UNVEILING THE POWER OF R SHINY DASHBOARDS

Jehangeer Aswani

ABOUT ME

I am Jehangeer, a recent statistics graduate from Quaid-e-Azam University.

I offer my services as a professional R & Shiny freelancer on the Upwork platform.

Contact me:
www.linkedin.com/in/aswanijehangeer/

AGENDA

We will talk in this presentation about:

- What is Shiny?
- What You Need to Get Started.
- Structure/Anatomy of Shiny and Hello Shiny App.
- Shiny Inputs & Outputs.
- Understanding Reactivity and Reactive Values.
- Sharing Apps - Deployment.
- What's Next - (Shiny Learning Resources).
- **Ask questions in the chat throughout!**

TALK RESOURCES

You can find all the material we'll use for this presentation in this GitHub repository.

GitHub: github.com/aswanijehangeer/Unveiling-the-Power-of-RShiny-Dashboards

WHAT IS SHINY?

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

It allows users to convert R code into web applications without needing expertise in web development languages like HTML, CSS, or JavaScript.

WHAT YOU NEED TO GET STARTED IN SHINY?

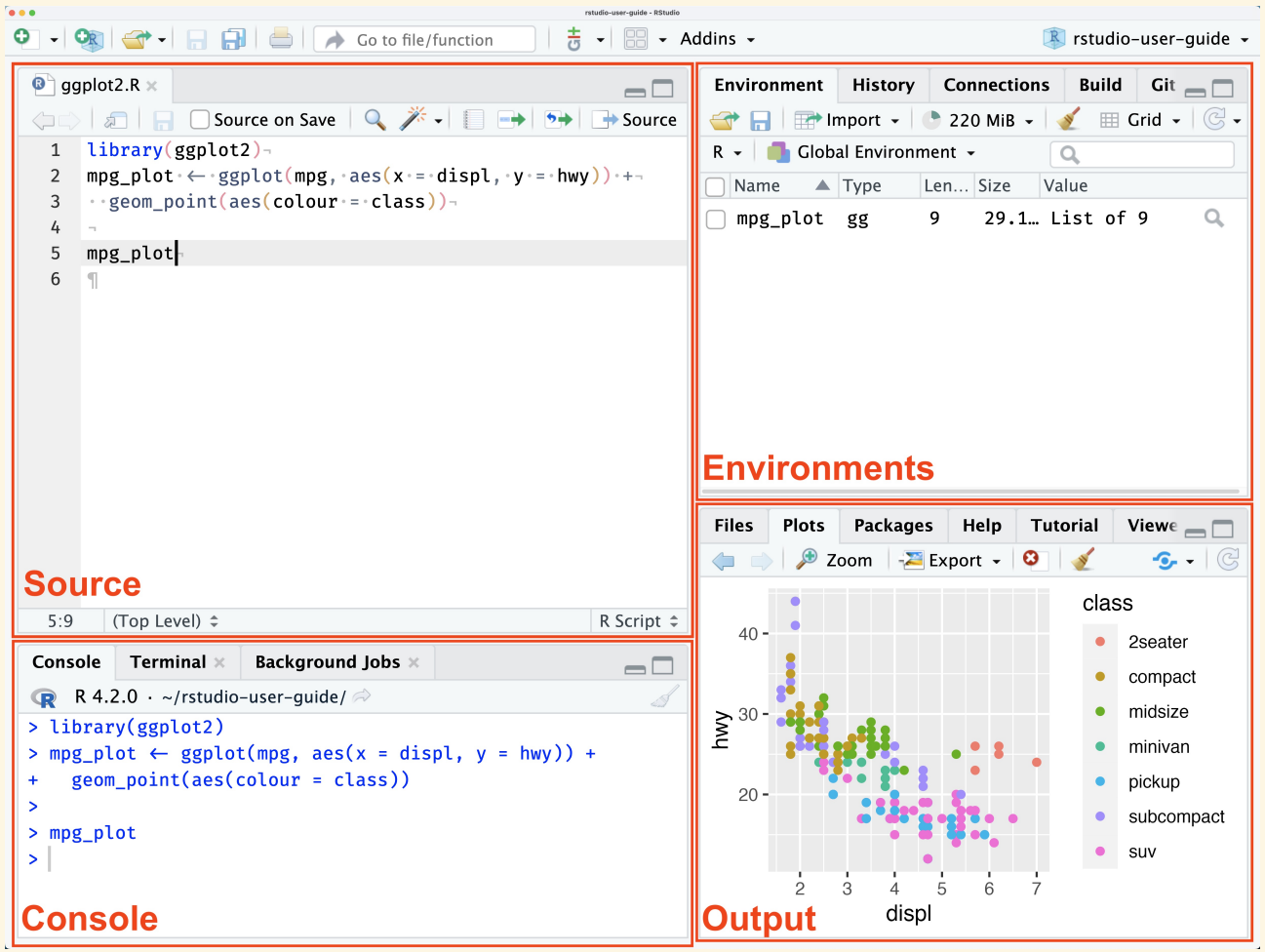
You just need **R** and **RStudio (IDE)** to get started in Shiny.

R is a free, open-source programming language that is used for data mining, statistical analysis, data visualization, and machine learning.
Download R here: cran.r-project.org/

RStudio is a free, open-source, integrated development environment (IDE) for the R programming language.
Download RStudio here: posit.co/download/rstudio-desktop/

RSTUDIO INTERFACE

This is how your rstudio interface should look like after complete installation.



STRUCTURE OF A SHINY APP

Shiny apps can be developed in a single script named `app.R`. `app.R` has three components:

- a user interface (UI) object.
- a server function.
- a call to the `shinyApp` function.

The `ui` object shapes the app's look and feel, while the server function holds the app's instructions for execution. The `shinyApp` function combines the UI and `server` logic to create the Shiny app object.

STRUCTURE OF A SHINY APP CONT...

However, for larger applications, splitting the code into two separate files, namely `ui.R` and `server.R`, is a good option.

This separation allows for better organization and management of the user interface (UI) elements in `ui.R` and the server-side logic in `server.R`.

