FEDERAL INSTITUTE OF SCIENCE AND TECHNOLOGY (FISAT)™

HORMIS NAGAR, MOOKKANNOOR, ANGAMALY-683577

FOCUS ON EXCELLENCE

20MCA241 DATA SCIENCE LAB LABORATORY RECORD

Name: MANEESHA M V

Branch: MASTER OF COMPUTER APPLICATIONS

Semester: 3 Batch: B Roll No: 23

University Exam Reg. No.: FIT22MCA-2082

DECEMBER 2023

FEDERAL INSTITUTE OF SCIENCE AND TECHNOLOGY (FISAT)™

HORMIS NAGAR, MOOKKANNOOR, ANGAMALY-683577

FOCUS ON EXCELLENCE

CERTIFICATE

This is to certify that this is a Bonafide record of the Practical work done by **MANEESHA M V (FIT22MCA-2082)** *in the* **20MCA241 DATA SCIENCE LAB** *Laboratory towards the partial fulfilment for the award of the Master Of Computer Applications during the academic year 2023-2024.*

Signature of Staff in Charge

Ms. Joice T

Signature of HOD

Dr. Deepa Mary Mathews

Date of University practical examination ……

Signature of
Internal Examiner

Signature of
External Examiner

CONTENTS

16	07/11/2023	Linear Regression using cars1 dataset		
17	07/11/2023	Regression using iris dataset		
18	07/11/2023	Prediction using a Combination of Classifiers on Hepatitis dataset		
19	14/11/2023	Classification using SVM for iris dataset		
20	14/11/2023	Classification using SVM for fruit dataset		
21	21/11/2023	K means clustering and plot graph		
22	21/11/2023	K means clustering on iris dataset		
23	24/11/2023	K means clustering on Array values		
24	24/11/2023	Chance to quit company using MLP Classifier		
25	05/11/2023	Gate implementation using Perceptron		
26	05/11/2023	Image Classification using CNN on Cifar10 dataset		
27	05/11/2023	Sentiment identification using MLP		

EXPERIMENT NO :1

AIM: Python Basics

CODE

```
a=5
print(a)
print(type(a))
```

OUTPUT

```
5
<class 'int'>
```

CODE

```
f=1.5
print(f)
print(type(f))
```

OUTPUT

```
1.5
<class 'float'>
```

CODE

```
s="hello"
print(s)
print(type(s))
```

OUTPUT

```
hello
<class 'str'>
```

CODE

```
b=True
print(b)
print(type(b))
```

OUTPUT

```
True
<class 'bool'>
```

CODE

```
t=5+3j
print(t)
```

```
print(type(t))
```

OUTPUT

```
(5+3j)
<class 'complex'>
```

CODE

```
a = 7
b = 3
ab_sum = a + b
print(ab_sum)
```

OUTPUT

```
10
```

CODE

```
ab_dif = a - b
print(ab_dif)
```

OUTPUT

```
4
```

CODE

```
a = 7
b = 3
ab_sum = a + b
print(ab_sum)
```

OUTPUT

```
21
```

CODE

```
ab_quo = a / b
print(ab_quo)
```

OUTPUT

```
2.3333333333333335
```

CODE

```
ab_iquo = a//b
print(ab_iquo)
```

OUTPUT

2

CODE

```
ab_rem = a % b
print(ab_rem)
```

OUTPUT

1

CODE

```
ab_pow = a**b
print(ab_rem)
```

OUTPUT

1

CODE

```
T=True
F=False
print(T,F)
```

OUTPUT

True False

CODE

```
p = 5 > 3
print(p)
```

OUTPUT

True

CODE

```
q = -1 < -12.5
print(q)
```

OUTPUT

False

CODE

```
print(p and q)
print(p or q)
print(not q)
```

OUTPUT

```
False
True
True
```

CODE

```
s ='hello'
u="hello"
print(s)
print(u)
```

OUTPUT

```
hello
hello
```

CODE

```
s1 = "python"
s2 = ' w o r l d '
s3 = s1 + ' ' + s2
print(s3)
print(len(s3))
s3 = '%s %s %d' %(s1, s2, 1011)
print(s3)
```

OUTPUT

```
python  w o r l d
18
python  w o r l d  1011
```

CODE

```
print(s3.upper())
print(s3.capitalize())
print(s3.lower())
print('hello world how are you'.split(' '))
print('book'.replace('o','e'))
word='jewellery'
print(word.find('well'))
print(word.find('is'))
```

OUTPUT

```
PYTHON  W O R L D  1011
Python  w o r l d  1011
python  w o r l d  1011
['hello', 'world', 'how', 'are', 'you']
beek
2
-1
```

CODE

```
number=int(input('Enter the number'))
if number > 99 and number < 1000:
  print('3 digit')
else:
  print('Not 3 digit')
```

OUTPUT

```
Enter the number23
Not 3 digit
```

CODE

```
response = input('Are you f a m i l i a r with python : ' )
if response.upper() == "YES":
  print("You can skip t h i s course : - |" )
elif response.upper() == "NO":
  print("You are at the r i g h t place : - ) " )
else:
  print('Sorry wrong input : - ( ' )
```

OUTPUT

```
Are you f a m i l i a r with python : no
You are at the r i g h t place : - )
```

CODE

```
for x in range(10):
  print(x,end=' ' )
limit = int(input(' Enter a limit : ' ) )
sum = 0
for i in range(1,limit + 1 ) :
  if i%2 != 0:
    sum += i
print("Odd sum = ",sum)
```

OUTPUT

```
0 1 2 3 4 5 6 7 8 9  Enter a limit : 15
Odd sum =  64
```

CODE

```
number=int(input('Enter number:'))
s=0
while number>0:
  s+=number%10
  number=number//10
print(s)
```

OUTPUT

```
Enter number:1254
12
```

CODE

```
limit = int(input('Enter number :'))
for num in range(2,limit+1):
  is_divisible = False
  k=2
  while k <= num//2 :
    if num % k == 0:
      is_divisible =True
    break;
  k+=1
  if not is_divisible:
    print(num,end=' ' )
```

OUTPUT

```
Enter number :400
2 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59 61 63 65 67
69 71 73 75 77 79 81 83 85 87 89 91 93 95 97 99 101 103 105 107 109 111 113 115 117 119 121
123 125 127 129 131 133 135 137 139 141 143 145 147 149 151 153 155 157 159 161 163 165 167
169 171 173 175 177 179 181 183 185 187 189 191 193 195 197 199 201 203 205 207 209 211 213
215 217 219 221 223 225 227 229 231 233 235 237 239 241 243 245 247 249 251 253 255 257 259
261 263 265 267 269 271 273 275 277 279 281 283 285 287 289 291 293 295 297 299 301 303 305
307 309 311 313 315 317 319 321 323 325 327 329 331 333 335 337 339 341 343 345 347 349 351
353 355 357 359 361 363 365 367 369 371 373 375 377 379 381 383 385 387 389 391 393 395 397
399
```

CODE

```
mylist=['a','b',1, 1.2, True]
print(mylist)
```

```
mylist.append('new')
print(mylist)
print(mylist.pop())
mylist.insert(2,'new')
print(mylist)
mylist.remove('new')
print(mylist)
```

OUTPUT

```
['a', 'b', 1, 1.2, True]
['a', 'b', 1, 1.2, True, 'new']
new
['a', 'b', 'new', 1, 1.2, True]
['a', 'b', 1, 1.2, True]
```

CODE

```
b=[1,2,3]
mylist.append(b)
print(mylist)
mylist.remove(b)
print(mylist)
mylist.extend(b)
print(mylist)
a=[2,3,1,4,5]
a.sort()
print(a)
print(list('hello'))
```

OUTPUT

```
['a', 'b', 1, 1.2, True, [1, 2, 3]]
['a', 'b', 1, 1.2, True]
['a', 'b', 1, 1.2, True, 1, 2, 3]
[1, 2, 3, 4, 5]
['h', 'e', 'l', 'l', 'o']
```

CODE

```
numbers=[1,2,3,4,5,6,7,8,9,10]
print(numbers[1],numbers[-1])
sliced = numbers[5:10]
print( sliced)
sliced = numbers[5:]
print( sliced)
sliced = numbers[:7]
print( sliced)
```

```
sliced = numbers[-2:]
print(sliced)
```

OUTPUT

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

CODE

```
numbers=list(range(1,8))
print(numbers)
square=[]
for i in numbers:
  square.append(pow(i,2))
print(square)
```

OUTPUT

```
[1, 2, 3, 4, 5, 6, 7]
[1, 4, 9, 16, 25, 36, 49]
```

CODE

```
square = [x**2 for x in numbers]
print(square)
odd_square =[x**2 for x in numbers if x%2 != 0]
print(odd_square)
```

OUTPUT

```
[1, 4, 9, 16, 25, 36, 49]
[1, 9, 25, 49]
```

CODE

```
A=[4,6,8,9]
AxA=[(a,b) for a in A for b in A if a!=b]
print(AxA)
```

OUTPUT

```
[(4, 6), (4, 8), (4, 9), (6, 4), (6, 8), (6, 9), (8, 4), (8, 6), (8, 9), (9, 4), (9, 6), (9, 8)]
```

CODE

```
person = {'name' : 'Manu', 'age': 28}
```

```
print(person['name'])
print('name' in person)
print('sex' in person)
person['sex'] = 'male'
print(person)
for item in person:
  print(item,person[item])
```

OUTPUT

```
Manu
True
False
{'name': 'Manu', 'age': 28, 'sex': 'male'}
name Manu
age 28
sex male
```

CODE

```
t1 = (1, 2, 3)
t2 = (4,5,6)
print(t1,t2)
t3=t1+t2
print(t3)
lt=tuple(['a','b','c','d'])
print(lt)
```

OUTPUT

```
(1, 2, 3) (4, 5, 6)
(1, 2, 3, 4, 5, 6)
('a', 'b', 'c', 'd')
```

CODE

```
s={1,2,3}
print(s,type(s))
```

OUTPUT

```
{1, 2, 3} <class 'set'>
```

CODE

```
fset={"apple", "banana", "cherry"}
fset.remove("banana")
print(fset)
```

OUTPUT

{'cherry', 'apple'}

CODE

```
fset={"apple", "banana", "cherry"}
fset.discard("banana")
print(fset)
```

OUTPUT

{'cherry', 'apple'}

CODE

```
fset={"apple", "banana", "cherry"}
fset.clear()
print(fset)
```

OUTPUT

set()

CODE

```
set1 = {" a ", "b" , "c"}
set2 = { 1 , 2, 3}
set3 = set1.union(set2)
print(set3)
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}
z=x.intersection(y)
print(z)
lst=[1, 2, 3,4,5,5,5, 7,6]
myset=list(set(lst))
print(myset)
```

OUTPUT

{1, 2, 3, 'b', ' a ', 'c'}
{'apple'}
[1, 2, 3, 4, 5, 6, 7]

CODE

```
def twice(number):
  return
```

```
  2*number
t = twice(5)
print(t)

def isPrime(number):
  for factor in range(2, (number//2)+1):
    if number%factor == 0:
      return False
    return True
number = int(input(' E n t e r the number ' ) )
print(isPrime(number))
```

OUTPUT

```
None
 E n t e r the number 85
True
```

CODE

```
def printPrimes(llimit,ulimit):
  for num in range(llimit,ulimit+1):
    if isPrime(num) == True:
      print(num,end = ' ' )

printPrimes(5,50)
# 5 7 11 13 17 19 23 29 31 37 41 43 47
def swap(x,y):
  t = x
  x = y
  y = t
  return x, y
a=5
b=7
a,b = swap(a,b)
print(a,b)
```

OUTPUT

```
5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 7 5
```

CODE

```
def calculatePayable(p, y = 1, r = 5 ):
  return p*(1 + r * y/100)

print(calculatePayable(1000))
```

```
print(calculatePayable(1000, y =3))

print(calculatePayable(1000, r = 10, y=3))

print(calculatePayable(5000, r = 3))
```

OUTPUT

```
1050.0
1150.0
1300.0
5150.0
```

CODE

```
class Adder:
  def init (self) :
    self.x=0
    self.y=0
  def setValues(self,x,y) :
    self.x=x
    self.y=y
  def calculate(self1):
    self1.sum=self1.x+self1.y
  def getSum(self):
    return self.sum

  adder = Adder()
  adder.setValues(5,4)
  adder.calculate()
  print(adder.getSum())
```

OUTPUT

```
9
```

EXPERIMENT NO :2

AIM: Python Basics using numpy operation

CODE

```
x = np.array([1,2,3,4]) #creating numpy array from list
print("X: ",x)
print("Type of X: ",type(x))
print("Shape of X :",x.shape) #printing the dimensoin of the array
```

OUTPUT

```
X:  [1 2 3 4]
Type of X:  <class 'numpy.ndarray'>
Shape of X : (4,)
```

CODE

```
y = np.array([[1,2],[3,4]])
print("Y: ",y)
print("Shape of Y: ",y.shape)
```

OUTPUT

```
Y:  [[1 2]
 [3 4]]
Shape of Y:  (2, 2)
```

CODE

```
z = np.array([[1+0.j,2+5.j]])  #printing the dimension of the numpy array
print("Z ",z)
print("Shape of Z ",z.shape)
```

OUTPUT

```
[[1.+0.j 2.+5.j]]
(1, 2)
```

CODE

```
a = np.zeros((2,3)) #creating numpy array of zeros , of dimension (2,3)
print(a)
print(a.shape)
```

OUTPUT

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

```
 [0. 0. 0.]]
(2, 3)
```

CODE

```
b = np.ones((2,3), dtype=int) #creating numpy array of ones , of dimension (2,3) and integer type
print(b)
```

OUTPUT

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

CODE

```
d = np.eye(3) ## identity matrix
print(d)
```

OUTPUT

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

CODE

```
e = np.arange(10) ##similar to Range function
print("e: ",e)
e1 = np.arange(12, 21)
print("e1: ",e1)
e2= np.arange(5,20,3)
print("e2: ",e2)
```

OUTPUT

```
e:  [0 1 2 3 4 5 6 7 8 9]
e1:  [12 13 14 15 16 17 18 19 20]
e2:  [ 5  8 11 14 17]
```

CODE

```
g = np.random.random((3,4))#creating a numpy array having dimension (3,4) with random elements
print(g)
print("\nShape of g: ",g.reshape(2,2,3)) #reshaping the (3,4) array to (2,2,3)
```

OUTPUT

```
[[0.30531394 0.82028638 0.38373809 0.04610537]
 [0.50159541 0.96532078 0.93011033 0.81773299]
```

```
 [0.97444136 0.60486075 0.76807379 0.17648357]]

Shape of g:  [[[0.30531394 0.82028638 0.38373809]
  [0.04610537 0.50159541 0.96532078]]

 [[0.93011033 0.81773299 0.97444136]
  [0.60486075 0.76807379 0.17648357]]]
```

CODE

```
x = np.arange(12)
print("X:",x)
print("x[4]:",x[4])
print("x[-1]",x[-1])
x.resize(3,4)
print('X:',x)
print("x[-1,-1]:",x[-1,-1])
print("x[2][3]:",x[2][3])
```

OUTPUT

```
X: [ 0  1  2  3  4  5  6  7  8  9 10 11]
x[4]: 4
x[-1] 11
X: [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
x[-1,-1]: 11
x[2][3]: 11
```

CODE

```
y = np.arange(1,26)
print("Y:",y)
print("y[:3]=",y[:3])
print("y[10:]=",y[10:])
print("y[10:15]=",y[10:15])
print("y[-5:]=",y[-5:])
print("y[3:-3]=",y[3:-3])
print("y[::3]=",y[::3])
y = y.reshape((5,5))
print("Y=\n ",y)
print('y[:5,:5]=\n',y[:5,:5])
print("y[2:-1,1:-1]=\n",y[2:-1,1:-1])
print("y[:,:-1]=\n",y[:,:-1])
print("y[:,-1]=\n",y[:,-1])
print("y[::,::2]=\n",y[::,::2])
print("y[1::2,1::2]:=\n",y[1::2,1::2])
print(y[::,::2])
```

OUTPUT

```
Y: [ 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]
y[:3]= [1 2 3]
y[10:]= [11 12 13 14 15 16 17 18 19 20 21 22 23 24 25]
y[10:15]= [11 12 13 14 15]
y[-5:]= [21 22 23 24 25]
y[3:-3]= [ 4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22]
y[::3]= [ 1  4  7 10 13 16 19 22 25]
Y=
  [[ 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]]
y[:5,:5]=
 [[ 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]]
y[2:-1,1:-1]=
 [[12 13 14]
 [17 18 19]]
y[:,:-1]=
 [[ 1  2  3  4]
 [ 6  7  8  9]
 [11 12 13 14]
 [16 17 18 19]
 [21 22 23 24]]
y[:,-1]=
 [ 5 10 15 20 25]
y[::,::2]=
 [[ 1  3  5]
 [ 6  8 10]
 [11 13 15]
 [16 18 20]
 [21 23 25]]
y[1::2,1::2]:=
 [[ 7  9]
 [17 19]]

[[ 1  3  5]
 [ 6  8 10]
 [11 13 15]
 [16 18 20]
 [21 23 25]]
```

CODE

```
a = np.arange(1,6)
b = np.arange(6,11)
print("a+b= ",a+b)
print('a-b= ',a-b)
print('b-a= ',b-a)
print('a**2= ',a**2)
print('a>3= ',a>3)
```

OUTPUT

```
a+b=  [ 7  9 11 13 15]
a-b=  [-5 -5 -5 -5 -5]
b-a=  [5 5 5 5 5]
a**2=  [ 1  4  9 16 25]
a>3=  [False False False  True  True]
```

CODE

```
a = np.arange(0,4).reshape((2,2))
b = np.eye(2)
print("a*b= ",a*b) ##Wrong
print("\n",np.dot(a,b)) ##Matrix multiplication
```

OUTPUT

```
a*b=  [[0. 0.]
 [0. 3.]]

 [[0. 1.]
 [2. 3.]]
```

CODE

```
x = np.arange(1,10).reshape(3,3)
print("X = ",x)
print("sum =",x.sum())
print("sum(axis=0)\n",x.sum(axis=0))
print("sum(axis=1)\n",x.sum(axis=1))
```

OUTPUT

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

```
sum = 45
sum(axis=0)
 [12 15 18]
sum(axis=1)
 [ 6 15 24]
```

CODE

```
x = np.arange(1,10).reshape(3,3)
print("Max= ",x.max())
print("max(axis=0)\n",x.max(axis=0))
print("Transpose\n",x.transpose())
```

OUTPUT

```
Max=  9
max(axis=0)
 [7 8 9]
Transpose
 [[1 4 7]
 [2 5 8]
 [3 6 9]]
```

EXPERIMENT NO :3

AIM: Python Basics using pandas

CODE

```
s = pd.Series([1, 3, 5, 6, 8])
print(s)
```

OUTPUT

```
0    1
1    3
2    5
3    6
4    8
dtype: int64
```

CODE

```
dict = {"country": ["Brazil", "Russia", "India", "China", "South Africa"],
       "capital": ["Brasilia", "Moscow", "New Dehli", "Beijing", "Pretoria"],
       "area": [8.516, 17.10, 3.286, 9.597, 1.221],
       "population": [200.4, 143.5, 1252, 1357, 52.98] }
b = pd.DataFrame(dict)
print(b)
b.index = ["BR", "RU", "IN", "CH", "SA"]
print("\n",b)
```

OUTPUT

```
       country    capital    area  population
0       Brazil   Brasilia   8.516      200.40
1       Russia     Moscow  17.100      143.50
2        India  New Dehli   3.286     1252.00
3        China    Beijing   9.597     1357.00
4  South Africa   Pretoria   1.221       52.98


        country    capital    area  population
BR       Brazil   Brasilia   8.516      200.40
RU       Russia          Moscow      17.100      143.50
```

IN	India	New Dehli	3.286	1252.00
CH	China	Beijing	9.597	1357.00
SA	South Africa	Pretoria	1.221	52.98

CODE

```
import pandas as pd
cars = pd.read_csv('cars1.csv')
print(cars)
# Print out first 4 observations
print(cars[0:4])

# Print out fifth and sixth observation
print(cars[4:6])
print(cars.iloc[2])
```

OUTPUT

```
         Car      Model  Volume  Weight  CO2
0     Toyoty       Aygo    1000     790   99
1  Mitsubishi  Space Star    1200    1160   95
2      Skoda     Citigo    1000     929   95
3       Fiat        500     900     865   90
4       Mini     Cooper    1500    1140  105
5         VW        Up!    1000     929  105
6      Skoda      Fabia    1400    1109   90
7   Mercedes     A-Class    1500    1365   92
8       Ford      Fiesta    1500    1112   98
9       Audi          A1    1600    1150   99
10   Hyundai         I20    1100     980   99
11    Suzuki       Swift    1300     990  101
12     Ford      Fiesta    1000    1112   99
13     Honda       Civic    1600    1252   94
14    Hundai         I30    1600    1326   97
15     Opel       Astra    1600    1330   97
16      BMW           1    1600    1365   99
17     Mazda           3    2200    1280  104
18     Skoda      Rapid    1600    1119  104
19      Ford      Focus    2000    1328  105
20      Ford     Mondeo    1600    1584   94
21      Opel    Insignia    2000    1428   99
22   Mercedes     C-Class    2100    1365   99
23     Skoda     Octavia    1600    1415   99
24     Volvo         S60    2000    1415   99
25   Mercedes        CLA    1500    1465  102
26      Audi          A4    2000    1490  104
```

```
27       Audi         A6    2000    1725  114
28      Volvo         V70    1600    1523  109
29        BMW           5    2000    1705  114
30    Mercedes     E-Class    2100    1605  115
31      Volvo        XC70    2000    1746  117
32       Ford      B-Max    1600    1235  104
33        BMW         216    1600    1390  108
34       Opel      Zafira    1600    1405  109
35    Mercedes         SLK    2500    1395  120


       Car       Model  Volume  Weight  CO2
0     Toyoty        Aygo    1000     790   99
1  Mitsubishi  Space Star    1200    1160   95
2      Skoda      Citigo    1000     929   95
3       Fiat         500     900     865   90
   Car   Model  Volume  Weight  CO2
4  Mini  Cooper    1500    1140  105
5    VW     Up!    1000     929  105


Car        Skoda
Model      Citigo
Volume      1000
Weight       929
CO2           95
Name: 2, dtype: object
```

CODE

```
#Slicing dataframe
df = pd.DataFrame([['Jay','M',18],['Jennifer','F',17],
                   ['Preity','F',19],['Neil','M',17]],
                  columns = ['Name','Gender','Age'])
print("df\n",df)
df1 = df.iloc[2:,:]
print("df1\n",df1)
df2 = df.iloc[:2,:]
print("df2\n",df2)
```

OUTPUT

```
df
       Name Gender  Age
0       Jay      M   18
1  Jennifer      F   17
2    Preity      F   19
3      Neil      M   17
```

df1

```
     Name Gender  Age
2  Preity      F   19
3    Neil      M   17
```

df2

```
       Name Gender  Age
0       Jay      M   18
1  Jennifer      F   17
```

CODE

```
#Create a series with 4 random numbers
s = pd.Series(np.random.randn(4))
print(s)
print ("The actual data series is:")
print( s.values)
print ("\nhead(2)\n",s.head(2))
print("\nhead(3)\n",s.tail(3))
```

OUTPUT

```
0   -0.098269
1    0.775403
2    0.042976
3    0.628384
dtype: float64
The actual data series is:
[-0.09826926  0.77540306  0.0429757   0.6283836 ]
```

head(2)

```
 0   -0.098269
1    0.775403
dtype: float64
```

head(3)

```
 1    0.775403
2    0.042976
3    0.628384
dtype: float64
```

CODE

```
d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack']),
   'Age':pd.Series([25,26,25,23,30,29,23]),
   'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])}

# Create a DataFrame
df = pd.DataFrame(d)
print(df)
print ("The transpose of the data series is:")
print(df.T)
print ("Row axis labels and column axis labels are:")
print (df.axes)
print ("The data types of each column are:")
print (df.dtypes)
print ("Is the object empty?")
print (df.empty)
print ("Our object is:")
print (df)
print ("The dimension of the object is:")
print (df.ndim)
print("The shape of the object is:")
print (df.shape)
print("The size of the object is:")
print (df.size)
print("The values of the object is:")
print (df.values)
print("The checking null values of the object is:")
df.isnull()
print("sum returns the number of missing values:")
df.isnull().sum() #sum returns the number of missing values
```

OUTPUT

```
    Name  Age  Rating
0    Tom   25    4.23
1  James   26    3.24
2  Ricky   25    3.98
3    Vin   23    2.56
4  Steve   30    3.20
5  Smith   29    4.60
6   Jack   23    3.80
The transpose of the data series is:
          0     1      2     3      4      5     6
Name    Tom  James  Ricky   Vin  Steve  Smith  Jack
Age      25    26     25    23     30     29    23
Rating 4.23  3.24   3.98  2.56    3.2    4.6   3.8
Row axis labels and column axis labels are:
[RangeIndex(start=0, stop=7, step=1), Index(['Name', 'Age', 'Rating'], dtype='object')]
```

The data types of each column are:
Name object
Age int64
Rating float64
dtype: object
Is the object empty?
False
Our object is:

	Name	Age	Rating
0	Tom	25	4.23
1	James	26	3.24
2	Ricky	25	3.98
3	Vin	23	2.56
4	Steve	30	3.20
5	Smith	29	4.60
6	Jack	23	3.80

The dimension of the object is:
2
The shape of the object is:
(7, 3)
The size of the object is:
21
The values of the object is:
[['Tom' 25 4.23]
['James' 26 3.24]
['Ricky' 25 3.98]
['Vin' 23 2.56]
['Steve' 30 3.2]
['Smith' 29 4.6]
['Jack' 23 3.8]]
The checking null values of the object is:

	Name	Age	Rating
0	False	False	False
1	False	False	False
2	False	False	False
3	False	False	False
4	False	False	False
5	False	False	False
6	False	False	False

sum returns the number of missing values:
Name 0

```
Age       0
Rating    0
dtype: int64
```

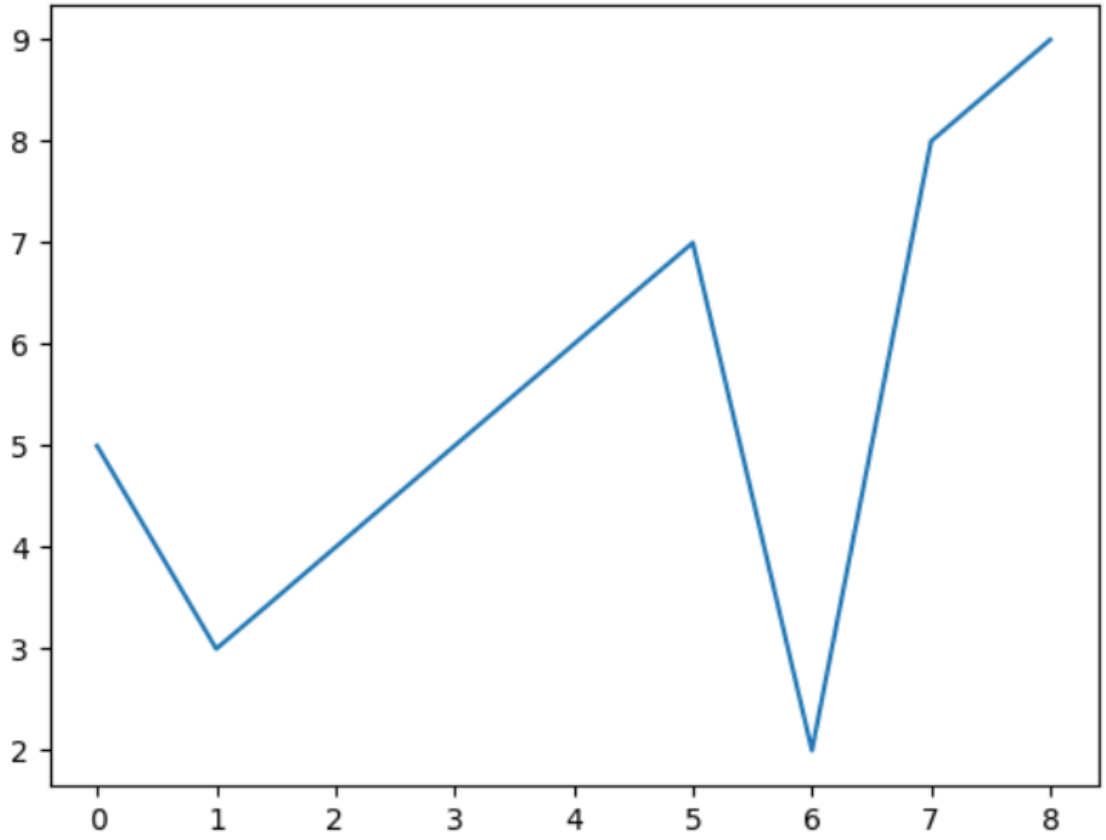
EXPERIMENT NO :4

AIM: Data Visualization

CODE

```
y = [5,3,4,5,6,7,2,8,9]
plt.plot(y)
plt.show()
```

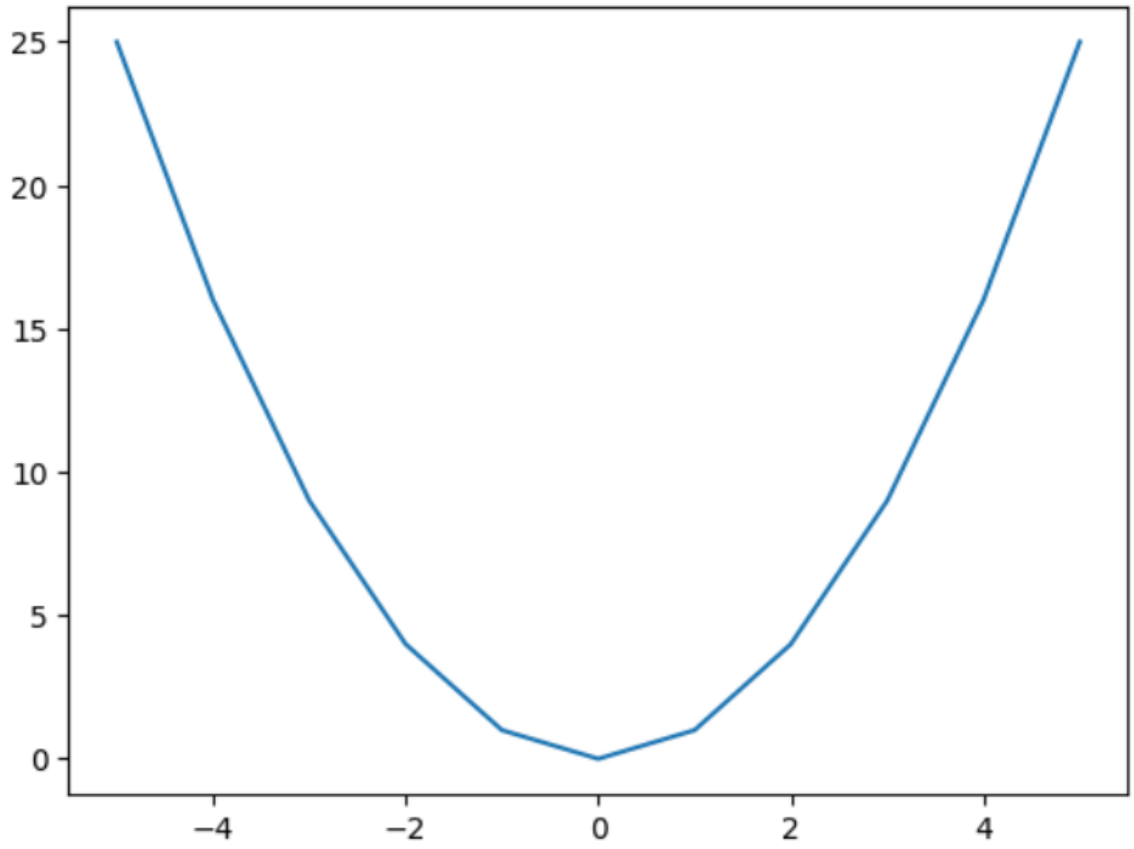
OUTPUT



CODE

```
x = [-5,-4,-3,-2,-1,0,1,2,3,4,5]
#y = [25,16,9,4,1,0,1,4,9,16,25]
y = [i**2 for i in x]
plt.plot(x,y)
plt.show()
```

OUTPUT

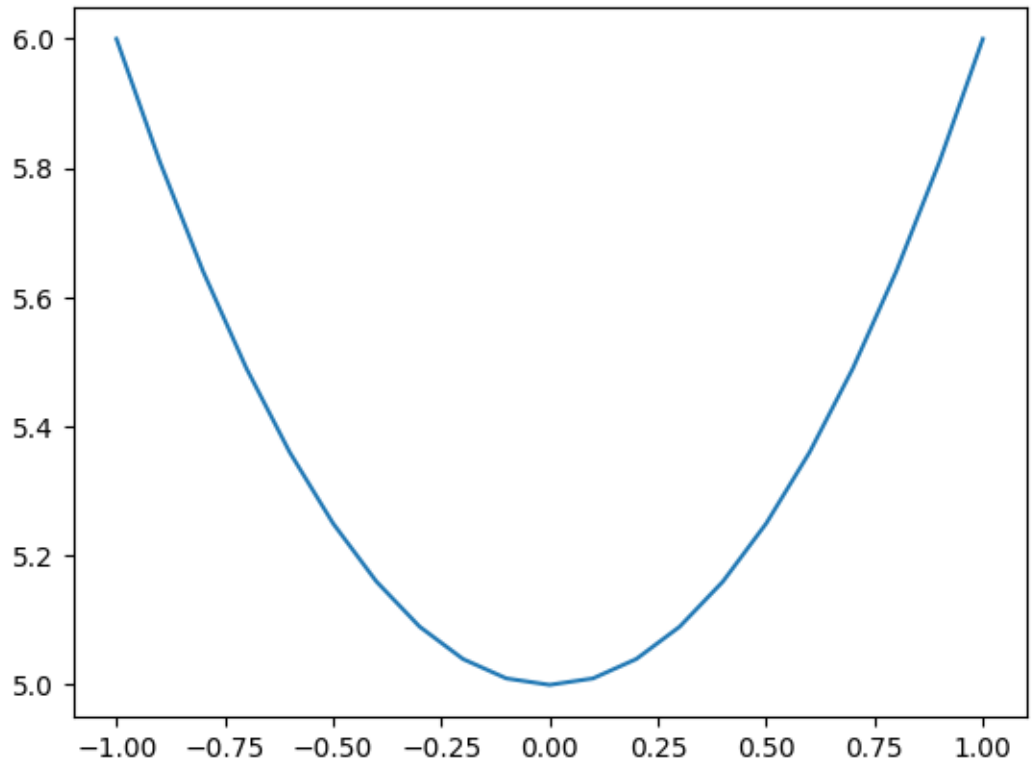


CODE

```
import numpy as np
import math

x = np.arange(-1,1.1,0.1).tolist()
y = [i**2 + 5 for i in x]
print(x)
print(y)

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

OUTPUT

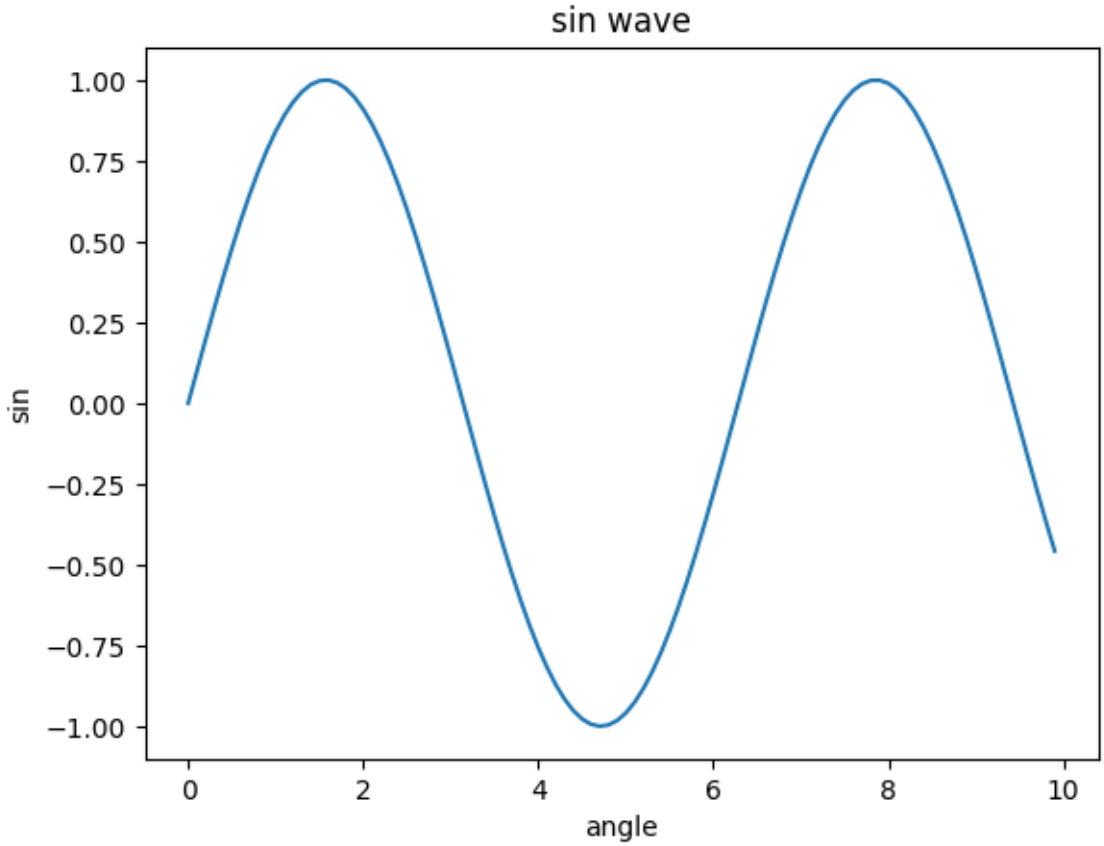
[-1.0, -0.9, -0.8, -0.7000000000000001, -0.6000000000000001, -0.5000000000000001, -0.40000000000000013, -0.30000000000000016, -0.20000000000000018, -0.1000000000000002, -2.220446049250313e-16, 0.09999999999999964, 0.19999999999999973, 0.2999999999999998, 0.3999999999999997, 0.49999999999999956, 0.5999999999999996, 0.6999999999999997, 0.7999999999999996, 0.8999999999999995, 0.9999999999999996]
[6.0, 5.8100000000000005, 5.640000000000001, 5.49, 5.36, 5.25, 5.16, 5.09, 5.04, 5.01, 5.0, 5.01, 5.04, 5.09, 5.16, 5.25, 5.359999999999999, 5.489999999999999, 5.64, 5.809999999999999, 5.999999999999999]



CODE

```
import numpy as np
x = np.arange(0,10,0.1)
y = np.sin(x)
plt.plot(x,y)
plt.xlabel('angle')
plt.ylabel('sin')
plt.title('sin wave')
plt.show()
```

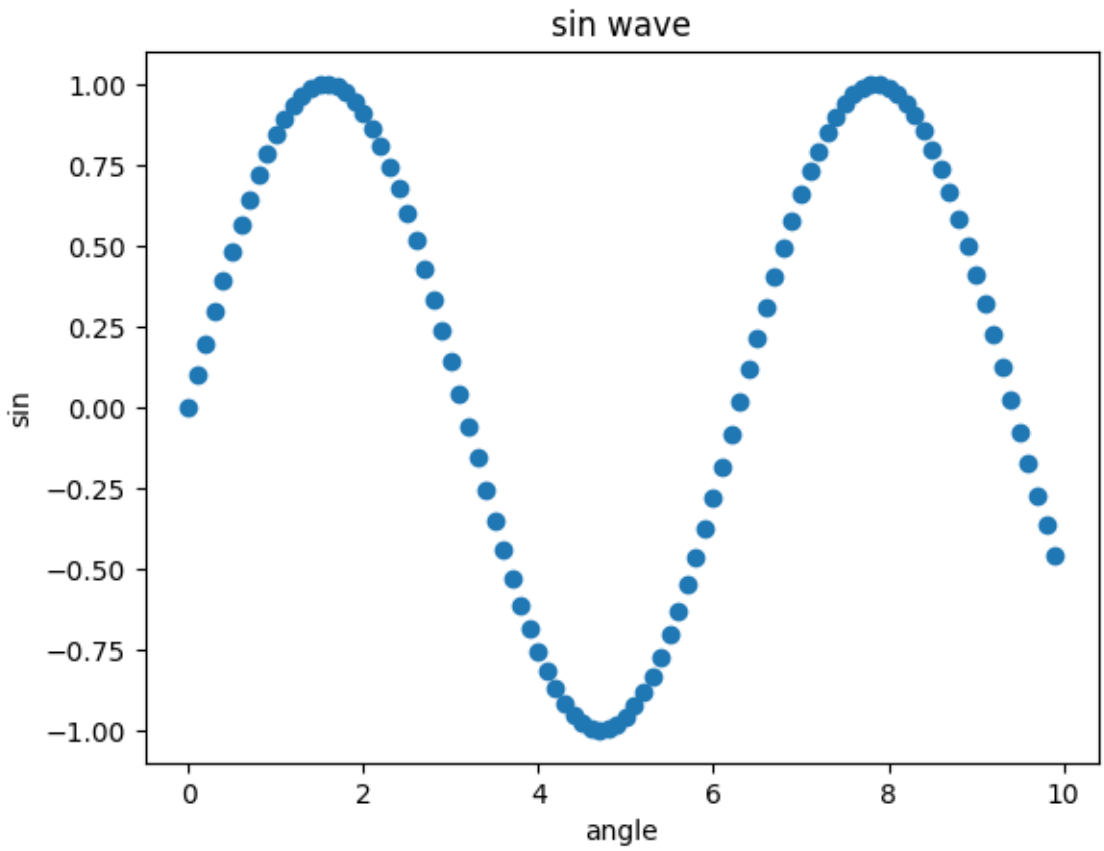
OUTPUT



CODE

