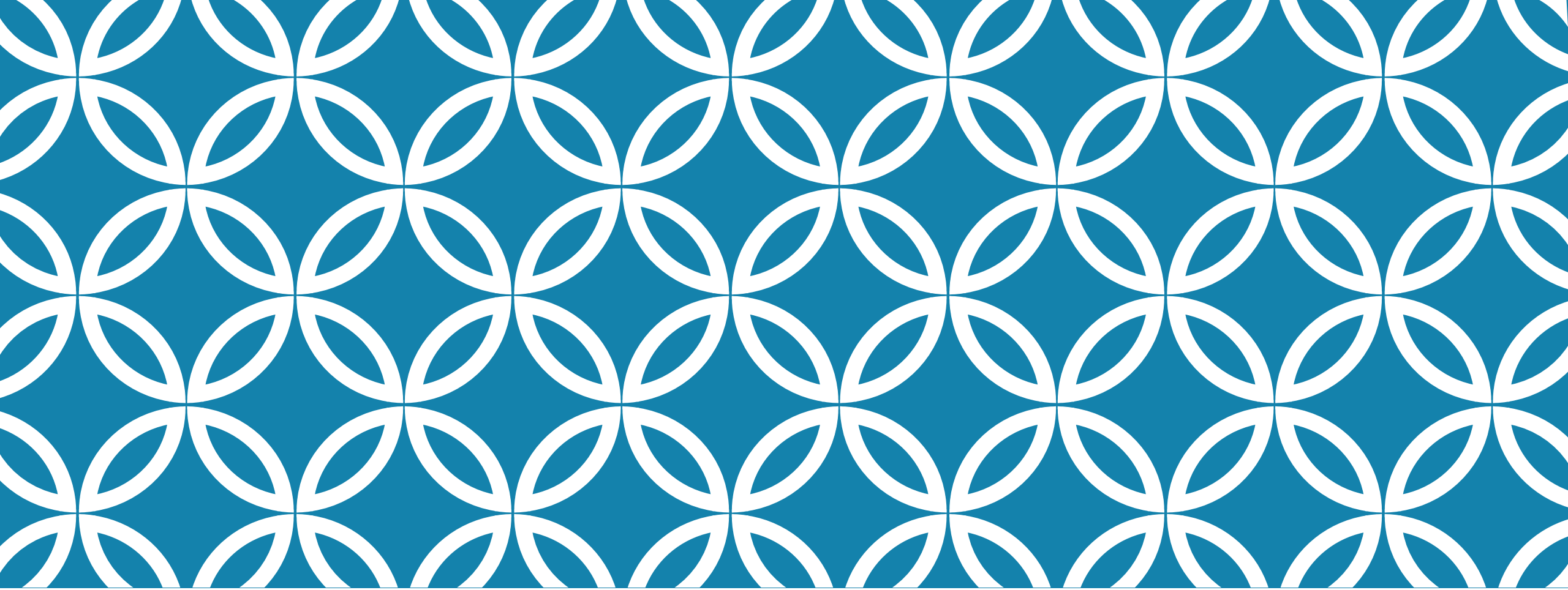


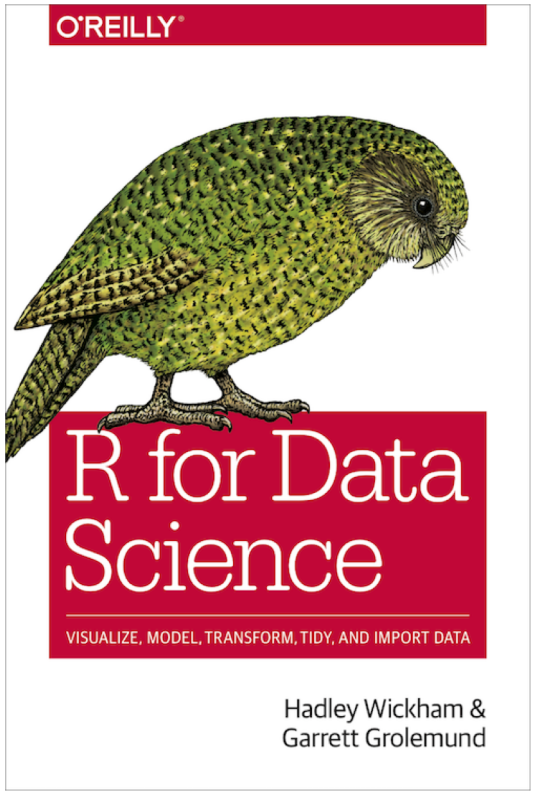
Shiny at CHOP
From Zero to Deploy in One Hour

Stephan Kadauke MD PhD
Dept of Pathology and Lab Medicine
CHOP R Users Group
2019-11-12

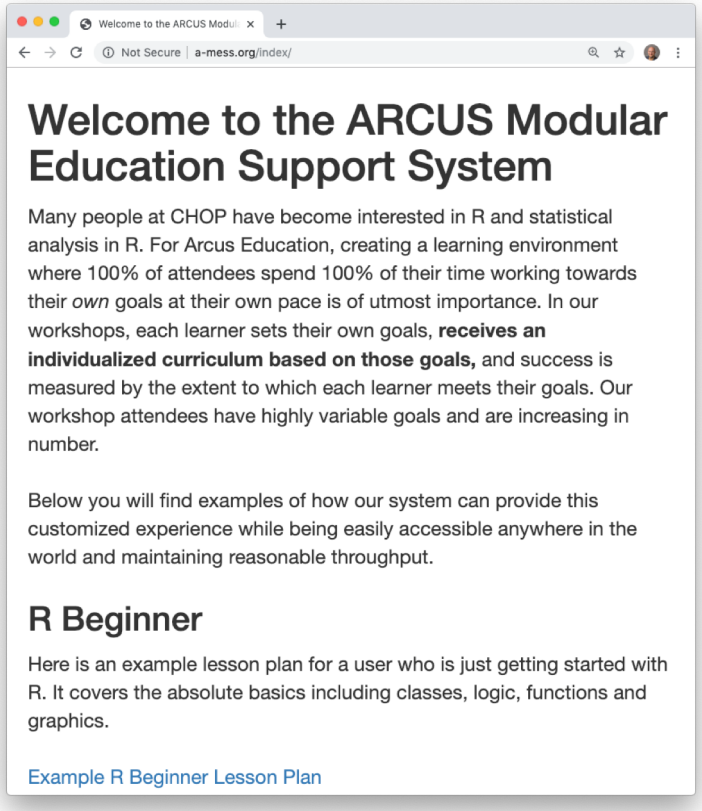
Assumptions

- You are familiar with **R programming** concepts such as **variables, functions, lists,** and **packages**
- You are familiar with the idea that **web apps** are hosted on a **web server** that sends **pages written in HTML** to your web browser
- You are interested in learning about **how you can use Shiny** to **build reactive web apps**

Learn R



r4ds.had.co.nz



a-mess.org

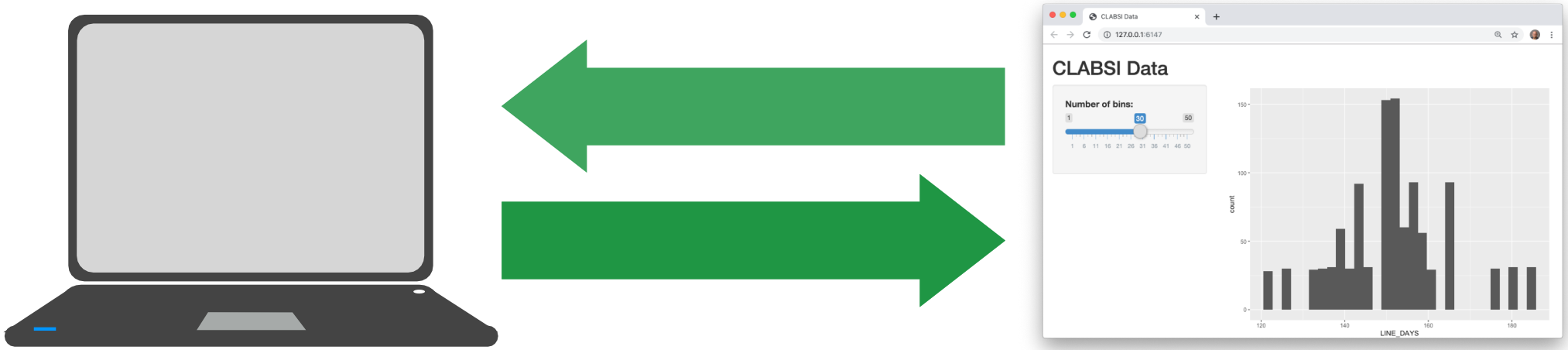
bit.ly/chop-r

Objectives

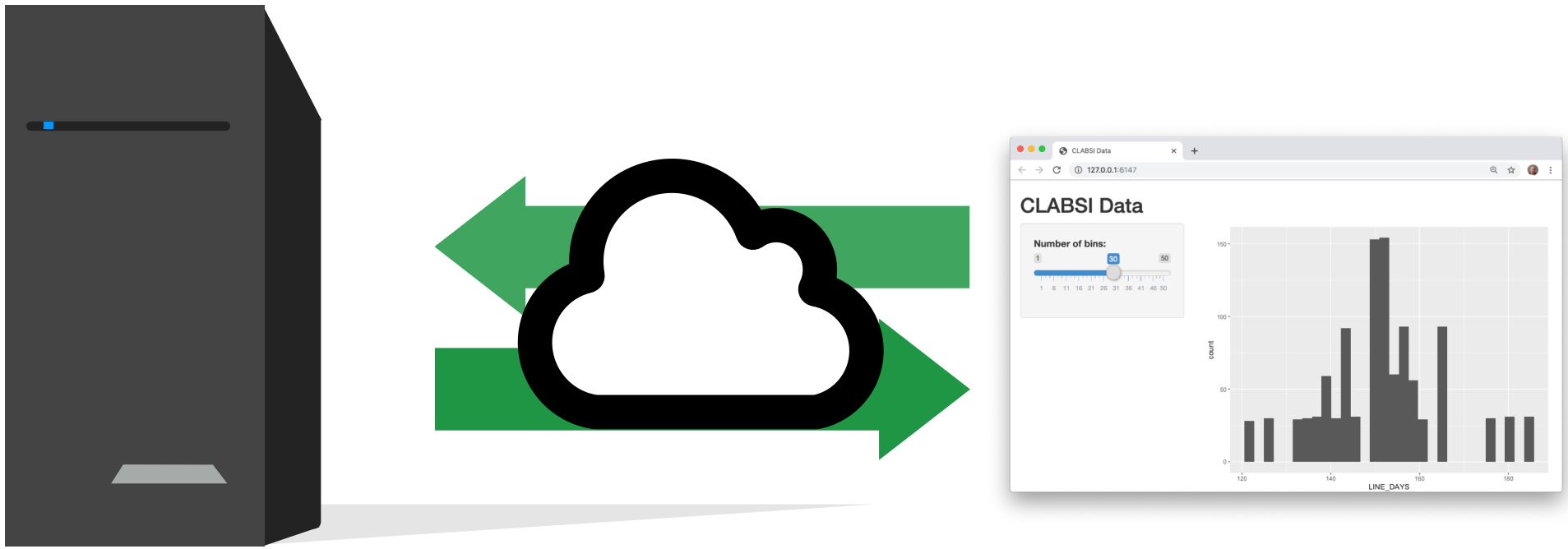
- Learn how to build a very basic Shiny app
- Deploy a Shiny app to CHOP's RStudio Connect server
- Style the app to look CHOP-branded

