

CSSS 569 Visualizing Data and Models

Lab 8: Interactive Visual Display with R + Shiny

Brian Leung

Department of Political Science, UW

December 29, 2021

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)
```

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

```
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

```
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

```
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

```
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

```
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

```
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?

- ▶ 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
 - ▶ Allow Shiny to be responsive but computationally efficient (lazy)

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
 - ▶ Allow Shiny to be responsive but computationally efficient (lazy)
 - ▶ You can't read `input$...` or modify `output$...` outside of a reactive context

Basic Input functions

Buttons

Action

Submit

`actionButton()`
`submitButton()`

Single checkbox

☒ Choice A

`checkboxInput()`

Checkbox group

☒ Choice 1

☐ Choice 2

☐ Choice 3

`checkboxGroupInput()` `dateInput()`

Date input

2014-01-01

Date range

2014-01-24 to 2014-01-24

`dateRangeInput()`

File input

No file chosen

`fileInput()`

Numeric input

1

`numericInput()`

Password Input

`passwordInput()`

Radio buttons

☒ Choice 1

☐ Choice 2

☐ Choice 3

`radioButtons()`

Select box

Choice 1

`selectInput()`

Sliders



`sliderInput()`

Text input

Enter text...

`textInput()`

© CC 2015 RStudio, Inc.

- ▶ Taken from R Studio Shiny tutorial
- ▶ See more in [Shiny Widgets Gallery](#)

Basic Output and render functions

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

Practice time!

► Start with these four lines of code:

```
library(shiny)
```

```
ui <- fluidPage()
```

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

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

Layouts in UI: Sidebar Layout

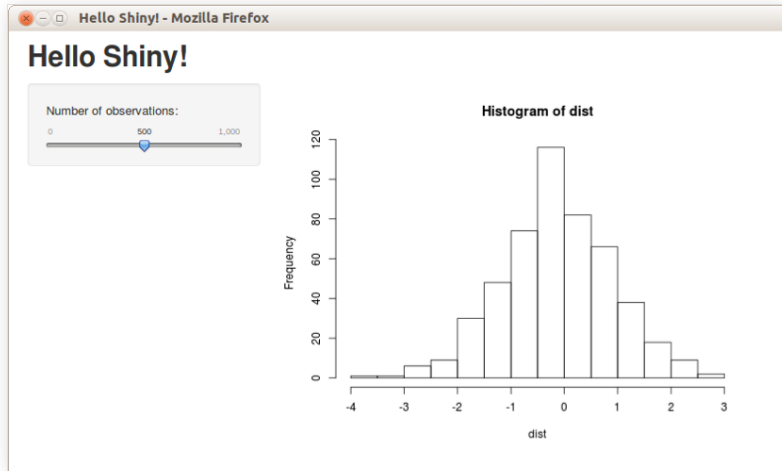
► See more here [Application layout guide](#)

```
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:



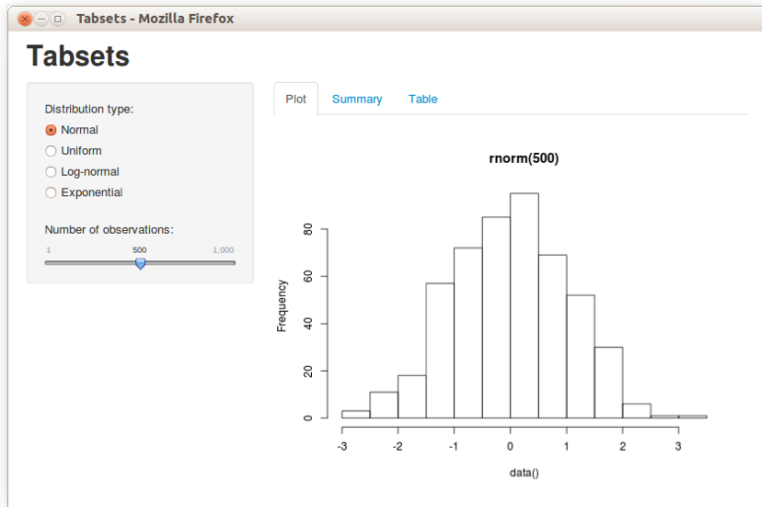
Layouts in UI: Tabsets

```
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)

Extension packages to check out

- ▶ `plotly` for interactive plots (e.g. hovering over points)
- ▶ `shinyWidgets` for even more widgets

Extension packages to check out

- ▶ `plotly` for interactive plots (e.g. hovering over points)
- ▶ `shinyWidgets` for even more widgets
- ▶ `shinythemes` for Shiny themes

Extension packages to check out

- ▶ `plotly` for interactive plots (e.g. hovering over points)
- ▶ `shinyWidgets` for even more widgets
- ▶ `shinythemes` for Shiny themes
- ▶ A complete list of extension packages [here](#)