NumPy-II & Pandas - I

🎯 Session Objectives:

- ✅ Learn statistical and transformation operations on arrays.
- ✅ Understand what Pandas is and its importance
- ✅ Install and import the Pandas library
- ✅ Understand data structures in Pandas
- ✅ Understand what a Series is
- ✅ Differentiate Pandas Series vs NumPy Arrays
- ✅ Create Series from scalar, list, array, and dictionary
- ✅ Access Series elements using indexing and slicing
- ✅ Understand attributes of Series
- ✅ Learn basic mathematical operations on Series

Sorting An Array:

Sorting means arranging an elements in ascending (default) or descending order

```
import numpy as np
arr_1d = np.array([77,11,22,55,33,44,99,66,88])
print(arr_1d)
print(np.sort(arr_1d))
```

```
[77 11 22 55 33 44 99 66 88]
[11 22 33 44 55 66 77 88 99]
```

```
# Descending Order using slicing [::-1]
print(np.sort(arr_1d)[::-1])
```

```
[99 88 77 66 55 44 33 22 11]
```

```
# Axis = 0 [Horizontal Axis][Rows] & Axis = 1 [Vertical Axis][Cols]
arr_2d = np.array([
    [11,22,33,44,55],
    [99,88,77,66,55],
    [11,33,55,77,99],
    [22,44,66,88,10],
    [11,15,17,19,21]
])
np.sort(arr_2d , axis = 0)
```

```
array([[11, 15, 17, 19, 10],
       [11, 22, 33, 44, 21],
       [11, 33, 55, 66, 55],
       [22, 44, 66, 77, 55],
       [99, 88, 77, 88, 99]])
```

```
# axis = 1 [Vertical Axis] [Col]
arr_2d = np.array([
    [11,22,33,44,55],
    [99,88,77,66,55],
    [11,33,55,77,99],
    [22,44,66,88,10],
    [11,15,17,19,21]
])
np.sort(arr_2d , axis = 1)
```

```
array([[11, 22, 33, 44, 55],
       [55, 66, 77, 88, 99],
       [11, 33, 55, 77, 99],
       [10, 22, 44, 66, 88],
       [11, 15, 17, 19, 21]])
```

```
# argsort [Indexing Sort]
arr_1d = np.array([55,11,44,33,22])
pos_arr = np.argsort(arr_1d)
print(arr_1d)
print(pos_arr)
```

```
[55 11 44 33 22]
[1 4 3 2 0]
```

```
# Indexing
arr_1d[1] # 11
```

```
11
```

```
arr_1d[4] # 22
```

```
22
```

```
arr_1d[3] # 33
```

```
33
```

```
# Returns the indices that would sort an array. [argsort]
arr_1d[pos_arr]
```

```
array([11, 22, 33, 44, 55])
```

```
# argsort [Indexing Sort] [Descending]
arr_1d = np.array([55,11,44,33,22])
pos_arr = np.argsort(arr_1d)[::-1]
print(arr_1d)
print(pos_arr)
```

```
[55 11 44 33 22]
[0 2 3 4 1]
```

```
arr_1d[pos_arr] # Descending
```

```
array([55, 44, 33, 22, 11])
```

```
# argsort -> 2D Matrix
# axis = 1 [Vertical] [Col]
arr_2d = np.array([
    [11,22,33,44,55],
    [99,88,77,66,55],
