Data Types- Integer, String, Float, Complex, Boolean

Variable is a memory container use for store values into the memory location

In [6]:

```
EmpCode=1001
print(type(EmpCode))
print(EmpCode)
```

```
<class 'int'>
1001
```

In [8]:

```
EmpName='James'
print(EmpName)
print(type(EmpName))
```

```
James
<class 'str'>
```

In [12]:

```
Salary=55000.25
print(Salary)
print(type(Salary))
```

```
55000.25
<class 'float'>
```

In [15]:

```
#First is Real and Second is Imaginary Number
Cmpx_Number=5+4j
print(Cmpx_Number)
print(type(Cmpx_Number))
```

```
(5+4j)
<class 'complex'>
```

In [19]:

```
#Boolean show True or False only
Result=True
print(Result)
print(type(Result))
```

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

Data Structure with Python - List[], Tuple(), Set{}, Dictionery{key,value}

List is Mutable means it can be changed, but slow as compare to tuple, also have lots of in-build function as compare to tulpe

In [22]:

```
EmpInfo=[1001,"Gita",50000.15,250001]
print(EmpInfo)
print(type(EmpInfo))
```

```
[1001, 'Gita', 50000.15, 250001]
<class 'list'>
```

In [25]:

```
#We printing list values according to index number, index number started from zero
print(EmpInfo[0])
print(EmpInfo[1])
```

```
1001
Gita
```

In [34]:

```
color=["red","green","blue","pink"]
print(color)
```

```
['red', 'green', 'blue', 'pink']
```

List Slicing

In [41]:

```
# :(colon) work for range purpose only
print(color[0:3])
print(color[1:4])
print(color[2:])
print(color[:2])
```

```
['red', 'green', 'blue']
['green', 'blue', 'pink']
['blue', 'pink']
['red', 'green']
['red', 'green', 'blue', 'pink']
```

In [42]:

```
print(color[::-1])  #-1 use for reverse the list
```

```
['pink', 'blue', 'green', 'red']
```

In [46]:

```
Numbers=[11,22,33,44,55,66,77,88,99]
print(Numbers[::2])
```

```
[11, 33, 55, 77, 99]
```

In [47]:

```
print(Numbers[0:len(Numbers):2])  #len is length use for calculate the length
```

```
[11, 33, 55, 77, 99]
```

In [53]:

```
#built function of list
mylist=[101,102,103,104,105]
print(mylist)
mylist.reverse()
print(mylist)
```

```
[101, 102, 103, 104, 105]
[105, 104, 103, 102, 101]
```

In [55]:

```
#Mutable we can change values
mylist[0]=5001
print(mylist)
```

```
[5001, 104, 103, 102, 101]
```

Tuple is Immutable we can not change after assign. but fast as compare to list

In [57]:

```
Std_Info=(101,"Anita",80.21,"Delhi")
print(Std_Info)
print(type(Std_Info))
```

```
(101, 'Anita', 80.21, 'Delhi')
<class 'tuple'>
```

In [60]:

```
# Std_Info[1]="Gita"  we can not change
print(Std_Info[1])
```

```
Anita
```

Set show only Unique Values

In [61]:

```
Emp_Id={1001,1002,1002,1003,1004,1004,1005}
print(Emp_Id)
```

```
{1001, 1002, 1003, 1004, 1005}
```

Dictionery store data into key,Value pair

In [77]:

```
Employee={101:"James",102:"Anita",103:"Ritu",104:"Ram"}
print(Employee)
print(type(Employee))
```

```
{101: 'James', 102: 'Anita', 103: 'Ritu', 104: 'Ram'}
<class 'dict'>
```

In [67]:

```
print(Employee.keys())
```

```
dict_keys([101, 102, 103, 104])
```

In [70]:

```
print(Employee.values())
```

```
dict_values(['James', 'Anita', 'Ritu', 'Ram'])
```

In [76]:

```
print(Employee[102])
```

```
Anita
```

Exercise for store Five students information

In [80]:

```
Emp_Record={
           101:["Ram",65000,"Meerut","IT Mangaer"],
           102:["Gita",70000,"Delhi","Data Analyst"],
           103:["James",45000,"Kanpur","Business Manager"],
           104:["Kavita",90000,"Banglore","Full Stack"]
           }
print(Emp_Record[102])
```

```
['Gita', 70000, 'Delhi', 'Data Analyst']
```

In [85]:

```
print(Emp_Record[101])
print(Emp_Record[102])
print(Emp_Record[103])
print(Emp_Record[104])
```

```
['Ram', 65000, 'Meerut', 'IT Mangaer']
['Gita', 70000, 'Delhi', 'Data Analyst']
['James', 45000, 'Kanpur', 'Business Manager']
['Kavita', 90000, 'Banglore', 'Full Stack']
```

String Slicing

In [95]:

```
State="uttar pradesh"
print("Normal - ",State)
print("Upper -  ",State.upper())
print("Lower - ",State.lower())
print("title - ",State.title())
print("Length - ",len(State))
```

```
Normal -  uttar pradesh
Upper -   UTTAR PRADESH
Lower -  uttar pradesh
title -  Uttar Pradesh
Length -  13
```

Some More List Functions

In [101]:

```
lst1=[10,20,30]
lst2=[11,22,33]
lst1.extend(lst2)   #join list
print(lst1)
lst1.sort()   #sorting ascending manner
print(lst1)
lst1.pop()   #delete values from last one
print(lst1)
lst1.remove(20) #delete data according to values
print(lst1)
```

```
[10, 20, 30, 11, 22, 33]
[10, 11, 20, 22, 30, 33]
[10, 11, 20, 22, 30]
[10, 11, 22, 30]
```

Type Casting List into Set

In [116]:

```
Prod_Code=['P001','P002','P001','P003','P004']
print(Prod_Code)
print(type(Prod_Code))

Prod_Code=set(Prod_Code)
print("Set ",Prod_Code)  #type casting list into set

Prod_Code=list(Prod_Code) #type casting set into list
Prod_Code.sort()
print("List ", Prod_Code)
```

```
['P001', 'P002', 'P001', 'P003', 'P004']
<class 'list'>
Set  {'P003', 'P004', 'P002', 'P001'}
List  ['P001', 'P002', 'P003', 'P004']
```

Condition using with IF, ELSE, ELIF etc

In [123]:

```
marks=int(input("Enter your Marks "))
if marks>=40:
    print("Pass with - ",marks," Marks")
else:
    print("Fail with - ",marks," Marks")
```

```
Enter your Marks 60
Pass with -  60  Marks
```

Numpy

In [16]:

```
myarry=[[1,2,3],[4,5,6]]
print(myarry)
print(myarry[0][0])
print(myarry[0][1])
```

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

In [24]:

```
import numpy as np
array1=np.array([[1,2],[3,4]])
print(array1)
```

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

Pandas

In [26]:

```
import pandas as pd
lst=['Henry','Harvin','Python','Development','course']
df=pd.DataFrame(lst)
print(df)
```

```
             0
0        Henry
1       Harvin
2       Python
3  Development
4       course
```

read_json use for read external json file

In [27]:

```
import pandas as pd
df=pd.read_json('employeedata.json')
print(df.to_string())
```

```
   empid                   title  salary
0    101     IT Company Employee   41000
1    102  Sales Company Employee   20000
```

read_csv file

In [28]:

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

```
    Duration  Pulse  Maxpulse  Calories
0        411    NaN        95       NaN
1         60  110.0       130     409.1
2         60  117.0       145     479.0
3         60  103.0       135     340.0
4         45  109.0       175     282.4
5         45  117.0       148     406.0
6         60  102.0       127     300.0
7         60  110.0       136     374.0
8         45  104.0       134     253.3
9         30  109.0       133     195.1
10        60   98.0       124     269.0
11        60  103.0       147     329.3
12        60  100.0       120     250.7
13        60  106.0       128     345.3
14        60  104.0       132     379.3
15        60   98.0       123     275.0
16        60   98.0       120     215.2
17        60  100.0       120     300.0
```

read_excel file

In [32]:

```
df=pd.read_excel('SampleExcelFile.xlsx')
print(df.to_string())
```

```
   First Name Last Name       Dept   Branch  Years  Total Salary
0     Parvati    Khanna       Mktg  Mathura   1986         10500
1       Hajra   Hoonjan      Admin   Jaipur   1996          9625
2       Tapan   Ghoshal        CCD   Ambala   1997          4000
3      Zarina      Vora        CCD  Lucknow   1991          8750
4        Maya   Panchal       Mktg     Agra   1990         17500
5      Waheda    Sheikh        R&D    Jammu   1988         17500
6       Parul      Shah  Personnel     Agra   1988         17500
7     Laveena    Shenoy        CCD   Jaipur   1988         17500
8     Drishti      Shah        R&D    Delhi   1988         17500
9       Kunal      Shah        CCD  Aligarh   1999          3500
10       Ruby    Joseph        R&D     Agra   1998          7000
11     Chetan     Dalvi        CCD    Delhi   1989         17325
```

Numpy Array

In [37]:

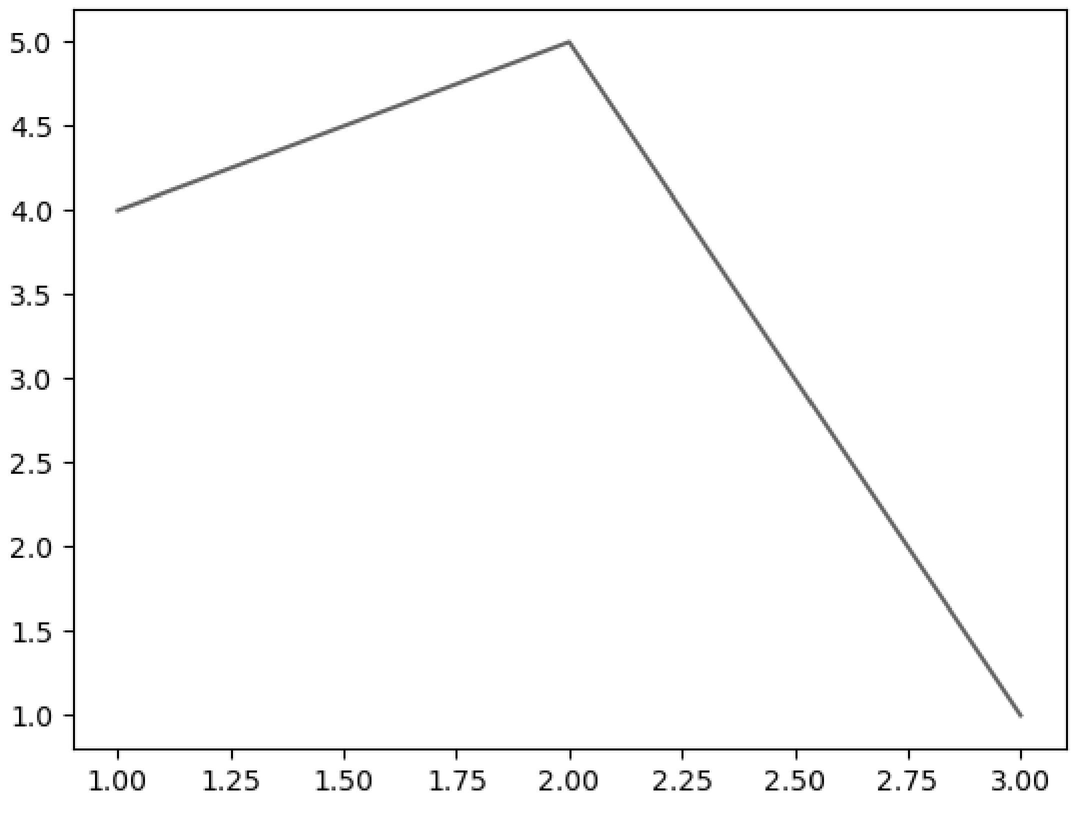
```
import numpy as np
#Single Dimentional Array
arr=np.array([1,2,3])
print(arr)
print("-----------------------")
#Two Dim. Array
arr2=np.array([[1,2,3],[4,5,6]])
print(arr2)
print("----------------------")
arr3=np.array((1,3,2))
print(arr3)
```

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

Matplotlib use for Data Visualization

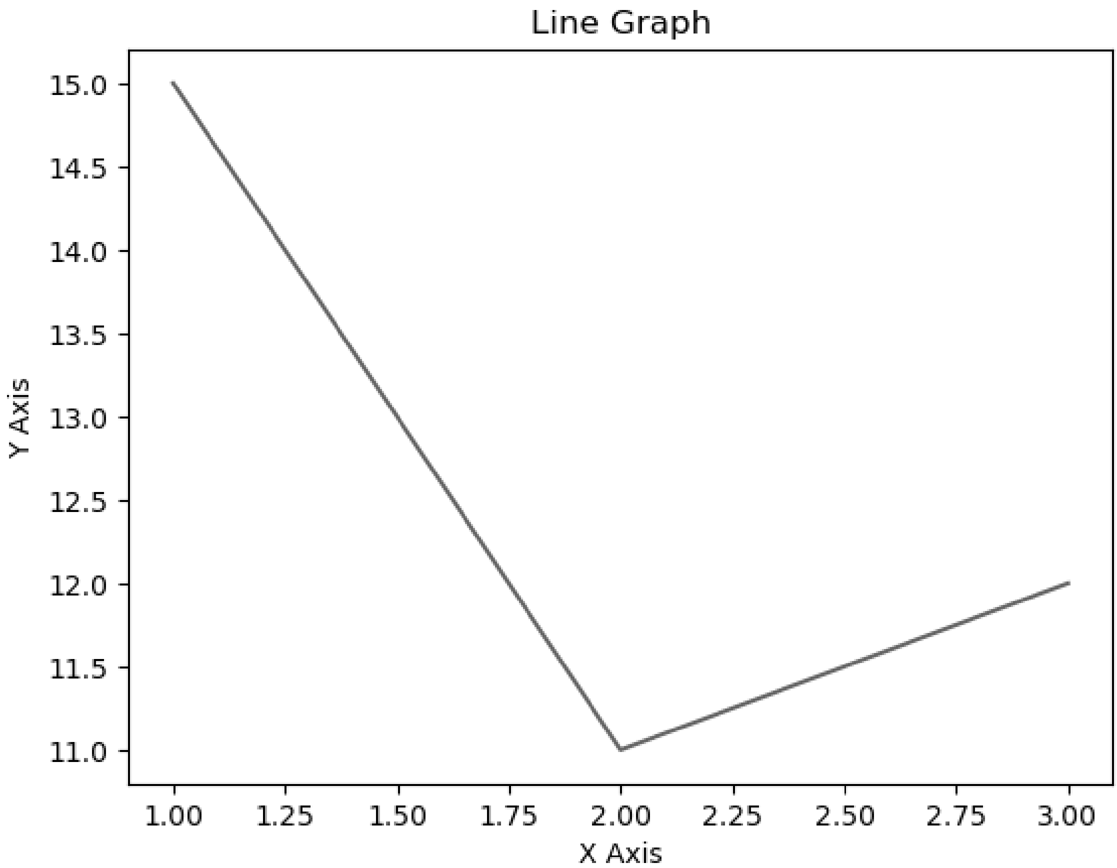
In [2]:

```
import matplotlib
import matplotlib.pyplot as plt
plt.plot([1,2,3],[4,5,1])
plt.show()
```



In [4]:

```
from matplotlib import pyplot as plt2
x=[1,2,3]
y=[15,11,12]
plt2.plot(x,y)
plt.title("Line Graph")
plt.ylabel("Y Axis")
plt.xlabel("X Axis")
plt2.show()
```



Scatter Chart

In [11]:

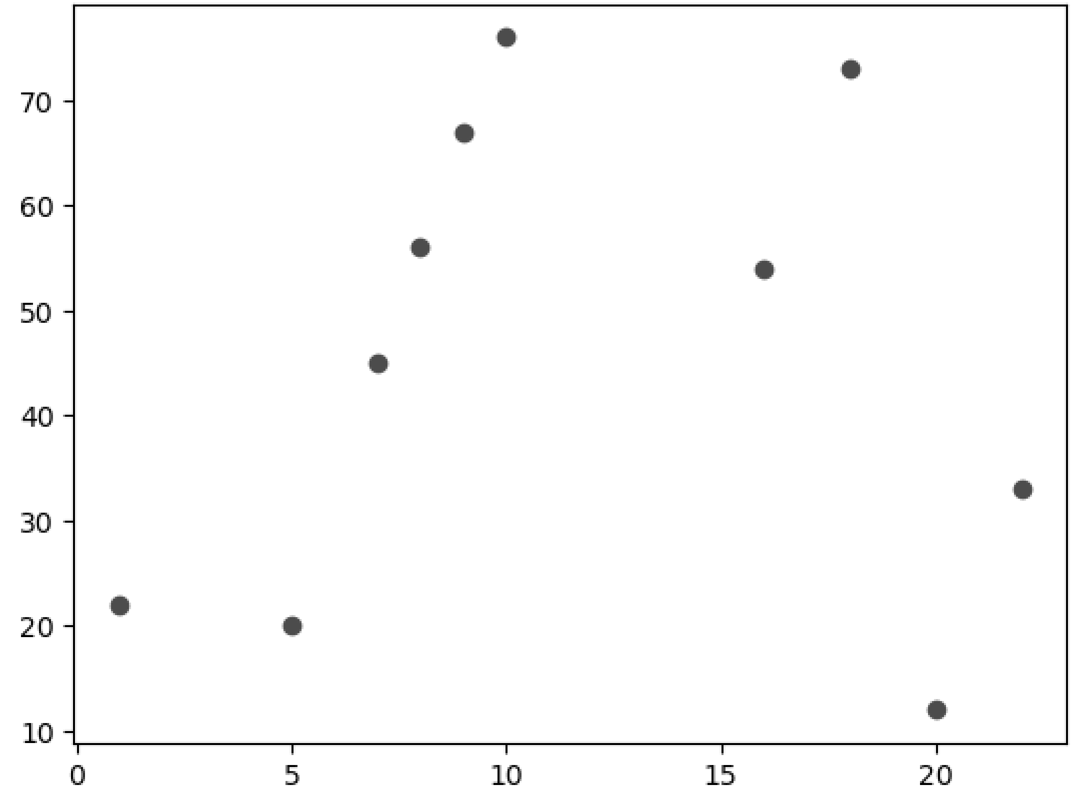
```
import matplotlib
import matplotlib.pyplot as plt
x1=[1,5,7,8,9,10,16,18,20,22]
y1=[22,20,45,56,67,76,54,73,12,33]
```

In [12]:

```
plt.scatter(x1,y1,c="red")
```

Out[12]:

<matplotlib.collections.PathCollection at 0x28edd3e18b0>



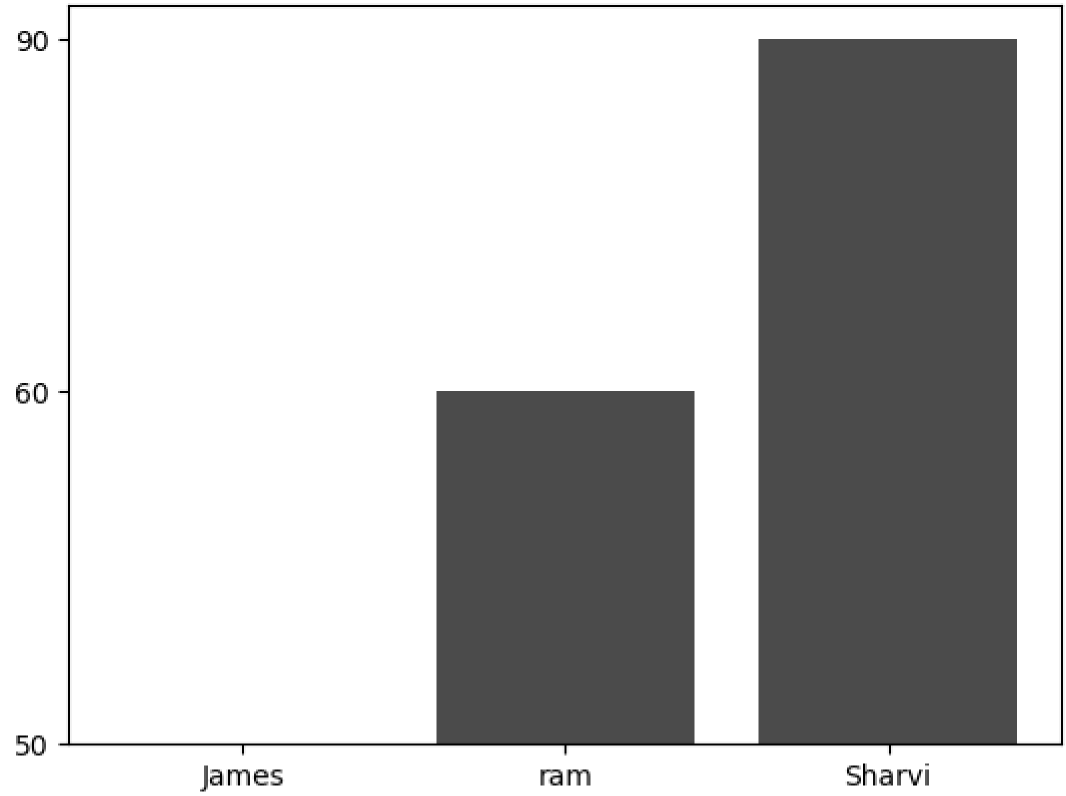
Bar Chart

In [22]:

```
Name=["James","ram","Sharvi"]
Marks=["50","60","90"]
plt.bar(Name,Marks,color="green")
```

Out[22]:

```
<BarContainer object of 3 artists>
```

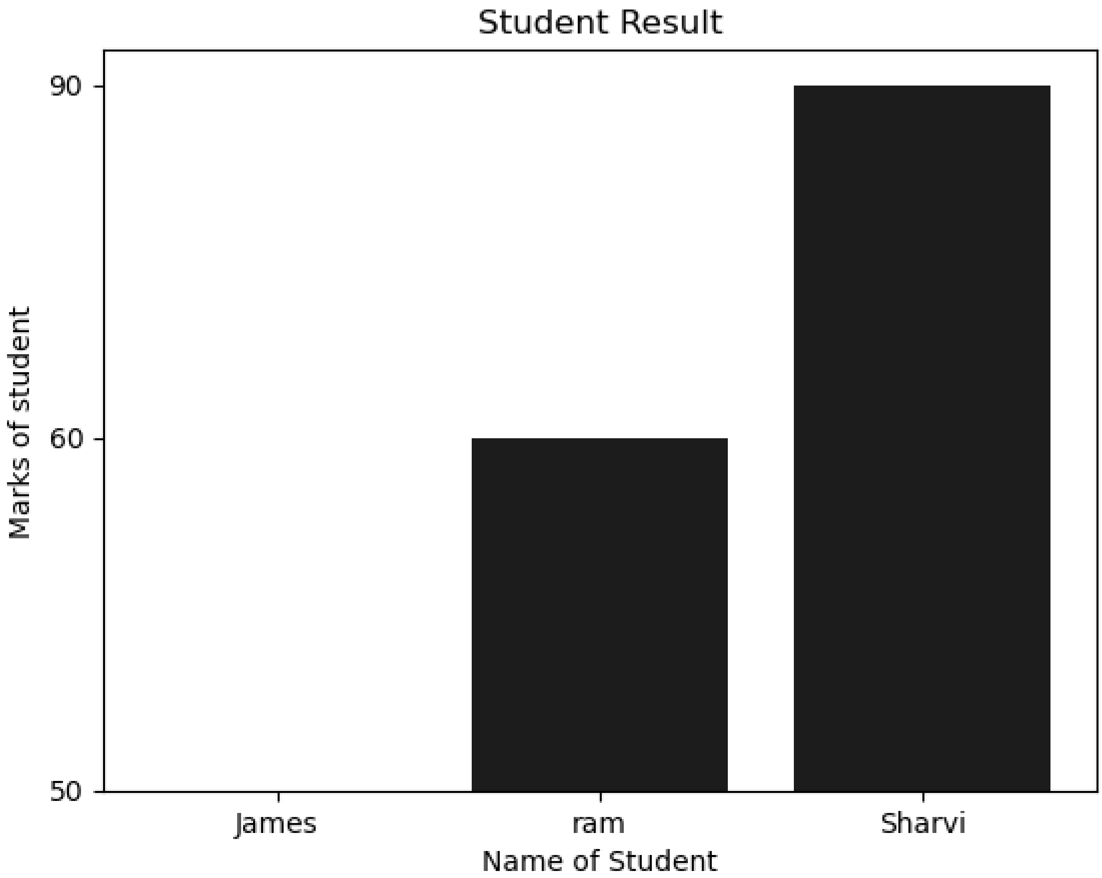


In [23]:

```
plt.bar(Name,Marks,color="blue")
plt.title("Student Result")
plt.xlabel('Name of Student')
plt.ylabel('Marks of student')
```

