Creating an R Markdown Document 2

Document Structure 5

The YAML Metadata 6

The Code Chunks 7

Markdown Text 9

Inline R Code 11

Running Code Chunks Interactively 13

Quarto 14

Conclusion 15



Writing Reports in R Markdown

Imagine this: You’ve collected surveys about customer satisfaction with your new product. Now you’re ready to analyze the data and write up your results. Your workflow looks something like this:

1. Download your data from Google Sheets and import it into a statistical analysis tool like SPSS.
2. Use SPSS to clean and analyze your data.
3. Export summaries of your data as Excel spreadsheets.
4. Use Excel to make some charts.
5. Write your report in Word, copying in your charts from Excel along the way.

Sound familiar? If so, you’re not alone. Many people use this workflow for data analysis. But what happens when, the next month, new surveys roll in, and you have to redo your report? Yup, back through steps one through five. You might think this multi-tool process works for one-time project, but let’s be honest: few projects are really one-time. For example, you might realize you forgot to include a few surveys in your original analysis or catch a mistake.

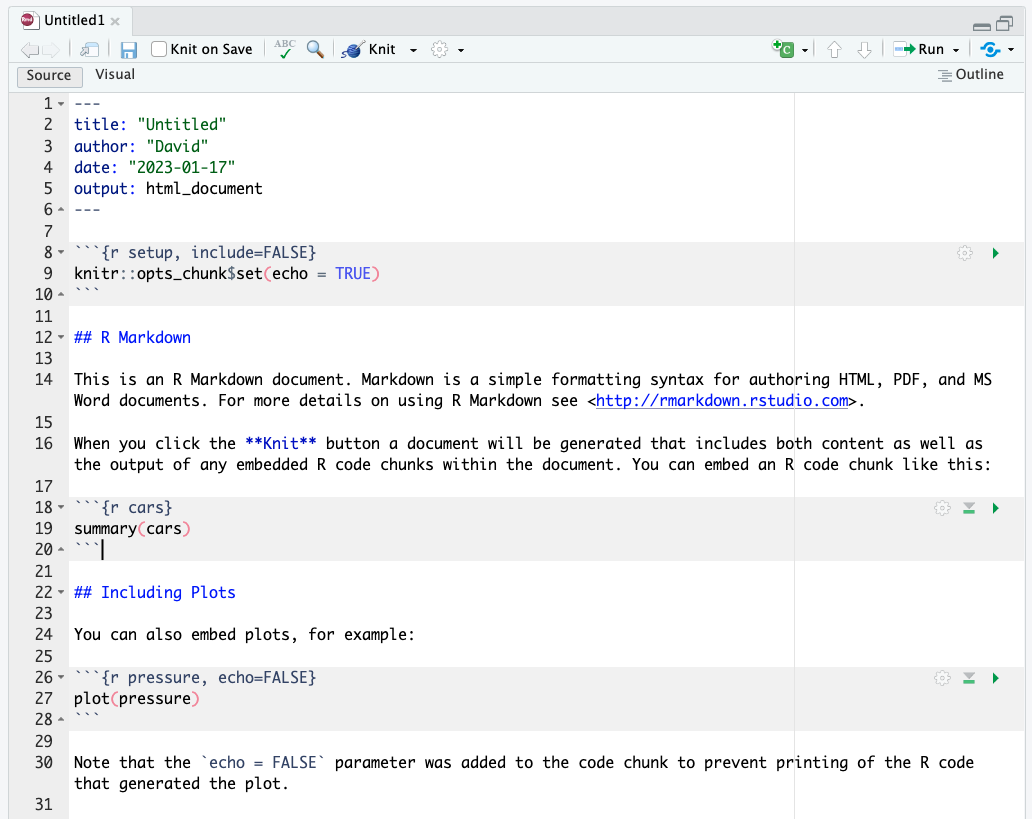
R Markdown combines data analysis, data visualization, and other R code with narrative text to create a document that can be exported to many formats, including Word, PDF, and HTML, so that you can share them with non-R users. When you use a single tool, your workflow becomes way more efficient. Need to recreate that January report in February to see if customers are liking your product more this month? Just rerun your code and you’ve got a new report, complete with the newest data. Need to fix an error in your analysis? Adjust your code, rerun it, and your corrected report is ready to go.