```
plt.scatter(x,y)
plt.xlabel('angle')
plt.ylabel('sin')
plt.title('sin wave')
plt.show()
```

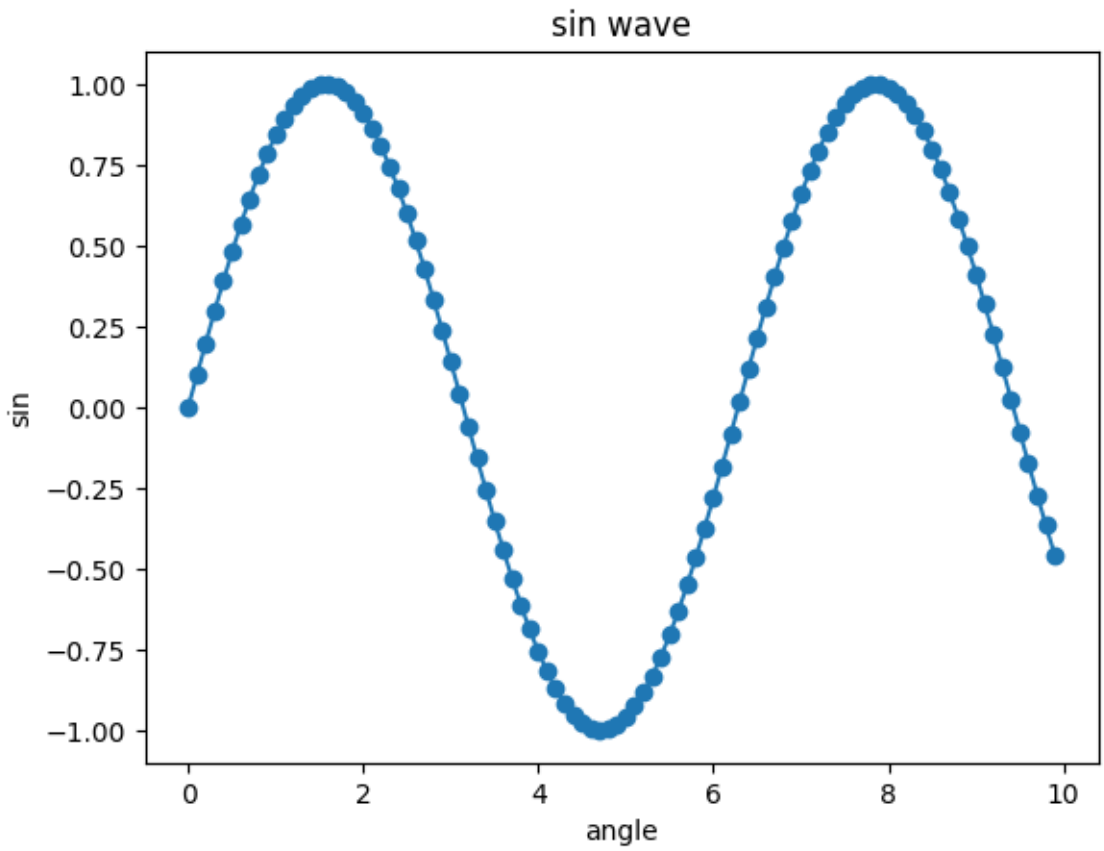
OUTPUT



CODE

```
plt.plot(x,y)
plt.scatter(x,y)
plt.xlabel('angle')
plt.ylabel('sin')
plt.title('sin wave')
plt.show()
```

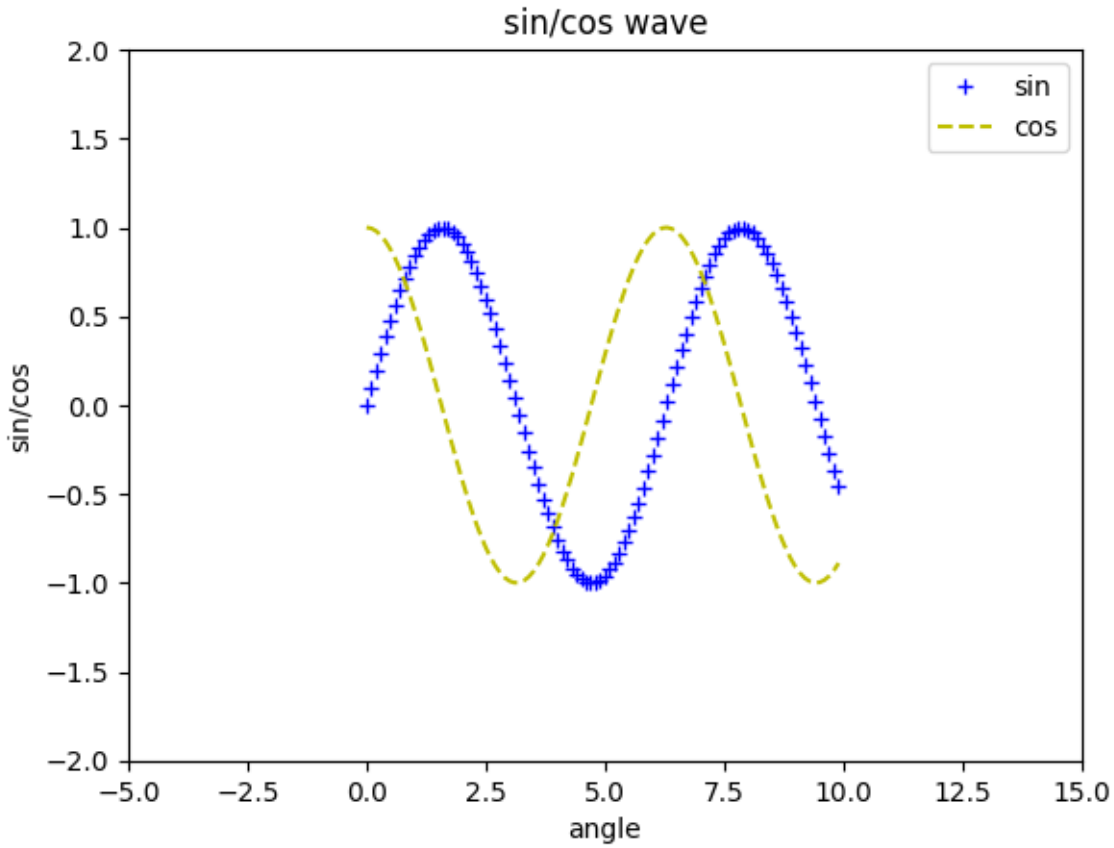
OUTPUT



CODE

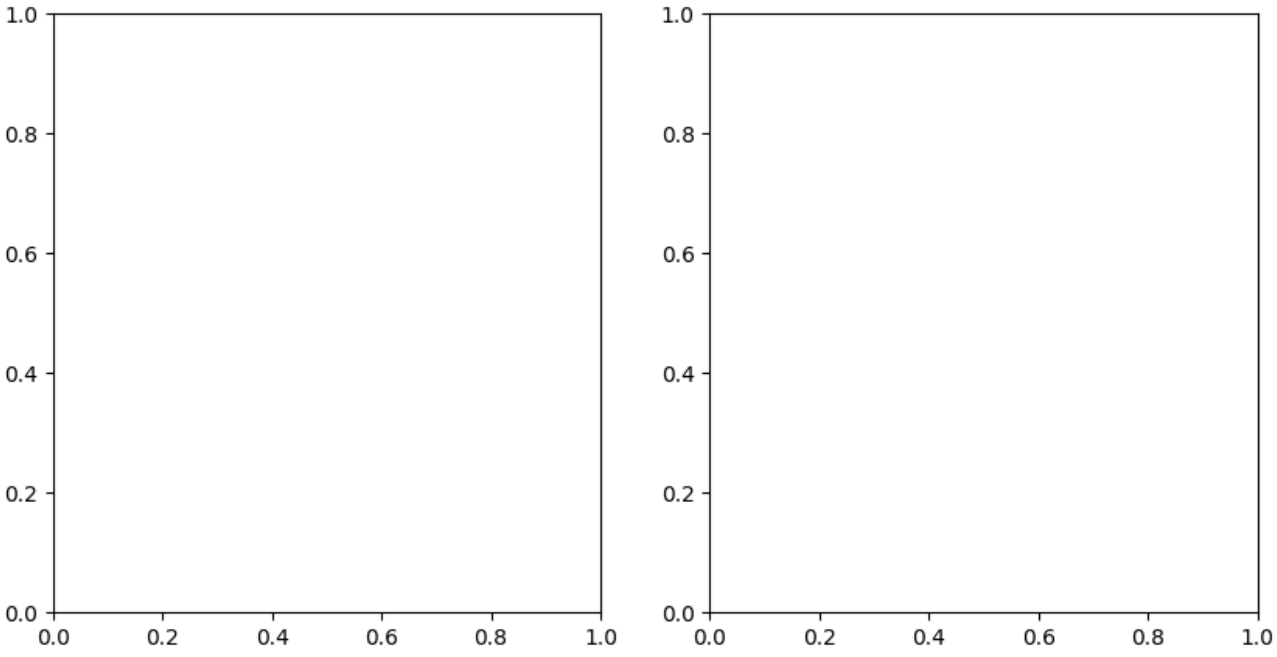
```
plt.plot(x,np.sin(x), 'b+', label='sin')
plt.plot(x,np.cos(x) ,'y--', label='cos') #Scatter connections
plt.xlabel('angle')
plt.ylabel('sin/cos')
plt.title('sin/cos wave')
plt.ylim(-2,2)
plt.xlim(-5,15)
plt.legend()
plt.show()
```

OUTPUT



CODE

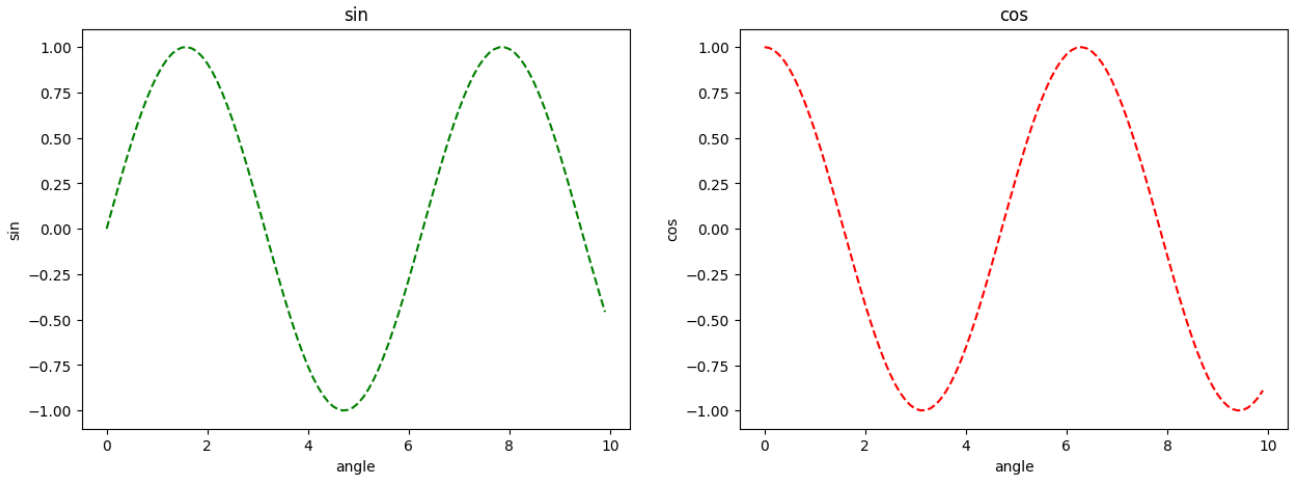
```
fig, axis = plt.subplots(1,2, figsize=(10,5))

print(axis.shape)
```



CODE

```
fig, axis = plt.subplots(1,2, figsize=(15,5))
x = np.arange(0,10,0.1)
axis[0].plot(x,np.sin(x), 'g--')
axis[0].set_title('sin')
axis[0].set_xlabel('angle')
axis[0].set_ylabel('sin')
axis[1].plot(x,np.cos(x), 'r--')
axis[1].set_title('cos')
axis[1].set_xlabel('angle')
axis[1].set_ylabel('cos')
plt.show()
```

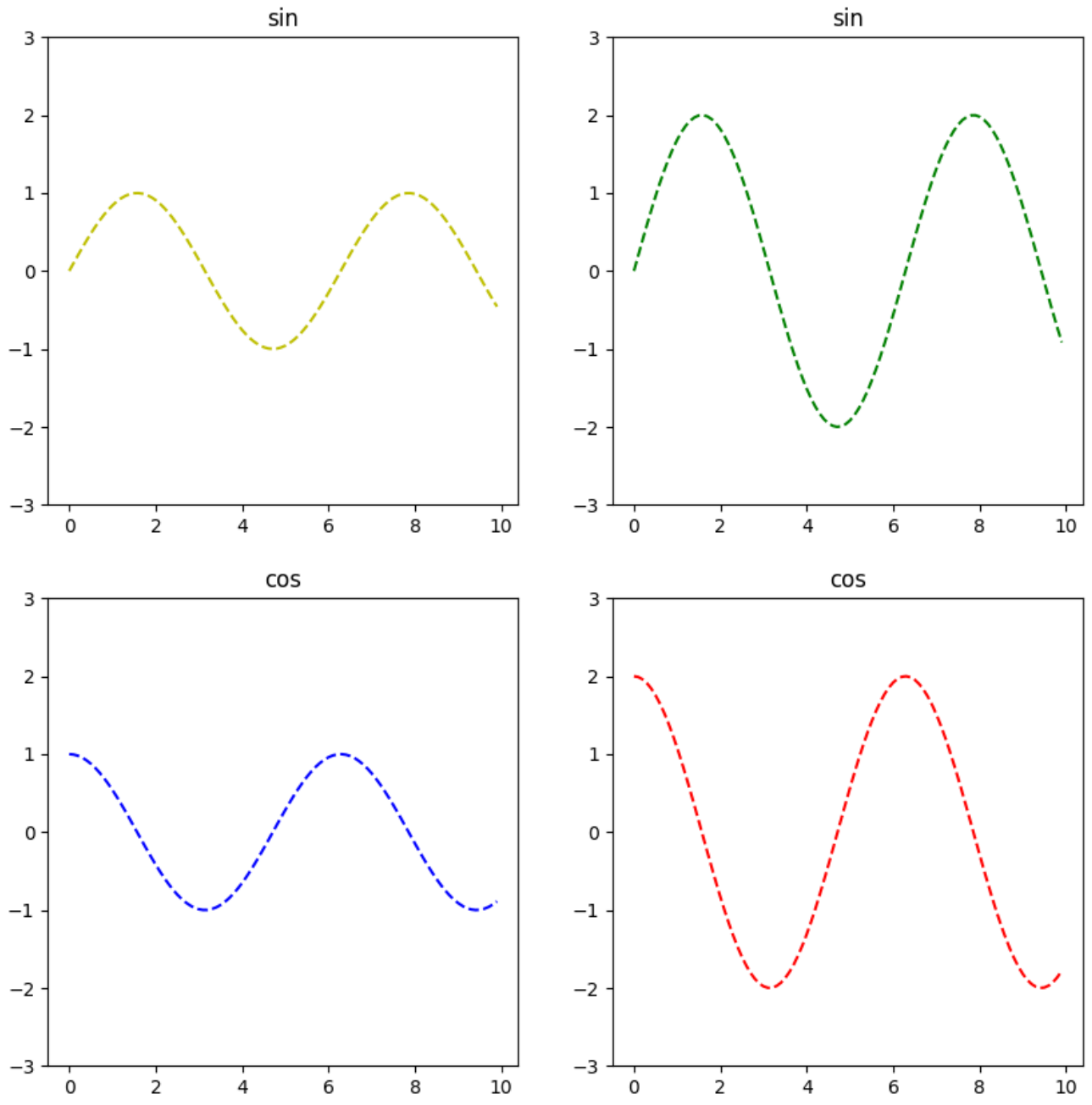
OUTPUT



CODE

```
fig, axis = plt.subplots(2,2, figsize=(10,10))
x = np.arange(0,10,0.1)
axis[0][0].plot(x,np.sin(x), 'y--')
axis[0][0].set_title('sin')
axis[0][0].set_ylim(-3,3)
axis[0][1].plot(x,2*np.sin(x), 'g--')
axis[0][1].set_title('sin')
axis[0][1].set_ylim(-3,3)
axis[1][0].plot(x,np.cos(x), 'b--')
axis[1][0].set_title('cos')
axis[1][0].set_ylim(-3,3)
axis[1][1].plot(x,2*np.cos(x), 'r--')
axis[1][1].set_title('cos')
axis[1][1].set_ylim(-3,3)
plt.show()
```

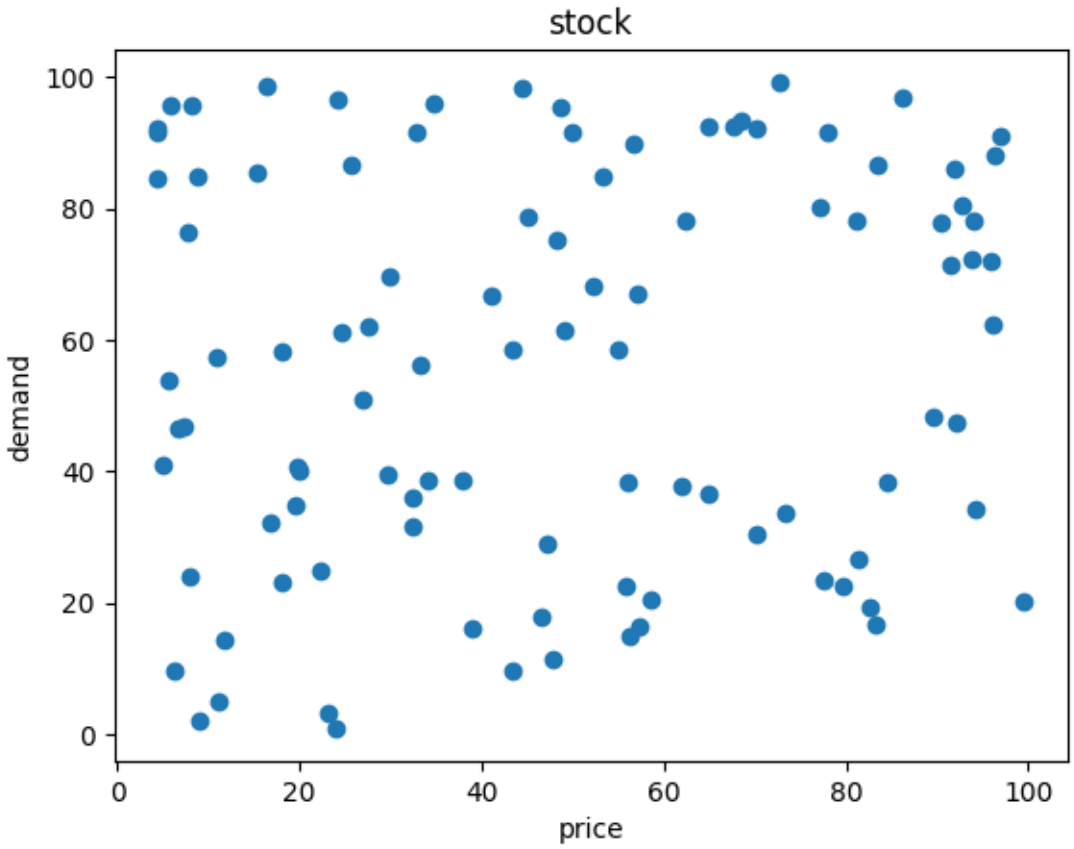
OUTPUT



CODE

```
x = np.random.random(100)*100
y = np.random.random(100)*100
plt.scatter(x,y)
plt.xlabel('price')
plt.ylabel('demand')
plt.title('stock')
plt.show()
```

OUTPUT

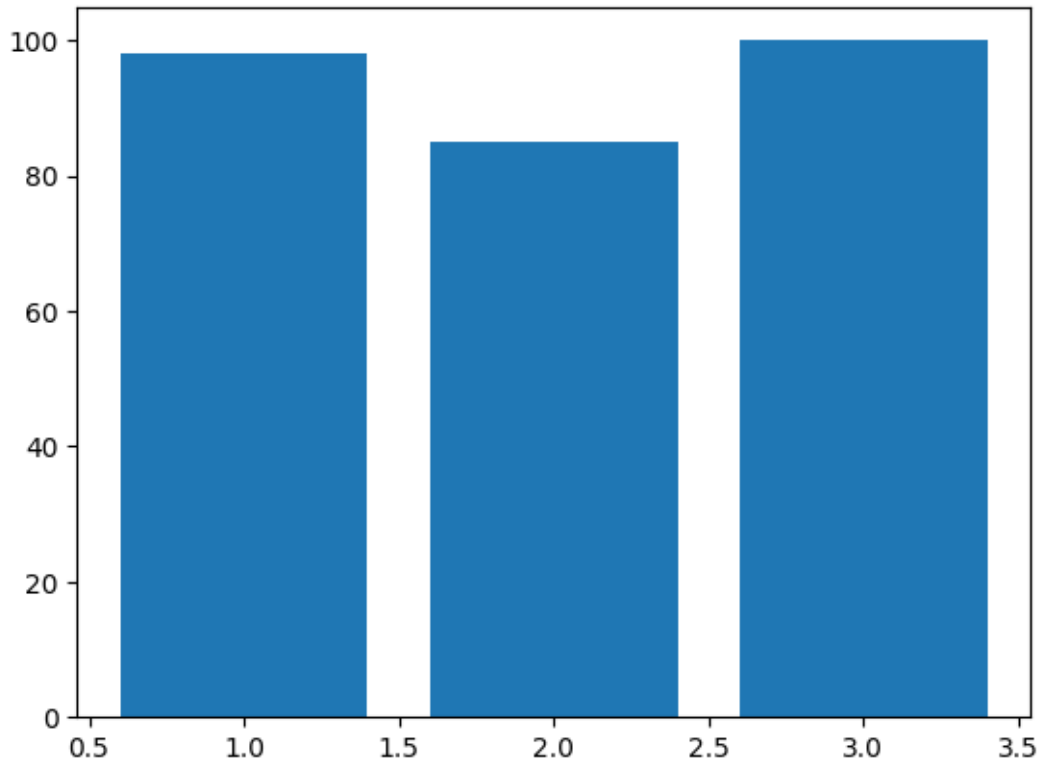


CODE

CODE

```
x = np.array([1,2,3])
y = [98,85,100]
plt.bar(x,y)
plt.show()
```

OUTPUT



CODE

```
import matplotlib.pyplot as plt
import seaborn as sns

sns.distplot([0, 1, 2, 3, 4, 5,3,2,3])

plt.show()
```

OUTPUT

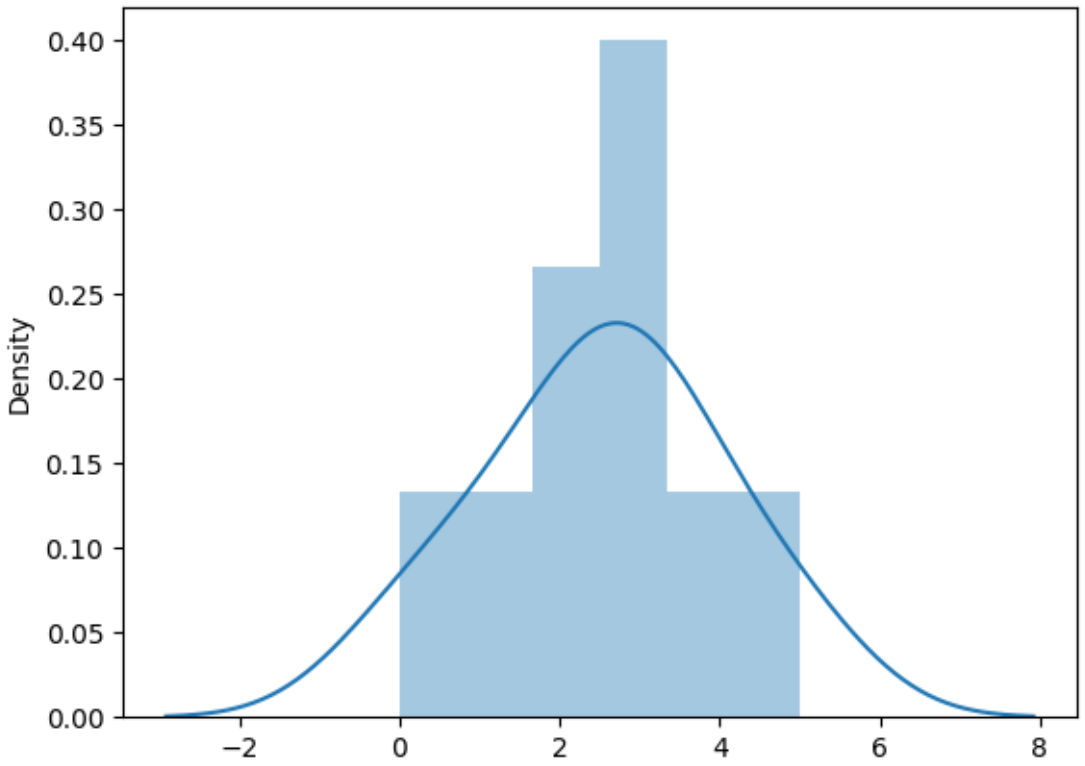
<ipython-input-17-592937749c9a>:4: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

sns.distplot([0, 1, 2, 3, 4, 5,3,2,3])



CODE

```
from numpy import random
import matplotlib.pyplot as plt
import seaborn as sns

x = np.random.random((1000,3))


sns.distplot(x[:,0], hist=True)
plt.show()

sns.distplot(x[:,1], hist=True)
plt.show()

sns.distplot(x[:,2], hist=True)
plt.show()
```

OUTPUT

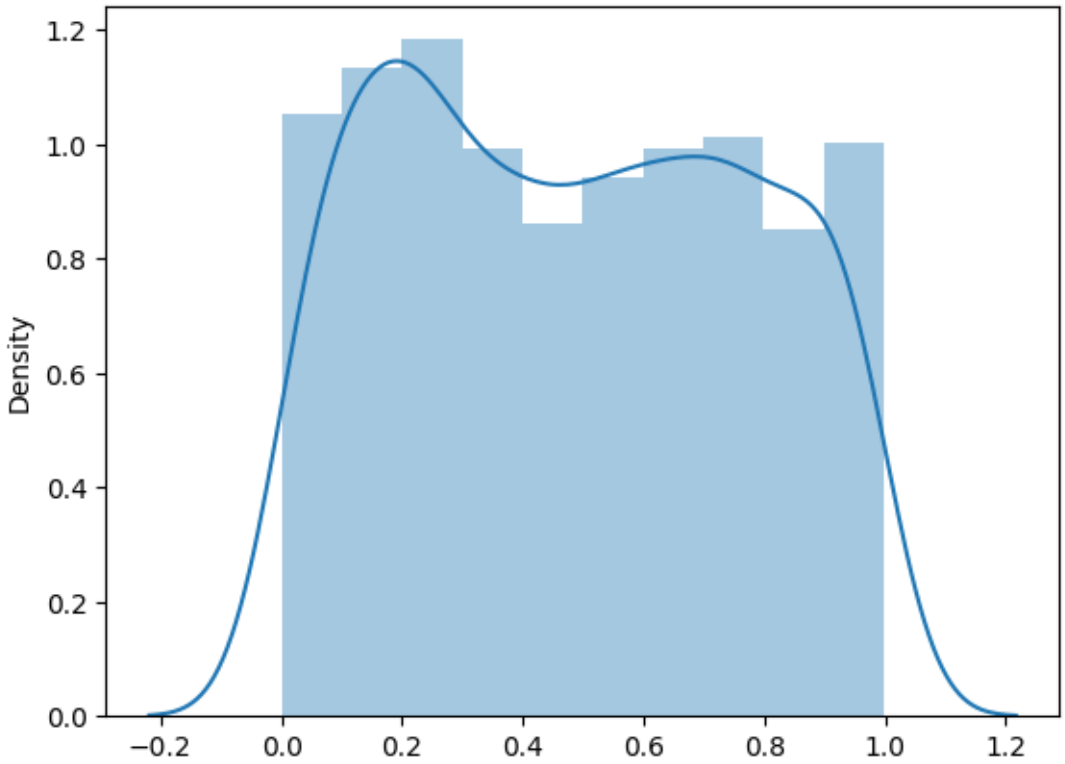
<ipython-input-18-044a4496ea0d>:8: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

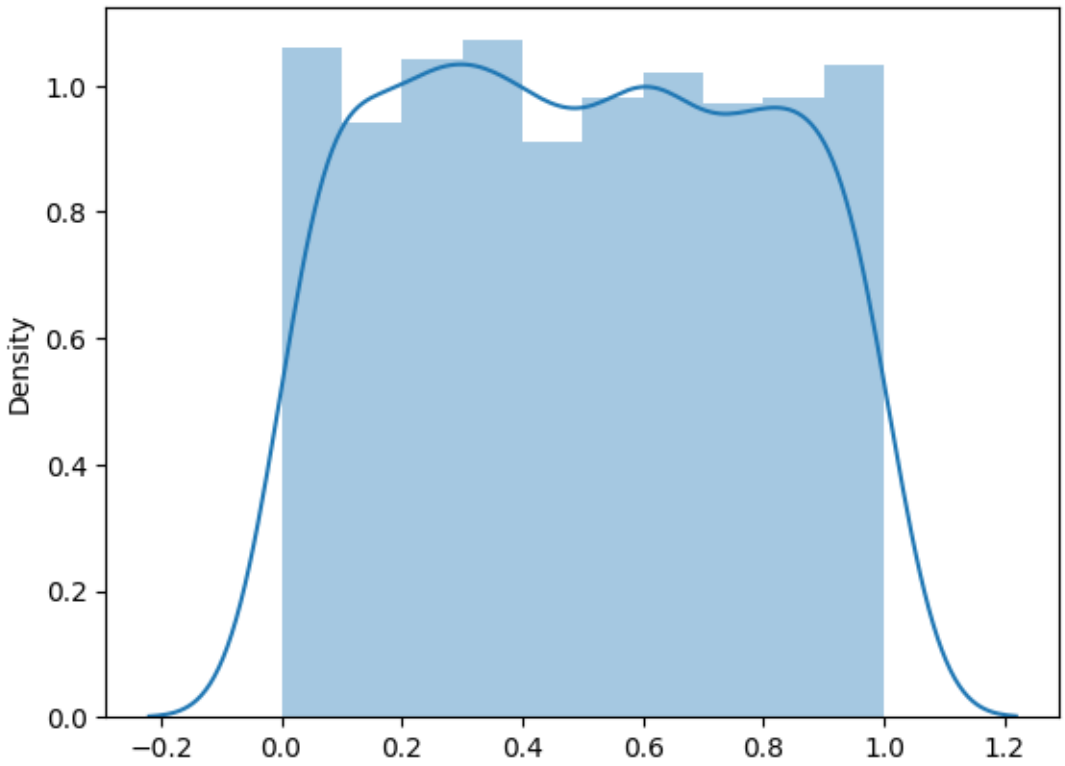
For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

```
sns.distplot(x[:,0], hist=True)
```




```
<ipython-input-18-044a4496ea0d>:11: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751
```

```
sns.distplot(x[:,1], hist=True)
```



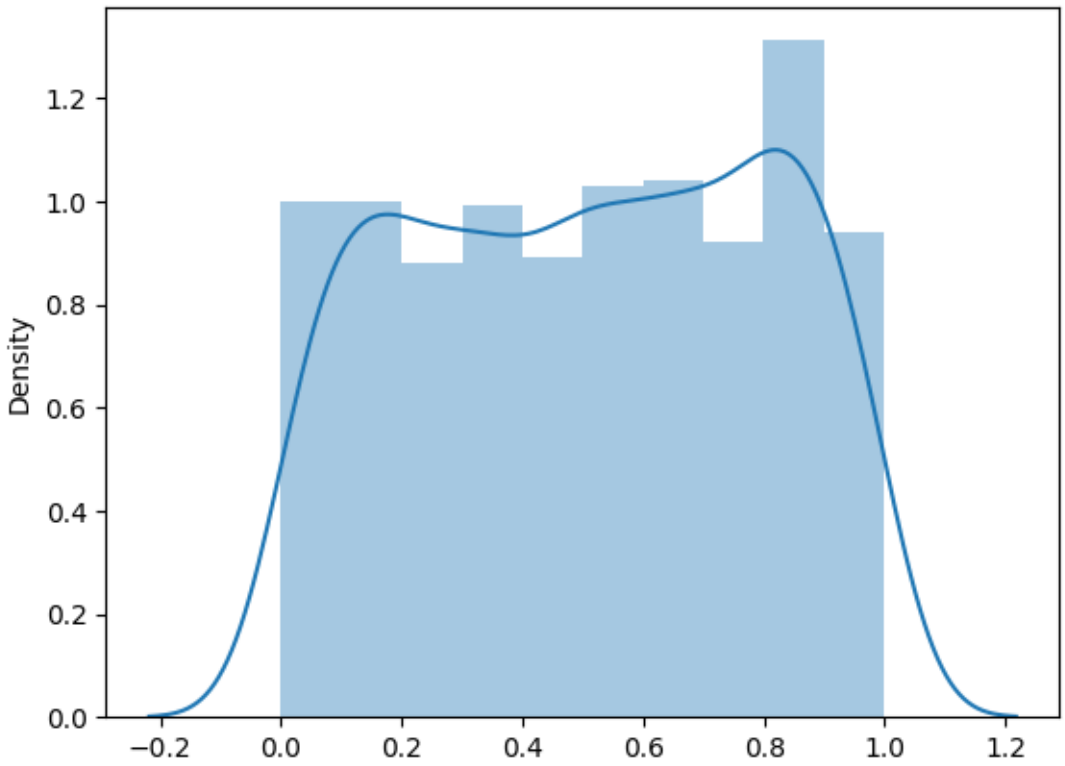
<ipython-input-18-044a4496ea0d>:14: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

```
sns.distplot(x[:,2], hist=True)
```

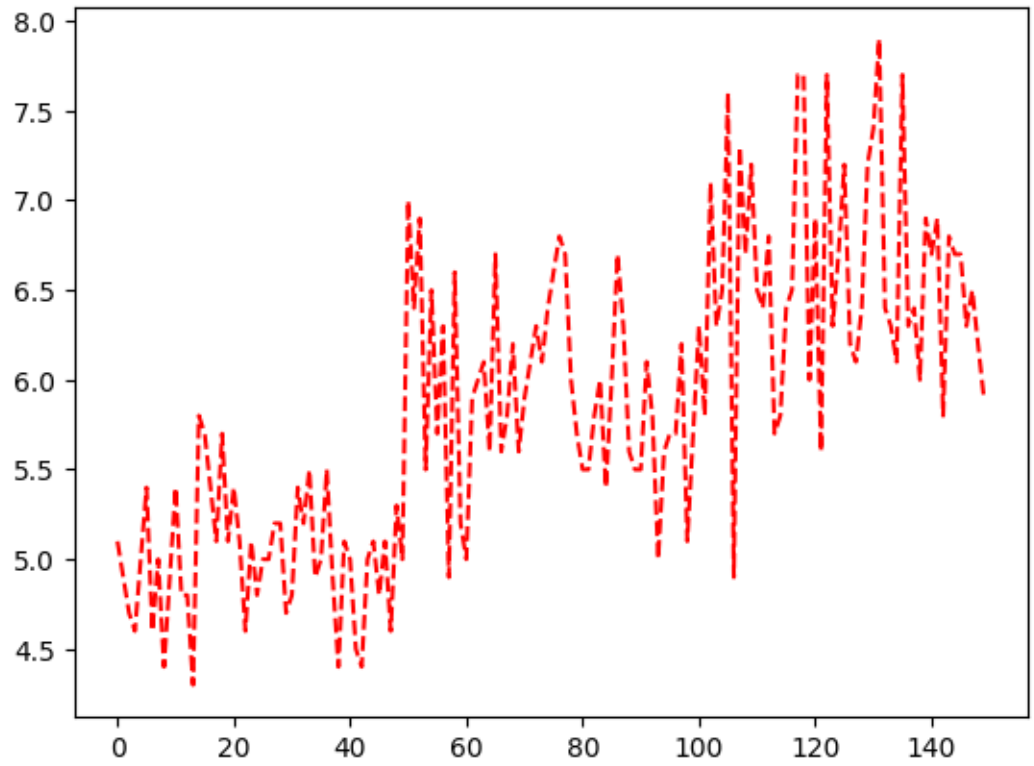


CODE

```
## Plotting Using Matplotlib

import matplotlib.pyplot as plt
plt.plot(iris["sepal.length"], "r--")
plt.show
```

OUTPUT

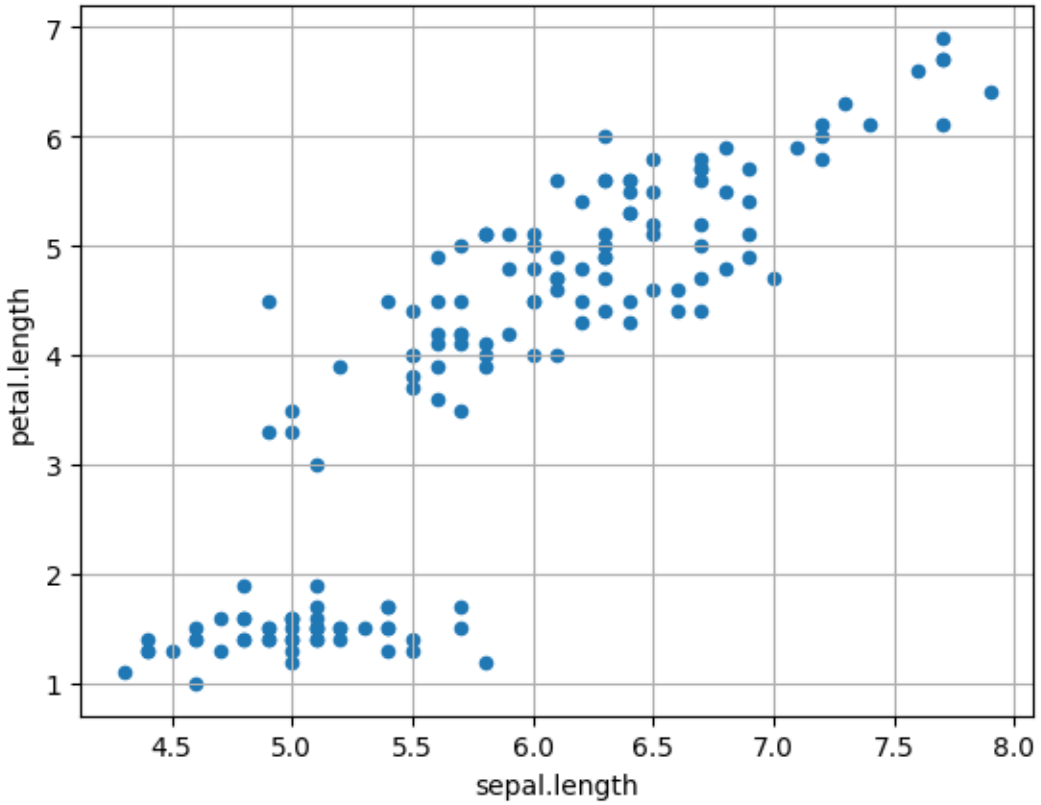
<function matplotlib.pyplot.show(close=None, block=None)>



CODE