Out[23]:

```
Text(0, 0.5, 'Marks of student')
```



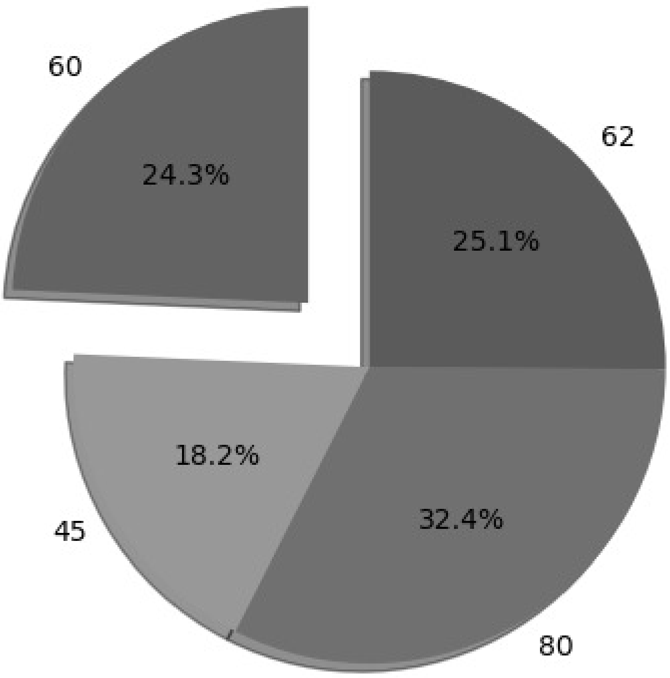
Pie Chart

In [30]:

```
from matplotlib import pyplot as plt
Student_Name=["James","Ravi","Gita","Javed"]
Score=["60","45","80","62"]


fig1,ax1=plt.subplots()
ax1.pie(Score,explode=explode,labels=Score,autopct='%1.1f%%',shadow=True,startangle=90)
```

Out[30]:

```
([<matplotlib.patches.Wedge at 0x28ede907850>,
  <matplotlib.patches.Wedge at 0x28ede916220>,
  <matplotlib.patches.Wedge at 0x28ede916be0>,
  <matplotlib.patches.Wedge at 0x28ede9235b0>],
 [Text(-0.9676715410398422, 1.0117370155636183, '60'),
  Text(-0.9502875973740678, -0.5540338277370993, '45'),
  Text(0.5720619285911781, -0.9395451824454969, '80'),
  Text(0.7802868621087867, 0.7753401916710003, '62')],
 [Text(-0.6220745620970414, 0.6504023671480403, '24.3%'),
  Text(-0.5183386894767642, -0.30220026967478136, '18.2%'),
  Text(0.3120337792315516, -0.5124791904248164, '32.4%'),
  Text(0.4256110156957018, 0.42291283182054556, '25.1%')])
```



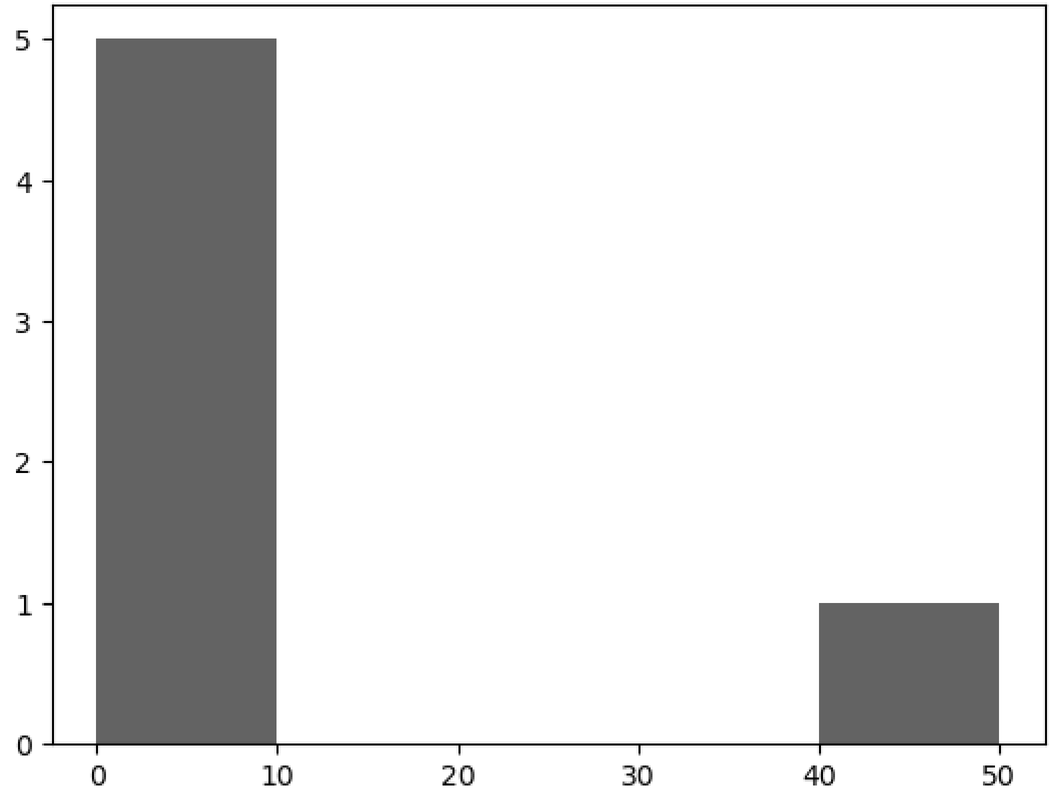
Histogram Chart

In [41]:

```
from matplotlib import pyplot as plt
Marks=[50,60,70,80,90,55,60,70,60,70,80,5,6,7,8,9]
student=[0,10,20,30,40,50]
plt.hist(Marks,student,histtype='bar',linewidth=0.8)
```

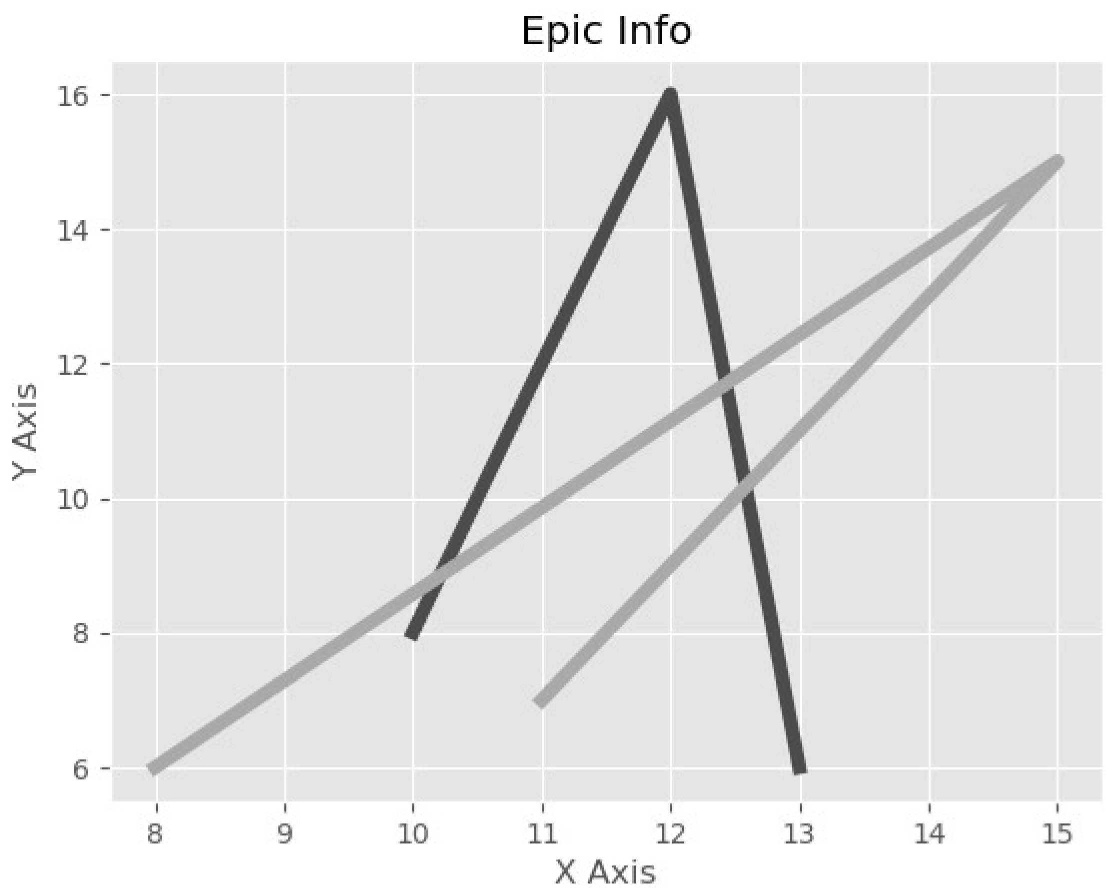
Out[41]:

```
(array([5., 0., 0., 0., 1.]),
 array([ 0, 10, 20, 30, 40, 50]),
 <BarContainer object of 5 artists>)
```



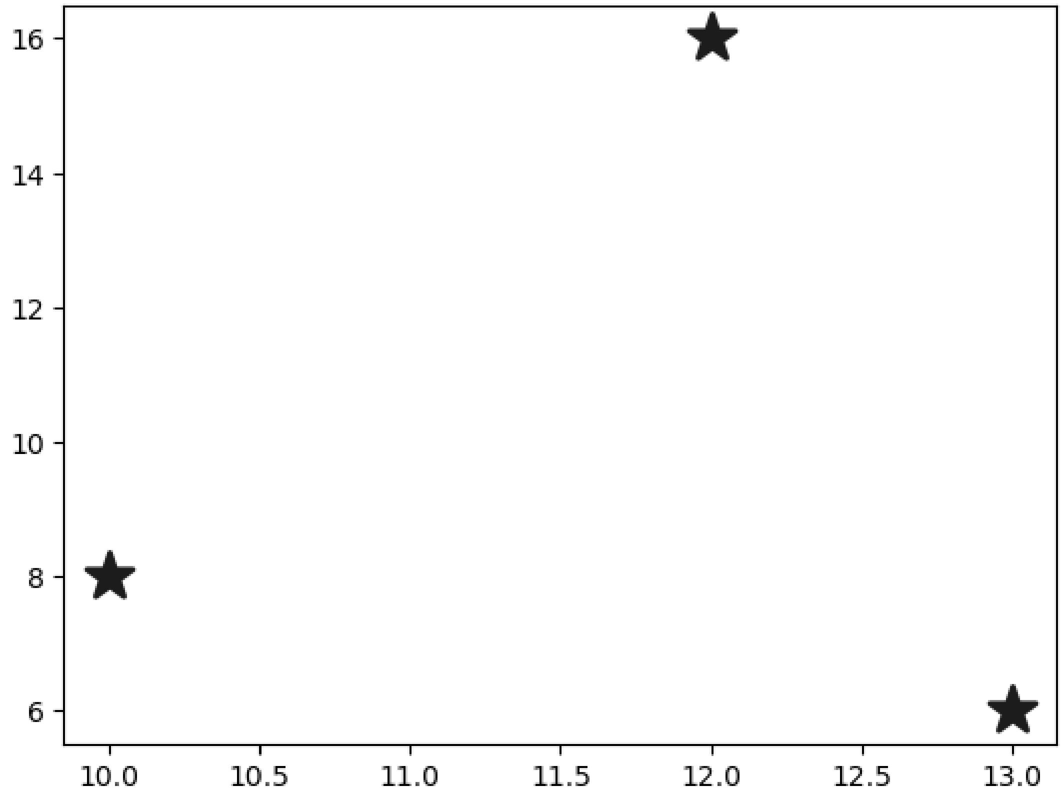
In [87]:

```
from matplotlib import pyplot as plt2
from matplotlib import style

style.use('ggplot')
x=[10,12,13]
y=[8,16,6]
x2=[8,15,11]
y2=[6,15,7]
plt.plot(x,y,'r',label="Line One",linewidth=5)
plt.plot(x2,y2,'y',label="Line Two",linewidth=5)
plt.title("Epic Info")
# fig=plt.figure() use for create new plot
plt.ylabel('Y Axis')
plt.xlabel('X Axis')
plt.show()
```



In [1]:

```
import matplotlib
import matplotlib.pyplot as plt
x=[10,12,13]
y=[8,16,6]
plt.scatter(x,y,marker='*',c="blue",s=300,linewidths=2)
plt.show()
```



3D Chart

In [3]:

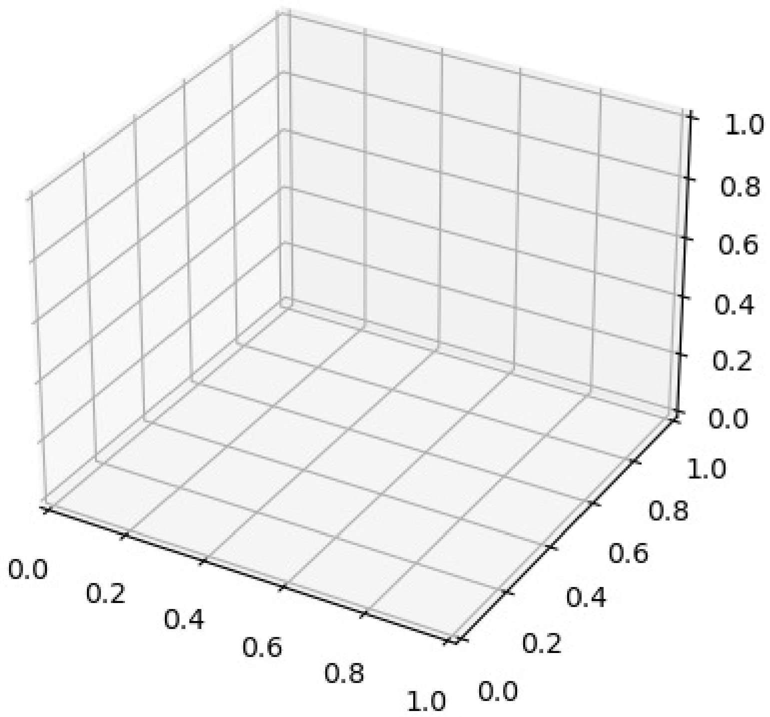
```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
```

In [4]:

```
x=[1,5,7,9,1]
y=[13,22,14,19,21]
plt.show()
```

In [5]:

```
fig=plt.figure()
ax=plt.axes(projection='3d')
```



In [9]:

```
#List
arr1=[10,20,30,40,"ram",80.25]
print(arr1)
print(type(arr1))
```

```
[10, 20, 30, 40, 'ram', 80.25]
<class 'list'>
```

In [10]:

```
import numpy as np
arr2=np.array([10,30,40,50,60,'hello',90.25])
print(arr2)
print(type(arr2))
```

```
['10' '30' '40' '50' '60' 'hello' '90.25']
<class 'numpy.ndarray'>
```

In [83]:

```
import pandas as pd
dataForIndex=pd.DataFrame({'city':['Delhi','Delhi','Chandigarh','Chandigarh','Chandigarh
                          'institute':['AIIT','ABS','AIIT','ABS','ALS']})
print(dataForIndex)
print(type(dataForIndex))
print(dataForIndex.describe())
print("Shape--",dataForIndex.shape)
print("Size --",dataForIndex.size)
print("Axes --",dataForIndex.axes)
print("Dtype --",dataForIndex.dtypes)
print("values  --",dataForIndex.values)
print("Index--- ",dataForIndex.index)
print("Sorting--",dataForIndex.sort_index()) #short by row name
print("Sorting Descending--",dataForIndex.sort_index(ascending=False))

midxsample = pd.MultiIndex(levels=[['zero', 'one'], ['x', 'y']],codes=[[1, 1, 0, 0], [1,
print(midxsample)
```

```
         city institute
0       Delhi      AIIT
1       Delhi       ABS
2  Chandigarh      AIIT
3  Chandigarh       ABS
4  Chandigarh       ALS
<class 'pandas.core.frame.DataFrame'>
              city institute
count            5         5
unique           2         3
top     Chandigarh      AIIT
freq             3         2
Shape-- (5, 2)
Size -- 10
Axes -- [RangeIndex(start=0, stop=5, step=1), Index(['city', 'institute'],
dtype='object')]
Dtype -- city         object
institute    object
dtype: object
values  -- [['Delhi' 'AIIT']
 ['Delhi' 'ABS']
 ['Chandigarh' 'AIIT']
 ['Chandigarh' 'ABS']
 ['Chandigarh' 'ALS']]
Index---  RangeIndex(start=0, stop=5, step=1)
Sorting--          city institute
0       Delhi      AIIT
1       Delhi       ABS
2  Chandigarh      AIIT
3  Chandigarh       ABS
4  Chandigarh       ALS
Sorting Descending--          city institute
4  Chandigarh       ALS
3  Chandigarh       ABS
2  Chandigarh      AIIT
1       Delhi       ABS
0       Delhi      AIIT
MultiIndex([( 'one', 'y'),
            ( 'one', 'x'),
            ('zero', 'y'),
            ('zero', 'x')],
           )
```

In [84]:

```
#List
Std_Record=[101,"James","Delhi",250001]
print(Std_Record)
Std_Record.pop()
print(Std_Record)
Std_Record.remove("James")
print(Std_Record)
```

```
[101, 'James', 'Delhi', 250001]
[101, 'James', 'Delhi']
[101, 'Delhi']
```

Panda use for fetch external file into python

In [85]:

```
import pandas as pd
pf=pd.read_csv("Financial-survey-2021.csv")
print(pf)
```

```
       Year Industry_aggregation_NZSIOC Industry_code_NZSIOC  \
0      2021                     Level 1                99999   
1      2021                     Level 1                99999   
2      2021                     Level 1                99999   
3      2021                     Level 1                99999   
4      2021                     Level 1                99999   
...     ...                         ...                  ...   
41710  2013                     Level 3                 ZZ11   
41711  2013                     Level 3                 ZZ11   
41712  2013                     Level 3                 ZZ11   
41713  2013                     Level 3                 ZZ11   
41714  2013                     Level 3                 ZZ11   

            Industry_name_NZSIOC               Units Variable_code  \
0                 All industries  Dollars (millions)           H01   
1                 All industries  Dollars (millions)           H04   
2                 All industries  Dollars (millions)           H05   
3                 All industries  Dollars (millions)           H07   
4                 All industries  Dollars (millions)           H08   
...                          ...                 ...           ...   
41710  Food product manufacturing          Percentage           H37   
41711  Food product manufacturing          Percentage           H38   
41712  Food product manufacturing          Percentage           H39   
41713  Food product manufacturing          Percentage           H40   
41714  Food product manufacturing          Percentage           H41   

                                          Variable_name      Variable_categ
ory  \
0                                          Total income  Financial performa
nce   
1      Sales, government funding, grants and subsidies  Financial performa
nce   
2                    Interest, dividends and donations  Financial performa
nce   
3                                 Non-operating income  Financial performa
nce   
4                                    Total expenditure  Financial performa
nce   
...                                                 ...                    
...   
41710                                      Quick ratio       Financial rat
ios   
41711              Margin on sales of goods for resale       Financial rat
ios   
41712                                 Return on equity       Financial rat
ios   
41713                           Return on total assets       Financial rat
ios   
41714                            Liabilities structure       Financial rat
ios   

         Value                             Industry_code_ANZSIC06  
0      757,504  ANZSIC06 divisions A-S (excluding classes K633...  
1      674,890  ANZSIC06 divisions A-S (excluding classes K633...  
2       49,593  ANZSIC06 divisions A-S (excluding classes K633...  
3       33,020  ANZSIC06 divisions A-S (excluding classes K633...  
4      654,404  ANZSIC06 divisions A-S (excluding classes K633...  
...        ...                                                ...  
41710       52  ANZSIC06 groups C111, C112, C113, C114, C115, ...  
41711       40  ANZSIC06 groups C111, C112, C113, C114, C115, ...  
41712       12  ANZSIC06 groups C111, C112, C113, C114, C115, ...  
```

```
41713        5  ANZSIC06 groups C111, C112, C113, C114, C115, ...
41714       46  ANZSIC06 groups C111, C112, C113, C114, C115, ...

[41715 rows x 10 columns]
```

Statistics

In [55]:

```
import statistics as st
data=[10,20,30,40,50,60,10]
print("Mean ",st.mean(data))  #Average
print("Median ",st.median(data)) # show middle value after arrange the data in ascending
print("Mode ",st.mode(data)) #mode show the most occurence value of data
print("Sample Variance ",st.variance(data)) #show the variance of average or data distri
print("Sample Standard Deviation ",st.stdev(data)) #show how the data far away from mean

print("Population Variance ",st.pvariance(data))
print("Population Standard Deviation ",st.pstdev(data))
```

```
Mean  31.428571428571427
Median  30
Mode  10
Sample Variance  380.95238095238096
Sample Standard Deviation  19.518001458970662
Population Variance  326.53061224489795
Population Standard Deviation  18.070158058105026
```

Statistics with numpy library

In [61]:

```
import numpy as np
data=[10,20,30,40,50,60,10]
print("Mean ",np.mean(data))
print("Mode ",np.median(data))
print("Varince ",np.var(data))

print("Population Varince ",np.var(data))
print("Population Standard Deviation ",np.std(data))
```

```
Mean  31.428571428571427
Mode  30.0
Varince  326.5306122448979
Population Varince  326.5306122448979
Population Standard Deviation  18.070158058105022
```

In [62]:

```
#Pandas Data Frame Line of code
```

In [66]:

```
data=[10,20,30,40,50,60]
print(data)
print(data[0:3])
```

```
[10, 20, 30, 40, 50, 60]
[10, 20, 30]
```

In [76]:

```
import pandas as pd
df=pd.read_csv("dataforstat.csv")
df
```

Out[76]:

	Gender	Height	Weight	bmi	Age
0	Male	174	80	26.4	25
1	Male	189	87	24.4	27
2	Female	185	80	23.4	30
3	Female	165	70	25.7	26
4	Male	149	61	27.5	28
5	Male	177	70	22.3	29
6	Female	149	65	30.1	31
7	Male	154	62	26.1	32
8	Male	174	90	29.7	27

In [78]:

```
df.iloc[1:3]  #print selected rows
```

Out[78]:

	Gender	Height	Weight	bmi	Age
1	Male	189	87	24.4	27
2	Female	185	80	23.4	30

In [82]:

```
df.iloc[1:3,0:2]
```

Out[82]:

	Gender	Height
1	Male	189
2	Female	185

Merge Different Data into dataframe

In [86]:

```
df1 = pd.DataFrame({'lkey': ['foo', 'bar', 'baz', 'foos'],
                    'value': [1, 2, 3, 5]})
df2 = pd.DataFrame({'rkey': ['foo', 'bar', 'baz', 'foo'],
                    'value': [5, 6, 7, 8]})
df1
df2
```

Out[86]:

	rkey	value
0	foo	5
1	bar	6
2	baz	7
3	foo	8

In [94]:

```
df1.merge(df2, left_on='lkey', right_on='rkey')
df1
```

Out[94]:

	lkey	value
0	foo	1
1	bar	2
2	baz	3
3	foos	5

In [98]:

```
df1 = pd.DataFrame({'name': ['Gita', 'Ram', 'James', 'Gita'],
                    'value': [5, 2, 3, 5]})
print(df1)
print(df1.duplicated()) #show duplicate
```

```
    name  value
0   Gita      5
1    Ram      2
2  James      3
3   Gita      5
0    False
1    False
2    False
3     True
dtype: bool
```

In [100]:

```
print(df1.drop_duplicates())
```

```
    name  value
0   Gita      5
1    Ram      2
2  James      3
```

Load Json API data

In [103]:

```
import requests
import json
response = requests.get('https://api.covid19api.com/summary').text
response_info = json.loads(response)
response_info
```

Out[103]:


```
{'ID': 'c20f699c-c622-40ca-9708-7162348248e2',
 'Message': '',
 'Global': {'NewConfirmed': 177325,
  'TotalConfirmed': 674300771,
  'NewDeaths': 1319,
  'TotalDeaths': 6793224,
  'NewRecovered': 0,
  'TotalRecovered': 0,
  'Date': '2023-04-30T06:49:41.244Z'},
 'Countries': [{'ID': '4e25613d-6218-44e5-80d5-ce1ebc8b26ad',
   'Country': 'Afghanistan',
   'CountryCode': 'AF',
   'Slug': 'afghanistan',
   'NewConfirmed': 0,
   'TotalConfirmed': 209451,
   'NewDeaths': 0,
   'TotalDeaths': 7896,
   'NewRecovered': 0,
```


Load API data from other library

In [105]:

```
first_response = requests.get("https://cat-fact.herokuapp.com/facts/random?animal_type=c
response_list=first_response.json()
# print(response_list)

data=[]
for response in response_list:
    data.append({
        "used": response.get('used'),
        "source": response.get('source'),
        "text": response.get('text'),
        "updatedAt": response.get('updatedAt'),
        "createdAt": response.get('createdAt'),
        "user": response.get('user')

    })
catfacts_df=pd.DataFrame(data)
catfacts_df.head()
```

Out[105]:

	used	source	text	updatedAt	createdAt	
0	None	None	Cats are really amazing pets.	2022-05-01T16:24:47.979Z	2022-05-01T16:24:47.979Z	626bd5af41f4aa428
1	None	None	Don't kniw.	2023-04-25T02:24:13.872Z	2023-04-25T02:24:13.872Z	644739a3d5e1289d3
2	None	None	98765432123456789.	2023-03-22T18:24:40.964Z	2023-03-22T18:24:40.964Z	6241761ba7107c0e1
3	None	None	Cats liove the dogs.	2023-03-09T20:32:30.955Z	2023-03-09T20:32:30.955Z	640615cdebdaaca99
4	None	None	Кошки ловят мышей и мотыльков.	2022-05-06T23:49:03.961Z	2022-05-06T23:49:03.961Z	626c5c3341f4aa42

Groupby function with panda

In [91]:

```
import pandas as pd
pd=pd.DataFrame({'City':['Meerut','Kanpur',"Delhi","Meerut"],"Sales":[5000,9000,8000,600
pd
```

