Creating a Report Template in R Markdown 3

Using Parameters 6

Generating Numbers with Parameters 7

Creating an R Script 9

Knitting the Document Using Code 10

Creating a Vector of All States 10

Creating a Tibble with Parameter Data 11

Rendering the Reports 12

Best Practices 13

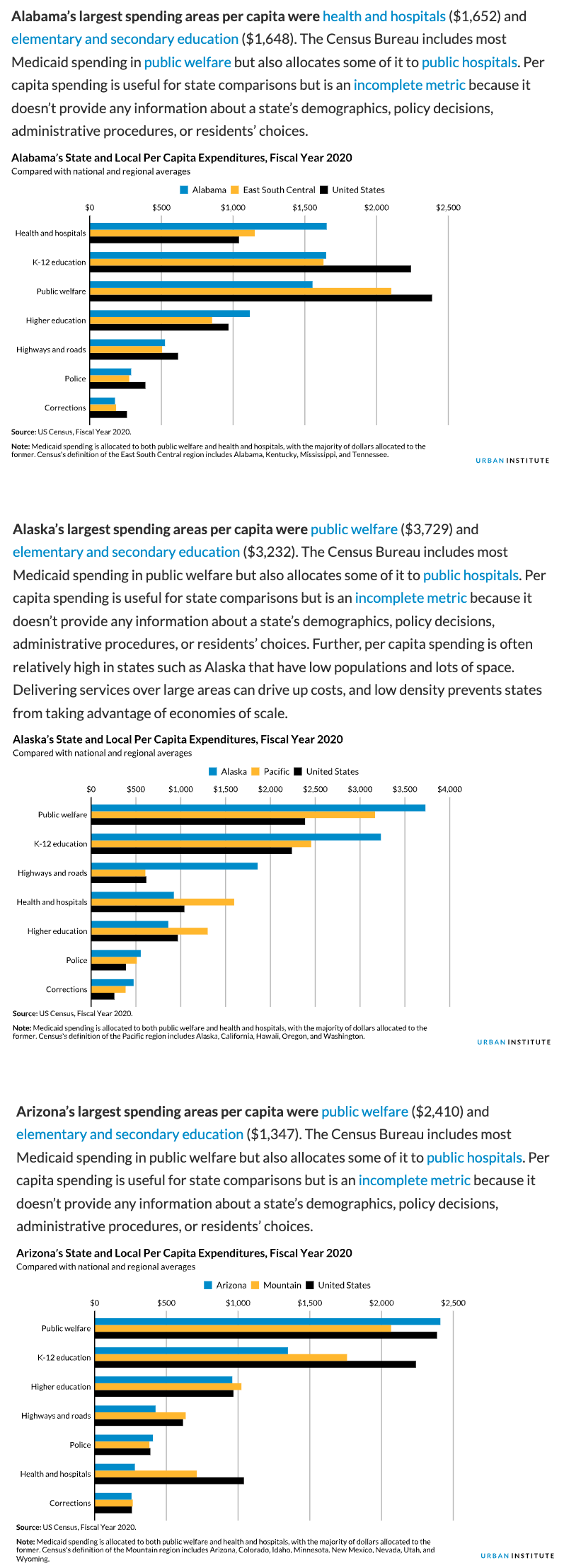
In Conclusion: Parameterized Reporting Makes New Reporting Options Possible 14

Parameterized Reporting

In mid-2020, staff at the Urban Institute, a think tank based in Washington, DC, was tasked with developing fiscal briefs for all US states, as well as the District of Columbia. COVID-19 had brought much economic activity to a halt, and these reports would show just how bad the states’ finances had become.

However, each report required extensive text and multiple charts, so creating them by hand wasn’t feasible. The staff needed a way to automate the process. To do so, employees Safia Sayed, Livia Mucciolo, and Aaron Williams relied on *parameterized reporting*, a technique that uses R Markdown to make multiple reports simultaneously. Their 51 beautiful reports appeared on the Urban Institute website. A snippet is shown in Figure 7-1.