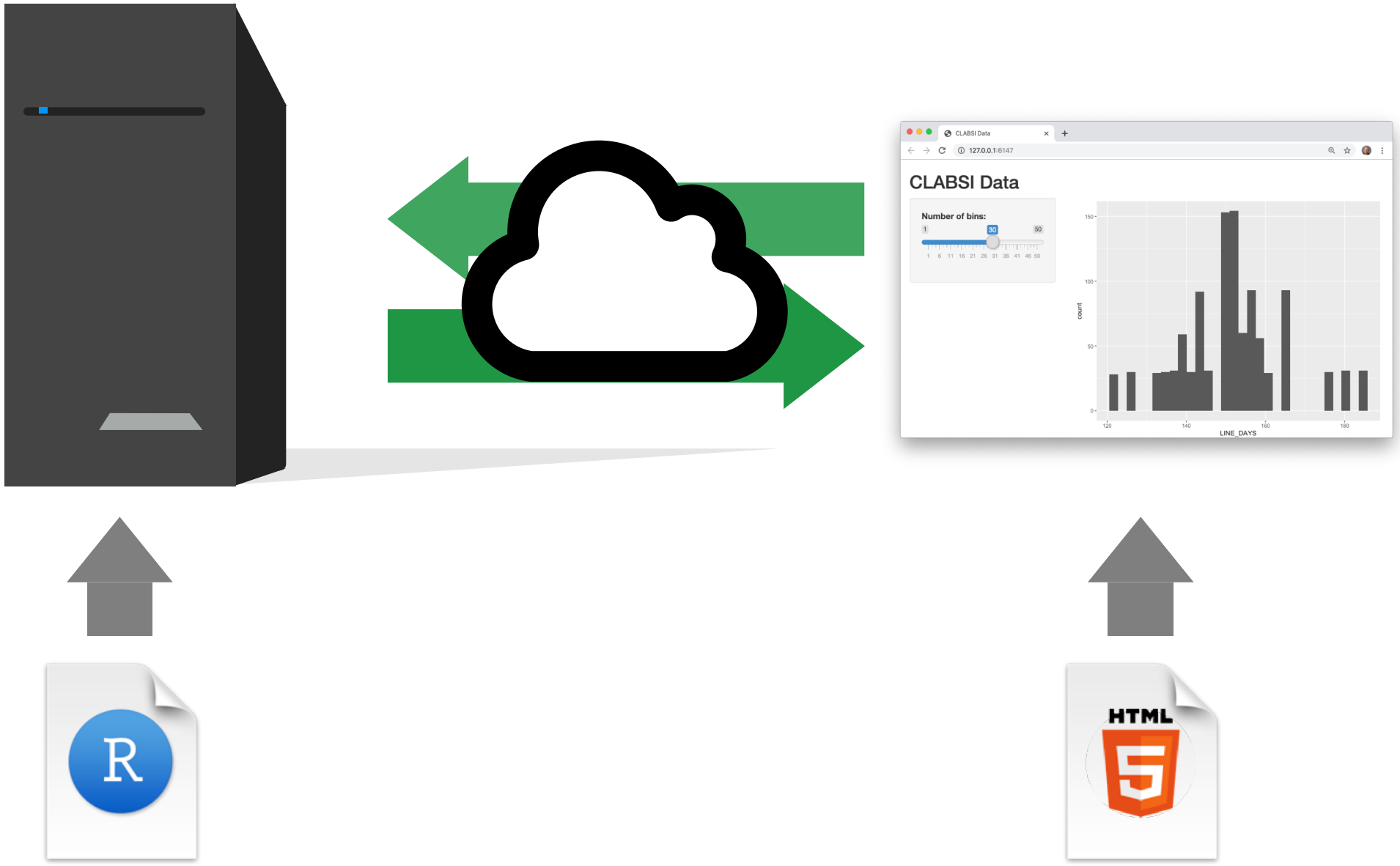
Understand the Architecture

Every Shiny app is maintained by a computer running R



Every Shiny app is maintained by a computer running R





Server Instructions

User Interface (UI)

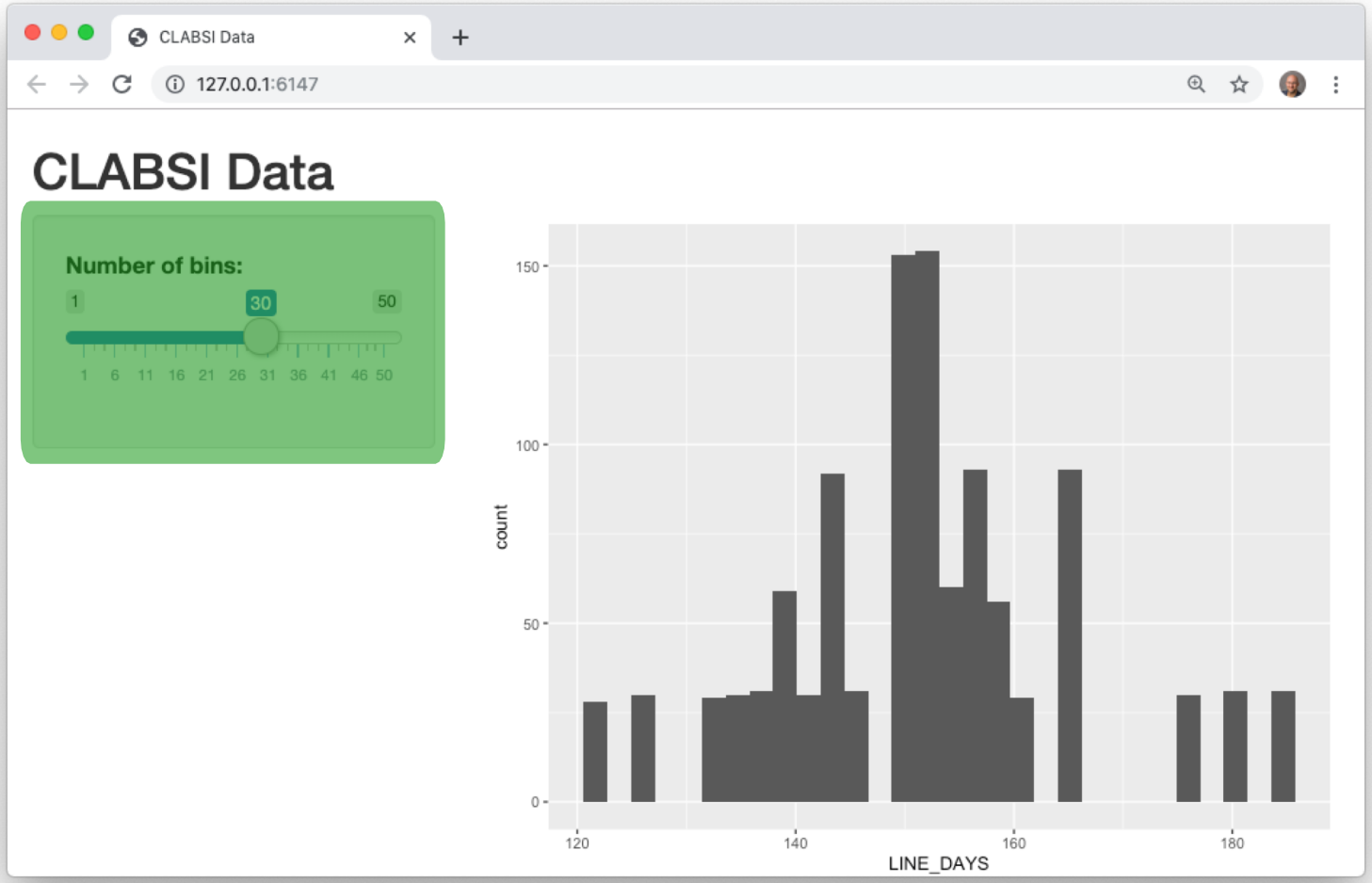
Use the Template

Shiny app template

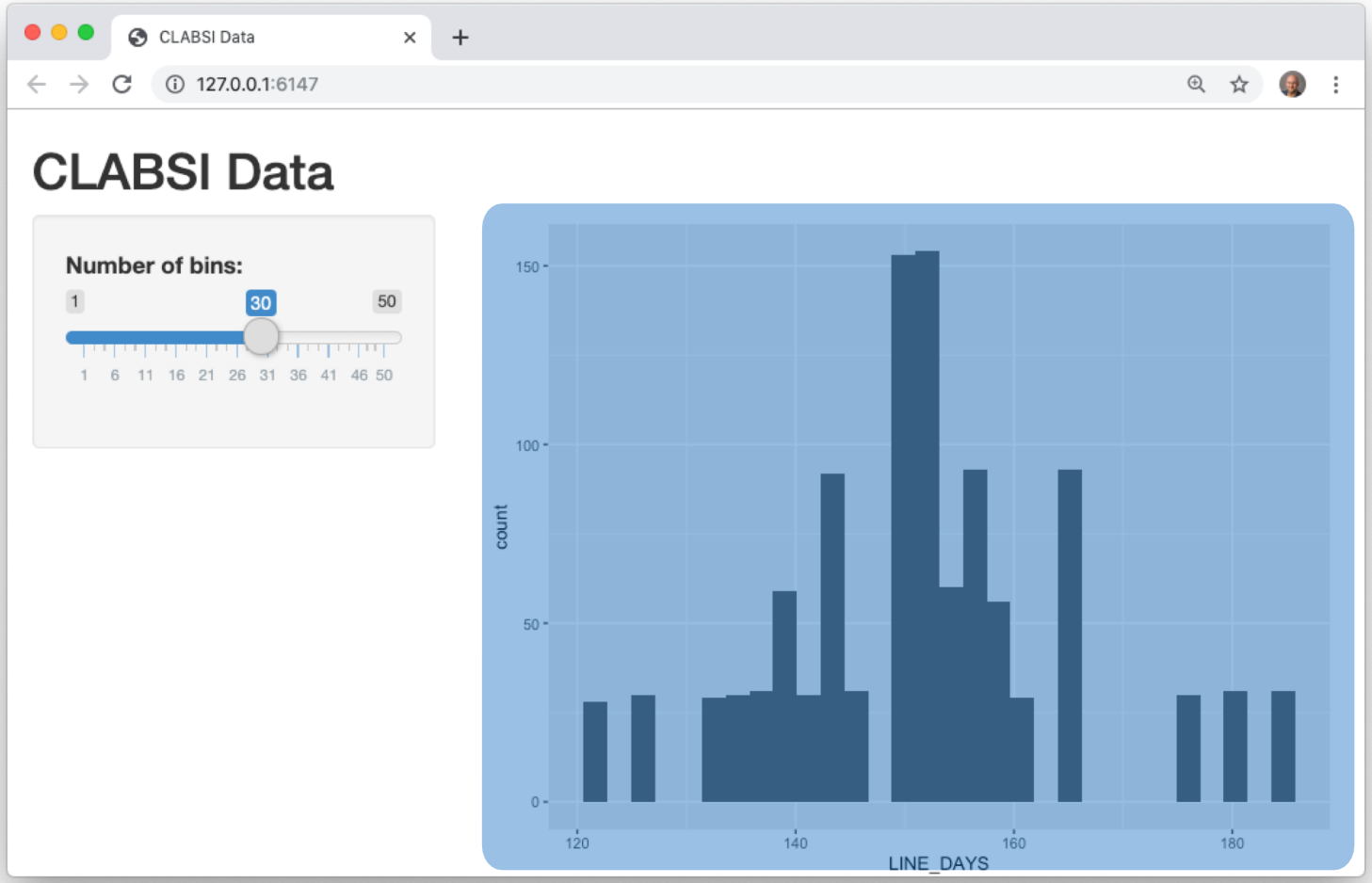
```
library(shiny)

ui <- fluidPage()

server <- function(input, output) {}

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

Build Your App Around Inputs and Outputs

Build your app around **inputs** and **outputs**



Build your app around **inputs** and **outputs**



Add elements to your app as arguments to `fluidPage()`

