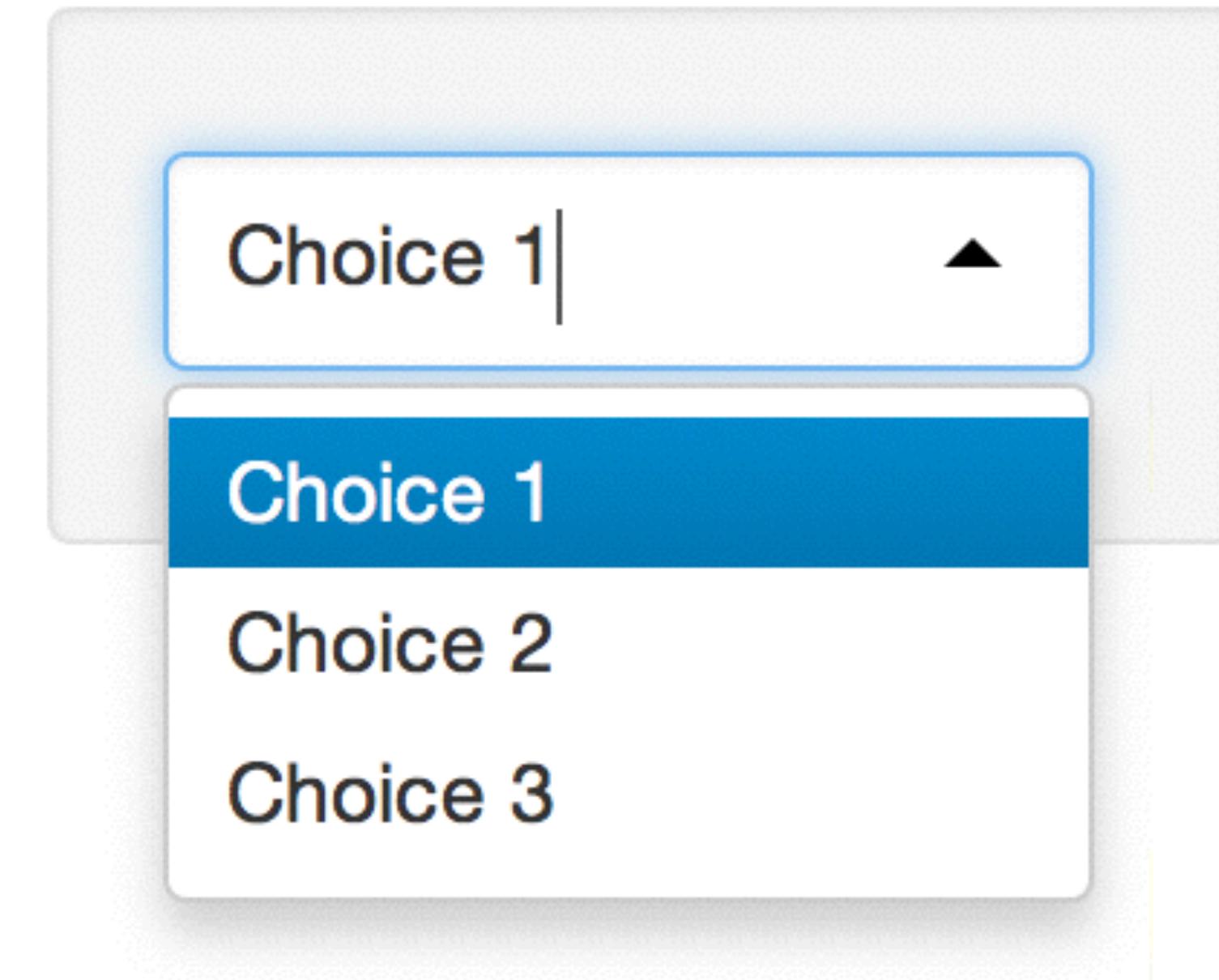


All Training materials are provided "as is" and without warranty and RStudio disclaims any and all express and implied warranties including without limitation the implied warranties of title, fitness for a particular purpose, merchantability and noninfringement.

The Training Materials are licensed under the Creative Commons Attribution-Noncommercial 3.0 United States License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc/3.0/us/> or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

# Reproducible expertise with Shiny

Empower your audience



## Joe Cheng

Software Engineer  
September 2015  
Email: [joe@rstudio.com](mailto:joe@rstudio.com)

# Reproducible data science

1

**Code**, results, and reasoning

**R Markdown**



2

**Computing Environment**  
(R version, packages)

**Packrat** (projects)



3

**Expertise** to extend, reapply,  
and reuse code

**Shiny**



**what is  
shiny?**

# Motivation

R is a powerful platform for data analysis

- State of the art statistical power
- Excellent for visualization
- Large and enthusiastic community

# Motivation

But...

- A personal experience, not a shared one
- Output is usually in static formats
- Modern visualization is interactive, browser-based

# What is Shiny?

- Interactive web applications around your R analyses
- Zero HTML/CSS/JavaScript knowledge is required...
- ...but fully customizable and extensible with HTML/CSS/JavaScript

# What is Shiny?

- Modern web UI with attractive defaults, but also infinitely customizable
- Designed to integrate with existing JavaScript libraries
- Uses a reactive programming model which allows dramatically simpler code than traditional UI or web programming

<http://shiny.rstudio.com/gallery/kmeans-example.html>

Shiny - Gallery Garrett

[shiny.rstudio.com/gallery/](http://shiny.rstudio.com/gallery/)

