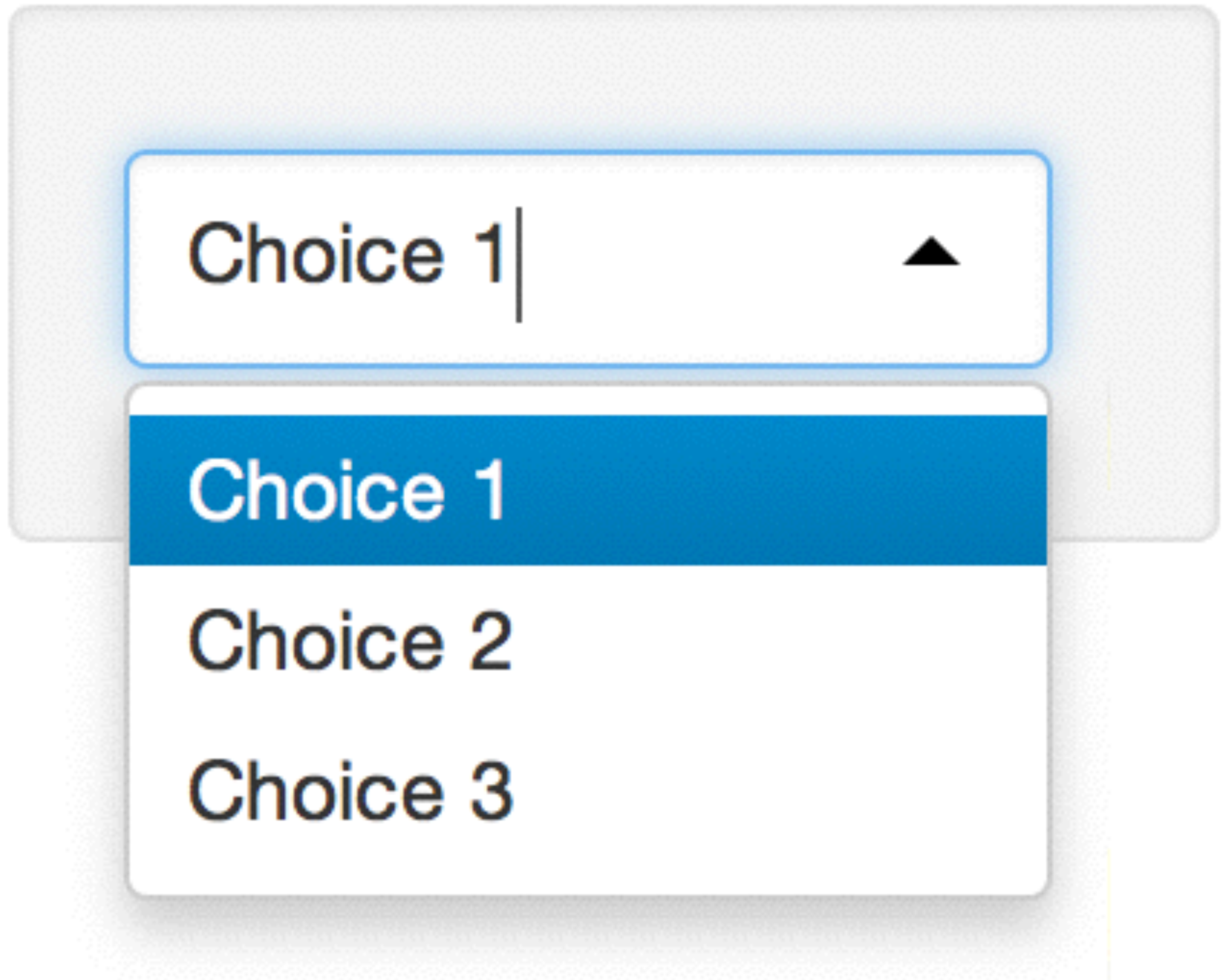
How to start with Shiny, Part 1

How to build a Shiny App



Garrett Grolemund
Data Scientist and Master Instructor
May 2015
Email: garrett@rstudio.com

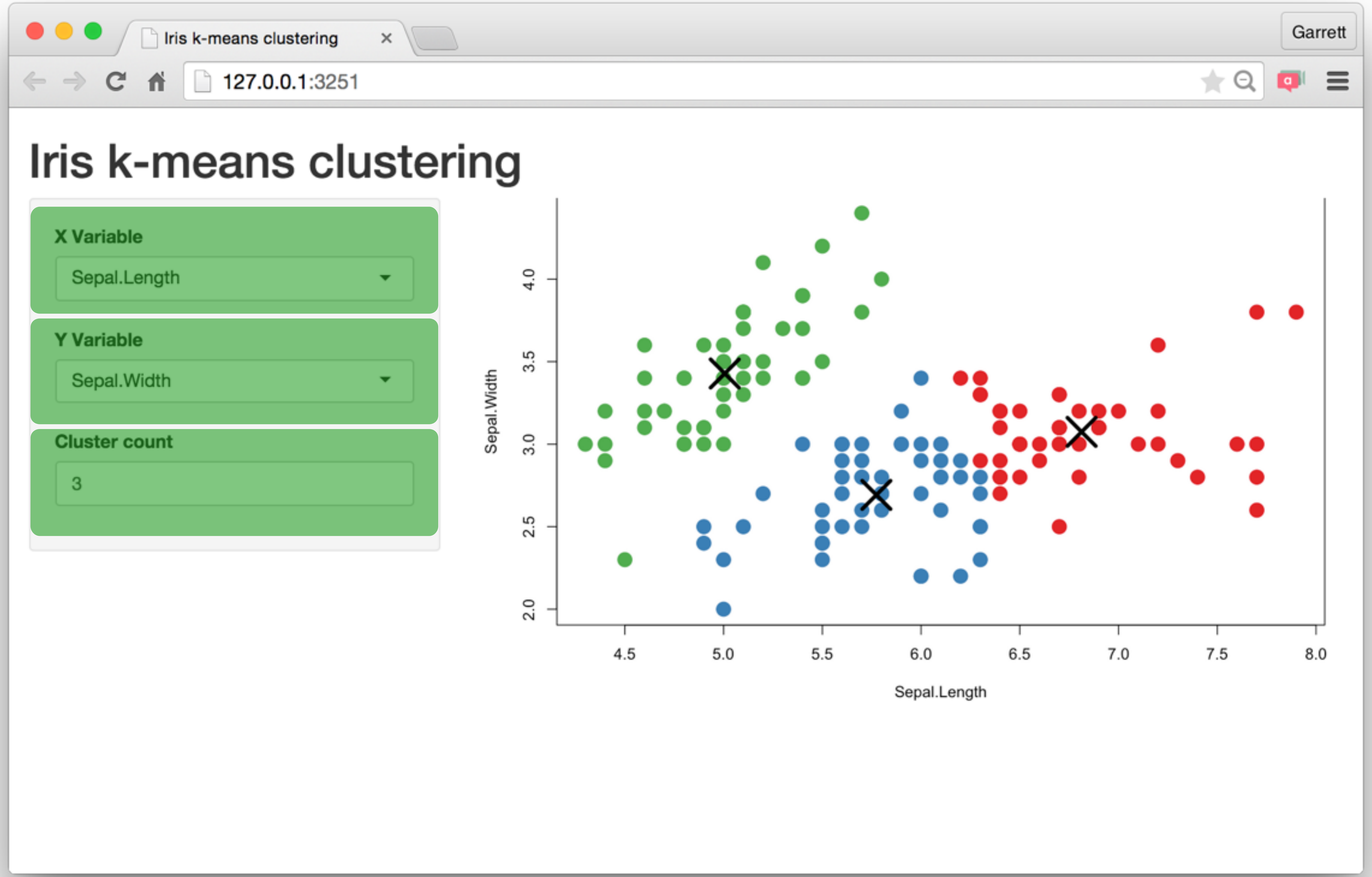
Add elements to your app as arguments to `fluidPage()`

```
library(shiny)
ui <- fluidPage("Hello World")

server <- function(input, output) {}

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

Build your app around

Inputs and Outputs

Build your app around inputs and outputs



Iris k-means clustering

127.0.0.1:3251

Iris k-means clustering

X Variable

Sepal.Length

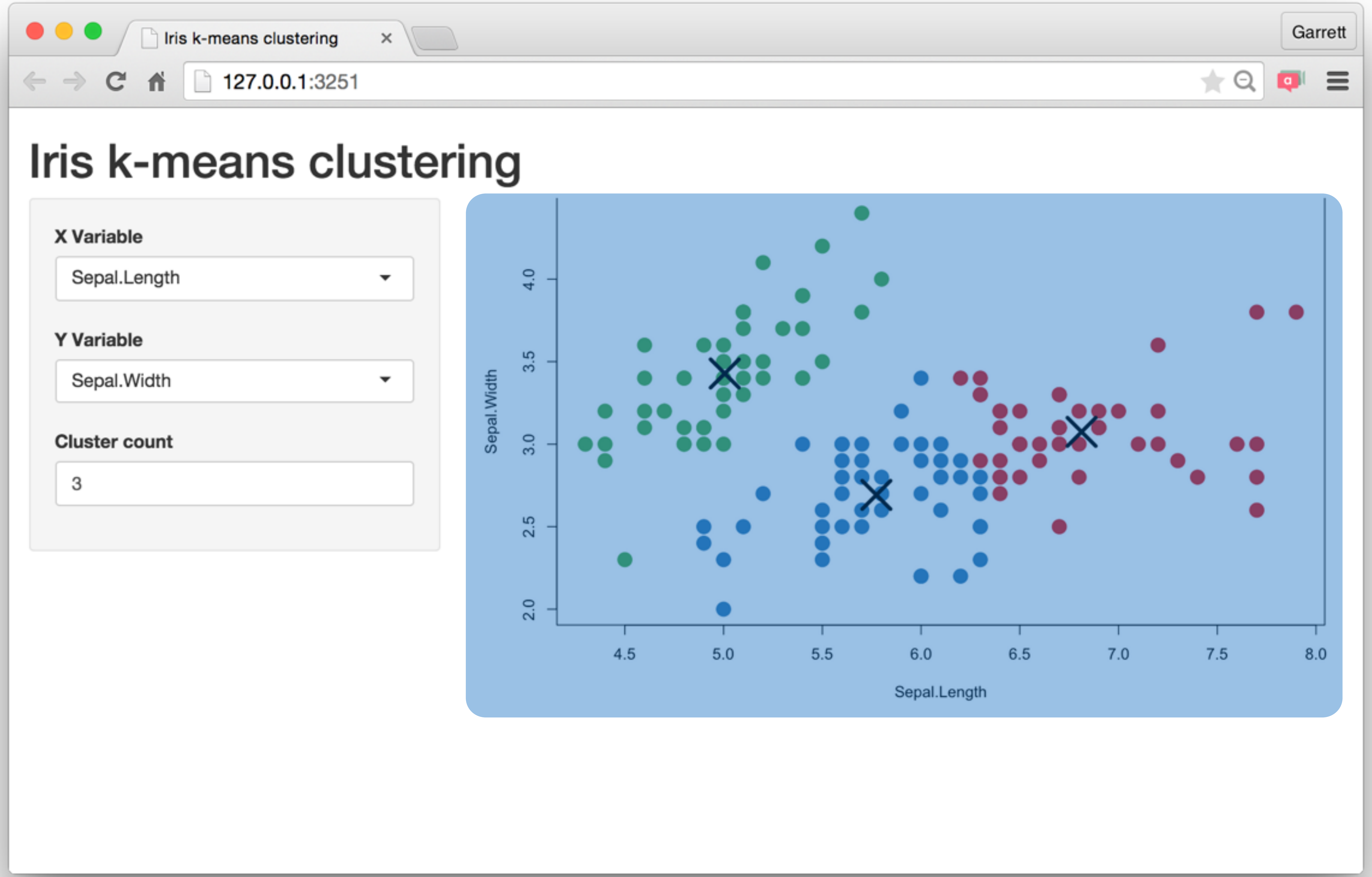
Y Variable

Sepal.Width

Cluster count

3

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


Iris k-means clustering

127.0.0.1:3251

X Variable: Sepal.Length

Y Variable: Sepal.Width

Cluster count: 3

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

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

Inputs

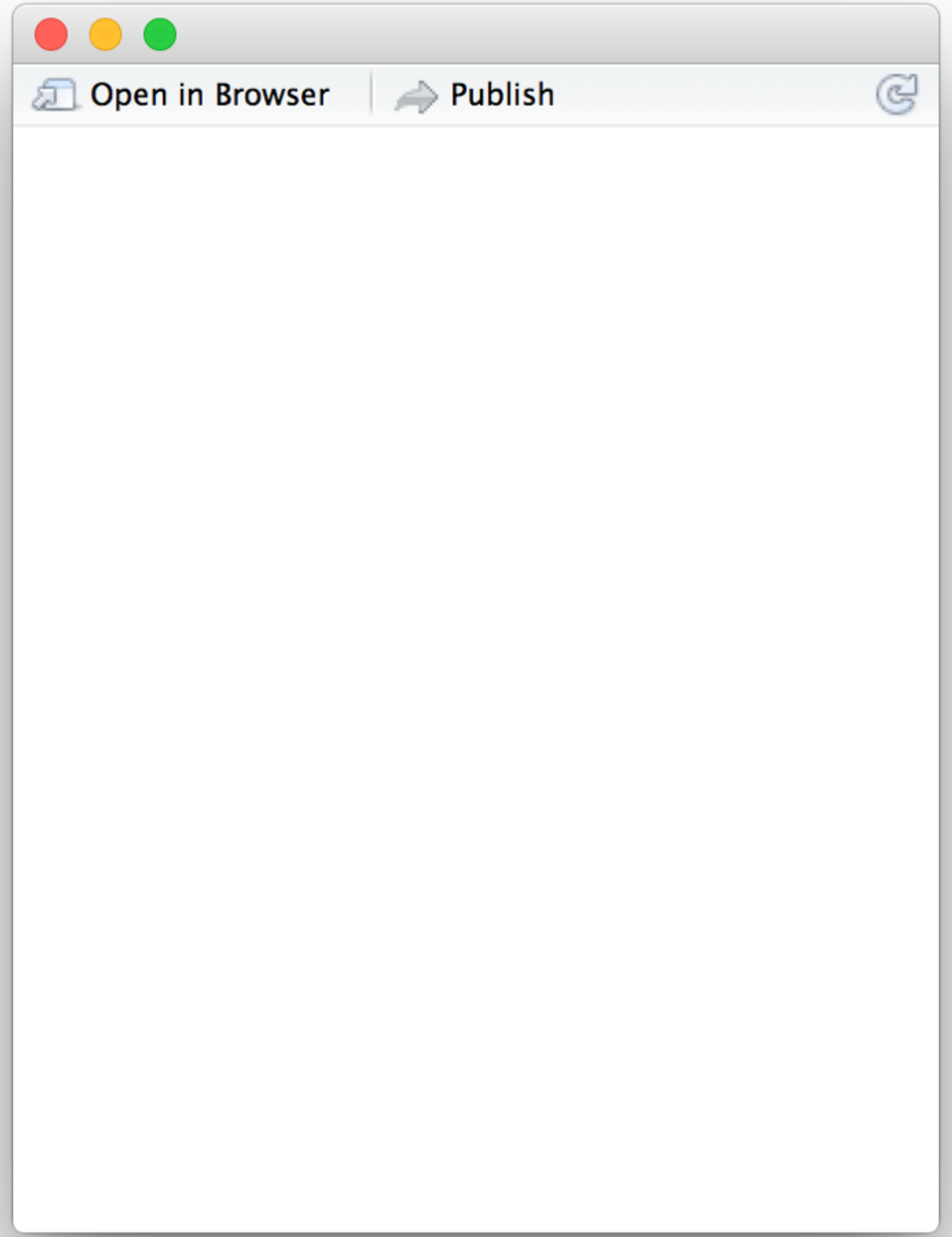
Create an input with an ***Input()** function.

```
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>
```

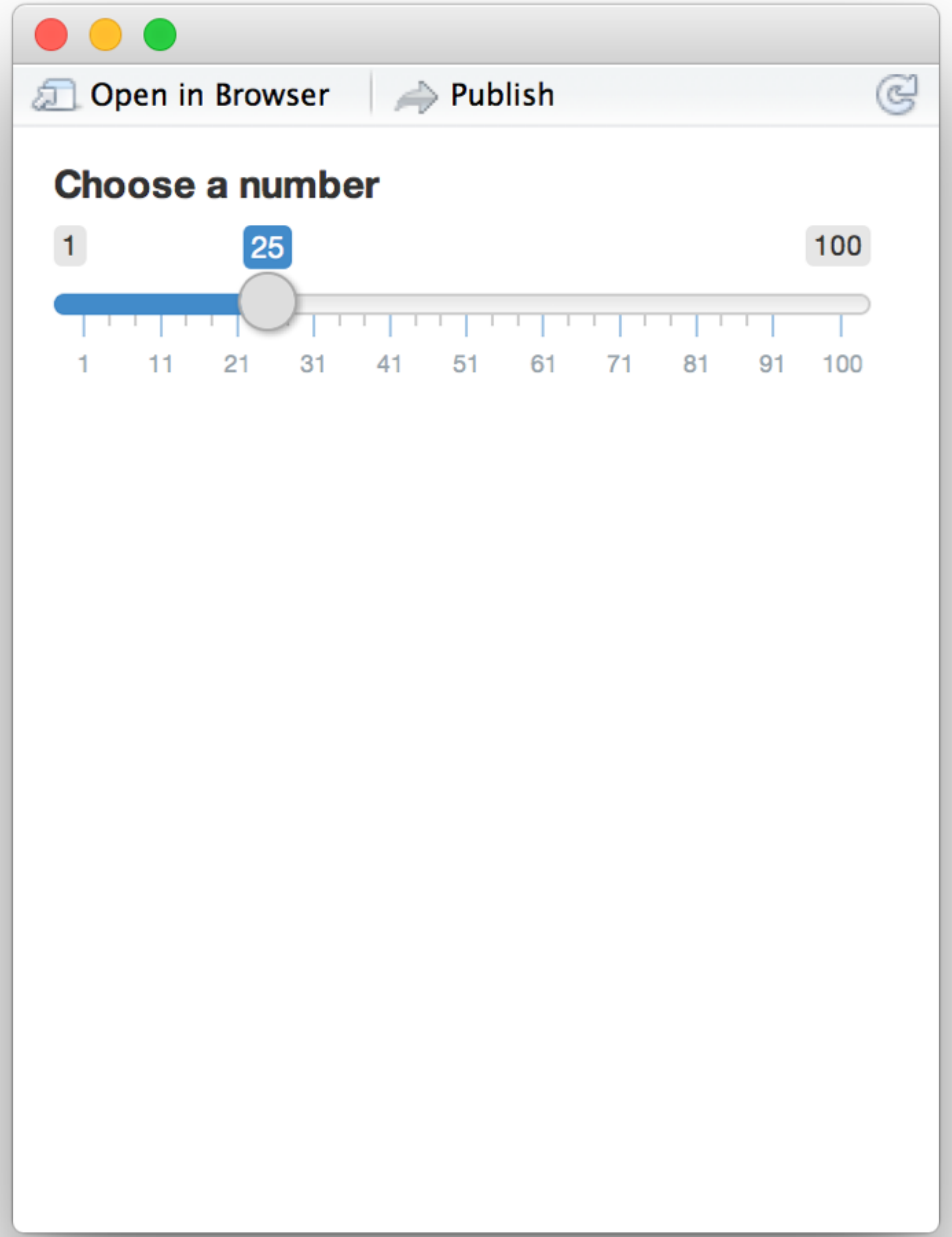
Create an input with an input function.

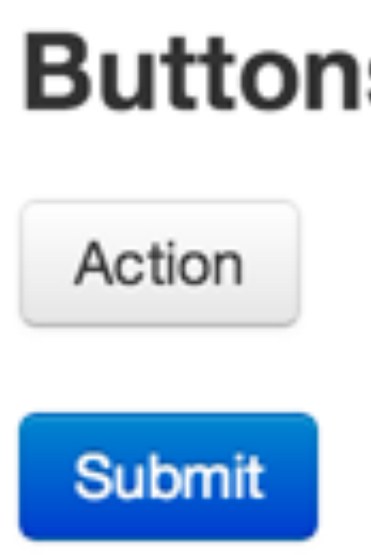
```
library(shiny)
ui <- fluidPage(