```
ui <- fluidPage(
  # *Input() functions,
  # *Output() functions
)
```

Inputs

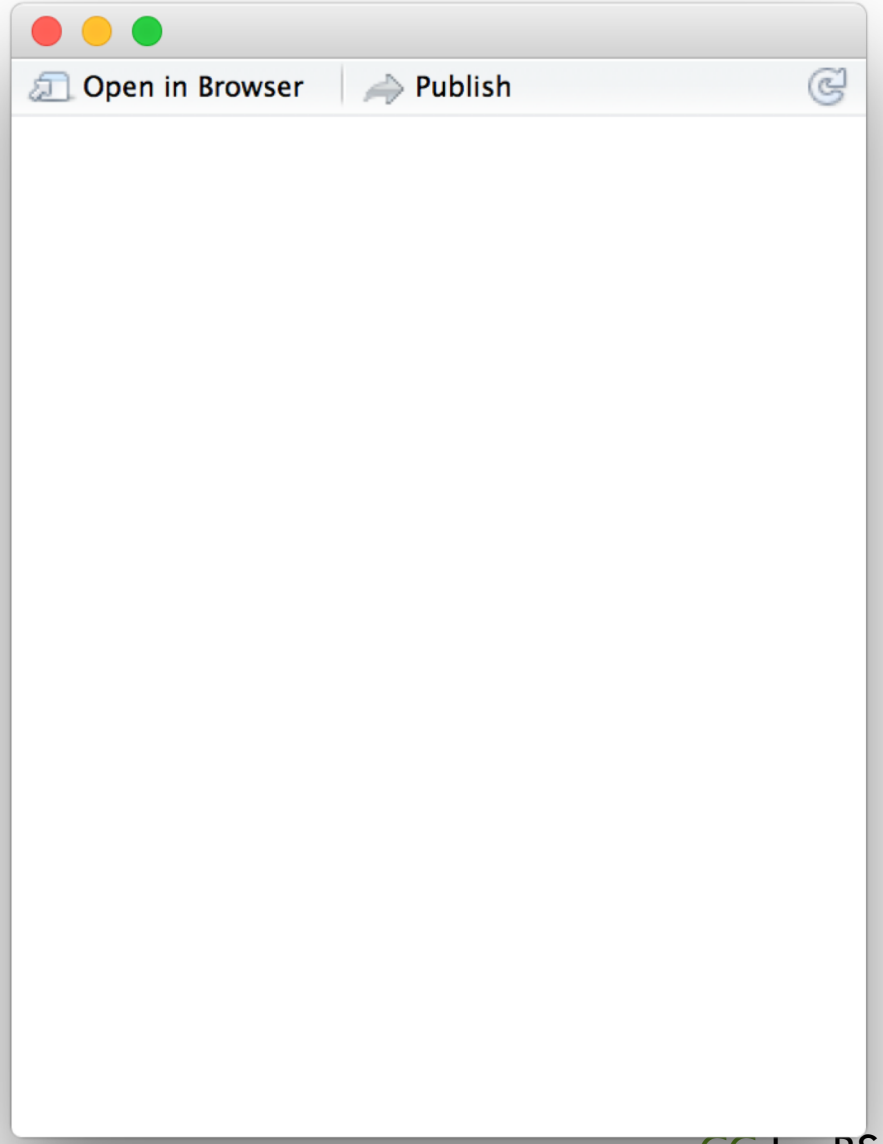
Create an input with an *Input() function.

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

```
<div class="form-group shiny-input-container">
    <label class="control-label" for="num">Choose a number</label>
    <input class="js-range-slider" id="num" data-min="1" data-max="100"
    data-from="25" data-step="1" data-grid="true" data-grid-num="9.9"
    data-grid-snap="false" data-prettify-separator="," data-keyboard="true"
    data-keyboard-step="1.01010101010101"/>
</div>
```

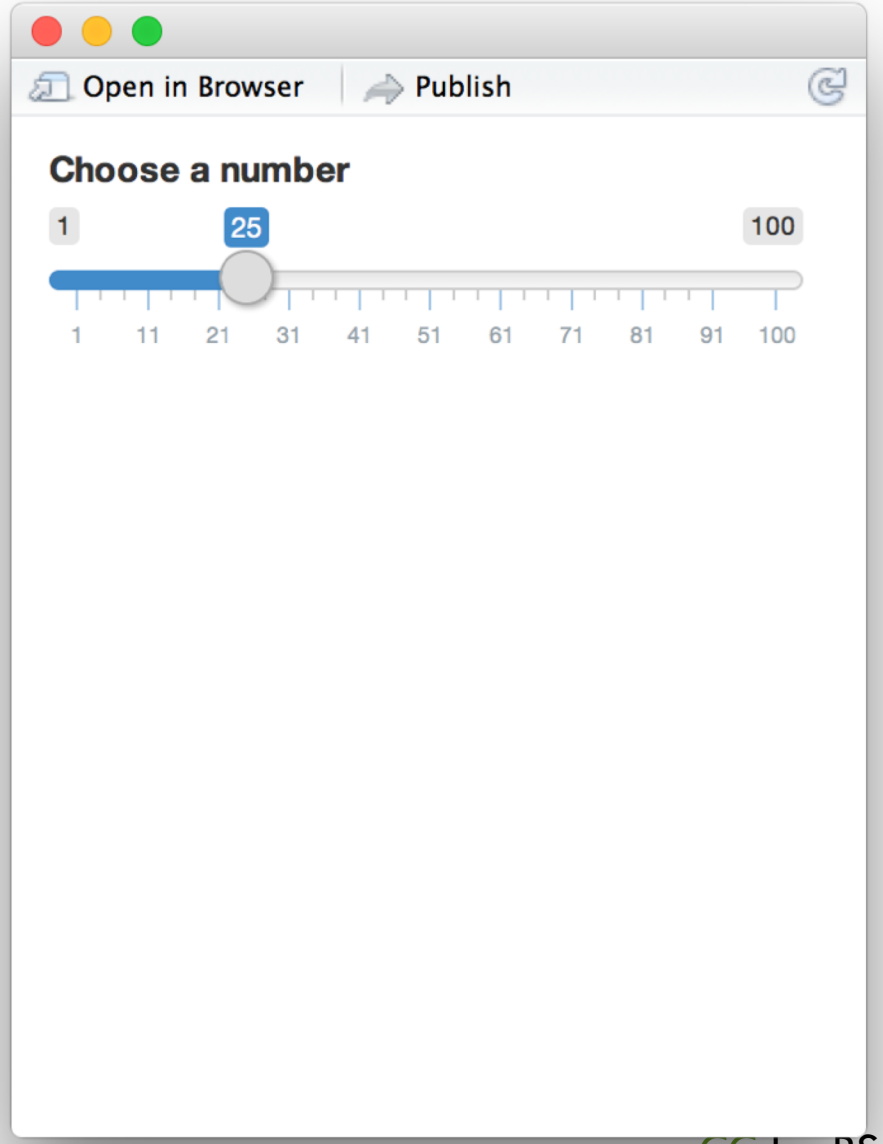
Create an input with an *Input() function.

```
library(shiny)
ui <- fluidPage(



)

server <- function(input, output) {}

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



Create an input with an *Input() function.

