Shiny

James Balamuta

Department of Informatics, Statistics
University of Illinois at Urbana-Champaign

July 19, 2017

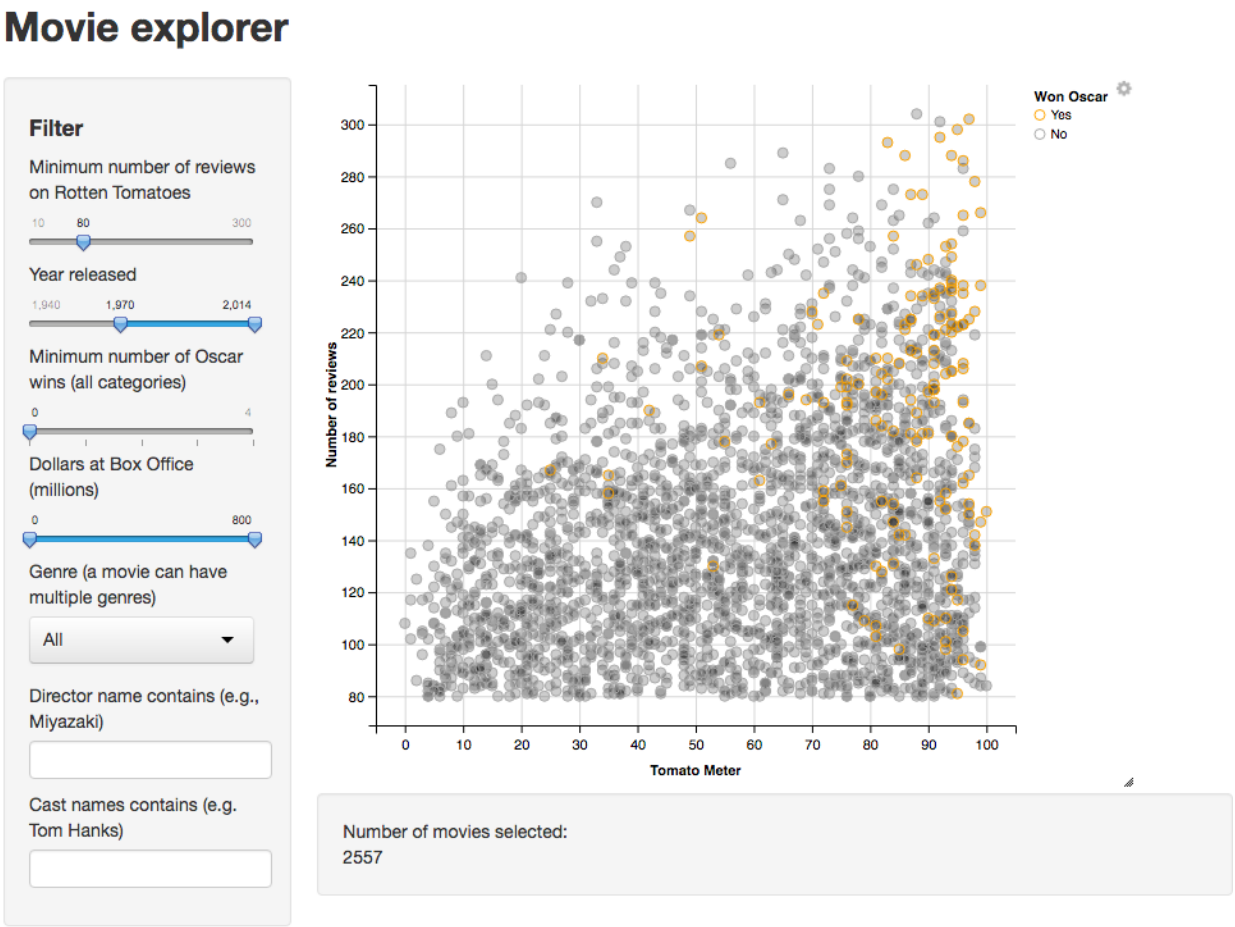
On the Agenda

- Shiny
  - Background information
  - Making an App
    - Frontend vs. Backend

On the Agenda

What is Shiny?

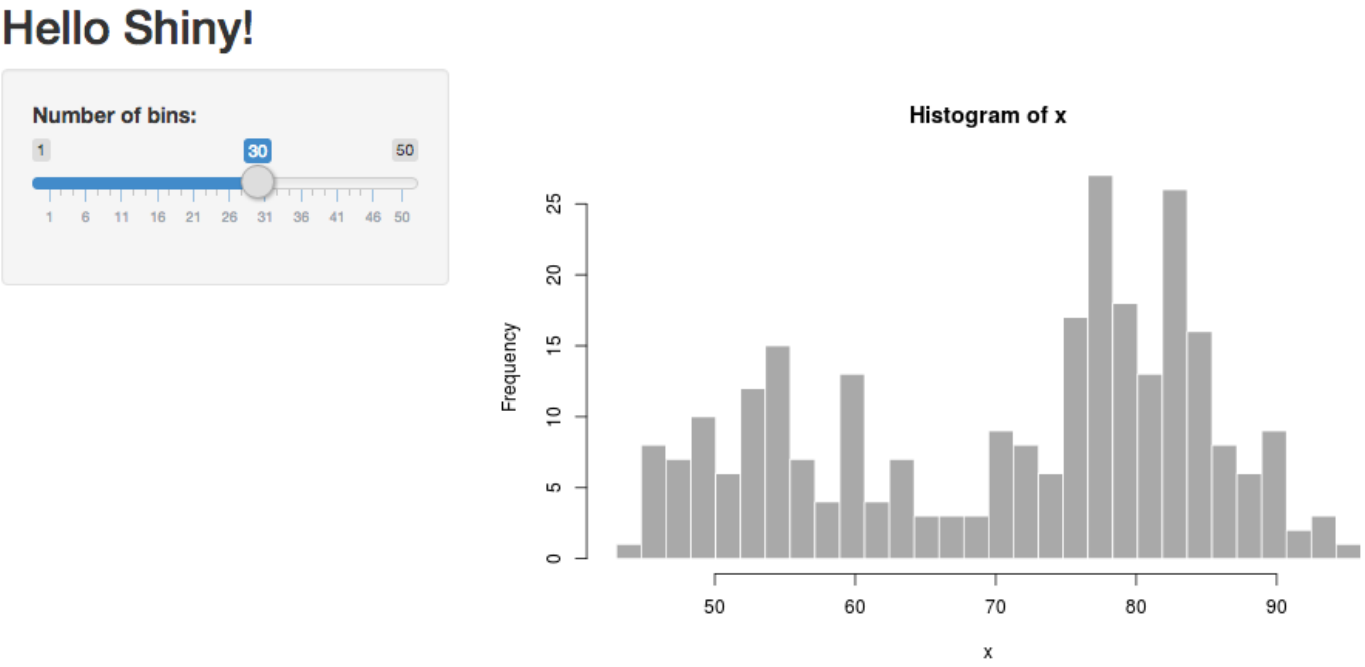
Shiny is an R package that makes it easy to build interactive web applications (apps) straight from R.