)

server <- function(input, output) {}

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



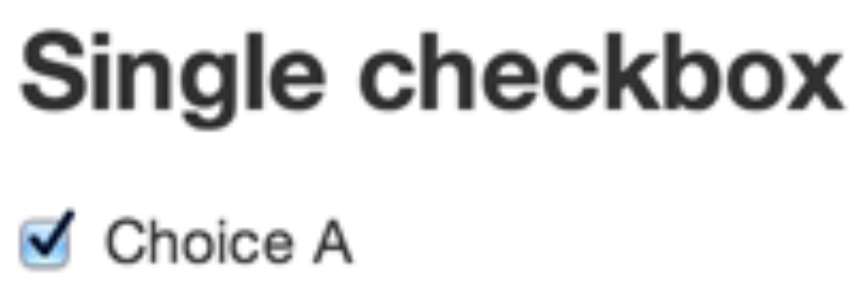
Create an input with an input function.

```
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)
```

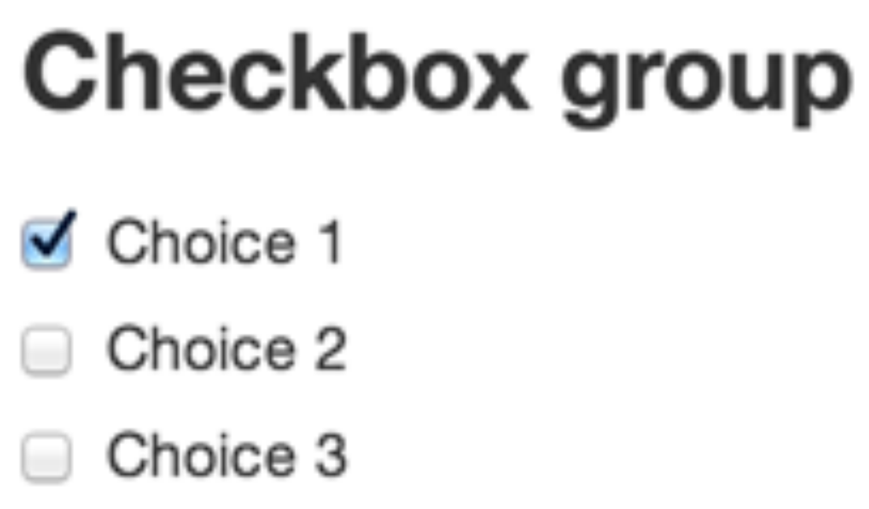




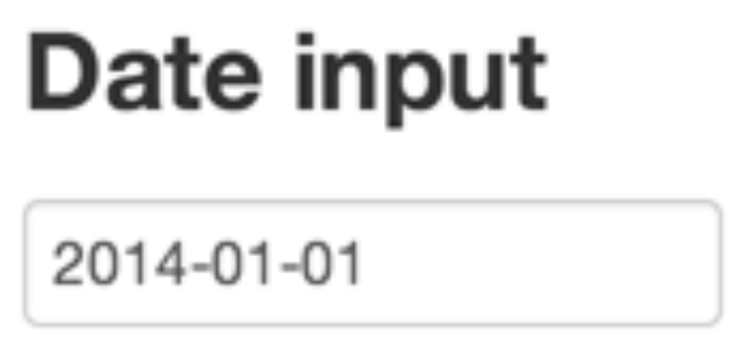
actionButton()
submitButton()



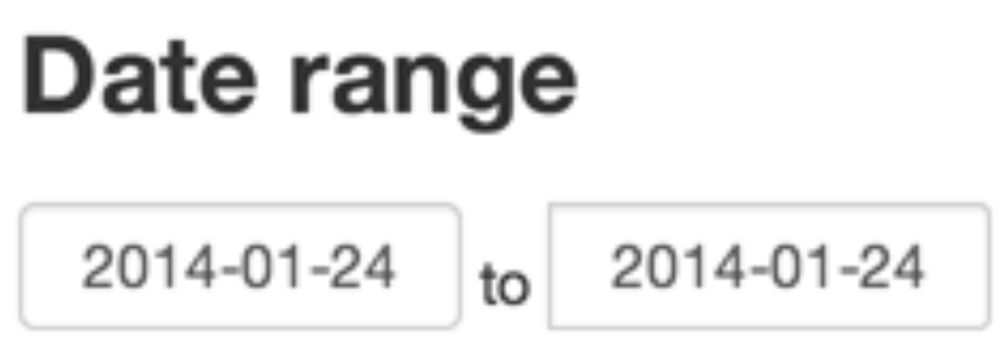
checkboxInput()



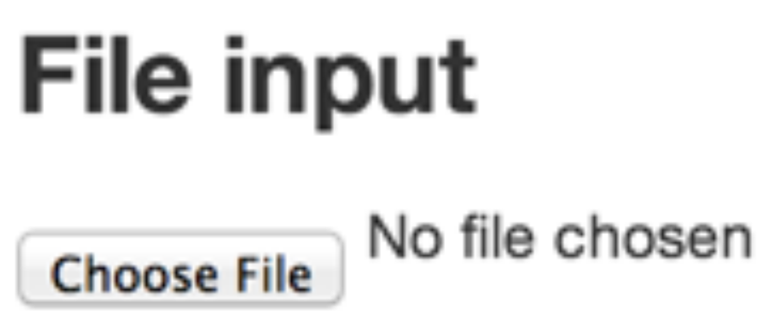
checkboxGroupInput()



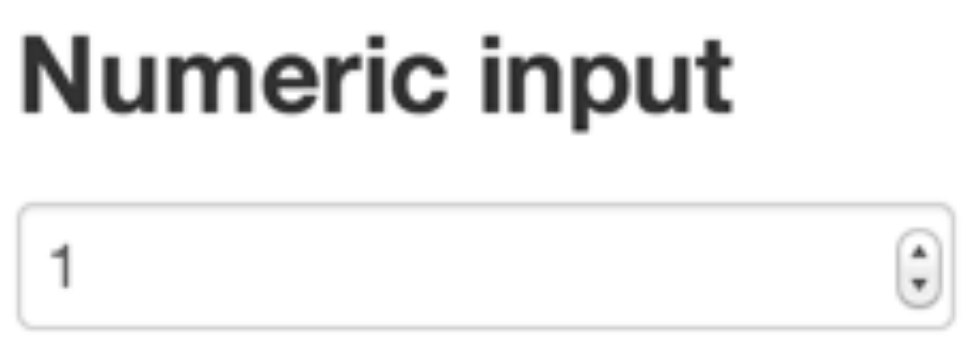
dateInput()



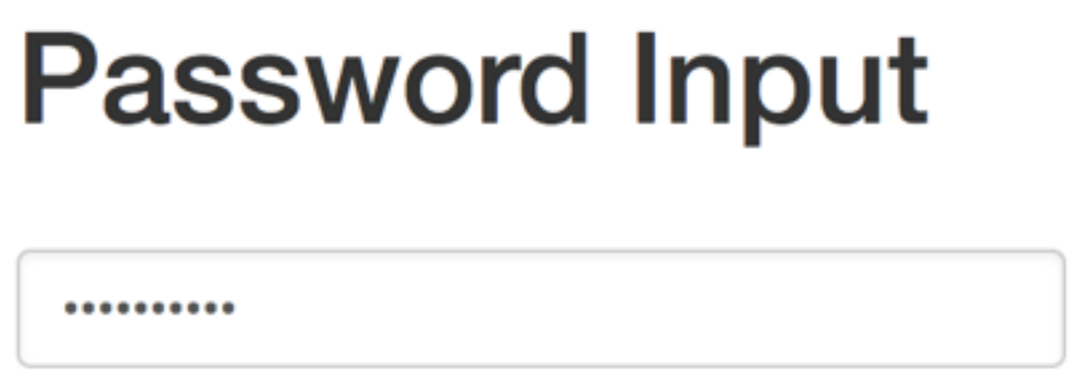
dateRangeInput()



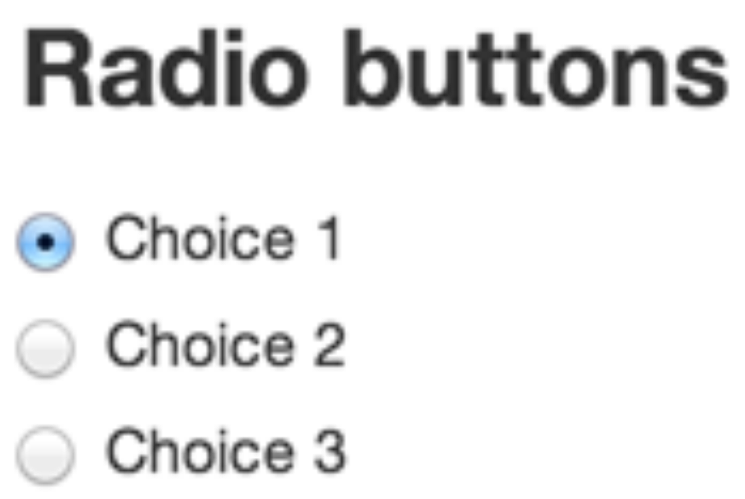
fileInput()



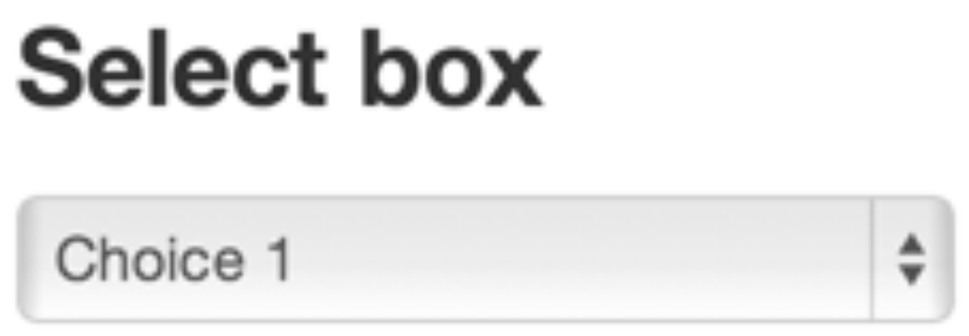
numericInput()



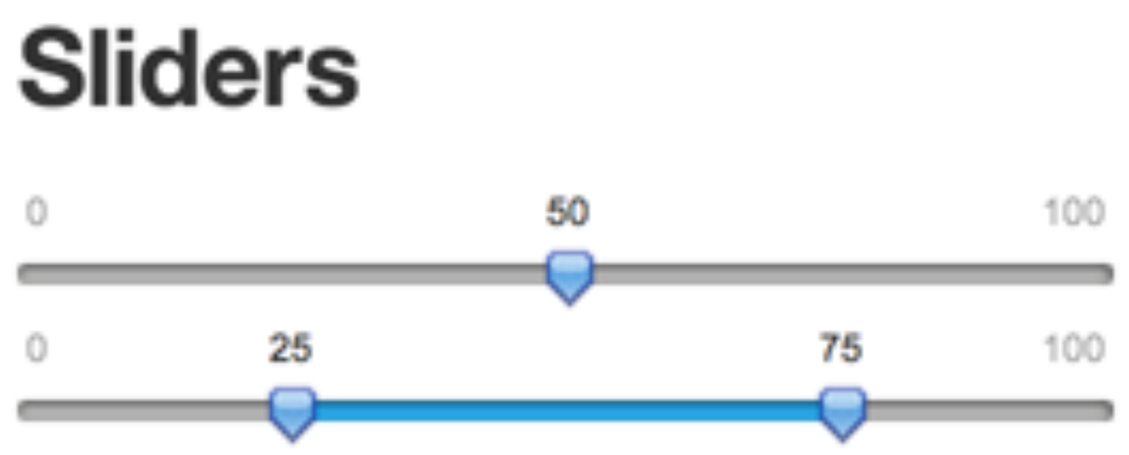
passwordInput()



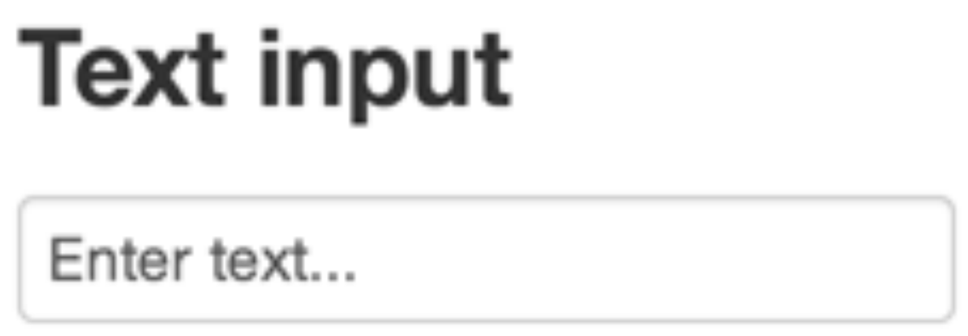
radioButtons()



selectInput()

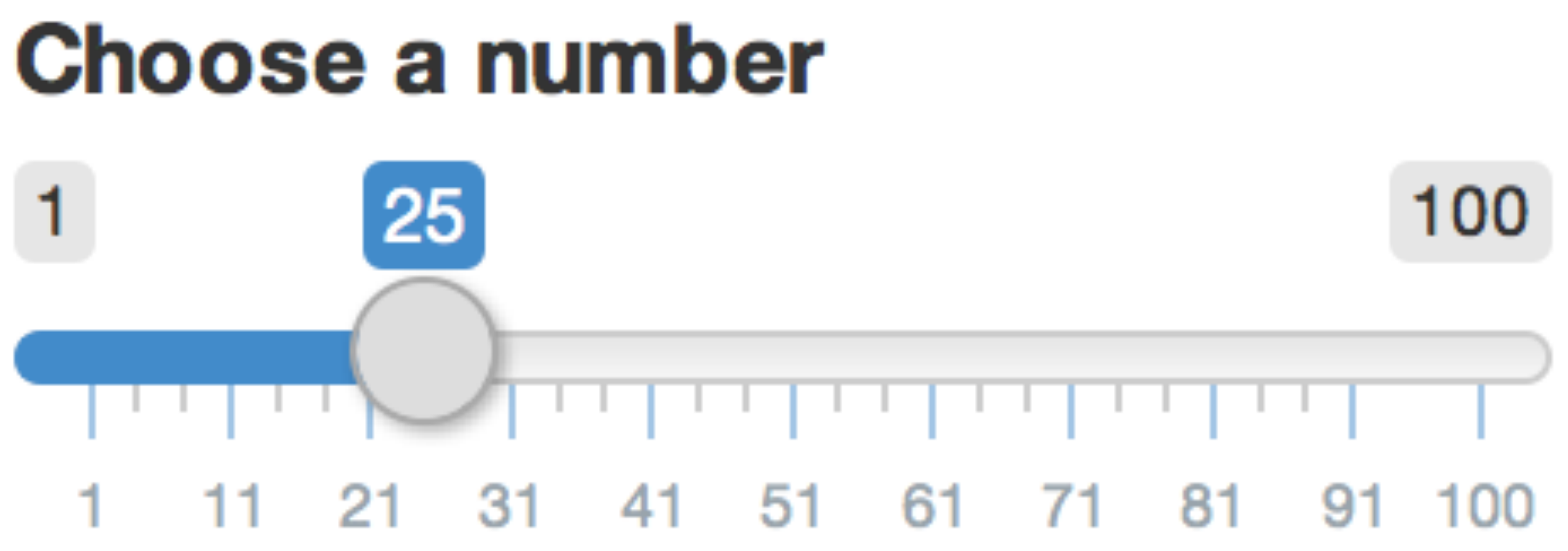


sliderInput()



textInput()

Syntax



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

input name
(for internal use)

