R Crash Course

Dr. F.J. Rodenburg

Contents

Introduction

This one-week course will improve your programming to the level required to pass General Research Skills. Each day consists of about half an hour of explanation and 1–2 hours of assignments.

Structure of This Course

1. Installation and `swirl`
2. Base graphics using `plot`
3. Specialized graphics using `ggplot2`
4. Reading and manipulating data
5. Creating a Minimal Working Example

The `swirl` assignment on day 1 is a great way to get used to the basic syntax (like remembering to close brackets). But if we build up your programming skill all the way from the fundamentals, explaining every step along the way, there is no way you could do anything remotely useful after only a week. Also, it would be excruciatingly boring.

Hence, every day after the first, we will begin with an example of something useful you can do in R. I will explain how it works and then it is your job to edit the code provided and make it do something else. In the beginning that may feel a lot like you are just copy-pasting the example. But the more you practice, the more you will feel comfortable actually changing code. At some point, you may even know a fair few functions by heart, but even if you don't, always keep in mind: It is perfectly fine to search online.

Learning a programming language is similar to learning any other language: You could learn long lists of words and endless rules about grammar (as is unfortunately how language is usually taught in school), but this will never make you fluent in a language, let alone in one week. To speak the language, it is *much* more efficient to start by learning some useful phrases and building your knowledge from there. The same goes with programming in R: If you know how to make a certain figure, then who cares that you don't know exactly how

it works, or what exactly happens under the hood? If you are able to adjust the code to work with new data, then you can make that figure for any data set!

Examination

This course has no credits, exam or grade. However, during Of course, the things you pick up during this course can be used during the graded assignment of General Research Skills, which is also in R.

In the live version of this course, if time permits, we will go over some student solutions at the end of each day and discuss what works, what doesn't and why.

 This course is a work in progress. Please check back later for updates. For students at Leiden University, always consult Brightspace for the latest information on the schedule, the course material and the exam.

Chapter 1

Installation & `swirl`

The goal for this day is to install and setup RStudio and to verify your installation is working correctly.

Total video length: 20 min.
Exercise length: 30–90 min.

In order for you not to get stuck on errors, it is vital that RStudio is installed correctly. I recommend a clean install even if you already have the software.

1.1 Installation Guide

Watch the video below and install RStudio:

```
## PhantomJS not found. You can install it with webshot::install_phantomjs(). If it is installed,
```

Note: TinyTeX is not required for this course. All you need is a working version of RStudio, and to be able to knit to HTML.

1.2 What is RStudio?

Watch the video below and change the options as suggested (e.g. disable inline output):

1.3 Exercises

If you are following the live version of the course, make sure you address installation issues today. From day 2 onward, you are expected to have a working version of RStudio. You can ask any of the assistants or teachers for help.

Once you have RStudio installed, start with `swirl`:

1. Install `swirl` as explained in the video below (last part);
2. Choose the R Programming course;
3. Complete chapters 1, 2, 4, 7 & 8. Each take about 5–15 minutes.

If you have time left and want to prepare for the rest of the course, please try running the following in the console (copy it, paste it in the console and press enter):

```
install.packages("COVID19")
install.packages("ggplot2")
install.packages("rgeos")
install.packages("rnaturalearth")
install.packages("rnaturalearthdata")
install.packages("sf")
```

These packages will be used during the rest of the course. If you have installation problems, we can address them early on.

Chapter 2

Base Graphics

Producing figures using the built-in functions in R.

Total video length: 9 min.
Exercise length: 30–60 min.

An easy way to see how R works is by making plots.

