Introduction to R

Francis Yuen

2019-12-07

1. Introduction to R and why it's awesome
2. Install R and RStudio
3. Understand the basics of R
4. Learn how to use simple functions

Introduction to R

R is essentially another option for computing statistics. It is (initially) less intuitive to use compared to SPSS and Excel, but it has several advantages that make it an awesome tool. Over the next few months, you will most likely consistently ask yourself: "Why am I struggling with this instead of just using SPSS or Excel?" That's totally normal, and you just have to remind yourself of the two major advantages of R:

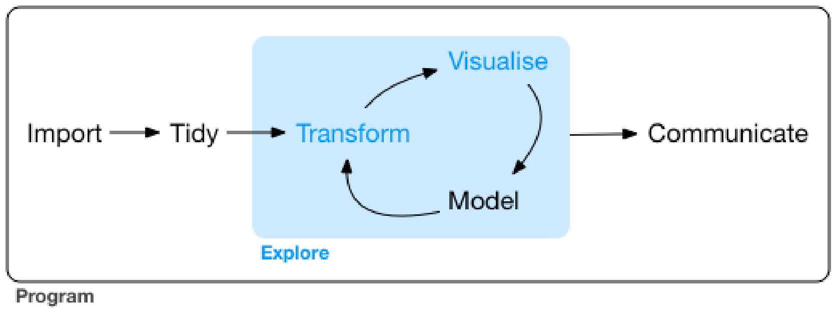

Figure 1: Workflow in R, image from https://r4ds.had.co.nz/explore-intro.html

1. Flexibility: R has many more options as your computation gets more complicated
2. Reproducibility: R calculations are 100% reproducible, and even manuscripts can be reproducible when used in conjunction with RMarkdown (more on this later)

R is essentially a one-stop-shop for all your research needs: from data cleaning to analysis to visualization, everything can be done in one place!

Installing R and RStudio

To get things started, you will need to download R as well as RStudio:

For Windows 10/8/7:
R and **RStudio**
For Mac:
R and **RStudio**

Rstudio is a set of tools that makes R easier to use. For the purposes of this workshop, we will be only working in RStudio (but you still need to install both).

Basics of R

The language of R

Remember that because R uses 'computer' language, it can be very picky and 'black and white' when it comes to certain things:

- Capitalization: "With" ≠ "with"
- Spelling errors: "blue" ≠ "bleu"

There are also different types of data. The most common ones that we will be working with are **character** and **numeric**.

- **Character** data are essentially words or letters. You can think of them as individual 'objects', and they need to be surrounded by quotation marks (e.g. "With") when you enter them.
- **Numeric** data are continuous numbers and can be entered as is.

Math operations

Now that we have RStudio installed, let's take a look at the different parts of the interface. First, let's explore what R can do by looking at the console. The console is where you give 'commands' to R, ranging anywhere from simple math questions to several lines of code.

To illustrate, let's ask R to solve a math questions that's literally impossible for an average human to compute:

```
9 + 10
```

```
## [1] 19
```

Here are some common operations and how to enter them:

operation	operator/symbol	example input	example output
addition	+	2 + 2	4
subtraction	-	5- 1	4

operation	operator/symbol	example input	example output
multiplication	*	2 * 2	4
division	/	8 / 2	4
power	^	2 ^ 2	4

One thing to note here is that R doesn't care about spacing: 1+1 and 1 + 1 is treated the same way, so you can organize your eventual code according to how you like it visually.

Exercise 1

Help each other solve some *actually* difficult math problems.

Storing variables

Using R like a calculator is fun, but we can also store our answers as temporary variables in our workspace using the <- operator. Try typing the following in your console:

```
answer <- 3 + 5
```

Notice now it doesn't give you the answer as an output, but it stores the answer as an object called "answer" in your workspace (or your Environment). Now, type in answer in your console to recall it:

```
answer
```

```
## [1] 8
```

This will be especially useful when you have to do more complex things like applying functions to larger data sets. Notice that you have to type in the variable name every single time you use it, so try to avoid long but more informative variable names like "Number of times baby chose blue". Instead, here are some commons ways people standardize variables:

- choice_blue
- blue.choice.num
- ChoseBlue

Note that none of these variable names have spaces, and this is because R uses white space as a separator, so anything separated by white space is treated as two objects.

It is also useful to have these commands as actual lines that you can rerun instead of a one-time thing, so let's open up an R script. Go to File -> New File -> R script. In your new script, type in the exact same thing as before: answer <- 3 + 5.

Now, if you press Enter, all it will do is move on to the next line. In order to tell R that you want to execute this line, click anywhere on the line you want to run (or to run multiple lines, highlight the lines you want to run) and click the Run button on the top right of your script section, or hold the Ctrl key (command key for Mac users) and press Enter.

Exercise 2

Pretend you ran 5 babies in a looking time study, and their looking times on the test trial are 5, 7, 15, 21, and 25 seconds. Create a variable to store the sum of their looking time, a variable that represents the number of babies you ran. Finally, using those two variable, create a third variable that represents the mean looking time for your study.

Challenge: Instead of doing this in three steps in your console, write three lines in your script that performs these actions and run them all at once!

Using functions

Functions and Arguments

Now we know how to store variables, we can start doing cool things to them using functions. Functions are essentially a series of commands someone built in advance to be applied to a given set of input, and all we have to do is provide the correct input. Some functions are very simple, like `nchar()` which just counts the number of characters in a given object:

```r
nchar("Francis")
```

```
## [1] 7
```

Some can do useful statistical computations, like `sd()` which calculates the standard deviation of a set of numbers. Generally, functions are used using the format:

function(argument1, argument2, argument3, etc…)

Arguments are the different inputs or settings you can give to a function. Every function has its own unique set of arguments, and they are very well documented. As an example, let's take a look at the function `round()`. First, type in `?round()` in your console, which will give you the explanation of the function. Scroll down to the 'arguments' section, and notice that it takes two main arguments: x and digits. That means you have to specificy x (what number/vector you want rounded) and digits (how many digits to round to). Now that we know what arguments this function takes, try to round 3.14159 to 3 digits:

```
round(x = 3.14159, digits = 3)
```

```
## [1] 3.142
```

Awesome! But what if we don't specify and just try to type it in?

```
round(3.14159, 3)
```

```
## [1] 3.142
```

Notice that it still works because the things we put in for the arguments are in the correct order. However, if we do this:

```
round(2, 3.14159)
```

```
## [1] 2
```

It doesn't work the same way. Generally, it is highly recommended to name your arguments even if you know you are doing it in the correct order because:

1. You can use them out of order (e.g. round(digits = 3, x = 3.14159) will work!)
2. The next person reading your code will know what the heck is going on
3. Future you will know what is going on
4. As functions get more complicated (5+ arguments), you will probably want to keep track to prevent mistakes (e.g. specifying the variable as either between subject or within subject will make a huge difference when doing your anova arguments!)

Packages

Where R truly shines is the fact that it is open source, meaning that everyone can contribute to its development. This means that people are constantly building new functions and combining them into 'packages'. The best part? These packages are free for you to use! Instead of being limited by whatever your program offers (like Excel or SPSS) and having to pay subscription, the world is your oyster. To conclude this tutorial and to segway into the next one, we will install two of my favorite packages for reproducible research: "tidyverse" and "here". To install new packages, the command is `install.packages("package_name")`:

- `install.packages("tidyverse")`
- `install.packages("here")`

Before you can use any of your installed packages, you have to first 'load' them by adding them to your current working library using the `library()` function:

```
library(tidyverse)
library(here)
```

Now we have some useful packages, we are ready to start tackling some datasets!