NumPy Introduction

- NumPy stands for Numerical Python, is an open-source Python library that provides support for large, multi-dimensional arrays and matrices.
- It also has a collection of high-level mathematical functions to operate on arrays. It was created by Travis Oliphant in 2005.
- It provides a high-performance multidimensional array object and tools for working with these arrays.

Features of NumPy

- A powerful N-dimensional array object
- Sophisticated (broadcasting) functions
- Tools for integrating C/C++ and Fortran code
- Useful linear algebra, Fourier transform, and random number capabilities.

Install Python NumPy

Step 1: open cmd
Step 2: run command pip install numpy

Install NumPy in Your PyCharm Project Environment:

Steps:

1. Open PyCharm and load your project.
2. Access the Python Interpreter:

    - Go to File → Settings → Project: [Your Project Name] → Python Interpreter.
3. Check Existing Packages:
    - Under the Python Interpreter settings, you'll see a list of installed packages.
4. Install NumPy:
    - Click the + (Add) icon.
    - Search for numpy in the search bar.
    - Select NumPy from the list and click Install Package.

Verify Installation:

```
import numpy
print(numpy.__version__)
```

NumPy Array

NumPy array is a powerful N-dimensional array object and is used in linear algebra, Fourier transform, and random number capabilities.
It provides an array object much faster than traditional Python lists.

Types of Array:

1. One Dimensional Array
2. Two Dimensional Array
3. Multi-Dimensional Array

Example:

```
import numpy as np
```

```
l1=np.array([1,2,3,4])
print(l1)

l2=np.array([[2,3,4],[5,6,7]])
print(l2)

l3 = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
print(l3)
```

```
#inspect the array
import numpy as np
l1=[[1,2],[3,4],[5,6]]
arr1=np.array(l1)
print(arr1)
print(arr1.shape)#return the shape of the array
print(len(arr1))#return  array length
print(np.size(arr1))#return no of elements present inside the your array
print(type(arr1))#return the type od array
print(arr1.dtype)#return the type of data present in array
print(arr1.astype(str))# type coversion operation
```

NumPy Array Copy vs View

- The main difference between a copy and a view of an array is that the copy is a new array, and the view is just a view of the original array.
- The copy *owns* the data and any changes made to the copy will not affect the original array, and any changes made to the original array will not affect the copy.
- The view *does not own* the data and any changes made to the view will affect the original array, and any changes made to the original array will affect the view.

Copy Example:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])
x = arr.copy()
arr[0] = 42

print(arr)
print(x)
```

View Example:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])
x = arr.view()
arr[0] = 42

print(arr)
print(x)
```

Reshaping arrays

- Reshaping means changing the shape of an array.
- The shape of an array is the number of elements in each dimension.
- By reshaping we can add or remove dimensions or change number of elements in each dimension.

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])

newarr = arr.reshape(4, 3)

print(newarr)
```

NumPy Array Iterating

Iterating means going through elements one by one.

As we deal with multi-dimensional arrays in numpy, we can do this using basic for loop of python.

Example of iterating 1-D array:

```
import numpy as np

arr = np.array([1, 2, 3])

for x in arr:

  print(x)
```

Iterating Arrays Using nditer()

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

```
import numpy as np

arr = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])

for x in np.nditer(arr):

 print(x)
```

Enumerated Iteration Using ndenumerate()

Enumeration means mentioning sequence number of somethings one by one.

```
import numpy as np

arr = np.array([1, 2, 3])

for idx, x in np.ndenumerate(arr):

  print(idx, x)
```

Array Creation Techniques

a. arange()

- Creates arrays with evenly spaced values within a given range.
- Syntax: numpy.arange(start, stop, step)

b. zeros()

- Creates an array filled with zeros.
- Syntax: numpy.zeros(shape, dtype)

c. ones()

- Creates an array filled with ones.
- Syntax: numpy.ones(shape, dtype)

d. linspace()

- Generates evenly spaced numbers over a specified interval.
- Syntax: numpy.linspace(start, stop, num)
-

e. full()

- Creates an array filled with a specified value.
- Syntax: numpy.full(shape, fill_value)

NumPy Joining Array

Joining means putting contents of two or more arrays in a single array.

In SQL we join tables based on a key, whereas in NumPy we join arrays by axes.