Why Shiny?

- Access features in the *R* ecosystem without knowing *R*!
- Standardized interactive explorations of data
- Easy deployments via:
  - **Local:** shiny::runApp()
    - development and package inclusion
  - **Server:** shiny-server
    - On premise use for companies
    - STATS@UIUC runs this on: rstudio.stat.illinois.edu/shiny
  - **Cloud:** shinyapps.io
    - Avoids management headaches and have easy access to scaling computational resources.

Hello Shiny World!

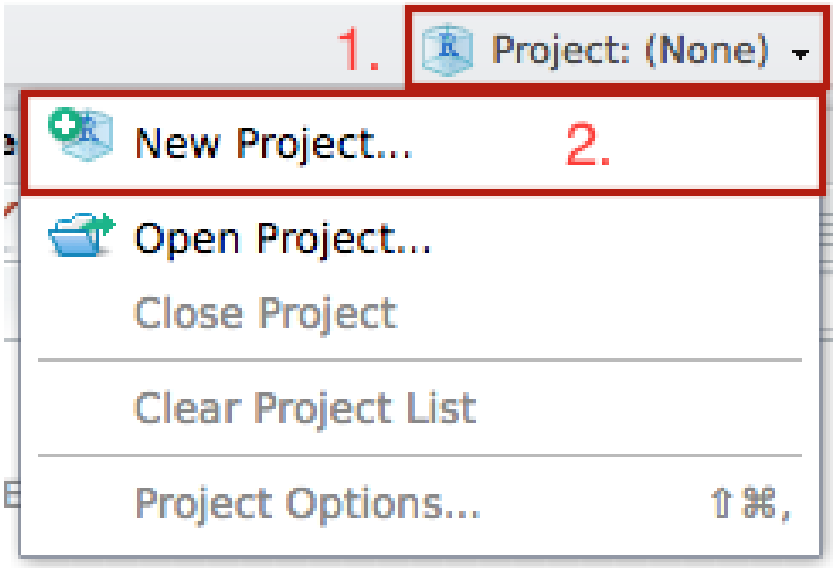



```
# install.packages("shiny") # Install if on local
library(shiny)            # Load Shiny
runExample("01_hello")    # Run above example
```

On the Agenda

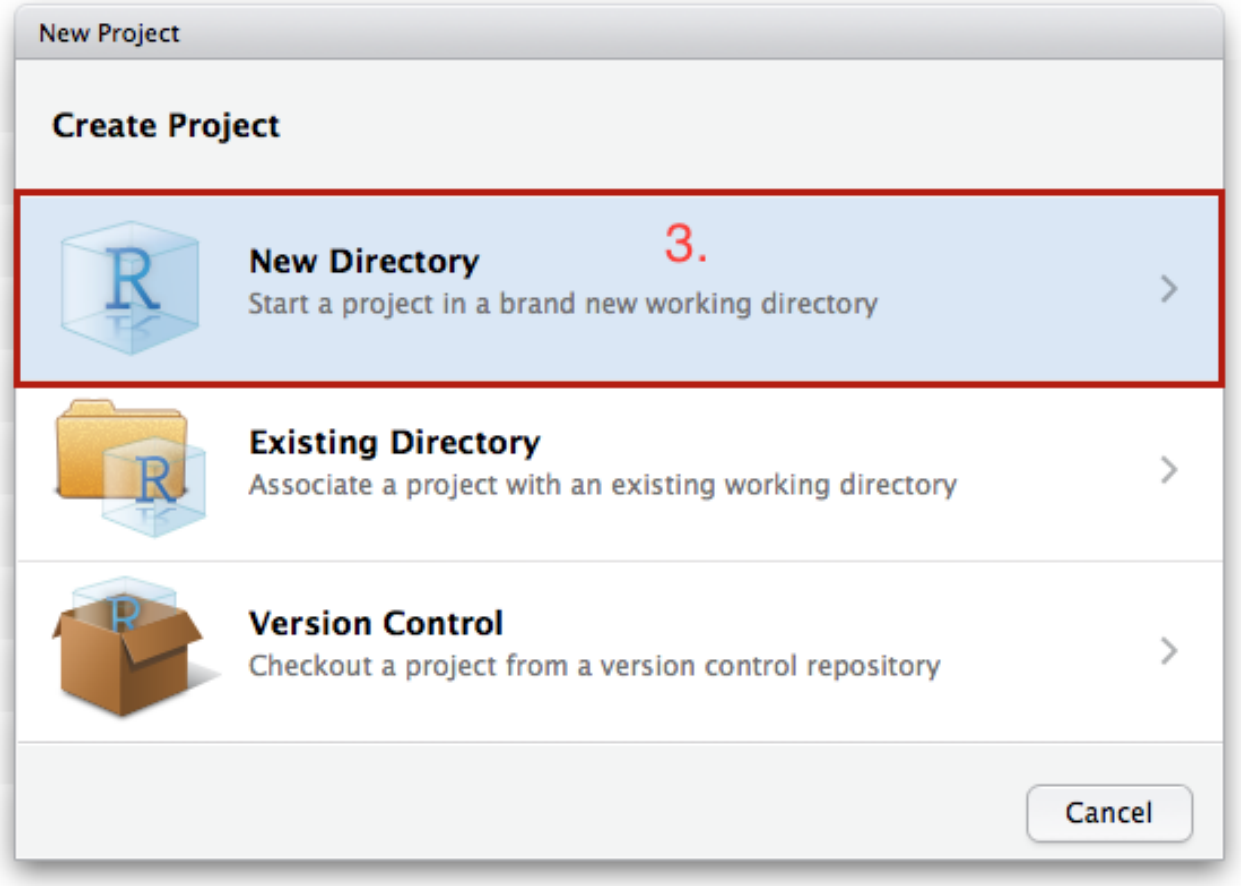
Setting up a Shiny Project - Dropdown Menu

