R Shiny Supplement

What is R Shiny?

```
install.packages("shiny")
```

Shiny is an R package that allows you to build interactive web applications straight from R.

Structure of a Shiny app

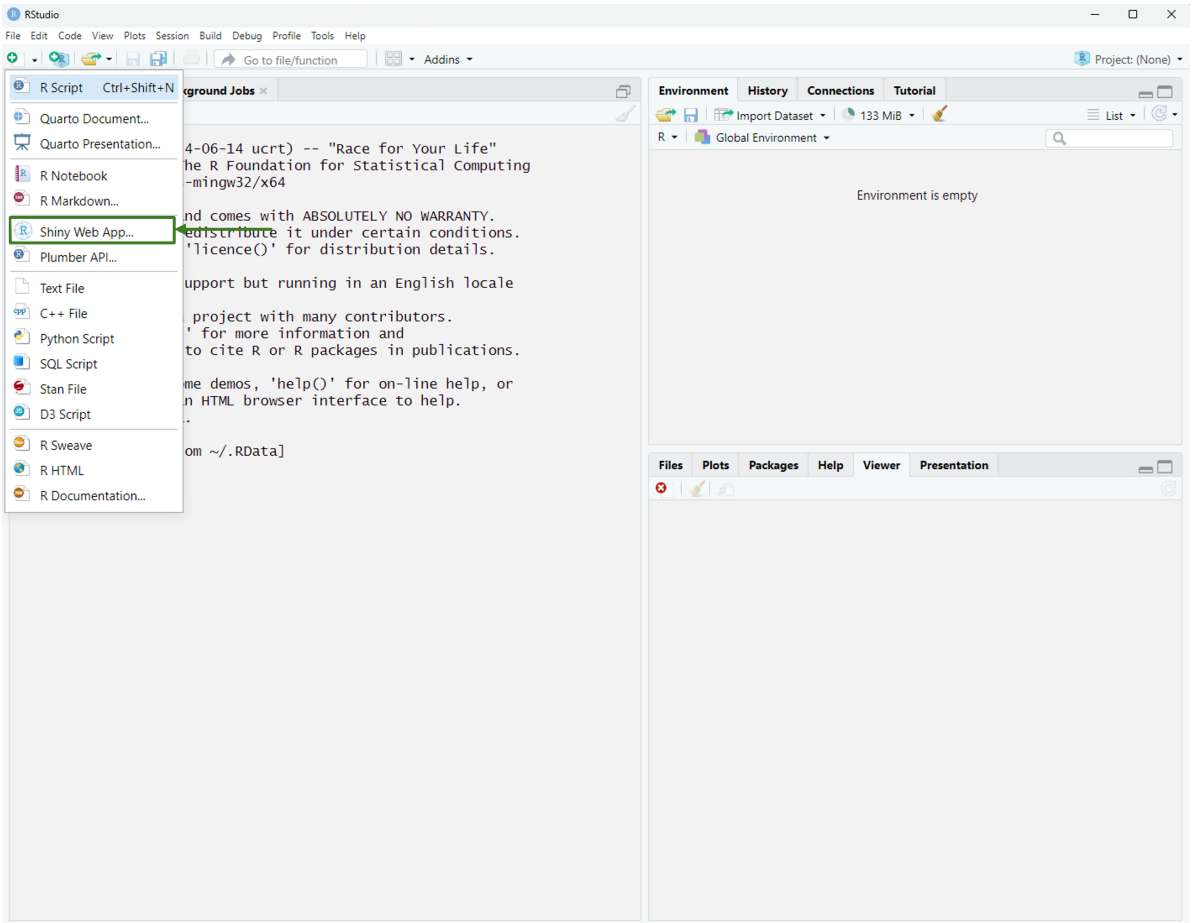
Shiny apps are contained in a single script called `app.R` and has three components:

- a user interface (`ui`) object which controls the layout and appearance of the app
- a server function which defines how the app behaves
- a call to the `shinyApp` function

In this supplement, we will walk through adapted portions of the Posit Shiny Basics.