Notice:
Id not ID

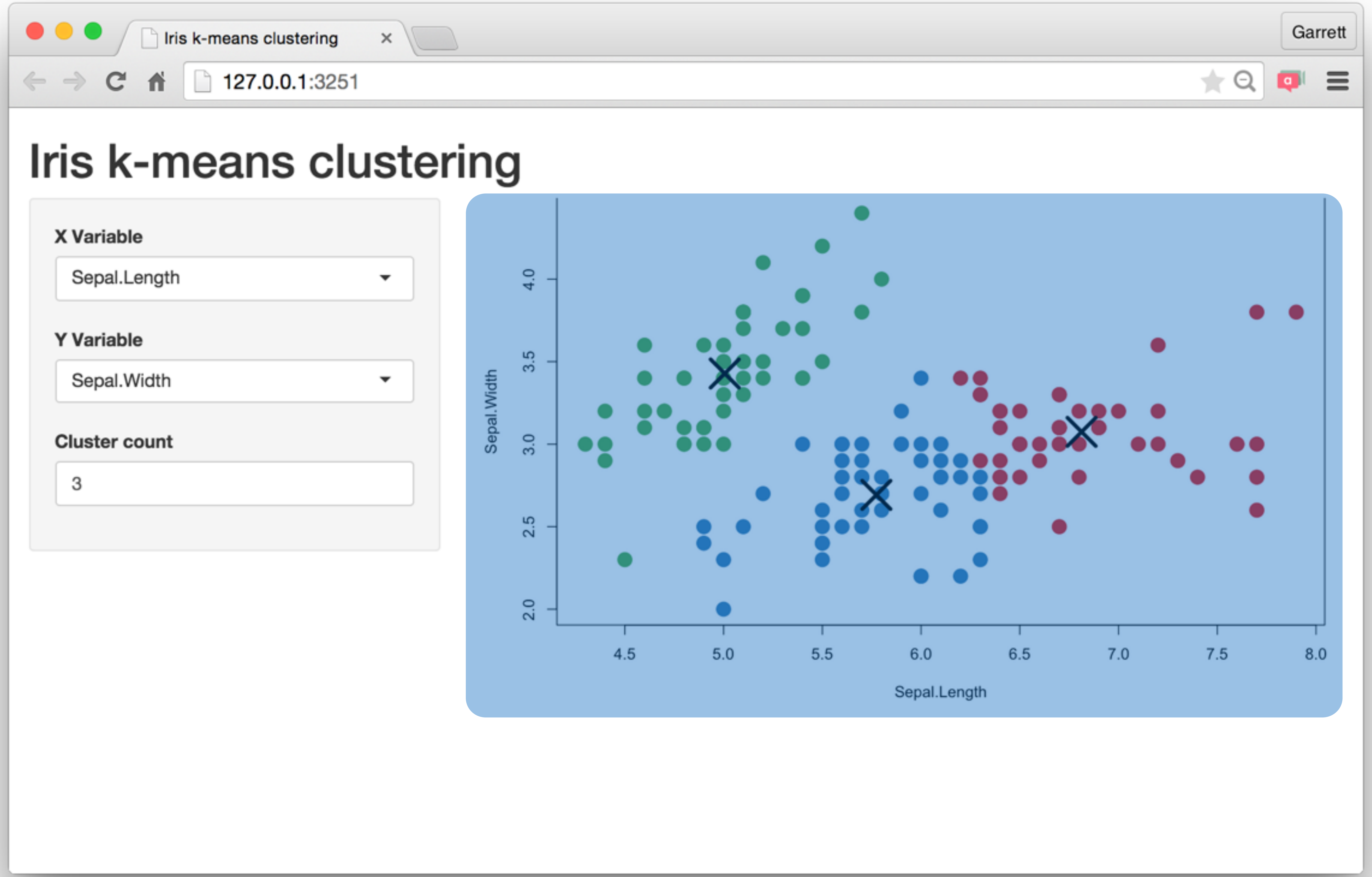
label to
display

input specific
arguments

```
?sliderInput
```

Outputs

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



Iris k-means clustering

X Variable

Sepal.Length

Y Variable

Sepal.Width

Cluster count

3

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

*Output()

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

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

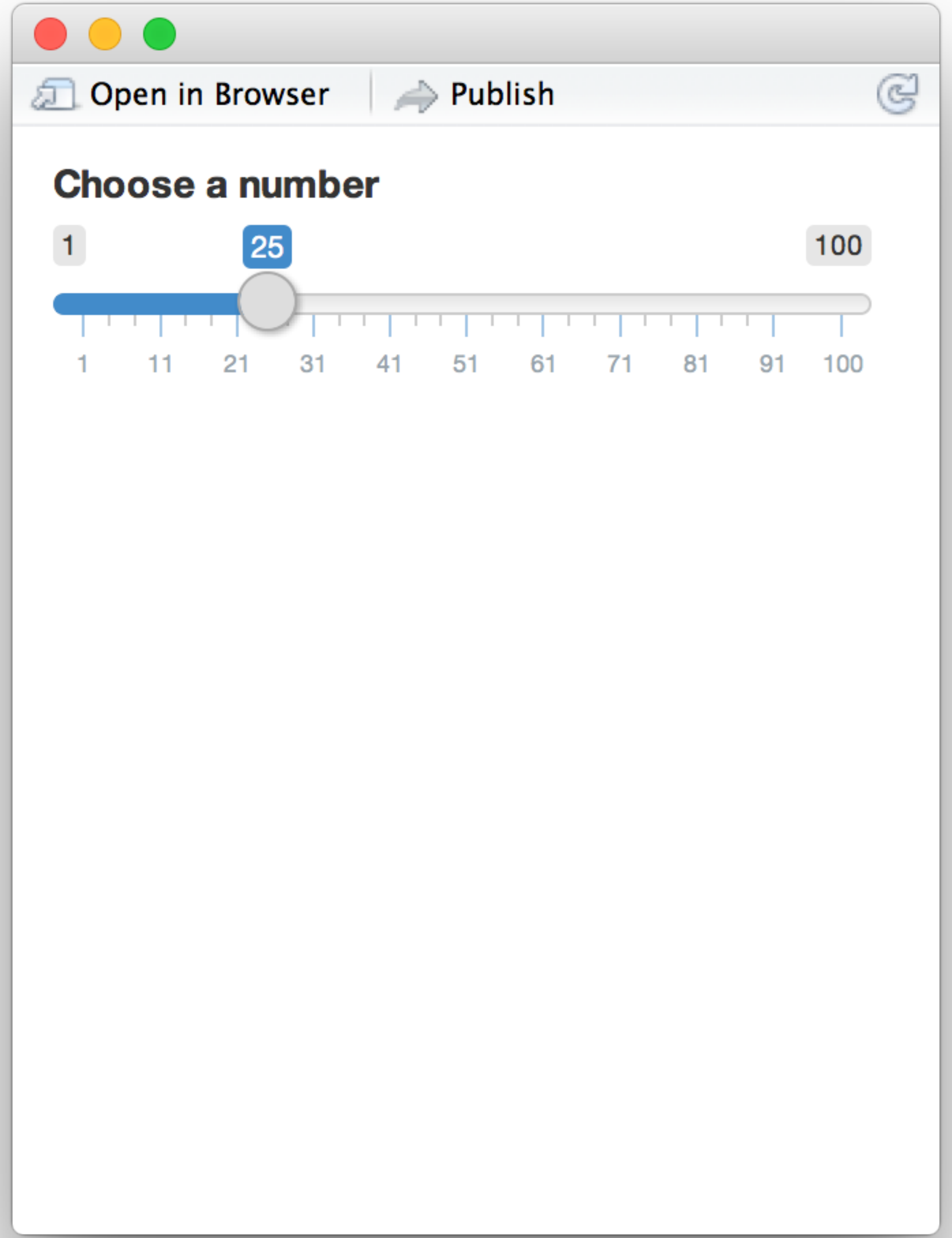
the type of output to display

name to give to the output object

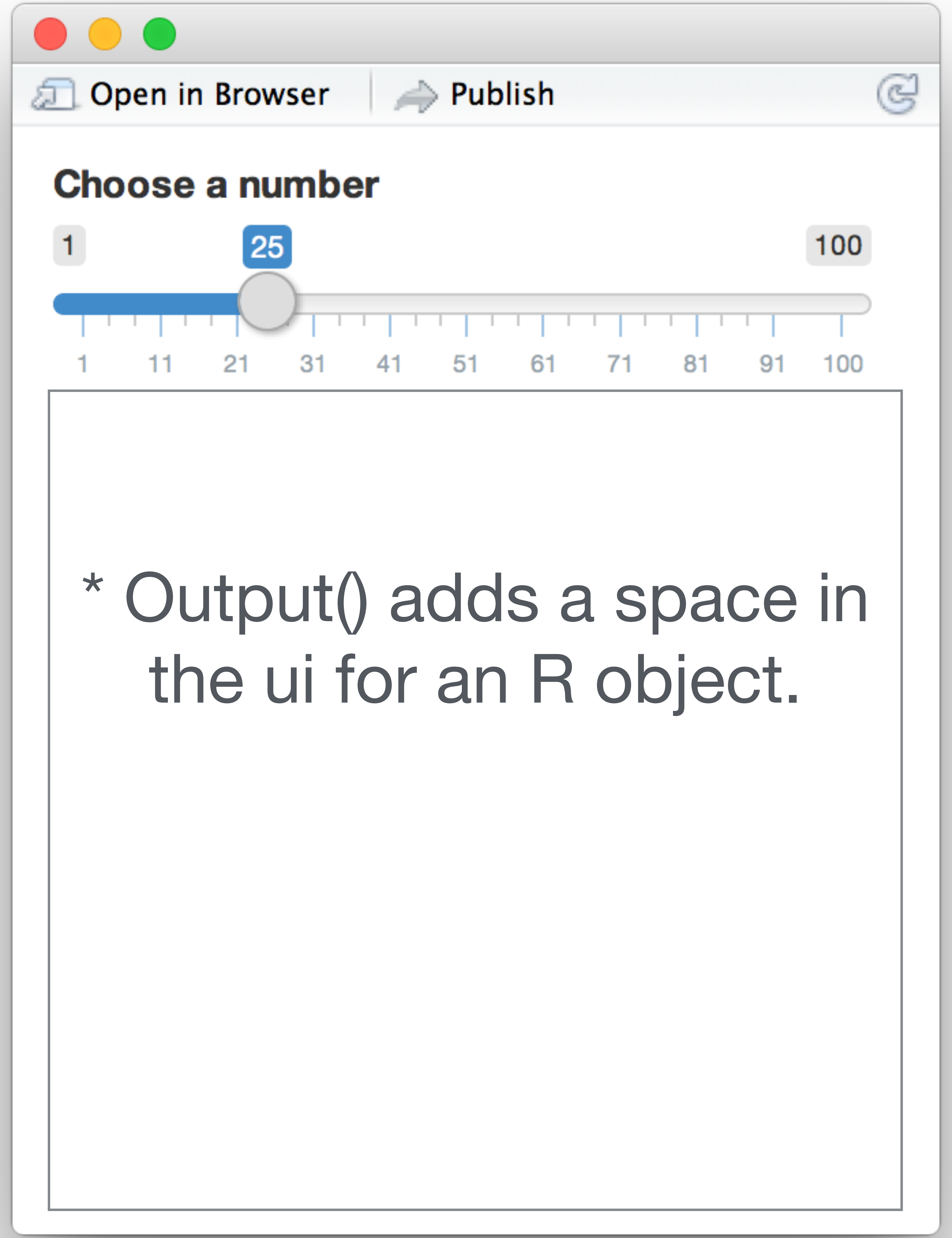
```r
library(shiny)

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

server <- function(input, output) {}

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

Comma between arguments

```r
library(shiny)

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

server <- function(input, output) {}

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

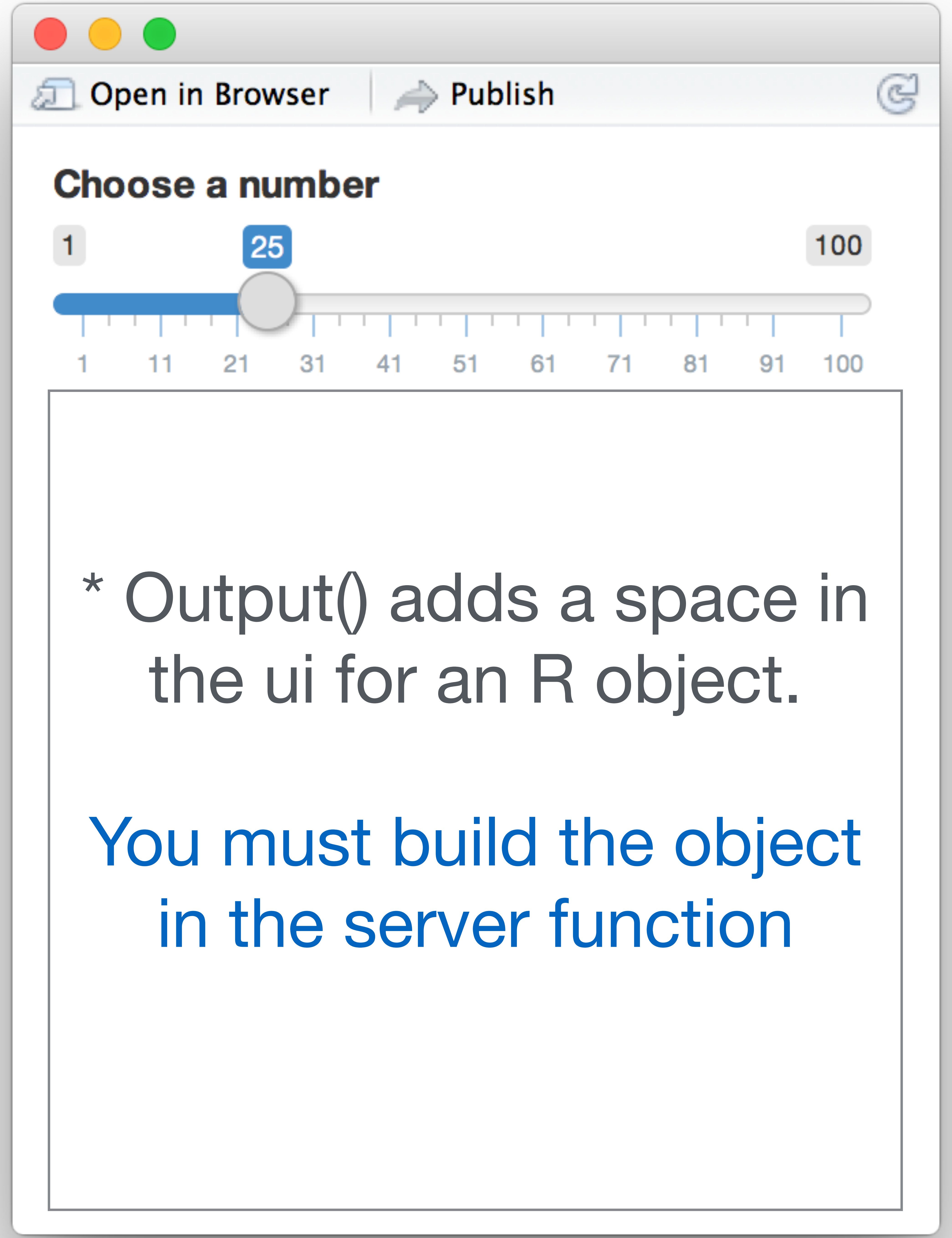


```r
library(shiny)

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

server <- function(input, output) {}

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



* Output() adds a space in the ui for an R object.

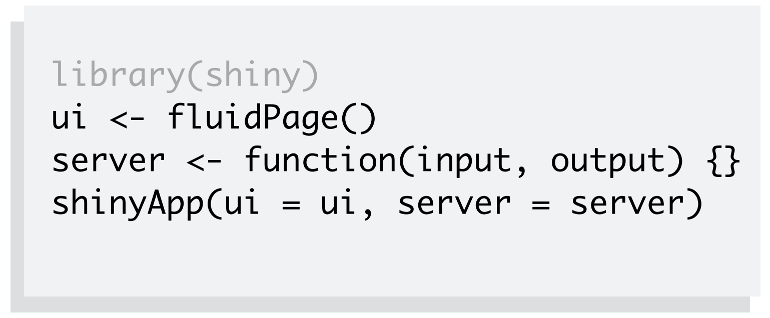
```r
library(shiny)

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

server <- function(input, output) {}

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



* Output() adds a space in the ui for an R object.

You must build the object in the server function

Recap

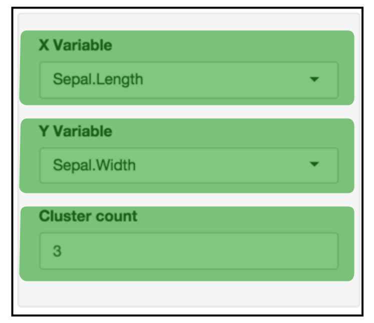


```
library(shiny)
ui <- fluidPage()
server <- function(input, output) {}
shinyApp(ui = ui, server = server)
```

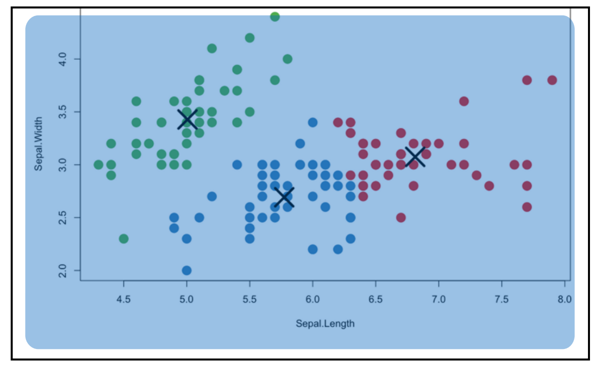
Begin each app with the template

Hello World

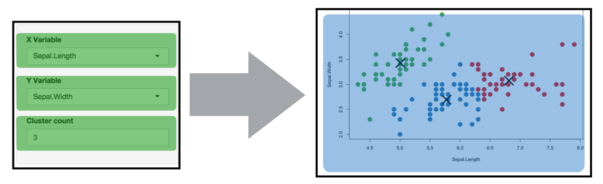
Add elements as arguments to **fluidPage()**



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) {



}
```

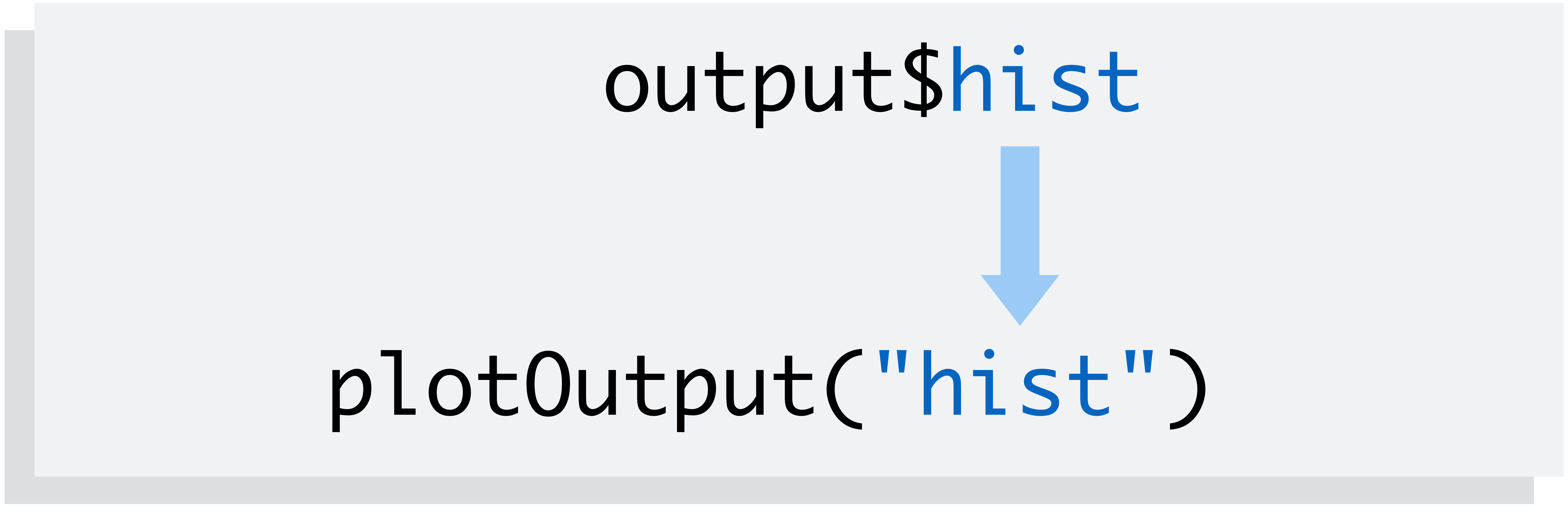
1

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 objects to display with render*()

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

  })
}
```

Use the **render*()** function that creates the type of output you wish to make.