- Select the project dropdown menu and press **New Project**



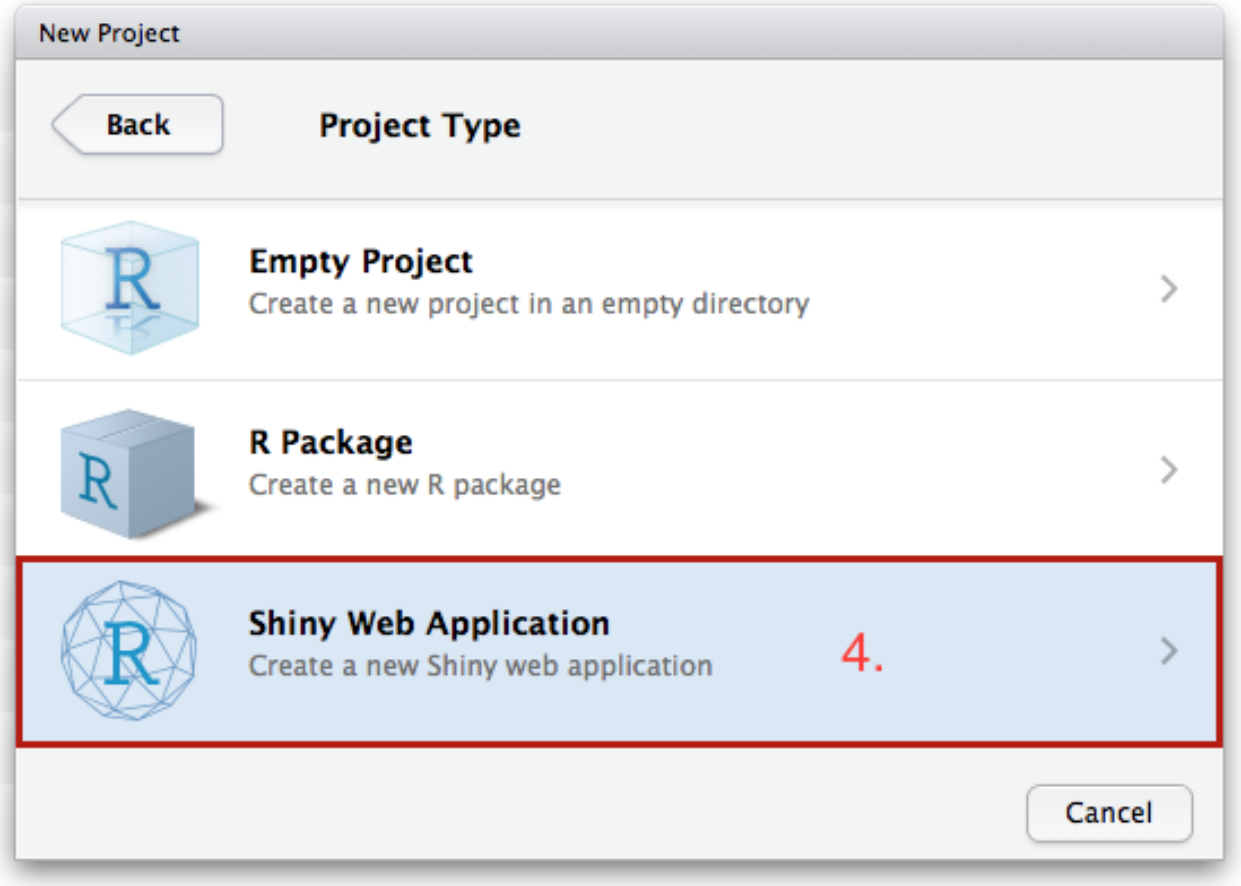
Setting up a Shiny Project - New Directory

- Select **New Directory**



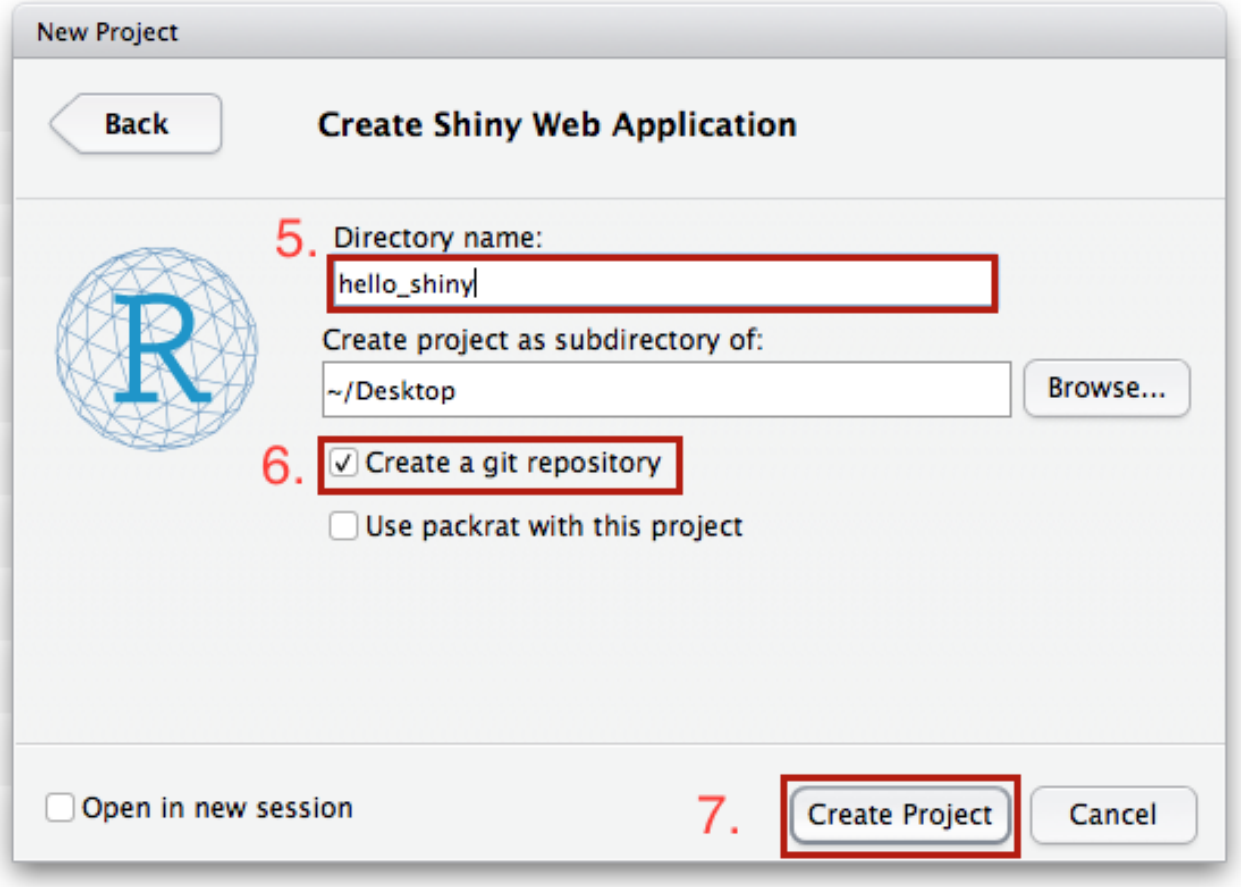
Setting up a Shiny Project - Project Type