## Shiny by RStudio

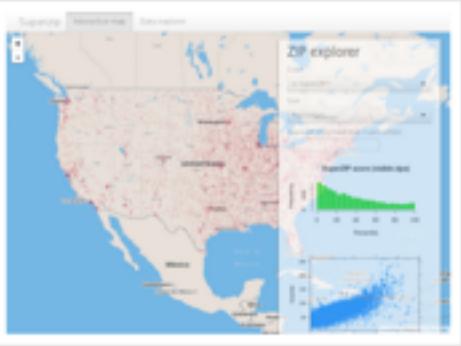
OVERVIEW TUTORIAL ARTICLES GALLERY REFERENCE DEPLOY HELP

# Gallery

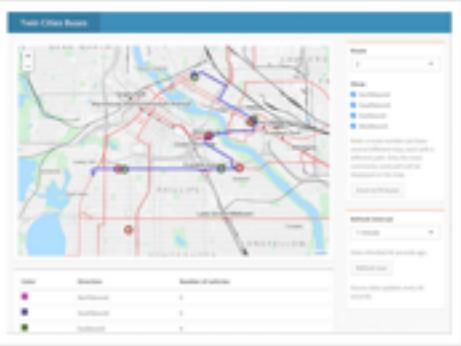
This gallery contains useful examples to learn from. Visit the [Shiny User Showcase](#) to see an inspiring set of sophisticated apps.

### Interactive visualizations

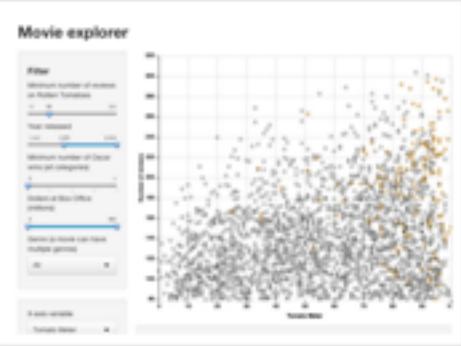
Shiny is designed for fully interactive visualization, using JavaScript libraries like [d3](#), [Leaflet](#), and [Google Charts](#).



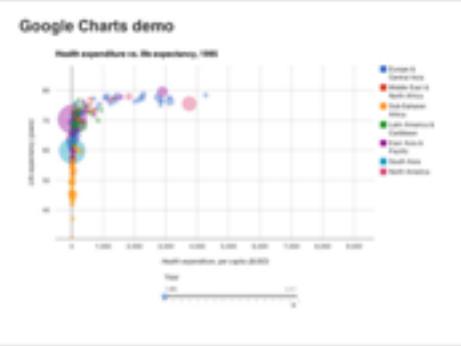
SuperZip example



Bus dashboard



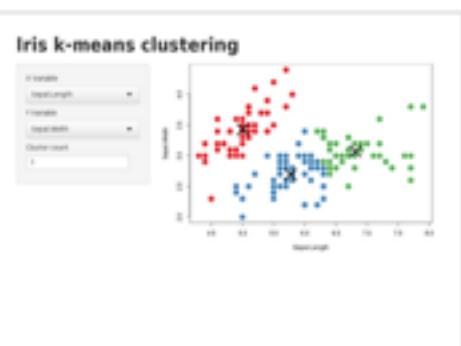
Movie explorer



Google Charts

### Start simple

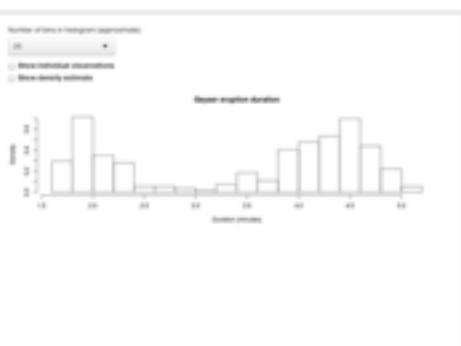
If you're new to Shiny, these simple but complete applications are designed for you to study.



Iris k-means clustering



Telephones by region



Faithful



Word cloud

[shiny.rstudio.com/gallery/](http://shiny.rstudio.com/gallery/)

Shiny User Showcase - RS Garrett

[www.rstudio.com/products/shiny/shiny-user-showcase/](http://www.rstudio.com/products/shiny/shiny-user-showcase/)

## R Studio

Products Resources Pricing About Us Blog

### Shiny User Showcase

Home / Overview / Shiny / Shiny User Showcase

Shiny is designed for fully interactive visualization, using JavaScript libraries like d3, Leaflet, and Google Charts. Our users create fantastic examples, and some have shared them with the community. Here are some examples that we particularly like.

### Shiny Apps for the Enterprise



Shiny Dashboard Demo

A dashboard built with Shiny.



Location tracker

Track locations over time with streaming data.



Download monitor

Streaming download rates visualized as a bubble chart.



Supply and Demand

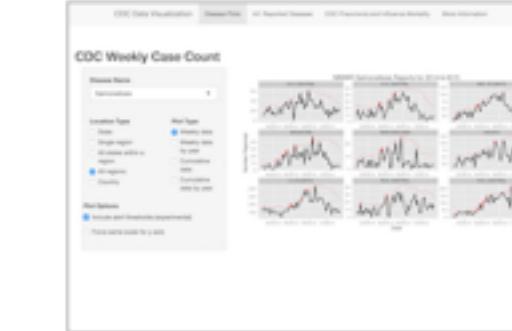
Forecast demand to plan resource allocation.

### Industry Specific Shiny Apps



Emergency Department Simulation

[Monitor](http://www.rstudio.com/products/shiny/shiny-user-showcase/#monitor)



CDC Weekly Case Count



Ebola Model



Pharmacometrics

<http://www.rstudio.com/products/shiny/shiny-user-showcase/>

© 2015 RStudio, Inc. All rights reserved.

