Lesson notes | Data structures

GRAPH Network & WHO, supported by the Global Fund to fight HIV, TB & Malaria

April 2024

This lesson was created by the GRAPH Network, a non-profit headquartered at the University of Geneva Global Health Institute, in collaboration with the World Health Organization, under a Global Fund 2023 grant to create e-learning modules to build in-country data capacity for epidemiological data analysis

Intros

In this lesson, we'll take a brief look at data structures in R. Understanding data structures is crucial for data manipulation and analysis. We will start by exploring vectors, the basic data structure in R. Then, we will learn how to combine vectors into data frames, the most common structure for organizing and analyzing data.

Learning objectives

1. You can create vectors with the `c()` function.
2. You can combine vectors into data frames.
3. You understand the difference between a tibble and a data frame.

Packages

Please load the packages needed for this lesson with the code below:

```
if(!require(pacman)) install.packages("pacman")
pacman::p_load(tidyverse)
```

Introducing vectors

The most basic data structures in R are vectors. Vectors are a collection of values that all share the same class (e.g., all numeric or all character). It may be helpful to think of a

vector as a column in an Excel spreadsheet.

Creating vectors

Vectors can be created using the `c()` function, with the components of the vector separated by commas. For example, the code `c(1, 2, 3)` defines a vector with the elements 1, 2 and 3.

In your script, define the following vectors:

```
age <- c(18, 25, 46)
sex <- c('M', 'F', 'F')
positive_test <- c(T, T, F)
id <- 1:3 # the colon creates a sequence of numbers
```

You can also check the classes of these vectors:

```
class(age)
```

```
## [1] "numeric"
```

```
class(sex)
```

```
## [1] "character"
```

```
class(positive_test)
```

```
## [1] "logical"
```

Each line of code below tries to define a vector with three elements but has a mistake. Fix the mistakes and perform the assignment.

```
my_vec_1 <- (1,2,3)
my_vec_2 <- c("Obi", "Chika" "Nonso")
```

The individual values within a vector are called *components* or elements. So the vector `c(1, 2, 3)` has three components/elements.

Manipulating vectors

Many of the functions and operations you have encountered so far in the course can be applied to vectors.

For example, we can multiply our age object by 2:

```
age
```

```
## [1] 18 25 46
```

```
age * 2
```

```
## [1] 36 50 92
```

Notice that every element in the vector was multiplied by 2.

Or, below we take the square root of age:

```
age
```

```
## [1] 18 25 46
```

```
sqrt(age)
```

```
## [1] 4.242641 5.000000 6.782330
```

You can also can add (numeric) vectors to each other:

```
age + id
```

```
## [1] 19 27 49
```

Note that the first element of age is added to the first element of id and the second element of age is added to the second element of id and so on.

From vectors to data frames

Now that we have a handle on creating vectors, let's move on to the most commonly used object in R: data frames. A data frame is just a collection of vectors of the same length with some helpful metadata. We can create one using the `data.frame()` function.

We previously created vector variables (id, age, sex and positive_test) for three individuals:

We can now use the `data.frame()` function to combine these into a single tabular structure:

```
data_epi <- data.frame(id, age, sex, positive_test)
data_epi
```

```
##   id age sex positive_test
## 1  1  18   M          TRUE
## 2  2  25   F          TRUE
## 3  3  46   F         FALSE
```

Note that instead of creating each vector separately, you can create your data frame defining each of the vectors inside the `data.frame()` function.

```
data_epi_2 <- data.frame(age = c(18, 25, 46),
                         sex = c('M', 'F', 'F'))

data_epi_2
```

```
##   age sex
## 1  18   M
## 2  25   F
## 3  46   F
```

SIDE NOTE

Most of the time you work with data in R, you will be importing it from external contexts. But it is sometimes useful to create datasets *within* R itself. It is in such cases that the `data.frame()` function will come in handy.

To extract the vectors back out of the data frame, use the $ syntax. Run the following lines of code in your console to observe this.

```r
data_epi$age
is.vector(data_epi$age) # verify that this column is indeed a vector
class(data_epi$age) # check the class of the vector
```

Combine the vectors below into a data frame, with the following column names: "name" for the character vector, "number_of_children" for the numeric vector and "is_married" for the logical vector.

```r
character_vec <- c("Bob", "Jane", "Joe")
numeric_vec <- c(1, 2, 3)
logical_vec <- c(T, F, F)
```

Use the `data.frame()` function to define a data frame in R that resembles the following table:

room	num_windows
dining	3
kitchen	2
bedroom	5

Tibbles

The default version of tabular data in R is called a data frame, but there is another representation of tabular data provided by the *tidyverse* package. It's called a `tibble`, and it is an improved version of the data frame.

