Getting Started with R Shiny: Build and Deploy Interactive Web Apps

Coffee, Cookie and Coding (C^3) Workshop supported by the Public Health Data Science and Data Equity team

Howard Baik, M.S.

Feb 17th, 2025

Yale SCHOOL OF PUBLIC HEALTH
Data Science and Data Equity

Howard Baik, M.S.

- Worked 1.5 years as a Software Development Engineer on R packages and Shiny applications.
- Received a Masters in Biostatistics from the University of Washington in 2023.

Today's Learning Objectives

Yale SCHOOL OF PUBLIC HEALTH
Data Science and Data Equity

01 Understand the basic structure of a Shiny app, including UI (User Interface) and server components.

02 Learn to use reactive elements to make your app dynamic and interactive.

03 Create a simple interactive application using a data example.

04 Learn how to deploy Shiny apps using a self-service platform like shinyapps.io

Our Choice Resources

- Mastering Shiny: https://mastering-shiny.org/
- Shiny Basics: https://shiny.posit.co/r/getstarted/shiny-basics/lesson1/
- Shiny Gallery: https://shiny.posit.co/r/gallery/

Introduction to Shiny in R

Yale SCHOOL OF PUBLIC HEALTH
Data Science and Data Equity

Shiny for R

Shiny is an R package that makes it easy to build interactive web apps straight from R. You can also extend your Shiny apps with CSS, HTML, and JavaScript.

Example of a Shiny App

SAMBA-EHR

SAMBA-EHR: Sampling And Misclassification Bias Analysis in Genome and Phenome-wide Association Studies using Electronic Health Records

This tool allows researchers to explore the impact of disease-related sampling and phenotype misclassification on the bias in association estimates from GWAS/PheWAS studies based on large-scale observational databases such as electronic health records.
Click here for example vignette

Model Specification | Bias of Simple Analysis | Sensitivity Analysis for "True" Effect of G

Plotted Quantity:

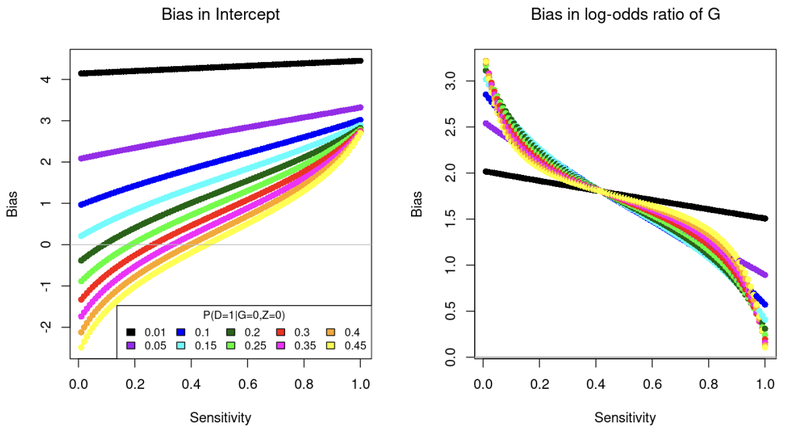
Bias of Simple Analysis Estimate

True Log-Odds Ratio: -1.8 (-3 to 3)

Sampling Ratio: 8.1 (0 to 10)

1-Specificity: 0.42 (0 to 0.5)

Genetic Information:

