Week-8: Visualizing data using `Shiny`

NM2207: Computational Media Literacy

Narayani Vedam, Ph.D.

Department of Communications and New Media

NUS National University of Singapore | Faculty of Arts & Social Sciences

This week

Table of Contents

- What is `Shiny` ?
- How can mastering `Shiny` benefit you?
- All else you need to know about `Shiny`!

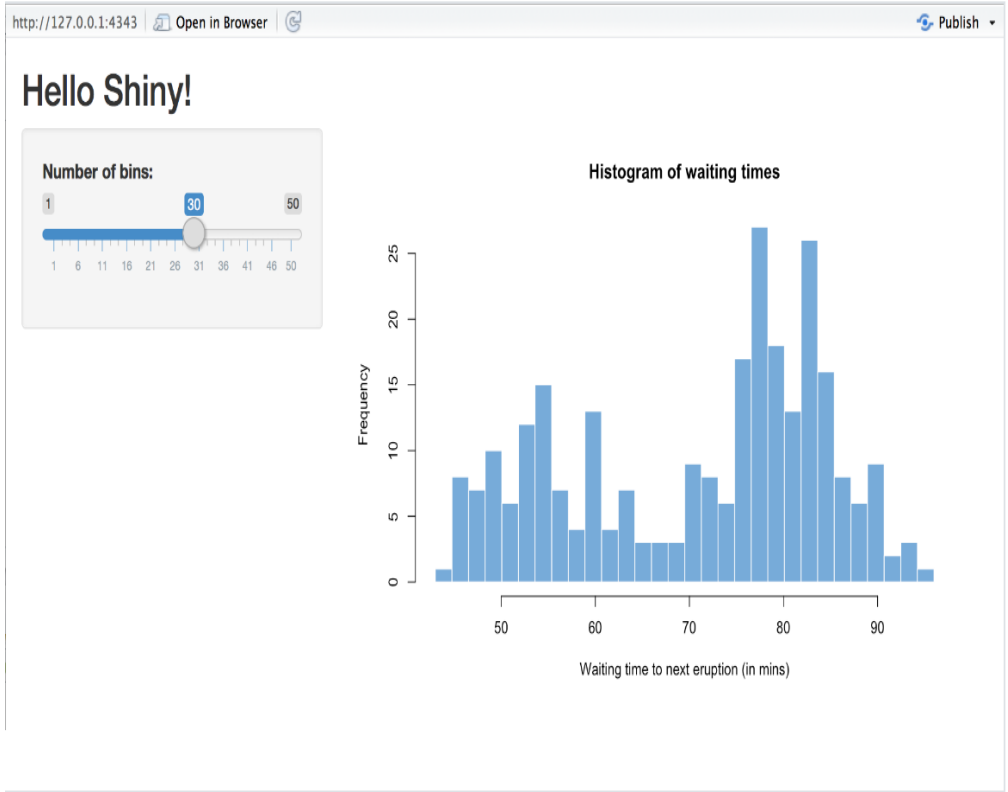


Figure: Example Shiny app

What is `Shiny`?

- `Shiny` is an R package
- It allows you to create rich and interactive web applications from R
- In the past, creating web apps was hard for most R users because,
  - You need a deep knowledge of web technologies like HTML, CSS and JavaScript
  - Making complex interactive apps requires careful analysis of interaction flows to make sure that when an input changes, only the related outputs are updated.
- `Shiny` makes it significantly easier for the R programmer to create web apps by:
  - Providing a carefully curated set of user interface (UI for short) functions that generate the HTML, CSS, and JavaScript needed for common tasks. You now dont have to know HTML, CSS or JavaScript!
  - Introducing a new style of programming called ***reactive programming*** which automatically tracks the dependencies of pieces of code

How does mastering `Shiny` benefit us?