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

server <- function(input, output) {}

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



Buttons

Action

Apply Changes

`actionButton()`
`submitButton()`

Single checkbox

Check me

`checkboxInput()`

Checkbox group

Choice 1
Choice 2
Choice 3

`checkboxGroupInput()`

Date input

2014-01-01

`dateInput()`

Date range

2014-01-24 to 2014-01-24

`dateRangeInput()`

File input

Choose File No file chosen

`fileInput()`

Numeric input

1

`numericInput()`

Password Input

`passwordInput()`

Radio buttons

Choice A
Choice B
Choice C

`radioButtons()`

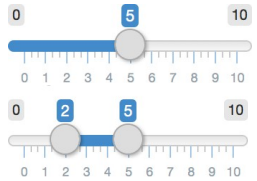
Select box

Choice 1

Choice 1
Choice 2

`selectInput()`

Sliders



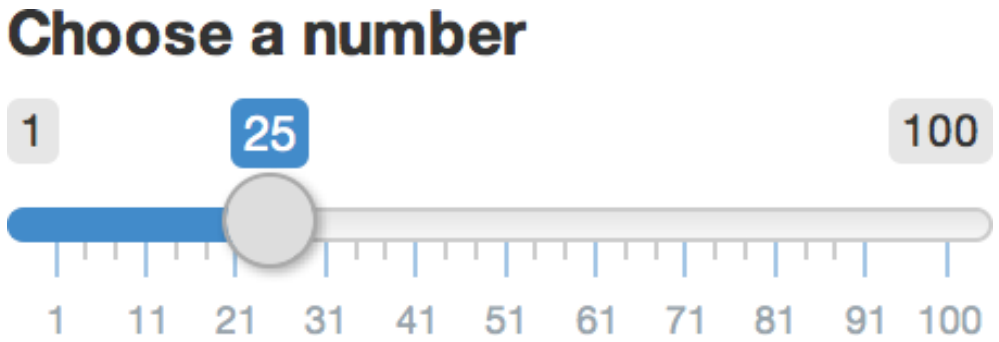
`sliderInput()`

Text input

Enter text

`textInput()`
`textAreaInput()`

How to use *Input() functions




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

input name

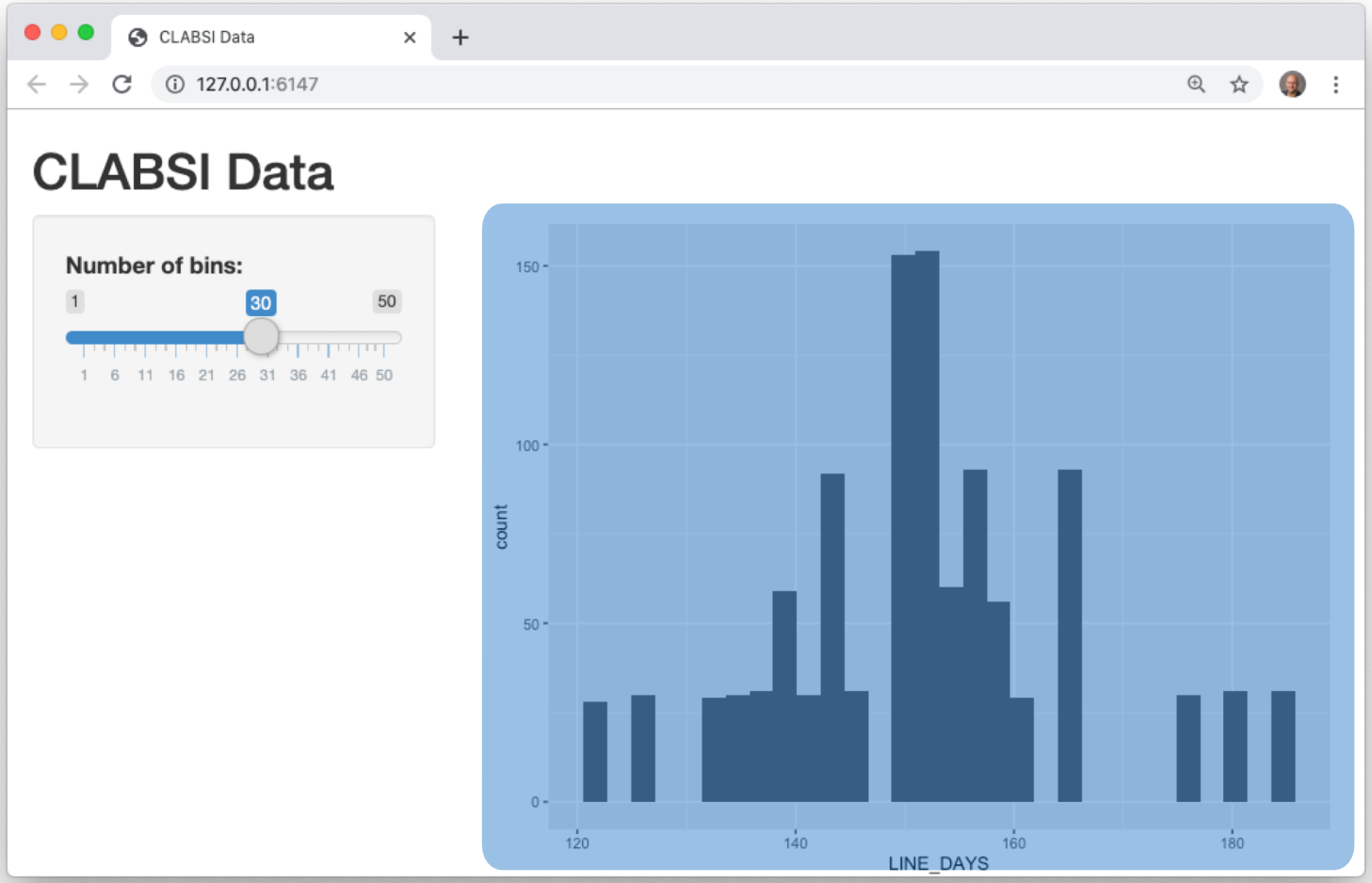
label to display

input-specific arguments

```
?sliderInput
```

Outputs

Build your app around **inputs** and **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

How to use *Output() functions

```
plotOutput(outputId = "hist")
```

type of output to display

output object name

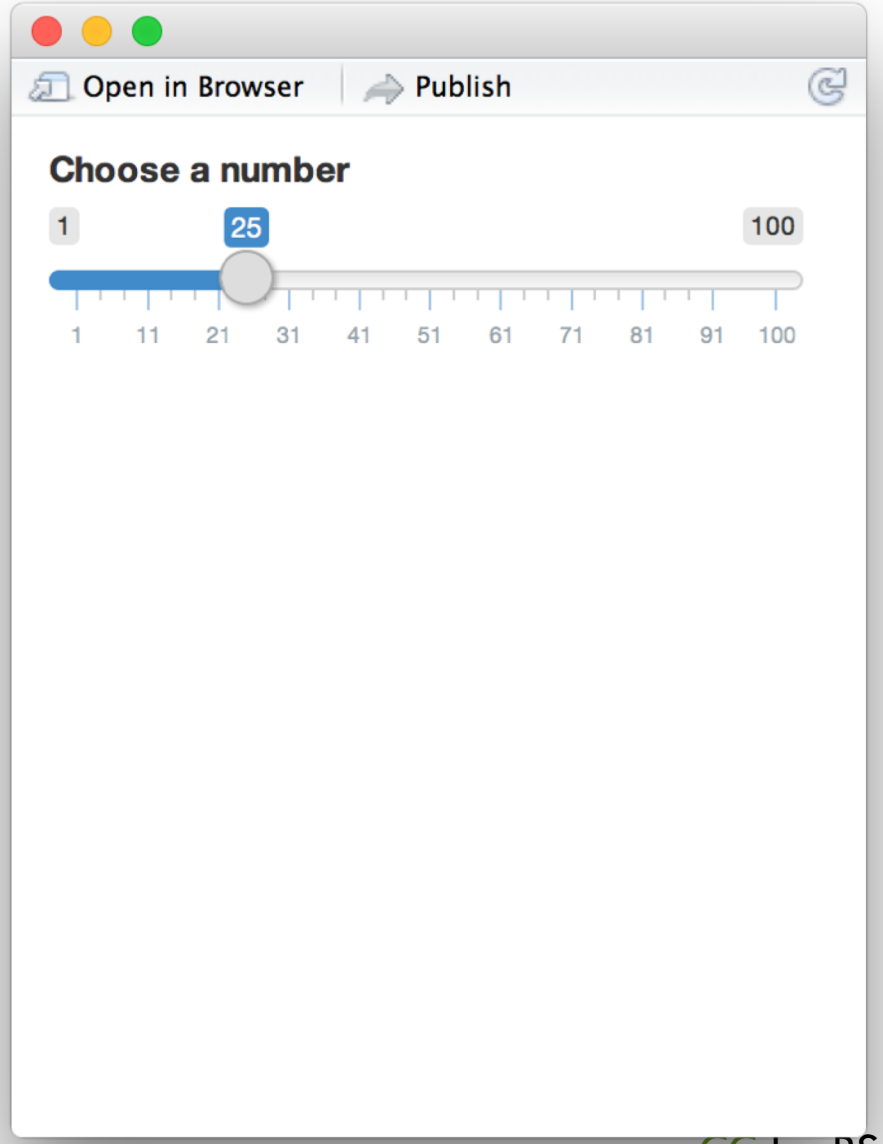
Add an output with an *Output() function.

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

server <- function(input, output) {}

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

Comma between arguments

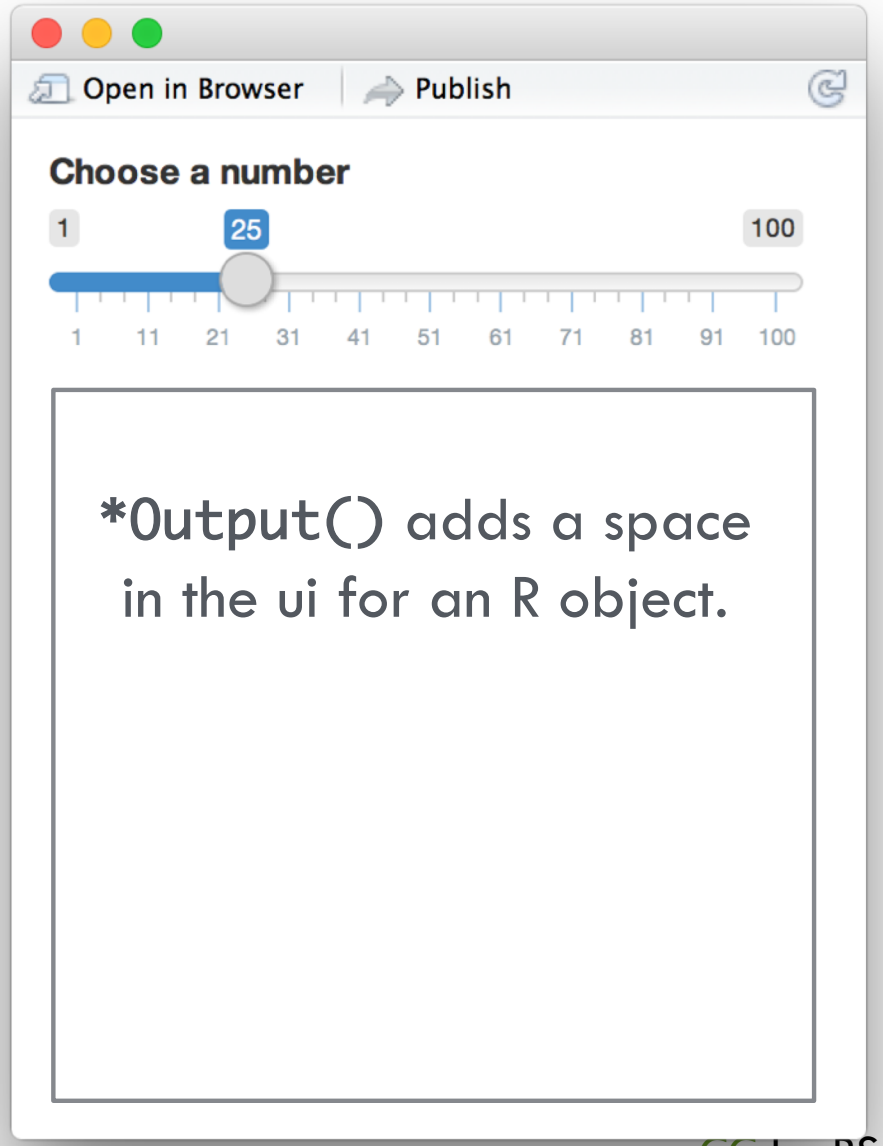
Add an output with an *Output() function.

