CSSS 569 Visualizing Data and Models

Lab 8: Interactive Visual Display with R + Shiny

Ramses Llobet

Department of Political Science, UW

March 4, 2026

Basic structure of a Shiny app

- Four lines to build a Shiny app

```r
library(shiny)
ui <- fluidPage(...)
server <- function(input, output) {...}
shinyApp(ui = ui, server = server)
```

Basic structure of a Shiny app

- Four lines to build a `Shiny` app

```
library(shiny)
ui <- fluidPage(...)
server <- function(input, output) {...}
shinyApp(ui = ui, server = server)
```

1. `ui`: front end interface

Basic structure of a Shiny app

- Four lines to build a `Shiny` app

```
library(shiny)
ui <- fluidPage(...)
server <- function(input, output) {...}
shinyApp(ui = ui, server = server)
```

1. `ui`: front end interface
   - Inside `fluidPage()`

Basic structure of a Shiny app

- Four lines to build a `Shiny` app

```r
library(shiny)
ui <- fluidPage(...)
server <- function(input, output) {...}
shinyApp(ui = ui, server = server)
```

1. `ui`: front end interface
   - Inside `fluidPage()`
   - `Input` and `Output` functions

Basic structure of a Shiny app

- Four lines to build a Shiny app

```
library(shiny)
ui <- fluidPage(...)
server <- function(input, output) {...}
shinyApp(ui = ui, server = server)
```

1. ui: front end interface
   - Inside fluidPage()
   - Input and Output functions
   - Others: Layout functions

Basic structure of a Shiny app

- Four lines to build a Shiny app

```r
library(shiny)
ui <- fluidPage(...)
server <- function(input, output) {...}
shinyApp(ui = ui, server = server)
```

1. ui: front end interface
   - Inside fluidPage()
   - Input and Output functions
   - Others: Layout functions
2. server function: back end logic

Basic structure of a Shiny app

- Four lines to build a Shiny app

```r
library(shiny)
ui <- fluidPage(...)
server <- function(input, output) {...}
shinyApp(ui = ui, server = server)
```

1. ui: front end interface
   - Inside fluidPage()
   - Input and Output functions
   - Others: Layout functions
2. server function: back end logic
   - Access input values via input$... **in a reactive context**

Basic structure of a Shiny app

- Four lines to build a `Shiny` app

```r
library(shiny)
ui <- fluidPage(...)
server <- function(input, output) {...}
shinyApp(ui = ui, server = server)
```

1. `ui`: front end interface
   - Inside `fluidPage()`
   - `Input` and `Output` functions
   - Others: `Layout` functions
2. `server` function: back end logic
   - Access input values via `input$...` **in a reactive context**
   - Create output values via `render()` or `reactive()` **in a reactive context**

Basic structure of a Shiny app

- Four lines to build a `Shiny` app

```r
library(shiny)
ui <- fluidPage(...)
server <- function(input, output) {...}
shinyApp(ui = ui, server = server)
```

1. `ui`: front end interface
   - Inside `fluidPage()`
   - `Input` and `Output` functions
   - Others: `Layout` functions
2. `server` function: back end logic
   - Access input values via `input$...` **in a reactive context**
   - Create output values via `render()` or `reactive()` **in a reactive context**
     - Within `render()` or `reactive()`, write code to perform some tasks

Basic structure of a Shiny app

- Four lines to build a `Shiny` app

```r
library(shiny)
ui <- fluidPage(...)
server <- function(input, output) {...}
shinyApp(ui = ui, server = server)
```

1. `ui`: front end interface
   - Inside `fluidPage()`
   - `Input` and `Output` functions
   - Others: `Layout` functions
2. `server` function: back end logic
   - Access input values via `input$...` **in a reactive context**
   - Create output values via `render()` or `reactive()` **in a reactive context**
     - Within `render()` or `reactive()`, write code to perform some tasks
     - Store them as elements of `output` via `output$...`

What is reactivity?

What is reactivity?

- Reactivity: connecting inputs to outputs

What is reactivity?

- Reactivity: connecting inputs to outputs
  - Allow outputs to automatically update when an input is changed by the users

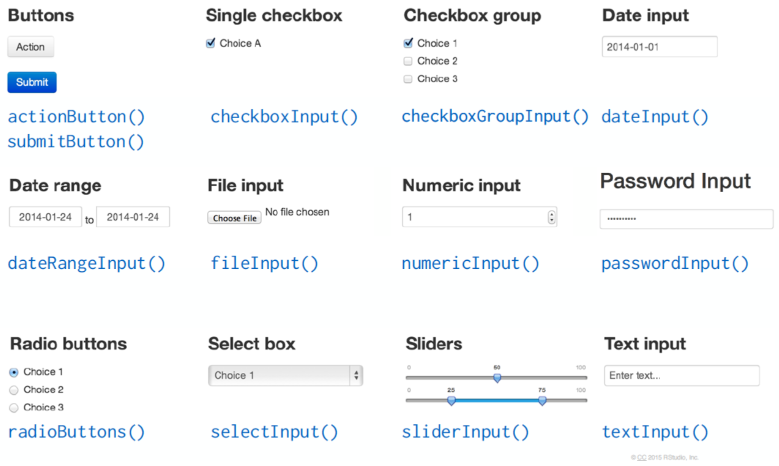
What is reactivity?