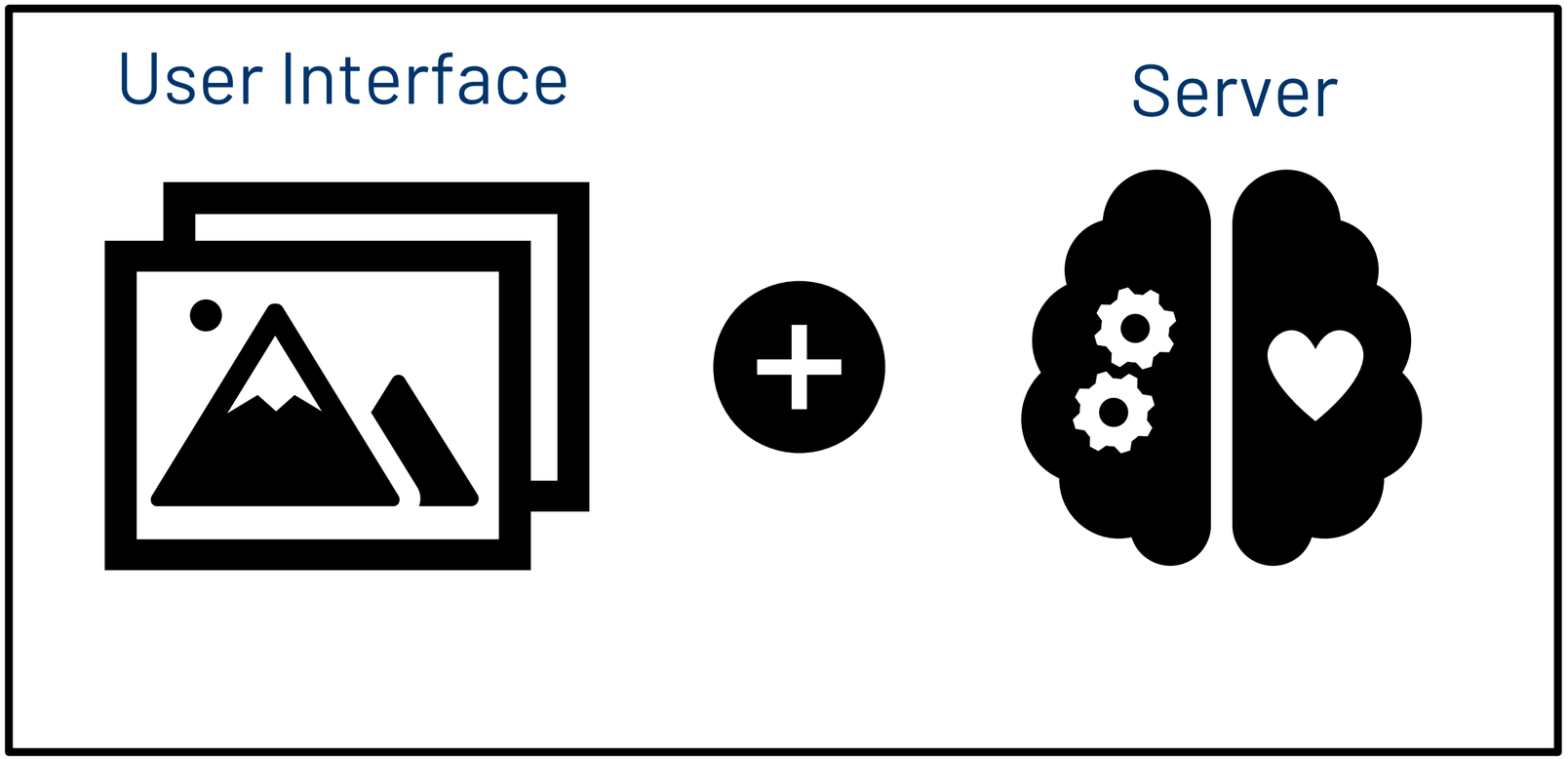


These plots provide expressions for the bias in estimating the effect of G and the intercept when we fit a simple logistic regression model for D*|G,Z,S=1 in our sampled dataset. We assume that the data follow the proposed model.

