Assignment of Data Mining

[Basic python, Numpy, Pandas]

Submitted To-

Dr.Md.Mawarul Islam
Associate Professor

Submitted By-

Susmita Rani Saha (B180305047)
Tanvir Ahammed Hridoy (b180305020)

Department of computer Sceience and Engineering,

Jagannath university

String

A string is a sequence of characters, can contain letters, numbers, symbols and even spaces.

➢ Strings are immutable:

Once we create string, then we can't change the content of this object. If we want to change then we must create a new string.

Example:

```
In [1]: str="priya"
        print("the string is : ",str)

        the string is :  priya

In [2]: str[1]='a'
        str

        ---------------------------------------------------------------------------
        TypeError                                 Traceback (most recent call last)
        ~\AppData\Local\Temp\ipykernel_1336\661536508.py in <module>
        ----> 1 str[1]='a'
              2 str

        TypeError: 'str' object does not support item assignment
```

Lists

Python has a data type known as list. Lists are same as arrays. That is, List is a collection that allows us to put many variable in a single variable.

➢ Lists are Mutable:

We can change an element in any index of list.

Example:

```
In [3]: list1=[4,7,2,9]
        print("The list is : ",list1)
        list1[2]=5
        print("After change the content of index 2 : ",list1)

        The list is :  [4, 7, 2, 9]
        After change the content of index 2 :  [4, 7, 5, 9]
```

➢ Indexing & negative indexing:

Lists have zero based indexing from front and also have negative indexing from end. We can access any element using index operator.

Example:

```
In [4]: list2=[4,7,2,9]
        print("The list is : ",list2)
        print("the contnt of index 2 : ",list2[2])
        print("the element in -1 : ",list2[-1])

        The list is :  [4, 7, 2, 9]
        the contnt of index 2 :  2
        the element in -1 :  9
```

➢ Zeros array:

We can declare an array that fill with zero.

Example:

```
In [5]: zlist=[0]*10
        print("the element of zlist is :",zlist)

        the element of zlist is : [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

➢ Iterate through lists:

We use loop for iterate list. Loop is used to repeat a block of code until the specified is met. For access elements in a list we can use loop.

Example:

```
In [10]: list1=[3,7,4,0,2]
         for i in list1:
             print(i)

         3
         7
         4
         0
         2
```

➢ Iterate through list using range method:

We can access the elements in list define start and end point. Range method is use for define limit.

Syntax: range(start-inclusive, end-exclusive)

Example : We can apply any operation in list using range

```
In [39]: list1=[1,2,3,4,5]
         print("Initial list is :",list1)
         for i in range(len(list1)):
             list1[i]=list1[i]+1
         print("Now the list is :",list1)

         Initial list is : [1, 2, 3, 4, 5]
         Now the list is : [2, 3, 4, 5, 6]
```

• Basic methods:

There's some basic method, that can directly modify the lists.

➢ Append() method:

Append or add an item to the end of the list.

Syntax: list1.append(item)

Example:

```
In [25]: list3=[3,9,2,7]
         print("the initial list is :",list3)
         list3.append(9)
         print("After append the list is :",list3)

         the initial list is : [3, 9, 2, 7]
         After append the list is : [3, 9, 2, 7, 9]
```

➢ Insert() method:

Insert an item at the specified index.

Syntax: list1.insert(index, item)

Example:

```
In [27]: list3=[3,9,2,7]
         print("the initial list is :",list3)
         list3.insert(1,8)
         print("After insert the list is :",list3)

         the initial list is : [3, 9, 2, 7]
         After insert the list is : [3, 8, 9, 2, 7]
```

➢ Remove() method:

Remove the first occurrence of item from the list.

Syntax: list1.remove(item)

Example 1:

```
In [28]: list3=[3,9,2,7,8]
         print("the initial list is :",list3)
         list3.remove(8)
         print("After remove, the list is :",list3)

         the initial list is : [3, 9, 2, 7, 8]
         After remove, the list is : [3, 9, 2, 7]
```

Example 2: we can remove elements using range.

```
In [14]: list1=[1,2,3,4,5]
         for i in range(1,3):
             list1.remove(i)
         print(list1)

         [3, 4, 5]
```

➢ Extend() method:

Using this method we can append another list to the list.

Syntax: list1.extend(list2)

Example:

```
In [29]: list3=[3,9,2,7,8]
         list4=['a','b']
         print("the initial list is :",list3)
         list3.extend(list4)
         print("After extend, the list is :",list3)

         the initial list is : [3, 9, 2, 7, 8]
         After extend, the list is : [3, 9, 2, 7, 8, 'a', 'b']
```

➢ Count() method:

This method returns the number of times element occurs in the list.

Syntax: list1.count(element)

Example:

```
In [30]: list3=[3,9,2,7,8,7,7]
         print("the initial list is :",list3)
         n=list3.count(7)
         print("7 in the list :",n,"times")

         the initial list is : [3, 9, 2, 7, 8, 7, 7]
         7 in the list : 3 times
```

➢ Sort() method:

For sort the elements of list we use sort method. Sort items in a list in ascending order.

Syntax: list.sort()

Example:

```
In [31]: list3=[3,9,2,7,8,7,7]
         print("the initial list is :",list3)
         list3.sort()
         print("After sort, the list is :",list3)

         the initial list is : [3, 9, 2, 7, 8, 7, 7]
         After sort, the list is : [2, 3, 7, 7, 7, 8, 9]
```

➢ Reverse() method:

This reverse method uses for reverse the list. That is, it reverses the order of items in the list.

Syntax: list.reverse()

Example:

```
In [32]: list3=[3,9,2,7,8,7,7]
         print("the initial list is :",list3)
         list3.reverse()
         print("After reverse, the list is :",list3)

         the initial list is : [3, 9, 2, 7, 8, 7, 7]
         After reverse, the list is : [7, 7, 8, 7, 2, 9, 3]
```

➢ Copy() method:

Its copy the elements of a list and return copied list.

Syntax: list1=list.copy()

Example:

```
In [33]: list3=[3,9,2,7,8,7,7]
         print("the initial list is :",list3)
         list4=list3.copy()
         print("After copy,new list is :",list4)

         the initial list is : [3, 9, 2, 7, 8, 7, 7]
         After copy,new list is : [3, 9, 2, 7, 8, 7, 7]
```

➢ Pop() method:

This method removes and returns an element at the given index.

Syntax: n=list.pop(index)

Example:

```
In [35]: list3=[3,9,2,7,8,7,7]
         print("the initial list is :",list3)
         n=list3.pop(4)
         print("the number of index 4 is :",n)
         print("Now list is :",list3)

         the initial list is : [3, 9, 2, 7, 8, 7, 7]
         the number of index 4 is : 8
         Now list is : [3, 9, 2, 7, 7, 7]
```

➢ Clear() method:

This method removes all items from the list.

Syntax: list.clear()

Example:

```
In [36]: list3=[3,9,2,7,8,7,7]
         print("the initial list is :",list3)
         list3.clear()
         print("Now list is :",list3)

         the initial list is : [3, 9, 2, 7, 8, 7, 7]
         Now list is : []
```

➢ Len() method:

If we measure the length of array then we can use len() method, this method return the length of list.

Syntax: n=len(list)

Example:

```
In [37]: list3=[3,9,2,7,8,7,7]
         print("the initial list is :",list3)
         n=len(list3)
         print("The length of the list is :",n)

         the initial list is : [3, 9, 2, 7, 8, 7, 7]
         The length of the list is : 7
```

➢ Slicing method:

Using Slicing method, we can get la sub list of a list. We can access elements in range. We can get all elements using slice operator.

Syntax: list[star-inclusive : end-exclusive]

Example:

```
In [22]: List = [1,2,5,9,4,8,7,2]
         print("Intial List: ",List)
         SList = List[1:3]
         print("Slicing elements in a range 1-3: ",SList)
         SList = List[2:]
         print("Elements sliced from 2th element till the end: ",SList)
         SList = List[:]
         print("Printing all elements using slice operation: ",SList)

         Intial List:  [1, 2, 5, 9, 4, 8, 7, 2]
         Slicing elements in a range 1-3:  [2, 5]
         Elements sliced from 2th element till the end:  [5, 9, 4, 8, 7, 2]
         Printing all elements using slice operation:  [1, 2, 5, 9, 4, 8, 7, 2]
```

➢ Split() method:

This method returns a list that split a string. String is converted to the elements of list.

Example:

```
In [18]: str1="i am priya"
         str2=str1.split()
         print(str2)

         ['i', 'am', 'priya']
```

• Mathematical methods:

There's some methods using for mathematical operations.

➢ Sum() method:

Using this method we can sums up the numbers in the list.

Syntax: n=sum(list)

Example:

```
In [40]: list3=[3,9,2,7,8,7,7]
         print("the initial list is :",list3)
         n=sum(list3)
         print("Sum of the list elements is :",n)

         the initial list is : [3, 9, 2, 7, 8, 7, 7]
         Sum of the list elements is : 43
```

➢ Max() method:

For finding maximum value in list we can use max method.

Syntax: n=max(list)

Example:

```
In [41]: list3=[3,9,2,7,8,7,7]
         print("the initial list is :",list3)
         n=max(list3)
         print("Max of the list elements is :",n)

         the initial list is : [3, 9, 2, 7, 8, 7, 7]
         Max of the list elements is : 9
```

➢ Min() method:

For finding minimum value in list we can use min method.

Syntax: n=min(list)

Example:

```
In [42]: list3=[3,9,2,7,8,7,7]
         print("the initial list is :",list3)
         n=min(list3)
         print("Min of the list elements is :",n)

         the initial list is : [3, 9, 2, 7, 8, 7, 7]
         Min of the list elements is : 2
```

Dictionary

A dictionary associates a simple data value called a key (most often string) with a value. And values can be of any python data type.
Syntax: dic {key1, value, ……}

➢ Create an empty dictionary:

We can create an empty dictionary without assigning any key or value.

Example:

```
In [43]: eDictionary = {}
         print ("The emptyDictionary is:", eDictionary )

         The emptyDictionary is: {}
```

➢ Create a dictionary:

Create a dictionary named grades which contains name as key and grade as value of dictionary.

Example:

```
In [44]: grades = { "a": 87, "b": 76, "c": 92, "d": 89 }
         print ("All grades:", grades )

         All grades: {'a': 87, 'b': 76, 'c': 92, 'd': 89}
```

➢ Update value for existing index:

We can update or change the value of already existing Index.

Example:

```
In [45]: grades = { "a": 87, "b": 76, "c": 92, "d": 89 }
         print ("a's current grade:", grades[ "a" ] )
         grades[ "a" ] = 90
         print ("a's new grade:", grades[ "a" ] )

         a's current grade: 87
         a's new grade: 90
```

➢ Added a new entry:

We can add a new entry that is, key and value pair of the dictionary.

Example:

```
In [47]: grades = { "a": 87, "b": 76, "c": 92, "d": 89 }
         print ("Initial all grades:", grades )
         grades[ "e" ] = 93
         print ("Dictionary grades after modification:",grades )

         Initial all grades: {'a': 87, 'b': 76, 'c': 92, 'd': 89}
         Dictionary grades after modification: {'a': 87, 'b': 76, 'c': 92, 'd': 89, 'e': 93}
```

➢ Delete entry from dictionary:

For remove an entry from dictionary we have to use del keyword, and then specific key for delete.

Example:

```
In [49]: grades = { "a": 87, "b": 76, "c": 92, "d": 89 }
         print ("Initial all grades:", grades )
         del grades[ "d" ]
         print ("Dictionary grades after deletion:",grades )

         Initial all grades: {'a': 87, 'b': 76, 'c': 92, 'd': 89}
         Dictionary grades after deletion: {'a': 87, 'b': 76, 'c': 92}
```

➢ Iterate through dictionary:

Using loop we can access all elements of the dictionary.

Example:

```
In [53]: grades = { "a": 87, "b": 76, "c": 92, "d": 89 }
         for key in grades:
             print(key)
         for key,value in grades.items():
             print(key, '-->', value)
```

```
a
b
c
d
a --> 87
b --> 76
c --> 92
d --> 89
```

- ## Some methods:

➢ Values() methods:

This method returns all values in dictionary.

Syntax: Dic.values()

Example:

```
In [1]: grades = { "a": 87, "b": 76, "c": 92, "d": 89 }
        value=grades.values()
        print(value)
```

```
dict_values([87, 76, 92, 89])
```

➢ Keys() method:

This method returns all keys in dictionary.

Syntax: Dic.keys()

Example:

```
In [2]: grades = { "a": 87, "b": 76, "c": 92, "d": 89 }
        key=grades.keys()
        print(key)

        dict_keys(['a', 'b', 'c', 'd'])
```

❖Some Python libraries:

i. Numpy
ii. Pandas
iii. Scipy
iv. Scikit-learn

We will discuss about only numpy and pandas.

Numpy

Numpy is a popular python library. It is the fundamental package needed for scientific computation with python.

It features:

i. Multidimensional array,
ii. Fast numerical computation,
iii. High level math function,

• Arrays:

Structured lists of numbers.

Two types:

i. Vectors (single dimensional array)
ii. Matrices (Multidimensional array)

➢ Single dimensional array create:

We can create a single dimensional array using numpy. We have to first import numpy then create array using this.

Example:

```
In [56]: import numpy as np
         sa=np.array([1,2,3])
         print(sa)

         [1 2 3]
```

➢ Multidimensional array create:

Like single dimensional array we can also create multidimensional array for store matrices values. We can also declare data type of its values.

Example:

```
In [57]: import numpy as np
         mA = np.array([[1, 2, 3], [3, 4, 5]])
         print(mA)
         mA = np.array([[1.1, 2, 3], [3, 4, 5]])
         print(mA)
         mA = np.array([[1, 2, 3], [3, 4, 5]], dtype = complex)
         print(mA)
```

```
[[1 2 3]
 [3 4 5]]
[[1.1 2.  3. ]
 [3.  4.  5. ]]
[[1.+0.j 2.+0.j 3.+0.j]
 [3.+0.j 4.+0.j 5.+0.j]]
```

➢ Basic properties (dimension, shape, data type):

For knowing the dimension (1D,2D) of dictionary we can use ndim method. It returns the dimensions.

Syntax: array.ndim

For knowing the shape (row, column) of dictionary we can use shape method. It returns the shape of dictionary.

Syntax: array.shape

For knowing the data type of the elements of dictionary we can dtype method, that returns the data type of elements.

Syntax: array.dtype

Example:

```
In [65]: import numpy as np
         a = np.array([[1, 2, 3], [3, 4, 5]])
         print("Array is :\n",a)
         print("dimension is :",a.ndim)
         print("Shape :",a.shape)
         print("data type is :",a.dtype)
```

```
Array is :
 [[1 2 3]
 [3 4 5]]
dimension is : 2
Shape : (2, 3)
data type is : int32
```

➢ Array addition:

We can add two array using add operator for create a new array, that represent the sum of this two array.

Example:

```
In [67]: import numpy as np
         A1 = np.array([[2, 4], [5, -6]])
         print("First Array is :\n",A1)
         A2 = np.array([[9, -3], [3, 6]])
         print("Second Array is :\n",A2)
         RA = A1 + A2
         print("After adding two array :\n",RA)
```

```
First Array is :
 [[ 2  4]
 [ 5 -6]]
Second Array is :
 [[ 9 -3]
 [ 3  6]]
After adding two array :
 [[11  1]
 [ 8  0]]
```

➢ Array multiplication:

We can multiply two array using dot method and store this result in an array.

Example:

```
In [68]: import numpy as np
         A1 = np.array([[2, 4], [5, -6]])
         print("First Array is :\n",A1)
         A2 = np.array([[9, -3], [3, 6]])
         print("Second Array is :\n",A2)
         RA = A1.dot(A2)
         print("After multiply two array :\n",RA)
```

```
First Array is :
 [[ 2  4]
 [ 5 -6]]
Second Array is :
 [[ 9 -3]
 [ 3  6]]
After multiply two array :
 [[ 30  18]
 [ 27 -51]]
```

- **Some methods:**

➢ zeros() method:

Using this method we can create an array of all zeros elements.

Syntax: np.zeros((row,column), dtype=data type)

Example:

```
In [10]: import numpy as np
         a=np.zeros(5)
         print(a)
         print()
         b=np.zeros((2,3))
         print(b)
         print()
         c=np.zeros((3,4),dtype=int)
         print(c)
         print()
```

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

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

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

➢ ones() method:

Using this method we can create an array(1D,2D) of all ones elements.

Syntax: np.ones((row,column), dtype=data type)

Example:

```
In [11]: import numpy as np
         a=np.ones(5)
         print(a)
         print()
         b=np.ones((2,3))
         print(b)
         print()
         c=np.ones((3,4),dtype=int)
         print(c)
         print()
```

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

[[1. 1. 1.]
 [1. 1. 1.]]

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

➢ arange() method:

This method takes start index, end index and step size and create an array using this info. Here start inclusive, end exclusive and step size by default 1.

Syntax: np.arange(start-inclusive, end-exclusive, step, dtype)

Example:

```
In [16]: import numpy as np
         a=np.arange(3,7)
         print(a)
         b=np.arange(2,9,2,float)
         print(b)
```

```
[3 4 5 6]
[2. 4. 6. 8.]
```

➢ concatenate() method:

This method concatenate two arrays.

Syntax: np.concatenate([array1,array2])

Example:

```
In [19]: import numpy as np
         a=np.ones((2,4))
         print("First array is:\n",a)
         b=np.zeros((3,4))
         print("Second array is:\n",b)
         c=np.concatenate([a,b])
         print("Concatenated array is:\n",c)
```

```
First array is:
 [[1. 1. 1. 1.]
 [1. 1. 1. 1.]]
Second array is:
 [[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
Concatenated array is:
 [[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

➢ astype() method:

This method use for type casting. It can change the data type of an array.

Syntax: np.astype(data type)

Example:

```
In [21]: import numpy as np
         a=np.array([[1,2,4],[5,6,7]])
         print("First array is:\n",a)
         print("\ndata type is",a.dtype)
         b=a.astype(complex)
         print("\nAfter changing datatype ,the array is:\n",b)
```

```
First array is:
 [[1 2 4]
 [5 6 7]]

data type is int32

After changing datatype ,the array is:
 [[1.+0.j 2.+0.j 4.+0.j]
 [5.+0.j 6.+0.j 7.+0.j]]
```

➢ random.rand() method:

This method is use for generate random values from 0 to <1. That is range is [0,1).

Syntax: np.random.rand(value) ,for single dimension

np.random.rand(row,column) ,for multidimension

Example 1:

```
In [38]: import numpy as np
         a=np.random.rand(10)
         print("The array is:\n",a)
```

```
The array is:
 [0.58937292 0.38379229 0.50001865 0.73802438 0.112956   0.50733631
 0.20506082 0.67445669 0.564774   0.99065735]
```

Example 2:

```
In [28]: import numpy as np
         a=np.random.rand(10,5)
         print("The array is:\n",a)
```

```
The array is:
 [[0.18314975 0.77138584 0.36113091 0.11726982 0.82778114]
 [0.47845453 0.82390423 0.80728759 0.85200303 0.0165412 ]
 [0.34852583 0.90841154 0.16361159 0.07124834 0.49171646]
 [0.27407696 0.18538947 0.01236812 0.90941582 0.03555404]
 [0.68552888 0.69428428 0.9987885  0.99543636 0.27461697]
 [0.81156401 0.55567354 0.52895814 0.94671858 0.47308452]
 [0.74065172 0.85703649 0.07976241 0.95640677 0.031331  ]
 [0.41642782 0.81730891 0.02027818 0.27348647 0.54193907]
 [0.60647663 0.14350598 0.30900947 0.66529633 0.83613161]
 [0.59739006 0.66250402 0.42227507 0.98204221 0.34404839]]
```

➢ linspace() method:

This method returns a numbers as sample numbers instead of step in arrange method. This method takes –

Start=starting point inclusive

Stop=stop point inclusive

Num= how many numbers in samples to generate

Endpoint= it includes last point. It always True by default.

Retstep=if true than result the sampling rate. By default it false.

Dtype=data type

Syntax: np.linspace(start,stop,num=n,endpoint=True,retstep=False,dtype=type)

Example:

```
In [37]: import numpy as np
         a=np.linspace(2,3,num=5,retstep=True,dtype=float,endpoint=True)
         print("The array is:\n",a)
```

```
The array is:
 (array([2.  , 2.25, 2.5 , 2.75, 3.  ]), 0.25)
```

Pandas (Part 1)

- Pandas is a Python library for data manipulation and analysis. It provides data structures and functions for working with structured data, such as tabular or time series data.

- The two primary data structures in Pandas are:
  i. Series and
  ii. DataFrame objects.

  A Series is a one-dimensional array-like object that can hold any data type, while a DataFrame is a two-dimensional table-like object that can hold multiple types of data.

- Pandas provides a wide range of functions for manipulating and analyzing data, such as filtering, sorting, grouping, merging, pivoting, and aggregating. It also has built-in support for handling missing data, time series data, and categorical data.

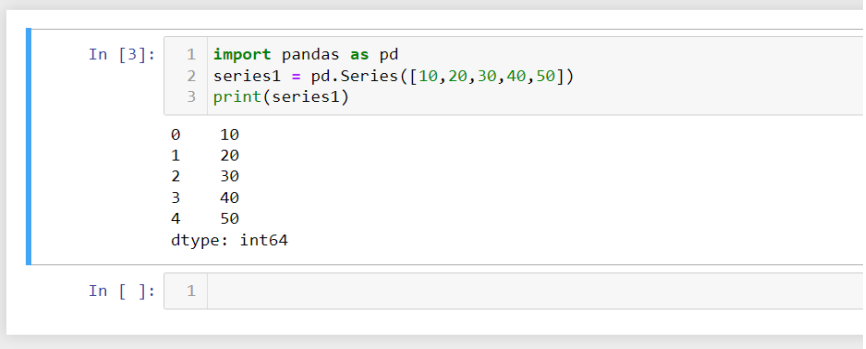
- Pandas is widely used in data analysis and scientific computing, and is often used in conjunction with other Python libraries such as NumPy, Matplotlib, and Scikit-Learn.

❖Series:

Series is like one dimensional array like other languages. It can store any data type and it have an index this is by default in numeric value.

➢ Create Series:

- First we have to import pandas library, Then create a series just like the example.



- We can store any type of value in series and also assign user-defined labels to the index and use them to access elements of a Series.

```
In [5]:
1 import pandas as pd
2 series2 = pd.Series(["Tanvir","Hridoy","Ahammed","Priya","Saha"], index=[2,3,5,1,4])
3 print(series2)
```

```
2     Tanvir
3     Hridoy
5    Ahammed
1      Priya
4       Saha
dtype: object
```

- Index also can be any data type. In this example I use string as data type in index.

```
In [6]:
1 import pandas as pd
2 series2 = pd.Series([2,3,5,1,4],index=["Tanvir","Hridoy","Ahammed","Priya","Saha"])
3 print(series2)
```

```
Tanvir     2
Hridoy     3
Ahammed    5
Priya      1
Saha       4
dtype: int64
```

➢ Creation of Series from NumPy Arrays:

- NumPy is another Library using in python. We can convert NumPy array (1D) to series just like the example below:

In [7]:
```
1 import numpy as np
2 import pandas as pd
3 array1 = np.array([1,2,3,4,5])
4 series3 = pd.Series(array1)
5 print(series3)
```

```
0    1
1    2
2    3
3    4
4    5
dtype: int32
```

- We can set index value but we have to ensure that index size must be matched with the NumPy array size. If index is not declared is take numeric automatically.

In [8]:
```
1 import numpy as np
2 import pandas as pd
3 array1 = np.array([1,2,3,4,5])
4 series4 = pd.Series(array1, index = ["Sat", "Sun", "Mon", "Tue", "Wed"])
5 print(series4)
```

```
Sat    1
Sun    2
Mon    3
Tue    4
Wed    5
dtype: int32
```

- If the index size is not matched with the array size it throw error just like the example.

```
In [9]:
1 import numpy as np
2 import pandas as pd
3 array1 = np.array([1,2,3,4,5])
4 series4 = pd.Series(array1, index = ["Sat", "Sun", "Mon", "Tue"])
5 print(series4)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_14152\3445760459.py in <module>
      2 import pandas as pd
      3 array1 = np.array([1,2,3,4,5])
----> 4 series4 = pd.Series(array1, index = ["Sat", "Sun", "Mon", "Tue"])
      5 print(series4)

C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\series.py in __init__
(self, data, index, dtype, name, copy, fastpath)
    440                 index = default_index(len(data))
    441             elif is_list_like(data):
--> 442                 com.require_length_match(data, index)
    443
    444             # create/copy the manager

C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\common.py in require_l
ength_match(data, index)
    555     """
    556     if len(data) != len(index):
--> 557         raise ValueError(
    558             "Length of values "
    559             f"({len(data)}) "

ValueError: Length of values (5) does not match length of index (4)
```

➢ Creation of Series from Dictionary:

- Python dictionary has [key:value] pairs, it can be converted into series. Dictionary key is use as index and value is use as value in the series just like the example.

```
In [11]:
1 dict1 = {'Bangladesh': 'Dhaka', 'India': 'NewDelhi', 'UK': 'London', 'Japan'
2 print(dict1)
3 series8 = pd.Series(dict1)
4 print(series8)
5
```

Dictionary Output

```
{'Bangladesh': 'Dhaka', 'India': 'NewDelhi', 'UK': 'London', 'Japan': 'Tokyo'}
Bangladesh       Dhaka
India          NewDelhi
UK               London
Japan             Tokyo
dtype: object
```

Dictionary converted into Series

➢ Accessing Elements of a Series:

- **Indexing:** Indexing in Series is similar to that for NumPy arrays, and is used to access elements in a series. Indexes are of two types: positional index and labelled index.

- **positional index:** Positional index takes an integer value and the series starting from 0 index just like the example.

```
In [12]:  1 seriesNum = pd.Series([10,20,30,40,50])
          2 seriesNum[2]

Out[12]: 30
```

- **labelled index:** Labelled index takes any user-defined label as index just like the example.

```
In [14]:  1 seriesDays = pd.Series([1,2,3,4,5],index=["Sat", "Sun", "Mon", "Tue", "Wed"]
          2 seriesDays["Tue"]

Out[14]: 4
```

- This is another example of labelled index.

```
In [17]:  1 seriesCapCntry = pd.Series(['Dhaka', 'NewDelhi', 'WashingtonDC', 'London', '
          2 index=['Bangladesh','India', 'USA', 'UK', 'France'])
          3 seriesCapCntry['Bangladesh']

Out[17]: 'Dhaka'
```

- We can also access an element of the series using the positional index 3 and 2 positions value is showed here.

```
In [21]: 1 seriesCapCntry[[3,2]]

Out[21]: UK           London
         USA    WashingtonDC
         dtype: object
```

- We simply can access the positional value without index.

```
In [22]: 1 seriesCapCntry[2]

Out[22]: 'WashingtonDC'
```

- We can access the series by it index values.

```
In [23]: 1 seriesCapCntry[['UK','USA']]

Out[23]: UK           London
         USA    WashingtonDC
         dtype: object
```

- The index value can be changed of a series and put a new index for the existing series.

```
In [26]: 1 seriesCapCntry.index=[1,2,3,4,5]
         2 seriesCapCntry

Out[26]: 1          Dhaka
         2       NewDelhi
         3   WashingtonDC
         4         London
         5          Paris
         dtype: object
```

➢ Slicing:

- There is a difference between slicing and indexing, in indexing we only can access the value which is given. But in slicing we can access a range for example seriesCapCntry[0:3] we can access 0 to 2 positional index value because 3 use here exclusive.

```
In [28]:
1 seriesCapCntry = pd.Series(['Dhaka', 'NewDelhi', 'WashingtonDC', 'London', '
2 index=['Bangladesh','India', 'USA', 'UK', 'France'])
3 seriesCapCntry[0:3]
```

```
Out[28]: Bangladesh          Dhaka
         India            NewDelhi
         USA          WashingtonDC
         dtype: object
```

- If labelled indexes are used for slicing, then value at the end index label is also included in the output just like the example.

```
In [29]:
1 seriesCapCntry['Bangladesh' : 'USA']
```

```
Out[29]: Bangladesh          Dhaka
         India            NewDelhi
         USA          WashingtonDC
         dtype: object
```

- We can get the series in reverse order just like the example.
  seriesName(starting_index : ending_index : step)

```
In [30]:
1 seriesCapCntry[ : : -1]
```

```
Out[30]: France             Paris
         UK                 London
         USA          WashingtonDC
         India            NewDelhi
         Bangladesh          Dhaka
         dtype: object
```

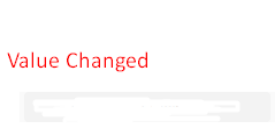
- We can use slicing to modify the series. In the example we use seriesAlpha[1:6]=99 that means from 1 to 5 index the value is updated to 99. Updating the values in a series using slicing excludes the value at the end index position

```
In [40]:
1 import numpy as np
2 seriesAlph = pd.Series(np.arange(10,20,1),
3 index = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j'])
4 seriesAlph

Out[40]: a    10
         b    11
         c    12
         d    13
         e    14
         f    15
         g    16
         h    17
         i    18
         j    19
         dtype: int32

In [41]:
1 seriesAlph[1:6] = 99
2 seriesAlph

Out[41]: a    10
         b    99
         c    99
         d    99
         e    99
         f    99
         g    16
         h    17
         i    18
         j    19
         dtype: int32
```



- We can use labelled index slicing for update values. In this type the end index position is inclusive.

```
In [42]:
1 seriesAlph['c':'g'] = 500
2 seriesAlph

Out[42]:
a     10
b     99
c    500
d    500
e    500
f    500
g    500
h     17
i     18
j     19
dtype: int32
```

Take a screenshot

➢ Attributes of Series:

We can access certain properties called attributes of a series by using that property with the series name.