In this chapter, we’ll break down the pieces of R Markdown documents, then talk about some potential pitfalls and best practices. You’ll learn how to work with YAML metadata, R code chunks, and Markdown-formatted text, create in-line R code that can change the report’s text dynamically, and run the document’s code in various ways.

Creating an R Markdown Document

To create an R Markdown document in RStudio, go to File ⏵ New File ⏵ R Markdown. Choose a title, author, and date, as well as your default output format (HTML, PDF, or Word). All of these values can be changed later. Press OK, and RStudio will create an R Markdown document with some placeholder content, as seen in Figure 6-1.

[F06001.png]



* + - * 1. The placeholder content in a new R Markdown document

My first step is always to delete the content and replace it with my own. Let’s create a report about penguins using data from the palmerpenguins package. I’ve broken it into pieces by year, and we’ll use just the 2007 data. Here is the content I’ll add to my R Markdown document.

---

title: "Penguins Report"

author: "David"

date: "2024-01-12"

output: word\_document

---

```{r setup, include = FALSE}

knitr::opts\_chunk$set(include = TRUE,

echo = FALSE,

message = FALSE,

warning = FALSE)

```

```{r}

library(tidyverse)

```

```{r}

penguins <- read\_csv("https://data.rwithoutstatistics.com/penguins-2007.csv")

```

# Introduction

We are writing a report about the \*\*Palmer Penguins\*\*. These penguins are \*really\* amazing. There are three species:

- Adelie

- Gentoo

- Chinstrap

## Bill Length

We can make a histogram to see the distribution of bill lengths.

```{r}

penguins %>%

ggplot(aes(x = bill\_length\_mm)) +

geom\_histogram() +

theme\_minimal()

```

```{r}

average\_bill\_length <- penguins %>%

summarize(avg\_bill\_length = mean(bill\_length\_mm,

na.rm = TRUE)) %>%

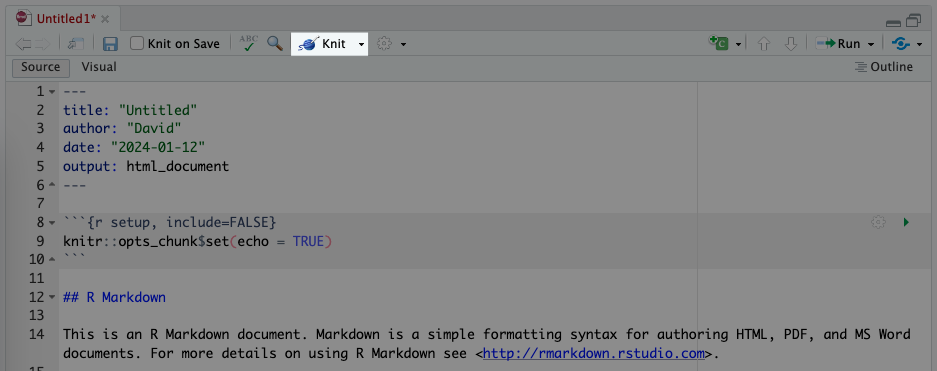
pull(avg\_bill\_length)

```

The chart shows the distribution of bill lengths. The average bill length is `r average\_bill\_length` millimeters.

This document has several pieces, each of which we will discuss below. First, though, let’s skip straight to the finish line by doing what’s called *knitting* our document (also known as rendering or, in plain English, exporting). The Knit button at the top of RStudio converts the R Markdown document into whatever format we selected upon creating it (Figure 6-2).

