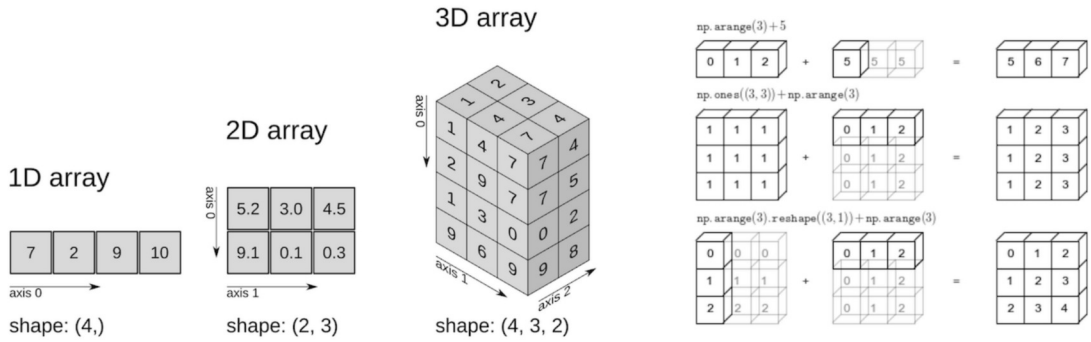
Numpy Library use for Array



In [1]:
```
import numpy as np
arr=np.array([10,20,30,40,50])
arr
```

Out[1]:
```
array([10, 20, 30, 40, 50])
```

In [2]:
```
type(arr)
```

Out[2]:
```
numpy.ndarray
```

In [3]:
```
arr2=np.array((10,20,30,40,50))
arr2
```

Out[3]:
```
array([10, 20, 30, 40, 50])
```

In [4]:
```
type(arr2)
```

Out[4]:
```
numpy.ndarray
```

Two Dimensional Array

In [5]:
```
tarr=np.array([[10,20,30],[40,50,60]])
print(tarr)
print(type(tarr))
```

```
[[10 20 30]
 [40 50 60]]
<class 'numpy.ndarray'>
```

3D Array

In [6]:
```
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])
print(arr)
```

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

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

Array Dimensions

In [7]:
```
a = np.array(42)
b = np.array([1, 2, 3, 4, 5])
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])
print(a.ndim)
print(b.ndim)
print(c.ndim)
print(d.ndim)
```

```
0
1
2
3
```

In [8]:
```
arr = np.array([1, 2, 3, 4,5,6,7,8], ndmin=2)
print(arr)
print('number of dimensions :', arr.ndim)
```

```
[[1 2 3 4 5 6 7 8]]
number of dimensions : 2
```

In [9]:
```
#List Slicing using index
lst=[10,20,30,40,50]
print(lst[-1])
print(lst[-5:-1])
```

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

NumPy Array Slicing

In [10]:
```
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[1:5])
```

```
[2 3 4 5]
```

In [11]:
```
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[-3:-1])
```

```
[5 6]
```

In [12]:
```
print(arr[-3:])
```

```
[5 6 7]
```

Slicing 2-D Arrays

In [13]:
```
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print(arr[1, 1:4])
print(arr[1, :])
```

```
[7 8 9]
[ 6  7  8  9 10]
```

In [16]:
```
print(arr[0:2])
print("----------------")
print(arr[0:2,2])
```

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

NumPy Data Types

i - integer • b - boolean • u - unsigned integer • f - float • c - complex float • m - timedelta • M - datetime • O - object • S - string • U - unicode string • V - fixed chunk of memory for other type (void)

In [21]:
```
import numpy as np
arr = np.array([1, 2, 3, 4])
print(arr.dtype)
```

```
int32
```

In [24]:
```
arr2 = np.array([1, 2, 3, 4],dtype='S')
print(arr2.dtype)
arr2 = np.array([1, 2, 3, 4],dtype='f')
print(arr2.dtype)
```

```
|S1
float32
```

Convert Float to integer

In [26]:
```
import numpy as np
arr = np.array([1.1, 2.1, 3.1])
newarr = arr.astype('i')
print(newarr)
print(newarr.dtype)
```

```
[1 2 3]
int32
```

NumPy Array Copy vs View

The copy owns the data and any changes made to the copy will not affect original array, and any changes made to the original array will not affect the copy.

The view does not own the data and any changes made to the view will affect the original array, and any changes made to the original array will affect the view

In [27]:
```
# COPY:
# Make a copy, change the original array, and display both arrays:
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
x = arr.copy()
arr[0] = 42
print(arr)
print(x)
```

```
[42  2  3  4  5]
[1 2 3 4 5]
```

In [30]:
```
# VIEW:
# Make a view, change the original array, and display both arrays:
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
x = arr.view()
arr[0] = 42
print(arr)
print(x)
```

```
[42  2  3  4  5]
[42  2  3  4  5]
```

Check if Array Owns its Data

Every NumPy array has the attribute base that returns None if the array owns the data.

In [36]:
```
print(x.base)
print(arr.base)
```

```
[42  2  3  4  5]
None
```

NumPy Array Shape

In [37]:
```
import numpy as np
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
print(arr.shape)
```

```
(2, 4)
```

In [40]:
```
import numpy as np
arr = np.array([1, 2, 3, 4], ndmin=5)
print(arr)
print('shape of array :', arr.shape)
```

```
[[[[[1 2 3 4]]]]]
shape of array : (1, 1, 1, 1, 4)
```

NumPy Array Reshaping

In [50]:
```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
newarr = arr.reshape(4, 3)
print(newarr)
```

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

In [48]:
```
newarr = arr.reshape(3,2,2)
print(newarr)
print(type(newarr))
print(newarr.ndim)
```

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

 [[ 5  6]
  [ 7  8]]

 [[ 9 10]
  [11 12]]]
<class 'numpy.ndarray'>
3
```

In [52]:
```
print(arr.reshape(4, 3).base)
```

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

In [54]:
```
import numpy as np
arr = np.array([[1, 2, 3], [4, 5, 6]])
newarr = arr.reshape(-1)
print(newarr)
```

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

In [55]:
```
arr = np.array([1, 2, 3])
for x in arr:
 print(x)
```

```
1
2
3
```

In [60]:
```
arr = np.array([[10, 20, 30], [40, 50, 60]])
for x in arr: # it is [10,20,30]
    for y in x:
        print(y,end=" ")
    print()
```

```
10 20 30
40 50 60
```

In [61]:
```
arr = np.array([[10, 20, 30], [40, 50, 60]])
for x in range(0,2):
    for y in range(0,3):
        print(arr[x][y],end=" ")
    print()
```

```
10 20 30
40 50 60
```

In [64]:
```
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
for x in arr:
    for y in x:
        for z in y:
            print(z,end=" ")
        print()
```

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

Iterating Arrays Using nditer()

The function nditer() is a helping function that can be used from very basic to very advanced iterations. It solves some basic issues which we face in iteration, lets go through it with examples.

In [1]:
```
import numpy as np
arr = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
for x in np.nditer(arr):
    print(x)
```

```
1
2
3
4
5
6
7
8
```

Transpose

In [3]:
```
arr = np.array([[1, 2], [6, 7]])
print(arr)
```

```
[[1 2]
 [6 7]]
```

In [6]:
```
#Tranpose using function
np.transpose(arr)
```

Out[6]:
```
array([[1, 6],
       [2, 7]])
```

In [14]:
```
#Transpose without function
newarr=np.empty([2, 2], dtype=int)
for i in range(0,2):
    for j in range(0,2):
        newarr[i][j]=arr[j][i]
print(newarr)
```

```
[[1 6]
 [2 7]]
```

NumPy Joining Array

In [32]:
```
import numpy as np
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
print(arr1+arr2)
print(np.concatenate((arr1, arr2)))
```

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

Joining/ Concatenating Array

In [44]:
```
arr1 = np.array([[1, 2], [3, 4]])
arr2 = np.array([[5, 6], [7, 8]])
arr = np.concatenate((arr1, arr2), axis=1)
# print(arr1[0],arr1[1],arr1[0][0])

arr = np.concatenate((arr1[0],arr1[1],arr2[0],arr2[1]))
print(arr)
```

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

Concatenating Array using Stack

you can set axis according to your matrix dimension

In [51]:
```
arr1 = np.array([[1, 2, 3],[11,22,33]])
arr2 = np.array([[4, 5, 6],[11,22,33]])
arr3 = np.array([[7, 8, 9],[11,22,33]])
arr = np.stack((arr1, arr2,arr3),axis=1)
print(arr)
```

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

 [[11 22 33]
  [11 22 33]
  [11 22 33]]]
```

In [67]:
```
arr = np.stack((arr1, arr2,arr3), axis=2)
print(arr)
```

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

 [[11 11 11]
  [22 22 22]
  [33 33 33]]]
```

Splitting NumPy Arrays

Splitting is reverse operation of Joining. Joining merges multiple arrays into one and Splitting breaks one array into multiple. We use array_split() for splitting arrays, we pass it the array we want to split and the number of splits.

In [72]:
```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6])
newarr = np.array_split(arr, 2)
print(newarr)
```

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

In [73]:
```
print(newarr[0])
```

```
[1 2 3]
```

In [74]:
```
arr = np.array([[1, 2], [3, 4], [5, 6], [7, 8], [9, 10], [11, 12]])
newarr = np.array_split(arr, 3)
print(newarr)
```

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

In [77]:
```
print(newarr[2])
```

```
[[ 9 10]
 [11 12]]
```

In [78]:
```
# newarr = np.array_split(arr, 3, axis=1)
```

NumPy Searching Arrays

In [1]:
```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 4, 4])
x = np.where(arr == 4)
print(x)
#it will show the index number of searching values
```

```
(array([3, 5, 6], dtype=int64),)
```

In [3]:
```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
x = np.where(arr%2 == 0)
print(x)
```

```
(array([1, 3, 5, 7], dtype=int64),)
```

Search Sorted

There is a method called searchsorted() which performs a binary search in the array, and returns the index where the specified value would be inserted to maintain the search order. The searchsorted() method is assumed to be used on sorted arrays

In [19]:
```
import numpy as np
arr = np.array([10, 7, 8, 5, 9])
x = np.searchsorted(arr, 8)
print(x)
```

```
2
```

Search From the Right Side

In [20]:
```
arr = np.array([6, 7, 8, 9])
x = np.searchsorted(arr, 7, side='right')
print(x)
```

```
2
```

Multiple Values

To search for more than one value, use an array with the specified values. Example Find the indexes where the values 2, 4, and 6 should be inserted:

In [21]:
```
import numpy as np
arr = np.array([1, 3, 5, 7])
x = np.searchsorted(arr, [2, 4, 6])
print(x)
```

```
[1 2 3]
```

In [26]:
```
import numpy as np
arr = np.array([3, 5, 7,10,30])
x = np.searchsorted(arr, [2, 4, 6])
print(x)
```

```
[0 1 2]
```

Sorting Arrays

Sorting means putting elements in an ordered sequence. Ordered sequence is any sequence that has an order corresponding to elements, like numeric or alphabetical, ascending or descending.

In [27]:
```
import numpy as np
arr = np.array([3, 2, 0, 1])
print(np.sort(arr))
```

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

In [35]:
```
arr = np.array([3, 2, 0, 1])
print(np.sort(arr))
print(np.sort(arr)[::-1])
```

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

Sorting a 2-D Array

In [37]:
```
import numpy as np
arr = np.array([[3, 2, 4], [5, 0, 1]])
print(np.sort(arr))
```

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

NumPy Filter Array

Filtering Arrays Getting some elements out of an existing array and creating a new array out of them is called filtering. In NumPy, you filter an array using a boolean index list. A boolean index list is a list of booleans corresponding to index

In [38]:
```
import numpy as np
arr = np.array([41, 42, 43, 44])
x = [True, False, True, False]
newarr = arr[x]
print(newarr)
```

```
[41 43]
```

Statistics

In [18]:
```
# Mean
import numpy as np
import statistics as st
data=np.array([182,161,152,137,121])
mn=st.mean(data)
print("Mean is ",mn)
print("Mode is ",st.mode(data))
print("Middle is ",st.median(data))
print("Population Variance is ",st.pvariance(data))
print("Sample Variance is ",st.variance(data))
print("Population Standard Deviation ",st.pstdev(data))
print("Sample Standard Deviation ",st.stdev(data))

#print with specifice Decimal
print("Mean is %0.02f"%st.pstdev(data))
```

```
Mean is  150
Mode is  182
Middle is  152
Population Variance is  431
Sample Variance is  539
Population Standard Deviation  20.760539492026695
Sample Standard Deviation  23.2163735324878
Mean is 20.76
```

In [ ]:
```

```

Pandas Library use for data Manipulation, Like Data Cleaning

Statistic using Pandas

In [17]:
```
import numpy as np
import pandas as pd
data=np.array([182,161,152,137,121])
print("Mean ",data.mean())
print("variance ",data.var())
print("Standard Deviation ",data.std())
```

```
Mean  150.6
variance  431.43999999999994
Standard Deviation  20.771133815947554
```

Convert your Matrix data in Table using with DataFrame

In [25]:
```
import numpy as np
import pandas as pd

#matx=np.array([[[1,2,3],[4,5,6]],[[11,22,33],[44,55,66]]])
matx=np.array([[1,2,3],[4,5,6],[7,8,9]])
matx=pd.DataFrame(matx)
matx
```

Out[25]:

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9

DataFrame

In [27]:
```
import pandas
mydataset = {
 'cars': ["BMW", "Volvo", "Ford"],
 'passings': [3, 7, 2]
}
myindex=[["One", "Two", "Three"]]
myvar = pandas.DataFrame(mydataset,index = myindex)
```

```
        cars  passings
One      BMW         3
Two    Volvo         7
Three   Ford         2
```

What is a Series?

A Pandas Series is like a column in a table.

In [19]:
```
import numpy as np
mydata2=[11,22,33]
myvar2 = pandas.Series(mydata2)
print(myvar2)
```

```
0    11
1    22
2    33
dtype: int64
```

In [21]:
```
myvarr = pd.Series(mydata2, index = ["x", "y", "z"])
print(myvarr)
```

```
x    11
y    22
z    33
dtype: int64
```

In [22]:
```
import pandas as pd
calories = {"day1": 420, "day2": 380, "day3": 390}
myvar = pd.Series(calories)
myvar
```

Out[22]:
```
day1    420
day2    380
day3    390
dtype: int64
```

Index line of Code () Locate Row As you can see from the result above, the DataFrame is like a table with rows and columns. Pandas use the loc attribute to return one or more specified row

In [ ]:
```
# import pandas
mydataset = {
 'cars': ["BMW", "Volvo", "Ford"],
 'passings': [3, 7, 2],
  'price':[100,200,300]
}
myvar = pandas.DataFrame(mydataset)
print(myvar.loc[1:3])
```

In [48]:
```
myvar = pandas.DataFrame(mydataset)
print(myvar.loc[[0,1]])
```

```
    cars  passings  price
0    BMW         3    100
1  Volvo         7    200
```

Read CSV file Using Pandas

In [49]:
```
import pandas as pd

rdcsv=pd.read_csv('CSVFileForPanda.csv')
rdcsv
```

Out[49]:

	empcode	First Name	Last Name	Dept	Region	Branch	Hiredate	Salary
0	5	Seema	Ranganathan	R&D	north	Kanpur	NaN	89000
1	84	Raja	Raymondekar	Sales	north	Ferozepur	1/1/1977	30000
2	39	Sheetal	Dodhia	Finance	north	Delhi	1/1/1977	24500
3	38	Chitra	Pednekar	Finance	north	Aligarh	10/1/1982	24500
4	90	Priyanka	Mehta	R&D	north	Jaipur	1/6/1980	22750
5	450	Tejal	Patel	Sales	South	Aligarh	2/21/1979	22750
6	67	Uday	Naik	Personnel	South	Lucknow	10/28/1988	20125
7	12	Shilpa	Lele	Admin	South	Jammu	3/1/1983	20000
8	62	Asha	Trivedi	Sales	South	Kanpur	11/26/1987	19250
9	61	Anuradha	Zha	Admin	South	Agra	11/25/1987	19250
10	52	Heena	Godbole	CCD	South	Lucknow	10/21/1988	19250
11	86	Sonia	Sasan	CCD	South	Jammu	1/15/1998	17500
12	80	Dayanand	Gandhi	Mktg	South	Ferozepur	10/30/1988	17500
13	79	Sagar	Bidkar	Mktg	South	Delhi	10/29/1988	17500
14	77	Drishti	Shah	R&D	West	Delhi	10/27/1988	17500
15	73	Laveena	Shenoy	CCD	West	Jaipur	10/23/1988	17500
16	66	Parul	Shah	Personnel	West	Agra	10/27/1988	17500
17	63	Waheda	Sheikh	R&D	West	Jammu	10/24/1988	17500
18	58	Maya	Panchal	Mktg	West	Agra	11/5/1990	17500
19	98	Chetan	Dalvi	CCD	West	Delhi	4/11/1989	17325
20	83	Kunal	Shah	CCD	West	Aligarh	3/2/1999	3500
21	33	Tapan	Ghoshal	CCD	West	Ambala	7/7/1997	4000
22	85	Ruby	Joseph	R&D	West	Agra	1/14/1998	7000
23	34	Zarina	Vora	CCD	West	Lucknow	9/29/1991	8750
24	29	Hajra	Hoonjan	Admin	West	Jaipur	5/4/1996	9625
25	19	Parvati	Khanna	Mktg	West	Mathura	8/13/1986	10500
26	45	Kinnari	Mehta	R&D	East	Ferozepur	7/7/1997	11375
27	1	Raja	Sadiq	Mktg	East	PUNE	6/5/1999	12000
28	4	Beena	Mavadia	Admin	East	Delhi	9/23/2014	12250
29	6	Julie	D'Souza	R&D	East	Mathura	9/4/1988	12425
30	8	Neena	Mukherjee	R&D	East	Agra	9/4/1989	12425
31	68	Mandakini	Desai	Sales	East	Delhi	12/1/1995	14000
32	69	Pravin	Joshi	Personnel	East	Delhi	3/3/1995	14000
33	9	Pankaj	Sutradhar	Sales	East	Ambala	12/12/1999	14875
34	15	K.Sita	Narayanan	Personnel	East	Jammu	12/13/1984	14875
35	16	Priya	Shirodkar	Personnel	East	Jaipur	12/14/1984	14875
36	31	Giriraj	Gupta	R&D	East	Agra	10/1/1982	15750
37	46	Jeena	Baig	Sales	East	Lucknow	9/29/1991	15750
38	47	Vicky	Joshi	Admin	East	Kanpur	2/7/1988	15750

	empcode	First Name	Last Name	Dept	Region	Branch	Hiredate	Salary
39	51	Mario	Fernandes	Sales	East	Jammu	10/20/1988	15750

In [52]:
```
print(rdcsv.to_string())
```

	empcode	First Name	Last Name	Dept	Region	Branch	Hireda
te	Salary						
0	5	Seema	Ranganathan	R&D	north	Kanpur	N
aN	89000						
1	84	Raja	Raymondekar	Sales	north	Ferozepur	1/1/19
77	30000						
2	39	Sheetal	Dodhia	Finance	north	Delhi	1/1/19
77	24500						
3	38	Chitra	Pednekar	Finance	north	Aligarh	10/1/19
82	24500						
4	90	Priyanka	Mehta	R&D	north	Jaipur	1/6/19
80	22750						
5	450	Tejal	Patel	Sales	South	Aligarh	2/21/19
79	22750						
6	67	Uday	Naik	Personnel	South	Lucknow	10/28/19
88	20125						
7	12	Shilpa	Lele	Admin	South	Jammu	3/1/19
83	20000						
8	62	Asha	Trivedi	Sales	South	Kanpur	11/26/19
87	19250						
9	61	Anuradha	Zha	Admin	South	Agra	11/25/19
87	19250						
10	52	Heena	Godbole	CCD	South	Lucknow	10/21/19
88	19250						
11	86	Sonia	Sasan	CCD	South	Jammu	1/15/19
98	17500						
12	80	Dayanand	Gandhi	Mktg	South	Ferozepur	10/30/19
88	17500						
13	79	Sagar	Bidkar	Mktg	South	Delhi	10/29/19
88	17500						
14	77	Drishti	Shah	R&D	West	Delhi	10/27/19
88	17500						
15	73	Laveena	Shenoy	CCD	West	Jaipur	10/23/19
88	17500						
16	66	Parul	Shah	Personnel	West	Agra	10/27/19
88	17500						
17	63	Waheda	Sheikh	R&D	West	Jammu	10/24/19
88	17500						
18	58	Maya	Panchal	Mktg	West	Agra	11/5/19
90	17500						
19	98	Chetan	Dalvi	CCD	West	Delhi	4/11/19
89	17325						
20	83	Kunal	Shah	CCD	West	Aligarh	3/2/19
99	3500						
21	33	Tapan	Ghoshal	CCD	West	Ambala	7/7/19
97	4000						
22	85	Ruby	Joseph	R&D	West	Agra	1/14/19
98	7000						
23	34	Zarina	Vora	CCD	West	Lucknow	9/29/19
91	8750						
24	29	Hajra	Hoonjan	Admin	West	Jaipur	5/4/19
96	9625						
25	19	Parvati	Khanna	Mktg	West	Mathura	8/13/19
86	10500						
26	45	Kinnari	Mehta	R&D	East	Ferozepur	7/7/19
97	11375						
27	1	Raja	Sadiq	Mktg	East	PUNE	6/5/19
99	12000						
28	4	Beena	Mavadia	Admin	East	Delhi	9/23/20
14	12250						
29	6	Julie	D'Souza	R&D	East	Mathura	9/4/19

```
88   12425
30       8      Neena    Mukherjee        R&D   East       Agra    9/4/19
89   12425
31      68  Mandakini        Desai      Sales   East      Delhi   12/1/19
95   14000
32      69     Pravin        Joshi  Personnel   East      Delhi    3/3/19
95   14000
33       9     Pankaj    Sutradhar      Sales   East     Ambala  12/12/19
99   14875
34      15     K.Sita    Narayanan  Personnel   East      Jammu  12/13/19
84   14875
35      16      Priya    Shirodkar  Personnel   East     Jaipur  12/14/19
84   14875
36      31    Giriraj        Gupta        R&D   East       Agra   10/1/19
82   15750
37      46      Jeena         Baig      Sales   East    Lucknow   9/29/19
91   15750
38      47      Vicky        Joshi      Admin   East     Kanpur    2/7/19
88   15750
39      51      Mario    Fernandes      Sales   East      Jammu  10/20/19
88   15750
```

In [56]:
```
print(rdcsv.loc[5:10])
```

```
    empcode First Name Last Name       Dept Region   Branch    Hiredate  \
5       450      Tejal     Patel      Sales  South  Aligarh   2/21/1979
6        67       Uday      Naik  Personnel  South  Lucknow  10/28/1988
7        12     Shilpa      Lele      Admin  South    Jammu    3/1/1983
8        62       Asha   Trivedi      Sales  South   Kanpur  11/26/1987
9        61   Anuradha       Zha      Admin  South     Agra  11/25/1987
10       52      Heena   Godbole        CCD  South  Lucknow  10/21/1988

    Salary
5    22750
6    20125
7    20000
8    19250
9    19250
10   19250
```

In [59]:
```
print(rdcsv.head(7))
```

```
   empcode First Name    Last Name       Dept Region     Branch    Hiredat
e  \
0        5      Seema  Ranganathan        R&D  north     Kanpur         Na
N
1       84       Raja  Raymondekar      Sales  north  Ferozepur    1/1/197
7
2       39    Sheetal       Dodhia    Finance  north      Delhi    1/1/197
7
3       38     Chitra     Pednekar    Finance  north    Aligarh   10/1/198
2
4       90   Priyanka        Mehta        R&D  north     Jaipur    1/6/198
0
5      450      Tejal        Patel      Sales  South    Aligarh   2/21/197
9
6       67       Uday         Naik  Personnel  South    Lucknow  10/28/198
8

   Salary
0   89000
1   30000
2   24500
3   24500
4   22750
5   22750
6   20125
```

In [62]:
```
print(rdcsv.tail(10))
```

```
    empcode First Name  Last Name       Dept Region   Branch    Hiredate
\
30        8      Neena  Mukherjee        R&D   East     Agra    9/4/1989
31       68  Mandakini      Desai      Sales   East    Delhi   12/1/1995
32       69     Pravin      Joshi  Personnel   East    Delhi    3/3/1995
33        9     Pankaj  Sutradhar      Sales   East   Ambala  12/12/1999
34       15     K.Sita  Narayanan  Personnel   East    Jammu  12/13/1984
35       16      Priya  Shirodkar  Personnel   East   Jaipur  12/14/1984
36       31    Giriraj      Gupta        R&D   East     Agra   10/1/1982
37       46      Jeena       Baig      Sales   East  Lucknow   9/29/1991
38       47      Vicky      Joshi      Admin   East   Kanpur    2/7/1988
39       51      Mario  Fernandes      Sales   East    Jammu  10/20/1988

    Salary
30   12425
31   14000
32   14000
33   14875
34   14875
35   14875
36   15750
37   15750
38   15750
39   15750
```

In [67]:
```
# print(rdcsv["Salary"])
print("Average Salary is ",rdcsv["Salary"].mean())
```

```
Average Salary is  17704.375
```

In [69]:
```
print(rdcsv.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 40 entries, 0 to 39
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   empcode     40 non-null     int64 
 1   First Name  40 non-null     object
 2   Last Name   40 non-null     object
 3   Dept        40 non-null     object
 4   Region      40 non-null     object
 5   Branch      40 non-null     object
 6   Hiredate    39 non-null     object
 7   Salary      40 non-null     int64 
dtypes: int64(2), object(6)
memory usage: 2.6+ KB
None
```

Read JSON Data

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

rjson=pd.read_json("employeedata.json")
rjson
```

Out[2]:

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

In [5]:
```
print(rjson.to_string())
```

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

In [8]:
```
rjson2=pd.read_json("https://dummyjson.com/products/1")
rjson2
```

Out[8]:

	id	title	description	price	discountPercentage	rating	stock	brand	category	
0	1	iPhone 9	An apple mobile which is nothing like apple	549	12.96	4.69	94	Apple	smartphones	https
1	1	iPhone 9	An apple mobile which is nothing like apple	549	12.96	4.69	94	Apple	smartphones	https
2	1	iPhone 9	An apple mobile which is nothing like apple	549	12.96	4.69	94	Apple	smartphones	https
3	1	iPhone 9	An apple mobile which is nothing like apple	549	12.96	4.69	94	Apple	smartphones	https
4	1	iPhone 9	An apple mobile which is nothing like apple	549	12.96	4.69	94	Apple	smartphones	https

Read Excel File

Install xlrd package before the reading Excel File . pip install xlrd

In [39]:
```
import pandas as pd

myexcel=pd.read_excel("SampleExcelFile.xlsx")
myexcel
```

Out[39]:

	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

In [40]:
```
d=myexcel.iloc[0:5,[0,3]]
d
```

Out[40]:

	First Name	Branch
0	Parvati	Mathura
1	Hajra	Jaipur
2	Tapan	Ambala
3	Zarina	Lucknow
4	Maya	Agra

In [36]:
```
myfile=myexcel[["First Name","Branch"]]
myfile
```

Out[36]:

	First Name	Branch
0	Parvati	Mathura
1	Hajra	Jaipur
2	Tapan	Ambala
3	Zarina	Lucknow
4	Maya	Agra
5	Waheda	Jammu
6	Parul	Agra
7	Laveena	Jaipur
8	Drishti	Delhi
9	Kunal	Aligarh
10	Ruby	Agra
11	Chetan	Delhi

Pandas - Cleaning Empty Cells

Remove Empty Cells using dropna

In [54]:
```
import pandas as pd

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
```

In [55]:
```
df2=df.dropna()
df2
```

Out[55]:

	Duration	Pulse	Maxpulse	Calories
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
...	...	...	...	...
165	60	105.0	140	290.8
166	60	110.0	145	300.0
167	60	115.0	145	310.2
168	75	120.0	150	320.4
169	75	125.0	150	330.4

164 rows × 4 columns

Note: By default, the dropna() method returns a new DataFrame, and will

not change the original. If you want to change the original DataFrame, use the inplace = True argument:

In [57]:
```
df.dropna(inplace=True)
df
```

Out[57]:

	Duration	Pulse	Maxpulse	Calories
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
...	...	...	...	...
165	60	105.0	140	290.8
166	60	110.0	145	300.0
167	60	115.0	145	310.2
168	75	120.0	150	320.4
169	75	125.0	150	330.4

164 rows × 4 columns

In [58]:
```
df
```

Out[58]:

	Duration	Pulse	Maxpulse	Calories
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
...	...	...	...	...
165	60	105.0	140	290.8
166	60	110.0	145	300.0
167	60	115.0	145	310.2
168	75	120.0	150	320.4
169	75	125.0	150	330.4

164 rows × 4 columns

Replace Empty Values

Another way of dealing with empty cells is to insert a new value instead. This way you do not have to delete entire rows just because of some empty cells. The fillna() method allows us to replace empty cells with a value:

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

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
```

In [76]:
```
df["Pulse"].fillna(1000,inplace=True)
```

In [2]:
```
df
```

Out[2]:

	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
...	...	...	...	...
165	60	105.0	140	290.8
166	60	110.0	145	300.0
167	60	115.0	145	310.2
168	75	120.0	150	320.4
169	75	125.0	150	330.4

170 rows × 4 columns

Replace Using Mean, Median, or Mode

In [4]:
```
import pandas as pd
df = pd.read_csv('data.csv')
x = df["Calories"].mean()
df["Calories"].fillna(x, inplace = True)
df
```

Out[4]:

	Duration	Pulse	Maxpulse	Calories
0	411	NaN	95	375.790244
1	60	110.0	130	409.100000
2	60	117.0	145	479.000000
3	60	103.0	135	340.000000
4	45	109.0	175	282.400000
...	...	...	...	...
165	60	105.0	140	290.800000
166	60	110.0	145	300.000000
167	60	115.0	145	310.200000
168	75	120.0	150	320.400000
169	75	125.0	150	330.400000

170 rows × 4 columns

In [9]:
```
import pandas as pd
dff = pd.read_csv('data.csv')
dff
```

Out[9]:

	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
...	...	...	...	...
165	60	105.0	140	290.8
166	60	110.0	145	300.0
167	60	115.0	145	310.2
168	75	120.0	150	320.4
169	75	125.0	150	330.4

170 rows × 4 columns

In [12]:
```
x = df["Calories"].median()
x
```

Out[12]:
```
318.6
```

In [15]:
```
x = df["Calories"].mode()[0]
x
```

Out[15]:
```
300.0
```

In [26]:
```
import pandas as pd
df=pd.read_csv("CSVFileForPanda.csv")
df
```

Out[26]:

	empcode	First Name	Last Name	Dept	Region	Branch	Hiredate	Salary
0	5	Seema	Ranganathan	R&D	north	Kanpur	NaN	89000
1	84	Raja	Raymondekar	Sales	north	Ferozepur	1/1/1977	30000
2	39	Sheetal	Dodhia	Finance	north	Delhi	1/1/1977	24500
3	38	Chitra	Pednekar	Finance	north	Aligarh	10/1/1982	24500
4	90	Priyanka	Mehta	R&D	north	Jaipur	1/6/1980	22750
5	450	Tejal	Patel	Sales	South	Aligarh	2/21/1979	22750
6	67	Uday	Naik	Personnel	South	Lucknow	10/28/1988	20125
7	12	Shilpa	Lele	Admin	South	Jammu	3/1/1983	20000
8	62	Asha	Trivedi	Sales	South	Kanpur	11/26/1987	19250
9	61	Anuradha	Zha	Admin	South	Agra	11/25/1987	19250
10	52	Heena	Godbole	CCD	South	Lucknow	10/21/1988	19250
11	86	Sonia	Sasan	CCD	South	Jammu	1/15/1998	17500
12	80	Dayanand	Gandhi	Mktg	South	Ferozepur	10/30/1988	17500
13	79	Sagar	Bidkar	Mktg	South	Delhi	10/29/1988	17500
14	77	Drishti	Shah	R&D	West	Delhi	10/27/1988	17500
15	73	Laveena	Shenoy	CCD	West	Jaipur	10/23/1988	17500
16	66	Parul	Shah	Personnel	West	Agra	10/27/1988	17500
17	63	Waheda	Sheikh	R&D	West	Jammu	10/24/1988	17500
18	58	Maya	Panchal	Mktg	West	Agra	11/5/1990	17500
19	98	Chetan	Dalvi	CCD	West	Delhi	4/11/1989	17325
20	83	Kunal	Shah	CCD	West	Aligarh	3/2/1999	3500
21	33	Tapan	Ghoshal	CCD	West	Ambala	7/7/1997	4000
22	85	Ruby	Joseph	R&D	West	Agra	1/14/1998	7000
23	34	Zarina	Vora	CCD	West	Lucknow	9/29/1991	8750
24	29	Hajra	Hoonjan	Admin	West	Jaipur	5/4/1996	9625
25	19	Parvati	Khanna	Mktg	West	Mathura	8/13/1986	10500
26	45	Kinnari	Mehta	R&D	East	Ferozepur	7/7/1997	11375
27	1	Raja	Sadiq	Mktg	East	PUNE	6/5/1999	12000
28	4	Beena	Mavadia	Admin	East	Delhi	9/23/2014	12250
29	6	Julie	D'Souza	R&D	East	Mathura	9/4/1988	12425
30	8	Neena	Mukherjee	R&D	East	Agra	9/4/1989	12425
31	68	Mandakini	Desai	Sales	East	Delhi	12/1/1995	14000
32	69	Pravin	Joshi	Personnel	East	Delhi	3/3/1995	14000
33	9	Pankaj	Sutradhar	Sales	East	Ambala	12/12/1999	14875
34	15	K.Sita	Narayanan	Personnel	East	Jammu	12/13/1984	14875
35	16	Priya	Shirodkar	Personnel	East	Jaipur	12/14/1984	14875
36	31	Giriraj	Gupta	R&D	East	Agra	10/1/1982	15750
37	46	Jeena	Baig	Sales	East	Lucknow	9/29/1991	15750
38	47	Vicky	Joshi	Admin	East	Kanpur	2/7/1988	15750

	empcode	First Name	Last Name	Dept	Region	Branch	Hiredate	Salary
39	51	Mario	Fernandes	Sales	East	Jammu	10/20/1988	15750

In [39]:
```
dff=df["Hiredate"].dropna()
dff
```

Out[39]:
```
1    1977-01-01
2    1977-01-01
3    1982-10-01
4    1980-01-06
5    1979-02-21
6    1988-10-28
7    1983-03-01
8    1987-11-26
9    1987-11-25
10   1988-10-21
11   1998-01-15
12   1988-10-30
13   1988-10-29
14   1988-10-27
15   1988-10-23
16   1988-10-27
17   1988-10-24
18   1990-11-05
19   1989-04-11
20   1999-03-02
21   1997-07-07
22   1998-01-14
23   1991-09-29
24   1996-05-04
25   1986-08-13
26   1997-07-07
27   1999-06-05
28   2014-09-23
29   1988-09-04
30   1989-09-04
31   1995-12-01
32   1995-03-03
33   1999-12-12
34   1984-12-13
35   1984-12-14
36   1982-10-01
37   1991-09-29
38   1988-02-07
39   1988-10-20
Name: Hiredate, dtype: datetime64[ns]
```

In [46]:
```
dt=pd.to_datetime(dff)
dt
```

Out[46]:
```
1    1977-01-01
2    1977-01-01
3    1982-10-01
4    1980-01-06
5    1979-02-21
6    1988-10-28
7    1983-03-01
8    1987-11-26
9    1987-11-25
10   1988-10-21
11   1998-01-15
12   1988-10-30
13   1988-10-29
14   1988-10-27
15   1988-10-23
16   1988-10-27
17   1988-10-24
18   1990-11-05
19   1989-04-11
20   1999-03-02
21   1997-07-07
22   1998-01-14
23   1991-09-29
24   1996-05-04
25   1986-08-13
26   1997-07-07
27   1999-06-05
28   2014-09-23
29   1988-09-04
30   1989-09-04
31   1995-12-01
32   1995-03-03
33   1999-12-12
34   1984-12-13
35   1984-12-14
36   1982-10-01
37   1991-09-29
38   1988-02-07
39   1988-10-20
Name: Hiredate, dtype: datetime64[ns]
```

Second Method to convert data into date time

In [45]:
```
df["Hiredate"]=pd.to_datetime(df["Hiredate"])
df["Hiredate"]
```

Out[45]:
```
0           NaT
1    1977-01-01
2    1977-01-01
3    1982-10-01
4    1980-01-06
5    1979-02-21
6    1988-10-28
7    1983-03-01
8    1987-11-26
9    1987-11-25
10   1988-10-21
11   1998-01-15
12   1988-10-30
13   1988-10-29
14   1988-10-27
15   1988-10-23
16   1988-10-27
17   1988-10-24
18   1990-11-05
19   1989-04-11
20   1999-03-02
21   1997-07-07
22   1998-01-14
23   1991-09-29
24   1996-05-04
25   1986-08-13
26   1997-07-07
27   1999-06-05
28   2014-09-23
29   1988-09-04
30   1989-09-04
31   1995-12-01
32   1995-03-03
33   1999-12-12
34   1984-12-13
35   1984-12-14
36   1982-10-01
37   1991-09-29
38   1988-02-07
39   1988-10-20
Name: Hiredate, dtype: datetime64[ns]
```

In [49]:
```
import pandas as pd
dff = pd.read_csv('data.csv')
dff.head(10)
```

Out[49]:

	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

In [54]:
```
dff.loc[4,'Duration'] = 450
dff
```

Out[54]:

	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	450	109.0	175	282.4
...	...	...	...	...
165	60	105.0	140	290.8
166	60	110.0	145	300.0
167	60	115.0	145	310.2
168	75	120.0	150	320.4
169	75	125.0	150	330.4

170 rows × 4 columns

Loop through all values in the "Duration" column.

If the value is higher than 120, set it to 120:

In [62]:
```
import pandas as pd
df = pd.read_csv('data.csv')
for x in df.index:
    #print(x)
    if df.loc[x, "Duration"] > 120:
        df.loc[x, "Duration"] = 200
print(df["Duration"].to_string())
```

