CSSS 569 Visualizing Data and Models
Lab 8: Interactive Visual Display with R + Shiny

Brian Leung

Department of Political Science, UW

February 24, 2023
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

```r
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

```r
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

```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
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

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?

- 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

<table>
<thead>
<tr>
<th>Buttons</th>
<th>Single checkbox</th>
<th>Checkbox group</th>
<th>Date input</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>actionButton()</code></td>
<td><code>checkboxInput()</code></td>
<td><code>checkboxGroupInput()</code></td>
<td><code>dateInput()</code></td>
</tr>
<tr>
<td><code>submitButton()</code></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Date range</th>
<th>File input</th>
<th>Numeric input</th>
<th>Password Input</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dateRangeInput()</code></td>
<td><code>fileInput()</code></td>
<td><code>numericInput()</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>Choose File</code></td>
<td></td>
<td><code>passwordInput()</code></td>
</tr>
<tr>
<td></td>
<td><code>No file chosen</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Radio buttons</th>
<th>Select box</th>
<th>Sliders</th>
<th>Text input</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>radioButtons()</code></td>
<td><code>selectInput()</code></td>
<td><code>sliderInput()</code></td>
<td><code>textInput()</code></td>
</tr>
</tbody>
</table>

- Taken from R Studio Shiny tutorial
- See more in Shiny Widgets Gallery
## Basic Output and `render` functions

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

▶ Start with these four lines of code:

```r
library(shiny)

ui <- fluidPage()

server <- function(input, output) {}

shinyApp(ui = ui, server = server)
```
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:
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)
- **highcharter** for R wrapper for Highcharts javascript library
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
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
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](#)