    [11,33,55,77,99],
    [22,44,66,88,10],
    [11,15,17,19,21]
])
print(arr_2d)
print(np.argsort(arr_2d , axis = 1))
```

```
[[11 22 33 44 55]
 [99 88 77 66 55]
 [11 33 55 77 99]
 [22 44 66 88 10]
 [11 15 17 19 21]]
[[0 1 2 3 4]
 [4 3 2 1 0]
 [0 1 2 3 4]
 [4 0 1 2 3]
 [0 1 2 3 4]]
```

```
# argsort -> 2D Matrix
# axis = 0 [Horizontal] [Row]
arr_2d = np.array([
    [11,22,33,44,55],
    [99,88,77,66,55],
    [11,33,55,77,99],
    [22,44,66,88,10],
    [11,15,17,19,21]
])
print(arr_2d)
print(np.argsort(arr_2d , axis = 0))
```

```
[[11 22 33 44 55]
 [99 88 77 66 55]
 [11 33 55 77 99]
 [22 44 66 88 10]
 [11 15 17 19 21]]
[[0 4 4 4 3]
 [2 0 0 0 4]
 [4 2 2 1 0]
 [3 3 3 2 1]
 [1 1 1 3 2]]
```

```
# Concatenating an Array: It means joining the array either row wise or column wise
X = np.array([11,22,33,44,55])
Y = np.array([66,77,88,99])
Z = np.array([1,2,3,4,5,6,7,8,9])
print(np.concatenate((X,Y,Z)))
```

```
[11 22 33 44 55 66 77 88 99  1  2  3  4  5  6  7  8  9]
```

```
print(np.concatenate((Y,X,Z)))
```

```
[66 77 88 99 11 22 33 44 55  1  2  3  4  5  6  7  8  9]
```

```
print(np.concatenate((Z,X,Y)))
```

```
[ 1  2  3  4  5  6  7  8  9 11 22 33 44 55 66 77 88 99]
```

```
# Concatenating on 2D Matrix
arr_2dA = np.array([
    [11,22,33],
    [44,55,66],
    [77,88,99],
]) # shape(3,3)

arr_2dB = np.array([
    [11,22],
    [44,55],
    [77,88],
]) # shape(3,2)

arr_2dC = np.array([
    [11,22,33],
    [44,55,66],
]) # shape(2,3)
```

```
print(np.concatenate((arr_2dA , arr_2dB) , axis = 1)) # Vertically Append [Col-wise]
```

```
[[11 22 33 11 22]
 [44 55 66 44 55]
 [77 88 99 77 88]]
```

```
print(np.concatenate((arr_2dA , arr_2dC) , axis = 1)) # Vertically Append [Col-wise]
ValueError: all the input array dimensions except for the concatenation axis must match exactly,
but along dimension 0, the array at index 0 has size 3 and the array at index 1 has size 2
```

```
print(np.concatenate((arr_2dA , arr_2dC) , axis = 0)) # Horizontal Append [Row-wise]
```

```
[[11 22 33]
 [44 55 66]
 [77 88 99]
 [11 22 33]
 [44 55 66]]
```

```
print(np.concatenate((arr_2dA , arr_2dB) , axis = 0)) # Horizontal Append [Row-wise]
ValueError: all the input array dimensions except for the concatenation axis must match exactly, but along dimension 1, the array at index 0 has size 3 and the array at index 1 has size 2
```

```
# arange(start,stop,step)
for i in range(1,11):
    print(i, end = " ")
```