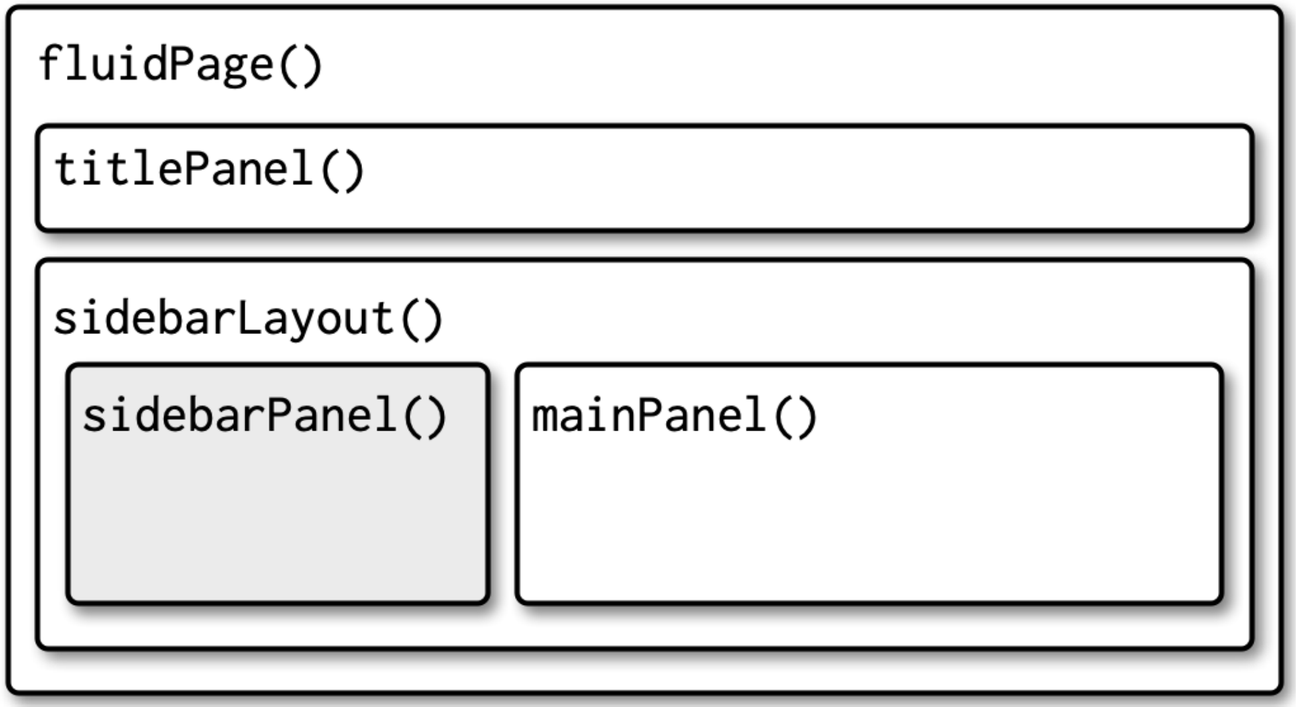
NOTE: If help information does not appear when you hover over the question marks, try reloading the page.

Developed by Dr. Lauren J Beesley, Department of Biostatistics, University of Michigan. Contact: lbeesley@umich.edu

Structure of a Shiny App

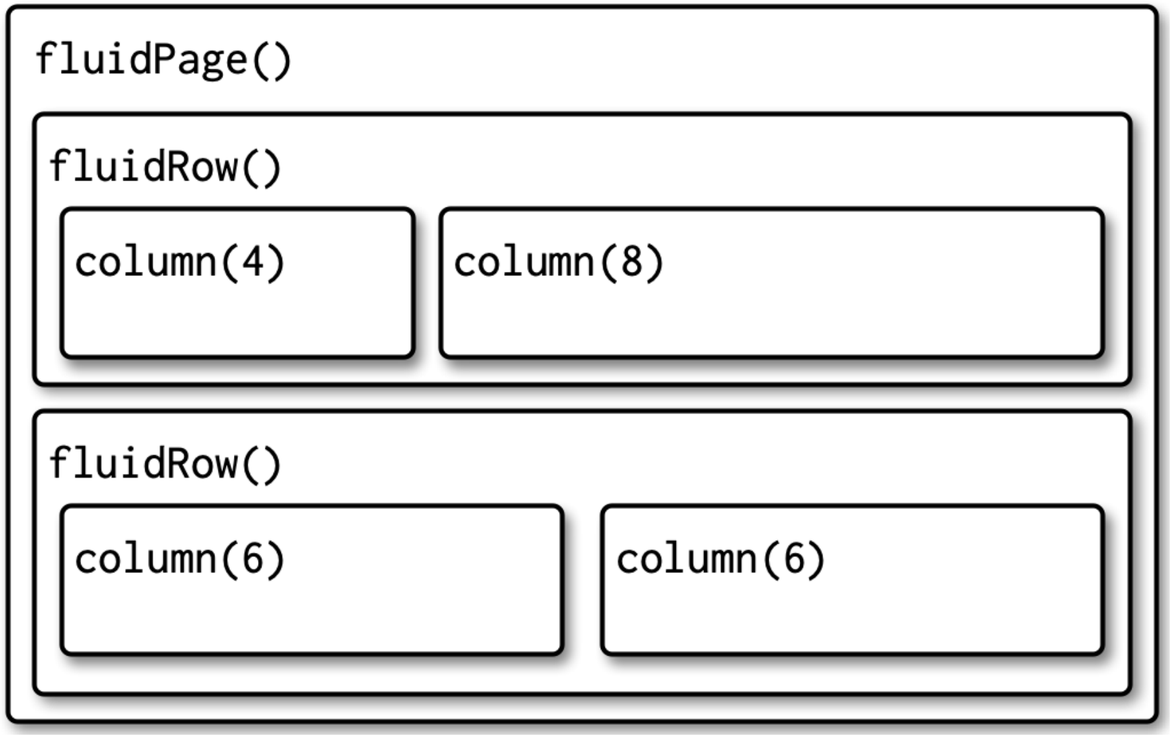


Sidebar Layout




```
fluidPage(
  titlePanel(),
  sidebarLayout(
    sidebarPanel(),
    mainPanel()
  )
)
```

