Create Beautiful Presentations with R Markdown

Remember how we talked in Chapter 6 about the multi-tool workflow from SPSS to Excel to Word? That leaves out one of the most common tools in many people’s workflow: PowerPoint. Reporting happens in presentations as often as it does in Word-based reports.

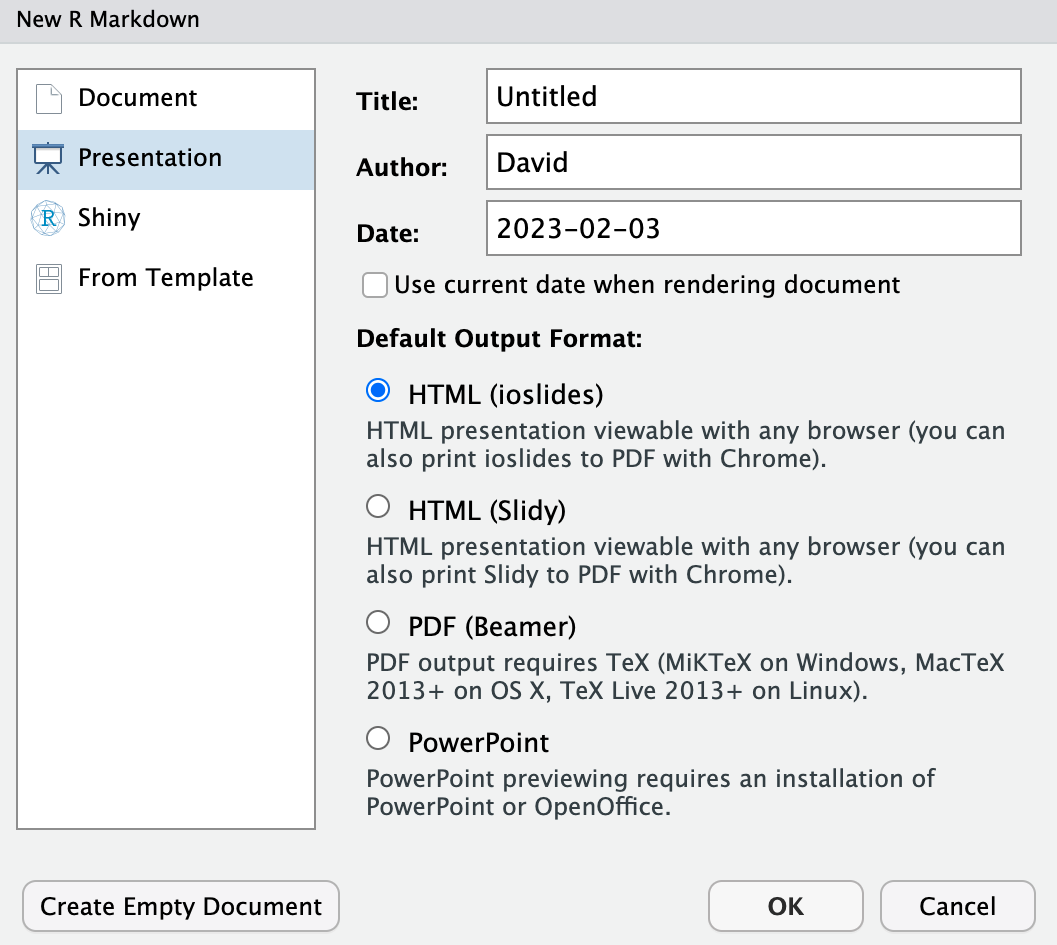
In Chapter 6, we talked about the R Markdown-based workflow having the benefit of keeping everything in one tool. You may be wondering at this point if creating presentations will require us to add another tool, like PowerPoint. Do you need to do your analysis, data visualization, and table making in R and then copy everything to PowerPoint by hand? No! R has robust presentation-making capabilities.

In this chapter, we’ll learn how to make presentations in R using the xaringan package. This package, which uses R Markdown to make presentations, is one of several you can use to make slides, but it is the most widely used. To learn more about the capabilities of xaringan, I spoke with Silvia Canelón, a data analyst in the Urban Health Lab at the University of Pennsylvania. Canelón has done extensive teaching on the xaringan package and has thought deeply about its benefits. As you’ll hear in this chapter, these benefits include but also go well beyond making good-looking slides.

How xaringan Works

To get started with xaringan, you need to first install the package using the standard approach: install.packages("xaringan"). Once it is installed, you can now go to File > New File > R Markdown. You’re probably thinking that you want to go to the Presentation tab. Doing so will give you several options for making slides, as you can see in Figure 8-1.

[F08001.png]



* + - * 1. The options presented to create a new presentation

These are other formats you can use to make slides (and yes, you can knit an R Markdown document to PowerPoint). However, if you want to make a presentation with xaringan (and, as I will explain below, I think it’s the best option), you should go to the From Template tab. From there, scroll down until you find the template called Ninja Presentation. Select that, hit OK, and you’ll get a blank R Markdown document. Just as when we created a regular R Markdown document, creating a new xaringan presentation will give us some default content. I’m going to delete that and add my own contents which you can see below.

---

title: "Penguins Report"

author: "David Keyes"

date: "2024-01-12"

