Fachhochschule Südwestfalen
University of Applied Sciences

Programmierung mit R - Exercise

(shiny) Web Apps

June 21, SS 2022 || Hannah Behrens

Wir geben Impulse

Today’s task

Remember our appointment from the 3rd of May 2022:

Aim

We are interested in considering the overnight trips of a specific region in a specific state in Australia. Optionally, we want to select a specific purpose for the overnight trips, e.g. we are interested in the time-dependent visits of business people in Adelaide in South Australia.

On the one hand, we want to get and save the filtered data and on the other hand we want to visualize the resulting time series in a nice ggplot. Furthermore, we are also interested in the total number of overnight trips for each region in a state like South Australia.

Today’s aim is to implement our program from the mentioned appointment in form of a web application based on the R package `shiny` (Chang et al. 2021).

First steps of creating a Shiny Web App

1. File → New File → Shiny Web App. . .
2.

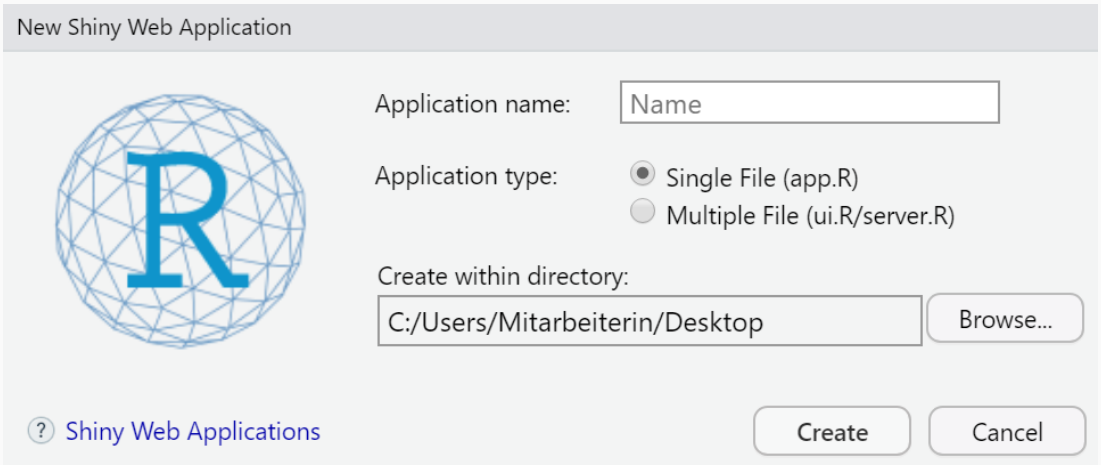

Figure 1: Creating a Shiny Web App.

→ Either a **single** file named `app.R` **or two files** named `ui.R` and `server.R` respectively will be opened. A folder with the application name you tipped in will be created in which the app's file(s) is/are saved.

ui and server

In both cases (`app.R` **or** `ui.R` and `server.R`):

- ui: user interface, where we can interact with the app by selecting/typing in values/words, clicking on buttons, ticking a box and so on
  $\rightarrow$ Consequently, on the ui are different boxes, fields (where we can choose values) and panels (where plot(s), text, etc. will be shown).
- server: we define how the different outputs (boxes, panels, text, tables, etc.) will be generated, when we interact with the ui e.g. by ticking a box

calling `shinyApp(ui = ui, server = server)` to create a Shiny app

Today's main references for constructing Shiny Web Apps: Take a look at the functions of `shiny` (Chang et al. 2021), at the Shiny Cheat Sheet (RStudio, Inc. 2015) and at the Shiny Gallery (RStudio, Inc. 2020).

Task - Sketch of Australian tourism Shiny Web App

Your turn

Make a sketch, how the Shiny Web App for the Australian tourism data should look like. How should the different elements of the ui interact with the server and vice versa?