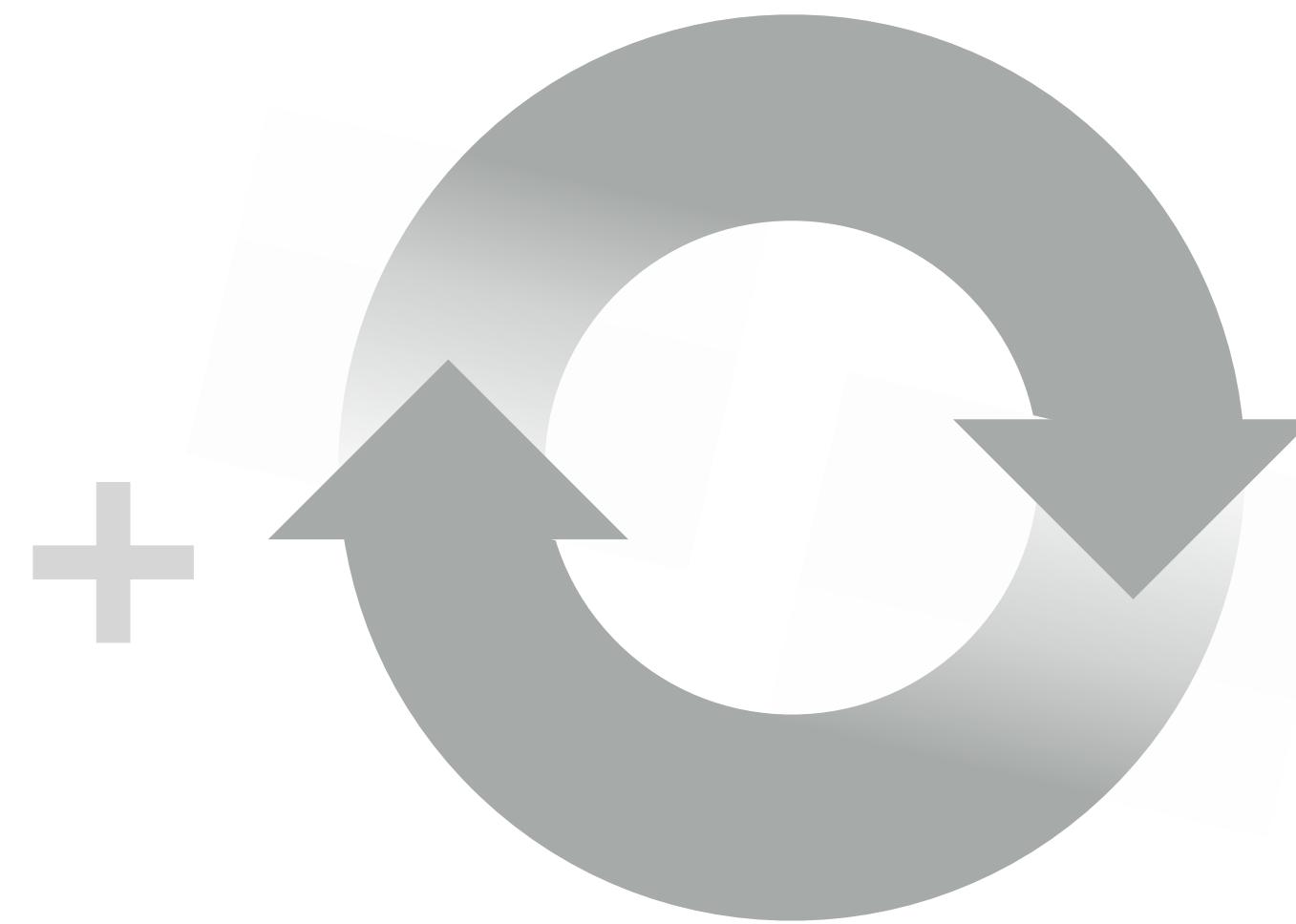
# Shiny



Computation  
using R



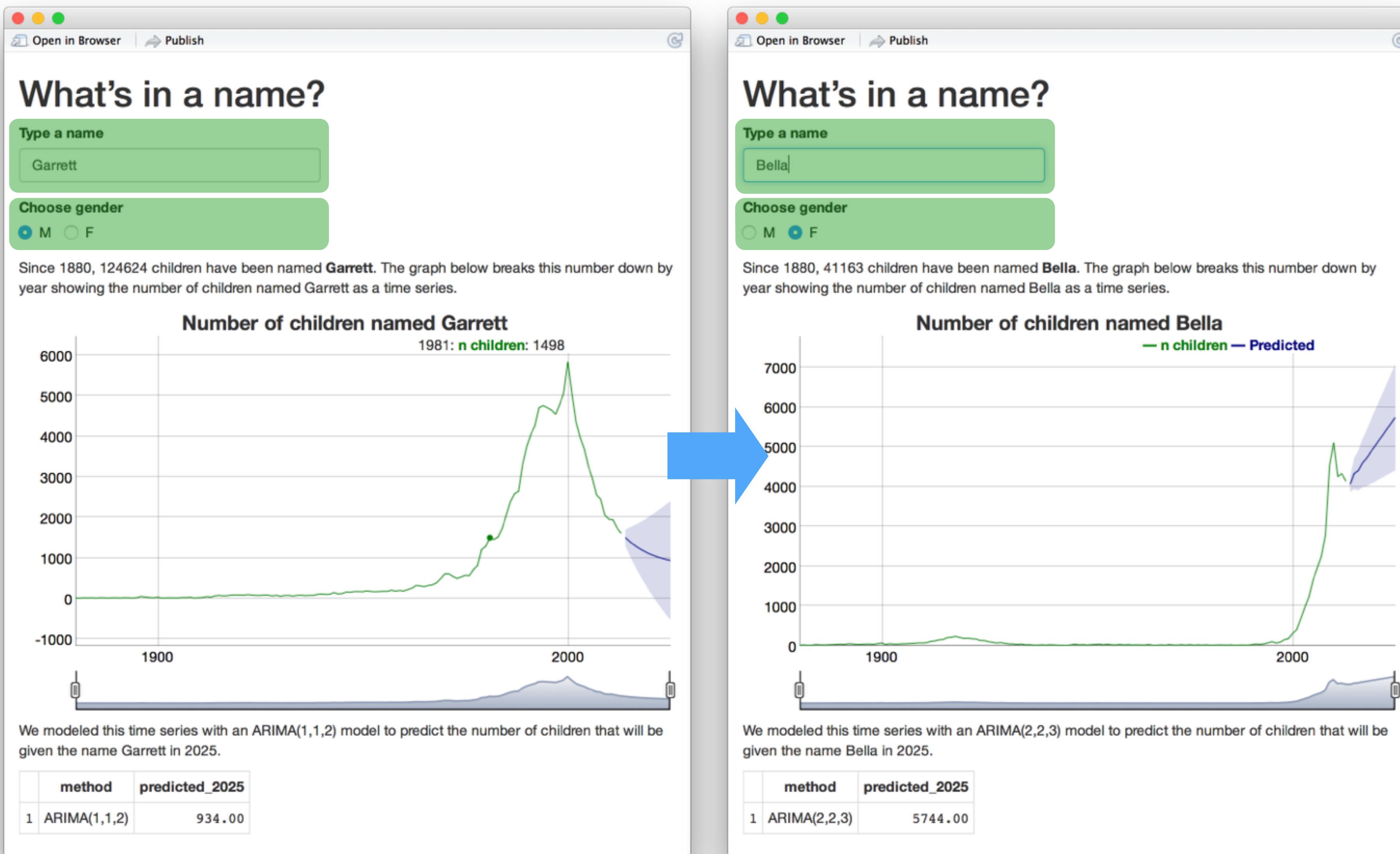
Web-based  
user interface



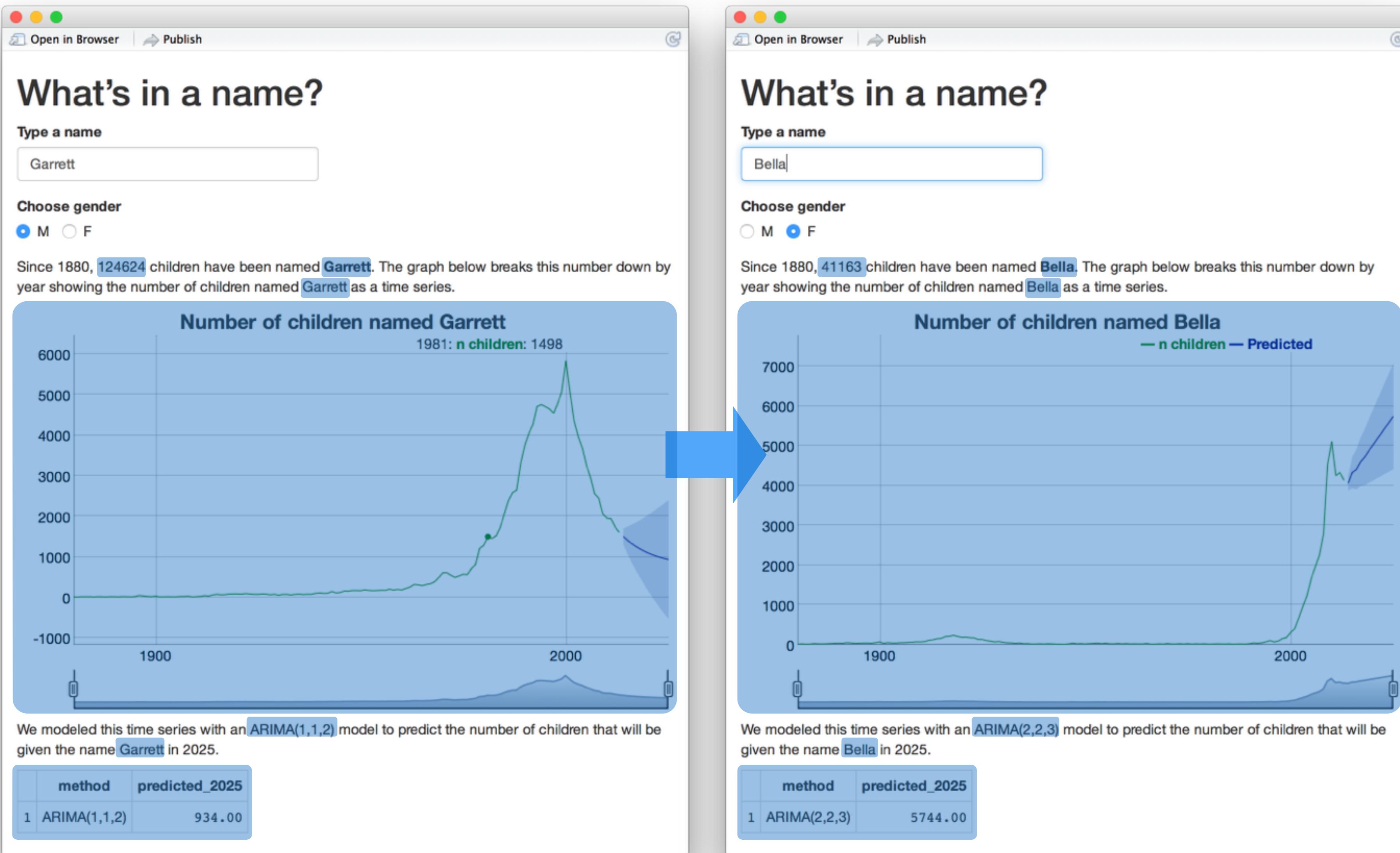
Reactive  
programming

<https://gallery.shinyapps.io/names-app/name-app.Rmd>

