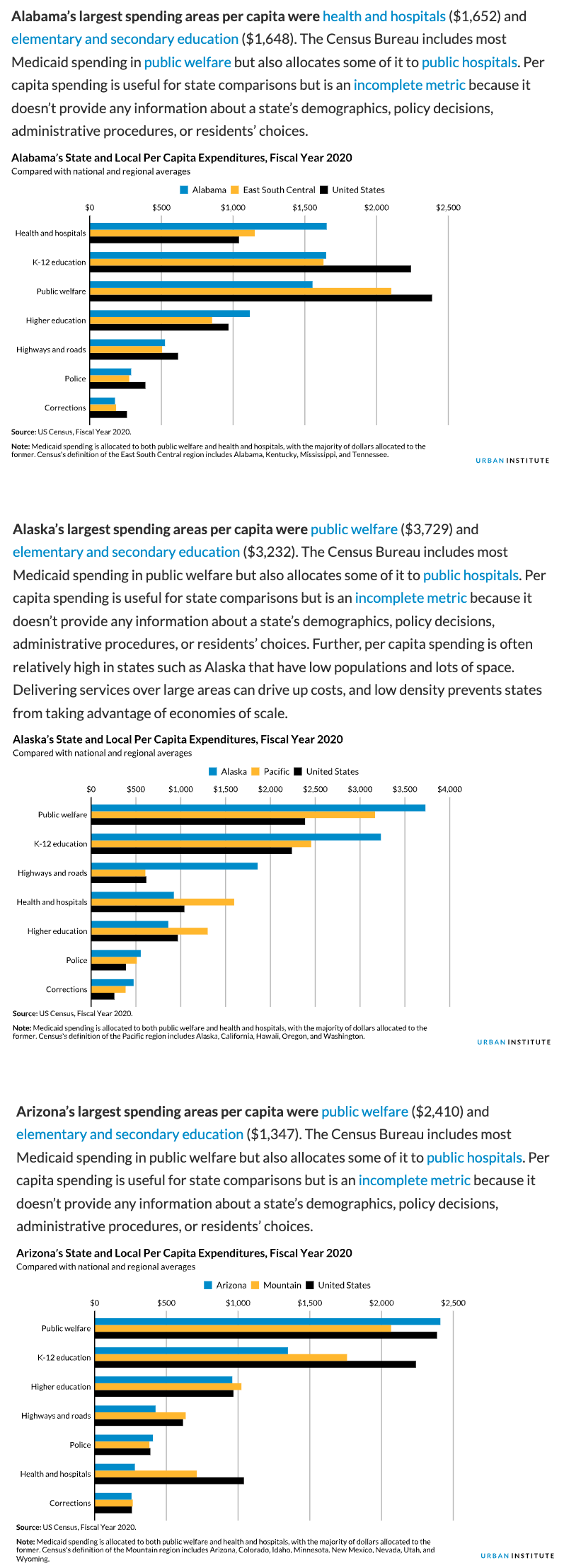
Generate Multiple Reports at Once with Parameterized Reporting

In mid-2020, staff at the Urban Institute were tasked with developing State Fiscal Briefs. The Urban Institute, a Washington D.C.-based think tank, regularly produces economic and social policy research and the State Fiscal Briefs were designed to provide an overview of the fiscal situation of all U.S. states as well as the District of Columbia. At a time when COVID-19 had brought much economic activity to a halt, these reports would show just how bad their fiscal situation had become.

For Urban Institute staff, the main challenge was how to produce these reports. Each one would have extensive text and multiple charts so creating the reports by hand was not feasible. They needed a way to automate the process. Fortunately, the Urban Institute has a strong cadre of R users. Three of them – Safia Sayed, Livia Mucciolo, and Aaron Williams – worked together to create the State Fiscal Briefs using parameterized reporting, a technique that uses R Markdown to make multiple reports simultaneously. Parameterized reporting allowed them to make 51 beautiful reports that could be embedded on the Urban Institute website. A snippet of three of these reports is shown in Figure 7-1.

[F07001.png]



* + - * 1. A portion of three of the State Fiscal Briefs

In this chapter, I’ll begin by explaining what parameterized reporting is. We’ll then work through a simplified version of the type of code that the Urban Institute used in order to demonstrate parameterized reporting in action. We’ll conclude with some reflections on the value of parameterized reporting. Throughout the chapter, you’ll hear insights about parameterized reporting from my interview with the Urban Institute staff members Sayed, Mucciolo, and Williams.

