Use R Markdown to Communicate Accurately and Efficiently

Imagine this scenario. It’s January, you’ve collected some surveys, and now you’re ready to do your data analysis and write up your results. Your workflow looks something like this:

1. Download your data from Google Sheets import it into SPSS.
2. Use SPSS to do your data cleaning and analysis.
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. It’s the workflow that many people use when doing data analysis. But what happens when new surveys roll in and you have to redo your analysis in February? Yup, back through steps one through five. Same thing in March. And April. You get the idea.

You may be thinking: okay, but the multi-tool workflow can work for a one-time project, right? Maybe, but let’s be honest: few projects are really one-time. Either you have to redo them the next month or you realize you forgot to include a few surveys in your original analysis or you realize you made a mistake in SPSS that means you need to redo steps one through six.

R Markdown is the solution to this problem. It combines data analysis, data visualization, and anything you can do with code (all the pieces you might do in SPSS and Excel in the multi-tool workflow) alongside narrative text (the Word piece). Your combination code and text R Markdown document can be exported to many formats, including Word, PDF, and HTML, so that you can share them with non-R users.

By combining all of the pieces from the multi-tool workflow in a single tool, your workflow becomes way more efficient. Need to recreate that January report in February? Just rerun your code and you’ve got a new report, complete with the newest data. Need to add in a few surveys you missed at any point? Rerun your code and generate a new report. Need to fix an error in your analysis? Make the fix to your code, rerun it, and your corrected report is ready to go.

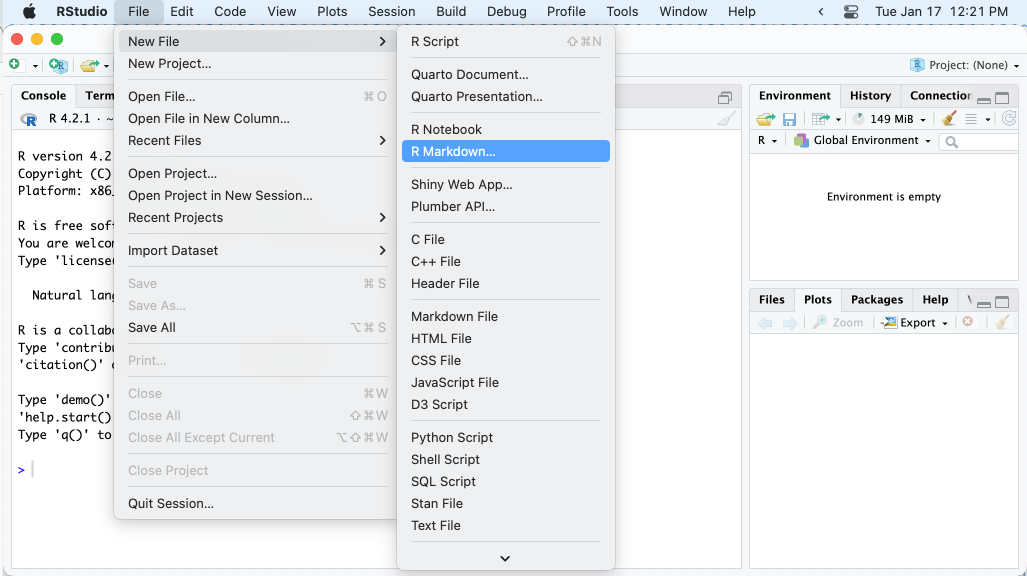
I interviewed Alison Hill to learn more about R Markdown from both a technical and conceptual perspective. Hill is currently Director of Knowledge, Product Strategy at Voltron Data, but previously worked at Posit (the company that makes the RStudio editor) doing all sorts of development and education work on R Markdown. Bits of her wisdom are sprinkled throughout, both in direct quotes and general knowledge about how R Markdown works.

In this chapter, we’ll begin by breaking down the pieces of R Markdown documents. We’ll then talk about some potential pitfalls and best practices when using R Markdown. And we’ll conclude with some reflections on the value of the R Markdown workflow.

How R Markdown Works

To create an R Markdown document while working in RStudio, you go to File > New File > R Markdown. You can choose the title, author, and date as well as your default output format (HTML, PDF, or Word). All of these can be changed later. Figure 6-1 shows how to create a new R Markdown document.