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

```
# arange(start,stop,step)
# Reshape(shape)
arr = np.arange(2,21,2) # [2,4,6...20]
arr
```

```
array([ 2,  4,  6,  8, 10, 12, 14, 16, 18, 20])
```

```
# 10 elements -> 2D Matrix [2*5] [5*2]
arr_2d = arr.reshape(2,5)
arr_2d
```

```
array([[ 2,  4,  6,  8, 10],
       [12, 14, 16, 18, 20]])
```

```
# 10 elements -> 2D Matrix [2*5] [5*2]
arr_2d = arr.reshape(5,2)
arr_2d
```

```
array([[ 2,  4],
       [ 6,  8],
       [10, 12],
       [14, 16],
       [18, 20]])
```

```
arr = np.arange(1,17,1) # [1,2,3,4....16]
arr
```

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16])
```

```
# 16 elements -> 2D,3D,4D
arr_2d = arr.reshape(4,4)
arr_2d
```

```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12],
       [13, 14, 15, 16]])
```

```
arr_2d = arr.reshape(8,2)
arr_2d
```

```
array([[ 1,  2],
       [ 3,  4],
       [ 5,  6],
       [ 7,  8],
       [ 9, 10],
       [11, 12],
       [13, 14],
       [15, 16]])
```

```
arr_2d = arr.reshape(2,8)
arr_2d
```

```
array([[ 1,  2,  3,  4,  5,  6,  7,  8],
       [ 9, 10, 11, 12, 13, 14, 15, 16]])
```

```
arr_3d = arr_2d.reshape(2,4,2)
arr_3d
```

```
array([[[ 1,  2],
        [ 3,  4],
        [ 5,  6],
        [ 7,  8]],

       [[ 9, 10],
        [11, 12],
        [13, 14],
        [15, 16]]])
```

```
arr_3d = arr_2d.reshape(2,2,4)
arr_3d
```

```
array([[[ 1,  2,  3,  4],
        [ 5,  6,  7,  8]],

       [[ 9, 10, 11, 12],
        [13, 14, 15, 16]]])
```

```
arr_3d = arr_2d.reshape(4,2,2)
arr_3d
```

```
array([[[ 1,  2],
        [ 3,  4]],

       [[ 5,  6],
        [ 7,  8]],

       [[ 9, 10],
        [11, 12]],

       [[13, 14],
        [15, 16]]])
```

```
arr_4d = arr_3d.reshape(2,2,2,2)
arr_4d
```

```
array([[[[ 1,  2],
         [ 3,  4]],

        [[ 5,  6],
         [ 7,  8]]],


       [[[ 9, 10],
         [11, 12]],

        [[13, 14],
         [15, 16]]]])
```

```
arr_4d = arr_2d.reshape(2,2,2,2)
arr_4d
```

```
array([[[[ 1,  2],
         [ 3,  4]],

        [[ 5,  6],
         [ 7,  8]]],


       [[[ 9, 10],
         [11, 12]],

        [[13, 14],
         [15, 16]]]])
```

```
arr_4d = arr.reshape(2,2,2,2)
arr_4d
```

```
array([[[[ 1,  2],
         [ 3,  4]],

        [[ 5,  6],
         [ 7,  8]]],


       [[[ 9, 10],
         [11, 12]],

        [[13, 14],
         [15, 16]]]])
```

```
# Flattening an array: n-dimensional to 1D Array
flatten_arr = arr_4d.reshape(-1)
flatten_arr
```

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16])
```

```
# Flattening an array: n-dimensional to 1D Array
flatten_arr = arr_2d.reshape(-1)
flatten_arr
```

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16])
```

```
flatten_arr = arr_3d.reshape(-1)
flatten_arr
```

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16])
```

```
# Splitting an Array
arr = np.arange(2,21,2) # 10 elements [2,4,6...20]
arr
```

```
array([ 2,  4,  6,  8, 10, 12, 14, 16, 18, 20])
```

```
split_arr = np.split(arr , 5)
print(split_arr)
```

```
[array([2, 4]), array([6, 8]), array([10, 12]), array([14, 16]), array([18, 20])]
```

```
split_arr
```

```
[array([2, 4]),
 array([6, 8]),
 array([10, 12]),
 array([14, 16]),
 array([18, 20])]
```

```
split_arr = np.split(arr , 3) # ValueError: array split does not result in an equal division
print(split_arr)
```

```
# Splitting an Array
arr = np.arange(1,16,1) # 15 elements [1,2,3,4...15]
arr
```

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15])
```

```
print(arr)
```

```
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15]
```

```
split_arr = np.split(arr , 3)
print(split_arr)
```

```
[array([1, 2, 3, 4, 5]), array([ 6,  7,  8,  9, 10]), array([11, 12, 13, 14, 15])]
```

```
for data in split_arr:
    print(data)
```

```
[1 2 3 4 5]
[ 6  7  8  9 10]
[11 12 13 14 15]
```

```
# axis = 0 [Horizontal axis] [rows]
arr_2d = np.array([
    [11,22,33,44,55],
    [99,88,77,66,55],
    [11,33,55,77,99],
    [22,44,66,88,10],
    [11,15,17,19,21]
])
split_arr = np.split(arr_2d, 5 , axis = 0)
split_arr
```