How Parameterized Reporting Works

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 (analysis in SPSS, data visualization in Excel, report writing in Word), it can take a long time to make just one report. Take that amount of work and multiply it by 10, 20, 50 or, in the case of the team at the Urban Institute, 51 and it can start to feel overwhelming. Parameterized reporting is the solution to this problem.

The workflow for making parameterized reports looks like this:

1. Make a report template in R Markdown
2. Add a parameter (for example, state) 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 to knit your report for one state
5. Run this function for all states

This five-step workflow can generate dozens, hundreds, even thousands of reports at once.

Creating an R Markdown Document with Parameters

It might sound a bit complicated so let’s show it in action. I’ve taken the code that the Urban Institute staff used to make their State Fiscal Briefs and simplified it significantly. And, instead of focusing on fiscal data, I’ve used data you may be more familiar with: COVID-19 rates (the data is from mid-2022). We can see the R Markdown document below.

---  
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("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`

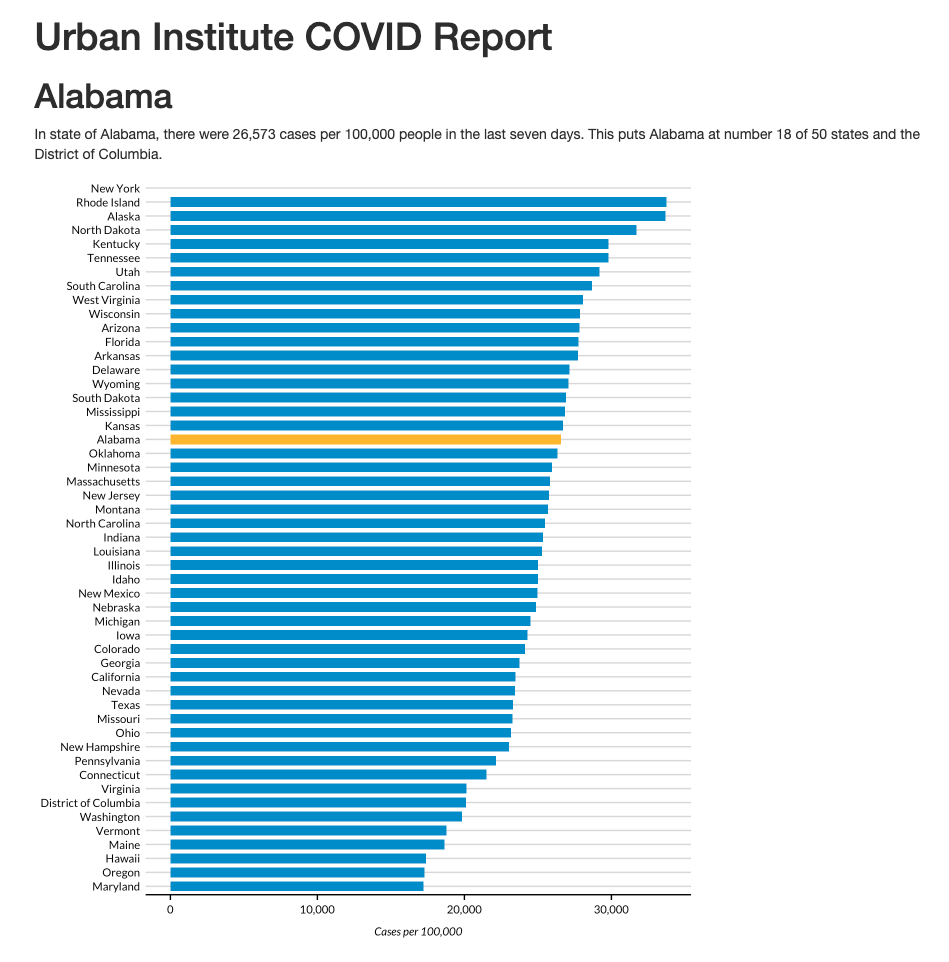
) %>%

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}  
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"  
 )  
```

If we knit this document, we end up with a simple HTML document, seen in Figure 7-2.

[F07002.png]



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

All of the text and charts in our report come the cases data frame. Let’s take a look at that so we can keep it in mind throughout the chapter.

#> # 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

Returning to the R Markdown document, the combination of YAML, R code chunks, and markdown text should look familiar if you’ve read chapter 6. There’s even some inline R code. The one piece that you haven’t seen is the two lines in the YAML that look like this:

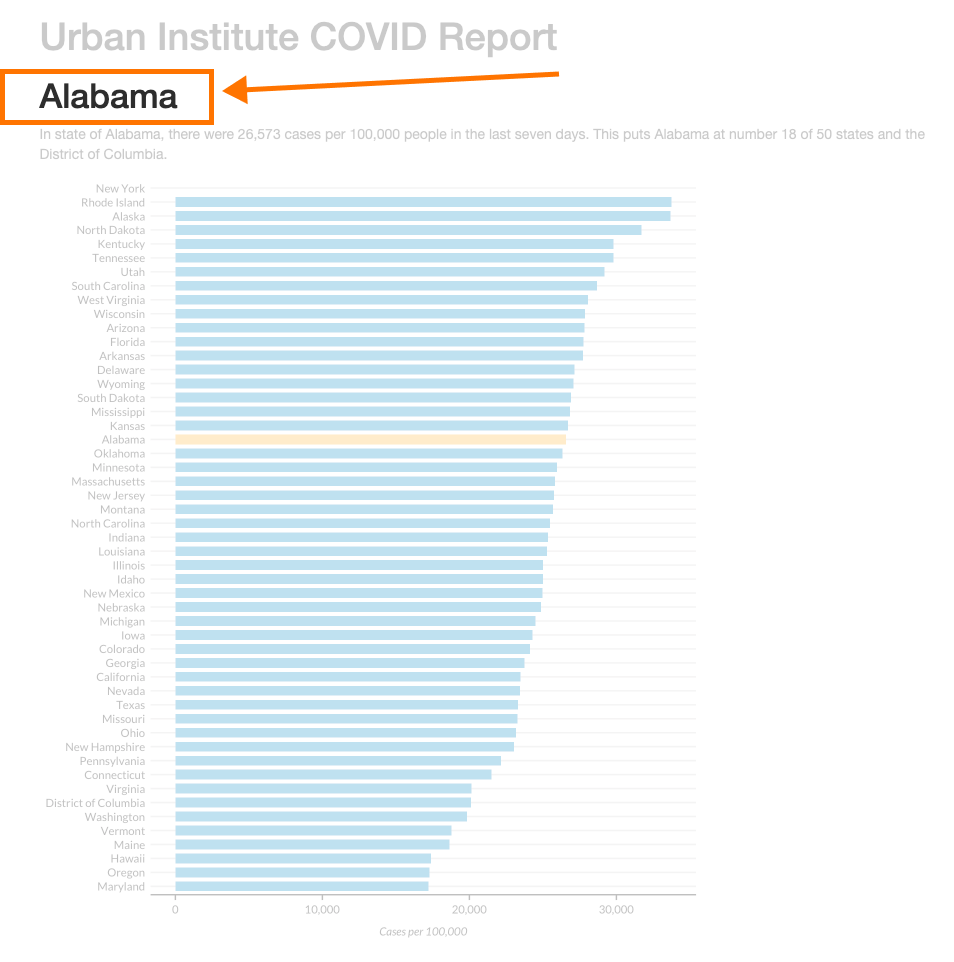
params:  
 state: "Alabama"

These lines allow us to define a variable, in this case state. We can then use this variable throughout the rest of our R Markdown document using this syntax: params$variable\_name (replacing variable\_name with state or any name you set in the YAML). Take a look at this line:

# `r params$state`

Where we see params$state in our inline R code, this is converted to Alabama when we knit (it also becomes a Heading 1 because the line starts with a hash). You can see the result in Figure 7-3.

[F07003.png]



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