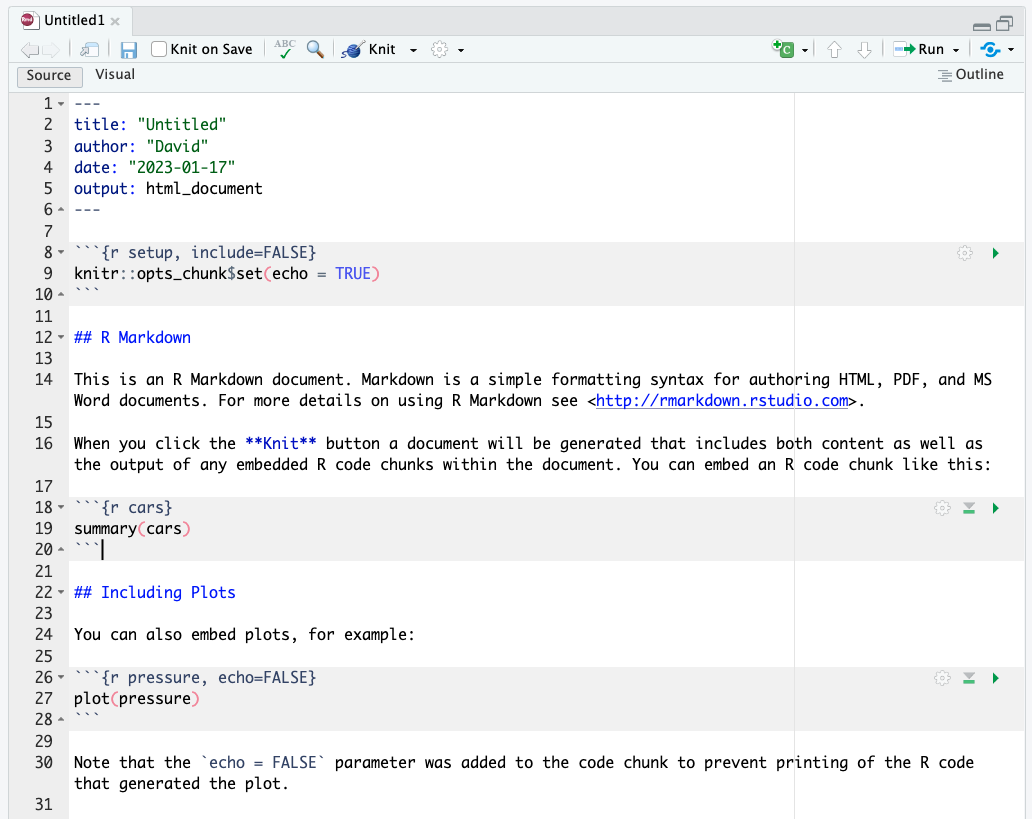
[F06001.png]



* + - * 1. How to create a new R Markdown file

Hit OK and RStudio will create an R Markdown document with some placeholder content, as seen in Figure 6-2.

[F06002.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. I’m going to create a report on penguins using data from the palmerpenguins package (for reasons that will become clear later, I’ve broken it into pieces by year; for now we’re just using 2007 data). Here is the content I’ll add to my R Markdown document.

---

title: "Penguins Report"

author: "David"

date: "2023-01-12"

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://raw.githubusercontent.com/rfortherestofus/r-without-statistics/main/data/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)) %>%