`Shiny` empowers you with some superpowers, and lets you

- create dashboards that track high-level performance indicators
- convert huge pdfs into apps, where the user can jump to the results they care about
- communicate complex models to non-technical audience through visualizations and interactive interface
- create interactive demos to demonstrate various concepts

Example

- The package has eleven built-in examples to demonstrate how `Shiny` works
- To explore the first example, run the code
- You could try to list all the examples using `runExample()`
- If you haven't been using `?` operator before every function you use, start doing so
- It reveals a lot of information about the function
- If you tried the code, move on to the next slide for the output

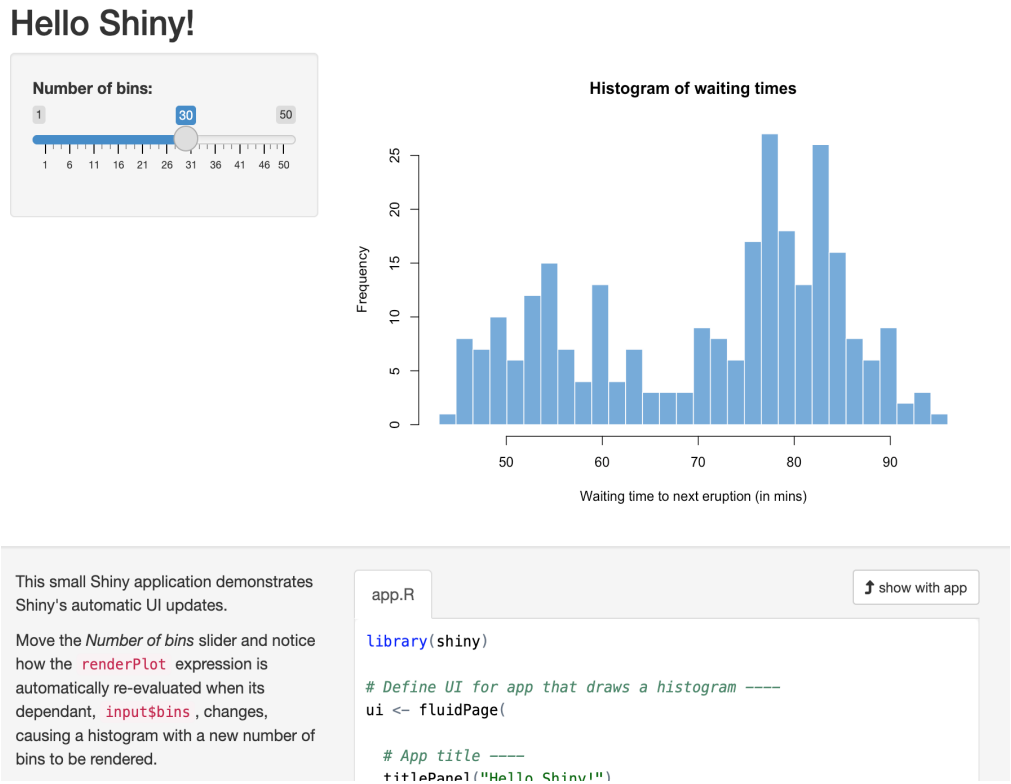
```
# Install package
install.packages("shiny")

# Invoke the package
library(shiny)

# Run an example from the library
runExample("01_hello")
```

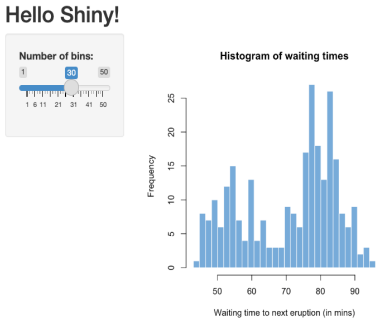
Example

`Shiny` dashboard with example - "01_hello"



Structure of a `Shiny` app

- `Shiny` apps are contained in a single script called "app.R"
- The script resides in a directory (say, "newdir/")
- You can run the app with `runApp("newdir")`
- "app.R" has three components
  - a user interface object
  - a server function
  - a call to `shinyApp` function



app.R

show below

```
library(shiny)

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

  # App title ----
  titlePanel("Hello Shiny!"),

  # Sidebar layout with input and output definitions ----
  sidebarLayout(

    # Sidebar panel for inputs ----
    sidebarPanel(

      # Input: Slider for the number of bins ----
      sliderInput(inputId = "bins",
                  label = "Number of bins:",
                  min = 1,
                  max = 50,
                  value = 30)

    ),

    # Main panel for displaying outputs ----
    mainPanel(

      # Output: Histogram ----
      plotOutput(outputId = "distPlot")

    )
  )
```