- Select **Shiny Web Application**



Setting up a Shiny Project - Initialization Values

- Enter a project name (directory) for your shiny app.
- Check the **Create a git repository**
- Press **Create Project**



Exploring the Default Shiny App - Structure

- Once the project is created, an example shiny app is centerfold:

ui.R server.R

Run App

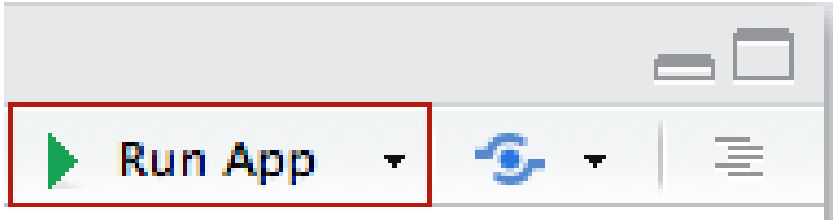
```
1
2  # This is the user-interface definition of a Shiny web application.
3  # You can find out more about building applications with Shiny here:
4  #
5  # http://shiny.rstudio.com
6  #
7
8  library(shiny)
9
10 shinyUI(fluidPage(
11
```

Note: The presence of two files *ui.R* and *server.R*

Exploring the Default Shiny App - Running

To run a shiny within a project there are three options:

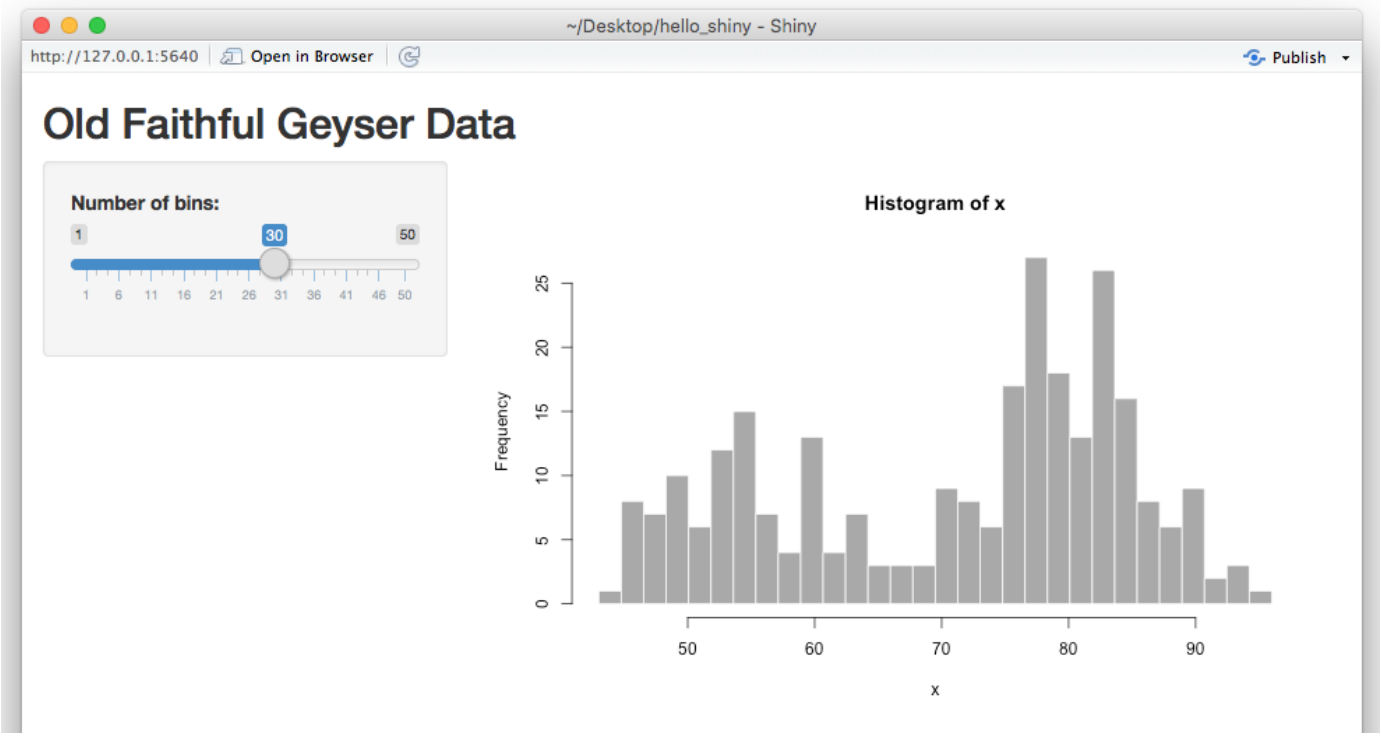
1. Type `runApp()` in **Console**
2. Use a keyboard shortcut
   - macOS: `Command` + `Shift` + `Enter`
   - Windows: `Control` + `Shift` + `Enter`
3. Press the `Run App` button at the upper right of the script editor.



