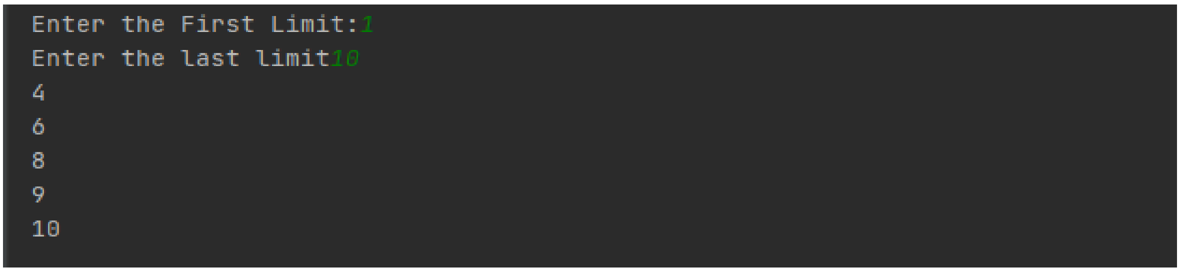
CYCLE 1

1. Program to Print all non-Prime Numbers in an Interval

PROGRAM:

```
first=int(input("Enter the First Limit:"))
last=int(input("Enter the last limit"))
for num in range(first,last + 1):
    if num > 1:
        for i in range(2,num):
            if(num % i == 0):
                print(num)
                break
```

OUTPUT:



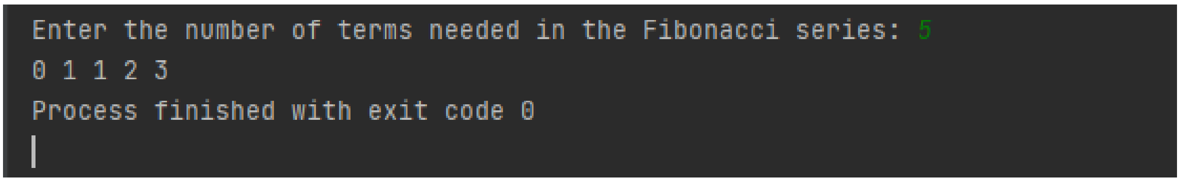
```
Enter the First Limit:1
Enter the last limit10
4
6
8
9
10
```

2. Program to print the first N Fibonacci numbers.

PROGRAM:

```
n = int (input("Enter the number of terms needed in the Fibonacci series: "))
if (n<0):
    print ("Enter a positive number")
else:
    f1, f2 = 0, 1
    if n == 1:
        print (f1)
    elif n == 2:
        print (f1,f2)
    else:
        print (f1,f2, end = ' ')
        for i in range (3, n+1):
            f3 = f1 + f2
            print (f3, end = ' ')
            f1 = f2
            f2 = f3
```

OUTPUT:



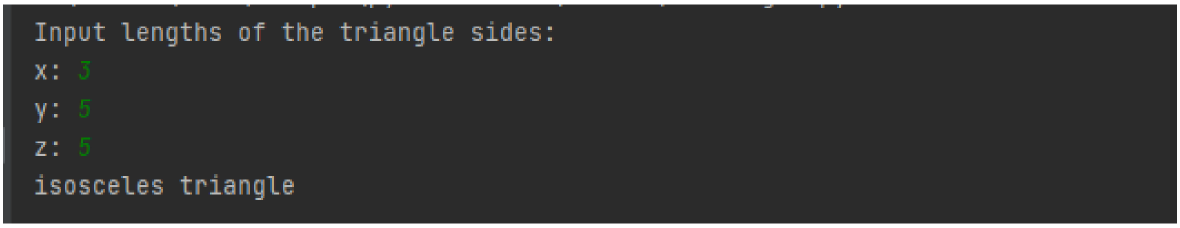
```
Enter the number of terms needed in the Fibonacci series: 5
0 1 1 2 3
Process finished with exit code 0
```

3. Given sides of a triangle, write a program to check whether given triangle is an isosceles, equilateral or scalene.

PROGRAM:

```
print("Input lengths of the triangle sides: ")
x = int(input("x: "))
y = int(input("y: "))
z = int(input("z: "))
if x == y == z:
   print("Equilateral triangle")
elif x==y or y==z or z==x:
   print("isosceles triangle")
else:
   print("Scalene triangle")
```

OUTPUT:



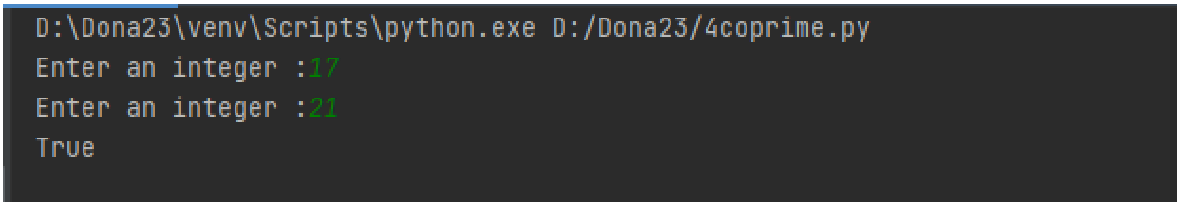
```
Input lengths of the triangle sides:
x: 3
y: 5
z: 5
isosceles triangle
```

4. Program to check whether given pair of number is coprime

PROGRAM:

```
def gcd(p,q):
    while q != 0:
        p, q = q, p%q
    return p
def is_coprime(x, y):
    return gcd(x, y) == 1
a = (int(input("Enter an integer :")))
b = (int(input("Enter an integer :")))
print(is_coprime(a , b))
```

OUTPUT:



```
D:\Dona23\venv\Scripts\python.exe D:/Dona23/4coprime.py
Enter an integer :17
Enter an integer :21
True
```

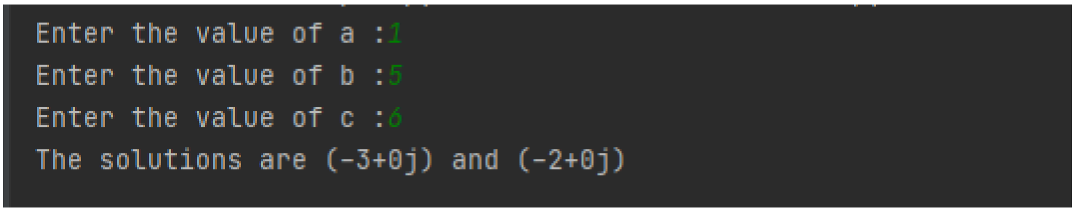
5. Program to find the roots of a quadratic equation (rounded to 2 decimal places)

PROGRAM:

```
import cmath
a = int(input("Enter the value of a :"))
b = int(input("Enter the value of b :"))
c = int(input("Enter the value of c :"))
```

```
d = (b**2) - (4*a*c)
sol1 = (-b-cmath.sqrt(d))/(2*a)
sol2 = (-b+cmath.sqrt(d))/(2*a)
print('The solutions are {0} and {1}'.format(sol1,sol2))
```

OUTPUT:



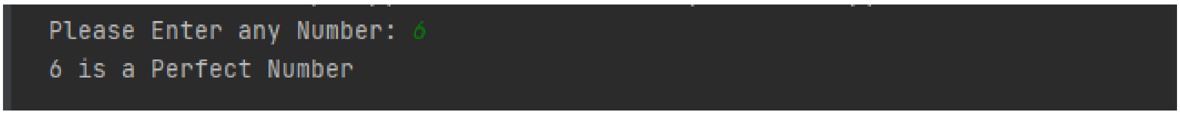
```
Enter the value of a :1
Enter the value of b :5
Enter the value of c :6
The solutions are (-3+0j) and (-2+0j)
```

6. Program to check whether a given number is perfect number or not(sum of factors =number)

PROGRAM:

```
n = int(input(" Please Enter any Number: "))
Sum = 0
for i in range(1, n):
    if(n % i == 0):
        Sum = Sum + i
if (Sum == n):
    print(" %d is a Perfect Number" %n)
else:
    print(" %d is not a Perfect Number" %n)
```

OUTPUT:



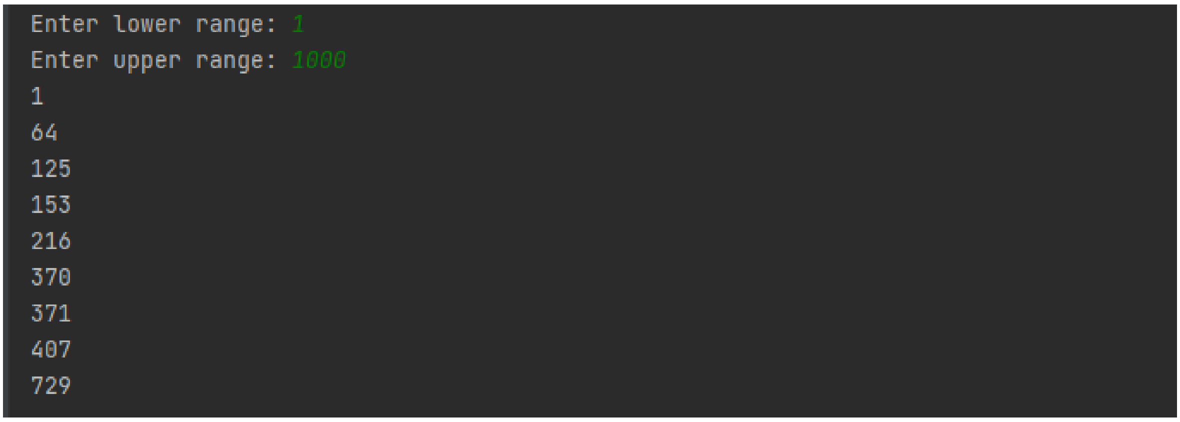
```
Please Enter any Number: 6
6 is a Perfect Number
```

7. Program to display amstrong numbers upto 1000

PROGRAM:

```
lower = int(input("Enter lower range: "))
upper = int(input("Enter upper range: "))

for num in range(lower, upper + 1):
    sum = 0
    temp = num
    while temp > 0:
        digit = temp % 10
        sum += digit ** 3
        temp //= 10
        if num == sum:
            print(num)
```

OUTPUT:



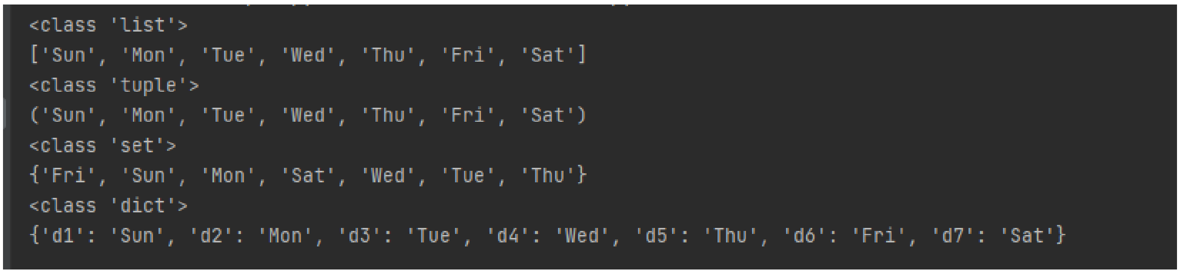
```
Enter lower range: 1
Enter upper range: 1000
1
64
125
153
216
370
371
407
729
```

8. Store and display the days of a week as a List, Tuple, Dictionary, Set. Also demonstrate different ways to store values in each of them. Display its type also.

PROGRAM:

```
list = ["Sun","Mon","Tue","Wed","Thu","Fri","Sat"]
print(type(list))
print(list)
tuple = ("Sun","Mon","Tue","Wed","Thu","Fri","Sat")
print(type(tuple))
print(tuple)
set = {"Sun","Mon","Tue","Wed","Thu","Fri","Sat"}
print(type(set))
print(set)
dict = {
    "d1" : "Sun",
    "d2" : "Mon",
    "d3" : "Tue",
    "d4" : "Wed",

 "d5" : "Thu",
  "d6" : "Fri",
    "d7" : "Sat"
}
print(type(dict))
print(dict)
```

OUTPUT:



```
<class 'list'>
['Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat']
<class 'tuple'>
('Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat')
<class 'set'>
{'Fri', 'Sun', 'Mon', 'Sat', 'Wed', 'Tue', 'Thu'}
<class 'dict'>
{'d1': 'Sun', 'd2': 'Mon', 'd3': 'Tue', 'd4': 'Wed', 'd5': 'Thu', 'd6': 'Fri', 'd7': 'Sat'}
```

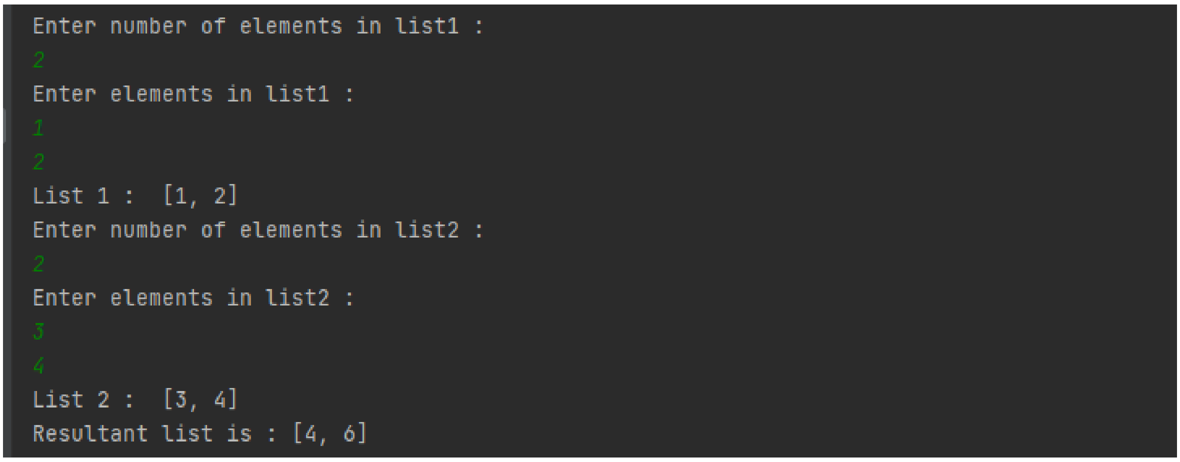
9. Write a program to add elements of given 2 lists

PROGRAM:

```
l1 = []
print("Enter number of elements in list1 : ")
```

```
n1 = int(input())
print("Enter elements in list1 : ")
for i in range(0, n1):
    ele1 = int(input())
    l1.append(ele1)
print("List 1 : ", l1)
l2 = []
print("Enter number of elements in list2 : ")
n2 = int(input())

print("Enter elements in list2 : ")
for i in range(0, n2):
    ele2 = int(input())
    l2.append(ele2)
print("List 2 : ", l2)
result = []
for i in range(0, len(l1)):
    result.append(l1[i] + l2[i])
print("Resultant list is : " + str(result))
```

OUTPUT:



```
Enter number of elements in list1 :
2
Enter elements in list1 :
1
2
List 1 :  [1, 2]
Enter number of elements in list2 :
2
Enter elements in list2 :
3
4
List 2 :  [3, 4]
Resultant list is : [4, 6]
```

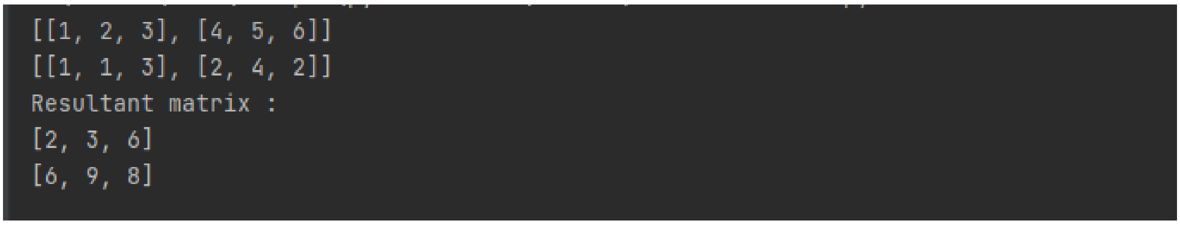
10. Write a program to find the sum of 2 matrices using nested List.

PROGRAM:

```
a = []
a1 = [1, 2, 3]
a.append(a1)
a2 = [4, 5, 6]
a.append(a2)
print(a)
b =[]
b1 = [1, 1, 3]
b.append(b1)
b2 = [2, 4, 2]
b.append(b2)
print(b)
result = [[0,0,0],
          [0,0,0]]
print("Resultant matrix : ")
```

```
for i in range(len(a)):
  for j in range(len(a[0])):
     result[i][j] = a[i][j] + b[i][j]
for r in result:
  print(r)
```

OUTPUT:



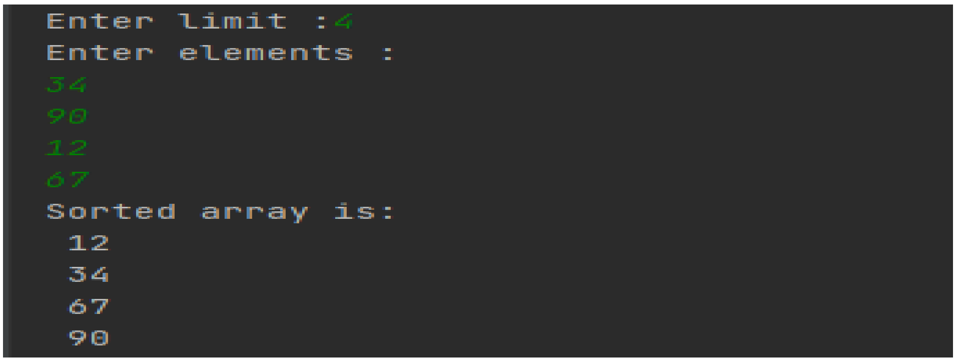
```
[[1, 2, 3], [4, 5, 6]]
[[1, 1, 3], [2, 4, 2]]
Resultant matrix :
[2, 3, 6]
[6, 9, 8]
```

11. Write a program to perform bubble sort on a given set of elements.

PROGRAM:

```
def bubbleSort(arr):
   for i in range(n - 1):
      for j in range(0, n - i - 1):
         if arr[j] > arr[j + 1]:
            arr[j], arr[j + 1] = arr[j + 1], arr[j]
arr = []
n = int(input("Enter limit :"))
print("Enter elements :")
for i in range(0,n):
   arr.append(int(input()))
bubbleSort(arr)
print("Sorted array is:")
for i in range(len(arr)):
   print("% d" % arr[i])
```

OUTPUT:



```
Enter limit :4
Enter elements :
34
90
12
67
Sorted array is:
 12
 34
 67
 90
```

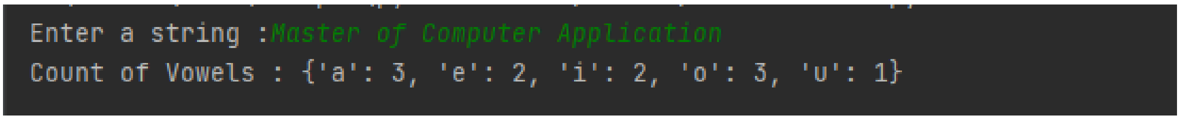
12. Program to find the count of each vowel in a string(use dictionary)

PROGRAM:

```
string = input("Enter a string :")
lowercase = string.lower()
vowel_counts = {}
for vowel in "aeiou":
   count = lowercase.count(vowel)
```

```
    vowel_counts[vowel] = count
print("Count of Vowels :", vowel_counts)
```

OUTPUT:



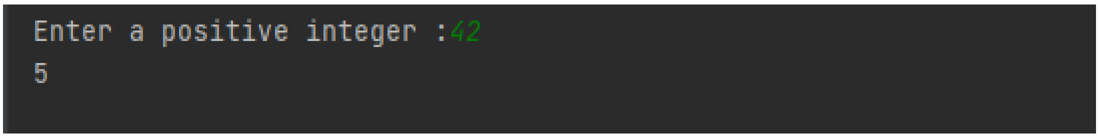
```
Enter a string :Master of Computer Application
Count of Vowels : {'a': 3, 'e': 2, 'i': 2, 'o': 3, 'u': 1}
```

13. Write a Python program that accept a positive number and subtract from this number the sum of its digits and so on. Continues this operation until the number is positive

PROGRAM:

```
def repeat_times(n):
  s = 0
  n_str = str(n)
  while (n > 0):
    n -= sum([int(i) for i in list(n_str)])
    n_str = list(str(n))
    s += 1
  return s
n=int(input("Enter a positive integer :"))
print(repeat_times(n))
```

OUTPUT:



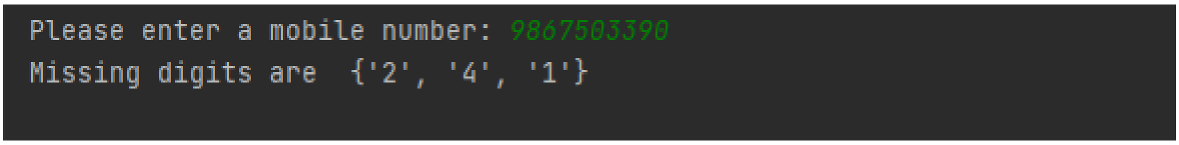
```
Enter a positive integer :42
5
```

14. Write a Python program that accepts a 10-digit mobile number, and find the digits which are absent in a given mobile number

PROGRAM:

```
mobile = input('Please enter a mobile number: ' )
all = '0123456789'
print('Missing digits are ', set(all) - set(mobile))
```

OUTPUT:



```
Please enter a mobile number: 9867503390
Missing digits are  {'2', '4', '1'}
```

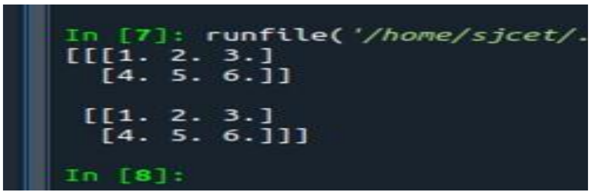
CYCLE 2

1. **Create a three-dimensional array specifying float data type and print it.**

PROGRAM:

```
import numpy as np

arr=np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]],dtype='f')
print(arr)
```

OUTPUT:



```
In [7]: runfile('/home/sjcet/.
[[[1. 2. 3.]
  [4. 5. 6.]]

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

In [8]:
```

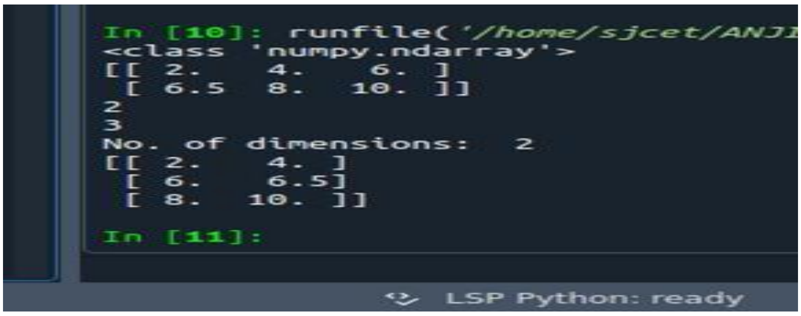
2. **Create a 2-dimensional array (2X3) with elements belonging to complex data type and print it. Also display**

a. the no: of rows and columns
b. dimension of an array
c. reshape the same array to 3X2

PROGRAM:

```
import numpy as np
x = np.array([[2, 4, 6], [6.5, 8, 10]])
print(type(
x))
print(x)
numOfRows = np. size(x, 0)
print(numOfRows)
numOfColumns = np. size(x,
1) print(numOfColumns)
print("No. of dimensions: ",
x.ndim) rs=np.reshape(x, (3, 2))
print(rs)
```

