Teach Yourself or Others R

Adam Rawles

Contents

Chapter 1

teacheR

1.1 Overview

This book is a collection of training materials for an introduction to the R statistical computing programming language. Broken down into chapters, I've aimed to cover most of the basics. Alongside each chapter is a xaringan presentation. This can help for those looking to learn, but can also function as a first step for those looking to begin teaching R but who don't have the time to fully develop their own training modules. Hopefully, all the topics covered in the text version will be covered in the presentation and vice versa, so if you learn visually then you can rest easy knowing that you're not missing out! You can find a link to each presentation on the first page of each chapter.

This is a work in progress, and so I would greatly appreciate any feedback. Anything from typos to content suggestions, feel free to raise a GitHub issue if you feel something should be changed.

1.2 About Me

I began using R in my second year of university, during an internship looking at publication bias correction methods. I was under the tutorship of a member of staff who helped me immensely, but I must confess that I have never taken an official course in R, online or in person. I like to think, however, that this is not always a bad thing. Learning from the bottom up and struggling along the way is a fantastic way to acquire knowledge, and I believe instills a very important lesson:

You're not going to know everything there is to know about R. Ever. But that's okay.

I'm now 4 years into my R career, and I use R every day. With that in mind, I don't think there has ever been a day where I haven't referred to a tutorial, or Stack Overflow, or even just Googled the name of a function that I've used 1000 times before. There is a great repository of knowledge for R, and it's one of the things I love most about the R community. So please never feel as though you're an impostor in a world of R gurus. In reality, everyone else is just as lost as you. But if you keep ticking along and never feel that learning something new in R isn't worth your time, you'll end up doing some great things.

And in a roundabout way, that is part of the reason I decided to develop these materials. I don't pretend to be the ultimate R programmer, because I still know what it's like to learn something from the start. And everyone has to start somewhere. So I hope that I can help impart some of the lessons that I've learnt over the 4 years to anyone who's looking to learn R in a way that won't leave you feeling lost - and hopefully I can save some poor soul from the same headaches I suffered along the way.

The only final note I have before we start learning how to use R is another bit of advice:

Don't believe everything you read

Whilst this is probably a good thing to keep in mind for any type of training, I feel it's particularly relevant with R for two reasons. Firstly, when it comes to programming languages, lots of people have opinions. Some are true, most are not. Most things you read are a mix between fact and opinion, so take everything with a pinch of salt. For example, the developers of the `ggplot2` package are fervently against arbitrary second axes and so support for them in `ggplot2` is limited. I also share this view, but that doesn't mean that I'm right - read, learn, but question and make your own mind up. Secondly, R and particularly all of its packages are prone to change. For this reason, people may make statements relative to one version of R that aren't necessarily true in the future. Things have changed over the years, and so answers from a 10 year-old Stack Overflow question may not still be true when you come across them. A microcosmic version of this are some recent changes in the `tidyr` package. Historically, converting data from/to long and short formats was done using the `spread()` and `gather()` functions. However, in newer releases, these functions are deprecated in favour of `pivot_wider()` and `pivot_longer()`, which provide the same functionality but also some extra bits. The practical implication of this suggestion is don't always read one tutorial on a subject before you dive in.

Chapter 2

Introduction to R

Chapter 2 is a general overview of R and its basics. I personally use R Studio as my development environment of choice (more on development environments later on), so I've also included a very brief overview of R Studio.

The xaringan presentation for this module is here.

2.1 What is R and RStudio?

2.1.1 R

R is a public license programming language. More specifically, it's a statistical programming language meaning that it's often used for statistical analysis rather than software development. R is also a functional programming, rather than an object-oriented programming language like Python. This means that operation in R are primarily performed by functions (input, do something, output), but more about that later.

