

### INTERACTIVE WEB APPS WITH SHINY

#### Katarzyna M Tyc, PhD

--- Guest Lecturer ---Virginia Commonwealth University 09/25/2023

### REFERENCES

Material covered in these slides is based on the following resources:

ShinyApp gallery: <a href="https://shiny.rstudio.com/gallery/">https://shiny.rstudio.com/gallery/</a>

Official Shiny tutorials: <u>https://shiny.rstudio.com/tutorial/written-tutorial/lesson1/</u> Materials from previous years: <u>https://bios524-r-2021.netlify.app/class/10-class/</u> Other resources: <u>https://debruine.github.io/shinyintro/index.html</u>

Example source: <u>https://www.r-bloggers.com/2019/12/r-shiny-for-beginners-annotated-starter-code/</u>

### PREREQUISITES

#### library(ggplot2)

Basic familiarity with R and RStudio: pets <- read.csv("pets.csv")</pre>

- Data import

- ...

- Data processing
- Data visualization
- If/else statements

```
if (dv == "score") {
  g <- ggplot(pets, aes(pet, score, fill = country))
} else if (dv == "weight") {
  g <- ggplot(pets, aes(pet, weight, fill = country))
}
g + geom_violin(alpha = 0.5)</pre>
```

dv <- sample(c("score", "weight"), 1)</pre>

install.packages("shiny")

### **LECTURE OUTLINE**

Setting up a folder to host your Shiny app

Source code structure of a Shiny app

Adding content into User Interface (UI)

Introducing interactive elements to control the app (a.k.a. widgets)

Connecting widgets to reactive output

### **EXAMPLE SHINYAPPS**

runExample("01\_hello", display.mode = "normal") # a histogram runExample("02\_text") # tables and data frames runExample("03\_reactivity") # a reactive expression runExample("04\_mpg") # global variables runExample("05\_sliders") # slider bars runExample("06\_tabsets") # tabbed panels runExample("07\_widgets") # help text and submit buttons runExample("08\_html") # Shiny app built from HTML runExample("09\_upload") # file upload wizard runExample("10\_download") # file download wizard runExample("11\_timer") # an automated timer

### YOUR FIRST DEMO

	New File	>	R Script 쇼 # N	
D - 🐼 👉 -	New Project		R Notebook	
Console Tern	Open File	жо	R Markdown	
~/ 🖈	Recent Files	>	Shiny Web App	
R version 4.1	Open Project		Plumber API	
Copyright (C) Platform: x86	Open Project in New Session Recent Projects	>	C File C++ File	
R is free sof	Import Dataset	>	Header File	
Type 'license	Save	жs	Markdown File	
Natural lan	Save As		HTML File	
	Save All	<b>℃</b> ₩S	CSS File	
R is a collab Type 'contrib	Publish		JavaScript File	
'citation()'	Print			
Type 'demo()'	Close	۹۴ ۱۸/	Python Script	
'help.start() Type 'a()' to	Close All	ۍ ¥ W	SQL Script	
	Close All Except Current	ጊ ጐ ፝ Ж W	Stan File	
			Text File	
>	Close Project			
>	Close Project Quit Session	жQ	R Sweave	

	Application name:	class	
R	Application type:	<ul> <li>Single File (app.R)</li> <li>Multiple File (ui.R/server.R)</li> </ul>	
	Create within direct	ory: Apps	Browse.
? Shiny Web Applications		Create	Cancel