Exploring the Default Shiny App - Live App

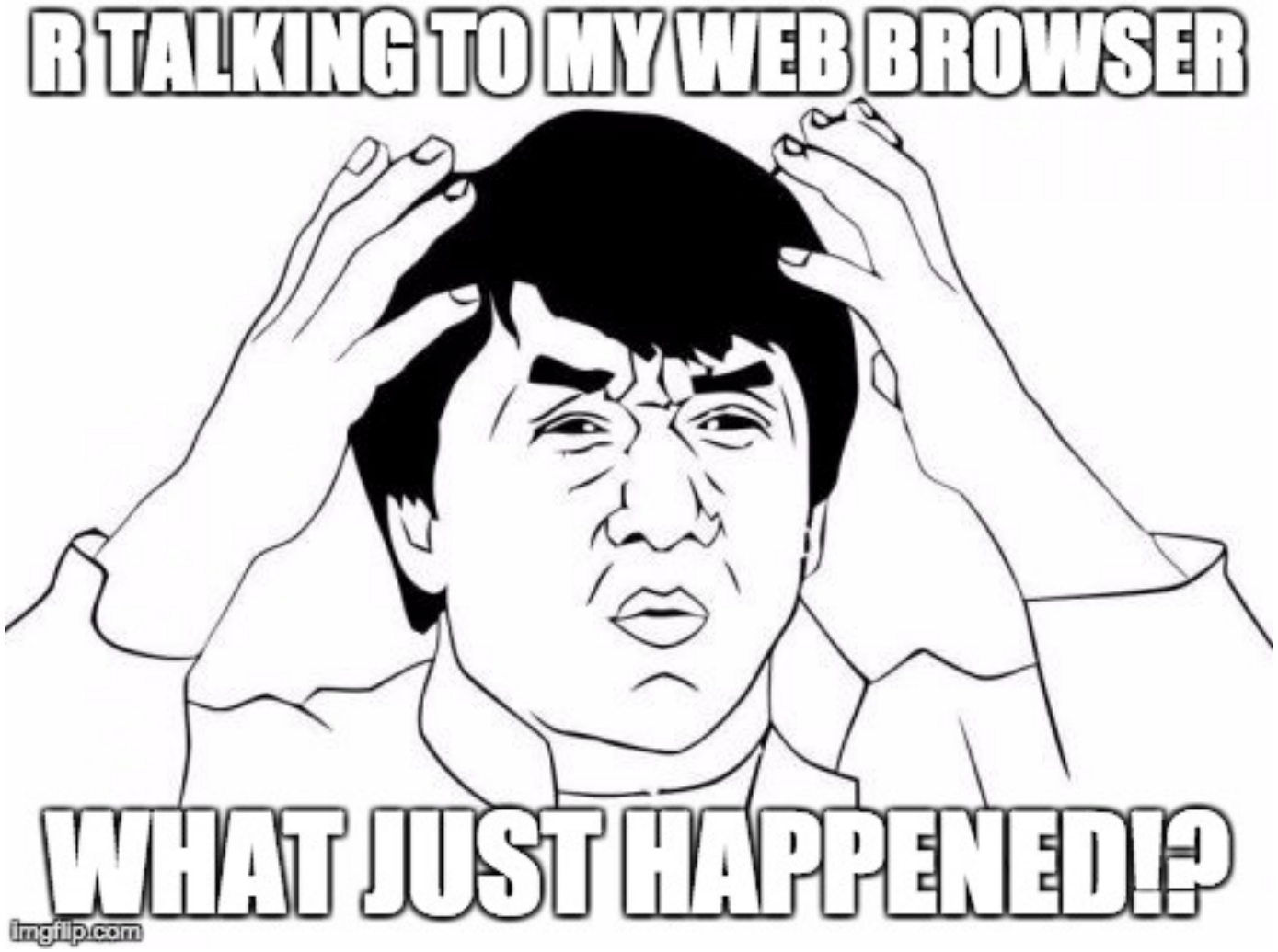
A secondary window will open and the Shiny app will be displayed.

- **Note:** Using RStudio on the analytical environment may require you to allow pop-ups!



Try moving the slider and comment to your group mates what happens to

Lions, Tigers, and Bears... Oh my!



On the Agenda

Behind the Scenes a Shiny App

As hinted to earlier, there are two files responsible for the creation of the shiny App: **ui.R** and **server.R**.

- **ui.R:** is responsible for providing the user interface (ui) or *frontend* for the shiny application.
- **server.R:** is responsible for providing the *backend* logic behind each change that occurs due to a button click, slider drag, et cetera on the frontend.

Behind the Scenes a Shiny App

The following is the bare minimum for a Shiny App to function.

ui.R

```
shinyUI(        # Initialize a UI container in Shiny
  fluidPage() # Make a page layout
)
```

server.R

```
shinyServer(                    # Initialize Server
  function(input, output) { # Input and output
  }
)
```

Blank Shiny

Note: Running the previous code will yield an empty app with a blank user-interface.