```
[array([[11, 22, 33, 44, 55]]),
 array([[99, 88, 77, 66, 55]]),
 array([[11, 33, 55, 77, 99]]),
 array([[22, 44, 66, 88, 10]]),
 array([[11, 15, 17, 19, 21]])]
```

```
for data in split_arr:
    print(data)
```

```
[[11 22 33 44 55]]
[[99 88 77 66 55]]
[[11 33 55 77 99]]
[[22 44 66 88 10]]
[[11 15 17 19 21]]
```

```
print(type(split_arr))
```

```
<class 'list'>
```

```
# axis = 1 [Vertical axis] [cols]
arr_2d = np.array([
    [11,22,33,44,55],
    [99,88,77,66,55],
    [11,33,55,77,99],
    [22,44,66,88,10],
    [11,15,17,19,21]
])
split_arr = np.split(arr_2d, 5 , axis = 1)
split_arr
```

```
[array([[11],
        [99],
        [11],
        [22],
        [11]]),
 array([[22],
        [88],
        [33],
        [44],
        [15]]),
 array([[33],
        [77],
        [55],
        [66],
        [17]]),
 array([[44],
        [66],
```

```
for data in split_arr
    print(data)
```

```
[[11]
 [99]
 [11]
 [22]
 [11]]
[[22]
 [88]
 [33]
 [44]
 [15]]
[[33]
 [77]
 [55]
 [66]
 [17]]
[[44]
 [66]
```

```
# Statistical Operations on Array
arr_2d = np.array([
    [11,22,33,44,55],
    [99,88,77,66,55],
    [11,33,55,77,99],
    [22,44,66,88,10],
    [11,15,17,19,21]
])
np.mean(arr_2d)
```

```
45.52
```

```
# axis = 0 [horizontal Axis] [rows]
np.mean(arr_2d , axis = 0)
```

```
array([30.8, 40.4, 49.6, 58.8, 48. ])
```

```
(11+99+11+22+11)/5
```

```
30.8
```

```
(22+88+33+44+15)/5
```

```
40.4
```

```
# axis = 1 [Vertical Axis] [cols]
np.mean(arr_2d , axis = 1)
```

```
array([33. , 77. , 55. , 46. , 16.6])
```

```
(11+22+33+44+55)/5
```

```
33.0
```

```
# Median
arr_2d = np.array([
    [11,22,33,44,55],
    [99,88,77,66,55],
    [11,33,55,77,99],
    [22,44,66,88,10],
    [11,15,17,19,21]
])
np.median(arr_2d)
```

```
44.0
```

```
# axis = 0 [horizontal Axis] [rows]
np.median(arr_2d , axis = 0)
```

```
array([11., 33., 55., 66., 55.])
```

```
# axis = 1 [Vertical Axis] [cols]
np.median(arr_2d , axis = 1)
```

```
array([33., 77., 55., 44., 17.])
```

```
# standard deviation
arr_2d = np.array([
    [11,22,33,44,55],
    [99,88,77,66,55],
    [11,33,55,77,99],
    [22,44,66,88,10],
    [11,15,17,19,21]
])
np.std(arr_2d)
```

```
29.48236082812908
```

```
# axis = 0 [horizontal Axis] [rows]
np.std(arr_2d , axis = 0)
```

```
array([34.36509857, 25.75732905, 21.85040045, 24.65278889, 31.21538082])
```

```
# axis = 1 [Vertical Axis] [cols]
np.std(arr_2d , axis = 1)
```

```
array([15.55634919, 15.55634919, 31.11269837, 28.42534081,  3.44093011])
```

```
# Min / Max -> Return Smallest and the largest values respectively.
# Min
arr_2d = np.array([
    [11,22,33,44,55],
    [99,88,77,66,55],
    [11,33,55,77,99],
    [22,44,66,88,10],
    [11,15,17,19,21]
])
np.min(arr_2d) # 10
```