```
0      200
1       60
2       60
3       60
4       45
5       45
6       60
7       60
8       45
9       30
10      60
11      60
12      60
13      60
14      60
15      60
16      60
17      60
18      45
```

Removing Rows

Another way of handling wrong data is to remove the rows that contains #remember to include the 'inplace = True' argument to make the changes in the original DataFrame object instead of returning a copy wrong data.

In [1]:
```
import pandas as pd
df = pd.read_csv('data.csv')
for x in df.index:
    if df.loc[x, "Duration"] > 100:
        df.drop(x, inplace = True)
print(df.to_string())
```

```
     Duration  Pulse  Maxpulse  Calories
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
18         45   90.0       112       NaN
```

Pandas - Removing

Duplicate Values

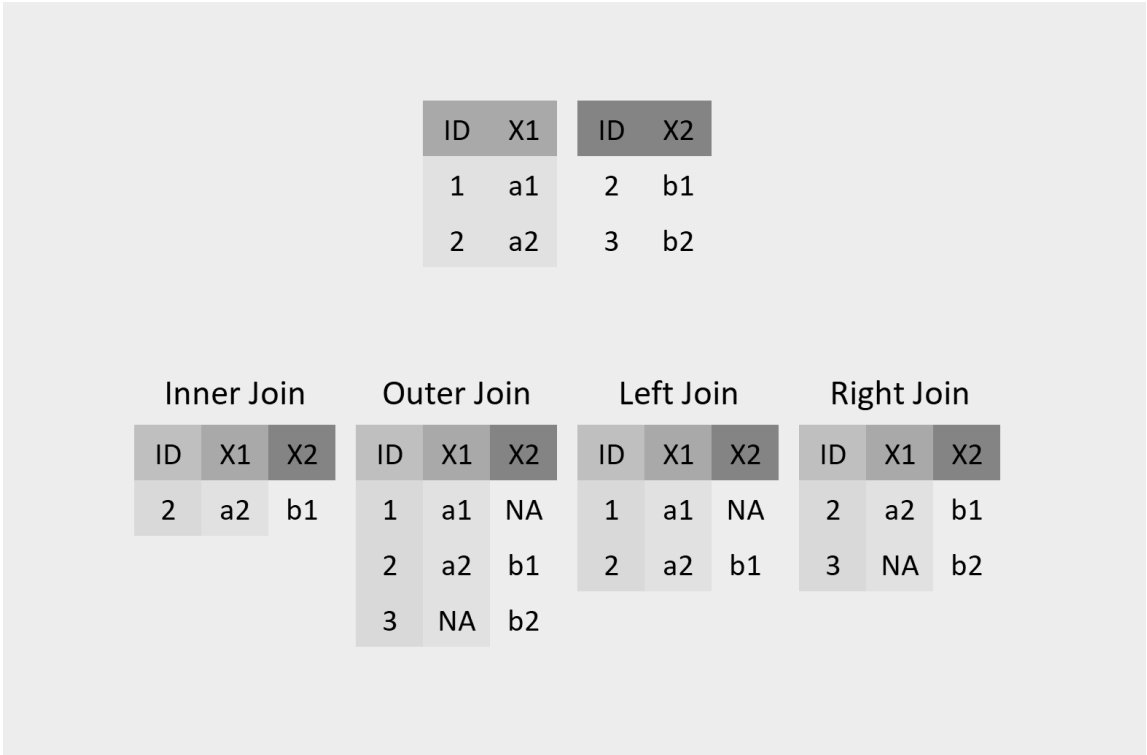
In [2]:
```
import pandas as pd
df = pd.read_csv('data.csv')
print(df.duplicated().to_string())
```

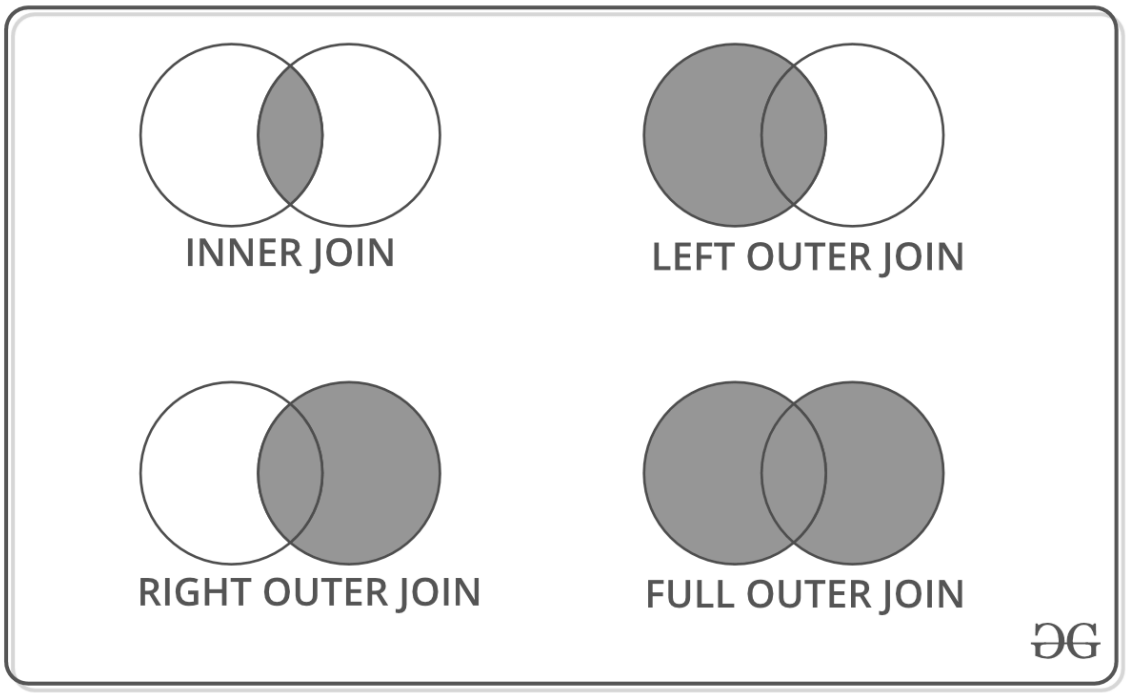
```
0      False
1      False
2      False
3      False
4      False
5      False
6      False
7      False
8      False
9      False
10     False
11     False
12     False
13     False
14     False
15     False
16     False
17     False
18     False
```

In [4]:
```
df.drop_duplicates(inplace = True)
df.dropna(inplace=True)
print(df.to_string())
```

```
     Duration  Pulse  Maxpulse  Calories
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
19         60  103.0       123     323.0
```

Merge two DataFrame





In [38]:
```
#Column Name should be same

import pandas as pd
df1=pd.DataFrame({'rkey':['foo','bar','baz','foo'],'value':[4,6,8,17]})
df2=pd.DataFrame({'rkey':['foo','bar','baz','foo'],'value':[4,6,7,8]})
print(df1)
print(df2)
```

```
  rkey  value
0  foo      4
1  bar      6
2  baz      8
3  foo     17
  rkey  value
0  foo      4
1  bar      6
2  baz      7
3  foo      8
```

In [39]:
```
#Inner Join
df11=df1.merge(df2)
df11
```

Out[39]:

	rkey	value
0	foo	4
1	bar	6

In [45]:
```
df1 = pd.DataFrame({'lkey': ['foo', 'bar', 'baz', 'foo'],
                    'value': [1, 2, 3, 5]})
df2 = pd.DataFrame({'rkey': ['foo', 'bar', 'baz', 'foo'],
                    'value': [5, 6, 7, 8]})
print(df1)
print(df2)
```

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

In [48]:
```
dd=df1.merge(df2, left_on='lkey', right_on='rkey')
dd
```

Out[48]:

	lkey	value_x	rkey	value_y
0	foo	1	foo	5
1	foo	1	foo	8
2	foo	5	foo	5
3	foo	5	foo	8
4	bar	2	bar	6
5	baz	3	baz	7

In [61]:
```
data1=pd.DataFrame({'city': ['Meerut', 'Lucknow'],'sales': [5000,6000]})
data2=pd.DataFrame({'city': ['Meerut', 'Kanpur'],'sales': [5000,5000]})
data=data1.merge(data2)
data
```

Out[61]:

	city	sales
0	Meerut	5000

In [62]:
```
mergedata=data1.merge(data2,left_on="city",right_on="city",how="left")
mergedata
```

Out[62]:

	city	sales_x	sales_y
0	Meerut	5000	5000.0
1	Lucknow	6000	NaN

In [66]:
```
msg=pd.concat([data1,data2])
msg
```

Out[66]:

	Mycity	sales	city
0	Meerut	5000	NaN
1	Lucknow	6000	NaN
0	NaN	5000	Meerut
1	NaN	5000	Kanpur

In [ ]:
```

```

Merge or Join Second Exercise

In [25]:
```
import numpy as np
import pandas as pd
data1 = {
  "name": ["Sally", "Mary", "John","Ram"],
  "age": [50, 40, 30,60]
}

data2 = {
  "name": ["Sally", "Peter", "Micky","Ram"],
  "age": [77, 44, 22,60]
}
d1=pd.DataFrame(data1)
d2=pd.DataFrame(data2)
print(d1)
print(d2)
```

```
    name  age
0  Sally   50
1   Mary   40
2   John   30
3    Ram   60
    name  age
0  Sally   77
1  Peter   44
2  Micky   22
3    Ram   60
```

In [30]:
```
#show Left table
leftable=d1.merge(d2,how="left")
leftable
```

Out[30]:

	name	age
0	Sally	50
1	Mary	40
2	John	30
3	Ram	60

In [31]:
```
#show Left table
leftable=d1.merge(d2,how="right")
leftable
```

Out[31]:

	name	age
0	Sally	77
1	Peter	44
2	Micky	22
3	Ram	60

In [28]:
```
#Left outer join
leftjoin=d1.merge(d2,left_on="name",right_on="name",how="left")
leftjoin
```

Out[28]:

	name	age_x	age_y
0	Sally	50	77.0
1	Mary	40	NaN
2	John	30	NaN
3	Ram	60	60.0

In [29]:
```
#Right outer join
leftjoin=d1.merge(d2,left_on="name",right_on="name",how="right")
leftjoin
```

Out[29]:

	name	age_x	age_y
0	Sally	50.0	77
1	Peter	NaN	44
2	Micky	NaN	22
3	Ram	60.0	60

In [33]:
```
#innerjoin
innerjoin=d1.merge(d2)
innerjoin
```

Out[33]:

	name	age
0	Ram	60

pandas.DataFrame.join

In [5]:
```
import pandas as pd
df= pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3', 'K4', 'K5'],
                  'A': ['A0', 'A1', 'A2', 'A3', 'A4', 'A5']})
other = pd.DataFrame({'key': ['K0', 'K1', 'K2'],
                      'B': ['B0', 'B1', 'B2']})
print(df)
print(other)
```

```
  key   A
0  K0  A0
1  K1  A1
2  K2  A2
3  K3  A3
4  K4  A4
5  K5  A5
  key   B
0  K0  B0
1  K1  B1
2  K2  B2
```

In [68]:
```
df.join(other, lsuffix=' ', rsuffix=' ')
```

Out[68]:

	key	A	key	B
0	K0	A0	K0	B0
1	K1	A1	K1	B1
2	K2	A2	K2	B2
3	K3	A3	NaN	NaN
4	K4	A4	NaN	NaN
5	K5	A5	NaN	NaN

In [20]:
```
df.join(other.set_index('key'), on='key')
```

Out[20]:

	key	A	B
0	K0	A0	B0
1	K1	A1	B1
2	K2	A2	B2
3	K3	A3	NaN
4	K4	A4	NaN
5	K5	A5	NaN

Pivot Return reshaped DataFrame organized by given index / column values.

Reshape data (produce a “pivot” table) based on column values. Uses unique values from specified index / columns to form axes of the resulting DataFrame. This function does not support data aggregation, multiple values will result in a MultiIndex in the columns. See the User Guide for more on reshaping.

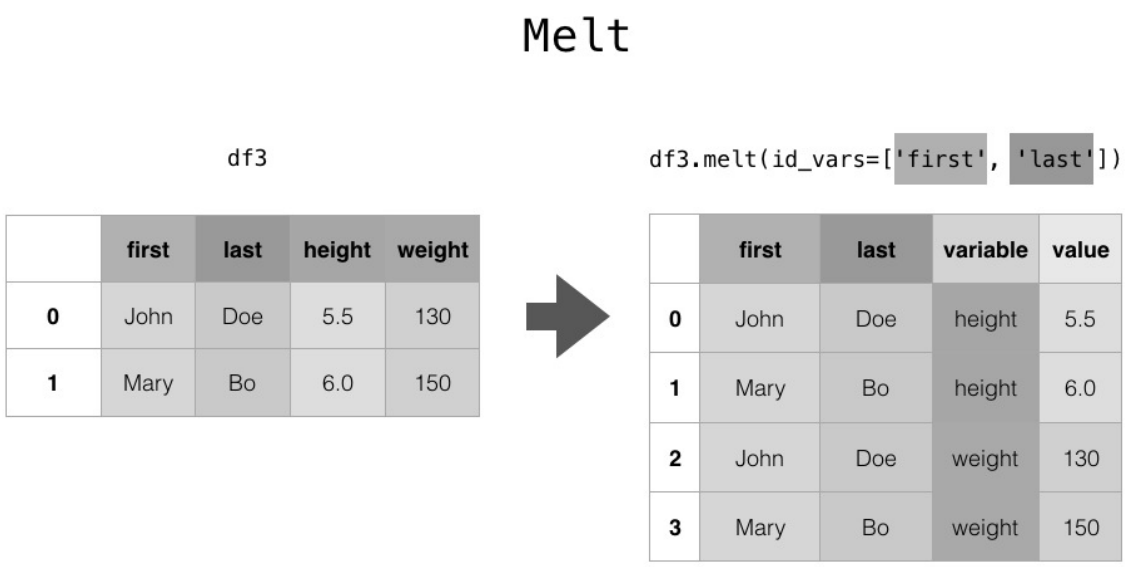
DataFrame.pivot(*, columns, index=typing.Literal[], values=typing.Literal[])[source]

In [15]:
```
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']})
df1=df.pivot(index='foo', columns='bar', values='baz')
df1
```

Out[15]:

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

Unpivot using melt



In [24]:
```
cheese = pd.DataFrame(
    {
        "first": ["John", "Mary"],
         "last": ["Doe", "Bo"],
        "height": [5.5, 6.0],
         "weight": [130, 150],
    }
)
cheese
```

Out[24]:

	first	last	height	weight
0	John	Doe	5.5	130
1	Mary	Bo	6.0	150

In [25]:
```
cheese.melt(id_vars=["first"])
```

Out[25]:

	first	variable	value
0	John	last	Doe
1	Mary	last	Bo
2	John	height	5.5
3	Mary	height	6.0
4	John	weight	130
5	Mary	weight	150

In [27]:
```
cheese.melt(id_vars=["first","last","height"])
```

Out[27]:

	first	last	height	variable	value
0	John	Doe	5.5	weight	130
1	Mary	Bo	6.0	weight	150

In [ ]:
```

```

MultiIndex, Stack and Unstack

Multiindex arranging your matrix data accoding to x,y cordinate stack transpose data from column to row unstack transpose data from row to column

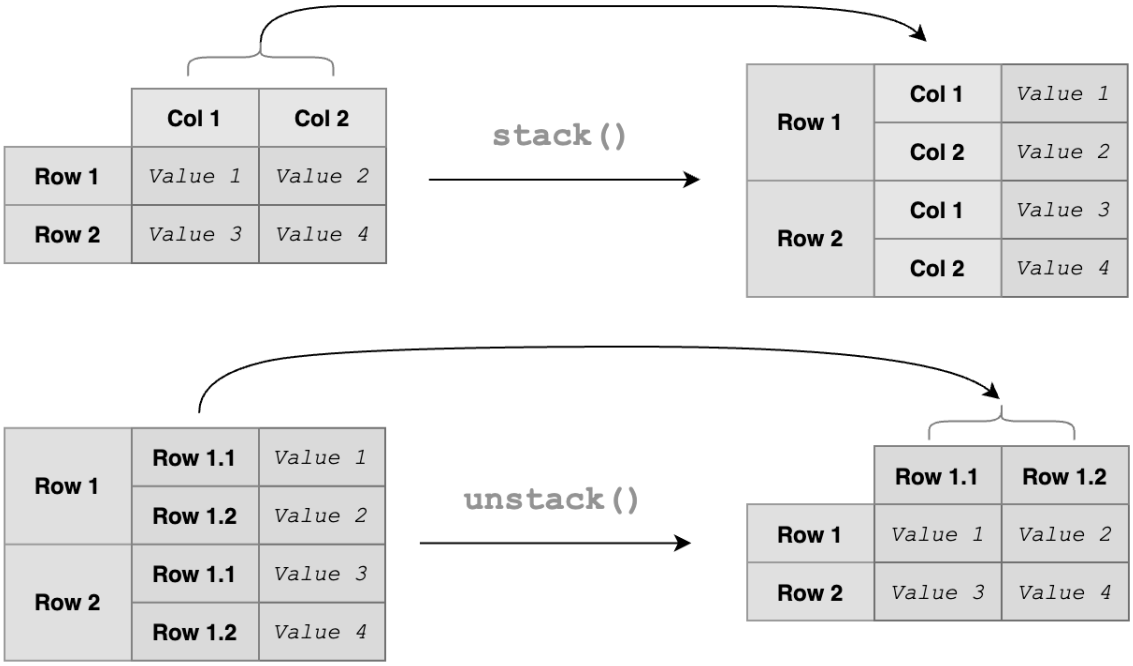
In [21]:
```
import pandas as pd
dfmulti=pd.DataFrame({'city':['Noida','Pune',"Delhi"],
                      'institue':['ABC','DEF','PQR']})
dfmulti
```

Out[21]:

	city	institue
0	Noida	ABC
1	Pune	DEF
2	Delhi	PQR

In [15]:
```
mdx=pd.MultiIndex(levels=[['Noida','Pune','Delhi'],
                          ['ABC','DEF','PQR']],
                          codes=[[0,1,0],[1,0,1]])
mdx
```

Out[15]:
```
MultiIndex([('Noida', 'DEF'),
            ( 'Pune', 'ABC'),
            ('Noida', 'DEF')],
           )
```



In [30]:
```
df_single_level_cols = pd.DataFrame([[0, 1], [2, 3]],
                                    index=['cat', 'dog'],
                                    columns=['weight', 'height'])
df_single_level_cols
```

Out[30]:

	weight	height
cat	0	1
dog	2	3

In [33]:
```
df_single_level_cols.stack()
```

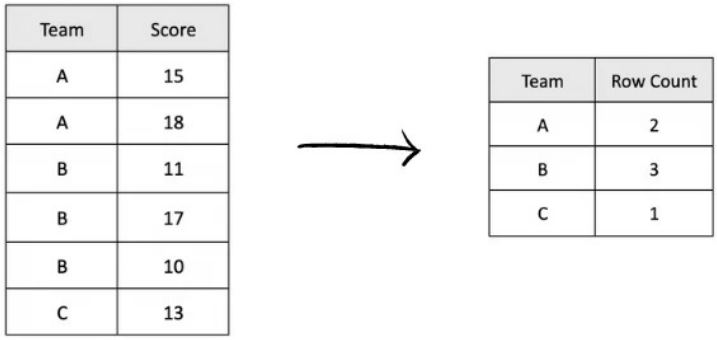
Out[33]:
```
cat  weight    0
     height    1
dog  weight    2
     height    3
dtype: int64
```

In [31]:
```
df_single_level_cols.unstack()
```

Out[31]:
```
weight  cat    0
        dog    2
height  cat    1
        dog    3
dtype: int64
```

A groupby operation involves some combination of splitting the object, applying a function, and combining the results. This can be used to group large amounts of data and compute operations on these groups.

pandas



Groupby Count Rows

In [34]:
```
df = pd.DataFrame({'Animal': ['Falcon', 'Falcon',
                              'Parrot', 'Parrot'],
                   'Max Speed': [380., 370., 24., 26.]})
df
```

Out[34]:

	Animal	Max Speed
0	Falcon	380.0
1	Falcon	370.0
2	Parrot	24.0
3	Parrot	26.0

In [36]:
```
df.groupby(['Animal']).count()
```

Out[36]:

Animal	Max Speed
Falcon	2
Parrot	2

In [37]:
```
df.groupby(['Animal']).sum()
```

Out[37]:

Animal	Max Speed
Falcon	750.0
Parrot	50.0

Sorting Values

In [39]:
```
#Ascending
df.sort_values(by='Max Speed', ascending=True)
```

Out[39]:

	Animal	Max Speed
2	Parrot	24.0
3	Parrot	26.0
1	Falcon	370.0
0	Falcon	380.0

In [42]:
```
#Descending
df.sort_values(by='Animal', ascending=True)
```

Out[42]:

	Animal	Max Speed
0	Falcon	380.0
1	Falcon	370.0
2	Parrot	24.0
3	Parrot	26.0

In [44]:
```
#Sorting with groupby
df.groupby(['Animal']).sum().sort_values(by='Max Speed', ascending=True)
```

Out[44]:

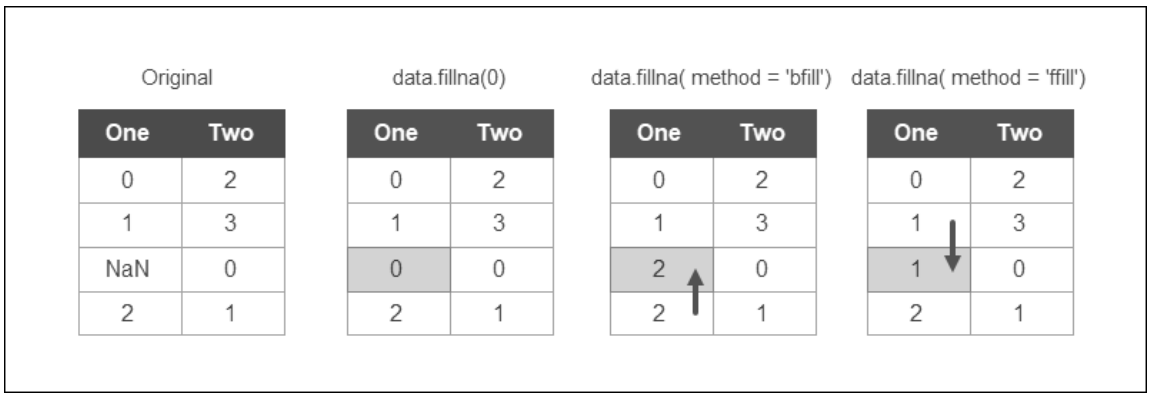
	Max Speed
Animal	
Parrot	50.0
Falcon	750.0

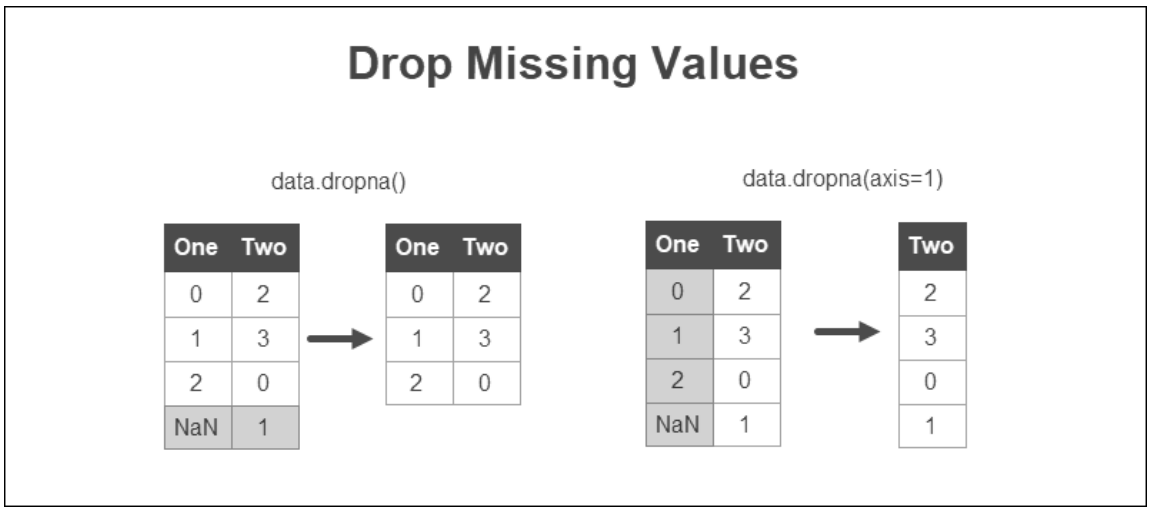
In [52]:
```
df.groupby(pd.Grouper(key='Animal')).sum()
```

Out[52]:

	Max Speed
Animal	
Falcon	750.0
Parrot	50.0

Working with missing data





In [100]:
```
import numpy as np
import pandas as pd
dfmulti=pd.DataFrame({'city':['Noida',np.NaN,"Delhi","Kanpur",np.NaN],
                      'institue':['ABC','DEF',np.NaN,"BDC",'CCC']})
dfmulti
```

Out[100]:

	city	institue
0	Noida	ABC
1	NaN	DEF
2	Delhi	NaN
3	Kanpur	BDC
4	NaN	CCC

In [72]:
```
#dfmulti.dropna()
dfmulti['city'].fillna('Meerut',inplace=True)
dfmulti['institue'].fillna('BBC',inplace=True)
```

In [73]:
```
dfmulti
```

Out[73]:

	city	institue
0	Noida	ABC
1	Meerut	DEF
2	Delhi	BBC
3	Kanpur	BDC

In [82]:
```
dfmulti.fillna({'city':'Lucknow', 'institue':'Aptech'}, inplace=True,limit=
```

In [84]:
```
dfmulti
```

Out[84]:

	city	institue
0	Noida	ABC
1	Lucknow	DEF
2	Delhi	Aptech
3	Kanpur	BDC
4	NaN	CCC

In [96]:
```
# Only replace the first NaN element.
dfmulti.fillna({'city':'Ghaziabad', 'institue':'Aptech'}, inplace=True,limi
```

In [98]:
```
dfmulti
```

Out[98]:

	city	institue
0	Noida	ABC
1	Ghaziabad	DEF
2	Delhi	Aptech
3	Kanpur	BDC
4	Ghaziabad	CCC

In [105]:
```
import numpy as np
import pandas as pd
df=pd.DataFrame({'city':['Noida',np.NaN,"Delhi","Kanpur",np.NaN],
                      'institute':['ABC','DEF',np.NaN,"BDC",'CCC']})


for col in ['city', 'institute']:
     df[col].fillna("Lucknow",inplace = True)
df
```

Out[105]:

	city	institute
0	Noida	ABC
1	Lucknow	DEF
2	Delhi	Lucknow
3	Kanpur	BDC
4	Lucknow	CCC

	Survived	Age	Fare
0	1.0	22.0	7.250
1	NaN	NaN	NaN
2	NaN	26.0	7.925
3	NaN	NaN	53.100
4	0.0	35.0	8.050
...	...	...	...
886	0.0	27.0	13.000
887	1.0	19.0	30.000
888	0.0	NaN	23.450
889	1.0	26.0	30.000
890	0.0	32.0	7.750

891 rows × 3 columns

In [37]:
```
import numpy as np
import pandas as pd
df=pd.DataFrame({'city':['Noida',np.NaN,"Delhi","Kanpur",np.NaN],
                 'institute':['ABC','DEF',np.NaN,"BDC",np.NaN]})
df
```

Out[37]:

	city	institute
0	Noida	ABC
1	NaN	DEF
2	Delhi	NaN
3	Kanpur	BDC
4	NaN	NaN

In [20]:
```
df['city'].fillna('kanpur',inplace=True)
df
```

Out[20]:

	city	institute
0	Noida	ABC
1	kanpur	DEF
2	Delhi	NaN
3	Kanpur	BDC
4	kanpur	NaN

In [38]:
```
#Fill on selected row and column
values = {'city': 100, 'institute': 200}
df.iloc[4] = df.iloc[4].fillna(value=values)
df
```

Out[38]:

	city	institute
0	Noida	ABC
1	NaN	DEF
2	Delhi	NaN
3	Kanpur	BDC
4	100	200

In [36]:
```
#fill on all rows and all columns
values = {'city': 100, 'institute': 200}
df.iloc[:,0:5] = df.iloc[:,0:5].fillna(value=values)
df
```

Out[36]:

	city	institute
0	Noida	ABC
1	100	DEF
2	Delhi	200
3	Kanpur	BDC
4	100	200

In [ ]:
```
df.iloc[4].fillna('Haridwar',inplace=True)
```

In [5]:
```
df
```

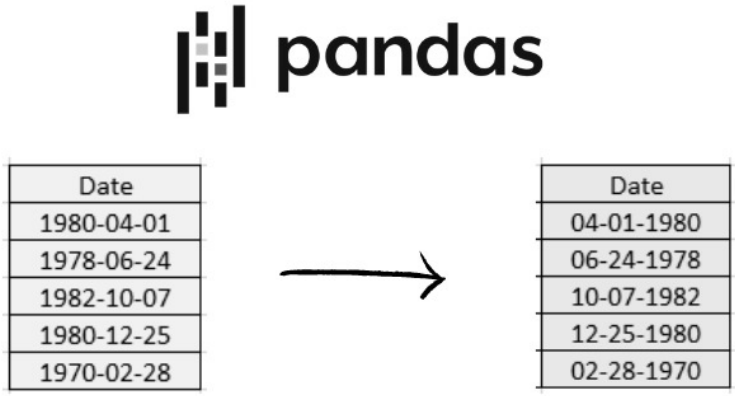
Out[5]:

	city	institute
0	Noida	ABC
1	NaN	DEF
2	Delhi	NaN
3	Kanpur	BDC
4	Haridwar	Haridwar

In [39]:
```
df.iloc[4].fillna('Haridwar',inplace=True)
```

pandas.DataFrame.select_dtypes

DataFrame.select_dtypes(include=None, exclude=None)[source] Return a subset of the DataFrame’s columns based on the column dtypes.



In [57]:
```
df = pd.DataFrame({'a': [1, 2] * 3,
                   'b': [True, False] * 3,
                   'c': [1.0, 2.0] * 3,
                    'd': [True, False] * 3,
                     'e':[101,'Meerut',True]*2}
                    )
df
```

Out[57]:

	a	b	c	d	e
0	1	True	1.0	True	101
1	2	False	2.0	False	Meerut
2	1	True	1.0	True	True
3	2	False	2.0	False	101
4	1	True	1.0	True	Meerut
5	2	False	2.0	False	True

In [58]:
```
df.dtypes
```

Out[58]:
```
a      int64
b       bool
c    float64
d       bool
e     object
dtype: object
```

In [53]:
```
df.select_dtypes(include='bool')
```

Out[53]:

	b	d
0	True	True
1	False	False
2	True	True
3	False	False
4	True	True
5	False	False

In [54]:
```
df.select_dtypes(exclude='bool')
```

Out[54]:

	a	c	e
0	1	1.0	101
1	2	2.0	Meerut
2	1	1.0	True
3	2	2.0	101
4	1	1.0	Meerut
5	2	2.0	True

In [55]:
```
df.select_dtypes(include=['float64'])
```

Out[55]:

	c
0	1.0
1	2.0
2	1.0
3	2.0
4	1.0
5	2.0

In [59]:
```
df.select_dtypes(exclude=['int64'])
```

Out[59]:

	b	c	d	e
0	True	1.0	True	101
1	False	2.0	False	Meerut
2	True	1.0	True	True
3	False	2.0	False	101
4	True	1.0	True	Meerut
5	False	2.0	False	True

pandas.DataFrame.empty

In [81]:
```
df_empty = pd.DataFrame({'A' : [],
                         'B':[],
                         'c':[]})
df_empty
```

Out[81]:

	A	B	c

In [80]:
```
df_empty.empty
```

Out[80]:
```
False
```

In [83]:
```
df = pd.DataFrame({'A' : [np.nan],
                   'B':"kanpur"})
df
```

Out[83]:

	A	B
0	NaN	kanpur

In [ ]:
```

```

In [88]:
```
import numpy as np
import pandas as pd
df = pd.DataFrame(np.nan, index=[0, 1, 2, 3], columns=['A', 'B'])