Making your first Shiny app

Create a new shiny app in `RStudio` by clicking **New File** and selecting **Shiny Web App...** from the menu:



From there, you will be prompted about the name and location of the app:

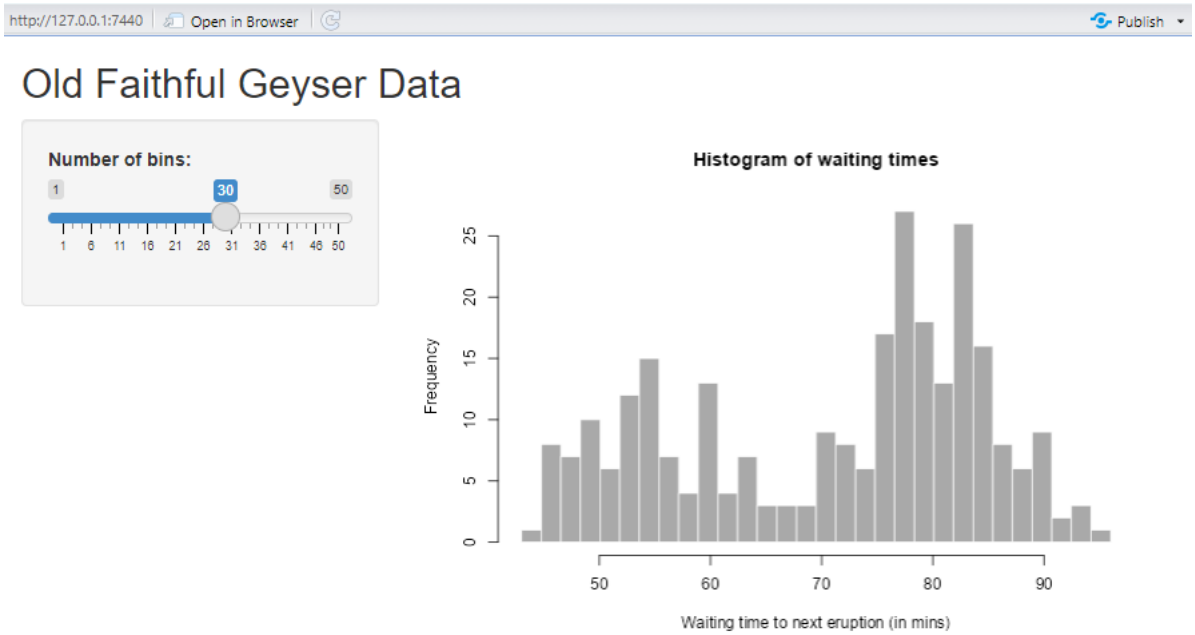
1) There are limitations on what we can name our app (i.e. no spaces, some special characters are off limits). Go ahead and name your application `my_first_app`.

2) We can choose whether we want there to be separate files for the user interface and server, which is useful if we are going to make an in-depth application. But for now, let's select **Single File (app.R)**.

3) We need to choose a location for our application. `R` will create a new folder within the selected directory that is the name of your application (from Step 1), within that folder it will create a single `app.R` file (or `ui.R` and `server.R` files if you selected **Multiple File** in Step 2).

Once you have created your app, you can press the **Run App** button (in the top right corner of the **Editor** pane).

You should get a pop-up that looks something like:



> **Tip**
>
> Before you close the app, notice that in the console it says something like `Listening on http://127.0.0.1:7440`. This is the URL where your app can be found.

> **Notice**
>
> The `R` prompt isn't visible. This is because Shiny apps block `R`, so we cannot run new code until the Shiny app stops. There are a few ways we can stop the app:
>
> - click the stop sign icon in the top of the **Console**
> - click on the **Console** and press **Esc**
> - close the Shiny app window

Let's start by changing the app that we currently have:

1. Set the minimum value of the slider bar to 5.
2. Replace the base `R` histogram with a `ggplot` histogram.

Solution

```r
library(shiny)
library(tidyverse)

# Define UI for application that draws a histogram
ui <- fluidPage(

    # Application title
    titlePanel("Old Faithful Geyser Data"),

    # Sidebar with a slider input for number of bins
    sidebarLayout(
        sidebarPanel(
          selectInput("var",
                      "Select a variable:",
                      choices = colnames(faithful)
          ),
          sliderInput("bins",
                      "Number of bins:",
                      min = 5,
                      max = 50,
                      value = 30
          )
        ),

        # Show a plot of the generated distribution
        mainPanel(
           plotOutput("histPlot")
        )
    )
)

# Define server logic required to draw a histogram
server <- function(input, output) {
    output$histPlot <- renderPlot({
      faithful %>%
        ggplot(aes(x = get(input$var))) +
        geom_histogram(bins = input$bins) +
        labs(x = input$var,
             title = paste0("Histogram of ", input$var)) +
        theme_bw()
    })
```

```
}

# Run the application
shinyApp(ui = ui, server = server)
```

Building `ui`

Replace all of the code in your `app.R` file with the following, then run your app:

```
library(shiny)
library(bslib)

# Define UI ----
ui <- page_sidebar(
  title = "title panel",
  sidebar = sidebar("sidebar"),
  "main contents"
)

# Define server logic ----
server <- function(input, output) {

}

# Run the app ----
shinyApp(ui = ui, server = server)
```

This creates a user interface with a title, sidebar panel, and main panel.