```
10
```

```
# axis = 0 [horizontal Axis] [rows]
np.min(arr_2d , axis = 0)
```

```
array([11, 15, 17, 19, 10])
```

```
# axis = 1 [Vertical Axis] [cols]
np.min(arr_2d , axis = 1)
```

```
array([11, 55, 11, 10, 11])
```

```
# Max
arr_2d = np.array([
    [11,22,33,44,55],
    [99,88,77,66,55],
    [11,33,55,77,99],
    [22,44,66,88,10],
    [11,15,17,19,21]
])
np.max(arr_2d) # 99
```

```
99
```

```
# axis = 0 [horizontal Axis] [rows]
np.max(arr_2d , axis = 0)
```

```
array([99, 88, 77, 88, 99])
```

```
# axis = 1 [Vertical Axis] [cols]
np.max(arr_2d , axis = 1)
```

```
array([55, 99, 99, 88, 21])
```

```
# Sum -> Calculating the total of each elements within the range
arr_2d = np.array([
    [11,22,33,44,55],
    [99,88,77,66,55],
    [11,33,55,77,99],
    [22,44,66,88,10],
    [11,15,17,19,21]
])
np.sum(arr_2d)
```

```
1138
```

```
# axis = 0 [horizontal Axis] [rows]
np.sum(arr_2d , axis = 0)
```

```
array([154, 202, 248, 294, 240])
```

```
# axis = 1 [Vertical Axis] [cols]
np.sum(arr_2d , axis = 1)
```

```
array([165, 385, 275, 230,  83])
```

```
# Other ways to Create Numpy Array.
np.zeros(11) # np.zeros(shape)
# 1D array filled with zeros 0. <float>
```

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

```
# Return a new array of given shape and type, filled with zeros.
np.zeros(9, dtype = int)
```

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

```
# 2D Matrix
np.zeros((7,2) , dtype = int)
```

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

```
# 3D Matrix
np.zeros((2,3,2)) # default dtype - float
```

```
array([[[0., 0.],
        [0., 0.],
        [0., 0.]],

       [[0., 0.],
        [0., 0.],
        [0., 0.]]])
```

```
# 4D Matrix
np.zeros((2,2,2,2), dtype = bool) # 0 in bool -> False
```

```
array([[[[False, False],
         [False, False]],

        [[False, False],
         [False, False]]],


       [[[False, False],
         [False, False]],

        [[False, False],
         [False, False]]]])
```

```
# np.ones(shape) -> filled with 1.
# 3D Matrix
np.ones((2,3,4) , dtype = int)
```

```
array([[[1, 1, 1, 1],
        [1, 1, 1, 1],
        [1, 1, 1, 1]],

       [[1, 1, 1, 1],
        [1, 1, 1, 1],
        [1, 1, 1, 1]]])
```

```
# 3D Matrix
np.ones((2,3,4) , dtype = bool) # 1 in bool -> True
```

```
array([[[ True,  True,  True,  True],
        [ True,  True,  True,  True],
        [ True,  True,  True,  True]],

       [[ True,  True,  True,  True],
        [ True,  True,  True,  True],
        [ True,  True,  True,  True]]])
```

```
# np.full(shape, fill_value)
# return a new array of given shape and type, filled with `fill_value`
np.full(11, 9)
```

```
array([9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9])
```

```
# 2D Matrix
np.full((4,5), 11.1)
```

```
array([[11.1, 11.1, 11.1, 11.1, 11.1],
       [11.1, 11.1, 11.1, 11.1, 11.1],
       [11.1, 11.1, 11.1, 11.1, 11.1],
       [11.1, 11.1, 11.1, 11.1, 11.1]])
```

```
# 3D Matrix
np.full((2,4,3), 11+9j)
```

```
array([[[11.+9.j, 11.+9.j, 11.+9.j],
        [11.+9.j, 11.+9.j, 11.+9.j],
        [11.+9.j, 11.+9.j, 11.+9.j],
        [11.+9.j, 11.+9.j, 11.+9.j]],

