UNIT-4

Introduction to NumPy:-

NumPy stands for Numerical Python. it is a powerful python library used for numerical and mathematical computations or calculations. NumPy provides a special data structure called ndarray(N-dimensional array), which is used to store and manipulate large amounts of numerical data efficiently.

NumPy is faster and more memory-efficient than normal python lists. it supports multi-dimensional arrays and provides many built-in mathematical functions such as sum, minimum, maximum, and more.

NumPy is widely used in data science, machine learning, statistics and scientific research. many popular libraries like TensorFlow and Scikit-learn are built using NumPy.

install NumPy using pip:-

pip install numpy

Ex:- (sub module of numpy)

```
import numpy as np
print(dir(np))
```

Array in NumPy:-

in NumPy, an array is a special data structure used to store multiple values of the same data type in a single variable. it is called ndarray(N-dimensional array)

NumPy arrays are faster and more memory-efficient than normal python lists.

Types of NumPy Arrays:-

1. one-Dimensional Array(1D array)
2. Two-Dimensional Array(2D array)
3. Three-Dimensional Array (3D array)

1. one-Dimensional Array(1D array):-

a 1D array in NumPy is a simple array that contains elements arranged in a single row. it is created using the np.array() function from the NumPy library.

a 1D array stores elements of the same data type and is accessed using a single index value starting from 0.

- It has only one axis, so it is called a 1D array.

- It is similar to a normal python list, but it performs operations faster.
- Each element in the array can be accessed using a single index value.
- They are mainly used in basic mathematical and statistical calculations.

Ex:-

```
import numpy as np
a = np.array([10, 20, 30, 40])
print(a)
```

2. Two-Dimensional Array(2D array):-

a 2D array in NumPy is an array that contains elements arranged in rows and columns. it looks like a table or matrix.

in NumPy, 2D arrays are created using the np.array() function by passing a list of lists.

- It has two axes, one for rows and one for columns.
- It is similar to a matrix or table structure.
- Each element is accessed using two index values(row index and column index)
- 2D arrays are widely used in data analysis and machine learning
- They are also used to represent tables, spreadsheets and datasets.

Ex:-

```
import numpy as np
a = np.array([[1, 2, 3],
              [4, 5, 6]])
print(a)
```

3. Three-Dimensional Array (3D array):-

A 3D array in NumPy is a collection of multiple 2D arrays arranged in depth rows, and columns. it is also called an N-dimensional array(ndarray). A 3D array is accessed using three index values: block index, row index, and column index. All inner arrays must have the same number of elements, and all elements must be of the same data type.

- It has three axes, representing layers, rows and columns.
- It is used to represent complex data structures.
- 3D arrays are commonly used in image processing and scientific computing.

Ex:-

```
import numpy as np
a = np.array([
```

```
        [[1, 2], [3, 4]],
        [[5, 6], [7, 8]]
       ])
print(a)
```

4. Multidimensional Array (n- Dimensional Array):-

- A multidimensional array contains more than three dimensions
- In numpy, these arrays are called ndarrays.
- They are used to represent highly complex datasets.
- Multidimensional arrays are widely used in machine learning and Artificial Intelligence.
- They allow efficient storage and manipulation of large data.

Ex:-

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

Indexing and Slicing in NumPy:-

indexing:-

indexing in NumPy is the process of accessing individual elements of an array using their index positions. the index number always starts from 0, which means the first element of the array is at index 0.

Ex:-

```
import numpy as np
a=np.array([10,20,30,40,50])
print(a[0])
print(a[3])
```

Ex:-

```
import numpy as np
a=np.array([10,20,30,40,50])
print(a[-1])
print(a[-2])
```

Ex:-

```
import numpy as np
a=np.array([[10,20,30],[40,50,60],[70,80,90]])
print(a[0,1])
print(a[2,2])
```

Slicing:-

slicing in NumPy is the process of accessing a group of elements from an array. instead of accessing a single element like indexing, slicing allows us to select multiple elements at a time.

Ex:-

```
import numpy as np
a=np.array([10,20,30,40,50,60])
print(a[1:4])
```

Ex:-

```
import numpy as np
a=np.array([[12,56,89],[96,32,41]])
print(a[0:2,2:3])
```

Operations on Arrays:-

Array operations in NumPy are used to perform mathematical calculations on array elements easily and quickly. NumPy allows operations like addition, subtraction, multiplication, and division directly on arrays without using loops.

When operations are performed between two arrays, the operation is applied element-wise, meaning each element in one array operates with the corresponding element in the other array.