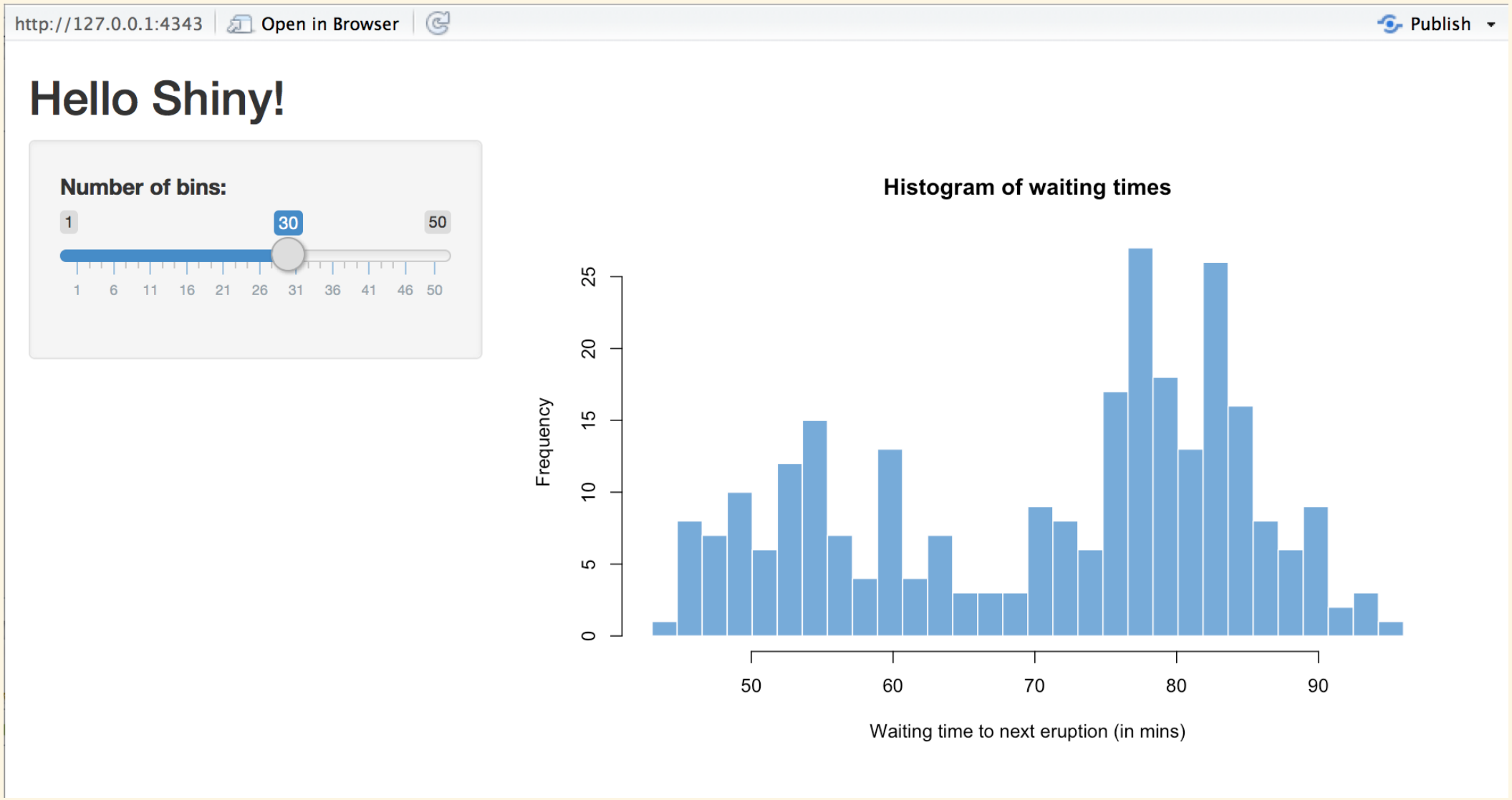
ANATOMY OF A SHINY APP

Here is a code template for an empty app.

Explain

```
# Loading Package ----
library(shiny)

# Define UI ----
ui <- fluidPage(

)

# Define server logic ----
server <- function(input, output) {

}

# Run the app ----
shinyApp(ui = ui, server = server)
```

LET'S BUILD A SIMPLE HELLO SHINY APP



SIMPLE HELLO SHINY APP - USER INTERFACE

Here is the `ui` object for the Hello Shiny App.

Explain

```r
library(shiny)

# Define UI for app that draws a histogram ----
ui <- fluidPage(

  # App title ----
  titlePanel("Hello Shiny!"),

  # Sidebar layout with input and output definitions ----
  sidebarLayout(

    # Sidebar panel for inputs ----
    sidebarPanel(

      # Input: Slider for the number of bins ----
      sliderInput(inputId = "bins",
                  label = "Number of bins:",
                  min = 1,
                  max = 50
```

www.aswanijehangeer.com

SIMPLE HELLO SHINY APP - SERVER

Here is the `server` function for the Hello Shiny App.

Explain

```
# Define server logic required to draw a histogram ----
server <- function(input, output) {

  # Histogram of the Old Faithful Geyser Data ----
  # with requested number of bins.
  # This expression that generates a histogram is wrapped in a call
  # to renderPlot to indicate that:
  #
  # 1. It is "reactive" and therefore should be automatically
  #    re-executed when inputs (input$bins) change.
  # 2. Its output type is a plot.
  output$distPlot <- renderPlot({

    x    <- faithful$waiting
    bins <- seq(min(x), max(x), length.out = input$bins + 1)

    hist(x, breaks = bins, col = "#007bc2", border = "white",
         xlab = "Waiting time to next eruption (in mins)",
         main = "Histogram of waiting times")
```

www.aswanijehangeer.com

RUNNING AN APP

Every Shiny app has the same structure: an `app.R` file that contains ui and server. You can create a Shiny app by making a new directory and saving an `app.R` file inside it.

> **Optional caption (note)**
>
> It is recommended that each app will live in its own unique directory.

You can run a Shiny app by giving the name of its directory to the function `runApp`.