- We can assign a name of the series just like the example and assign a name to the index of the series.

```
In [43]:
1 seriesCapCntry

Out[43]:
Bangladesh          Dhaka
India             NewDelhi
USA          WashingtonDC
UK                 London
France              Paris
dtype: object

In [45]:
1 seriesCapCntry.name = 'Capitals'
2 print(seriesCapCntry)

Bangladesh          Dhaka
India             NewDelhi
USA          WashingtonDC
UK                 London
France              Paris
Name: Capitals, dtype: object

In [46]:
1 seriesCapCntry.index.name = 'Countries'
2 print(seriesCapCntry)

Countries
Bangladesh          Dhaka
India             NewDelhi
USA          WashingtonDC
UK                 London
France              Paris
Name: Capitals, dtype: object
```

Add Attributes

- We can create a empty series and check it weather the series is empty or not. seriesCapCntry.empty prints True if the series is empty, and False otherwise.

```
In [51]:
1 import pandas as pd
2 seriesEmpt=pd.Series()
3 seriesEmpt.empty

C:\Users\My AsUs\AppData\Local\Temp\ipykernel_14152\3040714642.py:2: FutureWarn
ing: The default dtype for empty Series will be 'object' instead of 'float64' i
n a future version. Specify a dtype explicitly to silence this warning.
  seriesEmpt=pd.Series()

Out[51]: True
```

➢ Methods of Series:

- There are some methods that are available for Pandas Series which give the flex to the user.
- head(n) -> Returns the first n members of the series. If the value for n is not passed, then by default n takes 5 and the first five members are displayed.
- count() -> Returns the number of size of the series. It not include the non-NaN values.
- tail(n) -> Returns the last n members of the series. If the value for n is not passed, then by default n takes 5 and the last five members are displayed.

```
In [54]:
seriesTenTwenty=pd.Series(np.arange( 10,
40, 2 ))
print(seriesTenTwenty)
```

```
0     10
1     12
2     14
3     16
4     18
5     20
6     22
7     24
8     26
9     28
10    30
11    32
12    34
13    36
14    38
dtype: int32
```

```
In [56]:
seriesTenTwenty.head(4)
```

```
Out[56]: 0    10
1    12
2    14
3    16
dtype: int32
```

```
In [57]:
seriesTenTwenty.count()
```

```
Out[57]: 15
```

```
In [58]:
seriesTenTwenty.tail(3)
```

```
Out[58]: 12    34
13    36
14    38
dtype: int32
```

```
In [59]:
seriesTenTwenty.tail()
```

```
Out[59]: 10    30
11    32
12    34
13    36
14    38
dtype: int32
```

```
In [60]:
seriesTenTwenty.head()
```

```
Out[60]: 0    10
1    12
2    14
3    16
4    18
dtype: int32
```

➢ Mathematical Operations on Series:

- **Addition of two Series:**

  We can add two series like [seriesA+seriseB] it will add values based on the index value but if in one series there is not present a index value it will show NaN in the addition.

  But if we don't want to place NaN then we have to use

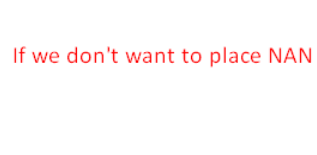
  [seriesA.add(seriesB, fill_value=0)] like that it will add 0 by default where there is absence of value.

```
In [6]:  1 seriesA+seriesB

Out[6]: a     -9.0
        b      NaN
        c    -47.0
        d      NaN
        e    105.0
        y      NaN
        z      NaN
        dtype: float64

In [7]:  1 seriesA.add(seriesB, fill_value=0)

Out[7]: a     -9.0
        b      2.0
        c    -47.0
        d      4.0
        e    105.0
        y     20.0
        z     10.0
        dtype: float64
```



- **Subtraction of two Series:**

  It is same as addition just it will subtract two series values and all the properties as same as addition.

- **Multiplication of two Series:**

  It is same as addition just it will multiplication two series values and all the properties as same as addition.

- **Division of two Series:**

It is same as addition just it will divide two series values and all the properties as same as addition.

```
In [8]:  1 seriesA * seriesB

Out[8]: a    -10.0
        b      NaN
        c   -150.0
        d      NaN
        e    500.0
        y      NaN
        z      NaN
        dtype: float64

In [9]:  1 seriesA.mul(seriesB, fill_value=0)

Out[9]: a    -10.0
        b      0.0
        c   -150.0
        d      0.0
        e    500.0
        y      0.0
        z      0.0
        dtype: float64

In [ ]:  1
```

❖DataFrame:

We learn before about pandas series, but Sometimes we need to work on multiple columns at a time, i.e., we need to process the tabular data. Pandas store such tabular data using a DataFrame.

A DataFrame is a two-dimensional labelled data structure like a table of MySQL. It contains rows and columns, and therefore has both a row and column index. Each column can have different data type value.

➢ Creation of DataFrame:

- In the following example we create a empty DataFrame.

```
In [11]:
1 import pandas as pd
2 dFrameEmt = pd.DataFrame()
3 dFrameEmt

Out[11]:
```

➢ Creation of DataFrame from NumPy n-dimension arrays:

- We can convert NumPy array into DataFrame by simply pass the array into DataFrame [dFrame4 = pd.DataFrame(array1)] .

```
In [16]:
1 import numpy as np
2 array1 = np.array([10,20,30])
3 array2 = np.array([100,200,300])
4 array3 = np.array([-10,-20,-30, -40])
5 dFrame4 = pd.DataFrame(array3)
6 dFrame4

Out[16]:
```

	0
0	-10
1	-20
2	-30
3	-40

- We can create a DataFrame using more than one n-dimension arrays just like the example.

```
In [17]:
1 dFrame5 = pd.DataFrame([array1, array3, array2], columns=[ 'A', 'B', 'C', 'D
2 dFrame5
```

Out[17]:

	A	B	C	D
0	10	20	30	NaN
1	-10	-20	-30	-40.0
2	100	200	300	NaN

➢ Creation of DataFrame from List of Dictionaries:

We can create DataFrame from a list of Dictionaries just like the example.

```
In [18]:
1 listDict = [{'a':10, 'b':20, 'd':50}, {'a':5, 'b':10, 'c':20}]
2 dFrameListDict = pd.DataFrame(listDict)
3 dFrameListDict
```

Out[18]:

	a	b	d	c
0	10	20	50.0	NaN
1	5	10	NaN	20.0

➢ Creation of DataFrame from Dictionary of Lists:

- DataFrames can also be created from a dictionary of lists. Dictionary keys become column labels by default in a DataFrame, and the lists become the rows.
- We can change the sequence of the column labels as like the example.

In [19]:
```
dictForest = {'State': ['Assam', 'Delhi', 'Kerala'],
'GArea': [78438, 1483, 38852] ,
'VDF' : [2797, 6.72,1663]}
dFrameForest= pd.DataFrame(dictForest)
dFrameForest
```

Out[19]:

	State	GArea	VDF
0	Assam	78438	2797.00
1	Delhi	1483	6.72
2	Kerala	38852	1663.00

In [20]:
```
dFrameForest1 = pd.DataFrame(dictForest,
columns = ['State','VDF', 'GArea'])
dFrameForest1
```

Out[20]:

	State	VDF	GArea
0	Assam	2797.00	78438
1	Delhi	6.72	1483
2	Kerala	1663.00	38852

➢ Creation of DataFrame from Series:

- We can combine multiple series to a DataFrame. Here are three series seriesA, seriesB, seriesC we convert them into dFrame8.

In [22]:
```
seriesA = pd.Series([1,2,3,4,5],
 index = ['a', 'b', 'c', 'd', 'e'])
seriesB = pd.Series ([1000,2000,-1000,-5000,1000],
 index = ['a', 'b', 'c', 'd', 'e'])
seriesC = pd.Series([10,20,-10,-50,100],
 index = ['z', 'y', 'a', 'c', 'e'])
dFrame8 = pd.DataFrame([seriesA, seriesA, seriesC])
dFrame8
```

Out[22]:

	a	b	c	d	e	z	y
0	1.0	2.0	3.0	4.0	5.0	NaN	NaN
1	1.0	2.0	3.0	4.0	5.0	NaN	NaN
2	-10.0	NaN	-50.0	NaN	100.0	10.0	20.0

➢ Creation of DataFrame from Dictionary of Series:

- A dictionary of series can also be used to create a DataFrame.In the example ResultSheet is a dictionary with 5 student as column and 3 subject as index.

In [23]:

```
ResultSheet={'Arnab': pd.Series([90, 91, 97],
 index=['Maths','Science','Hindi']),
'Ramit': pd.Series([92, 81, 96],
 index=['Maths','Science','Hindi']),
'Samridhi': pd.Series([89, 91, 88],
 index=['Maths','Science','Hindi']),
'Riya': pd.Series([81, 71, 67],
 index=['Maths','Science','Hindi']),
'Mallika': pd.Series([94, 95, 99],
 index=['Maths','Science','Hindi'])}
ResultDF = pd.DataFrame(ResultSheet)
ResultDF
```

Out[23]:

	Arnab	Ramit	Samridhi	Riya	Mallika
Maths	90	92	89	81	94
Science	91	81	91	71	95
Hindi	97	96	88	67	99

- If an individual dictionary element doesn't contain any value it will put NaN to that position.

In [27]:

```
ResultSheet={'Arnab': pd.Series([90, 91, 97],
 index=['Maths','Science','Hindi']),
'Ramit': pd.Series([92, 96],
 index=['Maths','Hindi']),
'Samridhi': pd.Series([89, 91, 88],
 index=['Maths','Science','Hindi']),
'Riya': pd.Series([81, 71, 67],
 index=['Maths','Science','Hindi']),
'Mallika': pd.Series([94, 95, 99],
 index=['Maths','Science','Hindi'])}
ResultDF = pd.DataFrame(ResultSheet)
ResultDF
```

NaN

Out[27]:

	Arnab	Ramit	Samridhi	Riya	Mallika
Hindi	97	96.0	88	67	99
Maths	90	92.0	89	81	94
Science	91	NaN	91	71	95

➢ Operations on rows and columns in DataFrames

- **Adding a New Column to a DataFrame:**

  We can add a new column in the DataFrame as like the example.

Out[27]:

	Arnab	Ramit	Samridhi	Riya	Mallika
Hindi	97	96.0	88	67	99
Maths	90	92.0	89	81	94
Science	91	NaN	91	71	95

In [28]:
```
1 ResultDF['Preeti']=[89,78,76]
2 ResultDF
```

Out[28]:

	Arnab	Ramit	Samridhi	Riya	Mallika	Preeti
Hindi	97	96.0	88	67	99	89
Maths	90	92.0	89	81	94	78
Science	91	NaN	91	71	95	76

- If we assign value in the existing column name the column value will be modified, it will not create a new column at the end.

Out[28]:

	Arnab	Ramit	Samridhi	Riya	Mallika	Preeti
Hindi	97	96.0	88	67	99	89
Maths	90	92.0	89	81	94	78
Science	91	NaN	91	71	95	76

In [29]:
```
1 ResultDF['Ramit']=[99, 98, 78]
2 ResultDF
```

Out[29]:

	Arnab	Ramit	Samridhi	Riya	Mallika	Preeti
Hindi	97	99	88	67	99	89
Maths	90	98	89	81	94	78
Science	91	78	91	71	95	76

- **Adding a New Row to a DataFrame:**

  We can add a new row to a DataFrame using the DataFrame.loc[ ] method. In the example, we add a new row which is English.

Out[29]:

	Arnab	Ramit	Samridhi	Riya	Mallika	Preeti
Hindi	97	99	88	67	99	89
Maths	90	98	89	81	94	78
Science	91	78	91	71	95	76

In [30]:

```
1 ResultDF.loc['English'] = [85, 86, 83, 80, 90, 89]
2 ResultDF
```

Out[30]:

	Arnab	Ramit	Samridhi	Riya	Mallika	Preeti
Hindi	97	99	88	67	99	89
Maths	90	98	89	81	94	78
Science	91	78	91	71	95	76
English	85	86	83	80	90	89

- We can set all the value of the DataFrame into one value as ResultDF[:] = Value. In the example we converted all value into 0.