1. Addition of Arrays:-

```
import numpy as np
a=np.array([10,20,30])
b=np.array([1,2,3])
print(a+b)
```

Output

```
[11 22 33]
```

Ex:-

```
import numpy as np
a=np.array([[[2,4,6],[2,3,2],[4,5,2],[1,3,2]]])
b=np.array([[[6,4,2],[2,3,1],[2,4,6],[2,4,3]]])
print(a+b)
```

output:-

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

2. Subtraction of Arrays:-

```
import numpy as np
a=np.array([10,20,30])
b=np.array([1,2,3])
print(a-b)
```

3. Multiplication of Arrays:-

```
import numpy as np
a=np.array([10,20,30])
b=np.array([1,2,3])
print(a*b)
```

4. Division of Arrays:-

```
import numpy as np
a=np.array([10,20,30])
b=np.array([1,2,3])
print(a/b)
```

5. sum of array elements:-

used to find the total sum of elements in an array.

Ex:-

```
import numpy as np
a=np.array([10,20,30,40])
print(np.sum(a))
```

6. Maximum value:-

used to find the largest element in an array.

Ex:-

```
import numpy as np
a=np.array([10,20,30,40])
print(np.max(a))
```

7. Minimum value

used to find the smallest element in an array.

Ex:-

```
import numpy as np
a=np.array([10,20,30,40])
print(np.min(a))
```

8. Mean (Average):-

used to calculate the average value of an array elements.

Ex:-

```
import numpy as np
a=np.array([10,20,30,40])
print(np.mean(a))
```

9. Square Root Operation:-

used to find the square root of each element.

Ex:-

```
import numpy as np
a=np.array([10,20,30,40])
print(np.sqrt(a))
```

10. Power Operation:-

used to raise elements to a specified power.

ex:-

```
import numpy as np
a=np.array([10,20,30,40])
print(np.power(a,2))
```

Concatenating Arrays in NumPy:-

concatenating arrays means joining two or more arrays together to form a single array. in NumPy , this is done using the numpy.concatenate() function. it is commonly used when we want to combine data from multiple arrays into one array for further processing or analysis.

Syntax

np.concatenate((array1, array2), axis)

Parameters:

array1, array2 → Arrays that we want to join
axis → Direction in which arrays will be joined
axis = 0 → Join arrays row-wise (vertical concatenation)
axis = 1 → Join arrays column-wise (horizontal concatenation)
Ex:-

```
import numpy as np
a = np.array([10,20,30])
b = np.array([40,50,60])
c = np.concatenate((a,b))
print(c)
```

1). Row-wise concatenation (axis=0)

in row-wise concatenation, arrays are joined one below another. for this operation, the arrays must have the same number of columns.

Ex:-

```
import numpy as np
a = np.array([[1,2,3],
        [4,5,6]])
b = np.array([[7,8,9],
        [10,11,12]])
c = np.concatenate((a,b), axis=0)
print(c)
```

2). Column-wise concatenation(axis=1):-

in column-wise concatenation, arrays are joined side by side. for this operation, the arrays must have the same number of rows.

Ex:-

```
import numpy as np
a = np.array([[1,2],
              [3,4]])
b = np.array([[5,6],
              [7,8]])
c = np.concatenate((a,b), axis=1)
print(c)
```

Ex:-

```
import numpy as np
a=np.array([[1,2,3],[4,5,6]])
print(a)
print('size:', a.size)
print("dimensions:", a.ndim)
b=np.array([[7,8,9],[10,11,12]])
print(b)
print('size:', b.size)
print("dimensions:", b.ndim)
c=np.concatenate((a,b), axis=0)
print(c)
print('size:', c.size)
print("dimensions:", c.ndim)
```

Reshaping Arrays in NumPy:-

Reshaping arrays means changing the structure(shape) of an array without changing its data. in NumPy, reshaping allows us to convert an array from one dimension to another dimension, such as converting a 1D array into a 2D or 3D array.

Reshaping is done using the reshape() method. this method reorganizes the elements of an array into a new shape while keeping the same number of elements.