For example if your Shiny app is in a directory called `my_app`, run it with the following code:

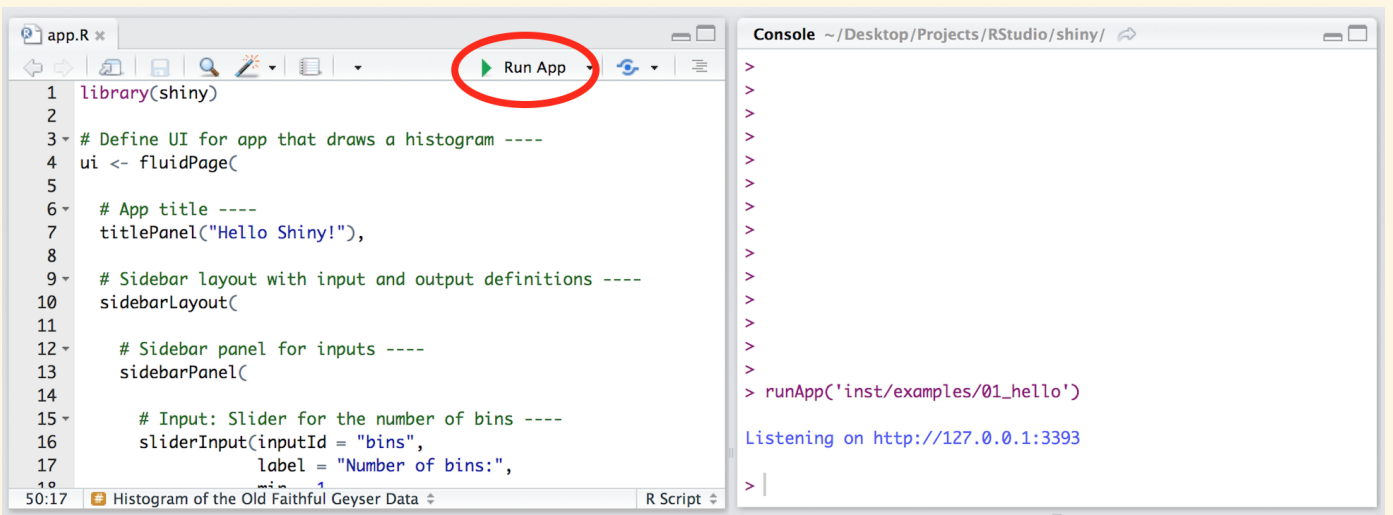
```
1  runApp("my_app")
```

www.aswanijehangeer.com

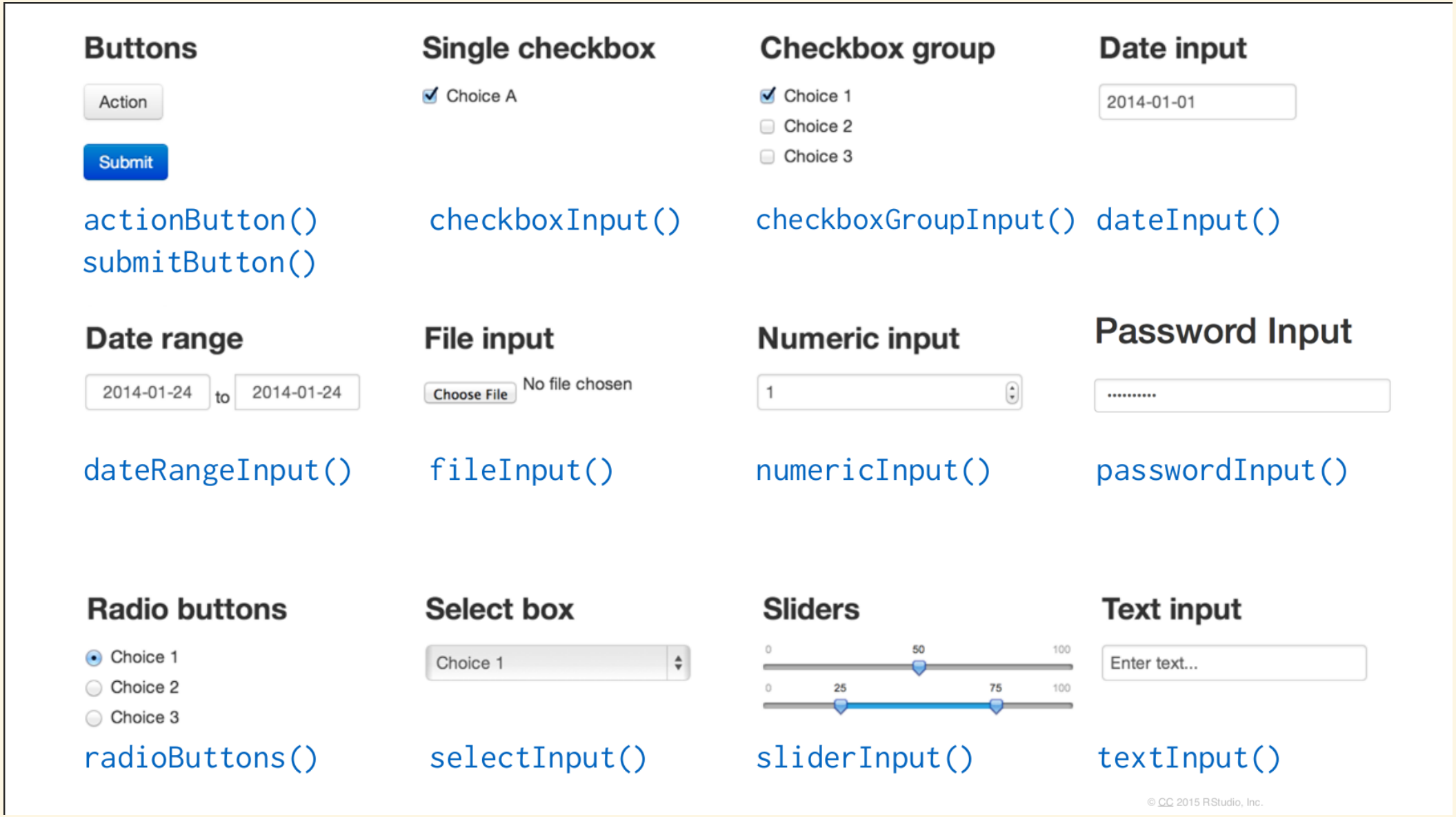
RELAUNCHING AN APP

To relaunch your Shiny app:

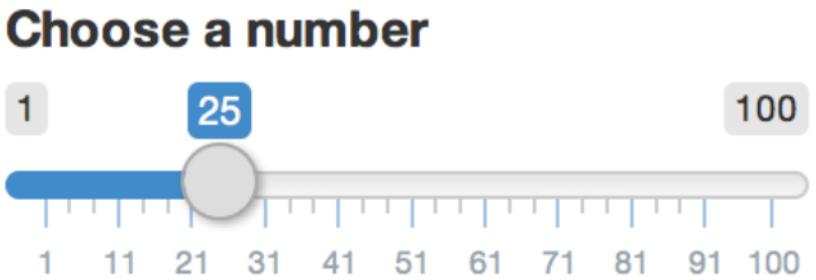
- Run runApp("myapp"), or
- Open the app.R script in your RStudio editor. RStudio will recognize the Shiny script and provide a Run App button (at the top of the editor).



THERE ARE MORE THAN 20 INPUTS IN SHINY



SHINY INPUT SYNTAX




```
sliderInput(inputId = "num", label = "Choose a number", …)
```

input name
(for internal use)

Notice:
Id not ID

label to display

input specific arguments

```
?sliderInput
```

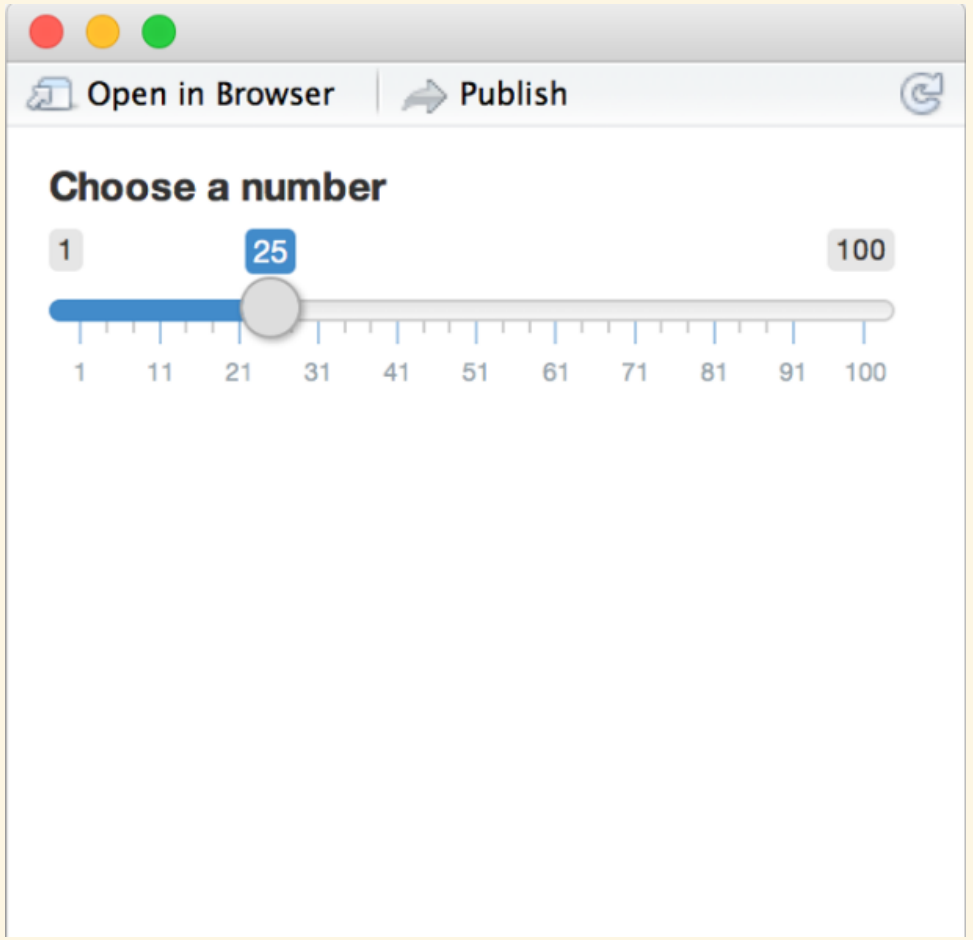
© CC 2015 RStudio, Inc.

SHINY INPUT - EXAMPLE

Create an input with an *Input() function.

Explain

```
1  library(shiny)
2
3  ui <- fluidPage(
4    sliderInput(
5      inputId = "num",
6      label = "Choose a number",
7      value = 25, min = 1, max = 100
8    )
9  )
10 server <- function(input, output)
11
12 }
13
14 shinyApp(ui = ui, server = server)
```



