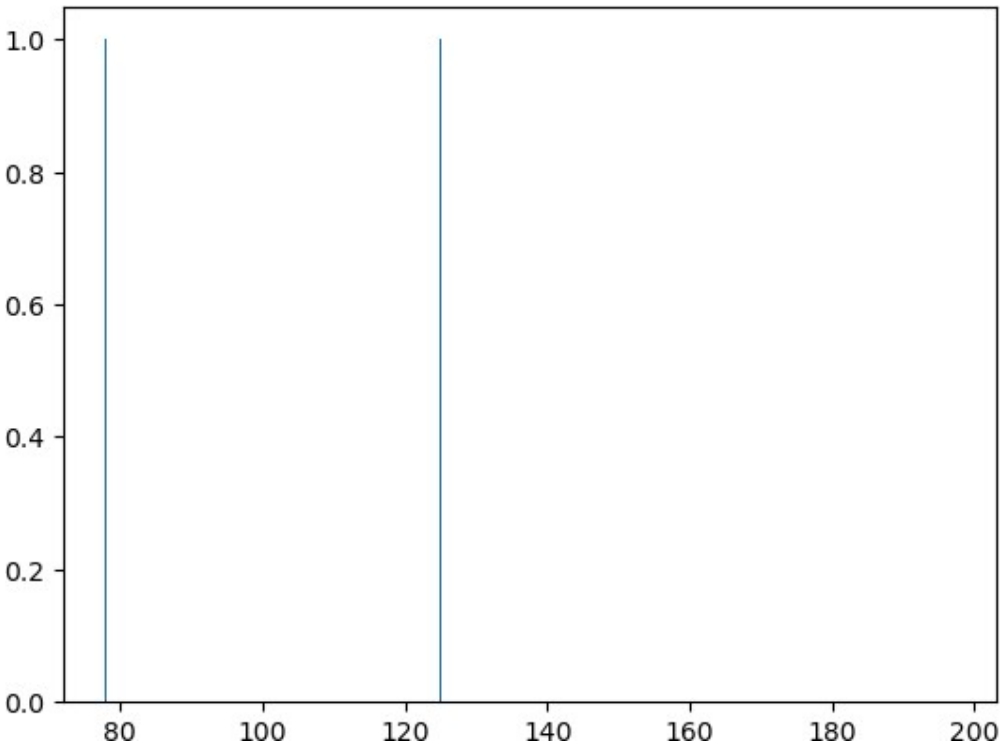
```
yes
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\base.py:493: UserWarning: X do
es not have valid feature names, but LogisticRegression was fitted with feature n
ames
  warnings.warn(
```

In [49]:
```
g=int(input("Enter Glucose"))
b=int(input("enter Blood Presure"))
s=int(input("enter skin thickness"))
```

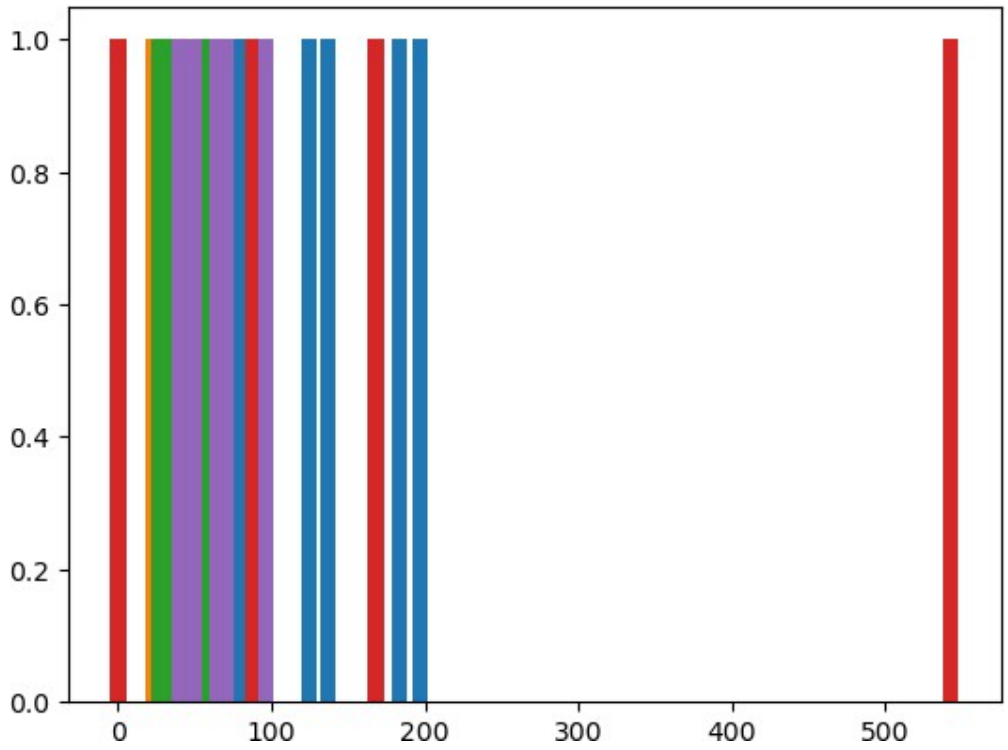
```
i=int(input("enter insullen"))
bm=float(input("enter bmi"))
d=float(input("enter dafunction"))
a=int(input("enter age"))
pr=clf.predict([[g,b,s,i,bm,d,a]])
if pr[0]==0:
    print("no")
else:
    print("yes")
```

```
yes
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\base.py:493: UserWarning: X do
es not have valid feature names, but LogisticRegression was fitted with feature n
ames
  warnings.warn(
```

In [53]:
```
import matplotlib.pyplot as mp
x=df.Glucose[1:10]
y=df.Outcome[1:10]
mp.bar(x,y,width=0.1)
mp.show()
```



In [57]:
```
x = df[['Glucose', 'BMI', 'Age', 'Insulin', 'BloodPressure']].iloc[1:10]
y = df.Outcome[1:10]
for col in x.columns:
    mp.bar(x[col], y, width=10)

mp.show()
```



In [63]:
```
from sklearn.metrics import accuracy_score,classification_report,confusion_matri
accuracy=accuracy_score(y_test,clf.predict(x_test))
print("accuracy:{:.2f}%".format(accuracy*100))
```

```
accuracy:71.43%
```

In [69]:
```
print("Confusion matrix")
confusion_matrix(y_test,clf.predict(x_test))
```

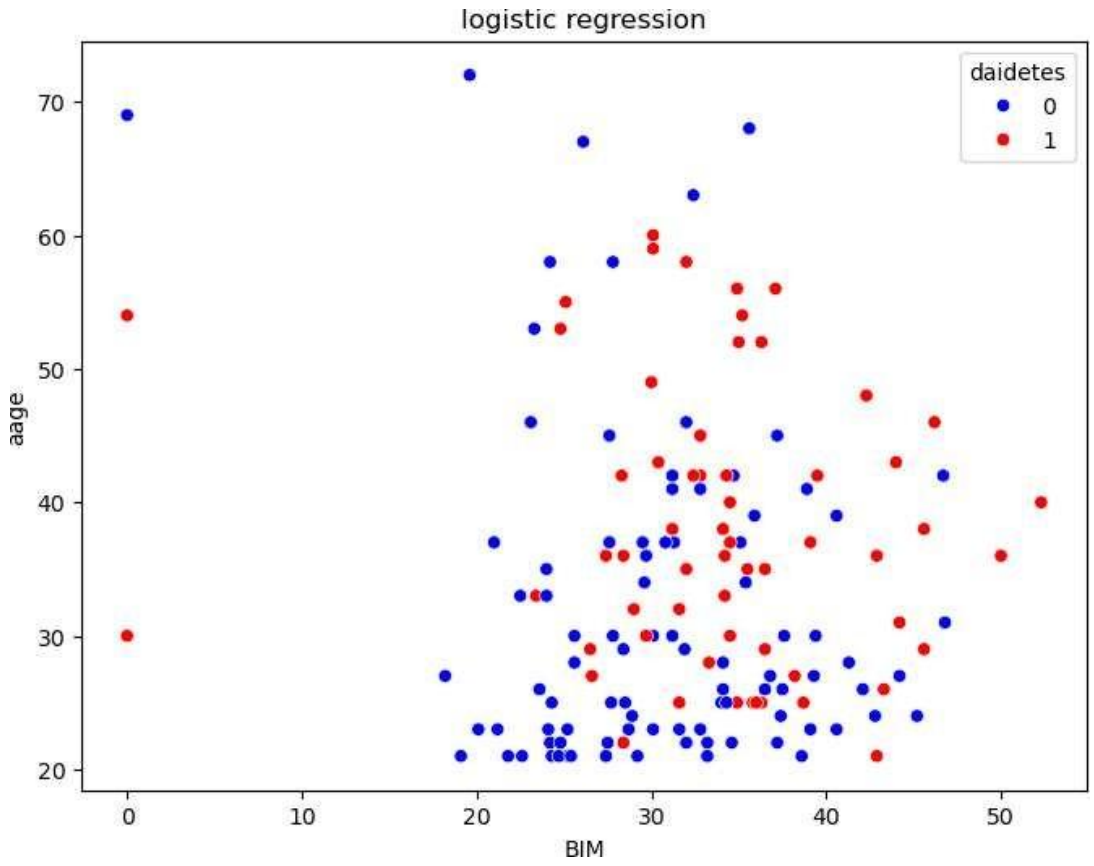
```
Confusion matrix
```

Out[69]:
```
array([[80, 13],
       [31, 30]], dtype=int64)
```

In [71]:
```
print("classification  report:\n",classification_report(y_test,clf.predict(x_test
```