This variable created using params in our YAML is the parameter that gives parameterized reporting its name. This use of our parameter to dynamically generate text using inline R code also shows up later on. Take a look at this 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. And state\_text is determined by this if\_else() statement:

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

This line of code says that, if params$states is District of Columbia, then make state\_text equal to “the District of Columbia.” If params$state does not equal District of Columbia, then state\_text gets the value “state of Alabama” (or whatever the state name is). 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.

The values of the state\_cases\_per\_100000 and state\_cases\_rank variables are also calcualted 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)

Writing sentences using inline R code that dynamically updates based on the value of our parameter allows us to create a report for any state with numbers specific to that state.

We can see them 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.”

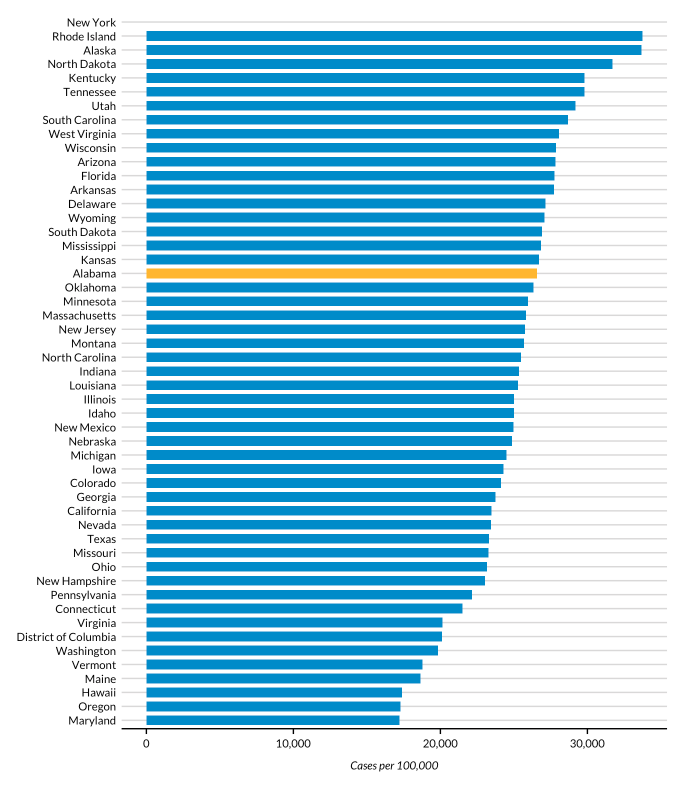
cases %>%  
 mutate(highlight\_state = if\_else(state.name == params$state, "Y", "N"))

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 File to Render Multiple Reports

At this point, we’re going to move out of R Markdown and into an R script file (I tend to call mine render.R). The first thing to know is that, while you’ve seen how to knit an R Markdown document using the Knit button, you can do the same thing with code. There is a package called rmarkdown and a function within it called render(). If I load the rmarkdown package and then use the render() function, as in the code below, the resulting HTML document will be generated.

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

Using this code, our HTML document will be called urban-covid-budget-report.html (the same name as the R Markdown document, but replacing the .Rmd with .html). We can change its name by using the output\_file argument in the render() function.

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

We can also tell the render() function to use parameters we give it (parameters we provide here override those in the R Markdown document itself). This code would tell R to use Alaska for our 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 works, but to get all 51 reports, we’d still have to manually change the state name in our YAML and update the render() function each time before we run it. Things really get interesting when we write code that generates all reports for us automatically. We do this in three steps.

Step 1: Create a Vector of All States

First, we create a vector (in colloquial terms, a list of items) of all state names and the District of Columbia. The team at the Urban Institute does this by using the built-in dataset state.name, which has all 50 state names in a vector. They turn it into a tibble (for our purposes, the same thing as a data frame) and then use the rbind() function to add on the District of Columbia. Finally, they use the pull() function to get one single column and save this as state.