Multi-row Layout




```
fluidPage(
  fluidRow(
    column(4, ),
    column(8, )
  ),
  fluidRow(
    column(6, ),
    column(6, )
  )
)
```

Other Layouts

- Tabsets: Use tabsetPanel()
- Navbars and navlists: Use navlistPanel() and navbarPage()
- Dashboards: shinydashboard and Quarto Dashboards

UI functions are just HTML

Shiny's interface is just a web page, and this can be seen by the fact that UI functions are just R functions that generate HTML.

We can run any of the following in our console and see the HTML output generated:

- `fluidPage()`
- `sidebarLayout(sidebarPanel(),mainPanel())`

Shiny Inputs

- Functions like `selectInput()`, `textInput()`, and `sliderInput()` insert inputs controls into your UI.
- All input functions have the same first argument: **inputID**.
- **inputID** is the identifier used to connect the front end with the back end. If UI has input with **inputID**, "name", the server will access it with **input$name.**

Shiny Inputs

Examples:

```
selectInput("state",
            "What's your favorite state?", state.name)
numericInput("num", "Number one",
             value = 0, min = 0, max = 100)
textInput("name", "What's your name?")
```

Shiny Inputs

Gallery with Shiny Input Widgets:

https://shiny.posit.co/r/gallery/widgets/widget-gallery/

Shiny Outputs

Each **output** function on the front end is coupled with a **render** function in the back end.

Three main types of output, corresponding to the three things you usually include in a report:

1. Text
2. Tables
3. Plots.

Shiny Outputs: Text

Regular text with **textOutput()** and **renderText()**

```
ui <- fluidPage(
  textOutput("text")
)
server <- function(input, output, session) {
  output$text <- renderText({
    "Hello friend!"
  })
}
```

Shiny Outputs: Tables

tableOutput() and **renderTable()** render a static table of data, showing all the data at once.

```
ui <- fluidPage(tableOutput("static"))

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

    output$static <- renderTable(head(mtcars))

}
```

Shiny Outputs: Plots

You can display any type of R graphic (base, ggplot2, or otherwise) with **plotOutput()** and **renderPlot()**:

```
ui <- fluidPage(
 plotOutput("plot”)
)
server <- function(input, output, session) {
 output$plot <- renderPlot(plot(1:5))
}
```

Reactivity

Introduction to Reactivity

Reactive programming is what connects your inputs to your outputs.

Inputs and outputs are defined by the elements in our UI.

Reactivity

UI Function

```
selectInput(“state”,  “Choose a State”,

              choices = c(“Washington”,
                          “Pennsylvania”,
                          “Connecticut”)

             )

plotOutput("plot”)
```

Reactivity

Server Function

```
output$plot <-  renderPlot({

    df  |>

    filter(Province_State == input$state)|>

    ggplot(aes(x = Date,

               y = Percentage_at_least_one_dose)) +

    geom_line()

})
```

Reactivity

UI Function

```
selectInput(“state”,
            “Choose a State”,
          choices = c(“Washington”,
                      “Pennsylvania”,
                       “Connecticut”)
)
plotOutput("plot”)
```

Server Function

```
output$plot <-  renderPlot({
     df  |>
     filter(Province_State == input$state)|>
     ggplot(aes(x = Date,
                      y = Percentage_at_least_one_dose)
            ) +
     geom_line()
})
```

Worked Through Example:

https://github.com/ysph-dsde/Getting-Started-with-R-Shiny-Workshop

Dataset Information

The dataset contains vaccination data of U.S. States.

Data Dictionary:

- **Date**: Date collection date
- **Province_State**: The name of the state or province.
- **People_at_least_one_dose**: Cumulative number of people who received at least one vaccine dose. When the person receives a prescribed second dose it is not counted twice.
- **People_fully_vaccinated**: Cumulative number of people who received a complete primary series. This means having received one dose of a single-dose vaccine or two doses on different days (regardless of time interval) of either a mRNA or a protein-based series.