Example app: user interface

The user interface (`ui`) object controls the layout and appearance of the app.

```
library(shiny)
# Define UI for app that draws a histogram ----
ui <- fluidPage(
  # App title ----
  titlePanel("Hello Shiny!"),
  # Sidebar layout with input and output definitions ----
  sidebarLayout(
    # Sidebar panel for inputs ----
    sidebarPanel(
      # Input: Slider for the number of bins ----
      sliderInput(inputId = "bins",
                  label = "Number of bins:",
                  min = 1,
                  max = 50,
                  value = 30)),
    # Main panel for displaying outputs ----
    mainPanel(
      # Output: Histogram ----
      plotOutput(outputId = "distPlot"))))
```

Example app: server function

The `server` function contains instructions for the computer to build the app.

```
# Define server logic required to draw a histogram ----
server <- function(input, output) {
  # Histogram of the Old Faithful Geyser Data ----
  # with requested number of bins
  # This expression that generates a histogram is wrapped in a call
  # to renderPlot to indicate that:
  #
  # 1. It is "reactive" and therefore should be automatically
  #    re-executed when inputs (input$bins) change
  # 2. Its output type is a plot
  output$distPlot <- renderPlot({
    x    <- faithful$waiting
    bins <- seq(min(x), max(x), length.out = input$bins + 1)
    hist(x, breaks = bins, col = "#007bc2", border = "white",
         xlab = "Waiting time to next eruption (in mins)",
         main = "Histogram of waiting times")
    })
}
```

Let us begin!

- Create a new directory in your working directory, say, "App-1"
  - Go to the "Files/Plots/Packages/Help/Viewer/Presentation" pane (typically in the lower right corner of your R Studio layout)
  - Click on "Files" tab
  - Click on the first icon to create a new folder
  - It will now prompt you to enter the name of the folder
  - Enter "App-1" (minus quotation marks)
- Let us create "app.R"
  - Follow on in the next slide

Let us begin!

- Create "app.R" file
  - Open the "01_hello" app
    - If you haven't closed the previous app instance, toggle the windows (use ctrl/command +`)
    - If not, follow the instructions (here) to relaunch the app
    - Copy the code in "app.R" (Use ctrl/command+C keys)
  - Create a new "R script" from RStudio toolbar
    - Paste the copied code (Use ctrl/command+V keys)
    - Save it as "app.R" in the "App-1" directory
- ***Congratulations!*** go ahead and launch your app!

```
library(shiny)
runApp("App-1")
```

App-1: customization

1. Open the file "app.R"
    - Change the title from "Hello Shiny!" to "Hello World!"
        - look for the function `titlePanel()`
    - Set the minimum value of the slider bar to 5
        - look for the function `sliderInput(...,min=5,...)`
    - Change the border color of histogram to yellow
        - look for `renderPlot(...,hist(...,border="yellow",...),...)`
2. Relaunch the app and see if the changes are reflected
3. Unlike the example, the "app.R" is not displayed when you run the app
    - To display "app.R" when you run the app, use `display.mode()`

```
runApp("App-1", display.mode = "showcase")
```

Quick recap

- To create your own Shiny app:
  - Make a directory named "myapp/" for your app.
  - Save your "app.R" script inside that directory.
  - Launch the app with `runApp` or RStudio's keyboard shortcuts.
  - Exit the Shiny app by clicking escape
- You can create an app by modifying one of the examples in the `Shiny` library

```
runExample("01_hello")      # a histogram
runExample("02_text")       # tables and data
runExample("03_reactivity") # a reactive expr
runExample("04_mpg")        # global variable
runExample("05_sliders")    # slider bars
runExample("06_tabsets")    # tabbed panels
runExample("07_widgets")    # help text and s
runExample("08_html")       # Shiny app built
runExample("09_upload")     # file upload wiz
runExample("10_download")   # file download w
runExample("11_timer")      # an automated ti
```

Building an app from scratch

- Let us use the same folder ("App-1") and script ("app.R")
- Edit the "app.R" by pasting the skeletal code below
- ***Notice***: three components of the app, ui, server function and call to run the app
- Run the code to see a blank app

```
library(shiny)

# Define UI ----
ui <- fluidPage(

)

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

}

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

User interface

- `Shiny` uses `fluidPage()` to create a layout of the app
  - Observe how the layout fits the dimensions of your screen
  - `titlePanel()` and `sidebarLayout()` are the two most popular elements to add to `fluidPage()`
  - `sidebarLayout()` always takes two arguments:
    - `sidebarPanel()`
    - `mainPanel()`