output: xaringan::moon\_reader

---

```{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-2008.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)) %>%

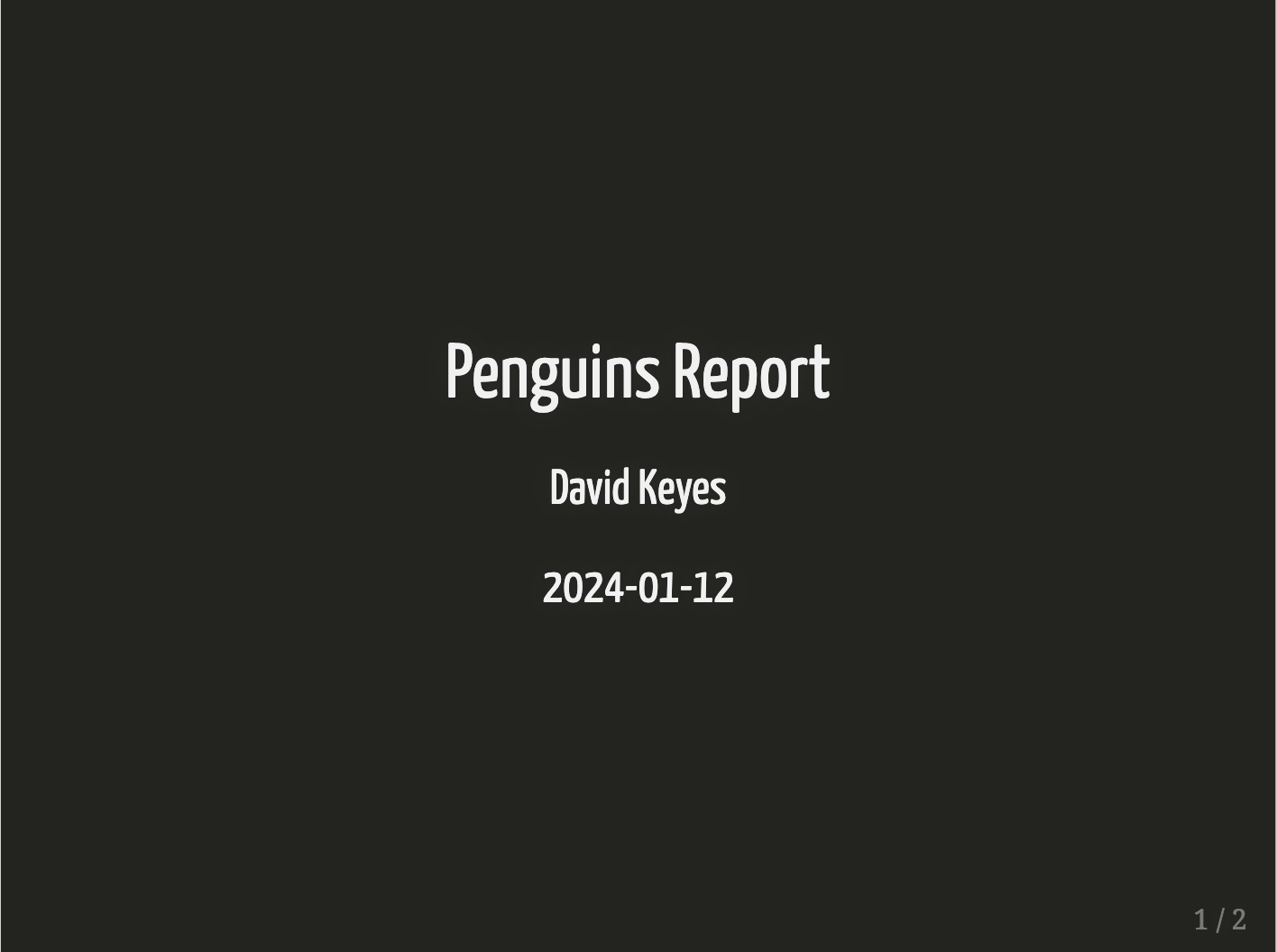
pull(avg\_bill\_length)

```

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

This code should look familiar because it is exactly the same as the R Markdown document we created in Chapter 6 with one line changed in the YAML. Instead of output: word\_document we now have output: xaringan::moon\_reader. With this change, we will now get slides rather than a Word document. Let’s try hitting the Knit button to see what it looks like. When I hit Knit, I get an HTML file with the same name as R Markdown document (in my case, xaringan-example.Rmd and xaringan-example.html). This is the first thing to know about xaringan: the slides it produces are HTML files. It may seem odd to get slides as HTML files, but it offers several advantages over formats like PowerPoint, as we’ll see below. If I open up the HTML document, I see Figure 8-2.

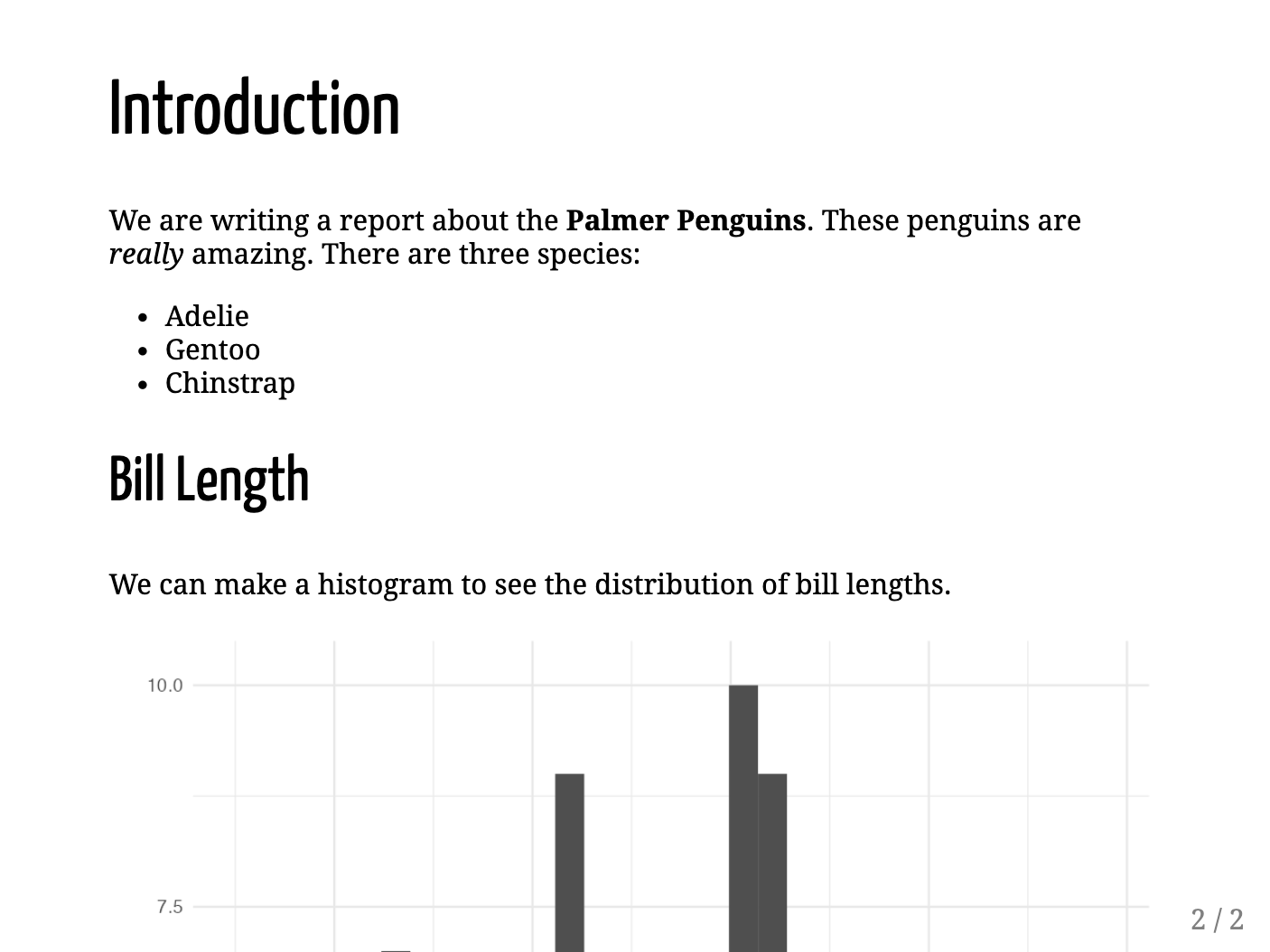
[F08002.png]