Syntax:-

```
array.reshape(rows, columns)
```

Ex:-

```
import numpy as np
a = np.array([1,2,3,4,5,6])
```

```
b = a.reshape(2,3)
print(b)
```

Output

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

Explanation:

Original → 1D array

Reshape → 2 rows and 3 columns

Ex:-

```
import numpy as np
a=np.array([1,2,3,4,5,6,7,8])
b=a.reshape(2,4)
print(b)
```

Ex:- **(Convert to 3D Array)**

```
import numpy as np
a=np.array([1,2,3,4,5,6,7,8])
b=a.reshape(2,2,2)  # 2 blocks,2 rows, 2columns
print(b)
```

Ex:-

```
import numpy as np
my_list=np.array([10,20,30,40,50,60,70,80])
print("size:", my_list.size)
print("dimensions:", my_list.ndim)
res=my_list.reshape(4,2)
print(res)
print("Dimensions:",res.ndim)
```

Splitting Arrays:-

Splitting arrays is an important operation in NumPy used to divide a large array into multiple smaller arrays. it helps in organizing and processing data more efficiently. NumPy provides several functions such as split(), array_split(), hsplit() and vsplit() to perform splitting operations. these functions allow arrays to be divided either row-wise or column-wise depending on the requirement.

Functions used for splitting:-

1. split()
2. array_split()
3. hsplit(Horizontal split)
4. vsplit(Vertical split)

1. split():-

the split() function in NumPy is used to divide an array into multiple equal parts. it splits the array into smaller sub-arrays based on the number of sections specified by the user. the important condition of the split() function is that the array must be divided equally, otherwise NumPy will generate an error.

Ex:-

```
import numpy as np
a=np.array([10,20,30,40,50,60])
b=np.split(a,3)
print(b)
```

2. array_split():-

The array_split() function in NumPy is used to divide an array into multiple smaller arrays. It is similar to the split() function, but array_split() is more flexible because it can split arrays even when the parts are not equal. This function helps in organizing and processing large arrays easily.

Ex:-

```
import numpy as np
a = np.array([1,2,3,4,5])
b = np.array_split(a,3)
print(b)
```

Ex:-

```
import numpy as np
a = np.array([[10,20,30,40],
        [50,60,70,80]])
b = np.array_split(a,2,axis=1)
print(b)
```

3. hsplit(Horizontal split):-

the hsplit() function in NumPy is used to split an array horizontally into multiple sub-arrays.

horizontal splitting means the array is divided column-wise (left to right). it is mainly used with two-dimensional arrays to separate columns into different arrays.

Ex:-

```
import numpy as np
a = np.array([[10,20,30,40],
        [50,60,70,80]])
b = np.hsplit(a,2)
print(b)
```

Ex:-

```
import numpy as np
a = np.array([[1,2,3,4],
        [5,6,7,8]])
b = np.hsplit(a,4)
print(b)
```

4. vsplit(Vertical split):-

The vsplit() function in NumPy is used to split an array vertically into multiple sub-arrays. Vertical splitting means the array is divided row-wise (top to bottom). It is mainly used with two-dimensional arrays to separate rows into different arrays.

Ex:-

```
import numpy as np
a = np.array([[10,20],
        [30,40],
        [50,60],
```

```
        [70,80]])
b = np.vsplit(a,2)
print(b)
```

Statistical Operations on Arrays:-

Statistical Operations on Arrays are used to perform mathematical and statistical calculations on array elements. NumPy provides many built-in functions to calculate values such as mean, median, sum, minimum, maximum, and standard deviation. these operations help in analysing and summarizing large datasets easily.
statistical operations are commonly used in data analysis, machine learning, and scientific computing.

1). Sum of Array Elements (np.sum()):-

the sum() function is used to calculate the total addition of all elements in the array.

Ex:-

```
import numpy as np
a = np.array([10,20,30,40])
print(np.sum(a))
```

Ex:-

```
import numpy as np
a=np.array([[10,20,30,40],[1,2,3,4]])
print(np.sum(a))
```

Result:- 110

Ex:-

```
import numpy as np
a=np.array([[10,20,30,40],[1,2,3,4]])
print(np.sum(a, axis=0))
```

result :- [11 22 33 44]

(axis=0 --> column wise sum)
(axis=1 --> row wise sum)

Ex:-
```
import numpy as np
a=np.array([[10,20,30,40],[1,2,3,4]])
print(np.sum(a, axis=1))
```

Result:- [100 10]

2). Mean/ Average (np.mean())

the mean() function calculates the average value of array elements.
formula:-
mean=sum of elements / number of elements

Ex:-
```
import numpy as np
a = np.array([10,20,30,40])
print(np.mean(a))
```

Ex:-
```
import numpy as np
a=np.array([[10,20,30,40],[1,2,3,4]])
print(np.mean(a))
```

Sum of all elements:-
10 + 20 + 30 + 40 + 1 + 2 + 3 + 4=110

Number of elements:-
8

mean=110/8
mean=13.75

Ex:-
```
import numpy as np
a=np.array([[10,20,30,40],[1,2,3,4]])
print(np.mean(a, axis=0))
```

(10 + 1) / 2 = 5.5

(20 + 2) / 2 = 11
(30 + 3) / 2 = 16.5
(30 + 3) / 2 = 16.5
output:- [5.5 11. 16.5 22.]

