Quarto, the Next-Generation Tool for Reproducible Reporting

In mid-2022, the team at Posit (the company that makes the RStudio editor, the tidyverse, and many other tools to support R programmers) made an announcement: they had developed a new tool called Quarto, which they described as the next-generation version of R Markdown. The Posit team explained a few of the reasons why they felt the need to develop a new tool for reproducible reporting:

1. More consistent syntax: As you’ve seen in this book, the syntax used to create different outputs with R Markdown varies. For example, the distill package has layout options that don’t work in xaringan. And xaringan uses three dashes to indicate new slides, while three dashes in other output formats would create a horizontal line. With Quarto, the idea is to make the syntax used to create different outputs more consistent.
2. Support for multiple languages: While R Markdown does work with languages such as Python, having the word R in the name was off-putting for non-R users. Creating a tool like Quarto was Posit’s “attempt to bring R Markdown to everyone.” Quarto makes it straightforward for users to include code from multiple languages in a single document.
3. Support for multiple code editors: R Markdown is designed to work in the RStudio IDE. With the move to support additional languages, it was important to support the code editors that users of those languages prefer. Quarto works in RStudio as well as editors such as VS Code and JupyterLab.

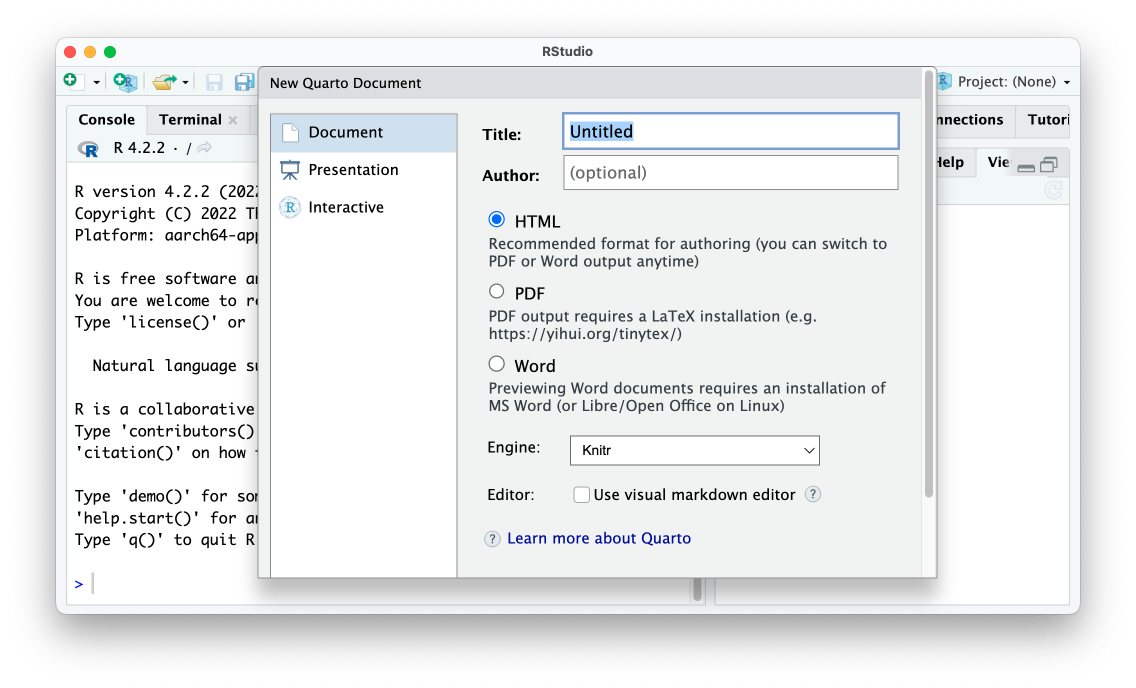
While the benefits of Quarto are clearest for non-R users, there are also many reasons why R users may want to use Quarto. I’ll focus in this chapter on the benefits for R users who use RStudio. I’ll begin by explaining how to set up and use Quarto before going over some of the most important differences between Quarto and R Markdown. I’ll then demonstrate how to make each of the products we made in previous chapters with R Markdown using Quarto. Parameterized reporting, presentations, and websites are all things that Quarto can do, and this chapter will show you how.

Getting Started with Quarto

To get started with Quarto, we first need to make sure it is installed. The good news is that newer versions of RStudio (2022.07.1 and later) come with Quarto installed. To check your RStudio version, click RStudio in the top menu bar, then click About RStudio. If you have an older version of RStudio, update it now (repeat the steps outlined in Chapter 2 on downloading RStudio) and Quarto will be installed for you.

Once you have Quarto installed, you’re ready to work with it. You can create a Quarto document by clicking File > New File > Quarto Document. You’ll see a menu, shown in Figure 10-1, that looks like what you saw when creating an R Markdown document.

[F10001.png]



* + - * 1. The RStudio menu for creating a new Quarto document

I’ll give my document a title and choose HTML as my output format. The engine option allows users to select a different way to render documents. I’ll keep the default of Knitr, which is the same tool that R Markdown uses to render documents. I’ll keep the Use visual markdown editor unchecked (this option allows you to use an interface that looks more like Microsoft Word, but I’ve found it finicky). The Quarto document that is created has default content, just like R Markdown documents.

---

title: "My Report"

format: html

---

## Quarto

Quarto enables you to weave together content and executable code into a finished document. To learn more about Quarto see <https://quarto.org>.

## Running Code

When you click the \*\*Render\*\* button a document will be generated that includes both content and the output of embedded code. You can embed code like this:

```{r}

1 + 1

```

You can add options to executable code like this

```{r}

#| echo: false

2 \* 2

```

The `echo: false` option disables the printing of code (only output is displayed).

There are some differences between R Markdown and Quarto and many things in common. Let’s explore them.

Comparing R Markdown and Quarto

Looking through our newly created Quarto document, you can see how similar Quarto and R Markdown are overall. Both have the same basic structure: YAML metadata, followed by a combination of markdown text and code chunks. Despite the similarities, there are some differences. In R Markdown, we would have created an HTML document with this YAML.

---

title: "My Report"

output: html\_document

---

With Quarto, output is replaced with format and we write html instead of html\_document.

---

title: "My Report"

format: html

---

A second difference between R Markdown and Quarto is that the setup code chunk seen in the former does not exist in the latter. Recall from Chapter 7 that the setup code chunk sets default options on things like whether we show code, charts and tables, and other elements in the rendered versions of our documents. With Quarto, these options are set in the YAML. For example, if we want to hide code as well as all warnings and messages from our rendered document, we add them under execute in our YAML as follows (Quarto also allows you to write true and false in lower case):

---

title: "My Report"

format: html

execute:

echo: false

warning: false

message: false

---

In R Markdown, if we want to override options at the individual code chunk level, we do it by adding them within the curly brackets that start a code chunk. The code chunk below would show both the code 2 \* 2 as well as the output (4).

```{r echo = TRUE}

2 \* 2

```

With Quarto, we use a slightly different syntax to set individual chunk-level options. You can see below how the chunk option is not set within the curly brackets, but instead in the code chunk itself. We use the text #| (known as the hash pipe) at the start of a line to indicate that we are setting options.

```{r}

#| echo: false

2 \* 2

```

A final difference you’re likely to see if you switch from R Markdown to Quarto is that option names that consist of two words are separated by a dash rather than a period. R Markdown, for example, uses the code chunk option fig.height to determine the height of plots.

```{r fig.height = 10}

library(palmerpenguins)

library(tidyverse)

ggplot(penguins, aes(x = bill\_length\_mm,

y = bill\_depth\_mm)) +

geom\_point()

```

In contrast, Quarto uses fig-height as follows.

```{r}

#| fig-height: 10

library(palmerpenguins)

library(tidyverse)

ggplot(penguins, aes(x = bill\_length\_mm,

y = bill\_depth\_mm)) +

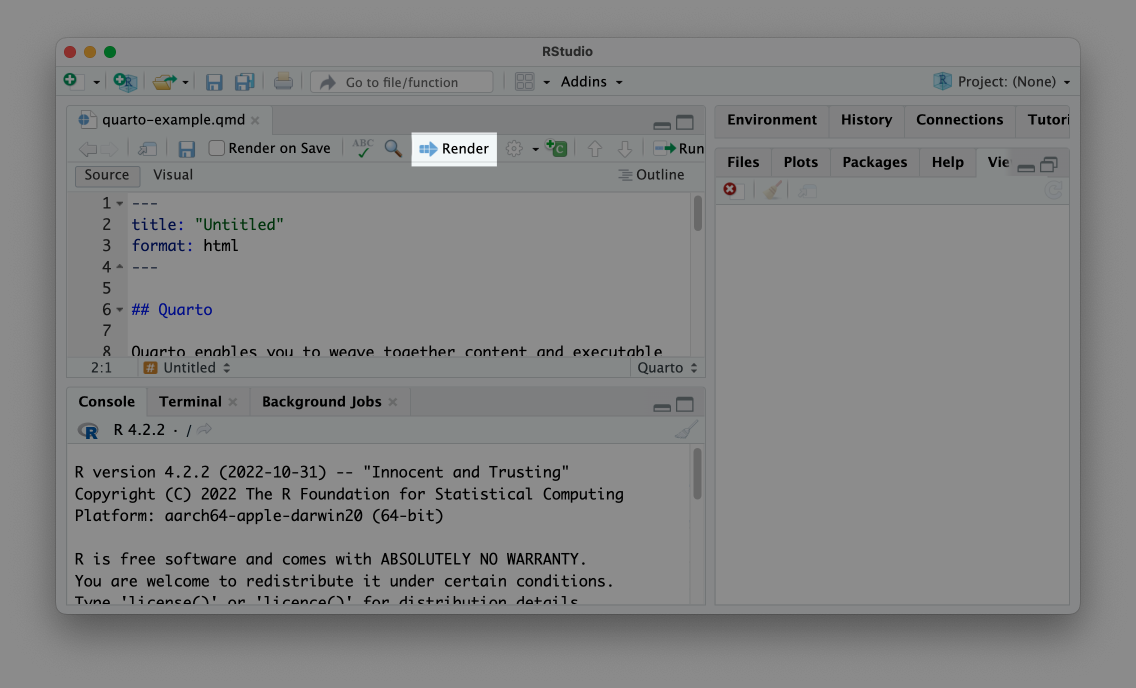
geom\_point()

```

Helpfully for those of us coming from R Markdown, fig.height and similar options with periods in them will continue to work if you forget to make the switch. A list of all code chunk options can be found on the Quarto website: <https://quarto.org/docs/reference/cells/cells-knitr.html>.

Rendering your Quarto document to HTML (or any other format) follows the same process as R Markdown. While you use the Knit button to render an R Markdown document, with Quarto the button is called Render. Figure 10-2 shows the Render button.

[F10002.png]



* + - * 1. The Render button

Clicking the Render button will turn the Quarto document into an HTML file, Word document, or any other output format you select.

