lab2_jupyter_arrays_matrics

January 29, 2022

Arrays and Matrices

Estimated time needed: **20** minutes

0.1 Objectives

After completing this lab you will be able to:

- Understand what is an array by coding practices
- Operate on arrays
- Understand what is a matrix by coding practices
- Operate on matrices

0.2 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 data 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.

[1]:
```
#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
```

1. 'Akira' 2. 'Toy Story' 3. 'Room' 4. 'The Wave' 5. 'Whiplash' 6. 'Star Wars' 7. 'The Ring' 8. 'The Artist' 9. 'Jumanji'

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

[2]:
```
movie_array <- array(movie_vector, dim = c(4,3))
movie_array
```

A matrix: 4 × 3 of type chr

Akira	Whiplash	Jumanji
Toy Story	Star Wars	Akira
Room	The Ring	Toy Story
The Wave	The Artist	Room

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:

[3]:
```
movie_array
```

A matrix: 4 × 3 of type chr

Akira	Whiplash	Jumanji
Toy Story	Star Wars	Akira
Room	The Ring	Toy Story
The Wave	The Artist	Room

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:

[4]:
```
movie_array[1,2] #[row, column]
```

'Whiplash'

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`.

[5]:
```
movie_array[1,]
```

1. 'Akira' 2. 'Whiplash' 3. 'Jumanji'

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

[6]:
```
movie_array[,2]
```

1. 'Whiplash' 2. 'Star Wars' 3. 'The Ring' 4. 'The Artist'

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

[7]:
```
dim(movie_array)
```

1. 4 2. 3

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

[8]:
```
length_vector <- c(125, 81, 118, 81, 106, 121, 95, 100, 104)
length_array <- array(length_vector, dim = c(3,3))
length_array
```

A matrix: 3 × 3 of type dbl

125	81	95
81	106	100
118	121	104

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

[9]:
```
length_array + 5
```

A matrix: 3 × 3 of type dbl

130	86	100
86	111	105
123	126	109

Coding Exercise: in the code cell below, create a new length vector with 12 elements and create a 4 x 3 array from it

[11]:
```
# Write your code below. Don't forget to press Shift+Enter to execute the cell
length_vector <- c(125, 81, 118, 81, 106, 121, 95, 100, 104,85,64,83)
length_array <- array(length_vector, dim = c(4,3))
length_array
```

A matrix: 4 × 3 of type dbl

125	106	104
81	121	85
118	95	64
81	100	83

Click here for the solution

```
new_length_vector <- c(125, 81, 118, 81, 106, 121, 95, 100, 104, 85, 64, 83)
new_length_array <- array(new_length_vector, dim = c(4,3))
new_length_array
```

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.

[12]:
```
mask_array <- length_array < 120
mask_array
```

A matrix: 4 × 3 of type lgl

FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	TRUE
TRUE	TRUE	TRUE

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

[13]:
```
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]
```

1. 'Toy Story' 2. 'Room' 3. 'The Wave' 4. 'Whiplash' 5. 'The Ring' 6. 'The Artist' 7. 'Jumanji' 8. NA 9. NA 10. NA

Coding Exercise: in the code cell below, find all movie titles with length less than or equal to 90 minutes

[16]:
```
# Write your code below. Don't forget to press Shift+Enter to execute the cell
length_vector <- c(125, 81, 118, 81, 106, 121, 95, 100, 104)
length_array <- array(length_vector, dim = c(3,3))
mask_array <- length_array < 90
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]
```

1. 'Toy Story' 2. 'The Wave'

Click here for the solution

```
mask_array <- length_array <= 90
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, one, two, 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.

[17]:
```
movie_matrix <- matrix(movie_vector, nrow = 3, ncol = 3)
```

[18]:
```
movie_matrix
```

A matrix: 3 × 3 of type chr

Akira	The Wave	The Ring
Toy Story	Whiplash	The Artist
Room	Star Wars	Jumanji

Coding Exercise: in the code cell below, create a new length vector with 12 elements and create a 4 x 3 matrix from it

[20]:
```
# Write your code below. Don't forget to press Shift+Enter to execute the cell
new_length_vector <- c(125, 81, 118, 81, 106, 121, 95, 100, 104, 85, 64, 80)
new_movie_matrix <- matrix(new_length_vector, nrow = 4, ncol = 3)
new_movie_matrix
```

A matrix: 4 × 3 of type dbl

125	106	104
81	121	85
118	95	64
81	100	80

Click here for the solution

```
new_length_vector <- c(125, 81, 118, 81, 106, 121, 95, 100, 104, 85, 64, 80)
new_movie_matrix <- matrix(new_length_vector, nrow = 4, ncol = 3)
new_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.

[21]:
```
movie_matrix[1,1]
```

'Akira'

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.

[22]:
```
movie_matrix[2:3, 1:2]
```

A matrix: 2 × 2 of type chr

Toy Story	Whiplash
Room	Star Wars

Coding Exercise: in the code cell below, get the second row of the matrix

[24]:
```
# Write your code below. Don't forget to press Shift+Enter to execute the cell
movie_matrix[2,]
```

1. 'Toy Story' 2. 'Whiplash' 3. 'The Artist'

Click here for the solution

```
movie_matrix[2,]
```

Coding Exercise: in the code cell below, get the second column of the matrix

[26]:
```
# Write your code below. Don't forget to press Shift+Enter to execute the cell
movie_matrix[,2]
```

1. 'The Wave' 2. 'Whiplash' 3. 'Star Wars'

Click here for the solution

```
movie_matrix[,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 a `upcoming_movie` variable and add them to the `movie_vector` variable to create a `new_vector` of movies.

[27]:
```
upcoming_movie <- c("Fast and Furious 8", "xXx: Return of Xander Cage",␣
↪"Suicide Squad")
```

[28]:
```
new_vector <- c(movie_vector, upcoming_movie)
```

[29]:
```
new_vector
```

1. 'Akira' 2. 'Toy Story' 3. 'Room' 4. 'The Wave' 5. 'Whiplash' 6. 'Star Wars' 7. 'The Ring' 8. 'The Artist' 9. 'Jumanji' 10. 'Fast and Furious 8' 11. 'xXx: Return of Xander Cage' 12. 'Suicide Squad'

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.

1.1 Authors

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.

1.1.1 Other Contributors

Yan Luo

1.2 Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2021-03-03	2.0	Yan	Added coding tasks

##

© IBM Corporation 2021. All rights reserved.