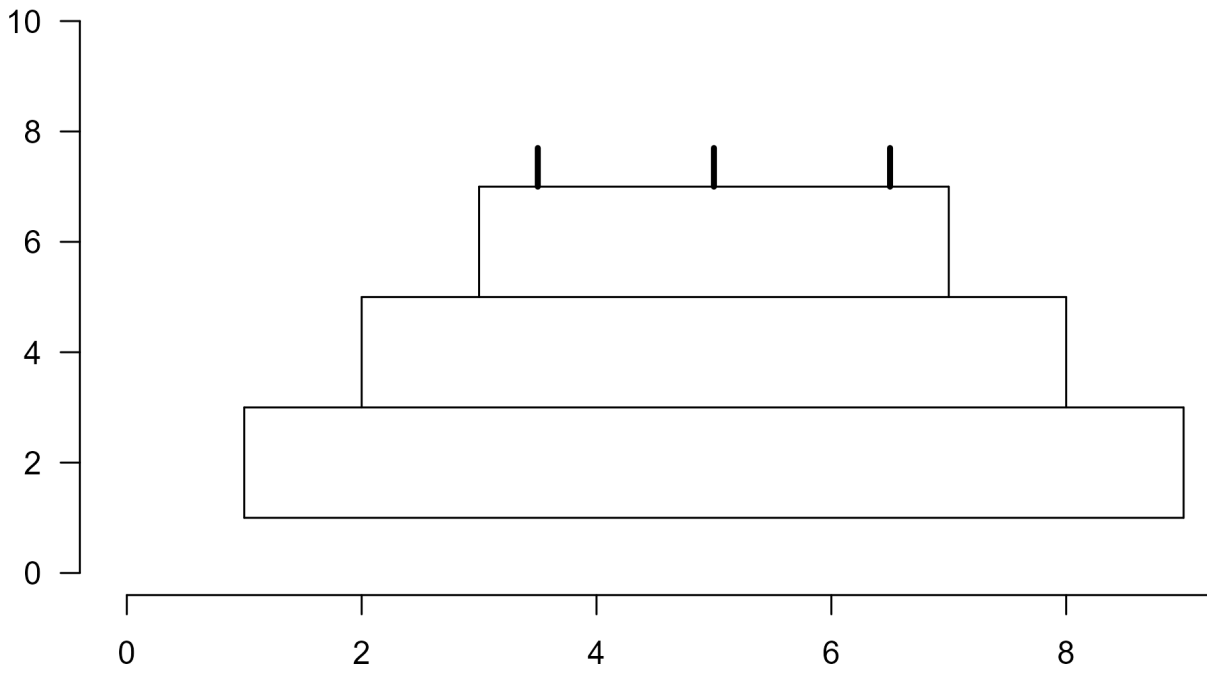
2.1 Exercise (easy)

Today is almost certainly someone's birthday. Let's make them a cake!

Below is an example of a simple drawing in base R:

```
plot(NA, xlim = c(0, 10), ylim = c(0, 10), xlab = "", ylab = "", las = 1, bty = "n")

# Level 1
## Horizontal lines
lines(c(1, 9), c(1, 1))
lines(c(1, 9), c(3, 3))
## Vertical lines
lines(c(1, 1), c(1, 3))
lines(c(9, 9), c(1, 3))

# Level 2
## Horizontal line
lines(c(2, 8), c(5, 5))
```

```
## Vertical lines
lines(c(2, 2), c(3, 5))
lines(c(8, 8), c(3, 5))

# Level 3
## Horizontal lines
lines(c(3, 7), c(7, 7))
## Vertical lines
lines(c(3, 3), c(5, 7))
lines(c(7, 7), c(5, 7))

# Candles
k <- 3                              # number of candles
x <- seq(3.5, 6.5, length.out = k) # position of the candles
h <- 0.7                            # height of the candles
segments(x0 = x, x1 = x, y0 = rep(7, k), y1 = rep(7 + h, k), lwd = 3)
```



1. Create a new R markdown file and copy the script to a new R chunk.
2. Run the entire chunk using the green play button on the top right of the chunk. This will generate the figure.
3. To see which line does what, run each line from top to bottom by clicking anywhere in the first line of code and pressing CTRL+Enter (Windows) or Command+Enter (MacOS). If this does not work, check the following:

- Do you have inline output disabled? Inline output may look convenient, but it usually causes more problems than it solves.
- Did you run the code in order?[1]

4. Now that you have seen what each line does, change the script to render 5 candles instead of 3.
5. Can you change the code to add a layer to the cake? *(You may have to make the other layers smaller, or move the candles up.)*

2.2 Exercise (medium)

Below is a video on how to read help files as a starting point for this part:

Alternatively, look for explanations online. This might feel like cheating, but getting good at searching how to do things in R is actually a valuable skill.