[F07001.png]



* + - * 1. An excerpt from the state fiscal briefs

This chapter explains what parameterized reporting is, then works through a simplified version of the code that the Urban Institute used. We’ll conclude with some reflections on the value of parameterized reporting.

Creating a Report Template in R Markdown

If you’ve ever had to make multiple reports at the same time, you know what a drag it can be, especially if you’re using the multi-tool workflow described in Chapter 6. Making just one report can take a long time. Take that work and multiply it by 10, 20, or, in the case of the team at the Urban Institute, 51, and it can start to feel overwhelming.

Parameterized reporting can generate thousands of reports at once using the following workflow:

1. Make a report template in R Markdown
2. Add a parameter (for example, one representing US states) in the YAML of your R Markdown document
3. Use that parameter to generate a report for one state, to make sure you can knit your document
4. Create a separate R script file with a function that knits your report for one state
5. Run this function for all states

Let’s begin by creating a report template for one state. To do this, I’ve taken the code that the Urban Institute staff used to make their state fiscal briefs and simplified it significantly. Instead of focusing on fiscal data, I’ve used data you may be more familiar with: COVID-19 rates from mid-2022. Here is the R Markdown document:

---  
title: "Urban Institute COVID Report"  
output: html\_document  
params:  
 state: "Alabama"  
---  
  
```{r setup, include=FALSE}  
knitr::opts\_chunk$set(  
 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\_dea

ths\_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)) %>%

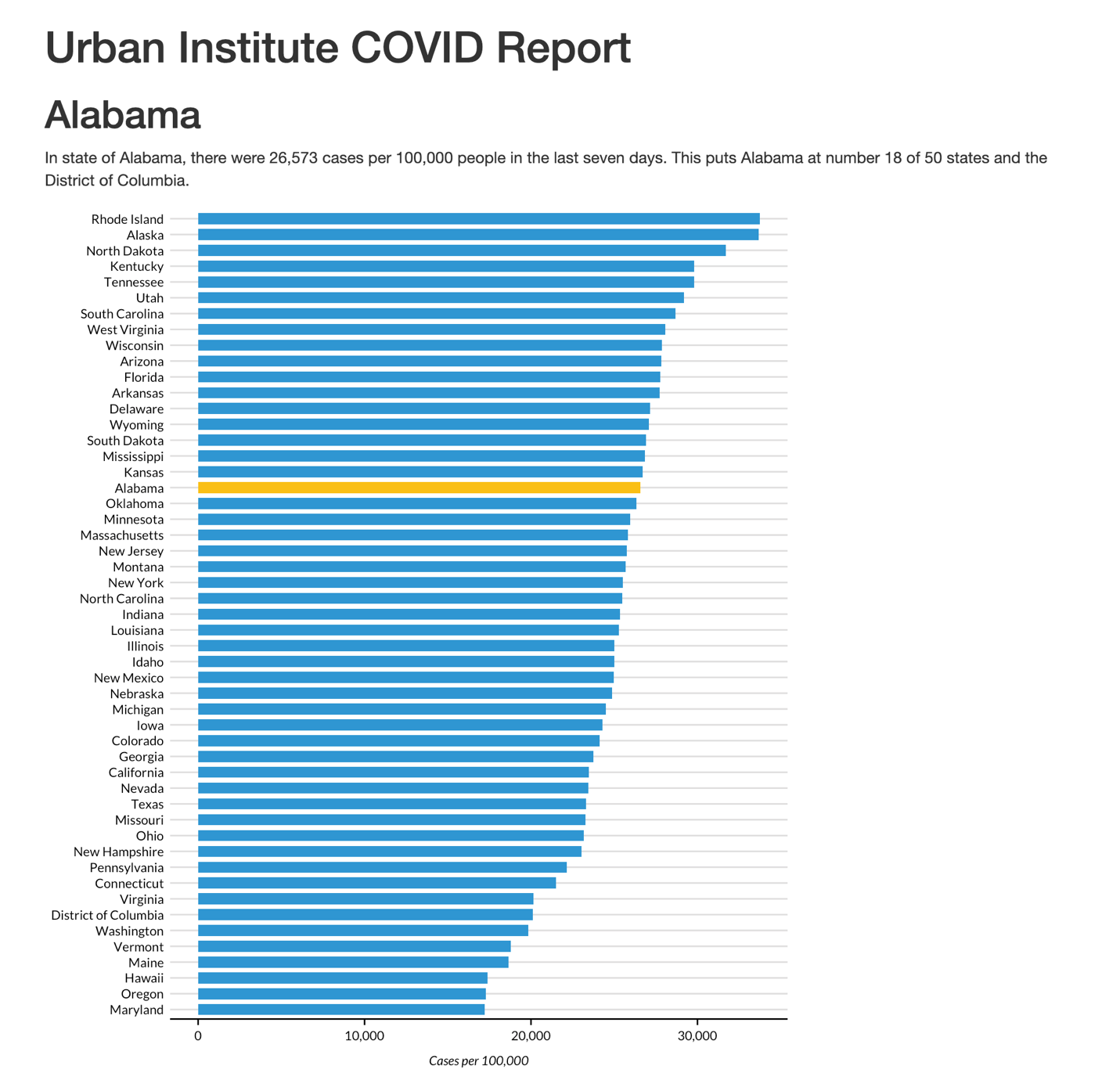
mutate(case\_rank = rank(-cases\_per\_100000, ties.method = "min"))  
```  
  
```{r}  
state\_text <- if\_else(params$state == "District of Columbia", str\_glue("the District of Columbia"), str\_glue("state of {params$state}"))  
  
state\_cases\_per\_100000 <- cases %>%  
 filter(state.name == params$state) %>%  
 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}  
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"  
 )  
```