Out[31]:

	Arnab	Ramit	Samridhi	Riya	Mallika	Preeti
Hindi	97	99	88	67	99	89
Maths	90	98	89	81	94	78
Science	91	78	91	71	95	76
English	95	86	95	80	95	99

In [34]:
```
1 ResultDF[: ] = 0
2 ResultDF
```

Out[34]:

	Arnab	Ramit	Samridhi	Riya	Mallika	Preeti
Hindi	0	0	0	0	0	0
Maths	0	0	0	0	0	0
Science	0	0	0	0	0	0
English	0	0	0	0	0	0

- ## Deleting Rows or Columns from a DataFrame:

We can use the DataFrame.drop() method to delete rows and columns from a DataFrame. If we put axis value is 0 it will delete the specified row on the other had putting axis value 1 it will delete specified column.

In [36]:
```
1 ResultDF = ResultDF.drop('Science', axis=0)
2 ResultDF
```

Out[36]:

	Arnab	Ramit	Samridhi	Riya	Mallika	Preeti
Hindi	0	0	0	0	0	0
Maths	0	0	0	0	0	0
English	0	0	0	0	0	0

for row delete axis=0

In [37]:
```
1 ResultDF = ResultDF.drop(['Samridhi','Ramit'], axis=1)
2 ResultDF
```

Out[37]:

	Arnab	Riya	Mallika	Preeti
Hindi	0	0	0	0
Maths	0	0	0	0
English	0	0	0	0

for column delete axis=1

- **Renaming Row Labels of a DataFrame:**

  We can change the labels of rows and columns in a DataFrame using the DataFrame.rename() method. In the following example Hindi, Maths, English, Bangla to sub1, sub2, sub3, sub4. In the axis field we have to put the value 'index' to rename row.

```
In [42]:
1 ResultDF
```

Out[42]:

	Arnab	Riya	Mallika	Preeti
Hindi	0	0	0	0
Maths	0	0	0	0
English	0	0	0	0
Bangla	87	86	95	99

```
In [46]:
1 ResultDF=ResultDF.rename({'Hindi':'Sub1', 'Maths':'Sub2',
2 'English':'Sub3',
3 'Bangla':'Sub4'}, axis='index')
4 print(ResultDF)
```

```
      Arnab  Riya  Mallika  Preeti
Sub1      0     0        0       0
Sub2      0     0        0       0
Sub3      0     0        0       0
Sub4     87    86       95      99
```

- We can choose which row name I want to change. If I don't want change any row name we have to leave just as it is.

Out[47]:

	Arnab	Ramit	Samridhi	Riya	Mallika
Maths	90	92	89	81	94
Science	91	85	91	71	95
Hindi	97	96	88	67	99

```
In [48]:
1 ResultDF=ResultDF.rename({'Maths':'Sub1','Hindi':'Sub3'}, axis='index')
2 print(ResultDF)
```

```
         Arnab  Ramit  Samridhi  Riya  Mallika
Sub1        90     92        89    81       94
Science     91     85        91    71       95
Sub3        97     96        88    67       99
```

- **Renaming Column Labels of a DataFrame:**
  We can alter the column name in a DataFrame using the DataFrame.rename() method. In the axis field we have to put the value 'columns' to rename column.

```
         Arnab  Ramit  Samridhi  Riya  Mallika
Sub1        90     92        89    81       94
Science     91     85        91    71       95
Sub3        97     96        88    67       99
```

```
In [51]:
1 ResultDF=ResultDF.rename({'Arnab':'Student1','Ramit':'Student2',
2 'Samridhi':'Student3','Mallika':'Student4'},axis='columns')
3 print(ResultDF)
```

```
         Student1  Student2  Student3  Riya  Student4
Sub1           90        92        89    81        94
Science        91        85        91    71        95
Sub3           97        96        88    67        99
```

➢ Accessing DataFrames Element through Indexing

- **Label Based Indexing:**

  DataFrame.loc[ ] is an important method that is used for label based indexing with DataFrames. In the example ResultDF.loc['science'] will show Science result of all the students.

```
         Student1  Student2  Student3  Riya  Student4
Sub1           90        92        89    81        94
Science        91        85        91    71        95
Sub3           97        96        88    67        99
```

```
In [52]:
1 ResultDF.loc['Science']
```

```
Out[52]: Student1    91
         Student2    85
         Student3    91
         Riya        71
         Student4    95
         Name: Science, dtype: int64
```

- When a single column label is passed, it returns the column as a Series. In the example it will show Riya result in list format.

```
In [54]: 1 ResultDF
```

Out[54]:

	Student1	Student2	Student3	Riya	Student4
Sub1	90	92	89	81	94
Science	91	85	91	71	95
Sub3	97	96	88	67	99

```
In [55]: 1 ResultDF.loc[:,'Riya']
```

```
Out[55]: Sub1       81
         Science    71
         Sub3       67
         Name: Riya, dtype: int64
```

- **Boolean Indexing:**

  Boolean means a binary variable that can be either True or False. In the following example if the student result is greater than 90 it will show True otherwise False.

Out[57]:

	Arnab	Ramit	Samridhi	Riya	Mallika
Maths	90	92	89	81	94
Science	91	85	91	71	95
Hindi	97	96	88	67	99

```
In [58]: 1 ResultDF.loc['Maths'] > 90
```

```
Out[58]: Arnab       False
         Ramit        True
         Samridhi    False
         Riya        False
         Mallika      True
         Name: Maths, dtype: bool
```

- To check in which subjects ‘Arnab’ has scored more than 90, we can write:

Out[60]:

	Arnab	Ramit	Samridhi	Riya	Mallika
Maths	90	92	89	81	94
Science	91	85	91	71	95
Hindi	97	96	88	67	99

In [62]:
```
1  ResultDF.loc[:,'Arnab']>90
```

Out[62]:
```
Maths      False
Science     True
Hindi       True
Name: Arnab, dtype: bool
```

➢ Accessing DataFrames Element through Slicing:

- We can use slicing to select a subset of rows and/or columns from a DataFrame. DataFrames slicing is inclusive of the end values. In the example it will take row from Maths to Science.

In [63]:
```
1  ResultDF
```

Out[63]:

	Arnab	Ramit	Samridhi	Riya	Mallika
Maths	90	92	89	81	94
Science	91	85	91	71	95
Hindi	97	96	88	67	99

In [64]:
```
1  ResultDF.loc['Maths': 'Science']
```

Out[64]:

	Arnab	Ramit	Samridhi	Riya	Mallika
Maths	90	92	89	81	94
Science	91	85	91	71	95

- We may use a slice of labels with a slice of column names to access values of those rows and columns:

	Arnab	Ramit	Samridhi	Riya	Mallika
Maths	90	92	89	81	94
Science	91	85	91	71	95
Hindi	97	96	88	67	99

```
In [64]: 1 ResultDF.loc['Maths': 'Science']
```

Out[64]:

	Arnab	Ramit	Samridhi	Riya	Mallika
Maths	90	92	89	81	94
Science	91	85	91	71	95

```
In [65]: 1 ResultDF.loc['Maths': 'Science', 'Arnab':'Samridhi']
```

Out[65]:

	Arnab	Ramit	Samridhi
Maths	90	92	89
Science	91	85	91

- **Filtering Rows in DataFrames:**

In DataFrames DataFrmae.loc[] method can be use as rows filtering. True(1) means it will show the row if it False(0) it will not show the row.

```
In [66]: 1 ResultDF
```

Out[66]:

	Arnab	Ramit	Samridhi	Riya	Mallika
Maths	90	92	89	81	94
Science	91	85	91	71	95
Hindi	97	96	88	67	99

This Row is deleted

```
In [67]: 1 ResultDF.loc[[True, False, True]]
```

Out[67]:

	Arnab	Ramit	Samridhi	Riya	Mallika
Maths	90	92	89	81	94
Hindi	97	96	88	67	99

➢ Joining, Merging and Concatenation of DataFrames

- **Joining:**

  We can use the pandas.DataFrame.append() method to merge two DataFrames. It appends rows of the second DataFrame at the end of the first DataFrame. If there the second DataFrame column is not present in the first DataFrame it will add new column.

In [68]:
```
dFrame1=pd.DataFrame([[1, 2, 3], [4, 5], [6]],
columns=['C1', 'C2', 'C3'], index=['R1', 'R2', 'R3'])
dFrame1
```

Out[68]:

	C1	C2	C3
R1	1	2.0	3.0
R2	4	5.0	NaN
R3	6	NaN	NaN

In [69]:
```
dFrame2=pd.DataFrame([[10, 20], [30], [40, 50]],
columns=['C2', 'C5'], index=['R4', 'R2', 'R5'])
dFrame2
```

Out[69]:

	C2	C5
R4	10	20.0
R2	30	NaN
R5	40	50.0

- In the example we merge dFrame1 with dFrame2 and display it.

In [72]:
```
dFrame1=dFrame1.append(dFrame2)
dFrame1
```

```
C:\Users\My AsUs\AppData\Local\Temp\ipykernel_12260\214336498.py:1: FutureWarning: The frame.append method is deprecated and wi
ll be removed from pandas in a future version. Use pandas.concat instead.
  dFrame1=dFrame1.append(dFrame2)
```

Out[72]:

	C1	C2	C3	C5
R1	1.0	2.0	3.0	NaN
R2	4.0	5.0	NaN	NaN
R3	6.0	NaN	NaN	NaN
R4	NaN	10.0	NaN	20.0
R2	NaN	30.0	NaN	NaN
R5	NaN	40.0	NaN	50.0
R4	NaN	10.0	NaN	20.0
R2	NaN	30.0	NaN	NaN
R5	NaN	40.0	NaN	50.0

In [ ]:

- In the previous example the column level is not sorted order, if we want to sort the join DataFrame in column order we can set the parameter sort=True.

In [79]:
```
dFrame2 =dFrame2.append(dFrame1, sort='True')
dFrame2
```

```
C:\Users\My AsUs\AppData\Local\Temp\ipykernel_12260\984493915.py:1: FutureWarning: The frame.append method is deprecated and wi
ll be removed from pandas in a future version. Use pandas.concat instead.
  dFrame2 =dFrame2.append(dFrame1, sort='True')
C:\Users\My AsUs\AppData\Local\Temp\ipykernel_12260\984493915.py:1: FutureWarning: Passing non boolean values for sort is depre
cated and will error in a future version!
  dFrame2 =dFrame2.append(dFrame1, sort='True')
```

Out[79]:

	C1	C2	C3	C5
R4	NaN	10.0	NaN	20.0
R2	NaN	30.0	NaN	NaN
R5	NaN	40.0	NaN	50.0
R1	1.0	2.0	3.0	NaN
R2	4.0	5.0	NaN	NaN
R3	6.0	NaN	NaN	NaN

- If we don't want to sort the Dataframe in column level we can set the parameter sort=False.

In [80]:
```
dFrame2 =dFrame2.append(dFrame1, sort='False')
dFrame2
```

```
C:\Users\My AsUs\AppData\Local\Temp\ipykernel_12260\3897692422.py:1: FutureWarning: The frame.append method is deprecated and w
ill be removed from pandas in a future version. Use pandas.concat instead.
  dFrame2 =dFrame2.append(dFrame1, sort='False')
C:\Users\My AsUs\AppData\Local\Temp\ipykernel_12260\3897692422.py:1: FutureWarning: Passing non boolean values for sort is depr
ecated and will error in a future version!
  dFrame2 =dFrame2.append(dFrame1, sort='False')
```

Out[80]:

	C1	C2	C3	C5
R4	NaN	10.0	NaN	20.0
R2	NaN	30.0	NaN	NaN
R5	NaN	40.0	NaN	50.0
R1	1.0	2.0	3.0	NaN
R2	4.0	5.0	NaN	NaN
R3	6.0	NaN	NaN	NaN
R1	1.0	2.0	3.0	NaN
R2	4.0	5.0	NaN	NaN
R3	6.0	NaN	NaN	NaN

- when we do not want to use row index labels we can set ignore_index =True. By default in the append function ignore_index = False.

In [81]:
```
dFrame1 = dFrame1.append(dFrame2, ignore_index=True)
dFrame1
```

```
C:\Users\My AsUs\AppData\Local\Temp\ipykernel_12260\2450925970.py:1: FutureWarning: The frame.append method is deprecated and w
ill be removed from pandas in a future version. Use pandas.concat instead.
  dFrame1 = dFrame1.append(dFrame2, ignore_index=True)
```

Out[81]:

	C1	C2	C3	C5
0	1.0	2.0	3.0	NaN
1	4.0	5.0	NaN	NaN
2	6.0	NaN	NaN	NaN
3	NaN	10.0	NaN	20.0
4	NaN	30.0	NaN	NaN
5	NaN	40.0	NaN	50.0
6	1.0	2.0	3.0	NaN
7	4.0	5.0	NaN	NaN
8	6.0	NaN	NaN	NaN
9	1.0	2.0	3.0	NaN
10	4.0	5.0	NaN	NaN
11	6.0	NaN	NaN	NaN

➢ Attributes of DataFrames:

In [2]:
```
import pandas as pd
ForestArea = {
'Assam' :pd.Series([78438, 2797, 10192, 15116],
index = ['GeoArea', 'VeryDense', 'ModeratelyDense', 'OpenForest']),
'Kerala' :pd.Series([ 38852, 1663, 9407, 9251],
index = ['GeoArea' ,'VeryDense', 'ModeratelyDense', 'OpenForest']),
'Delhi' :pd.Series([1483, 6.72, 56.24, 129.45],
index = ['GeoArea', 'VeryDense', 'ModeratelyDense', 'OpenForest'])}
ForestAreaDF = pd.DataFrame(ForestArea)
ForestAreaDF
```

Out[2]:

	Assam	Kerala	Delhi
GeoArea	78438	38852	1483.00
VeryDense	2797	1663	6.72
ModeratelyDense	10192	9407	56.24
OpenForest	15116	9251	129.45

- If we want to transpose the DataFrame we can use [DataFrame.T]. Means, row indices and column labels of the DataFrame replace each other's position

Out[2]:

	Assam	Kerala	Delhi
GeoArea	78438	38852	1483.00
VeryDense	2797	1663	6.72
ModeratelyDense	10192	9407	56.24
OpenForest	15116	9251	129.45

In [17]:
```
1 ForestAreaDF.T
```

Out[17]:

	GeoArea	VeryDense	ModeratelyDense	OpenForest
Assam	78438.0	2797.00	10192.00	15116.00
Kerala	38852.0	1663.00	9407.00	9251.00
Delhi	1483.0	6.72	56.24	129.45

- If we want to display the first n row we can use [DataFrame.head(n)]. In the same way, to display the last n row we can use [DataFrame.tail (n)].