[F06002.png]



* + - * 1. The knit button in RStudio

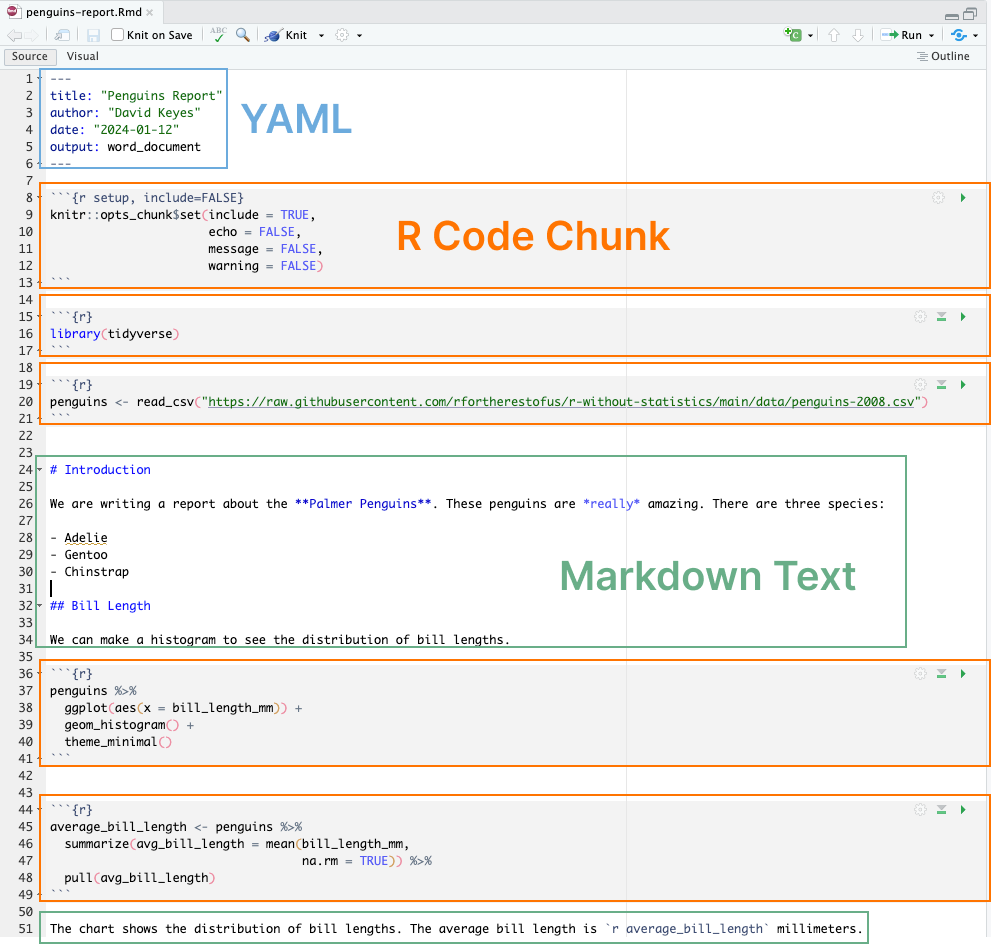
We’ve set the output format to be Word (see the output\_format: word\_document line), so you should now have a Word document. Some features were not visible in R Markdown but should appear in Word: the histogram, for example. This is because the R Markdown document doesn’t directly include this plot. Rather, it includes the code needed to produce the plot when knitted.

It may seem convoluted to constantly knit R Markdown documents to Word, but this workflow allows us to update our reports at any point with new code or data. This ability is known as *reproducibility*, and it is central to the value of R Markdown.

Document Structure

All R Markdown documents have three main pieces: one YAML section, multiple R code chunks, and sections of Markdown text. Figure 6-3 shows these parts of an R Markdown document.

[F06003.png]



* + - * 1. All pieces of an R Markdown document

Let’s take these pieces one at a time.

The YAML Metadata

The YAML section is the very beginning of an R Markdown document. (The name YAML comes from the recursive acronym *YAML ain’t markup language*, whose meaning isn’t important for our purposes.) Three dashes indicate its beginning and end, and the text inside of it contains metadata about the R Markdown document. Here is my YAML:

---

title: "Penguins Report"

author: "David Keyes"

date: "2024-01-12"

output: word\_document

---

As you can see, it provides the title, author, date, and output format. All elements of the YAML are given in *key: value* syntax, where key is the piece of metadata (for example, the title) followed by its value in quotes.

