1. NumPy

1.1. Introduction

It stands for Numerical Python and it is a python library used for working with arrays. NumPy is a Python library and is written partially in Python, but most of the parts that require fast computation are written in C or C++.

2. Functionality

- It can work in domain of linear algebra, fourier transform, and matrices.
- It is an open source project and can be used freely.
- NumPy provide an array object that is faster than Python lists.
- Arrays are very frequently used in data science, where speed and resources are very important.
- NumPy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently.
- It is optimized to work with latest CPU architectures.

2.1. Data Science

It is a branch of computer science where we study how to store, use and analyze data for deriving information from it.

3. Installation

```
>pip install numpy
```

4. Working

Now Numpy is imported and ready to use.

```
import numpy

ar = numpy.array([1, 2, 3, 4, 5])

print(ar)
```

5. NumPy as np

NumPy is usually imported under the np alias. In Python alias are an alternate name for referring to the same thing.

- **Example** :

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

- To create an ndarray, we can pass a list, tuple or any array-like object into the array() method, and it will be converted into an ndarray

- **Example** :
  a tuple to create a NumPy array

```
import numpy as np
arr = np.array((1, 2, 3, 4, 5))
print(arr)
```

6. Array Dimensions

6.1. 0-D Arrays

0-D arrays, or Scalars, are the elements in an array. Each value in an array is a 0-D array.

- **Example** :
  Create a 0-D array with value 14

```
import numpy as np
arr = np.array(42)
print(arr)
```

6.2. 1-D Arrays

An array that has 0-D arrays as its elements is called uni-dimensional or 1-D array.

- **Example** :
  Create a 1-D array containing the values 1,2,3,4,5

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

6.3. 2-D Arrays

An array that has 1-D arrays as its elements is called a 2-D array. These are often used to represent matrix or 2nd order tensors.

NumPy has a whole sub module dedicated towards matrix operations called numpy.mat

- **Example** :
  Create a 2-D array containing two arrays with the values 1,2,3 and 4,5,6

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

6.4. 3-D arrays

An array that has 2-D arrays (matrices) as its elements is called 3-D array.

These are often used to represent a 3rd order tensor.

- **Example** :
  Create a 3-D array with two 2-D arrays, both containing two arrays with the values 1,2,3 and 4,5,6

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

7. ndim

NumPy Arrays provides the ndim attribute that returns an integer that tells us how many dimensions the array have.

- **Example** :
  Check how many dimensions the arrays have

```
import numpy as np

a = np.array(42)
b = np.array([1, 2, 3, 4, 5])
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(a.ndim)
print(b.ndim)
print(c.ndim)
print(d.ndim)
```

7.1. ndim Functionality

When the array is created, you can define the number of dimensions by using the ndmin argument.

- **Example** :
  Create an array with 5 dimensions and verify that it has 5 dimensions

```
import numpy as np

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

print(arr)
print('number of dimensions :', arr.ndim)
```

In this array the innermost dimension (5th dim) has 4 elements, the 4th dim has 1 element that is the vector, the 3rd dim has 1 element that is the matrix with the vector, the 2nd dim has 1 element that is 3D array and 1st dim has 1 element that is a 4D array.

8. NumPy Array Indexing

You can access an array element by referring to its index number. The indexes in NumPy arrays start with 0, meaning that the first element has index 0, and the second has index 1 etc.

- **Example** :
  Get the first element from the following array

```
import numpy as np

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

print(arr[0])
```

8.1. Access 2-D Arrays

To access elements from 2-D arrays we can use comma separated integers representing the dimension and the index of the element.

- **Example** :
  Access the 2nd element on 1st dim

```
import numpy as np

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

print('2nd element on 1st dim: ', arr[0, 1])
```

8.2. Access 3-D Arrays

To access elements from 3-D arrays we can use comma separated integers representing the dimensions and the index of the element.

- Example :
  Access the third element of the second array of the first array

```
import numpy as np

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

print(arr[0, 1, 2])
```

8.3. Negative Indexing

Use negative indexing to access an array from the end.