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

We can see what state looks like below.

#> [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"

Step 2: Create a Tibble with Data to Render All Reports

The second step is to create a tibble with information needed to render all 51 reports. We do this by creating an object called reports. This object has multiple arguments that we can pass to the render() function. Above, we used render() with the input and output\_file arguments, but you can also pass the params argument to give it parameters to use when knitting. The code below generates a tibble with 51 rows and three variables.

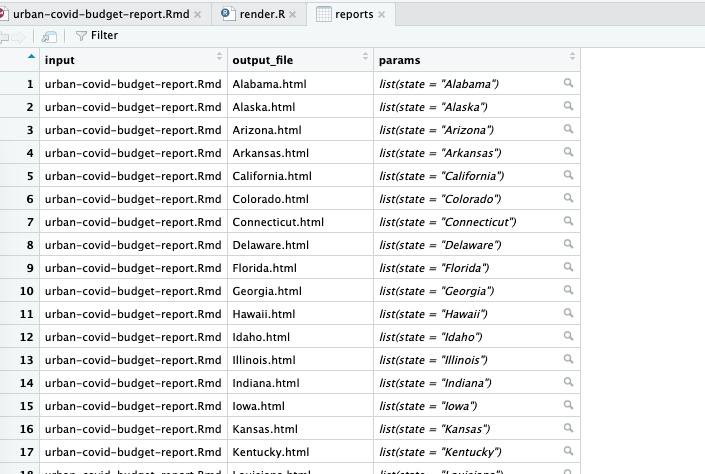
reports <- tibble(  
 input = "urban-covid-budget-report.Rmd",  
 output\_file = str\_glue("{state}.html"),  
 params = map(state, ~ list(state = .))  
)

In all rows, the input variable is set to urban-covid-budget-report.Rmd. The value of output\_file is set 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. 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 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.

#> # 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

If we look at the reports tibble, we can see these variables. The params variable just shows up as <named list>, but if we open it in the viewer (you can do so by clicking the name reports in your Environment tab in RStudio), we can see the output more clearly. Figure 7-5 shows this.

[F07005.png]



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

Step 3: Render All of the Reports

Once we’ve created the reports tibble, we’re ready for our third and final step: rendering all of our reports. The code that generates all of the reports is only one line. We use the pwalk() function from the purrr package. This function has two arguments: 1) a data frame or tibble (reports in our case), and 2) a function to run on each row of this tibble (render though note that you do not include open and close parentheses typically seen with functions).

pwalk(reports, render)

When we run this code, it runs the render() function for each row in reports, each time passing in the values for input, output\_file, and params. It is the equivalent of typing out code like this that runs 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")  
)  
  
# And so on for all states

Putting all three steps together, here’s what my 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)

Being able to tell R to run the Alabama row to generate its report, then the Alaska row, then the Arizona row, and then all other states is a huge time-saver. If I run the pwalk(reports, render) code, I will quickly see 51 HTML documents appear. And each one will be a report for that state, complete with a customized graph and accompanying text.