# Goal: make display output react to user *input*



# Goal: make display *output* react to user input



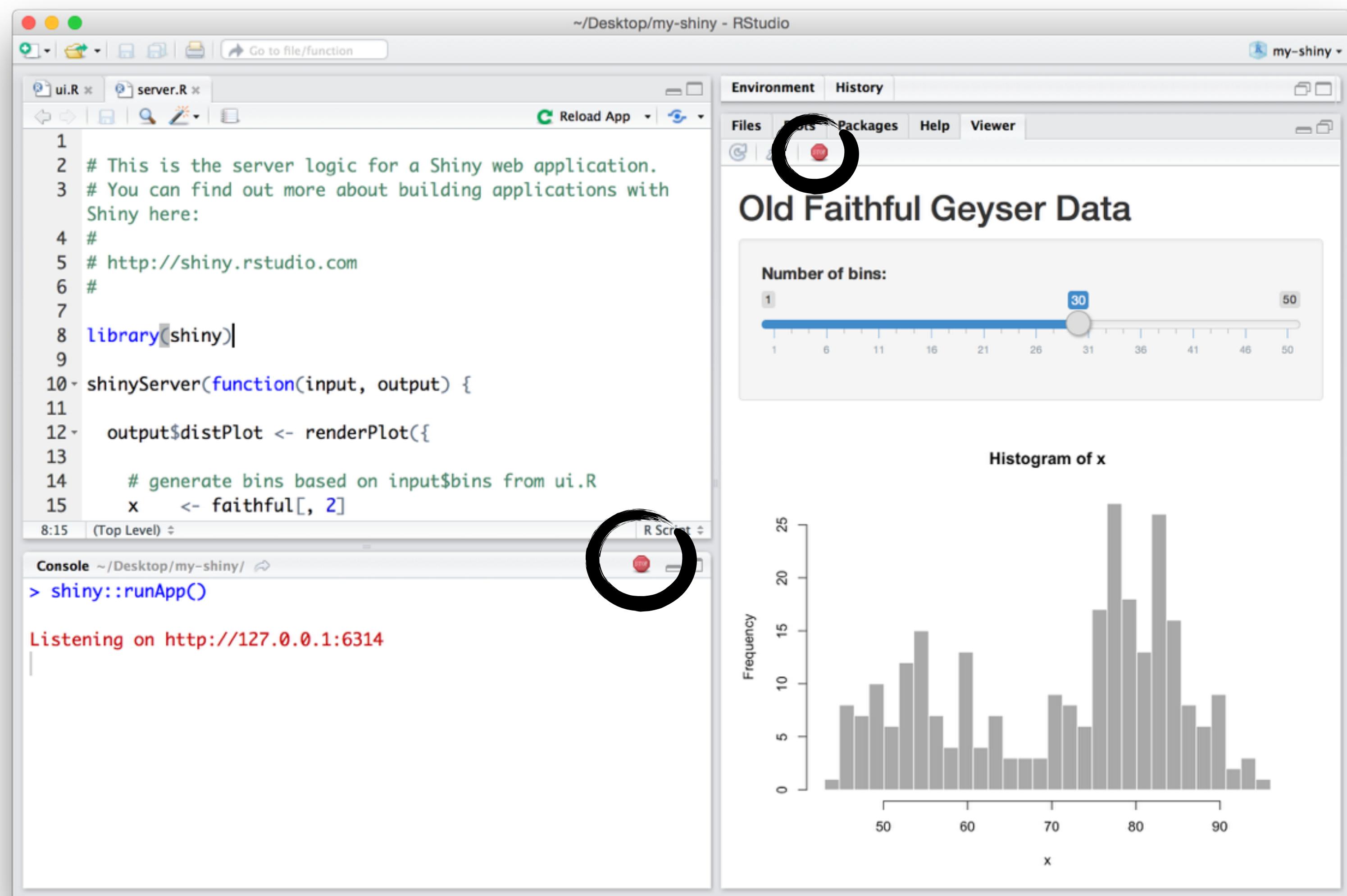
App  
template

# App template

## The shortest viable shiny app

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

# Close an app



# shinyApp

**shinyApp()** assembles the server and ui components into a complete app.

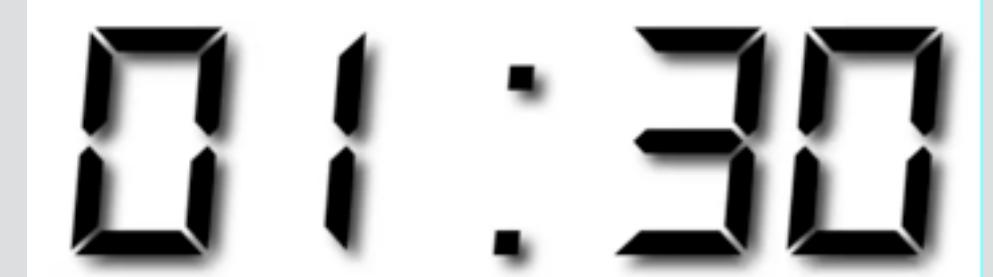
```
shinyApp(ui = my_ui, server = my_serv)
```