Cards

The `card()` function can be used to create a card in your Shiny app. Replace the `ui` in your `app.R` file with the following code and run your app:

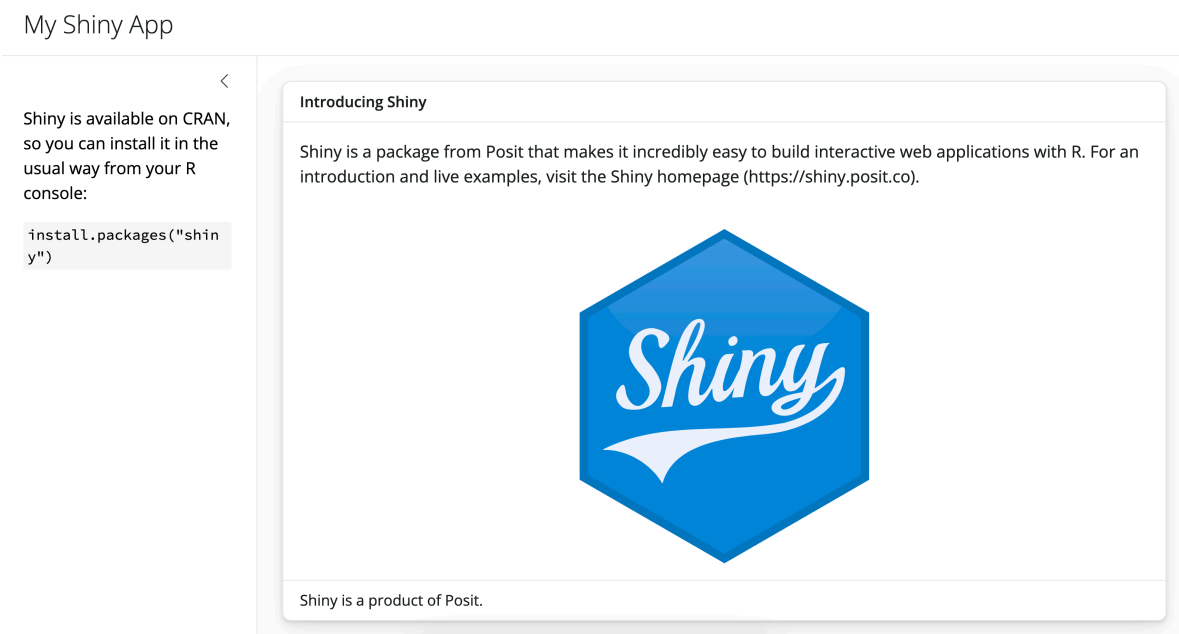
```
ui <- page_sidebar(
  title = "title panel",
  sidebar = sidebar("Sidebar"),
  card(
    "Card content"
  )
)
```

> **Tip**
>
> You can also use `card_header()`, `card_footer()`, and `card_image()` within the call to `card()` to add elements to a card.

Challenge

In your `app.R` file, recreate the following Shiny app to the best of your ability!

> **Tip**
>
> The logo is stored as "Shiny.png" in the .zip file, however, you should consider moving it to your Shiny app folder.



Solution

```
library(shiny)
library(bslib)

# Define UI for application that draws a histogram
ui <- page_sidebar(
  title = "My Shiny App",
  sidebar = sidebar("Shiny is available on CRAN, so you can install it in the
                    usual way from your R console:",
                    code('install.packages("shiny")')),
```

```r
  card(
    card_header("Introducing Shiny"),
    "Shiny is a package from Posit that makes it incredibly easy to build
    interactive web applications with R. For an introduction and live examples,
    visit the Shiny homepage (https://shiny.posit.co)",
    card_body(
      class = "align-items-center",
      card_image("Shiny.png",
                 width = "50%")
    ),
    card_footer("Shiny is a product of Posit.")

  )
)

# Define server logic required to draw a histogram
server <- function(input, output) {

}

# Run the application
shinyApp(ui = ui, server = server)
```