Out[91]:

	City	Sales
0	Meerut	5000
1	Kanpur	9000
2	Delhi	8000
3	Meerut	6000

In [92]:

```
df=pd.groupby('City').sum()
df
```

Out[92]:

	Sales
City	
Delhi	8000
Kanpur	9000
Meerut	11000

Pivot

In [95]:

```
import pandas as pd
df = pd.DataFrame({'foo': ['one', 'one', 'one', 'two', 'two','two'],
                   'bar': ['A', 'B', 'C', 'A', 'B', 'C'],
                   'baz': [1, 2, 3, 4, 5, 6],
                   'zoo': ['x', 'y', 'z', 'q', 'w', 't']})
df
```

Out[95]:

	foo	bar	baz	zoo
0	one	A	1	x
1	one	B	2	y
2	one	C	3	z
3	two	A	4	q
4	two	B	5	w
5	two	C	6	t

In [129]:

```
df.pivot(index='foo', columns='bar', values='baz')
```

Out[129]:

bar	A	B	C
foo			
one	1	2	3
two	4	5	6

In [132]:

```
df.pivot(index='foo', columns='bar')['zoo']
```

Out[132]:

bar	A	B	C
foo			
one	x	y	z
two	q	w	t

Stack and Unstack Data

In [28]:

```
multicol1 = pd.MultiIndex.from_tuples([('weight', 'kg'),('weight', 'pounds')])
df_multi_level_cols1 = pd.DataFrame([[1, 2], [2, 4]],index=['cat', 'dog'], columns=multi
df_multi_level_cols1
```

Out[28]:

	weight	
	kg	pounds
cat	1	2
dog	2	4

In [29]:

```
df_multi_level_cols1.stack()
```

Out[29]:

		weight
cat	kg	1
	pounds	2
dog	kg	2
	pounds	4

In [30]:

```
df_multi_level_cols1.unstack()
```

Out[30]:

```
weight  kg      cat    1
                dog    2
        pounds  cat    2
                dog    4
dtype: int64
```

Library

In [8]:

```
import numpy as np
```

In [9]:

```
import pandas as pd
```

In [10]:

```
import matplotlib.pyplot as plt
```

In [11]:

```
url="https://raw.githubusercontent.com/callxpert/datasets/master/data-scientist-salaries
```

In [12]:

```
names=['Years-experience','Salary']
```

In [13]:

```
df=pd.read_csv(url,names=names)
```

```
---------------------------------------------------------------------------
----
TimeoutError                              Traceback (most recent call l
ast)
F:\Software\Data Science\AnacondInstallFile\lib\urllib\request.py in do
_open(self, http_class, req, **http_conn_args)
   1345             try:
-> 1346                 h.request(req.get_method(), req.selector, req.d
ata, headers,
   1347                           encode_chunked=req.has_header('Transf
er-encoding'))

F:\Software\Data Science\AnacondInstallFile\lib\http\client.py in reque
st(self, method, url, body, headers, encode_chunked)
   1284         """Send a complete request to the server."""
-> 1285         self._send_request(method, url, body, headers, encode_c
hunked)
   1286
```

In [100]:

```
print(df)
```

```
   Years-experience  Salary
0                 1  110000
1                 2  120000
2                 3  130000
3                 4  140000
4                 5  150000
5                 6  160000
6                 7  170000
7                 8  180000
8                 9  190000
9                10  200000
```

In [106]:

```
print(df['Salary'])
```

```
0    110000
1    120000
2    130000
3    140000
4    150000
5    160000
6    170000
7    180000
8    190000
9    200000
Name: Salary, dtype: int64
```

In [32]:

```
print(df.shape)
```

```
(10, 2)
```

In [33]:

```
df
```

Out[33]:

	Years-experience	Salary
0	1	110000
1	2	120000
2	3	130000
3	4	140000
4	5	150000
5	6	160000
6	7	170000
7	8	180000
8	9	190000
9	10	200000

In [34]:

```
df.head()
```

Out[34]:

	Years-experience	Salary
0	1	110000
1	2	120000
2	3	130000
3	4	140000
4	5	150000

In [35]:

```
df.tail()
```

Out[35]:

	Years-experience	Salary
5	6	160000
6	7	170000
7	8	180000
8	9	190000
9	10	200000

In [36]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 2 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Years-experience  10 non-null     int64
 1   Salary            10 non-null     int64
dtypes: int64(2)
memory usage: 288.0 bytes
```

In [40]:

```
df.describe()
```

Out[40]:

	Years-experience	Salary
count	10.00000	10.000000
mean	5.50000	155000.000000
std	3.02765	30276.503541
min	1.00000	110000.000000
25%	3.25000	132500.000000
50%	5.50000	155000.000000
75%	7.75000	177500.000000
max	10.00000	200000.000000

In [41]:

```
df.isnull()
```

Out[41]:

	Years-experience	Salary
0	False	False
1	False	False
2	False	False
3	False	False
4	False	False
5	False	False
6	False	False
7	False	False
8	False	False
9	False	False

In [42]:

```
df.isnull().sum()
```

Out[42]:

```
Years-experience    0
Salary              0
dtype: int64
```

In [43]:

```
df.isnull().sum().sum()
```

Out[43]:

```
0
```

Data Visualization

In [3]:

```
import matplotlib.pyplot as plt
```

In [4]:

```
df.plot()
```

```
---------------------------------------------------------------------------
-
NameError                                 Traceback (most recent call las
t)
~\AppData\Local\Temp\ipykernel_1460\3380326488.py in <module>
----> 1 df.plot()

NameError: name 'df' is not defined
```

In [54]:

```
df.columns
```

Out[54]:

```
Index(['Years-experience', 'Salary'], dtype='object')
```

In [2]:

```
df.skew()
```

```
---------------------------------------------------------------------------
-
NameError                                 Traceback (most recent call las
t)
~\AppData\Local\Temp\ipykernel_1460\1665899112.py in <module>
----> 1 df.skew()

NameError: name 'df' is not defined
```

Skewness

In [8]:

```
df.plot(kind='hist')
df.plot(kind='density')
df.skew()
```

```
---------------------------------------------------------------------------
-
NameError                                 Traceback (most recent call las
t)
~\AppData\Local\Temp\ipykernel_1460\1997218613.py in <module>
----> 1 df.plot(kind='hist')
      2 df.plot(kind='density')
      3 df.skew()

NameError: name 'df' is not defined
```

Kurtosis

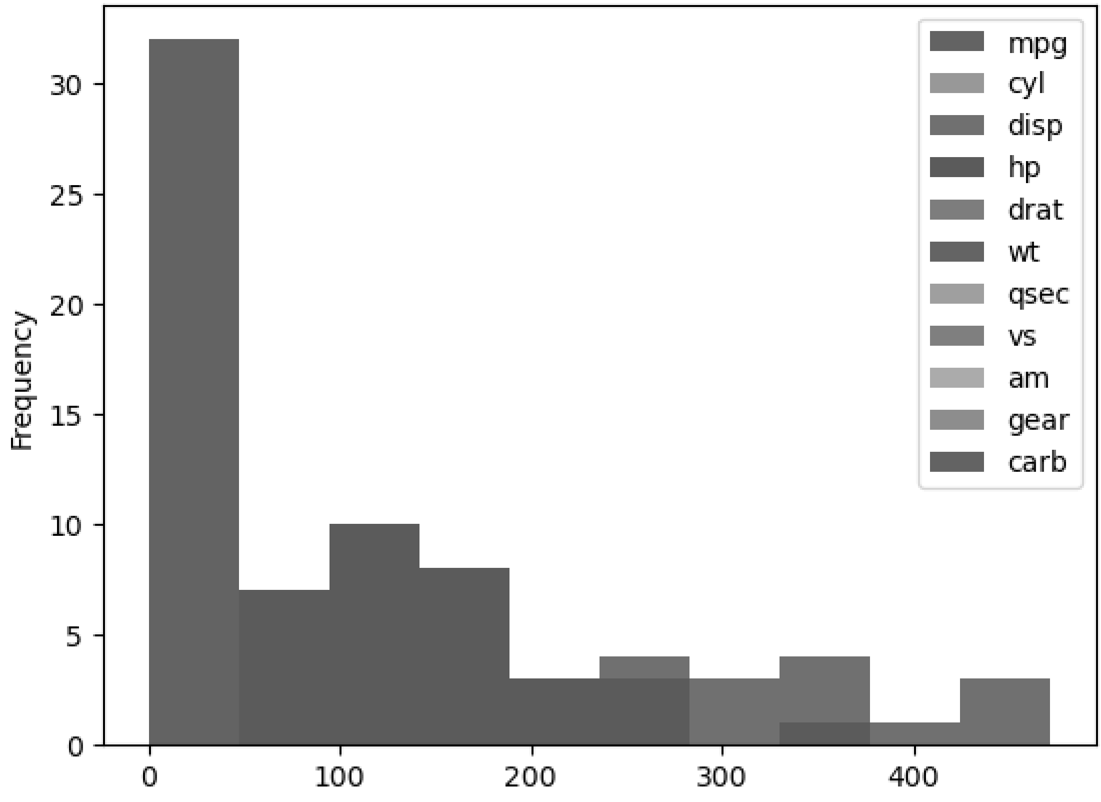
In [16]:

```
df.kurtosis()
df.plot(kind='hist')
```

```
C:\Users\Sachin sirohi\AppData\Local\Temp\ipykernel_23656\1981945559.py:1:
FutureWarning: Dropping of nuisance columns in DataFrame reductions (with
'numeric_only=None') is deprecated; in a future version this will raise Ty
peError.  Select only valid columns before calling the reduction.
  df.kurtosis()
```

Out[16]:

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



In [37]:

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

In [38]:

```
df=pd.read_csv("F:\Training\mtcars.csv")
df
```

Out[38]:

	Unnamed: 0	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
1	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
2	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
3	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
4	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
5	Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
6	Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
7	Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
8	Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
9	Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
10	Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
11	Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
12	Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
13	Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
14	Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4
15	Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4
16	Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4
17	Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
18	Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
19	Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1
20	Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1
21	Dodge Challenger	15.5	8	318.0	150	2.76	3.520	16.87	0	0	3	2
22	AMC Javelin	15.2	8	304.0	150	3.15	3.435	17.30	0	0	3	2
23	Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.41	0	0	3	4
24	Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.05	0	0	3	2
25	Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1
26	Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.70	0	1	5	2
27	Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.90	1	1	5	2
28	Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.50	0	1	5	4
29	Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.50	0	1	5	6
30	Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.60	0	1	5	8
31	Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.60	1	1	4	2

In [19]:

```
df.axes
```

Out[19]:

```
[RangeIndex(start=0, stop=32, step=1),
 Index(['Unnamed: 0', 'mpg', 'cyl', 'disp', 'hp', 'drat', 'wt', 'qsec', 'v
s',
        'am', 'gear', 'carb'],
       dtype='object')]
```

In [20]:

```
df.index
```

Out[20]:

```
RangeIndex(start=0, stop=32, step=1)
```

In [21]:

```
df.sort_index(ascending=True)
```

Out[21]:

	Unnamed: 0	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
1	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
2	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
3	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
4	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
5	Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
6	Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
7	Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
8	Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
9	Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
10	Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
11	Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
12	Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
13	Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
14	Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4
15	Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4
16	Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4
17	Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
18	Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
19	Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1
20	Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1
21	Dodge Challenger	15.5	8	318.0	150	2.76	3.520	16.87	0	0	3	2
22	AMC Javelin	15.2	8	304.0	150	3.15	3.435	17.30	0	0	3	2
23	Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.41	0	0	3	4
24	Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.05	0	0	3	2
25	Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1
26	Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.70	0	1	5	2
27	Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.90	1	1	5	2
28	Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.50	0	1	5	4
29	Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.50	0	1	5	6
30	Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.60	0	1	5	8
31	Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.60	1	1	4	2

In [22]:

```
df.iloc[1]
```

Out[22]:

```
Unnamed: 0    Mazda RX4 Wag
mpg                    21.0
cyl                       6
disp                  160.0
hp                      110
drat                    3.9
wt                    2.875
qsec                  17.02
vs                        0
am                        1
gear                      4
carb                      4
Name: 1, dtype: object
```

In [23]:

```
df.iloc[1:5]
```

Out[23]:

	Unnamed: 0	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
1	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
2	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
3	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
4	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2

In [24]:

```
df.iloc[1:5,2:7]
```

Out[24]:

	cyl	disp	hp	drat	wt
1	6	160.0	110	3.90	2.875
2	4	108.0	93	3.85	2.320
3	6	258.0	110	3.08	3.215
4	8	360.0	175	3.15	3.440

In [26]:

```
indx1=df.index
indx1
```

Out[26]:

```
RangeIndex(start=0, stop=32, step=1)
```

In [30]:

```
dfmulti=pd.DataFrame({'city':['Noida','Pune',"Delhi"],'institue':['ABC','DEF','PQR']})
dfmulti
```

Out[30]:

	city	institue
0	Noida	ABC
1	Pune	DEF
2	Delhi	PQR

In [41]:

```
mdx=pd.MultiIndex(levels=[['Noida','Pune','Delhi'],['ABC','DEF','PQR']],codes=[[0,1,0],[
mdx
```

Out[41]:

```
MultiIndex([('Noida', 'DEF'),
            ( 'Pune', 'ABC'),
            ('Noida', 'DEF')],
           )
```

In [42]:

```
df.mean()
```

```
C:\Users\Sachin sirohi\AppData\Local\Temp\ipykernel_23656\3698961737.py:1:
FutureWarning: Dropping of nuisance columns in DataFrame reductions (with
'numeric_only=None') is deprecated; in a future version this will raise Ty
peError.  Select only valid columns before calling the reduction.
  df.mean()
```

Out[42]:

```
mpg      20.090625
cyl       6.187500
disp    230.721875
hp      146.687500
drat      3.596563
wt        3.217250
qsec     17.848750
vs        0.437500
am        0.406250
gear      3.687500
carb      2.812500
dtype: float64
```

In [45]:

```
df.median()
```

```
C:\Users\Sachin sirohi\AppData\Local\Temp\ipykernel_23656\530051474.py:1:
FutureWarning: Dropping of nuisance columns in DataFrame reductions (with
'numeric_only=None') is deprecated; in a future version this will raise Ty
peError.  Select only valid columns before calling the reduction.
  df.median()
```

Out[45]:

```
mpg      19.200
cyl       6.000
disp    196.300
hp      123.000
drat      3.695
wt        3.325
qsec     17.710
vs        0.000
am        0.000
gear      4.000
carb      2.000
dtype: float64
```

In [46]:

```
df.hp.mean()
```

Out[46]:

```
146.6875
```

In [47]:

```
import statistics as st
st.mode(df.cyl)
```

Out[47]:

```
8
```

In [48]:

```
df.std()
```

```
C:\Users\Sachin sirohi\AppData\Local\Temp\ipykernel_23656\3390915376.py:1:
FutureWarning: Dropping of nuisance columns in DataFrame reductions (with
'numeric_only=None') is deprecated; in a future version this will raise Ty
peError.  Select only valid columns before calling the reduction.
  df.std()
```

Out[48]:

```
mpg       6.026948
cyl       1.785922
disp    123.938694
hp       68.562868
drat      0.534679
wt        0.978457
qsec      1.786943
vs        0.504016
am        0.498991
gear      0.737804
carb      1.615200
dtype: float64
```

In [49]:

```
st.stdev(df.mpg)
```

Out[49]:

```
6.026948052089104
```

In [50]:

```
st.variance(df.mpg)
```

Out[50]:

```
36.32410282258064
```

In [51]:

```
df.columns
```

Out[51]:

```
Index(['Unnamed: 0', 'mpg', 'cyl', 'disp', 'hp', 'drat', 'wt', 'qsec', 'v
s',
       'am', 'gear', 'carb'],
      dtype='object')
```

In [52]:

```
df.mpg.skew()
```

Out[52]:

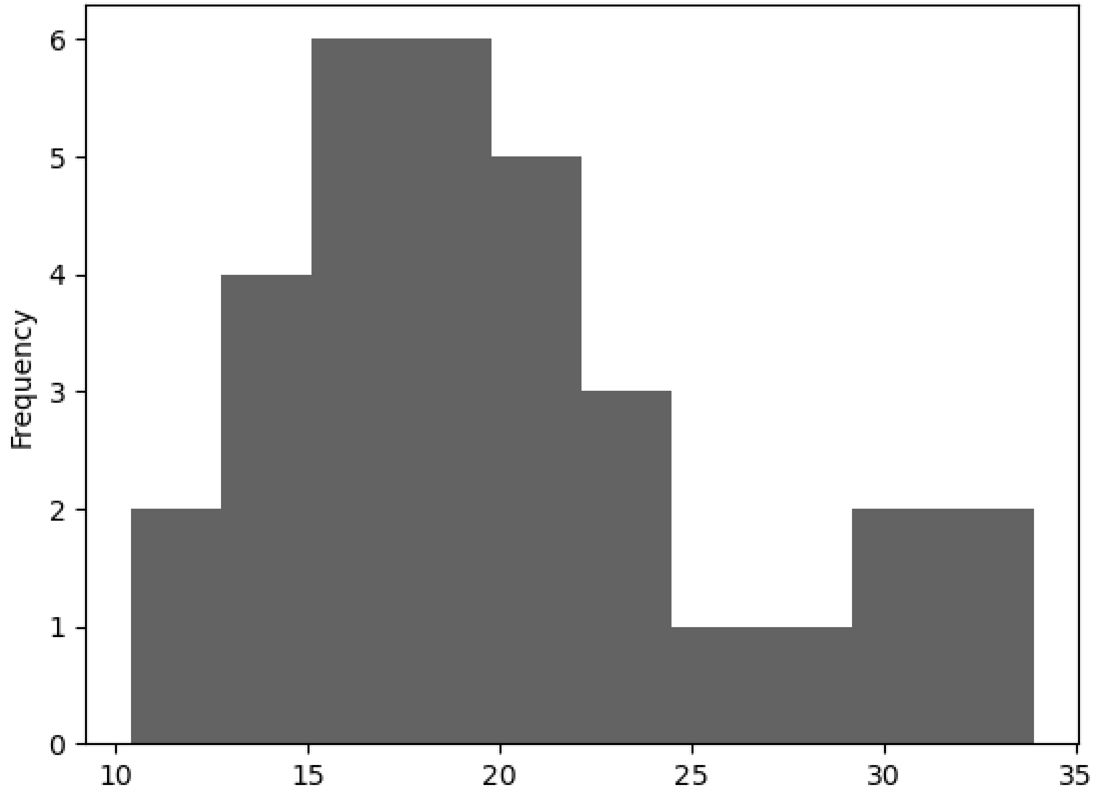
```
0.6723771376290805
```

In [53]:

```
df.mpg.plot(kind='hist')
```

Out[53]:

<AxesSubplot:ylabel='Frequency'>



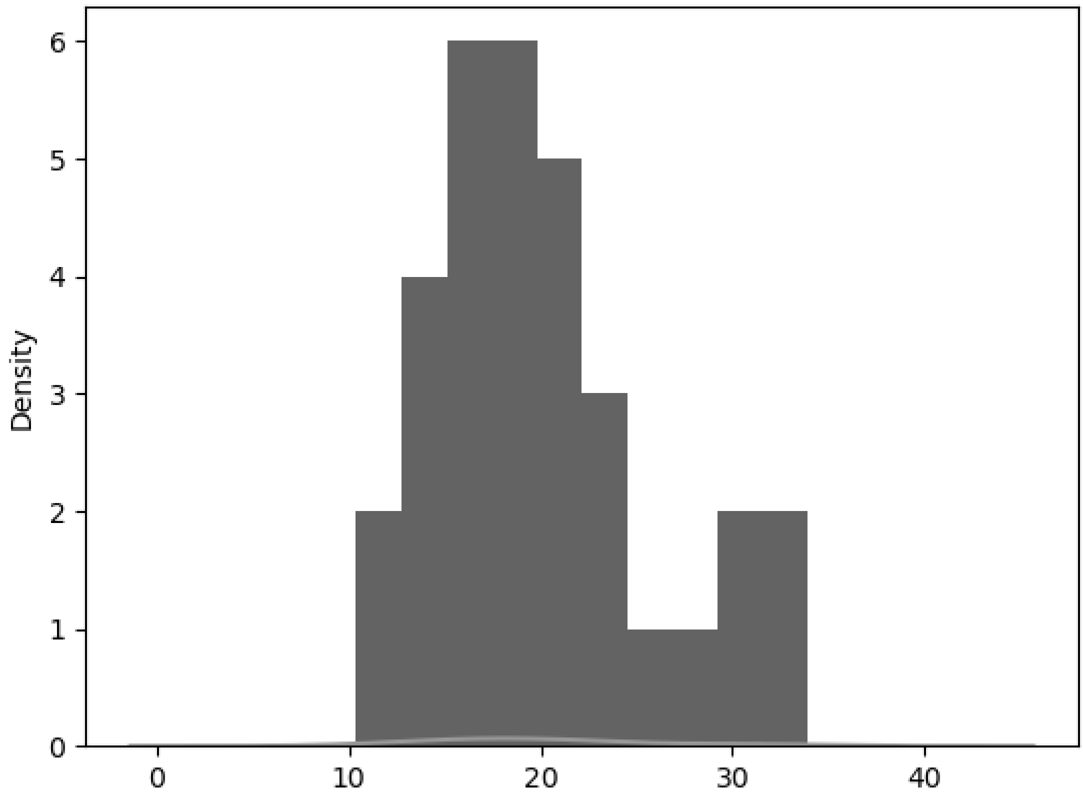
In [55]:

```
df.mpg.plot(kind='hist')
df.mpg.plot(kind='density')
df.skew()
```

```
C:\Users\Sachin sirohi\AppData\Local\Temp\ipykernel_23656\2307804638.py:3:
FutureWarning: Dropping of nuisance columns in DataFrame reductions (with
'numeric_only=None') is deprecated; in a future version this will raise Ty
peError.  Select only valid columns before calling the reduction.
  df.skew()
```

Out[55]:

```
mpg     0.672377
cyl    -0.192261
disp    0.420233
hp      0.799407
drat    0.292780
wt      0.465916
qsec    0.406347
vs      0.264542
am      0.400809
gear    0.582309
carb    1.157091
dtype: float64
```



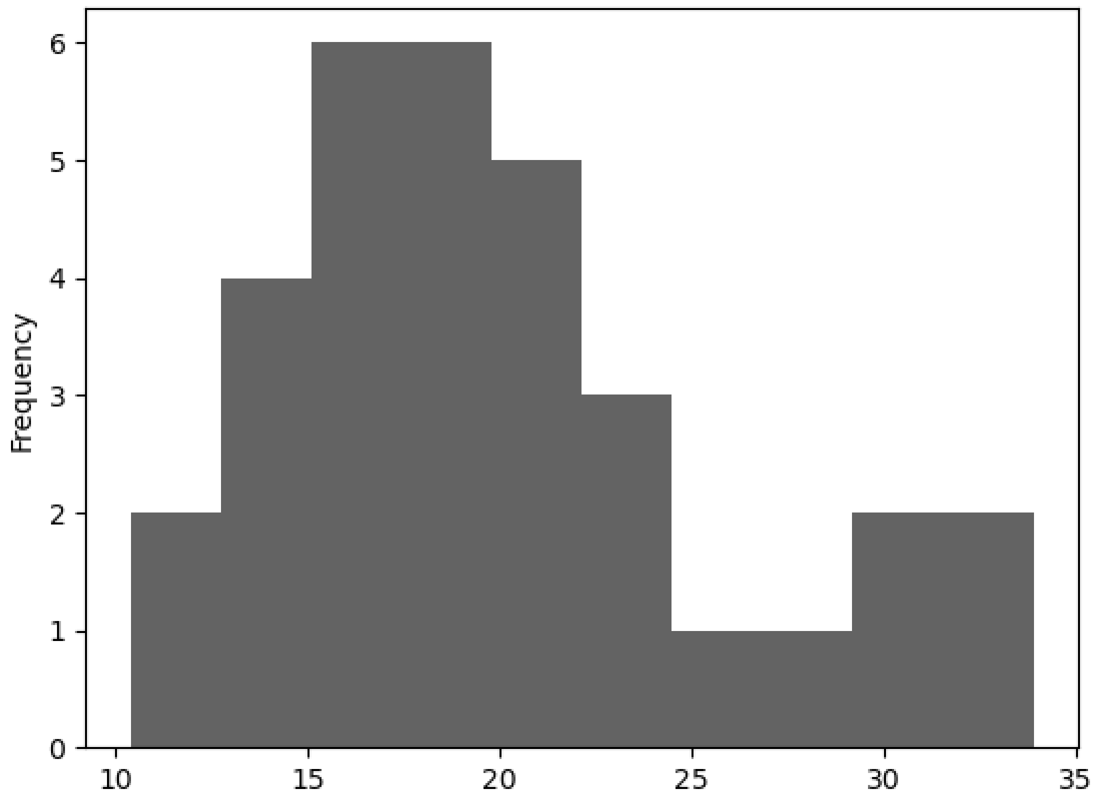
In [57]:

```
df.kurtosis()
df.mpg.kurtosis()
df.mpg.plot(kind='hist')
```

```
C:\Users\Sachin sirohi\AppData\Local\Temp\ipykernel_23656\1142384626.py:1:
FutureWarning: Dropping of nuisance columns in DataFrame reductions (with
'numeric_only=None') is deprecated; in a future version this will raise Ty
peError.  Select only valid columns before calling the reduction.
  df.kurtosis()
```

Out[57]:

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

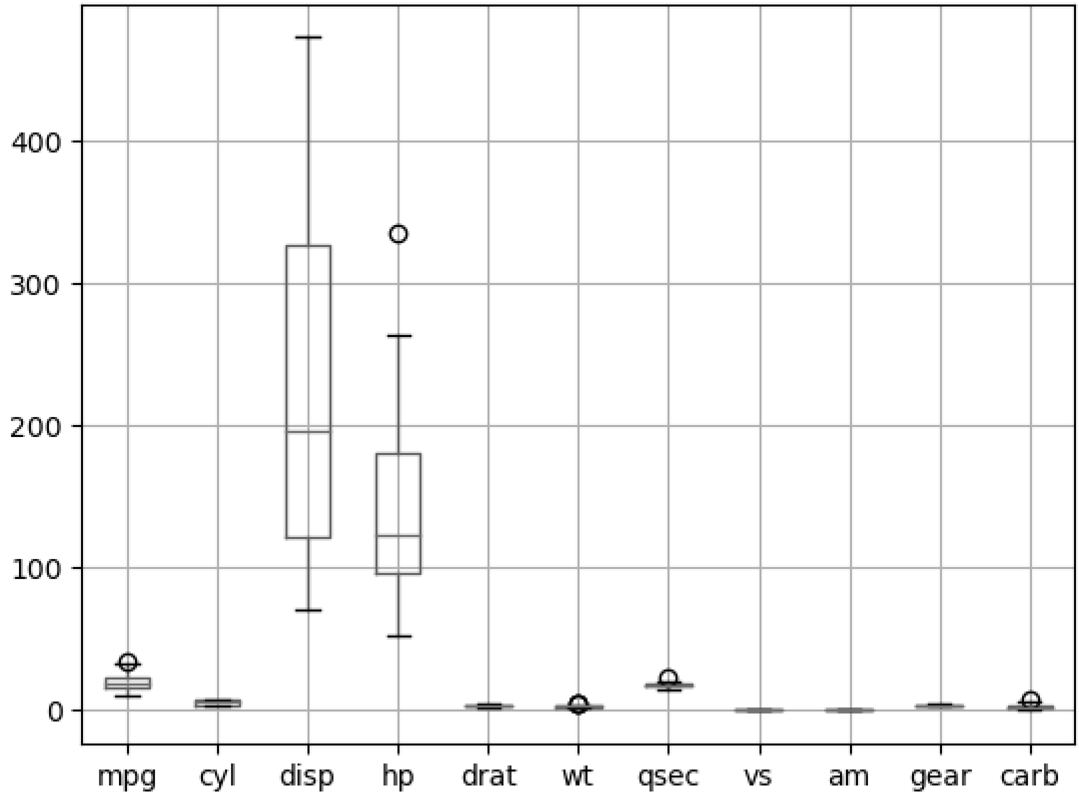


In [58]:

```
df.boxplot()
```

Out[58]:

<AxesSubplot:>

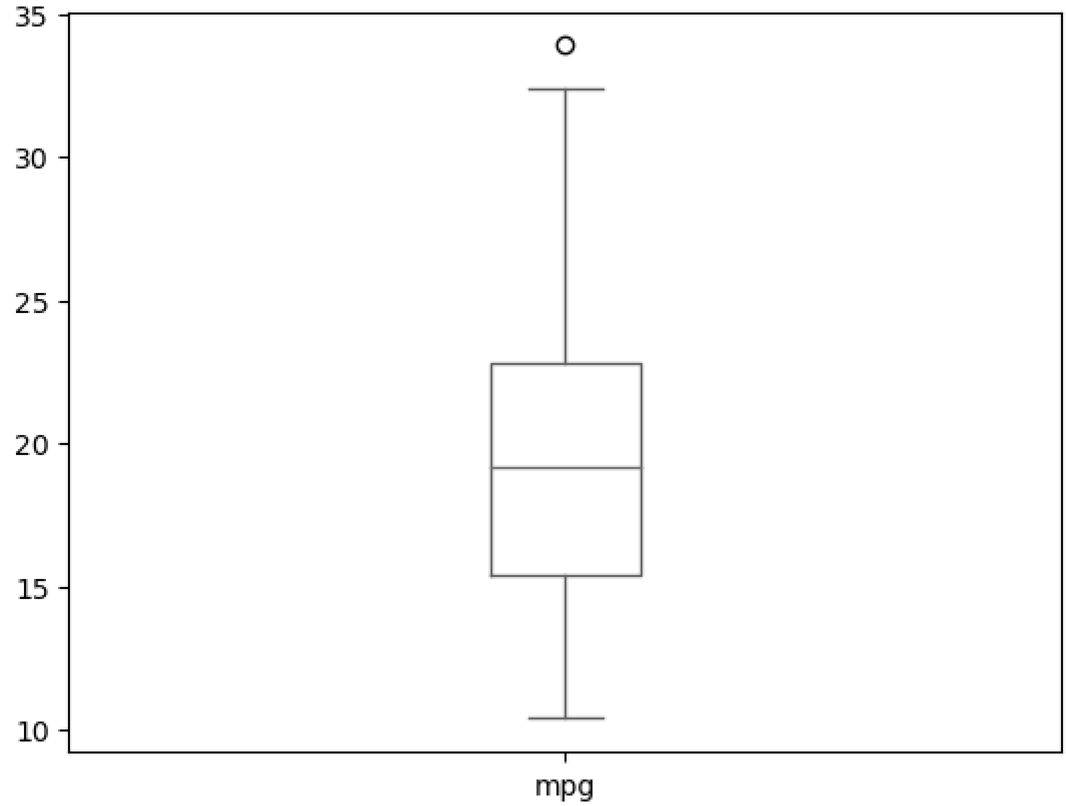


In [59]:

```
df.mpg.plot(kind='box')
```

Out[59]:

```
<AxesSubplot:>
```



In [60]:

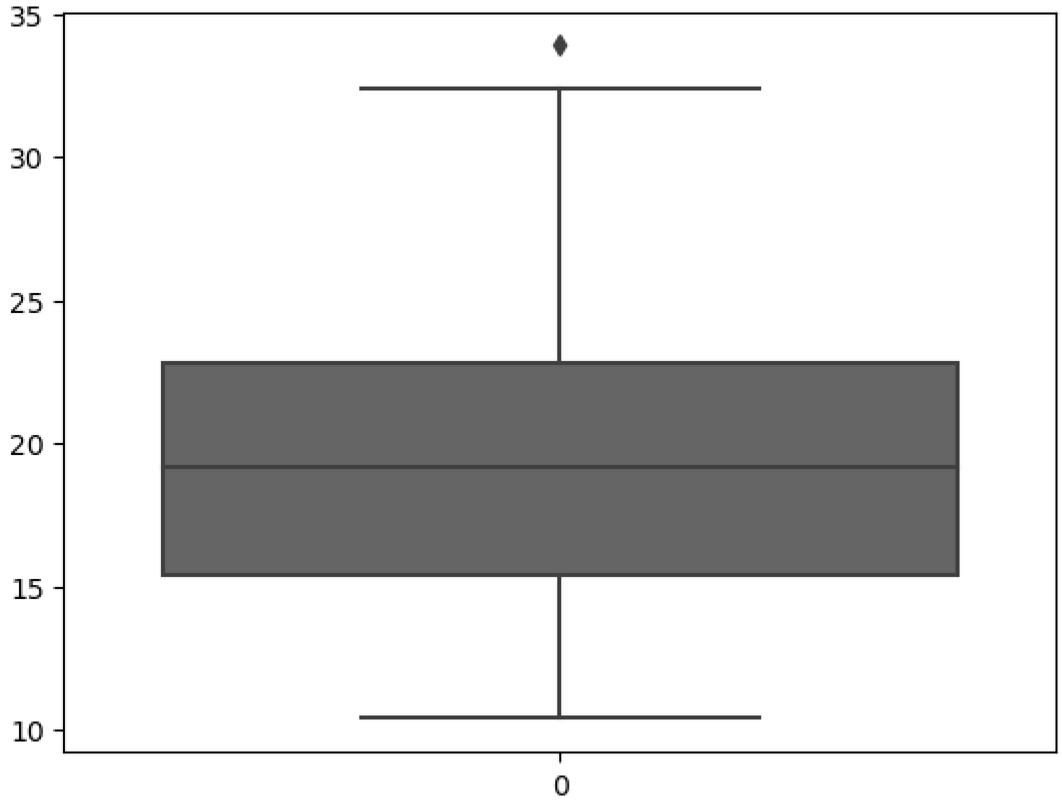
```
import seaborn as sns
```

In [61]:

```
sns.boxplot(df.mpg)
```

Out[61]:

<AxesSubplot:>



In [62]:

```
np.where(df['mpg']>30)
```

Out[62]:

```
(array([17, 18, 19, 27], dtype=int64),)
```

In [64]:

```
np.where(df['mpg']<15)
```

Out[64]:

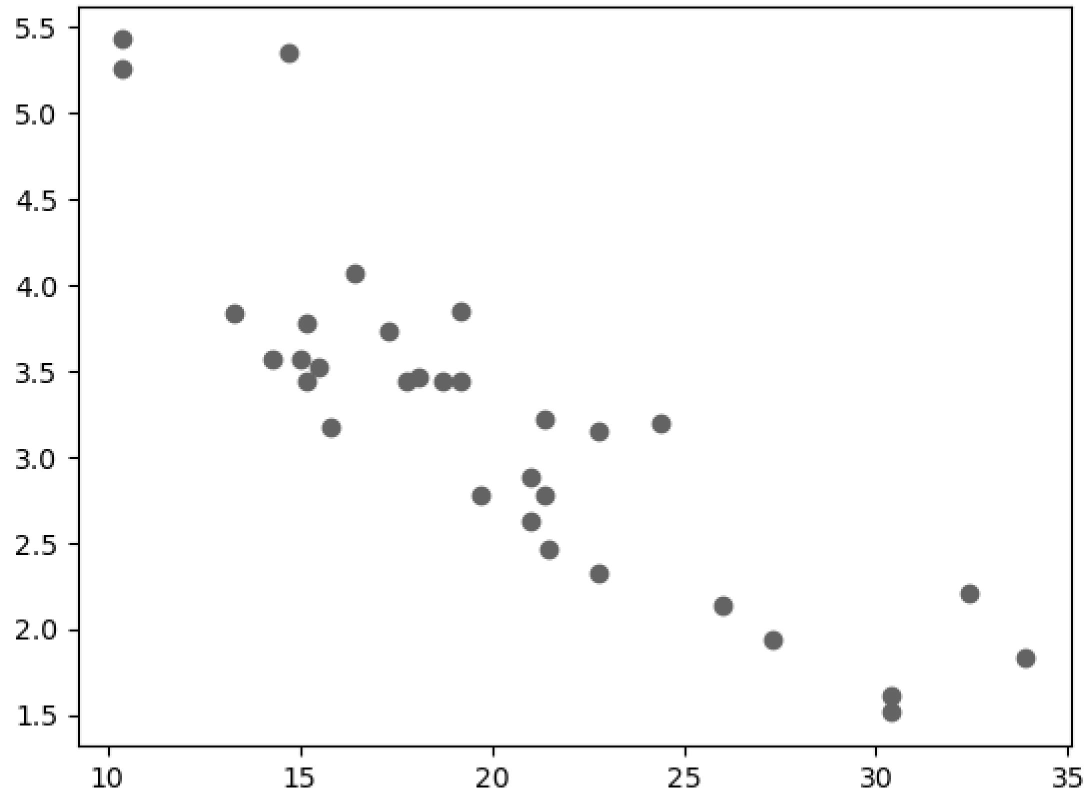
```
(array([ 6, 14, 15, 16, 23], dtype=int64),)
```

In [68]:

```
plt.scatter(df.mpg,df.wt)
np.where((df['mpg']<12,df['wt']>5))
df.iloc[[11,12,13],[2,7]]
```

Out[68]:

	cyl	qsec
11	8	17.4
12	8	17.6
13	8	18.0



In [69]:

```
Q1=np.percentile(df.mpg,25)
Q3=np.percentile(df.mpg,75)
IQR=Q3-Q1
IQR
```

Out[69]:

```
7.375
```

In [71]:

```
from scipy import stats
```

In [72]:

```
IQR2=stats.iqr(df.mpg)
IQR2
```

Out[72]:

7.375

In [73]:

```
upper_bound=Q3+(1.5*IQR)
lower_bound=Q1=(1.5*IQR)
print(upper_bound,lower_bound)
```

33.8625 11.0625

In [76]:

```
ary=df.mpg
ary
```

Out[76]:

```
0     21.0
1     21.0
2     22.8
3     21.4
4     18.7
5     18.1
6     14.3
7     24.4
8     22.8
9     19.2
10    17.8
11    16.4
12    17.3
13    15.2
14    10.4
15    10.4
16    14.7
17    32.4
18    30.4
19    33.9
20    21.5
21    15.5
22    15.2
23    13.3
24    19.2
25    27.3
26    26.0
27    30.4
28    15.8
29    19.7
30    15.0
31    21.4
Name: mpg, dtype: float64
```

In [78]:

```
outlier=[(ary<=lower_bound)|(ary>+upper_bound)]
outlier
```

Out[78]:

```
[0     False
 1     False
 2     False
 3     False
 4     False
 5     False
 6     False
 7     False
 8     False
 9     False
 10    False
 11    False
 12    False
 13    False
 14     True
 15     True
 16    False
 17    False
 18    False
 19     True
 20    False
 21    False
 22    False
 23    False
 24    False
 25    False
 26    False
 27    False
 28    False
 29    False
 30    False
 31    False
 Name: mpg, dtype: bool]
```

In [79]:

```
ary.values[outlier]
```

```
C:\Users\Sachin sirohi\AppData\Local\Temp\ipykernel_23656\1168532888.py:1:
FutureWarning: Using a non-tuple sequence for multidimensional indexing is
deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future thi
s will be interpreted as an array index, `arr[np.array(seq)]`, which will
result either in an error or a different result.
  ary.values[outlier]
```

Out[79]:

```
array([10.4, 10.4, 33.9])
```

In [80]:

```
df[df.mpg>=31]
```

Out[80]:

	Unnamed: 0	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
17	Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
19	Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1

New Topic Numpy, Pandas & Matplotlib

In [6]:

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

In [12]:

```
df=pd.read_csv("F:\Training\cars.csv")
df
```

Out[12]:

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	21.0	6.0	160.0	110.0	3.90	2.620	16.46	0.0	1.0	4.0	4.0
1	21.0	6.0	160.0	110.0	3.90	2.875	17.02	0.0	1.0	4.0	4.0
2	22.8	4.0	108.0	93.0	3.85	2.320	18.61	1.0	1.0	4.0	1.0
3	21.4	6.0	258.0	110.0	3.08	3.215	19.44	1.0	0.0	3.0	1.0
4	18.7	8.0	360.0	175.0	3.15	3.440	17.02	0.0	0.0	3.0	2.0
5	18.1	6.0	225.0	105.0	2.76	3.460	20.22	1.0	0.0	3.0	1.0
6	14.3	8.0	360.0	245.0	3.21	NaN	15.84	NaN	0.0	3.0	4.0
7	24.4	4.0	146.7	62.0	3.69	3.190	20.00	1.0	0.0	4.0	2.0
8	22.8	NaN	140.8	95.0	3.92	3.150	22.90	1.0	0.0	4.0	2.0
9	19.2	6.0	167.6	123.0	NaN	3.440	18.30	1.0	0.0	4.0	4.0
10	17.8	6.0	167.6	123.0	3.92	3.440	18.90	1.0	0.0	4.0	4.0
11	16.4	8.0	275.8	180.0	3.07	4.070	17.40	0.0	NaN	3.0	NaN
12	17.3	8.0	275.8	180.0	3.07	3.730	17.60	0.0	0.0	3.0	3.0
13	15.2	8.0	275.8	180.0	3.07	3.780	18.00	0.0	0.0	NaN	3.0
14	10.4	8.0	472.0	205.0	2.93	5.250	17.98	0.0	0.0	3.0	4.0
15	10.4	8.0	460.0	215.0	3.00	5.424	17.82	0.0	0.0	3.0	4.0
16	14.7	8.0	440.0	230.0	NaN	5.345	17.42	0.0	0.0	3.0	4.0
17	32.4	4.0	NaN	NaN	4.08	2.200	19.47	1.0	1.0	4.0	1.0
18	30.4	4.0	75.7	52.0	4.93	1.615	NaN	1.0	1.0	4.0	2.0
19	33.9	NaN	71.1	65.0	4.22	1.835	19.90	1.0	1.0	4.0	1.0
20	NaN	4.0	120.1	97.0	3.70	2.465	20.01	1.0	0.0	3.0	NaN
21	15.5	8.0	318.0	150.0	2.76	3.520	16.87	0.0	0.0	3.0	2.0
22	15.2	8.0	304.0	150.0	3.15	NaN	17.30	0.0	0.0	3.0	2.0
23	13.3	8.0	350.0	245.0	3.73	3.840	15.41	NaN	0.0	NaN	4.0
24	19.2	8.0	400.0	175.0	3.08	3.845	17.05	0.0	0.0	3.0	2.0
25	27.3	4.0	79.0	66.0	4.08	1.935	18.90	1.0	1.0	4.0	1.0
26	26.0	4.0	120.3	91.0	4.43	2.140	16.70	0.0	1.0	5.0	2.0
27	30.4	4.0	95.1	113.0	3.77	1.513	16.90	1.0	1.0	5.0	2.0
28	15.8	8.0	351.0	NaN	4.22	3.170	14.50	0.0	1.0	5.0	4.0
29	19.7	6.0	145.0	175.0	3.62	2.770	15.50	0.0	1.0	5.0	6.0
30	15.0	8.0	301.0	335.0	3.54	3.570	14.60	0.0	1.0	5.0	8.0
31	21.4	4.0	121.0	109.0	4.11	2.780	18.60	1.0	1.0	4.0	2.0

In [104]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32 entries, 0 to 31
Data columns (total 11 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   mpg     31 non-null     float64
 1   cyl     30 non-null     float64
 2   disp    31 non-null     float64
 3   hp      30 non-null     float64
 4   drat    30 non-null     float64
 5   wt      30 non-null     float64
 6   qsec    31 non-null     float64
 7   vs      30 non-null     float64
 8   am      31 non-null     float64
 9   gear    30 non-null     float64
 10  carb    30 non-null     float64
dtypes: float64(11)
memory usage: 2.9 KB
```

In [105]:

```
df.describe()
```

Out[105]:

	mpg	cyl	disp	hp	drat	wt	qsec	
count	31.000000	30.000000	31.000000	30.000000	30.000000	30.000000	31.000000	30.00
mean	20.045161	6.333333	235.625806	145.466667	3.598000	3.198233	17.827097	0.46
std	6.120993	1.748563	122.790966	65.760660	0.545302	1.008516	1.812209	0.50
min	10.400000	4.000000	71.100000	52.000000	2.760000	1.513000	14.500000	0.00
25%	15.350000	4.000000	130.900000	99.000000	3.080000	2.503750	16.885000	0.00
50%	19.200000	6.000000	225.000000	123.000000	3.695000	3.202500	17.600000	0.00
75%	22.800000	8.000000	334.000000	180.000000	3.920000	3.690000	18.900000	1.00
max	33.900000	8.000000	472.000000	335.000000	4.930000	5.424000	22.900000	1.00

In [106]:

```
df.isnull()
```

Out[106]:

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	False	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	False
5	False	False	False	False	False	False	False	False	False	False	False
6	False	False	False	False	False	True	False	True	False	False	False
7	False	False	False	False	False	False	False	False	False	False	False
8	False	True	False	False	False	False	False	False	False	False	False
9	False	False	False	False	True	False	False	False	False	False	False
10	False	False	False	False	False	False	False	False	False	False	False
11	False	False	False	False	False	False	False	False	True	False	True
12	False	False	False	False	False	False	False	False	False	False	False
13	False	False	False	False	False	False	False	False	False	True	False
14	False	False	False	False	False	False	False	False	False	False	False
15	False	False	False	False	False	False	False	False	False	False	False
16	False	False	False	False	True	False	False	False	False	False	False
17	False	False	True	True	False	False	False	False	False	False	False
18	False	False	False	False	False	False	True	False	False	False	False
19	False	True	False	False	False	False	False	False	False	False	False
20	True	False	False	False	False	False	False	False	False	False	True
21	False	False	False	False	False	False	False	False	False	False	False
22	False	False	False	False	False	True	False	False	False	False	False
23	False	False	False	False	False	False	False	True	False	True	False
24	False	False	False	False	False	False	False	False	False	False	False
25	False	False	False	False	False	False	False	False	False	False	False
26	False	False	False	False	False	False	False	False	False	False	False
27	False	False	False	False	False	False	False	False	False	False	False
28	False	False	False	True	False	False	False	False	False	False	False
29	False	False	False	False	False	False	False	False	False	False	False
30	False	False	False	False	False	False	False	False	False	False	False
31	False	False	False	False	False	False	False	False	False	False	False

In [107]:

```
df.isnull().sum()
```

Out[107]:

```
mpg     1
cyl     2
disp    1
hp      2
drat    2
wt      2
qsec    1
vs      2
am      1
gear    2
carb    2
dtype: int64
```

In [108]:

```
df.isnull().sum().sum()
```

Out[108]:

```
18
```

In [13]:

```
missing_values=['n/a','na','_']
```

In [14]:

```
df['mpg'].fillna(24,inplace=True)
df
```

Out[14]:

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	21.0	6.0	160.0	110.0	3.90	2.620	16.46	0.0	1.0	4.0	4.0
1	21.0	6.0	160.0	110.0	3.90	2.875	17.02	0.0	1.0	4.0	4.0
2	22.8	4.0	108.0	93.0	3.85	2.320	18.61	1.0	1.0	4.0	1.0
3	21.4	6.0	258.0	110.0	3.08	3.215	19.44	1.0	0.0	3.0	1.0
4	18.7	8.0	360.0	175.0	3.15	3.440	17.02	0.0	0.0	3.0	2.0
5	18.1	6.0	225.0	105.0	2.76	3.460	20.22	1.0	0.0	3.0	1.0
6	14.3	8.0	360.0	245.0	3.21	NaN	15.84	NaN	0.0	3.0	4.0
7	24.4	4.0	146.7	62.0	3.69	3.190	20.00	1.0	0.0	4.0	2.0
8	22.8	NaN	140.8	95.0	3.92	3.150	22.90	1.0	0.0	4.0	2.0
9	19.2	6.0	167.6	123.0	NaN	3.440	18.30	1.0	0.0	4.0	4.0
10	17.8	6.0	167.6	123.0	3.92	3.440	18.90	1.0	0.0	4.0	4.0
11	16.4	8.0	275.8	180.0	3.07	4.070	17.40	0.0	NaN	3.0	NaN
12	17.3	8.0	275.8	180.0	3.07	3.730	17.60	0.0	0.0	3.0	3.0
13	15.2	8.0	275.8	180.0	3.07	3.780	18.00	0.0	0.0	NaN	3.0
14	10.4	8.0	472.0	205.0	2.93	5.250	17.98	0.0	0.0	3.0	4.0
15	10.4	8.0	460.0	215.0	3.00	5.424	17.82	0.0	0.0	3.0	4.0
16	14.7	8.0	440.0	230.0	NaN	5.345	17.42	0.0	0.0	3.0	4.0
17	32.4	4.0	NaN	NaN	4.08	2.200	19.47	1.0	1.0	4.0	1.0
18	30.4	4.0	75.7	52.0	4.93	1.615	NaN	1.0	1.0	4.0	2.0
19	33.9	NaN	71.1	65.0	4.22	1.835	19.90	1.0	1.0	4.0	1.0
20	24.0	4.0	120.1	97.0	3.70	2.465	20.01	1.0	0.0	3.0	NaN
21	15.5	8.0	318.0	150.0	2.76	3.520	16.87	0.0	0.0	3.0	2.0
22	15.2	8.0	304.0	150.0	3.15	NaN	17.30	0.0	0.0	3.0	2.0
23	13.3	8.0	350.0	245.0	3.73	3.840	15.41	NaN	0.0	NaN	4.0
24	19.2	8.0	400.0	175.0	3.08	3.845	17.05	0.0	0.0	3.0	2.0
25	27.3	4.0	79.0	66.0	4.08	1.935	18.90	1.0	1.0	4.0	1.0
26	26.0	4.0	120.3	91.0	4.43	2.140	16.70	0.0	1.0	5.0	2.0
27	30.4	4.0	95.1	113.0	3.77	1.513	16.90	1.0	1.0	5.0	2.0
28	15.8	8.0	351.0	NaN	4.22	3.170	14.50	0.0	1.0	5.0	4.0
29	19.7	6.0	145.0	175.0	3.62	2.770	15.50	0.0	1.0	5.0	6.0
30	15.0	8.0	301.0	335.0	3.54	3.570	14.60	0.0	1.0	5.0	8.0
31	21.4	4.0	121.0	109.0	4.11	2.780	18.60	1.0	1.0	4.0	2.0

In [112]:

```
df.isnull().sum()
```

Out[112]:

```
mpg     0
cyl     2
disp    1
hp      2
drat    2
wt      2
qsec    1
vs      2
am      1
gear    2
carb    2
dtype: int64
```

In [15]:

```
medval=df['cyl'].median()
df['cyl'].fillna(medval,inplace=True)
df
```

Out[15]:

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	21.0	6.0	160.0	110.0	3.90	2.620	16.46	0.0	1.0	4.0	4.0
1	21.0	6.0	160.0	110.0	3.90	2.875	17.02	0.0	1.0	4.0	4.0
2	22.8	4.0	108.0	93.0	3.85	2.320	18.61	1.0	1.0	4.0	1.0
3	21.4	6.0	258.0	110.0	3.08	3.215	19.44	1.0	0.0	3.0	1.0
4	18.7	8.0	360.0	175.0	3.15	3.440	17.02	0.0	0.0	3.0	2.0
5	18.1	6.0	225.0	105.0	2.76	3.460	20.22	1.0	0.0	3.0	1.0
6	14.3	8.0	360.0	245.0	3.21	NaN	15.84	NaN	0.0	3.0	4.0
7	24.4	4.0	146.7	62.0	3.69	3.190	20.00	1.0	0.0	4.0	2.0
8	22.8	6.0	140.8	95.0	3.92	3.150	22.90	1.0	0.0	4.0	2.0
9	19.2	6.0	167.6	123.0	NaN	3.440	18.30	1.0	0.0	4.0	4.0
10	17.8	6.0	167.6	123.0	3.92	3.440	18.90	1.0	0.0	4.0	4.0
11	16.4	8.0	275.8	180.0	3.07	4.070	17.40	0.0	NaN	3.0	NaN
12	17.3	8.0	275.8	180.0	3.07	3.730	17.60	0.0	0.0	3.0	3.0
13	15.2	8.0	275.8	180.0	3.07	3.780	18.00	0.0	0.0	NaN	3.0
14	10.4	8.0	472.0	205.0	2.93	5.250	17.98	0.0	0.0	3.0	4.0
15	10.4	8.0	460.0	215.0	3.00	5.424	17.82	0.0	0.0	3.0	4.0
16	14.7	8.0	440.0	230.0	NaN	5.345	17.42	0.0	0.0	3.0	4.0
17	32.4	4.0	NaN	NaN	4.08	2.200	19.47	1.0	1.0	4.0	1.0
18	30.4	4.0	75.7	52.0	4.93	1.615	NaN	1.0	1.0	4.0	2.0
19	33.9	6.0	71.1	65.0	4.22	1.835	19.90	1.0	1.0	4.0	1.0
20	24.0	4.0	120.1	97.0	3.70	2.465	20.01	1.0	0.0	3.0	NaN
21	15.5	8.0	318.0	150.0	2.76	3.520	16.87	0.0	0.0	3.0	2.0
22	15.2	8.0	304.0	150.0	3.15	NaN	17.30	0.0	0.0	3.0	2.0
23	13.3	8.0	350.0	245.0	3.73	3.840	15.41	NaN	0.0	NaN	4.0
24	19.2	8.0	400.0	175.0	3.08	3.845	17.05	0.0	0.0	3.0	2.0
25	27.3	4.0	79.0	66.0	4.08	1.935	18.90	1.0	1.0	4.0	1.0
26	26.0	4.0	120.3	91.0	4.43	2.140	16.70	0.0	1.0	5.0	2.0
27	30.4	4.0	95.1	113.0	3.77	1.513	16.90	1.0	1.0	5.0	2.0
28	15.8	8.0	351.0	NaN	4.22	3.170	14.50	0.0	1.0	5.0	4.0
29	19.7	6.0	145.0	175.0	3.62	2.770	15.50	0.0	1.0	5.0	6.0
30	15.0	8.0	301.0	335.0	3.54	3.570	14.60	0.0	1.0	5.0	8.0
31	21.4	4.0	121.0	109.0	4.11	2.780	18.60	1.0	1.0	4.0	2.0

In [16]:

```
modev=df['disp'].mode()
modev
```

Out[16]:

```
0    275.8
Name: disp, dtype: float64
```

In [17]:

```
df['disp'].fillna(medval,inplace=True)
df
```

Out[17]:

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	21.0	6.0	160.0	110.0	3.90	2.620	16.46	0.0	1.0	4.0	4.0
1	21.0	6.0	160.0	110.0	3.90	2.875	17.02	0.0	1.0	4.0	4.0
2	22.8	4.0	108.0	93.0	3.85	2.320	18.61	1.0	1.0	4.0	1.0
3	21.4	6.0	258.0	110.0	3.08	3.215	19.44	1.0	0.0	3.0	1.0
4	18.7	8.0	360.0	175.0	3.15	3.440	17.02	0.0	0.0	3.0	2.0
5	18.1	6.0	225.0	105.0	2.76	3.460	20.22	1.0	0.0	3.0	1.0
6	14.3	8.0	360.0	245.0	3.21	NaN	15.84	NaN	0.0	3.0	4.0
7	24.4	4.0	146.7	62.0	3.69	3.190	20.00	1.0	0.0	4.0	2.0
8	22.8	6.0	140.8	95.0	3.92	3.150	22.90	1.0	0.0	4.0	2.0
9	19.2	6.0	167.6	123.0	NaN	3.440	18.30	1.0	0.0	4.0	4.0
10	17.8	6.0	167.6	123.0	3.92	3.440	18.90	1.0	0.0	4.0	4.0
11	16.4	8.0	275.8	180.0	3.07	4.070	17.40	0.0	NaN	3.0	NaN
12	17.3	8.0	275.8	180.0	3.07	3.730	17.60	0.0	0.0	3.0	3.0
13	15.2	8.0	275.8	180.0	3.07	3.780	18.00	0.0	0.0	NaN	3.0
14	10.4	8.0	472.0	205.0	2.93	5.250	17.98	0.0	0.0	3.0	4.0
15	10.4	8.0	460.0	215.0	3.00	5.424	17.82	0.0	0.0	3.0	4.0
16	14.7	8.0	440.0	230.0	NaN	5.345	17.42	0.0	0.0	3.0	4.0
17	32.4	4.0	6.0	NaN	4.08	2.200	19.47	1.0	1.0	4.0	1.0
18	30.4	4.0	75.7	52.0	4.93	1.615	NaN	1.0	1.0	4.0	2.0
19	33.9	6.0	71.1	65.0	4.22	1.835	19.90	1.0	1.0	4.0	1.0
20	24.0	4.0	120.1	97.0	3.70	2.465	20.01	1.0	0.0	3.0	NaN
21	15.5	8.0	318.0	150.0	2.76	3.520	16.87	0.0	0.0	3.0	2.0
22	15.2	8.0	304.0	150.0	3.15	NaN	17.30	0.0	0.0	3.0	2.0
23	13.3	8.0	350.0	245.0	3.73	3.840	15.41	NaN	0.0	NaN	4.0
24	19.2	8.0	400.0	175.0	3.08	3.845	17.05	0.0	0.0	3.0	2.0
25	27.3	4.0	79.0	66.0	4.08	1.935	18.90	1.0	1.0	4.0	1.0
26	26.0	4.0	120.3	91.0	4.43	2.140	16.70	0.0	1.0	5.0	2.0
27	30.4	4.0	95.1	113.0	3.77	1.513	16.90	1.0	1.0	5.0	2.0
28	15.8	8.0	351.0	NaN	4.22	3.170	14.50	0.0	1.0	5.0	4.0
29	19.7	6.0	145.0	175.0	3.62	2.770	15.50	0.0	1.0	5.0	6.0
30	15.0	8.0	301.0	335.0	3.54	3.570	14.60	0.0	1.0	5.0	8.0
31	21.4	4.0	121.0	109.0	4.11	2.780	18.60	1.0	1.0	4.0	2.0

In [18]:

```
df.isnull().sum()
```

Out[18]:

```
mpg     0
cyl     0
disp    0
hp      2
drat    2
wt      2
qsec    1
vs      2
am      1
gear    2
carb    2
dtype: int64
```

In [19]:

```
df['hp'].fillna(110.00,inplace=True)
df['wt'].fillna(2.5,inplace=True)
df
```

Out[19]:

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	21.0	6.0	160.0	110.0	3.90	2.620	16.46	0.0	1.0	4.0	4.0
1	21.0	6.0	160.0	110.0	3.90	2.875	17.02	0.0	1.0	4.0	4.0
2	22.8	4.0	108.0	93.0	3.85	2.320	18.61	1.0	1.0	4.0	1.0
3	21.4	6.0	258.0	110.0	3.08	3.215	19.44	1.0	0.0	3.0	1.0
4	18.7	8.0	360.0	175.0	3.15	3.440	17.02	0.0	0.0	3.0	2.0
5	18.1	6.0	225.0	105.0	2.76	3.460	20.22	1.0	0.0	3.0	1.0
6	14.3	8.0	360.0	245.0	3.21	2.500	15.84	NaN	0.0	3.0	4.0
7	24.4	4.0	146.7	62.0	3.69	3.190	20.00	1.0	0.0	4.0	2.0
8	22.8	6.0	140.8	95.0	3.92	3.150	22.90	1.0	0.0	4.0	2.0
9	19.2	6.0	167.6	123.0	NaN	3.440	18.30	1.0	0.0	4.0	4.0
10	17.8	6.0	167.6	123.0	3.92	3.440	18.90	1.0	0.0	4.0	4.0
11	16.4	8.0	275.8	180.0	3.07	4.070	17.40	0.0	NaN	3.0	NaN
12	17.3	8.0	275.8	180.0	3.07	3.730	17.60	0.0	0.0	3.0	3.0
13	15.2	8.0	275.8	180.0	3.07	3.780	18.00	0.0	0.0	NaN	3.0
14	10.4	8.0	472.0	205.0	2.93	5.250	17.98	0.0	0.0	3.0	4.0
15	10.4	8.0	460.0	215.0	3.00	5.424	17.82	0.0	0.0	3.0	4.0
16	14.7	8.0	440.0	230.0	NaN	5.345	17.42	0.0	0.0	3.0	4.0
17	32.4	4.0	6.0	110.0	4.08	2.200	19.47	1.0	1.0	4.0	1.0
18	30.4	4.0	75.7	52.0	4.93	1.615	NaN	1.0	1.0	4.0	2.0
19	33.9	6.0	71.1	65.0	4.22	1.835	19.90	1.0	1.0	4.0	1.0
20	24.0	4.0	120.1	97.0	3.70	2.465	20.01	1.0	0.0	3.0	NaN
21	15.5	8.0	318.0	150.0	2.76	3.520	16.87	0.0	0.0	3.0	2.0
22	15.2	8.0	304.0	150.0	3.15	2.500	17.30	0.0	0.0	3.0	2.0
23	13.3	8.0	350.0	245.0	3.73	3.840	15.41	NaN	0.0	NaN	4.0
24	19.2	8.0	400.0	175.0	3.08	3.845	17.05	0.0	0.0	3.0	2.0
25	27.3	4.0	79.0	66.0	4.08	1.935	18.90	1.0	1.0	4.0	1.0
26	26.0	4.0	120.3	91.0	4.43	2.140	16.70	0.0	1.0	5.0	2.0
27	30.4	4.0	95.1	113.0	3.77	1.513	16.90	1.0	1.0	5.0	2.0
28	15.8	8.0	351.0	110.0	4.22	3.170	14.50	0.0	1.0	5.0	4.0
29	19.7	6.0	145.0	175.0	3.62	2.770	15.50	0.0	1.0	5.0	6.0
30	15.0	8.0	301.0	335.0	3.54	3.570	14.60	0.0	1.0	5.0	8.0
31	21.4	4.0	121.0	109.0	4.11	2.780	18.60	1.0	1.0	4.0	2.0

In [21]:

```
df.isnull().sum()
```

Out[21]:

```
mpg     0
cyl     0
disp    0
hp      0
drat    2
wt      0
qsec    1
vs      2
am      1
gear    2
carb    2
dtype: int64
```

In [23]:

```
df.sort_values('mpg',ascending=True)
```

Out[23]:

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
15	10.4	8.0	460.0	215.0	3.00	5.424	17.82	0.0	0.0	3.0	4.0
14	10.4	8.0	472.0	205.0	2.93	5.250	17.98	0.0	0.0	3.0	4.0
23	13.3	8.0	350.0	245.0	3.73	3.840	15.41	NaN	0.0	NaN	4.0
6	14.3	8.0	360.0	245.0	3.21	2.500	15.84	NaN	0.0	3.0	4.0
16	14.7	8.0	440.0	230.0	NaN	5.345	17.42	0.0	0.0	3.0	4.0
30	15.0	8.0	301.0	335.0	3.54	3.570	14.60	0.0	1.0	5.0	8.0
13	15.2	8.0	275.8	180.0	3.07	3.780	18.00	0.0	0.0	NaN	3.0
22	15.2	8.0	304.0	150.0	3.15	2.500	17.30	0.0	0.0	3.0	2.0
21	15.5	8.0	318.0	150.0	2.76	3.520	16.87	0.0	0.0	3.0	2.0
28	15.8	8.0	351.0	110.0	4.22	3.170	14.50	0.0	1.0	5.0	4.0
11	16.4	8.0	275.8	180.0	3.07	4.070	17.40	0.0	NaN	3.0	NaN
12	17.3	8.0	275.8	180.0	3.07	3.730	17.60	0.0	0.0	3.0	3.0
10	17.8	6.0	167.6	123.0	3.92	3.440	18.90	1.0	0.0	4.0	4.0
5	18.1	6.0	225.0	105.0	2.76	3.460	20.22	1.0	0.0	3.0	1.0
4	18.7	8.0	360.0	175.0	3.15	3.440	17.02	0.0	0.0	3.0	2.0
9	19.2	6.0	167.6	123.0	NaN	3.440	18.30	1.0	0.0	4.0	4.0
24	19.2	8.0	400.0	175.0	3.08	3.845	17.05	0.0	0.0	3.0	2.0
29	19.7	6.0	145.0	175.0	3.62	2.770	15.50	0.0	1.0	5.0	6.0
0	21.0	6.0	160.0	110.0	3.90	2.620	16.46	0.0	1.0	4.0	4.0
1	21.0	6.0	160.0	110.0	3.90	2.875	17.02	0.0	1.0	4.0	4.0
3	21.4	6.0	258.0	110.0	3.08	3.215	19.44	1.0	0.0	3.0	1.0
31	21.4	4.0	121.0	109.0	4.11	2.780	18.60	1.0	1.0	4.0	2.0
8	22.8	6.0	140.8	95.0	3.92	3.150	22.90	1.0	0.0	4.0	2.0
2	22.8	4.0	108.0	93.0	3.85	2.320	18.61	1.0	1.0	4.0	1.0
20	24.0	4.0	120.1	97.0	3.70	2.465	20.01	1.0	0.0	3.0	NaN
7	24.4	4.0	146.7	62.0	3.69	3.190	20.00	1.0	0.0	4.0	2.0
26	26.0	4.0	120.3	91.0	4.43	2.140	16.70	0.0	1.0	5.0	2.0
25	27.3	4.0	79.0	66.0	4.08	1.935	18.90	1.0	1.0	4.0	1.0
27	30.4	4.0	95.1	113.0	3.77	1.513	16.90	1.0	1.0	5.0	2.0
18	30.4	4.0	75.7	52.0	4.93	1.615	NaN	1.0	1.0	4.0	2.0
17	32.4	4.0	6.0	110.0	4.08	2.200	19.47	1.0	1.0	4.0	1.0
19	33.9	6.0	71.1	65.0	4.22	1.835	19.90	1.0	1.0	4.0	1.0

In [26]:

```
df[df.mpg.duplicated()]
```

Out[26]:

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
1	21.0	6.0	160.0	110.0	3.90	2.875	17.02	0.0	1.0	4.0	4.0
8	22.8	6.0	140.8	95.0	3.92	3.150	22.90	1.0	0.0	4.0	2.0
15	10.4	8.0	460.0	215.0	3.00	5.424	17.82	0.0	0.0	3.0	4.0
22	15.2	8.0	304.0	150.0	3.15	2.500	17.30	0.0	0.0	3.0	2.0
24	19.2	8.0	400.0	175.0	3.08	3.845	17.05	0.0	0.0	3.0	2.0
27	30.4	4.0	95.1	113.0	3.77	1.513	16.90	1.0	1.0	5.0	2.0
31	21.4	4.0	121.0	109.0	4.11	2.780	18.60	1.0	1.0	4.0	2.0

Sklearn

In [31]:

```
from sklearn.preprocessing import StandardScaler
```

In [33]:

```
from sklearn.preprocessing import Normalizer
```

In [35]:

```
df.groupby(['mpg']).groups.keys()
```

Out[35]:

```
dict_keys([10.4, 13.3, 14.3, 14.7, 15.0, 15.2, 15.5, 15.8, 16.4, 17.3, 17.
8, 18.1, 18.7, 19.2, 19.7, 21.0, 21.4, 22.8, 24.0, 24.4, 26.0, 27.3, 30.4,
32.4, 33.9])
```

In [37]:

```
df.groupby(['gear']).groups.keys()
```

Out[37]:

```
dict_keys([3.0, 4.0, 5.0])
```

In [39]:

```
list1=[55,65,86,98,54,75,68,88]
meanv=np.mean(list1)
meanv
```

Out[39]:

```
73.625
```

In [41]:

```
from scipy import stats
modeV=stats.mode(list1)
modeV
```

```
C:\Users\Sachin sirohi\AppData\Local\Temp\ipykernel_7800\3747523138.py:2:
FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`),
the default behavior of `mode` typically preserves the axis it acts along.
In SciPy 1.11.0, this behavior will change: the default value of `keepdims
` will become False, the `axis` over which the statistic is taken will be
eliminated, and the value None will no longer be accepted. Set `keepdims`
to True or False to avoid this warning.
  modeV=stats.mode(list1)
```

Out[41]:

```
ModeResult(mode=array([54]), count=array([1]))
```

In [7]:

```
from sklearn.metrics import accuracy_score
from sklearn.impute import SimpleImputer
```

In [9]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
url="https://raw.githubusercontent.com/callxpert/datasets/master/data-scientist-salaries
names=['Years-experience','Salary']
df=pd.read_csv(url,names=names)
print(df)
```

```
   Years-experience  Salary
0                 1  110000
1                 2  120000
2                 3  130000
3                 4  140000
4                 5  150000
5                 6  160000
6                 7  170000
7                 8  180000
8                 9  190000
9                10  200000
```

In [11]:

```
from sklearn.model_selection import train_test_split
```

In [19]:

```
# X is standard if data we have
# x is predited for use
x=df['Years-experience']
y=df['Salary']
```

In [25]:

```
#Any four entry selected for test of 20% data
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25,random_state=2)
```

In [28]:

```
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeClassifier
model=LinearRegression()
# model.fit(x_train,y_train)
# print(model.score(x_test, y_test))
```

Statatistics- Numpy, Scipy

In [3]:

```
from scipy import stats
data=[98,80,70,40,65,68,72,62,62,45,62]
x=stats.mode(data)
print(x)
```

```
ModeResult(mode=array([62]), count=array([3]))
```

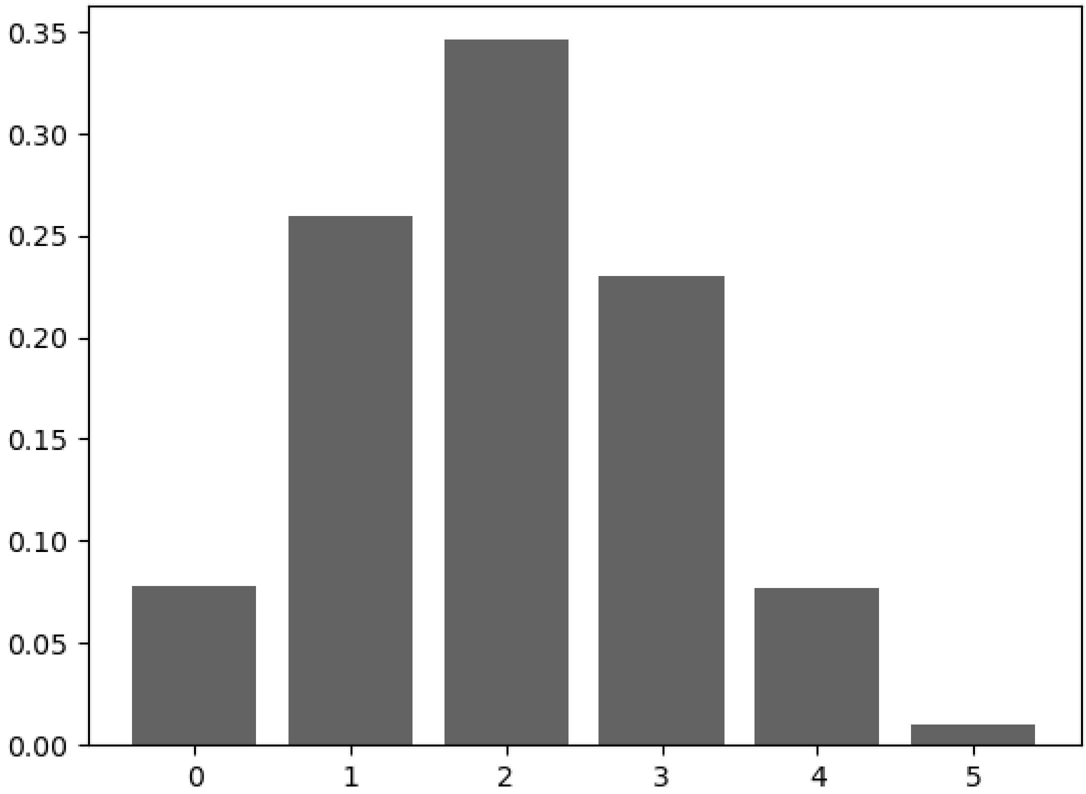
```
C:\Users\Sachin sirohi\AppData\Local\Temp\ipykernel_1528\3689626260.py:3:
FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`),
the default behavior of `mode` typically preserves the axis it acts along.
In SciPy 1.11.0, this behavior will change: the default value of `keepdims
` will become False, the `axis` over which the statistic is taken will be
eliminated, and the value None will no longer be accepted. Set `keepdims`
to True or False to avoid this warning.
  x=stats.mode(data)
```

Binomail Distribution

In [9]:

```
from scipy.stats import binom
import matplotlib.pyplot as plt
n=5
p=0.4
r_values=list(range(n+1))
dist=[binom.pmf(r,n,p) for r in r_values]
print(dist)
plt.bar(r_values,dist)
plt.show()
```

```
[0.07775999999999998, 0.2592000000000001, 0.34559999999999974, 0.230399999
99999994, 0.0768, 0.010240000000000003]
```



In [11]:

```
def Emp_Info():
    a1=50
    b1=60
    c=a1+b1
    print("Addition of two number ",c)
Emp_Info()
```

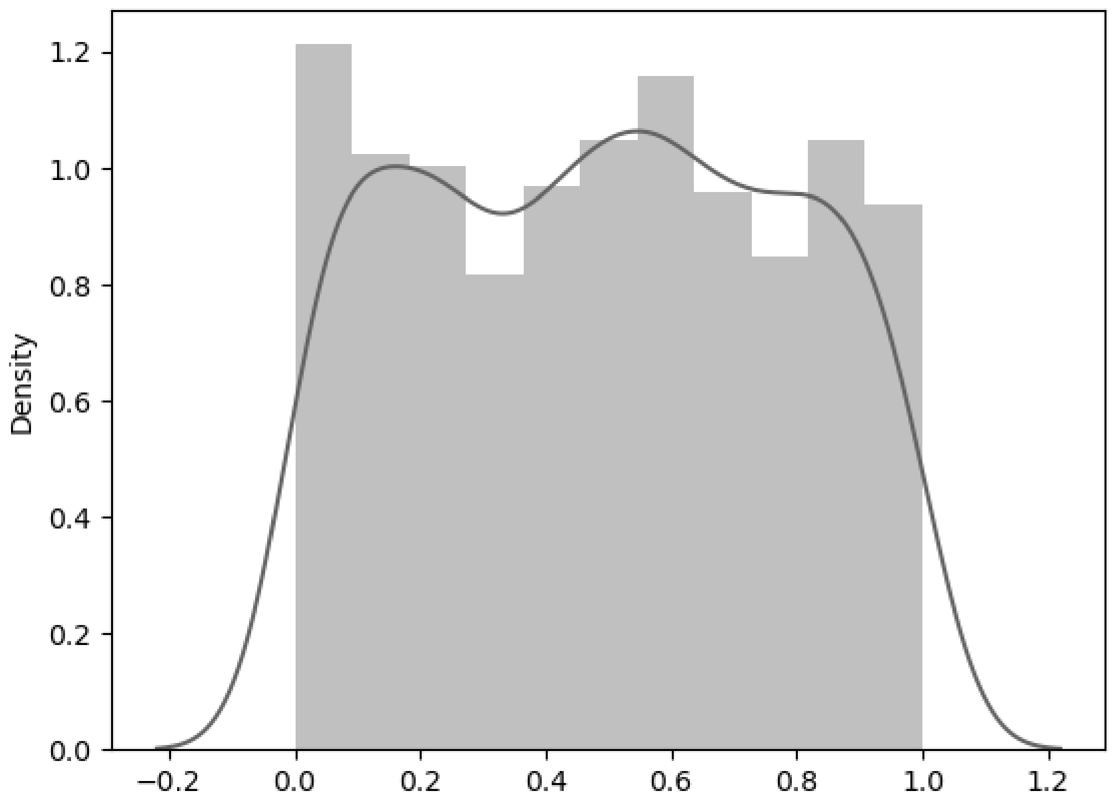
```
Addition of two number  110
```

Continous Uniform Distribution

In [14]:

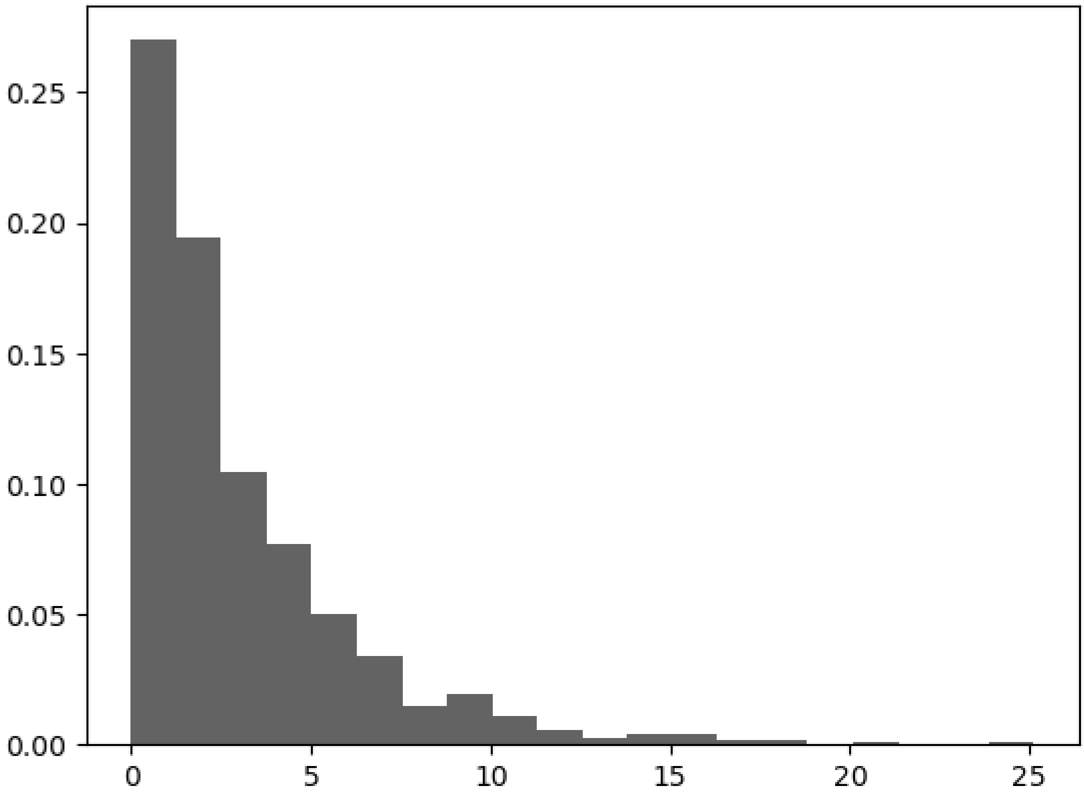
```
from numpy import random
import matplotlib.pyplot as plt
import seaborn as sb
def uniformDist():
    sb.distplot(random.uniform(size=1000),hist=True)
    plt.show()
uniformDist()
```

```
C:\Users\Sachin sirohi\AppData\Local\Temp\ipykernel_1528\25858918.py:5: Us
erWarning:

`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 wi
th
similar flexibility) or `histplot` (an axes-level function for histogram
s).