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

           0       0.72      0.86      0.78        93
           1       0.70      0.49      0.58        61

    accuracy                           0.71       154
   macro avg       0.71      0.68      0.68       154
weighted avg       0.71      0.71      0.70       154
```

In [79]:
```
import matplotlib.pyplot  as plt
import seaborn as sns
plt.figure(figsize=(8,6))
sns.scatterplot(x=x_test['BMI'],y=x_test['Age'],hue=y_test['Outcome'],palette={0
plt.xlabel("BIM")
plt.ylabel("aage")
plt.title("logistic regression")
plt.legend(title="daidetes",loc="upper right")
plt.show()
```


logistic regression
daidetes
0
1
aage
BIM
70
60
50
40
30
20
0
10
20
30
40
50

In [ ]:
```
)
```

Experiment:5

AIM:Implement the KNN algorithm for classification and demonstrate the process of finding out optimal —K value using Python Programming.

In [ ]:
```
Description:K-Nearest Neighbors (KNN) is a non-parametric, instance-based learning method. I Classification: For a new data point, the algorithm identifies its nearest neigh The predicted class is determined by the majority class among these neighbors. Regression: The algorithm predicts the value for a new data point by averaging t
```

In [1]:
```
import pandas as pd
```

In [5]:
```
df=pd.read_csv('iris11.csv')
```

In [7]:
```
df.head()
```

Out[7]:

	sepal.length	sepal.width	petal.length	petal.width	variety
0	5.1	3.5	1.4	0.2	Setosa
1	4.9	3.0	1.4	0.2	Setosa
2	4.7	3.2	1.3	0.2	Setosa
3	4.6	3.1	1.5	0.2	Setosa
4	5.0	3.6	1.4	0.2	Setosa

In [7]:
```
df.head()
```

Out[7]:

	sepal.length	sepal.width	petal.length	petal.width	variety
0	5.1	3.5	1.4	0.2	Setosa
1	4.9	3.0	1.4	0.2	Setosa
2	4.7	3.2	1.3	0.2	Setosa
3	4.6	3.1	1.5	0.2	Setosa
4	5.0	3.6	1.4	0.2	Setosa

In [9]:
```
df.isna().any()
```

Out[9]:
```
sepal.length    False
sepal.width     False
petal.length    False
petal.width     False
variety         False
dtype: bool
```

In [11]:
```
x=df.iloc[:,0:4]
y=df.iloc[:,-1]
```

In [13]:
```
x.head()
```

Out[13]:

	sepal.length	sepal.width	petal.length	petal.width
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

In [15]:
```
y.head()
```

Out[15]:
```
0    Setosa
1    Setosa
2    Setosa
3    Setosa
4    Setosa
Name: variety, dtype: object
```

In [17]:
```
df.shape
```

Out[17]:
```
(150, 5)
```

In [27]:
```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=12)
```

In [29]:
```
knn.fit(x,y)
```

Out[29]:
```
KNeighborsClassifier(n_neighbors=12)
```

In [33]:
```
knn.predict(x)
```

Out[33]:
```
array(['Setosa', 'Setosa', 'Setosa', 'Setosa', 'Setosa', 'Setosa',
       'Setosa', 'Setosa', 'Setosa', 'Setosa', 'Setosa', 'Setosa',
       'Setosa', 'Setosa', 'Setosa', 'Setosa', 'Setosa', 'Setosa',
       'Setosa', 'Setosa', 'Setosa', 'Setosa', 'Setosa', 'Setosa',
       'Setosa', 'Setosa', 'Setosa', 'Setosa', 'Setosa', 'Setosa',
       'Setosa', 'Setosa', 'Setosa', 'Setosa', 'Setosa', 'Setosa',
       'Setosa', 'Setosa', 'Setosa', 'Setosa', 'Setosa', 'Setosa',
      'Setosa', 'Setosa', 'Setosa', 'Setosa', 'Setosa', 'Setosa',
     'Setosa', 'Setosa', 'Versicolor', 'Versicolor', 'Versicolor',
     'Versicolor', 'Versicolor', 'Versicolor', 'Versicolor',
     'Versicolor', 'Versicolor', 'Versicolor', 'Versicolor',
     'Versicolor', 'Versicolor', 'Versicolor', 'Versicolor',
     'Versicolor', 'Versicolor', 'Versicolor', 'Versicolor',
     'Versicolor', 'Versicolor', 'Versicolor', 'Versicolor',
     'Versicolor', 'Versicolor', 'Versicolor', 'Versicolor',
     'Versicolor', 'Versicolor', 'Virginica', 'Versicolor',
     'Versicolor', 'Versicolor', 'Versicolor', 'Versicolor',
     'Versicolor', 'Versicolor', 'Versicolor', 'Versicolor',
     'Versicolor', 'Versicolor', 'Versicolor', 'Versicolor',
     'Versicolor', 'Versicolor', 'Versicolor', 'Virginica', 'Virginica',
     'Virginica', 'Virginica', 'Virginica', 'Virginica', 'Versicolor',
     'Virginica', 'Virginica', 'Virginica', 'Virginica', 'Virginica',
     'Virginica', 'Virginica', 'Virginica', 'Virginica', 'Virginica',
     'Virginica', 'Virginica', 'Virginica', 'Virginica', 'Virginica',
     'Virginica', 'Virginica', 'Virginica', 'Virginica', 'Virginica',
     'Virginica', 'Virginica', 'Virginica', 'Virginica', 'Virginica',
     'Virginica', 'Versicolor', 'Virginica', 'Virginica', 'Virginica',
     'Virginica', 'Virginica', 'Virginica', 'Virginica', 'Virginica',
     'Virginica', 'Virginica', 'Virginica'], dtype=object)
```

In [35]:
```
y_pred=knn.predict(x)
```

In [37]:
```
y_pred
```

Out[37]:
```
array(['Setosa', 'Setosa', 'Setosa', 'Setosa', 'Setosa', 'Setosa',
       'Setosa', 'Setosa', 'Setosa', 'Setosa', 'Setosa', 'Setosa',
       'Setosa', 'Setosa', 'Setosa', 'Setosa', 'Setosa', 'Setosa',
       'Setosa', 'Setosa', 'Setosa', 'Setosa', 'Setosa', 'Setosa',
       'Setosa', 'Setosa', 'Setosa', 'Setosa', 'Setosa', 'Setosa',
       'Setosa', 'Setosa', 'Setosa', 'Setosa', 'Setosa', 'Setosa',
      'Setosa', 'Setosa', 'Setosa', 'Setosa', 'Setosa', 'Setosa',
     'Setosa', 'Setosa', 'Versicolor', 'Versicolor', 'Versicolor',
     'Versicolor', 'Versicolor', 'Versicolor', 'Versicolor',
     'Versicolor', 'Versicolor', 'Versicolor', 'Versicolor',
     'Versicolor', 'Versicolor', 'Versicolor', 'Versicolor',
     'Versicolor', 'Versicolor', 'Versicolor', 'Versicolor',
     'Versicolor', 'Versicolor', 'Versicolor', 'Versicolor',
     'Versicolor', 'Versicolor', 'Versicolor', 'Versicolor',
     'Versicolor', 'Versicolor', 'Virginica', 'Versicolor',
     'Versicolor', 'Versicolor', 'Versicolor', 'Versicolor',
     'Versicolor', 'Versicolor', 'Versicolor', 'Versicolor',
     'Versicolor', 'Versicolor', 'Versicolor', 'Versicolor',
     'Versicolor', 'Versicolor', 'Versicolor', 'Virginica', 'Virginica',
     'Virginica', 'Virginica', 'Virginica', 'Virginica', 'Versicolor',
     'Virginica', 'Virginica', 'Virginica', 'Virginica', 'Virginica',
     'Virginica', 'Virginica', 'Virginica', 'Virginica', 'Virginica',
     'Virginica', 'Virginica', 'Virginica', 'Virginica', 'Virginica',
     'Virginica', 'Virginica', 'Virginica', 'Virginica', 'Virginica',
     'Virginica', 'Virginica', 'Virginica', 'Virginica', 'Virginica',
     'Virginica', 'Versicolor', 'Virginica', 'Virginica', 'Virginica',
     'Virginica', 'Virginica', 'Virginica', 'Virginica', 'Virginica',
     'Virginica', 'Virginica', 'Virginica'], dtype=object)
```

In [41]:
```
y_predict=pd.DataFrame([y_pred,y.values])
```

In [45]:
```
y_predict
```

Out[45]:

	0	1	2	3	4	5	6	7	8	9	...	
0	Setosa	Setosa	Setosa	Setosa	Setosa	Setosa	Setosa	Setosa	Setosa	Setosa	...	Vi
1	Setosa	Setosa	Setosa	Setosa	Setosa	Setosa	Setosa	Setosa	Setosa	Setosa	...	Vi

2 rows × 150 columns

In [47]:
```
y_predict.transpose()
```

Out[47]:

	0	1
0	Setosa	Setosa
1	Setosa	Setosa
2	Setosa	Setosa
3	Setosa	Setosa
4	Setosa	Setosa
...	...	...
145	Virginica	Virginica
146	Virginica	Virginica
147	Virginica	Virginica
148	Virginica	Virginica
149	Virginica	Virginica

150 rows × 2 columns

In [53]:
```
print(knn.score(x,y))
```