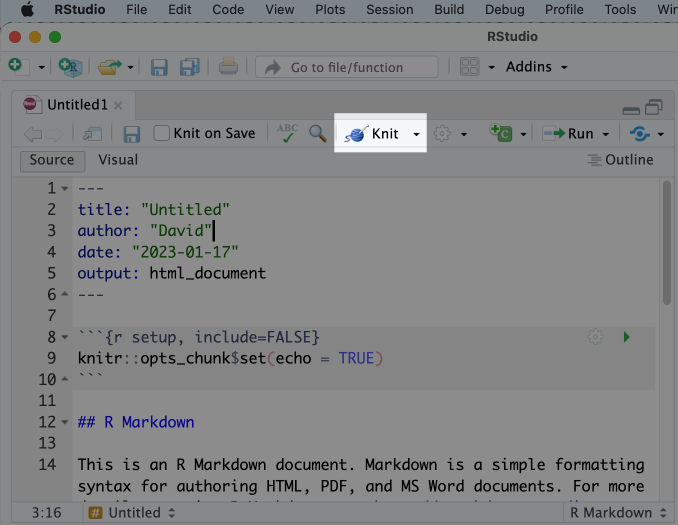
pull(avg\_bill\_length)

```

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. For now, though, let’s skip straight to the finish line by doing what’s called knitting (also known as rendering or, in plain English, exporting) our document. If we hit the knit button at the top of RStudio, our R Markdown document is converted into whatever format we selected upon creating it. Figure 6-3 shows where to find the Knit button.

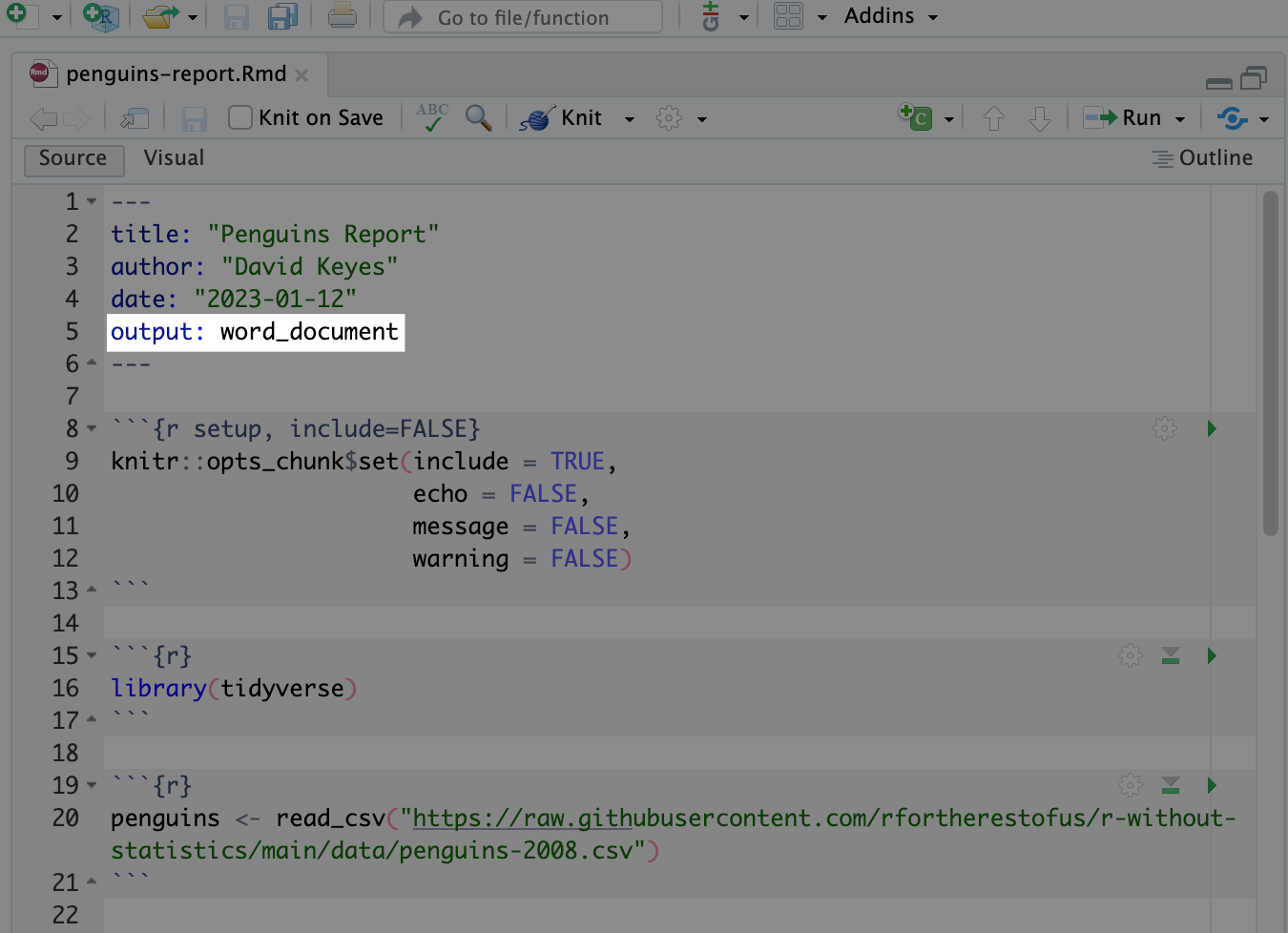
[F06003.png]



* + - * 1. The knit button in RStudio

I set the output format to be Word (see the line at the top that says “output\_format: word\_document” so I now have a Word document. You can see what it looks like in Figure 6-4.

[F06004.png]



* + - * 1. An R Markdown document with the output format set to Word

There are some things that were not visible in R Markdown but are in Word (the histogram, for example). This is because our R Markdown document does not include this plot directly, but instead has the code needed to produce the plot, which is then produced when we knit our document.

It may seem convoluted to constantly knit R Markdown documents to Word, but this workflow is what allows us to update our reports at any point with updated code or new data. This is known as reproducibility, and it is central to the value of R Markdown. But before we discuss reproducibility in depth, let’s make sure we understand how R Markdown documents are structured.

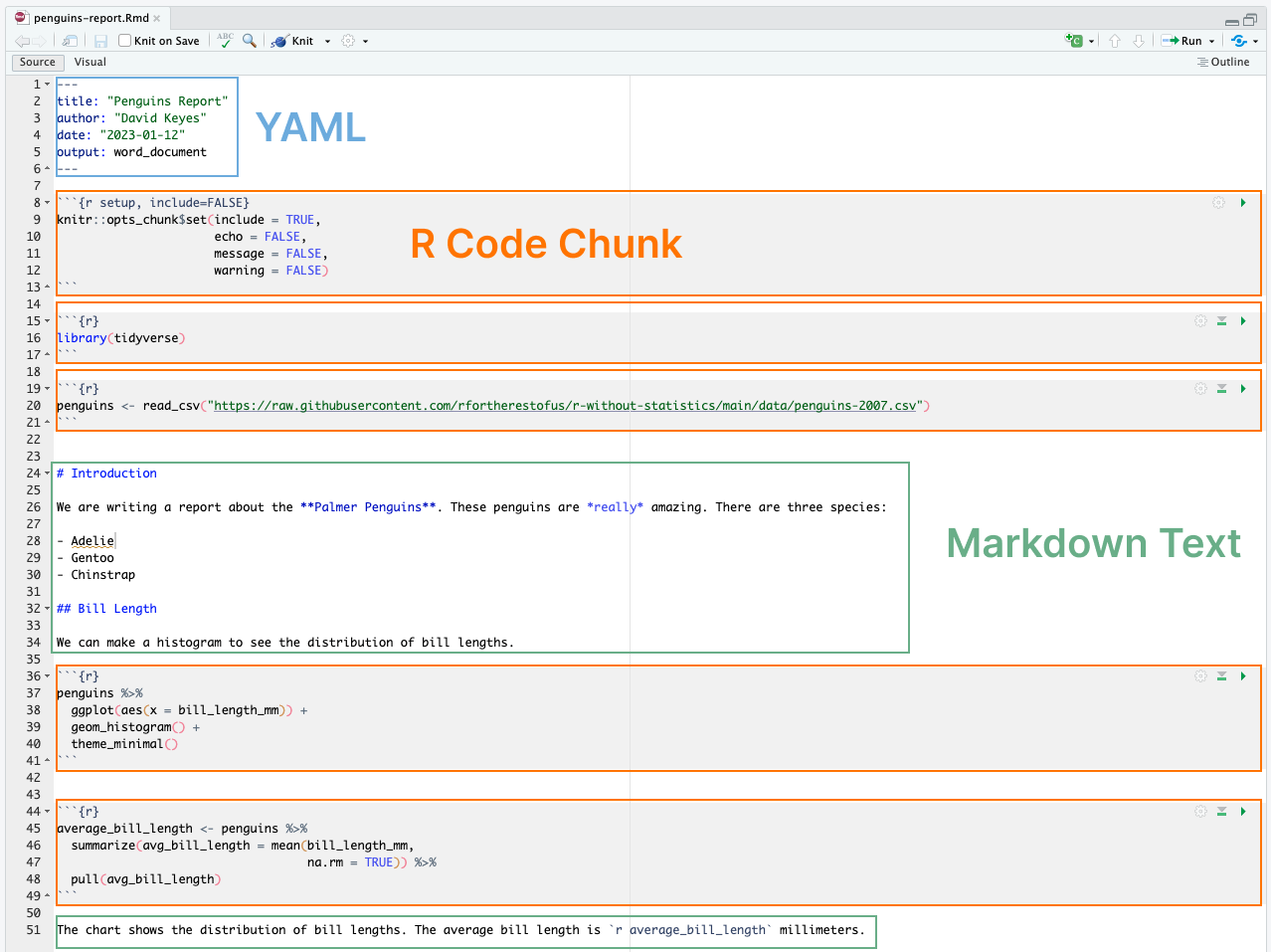
R Markdown Documents Structure

All R Markdown documents have three main pieces:

1. YAML
2. R code chunks
3. Markdown text

Each document has one YAML, but multiple code chunks and sections of markdown text. Figure 6-5 shows all of the pieces of an R Markdown document.

[F06005.png]



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

Let’s take these pieces one at a time.

The YAML

The YAML is the very first piece at the top of our R Markdown document (the name YAML comes from the recursive acronym “YAML ain’t markup language” whose meaning is not important for our purposes). With three dashes indicating its beginning and end, the text inside of the YAML contains metadata about the R Markdown document. My YAML, shown below, tell us the title, author, date, and the output format for when we knit.

---

title: "Penguins Report"

author: "David Keyes"

date: "2023-01-12"

output: word\_document

---

R Code Chunks

Listen to the name R Markdown and you get a sense of the other two pieces of an R Markdown document: R code and markdown text. Let’s discuss the R code first.

If you’ve only ever worked in R script files (they use the extension .R), you’ll need to reorient your thinking. In R script files everything is treated as code unless you comment it out by putting a # in front of it. In the code below, 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. After the YAML, everything is treated as text unless we specify otherwise. And we do this specifying by creating what are known as code chunks. These start with three back ticks, followed by the lower case letter r surrounded by curly brackets. Three back ticks indicate the end of the code chunk.