OUTPUT:



```
In [10]: runfile('/home/sjcet/ANJI
<class 'numpy.ndarray'>
[[ 2.    4.    6. ]
 [ 6.5   8.   10. ]]
2
3
No. of dimensions:  2
[[ 2.    4. ]
 [ 6.    6.5]
 [ 8.   10. ]]

In [11]:
```

3. Familiarize with the functions to create

a) an uninitialized array

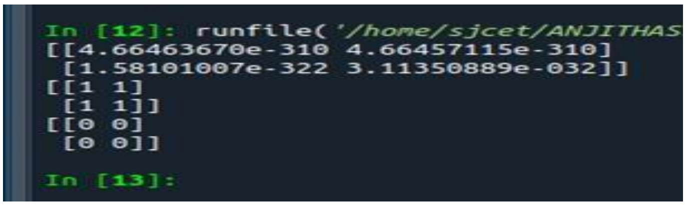
b) array with all elements as 1,

c)all elements as 0

PROGRAM:

```
import numpy as np

x=np.empty([2, 2])

print(x)

y=np.full((2, 2), 1)

print(y)

z=np.full((2, 2), 0)

print(z)
```

OUTPUT:



```
In [12]: runfile('/home/sjcet/ANJITHAS
[[4.66463670e-310 4.66457115e-310]
 [1.58101007e-322 3.11350889e-032]]
[[1 1]
 [1 1]]
[[0 0]
 [0 0]]

In [13]:
```

4. Create an one dimensional array using arange function containing 10 elements.

Display

a) First 4 elements

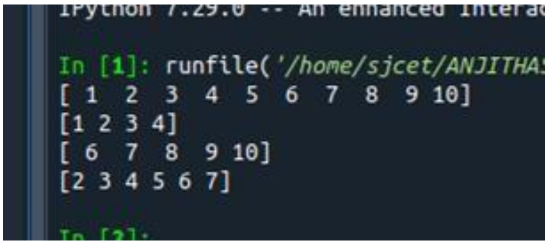
b) Last 6 elements

c) Elements from index 2 to 7

PROGRAM

```
import numpy as np

a = np.arange(1, 11, 1)

print(a)

first_element = a[:4]

print(first_element)

first_element1 = a[5:]

print(first_element1)

first_element2 = a[1:7]

print(first_element2)
```

OUTPUT



```
IPython 7.29.0 -- An enhanced Interac

In [1]: runfile('/home/sjcet/ANJITHA
[ 1  2  3  4  5  6  7  8  9 10]
[1 2 3 4]
[ 6  7  8  9 10]
[2 3 4 5 6 7]

In [2]:
```

5. **Create an 1D array with arange containing first 15 even numbers as elements**
   a. **Elements from index 2 to 8 with step 2(also demonstrate the same using slice function)**
   b. **Last 3 elements of the array using negative index**
   c. **Alternate elements of the array**
   d. **Display the last 3 alternate elements**

PROGRAM:

import numpy as np

a = np.arange(0, 15, 2) print(a)

print("Elements from index 2 to 8 with step 2")

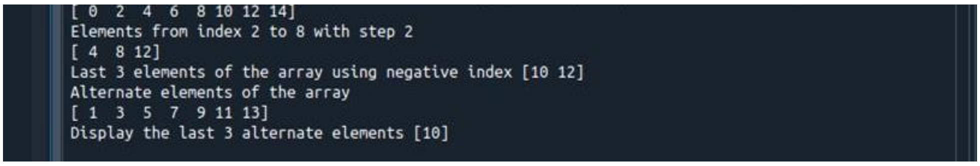
s2 = slice(2, 8, 2) print(a[s2])

print("Last 3 elements of the array using negative index",a[-3:-1]) print("Alternate elements of the array")

ab = np.arange(1, 15, 2) print(ab)

print("Display the last 3 alternate elements",a[-3:-1:2])

OUTPUT:



```
[ 0  2  4  6  8 10 12 14]
Elements from index 2 to 8 with step 2
[ 4  8 12]
Last 3 elements of the array using negative index [10 12]
Alternate elements of the array
[ 1  3  5  7  9 11 13]
Display the last 3 alternate elements [10]
```

6. **Create a 2 Dimensional array with 4 rows and 4 columns.**
   a. **Display all elements excluding the first row**
   b. **Display all elements excluding the last column**
   c. **Display the elements of 1 st and 2 nd column in 2 nd and 3 rd row**
   d. **Display the elements of 2 nd and 3 rd column**
   e. **Display 2 nd and 3 rd element of 1 st row**

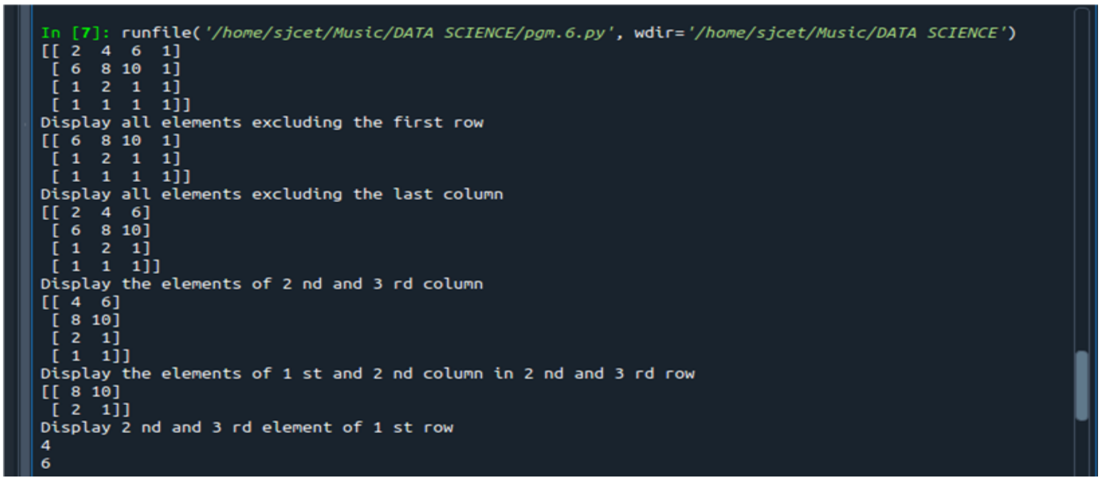
PROGRAM

import numpy as np

x = np.array([[2, 4, 6,1], [6, 8, 10,1],[1, 2, 1,1], [1, 1, 1,1]])

```
print(x)
print("Display all elements excluding the first row") print(x[1:])
print("Display all elements excluding the last column") print(x[:, :3])
print("Display the elements of 2 nd and 3 rd column") print(x[:, 1:3])
print("Display the elements of 1 st and 2 nd column in 2 nd and 3 rd row")
print(x[1:3,1:3])
print("Display 2 nd and 3 rd element of 1 st row") print(x[0,1])
print(x[0,2])
```

OUTPUT



```
In [7]: runfile('/home/sjcet/Music/DATA SCIENCE/pgm.6.py', wdir='/home/sjcet/Music/DATA SCIENCE')
[[ 2  4  6  1]
 [ 6  8 10  1]
 [ 1  2  1  1]
 [ 1  1  1  1]]
Display all elements excluding the first row
[[ 6  8 10  1]
 [ 1  2  1  1]
 [ 1  1  1  1]]
Display all elements excluding the last column
[[ 2  4  6]
 [ 6  8 10]
 [ 1  2  1]
 [ 1  1  1]]
Display the elements of 2 nd and 3 rd column
[[ 4  6]
 [ 8 10]
 [ 2  1]
 [ 1  1]]
Display the elements of 1 st and 2 nd column in 2 nd and 3 rd row
[[ 8 10]
 [ 2  1]]
Display 2 nd and 3 rd element of 1 st row
4
6
```

7. Create two 2D arrays using array object and

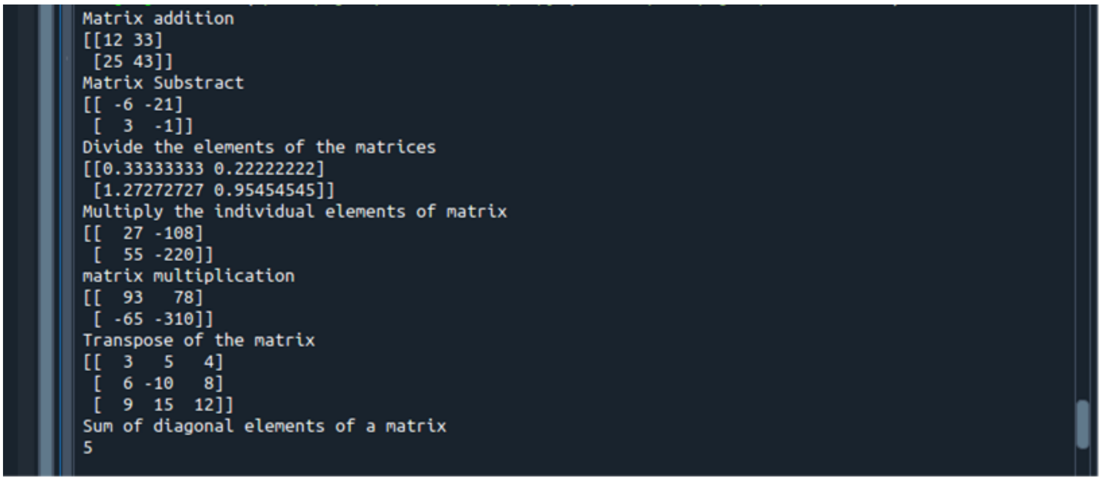
- **a) Add the 2 matrices and print it**
- **b) Subtract 2 matrices**
- **c) Multiply the individual elements of matrix**
- **d) Divide the elements of the matrices**
- **e) Perform matrix multiplication**
- **f) Display transpose of the matrix**
- **g) Sum of diagonal elements of a matrix**

PROGRAM:

```
import numpy as np
M1 = np.array([[3, 6], [14, 21]])
M2 = np.array([[9, 27], [11, 22]])
M3 = M1 + M2
print("Matrix addition") print(M3)
M1 = np.array([[3, 6], [14, 21]])
```

```
M2 = np.array([[9, 27], [11, 22]])
M3 = M1 - M2
print("Matrix Substract") print(M3)
M1 = np.array([[3, 6], [14, 21]])
M2 = np.array([[9, 27], [11, 22]])
M3 = M1 / M2
print("Divide the elements of the matrices") print(M3)
M1 = np.array([[3, 6], [5, -10]])
M2 = np.array([[9, -18], [11, 22]])
M3 = M1 * M2
print("Multiply the individual elements of matrix") print(M3)
M1 = np.array([[3, 6], [5, -10]])
M2 = np.array([[9, -18], [11, 22]]) M3 = M1.dot(M2)
print("matrix multiplication") print(M3)
M1 = np.array([[3, 6, 9], [5, -10, 15], [4,8,12]])
M2 = M1.transpose() print("Transpose of the matrix") print(M2)
M1 = np.array([[3, 6, 9], [5, -10, 15], [4,8,12]])
print("Sum of diagonal elements of a matrix") print(np.trace(M1))
```

OUTPUT



```
Matrix addition
[[12 33]
 [25 43]]
Matrix Substract
[[ -6 -21]
 [  3  -1]]
Divide the elements of the matrices
[[0.33333333 0.22222222]
 [1.27272727 0.95454545]]
Multiply the individual elements of matrix
[[  27 -108]
 [  55 -220]]
matrix multiplication
[[  93   78]
 [ -65 -310]]
Transpose of the matrix
[[  3   5   4]
 [  6 -10   8]
 [  9  15  12]]
Sum of diagonal elements of a matrix
5
```

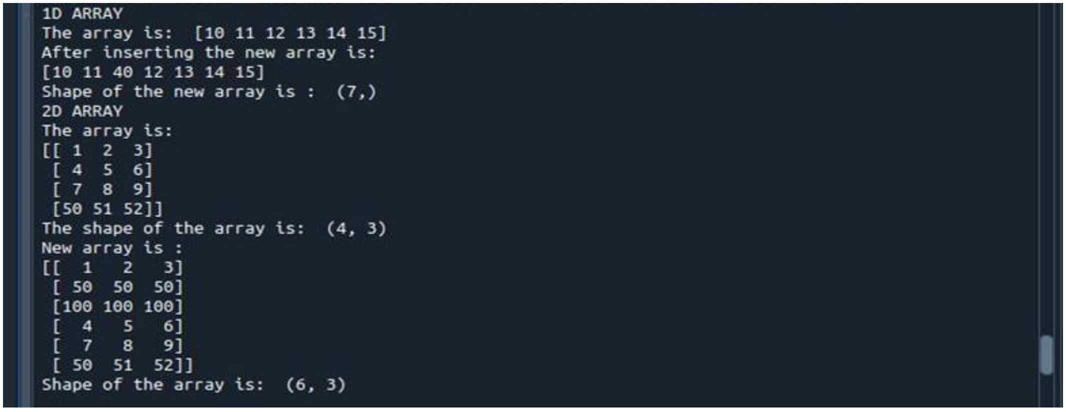
8. Demonstrate the use of insert() function in 1D and 2D array

PROGRAM

```
import numpy as np
arr1 = np.arange(10,
16)
```

```
print("1D ARRAY")
print("The array is: ",
arr1)
obj = 2
value = 40
arr = np.insert(arr1, obj, value,
axis=None) print("After inserting the
new array is: ") print(arr)
print("Shape of the new array is : ",
np.shape(arr)) print("2D ARRAY ")
arr1 = np.array([(1, 2, 3), (4, 5, 6), (7, 8, 9), (50, 51, 52)])
print("The array is:
") print(arr1)
print("The shape of the array is: ",
np.shape(arr1)) a = np.insert(arr1, 1, [[50],
[100], ], axis=0) print("New array is : ")
print(a)
print("Shape of the array is: ", np.shape(a))
```

OUTPUT



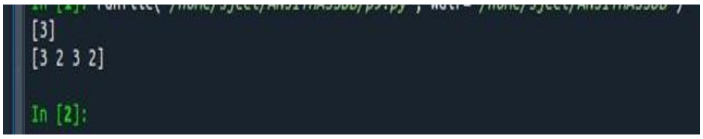
```
1D ARRAY
The array is:  [10 11 12 13 14 15]
After inserting the new array is:
[10 11 40 12 13 14 15]
Shape of the new array is :  (7,)
2D ARRAY
The array is:
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [50 51 52]]
The shape of the array is:  (4, 3)
New array is :
[[  1   2   3]
 [ 50  50  50]
 [100 100 100]
 [  4   5   6]
 [  7   8   9]
 [ 50  51  52]]
Shape of the array is:  (6, 3)
```

9. Demonstrate the use of diag() function in 1D and 2D array.

PROGRAM

```
import numpy as np
a= np.array([[3, 6,7,8]])
b=np.array([[3, 6,8,7], [4, 2,1,0],[3,1,3,3],[1,1,2,2]])
x=np.diag(
)
y=np.diag(
) print(x)
print(y)
```

OUTPUT



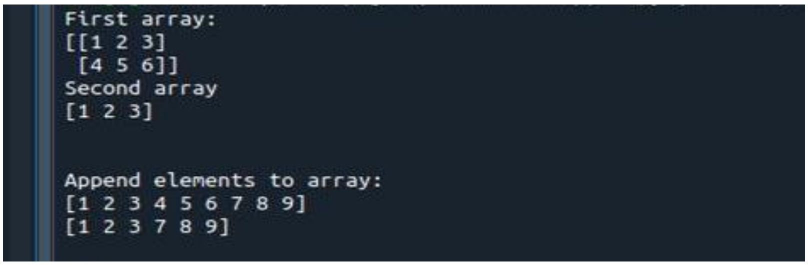
```
[3]
[3 2 3 2]

In [2]:
```

10. Demonstarte the use of append() function in 1D and 2D array.

PROGRAM

```
import numpy as np
a = np.array([[1,2,3],[4,5,6]])
b=np.array([1,2,3]) print("First array:") print (a) print("Second array") print(b)
print ("\n")
print ("Append elements to array:") print (np.append(a, [7,8,9]))
print (np.append(b, [7,8,9]))
```

OUTPUT



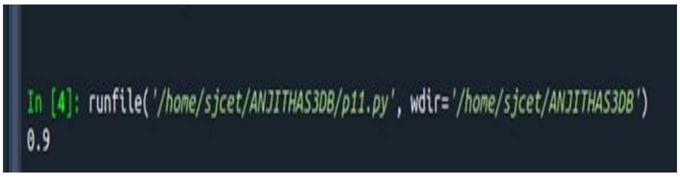
```
First array:
[[1 2 3]
 [4 5 6]]
Second array
[1 2 3]


Append elements to array:
[1 2 3 4 5 6 7 8 9]
[1 2 3 7 8 9]
```

11. Demonstarte the use of sum() function in 1D and 2D array.

PROGRAM

```
import numpy as np
a=np.array([0.4,0.5])
b=np.sum(a)
print (b)
```

OUTPUT



```
In [4]: runfile('/home/sjcet/ANJITHAS3DB/p11.py', wdir='/home/sjcet/ANJITHAS3DB')
0.9
```

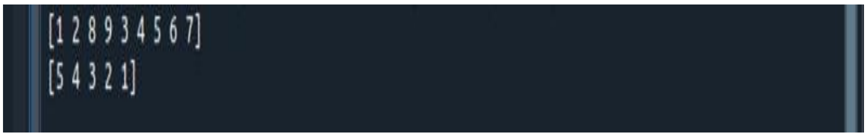
12. Display the elements from indices 4 to 10 in descending order(use – values)

Program

```
import numpy as np
a = np.array([1,2,8,9,3,4,5,6,7])
print(a)
array_copy = np.sort(a)[::-1]
```

```
print(array_copy[4:10])
```

OUTPUT



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

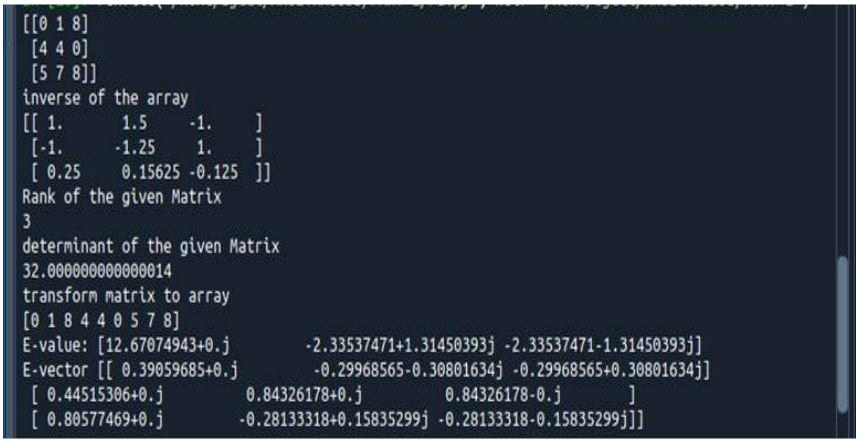
13. Create a square matrix with random integer values(use randint()) and use appropriate functions to find:

- **a. Inverse**
- **b. rank of matrix**
- **c. Determinant**
- **d. transform matrix into 1D array**
- **e. eigen values and vectors**

PROGRAM

```
import numpy as np

import numpy as nf

from numpy.linalg import eig

mat = np.random.randint(10, size=(3, 3))

array = nf.random.randint(10, size=(3, 3))

print(mat)

M_inverse = np.linalg.inv(mat)

print("inverse of the array")

print(M_inverse)

rank = np.linalg.matrix_rank(mat)

print("Rank of the given Matrix ")

print(rank)

det= np.linalg.det(mat)

print("determinant of the given Matrix ")

print(det)

arr=mat.flatten()

print("transform matrix to array ")

print(arr) w,v=eig(array)

print('E-value:', w)

print('E-vector', v)
```

OUTPUT



```
[[0 1 8]
 [4 4 0]
 [5 7 8]]
inverse of the array
[[ 1.       1.5     -1.     ]
 [-1.      -1.25     1.     ]
 [ 0.25     0.15625 -0.125  ]]
Rank of the given Matrix
3
determinant of the given Matrix
32.000000000000014
transform matrix to array
[0 1 8 4 4 0 5 7 8]
E-value: [12.67074943+0.j         -2.33537471+1.31450393j -2.33537471-1.31450393j]
E-vector [[ 0.39059685+0.j         -0.29968565-0.30801634j -0.29968565+0.30801634j]
 [ 0.44515306+0.j          0.84326178+0.j          0.84326178-0.j        ]
 [ 0.80577469+0.j         -0.28133318+0.15835299j -0.28133318-0.15835299j]]
```

14. Create a matrix X with suitable rows and columns

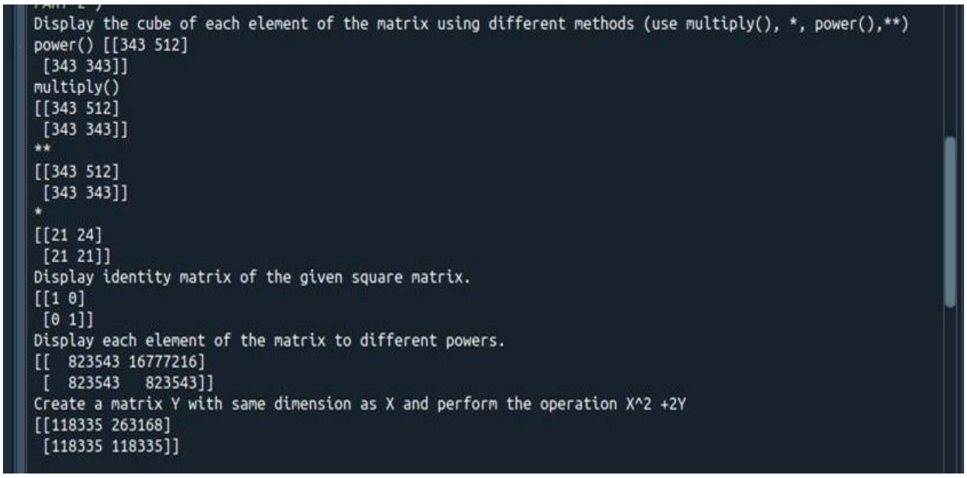
- **a) Display the cube of each element of the matrix using different methods (use multiply(), *, power(), **)**
- **b) Display identity matrix of the given square matrix.**
- **c) Display each element of the matrix to different powers.**
- **d) Create a matrix Y with same dimension as X and perform the operation X^2+2Y**

PROGRAM

```
import numpy as np

matrix=np.random.randint(0,10,4).reshape(2,2)

print("Display the cube of each element of the matrix using different methods (use multiply(), *, power(),**)")

x=np.power(matrix,3)

print("power()",x)

y=np.multiply(matrix,(matrix*matrix))

print("multiply()")

print(y)

z=matrix*matrix*matrix print("**")

print(z)

cube=matrix*3

print("*")

print(cube)

print("Display identity matrix of the given square matrix.")

identity=np.identity(2,dtype=int)

print(identity)

print("Display each element of the matrix to different powers.")
```

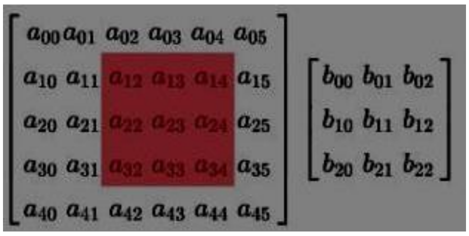
```
dpow=np.power(matrix,matrix)
print(dpow)
print("Create a matrix Y with same dimension as X and perform the operation X^2+2Y")
a=np.add((np.power(x,2)),(np.multiply(y,2)))
print(a)
```