Ex:-
```
import numpy as np
a=np.array([[10,20,30,40],[1,2,3,4]])
print(np.mean(a, axis=1))
```

Row1:-
(10+20+30+40)/4 = 25
Row2:-
(1+2+3+4)/4 = 2.5
Output:-
[25. 2.5]

3). Median (np.median()):-

the median() function returns the middle value of sorted data.
Ex:-
```
import numpy as np
a = np.array([5,10,15,20,25])
print(np.median(a))
```

ex:-
```
import numpy as np
a = np.array([[5,10,15,20,25],[1,2,3,4,5]])
print(np.median(a))
```

Explanation:-

Values list:
5,10,15,20,25,1,2,3,4,5
sorting the values:-(median first sorts the elements in ascending order)
[1,2,3,4,5,5,10,15,20,25]
find the middle values:-
number of elements=10
Median = average of the two middle values

Middle positions:
5th value = 5
6th value = 5
Median Calculation
Median = (5 + 5) / 2
Median = 10 / 2
Median = 5.0
Final Output:-
5.0

4). Minimum Value(np.min()):-

the min() function finds the smallest value in the array.

Ex:-

```
import numpy as np
a = np.array([8,3,15,6,2])
print(np.min(a))
```

Ex:-

```
import numpy as np
a=np.array([[12,52,30,8],[75,20,12,4]])
print(np.min(a))
```

Ex:-

```
import numpy as np
a=np.array([[12,52,30,8],[75,20,12,4]])
print(np.min(a, axis=0))
```

Result:- [12 20 12 4]

Ex:-

```
import numpy as np
a=np.array([[12,52,30,8],[75,20,12,4]])
print(np.min(a, axis=1))
```

Result:- [8 4]

5). Maximum value(np.max()):-

the max() function finds the largest value in the array.

Ex:-

```
import numpy as np
a = np.array([10,25,5,40,15])
print(np.max(a))
```

Ex:-

```
import numpy as np
a=np.array([[12,52,30,8],[75,20,12,4]])
print(np.max(a))
```

6. Standard Deviation(np.std()):-

- **Standard deviation** measures how much the values in the dataset **deviate from the mean**.
- It shows how **spread out the data values are**.
- A small standard deviation means values are **close to the mean**.
- A large standard deviation means values are **spread far from the mean**.
- In NumPy, the **std()** function is used to calculate it.

Ex:-

```
import numpy as np

a = np.array([10, 20, 30, 40, 50])
print(np.std(a))
```

7. Variance(np.var()):-

- **Variance** measures how far each number in the dataset is from the mean.
- It is the **square of the standard deviation**.
- Variance helps in understanding the **spread of data values**.
- In NumPy, the **var()** function is used to calculate variance.
- A higher variance means the data values are **more spread out**.

Ex:-

```
import numpy as np

a = np.array([10, 20, 30, 40, 50])
print(np.var(a))
```

8. Percentile (np.percentile())

- A **percentile** is a statistical measure that indicates the value below which a certain **percentage of data values fall**.
- It helps in understanding the **distribution of data** and where a particular value stands in the dataset.
- In NumPy, the **np.percentile()** function is used to calculate the percentile of an array.

Ex:-

```
import numpy as np
a = np.array([10, 20, 30, 40, 50])
p = np.percentile(a, 25)
print(p)
```

Ex:- All statistical operations)

```
import numpy as np

a = np.array([10,20,30,40,50])

print("Sum:",np.sum(a))
print("Mean:",np.mean(a))
print("Median:",np.median(a))
print("Min:",np.min(a))
print("Max:",np.max(a))
print("Standard Deviation:",np.std(a))
print("Variance:",np.var(a))
```

Data Handling in Python:-

data handling in python refers to the process of collecting, storing, organizing, processing and analyzing data using python tools like files, data structures and libraries (NumPy, Pandas).

in addition, python provides built-in data structures like lists, tuples, dictionaries and sets, which help in storing and organizing data as NumPy and pandas that make data processing, analysis, and manipulation faster and easier.

steps in Data Handling:-

1. Data Collection (collect data from different sources)
2. Data Storage (store data in files or databases)
3. Data Processing (perform operations on data)
4. Data Analysis (find useful information)
5. Data Presentation(show results using tables or charts)

Pandas:-

pandas is an open-source python library used for data manipulation, data analysis, and data handling. it is one of the most important libraries in python, especially for working with structured or tabular data such as data stored in spreadsheets, CSV files, and databases.