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

render*()

Builds reactive output to display in UI

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

type of object to build

code block that builds the object

2

Build objects to display with **render*()**

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

2

Build objects to display with **render*()**

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

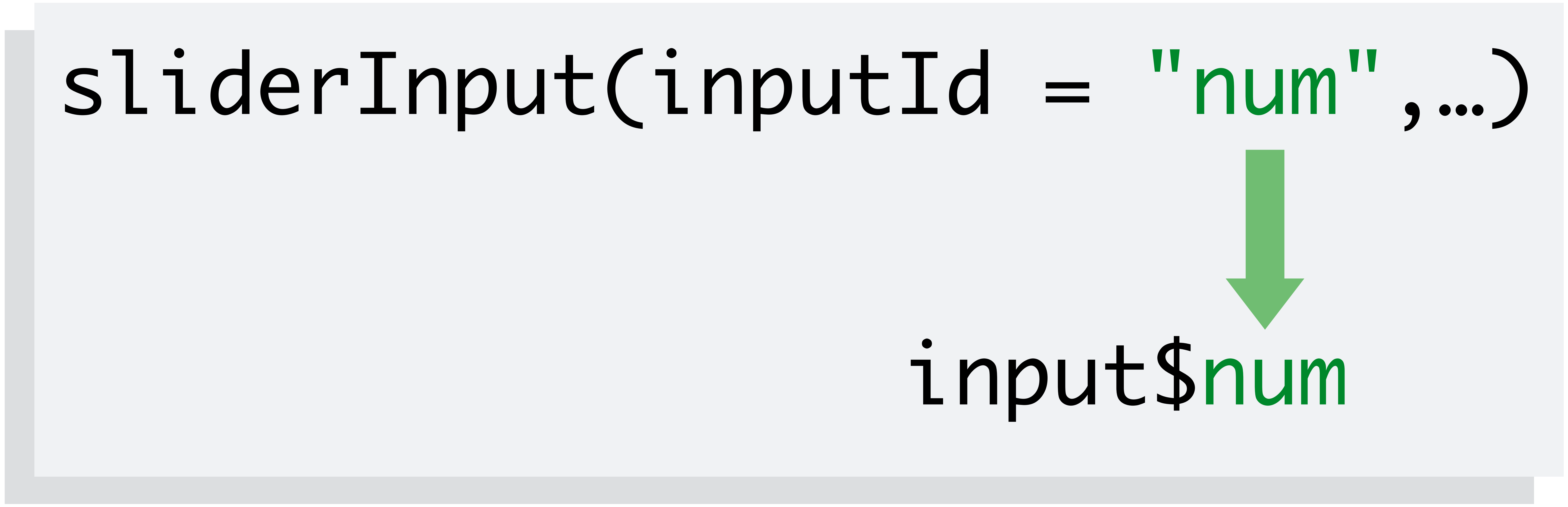
3

Access **input** values with input$

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

3

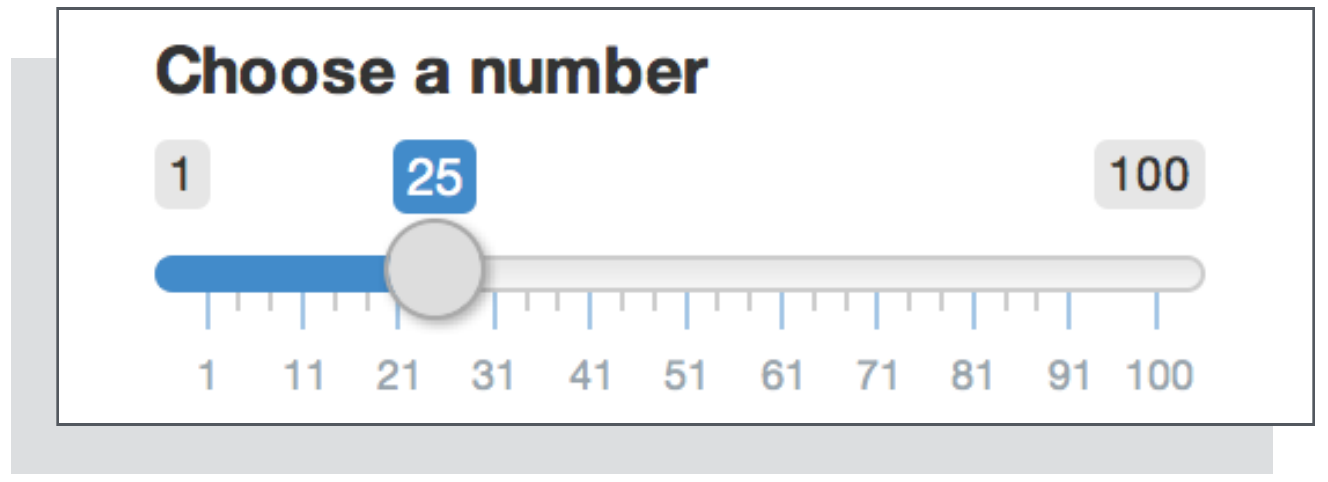
Access **input** values with input$



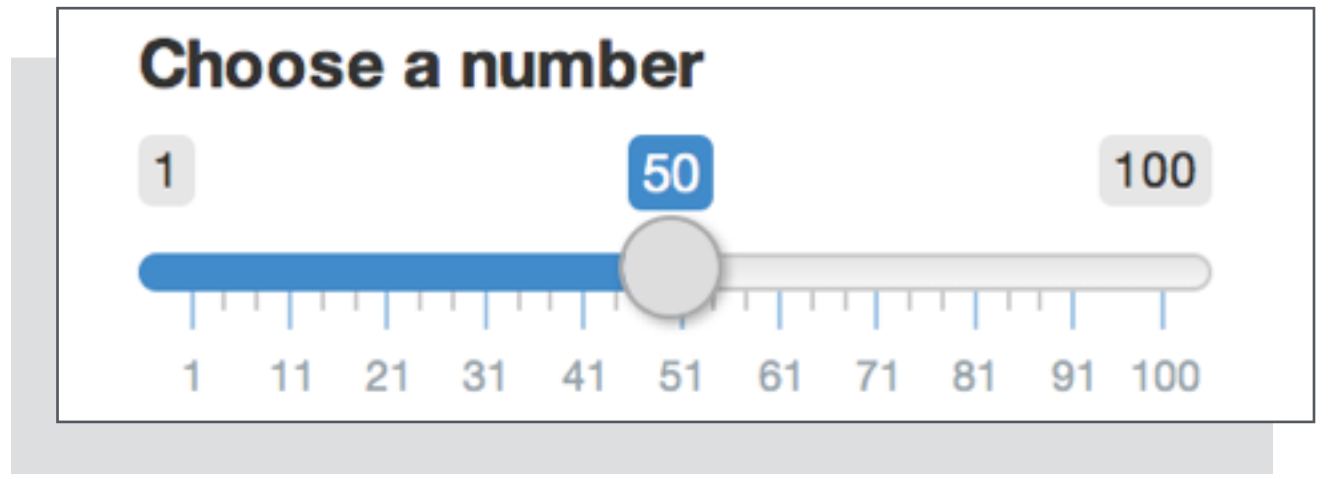
```
sliderInput(inputId = "num",…)

                     input$num
```

Input values

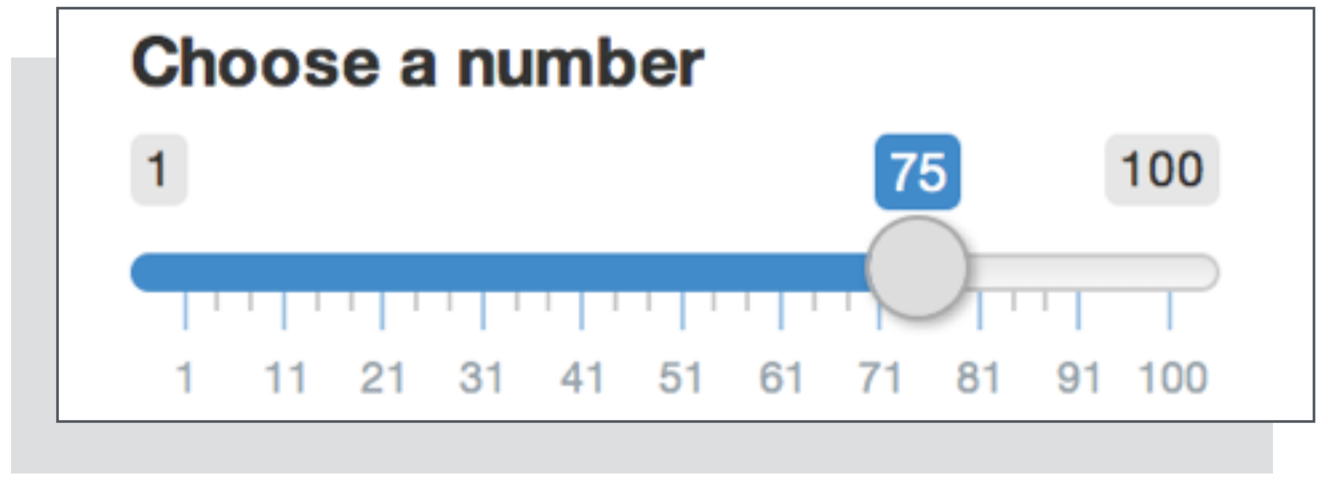
The input value changes whenever a user changes the input.



```
input$num = 25
```



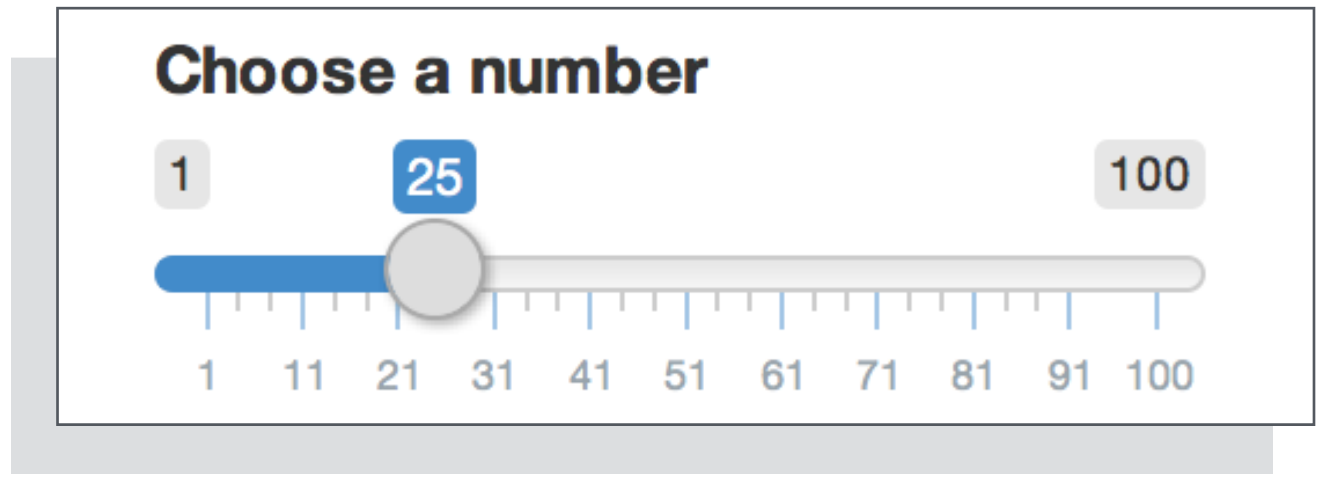
```
input$num = 50
```



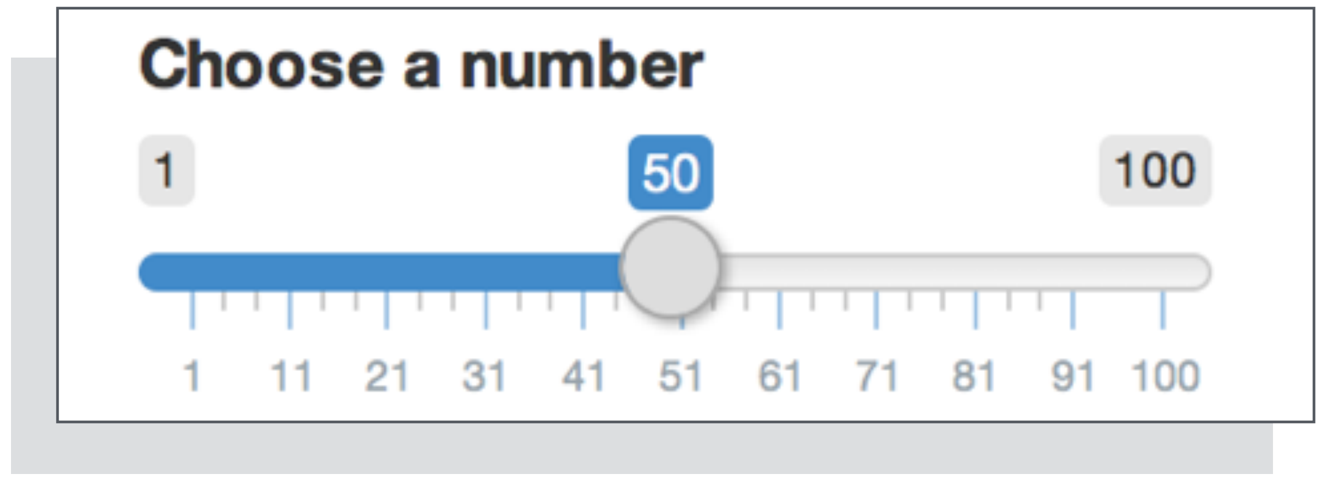
```
input$num = 75
```

Input values

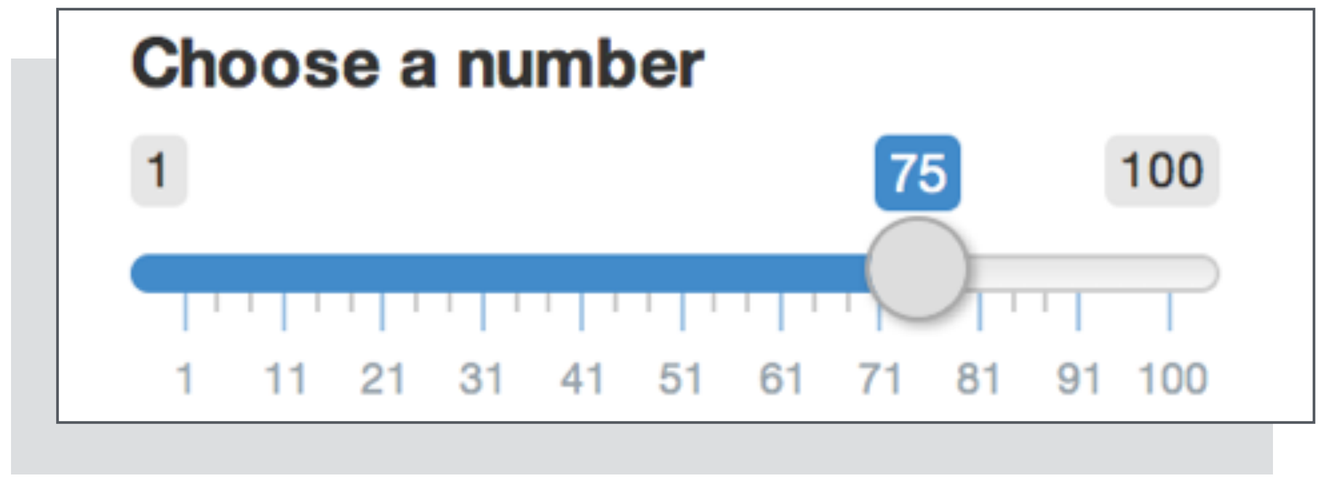
The input value changes whenever a user changes the input.



```
input$num = 25
```



```
input$num = 50
```