```{r}

library(tidyverse)

```

Working in RStudio, code chunks have a light gray background. Anything between the beginning and end of the code chunk is treated as R code when we knit. The code in this code chunk will give us a histogram in our Word document.

```{r}

penguins %>%

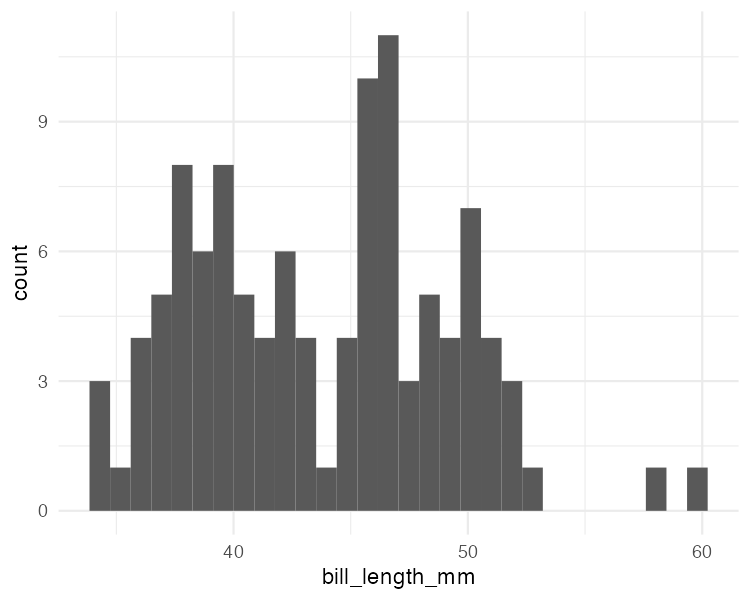
ggplot(aes(x = bill\_length\_mm)) +

geom\_histogram() +

theme\_minimal()

```

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



* + - * 1. A simple histogram

Code Chunk Options

There is one special code chunk at the top of each R Markdown document. It is known as the setup code chunk and it gives instructions for what happens when knitting a document. These instructions, more commonly known as code chunk options, determine what happens when we knit our document:

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

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

message: Do you want to include any messages that code generates? An example of such messages include this message that shows up when you run library(tidyverse):

── Attaching core tidyverse packages ───── tidyverse 1.3.2.9000 ──

✔ dplyr 1.0.10 ✔ readr 2.1.3

✔ forcats 0.5.2 ✔ stringr 1.5.0

✔ ggplot2 3.4.0 ✔ tibble 3.1.8

✔ lubridate 1.9.0 ✔ tidyr 1.2.1

✔ purrr 1.0.1

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

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

✖ dplyr::lag() masks stats::lag()

warning: Do you want to include any messages that code generates? Here is the message you get when creating a histogram using geom\_histogram():

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

This may all feel very confusing so let’s keep it simple. 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 output. To do this, have your setup code chunk look like mine in the report I’m creating:

```{r setup, include = FALSE}

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

echo = FALSE,

message = FALSE,

warning = FALSE)

```

These options in the setup code chunk apply to all code chunks below, unless you override them individually. If I wanted my Word document to show both the plot itself as well as the code used to make it, I could set echo = TRUE for that code chunk (include is already set to TRUE in my setup code chunk so I don’t need to specify that again).

