Data Engineering Notes

# NumPy

NumPy means “Numeric Python” and it’s a package of python. It is used for manipulating data and calculating results.

## Array

A type of data structure that stores elements and refer to their values using integer indices.

### N-D Array

N-D array means N Dimensional Array. It’s similar to python lists but faster because they work elementwise.

* 0-D array means a single data point.
* 1-D array means a sequence of data like data in a row or a column
* 2-D array means a grid or table of elements arranged in rows and columns. And actually 2-D arrays are 2 array of 1-D dimension put upon one another.
* If there is one square bracket in the code so it means its 1-D array and if there are two square brackets so it means that its 2-D array and so on.

## Why we use NumPy?

* We use NumPy because its computationally stable and efficient.
* It works in lower-level language like C and take short time for computations.
* Can compute a lot of values
* Deals with vectors and matrices
* It can import and manipulate data

## Difference between List and Array:

* Lists are printed in one line and arrays are printed like a table with rows and columns.
* Lists have lengths and arrays have shape.
* Arrays work elementwise while lists operate as a collection or sequence of elements. For example:

import numpy as np

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

result = arr + 2 # Adds 2 to each element

print(result) # Output: [3, 4, 5]

while list will have the following outcome:

lst = [1, 2, 3]

result = lst + [2] # Concatenates the two lists

print(result) # Output: [1, 2, 3, 2]

## Indexing in Array:

In NDArrays if we want to target one element so we need to use integers in square brackets. Using [0] after name of array (in case of 2-D Array) will bring first row. And adding another square bracket [1] will bring second element of the first row.

But a better approach to get the same result is using array[0,1] where 0 corresponds to row and 1 corresponds to column. This can be used to assign specific values to that elements as well.

We can use negative indices as well to target elements just like we do in the lists.

## Assigning Values:

Using array [0,1] where 0 corresponds to row and 1 corresponds to column. This can be used to assign specific values to that element as well.

## Mathematical operations:

Simple mathematical operations like addition, subtraction, multiplication and division can be directly done to they array which will make changes to each element of the array accordingly.

Data types:

NumPy array can save data of all kinds of numbers. It can save Boolean data as well. You just need to change the parameter dtype to that specific data type. For example

ar\_d = np.array([[1,2,3], [4,5,6]], dtype= np.float32) or

ar\_c = np.array([[1,2,3], [4,5,6]], dtype="float32")

if dtype is set to Boolean so it’ll change all values to true if the number is not 0. Only zero will be changed to false.

To change numbers into str just use str in dtype parameter.

## NumPy Functions:

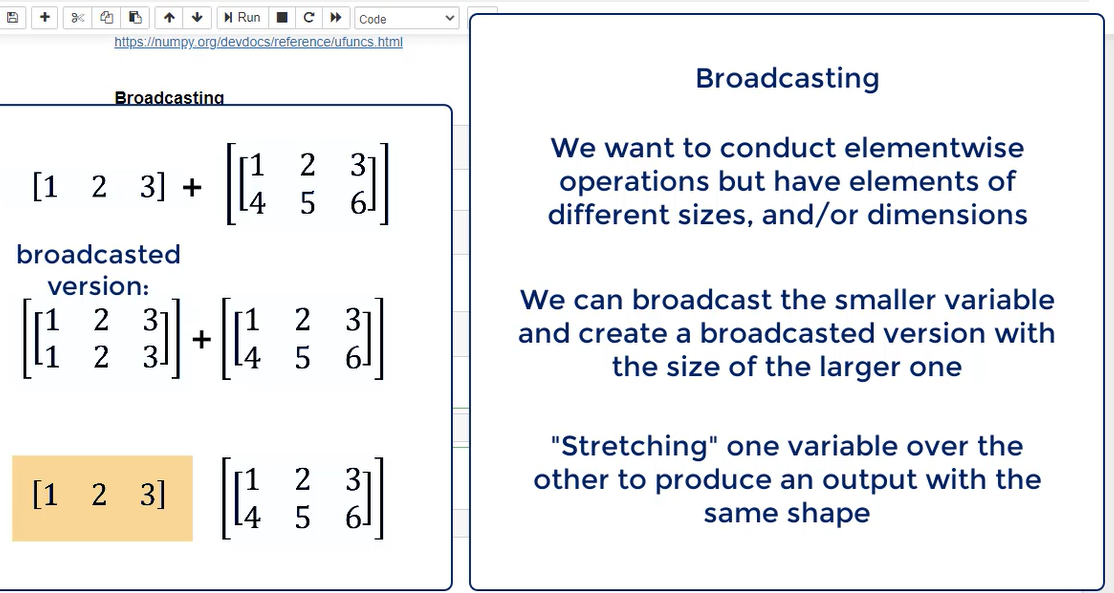
There are two kinds of numpy functions:

### 1. Universal Functions:

Universal functions (ufunc) basics. A universal function (or ufunc for short) is a function that operates on ndarrays in an element-by-element fashion, supporting array broadcasting, type casting, and several other standard features.

