LAB CYCLE-1

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

INPUT

```
a = int(input("enter lower bound :"))
b = int(input("enter upper bound :"))
for n in range(a,b+1):
if n > 1:
for i in range(2,n):
if (n % i) == 0 :
print(n)
break
```

OUTPUT

```
enter lower bound :0
enter upper bound :10
4
6
8
9
10
```

#2. Program to print the first N Fibonacci numbers.

INPUT

```
n=int(input("enter the value of n : "))
a=0
b=1
sum=0
count=1
print("fibonacci series : ")
while (count<=n):
print(sum,end="")
count+=1
a=b
b=sum
sum=a+b
```

OUTPUT

```
enter the value of n : 3
fibonacci series :
0 1 1
```

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

INPUT

```
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)

root1 = (-b-cmath.sqrt(d))/(2*a)
root2 = (-b+cmath.sqrt(d))/(2*a)

print("{0} and {1} are the roots of a quadratic equation.".format(root1,root2))
```

OUTPUT

```
enter the value of a : 1
enter the value of b : 5
enter the value of c : 6
(-3+0j) and (-2+0j) are the roots of a quadratic equation.
```

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

INPUT

```
n=int(input("enter any value : "))
sum=0
for i in range(1,n):
if n % i == 0:
sum=sum+i
if(sum==n):
print(n,"is a perfect number.")
else:
print(n,"is not a perfect number.")
```

OUTPUT

```
enter any value : 35
35 is not a perfect number.
```

#5. Program to display amstrong numbers upto 1000.

INPUT

```
for n in range(0,1000):
sum=0
temp=n
while temp>0:
a=temp%10
sum+=a**3
temp//=10
if n==sum:
print(n)
```

OUTPUT

```
1
64
125
153
216
370
371
407
729
```

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

INPUT

```
l=[]
n=int(input("enter the no. elements :"))
for i in range(0,n):
x=input()
l.append(x)
print("before sorting elements")
for i in l:
print(i,end="")
for i in range(0,len(l)):
for j in range(i+1,len(l)):
if l[j]< l[i]:
temp=l[j]
l[j]=l[i]
l[i]=temp
```

```
print("\nafter bubble sort")
for i in l:
print(i,end="")
```

OUTPUT

```
enter the no. elements :3
3
2
1
before sorting elements
3 2 1
after bubble sort
1 2 3
```

#7. 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.

INPUT

```
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

print(repeat_times(12))
print(repeat_times(9))
print(repeat_times(21))
```

OUTPUT

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

INPUT

```
def absent_digits(n):
all_nums = set([0,1,2,3,4,5,6,7,8,9])
```

```
n = set([int(i) for i in n])
n = n.symmetric_difference(all_nums)
n = sorted(n)
return n
print(absent_digits([9,8,3,2,2,0,9,7,6,3]))
```

OUTPUT

```
[1, 4, 5]
```

LAB CYCLE-2

#1. Create a 2 dimensional array (2X3) with elements belonging to complex datatype and print it. Also display

a. the no: of rows and columns

b. dimension of an array

c. reshape the same array to 3X2

INPUT

```
import numpy as np
array_2d=np.array([[complex(1,2),complex(2,3),complex(3,4)],[complex(4,5),complex(5,6),complex(6,7)]])
print(array_2d)
print("the no. of rows and columns :",array_2d.shape)
print("dimension of an array",array_2d.ndim)
print("reshape the same array to 3x2",array_2d.reshape(3,2))
```

OUTPUT

```
[[1.+2.j 2.+3.j 3.+4.j]
 [4.+5.j 5.+6.j 6.+7.j]]
the no. of rows and columns : (2, 3)
dimension of an array 2
reshape the same array to 3x2 [[1.+2.j 2.+3.j]
 [3.+4.j 4.+5.j]
 [5.+6.j 6.+7.j]]
```

#2. 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

INPUT

```
import numpy as np
array_1d=np.array([1,2,3,4,5,6,7,8,9,10])
print("First 4 elements ",array_1d[:4])
print("Last 6 elements ",array_1d[4:])
print("Elements from index 2 to 7 ",array_1d[2:7])
```

OUTPUT

```
First 4 elements  [1 2 3 4]
Last 6 elements  [ 5  6  7  8  9 10]
Elements from index 2 to 7  [3 4 5 6 7]
```

#3. 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

INPUT

```
import numpy as np

array_1d=np.array([0,2,4,6,8,10,12,14,16,18,20,22,24,26,28])
print("Elements from index 2 to 8 with step 2",array_1d[2:8:2])
print("Last 3 elements of the array using negative index",array_1d[-3:-1])
print("Alternate elements of the array",array_1d[::2])
print("Display the last 3 alternate elements",array_1d[-3:-1:2])
```

OUTPUT

```
Elements from index 2 to 8 with step 2 [ 4  8 12]
Last 3 elements of the array using negative index [24 26]
Alternate elements of the array [ 0  4  8 12 16 20 24 28]
Display the last 3 alternate elements [24]
```

#4. 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

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

INPUT

```
import numpy as np
array_2d=np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,16]])
print(array_2d)
print("Display all elements excluding the first row")
print(array_2d[1:4,:])
print("Display all elements excluding the last column")
print(array_2d[:,0:3])
print("Display the elements of 1 st and 2 nd column in 2 nd and 3 rd row")
print(array_2d[1:3,1:3])
print("Display the elements of 2 nd and 3 rd column")
print(array_2d[:,1:3])
print("Display 2 nd and 3 rd element of 1 st row")
print(array_2d[0,1:3])
```

```
array=np.array([0,1,2,3,4,5,6,7,8,9,10])
print("Display the elements from indices 4 to 10 in descending order(use–values)")
print(array[10:4:-1])
```

OUTPUT

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]]
Display all elements excluding the first row
[[ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]]
Display all elements excluding the last column
[[ 1  2  3]
 [ 5  6  7]
 [ 9 10 11]
 [13 14 15]]
Display the elements of 1 st and 2 nd column in 2 nd and 3 rd row
[[ 6  7]
 [10 11]]
Display the elements of 2 nd and 3 rd column
[[ 2  3]
 [ 6  7]
 [10 11]
 [14 15]]
Display 2 nd and 3 rd element of 1 st row
[2 3]
Display the elements from indices 4 to 10 in descending order(use-
values)
[10  9  8  7  6  5]
```

#5.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

INPUT

```
import numpy as np


x=np.array([[1,2],[3,4]])
y=np.array([[5,6],[7,8]])
print("Add the 2 matrices")
print(np.add(x,y))
print("Subtract 2 matrices")
print(np.subtract(x,y))
print("Multiply the individual elements of matrix")
print(np.multiply(x,y))
print("Divide the elements of the matrices")
print(np.divide(x,y))
print("Perform matrix multiplication")
print(np.dot(x,y))
print("Display transpose of the matrix")
print(x.transpose())
print(y.transpose())
print("Sum of diagonal elements of a matrix")
print(np.trace(x))
print(np.trace(y))
```

OUTPUT

```
Add the 2 matrices
[[ 6  8]
 [10 12]]
Subtract 2 matrices
[[-4 -4]
 [-4 -4]]
Multiply the individual elements of matrix
[[ 5 12]
 [21 32]]
Divide the elements of the matrices
[[0.2        0.33333333]
 [0.42857143 0.5       ]]
Perform matrix multiplication
[[19 22]
 [43 50]]
Display transpose of the matrix
[[1 3]
 [2 4]]
[[5 7]
 [6 8]]
Sum of diagonal elements of a matrix
5
13
```

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

i) inverse

ii) rank of matrix

iii) Determinant

iv) transform matrix into 1D array

v) eigen values and vectors.

INPUT

```
import numpy as np
matrix=np.random.randint(0,10,4).reshape(2,2)
print(matrix)
inverse=np.linalg.inv(matrix)
```

```
print("inverse of matrix")
print(inverse)
rank=np.linalg.matrix_rank(matrix)
print("rank of matrix",rank)
det=np.linalg.det(matrix)
print("Determinant of matrix",det)
array_1d=matrix.flatten()
print("transform matrix into 1D array")
print(array_1d)
eigen=np.linalg.eig(matrix)
print("eigen values and vectors")
print(eigen)
```

OUTPUT

```
[[3 1]
 [0 2]]
inverse of matrix
[[ 0.33333333 -0.16666667]
 [ 0.          0.5       ]]
rank of matrix 2
Determinant of matrix 6.0
transform matrix into 1D array
[3 1 0 2]
eigen values and vectors
(array([3., 2.]), array([[ 1.        , -0.70710678],
       [ 0.        ,  0.70710678]]))
```

#7. Create a matrix X with suitable rows and columns

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

ii) Display identity matrix of the given square matrix.

iii) Display each element of the matrix to different powers.

iv) Create a matrix Y with same dimension as X and perform the operation X 2 +2Y

```
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)
```

OUTPUT

```
Display the cube of each element of the matrix using different
methods (use multiply(), *, power(),**)
power() [[216   0]
 [125   0]]
multiply()
[[216   0]
 [125   0]]
**
[[216   0]
 [125   0]]
*
[[18  0]
 [15  0]]
Display identity matrix of the given square matrix.
[[1 0]
 [0 1]]
Display each element of the matrix to different powers.
[[46656     1]
 [ 3125     1]]
Create a matrix Y with same dimension as X and perform the
operation X^2 +2Y
[[47088     0]
 [15875     0]]
```

#8. Write a program to find out the value of X using solve(), given A and b.

INPUT

```
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)
```

OUTPUT

```
[[15.]
 [ 7.]
 [10.]]
```

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

INPUT

```
from numpy import array
from numpy import diag
from numpy import dot
from numpy import zeros
from scipy.linalg import svd
A = array([[1, 2], [3, 4], [5, 6]])
print(A)
U, s, VT = svd(A)
print(U)
print(s)
print(VT)
Sigma = zeros((A.shape[0], A.shape[1]))
Sigma[:A.shape[1], :A.shape[1]] = diag(s)
B = U.dot(Sigma.dot(VT))
print(B)
```

OUTPUT

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

LAB 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	$10,000

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

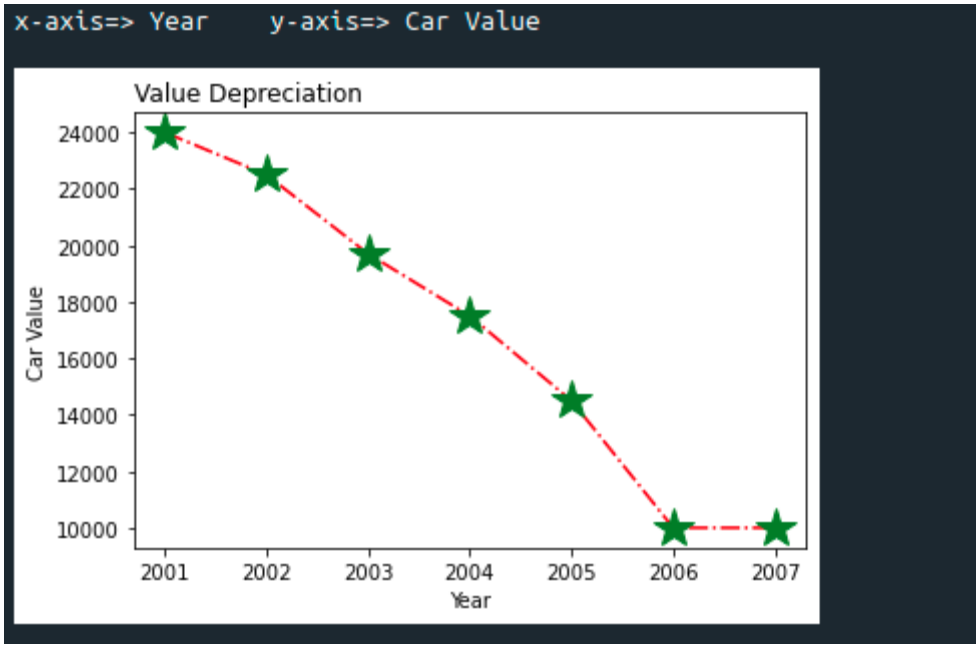
Subplot() provides multiple plots in one figure.

INPUT