2

	RStudio
👒 🐨 📲 🔚 🚔 🗼 Go to file/function 🔤 🛛 🔀 👻 Addins 👻	
app.R ×	_
	Run App 👻 💁 🗸
1 #	
2 # This is a Shiny web application. You can run the application by clicking	
3 # the 'Run App' button above.	
4 #	
5 # Find out more about building applications with Shiny here:	
6 #	
7 # http://shiny.rstudio.com/	
8 #	
9	
10 library(shiny)	
11	
12 # Define UI for application that draws a histogram	
13 ui <- fluidPage(	
.4	
<pre>15 # Application title</pre>	
16 titlePanel("Old Faithful Geyser Data"),	
.7	
8 # Sidebar with a slider input for number of bins	
19 SidebarLdyout(	
20 StaebarPanel	
21 Sliderinput("bins",	
number of biris: ,	
max = 50	
$r = \frac{1}{25}$	
26	
27	
28 # Show a plot of the generated distribution	
29 mainPanel(	
<pre>30 plotOutput("distPlot")</pre>	
31 )	
32 )	
33 )	
34	
35 # Define server logic required to draw a histogram	
<pre>36 * server &lt;- function(input, output) {</pre>	
37	
38 * output\$distPlot <- renderPlot({	
oucpues at set toe < Tender Foe()	

Refer to <u>https://debruine.github.io/shinyintro/index.html</u> for practicing!

### **DIRECTORY OF A SHINY APP**

Shiny apps are contained in a single script called **app.R** 

Once you save it in a directory, you can run the app by running runApp()

Example: library(shiny)

runApp("~/Downloads/ShinyApps/class/")
runApp("~/Downloads/ShinyApps/heads\_tails/")
runApp("~/Downloads/ShinyApps/myapp/")

display.mode = "showcase" # allows to see the app code



### STRUCTURE OF A SHINY APP

```
library(shiny)
```

# See above for the definitions of ui and server

ui <- ...

server <- ...

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

#### Example.

shinyapp {snippet}

# just start typing	9		
shiny			
🖹 shinymod	{snippet}	library(shiny)	
📄 shinyapp	{snippet}	ui <- fluidPage(	
🔷 shinyApp	{shiny}	\${0}	
🔷 shinyAppDir	{shiny}	)	
🔷 shinyAppFile	{shiny}	convon < function(input output cossion) {	
shinyAppTemplate	{shiny}	I	
shinyOptions	{shiny}	<u> </u>	
<u>, , , , , , , , , , , , , , , , , , , </u>	<u> </u>		

### **CREATE YOUR SHINY APP**

library(shiny)

# Define UI ----

ui <- fluidPage( )

# Define server logic ---server <- function(input, output) { }</pre>

# Run the app ----

shinyApp(ui = ui, server = server)

### ADD SOME LAYOUT

```
ui <- fluidPage(
titlePanel("title panel"),
sidebarLayout(
sidebarPanel("sidebar panel"),
mainPanel("main panel"))
```

### ADD SOME MORE LAYOUT

```
ui <- fluidPage(
titlePanel("My Shiny App"),
sidebarLayout(
           sidebarPanel(),
           mainPanel(
                      h1("First level title", align = "center"),
                      h2("Second level title"),
                      h3("Third level title"),
                      h4("Fourth level title"),
                      h5("Fifth level title"),
```

```
h6("Sixth level title"))))
```

### **TEXT FORMATTING**

```
ui <- fluidPage(
titlePanel("My Shiny App"),
sidebarLayout( sidebarPanel(),
            mainPanel(
            p("p creates a paragraph of text."),
            strong("strong() makes bold text."),
            em("em() creates italicized (i.e, emphasized) text."),
            br(),
            code("code displays your text similar to computer code"),
div("div creates segments of text with a similar style. This division of text is all blue because I passed the argument 'style = color:blue' to div", style = "color:blue") )))
```

**CONTROL WIDGETS** 

	Basic widgets			
Buttons	Buttons	Single checkbox	Checkbox group	Date input
Chackbox	Submit		Choice 2 Choice 3	2014-01-01
Спескрох	Date range	File input	Help text	Numeric input
File input	2017-06-21 to 2017-06-21	Browse No file selected	Note: help text isn't a true widget, but it provides an easy	1
Select box			way to add text to accompany other widgets.	
Sliders	Radio buttons	Select box	Sliders	Text input
Text or numeric input	<ul><li>Choice 1</li><li>Choice 2</li><li>Choice 3</li></ul>	Choice 1	0 50 100 0 10 20 30 40 50 60 70 80 90 100	Enter text
•••			0 25 75 100 0 10 20 30 40 50 60 70 80 90 100	

https://shiny.rstudio.com/tutorial/written-tutorial/lesson3/

### ADD SOME WIDGETS

```
sidebarPanel(
    helpText("some help text"),
    selectInput(inputId = "sample",
        label = "Select sample:", choices = c("a","b","c")),
    sliderInput(inputId = "cutoff",
        label = "Select a threshold:", min = 0, max = 255, value = 10)
    )
```