Sketch of Australian tourism Shiny Web App - answer

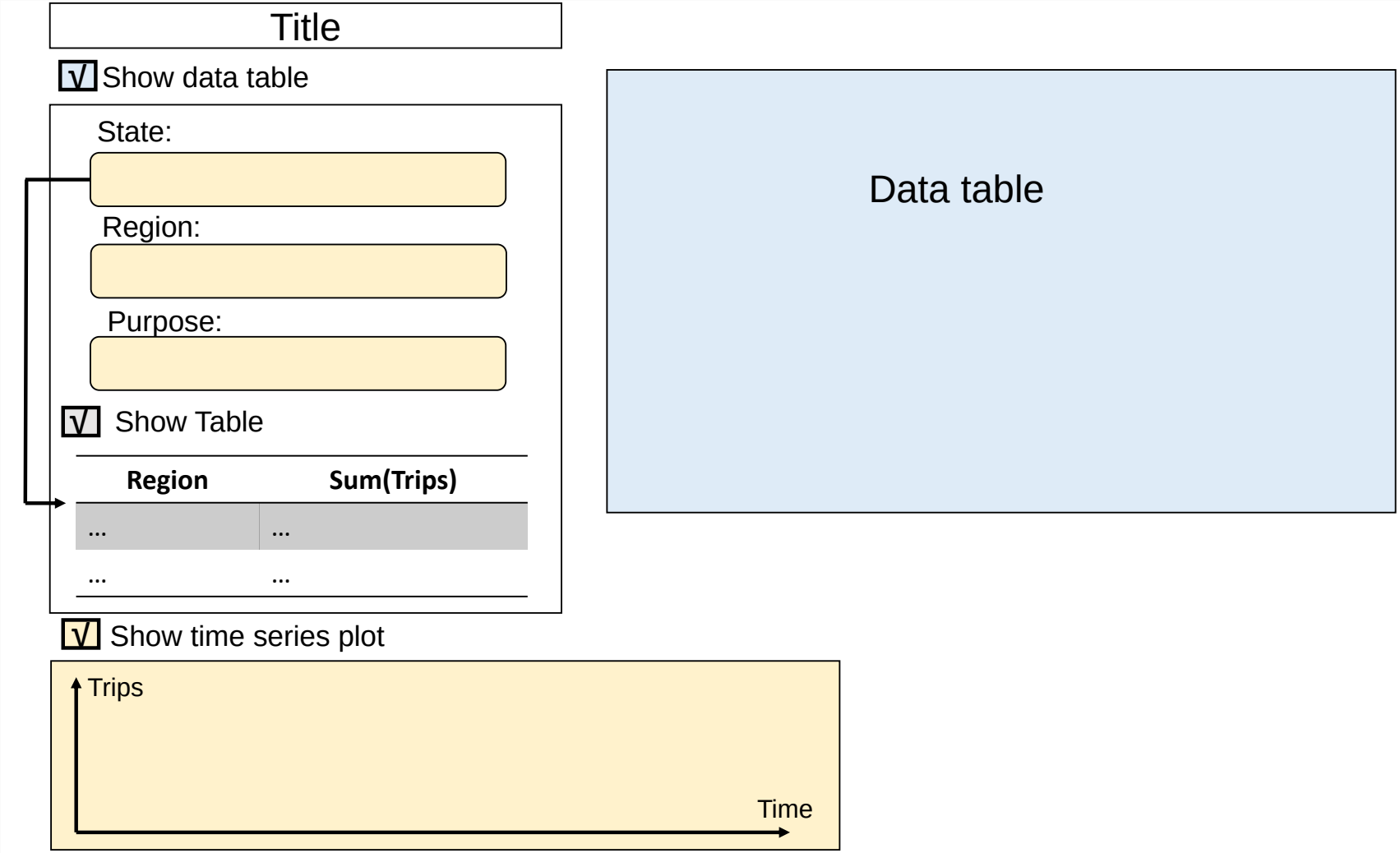

Figure 2: Sketch of Shiny Web App and how its elements are related to each other.

ui

What we have to define on the user interface:

- a title
- a box to tick: either show the data table or not
- a panel where to show the data table
- fields in which to select a state, a region and one or several purposes
- a panel where to output a table in which the sum of trips for every region dependent on the state is listed
- a box to tick: either show the previous table or not
- a box to tick: either show a times series plot or not
- a panel where to output a ggplot in which the number(s) of trips of the filtered time series is (are) plotted

What do we need to create our ui? (RStudio, Inc. 2015)

Your turn

Look at the first page of the Shiny Cheat Sheet and make yourself familiar with the possibilities of *Inputs* and *Outputs* in a Shiny Web App.
Then decide, which type of *Inputs* and *Outputs* we can use to create the different elements of our ui.

