Parameterized Reporting

Parameterized reporting is a technique that uses R Markdown to make multiple reports simultaneously. Using parameterized reporting, you can follow the same process to make 3,000 reports as you did to make one report. The technique also makes your work more accurate, as it avoids copy-and-paste errors.

For example, staff at the Urban Institute, a think tank based in Washington, DC, used parameterized reporting to develop fiscal briefs for all US states, as well as the District of Columbia. Each report required extensive text and multiple charts, so creating them by hand wasn’t feasible. Instead, employees Safia Sayed, Livia Mucciolo, and Aaron Williams automated the process. This chapter explains what parameterized reporting is and works through a simplified version of the code that the Urban Institute used.

Report Templates in R Markdown

If you’ve ever had to make multiple reports at the same time, you know how frustrating 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 to represent the values that will change between reports
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 that sets the value of the parameter and then knits a report
5. Run this script 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\_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"

)

```

The text and charts in the 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

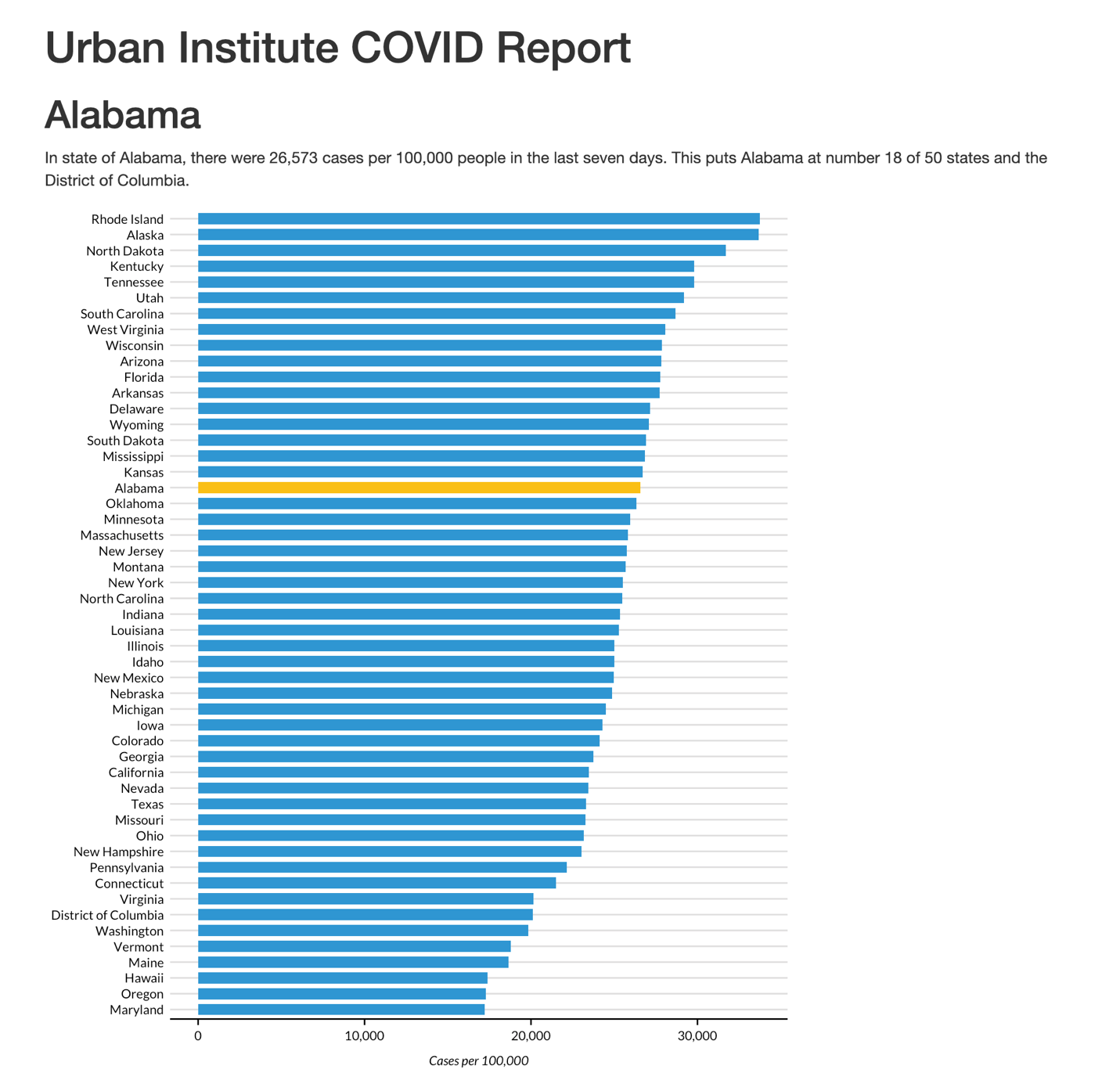
#> 9 5965411 Florida 27775 11

#> 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 7-1.

[F07001.png]



* + - * 1. A screenshot of the Alabama report generated via R Markdown

You should recognize the R Markdown document’s YAML, R code chunks, inline code, and Markdown text from Chapter 6.

Defining 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"

This code defines a variable called state. We can use this variable throughout the rest of the R Markdown document with the params$variable\_name syntax, 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. The line shown here becomes the first-level heading visible in Figure 7-2. This parameter (along with several others) appear 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 the document, we see the following text:

In the 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 a previous call to if\_else(), 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 parameters to generate numeric values to include in the text. For example, we calculate the values of the state\_cases\_per\_100000 and state\_cases\_rank variables dynamically using the state parameter, as shown here:

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 filter the cases data frame (which contains 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 from that data, and format it 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 the inline R code.

Including Parameters in Visualization Code