```
##  Scatter Plot

iris.plot(kind ="scatter",
          x ='sepal.length',
          y ='petal.length')
plt.grid()
```

OUTPUT

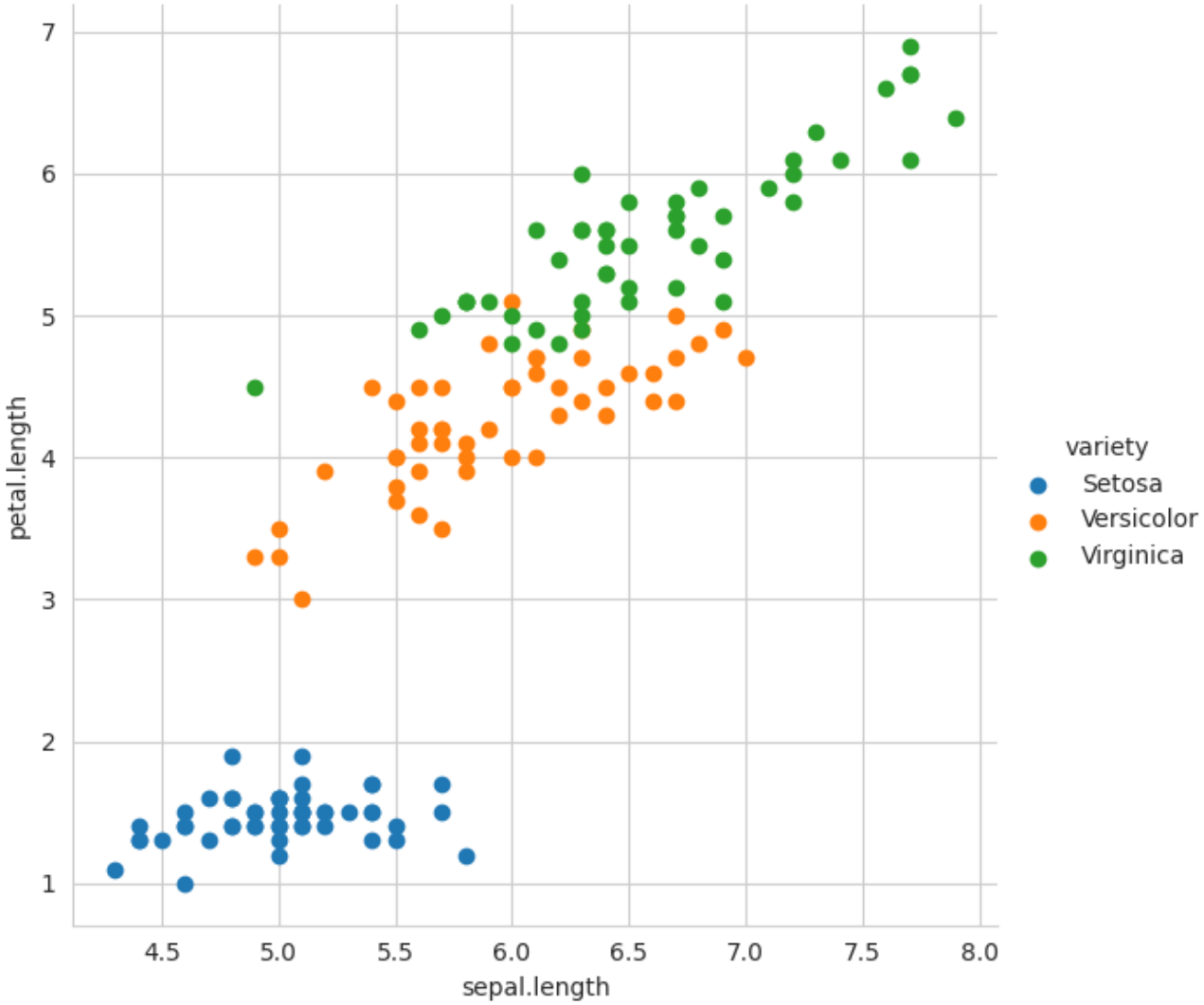


CODE

```
##  Plotting using Seaborn

import seaborn as sns

# style used as a theme of graph
# for example if we want black
# graph with grid then write "darkgrid"

sns.set_style("whitegrid")

# sepal.length, petal.length are iris
# feature, data height used to define
# Height of graph whereas hue store the
# class of iris dataset.

sns.FacetGrid(iris, hue ="variety",
              height = 6).map(plt.scatter,
                              'sepal.length',
                              'petal.length').add_legend()
```

OUTPUT

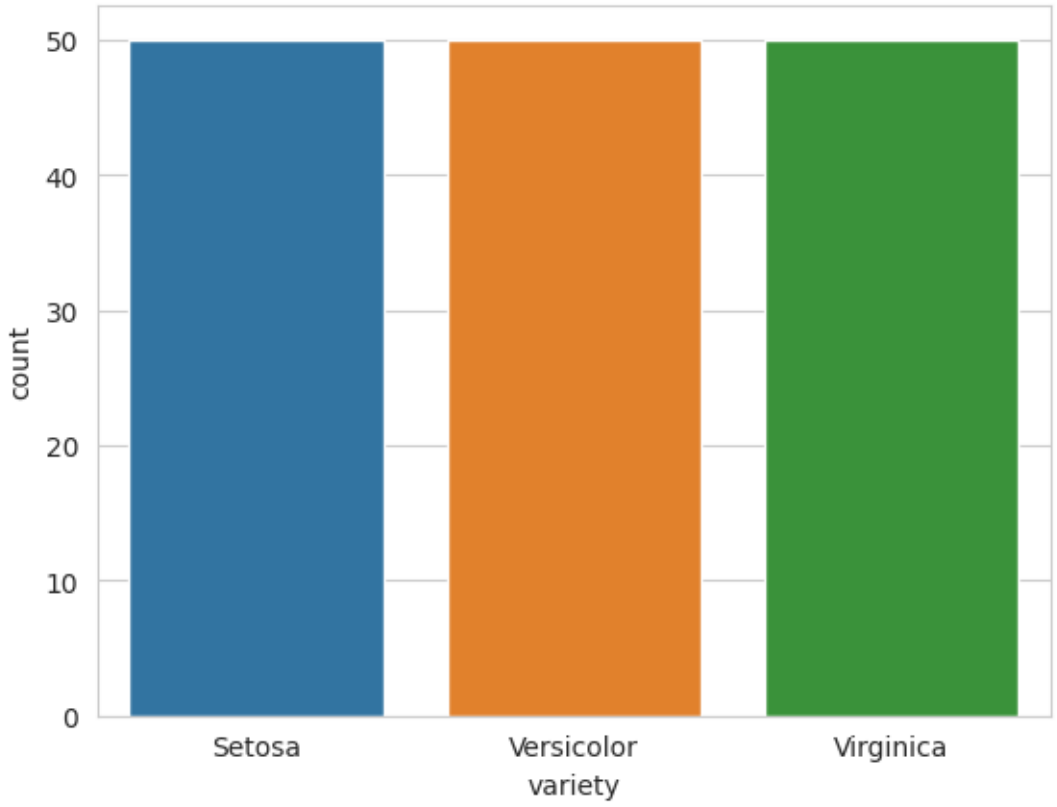
<seaborn.axisgrid.FacetGrid at 0x7ae0c9921cf0>



CODE

```
# Distribution Chart
#Visualizing the target(class label) column

sns.countplot(x='variety', data=iris, )
plt.show()
```

OUTPUT

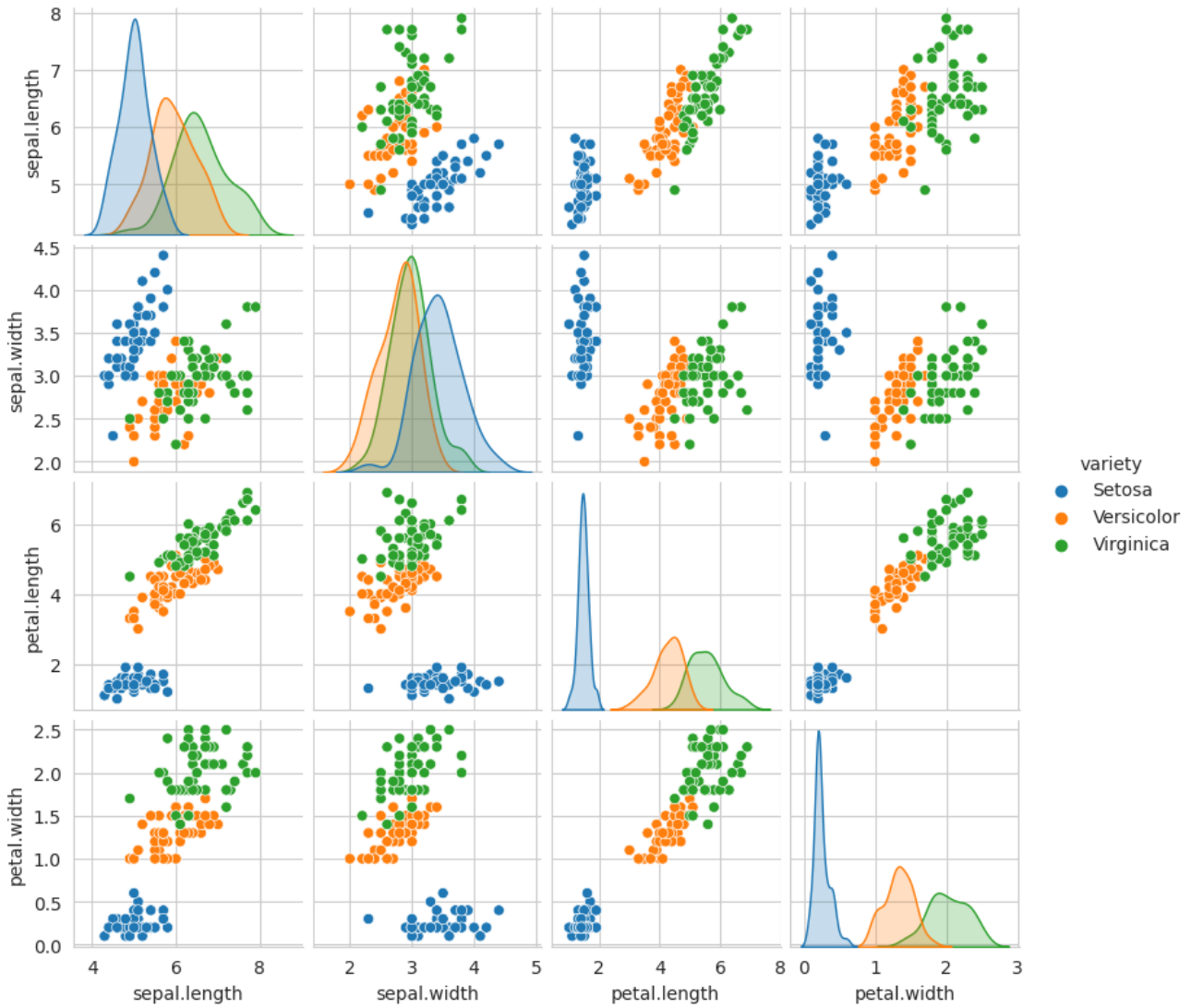


CODE

```
# plotting all the column's relationships using a pairplot. It can be used for multivariate analysis.

sns.pairplot(iris,hue='variety', height=2)
```

OUTPUT

<seaborn.axisgrid.PairGrid at 0x7ae0c98199c0>

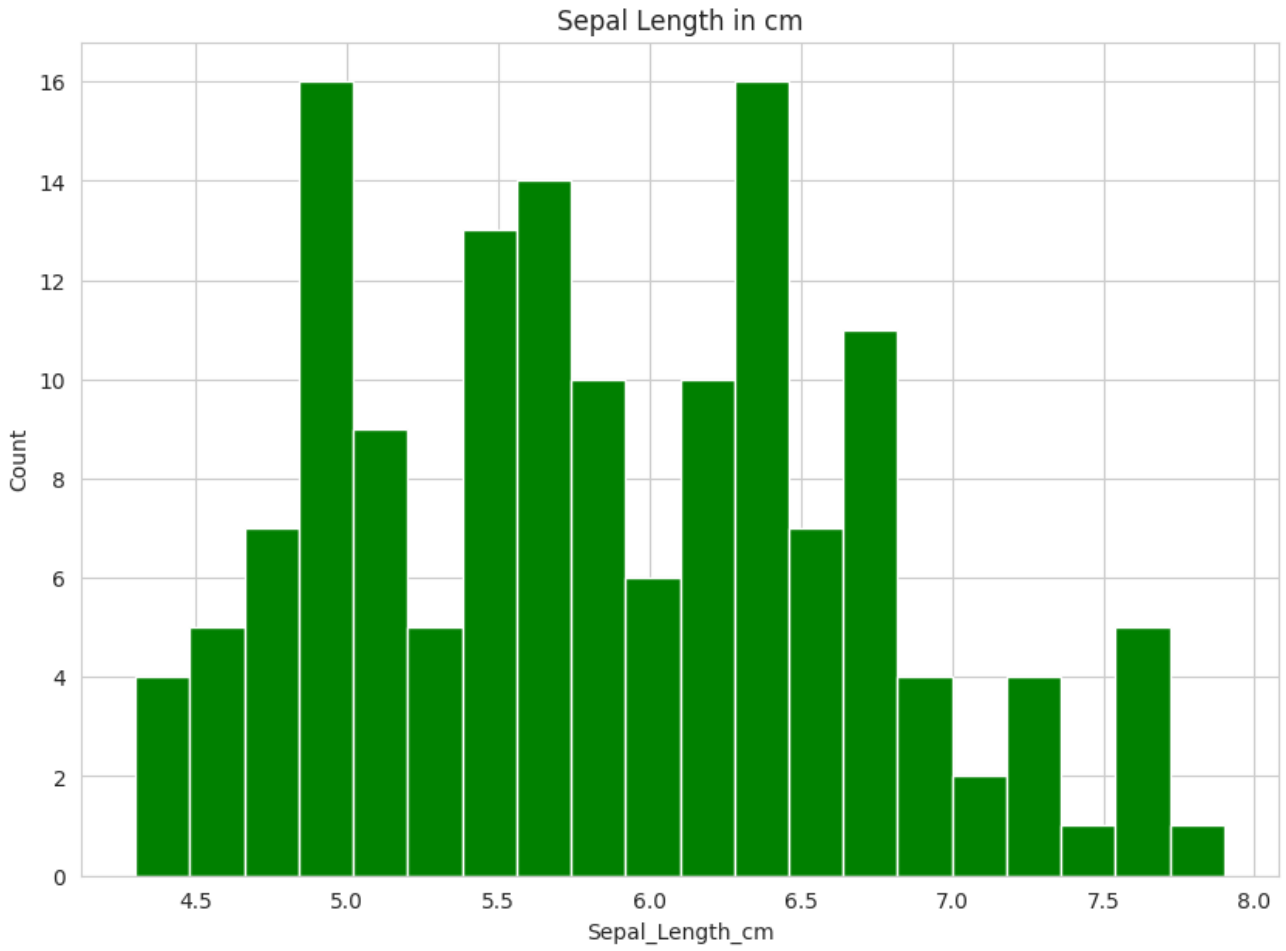


CODE

```
#Histogram for Sepal Length

plt.figure(figsize = (10, 7))
x = iris["sepal.length"]

plt.hist(x, bins = 20, color = "green")
plt.title("Sepal Length in cm")
plt.xlabel("Sepal_Length_cm")
plt.ylabel("Count")
```

OUTPUT

Text(0, 0.5, 'Count')

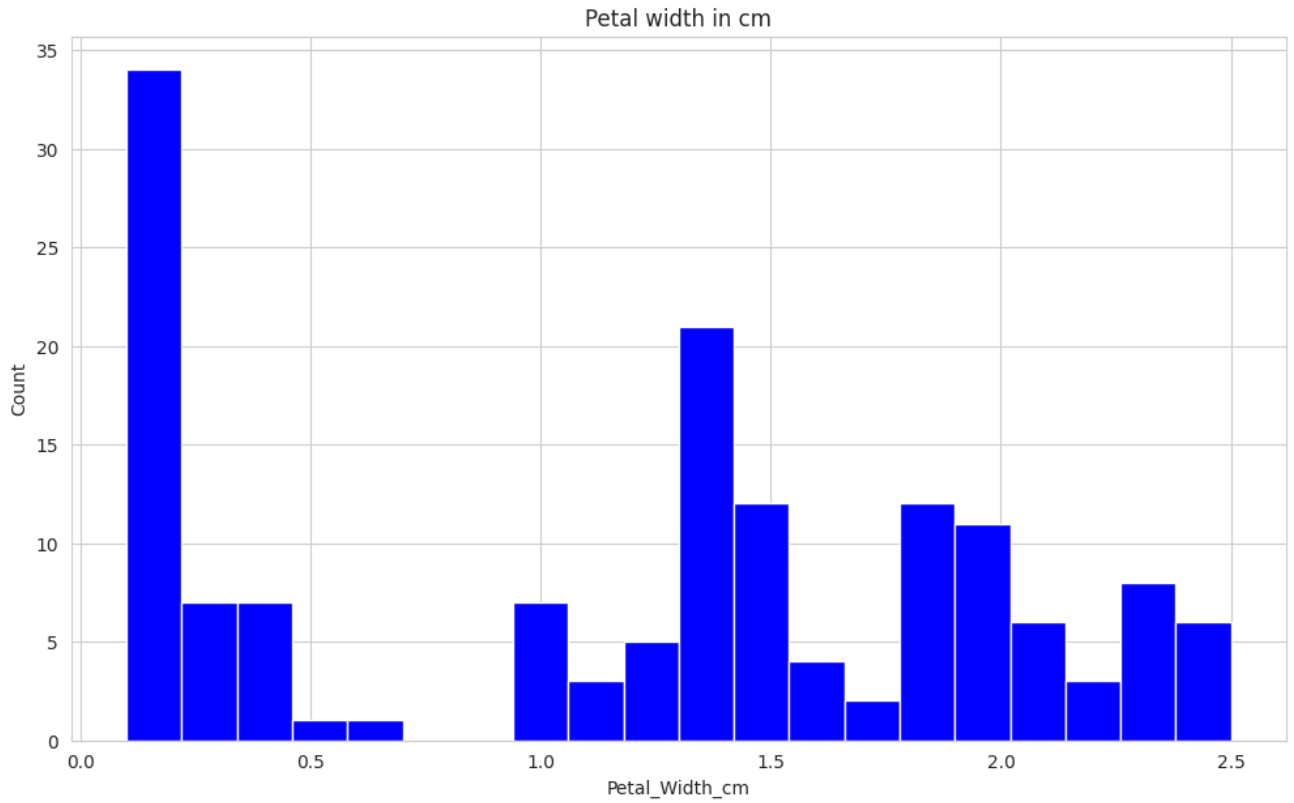


CODE

```
#Histogram for Petal Width
plt.figure(figsize = (12, 7))
x = iris["petal.width"]

plt.hist(x, bins =20, color = "blue")
plt.title("Petal width in cm")
plt.xlabel("Petal_Width_cm")
plt.ylabel("Count")
```

OUTPUT

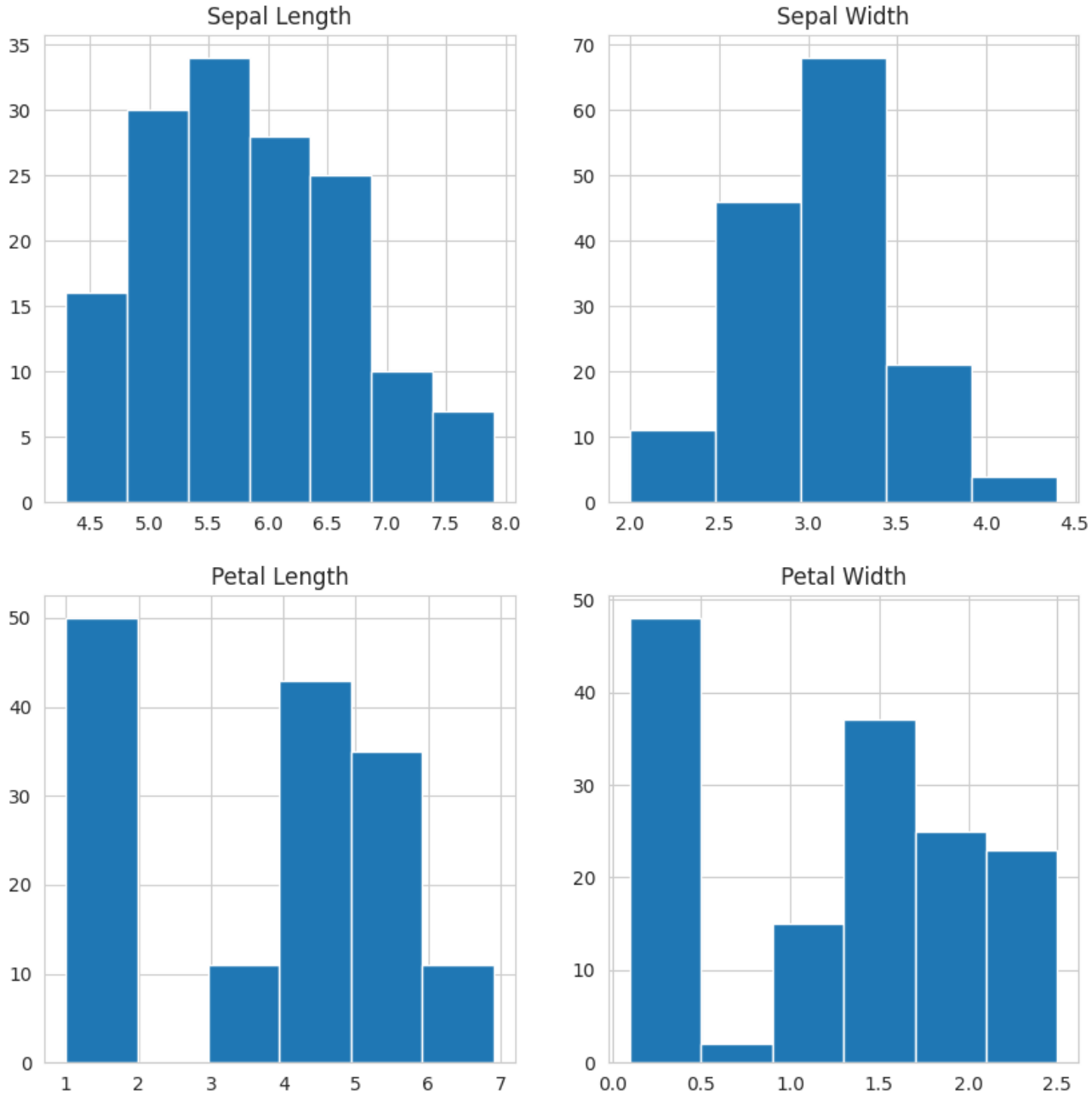
Text(0, 0.5, 'Count')



CODE

```
#Histograms allow seeing the distribution of data for various columns.
# It can be used for uni as well as bi-variate analysis.

fig, axes = plt.subplots(2, 2, figsize=(10,10))

axes[0,0].set_title("Sepal Length")
axes[0,0].hist(iris['sepal.length'], bins=7)

axes[0,1].set_title("Sepal Width")
axes[0,1].hist(iris['sepal.width'], bins=5);

axes[1,0].set_title("Petal Length")
axes[1,0].hist(iris['petal.length'], bins=6);

axes[1,1].set_title("Petal Width")
axes[1,1].hist(iris['petal.width'], bins=6);
```

OUTPUT



CODE

```
#Histograms with Distplot Plot

plot = sns.FacetGrid(iris, hue="variety")
plot.map(sns.distplot, "sepal.length").add_legend()

plot = sns.FacetGrid(iris, hue="variety")
plot.map(sns.distplot, "sepal.width").add_legend()

plot = sns.FacetGrid(iris, hue="variety")
plot.map(sns.distplot, "petal.length").add_legend()
```

```
plot = sns.FacetGrid(iris, hue="variety")
plot.map(sns.distplot, "petal.width").add_legend()

plt.show()

#In the case of Sepal Length, there is a huge amount of overlapping.
#In the case of Sepal Width also, there is a huge amount of overlapping.
#In the case of Petal Length, there is a very little amount of overlapping.
#In the case of Petal Width also, there is a very little amount of overlapping.
```

OUTPUT

```
/usr/local/lib/python3.10/dist-packages/seaborn/axisgrid.py:848: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  func(*plot_args, **plot_kwargs)
/usr/local/lib/python3.10/dist-packages/seaborn/axisgrid.py:848: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  func(*plot_args, **plot_kwargs)
/usr/local/lib/python3.10/dist-packages/seaborn/axisgrid.py:848: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  func(*plot_args, **plot_kwargs)
/usr/local/lib/python3.10/dist-packages/seaborn/axisgrid.py:848: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.
```

Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

func(*plot_args, **plot_kwargs)
/usr/local/lib/python3.10/dist-packages/seaborn/axisgrid.py:848: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

func(*plot_args, **plot_kwargs)
/usr/local/lib/python3.10/dist-packages/seaborn/axisgrid.py:848: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

func(*plot_args, **plot_kwargs)
/usr/local/lib/python3.10/dist-packages/seaborn/axisgrid.py:848: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

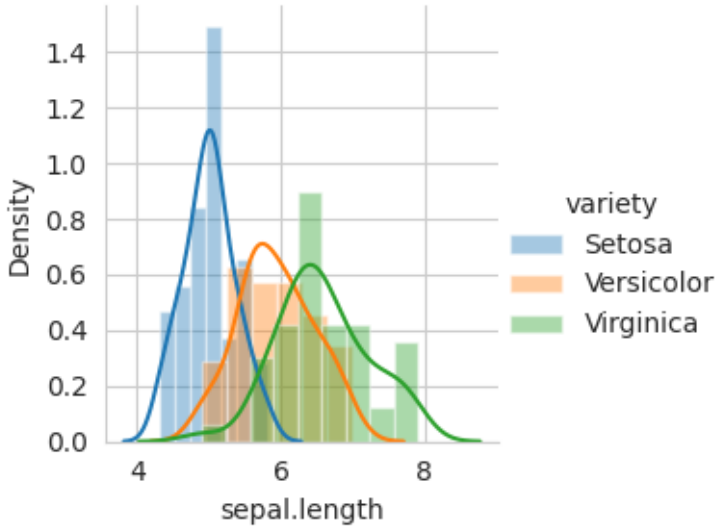
func(*plot_args, **plot_kwargs)
/usr/local/lib/python3.10/dist-packages/seaborn/axisgrid.py:848: UserWarning:

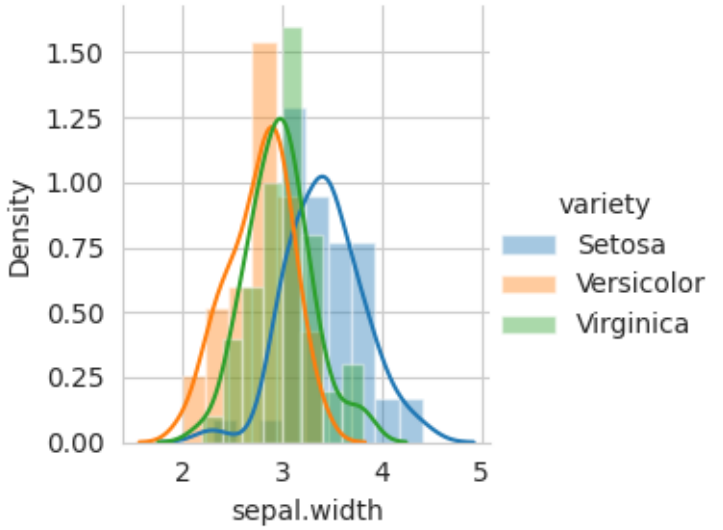
`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

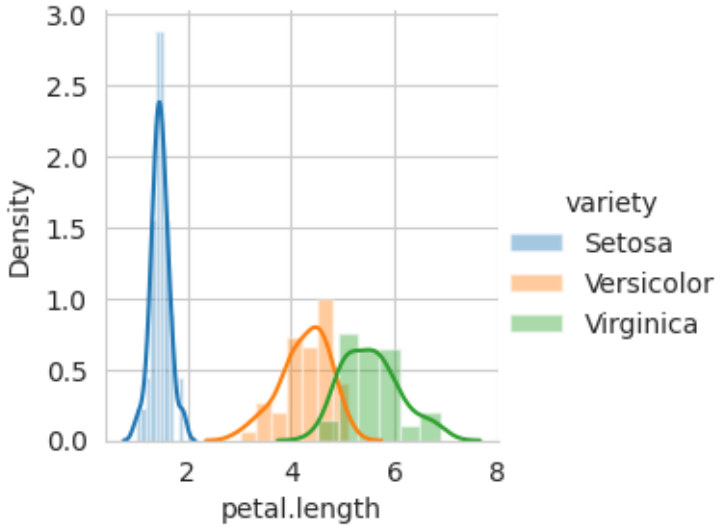
Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).

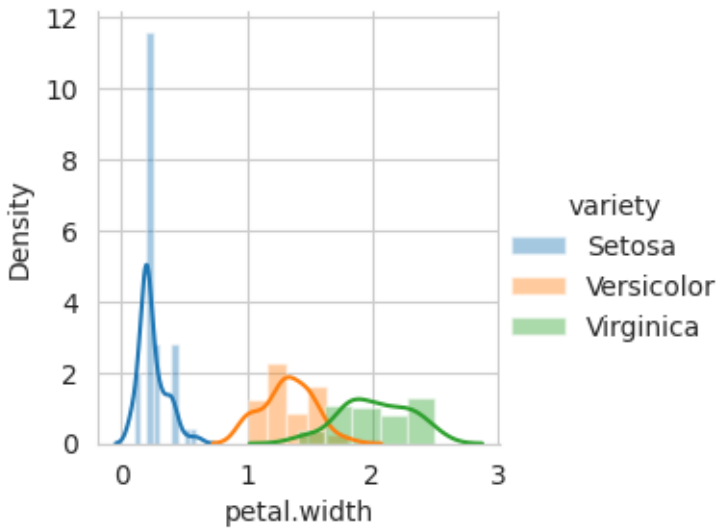
For a guide to updating your code to use the new functions, please see https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

```
  func(*plot_args, **plot_kwargs)
/usr/local/lib/python3.10/dist-packages/seaborn/axisgrid.py:848: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  func(*plot_args, **plot_kwargs)
/usr/local/lib/python3.10/dist-packages/seaborn/axisgrid.py:848: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  func(*plot_args, **plot_kwargs)
/usr/local/lib/python3.10/dist-packages/seaborn/axisgrid.py:848: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  func(*plot_args, **plot_kwargs)
/usr/local/lib/python3.10/dist-packages/seaborn/axisgrid.py:848: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  func(*plot_args, **plot_kwargs)
```


Density
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
4
6
8
sepal.length
variety
Setosa
Versicolor
Virginica


Density
1.50
1.25
1.00
0.75
0.50
0.25
0.00
2
3
4
5
sepal.width
variety
Setosa
Versicolor
Virginica


Density
3.0
2.5
2.0
1.5
1.0
0.5
0.0
2
4
6
8
petal.length
variety
Setosa
Versicolor
Virginica

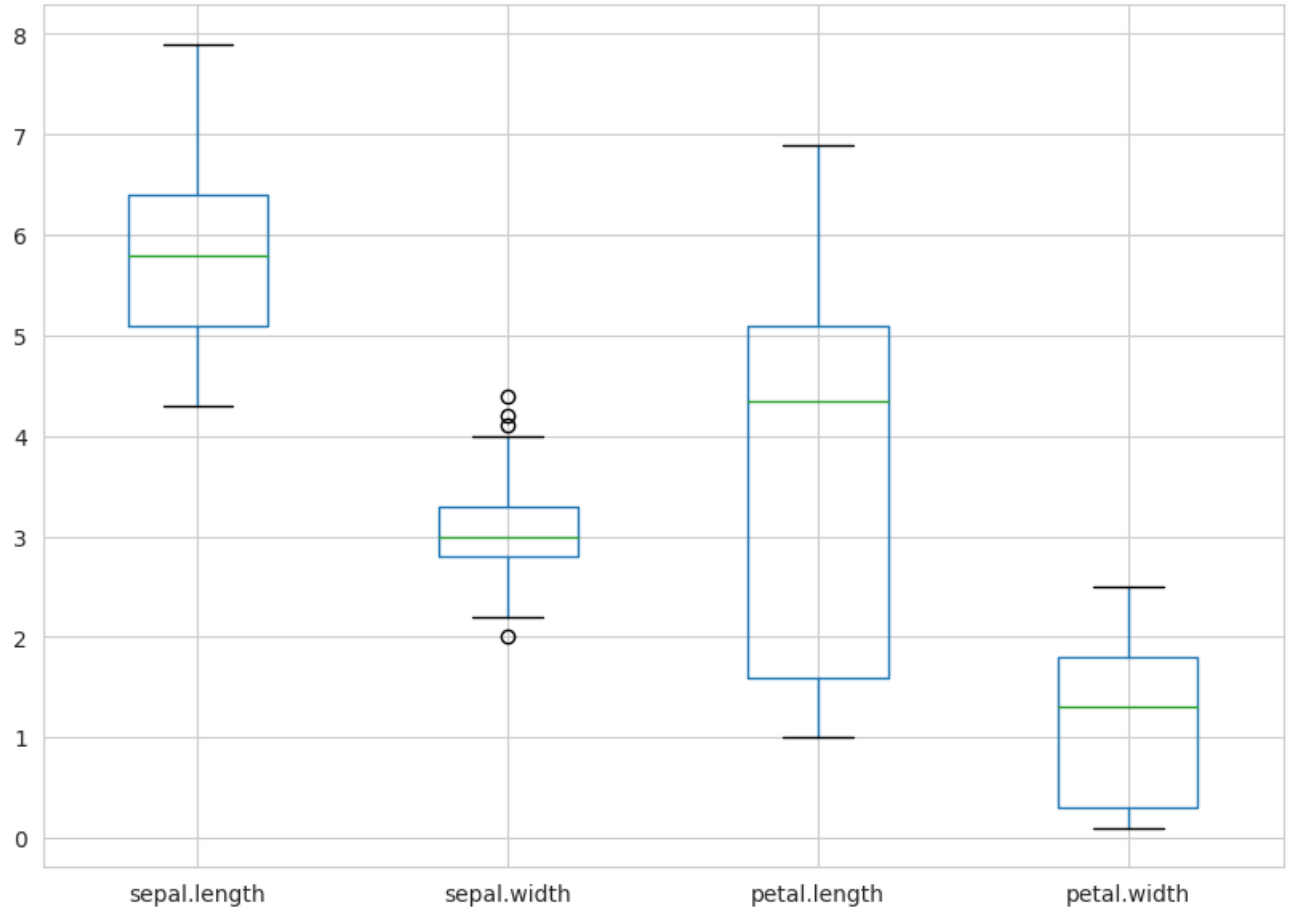

12
10
8
6
4
2
0
Density
0
1
2
3
petal.width
variety
Setosa
Versicolor
Virginica

CODE

```
# Box Plot for Iris Data

plt.figure(figsize = (10, 7))
iris.boxplot()
```

OUTPUT

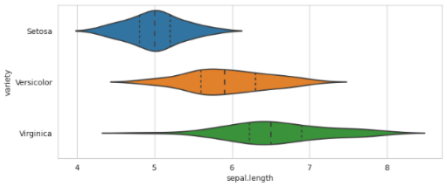
<Axes: >

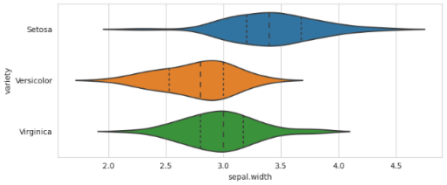


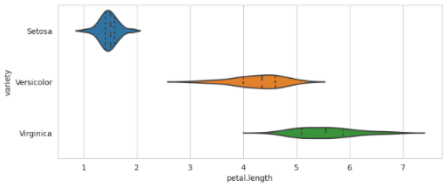
CODE

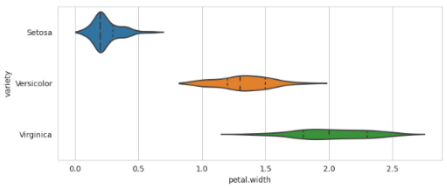
```
import matplotlib.gridspec as gridspec
fig = plt.figure(figsize=(9, 40))
outer = gridspec.GridSpec(4, 1, wspace=0.2, hspace=0.2)
for i, col in enumerate(iris.columns[:-1]):
  inner = gridspec.GridSpecFromSubplotSpec(2, 1,subplot_spec=outer[i], wspace=0.2, hspace=0.4)
  ax = plt.Subplot(fig, inner[1])
  _ = sns.violinplot(y="variety", x=f"{col}", data=iris, inner='quartile', ax=ax)
  fig.add_subplot(ax)
fig.show()
```

OUTPUT


Setosa
Versicolor
Virginica
variety
4
5
6
7
8
sepal.length


Setosa
Versicolor
Virginica
variety
2.0
2.5
3.0
3.5
4.0
4.5
sepal.width


Setosa
Versicolor
Virginica
variety
1
2
3
4
5
6
7
petal.length

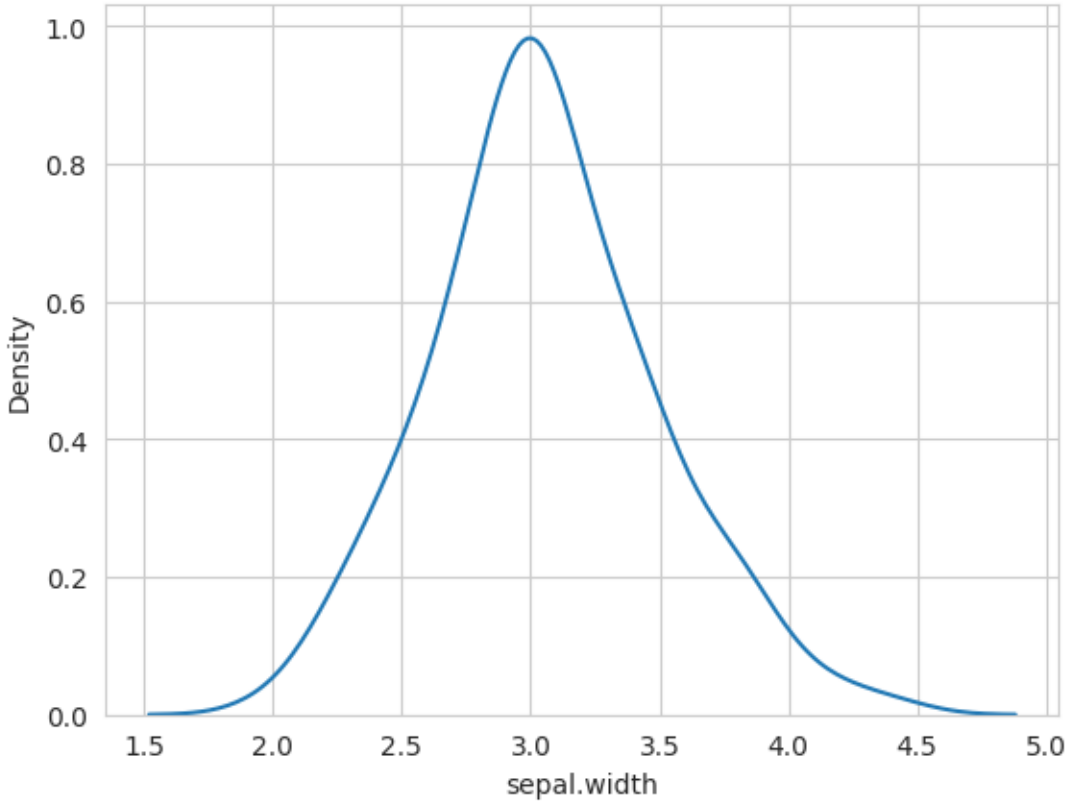

Setosa
Versicolor
Virginica
variety
0.0
0.5
1.0
1.5
2.0
2.5
petal.width

CODE

```
# Make default density plot
sns.kdeplot(iris['sepal.width'])
```

OUTPUT

<Axes: xlabel='sepal.width', ylabel='Density'>



EXPERIMENT NO :5

AIM: KNN Classification using iris dataset

CODE

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
df = pd.read_csv("iris.csv")
print(df)
```

