Why to learn R + Shiny?

▶ Very useful for exploratory data analysis.
Why to learn R + Shiny?

- Very useful for exploratory data analysis.
- Interactive visual displays that you can share with others.
Why to learn R + Shiny?

- Very useful for exploratory data analysis.
- Interactive visual displays that you can share with others.
- Business/work applications: teaching, software to implement research applications, operations research applications, etc.
Why to learn R + Shiny?

- Very useful for exploratory data analysis.
- Interactive visual displays that you can share with others.
- Business/work applications: teaching, software to implement research applications, operations research applications, etc.
- Nonetheless, it takes time and practice to learn.
Why to learn R + Shiny?

- Very useful for exploratory data analysis.
- Interactive visual displays that you can share with others.
- Business/work applications: teaching, software to implement research applications, operations research applications, etc.
- Nonetheless, it takes time and practice to learn.
  - Learn by replication.
By the end of this lab
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)
u <- fluidPage(...)
s <- function(input, output) {...}
shinyApp(u = u, s = s)
```

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
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$...` **in a reactive context**
   - Create output values via `render()` or `reactive()` **in a reactive context**
     - Within `render()` or `reactive()`, write code to perform some tasks
     - Store them as elements of output via `output$...`
Basic Input functions

Buttons
- `actionButton()`
- `submitButton()`

Single checkbox
- `checkboxInput()`
- `checkboxGroupInput()`

Checkbox group
- `radioButtons()`
- `selectInput()`

Date input
- `dateRangeInput()`
- `fileInput()`
- `numericInput()`
- `passwordInput()`
- `sliders()`
- `textInput()`

- 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 render</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataTableOutput()</td>
<td>an interactive table</td>
<td>renderDataTable()</td>
</tr>
<tr>
<td>imageOutput()</td>
<td>image</td>
<td>renderImage()</td>
</tr>
<tr>
<td>plotOutput()</td>
<td>plot</td>
<td>renderPlot()</td>
</tr>
<tr>
<td>tableOutput()</td>
<td>table</td>
<td>renderTable()</td>
</tr>
<tr>
<td>textOutput()</td>
<td>text</td>
<td>renderText()</td>
</tr>
<tr>
<td>verbatimTextOutput()</td>
<td>text</td>
<td>renderText()</td>
</tr>
<tr>
<td>uiOutput()</td>
<td>a Shiny UI element</td>
<td>renderUI()</td>
</tr>
<tr>
<td>htmlOutput()</td>
<td>raw HTML</td>
<td>renderUI()</td>
</tr>
</tbody>
</table>
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
What is reactivity?

Think Excel.
What is reactivity?

<p>| | | | | | | | | | | | | | | | | | | | |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |</p>
<table>
<thead>
<tr>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
<th>K</th>
<th>L</th>
<th>M</th>
<th>N</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>F4 + 1</td>
</tr>
</tbody>
</table>
What is reactivity?
What is reactivity?

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>999</td>
</tr>
<tr>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What is reactivity?

<table>
<thead>
<tr>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
<th>K</th>
<th>L</th>
<th>M</th>
<th>N</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1000 → 1001
What is reactivity?
library(shiny)

ui <- fluidPage()

server <- function(input, output) {}

shinyApp(ui = ui, server = server)
How to share shiny apps?

- When you run your shiny app, it’s running in your local computer.

- Nonetheless, you can share your shiny app uploading it at one online server.

  In doing that, you will have to name your script file as app.R.

- It is also possible to split your shiny app in two script files.

- See here more detail on how to share shiny apps.

- In addition, you can deploy shiny apps online using the RStudio platform.
How to share shiny apps?

- When you run your shiny app, it's running in your local computer.
- Nonetheless, you can share your shiny app uploading it at one online server.

See here more detail on how to share shiny apps.

In addition, you can deploy shiny apps online using the RStudio platform.
How to share shiny apps?

- When you run your shiny app, it’s running in your local computer.
- Nonetheless, you can share your shiny app uploading it at one online server.
  - In doing that, you will have to name your script file as `app.R`. 
How to share shiny apps?

▶ When you run your shiny app, it’s running in your local computer.
▶ Nonetheless, you can share your shiny app uploading it at one online server.
  ▶ In doing that, you will have to name your script file as `app.R`. 
  ▶ It is also possible to split your shiny app in two script files.
How to share shiny apps?

▶ When you run your shiny app, it’s running in your local computer.
▶ Nonetheless, you can share your shiny app uploading it at one online server.
  ▶ In doing that, you will have to name your script file as *app.R*.
  ▶ It is also possible to split your shiny app in two script files.
▶ See [here](#) more detail on how to share shiny apps.
How to share shiny apps?

► When you run your shiny app, it’s running in your local computer.
► Nonetheless, you can share your shiny app uploading it at one online server.
  ► In doing that, you will have to name your script file as *app.R*.
  ► It is also possible to split your shiny app in two script files.
► See [here](#) more detail on how to share shiny apps.
► In addition, you can deploy shiny apps online using the RStudio platform.
Other resources to check out

- Here the official tutorial (~ 3 hours).
Other resources to check out

- Here the official tutorial (~ 3 hours).
- Wickham (2020) - Masterign Shiny
Other resources to check out

- Here the official tutorial (~ 3 hours).
- Wickham (2020) - Mastering Shiny
- For application’s layout guide, see more in here.
Other resources to check out

- Here the official tutorial (~ 3 hours).
- Wickham (2020) - Masterign Shiny
- For application’s layout guide, see more in here.
- See Gallery with examples and code to replicate.
Other resources to check out

- Here the official tutorial (~ 3 hours).
- Wickham (2020) - Masterign Shiny
- For application’s layout guide, see more in here.
- See Gallery with examples and code to replicate.
- A complete list of extension packages here