```
import matplotlib.pyplot as plt
import numpy as np
x = np.array([2001,2002,2003,2004,2005,2006,2007])
y= np.array([24000,22500,19700,17500,14500,10000,10000])
print("x-axis=> Year    y-axis=> Car Value")
plt.plot(x,y, ls = '-.',color = 'r',marker= '*', ms='20', mfc='green', mec='green')
```

```
plt.title("Value Depreciation", loc='left')
plt.xlabel("Year")
plt.ylabel("Car Value")
plt.show()
```

OUTPUT



#2. 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

a) 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

b) Properties for the Graph 2:

X label- Days of Week

Y label-Sale of Food

Title-Sales Data2 (center aligned)

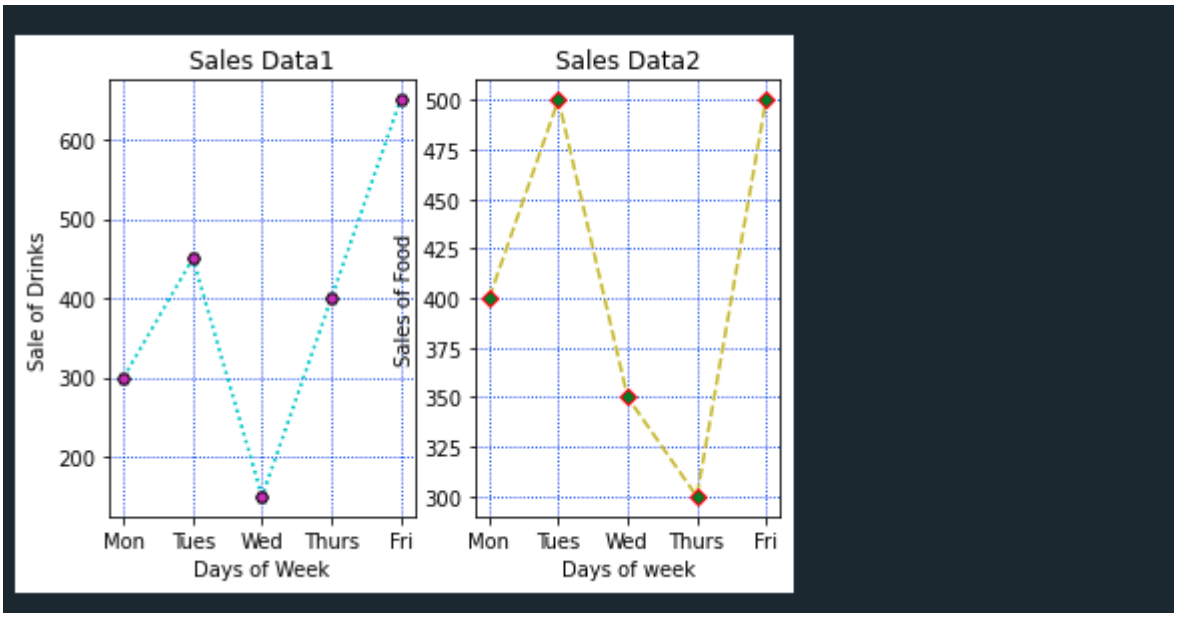
Line –dashed with yellow color

Points- diamond shape with color green and outline red

INPUT

```
import matplotlib.pyplot as plt
import numpy as np
x= np.array(['Mon','Tues','Wed','Thurs','Fri'])
y= np.array([300,450,150,400,650])
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", ls=':')
c= np.array(['Mon','Tues','Wed','Thurs','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("Sales of Food")
plt.plot(c,v, '--y')
plt.plot(c,v, 'Dg',mec='r')
plt.grid(color='blue', ls=':')
plt.show()
```

OUTPUT



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

Create scatter plot for each Segment with following properties within one graph

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

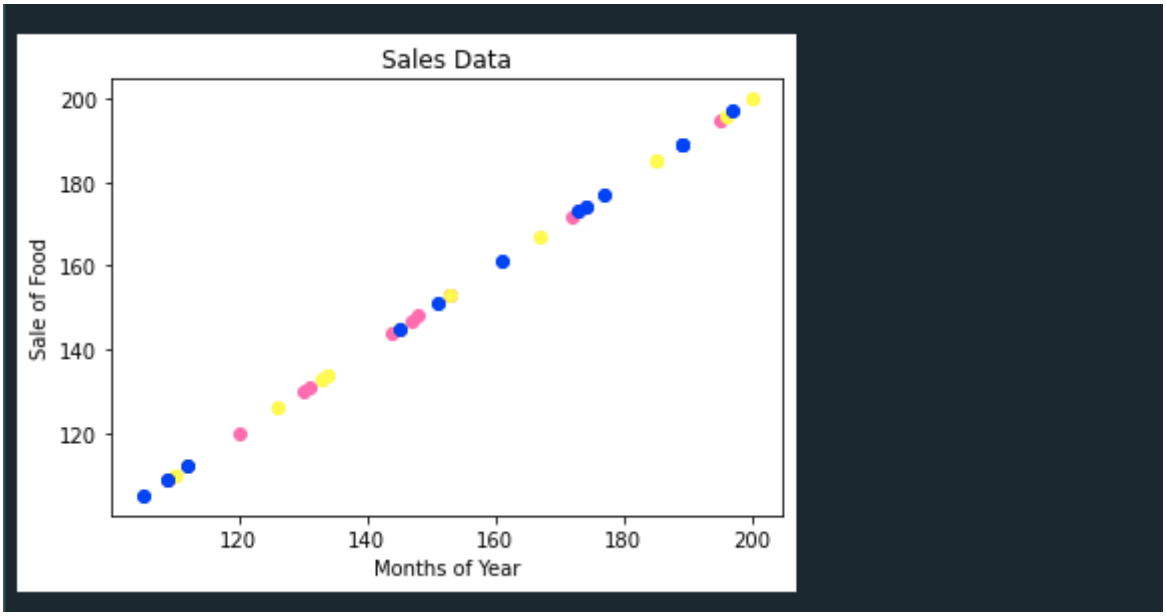
Color for Super luxury segment-blue

INPUT

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

plt.title('Sales Data')
plt.xlabel('Months of Year')
```

```
plt.ylabel('Sale of Food')
x = np.array([173,153,195,147,120,144,148,109,174,130,172,131])
y = np.array([173,153,195,147,120,144,148,109,174,130,172,131])
plt.scatter(x,y, color='hotpink')
x = np.array([185,185,126,134,196,153,112,133,200,145,167,110])
y = np.array([185,185,126,134,196,153,112,133,200,145,167,110])
plt.scatter(x, y, color='yellow')
x = np.array([189,189,105,112,173,109,151,197,174,145,177,161])
y = np.array([189,189,105,112,173,109,151,197,174,145,177,161])
plt.scatter(x, y, color='blue')
plt.show()
```

OUTPUT



#4. Display the above data using multiline plot(3 different lines in same graph)

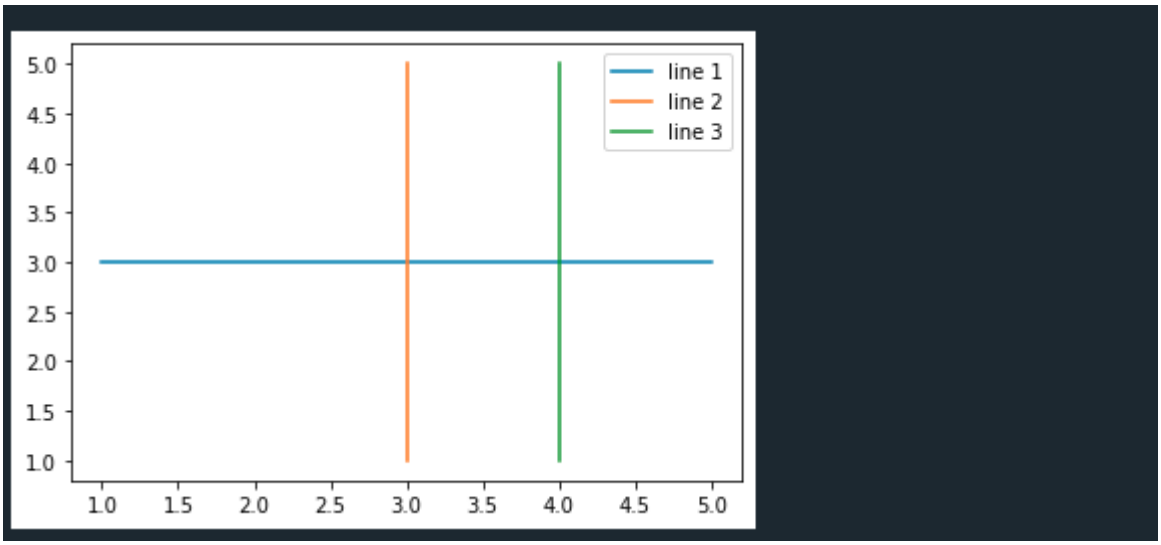
Display the description of the graph in upper right corner(use legend())

Use different colors and line styles for 3 different lines

INPUT

```
import matplotlib.pyplot as plt
import numpy as np
x = [1,2,3,4,5]
y = [3,3,3,3,3]
z = [4,4,4,4,4]
plt.plot(x, y, label='line 1')
plt.plot(y, x, label='line 2')
plt.plot(z, x, label='line 3')
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.

Create a bar graph with

X axis -mode of Transport and Y axis 'frequency'

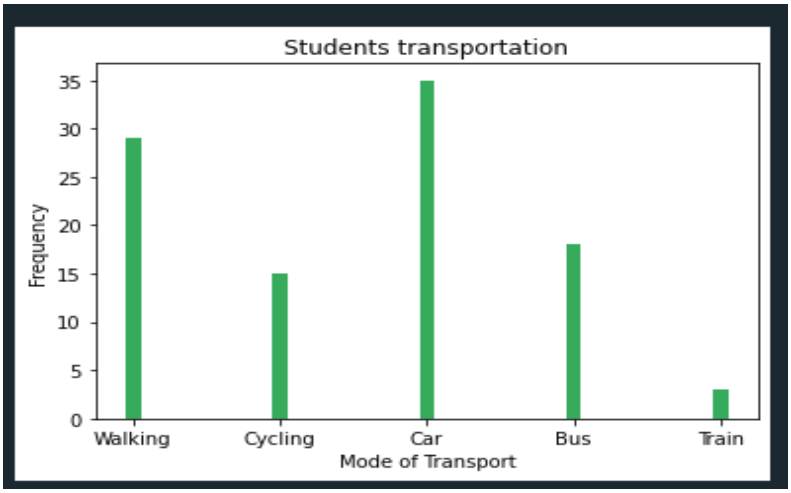
Provide appropriate labels and title

Width .1, color green

INPUT

```
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.