```
ui <- fluidPage(
                titlePanel("Insert your title here"),

                sidebarLayout(
                              position="left"
                              sidebarPanel("sidebar panel"),
                              mainPanel("main panel")
                             )
              )
```

User interface: output

- Launch the app by running `runApp("App-1")`
- The output should resemble the snapshot below

User interface: headers

To create a header element

- select one of the functions in the Table below
- pass `h#("Your heading")` as an argument to `titlePanel`, `sidebarPanel` or `mainPanel`
- multiple elements can be separated by a comma
- the text(s) will appear in the corresponding panel of the app

Shiny function	Header level
h1	1st
h2	2nd
h3	3rd
h4	4th
h5	5th
h6	6th

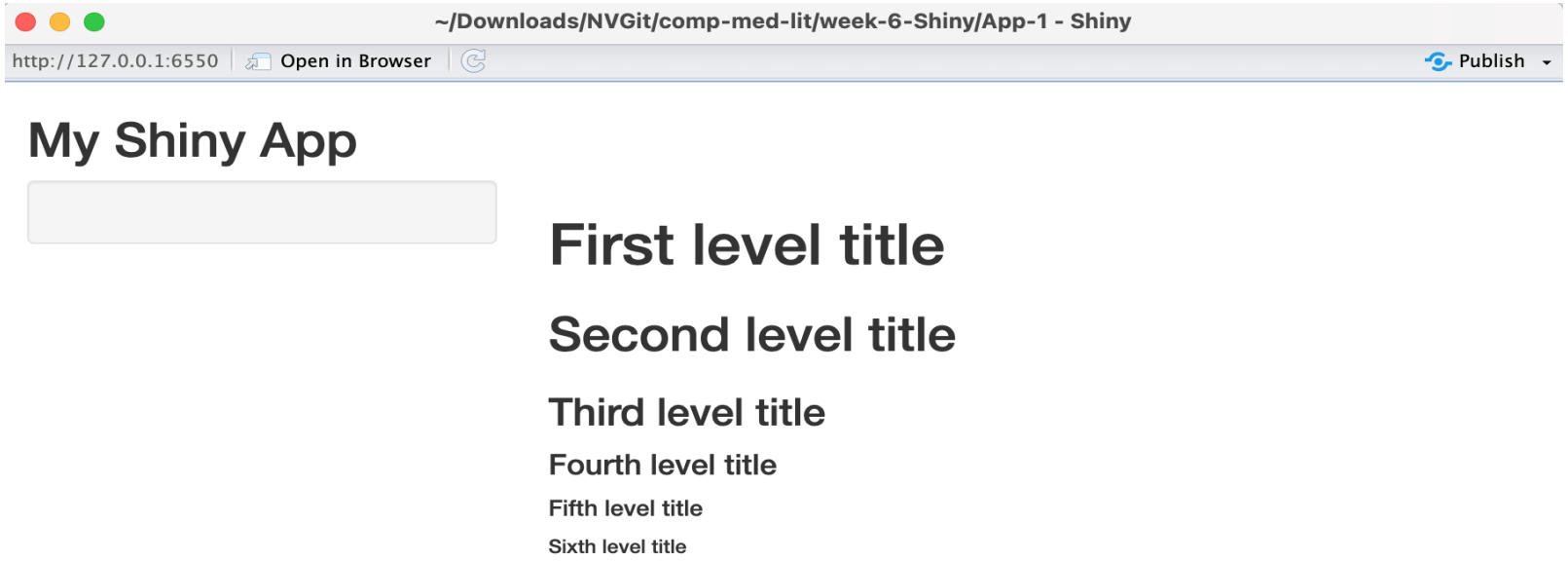
User interface: headers

- Paste the following code in "app.R"

```
ui <- fluidPage(
  titlePanel("My Shiny App"),
  sidebarLayout(
    sidebarPanel(),
    mainPanel(
      h1("First level title"),
      h2("Second level title"),
      h3("Third level title"),
      h4("Fourth level title"),
      h5("Fifth level title"),
      h6("Sixth level title")
    )
  )
)
```

- Run the app, `runApp("App-1")` or from RStudio toolbar

User interface: headers



User interface: formatted text

The functions listed in the Table below can be used to format text

Shiny function	Output
p	paragraph
a	hyperlink
br	line break
div	division of text with a uniform style
span	in-line division of text with a uniform style
code	formatted code block
img	an image
strong	bold text
em	italics

User interface: formatted text

- Paste the following code in "app.R"