* + - * 1. The first slide of my presentation

If we scroll to the next slide with the right arrow key, we’ll see content that looks familiar. Figure 8-3 shows this slide, which has the same text and a cut-off version of the histogram we made in Chapter 6.

[F08003.png]



* + - * 1. The second slide of my presentation

At this point, we’ve learned two things about how xaringan works:

1. **The overall syntax for making slides with** xaringan **is nearly identical to that used to make reports with R Markdown.** We copied in the content of our R Markdown document, changed one line, and we see the same output, just in slide format. The YAML, markdown text, and code chunks work exactly the same.
2. **We need to make a few tweaks to make our content fit on our slides.** When we’re working in an R Markdown document that will be knitted to Word, its length doesn’t matter because reports can be one page or 100 pages. Working with xaringan, however, we need to be conscientious about how much content will fit on one slide. Our cut-off histogram shows us what happens without doing this. Let’s fix it.

Breaking Content Across Slides

Let’s make our histogram fully visible by putting it on its own slide. To make a new slide, we add three dashes at any point in our R Markdown document. I’ve copied the relevant portion of the document and added these dashes.

---

## Bill Length

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

```{r}

penguins %>%

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

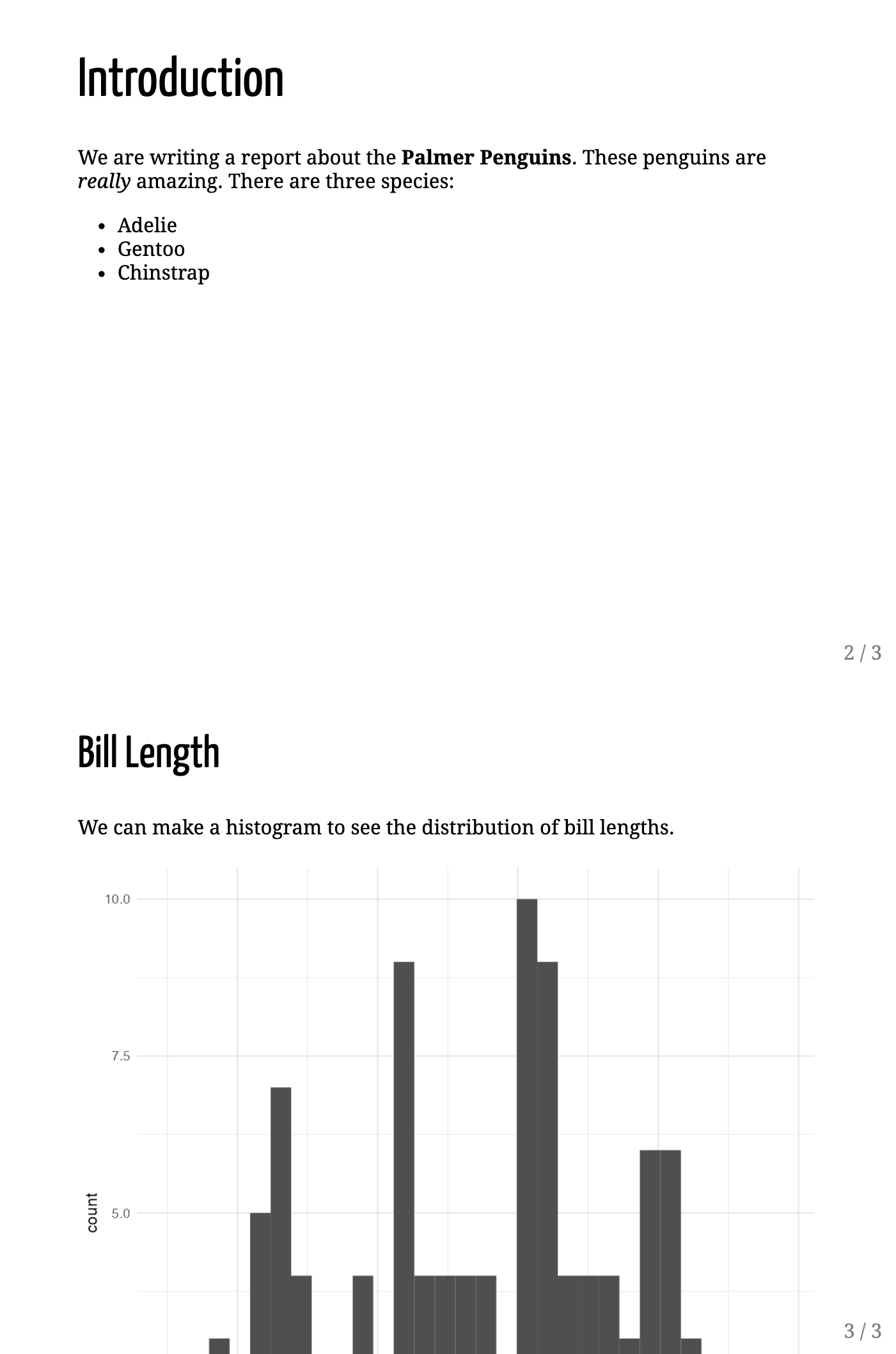
geom\_histogram() +

theme\_minimal()

```

If I knit again, what was one slide is now broken into two: an Introduction slide and a Bill Length slide. We can see both in Figure 8-4.