Beginning a Shiny App

- To motivatie our exploration of Shiny, we will create a shiny app that is able to *switch* between different datasets.
- We will begin by first constructing the User Interface (**ui.R**)
- Then we will write the backend logic (**server.R**)

Making Content

We can add content to the UI by using:

Function	Description
`titlePanel()`	Naming the application (e.g. Hello Shiny!)
`sidebarLayout()`	Creates a sidebar layout for the `fluidPage()`.
`sidebarPanel()`	Makes a side bar **menu** for UI Controls and Instructions
`mainPanel()`	Main content area to house graphs, tables, text output

Making Content for the Interface

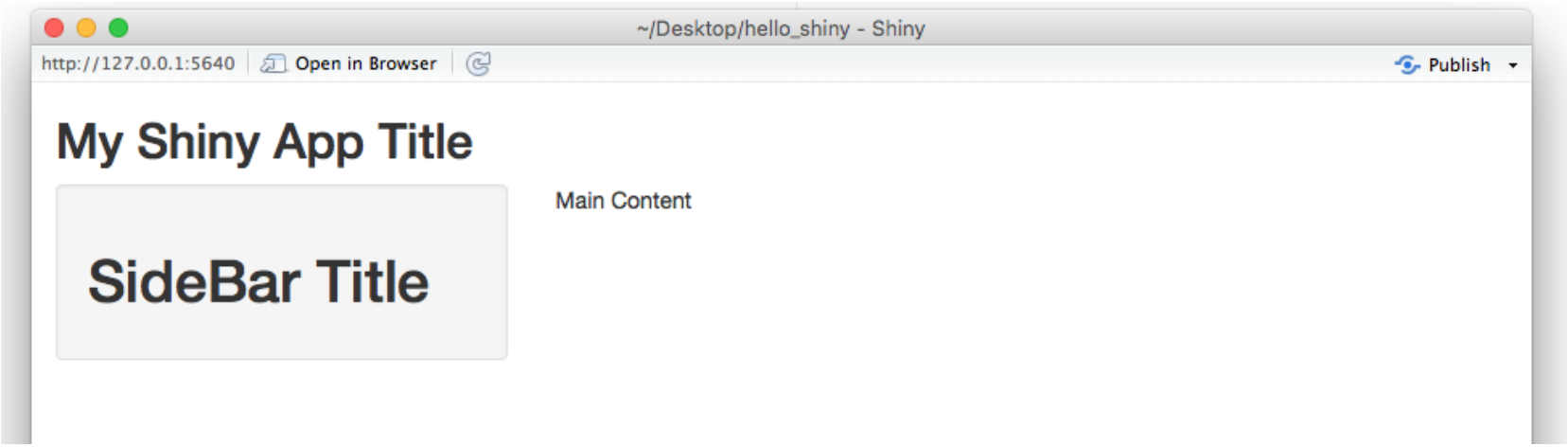
ui.R

```
shinyUI(
  fluidPage(
    titlePanel("My Shiny App Title"), # Title

    sidebarLayout(

      sidebarPanel(
        h1("SideBar Title")           # Sidebar Text
        ),                            # Note HTML

      mainPanel("Main Content")       # Content Text
    )
  )
)
```

Note: You can use attributes such as `align = "center"` by `h1("SideBar Title", align = "Center")`

Making Content for the Interface - Preview

If we run our app, we will get:



HTML in Shiny

Function	HTML	Description
`strong()`	`<strong></strong>`	Bold Text
`em()`	`<em></em>`	Italicize Text
`a()`	`<a></a>`	Makes a hyperlink
`p()`	`<p></p>`	Text Paragraph
`h1()`	`<h1></h1>`	Header (replace 1)
`br()`	` `	Creates a page break
`div()`	`<div></div>`	Division of text
`code()`	`<code></code>`	Code formated block
`HTML()`	-	Embed own HTML Code

Note: `h2()` up to `h6()` provides different heading styles.

- **More Shiny HTML Tags...** (About 110 of them!)
- **UI Customization with HTML**

Making Inputs

- Create HTML from within R is nice, but we want to be able to talk to R.
- To do that, we must make some sort of input control.
- In Shiny, the input control comes from *widgets*

Making Widgets for Input

To construct a **widget**, we must:

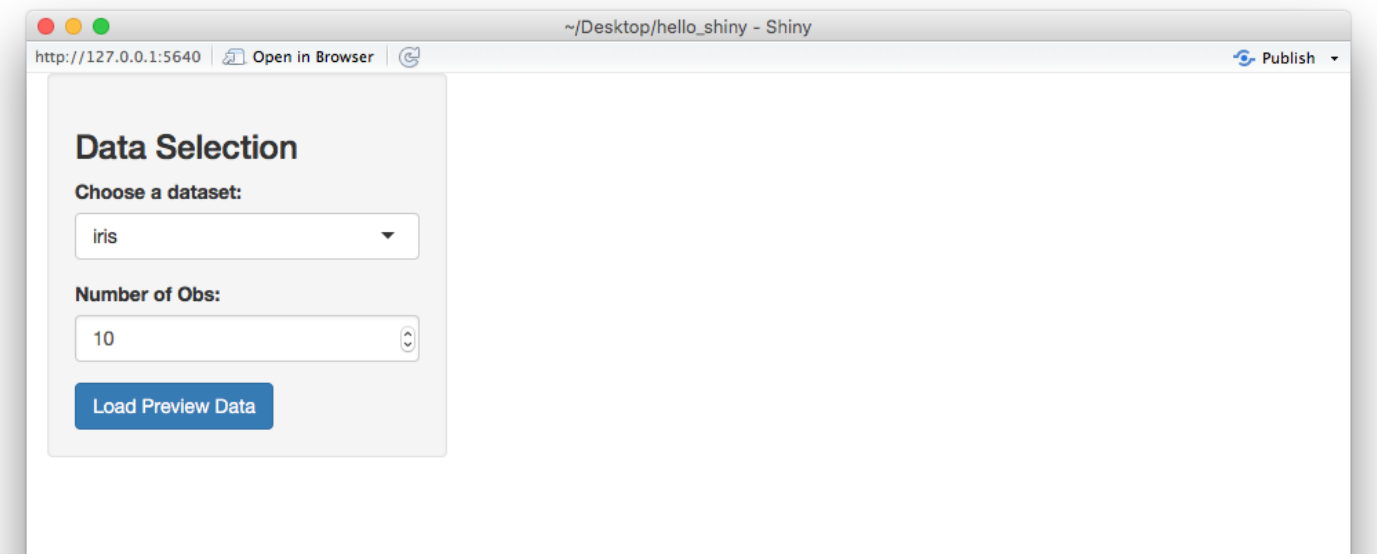
- Provide a `name=""`
  - We will use this to get the active value.
  - Users will not be able to see the name.
- Provide a `label=""`
  - This describes the widget to the user.