df
```

Out[88]:

	A	B
0	NaN	NaN
1	NaN	NaN
2	NaN	NaN
3	NaN	NaN

In [108]:
```
data=[[None,20],
      [1,3],
      [],
      [6,7]]
data

df = pd.DataFrame(data, index=[0, 1, 2, 3], columns=['A', 'B'])
df
```

Out[108]:

	A	B
0	NaN	20.0
1	1.0	3.0
2	NaN	NaN
3	6.0	7.0

In [111]:
```
df.drop(2, inplace = True)
df
```

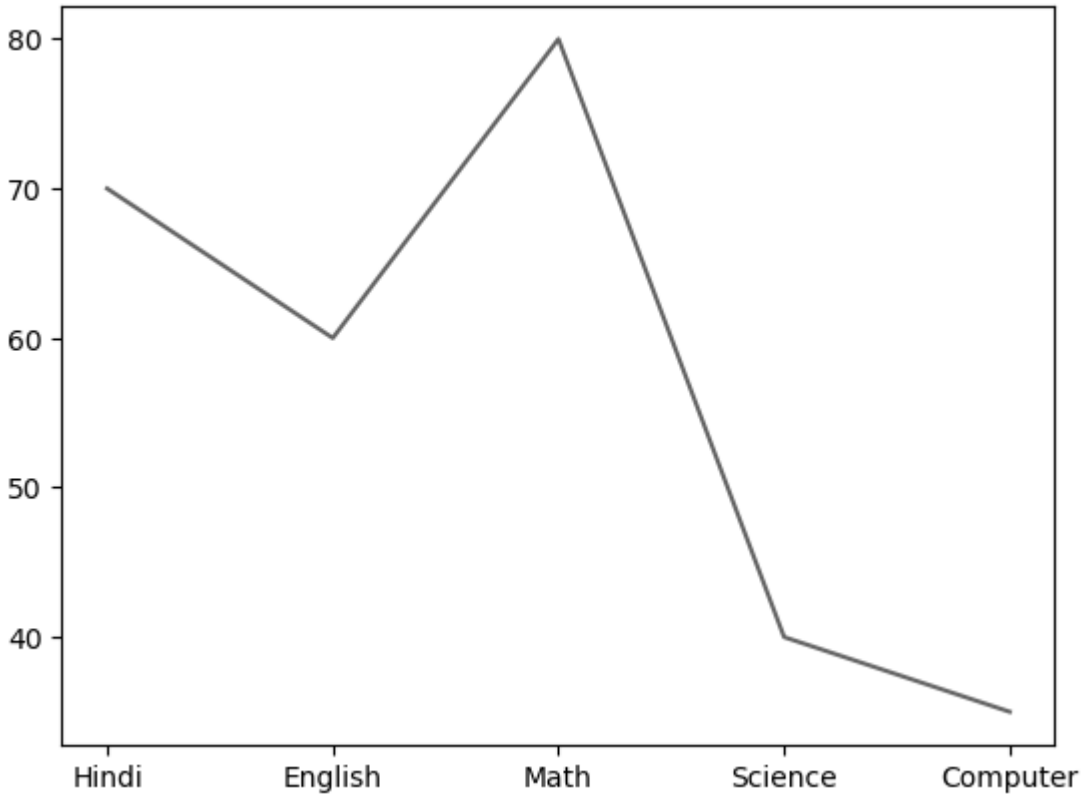
Out[111]:

	A	B
0	NaN	20.0
1	1.0	3.0
3	6.0	7.0

Data Visualization in python using Matplotlib

Line Chart

In [11]:
```
import matplotlib.pyplot as plt
import numpy as np

marks=np.array([70,60,80,40,35])
sub=np.array(['Hindi','English','Math','Science','Computer'])
plt.plot(sub,marks)
plt.show()
```



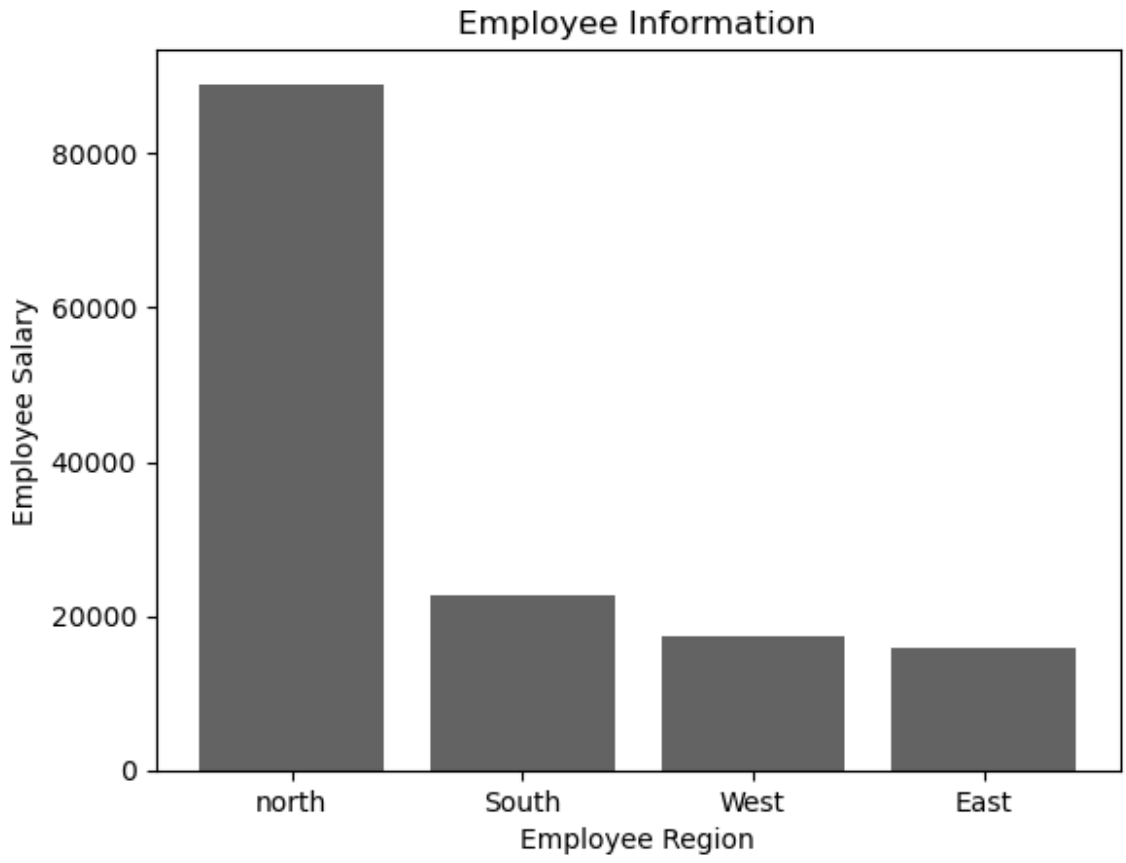
In [30]:
```
import pandas as pd

df=pd.read_csv('CSVFileForPanda.csv')
df.head()
```

Out[30]:

	empcode	First Name	Last Name	Dept	Region	Branch	Hiredate	Salary
0	5	Seema	Ranganathan	R&D	north	Kanpur	NaN	89000
1	84	Raja	Raymondekar	Sales	north	Ferozepur	1/1/1977	30000
2	39	Sheetal	Dodhia	Finance	north	Delhi	1/1/1977	24500
3	38	Chitra	Pednekar	Finance	north	Aligarh	10/1/1982	24500
4	90	Priyanka	Mehta	R&D	north	Jaipur	1/6/1980	22750

Column Chart

In [88]:
```
import matplotlib.pyplot as plt
EmpRegion=df['Region']
EmpSalary=df['Salary']
plt.bar(EmpRegion,EmpSalary)
plt.title("Employee Information")
plt.xlabel('Employee Region')
plt.ylabel('Employee Salary')
plt.show()
```

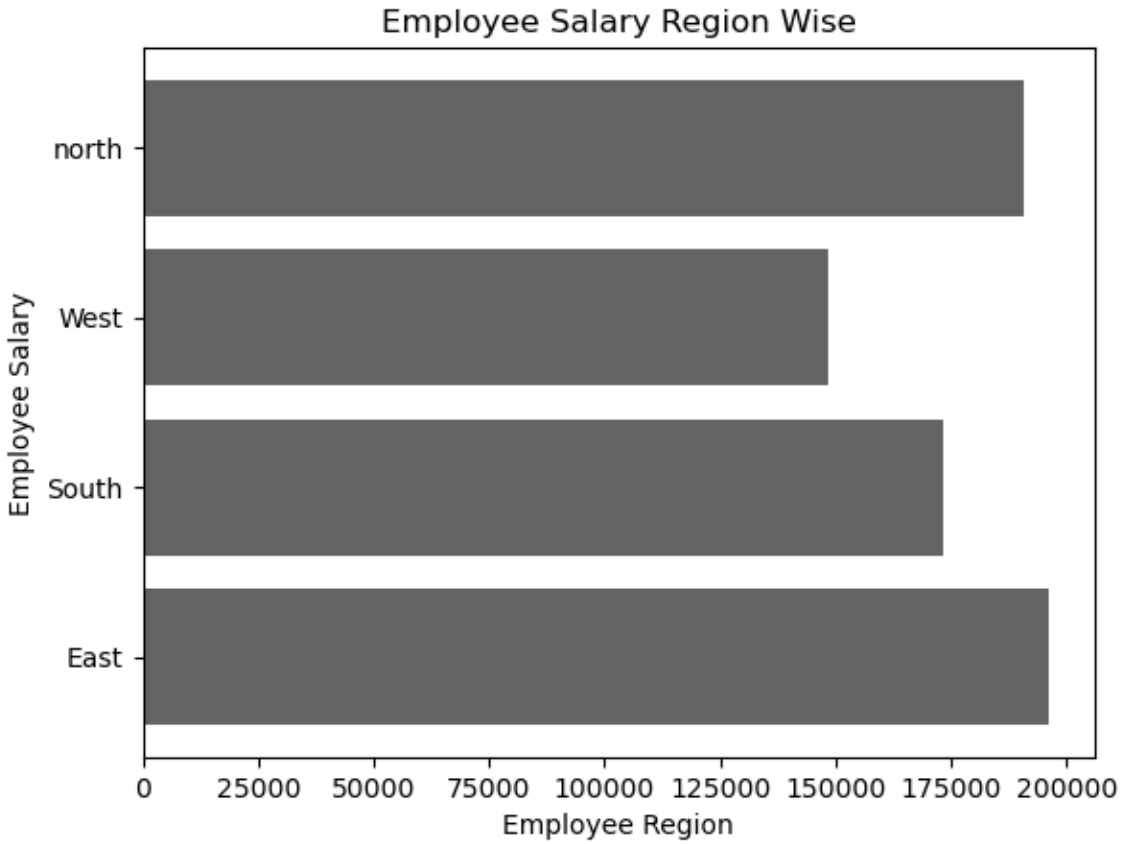


In [31]:
```
dfd=df[['Region','Salary']]
# print(dfd)
pie_df=dfd.groupby('Region').sum()
pie_df=pd.DataFrame(pie_df)

pie_df.reset_index(inplace=True)
print(pie_df)
plt.bar(pie_df['Region'],pie_df['Salary'])
plt.title("Employee Salary Region Wise")
plt.xlabel('Employee Region')
plt.ylabel('Employee Salary')
plt.show()
```

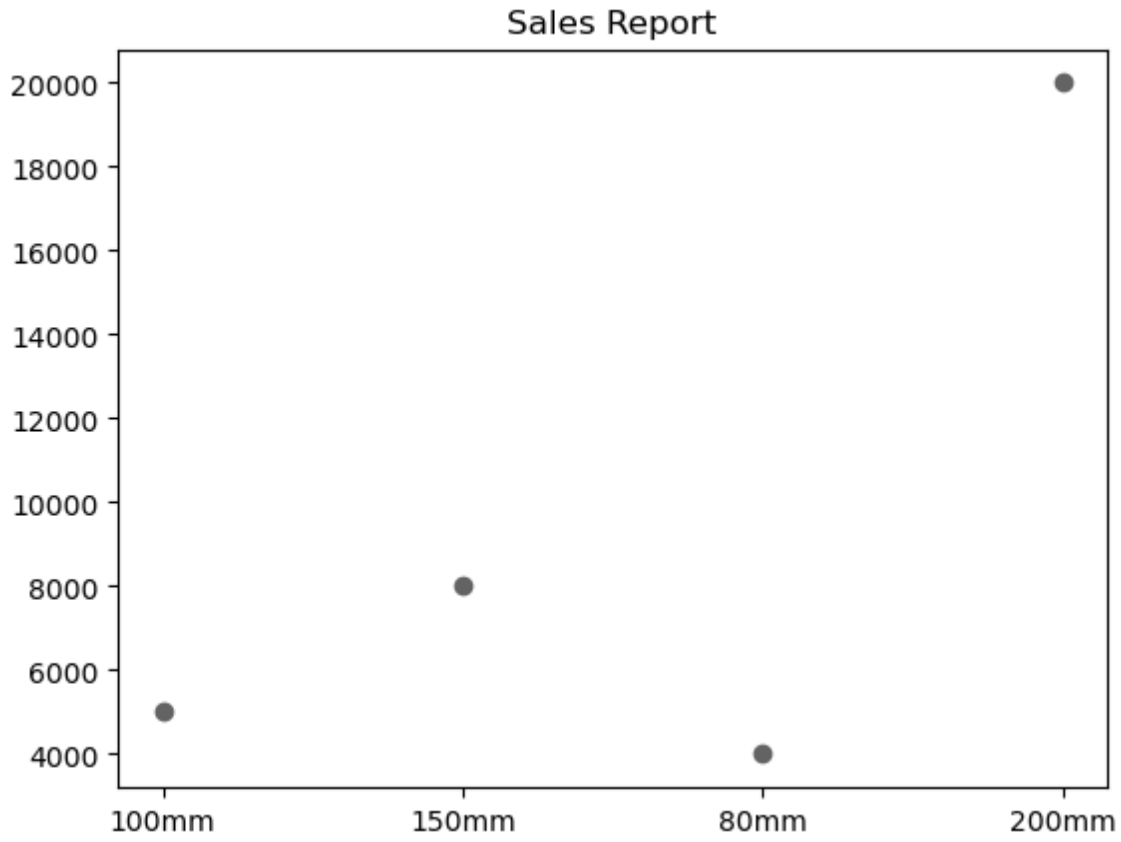
```
  Region  Salary
0   East  196100
1  South  173125
2   West  148200
3  north  190750
```

Bar Chart

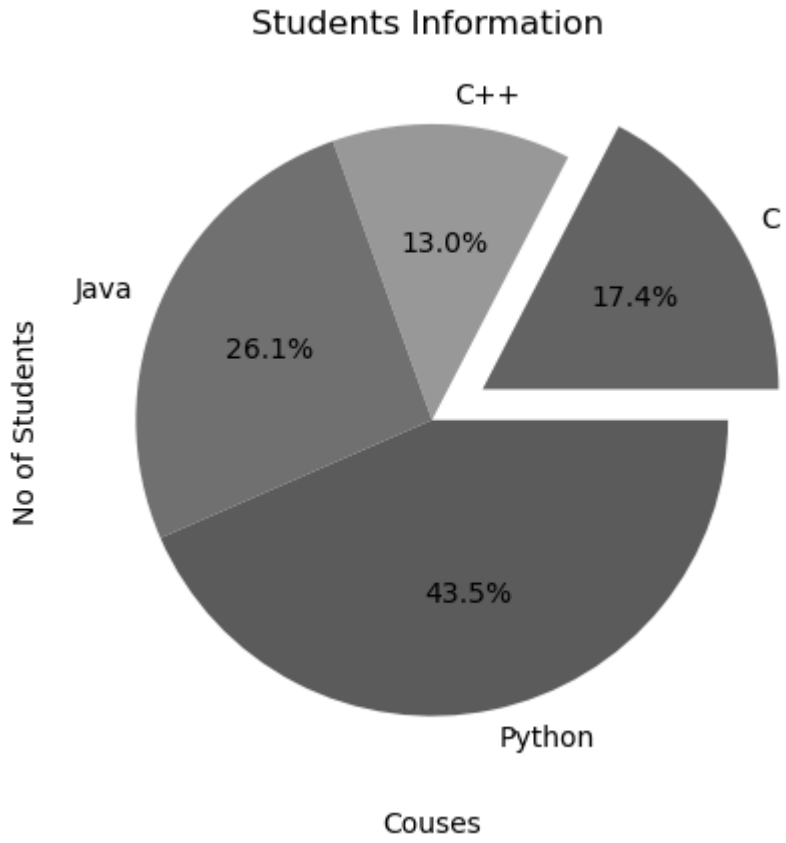
In [22]:
```
plt.barh(pie_df['Region'],pie_df['Salary'])
plt.title("Employee Salary Region Wise")
plt.xlabel('Employee Region')
plt.ylabel('Employee Salary')
plt.show()
```



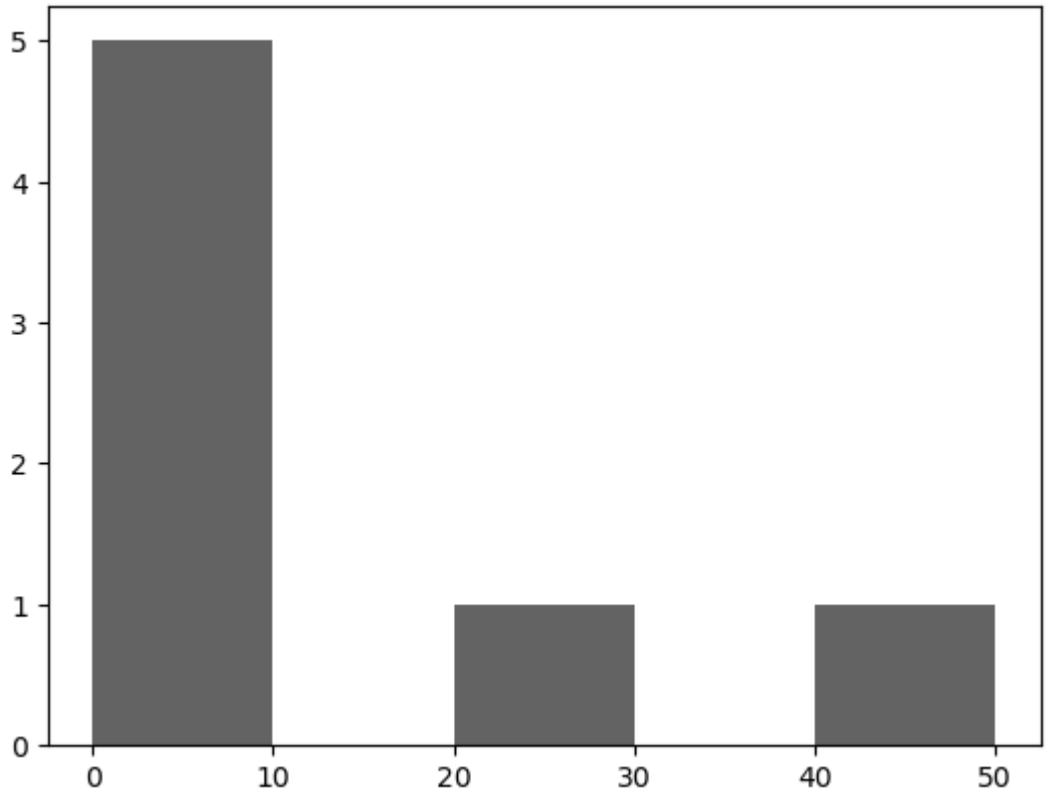
In [23]:
```
#scatter Chart
plt.title("Sales Report")
rain=np.array(['100mm','150mm','80mm','200mm'])
sales=np.array([5000,8000,4000,20000])

plt.scatter(rain,sales)
plt.show()
```



In [15]:
```
from matplotlib import pyplot as plt
import numpy as np

course=np.array(['C','C++','Java','Python'])
marks=np.array([20,15,30,50])
explode=(.2,0,0,0)
plt.pie(marks,labels=course,autopct='%1.1f%%',explode=explode)
plt.title("Students Information ")
plt.xlabel("Couses")
plt.ylabel("No of Students")
#plt.legend(course,frameon=False,loc="upper left",ncol=1)
plt.show()
```



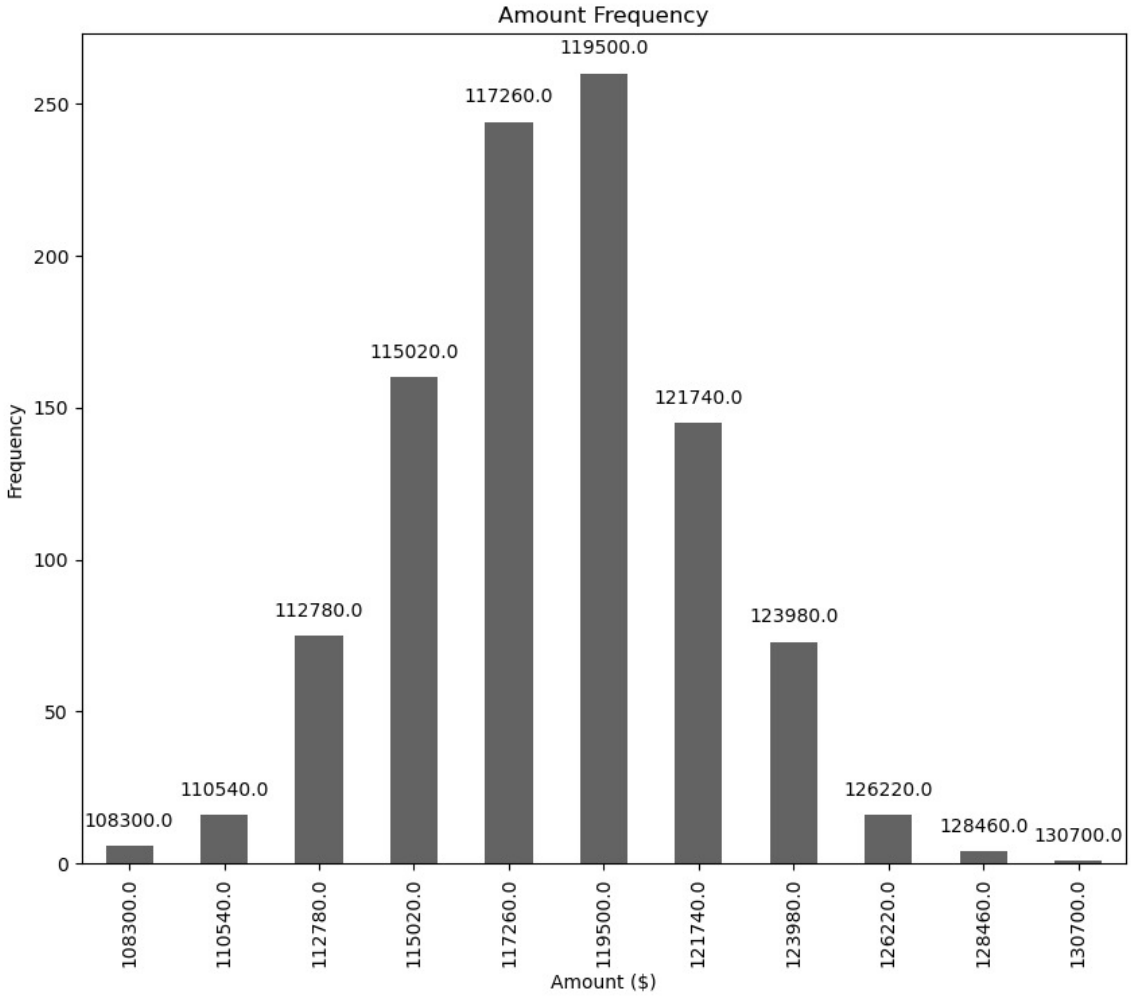
In [26]:
```
#Histogram Chart
#A histogram is a chart that groups numeric data into bins, displaying the
from matplotlib import pyplot as plt
Marks=[50,60,70,80,90,55,60,70,60,70,80,25,6,7,8,9,3]
ranges=[0,10,20,30,40,50]
plt.hist(Marks,ranges,histtype='bar',linewidth=0.1)
plt.show()
```



Grouped bar chart with labels

In [57]:

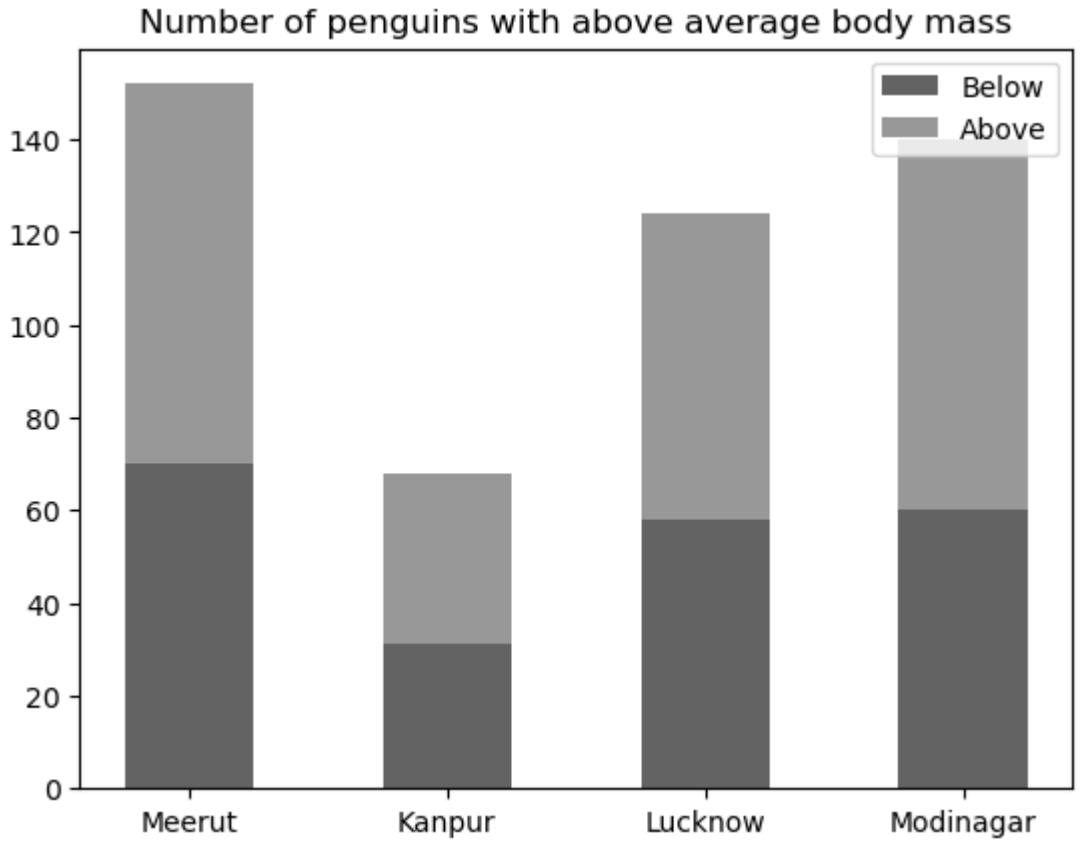
```
import pandas as pd
import matplotlib.pyplot as plt

frequencies = [6, 16, 75, 160, 244, 260, 145, 73, 16, 4, 1,]
freq_series = pd.Series(frequencies)
x_labels = [
    108300.0,
    110540.0,
    112780.0,
    115020.0,
    117260.0,
    119500.0,
    121740.0,
    123980.0,
    126220.0,
    128460.0,
    130700.0,
]