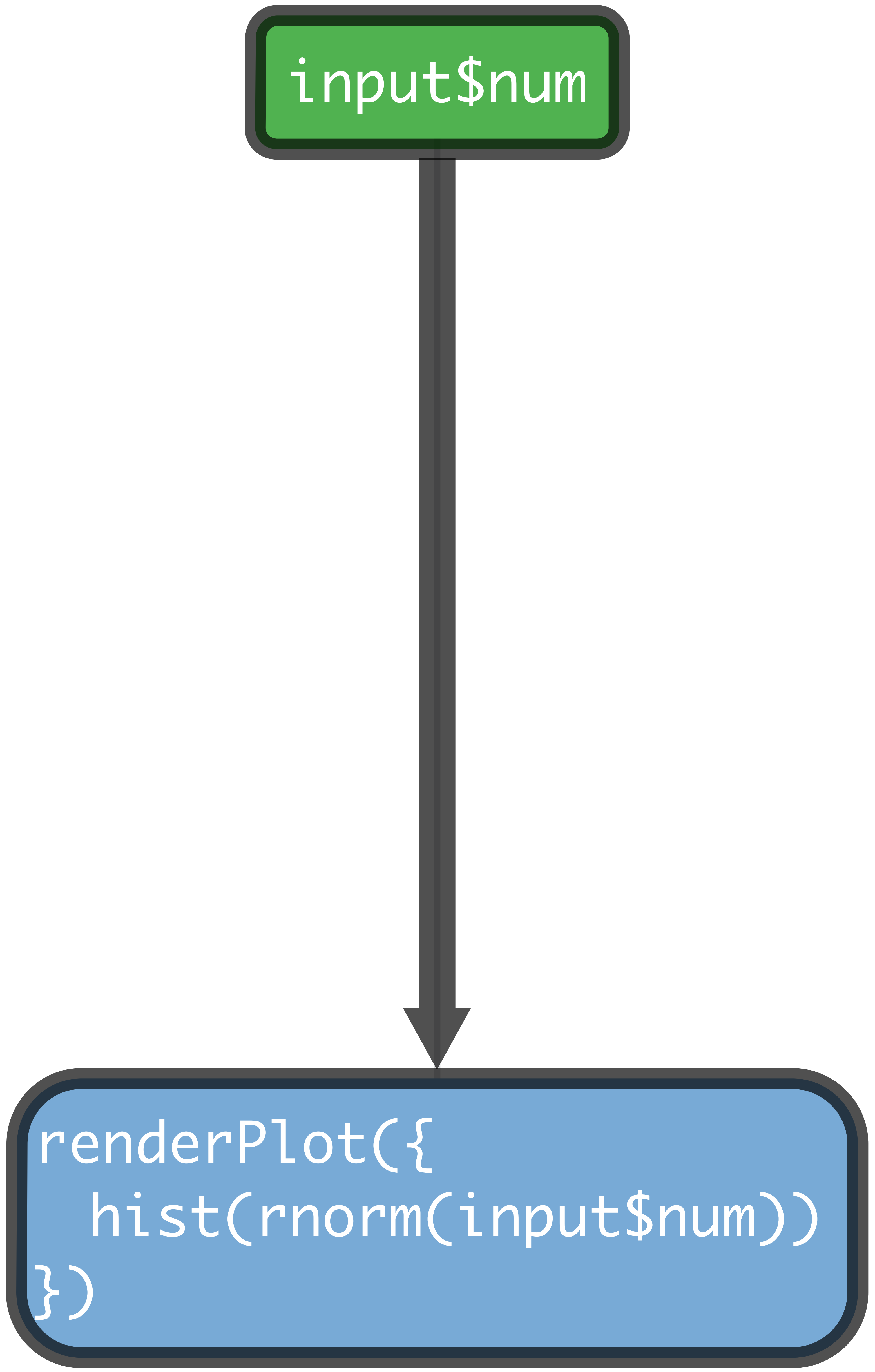
```
input$num =
```

Output will automatically update if you follow the 3 rules

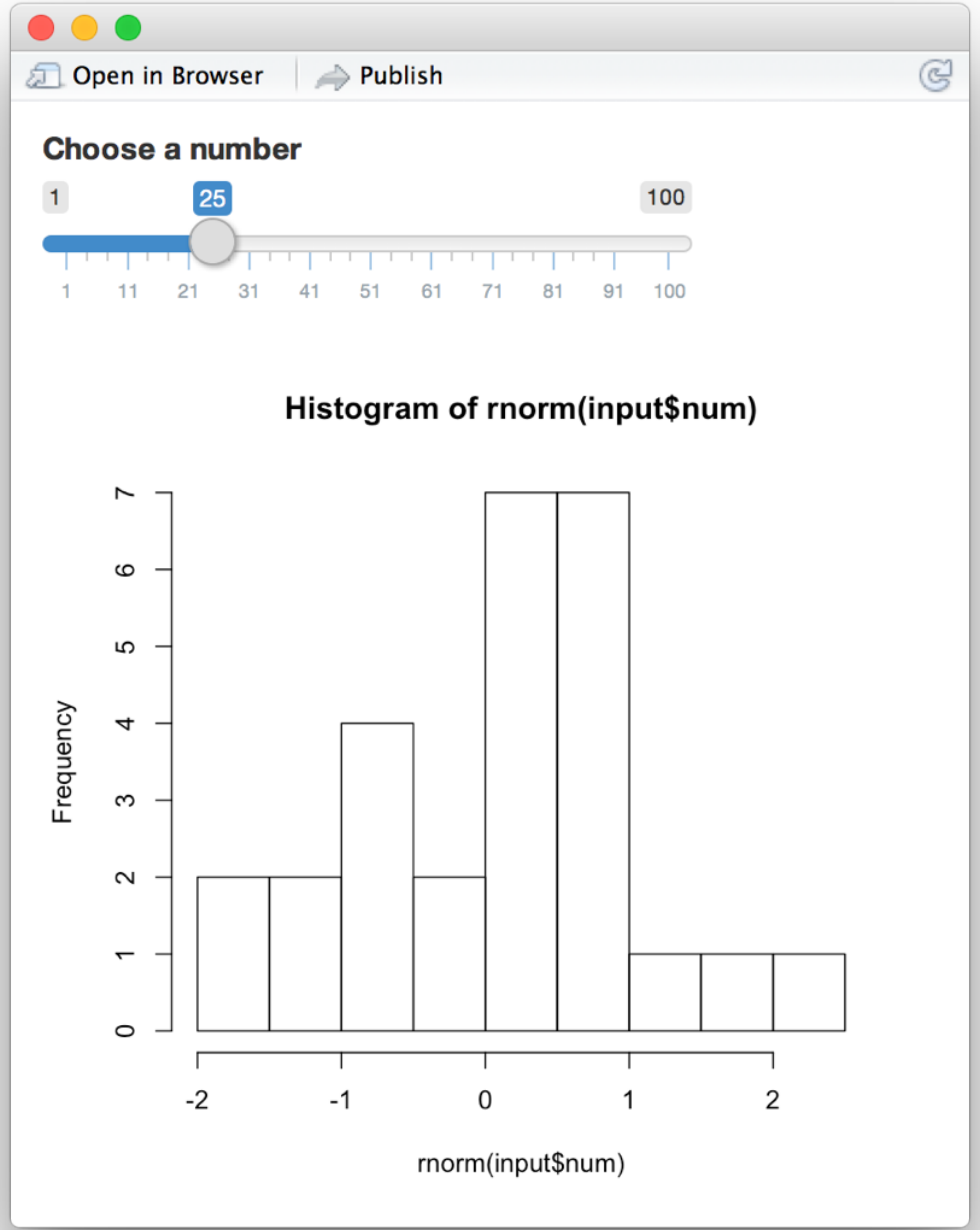
Reactivity 101

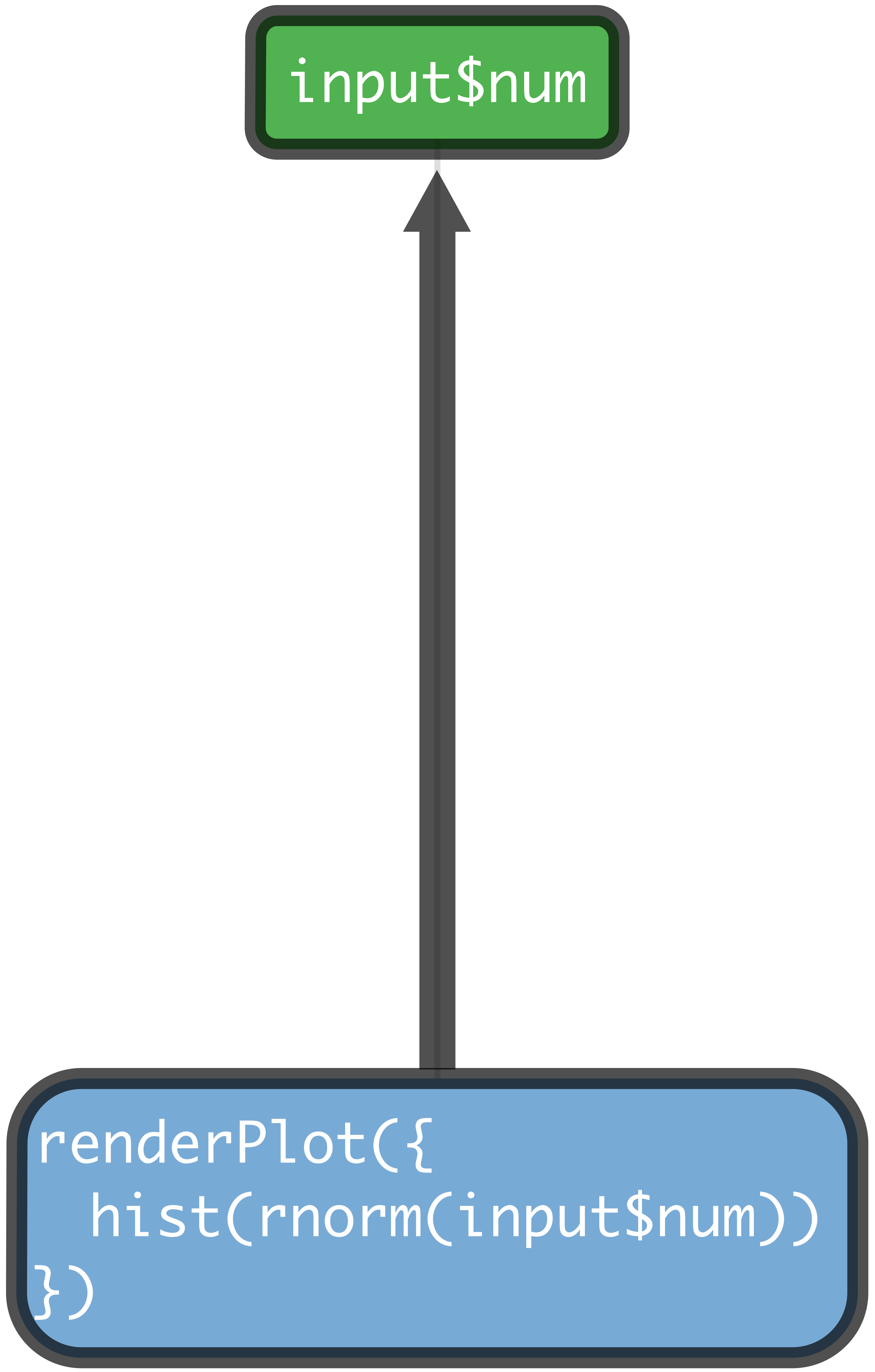
Reactivity automatically occurs whenever you use an input value to render an output object

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

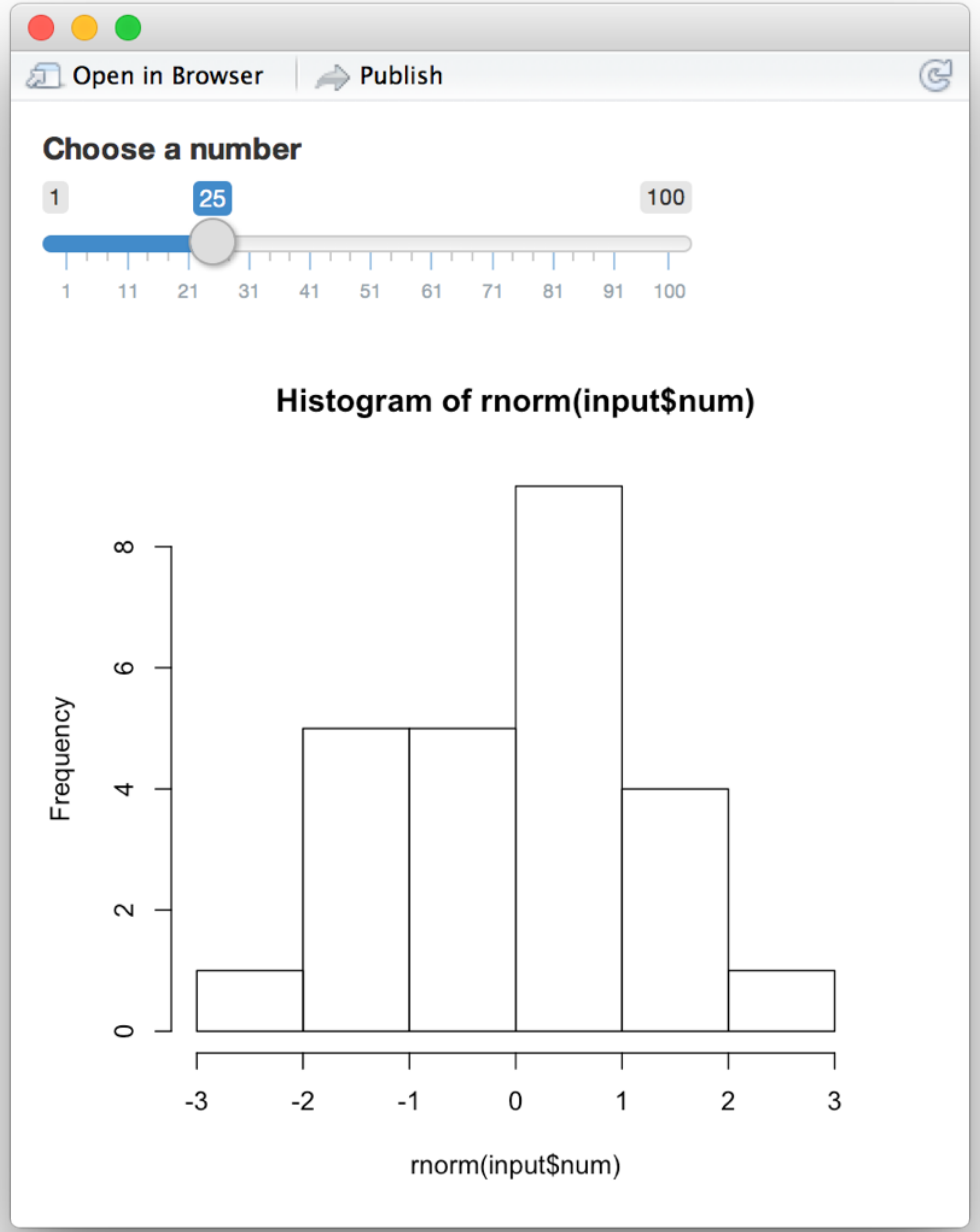


```
renderPlot({
  hist(rnorm(input$num))
})
```





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



Recap: Server

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

```
output$hist <-
```

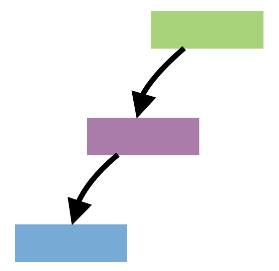
1. Save the output that you build to **output$**