Making Widgets for Input - Example

ui.R

```
sidebarLayout(
  sidebarPanel(
    h3("Data Selection"),              # Note the ,

    # Dropdown
    selectInput("ds",                  # Name
                "Choose a dataset:",   # Label
                choices = c("iris", "Spam", "mtcars")),

    numericInput("obs",                # Name
                 "Number of Obs:",     # Label
                 10),                  # Default Value

    submitButton("Load Preview Data") # Update data
  ),
  mainPanel())# Not Displayed          # Content
```

Making Widgets for Input - Preview



UI Input Controls

Shiny features a lot of different ways to accept user input

Function	Description
`numericInput()`	Number entry input
`radioButtons()`	Radio button selection
`selectInput()`	Dropdown menu
`sliderInput()`	Range slider (1/2 values)
`submitButton()`	Submission button
`textInput()`	Text input box
`checkboxInput()`	Single checkbox input
`dateInput()`	Date Selection input
`fileInput()`	Upload a file to Shiny
`helpText()`	Describe input field

See **Shiny Widgets Gallery** for examples.

Making Render UI Areas

- So far, we have managed to make stylistic features and input controls.
- However, in order for the *Shiny* app to be dynamic and display data, we must have output control or render areas.
- To do so:
  1. We add an output control to **ui.R**.
  2. Make some logic in **server.R** to talk with it! (Yes, we're almost there.)

Making Render UI Areas - Example

```
sidebarLayout(
  sidebarPanel(), # Given previously
  mainPanel(
    h3("Head of the Dataset"),     # HTML
    tableOutput("view"),           # Table View

    h3("Dataset Summary"),         # HTML
    verbatimTextOutput("summary") # Output Asis
  )
)
```

Note: Like the input control, we do *name* the output values.

UI Output Controls

There are many ways to render the results

Function	Description
`plotOutput()`	Display a rendered plot
`tableOutput()`	Display in Table
`textOutput()`	Formatted Text Output
`uiOutput()`	Dynamic UI Elements
`verbatimTextOutput()`	"as is" Text Output
`imageOutput()`	Render an Image
`htmlOutput()`	Render Pure HTML

Also see:

- **Dyanmically Generated User Interface Components**
- **Changing the Values of Inputs from the Server**

Moving over to **server.R**

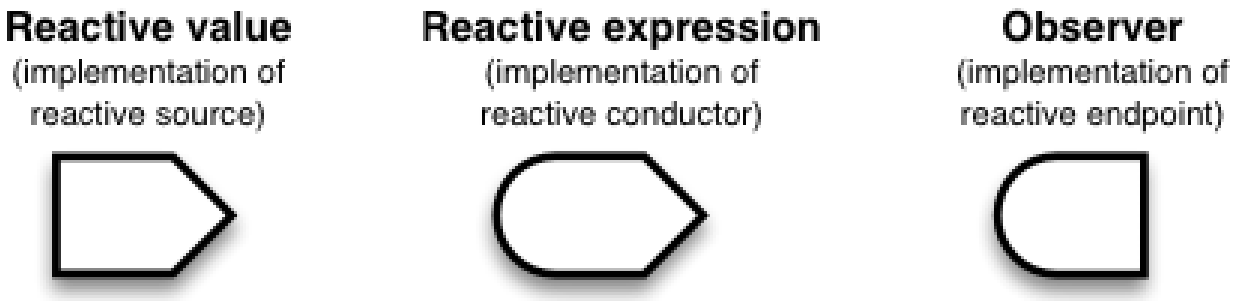
- We've finished what we needed to accomplish in the **ui.R** file.
- Now, we must write the backend logic in **server.R**.

What is Reactivity?

"For every action, there is an equal and opposite reaction."
– Issac Newton

What is Reactivity?

- **Reactive Sources (Reactive Values)**
  - UI element inputs
- **Reactive Conductors (Reactive Expressions)**
  - Server Catches for UI elements `reactive({})`
- **Reactive Endpoints (Observers)**
  - Render functions in the UI and `observer({})` in Server



View **Reactivity Explanation**

Note: Reactive expressions return values, but observers don't.

Accessing a Reactive Element

Reactive elements can be found living in either the `input`, `output`, or `session` variables. The later of which is only found in Shiny Server Pro applications.

To access a reactive source from the UI use name you gave to the component:

```
input$name
```

So, to access to the data set choice, we would use:

```
input$ds
```

Creating a Reactive Catch

server.R

```
library("msos"); library("dataset")
data("Spam")
shinyServer(function(input, output) {

  dsInput = reactive({   # Reactive
    switch(input$ds,     # Load dataset
           "iris" = iris,
           "Spam" = Spam,
           "mtcars" = mtcars)
  })

})
```

Observers In-depth

Observers perform *actions* and do *not* return values when either reactive values or expressions change.

There are two forms of observers:

- **Implicit**: Triggered whenever *any* reactive values or expressions changed within the scope:

```
observe({...})
```

- **Explicit**: Code is triggered only when a specific reactive value or expression changes. Any other reactive values outside of the **observed reactive event** are ignored.

```
observeEvent(observed_reactive_event, {...})
```

Observers example

```
shinyServer(function(input, output) {

  # Whenever either input$x or input$y change, execute
  observe({
    cat("input$x has", input$x,
        "and input$y has", input$y ,"\n")
  })

  # Execute only when form submission occurs
  observeEvent(input$form_submission, {
    cat("On form submit, we have: input$x with", input$x,
        "and input$y with", input$y ,"\n")
  })
}
```

Creating Output Hooks

Output hooks defined as:

```
output$view = renderTable({...})
```

serves as a recipe for what should be used when updating `view`.