<u>OUTPUT</u>



```
Display the cube of each element of the matrix using different methods (use multiply(), *, power(),**)
power() [[343 512]
 [343 343]]
multiply()
[[343 512]
 [343 343]]
**
[[343 512]
 [343 343]]
*
[[21 24]
 [21 21]]
Display identity matrix of the given square matrix.
[[1 0]
 [0 1]]
Display each element of the matrix to different powers.
[[  823543 16777216]
 [  823543   823543]]
Create a matrix Y with same dimension as X and perform the operation X^2 +2Y
[[118335 263168]
 [118335 118335]]
```

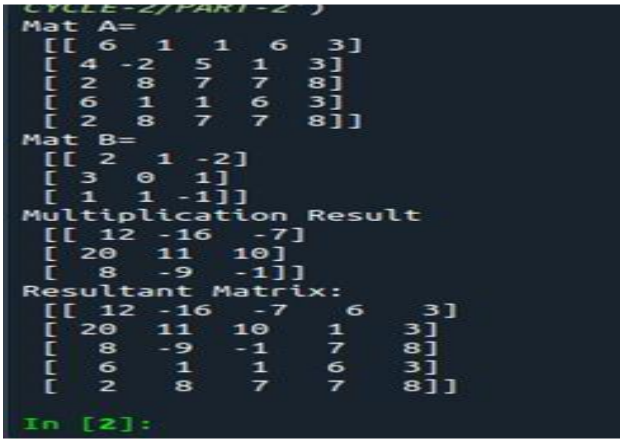
15. Multiply a matrix with a submatrix of another matrix and replace the samein larger matrix.


a_{00} a_{01} a_{02} a_{03} a_{04} a_{05}
a_{10} a_{11} a_{12} a_{13} a_{14} a_{15}
a_{20} a_{21} a_{22} a_{23} a_{24} a_{25}
a_{30} a_{31} a_{32} a_{33} a_{34} a_{35}
a_{40} a_{41} a_{42} a_{43} a_{44} a_{45}
b_{00} b_{01} b_{02}
b_{10} b_{11} b_{12}
b_{20} b_{21} b_{22}

<u>PROGRAM</u>

```
import numpy as np
A = np.array([[6, 1, 1,6,3],
          [4, -2, 5,1,3],
          [2, 8, 7,7,8],
          [6, 1, 1,6,3],
          [2, 8, 7,7,8]])
B=np.array([[2, 1, -2],
          [3, 0, 1],
          [1, 1, -1]])
  print("MatA=\n",A)
print("MatB=\n",B)
C=A[:3, :3]
res = np.dot(B,C)
print("Multiplication Result\n",res)
A[:3,:3]=res[:3,:3]
print("Resultant Matrix:\n",A)
```

<u>OUTPUT</u>



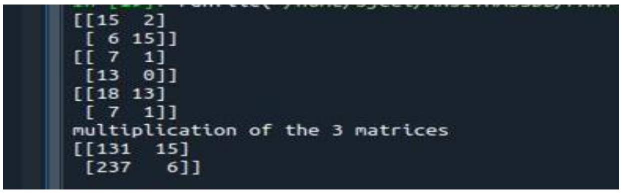
```
Mat A=
 [[ 6  1  1  6  3]
 [ 4 -2  5  1  3]
 [ 2  8  7  7  8]
 [ 6  1  1  6  3]
 [ 2  8  7  7  8]]
Mat B=
 [[ 2  1 -2]
 [ 3  0  1]
 [ 1  1 -1]]
Multiplication Result
 [[ 12 -16  -7]
 [ 20  11  10]
 [  8  -9  -1]]
Resultant Matrix:
 [[ 12 -16  -7   6   3]
 [ 20  11  10   1   3]
 [  8  -9  -1   7   8]
 [  6   1   1   6   3]
 [  2   8   7   7   8]]

In [2]:
```

16. **Given 3 MatricesA, B and C. Write a program to perform matrix multiplication of the 3 matrices.**

<u>PROGRAM</u>

```
import numpy as np
m1 = np.random.randint(20, size=(2, 2)) print(m1)
m2 = np.random.randint(20, size=(2, 2)) print(m2)
m3 = np.random.randint(20, size=(2, 2)) print(m3)
print("multiplication of the 3 matrices")
m4 = np.dot(m1,m2,m3)
print(m4)
```

Output



```
[[15  2]
 [ 6 15]]
[[ 7  1]
 [13  0]]
[[18 13]
 [ 7  1]]
multiplication of the 3 matrices
[[131  15]
 [237   6]]
```

17. **Write a program to check whether given matrix is symmetric or Skew Symmetric. Solving systems of equations with numpy One of the more common problems in linear algebra is solving a matrix-vector equation.Here is an example. We seek the vector x that solves the equation**
    A X = b

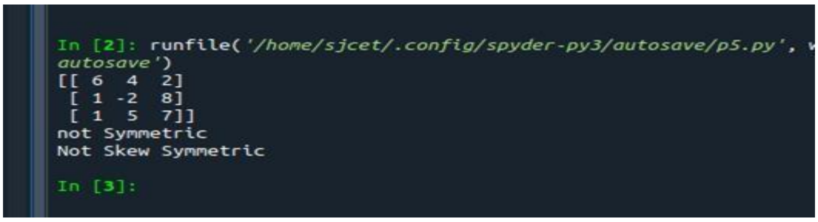
$$A = \begin{bmatrix} 2 & 1 & -2 \\ 3 & 0 & 1 \\ 1 & 1 & -1 \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} -3 \\ 5 \\ -2 \end{bmatrix}$$

And $X=A^{-1}$ b. NumPy provides a function called solve for solving such equations.

PROGRAM

```
import numpy as np
A = np.array([[6, 1, 1],
          [4, -2, 5],
          [2, 8, 7]])
inv=np.transpose(A)
 print (inv)
neg=np.negative(A)
comparison = A == inv
comparison1 = inv== neg
equal_arrays = comparison.all()
skew=comparison1.all()
if equal_arrays :
          print("Symmetric")
else:
          print("not Symmetric")
if skew:
          print("Skew Symmetric")
else:
          print("Not Skew Symmetric")
```

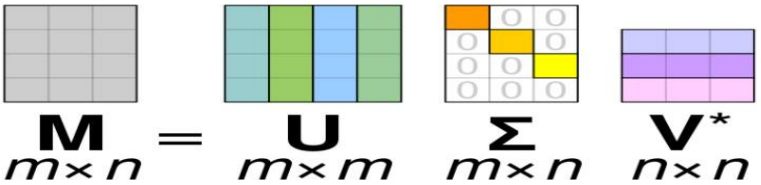
OUTPUT



```
In [2]: runfile('/home/sjcet/.config/spyder-py3/autosave/p5.py', w
autosave')
[[ 6  4  2]
 [ 1 -2  8]
 [ 1  5  7]]
not Symmetric
Not Skew Symmetric

In [3]:
```

18. Write a program to find out the value of X using solve(), given A and b as above Singular value Decomposition Matrix decomposition, also known as matrix fac- torization, involves describing a given matrix using its constituent elements.The Sin- gular-Value Decomposition, or SVD for short, is a matrix decomposition method for reducing a matrix to its constituent parts in order to make certain subsequent matrix calculations simpler. This approach is commonly used in reducing the no: of attributes in the given data set.

M= U ∑V^T



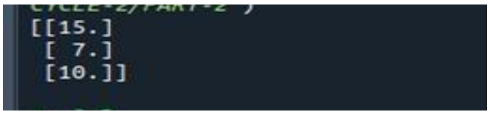
- **M-is original matrix we want to decompose**
  - **U-is left singular matrix (columns are left singular vectors). U columns contain eigenvectors of matrix MM^t**
  - **Σ-is a diagonal matrix containing singular (eigen) values.**
  - **V-is right singular matrix (columns are right singular vectors). V columns con- tain eigenvectors of matrix M^{t}M**

Numpy provides a function for performing svd, which decomposes the given matrix into 3 matrices.

<u>PROGRAM</u>

```
import numpy as np

A = np.array([[2, 1, -2],

            [3, 0, 1],[1, 1, -1]])

b=np.array([[3],[5],[-2]])

inv=np.linalg.inv(A)

x=np.linalg.solve(inv,b)

print(x)
```

<u>OUTPUT</u>



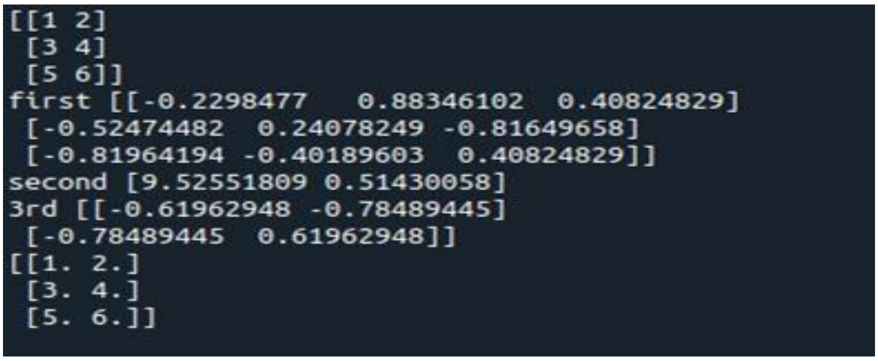
```
CYCLE-2/PART-2 )
[[15.]
 [ 7.]
 [10.]]
```

19. Write a program to perform the SVD of a given matrix. Also reconstruct the given matrix from the 3 matrices obtained after performing SVD.

<u>PROGRAM</u>

```
from numpy import array
from scipy.linalg import svd
from numpy import diag
from numpy import dot from
numpy import zeros

A = array([[1, 2], [3, 4],[5,6]])

print(A)

# SVD

U, s, VT = svd(A)

print("first" ,U)

print("second",s)

print("3rd" ,VT)

Sigma = zeros((A.shape[0], A.shape[1]))

# populate Sigma with n x n diagonal matrix

Sigma[:A.shape[1], :A.shape[1]] = diag(s)

# reconstruct matrix

B = U.dot(Sigma.dot(VT))

print(B)
```

OUTPUT



```
[[1 2]
 [3 4]
 [5 6]]
first [[-0.2298477   0.88346102  0.40824829]
 [-0.52474482  0.24078249 -0.81649658]
 [-0.81964194 -0.40189603  0.40824829]]
second [9.52551809 0.51430058]
3rd [[-0.61962948 -0.78489445]
 [-0.78489445  0.61962948]]
[[1. 2.]
 [3. 4.]
 [5. 6.]]
```

CYCLE 3

1. **Sarah bought a new car in 2001 for $24,000. The dollar value of her car changed each year as shown in the table below.**

Value of Sarah's Car

Year	**Value**
2001	**$24,000**
2002	**$22,500**
2003	**$19,700**
2004	**$17,500**
2005	**$14,500**
2006	**$10,000**
2007	**$ 5,800**

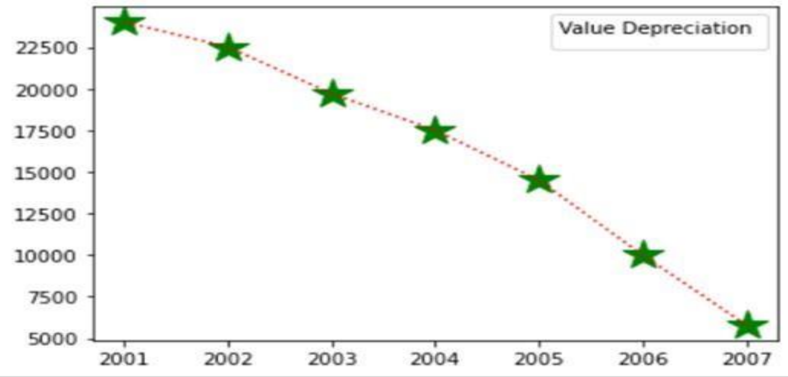
Represent the following information using a line graph with following style properties

X- axis – Year, Y –axis - Car Value, title –Value Depreciation (left Aligned)

Line Style dashdot and Line-color should be red, point using * symbol with green color and size 20 . Subplot() provides multiple plots in one figure.

PROGRAM:

```
import matplotlib.pyplot as plt

import numpy as np

xpoints = np.array([2001, 2002,2003,2004,2005,2006,2007])

ypoints = np.array([24000, 22500,19700,17500,14500,10000,5800])

plt.plot(xpoints, ypoints, '*g',ms = 20)

plt.plot(xpoints, ypoints, ':r')

leg = plt.legend(title="Value Depreciation ")

leg._legend_box.align = "left"

plt.show()
```



2. **Following table gives the daily sales of the following items in a shop**

Mon	Tues	Wed	Thurs	Fri
300	450	150	400	650
400	500	350	300	500

Use subplot function to draw the line graphs with grids(color as blue and line style dotted) for the above information as 2 separate graphs in two rows

- **Properties for the Graph 1:**
  - **X label- Days of week**
  - **Y label-Sale of Drinks**
  - **Title-Sales Data1 (right aligned)**
  - **Line –dotted with cyan color**
  - **Points- hexagon shape with color magenta and outline black**

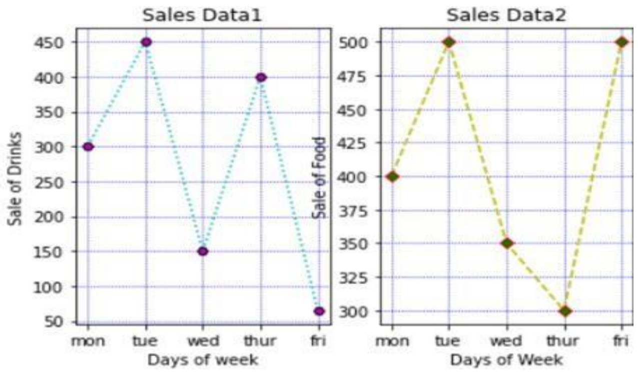
- **Properties for the Graph 2:**
  - **X label- Days of Week**
  - **Y label-Sale of Food**
  - **Title-Sales Data2 (center aligned)**
  - **Line –dashed with yellow color**

  Points- diamond shape with color green and outline red

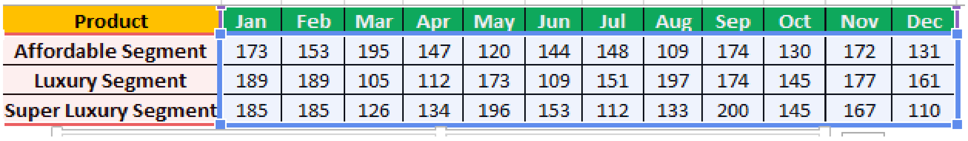
PROGRAM

```
import matplotlib.pyplot as plt
import numpy as np
#plot 1:

x = np.array(['mon', 'tue', 'wed', 'thur','fri'])

y = np.array([300, 450, 150, 400,65])

plt.subplot(1, 2, 1)

plt.title("Sales Data1")

plt.xlabel("Days of week")

plt.ylabel("Sale of Drinks")

plt.plot(x,y,':c')
plt.plot(x,y,'Hm',mec = 'k')
plt.grid(color = 'blue', linestyle = 'dotted')
#plot 2:
c = np.array(['mon', 'tue', 'wed', 'thur','fri'])
v = np.array([400, 500, 350, 300,500])
plt.subplot(1, 2, 2) plt.title("Sales Data2")
plt.xlabel("Days of Week")
plt.ylabel("Sale of Food")
plt.plot(c,v,'--y')
plt.plot(c,v,'Dg',mec = 'r')
plt.grid(color = 'blue', linestyle = 'dotted')
plt.show()
```

OUTPUT



3. **Create scatter plot for the below data:(use Scatter function)**



Product	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Affordable Segment	173	153	195	147	120	144	148	109	174	130	172	131
Luxury Segment	189	189	105	112	173	109	151	197	174	145	177	161
Super Luxury Segment	185	185	126	134	196	153	112	133	200	145	167	110

Create scatter plot for each Segment with following properties within one

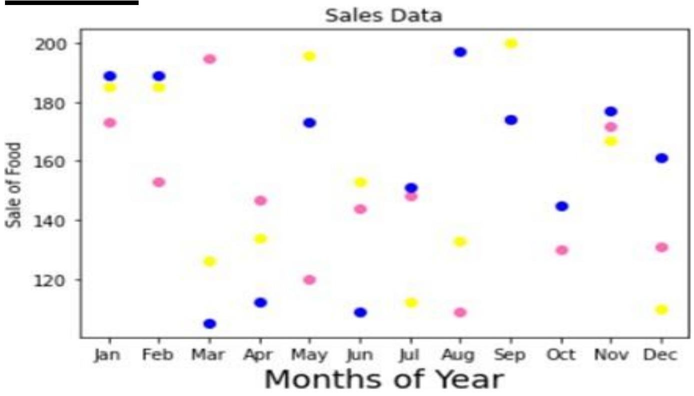
- **X Label- Months of Year with font size 18**
- **Y-Label- Sales of Segments**
- **Title –Sales Data**
- **Color for Affordable segment- pink**
- **Color for Luxury Segment- Yellow**
- **Color for Super luxury segment-blue**

PROGRAM

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

```
plt.title("Sales Data")
plt.xlabel("Months of Year"size=18)
plt.ylabel("Sale of Food")
x = np.array(['Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec'])
y1 = np.array([173,153,195,147,120,144,148,109,174,130,172,131])
plt.scatter(x,y1, color = 'hotpink')
y2 = np.array([185,185,126,134,196,153,112,133,200,145,167,110])
plt.scatter(x, y2, color = 'yellow')
y3 = np.array([189,189,105,112,173,109,151,197,174,145,177,161])
plt.scatter(x, y3, color = 'blue')
plt.show()
```

OUTPUT

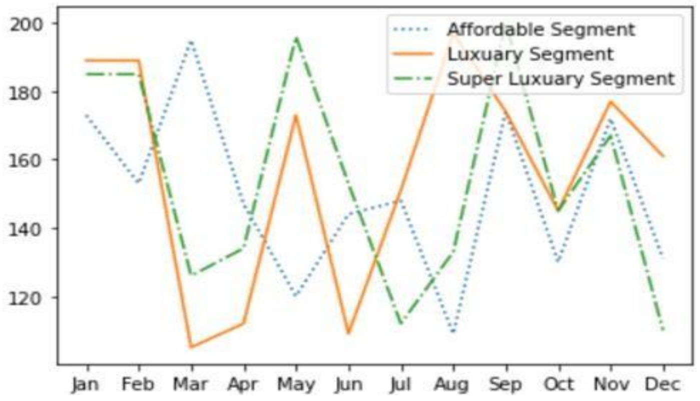


4. **Display the above data using multiline plot(3 different lines in same graph)**
   - **Display the description of the graph in upper right corner(use legend())**
   - **Use different colors and line styles for 3 different lines**

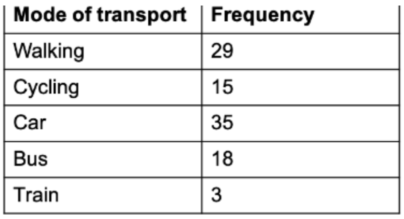
PROGRAM

```
 import matplotlib.pyplot as
 plt import numpy as np
x =
 np.array(['Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec'])
y1 = np.array([173,153,195,147,120,144,148,109,174,130,172,131])
 y2=
 np.array([189,189,105,112,173,109,151,197,174,145,177,161])
 y3=
 np.array([185,185,126,134,196,153,112,133,200,145,167,110])
plt.plot(x,y1,label = "Affordable Segment",ls=':')
 plt.plot(x,y2,label = "Luxuary Segment",ls="-")
 plt.plot(x,y3,label = "Super Luxuary Segment",ls="-
 .") plt.legend()
 plt.show()
```

OUTPUT



5. **100 students were asked what their primary mode of transport for getting to school was. The results of this survey are recorded in the table below. Construct a bar graph representing this information.**



Mode of transport	Frequency
Walking	29
Cycling	15
Car	35
Bus	18
Train	3

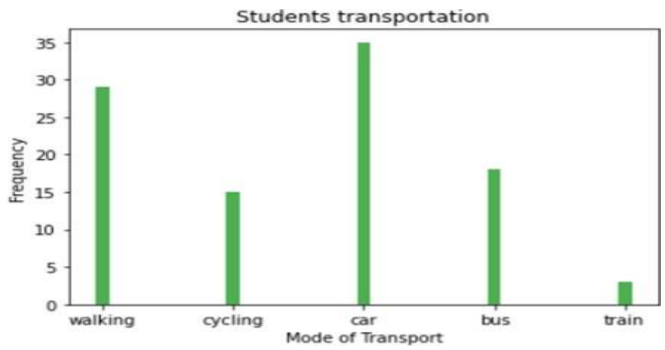
Create a bar graph with

- **X axis -mode of Transport and Y axis ‘frequency’**
- **Provide appropriate labels and title**
- **Width .1, color green**

PROGRAM

```
import matplotlib.pyplot as plt import numpy
as np plt.title("Students transportation")
plt.xlabel("Mode of Transport")
plt.ylabel("Frequency")
x = np.array(["walking","cycling","car","bus","train"])

y = np.array([29,15,35,18,3])

plt.bar(x, y, color = "#4CAF50",width = 0.1) plt.show()
```

OUTPUT

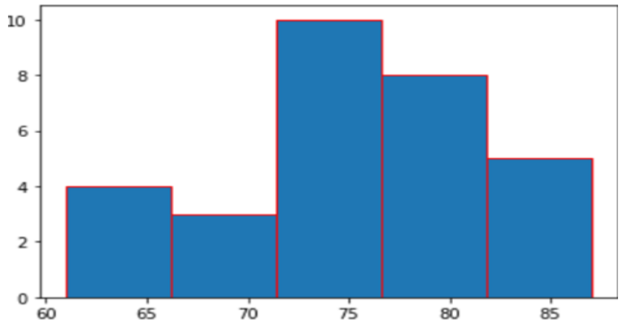


6. We are provided with the height of 30 cherry trees. The height of the trees (in inches): 61, 63, 64, 66, 68, 69, 71, 71.5, 72, 72.5,73,73.5, 74, 74.5, 76, 76.2,76.5, 77, 77.5, 78, 78.5, 79, 79.2, 80, 81, 82, 83, 84,85,87. Create a histogram with a bin size of 5

PROGRAM

```
import matplotlib.pyplot as plt

height=[61,63,64,66,68,69,71,71.5,7
2,72.5,73,73.5,74,74.5,76,76.2,76.5,
77,77.5,78,78.5,79,79.2,80,81,82,83,
84,85,87]

plt.hist(height, edgecolor="red", bins=5)
plt.show()
```

OUTPUT



Data Handling using 'Pandas' and Data Visualization using 'Seaborn'