# Plot the figure.
plt.figure(figsize=(10, 8))
ax = freq_series.plot(kind="bar")
ax.set_title("Amount Frequency")
ax.set_xlabel("Amount ($)")
ax.set_ylabel("Frequency")
ax.set_xticklabels(x_labels)

rects = ax.patches

for rect, label in zip(rects, x_labels):
    height = rect.get_height()
    ax.text( rect.get_x() + rect.get_width() / 2, height + 5, label, ha="ce

plt.show()
```



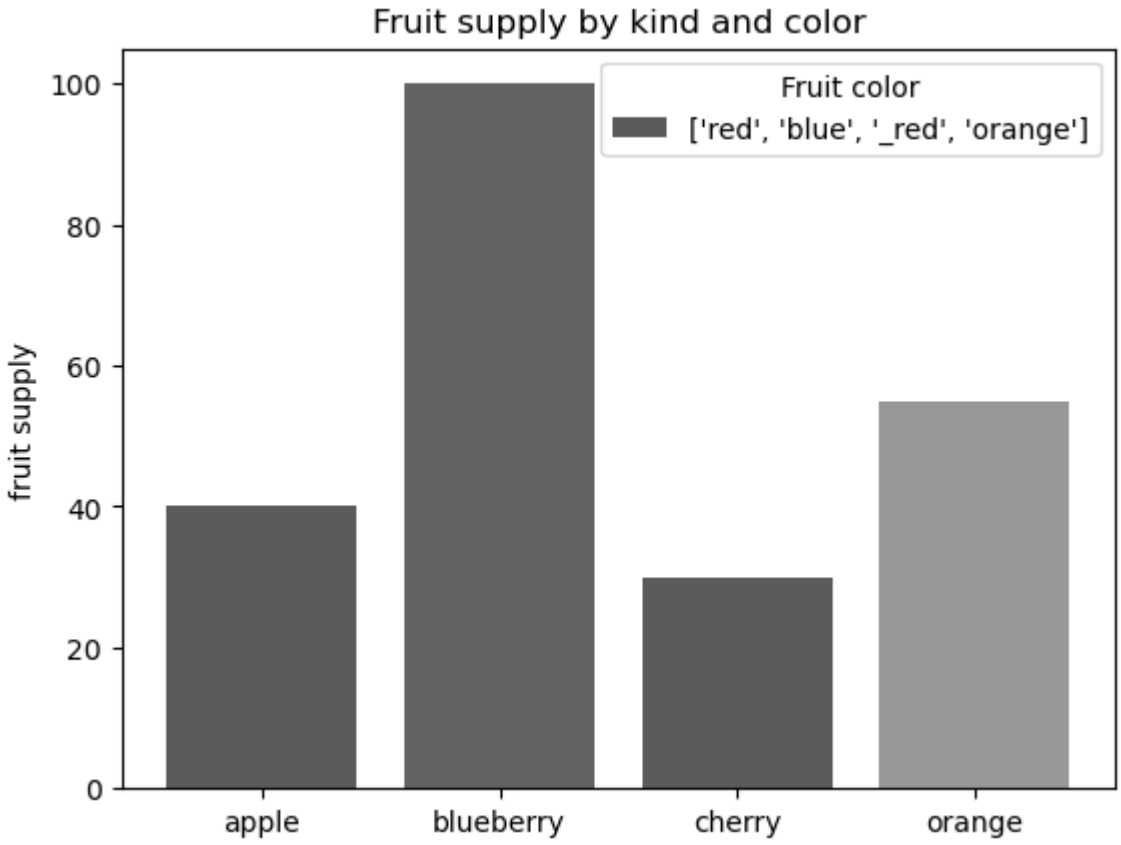
Stacked bar chart

In [89]:
```
import matplotlib.pyplot as plt
import numpy as np

species = ('Meerut','Kanpur','Lucknow','Modinagar')
weight_counts = {  "Below": [70, 31, 58,60],
                   "Above": [82, 37, 66,80],
                 }
fig, ax = plt.subplots()
bottom = np.zeros(4)

for boolean, weight_count in weight_counts.items():
    print(boolean)
    print(weight_count)
    p = ax.bar(species, weight_count, 0.5, label=boolean, bottom=bottom)
    print(bottom)
    bottom += weight_count

ax.set_title("Number of penguins with above average body mass")
ax.legend(loc="upper right")

plt.show()
```

```
Below
[70, 31, 58, 60]
[0. 0. 0. 0.]
Above
[82, 37, 66, 80]
[70. 31. 58. 60.]
```



In [91]:
```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()

fruits = ['apple', 'blueberry', 'cherry', 'orange']
counts = [40, 100, 30, 55]
bar_labels = ['red', 'blue', '_red', 'orange']
bar_colors = ['tab:red', 'tab:blue', 'tab:red', 'tab:orange']

ax.bar(fruits, counts, label=bar_labels, color=bar_colors)

ax.set_ylabel('fruit supply')
ax.set_title('Fruit supply by kind and color')
ax.legend(title='Fruit color')

plt.show()
```



MatplotLib Documentation Library Link

https://matplotlib.org/stable/gallery/lines_bars_and_markers/bar_colors.html#sphx-glr-gallery-lines-bars-and-markers-bar-colors-py (https://matplotlib.org/stable/gallery/lines_bars_and_markers/bar_colors.html#sphx-glr-gallery-lines-bars-and-markers-bar-colors-py)

Seaborn Library

Seaborn is a library mostly used for statistical plotting in Python. It is built on top of Matplotlib and provides beautiful default styles and color palettes to make statistical plots more attractive.

seaborn

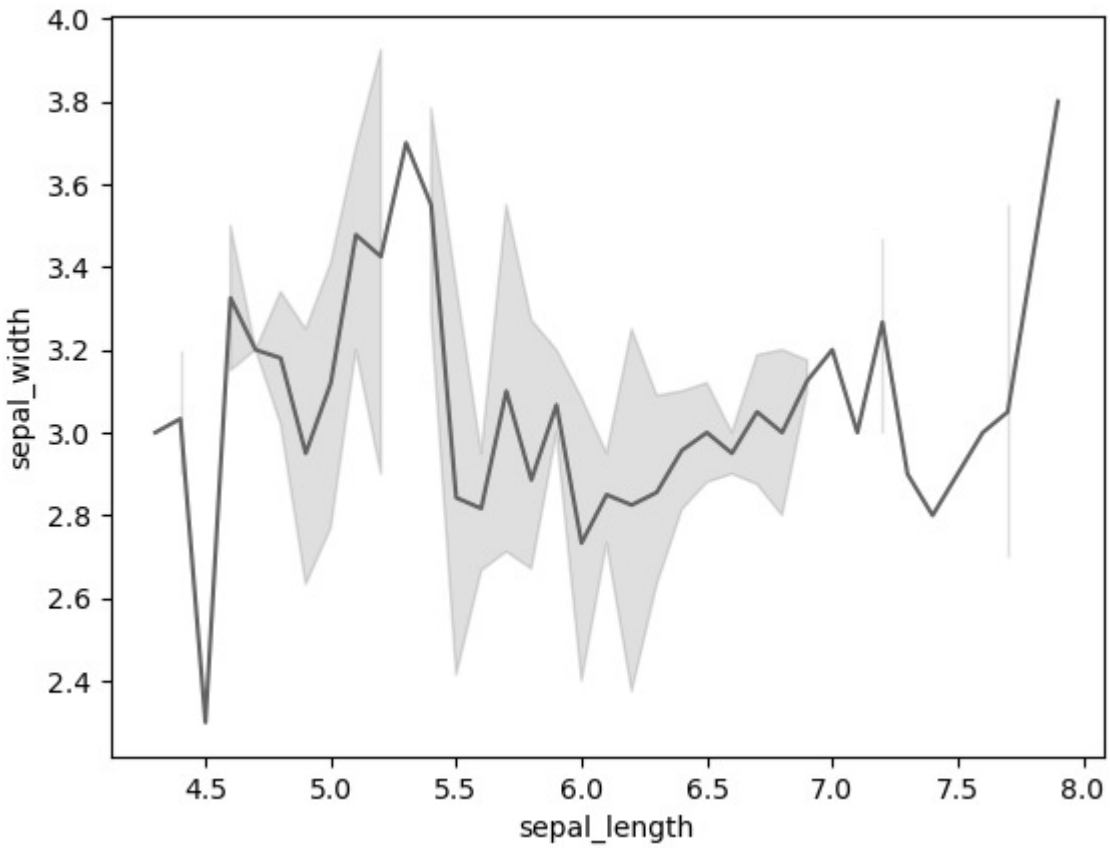
iris setosa

iris versicolor

iris virginica

In [2]:
```
import seaborn as sns
# loading dataset
data = sns.load_dataset("iris")
# draw lineplot

sns.lineplot(x="sepal_length", y="sepal_width", data=data)
```

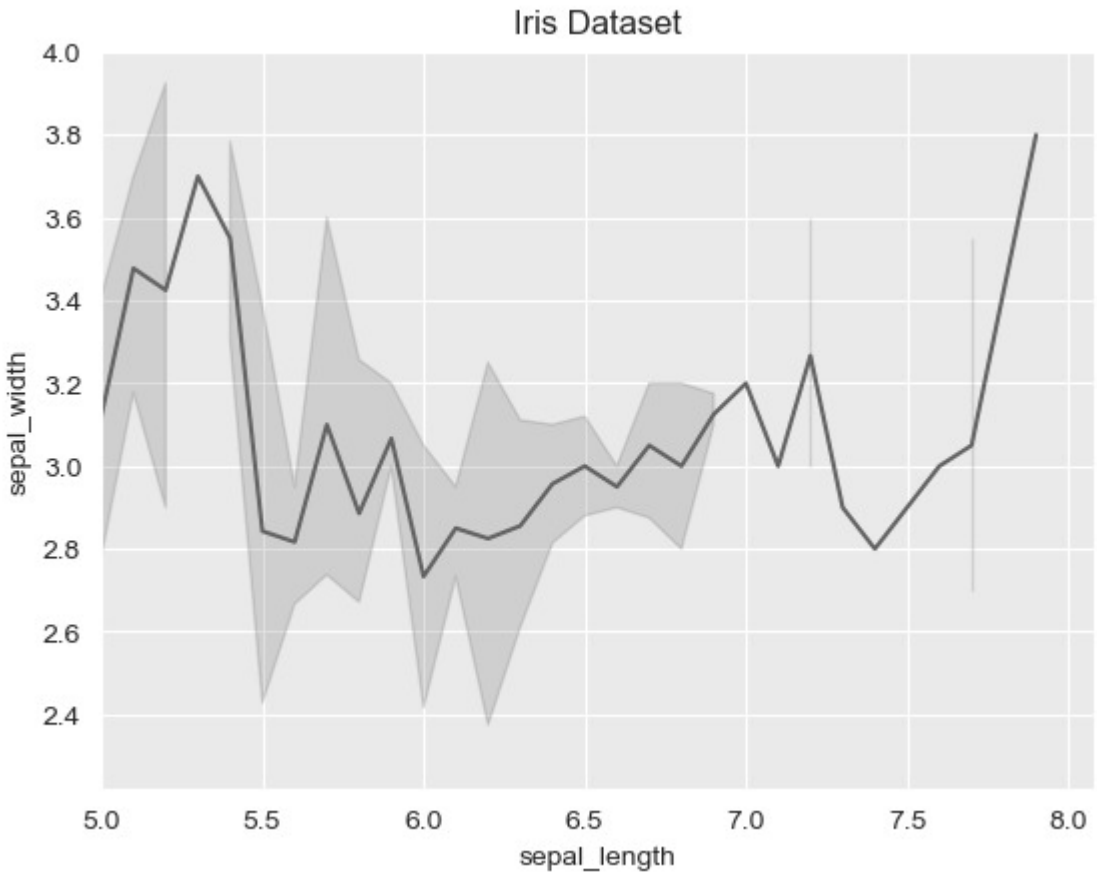
Out[2]: <AxesSubplot:xlabel='sepal_length', ylabel='sepal_width'>



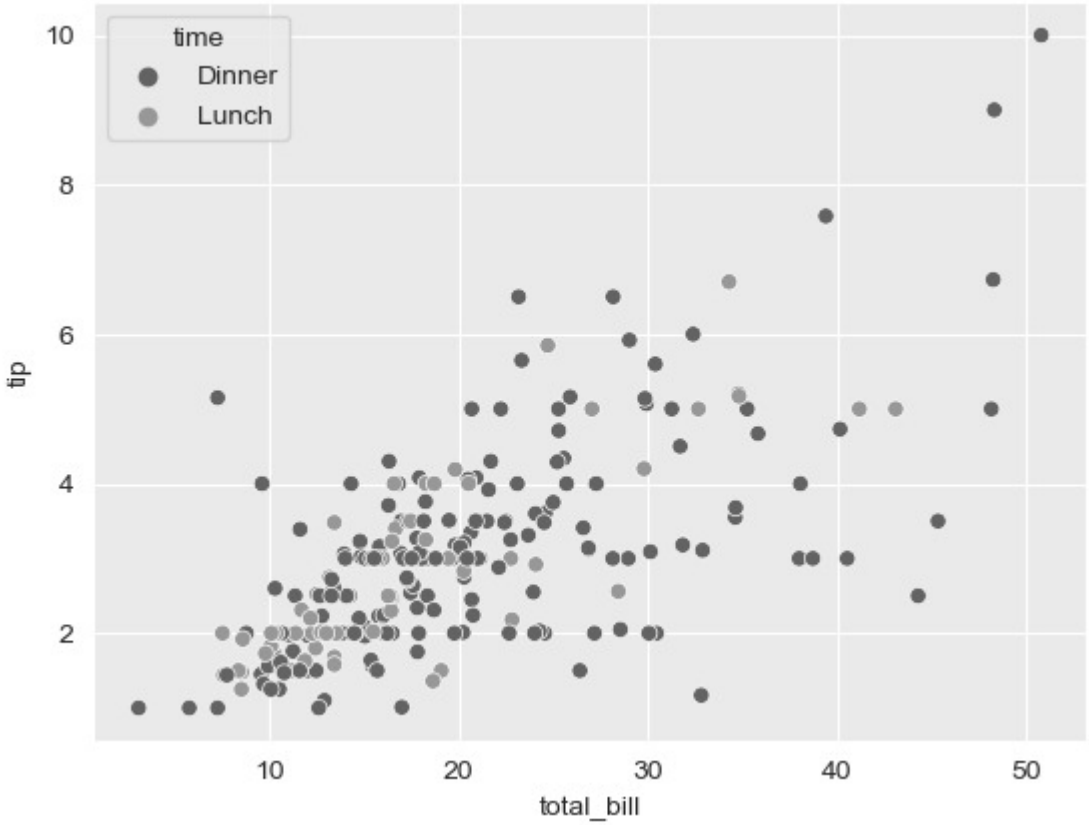
set_style() method is used to set the aesthetic of the plot. It means it affects things like the color of the axes, whether the grid is active or not, or other aesthetic elements. There are five themes available in Seaborn.

• darkgrid • whitegrid • dark • white • ticks

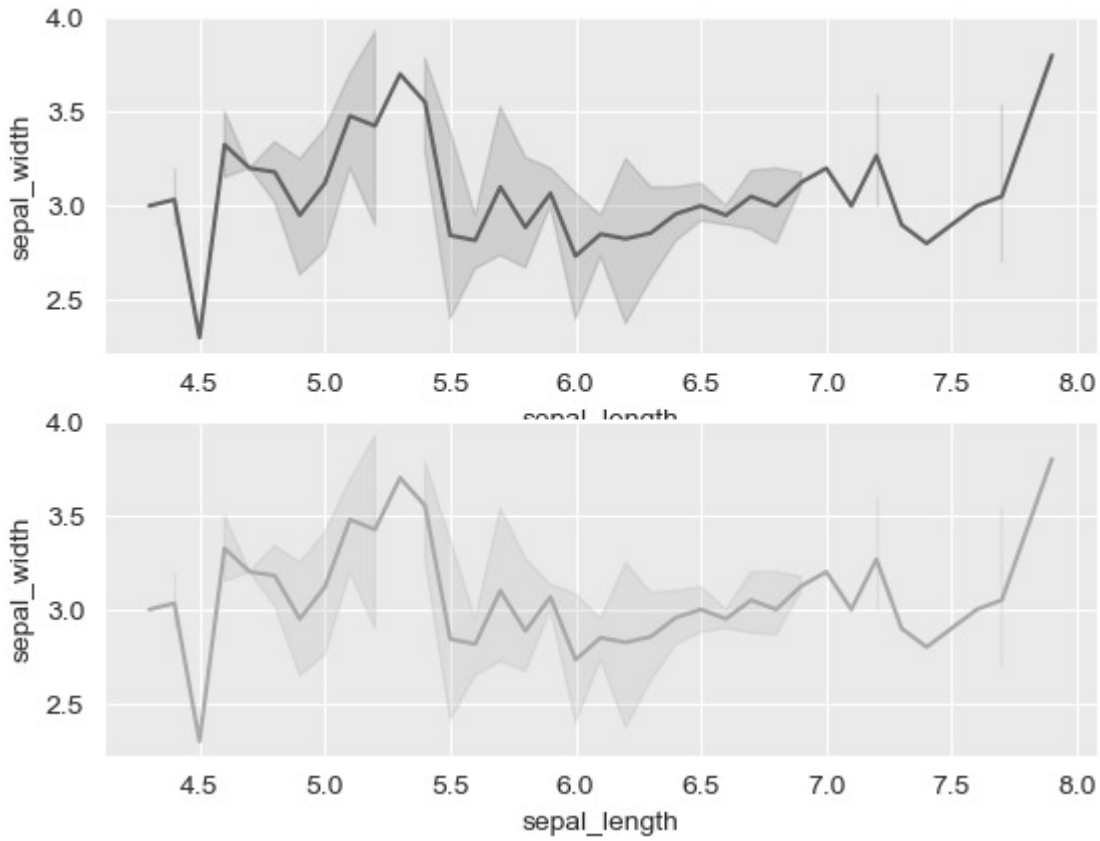
In [16]:
```
#Seaborn using by
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt

# loading dataset
data=pd.read_csv("iris.csv")

sns.lineplot(x="sepal_length", y="sepal_width", data=data)
plt.title("Iris Dataset")
plt.xlim(5) #start for x dimentions
sns.set_style("darkgrid")
plt.show()
```



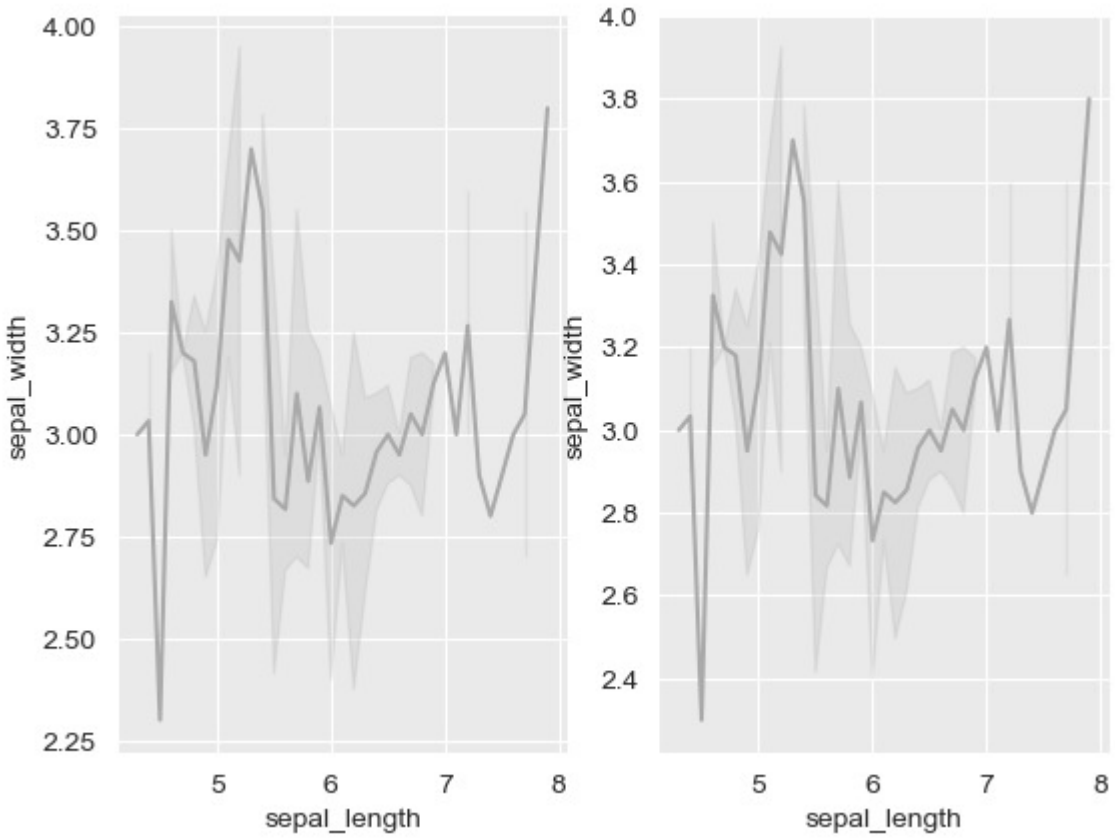
In [49]:
```
import seaborn as sns
import matplotlib.pyplot as plt

df=pd.read_csv("tips.csv")

sns.scatterplot(data=df, x='total_bill', y='tip', hue='time')

plt.show()
```



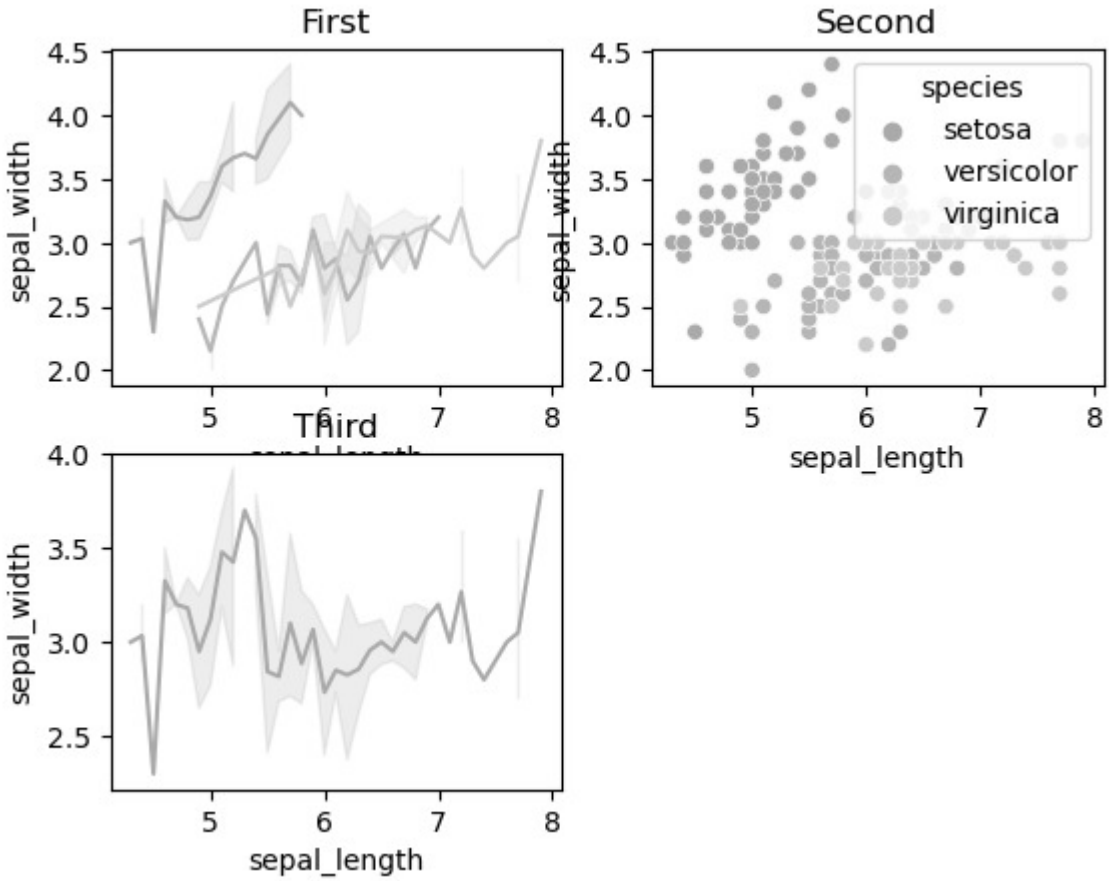
In [53]:
```
import seaborn as sns
import matplotlib.pyplot as plt
# loading dataset
data = sns.load_dataset("iris")
def plot():
    sns.lineplot(x="sepal_length", y="sepal_width", data=data)

 # Adding the subplot
plt.subplot(211)
sns.set_palette('Accent')
plot()


plt.subplot(212)
plot()
```



In [56]:
```
import seaborn as sns
import matplotlib.pyplot as plt
# loading dataset
data = sns.load_dataset("iris")
def plot():
    sns.lineplot(x="sepal_length", y="sepal_width", data=data)

 # Adding the subplot
plt.subplot(121)
sns.set_palette('Accent')
plot()


plt.subplot(122)
plot()
```



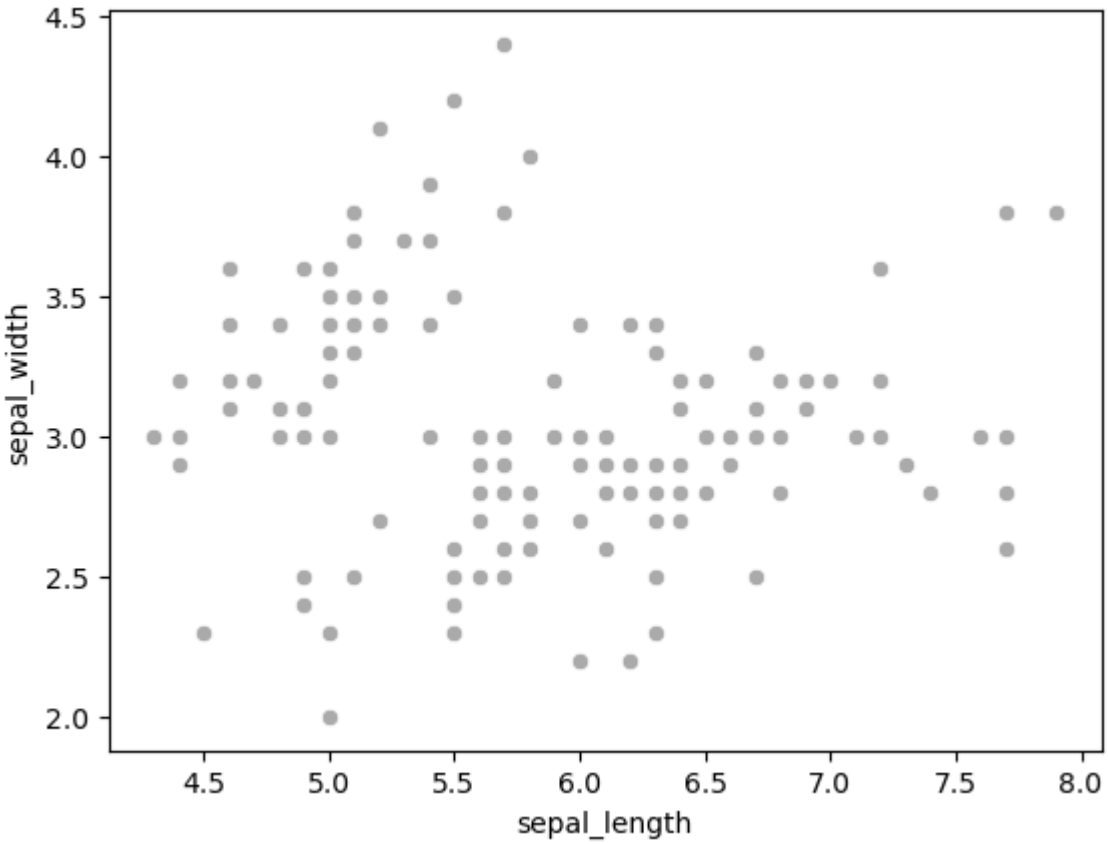
In [1]:
```
import seaborn as sns
import matplotlib.pyplot as plt
# loading dataset
data = sns.load_dataset("iris")
def plot():
    sns.lineplot(x="sepal_length", y="sepal_width", data=data)

 # Adding the subplot

plt.subplot(2,2,1) #row,col,dim
sns.set_palette('Accent')
plt.title("First")
sns.lineplot(x="sepal_length", y="sepal_width", data=data,hue="species",leg

plt.subplot(2,2,2)
plt.title("Second")
sns.scatterplot(data=data,x="sepal_length", y="sepal_width",hue='species')