What do we need to create our ui? - answer (Chang et al. 2021; RStudio, Inc. 2015)

What we have to define on the user interface:

- a box to tick: either show the data table or not → `checkboxInput()` (*Inputs*)
- to output the data table → `dataTableOutput()` (*Outputs*)
- fields in which to select a state, a region and one or several purposes → `selectInput()` (*Inputs*)
- to output a table in which the sum of trips for every region dependent on the state is listed → `tableOutput()` (*Outputs*)
- a box to tick: show the previous table or not → `checkboxInput()` (*Inputs*)
- a box to tick: show a times series plot or not → `checkboxInput()` (*Inputs*)
- to output a ggplot in which the number(s) of trips of the filtered time series is (are) plotted
  → `plotOutput()` (*Outputs*)

What do we need to create our ui? - answer

We also have to specify the layout, i.e. **where** should the boxes, table, plot etc. be shown? How do we **organize** our ui?

Take a look at the second page of the Shiny Cheat Sheet under *Layouts*:

- add a title in form of a `titlePanel()`
- do the filtering and present the table of the summarized trips in a `sidebarPanel()` and show the data table in a `mainPanel()` next to it, both elements are wrapped in a `sidebarLayout()`
- show the time series plot underneath the `sidebarLayout()`

$\rightarrow$ Any element that is not specified within a layout function will be presented either above or underneath the defined layout.

Create an ui

Your turn

Based on the ideas we have collected create an appropriate ui.

Create an ui - answer

```
ui <- fluidPage( # ui is defined as fluidPage()
    titlePanel("Australian tourism data"),  # application title
    checkboxInput("show_data", "Show data table", TRUE),

    sidebarLayout(
      sidebarPanel(
             selectInput("State", "State:",
                         c("All", unique(as.character(tourism$State)))),
             selectInput("Region", "Region:",
                         c("All", unique(as.character(tourism$Region)))),
             selectInput("Purpose", "Purpose:",
                        c("All", unique(as.character(tourism$Purpose)))),

      checkboxInput("showSumTripsRegion", "Show the sum of trips by region dependent
on the chosen state", TRUE),

      tableOutput("summaryByRegion")
    ), # end sidebarPanel()
    mainPanel(
    DT::dataTableOutput("table"),
    )), # end sidebarLayout()

   checkboxInput("show_plot", "Show time series plot", FALSE),
   plotOutput('plot_ts')
) # end ui
```

ui (Chang et al. 2021; RStudio, Inc. 2015, 2020)

What we have seen:

- the single elements are separated by a comma,
- each element has an **Id** (to access the value!!!),
- some of the elements consist of a label and
- for some elements initial values and/or a list of values are/is specified

Look at the different functions: There might be further arguments to specify.

First run

Your turn

What happens when the ui is defined, the server looks like this

```
server <- function(input, output) {
}
```

and the app is run by

```
shinyApp(ui = ui, server = server)
```

?

First run - answer

It looks like this:

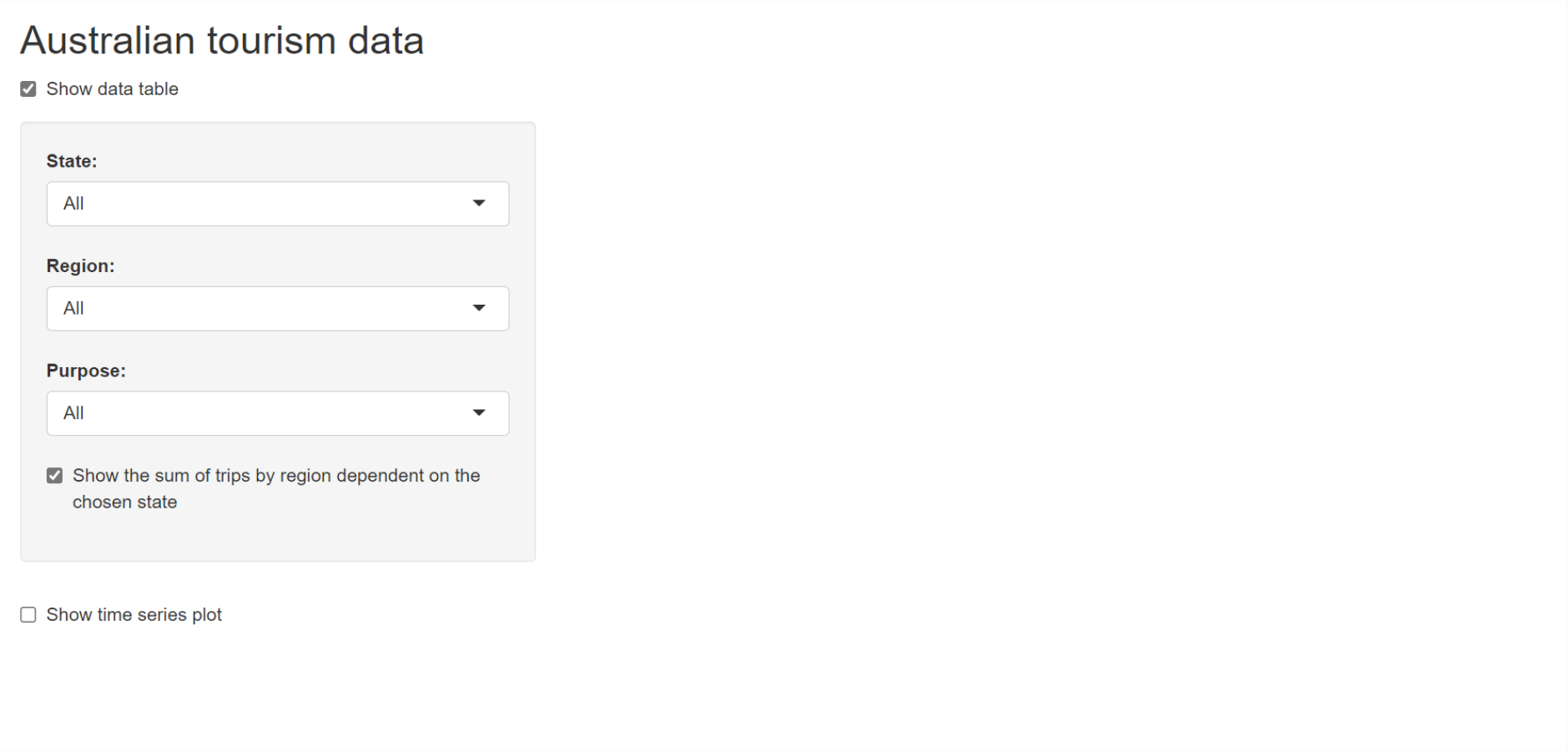

Figure 3: First run of tourism Shiny Web App with a defined ui but an empty server function.

First run - answer

1.) So, we have to define a server.

2.) But before that, we have to optimize our ui since only regions which belong to a pre-selected state shall be listed as well as purposes which belong to a pre-selected state and region.

Optimized ui - uiOutput

```
ui <- fluidPage( # ui is defined as fluidPage()
  # Application title
    titlePanel("Australian tourism data"),
    checkboxInput("show_data", "Show data table", TRUE),

    sidebarLayout(
      sidebarPanel(
            selectInput("State", "State:",
                        c("All", unique(as.character(tourism$State)))),
            uiOutput("Regionselection"), # will be specified in the server function
            uiOutput("Purposeselection"), # will be specified in the server function

      checkboxInput("showSumTripsRegion", "Show the sum of trips by region dependent
on the chosen state", TRUE),

      tableOutput("summaryByRegion")
    ), # end sidebarPanel()
    mainPanel(
    DT::dataTableOutput("table"),
    )), # end sidebarLayout()

   checkboxInput("show_plot", "Show time series plot", FALSE),
   plotOutput('plot_ts')
) # end ui
```

Create a server function (Chang et al. 2021; RStudio, Inc. 2015)

To every element we have defined as an `Output` on the ui belongs an appropriate `render*()` function.
Defining the different types of outputs like plots, tables etc. works like this:

```
output$Id_uiOutput <- renderFunction({
  ...
})
```

Access (a) value(s) of one of the `inputs` on the ui by calling `input$Id_uiInput`, e.g.

```
output$summaryByRegion <- renderTable({
      ...
      if(input$showSumTripsRegion == TRUE){ # checkboxInput whether to show a table
        # in which the sum of trips for every region dependent on the state is listed
        ...
      }
     ...
    })
```

Create a server function (Chang et al. 2021; RStudio, Inc. 2015)

Since we have not defined the possible inputs for the variables `Region` and `Purpose` on the ui, it will be done in the server function based on `renderUI()`:

```
# defining the values of the selectInput for the regions
output$Regionselection <- renderUI({
    selectInput("Region", "Region:",
       choices = c("All", unique(tourism$Region[which(tourism$State == input$State)])))
    })
```

Create a server function - reactive expressions (Chang et al. 2021; RStudio, Inc. 2015, 2019)

How is it possible to get the filtered data and to work with it based on the interactive inputs we choose without running the app each time again when we select another input?

$\rightarrow$ Create a reactive expression like this:

```
selectedData <- reactive({ # reactive expression
        filtered_Data <- ...
      })
```

and call it in the server function by

```
selectedData()
```

Your turn

Look again at the possibilities of creating *Outputs* in the Shiny Cheat Sheet.
Then, create an appropriate server function by defining the outputs and by implementing the filtered data as a reactive expression. Finally, run your app!