- **Example** :
  Print the last element from the 2nd dim

```
import numpy as np

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

print('Last element from 2nd dim: ', arr[1, -1])
```

9. NumPy Array Slicing

Slicing in python means taking elements from one given index to another given index.

We pass slice instead of index like this: [start:end].

We can also define the step, like this: [start:end:step].

- **Example** :
  Slice elements from index 1 to index 5 from the following array

```
import numpy as np

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

print(arr[1:5])
```

> **Note** : The result includes the start index, but excludes the end index.

9.1. Negative Slicing

Use the minus operator to refer to an index from the end

- **Example** :
  Slice from the index 3 from the end to index 1 from the end:

```
import numpy as np

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

print(arr[-3:-1])
```

9.2. STEP

Use the step value to determine the step of the slicing

- **Example** :
  Return every other element from index 1 to index 5

```
import numpy as np

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

print(arr[1:5:2])
```

9.3. Slicing 2-D Arrays

- **Example** :
  From the second element, slice elements from index 1 to index 4 (not included)

```
import numpy as np

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

print(arr[1, 1:4])
```

Note : Remember that second element has index 1.

10. NumPy Data Types

Below is a list of all data types in NumPy and the characters used to represent them.

i - integer
b - boolean
u - unsigned integer
f - float
c - complex float
m - timedelta
M - datetime
O - object
S - string
U - unicode string
V - fixed chunk of memory for other type (void)

10.1. Checking the Data Type of an Array

The NumPy array object has a property called dtype that returns the data type of the array

- **Example** :
  Get the data type of an array object

```
import numpy as np

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

print(arr.dtype)
```

10.2. Creating Arrays With a Defined Data Type

*We use the array() function to create arrays, this function can take an optional argument: dtype that allows us to define the expected data type of the array elements. For i, u, f, S and U we can define size as well.
*

- **Example** :
  Create an array with data type string

```
import numpy as np

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

print(arr)
print(arr.dtype)
```

10.3. Converting Data Type on Existing Arrays

The best way to change the data type of an existing array, is to make a copy of the array with the astype() method.

The astype() function creates a copy of the array, and allows you to specify the data type as a parameter.

- **Example** :
  Change data type from float to integer by using 'i' as parameter value

```
import numpy as np

arr = np.array([1.1, 2.1, 3.1])

newarr = arr.astype('i')

print(newarr)
print(newarr.dtype)
```

11. 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 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.*

11.1. COPY

The copy SHOULD NOT be affected by the changes made to the original array.

- Example :
  Make a copy, change the original array, and display both arrays

```
import numpy as np

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

print(arr)
print(x)
```

11.2. VIEW

The view SHOULD be affected by the changes made to the original array.

- **Example** :
  Make a view, change the original array, and display both arrays

```
import numpy as np

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

print(arr)
print(x)
```

11.2.1. Make Changes in the VIEW

The original array SHOULD be affected by the changes made to the view.

- **Example** :
  Make a view, change the view, and display both arrays

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])
x = arr.view()
x[0] = 31

print(arr)
print(x)
```

11.3. Check if Array Owns it's Data

Every NumPy array has the attribute base that returns None if the array owns the data. Otherwise, the base attribute refers to the original object.

- The copy returns None.
- The view returns the original array
- **Example** :
  Print the value of the base attribute to check if an array owns it's data or not

```
import numpy as np

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

x = arr.copy()
y = arr.view()

print(x.base)
print(y.base)
```

12. NumPy Array Shape

The shape of an array is the number of elements in each dimension.NumPy arrays have an attribute called shape that returns a tuple with each index having the number of corresponding elements.

- **Example** :
  Print the shape of a 2-D array

```
import numpy as np

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

print(arr.shape)
```

> The example above returns (2, 4), which means that the array has 2 dimensions, and each dimension has 4 elements.

13. NumPy Array Reshaping

By reshaping we can add or remove dimensions or change number of elements in each dimension.

- **Example** :
  Convert the following 1-D array with 12 elements into a 2-D array.

The outermost dimension will have 4 arrays, each with 3 elements