The text and charts in our report come from the cases data frame, shown here:

#> # A tibble: 51 × 4  
#> total\_cases state.name cases\_per\_100000 case\_rank  
#> <chr> <chr> <dbl> <int>  
#> 1 1302945 Alabama 26573 18  
#> 2 246345 Alaska 33675 2  
#> 3 2025435 Arizona 27827 10  
#> 4 837154 Arkansas 27740 12  
#> 5 9274208 California 23472 35  
#> 6 1388702 Colorado 24115 33  
#> 7 766172 Connecticut 21490 42  
#> 8 264376 Delaware 27150 13  
#> 9 5965411 Florida 27775 11  
#> 10 2521664 Georgia 23750 34  
#> # … with 41 more rows

If we knit the document, we end up with a simple HTML file, shown in Figure 7-2.

[F07002.png]



* + - * 1. A screenshot of the Alabama report

The R Markdown document’s YAML, R code chunks, inline code, and Markdown text should look familiar if you’ve read Chapter 6.

Using Parameters

In R Markdown, *parameters* are variables that we set in the YAML to allow us to create multiple reports. Take a look at these two lines in the YAML:

params:  
 state: "Alabama"

These lines allow us to define a variable, in this case state. We can then use this variable throughout the rest of the R Markdown document using this syntax: params$*variable\_name*, replacing *variable\_name* with state or any name you set in the YAML. For example, take a look at this inline R code:

# `r params$state`

Any instance of params$state will be converted to "Alabama" when we knit it. For example, the line shown here becomes the first-level heading visible in Figure 7-2. This parameter shows up again in the following code:

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.

When we knit our document, we see the following text:

In state of Alabama, there were 26,573 cases per 100,000 people in the last seven days. This puts Alabama at number 18 of 50 states and the District of Columbia.

This text is automatically generated. The inline R code `r state\_text` prints the value of the variable state\_text, whose value is determined by an if\_else() created previously, in this code chunk:

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

If the value of params$states is the District of Columbia, this code makes state\_text equal to "the District of Columbia". If params$state does not equal District of Columbia, then state\_text gets the value "state of", followed by the state name. This allows us to put state\_text in a sentence and have it work no matter whether our state parameter is a state or the District of Columbia.

