1 Working with NumPy

Outline

1.1 Introduction

Python has become a defacto choice for many data manipulation and machine learning tasks. Much of the credit for this goes to two very useful and powerful libraries of Python : **Pandas** and **NumPy** (pronounced mostly as **Num Pie**).

In unit 1 of this subject, you shall be learning both of these Python libraries along with **PyPlot**. But **Panda** library has certain dependency on **NumPy** library, thus we are covering NumPy before Pandas.

This chapter will talk about the NumPy's most important data structure – NumPy arrays and their application. Although, we have informally introduced NumPy in Python Panda's discussion in class XI, here in this chapter, we shall cover NumPy in details, as per the syllabus.

But before we proceed, recall that in order to use any Python library, you need to import it using **import** statement. In order to use NumPy, you must import it in your module by using a statement like :

```
import numpy as np
```

You can use any identifier name in place of np but np has been a preferred choice, generally

(Conventionally, you give *import* statements at the top of a script or code). The above import statement has given **np** as an alias name for *numpy module*. Once imported with as <name>, you can use both names *i.e.*, **numpy** or **np** for functions *e.g.*, **numpy.array( )** is same as **np.array( )** after above import statement.

NOTE

Please note that if you have installed Python through Anaconda installer then *NumPy, SciPy, Pandas, Matplotlib* etc., are by default installed with Python, otherwise you need to install these separately through **pip** installer of Python.

1.2 What are NumPy Arrays ?

Array in general refers to a named group of homogeneous (of same type) elements. For instance, if you store the similar details of all sections together, *e.g.*, if you store *number of students* in each section of class X in a school, in a common name ***Students***, containing 5 entries as [34, 37, 36, 41, 40] then ***Students*** is actually an array that represents number of students in each section of class X. Like lists, you can access individual elements by giving *index* with array name *e.g.*, **Students[1]** will give details about 2nd section, **Students[3]** will give you details about 4th section and so on. Like lists, arrays also have first index as 0 (zero).

NUMPY ARRAY

> A NumPy array is simply a grid that contains values of the same/homogeneous type.

Please note that we are covering below just the basics of NumPy arrays so as to give you an idea of what NumPy arrays are like and how you can use them.

A NumPy array is simply a grid that contains values of the same/homogeneous type. You can think of a NumPy array like a *Python list* having all elements of similar types (but NumPy arrays are different from Python lists in functionality, which will be clear to you in coming lines.)

NumPy Arrays come in two forms :

- 1D (one dimensional) arrays knows as **Vectors** (have single row/column only).
- Multi-dimensional arrays known as **Matrices** (can have multiple rows and columns). In some cases, Matrices can still have only one row or one column.
  - 2D arrays are a type of multidimensional arrays.

1D ARRAY

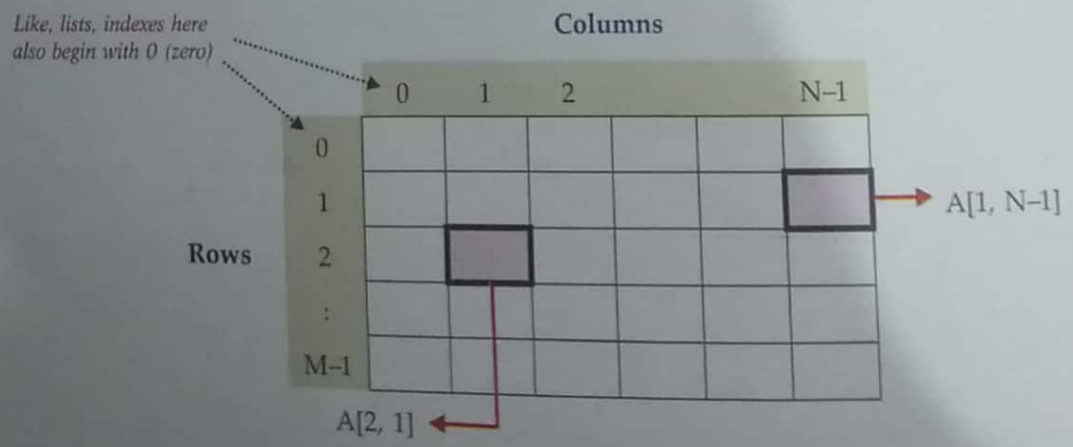
> A **one-dimensional array (1D array)** is a named group of contiguous set of elements having same data type.

