Vectors and Factors

Welcome!

By the end of this notebook, you will have learned about **vectors and factors**, two very important data structures in R.

Estimated time needed: **30** minutes

1 Objectives

After completing this lab you will be able to: - Understand the data structure of vector and factor in R. - Perform vector and factor operations in R.

2 About the Dataset

Suppose 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 Vectors

Vectors are collections of numbers, characters or logical data organized in one-dimensional arrays. In other words, a vector is a simple tool to store your grouped data, element by element.

In R, you create a vector with the "combine function" (`c()`). Then, you place its elements separated by commas as its arguments.

Vectors will be handy in the future as they allow you to efficiently apply operations on a series of data.

Note that the elements in a vector must be of the same class. For example, all should be either number, character, or logical.

3.1 Numeric Vectors

Let's say we have four movie release dates (1985, 1999, 2015, 1964), and we want to assign them to a single variable, `release_year`. This means we'll need to create a vector using the `c()` function to combine them.

Using numbers, this becomes a **numeric vector**.

```
release_year <- c(1985, 1999, 2015, 1964)
release_year
```

```
## [1] 1985 1999 2015 1964
```

3.2 Character Vectors

What if we put strings as arguments for `c()` function? Then this becomes a **character vector**.

```
titles <- c("Toy Story", "Akira", "The Breakfast Club")
titles
```

```
## [1] "Toy Story"          "Akira"              "The Breakfast Club"
```

3.3 Logical Vectors

There are also **logical vectors**, which consist of `TRUE`s and `FALSE`s. They're particular important when you want to apply element-wise predicates.

```
titles == "Akira" # which item in `titles` is equal to "Akira"?
```

```
## [1] FALSE  TRUE FALSE
```

3.4 Coding Exercise 1

In the code cell below, find which item in `titles` equals to "Toy Story"

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

3.5 [Tip] `TRUE` and `FALSE` in R

Did you know? R only recognizes `TRUE`, `FALSE`, `T` and `F` as special values for true and false. That means all other spellings, including `True` and `true`, are not interpreted by R as logical values.

4 Vector Operations

4.1 Adding more elements to a vector

You can add more elements to a vector with the same `c()` function you use to create vectors:

```
release_year <- c(1985, 1999, 2015, 1964)
release_year
```

```
## [1] 1985 1999 2015 1964
```

```
release_year <- c(release_year, 2016:2018)
release_year
```

```
## [1] 1985 1999 2015 1964 2016 2017 2018
```

4.2 Length of a vector

How do we check how many items there are in a vector? We can use the `length()` function:

```
release_year
```

```
## [1] 1985 1999 2015 1964 2016 2017 2018
```

```
length(release_year)
```

```
## [1] 7
```

4.3 Head and Tail of a vector

We can also retrieve just the **first few items** using the `head()` function. By default, the first six items of the vector is returned.

```
head(release_year) # first six items
```

```
## [1] 1985 1999 2015 1964 2016 2017
```

We can specify the number of items in the vectors to be returned with `n` parameter of function `head()`.

```
head(release_year, n = 2) # first 2 items
```

```
## [1] 1985 1999
```

```
# The following line also works as n is the 2nd parameter of function "head".
# head(release_year, 2)
```

We can also retrieve just the **last few items** using the `tail()` function. Similar to `head()` function, it returns the last six items of a vector by default:

```
tail(release_year) # last six items
```

```
## [1] 1999 2015 1964 2016 2017 2018
```

We can specify the number of items in the vectors to be returned with `n` parameter of function `tail()`

```
tail(release_year, 2) # last two items
```

```
## [1] 2017 2018
```

4.4 Sorting and Extremum of Vectors

We can also sort a vector:

```
sort(release_year)
```

```
## [1] 1964 1985 1999 2015 2016 2017 2018
```

We can also **sort in decreasing order**:

```
sort(release_year, decreasing = TRUE)
```

```
## [1] 2018 2017 2016 2015 1999 1985 1964
```

But if you just want the minimum and maximum values of a vector, you can use the `min()` and `max()` functions:

```
min(release_year)
```

```
## [1] 1964
```

```
max(release_year)
```

```
## [1] 2018
```

4.5 Average of Numbers

If you want to check the average cost of movies produced in 2014, what would you do?