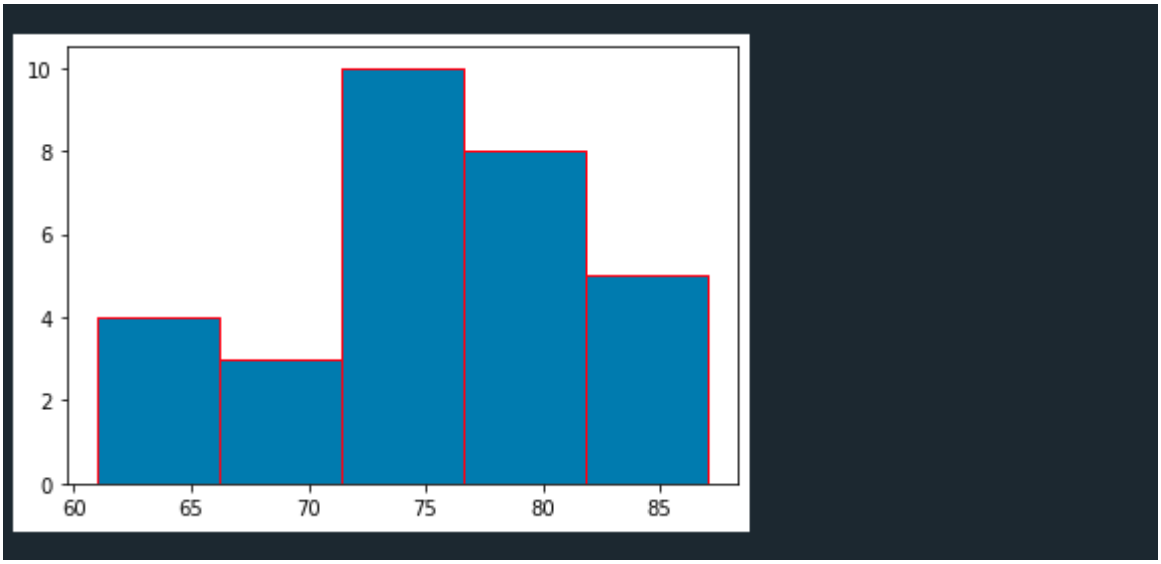
The height of the trees (in inches): 61, 63, 64, 66, 68, 69, 71, 71.5, 72, 72.5, 73, 73.5, 74, 74.5, 76, 76.2,76.5, 77, 77.5, 78, 78.5, 79, 79.2, 80, 81, 82, 83, 84, 85, 87.

Create a histogram with a bin size of 5

INPUT

```
import matplotlib.pyplot as plt
height = [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]
plt.hist(height, edgecolor='red', bins=5)
plt.show()
```

OUTPUT



DATA HANDLING USING 'Pandas' and DATA VISUALIZATION USING 'Seaborn'

#7. Using the pandas function read_csv(), read the given 'iris' data set.

INPUT

```
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)
```

i) Shape of the data set.

```
print("shape :",iris.shape)
```

shape : (151, 5)

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

```
print("first five rows")
print(iris.head())
print("****************")
print("last five rows")
print(iris.tail())
```

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
***************
last five rows
    sepal_length sepal_width petal_length petal_width       type
146          6.7           3          5.2         2.3  Virginica
147          6.3         2.5            5         1.9  Virginica
148          6.5           3          5.2           2  Virginica
149          6.2         3.4          5.4         2.3  Virginica
150          5.9           3          5.1         1.8  Virginica
```

iii) Size of dataset.

```
print("size :",iris.size)
```

OUTPUT

```
size : 755
```

iv) No. of samples available for each variety.

```
print("no. of samples available for each type")
print(iris["type"].value_counts())
```

OUTPUT

```
no. of samples available for each type
Setosa        50
Versicolor    50
Virginica     50
variety        1
Name: type, dtype: int64
```

v) Description of the data set(use describe).

print("description of the data set")

print(iris.describe())

OUTPUT

```
description of the data set
       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
```

matplotlib inline

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

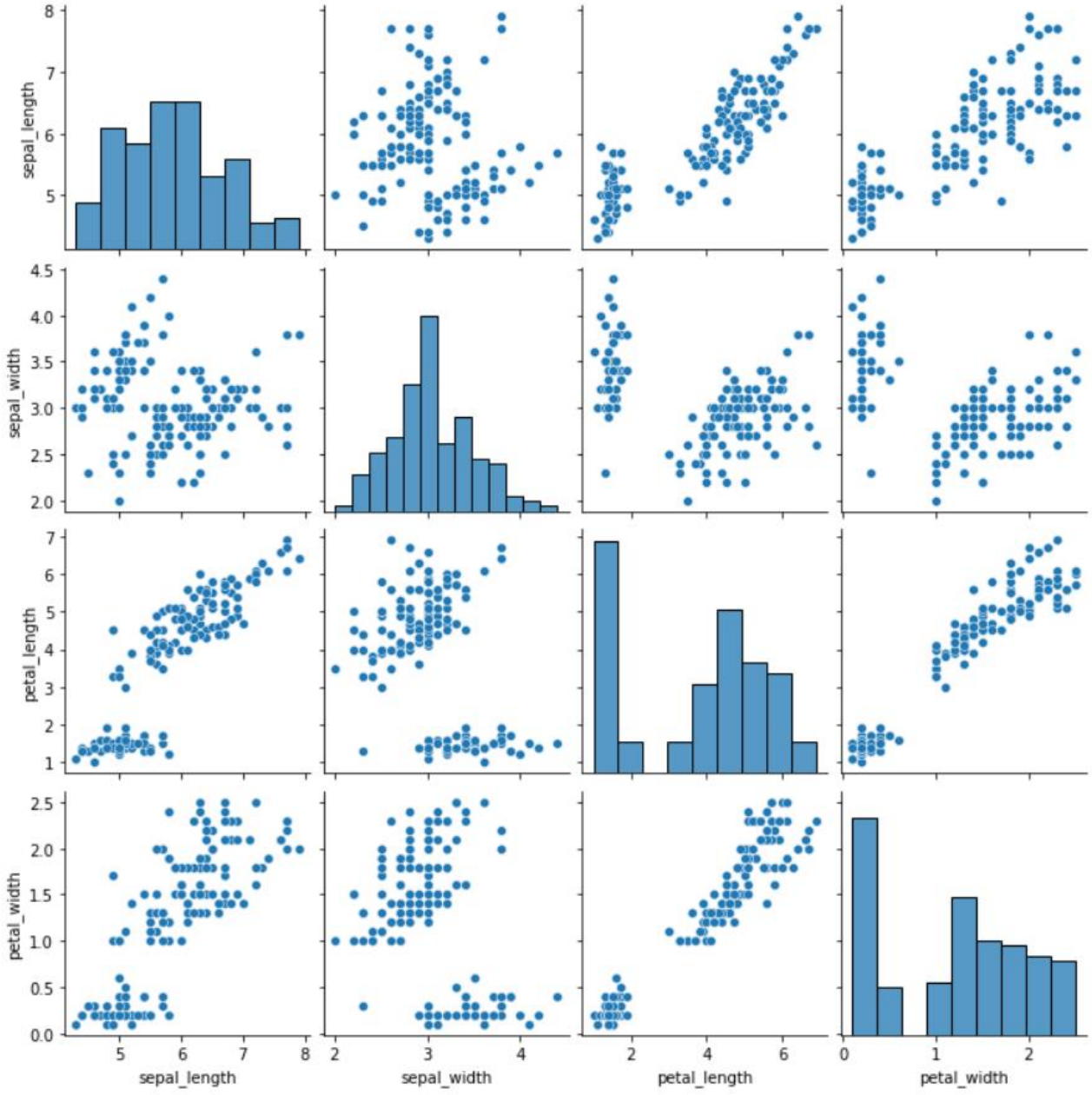
INPUT

iris = sns.load_dataset("iris")

sns.pairplot(iris)

OUTPUT

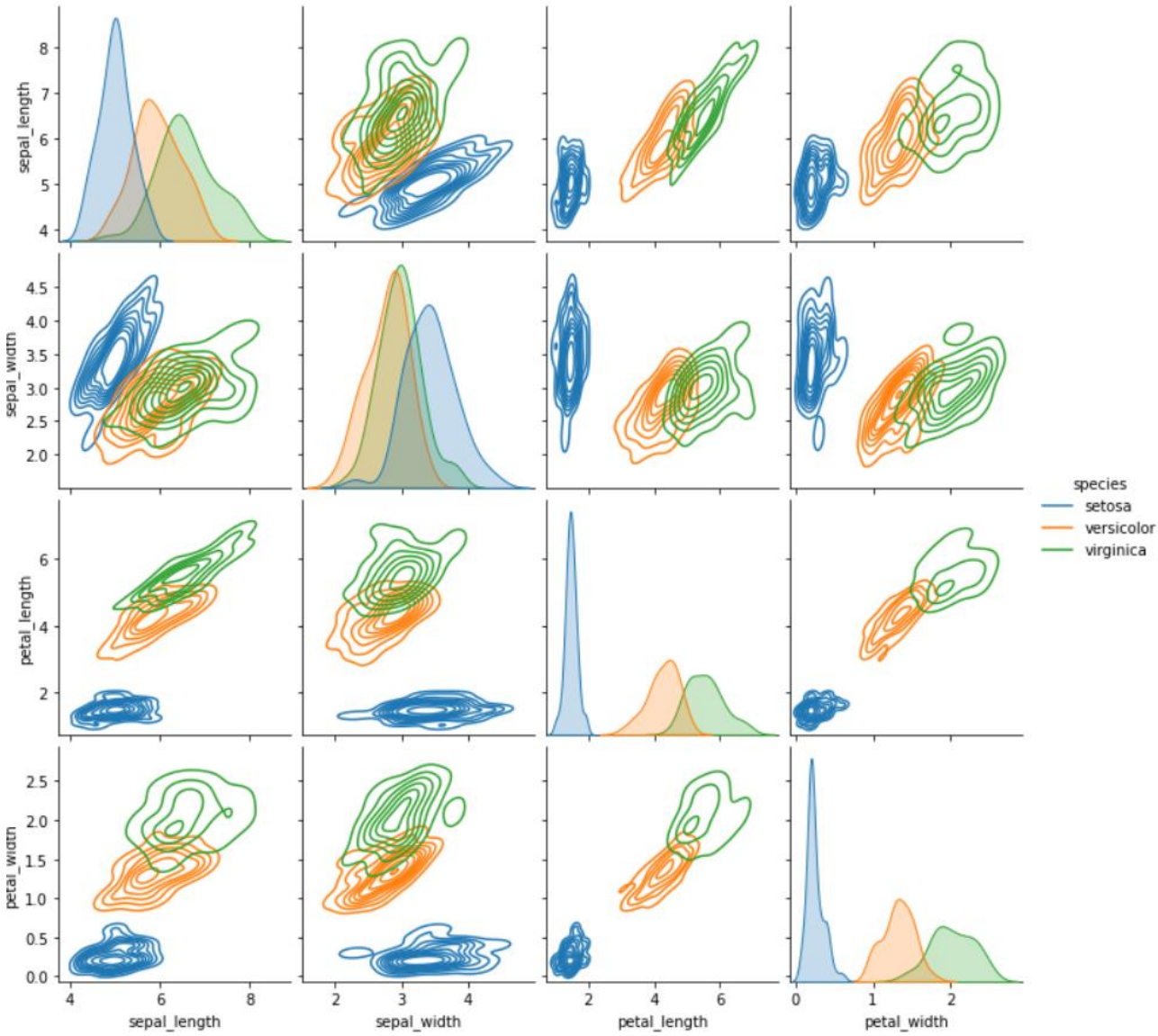
```
<seaborn.axisgrid.PairGrid at 0x7f5620ec8520>
```