We can see the parameter used in other places as well, such as to highlight a state in the report’s bar chart. To see how we accomplish this, take a look at the following section from the last code chunk:

cases %>%

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

This code creates a variable called highlight\_state. Within the cases data frame, we check whether state.name is equal to params$state. If it is, highlight\_state gets the value Y. If not, it gets N. Here is what the relevant columns in the data look like after we run 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

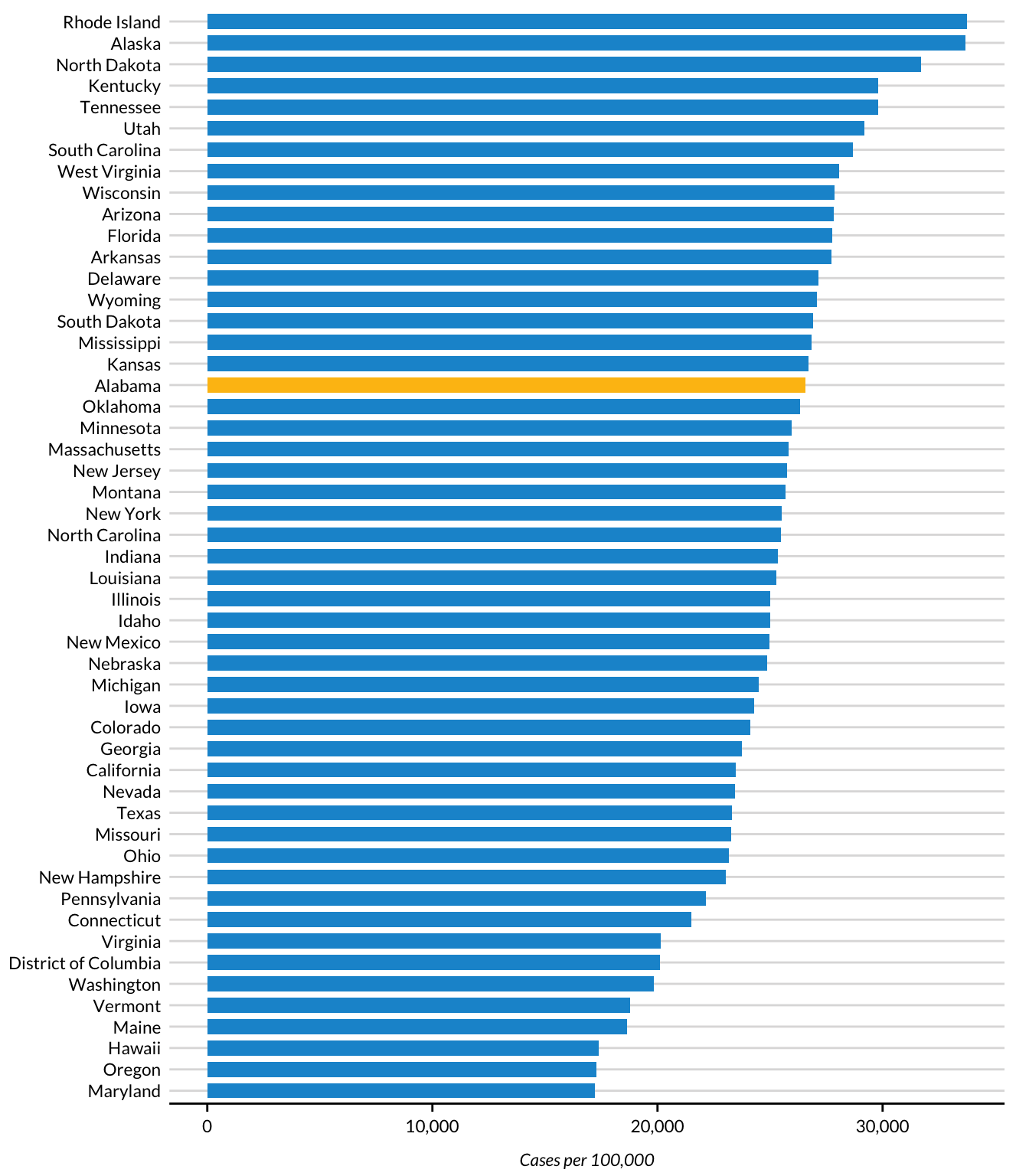
#> 9 Florida N

#> 10 Georgia N

#> # … with 41 more rows

Later, the ggplot code uses the highlight\_state variable for the bar chart’s fill aesthetic, highlighting the state in params$state in yellow and coloring the other states blue. Figure 7-2 shows the chart with Alabama highlighted.

[F07002.png]



* + - * 1. Highlighting data in a bar chart using parameters

You’ve seen how setting a parameter in the YAML gives us the ability to dynamically generate text and charts in the knitted report. But we’ve only generated one report so far. How can we create all 51 reports? Your first thought might be to manually update the YAML by changing the parameter’s value from Alabama to, say, Alaska, then knitting the document again. While you could follow this process for all states, it would be tedious, and we’re trying to avoid tedium. Let’s automate the report generation instead.

Creating an R Script

To automatically generate multiple reports based on the template we created, we’ll use an R script that changes the value 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 with 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",

output\_file = "Alaska.html",

params = list(state = "Alaska")

)

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, with a different extension. We change its name by using the output\_file argument. We also tell render() to use parameters we give it. These parameters will override those in the R Markdown document itself. For example, the code we’ve written would tell R to use Alaska for the state parameter and save the resulting HTML file as Alaska.html.

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

Creating a Tibble with Parameter Data

Let’s write code that generates all reports for us automatically. First, we must create avector (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 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"

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 the 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 puts the data in the state: state\_name format needed for the R Markdown document’s YAML. We use the map() function from the purrr package to create the named list, telling R to set the value of each row as state = "Alabama", then state = "Alaska", and so on, for all states. If you look at the reports tibble, you 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>

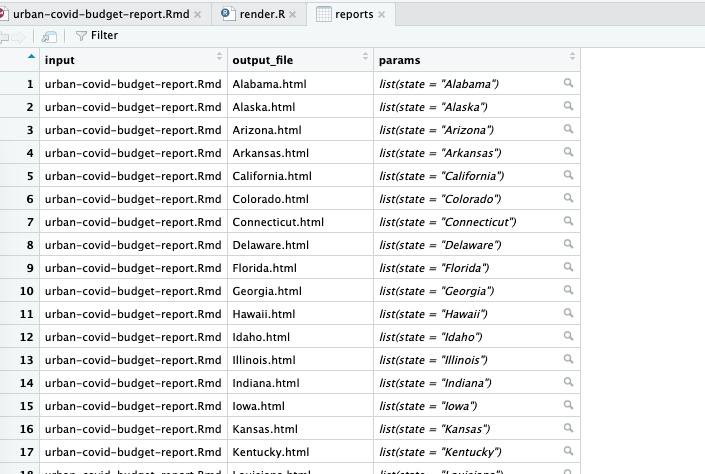