       [[11.+9.j, 11.+9.j, 11.+9.j],
        [11.+9.j, 11.+9.j, 11.+9.j],
        [11.+9.j, 11.+9.j, 11.+9.j],
        [11.+9.j, 11.+9.j, 11.+9.j]]])
```

```
# 3D Matrix
np.full((2,4,3), 'abc')
```

```
array([[['abc', 'abc', 'abc'],
        ['abc', 'abc', 'abc'],
        ['abc', 'abc', 'abc'],
        ['abc', 'abc', 'abc']],

       [['abc', 'abc', 'abc'],
        ['abc', 'abc', 'abc'],
        ['abc', 'abc', 'abc'],
        ['abc', 'abc', 'abc']]], dtype='<U3')
```

```
# 3D Matrix
np.full((2,4,3), 'Python')
```

```
array([[['Python', 'Python', 'Python'],
        ['Python', 'Python', 'Python'],
        ['Python', 'Python', 'Python'],
        ['Python', 'Python', 'Python']],

       [['Python', 'Python', 'Python'],
        ['Python', 'Python', 'Python'],
        ['Python', 'Python', 'Python'],
        ['Python', 'Python', 'Python']]], dtype='<U6')
```

```
# Identity Matrix -> having digonal elements filled with 1 and other non-diagonal are filled with 0.
# np.identity() -> square matrix
# np.eye() -> Rectangular/Square Matrix, k-factor
# 3*3 Matrix
np.eye(3, dtype=int)
```

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

```
# 3*3 Matrix
np.eye(3, dtype=bool)
```

```
array([[ True, False, False],
       [False,  True, False],
       [False, False,  True]])
```

```
# Rectangular Matrix
# m * n Matrix
np.eye(4,7, dtype=float)
```

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

```
# Rectangular Matrix with k-factor
# m * n Matrix
np.eye(4,7,k=1,dtype=float)
```

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

```
# Rectangular Matrix with k-factor
# m * n Matrix
np.eye(4,7,k=2,dtype=float)
```

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

```
# Rectangular Matrix with k-factor
# m * n Matrix
np.eye(4,7,k=-1,dtype=float)
```

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

```
# Rectangular Matrix with k-factor
# m * n Matrix
np.eye(4,7,k=-2,dtype=float)
```

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

```
# Square Matrix
np.eye(7, k = 0)
```

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

```
# Square Matrix
np.eye(7, k = 1)
```

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

```
# Square Matrix
np.eye(7, k = 4)
```

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

```
# Square Matrix
np.eye(7, k = -5)
```

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

```
# Square Matrix
np.identity(4)
```

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

```
# Return the identity array.
# The identity array is a square array with ones on the main diagonal.
np.identity(7)
```

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

```
# Rectangular Matrix is not allowed ❌❌
np.identity(7,2) # TypeError: Cannot interpret '2' as a data type
```

```
np.identity(7,k=-3) # TypeError: identity() got an unexpected keyword argument 'k'
```

```
# np.arange(start , stop, step)
np.arange(1,11,2) # odd numpy array [1,3,..9]
```

```
array([1, 3, 5, 7, 9])
```

```
np.arange(2,21,2) # Even Numpy Array [2,4...20]
```

```
array([ 2,  4,  6,  8, 10, 12, 14, 16, 18, 20])
```

```
np.arange(1,15,-1) # []
```

```
array([], dtype=int32)
```

```
np.arange(15,0,-2) # [15,13,11...1]
```

```
array([15, 13, 11,  9,  7,  5,  3,  1])
```

```
# np.random -> shape()
dir(np.random)
```

```
 'power',
 'rand',
 'randint',
 'randn',
 'random',
 'random_integers',
 'random_sample',
 'ranf',
 'rayleigh',
 'sample',
 'seed',
 'set_bit_generator',
 'set_state',
 'shuffle',
 'standard_cauchy',
 'standard_exponential',
 'standard_gamma',
 'standard_normal'
```

```
# 1D Matrix
# np.random.random(shape) -> [0,1) -> Range
np.random.random(11)
```