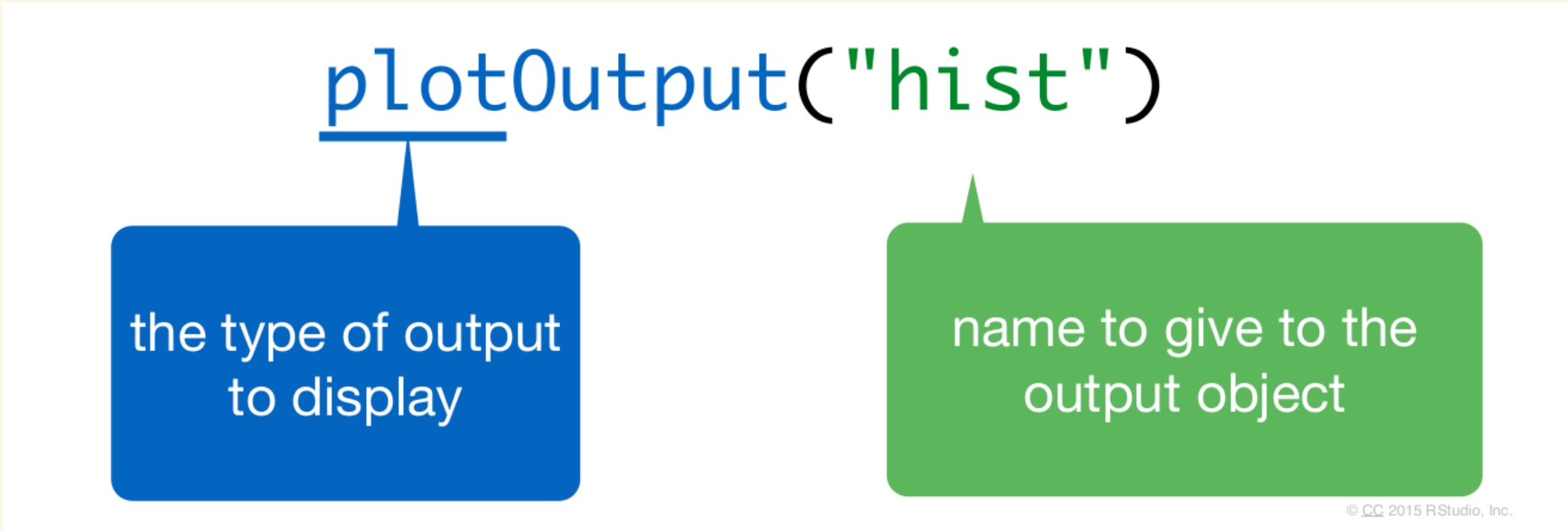
SHINY OUTPUTS

Function	Inserts
`dataTableOutput()`	an interactive table
`htmlOutput()`	raw HTML
`imageOutput()`	image
`plotOutput()`	plot
`tableOutput()`	table
`textOutput()`	text
`uiOutput()`	a Shiny UI element
`verbatimTextOutput()`	text

© CC 2015 RStudio, Inc.

SHINY OUTPUT SYNTAX

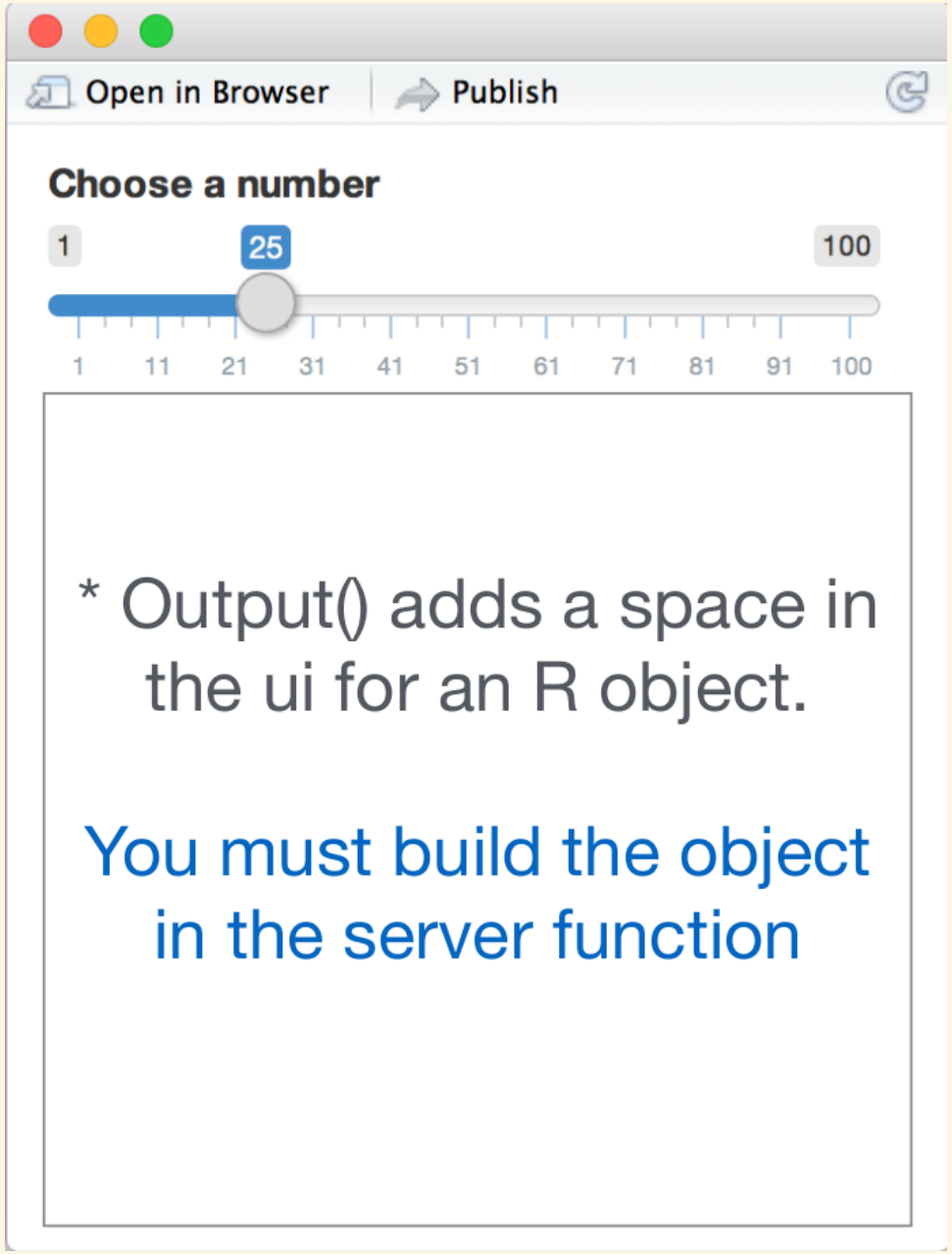
To display output, add it to `fluidPage()` with an `*Output()` function.