plt.subplot(2,2,3)
plt.title("Third")
plot()
```



Scatter Plot

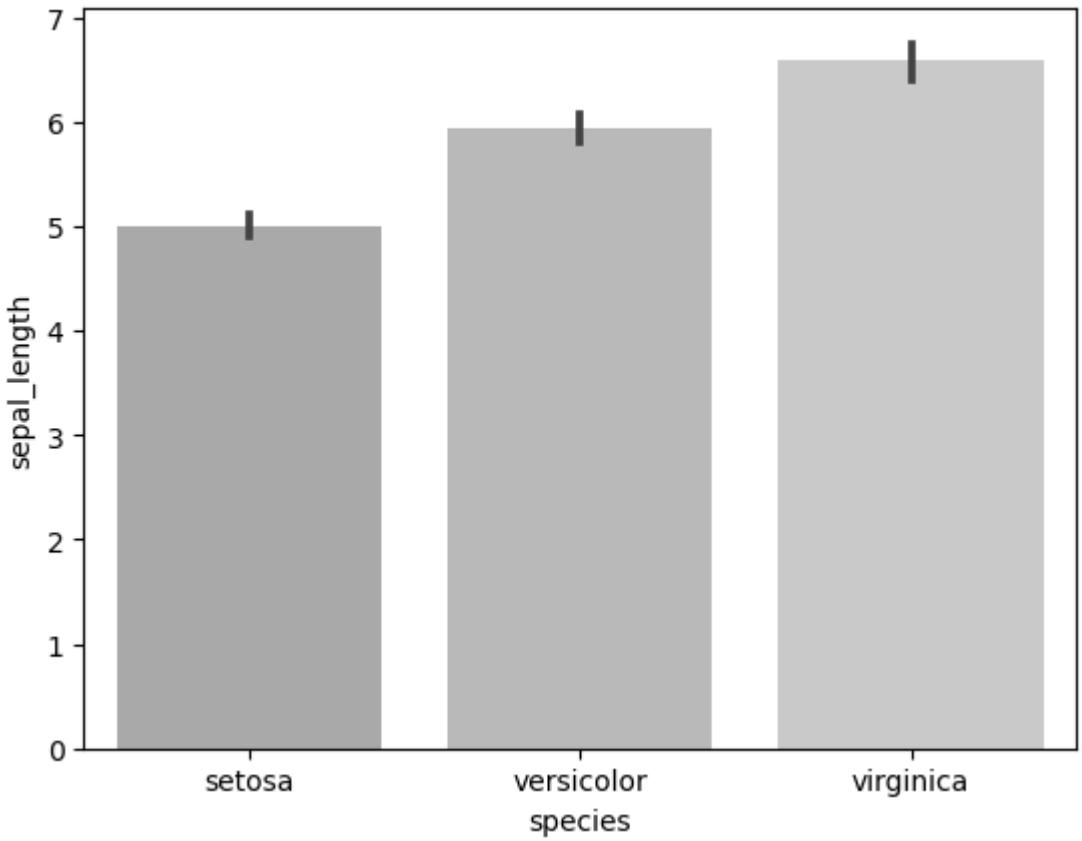
The scatter plot is a mainstay of statistical visualization.

In [3]:
```
import seaborn as sns
import matplotlib.pyplot as plt
data = sns.load_dataset("iris")
sns.scatterplot(x='sepal_length', y='sepal_width', data=data)
plt.show()
```



Seaborn Bar Chart

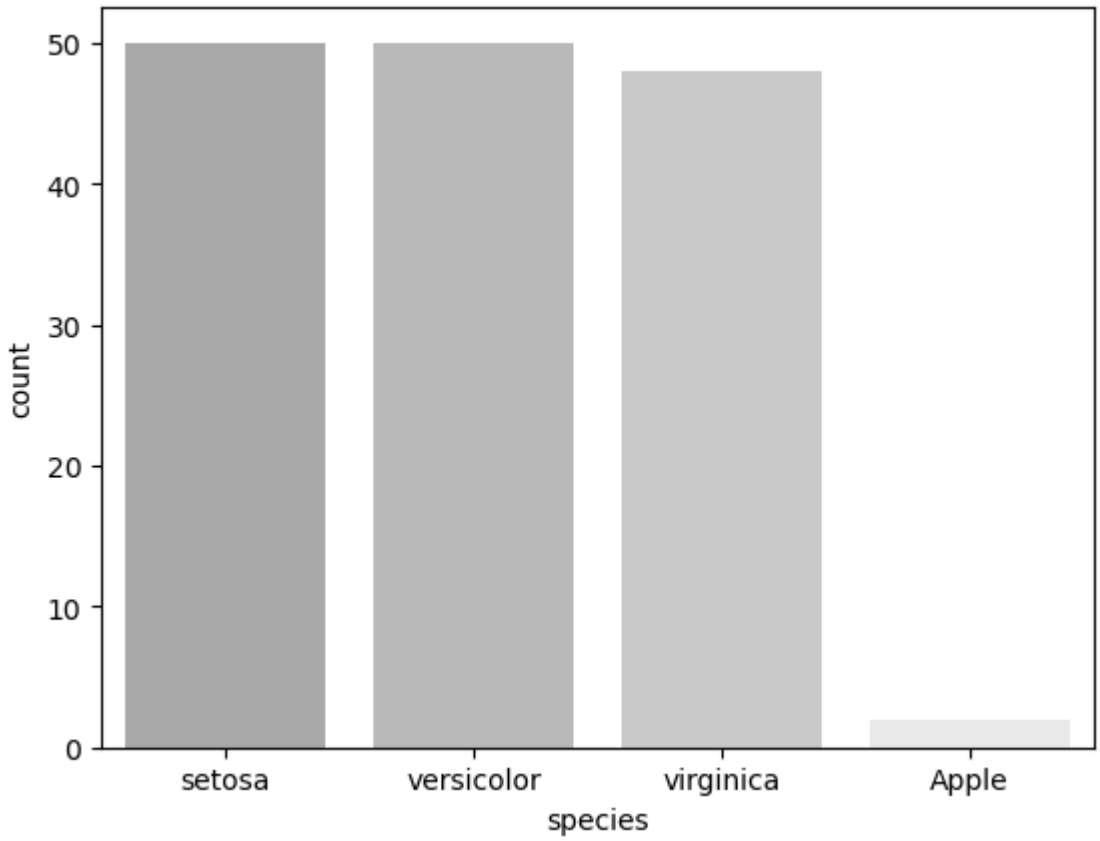
In [7]:
```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
# Loading
data=pd.read_csv("iris.csv")
sns.barplot(x='species', y='sepal_length', data=data)
plt.show()
```



A countplot basically counts the categories and returns a count of their

occurrences. It is one of the most simple plots provided by the seaborn library. It can be created using the countplot() method. countplot([x, y, hue, data, order, …])

In [14]:
```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
# loading
data=pd.read_csv("iris.csv")
#print(data['species'])
sns.countplot(x='species', data=data)
plt.show()
```



A boxplot

is sometimes known as the box and whisker plot.It shows the distribution of the quantitative data that represents the comparisons between variables. boxplot shows the quartiles of the dataset while the whiskers extend to show the rest of the distribution i.e. the dots indicating the presence of outliers. It is created using the boxplot() method

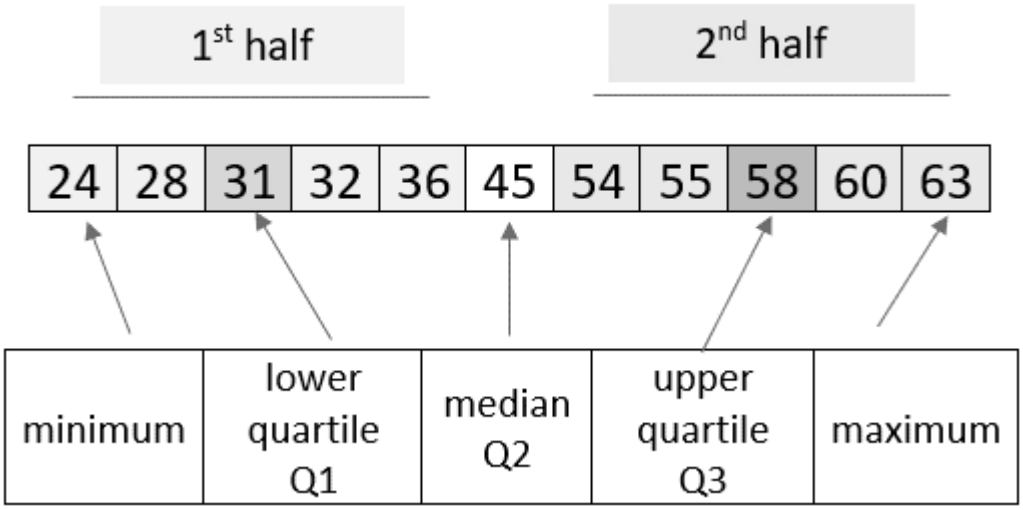
Quartile Formula

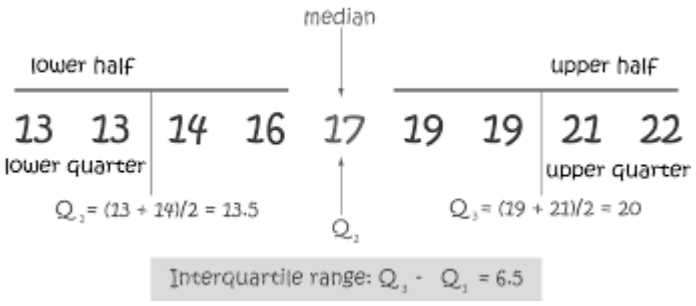
$$\text{Lower Quartile (Q1)} = (N+1) \times \frac{1}{4}$$

$$\text{Middle Quartile (Q2)} = (N+1) \times \frac{2}{4}$$

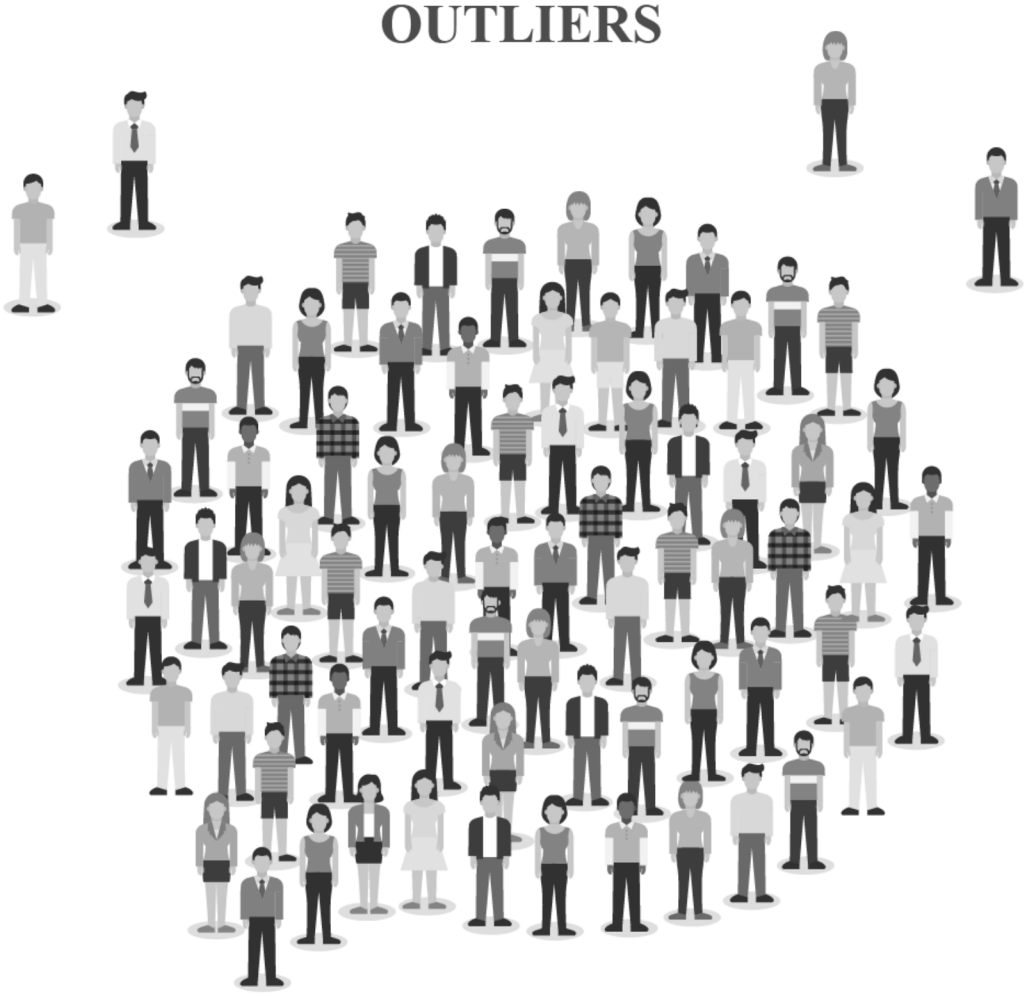
$$\text{Upper Quartile (Q3)} = (N+1) \times \frac{3}{4}$$

EDUCBA

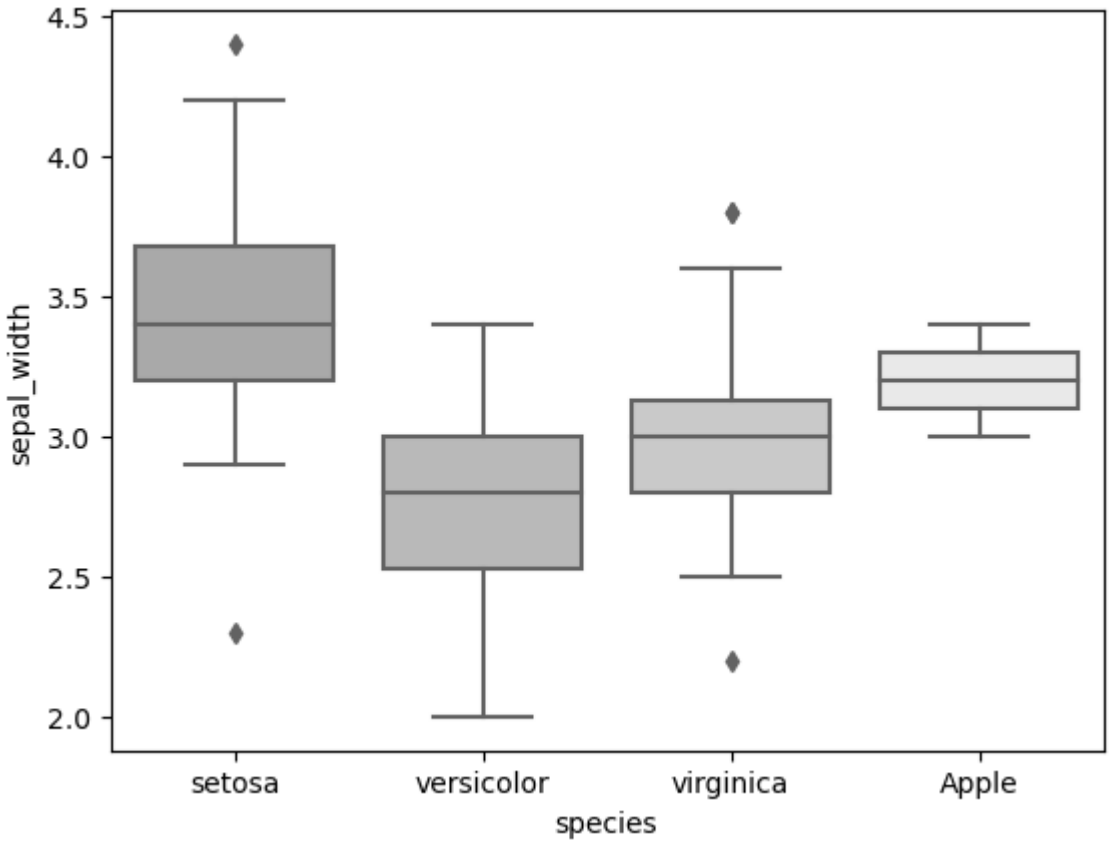




OUTLIERS



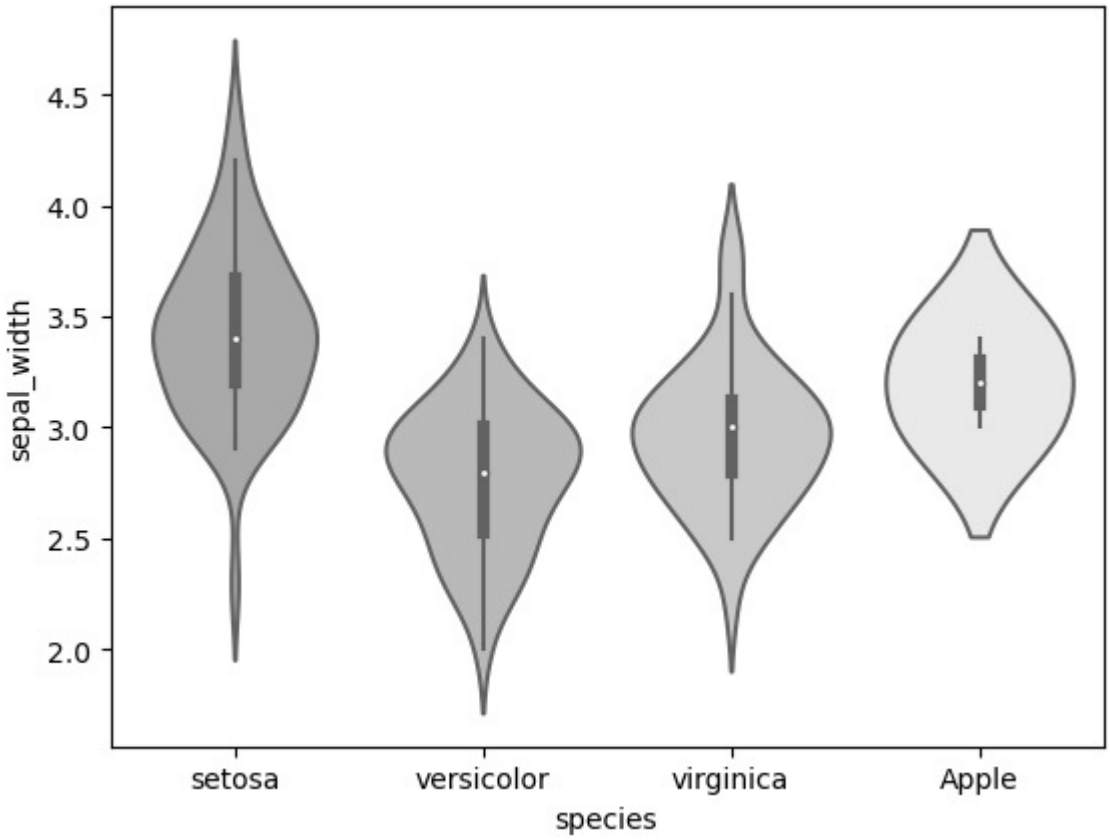
In [15]:
```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
# Loading
data=pd.read_csv("iris.csv")
sns.boxplot(x='species', y='sepal_width', data=data)
plt.show()
```



Violinplot Chart

It is similar to the boxplot except that it provides a higher, more advanced visualization and uses the kernel density estimation to give a better description about the data distribution

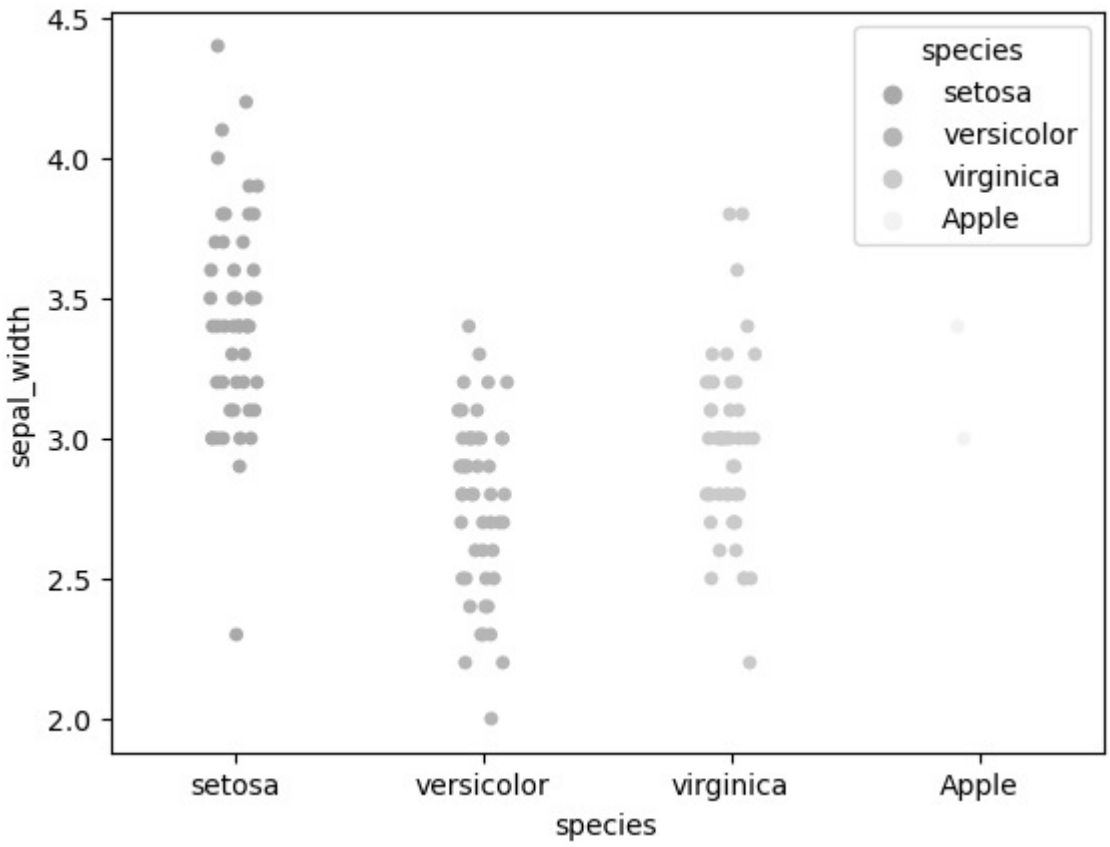
In [17]:
```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
# Loading
data=pd.read_csv("iris.csv")
sns.violinplot(x='species', y='sepal_width', data=data)
plt.show()
```



Stripplot Chart

It basically creates a scatter plot based on the category. It is created using the stripplot() method

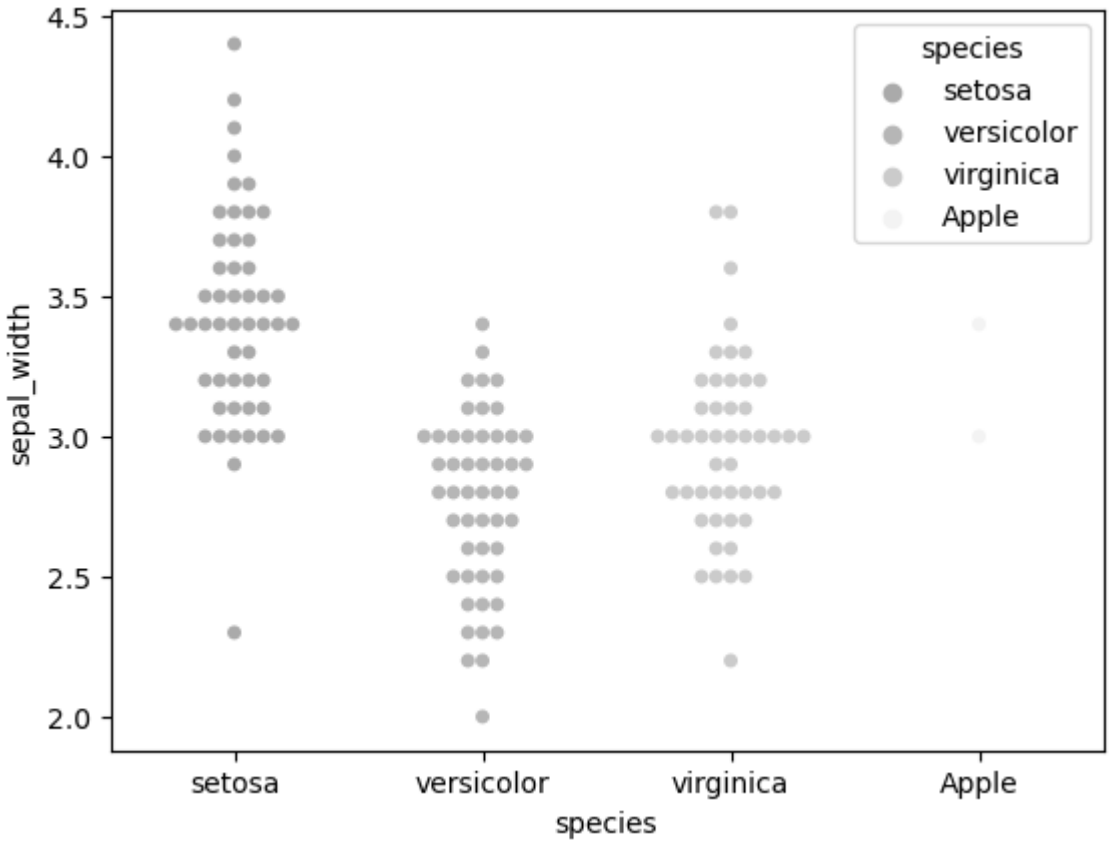
In [20]:
```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
# Loading
data=pd.read_csv("iris.csv")
sns.stripplot(x='species', y='sepal_width', data=data,hue='species')
plt.show()
```



Swarmplot chart

Swarmplot is very similar to the stripplot except the fact that the points are adjusted so that they do not overlap.Some people also like combining the idea of a violin plot and a stripplot to form this plot. One drawback to using swarmplot is that sometimes they dont scale well to really large numbers and takes a lot of computation to arrange them. So in case we want to visualize a swarmplot properly we can plot it on top of a violinplot. It is plotted using the swarmplot() method

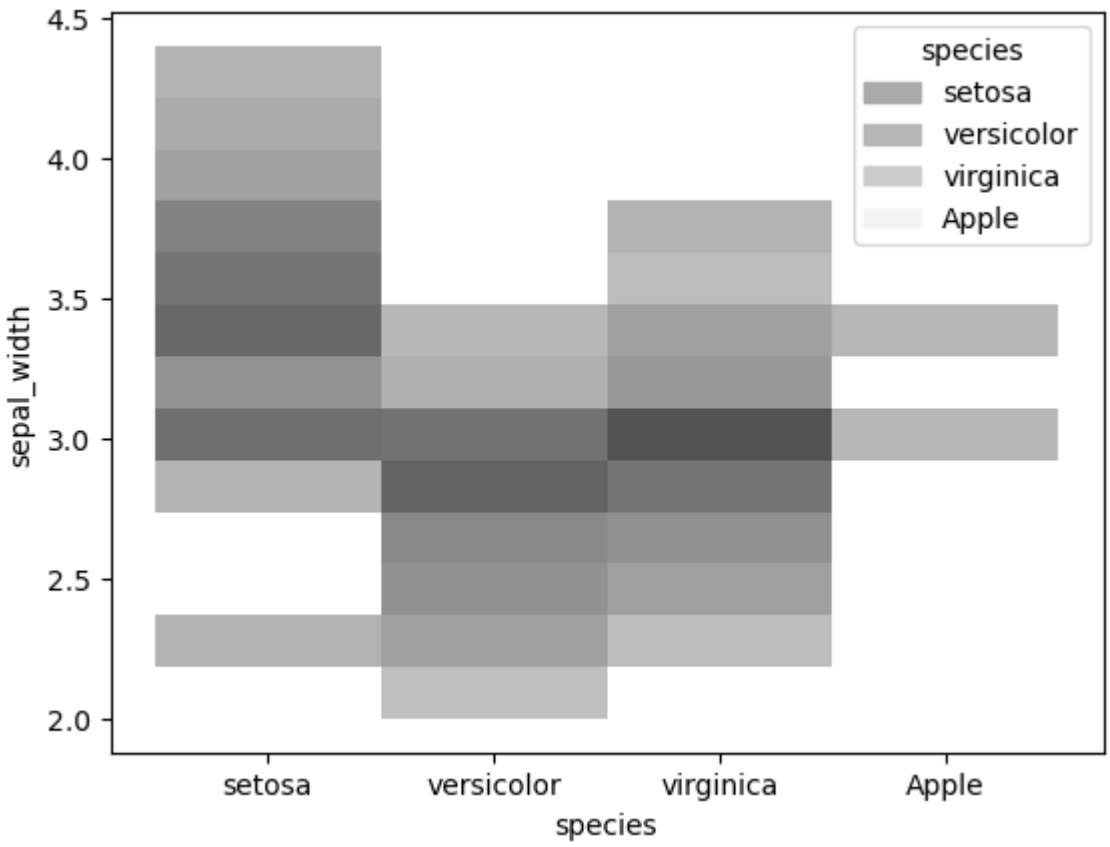
In [21]:
```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
# Loading
data=pd.read_csv("iris.csv")
sns.swarmplot(x='species', y='sepal_width', data=data,hue='species')
plt.show()
```



Histogram Chart

A histogram is basically used to represent data provided in a form of some groups.It is accurate method for the graphical representation of numerical data distribution. It can be plotted using the histplot() function

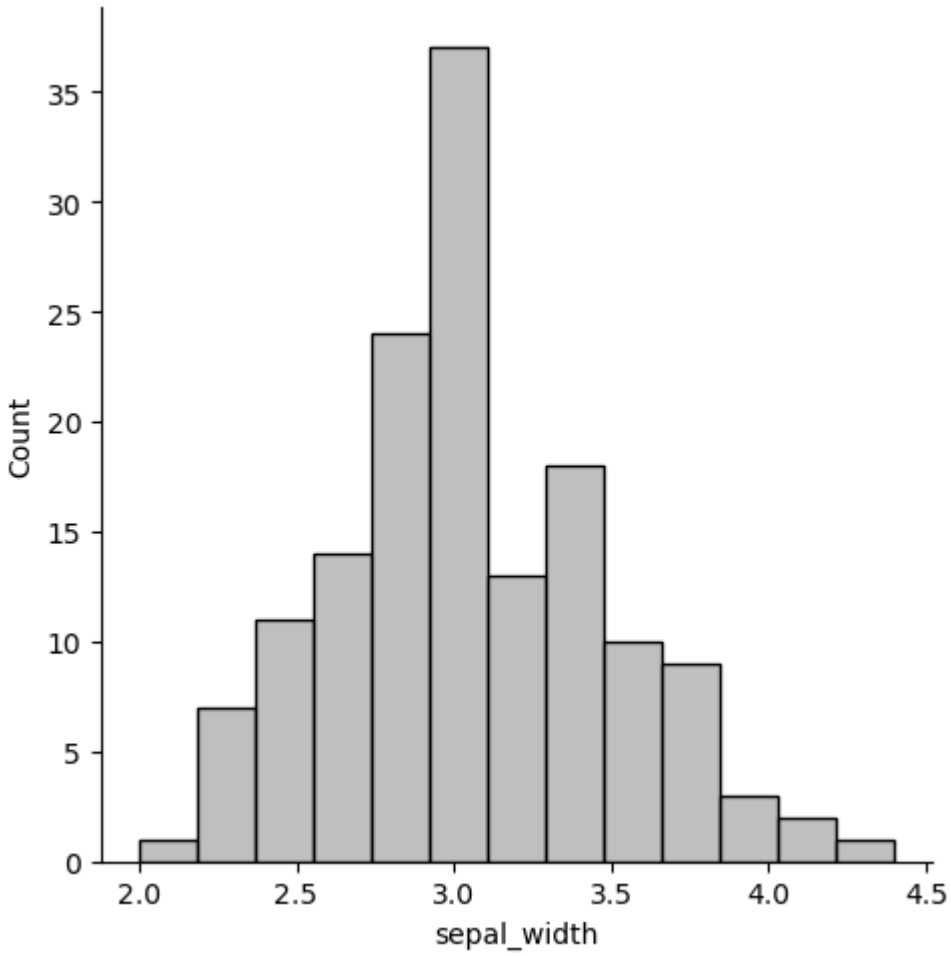
In [27]:
```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
# Loading
data=pd.read_csv("iris.csv")
sns.histplot(x='species', y='sepal_width', data=data,hue='species')
plt.show()
```



Distplot Chart

Distplot is used basically for univariant set of observations and visualizes it through a histogram i.e. only one observation and hence we choose one particular column of the dataset. It is potted using the distplot() method.

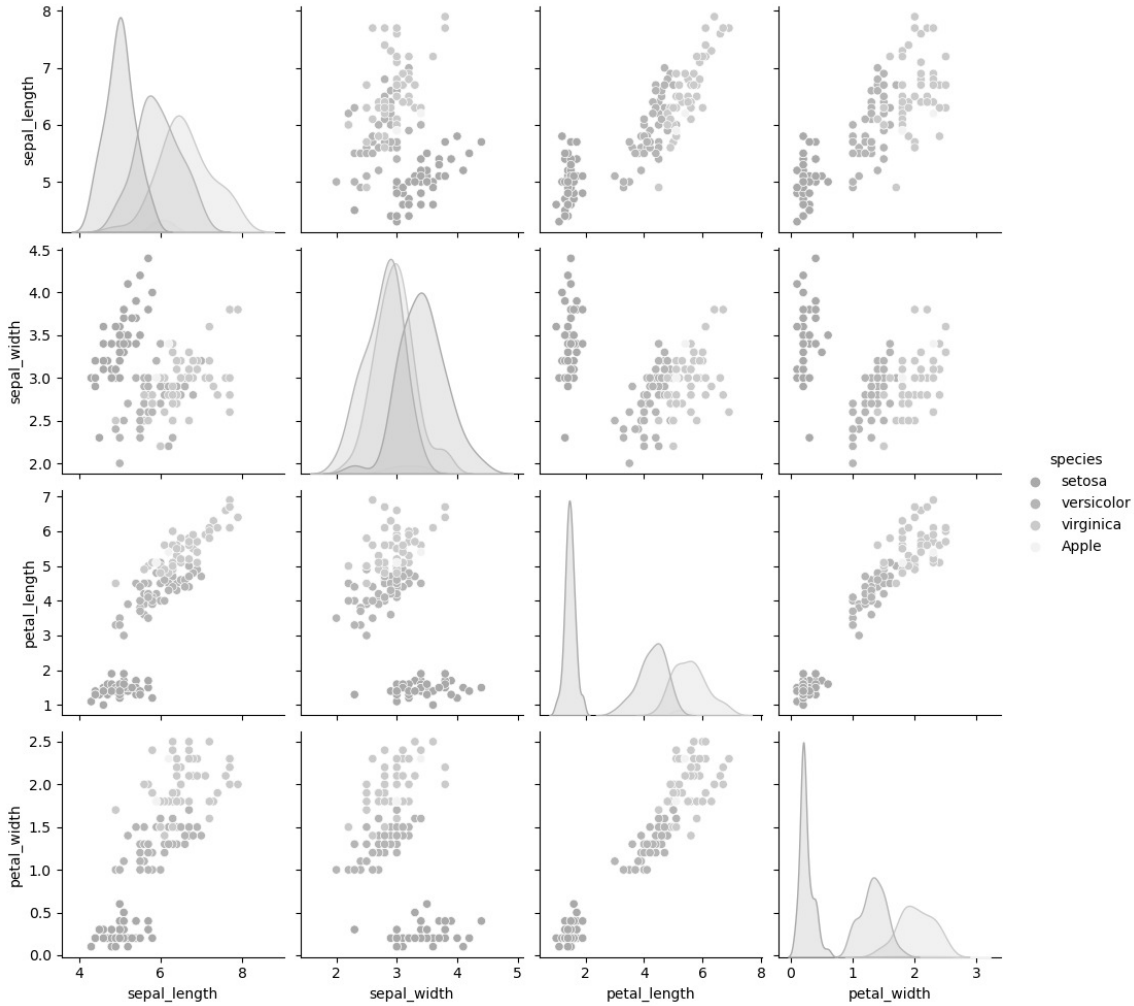
In [29]:
```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
# Loading
data=pd.read_csv("iris.csv")
sns.displot(data['sepal_width'])
plt.show()
```



Pairplot Chart

represents pairwise relation across the entire dataframe and supports an additional argument called hue for categorical separation. What it does basically is create a jointplot between every possible numerical column and takes a while if the dataframe is really huge. It is plotted using the pairplot() method.

In [39]:
```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
# Loading
data=pd.read_csv("iris.csv")
sns.pairplot(data=data, hue='species')
plt.show()
```



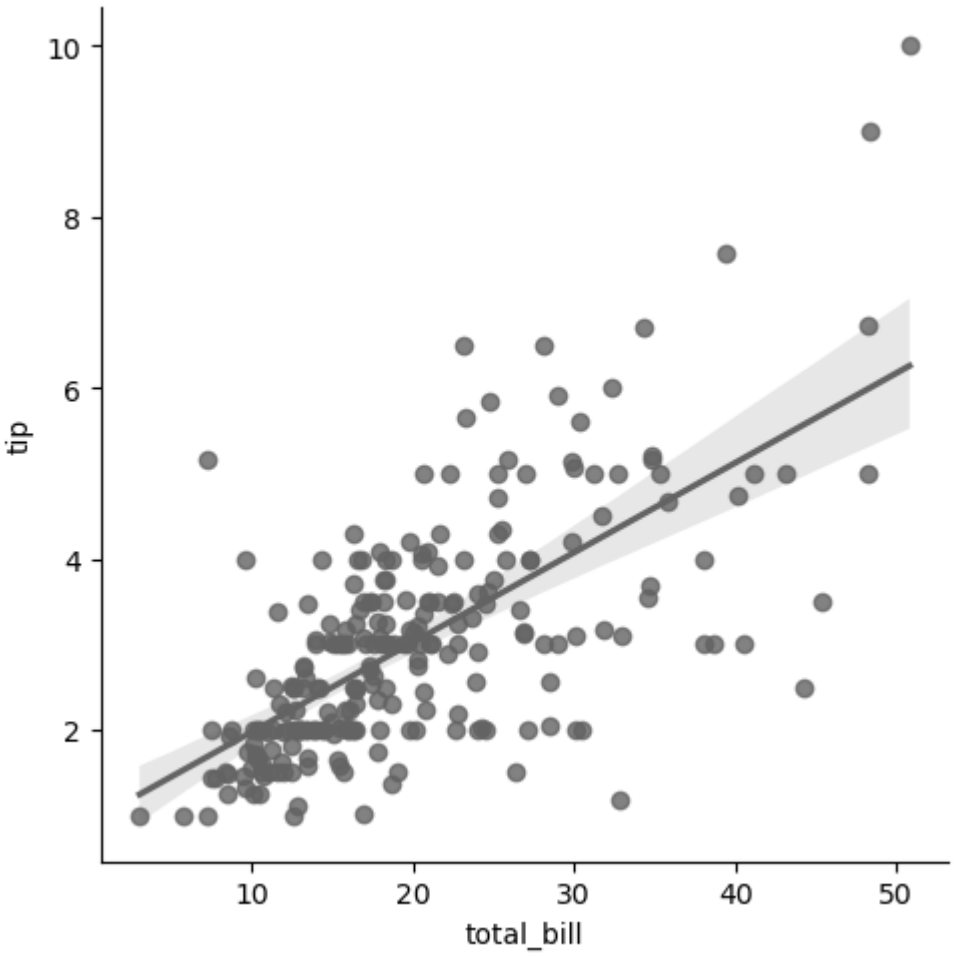
Regression Plots

The regression plots are primarily intended to add a visual guide that helps to emphasize patterns in a dataset during exploratory data analyses. Regression plots as the name suggests creates a regression line between two parameters and helps to visualize their linear relationships.

Implot Chart

Implot() method can be understood as a function that basically creates a linear model plot. It creates a scatter plot with a linear fit on top of it

In [1]:
```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
# loading
data=pd.read_csv("tips.csv")
sns.lmplot(x='total_bill', y='tip', data=data)
plt.show()
```



Heatmap

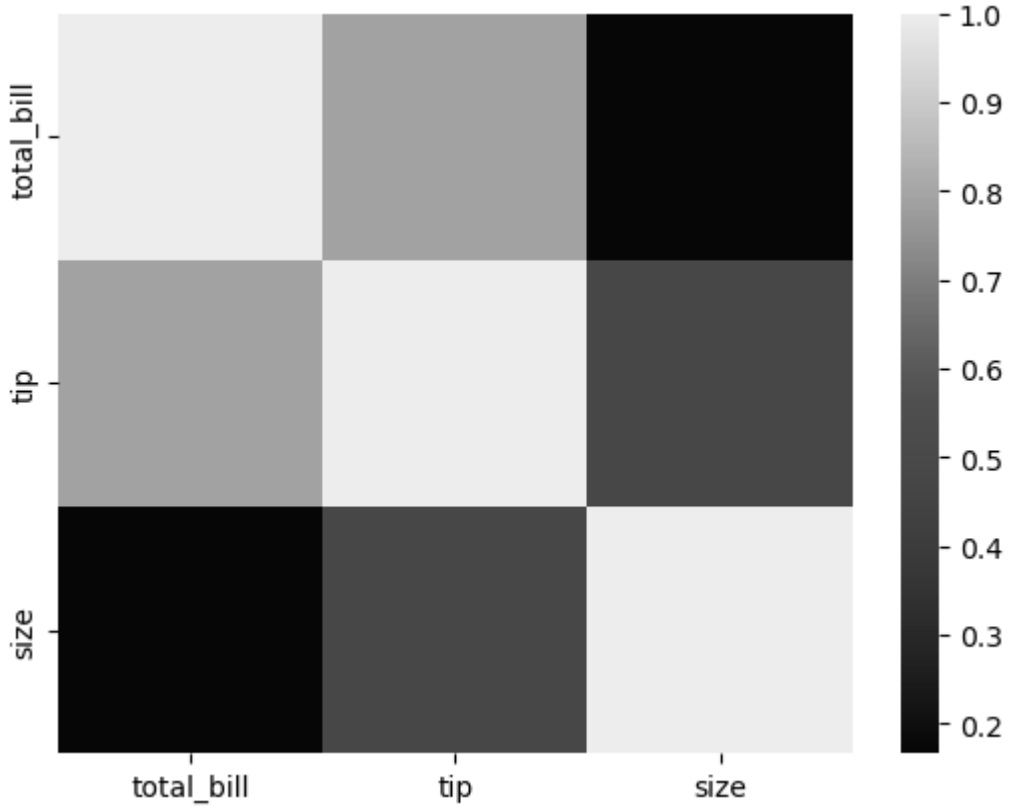
Heatmap is defined as a graphical representation of data using colors to visualize the value of the matrix. In this, to represent more common values or higher activities brighter colors basically reddish colors are used and to represent less common or activity values, darker colors are preferred. it can be plotted using the heatmap() function

correlation of r value

Interval correlation	Level of correlation
0.80 – 1.000	Very strong

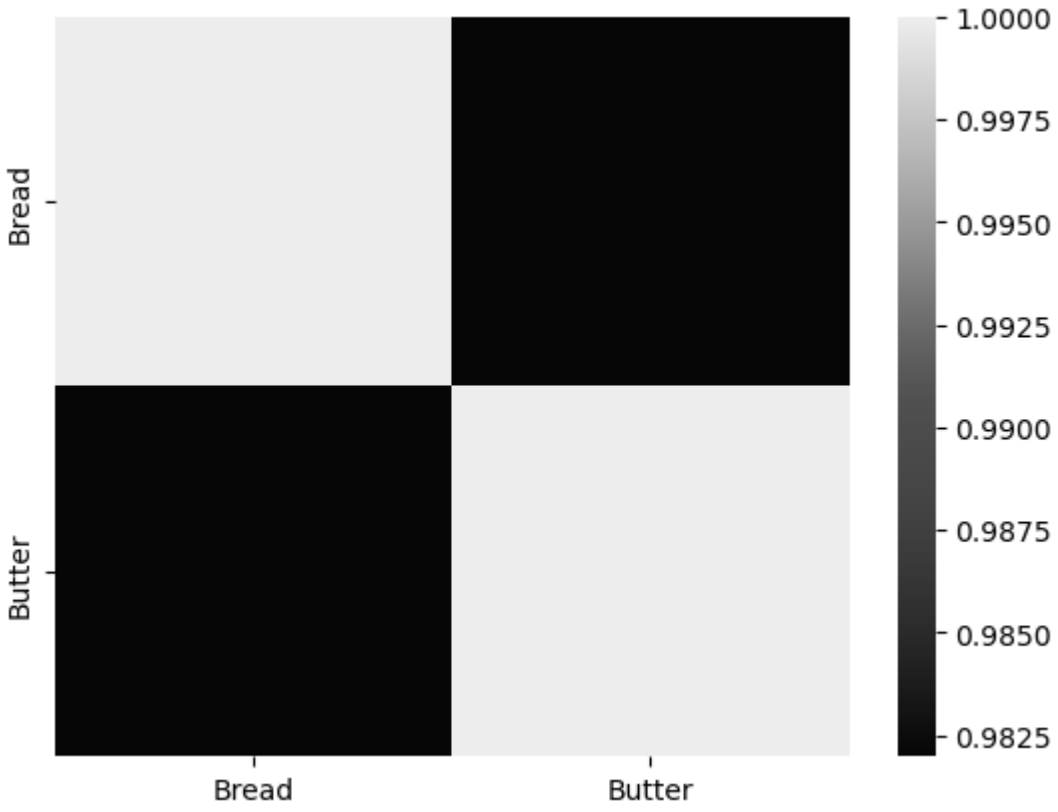
In [13]:
```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
# loading
data=pd.read_csv("tips.csv")
print(data.head())
data=data.head()
# data.describe()
tc=data.corr()
print(tc)
sns.heatmap(tc)
plt.show()
```