the name "pandas" is derived from the term "panel data", which refers to multi-dimensional data used in statistics and economics. pandas was developed by Wes McKinney to provide a fast, flexible and easy-to-use data analysis tool for python.

pandas provides powerful and easy-to-use data structures such as series(one-dimensional or single column) and DataFrame(two-dimensional or table format). these structures allow users to organize, filter, clean and analyse data efficiently.
one of the key advantage of pandas is its ability to work with different file formats like CSV, Excel, JSON and SQL databases.

pandas is widely used in various fields such as data science, machine learning, business analytics, finance and research.

installing pandas:-

pip install pandas

check installation:-

```
import pandas as pd
print(pd.__version__)
```

Series:-

A series in pandas is a one-dimensional labelled array that can store data of any type such as integers, floats, strings, etc.
in simple series is a single column of data with labels(index). each element in a series has value(data) and index (label).

Ex:-

```
import pandas as pd
data=[10,20,30]
s=pd.Series(data)
print(s)
```

Ex:- (with custom index)

```
import pandas as pd
s=pd.Series([10,20,30],index=["a", "b","c"])
print(s)
```

Ex:- (from dictionary)

```
import pandas as pd
data={'Math':90,'Science':85}
s=pd.Series(data)
print(s)
```

Accessing Elements:-

Ex:- (using index(label))

```
import pandas as pd
s=pd.Series([10,20,30], index=['a','b','c'])
print(s['a'])
```

Ex:- (using position)

```
import pandas as pd
s=pd.Series([10,20,30])
print(s[1])
```

Operations on Series:-

1. Arithmetic Operations:-

EX:-

```
import pandas as pd
```

```
s=pd.Series([10,20,30])
print(s+5)
print(s-5)
print(s *2)
print(s /2)
```

Ex:-

```
import pandas as pd
s1=pd.Series([10,20,30])
s2=pd.Series([1,2,3])
print(s1+s2)
```

Ex:- (Comparison Operations)

```
import pandas as pd
s=pd.Series([10,20,30,40])
print(s >20)
```

Ex:-

```
import pandas as pd
s=pd.Series([10,20,30,40,20])
print(s == 20)
```

Ex:-

```
import pandas as pd
s=pd.Series([10,20,30,40,20])
print(s != 20)
```

Ex:- (statistical operations)

```
import pandas as pd
s=pd.Series([10,20,30,40,50])
print(s)
print('total')
print(s.sum())
print('Average(mean)')
print(s.mean())
print('max')
print(s.max())
```

```
print('min')
print(s.min())
print('count')
print(s.count())
print('median')
print(s.median())
```

important Attributes:-

attributes in a pandas series are properties used to get information about the object. they help us understand details such as size, index labels, data type, and values stored in the series.

1.index attribute

the index attribute returns the labels of the series elements.

Ex:- (displaying index range)

```
import pandas as pd
s=pd.Series([10,20,30])
print(s.index)
```

2. values attribute:-

the values attribute returns the actual data stored in the series as a numpy array.

Ex:-

```
import pandas as pd
s=pd.Series([10,20,30,'sgdc'])
print(s.values)
```

3. dtype attribute:-

the dtype attribute shows the data type of the elements in the series.

Ex:-

```
import pandas as pd
s=pd.Series([10,20,30])
print(s.dtype)
```

Ex:-

```
import pandas as pd
s=pd.Series([10,20,30,"hello"])
print(s)
print(s.dtype)
```

result :- **the dtype becomes object.**
(in pandas series, all elements should ideally have the same data type. but if we mix numbers and strings, pandas cannot keep the data as integer or float type. so pandas converts the entire series to object type.
object type means the series can store different type of data (like strings, numbers, etc.)

4. shape attribute:-
the shape attribute returns the dimension of the series.
Ex:-
```
import pandas as pd
s=pd.Series([10,20,30])
print(s.shape)
```

5. size attribute:-
the size attribute returns the total number of elements in the series.
Ex:-
```
import pandas as pd
s=pd.Series([10,20,30])
print(s.size)
```

6. ndim attribute:-
the ndim attribute returns the number of dimensions of the series.
Ex:-
```
import pandas as pd
s=pd.Series([10,20,30])
print(s.ndim)
```

7.name attribute:-
the name attribute returns the name of the series.
Ex:-
```
import pandas as pd
s=pd.Series([10,20,30], name="Marks")
print(s.name)
print(s)
```

important Functions:-

functions in a pandas series are used to perform operations on the data stored in the series. these functions help in analysing, modifying, and retrieving information from the data.

functions are written with parentheses().