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
..            ...          ...           ...          ...        ...
145           6.7          3.0           5.2          2.3  Virginica
146           6.3          2.5           5.0          1.9  Virginica
147           6.5          3.0           5.2          2.0  Virginica
148           6.2          3.4           5.4          2.3  Virginica
149           5.9          3.0           5.1          1.8  Virginica

[150 rows x 5 columns]
```

CODE

```
df['variety'].value_counts()
```

OUTPUT

```
Setosa        50
Versicolor    50
Virginica     50
Name: variety, dtype: int64
```

CODE

```
X = df.drop('variety', axis=1)
y = df['variety']
```

```
# splitting to trainset and Test set in the ratio 70:30

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30)
print(X_train)
print(" ")
print(X_test)
```

OUTPUT

```
     sepal.length  sepal.width  petal.length  petal.width
146           6.3          2.5           5.0          1.9
49            5.0          3.3           1.4          0.2
123           6.3          2.7           4.9          1.8
21            5.1          3.7           1.5          0.4
63            6.1          2.9           4.7          1.4
..            ...          ...           ...          ...
82            5.8          2.7           3.9          1.2
42            4.4          3.2           1.3          0.2
104           6.5          3.0           5.8          2.2
28            5.2          3.4           1.4          0.2
71            6.1          2.8           4.0          1.3

[105 rows x 4 columns]
```

	sepal.length	sepal.width	petal.length	petal.width
20	5.4	3.4	1.7	0.2
130	7.4	2.8	6.1	1.9
62	6.0	2.2	4.0	1.0
11	4.8	3.4	1.6	0.2
143	6.8	3.2	5.9	2.3
50	7.0	3.2	4.7	1.4
35	5.0	3.2	1.2	0.2
148	6.2	3.4	5.4	2.3
112	6.8	3.0	5.5	2.1
128	6.4	2.8	5.6	2.1
93	5.0	2.3	3.3	1.0
31	5.4	3.4	1.5	0.4
74	6.4	2.9	4.3	1.3
139	6.9	3.1	5.4	2.1
133	6.3	2.8	5.1	1.5
117	7.7	3.8	6.7	2.2
68	6.2	2.2	4.5	1.5
44	5.1	3.8	1.9	0.4
115	6.4	3.2	5.3	2.3
6	4.6	3.4	1.4	0.3
105	7.6	3.0	6.6	2.1
64	5.6	2.9	3.6	1.3
147	6.5	3.0	5.2	2.0
73	6.1	2.8	4.7	1.2
18	5.7	3.8	1.7	0.3
52	6.9	3.1	4.9	1.5
46	5.1	3.8	1.6	0.2
109	7.2	3.6	6.1	2.5
126	6.2	2.8	4.8	1.8
72	6.3	2.5	4.9	1.5
149	5.9	3.0	5.1	1.8
22	4.6	3.6	1.0	0.2

24	4.8	3.4	1.9	0.2
16	5.4	3.9	1.3	0.4
58	6.6	2.9	4.6	1.3
67	5.8	2.7	4.1	1.0
120	6.9	3.2	5.7	2.3
94	5.6	2.7	4.2	1.3
141	6.9	3.1	5.1	2.3
97	6.2	2.9	4.3	1.3
59	5.2	2.7	3.9	1.4
89	5.5	2.5	4.0	1.3
48	5.3	3.7	1.5	0.2
4	5.0	3.6	1.4	0.2
12	4.8	3.0	1.4	0.1

CODE

```
print("Number transactions X_train dataset: ", X_train.shape)
print("Number transactions y_train dataset: ", y_train.shape)
print("Number transactions X_test dataset: ", X_test.shape)
print("Number transactions y_test dataset: ", y_test.shape)
```

OUTPUT

```
Number transactions X_train dataset:  (105, 4)
Number transactions y_train dataset:  (105,)
Number transactions X_test dataset:  (45, 4)
Number transactions y_test dataset:  (45,)
```

CODE

```
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
print(y_pred)
print(' ')
print(y_test)
```

OUTPUT

```
['Setosa' 'Virginica' 'Versicolor' 'Setosa' 'Virginica' 'Versicolor'
 'Setosa' 'Virginica' 'Virginica' 'Virginica' 'Versicolor' 'Setosa'
 'Versicolor' 'Virginica' 'Virginica' 'Virginica' 'Virginica' 'Setosa'
 'Virginica' 'Setosa' 'Virginica' 'Versicolor' 'Virginica' 'Versicolor'
 'Setosa' 'Versicolor' 'Setosa' 'Virginica' 'Virginica' 'Virginica'
 'Virginica' 'Setosa' 'Setosa' 'Setosa' 'Versicolor' 'Versicolor'
 'Virginica' 'Versicolor' 'Virginica' 'Versicolor' 'Versicolor'
 'Versicolor' 'Setosa' 'Setosa' 'Setosa']
```

```
20         Setosa
130     Virginica
62     Versicolor
11         Setosa
143     Virginica
50     Versicolor
35         Setosa
148     Virginica
112     Virginica
128     Virginica
93     Versicolor
31         Setosa
74     Versicolor
139     Virginica
133     Virginica
117     Virginica
68     Versicolor
44         Setosa
115     Virginica
6          Setosa
105     Virginica
64     Versicolor
147     Virginica
73     Versicolor
18         Setosa
52     Versicolor
46         Setosa
109     Virginica
126     Virginica
72     Versicolor
149     Virginica
```

```
22         Setosa
24         Setosa
16         Setosa
58     Versicolor
67     Versicolor
120     Virginica
94     Versicolor
141     Virginica
97     Versicolor
59     Versicolor
89     Versicolor
48         Setosa
4          Setosa
12         Setosa
Name: variety, dtype: object
```

CODE

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

OUTPUT

```
[[14  0  0]
 [ 0 13  2]
 [ 0  0 16]]
              precision    recall  f1-score   support

      Setosa       1.00      1.00      1.00        14
  Versicolor       1.00      0.87      0.93        15
   Virginica       0.89      1.00      0.94        16

    accuracy                           0.96        45
   macro avg       0.96      0.96      0.96        45
weighted avg       0.96      0.96      0.96        45
```

CODE

```
pred_op=classifier.predict([[5.1,3.5,1.4,0.2]])
print(pred_op)
```

OUTPUT

```
['Setosa']
```

EXPERIMENT NO :6

AIM: KNN classification using Fruit data

CODE

```
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
fruits=pd.read_table('/content/fruit_data_with_colors.txt')
fruits.head()
```

OUTPUT

	fruit_label	fruit_name	fruit_subtype	mass	width	height	color_score
0	1	apple	granny_smith	192	8.4	7.3	0.55
1	1	apple	granny_smith	180	8.0	6.8	0.59
2	1	apple	granny_smith	176	7.4	7.2	0.60
3	2	mandarin	mandarin	86	6.2	4.7	0.80
4	2	mandarin	mandarin	84	6.0	4.6	0.79

CODE

```
fruits.shape
```

OUTPUT

(59, 7)

CODE

```
predct = dict(zip(fruits.fruit_label.unique(), fruits.fruit_name.unique()))
predct
```

OUTPUT

{1: 'apple', 2: 'mandarin', 3: 'orange', 4: 'lemon'}

CODE

```
fruits['fruit_name'].value_counts()
```

OUTPUT

```
apple       19
orange      19
lemon       16
mandarin     5
Name: fruit_name, dtype: int64
```

CODE

```
apple_data=fruits[fruits['fruit_name']=='apple']
orange_data=fruits[fruits['fruit_name']=='orange']
lemon_data=fruits[fruits['fruit_name']=='lemon']
mandarin_data=fruits[fruits['fruit_name']=='mandarin']
apple_data.head()
```

OUTPUT

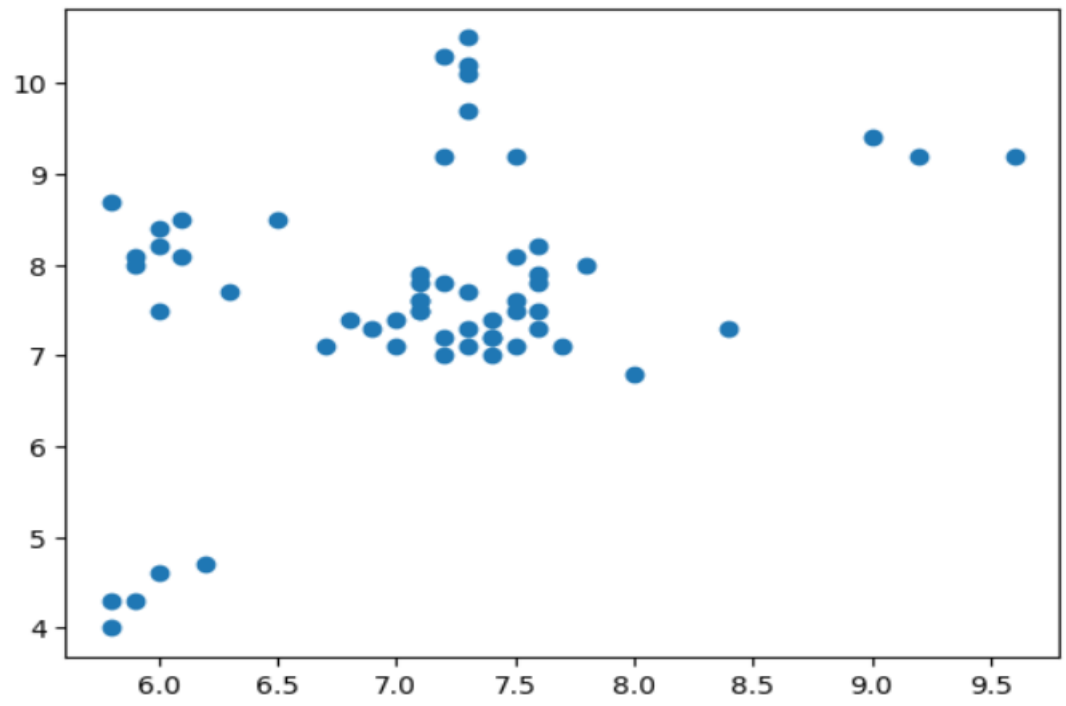
	fruit_label	fruit_name	fruit_subtype	mass	width	height	color_score
0	1	apple	granny_smith	192	8.4	7.3	0.55
1	1	apple	granny_smith	180	8.0	6.8	0.59
2	1	apple	granny_smith	176	7.4	7.2	0.60
8	1	apple	braeburn	178	7.1	7.8	0.92
9	1	apple	braeburn	172	7.4	7.0	0.89

CODE

```
plt.scatter(fruits['width'],fruits['height'])
```

OUTPUT

```
<matplotlib.collections.PathCollection at 0x7b0418bbed40>
```

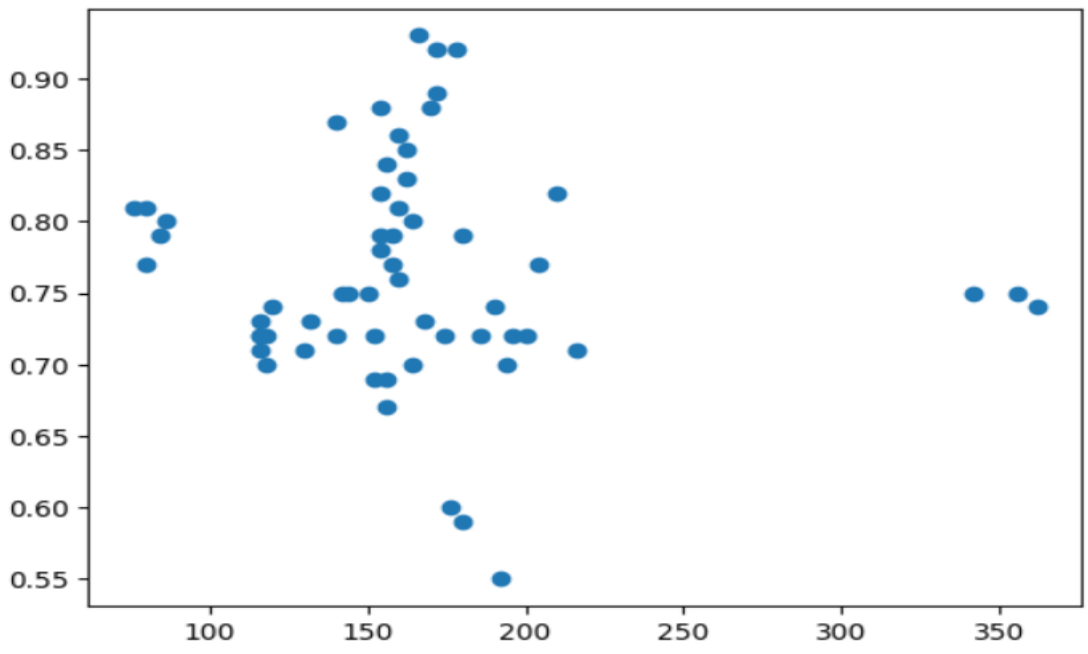


CODE

```
plt.scatter(fruits['mass'],fruits['color_score'])
```

OUTPUT

```
<matplotlib.collections.PathCollection at 0x7b0418b05d20>
```



CODE

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
X=fruits[['mass','width','height']]
Y=fruits['fruit_label']
X_train,X_test,y_train,y_test=train_test_split(X,Y,random_state=0)
X_train.describe()
```

OUTPUT

	mass	width	height
count	44.000000	44.000000	44.000000
mean	159.090909	7.038636	7.643182
std	53.316876	0.835886	1.370350
min	76.000000	5.800000	4.000000
25%	127.500000	6.175000	7.200000
50%	157.000000	7.200000	7.600000
75%	172.500000	7.500000	8.250000
max	356.000000	9.200000	10.500000

CODE

```
X_test.describe()
```

OUTPUT

	mass	width	height
count	15.000000	15.00000	15.000000
mean	174.933333	7.30000	7.840000
std	60.075508	0.75119	1.369463
min	84.000000	6.00000	4.600000
25%	146.000000	7.10000	7.250000
50%	166.000000	7.20000	7.600000
75%	185.000000	7.45000	8.150000
max	362.000000	9.60000	10.300000

CODE

```
knn=KNeighborsClassifier()
knn.fit(X_train,y_train)
```

OUTPUT

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

CODE

```
knn.score(X_test,y_test)
```

OUTPUT

0.5333333333333333

CODE

```
prediction1=knn.predict([[100,6.3,8]])
print([prediction1[0]])
print("--------------------")
predct[prediction1[0]]
```

OUTPUT

```
[4]
--------------------
'lemon'
```

CODE

```
prediction2=knn.predict([[300,7,10]])
predct[prediction2[0]]
```

OUTPUT

'orange'

EXPERIMENT NO :7

AIM: Classification using KNN

CODE

```
weather=['Sunny','Sunny','Overcast','Rainy','Rainy','Rainy','Overcast',
'Sunny','Sunny','Rainy','Sunny','Overcast','Overcast','Rainy']

# Second Feature
temp=['Hot','Hot','Hot','Mild','Cool','Cool','Cool','Mild','Cool','Mild',
'Mild','Mild','Hot','Mild']

# Label or target varible

play=['No','No','Yes','Yes','Yes','No','Yes','No','Yes','Yes','Yes','Yes',
'Yes','No']

# Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python

from sklearn import preprocessing
#creating labelEncoder
le = preprocessing.LabelEncoder()
# Converting string labels into numbers.
weather_encoded=le.fit_transform(weather)
print("Weather before conversion:",weather)
print("New Weather Codes:",weather_encoded)
```

OUTPUT

Weather before conversion: ['Sunny', 'Sunny', 'Overcast', 'Rainy', 'Rainy', 'Rainy', 'Overcast', 'Sunny', 'Sunny', 'Rainy', 'Sunny', 'Overcast', 'Overcast', 'Rainy']

New Weather Codes: [2 2 0 1 1 1 0 2 2 1 2 0 0 1]

CODE

```
temp_encoded=le.fit_transform(temp)
print("Temperature before conversion:",temp)
print("New Temp Codes:",temp_encoded)
print(" ")
label=le.fit_transform(play)
print("Play before conversion:",play)
print("New Play Codes:",label)
```

OUTPUT

Temperature before conversion: ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Cool', 'Mild', 'Cool', 'Mild', 'Mild', 'Mild', 'Hot', 'Mild']

New Temp Codes: [1 1 1 2 0 0 0 2 0 2 2 2 1 2]

Play before conversion: ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']
New Play Codes: [0 0 1 1 1 0 1 0 1 1 1 1 1 0]

CODE

```
features=list(zip(weather_encoded,temp_encoded))
print(features)
```

OUTPUT

[(2, 1), (2, 1), (0, 1), (1, 2), (1, 0), (1, 0), (0, 0), (2, 2), (2, 0), (1, 2), (2, 2), (0, 2), (0, 1), (1, 2)]

CODE

```
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors=3)
# Train the model using the training sets
model.fit(features,label)
predicted= model.predict([[0,1]]) # 0:Overcast, 1:Hot
print(predicted)
```

OUTPUT

[1]

CODE

```
#The following code creates a model with original "Play" codes
model1 = KNeighborsClassifier(n_neighbors=3)
model1.fit(features,play)
predicted1= model1.predict([[0,1]]) # 0:Overcast, 1:Hot
print(predicted1)
```

OUTPUT

['Yes']

EXPERIMENT NO :8

AIM : SVD

CODE

```
import numpy as np
from numpy import array
from scipy.linalg import svd

# define a matrix
A = array([[1, 2], [3, 4], [5, 6]])
print("Original Matrix")
print(A)
```

OUTPUT

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

CODE

```
# SVD
U, s, VT = svd(A)
print("Matrix- U")
print(U)
print("Matrix- s")
print(s)
print("Matrix- VT")
print(VT)
```

OUTPUT

```
Matrix- U
[[-0.2298477   0.88346102  0.40824829]
 [-0.52474482  0.24078249 -0.81649658]
 [-0.81964194 -0.40189603  0.40824829]]
Matrix- s
[9.52551809 0.51430058]
Matrix- VT
[[-0.61962948 -0.78489445]
 [-0.78489445  0.61962948]]
```

CODE

```
smat=np.zeros((3,2))
print(smat)
smat[:2,:2]=np.diag(s)
print(smat)
```

OUTPUT

```
[[0. 0.]
 [0. 0.]
 [0. 0.]]
[[9.52551809 0.        ]
 [0.         0.51430058]
 [0.         0.        ]]
```

CODE

```
from numpy.linalg import multi_dot
print(multi_dot([U,smat,VT]) ,"\n",A)
```

OUTPUT

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

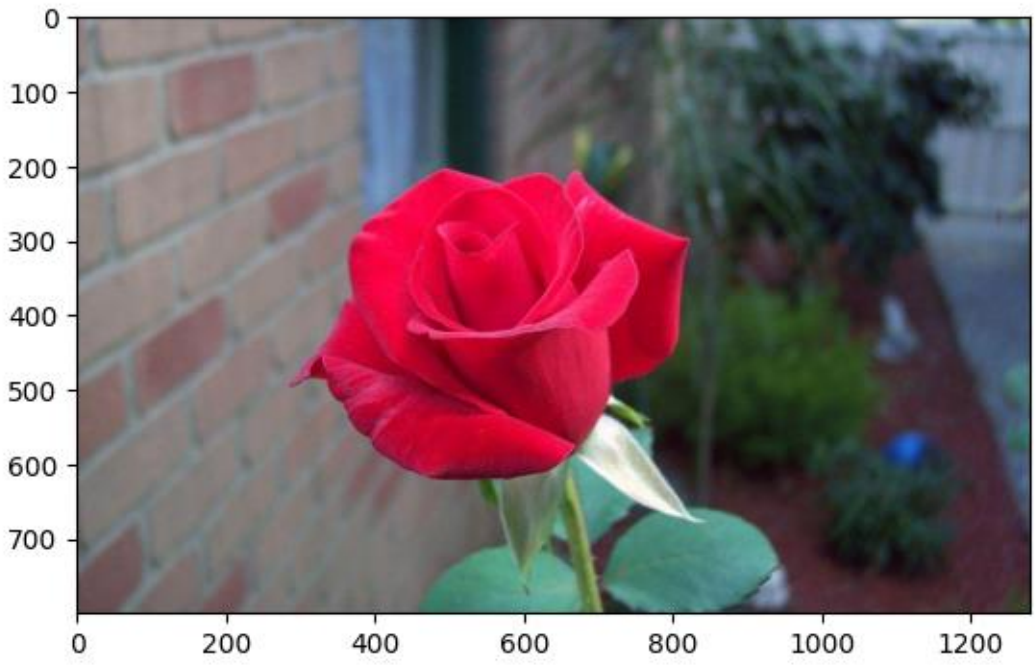
CODE

#SVD image compression

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

img_eg = mpimg.imread("rose.jpg")
plt.imshow(img_eg)
print(img_eg.shape) #Operation results: (800, 1280,3)
```

OUTPUT

```
(800, 1280, 3)
```

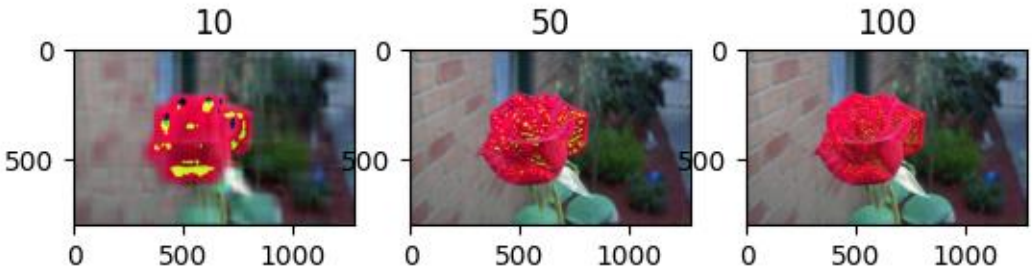


CODE

```
#Converting image data into two-dimensional matrix and singular value decomposition
img_temp = img_eg.reshape(800, 1280 * 3)
U,Sigma,VT = np.linalg.svd(img_temp)


# Take the first 10 singular values
sval_nums = 10
img_restruct1 = (U[:,0:sval_nums]).dot(np.diag(Sigma[0:sval_nums])).dot(VT[0:sval_nums,:])
img_restruct1 = img_restruct1.reshape(800, 1280,3)
img_restruct1.tolist()

# Take the first 50 singular values
sval_nums = 50
img_restruct2 = (U[:,0:sval_nums]).dot(np.diag(Sigma[0:sval_nums])).dot(VT[0:sval_nums,:])
img_restruct2 = img_restruct2.reshape(800, 1280,3)

# Take the first 100 singular values
sval_nums = 100
img_restruct3 = (U[:,0:sval_nums]).dot(np.diag(Sigma[0:sval_nums])).dot(VT[0:sval_nums,:])
img_restruct3 = img_restruct3.reshape(800, 1280,3)

#Exhibition
fig, ax = plt.subplots(nrows=1, ncols=3)
ax[0].imshow(img_restruct1.astype(np.uint8))
ax[0].set(title = "10")
ax[1].imshow(img_restruct2.astype(np.uint8))
ax[1].set(title = "50")
ax[2].imshow(img_restruct3.astype(np.uint8))
```

```
ax[2].set(title = "100")
plt.show()
```

OUTPUT



EXPERIMENT NO :9

AIM: Data Preprocess

CODE

```
import pandas as pd
import matplotlib.pyplot as plt
dataset=pd.read_csv("Data.csv")
print(dataset.head)
```

OUTPUT

```
<bound method NDFrame.head of       Country   Age   Salary Purchased
0       India  34.0  92000.0       Yes
1   Sri lanka  22.0  25000.0       Yes
2       China  31.0  74000.0       Yes
3   Sri lanka  29.0      NaN       No
4       China  55.0  98000.0       Yes
5       India  24.0  30000.0       No
6   Sri lanka  28.0  40000.0       No
7       India   NaN  60000.0       No
8       China  51.0  89000.0       Yes
9       India  44.0  78000.0       Yes
10  Sri lanka  21.0  20000.0       No
11      China  25.0  30000.0       Yes
12      India  33.0  45000.0       Yes
13      India  42.0  65000.0       Yes
14  Sri lanka  33.0  22000.0       No>
```

CODE

```
print(dataset.isna().sum())
```

OUTPUT

```
Country      0
Age          1
Salary       1
Purchased    0
dtype: int64
```

CODE

```
#Handling missing values
dataset['Age'].fillna(dataset['Age'].median(),inplace=True)
```

```
dataset['Salary'].fillna(dataset['Salary'].median(),inplace=True)
print(dataset.isna().sum())
```

OUTPUT

```
Country      0
Age          0
Salary       0
Purchased    0
dtype: int64
```

CODE

```
dataset1=pd.read_csv("dirtydata.csv")
dataset1.head()
```

OUTPUT

	Duration	Date	Pulse	Maxpulse	Calories
0	60	'2020/12/01'	110	130	409.1
1	60	'2020/12/02'	117	145	479.0
2	60	'2020/12/03'	103	135	340.0
3	45	'2020/12/04'	109	175	282.4
4	45	'2020/12/05'	117	148	406.0

CODE

```
print (dataset1)
```

OUTPUT

```
   Duration         Date  Pulse  Maxpulse  Calories
0        60  '2020/12/01'    110       130     409.1
1        60  '2020/12/02'    117       145     479.0
2        60  '2020/12/03'    103       135     340.0
3        45  '2020/12/04'    109       175     282.4
4        45  '2020/12/05'    117       148     406.0
5        60  '2020/12/06'    102       127     300.0
6        60  '2020/12/07'    110       136     374.0
7       450  '2020/12/08'    104       134     253.3

8        30  '2020/12/09'    109       133     195.1
9        60  '2020/12/10'     98       124     269.0
```

```
10       60  '2020/12/11'    103      147    329.3
11       60  '2020/12/12'    100      120    250.7
12       60  '2020/12/12'    100      120    250.7
13       60  '2020/12/13'    106      128    345.3
14       60  '2020/12/14'    104      132    379.3
15       60  '2020/12/15'     98      123    275.0
16       60  '2020/12/16'     98      120    215.2
17       60  '2020/12/17'    100      120    300.0
18       45  '2020/12/18'     90      112      NaN
19       60  '2020/12/19'    103      123    323.0
20       45  '2020/12/20'     97      125    243.0
21       60  '2020/12/21'    108      131    364.2
22       45           NaN    100      119    282.0
23       60  '2020/12/23'    130      101    300.0
24       45  '2020/12/24'    105      132    246.0
25       60  '2020/12/25'    102      126    334.5
26       60      20201226    100      120    250.0
27       60  '2020/12/27'     92      118    241.0
28       60  '2020/12/28'    103      132      NaN
29       60  '2020/12/29'    100      132    280.0
30       60  '2020/12/30'    102      129    380.3
31       60  '2020/12/31'     92      115    243.0
```

CODE

```
#to locate where is the missing value
dataset1[dataset1.isna().any(axis=1)]
```

OUTPUT

	Duration	Date	Pulse	Maxpulse	Calories
18	45	'2020/12/18'	90	112	NaN
22	45	NaN	100	119	282.0
28	60	'2020/12/28'	103	132	NaN

CODE

```
#drop the missing value rows
dropped_dataset=dataset1.dropna()
print(dropped_dataset.isna().sum())
```

OUTPUT

```
Duration    0
Date        0
Pulse       0
Maxpulse    0
Calories    0
dtype: int64
```

CODE

```
#removed missing value rows
print (dropped_dataset)
```

OUTPUT

	Duration	Date	Pulse	Maxpulse	Calories
0	60	'2020/12/01'	110	130	409.1
1	60	'2020/12/02'	117	145	479.0
2	60	'2020/12/03'	103	135	340.0
3	45	'2020/12/04'	109	175	282.4
4	45	'2020/12/05'	117	148	406.0
5	60	'2020/12/06'	102	127	300.0
6	60	'2020/12/07'	110	136	374.0
7	450	'2020/12/08'	104	134	253.3
8	30	'2020/12/09'	109	133	195.1
9	60	'2020/12/10'	98	124	269.0
10	60	'2020/12/11'	103	147	329.3
11	60	'2020/12/12'	100	120	250.7
12	60	'2020/12/12'	100	120	250.7
13	60	'2020/12/13'	106	128	345.3
14	60	'2020/12/14'	104	132	379.3
15	60	'2020/12/15'	98	123	275.0
16	60	'2020/12/16'	98	120	215.2
17	60	'2020/12/17'	100	120	300.0
19	60	'2020/12/19'	103	123	323.0
20	45	'2020/12/20'	97	125	243.0
21	60	'2020/12/21'	108	131	364.2
23	60	'2020/12/23'	130	101	300.0
24	45	'2020/12/24'	105	132	246.0
25	60	'2020/12/25'	102	126	334.5
26	60	20201226	100	120	250.0
27	60	'2020/12/27'	92	118	241.0
29	60	'2020/12/29'	100	132	280.0
30	60	'2020/12/30'	102	129	380.3
31	60	'2020/12/31'	92	115	243.0

EXPERIMENT NO :10

AIM: Classification using Naïve Bayes with iris dataset

CODE

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

df = pd.read_csv("iris.csv")
X = df.iloc[:,:4].values
y = df['variety'].values
df.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

CODE

```
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.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)
print(X_train)
```

OUTPUT

```
[[-1.50191773  1.17911545 -1.54367866 -1.29120907]
 [-0.51435539  1.83824831 -1.3699062  -1.02267339]
 [-0.02057422 -0.57857218  0.83121159  1.66268343]
 [-0.76124597  2.27767021 -1.2540579  -1.42547691]
```

```
[ 1.21387871 -0.57857218  0.65743913  0.32000502]
[-1.50191773  0.30027164 -1.31198205 -1.29120907]
[ 2.32488634 -0.13915027  1.41045311  1.52841559]
[-0.76124597 -0.79828313  0.13612176  0.32000502]
[ 1.09043342  0.51998259  1.1787565   1.79695127]
[-0.2674648  -0.13915027  0.25197006  0.18573718]
[ 0.34976166 -0.13915027  0.54159082  0.32000502]
[ 0.96698813 -0.13915027  0.42574252  0.32000502]
[-0.88469126  0.51998259 -1.13820959 -0.88840555]
[ 1.95455047 -0.57857218  1.41045311  0.99134422]
[-0.88469126  1.61853735 -1.19613375 -1.29120907]
[ 1.337324    0.08056068  1.00498404  1.2598799 ]
[-1.00813656  0.73969354 -1.19613375 -1.02267339]
[-0.88469126  1.3988264  -1.2540579  -1.02267339]
[ 1.21387871 -0.13915027  1.0629082   1.2598799 ]
[-0.39091009  0.95940449 -1.3699062  -1.29120907]
[ 1.337324    0.08056068  0.83121159  1.52841559]
[ 1.337324    0.08056068  0.71536328  0.45427286]
[ 1.4607693   0.30027164  0.59951498  0.32000502]
[-1.13158185  0.08056068 -1.2540579  -1.42547691]
[ 0.47320695 -1.89683789  0.48366667  0.45427286]
[-0.39091009 -1.45741599  0.02027345 -0.21706634]
[-0.39091009 -1.01799408  0.42574252  0.05146934]
[ 0.34976166 -0.57857218  0.19404591  0.18573718]
[ 1.21387871  0.30027164  1.2946048   1.52841559]
[-1.13158185 -0.13915027 -1.31198205 -1.29120907]
[-0.88469126 -1.23770503 -0.38519561 -0.0827985 ]
[ 2.20144105 -0.13915027  1.70007387  1.2598799 ]
[-1.37847243  0.30027164 -1.19613375 -1.29120907]
[-0.14401951 -0.13915027  0.30989421  0.05146934]
```

```
[-1.00813656 -1.67712694 -0.21142316 -0.21706634]
[ 0.59665225 -1.67712694  0.42574252  0.18573718]
[ 0.10287108  0.30027164  0.65743913  0.85707638]
[ 1.09043342  0.08056068  0.59951498  0.45427286]
[ 1.70765988  0.30027164  1.35252896  0.85707638]
[-0.14401951  2.93680307 -1.2540579  -1.02267339]
[-0.39091009 -1.23770503  0.19404591  0.18573718]
[-0.14401951 -0.57857218  0.25197006  0.18573718]
[ 0.84354283 -0.13915027  1.23668065  1.39414775]
[-1.13158185 -1.23770503  0.48366667  0.72280854]
[ 0.10287108 -0.13915027  0.30989421  0.45427286]
[ 0.84354283  0.30027164  0.83121159  1.12561206]
[-1.25502714  0.73969354 -1.19613375 -1.29120907]
[ 1.09043342 -0.13915027  0.88913574  1.52841559]
[ 0.47320695 -0.35886122  0.36781837  0.18573718]
[ 0.22631637  0.73969354  0.48366667  0.5885407 ]
[ 0.22631637 -0.13915027  0.65743913  0.85707638]
[ 0.22631637 -1.89683789  0.19404591 -0.21706634]
[-0.76124597  0.95940449 -1.2540579  -1.29120907]
[ 1.83110517 -0.35886122  1.52630141  0.85707638]
[-0.14401951  1.61853735 -1.13820959 -1.15694123]
[ 0.59665225 -0.57857218  0.83121159  0.45427286]
[-0.39091009  2.49738116 -1.31198205 -1.29120907]
[-0.51435539  1.3988264  -1.2540579  -1.29120907]
[ 0.59665225 -0.35886122  1.12083235  0.85707638]
[-0.39091009 -1.67712694  0.19404591  0.18573718]
[-1.25502714  0.73969354 -1.02236129 -1.29120907]
[-1.00813656 -2.3362598  -0.09557485 -0.21706634]
[-0.51435539 -0.13915027  0.48366667  0.45427286]
[ 1.70765988  1.17911545  1.41045311  1.79695127]
```

```
[-0.51435539  0.73969354 -1.2540579  -1.02267339]
[-0.88469126  1.61853735 -1.2540579  -1.15694123]
[ 1.337324    0.30027164  1.1787565   1.52841559]
[-1.50191773  0.73969354 -1.31198205 -1.15694123]
[-0.88469126  0.95940449 -1.31198205 -1.15694123]
[ 0.59665225 -1.23770503  0.71536328  0.45427286]
[ 1.09043342  0.51998259  1.1787565   1.2598799 ]
[-0.02057422 -1.01799408  0.19404591  0.05146934]
[ 1.70765988 -0.13915027  1.23668065  0.5885407 ]
[ 2.57177693  1.61853735  1.58422557  1.12561206]
[ 0.72009754 -0.35886122  0.36781837  0.18573718]
[-1.74880831  0.30027164 -1.3699062  -1.29120907]
[ 0.22631637 -1.89683789  0.77328743  0.45427286]
[-1.00813656  0.51998259 -1.31198205 -1.29120907]
[-1.13158185 -1.45741599 -0.21142316 -0.21706634]
[ 0.34976166 -1.01799408  1.12083235  0.32000502]
[-1.25502714 -0.13915027 -1.31198205 -1.42547691]
[-1.74880831 -0.13915027 -1.3699062  -1.29120907]
[-1.00813656  0.95940449 -1.3699062  -1.15694123]
[-1.00813656  0.73969354 -1.2540579  -1.29120907]
[ 0.47320695 -0.57857218  0.65743913  0.85707638]
[-1.25502714  0.08056068 -1.19613375 -1.29120907]
[ 0.96698813 -0.35886122  0.54159082  0.18573718]
[-0.76124597  0.73969354 -1.31198205 -1.29120907]
[-0.63780068  1.3988264  -1.2540579  -1.29120907]
[ 0.72009754  0.30027164  0.94705989  1.52841559]
[ 0.34976166 -0.35886122  0.59951498  0.32000502]
[-0.02057422 -0.79828313  0.13612176  0.05146934]
[-1.62536302 -1.67712694 -1.3699062  -1.15694123]
[-1.50191773  0.08056068 -1.2540579  -1.29120907]
```

```
[ 0.84354283 -0.57857218  0.54159082  0.45427286]
[-0.14401951 -0.57857218  0.48366667  0.18573718]
[ 0.22631637 -0.79828313  0.83121159  0.5885407 ]
[ 1.09043342  0.08056068  1.12083235  1.66268343]
[-0.88469126  1.61853735 -1.02236129 -1.02267339]
[-1.8722536  -0.13915027 -1.48575451 -1.42547691]
[-0.51435539  0.73969354 -1.13820959 -1.29120907]
[ 0.59665225 -0.79828313  0.71536328  0.85707638]
[-0.14401951 -0.35886122  0.30989421  0.18573718]
[-0.02057422 -0.79828313  0.83121159  0.99134422]
[-0.88469126  0.73969354 -1.2540579  -1.29120907]
[ 0.59665225  0.51998259  0.59951498  0.5885407 ]
[-0.2674648  -0.57857218  0.71536328  1.12561206]
[ 0.72009754 -0.79828313  0.94705989  0.99134422]
[ 0.72009754 -0.57857218  1.12083235  1.2598799 ]
[ 0.72009754  0.30027164  0.48366667  0.45427286]
[-0.2674648  -0.79828313  0.30989421  0.18573718]
[-1.25502714 -0.13915027 -1.31198205 -1.15694123]
[-0.02057422 -0.79828313  0.83121159  0.99134422]
[ 1.09043342 -1.23770503  1.23668065  0.85707638]
[-0.02057422 -0.79828313  0.25197006 -0.21706634]
[ 0.59665225 -1.23770503  0.77328743  0.99134422]
[-0.51435539  1.83824831 -1.13820959 -1.02267339]
[-1.00813656  0.95940449 -1.19613375 -0.7541377 ]
[-0.02057422  2.05795926 -1.42783035 -1.29120907]
[ 0.84354283 -0.13915027  0.88913574  1.12561206]]
```

CODE

```
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
```

```
classifier.fit(X_train, y_train)
```

OUTPUT

```
▾ GaussianNB
GaussianNB()
```

CODE

```
classifier.score(X_test,y_test)
```

OUTPUT

0.9

CODE

```
y_pred = classifier.predict(X_test)

y_pred
```

OUTPUT

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

CODE

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

OUTPUT

```
[[ 8  0  0]
 [ 0  9  0]
 [ 0  3 10]]
              precision    recall  f1-score   support

      Setosa       1.00      1.00      1.00         8
  Versicolor       0.75      1.00      0.86         9
   Virginica       1.00      0.77      0.87        13

    accuracy                           0.90        30
   macro avg       0.92      0.92      0.91        30
weighted avg       0.93      0.90      0.90        30
```

CODE

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

OUTPUT

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

17	Versicolor	Versicolor
18	Versicolor	Versicolor
19	Setosa	Setosa
20	Virginica	Versicolor
21	Virginica	Virginica
22	Setosa	Setosa
23	Virginica	Virginica
24	Virginica	Versicolor
25	Setosa	Setosa
26	Versicolor	Versicolor
27	Setosa	Setosa
28	Virginica	Virginica
29	Versicolor	Versicolor

EXPERIMENT NO :11

AIM: Classification using Naïve bayes with social network ads data

CODE

```
import pandas as pd
dataset = pd.read_csv("Social_Network_Ads.csv")
print(dataset.describe())
```

OUTPUT

```
             User ID         Age  EstimatedSalary   Purchased
count  4.000000e+02  400.000000       400.000000  400.000000
mean   1.569154e+07   37.655000     69742.500000    0.357500
std    7.165832e+04   10.482877     34096.960282    0.479864
min    1.556669e+07   18.000000     15000.000000    0.000000
25%    1.562676e+07   29.750000     43000.000000    0.000000
50%    1.569434e+07   37.000000     70000.000000    0.000000
75%    1.575036e+07   46.000000     88000.000000    1.000000
max    1.581524e+07   60.000000    150000.000000    1.000000
```

CODE

```
print(dataset.head(5))
```

OUTPUT

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

CODE

```
X = dataset.iloc[:, [1, 2, 3]].values
y = dataset.iloc[:, -1].values
X
```

```
y
```

OUTPUT

```
array([['Male', 19, 19000],
       ['Male', 35, 20000],
       ['Female', 26, 43000],
       ...,
       ['Female', 50, 20000],
       ['Male', 36, 33000],
       ['Female', 49, 36000]], dtype=object)
```

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

CODE

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
X[:,0] = le.fit_transform(X[:,0])
X
```

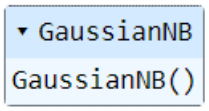
OUTPUT

```
array([[1, 19, 19000],
       [1, 35, 20000],
       [0, 26, 43000],
       ...,
       [0, 50, 20000],
       [1, 36, 33000],
       [0, 49, 36000]], dtype=object)
```