```
   total_bill   tip     sex smoker  day    time  size
0       16.99  1.01  Female     No  Sun  Dinner     2
1       10.34  1.66    Male     No  Sun  Dinner     3
2       21.01  3.50    Male     No  Sun  Dinner     3
3       23.68  3.31    Male     No  Sun  Dinner     2
4       24.59  3.61  Female     No  Sun  Dinner     4
            total_bill       tip      size
total_bill    1.000000  0.791002  0.166315
tip           0.791002  1.000000  0.475753
size          0.166315  0.475753  1.000000
```



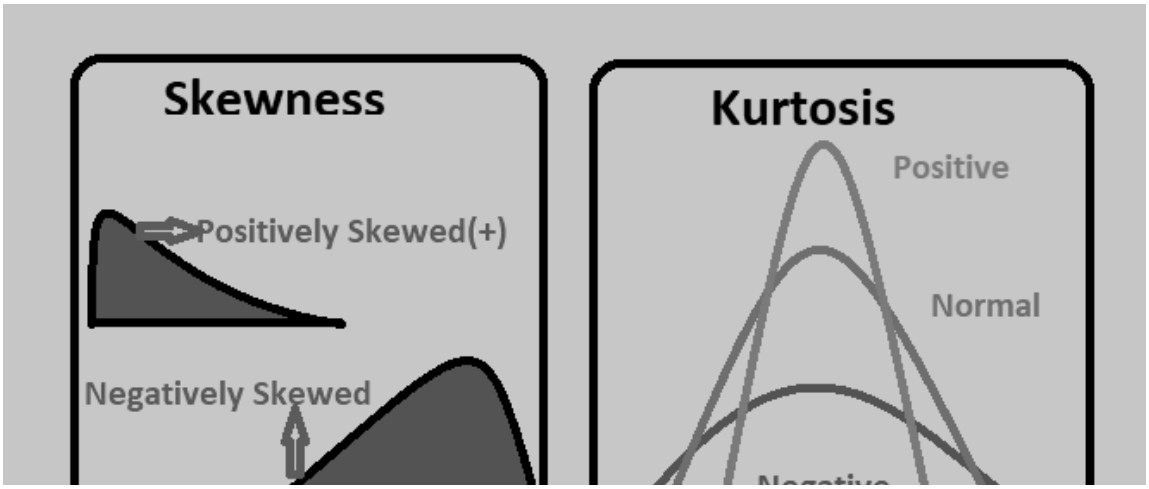
In [25]:
```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

data={'Bread':[40,60,20],
     'Butter':[4000,8000,2000]}
data=pd.DataFrame(data)
print(data)
tc=data.corr()
print(tc)
sns.heatmap(tc)
plt.show()
```

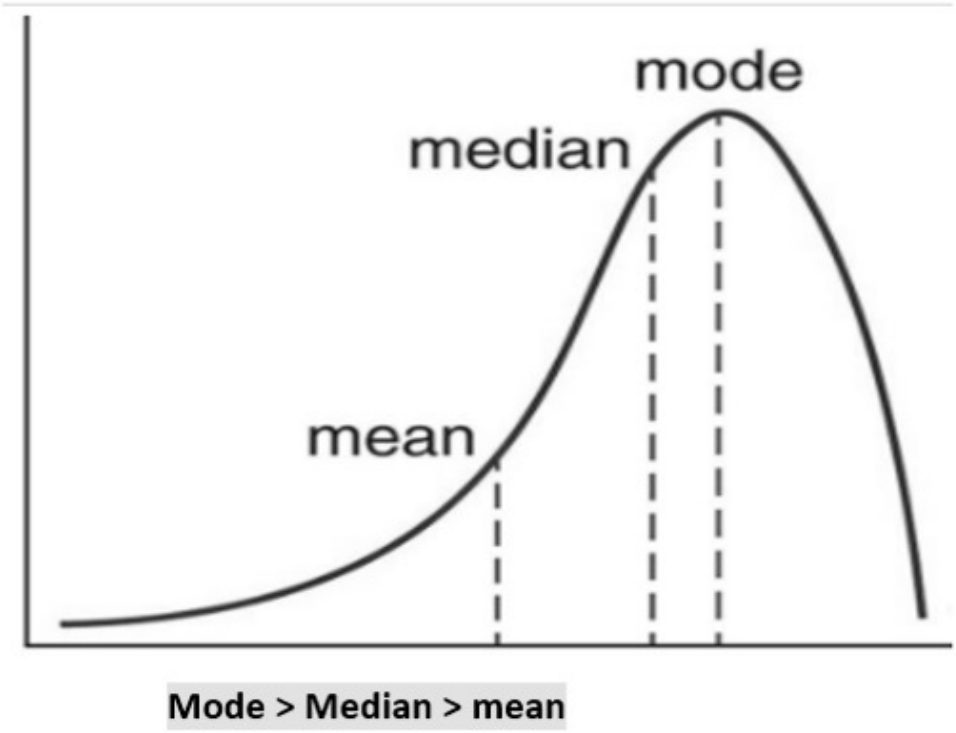
```
   Bread  Butter
0     40    4000
1     60    8000
2     20    2000
           Bread    Butter
Bread   1.000000  0.981981
Butter  0.981981  1.000000
```



Kurtosis

is a measure of the combined weight of a distribution's tails relative to the center of the distribution curve (the mean).





Skewness show Negative and Positive

0=normal, >0 Left Tail, <0 Right Tail

In [33]:
```
import pandas as pd
df=pd.read_csv("DataForStatistics.csv")
df
```

Out[33]:

	Gender	Age
0	Male	25
1	Male	27
2	Female	30
3	Female	26
4	Male	28
5	Male	29
6	Female	31
7	Male	32
8	Male	27

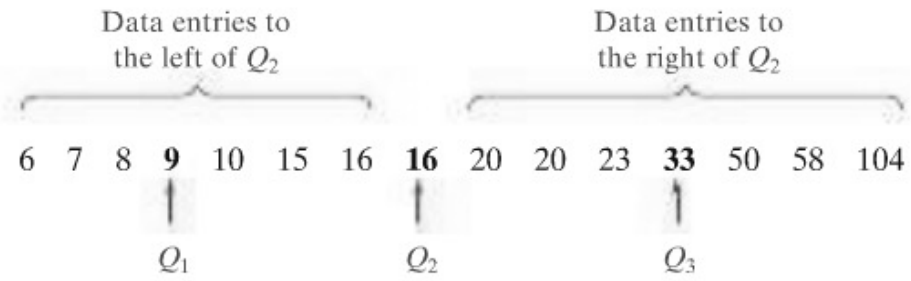
Finding the Quartiles of a Data Set

The number of nuclear power plants in the top 15 nuclear power-producing countries in the world are listed. Find the first, second, and third quartiles of the data set. What do you observe? *(Source: International Atomic Energy Agency)*

7 20 16 6 58 9 20 50 23 33 8 10 15 16 104

Solution

First, order the data set and find the median Q_2. The first quartile Q_1 is the median of the data entries to the left of Q_2. The third quartile Q_3 is the median of the data entries to the right of Q_2.

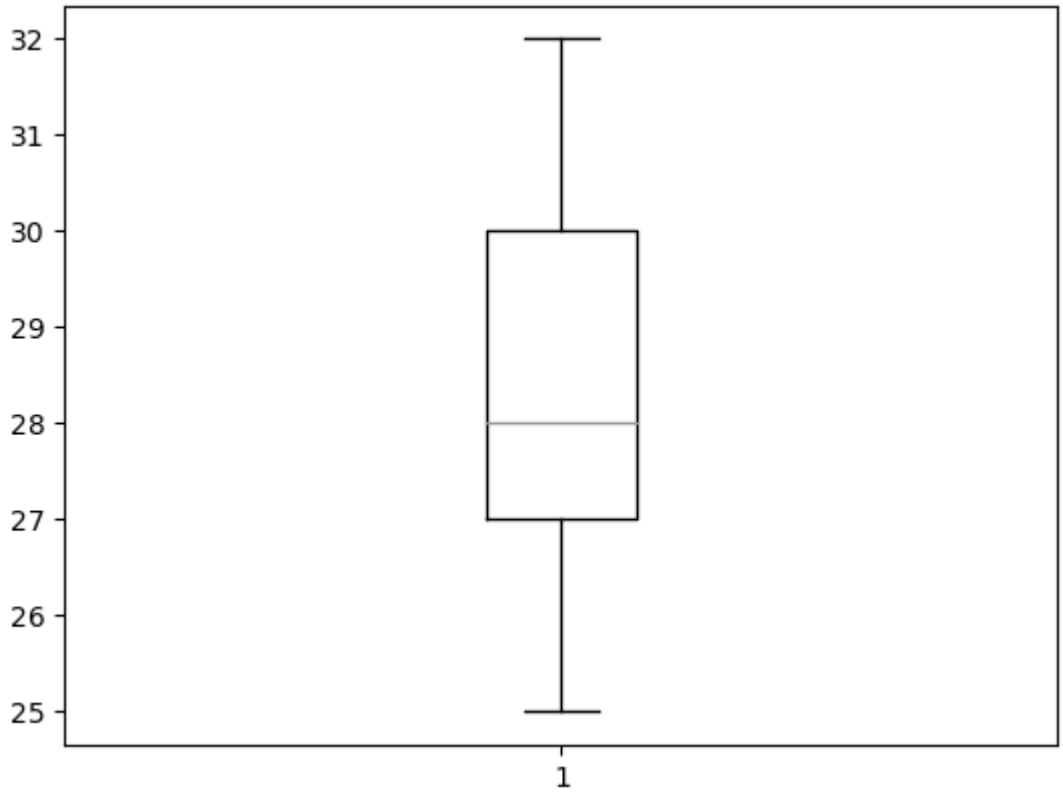


Interquartile range (IQR) Quartiles describe the spread of data by breaking into quarters. The median exactly divides the data into two parts. Q1(Lower quartile) is the middle value in the first half of the sorted dataset. Q2– is the median value Q3 (Upper quartile) is the middle value in the second half of the sorted dataset The interquartile range is the difference between the 75th percentile(Q3) and the 25th percentile(Q1). 50% of data fall within this range

In [34]:
```
Q1=df["Age"].quantile(0.25)   #it will find out the mean of First Quater
print("Quater One ",Q1)
Q2=df["Age"].quantile(0.50)
print("Quater Two ",Q2)
Q3=df["Age"].quantile(0.75)
print("Quater Three ",Q3)
IQR=Q3-Q1
print("IQR ",IQR)
```

```
Quater One  27.0
Quater Two  28.0
Quater Three  30.0
IQR  3.0
```

In [39]:
```
import matplotlib.pyplot as plt
plt.boxplot(df['Age'])
plt.show()
```



#Percentile

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

data3=np.array([14,20,22,25,40,80,90,70,60,80,60])
q3,q1=np.percentile(data3,[75,25])
print("Qtr1 ",q1)
print("Qtr3 ",q3)
iqr=q3-q1
iqr
```

```
Qtr1  23.5
Qtr3  75.0
```

Out[1]:
```
51.5
```

Percentile

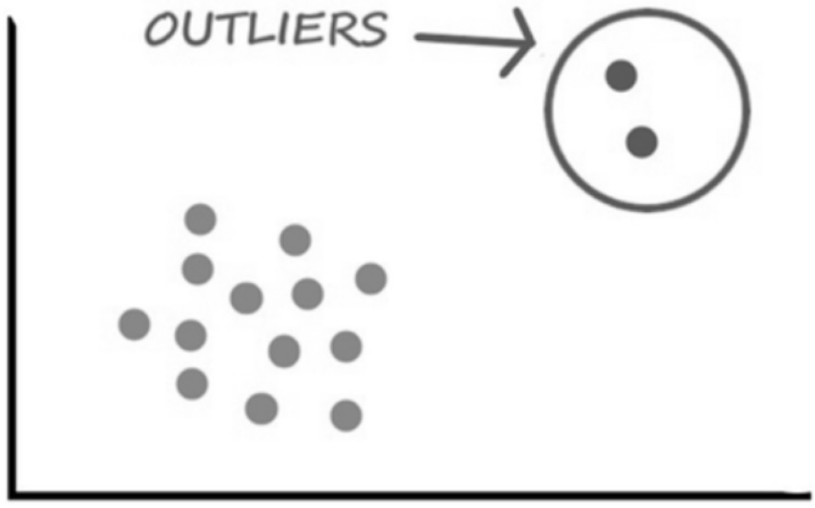
In [6]:
```
import numpy
ages=[5,31,43,48,50,41,7,11,15,39,80,82,32,2,8,6,25,36,27,61,31]
x=numpy.percentile(ages,75)
x
#mean 2 se 43 tak 75% person aate hai
#or 43 se jyada age ke 25% aate hai
```

Out[6]:
```
43.0
```

In [11]:
```
print(np.sort(ages))
```

```
[ 2  5  6  7  8 11 15 25 27 31 31 32 36 39 41 43 48 50 61 80 82]
```

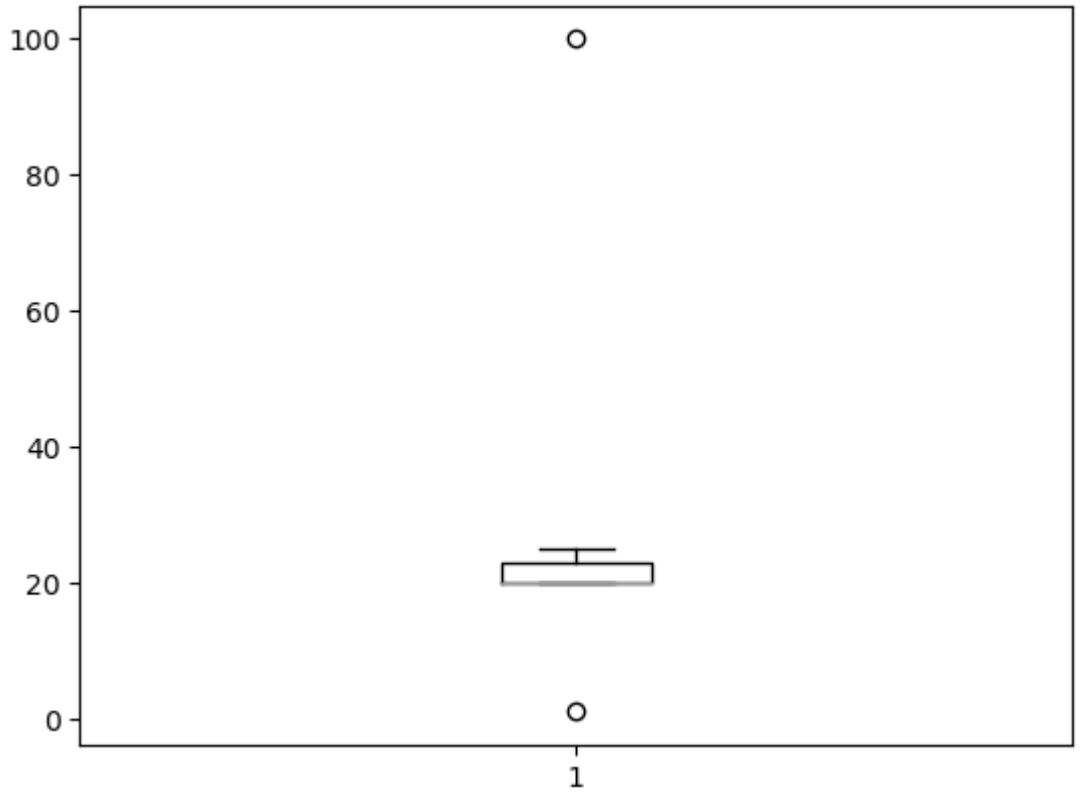
Outlier



In [30]:
```
import numpy as np

data=[1,20,20,20,21,25,100]

def dectect_outlier(data):
    q1,q3=np.percentile(sorted(data),[25,75])
    iqr=q3-q1

    #find lower and upper bounds
    lower_bound=q1-(1.5*iqr)
    upper_bound=q3+(1.5*iqr)

    outlier=[x for x in data if x<=lower_bound or x>=upper_bound]

    return outlier
dat=dectect_outlier(data)
print(dat)
```

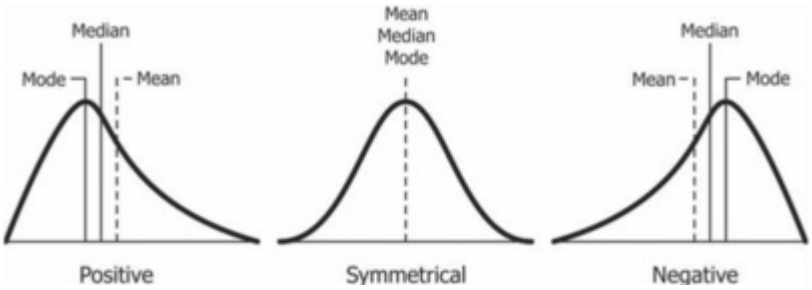
```
[1, 100]
```

In [32]:
```
import matplotlib.pyplot as plt

plt.boxplot(data)
plt.show()
```



Skewness

$$\text{Skewness} = \frac{3\,(Mean - Median)}{Standard\ Deviation}$$

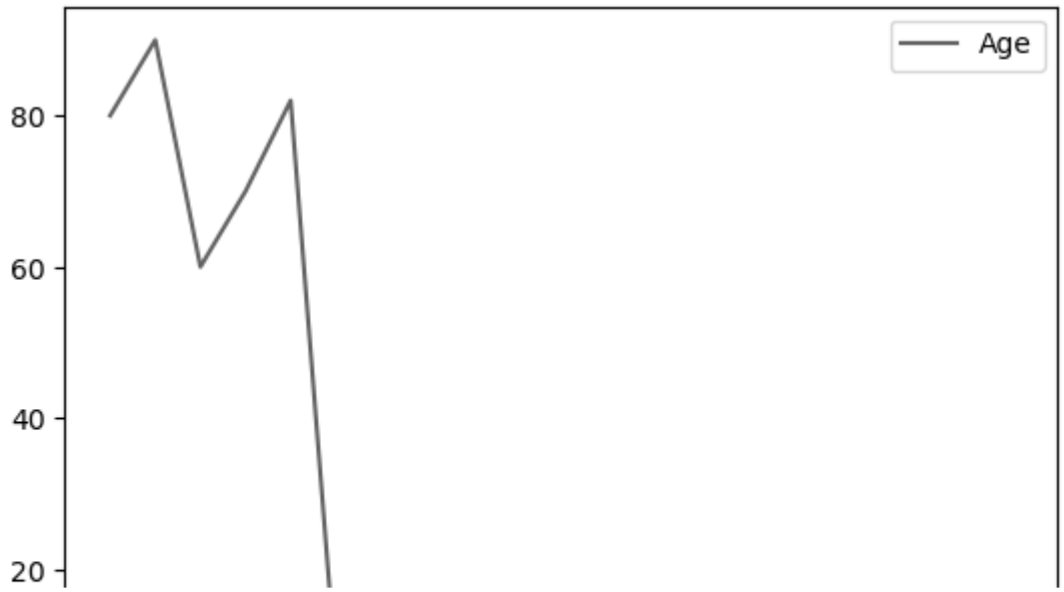


In [53]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
data=np.array([80,90,60,70,82,7,9,2,5,2,1,3,7,10,3,4,6,7,8,9,2])
df=pd.DataFrame(data)
df.columns=["Age"]
print(df)
```

```
    Age
0    80
1    90
2    60
3    70
4    82
5     7
6     9
7     2
8     5
9     2
10    1
11    3
12    7
13   10
14    3
15    4
16    6
17    7
18    8
19    9
20    2
```

In [52]:
```
skw=df['Age'].skew()
print(skw)
df.plot()
plt.show()
```

```
1.3994219493683466
```



In [ ]:

Kurtosis

is a statistical measure used to describe the distribution of observed data around the mean. Kurtosis is a statistical measure used to describe the degree to which scores cluster in the tails or the peak of a frequency distribution.

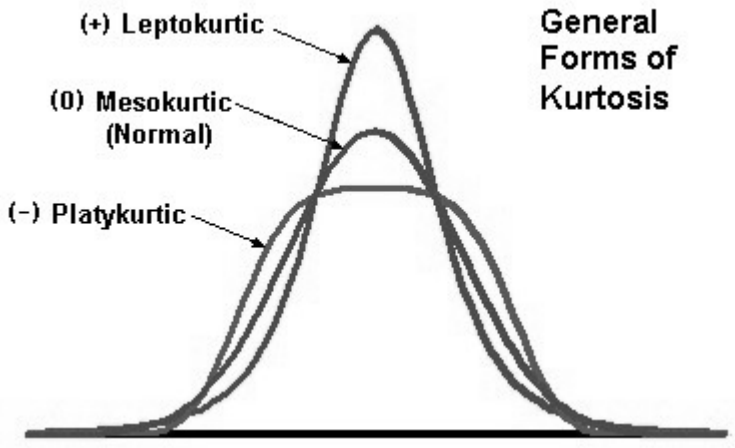
$$\text{Kurtosis} = \frac{\sum_{i=1}^{N} \frac{\left(X_i - \bar{X}\right)}{N}}{s^4}$$

where,

$\bar{X}$ is the mean,

s is the standard deviation

and N is the sample size

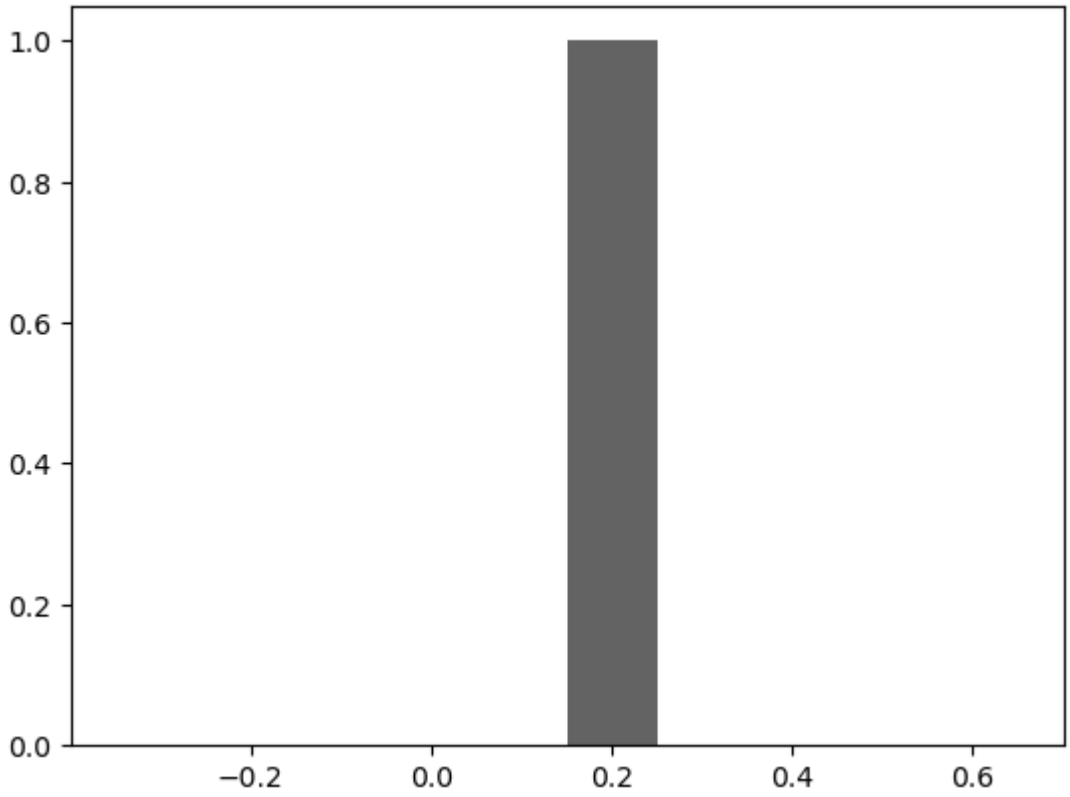


Kurtosis Excess Kurtosis Leptokurtic >3 >0 Platykurtic <3 <0 Mesokurtic =3 =0

In [42]:
```
krto=df['Age'].kurtosis()
print(krto)
```

```
0.15058344532268908
```

In [45]:
```
plt.hist(krto)
plt.show()
```



https://docs.scipy.org/doc/scipy/reference/gene
(https://docs.scipy.org/doc/scipy/reference/gen

Hypothesis Testing Python¶

Null hypothesis and alternative hypothesis are the two different methods of hypothesis testing. The premise for a null hypothesis is an occurrence (also called the ground truth). An alternative hypothesis is a presumption that disputes the primary hypothesis.

Students who eat breakfast will perform better on a math exam than students who do not eat breakfast

Data on the hypothesis should be gathered and examined to determine whether or not H0 may be accepted. When doing that, there's a chance that the following things will occur:

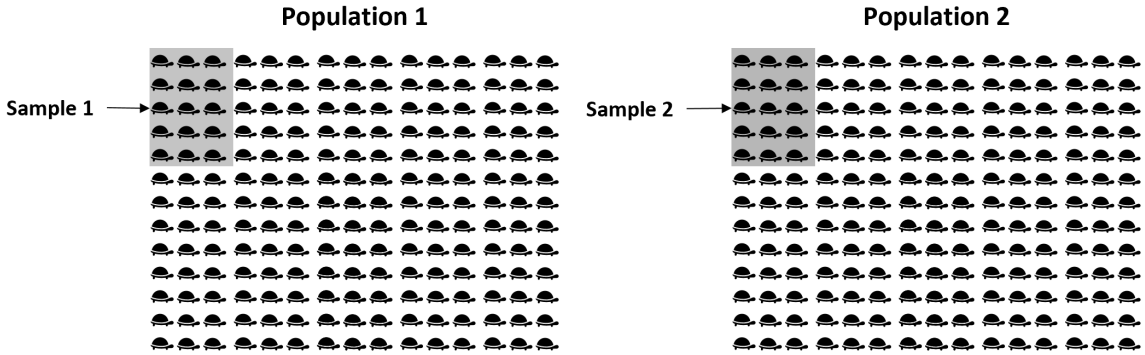
H0 is acknowledged as the ground truth since it is true. Since H0 is denied and H1 is accepted, the ground truth is incorrect.

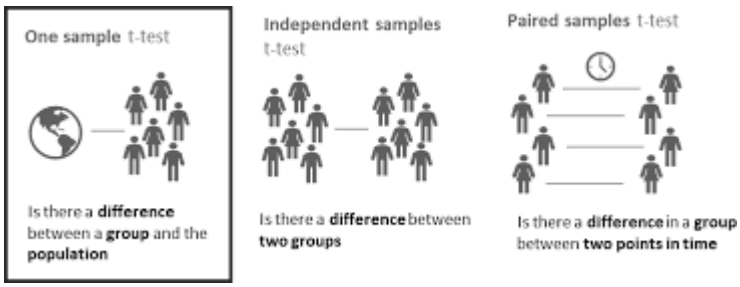
T- Test¶

When comparing the mean values of two samples that specific characteristics may connect, a t-test is performed to see if there exists a substantial difference. It is typically employed when data sets, such as those obtained from tossing a coin 100 times and stored as results, would exhibit a normal distribution. It could have unknown variances. The t-test is a method for evaluating hypotheses that allows you to assess a population-applicable assumption.

Assumptions

Each sample's data is randomly and uniformly distributed (iid). Each sample's data have a normal distribution. Every sample's data share the same variance. T-tests are of two types: 1. one-sampled t-test and 2. two-sampled t-test.





In [2]:
```
# Python program to implement T-Test on a sample of ages

# Importing the required libraries
from scipy.stats import ttest_1samp
import numpy as np

# Creating a sample of ages
ages = [45, 89, 23, 46, 12, 69, 45, 24, 34, 67]
print(ages)

# Calculating the mean of the sample
mean = np.mean(ages)
print(mean)

# Performing the T-Test
t_test, p_val = ttest_1samp(ages, 30)
print("P-value is: ", p_val)

# # taking the threshold value as 0.05 or 5%
if p_val < 0.05:
    print(" We can reject the null hypothesis")
else:
    print("We can accept the null hypothesis")
```

```
[45, 89, 23, 46, 12, 69, 45, 24, 34, 67]
45.4
P-value is:  0.07179988272763554
We can accept the null hypothesis
```

Two sampled t-test:-

To ascertain if there exists any statistical confirmation that the related population means are statistically substantially distinct, the Independent Samples T-Test, also known as the 2-sample T-test, analyses the mean values of two independent samples. The Independent Samples T-Test is also a parametric test. The Independent t Test is another name for this test.

For instance, can we check if there is a correlation between the two data groups defined in the code below?

In [14]:
```
# Python program to implement Independent T-Test on the two independent sam

# Importing the required libraries
from scipy.stats import ttest_ind
import numpy as np

# Creating the data groups
data_group1 = np.array([12, 18, 12, 13, 15, 1, 7,
                        20, 21, 25, 19, 31, 21, 17,
                        17, 15, 19, 15, 12, 15])
data_group2 = np.array([23, 22, 24, 25, 21, 26, 21,
                        21, 25, 30, 24, 21, 23, 19,
                        14, 18, 14, 12, 19, 15])

# Calculating the mean of the two data groups
mean1 = np.mean(data_group1)
mean2 = np.mean(data_group2)

# Print mean values
print("Data group 1 mean value:", mean1)
print("Data group 2 mean value:", mean2)

# Calculating standard deviation
std1 = np.std(data_group1)
std2 = np.std(data_group2)

# Printing standard deviation values
print("Data group 1 std value:", std1)
print("Data group 2 std value:", std2)

# Implementing the t-test
t_test,p_val = ttest_ind(data_group1, data_group2)
print("The P-value is: ", p_val)

# taking the threshold value as 0.05 or 5%
if p_val < 0.05:
    print("We can reject the null hypothesis")
else:
    print("We can accept the null hypothesis")
```

```
Data group 1 mean value: 16.25
Data group 2 mean value: 20.85
Data group 1 std value: 6.171507109288622
Data group 2 std value: 4.452808102759426
The P-value is:  0.012117171124028792
We can reject the null hypothesis
```

In [ ]:
```
```

Paired sampled t-test:-

The dependent samples t-test is another name for the paired samples t-test. A substantial difference between two related variables is tested using a univariate test. An illustration of this would be taking a person's blood pressure before and after a specific medication, condition, or period.

In [3]:
```
# Python program to implement Paired Sample T-Test on the two dependent sam

# Importing the required libraries
import pandas as pd
from scipy import stats

# Creating two samples
sample1 = [29, 30, 33, 41, 38, 36,
       35, 31, 29, 30]
sample2 = [31, 32, 33, 39, 30, 33,
        30, 28, 29, 31]

# Performing paired sample t-test
t_test, p_val = stats.ttest_rel(sample1, sample2)
print("The P-value of the test is: ", p_val)

# taking the threshold value as 0.05 or 5%
if p_val < 0.05:
    print("We can reject the null hypothesis")
else:
    print("We can accept the null hypothesis")
```