1.head():-

the head() function returns the first few elements of the series. by default, it shows first 5 elements.

Ex:-

```
import pandas as pd
s=pd.Series([10,20,30,40,50,60])
print(s.head())
```

Ex:-

```
print(s.head(3))
```

2. tail() :-

the tail() function returns the last few elements of the series.

Ex:-

```
import pandas as pd
s=pd.Series([10,20,30,40,50,60])
print(s.tail())
```

3. sum() :-

the sum() function calculates the total of all elements.

Ex:-

```
import pandas as pd
s=pd.Series([10,20,30,40,50,60])
print(s.sum())
```

4. mean() :-

the mean() function calculates the average value.

Ex:-

```
import pandas as pd
s=pd.Series([10,20,30,40,50,60])
print(s.mean())
```

5. max():-

the max() function finds the maximum value.

Ex:-

```
import pandas as pd
s=pd.Series([10,20,30,40,50,60])
print(s.max())
```

6. min() :-

the min() function finds the minimum value.

Ex:-

```
import pandas as pd
s=pd.Series([10,20,30,40,50,60])
print(s.min())
```

7. count():-

the count() function returns the number of elements in the series.

Ex:-

```
import pandas as pd
s=pd.Series([10,20,30,40,50,60])
print(s.count())
```

8. describe() :-

the describe() function provides a summary of statistical information.

Ex:-

```
import pandas as pd
s=pd.Series([10,20,30,40,50,60])
print(s.describe())
```

9. unique():-

the unique() function returns distinct values in the series.

Ex:-

```
import pandas as pd
s=pd.Series([10,10,10,20,30,40,20])
print(s.unique())
```

10. value_counts():-

the value_counts() function in pandas series is used to count the number of occurrences of each unique value in the series and display them in descending order.

Ex:-

```
import pandas as pd
s=pd.Series([10,20,30,40,50,60,10,10,40])
print(s.value_counts())
```

Ex:-

```
import pandas as pd
s=pd.Series(["apple","banana","apple","orange","banana"])
print(s.value_counts())
```

DataFrames in Pandas:-

A pandas dataframe is a two-dimensional table-like structure in python where data is arranged in rows and columns. it's one of the most commonly used tools for handling data and makes it easy to organize, analyse and manipulate data. it can store different types of data such as numbers, text and dates across its columns. the main parts of a dataframe are:

- Data: Actual values in the table.
- Rows: Labels that identify each row.
- Columns: labels that define each data category.

Ex:-

```
import pandas as pd
data=['ram','sita','ravi']
df=pd.DataFrame(data)
print(df)
```

Ex:-

```
import pandas as pd
data={
   "Name": ['Ram','sita','Ravi'],
   "Marks": [98,89,99]
}
```

```
df=pd.DataFrame(data)
print(df)

Ex:-
import pandas as pd
std_data=[
   (1,"varun",30,'male','tirupati'),
   (2, 'ravi',21,'male','kadapa'),
   (3,'sita',25,'female','madanapalle')
]
# df=pd.DataFrame(std_data,columns=['stu id','name','age','gender','location'])
df=pd.DataFrame(std_data, index=['Student 1', 'Student 2','student 3'])
print(df)
```

Index and Columns in Pandas DataFrame:-

A DataFrame in Pandas is like a table (similar to an Excel sheet).
This table has rows and columns.

Index → row numbers
Columns → column names (headings)

1. changing column names:-
we can change column names using the columns attribute.

Ex:-

```
df=pd.DataFrame(data, columns=['Name'])
```

2. changing index names
we can change index values using the index attribute.
Ex:-

```
df=pd.DataFrame(std_data, index=['Student 1', 'Student 2','student 3'])
```

Operations in Pandas DataFrame:-

Selecting Data (Rows and Columns):-

Ex:- (Column select)

```
import pandas as pd
data = {"Name":["Ram","Sita","Ravi"],
      "Marks":[85,90,80]}
df = pd.DataFrame(data["Marks"])
print(df)
```

Ex:- (multiple column select)

```
df[["Name","Marks"]]
```

Ex:-

```
import pandas as pd
data=[
   (1,'Ravi',23,'male','madanapalle'),
   (2,'Veera', 24, 'male', 'tirupati'),
   (3, 'sekar', 26, 'male', 'kadapa')
   ]
df=pd.DataFrame(data, columns=['Roll_no', 'Name', 'Age', 'Gender', 'Location'])
df=df['Name']
print(df)
```

Ex:-

```
import pandas as pd
data=[
   (1,"varun",24,'Male','Tirupati'),
   (2, 'Ravi', 21, 'Male', "Kadapa"),
   (3, 'Sita', 26, 'Female', 'Madanapalle')
   ]

df=pd.DataFrame(data,          columns=['Roll_no','Name','Age','Gender',          'Location'],
index=['Stu_1', 'Stu_2', 'Stu_3'])
print(df['Location'])
```

selecting Row:-

using loc (location):-

loc is a label-based indexing method used in a Pandas DataFrame to select rows and columns using their labels (names or index values).