[F08004.png]



* + - * 1. The presentation broken into two slides

If I look closely, I can see that the bottom of histogram is ever so slightly cut off. I can fix this by adjusting its size. We do this using the code chunk option fig.height as follows:

---

## Bill Length

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

```{r fig.height = 4}

penguins %>%

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

geom\_histogram() +

theme\_minimal()

```

Doing this makes the histogram fit on the slide and also reveals the text that was hidden below.

Incrementally Revealing Content

When presenting, it’s often useful to only show a portion of the content on each slide at a time. Let’s say, for example, we know that when we’re presenting the first slide, we want to talk a bit about each penguin species. Rather than have all three species visible when we open this slide, it would be nice to have the names come up one at a time. We can do this using what xaringan calls incremental reveal.

To use this feature, we put two dashes between any content we want to incrementally reveal. This code, for example, will let us show Adelie on the screen, then Adelie and Gentoo, then Adelie, Gentoo, and Chinstrap.

# Introduction

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

- Adelie

--

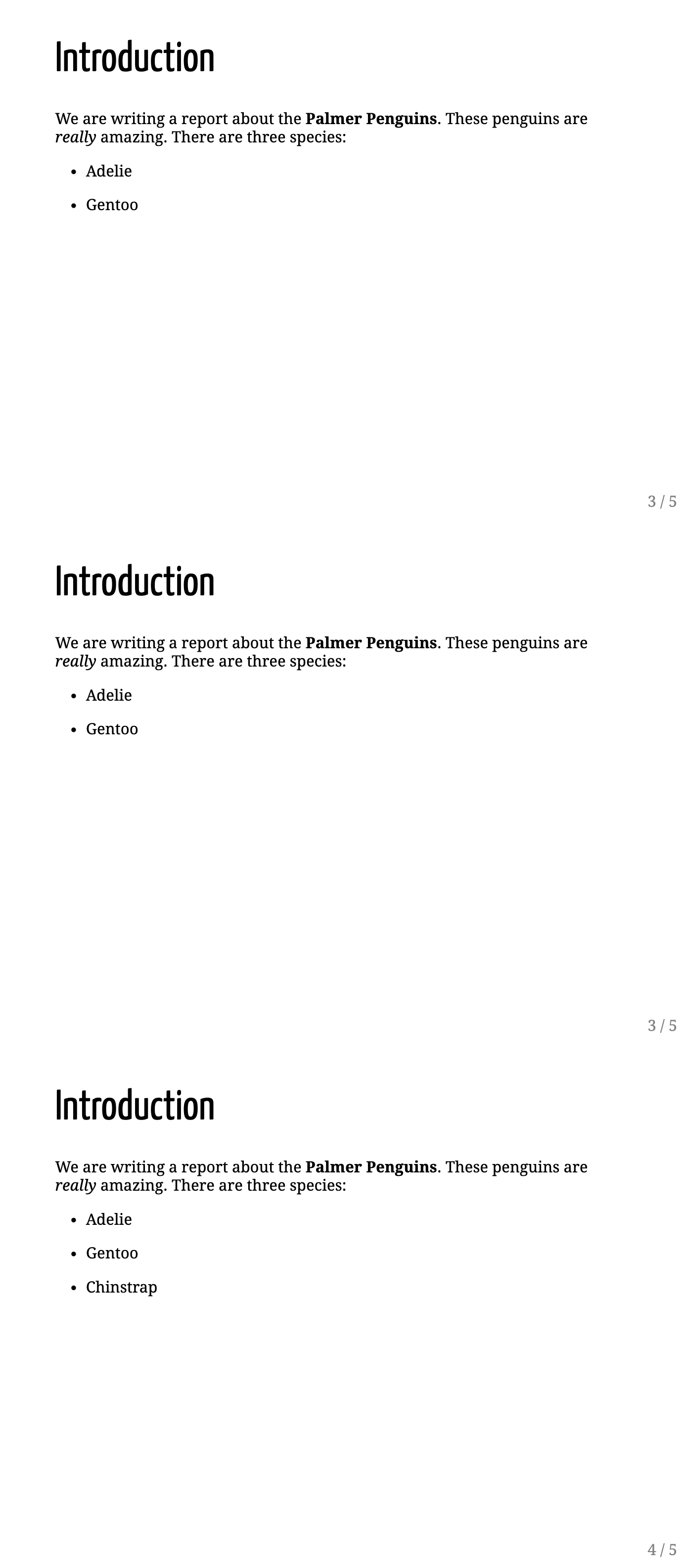
- Gentoo

--

- Chinstrap

When presenting your slides, you’ll use the right arrow to incrementally reveal the species. We can see what this looks like in Figure 8-5.

[F08005.png]



* + - * 1. One slide shown with incremental reveal

Aligning Content

When designing your presentation, you’ll also likely want to control the alignment of content. We do this by adding what are known as content classes. You can surround any content with the classes .left[], right[], and center[] to align them. For example, on our slide that creates a histogram and associated text, we could put .center[] around the code chunk that makes the histogram:

## Bill Length

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

.center[

```{r fig.height = 4}

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)) %>%

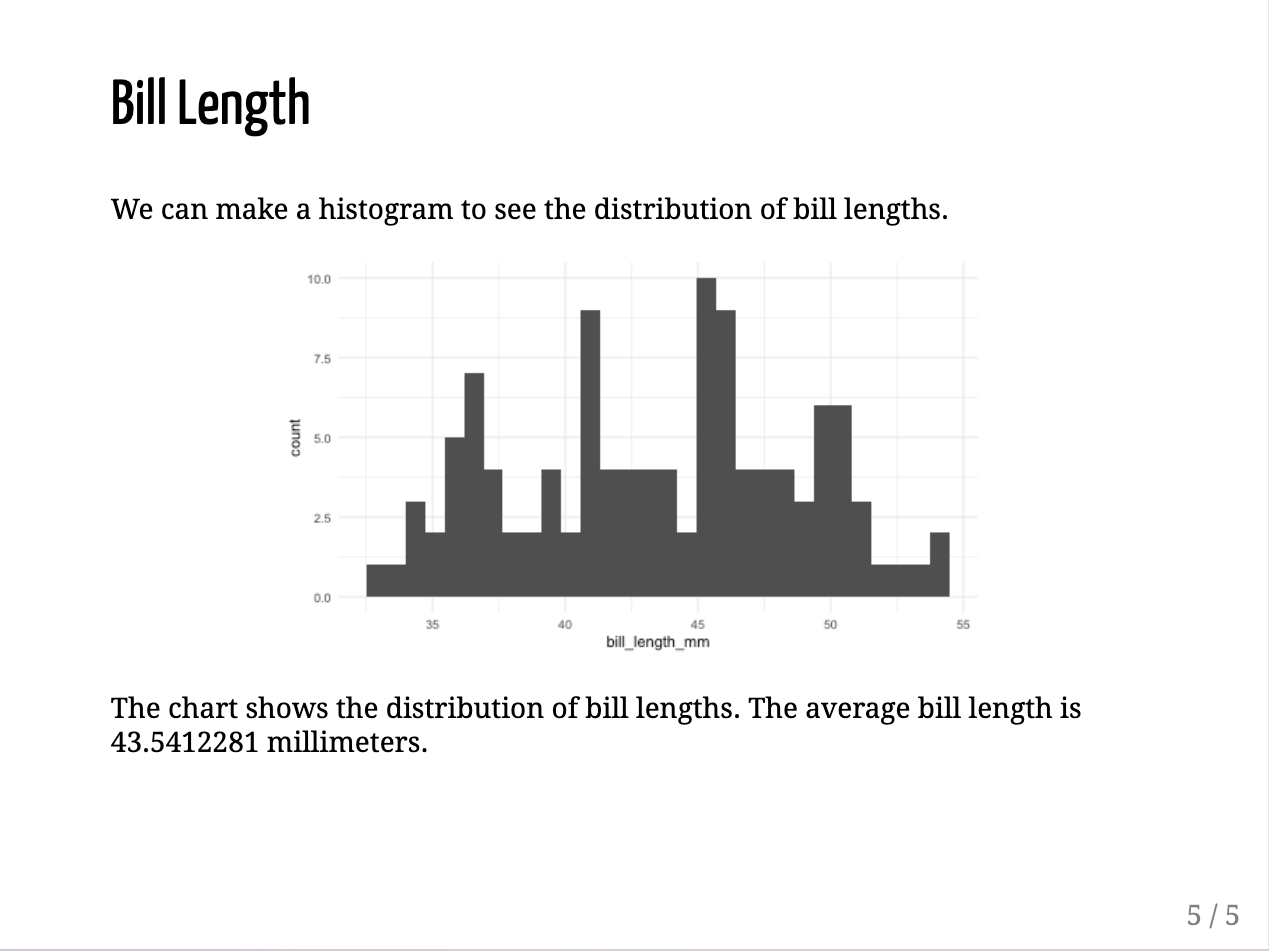
pull(avg\_bill\_length)

```

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

Doing this would center the chart on our slide, as we can see in Figure 8-6.

[F08006.png]



* + - * 1. A slide with the chart centered

There are also built-in options to make two-column layouts. Adding pull-left[] and pull-right[] in this way will make two equally spaced columns.

## Bill Length

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

.pull-left[

```{r fig.height = 4}

penguins %>%

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

geom\_histogram() +

theme\_minimal()

```

]

.pull-right[

```{r}

average\_bill\_length <- penguins %>%

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

na.rm = TRUE)) %>%

pull(avg\_bill\_length)

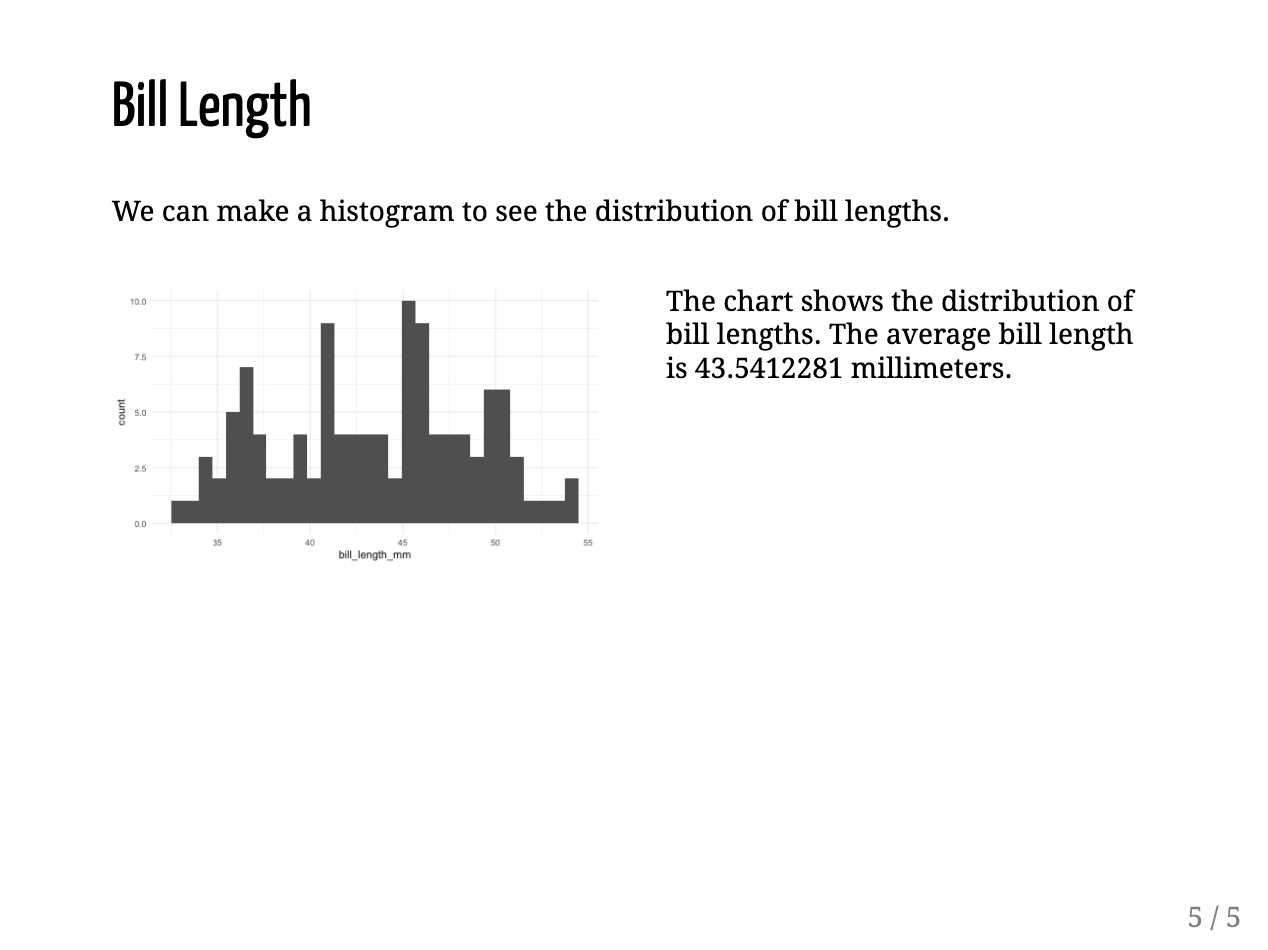
```

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

]

We can see what this looks like in Figure 8-7 below.