```
0.98
```

In [65]:
```
knn.predict([[1,2,3,4]])
```

```
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\base.py:493: UserWarning: X do
es not have valid feature names, but KNeighborsClassifier was fitted with feature
names
  warnings.warn(
```

Out[65]: array(['Versicolor'], dtype=object)

In [67]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2)
```

In [69]:
```
knn.fit(x,y)
```

Out[69]:

KNeighborsClassifier

KNeighborsClassifier(n_neighbors=12)

In [71]:
```
knn.score(x_test,y_test)
```

Out[71]: 1.0

In [73]:
```
knn.predict_proba(x_test)
```

Out[73]:
```
array([[1.        , 0.        , 0.        ],
       [0.        , 1.        , 0.        ],
       [0.        , 0.        , 1.        ],
       [1.        , 0.        , 0.        ],
       [0.        , 0.91666667, 0.08333333],
       [0.        , 0.16666667, 0.83333333],
       [0.        , 0.        , 1.        ],
       [1.        , 0.        , 0.        ],
       [0.        , 0.08333333, 0.91666667],
       [0.        , 0.        , 1.        ],
       [0.        , 0.58333333, 0.41666667],
       [1.        , 0.        , 0.        ],
       [0.        , 0.        , 1.        ],
       [1.        , 0.        , 0.        ],
       [0.        , 1.        , 0.        ],
       [0.        , 0.        , 1.        ],
       [1.        , 0.        , 0.        ],
       [0.        , 1.        , 0.        ],
       [0.        , 1.        , 0.        ],
       [0.        , 0.25      , 0.75      ],
       [0.        , 0.66666667, 0.33333333],
       [1.        , 0.        , 0.        ],
       [1.        , 0.        , 0.        ],
       [0.        , 1.        , 0.        ],
       [0.        , 1.        , 0.        ],
       [0.        , 0.        , 1.        ],
       [1.        , 0.        , 0.        ],
       [0.        , 0.08333333, 0.91666667],
       [1.        , 0.        , 0.        ],
       [0.        , 0.41666667, 0.58333333]])
```

In [81]:
```
yp=knn.predict(x_test)
yp
```

Out[81]:
```
array(['Setosa', 'Versicolor', 'Virginica', 'Setosa', 'Versicolor',
       'Virginica', 'Virginica', 'Setosa', 'Virginica', 'Virginica',
       'Versicolor', 'Setosa', 'Virginica', 'Setosa', 'Versicolor',
       'Virginica', 'Setosa', 'Versicolor', 'Versicolor', 'Virginica',
       'Versicolor', 'Setosa', 'Setosa', 'Versicolor', 'Versicolor',
       'Virginica', 'Setosa', 'Virginica', 'Setosa', 'Virginica'],
      dtype=object)
```

In [85]:
```
ypredict=pd.DataFrame([yp,y.values])
```

In [87]:
```
ypredict.transpose()
```

Out[87]:

	0	1
0	Setosa	Setosa
1	Versicolor	Setosa
2	Virginica	Setosa
3	Setosa	Setosa
4	Versicolor	Setosa
...	...	...
145	None	Virginica
146	None	Virginica
147	None	Virginica
148	None	Virginica
149	None	Virginica

150 rows × 2 columns

In [89]:
```
print(knn.score(x_test,y_test))
```

```
1.0
```

In [91]:
```
knn.predict([[1,2,3,4]])
```

```
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\base.py:493: UserWarning: X do
es not have valid feature names, but KNeighborsClassifier was fitted with feature
names
  warnings.warn(
```

Out[91]: array(['Versicolor'], dtype=object)

In [ ]:

Experiment-6

In [1]:
```
Aim: Construct an SVM classifier using python programming.
```

In [ ]:
```
Description: A support vector machine (SVM) is a supervised machine learning
algorithm that classifies data by finding an optimal line or hyperplane
that maximizes the distance between each class in an N-dimensional space.
```

In [1]:
```
import pandas as pd
df=pd.read_csv("Social_Network_Ads.csv")
```

In [3]:
```
df.head()
```

Out[3]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

In [5]:
```
x=df.iloc[:,[2,3]]#(:,2:4)
y=df.iloc[:,-1]
```

In [7]:
```
import numpy as np
x.head()
```

Out[7]:

	Age	EstimatedSalary
0	19	19000
1	35	20000
2	26	43000
3	27	57000
4	19	76000

In [9]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test=train_test_split(x,y,
                    test_size=0.2,random_state=0)
```

In [11]:
```
#normalisation
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
X_train=sc.fit_transform(X_train)
X_test=sc.fit_transform(X_test)
```

In [13]:
```
from sklearn.svm import SVC
classifer=SVC(kernel='linear',random_state=0)
classifer.fit(X_train,Y_train)
```

Out[13]:

SVC

SVC(kernel='linear', random_state=0)

In [15]:
```
classifer.predict(X_train)
```

Out[15]:
```
array([1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0,
       0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1,
       0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1,
       0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1,
       0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0,
       1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0,
       1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0,
       0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0,
       0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0,
       0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1,
       1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int64)
```

In [17]:
```
classifer.score(X_train,Y_train)
```

Out[17]:
```
0.821875
```

In [19]:
```
from sklearn.svm import SVC
classifer=SVC(kernel='rbf',random_state=0)
classifer.fit(X_train,Y_train)
```

Out[19]:

SVC

SVC(random_state=0)

In [21]:
```
classifer.predict(X_train)
```

Out[21]:
```
array([1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1,
       0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1,
       0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1,
       0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1,
       1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1,
       0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1,
       0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1,
       0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0,
       1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0,
       1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0,
       0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0,
       0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0,
       0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1,
       1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1], dtype=int64)
```

In [23]:
```
classifer.score(X_train,Y_train)
```

Out[23]:
```
0.903125
```

In [29]:
```
from sklearn.metrics import accuracy_score,classification_report
from sklearn.metrics import confusion_matrix,roc_curve,auc
accuracy=accuracy_score(Y_test,classifer.predict(X_test))
print("Accuracy:{:.2f}%".format(accuracy*100))
```

```
Accuracy:93.75%
```

In [31]:
```
cm=confusion_matrix(Y_test,classifer.predict(X_test))
cm
```

Out[31]:
```
array([[54,  4],
       [ 1, 21]], dtype=int64)
```

In [ ]:
```
Output
```

In [33]:
```
#output
print("Classification Report:\n",classification_report
      (Y_test,classifer.predict(X_test)))
```

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

           0       0.98       0.93       0.96         58
           1       0.84       0.95       0.89         22

    accuracy                            0.94         80
   macro avg       0.91       0.94       0.92         80
weighted avg       0.94       0.94       0.94         80
```

In [ ]:

Experiment-7

In [ ]:
```
AIM:Demonstrate the process of the Decision Tree construction for Classification
```

In [ ]:
```
Description:Decision trees are an approach used in supervised machine learning,
The approach is used mainly to solve classification problems,.
which is the use of a model to categorise or classify an object.
```

In [166…
```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
```

In [168…
```
df=pd.read_csv(r'D:\22a81a05f9\diabetes.csv')
```

In [170…
```
df.head()
```

Out[170…

	Preg ancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeF
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	

In [172…
```
#data preprocessing
df.isnull().sum()
```

Out[172…
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [174…
```
df.isna().any()
```

Out[174…
```
Pregnancies                 False
Glucose                     False
BloodPressure               False
SkinThickness               False
Insulin                     False
BMI                         False
DiabetesPedigreeFunction    False
Age                         False
Outcome                     False
dtype: bool
```

In [176…
```
#feature extraction
features=['Pregnancies',         'Glucose','BloodPressure','SkinThickness','Insul
```

```
ind=df[features]
dep=df.Outcome
```

In [178…
```
#training the data
x_train,x_test,y_train,y_test=train_test_split(ind,dep,test_size=0.3,random_stat
```

In [180…
```
#implement the model
dt=DecisionTreeClassifier()
dt.fit(x_train,y_train)
```

Out[180…
```
DecisionTree lassifier
DecisionTreeClassifier()
```

In [182…
```
#improve accuracy
dt=DecisionTreeClassifier(criterion='entropy',max_depth=3)
dt.fit(x_train,y_train)
```