```
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)
```

- **Example** :
  Convert the array into a 1D array

```
import numpy as np

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

newarr = arr.reshape(-1)

print(newarr)
```

14. NumPy Array Iterating

In basic for loops, iterating through each scalar of an array we need to use n for loops which can be difficult to write for arrays with very high dimensionality.

- Example :
  Iterate through the following 3-D array

```
import numpy as np

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

for x in np.nditer(arr):
  print(x)
```

15. NumPy Joining Array

Joining means putting contents of two or more arrays in a single array. We pass a sequence of arrays that we want to join to the concatenate() function, along with the axis. If axis is not explicitly passed, it is taken as 0.

- **Example** :
  Join two arrays

```
import numpy as np

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

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

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

print(arr)
```

- **Example** :
  Join two 2-D arrays along rows (axis=1)

```
import numpy as np

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

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

arr = np.concatenate((arr1, arr2), axis=1)

print(arr)
```

15.1. Joining Arrays Using Stack Functions

Stacking is same as concatenation, the only difference is that stacking is done along a new axis.

- **Example** :

```
import numpy as np

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

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

arr = np.stack((arr1, arr2), axis=1)

print(arr)
```

15.1.1. Stacking Along Rows

NumPy provides a helper function: hstack() to stack along rows.

- **Example** :

```
import numpy as np

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

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

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

print(arr)
```

15.1.2. Stacking Along columns

NumPy provides a helper function: vstack() to stack along columns.

- **Example** :

```
import numpy as np

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

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

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

print(arr)
```

15.1.3. Stacking Along Height (depth)

NumPy provides a helper function: dstack() to stack along height, which is the same as depth.

- **Example** :

```
import numpy as np

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

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

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

print(arr)
```

16. NumPy Splitting Array

We use array_split() for splitting arrays, we pass it the array we want to split and the number of splits.

- Example :
  Split the array in 3 parts

```
import numpy as np

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

newarr = np.array_split(arr, 3)

print(newarr)
```

> If the array has less elements than required, it will adjust from the end accordingly.

- **Example** :
  Split the array in 4 parts

```
import numpy as np

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

newarr = np.array_split(arr, 4)

print(newarr)
```

17. NumPy Searching Arrays

To search an array, use the where() method.

- **Example** :
  Find the indexes where the value is 4

```
import numpy as np

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

x = np.where(arr == 4)

print(x)
```

The example above will return a tuple: (array([3, 5, 6],)
Which means that the value 4 is present at index 3, 5, and 6.

- **Example** :
  Find the indexes where the values are even

```
import numpy as np

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

x = np.where(arr%2 == 0)

print(x)
```

- **Example** :
  Find the indexes where the values are odd

```
import numpy as np

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

x = np.where(arr%2 == 1)

print(x)
```

17.1. Search Sorted

There is a method called searchsorted() which performs a binary search in the array, and returns the index where the specified value would be inserted to maintain the search order.

The searchsorted() method is assumed to be used on sorted arrays.

- **Example** :
  Find the indexes where the value 7 should be inserted:

```
import numpy as np

arr = np.array([6, 7, 8, 9])

x = np.searchsorted(arr, 7)

print(x)
```

EXPLANATION: The number 7 should be inserted on index 1 to remain the sort order.
The method starts the search from the left and returns the first index where the number 7 is no longer larger than the next value.

17.2. Search From the Right Side

By default the left most index is returned, but we can give side='right' to return the right most index instead.

- **Example** :
  Find the indexes where the value 7 should be inserted, starting from the right

```
import numpy as np

arr = np.array([6, 7, 8, 9])

x = np.searchsorted(arr, 7, side='right')

print(x)
```

17.3. Multiple Values

To search for more than one value, use an array with the specified values.

- **Example** :
  Find the indexes where the values 2, 4, and 6 should be inserted