The Code Chunks

R Markdown documents have a different structure from the R script files you might be familiar with (those with the *.R* extension). R script files treat all content as code unless you comment out a line by putting a pound sign (#) in front of it. In the following code, the first line is a comment while the second line is code.

```{r}

# Import our data

data <- read\_csv("data.csv")

```

In R Markdown, the situation is reversed. Everything after the YAML is treated as text unless we specify otherwise by creating what are known as *code chunks*. These start with three back ticks (```), followed by the lowercase letter *r* surrounded by curly brackets ( {} ). Another three back ticks indicate the end of the code chunk:

```{r}

library(tidyverse)

```

If you’re working in RStudio, code chunks should have a light gray background.

Anything in the code chunk is treated as R code when we knit. For example, this code chunk will produce a histogram in the final Word document.

```{r}

penguins %>%

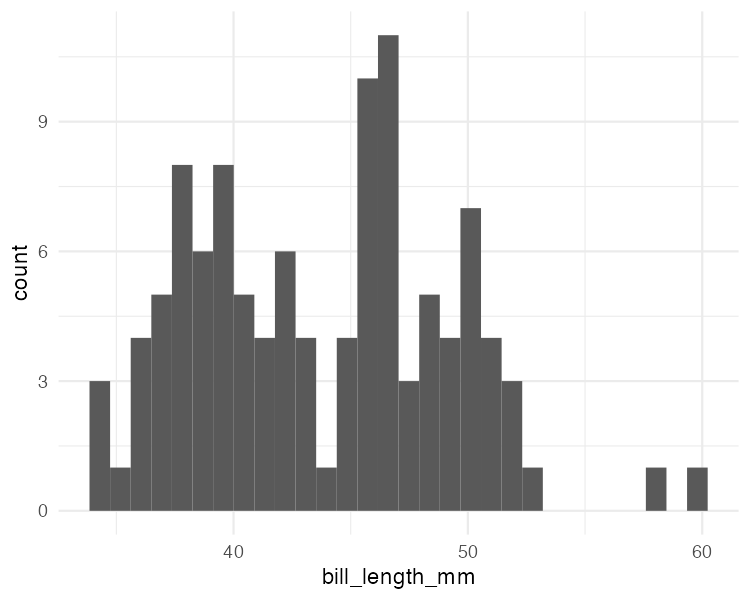
ggplot(aes(x = bill\_length\_mm)) +

geom\_histogram() +

theme\_minimal()

```

The histogram made from this code can be seen in Figure 6-4.



* + - * 1. A simple histogram

A special code chunk at the top of each R Markdown document, known as the *setup code chunk*, gives instructions for what should happen when knitting a document. It contains the following code chunk options:

echo

Do you want to show the code itself in the knitted document?

Include

Do you want to show the output of the code chunk?

Message

Do you want to include any messages that code might generate? For example, this message shows up when you run library(tidyverse):

── Attaching core tidyverse packages ───── tidyverse 1.x.x──

✔ dplyr 1.x.x ✔ readr 2.x.x

✔ forcats 0.x.x ✔ stringr 1.x.x

✔ ggplot2 3.x.x ✔ tibble 3.x.x

✔ lubridate 1.x.x ✔ tidyr 1.x.x

✔ purrr 1.x.x

── Conflicts───── tidyverse\_conflicts() ──

✖ dplyr::filter() masks stats::filter()

✖ dplyr::lag() masks stats::lag()

warning

Do you want to include any messages that the code might generate? For example, here is the message you get when creating a histogram using geom\_histogram():

`stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.

In cases where you’re using R Markdown to generate a report for a non-R user, you likely want to hide the code, messages, and warnings but show the output (which would include any visualizations you generate). To do this, create a setup code chunk that looks like this:

```{r setup, include = FALSE}

knitr::opts\_chunk$set(include = TRUE,

echo = FALSE,

message = FALSE,

warning = FALSE)

```

The include = FALSE option on the first line applies to the setup code chunk itself. It tells R Markdown to not include the output of the setup code chunk on knitting. The options within knitr::opts\_chunk$set() apply to all future code chunks. However, you can also override these global code chunk options on individual chunks. If I wanted my Word document to show both the plot itself and the code used to make it, I could set echo = TRUE for that code chunk only:

```{r echo = TRUE}

penguins %>%

ggplot(aes(x = bill\_length\_mm)) +

geom\_histogram() +

theme\_minimal()