Out[18]:

	Assam	Kerala	Delhi
GeoArea	78438	38852	1483.00
VeryDense	2797	1663	6.72
ModeratelyDense	10192	9407	56.24
OpenForest	15116	9251	129.45

In [19]:
```
1 ForestAreaDF.head(2)
```

Out[19]:

	Assam	Kerala	Delhi
GeoArea	78438	38852	1483.00
VeryDense	2797	1663	6.72

In [21]:
```
1 ForestAreaDF.tail(2)
```

Out[21]:

	Assam	Kerala	Delhi
ModeratelyDense	10192	9407	56.24
OpenForest	15116	9251	129.45

- [DataFrame.empty] return a Boolean value if the DataFrame is empty it return True otherwise False.

In [22]:
```
1 ForestAreaDF.empty
```

Out[22]: False

In [23]:
```
1 df=pd.DataFrame()
2 df.empty
```

Out[23]: True

- DataFrame.size display the size or total number of tuples in the DataFrame.
- DataFrame.shape display the number of rows and number of columns.

```
In [24]: 1 ForestAreaDF.size
Out[24]: 12

In [25]: 1 ForestAreaDF.shape
Out[25]: (4, 3)
```

- DataFrame.values display all the values in the DataFrame without the axes labels.
- DataFrame.dtypes display the data type of each column in the DataFrame.

```
In [26]: 1 ForestAreaDF.values
Out[26]: array([[7.8438e+04, 3.8852e+04, 1.4830e+03],
                [2.7970e+03, 1.6630e+03, 6.7200e+00],
                [1.0192e+04, 9.4070e+03, 5.6240e+01],
                [1.5116e+04, 9.2510e+03, 1.2945e+02]])

In [27]: 1 ForestAreaDF.dtypes
Out[27]: Assam       int64
         Kerala      int64
         Delhi     float64
         dtype: object
```

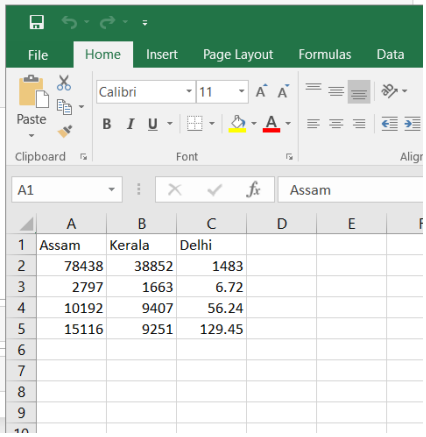
➢ Exporting a DataFrame to a CSV file:

- We can use the to_csv() function to save a DataFrame to a text or csv file. In the following example we convert the ForestAreaDF DataFrame to csv file.

```
9407, 9251], index = ['GeoArea' ,'VeryDense',
'ModeratelyDense', 'OpenForest']),
 'Delhi' :pd.Series([1483, 6.72, 56.24,
129.45], index = ['GeoArea', 'VeryDense',
'ModeratelyDense', 'OpenForest'])}
>>> ForestAreaDF = pd.DataFrame(ForestArea)
>>> ForestAreaDF
```

28]:

	Assam	Kerala	Delhi
GeoArea	78438	38852	1483.00
VeryDense	2797	1663	6.72
ModeratelyDense	10192	9407	56.24
OpenForest	15116	9251	129.45



46]:
```
ForestAreaDF.to_csv('C:\src/filename.csv', index = False, encoding='utf-8')
```

]:

In [46]:
```
ForestAreaDF.to_csv('C:\src/filename.csv', index = False, encoding='utf-8')
```

In [48]:
```
ForestArea = pd.read_csv("C:\src/filename.csv",sep =",", header=0)
ForestArea
```

Out[48]:

	Assam	Kerala	Delhi
0	78438	38852	1483.00
1	2797	1663	6.72
2	10192	9407	56.24
3	15116	9251	129.45

➢ Importing a CSV file to a DataFrame:

- We can load the data from the ResultData.csv file into a DataFrame, In the example using Pandas read_csv() function as shown below:

In [49]:
```
ForestArea = pd.read_csv("C:\src/filename.csv",sep =",", names=['Area1', 'Area2', 'Area3'])
ForestArea
```

Out[49]:

	Area1	Area2	Area3
0	Assam	Kerala	Delhi
1	78438	38852	1483.0
2	2797	1663	6.72
3	10192	9407	56.24
4	15116	9251	129.45

Pandas (Part 2)

As discussed in before part(part 1) about pandas two primary data structure series and dataframe and basic operation on them like creating and accessing data from them.
In this part, we will discuss about more advanced features of dataframe, like sorting data, answering analytical questions using data, cleaning data and applying different useful functions on the data.

❖ Create dataframe:

For store the result data in dataframe we first create a dataframe from a dictionary of list using pandas.

Example:

In [36]:
```
import pandas as pd
result={'Name':['priya','priya','hridoy','hridoy','tanvir','tanvir','susmita','susmita'],
        'ut':[1,2,1,2,1,2,1,2],
        'Math':[90,91,92,91,94,92,90,91],
        'stat':[80,81,82,81,83,82,81,80],
        'DM':[90,91,92,93,91,92,93,91]}
df=pd.DataFrame(result)
print("The Result table is:")
print(df)
```

```
The Result table is:
      Name  ut  Math  stat  DM
0    priya   1    90    80  90
1    priya   2    91    81  91
2   hridoy   1    92    82  92
3   hridoy   2    91    81  93
4   tanvir   1    94    83  91
5   tanvir   2    92    82  92
6  susmita   1    90    81  93
7  susmita   2    91    80  91
```

❖ Descriptive Statistics:

Descriptive statistics are used to summarize the given data. We will applied statistical method to a DataFrame. These are –

i. Max
ii. Min
iii. Count
iv. Sum

v. Mean
vi. Median
vii. Mode
viii. Quartiles
ix. Variance
x. Standard deviation

➢ Some parameters for statistical methods:

- **Numerical_only:**

  If we want to find the maximum value for the column that have numeric numbers than we have to set numerical_only=True in these method.

  Syntax: df.max(numerical_only=True)

  Example:

```
In [39]: import pandas as pd
result={'Name':['priya','priya','hridoy','hridoy','tanvir','tanvir','susmita','susmita'],
        'ut':[1,2,1,2,1,2,1,2],
        'Math':[90,91,92,91,94,92,90,91],
        'stat':[80,81,82,81,83,82,81,80],
        'DM':[90,91,92,93,91,92,93,91]}
df=pd.DataFrame(result)
print("The Result table is:")
print(df)
print("\nMaximum values are:\n",df.max(numeric_only=True))
```

```
The Result table is:
      Name  ut  Math  stat  DM
0    priya   1    90    80  90
1    priya   2    91    81  91
2   hridoy   1    92    82  92
3   hridoy   2    91    81  93
4   tanvir   1    94    83  91
5   tanvir   2    92    82  92
6  susmita   1    90    81  93
7  susmita   2    91    80  91

Maximum values are:
 ut       2
Math    94
stat    83
DM      93
dtype: int64
```

- **Relational operators:**

  If we want to calculate max value based on specific condition than we can use relational operator and apply methods.

  Syntax: df2=df[df[‘ut’]==2].max(numerical_only=True)
  print(df2)

or,

df2=df[df.ut==2]

df2.max(numerical_only=True)

or,

df[‘Maths’].min()

Example 1: Find the max marks of unit test(ut)=2

```
In [40]: import pandas as pd
result={'Name':['priya','priya','hridoy','hridoy','tanvir','tanvir','susmita','susmita'],
        'ut':[1,2,1,2,1,2,1,2],
        'Math':[90,91,92,91,94,92,90,91],
        'stat':[80,81,82,81,83,82,81,80],
        'DM':[90,91,92,93,91,92,93,91]}
df=pd.DataFrame(result)
print("The Result table is:")
print(df)
df2=df[df.ut==2]
print("\nThe Result of unit test 2:")
print(df2)
print("\nMaximum values are:\n",df2.max(numeric_only=True))
```

```
The Result table is:
      Name  ut  Math  stat  DM
0    priya   1    90    80  90
1    priya   2    91    81  91
2   hridoy   1    92    82  92
3   hridoy   2    91    81  93
4   tanvir   1    94    83  91
5   tanvir   2    92    82  92
6  susmita   1    90    81  93
7  susmita   2    91    80  91

The Result of unit test 2:
      Name  ut  Math  stat  DM
1    priya   2    91    81  91
3   hridoy   2    91    81  93
5   tanvir   2    92    82  92
7  susmita   2    91    80  91


Maximum values are:
 ut       2
Math    92
stat    82
DM      93
dtype: int64
```

Example 2: find min marks obtain by susmita in each subject .

In [50]:
```
import pandas as pd
result={'Name':['priya','priya','hridoy','hridoy','tanvir','tanvir','susmita','susmita'],
        'ut':[1,2,1,2,1,2,1,2],
        'Math':[90,91,92,91,94,92,90,91],
        'Stat':[80,81,82,81,83,82,81,80],
        'DM':[90,91,92,93,91,92,93,91]}
df=pd.DataFrame(result)
df2=df.loc[df.Name=='susmita']
print("\nThe Result of Susmita:")
print(df2)
print('\nMinimum values are:\n',df2[['Math','Stat','DM']].min(numeric_only=True))
```

```
The Result of Susmita:
      Name  ut  Math  Stat  DM
6  susmita   1    90    81  93
7  susmita   2    91    80  91

Minimum values are:
 Math    90
Stat     80
DM       91
dtype: int64
```

- **Axis:**

  Calculate maximum value row wise then use axis=1, if column wise then use axis =0

  Syntax: df.max(axis=1)

  Example:

```
In [41]: import pandas as pd
result={'Name':['priya','priya','hridoy','hridoy','tanvir','tanvir','susmita','susmita'],
        'ut':[1,2,1,2,1,2,1,2],
        'Math':[90,91,92,91,94,92,90,91],
        'stat':[80,81,82,81,83,82,81,80],
        'DM':[90,91,92,93,91,92,93,91]}
df=pd.DataFrame(result)
print("The Result table is:")
print(df)
print("\nMaximum values are:\n",df.max(numeric_only=True,axis=1))
```

```
The Result table is:
      Name  ut  Math  stat  DM
0    priya   1    90    80  90
1    priya   2    91    81  91
2   hridoy   1    92    82  92
3   hridoy   2    91    81  93
4   tanvir   1    94    83  91
5   tanvir   2    92    82  92
6  susmita   1    90    81  93
7  susmita   2    91    80  91

Maximum values are:
 0    90
1    91
2    92
3    93
4    94
5    92
6    93
7    91
dtype: int64
```

➢ Calculate Maximum values:

If we want to calculate maximum value for each column then we can simply use max function.

Syntax: dataframe.max()

Example:

In [37]:
```
import pandas as pd
result={'Name':['priya','priya','hridoy','hridoy','tanvir','tanvir','susmita','susmita'],
        'ut':[1,2,1,2,1,2,1,2],
        'Math':[90,91,92,91,94,92,90,91],
        'stat':[80,81,82,81,83,82,81,80],
        'DM':[90,91,92,93,91,92,93,91]}
df=pd.DataFrame(result)
print("The Result table is:")
print(df)
print("\nMaximum values are:\n",df.max())
```

```
The Result table is:
      Name  ut  Math  stat  DM
0    priya   1    90    80  90
1    priya   2    91    81  91
2   hridoy   1    92    82  92
3   hridoy   2    91    81  93
4   tanvir   1    94    83  91
5   tanvir   2    92    82  92
6  susmita   1    90    81  93
7  susmita   2    91    80  91

Maximum values are:
 Name    tanvir
ut           2
Math        94
stat        83
DM          93
dtype: object
```

➢ Calculate Maximum values:

If we want to calculate minimum value for each column then we can simply use min function.

Syntax: dataframe.min()

Example:

In [38]:
```
import pandas as pd
result={'Name':['priya','priya','hridoy','hridoy','tanvir','tanvir','susmita','susmita'],
        'ut':[1,2,1,2,1,2,1,2],
        'Math':[90,91,92,91,94,92,90,91],
        'stat':[80,81,82,81,83,82,81,80],
        'DM':[90,91,92,93,91,92,93,91]}
df=pd.DataFrame(result)
print("The Result table is:")
print(df)
print("\nMinimum values are:\n",df.min())
```

```
The Result table is:
      Name  ut  Math  stat  DM
0    priya   1    90    80  90
1    priya   2    91    81  91
2   hridoy   1    92    82  92
3   hridoy   2    91    81  93
4   tanvir   1    94    83  91
5   tanvir   2    92    82  92
6  susmita   1    90    81  93
7  susmita   2    91    80  91

Minimum values are:
 Name    hridoy
ut           1
Math        90
stat        80
DM          90
dtype: object
```

➢ Calculate sum of values:

We can calculate sum of each column.

Syntax: df.sum()

We can also use parameters like numerical_only ,axis or relational operator.

Example: Calculate sum for specific entity for each sub only.

In [51]:
```
import pandas as pd
result={'Name':['priya','priya','hridoy','hridoy','tanvir','tanvir','susmita','susmita'],
        'ut':[1,2,1,2,1,2,1,2],
        'Math':[90,91,92,91,94,92,90,91],
        'Stat':[80,81,82,81,83,82,81,80],
        'DM':[90,91,92,93,91,92,93,91]}
df=pd.DataFrame(result)
df2=df.loc[df.Name=='susmita']
print("\nThe Result of Susmita:")
print(df2)
print('\nTotal marks in each subject is:\n',df2[['Math','Stat','DM']].sum())
```

```
The Result of Susmita:
      Name  ut  Math  Stat  DM
6  susmita   1    90    81  93
7  susmita   2    91    80  91

Total marks in each subject is:
 Math    181
Stat     161
DM       184
dtype: int64
```

➢ Calculate Number of values:

For calculate total number of values in each column or row than use count method. Can use parameters.

