Data Science for Biologists - 5023Y

Philip Leftwich

2021-09-27

Contents

Chapter 1

Introduction

1.1 Approach and style

This book is designed to accompany the module BIO-5023Y for those new to R looking for best practices and tips. So it must be both accessible and succinct. The approach here is to provide just enough text explanation that someone very new to R can apply the code and follow what the code is doing. It is not a comprehensive textbook.

A few other points:

This is a code reference book accompanied by relatively brief examples - not a thorough textbook on R or data science

This is intended to be a living document - optimal R packages for a given task change often and we welcome discussion about which to emphasize in this handbook

Top tips for the course:

DON'T worry if you don't understand everything

DO ask lots of questions!

1.2 Teaching

We have:

- one lecture per week
- one workshop per week

These are both timetabled in-person sessions, and you should check Timetabler for up to-date information on scheduling. However, all lessons can be accessed

remotely through Collaborate, and **everything** you need to complete workshops will be available on this site.

If you feel unwell, or cannot attend a session in-person because you need to self-isolate then don't worry you can access everything, and follow along in real time, or work at your own pace.

Questions/issues/errors can all be posted on our Yammer page.

1.2.1 Workshops

The workshops are the best way to learn, they teach you the practical skills you need to become an R wizard

Figure 1.1: courtesy of Allison Horst

1.3 Introduction to R

R is the name of the programming language itself and RStudio is a convenient interface.], which we will be using throughout the course in order to learn how to organise data, produce accurate data analyses & data visualisations.

Eventually we will also add extra tools like GitHub and RMarkdown for data reproducibility and collaborative programming, check out this short (and very cheesy) intro video.], which are collaboration and version control systems that we will be using throughout the course. More on this in future weeks.

By the end of this module I hope you will have the tools to confidently analyze real data, make informative and beautiful data visuals, and be able to analyse lots of different types of data.

The taught content this autumn will be given to you in several **worksheets**, these will be added to this dynamic webpage each week.

1.4 Getting around on RStudio

1.4.1 An intro to RStudio

RStudio

> Note - people often mix up R and RStudio. R is the programming language (the engine), RStudio is a handy interface/wrapper that makes things a bit easier to use.

1.4.2 Using R Studio Cloud

RStudio Cloud works in exactly the same way as RStudio, but means you don't have to download any software. You can access the hosted cloud server and your projects through any browser connection (Chrome works best), from any computer.

1.5 Get Help!

There are a **lot** of sources of information about using R out there. Here are a few helpful places to get help when you have an issue, or just to learn more

- The R help system itself - we cover this in @ref(Getting to know R - Week One)
- Vignettes - type `browseVignettes()` into the console and hit Enter, a list of available vignettes for all the packages we have will be displayed
- Cheat Sheets - available at RStudio.com. Most common packages have an associate cheat sheet covering the basics of how to use them. Download/bookmark ones we will use commonly such as `ggplot2`, `Data transformation with dplyr`, `Data tidying with tidyr` & `Data import`.
- Google - I use Google constantly, because I continually forget how to do even basic tasks. If I want to remind myself how to round a number, I might type something like `R round number` - if I am using a particular package I should include that in the search term as well
- Ask for help - If you are stuck, getting an error message, can't think what to do next, then ask someone. It could be me, it could be a classmate. When you do this it is very important that you **show the code**, include the **error message**. "This doesn't work" is not helpful. "Here is my code, this is the data I am using, I want it to do X, and here's the problem I get".

Note - It may be daunting to send your code to someone for help. It is natural and common to feel apprehensive, or to think that your code is really bad. I still feel the same! But we learn when we share our mistakes, and eventually you will find it funny when you look back on your early mistakes, or laugh about the mistakes you still occasionally make!

Chapter 2

Getting to know R - Week One

Go to RStudio Cloud and enter the Project labelled `Week One` - this will clone the project and provide you with your own workspace.

Follow the instructions below to get used to the R command line, and how R works as a language.

2.1 Your first R command

In the RStudio pane, navigate to the console (bottom left) and `type or copy` the below it should appear at the >

Hit Enter on your keyboard.

```
10 + 20
```

You should now be looking at the below:

```
> 10 + 20
[1] 30
```

The first line shows the request you made to R, the next line is R's response

You didn't type the `>` symbol: that's just the R command prompt and isn't part of the actual command.

It's important to understand how the output is formatted. Obviously, the correct answer to the sum `10 + 20` is `30`, and not surprisingly R has printed that out as part of its response. But it's also printed out this `[1]` part, which probably doesn't make a lot of sense to you right now. You're going to see that a lot.

You can think of `[1] 30` as if R were saying "the answer to the 1st question you asked is 30".

2.1.1 Typos

Before we go on to talk about other types of calculations that we can do with R, there's a few other things I want to point out. The first thing is that, while R is good software, it's still software. It's pretty stupid, and because it's stupid it can't handle typos. It takes it on faith that you meant to type *exactly* what you did type. For example, suppose that you forgot to hit the shift key when trying to type `+`, and as a result your command ended up being `10 = 20` rather than `10 + 20`. Try it for yourself and replicate this error message:

```
10 = 20
```

```
## Error in 10 = 20: invalid (do_set) left-hand side to assignment
```

What's happened here is that R has attempted to interpret `10 = 20` as a command, and spits out an error message because the command doesn't make any sense to it. When a *human* looks at this, and then looks down at his or her keyboard and sees that `+` and `=` are on the same key, it's pretty obvious that the command was a typo. But R doesn't know this, so it gets upset. And, if you look at it from its perspective, this makes sense. All that R "knows" is that `10` is a legitimate number, `20` is a legitimate number, and `=` is a legitimate part of the language too. In other words, from its perspective this really does look like the user meant to type `10 = 20`, since all the individual parts of that statement are legitimate and it's too stupid to realise that this is probably a typo. Therefore, R takes it on faith that this is exactly what you meant… it only "discovers" that the command is nonsense when it tries to follow your instructions, typo and all. And then it whinges, and spits out an error.