As you’ve seen, the differences between R Markdown and Quarto are mostly around syntax. All of the general concepts you learned about generating reports with the single-tool workflow described in Chapter 7 apply to Quarto as well.

Making Different Types of Outputs with Quarto

Now that you’ve learned how Quarto works to create a simple document, let’s make a few different things with it. We’ll go through the outputs that you’ve learned to make in Chapters ??, 9, and 10.

Parameterized Reporting with Quarto

Let’s start with parameterized reporting. The process of making parameterized reports with Quarto is nearly identical to how it’s done with R Markdown. Below, I’ve taken the R Markdown document used to make the Urban Institute COVID Report in Chapter ?? and adapted it for Quarto. I did so by copying the .Rmd file and changing its extension to .qmd. From there, I made the following changes:

1. Switched output: html\_document to format: html in the YAML. 1
2. Removed the setup code chunk and put the options that were there in the YAML. 2
3. Switched the fig.height option in the last code chunk to fig-height and used the hash pipe. 3

---

title: "Urban Institute COVID Report"

format: html 1

params:

state: "Alabama"

execute: 2

echo: false

warning: false

message: false

---

```{r}

library(tidyverse)

library(urbnthemes)

library(here)

library(scales)

```

# `r params$state`

```{r}

cases <- tibble(state.name) %>%

rbind(state.name = "District of Columbia") %>%

left\_join(read\_csv("https://data.rwithoutstatistics.com/united\_states\_covid19\_cases\_deaths\_and\_testing\_by\_state.csv", skip = 2),

by = c("state.name" = "State/Territory")) %>%

select(total\_cases = `Total Cases`, state.name,

cases\_per\_100000 = `Case Rate per 100000`) %>%

mutate(cases\_per\_100000 = parse\_number(cases\_per\_100000)) %>%

mutate(case\_rank = rank(-cases\_per\_100000, ties.method = "min"))

```

```{r}

state\_text <- if\_else(params$state == "District of Columbia", str\_glue("the District of Columbia"), str\_glue("state of {params$state}"))

state\_cases\_per\_100000 <- cases %>%

filter(state.name == params$state) %>%

pull(cases\_per\_100000) %>%

comma()

state\_cases\_rank <- cases %>%

filter(state.name == params$state) %>%

pull(case\_rank)

```

In `r state\_text`, there were `r state\_cases\_per\_100000` cases per 100,000 people in the last seven days. This puts `r params$state` at number `r state\_cases\_rank` of 50 states and the District of Columbia.

```{r}

#| fig-height: 8 3

set\_urbn\_defaults(style = "print")

cases %>%

mutate(highlight\_state = if\_else(state.name == params$state, "Y", "N")) %>%

mutate(state.name = fct\_reorder(state.name, cases\_per\_100000)) %>%

ggplot(aes(x = cases\_per\_100000,

y = state.name,

fill = highlight\_state)) +

geom\_col() +

scale\_x\_continuous(labels = comma\_format()) +

theme(legend.position = "none") +

labs(y = NULL,

x = "Cases per 100,000")

```

Despite these small differences, the rendered report looks almost exactly the same as the one we produced in Chapter 7. Also nearly identical is the process of creating one report for each state. We can take the render.R script file we used to make parameterized reports in Chapter ?? and, with a few small tweaks, it works. In the updated render.R file below, I made the following tweaks:

1. Loaded the quarto package instead of the rmarkdown package. 1
2. Made the input file urban-covid-budget-report.qmd instead of urban-covid-budget-report.Rmd. 2
3. In the reports tibble I created, used execute\_params instead of params. 3
4. On the last line, used the quarto\_render() function instead of the render() function from the markdown package. 4

# Load packages

library(tidyverse)

library(quarto) 1

# Create a vector of all states and the District of Columbia

state <- tibble(state.name) %>%

rbind("District of Columbia") %>%

pull(state.name)

# Create a tibble with information on the:

# input R Markdown document

# output HTML file

# parameters needed to knit the document

reports <- tibble(

input = "urban-covid-budget-report.qmd", 2

output\_file = str\_glue("{state}.html"),

execute\_params = map(state, ~list(state = .)) 3

)

# Generate all of our reports

reports %>%

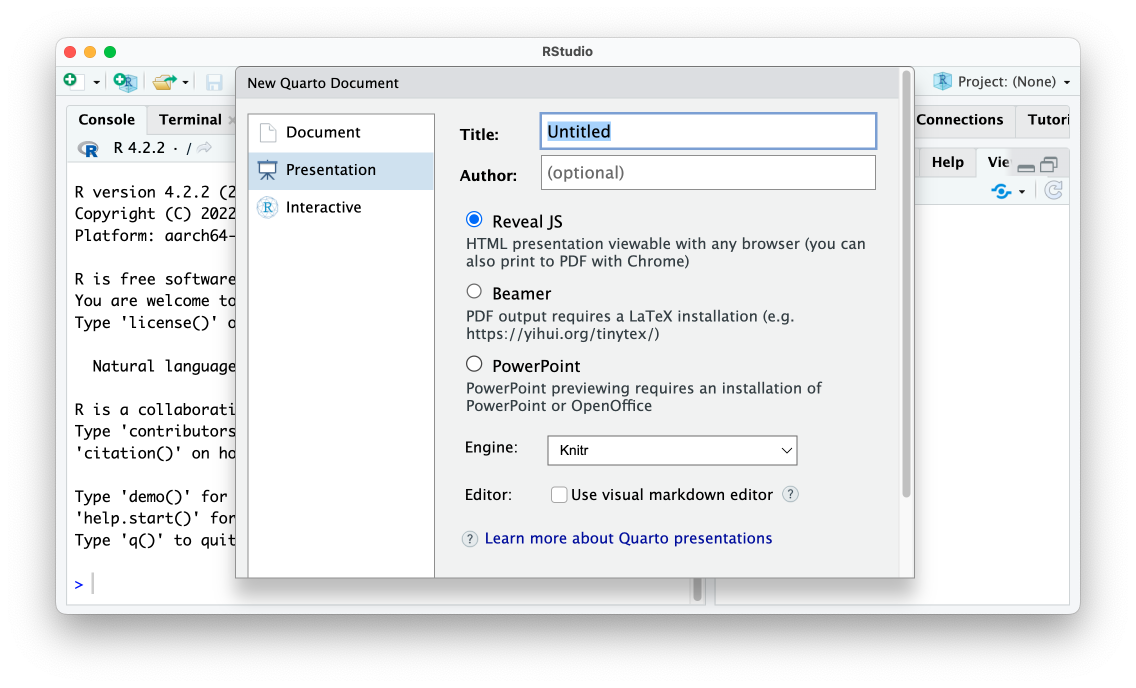
pwalk(quarto\_render) 4

Just as in Chapter 7, running this code will produce one report for each state.

Making Presentations with Quarto

Quarto can also produce presentations like those we made in Chapter 9. Again, the process is nearly identical to how we made slides with the xaringan package. To make a presentation with Quarto, click on File > New File > Quarto Presentation. You’ll want to choose Reveal JS to make your slides and leave the Engine and Editor options untouched, as seen in Figure 10-3.

[F10003.png]



* + - * 1. The RStudio menu to make a new Quarto presentation

The slides we’ll make use the reveal.js javascript library under the hood. Making slides in this format is the Quarto approach that is most similar to making slides with xaringan. Below I’ve taken the code used to make a presentation with xaringan in Chapter 9 and made a few updates to make it work with Quarto.

---

title: "Penguins Report"

author: "David Keyes"

format: revealjs

execute:

echo: false

warning: false

message: false

---

# Introduction

```{r}

library(tidyverse)

```

```{r}

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

```

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)) %>%

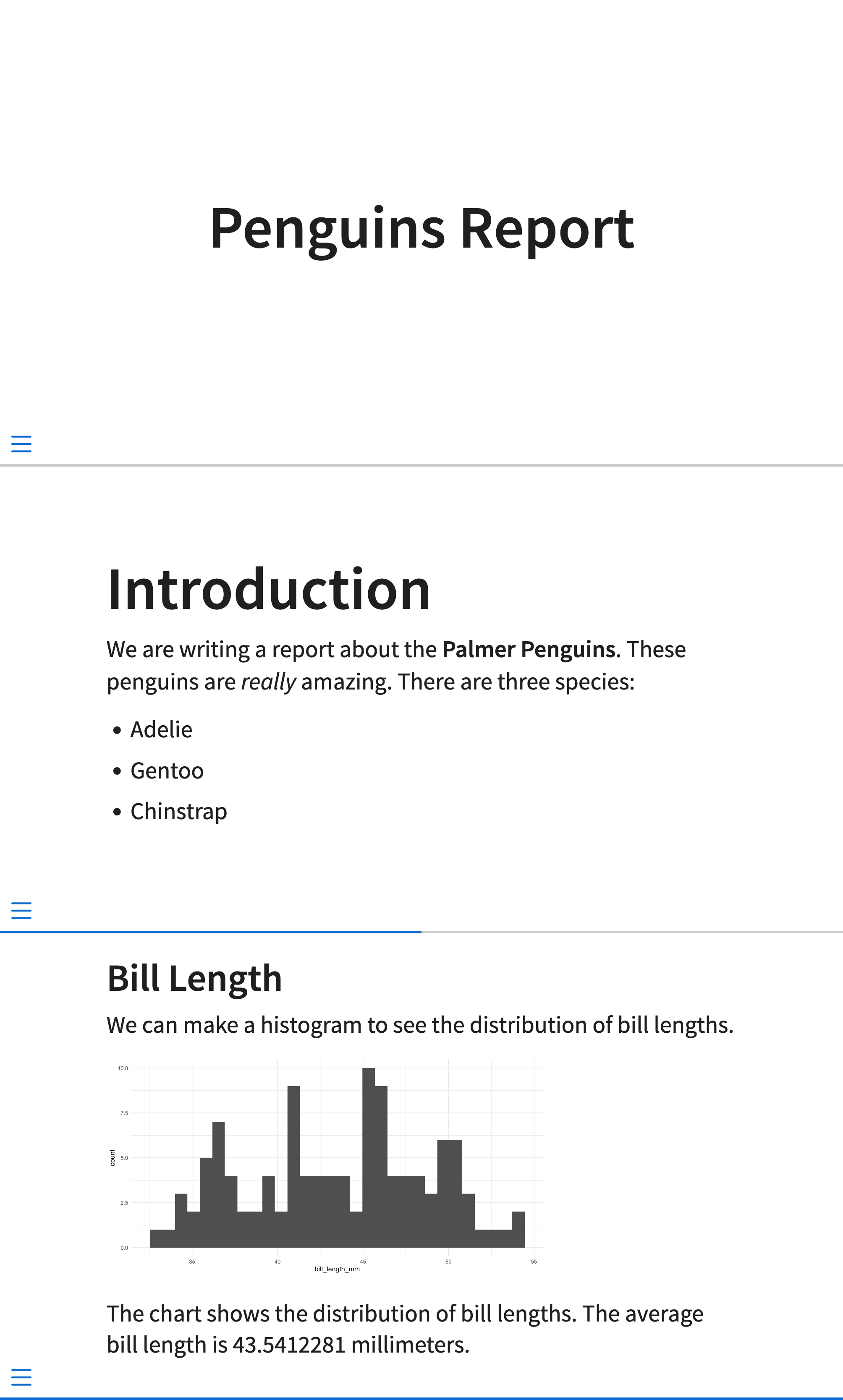
pull(avg\_bill\_length)

```

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

When we render this, we get an HTML file with our slides, seen in Figure 10-4 below.

[F10004.png]



* + - * 1. Slides made with Quarto

Overall, the output looks similar to the default xaringan slides we made. However, remember how we had to manually add three dashes to make slides with xaringan? With Quarto, any first- or second-level headers make new slides so I was able to remove these (though you can use three dashes to manually add slide breaks).

Incrementally Revealing Content

Quarto slides also make it possible to incrementally reveal content. Bulleted and numbered lists can be made to incrementally reveal by default by adjusting the YAML. See here how I add incremental: true to the YAML.

---

title: "Penguins Report"

author: "David Keyes"

format:

revealjs:

incremental: true

execute:

echo: false

warning: false

message: false

---

When I do this, the content in all lists in my presentation will be revealed one item at a time. You can also set just some lists to incrementally reveal using this format:

::: {.incremental}

- Adelie

- Gentoo

- Chinstrap

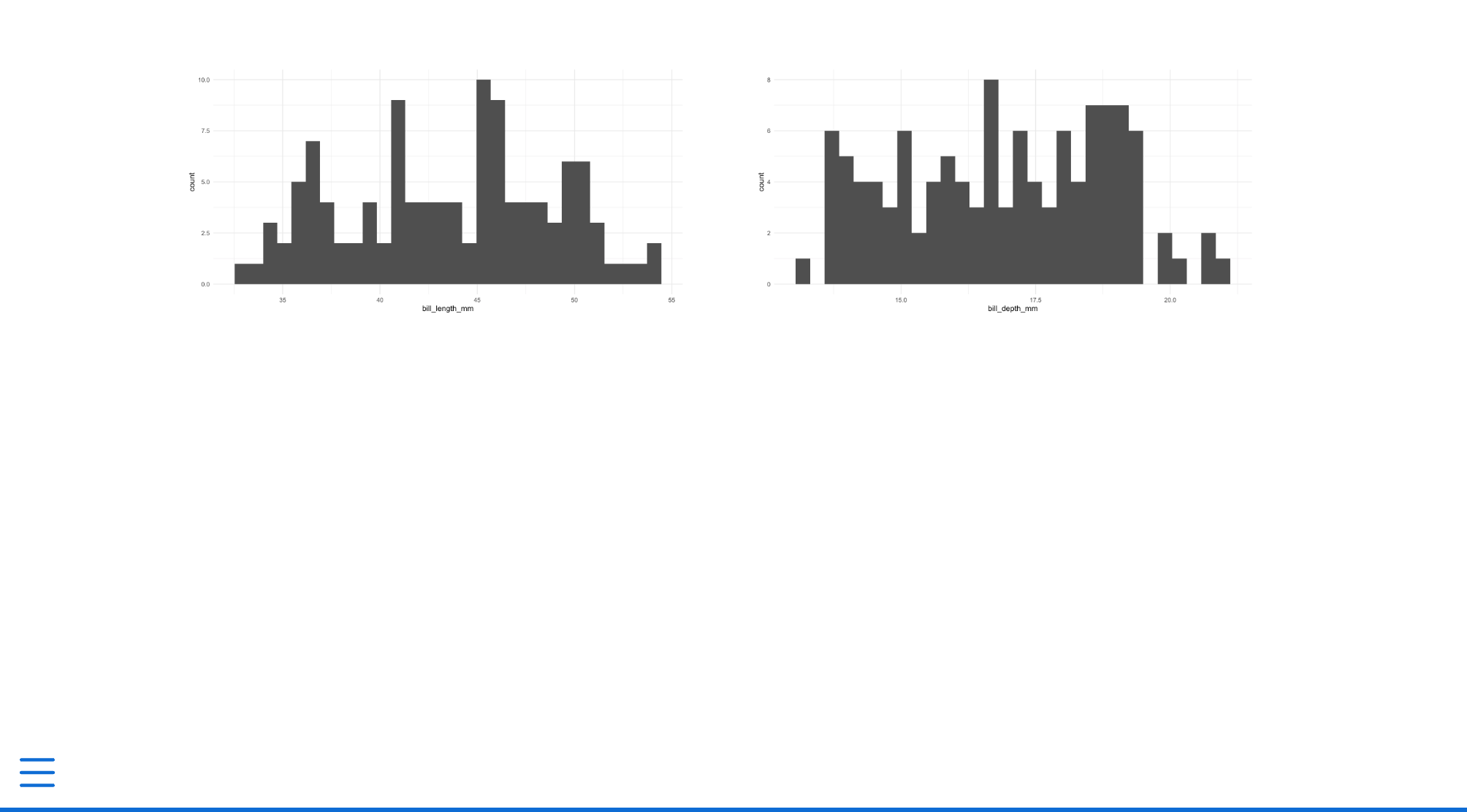
:::

This pattern using ::: to start and end a section creates a section in the resulting HTML file known as a <div>. The HTML <div> tag allows you to define properties within that section. In the code above, adding {.incremental} sets a custom CSS class that makes the list reveal incrementally.

Aligning Content

We see a similar pattern when creating columns in Quarto slides. Let’s say we want to create a slide with content in two columns, as in Figure 10-5.

[F10005.png]



* + - * 1. A slide with two columns

The code below is what I used to create the two-column slide. You’ll see the ::: as well as ::::. This creates nested divs. We first have a columns class, which tells the HTML that all content within the :::: should be laid out as columns. Then, we use ::: {.column width="50%"} to start a div that takes up half of the width of the slide. With both :::: and :::, we have to also have a matching closing :::: and ::: to indicate the end of the section.

:::: {.columns}

::: {.column width="50%"}

```{r}

penguins %>%

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

geom\_histogram() +

theme\_minimal()

```

:::

::: {.column width="50%"}

```{r}

penguins %>%

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

geom\_histogram() +

theme\_minimal()

```

:::

::::

With xaringan we were also able to easily center content on our slides by surrounding it with .center[]. Doing the same thing in Quarto is slightly more complicated. Quarto has no built-in CSS class to center content so we need to create it ourselves. I begin by creating a CSS code chunk and creating a custom class called center-slide. Using a bit of CSS, I tell it to center align all content (despite it saying text-align you’ll see that it also aligns images).

```{css}

.center-slide {

text-align: center;

