Arrays and Matrices

Estimated time needed: **20** minutes

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

2 About the Dataset

You have received many movie recommendations from your friends and compiled all of the recommendations into a table, with information about each movie.

This 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 length 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;
- **foreign** - Indicative of whether the movie is foreign (1) or domestic (0);

- **age_restriction** - The age restriction for the movie;

Here's what the data looks like:

name	year	length_min	genre	average_rating	cost_millions	foreign	age_restriction
Toy Story	1995	81	Animation	8.3	30.0	0	0
Akira	1998	125	Animation	8.1	10.4	1	14
The Breakfast Club	1985	97	Drama	7.9	1.0	0	14
The Artist	2011	100	Romance	8.0	15.0	1	12
Modern Times	1936	87	Comedy	8.6	1.5	0	10
Fight Club	1999	139	Drama	8.9	63.0	0	18
City of God	2002	130	Crime	8.7	3.3	1	18
The Untouchables	1987	119	Drama	7.9	25.0	0	14
Star Wars Episode IV	1977	121	Action	8.7	11.0	0	10
American Beauty	1999	122	Drama	8.4	15.0	0	14

3 What is an Array?

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

3.1 What is the difference between an array and a vector?

Vectors are always one dimensional like a single row/column 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.

3.2 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
```

```
## [1] "Akira"      "Toy Story"  "Room"       "The Wave"   "Whiplash"
## [6] "Star Wars"  "The Ring"   "The Artist" "Jumanji"
```

To create an array, we can use the `array` function.

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

```
##      [,1]        [,2]         [,3]
## [1,] "Akira"     "Whiplash"   "Jumanji"
## [2,] "Toy Story" "Star Wars"  "Akira"
## [3,] "Room"      "The Ring"   "Toy Story"
## [4,] "The Wave"  "The Artist" "Room"
```

Note that **arrays are filled in column-wise order**. 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 up the entire array (that should have $3 \times 4 = 12$ 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 dimension of the array we wanted, namely the number of rows (4) and columns (3).

3.3 [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`

4 Array Indexing

Let’s look at our array again:

```
movie_array
```

```
##      [,1]        [,2]         [,3]
## [1,] "Akira"     "Whiplash"   "Jumanji"
## [2,] "Toy Story" "Star Wars"  "Akira"
## [3,] "Room"      "The Ring"   "Toy Story"
## [4,] "The Wave"  "The Artist" "Room"
```

To access an element of an array, we should pass in `[row, column]` as the row and column index of that element. For example, here we retrieve “Whiplash” at row 1,column 2:

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

```
## [1] "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`.

```
movie_array[1, ]
```

```
## [1] "Akira"    "Whiplash" "Jumanji"
```

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

```
movie_array[, 2]
```

```
## [1] "Whiplash"   "Star Wars"  "The Ring"   "The Artist"
```

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

```
dim(movie_array)
```

```
## [1] 4 3
```

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
```

```
##      [,1] [,2] [,3]
## [1,]  125   81   95
## [2,]   81  106  100
## [3,]  118  121  104
```

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

```
length_array + 5
```

```
##      [,1] [,2] [,3]
## [1,]  130   86  100
## [2,]   86  111  105
## [3,]  123  126  109
```

Note: 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, i.e. `length_array = length_array + 5`.

4.1 Coding Exercise 1

In the code cell below, create a new length vector with 12 elements and create a 4×3 array from it.

```
# Write your code below. Don't forget to press Shift+Enter to execute the cell
```

5 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
```

```
##       [,1]  [,2] [,3]
## [1,] FALSE  TRUE TRUE
## [2,]  TRUE  TRUE TRUE
## [3,]  TRUE FALSE TRUE
```

Using this array of `TRUE` s and `FALSE` s, 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]
```

```
## [1] "Toy Story"  "Room"       "The Wave"   "Whiplash"   "The Ring"
## [6] "The Artist" "Jumanji"
```

5.1 Coding Exercise 2

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

```
# Write your code below. Don't forget to press Shift + Enter to execute the cell
```

6 What is a Matrix?

Matrices are a special type 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.

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

```
##      [,1]        [,2]        [,3]
## [1,] "Akira"     "The Wave"  "The Ring"
## [2,] "Toy Story" "Whiplash"  "The Artist"
## [3,] "Room"      "Star Wars" "Jumanji"
```

6.1 Coding Exercise 3

In the code cell below, create a new length vector with 12 elements and create a 4×3 matrix from it.

```
# Write your code below. Don't forget to press Shift + Enter to execute the cell
```

6.2 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]
```

```
## [1] "Akira"
```

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

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

```
##      [,1]        [,2]
## [1,] "Toy Story" "Whiplash"
## [2,] "Room"      "Star Wars"
```

6.3 Coding Exercise 4

In the code cell below, get the second row of the matrix `movie_matrix` .

```
# Write your code below. Don't forget to press Shift + Enter to execute the cell
```

6.4 Coding Exercise 5

In the code cell below, get the second column of the matrix `movie_matrix` .

```
# Write your code below. Don't forget to press Shift + Enter to execute the cell
```

7 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.

```
upcoming_movie <- c("Fast and Furious 8", "xXx: Return of Xander Cage", "Suicide Squad")
new_vector <- c(movie_vector, upcoming_movie)
new_vector
```

```
##  [1] "Akira"                      "Toy Story"
##  [3] "Room"                       "The Wave"
##  [5] "Whiplash"                   "Star Wars"
##  [7] "The Ring"                   "The Artist"
##  [9] "Jumanji"                    "Fast and Furious 8"
## [11] "xXx: Return of Xander Cage" "Suicide Squad"
```

8 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 (http://cocl.us/dsx_rp0101en), which allows you to run analyses in R with two Spark executors for free.

9 Author(s)

Weiqing Wang (https://www.linkedin.com/in/weiqing-wang-641640133/)

Weiqing is a Data Scientist intern at IBM Canada Ltd. Weiqing holds an Honours Bachelor of Science from the University of Toronto with two specialist degrees, respectively in computer science and statistical sciences. He is presently working towards a graduate degree in computer science at the University of Toronto.

Weiqing avail himself of this opportunity to acknowledge that this notebook is based heavily on past offerings of this course, in particular Helly Patel (https://ca.linkedin.com/in/helly-patel-90344750)'s, former Junior Software Engineer at IBM.

9.1 Other contributors

Yan Luo, PhD (https://www.linkedin.com/in/yan-luo-96288783/).

10 Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2021-07-10	1.0	Weiqing Wang	Initial Version Created.