Syntax: df.count()

Example:

```
In [52]: import pandas as pd
result={'Name':['priya','priya','hridoy','hridoy','tanvir','tanvir','susmita','susmita'],
        'ut':[1,2,1,2,1,2,1,2],
        'Math':[90,91,92,91,94,92,90,91],
        'Stat':[80,81,82,81,83,82,81,80],
        'DM':[90,91,92,93,91,92,93,91]}
df=pd.DataFrame(result)
print("The Result table is:")
print(df)
print('\nNumber of values in each row is:\n',df.count(axis=1))
```

```
The Result table is:
      Name  ut  Math  Stat  DM
0    priya   1    90    80  90
1    priya   2    91    81  91
2   hridoy   1    92    82  92
3   hridoy   2    91    81  93
4   tanvir   1    94    83  91
5   tanvir   2    92    82  92
6  susmita   1    90    81  93
7  susmita   2    91    80  91

Number of values in each row is:
 0    5
1    5
2    5
3    5
4    5
5    5
6    5
7    5
dtype: int64
```

➢ Calculate mean:

If we want to calculate the mean (average) of each column or row then use mean method. We can use parameters.

Syntax: df.mean()

Example:

```
In [54]: import pandas as pd
result={'Name':['priya','priya','hridoy','hridoy','tanvir','tanvir','susmita','susmita'],
        'ut':[1,2,1,2,1,2,1,2],
        'Math':[90,91,92,91,94,92,90,91],
        'Stat':[80,81,82,81,83,82,81,80],
        'DM':[90,91,92,93,91,92,93,91]}
df=pd.DataFrame(result)
df2=df[df.Name=='susmita']
print("\nThe Result of Susmita:")
print(df2)
print('\nAverage of marks obtain by Susmita:\n',df2[['Math','Stat','DM']].mean())
```

```
The Result of Susmita:
      Name  ut  Math  Stat  DM
6  susmita   1    90    81  93
7  susmita   2    91    80  91

Average of marks obtain by Susmita:
 Math    90.5
Stat    80.5
DM      92.0
dtype: float64
```

➢ Calculate median:

If we want to calculate the middle value of each column or row then use medin method. We can use parameters.

Syntax: df.median()

Example:

```
In [55]: import pandas as pd
result={'Name':['priya','priya','hridoy','hridoy','tanvir','tanvir','susmita','susmita'],
        'ut':[1,2,1,2,1,2,1,2],
        'Math':[90,91,92,91,94,92,90,91],
        'Stat':[80,81,82,81,83,82,81,80],
        'DM':[90,91,92,93,91,92,93,91]}
df=pd.DataFrame(result)
df1=df['Math']
df2=df1[df.ut==1]
print("\nThe Result of math in ut 1:")
print(df2)
print('\nMedian of math in ut 1:\n',df2.median())
```

```
The Result of math in ut 1:
0    90
2    92
4    94
6    90
Name: Math, dtype: int64

Median of math in ut 1:
 91.0
```

➢ Calculate mode:

If we want to calculate the value that is appears most numbers of times in data of each column or row then use mode method. We can use parameters.

Syntax: df.mode()

Example:

```
In [55]: import pandas as pd
result={'Name':['priya','priya','hridoy','hridoy','tanvir','tanvir','susmita','susmita'],
        'ut':[1,2,1,2,1,2,1,2],
        'Math':[90,91,92,91,94,92,90,91],
        'Stat':[80,81,82,81,83,82,81,80],
        'DM':[90,91,92,93,91,92,93,91]}
df=pd.DataFrame(result)
df1=df['Math']
df2=df1[df.ut==1]
print("\nThe Result of math in ut 1:")
print(df2)
print('\nMedian of math in ut 1:\n',df2.median())
```

```
The Result of math in ut 1:
0    90
2    92
4    94
6    90
Name: Math, dtype: int64

Median of math in ut 1:
 91.0
```

➢ Calculate quartile:

If we want to calculate the quartile value of each column or row then use quantile method. We can use parameters. And special parameters for this method is q. If q=.25 then denote first quartile,
If q=.75 then denote third quartie,
By default it denote second quartile that is median value.

Syntax: df.quantile()
Example 1: For a single column

In [61]:
```
import pandas as pd
result={'Name':['priya','priya','hridoy','hridoy','tanvir','tanvir','susmita','susmita'],
        'ut':[1,2,1,2,1,2,1,2],
        'Math':[90,91,92,91,94,92,90,91],
        'Stat':[80,81,82,81,83,82,81,80],
        'DM':[90,91,92,93,91,92,93,91]}
df=pd.DataFrame(result)
df1=df['Math']
print("\nThe Result of math:")
print(df1)
print('\n1st quartile value:',df1.quantile(q=.25))
print('2nd quartile value or median:',df1.quantile())
print('3rd quartile value:',df1.quantile(q=.75))
```

```
The Result of math:
0    90
1    91
2    92
3    91
4    94
5    92
6    90
7    91
Name: Math, dtype: int64

1st quartile value: 90.75
2nd quartile value or median: 91.0
3rd quartile value: 92.0
```

Example 2: For multiple column

In [66]:
```
import pandas as pd
result={'Name':['priya','priya','hridoy','hridoy','tanvir','tanvir','susmita','susmita'],
        'ut':[1,2,1,2,1,2,1,2],
        'Math':[90,91,92,91,94,92,90,91],
        'Stat':[80,81,82,81,83,82,81,80],
        'DM':[90,91,92,93,91,92,93,91]}
df=pd.DataFrame(result)
df1=df[['Math','Stat','DM']]
print("\nThe Result of all subject:")
print(df1)
print('\n1st quartile value:')
print(df1.quantile([.25,.75]))
```

```
The Result of all subject:
   Math  Stat  DM
0    90    80  90
1    91    81  91
2    92    82  92
3    91    81  93
4    94    83  91
5    92    82  92
6    90    81  93
7    91    80  91

1st quartile value:
       Math   Stat     DM
0.25  90.75  80.75  91.00
0.75  92.00  82.00  92.25
```

➢ Calculate variance:

It is the average of squared differences from the mean. If we want to calculate the variance of each column or row then use var method. We can use parameters.

Syntax: df.var()

Example :

```
In [67]: import pandas as pd
result={'Name':['priya','priya','hridoy','hridoy','tanvir','tanvir','susmita','susmita'],
        'ut':[1,2,1,2,1,2,1,2],
        'Math':[90,91,92,91,94,92,90,91],
        'Stat':[80,81,82,81,83,82,81,80],
        'DM':[90,91,92,93,91,92,93,91]}
df=pd.DataFrame(result)
df1=df['Math']
print("\nThe Result of math:")
print(df1)
print('\nvariance:',df1.var())
```

```
The Result of math:
0    90
1    91
2    92
3    91
4    94
5    92
6    90
7    91
Name: Math, dtype: int64

variance: 1.6964285714285714
```

➢ Calculate standard deviation:

It is the square root of the variance. If we want to calculate the standard deviation of each column or row then use std method. We can use parameters.

Syntax: df.std()

Example :

In [68]:
```
import pandas as pd
result={'Name':['priya','priya','hridoy','hridoy','tanvir','tanvir','susmita','susmita'],
        'ut':[1,2,1,2,1,2,1,2],
        'Math':[90,91,92,91,94,92,90,91],
        'Stat':[80,81,82,81,83,82,81,80],
        'DM':[90,91,92,93,91,92,93,91]}
df=pd.DataFrame(result)
df1=df['Math']
print("\nThe Result of math:")
print(df1)
print('\nstandard deviation:',df1.std())
```

```
The Result of math:
0    90
1    91
2    92
3    91
4    94
5    92
6    90
7    91
Name: Math, dtype: int64

standard deviation: 1.3024701806293193
```

➢ Describe() method:

This method display the descriptive statistical values in a single command.

Syntax: df.describe()

Example:

```
In [73]: import pandas as pd
result={'Name':['priya','priya','hridoy','hridoy','tanvir','tanvir','susmita','susmita'],
        'ut':[1,2,1,2,1,2,1,2],
        'Math':[90,91,92,91,94,92,90,91],
        'Stat':[80,81,82,81,83,82,81,80],
        'DM':[90,91,92,93,91,92,93,91]}
df=pd.DataFrame(result)
print("\nThe Result table:\n",df)
print('\nstatistical values:\n',df.describe())
```

```
The Result table:
       Name  ut  Math  Stat  DM
0     priya   1    90    80  90
1     priya   2    91    81  91
2    hridoy   1    92    82  92
3    hridoy   2    91    81  93
4    tanvir   1    94    83  91
5    tanvir   2    92    82  92
6   susmita   1    90    81  93
7   susmita   2    91    80  91

statistical values:
              ut       Math       Stat        DM
count  8.000000   8.00000   8.000000   8.00000
mean   1.500000  91.37500  81.250000  91.62500
std    0.534522   1.30247   1.035098   1.06066
min    1.000000  90.00000  80.000000  90.00000
25%    1.000000  90.75000  80.750000  91.00000
50%    1.500000  91.00000  81.000000  91.50000
75%    2.000000  92.00000  82.000000  92.25000
max    2.000000  94.00000  83.000000  93.00000
```

❖ Data Aggregations:

Aggregation means to transform the dataset and produce a single numeric value. Can be applied to one or more columns together. We can use one or more statistical method(max,min,sum,count,std,var,mean,mode,median) together.

Syntax: df.aggregation(‘function name’)

Example 1: Single function using aggregation

In [74]:
```
import pandas as pd
result={'Name':['priya','priya','hridoy','hridoy','tanvir','tanvir','susmita','susmita'],
        'ut':[1,2,1,2,1,2,1,2],
        'Math':[90,91,92,91,94,92,90,91],
        'stat':[80,81,82,81,83,82,81,80],
        'DM':[90,91,92,93,91,92,93,91]}
df=pd.DataFrame(result)
print("The Result table is:")
print(df)
print("\nMaximum values are:\n",df.aggregate('max'))
```

```
The Result table is:
      Name  ut  Math  stat  DM
0    priya   1    90    80  90
1    priya   2    91    81  91
2   hridoy   1    92    82  92
3   hridoy   2    91    81  93
4   tanvir   1    94    83  91
5   tanvir   2    92    82  92
6  susmita   1    90    81  93
7  susmita   2    91    80  91

Maximum values are:
 Name    tanvir
ut           2
Math        94
stat        83
DM          93
dtype: object
```

Example 2: Multiple aggregation function in a single statement

In [80]:
```
import pandas as pd
result={'Name':['priya','priya','hridoy','hridoy','tanvir','tanvir','susmita','susmita'],
        'ut':[1,2,1,2,1,2,1,2],
        'Math':[90,91,92,91,94,92,90,91],
        'stat':[80,81,82,81,83,82,81,80],
        'DM':[90,91,92,93,91,92,93,91]}
df=pd.DataFrame(result)
print("The Result table is:")
print(df)
print("\nvalues are:\n",df.aggregate(['max','min','count']))
```

```
The Result table is:
      Name  ut  Math  stat  DM
0    priya   1    90    80  90
1    priya   2    91    81  91
2   hridoy   1    92    82  92
3   hridoy   2    91    81  93
4   tanvir   1    94    83  91
5   tanvir   2    92    82  92
6  susmita   1    90    81  93
7  susmita   2    91    80  91

values are:
          Name  ut  Math  stat  DM
max    tanvir   2    94    83  93
min    hridoy   1    90    80  90
count       8   8     8     8   8
```

Example 3: Multiple aggregation function in a single statement with axis parameter.

```
In [88]: import pandas as pd
result={'Name':['priya','priya','hridoy','hridoy','tanvir','tanvir','susmita','susmita'],
        'ut':[1,2,1,2,1,2,1,2],
        'Math':[90,91,92,91,94,92,90,91],
        'stat':[80,81,82,81,83,82,81,80],
        'DM':[90,91,92,93,91,92,93,91]}
df=pd.DataFrame(result)
print("The Result table is:")
print(df)
print("\nvalues are:\n",df[['Math','DM']].aggregate(['max','min','count','sum'],axis=1))
```

```
The Result table is:
      Name  ut  Math  stat  DM
0    priya   1    90    80  90
1    priya   2    91    81  91
2   hridoy   1    92    82  92
3   hridoy   2    91    81  93
4   tanvir   1    94    83  91
5   tanvir   2    92    82  92
6  susmita   1    90    81  93
7  susmita   2    91    80  91

values are:
    max  min  count  sum
0   90   90      2  180
1   91   91      2  182
2   92   92      2  184
3   93   91      2  184
4   94   91      2  185
5   92   92      2  184
6   93   90      2  183
7   91   91      2  182
```

❖ Sorting a dataframe:

Sorting refers to the arrangement of data elements in a specified order,which can either be ascending and descending. For sorting dataframe we can use sort_value method.

Syntax: df.sort_value(by=[‘label’],axis=0,ascending=True) (by default)

Example 1: sort by single attribute/column

```
In [90]: import pandas as pd
result={'Name':['priya','priya','hridoy','hridoy','tanvir','tanvir','susmita','susmita'],
        'ut':[1,2,1,2,1,2,1,2],
        'Math':[90,91,92,91,94,92,90,91],
        'stat':[80,81,82,81,83,82,81,80],
        'DM':[90,91,92,93,91,92,93,91]}
df=pd.DataFrame(result)
print("The Result table is:")
print(df)
print("\nvalues are:\n",df.sort_values(by=['Name'],ascending=True))
```

```
The Result table is:
      Name  ut  Math  stat  DM
0    priya   1    90    80  90
1    priya   2    91    81  91
2   hridoy   1    92    82  92
3   hridoy   2    91    81  93
4   tanvir   1    94    83  91
5   tanvir   2    92    82  92
6  susmita   1    90    81  93
7  susmita   2    91    80  91

values are:
       Name  ut  Math  stat  DM
2   hridoy   1    92    82  92
3   hridoy   2    91    81  93
0    priya   1    90    80  90
1    priya   2    91    81  91
6  susmita   1    90    81  93
7  susmita   2    91    80  91
4   tanvir   1    94    83  91
5   tanvir   2    92    82  92
```

Example 2: sort by multiple attributes/columns

```
In [91]: import pandas as pd
result={'Name':['priya','priya','hridoy','hridoy','tanvir','tanvir','susmita','susmita'],
        'ut':[1,2,1,2,1,2,1,2],
        'Math':[90,91,92,91,94,92,90,91],
        'stat':[80,81,82,81,83,82,81,80],
        'DM':[90,91,92,93,91,92,93,91]}
df=pd.DataFrame(result)
print("The Result table is:")
print(df)
print("\nvalues are:\n",df.sort_values(by=['Name','Math'],ascending=True))
```

```
The Result table is:
      Name  ut  Math  stat  DM
0    priya   1    90    80  90
1    priya   2    91    81  91
2   hridoy   1    92    82  92
3   hridoy   2    91    81  93
4   tanvir   1    94    83  91
5   tanvir   2    92    82  92
6  susmita   1    90    81  93
7  susmita   2    91    80  91

values are:
       Name  ut  Math  stat  DM
3   hridoy   2    91    81  93
2   hridoy   1    92    82  92
0    priya   1    90    80  90
1    priya   2    91    81  91
6  susmita   1    90    81  93
7  susmita   2    91    80  91
5   tanvir   2    92    82  92
4   tanvir   1    94    83  91
```

❖ Group by function:

Groupby function is used to split the data into groups based on some criteria. This function works based on a split-apply-combine strategy which is shown below using a 3-step process:

Step 1: Split the data into groups by creating a groupby object from the original DataFrame.