CODE

```
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, random_state = 0)
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)
```

OUTPUT



CODE

```
y_pred  =  classifier.predict(X_test)
y_pred
```

OUTPUT

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

CODE

```
y_test
```

OUTPUT

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

CODE

```
# Method 1 to print accuracy
classifier.score(X_test, y_test)
```

OUTPUT

0.925

CODE

```
from sklearn.metrics import confusion_matrix,accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

OUTPUT

```
[[56  2]
 [ 4 18]]
```

CODE

```
# Method 2 to print accuracy
ac = accuracy_score(y_test,y_pred)
print(ac)
```

OUTPUT

0.925

CODE

```
# Testing accuracy by changing train-test ratio 70:30
X_train1, X_test1, y_train1, y_test1 = train_test_split(X, y, test_size = 0.30, random_state = 0)
classifier.fit(X_train1, y_train1)
classifier.score(X_test1, y_test1)
```

OUTPUT

0.8916666666666667

EXPERIMENT NO : 12

AIM :Decision tree using iris dataset

CODE

```
import numpy as np
import pandas as pd
#import matplotlib.pyplot as plt
df = pd.read_csv("iris.csv")
X = df.drop('variety', axis=1)
y = df['variety']
print(X.shape,y.shape)
```

OUTPUT

(150, 4) (150,)

CODE

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X , y, test_size = 25, random_state = 10)
from sklearn.tree import DecisionTreeClassifier
clf=DecisionTreeClassifier()
clf.fit(X_train,y_train)
#Printing Accuracy
print ("Accuracy: ",clf.score(X_test, y_test))
```

OUTPUT

Accuracy: 0.96

CODE

```
# Printing detailed result using classification Report generation
from sklearn.metrics import classification_report
```

```
y_pred =clf.predict(X_test)
print("Classification report - \n", classification_report(y_test,y_pred))
```

OUTPUT

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

      Setosa       1.00      1.00      1.00         9
  Versicolor       1.00      0.90      0.95        10
   Virginica       0.86      1.00      0.92         6

    accuracy                           0.96        25
   macro avg       0.95      0.97      0.96        25
weighted avg       0.97      0.96      0.96        25
```

CODE

```
# Printing confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

OUTPUT

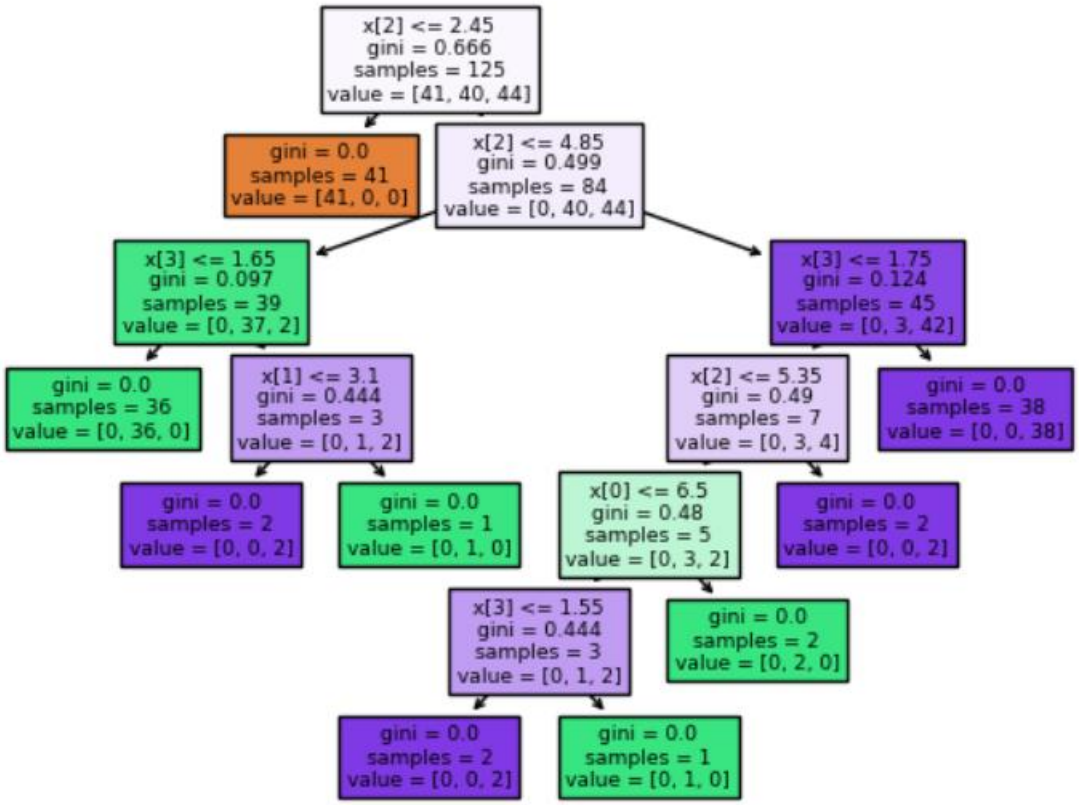
```
[[9 0 0]
 [0 9 1]
 [0 0 6]]
```

CODE

```
from sklearn import tree
tree.plot_tree(clf,filled=True)
```

OUTPUT

[Text(0.4, 0.9285714285714286, 'x[2] <= 2.45\ngini = 0.666\nsamples = 125\nvalue = [41, 40, 44]'),
Text(0.3, 0.7857142857142857, 'gini = 0.0\nsamples = 41\nvalue = [41, 0, 0]'),
Text(0.5, 0.7857142857142857, 'x[2] <= 4.85\ngini = 0.499\nsamples = 84\nvalue = [0, 40, 44]'),
Text(0.2, 0.6428571428571429, 'x[3] <= 1.65\ngini = 0.097\nsamples = 39\nvalue = [0, 37, 2]'),
Text(0.1, 0.5, 'gini = 0.0\nsamples = 36\nvalue = [0, 36, 0]'),
Text(0.3, 0.5, 'x[1] <= 3.1\ngini = 0.444\nsamples = 3\nvalue = [0, 1, 2]'),
Text(0.2, 0.35714285714285715, 'gini = 0.0\nsamples = 2\nvalue = [0, 0, 2]'),
Text(0.4, 0.35714285714285715, 'gini = 0.0\nsamples = 1\nvalue = [0, 1, 0]'),
Text(0.8, 0.6428571428571429, 'x[3] <= 1.75\ngini = 0.124\nsamples = 45\nvalue = [0, 3, 42]'),
Text(0.7, 0.5, 'x[2] <= 5.35\ngini = 0.49\nsamples = 7\nvalue = [0, 3, 4]'),
Text(0.6, 0.35714285714285715, 'x[1] <= 2.9\ngini = 0.48\nsamples = 5\nvalue = [0, 3, 2]'),
Text(0.5, 0.21428571428571427, 'x[3] <= 1.55\ngini = 0.444\nsamples = 3\nvalue = [0, 1, 2]'),
Text(0.4, 0.07142857142857142, 'gini = 0.0\nsamples = 2\nvalue = [0, 0, 2]'),
Text(0.6, 0.07142857142857142, 'gini = 0.0\nsamples = 1\nvalue = [0, 1, 0]'),
Text(0.7, 0.21428571428571427, 'gini = 0.0\nsamples = 2\nvalue = [0, 2, 0]'),
Text(0.8, 0.35714285714285715, 'gini = 0.0\nsamples = 2\nvalue = [0, 0, 2]'),
Text(0.9, 0.5, 'gini = 0.0\nsamples = 38\nvalue = [0, 0, 38]')]


x[2] <= 2.45
gini = 0.666
samples = 125
value = [41, 40, 44]
gini = 0.0
samples = 41
value = [41, 0, 0]
x[2] <= 4.85
gini = 0.499
samples = 84
value = [0, 40, 44]
x[3] <= 1.65
gini = 0.097
samples = 39
value = [0, 37, 2]
x[3] <= 1.75
gini = 0.124
samples = 45
value = [0, 3, 42]
gini = 0.0
samples = 36
value = [0, 36, 0]
x[1] <= 3.1
gini = 0.444
samples = 3
value = [0, 1, 2]
x[2] <= 5.35
gini = 0.49
samples = 7
value = [0, 3, 4]
gini = 0.0
samples = 38
value = [0, 0, 38]
gini = 0.0
samples = 2
value = [0, 0, 2]
gini = 0.0
samples = 1
value = [0, 1, 0]
x[0] <= 6.5
gini = 0.48
samples = 5
value = [0, 3, 2]
gini = 0.0
samples = 2
value = [0, 0, 2]
x[3] <= 1.55
gini = 0.444
samples = 3
value = [0, 1, 2]
gini = 0.0
samples = 2
value = [0, 2, 0]
gini = 0.0
samples = 2
value = [0, 0, 2]
gini = 0.0
samples = 1
value = [0, 1, 0]

EXPERIMENT NO :13

AIM: Calculate the Entropy and Gini Metrics of a model

CODE

```
import pandas as pd
import numpy as np
```

```
lst = ['apple']*3 + ['orange']*2 + ['banana']*2
fruits = pd.Series(lst)
print(fruits)
```

OUTPUT

```
0     apple
1     apple
2     apple
3    orange
4    orange
5    banana
6    banana
dtype: object
```

CODE

```
probs = fruits.value_counts(normalize=True)
probs
```

OUTPUT

```
apple     0.428571
orange    0.285714
banana    0.285714
dtype: float64
```

CODE

```
probs_by_hand = [3/7, 2/7, 2/7]
print(probs_by_hand)
```

OUTPUT

```
[0.42857142857142855, 0.2857142857142857, 0.2857142857142857]
```

CODE

```
entropy = -1 * np.sum(np.log2(probs) * probs)
entropy
```

OUTPUT

```
1.5566567074628228
```

CODE

```
gini_index = 1 - np.sum(np.square(probs))
gini_index
```

OUTPUT

```
0.653061224489796
```

CODE

```
lst2 = ['apple', 'orange', 'banana', 'mango', 'blueberry', 'watermelon', 'pear']
fruits2 = pd.Series(lst2)
print(fruits2)
probs2 = fruits2.value_counts(normalize=True)
probs2
```

OUTPUT

```
0         apple
1        orange
2        banana
3         mango
4     blueberry
5    watermelon
6          pear
dtype: object
apple         0.142857
orange        0.142857
banana        0.142857
mango         0.142857
blueberry     0.142857
watermelon    0.142857
pear          0.142857
dtype: float64
```

CODE

```
entropy = -1 * np.sum(np.log2(probs2) * probs2)
entropy
```

OUTPUT

```
2.807354922057604
```

CODE

```
gini_index = 1 - np.sum(np.square(probs2))
gini_index
```

OUTPUT

```
0.8571428571428572
```

EXPERIMENT NO :14

AIM : Decision tree using titanic dataset

CODE

```
import pandas as pd
#df = pd.read_csv('titanic.csv', index_col='PassengerId')
df = pd.read_csv('titanic.csv')
print(df.head(2))
```

OUTPUT

```
PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1

   Parch     Ticket     Fare Cabin Embarked
0      0  A/5 21171   7.2500   NaN        S
1      0   PC 17599  71.2833   C85        C
```

CODE

```
df.shape
```

OUTPUT

```
(891, 12)
```

CODE

```
df = df[['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Survived']]
df.shape
```

OUTPUT

```
(891, 7)
```

CODE

```
df['Sex'] = df['Sex'].map({'male': 0, 'female': 1})
df = df.dropna()
df.shape
```

OUTPUT

(714, 7)

CODE

```
X = df.drop('Survived', axis=1)
y = df['Survived']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)

from sklearn import tree
model = tree.DecisionTreeClassifier()
model.fit(X_train, y_train)
model.score(X_test, y_test)
```

OUTPUT

0.8156424581005587

CODE

```
from sklearn.metrics import accuracy_score
y_predict = model.predict(X_test)
print("Accuracy:",accuracy_score(y_test, y_predict) )
```

OUTPUT

Accuracy: 0.8156424581005587

CODE

```
from sklearn.metrics import confusion_matrix
pd.DataFrame(
    confusion_matrix(y_test, y_predict),
    columns=['Predicted Not Survival', 'Predicted Survival'],
    index=['True Not Survival', 'True Survival']
```

OUTPUT

	Predicted Not Survival	Predicted Survival
True Not Survival	97	15
True Survival	18	49

CODE

```
from sklearn import tree
tree.plot_tree(model,filled=True)
```

OUTPUT

```
[Text(0.5014987088047221, 0.9761904761904762, 'x[1] <= 0.5\ngini = 0.486\nsamples =
535\nvalue = [312, 223]'),
 Text(0.18081345302508608, 0.9285714285714286, 'x[0] <= 1.5\ngini = 0.331\nsamples =
335\nvalue = [265, 70]'),
 Text(0.09149040826364978, 0.8809523809523809, 'x[2] <= 36.5\ngini = 0.481\nsamples =
77\nvalue = [46, 31]'),
 Text(0.023610427939006393, 0.8333333333333334, 'x[5] <= 37.812\ngini = 0.475\nsamples =
31\nvalue = [12, 19]'),
 Text(0.01574028529267093, 0.7857142857142857, 'gini = 0.0\nsamples = 7\nvalue = [0, 7]'),
 Text(0.03148057058534186, 0.7857142857142857, 'x[2] <= 17.5\ngini = 0.5\nsamples = 24\nvalue
= [12, 12]'),
 Text(0.023610427939006393, 0.7380952380952381, 'gini = 0.0\nsamples = 4\nvalue = [0, 4]'),
 Text(0.039350713231677326, 0.7380952380952381, 'x[5] <= 379.925\ngini = 0.48\nsamples =
20\nvalue = [12, 8]'),
 Text(0.03148057058534186, 0.6904761904761905, 'x[5] <= 77.008\ngini = 0.444\nsamples =
18\nvalue = [12, 6]'),
 Text(0.01574028529267093, 0.6428571428571429, 'x[5] <= 51.798\ngini = 0.5\nsamples =
10\nvalue = [5, 5]'),
 Text(0.007870142646335464, 0.5952380952380952, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
 Text(0.023610427939006393, 0.5952380952380952, 'x[2] <= 21.0\ngini = 0.408\nsamples =
7\nvalue = [2, 5]'),
 Text(0.01574028529267093, 0.5476190476190477, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.03148057058534186, 0.5476190476190477, 'x[2] <= 28.0\ngini = 0.278\nsamples =
6\nvalue = [1, 5]'),
 Text(0.023610427939006393, 0.5, 'gini = 0.0\nsamples = 4\nvalue = [0, 4]'),
 Text(0.039350713231677326, 0.5, 'x[2] <= 30.0\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
 Text(0.03148057058534186, 0.4523809523809524, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.04722085587801279, 0.4523809523809524, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.04722085587801279, 0.6428571428571429, 'x[4] <= 1.5\ngini = 0.219\nsamples = 8\nvalue
= [7, 1]'),
 Text(0.039350713231677326, 0.5952380952380952, 'gini = 0.0\nsamples = 6\nvalue = [6, 0]'),
 Text(0.055090998524348254, 0.5952380952380952, 'x[3] <= 0.5\ngini = 0.5\nsamples = 2\nvalue =
[1, 1]'),
 Text(0.04722085587801279, 0.5476190476190477, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.06296114117068372, 0.5476190476190477, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.04722085587801279, 0.6904761904761905, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.15937038858829317, 0.8333333333333334, 'x[2] <= 75.5\ngini = 0.386\nsamples =
46\nvalue = [34, 12]'),
 Text(0.15150024594195768, 0.7857142857142857, 'x[2] <= 53.0\ngini = 0.369\nsamples =
45\nvalue = [34, 11]'),
 Text(0.12985735366453516, 0.7380952380952381, 'x[2] <= 47.5\ngini = 0.43\nsamples =
32\nvalue = [22, 10]'),
 Text(0.11018199704869651, 0.6904761904761905, 'x[2] <= 43.0\ngini = 0.308\nsamples =
21\nvalue = [17, 4]'),
```

```
 Text(0.10231185440236104, 0.6428571428571429, 'x[2] <= 41.0\ngini = 0.444\nsamples =
12\nvalue = [8, 4]'),
 Text(0.08657156910969012, 0.5952380952380952, 'x[5] <= 30.35\ngini = 0.346\nsamples =
9\nvalue = [7, 2]'),
 Text(0.07870142646335465, 0.5476190476190477, 'gini = 0.0\nsamples = 5\nvalue = [5, 0]'),
 Text(0.09444171175602557, 0.5476190476190477, 'x[5] <= 52.827\ngini = 0.5\nsamples =
4\nvalue = [2, 2]'),
 Text(0.08657156910969012, 0.5, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.10231185440236104, 0.5, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.11805213969503198, 0.5952380952380952, 'x[3] <= 0.5\ngini = 0.444\nsamples = 3\nvalue
= [1, 2]'),
 Text(0.11018199704869651, 0.5476190476190477, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.12592228234136743, 0.5476190476190477, 'x[5] <= 52.277\ngini = 0.5\nsamples =
2\nvalue = [1, 1]'),
 Text(0.11805213969503198, 0.5, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.1337924249877029, 0.5, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.11805213969503198, 0.6428571428571429, 'gini = 0.0\nsamples = 9\nvalue = [9, 0]'),
 Text(0.14953271028037382, 0.6904761904761905, 'x[5] <= 53.95\ngini = 0.496\nsamples =
11\nvalue = [5, 6]'),
 Text(0.14166256763403837, 0.6428571428571429, 'gini = 0.0\nsamples = 4\nvalue = [0, 4]'),
 Text(0.1574028529267093, 0.6428571428571429, 'x[5] <= 122.267\ngini = 0.408\nsamples =
7\nvalue = [5, 2]'),
 Text(0.14953271028037382, 0.5952380952380952, 'x[2] <= 48.5\ngini = 0.278\nsamples =
6\nvalue = [5, 1]'),
 Text(0.14166256763403837, 0.5476190476190477, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.1574028529267093, 0.5476190476190477, 'gini = 0.0\nsamples = 5\nvalue = [5, 0]'),
 Text(0.16527299557304476, 0.5952380952380952, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.17314313821938024, 0.7380952380952381, 'x[5] <= 78.244\ngini = 0.142\nsamples =
13\nvalue = [12, 1]'),
 Text(0.16527299557304476, 0.6904761904761905, 'gini = 0.0\nsamples = 10\nvalue = [10, 0]'),
 Text(0.1810132808657157, 0.6904761904761905, 'x[4] <= 1.5\ngini = 0.444\nsamples = 3\nvalue =
[2, 1]'),
 Text(0.17314313821938024, 0.6428571428571429, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.18888342351205115, 0.6428571428571429, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.16724053123462862, 0.7857142857142857, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.2701364977865224, 0.8809523809523809, 'x[2] <= 9.5\ngini = 0.257\nsamples =
258\nvalue = [219, 39]'),
 Text(0.20462370880472208, 0.8333333333333334, 'x[3] <= 2.5\ngini = 0.499\nsamples =
19\nvalue = [9, 10]'),
 Text(0.19675356615838663, 0.7857142857142857, 'gini = 0.0\nsamples = 9\nvalue = [0, 9]'),
 Text(0.21249385145105756, 0.7857142857142857, 'x[4] <= 1.5\ngini = 0.18\nsamples = 10\nvalue
= [9, 1]'),
 Text(0.20462370880472208, 0.7380952380952381, 'gini = 0.0\nsamples = 6\nvalue = [6, 0]'),
 Text(0.22036399409739302, 0.7380952380952381, 'x[2] <= 3.5\ngini = 0.375\nsamples = 4\nvalue
= [3, 1]'),
 Text(0.21249385145105756, 0.6904761904761905, 'x[2] <= 2.0\ngini = 0.5\nsamples = 2\nvalue =
[1, 1]'),
 Text(0.20462370880472208, 0.6428571428571429, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.22036399409739302, 0.6428571428571429, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.22823413674372847, 0.6904761904761905, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
```

```
 Text(0.3356492867683227, 0.8333333333333334, 'x[4] <= 0.5\ngini = 0.213\nsamples =
239\nvalue = [210, 29]'),
 Text(0.3277791441219872, 0.7857142857142857, 'x[2] <= 14.0\ngini = 0.234\nsamples =
214\nvalue = [185, 29]'),
 Text(0.25184456468273486, 0.7380952380952381, 'x[5] <= 15.015\ngini = 0.5\nsamples =
2\nvalue = [1, 1]'),
 Text(0.2439744220363994, 0.6904761904761905, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.2597147073290703, 0.6904761904761905, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.4037137235612395, 0.7380952380952381, 'x[5] <= 7.91\ngini = 0.229\nsamples =
212\nvalue = [184, 28]'),
 Text(0.2754549926217413, 0.6904761904761905, 'x[2] <= 32.5\ngini = 0.172\nsamples =
84\nvalue = [76, 8]'),
 Text(0.2675848499754058, 0.6428571428571429, 'x[2] <= 19.5\ngini = 0.225\nsamples =
62\nvalue = [54, 8]'),
 Text(0.2597147073290703, 0.5952380952380952, 'gini = 0.0\nsamples = 10\nvalue = [10, 0]'),
 Text(0.2754549926217413, 0.5952380952380952, 'x[5] <= 7.01\ngini = 0.26\nsamples = 52\nvalue
= [44, 8]'),
 Text(0.24840137727496311, 0.5476190476190477, 'x[5] <= 5.494\ngini = 0.5\nsamples = 2\nvalue
= [1, 1]'),
 Text(0.24053123462862763, 0.5, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.2562715199212986, 0.5, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.3025086079685194, 0.5476190476190477, 'x[2] <= 20.25\ngini = 0.241\nsamples =
50\nvalue = [43, 7]'),
 Text(0.2720118052139695, 0.5, 'x[5] <= 7.14\ngini = 0.444\nsamples = 3\nvalue = [2, 1]'),
 Text(0.26414166256763405, 0.4523809523809524, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.27988194786030496, 0.4523809523809524, 'x[5] <= 7.542\ngini = 0.5\nsamples = 2\nvalue
= [1, 1]'),
 Text(0.2720118052139695, 0.40476190476190477, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.2877520905066404, 0.40476190476190477, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.33300541072306933, 0.5, 'x[5] <= 7.742\ngini = 0.223\nsamples = 47\nvalue = [41, 6]'),
 Text(0.31136251844564683, 0.4523809523809524, 'x[5] <= 7.183\ngini = 0.105\nsamples =
18\nvalue = [17, 1]'),
 Text(0.3034923757993114, 0.40476190476190477, 'x[5] <= 7.133\ngini = 0.375\nsamples =
4\nvalue = [3, 1]'),
 Text(0.2956222331529759, 0.35714285714285715, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
 Text(0.31136251844564683, 0.35714285714285715, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.3192326610919823, 0.40476190476190477, 'gini = 0.0\nsamples = 14\nvalue = [14, 0]'),
 Text(0.3546483030004919, 0.4523809523809524, 'x[2] <= 28.5\ngini = 0.285\nsamples =
29\nvalue = [24, 5]'),
 Text(0.33497294638465325, 0.40476190476190477, 'x[5] <= 7.798\ngini = 0.188\nsamples =
19\nvalue = [17, 2]'),
 Text(0.32710280373831774, 0.35714285714285715, 'x[2] <= 25.5\ngini = 0.375\nsamples =
8\nvalue = [6, 2]'),
 Text(0.3192326610919823, 0.30952380952380953, 'x[2] <= 21.5\ngini = 0.444\nsamples =
6\nvalue = [4, 2]'),
 Text(0.3034923757993114, 0.2619047619047619, 'x[5] <= 7.785\ngini = 0.5\nsamples = 2\nvalue =
[1, 1]'),
 Text(0.2956222331529759, 0.21428571428571427, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.31136251844564683, 0.21428571428571427, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
```

```
 Text(0.33497294638465325, 0.2619047619047619, 'x[5] <= 7.785\ngini = 0.375\nsamples =
4\nvalue = [3, 1]'),
 Text(0.32710280373831774, 0.21428571428571427, 'gini = 0.5\nsamples = 2\nvalue = [1, 1]'),
 Text(0.3428430890309887, 0.21428571428571427, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.33497294638465325, 0.30952380952380953, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.3428430890309887, 0.35714285714285715, 'gini = 0.0\nsamples = 11\nvalue = [11, 0]'),
 Text(0.3743236596163306, 0.40476190476190477, 'x[2] <= 29.5\ngini = 0.42\nsamples =
10\nvalue = [7, 3]'),
 Text(0.3585833743236596, 0.35714285714285715, 'x[5] <= 7.763\ngini = 0.5\nsamples = 4\nvalue
= [2, 2]'),
 Text(0.35071323167732416, 0.30952380952380953, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.36645351696999506, 0.30952380952380953, 'x[5] <= 7.885\ngini = 0.444\nsamples =
3\nvalue = [2, 1]'),
 Text(0.3585833743236596, 0.2619047619047619, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.3743236596163306, 0.2619047619047619, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.3900639449090015, 0.35714285714285715, 'x[5] <= 7.815\ngini = 0.278\nsamples =
6\nvalue = [5, 1]'),
 Text(0.382193802262666, 0.30952380952380953, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
 Text(0.39793408755533693, 0.30952380952380953, 'x[5] <= 7.875\ngini = 0.444\nsamples =
3\nvalue = [2, 1]'),
 Text(0.3900639449090015, 0.2619047619047619, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.4058042302016724, 0.2619047619047619, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.28332513526807673, 0.6428571428571429, 'gini = 0.0\nsamples = 22\nvalue = [22, 0]'),
 Text(0.5319724545007378, 0.6904761904761905, 'x[5] <= 7.988\ngini = 0.264\nsamples =
128\nvalue = [108, 20]'),
 Text(0.4825381210034432, 0.6428571428571429, 'x[2] <= 35.5\ngini = 0.494\nsamples = 9\nvalue
= [5, 4]'),
 Text(0.4746679783571077, 0.5952380952380952, 'x[2] <= 20.5\ngini = 0.408\nsamples = 7\nvalue
= [5, 2]'),
 Text(0.4589276930644368, 0.5476190476190477, 'x[3] <= 0.5\ngini = 0.5\nsamples = 2\nvalue =
[1, 1]'),
 Text(0.4510575504181013, 0.5, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.46679783571077227, 0.5, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.49040826364977863, 0.5476190476190477, 'x[2] <= 30.0\ngini = 0.32\nsamples = 5\nvalue
= [4, 1]'),
 Text(0.4825381210034432, 0.5, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.49827840629611414, 0.5, 'gini = 0.444\nsamples = 3\nvalue = [2, 1]'),
 Text(0.49040826364977863, 0.5952380952380952, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.5814067879980325, 0.6428571428571429, 'x[3] <= 0.5\ngini = 0.233\nsamples =
119\nvalue = [103, 16]'),
 Text(0.5484505656665027, 0.5952380952380952, 'x[5] <= 41.248\ngini = 0.264\nsamples =
96\nvalue = [81, 15]'),
 Text(0.5218888342351206, 0.5476190476190477, 'x[5] <= 20.656\ngini = 0.245\nsamples =
91\nvalue = [78, 13]'),
 Text(0.514018691588785, 0.5, 'x[5] <= 17.444\ngini = 0.259\nsamples = 85\nvalue = [72, 13]'),
 Text(0.5061485489424495, 0.4523809523809524, 'x[2] <= 26.5\ngini = 0.245\nsamples =
84\nvalue = [72, 12]'),
 Text(0.4608952287260207, 0.40476190476190477, 'x[5] <= 8.175\ngini = 0.184\nsamples =
39\nvalue = [35, 4]'),
```

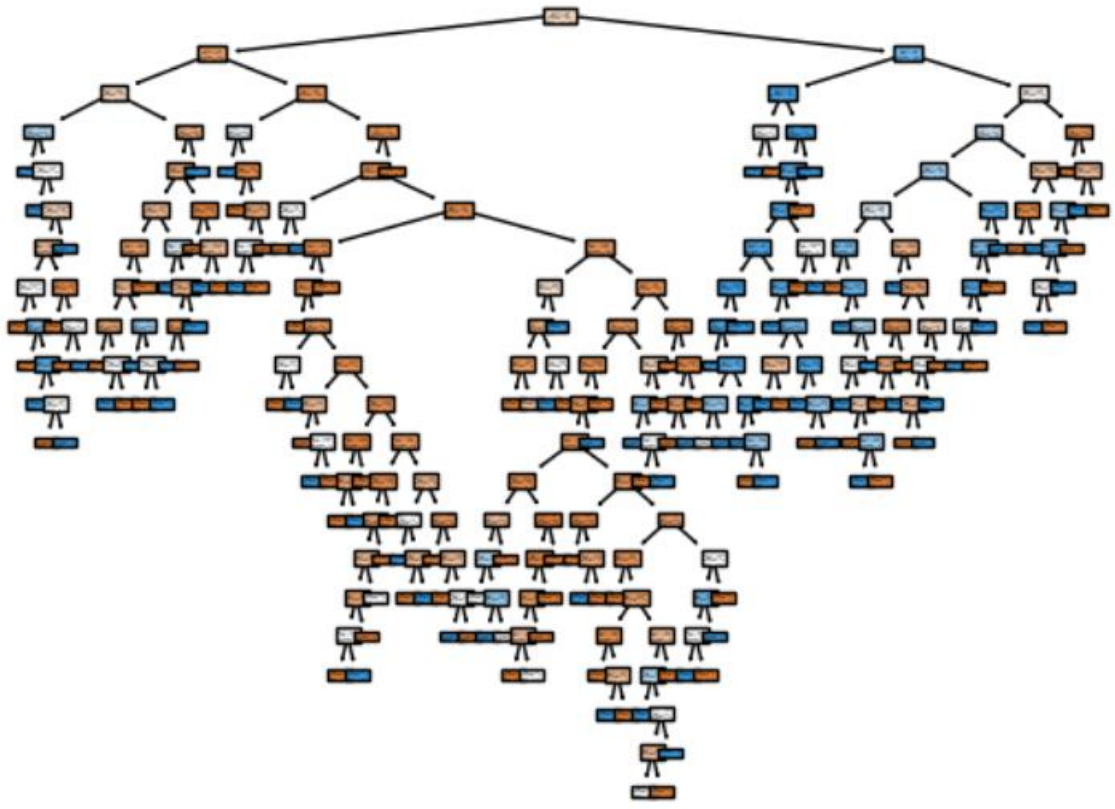
```
 Text(0.43728480078701426, 0.35714285714285715, 'x[2] <= 20.0\ngini = 0.444\nsamples =
9\nvalue = [6, 3]'),
 Text(0.4294146581406788, 0.30952380952380953, 'x[2] <= 17.0\ngini = 0.48\nsamples = 5\nvalue
= [2, 3]'),
 Text(0.42154451549434335, 0.2619047619047619, 'gini = 0.5\nsamples = 2\nvalue = [1, 1]'),
 Text(0.43728480078701426, 0.2619047619047619, 'x[2] <= 18.5\ngini = 0.444\nsamples =
3\nvalue = [1, 2]'),
 Text(0.4294146581406788, 0.21428571428571427, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.4451549434333497, 0.21428571428571427, 'gini = 0.5\nsamples = 2\nvalue = [1, 1]'),
 Text(0.4451549434333497, 0.30952380952380953, 'gini = 0.0\nsamples = 4\nvalue = [4, 0]'),
 Text(0.48450565666502704, 0.35714285714285715, 'x[0] <= 2.5\ngini = 0.064\nsamples =
30\nvalue = [29, 1]'),
 Text(0.4766355140186916, 0.30952380952380953, 'x[5] <= 11.0\ngini = 0.133\nsamples =
14\nvalue = [13, 1]'),
 Text(0.46876537137235613, 0.2619047619047619, 'x[2] <= 21.0\ngini = 0.32\nsamples = 5\nvalue
= [4, 1]'),
 Text(0.4608952287260207, 0.21428571428571427, 'x[2] <= 17.5\ngini = 0.444\nsamples =
3\nvalue = [2, 1]'),
 Text(0.45302508607968517, 0.16666666666666666, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.46876537137235613, 0.16666666666666666, 'gini = 0.5\nsamples = 2\nvalue = [1, 1]'),
 Text(0.4766355140186916, 0.21428571428571427, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.48450565666502704, 0.2619047619047619, 'gini = 0.0\nsamples = 9\nvalue = [9, 0]'),
 Text(0.4923757993113625, 0.30952380952380953, 'gini = 0.0\nsamples = 16\nvalue = [16, 0]'),
 Text(0.5514018691588785, 0.40476190476190477, 'x[5] <= 8.658\ngini = 0.292\nsamples =
45\nvalue = [37, 8]'),
 Text(0.5159862272503689, 0.35714285714285715, 'x[2] <= 44.5\ngini = 0.153\nsamples =
12\nvalue = [11, 1]'),
 Text(0.5081160846040335, 0.30952380952380953, 'gini = 0.0\nsamples = 8\nvalue = [8, 0]'),
 Text(0.5238563698967044, 0.30952380952380953, 'x[2] <= 47.5\ngini = 0.375\nsamples =
4\nvalue = [3, 1]'),
 Text(0.5159862272503689, 0.2619047619047619, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.5317265125430398, 0.2619047619047619, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
 Text(0.5868175110673881, 0.35714285714285715, 'x[0] <= 2.5\ngini = 0.334\nsamples =
33\nvalue = [26, 7]'),
 Text(0.5553369404820462, 0.30952380952380953, 'x[2] <= 30.5\ngini = 0.252\nsamples =
27\nvalue = [23, 4]'),
 Text(0.5474667978357107, 0.2619047619047619, 'gini = 0.0\nsamples = 9\nvalue = [9, 0]'),
 Text(0.5632070831283817, 0.2619047619047619, 'x[5] <= 12.938\ngini = 0.346\nsamples =
18\nvalue = [14, 4]'),
 Text(0.5395966551893753, 0.21428571428571427, 'x[2] <= 59.5\ngini = 0.198\nsamples =
9\nvalue = [8, 1]'),
 Text(0.5317265125430398, 0.16666666666666666, 'gini = 0.0\nsamples = 6\nvalue = [6, 0]'),
 Text(0.5474667978357107, 0.16666666666666666, 'x[2] <= 64.0\ngini = 0.444\nsamples =
3\nvalue = [2, 1]'),
 Text(0.5395966551893753, 0.11904761904761904, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.5553369404820462, 0.11904761904761904, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.5868175110673881, 0.21428571428571427, 'x[2] <= 45.0\ngini = 0.444\nsamples =
9\nvalue = [6, 3]'),
 Text(0.5789473684210527, 0.16666666666666666, 'x[2] <= 32.5\ngini = 0.48\nsamples = 5\nvalue
= [2, 3]'),
```

```
 Text(0.5710772257747172, 0.11904761904761904, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.5868175110673881, 0.11904761904761904, 'x[2] <= 40.5\ngini = 0.5\nsamples = 4\nvalue =
[2, 2]'),
 Text(0.5789473684210527, 0.07142857142857142, 'x[2] <= 36.5\ngini = 0.444\nsamples =
3\nvalue = [2, 1]'),
 Text(0.5710772257747172, 0.023809523809523808, 'gini = 0.5\nsamples = 2\nvalue = [1, 1]'),
 Text(0.5868175110673881, 0.023809523809523808, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.5946876537137236, 0.07142857142857142, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.5946876537137236, 0.16666666666666666, 'gini = 0.0\nsamples = 4\nvalue = [4, 0]'),
 Text(0.6182980816527299, 0.30952380952380953, 'x[5] <= 9.492\ngini = 0.5\nsamples = 6\nvalue
= [3, 3]'),
 Text(0.6104279390063945, 0.2619047619047619, 'x[2] <= 28.0\ngini = 0.375\nsamples = 4\nvalue
= [3, 1]'),
 Text(0.602557796360059, 0.21428571428571427, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.6182980816527299, 0.21428571428571427, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
 Text(0.6261682242990654, 0.2619047619047619, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.5218888342351206, 0.4523809523809524, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.529758976881456, 0.5, 'gini = 0.0\nsamples = 6\nvalue = [6, 0]'),
 Text(0.5750122970978849, 0.5476190476190477, 'x[5] <= 64.998\ngini = 0.48\nsamples =
5\nvalue = [3, 2]'),
 Text(0.5671421544515495, 0.5, 'x[2] <= 27.0\ngini = 0.444\nsamples = 3\nvalue = [1, 2]'),
 Text(0.559272011805214, 0.4523809523809524, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.5750122970978849, 0.4523809523809524, 'x[2] <= 30.0\ngini = 0.5\nsamples = 2\nvalue =
[1, 1]'),
 Text(0.5671421544515495, 0.40476190476190477, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.5828824397442204, 0.40476190476190477, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.5828824397442204, 0.5, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.6143630103295622, 0.5952380952380952, 'x[5] <= 25.0\ngini = 0.083\nsamples =
23\nvalue = [22, 1]'),
 Text(0.6064928676832267, 0.5476190476190477, 'gini = 0.0\nsamples = 13\nvalue = [13, 0]'),
 Text(0.6222331529758977, 0.5476190476190477, 'x[5] <= 26.5\ngini = 0.18\nsamples = 10\nvalue
= [9, 1]'),
 Text(0.6143630103295622, 0.5, 'x[2] <= 33.0\ngini = 0.32\nsamples = 5\nvalue = [4, 1]'),
 Text(0.6064928676832267, 0.4523809523809524, 'x[2] <= 28.5\ngini = 0.5\nsamples = 2\nvalue =
[1, 1]'),
 Text(0.5986227250368913, 0.40476190476190477, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.6143630103295622, 0.40476190476190477, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.6222331529758977, 0.4523809523809524, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
 Text(0.6301032956222331, 0.5, 'gini = 0.0\nsamples = 5\nvalue = [5, 0]'),
 Text(0.34351942941465813, 0.7857142857142857, 'gini = 0.0\nsamples = 25\nvalue = [25, 0]'),
 Text(0.822183964584358, 0.9285714285714286, 'x[0] <= 2.5\ngini = 0.36\nsamples = 200\nvalue =
[47, 153]'),
 Text(0.7088047220855878, 0.8809523809523809, 'x[2] <= 2.5\ngini = 0.12\nsamples = 125\nvalue
= [8, 117]'),
 Text(0.6930644367929168, 0.8333333333333334, 'x[0] <= 1.5\ngini = 0.5\nsamples = 2\nvalue =
[1, 1]'),
 Text(0.6851942941465814, 0.7857142857142857, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.7009345794392523, 0.7857142857142857, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.7245450073782588, 0.8333333333333334, 'x[5] <= 28.856\ngini = 0.107\nsamples =
123\nvalue = [7, 116]'),
```

```
 Text(0.7166748647319232, 0.7857142857142857, 'x[5] <= 28.231\ngini = 0.219\nsamples = 56\nvalue = [7, 49]'),
 Text(0.7088047220855878, 0.7380952380952381, 'x[2] <= 56.0\ngini = 0.194\nsamples = 55\nvalue = [6, 49]'),
 Text(0.6851942941465814, 0.6904761904761905, 'x[3] <= 0.5\ngini = 0.171\nsamples = 53\nvalue = [5, 48]'),
 Text(0.661583866207575, 0.6428571428571429, 'x[5] <= 13.25\ngini = 0.111\nsamples = 34\nvalue = [2, 32]'),
 Text(0.6537137235612396, 0.5952380952380952, 'x[5] <= 12.825\ngini = 0.188\nsamples = 19\nvalue = [2, 17]'),
 Text(0.6458435809149041, 0.5476190476190477, 'gini = 0.0\nsamples = 6\nvalue = [0, 6]'),
 Text(0.661583866207575, 0.5476190476190477, 'x[2] <= 26.0\ngini = 0.26\nsamples = 13\nvalue = [2, 11]'),
 Text(0.6458435809149041, 0.5, 'x[2] <= 21.0\ngini = 0.444\nsamples = 3\nvalue = [1, 2]'),
 Text(0.6379734382685687, 0.4523809523809524, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.6537137235612396, 0.4523809523809524, 'gini = 0.5\nsamples = 2\nvalue = [1, 1]'),
 Text(0.6773241515002459, 0.5, 'x[2] <= 37.0\ngini = 0.18\nsamples = 10\nvalue = [1, 9]'),
 Text(0.6694540088539105, 0.4523809523809524, 'gini = 0.0\nsamples = 7\nvalue = [0, 7]'),
 Text(0.6851942941465814, 0.4523809523809524, 'x[2] <= 39.0\ngini = 0.444\nsamples = 3\nvalue = [1, 2]'),
 Text(0.6773241515002459, 0.40476190476190477, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.6930644367929168, 0.40476190476190477, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.6694540088539105, 0.5952380952380952, 'gini = 0.0\nsamples = 15\nvalue = [0, 15]'),
 Text(0.7088047220855878, 0.6428571428571429, 'x[2] <= 25.0\ngini = 0.266\nsamples = 19\nvalue = [3, 16]'),
 Text(0.7009345794392523, 0.5952380952380952, 'gini = 0.0\nsamples = 6\nvalue = [0, 6]'),
 Text(0.7166748647319232, 0.5952380952380952, 'x[2] <= 27.5\ngini = 0.355\nsamples = 13\nvalue = [3, 10]'),
 Text(0.7009345794392523, 0.5476190476190477, 'x[5] <= 17.429\ngini = 0.444\nsamples = 3\nvalue = [2, 1]'),
 Text(0.6930644367929168, 0.5, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.7088047220855878, 0.5, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.7324151500245942, 0.5476190476190477, 'x[2] <= 43.0\ngini = 0.18\nsamples = 10\nvalue = [1, 9]'),
 Text(0.7245450073782588, 0.5, 'gini = 0.0\nsamples = 7\nvalue = [0, 7]'),
 Text(0.7402852926709297, 0.5, 'x[2] <= 44.5\ngini = 0.444\nsamples = 3\nvalue = [1, 2]'),
 Text(0.7324151500245942, 0.4523809523809524, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.7481554353172651, 0.4523809523809524, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.7324151500245942, 0.6904761904761905, 'x[0] <= 1.5\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
 Text(0.7245450073782588, 0.6428571428571429, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.7402852926709297, 0.6428571428571429, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.7245450073782588, 0.7380952380952381, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.7324151500245942, 0.7857142857142857, 'gini = 0.0\nsamples = 67\nvalue = [0, 67]'),
 Text(0.9355632070831283, 0.8809523809523809, 'x[5] <= 20.8\ngini = 0.499\nsamples = 75\nvalue = [39, 36]'),
 Text(0.8947368421052632, 0.8333333333333334, 'x[2] <= 27.5\ngini = 0.491\nsamples = 60\nvalue = [26, 34]'),
 Text(0.8445646827348746, 0.7857142857142857, 'x[4] <= 0.5\ngini = 0.458\nsamples = 45\nvalue = [16, 29]'),
```

```
 Text(0.7914412198721101, 0.7380952380952381, 'x[5] <= 7.89\ngini = 0.498\nsamples =
30\nvalue = [14, 16]'),
 Text(0.763895720609936, 0.6904761904761905, 'x[5] <= 6.987\ngini = 0.375\nsamples =
16\nvalue = [4, 12]'),
 Text(0.7560255779636006, 0.6428571428571429, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.7717658632562715, 0.6428571428571429, 'x[5] <= 7.742\ngini = 0.32\nsamples =
15\nvalue = [3, 12]'),
 Text(0.763895720609936, 0.5952380952380952, 'gini = 0.0\nsamples = 7\nvalue = [0, 7]'),
 Text(0.779636005902607, 0.5952380952380952, 'x[5] <= 7.815\ngini = 0.469\nsamples = 8\nvalue
= [3, 5]'),
 Text(0.7717658632562715, 0.5476190476190477, 'x[2] <= 17.0\ngini = 0.5\nsamples = 6\nvalue =
[3, 3]'),
 Text(0.763895720609936, 0.5, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.779636005902607, 0.5, 'x[2] <= 21.5\ngini = 0.48\nsamples = 5\nvalue = [3, 2]'),
 Text(0.7717658632562715, 0.4523809523809524, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.7875061485489424, 0.4523809523809524, 'x[2] <= 23.5\ngini = 0.444\nsamples = 3\nvalue
= [1, 2]'),
 Text(0.779636005902607, 0.40476190476190477, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.7953762911952779, 0.40476190476190477, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.7875061485489424, 0.5476190476190477, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.8189867191342843, 0.6904761904761905, 'x[2] <= 14.25\ngini = 0.408\nsamples =
14\nvalue = [10, 4]'),
 Text(0.8111165764879489, 0.6428571428571429, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.8268568617806198, 0.6428571428571429, 'x[2] <= 23.5\ngini = 0.355\nsamples =
13\nvalue = [10, 3]'),
 Text(0.8111165764879489, 0.5952380952380952, 'x[2] <= 19.0\ngini = 0.219\nsamples = 8\nvalue
= [7, 1]'),
 Text(0.8032464338416134, 0.5476190476190477, 'x[5] <= 12.148\ngini = 0.375\nsamples =
4\nvalue = [3, 1]'),
 Text(0.7953762911952779, 0.5, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.8111165764879489, 0.5, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
 Text(0.8189867191342843, 0.5476190476190477, 'gini = 0.0\nsamples = 4\nvalue = [4, 0]'),
 Text(0.8425971470732907, 0.5952380952380952, 'x[5] <= 15.975\ngini = 0.48\nsamples =
5\nvalue = [3, 2]'),
 Text(0.8347270044269552, 0.5476190476190477, 'x[5] <= 12.35\ngini = 0.5\nsamples = 4\nvalue =
[2, 2]'),
 Text(0.8268568617806198, 0.5, 'x[2] <= 25.5\ngini = 0.444\nsamples = 3\nvalue = [2, 1]'),
 Text(0.8189867191342843, 0.4523809523809524, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.8347270044269552, 0.4523809523809524, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.8425971470732907, 0.5, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.8504672897196262, 0.5476190476190477, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.897688145597639, 0.7380952380952381, 'x[5] <= 15.494\ngini = 0.231\nsamples =
15\nvalue = [2, 13]'),
 Text(0.8898180029513035, 0.6904761904761905, 'x[5] <= 14.331\ngini = 0.375\nsamples =
8\nvalue = [2, 6]'),
 Text(0.8819478603049681, 0.6428571428571429, 'x[2] <= 3.0\ngini = 0.245\nsamples = 7\nvalue =
[1, 6]'),
 Text(0.8740777176586325, 0.5952380952380952, 'x[2] <= 1.5\ngini = 0.5\nsamples = 2\nvalue =
[1, 1]'),
 Text(0.8662075750122971, 0.5476190476190477, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
```