}

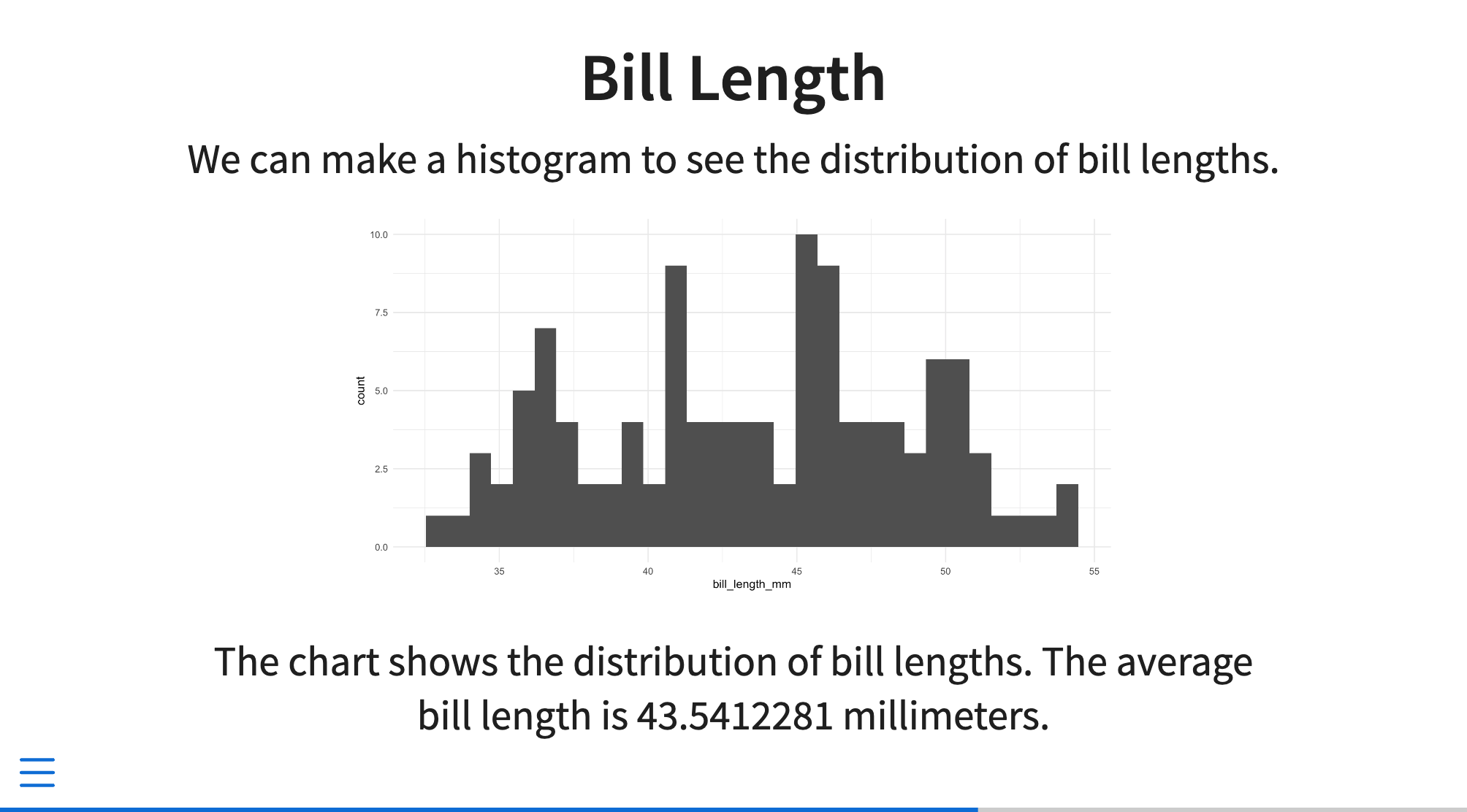
```

I then apply the new center-slide class by putting it next to the title of the slide as follows.

## Bill Length {.center-slide}

With our custom CSS applied, the slide now has all of its content centered, as seen in Figure 10-6.

[F10006.png]



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

Adding Background Images to Slides

Working in xaringan we added a background image to one of our slides. We can do the same thing in Quarto by applying the background-image attribute to a slide as follows.

## Penguins {background-image="penguins.jpg"}

This will add a slide with the text Penguins in front of an image of penguins, as seen in Figure 10-7.

[F10007.png]



* + - * 1. A slide with a background image

Customizing Your Slides with Themes and CSS

We’ve started making some changes to the look-and-feel of our Quarto slides. The sky is the limit in terms of further customizations. As with xaringan, there are two main ways to customize your slides:

1. Using existing themes
2. Changing the CSS

Themes are the easiest way to change what your slides look like. Quarto has many themes you can apply by adding their name to your YAML as follows:

---

title: "Penguins Report"

format:

revealjs:

theme: dark

---

Using this theme will change from the default light theme to a dark theme. The title slide with the dark theme applied can be seen in Figure 10-8.

[F10008.png]



* + - * 1. A slide with the dark theme applied

To see the full list of available themes, go to <https://quarto.org/docs/presentations/revealjs/themes.html>.

You can also write custom CSS to change your slides further. Quarto uses a type of CSS called Sass. The most important practical implication of using Sass is that we can use variables in our CSS. There are a number of variables that we can target to make changes to our slides. If you recall how the xaringanthemer package allowed us to target things like header\_h2\_font\_size and header\_color, using pre-existing variables with Sass will feel very similar.

To start, I’ll create a Sass file called theme.scss and add two sections to it as follows:

/\*-- scss:defaults --\*/

/\*-- scss:rules --\*/

In the scss:defaults section, we can use the Quarto Sass variables. For example, to change the color and size of first-level headers, I would add this code:

/\*-- scss:defaults --\*/

$presentation-heading-color: red;

$presentation-h1-font-size: 150px;

/\*-- scss:rules --\*/

To apply these tweaks to my slides, I need to adjust my YAML to tell Quarto use my custom theme.scss file. I do this as follows:

---

title: "Penguins Reports"

format:

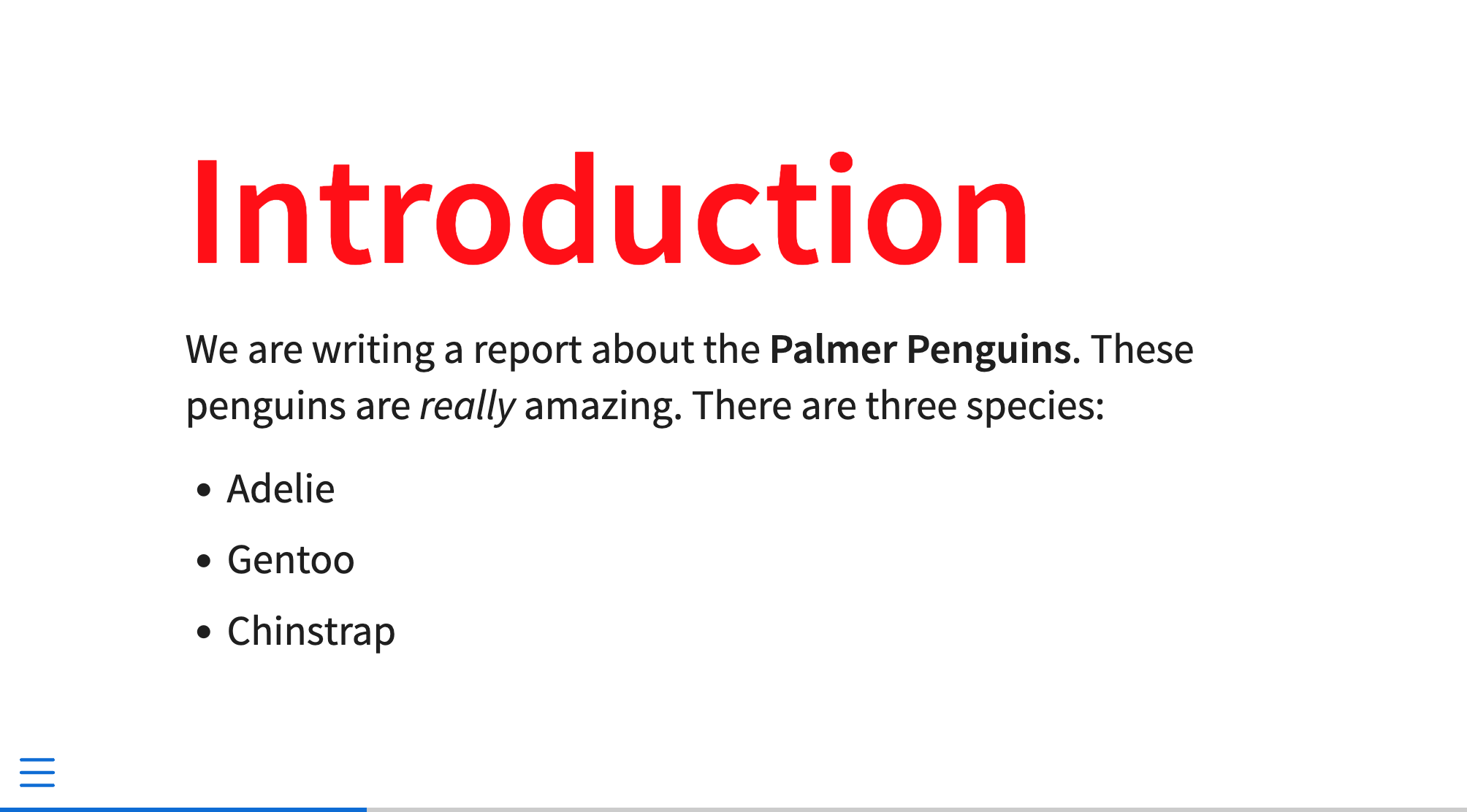
revealjs:

theme: theme.scss

---

Figure 10-9 shows the changes applied to my rendered slides.

[F10009.png]



* + - * 1. A slide with custom CSS applied to it

The scss:defaults section is for using available variables (the full list of variables is available at <https://quarto.org/docs/presentations/revealjs/themes.html#sass-variables>). The sass:rules section is where you would add additional tweaks that can’t be done by targeting variables. For example, I could take my code from above to center all of the content in a slide and add it to the scss:defaults section.

/\*-- scss:defaults --\*/

$presentation-heading-color: red;

$presentation-h1-font-size: 150px;

/\*-- scss:rules --\*/

.center-slide {

text-align: center;

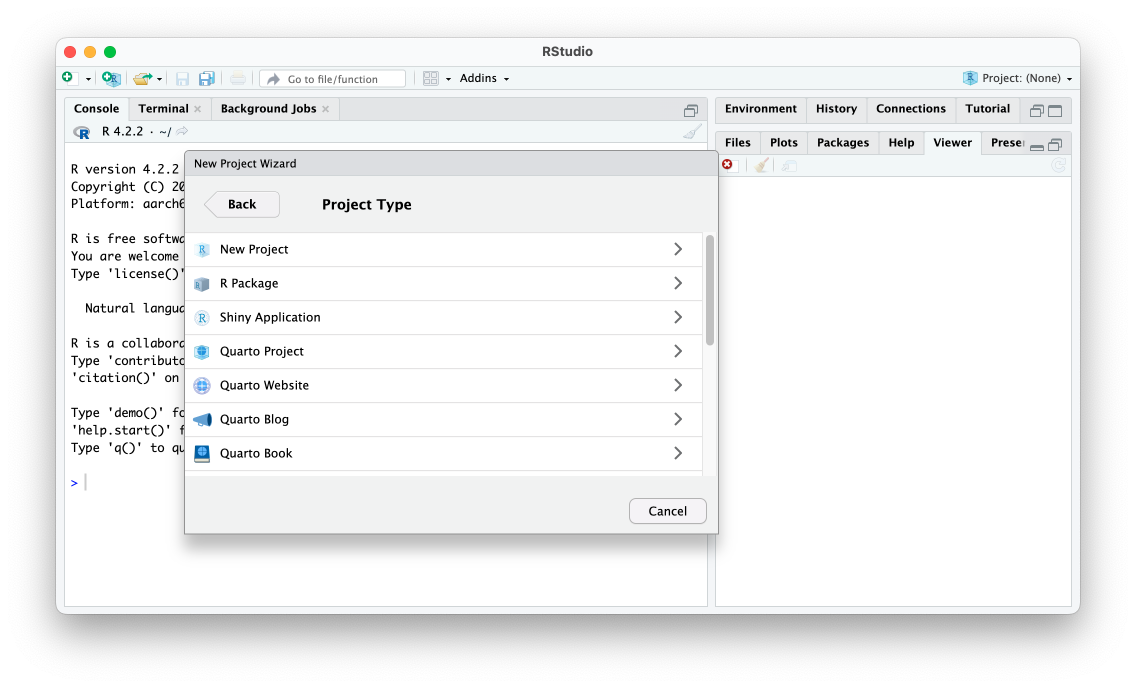
}

Because rendered Quarto slides are HTML documents, you can tweak them however you’d like with custom CSS. What’s more, because our slides use reveal.js under the hood, all of the features built into that Javascript library work in Quarto. There are easy ways to add transitions, animations, interactive content, and much more. The demo Quarto presentation, available at <https://quarto.org/docs/presentations/revealjs/demo/>, shows many of these features in action.

Making Websites with Quarto

Quarto can also make websites. In the same way that we were able to make slides without an external package like xaringan, we can also make websites without an external package like distill. To create a Quarto website, go to File > New Project. Then select New Directory and you’ll be presented with the menu in Figure 10-10.

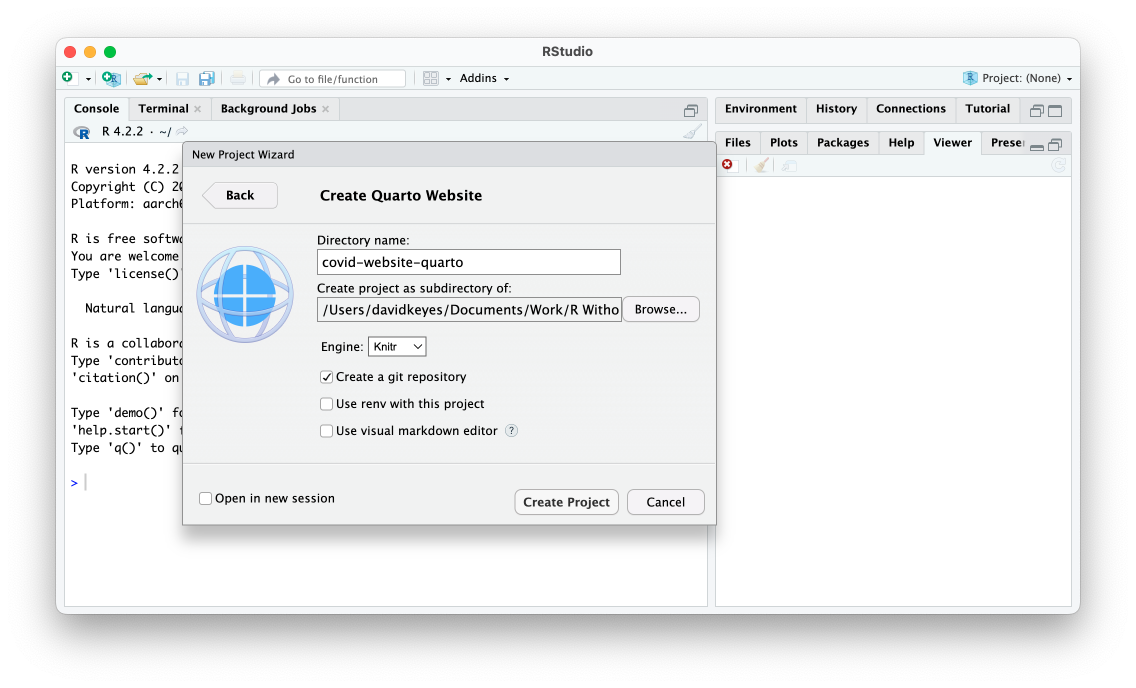
[F10010.png]



* + - * 1. The RStudio new project menu

Select Quarto website and you’ll be prompted to choose a directory to place it in. Keep the default Engine (Knitr), check Create a git repository (this will only show up if you have already installed git) and leave everything else unchecked. Your screen should look like Figure 10-11.

[F10011.png]



* + - * 1. The RStudio menu to create a Quarto website

Once you hit the Create Project button, you’ll see a series of files created. There is an index.qmd and about.qmd file as well as a \_quarto.yml and styles.css file. These files should sound familiar if you read Chapter 10 on making websites with the distill package. The .qmd files are where we’ll add content; the \_quarto.yml file is where we’ll set options for the website as a whole; and the styles.css file is where we’ll add CSS to customize the visual appearance of our website.

Building the Website

Let’s start with the .qmd files. Open index.qmd or about.qmd and you’ll see some default content. I’m going to delete the default content from the index.qmd and replace it with the content from website we made in Chapter 10. The only element I have removed is the layout = "l-page" that we used to make a wider layout for the map (we’ll discuss how to make different layouts in Quarto below).