[F08007.png]



* + - * 1. A slide with two columns

You can also use the content classes .left-column[] and .right-column[] to make a layout with a narrow left column and wide right column. The code below puts the text on the left and the histogram on the right.

## Bill Length

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

.right-column[

```{r fig.height = 4}

penguins %>%

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

geom\_histogram() +

theme\_minimal()

```

]

.left-column[

```{r}

average\_bill\_length <- penguins %>%

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

na.rm = TRUE)) %>%

pull(avg\_bill\_length)

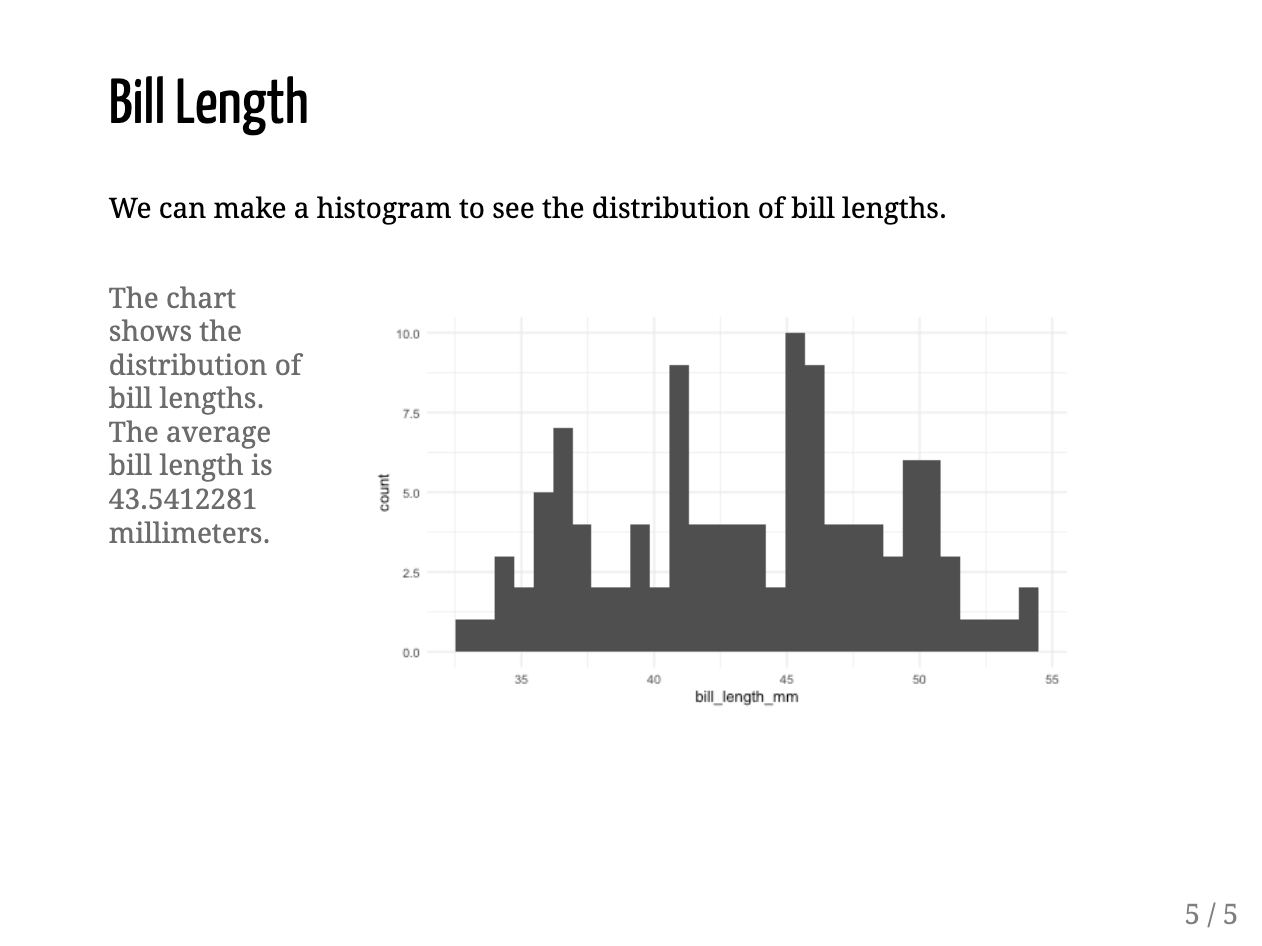
```

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

]

We can see the slide output in Figure 8-8.

[F08008.png]



* + - * 1. A slide with a smaller left column and a larger right column

In addition to aligning particular pieces of content on slides, we can also everything using the .left, right, and center classes. To do this, you specify the class right after the three dashes that indicate a new slide and before any content.

