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

```
[11,22,33,44,55],
[99,88,77,66,55],
[11,33,55,77,99],
[22,44,66,88,10],
[10,33,66,77,99]
```

axis = 0 [rows]

```
row-0 [11,22,33,44,55]
row-1 [99,88,77,66,55]
row-2 [11,33,55,77,99]
row-3 [22,44,66,88,10]
row-4 [10,33,66,77,99]

[30.6, 44., 59.4, 70.4, 63.6]
```

```python
# Statistical Operations on Array

import numpy as np
arr_2d = np.array([
    [11,22,33,44,55],
    [99,88,77,66,55],
    [11,33,55,77,99],
    [22,44,66,88,10],
    [10,33,66,77,99]
])
np.mean(arr_2d)
```

```
53.6
```

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

```
array([30.6, 44. , 59.4, 70.4, 63.6])
```

```python
# axis = 1 [Vertical axis => Columns]
np.mean(arr_2d , axis = 1)
```

```
array([33., 77., 55., 46., 57.])
```

```python
# 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],
    [10,33,66,77,99]
])
np.median(arr_2d)
```

```
55.0
```

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

```
array([11., 33., 66., 77., 55.])
```

```python
# axis = 1 [Vertical axis => Columns]
np.median(arr_2d , axis = 1)
```

```
array([33., 77., 55., 44., 66.])
```

```python
# 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],
    [10,33,66,77,99]
])
np.std(arr_2d)
```

```
29.357111574540163
```

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

```
array([34.48245931, 23.07379466, 14.92112596, 14.92112596, 33.24815784])
```

```python
# axis = 1 [Vertical axis => Columns]
np.std(arr_2d , axis = 1)
```

```
array([15.55634919, 15.55634919, 31.11269837, 28.42534081, 31.71750305])
```

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

```
10
```

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

```
array([10, 22, 33, 44, 10])
```

```python
# axis = 1 [Vertical axis => Columns]
np.min(arr_2d , axis = 1)
```

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

```python
arr_2d = np.array([
    [11,22,33,44,55],
    [99,88,77,66,55],
    [11,33,55,77,99],
    [22,44,66,88,10],
    [10,33,66,77,99]
])
np.max(arr_2d)
```

```
99
```

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

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

```python
# axis = 1 [Vertical axis => Columns]
np.max(arr_2d , axis = 1)
```

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

```python
# Sum -> Calculating the totals of all elements
arr_2d = np.array([
    [11,22,33,44,55],
    [99,88,77,66,55],
    [11,33,55,77,99],
    [22,44,66,88,10],
    [10,33,66,77,99]
])
np.sum(arr_2d)
```

```
1340
```

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

```
array([153, 220, 297, 352, 318])
```

```python
# axis = 1 [Vertical axis => Columns]
np.sum(arr_2d , axis = 1)
```

```
array([165, 385, 275, 230, 285])
```

```python
# Other Ways to Create Numpy Arrays.
np.zeros(11) # 1D Array filled with 11 zeros
```

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

```python
np.zeros(9, dtype=int)
```

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

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

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

```python
# np.ones(shape) -> which dimension -> ndim
np.ones(11)
```

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

```python
# 2D Matrix
np.ones((4,4) , dtype = int)
```

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

```python
# 3D Matrix
np.zeros((2,3,2))
```

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

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

```python
# 4D Matrix
np.zeros((2,2,2,2) , dtype = bool)
```

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

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


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

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

```python
# 3D Matrix
np.ones((2,3,4) , dtype = bool)
```

```
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]]])
```

```python
# np.full(shape , value) -> Custom Value Fill
# 1D Matrix
np.full(11 , 9)
```

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

```python
# 2D Matrix
np.full((4,5) , 11)
```

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

```python
# 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]]])
```

```python
# 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')
```

```
# Identity Matrix -> having diagonal elements filled with 1 and other non-diagonal are filled with 0
# np.identity() -> Square Matrix
# np.eye() -> Rectangular/Square Matrix , k-factor
```

```
# 3 -> 3X3 Matrix
np.eye(3 , dtype = int)
```

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

```
np.eye(3 , dtype = bool)
```

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

```
# Rectangular 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
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
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
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
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.]])
```

```
# Diagonal 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.]])
```

```
# Diagonal 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.]])
```

```
# Diagonal 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.]])
```

```
# Square Matrix
np.identity(4,3) # TypeError: Cannot interpret '3' as a data type
```

```
np.identity(4 , k = 2) # TypeError: identity() got an unexpected keyword argument 'k'
```

```
# arange(start = 0 , stop = length[Non Inclusive] , step[1 by default])
np.arange(1,11,2)
```

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

```
np.arange(2,21,2)
```

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

```
np.arange(1,15,-2)
```

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

```
np.arange(15,0,-2)
```

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

```
# np.random
dir(np.random)
```

```
['BitGenerator',
 'Generator',
 'MT19937',
 'PCG64',
 'PCG64DXSM',
 'Philox',
 'RandomState',
 'SFC64',
 'SeedSequence',
 '__RandomState_ctor',
 '__all__',
 '__builtins__',
 '__cached__',
 '__doc__',
 '__file__',
 '__loader__',
 '__name__',
 '__package__',
 '__path__',
 '__spec__',
```

```
# 1D Matrix
# np.random.random(shape) -> (0,1) range
np.random.random(11).round(2)
```

```
array([0.38, 0.72, 0.46, 0.8 , 0.24, 0.39, 0.52, 0.7 , 0.22, 0.41, 0.92])
```