```

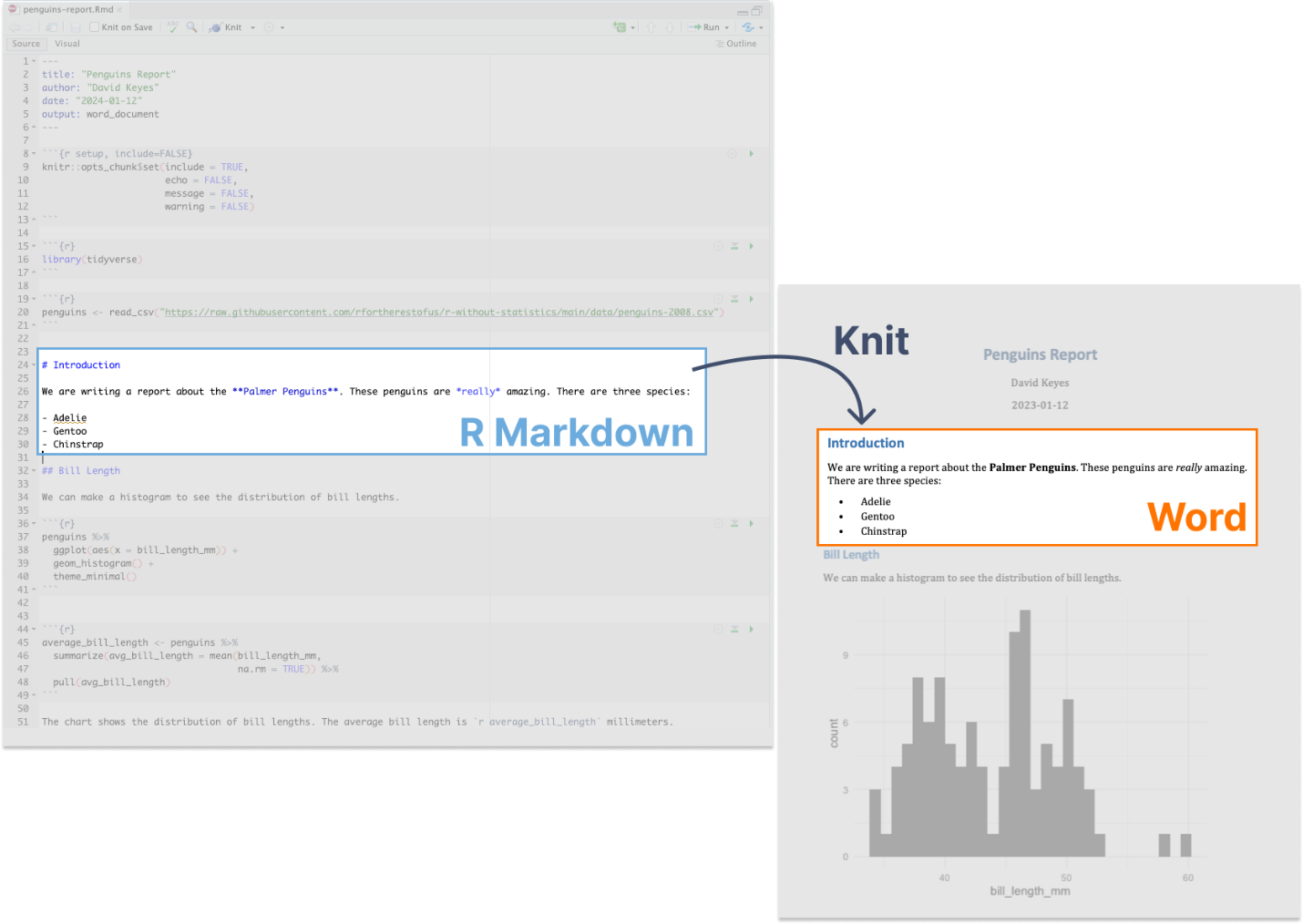
Because include is already set to TRUE within knitr::opts\_chunk$set() in the setup code chunk, I don’t need to specify it again.