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

server <- function(input, output) {}

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



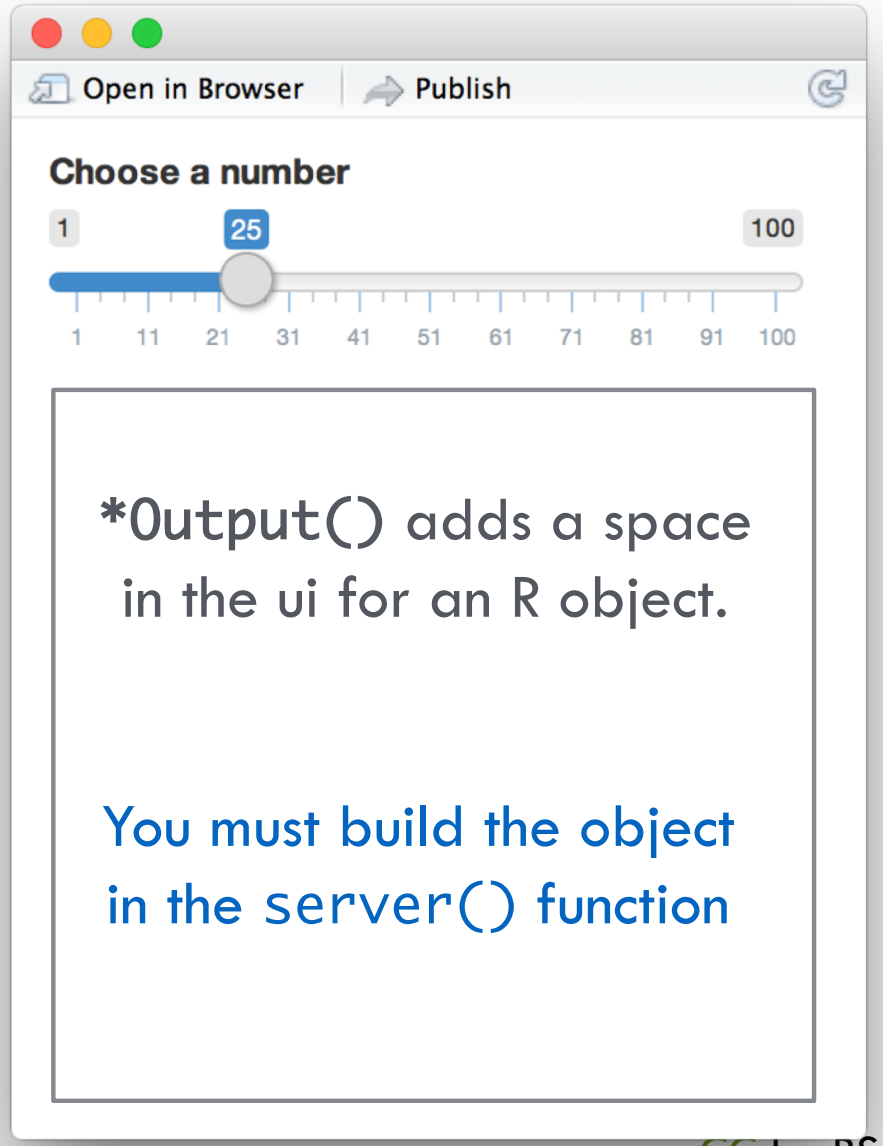
Add an output with an *Output() function.

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

server <- function(input, output) {}

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



Add an output with an *Output() function.

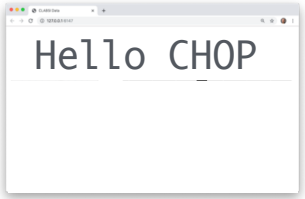
```
library(shiny)
ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  plotOutput("hist")
)

server <- function(input, output) {}

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



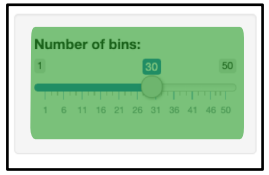
Recap

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

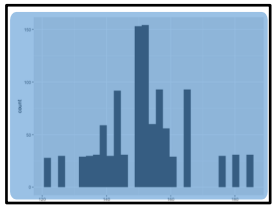
Begin each app with the template



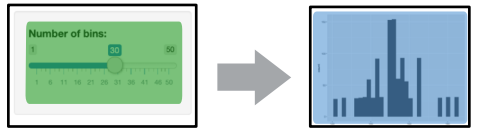
Add elements as arguments to `fluidPage()`



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



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



Assemble outputs from inputs in the `server()` function

Tell the Server How to Assemble Inputs into Outputs

Use 3 rules to write the server function

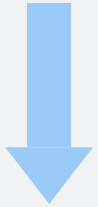
```
server <- function(input, output) {




}
```

1 Save objects to display to output$

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


}
```

1 Save objects to display to output$

```
output$hist
```



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

2

Build reactive objects with render*()

```
server <- function(input, output) {
  output$hist <- renderPlot({


  })
}
```

Use render*() functions to build reactive objects

Function	Builds
`renderDataTable()`	an interactive table
`renderImage()`	an image
`renderPlot()`	a plot
`renderPrint()`	a code block of printed output
`renderTable()`	a table
`renderText()`	a character string
`renderUI()`	a Shiny UI element

How to use render*() functions

```
renderPlot({ hist(rnorm(100) })
```

type of object to build

code block that builds the object

2

Build reactive objects with render*()

```
server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(100))
  })
}
```

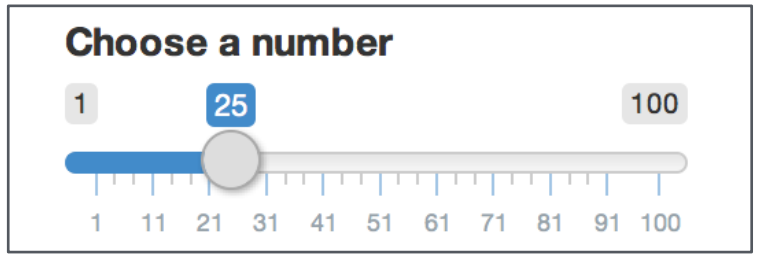
3

Access input values with input$

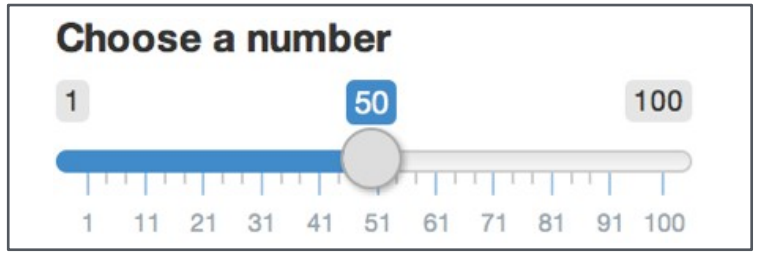
```
sliderInput(inputId = "num", …)
```

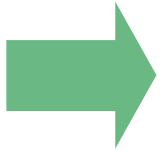
```
input$num
```

The input value changes whenever a user changes the input.

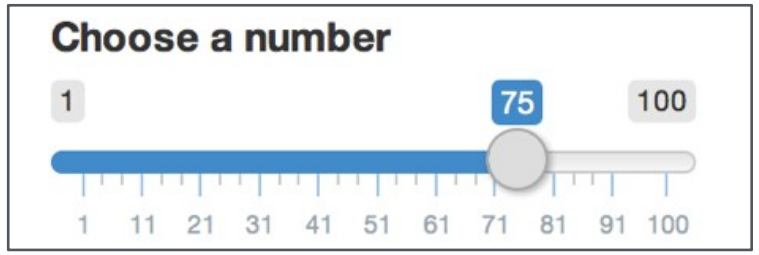


```
input$num = 25
```





```
input$num = 50
```



```
input$num = 75
```