Step 2: Apply the required function(size,sum,mean,get_group…).

Step 3: Combine the results to form a new DataFrame.

Syntax: g1=df.groupby(‘column name’)

Df1=g1.size()

Example 1: display the first entry from each group

In [96]:
```
import pandas as pd
result={'Name':['priya','priya','hridoy','hridoy','tanvir','tanvir','susmita','susmita'],
        'ut':[1,2,1,2,1,2,1,2],
        'Math':[90,91,92,91,94,92,90,91],
        'stat':[80,81,82,81,83,82,81,80],
        'DM':[90,91,92,93,91,92,93,91]}
df=pd.DataFrame(result)
print("The Result table is:")
print(df)
g1=df.groupby('Name')
print(g1.first())
```

```
The Result table is:
      Name  ut  Math  stat  DM
0    priya   1    90    80  90
1    priya   2    91    81  91
2   hridoy   1    92    82  92
3   hridoy   2    91    81  93
4   tanvir   1    94    83  91
5   tanvir   2    92    82  92
6  susmita   1    90    81  93
7  susmita   2    91    80  91
         ut  Math  stat  DM
Name
hridoy    1    92    82  92
priya     1    90    80  90
susmita   1    90    81  93
tanvir    1    94    83  91
```

Example 2: display the size of each group

In [103]:
```
import pandas as pd
result={'Name':['priya','priya','hridoy','hridoy','tanvir','tanvir','susmita','susmita'],
        'ut':[1,2,1,2,1,2,1,2],
        'Math':[90,91,92,91,94,92,90,91],
        'stat':[80,81,82,81,83,82,81,80],
        'DM':[90,91,92,93,91,92,93,91]}
df=pd.DataFrame(result)
g1=df.groupby('Name')
print(g1.size())
```

```
Name
hridoy     2
priya      2
susmita    2
tanvir     2
dtype: int64
```

Example 3: display data of a single group

In [98]:
```
import pandas as pd
result={'Name':['priya','priya','hridoy','hridoy','tanvir','tanvir','susmita','susmita'],
        'ut':[1,2,1,2,1,2,1,2],
        'Math':[90,91,92,91,94,92,90,91],
        'stat':[80,81,82,81,83,82,81,80],
        'DM':[90,91,92,93,91,92,93,91]}
df=pd.DataFrame(result)
g1=df.groupby('Name')
print(g1.get_group('susmita'))
```

```
      Name  ut  Math  stat  DM
6  susmita   1    90    81  93
7  susmita   2    91    80  91
```

Example 4: display all groups data

In [102]:
```
import pandas as pd
result={'Name':['priya','priya','hridoy','hridoy','tanvir','tanvir','susmita','susmita'],
        'ut':[1,2,1,2,1,2,1,2],
        'Math':[90,91,92,91,94,92,90,91],
        'stat':[80,81,82,81,83,82,81,80],
        'DM':[90,91,92,93,91,92,93,91]}
df=pd.DataFrame(result)
g1=df.groupby('Name')
print(g1.groups)
```

```
{'hridoy': [2, 3], 'priya': [0, 1], 'susmita': [6, 7], 'tanvir': [4, 5]}
```

Example 5: grouping with multiple attributes

In [105]:
```
import pandas as pd
result={'Name':['priya','priya','hridoy','hridoy','tanvir','tanvir','susmita','susmita'],
        'ut':[1,2,1,2,1,2,1,2],
        'Math':[90,91,92,91,94,92,90,91],
        'stat':[80,81,82,81,83,82,81,80],
        'DM':[90,91,92,93,91,92,93,91]}
df=pd.DataFrame(result)
g1=df.groupby(['Name','ut'])
print(g1.first())
```

```
             Math  stat  DM
Name    ut
hridoy  1     92    82  92
        2     91    81  93
priya   1     90    80  90
        2     91    81  91
susmita 1     90    81  93
        2     91    80  91
tanvir  1     94    83  91
        2     92    82  92
```

Example 6: calculate average of each group

In [106]:
```
import pandas as pd
result={'Name':['priya','priya','hridoy','hridoy','tanvir','tanvir','susmita','susmita'],
        'ut':[1,2,1,2,1,2,1,2],
        'Math':[90,91,92,91,94,92,90,91],
        'stat':[80,81,82,81,83,82,81,80],
        'DM':[90,91,92,93,91,92,93,91]}
df=pd.DataFrame(result)
g1=df.groupby(['ut'])
print(g1.aggregate('mean'))
```

```
     Math  stat     DM
ut
1   91.50  81.5  91.50
2   91.25  81.0  91.75
```

Example 7: calculate average of each group with single attribute

In [108]:
```
import pandas as pd
result={'Name':['priya','priya','hridoy','hridoy','tanvir','tanvir','susmita','susmita'],
        'ut':[1,2,1,2,1,2,1,2],
        'Math':[90,91,92,91,94,92,90,91],
        'stat':[80,81,82,81,83,82,81,80],
        'DM':[90,91,92,93,91,92,93,91]}
df=pd.DataFrame(result)
g1=df.groupby(['ut'])
print(g1['Math'].aggregate('mean'))
```

```
ut
1    91.50
2    91.25
Name: Math, dtype: float64
```

Example 8: calculate statistical data of each group with single attribute and multiple aggregate functions

In [110]:
```
import pandas as pd
result={'Name':['priya','priya','hridoy','hridoy','tanvir','tanvir','susmita','susmita'],
        'ut':[1,2,1,2,1,2,1,2],
        'Math':[90,91,92,91,94,92,90,91],
        'stat':[80,81,82,81,83,82,81,80],
        'DM':[90,91,92,93,91,92,93,91]}
df=pd.DataFrame(result)
g1=df.groupby(['ut'])
print(g1['Math'].aggregate(['mean','max','min','var','std']))
```

```
     mean  max  min       var       std
ut
1   91.50   94   90  3.666667  1.914854
2   91.25   92   91  0.250000  0.500000
```

❖ Altering the index:

Depending on our requirements, we can select some other column to be the index or we can add another index column (specially in slicng).

Syntax: df.reset_index(inplace=True)

Example 1: In slicing, altering the index

```
In [112]: import pandas as pd
result={'Name':['priya','priya','hridoy','hridoy','tanvir','tanvir','susmita','susmita'],
        'ut':[1,2,1,2,1,2,1,2],
        'Math':[90,91,92,91,94,92,90,91],
        'stat':[80,81,82,81,83,82,81,80],
        'DM':[90,91,92,93,91,92,93,91]}
df=pd.DataFrame(result)
df2=df[df.ut==2]
print("\nThe Result of unit test 2:")
print(df2,"\n")
df2.reset_index(inplace=True)
print(df2)
```

```
The Result of unit test 2:
      Name  ut  Math  stat  DM
1    priya   2    91    81  91
3   hridoy   2    91    81  93
5   tanvir   2    92    82  92
7  susmita   2    91    80  91

   index     Name  ut  Math  stat  DM
0      1    priya   2    91    81  91
1      3   hridoy   2    91    81  93
2      5   tanvir   2    92    82  92
3      7  susmita   2    91    80  91
```

Example 2: In slicing, drop the original index after creating new index

In [133]:
```
import pandas as pd
result={'Name':['priya','priya','hridoy','hridoy','tanvir','tanvir','susmita','susmita'],
        'ut':[1,2,1,2,1,2,1,2],
        'Math':[90,91,92,91,94,92,90,91],
        'stat':[80,81,82,81,83,82,81,80],
        'DM':[90,91,92,93,91,92,93,91]}
df=pd.DataFrame(result)
df2=df[df.ut==2]
df2.reset_index(inplace=True)
print(df2)
print(df2.drop(columns=['index']))
```

```
   index     Name  ut  Math  stat  DM
0      1    priya   2    91    81  91
1      3   hridoy   2    91    81  93
2      5   tanvir   2    92    82  92
3      7  susmita   2    91    80  91
      Name  ut  Math  stat  DM
0    priya   2    91    81  91
1   hridoy   2    91    81  93
2   tanvir   2    92    82  92
3  susmita   2    91    80  91
```

Example 3: Select another column as index and then reset the index

Set -

In [120]:
```
import pandas as pd
result={'Name':['priya','priya','hridoy','hridoy','tanvir','tanvir','susmita','susmita'],
        'ut':[1,2,1,2,1,2,1,2],
        'Math':[90,91,92,91,94,92,90,91],
        'stat':[80,81,82,81,83,82,81,80],
        'DM':[90,91,92,93,91,92,93,91]}
df=pd.DataFrame(result)
df2=df[df.ut==2]
print("\nThe Result of unit test 2:")
print(df2,"\n")
df2.set_index('Name',inplace=True)
print(df2)
```

```
The Result of unit test 2:
      Name  ut  Math  stat  DM
1    priya   2    91    81  91
3   hridoy   2    91    81  93
5   tanvir   2    92    82  92
7  susmita   2    91    80  91

         ut  Math  stat  DM
Name
priya     2    91    81  91
hridoy    2    91    81  93
tanvir    2    92    82  92
susmita   2    91    80  91
```

Reset-

```
In [126]: import pandas as pd
result={'Name':['priya','priya','hridoy','hridoy','tanvir','tanvir','susmita','susmita'],
        'ut':[1,2,1,2,1,2,1,2],
        'Math':[90,91,92,91,94,92,90,91],
        'stat':[80,81,82,81,83,82,81,80],
        'DM':[90,91,92,93,91,92,93,91]}
df=pd.DataFrame(result)
df2=df[df.ut==2]
df2.set_index('Name',inplace=True)
print(df2)
df2.reset_index('Name',inplace=True)
print(df2)
```

```
         ut  Math  stat  DM
Name
priya     2    91    81  91
hridoy    2    91    81  93
tanvir    2    92    82  92
susmita   2    91    80  91
      Name  ut  Math  stat  DM
0    priya   2    91    81  91
1   hridoy   2    91    81  93
2   tanvir   2    92    82  92
3  susmita   2    91    80  91
```

❖ Reshaping data:

The way a dataset is arranged into rows and columns is referred to as the shape of data. Reshaping data refers to the process of changing the shape of the dataset to make it suitable for some analysis problems.

For reshaping data, two basic functions are available in Pandas,

i. pivot and
ii. pivot_table.

➢ Pivot:

The pivot function is used to reshape and create a new DataFrame from the original one. In previous section, we have to slice the data corresponding to a particular attribute and then apply the statistical method for finding descriptive statistical data. But reshaping has transformed the structure of the data, which makes it more readable and easy to analyze the data.

• Pivoting by single column:

Syntax: pivot1=df.pivot(index='attribute',columns='attribute',values='attribute')

pivot1.loc['index_value'].sum()

Example :

```
In [8]: import pandas as pd
data={'Store':['S1','S4','S3','S1','S2','S3','S1','S2','S3'],
      'Year':[2016,2016,2016,2017,2017,2017,2018,2018,2018],
      'Total_sales(Rs)':[12000,330000,420000,20000,10000,450000,30000, 11000,89000],
      'Total_profit(Rs)':[1100,5500,21000,32000,9000,45000,3000,1900,23000]
}
df=pd.DataFrame(data)
print(df,'\n')
pivot1=df.pivot(index='Store',columns='Year',values='Total_sales(Rs)')
print(pivot1)
print("\nThe total sale of store s1 in all year is :",pivot1.loc['S1'].sum())
```

```
  Store  Year  Total_sales(Rs)  Total_profit(Rs)
0    S1  2016            12000              1100
1    S4  2016           330000              5500
2    S3  2016           420000             21000
3    S1  2017            20000             32000
4    S2  2017            10000              9000
5    S3  2017           450000             45000
6    S1  2018            30000              3000
7    S2  2018            11000              1900
8    S3  2018            89000             23000

Year       2016      2017     2018
Store
S1      12000.0   20000.0  30000.0
S2          NaN   10000.0  11000.0
S3     420000.0  450000.0  89000.0
S4     330000.0       NaN      NaN

The total sale of store s1 in all year is : 62000.0
```

- ## Pivoting by multiple columns:

Syntax:

pivot1=df.pivot(index='attribute',columns='attribute',values=[’attribute1','attribute’ ,….])

pivot1.loc[‘index_value’].sum()

Example :

```
In [10]: import pandas as pd
data={'Store':['S1','S4','S3','S1','S2','S3','S1','S2','S3'],
      'Year':[2016,2016,2016,2017,2017,2017,2018,2018,2018],
      'Total_sales(Rs)':[12000,330000,420000,20000,10000,450000,30000, 11000,89000],
      'Total_profit(Rs)':[1100,5500,21000,32000,9000,45000,3000,1900,23000]
}
df=pd.DataFrame(data)
print(df,'\n')
pivot2=df.pivot(index='Store',columns='Year',values=['Total_sales(Rs)','Total_profit(Rs)'])
print(pivot2)
```

```
  Store  Year  Total_sales(Rs)  Total_profit(Rs)
0    S1  2016            12000              1100
1    S4  2016           330000              5500
2    S3  2016           420000             21000
3    S1  2017            20000             32000
4    S2  2017            10000              9000
5    S3  2017           450000             45000
6    S1  2018            30000              3000
7    S2  2018            11000              1900
8    S3  2018            89000             23000

      Total_sales(Rs)                   Total_profit(Rs)
Year             2016      2017     2018             2016     2017     2018
Store
S1            12000.0   20000.0  30000.0           1100.0  32000.0   3000.0
S2                NaN   10000.0  11000.0              NaN   9000.0   1900.0
S3           420000.0  450000.0  89000.0          21000.0  45000.0  23000.0
S4           330000.0       NaN      NaN           5500.0      NaN      NaN
```

➢ Pivot table:

Duplicate data can't be reshaped using pivot function. That's why we may have to use pivot_table function instead. It works like a pivot function, but aggregates the values from rows with duplicate entries for the specified columns.