Markdown Text

Markdown is a way to style plain text. If you were writing directly in Word, you could just press the B button to make text bold, for example, but R doesn’t have such a button. If you want your knitted Word document to include bold text, you need to use Markdown indicate this in the document.

Markdown text sections (which have a white background in R Studio) will be converted into formatted text in the Word document after knitting. Figure 6-5 highlights the equivalent sections in our R Markdown and Word documents.

[F06005.png]



* + - * 1. Markdown text in R Markdown and its equivalent in a knitted Word document

As you can see, the text # Introduction in R Markdown gets converted to a first-level heading, while ## Bill Length becomes a second-level heading. By adding hashes, you can create up to six levels of headings. In RStudio, headers are easy to find because they show up in blue.

Text without anything before it becomes body text in Word. To create italic text, add single asterisks around it (\*like this\*). To make text bold, use double asterisks (\*\*Palmer Penguins\*\*).

You can make bulleted lists by placing a dash at the beginning of a line and adding your text after it:

- Adelie

- Gentoo

- Chinstrap

To make ordered lists, replace the dashes with numbers. You can either number each line consecutively or, as I’ve done below, just repeat 1. In the knitted document, the proper numbers will automatically generate.

1. Adelie

2. Gentoo

3. Chinstrap

Formatting text in Markdown might seem more complicated than doing so in Word. But if we want to switch from a multi-tool workflow to a reproducible R Markdown-based workflow, we need to remove all manual actions from the process so we can easily repeat it in the future.

Inline R Code

R Markdown documents can also include little bits of code within Markdown text. To see how this inline code works, take a look at the following sentence of the R Markdown document:

The average bill length is `r average\_bill\_length` millimeters.

Inline R code begins with a backtick and the lowercase letter *r* and ends with another backtick. Here, it tells R to print the value of the variable average\_bill\_length, which we’ve defined as follows in the code chunk above the inline code:

```{r}

average\_bill\_length <- penguins %>%

summarize(avg\_bill\_length = mean(bill\_length\_mm,

na.rm = TRUE)) %>%

pull(avg\_bill\_length)

```

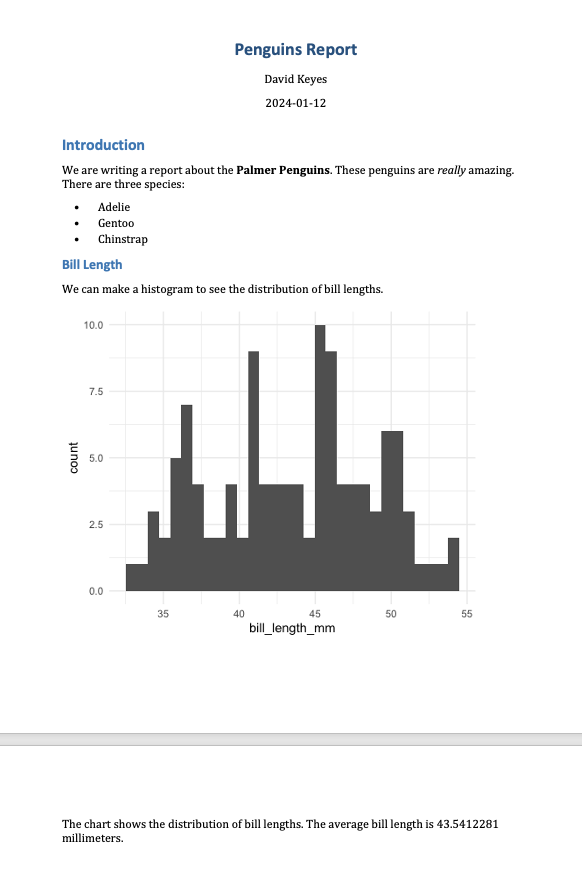
This code calculates the average bill length and saves it as average\_bill\_length. Having created this variable, I can then use it in my inline R code. As a result, the Word document includes the sentence “The average bill length is 43.9219298.”