Widgets

function	widget
`actionButton`	Action Button
`checkboxGroupInput`	A group of check boxes
`checkboxInput`	A single check box
`dateInput`	A calendar to aid date selection
`dateRangeInput`	A pair of calendars for selecting a date range
`fileInput`	A file upload control wizard
`helpText`	Help text that can be added to an input form
`numericInput`	A field to enter numbers
`radioButtons`	A set of radio buttons
`selectInput`	A box with choices to select from
`sliderInput`	A slider bar
`submitButton`	A submit button
`textInput`	A field to enter text

Replace all of the code in your `app.R` file with the following, then run your app. Play with each widget to get a feel for what it does. Change values of the widget functions to observe the impact.

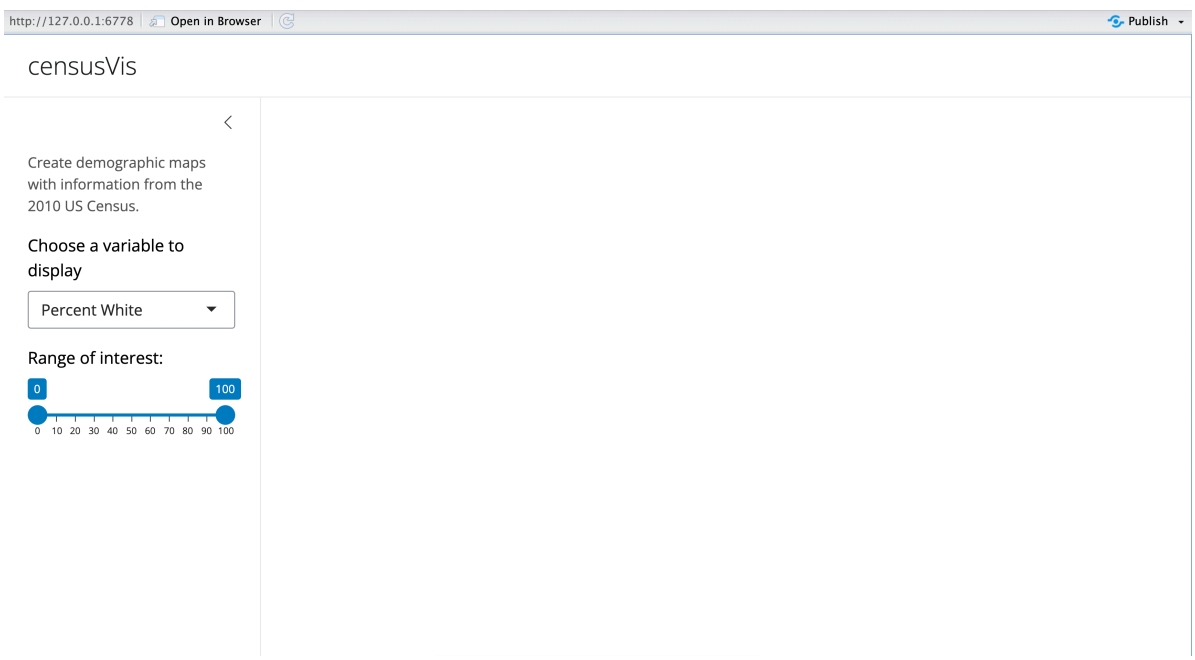
```r
library(shiny)
library(bslib)

# Define UI ----
ui <- page_fluid(
  titlePanel("Basic widgets"),
  layout_columns(
    col_width = 3,
    card(
      card_header("Buttons"),
      actionButton("action", "Action"),
      submitButton("Submit")
    ),
    card(
      card_header("Single checkbox"),
      checkboxInput("checkbox", "Choice A", value = TRUE)
    ),
    card(
      card_header("Checkbox group"),
      checkboxGroupInput(
        "checkGroup",
        "Select all that apply",
        choices = list("Choice 1" = 1, "Choice 2" = 2, "Choice 3" = 3),
        selected = 1
      )
    ),
    card(
      card_header("Date input"),
      dateInput("date", "Select date")
    ),
    card(
      card_header("Date range input"),
      dateRangeInput("dates", "Select dates")
    ),
    card(
      card_header("File input"),
      fileInput("file", label = NULL)
    ),
    card(
```

```
      card_header("Help text"),
      helpText(
        "Note: help text isn't a true widget,",
        "but it provides an easy way to add text to",
        "accompany other widgets."
      )
    ),
    card(
      card_header("Numeric input"),
      numericInput("num", "Input number", value = 1)
    ),
    card(
      card_header("Radio buttons"),
      radioButtons(
        "radio",
        "Select option",
        choices = list("Choice 1" = 1, "Choice 2" = 2, "Choice 3" = 3),
        selected = 1
      )
    ),
    card(
      card_header("Select box"),
      selectInput(
        "select",
        "Select option",
        choices = list("Choice 1" = 1, "Choice 2" = 2, "Choice 3" = 3),
        selected = 1
      )
    ),
    card(
      card_header("Sliders"),
      sliderInput(
        "slider1",
        "Set value",
        min = 0,
        max = 100,
        value = 50
      ),
      sliderInput(
        "slider2",
        "Set value range",
        min = 0,
```

```
        max = 100,
        value = c(25, 75)
      )
    ),
    card(
      card_header("Text input"),
      textInput("text", label = NULL, value = "Enter text...")
    )
  )
)

# Define server logic ----
server <- function(input, output) {

}

# Run the app ----
shinyApp(ui = ui, server = server)
```