Create a server function - answer

```
server <- function(input, output) {
    output$Regionselection <- renderUI({ # defining the values of selectInput: regions
      selectInput("Region", "Region:",
      choices = c("All", unique(tourism$Region[which(tourism$State == input$State)])))
    })

    output$Purposeselection <- renderUI({ # defining the values of selectInput: purposes
      selectInput("Purpose", "Purpose:", choices = c("All",
        unique(tourism$Purpose[which(tourism$State == input$State &
               tourism$Region == input$Region)])))
    })

# table of summarized trips for each region dependent on the selected state
    output$summaryByRegion <- renderTable({
      sum_R <- tourism %>%
        filter(State == input$State) %>%
        group_by(Region) %>%
        summarize(Trips = sum(Trips))
      sum_R <- as.data.frame(sum_R)
      if(input$showSumTripsRegion == TRUE){
        sum_R
      }
    })
# ...
```

Create a server function - answer

```
# generate a datatable which can be filtered, sorted in decreasing/ascending order etc.
      output$table <- DT::renderDataTable(DT::datatable({
        data <- tourism
        if (input$State != "All") {
          data <- data[data$State == input$State,]
        }
        if (input$Region != "All") {
          data <- data[data$Region == input$Region,]
        }
        if (input$Purpose != "All") {
          data <- data[data$Purpose == input$Purpose,]
        }
        if(input$show_data == TRUE){
        data
        }
      }))

      selectedData <- reactive({ # filtered data as a reactive expression
        filtered_Data <- tourism %>% filter(State==input$State, Region == input$Region)
        if (input$Purpose != "All") {
          filtered_Data<-filtered_Data %>% filter(Purpose == input$Purpose)
        }
        filtered_Data
      })
  # ...
```

Create a server function - answer

```
  # create time series plot
    output$plot_ts <- renderPlot({
      if(input$show_plot == TRUE){
      ts_plot <- ggplot(selectedData(), aes(x=as.Date(Quarter),
                                            y = Trips, color = Purpose))+
          geom_line()+
          xlab("Time [Quarter]")+
          ylab("Overnight trips ('000)")
          ts_plot
        }
      })
}
```

Some optimizations (Chang et al. 2021; RStudio, Inc. 2015)

- Creating an interactive plot based on the R package `plotly` (Sievert 2020)
  - by changing `plotOutput()` to `plotlyOutput()` and `renderPlot()` to `renderPlotly()`
- Coloring the app, e.g. adding a background color by using `tags$functionName`:

```
tags$style('.container-fluid {
             background-color: #add8e6;
             }') # for help see for example:
# "Change Background color of fluid page in R shiny", question by chas, November 7,
# 2019 8:55, answer by Eli Berkow, November 7, 2019 9:22 URL: https://stackoverflow.com
# /questions/58745090/change-background-color-of-fluid-page -in-r-shiny
# [accessed: June 21, 2022 9:21]
```

$\rightarrow$ here, HTML tags have been used based on the R package `htmltools` (Cheng et al. 2021)

Helpful links

- Shiny Gallery (Shiny Web Apps - Examples)
- Shiny Homepage
- Shiny Cheat Sheet

References i

Buchwitz, B. 2021. *Computational Statistics*. https://bchwtz.github.io/bchwtz-cswr/.

Chang, Winston, Joe Cheng, JJ Allaire, Carson Sievert, Barret Schloerke, Yihui Xie, Jeff Allen, Jonathan McPherson, Alan Dipert, and Barbara Borges. 2021. *Shiny: Web Application Framework for r*. https://CRAN.R-project.org/package=shiny.

Cheng, Joe, Carson Sievert, Barret Schloerke, Winston Chang, Yihui Xie, and Jeff Allen. 2021. *Htmltools: Tools for HTML*. https://CRAN.R-project.org/package=htmltools.

RStudio, Inc. 2015. *Interactive Web Apps with Shiny Cheat Sheet*. https://shiny.rstudio.com/images/shiny-cheatsheet.pdf.

———. 2019. *Build a Dynamic UI That Reacts to User Input*. https://shiny.rstudio.com/articles/dynamic-ui.html.

———. 2020. *Gallery*. https://shiny.rstudio.com/gallery/.

Sievert, Carson. 2020. *Interactive Web-Based Data Visualization with r, Plotly, and Shiny*. Chapman; Hall/CRC. https://plotly-r.com.