Text(0.8819478603049681, 0.5476190476190477, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(0.8898180029513035, 0.5952380952380952, 'gini = 0.0\nsamples = 5\nvalue = [0, 5]'),
Text(0.897688145597639, 0.6428571428571429, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(0.9055582882439744, 0.6904761904761905, 'gini = 0.0\nsamples = 7\nvalue = [0, 7]'),
Text(0.9449090014756517, 0.7857142857142857, 'x[5] <= 14.85\ngini = 0.444\nsamples = 15\nvalue = [10, 5]'),
Text(0.9291687161829808, 0.7380952380952381, 'x[2] <= 54.0\ngini = 0.198\nsamples = 9\nvalue = [8, 1]'),
Text(0.9212985735366453, 0.6904761904761905, 'gini = 0.0\nsamples = 8\nvalue = [8, 0]'),
Text(0.9370388588293163, 0.6904761904761905, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(0.9606492867683227, 0.7380952380952381, 'x[2] <= 38.5\ngini = 0.444\nsamples = 6\nvalue = [2, 4]'),
Text(0.9527791441219872, 0.6904761904761905, 'x[5] <= 16.45\ngini = 0.32\nsamples = 5\nvalue = [1, 4]'),
Text(0.9449090014756517, 0.6428571428571429, 'x[5] <= 15.373\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
Text(0.9370388588293163, 0.5952380952380952, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(0.9527791441219872, 0.5952380952380952, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(0.9606492867683227, 0.6428571428571429, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
Text(0.9685194294146582, 0.6904761904761905, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(0.9763895720609936, 0.8333333333333334, 'x[5] <= 31.331\ngini = 0.231\nsamples = 15\nvalue = [13, 2]'),
Text(0.9685194294146582, 0.7857142857142857, 'gini = 0.0\nsamples = 9\nvalue = [9, 0]'),
Text(0.9842597147073291, 0.7857142857142857, 'x[5] <= 32.881\ngini = 0.444\nsamples = 6\nvalue = [4, 2]'),
Text(0.9763895720609936, 0.7380952380952381, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(0.9921298573536645, 0.7380952380952381, 'gini = 0.0\nsamples = 4\nvalue = [4, 0]')]



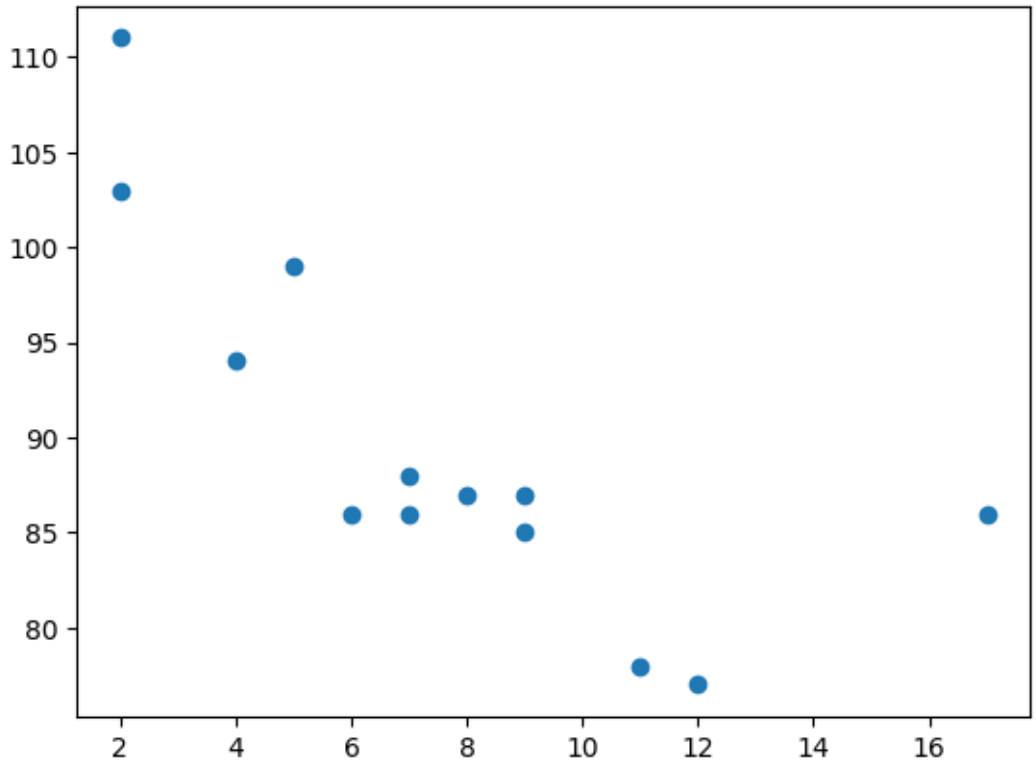
EXPERIMENT NO : 15

AIM : Linear Regression

CODE

```
import matplotlib.pyplot as plt
x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]
plt.scatter(x, y)
plt.show()
```

OUTPUT



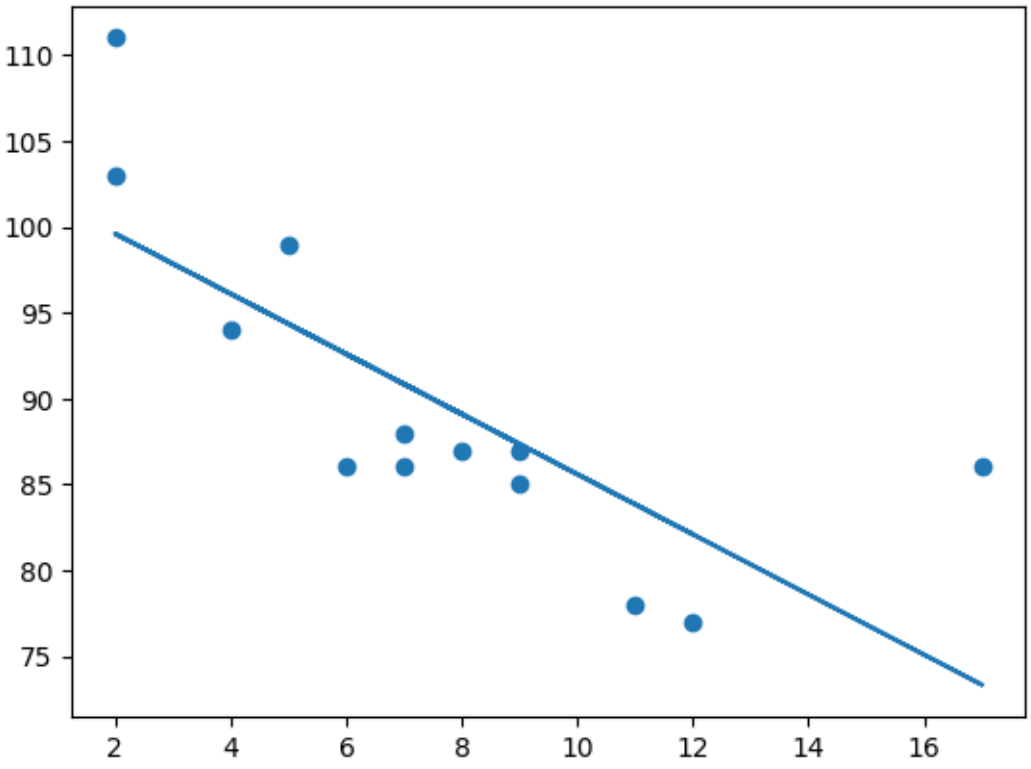
CODE

```
import matplotlib.pyplot as plt
from scipy import stats

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

slope, intercept, r, p, std_err = stats.linregress(x, y) # r correlation coefficiant   # p probability of hypothesis

def myfunc(x):
  return slope * x + intercept
```

```
mymodel = list(map(myfunc, x))

plt.scatter(x, y)
plt.plot(x, mymodel)
```

OUTPUT

[<matplotlib.lines.Line2D at 0x7c3875f89060>]



EXPERIMENT NO : 16

AIM : Linear Regression using cars1 dataset

CODE

```
import pandas
import warnings
warnings.filterwarnings("ignore")
df = pandas.read_csv("cars1.csv")
```

```
df.head(5)
```

	Car	Model	Volume	Weight	CO2
0	Toyoty	Aygo	1000	790	99
1	Mitsubishi	Space Star	1200	1160	95
2	Skoda	Citigo	1000	929	95
3	Fiat	500	900	865	90
4	Mini	Cooper	1500	1140	105

```
X = df[['Weight', 'Volume']]
y = df['CO2']
```

```
from sklearn import linear_model
regr = linear_model.LinearRegression()
regr.fit(X, y)
```

```
▾ LinearRegression
LinearRegression()
```

```
predictedCO2 = regr.predict([[2300, 1000]])
print(predictedCO2)
```

OUTPUT

```
[104.86715554]
```

EXPERIMENT NO :17

AIM: Regression using iris dataset

CODE

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv("iris.csv")
print(dataset)
```

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
..            ...          ...           ...          ...        ...
145           6.7          3.0           5.2          2.3  Virginica
146           6.3          2.5           5.0          1.9  Virginica
147           6.5          3.0           5.2          2.0  Virginica
148           6.2          3.4           5.4          2.3  Virginica
149           5.9          3.0           5.1          1.8  Virginica

[150 rows x 5 columns]
```

CODE

```
# Splitting the dataset into the Training set and Test set
X = dataset.iloc[:, [0,1,2, 3]].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 = 0.25, random_state = 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Fitting Logistic Regression to the Training set
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0, solver='lbfgs', multi_class='auto')
classifier.fit(X_train, y_train)
```

OUTPUT

```
LogisticRegression
LogisticRegression(random_state=0)
```

CODE

```
# Predicting the Test set results
y_pred = classifier.predict(X_test)
print(y_pred)
```

OUTPUT

```
['Virginica' 'Versicolor' 'Setosa' 'Virginica' 'Setosa' 'Virginica'
 'Setosa' 'Versicolor' 'Versicolor' 'Versicolor' 'Virginica' 'Versicolor'
 'Versicolor' 'Versicolor' 'Versicolor' 'Setosa' 'Versicolor' 'Versicolor'
 'Setosa' 'Setosa' 'Virginica' 'Versicolor' 'Setosa' 'Setosa' 'Virginica'
 'Setosa' 'Setosa' 'Versicolor' 'Versicolor' 'Setosa' 'Virginica'
 'Versicolor' 'Setosa' 'Virginica' 'Virginica' 'Versicolor' 'Setosa'
 'Virginica']
```

CODE

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

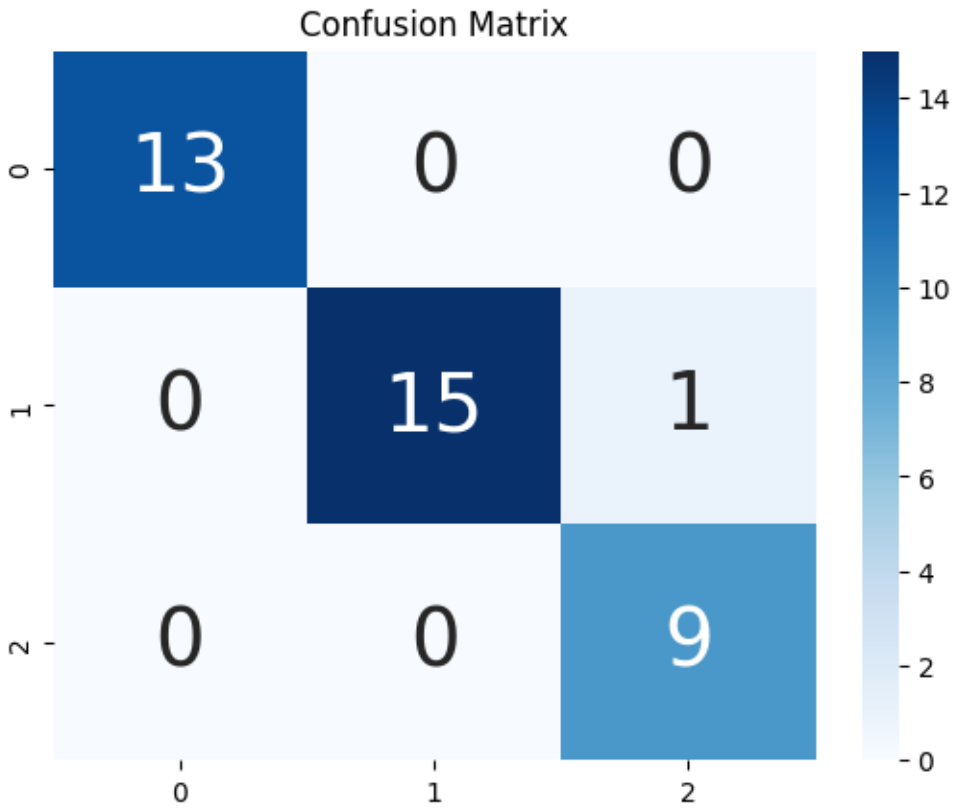
OUTPUT

```
[[13  0  0]
 [ 0 15  1]
 [ 0  0  9]]
```

CODE

```
# Plot confusion matrix
import seaborn as sns
import pandas as pd

# confusion matrix sns heatmap
ax = plt.axes()
df_cm = cm
sns.heatmap(df_cm, annot=True, annot_kws={"size": 30}, fmt='d',cmap="Blues", ax = ax )
ax.set_title('Confusion Matrix')
plt.show()
```

OUTPUT



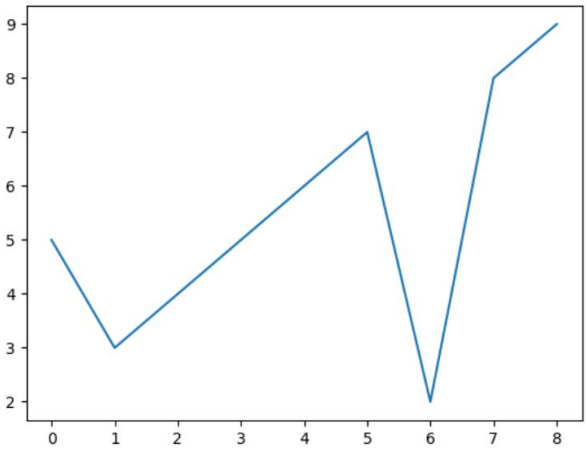
EXPERIMENT NO :4

AIM: Data Visualization

CODE

```
y = [5,3,4,5,6,7,2,8,9]
plt.plot (y)
plt.show()
```

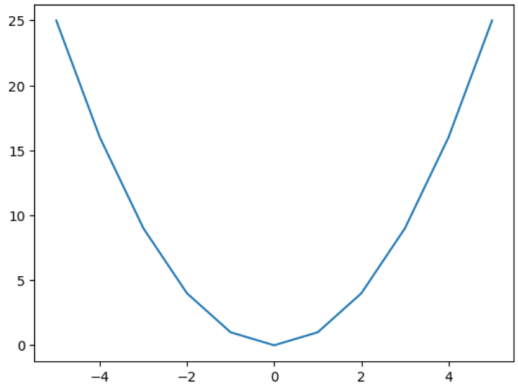
OUTPUT



CODE

```
x = [-5,-4,-3,-2,-1,0,1,2,3,4,5]
#y = [25,16,9,4,1,0,1,4,9,16,25]
y = [i**2 for i in x]
plt.plot(x,y)
plt.show()
```

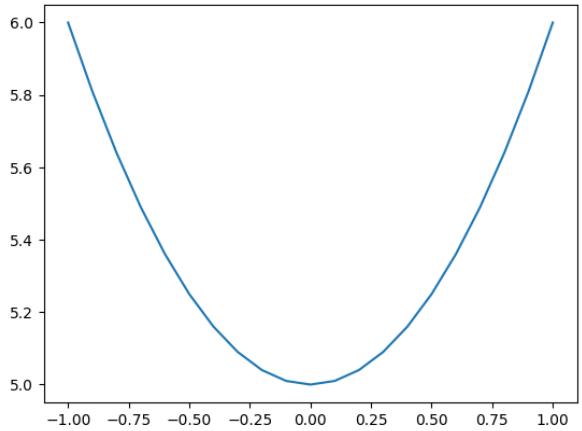
OUTPUT



CODE

```
import numpy as np
import math

x = np.arange(-1,1.1,0.1).tolist()
y = [i**2 + 5 for i in x]

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

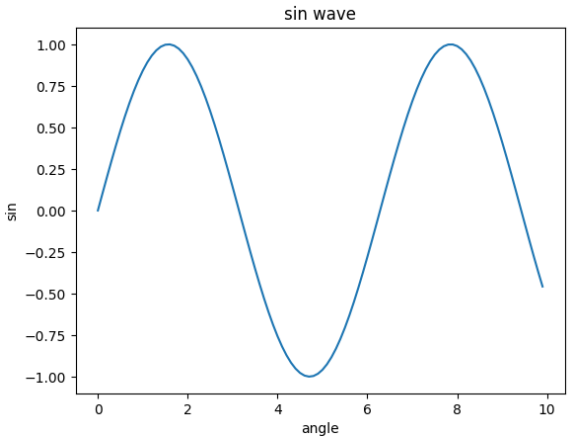
OUTPUT



CODE

```
import numpy as np
x = np.arange(0,10,0.1)
y = np.sin(x)
plt.plot(x,y)
plt.xlabel('angle')
plt.ylabel('sin')
plt.title('sin wave')
plt.show()
```

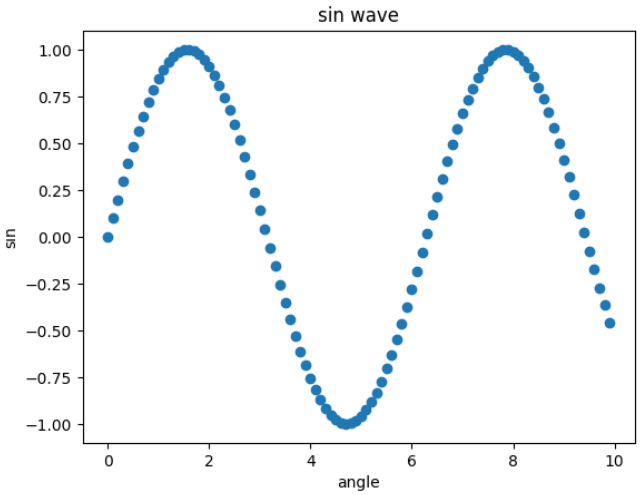
OUTPUT



CODE

```
plt.scatter(x,y)
plt.xlabel('angle')
plt.ylabel('sin')
plt.title('sin wave')
plt.show()
```

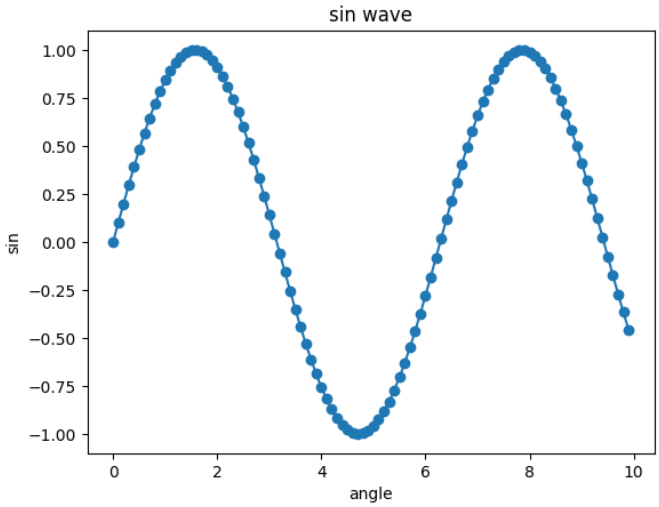
OUTPUT



CODE

```
plt.plot(x,y)
plt.scatter(x,y)
plt.xlabel('angle')
plt.ylabel('sin')
plt.title('sin wave')
plt.show()
```

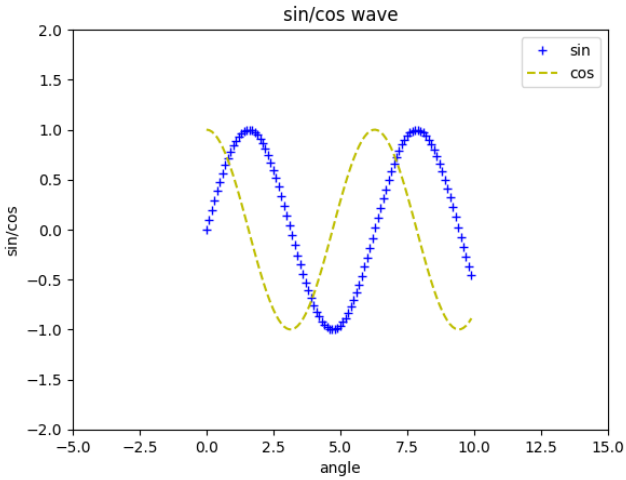
OUTPUT



CODE

```
plt.plot(x,np.sin(x), 'b+', label='sin')
plt.plot(x,np.cos(x) ,'y--', label='cos') #Scatter connections
plt.xlabel('angle')
plt.ylabel('sin/cos')
plt.title('sin/cos wave')
plt.ylim(-2,2)
plt.xlim(-5,15)
plt.legend()
plt.show()
```

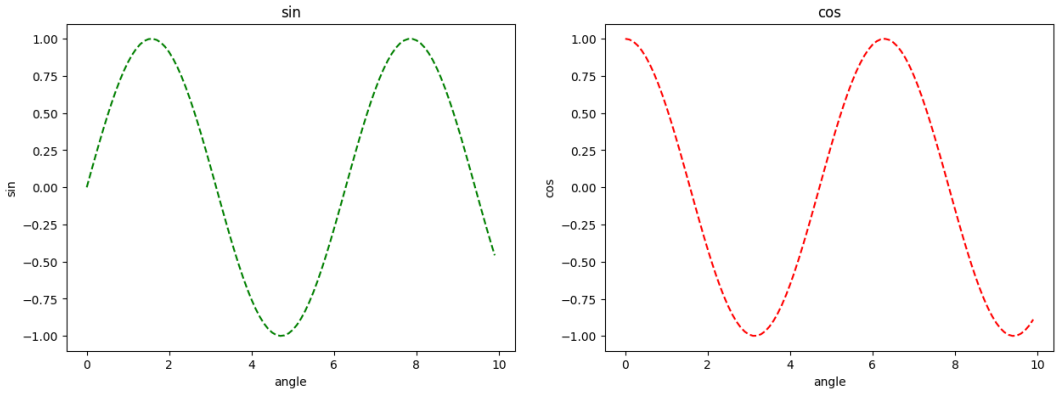
OUTPUT



CODE

```
fig, axis = plt.subplots(1,2, figsize=(15,5))
x = np.arange(0,10,0.1)
axis[0].plot(x,np.sin(x), 'g--')
axis[0].set_title('sin')
axis[0].set_xlabel('angle')
axis[0].set_ylabel('sin')
axis[1].plot(x,np.cos(x), 'r--')
axis[1].set_title('cos')
axis[1].set_xlabel('angle')
axis[1].set_ylabel('cos')
plt.show()
```

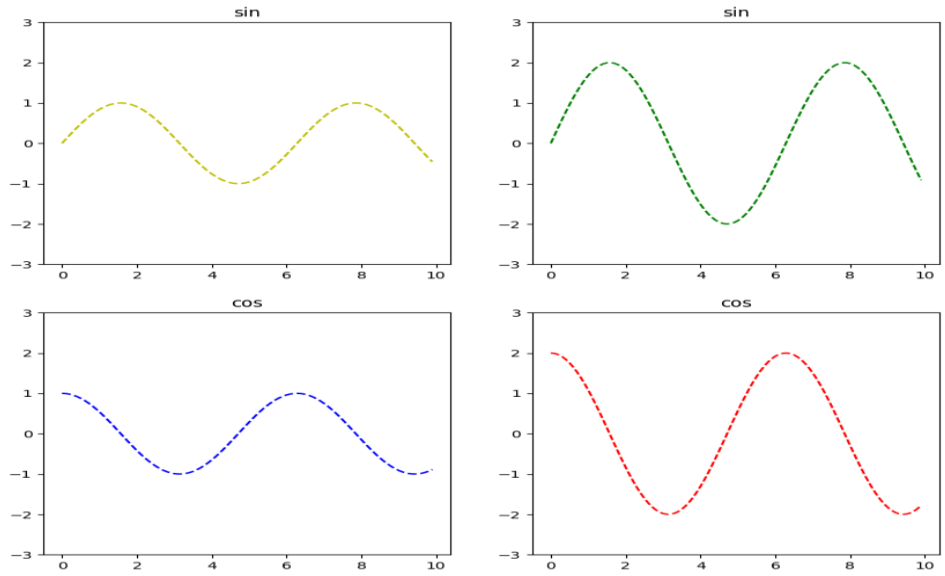
OUTPUT



CODE

```
fig, axis = plt.subplots(2,2, figsize=(10,10))
x = np.arange(0,10,0.1)
axis[0][0].plot(x,np.sin(x), 'y--')
axis[0][0].set_title('sin')
axis[0][0].set_ylim(-3,3)
axis[0][1].plot(x,2*np.sin(x), 'g--')
axis[0][1].set_title('sin')
axis[0][1].set_ylim(-3,3)
axis[1][0].plot(x,np.cos(x), 'b--')
axis[1][0].set_title('cos')
axis[1][0].set_ylim(-3,3)
axis[1][1].plot(x,2*np.cos(x), 'r--')
axis[1][1].set_title('cos')
axis[1][1].set_ylim(-3,3)
plt.show()
```

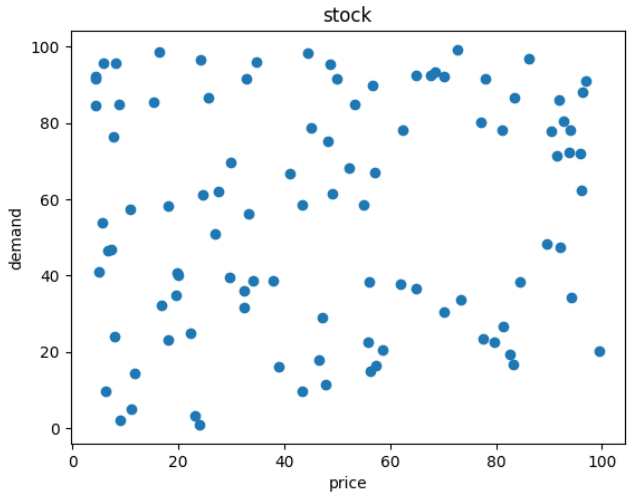
OUTPUT



CODE

```
x = np.random.random(100)*100
y = np.random.random(100)*100
plt.scatter(x,y)
plt.xlabel('price')
plt.ylabel('demand')
plt.title('stock')
plt.show()
```

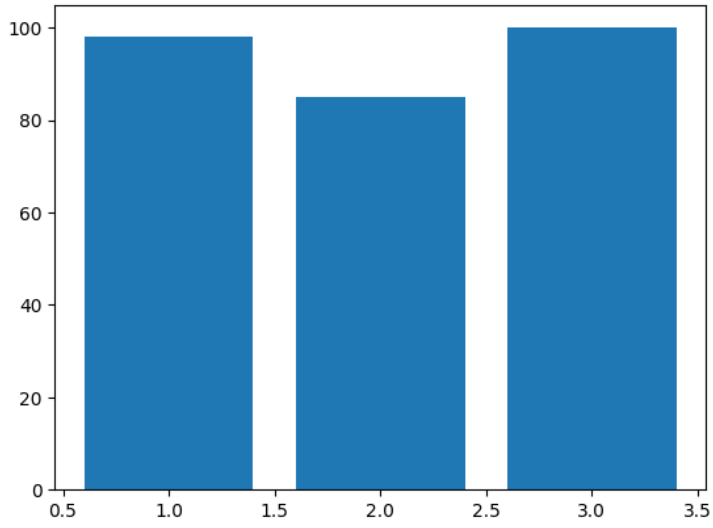
OUTPUT



CODE

```
x = np.array([1,2,3])
y = [98,85,100]
plt.bar(x,y)
plt.show()
```

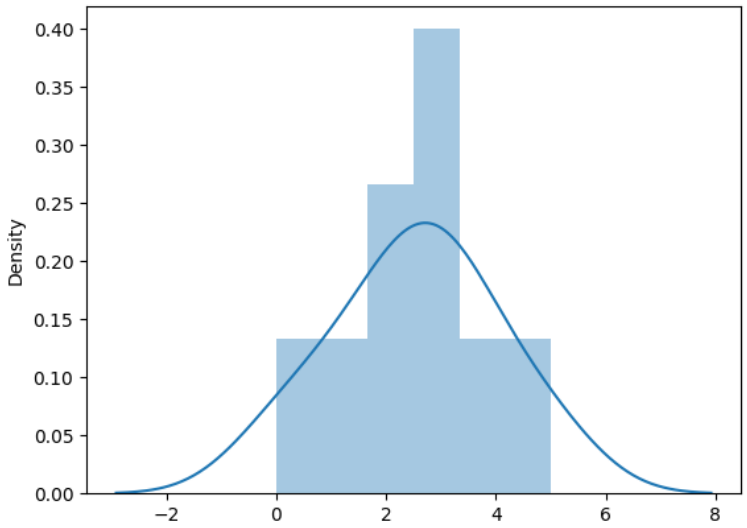
OUTPUT



CODE

```
import matplotlib.pyplot as plt
import seaborn as sns
sns.distplot([0, 1, 2, 3, 4, 5,3,2,3])
plt.show()
```

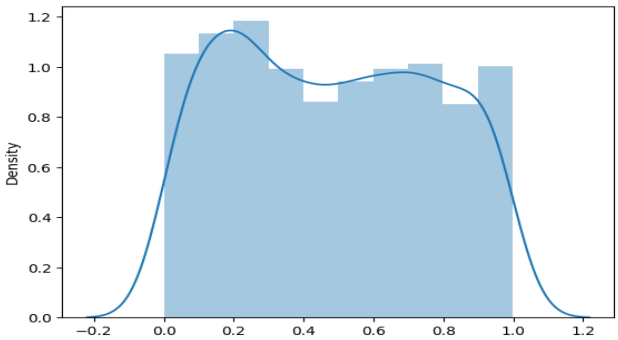
OUTPUT

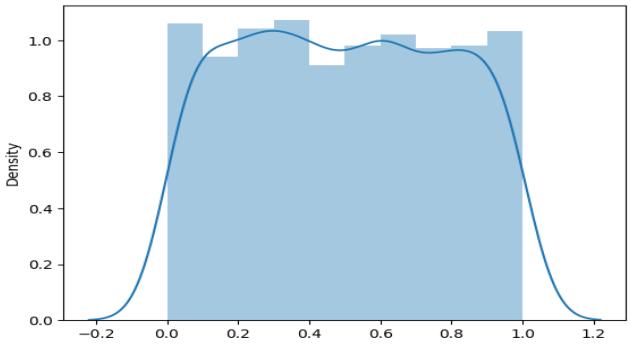


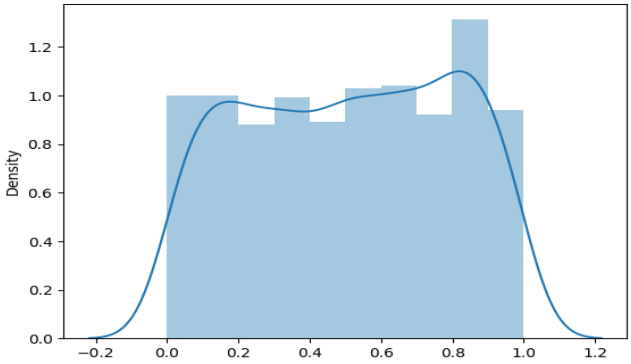
CODE

```
from numpy import random
import matplotlib.pyplot as plt
import seaborn as sns
x = np.random.random((1000,3))
sns.distplot(x[:,0], hist=True)
plt.show()
sns.distplot(x[:,1], hist=True)
plt.show()
sns.distplot(x[:,2], hist=True)
plt.show()
```

OUTPUT



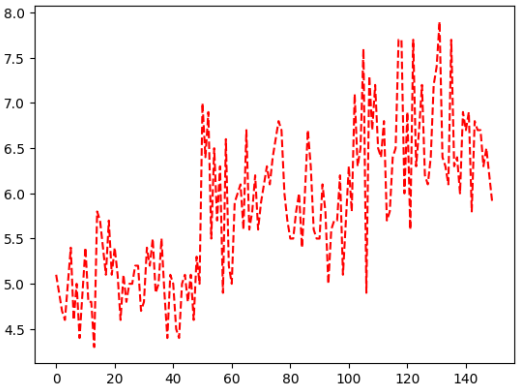




CODE

```
import matplotlib.pyplot as plt
plt.plot(iris["sepal.length"], "r--")
plt.show
```

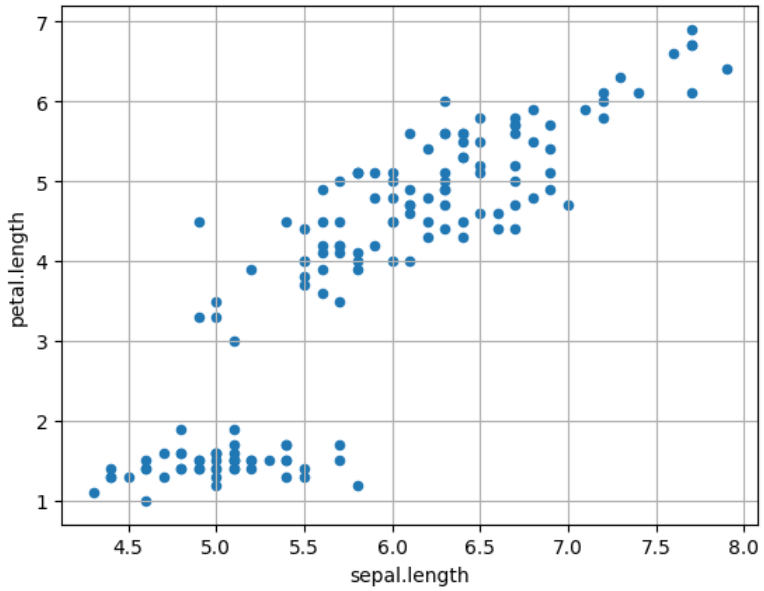
OUTPUT



CODE