Best Practices for Working with Parameterized Reporting

Working with parameterized reporting is incredibly powerful. It can also present some challenges. I asked the team at the Urban Institute for their thoughts on best practices for parameterized reporting. Their feedback focused on one topic: considering outliers.

The first example they gave is one we have already seen: Washington D.C. In a project making state-level reports, Washington D.C. is an outlier because it is not technically a state. Having done extensive parameterized reporting, the Urban Institute team knew that they would need to alter the language in the text so that it didn’t talk about Washington D.C. as a state. As we saw above, a quick if\_else() statement made it possible to avoid referring to Washington D.C. as a state in the inline R code.

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 and 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 example of the State Fiscal Briefs that the Urban Institute team made using parameterized reporting. Automating the production of 51 reports was a huge time-saver, though at this scale, it still would be feasible to make these reports by hand. But near the end of our interview, Aaron Williams gave me an example of another project the Urban Institute team works on. This project involves making county-level reports. With over 3,000 counties in the United States, making these reports by hand is not realistic. With parameterized reporting, though, it is the same exact process to make 3,000 reports as it is to make 51. Parameterized reporting makes new reporting options possible.

Parameterized reporting also makes your work more accurate. If the Urban Institute were to make these reports using, say, 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 Safia Sayed told me, “using R Markdown made it not only quicker, but much more accurate, and we were able to pull in more information because of that and make more interesting calculations and observations.”

In Chapter 6, I talked about reproducibility as being able to update a monthly report. Parameterized reporting is another example of reproducibility. Not only can we reproduce reports across time, we can also reproduce them across all states (or any other parameter). It’s just another example of how R doesn’t simply replace other tools, but makes things you had previously considered impossible suddenly feel possible.

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. There can be challenges with outliers. You have to master the syntax to knit your reports from an R script file. But once you have your R Markdown document and accompanying R script file for rendering, it is straightforward to produce multiple reports at once. It may be more work in the beginning, but it is far, far less work in the end.