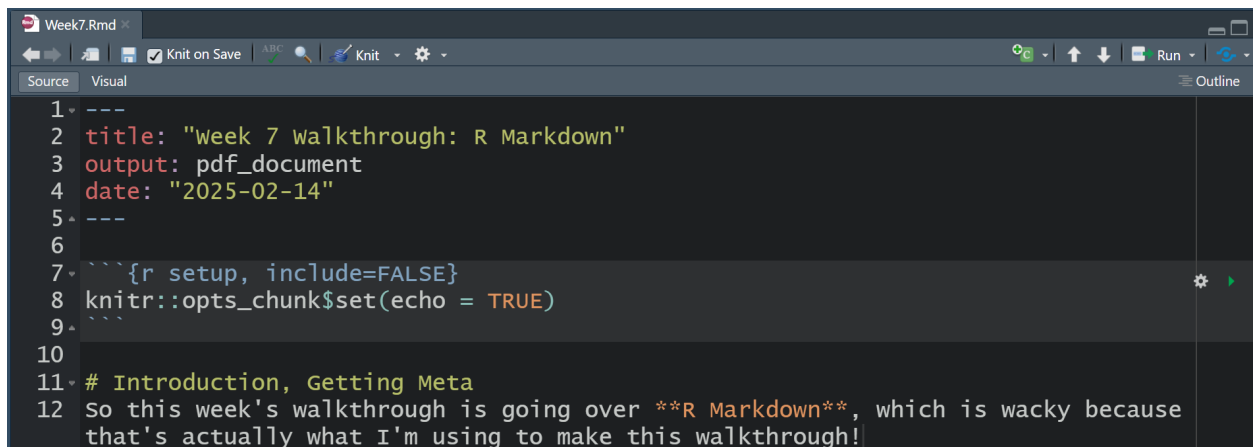


Week 7 Walkthrough: R Markdown

2025-02-14

Introduction, Getting Meta

So this week's walkthrough is going over **R Markdown**, which is wacky because that's actually what I'm using to make this walkthrough!

A screenshot of an R Markdown script in a code editor. The editor has a dark theme and a toolbar at the top with icons for navigation, search, and execution. The script content is as follows:

```
1 ---
2 title: "Week 7 walkthrough: R Markdown"
3 output: pdf_document
4 date: "2025-02-14"
5 ---
6
7 ```{r setup, include=FALSE}
8 knitr::opts_chunk$set(echo = TRUE)
9 ```
10
11 # Introduction, Getting Meta
12 So this week's walkthrough is going over R Markdown, which is wacky because
    that's actually what I'm using to make this walkthrough!
```

So this will be kind of weird as I'm going to keep including screenshots of what I'm doing *in* what I'm doing. The purpose of **R Markdown** is to mix code output and normal text in a convenient, readable way and honestly it's pretty good at that.

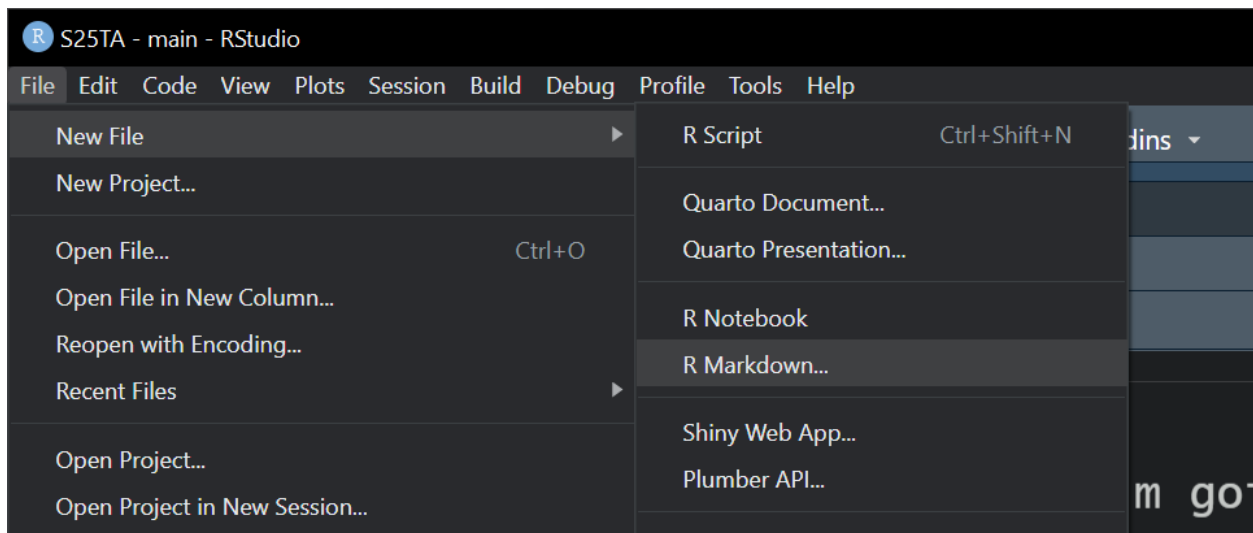
I cannot recommend **the R Markdown cheatsheet** enough, it has everything you need to be reminded of at any given time and puts it all in one place.