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

```
array([[0.54, 0.76, 0.82],
       [0.89, 0.48, 0.52],
       [0.11, 0.87, 0.65],
       [0.22, 0.08, 0.61],
       [0.08, 0.01, 0.85]])
```

```
# linspace(start, stop , num) # 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.        ])
```

```
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(1,100,50)
```

```
array([  1.        ,   3.02040816,   5.04081633,   7.06122449,
         9.08163265,  11.10204082,  13.12244898,  15.14285714,
        17.16326531,  19.18367347,  21.20408163,  23.2244898 ,
        25.24489796,  27.26530612,  29.28571429,  31.30612245,
        33.32653061,  35.34693878,  37.36734694,  39.3877551 ,
        41.40816327,  43.42857143,  45.44897959,  47.46938776,
        49.48979592,  51.51020408,  53.53061224,  55.55102041,
        57.57142857,  59.59183673,  61.6122449 ,  63.63265306,
        65.65306122,  67.67346939,  69.69387755,  71.71428571,
        73.73469388,  75.75510204,  77.7755102 ,  79.79591837,
        81.81632653,  83.83673469,  85.85714286,  87.87755102,
        89.89795918,  91.91836735,  93.93877551,  95.95918367,
        97.97959184, 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]
```

```python
# Series(data , index) [By Default indexing is 0 based indexing]
data = [11,22,33,44,55,66,77,88,99]
series = pd.Series(data)
series
```

```
0    11
1    22
2    33
3    44
4    55
5    66
6    77
7    88
8    99
dtype: int64
```

```python
data = [11,22,33,44,55,66,77,88,99]
label = ['a','b','c','d','e','f','g','h','i']
series = pd.Series(data , label)
series
```

```
a    11
b    22
c    33
d    44
e    55
f    66
g    77
h    88
i    99
dtype: int64
```

```python
data = [11,22,33,44,55,66,77,88,99]
label = range(1,10) # [1,.....9]
series = pd.Series(data , label)
series
```

```
1    11
2    22
3    33
4    44
5    55
6    66
7    77
8    88
9    99
dtype: int64
```

```python
# Dictionary [Key [Label/Index] : Value[Data]]
_employee_dict = {
    'emp_id' : 'emp101',
    'name' : 'Utkarsh',
    'age' : 29,
    'gender' : 'M',
    'salary' : '$10,00,000',
    'designation' : 'Senior Analyst',
    'email' : 'utk232@gmail.com',
    'state' : 'Delhi',
    'country' : 'India'
}
series = pd.Series(_employee_dict)
series
```

```
emp_id                   emp101
name                     Utkarsh
age                           29
gender                         M
salary                $10,00,000
designation       Senior Analyst
email           utk232@gmail.com
state                      Delhi
country                    India
dtype: object
```

```python
print(type(series))
```

```
<class 'pandas.core.series.Series'>
```

```python
# How to access an elements from a Series
# .iloc [Positional Based Indexing]
# .loc [Label Based Indexing] [Inclusive]

data = [11,22,33,44,55,66,77,88,99]
series = pd.Series(data)
print(series)
# print(series[-1]) # KeyError [index can't be negative in pd.Series]
print(series[6]) # 77
print(series.iloc[0]) # 11
print(series.iloc[-1]) # 99
print(series.loc[4]) # 55 [label_based]
```

```
0    11
1    22
2    33
3    44
4    55
5    66
6    77
7    88
8    99
dtype: int64
77
11
99
55
```

```python
data = [11,22,33,44,55,66,77,88,99]
label = ['a','b','c','d','e','f','g','h','i']
series = pd.Series(data , label)
print(series)
# print(series[0]) # KeyError
print(series['a']) # 11
print(series.iloc[-3]) # 77
# print(series.Loc[7]) # Searching the index [which doesn't have 7 in it.] # KeyError
print(series.loc['e']) # 55
```

```
a    11
b    22
c    33
d    44
e    55
f    66
g    77
h    88
i    99
dtype: int64
11
77
55
```

```python
# slicing
data = [11,22,33,44,55,66,77,88,99]
label = ['a','b','c','d','e','f','g','h','i']
series = pd.Series(data , label)
print(series)
print("\n Label Based Slicing using .loc")
print(series.loc['a':'e']) # Both Inclusive [11,22,33,44,55]
print("\n Positional Based Slicing using .iloc")
print(series.iloc[0:6]) # 6 is non-inclusive [11,22,33,44,55,66]
```

```
a    11
b    22
c    33
d    44
e    55
f    66
g    77
h    88
i    99
dtype: int64

 Label Based Slicing using .loc
a    11
b    22
c    33
d    44
e    55
dtype: int64
```

```
 Positional Based Slicing using .iloc
a    11
b    22
c    33
d    44
e    55
f    66
dtype: int64
```