SHINY OUTPUT - EXAMPLE

Explain

```
1  library(shiny)
2
3  ui <- fluidPage(
4    sliderInput(
5      inputId = "num",
6      label = "Choose a number",
7      value = 25, min = 1, max = 100
8    ),
9    plotOutput("hist")
10 )
11 server <- function(input, output)
12
13 }
14
15 shinyApp(ui = ui, server = server)
```



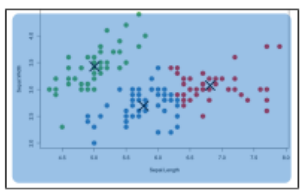
QUICK RECAP - WHAT WE HAVE LEARN

```
library(shiny)
ui <- fluidPage()
server <- function(input, output) {}
shinyApp(ui = ui, server = server)
```

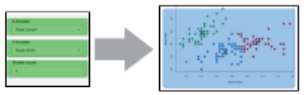
Begin each app with the template

Add elements as arguments to **fluidPage()**

Create reactive inputs with an ***Input()** function



Display reactive results with an ***Output()** function



Assemble outputs from inputs in the server function

© CC 2015 RStudio, Inc.

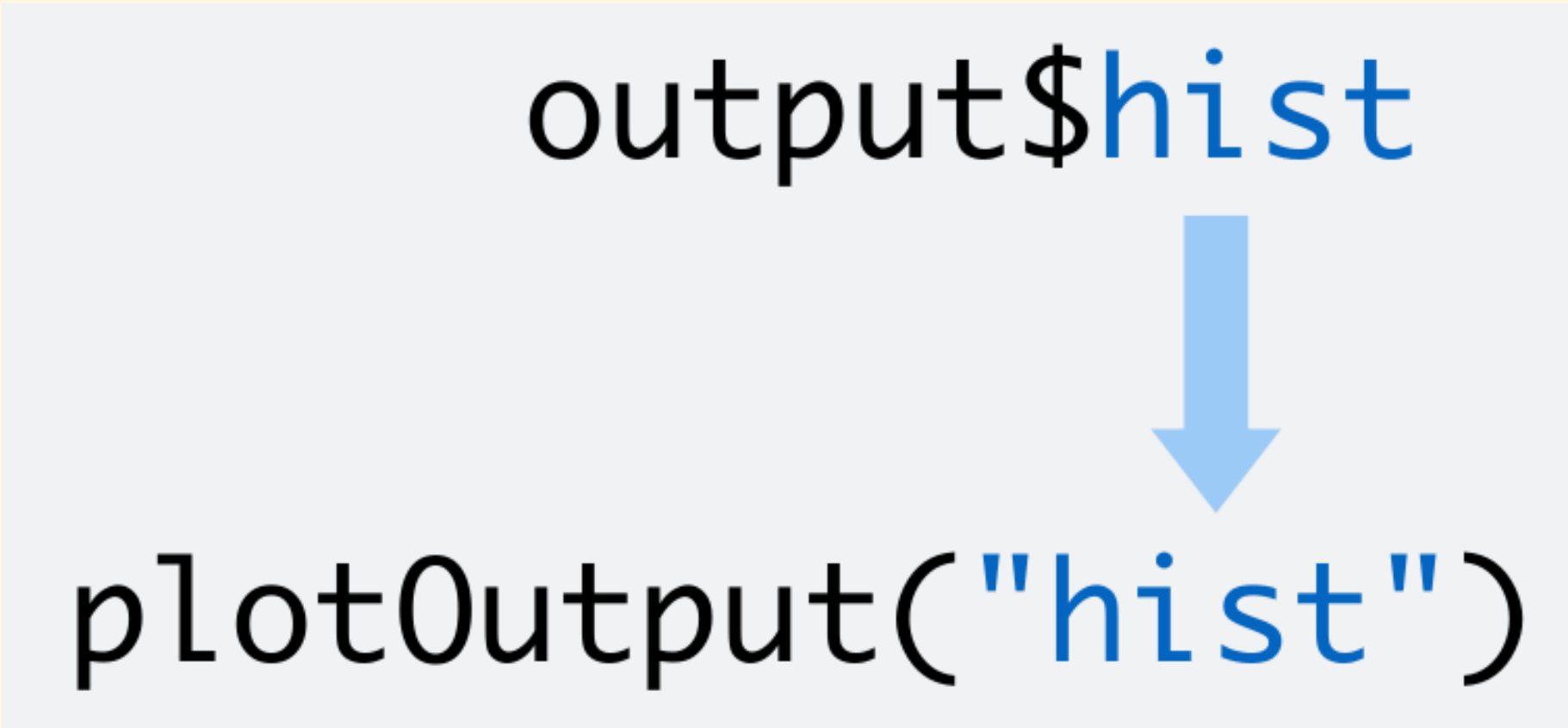
SERVER: HOW TO ASSESMBE INPUTS INTO OUTPUTS?

To write a server function, there are three basic rules. If you follow the three rules, the output will automatically update.

1. SAVE OBJECTS TO DISPLAY TO OUTPUT$.

Explain

```
server <- function(input, output)
{
        output$hist <- #code
}
```



2. BUILD OBJECTS TO DISPLAY WITH RENDER*()

Explain

```
server <- function(input, output) {
  output$hist <- renderPlot({
    title <- "100 random normal values"
    hist(rnorm(100), main = title)
  })
}
```

function	creates
renderDataTable()	An interactive table (from a data frame, matrix, or other table-like structure)
renderImage()	An image (saved as a link to a source file)
renderPlot()	A plot
renderPrint()	A code block of printed output
renderTable()	A table (from a data frame, matrix, or other table-like structure)
renderText()	A character string
renderUI()	a Shiny UI element