Generating Numbers with Parameters

We can also use the parameter to generate numeric values to include in the text. The values of the state\_cases\_per\_100000 and state\_cases\_rank variables are also calculated dynamically using our state parameter. This section of code shows how we filter the cases data frame (which has data for all states) to keep just the data for the state in params$state. We then use the pull() function to get a single value and do a bit of formatting with the comma() function from the scales package to make state\_cases\_per\_100000 show up as 26,573 (rather than 26573) before putting these variables into our inline R code.

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)

We can see our parameter used in other places as well. Take a look at this section from the last code chunk. This creates a variable called highlight\_state. Working in the cases data frame, we check if the state.name is equal to params$state. If it is, highlight\_state gets the value “Y”. If not, it gets “N.”

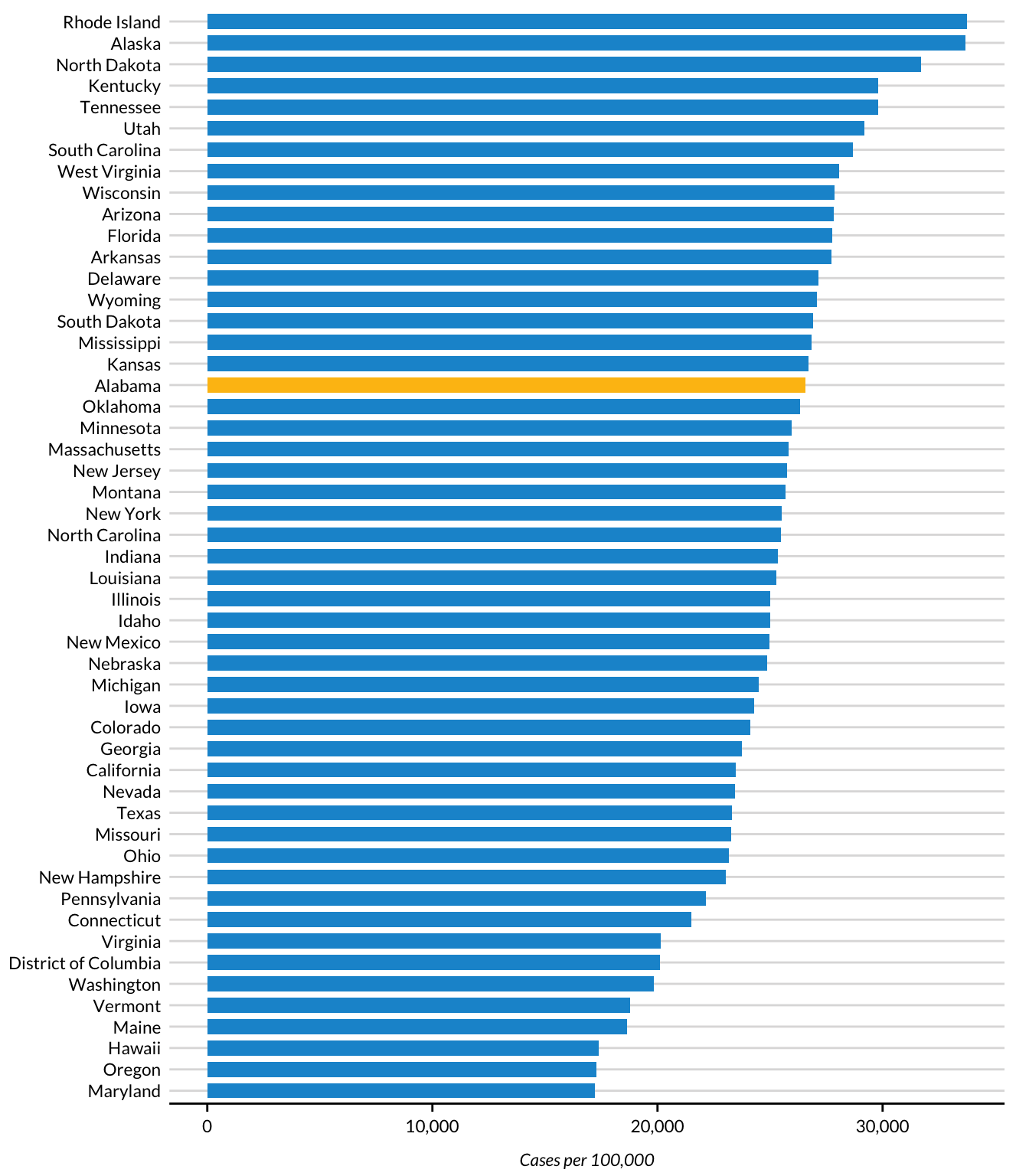
cases %>%  
 mutate(highlight\_state = if\_else(state.name == params$state, "Y", "N"))