Even more subtle is the fact that some typos won't produce errors at all, because they happen to correspond to "well-formed" R commands. For instance, suppose that not only did I forget to hit the shift key when trying to type `10 + 20`, I also managed to press the key next to one I meant do. The resulting typo would produce the command `10 - 20`. Clearly, R has no way of knowing that you meant to *add* 20 to 10, not *subtract* 20 from 10, so what happens this time is this:

```
10 - 20
```

```
## [1] -10
```

In this case, R produces the right answer, but to the the wrong question.

2.1.2 More simple arithmetic

One of the best ways to get to know R is to play with it, it's pretty difficult to break it so don't worry too much. Type whatever you want into to the console and see what happens.

If the last line of your console looks like this

```
> 10+
+
```

and there's a blinking cursor next to the plus sign. This means is that R is still waiting for you to finish. It "thinks" you're still typing your command, so it hasn't tried to execute it yet. In other words, this plus sign is actually another command prompt. It's different from the usual one (i.e., the `>` symbol) to remind you that R is going to "add" whatever you type now to what you typed last time. For example, type `20` and hit enter, then it finishes the command:

```
> 10 +
+ 20
[1] 30
```

Alternatively hit escape, and R will forget what you were trying to do and return to a blank line.

2.1.3 Try some maths

```
1+7
```

```
13-10
```

```
4*6
```

```
12/3
```

Raise a number to the power of another

```
5^4
```

As I'm sure everyone will probably remember the moment they read this, the act of multiplying a number x by itself n times is called "raising x to the n-th power". Mathematically, this is written as x^n. Some values of n have special names: in particular x^2 is called x-squared, and x^3 is called x-cubed. So, the 4th power of 5 is calculated like this:

$$5^4 = 5 \times 5 \times 5 \times 5$$

2.1.4 Perform some combos

Perform some mathematical combos, noting that the order in which R performs calculations is the standard one.

That is, first calculate things inside **B**rackets `()`, then calculate **O**rders of (exponents) `^`, then **D**ivision `/` and **M**ultiplication `*`, then **A**ddition `+` and **S**ubtraction `-`.

Notice the different outputs of these two commands.

```
3^2-5/2
```

```
(3^2-5)/2
```

Similarly if we want to raise a number to a fraction, we need to surround the fraction with parentheses ()

```
16^1/2
```

```
16^(1/2)
```

The first one calculates 16 raised to the power of 1, then divided this answer by two. The second one raises 16 to the power of a half. A big difference in the output.

> **Note - While the cursor is in the console, you can press the up arrow to see all your previous commands. You can run them again, or edit them. Later on we will look at scripts, as an essential way to re-use, store and edit commands.

2.2 Storing outputs

2.2.1 Quitting R

There's one last thing I should cover in this chapter: how to quit R. When I say this, I'm not trying to imply that R is some kind of pathological addition and that you need to call the R QuitLine or wear patches to control the cravings (although you certainly might argue that there's something seriously pathological about being addicted to R). I just mean how to exit the program. Assuming you're running R in the usual way (i.e., through RStudio or the CLoud), then you can just shut down the application in the normal way by clicking the close windown button. However, R also has a function, called `q()` that you can use to quit, which is pretty handy if you're running R in a terminal window e.g. when using servers - we will cover some of this later in the course.

Regardless of what method you use to quit R, when you do so for the first time R will probably ask you if you want to save the "workspace image". I talked about this in one of my videos, it's annoying and occasionally introduces complications, so if prompted it is usually better to click `NO` (but don't worry if you hit the wrong button).

What does this actually *mean*? What's going on is that R wants to know if you want to save **all** those variables that you've been creating, so that you can use them later. This sounds like a great idea, so it's really tempting to type `y` or click the "Save" button. To be honest though, I very rarely do this, and it kind of annoys me a little bit… what R is *really* asking is if you want it to store these variables in a "default" data file (R.data), which it will automatically reload for you next time you open R. But as we will learn to work with scripts, this is unnecessary.

In fact, every time I install R on a new machine one of the first things I do is change the settings so that it never asks me again. You can do this in RStudio really easily: use the menu system to find the RStudio option; the dialog box that comes up will give you an option to tell R never to whine about this again.

We practice what we preach! This site is created with Git and R markdown, using the `bookdown` package. Go ahead and peek behind the scenes.

Long-term, you should understand more about what you are doing. Rote clicking in RStudio may be a short-term survival method but won't work for long.

- trygit is to (command line) Git as swirl is to R. Learn by doing, in small bites.
- The book Pro Git is fantastic and comprehensive.
- Git in Practice by Mike McQuaid is an more approachable book, probably better than Pro Git for most people starting out. Ancillary materials on GitHub.
- Git for Humans is a great set of slides by Alice Bartlett, originally delivered in 2016 at UX Brighton.
- GitHub's own training materials may be helpful. They also point to many other resources
- Find a powerful Git client (chapter **??**) if you'd like to minimize your usage of Git from the command line.
- Hadley Wickham's book R Packages has an excellent chapter on the use of Git, GitHub, and RStudio in R package development. He covers more advanced usage, such as commit best practices, issues, branching, and pull requests.
- Ten Simple Rules for Taking Advantage of Git and GitHub http://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1004947

- RStudio's guide Version Control with Git and SVN
- The book *Team Geek* has insightful advice for the human and collaborative aspects of version control. It proposes Git strategies suited to different characteristics of teams.