Deploy Shiny apps using shinyapps.io

1. Go to shinyapps.io
2. Sign up or log in using email and password
3. Once logged in, navigate to **Account > Tokens**.
4. Click **Add Token > Copy to clipboard.**
5. Install the rsconnect R package (`install.packages("rsconnect")`)
6. In your R console, authenticate your shinyapps.io account by pasting the token details you copied:

```
rsconnect::setAccountInfo(name = 'your_account_name',
                          token = 'your_token',
                          secret = 'your_secret')
```

7. Deploy the app: `rsconnect::deployApp()`

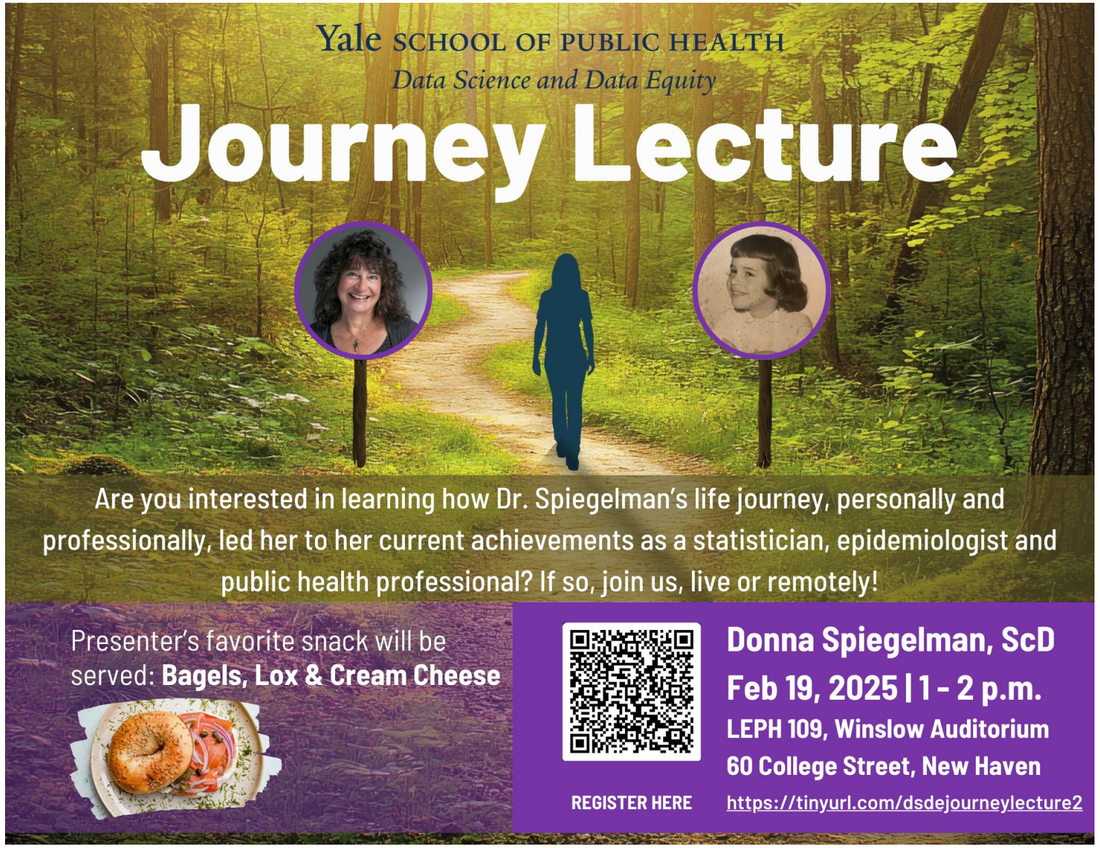
Appendix

References

- GLEON 2021 Workshop – R shiny apps for beginners: https://youtu.be/U3w9ftlyh_E?si=BQ-FJJXzfcCtGle3
- Workshop at Posit Conference 2024 – Introduction to Shiny for R: https://github.com/posit-conf-2024/shiny-r-intro
- Mastering Shiny: https://mastering-shiny.org/index.html

Evaluation

Next DSDE Event



sph.yale.edu/dsde

@YaleSPH

Public Health Data Science and Data Equity
Yale School of Public Health
60 College Street, New Haven, CT 06510

Yale SCHOOL OF PUBLIC HEALTH