1. Using `polygon`, color your layered cake with at least three different colors. You can view the help file by typing `?polygon` in the console and pressing enter.
2. Now that everything is in place, let's add the title "Happy Birthday" and hide the axes. It is up to you to find out how, using whatever resource you want.

2.3 Exercise (hard)

Now let's add sprinkling to the cake. We could tell R exactly where to draw the sprinkling, but nobody sprinkles like that. So let's make it a bit more realistic:

1. Use the function `runif` to randomly draw x-positions for the candles from a uniform distribution. Look up how to use the function by entering `?runif` in the console, or by searching online.
2. Every time you run the script, a new set of random coordinates will be chosen for your sprinkling. To make your cake reproducible, set a seed with the `set.seed` function. You can search online, or in the help files how this function works, and where you should place it in your script.
3. If you have time left, personalize your cake with base R functions however you see fit!

[1] The function `lines` adds a line to an existing plot. This is only possible if you run `plot` first, so run the code in order, from top to bottom.

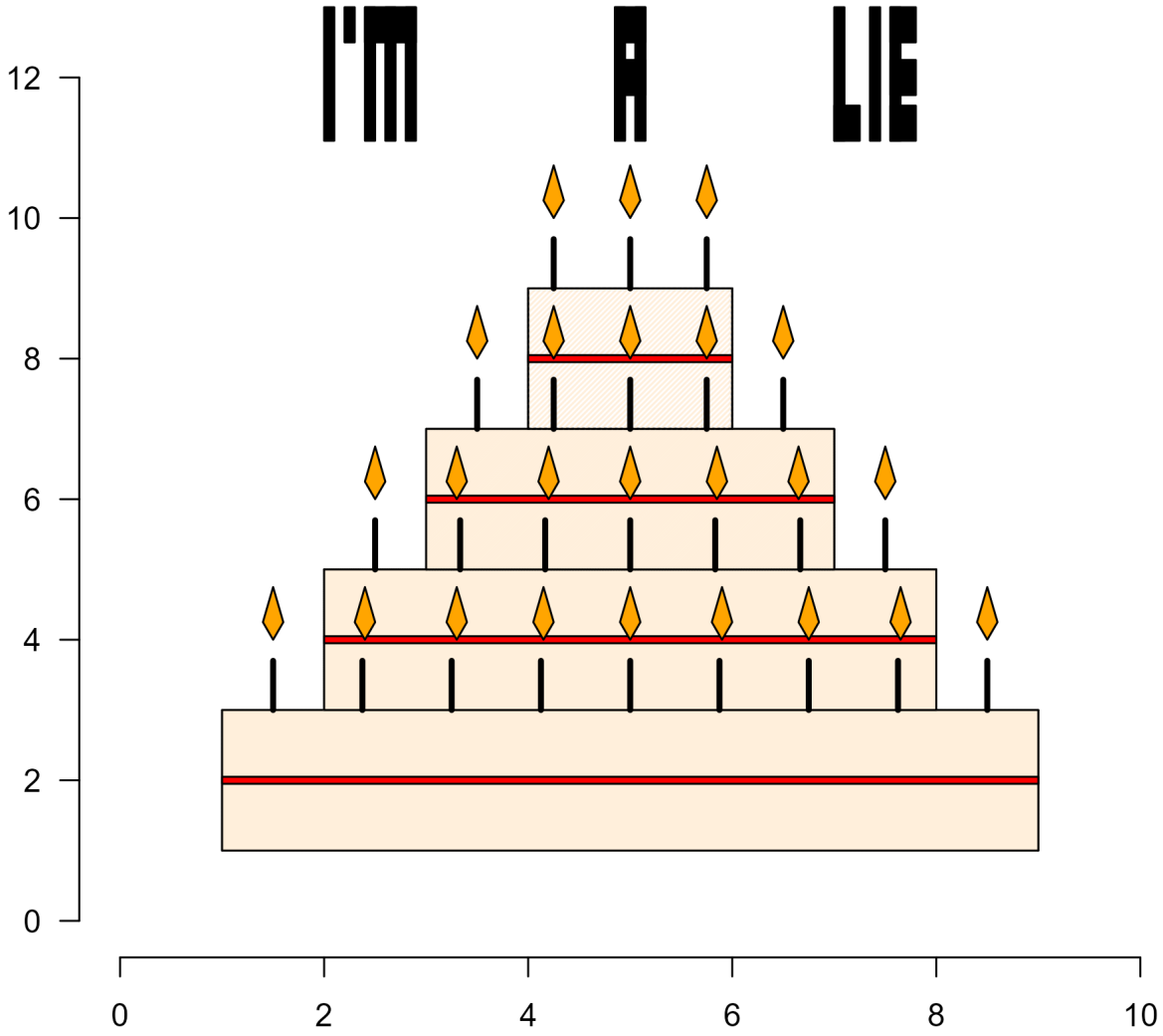
2.4 Exercise (`swirl`)