---

class: center

## Bill Length

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

```{r fig.height = 4}

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)) %>%

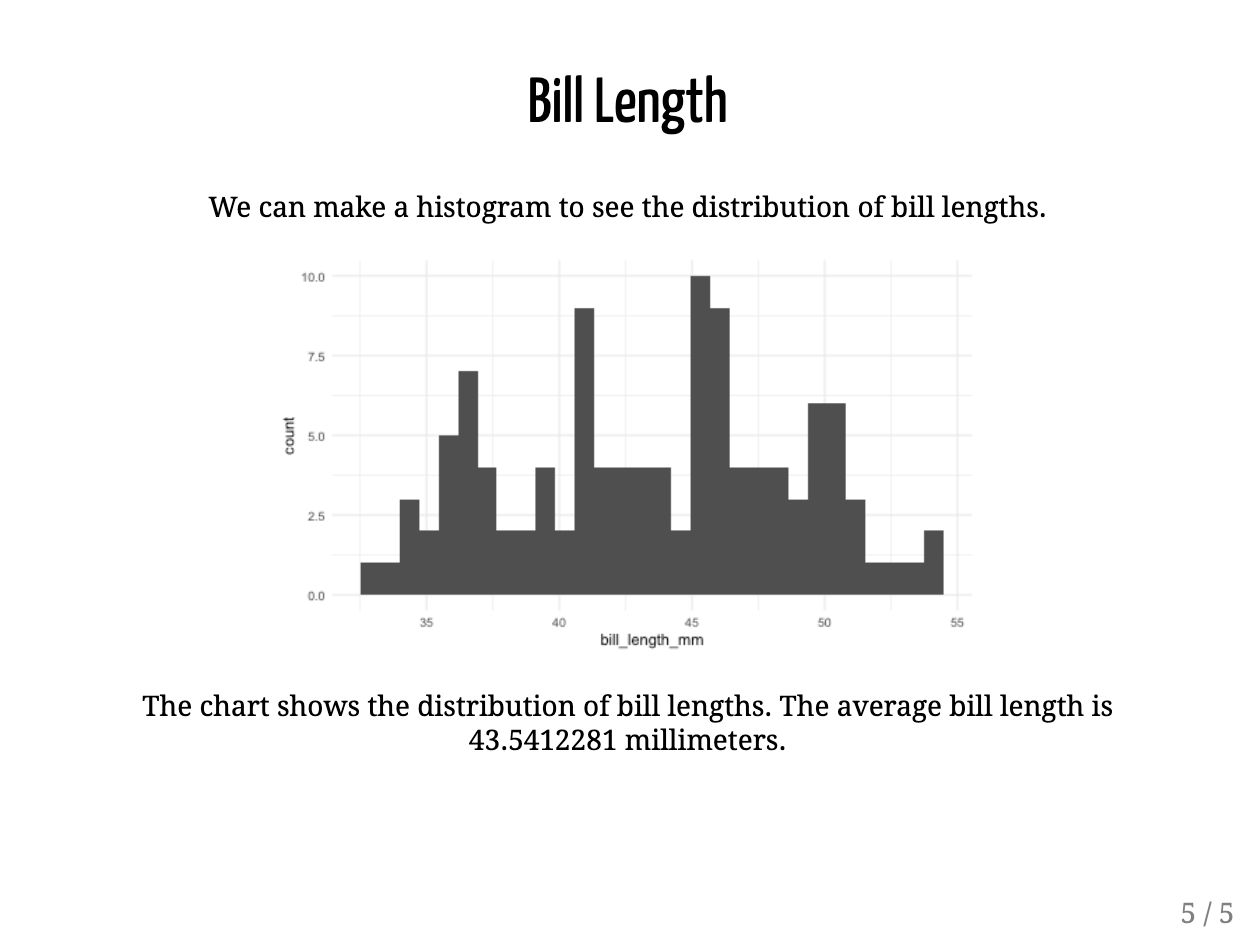
pull(avg\_bill\_length)

```

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

Doing this would give us a fully-centered slide, as seen in Figure 8-9 below.

[F08009.png]



* + - * 1. A fully centered slide

You can also use the class .middle to vertically center the slide, as we’ll see in the next section.

Adding Background Images to Slides

The same syntax that we just used to center our entire slide can also enable us to add a background image. Below I’ve created a new slide, added the classes center and middle to horizontally and vertically align the content, and added a background image, surrounding the path to the image with url().

class: center, middle

background-image: url("penguins.jpg")

## Penguins

Doing this gives me a slide with a picture of penguins in the background with the text Penguins in front of it. Figure 8-10 shows the output.

[F08010.png]



* + - * 1. Figure 8-10: A slide that uses a background image

Customizing our Slides Further

One issue with the slide we just made is that it’s hard to read the word “Penguins.” It would probably be best if we could make the text bigger and a different color. To do this, we need to use some CSS, the language used to style HTML documents (remember, when we knit our xaringan presentation, we end up with an HTML document). If you’re thinking, “I’m reading this book to learn R, not CSS,” don’t worry. You just have to know a bit of CSS to make tweaks to your slides.

To add custom CSS, I’m going to create a code chunk. To tell R Markdown that this code chunk contains CSS, not R, I put the text “css” in between the curly brackets. I can then add CSS like this to tell R Markdown to make the h2 (the second-level header) 150px and white. I have to add the .remark-slide-content before the h2 in order to make sure we target the specific element in our presentation (remark comes from remark.js, a JavaScript library to make presentations that xaringan uses under the hood).

We can see our slide in Figure 8-11.

[F08011.png]



* + - * 1. Figure 8-11: The title slide with changes to the font to make the text more visible

If I wanted to change the font, I could do so with some additional CSS. The first line of this code will make a font called Inter available to use in our slides (we do this because not everyone has this font installed on their computers). The two lines I added will make the h2 use the Inter font and make it bold.

The slide with bold Inter font is visible in Figure 8-12.

[F08012.png]



* + - * 1. Figure 8-12: The title slide with changes to the font to make the text more visible

Because xaringan slides are built as HTML documents, the sky is the limit in terms of customizing them with CSS. Of course, you have to know the ins and outs of CSS, and if you’re reading this book, this probably isn’t a skill you possess. Fortunately, there are two ways you can customize your slides without knowing CSS.

R users have made xaringan themes that you can use as well. If you run this code, you will get a list of all of these themes.

names(xaringan:::list\_css())

Doing this will show the following output:

#> [1] "chocolate-fonts" "chocolate"

#> [3] "default-fonts" "default"

#> [5] "duke-blue" "fc-fonts"

#> [7] "fc" "glasgow\_template"

#> [9] "hygge-duke" "hygge"

#> [11] "ki-fonts" "ki"

#> [13] "kunoichi" "lucy-fonts"

#> [15] "lucy" "metropolis-fonts"

#> [17] "metropolis" "middlebury-fonts"

#> [19] "middlebury" "nhsr-fonts"

#> [21] "nhsr" "ninjutsu"

#> [23] "rladies-fonts" "rladies"

#> [25] "robot-fonts" "robot"

#> [27] "rutgers-fonts" "rutgers"

#> [29] "shinobi" "tamu-fonts"

#> [31] "tamu" "uio-fonts"

#> [33] "uio" "uo-fonts"

#> [35] "uo" "uol-fonts"