Out[182…
```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=3)
```

In [184…
```
dt.predict(x_test)
```

Out[184…
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0,
       1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,
       1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0], dtype=int64)
```

In [186…
```
pred=dt.predict(x_test)
```

In [188…
```
print("classification accuracy is",metrics.accuracy_score(y_test,pred))
```

```
classification accuracy is 0.7705627705627706
```

In [190…
```
pip install pydotplus
```

```
Requirement already satisfied: pydotplus in c:\users\hp\anaconda3\lib\site-packag
es (2.0.2)
Requirement already satisfied: pyparsing>=2.0.1 in c:\users\hp\anaconda3\lib\site
-packages (from pydotplus) (3.1.2)
Note: you may need to restart the kernel to use updated packages.
```

In [193…
```
conda install python-graphviz
```

```
Channels:
 - defaults
Platform: win-64
Collecting package metadata (repodata.json): ...working... done
Solving environment: ...working... done

# All requested packages already installed.


Note: you may need to restart the kernel to use updated packages.
```

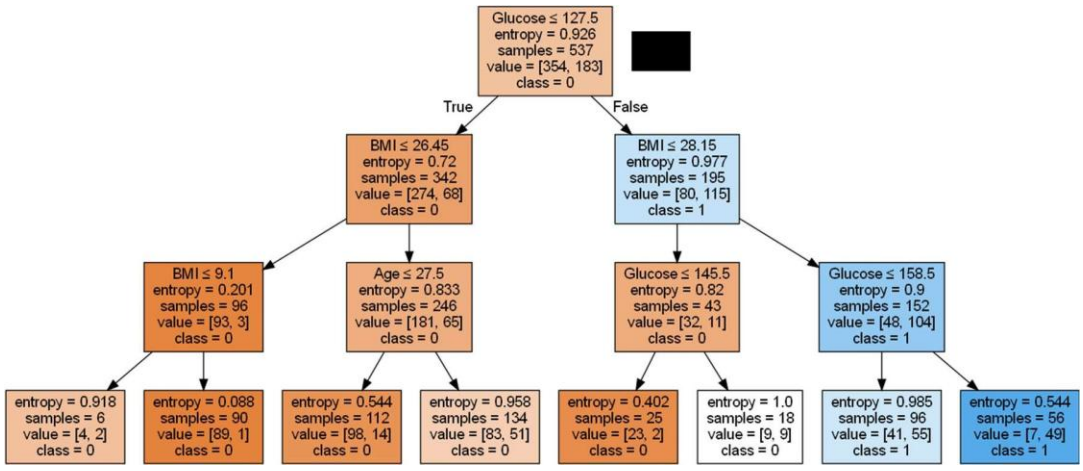
In [194…]
```
import graphviz
print(graphviz.version())
```

```
(2, 50, 0)
```

In [ ]:
```
#visualization
sfrom sklearn.tree import export_graphviz
from six import StringIO
from IPython.display import Image
import pydotplu
```

In [192…]
```
dot_data=StringIO()
export_graphviz(dt,out_file=dot_data,filled=True,special_characters=True,feature
graph=pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('diabetes.png')
Image(graph.create_png())
```

Out[192…]



In [197…]
```
from IPython.display import Image
print(Image)
```

```
<class 'IPython.core.display.Image'>
```

In [ ]:
```

```

Experiment-8

Aim: Implement an Ensemble Learner using Random Forest Algorithm using python programming.

Description:

- A Random Forest is a collection of decision trees that work together to make predictions.
- It takes different random parts of the dataset to train each tree and then it combines the results by averaging them.
- This approach helps improve the accuracy of predictions.
- Random Forest is based on ensemble learning.

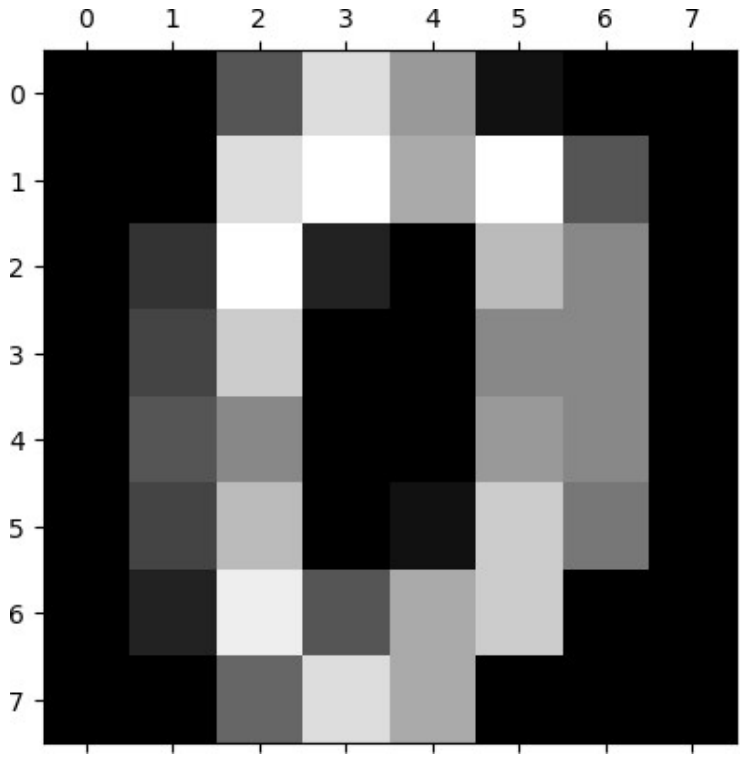
In [1]:
```
import pandas as pd
from sklearn.datasets import load_digits
digits=load_digits()
```

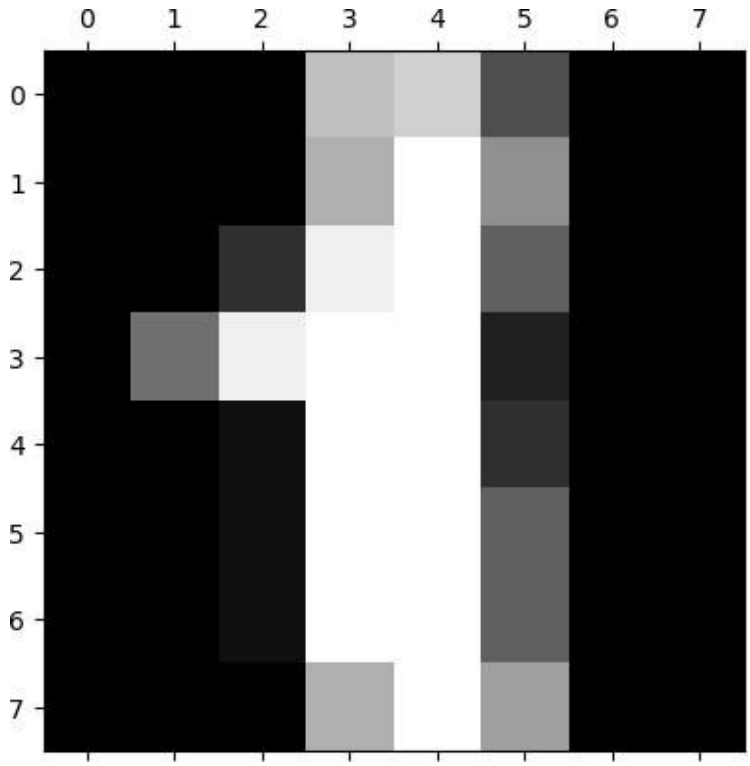
In [2]:
```
dir(digits)
```

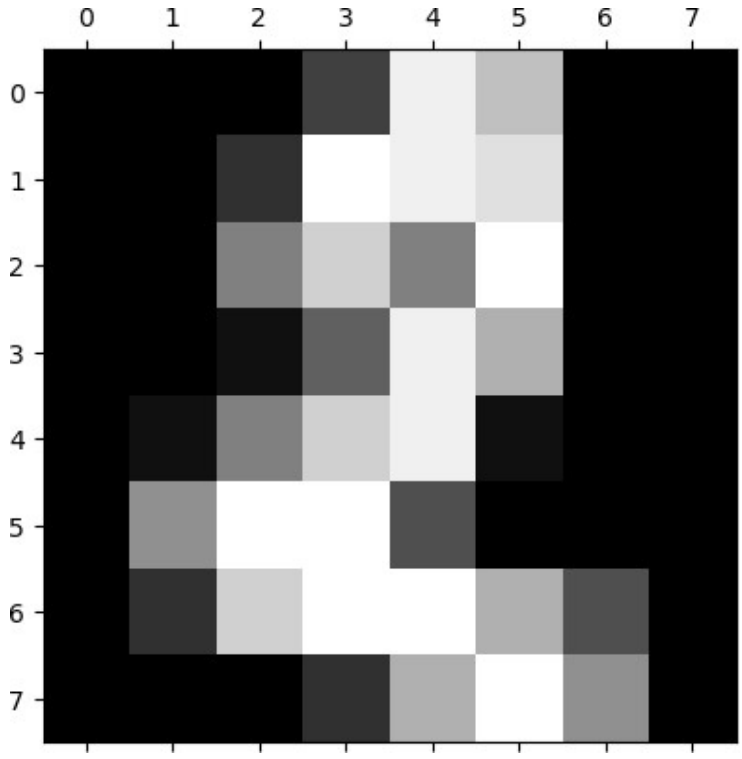
Out[2]:
```
['DESCR', 'data', 'feature_names', 'frame', 'images', 'target', 'target_names']
```

In [15]:
```
#import matplotlib.pyplot as plt
#plt.gray()
for i in range(3):
    plt.matshow(digits.images[i])
```