We can see what the relevant columns in our data look like after these two lines:

#> # A tibble: 51 × 2  
#> state.name highlight\_state  
#> <chr> <chr>   
#> 1 Alabama Y   
#> 2 Alaska N   
#> 3 Arizona N   
#> 4 Arkansas N   
#> 5 California N   
#> 6 Colorado N   
#> 7 Connecticut N   
#> 8 Delaware N   
#> 9 Florida N   
#> 10 Georgia N   
#> # … with 41 more rows

Later down, our ggplot code uses the highlight\_state variable for the fill aesthetic. What this means is that, when we create our bar chart, the state that is in our variable params$state (Alabama) is highlighted in yellow while all of the other states are blue. If you’re eagle-eyed, you may have noticed a reference to the urbnthemes package and the line of code that says set\_urbn\_defaults(style = "print"). This package provides a custom ggplot theme for all graphs and the set\_urbn\_defaults(style = "print") line applies that theme (in a slightly different way than we saw in Chapter 3, but with the same result). The Urban Institute-styled chart with Alabama highlighted is seen in Figure 7-4 below.

[F07004.png]



* + - * 1. A bar chart showing Alabama highlighted

We’ve now seen how setting a parameter in the YAML gives us the ability to dynamically generate text and charts in our knitted report. But we’ve only generated one report so far. How can we now create all 51 reports? Your first thought might be to manually update the YAML. You could go in, change Alabama to Alaska, and knit again in order to get a report for that state. You could then do this same thing for all states. But it would be tedious, and we’re trying to avoid that. Let’s automate it instead.

Creating an R Script

To automatically generate multiple reports based on our template, we’ll use an R script that changes the values of the parameters in the R Markdown document and then knits it. Begin by creating an R script file named *render.R*.

Knitting the Document Using Code

Our script needs the ability to knit an R Markdown document. While you’ve seen how to do this using the Knit button, you can do the same thing with code. Load the rmarkdown package and then use its render() function, as shown here:

library(rmarkdown)  
render(input = "urban-covid-budget-report.Rmd")

This function generates an HTML document called *urban-covid-budget-report.html*. (By default, the generated file has the same name as the R Markdown document, save the extension.) Change its name by using the output\_file argument:

render(  
 input = "urban-covid-budget-report.Rmd",  
 output\_file = "Alabama.html"  
)

We can also tell the render() function to use parameters we give it. These parameters will override those in the R Markdown document itself. For example, this code would tell R to use Alaska for the state parameter and save the resulting HTML file as *Alaska.html*:

render(  
 input = "urban-covid-budget-report.Rmd",  
 output\_file = "Alaska.html",  
 parms = list(state = "Alaska")  
)

This approach to generating reports works, but to get all 51 reports, we’d have to manually change the state name in our YAML and update the render() function before we run it for each report.

Creating a Vector of All States

Let’s write code that generates all reports for us automatically. First, we must create a *vector* (in colloquial terms, a list of items) of all state names and the District of Columbia. To do this, we’ll use the built-in dataset state.name, which has all 50 state names in a vector:

state <- tibble(state.name) %>%  
 rbind("District of Columbia") %>%  
 pull(state.name)

We turn it into a tibble (which, for our purposes, is the same thing as a data frame) and then use the rbind() function to add the District of Columbia to the list. Finally, we use the pull() function to get one single column and save this as state. Here is what the state vector looks like:

#> [1] "Alabama" "Alaska"   
#> [3] "Arizona" "Arkansas"   
#> [5] "California" "Colorado"   
#> [7] "Connecticut" "Delaware"   
#> [9] "Florida" "Georgia"   
#> [11] "Hawaii" "Idaho"   
#> [13] "Illinois" "Indiana"   
#> [15] "Iowa" "Kansas"   
#> [17] "Kentucky" "Louisiana"   
#> [19] "Maine" "Maryland"   
#> [21] "Massachusetts" "Michigan"   
#> [23] "Minnesota" "Mississippi"   
#> [25] "Missouri" "Montana"   
#> [27] "Nebraska" "Nevada"   
#> [29] "New Hampshire" "New Jersey"   
#> [31] "New Mexico" "New York"   
#> [33] "North Carolina" "North Dakota"   
#> [35] "Ohio" "Oklahoma"   
#> [37] "Oregon" "Pennsylvania"   
#> [39] "Rhode Island" "South Carolina"   
#> [41] "South Dakota" "Tennessee"   
#> [43] "Texas" "Utah"   
#> [45] "Vermont" "Virginia"   
#> [47] "Washington" "West Virginia"   
#> [49] "Wisconsin" "Wyoming"   
#> [51] "District of Columbia"

Creating a Tibble with Parameter Data

Rather than use render() with the input and output\_file arguments, as we did earlier, we can pass it the params argument to give it parameters to use when knitting. Let’s create a tibble with the information needed to render all 51 reports and save it as an object called reports, which we’ll pass to the render() function:

reports <- tibble(  
 input = "urban-covid-budget-report.Rmd",  
 output\_file = str\_glue("{state}.html"),  
 params = map(state, ~ list(state = .))  
)

This code generates a tibble with 51 rows and three variables. In all rows, we set the input variable to the name of our R Markdown document. We set the value of output\_file with str\_glue() to be equal to the name of the state, followed by *.html* (for example, *Alabama.html*).

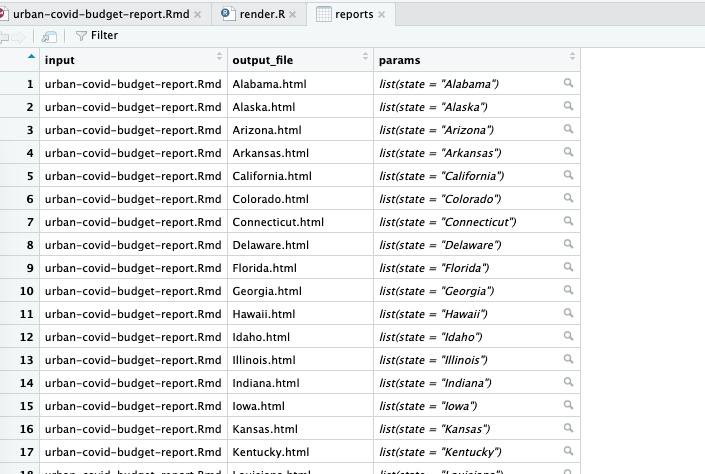
The params variable is the most complicated of the three. It is what’s known as a named list. This data structure is what is needed to use parameters in our R Markdown document, where, for example, we set state to be equal to Alabama. We need our data in this structure so we can put it in the YAML of the R Markdown document; recall the line state: "Alabama" telling R Markdown to create the report for that state. We create the params variable with the map() function from the purrr package, which creates our named list, telling R to set the value of each row as state = "Alabama" and so on for all states.

If we look at the reports tibble, we can see these variables:

#> # A tibble: 51 × 3  
#> input output\_file params   
#> <chr> <glue> <list>   
#> 1 urban-covid-budget-report.Rmd Alabama.html <named list>  
#> 2 urban-covid-budget-report.Rmd Alaska.html <named list>  
#> 3 urban-covid-budget-report.Rmd Arizona.html <named list>  
#> 4 urban-covid-budget-report.Rmd Arkansas.html <named list>  
#> 5 urban-covid-budget-report.Rmd California.ht… <named list>  
#> 6 urban-covid-budget-report.Rmd Colorado.html <named list>  
#> 7 urban-covid-budget-report.Rmd Connecticut.h… <named list>  
#> 8 urban-covid-budget-report.Rmd Delaware.html <named list>  
#> 9 urban-covid-budget-report.Rmd Florida.html <named list>  
#> 10 urban-covid-budget-report.Rmd Georgia.html <named list>  
#> # … with 41 more rows

The params variable shows up as <named list>, but if you open it in the RStudio viewer by clicking reports in your Environment tab, you can see the output more clearly (Figure 7-5).