```
sns.pairplot(iris, hue="species", kind="kde")
```

OUTPUT

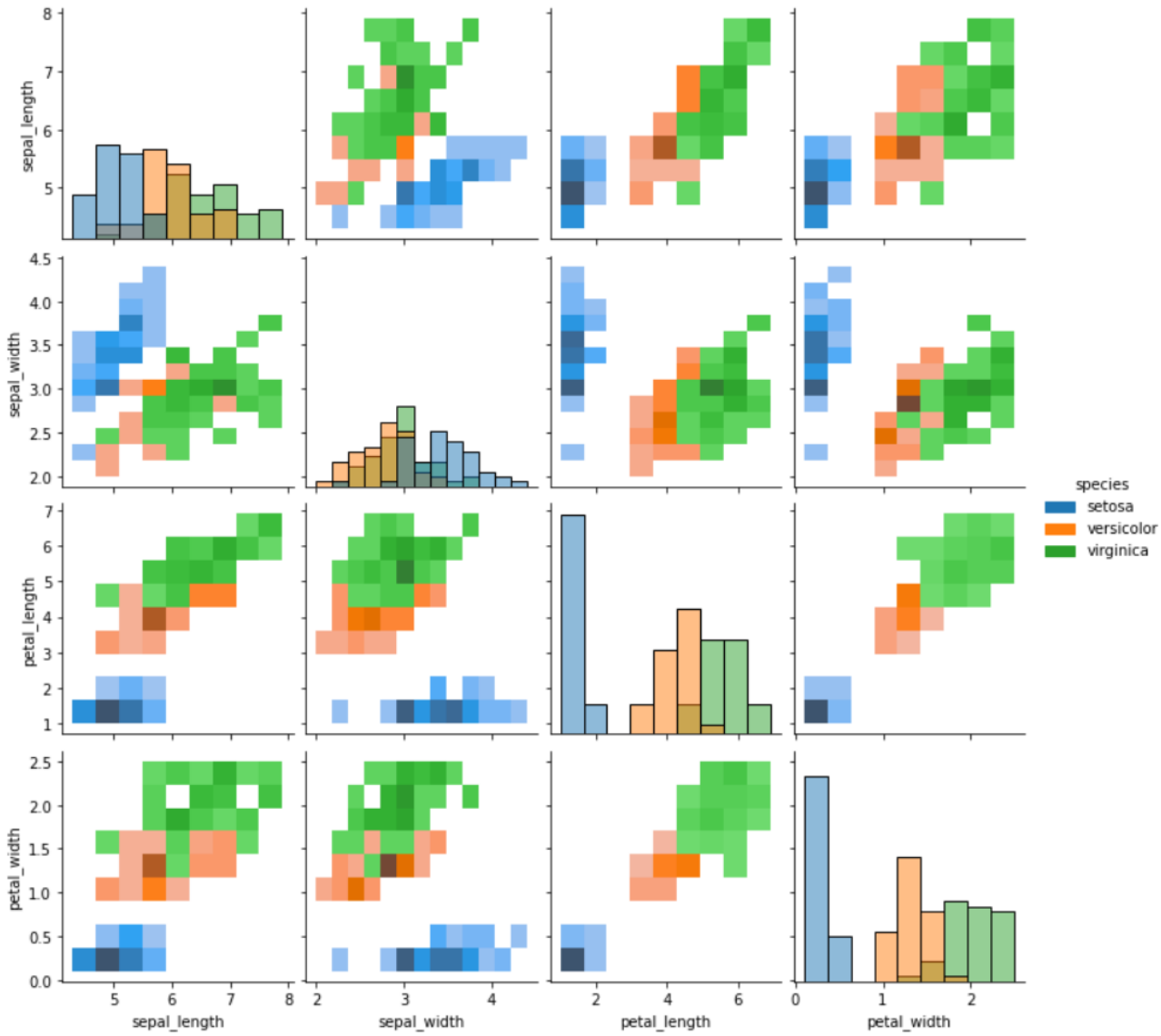
```
<seaborn.axisgrid.PairGrid at 0x7f5607971910>
```



sns.pairplot(iris, hue="species", kind="hist")

OUTPUT

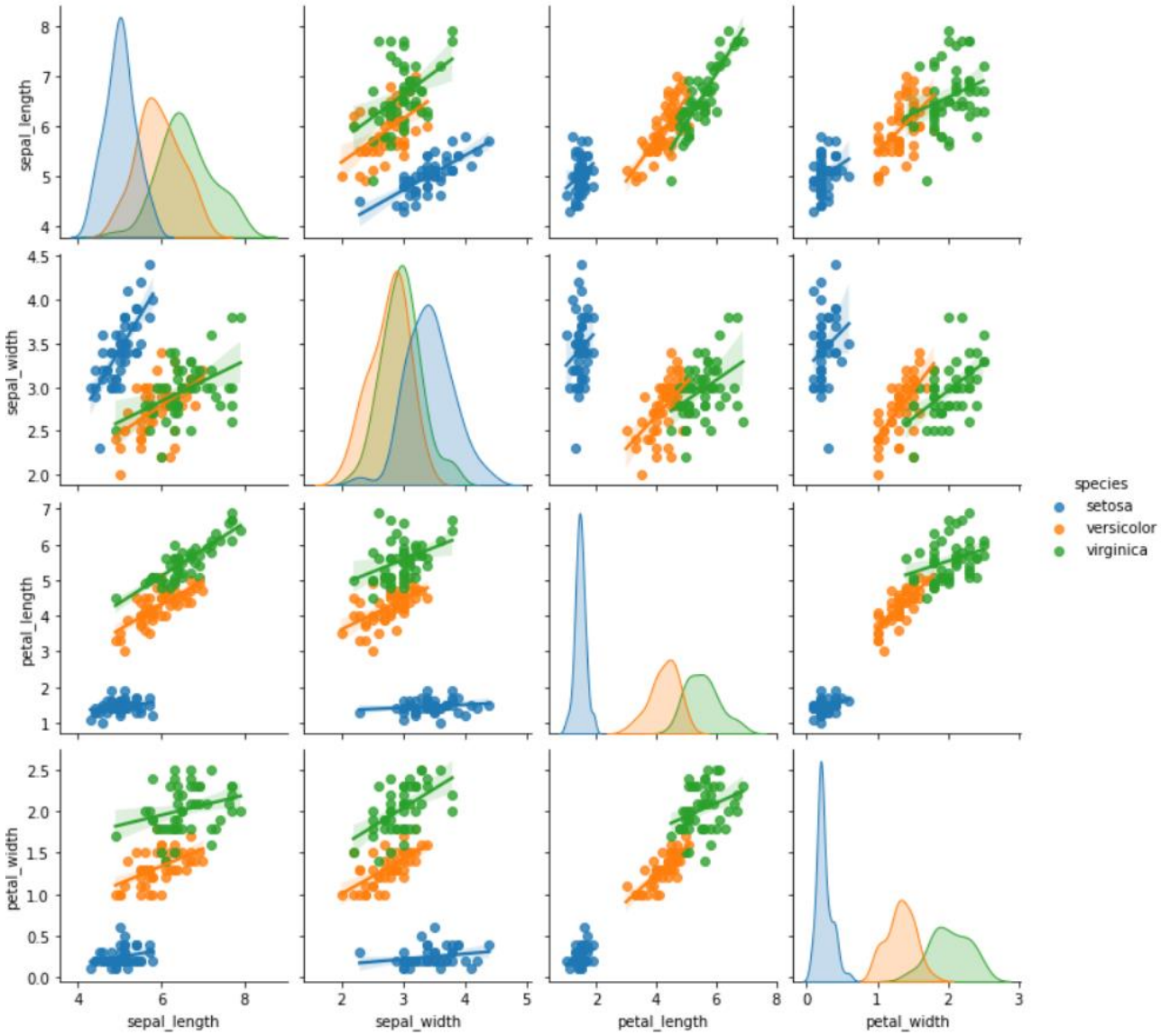
```
<seaborn.axisgrid.PairGrid at 0x7f5606015850>
```




```
sns.pairplot(iris, hue="species", kind="reg")
```

OUTPUT

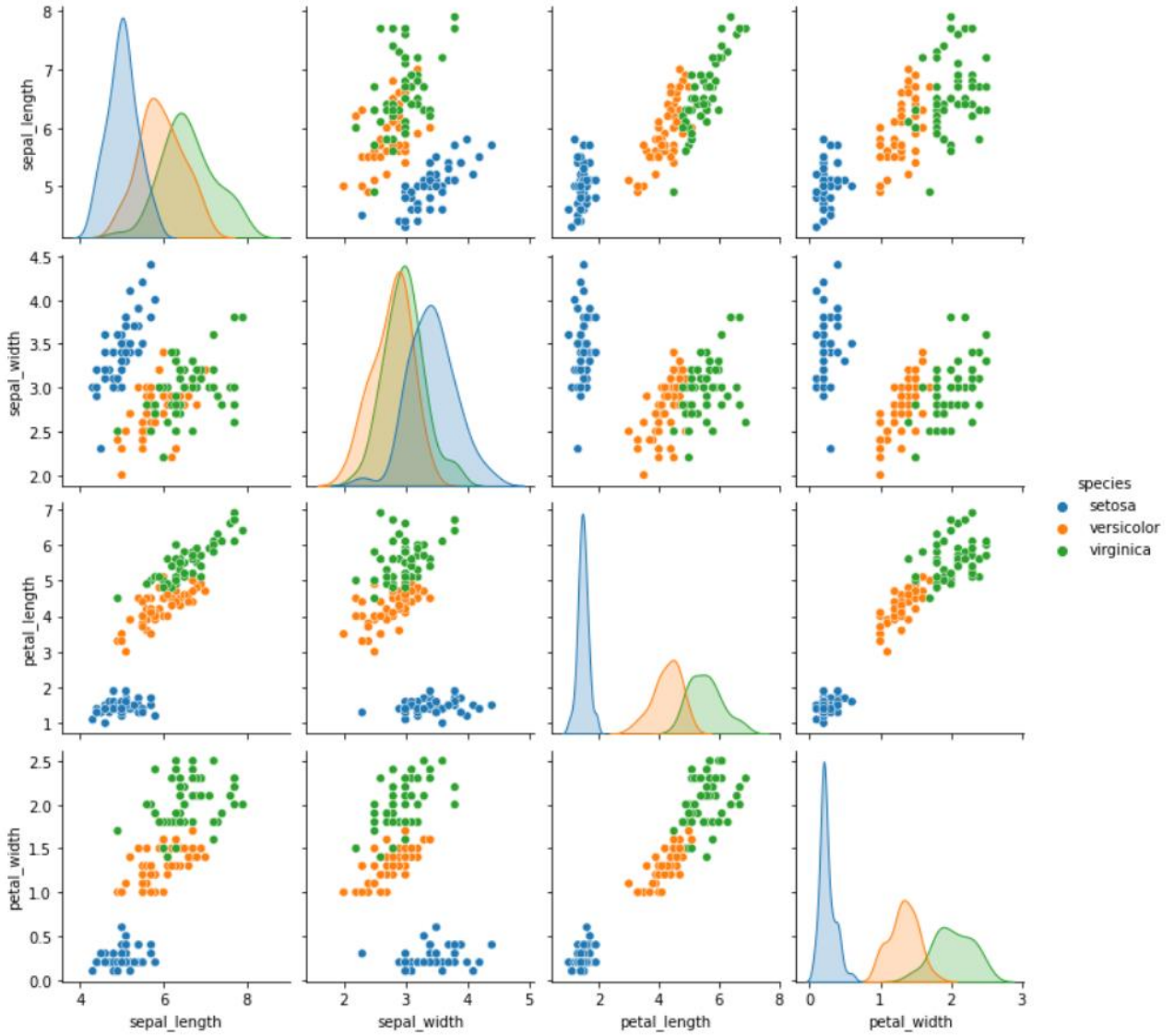
```
<seaborn.axisgrid.PairGrid at 0x7f56067c9be0>
```