Of course, one way is to add all the numbers together, then divide by the number of movies:

```
cost_2014 <- c(8.6, 8.5, 8.1)

# sum results in the sum of all elements in the vector
avg_cost_2014 <- sum(cost_2014) / 3
avg_cost_2014
```

```
## [1] 8.4
```

You also can use the `mean()` function to find the average of the numeric values in a vector:

```
mean_cost_2014 <- mean(cost_2014)
mean_cost_2014
```

```
## [1] 8.4
```

4.6 Giving Names to Values in a Vector

Suppose you want to remember which year corresponds to which movie.

With vectors, you can give names to the elements of a vector using the `names()` function:

```
#Creating a year vector
release_year <- c(1985, 1999, 2010, 2002)

#Assigning names
names(release_year) <-
  c("The Breakfast Club", "American Beauty", "Black Swan", "Chicago")

release_year
```

```
## The Breakfast Club    American Beauty         Black Swan            Chicago
##               1985               1999               2010               2002
```

Now, you can retrieve the values based on the names:

```
release_year[c("American Beauty", "Chicago")]
```

```
## American Beauty         Chicago
##            1999            2002
```

Note that the values of the vector are still the years. We can see this in action by adding a number to the first item:

```
release_year[1] + 100 #adding 100 to the first item changes the year
```

```
## The Breakfast Club
##               2085
```

And you can retrieve the names of the vector using `names()`:

```
names(release_year)
```

```
## [1] "The Breakfast Club" "American Beauty"    "Black Swan"
## [4] "Chicago"
```

4.7 Coding Exercise 2

In the code cell below, calculate the release year difference between 'Black Swan' and 'The Breakfast Club':

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

4.8 Summarizing Vectors

You can also use the `summary()` function for simple descriptive statistics: minimum, first quartile, median, mean, third quartile, maximum:

```
summary(cost_2014)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    8.10    8.30    8.50    8.40    8.55    8.60
```

4.9 Using Logical Operations on Vectors

A vector can also be comprised of `TRUE` and `FALSE`, which are the **logical values** in R. These boolean values are used to indicate whether a condition is true or false.

Let's check whether a movie year of 1997 is newer than (**greater in value than**) one made in 2000.

```
movie_year <- 1997
movie_year > 2000
```

```
## [1] FALSE
```

You can also make a logical comparison across multiple items in a vector. Which movie release years here are "greater" than 2014?

```
movies_years <- c(1998, 2010, 2016)
movies_years > 2014
```

```
## [1] FALSE FALSE  TRUE
```

We can also check for **equivalence**, using `==` operator. Let's check which movie year is equal to 2015.

```
movies_years == 2015 # is equal to 2015?
```

```
## [1] FALSE FALSE FALSE
```

If you want to check which ones are **not equal** to 2015, you can use `!=` operator:

```
movies_years != 2015
```

```
## [1] TRUE TRUE TRUE
```

4.9.1 **[Tip]** Logical Operators in R

You can do a variety of logical operations in R including:

- Checking equivalence ($=$) – R command `x == y` returns `TRUE` if variables `x` and `y` share the same value or otherwise `FALSE`.
- Checking non-equivalence ($\neq$) – R command `x != y` returns `FALSE` if variables `x` and `y` share the same value or otherwise `TRUE`.
- Greater than ($>$) – R command `x > y` returns `TRUE` if variables `x` has a greater value compared to `y` or otherwise `FALSE`.
- Greater than or equal to ($\geqslant$) – R command `x >= y` returns `TRUE` if variable `x` has a value that is greater than or equal to that of the variable `y`, and `FALSE` otherwise.
- Less than ($<$) – R command `x < y` returns `TRUE` if variable `x` has a smaller value compared to `y` or otherwise `FALSE`.
- Less than or equal to ($\leqslant$) – R command `x <= y` returns `TRUE` if variable `x` has a value that is smaller than or equal to that of the variable `y`, and `FALSE` otherwise.