### Numpy Broadcasting:

A good definition is in the following picture:



### Running a Function on specific axis:

For this purpose, it is necessary to set axis parameter to 0 for columns and 1 for rows to run any function on column or row.

mean = np.mean(ar, axis=0) this will find mean of each column values

mean = np.mean(ar, axis=1) this will find mean of each row values.

## Slicing Array:

For slicing purpose, we’ll use [] which is used for index and add colon [:] to it. A number before colon represents starting row from where want slice and number after colon will be the limit and the slice will end before that.

If number before colon is not mentioned so it will be automatically considered as zero i.e. first row.

Slicing can be done by indexing as well but there is a little difference:

**Indexing**: Returns a **1D array** when accessing a single row/column.

**Slicing**: Always returns a **2D array** when accessing a range of rows/columns, even if the range contains only one row or column.

### Slicing Columns:

For column slicing in numpy, array[:, column\_index] method is used. The first colon before comma means that we need data from all rows, and there are 4 cases possible:

1. Need to slice a single column arr[:, col\_index]
2. Need to slice several specific columns arr[:, [col1, col2, ...]]
3. Need to slice a range of columns arr[:, start:end]
4. Need to slice a range with steps arr[:, start:end:step]

### Conditional Slicing

Conditional slicing in NumPy involves selecting elements or rows/columns from an array that meet a specific condition. It’s a powerful way to filter data based on criteria.

The way to do conditional slicing is to create a Boolean mask which means an array of True or False values created by applying a condition to an array. Elements that meet the condition are True, and those that don’t are False. For example:

arr = np.array([10, 15, 20, 25, 30])

ar = arr > 15

ar = [20,25,30]

## Initializing arrays:

There are several ways to initialize arrays:

1. If you wanna create array of only zeros so the code is np.zeros((dimension)).
2. If you wanna create array of only ones so the code is np.ones((dimension))
3. If you wanna create array of only a specific number so the code is np.full((dimension), number)
4. If you wanna create array of only a specific number with the shape of any previous array so the code is np.full\_like(previous\_array, number)
5. If you wanna create array of random number use this code np.random.rand(dimension). Note that this time the dimension will be mentioned directly in the parenthesis of method. This will give you float values.
6. For integers use this code np.random.randint(ending\_limit, size = (dimension))

## Mathematics

For multiplication you can use the following 4 methods:

1. Np.matmul(a,b)
2. Np.dot(a,b)
3. a.dot(b)
4. a@b

## Reading data from txt:

For reading data from txt where data is in specific format like csv, two methods can be used:

1. np.genfromtxt("data.txt", delimiter=",", dtype=np.int32)

this code can handle missing values as well. You have to add another parameter names “filling\_values” and give it a value which you want to appear at the place of missing value.

This method also supports mixed type of values.

1. np.loadtxt("data.txt", delimiter=',', dtype=np.int32)

the given two qualities are not available here but this method is usually considered fast pace.

## Reshape

To reshape arrays you can use this code:

data1.reshape(1,54)  
but its necessary that the next array must be compatible and makeable from the given shape.

## Concatenation of Array:

If you want to concatenate two arrays so there are 4 ways:

1. if both arrays are 1d, so use this code: np.concatenate((a,b)). This code will add the second array after elements of the first array and the resulted array will be 1d.
2. if arrays are 2d, so use this code: np.concatenate((a,b), axis=). But here axis parameter accepts 3 values. If axis parameter is not added or the value is given as 0, so in both cases it will add the second array to the first array vertically means that rows of the resulted array will increase accordingly.
3. If the parameter is set as axis=1, in this case the elements of the second array will be added to the first as horizontally. Means that the resulted array’s columns will increase accordingly.
4. If the parameter is set as axis=None, in this case the code will initially change the shape of array to 1d and then concatenate it like the first point.