Lab 3

Steven Boyd

10/14/2021

This week, I am not going to provide you with an .Rmd. Please create a new one to work in for this lab. First, take a moment to review `tex-in-markdown.pdf` in the `explainers` folder of the GitHub repository. If you still cannot knit to pdf, go to quicklatex.com. It will render latex code in your browser and should be sufficient for the exercises today.

What is LaTeX?

At its core, LaTeX (stylized LaTeX) is a typesetting language. It renders text into formatted tables, mathematical expressions, appendices, bibliographies, etc. If you invest time in learning the syntax, it can produce better looking documents in less time (and with fewer headaches) than most word processing software. This is especially true if your work includes a lot of mathematical symbols or tables. As quantitative social scientists, it probably will!

A quick note on LaTeX and Markdown. Many of the formatting tricks you have seen us use in R Studio come from Markdown, which is its own plain text formatting language, built for conversion to HTML. LaTeX is built for conversion to pdf. Our R Markdown documents frequently combine the two languages. R Studio relies on software called PanDoc to convert mixed input into the format needed for whichever type of document you are generating. BUT it's important to note that when you knit to HTML, your raw LaTeX *might* not render like it does if you knit to PDF. That is because when you knit to a pdf, PanDoc first converts the whole file into a .tex file, which contains only LaTeX code.

If you ever need to create .tex documents from scratch, there is other software available that is specifically for LaTeX, and there are some additional steps you have to take to actually generate a document. If you start googling questions about LaTeX code or formatting, you will find most answers assume you are working in a Tex editor, but adding "in R" or "in R markdown" should help you find useful information. If you want to learn more about how RMarkdown works with LaTeX, I recommend the R Markdown Cookbook (especially chapters 2 and 6).

Math mode!

The most important LaTeX tool for this class is called *math mode*. Math mode basically tells R that you want the text rendered as a mathematical expression. It is helpful for anything with Greek letters, fractions, equations, etc.

You can use some symbols in math mode as you normally would. For example: $+- =><$. Here is a quick guide to some of the symbols I use frequently (and you probably will too). They require a little more work:

Symbol	Command
$\neq$	`\neq`

Symbol	Command
$\leq$	`\leq`
$\geq$	`\geq`
$\times$	`\times`
$\cap$	`\cap`
$\cup$	`\cup`
$\land$	`\land`
$\lor$	`\lor`
$\in$	`\in`
$\notin$	`\notin`
$\subset$	`\subset`
$\subseteq$	`\subseteq`
$\nsubseteq$	`\nsubseteq`
$\mathbb{R}$	`\mathbb{R}`

For a good cheat sheet organized by category, I recommend this pdf. For a comprehensive list of latex symbols (more than 18,000 of them!) see this pdf from CTAN, the TeX equivalent of CRAN.

Other important stuff

The explainer shows how to use exponents and that we can nest exponents within exponents. The same principles apply to subscripts, but instead of `^`, use `_`:

$$P_A(X = x) = .5$$

$$P_{A_1}^{2^x}$$

Another helpful tool is `cases`. Look at this example to see how it works:

```
f(x) = \begin{cases}
\frac{1}{6} & x = 1 \\
\frac{1}{6} & x = 2 \\
\frac{1}{6} & x = 3 \\
\frac{1}{6} & x = 4 \\
\frac{1}{6} & x = 5 \\
\frac{1}{6} & x = 6 \\
\end{cases}
```

$$f(x) = \begin{cases} \frac{1}{6} & x = 1 \\ \frac{1}{6} & x = 2 \\ \frac{1}{6} & x = 3 \\ \frac{1}{6} & x = 4 \\ \frac{1}{6} & x = 5 \\ \frac{1}{6} & x = 6 \end{cases}$$

Two things to note: `\\` tells LaTeX to skip to a new line and `&` is an alignment character. `&` is not printed in the output. Rather, it tells latex where to match up the lines vertically. By moving the `&` we can change the alignment.

This is also important when we have an equation that spans several lines, which is very useful for showing your work on more complex math problems and proofs. We can define an aligned environment like this:

```
\begin{align*}
f(x) & = (x-2)(x+3) - x(x+5) \\
& = x^{2} - 2x + 3x - 6 - x^{2} - 5x \\
&= -4x - 6
\end{align*}
```

$$\begin{aligned} f(x) &= (x-2)(x+3) - x(x+5) \\ &= x^2 - 2x + 3x - 6 - x^2 - 5x \\ &= -4x - 6 \end{aligned}$$

Again, a few things to note. First, we don't need `$` or `$$` to enter math mode if we are working within `align` (or `equation`, which operates similarly but is generally best for single line expressions). Second, there is a `*` following align. This overrides the default behavior of `align` to number each section of mathematical notation in sequential order. There are cases where you might want these secionts to be numbered (perhaps they are theorems you want to reference later). Simply remove `*` and they will be numbered for you!

Practice

Let's put all of this together and recreate some expressions you have already seen (and will probably see again):

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

$$P(B) = \sum_i P(B \cap A_i)$$

$$f(x) = \Pr[X = x], \forall x \in \mathbb{R}$$

$$\Phi(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-\frac{u^2}{2}} du$$

Functions

Switching gears now. A function in R is a mapping from inputs (or arguments) to an output. This is very similar to the mathematical definition of a function. You have already worked with many R functions, even if you haven't thought of them in these terms (e.g. `mean()`, `mutate()`, `pivot_longer()`).

What arguments do you pass into these various functions? What do they give you as output?

These functions are defined for you by either base R or tidyverse packages, but you are free to define your own functions too! This is an extremely powerful tool for tasks that need to be repeated many times.

The basic components of a function are: name (should be descriptive), arguments (these are your inputs), and body (instructions for manipulating the inputs).

Interpreting Functions

Take a look at this function. Can you describe what it does? What are the components?

```
my_function <- function(x) {
x - mean(x, na.rm = TRUE)
}
```

Generate some data to test your prediction (any method is fine, but make sure it's small enough that you can quickly check if it is working correctly). Put the function in a chunk and call it on your data. Does it work?

Writing Functions

Ok, now it's your turn! Create a function that returns the *range* of observations as a single number. Before you code anything, think about what information you need to extract from the data to answer this question. Then figure out how to use those data points to produce the value you want. Then try to produce code to do it for you.

Next, generate 100 data points with `rnorm()` (don't forget to set your seed). If you are unfamiliar with `rnorm()`, check the documentation. Run your function on this randomly generated data.

Now, generate another 100 points with `rnorm()`, but alter the argument for standard deviation. What do you expect to happen to the range of the data? Call your function again to check your prediction.

Repeat this process but change the mean instead of standard deviation. Was your prediction correct this time?