Output function	Creates
dataTableOutput	DataTable
htmlOutput	raw HTML
imageOutput	image
plotOutput	plot
tableOutput	table
textOutput	text
ui0utput	raw HTML
verbatimTextOutput	text

## **DISPLAY REACTIVE OUTPUT**

Step 1: Add an R object to the UI

mainPanel( textOutput("selected\_cutoff") )

Step 2: Provide R code to build the object (happens inside `server`).

```
server <- function(input, output) {
    output$selected_cutoff <- renderText({ paste("You have selected", input$cutoff) })
    }
</pre>
```

### HEADS AND TAILS

# Define UI ----

```
ui <- fluidPage(
```

```
sidebarLayout(
```

```
sidebarPanel(sliderInput(inputId = "n", label = "No of flips:", min = 10, max = 1000, value = 10),
```

```
sliderInput(inputId = "prob", label = "Success rate:", min = 0, max = 1,
value = 0.5)
),
mainPanel( plotOutput(outputId = "bars") )
))
```

### **HEADS AND TAILS: OUTPUT**

#### **Testing first:**

rbinom(n=25, size = 1, prob = 0.5)

barplot(table(rbinom(n=25, size = 1, prob = 0.5)))

#### Wrap it in the output:

output\$bars <- renderPlot({ barplot(table(rbinom(n=25, size = 1, prob =0.5))) })

#### Connect to dynamic input:

output\$bars <- renderPlot({ barplot(table(rbinom(n = input\$n,size = 1,prob = input\$prob)))
})</pre>

### **REACTIVE EXPRESSION**

data <- reactive(table(rbinom(input\$n, 1, input\$prob)))</pre>



### ADD REACTIVE EXPRESSION

# and update the server:

```
server <- function(input, output) {</pre>
```

```
data <- reactive(table(rbinom(input$n, input$size, input$prob)))
output$bars <- renderPlot({ barplot(data()) })
output$hist <- renderPlot({ hist(data()) })</pre>
```

### **EXAMPLE: MORE TESTING OF REACTIVE EXPRESSION**

library(shiny)

ui <- fluidPage(

sidebarLayout(

```
sidebarPanel(sliderInput(inputId = "n", label = "No. of coin flips", min = 10, max = 1000, value = 10),
```

```
sliderInput(inputId = "prob", label = "Success of rate", min = 0, max = 1, value = 0.5) ),
```

```
mainPanel( plotOutput(outputId = "xxxxx"),
```

```
plotOutput(outputId = "aaa"),
```

```
plotOutput(outputId = "ccccc") ) )
```

#### )

```
server <- function(input, output, session) {</pre>
```

```
call_it_something <- reactive(table(rbinom(n=input$n, size = 1, prob = input$prob))) # values are drawn just once and saved in this reactive expression
```

```
output$xxxxx <- renderPlot({ barplot(call_it_something() ) })</pre>
```

```
output$aaa <- renderPlot({ barplot(call_it_something() ) })</pre>
```

```
output$ccccc <- renderPlot({barplot(table(rbinom(n=input$n, size = 1, prob = input$prob))) }) # since we do not use reactive expression, values will be drawn on the fly and the result will be different from the two above
```

}

shinyApp(ui, server)

# Example from: https://www.r-bloggers.com/2019/12/r-shiny-for-beginners-annotated-starter-code/

### HEADS AND TAILS: ESTHETICS

```
output$bars <- renderPlot({</pre>
    flips <- tibble(flips = rbinom(inputn, 1, inputprob)) %>%
    mutate(flips = if else(flips == 1, "Heads", "Tails"))
  flips %>%
    count(flips) %>%
    ggplot(aes(flips, n, fill = flips)) +
    qeom col() +
    geom_label(aes(flips, n, label = n), size = 5) +
    theme(legend.position = "none",
        axis.text = element_text(size = 15)) +
    labs(x = "", y = "") +
    ggtitle(str_c("Results of ", input$n,
              " flips with Heads probability ",
              sprintf("%.2f", input$prob)))
   })
```

library(dplyr) library(ggplot2) library(stringr) library(tibble)