```
sns.pairplot(iris, hue="species", kind="scatter")
```

OUTPUT

```
<seaborn.axisgrid.PairGrid at 0x7f56079710d0>
```



#9. Using the iris data set,get familiarize with functions:

1) displot()

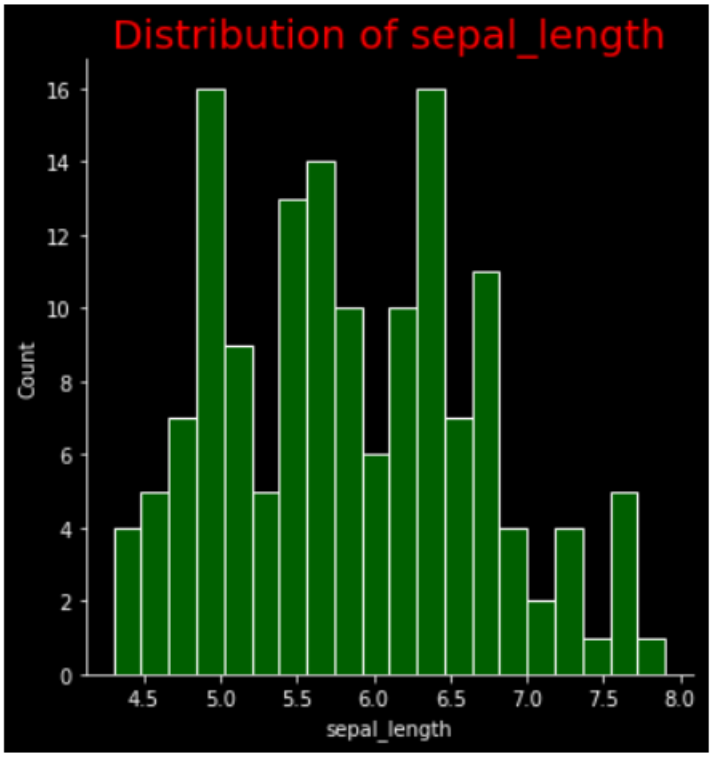
2) histplot()

3) relplot()

INPUT

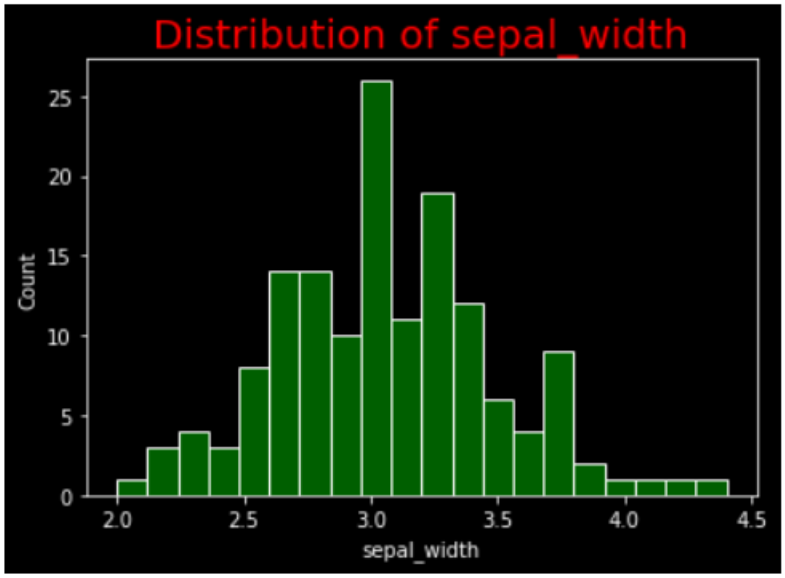
```
plt.style.use("dark_background")
sns.displot(iris.sepal_length, bins=20, color="g")
plt.title("Distribution of sepal_length", fontsize=20, color = 'red')
plt.show()
```

OUTPUT




```
sns.histplot(iris.sepal_width, bins=20, color="g")
plt.title("Distribution of sepal_width", fontsize=20, color = 'red')
plt.show()
```

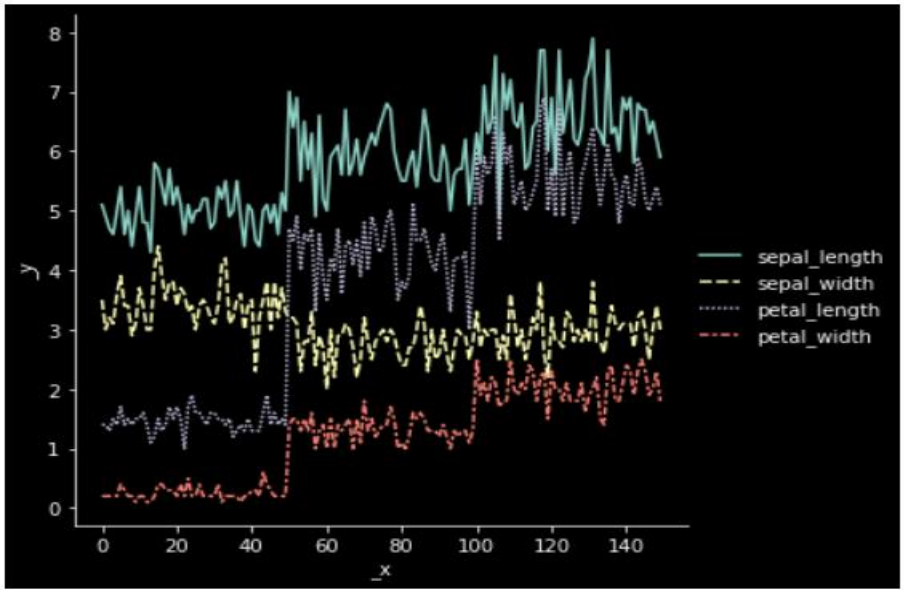
OUTPUT



sns.relplot(data=iris,kind="line")

OUTPUT

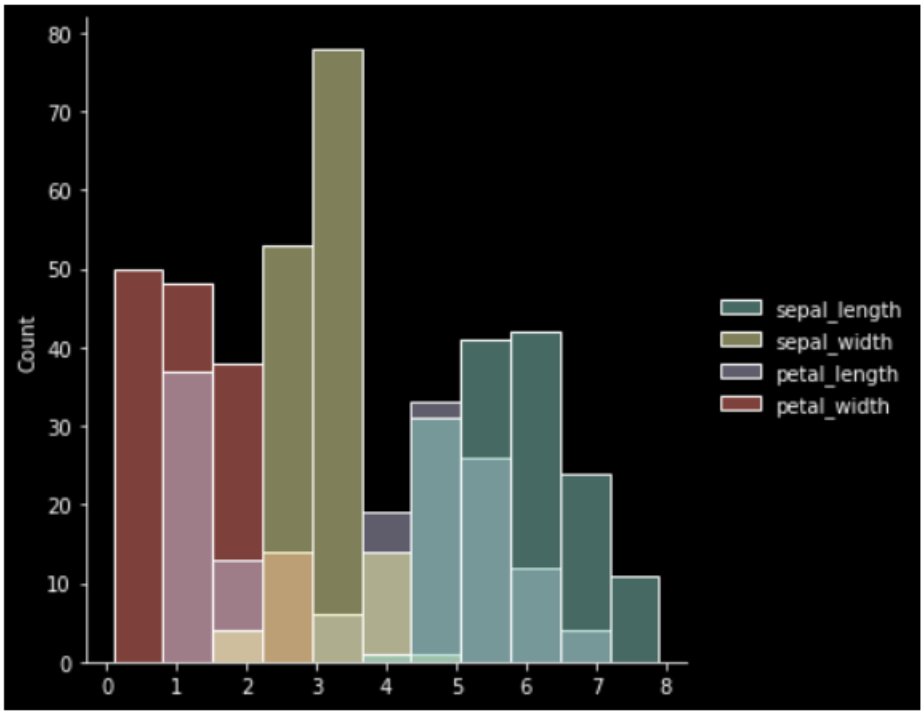
```
<seaborn.axisgrid.FacetGrid at 0x7f560452f430>
```



sns.displot(iris)

OUTPUT

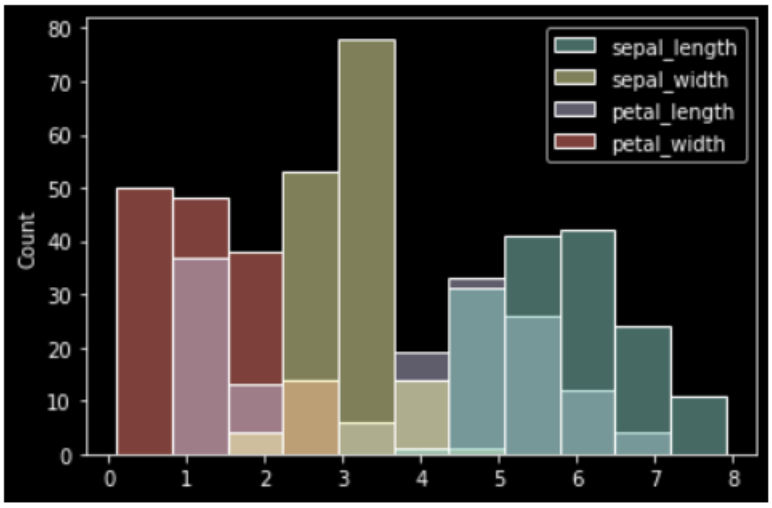
```
<seaborn.axisgrid.FacetGrid at 0x7f56043dc550>
```



sns.histplot(iris)

OUTPUT

```
<AxesSubplot:ylabel='Count'>
```



LAB CYCLE-4

KNN Algorithm

#1. 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.

INPUT

```
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))
```

OUTPUT

```
precision    recall  f1-score   support

      Setosa       1.00      1.00      1.00        13
  Versicolor       1.00      1.00      1.00        10
   Virginica       1.00      1.00      1.00         7

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        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})
```

OUTPUT

```
Accuracy :  0.9333333333333333
```

#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.

INPUT

```
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
 . . .
 31  0.1189    568 non-null    float64
dtypes: float64(30), int64(1), object(1)
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'
```

INPUT

```
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))
```

OUTPUT

```
              precision    recall  f1-score   support

           B       0.93      0.97      0.95        78
           M       0.94      0.83      0.88        36

    accuracy                           0.93       114
   macro avg       0.93      0.90      0.92       114
weighted avg       0.93      0.93      0.93       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})
```

OUTPUT