#> [37] "uol" "useR-fonts"

#> [39] "useR" "uwm-fonts"

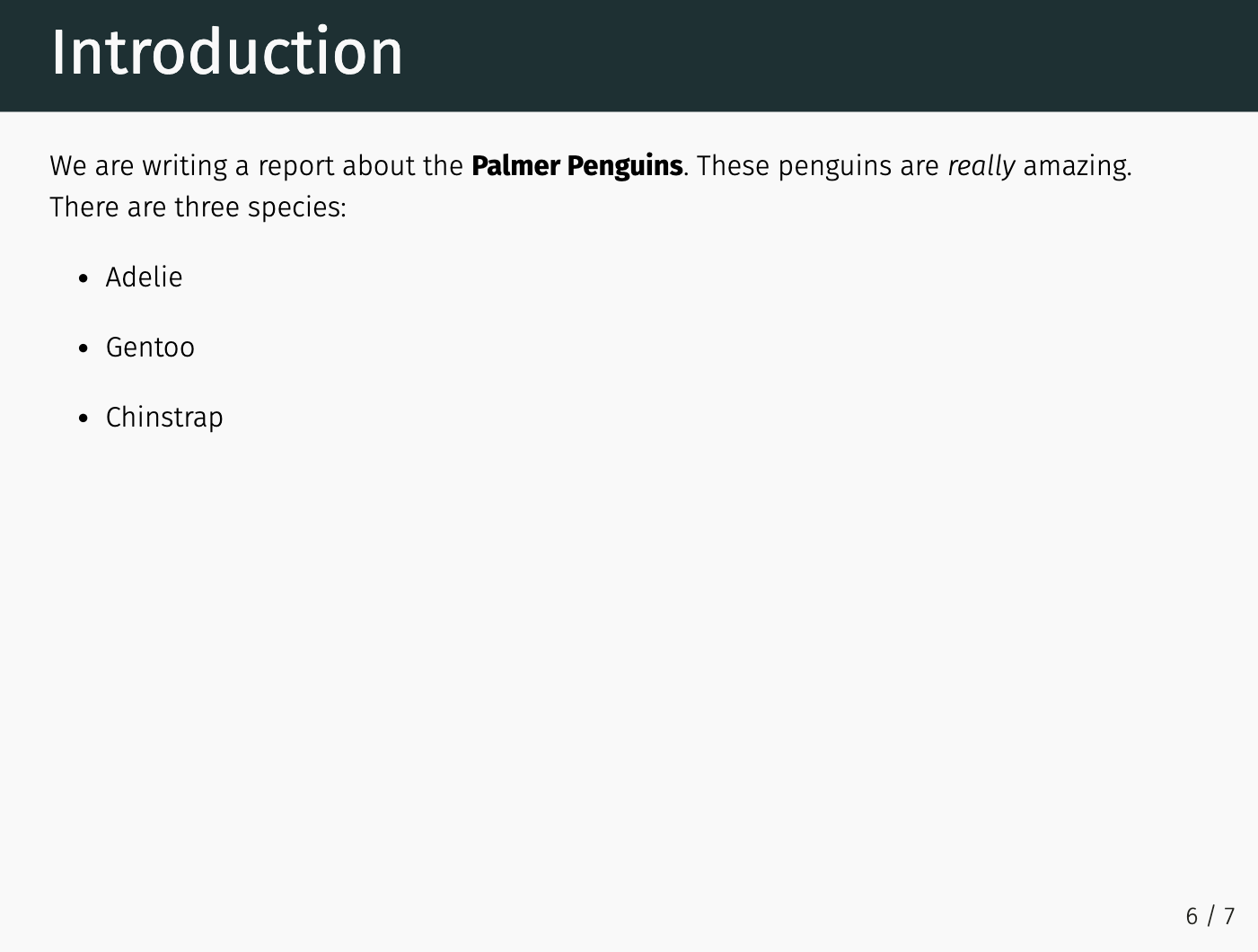
#> [41] "uwm" "wic-fonts"

#> [43] "wic"

You can then choose a theme you’d like to use. To use the theme, you adjust your YAML as follows. This code tells xaringan to use the default CSS as well as customizations made in the metropolis and metropolis-fonts CSS files (these come bundled with xaringan so you don’t need to do anything beyond installing the package to access them).

You can see how this theme changes the look-and-feel of our slides in Figure 8-13 below.

[F08013.png]



* + - * 1. Figure 8-13: A slide using the metropolis theme

If writing custom CSS is the totally flexible but more challenging option to tweaking your xaringan slides, using a custom theme is way simpler but a lot less flexible. A nice middle ground is to use the xaringanthemer package by Garrick Aden-Buie. This package has several built-in themes and also allows you to easily create your own custom xaringan theme. After installing the xaringanthemer package, you adjust the css line in your YAML to use the xaringan-themer.css file.

With this in place, you can now customize your slides by using the style\_xaringan() function. This function has over 60 arguments, allowing you to tweak nearly any part of your xaringan slides. To make the same changes that we made above with custom CSS, I’ll use just a few of the arguments. One particularly nice thing about the xaringanthemer package is that you can use any font available on Google Fonts by simply adding its name to header\_font\_family or any other similar argument (no need to run the line above that made the Inter font available to us). Here’s how I would make the same changes that I made with custom CSS above.

I could show you what my slide with the penguin image background looks like, but trust me, it’s exactly identical to the slide we made previously with custom CSS.

In Conclusion: The Advantages of xaringan

Now that we’ve discussed how to use xaringan, let’s talk a bit about why you might consider switching to it for your presentations. In my conversation with Silvia Canelón, she brought up three main reasons.

First, if you’re already working in R and R Markdown, being able to produce slides in the same tool is a game changer. Say you’ve written a report that you’ve knitted to a Word document. You can now reuse your code from that R Markdown document and use it to make figures, tables, and more in your xaringan slides.

Second, because xaringan creates slides as HTML documents, you can post them online (we’ll discuss ways to do this in Chapter ??). No need to email or print out your slides for your viewers. Just share the link to your slides and you’re done.

The third benefit of using xaringan is accessibility. As Canelón put it to me, “when [people] have the HTML version of the slides, they have some control over what it looks like.” People with limited vision are able to access HTML documents in ways that are accessible to them. They can also increase the text size or use screen readers. Making presentations with xaringan isn’t just a cool trick. It also means more people can engage with your slides.