```
<Figure size 640x480 with 0 Axes>
```







In [3]:
```
df=pd.DataFrame(digits.data)
digits.data
```

Out[3]:
```
array([[ 0.,  0.,  5., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ..., 10.,  0.,  0.],
       [ 0.,  0.,  0., ..., 16.,  9.,  0.],
       ...,
       [ 0.,  0.,  1., ...,  6.,  0.,  0.],
       [ 0.,  0.,  2., ..., 12.,  0.,  0.],
       [ 0.,  0., 10., ..., 12.,  1.,  0.]])
```

In [4]:
```
df.head()
```

Out[4]:

	0	1	2	3	4	5	6	7	8	9	…	54	55	56	57	58	59	6
0	0.0	0.0	5.0	13.0	9.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	6.0	13.0	10.
1	0.0	0.0	0.0	12.0	13.0	5.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	11.0	16.
2	0.0	0.0	0.0	4.0	15.0	12.0	0.0	0.0	0.0	0.0	...	5.0	0.0	0.0	0.0	0.0	3.0	11.
3	0.0	0.0	7.0	15.0	13.0	1.0	0.0	0.0	0.0	8.0	...	9.0	0.0	0.0	0.0	7.0	13.0	13.
4	0.0	0.0	0.0	1.0	11.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	2.0	16.

5 rows × 64 columns

In [6]:
```
df['target']=digits.target
```

In [7]:
```
df.head()
```

Out[7]:

	0	1	2	3	4	5	6	7	8	9	…	55	56	57	58	59	60	6
0	0.0	0.0	5.0	13.0	9.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	6.0	13.0	10.0	0
1	0.0	0.0	0.0	12.0	13.0	5.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	11.0	16.0	10
2	0.0	0.0	0.0	4.0	15.0	12.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	3.0	11.0	16
3	0.0	0.0	7.0	15.0	13.0	1.0	0.0	0.0	0.0	8.0	...	0.0	0.0	0.0	7.0	13.0	13.0	9
4	0.0	0.0	0.0	1.0	11.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	2.0	16.0	4

5 rows × 65 columns

In [8]:
```
ind=df.drop(['target'],axis=1)
dep=df['target']
```

In [9]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(ind,dep,test_size=0.3)
```

In [11]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [12]:
```
model=RandomForestClassifier()
```

In [13]:
```
model.fit(x_train,y_train)
```

Out[13]:

RandomForest lassifier

RandomForestClassifier()

In [16]:
```
model.estimators_
```

Out[16]:

```
[DecisionTreeClassifier(max_features='sqrt', random_state=1897489940),
 DecisionTreeClassifier(max_features='sqrt', random_state=897628562),
 DecisionTreeClassifier(max_features='sqrt', random_state=713547363),
 DecisionTreeClassifier(max_features='sqrt', random_state=729170825),
 DecisionTreeClassifier(max_features='sqrt', random_state=542770258),
 DecisionTreeClassifier(max_features='sqrt', random_state=1538013650),
 DecisionTreeClassifier(max_features='sqrt', random_state=1468164770),
 DecisionTreeClassifier(max_features='sqrt', random_state=1946700223),
 DecisionTreeClassifier(max_features='sqrt', random_state=509919954),
 DecisionTreeClassifier(max_features='sqrt', random_state=176266207),
 DecisionTreeClassifier(max_features='sqrt', random_state=213695991),
 DecisionTreeClassifier(max_features='sqrt', random_state=1533778609),
 DecisionTreeClassifier(max_features='sqrt', random_state=776242420),
 DecisionTreeClassifier(max_features='sqrt', random_state=1846397509),
 DecisionTreeClassifier(max_features='sqrt', random_state=1164452271),
 DecisionTreeClassifier(max_features='sqrt', random_state=987234111),
 DecisionTreeClassifier(max_features='sqrt', random_state=235332938),
 DecisionTreeClassifier(max_features='sqrt', random_state=648220777),
 DecisionTreeClassifier(max_features='sqrt', random_state=821773154),
 DecisionTreeClassifier(max_features='sqrt', random_state=510254726),
 DecisionTreeClassifier(max_features='sqrt', random_state=1967728618),
 DecisionTreeClassifier(max_features='sqrt', random_state=140168056),
 DecisionTreeClassifier(max_features='sqrt', random_state=428067777),
 DecisionTreeClassifier(max_features='sqrt', random_state=1647453830),
 DecisionTreeClassifier(max_features='sqrt', random_state=2015907756),
 DecisionTreeClassifier(max_features='sqrt', random_state=1746990432),
 DecisionTreeClassifier(max_features='sqrt', random_state=309673424),
 DecisionTreeClassifier(max_features='sqrt', random_state=1300222503),
 DecisionTreeClassifier(max_features='sqrt', random_state=409710219),
 DecisionTreeClassifier(max_features='sqrt', random_state=212926445),
 DecisionTreeClassifier(max_features='sqrt', random_state=256855906),
 DecisionTreeClassifier(max_features='sqrt', random_state=1712464546),
 DecisionTreeClassifier(max_features='sqrt', random_state=2056713995),
 DecisionTreeClassifier(max_features='sqrt', random_state=509716612),
 DecisionTreeClassifier(max_features='sqrt', random_state=828650252),
 DecisionTreeClassifier(max_features='sqrt', random_state=26033132),
 DecisionTreeClassifier(max_features='sqrt',  random_state=547877997),
 DecisionTreeClassifier(max_features='sqrt',    random_state=2034223410),
 DecisionTreeClassifier(max_features='sqrt',    random_state=1535164761),
 DecisionTreeClassifier(max_features='sqrt', random_state=404278178)]
```

In [17]:
```
model.score(x_test,y_test)
```

Out[17]: 0.9833333333333333

In [19]:
```
y_predicted=model.predict(x_test)
print(y_predicted)
```

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

In [20]:
```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_predicted)
print(cm)
```

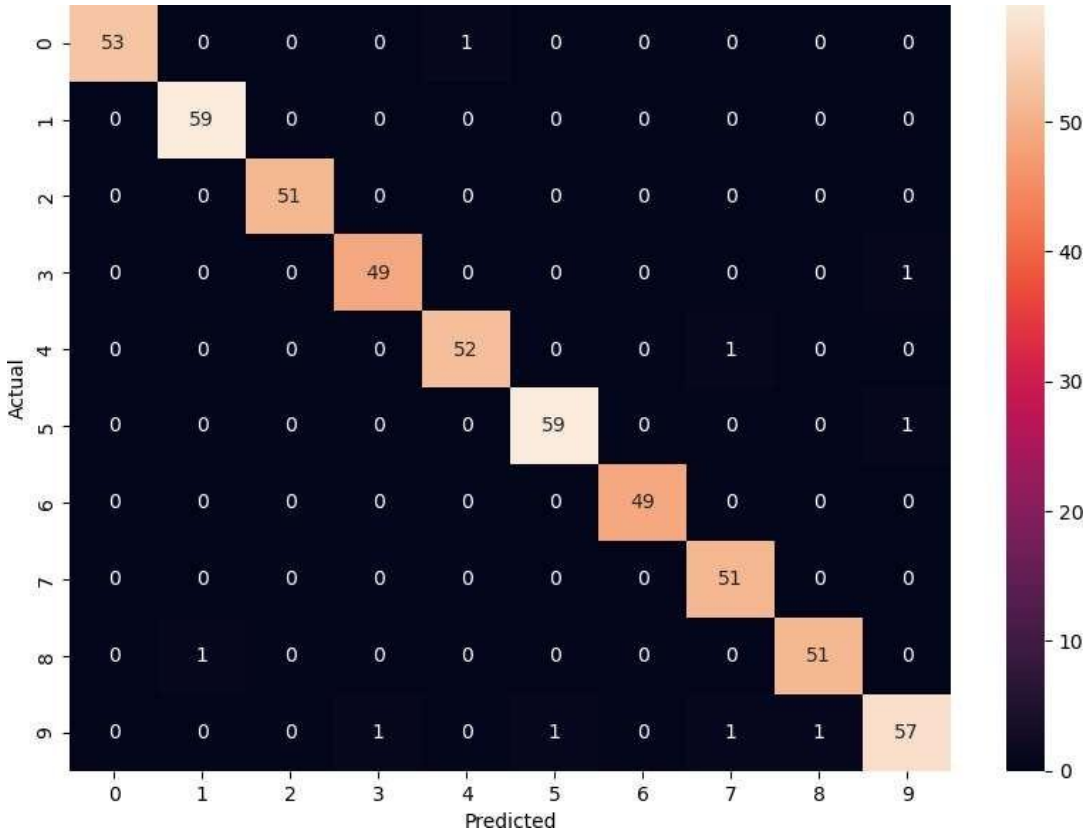
```
[[53  0  0  0  1  0  0  0  0  0]
 [ 0 59  0  0  0  0  0  0  0  0]
 [ 0  0 51  0  0  0  0  0  0  0]
 [ 0  0  0 49  0  0  0  0  0  1]
 [ 0  0  0  0 52  0  0  1  0  0]
 [ 0  0  0  0  0 59  0  0  0  1]
 [ 0  0  0  0  0  0 49  0  0  0]
 [ 0  0  0  0  0  0  0 51  0  0]
 [ 0  1  0  0  0  0  0  0 51  0]
 [ 0  0  0  1  0  1  0  1  1 57]]
```