[F07005.png]



* + - * 1. The named list column shown in the RStudio viewer

This view allows us to see our named list in the params variable, with the state variable equal to the name of the state.

Rendering the Reports

Once we’ve created the reports tibble, we’re ready to render the reports. The code to do so is only one line long:

pwalk(reports, render)

We use the pwalk() function from the purrr package. This function has two arguments: a data frame or tibble (reports, in our case), and a function that runs for each row of this tibble (render()). Note that we do not include open and closing parentheses when passing this function name to pwalk().

Running this code runs the render() function for each row in reports, passing in the values for input, output\_file, and params. It is the equivalent of entering code like the following to run the render() function for each of the 51 states:

render(  
 input = "urban-covid-budget-report.Rmd",  
 output\_file = "Alabama.html",  
 params = list(state = "Alabama")  
)  
  
render(  
 input = "urban-covid-budget-report.Rmd",  
 output\_file = "Alaska.html",  
 params = list(state = "Alaska")  
)  
  
render(  
 input = "urban-covid-budget-report.Rmd",  
 output\_file = "Arizona.html",  
 params = list(state = "Arizona")  
)

Here is what the full R script file looks like:

# Load packages  
library(tidyverse)  
library(rmarkdown)  
  
# 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.Rmd",  
 output\_file = str\_glue("{state}.html"),  
 params = map(state, ~ list(state = .))  
)  
  
# Generate all of our reports  
pwalk(reports, render)

If you run the pwalk(reports, render) code, you should quickly see 51 HTML documents appear. Each one should consist of a report for that state, complete with a customized graph and accompanying text.

Best Practices

While powerful, parameterized reporting can also present some challenges. For example, the team at the Urban Institute had to consider outliers in their data. In a project about making state-level reports, Washington D.C. is an outlier because it is not technically a state. So, the team knew they would need to alter the language in the text so that it didn’t refer to Washington D.C. as a state. As we saw in this chapter, a quick if\_else() statement made this possible.

Another best practice that the Urban Institute team recommends is to manually generate and review reports with the shortest and longest text length of the parameter you’re working with (in the state fiscal briefs, this would be Iowa, Ohio, or Utah and the District of Columbia). Making and reviewing these reports manually allows you to see places where the length of the text may cause unexpected results. Titles in charts can be cut off, page breaks in PDF or Word documents may be messed up by text that runs onto multiple lines, and so on. A few minutes of manual review early on can make the automated process of generating multiple reports much smoother in the end.

In Conclusion: Parameterized Reporting Makes New Reporting Options Possible

In this chapter, we’ve worked through the state fiscal briefs that the Urban Institute team made using parameterized reporting. Automating the production of 51 reports was a huge time-saver. For a more extreme example, consider another project at the Urban Institute: making county-level reports. With over 3,000 counties in the United States, making these reports by hand is not realistic. Using parameterized reporting, the team can use the same process to make 3,000 reports as it did to make 51.

Parameterized reporting also makes your work more accurate. If the Urban Institute were to make its reports using SPSS, Excel, and Word, there would be a ton of copying and pasting between programs. Humans are fallible, and mistakes occur no matter how hard we try to avoid them. Computers, on the other hand, do not make copy-paste errors. Letting computers handle the tedious work of making multiple reports significantly reduces the chance of error.

As with many things with R, when you’re starting out it can feel like a heavy lift to produce reports using parameterized reporting. Initially, it is. You have to make sure that your code works not just for, say, one state, but for all 51. But once you have your R Markdown document and accompanying R script file for rendering, it is straightforward to produce multiple reports at once, saving you work in the end.