R Markdown is actually a *huge* topic and *very* important. This is likely to be the thing that you use the most after this class. Why, you ask? Because I can make a monthly report, quarterly report, whatever exactly *once* and then use it for *every* month, quarter, whatever after that. I talk a lot about coders being lazy and this is the keys to the kingdom.

There's a big focus put on reproducibility and transparency in the course notes, which is very important for open science but not necessarily what you'll be using it for. **R Markdown**'s reproducibility is the real key, but it doesn't *have* to come with transparency. Not that I would encourage such behavior, but you *could* (as an example) be asked by your boss to do a big monthly report. You could then *hypothetically* emphasize what a *big* job it is and how it will take a *huge* chunk of your day/week. But since you wrote an **R Markdown** script for this purpose last month, you *could* just take the 30 minutes required to update your script and then kick your feet up for a few hours while your boss thinks you're carrying the company on your back. Hypothetically.

Making an R Markdown File

R Markdown files use the `.Rmd` file extension and you can make them the same way you make `.R` files!



Ta-dah! You'll be asked what kind of document you want the **R Markdown** file to generate, either HTML (best for webpages, opened in your web browser) or PDF (best for reports, anything you would print).

You'll get what's called the **boilerplate** document, which has all the options filled out and some example text (that actually explains more about **R Markdown**, check it out!).

YAML Headers

Once you've made your **R Markdown** document, you'll see at the beginning something like the below. This is called the **YAML Header**, a place to put options that let you tailor your output a little bit more.

```
1 ---
2 title: "Week 7 walkthrough: R Markdown"
3 output: pdf_document
4 date: "2025-02-14"
5 ---
6
```

Most of the time you won't need to change this, but it can be nice sometimes when you want some fancier stuff like a table of contents.

Code in R Markdown

So whenever we want to insert code, we'll need to begin the **code chunk** with ````${code}```` and end it with `````. Note that you will want to give each of these their own line! That ``` character is in the top-left of your keyboard, it's the key you use to type a tilde (~). I call this little thingy a **backtick** but it probably has a cooler name that I'm not cool enough to call it.

For example, all the code chunks you've seen in previous comments were generated this way. The below is generated in just this way!

```
aVector<-c(1,2,3,4,4,4,4)
print(aVector)
```

```
## [1] 1 2 3 4 4 4 4
```

```
aVector
```

```
## [1] 1 2 3 4 4 4 4
```

```
mean(aVector)
```

```
## [1] 3.142857
```

So that's what default behaviors look like.

Inline Code

Inline code is the key to fancy, quickly adjustable files filled with all the important numbers management wants to see. As a silly example, maybe management *really* cares how many rows are in a dataset. While management cares *a lot*, I really don't care at all. Were I to know the number of rows, my life would go on totally unchanged save for one extra useless fact in my head. Let's take the `iris` dataset as an example. Do I know how many rows there are, even roughly? Absolutely not! Yet somehow I can confidently state that there are 150 rows. Maybe I want to get a little more specialized and say how many "setosa" irises were measured. Again, no clue how many there are and again I can say there are *exactly* 50 such observations. Why should *I* look at the data directly when the computer is so much better at it? It's all well and good for me to *tell* you that I totally didn't know this numbers beforehand, but I'll offer you proof in the form of a peek behind the veil at how inline code was used:

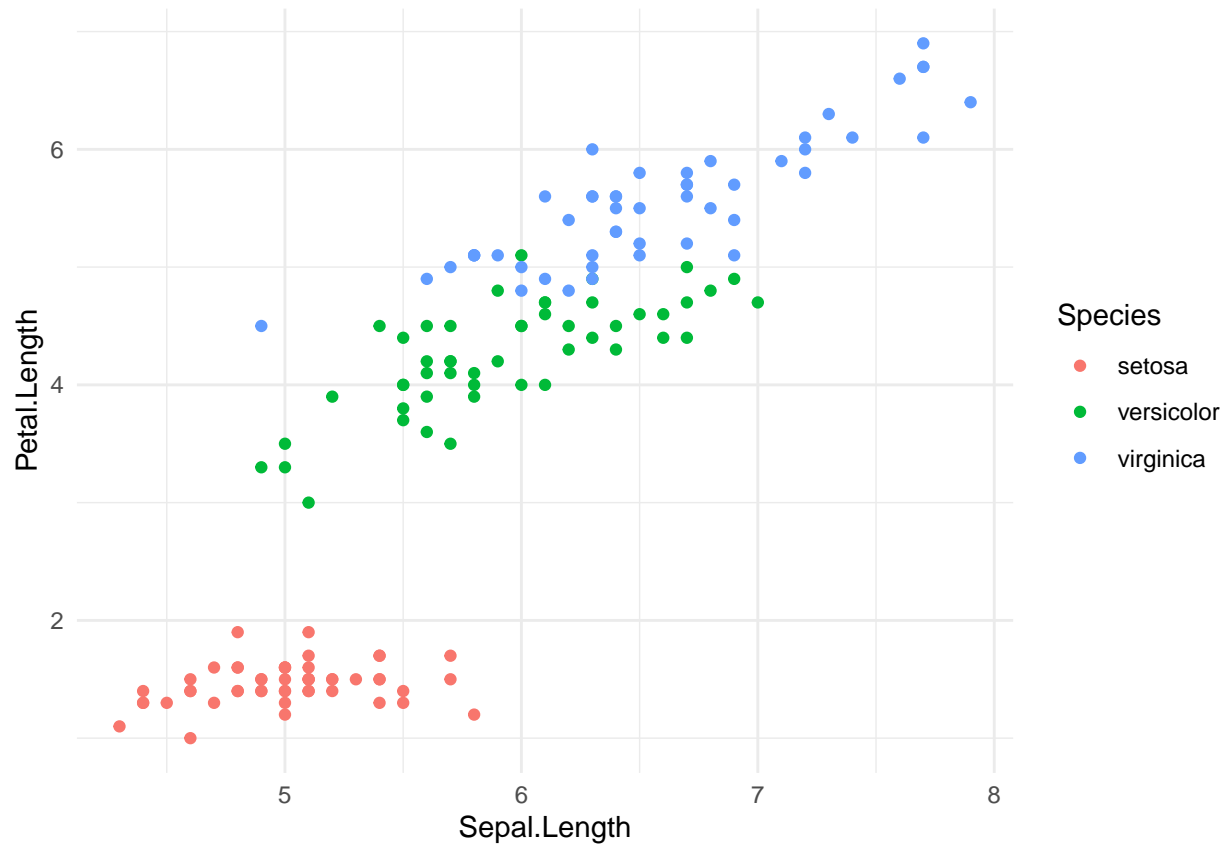
```
example. Do I know how many rows there are, even roughly? Absolutely
not! Yet somehow I can confidently state that there are `r nrow(iris)`
rows. Maybe I want to get a little more specialized and say how many
"setosa" irises were measured. Again, no clue how many there are and
again I can say there are *exactly* `r sum(iris$Species=='setosa')` such
observations. Why should *I* look at the data directly when the computer
is so much better at it? It's all well and good for me to *tell* you
that I totally didn't know this numbers beforehand, but I'll offer you
proof in the form of a peek behind the veil at how inline code was used:
56
```