A ui object  
that builds a web page  
for your app

A server function  
that builds the R  
components of the app

# Warm up

`exercise(20)` opens a copy of the app template.  
Launch the app by copying and pasting the code into  
the command line.



ui

1

ui contains a function that writes an HTML page for your app.

ui starts with the following code (or its equivalent).

```
fluidPage()
```

# 2

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

```
fluidPage(  
  # input controls,  
  # output controls,  
  # layout objects,  
  # custom HTML/CSS/JavaScript  
)
```

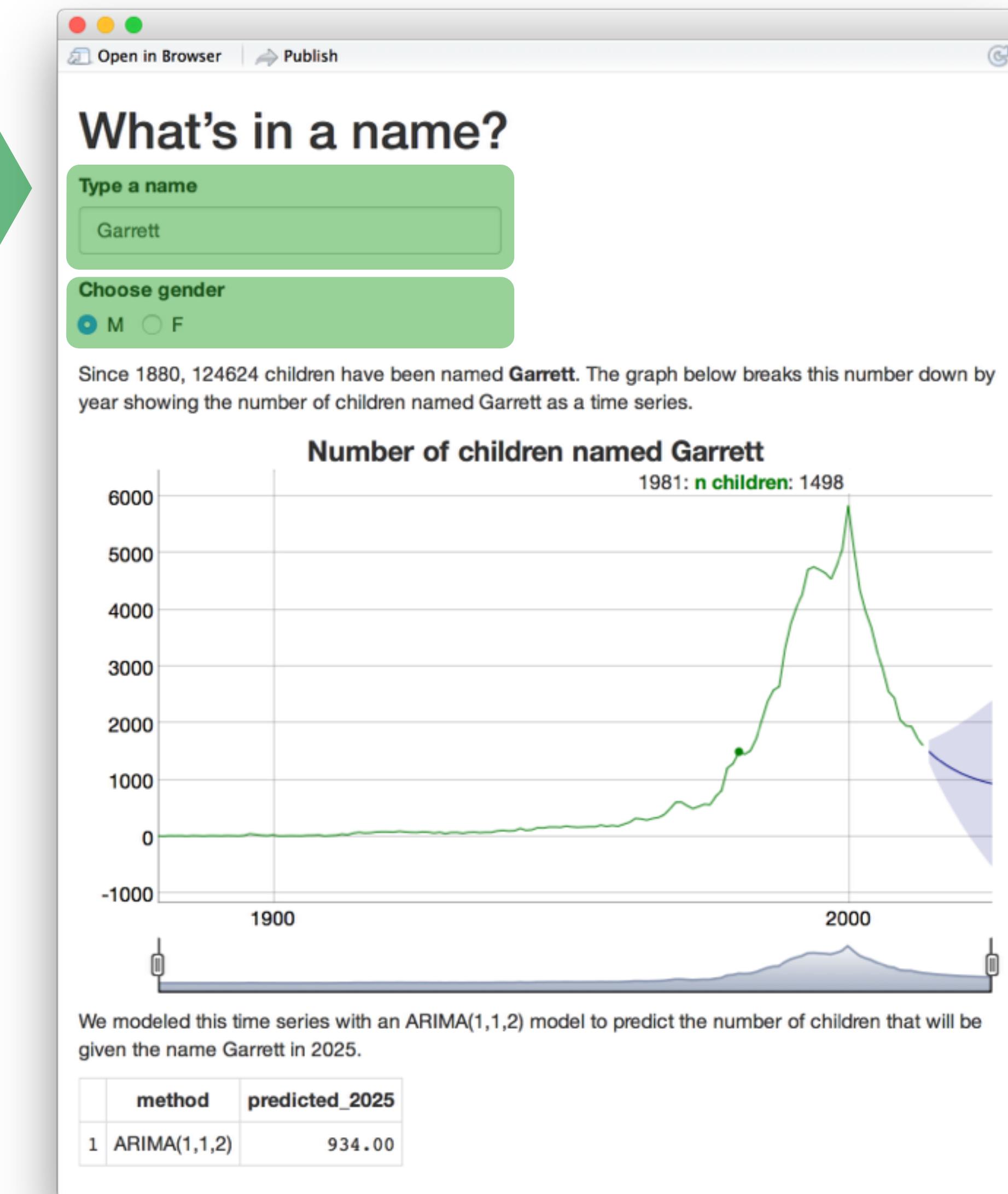
## 2

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

```
fluidPage(  
  # input controls,  
  # output controls,  
  # layout objects,  
  # custom HTML/CSS/JavaScript  
)
```