```{r}

# Load packages

library(tidyverse)

library(janitor)

library(tigris)

library(gt)

library(lubridate)

library(reactable)

```

```{r}

# Import data

us\_states <- states(cb = TRUE,

resolution = "20m",

progress\_bar = FALSE) %>%

shift\_geometry() %>%

clean\_names() %>%

select(geoid, name) %>%

rename(state = name) %>%

filter(state %in% state.name)

covid\_data <- read\_csv("https://raw.githubusercontent.com/nytimes/covid-19-data/master/rolling-averages/us-states.csv") %>%

filter(state %in% state.name) %>%

mutate(geoid = str\_remove(geoid, "USA-"))

most\_recent\_day <- covid\_data %>%

slice\_max(order\_by = date,

n = 1) %>%

distinct(date) %>%

mutate(date\_nice\_format = str\_glue("{month(date, label = TRUE, abbr = FALSE)} {day(date)}, {year(date)}")) %>%

pull(date\_nice\_format)

```

# COVID Death Rates as of `r most\_recent\_day`

This table shows COVID death rates per 100,000 people in four states states.

```{r}

# Make table

covid\_data %>%

slice\_max(order\_by = date,

n = 1) %>%

select(state, deaths\_avg\_per\_100k) %>%

arrange(state) %>%

set\_names("State", "Death rate") %>%

reactable()

```

We can see this same death rate data for all states on a map.

```{r}

# Make map

most\_recent <- us\_states %>%

left\_join(covid\_data, by = "state") %>%

slice\_max(order\_by = date,

n = 1)

most\_recent %>%

ggplot(aes(fill = deaths\_avg\_per\_100k)) +

geom\_sf() +

scale\_fill\_viridis\_c(option = "rocket") +

labs(fill = "Deaths per\n100,000 people") +

theme\_void()

```

# COVID Death Rates Over Time

The following chart shows COVID death rates from the start of COVID in early 2020 until `r most\_recent\_day`.

```{r}

# Make chart

library(plotly)

covid\_chart <- covid\_data %>%

filter(state %in% c("Alabama",

"Alaska",

"Arizona",

"Arkansas")) %>%

ggplot(aes(x = date,

y = deaths\_avg\_per\_100k,

group = state,

fill = deaths\_avg\_per\_100k)) +

geom\_col() +

scale\_fill\_viridis\_c(option = "rocket") +

theme\_minimal() +

labs(title = "Deaths per 100,000 people over time") +

theme(legend.position = "none",

plot.title.position = "plot",

plot.title = element\_text(face = "bold"),

panel.grid.minor = element\_blank(),

axis.title = element\_blank()) +

facet\_wrap(~state,

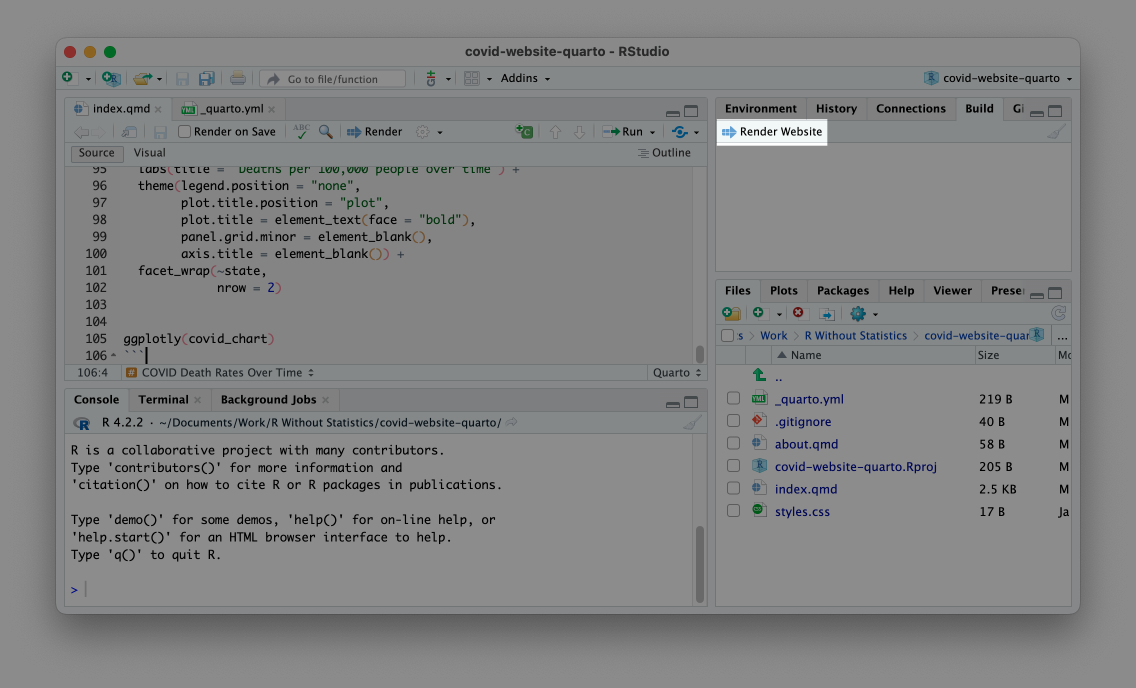
nrow = 2)