#> 9 urban-covid-budget-report.Rmd Florida.html <named list>

#> 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 the tibble in the RStudio viewer by clicking reports in your Environment tab, you can see the output more clearly (Figure 7-4).

[F07005.png]



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

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

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 see 51 HTML documents appear in the files panel in RStudio. 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, make sure to consider outliers in your data. In the case of the state reports, Washington DC is an outlier because it is not technically a state. The Urban Institute team altered the language in the report text so that it didn’t refer to Washington DC as a state by using an if\_else() statement, as you saw in this chapter.

Another best practice is to manually generate and review the reports whose parameter values have the shortest and longest text lengths. In the state fiscal briefs, these include Iowa, Ohio, or Utah and the District of Columbia. Reviewing these reports manually allows you to identify places where the length of the text may cause unexpected results, such as titles in charts being cut off, page breaks disrupted up by text that runs onto multiple lines, and so on. A few minutes of manual review can make the automated process of generating multiple reports much smoother.

Conclusion

In this chapter, we recreated the Urban Institute’s state fiscal briefs using parameterized reporting. You learned how to add a parameter to your R Markdown document, then use an R script to set the value of that parameter and knit the report.

Automating the production of reports can be a huge time-saver, especially as the number of reports to generate grows. 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. Additionally, if the Urban Institute were to make its reports using SPSS, Excel, and Word, they would have to copy and paste values 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-and-paste errors. Letting computers handle the tedious work of making multiple reports significantly reduces the chance of error.

When you’re starting out, parameterized reporting might feel like a heavy lift, as you have to make sure that your code works for all versions of your report. But once you have your R Markdown document and accompanying R script file, you’ll find it easy to produce multiple reports at once, saving you work in the end.

Learn More

Consult the following resources to learn how the Urban Institute has created parameterized reports and how you can make them yourself:

“Using R Markdown to Track and Publish State Data” by the Data@Urban team (2021), https://urban-institute.medium.com/using-r-markdown-to-track-and-publish-state-data-d1291bfa1ec0

“Iterated fact sheets with R Markdown” by the Data@Urban team (2018), https://urban-institute.medium.com/iterated-fact-sheets-with-r-markdown-d685eb4eafce