Strictly speaking, R is not just a functional programming language. In reality, a language is never purely one type and R is no exception. There are object-oriented systems in R (three main ones), meaning that object-oriented programming is possible and relatively straightforward in R. Having said that, I feel now is a good time to refer back to the mantra from the Overview chapter:

Don't believe everything you read

So basically, R is a functional programming language with some object-oriented systems. If that means very little to you, don't worry. For the vast majority of users, this is a purely academic definition.

One important attribute about R however that may affect you, is that R is an interpreted language. This essentially means that when you send someone some R code, they need R installed to be able to run it. This means that making full programs is difficult. Later on, we'll look at the `shiny` package, which can be used to quickly make web apps based on R code. These apps are no different in the sense that they also need R installed to be able to run, but because they are web-based, they are significantly easier to share.

For the most part however, if you want to share R code with colleagues, they'll need to have R installed as well.

2.1.2 Should I use R over another language?

People have and will continue to argue about which is better, R or Python or Java or C or writing down mathematical equations on a piece of paper and handing it to a monkey to solve. I imagine you're reading this because you heard that R was good for data analysis, and it absolutely is. And so is Python. And so is VBA. They're just… different. Personally, I prefer to use R but I understand that other people don't.

Importantly though, never feel as though you've missed a trick by picking a particular language. Programming is not just a practice, it's a way of thinking, and experience is almost always transferable across languages.

For reference however, here are a few of the things that you can use R for:

- data analysis
- reporting and writing
- web apps
- text analysis

If you're interested in any of these, then you're in the right place.

2.1.3 RStudio

RStudio is separate from R. R is a programming language and RStudio is an integrated development environment or IDE. This means that RStudio doesn't actually run any code, it just passes it to R for you, meaning that you'll need R to really use RStudio.

RStudio is a massive part of how you interact with R however. For example, with the exception of a few days when I was waiting for RStudio to be installed, I can't ever remember using R without RStudio.

2.1.3.1 What is an IDE?

At its simplest definition, an IDE helps you get work done in your programming language of choice. It can help you save blocks of code, organise projects, save plots and everything in between. R comes with a basic GUI when you install it, but RStudio provides lot more functionality to help you interact with the R console.

2.1.3.2 RStudio Panes

TO DO

2.2 Operators

2.2.1 Arithmetic operators

At the base of lots of programming languages are the arithmetic operators. These are your symbols that perform things like addition, subtraction, multiplication, etc. Because these operations are so uniquitous however, the symbols that are used are often very similar across languages, so if you've used Excel or Python or SPSS or anything similar before, then these should be fairly straightforward.

Here are the main operators in use:

```
2 + 2  # addition
```

```
## [1] 4
```

```
10 - 5 # subtraction
```

```
## [1] 5
```

```
5 * 4 # multiplication
```

```
## [1] 20
```

```
100 / 25 # division
```

```
## [1] 4
```

2.2.2 Logical operators

Logical operators are slightly different to arithmetic operators - they are used to evaulate a particular criteria. For example, are two values equal. Or, are two values equal *and* two other values different.

To compare whether two things are equal, we use two equal signs (`==`) in R:

```
1 == 1 # equal
```

```
## [1] TRUE
```

Why two I hear you say? Well, a bit later on we'll see that we use a single equals sign for something else.

To compare whether two things are different (not equal), we use `!=`:

```
1 != 2 # not equal
```

```
## [1] TRUE
```

The `!` sign is also used in other types of criteria, so the best way to think about it is that it inverts the criteria you're testing. So in this case, it's inverting the "equals" criteria, making it "not equal".

Testing whether a value is smaller or larger than another is done with the `<` and `>` operators:

```
2 > 1  # greater than
```

```
## [1] TRUE
```

```
2 < 4 # less than
```

```
## [1] TRUE
```

Applying our logic with the `!` sign, we can also test whether something is *not* smaller or *not* larger:

```
1 >! 2 # not greater than
```

```
## [1] TRUE
```

```
2 <! 4 # not less than
```

```
## [1] FALSE
```

Why is the `!` sign before the equals sign in the "not equal" to code, but after the "less than/greater than" sign? No idea. It'd probably make more sense if they were the same, but I suppose worse things happen at sea.

