

Lab1.0-Vectors_Factors

May 18, 2021

VECTORS and FACTORS in R

0.0.1 Welcome!

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

0.1 Table of Contents

About the Dataset

Vectors

Vector Operations

Subsetting Vectors

Factors

Estimated Time Needed: 25 min

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
- **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

Here's what the data looks like:

Remember: To run the grey code cells in this exercise, click on the code cell, and then press Shift + Enter.

Vectors

Vectors are strings of numbers, characters or logical data (one-dimension array). In other words, a vector is a simple tool to store your grouped data.

In R, you create a vector with the combine function `c()`. You place the vector elements separated by a comma between the brackets. Vectors will be very useful in the future as they allow you to apply operations on a series of data easily.

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

0.1.1 Numeric, Character and Logical 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 `c()`.

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

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

```
[ ]: release_year
```

What if we use quotation marks? Then this becomes a **character vector**.

```
[ ]: # Create genre vector and assign values to it
titles <- c("Toy Story", "Akira", "The Breakfast Club")
titles
```

There are also **logical vectors**, which consist of TRUE's and FALSE's. They're particular important when you want to check its contents

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

[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.

Vector Operations

0.1.2 Adding more elements to a vector

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

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

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

0.1.3 Length of a vector

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

```
[ ]: release_year  
length(release_year)
```

0.1.4 Head and Tail of a vector

We can also retrieve just the **first few items** using the **head()** function:

```
[ ]: head(release_year) #first six items
```

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

```
[ ]: head(release_year, 2)
```

We can also retrieve just the **last few items** using the **tail()** function:

```
[ ]: tail(release_year) #last six items
```

```
[ ]: tail(release_year, 2) #last two items
```

0.1.5 Sorting a vector

We can also sort a vector:

```
[ ]: sort(release_year)
```

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

```
[ ]: sort(release_year, decreasing = TRUE)
```

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

```
[ ]: min(release_year)  
max(release_year)
```

0.1.6 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
```

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
```

0.1.7 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
```

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

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

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
```

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

```
[ ]: names(release_year)[1:3]
```

0.1.8 Summarizing Vectors

You can also use the “**summary**” function for simple descriptive statistics: minimum, first quartile, mean, third quartile, maximum:

```
[ ]: summary(cost_2014)
```

0.1.9 Using Logical Operations on Vectors

A vector can also be comprised of **TRUE** and **FALSE**, which are special **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 older than (**greater in value than**) 2000.

```
[ ]: movie_year <- 1997
movie_year > 2000
```

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
```

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

```
[ ]: movies_years == 2015 # is equal to 2015?
```

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

```
[ ]: movies_years != 2015
```

[Tip] Logical Operators in R

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

Checking equivalence: **1 == 2**

Checking non-equivalence: **TRUE != FALSE**

Greater than: **100 > 1**

Greater than or equal to: **100 >= 1**

Less than: **1 < 2**

Less than or equal to: **1 <= 2**

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
```

To retrieve the **second year**, you can use square brackets **[]**:

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

To retrieve the **third year**, you can use:

```
[ ]: movie_years[3]
```

And if you want to retrieve **multiple items**, you can pass in a vector:

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

Retrieving a vector without some of its items

To retrieve a vector without an item, 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]
```

You can save the new vector using a variable:

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

**** 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
```

[Tip] Checking NA in R

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

Check if NA: **is.na(NA)**

Check if not NA: **!is.na(2)**

0.1.10 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
```

To retrieve the actual movie years after year 2000, you can simply subset the vector using the logical vector within **square brackets “[]”**:

```
[ ]: release_year[release_year > 2000] #returns a vector for elements that returned TRUE
↪ TRUE for the condition
```

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
release_year[c(T, F, F, F)] #returns the values that are TRUE
```

Factors

Factors are variables in R which take on a limited number of different values; such variables are often referred to as **categorical variables**. The difference between a categorical variable and a continuous variable is that a categorical variable can belong to a limited number of categories.

A continuous variable, on the other hand, can correspond to an infinite number of values. For example, the height of a tree is a continuous variable, but the titles of books would be a categorical variable.

One of the most important uses of factors is in statistical modeling; since categorical variables enter into statistical models differently than continuous variables, storing data as factors insures that the modeling functions will treat such data correctly.

Let's start with a *vector* of genres:

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

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 **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)
```

0.1.11 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)
```

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

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

0.1.12 Ordered factors

There are two types of categorical variables: a **nominal categorical variable** and an **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 *Comedy*, *Animation*, *Crime*, *Comedy*, *Animation*. 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
```

`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 **ordered 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
```

Now, let's look at the summary of the ordered factor, `factor_mvlength_vector`:

```
[ ]: summary(movie_length_ordered)
```

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.

0.1.13 About the Author:

Hi! It's [Helly Patel](#), the author of this notebook. I hope you found R easy to learn! There's lots more to learn about R but you're well on your way. 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](#).