Challenge

In your `app.R` file, recreate the following Shiny app to the best of your ability!



Solution

```r
library(shiny)
library(bslib)

# Define ui
ui <- page_sidebar(
  title = "censusVis",
  sidebar = sidebar(
    helpText("Create demographic maps with information from the 2010 US Census."),
    selectInput("var",
                "Choose a variable to display",
                choices = list("Percent White",
                               "Percent Black",
                               "Percent Asian",
                               "Percent Hispanic"),
                selected = 1),
    sliderInput("slider",
                "Range of interest:",
                min = 0,
                max = 100,
                value = c(0, 100))
  )
)

# Define server
server <- function(input, output){

}

# Run app
shinyApp(ui = ui, server = server)
```

Reactive output

Reactive output automatically responds when the user interacts with a widget!

Output function	Creates
`dataTableOutput`	DataTable
`htmlOutput`	raw HTML
`imageOutput`	image
`plotOutput`	plot
`tableOutput`	table
`textOutput`	text
`uiOutput`	raw HTML
`verbatimTextOutput`	text

Add `textOutput("selected_var")` to your `ui` from the last challenge.

> **Notice**
>
> `textOutput` takes an argument. Each of the `*Output` functions require a single argument: a character string that Shiny will use as the name of your reactive element. You will use this later!

Changing the `server` function

Placing an `*Output` function in `ui` tells Shiny where to display the object, but we need to tell Shiny how to build that object!

We can add `output$selected_var` to the `server` function, which matches the `selected_var` in the `ui`:

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

  output$selected_var <- renderText({
    "You have selected this"
  })

}
```

Each entry to `output` should contain one of Shiny's `render*` functions:

render function	Creates
`renderdataTable`	DataTable
`renderImage`	images
`renderPlot`	plots
`renderPrint`	any printed output
`renderTable`	data frame, matrix, other table
`renderText`	character strings
`renderUI`	a Shiny tag object or HTML

Using widget values

Now replace the `server` function with the following and run the app:

> **Tip**
>
> Make sure that your `inputID` (first argument of `selectInput`) is `"var"` or that you change `input$var` below to match your `inputID`.

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

  output$selected_var <- renderText({
    paste("You have selected", input$var)
  })

}
```

Change the selection in the drop-down menu and watch the text update based on your choice.

> **Tip**
>
> This is how to create reactive Shiny applications - connect values of `input` to objects in `output`. Shiny will track which outputs depend on widgets and take care of the rest!

Challenge:

Add a second line of reactive text to the main panel of your Shiny app. This line should display "You have chosen a range that goes from *something* to *something*," and each *something* should show the current minimum (min) or maximum (max) value of the slider widget.

Use R scripts and data

`counties.rds` is a data set that contains demographic data for each county in the United States collected as part of the 2010 Census. The data contains:

- the name of each county in the United States
- the total population of the county
- the percent of residents in the county who are White, Black, Hispanic, or Asian

> **Tip**
>
> The data is stored in the `counties.rds` file in today's zip folder. You should copy this file to the "census_map_app" folder!