5 Subsetting Vectors

What if you wanted to retrieve the second year from the following **vector of movie years**?

```
movie_years <- c(1985, 1999, 2002, 2010, 2012)
movie_years
```

```
## [1] 1985 1999 2002 2010 2012
```

5.1 Retreiving entries of vectors at specified index(es)

To retrieve the **second year**, you can use square brackets `[]` and place the index inside:

```
movie_years[2] # The second item
```

```
## [1] 1999
```

Similarly, to retrieve the **third year**, you can use the following line:

```
movie_years[3] # The third item
```

```
## [1] 2002
```

And if you want to retrieve **multiple items**, you can pass in a vector of indexes to the square brackets `[]`:

```
movie_years[c(1, 3)] # The first and third items
```

```
## [1] 1985 2002
```

You can also get a sequential subset as follows:

```
movie_years[c(2:4)] # The second to the fourth items
```

```
## [1] 1999 2002 2010
```

or, more succinctly:

```
movie_years[2:4]
```

```
## [1] 1999 2002 2010
```

5.2 Reverse Selection

To retrieve a vector without one or more items, you can use negative indexing. For example, the following returns a vector slice **without the first item**.

```
titles <-
  c("Black Swan", "Jumanji", "City of God", "Toy Story", "Casino")
titles[-1]
```

```
## [1] "Jumanji"     "City of God" "Toy Story"   "Casino"
```

You can assign the new vector to a new variable:

```
new_titles <- titles[-1] # removes "Black Swan", the first item of vector 'titles'
new_titles
```

```
## [1] "Jumanji"     "City of God" "Toy Story"   "Casino"
```

5.3 Missing Values (NA)

Sometimes values in a vector are missing and you have to show them using `NA`, which is a special value in R for "Not Available". For example, if you don't know the age restriction for some movies, you can use `NA`.

```
age_restric <- c(14, 12, 10, NA, 18, NA)
age_restric
```

```
## [1] 14 12 10 NA 18 NA
```

5.3.1 [Tip] Checking `NA`s in R

You can check if a value is NA by using the `is.na()` function, which returns `TRUE` or `FALSE`.

- Check if a value / variable is `NA`: `is.na(NA)`
- Check if a value / variable is NOT `NA`: `!is.na(2)`

5.4 Subsetting vectors based on a logical condition

What if we want to know which movies were created after year 2000? We can simply apply a logical comparison across all the items in a vector:

```
release_year > 2000
```

```
## The Breakfast Club    American Beauty         Black Swan            Chicago
##              FALSE              FALSE               TRUE               TRUE
```

When passing in a vector of booleans to a vector `x` with the same length, a subset of `x` with entries at `TRUE`s in the vector of booleans will be returned. For example, the returned vector of booleans above has `TRUE` at indexes of `realease_year` where the year is strictly greater than 2000. To retrieve the actual movie years after year 2000, you can simply place logical vector in the **square brackets** `[]`:

```
release_year[release_year > 2000] # returns a vector for elements that returned TRUE for the con
dition
```

```
## Black Swan    Chicago
##       2010       2002
```

As you may notice, subsetting vectors in R works by retrieving items that were TRUE for the provided condition. For example, `year[year > 2000]` can be verbally explained as:

> "From the vector `year`, return only values where the values are TRUE for `year > 2000`"

You can even manually write out `TRUE` or `T` for the values you want to subset:

```
release_year
```

```
## The Breakfast Club    American Beauty         Black Swan            Chicago
##               1985               1999               2010               2002
```

```
release_year[c(T, F, F, F)] #returns the values for which the value's index is TRUE
```

```
## The Breakfast Club
##               1985
```

5.5 Coding Exercise 3

In the code cell below, find movies whose release year in or prior to and including year 1999.

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

6 Factors

Factors are variables in R which take values from a fixed, discrete set of values or levels. Such variables are commonly being referred to as **categorical variables**.

The difference between a categorical variable and a numeric variable is that a categorical variable expresses qualitative values – attributes such as “age group”, “gender”, or “favourite music genre”, while numerical variables communicate quantitative information – data that can be interpreted mathematically, such as “fuel economy in MPG”, “price per bushel” and “age”. In addition, numerical variables can be either continuous or discrete.