```
array([0.3749287 , 0.9242952 , 0.1686759 , 0.77277538, 0.59295769,
       0.9331255 , 0.39123287, 0.42077007, 0.03007149, 0.62073985,
       0.76482758])
```

```
np.random.random(11).round(2)
```

```
array([0.5 , 0.17, 0.71, 0.98, 0.15, 0.32, 0.34, 0.21, 0.23, 0.89, 0.48])
```

```
# 2D Matrix
np.random.random((5,3)).round(2)
```

```
array([[0.81, 0.37, 0.8 ],
       [0.67, 0.12, 0.81],
       [0.63, 0.  , 0.78],
       [0.84, 0.81, 0.82],
       [0.18, 0.08, 0.15]])
```

```
# 3D Matrix
# Return random integers from `low` (inclusive) to `high` (exclusive)
np.random.randint(100,1000,(2,4,3)) # [100,1000)
```

```
array([[[429, 831, 887],
        [999, 494, 432],
        [205, 428, 500],
        [506, 132, 677]],

       [[770, 827, 593],
        [105, 838, 954],
        [902, 767, 463],
        [882, 481, 275]]])
```

```
# linspace(start,stop,number of elements) # Evenly spaced data
np.linspace(1,10,10)
```

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

```
np.linspace(1,10,20)
```

```
array([ 1.        ,  1.47368421,  1.94736842,  2.42105263,  2.89473684,
        3.36842105,  3.84210526,  4.31578947,  4.78947368,  5.26315789,
        5.73684211,  6.21052632,  6.68421053,  7.15789474,  7.63157895,
        8.10526316,  8.57894737,  9.05263158,  9.52631579, 10.        ])
```

```
1.94736842 - 1.47368421
```

```
0.47368421000000005
```

```
1.94736842 + 0.47368421
```

```
2.42105263
```

```
np.linspace(1,10,19)
```

```
array([ 1. ,  1.5,  2. ,  2.5,  3. ,  3.5,  4. ,  4.5,  5. ,  5.5,  6. ,
        6.5,  7. ,  7.5,  8. ,  8.5,  9. ,  9.5, 10. ])
```

```
np.linspace(1,100,51)
```

```
array([  1.  ,   2.98,   4.96,   6.94,   8.92,  10.9 ,  12.88,  14.86,
        16.84,  18.82,  20.8 ,  22.78,  24.76,  26.74,  28.72,  30.7 ,
        32.68,  34.66,  36.64,  38.62,  40.6 ,  42.58,  44.56,  46.54,
        48.52,  50.5 ,  52.48,  54.46,  56.44,  58.42,  60.4 ,  62.38,
        64.36,  66.34,  68.32,  70.3 ,  72.28,  74.26,  76.24,  78.22,
        80.2 ,  82.18,  84.16,  86.14,  88.12,  90.1 ,  92.08,  94.06,
        96.04,  98.02, 100.  ])
```

What is Pandas?

A high level data manipulation tools build on Numpy And Matplotlib.

It is used to :

- Import / Export Data Easily.
- Clean and Analyze the Data.
- Perform Statistical Operations.
- Visualize the data.

Why is Pandas Important?

1. Simple Syntax for Complex Task.
2. Effecient Operations using Numpy in Backend.
3. Work with multiple formats - .csv , .excel , .json, .sql
4. Data Cleaning - Handle Missing or Inconsistent Value
5. Powerful Analysis Tools - Filtering , Grouping , Pivoting , Melting , Aggregations, etc..

```
pip install pandas
conda install pandas
```

It has 2 types of Structures:

1. Series -> One Dimensional Array (Like One Column)
2. DataFrame -> Two Dimensional Array (Like a Table having rows or cols) {Spreadsheet}

```
import pandas as pd
```

What is Series?

A Series is :

1. A 1D Labelled Array of the data
2. Each element has an index
3. Can Store int , float, str, bool and object
4. Mutable (values can be updated)

```
Note: Think of it as a single column from an Excel Sheet [Univariate Analysis]
```