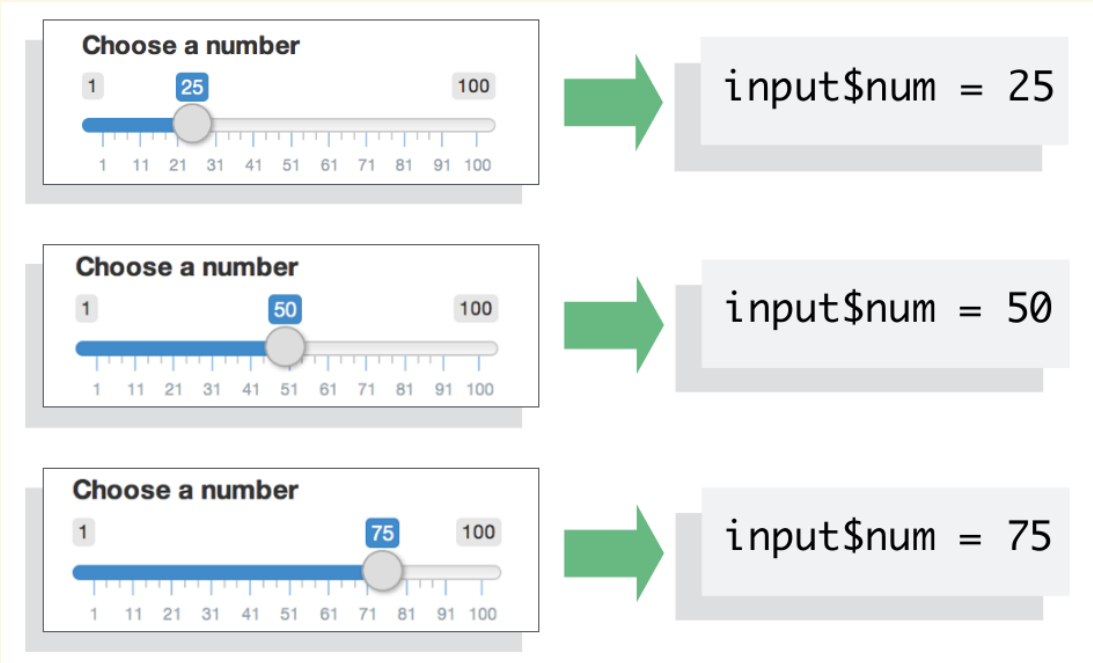
© CC 2015 RStudio, Inc.

3. ACCESS INPUT VALUES WITH INPUT$

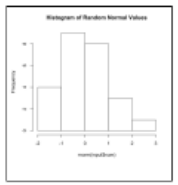
Explain

```
1  server <- function(input, output) {
2    output$hist <- renderPlot({
3      hist(rnorm(input$num))
4    })
5  }
```

The input value changes whenever a user changes the input.



QUICK RECAP: RENDERS



render*() functions make **objects to display**

output$** Always save the result to **output$

```
output$hist <- renderPlot({
  hist(rnorm(input$num),
    main = input$title)
})
```

render*() makes an observer object that has a **block of code** associated with it

```
renderPlot(  { hist(rnorm(input$num)) })
```

The object will **rerun the entire code block** to update itself whenever it is invalidated

© CC 2015 RStudio, Inc.

QUICK RECAP: SERVER

Use the server function to assemble inputs into outputs. Follow 3 rules:

`output$hist <-`

1. Save the output that you build to **output$**

```
renderPlot({
  hist(rnorm(input$num))
})
```

2. Build the output with a **render*()** function

`input$num`

3. Access input values with **input$**

Create reactivity by using **Inputs** to build **rendered Outputs**

© CC 2015 RStudio, Inc.

WHAT IS REACTIVITY?

Reactivity in shiny refers to the automatic updating of outputs based on changes in inputs or data.

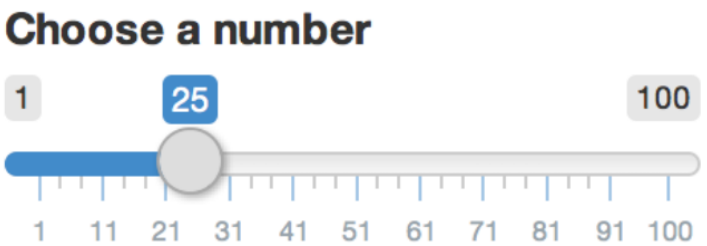
Reactivity automatically occurs whenever we use an input value to render an output object.

Explain

```
function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
}
```

WHAT IS A REACTIVE VALUE?

A reactive value is an object that causes reactive functions to re-execute when their value changes.
For example, user inputs to a Shiny application are reactive value objects.




```
sliderInput(inputId = "num", label = "Choose a number", …)
```

this input will provide a value saved as **input$num**

REACTIVE VALUES CONT...

Reactive values work together with reactive functions. We cannot call a reactive value from outside of one.

```
renderPlot({ hist(rnorm(100, input$num)) })
```

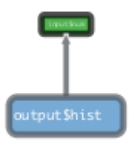
```
hist(rnorm(100, input$num))
```

QUICK RECAP: REACTIVE VALUES

Reactive values act as the data streams that flow through your app.

The **input** list is a list of reactive values. The values show the current state of the inputs.

You can only call a reactive value from a function that is designed to work with one



Reactive values notify. The objects created by **reactive functions respond.**

© CC 2015 RStudio, Inc.

SHARE YOUR APPS

When it comes to sharing Shiny apps, we have two basic options:

1. **Share your Shiny app as R script**. This is the simplest way to share an app, but it works only if your users have R on their own computer (and know how to use it).
2. **Share your Shiny app as a web page**. This is definitely the most user friendly way to share a Shiny app. Your users can navigate to your app through the internet with a web browser.

SHARE AS A WEB PAGE

Sharing Shiny apps as a web page can be accomplished through various methods:

- Shinyapps.io.
- Shiny Server.
- Posit Connect.

SHINYAPPS.IO

Posit's Shinyapps.io platform allows for easy deployment and sharing of Shiny applications. We can publish our apps directly from RStudio to Shinyapps.io.