For example, the height of a tree is a continuous numeric variable, while the number of books held in a particular library is a discrete numerical variable. The various species of animals on earth are (nominal) categorical variables. Categorical variables need not be expressed as text. Indeed, factors in R are actually stored as a vector of integer values – they carry a corresponding set of character values, but these are only used for display purposes.

One of the most critical uses of factors is in statistical modelling. Since categorical variables are being handled by statistical models differently than continuous variables, storing data as factors ensures that the modelling functions will treat such data correctly.

Let’s start with a vector of genres:

```
genre_vector <- c("Comedy", "Animation", "Crime", "Comedy", "Animation")
genre_vector
```

```
## [1] "Comedy"    "Animation" "Crime"     "Comedy"    "Animation"
```

As you may have noticed, you can theoretically group the items above into three categories of genres: *Animation*, *Comedy* and *Crime*. In R-terms, we call these categories **factor levels**.

The function `as.factor()` converts a vector into a factor, and creates a factor level for each unique element.

```
genre_factor <- as.factor(genre_vector)
levels(genre_factor)
```

```
## [1] "Animation" "Comedy"    "Crime"
```

6.1 Summarizing Factors

When you have a large vector, it becomes difficult to identify which levels are most common (e.g., “How many ‘Comedy’ movies are there?”).

To answer this, we can use `summary()`, which produces a **frequency table**, as a named vector.

```
summary(genre_factor)
```

```
## Animation    Comedy     Crime
##         2         2         1
```

And recall that you can sort the values of the table using **sort()**.

```
sort(summary(genre_factor)) #sorts values by ascending order
```

```
##     Crime Animation    Comedy
##         1         2         2
```

6.2 Ordinal Factors

There are two types of categorical variables: **nominal categorical variable** and **ordinal categorical variable**.

A **nominal variable** is a categorical variable for names, without an implied order. This means that it is impossible to say that ‘one is better or larger than the other’. For example, consider `movie_genre` with the categories *Animation*, *Comedy* and *Crime*. Here, there is no implicit order of low-to-high or high-to-low between the categories.

In contrast, **ordinal variables** do have a natural ordering. Consider for example, `movie_length` with the categories: *Very short*, *Short* , *Medium*, *Long*, *Very long*. Here it is obvious that *Medium* stands above *Short*, and *Long* stands above *Medium*.

```
movie_length <- c("Very Short",
                  "Short",
                  "Medium",
                  "Short",
                  "Long",
                  "Very Short",
                  "Very Long")
movie_length
```

```
## [1] "Very Short" "Short"      "Medium"     "Short"      "Long"
## [6] "Very Short" "Very Long"
```

`movie_length` should be converted to an ordinal factor since its categories have a natural ordering. By default, the function `factor()` transforms `movie_length` into an unordered factor.

To create an **ordinal factor**, you have to add two additional arguments: `ordered` and `levels`.

- `ordered` : When set to `TRUE` in `factor()`, you indicate that the factor is ordered.
- `levels` : In this argument in `factor()`, you give the values of the factor in the correct order.

```
movie_length_ordered <- factor(
  movie_length,
  ordered = TRUE,
  levels = c("Very Short", "Short", "Medium", "Long", "Very Long")
)
movie_length_ordered
```

```
## [1] Very Short Short      Medium     Short      Long       Very Short Very Long
## Levels: Very Short < Short < Medium < Long < Very Long
```

Now, lets look at the summary of the ordered factor, `movie_length_ordered` :

```
summary(movie_length_ordered)
```

```
## Very Short      Short     Medium       Long  Very Long
##          2          2          1          1          1
```

7 Coding Exercise 4

In the code cell below, update the order of the movie_length factor from “Very Long” to “Very Short”.

```
# Write your code below. Don't forget to press Shift+Enter to execute the cell'
# movie_length_ordered <- factor(...)
# movie_length_ordered
```

Excellent! You have just completed the Vectors and Factors lab.

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 was written 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 RMD Version Created.