Lab-Arrays_Matrices

May 18, 2021

ARRAYS & MATRICES

0.1 Table of Contents

This lesson will cover some basics concepts about vector and factors in R:

- What is an Array?

- Array Indexing

- What is a Matrix?

- Concatenation Function

---

1 About the Dataset

Imagine you got many movie recomendations from your friends and compiled all of the recomendations in a table, with specific info about each movie.

The table has one row for each movie and several columns

- **name** - The name of the movie
- **year** - The year the movie was released
- **length_min** - The lenght of the movie in minutes
- **genre** - The genre of the movie
- **average_rating** - Average rating on Imdb
- **cost_millions** - The movie's production cost in millions
- **sequences** - The amount of sequences
- **foreign** - Indicative of whether the movie is foreign (1) or domestic (0)
- **age_restriction** - The age restriction for the movie

You can see part of the dataset below

What is an Array?

An array is a structure that holds values grouped together, like a 2 x 2 table of 2 rows and 2 columns. Arrays can also be **multidimensional**, such as a 2 x 2 x 2 array.

What is the difference between an array and a vector? Vectors are always one dimensional like a single row of data. On the other hand, an array can be multidimensional (stored as rows and columns). The "dimension" indicates how many rows of data there are.

Let's create a 4 x 3 array (4 rows, 3 columns) The example below is a vector of 9 movie names, hence the data type is the same for all the elements.

```
[ ]: #lets first create a vector of nine movies
     movie_vector <- c("Akira", "Toy Story", "Room", "The Wave", "Whiplash",
                       "Star Wars", "The Ring", "The Artist", "Jumanji")
     movie_vector
```

To create an array, we can use the **array()** function.

```
[ ]: movie_array <- array(movie_vector, dim = c(4,3))
     movie_array
```

Note that **arrays are created column-wise**. Did you also notice that there were only 9 movie names, but the array was 4 x 3? The original **vector doesn't have enough elements** to fill the entire array (that should have 3 x 4 = 12 elements). So R simply fills rest of the empty values by going back to the beginning of the vector and starting again ("Akira", "Toy story", "Room" in this case).

We also needed to provide `c(4,3)` as a second *argument* to specify the number of rows (4) and columns (3) that we wanted.

[Tip] What is an "argument"? How are "arguments" different from "*parameters*"?
Arguments and parameters are terms you will hear constantly when talking about **functions**.
- The *parameters* are the input variables used in a function, like **dim** in the function **array()**.
- The *arguments* refer to the *values* for those parameters that a function takes as inputs, like **c(4,3)**
We actually don't need to write out the name of the parameter (dim) each time, as in:
`array(movie_vector, c(4,3))`
As long as we write the arguments out in the correct order, R can interpret the code.

Arguments in a function may sometimes need to be of a **specific type**. For more information on each function, you can open up the help file by running the function name with a ? beforehand, as in:
`?array`

Array Indexing

Let's look at our array again:

```
[ ]: movie_array
```

To access an element of an array, we should pass in [**row, column**] as the row and column number of that element.
For example, here we retrieve **Whiplash** from row 1 and column 2:

```
[ ]: movie_array[1,2] #[row, column]
```

To display all the elements of the first row, we should put 1 in the row and nothing in the column part. Be sure to keep in the comma after the `1`.

```
[ ]: movie_array[1,]
```

Likewise, you can get the elements by column as below.

```
[ ]: movie_array[,2]
```

To get the dimension of the array, **dim()** should be used.

```
[ ]: dim(movie_array)
```

We can also do math on arrays. Let's create an array of the lengths of each of the nine movies used earlier.

```
[ ]: length_vector <- c(125, 81, 118, 81, 106, 121, 95, 100, 104)
     length_array <- array(length_vector, dim = c(3,3))
     length_array
```

Let's add 5 to the array, to account for a 5-min bathroom break:

```
[ ]: length_array + 5
```

Tip: Performing operations on objects, like adding 5 to an array, does not change the object. **To change the object, we would need to assign the new result to itself**.

Using Logical Conditions to Subset Arrays

Which movies can I finish watching in two hours? Using a logical condition, we can check which movies are less than 2 hours long.

```
[ ]: mask_array <- length_array > 120
     mask_array
```

Using this array of TRUEs and FALSEs, we can subset the array of movie names:

```
[ ]: x_vector <- c("Akira", "Toy Story", "Room", "The Wave", "Whiplash",
                    "Star Wars", "The Ring", "The Artist", "Jumanji")
     x_array <- array(x_vector, dim = c(3,3))

     x_array[mask_array]
```

What is a Matrix?

Matrices are a subtype of arrays. A matrix **must** have 2 dimensions, whereas arrays are more flexible and can have, 1, 2 or more dimensions.

To create a matrix out of a vector , you can use **matrix()**, which takes in an argument for the vector, an argument for the number of rows and another for the number of columns.

```
[ ]: movie_matrix <- matrix(movie_vector, nrow = 3, ncol = 3)
```

```
[ ]: movie_matrix
```

1.0.1 Accessing elements of a matrix

As with arrays, you can use [**row, column**] to access elements of a matrix. To retrieve "Akira", you should use [1,1] as it lies in the first row and first column.

```
[ ]: movie_matrix[1,1]
```

To get data from a certain range, the folowing code can help. This takes the elements from rows 2 to 3, and from columns 1 to 2.

```
[ ]: movie_matrix[2:3, 1:2]
```

Concatenation function

A concatenation function is used to combine two vectors into one vector. It combines values of both vectors. Lets create a new vector for the upcoming movies as upcoming_movie and add them to the movie_vector to create a new_vector of movies.

```
[ ]: upcoming_movie <- c("Fast and Furious 8", "xXx: Return of Xander Cage",␣
     ↪"Suicide Squad")
```

```
[ ]: new_vector <- c(movie_vector, upcoming_movie)
```

```
[ ]: new_vector
```

Scaling R with big data As you learn more about R, if you are interested in exploring platforms that can help you run analyses at scale, you might want to sign up for a free account on IBM Watson Studio, which allows you to run analyses in R with two Spark executors for free.

1.0.2 About the Author:

Hi! It's Hiten Patel, the author of this notebook. I hope you found arrays and matrices easy to learn! As you start learning the foundations of R, feel free to connect with me if you have any questions.

Copyright © IBM Cognitive Class. This notebook and its source code are released under the terms of the MIT License.