You can convert from a data frame to a tibble with the `as_tibble()` function:

```r
data_epi
```

```
##   id age sex positive_test
## 1  1  18   M          TRUE
## 2  2  25   F          TRUE
## 3  3  46   F         FALSE
```

```r
tibble_epi <- as_tibble(data_epi)
tibble_epi
```

```
## # A tibble: 3 × 4
##       id   age sex   positive_test
##    <int> <dbl> <chr> <lgl>
## 1     1    18 M     TRUE
```

```
## 2     2    25 F     TRUE
## 3     3    46 F     FALSE
```

Notice that the tibble gives the data dimensions in the first line:

```
👉# A tibble: 3 × 4👈
     id   age sex   positive_test
  <int> <dbl> <chr> <lgl>
1     1    18 M     TRUE
2     2    25 F     TRUE
3     3    46 F     FALSE
```

And also tells you the data types, at the top of each column:

```
# A tibble: 3 × 4
     id   age sex   positive_test
👉   <int> <dbl> <chr> <lgl> 👈
1     1    18 M     TRUE
2     2    25 F     TRUE
3     3    46 F     FALSE
```

There, "int" stands for integer, dbl" stands for double (which is a kind of numeric class), "chr" stands for character, and "lgl" for logical.

The other benefit of tibbles is they avoid flooding your console when you print a long table.

Consider the console output of the lines below, for example:

```
# print the infert data frame (a built in R dataset)
infert # Veryyy long print
as_tibble(infert) # more manageable print
```

For your most of your data analysis needs, you should prefer tibbles over regular data frames.

`read_csv()` creates tibbles

When you import data with the `read_csv()` function from {readr}, you get a tibble:

```
ebola_tib <- read_csv("https://tinyurl.com/ebola-data-sample") # Needs
internet to run
class(ebola_tib)
```

```
## [1] "spec_tbl_df" "tbl_df"      "tbl"         "data.frame"
```

But when you import data with the base `read.csv()` function, you get a data.frame:

```
ebola_df <- read.csv("https://tinyurl.com/ebola-data-sample") # Needs internet
to run
class(ebola_df)
```

```
## [1] "data.frame"
```

Try printing `ebola_tib` and `ebola_df` to your console to observe the different printing behavior of tibbles and data frames.

This is one reason we recommend using `read_csv()` instead of `read.csv()`.

Wrap-up

With your understanding of data classes and structures, you are now well-equipped to perform data manipulation tasks in R. In the upcoming lessons, we will explore the powerful data transformation capabilities of the dplyr package, which will further enhance your data analysis skills.

Congratulations on making it this far! You have covered a lot and should be proud of yourself.

Solutions

Solution to the first r-practice block:

```
my_vec_1 <- c(1,2,3) # Use 'c' function to create a vector
my_vec_2 <- c("Obi", "Chika", "Nonso") # Separate each string with a comma
```

Solution to the second r-practice block:

```
df <- data.frame(name = character_vec,
                 number_of_children = numeric_vec,
                 is_married = logical_vec)
```

Solution to the third r-practice block:

```
# Solution to the third r-practice block
rooms <- data.frame(room = c("dining", "kitchen", "bedroom"),
                    num_windows = c(3, 2, 5))
```

Contributors

The following team members contributed to this lesson:

DANIEL CAMARA

Data Scientist at the GRAPH Network and fellowship as Public Health researcher at Fiocruz, Brazil
Passionate about lots of things, especially when it involves people leading lives with more equality and freedom

EDUARDO ARAUJO

Student at Universidade Tecnologica Federal do Parana
Passionate about reproducible science and education

LAURE VANCAUWENBERGHE

Data analyst, the GRAPH Network
A firm believer in science for good, striving to ally programming, health and education

KENE DAVID NWOSU

Data analyst, the GRAPH Network
Passionate about world improvement

References

Some material in this lesson was adapted from the following sources:

- Wickham, H., & Grolemund, G. (n.d.). *R for data science*. 15 Factors | R for Data Science. Accessed October 26, 2022. https://r4ds.had.co.nz/factors.html.