3

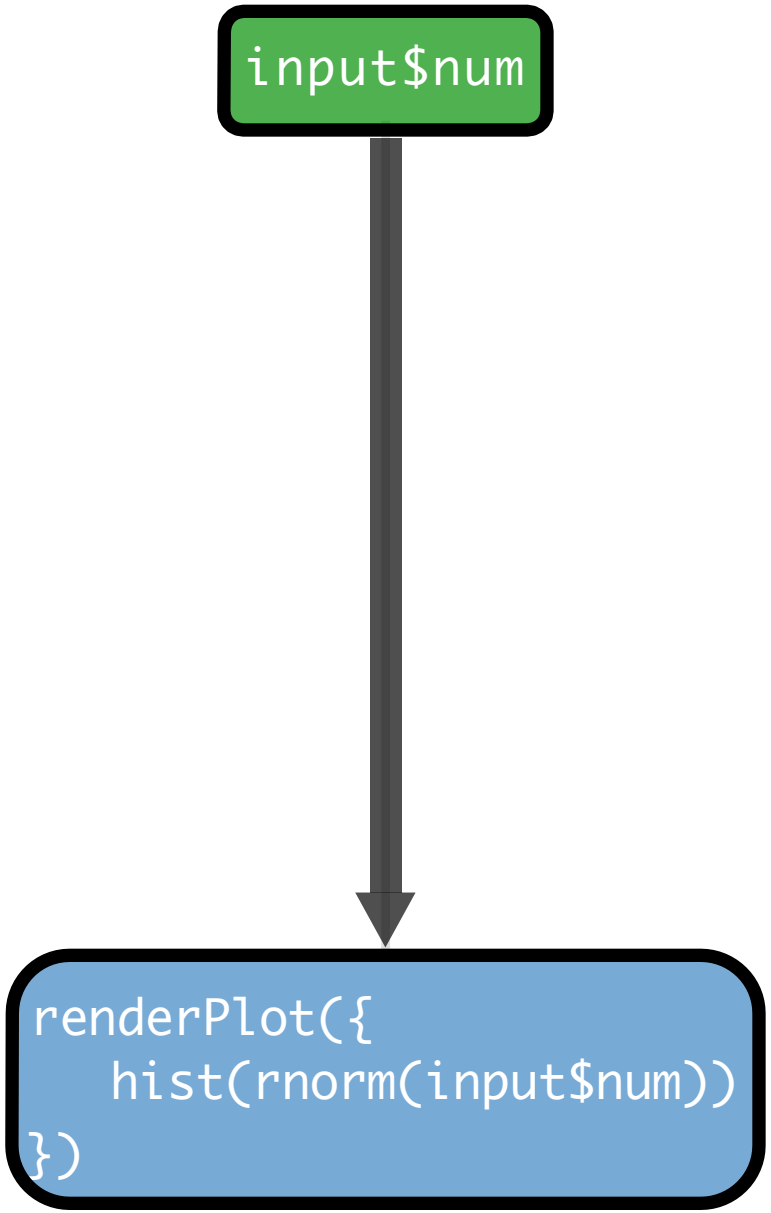
Access input values with input$

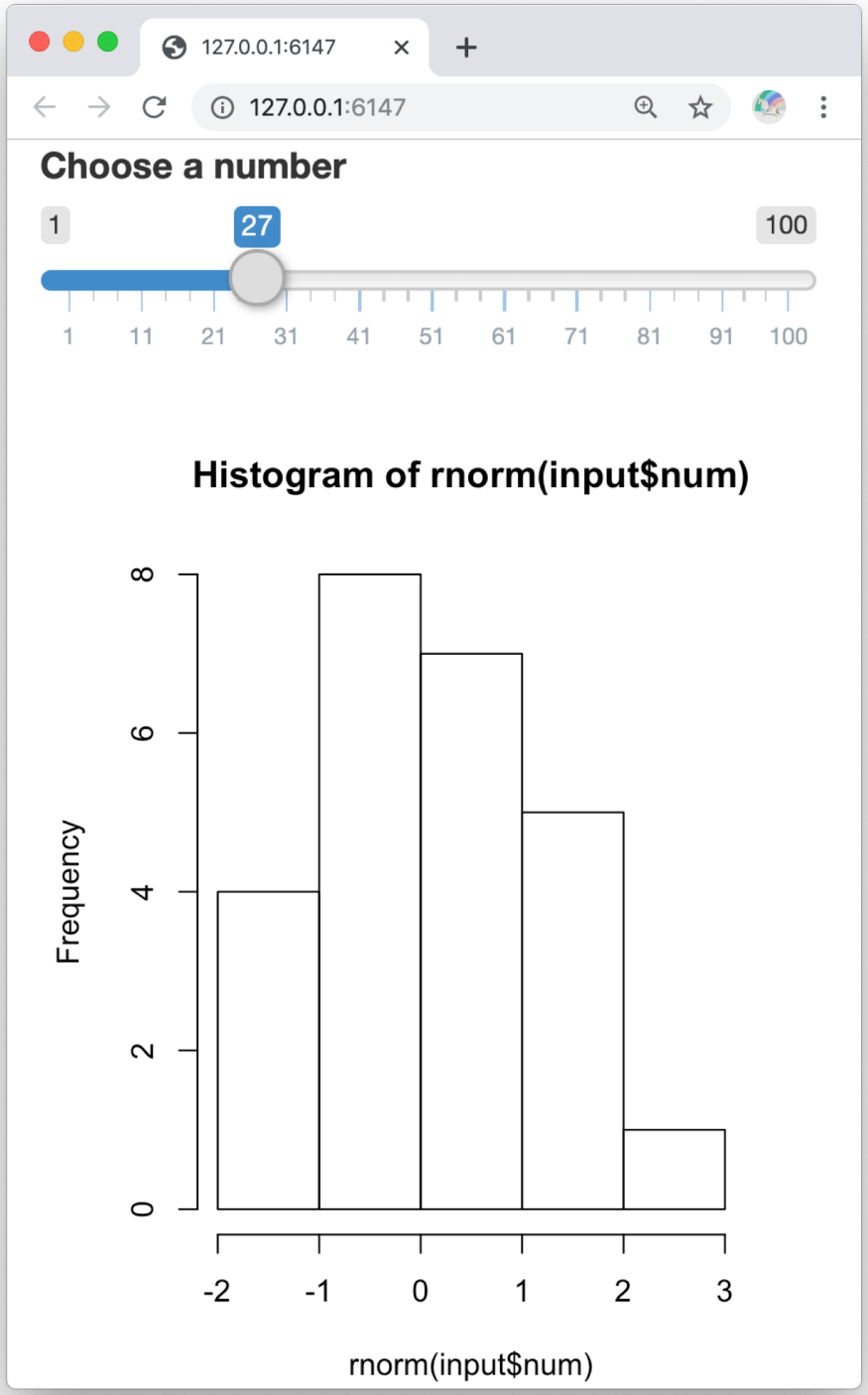
```
server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
}
```

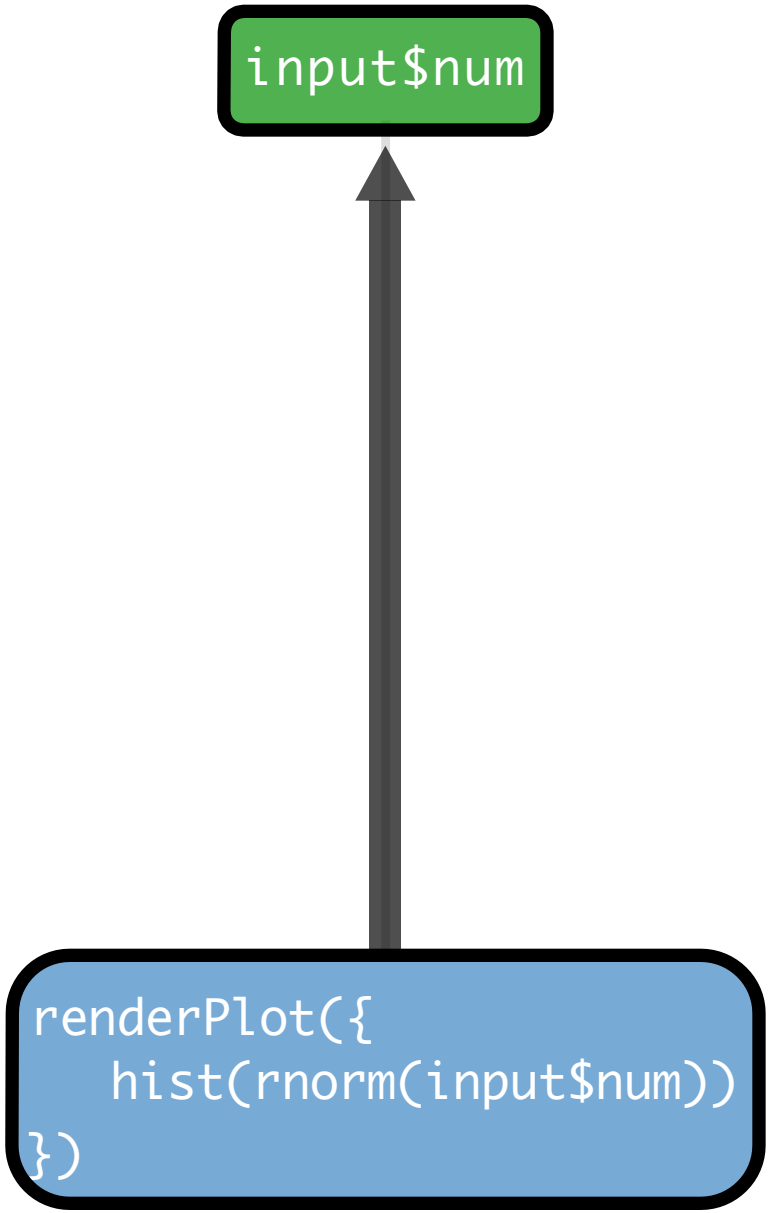
Reactivity occurs **automatically** if you use an reactive input value to render a reactive output object.

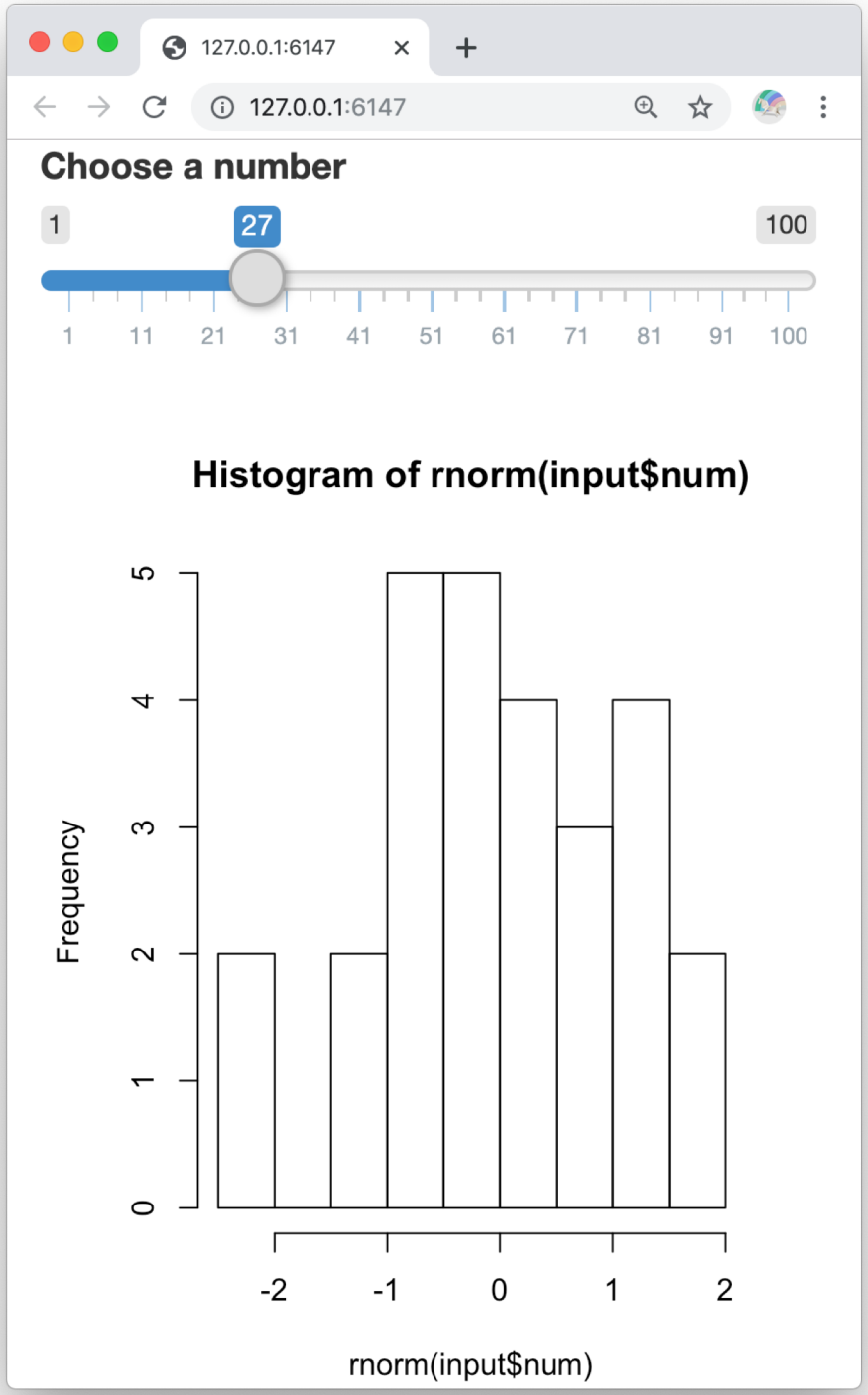
```
server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
}
```

Output will automatically update if you follow the **3 rules**


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


127.0.0.1:6147
127.0.0.1:6147
Choose a number
1
27
100
1 11 21 31 41 51 61 71 81 91 100
Histogram of rnorm(input$num)
Frequency
0 2 4 6 8
-2 -1 0 1 2 3
rnorm(input$num)