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

  sb.distplot(random.uniform(size=1000),hist=True)
```



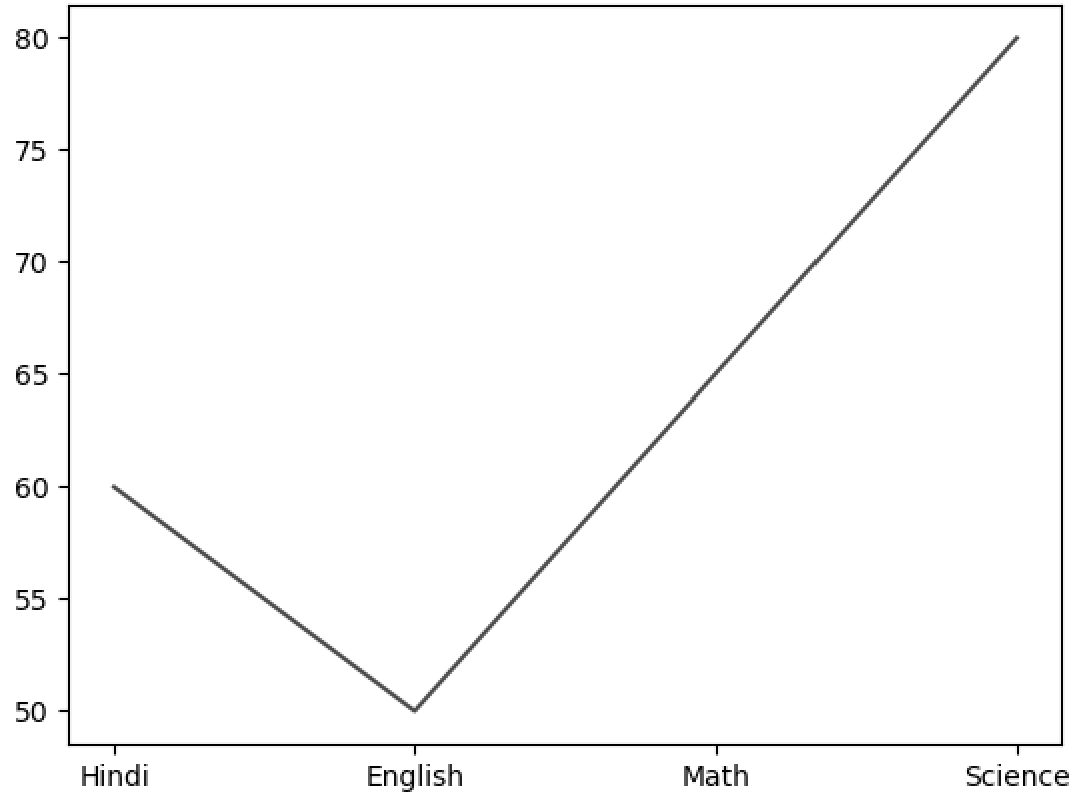
In [23]:

```
import numpy as np
import matplotlib.pyplot as plt
g=np.random.exponential(3,1000)
a,b,c=plt.hist(g,20,density=True)
plt.show()
```



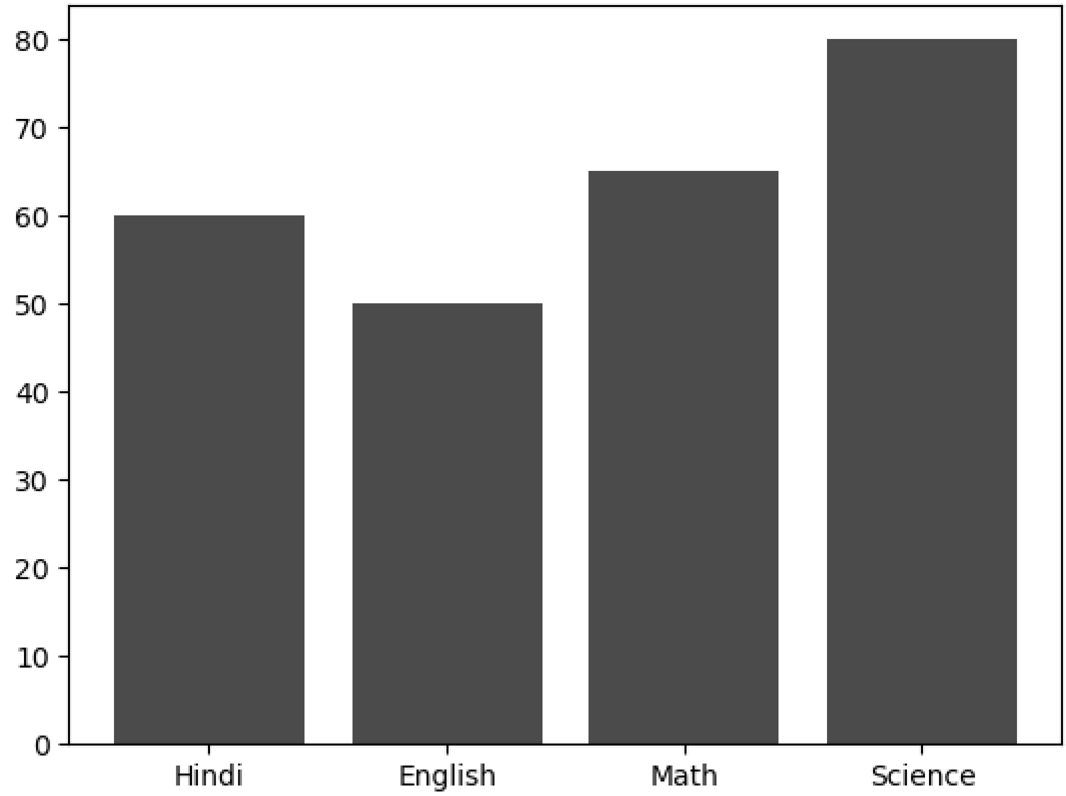
In [60]:

```
from matplotlib import pyplot as plt
Sub=['Hindi','English','Math','Science']
Marks=[60,50,65,80]
plt.plot(Sub,Marks,color='red')
plt.show()
```



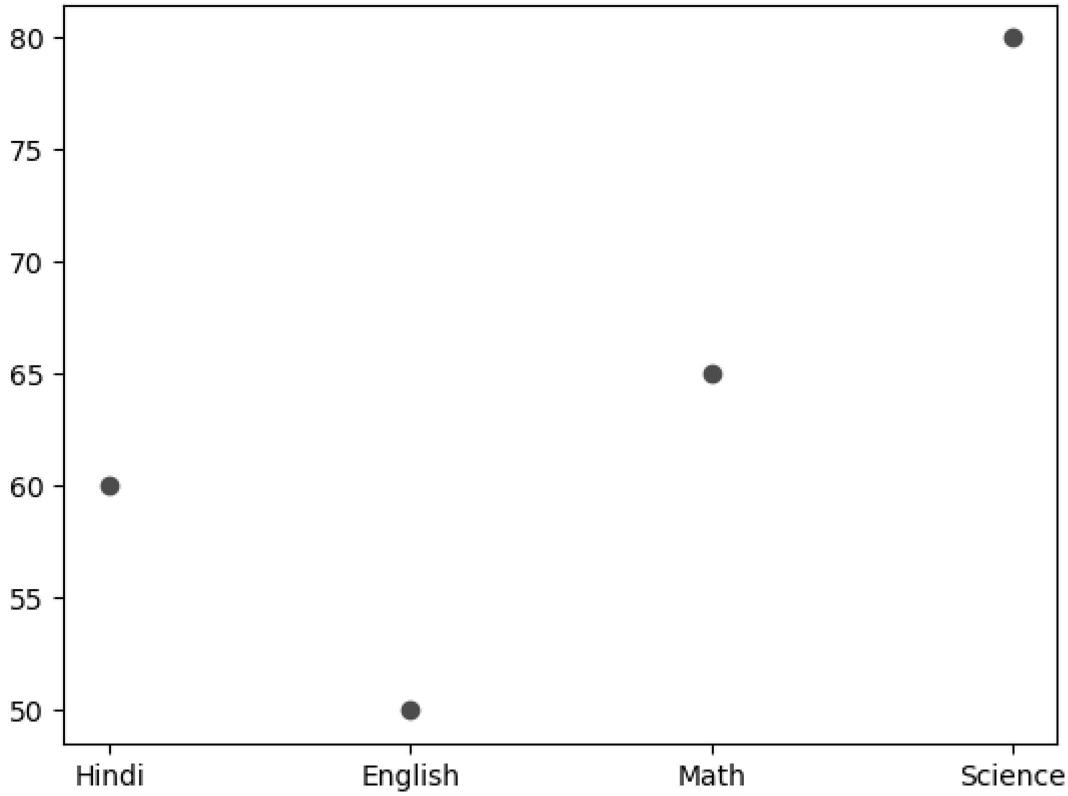
In [57]:

```
import matplotlib.pyplot as plt
Sub=['Hindi','English','Math','Science']
Marks=[60,50,65,80]
plt.bar(Sub,Marks,color='green')
plt.show()
```



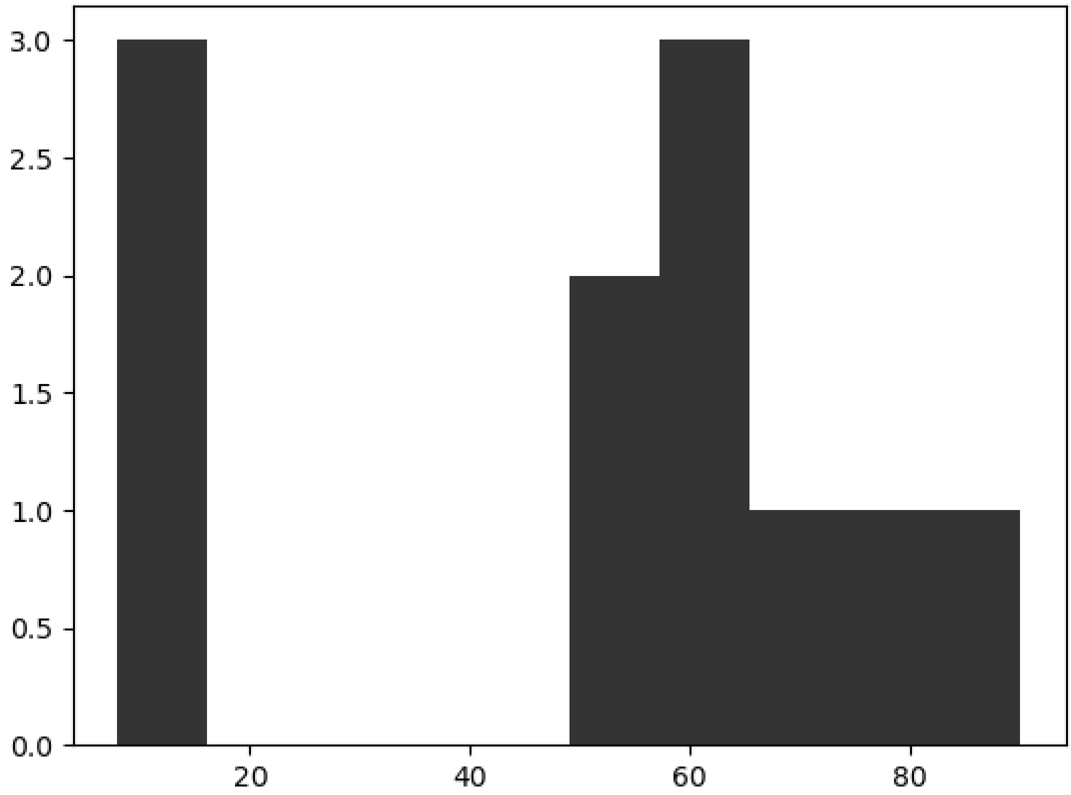
In [62]:

```
import matplotlib.pyplot as plt
Sub=['Hindi','English','Math','Science']
Marks=[60,50,65,80]
plt.scatter(Sub,Marks,color='red')
plt.show()
```



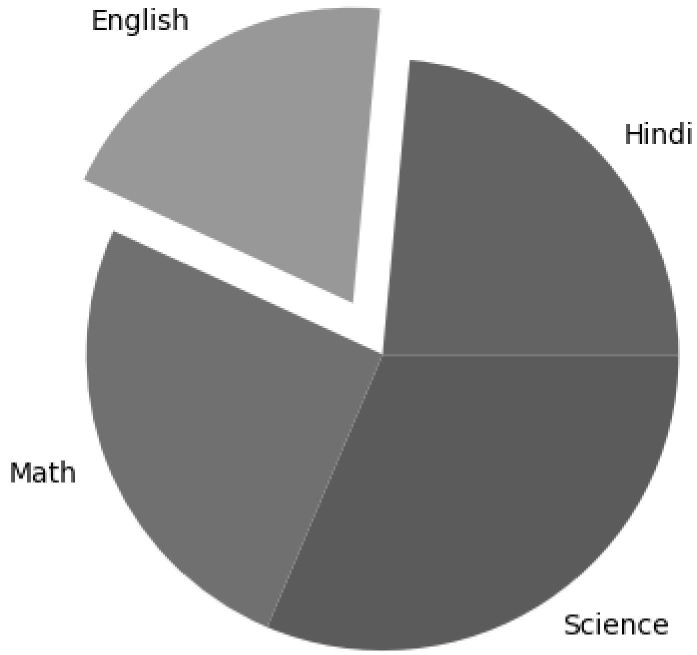
In [67]:

```
import matplotlib.pyplot as plt
Marks=[60,50,65,80,70,90,8,50,60,10,10,]
plt.hist(Marks,color="purple")
plt.show()
```



In [83]:

```
import matplotlib.pyplot as plt
Sub=['Hindi','English','Math','Science']
Marks=[60,50,65,80]
explode=(0,.2,0,0)
plt.pie(Marks,labels=Sub,explode=explode)
plt.show()
```



In [1]:

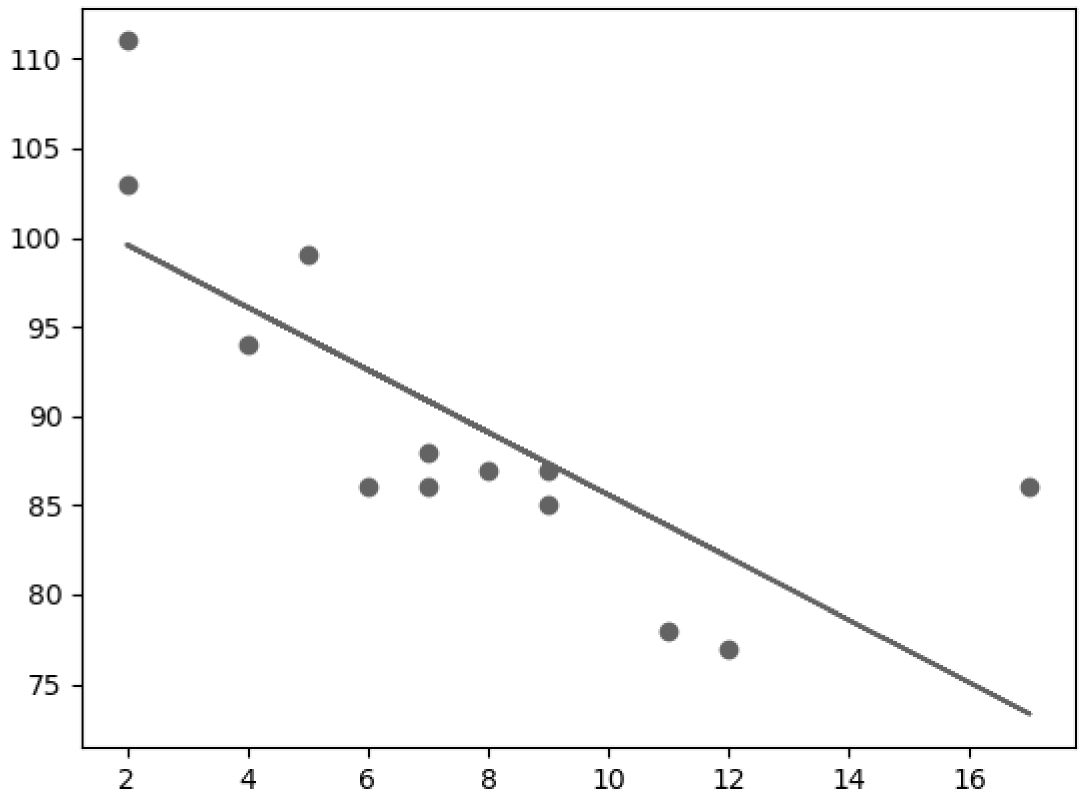
```
import pandas as pd
import matplotlib.pyplot as plt
lst=['Henry','Python','Development','course']
df=pd.DataFrame(lst)
df
```

Out[1]:

	0
0	Henry
1	Python
2	Development
3	course

Linear Regression

Linear regression uses the relationship between the data-points to draw a straight line through all them.



110
105
100
95
90
85
80
75
2
4
6
8
10
12
14
16

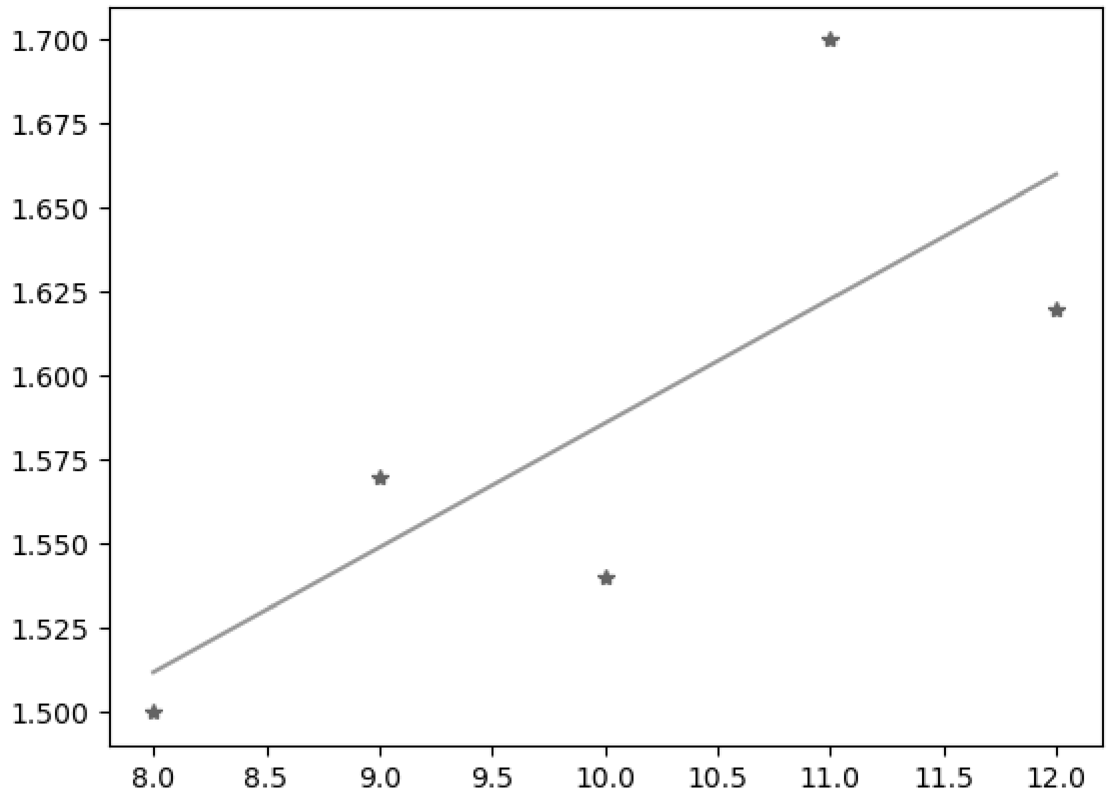
In [4]:

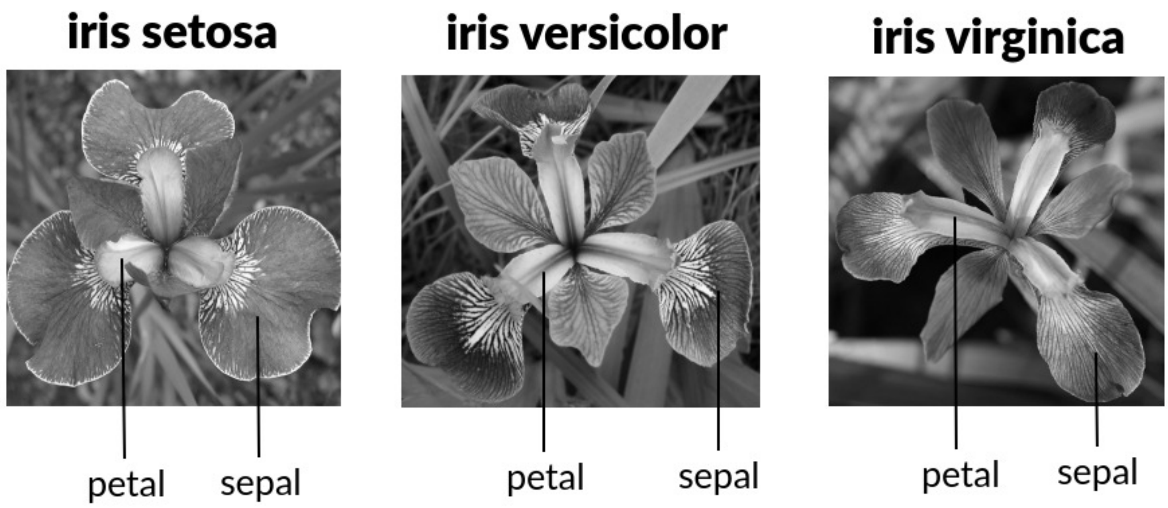
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
x = np.array([8,9,10,11,12])
y = np.array([1.5,1.57,1.54,1.7,1.62])
k, d = np.polyfit(x, y, 1)
print(k," ",d)
y_pred = k*x + d
print(y_pred)
plt.plot(x, y, '*')
plt.plot(x, y_pred)
# plt.show()
```