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

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

```
input$num
```

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

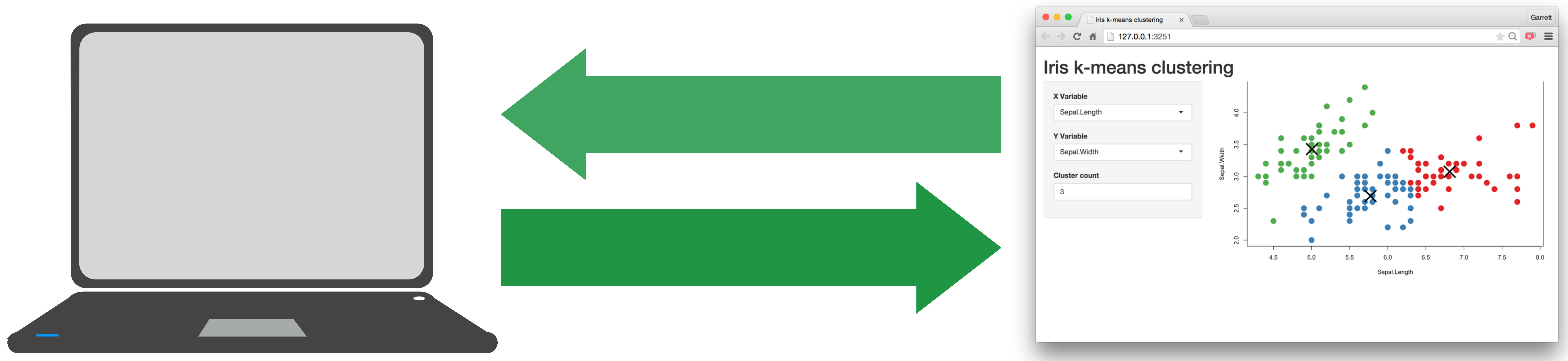


Create reactivity by using **Inputs** to build **rendered Outputs**

Share

your app

Every Shiny app is maintained by a computer running R



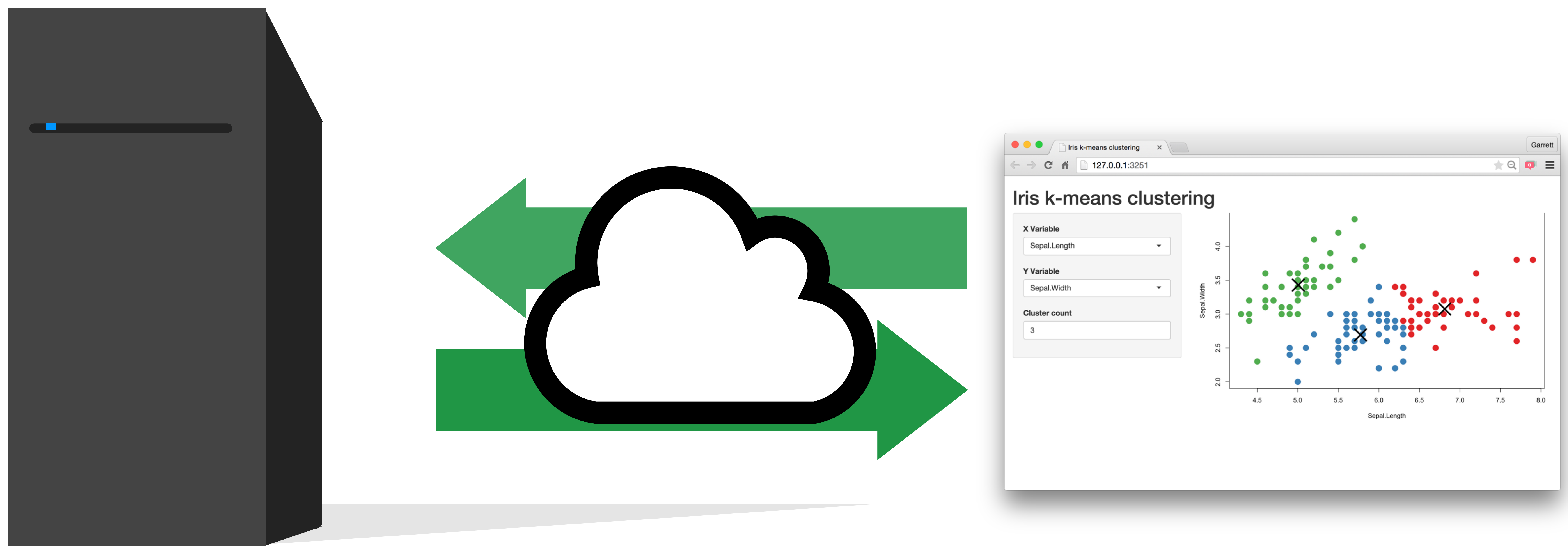
Iris k-means clustering

X Variable: Sepal.Length

Y Variable: Sepal.Width

Cluster count: 3

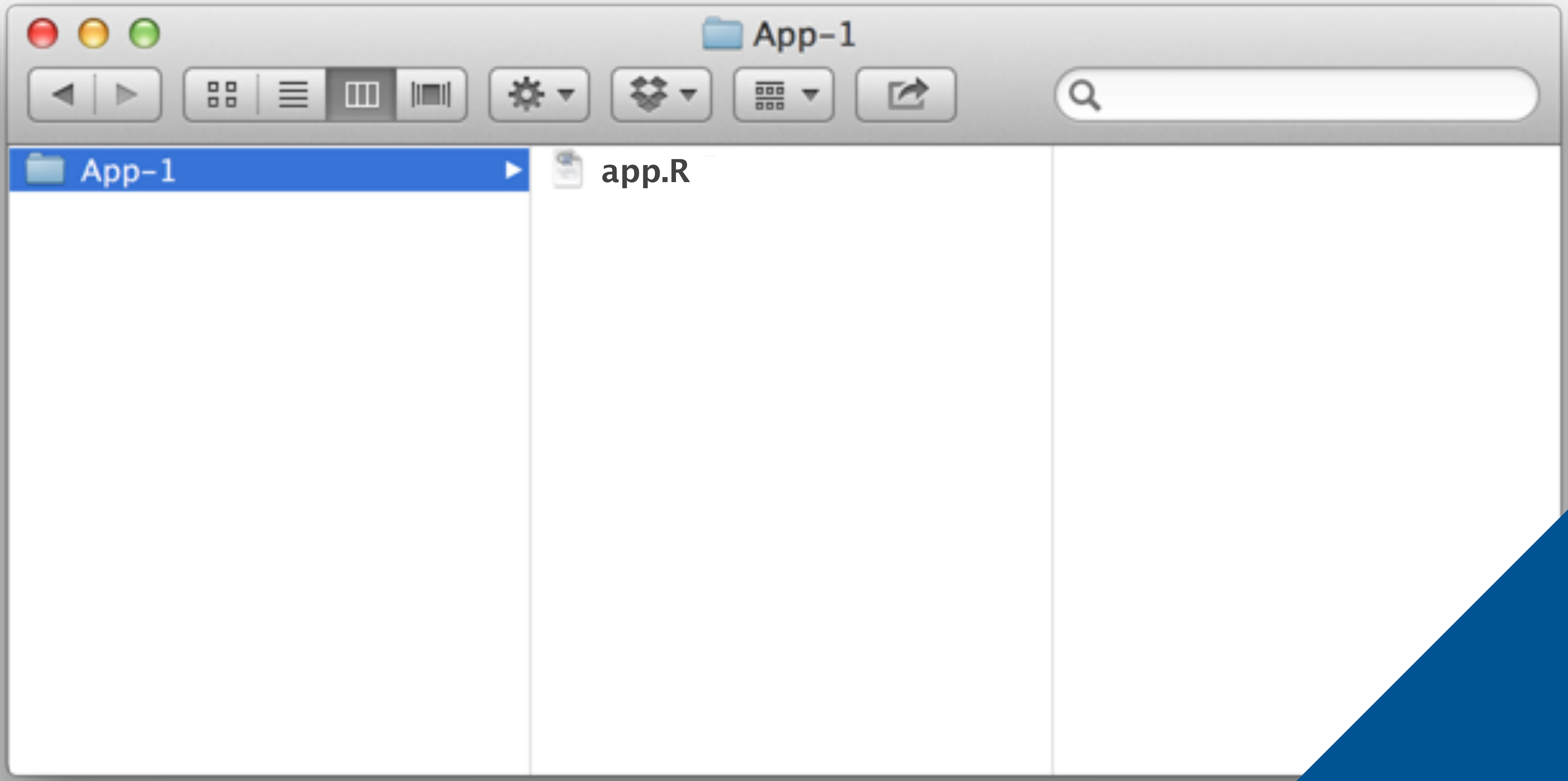
Every Shiny app is maintained by a computer running R



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 must use this exact name (**app.R**)

Two file apps

```r
library(shiny)

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

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

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

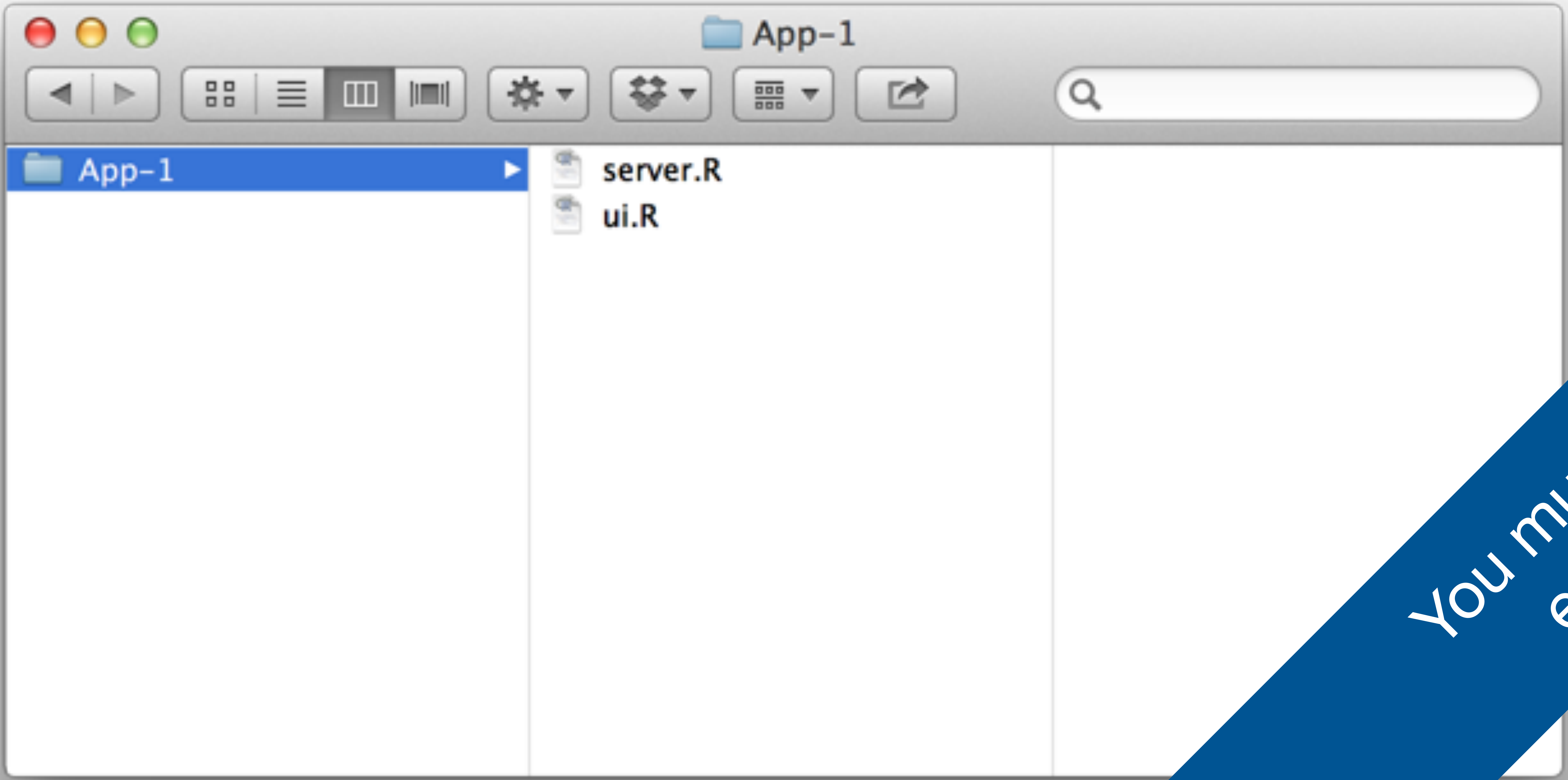
```r
# ui.R
library(shiny)
fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  plotOutput("hist")
)
```

```r
# server.R
library(shiny)
function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
}
```

Two file apps

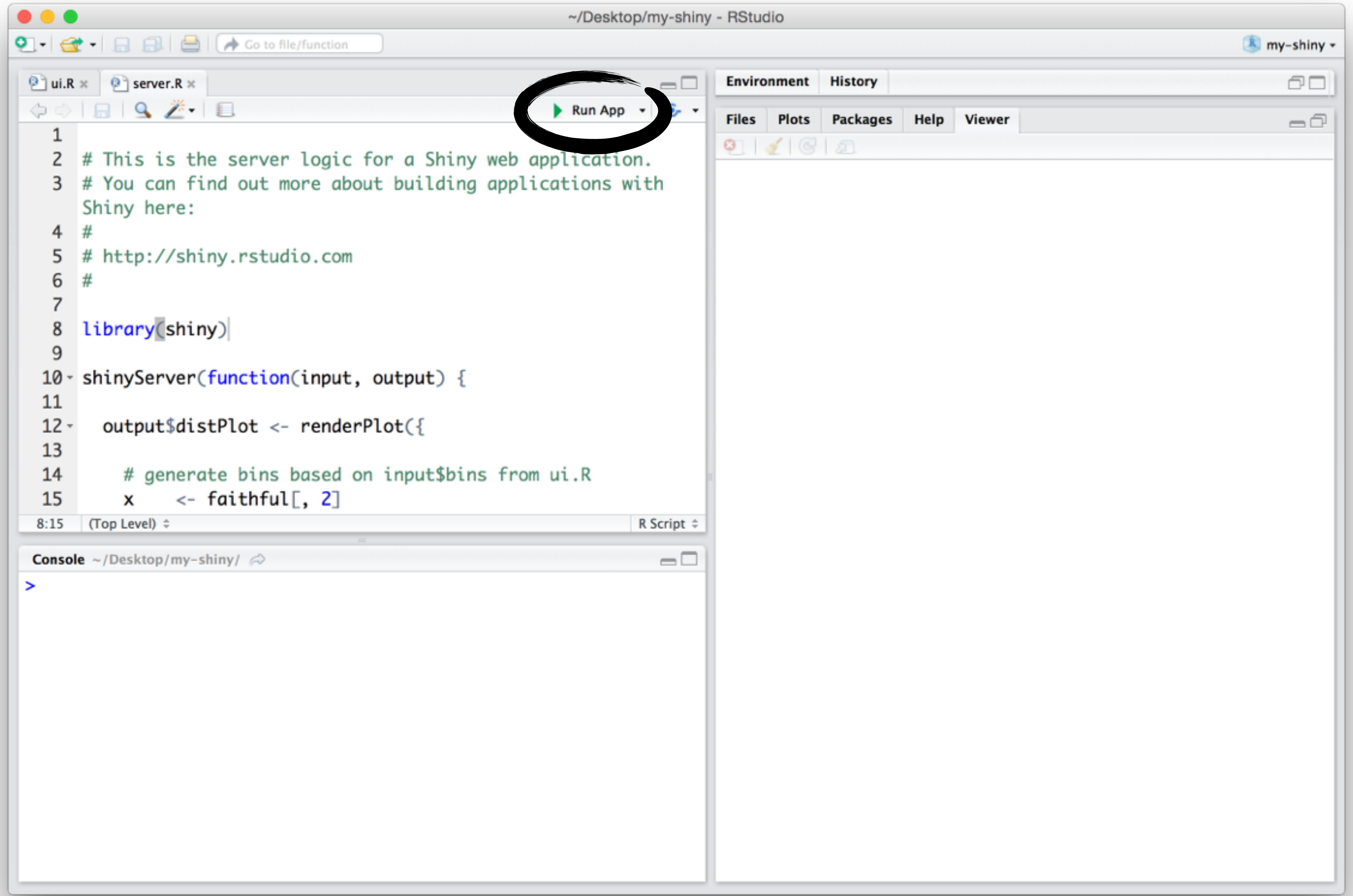
One directory with two files:

- server.R
- ui.R



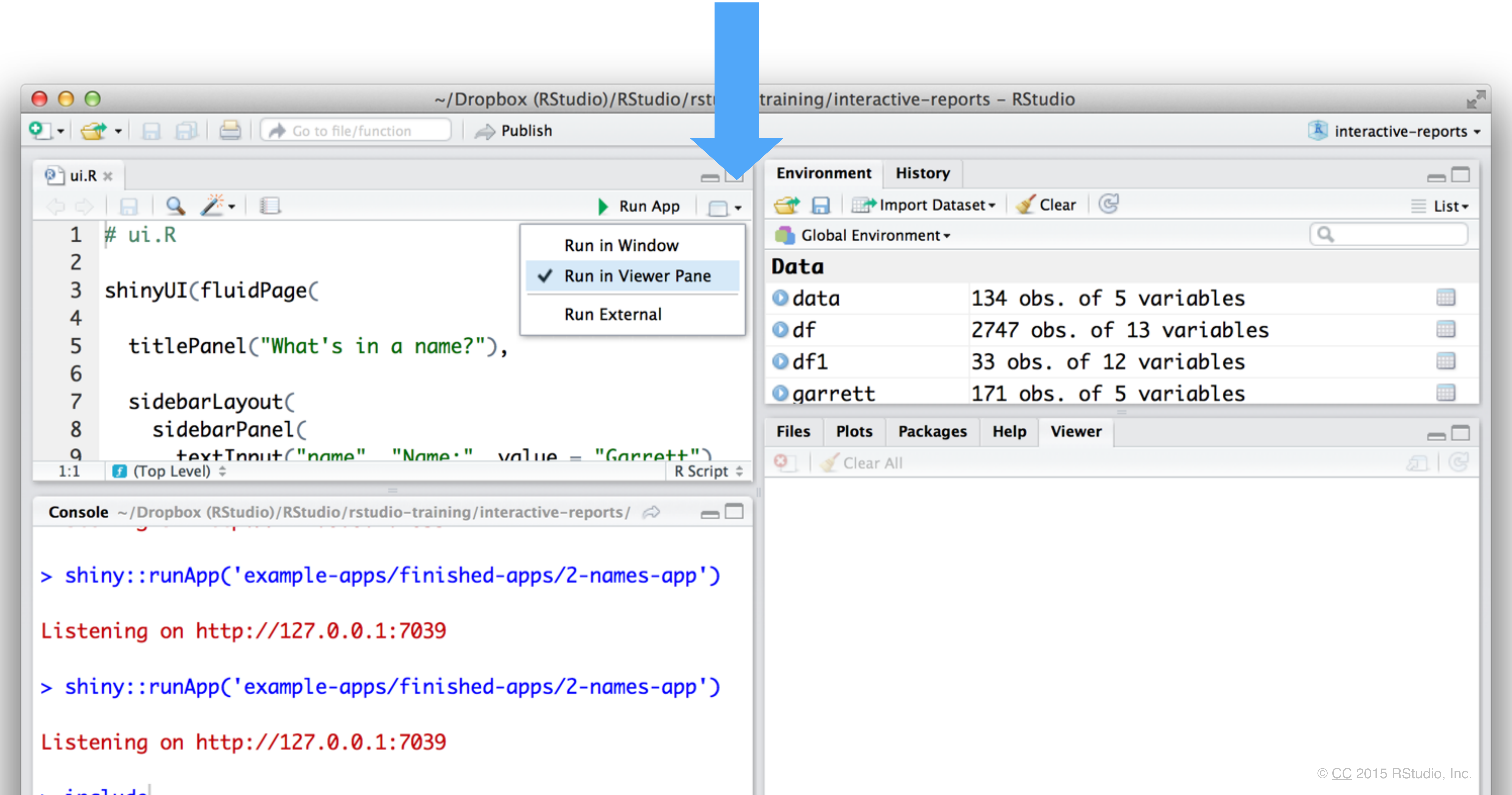
You must use these exact names

Launch an app



```

# This is the server logic for a Shiny web application.
# You can find out more about building applications with
Shiny here:
#
# http://shiny.rstudio.com
#

library(shiny)

shinyServer(function(input, output) {

  output$distPlot <- renderPlot({

    # generate bins based on input$bins from ui.R
    x    <- faithful[, 2]
```

Display options



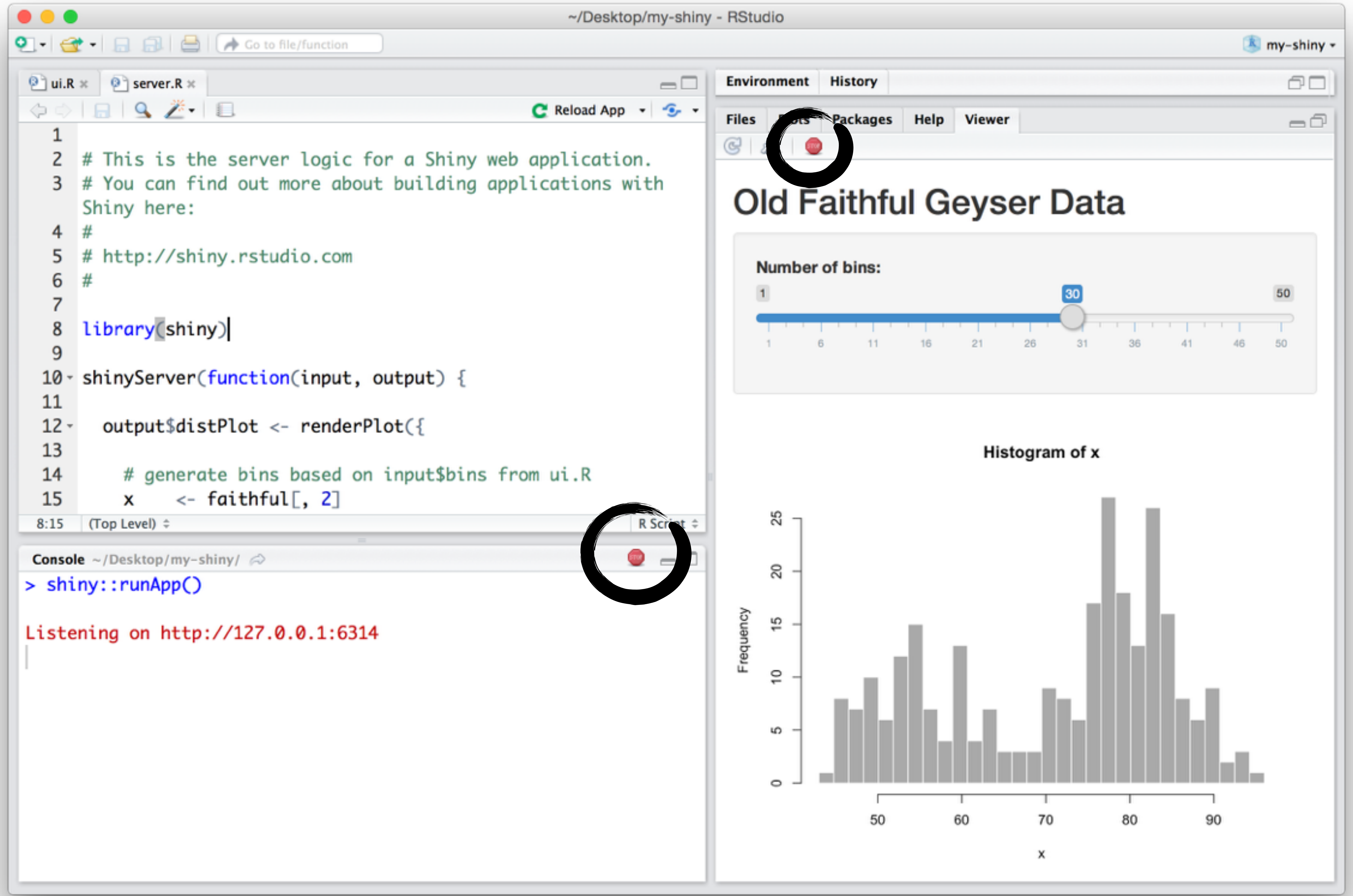
```
# ui.R

shinyUI(fluidPage(

  titlePanel("What's in a name?"),

  sidebarLayout(
    sidebarPanel(
      textInput("name", "Name:", value = "Garrett")
```

Data

data	134 obs. of 5 variables
df	2747 obs. of 13 variables
df1	33 obs. of 12 variables
garrett	171 obs. of 5 variables