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


127.0.0.1:6147
127.0.0.1:6147
Choose a number
1
27
100
1 11 21 31 41 51 61 71 81 91 100
Histogram of rnorm(input$num)
Frequency
0 1 2 3 4 5
-2 -1 0 1 2
rnorm(input$num)

Recap: Server

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

`output$hist <-`

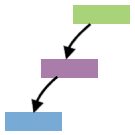
1. Save the output that you build to `output$`

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

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

`input$num`

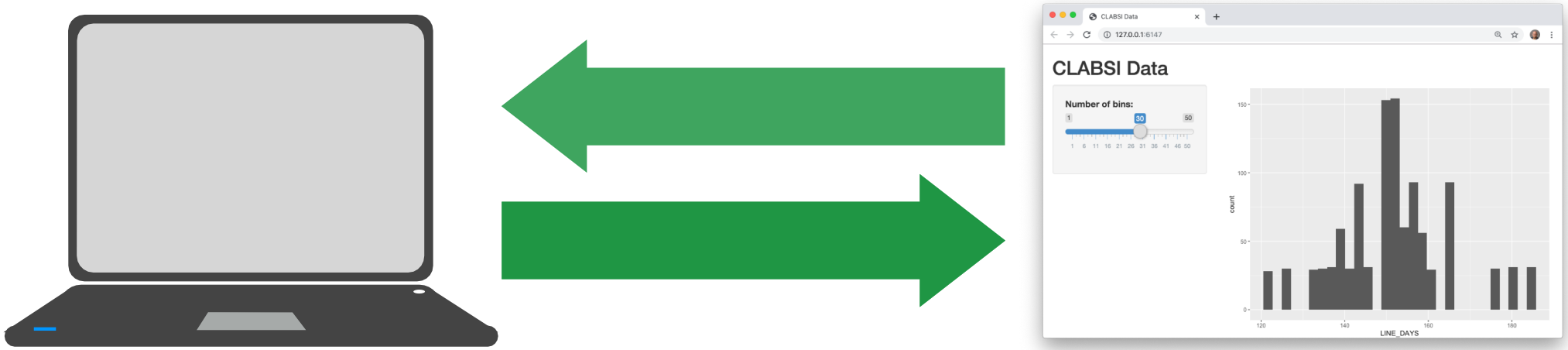
3. Access input values with `input$`



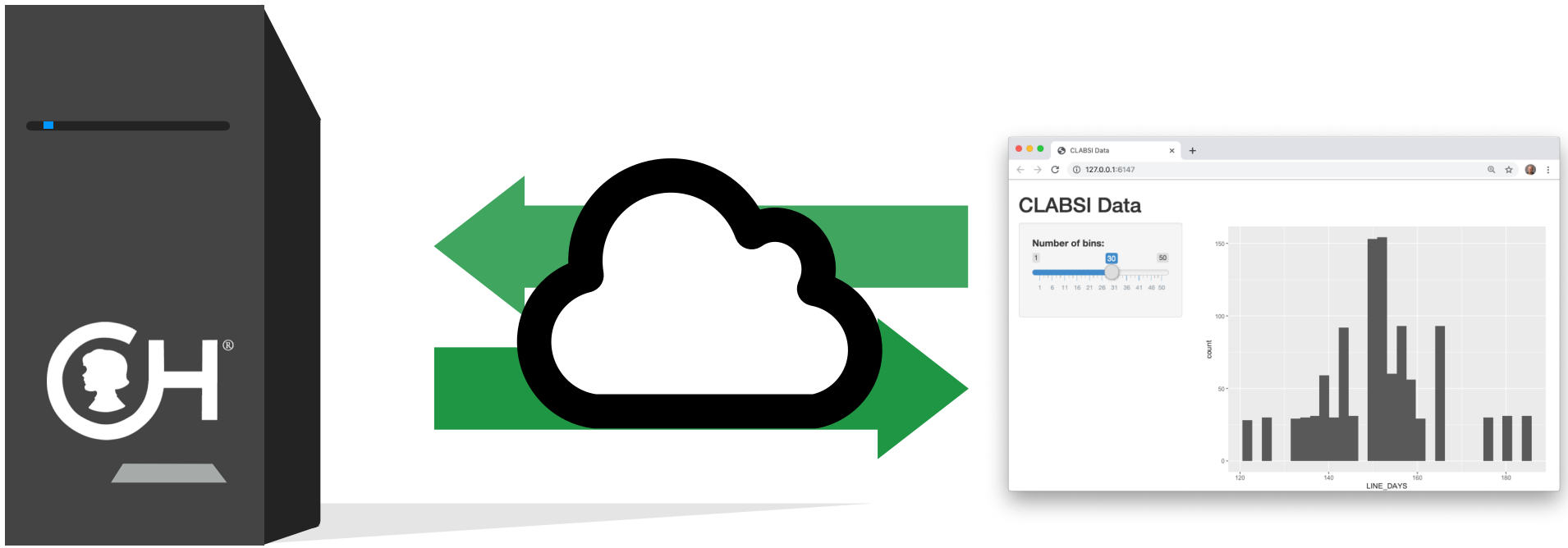
Create reactivity by using **Inputs** to build **rendered Outputs**

Deploy Your App!

Every Shiny app is maintained by a computer running R



Every Shiny app is maintained by a computer running R

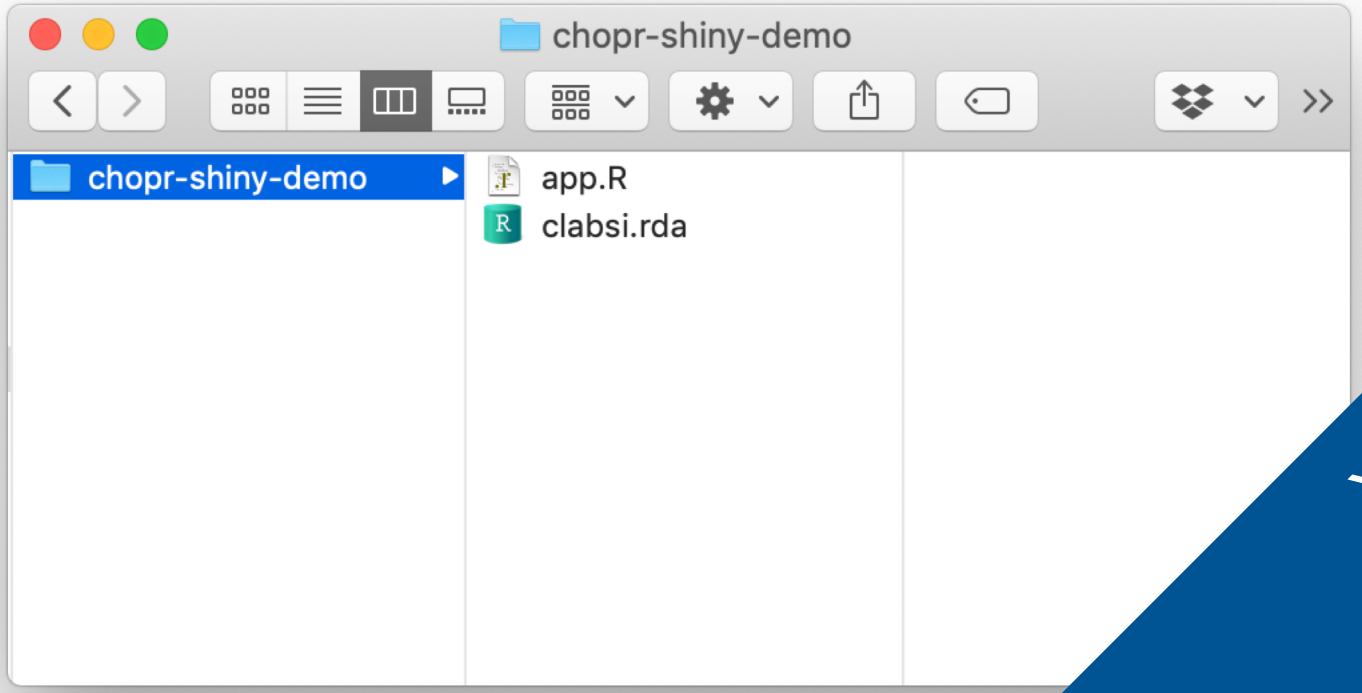


rstudio-connect.chop.edu

How to save your app

One directory with every file the app needs:

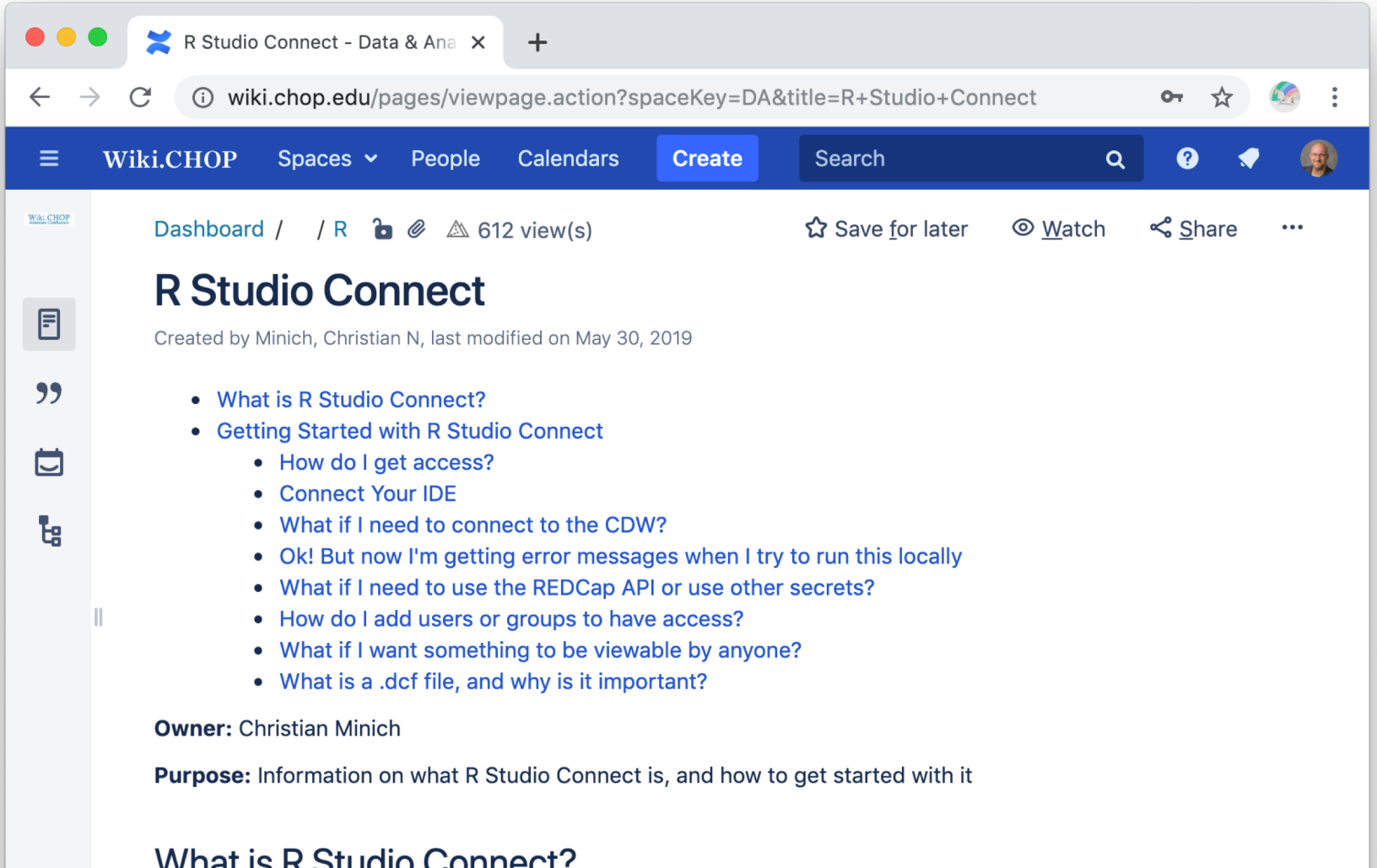
- app.R *(your script which ends with a call to* `shinyApp()`*)*
- datasets, images, css, helper scripts, etc.