```
import numpy as np

arr1 = np.array([1, 2, 3])
```

```
arr2 = np.array([4, 5, 6])

arr = np.concatenate((arr1, arr2))

print(arr)
```

np.vstack() and np.hstack(): Stack arrays vertically and horizontally.

```
import numpy as np
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])

# Vertical stacking (along rows)
vstacked = np.vstack((arr1, arr2))
print("Vertical Stack:\n", vstacked)

# Horizontal stacking (along columns)
hstacked = np.hstack((arr1, arr2))
print("Horizontal Stack:", hstacked)
```

np.dstack(): Stack arrays in the third dimension.

```
arr1 = np.array([1, 2, 3])
```

```
arr2 = np.array([4, 5, 6])


# Stack along the third dimension (depth)
dstacked = np.dstack((arr1, arr2))
print("Depth Stack:\n", dstacked)
```

Array Manipulation Functions:

1. np.split() - Splits an array into multiple sub-arrays.
2. np.insert() - Insert elements into an array.
3. np.delete() - Delete elements from an array.
4. np.repeat() - Repeat elements of an array.

Array Operations in NumPy

a) Arithmetic Operations:

addition(+)
subtraction(-)
Multiplication(*)
Division(/)
Floor division(//)
Exponential(**)

b) Logical Operations:

Logical AND (np.logical_and)
Logical OR (np.logical_or)
Logical NOT (np.logical_not)

c) Comparison Operations
Greater Than (>)
Equality (==)
Less Than or Equal (<=)

D)Broadcasting in array
Allows operations between arrays of different shapes by "stretching" the smaller array.

Example:

```
a = np.array([1, 2, 3])
b = 2
result = a * b
print(result)
```

E) statistical operations:

1. Mean

The mean is the average of the elements in the array, calculated by summing all elements and dividing by the total number of elements.

- Syntax: np.mean(arr, axis=None)

2. Median

The median is the middle value of the array when the values are sorted. If the number of elements is even, the median is the average of the two middle elements.

- Syntax: np.median(arr, axis=None)

3.Min and Max

The min and max functions find the minimum and maximum values in the array, respectively.

- Syntax:
    - np.min(arr)
    - np.max(arr)

4. Percentiles

The percentile is used to compute the value below which a given percentage of observations fall. For example, the 50th percentile is the median.

- Syntax: np.percentile(arr, q)

5. Count Non-Zero Elements

The non-zero elements function counts the number of non-zero elements in the array.

- Syntax: np.count_nonzero(arr)

6. All and Any

These functions are used for logical comparisons over arrays:

- np.all(arr): Returns True if all elements are true.
- np.any(arr): Returns True if any element is true.

7 . argmin() and argmax()

- Returns the index of the smallest and largest elements.

Sorting Arrays

1. sort(): Sorts the elements of an array (default: ascending order).
2. np.argsort(): Returns the indices that would sort an array.
3. np.partition(): Partially sorts the array (useful for finding k smallest/largest elements).

Random Functions:

1. np.random.rand()
- Purpose: Generates random float values between 0 and 1

2. np.random.randint()
- Purpose: Generates random integers between a specified low (inclusive) and high (exclusive) value.

3. np.random.choice()
- Purpose: Randomly selects elements from a given array, with or without replacement.

Function	Purpose	Example
`np.random.rand()`	Random floats in [0, 1)	`np.random.rand(3, 2)`
`np.random.randn()`	Standard normal distribution (mean=0, std=1)	`np.random.randn(3, 2)`
`np.random.randint()`	Random integers within a range	`np.random.randint(1, 10, size=(3, 2))`
`np.random.choice()`	Randomly selects from a given array	`np.random.choice([1, 2, 3], size=2)`
`np.random.uniform()`	Random floats in a custom range	`np.random.uniform(5, 10, size=(3, 2))`
`np.random.normal()`	Random values from a normal distribution	`np.random.normal(0, 1, size=(3, 2))`
`np.random.bytes()`	Generates random bytes	`np.random.bytes(10)`
`np.random.seed()`	Sets the seed for reproducibility	`np.random.seed(42)`
`np.random.permutation()`	Random permutation of a sequence	`np.random.permutation([1, 2, 3])`
`np.random.shuffle()`	Shuffle an array in-place	`np.random.shuffle(arr)`

Common Functions

a. reshape()

- Changes the shape of an array without altering the data.
- Syntax: numpy.reshape(array, new_shape)

b. ravel()

- Flattens a multi-dimensional array into a 1D array.
- Syntax: numpy.ravel(array)

c. transpose()

- Swaps the rows and columns of a multi-dimensional array.
- Syntax: numpy.transpose(array)

D. np.flatten()

- Purpose: Flattens a multi-dimensional array into a 1D array.
- Example:

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

flattened_arr = arr.flatten()

print(flattened_arr)
```