Two-dimensional NumPy Arrays

A two-dimensional array is an array in which each element is itself a 1D array. For instance, an array A [*M*, *N*] is an *M* by *N* table with *M* rows and *N* columns containing $M \times N$ elements.

2D ARRAY

> A **two-dimensional array** (2D array) is an array in which each element is itself a 1D array.



The number of elements in a 2D array can be determined by : (*i*) calculating number of rows and number of columns, and then (*ii*) by multiplying number of rows with number of columns. For example, the number of elements in an array A [5, 7] is calculated as $5 \times 7 = 35$ elements.

You can check data type of a NumPy array's elements **using <arrayname>.dtype** *e.g.*, (see on the right)

```
In [11]: a1.dtype
Out[11]: dtype('int32')
```

Following Fig. 1.3 illustrates the basic terms associated with a NumPy array :

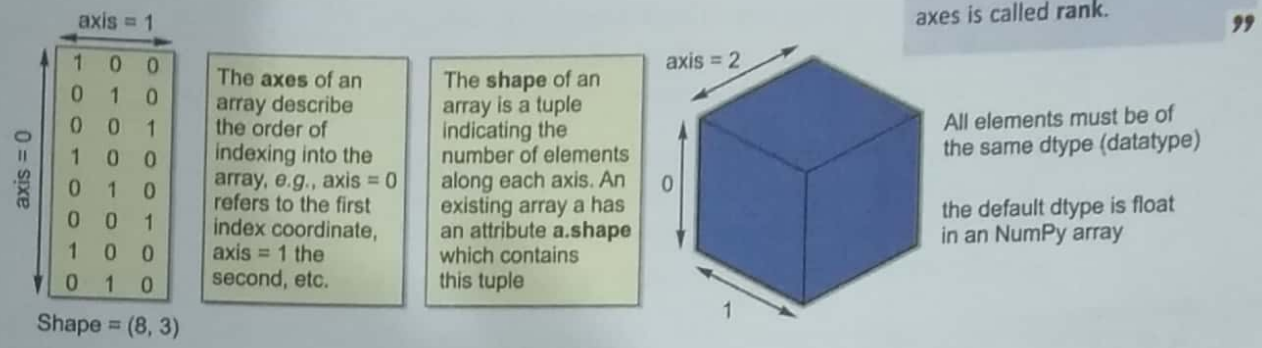


Figure 1.3 Anatomy of a NumPy Array

1.2.2 NumPy Arrays vs. Python Lists

Although NumPy arrays also hold elements just like Python lists and you can access elements of NumPy array in the same way you access a list's elements, yet NumPy arrays are different data structures from Python lists. Let us see how these are similar or different.

Indexing in NumPy Arrays – Similar to Lists

The NumPy array elements are also indexed in memory in the same manner as lists' elements, *e.g.*, if **npar** is a NumPy array with elements *12, 24, 36* and *48*, then its indexing will be just the same as lists. In other words, **forward indexing** starting from *0, 1, 2,* and so on and backward indexing with *–1, –2 , –3* and so on backwards. Let us see how these are similar or different.

So, just like lists, NumPy array's elements will be accessed using its indexes. That is,

```
npar[0] = 12  npar[-4] = 12        npar[1] = 24  npar[-3] = 24
npar[2] = 36  npar[-2] = 36        npar[3] = 48  npar[-1] = 48
```

Differences of NumPy Arrays with Python Lists

You just saw that the NumPy arrays and Lists have similar indexing and you can access the elements in the same manner. But the similarity ends here. NumPy arrays are very much different from Lists in many ways.

The key differences between a NumPy array and a list are :

- Unlike Python lists, once a NumPy array is created, you cannot change its size. You will have to create a new array or overwrite the existing one.