```
Accuracy :  0.9298245614035088
```

Naive Bayes Classification Algorithm

#3. 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**

```
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)
```

OUTPUT

	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

INPUT

```
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
```

OUTPUT

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

INPUT

```
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
```

OUTPUT

```
Accuracy :  0.9333333333333333


array([[10,  0,  0],
       [ 0, 11,  0],
       [ 0,  2,  7]])
```

INPUT

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

OUTPUT

	Real Values	Predicted Values
0	Setosa	Setosa
1	Versicolor	Versicolor
2	Versicolor	Versicolor
3	Versicolor	Versicolor
4	Setosa	Setosa
5	Virginica	Virginica
6	Virginica	Versicolor
7	Versicolor	Versicolor
8	Virginica	Virginica
9	Virginica	Virginica
10	Setosa	Setosa
11	Setosa	Setosa
12	Setosa	Setosa
13	Versicolor	Versicolor

	Real Values	Predicted Values
14	Virginica	Virginica
15	Versicolor	Versicolor
16	Versicolor	Versicolor
17	Setosa	Setosa
18	Virginica	Versicolor
19	Setosa	Setosa
20	Setosa	Setosa
21	Virginica	Virginica
22	Setosa	Setosa
23	Versicolor	Versicolor
24	Versicolor	Versicolor
25	Virginica	Virginica
26	Versicolor	Versicolor
27	Virginica	Virginica
28	Setosa	Setosa
29	Versicolor	Versicolor

Real Values **Predicted Values**

Decision Tree Algorithm

#4. Use car details CSV file and implement decision tree algorithm

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

INPUT

```
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()
```

OUTPUT

	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()

OUTPUT

```
<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
```

INPUT

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


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

OUTPUT

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

INPUT

```
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()
```

OUTPUT

	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()
```

OUTPUT

```
<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
```

INPUT

```
X = data.iloc[:,:-1]
y = data.iloc[:,-1]
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y, test_size=0.3, random_state=0)
dtree = tree.DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=0)
dtree.fit(X_train, y_train)
```

OUTPUT

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

INPUT

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

OUTPUT

```
Accuracy: 0.82
```

INPUT

```
count_misclassified = (y_test != y_pred).sum()
print('Misclassified samples: {}'.format(count_misclassified))
```

OUTPUT

```
Misclassified samples: 96
```

Simple Linear Regression

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

INPUT

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

student = pd.read_csv('student_scores.csv')
student.head()
```

OUTPUT

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

INPUT

student.describe()

OUTPUT

	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()

OUTPUT

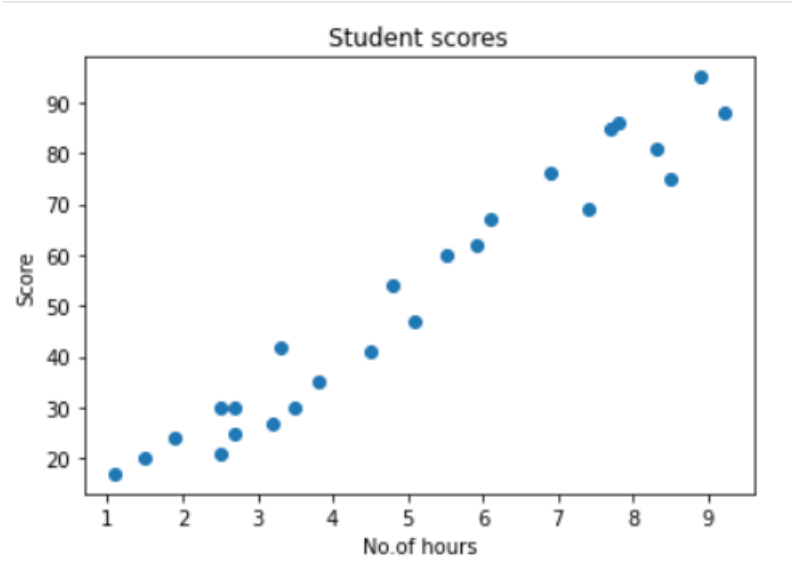
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25 entries, 0 to 24
```

```
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
```

INPUT

```
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()
```

OUTPUT



#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

INPUT

```
X = student.iloc[:, :-1]
y = student.iloc[:, 1]
print(X)
```

OUTPUT

```
    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
...
24    7.8
```

INPUT

```
print(y)
```

OUTPUT

```
0     21
1     47
2     27
3     75
4     30
5     20
6     88
7     60
8     81
9     25
...
23    76
24    86
Name: Scores, dtype: int64
```

INPUT

```
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)
```

OUTPUT

```
    Hours
24    7.8
5     1.5
21    4.8
15    8.9
```

```
20    2.7
9     2.7
10    7.7
6     9.2
12    4.5
1     5.1
14    1.1
3     8.5
0     2.5
13    3.3
8     8.3
4     3.5
23    6.9
17    1.9
7     5.5
16    2.5
```

INPUT

```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

OUTPUT

```
LinearRegression()
```

INPUT

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

OUTPUT

```
3.3679146249897656
```

INPUT

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

OUTPUT

```
[9.70315174]
```

INPUT

```
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())
```

OUTPUT

```
Actual value : 62 Predicted value : 60.61650991569965
Actual value : 27 Predicted value : 34.41800020639174
Actual value : 67 Predicted value : 62.55714026453727
Actual value : 35 Predicted value : 40.239891252904606
Actual value : 69 Predicted value : 75.17123753198183
Number of mislabeled points from test data set : 5
```

INPUT

```
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)))
```

OUTPUT

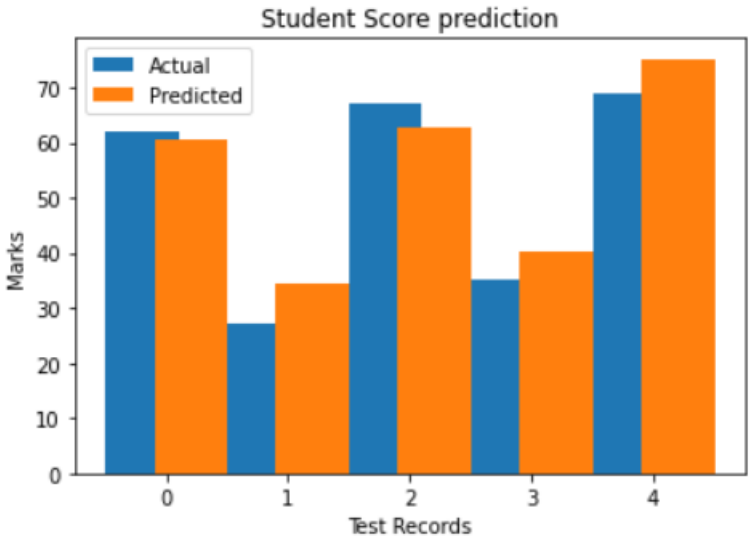
```
Mean Absolute error : 4.931095762208251
Mean Squared error : 28.444081504557726
Root Mean Squared error : 5.333299307610415
```

INPUT

```
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()
```

OUTPUT

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



Multiple Linear Regression

INPUT

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
advertising = pd.read_csv('Company_data.csv')
advertising.head()
```

OUTPUT

	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()
```

OUTPUT

	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()

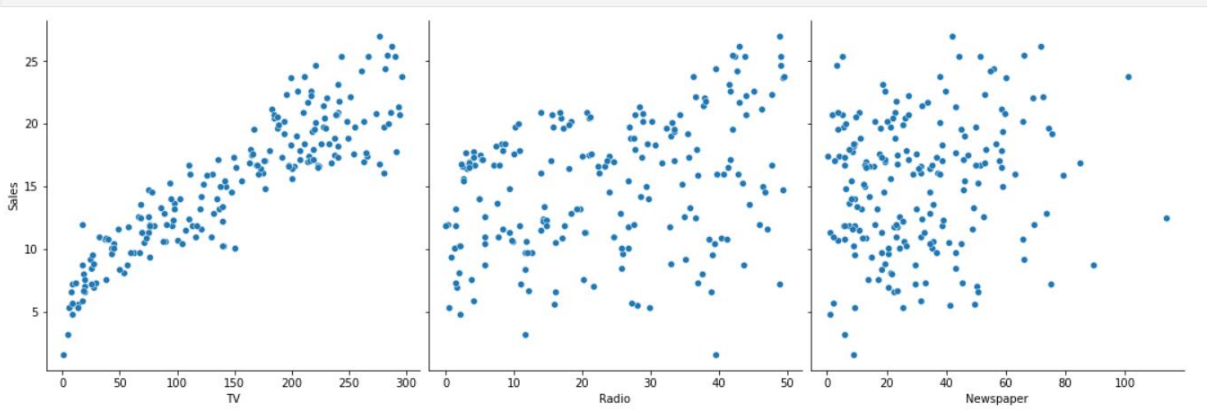
OUTPUT

```
<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
```

INPUT

```
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()
```

OUTPUT



INPUT

```
#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)
```

OUTPUT

```
        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]
```

INPUT

```
y = advertising.iloc[:, -1]
print(y)
```

OUTPUT

```
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
```

INPUT

```
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)
```

OUTPUT

```
        TV  Radio  Newspaper
190   39.5   41.1        5.8
161   85.7   35.8       49.3
37    74.7   49.4       45.7
87   110.7   40.6       63.2
97   184.9   21.0       22.0
..     ...    ...        ...
119   19.4   16.0       22.3
175  276.9   48.9       41.8
126    7.8   38.9       50.6
86    76.3   27.5       16.0
73   129.4    5.7       31.3

[140 rows x 3 columns]
```

INPUT

```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

OUTPUT

```
LinearRegression()
```

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

OUTPUT

```
4.479303977475622
```

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

OUTPUT

```
[0.05389537 0.11490155 0.00179435]
```

INPUT

```
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())
```

OUTPUT

```
Actual value : 11.5 Predicted value : 11.895846240525387
Actual value : 9.7 Predicted value : 9.317581242602184
Actual value : 19.4 Predicted value : 20.167393204252356
Actual value : 10.3 Predicted value : 12.275285815341341
Actual value : 18.2 Predicted value : 18.27397334414116
```

```
Actual value : 12.9 Predicted value : 13.753596405268244
...
Actual value : 7.6 Predicted value : 7.875678071895033
Actual value : 7.6 Predicted value : 6.988004625036124
Actual value : 25.4 Predicted value : 23.90563646791189
Number of mislabeled points from test data set : 60
```

INPUT

```
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)))
```

OUTPUT

```
Mean Absolute error : 1.2379439849720684
Mean Squared error : 3.342870135490751
Root Mean Squared error : 1.8283517537636873
```

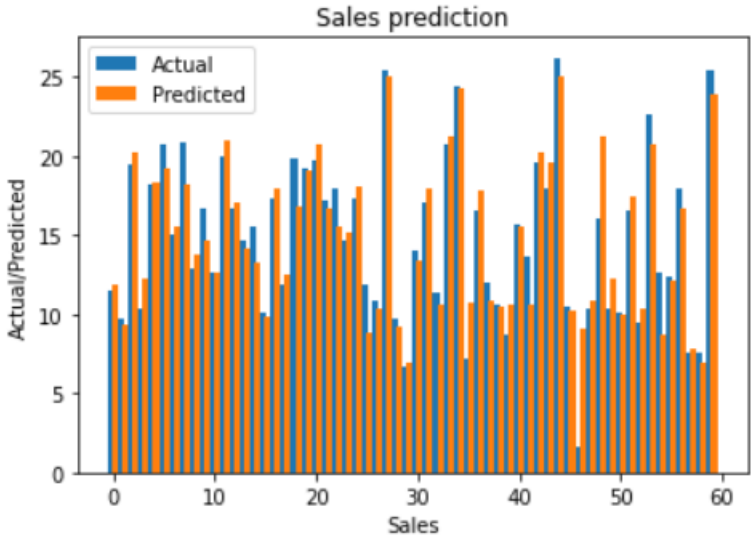
INPUT

```
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()
```

OUTPUT

```
[ 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]
```



Neural Networks

#7. 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.