One benefit of using inline R code is that you avoid having to copy and paste values, which is error-prone. Inline R code also makes it possible to automatically calculate values on the fly whenever we re-knit the R Markdown document with new data. To show you how this works, let’s make a new report using data from 2008. To do this, I need to change only one line, the one that reads the data:

penguins <- read\_csv("https://data.rwithoutstatistics.com/penguins-2008.csv")

Now that I’ve switched *penguins-2007.csv* to *penguins-2008.csv*, I can re-knit my report and get a new Word document, complete with updated results. Figure 6-8 shows our new knitted Word document.

[F06007.png]



* + - * 1. The knitted Word document with 2008 data

The histogram is based on 2008 data, as is the average bill length of 43.5412281. These update automatically because every time I hit knit, the code is rerun, regenerating plots and recalculating values. As long as the data has the same structure, updating a report requires just a click of the knit button.

Running Code Chunks Interactively

You can run the code within an R Markdown document in two ways. The first is by knitting the entire document. The second way is to run code chunks manually (also known as *interactively*) by hitting the little green play button at the top-right of a code chunk. The down arrow next to it will run all code until that point. We can see these buttons in Figure 6-7.

[F06007.png]



* + - * 1. The buttons on code chunks in RStudio

You can also use the Cmd key on Mac and the Ctrl key on Windows, followed by Enter, to run pieces of code, just like in an R script file. Running code interactively is a good way to test that portions of code work before you knit the entire document.

The one downside to running code interactively is that you can sometimes make mistakes that make your R Markdown document fail to knit. That is because, in order to knit, an R Markdown document must contain all the code it uses. If you are working interactively and, say, load data in a separate file, you will be unable to knit your R Markdown document. When working in R Markdown, always keep all code within a single document.

The code must also be in the right order. An R Markdown document that looks like this, for example, will give you an error if you try to knit it:

---

title: "Penguins Report"

author: "David Keyes"

date: "2024-01-12"

output: word\_document

---

```{r setup, include=FALSE}

knitr::opts\_chunk$set(include = TRUE,

echo = FALSE,

message = FALSE,

warning = FALSE)

```

```{r}

penguins <- read\_csv("https://data.rwithoutstatistics.com/penguins-2008.csv")

```

```{r}

penguins %>%

ggplot(aes(x = bill\_length\_mm)) +

geom\_histogram() +

theme\_minimal()

```

```{r}

library(tidyverse)

```

This happens because you are attempting to use tidyverse functions (read\_csv(), as well as various ggplot functions) before you load the tidyverse package.

Alison Hill, one of the most prolific R Markdown educators, tells her students to “knit early and often.” This practice makes it easier to isolate issues that make knitting fail. Hill describes her typical R Markdown workflow as spending 75 percent of her time working on a new document and 25 percent of her time knitting to check that the R Markdown document works.

Quarto

In 2022, a new publishing tool similar to R Markdown was released. Known as Quarto, this tool takes what R Markdown has done for R and extends it to other languages, including Python, Julia, and Observable JS. As I write this book, Quarto is starting to gain more traction. Luckily, the concepts you’ve learned about in this chapter apply to Quarto as well. Quarto documents have a YAML section, code chunks, and Markdown text. You can export Quarto documents to HTML, PDF, and Word documents. There are some minor differences in syntax between R Markdown and Quarto documents, but if you know how to use R Markdown, you should easily pick up Quarto as well. The documentation at *https://quarto.org/* is a great place to read more about all of the Quarto features and learn how to get started using it.

Conclusion

We started this chapter with the example of a report that needs to be regenerated monthly. Using R Markdown, we can reproduce this report every month without changing our code. Even if we lost the final Word document, we could quickly recreate it.

Best of all, working with R Markdown makes it possible to do things in seconds that would have previously taken hours. In a world where making a single report requires three tools and five steps, you may not want to work on it. As a research scientist who used R Markdown regularly, Alison Hill says it enabled her to work on reports before she had received all of the data. She could write code that worked with partial data and rerun it with the final data at any time.

In this chapter, we’ve just scratched the surface of what R Markdown can do. The next chapter will show how to use it to instantly generate hundreds of reports. Magic indeed!