Done drawing your cake? Want a bit more explanation of how base graphics actually work? Complete the `swirl` R Programming course, chapter 15.

Hall of Fame

The best looking cake in R, using only base R functions will be displayed here for future visitors! Your name will not be included unless you consent.

R Crash Course (2022)



Watch the discussion here.

Download the source code here.

Advanced Statistics (2021)

Watch the discussion here.

Download the source code here.

Chapter 3

Graphics with ggplot2

A demonstration of one of the many specialized plotting routines offered by `ggplot2`.

Total video length: … min.
Exercise length: 30–90 min.

Can you plot the number of COVID-19 infections across the world on a specific day using just `plot`, `lines`, `points`, `text` and `polygon`? Of course… but it would probably take a very long time. If you install the following specialized packages:

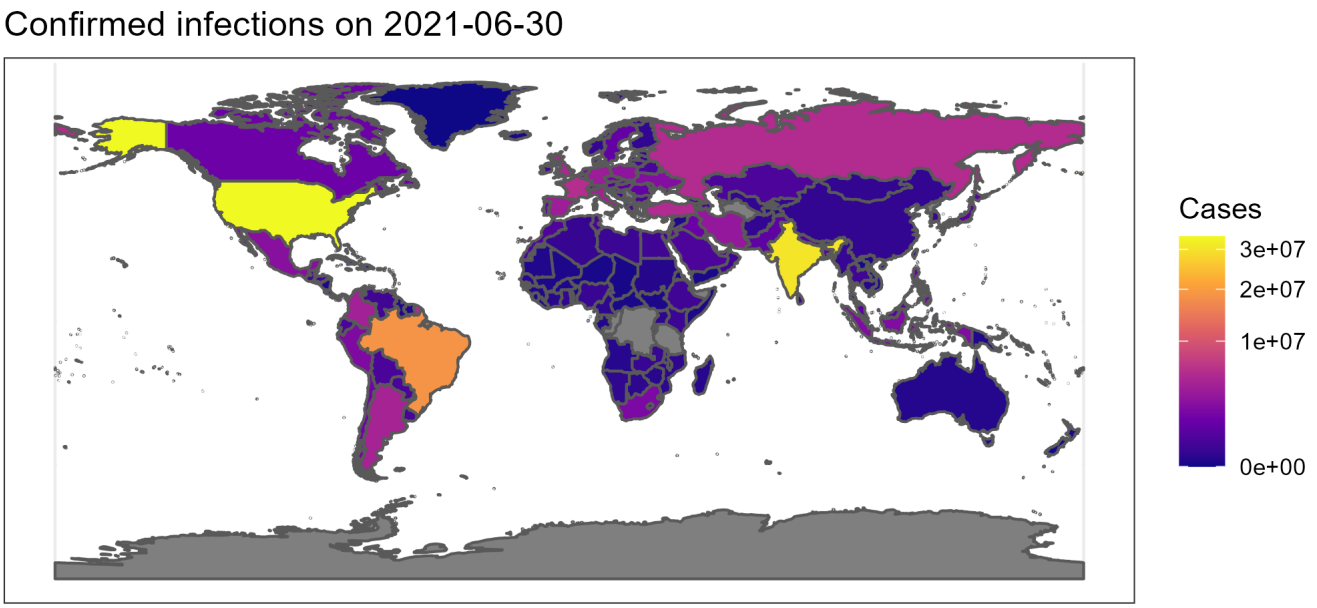
```
install.packages("COVID19")
install.packages("ggplot2")
install.packages("rgeos")
install.packages("rnaturalearth")
install.packages("rnaturalearthdata")
install.packages("sf")
```