```
ui <- fluidPage(
  titlePanel("My Shiny App"),
  sidebarLayout(
    sidebarPanel(),
    mainPanel(
      p("p creates a paragraph of text."),
      p("A new p() command starts a new paragraph. Supply a style attribute to change the format o
      strong("strong() makes bold text."),
      em("em() creates italicized (i.e, emphasized) text."),
      br(),
      code("code displays your text similar to computer code"),
      div("div creates segments of text with a similar style. This division of text is all blue be
      br(),
      p("span does the same thing as div, but it works with",
        span("groups of words", style = "color:blue"),
        "that appear inside a paragraph.")
      [Shiny tutorial](a(https://shiny.posit.co/r/getstarted/shiny-basics/lesson1/))
    )
  )
)
```

- Run the app, `runApp("App-1")` or from RStudio toolbar

User interface: images

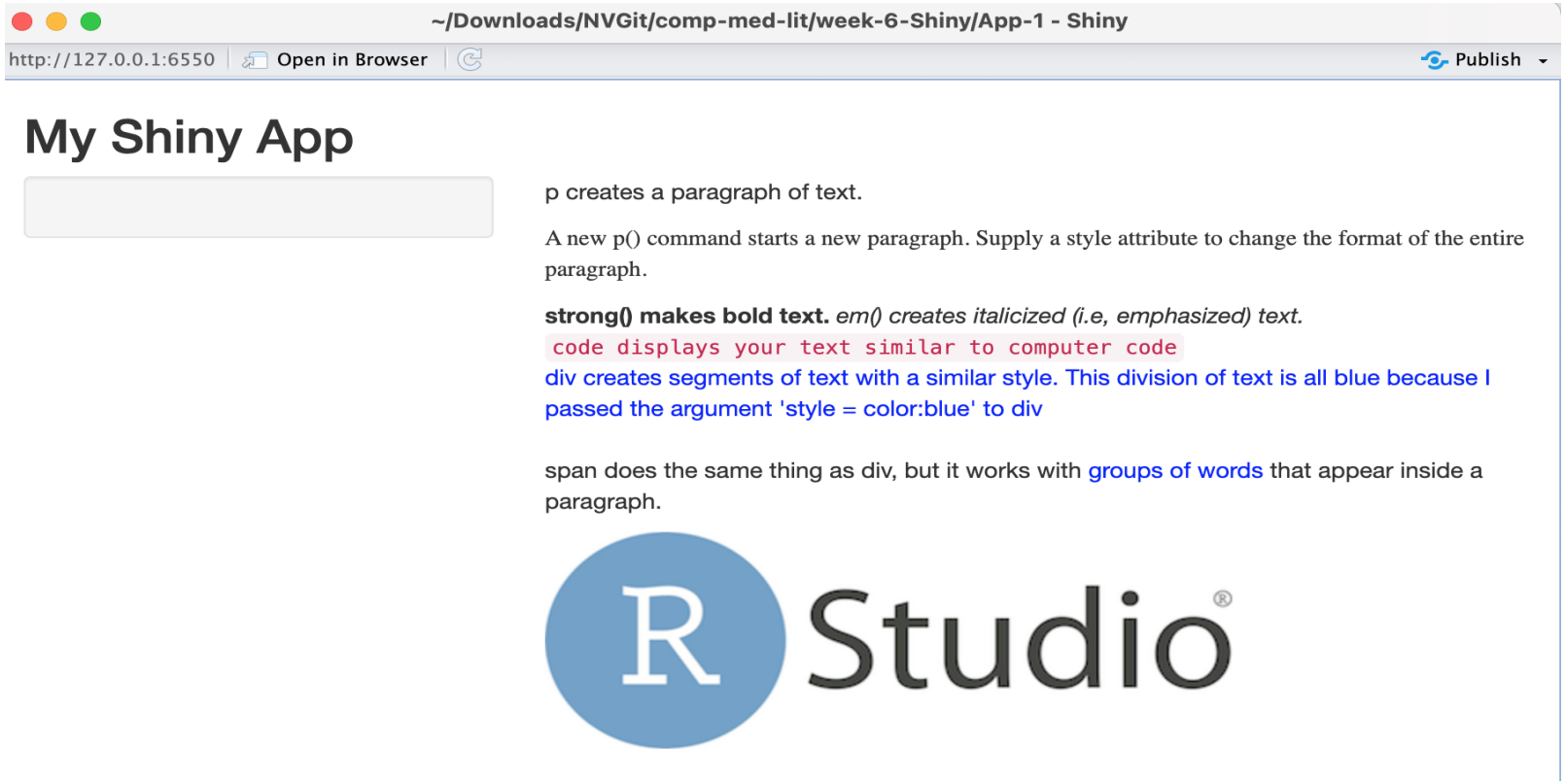
To insert an image, use the `img` function

- Pass the image file as the `src` argument to the `img` function
- The function looks for the image in a specific place
    - Inside the "App-1" directory create a new folder
    - Name the folder "www" (***always***!!)
    - Insert the below code either in `titlePanel`, `sidePanel` or `mainPanel`