Ex:-

```
import pandas as pd
data = {
   "ID": [101,102,103,104],
   "Name": ["Ram","Sita","Ravi","Anu"],
   "Marks": [85,90,80,88],
}
df = pd.DataFrame(data)
print(df.loc[0])
```

Ex:-

```
import pandas as pd
data=[
   (1,"varun",24,'Male','Tirupati'),
   (2, 'Ravi', 21, 'Male', "Kadapa"),
   (3, 'Sita', 26, 'Female', 'Madanapalle')
   ]


df=pd.DataFrame(data, columns=['Roll_no','Name','Age','Gender', 'Location'], index=['Stu_1', 'Stu_2', 'Stu_3'])
print(df.loc['Stu_1'])
```

Adding Column:-

```
import pandas as pd
data = {
   "ID": [101,102,103,104],
   "Name": ["Ram","Sita","Ravi","Anu"],
   "Marks": [85,90,80,88]
```

```
}
df = pd.DataFrame(data)
df["City"] = ["Hyderabad","Chennai","Delhi","Mumbai"]
print(df)
```

Ex:-

```
import pandas as pd
data=[
   (1,"varun",24,'Male','Tirupati'),
   (2, 'Ravi', 21, 'Male', "Kadapa"),
   (3, 'Sita', 26, 'Female', 'Madanapalle')
   ]

df=pd.DataFrame(data, columns=['Roll_no','Name','Age','Gender', 'Location'],
index=['Stu_1', 'Stu_2', 'Stu_3'])
df['University']=['SV University','YV University','SK University']
print(df)
```

Ex:-

```
import pandas as pd
data = {
   "ID": [101,102,103,104],
   "Name": ["Ram","Sita","Ravi","Anu"],
   "Marks": [85,90,80,88],
}
df = pd.DataFrame(data)
df.loc[3]=[105,"vishnu",94]
print(df)
```

Deleting Column:-

```
import pandas as pd
data = {
   "ID": [101,102,103,104],
```

```
   "Name": ["Ram","Sita","Ravi","Anu"],
   "Marks": [85,90,80,88],
}
df = pd.DataFrame(data)
del df["Marks"]
print(df)
```

Ex:-

```
import pandas as pd
data = {
   "ID": [101,102,103,104],
   "Name": ["Ram","Sita","Ravi","Anu"],
   "Marks": [85,90,80,88],
}
df = pd.DataFrame(data)
df=df.drop(columns=['Marks'])
print(df)
```

Deleting a row from Dataframe

```
import pandas as pd
data = {
   "ID": [101,102,103,104],
   "Name": ["Ram","Sita","Ravi","Anu"],
   "Marks": [85,90,80,88],
}
df = pd.DataFrame(data)
df=df.drop(2)
print(df)
```

Updating Column:-

```
import pandas as pd
data = {
   "ID": [101,102,103,104],
   "Name": ["Ram","Sita","Ravi","Anu"],
   "Marks": [85,90,80,88],
}
df = pd.DataFrame(data)
```

```
df.loc[2, 'ID']=10030
print(df)
```

Rename:-

```
import pandas as pd
data = {
   "ID": [101,102,103,104],
   "Name": ["Ram","Sita","Ravi","Anu"],
   "Marks": [85,90,80,88],
}
df = pd.DataFrame(data)
df=df.rename(columns={'Name':'Student_Name'})
print(df)
```

Filtering Data:-

Filtering data means selecting only the rows that satisfy a specific condition.
It helps us display only the required data from a DataFrame.

Ex:-

```
import pandas as pd
data = {
   "ID": [101,102,103,104],
   "Name": ["Ram","Sita","Ravi","Anu"],
   "Marks": [81,90,80,88],
}
df = pd.DataFrame(data)
print(df[df["Marks"] > 85])
```

Ex:- (Filtering Equal Values)

```
print(df[df["Marks"] == 90])
```

Sorting Data:-

Sorting means arranging the data in ascending (small → large) or descending (large → small) order based on a column.