ggplotly(covid\_chart)

```

At this point, I could render the index.qmd file individually, but I know I want to render the entire website so let’s do that. You can build a Quarto website just as you did a distill website. Look for the Build tab in the top right of RStudio and click the button that says Render Website. Figure 10-12 shows what you should see.

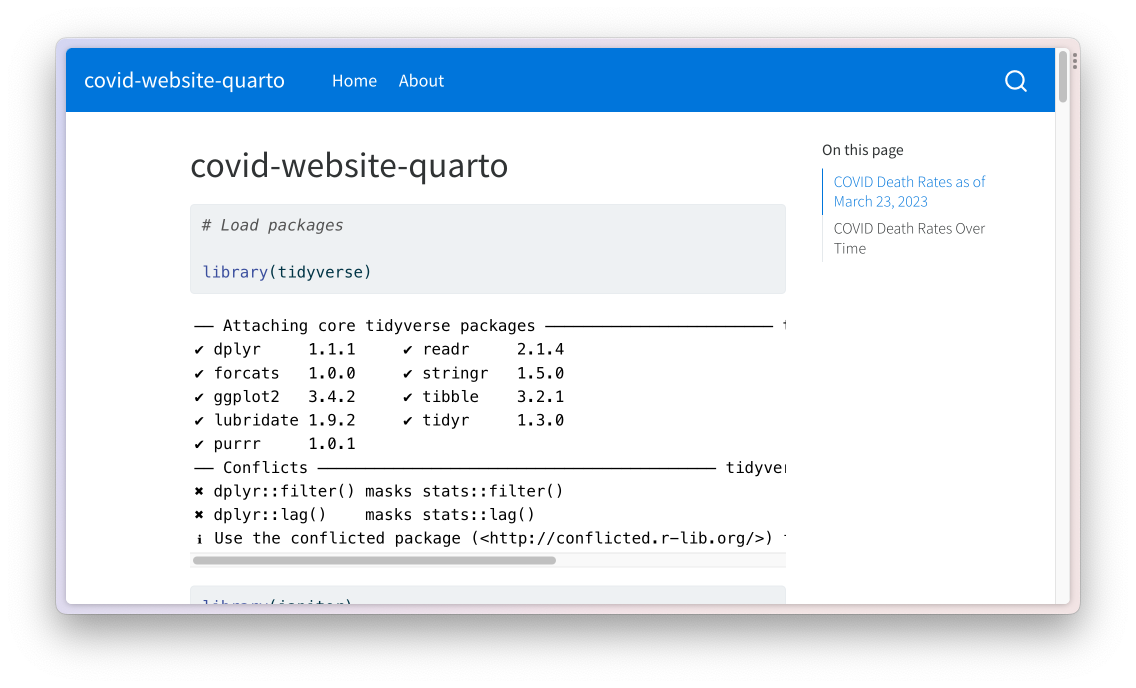
[F10012.png]



* + - * 1. The render website button

The rendered website should now appear in the Viewer pane on the bottom right panel of RStudio. If you go to the Files pane on the same panel you will also see that a \_site folder has been created. If you go into this folder you will see all of the content in your rendered site. I’ll open the index.html file in my web browser. You can see the website in Figure 10-13.

[F10013.png]



* + - * 1. The Quarto website with warnings and messages

As you can see, there are a lot of warnings and messages that we don’t want to show. In R Markdown we removed these in our setup code chunk, but with Quarto we do it in the YAML. I could add this code to the index.qmd YAML, as we did when making a presentation, to remove all code, warnings, and messages from the output:

execute:

echo: false

warning: false

message: false

However, doing this will only make changes to that one file. Instead, we want to add them to the \_quarto.yml file.

Setting Options for the Website

With distill we used the \_site.yml file to make changes to all files in our website. In With Quarto, we use the \_quarto.yml file for the same purpose. If you open the \_quarto.yml file, you should see three sections:

1. The top sets the project type (in this case, a website).
2. The middle section is where we set the title of the website as a whole and determine the options for our navigation bar.
3. The bottom section deals with the appearance.

project:

type: website

website:

title: "covid-website-quarto"

navbar:

left:

- href: index.qmd

text: Home

- about.qmd

format:

html:

theme: cosmo

css: styles.css

toc: true

Let’s start from the bottom. To remove code, warnings, and messages for all pages in our website, add the portion of the YAML we created above to the \_quarto.yml file. The bottom section should now look like this:

format:

html:

theme: cosmo

css: styles.css

toc: true

execute:

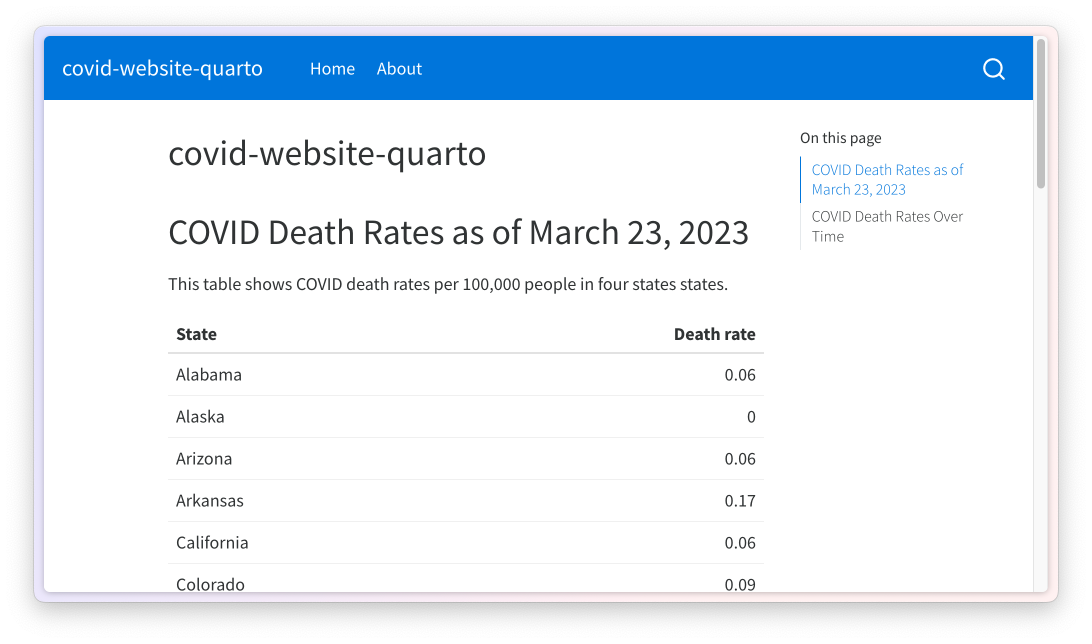
echo: false

warning: false

message: false

If I now build the website again, I will see it, as in Figure 10-14, with just the content I want to show.

[F10014.png]



* + - * 1. The website with warnings and messages removed

You can add any additional options that you would add to a single .qmd file to this section of the \_quarto.yml file and it will apply them across all pages of your website.

Changing the Appearance of the Website with Themes and CSS

The other parts of the bottom section of the \_quarto.yml file deal with the appearance of rendered files. The line theme: cosmo tells Quarto use the cosmo theme (the default when making websites). There are many themes available (the full list is available at <https://quarto.org/docs/output-formats/html-themes.html>) and I can choose a different one to show how it changes the output.

format:

html:

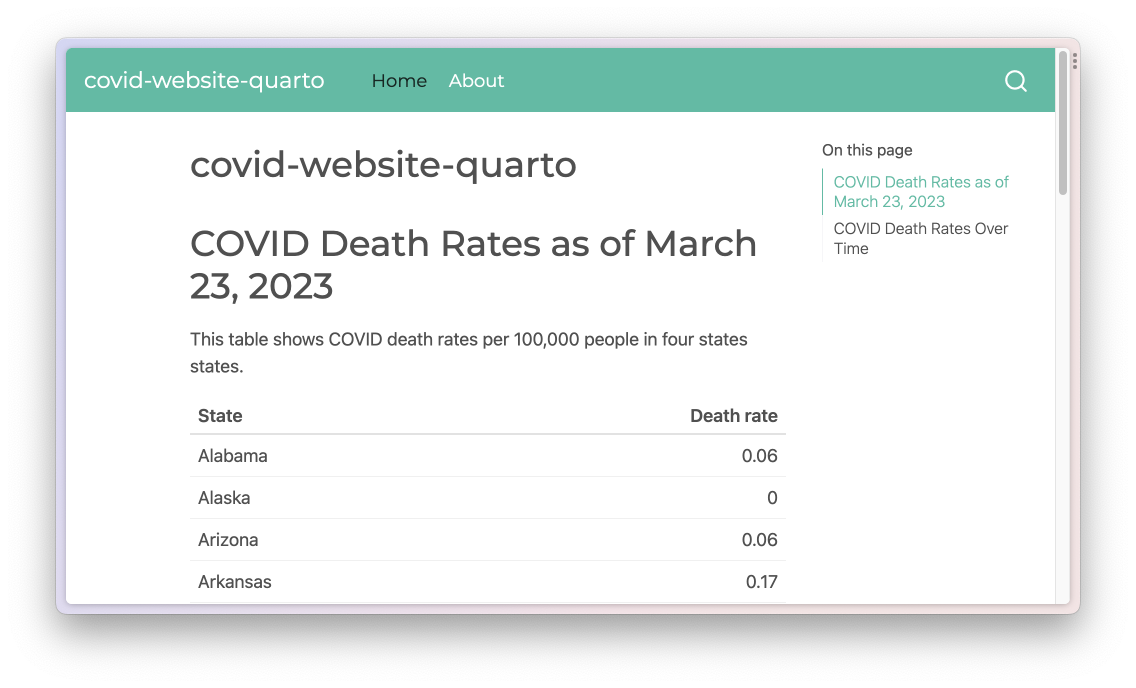
theme: minty

css: styles.css

toc: true

Using the minty theme results in changes to the colors and fonts on the website, as seen in Figure 10-15.

[F10015.png]



* + - * 1. The website with the minty theme

In addition to using pre-built themes, you can also customize your website with CSS. The css: styles.css section in the \_quarto.yml file indicates that Quarto will use any CSS in the styles.css file when rendering. For example, I can add this to the styles.css to make first-level headers red and 50 pixels.

h1 {

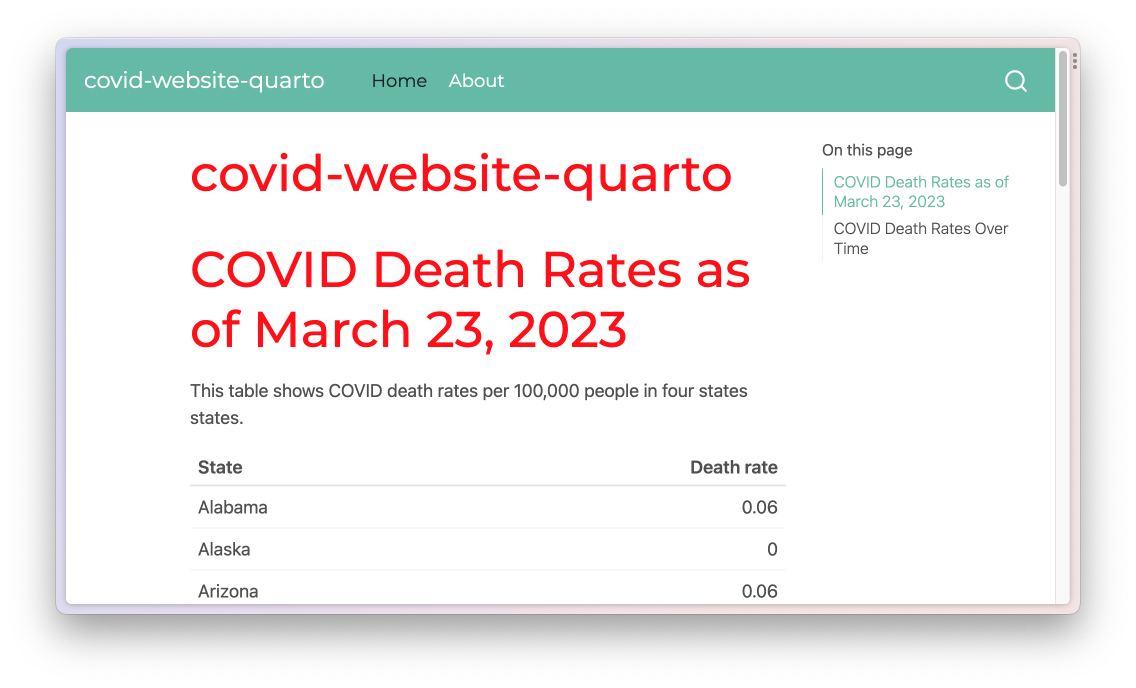
color: red;

font-size: 50px;

}

The re-rendered index.html, seen in Figure 10-16, shows the large red headings.

[F10016.png]



* + - * 1. The website with custom CSS applied

An alternative approach to customizing your website is to use Sass variables. The process for using Sass variables in a .scss file is identical to that laid out above. For example, I can create a file called styles.scss and add a line like this to change the body background to be a light gray color.

/\*-- scss:defaults --\*/

$body-bg: #eeeeee;

To get Quarto to use the styles.scss file, I adjust the theme line as follows.

format:

html:

theme: [minty, styles.scss]

css: styles.css

toc: true

The syntax theme: [minty, styles.scss] tells Quarto to use the minty theme and then make any additional tweaks based on the styles.scss file. If I render my website again, I can see the light gray background throughout. Figure 10-17 shows the current version.

[F10017.png]

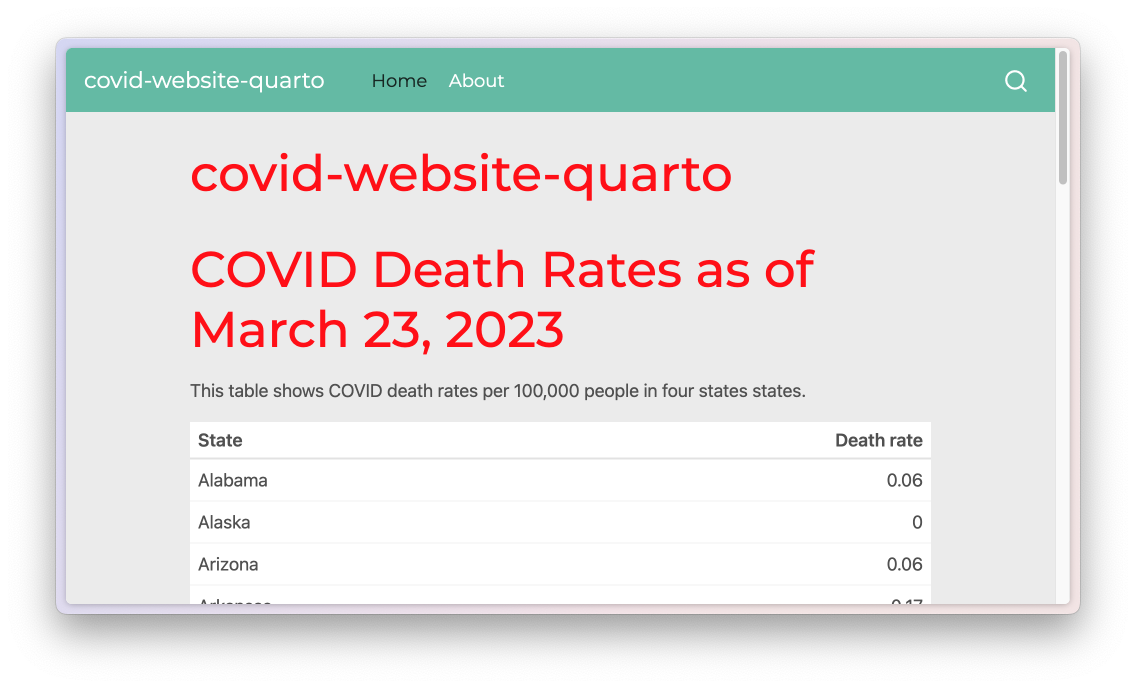


* + - * 1. The website with custom CSS applied through styles.scss

Note that when you add a .SCSS file, the tweaks made in your CSS file are no longer applied. If you wanted to use those, you’d need to add them to the styles.scss file.

The line toc: true creates a table of contents on the side of rendered HTML files that you can see in the screenshots above. You can remove it by changing true to false, as seen in Figure 10-18 below.

[F10018.png]



* + - * 1. The website without the table of contents on pages

This section of the \_quarto.yml file is where you would add any additional options, such as figure height.

Adjusting the Website Title and Navigation Bar

The middle section of the \_quarto.yml file deals with the title of the website as a whole as well as its navigation. I can change the title by adjusting that line. The navbar section functions nearly identically to how it did in Chapter 10 when working with distill. The href is where you list the file you want to link to. The optional text line is where you manually specify the text that should show up for that link. Here are a couple changes to the default \_quarto.yml to change the website title and the text for the About page link.

website:

title: "Quarto COVID Website"

navbar:

left:

- href: index.qmd

text: Home

- href: about.qmd

text: About this Website

Figure 10-19 shows these changes applied to the website.

[F10019.png]



* + - * 1. The website with changes to the navigation bar

The title on the home page is still covid-website-quarto, but you could change that in the index.qmd file.

Finally, we come to the top section of the \_quarto.yml. This is the most straightforward. The line type: website indicates that the current project is a website (other project types include books, wikis, and more).

Creating Wider Layouts

When we made our website with distill we used layout = "l-page" to make the map wider. We can accomplish the same thing with Quarto, though the syntax is slightly different. With Quarto, you can adjust the width of the output using the ::: syntax we saw above to add HTML <div> tags. To make the map larger on our Quarto website, we add :::{.column-screen-inset} at the beginning and ::: at the end of the code block that makes it.

:::{.column-screen-inset}