In [21]:
```
print(x_train.columns)
```

```
Index([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35,
       36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53,
       54, 55, 56, 57, 58, 59, 60, 61, 62, 63],
      dtype='object')
```

In [24]:
```
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(10,7))
sns.heatmap(cm,annot=True)
plt.xlabel('Predicted')
plt.ylabel('Actual')
```

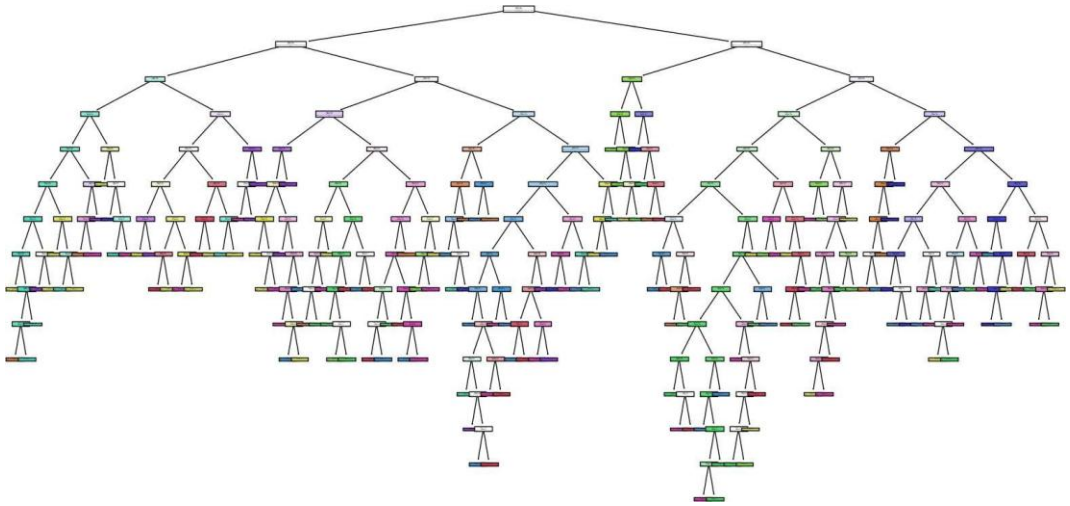
Out[24]:
```
Text(95.72222222222221, 0.5, 'Actual')
```

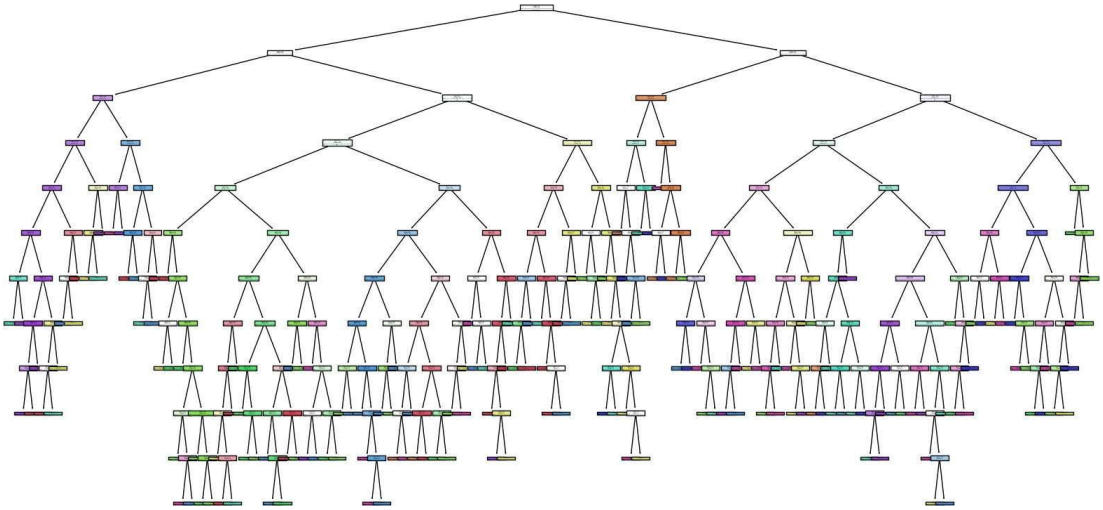


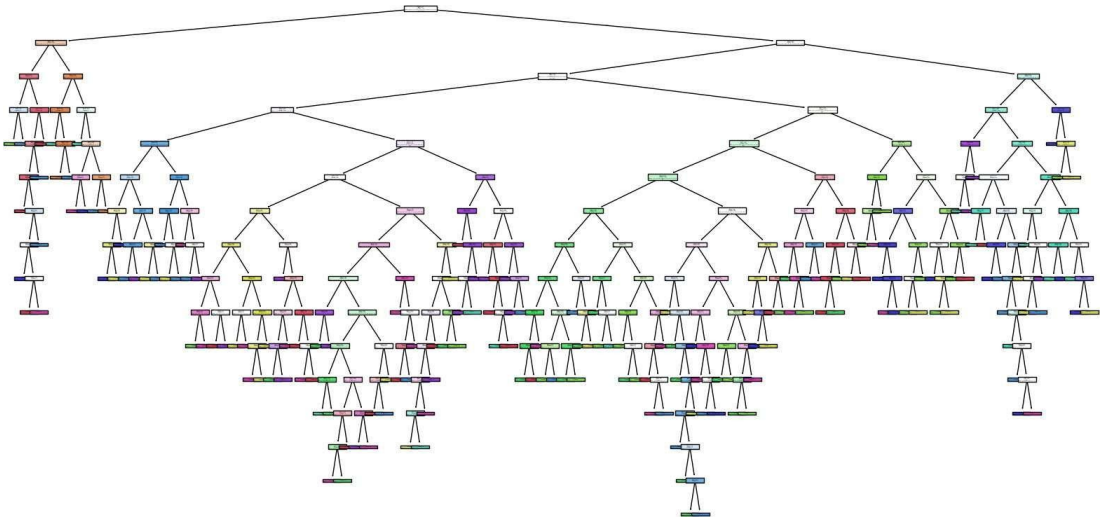
In [25]:
```
#Visualisation
from sklearn.tree import export_graphviz
from six import StringIO
from IPython.display import Image
```

In [26]:
```
from sklearn.tree import plot_tree
```

In [27]:
```
for i in range(3):
    treeplot=model.estimators_[i]
    plt.figure(figsize=(20,10))
    plot_tree(treeplot,filled=True)
    plt.show()
```







In [ ]:

Experiment-9

In [ ]:
```
AIM: To implement an ensemble learner using AdaBoostAlgorithm using Python
    programming.
```

In [ ]:
```
Description: AdaBoostClassifier stands for Adaptive Boosting Classifier.
             It is an ensemble learning method – meaning it builds a strong
             classifier by combining many weak classifiers (like decision trees).
```

In [2]:
```
#import libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

In [6]:
```
#import dataset
df=pd.read_csv(r'D:\22A81A05G0(ML)\diabetes.csv')
df.head()
```

Out[6]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunc
0	6	148	72	35	0	33.6	0
1	1	85	66	29	0	26.6	0
2	8	183	64	0	0	23.3	0
3	1	89	66	23	94	28.1	0
4	0	137	40	35	168	43.1	2

In [8]:
```
#data preprocessing
df.isnull().sum()
```

Out[8]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [10]:
```
df.isna().any()
```

Out[10]:
```
Pregnancies                 False
Glucose                     False
BloodPressure               False
SkinThickness               False
Insulin                     False
BMI                         False
DiabetesPedigreeFunction    False
Age                         False
Outcome                     False
dtype: bool
```

In [12]:
```
#feature extraction
ind=df.drop(['Outcome'],axis=1)
dep=df['Outcome']
```

In [14]:
```
#training the data
x_train,x_test,y_train,y_test=train_test_split(ind,dep,test_size=0.2)
```

In [16]:
```
#import AdaBoodtClassifier
from sklearn.ensemble import AdaBoostClassifier
```

In [22]:
```
#model fitting
ada=AdaBoostClassifier(n_estimators=100,learning_rate=1)
model=ada.fit(x_train,y_train)
```

In [24]:
```
#model prediction
model.predict(x_test)
```

Out[24]:
```
array([0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1,
       0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0,
       0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0,
       0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0],
      dtype=int64)
```

In [26]:
```
#Accuracy score
print(accuracy_score(y_test,model.predict(x_test))*100)
```