7. Use appropriate functions in pandas to display

1.Shap of the data set

2. **First 5 and last five rows of data set(head and tail)**

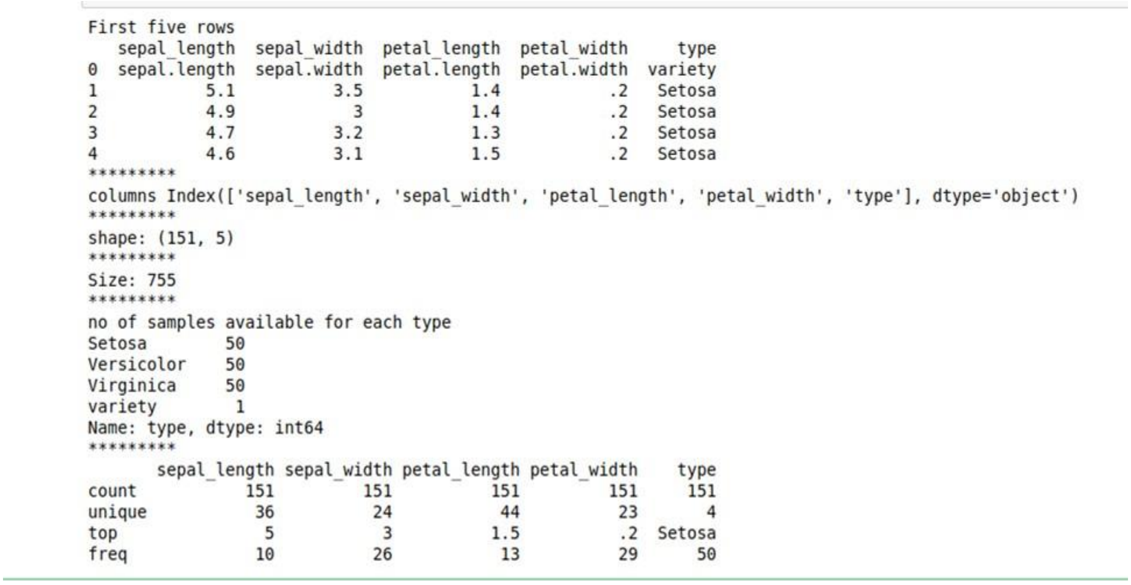
3.Size of dataset
4.No:of samples available for each variety
5.Description of the data set(use describe

PROGRAM

```
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

import pandas as pd
col=['sepal_length','sepal_width','petal_length','petal_width','type']
iris=pd.read_csv("iris.csv",names=col)
```

```
print("First five rows")
print(iris.head())
print("*********")
print("columns",iris.columns)
print("*********")
print("shape:",iris.shape)
print("*********")
print("Size:",iris.size)
print("*********")

print("no of samples available for each type")

print(iris["type"].value_counts())
print("*********")
print(iris.describe())
```

OUTPUT



```
First five rows
   sepal_length  sepal_width  petal_length  petal_width     type
0  sepal.length  sepal.width  petal.length  petal.width  variety
1           5.1          3.5           1.4           .2   Setosa
2           4.9            3           1.4           .2   Setosa
3           4.7          3.2           1.3           .2   Setosa
4           4.6          3.1           1.5           .2   Setosa
*********
columns Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'type'], dtype='object')
*********
shape: (151, 5)
*********
Size: 755
*********
no of samples available for each type
Setosa        50
Versicolor    50
Virginica     50
variety        1
Name: type, dtype: int64
*********
       sepal_length sepal_width petal_length petal_width    type
count           151         151          151         151     151
unique           36          24           44          23       4
top               5           3          1.5          .2  Setosa
freq             10          26           13          29      50
```

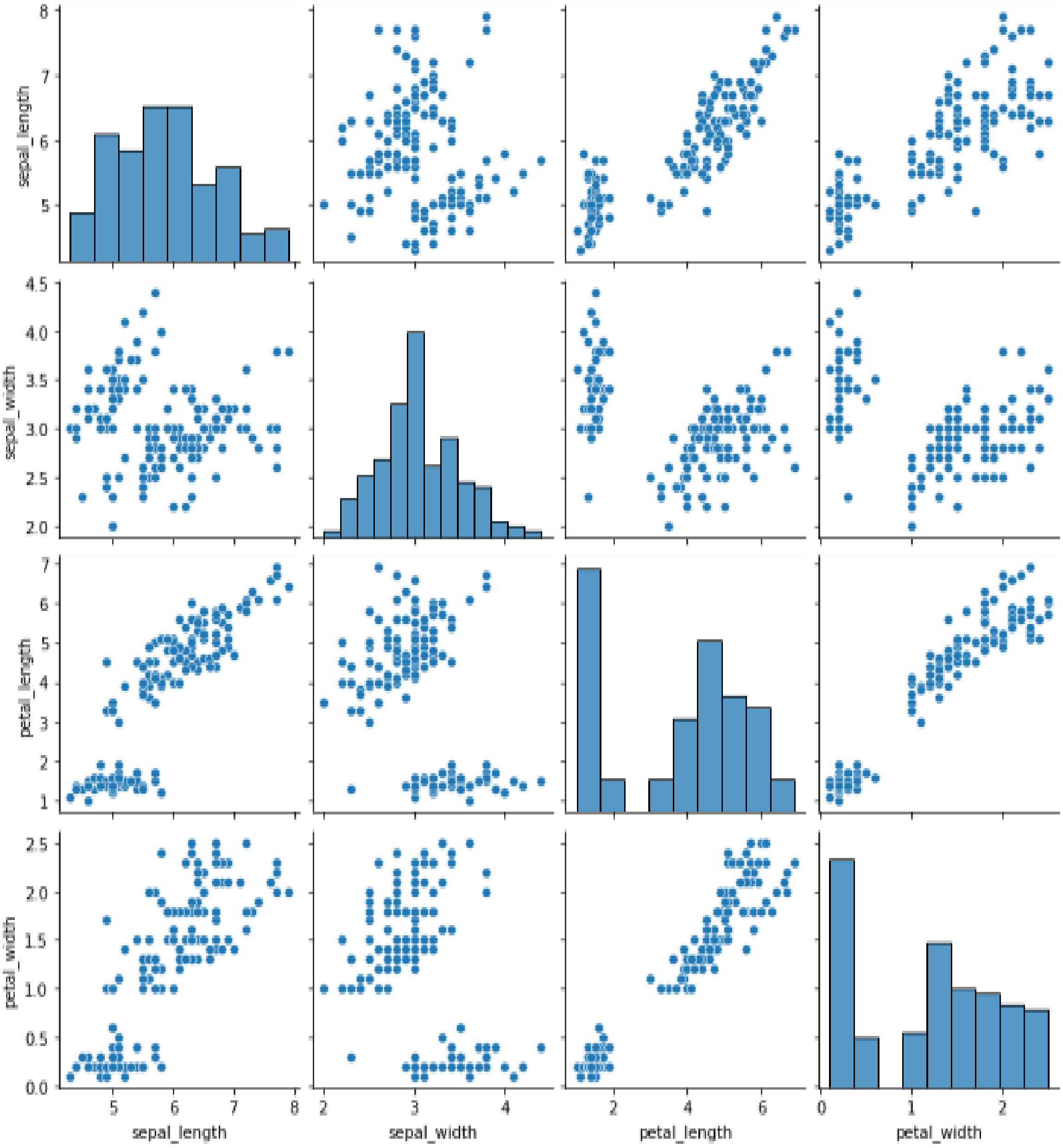
8. **Use pairplot() function to display pairwise relationships between attributes. Try different kind of plots {*'scatter', 'kde', 'hist', 'reg'*} and different kind of markers**

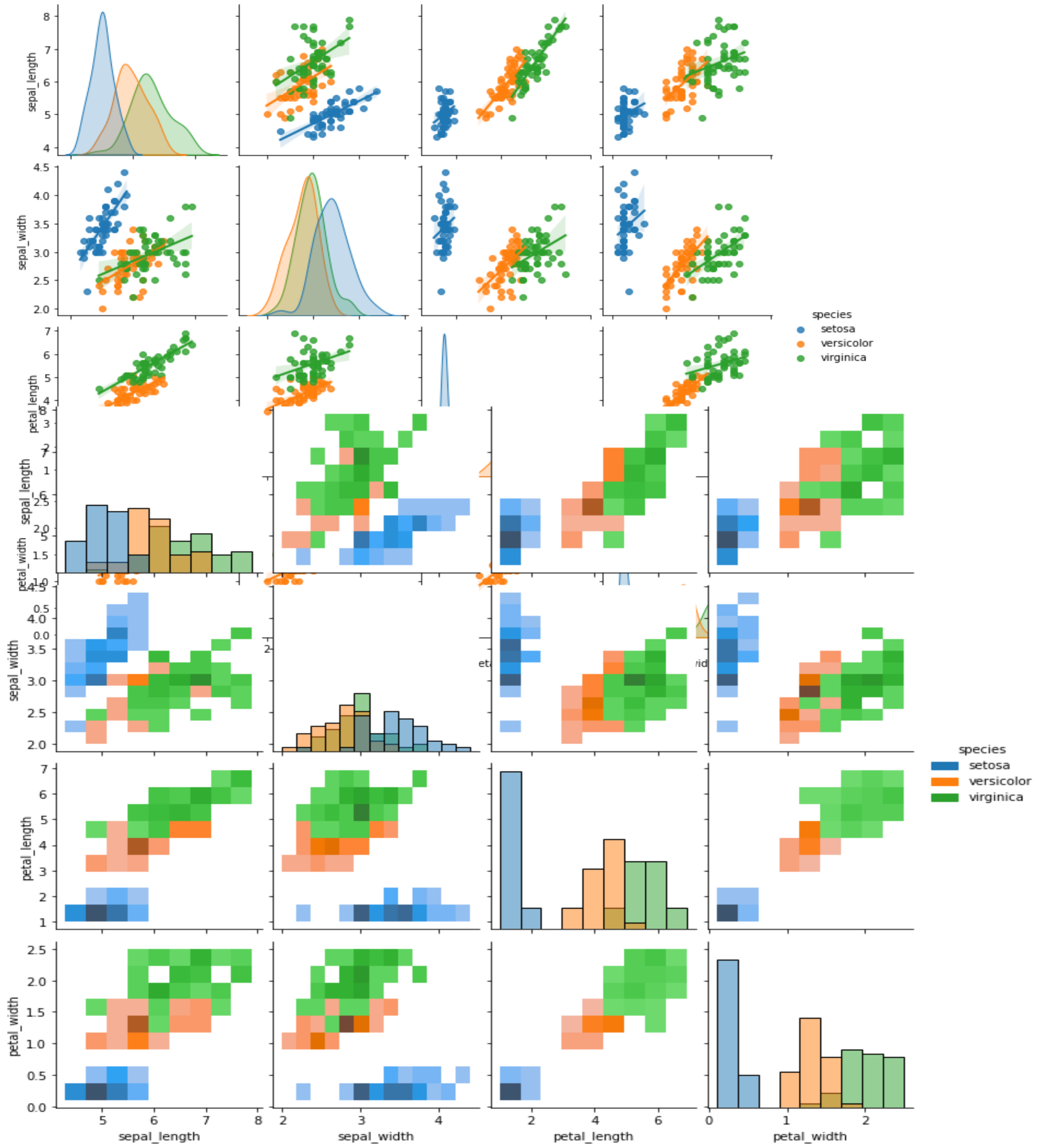
PROGRAM

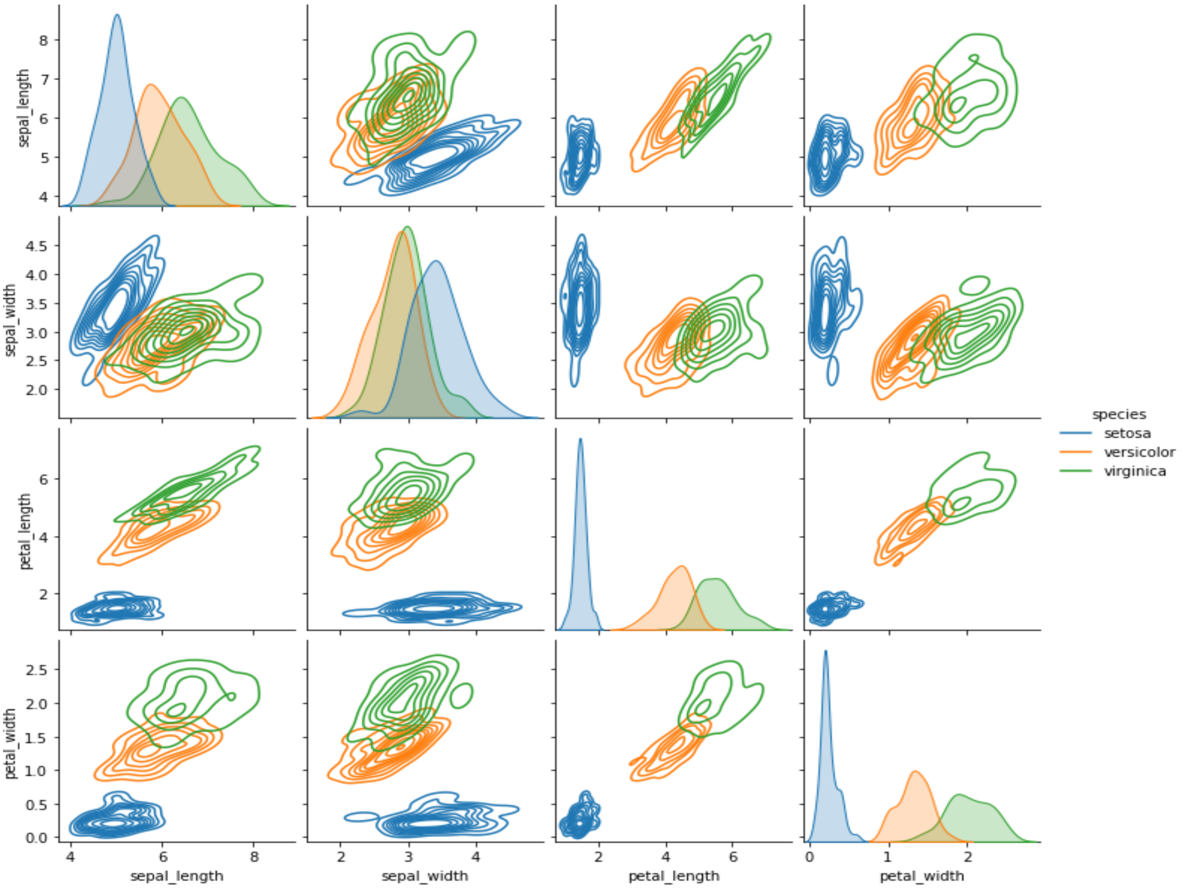
```
import numpy as np import pandas as pd import seaborn as sns

import matplotlib.pyplot as plt

%matplotlib inline

iris = sns.load_dataset('iris')

my_data_frame = pd.DataFrame(iris)

g = sns.pairplot(my_data_frame)
```

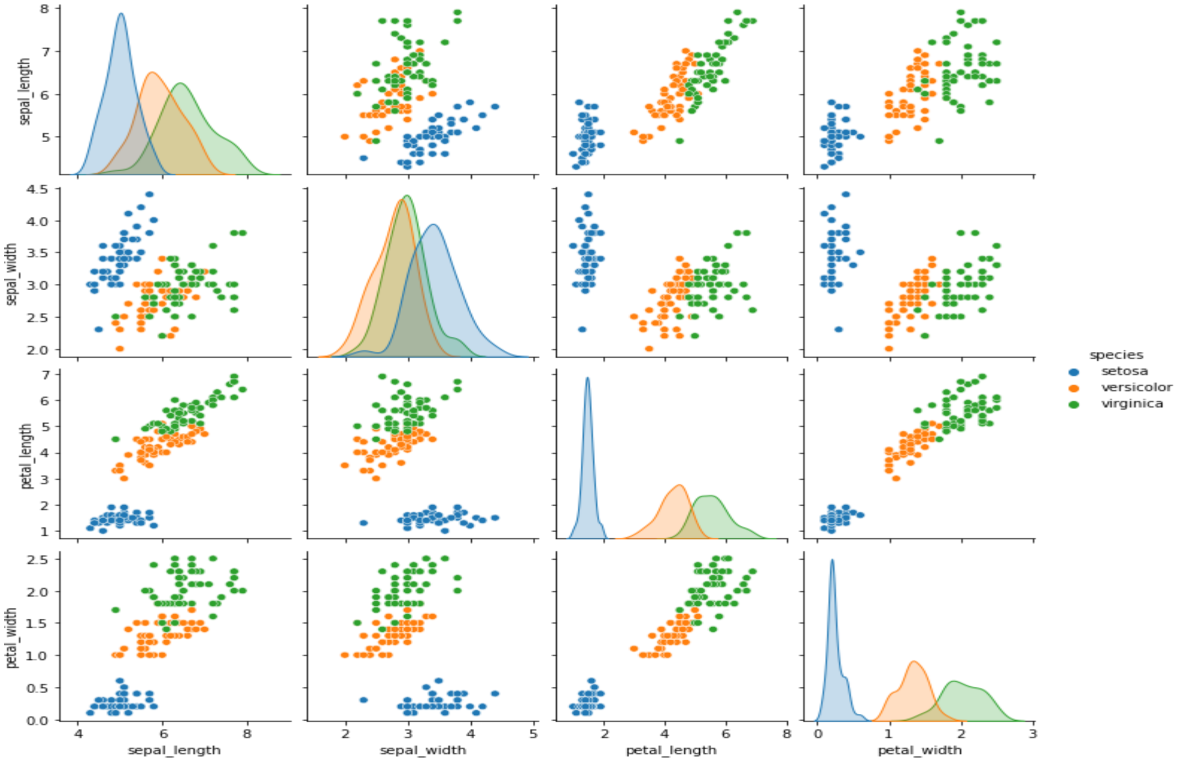
```
g = sns.pairplot(iris, kind="reg", hue="species")
g = sns.pairplot(iris, kind="hist", hue="species")
g = sns.pairplot(iris, kind="kde", hue="species")
g = sns.pairplot(iris, kind="scatter", hue="species")
```

OUTPUT




sepal_length
sepal_width
petal_length
petal_width
species
setosa
versicolor
virginica
8
7
6
5
4
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
sepal_length
sepal_width
petal_length
petal_width


sepal_length
8
7
6
5
4
sepal_width
4.5
4.0
3.5
3.0
2.5
2.0
petal_length
6
4
2
petal_width
2.5
2.0
1.5
1.0
0.5
0.0
4 6 8
sepal_length
2 3 4
sepal_width
2 4 6
petal_length
0 1 2
petal_width
species
setosa
versicolor
virginica

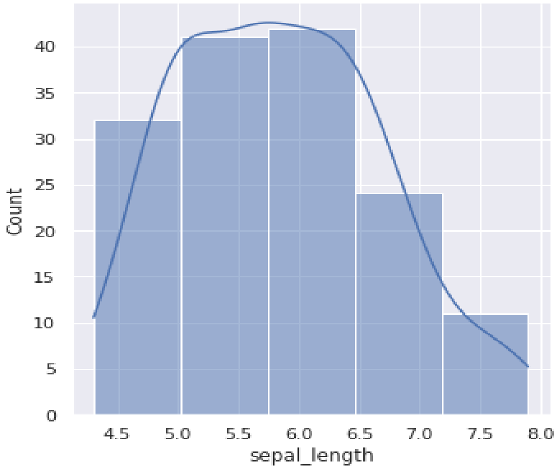

sepal_length
8
7
6
5
sepal_width
4.5
4.0
3.5
3.0
2.5
2.0
petal_length
7
6
5
4
3
2
1
petal_width
2.5
2.0
1.5
1.0
0.5
0.0
4 6 8
sepal_length
2 3 4 5
sepal_width
2 4 6 8
petal_length
0 1 2 3
petal_width
species
setosa
versicolor
virginica

9. Using the iris data set,get familiarize with functions:
1)displot()

PROGRAM

```
import seaborn as
sns import pandas

Import matplotlib.pyplot as plt
sns.get_dataset_names()

df = sns.load_dataset('iris')
df.head()

sns.displot(x = 'sepal_length',kde=True,bins = 5 , data =df)
```

OUTPUT

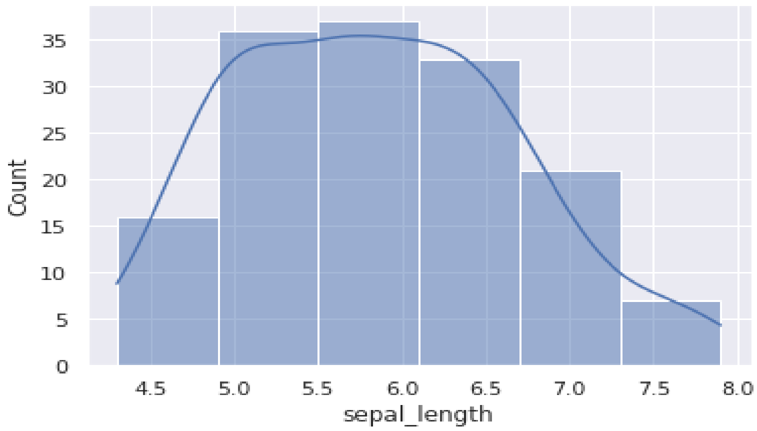


2) histplot()

PROGRAM

```
import seaborn as
sns import pandas
import matplotlib.pyplot as
plt sns.get_dataset_names()
df =
sns.load_dataset('iris')
df.head()
sns.histplot(x = 'sepal_length',kde=True,bins = 6 , data =df)
```

OUTPUT

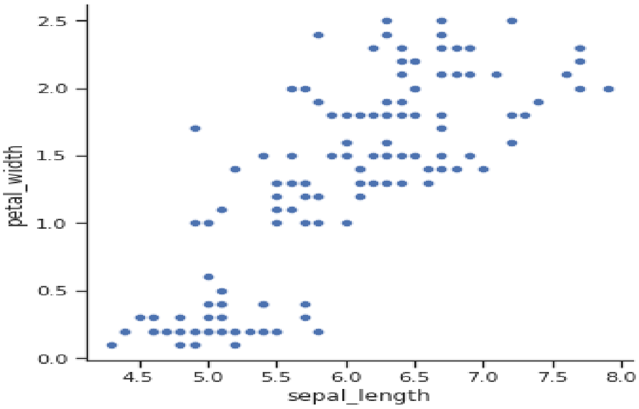


3)relplot()

PROGRAM

```
import seaborn as sns

sns.set(style ="ticks")

dataset = sns.load_dataset('iris') col=['sepal_length','sepal_width','petal_length','petal_width','type']

sns.relplot(x ="sepal_length", y ="petal_width", data = dataset)
```

OUTPUT



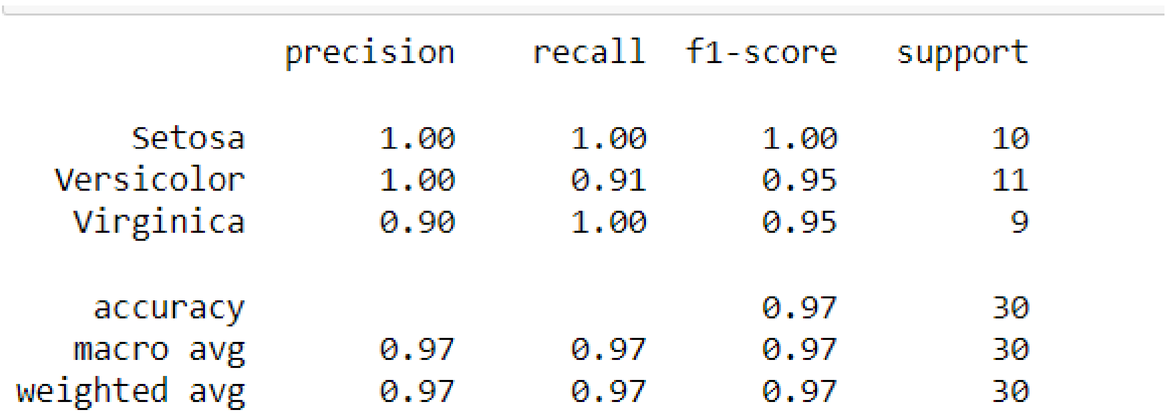
CYCLE 4

1. KNN Algorithm

Using the iris data set implement the KNN algorithm. Take different values for Test and training data set. Also use different values for k. Also find the accuracy level.