```{r}

#| out-width: 100%

# Make map

most\_recent <- us\_states %>%

left\_join(covid\_data, by = "state") %>%

slice\_max(order\_by = date,

n = 1)

most\_recent %>%

ggplot(aes(fill = deaths\_avg\_per\_100k)) +

geom\_sf() +

scale\_fill\_viridis\_c(option = "rocket") +

labs(fill = "Deaths per\n100,000 people") +

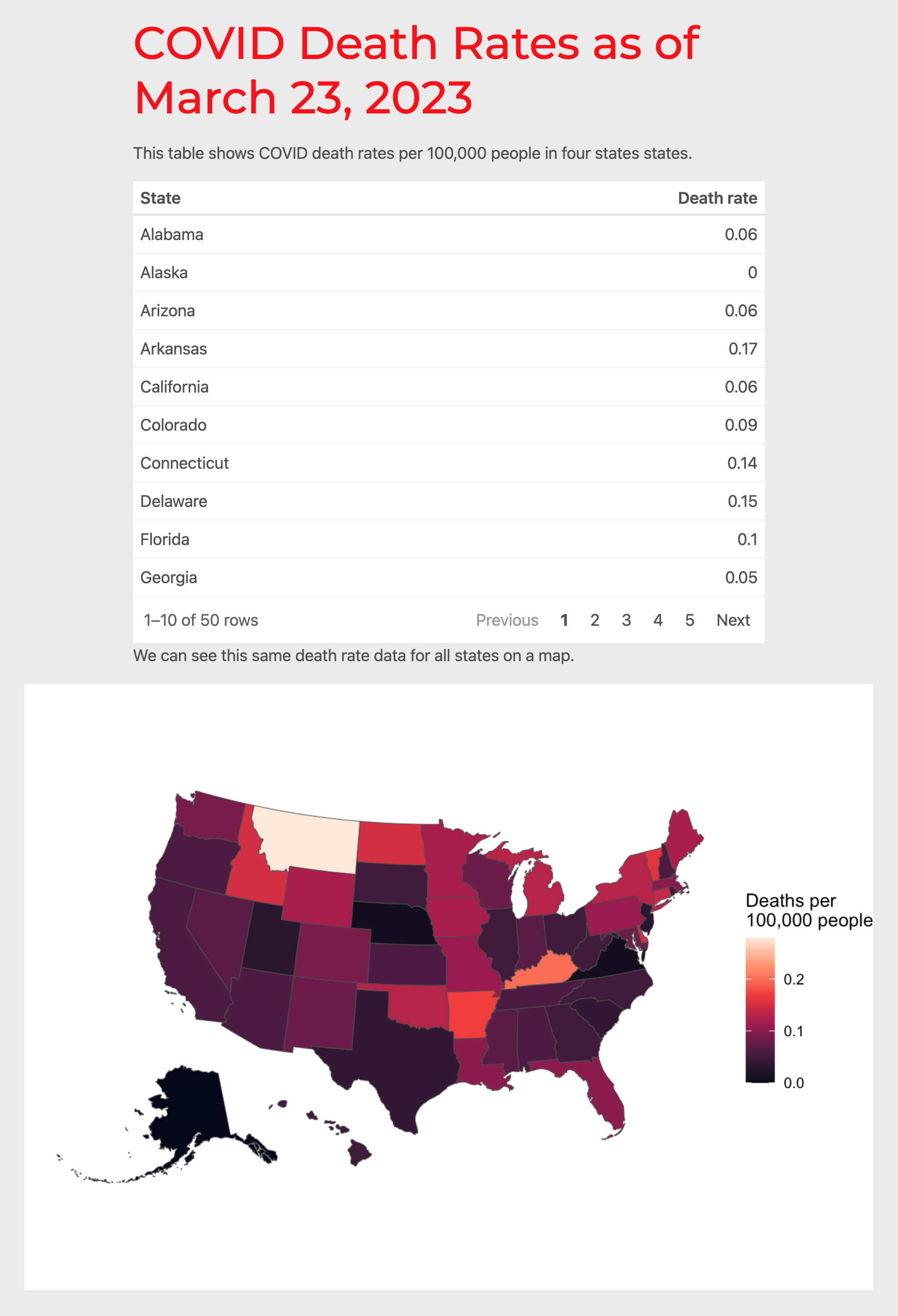
theme\_void()

```

:::

Figure 10-20 shows the wider map.

[F10020.png]



* + - * 1. The website with a wider section

You may have noticed that I also added the line #| out-width: 100% in the code chunk. This is because we need to specify that the map should take up all of the available width. Without this line, the map would only take up a portion of the available width. There are a number of different output widths you can use (the full list is available at <https://quarto.org/docs/authoring/article-layout.html>).

Hosting Your Quarto Website

After we finished making our distill website in Chapter 10, we discussed hosting it using GitHub Pages. You can do the same thing with a Quarto website. Recall that GitHub Pages requires you to publish your website based on the files in the root of your repository or in the docs folder. I’ll do the latter by changing my \_quarto.yml file so that the site outputs in the docs folder. To do this, I add a line to the top section of the \_quarto.yml file.

project:

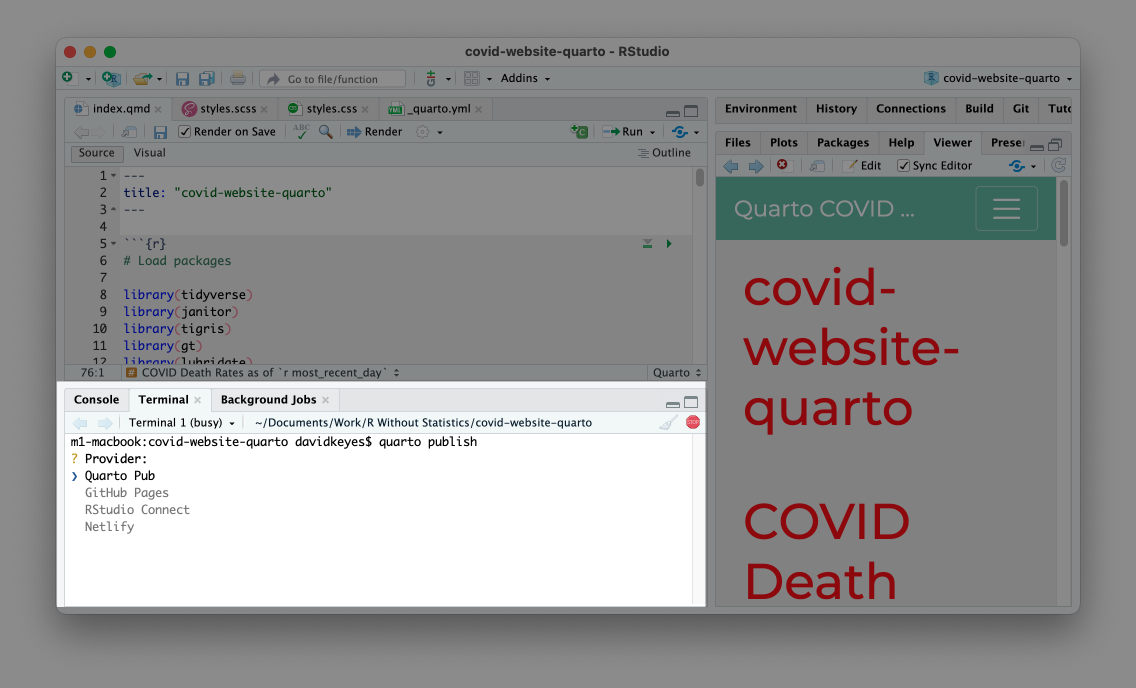
type: website

output-dir: docs

Now, when I render my site, the HTML and other files show up in the docs directory. At this point, I can push my repository to GitHub, adjust the GitHub Pages settings as I did in Chapter 10 and I will be given a URL where my Quarto website will live.

In addition to using GitHub Pages, Quarto has a free service called Quarto Pub that makes it straightforward to get your materials online. Especially if you are not a GitHub user, this is a great option to publish your work. To show you how Quarto Pub works, let me publish the website we just made to it. To get started, look for the Terminal tab on the bottom left panel of RStudio. Click it and at the prompt enter the text quarto publish and hit enter. Doing so will bring up a list of ways you can publish your website. See Figure 10-21 below.

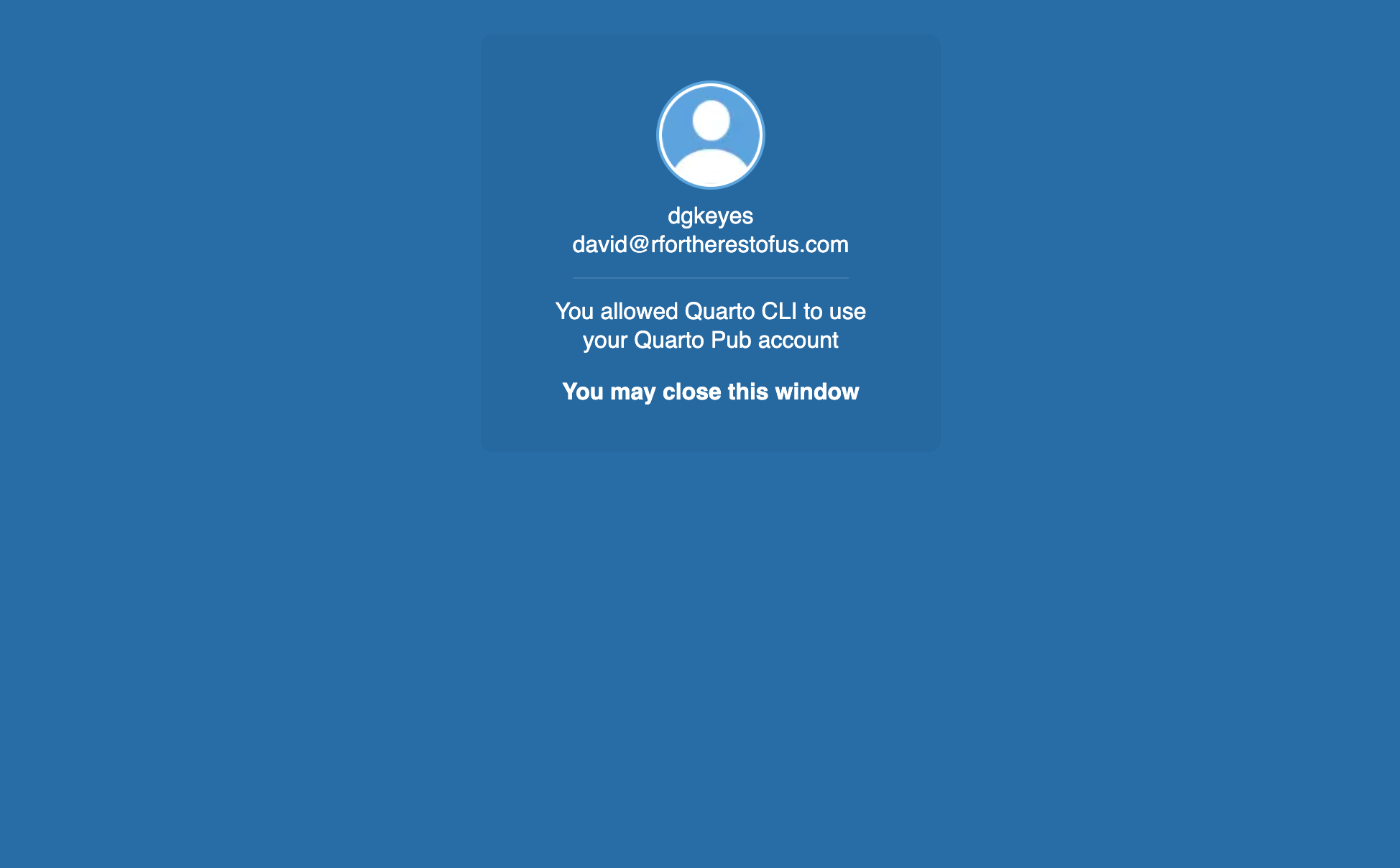
[F10021.png]



* + - * 1. The list of providers to publish your Quarto website

Press enter to select Quarto Pub. You’ll then be asked to authorize. Type Y to do so, which will take you to quartopub.com. Sign up for an account (or sign in if you already have one). You should see a screen like Figure ??, indicating that you have successfully signed in and authorized RStudio to connect with Quarto Pub.