INPUT

```
import tensorflow as tf
import keras
import pandas
import sklearn
```

```
import matplotlib
```

OUTPUT

```
2022-02-15 15:30:27.597560: W
tensorflow/stream_executor/platform/default/dso_loader.cc:64]
Could not load dynamic library 'libcudart.so.11.0'; dlerror:
libcudart.so.11.0: cannot open shared object file: No such file or
directory

2022-02-15 15:30:27.597631: I
tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above
cudart dlerror if you do not have a GPU set up on your machine.
```

INPUT

```
import pandas as pd

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

df
```

OUTPUT

	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

INPUT

```
dataset = df.values
dataset
```

OUTPUT

```
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]])
```

INPUT

```
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
```

OUTPUT

```
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]])
```

INPUT

```
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)
```

OUTPUT

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

INPUT

```
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))
```

OUTPUT

```
2022-02-15 15:32:11.492039: W
tensorflow/stream_executor/platform/default/dso_loader.cc:64]
Could not load dynamic library 'libcuda.so.1'; dlerror:
libcuda.so.1: cannot open shared object file: No such file or
directory

2022-02-15 15:32:11.492109: W
tensorflow/stream_executor/cuda/cuda_driver.cc:269] failed call to
cuInit: UNKNOWN ERROR (303)

2022-02-15 15:32:11.492153: I
tensorflow/stream_executor/cuda/cuda_diagnostics.cc:156] kernel
driver does not appear to be running on this host (Z238-UL):
/proc/driver/nvidia/version does not exist

2022-02-15 15:32:11.518227: I
tensorflow/core/platform/cpu_feature_guard.cc:151] This TensorFlow
binary is optimized with oneAPI Deep Neural Network Library
(oneDNN) to use the following CPU instructions in performance-
critical operations:  AVX2 FMA

To enable them in other operations, rebuild TensorFlow with the
appropriate compiler flags.

Epoch 1/100

32/32 [==============================] - 1s 5ms/step - loss:
0.6723 - accuracy: 0.5059 - val_loss: 0.6751 - val_accuracy:
0.4886
```

```
Epoch 2/100

32/32 [==============================] - 0s 2ms/step - loss:
0.6648 - accuracy: 0.5039 - val_loss: 0.6690 - val_accuracy:
0.4886

Epoch 3/100

32/32 [==============================] - 0s 2ms/step - loss:
0.6578 - accuracy: 0.5098 - val_loss: 0.6634 - val_accuracy:
0.5662

...

Epoch 100/100

32/32 [==============================] - 0s 2ms/step - loss:
0.2789 - accuracy: 0.8885 - val_loss: 0.3786 - val_accuracy:
0.8584
```

INPUT

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

OUTPUT

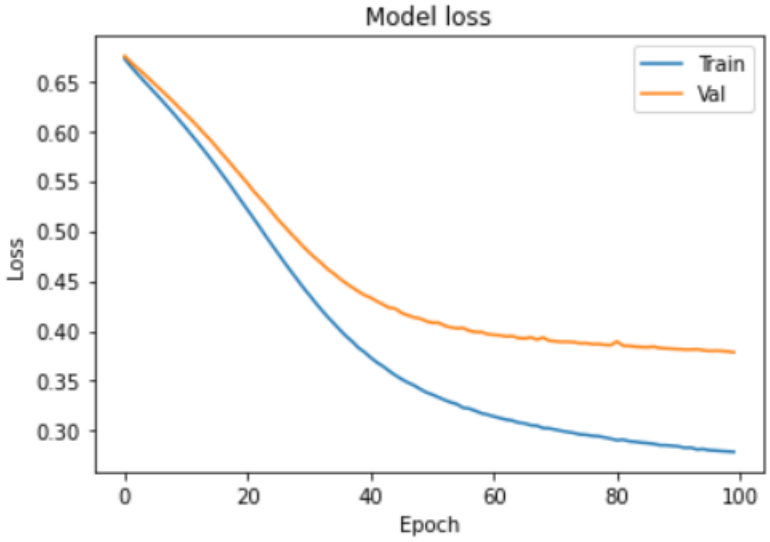
```
7/7 [==============================] - 0s 1ms/step - loss: 0.2081
- accuracy: 0.9224


0.922374427318573
```

INPUT

```
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()
```

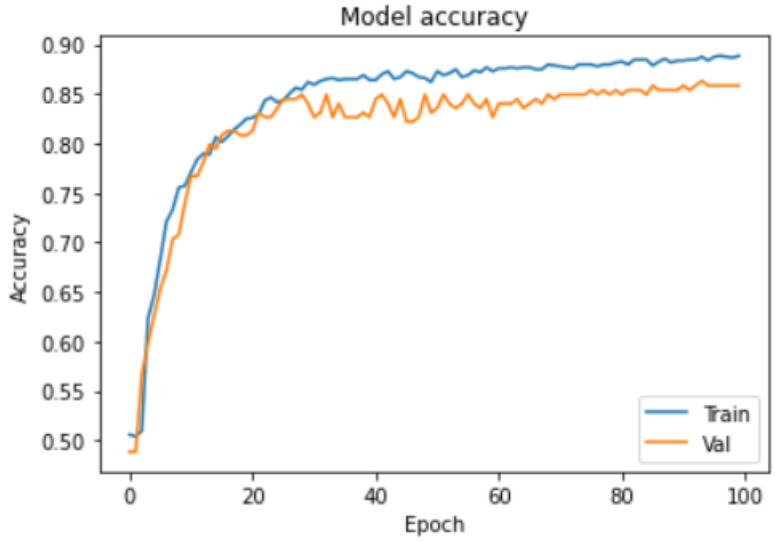
OUTPUT



INPUT

```
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()
```

OUTPUT



INPUT

```
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))
```

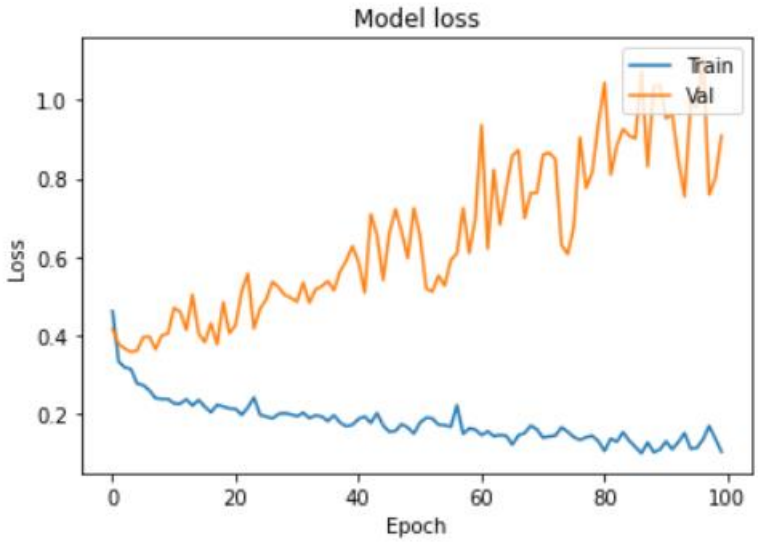
OUTPUT

```
Epoch 1/100

32/32 [==============================] - 1s 23ms/step - loss:
0.4627 - accuracy: 0.7759 - val_loss: 0.4173 - val_accuracy:
0.8311

Epoch 2/100

32/32 [==============================] - 1s 17ms/step - loss:
0.3340 - accuracy: 0.8552 - val_loss: 0.3791 - val_accuracy:
0.8858

Epoch 3/100

32/32 [==============================] - 1s 16ms/step - loss:
0.3195 - accuracy: 0.8679 - val_loss: 0.3670 - val_accuracy:
0.8767

...

Epoch 100/100

32/32 [==============================] - 0s 15ms/step - loss:
0.1051 - accuracy: 0.9589 - val_loss: 0.9088 - val_accuracy:
0.8904
```

INPUT

```
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()
```

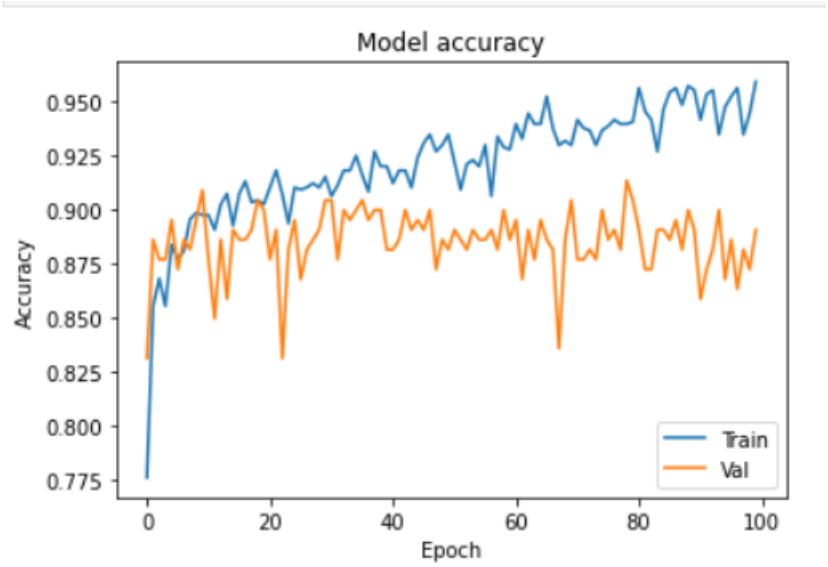
OUTPUT



INPUT

```
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()
```

OUTPUT



INPUT

```
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))
```

OUTPUT

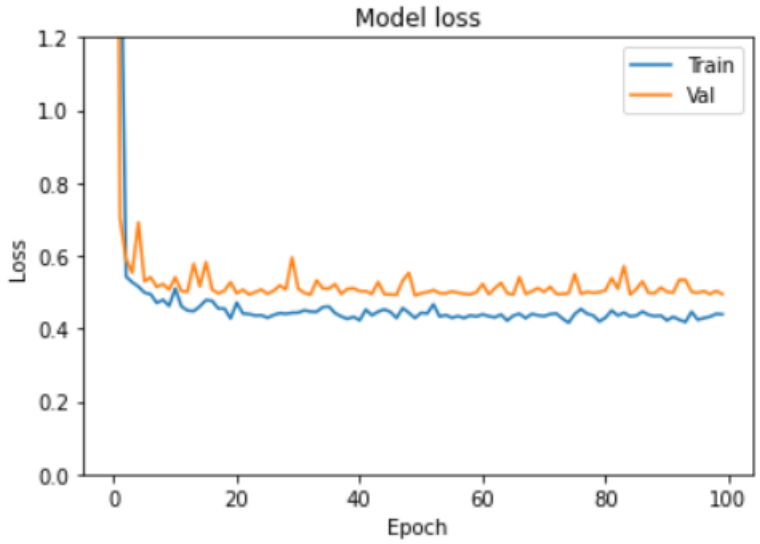
```
Epoch 1/100
32/32 [==============================] - 1s 28ms/step - loss:
14.2275 - accuracy: 0.5930 - val_loss: 3.9129 - val_accuracy:
0.7260
Epoch 2/100
32/32 [==============================] - 1s 24ms/step - loss:
1.6881 - accuracy: 0.8102 - val_loss: 0.7029 - val_accuracy:
0.8584
Epoch 3/100
32/32 [==============================] - 1s 24ms/step - loss:
0.5424 - accuracy: 0.8640 - val_loss: 0.5901 - val_accuracy:
0.8402
...
Epoch 100/100
32/32 [==============================] - 1s 27ms/step - loss:
0.4391 - accuracy: 0.8777 - val_loss: 0.4947 - val_accuracy:
0.8630
```