```
69.48051948051948
```

In [32]:
```
#implementing svm with adaboost
from sklearn.svm import SVC
svc=SVC(probability=True,kernel='linear')
abc=AdaBoostClassifier(n_estimators=50,estimator=svc,learning_rate=1)
model=abc.fit(x_train,y_train)
print(model.predict(x_test))
```

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

In [ ]:
```
OUTPUT:
```

In [34]:
```
print(accuracy_score(y_test,model.predict(x_test))*100)
```

```
73.37662337662337
```

In [ ]:
```

```

Experiment-10

In [ ]:
```
AIM:Demonstrate the working of Multi-layer perceptron with MLPClassifier()
using Python programming.
Description: A Multi-Layer Perceptron (MLP) is a type of neural network that con
of multiple layers of neurons. It is a supervised learning algorithm used
for classification and regression tasks. The MLPClassifier() is part of
the sklearn.neural_network module in Python Scikit-learn library and is
used for classification tasks.
```

In [1]:
```
import pandas as pd
```

In [3]:
```
df=pd.read_csv('iris.csv')
```

In [5]:
```
df.head()
```

Out[5]:

	sep al_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [7]:
```
df.isnull().sum()
```

Out[7]:
```
sepal_length    0
sepal_width     0
petal_length    0
petal_width     0
species         0
dtype: int64
```

In [9]:
```
df.isna().any()
```

Out[9]:
```
sepal_length    False
sepal_width     False
petal_length    False
petal_width     False
species         False
dtype: bool
```

In [13]:
```
x=df.drop(['species'],axis=1)
y=df['species']
```

In [15]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42
```

In [17]:
```
from sklearn.neural_network import MLPClassifier
```

In [19]:
```
mlp=MLPClassifier(hidden_layer_sizes=(100,),max_iter=1000,random_state=42)
```

In [21]:
```
mlp.fit(x_train,y_train)
```

Out[21]:
```
MLPClassifier(max_iter=1000, random_state=42)
```

In [23]:
```
y_pred=mlp.predict(x_test)
y_pred
```

Out[23]:
```
array(['versicolor', 'setosa', 'virginica', 'versicolor', 'versicolor',
       'setosa', 'versicolor', 'virginica', 'versicolor', 'versicolor',
       'virginica', 'setosa', 'setosa', 'setosa', 'setosa', 'versicolor',
       'virginica', 'versicolor', 'versicolor', 'virginica', 'setosa',
       'virginica', 'setosa', 'virginica', 'virginica', 'virginica',
       'virginica', 'virginica', 'setosa', 'setosa'], dtype='<U10')
```

In [25]:
```
from sklearn.metrics import accuracy_score
accuracy=accuracy_score(y_test,y_pred)
```

In [ ]:
```
#OUTPUT:
```

In [27]:
```
print(f"Neural Network weights:{mlp.coefs_}")
```

```
Neural Network weights:[array([[-8.33064203e-08,  2.64530294e-01,  2.39491291e-0
1,
          6.60616023e-02, -1.77318496e-01, -3.90935072e-02,
           -6.08578520e-03, 1.50144522e-01, 3.11578070e-01,
          3.02201155e-05,  -9.12177664e-03,   2.02655404e-01,
          2.10372968e-01, -5.14193482e-02, -1.09917550e-01,
          -8.45916027e-04, -1.69678398e-05,  2.32425848e-03,
          1.69035716e-02,  -6.57244987e-02,   9.01234575e-02,
         -7.02452058e-02, -2.99295755e-05, -2.11751417e-07,
          6.82856853e-12,   1.07871918e-01,  -3.17160098e-01,
          5.50409231e-02,   8.63637149e-03,  -6.94567121e-03,
          7.89375012e-02,  -3.75472085e-02,   5.13308896e-02,
          2.14704205e-01,   4.83802558e-01,   2.09645651e-01,
         -1.66584123e-05,   2.31989890e-01,   1.24555332e-01,
         -5.72890937e-02,  -2.45885725e-01,   3.24801825e-02,
           -7.91452351e-03, 1.84369295e-01, 2.59767438e-01,
          1.03281201e-01,   1.44109458e-01,  -3.88407562e-03,
          7.50551555e-02,  -8.20089328e-04,   2.20941852e-01,
          1.68166080e-01,   1.67221137e-01,   1.71212978e-01,
          1.58067465e-01,   2.21095010e-01,  -4.17099114e-03,
         -2.17826207e-01, -7.04030516e-03, -5.35127151e-06,
         -8.97235046e-09,  -9.19505769e-02,   1.42854394e-01,
         -2.06841302e-02, -1.34052102e-01, -3.14956209e-02,
          -1.89814640e-03,  1.37521722e-01, -4.99044721e-03,
          2.33663749e-01,   1.80211487e-01,  -3.47744370e-02,
           -1.05610089e-02, 1.82029635e-01, 1.89951807e-01,
          8.97794754e-02,   8.11836319e-02,  -1.63030424e-02,
         -4.63399658e-02,  -2.83335976e-03,   1.36948798e-01,
         -1.33141338e-02, -1.10961921e-01, -2.02508303e-02,
           -2.05699583e-02, 2.63577630e-01, 1.53148498e-01,
          4.31610004e-02,   1.63246148e-01,  -8.98928129e-02,
           -2.67764223e-03, 2.25049169e-01, 8.71078875e-02,
          1.94489172e-01,   3.08360750e-01,  -2.22014711e-02,
          3.84682401e-02,   1.00829961e-01,  -1.23011061e-01,
         7.29248301e-02],
          [-8.16442464e-03, 1.98078234e-01, 2.25112445e-01,
          6.66103604e-02,   1.83537754e-01,  -2.37077169e-02,
          8.02412482e-10,  -1.59481285e-02,   4.46465221e-01,
         -4.84066458e-03, -3.32586654e-05, -3.01929004e-01,
          3.55206979e-01,   3.14390799e-02,   2.43652552e-02,
          2.32799203e-03,   6.35750722e-04,  -1.59908696e-01,
          3.34977513e-01,  -4.92626741e-02,   2.98673265e-01,
          1.43282691e-02, -8.21040809e-06, -3.08875112e-03,
          -2.86078447e-04,  3.99339577e-02, -9.24388957e-02,
          3.27090757e-01, -2.71819023e-01, -7.63105384e-14,
         -1.51640107e-01,  -2.79910811e-02,   1.89471101e-01,
           -2.08560950e-01, 4.59384207e-01, 1.09821145e-01,
          5.75379134e-13,   4.79489992e-01,   2.56612521e-02,
          1.97696237e-01,   1.71565286e-02,  -2.01615458e-01,
          4.31808620e-14,  -2.28434814e-01,   4.93158163e-01,
         -2.49102740e-01,   3.43299615e-01,  -1.78167729e-01,
         -2.23997694e-01,  -5.28546656e-05,   1.05589592e-01,
          -8.02951889e-03, -3.11088781e-01, -2.95090776e-02,
          3.64484930e-01,  -3.60680437e-01,   4.57948290e-06,
         -1.03270714e-01,  -2.16476746e-04,   6.89990180e-05,
         -1.82046911e-07,  -7.57862370e-03,   1.13382575e-01,
         -1.11680970e-02,  -2.25590600e-01,   1.34831832e-01,
          -7.00697898e-06, -3.19476017e-01, -7.39163039e-03,
          1.76353433e-01,   2.52087476e-01,  -7.96644113e-02,
         -2.36546186e-14, -7.82337907e-02, -1.71143400e-02,
```

Experiment-11

In [ ]:
```
Aim:Demonstrate the K-Means algorithm for the given dataset using python program
```

In [ ]:
```
Description:K-Means Clustering is an Unsupervised Learning algorithm, which grou
            It is a centroid-based algorithm, where each cluster is associated w
            The main aim of this algorithm is to minimize the sum of distances b
```

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
```

In [3]:
```
iris=load_iris()
```

In [5]:
```
x=iris.data
y=iris.target
```

In [7]:
```
scaler=StandardScaler()
x_scaled=scaler.fit_transform(x)
```

In [9]:
```
km = KMeans(n_clusters=3,random_state=42)
km.fit(x_scaled)
```