Here is how you could do that with `ggplot2`:

```
# Load package required for easy access to COVID-19 data
library("COVID19")

# Load packages required for easy access to Earth data
library("rnaturalearth")
library("rnaturalearthdata")
library("sf")
```

```r
# Load ggplot2 and set background to white
library("ggplot2")
theme_set(theme_bw())

# Load Earth data from rnaturalearthdata package
Earth <- ne_countries(scale = "medium", returnclass = "sf")

# Load COVID-19 data for a specific date
COVID19 <- covid19()
maxc    <- max(COVID19$confirmed, na.rm = TRUE) # maximum number of cases
date    <- "2021-06-30"
Cases   <- COVID19[COVID19$date == date, ]

# Data cleaning
countries                                <- Cases$administrative_area_level_1
countries[countries == "United States"] <- "United States of America"
countries[countries == "Korea, South"]  <- "South Korea"
countries[countries == "Cote d'Ivoire"] <- "Ivory Coast"

# Match the confirmed cases to the list of countries in the Earth data
wh          <- match(Earth$admin, countries)
Earth$Cases <- Cases$confirmed[wh]

# Plot the data
ggplot(data = Earth) + geom_sf(aes(fill = Cases)) +
  scale_fill_viridis_c(limits = c(0, maxc), option = "plasma", trans = "sqrt") +
  ggtitle(paste("Confirmed infections on", date))
```

Provided you have the latest version of R installed and the required packages were installed correctly, this will return the following:



Using less than 50 lines of code, we now have a script that generates a world map colored by the number of infections on a particular day.

3.1 Exercises

If you get stuck on any of the exercises, you can watch the video below to get unstuck. In case of installation problems, contact one of the supervisors or assistants, or work together with a student who has successfully installed the packages.

1. Install the required packages and run the script to reproduce the figure.
2. Look at the legend. What does "3e+07" mean?
3. Read through the script. Can you change the figure to show the number of infections one month earlier? *(Hint: Only one line in the code needs to be changed.)*
4. Delete `, trans = "sqrt"` from the script and run it again. What did this part do, and why would that be desirable? *(You may restore the original script after this.)*
5. Delete the entire second to last line from the script. What did this line do? *(You may restore the original script after this.)*
6. Essentially, `ggplot` works like this: `ggplot(dataset) + whattoshow() + extraoption1() + extraoption2 + ...` Add the following to the

`ggplot` chain in the script: `+ coord_sf(xlim = c(-10, 35), ylim = c(35, 70))` Run the script again. What does this do?

7. Can you change the `xlim` and `ylim` values from the previous question to center the map around Africa?
8. Go to the github page for the `viridis` package. What is the point of this package?
9. Can you change the `ggplot` part of our script to use a different color scheme? *(Hint: We are already using one of the viridis color schemes, so you only have to change something, not add anything new.)*
10. If you click on `Earth` in the environment pane, or if you run `View(Earth)` in the console, you can see all the variables stored in the `Earth` data set. Can you change the original script to show the population estimates instead of the number of infections?