Code Chunk Options

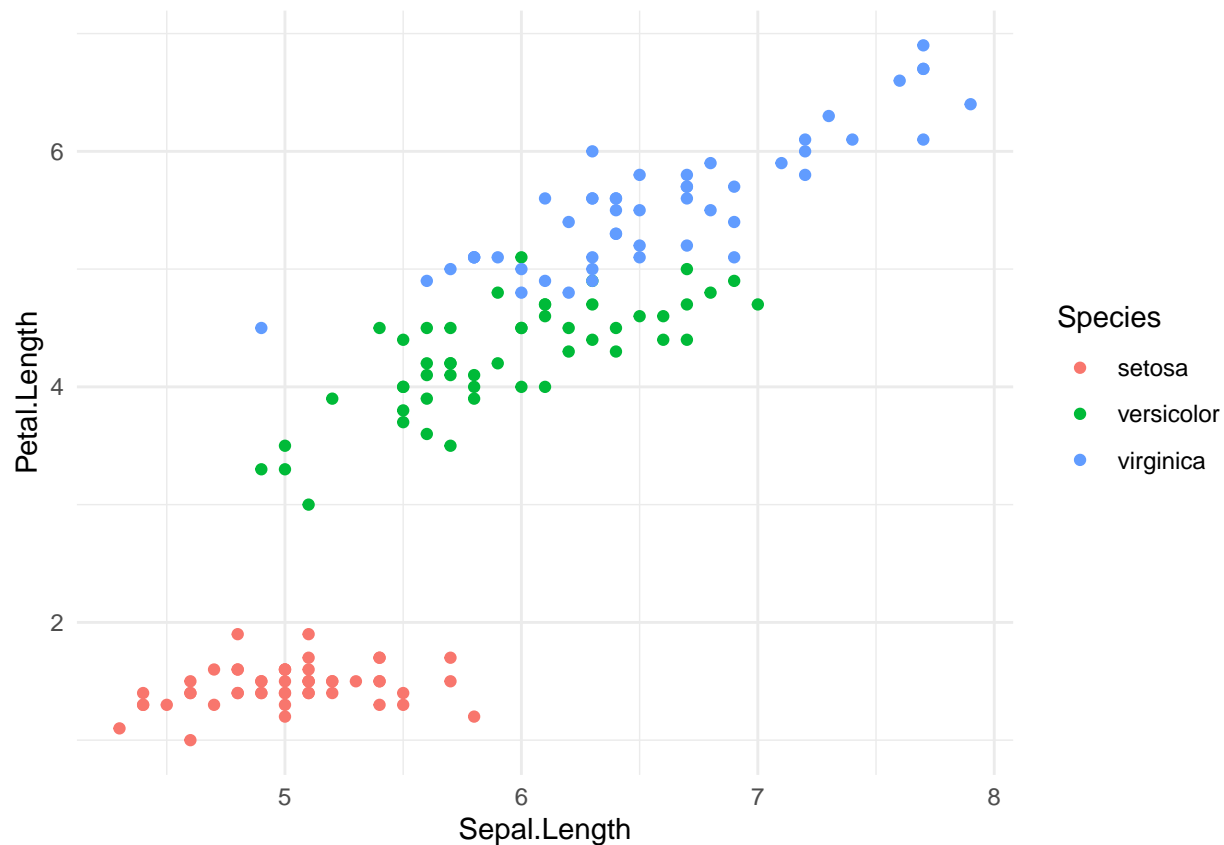
When working on a dynamic document, one of the best things is getting graphs quickly and easily in a way that *doesn't* require you to save it elsewhere first. We'll keep using the `iris` dataset because it's convenient for examples, but understand you can do this with anything!

Check out the code below and the graph it makes.

```
library(ggplot2)
ggplot(data=iris) +
  geom_point(aes(x=Sepal.Length, y=Petal.Length, color=Species)) +
  theme_minimal()
```



Now, see that if I don't want anyone to see my code (management doesn't care about code!), I can still get the same thing!



This is thanks to the `echo` option in the code chunk! With this you can set `echo=TRUE` or `echo=FALSE` to decide whether you do or do not want to show your code. You can also set `error=TRUE` so that your output (HTML or PDF) file will be made even in the presence of errors! This can be a big deal when trying to turn in hard assignments where things just aren't quite working out how you wanted. I actually used this in previous walkthroughs to show you some examples of errored code!

```

75
76 ```{r, echo=FALSE}
77 library(ggplot2)
78 ggplot(data=iris) +
79   geom_point(aes(x=Sepal.Length,y=Petal.Length,color=species)) +
80   theme_minimal()
81 ```
82

```

There are a lot of options that are useful in different contexts *and* are covered in **the R Markdown cheatsheet**! So click that and check it out there whenever things aren't quite how you want them to be.