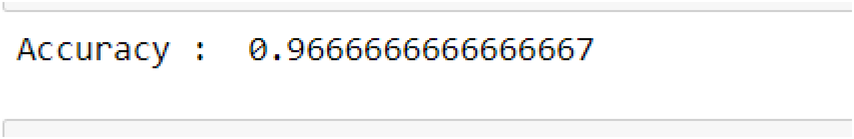
PROGRAM

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
dataset = pd.read_csv("iris.csv")
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 4].values
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
from sklearn.metrics import classification_report, confusion_matrix
print(classification_report(y_test, y_pred))
```



```
              precision    recall  f1-score   support

      Setosa       1.00      1.00      1.00        10
  Versicolor       1.00      0.91      0.95        11
   Virginica       0.90      1.00      0.95         9

    accuracy                           0.97        30
   macro avg       0.97      0.97      0.97        30
weighted avg       0.97      0.97      0.97        30
```

```
from sklearn.metrics import accuracy_score
print ("Accuracy : ", accuracy_score(y_test, y_pred))
df = pd.DataFrame({'Real Values':y_test, 'Predicted Values':y_pred})
```



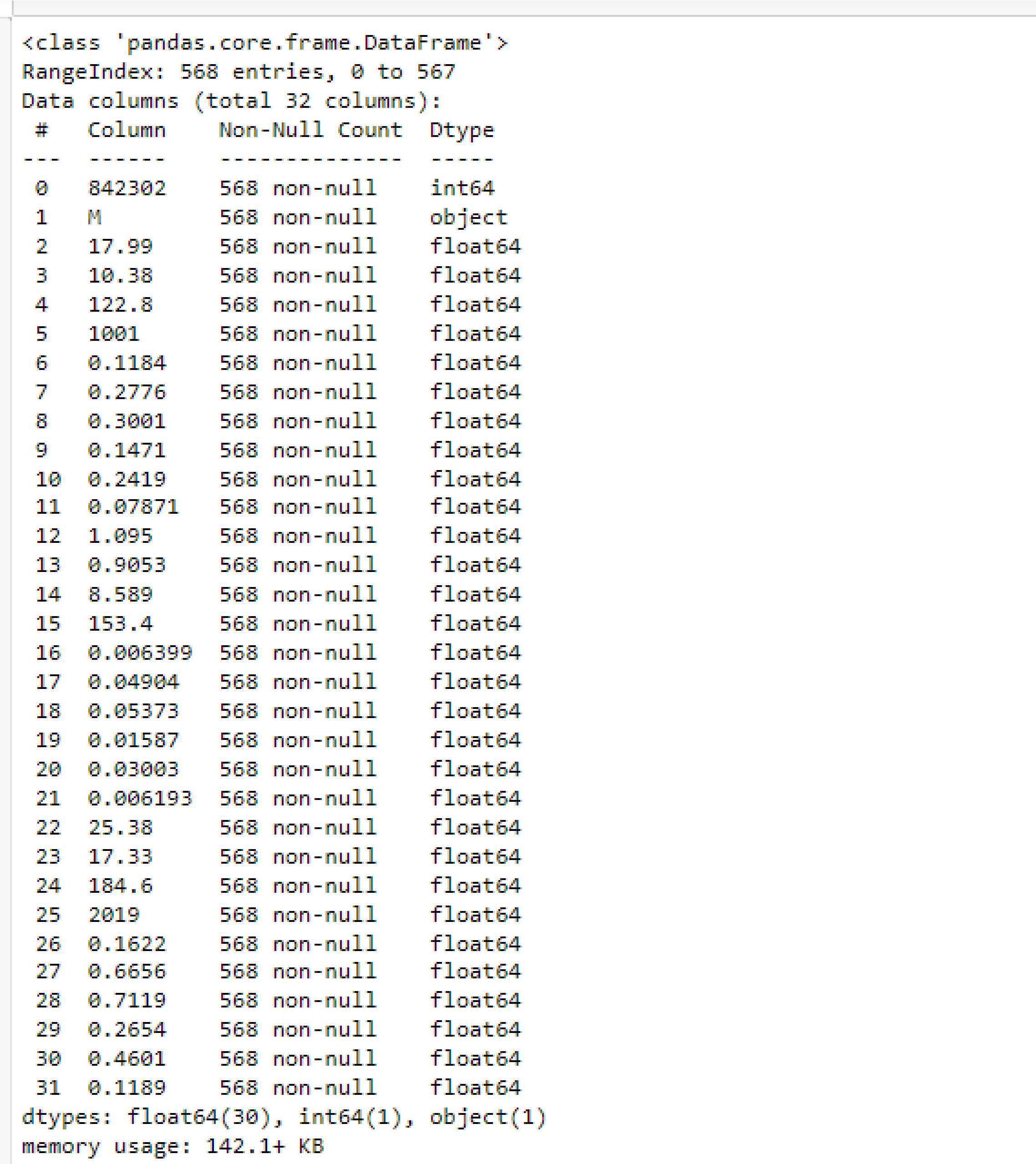
```
Accuracy :  0.9666666666666667
```

2. Download another data set suitable for the KNN and implement the KNN algorithm. Take different values for Test and training data set. Also use different values for k.

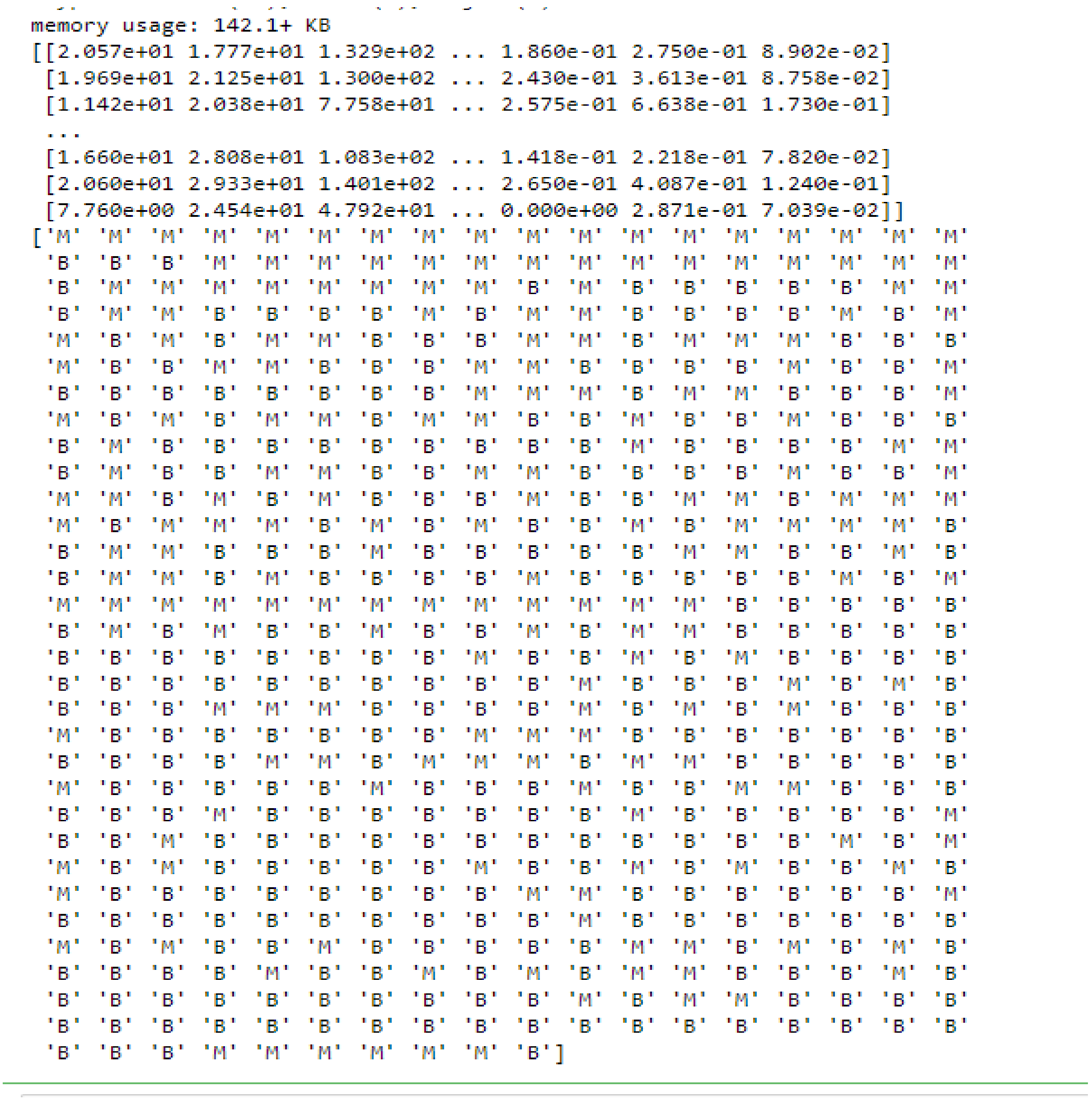
PROGRAM

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
dataset = pd.read_csv("cancer.csv")
dataset.head()
dataset.info()
X = dataset.iloc[:, 2:35].values
print(X)
y = dataset.iloc[:, 1].values
print(y)
```

OUTPUT:

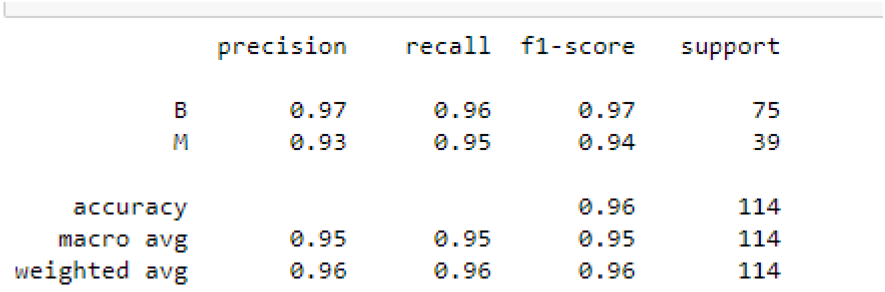


```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 568 entries, 0 to 567
Data columns (total 32 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   842302    568 non-null    int64  
 1   M         568 non-null    object 
 2   17.99     568 non-null    float64
 3   10.38     568 non-null    float64
 4   122.8     568 non-null    float64
 5   1001      568 non-null    float64
 6   0.1184    568 non-null    float64
 7   0.2776    568 non-null    float64
 8   0.3001    568 non-null    float64
 9   0.1471    568 non-null    float64
 10  0.2419    568 non-null    float64
 11  0.07871   568 non-null    float64
 12  1.095     568 non-null    float64
 13  0.9053    568 non-null    float64
 14  8.589     568 non-null    float64
 15  153.4     568 non-null    float64
 16  0.006399  568 non-null    float64
 17  0.04904   568 non-null    float64
 18  0.05373   568 non-null    float64
 19  0.01587   568 non-null    float64
 20  0.03003   568 non-null    float64
 21  0.006193  568 non-null    float64
 22  25.38     568 non-null    float64
 23  17.33     568 non-null    float64
 24  184.6     568 non-null    float64
 25  2019      568 non-null    float64
 26  0.1622    568 non-null    float64
 27  0.6656    568 non-null    float64
 28  0.7119    568 non-null    float64
 29  0.2654    568 non-null    float64
 30  0.4601    568 non-null    float64
 31  0.1189    568 non-null    float64
dtypes: float64(30), int64(1), object(1)
memory usage: 142.1+ KB
```



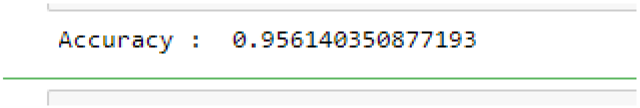
```
memory usage: 142.1+ KB
[[2.057e+01 1.777e+01 1.329e+02 ... 1.860e-01 2.750e-01 8.902e-02]
 [1.969e+01 2.125e+01 1.300e+02 ... 2.430e-01 3.613e-01 8.758e-02]
 [1.142e+01 2.038e+01 7.758e+01 ... 2.575e-01 6.638e-01 1.730e-01]
 ...
 [1.660e+01 2.808e+01 1.083e+02 ... 1.418e-01 2.218e-01 7.820e-02]
 [2.060e+01 2.933e+01 1.401e+02 ... 2.650e-01 4.087e-01 1.240e-01]
 [7.760e+00 2.454e+01 4.792e+01 ... 0.000e+00 2.871e-01 7.039e-02]]
['M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M'
 'B' 'B' 'B' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M'
 'B' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'M' 'M'
 'B' 'M' 'M' 'B' 'B' 'B' 'B' 'M' 'B' 'M' 'M' 'B' 'B' 'B' 'B' 'M' 'B' 'M'
 'M' 'B' 'M' 'B' 'M' 'M' 'B' 'B' 'B' 'M' 'M' 'B' 'M' 'M' 'M' 'B' 'B' 'B'
 'M' 'B' 'B' 'M' 'M' 'B' 'B' 'B' 'M' 'M' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'M'
 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'M' 'M' 'B' 'M' 'M' 'B' 'B' 'B' 'M'
 'M' 'B' 'M' 'B' 'M' 'M' 'B' 'M' 'M' 'B' 'B' 'M' 'B' 'B' 'M' 'B' 'B' 'B'
 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'M' 'M'
 'B' 'M' 'B' 'B' 'M' 'M' 'B' 'B' 'M' 'M' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'M'
 'M' 'M' 'B' 'M' 'B' 'M' 'B' 'B' 'B' 'M' 'B' 'B' 'M' 'M' 'B' 'M' 'M' 'M'
 'M' 'B' 'M' 'M' 'M' 'B' 'M' 'B' 'M' 'B' 'B' 'M' 'B' 'M' 'M' 'M' 'M' 'B'
 'B' 'M' 'M' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'M' 'M' 'B' 'B' 'M' 'B'
 'B' 'M' 'M' 'B' 'M' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'M'
 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'B' 'B' 'B' 'B' 'B'
 'B' 'M' 'B' 'M' 'B' 'B' 'M' 'B' 'B' 'M' 'B' 'M' 'M' 'B' 'B' 'B' 'B' 'B'
 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'M' 'B' 'M' 'B' 'B' 'B' 'B'
 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'M' 'B' 'M' 'B'
 'B' 'B' 'B' 'M' 'M' 'M' 'B' 'B' 'B' 'B' 'M' 'B' 'M' 'B' 'M' 'B' 'B' 'B'
 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'M' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'B'
 'B' 'B' 'B' 'B' 'M' 'M' 'B' 'M' 'M' 'M' 'B' 'M' 'M' 'B' 'B' 'B' 'B' 'B'
 'M' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'M' 'B' 'B' 'M' 'M' 'B' 'B' 'B'
 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'M'
 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'M'
 'M' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'M' 'B' 'M' 'B' 'B' 'M' 'B'
 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'M'
 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'B'
 'M' 'B' 'M' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'M' 'M' 'B' 'M' 'B' 'M' 'B'
 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'M' 'B' 'M' 'B' 'M' 'M' 'B' 'B' 'B' 'M' 'B'
 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'M' 'M' 'B' 'B' 'B' 'B'
 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B'
 'B' 'B' 'B' 'M' 'M' 'M' 'M' 'M' 'M' 'B']
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)

from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n_neighbors=5)

classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix

print(classification_report(y_test, y_pred))
```



```
              precision    recall  f1-score   support

           B       0.97      0.96      0.97        75
           M       0.93      0.95      0.94        39

    accuracy                           0.96       114
   macro avg       0.95      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114
```

```
from sklearn.metrics import accuracy_score

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

df = pd.DataFrame({'Real Values':y_test, 'Predicted Values':y_pred})
```



```
Accuracy :  0.956140350877193
```

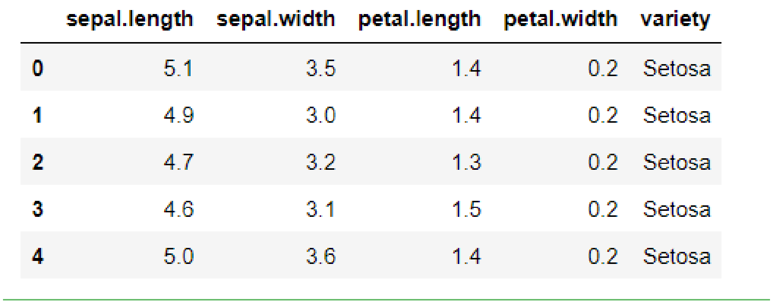
3. Naive Bayes Classification Algorithm

Using iris data set, implement naive bayes classification for different naive Bayes classification algorithms. ((i) gaussian (ii) bernoulli etc)

- **Find out the accuracy level w.r.t to each algorithm**
- **Display the no:of mislabeled classification from test data set**
- **List out the class labels of the mismatching records**

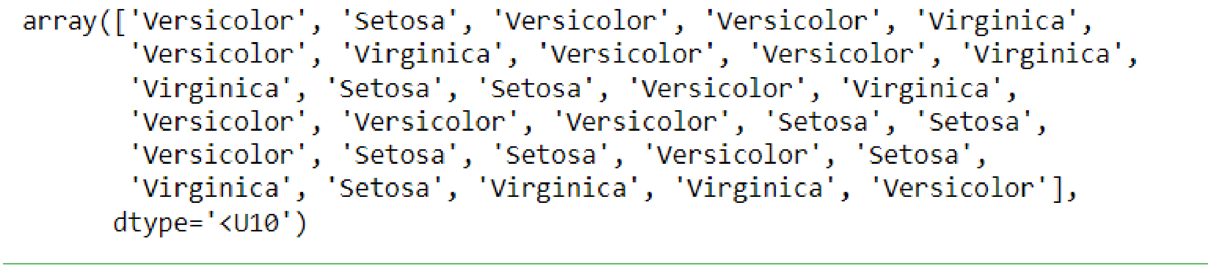
PROGRAM

```
import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

dataset = pd.read_csv('iris.csv')

X = dataset.iloc[:,:4].values

y = dataset['variety'].values

dataset.head(5)
```



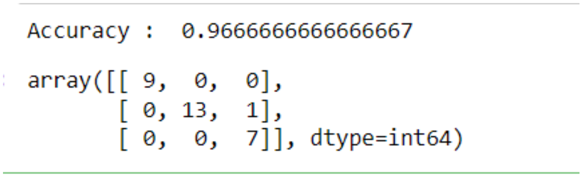
	sepal.length	sepal.width	petal.length	petal.width	variety
0	5.1	3.5	1.4	0.2	Setosa
1	4.9	3.0	1.4	0.2	Setosa
2	4.7	3.2	1.3	0.2	Setosa
3	4.6	3.1	1.5	0.2	Setosa
4	5.0	3.6	1.4	0.2	Setosa

```
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)

from sklearn.naive_bayes import GaussianNB

classifier = GaussianNB()

classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

y_pred
```



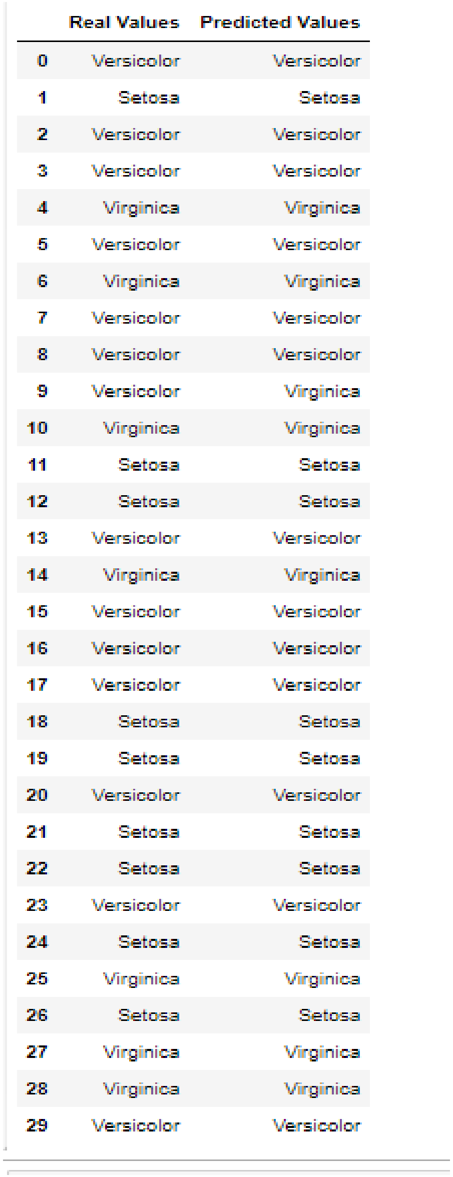
```
array(['Versicolor', 'Setosa', 'Versicolor', 'Versicolor', 'Virginica',
       'Versicolor', 'Virginica', 'Versicolor', 'Versicolor', 'Virginica',
       'Virginica', 'Setosa', 'Setosa', 'Versicolor', 'Virginica',
       'Versicolor', 'Versicolor', 'Versicolor', 'Setosa', 'Setosa',
       'Versicolor', 'Setosa', 'Setosa', 'Versicolor', 'Setosa',
       'Virginica', 'Setosa', 'Virginica', 'Virginica', 'Versicolor'],
      dtype='<U10')
```

```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)

from sklearn.metrics import accuracy_score

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

cm
```



```
Accuracy :  0.9666666666666667

array([[ 9,  0,  0],
       [ 0, 13,  1],
       [ 0,  0,  7]], dtype=int64)
```

```
df = pd.DataFrame({'Real Values':y_test, 'Predicted Values':y_pred})

df
```



	Real Values	Predicted Values
0	Versicolor	Versicolor
1	Setosa	Setosa
2	Versicolor	Versicolor
3	Versicolor	Versicolor
4	Virginica	Virginica
5	Versicolor	Versicolor
6	Virginica	Virginica
7	Versicolor	Versicolor
8	Versicolor	Versicolor
9	Versicolor	Virginica
10	Virginica	Virginica
11	Setosa	Setosa
12	Setosa	Setosa
13	Versicolor	Versicolor
14	Virginica	Virginica
15	Versicolor	Versicolor
16	Versicolor	Versicolor
17	Versicolor	Versicolor
18	Setosa	Setosa
19	Setosa	Setosa
20	Versicolor	Versicolor
21	Setosa	Setosa
22	Setosa	Setosa
23	Versicolor	Versicolor
24	Setosa	Setosa
25	Virginica	Virginica
26	Setosa	Setosa
27	Virginica	Virginica
28	Virginica	Virginica
29	Versicolor	Versicolor

4. Decision Tree Algorithm

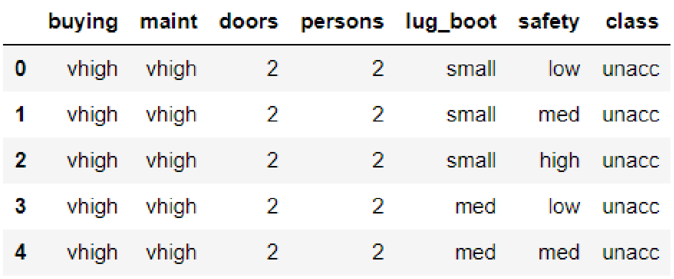
Use car details CSV file and implement decision tree algorithm

- **a. Find out the accuracy level.**
- **b. Display the no: of mislabelled classification from test data set**
- **c. List out the class labels of the mismatching records**