E. np.unique()

- Purpose: Finds the unique elements in an array.
- Example:

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

unique_elements = np.unique(arr)

print(unique_elements)
```

F. np.dot() and np.matmul()

- Purpose: Performs matrix multiplication. Both can be used for dot product, but np.matmul() is more flexible.
- Example:

```
arr1 = np.array([[1, 2], [3, 4]])

arr2 = np.array([[5, 6], [7, 8]])

print(np.dot(arr1, arr2))   # Dot product

print(np.matmul(arr1, arr2))  # Matrix multiplication
```

G. np.where()

- Purpose: Returns elements chosen from two arrays based on condition.
- Example:

  arr = np.array([1, 2, 3, 4])

  result = np.where(arr % 2 == 0, "Even", "Odd")

  print(result)

Reading and writing arrays to files

Writing Arrays to Files

- np.save(): Saves an array to a binary .npy file (NumPy's standard binary format).
  - Syntax: np.save(file, arr)
  - Parameters:
    - file: The name of the file where the array is saved (it can be a string or file object).
    - arr: The NumPy array to be saved.

  Example:

  import numpy as np

  arr = np.array([1, 2, 3, 4, 5])

  np.save('array_data.npy', arr)

2. np.savetxt(): Saves an array to a text file (e.g., .txt or .csv).

- Syntax: np.savetxt(fname, arr, delimiter=' ', fmt='%s')
- Parameters:
    - fname: The name of the file to save the array to.
    - arr: The NumPy array to be saved.
    - delimiter: String to separate values in the file (defaults to space).
    - fmt: Format for the numbers (e.g., %d, %f for integer and float formats).

Example:

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

np.savetxt('array_data.txt', arr, delimiter=',')
```

Reading Arrays from Files

1. np.load(): Loads an array from a binary .npy file.
    - Syntax: np.load(file)
    - Parameters:
        - file: The name of the file from which to load the array.

    Example:

    ```
    loaded_arr = np.load('array_data.npy')  # Load array from binary file

    print(loaded_arr)
    ```

2. np.loadtxt(): Loads an array from a text file (e.g., .txt or .csv).

- Syntax: np.loadtxt(fname, delimiter=' ', dtype=float)
- Parameters:
    - fname: The name of the file to load the array from.
    - delimiter: The string separating values in the file.

- dtype: The data type to convert the array into (e.g., float, int).

Example:

loaded_arr_txt = np.loadtxt('array_data.txt', delimiter=',')

print(loaded_arr_txt)

Saving Multiple Arrays

1. np.savez(): Saves multiple arrays into a .npz (zipped) file. This can be used to save several arrays in one compressed file.
   - Syntax: np.savez(file, array1=arr1, array2=arr2, ...)

Example:

arr1 = np.array([1, 2, 3])

arr2 = np.array([4, 5, 6])

np.savez('arrays_data.npz', array1=arr1, array2=arr2)

np.load(): Loads multiple arrays from a .npz file.

- Syntax: np.load(file)
- Example:

  data = np.load('arrays_data.npz')

  arr1 = data['array1']

  arr2 = data['array2']

  print(arr1, arr2)