In Pandas, sorting is done using the sort_values() method.

ex:-

```
import pandas as pd
data = {
   "ID": [101,102,103,104],
   "Name": ["Ram","Sita","Ravi","Anu"],
   "Marks": [81,90,80,88],
}
df = pd.DataFrame(data)
print(df.sort_values("Marks"))
```

Ex:-(Sorting in Descending Order)

```
print(df.sort_values("Marks", ascending=False))
```

Statistical Operations in DataFrame:-

Statistical operations in pandas dataframe are used to analyse numerical data such as finding sum, average, minimum, maximum, count, etc.
these operations help in data analysis and summarizing data.

Ex:-

```
import pandas as pd
data={
   'maths' :[85,96,47,83],
   'science' : [88,96,56,99],
   'English' : [75,85,62,92]
}
df=pd.DataFrame(data)
print(df)
```

1.sum:-

used to calculate total values.
ex:-

```
print(df.sum())
```

2. Mean (Average):-

used to calculate the average value.
ex:-

```
print(df.mean())
```

3. Minimum value:-

finds the smallest value

Ex:-

```
print(df.min())
```

4. Maximum value:-

finds the largest value

Ex:-

```
print(df.max())
```

5. count

counts the number of values.

Ex:-

```
print(df.count())
```

6.Standard deviation:-

shows variation in data.

Ex:-

```
 print(df.std())
```

7. Describe Function:-

displays all statistical information together.

Ex:-

```
print(df.describe())
```

Complete Example:-

```
import pandas as pd
data={
   'maths' :[85,96,47,83],
   'science' : [88,96,56,99],
   'English' : [75,85,62,92]
}
df=pd.DataFrame(data)
print(df)
print(df.sum())
print(df.mean())
```

```
print(df.min())
print(df.max())
print(df.count())
print(df.std())
print(df.describe())
```

CSV (Comma Separated Values)

CSV stands for Comma Separated Values. It is a simple file format used to store tabular data such as rows and columns. Each line in a CSV file represents a row, and the values in that row are separated by commas (,).

CSV files are widely used in data storage, data exchange, spreadsheets, and databases because they are simple and supported by many programs like Excel, Python, MySQL, and Google Sheets.

in python, the pandas library is commonly used to work with CSV files. Pandas provides powerful data structures called DataFrames, which store data in rows and columns similar to a spreadsheet or database table. using pandas, we can easily import data from CSV files into DataFrame and Export dataframe back to CSV files for storage or sharing.

1. importing Data from CSV to DataFrame :- (Reading CSV file)

importing means reading data from a CSV file and loading it into a pandas DataFrame. this allows users to analyse, modify, and manipulate the data easily. the function used for importing CSV data is read_csv().

syntax:-

```
import pandas as pd
df=pd.read_csv('filename.csv')
```

Example:-

suppose we have a CSV file named students.csv

```
ID, Name, Marks
1, 'Ravi', 87
2, 'Sita', 90
3, 'Rahul', 78
```

Program:-

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

here, the CSV data is converted into DataFrame , where pandas automatically creates an index column.

2. Exporting Data from DataFrame to CSV :- (writing CSV file)

exporting means saving the dataframe data into a CSV file. this is useful for storing processed data or sharing it with other applications such as Excel.
the function used for exporting data is to_csv().

syntax:-

```
df.to_csv("filename.csv")
```

Example:-

```
import pandas as pd
data={
   "subject":['python', 'HTML', 'CSS'],
   "marks" : [85,96,89]
}
df=pd.DataFrame(data)
df.to_csv('a123.csv')
print(df)
```

Advantages of using CSV with pandas:-

1. CSV files are simple and easy to use.
2. they are supported by many application like Excel, database, and programming language.
3. pandas allows fast reading and writing of large datasets.
4. dataframes provide powerful data analysis and manipulation features.

viewing Data in CSV file :-

pandas provide some functions to see the data.

view first 5 rows:-

df.head()

view last 5 rows:-

df.tail()

display column names

df.columns

display basic information

df.info()

IMP Questions:-

1. Explain Numpy array and its types in Python and Explain Operations of numpy?
2. Explain in detail about Reshaping array ? (5M & 10 M)
3. how to import and export CSV files to data frames.
4. what is pandas ? explain series and dataframe structures in detail with examples.
5. Explain the various data handling operations available in pandas. Discuss data selection, filtering and summary functions.
6. Explain Arithmetic operations on numpy arrays with examples. (5M)
7. write a short note on statistical functions in numpy. (5M)

B. Devendra Msc-CS
Dept of Computers
Shri Gnanambica Degree College(A)