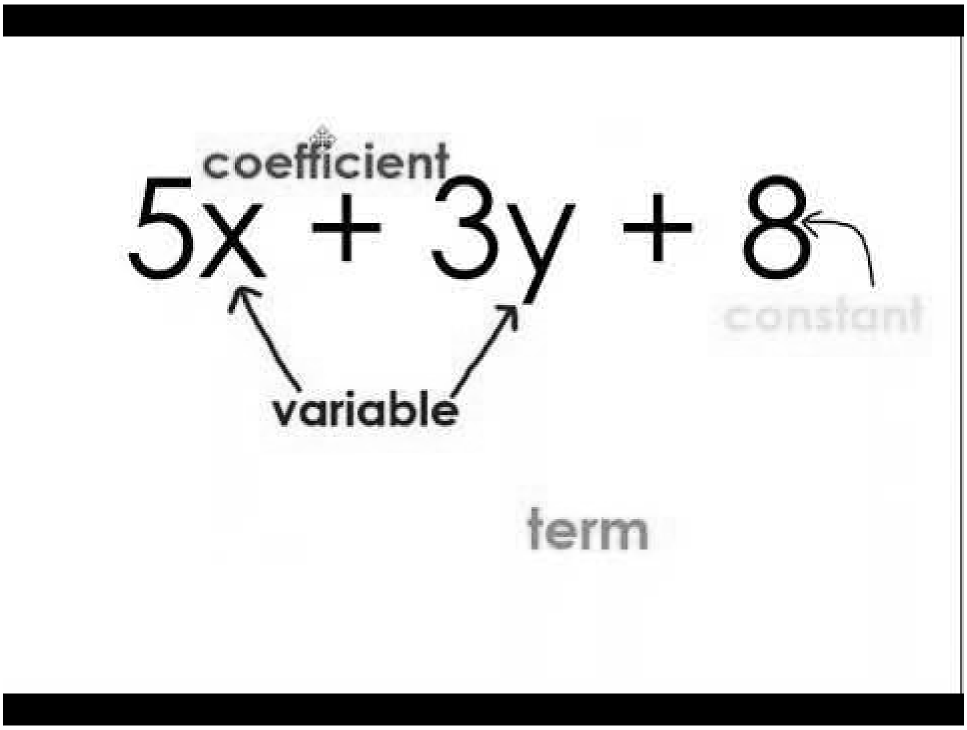
```
0.037000000000000026   1.2159999999999999
[1.512 1.549 1.586 1.623 1.66 ]
```

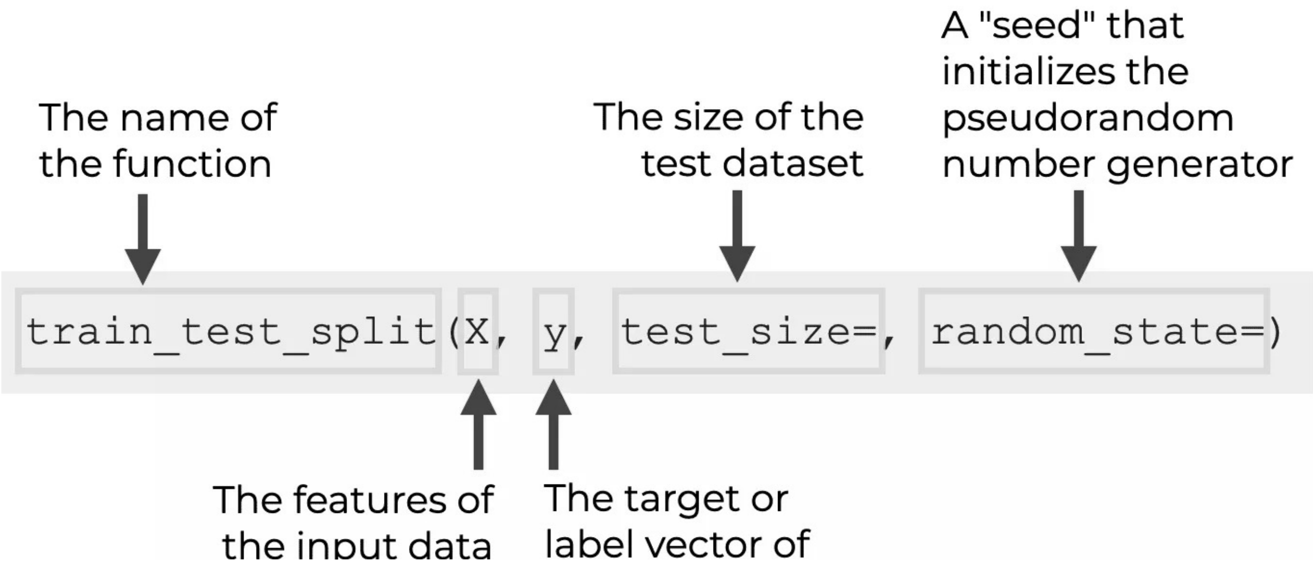
Out[4]:

```
[<matplotlib.lines.Line2D at 0x18f7c9898b0>]
```




iris setosa
iris versicolor
iris virginica
petal
sepal
petal
sepal
petal
sepal


coefficient
5x + 3y + 8
constant
variable
term



A "seed" that initializes the pseudorandom number generator
The name of the function
The size of the test dataset
train_test_split(X, y, test_size=, random_state=)
The features of the input data
The target or label vector of

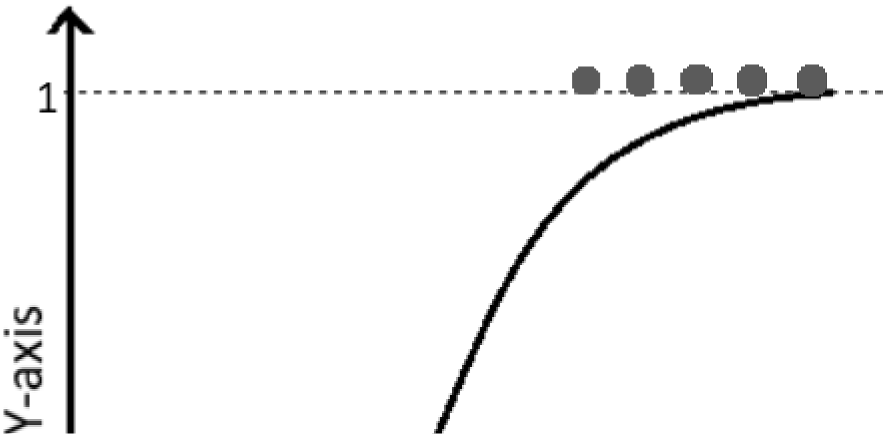
In [5]:

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.datasets import load_iris
# Loading our load_iris dataset
X, Y = load_iris(return_X_y=True)
print(X,Y)
#Printing the shape of the complete dataset
print(X.shape)
# Splitting the dataset into the training and validating datasets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.40, random_state
# # Printing the shape of training and validation data
print(X_train.shape, Y_train.shape)
print(X_test.shape, Y_test.shape)
# # Training the model using the training dataset
lreg = LinearRegression()
lreg.fit(X_train, Y_train)
# # Printing the Coefficients of the linear Regression model
print("Coefficients of each feature: ", lreg.coef_)
# # Printing the accuracy score of the trained model
score = lreg.score(X_test, Y_test)
print("Accuracy Score: ", score)
```

```
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]
 [4.8 3.4 1.6 0.2]
 [4.8 3.  1.4 0.1]
 [4.3 3.  1.1 0.1]
 [5.8 4.  1.2 0.2]
 [5.7 4.4 1.5 0.4]
 [5.4 3.9 1.3 0.4]
 [5.1 3.5 1.4 0.3]
 [5.7 3.8 1.7 0.3]
```

Logistic Regression

Logistic regression aims to solve classification problems. It does this by predicting categorical outcomes, unlike linear regression that predicts a continuous outcome.



In [1]:

```
import seaborn as sns
import pandas as pd
#import dataset from CSV file on Github
url = "https://raw.githubusercontent.com/Statology/Python-Guides/main/default.csv"
df = pd.read_csv(url)
#view first six rows of dataset
sns.regplot(x=x, y=y, data=df, logistic=True, ci=None)
df[0:6]
```

```
---------------------------------------------------------------------------
----
TimeoutError                              Traceback (most recent call l
ast)
F:\Software\Data Science\AnacondInstallFile\lib\urllib\request.py in do
_open(self, http_class, req, **http_conn_args)
   1345             try:
-> 1346                 h.request(req.get_method(), req.selector, req.d
ata, headers,
   1347                           encode_chunked=req.has_header('Transf
er-encoding'))

F:\Software\Data Science\AnacondInstallFile\lib\http\client.py in reque
st(self, method, url, body, headers, encode_chunked)
   1284         """Send a complete request to the server."""
-> 1285         self._send_request(method, url, body, headers, encode_c
hunked)
   1286
```

In [62]:

```
from sklearn.datasets import load_iris
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
# Loading our dataset
data = load_iris()
# Splitting the independent and dependent variables
X = data.data
Y = data.target
print(X," ",Y)
print("The size of the complete dataset is: ", len(X))
# Creating an instance of the LogisticRegression class for implementing logistic regress
log_reg = LogisticRegression()
# Segregating the training and testing dataset
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state =
# Performing the logistic regression on train dataset
log_reg.fit(X_train, Y_train)
# Printing the accuracy score
print("Accuracy score of the predictions made by the model: ", accuracy_score(log_reg.pr
```

```
 [4.5 2.3 1.3 0.3]
 [4.4 3.2 1.3 0.2]
 [5.  3.5 1.6 0.6]
 [5.1 3.8 1.9 0.4]
 [4.8 3.  1.4 0.3]
 [5.1 3.8 1.6 0.2]
 [4.6 3.2 1.4 0.2]
 [5.3 3.7 1.5 0.2]
 [5.  3.3 1.4 0.2]
 [7.  3.2 4.7 1.4]
 [6.4 3.2 4.5 1.5]
 [6.9 3.1 4.9 1.5]
 [5.5 2.3 4.  1.3]
 [6.5 2.8 4.6 1.5]
 [5.7 2.8 4.5 1.3]
 [6.3 3.3 4.7 1.6]
 [4.9 2.4 3.3 1. ]
 [6.6 2.9 4.6 1.3]
 [5.2 2.7 3.9 1.4]
 [5.  2.  3.5 1. ]
```

In [ ]:

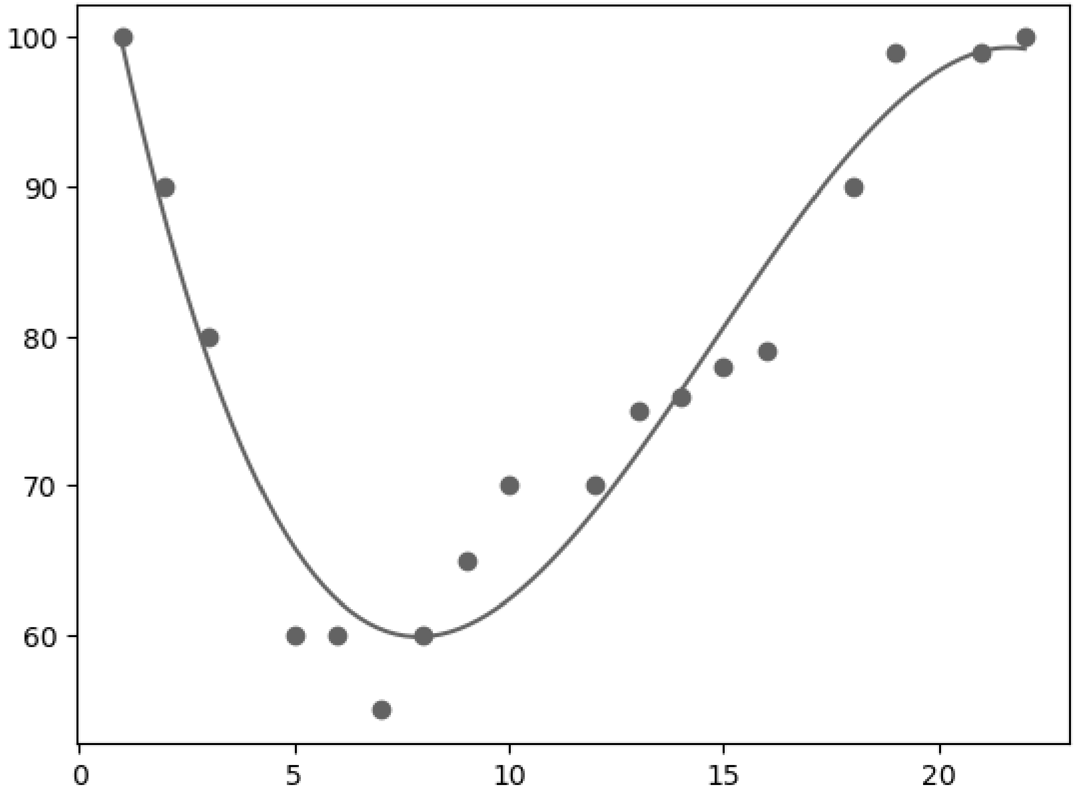
Polynomial Regression

If your data points clearly will not fit a linear regression (a straight line through all data points), it might be ideal for polynomial regression.

Polynomial regression, like linear regression, uses the relationship between the variables x and y to find the best way to draw a line through the data points.

In [67]:

```
import numpy
import matplotlib.pyplot as plt

x = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,18,19,21,22]
y = [100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,100]

mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))

myline = numpy.linspace(1, 22, 100)

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



Train And Test Data With SkLearn

In [74]:

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
iris = load_iris()
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size = 0.3, random_state
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(105, 4)
(45, 4)
(105,)
(45,)
```

Load Iris Data

In [9]:

```
import seaborn as sns
iris = sns.load_dataset('iris')
iris.head()
```

Out[9]:

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

PreProcessing with SKlearn

In [78]:

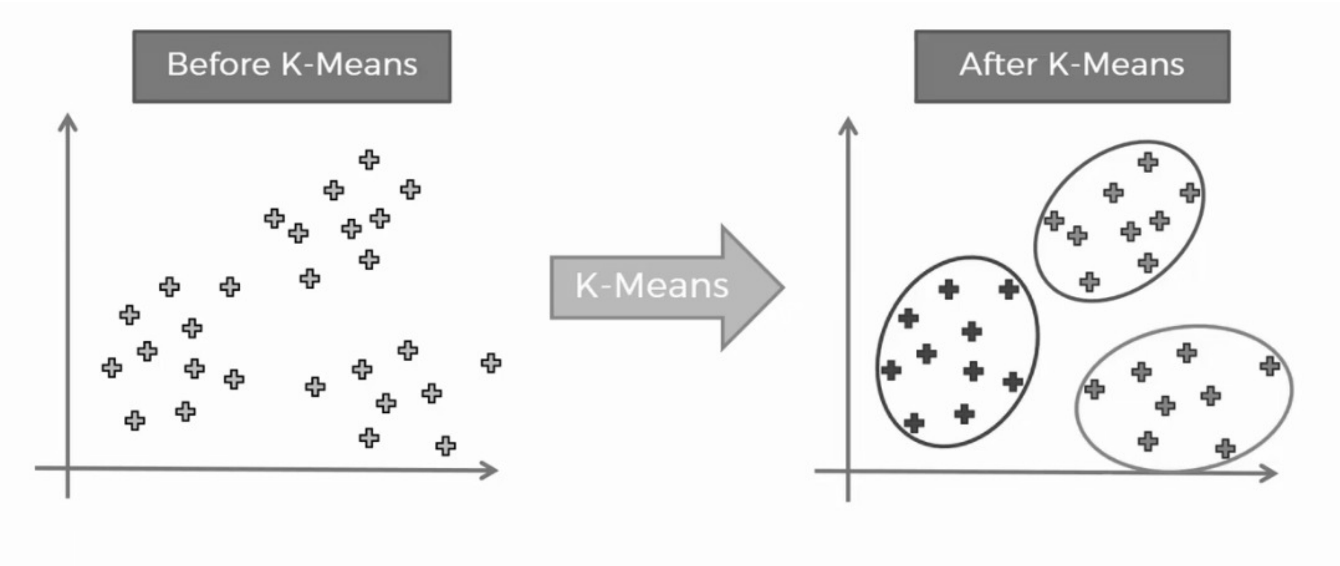
```
import numpy as np
from sklearn import preprocessing
input_data = np.array([
 [2.1, -1.9, 5.5],
 [-1.5, 2.4, 3.5],
 [0.5, -7.9, 5.6],
 [5.9, 2.3, -5.8]]
)
print(input_data)
data_binarized = preprocessing.Binarizer(threshold=0.5).transform(input_data)
print("\nBinarized data:\n", data_binarized)
```

```
[[ 2.1 -1.9  5.5]
 [-1.5  2.4  3.5]
 [ 0.5 -7.9  5.6]
 [ 5.9  2.3 -5.8]]

Binarized data:
 [[1. 0. 1.]
 [0. 1. 1.]
 [0. 0. 1.]
 [1. 1. 0.]]
```

K-means

K-means is an unsupervised learning method for clustering data points. The algorithm iteratively divides data points into K clusters by minimizing the variance in each cluster.

Here, we will show you how to estimate the best value for K using the elbow method, then use K-means clustering to group the data points into clusters.



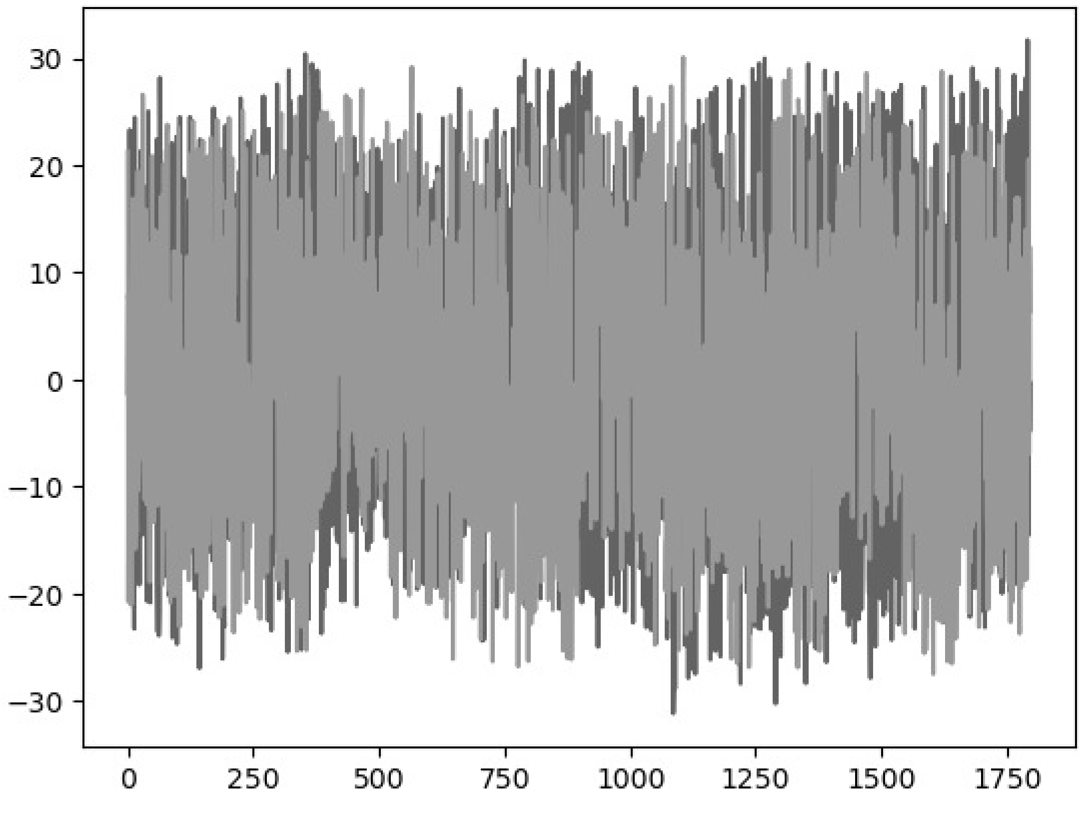
In [113]:

```
from sklearn.datasets import load_digits
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from matplotlib import pyplot as plt
import numpy as np
data=load_digits().data
print(data.shape)
pca=PCA(2)
# print(pca)
#Transpose the data
df=pca.fit_transform(data)
df.shape
plt.plot(df)
```

```
(1797, 64)
```

Out[113]:

```
[<matplotlib.lines.Line2D at 0x16a1b0ef7f0>,
 <matplotlib.lines.Line2D at 0x16a1b0ef190>]
```



Steps for Plotting K-Means Clusters

In [115]:

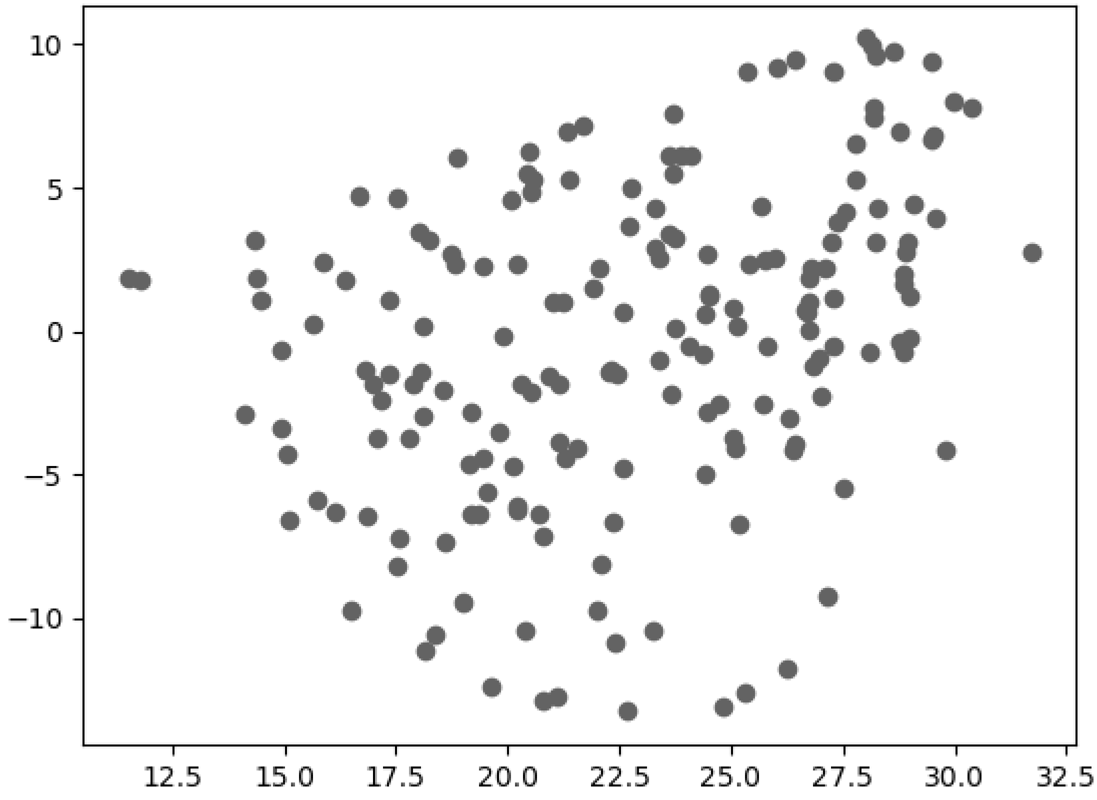
```
from sklearn.cluster import KMeans
kmeans=KMeans(n_clusters=10)
#predict the lebels of clusters
label=kmeans.fit_predict(df)
print(label)
```

```
[6 4 7 ... 7 2 1]
```

Find Particular Label

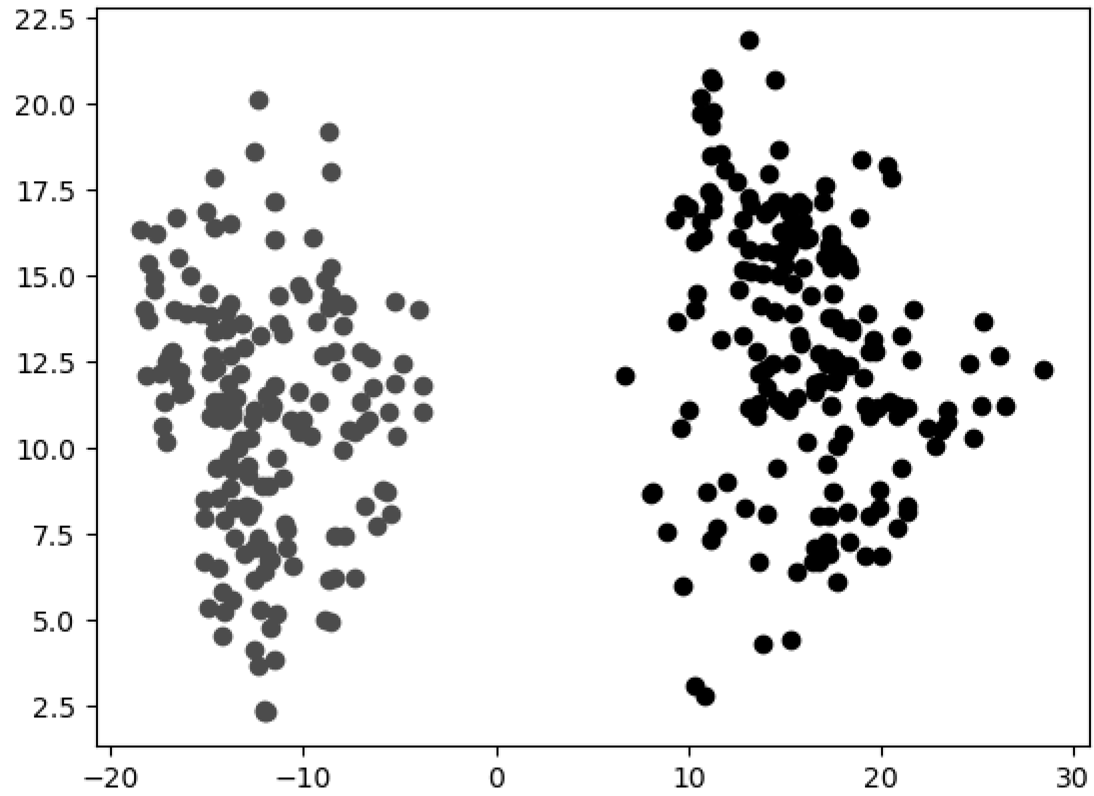
In [117]:

```
import matplotlib.pyplot as plt
flabel=df[label==0]
plt.scatter(flabel[:,0],flabel[:,1])
plt.show()
```



In [121]:

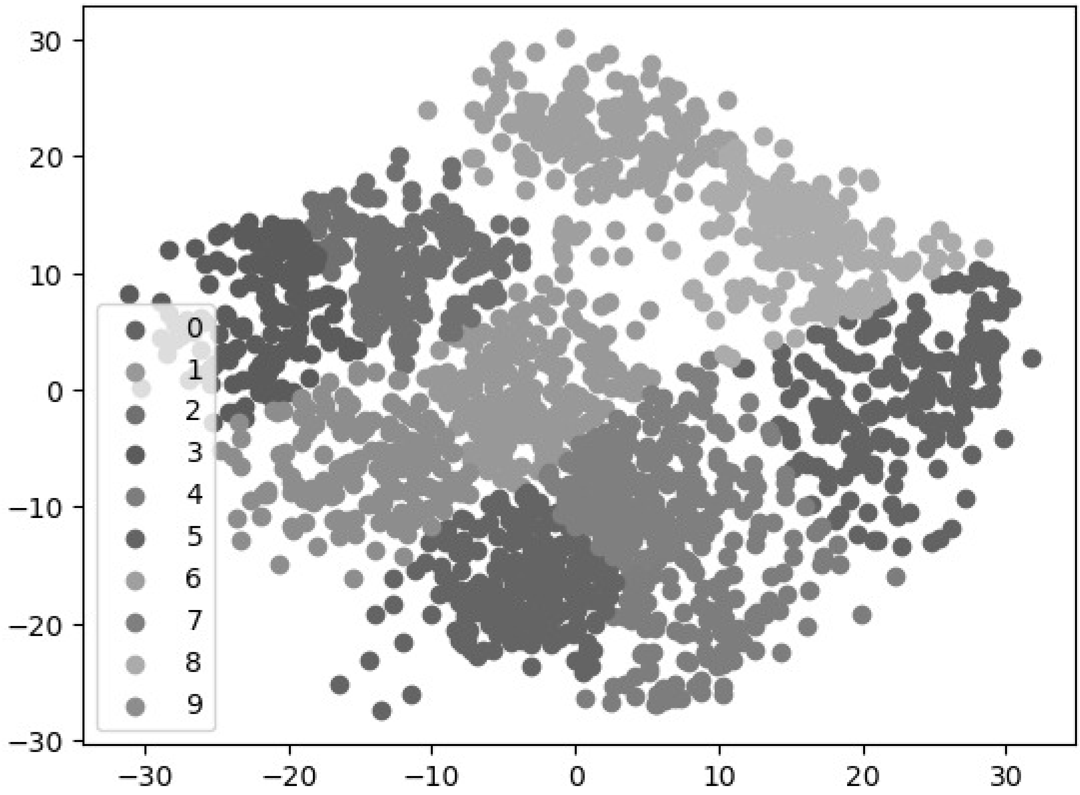
```
flabel2=df[label==2]
flabel8=df[label==8]
plt.scatter(flabel2[:,0],flabel2[:,1],color='red')
plt.scatter(flabel8[:,0],flabel8[:,1],color='black')
plt.show()
```



Show Unique Labels

In [124]:

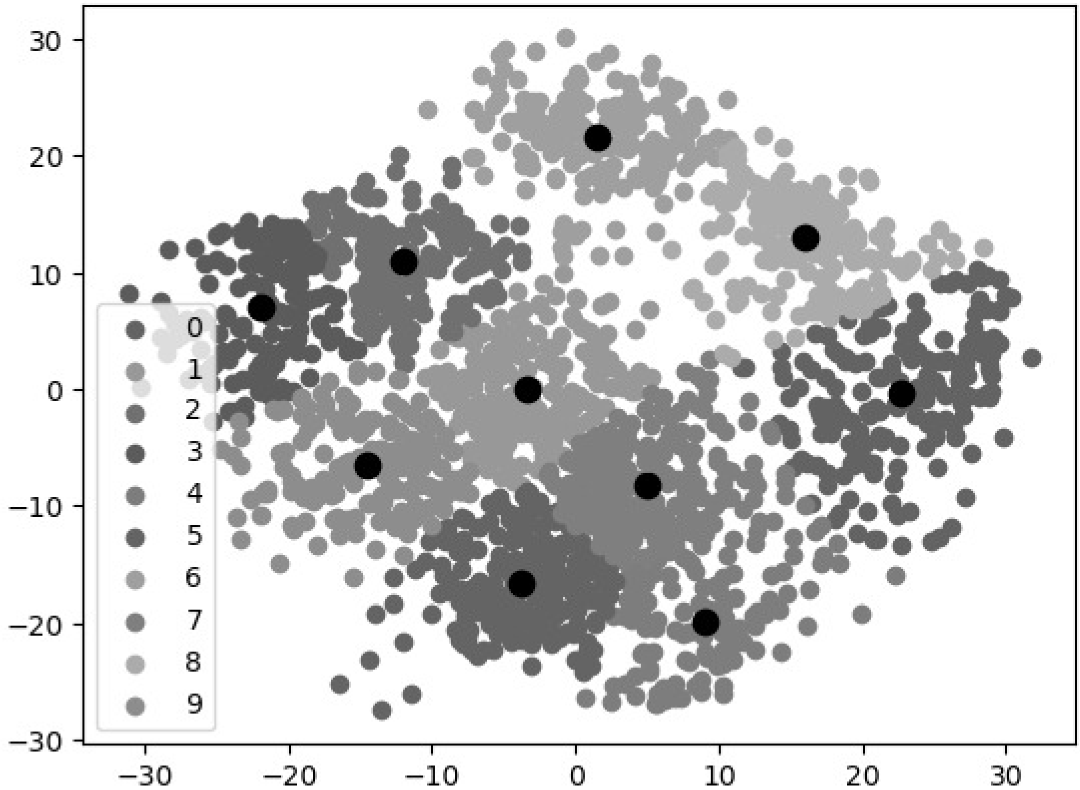
```
u_labels=np.unique(label)
for i in u_labels:
    plt.scatter(df[label==i,0],df[label==i,1],label=i)
plt.legend()
plt.show()
```



Centroids or Center Values using K-Means

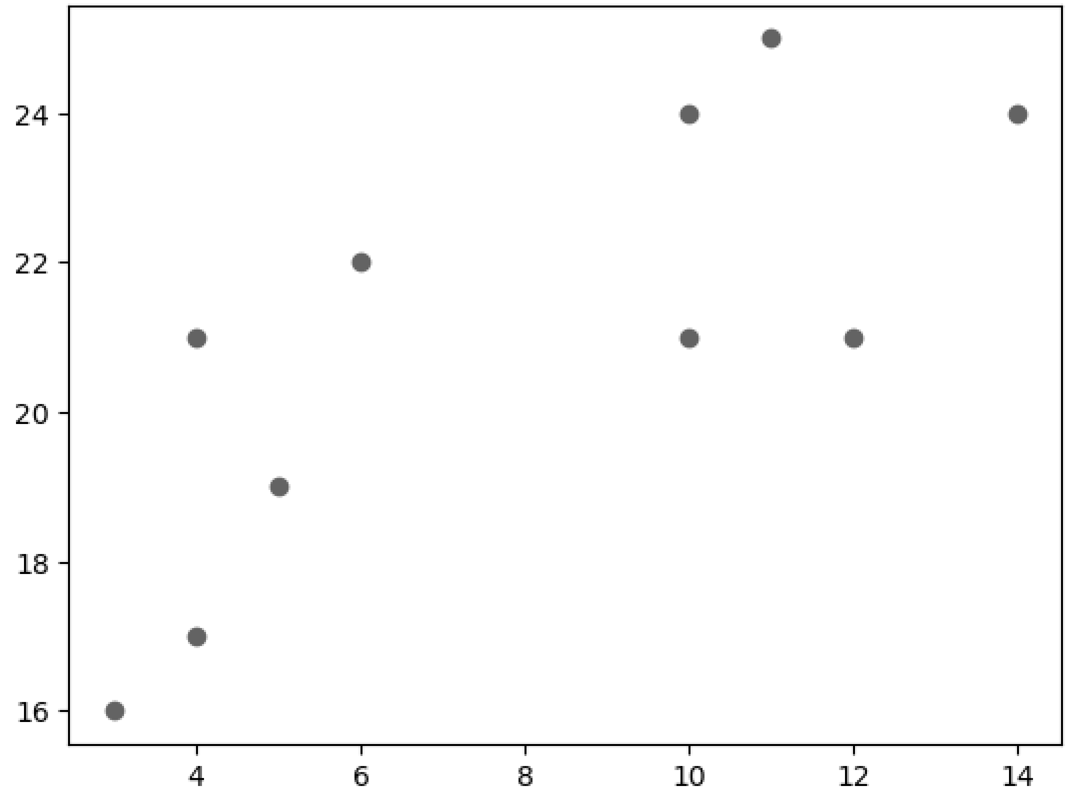
In [131]:

```
#Getting the centroids
centroids=kmeans.cluster_centers_
u_labels=np.unique(label)
for i in u_labels:
    plt.scatter(df[label==i,0],df[label==i,1],label=i)
plt.scatter(centroids[:,0],centroids[:,1],s=80,color='k')
plt.legend()
plt.show()
```



In [68]:

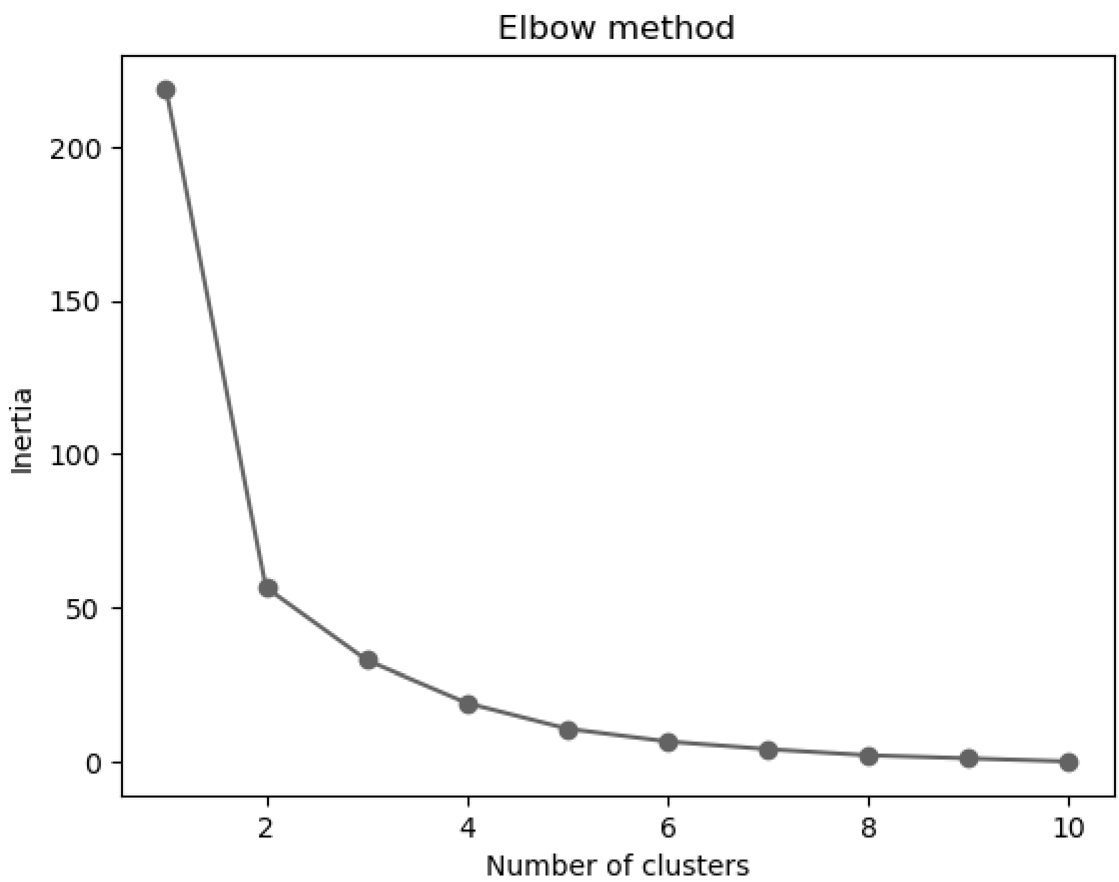
```
import matplotlib.pyplot as plt

x = [4, 5, 10, 4, 3, 11, 14 , 6, 10, 12]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]

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



In [70]:

```
from sklearn.cluster import KMeans

data = list(zip(x, y))
inertias = []

for i in range(1,11):
    kmeans = KMeans(n_clusters=i)
    kmeans.fit(data)
    inertias.append(kmeans.inertia_)

plt.plot(range(1,11), inertias, marker='o')
plt.title('Elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.show()
```

```
F:\Software\Data Science\AnacondInstallFile\lib\site-packages\sklearn\clus
ter\_kmeans.py:1036: UserWarning: KMeans is known to have a memory leak on
Windows with MKL, when there are less chunks than available threads. You c
an avoid it by setting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
```



In [ ]:

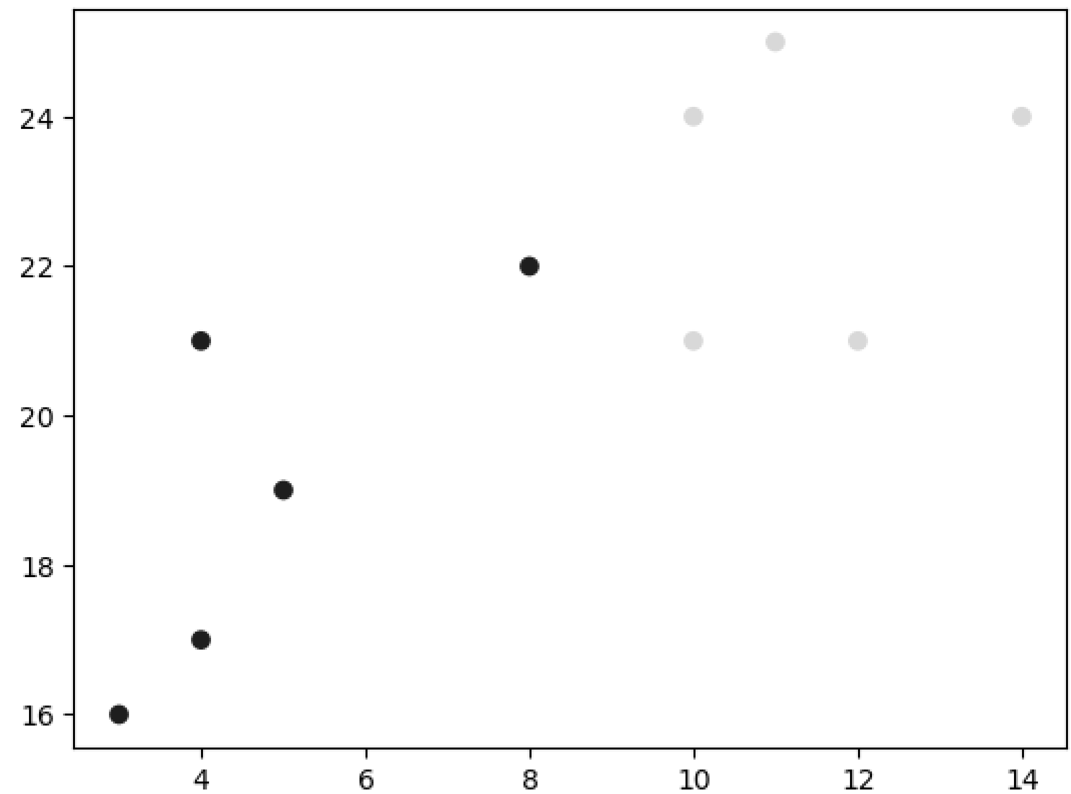
```

```

K-nearest neighbors (KNN)

KNN KNN is a simple, supervised machine learning (ML) algorithm that can be used for classification or regression tasks - and is also frequently used in missing value imputation. It is based on the idea that the observations closest to a given data point are the most "similar" observations in a data set, and we can therefore classify unforeseen points based on the values of the closest existing points. By choosing K, the user can select the number of nearby observations to use in the algorithm

In [1]:

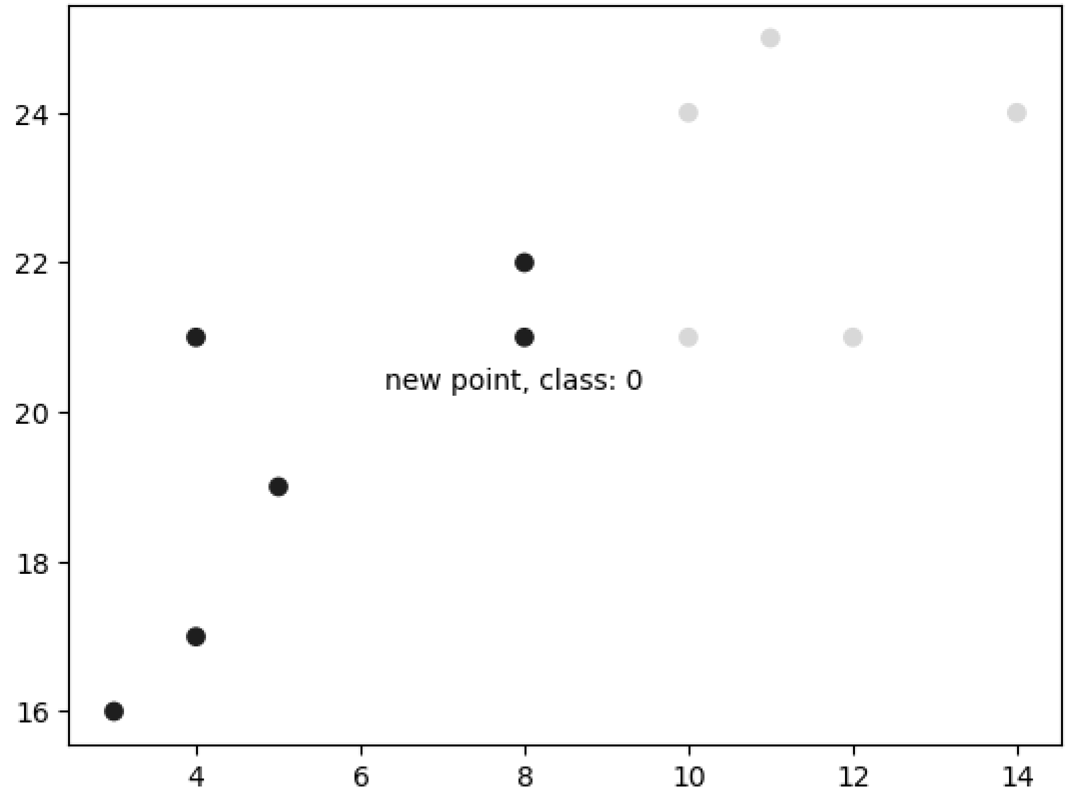
```
import matplotlib.pyplot as plt

x = [4, 5, 10, 4, 3, 11, 14 , 8, 10, 12]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]
classes = [0, 0, 1, 0, 0, 1, 1, 0, 1, 1]

plt.scatter(x, y, c=classes)
plt.show()
```



In [3]:

```
from sklearn.neighbors import KNeighborsClassifier

data = list(zip(x, y))
knn = KNeighborsClassifier(n_neighbors=1)

knn.fit(data, classes)
new_x = 8
new_y = 21
new_point = [(new_x, new_y)]

prediction = knn.predict(new_point)

plt.scatter(x + [new_x], y + [new_y], c=classes + [prediction[0]])
plt.text(x=new_x-1.7, y=new_y-0.7, s=f"new point, class: {prediction[0]}")
plt.show()
```

```
F:\Software\Data Science\AnacondInstallFile\lib\site-packages\sklearn\neig
hbors\_classification.py:228: FutureWarning: Unlike other reduction functi
ons (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically pr
eserves the axis it acts along. In SciPy 1.11.0, this behavior will chang
e: the default value of `keepdims` will become False, the `axis` over whic
h the statistic is taken will be eliminated, and the value None will no lo
nger be accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```



Monkey Learn Boilerplate Extractor

https://app.monkeylearn.com/ (https://app.monkeylearn.com/)

In [7]:

```
from monkeylearn import MonkeyLearn

ml = MonkeyLearn('f3d92a360e53c956569dffea87bec21f7faef824')
data = ["<!DOCTYPE html>\r\n<head><meta charset=\"UTF-8\"><\/head>\r\n<body>\r\n<h1>Ford
model_id = 'ex_RK5ApHnN'
result = ml.extractors.extract(model_id, data)
print(result.body)
```


```
[{'text': '<!DOCTYPE html>\r\n<head><meta charset="UTF-8"><\\/head>\r\n<bo
dy>\r\n<h1>Ford disguised a man as a car seat to research self-driving <
\\/h1>\r\n<p id="speakable-summary">Yes, you read that correctly: Ford put
a man in a car seat disguise so that a Ford Transit could masquerade as a
true self-driving vehicle. Why? To evaluate how passers-by, other drivers
on the road and cyclists reacted to sharing the road with an autonomous ve
hicle.<\\/p>\r\n<p>The trial, conducted with the Virginia Tech Transportat
ion Institute, also made use of a light bar mounted on the top of the wind
shield to provide communication about what the car was doing, including yi
elding, driving autonomously or accelerating from a full stop.\r\n<\\/body
>\r\n<\\/html>', 'external_id': None, 'error': False, 'extractions': [{'pa
rsed_value': 'Ford disguised a man as a car seat to research self-driving
<\\/h1>', 'tag_name': 'header'}, {'parsed_value': 'Yes, you read that corr
ectly: Ford put a man in a car seat disguise so that a Ford Transit could
masquerade as a true self-driving vehicle. Why? To evaluate how passers-b
y, other drivers on the road and cyclists reacted to sharing the road with
an autonomous vehicle.<\\/p>', 'tag_name': 'paragraph'}, {'parsed_value':
'The trial, conducted with the Virginia Tech Transportation Institute, als
o made use of a light bar mounted on the top of the windshield to provide
communication about what the car was doing, including yielding, driving au
tonomously or accelerating from a full stop.\n<\\/body>\n<\\/html>', 'tag_
name': 'paragraph'}]}]
```


Face completion with a multi-output estimators

This example shows the use of multi-output estimator to complete images. The goal is to predict the lower half of a face given its upper half.

The first column of images shows true faces. The next columns illustrate how extremely randomized trees, k nearest neighbors, linear regression and ridge regression complete the lower half of those faces.

In [2]:

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

from sklearn.datasets import fetch_olivetti_faces
from sklearn.utils.validation import check_random_state

from sklearn.ensemble import ExtraTreesRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import RidgeCV

# Load the faces datasets
data, targets = fetch_olivetti_faces(return_X_y=True)

train = data[targets < 30]
test = data[targets >= 30]  # Test on independent people

# Test on a subset of people
n_faces = 5
rng = check_random_state(4)
face_ids = rng.randint(test.shape[0], size=(n_faces,))
test = test[face_ids, :]

n_pixels = data.shape[1]
# Upper half of the faces
X_train = train[:, : (n_pixels + 1) // 2]
# Lower half of the faces
y_train = train[:, n_pixels // 2 :]
X_test = test[:, : (n_pixels + 1) // 2]
y_test = test[:, n_pixels // 2 :]

# Fit estimators
ESTIMATORS = {
    "Extra trees": ExtraTreesRegressor(
        n_estimators=10, max_features=32, random_state=0
    ),
    "K-nn": KNeighborsRegressor(),
    "Linear regression": LinearRegression(),
    "Ridge": RidgeCV(),
}

y_test_predict = dict()
for name, estimator in ESTIMATORS.items():
    estimator.fit(X_train, y_train)
    y_test_predict[name] = estimator.predict(X_test)

# Plot the completed faces
image_shape = (64, 64)

n_cols = 1 + len(ESTIMATORS)
plt.figure(figsize=(2.0 * n_cols, 2.26 * n_faces))
plt.suptitle("Face completion with multi-output estimators", size=16)

for i in range(n_faces):
    true_face = np.hstack((X_test[i], y_test[i]))

    if i:
        sub = plt.subplot(n_faces, n_cols, i * n_cols + 1)
    else:
```

```
        sub = plt.subplot(n_faces, n_cols, i * n_cols + 1, title="true faces")

    sub.axis("off")
    sub.imshow(
        true_face.reshape(image_shape), cmap=plt.cm.gray, interpolation="nearest"
    )

    for j, est in enumerate(sorted(ESTIMATORS)):
        completed_face = np.hstack((X_test[i], y_test_predict[est][i]))

        if i:
            sub = plt.subplot(n_faces, n_cols, i * n_cols + 2 + j)

        else:
            sub = plt.subplot(n_faces, n_cols, i * n_cols + 2 + j, title=est)

        sub.axis("off")
        sub.imshow(
            completed_face.reshape(image_shape),
            cmap=plt.cm.gray,
            interpolation="nearest",
        )

plt.show()
```

Face completion with multi-output estimators
true faces
Extra trees
K-nn
Linear regression
Ridge


```
In [ ]:
```

```
In [ ]:
```