```
import numpy as np

arr = np.array([1, 3, 5, 7])

x = np.searchsorted(arr, [2, 4, 6])

print(x)
```

The return value is an array: [1 2 3] containing the three indexes where 2, 4, 6 would be inserted in the original array to maintain the order.

18. NumPy Sorting Arrays

Sorting means putting elements in a ordered sequence. Ordered sequence is any sequence that has an order corresponding to elements, like numeric or alphabetical, ascending or descending.

The NumPy ndarray object has a function called sort(), that will sort a specified array.

- **Example** :
  Sort the array

```
import numpy as np

arr = np.array([3, 2, 0, 1])

print(np.sort(arr))
```

19. NumPy Filter Array

Getting some elements out of an existing array and creating a new array out of them is called filtering.

- **Example** :
  Create an array from the elements on index 0 and 2

```
import numpy as np

arr = np.array([41, 42, 43, 44])

x = [True, False, True, False]

newarr = arr[x]

print(newarr)
```

19.1. Creating the Filter Array

In the example above we hard-coded the True and False values, but the common use is to create a filter array based on conditions.

- **Example** :
  Create a filter array that will return only values higher than 42

```
import numpy as np

arr = np.array([41, 42, 43, 44])

#Create an empty list
filter_arr = []

# go through each element in arr
for element in arr:
  # if the element is higher than 42, set the value to True, otherwise False:
  if element > 42:
    filter_arr.append(True)
  else:
    filter_arr.append(False)

newarr = arr[filter_arr]

print(filter_arr)
print(newarr)
```

- **Example** :
  Create a filter array that will return only even elements from the original array

```
import numpy as np

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

#Create an empty list
filter_arr = []

#Go through each element in arr
for element in arr:
  # if the element is completely divisble by 2, set the value to True, otherwise False
  if element % 2 == 0:
    filter_arr.append(True)
  else:
    filter_arr.append(False)

newarr = arr[filter_arr]

print(filter_arr)
print(newarr)
```

19.2. Creating Filter Directly From Array

We can directly substitute the array instead of the iterable variable in our condition and it will work just as we expect it to.

- Example :
  Create a filter array that will return only values higher than 42

```
import numpy as np

arr = np.array([41, 42, 43, 44])

filter_arr = arr > 42

newarr = arr[filter_arr]

print(filter_arr)
print(newarr)
```

- **Example** :
  Create a filter array that will return only even elements from the original array

```
import numpy as np

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

filter_arr = arr % 2 == 0

newarr = arr[filter_arr]
```

```
print(filter_arr)
print(newarr)
```

20. NumPy ufuncs

ufuncs stands for "Universal Functions" and they are NumPy functions that operates on the ndarray object.

20.1. Use

ufuncs are used to implement vectorization in NumPy which is way faster than iterating over elements.They also provide broadcasting and additional methods like reduce, accumulate etc. that are very helpful for computation. ufuncs also take additional arguments, like:

- where : boolean array or condition defining where the operations should take place.
- dtype : defining the return type of elements.
- out : output array where the return value should be copied.

20.2. Vectorization

Converting iterative statements into a vector based operation is called vectorization.

It is faster as modern CPUs are optimized for such operations.

- Add the Elements of Two Lists :
  list 1: [1, 2, 3, 4]

list 2: [4, 5, 6, 7]

> One way of doing it is to iterate over both of the lists and then sum each elements.

- Example :
  Without ufunc, we can use Python's built-in zip() method

```
x = [1, 2, 3, 4]
y = [4, 5, 6, 7]
z = []

for i, j in zip(x, y):
  z.append(i + j)
print(z)
```

> NumPy has a ufunc for this, called add(x, y) that will produce the same result.

- Example :
  With ufunc, we can use the add() function

```
import numpy as np

x = [1, 2, 3, 4]
y = [4, 5, 6, 7]
```

```
z = np.add(x, y)

print(z)
```