[**video explaining the script to be included**]

3.2 Extra: Data Cleaning (*)

If you have time left, you can try and complete the additional exercises here.

The COVID-19 and Earth data sets use different names for countries. So how did I know what names to replace with what?

You could simply print both lists of countries and manually look for names that are written differently in both data sets:

```
# Country names in the Earth data set
Earth$admin

# Country names in the COVID-19 data set
countries
```

But these commands will return lists of over 200 countries… A much easier way is to simply look at which countries were unavailable after the matching step. For example:

```
countries <- Cases$administrative_area_level_1
wh        <- match(Earth$admin, countries)
Earth$admin[is.na(wh)]
```

```
##  [1] "Aruba"
##  [2] "Anguilla"
##  [3] "Aland"
##  ...  (49 countries omitted for brevity)
```

```
## [53] "United States of America"
## [54] "Vatican"
## [55] "British Virgin Islands"
## [56] "United States Virgin Islands"
## [57] "Wallis and Futuna"
```

This gives us a list of *only* the countries in the `Earth` data set that were *not* matched. Here we see that, for instance, `"United States of America"` was not matched to the COVID-19 data, even though we know there were large number of cases in the US.

The next step is to see how this country is called in the other data set. Again, you could just look at all the names in the variable `countries`, but how about we only check countries which name contains a capital letter "U":

```
# Countries in the COVID-19 data set with a capital letter U
countries[grep("U", countries)]
```

```
## [1] "United Kingdom"           "Virgin Islands, U.S."
## [3] "United States of America" "Uruguay"
## [5] "Uganda"                   "Uzbekistan"
## [7] "United Arab Emirates"     "Ukraine"
```

```
# Countries in the Earth data set with a capital letter U
Earth$admin[grep("U", Earth$admin)]
```

```
## [1] "United Arab Emirates"         "United Kingdom"
## [3] "United Republic of Tanzania"  "Uganda"
## [5] "Ukraine"                      "Uruguay"
## [7] "United States of America"     "Uzbekistan"
## [9] "United States Virgin Islands"
```

Here we see the problem: In the COVID-19 data set, the US is called `"United States"`, whereas in the `Earth` data set it is called `"United States of America"`.

The solution is to overwrite the name so they are both the same:

```
countries[countries == "United States"] <- "United States of America"
```

And like that, you can check all the unmatched countries for perhaps other names they might be known by.

Here is another example:

```
# Countries containing "Congo" in the COVID-19 data set
countries[grep("Congo", countries)]
```

```
## [1] "Congo"
## [2] "Congo, the Democratic Republic of the"
```

```
# Countries containing "Congo" in the Earth data set
Earth$admin[grep("Congo", Earth$admin)]
```

```
## [1] "Democratic Republic of the Congo" "Republic of Congo"
```

1. As you can see above, the two Congos have different names in either data set. Read the original script carefully. Add what you think is necessary to color the two Congos as well. To see if it works, simply run the script again to produce the (new) figure.
2. Using the method described above, can you color any other countries which are now grey? *(Hint: Some countries do not have any data on confirmed cases, like Turkmenistan and North Korea. You can leave these uncolored.)*

Chapter 4

Reading & Manipulating Data

An introduction to storing and converting data using the tidy format.

Total video length: 4 min.
Exercise length: 30–90 min.

4.1 Reading Data

There are many ways to read data into R. A simple method is explained in the video below:

4.2 What Is Tidy Data?

A standardized way to store data is by having **rows represent observations** and **columns represent variables**.

For example, take a look at the first few rows of the `iris` data set:

Tidy data. Every row is a single flower. Every column is a single variable.

Sepal.Length

Sepal.Width

Petal.Length

Petal.Width
Species
5.1
3.5
1.4
0.2
setosa
4.9
3.0
1.4
0.2
setosa
4.7
3.2
1.3
0.2
setosa
4.6
3.1
1.5
0.2
setosa
5.0
3.6
1.4
0.2
setosa
5.4
3.9
1.7
0.4
setosa

4.3 What Is Untidy Data?

Sometimes it is tempting to use a different format. For example, here are the first few rows of a fictional qPCR data set performed in triplicates:

Untidy data. A single row contains three different experimental observations.