Please note that in order to specify NumPy's own datatypes, you can use them with their qualified names that is, either as :

```
numpy.<datatypename>
```

or with alternate name of numpy that you specified in import statement :

```
np.<datatype name>
```

e.g., to use **int64** datatype, you can give numpy.int64 or np.int64

Now, armed with this knowledge, you are ready to create NumPy arrays. So, let us move on to the same.

1.4 Creating NumPy Arrays

The NumPy library offers many functions for creating ndarrays of any rank. In this section, you shall learn to create ***rank 1 ndarrays***, *i.e.*, 1D ndarrays and ***rank 2 ndarrays***, *i.e.*, 2D ndarrays. The method of creating both 1D and 2D ndarray is similar, just the shape changes. Following examples will make it clear.

There are numerous ways through which you can create NumPy arrays. In earlier examples, we have shown you briefly how NumPy arrays can be created using **array( )**. In the following lines, we are discussing different ways of creating NumPy arrays.

1.4.1 Creating NumPy 1D Arrays

The most common method of creating NumPy arrays is **array( )** of NumPy library. But this function is mostly used for creating numeric arrays, although you can create other types of arrays also from it. NumPy makes available a plethora of functionc to create ndarrays.

We shall talk about some of these in this section.[1] Let us first talk about the most common functions for creating ndarrays. Please note that in this section, we shall discuss the minimal syntax of the functions (suits the beginners.)

(i) Creating ndarrays using array( )

NumPy's array( ) creates ndarray as per following syntax[2] :

```
NumPy.array(<arrayconvertibel object>, [<dtype>])
```

← *Optional argument*

e.g., (assuming NumPy has been imported as **np**)

```
nar1 = np.aray([2, 5.2, 1.0])
```

The above statement will do the following :

- **nar1** will be created as an ndarray object
- **nar1** will have 3 elements ; each element of the passed list will become an element of **nar1**.
- A datatype (dtype) will be assigned by default to the elements of the ndarray ; the datatype will be so chosen that can store all the elements of the ndarray, *e.g.*, for above list, a **float** type will be chosen as it can store all the three values 2, 5.2 and 1.0.

1. NumPy provides about more than 40 functions to create NumPy arrays. Covering all of these here would not be possible. So we shall talk about some most used functions for the same.
2. For complete syntax, refer to NumPy documentation as the syntax contains many optional arguments, not required for beginners.

Solution.

(a) A*2

```
 2  2  4
 6 10 16
26 42 68
```

(b) B/3

```
0  3
1  4
2  5
```

(c) A*x

```
[[1, 1, 2]
 [9, 15, 24]
 [26, 42, 68]]
```

(d) y*A

```
[[1, 3, 4],
 [3, 15, 16],
 [13, 63, 68]]
```

3. Write sh

(a) 2

4. Write of thi

5. Creat

6. Cons

GLOSSARY

ndarray	NumPy Data Array.
NumPy Array	Array of homogeneous elements.
Axes	Dimensions of NumPy array.
Rank	Number of axis in a NumPy array.

7.

Assignments

Type A : Short Answer Questions/Conceptual Questions

1. What is an array ?
2. How are 2D arrays internally stored ?
3. Define the following terms :
   (*a*) ndarrays (*b*) array slice (*c*) array subset
4. What are contiguous and non-contiguous subsets ?
5. What functions can you use for joining two or more ndarrays ?
6. What are functions using which you can extract contiguous subsets of ndarrays ?
7. Which functions allow you to extract only rows or columns from an ndarray ?
8. What are covariance, correlation and linear regression ?
9. Name functions using which you can perform arithmetic operations on ndarrays.

Type B : Application Based Questions

1. If a Python list is having 7 integers and a numpy array is also having 7 integers, then how are these two data structures similar or different from one another ?
2. Given a list L = [3, 4, 5] and an ndarray N having elements 3, 4, 5. What will be the result produced by :
   (*a*) L * 3 ? (*b*) N * 3 ? (*c*) L + L ? (*d*) N + N ?