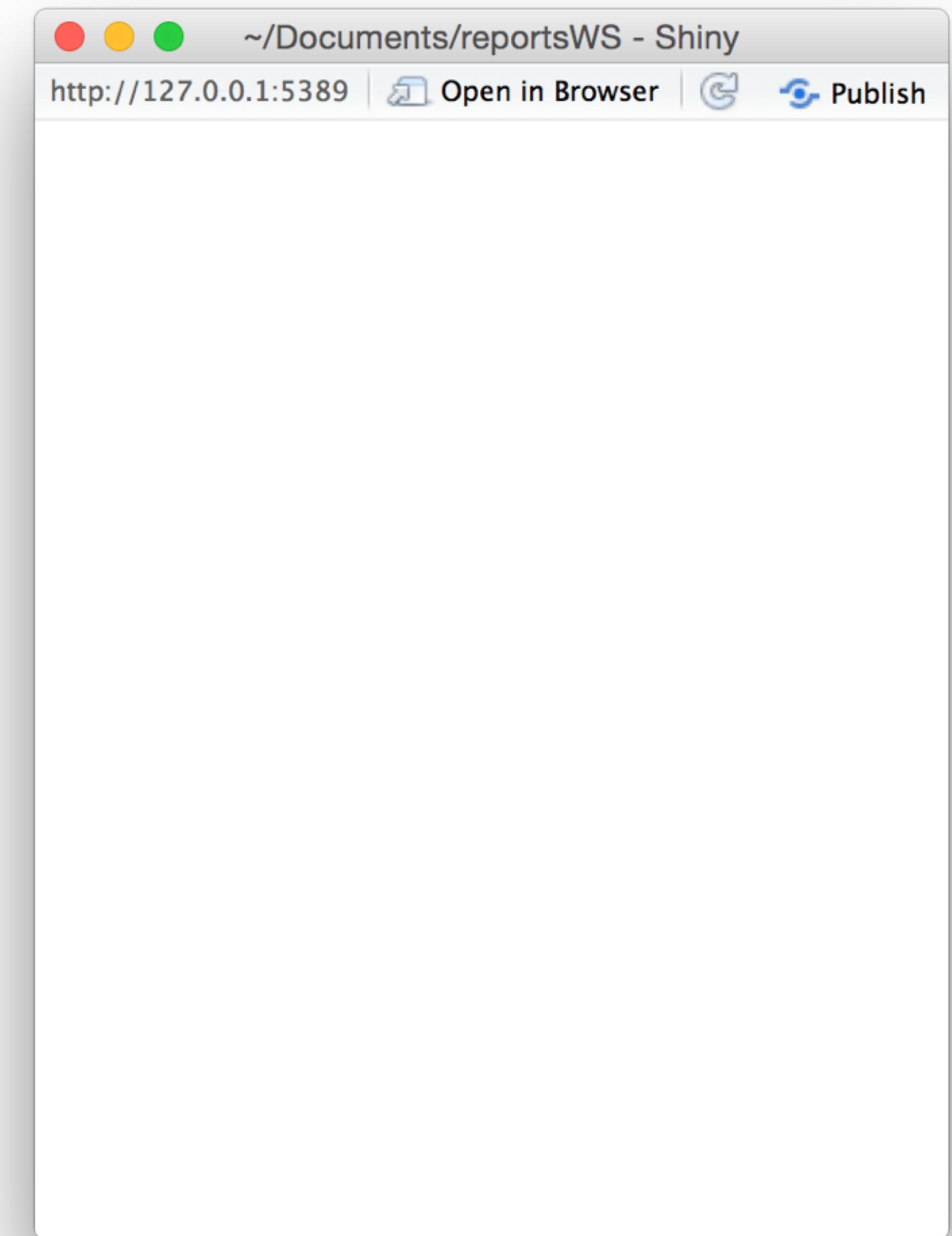
# Inputs

**Inputs** - let users set a value by typing, clicking, etc.



# 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 = "sl",
    "How loud should the guitar be?",
    min = 0, max = 11, value = 5)
)

server <- function(input, output) {}

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



## Buttons

Action

Submit

actionButton  
submitButton

## Date range

2014-01-24 to 2014-01-24

dateRangeInput

## Radio buttons

- Choice 1
- Choice 2
- Choice 3

radioButtons

## Single checkbox

Choice A

checkboxInput

## File input

No file chosen

fileInput

## Select box

selectInput

## Checkbox group

Choice 1

Choice 2

Choice 3

checkboxGroupInput

## Date input

2014-01-01

dateInput

## Numeric input

1

numericInput

## Password Input

.....

passwordInput

## Sliders



sliderInput

## Text input

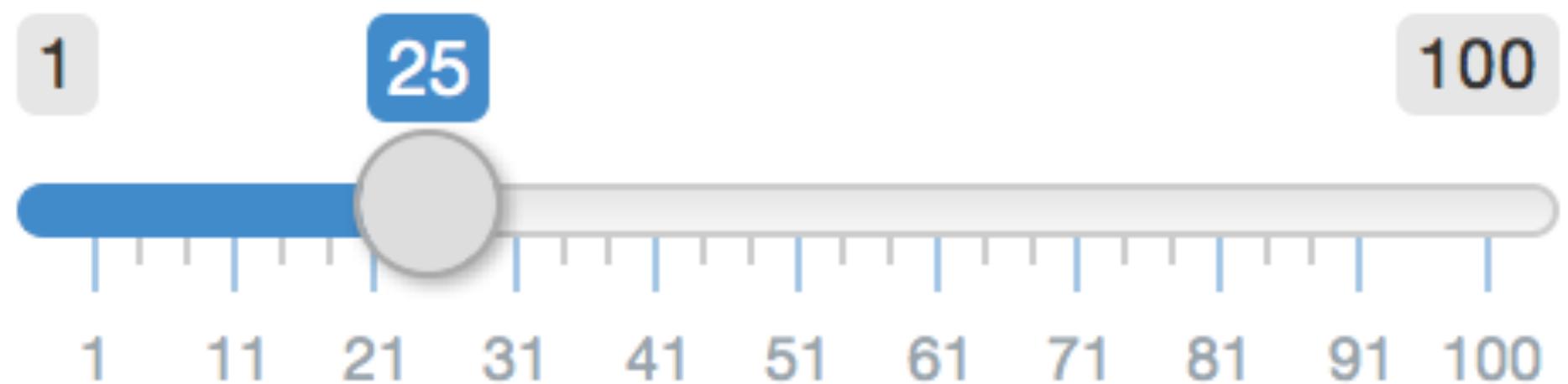
Enter text...

textInput

function	input object
actionButton	Action button
actionLink	Action link
checkboxGroupInput	Group of checkboxes
checkboxInput	Single checkbox
dateInput	Calendar to aid date selection
dateRangeInput	Pair of calendars for selecting date range
fileInput	File upload control wizard
numericInput	Field to enter numbers
passwordInput	Field to enter hidden text
radioButtons	Set of radio buttons
selectInput	Box with choices to select from
sliderInput	Slider bar
textInput	Field to enter text

# Syntax

**Choose a number**



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

input  
function

input name  
(for internal use)

Notice:  
Id not ID

label to  
display

input specific  
arguments

# Your Turn

Add a slider input named "num" to your empty app (exercise(20)).