PROGRAM

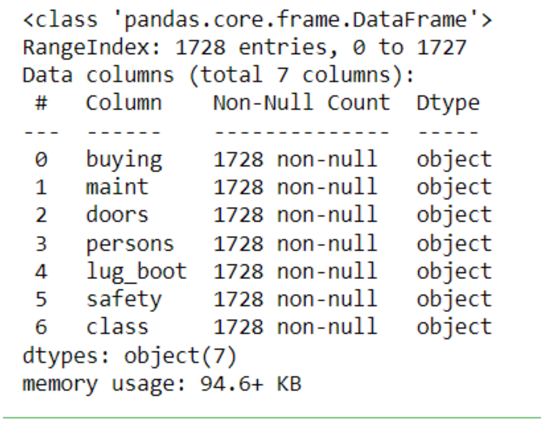
```
import os

import numpy as np

import pandas as pd

import numpy as np, pandas as pd

import matplotlib.pyplot as plt

from sklearn import tree, metrics, model_selection
```

```
data = pd.read_csv('car.csv',names=['buying','maint','doors','persons','lug_boot','safety','class'])

data.head()
```



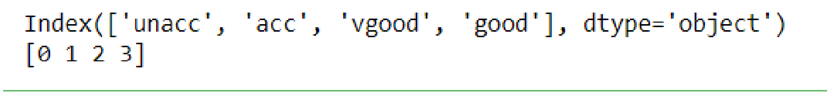
	buying	maint	doors	persons	lug_boot	safety	class
0	vhigh	vhigh	2	2	small	low	unacc
1	vhigh	vhigh	2	2	small	med	unacc
2	vhigh	vhigh	2	2	small	high	unacc
3	vhigh	vhigh	2	2	med	low	unacc
4	vhigh	vhigh	2	2	med	med	unacc

```
data.info()
```



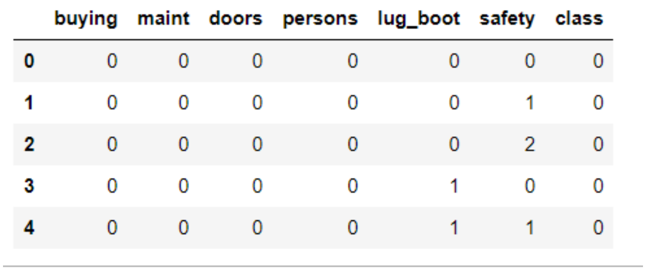
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1728 entries, 0 to 1727
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   buying    1728 non-null   object
 1   maint     1728 non-null   object
 2   doors     1728 non-null   object
 3   persons   1728 non-null   object
 4   lug_boot  1728 non-null   object
 5   safety    1728 non-null   object
 6   class     1728 non-null   object
dtypes: object(7)
memory usage: 94.6+ KB
```

```
data['class'],class_names = pd.factorize(data['class'])

print(class_names)

print(data['class'].unique())
```



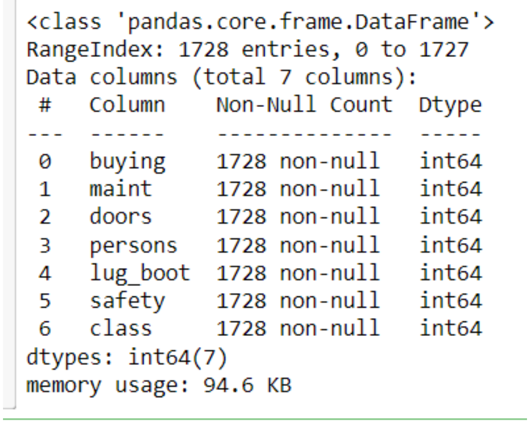
```
Index(['unacc', 'acc', 'vgood', 'good'], dtype='object')
[0 1 2 3]
```

```
data['buying'],_ = pd.factorize(data['buying'])

data['maint'],_ = pd.factorize(data['maint'])

data['doors'],_ = pd.factorize(data['doors'])

data['persons'],_ = pd.factorize(data['persons'])

data['lug_boot'],_ = pd.factorize(data['lug_boot'])

data['safety'],_ = pd.factorize(data['safety'])

data.head()
```



	buying	maint	doors	persons	lug_boot	safety	class
0	0	0	0	0	0	0	0
1	0	0	0	0	0	1	0
2	0	0	0	0	0	2	0
3	0	0	0	0	1	0	0
4	0	0	0	0	1	1	0

data.info()



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1728 entries, 0 to 1727
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   buying    1728 non-null   int64
 1   maint     1728 non-null   int64
 2   doors     1728 non-null   int64
 3   persons   1728 non-null   int64
 4   lug_boot  1728 non-null   int64
 5   safety    1728 non-null   int64
 6   class     1728 non-null   int64
dtypes: int64(7)
memory usage: 94.6 KB
```

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

y = data.iloc[:,-1]

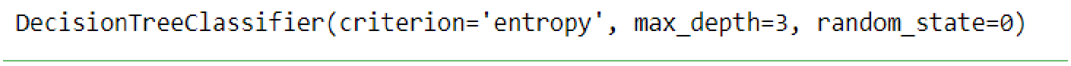
# split data randomly into 70% training and 30% test

X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y, test_size=0.3, random_state=0)

# train the decision tree

dtree = tree.DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=0)

dtree.fit(X_train, y_train)



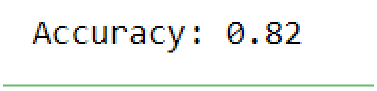
```
DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=0)
```

# use the model to make predictions with the test data

y_pred = dtree.predict(X_test)

# how did our model perform?

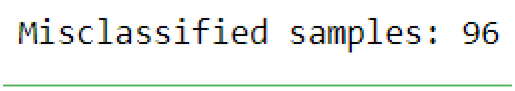
accuracy = metrics.accuracy_score(y_test, y_pred) print('Accuracy: {:.2f}'.format(accuracy))



```
Accuracy: 0.82
```

count_misclassified = (y_test != y_pred).sum()

print('Misclassified samples: {}'.format(count_misclassified))



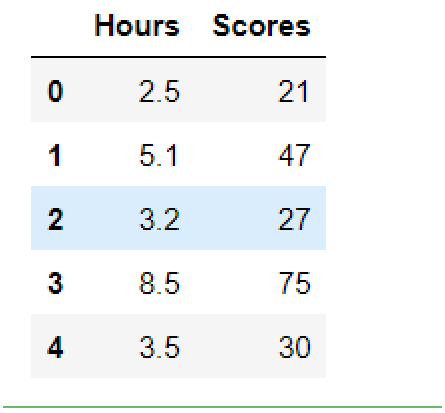
```
Misclassified samples: 96
```

5. Implement Simple and multiple linear regression for the data sets 'student_score.csv' and 'company_data .csv' respectively

Simple Linear Regression

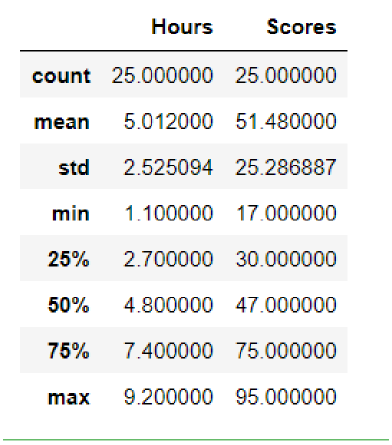
PROGRAM

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
#data set contains details of no.of hours spend by students for studt and their marks
student = pd.read_csv('student_scores.csv')
student.head()
```



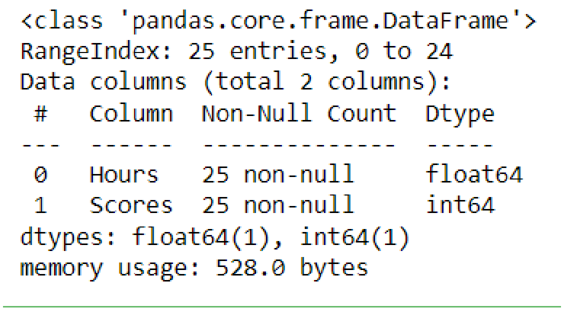
	Hours	Scores
0	2.5	21
1	5.1	47
2	3.2	27
3	8.5	75
4	3.5	30

```
student.describe()
```



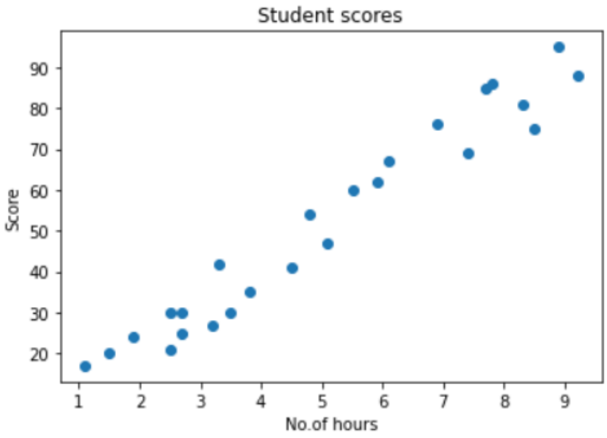
	Hours	Scores
count	25.000000	25.000000
mean	5.012000	51.480000
std	2.525094	25.286887
min	1.100000	17.000000
25%	2.700000	30.000000
50%	4.800000	47.000000
75%	7.400000	75.000000
max	9.200000	95.000000

student.info()



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25 entries, 0 to 24
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Hours   25 non-null     float64
 1   Scores  25 non-null     int64  
dtypes: float64(1), int64(1)
memory usage: 528.0 bytes
```

```
import matplotlib.pyplot as plt

Xax=student.iloc[:,0]

Yax=student.iloc[:,1]

plt.scatter(Xax,Yax)

plt.xlabel("No.of hours")

plt.ylabel("Score")

plt.title("Student scores")

plt.show()
```



#Perform the simple linear regression model

#Equation: Y=w0+w1.x

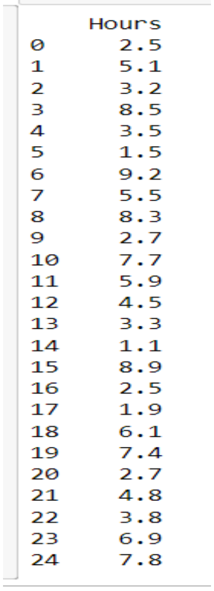
#Here Y(marks)=w0+w1.x

#Create x as hours and Y as marks

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

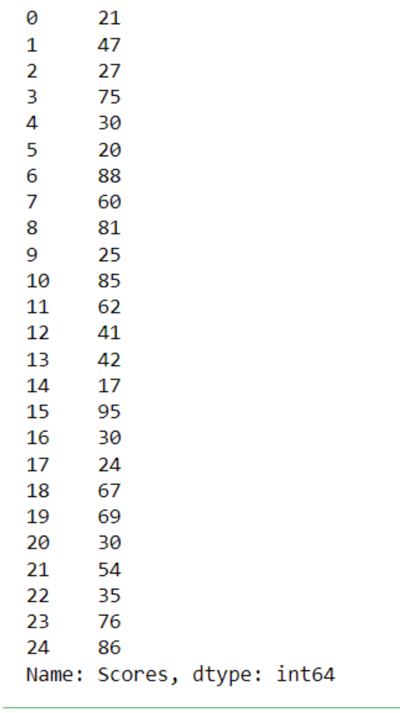
y = student.iloc[:, 1]

print(X)



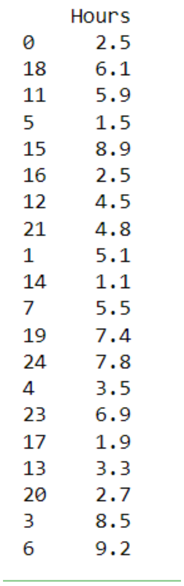
```
    Hours
0     2.5
1     5.1
2     3.2
3     8.5
4     3.5
5     1.5
6     9.2
7     5.5
8     8.3
9     2.7
10    7.7
11    5.9
12    4.5
13    3.3
14    1.1
15    8.9
16    2.5
17    1.9
18    6.1
19    7.4
20    2.7
21    4.8
22    3.8
23    6.9
24    7.8
```

print(y)



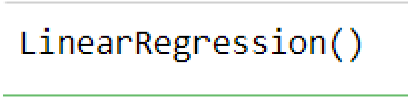
```
0     21
1     47
2     27
3     75
4     30
5     20
6     88
7     60
8     81
9     25
10    85
11    62
12    41
13    42
14    17
15    95
16    30
17    24
18    67
19    69
20    30
21    54
22    35
23    76
24    86
Name: Scores, dtype: int64
```

```
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)

print(X_train)
```



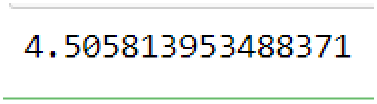
```
      Hours
0       2.5
18      6.1
11      5.9
5       1.5
15      8.9
16      2.5
12      4.5
21      4.8
1       5.1
14      1.1
7       5.5
19      7.4
24      7.8
4       3.5
23      6.9
17      1.9
13      3.3
20      2.7
3       8.5
6       9.2
```

```
from sklearn.linear_model import LinearRegression

regressor = LinearRegression()

regressor.fit(X_train, y_train)
```



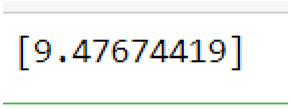
```
LinearRegression()
```

```
print(regressor.intercept_)
```



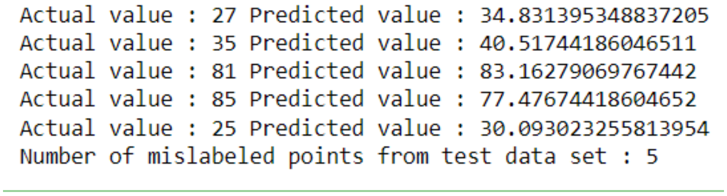
```
4.505813953488371
```

```
print(regressor.coef_)
```



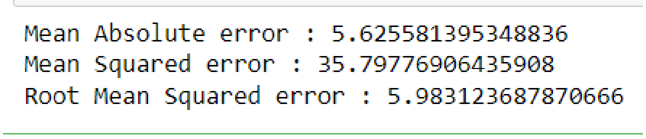
```
[9.47674419]
```

```
y_pred = regressor.predict(X_test)
for(i,j) in zip(y_test,y_pred):
   if i!=j:
       print("Actual value :",i,"Predicted value :",j)
print("Number of mislabeled points from test data set :", (y_test != y_pred).sum())
```



```
Actual value : 27 Predicted value : 34.831395348837205
Actual value : 35 Predicted value : 40.51744186046511
Actual value : 81 Predicted value : 83.16279069767442
Actual value : 85 Predicted value : 77.47674418604652
Actual value : 25 Predicted value : 30.093023255813954
Number of mislabeled points from test data set : 5
```

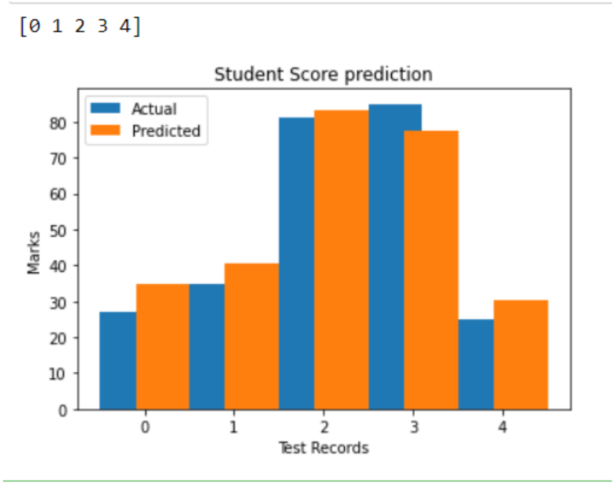
```
from sklearn import metrics
print("Mean Absolute error :", metrics.mean_absolute_error(y_test,y_pred))
print("Mean Squared error :", metrics.mean_squared_error(y_test,y_pred))
print("Root Mean Squared error :", np.sqrt(metrics.mean_squared_error(y_test,y_pred)))
```



```
Mean Absolute error : 5.625581395348836
Mean Squared error : 35.79776906435908
Root Mean Squared error : 5.983123687870666
```

```
import matplotlib.pyplot as plt
c=X_test['Hours'].count()
xax=np.arange(c)
print(xax)
```

```
X_axis = np.arange(len(xax))

plt.bar(X_axis-0.2, y_test, 0.6, label='Actual')

plt.bar(X_axis+0.2, y_pred, 0.6, label='Predicted')

plt.xlabel("Test Records")

plt.ylabel("Marks")

plt.title("Student Score prediction")

plt.legend()

plt.show()
```

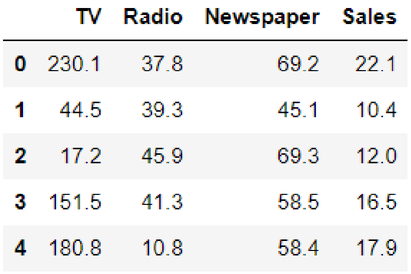


```
[0 1 2 3 4]
```

Multiple Linear Regression

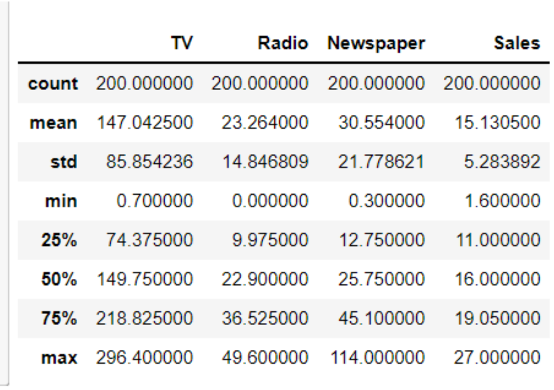
PROGRAM

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

advertising = pd.read_csv('Company_data.csv')

advertising.head()
```



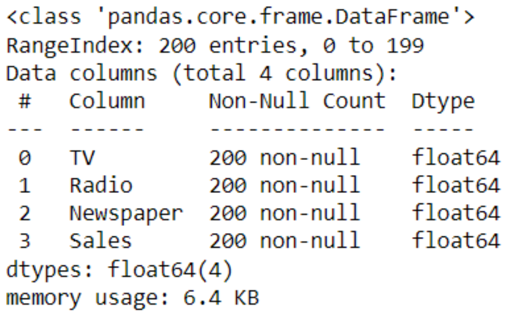
	TV	Radio	Newspaper	Sales
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	12.0
3	151.5	41.3	58.5	16.5
4	180.8	10.8	58.4	17.9

advertising.describe()



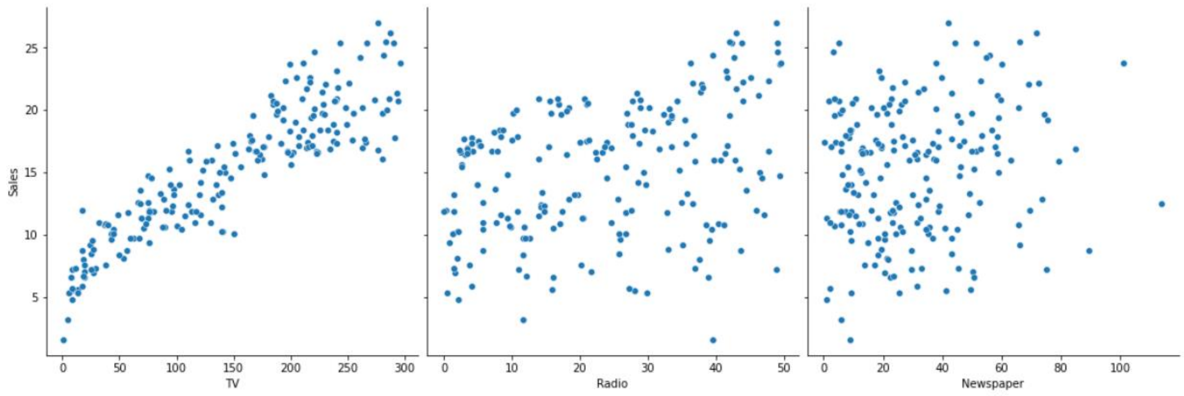
	TV	Radio	Newspaper	Sales
count	200.000000	200.000000	200.000000	200.000000
mean	147.042500	23.264000	30.554000	15.130500
std	85.854236	14.846809	21.778621	5.283892
min	0.700000	0.000000	0.300000	1.600000
25%	74.375000	9.975000	12.750000	11.000000
50%	149.750000	22.900000	25.750000	16.000000
75%	218.825000	36.525000	45.100000	19.050000
max	296.400000	49.600000	114.000000	27.000000

advertising.info()



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   TV         200 non-null    float64
 1   Radio      200 non-null    float64
 2   Newspaper  200 non-null    float64
 3   Sales      200 non-null    float64
dtypes: float64(4)
memory usage: 6.4 KB
```

```
import matplotlib.pyplot as plt

import seaborn as sns

sns.pairplot(advertising, x_vars=['TV', 'Radio', 'Newspaper'],

        y_vars='Sales', height=5, aspect=1, kind='scatter')

plt.show()
```



#perform the multiple linear regression model

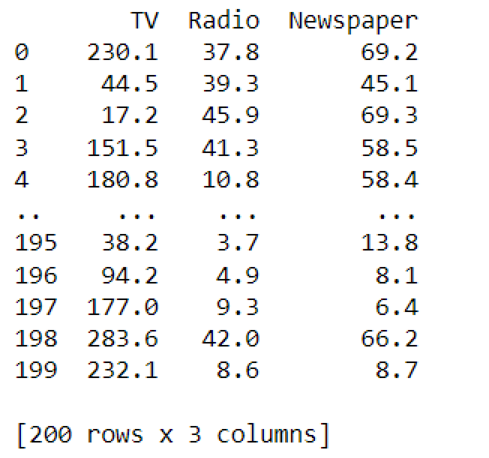
#Equation : Y=w0+w1.x1 + w2.x2 + w3.x3

#Here Y(sales)=w0+w1.x1(TV)+w2.x2(Radio)+w3.x3(Newspaper)

#create x and Y as sales

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

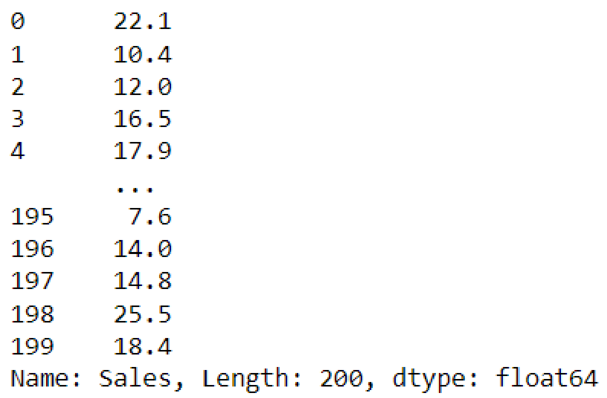
print(X)



```
        TV  Radio  Newspaper
0    230.1   37.8       69.2
1     44.5   39.3       45.1
2     17.2   45.9       69.3
3    151.5   41.3       58.5
4    180.8   10.8       58.4
..     ...    ...        ...
195   38.2    3.7       13.8
196   94.2    4.9        8.1
197  177.0    9.3        6.4
198  283.6   42.0       66.2
199  232.1    8.6        8.7

[200 rows x 3 columns]
```

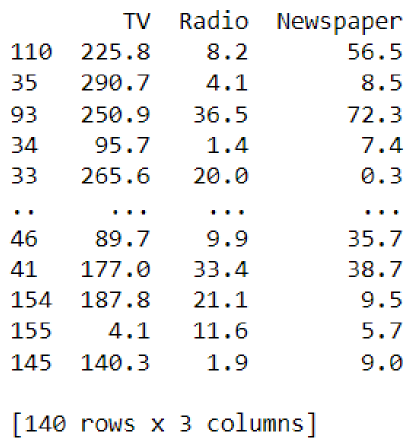
y = advertising.iloc[:, -1]

print(y)