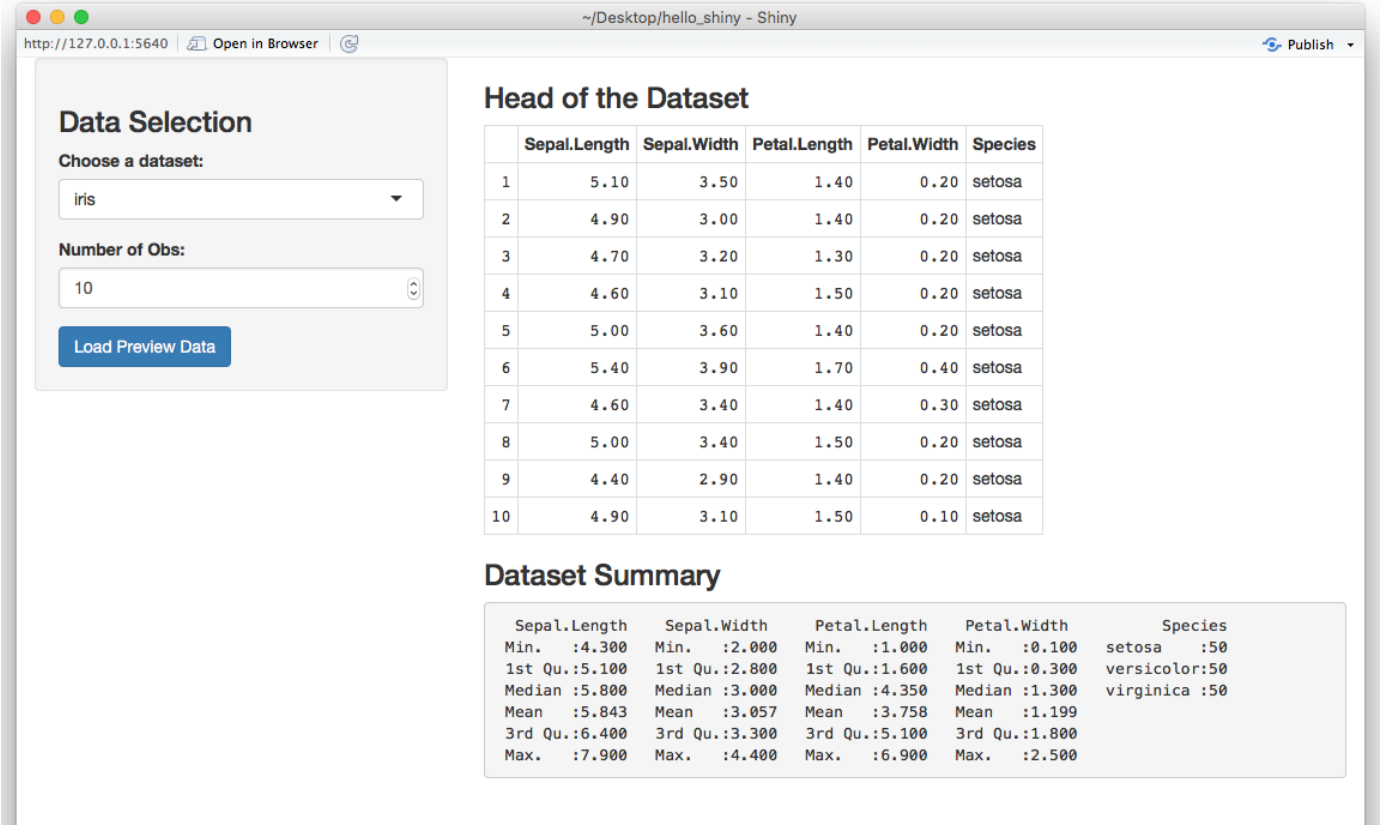
Avoid making the mistake of interpreting the code as triggering the update to `view` with the results.

Creating Output Hooks

server.R

```
shinyServer(function(input, output) {

  ## Hiding data set reactive

  output$summary = renderPrint({    # Summary Render
    summary(dsInput())
  })

  output$view = renderTable({       # Table Render
    head(dsInput(), n = input$obs)
  })
})
```

Creating Observer Hooks - Preview



	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.10	3.50	1.40	0.20	setosa
2	4.90	3.00	1.40	0.20	setosa
3	4.70	3.20	1.30	0.20	setosa
4	4.60	3.10	1.50	0.20	setosa
5	5.00	3.60	1.40	0.20	setosa
6	5.40	3.90	1.70	0.40	setosa
7	4.60	3.40	1.40	0.30	setosa
8	5.00	3.40	1.50	0.20	setosa
9	4.40	2.90	1.40	0.20	setosa
10	4.90	3.10	1.50	0.10	setosa

```
 Sepal.Length    Sepal.Width     Petal.Length    Petal.Width          Species
 Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100   setosa    :50
 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300   versicolor:50
 Median :5.800   Median :3.000   Median :4.350   Median :1.300   virginica :50
 Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
 3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
 Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
```

Displaying Reactivity

The functions below are meant to interface with the `*Output()` UI functions.

Function	Description
`renderPlot()`	Display Plots
`renderPrint()`	Output Print (Verbatim)
`renderTable()`	Tables for 2D Data Structures
`renderText()`	Display Character Strings
`renderUI()`	Dynamic UI render
`renderImage()`	Saved Images on Disk

Understanding Shiny Runtime Components

Shiny runtime components is slightly different than normal. Certain areas of the **server.R** are either run:

- Once on startup
  - Initializing the application on server
- Once per user visit
  - Loading user info
- Many times per session
  - Reactive control

Understanding Shiny Runtime Components - Startup

server.R

```
load("data.rda")                # Once during startup

shinyServer(                    # Once during startup

  function(input, output) {
    toad = "Hello"

    output$test = renderUI({

    })
  }
)
```

Understanding Shiny Runtime Components - User Session

server.R

```
load("data.rda")

shinyServer(

  function(input, output) {  # Once per user
    toad = "Hello"

    output$test = renderUI({

    })
  }
)
```

Understanding Shiny Runtime Components - Actions

server.R

```
load("data.rda")

shinyServer(

  function(input, output) {
    toad = "Hello"

    output$test = renderUI({
                                    # Many Times
    })
  }
)
```

Resources for Shiny

Get inspired
(gallery)

Get started
(tutorial)

Go deeper
(articles)

Shiny Page - **Real Live Apps** - **Video** and **Written Tutorials**

More Resources for Shiny

- Shiny on Github
- Shiny Development Mailing List
- Shiny Function Reference
- Shiny Debugging Tips and Tricks

Acknowledgement

This lecture goes into depth about the Shiny More Widgets Example on Shiny Gallery and discusses reactivity based on Joe Cheng's "Ladder of Enlightenment" for Shiny.