- Reactivity: connecting inputs to outputs
  - Allow outputs to automatically update when an input is changed by the users
  - Output has a *reactive dependency* on input

What is reactivity?

- Reactivity: connecting inputs to outputs
  - Allow outputs to automatically update when an input is changed by the users
  - Output has a *reactive dependency* on input
  - You can't read `input$...` or modify `output$...` outside of a reactive context

Basic Input functions



- Taken from R Studio Shiny tutorial
- See more in Shiny Widgets Gallery

Basic `Output` and `render` functions

`Output` functions	Insert	Corresponding `render`
dataTableOutput()	an interactive table	renderDataTable()
imageOutput()	image	renderImage()
plotOutput()	plot	renderPlot()
tableOutput()	table	renderTable()
textOutput()	text	renderText()
verbatimTextOutput()	text	renderText()
uiOutput()	a Shiny UI element	renderUI()
htmlOutput()	raw HTML	renderUI()

Practice time!

- Start with these four lines of code:

```r
library(shiny)

ui <- fluidPage()

server <- function(input, output) {}

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

Layouts in UI: Sidebar Layout

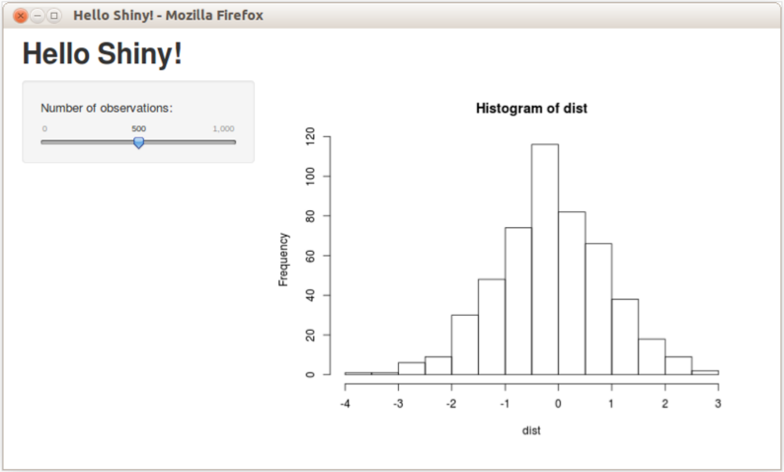
- See more here Application layout guide

```r
ui <- fluidPage(

  titlePanel("Hello Shiny!"),

  sidebarLayout(

    sidebarPanel(
      sliderInput("obs", "Number of observations:",
                  min = 1, max = 1000, value = 500)
    ),

    mainPanel(
      plotOutput("distPlot")
    )
  )
)
```

Layouts in UI: Sidebar Layout

Sidebar Layout

The sidebar layout is a useful starting point for most applications. This layout provides a sidebar for inputs and a large main area for output:



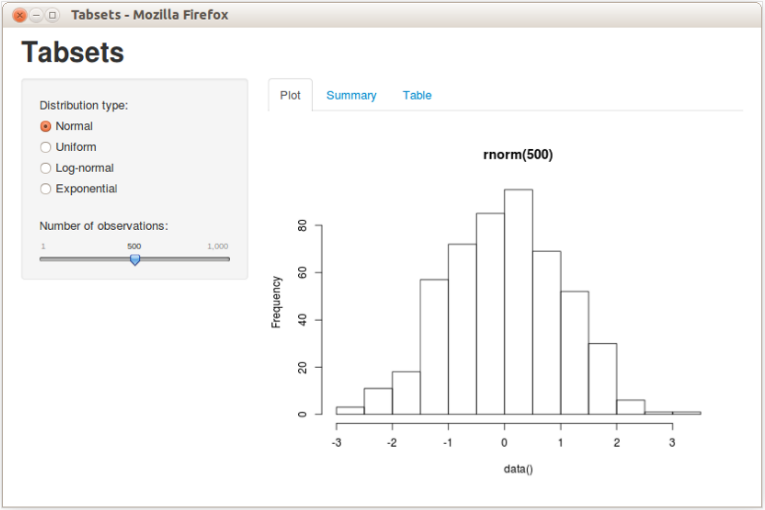
Layouts in UI: Tabsets

```r
ui <- fluidPage(

  titlePanel("Tabsets"),

  sidebarLayout(

    sidebarPanel(
      # Inputs excluded for brevity
    ),

    mainPanel(
      tabsetPanel(
        tabPanel("Plot", plotOutput("plot")),
        tabPanel("Summary", verbatimTextOutput("summary")),
        tabPanel("Table", tableOutput("table"))
      )
    )
  )
)
```

Layouts in UI: Tabsets

Tabsets

Often applications need to subdivide their user-interface into discrete sections. This can be accomplished using the `tabsetPanel()` function. For example:



Extension packages to check out

- `plotly` for interactive plots (e.g. hovering over points)
- `highcharter` for R wrapper for Highcharts javascript library
- `shinyWidgets` for even more widgets
- `shinythemes` for Shiny themes
- A complete list of extension packages here

Other ressources

Introductory book on R Shiny:

- Wickham (2021) - Mastering Shiny

Deployment of Shiny apps on the web:

- Hosting and deployment.
- Deploying Shiny apps to the web
- How to publish a Shiny app? An example with shinyapps.io.
- Deploy Shiny App on Github Pages

You can also find video tutorials in YouTube!