```{r echo = TRUE}

penguins %>%

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

geom\_histogram() +

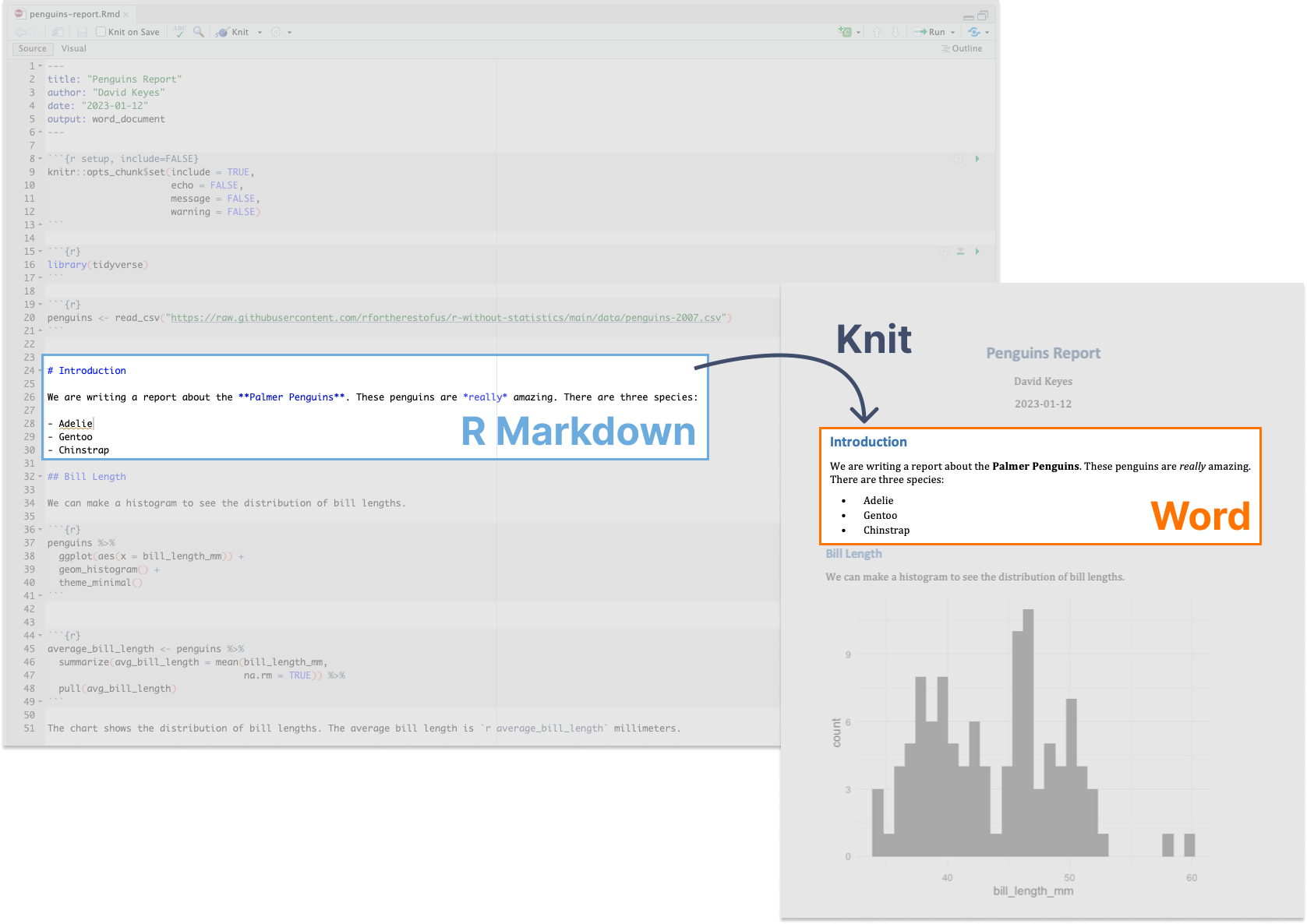
theme\_minimal()

```

Markdown Text

We’ve discussed the R portion of R Markdown. Let’s now discuss markdown. Markdown is a way to write using only plain text. If you look at the text sections (they have a plain white background) in our R Markdown document, you can see how markdown is converted into what we see in our Word document after knitting. Figure 6-7 highlights the equivalent sections in our R Markdown and Word documents.

[F06006.png]



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

The text # Introduction in R Markdown gets converted into a Heading 1 while ## Bill Length becomes a Heading 2. You can go all the way down to Heading 6 by using six hashes. Working in RStudio headers are easy to find because they show up in blue.

Text without anything before it becomes body text in Word. If you want to make words italics, add single asterisks around them (\*really\*). To make them bold, use double asterisks (\*\*Palmer Penguins\*\*).

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

- Adelie

- Gentoo

- Chinstrap

To make ordered lists you’d do the same but replacing the dashes with numbers. You can either put increasing numbers or, as I’ve done below, just keep repeating 1.

1. Adelie

1. Gentoo

1. Chinstrap

On knitting, the numbers will be converted to 1, 2, 3 like so:

1. Adelie
2. Gentoo
3. Chinstrap

You may be thinking at this point: this seems very complicated for just writing some simple text. Why do I need to learn a new way to write? Hitting the B button for bold has always worked just fine. The answer is that R cannot hit the B button for you to make text bold in the process of knitting. If you want your knitted Word document to have bold text, you need a way to tell R that in the R Markdown document itself. R Markdown documents are plain text so we need a way in plain text to make knitted text bold. If we want to switch from a multi-tool workflow to a single, reproducible R Markdown-based workflow, we need to remove all manual actions from the process. It may feel like a bit of a hassle to have to learn markdown, but it opens up all sorts of possibilities.

Inline R Code

I told you there were three main parts of an R Markdown document (YAML, code chunks, and markdown text). There’s one smaller piece called inline R code. As the name implies, inline R code allows us to use little bits of R code within markdown text. To explain how it works, let’s again compare our R Markdown document with the knitted Word document. Specifically, let’s look at the very end of each.

The last sentence of my R Markdown document looks like this:

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

But the last sentence of my Word document simply says, “The average bill length is 43.9219298.” How did this happen? To create inline R code, you add a backtick and the lower case letter r. Your R code goes next, followed by a back tick at the end. In my inline R code, I’m telling R to print the value of the variable average\_bill\_length. If you’re wondering where the variable comes from, take a look at the code chunk above the inline R code.