INPUT

```
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()
```

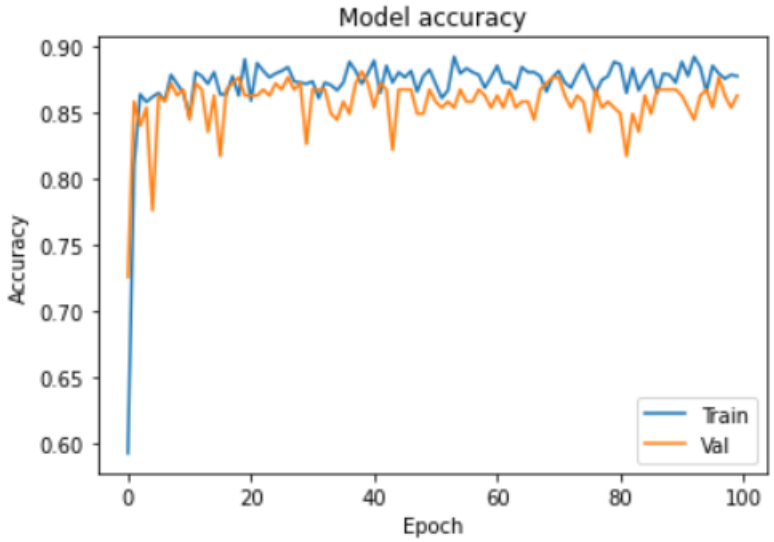
OUTPUT



INPUT

```
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()
```

OUTPUT


Model accuracy
Accuracy
Epoch
Train
Val
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0
20
40
60
80
100

LAB CYCLE-5

SVM Classification

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.

INPUT

```
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))
```

OUTPUT

```
              precision    recall  f1-score   support

           0       0.75      0.79      0.77       159
           1       0.53      0.52      0.52       147
           2       0.56      0.55      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       0.70      0.70      0.70       600


/home/sjcet/anaconda3/lib/python3.9/site-
packages/sklearn/svm/_base.py:985: ConvergenceWarning: Liblinear
failed to converge, increase the number of iterations.
```

```
  warnings.warn("Liblinear failed to converge, increase "
```

INPUT

```
print(confusion_matrix(y_test, y_pred))
```

OUTPUT

```
[[126  18  14   1]
 [ 20  76  49   2]
 [ 19  48  84   3]
 [  3   1   3 133]]
```

K-means

#2. 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.

INPUT

```
import pandas as pd
customers = pd.read_csv('customer_data.csv')
customers.head()
```

OUTPUT

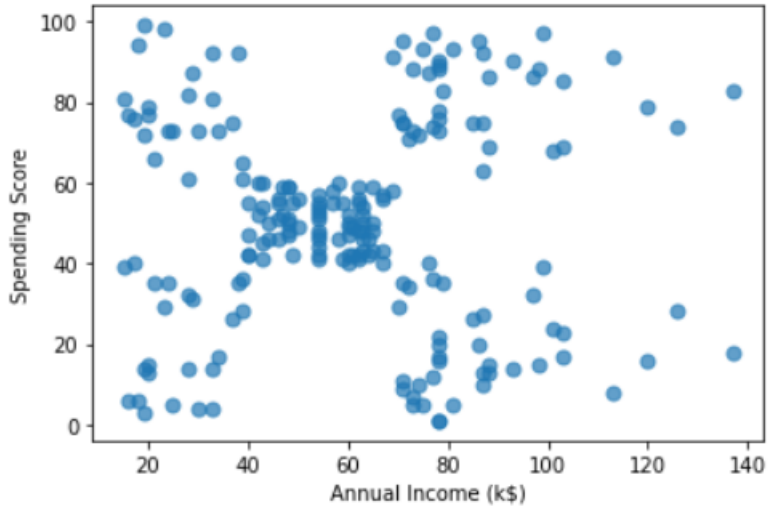
	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

INPUT

```
import matplotlib.pyplot as plt
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')
```

OUTPUT

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



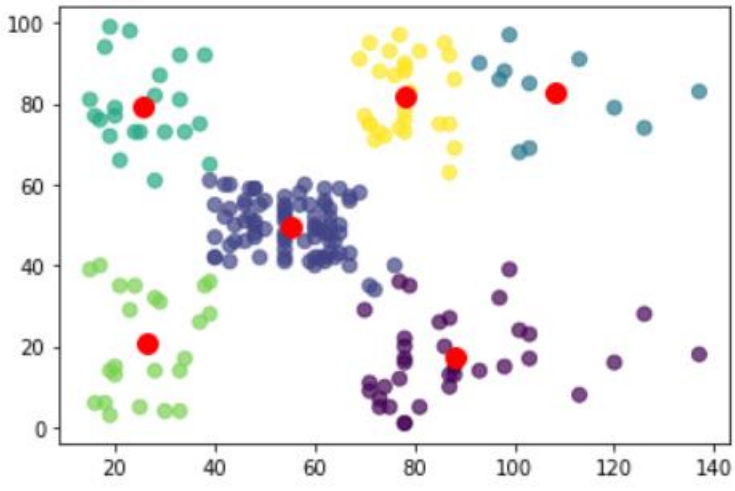
#Q. Using k means clustering create 6 clusters of customers based on their spending pattern.Visualize the same in a scatter plot with each cluster in a different color scheme.

INPUT

```
from sklearn.cluster import KMeans


kmeans = KMeans(n_clusters=6, 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')
centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='red', s=100)
```

OUTPUT

```
<matplotlib.collections.PathCollection at 0x7f7ccf935ee0>
```



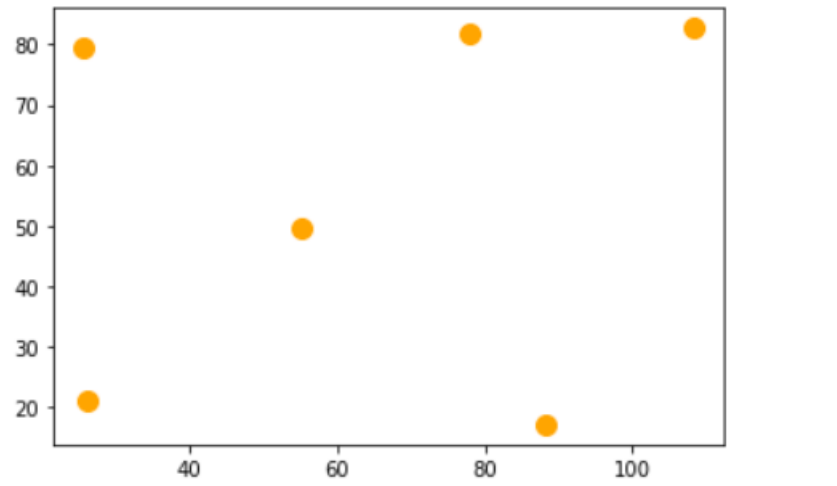
#Display the cluster centers.

INPUT

```
centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='orange', s=100)
```

OUTPUT

```
<matplotlib.collections.PathCollection at 0x7f7ccf809190>
```



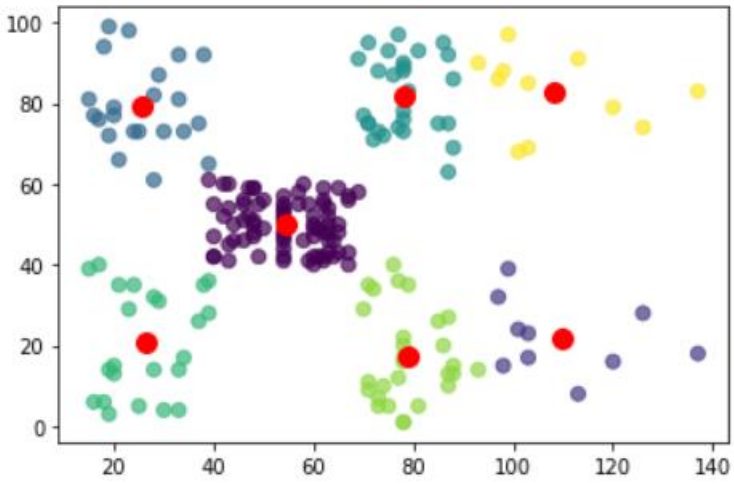
#Use different values of K and visualize the same using scatter plot.

INPUT

```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=7, 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')
centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='red', s=100)
```

OUTPUT

```
<matplotlib.collections.PathCollection at 0x7f7ccf7ea910>
```



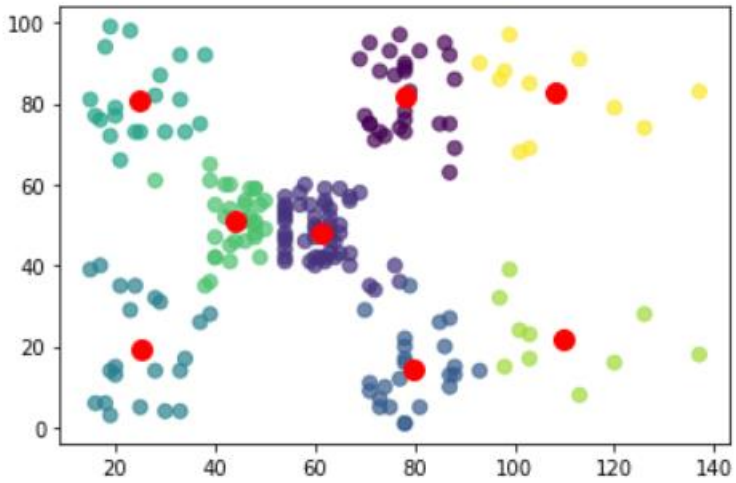
#Use different values of K and visualize the same using scatter plot.

INPUT

```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=8, 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')
centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='red', s=100)
```

OUTPUT

```
<matplotlib.collections.PathCollection at 0x7f7ccf322790>
```



#Display the cluster labels of each point.(print cluster indexes)

INPUT

print(predicted_cluster_indexes)

OUTPUT

```
[4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4
 2 4 2 4 2 4 0 4 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 3 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5 3
 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5 6 1 6 1 6 1
 6 1 6 1 6 1 6 1 6 1 6 1 6 1 6]
```

NLP

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

INPUT

```
from nltk.corpus import stopwords
import nltk

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))
```

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

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

OUTPUT

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

INPUT

```
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"]
```

INPUT

```
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']
```

```
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')]
```

INPUT

```
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']
```

INPUT

```
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']
```