```
> shiny::runApp('example-apps/finished-apps/2-names-app')

Listening on http://127.0.0.1:7039

> shiny::runApp('example-apps/finished-apps/2-names-app')

Listening on http://127.0.0.1:7039

> include
```

Close an app



```
# This is the server logic for a Shiny web application.
# You can find out more about building applications with
Shiny here:
#
# http://shiny.rstudio.com
#

library(shiny)

shinyServer(function(input, output) {

  output$distPlot <- renderPlot({

    # generate bins based on input$bins from ui.R
    x    <- faithful[, 2]
```

```
> shiny::runApp()

Listening on http://127.0.0.1:6314
```

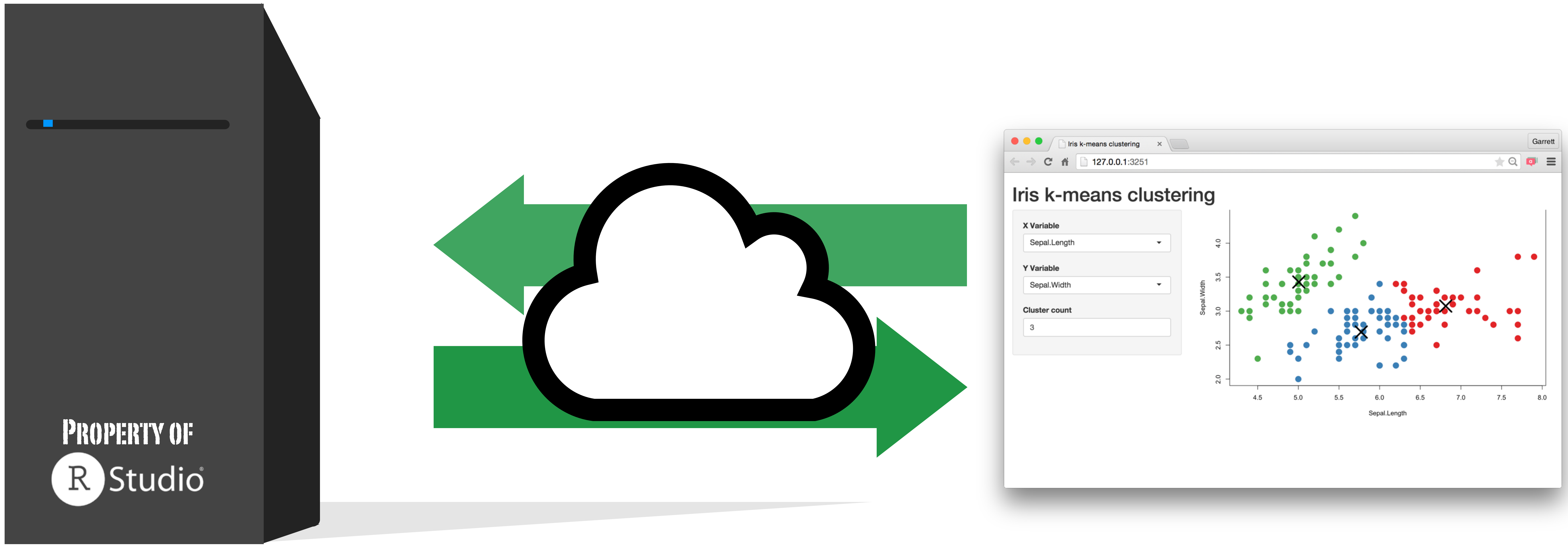
Use shinyapps.io

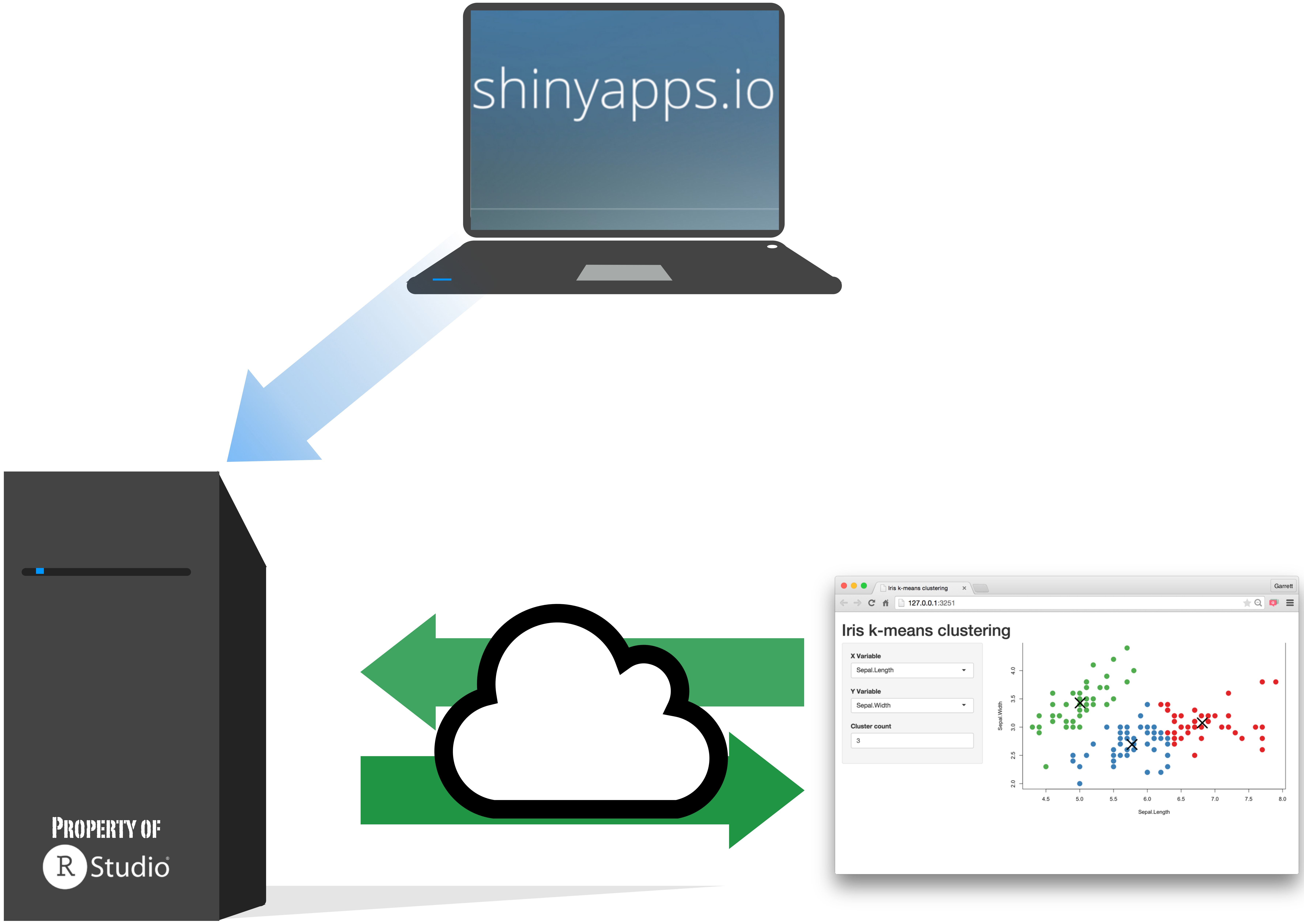
by RStudio®

Shinyapps.io

A server maintained by RStudio

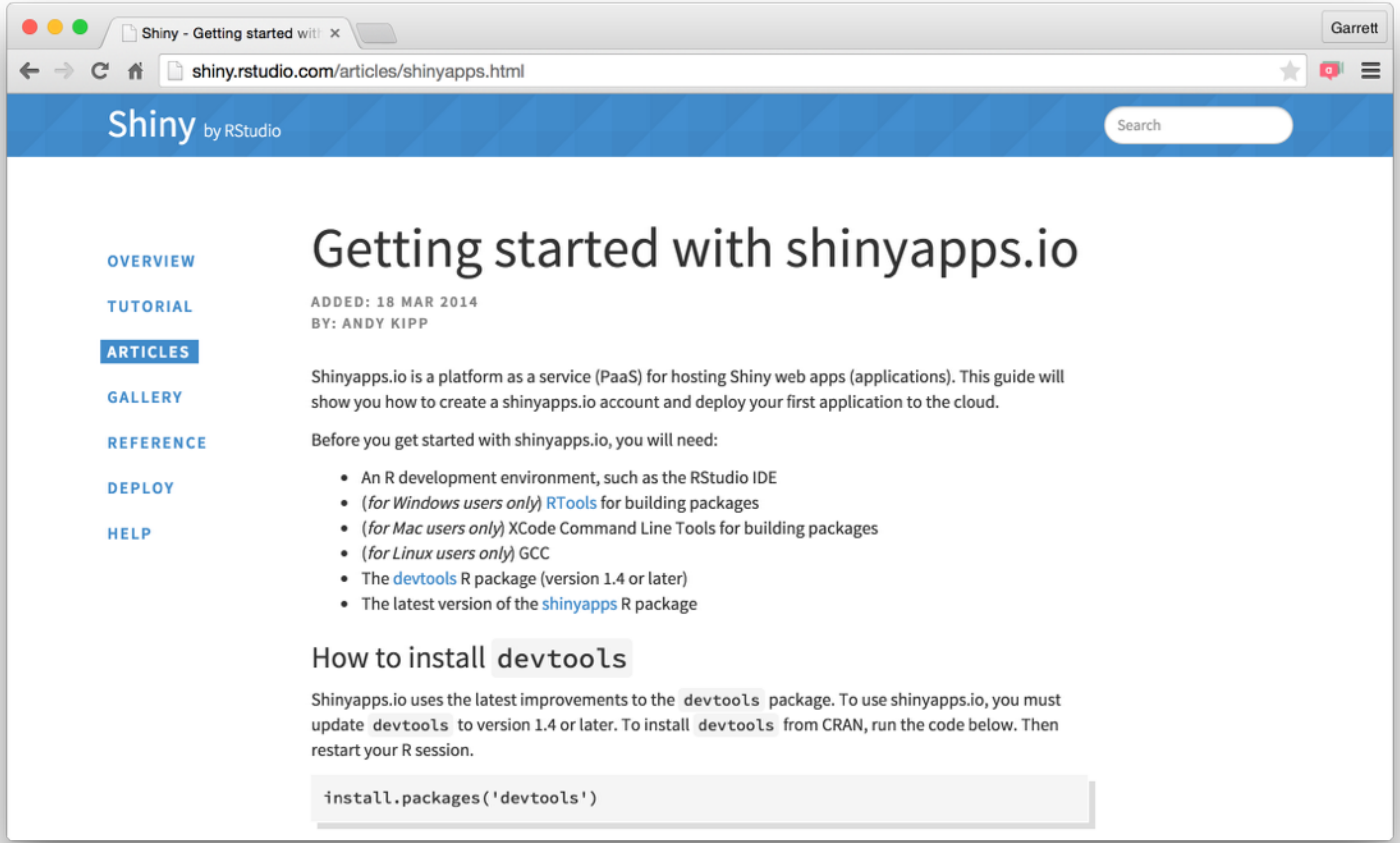
- free
- easy to use
- secure
- scalable




shinyapps.io
PROPERTY OF
RStudio
Iris k-means clustering
127.0.0.1:3251
X Variable
Sepal.Length
Y Variable
Sepal.Width
Cluster count
3

Getting started guide

shiny.rstudio.com/articles/shinyapps.html



Getting started with shinyapps.io

ADDED: 18 MAR 2014
BY: ANDY KIPP

Shinyapps.io is a platform as a service (PaaS) for hosting Shiny web apps (applications). This guide will show you how to create a shinyapps.io account and deploy your first application to the cloud.

Before you get started with shinyapps.io, you will need:

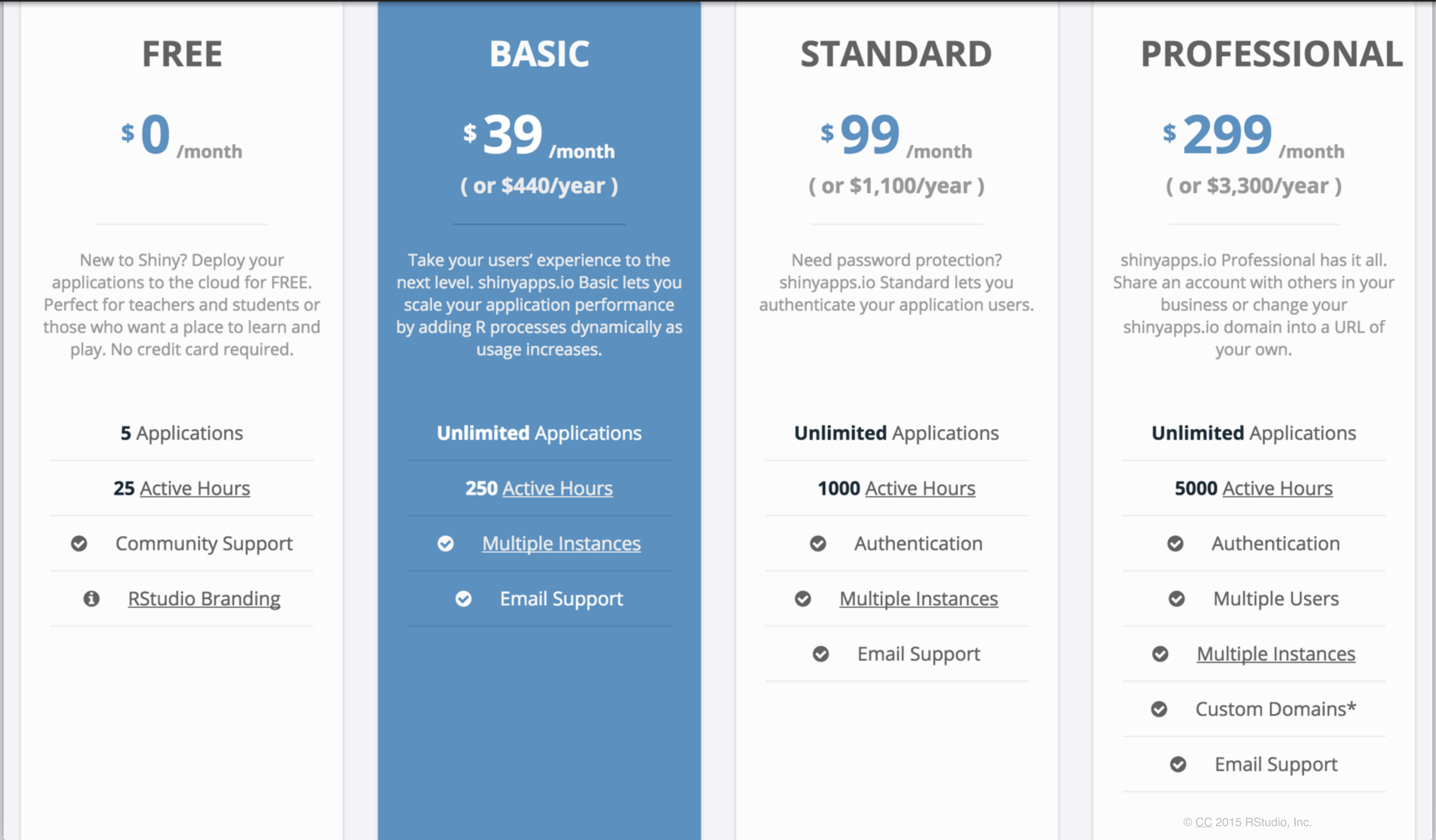
- An R development environment, such as the RStudio IDE
- (*for Windows users only*) RTools for building packages
- (*for Mac users only*) XCode Command Line Tools for building packages
- (*for Linux users only*) GCC
- The devtools R package (version 1.4 or later)
- The latest version of the shinyapps R package

How to install `devtools`

Shinyapps.io uses the latest improvements to the `devtools` package. To use shinyapps.io, you must update `devtools` to version 1.4 or later. To install `devtools` from CRAN, run the code below. Then restart your R session.