Gene

Treatment

Ct value in exp. 1

Ct value in exp. 2

Ct value in exp. 3

A

control

22.8

16.0

17.2

B

control

22.2

15.7

19.5

C

control

23.0

20.4

23.7

D

control

14.5

18.9

18.9

E

control

17.6

22.2

17.5

A

ADH-/-

20.8

15.6

20.5

This may look like a more intuitive way to store your data, but most statistical software will not know what to do with it. Every row should contain a single gene's C_t-value from a single replicate. You can view the correct way to store it here.

Storing data in tidy format will greatly simplify analysis and almost everything we use in the GRS course expects data to be in this format.

4.4 Exercise (easy)

1. Watch the video on how to read data in R;
2. Download this Excel file;
3. Read the data into R as explained in step 1;
4. Convert the data set to tidy format using R code.

There are different ways to complete this exercise. For example you could use the dollar sign (`$`) or square brackets (`[ , ]`) to indicate which column you want.

5. Run a *t*-test to compare male and female height. See the help page for `t.test`. If you don't remember which *t*-test to use when, you may use the default;
6. Report your conclusion at a significance level of 0.05.

HINT: The following are useful functions you might use for this: `c` (combine/concatenate), `rep` (repeat/replicate), `rbind` (rowbind), `cbind` (column bind). Use the help files, or Google if needed to find out how they work.

4.5 Exercise (medium)

The following Google Sheets file contains shows three versions of the same data set. The tab "tidy" is how I wish people sent me data. The tab "untidy" is what students often send me after at least trying to put it in the right format. The tab "headache" is closer to what people send me for consultancy.

1. Open the Google Sheets file;
2. Download the "untidy" tab as a .csv file and read it into R;
3. Convert the file to tidy format;
4. Compare the resulting data frame with the "tidy" tab, to see if you got it right.

HINT: The following are useful functions: `c` (combine/concatenate), `rep` (repeat/replicate), `rbind` (rowbind), `cbind` (column bind). Use the help files, or Google if needed to find out how they work.

4.6 Exercise (hard)

These two toy examples are very small data sets, so you might argue that it would be simpler to just fix them in Excel (I wouldn't disagree). However, if these were several dozen, or even hundreds of genes, then copy-pasting is dangerous, error-prone business.

1. Open the Google Sheets file;
2. Download the "headache" tab as a .csv file and read it into R;
3. Convert the file to tidy format;
4. Compare the resulting data frame with the "tidy" tab, to see if you got it right;
5. Using the internet and any other resource you like, try plotting the different gene/treatment combinations in a single plot;
6. Bonus points if you can complete this exercise without developing a headache.

Chapter 5

Debugging

What do you do when your script returns a warning or error? Where do you find relevant information? How do you write a good question about programming with a minimal working example?

Total video length: … min.
Exercise length: …–… min.

Getting stuck on a programming problem can be very frustrating. A minimal working example (MWE) is a script that contains **only** the necessary code and data for others to reproduce your error. If you add an MWE to your question, you are just about guaranteed to get better, and quicker answers to your coding questions.

5.1 Useful information:

- `dput`: Use this function save your data set into code. If someone runs this code, the output will be your data set. Try for instance `dput(iris)`.
- `set.seed`: Use this function before you simulate data to initialize the random number generator. Anyone who runs your code with the same seed as you will end up with exactly the same data.
- Google's (old) R style guide.[1]
- …

[1]Their new style guide focuses on the `tidyverse`. The `tidyverse` is essentially a way to make code look 'cleaner', but for an absolute beginner in R, it is another layer of complexity, as you will still need to use base R code in between. Hence why it is not a part of this course.

Self-Test

Test your R knowledge by answering the following questions.

...