`.rds` files are `R` data files, and they can be read into `R` using the following code:

```
counties <- readRDS("counties.rds")
```

It may be useful to write your own functions when creating a Shiny app. Instead of writing the functions in the `app.R` file, we should write them in another file that we can reference in our `app.R` file. Typically, we write the functions in **R Script** (`.R`) files which we can reference in our app using the following code:

```
source("helper.R")
```

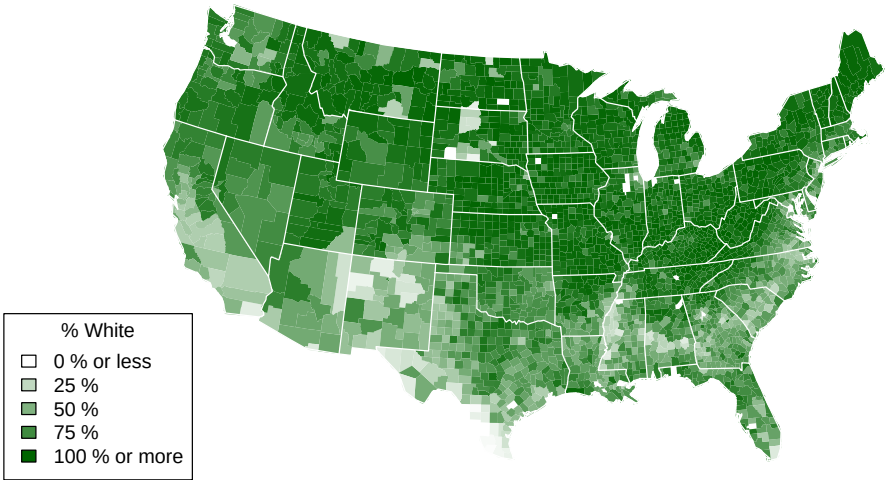
> **Tip**
>
> You should also copy the `helper.R` file in today's zip folder to the "census_map_app" folder!

This `helper.R` script contains a function called `percent_map`, that takes the following arguments:

- `var`
- `color`
- `legend.title`
- `min`
- `max`

We can make a map of the `Percent White` variable using the following code:

```
percent_map(counties$white, "darkgreen", "% White")
```



Challenge

Generalize the `percent_map` code to incorporate it into our Shiny app where the content, color, and legend titles change based on the variable selected from the drop-down menu.

To get you started, here is the code to `switch` the values for input as the first argument of the `percent_map` function:

```
data <- switch(input$var,
               "Percent White" = counties$white,
               "Percent Black" = counties$black,
               "Percent Hispanic" = counties$hispanic,
               "Percent Asian" = counties$asian)
```

> **Tip**
>
> This code should go inside the `renderPlot` function in the `server` code!

```r
library(shiny)
library(bslib)
library(tidyverse)

counties <- readRDS("counties.rds")

source("helper.R")

# Define ui
ui <- page_sidebar(
  title = "censusVis",
  sidebar = sidebar(
    helpText("Create demographic maps with information from the 2010 US Census."),
    selectInput("var",
                "Choose a variable to display",
                choices = list("Percent White",
                               "Percent Black",
                               "Percent Asian",
                               "Percent Hispanic"),
                selected = 1),
    sliderInput("slider",
                "Range of interest:",
                min = 0,
                max = 100,
                value = c(0, 100))
  ),
  # main panel
  plotOutput("map")
)

# Define server
server <- function(input, output) {

  output$map <- renderPlot({
    data <- switch(input$var,
                   "Percent White" = counties$white,
                   "Percent Black" = counties$black,
                   "Percent Hispanic" = counties$hispanic,
                   "Percent Asian" = counties$asian)

    color <- switch(input$var,
                    "Percent White" = "darkgreen",
```

```
                    "Percent Black" = "dodgerblue3",
                    "Percent Hispanic" = "darkturquoise",
                    "Percent Asian" = "mediumpurple2")

    title <- switch(input$var,
                    "Percent White" = "% White",
                    "Percent Black" = "% Black",
                    "Percent Hispanic" = "% Hispanic",
                    "Percent Asian" = "% Asian")

    percent_map(data, color, title,
                min = input$slider[1], max = input$slider[2])
  })

}

# Run app
shinyApp(ui = ui, server = server)
```

Sharing your Shiny app:

1. **Share your Shiny app as `R` scripts.** Requires all of the people that you are sharing with to have `R` and Shiny installed on their computer.
2. **Share your Shiny app as a web page.** Allows you to share with anyone!

Sharing as a web page using shinyapps.io

You can go to shinyapps.io to create an account. Then, you can publish your Shiny app on the web!

Check out the Shiny app I made to help some collaborators visualize estimated models in three dimensions (here).