```{r}

average\_bill\_length <- penguins %>%

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

na.rm = TRUE)) %>%

pull(avg\_bill\_length)

```

In that code, I’m calculating the average bill length and saving it as average\_bill\_length. Having created this variable, I can then use it in my inline R code.

The benefit of inline R code is that you avoid having to copy and paste in the value of the average bill length. Copying and pasting is error-prone so using inline R code gets us around these issues. Just as important, using inline R code makes it possible to automatically calculate values on the fly whenever we re-knit our R Markdown document. To show you how this works, let’s make a new report using new data.

Example of Re-Running Code with New Data

The report that we’ve been working with up to this point had data from 2007. To show where R Markdown really shines, let’s create a new report using 2008 data. To do this, I only need to change one line. I go from this:

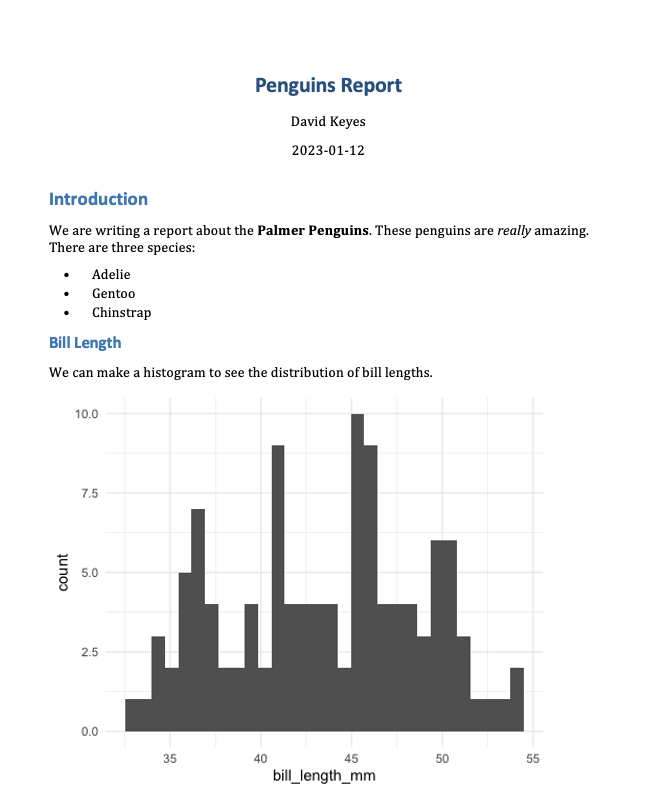
penguins <- read\_csv("https://raw.githubusercontent.com/rfortherestofus/r-without-statistics/main/data/penguins-2007.csv")

To this:

penguins <- read\_csv("https://raw.githubusercontent.com/rfortherestofus/r-without-statistics/main/data/penguins-2008.csv")

Now that I’ve switched penguins-2007.csv to penguins-2008.csv, I can re-knit my report. When I do so, everything runs and I 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 are updated 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, an updated report is just a click of the knit button away.

Best Practices for Working with R Markdown

Now that we’ve discussed how R Markdown works, let’s talk about how to use it well.

First, let’s discuss the two different ways your can run code within an R Markdown document. The first is what we’ve discussed in this chapter: knitting the entire document. A second way is to run code chunks manually (also known as interactively). Hitting the little green play button at the top right of a code chunk will run its contents. The down arrow next to it will run all code down to that point. We can see these buttons in Figure 6-9 below.

[F06008.png]



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

And you can hit command/control (on Mac/Windows) + enter to run pieces of code, just like in an R script file. Running code interactively is a good way to test that pieces 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 have all code within it. 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.

Related, all code must be in the right order. Knitting an R Markdown document causes code to be from top to bottom. 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: "2023-01-12"

output: word\_document

---