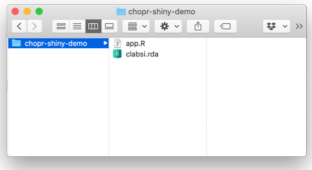
You should use this exact name (app.R)

Getting started guide

wiki.chop.edu/display/DA/R+Studio+Connect

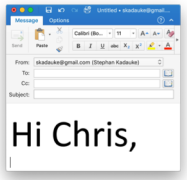


Recap: Sharing



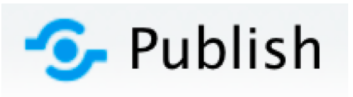
Save your app in its own directory as **app.R**

To deploy apps to CHOP's RStudio Connect:



1. Request access from minichc@email.chop.edu

2. Install the `rsconnect` package

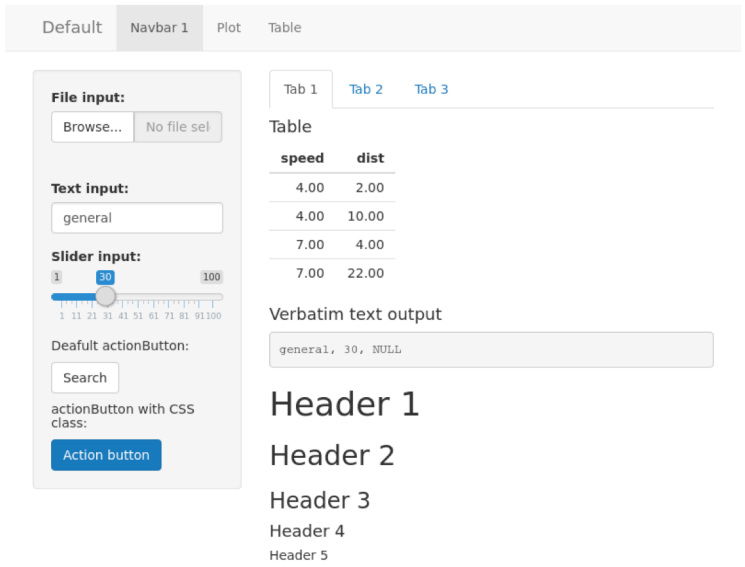


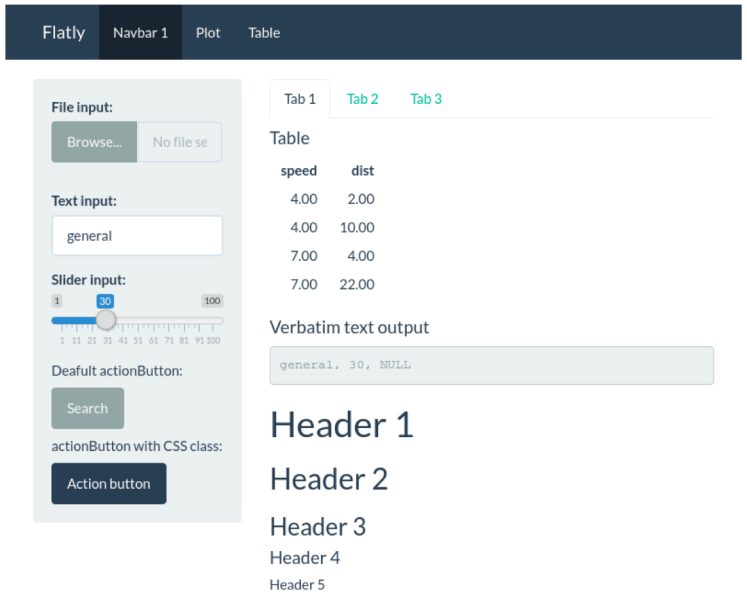
3. Push the Publish button

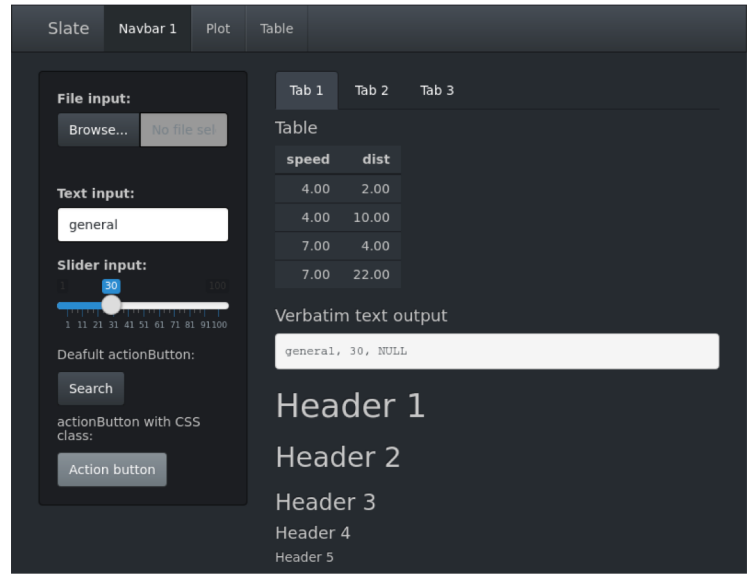
Make Your App CHOP-Themed

Shiny themes

rstudio.github.io/shinythemes







CHOP Bootstrap Theme

CHOP theme for Bootstrap. This theme was generated using Bootswatch. The colors in this theme are based off of the CHOP style guide.

The following versions of Bootstrap are supported:

- Bootstrap 3
- Bootstrap 4

Usage

In your HTML, replace the original Bootstrap CSS (`bootstrap.css` or `bootstrap.min.css`) with the one from the applicable Bootstrap version directory.

This repo is hosted with GitHub pages and both the unminified and minified CSS are available.

```
<!-- Bootstrap 4 - CHOP Theme (Recommended) -->
<link rel="stylesheet" href="https://github.research.chop.edu/pages/CQI/chop-bootstrap/bootstrap-4/bootstr

<!-- Bootstrap 3 - CHOP Theme -->
<link rel="stylesheet" href="https://github.research.chop.edu/pages/CQI/chop-bootstrap/bootstrap-3/bootstr
```

github.research.chop.edu/pages/CQI/chop-bootstrap

rocqi

build passing coverage 100%

Overview

R package to gather, explore, analyze, and present data for CHOP.

Installation

Install the latest version of `rocqi` from CHOPRAN

```
install.packages("rocqi")
```

github.research.chop.edu/CQI/rocqi

CHOPRAN

A Repository for internal CHOP R packages

Usage

To install packages from CHOPRAN, you must first add it to your list of package repos. We recommend you edit your `.Rprofile`:

```
# Open .Rprofile for editing
> usethis::edit_r_profile()
```

and add the following:

```
# Set CRAN and CHOPRAN repositories
options(
  repos = c(
    CRAN = "https://cran.rstudio.com/",
    CHOPRAN = "https://github.research.chop.edu/pages/CQI/chopran/"
  )
)
```

github.research.chop.edu/pages/CQI/chopran

Recap: Styling

Use the `rocqi` package and the `chop-bootstrap` theme to help make your Shiny apps follow the CHOP design system.

Install `rocqi` from `chopran`

Pass a link to the `chop-bootstrap` CSS file to `fluidPage()` as the `theme` argument

Add `+ theme_chop()` to ggplots

Learn More

shiny.rstudio.com/tutorial

Shiny - Tutorial

shiny.rstudio.com/tutorial/

Shiny from RStudio

Learn Shiny

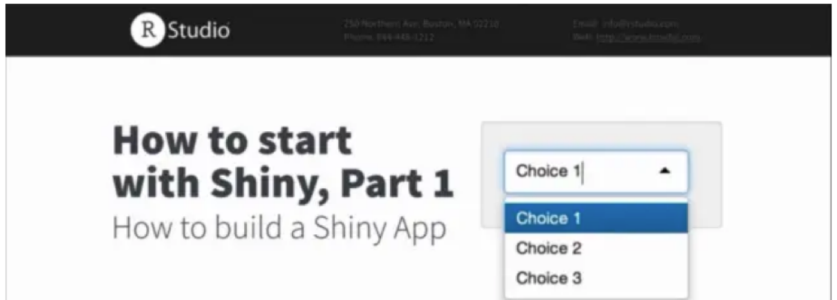
The video and written tutorials on this page are primarily designed for users who are new to Shiny and want a guided introduction.

If you use Shiny on a regular basis, you may want to skip these tutorials and visit the articles section where we cover individual Shiny topics at a more advanced level.

Video tutorials

How to Start Shiny tutorial

The How to Start Shiny video series will take you from R programmer to Shiny developer. Watch the complete tutorial, or jump to a specific chapter by clicking a link below. The entire tutorial is two hours and 25 minutes long. Download the slides and exercises here.



mastering-shiny.org

Mastering Shiny

mastering-shiny.org

Mastering Shiny

Hadley Wickham

Welcome

This is the online version of *Mastering Shiny*, a book **currently under early development** and intended for a late 2020 release by O'Reilly Media.

Shiny is a framework for creating web applications using R code. It is designed primarily with data scientists in mind, and to that end, you can create pretty complicated Shiny apps with no knowledge of HTML, CSS, or JavaScript. On the other hand, Shiny doesn't limit you to creating trivial or prefabricated apps: its user interface components can be easily customized or extended, and its server uses reactive programming to let you create any type of backend logic you want. Shiny is designed to feel almost magically easy when you're getting started, and yet the deeper you get into how it works, the more you realize it's built out of general building blocks that have strong software engineering principles behind them.

Today, Shiny is used in almost as many niches and industries as R itself is. It's used in academia as a teaching tool for statistical concepts, a way to get undergrads excited about learning to write code, a splashy medium for showing off novel statistical methods or models. It's used by big pharma companies to speed collaboration

Thank you!