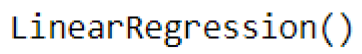
```
0      22.1
1      10.4
2      12.0
3      16.5
4      17.9
       ...
195     7.6
196    14.0
197    14.8
198    25.5
199    18.4
Name: Sales, Length: 200, dtype: float64
```

```
from sklearn.model_selection import train_test_split

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

print(X_train)
```



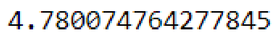
```
        TV  Radio  Newspaper
110  225.8    8.2       56.5
35   290.7    4.1        8.5
93   250.9   36.5       72.3
34    95.7    1.4        7.4
33   265.6   20.0        0.3
..     ...    ...        ...
46    89.7    9.9       35.7
41   177.0   33.4       38.7
154  187.8   21.1        9.5
155    4.1   11.6        5.7
145  140.3    1.9        9.0

[140 rows x 3 columns]
```

```
from sklearn.linear_model import LinearRegression

regressor = LinearRegression()

regressor.fit(X_train, y_train)
```



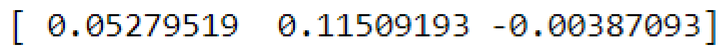
```
LinearRegression()
```

```
print(regressor.intercept_)
```



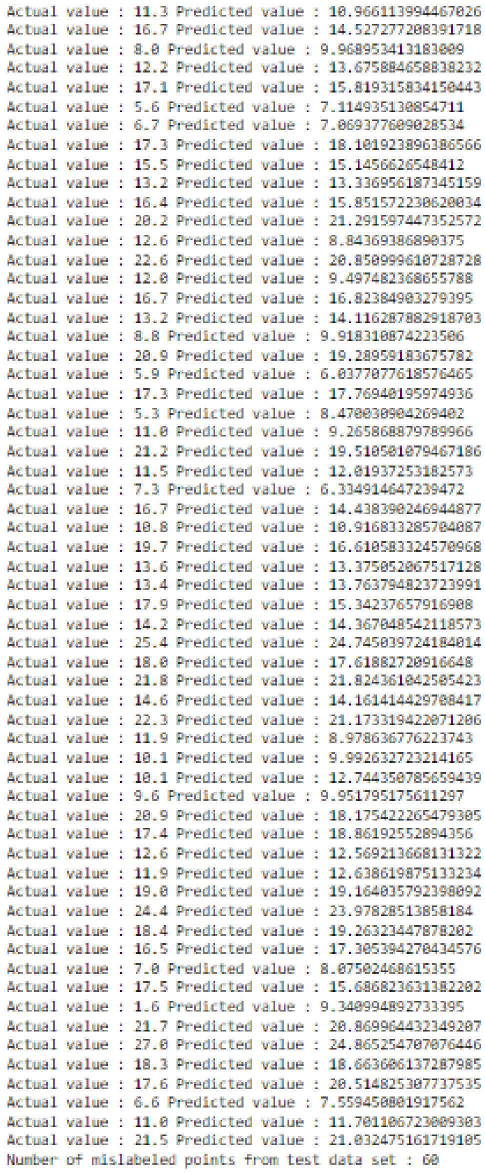
```
4.780074764277845
```

```
print(regressor.coef_)
```



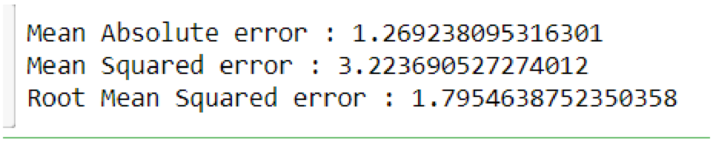
```
[ 0.05279519  0.11509193 -0.00387093]
```

```
y_pred = regressor.predict(X_test)
for(i,j) in zip(y_test,y_pred):
   if i!=j:
      print("Actual value :",i,"Predicted value :",j)
print("Number of mislabeled points from test data set :", (y_test != y_pred).sum())
```



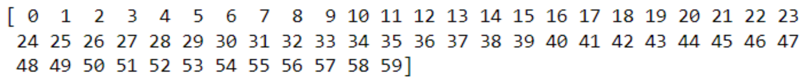
```
Actual value : 11.3 Predicted value : 10.966113994467026
Actual value : 16.7 Predicted value : 14.527277208391718
Actual value : 8.0 Predicted value : 9.968953413183009
Actual value : 12.2 Predicted value : 13.675884658838232
Actual value : 17.1 Predicted value : 15.819315834150443
Actual value : 5.6 Predicted value : 7.114935130854711
Actual value : 6.7 Predicted value : 7.069377609028534
Actual value : 17.3 Predicted value : 18.101923896386566
Actual value : 15.5 Predicted value : 15.1456626548412
Actual value : 13.2 Predicted value : 13.336956187345159
Actual value : 16.4 Predicted value : 15.851572230620034
Actual value : 20.2 Predicted value : 21.291597447352572
Actual value : 12.6 Predicted value : 8.84369386890375
Actual value : 22.6 Predicted value : 20.850999610728728
Actual value : 12.0 Predicted value : 9.497482368655788
Actual value : 16.7 Predicted value : 16.82384903279395
Actual value : 13.2 Predicted value : 14.116287882918703
Actual value : 8.8 Predicted value : 9.918310874223506
Actual value : 20.9 Predicted value : 19.28959183675782
Actual value : 5.9 Predicted value : 6.0377077618576465
Actual value : 17.3 Predicted value : 17.76940195974936
Actual value : 5.3 Predicted value : 8.470030904269402
Actual value : 11.0 Predicted value : 9.265868879789966
Actual value : 21.2 Predicted value : 19.510501079467186
Actual value : 11.5 Predicted value : 12.01937253182573
Actual value : 7.3 Predicted value : 6.334914647239472
Actual value : 16.7 Predicted value : 14.438390246944877
Actual value : 10.8 Predicted value : 10.916833285704087
Actual value : 19.7 Predicted value : 16.610583324570968
Actual value : 13.6 Predicted value : 13.375052067517128
Actual value : 13.4 Predicted value : 13.763794823723991
Actual value : 17.9 Predicted value : 15.34237657916908
Actual value : 14.2 Predicted value : 14.367048542118573
Actual value : 25.4 Predicted value : 24.745039724184014
Actual value : 18.0 Predicted value : 17.61882720916648
Actual value : 21.8 Predicted value : 21.824361042505423
Actual value : 14.6 Predicted value : 14.161414429708417
Actual value : 22.3 Predicted value : 21.173319422071206
Actual value : 11.9 Predicted value : 8.978636776223743
Actual value : 10.1 Predicted value : 9.992632723214165
Actual value : 10.1 Predicted value : 12.744350785659439
Actual value : 9.6 Predicted value : 9.951795175611297
Actual value : 20.9 Predicted value : 18.175422265479305
Actual value : 17.4 Predicted value : 18.86192552894356
Actual value : 12.6 Predicted value : 12.569213668131322
Actual value : 11.9 Predicted value : 12.638619875133234
Actual value : 19.0 Predicted value : 19.164035792398092
Actual value : 24.4 Predicted value : 23.97828513858184
Actual value : 18.4 Predicted value : 19.26323447878202
Actual value : 16.5 Predicted value : 17.305394270434576
Actual value : 7.0 Predicted value : 8.07502468615355
Actual value : 17.5 Predicted value : 15.686823631382202
Actual value : 1.6 Predicted value : 9.340994892733395
Actual value : 21.7 Predicted value : 20.869964432349207
Actual value : 27.0 Predicted value : 24.865254707076446
Actual value : 18.3 Predicted value : 18.663606137287985
Actual value : 17.6 Predicted value : 20.514825307737535
Actual value : 6.6 Predicted value : 7.559450801917562
Actual value : 11.0 Predicted value : 11.701106723009303
Actual value : 21.5 Predicted value : 21.032475161719105
Number of mislabeled points from test data set : 60
```

```
from sklearn import metrics

print("Mean Absolute error :", metrics.mean_absolute_error(y_test,y_pred))

print("Mean Squared error :", metrics.mean_squared_error(y_test,y_pred))

print("Root Mean Squared error :", np.sqrt(metrics.mean_squared_error(y_test,y_pred)))
```

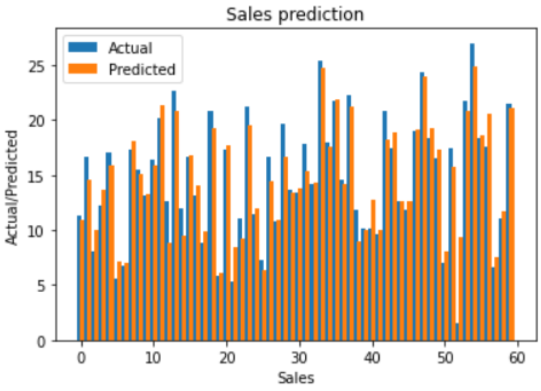


```
Mean Absolute error : 1.269238095316301
Mean Squared error : 3.223690527274012
Root Mean Squared error : 1.7954638752350358
```

```
import matplotlib.pyplot as plt

c=X_test['TV'].count()

xax=np.arange(c)

print(xax)

X_axis = np.arange(len(xax))

plt.bar(X_axis-0.2, y_test, 0.6, label='Actual')

plt.bar(X_axis+0.2, y_pred, 0.6, label='Predicted')

plt.xlabel("Sales")

plt.ylabel("Actual/Predicted")

plt.title("Sales prediction")

plt.legend()

plt.show()
```



```
[ 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]
```

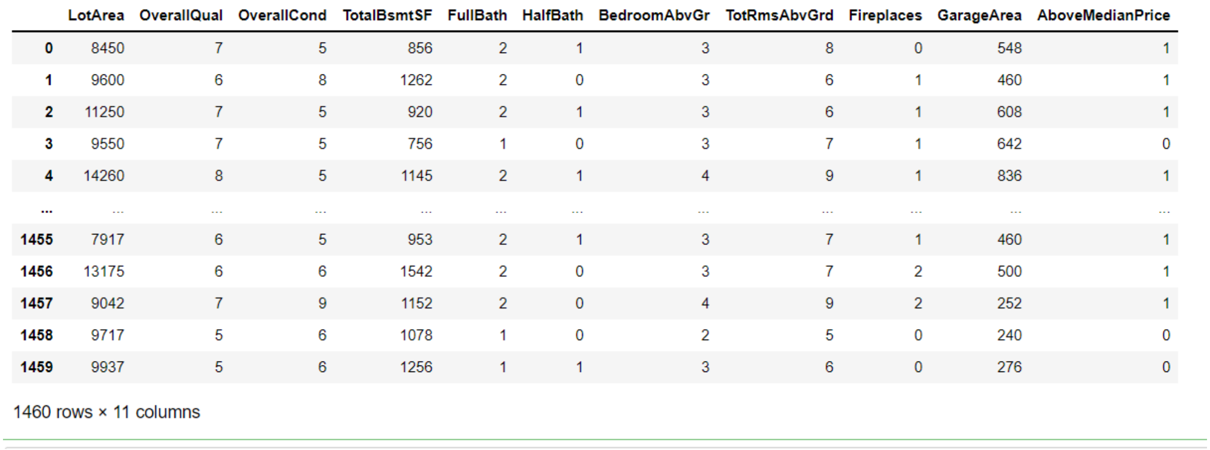


6. Neural Networks

Create a neural network for the given 'houseprice.csv' to predict the whether price of the house is above or below median value or not

PROGRAM:

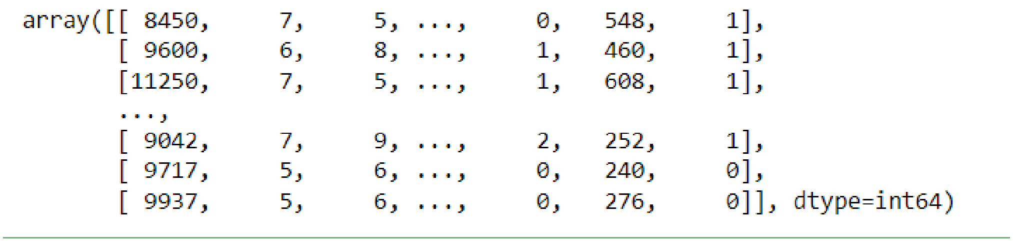
```
import pandas as pd

df = pd.read_csv('housepricedata.csv')

df
```



	LotArea	OverallQual	OverallCond	TotalBsmtSF	FullBath	HalfBath	BedroomAbvGr	TotRmsAbvGrd	Fireplaces	GarageArea	AboveMedianPrice
0	8450	7	5	856	2	1	3	8	0	548	1
1	9600	6	8	1262	2	0	3	6	1	460	1
2	11250	7	5	920	2	1	3	6	1	608	1
3	9550	7	5	756	1	0	3	7	1	642	0
4	14260	8	5	1145	2	1	4	9	1	836	1
...	...	...	...	...	...	...	...	...	...	...	...
1455	7917	6	5	953	2	1	3	7	1	460	1
1456	13175	6	6	1542	2	0	3	7	2	500	1
1457	9042	7	9	1152	2	0	4	9	2	252	1
1458	9717	5	6	1078	1	0	2	5	0	240	0
1459	9937	5	6	1256	1	1	3	6	0	276	0

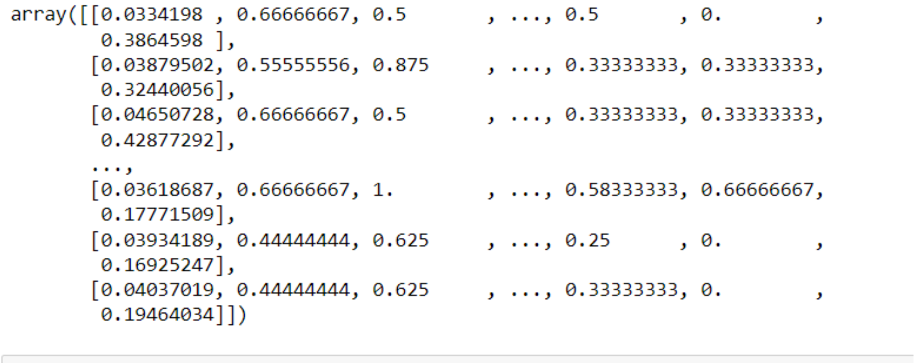
1460 rows × 11 columns

```
dataset = df.values

dataset
```



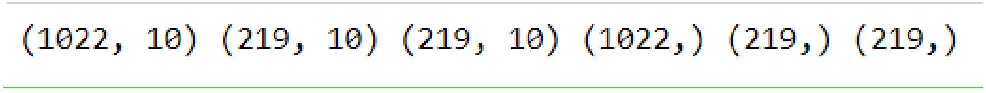
```
array([[ 8450,     7,     5, ...,     0,   548,     1],
       [ 9600,     6,     8, ...,     1,   460,     1],
       [11250,     7,     5, ...,     1,   608,     1],
       ...,
       [ 9042,     7,     9, ...,     2,   252,     1],
       [ 9717,     5,     6, ...,     0,   240,     0],
       [ 9937,     5,     6, ...,     0,   276,     0]], dtype=int64)
```

```
X = dataset[:,0:10]

Y = dataset[:,10]

from sklearn import preprocessing

min_max_scaler = preprocessing.MinMaxScaler()

X_scale = min_max_scaler.fit_transform(X)

X_scale
```



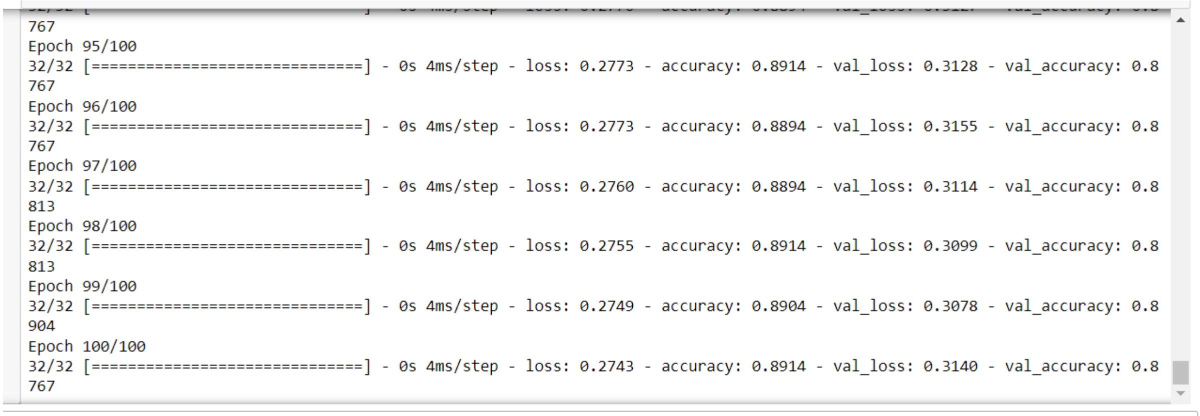
```
array([[0.0334198 , 0.66666667, 0.5       , ..., 0.5       , 0.        ,
        0.3864598 ],
       [0.03879502, 0.55555556, 0.875     , ..., 0.33333333, 0.33333333,
        0.32440056],
       [0.04650728, 0.66666667, 0.5       , ..., 0.33333333, 0.33333333,
        0.42877292],
       ...,
       [0.03618687, 0.66666667, 1.        , ..., 0.58333333, 0.66666667,
        0.17771509],
       [0.03934189, 0.44444444, 0.625     , ..., 0.25      , 0.        ,
        0.16925247],
       [0.04037019, 0.44444444, 0.625     , ..., 0.33333333, 0.        ,
        0.19464034]])
```

```
from sklearn.model_selection import train_test_split
X_train, X_val_and_test, Y_train, Y_val_and_test = train_test_split(X_scale, Y, test_size=0.3)
X_val, X_test, Y_val, Y_test = train_test_split(X_val_and_test, Y_val_and_test, test_size=0.5)
print(X_train.shape, X_val.shape, X_test.shape, Y_train.shape, Y_val.shape, Y_test.shape)
```



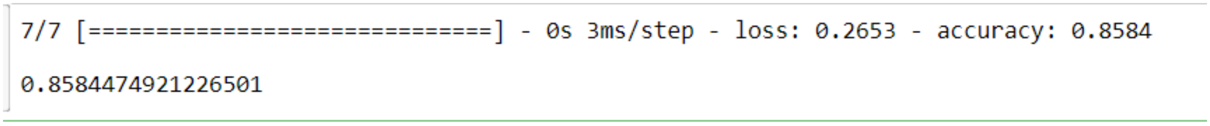
```
(1022, 10) (219, 10) (219, 10) (1022,) (219,) (219,)
```

```
from keras.models import Sequential
from keras.layers import Dense
model = Sequential([
    Dense(32, activation='relu', input_shape=(10,)),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid'),
])
model.compile(optimizer='sgd',
              loss='binary_crossentropy',
              metrics=['accuracy'])
hist = model.fit(X_train, Y_train,
          batch_size=32, epochs=100,
          validation_data=(X_val, Y_val))
```



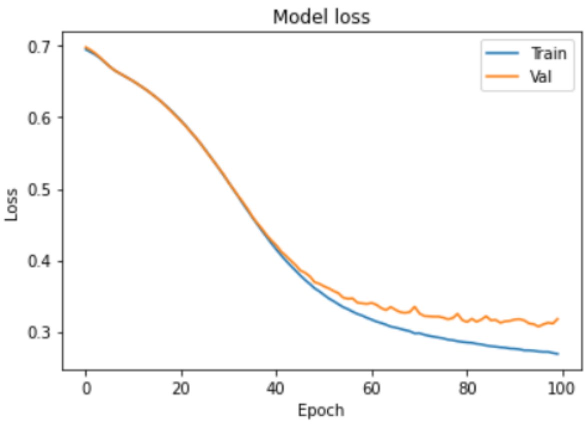
```
767
Epoch 95/100
32/32 [==============================] - 0s 4ms/step - loss: 0.2773 - accuracy: 0.8914 - val_loss: 0.3128 - val_accuracy: 0.8
767
Epoch 96/100
32/32 [==============================] - 0s 4ms/step - loss: 0.2773 - accuracy: 0.8894 - val_loss: 0.3155 - val_accuracy: 0.8
767
Epoch 97/100
32/32 [==============================] - 0s 4ms/step - loss: 0.2760 - accuracy: 0.8894 - val_loss: 0.3114 - val_accuracy: 0.8
813
Epoch 98/100
32/32 [==============================] - 0s 4ms/step - loss: 0.2755 - accuracy: 0.8914 - val_loss: 0.3099 - val_accuracy: 0.8
813
Epoch 99/100
32/32 [==============================] - 0s 4ms/step - loss: 0.2749 - accuracy: 0.8904 - val_loss: 0.3078 - val_accuracy: 0.8
904
Epoch 100/100
32/32 [==============================] - 0s 4ms/step - loss: 0.2743 - accuracy: 0.8914 - val_loss: 0.3140 - val_accuracy: 0.8
767
```

```
model.evaluate(X_test, Y_test)[1]
```

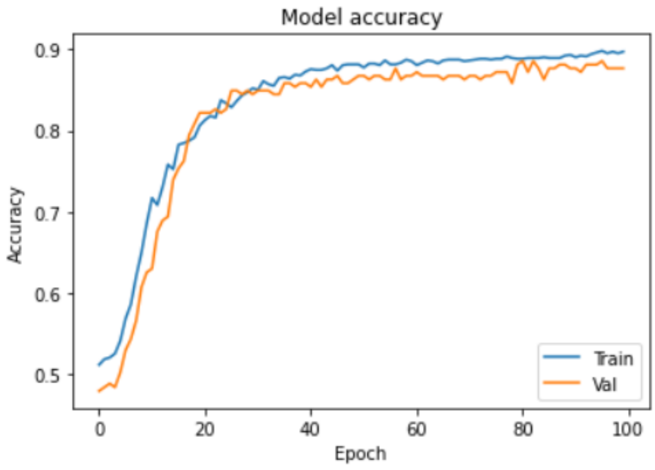


```
7/7 [==============================] - 0s 3ms/step - loss: 0.2653 - accuracy: 0.8584

0.8584474921226501
```

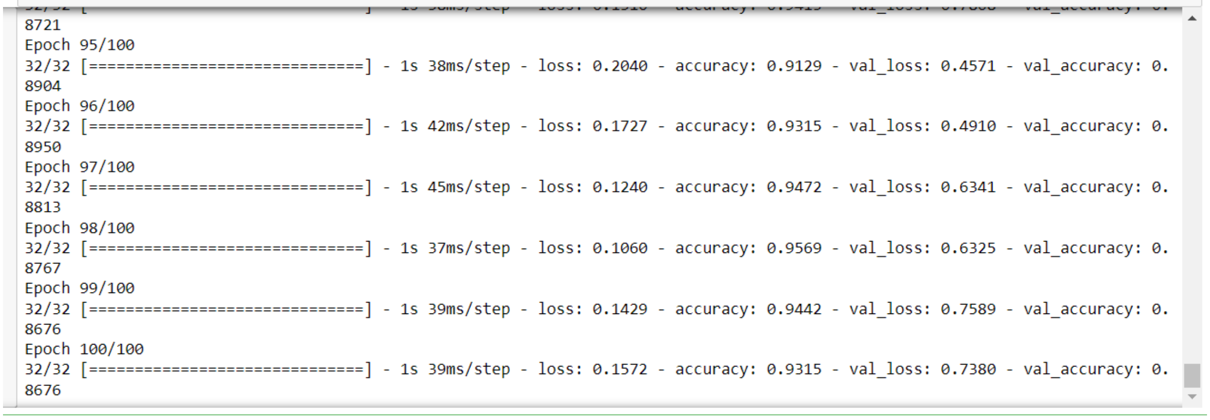
```
import matplotlib.pyplot as plt
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper right')
plt.show()
```



```
plt.plot(hist.history['accuracy'])

plt.plot(hist.history['val_accuracy'])

plt.title('Model accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(['Train', 'Val'], loc='lower right')

plt.show()
```