The default aggregate function is mean.

Syntax:

pd.pivot_table(data,values=None,index=None,columns=None,aggfunc='mean')

The parameter aggfunc can have values among sum,max, min, len, np.mean, np.median wherever we have duplicate entries.

For calculating mean,median we have to import numpy as np.

Example:

```
In [17]: import pandas as pd
         import numpy as np
         data={'Item':['Pen','Pen','Pencil','Pencil','Pen','Pen'],
               'Color':['Red','Red','Black','Black','Blue','Blue'],
               'Price(Rs)':[10,25,7,5,50,20],
               'Units_in_stock':[50,10,47,34,55,14]
         }
         df=pd.DataFrame(data)
         print(df)
         pivot3=df.pivot_table(index='Item',columns='Color',values='Units_in_stock',aggfunc=[sum,max,np.mean,len])
         print(pivot3)
```

```
     Item  Color  Price(Rs)  Units_in_stock
0     Pen    Red         10              50
1     Pen    Red         25              10
2  Pencil  Black          7              47
3  Pencil  Black          5              34
4     Pen   Blue         50              55
5     Pen   Blue         20              14
         sum              max             mean              len
Color  Black  Blue   Red Black  Blue   Red Black  Blue   Red Black Blue  Red
Item
Pen      NaN  69.0  60.0   NaN  55.0  50.0   NaN  34.5  30.0   NaN  2.0  2.0
Pencil  81.0   NaN   NaN  47.0   NaN   NaN  40.5   NaN   NaN   2.0  NaN  NaN
```

❖ Handling missing value:

As we know that a DataFrame can consist of many rows (objects) where each row can have values for various columns (attributes). If a value corresponding to a column is not present, it is considered to be a missing value. A missing value is denoted by NaN. Missing values create a lot of problems during data analysis and have to be handled properly. The two most common strategies for handling missing values explained in this section are:

i. drop the object having missing values,
ii. fill or estimate the missing value

➢ Checking missing values:

For checking missing values there are some method. They are-

- **Isnull() method:**

Pandas provide a function isnull() to check whether any value is missing or not in the DataFrame. This function checks all attributes and returns True in case that attribute has missing values, otherwise returns False.

We can check for each individual attribute also.

Syntax: df.isnull()

Example:

```
In [18]: import pandas as pd
         import numpy as np
         result={'Name':['priya','priya','hridoy','hridoy','tanvir','tanvir','susmita','susmita'],
                 'ut':[1,2,1,2,1,2,1,2],
                 'Math':[90,91,np.NaN,91,94,92,90,91],
                 'Stat':[80,81,82,81,83,82,81,80],
                 'DM':[90,91,92,93,91,np.NaN,93,91]}
         df=pd.DataFrame(result)
         print("The Result table is:")
         print(df)
         print(df.isnull())
```

```
The Result table is:
      Name  ut  Math  Stat    DM
0    priya   1  90.0    80  90.0
1    priya   2  91.0    81  91.0
2   hridoy   1   NaN    82  92.0
3   hridoy   2  91.0    81  93.0
4   tanvir   1  94.0    83  91.0
5   tanvir   2  92.0    82   NaN
6  susmita   1  90.0    81  93.0
7  susmita   2  91.0    80  91.0
    Name     ut   Math   Stat     DM
0  False  False  False  False  False
1  False  False  False  False  False
2  False  False   True  False  False
3  False  False  False  False  False
4  False  False  False  False  False
5  False  False  False  False   True
6  False  False  False  False  False
7  False  False  False  False  False
```

- **Isnull().any() method:**

To check whether a column (attribute) has a missing value in the entire dataset, any() function is used. It returns True in case of missing value else returns False.

We can check for each individual attribute also.

Syntax: df.isnull().any()

Example:

In [19]:
```
import pandas as pd
import numpy as np
result={'Name':['priya','priya','hridoy','hridoy','tanvir','tanvir','susmita','susmita'],
        'ut':[1,2,1,2,1,2,1,2],
        'Math':[90,91,np.NaN,91,94,92,90,91],
        'Stat':[80,81,82,81,83,82,81,80],
        'DM':[90,91,92,93,91,np.NaN,93,91]}
df=pd.DataFrame(result)
print("The Result table is:")
print(df)
print(df.isnull().any())
```

```
The Result table is:
      Name  ut  Math  Stat    DM
0    priya   1  90.0    80  90.0
1    priya   2  91.0    81  91.0
2   hridoy   1   NaN    82  92.0
3   hridoy   2  91.0    81  93.0
4   tanvir   1  94.0    83  91.0
5   tanvir   2  92.0    82   NaN
6  susmita   1  90.0    81  93.0
7  susmita   2  91.0    80  91.0
Name    False
ut      False
Math     True
Stat    False
DM       True
dtype: bool
```

- **Isnull().sum() method:**

To find the number of NaN values corresponding to each attribute, one can use the sum() function along with isnull() function.

Syntax: df.isnull().sum()

Example:

```
In [20]: import pandas as pd
         import numpy as np
         result={'Name':['priya','priya','hridoy','hridoy','tanvir','tanvir','susmita','susmita'],
                 'ut':[1,2,1,2,1,2,1,2],
                 'Math':[90,91,np.NaN,91,94,92,90,91],
                 'Stat':[80,81,82,81,83,82,81,80],
                 'DM':[90,91,92,93,91,np.NaN,93,91]}
         df=pd.DataFrame(result)
         print("The Result table is:")
         print(df)
         print(df.isnull().sum())
```

```
The Result table is:
      Name  ut  Math  Stat    DM
0    priya   1  90.0    80  90.0
1    priya   2  91.0    81  91.0
2   hridoy   1   NaN    82  92.0
3   hridoy   2  91.0    81  93.0
4   tanvir   1  94.0    83  91.0
5   tanvir   2  92.0    82   NaN
6  susmita   1  90.0    81  93.0
7  susmita   2  91.0    80  91.0
Name    0
ut      0
Math    1
Stat    0
DM      1
dtype: int64
```

- ## Isnull().sum().sum() method:

To find the total number of NaN in the whole dataset, one can use this method.

Syntax: df.isnull().sum().sum().

Example:

```
In [22]: import pandas as pd
         import numpy as np
         result={'Name':['priya','priya','hridoy','hridoy','tanvir','tanvir','susmita','susmita'],
                 'ut':[1,2,1,2,1,2,1,2],
                 'Math':[90,91,np.NaN,91,94,92,90,91],
                 'Stat':[80,81,82,81,83,82,81,80],
                 'DM':[90,91,92,93,91,np.NaN,93,91]}
         df=pd.DataFrame(result)
         print("The Result table is:")
         print(df)
         print("Total null value:", df.isnull().sum().sum())
```

```
The Result table is:
      Name  ut  Math  Stat    DM
0    priya   1  90.0    80  90.0
1    priya   2  91.0    81  91.0
2   hridoy   1   NaN    82  92.0
3   hridoy   2  91.0    81  93.0
4   tanvir   1  94.0    83  91.0
5   tanvir   2  92.0    82   NaN
6  susmita   1  90.0    81  93.0
7  susmita   2  91.0    80  91.0
Total null value: 2
```

➢ Dropping missing values:

Missing values can be handled by either dropping the entire row having missing value or replacing it with appropriate value. Dropping will remove the entire row (object) having the missing value(s). The dropna() function can be used to drop an entire row from the DataFrame.

Syntax: df.dropna()

Example:

```
In [26]: import pandas as pd
         import numpy as np
         result={'Name':['priya','priya','hridoy','hridoy','tanvir','tanvir','susmita','susmita'],
                 'ut':[1,2,1,2,1,2,1,2],
                 'Math':[90,91,np.NaN,91,94,92,90,91],
                 'Stat':[80,81,82,81,83,82,81,80],
                 'DM':[90,91,92,93,91,np.NaN,93,91]}
         df=pd.DataFrame(result)
         print("The Result table is:")
         print(df)
         df1 = df.dropna()
         print("New dataframe:\n",df1)
```

```
The Result table is:
      Name  ut  Math  Stat    DM
0    priya   1  90.0    80  90.0
1    priya   2  91.0    81  91.0
2   hridoy   1   NaN    82  92.0
3   hridoy   2  91.0    81  93.0
4   tanvir   1  94.0    83  91.0
5   tanvir   2  92.0    82   NaN
6  susmita   1  90.0    81  93.0
7  susmita   2  91.0    80  91.0
New dataframe:
       Name  ut  Math  Stat    DM
0    priya   1  90.0    80  90.0
1    priya   2  91.0    81  91.0
3   hridoy   2  91.0    81  93.0
4   tanvir   1  94.0    83  91.0
6  susmita   1  90.0    81  93.0
7  susmita   2  91.0    80  91.0
```

➢ Estmaing missing values:

Missing values can be filled by using estimations or approximations e.g a value just before or after the missing value. In some cases, missing values are replaced by zeros or ones.

- **Fillna(num) method:**

The fillna(num) function can be used to replace missing values by the value specified in num.

i. fillna(0) replaces missing value by 0.
ii. fillna(1) replaces missing value by 1.

Syntax: df. fillna(num)

Example:

```
In [29]: import pandas as pd
         import numpy as np
         result={'Name':['priya','priya','hridoy','hridoy','tanvir','tanvir','susmita','susmita'],
                 'ut':[1,2,1,2,1,2,1,2],
                 'Math':[90,91,np.NaN,91,94,92,90,91],
                 'Stat':[80,81,82,81,83,82,81,80],
                 'DM':[90,91,92,93,91,np.NaN,93,91]}
         df=pd.DataFrame(result)
         print("The Result table is:")
         print(df)
         df1 = df.fillna(0)
         print("New dataframe:\n",df1)
```

```
The Result table is:
      Name  ut  Math  Stat    DM
0    priya   1  90.0    80  90.0
1    priya   2  91.0    81  91.0
2   hridoy   1   NaN    82  92.0
3   hridoy   2  91.0    81  93.0
4   tanvir   1  94.0    83  91.0
5   tanvir   2  92.0    82   NaN
6  susmita   1  90.0    81  93.0
7  susmita   2  91.0    80  91.0
New dataframe:
       Name  ut  Math  Stat    DM
0    priya   1  90.0    80  90.0
1    priya   2  91.0    81  91.0
2   hridoy   1   0.0    82  92.0
3   hridoy   2  91.0    81  93.0
4   tanvir   1  94.0    83  91.0
5   tanvir   2  92.0    82   0.0
6  susmita   1  90.0    81  93.0
7  susmita   2  91.0    80  91.0
```

- **fillna(method='pad') method:**

This method replaces the missing value by the value before the missing value.

Syntax: df.fillna(method='pad')

Example:

```
In [30]: import pandas as pd
         import numpy as np
         result={'Name':['priya','priya','hridoy','hridoy','tanvir','tanvir','susmita','susmita'],
                 'ut':[1,2,1,2,1,2,1,2],
                 'Math':[90,91,np.NaN,91,94,92,90,91],
                 'Stat':[80,81,82,81,83,82,81,80],
                 'DM':[90,91,92,93,91,np.NaN,93,91]}
         df=pd.DataFrame(result)
         print("The Result table is:")
         print(df)
         df1 = df.fillna(method='pad')
         print("New dataframe:\n",df1)
```

```
The Result table is:
      Name  ut  Math  Stat    DM
0    priya   1  90.0    80  90.0
1    priya   2  91.0    81  91.0
2   hridoy   1   NaN    82  92.0
3   hridoy   2  91.0    81  93.0
4   tanvir   1  94.0    83  91.0
5   tanvir   2  92.0    82   NaN
6  susmita   1  90.0    81  93.0
7  susmita   2  91.0    80  91.0
New dataframe:
       Name  ut  Math  Stat    DM
0    priya   1  90.0    80  90.0
1    priya   2  91.0    81  91.0
2   hridoy   1  91.0    82  92.0
3   hridoy   2  91.0    81  93.0
4   tanvir   1  94.0    83  91.0
5   tanvir   2  92.0    82  91.0
6  susmita   1  90.0    81  93.0
7  susmita   2  91.0    80  91.0
```

- **fillna(method=’bfill’) method:**

This method replaces the missing value by the value after the missing value.

Syntax: df.fillna(method='bfill')

Example:

In [31]:
```
import pandas as pd
import numpy as np
result={'Name':['priya','priya','hridoy','hridoy','tanvir','tanvir','susmita','susmita'],
        'ut':[1,2,1,2,1,2,1,2],
        'Math':[90,91,np.NaN,91,94,92,90,91],
        'Stat':[80,81,82,81,83,82,81,80],
        'DM':[90,91,92,93,91,np.NaN,93,91]}
df=pd.DataFrame(result)
print("The Result table is:\n",df)
df1 = df.fillna(method='bfill')
print("New dataframe:\n",df1)
```

```
The Result table is:
       Name  ut  Math  Stat    DM
0    priya   1  90.0    80  90.0
1    priya   2  91.0    81  91.0
2   hridoy   1   NaN    82  92.0
3   hridoy   2  91.0    81  93.0
4   tanvir   1  94.0    83  91.0
5   tanvir   2  92.0    82   NaN
6  susmita   1  90.0    81  93.0
7  susmita   2  91.0    80  91.0
New dataframe:
       Name  ut  Math  Stat    DM
0    priya   1  90.0    80  90.0
1    priya   2  91.0    81  91.0
2   hridoy   1  91.0    82  92.0
3   hridoy   2  91.0    81  93.0
4   tanvir   1  94.0    83  91.0
5   tanvir   2  92.0    82  93.0
6  susmita   1  90.0    81  93.0
7  susmita   2  91.0    80  91.0
```

END