```
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1446:  UserW
arning: KMeans is known to have a memory leak on Windows with MKL, when there are
less chunks than available threads. You can avoid it by setting the environment v
ariable OMP_NUM_THREADS=1.
  warnings.warn(
```

Out[9]:
```
KMeans(n_clusters=3, random_state=42)
```

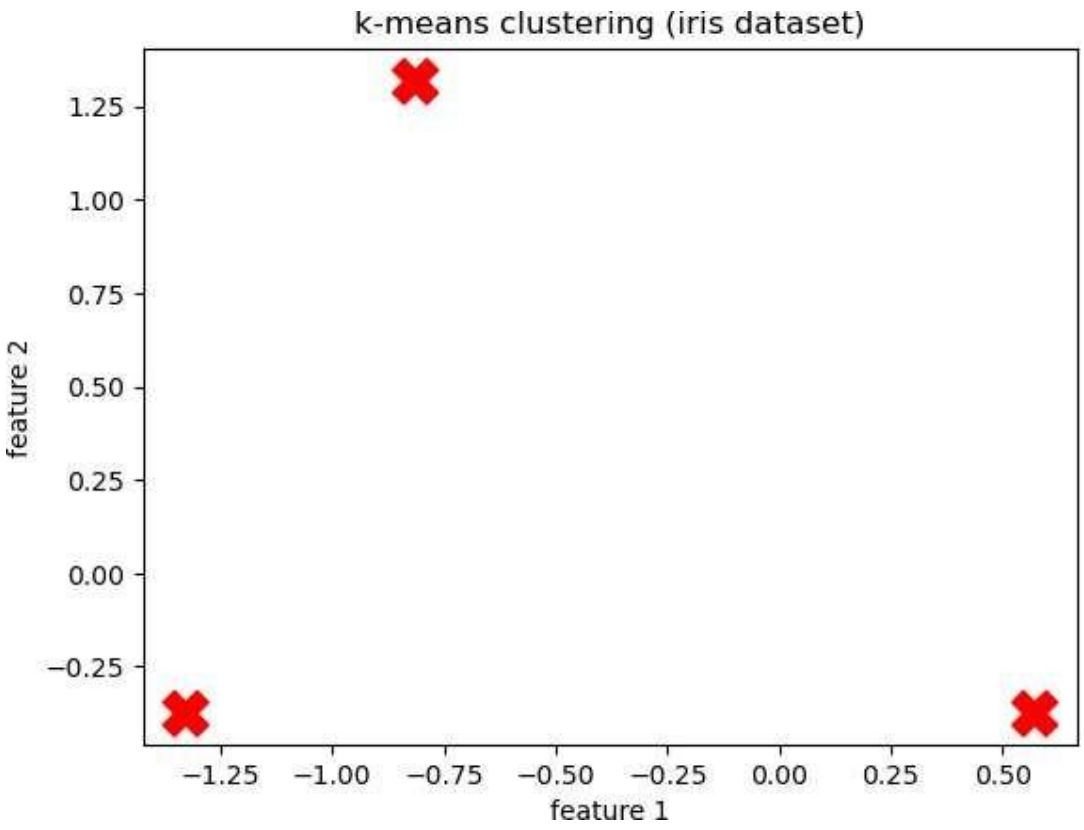
In [11]:
```
print("cluster centers:")
print(km.cluster_centers_)
```

```
cluster centers:
[[ 0.57100359 -0.37176778   0.69111943  0.66315198]
 [-0.81623084  1.31895771  -1.28683379 -1.2197118 ]
 [-1.32765367 -0.373138    -1.13723572 -1.11486192]]
```

In [15]:
```
print("\n predicted labels:")
print(km.labels_)
plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],s=300,c='red',mark
plt.title("k-means clustering (iris dataset)")
plt.xlabel("feature 1")
plt.ylabel("feature 2")
```

```
 predicted labels:
[1 2 2 2 1 1 1 1 2 2 1 1 2 2 1 1 1 1 1 1 1 1 1 2 1 1 1 2 2 1 1 1 2 2 1
 1 2 1 1 2 2 1 1 2 1 2 1 1 0 0 0 0 0 0 0 2 0 0 2 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 0 0 0 0 2 0 0 0 0 2 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 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0]
```

Out[15]:
```
Text(0, 0.5, 'feature 2')
```

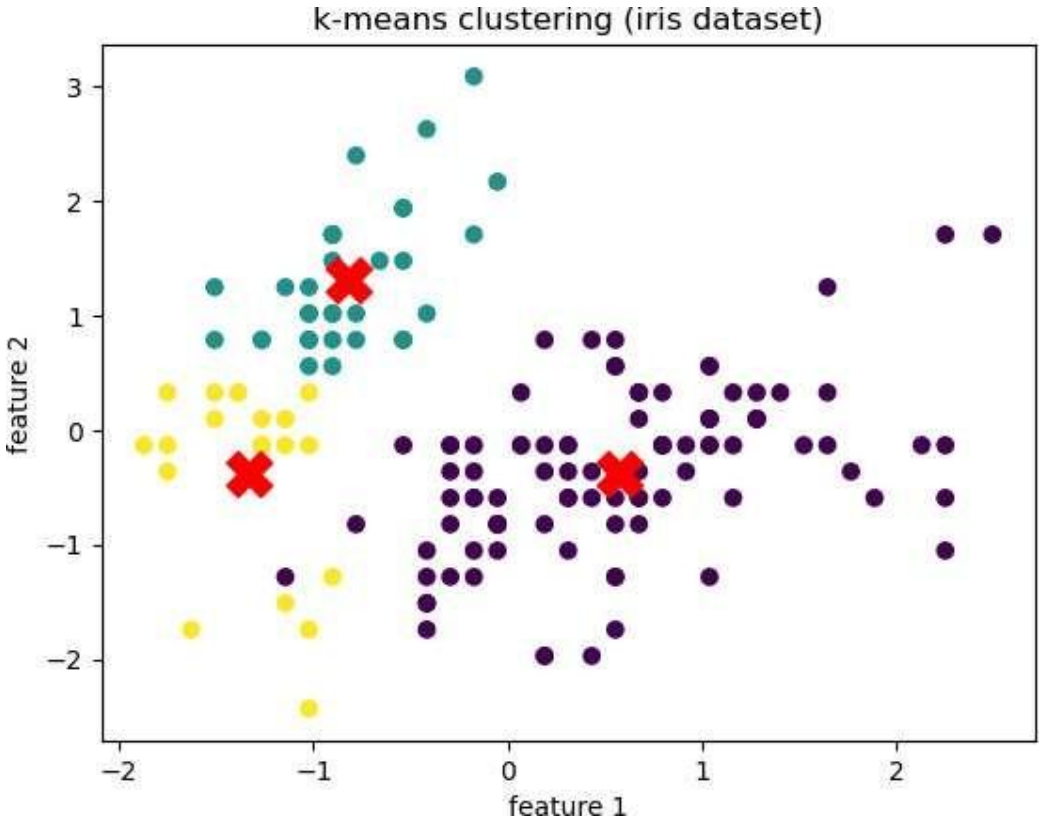


In [ ]:
```
OUTPUT:
```

In [17]:
```
print("\n predicted labels:")
print(km.labels_)
plt.scatter(x_scaled[:,0],x_scaled[:,1],c=km.labels_,cmap='viridis')
plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],s=300,c='red',mark
plt.title("k-means clustering (iris dataset)")
plt.xlabel("feature 1")
plt.ylabel("feature 2")
```

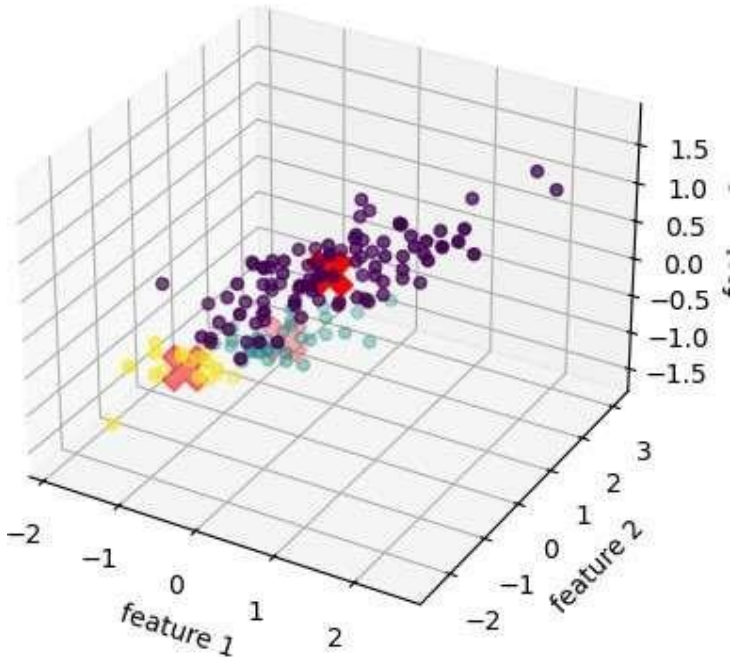
```
predicted labels:
[1 2 2 2 1 1 1 1 2 2 1 1 2 2 1 1 1 1 1 1 1 1 1 1 2 1 1 1 2 2 1 1 1 2 2 1
 1 2 1 1 2 2 1 1 2 1 2 1 1 0 0 0 0 0 0 0 2 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0]
```

Out[17]:
```
Text(0, 0.5, 'feature 2')
```



In [31]:
```
fig=plt.figure()
pl=fig.add_subplot(projection='3d')
pl.scatter(x_scaled[:,0],x_scaled[:,1],x_scaled[:,2],c=km.labels_,cmap='viridis'
pl.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],km.cluster_centers_
pl.set_title("k-means clustering (iris dataset)")
pl.set_xlabel("feature 1")
pl.set_ylabel("feature 2")
pl.set_zlabel("feature 3")
```

Out[31]:
```
Text(0.5, 0, 'feature 3')
```


k-means clustering (iris dataset)
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2
−1
0
1
2
feature 1
3
2
1
0
−1
−2
feature 2

I