# (1) SHARE YOUR APP VIA GITHUB

Host your code on GitHub: <u>https://github.com/rstudio/shiny\_example/</u> (repository must be public) and run your app from within R using runGitHub() or runUrl().

#### Example:

shiny::runGitHub("shiny\_example", "rstudio")
shiny::runUrl('https://github.com/rstudio/shiny\_example/archive/main.tar.gz')

# Hosted on my GitHub account: https://github.com/kmt555/MyShinyApps shiny::runGitHub(repo = "MyShinyApps", username = "kmt555", ref = "main")

# (2) SHARE YOUR APP WITH SHINYAPPS.IO

Help	Account: kmt567	
	A Profile	5 ne R console.
	Authentication	Show Secret
	🔂 Tokens	Show Secret
	亡 Log out	Copy to clipboard
		e R console.
		Hide secret
	T Add Ken	

### **CONNECT YOUR RSTUDIO WITH SHINYAPPS.IO**

### Run App • • = • Publish Application... Manage Accounts...

7







### SHINYAPPS.IO ...

Connect Accou	nt
Back	Connect ShinyApps.io Account
	Go to <u>your account on ShinyApps</u> and log in.
	Click your name, then choose <b>Tokens</b> from your account menu.
	Click <b>Show</b> on the token you want to use, then <b>Show Secret</b> and <b>Copy to Clipboard.</b> Paste the result here:
	Copy-paste your token here!
	Need a ShinyApps.io account? <u>Get started here.</u>



#### Publish to Server Publish Files From: Publish From Account: Add New Account ~/Downloads/ShinyApps/heads\_tails **kmt567:** shinyapps.io 🖌 🛛 app.R Title: heads\_tails ✓ Launch browser Public Cancel

## **ACTION BUTTONS**

library(shiny)

```
ui <- fluidPage(
actionButton(inputId = "clicks", label = "Click me") )</pre>
```

```
server <- function(input, output) {
observeEvent(input$clicks, { print(as.numeric(input$clicks)) })
}</pre>
```

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

### **USE ACTION BUTTONS TO DELAY REACTIONS**

```
library(shiny)
```

```
ui <- fluidPage(
```

```
sliderInput(inputId = "num", label = "Choose a number", value = 25, min = 1, max =
100),
```

```
plotOutput("hist") )
```

```
server <- function(input, output) {</pre>
```

```
output$hist <- renderPlot({ hist(rnorm(input$num))
}) }</pre>
```

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

### ADD BUTTON

```
library(shiny)
ui <- fluidPage(
sliderInput(inputId = "num", label = "Choose a number", value = 25, min = 1, max =
100),</pre>
```

```
actionButton(inputId = "go", label = "Update"), plotOutput("hist") )
```

```
server <- function(input, output) {
data <- eventReactive(input$go, { })
output$hist <- renderPlot({ hist(rnorm(input$num))
}) }</pre>
```

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

#### server <- function(input, output) {</pre>

```
data <- eventReactive(input$go, { rnorm(input$num) })
output$hist <- renderPlot({ hist(data())
})
})</pre>
```

# REACTIVEVALUES()

library(shiny)

ui <- fluidPage(</pre>

```
actionButton(inputId = "norm", label = "Normal"),
actionButton(inputId = "unif", label = "Uniform"),
plotOutput("hist") )
```

```
server <- function(input, output) {</pre>
```

```
rv <- reactiveValues(data = rnorm(100))
observeEvent(input$norm, { rv$data <- rnorm(100) })
observeEvent(input$unif, { rv$data <- runif(100) })</pre>
```

```
output$hist <- renderPlot({ hist(rv$data) }) }</pre>
```

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