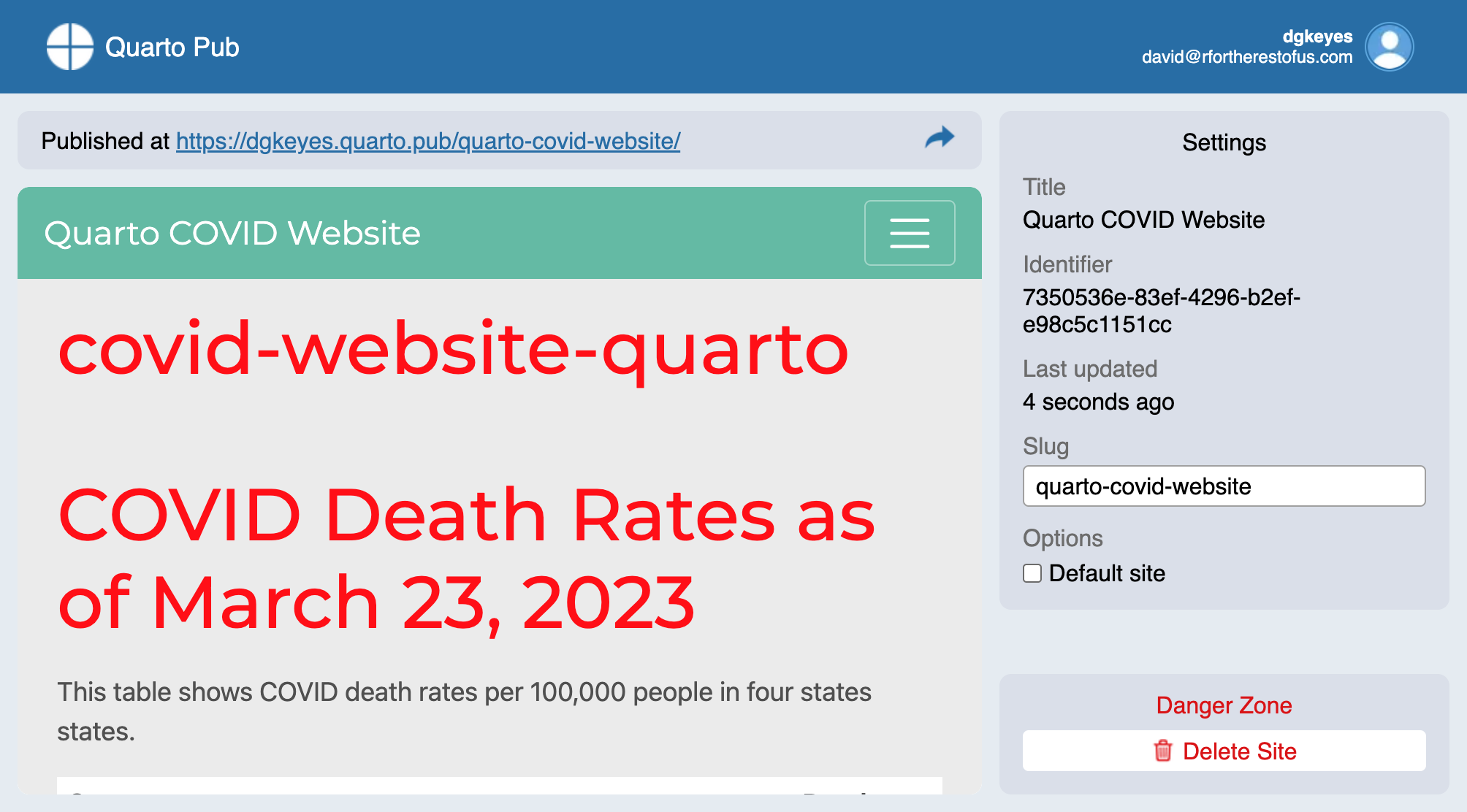
[F10022.png]



* + - * 1. Figure 10-22: The confirmation screen from Quarto Pub

From there, you can return to RStudio. You’ll be prompted to select a name for your website on Quarto Pub. The easiest thing is to select the same name as your project (in my case, that’s quarto-website-covid). Enter the name and hit enter. The site will then be published and you will be taken to the site, as seen in Figure 10-23.

[F10023.png]



* + - * 1. The website published on Quarto Pub

When you make updates to your site, you can republish to Quarto Pub using the same steps. Quarto Pub is probably the easiest way to publish your HTML files made with Quarto.

Conclusion: Should You Use R Markdown or Quarto?

As you’ve seen throughout this chapter, we can do everything we did with R Markdown in Quarto as well. And, unlike R Markdown, we didn’t need to load any external packages to make different outputs. What’s more, the syntax between different output formats is more consistent. For example, because you can make new slides in Quarto by adding first- or second-level headers, the Quarto documents you make to create reports should translate much more easily to presentations.

You’re probably wondering at this point whether you should use R Markdown or Quarto. It’s a big question, and one many in the R community are thinking about.

The first thing to know is that R Markdown is not going away. If you already use R Markdown, you don’t need to switch. As of the writing of this book, I still use R Markdown for most of my work. If you have developed a workflow that uses R Markdown, by all means stick with it.

If you are new to R and have never used R Markdown or Quarto, you may be a good candidate to start with Quarto. The developers of R Markdown have promised to continue its development, but have also said that features they develop for Quarto in the future may not be backported to R Markdown. So, if you’re starting from scratch, start with Quarto.

My strongest recommendation, though, is to use either R Markdown or Quarto. As you’ve seen throughout this chapter, the differences between R Markdown and Quarto are relatively small and the impact on your workflow of switching should be minor. The major impact will come if you switch from a multi-tool workflow to a single tool workflow. Both R Markdown and Quarto can help you become more efficient, avoid manual errors, and share results in a wide variety of formats.