Chapter 2: Numpy

- NumPy (Numerical Python) is a Python library used for working with arrays
- NumPy aims to provide an array that is up to 50x faster.
- Arrays are very frequently used in data science.

Data Science: is a branch of computer science where we study how to store, use and analyze data.

1. Installation of NumPy

```
pip install numpy
```

2. Import NumPy

You can refer as np instead of numpy: `import numpy as np`

3. Create a NumPy array Object

NumPy is used to work with arrays.

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

4. Dimensions in Arrays

a) 1-D Array: uni-dimensional:

1-D array containing the values 1, 2, 3, 4, 5:

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

b) 2-D Arrays: matrix:

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)
print(arr)
print(type(arr))
```

c) Number of dimensions

Check how many dimensions the arrays have:

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

5. NumPy Array Indexing

a) Access Array Elements

The indexes in NumPy arrays start with 0.

```
import numpy as np
arr = np.array([1, 2, 3, 4])
print(arr[0])   # First element = 1
print(arr[1])   # Second element = 2
```

b) Access 2-D Arrays

The dimension represents the row and the index represents the column

```
import numpy as np
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print('2nd element on 1st row: ', arr[0, 1])
```

Access the element on the 2nd row, 5th column:

```
import numpy as np
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print('5th element on 2nd row: ', arr[1, 4])
```

Print the last element from the 2nd dim: negative indexing to access an array from the end

```
import numpy as np
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print('Last element from 2nd row: ', arr[1, -1])
```

6. NumPy Array Slicing

a) Slicing arrays

Slicing in python means taking elements from one given index to another given index. *[start:end[* or *[start:end:step[.*

- If we don't pass start, it’s considered 0
- If we don't pass end it’s considered length of array
- If we don't pass step it’s considered 1

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.

Slice elements from index 4 to the end of the array:

```
import numpy as np

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

print(arr[4:])
```

Slice elements from the beginning to index 4 (not included):

```
import numpy as np

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

print(arr[:4])
```

b) Negative Slicing

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

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])
```

STEP: to determine the step of the slicing:

Return every other element from index 1 to index 6:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[1:6:2])
```

Return every other element from the entire array:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[::2])
```

c) Slicing 2-D Arrays

From the second row, slice elements from index 1 to index 4 (excluded):

```
import numpy as np
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print(arr[1, 1:4])
```

From both rows, return index 2:

```
import numpy as np
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print(arr[0:2, 2])
```

From both rows, slice index 1 to index 4 (excluded and return 2-D array:

```
import numpy as np
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print(arr[0:2, 1:4])
```

7. NumPy Array Methods

a) The difference between Copy and View

The main difference is that the copy is a new array, and the view is just a view of the original array.

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

```
import numpy as np
arr1 = np.array([1, 2, 3, 4, 5])
arr2 = arr1.copy()
arr1[0] = 42
print(arr1)
print(arr2)
```

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

```
import numpy as np
arr1 = np.array([1, 2, 3, 4, 5])
arr2 = arr1.view()
arr1[0] = 42
print(arr1)
print(arr2)
```

b) Make Changes in the VIEW:

Make a view, change the view, and display both arrays:

```
import numpy as np
arr1 = np.array([1, 2, 3, 4, 5])
arr2 = arr1.view()
arr2[0] = 31
print(arr1)
print(arr2)
```

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

c) Shape of an Array

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.

```
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 each one has 4 elements.

d) Reshape from 1-D to 2-D

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

```
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)
```

The dimension will have 4 arrays, each with 3 elements

e) Flattening the arrays

Convert the array into a 1D array: reshape(-1) or flatten

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

f) Iterating Arrays

Iterating means going through elements one by one.

```
import numpy as np
arr = np.array([1, 2, 3])
for x in arr: print(x)
```

g) Iterating 2-D Arrays

Iterate on each scalar element of the 2-D array:

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

h) Joining NumPy Arrays

Joining means putting contents of two or more arrays in a single array using concatenate() or stack() function.

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)
```

i) Stacking along Rows: NumPy provides a helper function: hstack()

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

j) Stacking Along Columns: NumPy provides a helper function: vstack()

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

k) Stacking along Height (depth): NumPy provides a helper function: dstack()

```
import numpy as np

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

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

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

print(arr)
```

l) Searching Arrays

You can search an array for a certain value, and return the indexes that get a match using the where() method.

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.

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)
```

m)Sorting Arrays

Sorting means putting elements in an *ordered sequence* ascending using a function called sort().

```
import numpy as np

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

print(np.sort(arr))  #ascending

print(-np.sort(-arr)) #descending
```

Note: This method returns a copy of the array, leaving the original array unchanged.

n) Create an array of all zeros

```
import numpy as np
arr = np.zeros([3, 3])
print(arr)
```

o) Create an array of all ones

```
import numpy as np
arr = np.ones([2, 3])  #2 rows, 3 columns
print(arr)
```

p) Create constant array

```
import numpy as np
arr = np.full([3, 3], 5)
print(arr)
```

q) Create an array filled with random values

```
import numpy as np
arr = np.random.random([3, 3])
print(arr)
```

Example

```
import numpy as np
arr = np.array([
                    [
                      [1, 2, 3], [4, 5, 6]
                    ],[
                      [7, 8, 9], [2, 2, 2]
                    ]
                ])
print(arr.shape)
print(arr[0,1,0])
```