```
iris.plot(kind ="scatter",
          x ='sepal.length',
          y ='petal.length')
plt.grid()
```

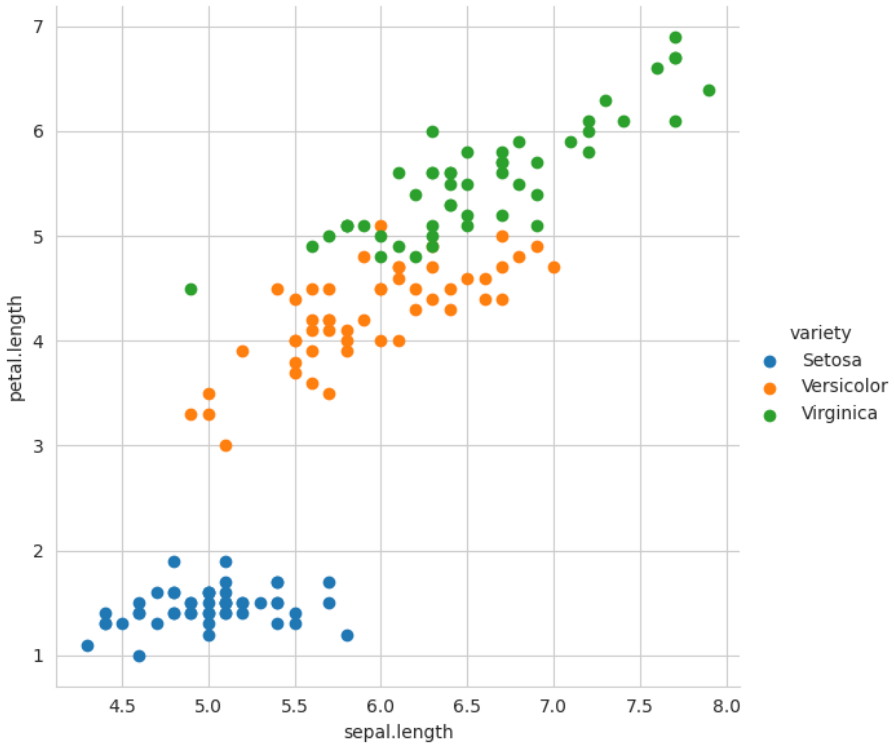
OUTPUT



CODE

```
import seaborn as sns
sns.set_style("whitegrid")
sns.FacetGrid(iris, hue ="variety",
              height = 6).map(plt.scatter,
                              'sepal.length',
                              'petal.length').add_legend()
```

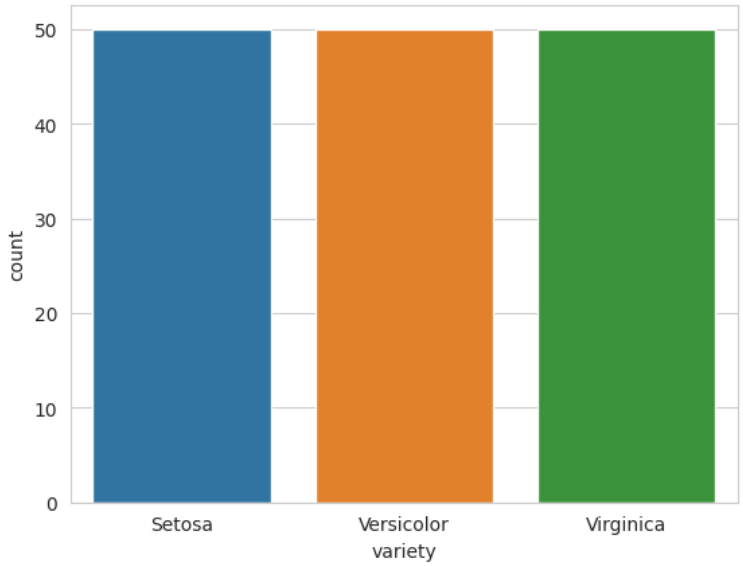
OUTPUT



CODE

```
sns.countplot(x='variety', data=iris, )
plt.show()
```

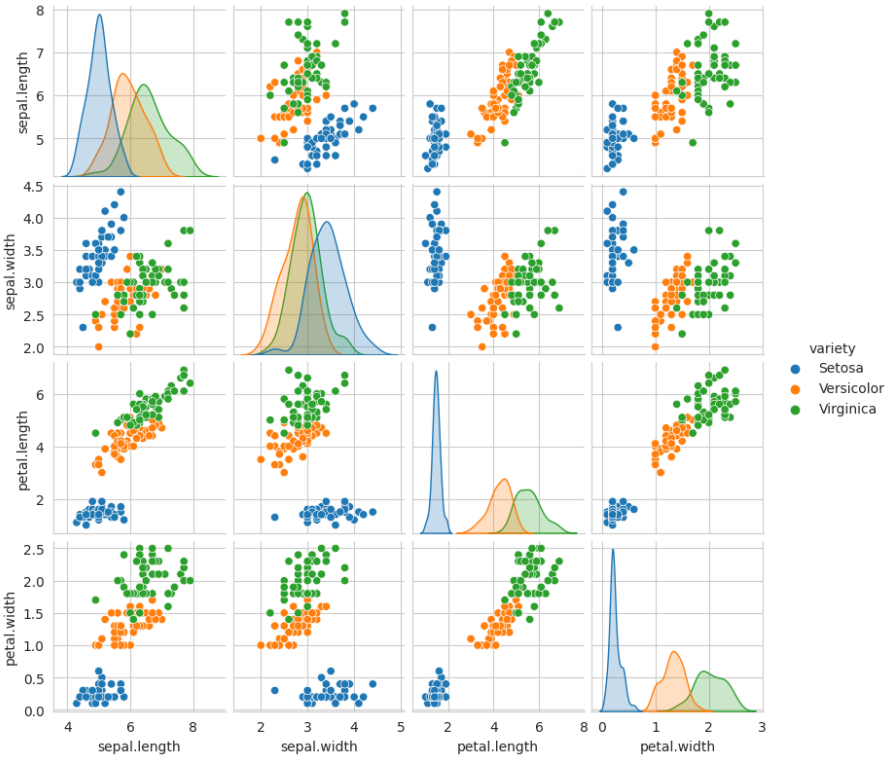
OUTPUT



CODE

```
sns.pairplot(iris,hue='variety', height=2)
```

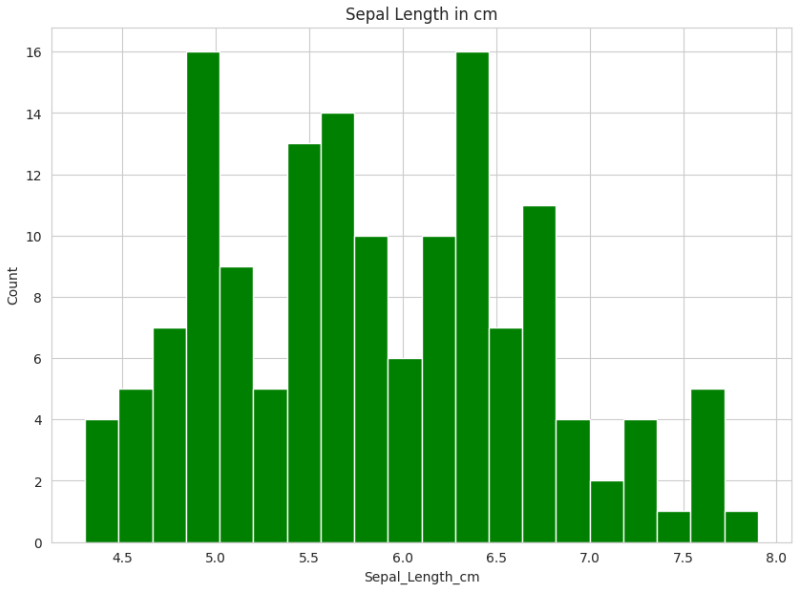
OUTPUT



CODE

```
plt.figure(figsize = (10, 7))
x = iris["sepal.length"]
plt.hist(x, bins = 20, color = "green")
plt.title("Sepal Length in cm")
plt.xlabel("Sepal_Length_cm")
plt.ylabel("Count")
```

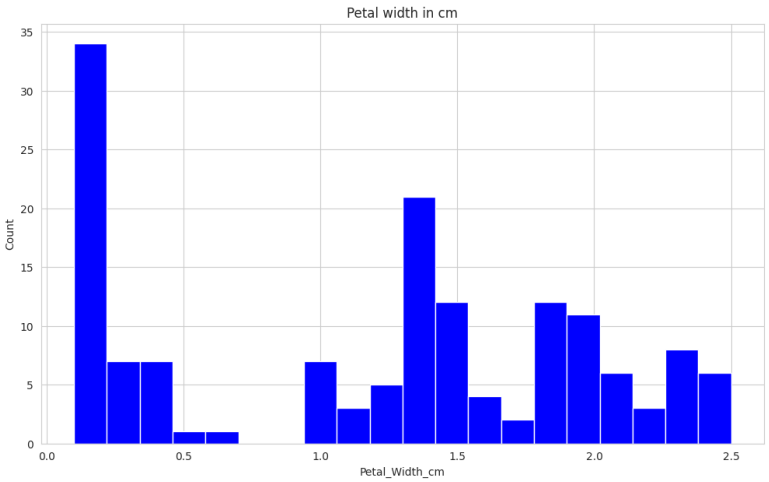
OUTPUT



CODE

```
plt.figure(figsize = (12, 7))
x = iris["petal.width"]
plt.hist(x, bins =20, color = "blue")
plt.title("Petal width in cm")
plt.xlabel("Petal_Width_cm")
plt.ylabel("Count")
```

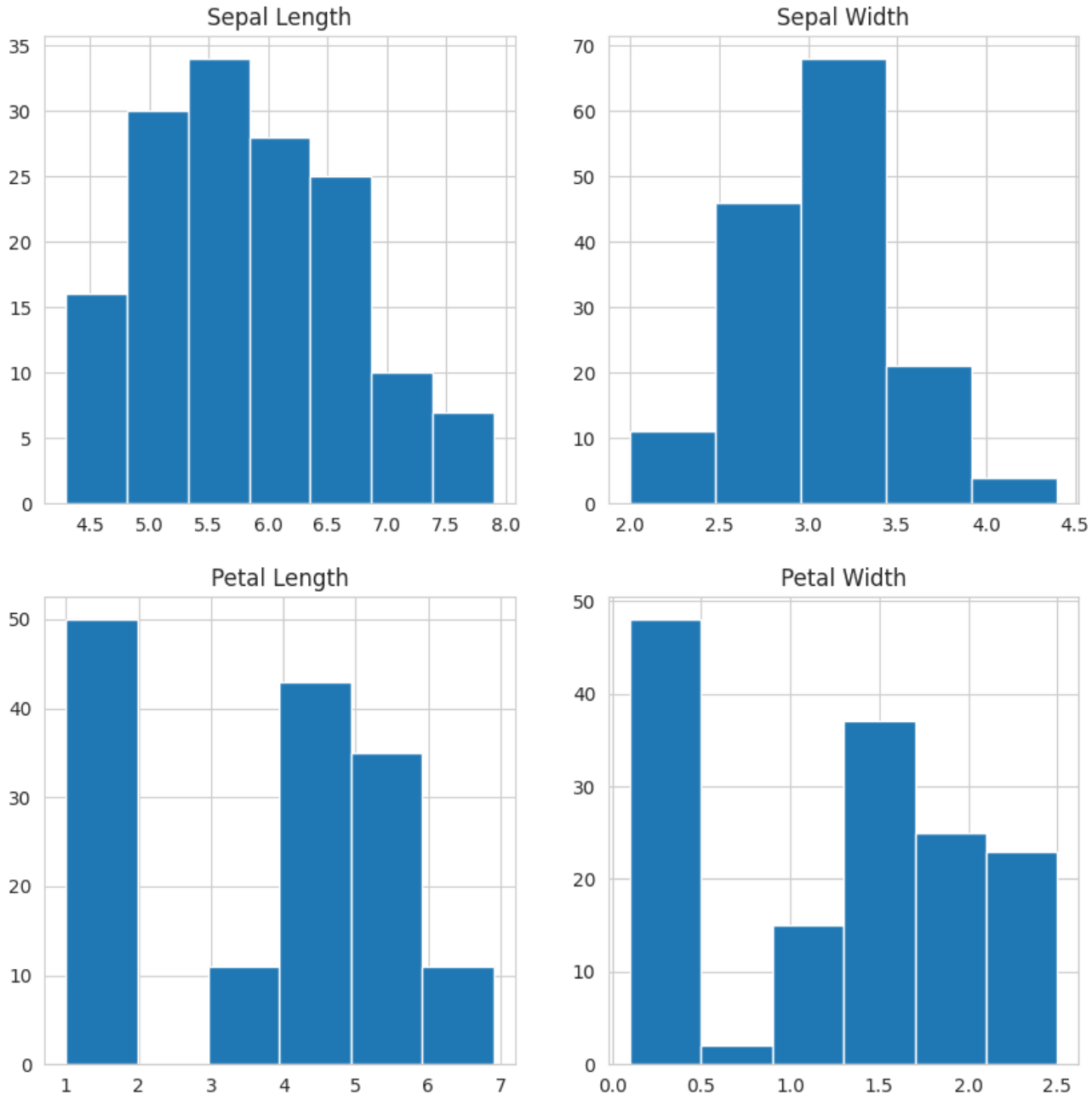
OUTPUT



CODE

```
fig, axes = plt.subplots(2, 2, figsize=(10,10))
axes[0,0].set_title("Sepal Length")
axes[0,0].hist(iris['sepal.length'], bins=7)
axes[0,1].set_title("Sepal Width")
axes[0,1].hist(iris['sepal.width'], bins=5);
axes[1,0].set_title("Petal Length")
axes[1,0].hist(iris['petal.length'], bins=6);
axes[1,1].set_title("Petal Width")
axes[1,1].hist(iris['petal.width'], bins=6);
```

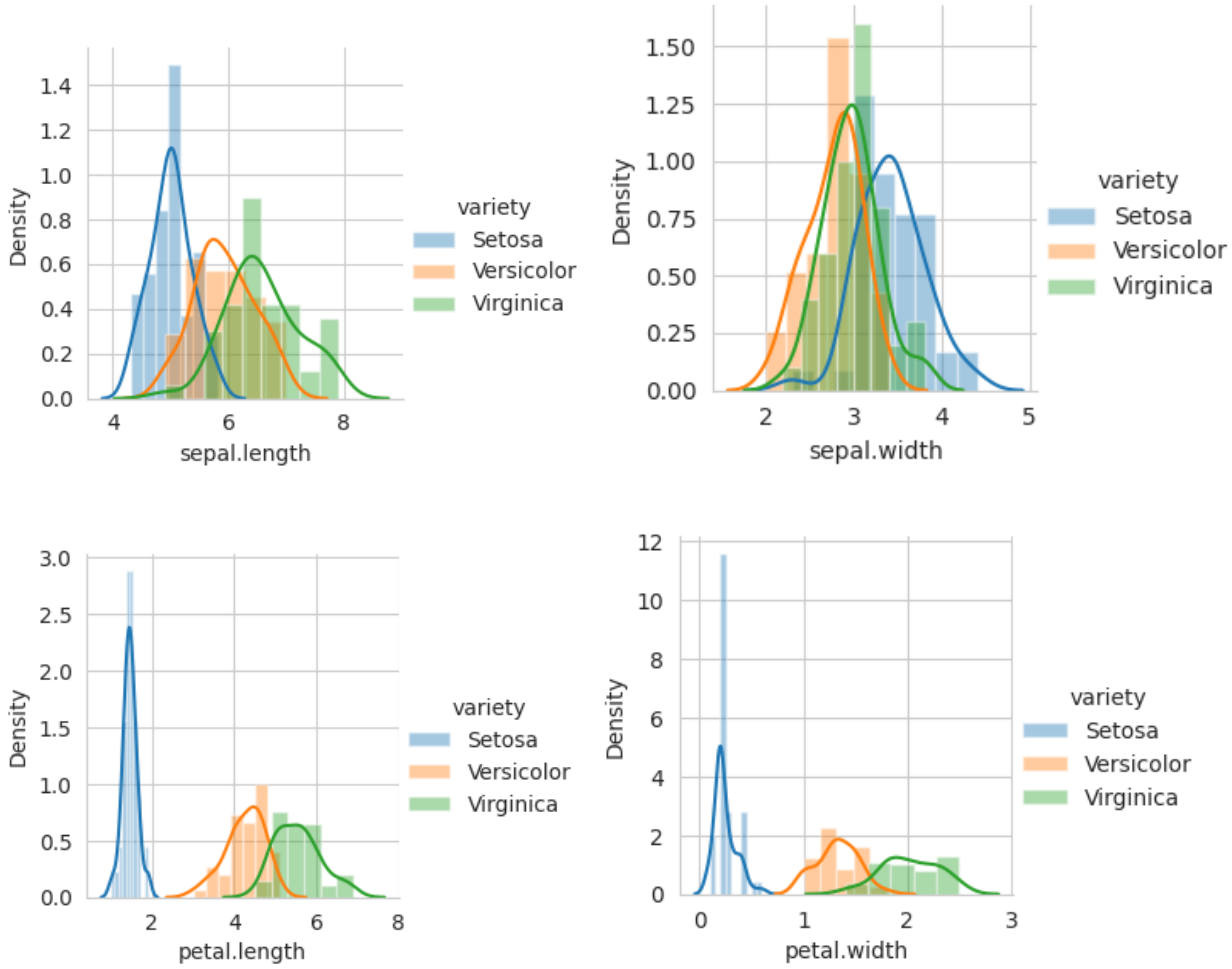
OUTPUT



CODE

```
plot = sns.FacetGrid(iris, hue="variety")
plot.map(sns.distplot, "sepal.length").add_legend()
plot = sns.FacetGrid(iris, hue="variety")
plot.map(sns.distplot, "sepal.width").add_legend()
plot = sns.FacetGrid(iris, hue="variety")
plot.map(sns.distplot, "petal.length").add_legend()
plot = sns.FacetGrid(iris, hue="variety")
plot.map(sns.distplot, "petal.width").add_legend()
plt.show()
```

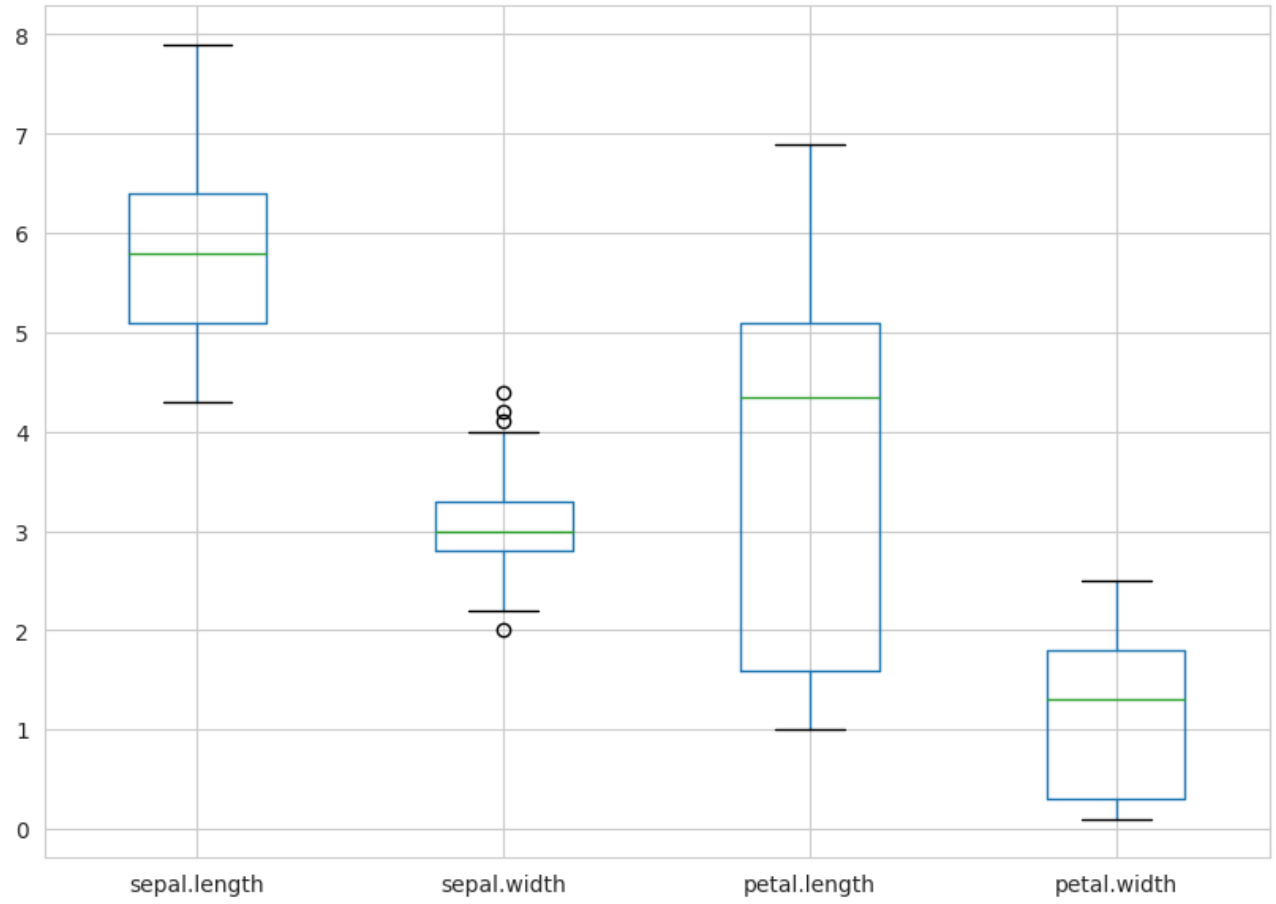
OUTPUT



CODE

```
plt.figure(figsize = (10, 7))
iris.boxplot()
```

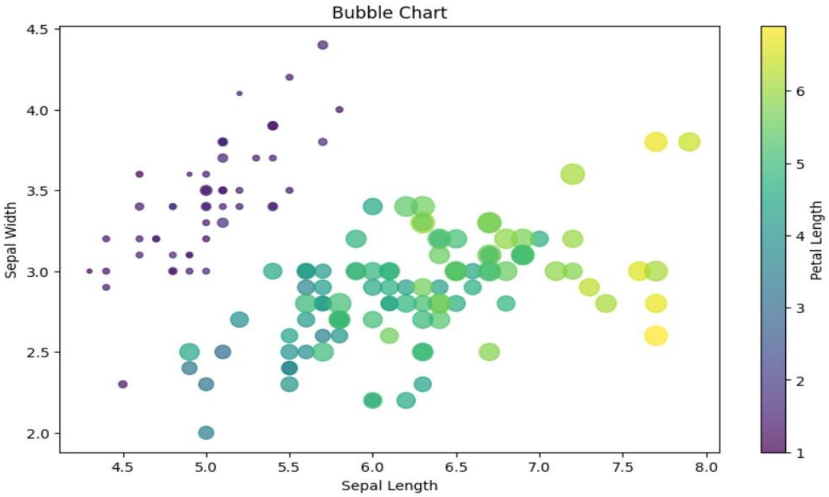
OUTPUT



CODE

```
import seaborn as sns

plt.figure(figsize=(10, 6))

plt.scatter(x=iris["sepal_length"], y=iris["sepal_width"], c=iris["petal_length"], s=iris["petal_width"] * 100, cmap="viridis", alpha=0.7)

plt.colorbar(label="Petal Length")

plt.xlabel("Sepal Length")

plt.ylabel("Sepal Width")

plt.title("Bubble Chart")

plt.show()
```

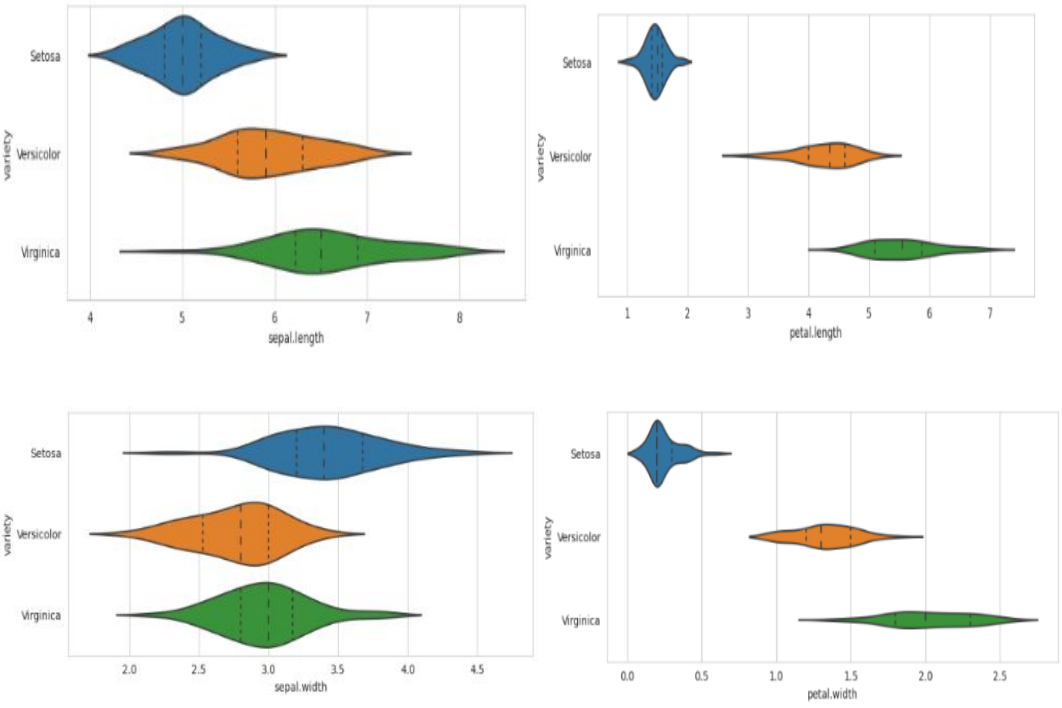
OUTPUT



CODE

```
import matplotlib.gridspec as gridspec

fig = plt.figure(figsize=(9, 40))

outer = gridspec.GridSpec(4, 1, wspace=0.2, hspace=0.2)
for i, col in enumerate(iris.columns[:-1]):
  inner = gridspec.GridSpecFromSubplotSpec(2, 1,subplot_spec=outer[i], wspace=0.2, hspace=0.4)
  ax = plt.Subplot(fig, inner[1])
  _ = sns.violinplot(y="variety", x=f"{col}", data=iris, inner='quartile', ax=ax)
  fig.add_subplot(ax)
fig.show()
```

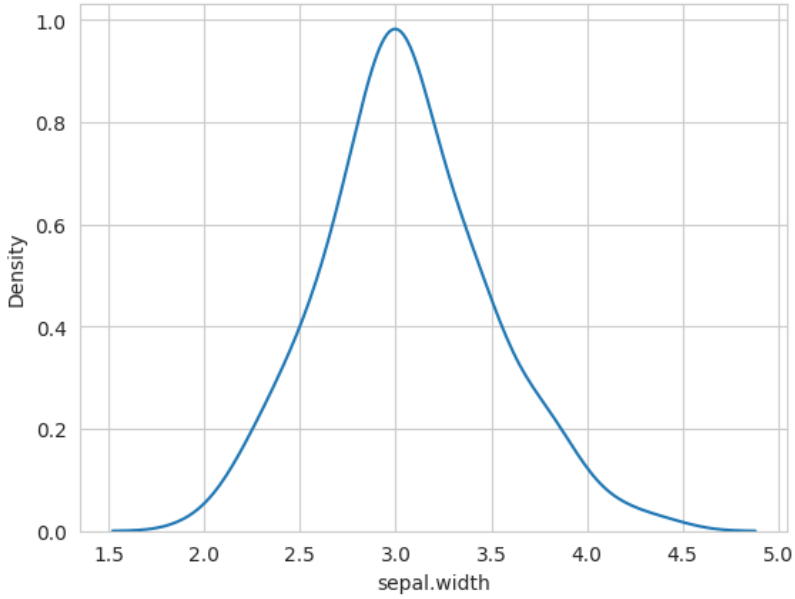
OUTPUT



CODE

```
sns.kdeplot(iris['sepal.width'])
```

OUTPUT



EXPERIMENT NO :10

AIM: Classification using Naïve Bayes with iris dataset

CODE

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

df = pd.read_csv("iris.csv")
X = df.iloc[:,:4].values
y = df['variety'].values
df.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

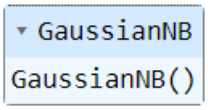
CODE

```
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.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)
```

CODE

```
from sklearn.naive_bayes import GaussianNB

classifier = GaussianNB()

classifier.fit(X_train, y_train)
```

OUTPUT



CODE

```
classifier.score(X_test,y_test)
```

OUTPUT

0.9

CODE

```
y_pred = classifier.predict(X_test)

y_pred
```

OUTPUT

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

CODE

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

OUTPUT

```
[[ 8  0  0]
 [ 0  9  0]
 [ 0  3 10]]
              precision    recall  f1-score   support

      Setosa       1.00      1.00      1.00         8
  Versicolor       0.75      1.00      0.86         9
   Virginica       1.00      0.77      0.87        13

    accuracy                           0.90        30
   macro avg       0.92      0.92      0.91        30
weighted avg       0.93      0.90      0.90        30
```

CODE

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

```
df_result
```

OUTPUT

	Real Values	Predicted Values
0	Virginica	Virginica
1	Setosa	Setosa
2	Versicolor	Versicolor
3	Virginica	Virginica
4	Virginica	Virginica
5	Versicolor	Versicolor

EXPERIMENT NO :14

AIM : Decision tree using titanic dataset

CODE

```
import pandas as pd
#df = pd.read_csv('titanic.csv', index_col='PassengerId')
df = pd.read_csv('titanic.csv')
print(df.head(2))
```

OUTPUT

```
PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1

   Parch     Ticket     Fare Cabin Embarked
0      0  A/5 21171   7.2500   NaN        S
1      0   PC 17599  71.2833   C85        C
```

CODE

```
df.shape
```

OUTPUT

```
(891, 12)
```

CODE

```
df = df[['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Survived']]
df.shape
```

OUTPUT

```
(891, 7)
```

CODE

```
df['Sex'] = df['Sex'].map({'male': 0, 'female': 1})
df = df.dropna()
df.shape
```

OUTPUT

(714, 7)

CODE

```
X = df.drop('Survived', axis=1)
y = df['Survived']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)

from sklearn import tree
model = tree.DecisionTreeClassifier()
model.fit(X_train, y_train)
model.score(X_test, y_test)
```

OUTPUT

0.8156424581005587

CODE

```
from sklearn.metrics import accuracy_score
y_predict = model.predict(X_test)
print("Accuracy:",accuracy_score(y_test, y_predict) )
```

OUTPUT

Accuracy: 0.8156424581005587

CODE

```
from sklearn.metrics import confusion_matrix
pd.DataFrame(
    confusion_matrix(y_test, y_predict),
    columns=['Predicted Not Survival', 'Predicted Survival'],
    index=['True Not Survival', 'True Survival']
```

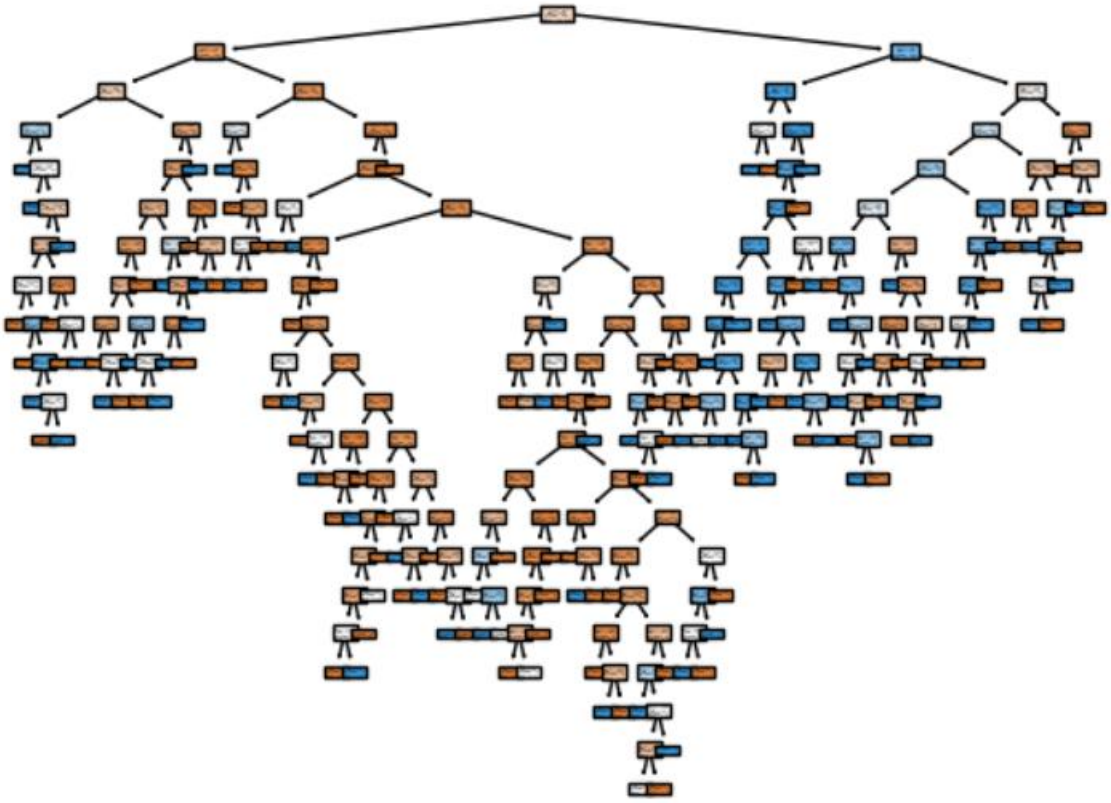
OUTPUT

	Predicted Not Survival	Predicted Survival
True Not Survival	97	15
True Survival	18	49

CODE

```
from sklearn import tree
tree.plot_tree(model,filled=True)
```

OUTPUT



EXPERIMENT NO :18

AIM: Prediction using a Combination of Classifiers on Hepatitis data

CODE

```
# Importing the libraries
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
from sklearn.metrics import auc, roc_auc_score, roc_curve, recall_score
import pandas as pd

df = pd.read_csv("hepatitis.csv")
X = df.drop('pstatus', axis=1)
y = df['pstatus']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30)
print(df.columns)
```

OUTPUT

```
Index(['pstatus', 'age', 'sex', 'steroid', 'antivirals', 'fatigue',
       'malaise','anorexia', 'liver_big', 'liver_firm', 'spleen_palable',
       'spiders','ascites', 'varices', 'bilirubin', 'alk_phosphate', 'sgot',
       'albumin','protime', 'histology'],
    dtype='object')
```

CODE

```
# KNN Classifier
from sklearn.neighbors import KNeighborsClassifier
classifier1 = KNeighborsClassifier(n_neighbors=5)
classifier1.fit(X_train, y_train)
```

OUTPUT

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

CODE

```
y_pred1 = classifier1.predict(X_test)
accuracy = classifier1.score(X_test, y_test)
print("Accuracy:", accuracy)
print(confusion_matrix(y_test, y_pred1))
print(classification_report(y_test, y_pred1))
```

OUTPUT

Accuracy: 0.813953488372093

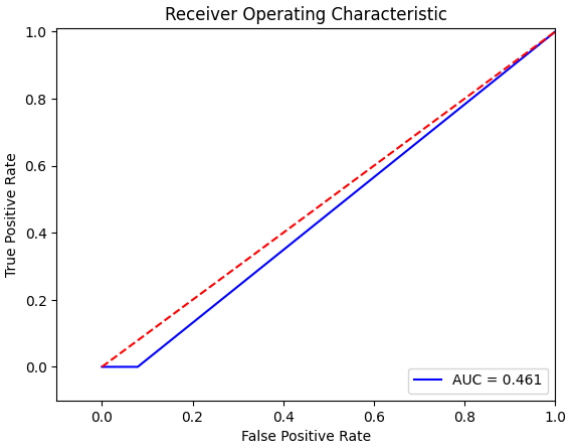
```
[[35  3]
 [ 5  0]]
```

	precision	recall	f1-score	support
0	0.88	0.92	0.90	38
1	0.00	0.00	0.00	5
accuracy			0.81	43
macro avg	0.44	0.46	0.45	43
weighted avg	0.77	0.81	0.79	43

CODE

```
fpr1, tpr1, thresholds = roc_curve(y_test, y_pred1)
roc_auc1 = auc(fpr1,tpr1)
plt.title('Receiver Operating Characteristic')
plt.plot(fpr1, tpr1, 'b',label='AUC = %0.3f'% roc_auc1)
plt.legend(loc='lower right')
plt.plot([0,1],[0,1],'r--')
plt.xlim([-0.1,1.0])
plt.ylim([-0.1,1.01])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

OUTPUT



CODE

```
# Naive Bayes Classifier
from sklearn.naive_bayes import GaussianNB
classifier2 = GaussianNB()
classifier2.fit(X_train, y_train)
```

OUTPUT

```
▾ GaussianNB
GaussianNB()
```

CODE

```
y_pred2 = classifier2.predict(X_test)
accuracy = classifier2.score(X_test, y_test)
print("Accuracy:", accuracy)
print(confusion_matrix(y_test, y_pred2))
print(classification_report(y_test, y_pred2))
```

OUTPUT

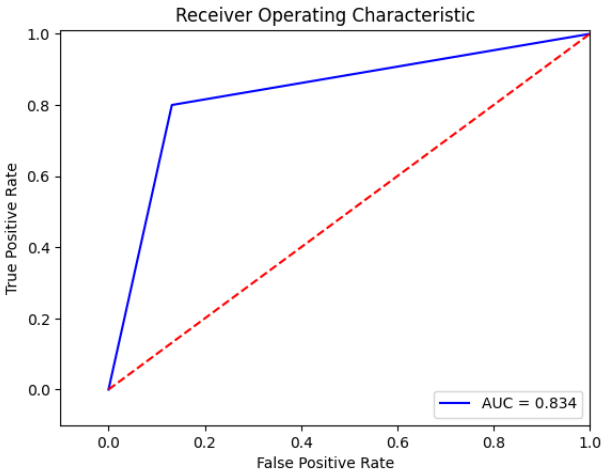
```
Accuracy: 0.8604651162790697

[[33  5]
 [ 1  4]]

              precision    recall  f1-score   support

           0       0.97      0.87      0.92        38
           1       0.44      0.80      0.57         5

    accuracy                           0.86        43
   macro avg       0.71      0.83      0.74        43
weighted avg       0.91      0.86      0.88        43
```

CODE

```
fpr2, tpr2, thresholds = roc_curve(y_test, y_pred2)
roc_auc2 = auc(fpr2,tpr2)
plt.title('Receiver Operating Characteristic')
plt.plot(fpr2, tpr2, 'b',label='AUC = %0.3f'% roc_auc2)
plt.legend(loc='lower right')
plt.plot([0,1],[0,1],'r--')
plt.xlim([-0.1,1.0])
plt.ylim([-0.1,1.01])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
```

plt.show()

OUTPUT



CODE

```
# Decision tree Classifier
from sklearn.tree import DecisionTreeClassifier
classifier3=DecisionTreeClassifier()
classifier3.fit(X_train,y_train)
```

OUTPUT

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

CODE

```
y_pred3 = classifier3.predict(X_test)
accuracy = classifier3.score(X_test, y_test)
print("Accuracy:", accuracy)
print(confusion_matrix(y_test, y_pred3))
print(classification_report(y_test, y_pred3))
```

OUTPUT

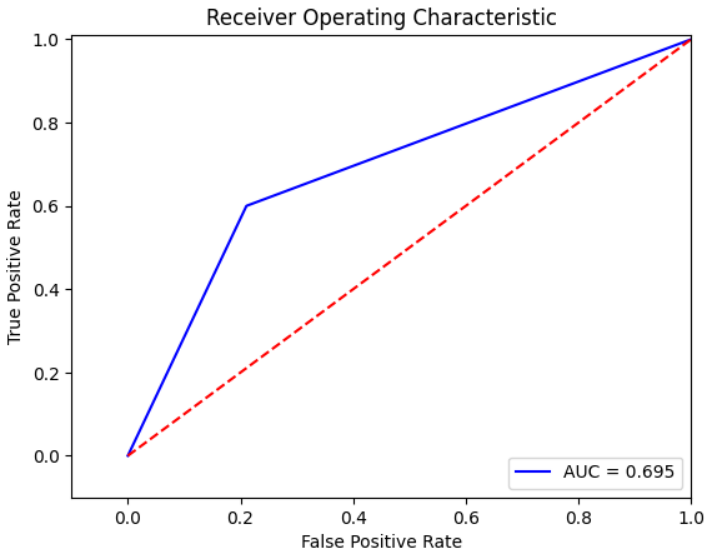
Accuracy: 0.7441860465116279

```
[[29  9]
 [ 2  3]]
              precision    recall  f1-score   support

           0       0.94      0.76      0.84        38
           1       0.25      0.60      0.35         5

    accuracy                           0.74        43
   macro avg       0.59      0.68      0.60        43
weighted avg       0.86      0.74      0.78        43
```