There are three more logical operators, and they are the "and", "or", and "xor" operators. These are used to test whether at least one or more than one or only one of the logical comparisons are true or false:

```
1 == 1 | 2 == 3 # or (i.e. are either of these TRUE)
```

```
## [1] TRUE
```

```
1 == 1 & 2 == 3 # and (i.e. are these both TRUE)
```

```
## [1] FALSE
```

The xor operator is a bit different:

```
xor(1 == 1, 2 == 3) # TRUE because only 1 is
```

```
## [1] TRUE
```

```
xor(1 == 1, 2 == 2) # FALSE because both are
```

```
## [1] FALSE
```

For `xor()`, you need to provide your criteria in brackets, but this will make much more sense once we look at functions.

2.3 Variable assigment

Do you ever tell a story to a friend, and then someone else walks in once you've finished and so you have to tell the whole thing again?

Well, imagine after the second friend walks in, another friend comes in, and you have to start the story over again, and then another friend comes in and so on and so forth. What would be the best way to save you repeating yourself? As weird as it would look, if you wrote the story down, then anyone who came in could just read it, rather than you having to go through the effort of explaining the whole thing each time.

This is essentially what we can do in R. Sometimes you'll use the same value again and again in your script. For example, say you're looking at total expenditure over a year, the value for the amount spend would probably come up quite a lot. Now, you could just type that value in every time you need it, but what happens if the value changed? You'd then have to go through and change it every time it appears.

Instead, you could store the value in a variable, and then reference the variable every time you need it. This way, if you ever have to change the value, you only need to change it once.

Creating variables in R is really easy. All you need to do is provide a valid name, use the `<-` symbol, and then provide a value to assign:

```
hello_im_a_variable <- 100
hello_im_a_variable
```

```
## [1] 100
```

Now, whenever you want to use your variable, you just need to provide the variable name in place of the value:

```
hello_im_a_variable / 10
```

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

You can even use your variable to create new variables:

```
hello_im_another_variable <- hello_im_a_variable / 20
hello_im_another_variable
```

```
## [1] 5
```

When you come across other people's work, you may see that they use `=` instead of `<-` when they create their variables. Even though it's not the end of the world if you do do that, I would recommend getting into the habit of using `<-`. `<-` is purely used for assignment, whereas `=` is actually also used when we call functions, and so it can get a bit confusing if you use them interchangeably.

As a side note, you'll see that the value of the variable isn't outputted when we assign it. If we want to see the value, we need just the name.

2.4 Data types

filler

2.5 Data structures (and subsetting)

filler

2.6 Functions

Chapter 3

Data Analysis in R

In Module 2, we'll look more specifically at how one might do some simple data analysis in R. For a more in-depth view, I would highly recommend Hadley's R4DS

The xaringan presentation for this module can be found here.

3.1 Installing packages

filler

3.2 Loading data

filler

3.3 Cleaning data

filler

3.4 Summary statistics

filler

3.5 Plots

Chapter 4

Programming in R

In this Module, we'll look at some of the programming concepts and syntax in R.

The xaringan presentation for this module can be found here.

4.1 User-defined functions

filler

4.2 For loops

filler

4.3 If/else statements

filler

Chapter 5

Tidyverse

In this module, we'll take a quick look over some of the packages in the tidyverse.

The xaringan presentation for this module can be found here.

5.1 tidyr

filler

5.2 dplyr

filler

5.3 stringr

filler

5.4 ggplot2

filler

Chapter 6

Modelling

6.1 Linear modelling

6.2 Clustering

6.3 Programming and modelling

Chapter 7

Exercises

Here are some interactive exercises covering the basic concepts we've looked at in the last 4 modules.

This exercises are hosted on a Shiny server here