```
img(src = "rstudio.png", height = 140, width = 400)
```

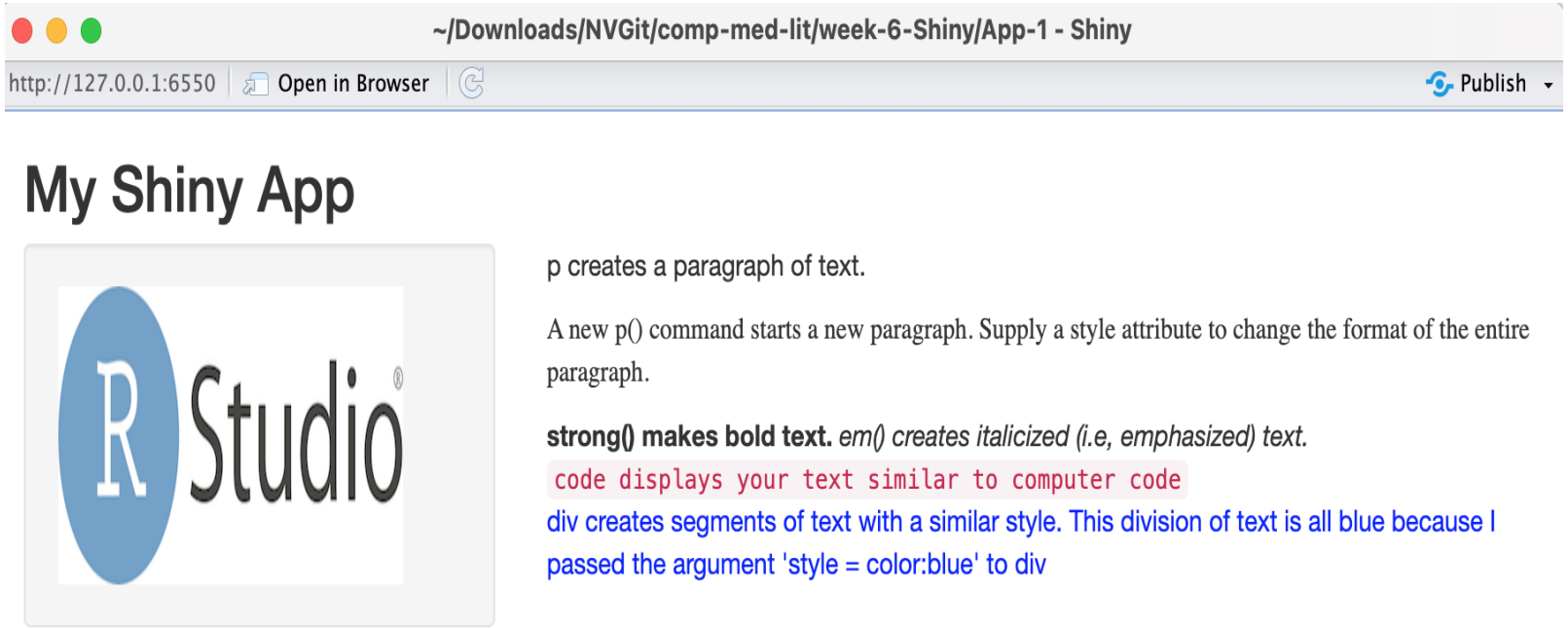
User interface: images

- Insert it in the `mainPanel` and launch the app



User interface: images

- Insert it in the `sidePanel` and relaunch the app
- Adjust the dimensions for the image to fit the side bar



Thanks!

Slides created via the R packages:

xaringan
gadenbuie/xaringanthemer.