CODE

```
fpr3, tpr3, thresholds = roc_curve(y_test, y_pred3)
roc_auc3 = auc(fpr3,tpr3)
plt.title('Receiver Operating Characteristic')
plt.plot(fpr3, tpr3, 'b',label='AUC = %0.3f'% roc_auc3)
plt.legend(loc='lower right')
plt.plot([0,1],[0,1],'r--')
plt.xlim([-0.1,1.0])
plt.ylim([-0.1,1.01])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

OUTPUT



CODE

```
# Logistic Regression
from sklearn.linear_model import LogisticRegression
classifier4 = LogisticRegression(random_state = 0, solver='lbfgs', multi_class='auto')
classifier4.fit(X_train, y_train)
```

OUTPUT

```
LogisticRegression
LogisticRegression(random_state=0)
```

CODE

```
y_pred4 = classifier4.predict(X_test)
accuracy = classifier4.score(X_test, y_test)
print("Accuracy:", accuracy)
print(confusion_matrix(y_test, y_pred4))
print(classification_report(y_test, y_pred4))
```

OUTPUT

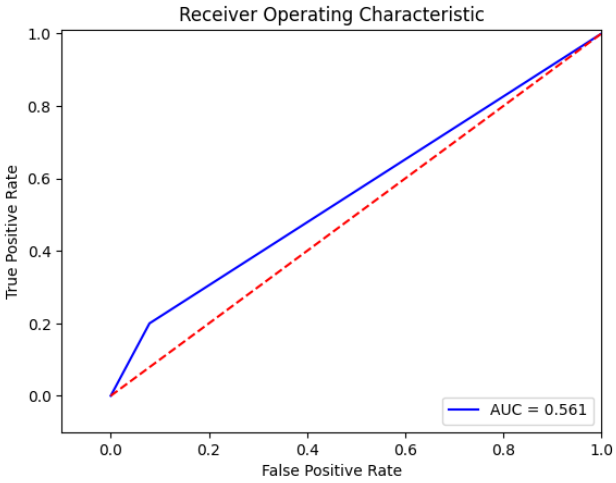
```
Accuracy: 0.8372093023255814

[[35  3]
 [ 4  1]]
              precision    recall  f1-score   support

           0       0.90      0.92      0.91        38
           1       0.25      0.20      0.22         5

    accuracy                           0.84        43
   macro avg       0.57      0.56      0.57        43
weighted avg       0.82      0.84      0.83        43
```

CODE

```
fpr4, tpr4, thresholds = roc_curve(y_test, y_pred4)
roc_auc4 = auc(fpr4,tpr4)
plt.title('Receiver Operating Characteristic')
plt.plot(fpr4, tpr4, 'b',label='AUC = %0.3f'% roc_auc4)
plt.legend(loc='lower right')
plt.plot([0,1],[0,1],'r--')
plt.xlim([-0.1,1.0])
plt.ylim([-0.1,1.01])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
```

```
plt.show()
```

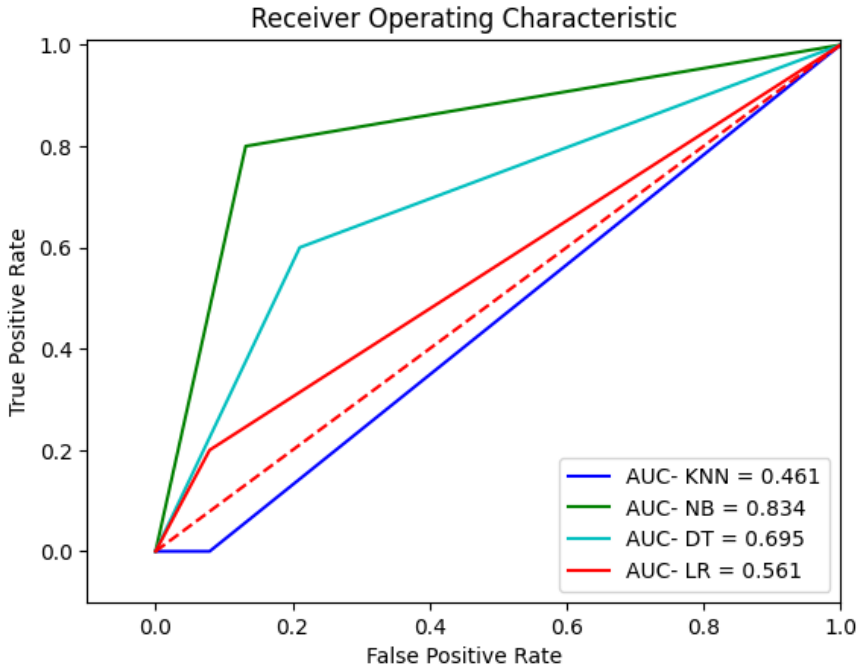
OUTPUT



CODE

```
#Comparison of 4 models by means of ROC curve
fpr4, tpr4, thresholds = roc_curve(y_test, y_pred4)
roc_auc4 = auc(fpr4,tpr4)
plt.title('Receiver Operating Characteristic')
plt.plot(fpr1, tpr1, 'b',label='AUC- KNN = %0.3f'% roc_auc1)
plt.plot(fpr2, tpr2, 'g',label='AUC- NB = %0.3f'% roc_auc2)
plt.plot(fpr3, tpr3, 'c',label='AUC- DT = %0.3f'% roc_auc3)
plt.plot(fpr4, tpr4, 'r',label='AUC- LR = %0.3f'% roc_auc4)
plt.legend(loc='lower right')
plt.plot([0,1],[0,1],'r--')
plt.xlim([-0.1,1.0])
plt.ylim([-0.1,1.01])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

OUTPUT


Receiver Operating Characteristic
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
AUC- KNN = 0.461
AUC- NB = 0.834
AUC- DT = 0.695
AUC- LR = 0.561

EXPERIMENT NO :19

AIM: Classification using SVM for iris data

CODE

```
# Importing the libraries
import pandas as pd
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
# Importing the dataset
df = pd.read_csv("iris.csv")
X = df.drop('variety', axis=1)
y = df.variety
print ("Number of data points ::", X.shape[0])
print("Number of features ::", X.shape[1])
```

OUTPUT

```
Number of data points :: 150
Number of features :: 4
```

CODE

```
#Using Standard Scaler to transform the data.
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(
                    X_scaled, y, test_size=0.2, random_state=42)
#Create the Non Linear SVM model
from sklearn.svm import SVC
classifier = SVC(kernel = 'linear', random_state = 0)
#Fit the model for the data
classifier.fit(X_train, y_train)
```

OUTPUT

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

CODE

```
#Make the prediction
y_pred = classifier.predict(X_test)
print("Accuracy:",accuracy_score(y_test, y_pred))
print(confusion_matrix(y_test,y_pred))
print("Classification report - \n", classification_report(y_test,y_pred))
```

OUTPUT

```
Accuracy: 0.9666666666666667

[[10 3 0]
 [ 0 8 1]
 [ 0 0 11]]

Classification report -
              precision    recall  f1-score   support

      Setosa       1.00      1.00      1.00        10
  Versicolor       1.00      0.89      0.94         9
   Virginica       0.92      1.00      0.96        11

    accuracy                           0.97        30
   macro avg       0.97      0.96      0.97        30
weighted avg       0.97      0.97      0.97        30
```

CODE

```
# Create the SVM model using LinearSVC
from sklearn.svm import LinearSVC
clf = LinearSVC(random_state = 0)
#Fit the model for the data
clf.fit(X_train, y_train)
```

OUTPUT

```
LinearSVC
LinearSVC(random_state=0)
```

CODE

```
#Make the prediction
y_pred1 = clf.predict(X_test)
print("Accuracy:",accuracy_score(y_test, y_pred1))
print(confusion_matrix(y_test,y_pred1))
print("Classification report - \n", classification_report(y_test,y_pred1))
```

OUTPUT

Accuracy: 1.0

```
[[10 3 0]
 [ 0 9 1]
 [ 0 0 11]]
```

Classification report -

	precision	recall	f1-score	support
Setosa	1.00	1.00	1.00	10
Versicolor	1.00	1.00	1.00	9
Virginica	1.00	1.00	1.00	11
accuracy			1.00	30
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30

EXPERIMENT NO :20

AIM: Classification of Fruit data using SVM

CODE

```
# Importing the libraries
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
# Importing the dataset
data = pd.read_csv("apples_and_oranges.csv")
training_set, test_set = train_test_split(
                                        data, test_size = 0.2, random_state = 1)
X_train = training_set.iloc[:,0:2].values
Y_train = training_set.iloc[:,2].values
X_test = test_set.iloc[:,0:2].values
Y_test = test_set.iloc[:,2].values

#Use of SVC with kernal='rbf'
from sklearn.svm import SVC
classifier = SVC(kernel='rbf', random_state = 1)
classifier.fit(X_train,Y_train)
```

OUTPUT

```
SVC
SVC(random_state=1)
```

CODE

```
Y_pred = classifier.predict(X_test)
test_set["Predictions"] = Y_pred
print("Accuracy:",accuracy_score(Y_test,Y_pred) )
print("Classification report - \n", classification_report(Y_test,Y_pred))
ax = plt.axes()
df_cm = cm
sns.heatmap(df_cm, annot=True, annot_kws={"size": 30},
                                            fmt='d',cmap="Blues", ax = ax )
```

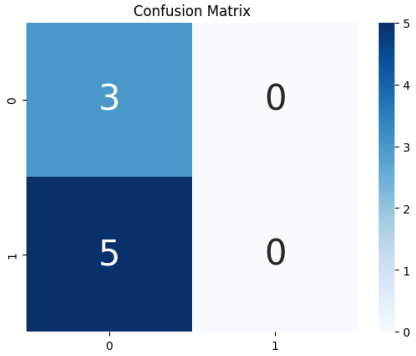
```
ax.set_title('Confusion Matrix')
plt.show()
```

OUTPUT

Accuracy: 0.375

Classification report -

	precision	recall	f1-score	support
apple	0.38	1.00	0.55	3
orange	0.00	0.00	0.00	5
accuracy			0.38	8
macro avg	0.19	0.50	0.27	8
weighted avg	0.14	0.38	0.20	8



CODE

```
#Use of SVC with kernal='linear'
classifier1 = SVC(kernel='linear', random_state = 1)
classifier1.fit(X_train,Y_train)
Y_pred1 = classifier1.predict(X_test)
ax = plt.axes()
df_cm = cm1
sns.heatmap(df_cm, annot=True, annot_kws={"size": 30},
                                                    fmt='d',cmap="Blues", ax = ax )
ax.set_title('Confusion Matrix')
plt.show()
```

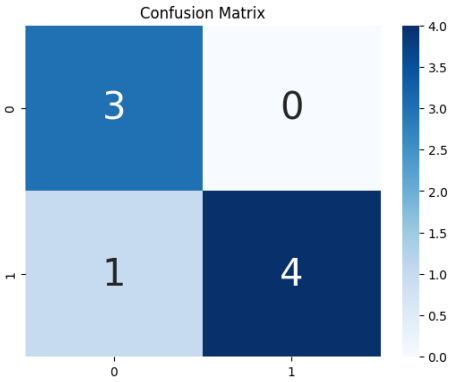
OUTPUT

```
SVC
SVC(kernel='linear', random_state=1)
```

Accuracy: 0.875

Classification report -

	precision	recall	f1-score	support
apple	0.75	1.00	0.86	3
orange	1.00	0.80	0.89	5
accuracy			0.88	8
macro avg	0.88	0.90	0.87	8
weighted avg	0.91	0.88	0.88	



CODE

```
plt.figure(figsize = (7,7))
X_set, y_set = X_train, Y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01), np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape), alpha = 0.75, cmap = ListedColormap(('black', 'white')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
  plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'orange'))(i), label = j)
plt.title('Apples Vs Oranges')
plt.xlabel('Weight In Grams')
plt.ylabel('Size in cm')
plt.legend()
plt.show()
```

OUTPUT


Apples Vs Oranges
0
1
Size in cm
Weight In Grams
6.5
6.0
5.5
5.0
4.5
4.0
3.5
64
66
68
70
72
74

EXPERIMENT NO :21

AIM: Using k-means clustering on dataset and plot a graph

CODE

```
import numpy as np
import pandas as pd
%matplotlib inline
import matplotlib.pyplot as plt
dataset= pd.read_csv('./CC GENERAL.csv')
print(dataset.isnull().sum())
```

OUTPUT

```
CUST_ID                             0
BALANCE                             0
BALANCE_FREQUENCY                   0
PURCHASES                           0
ONEOFF_PURCHASES                    0
INSTALLMENTS_PURCHASES              0
CASH_ADVANCE                        0
PURCHASES_FREQUENCY                 0
ONEOFF_PURCHASES_FREQUENCY          0
PURCHASES_INSTALLMENTS_FREQUENCY    0
CASH_ADVANCE_FREQUENCY              0
CASH_ADVANCE_TRX                    0
PURCHASES_TRX                       0
CREDIT_LIMIT                        1
PAYMENTS                            0
MINIMUM_PAYMENTS                  313
PRC_FULL_PAYMENT                    0
TENURE                              0
dtype: int64
```

CODE

```
dataset['CREDIT_LIMIT'].fillna(dataset.CREDIT_LIMIT.mean(), inplace = True)
dataset['MINIMUM_PAYMENTS'].fillna(dataset.MINIMUM_PAYMENTS.mean(), inplace =
                                                                    True)
dataset.drop(['CUST_ID'], axis= 1, inplace = True)
X = dataset.iloc[:,:].values
from sklearn.preprocessing import StandardScaler
standardscaler= StandardScaler()
X = standardscaler.fit_transform(X)
print(X)
```

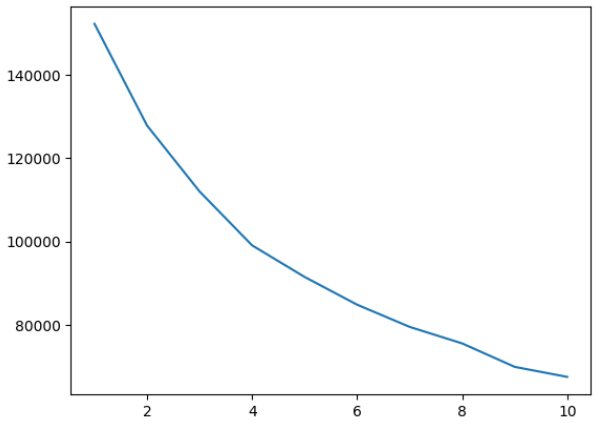
OUTPUT

```
[[-0.73198937 -0.24943448 -0.42489974 ... -0.31096755 -0.52555097
  0.36067954]

.....

[-0.57257511 -0.88903307  0.04214581 ... -0.33294642 -0.52555097
 -4.12276757]]
```

CODE

```
from sklearn.cluster import KMeans
wss= []
for i in range(1, 11):
    kmeans= KMeans(n_clusters = i, init = 'k-means++', random_state = 0)
    kmeans.fit(X)
    wss.append(kmeans.inertia_)
plt.plot(range(1,11), wss)
```

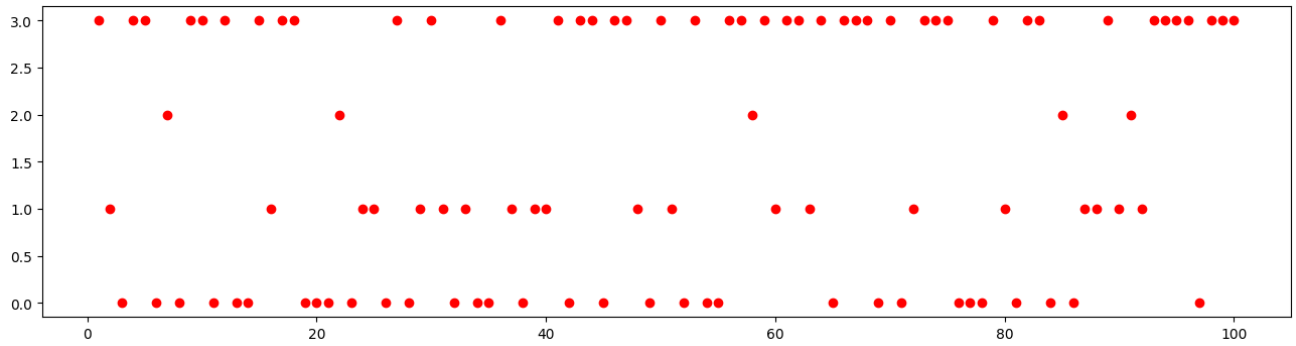
OUTPUT



CODE

```
kmeans = KMeans(n_clusters = k, init= 'k-means++', random_state = 0)
kmeans.fit(X)
Y_pred_K= kmeans.predict(X)
print(Y_pred_K)
plt.figure(figsize=(16,4))
plt.plot(range(1,100+1),Y_pred_K[:100],'ro')
```

OUTPUT



EXPERIMENT NO :22

AIM: K-means clustering on iris data

CODE

```
import numpy as np
from sklearn.cluster import KMeans
from sklearn.datasets import load_iris
%matplotlib inline
import matplotlib.pyplot as plt

iris = load_iris()
X = iris.data
kmeans= KMeans(n_clusters = 3, init = 'k-means++', random_state = 0)
kmeans.fit(X)
Y_pred_K= kmeans.predict(X)
print(Y_pred_K)
```

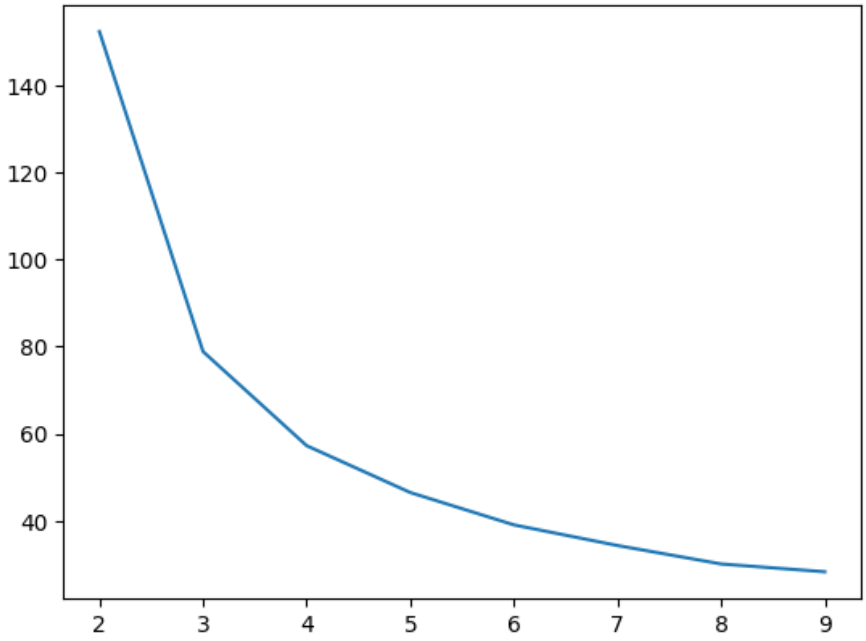
OUTPUT

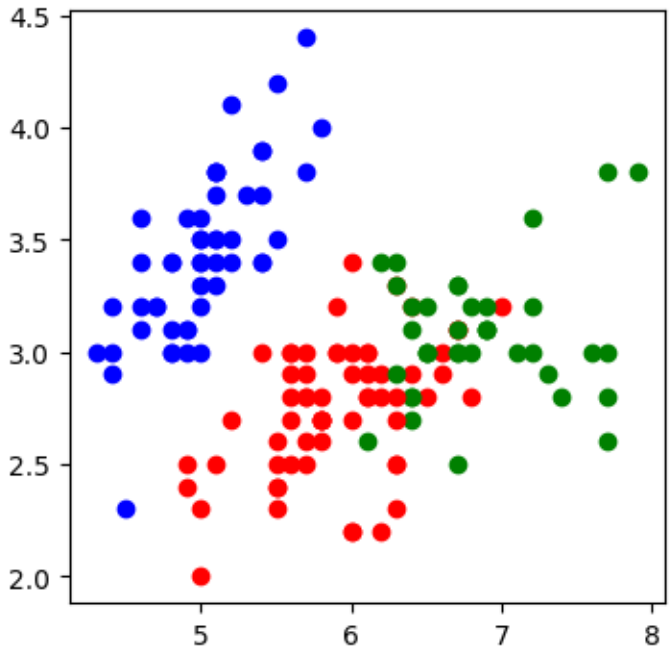
```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 2 2 2 2 0 2 2 2 2
 2 2 0 0 2 2 2 2 0 2 0 2 0 2 2 0 0 2 2 2 2 2 0 2 2 2 2 0 2 2 2 0 2 2 2 0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

CODE

```
inertia = []
ax = []
for i in range(2,10):
    ax.append(i)
    kmeans= KMeans(n_clusters = i, init = 'k-means++', random_state = 0)
    kmeans.fit(X)
    inertia.append(kmeans.inertia_)
plt.plot(ax,inertia)


kmeans= KMeans(n_clusters = 3, init = 'k-means++', random_state = 0)
kmeans.fit(X)
plt.figure(figsize=(4,4))
Y_pred_K = kmeans.predict(X)
colors = ['red','blue','green','yellow','cyan']
for x,y in zip(X,Y_pred_K):
    plt.scatter(x[0],x[1],color = colors[y])
```

OUTPUT





EXPERIMENT NO :23

AIM: Clustering using k-means on Array values

CODE

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
x1=10*np.random.rand(100,2)
x1.shape
kmean=KMeans(n_clusters=3)
kmean.fit(x1)
```

OUTPUT

(100, 2)

```
KMeans
KMeans(n_clusters=3)
```

CODE

```
kmean.cluster_centers_
```

OUTPUT

array([[1.89434548, 7.07603256], [4.78173856, 2.35812311], [6.46432365, 8.01961652]])

CODE

```
kmean.labels_
```

OUTPUT

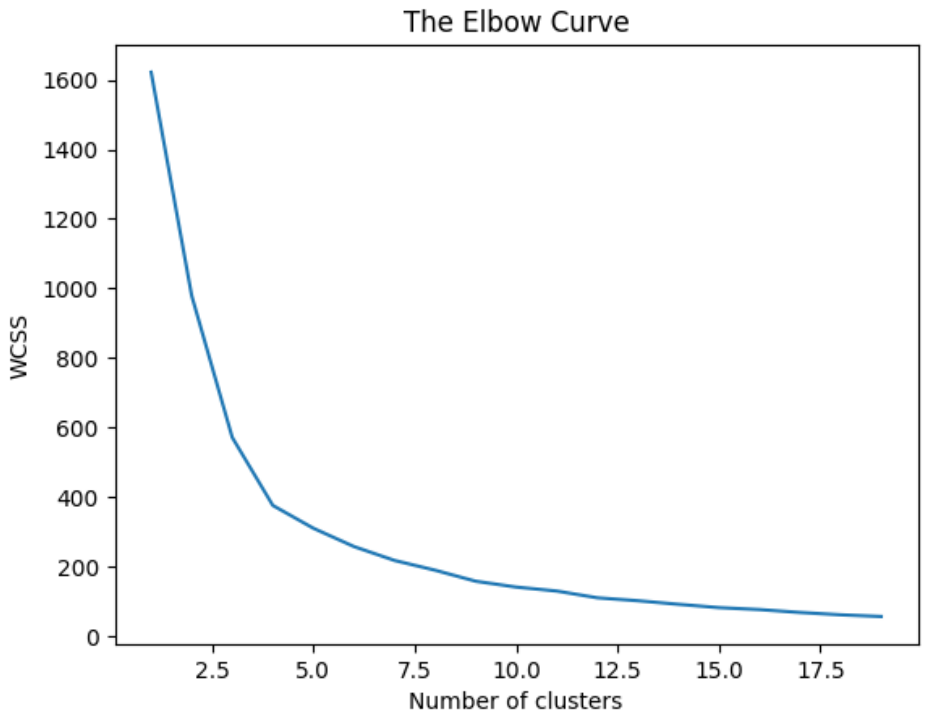
array([1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 2, 1, 0, 2, 2, 2, 0, 0, 1, 1, 1, 1, 1, 2, 1, 2, 0, 1, 1, 2, 2, 2, 2, 2, 2, 1, 1, 2, 2, 0, 2, 1, 2, 2, 0, 0, 0, 1, 2, 0, 1, 2, 0, 1, 1, 1, 2, 2, 1, 1, 2, 0, 0, 2, 2, 0, 1, 1, 0, 1, 2, 1, 1, 1, 2, 1, 0, 0, 0, 0, 2, 1, 1, 0, 1, 1, 1, 0, 2, 2, 1, 0], dtype=int32)

CODE

```
wcss = []
for i in range(1,20):
 kmeans = KMeans(n_clusters=i,init= 'k-means++',max_iter=300,n_init=10,random_state=0)
 kmeans.fit(x1)
 wcss.append(kmeans.inertia_)
 print('Cluster', i, 'Inertia', kmeans.inertia_)
plt.plot(range(1,20),wcss)
plt.title('The Elbow Curve')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS') ##WCSS stands for total within-cluster sum of square
plt.show()
```

OUTPUT

```
Cluster 1 Inertia 1621.5610439853108
Cluster 2 Inertia 979.2750965451323
Cluster 3 Inertia 571.8725041998405
Cluster 4 Inertia 376.51322630137406
Cluster 5 Inertia 310.9209665248758
Cluster 6 Inertia 258.33695345828636
Cluster 7 Inertia 218.39362280255392
Cluster 8 Inertia 190.3502832140186
Cluster 9 Inertia 158.69900691645432
Cluster 10 Inertia 141.82177619385274
Cluster 11 Inertia 130.48037026726755
Cluster 12 Inertia 111.02532496788564
Cluster 13 Inertia 102.61817248811872
Cluster 14 Inertia 92.34445308678943
Cluster 15 Inertia 82.79415350583423
Cluster 16 Inertia 77.13096946511988
Cluster 17 Inertia 68.88110169270776
Cluster 18 Inertia 62.126506822214594
Cluster 19 Inertia 57.31168693060516
```


The Elbow Curve
WCSS
1600
1400
1200
1000
800
600
400
200
0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Number of clusters

EXPERIMENT NO :24

AIM: Chance to quit company using MLP Classifier

CODE

```
import pandas as pd
from sklearn import preprocessing
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
data=pd.read_csv('HR_comma_sep (1).csv')
data.head()
```

OUTPUT

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Work_accident	left	promotion_last_5years	sales	salary
0	0.38	0.53	2	157	3	0	1	0	sales	low
1	0.80	0.86	5	262	6	0	1	0	sales	medium
2	0.11	0.88	7	272	4	0	1	0	sales	medium
3	0.72	0.87	5	223	5	0	1	0	sales	low
4	0.37	0.52	2	159	3	0	1	0	sales	low

CODE

```
le = preprocessing.LabelEncoder()
data['salary']=le.fit_transform(data['salary'])
data['sales']=le.fit_transform(data['sales'])
X=data[['satisfaction_level', 'last_evaluation', 'number_project',
        'average_montly_hours', 'time_spend_company', 'Work_accident',
        'promotion_last_5years', 'sales', 'salary']]
y=data['left']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

CODE

```
from sklearn.neural_network import MLPClassifier
clf = MLPClassifier(hidden_layer_sizes=(6,5),
                    random_state=5,
                    verbose=False,
                    learning_rate_init=0.01)
clf.fit(X_train,y_train)
```

OUTPUT

```
MLPClassifier(hidden_layer_sizes=(6, 5), learning_rate_init=0.01,
              random_state=5)
```

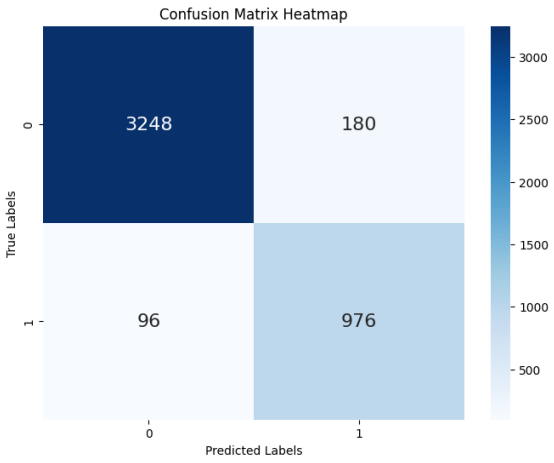
CODE

```
import seaborn as sns
import matplotlib.pyplot as plt
ypred=clf.predict(X_test)
print ("Accuracy:",accuracy_score(y_test,ypred))
print(classification_report(y_test, ypred))
cm = confusion_matrix(y_test, ypred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', annot_kws={"size": 16})
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix Heatmap')
plt.show()
```

OUTPUT

Accuracy: 0.9386666666666666

	precision	recall	f1-score	support
0	0.97	0.95	0.96	3428
1	0.84	0.91	0.88	1072
accuracy			0.94	4500
macro avg	0.91	0.93	0.92	4500
weighted avg	0.94	0.94	0.94	4500



EXPERIMENT NO :25

AIM: Gate Implementation using Perceptron

CODE

```
from sklearn.linear_model import Perceptron
# Learning Perceptron for AND gate
X_train = [[0,0],[0,1],[1,0],[1,1]]
y_train = [0,0,0,1]
clf= Perceptron(tol=1e-3,random_state=0)
clf.fit(X_train, y_train)
```

OUTPUT

```
Perceptron
Perceptron()
```

CODE

```
y_pred = clf.predict(X_train)
print(y_pred)
print(clf.coef_, clf.intercept_)
```

OUTPUT

```
[0 0 0 1]
[[2. 2.]] [-2.]
```

CODE

```
# Learning Perceptron for OR gate
X_train = [[0,0],[0,1],[1,0],[1,1]]
y_train1 = [0,1,1,1]
clf= Perceptron(tol=1e-3,random_state=0)
clf.fit(X_train, y_train1)
y_pred1 = clf.predict(X_train)
print(y_pred1)
print(clf.coef_, clf.intercept_)
```

OUTPUT

```
[0 1 1 1]
[[2. 2.]] [-1.]
```

CODE

```
# Learning Perceptron for NAND gate
X_train = [[0,0],[0,1],[1,0],[1,1]]
y_train2 = [1,1,1,0]
clf= Perceptron(tol=1e-3,random_state=1)
clf.fit(X_train, y_train2)
y_pred2 = clf.predict(X_train)
print(y_pred2)
print(clf.coef_, clf.intercept_)
```

OUTPUT

```
[1 1 1 0]
[[-2. -2.]] [3.]
```

CODE

```
# Learning Perceptron for NOR gate
X_train = [[0,0],[0,1],[1,0],[1,1]]
y_train3 = [1,0,0,0]
clf= Perceptron(tol=1e-3,random_state=0)
clf.fit(X_train, y_train3)
y_pred3 = clf.predict(X_train)
print(y_pred3)
print(clf.coef_, clf.intercept_)
```

OUTPUT

```
[1 0 0 0]
[[-2. -2.]] [1.]
```

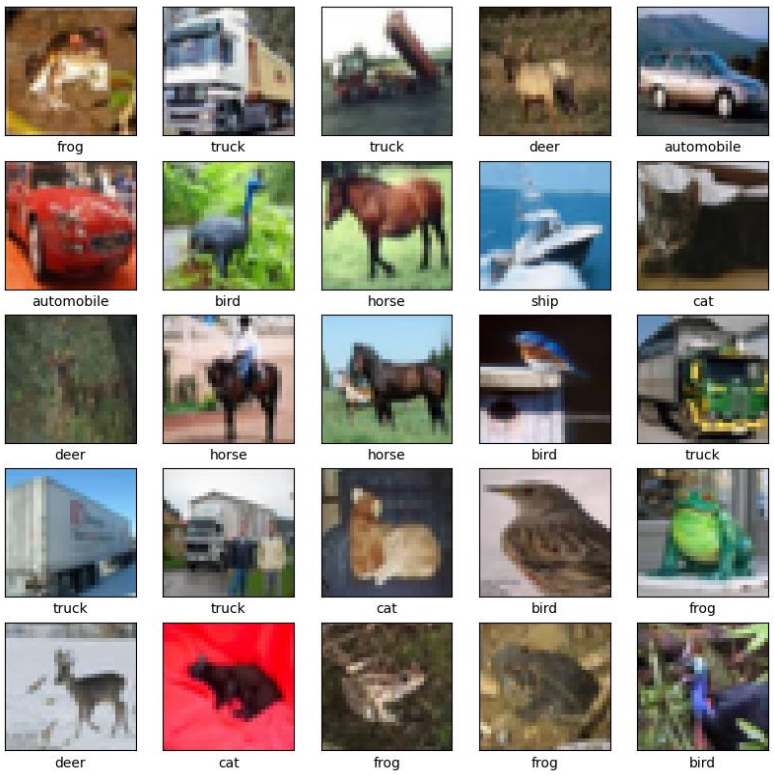
EXPERIMENT NO :26

AIM: Image classification using CNN on Cifar10 dataset

CODE

```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
%matplotlib inline
#The CIFAR10 dataset contains 60,000 color images in 10 classes, with 6,000 #images in each class
(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()
train_images, test_images = train_images / 255.0, test_images / 255.0
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i])
    plt.xlabel(class_names[train_labels[i][0]])
```

OUTPUT



CODE

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                                                    input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.summary()
```

OUTPUT

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 32)        896

 max_pooling2d (MaxPooling2  (None, 15, 15, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 13, 13, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 6, 6, 64)          0
 g2D)

 conv2d_2 (Conv2D)           (None, 4, 4, 64)          36928

=================================================================
Total params: 56320 (220.00 KB)
Trainable params: 56320 (220.00 KB)
Non-trainable params: 0 (0.00 Byte)
```

CODE

```
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10))
model.summary()
```

OUTPUT

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 32)        896

 max_pooling2d (MaxPooling2  (None, 15, 15, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 13, 13, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 6, 6, 64)          0
 g2D)

 conv2d_2 (Conv2D)           (None, 4, 4, 64)          36928

 flatten (Flatten)           (None, 1024)              0

 dense (Dense)               (None, 64)                65600

 dense_1 (Dense)             (None, 10)                650

=================================================================
Total params: 122570 (478.79 KB)
Trainable params: 122570 (478.79 KB)
Non-trainable params: 0 (0.00 Byte)
```

CODE

```
model.compile(optimizer='adam',
   loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
   metrics=['accuracy'])
history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))
# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
print(f'Test accuracy: {test_acc}')
```

OUTPUT

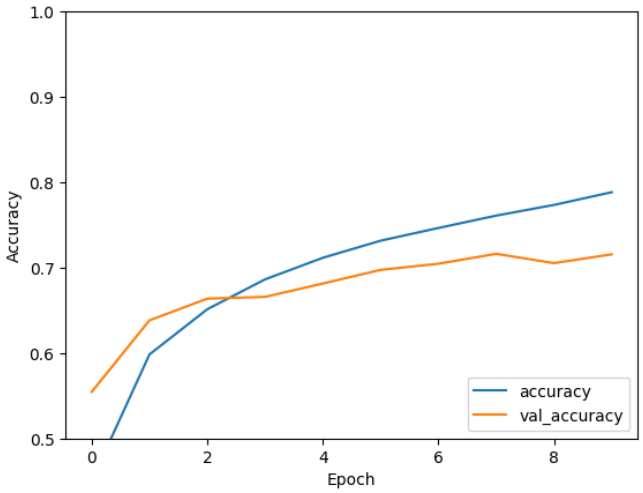
```
Epoch 10/10 1563/1563 [==============================] - 70s 44ms/step - loss: 0.6037 -
accuracy: 0.7884 - val_loss: 0.8472 - val_accuracy: 0.7156
Test accuracy: 0.6276000142097473
```

CODE

```
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')
plt.show()
```

OUTPUT

313/313 - 4s - loss: 0.8472 - accuracy: 0.7156 - 4s/epoch - 12ms/step



EXPERIMENT NO :27

AIM: Sentiment identification using NLP

CODE

```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize, sent_tokenize
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
stop_words = set(stopwords.words('english'))
txt = "Hello. MCA S3 is fantastic. We learn many new concepts and implement them in our practical exams. "\
"1st of all the data science is a new paper."
tokenized = sent_tokenize(txt)
for i in tokenized:
    wordsList = nltk.word_tokenize(i)
    wordsList = [w for w in wordsList if not w in stop_words]
    tagged = nltk.pos_tag(wordsList)
    print(tagged)
```

OUTPUT

```
[('Hello', 'NNP'), ('.', '.')]
[('MCA', 'NNP'), ('S3', 'NNP'), ('fantastic', 'JJ'), ('.', '.')]
[('We', 'PRP'), ('learn', 'VBP'), ('many', 'JJ'), ('new', 'JJ'), ('concepts', 'NNS'), ('implement', 'JJ'), ('practical', 'JJ'), ('exams', 'NN'), ('.', '.')][('1st', 'CD'), ('data', 'NNS'), ('science', 'NN'), ('new', 'JJ'), ('paper', 'NN'), ('.', '.')]
```

CODE

```
#N-gram model
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.style.use(style='seaborn')
colnames=['Sentiment', 'news']
df=pd.read_csv('all-data.csv',encoding = "ISO-8859-1", names=colnames,
                                                    header = None)
df.head()
```

OUTPUT

	Sentiment	news
0	neutral	According to Gran , the company has no plans t...
1	neutral	Technopolis plans to develop in stages an area...
2	negative	The international electronic industry company ...
3	positive	With the new production plant the company woul...
4	positive	According to the company 's updated strategy f...

CODE

```
df['Sentiment'].value_counts()
```

OUTPUT

```
neutral 2879
positive 1363
negative 604
Name: Sentiment, dtype: int64
```

CODE

```
x=df['news'].values
y=df['Sentiment'].values
from sklearn.model_selection import train_test_split
(x_train,x_test,y_train,y_test)=train_test_split(x,y,test_size=0.4)
df1=pd.DataFrame(x_train)
df1=df1.rename(columns={0:'news'})
df2=pd.DataFrame(y_train)
df2=df2.rename(columns={0:'sentiment'})
df_train=pd.concat([df1,df2],axis=1)
df_train.head()
```

OUTPUT

	news	sentiment
0	Finnish automation solutions developer Cencorp...	negative
1	Finnish textiles and clothing group Marimekko ...	negative
2	Nokia will continue to invest in future develo...	positive
3	This order , when delivered , will bring the t...	neutral
4	The circuit 's overall production rate on a we...	positive

CODE

```
df3=pd.DataFrame(x_test)
df3=df3.rename(columns={0:'news'})
df4=pd.DataFrame(y_test)
df4=df2.rename(columns={0:'sentiment'})
df_test=pd.concat([df3,df4],axis=1)
df_test.head()
```

OUTPUT

	news	sentiment
0	The use case dramatically narrows if you go on...	negative
1	Employing 112 in Finland and 280 abroad , the ...	negative
2	Tieto 's service is also used to send , proces...	positive
3	The share subscription period will expire on 3...	neutral
4	While I cant understand what theyre saying , i...	positive

CODE

```
import string
string.punctuation
def remove_punctuation(text):
  if(type(text)==float):
    return text
  ans=""
  for i in text:
    if i not in string.punctuation:
      ans+=i
  return ans
df_train['news']= df_train['news'].apply(lambda x:remove_punctuation(x))
df_test['news']= df_test['news'].apply(lambda x:remove_punctuation(x))
df_train.head()
```

OUTPUT

	news	sentiment
0	Finnish automation solutions developer Cencorp...	negative
1	Finnish textiles and clothing group Marimekko ...	negative
2	Nokia will continue to invest in future develo...	positive
3	This order when delivered will bring the tot...	neutral
4	The circuit s overall production rate on a wee...	positive

CODE

```
def generate_N_grams(text,ngram=1):
  words=[word for word in text.split(" ") if word not in set(stopwords.words('english'))]
  print("Sentence after removing stopwords:",words)
  temp=zip(*[words[i:] for i in range(0,ngram)])
  ans=[' '.join(ngram) for ngram in temp]
  return ans
```

CODE

```
print(generate_N_grams("The sun rises in the east",2))
print(generate_N_grams("The sun rises in the east",3))
print(generate_N_grams("The sun rises in the east",4))
```

OUTPUT

```
Sentence after removing stopwords: ['The', 'sun', 'rises', 'east']
['The sun', 'sun rises', 'rises east']

Sentence after removing stopwords: ['The', 'sun', 'rises', 'east']
['The sun rises', 'sun rises east']

Sentence after removing stopwords: ['The', 'sun', 'rises', 'east']
['The sun rises east']
```