**Recap**

In this post, we will look at translating R arrays (and matrixes) into python.

**1D R array**

A 1D R array prints like a vector.

library(tidyverse)

library(reticulate)

py\_run\_string("import numpy as np")

py\_run\_string("import pandas as pd")

(OneD<-array(1:6))

## [1] 1 2 3 4 5 6

But it is not truly a vector

OneD %>% is.vector()

## [1] FALSE

It is more specifically an atomic.

OneD %>% is.atomic()

## [1] TRUE

An atomic is sometimes termed as an atomic vector, which adds more to the confusion. ?is.atomic explains that “It is common to call the atomic types ‘atomic vectors’, but note that is.vector imposes further restrictions: an object can be atomic but not a vector (in that sense)”. Thus, OneD can be an atomic type but not a vector structure.

**1D R array is a python…**

No tricks here. A R array is translated into a python array. Thus, a 1D R array is translated into a 1D python array. The name of the python array is known as ndarray and is governed by the python packaged called numpy.

r.OneD

## array([1, 2, 3, 4, 5, 6])

type(r.OneD)

##

r.OneD.ndim

## 1

All python code for this post will be run within the {python} code chunk to explicitly print out the display for python array (i.e. array([ ]) )

**1D python array is a R…**

1 dimension python arrays are commonly used in data science for python.

p\_one= np.arange(6)

p\_one

## array([0, 1, 2, 3, 4, 5])

The 1D python array is translated into a 1D R array.

py$p\_one %>% class()

## [1] "array"

The translated array is an atomic type.

py$p\_one %>% is.atomic()

## [1] TRUE

An the translated array is not a vector which is expected of a 1D R array.

py$p\_one %>% is.vector()

## [1] FALSE

**2D R array**

A 2D R array is also known as a matrix.

(TwoD<-array(1:6, dim=c(2,3)))

## [,1] [,2] [,3]

## [1,] 1 3 5

## [2,] 2 4 6

TwoD %>% class()

## [1] "matrix"

**2D R array is a python…**

A 2D python array. python does not name have a special name for their 2D array.

r.TwoD

## array([[1, 3, 5],

## [2, 4, 6]])

type(r.TwoD)

##

r.TwoD.ndim

## 2

**2D python array**

Besides from 1D python array, 2D python array are also common in data science with python.

p\_two=np.random.randint(6, size=(2,3))

p\_two

## array([[2, 5, 2],

## [1, 0, 1]])

A 2D python array is translated into a 2D R array/ matrix.

py$p\_two %>% class()

## [1] "matrix"

**Reshaping 1D python array into 2D array**

Sometimes a python function requires a 2 dimension array and your input variable is a 1 dimension array. Thus, you will need to reshape your 1 dimension array into a 2 dimension array with numpy’s reshape function. Let us convert our 1 dimension array into a 2 dimension array which has 2 rows and 3 columns.

np.reshape(p\_one, (2,3))

## array([[0, 1, 2],

## [3, 4, 5]])

Let’s convert it into a 2D array which has 6 rows and 1 column.

np.reshape(p\_one, (6,1))

## array([[0],

## [1],

## [2],

## [3],

## [4],

## [5]])

The rows for the above is the same as the length of the 1D array. Thus, if you replace the 6 with the length of the 1D array, you will achieve the same result.

np.reshape(p\_one, (len(p\_one),1))

## array([[0],

## [1],

## [2],

## [3],

## [4],

## [5]])

Alternatively, you can also replace it with -1 if the input is a 1D array. -1 means that it is unspecified

np.reshape(p\_one, (-1,1))

## array([[0],

## [1],

## [2],

## [3],

## [4],

## [5]])

#Difference between R and python array  
One of the differences is the printing of values in the array.  
R are column-major arrays. The tables are filled column-wise. In other words, the left most column is filled from the top to the bottom before moving to neighbouring right column. This neighbouring column is filled up in a top-down fashion.

TwoD

## [,1] [,2] [,3]

## [1,] 1 3 5

## [2,] 2 4 6

The integrity of this column-major display is maintained when it is translated into python.

r.TwoD

## array([[1, 3, 5],

## [2, 4, 6]])

You would have noticed that python prints its array without the row (eg.[1,]) and column names (e.g. [,1]).

While python is able to use column-major ordered arrays, but it defaults to row-major ordering when arrays are created in python. In other words, values are filled from the first row in a left-to-right fashion before moving to the next row.

np.reshape(p\_one, (2,3))

## array([[0, 1, 2],

## [3, 4, 5]])

**python series**

Besides lists, 1D arrays, 2D arrays, there are other python data structures which are commonly used in data science with python. They are series and data frames which are governed by the pandas library. We will look at series in this post and data frames will be covered in a separate post. Series is a 1D array with axis labels.

PD=pd.Series(['banana',2])

PD

## 0 banana

## 1 2

## dtype: object

As series is a 1D array, when translated to R it will be classified as a R array.

py$PD %>% class()

## [1] "array"

However, the translated series appears as a R named list. The index of the series appear as the names in the R list.

py$PD

## $`0`

## [1] "banana"

##

## $`1`

## [1] 2

What did you know? A translated series is both a R array and R list

py$PD %>% is.array()

## [1] TRUE

py$PD %>% is.list()

## [1] TRUE