```{r setup, include=FALSE}

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

echo = FALSE,

message = FALSE,

warning = FALSE)

```

```{r}

penguins <- read\_csv("https://raw.githubusercontent.com/rfortherestofus/r-without-statistics/main/data/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. Figure 6-10 highlights the problem.

[F06009.png]



* + - * 1. Figure 6-10: An R Markdown document with code chunks in the wrong order

If you look back at the penguin report I made, it loaded the tidyverse, then imported data, then plotted. This R Markdown document can be knitted because everything is in the correct order.

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% of my time working and iterating on a new document and then 25% of my 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 is designed to take what R Markdown has done for R and extend it to other languages (Python, Julia, and Observable JS). As I write this book, Quarto is starting to gain more and more traction. Over time, I suspect it will likely replace R Markdown. But R Markdown will never go away completely (its developers have made this promise). And, just as importantly, the concepts that you’ve learned about in this chapter apply to R Markdown and to Quarto. Quarto documents have a YAML, 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 will be able to easily pick up Quarto as well. The documentation at quarto.org is a great place to read more about all of the Quarto features and learn how to get started using it.

In Conclusion: R Markdown Opens up All Sorts of Possibilities

We started this chapter with the example of a report that needs to be regenerated monthly. I hope the rest of the chapter has convinced you that using R Markdown is a better approach than the SPSS to Excel to Word dance that so many people use. R Markdown better because it is reproducible. We can reproduce a report every month because we’ve used a tool that makes this possible.

Working with R Markdown requires a mindset shift. In the multi-tool approach, you’d be devastated if you lost the Word document where you were writing your report. With R Markdown, it’s not the knitted Word document that matters – it’s the R Markdown document we care about. As Alison Hill puts it, “you preserve the source and you edit that and then you can render when you need to.” Or, as Jenny Bryan and Jim Hester, as part of their rstats.wtf workshop, put it in Figure ??:

[F06010.png]



* + - * 1. Figure 6-11: A meme explaining why you should save your source and not care about knitted documents

This mindset shift can take some getting used to, but, trust me, it’s worth the effort. Being able to produce reports on demand every month can, as illustrator Allison Horst suggests in Figure 6-12, lead others to see you as a wizard.

[F06011.png]



* + - * 1. Figure 6-12: An illustration by Allison Horst showing the power of R Markdown

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 be disinclined to work on it. But if you can generate reports on the fly with R Markdown, it opens up all sorts of possibilities. As Alison Hill puts it, “your ideas start to expand about what you can do and what you can make and what your job actually entails.”

Prior to working for Posit, Alison Hill was a research scientist who used R Markdown regularly. Reflecting on this period, Hill says that being able to produce updated reports on demand is one of the greatest benefits of R Markdown. She’d often receive data over time. With R Markdown, she could write code in a way that worked with partial data (allowing her to avoid the last-minute rush common in a multi-tool workflow) and could be re-run at any time. As she told me, “I knew I could go ahead and start working on it before the data was final, final, final, because you never know when it’s going to be final.”

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