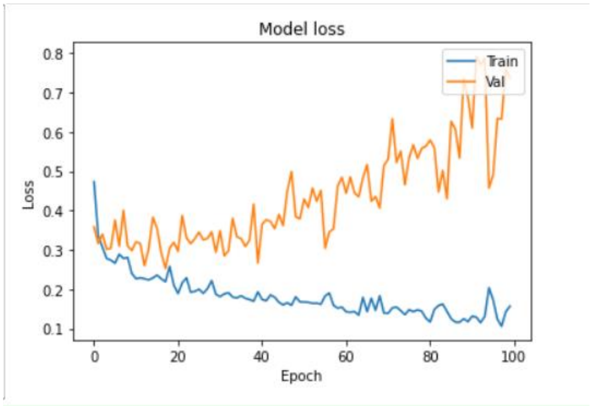


```
model_2 = Sequential([

    Dense(1000, activation='relu', input_shape=(10,)),

    Dense(1000, activation='relu'),

    Dense(1000, activation='relu'),

    Dense(1000, activation='relu'),

    Dense(1, activation='sigmoid'),

])

model_2.compile(optimizer='adam',

              loss='binary_crossentropy',

              metrics=['accuracy'])

hist_2 = model_2.fit(X_train, Y_train,

          batch_size=32, epochs=100,

          validation_data=(X_val, Y_val))
```

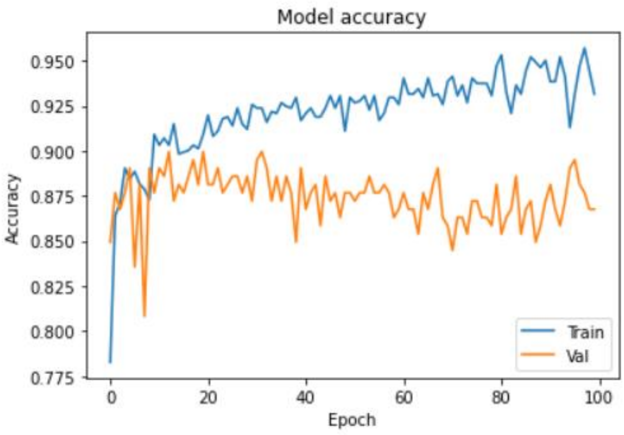


```
8721
Epoch 95/100
32/32 [==============================] - 1s 38ms/step - loss: 0.2040 - accuracy: 0.9129 - val_loss: 0.4571 - val_accuracy: 0.
8904
Epoch 96/100
32/32 [==============================] - 1s 42ms/step - loss: 0.1727 - accuracy: 0.9315 - val_loss: 0.4910 - val_accuracy: 0.
8950
Epoch 97/100
32/32 [==============================] - 1s 45ms/step - loss: 0.1240 - accuracy: 0.9472 - val_loss: 0.6341 - val_accuracy: 0.
8813
Epoch 98/100
32/32 [==============================] - 1s 37ms/step - loss: 0.1060 - accuracy: 0.9569 - val_loss: 0.6325 - val_accuracy: 0.
8767
Epoch 99/100
32/32 [==============================] - 1s 39ms/step - loss: 0.1429 - accuracy: 0.9442 - val_loss: 0.7589 - val_accuracy: 0.
8676
Epoch 100/100
32/32 [==============================] - 1s 39ms/step - loss: 0.1572 - accuracy: 0.9315 - val_loss: 0.7380 - val_accuracy: 0.
8676
```

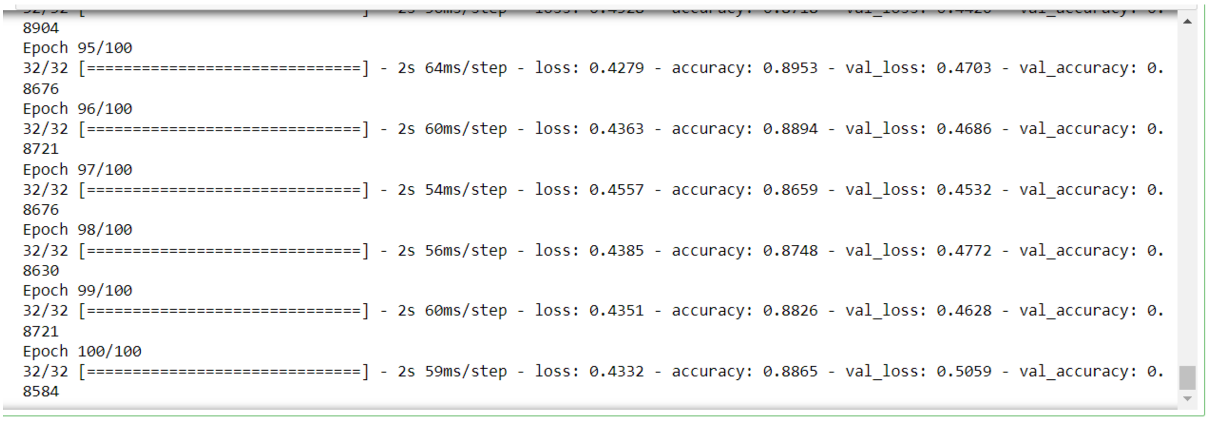
```
plt.plot(hist_2.history['loss'])
plt.plot(hist_2.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper right')
plt.show()
```



```
plt.plot(hist_2.history['accuracy'])
plt.plot(hist_2.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='lower right')
plt.show()
```

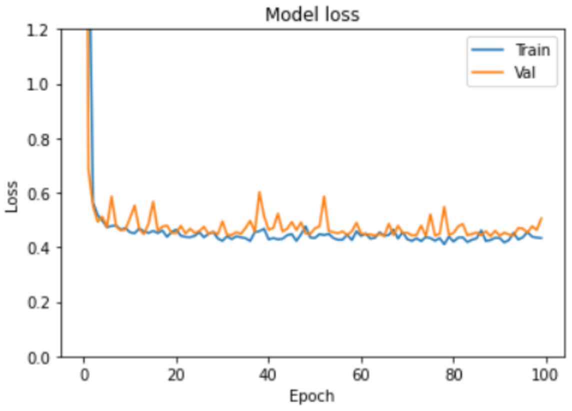


```
from keras.layers import Dropout
from keras import regularizers
model_3 = Sequential([
    Dense(1000, activation='relu', kernel_regularizer=regularizers.l2(0.01), input_shape=(10,)),
    Dropout(0.3),
    Dense(1000, activation='relu', kernel_regularizer=regularizers.l2(0.01)),
    Dropout(0.3),
    Dense(1000, activation='relu', kernel_regularizer=regularizers.l2(0.01)),
    Dropout(0.3),
    Dense(1000, activation='relu', kernel_regularizer=regularizers.l2(0.01)),
    Dropout(0.3),
    Dense(1, activation='sigmoid', kernel_regularizer=regularizers.l2(0.01)),
])
model_3.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
hist_3 = model_3.fit(X_train, Y_train,
          batch_size=32, epochs=100,
          validation_data=(X_val, Y_val))
```

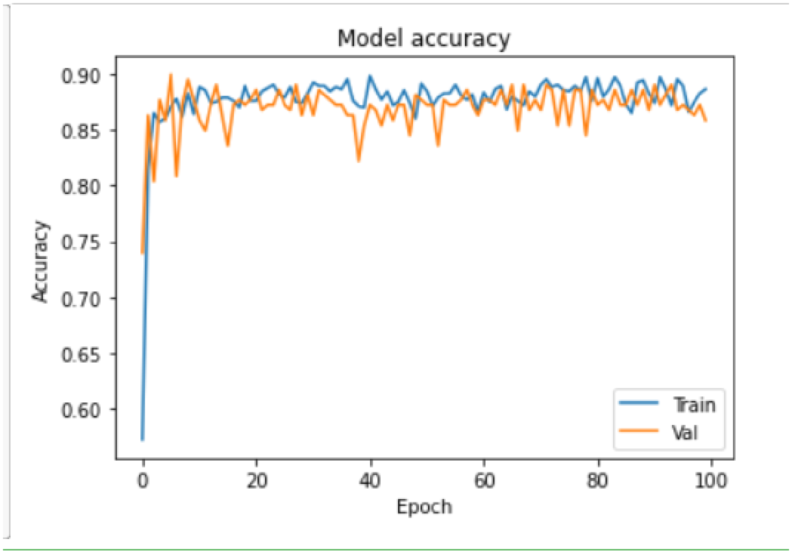


```
8904
Epoch 95/100
32/32 [==============================] - 2s 64ms/step - loss: 0.4279 - accuracy: 0.8953 - val_loss: 0.4703 - val_accuracy: 0.
8676
Epoch 96/100
32/32 [==============================] - 2s 60ms/step - loss: 0.4363 - accuracy: 0.8894 - val_loss: 0.4686 - val_accuracy: 0.
8721
Epoch 97/100
32/32 [==============================] - 2s 54ms/step - loss: 0.4557 - accuracy: 0.8659 - val_loss: 0.4532 - val_accuracy: 0.
8676
Epoch 98/100
32/32 [==============================] - 2s 56ms/step - loss: 0.4385 - accuracy: 0.8748 - val_loss: 0.4772 - val_accuracy: 0.
8630
Epoch 99/100
32/32 [==============================] - 2s 60ms/step - loss: 0.4351 - accuracy: 0.8826 - val_loss: 0.4628 - val_accuracy: 0.
8721
Epoch 100/100
32/32 [==============================] - 2s 59ms/step - loss: 0.4332 - accuracy: 0.8865 - val_loss: 0.5059 - val_accuracy: 0.
8584
```

```
plt.plot(hist_3.history['loss'])
plt.plot(hist_3.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper right')
plt.ylim(top=1.2, bottom=0)
plt.show()
```



```
plt.plot(hist_3.history['accuracy'])
plt.plot(hist_3.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='lower right')

plt.show()
```



CYCLE 5

Given a data set of support tickets. Each ticket also has an associated "urgency score" of between 0 and 3, and where 0 is "very urgent" and 3 is "not urgent". It would be useful if we could have a machine guess how urgent a ticket is, based on the description, so the urgent tickets can be resolved first

1. **For the given data set, perform text classification using SVM and find out the accuracy of the model.**

PROGRAM:

```
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer


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


from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_val_predict
from sklearn.model_selection import train_test_split


from sklearn.svm import LinearSVC
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
with open("tickets.txt") as f:
    tickets = f.read().strip().split("\n")


with open("labels_4.txt") as f:
    labels = f.read().strip().split("\n")


X_train, X_test, y_train, y_test = train_test_split(tickets, labels, test_size=0.1,
random_state=1337)
vectorizer = CountVectorizer()
svm = LinearSVC()
```

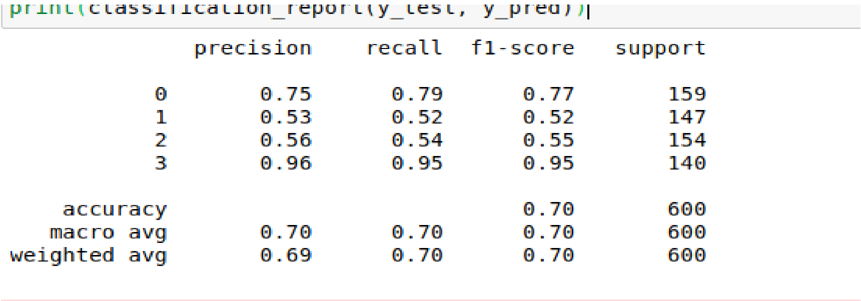
X_train = vectorizer.fit_transform(X_train)

X_test = vectorizer.transform(X_test)

_ = svm.fit(X_train, y_train)

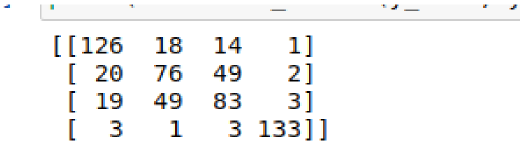
y_pred = svm.predict(X_test)

print(classification_report(y_test, y_pred))



```
print(classification_report(y_test, y_pred))
              precision    recall  f1-score   support

           0       0.75      0.79      0.77       159
           1       0.53      0.52      0.52       147
           2       0.56      0.54      0.55       154
           3       0.96      0.95      0.95       140

    accuracy                           0.70       600
   macro avg       0.70      0.70      0.70       600
weighted avg       0.69      0.70      0.70       600
```

print(confusion_matrix(y_test, y_pred))



```
[[126  18  14   1]
 [ 20  76  49   2]
 [ 19  49  83   3]
 [  3   1   3 133]]
```

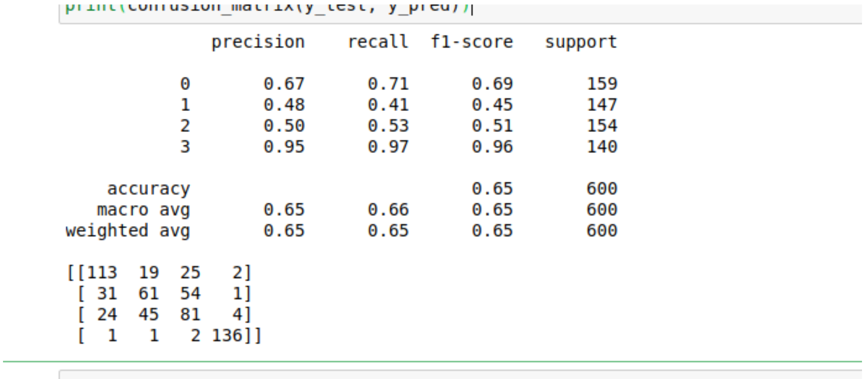
dt = DecisionTreeClassifier()

dt.fit(X_train, y_train)

y_pred = dt.predict(X_test)

print(classification_report(y_test, y_pred))

print(confusion_matrix(y_test, y_pred))



```
print(confusion_matrix(y_test, y_pred))
              precision    recall  f1-score   support

           0       0.67      0.71      0.69       159
           1       0.48      0.41      0.45       147
           2       0.50      0.53      0.51       154
           3       0.95      0.97      0.96       140

    accuracy                           0.65       600
   macro avg       0.65      0.66      0.65       600
weighted avg       0.65      0.65      0.65       600

[[113  19  25   2]
 [ 31  61  54   1]
 [ 24  45  81   4]
 [  1   1   2 136]]
```

2. K-means

Given dataset contains 200 records and five columns, two of which describe the customer's annual income and spending score. The latter is a value from 0 to 100. The higher the number, the more this customer has spent with the company in the past:

Functions to familiarize:

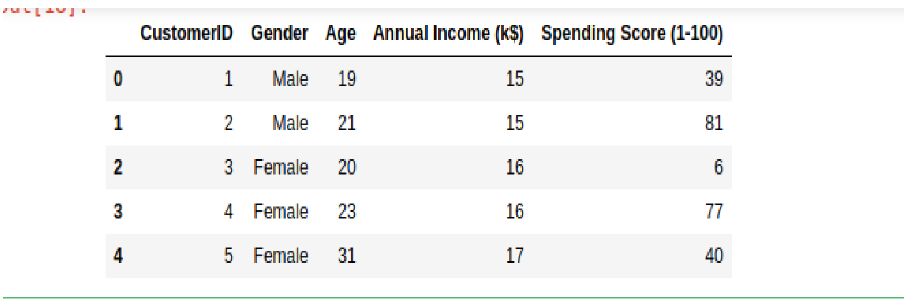
- The purpose of Kmeans.fit() is to train the model with data.
- The purpose of Kmeans.predict() is to apply a trained model to data

Using k means clustering create 6 clusters of customers based on their spending pattern.

- Visualize the same in a scatter plot with each cluster in a different color scheme.
- Display the cluster labels of each point.(print cluster indexes)
- Display the cluster centers.
- Use different values of K and visualize the same using scatter plot

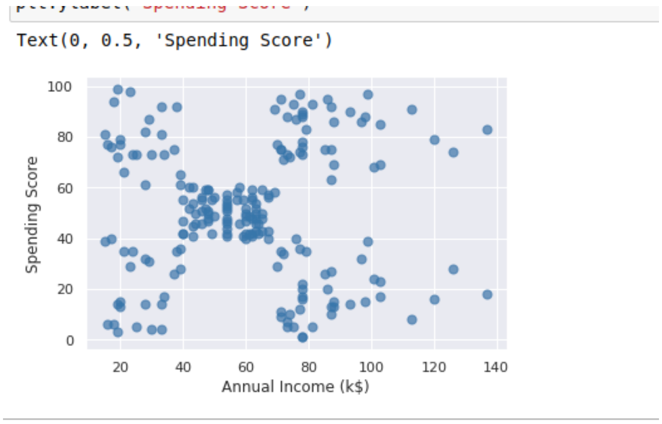
PROGRAM:

```
from sklearn.cluster import KMeans

from sklearn.datasets import make_blobs

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

sns.set()

%matplotlib inline

import pandas as pd

customers = pd.read_csv('customer_data.csv')

customers.head()
```



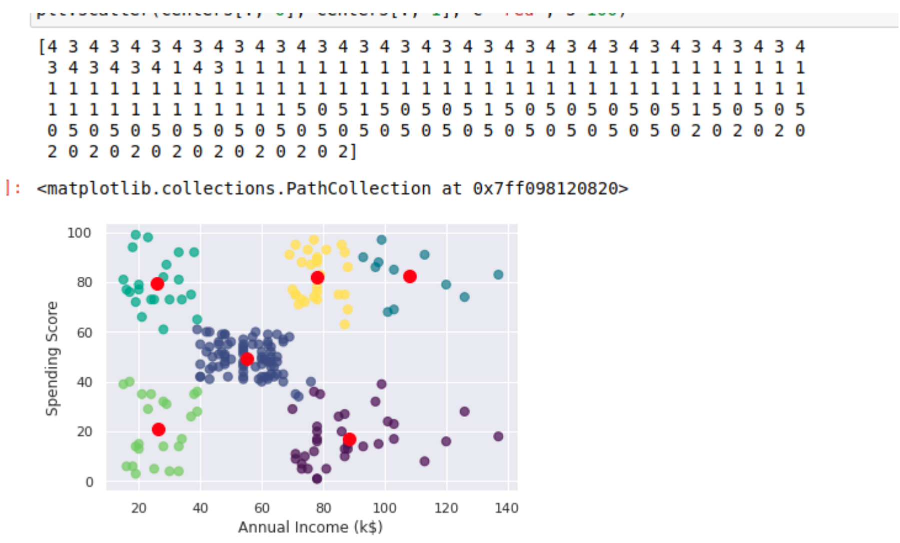
	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

```
points = customers.iloc[:, 3:5].values
x = points[:, 0]
y = points[:, 1]
plt.scatter(x, y, s=50, alpha=0.7)
plt.xlabel('Annual Income (k$)')plt.ylabel('Spending Score')
```



```
Text(0, 0.5, 'Spending Score')
```

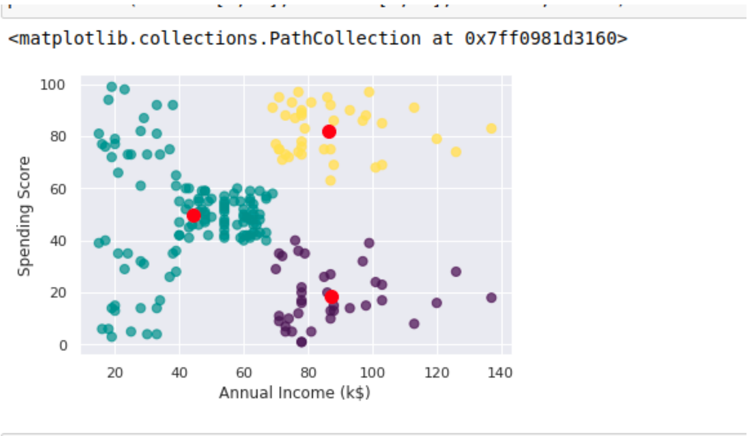
```
kmeans = KMeans(n_clusters=6, random_state=0)
kmeans.fit(points)
predicted_cluster_indexes = kmeans.predict(points)
print(predicted_cluster_indexes)
plt.scatter(x, y, c=predicted_cluster_indexes, s=50, alpha=0.7, cmap='viridis')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score')
centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='red', s=100)
```



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

```
]: <matplotlib.collections.PathCollection at 0x7ff098120820>
```

```
kmeans = KMeans(n_clusters=3, random_state=0)
kmeans.fit(points)
predicted_cluster_indexes = kmeans.predict(points)
plt.scatter(x, y, c=predicted_cluster_indexes, s=50, alpha=0.7, cmap='viridis')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score')
centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='red', s=100)
```



```
<matplotlib.collections.PathCollection at 0x7ff0981d3160>
```

3. NLP

For a given text,perform word and sentence tokenization.Remove the stop words from given text and create n-grams for different values of n.

PROGRAM:

```
import nltk


nltk.download('punkt')

import nltk

from nltk.corpus import stopwords

from nltk.tokenize import sent_tokenize,word_tokenize

text1 = "The data set given satisfies the requirement for model generation. This is used in Data Science Lab."

print(sent_tokenize(text1))
```

OUTPUT:

```
['The data set given satisfies the requirement for model generation.', 'This is used in Data Science
Lab.']
```

PROGRAM:

```
print(word_tokenize(text1))
```

OUTPUT:

```
['The', 'data', 'set', 'given', 'satisfies', 'the', 'requirement', 'for', 'model', 'generation', '.', 'This', 'is',
'used', 'in', 'Data', 'Science', 'Lab', '.']
```

PROGRAM:

```
import nltk

nltk.download('stopwords')

print(stopwords.words('english'))
```

OUTPUT:

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd",
'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself',
'it', "it's", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this',
'that', "that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had',
'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of',
'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above',
'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once',
'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other',
'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just',
'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn',
"couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't",
'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't",
'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]
```

PROGRAM:

```
text = word_tokenize(text1)

text= [word for word in text if word not in stopwords.words('english')]

print(text)
```

OUTPUT:

```
['The', 'data', 'set', 'given', 'satisfies', 'requirement', 'model', 'generation', '.', 'This', 'used', 'Data',
'Science', 'Lab', '.']
```

PROGRAM:

```
import nltk

nltk.download('averaged_perceptron_tagger')

print(nltk.pos_tag(text))
```

OUTPUT:

```
[('The', 'DT'), ('data', 'NN'), ('set', 'NN'), ('given', 'VBN'), ('satisfies', 'NNS'), ('requirement', 'VBP'),
('model', 'NN'), ('generation', 'NN'), ('.', '.'), ('This', 'DT'), ('used', 'VBN'), ('Data', 'NNP'), ('Science',
'NNP'), ('Lab', 'NNP'), ('.', '.')]
```

PROGRAM:

```
temp=zip(*[text[i:] for i in range(0,2)])

ans=[' '.join(ngram) for ngram in temp]

print(ans)
```

OUTPUT:

```
['The data', 'data set', 'set given', 'given satisfies', 'satisfies requirement', 'requirement model',
'model generation', 'generation .', '. This', 'This used', 'used Data', 'Data Science', 'Science Lab',
'Lab .']
```

PROGRAM:

```
temp=zip(*[text[i:] for i in range(0,4)])

ans=[' '.join(ngram) for ngram in temp]

print(ans)
```

OUTPUT:

```
['The data set given', 'data set given satisfies', 'set given satisfies requirement', 'given satisfies
requirement model', 'satisfies requirement model generation', 'requirement model generation .',


'model generation . This', 'generation . This used', '. This used Data', 'This used Data Science', 'used
Data Science Lab', 'Data Science Lab .']
```