8 Workflow: getting help

This book is not an island; there is no single resource that will allow you to master R. As you begin to apply the techniques described in this book to your own data, you will soon find questions that we do not answer. This section describes a few tips on how to get help and to help you keep learning.

8.1 Google is your friend

If you get stuck, start with Google. Typically adding "R" to a query is enough to restrict it to relevant results: if the search isn't useful, it often means that there aren't any R-specific results available. Additionally, adding package names like "tidyverse" or "ggplot2" will help narrow down the results to code that will feel more familiar to you as well, e.g., "how to make a boxplot in R" vs. "how to make a boxplot in R with ggplot2". Google is particularly useful for error messages. If you get an error message and you have no idea what it means, try googling it! Chances are that someone else has been confused by it in the past, and there will be help somewhere on the web. (If the error message isn't in English, run `Sys.setenv(LANGUAGE = "en")` and re-run the code; you're more likely to find help for English error messages.)

If Google doesn't help, try Stack Overflow. Start by spending a little time searching for an existing answer, including `[R]`, to restrict your search to questions and answers that use R.

8.2 Making a reprex

If your googling doesn't find anything useful, it's a really good idea to prepare a **reprex,** short for minimal **repr**oducible **ex**ample. A good reprex makes it easier for other people to help you, and often you'll figure out the problem yourself in the course of making it. There are two parts to creating a reprex:

- First, you need to make your code reproducible. This means that you need to capture everything, i.e. include any `library()` calls and create all necessary objects. The easiest way to make sure you've done this is using the reprex package.

- Second, you need to make it minimal. Strip away everything that is not directly related to your problem. This usually involves creating a much smaller and simpler R object than the one you're facing in real life or even using built-in data.

That sounds like a lot of work! And it can be, but it has a great payoff:

- 80% of the time, creating an excellent reprex reveals the source of your problem. It's amazing how often the process of writing up a self-contained and minimal example allows you to answer your own question.

- The other 20% of the time, you will have captured the essence of your problem in a way that is easy for others to play with. This substantially improves your chances of getting help!

When creating a reprex by hand, it's easy to accidentally miss something, meaning your code can't be run on someone else's computer. Avoid this problem by using the reprex package, which is installed as part of the tidyverse. Let's say you copy this code onto your clipboard (or, on RStudio Server or Cloud, select it):

```r
y <- 1:4
mean(y)
```

Then call `reprex()`, where the default output is formatted for GitHub:

```r
reprex::reprex()
```

A nicely rendered HTML preview will display in RStudio's Viewer (if you're in RStudio) or your default browser otherwise. The reprex is automatically copied to your clipboard (on RStudio Server or Cloud, you will need to copy this yourself):

````
``` r
y <- 1:4
mean(y)
#> [1] 2.5
```
````

This text is formatted in a special way, called Markdown, which can be pasted to sites like StackOverflow or Github and they will automatically render it to look like code. Here's what that Markdown would look like rendered on GitHub:

```r
y <- 1:4
mean(y)
#> [1] 2.5
```

Anyone else can copy, paste, and run this immediately.

There are three things you need to include to make your example reproducible: required packages, data, and code.

1. **Packages** should be loaded at the top of the script so it's easy to see which ones the example needs. This is a good time to check that you're using the latest version of each package; you may have discovered a bug that's been fixed since you installed or last updated the package. For packages in the tidyverse, the easiest way to check is to run `tidyverse_update()`.

2. The easiest way to include **data** is to use `dput()` to generate the R code needed to recreate it. For example, to recreate the `mtcars` dataset in R, perform the following steps:

    1. Run `dput(mtcars)` in R
    2. Copy the output
    3. In reprex, type `mtcars <-`, then paste.

    Try to use the smallest subset of your data that still reveals the problem.

3. Spend a little bit of time ensuring that your **code** is easy for others to read:

    - Make sure you've used spaces and your variable names are concise yet informative.
    - Use comments to indicate where your problem lies.
    - Do your best to remove everything that is not related to the problem.

> The shorter your code is, the easier it is to understand and the easier it is to fix.

Finish by checking that you have actually made a reproducible example by starting a fresh R session and copying and pasting your script.

Creating reprexes is not trivial, and it will take some practice to learn to create good, truly minimal reprexes. However, learning to ask questions that include the code, and investing the time to make it reproducible will continue to pay off as you learn and master R.

8.3 Investing in yourself

You should also spend some time preparing yourself to solve problems before they occur. Investing a little time in learning R each day will pay off handsomely in the long run. One way is to follow what the tidyverse team is doing on the tidyverse blog. To keep up with the R community more broadly, we recommend reading R Weekly: it's a community effort to aggregate the most interesting news in the R community each week.

8.4 Summary

This chapter concludes the Whole Game part of the book. You've now seen the most important parts of the data science process: visualization, transformation, tidying and importing. Now you've got a holistic view of the whole process, and we start to get into the details of small pieces.

The next part of the book, Visualize, does a deeper dive into the grammar of graphics and creating data visualizations with ggplot2, showcases how to use the tools you've learned so far to conduct exploratory data analysis, and introduces good practices for creating plots for communication.

R for Data Science (2e) was written by Hadley Wickham, Mine Çetinkaya-Rundel, and Garrett Grolemund. This book was built with Quarto.