R Markdown Tutorial- In an R Markdown File!

YOUR NAME

2024-07-26

Welcome! This is a brief tutorial on R Markdown, which I've made in an R Markdown file! Now, as you're used to the tutorial format, you may be feeling a bit intimidated by this less structured mode of instruction, but I assure you, you will be doing the same coding you've done thus far, rmd files mostly change your formatting and the way in which your file is stored and exported. (Fun fact, the tutorial based labs you're used to were actually still made using an R Markdown file, it was just modified with a package called learnr. We won't cover how to accomplish that in this course but that's just one example of all the cools things you can do with a mastery of R Markdown.) So, without further ado, let's dive in.

What is R Markdown?

To start, let's talk about what an R Markdown file actually is. Markdown is a kind of file that exists beyond R itself, which allows you to integrate text with code. An Rmd is simply a markdown file made especially to work with R. In RStudio, you can create multiple R Markdown Files within a project, just like you can save multiple R scripts.

Running Code in R Markdown

Right now, you are reading text which I have written in the text apparatus of R Markdown. This text is the first half of the basic R Markdown (RMD) structure, the other main component are chunks in which you can write your code. I've put an example of one such chunk below:

```
#within the chunks, we can write and comment code as normal. Let's use a very simple
↪  example to demonstrate how running code in the chunks works
a<- c(2, 5, 7, 6) #creating a vector called a
print(a) #printing it
```

```
## [1] 2 5 7 6
```

Observe that the code chunks are grey and bounded by a sequence of 3 back ticks at the top and bottom of the chunk. The top boundary also has curly brackets, which in its most basic form contain just r, but you can add additional conditions which impact how the chunk functions, which we will get to later.

To see how the chunks output our code, let's run our chunk. There are few ways to do that. We can use the "Run" button at the top right of the panel we have the Rmd file in. If you click the drop down arrow on the "Run" button you will see there's quite a few options for running chunks, along with keyboard shortcuts for each associated action, in case you get tired of using the buttons. Take a minute now to play around with those buttons, selecting certain lines of code in your chunk to run or running the code in its entirety. There is also a simple way to run the code in the chunk within the chunk itself. The rightmost symbol on the topmost line of the chunk is a green triangle, which when pressed will run the chunk in its entirety. This is the most common way that I run code when I use RMD files, but feel free to try out the different methods and decide which one works best for you.

Now that we've gone over how to run our code chunk, let us turn to the output of running said chunk. In the example above, I created a vector which I named `a` and then printed it. Recall Lab 1 and its instructions on the layout of R Studio. Because we're working in an RMD you can actually see the Console, Environment,

and available files as you work. When you run the line creating the vector, it copies the code down into the console and adds the variable `a` into your environment. When you run the line which prints `a` the vector appears in the RMD, in a box just below the chunk itself. Certain functions (especially in dplyr, which we will get to in the rest of Lab 3 today), display results directly in the file without the use of `print()`. This output is relevant because much of the aforementioned chunk customization allows you to manipulate the output when a chunk is run. Before we get to playing with the chunk headers, though, let's first tackle creating a chunk.

Creating Code Chunks

There are 3 main ways to create a new code chunk. First and most direct is manually typing the back ticks and curly brackets, however, this method is also the most tedious. Second is to use the green C button directly to the left of the "Run" button. If you click the button's drop down arrow, you'll see that it can actually generate chunks for coding languages beyond just R, including Python and Bash. The only coding this class will cover is in R, but if you know or want to learn these other languages you can use RMD to interact with them! Finally, you can use a key board shortcut to generate an R coding chunk: Ctrl + Alt + I (or Cmd + Option + I if you're using a Mac). Try now to create a chunk of code using one of these methods.

Knitting and Output Formatting

Before we talk about manipulating the output of a code chunk in the final file of an RMD, we have to talk about how to generate a compiled file, aka knitting. At the top of this panel, to the right of all those other code and running buttons we talked about, is the "knit button". (btw, you'll never miss this particular button, because it has a super cute icon of a ball of yarn with a knitting needle sticking out of it). If we click the drop down arrow next to the button, we can see some of the options for the file we'll produce upon knitting. Take a look at the options now. In this class, we'll always knit to either pdf or html but you can also knit to a Word document. Try knitting this file now, first to html and then to pdf. Once knit, you can download the file and save it or send it somewhere. Once we move to RMD style labs (starting next week) you'll upload the pdf or html file to Moodle in order to turn them in. When you look at the knit file, notice the appearance of the header and the subheadings I've made for each section.

The header includes the information one fills in when prompted to provide a title, author, and date. Under date is a neat bit of code, 2024-07-26 which automatically fills in the current date! Very cool and streamlined if you ask me.

Moving onto the subheadings, in the file itself you may notice the presence of two hashtags preceding the title and the blue color. These hashtags let RMD know you're creating a header, the more hashtags you include, the smaller the font becomes upon knitting. In order for this to work, remember to include a space between the end of the hashtags and the start of the words in your heading. Try a few below, and feel free to play around with the sizes:

Another cool effect you can add to a knit rmd is the inclusion of code in the text portion of your file. If you enclose a piece of code (most often a function) in back ticks, it will appear formatted as code in the knit file. For example, I might want to say that in the following chunk I will be using the function head to look at some data. I can write it like `head()` and have it appear as the function would in the knit file.

Let's now look at a few of the options we have for controlling what an R markdown file looks like when it's knit. The two main options we'll cover here are `echo =` and `eval =` . We incorporate them in the top line of the chunk like this:

As you can see, they take logical variables as their input, which tells R what qualities are true or false about the given chunk. `echo` pertains to the chunk's appearance in the knitted file. When `echo = false` the chunk will essentially be made invisible, the code will still be there and be executed when the document is run, but you won't see the code in the knitted file. If you have code that you want to run in the background but not appear in the final, shareable file, you can use `echo= FALSE`. Create a chunk with some code (perhaps creating a simple vector like earlier), include `echo=FALSE` and knit the file to see the result

`eval =` determines whether the output of a code chunk is included in the knitted file. This is especially useful if you need to look at a large data set or iterate on a graph but don't want these large outputs to clog your final file. To demonstrate, I'm going to put a simple plot with data from a data set about seals! It's a count of the amount of two different species of seal in different locations over time, as a part of environmental/conservation efforts. (your student coder was going through a seal phase when they wrote this lesson, get excited!) Knit this file once with eval equal to false as I have it set up, then make eval equal to true and knit again.

```
seals<-read.csv("AllSealCounts.csv") #reading in seal data
hist(seals$Harbour) #making a histogram with the count of the Harbour Seals in the data
```

The last thing I'll touch on is how to add images to an RMD file. In order to do this, you need to first upload your photo into your files, making sure if you're working with multiple folders, it's in the same one as your RMD. You should be able to upload it through the square with a yellow arrow button (aka the upload button). Once it's in your files, you need to provide a path for it into the environment, similar to reading in a csv. The syntax is as follows:

 {width = 70%}

In the square brackets is a caption for the picture, in the parentheses is the name of the folder that the picture is in (images) and the exact name of the file as it appears in the images folder of this project broken up by a slash, this tells R to look in that folder for the picture. (feel free to go look in that folder). In the squiggly brackets is an optional argument which specifies the percent of the page width I'd like the image to take up.