```
The P-value of the test is:  0.15266056244408904
We can accept the null hypothesis
```

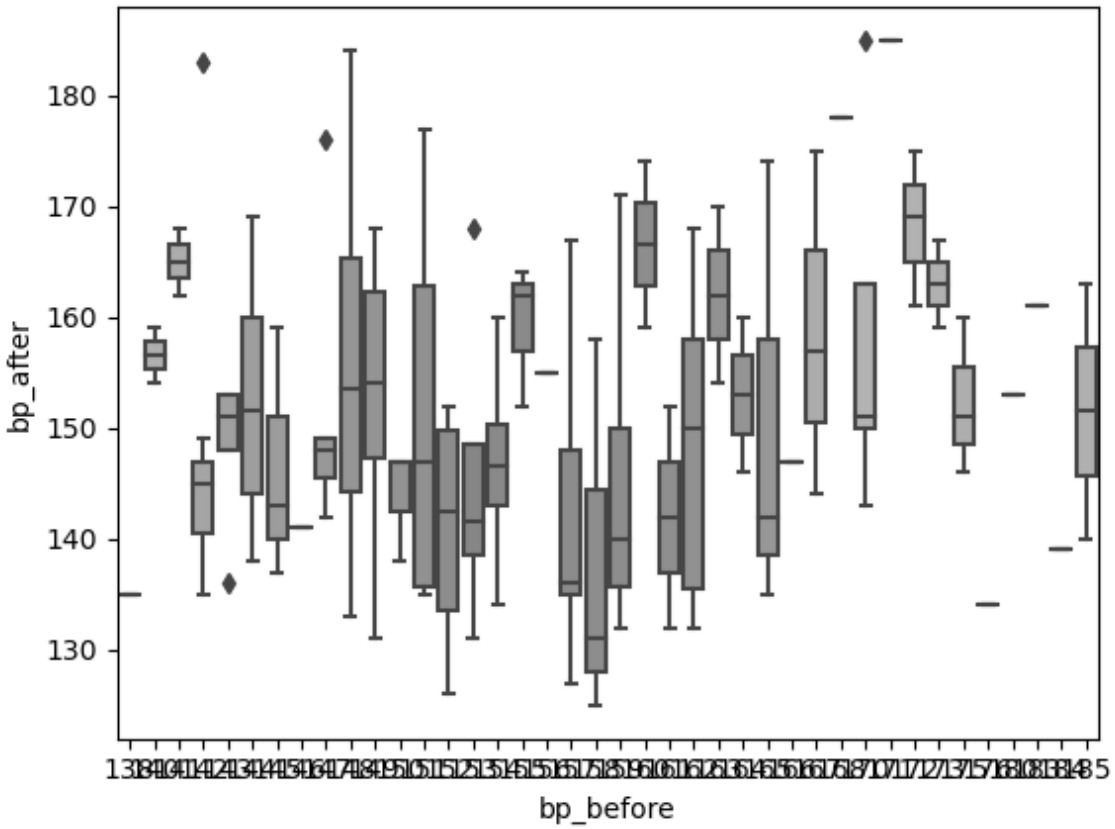
Bood Pessure Analysis After Meal and Before Meal

In [10]:
```
import pandas as pd

df = pd.read_csv("blood_pressure.csv")

df[['bp_before','bp_after']].describe()
```

Out[10]:

	bp_before	bp_after
count	120.000000	120.000000
mean	156.450000	151.358333
std	11.389845	14.177622
min	138.000000	125.000000
25%	147.000000	140.750000
50%	154.500000	149.500000
75%	164.000000	161.000000
max	185.000000	185.000000

In [13]:
```
import matplotlib.pyplot as plt
import seaborn as sns
sns.boxplot(x='bp_before', y='bp_after', data=df)
plt.show()
```



In [6]:
```
from scipy import stats
stats.shapiro(df['bp_before'])
```

Out[6]:
```
ShapiroResult(statistic=0.9547787308692932, pvalue=0.0004928423441015184)
```

In [8]:
```
stats.shapiro(df['bp_after'])
```

Out[8]:
```
ShapiroResult(statistic=0.9740639328956604, pvalue=0.020227791741490364)
```

In [9]:
```
stats.ttest_rel(df['bp_before'], df['bp_after'])
```

Out[9]:
```
Ttest_relResult(statistic=3.3371870510833657, pvalue=0.0011291791464484082
3)
```

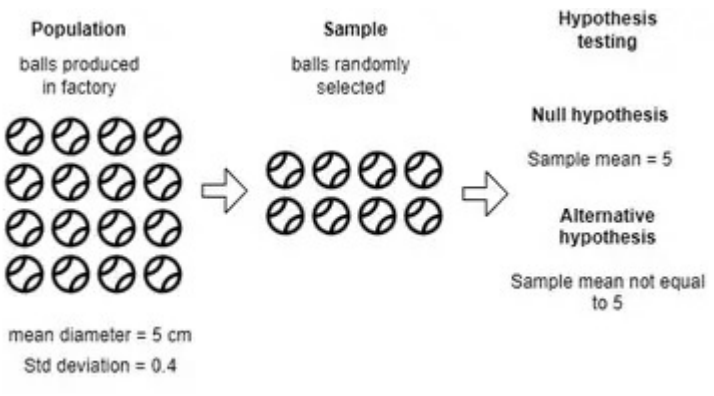
Z-Test

In the z-test, which also functions as a hypothesis test, the z-statistic has a normal distribution. As the central limit theorem states, observations are assumed to be roughly normally distributed as sample size increases. Hence the z-test is most effective for samples bigger than 30.

The null and alternate hypotheses, alpha, and z-score values must all be specified when doing a z-test. The test statistic must then be computed, followed by a statement of the findings and conclusions. A z-statistic, also known as a z-score, measures how many

standard deviations a score resulting from a z-test is either above or under the mean population.

One-sample location tests, two-sample location tests, paired difference tests, and maximum likelihood estimates are a few tests that may be performed as z-tests.



One-Sample Z-Test:-

Assume a trader wants to determine if a stock's daily mean gain is more than 3%. The average is defined for a straightforward arbitrary sample of 50 results, which is 2%. Assume that the profits' standard deviation is 1.5 percent. Therefore, the null hypothesis in this situation is whenever the mean is equal to 3%.

On the other hand, the alternate hypothesis is if the average return is more or lower than 3%. We can assume a two-tailed z-test is used to choose an alpha value of 0.05%. As a result, the alpha has a significance level of 1.96 or -1.96, and 0.025% of the observations are distributed evenly across the two tails. We will disprove the null hypothesis if z is larger than 1.96 or smaller than -1.96.

The answer for z is determined by deducting the observed mean of the samples from the value of the mean daily return chosen for the test, in this instance, 1%. After that, multiply the result by the square root of the total number of observations in the sample divided by the standard deviation.

In [16]:
```
# Python program to implement One Sample Z-Test

# Importing the required libraries
import pandas as pd
from scipy import stats
from statsmodels.stats import weightstats as stests

# Creating a dataset
data = [89, 93, 95, 93, 97, 98, 96, 99, 93, 97,
        110, 104, 119, 105, 104, 110, 110, 112, 115, 114]

# Performing the z-test
z_test ,p_val = stests.ztest(data, x2 = None, value = 160)
print(p_val)
print(z_test)

# taking the threshold value as 0.05 or 5%
if p_val < 0.05:
    print("We can reject the null hypothesis")
else:
    print("We can accept the null hypothesis")
```

```
2.417334226169332e-186
-29.11357036941381
We can reject the null hypothesis
```

scipy.stats.shapiro

scipy.stats.shapiro(x)[source] Perform the Shapiro-Wilk test for normality.

The Shapiro-Wilk test tests the null hypothesis that the data was drawn from a normal distribution.

In [2]:
```
import numpy as np
x = np.array([148, 154, 158, 160, 161, 162, 166, 170, 182, 195, 236])
x
```

Out[2]:
```
array([148, 154, 158, 160, 161, 162, 166, 170, 182, 195, 236])
```

In [3]:
```
from scipy import stats
res = stats.shapiro(x)
res.statistic
```

Out[3]:
```
0.7888146638870239
```

ANOVA (F-TEST)¶

The t-test is effective for comparing two groups; however, there are situations when we wish to examine more than two datasets at once. For instance, we would need to analyze the averages of each class or group to determine whether voter age varied depending on a categorical variable like race. We could do a separate t-test for each combination of groups,

which raises the possibility of false positive results. Analysis of variance, or ANOVA, is a statistical inference technique that permits the parallel comparison of the distribution of several data groups.

One-Way F-test(Anova):-

Depending on the average similarity and f-score of two or more data groups, it may be determined if they are similar or not.

In [4]:
```
# Python program to implement One-Way f-test

# Importing the required libraries
import scipy.stats

# Creating sample data
data1 = [0.0842, 0.0368, 0.0847, 0.0935, 0.0376, 0.0963, 0.0684,
             0.0758, 0.0854, 0.0855]
data2 = [0.0785, 0.0845, 0.0758, 0.0853, 0.0946, 0.0785, 0.0853,
           0.0685]
data3 = [0.0864, 0.2522, 0.0894, 0.2724, 0.0853, 0.1367, 0.853]

# Performing the F-Test
f_test, p_val = scipy.stats.f_oneway(data1, data2, data3)
print("p-value is: ", p_val)

# taking the threshold value as 0.05 or 5%
if p_val < 0.05:
    print(" We can reject the null hypothesis")
else:
    print("We can accept the null hypothesis")
```

```
p-value is:  0.04043792126789144
 We can reject the null hypothesis
```

Two-Way F-test:-

When there are two independent variables and two or more groups, we utilize the two-way F-test, a generalization of the one-way F-test. We cannot find the dominant factor using the 2-way F-test. If the personalized significance value needs to be validated, post-hoc testing is required.

In [6]:
```
# Python program to perform 2-way F-test

# Importing the required modules
import pandas as pd
import numpy as np
import statsmodels.api as sm
from statsmodels.formula.api import ols

# Create a dataframe
df = pd.DataFrame({'Frequency_fertilizers': np.repeat(['daily', 'weekly'],
                           'Frequency_Watering': np.repeat(['daily', 'weekly
                           'Crop_height': [12, 14, 15, 12, 17, 21, 19, 8,
                                           5, 12, 19, 23, 23, 14, 16, 21,
                                           25, 16, 17, 13, 24, 9, 19, 4,
                                           12, 14, 15, 12, 17, 15, 18, 14]})
# df
f_test = ols('Crop_height ~ C(Frequency_fertilizers) * C(Frequency_Watering
             C(Frequency_fertilizers):C(Frequency_Watering)',
            data = df).fit()
result = sm.stats.anova_lm(f_test, type = 2)

# Printing the result
print(result)
```

```
                                            df      sum_sq    mean_s
q  \
C(Frequency_fertilizers)                   1.0    1.531250   1.53125
0
C(Frequency_Watering)                      1.0    3.604114   3.60411
4
C(Frequency_fertilizers):C(Frequency_Watering)   1.0    0.043700   0.04370
0
Residual                                  30.0  802.437500  26.74791
7

                                                  F    PR(>F)
C(Frequency_fertilizers)                   0.057247  0.812528
C(Frequency_Watering)                      0.134744  0.716140
C(Frequency_fertilizers):C(Frequency_Watering)  0.001634  0.968026
Residual                                        NaN       NaN
```

Second Example of Hypothesis Analysis from Kaggle

https://www.kaggle.com/code/bhagyashree12/anova-test-on-iris-dataset/notebook (https://www.kaggle.com/code/bhagyashree12/anova-test-on-iris-dataset/notebook)

Chi-Square Test¶

This test is used when two categorized variables are from the same population. Its purpose is to decide if the two elements are significantly associated.

For example, we may group people in an election campaign survey based on their preferred method of voting and gender (male or female) (Democratic, Republican, or Independent). To determine if gender affects voting choice, we may apply a chi-square test evaluating

independence.

Another Example of Kaggle.
https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.chisquare.html
(https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.chisquare.html)

In [7]:
```
# Python program to perform a chi-square test

# Importing the required modules
from scipy.stats import chi2_contingency

# defining our data
data = [[231, 256, 321],
        [245, 312, 213]]

# Performing chi-square test
test, p_val, dof, expected_val = chi2_contingency(data)

# interpreting the p-value
alpha = 0.05
print("The p-value of our test is " + str(p_val))

# Checking the hypothesis
if p_val <= alpha:
    print('We can reject the null hypothesis')
else:
    print('We can accept the null hypothesis')
```

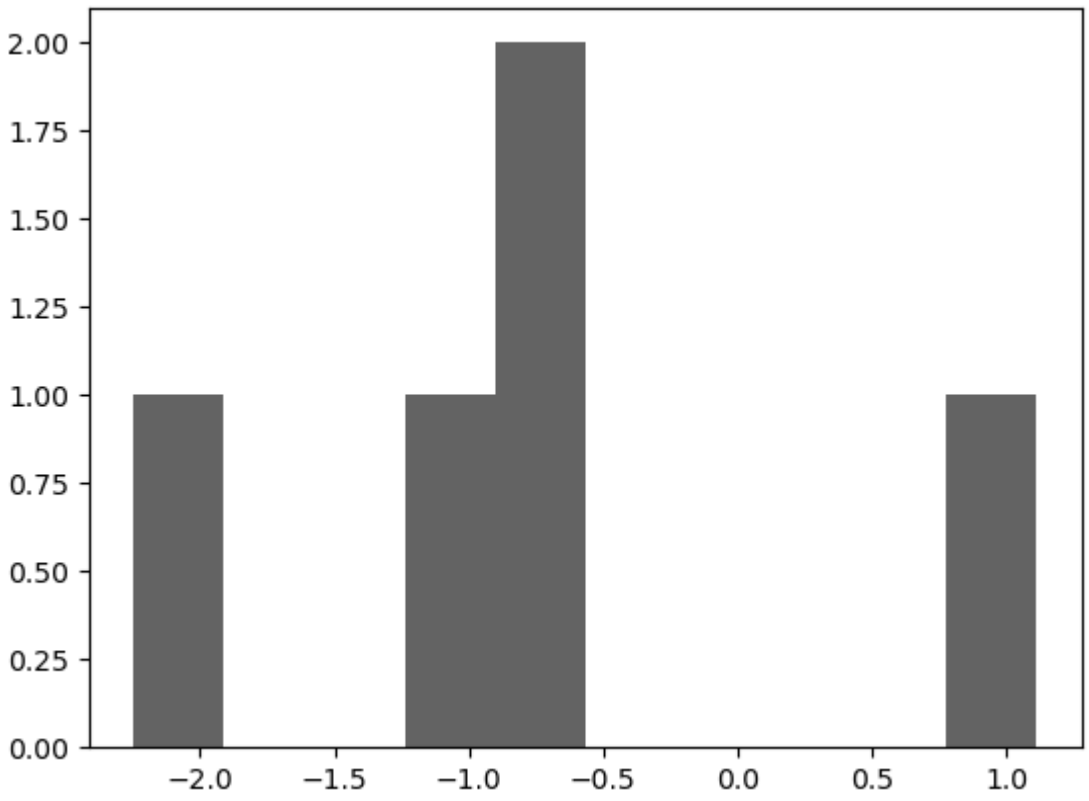
```
The p-value of our test is 1.4585823594475804e-06
We can reject the null hypothesis
```

Normal Data Distribution

In probability theory this kind of data distribution is known as the normal data distribution, or the Gaussian data distribution, after the mathematician Carl Friedrich Gauss who came up with the formula of this data distribution.

In [18]:
```
import numpy
import matplotlib.pyplot as plt

x = numpy.random.normal(0, 1,5)
print(x)
plt.hist(x, 10)
plt.show()
```

```
[ 1.11217091 -0.72295148 -1.09895512 -0.57774395 -2.24725491]
```

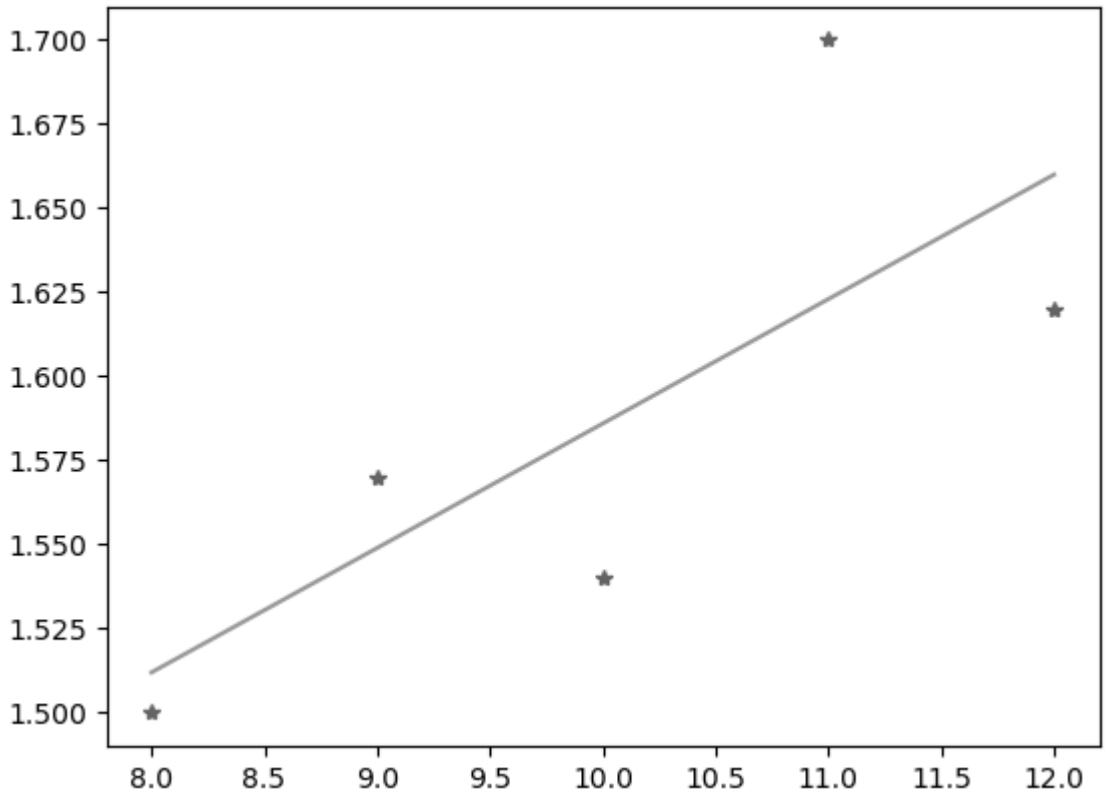


Linear Regression¶

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

In [10]:
```
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,True)
print(k," ",d)
y_pred = k*x + d
print(y_pred)
plt.plot(x, y, '*')
plt.plot(x, y_pred)
plt.show()
```

```
0.037000000000000026   1.215999999999999
[1.512 1.549 1.586 1.623 1.66 ]
```

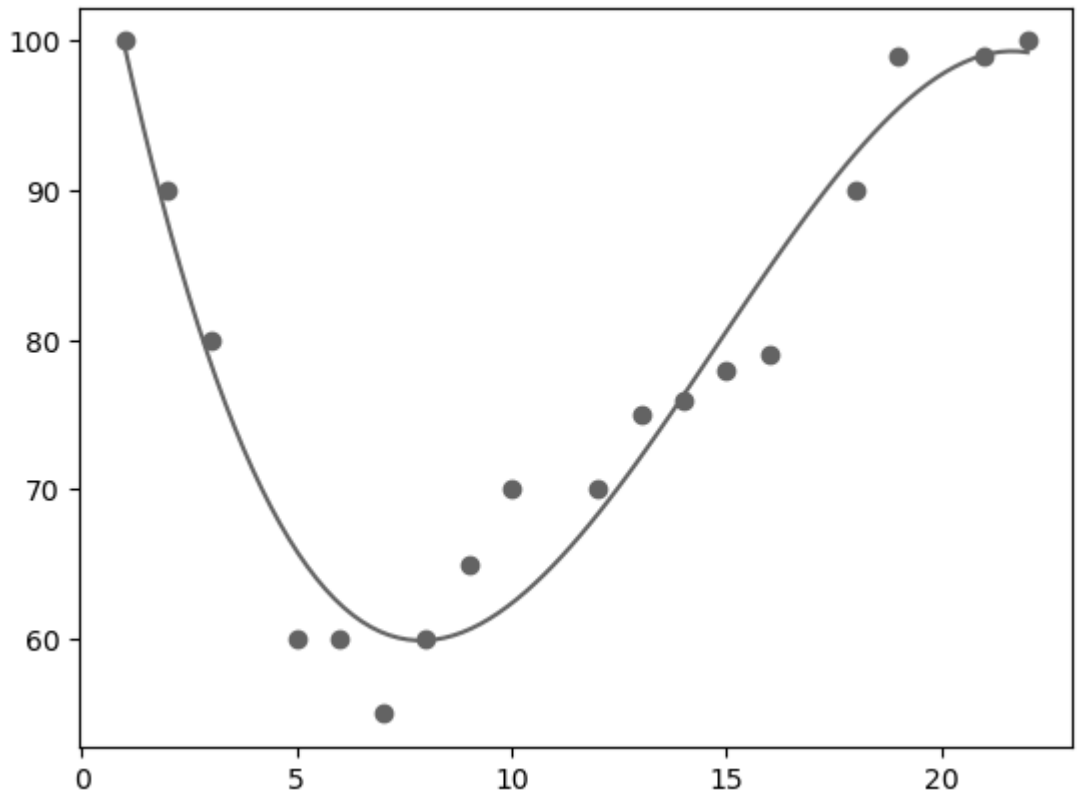


Polynomial Regression¶

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 [17]:
```
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()
```



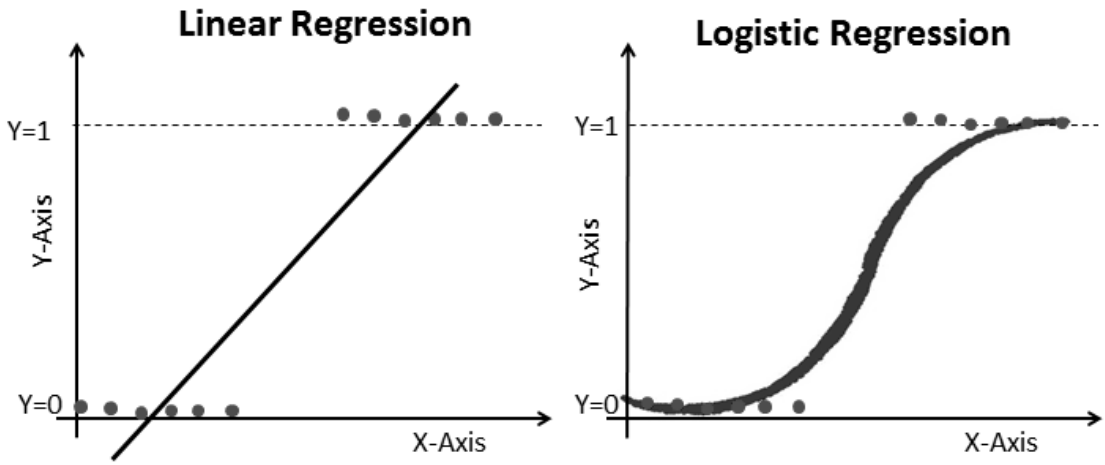
Logistic Regression

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

Logistic regression analysis is used to examine the association of (categorical or continuous) independent variable(s) with one dichotomous dependent variable. it work on both data categorical or continuous

In [34]:
```
import numpy as np
from sklearn.datasets import load_iris

iris = load_iris()
print('The data matrix:\n',iris['data'])
print('The classification target:\n',iris['target'])
print('The names of the dataset columns:\n',iris['feature_names'])
print('The names of target classes:\n',iris['target_names'])
print('The full description of the dataset:\n',iris['DESCR'])
print('The path to the location of the data:\n',iris['filename'])
```

```
The data matrix:
 [[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]
```



In [18]:

```
import numpy as np
from sklearn.datasets import load_iris
# 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
import matplotlib.pyplot as plt

data = load_iris()
#print(data)
# Splitting the independent and dependent variables
X = data.data
Y = data.target
print(X)
print(Y)
print("The size of the complete X dataset is: ", len(X))
print("The size of the complete Y dataset is: ", len(Y))
# # Creating an instance of the LogisticRegression class for implementing l
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, ra
print(X_train)
print(Y_train)
print(len(X_train))
print(len(Y_train))
print("------------")
print(X_test)
print(Y_test)
print(len(X_test))
print(len(Y_test))
# # # 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_sco


# Create a scatter plot
plt.scatter(log_reg.predict(X_test), Y_test)
plt.xlabel("Predicted Values")
plt.ylabel("Actual Values")
plt.title("Scatter Plot of Predicted vs. Actual Values")
plt.show()
```

```
[[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]
```

Train And Test Data With SkLearn

In [19]:
```
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,
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

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

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.

Why Combine Principal Component Analysis(PCA) and K-means Clustering?

There are varying reasons for using a dimensionality reduction step such as PCA prior to data segmentation. Chief among them? By reducing the number of features, we're improving the performance of our algorithm. On top of that, by decreasing the number of features the noise is also reduced.

It is used to find interrelations between variables in the data.

It is used to interpret and visualize data.

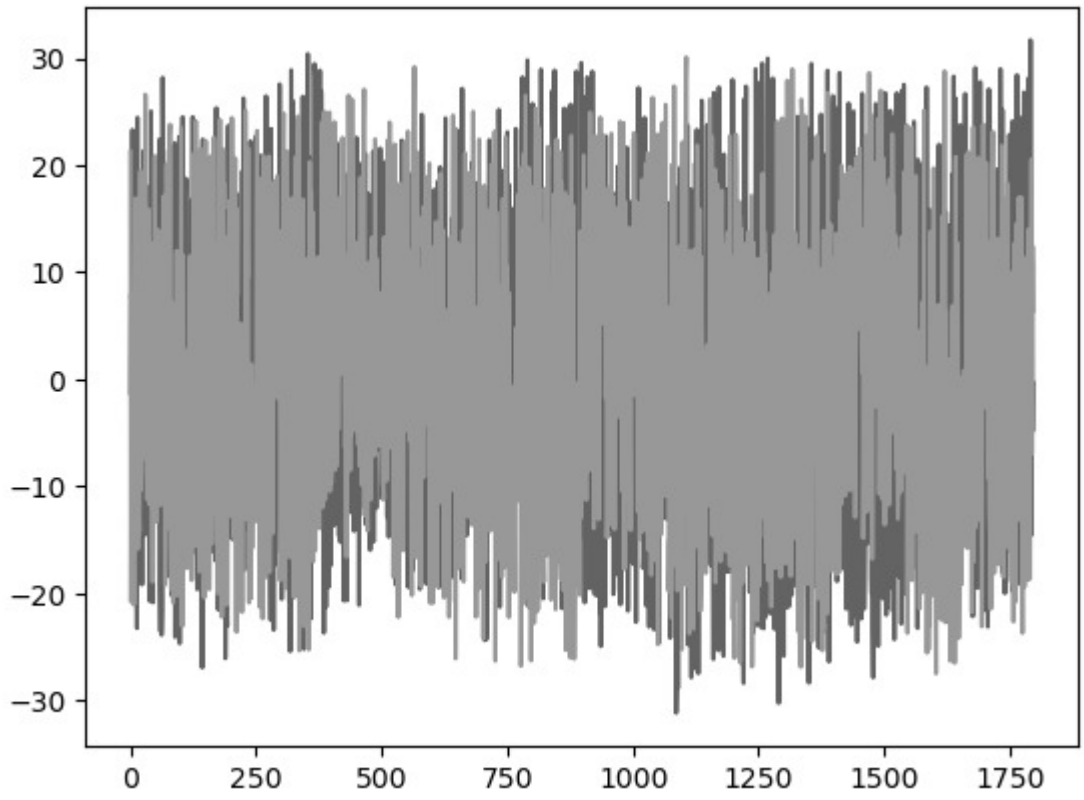
The number of variables is decreasing which makes further analysis simpler.

It's often used to visualize genetic distance and relatedness between populations.

If our sole intention of doing PCA is for data visualization, the best number of components is 2 or 3.

In [23]:
```
from sklearn.datasets import load_digits
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from matplotlib import pyplot as plt
import numpy as np

data=load_digits().data
print(data)
print(data.shape)
pca=PCA(2)
print(pca)
#Transpose the data
df=pca.fit_transform(data)
df.shape
print(df.shape)
plt.plot(df)
plt.show()
```

```
[[ 0.  0.  5. ...  0.  0.  0.]
 [ 0.  0.  0. ... 10.  0.  0.]
 [ 0.  0.  0. ... 16.  9.  0.]
 ...
 [ 0.  0.  1. ...  6.  0.  0.]
 [ 0.  0.  2. ... 12.  0.  0.]
 [ 0.  0. 10. ... 12.  1.  0.]]
(1797, 64)
PCA(n_components=2)
(1797, 2)
```



Steps for Plotting K-Means Clusters

In [24]:
```
from sklearn.cluster import KMeans
kmeans=KMeans(n_clusters=10)
#predict the lebels of clusters
label=kmeans.fit_predict(df)
print(label)
```

```
[2 8 1 ... 1 3 6]
```

K-Means- Sample Program

In [55]:
```
from sklearn.cluster import KMeans
import numpy as np
X = np.array([[3, 2], [1, 4], [1, 0],[10, 2], [10, 4], [10, 0]])
X
```

Out[55]:
```
array([[ 3,  2],
       [ 1,  4],
       [ 1,  0],
       [10,  2],
       [10,  4],
       [10,  0]])
```

In [56]:
```
kmeans=KMeans(n_clusters=2, random_state=0, n_init=10).fit(X)
kmeans.labels_
```

Out[56]:
```
array([1, 1, 1, 0, 0, 0])
```

In [60]:
```
kmeans.predict([[0, 0], [12, 3]])
```

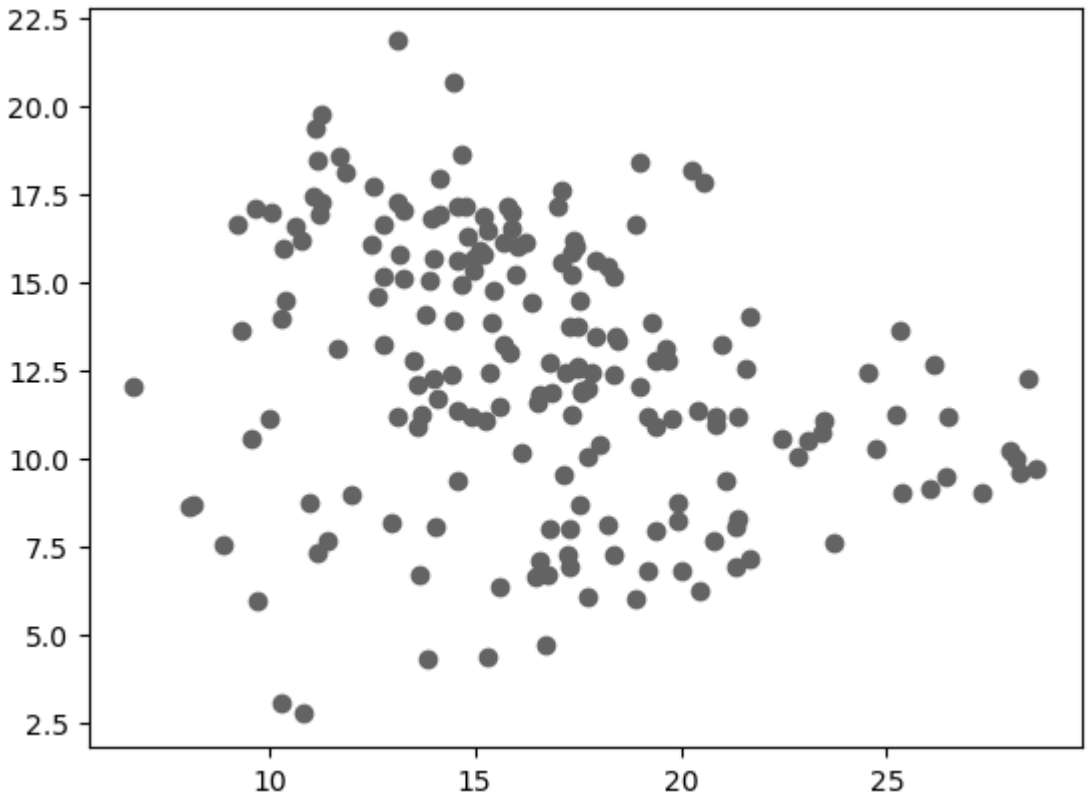
Out[60]:
```
array([1, 0])
```

In [62]:
```
kmeans.cluster_centers_
```

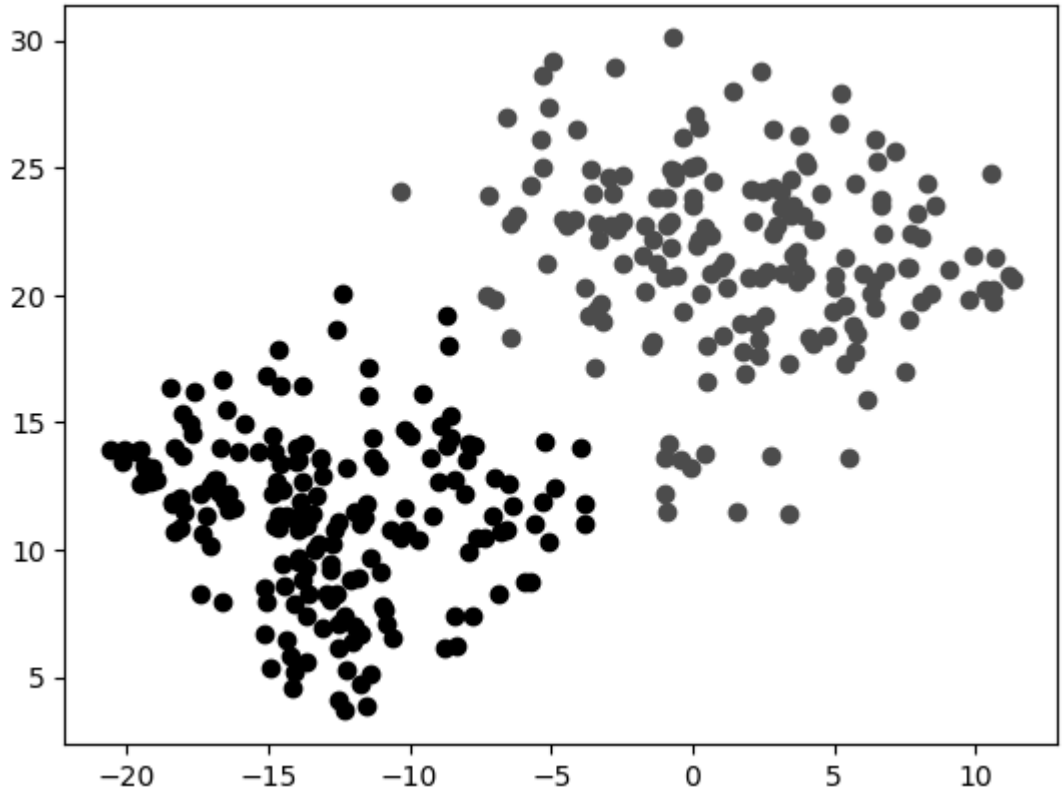
Out[62]:
```
array([[10.        ,  2.        ],
       [ 1.66666667,  2.        ]])
```