This a shiny app that I have deployed on shinyapps.io this will be how a link looks like and you can share with anyone.

aswanijahangeer.shinyapps.io/chess_grandmasters_shinyapp/

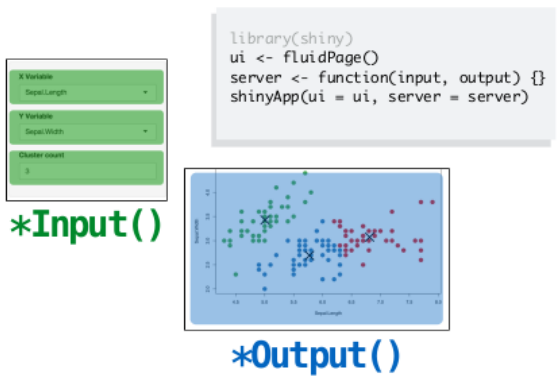
Github of this App: aswanijehangeer/Chess-Grandmaster-Shiny-App

SHINY SERVER & POSIT CONNECT

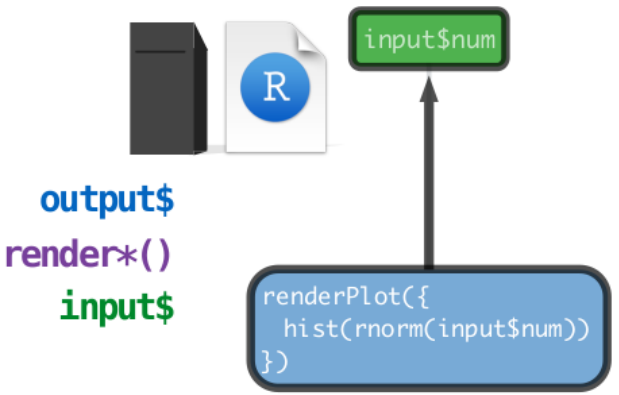
Shiny Server is a companion program to Shiny that builds a web server designed to host Shiny apps. It's free, open source, and available from GitHub.

For enterprise-level deployment and sharing, RStudio Connect provides a secure and managed platform to publish and share Shiny applications within organizations.

YOU KNOW NOW HOW TO ...



Build an app

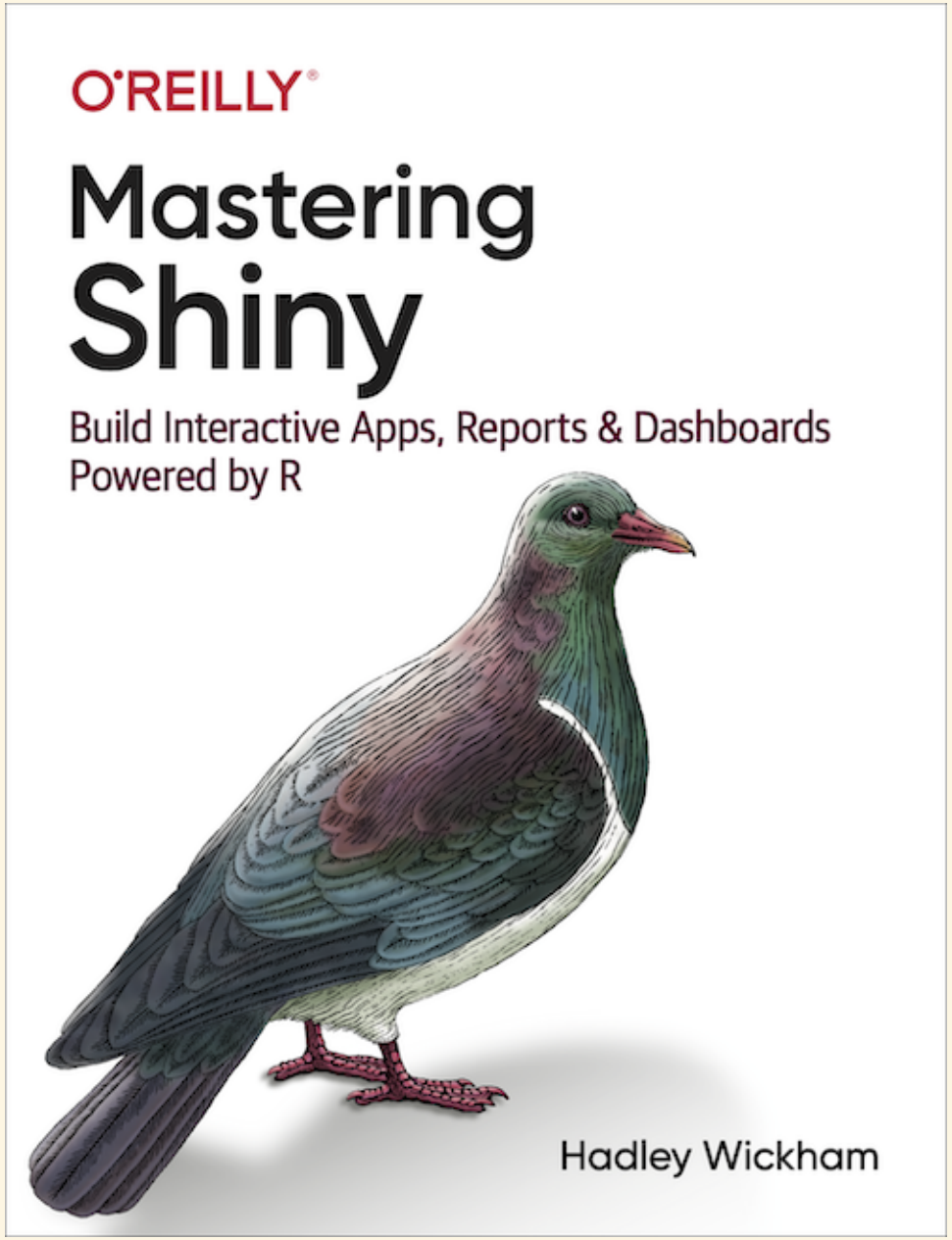


Create interactions

Share your apps

WHAT'S NEXT - SHINY LEARNING RESOURCES

- mastering-shiny.org
- Course developed by Chanin Nantasenamat (aka Data Professor)
- Publishing Visualizations in R with Shiny and flexdashboard
- Shiny Fundemantals Track with R - DataCamp
- **Ask:** RStudio Community



THANKS YOU!

Connect me here:

www.aswanijehangeer.com

www.linkedin.com/in/aswanijehangeer/