```
install.packages('devtools')
```

RStudio



FREE	BASIC	STANDARD	PROFESSIONAL
$0 /month	$39 /month (or $440/year)	$99 /month (or $1,100/year)	$299 /month (or $3,300/year)
New to Shiny? Deploy your applications to the cloud for FREE. Perfect for teachers and students or those who want a place to learn and play. No credit card required.	Take your users' experience to the next level. shinyapps.io Basic lets you scale your application performance by adding R processes dynamically as usage increases.	Need password protection? shinyapps.io Standard lets you authenticate your application users.	shinyapps.io Professional has it all. Share an account with others in your business or change your shinyapps.io domain into a URL of your own.
5 Applications	**Unlimited** Applications	**Unlimited** Applications	**Unlimited** Applications
25 Active Hours	**250** Active Hours	**1000** Active Hours	**5000** Active Hours
✓ Community Support	✓ Multiple Instances	✓ Authentication	✓ Authentication
ℹ RStudio Branding	✓ Email Support	✓ Multiple Instances	✓ Multiple Users
		✓ Email Support	✓ Multiple Instances
			✓ Custom Domains*
			✓ Email Support

Build your own server

Shiny Server Free!

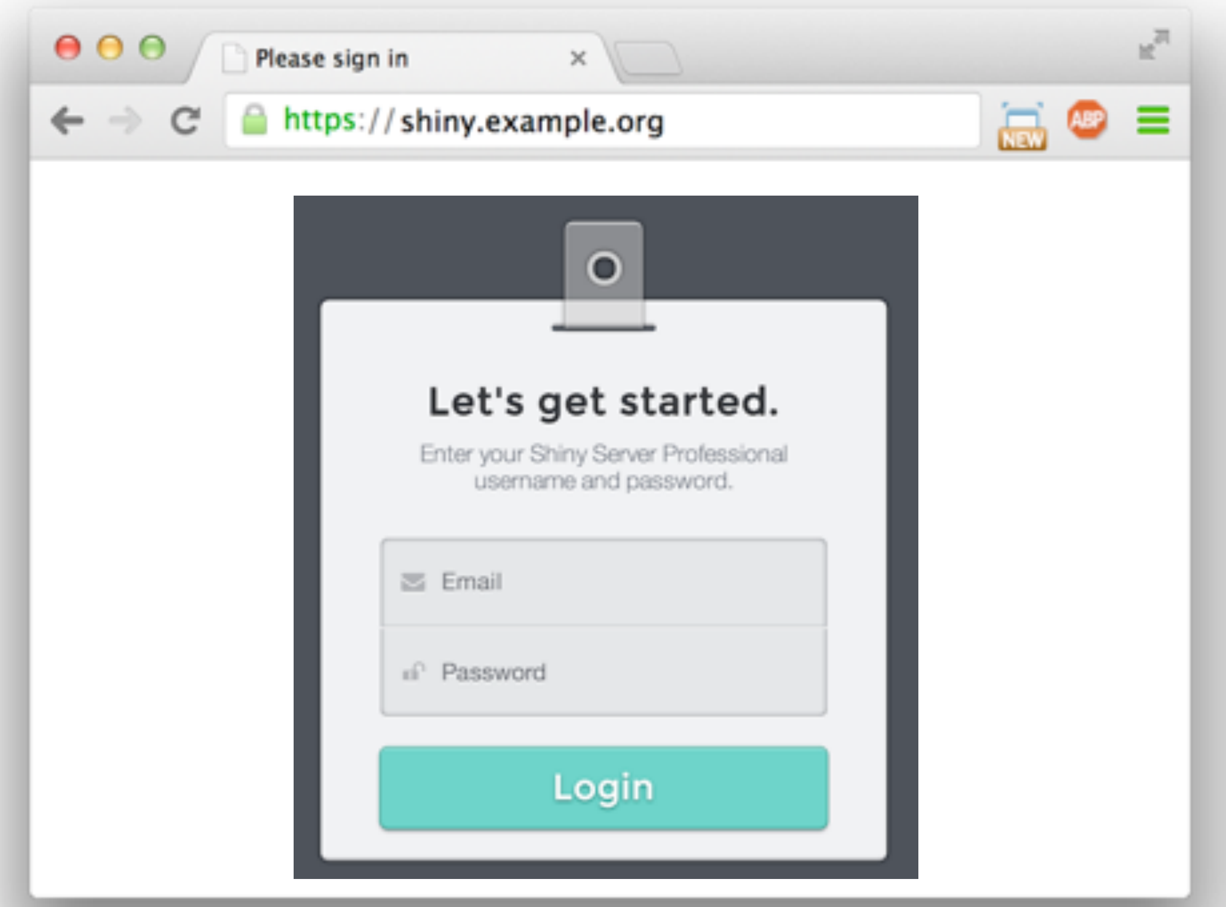
www.rstudio.com/products/shiny/shiny-server/

A back end program that builds a linux web server specifically designed to host Shiny apps.

Shiny Server **Pro**

www.rstudio.com/products/shiny/shiny-server/

- ✓ **Secure access** - LDAP, GoogleAuth, SSL, and more
- ✓ **Performance** - fine tune at app and server level
- ✓ **Management** - monitor and control resource use
- ✓ **Support** - direct priority support


Please sign in

https://shiny.example.org

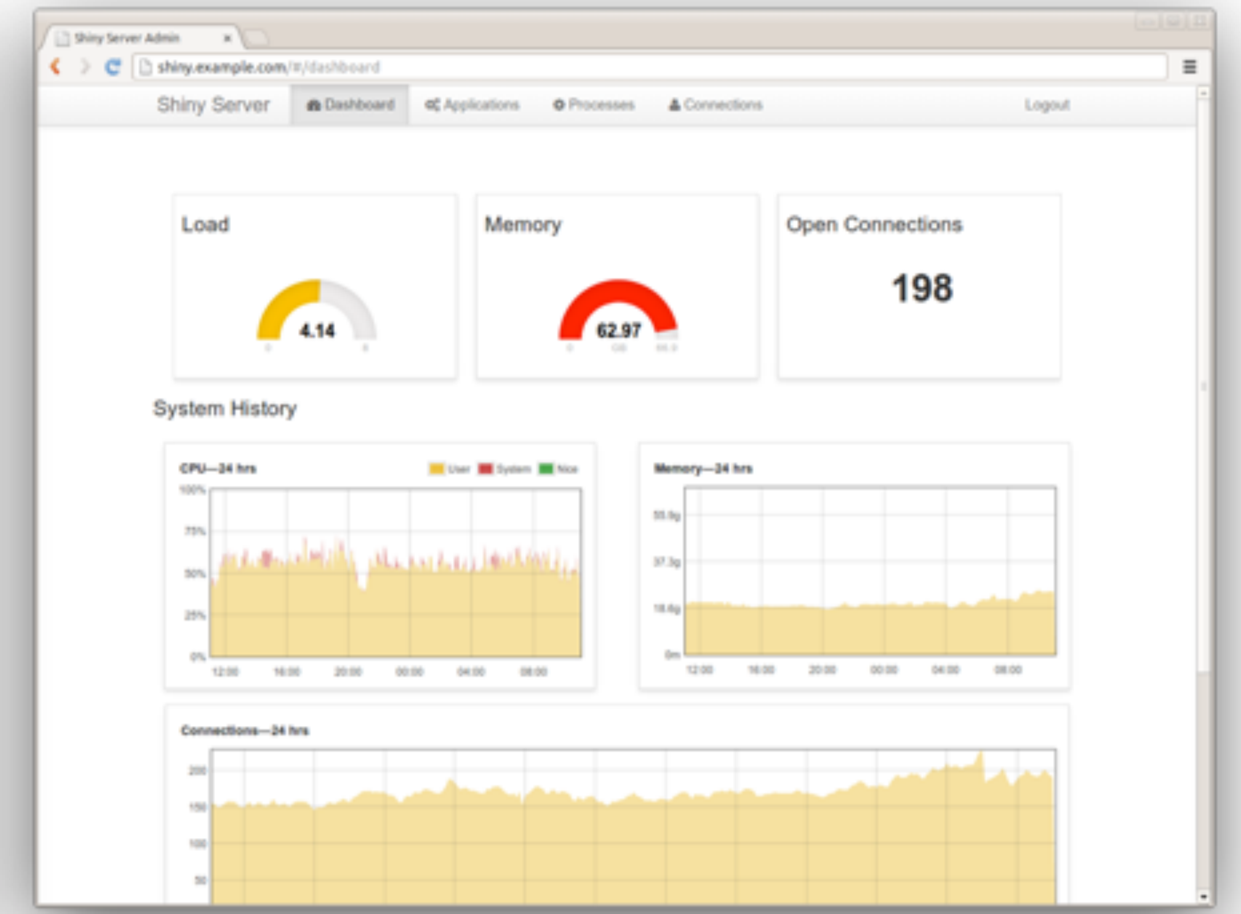
Let's get started.

Enter your Shiny Server Professional username and password.

Email

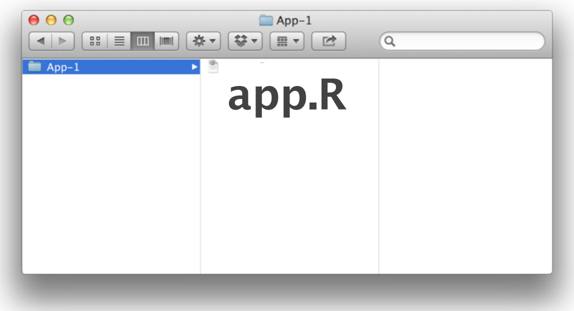
Password

Login



45 day evaluation free trial

Recap: Sharing



Save your app in its own directory as **app.R**, or **ui.R** and **server.R**

Host apps at **shinyapps.io** by:

1. Sign up for a free **shinyapps.io** account

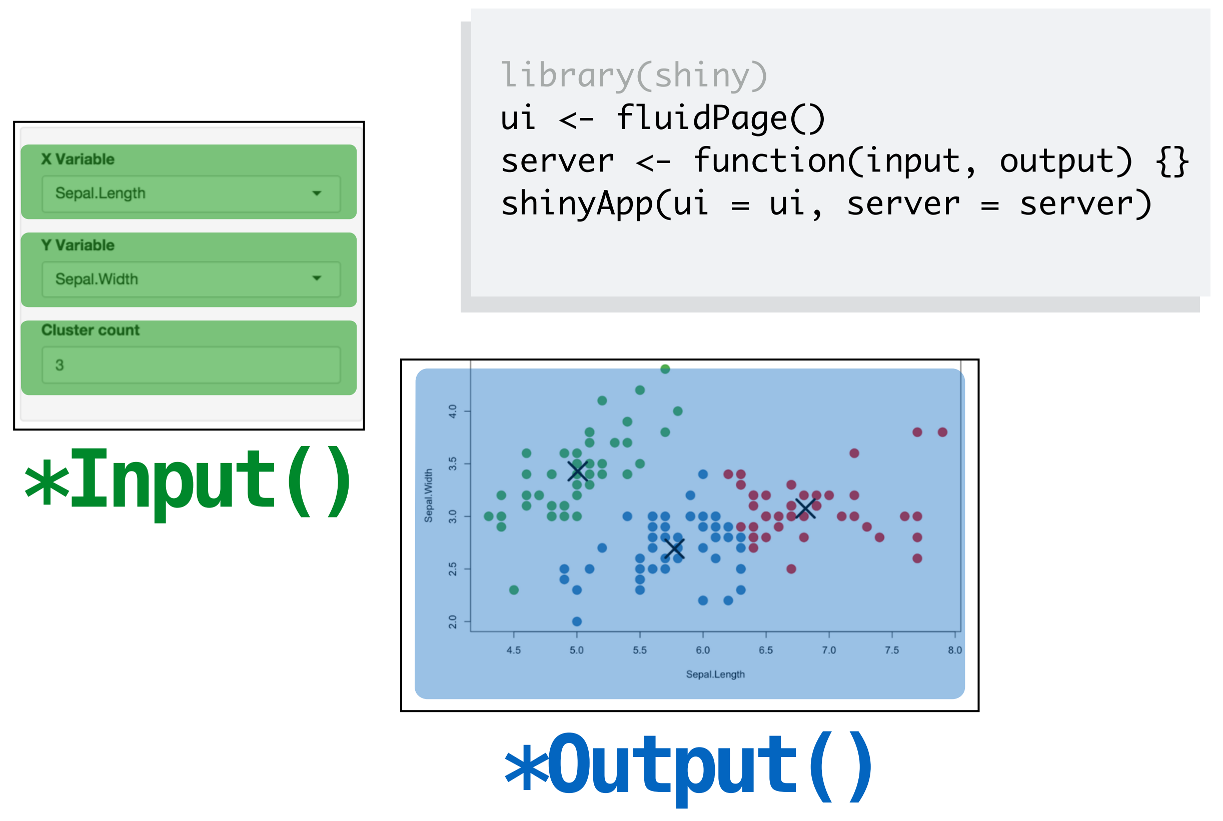
```
library(shinyapps)
```

2. Install the **`shinyapps`** package

Build your own server with **Shiny Server** or **Shiny Server Pro**

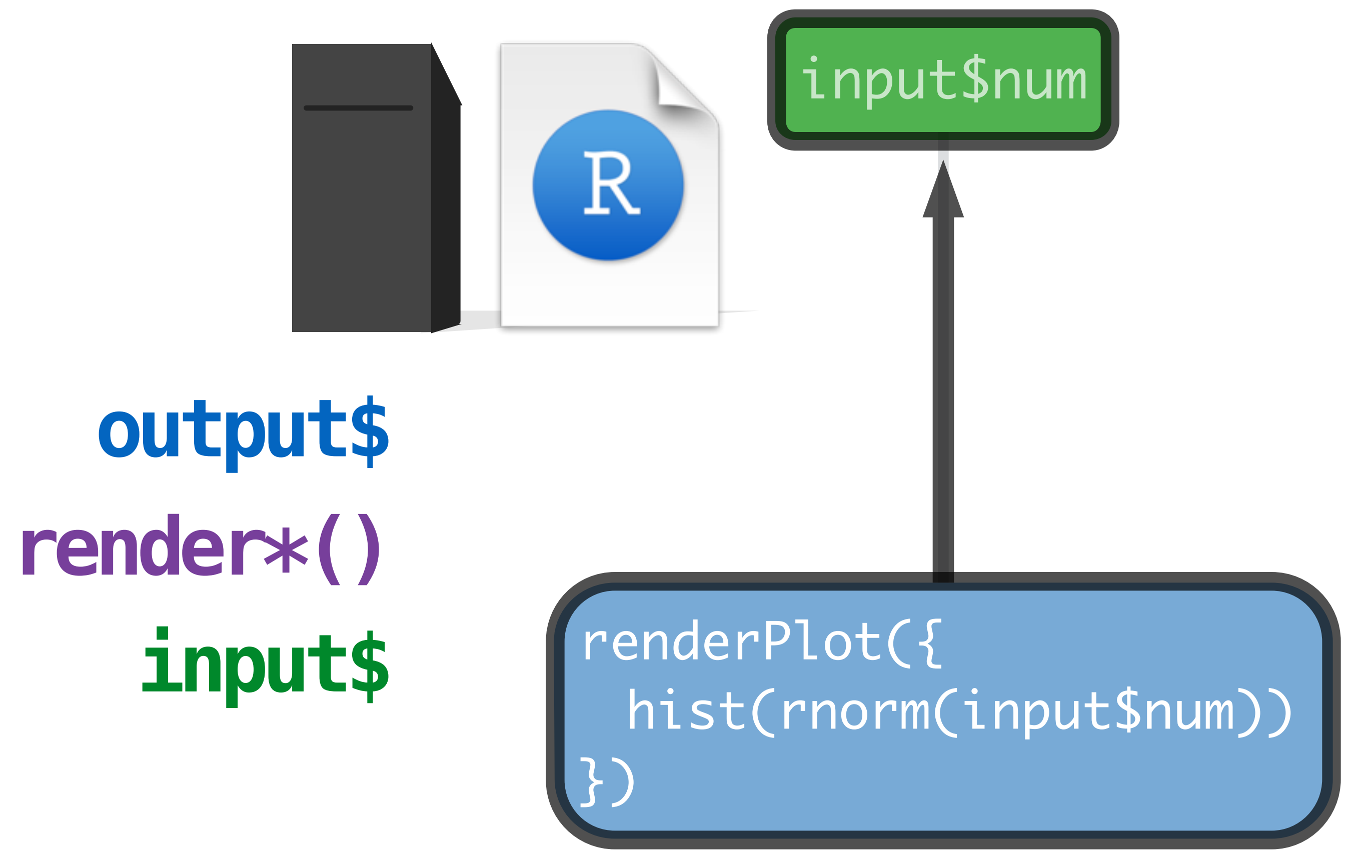
Learn more

You now how to



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

Build an app

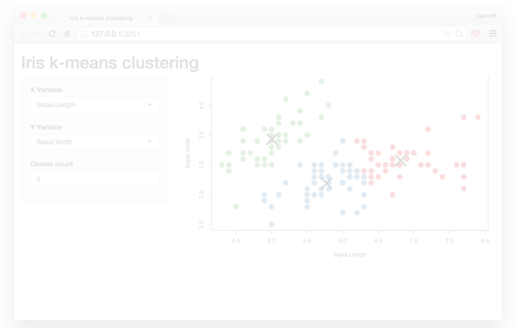


```
renderPlot({
  hist(rnorm(input$num))
})
```

Create interactions

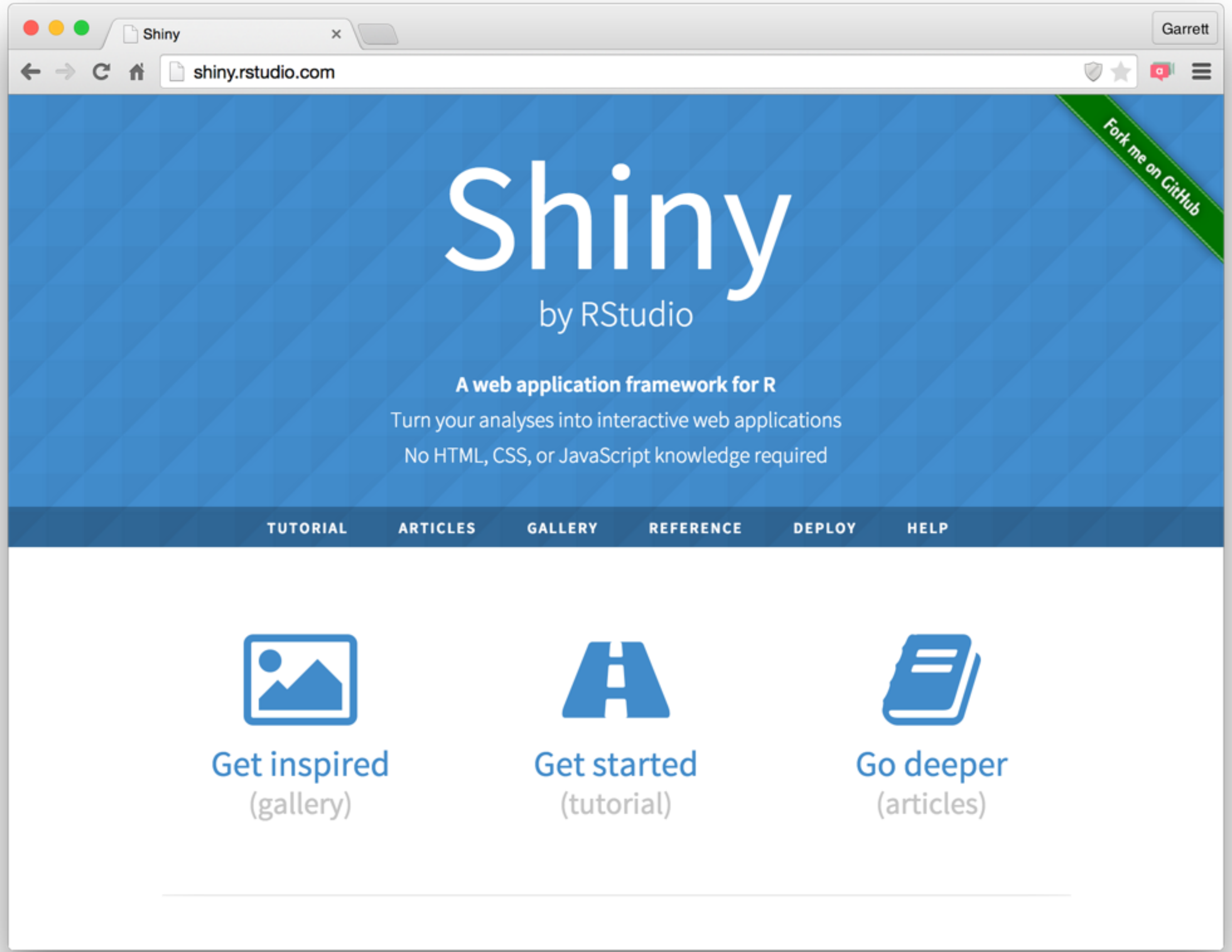
Share your apps

How to start with Shiny



1. How to build a Shiny app (Today)
2. How to customize reactions (May 27)
3. How to customize appearance (June 3)

The Shiny Development Center
shiny.rstudio.com


Shiny

by RStudio

A web application framework for R

Turn your analyses into interactive web applications

No HTML, CSS, or JavaScript knowledge required