Find Particular Label

In [30]:
```
import matplotlib.pyplot as plt
flabel=df[label==0]
# print(flabel)
plt.scatter(flabel[:,0],flabel[:,1])
plt.show()
```

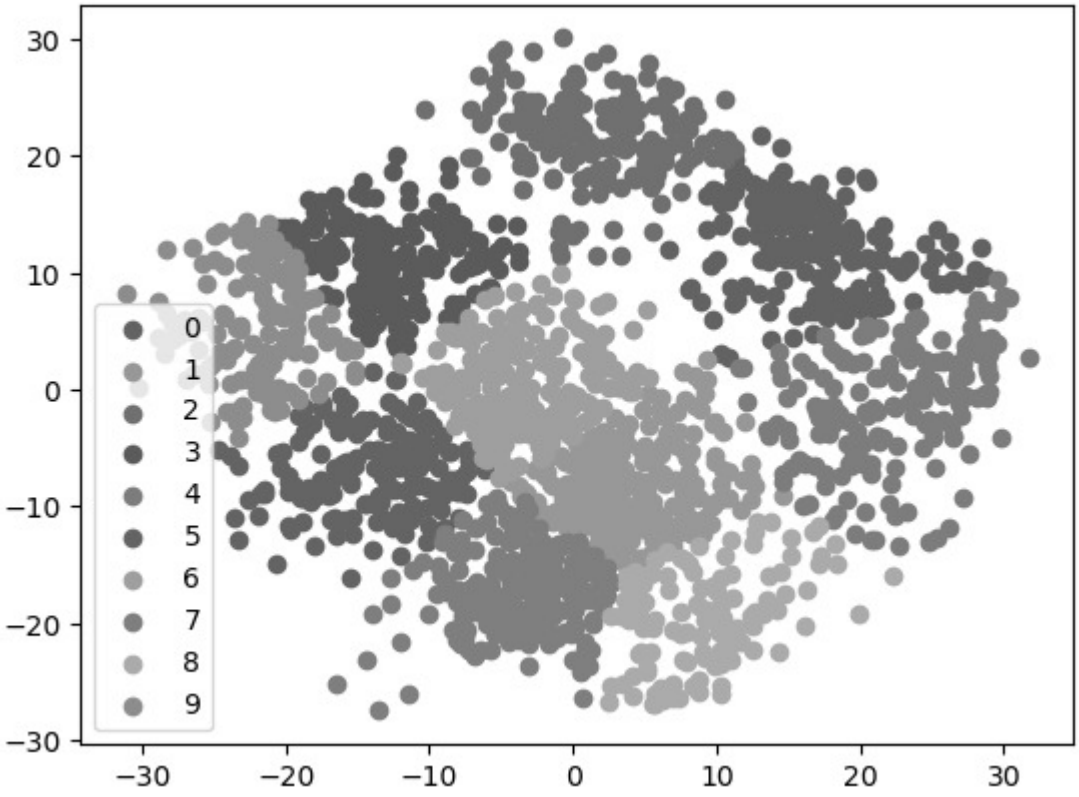


In [32]:
```
flabel2=df[label==2]
flabel8=df[label==3]
plt.scatter(flabel2[:,0],flabel2[:,1],color='red')
plt.scatter(flabel8[:,0],flabel8[:,1],color='black')
plt.show()
```



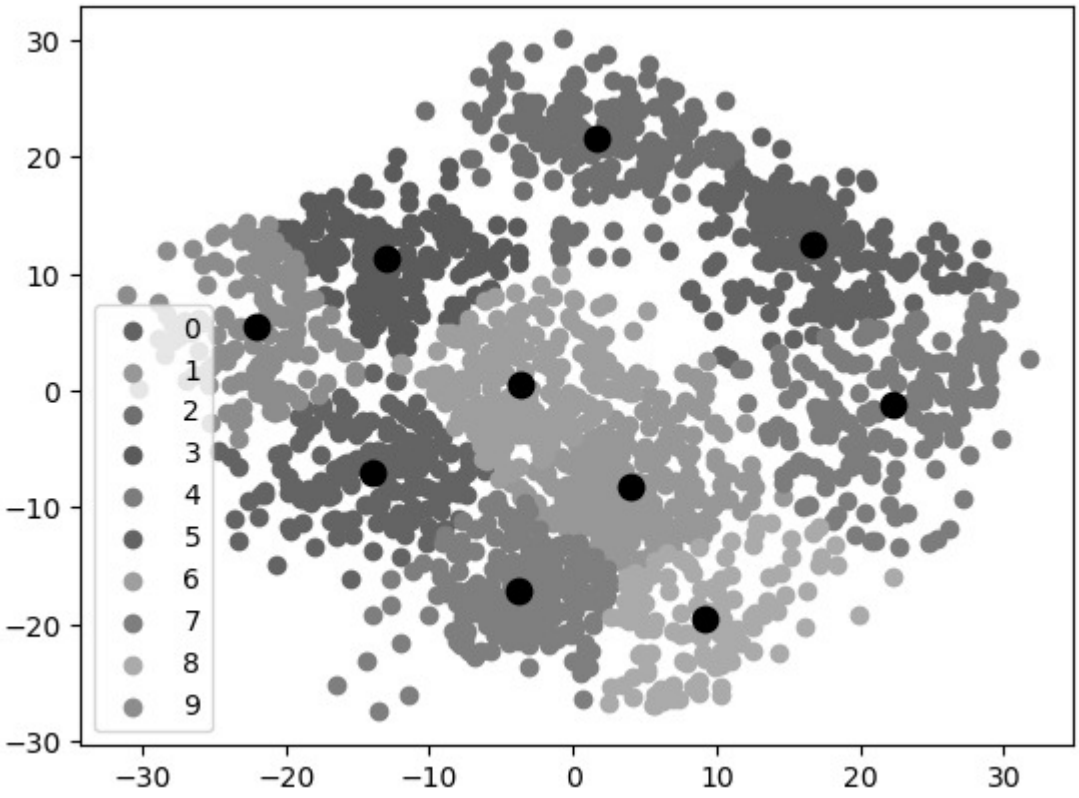
Show Unique Labels

In [33]:
```
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

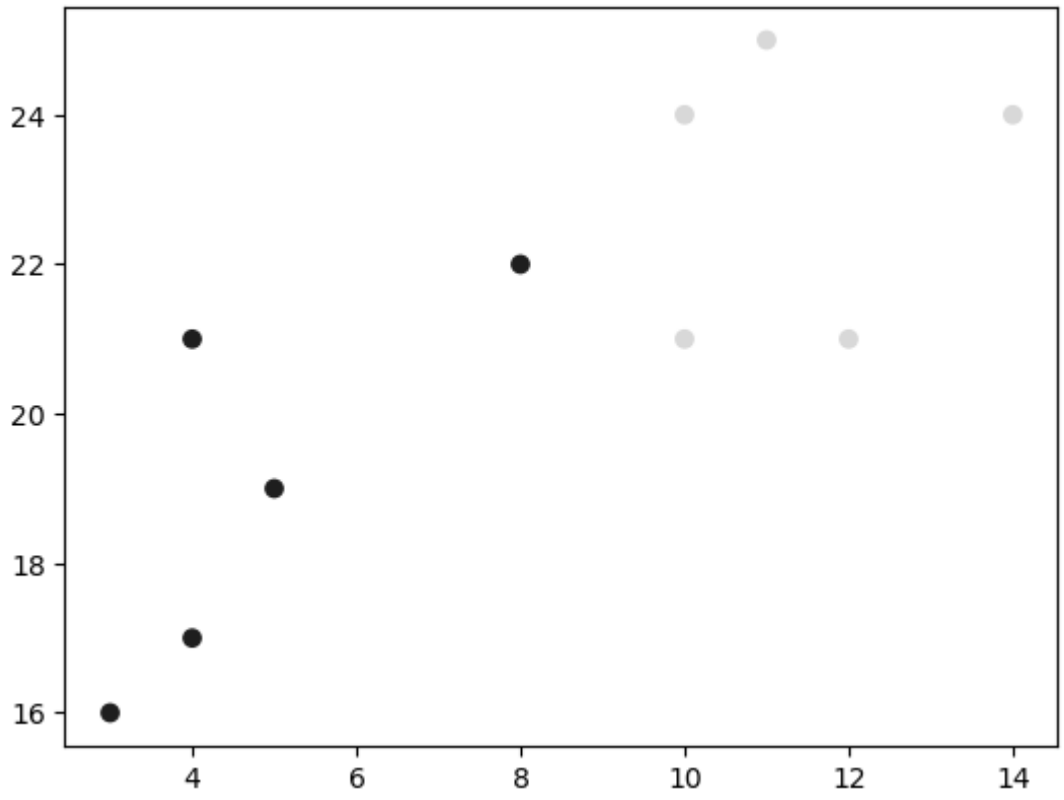
In [34]:
```
#Getting the centroids
centroids=kmeans.cluster_centers_
u_labels=np.unique(label)
for i in u_labels:
    plt.scatter(df[label==i,0],df[label==i,1],label=i)
plt.scatter(centroids[:,0],centroids[:,1],s=80,color='k')
plt.legend()
plt.show()
```



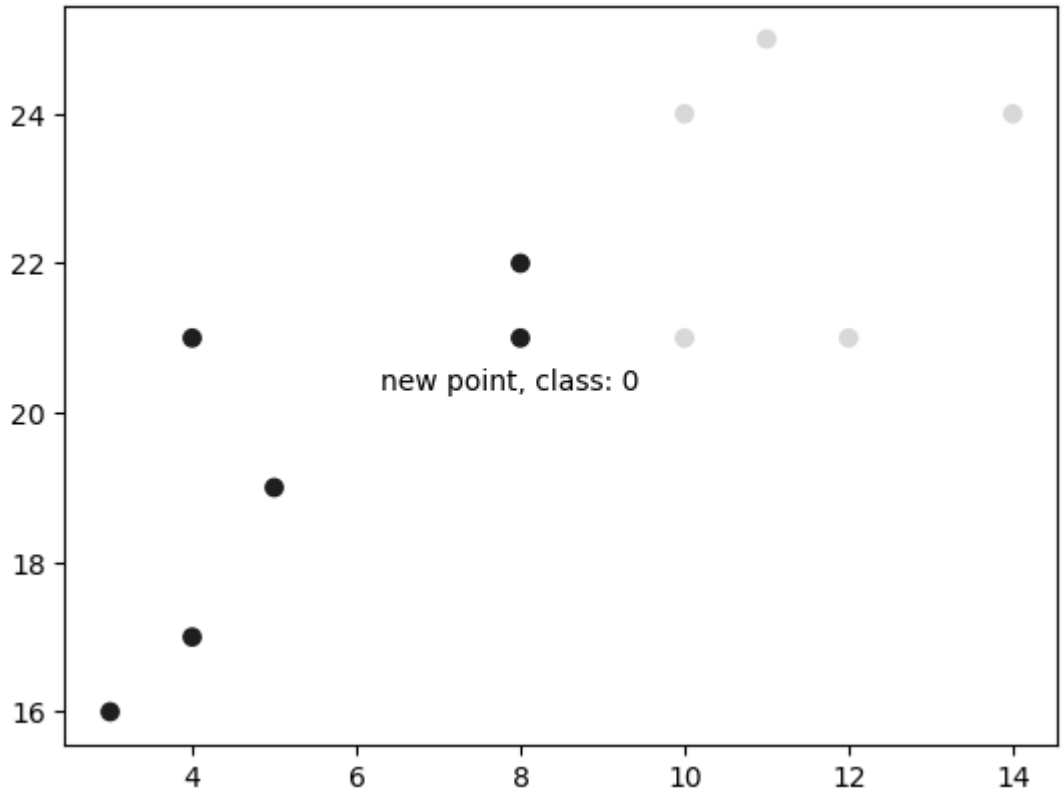
K-nearest neighbors (KNN)

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 [36]:
```
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()
```

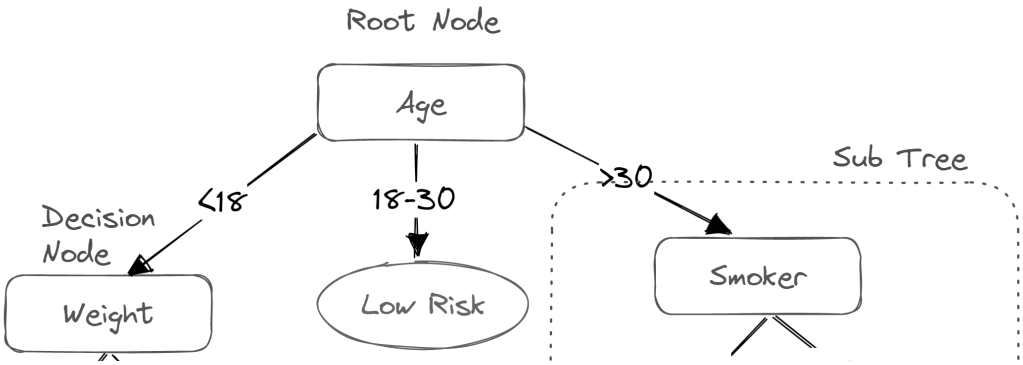


In [37]:
```
from sklearn.neighbors import KNeighborsClassifier

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()
```



Machine Learning- Deep Learning Classification¶

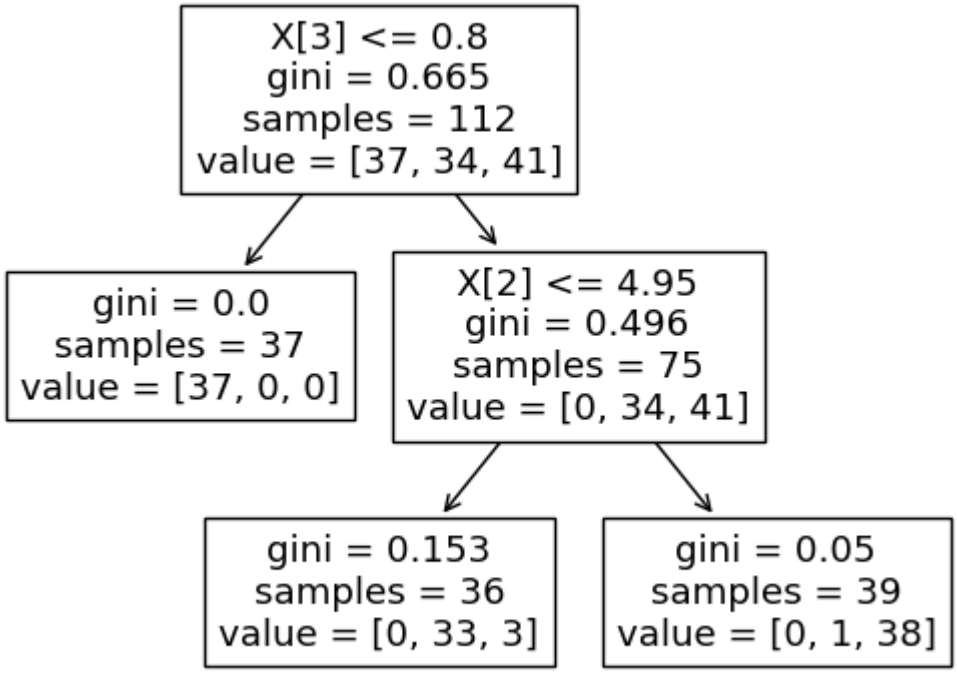
Decision Tree Classifier is one of the most powerful and popular algorithm. Decision-tree algorithm falls under the category of supervised learning algorithms. It works for both continuous as well as categorical output variables.



In [40]:
```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_iris
from sklearn import tree
from sklearn import metrics

iris=load_iris()
X=iris.data
Y=iris.target
X_train, X_test, Y_train, Y_test = train_test_split(X, Y,random_state = 0)
clf=DecisionTreeClassifier(max_leaf_nodes=3,random_state=0)

# Train Decision Tree Classifer
clf.fit(X_train,Y_train)
#Predict the response for test dataset
y_pred = clf.predict(X_test)
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(Y_test, y_pred))
```

```
Accuracy: 0.8947368421052632
```

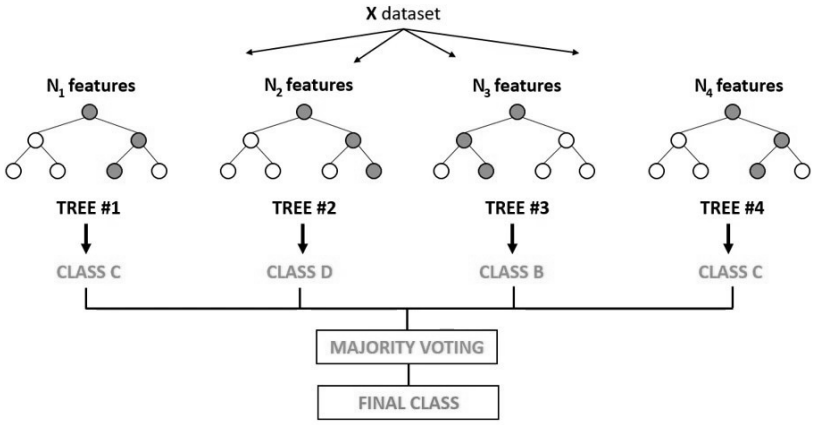
In [41]:
```
tree.plot_tree(clf)
plt.show()
```



What is Random Forest Regression?

Random Forest is an ensemble technique capable of performing both regression and classification tasks with the use of multiple decision trees and a technique called Bootstrap and Aggregation, commonly known as bagging. The basic idea behind this is to combine multiple decision trees in determining the final output rather than relying on individual decision trees.

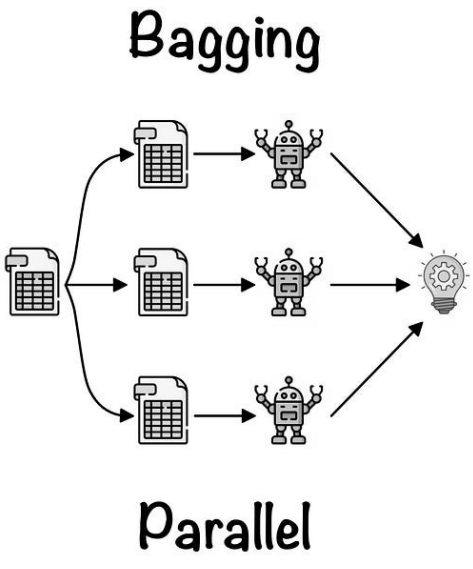
Random Forest has multiple decision trees as base learning models. We randomly perform row sampling and feature sampling from the dataset forming sample datasets for every model. This part is called Bootstrap.

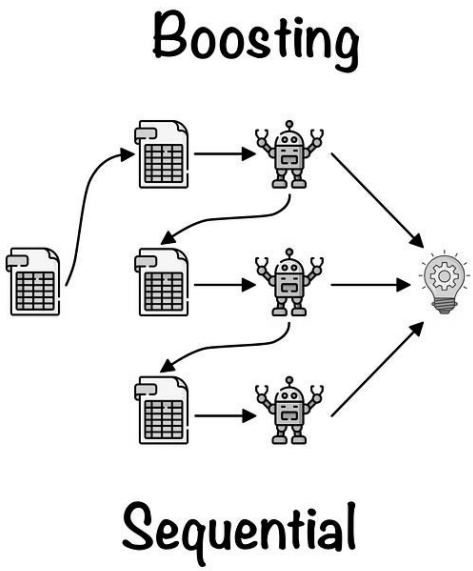


https://www.kaggle.com/code/prashant111/ranc
forest-classifier-tutorial
(https://www.kaggle.com/code/prashant111/ran
forest-classifier-tutorial)

Bagging & Boosting

Bagging different training data subsets are randomly drawn with replacement from the entire training dataset. In Boosting every new subsets contains the elements that were misclassified by previous models. Bagging tries to solve over-fitting problem while Boosting tries to reduce bias.





In [22]:
```
import pandas as pd
df = pd.read_csv("diabetes.csv")
df.head()
```

Out[22]:

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

In [23]:
```
df.isnull().sum()
```

Out[23]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [37]:
```
X = df.drop("Outcome",axis="columns")
y = df.Outcome
print(X)
# print(y)
```

```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0              6      148             72             35        0  33.6
1              1       85             66             29        0  26.6
2              8      183             64              0        0  23.3
3              1       89             66             23       94  28.1
4              0      137             40             35      168  43.1
..           ...      ...            ...            ...      ...   ...
763           10      101             76             48      180  32.9
764            2      122             70             27        0  36.8
765            5      121             72             23      112  26.2
766            1      126             60              0        0  30.1
767            1       93             70             31        0  30.4

     DiabetesPedigreeFunction  Age
0                       0.627   50
1                       0.351   31
2                       0.672   32
3                       0.167   21
4                       2.288   33
..                        ...  ...
763                     0.171   63
764                     0.340   27
765                     0.245   30
766                     0.349   47
767                     0.315   23

[768 rows x 8 columns]
```

Dataset scaling

Dataset scaling is transforming a dataset to fit within a specific range. For example, you can scale a dataset to fit within a range of 0-1, -1-1, or 0-100.

In [25]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_scaled[:3]
```

Out[25]:
```
array([[ 0.63994726,  0.84832379,  0.14964075,  0.90726993, -0.69289057,
         0.20401277,  0.46849198,  1.4259954 ],
       [-0.84488505, -1.12339636, -0.16054575,  0.53090156, -0.69289057,
        -0.68442195, -0.36506078, -0.19067191],
       [ 1.23388019,  1.94372388, -0.26394125, -1.28821221, -0.69289057,
        -1.10325546,  0.60439732, -0.10558415]])
```

Splitting the Dataset

We will split the scaled dataset into training and testing. To split the dataset, we will use the train_test_split method.

In [38]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, stratify=y
print(X_train.shape)
print(X_test.shape)
```

```
(576, 8)
(192, 8)
```

Model building using Decision Tree Classifier

The decision tree classifier is the Scikit-learn algorithm used for classification. To import this algorithm, use this code:

In [27]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score
scores = cross_val_score(DecisionTreeClassifier(), X, y, cv=5)
scores
```

Out[27]:
```
array([0.66883117, 0.67532468, 0.68181818, 0.79084967, 0.7254902 ])
```

Getting the Mean Accuracy Score

In [28]:
```
scores.mean()
```

Out[28]:
```
0.7084627790510143
```

Building the model using Bagging Classifier

In [39]:
```
from sklearn.ensemble import BaggingClassifier
bag_model = BaggingClassifier(
base_estimator=DecisionTreeClassifier(),
n_estimators=50,
max_samples=0.8,
bootstrap=True,
oob_score=True,
random_state=0
)
bag_model.fit(X_train, y_train)
# Accuracy Score
bag_model.oob_score_
```

Out[39]:
```
0.7447916666666666
```

In [40]:
```
bag_model.score(X_test, y_test)
```

Out[40]:
```
0.7916666666666666
```

Random Forest Classifier

Random Forest Classifier has several decision trees trained on the various subsets. This algorithm is a typical example of a bagging algorithm.

In [41]:
```
from sklearn.ensemble import RandomForestClassifier
scores = cross_val_score(RandomForestClassifier(n_estimators=50), X, y, cv=
scores.mean()
```

Out[41]:
```
0.7696205755029284
```

Data Visualization for K-Means

In [42]:
```
import pandas as pd
df=pd.read_csv('iris.csv')
df=pd.DataFrame(df)
df.head()
```

Out[42]:

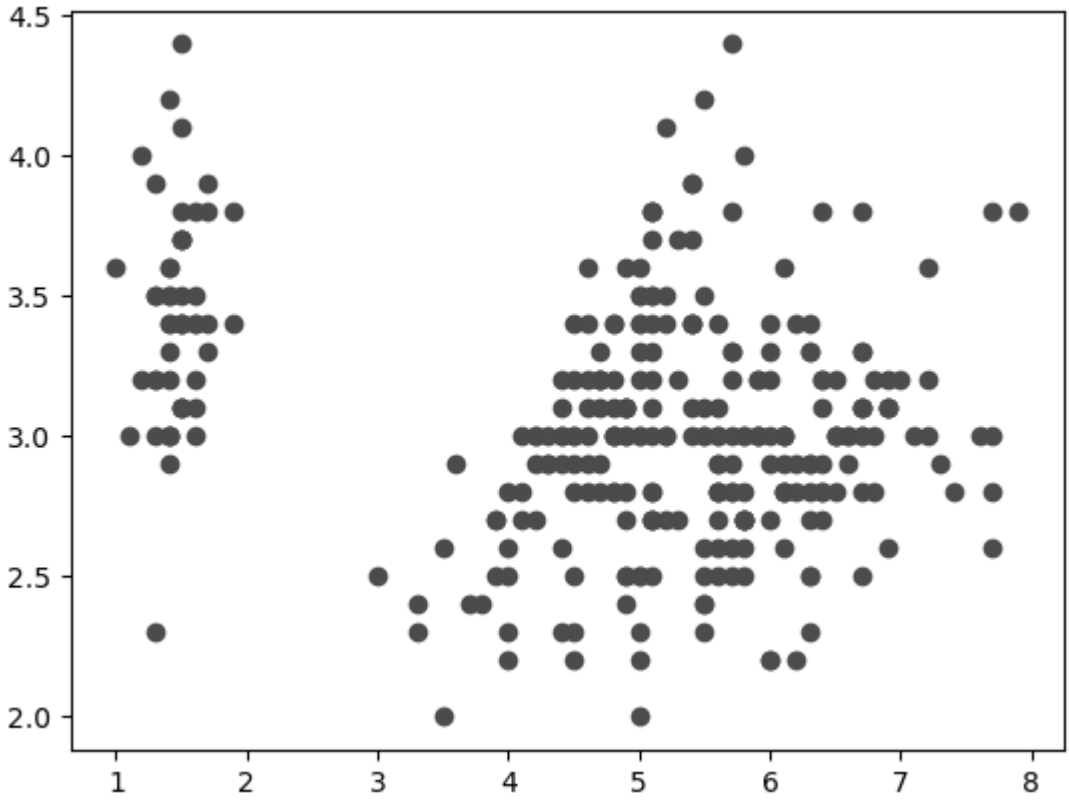
	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [43]:
```
label_0=df['sepal_length']
label_1=df['sepal_width']
label_2=df['petal_length']
```

In [44]:
```
import matplotlib.pyplot as plt
cols=df.columns
cols
```

Out[44]:
```
Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width',
       'species'],
      dtype='object')
```

In [46]:
```
plt.scatter(df['sepal_length'],df['sepal_width'],color='red')
plt.scatter(df['petal_length'],df['sepal_width'],color='green')
plt.show()
```



What Is Web Scraping?¶

Web scraping is the process of gathering information from the Internet. Even copying and pasting the lyrics of your favorite song is a form of web scraping! However, the words “web scraping” usually refer to a process that involves automation. Some websites don’t like it when automatic scrapers gather their data, while others don’t mind.

In [1]:
```
import requests
URL = " https://www.aptechmeerut.com/contact-us/"
r = requests.get(URL)
#print(r.content)
print(r.text)
```

```
<!DOCTYPE html>
<html lang="en">
<head>

  <meta charset='utf-8'>
  <meta name="viewport" content="width=device-width, initial-scale=1" i
d="wixDesktopViewport" />
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="generator" content="Wix.com Website Builder"/>

  <link rel="icon" sizes="192x192" href="https://static.wixstatic.com/m
edia/ebe481_de8d580afc8f4d188d52742ffee95a11%7Emv2.jpg/v1/fill/w_192%2C
h_192%2Clg_1%2Cusm_0.66_1.00_0.01/ebe481_de8d580afc8f4d188d52742ffee95a
11%7Emv2.jpg" type="image/jpeg"/>
  <link rel="shortcut icon" href="https://static.wixstatic.com/media/eb
e481_de8d580afc8f4d188d52742ffee95a11%7Emv2.jpg/v1/fill/w_32%2Ch_32%2Cl
g_1%2Cusm_0.66_1.00_0.01/ebe481_de8d580afc8f4d188d52742ffee95a11%7Emv2.
jpg" type="image/jpeg"/>
  <link rel="apple-touch-icon" href="https://static.wixstatic.com/medi
```

In [2]:
```
import requests
URL = " https://www.naukri.com/mnjuser/homepage"
r = requests.get(URL)
#print(r.content)
print(r.text)
```

```
<!DOCTYPE html><html lang="en"><head><link rel="stylesheet" href="http
s://static.naukimg.com/s/9/105/_next/static/css/d4dcda8a74d605fe.css" d
ata-precedence="high"/><link rel="stylesheet" href="https://static.nauk
img.com/s/9/105/_next/static/css/a26be15eaa27d0fa.css" data-precedence
="high"/><link rel="stylesheet" href="https://static.naukimg.com/s/9/10
5/_next/static/css/167f6f4dfeab5db1.css" data-precedence="high"/><link
rel="stylesheet" href="https://static.naukimg.com/s/9/105/_next/static/
css/9a0ea302f8070447.css" data-precedence="high"/><link rel="styleshee
t" href="https://static.naukimg.com/s/9/105/_next/static/css/7ac4a69500
80226a.css" data-precedence="high"/><link rel="stylesheet" href="http
s://static.naukimg.com/s/9/105/_next/static/css/a929559f825f066c.css" d
ata-precedence="high"/><link rel="stylesheet" href="https://static.nauk
img.com/s/9/105/_next/static/css/763ea3f5f5ed13e8.css" data-precedence
="high"/><script src="https://static.naukimg.com/s/9/105/_next/static/c
hunks/polyfills-c67a75d1b6f99dc8.js" nomodule=""></script></head><body>
<img width="99999" height="99999" style="pointer-events:none;position:a
bsolute;top:0;left:0;width:95vw;height:95vh;max-width:95vw;max-height:9
5vh;z-index:1000" src="
4wI iBlbmNvZGluZz0iVVRGLTgiPz48c3ZnIHdpZHRoPSI5OTk5OXB4IiBoZWlnaHQ9Ijk5
```

In [3]:
```
import requests
from bs4 import BeautifulSoup
URL = "https://realpython.github.io/fake-jobs/"
page = requests.get(URL)
soup = BeautifulSoup(page.content, "html.parser")
print(soup.prettify())
```

```
m/media/real-python-logo-thumbnail.7f0db70c2ed2.jpg?__no_cf_polish=1"/>
          </figure>
         </div>
         <div class="media-content">
          <h2 class="title is-5">
           Barrister
          </h2>
          <h3 class="subtitle is-6 company">
           Washington-Castillo
          </h3>
         </div>
        </div>
        <div class="content">
         <p class="location">
          Perezton, AE
         </p>
         <p class="is-small has-text-grey">
          <time datetime="2021-04-08">
           2021-04-08
```

In [5]:
```
results = soup.find(id="ResultsContainer")
job_elements = results.find_all("div", class_="card-content")
print(results.prettify())
```

```
<div class="columns is-multiline" id="ResultsContainer">
 <div class="column is-half">
  <div class="card">
   <div class="card-content">
    <div class="media">
     <div class="media-left">
      <figure class="image is-48x48">
       <img alt="Real Python Logo" src="https://files.realpython.com/me
dia/real-python-logo-thumbnail.7f0db70c2ed2.jpg?__no_cf_polish=1"/>
      </figure>
     </div>
     <div class="media-content">
      <h2 class="title is-5">
       Senior Python Developer
      </h2>
      <h3 class="subtitle is-6 company">
       Payne, Roberts and Davis
      </h3>
     </div>
```

In [6]:
```
for job_element in job_elements:
 title_element = job_element.find("h2", class_="title")
 company_element = job_element.find("h3", class_="company")
 location_element = job_element.find("p", class_="location")
 print(title_element)
 print(company_element)
 print(location_element)
```

```
        Kelseystad, AA
      </p>
<h2 class="title is-5">Audiological scientist</h2>
<h3 class="subtitle is-6 company">Salazar-Meyers</h3>
<p class="location">
        Williamsburgh, AE
      </p>
<h2 class="title is-5">English as a second language teacher</h2>
<h3 class="subtitle is-6 company">Parker, Murphy and Brooks</h3>
<p class="location">
        Mitchellburgh, AE
      </p>
<h2 class="title is-5">Surgeon</h2>
<h3 class="subtitle is-6 company">Cruz-Brown</h3>
<p class="location">
        West Jessicabury, AA
      </p>
<h2 class="title is-5">Equities trader</h2>
<h3 class="subtitle is-6 company">Macdonald-Ferguson</h3>
```

In [7]:
```
for job_element in job_elements:
 title_element = job_element.find("h2", class_="title")
 company_element = job_element.find("h3", class_="company")
 location_element = job_element.find("p", class_="location")
 print(title_element.text.strip())
 print(company_element.text.strip())
 print(location_element.text.strip())
```

```
South Christopher, AE
Textile designer
Meyers-Johnson
Port Jonathan, AE
Television floor manager
Hughes-Williams
Osbornetown, AE
Waste management officer
Jones, Williams and Villa
Scotttown, AP
Software Engineer (Python)
Garcia PLC
Ericberg, AE
Interpreter
Gregory and Sons
Ramireztown, AE
Architect
Clark, Garcia and Sosa
Figueroaview, AA
Meteorologist
```

In [8]:
```
import requests
URL = "https://dummyjson.com/products/1"
r = requests.get(URL)
#print(r.content)
print(r.text)
```


```
{"id":1,"title":"iPhone 9","description":"An apple mobile which is nothing
like apple","price":549,"discountPercentage":12.96,"rating":4.69,"stock":9
4,"brand":"Apple","category":"smartphones","thumbnail":"https://i.dummyjso
n.com/data/products/1/thumbnail.jpg","images":["https://i.dummyjson.com/da
ta/products/1/1.jpg","https://i.dummyjson.com/data/products/1/2.jpg","http
s://i.dummyjson.com/data/products/1/3.jpg","https://i.dummyjson.com/data/p
roducts/1/4.jpg","https://i.dummyjson.com/data/products/1/thumbnail.jpg"]}
```