The Slider should begin at 25 and go from 1 to 100.

Use the help page at `?sliderInput` to determine which arguments to use:

1. `inputId`
2. `label`
3. `?`
4. `?`
5. `?`

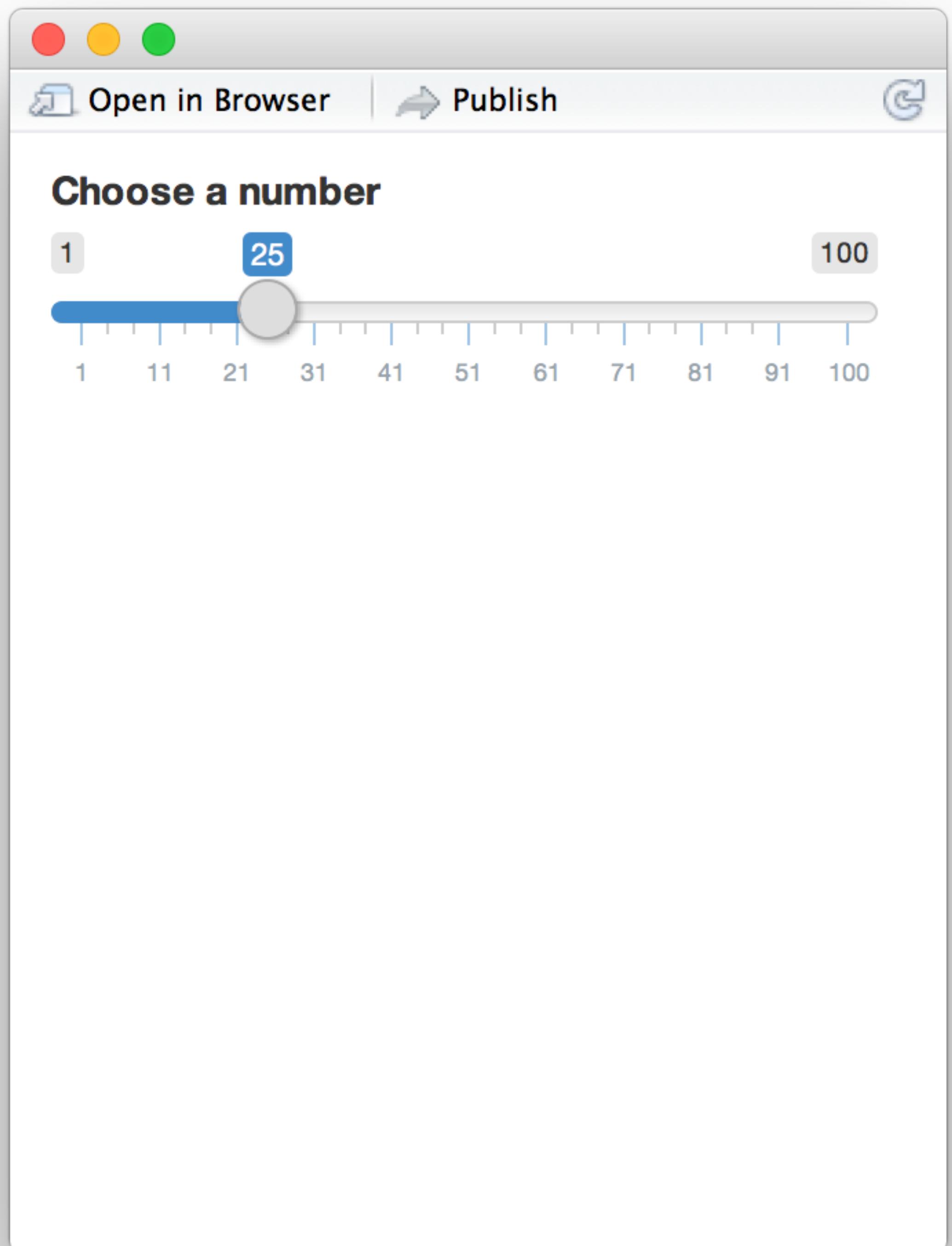


```
library(shiny)

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

server <- function(input, output) {}

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



# Outputs

## 2

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

```
fluidPage(  
  # input controls,  
  # output controls,  
  # layout objects,  
  # custom HTML/CSS/JavaScript  
)
```

**Outputs** - respond, if needed, when reactive values change



# \*Output()

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

```
plotOutput(outputId = "barPlot")
```

the type of  
output to  
display

an \*Output  
function

name of object

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

# To display output, add it to the UI with an \*Output function

```
library(shiny)
ui <- fluidPage(
  sliderInput("sl", "How loud should the
    guitar be?", min = 0, max = 11, val = 5),
  plotOutput("barPlot")
)

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

Comma between  
arguments

# To display output, add it to the UI with an \*Output function

```
library(shiny)
ui <- fluidPage(
  sliderInput("sl", "How loud should the
    guitar be?", min = 0, max = 11, val = 5),
  plotOutput("barPlot")
)
server <- function(input, output) {}
shinyApp(server = server, ui = ui)
```

Does this create a plot?

# Your turn

Return to the last exercise.

Place a plot named "hist" in your ui

Run the app. Does a plot appear?



```
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)
```

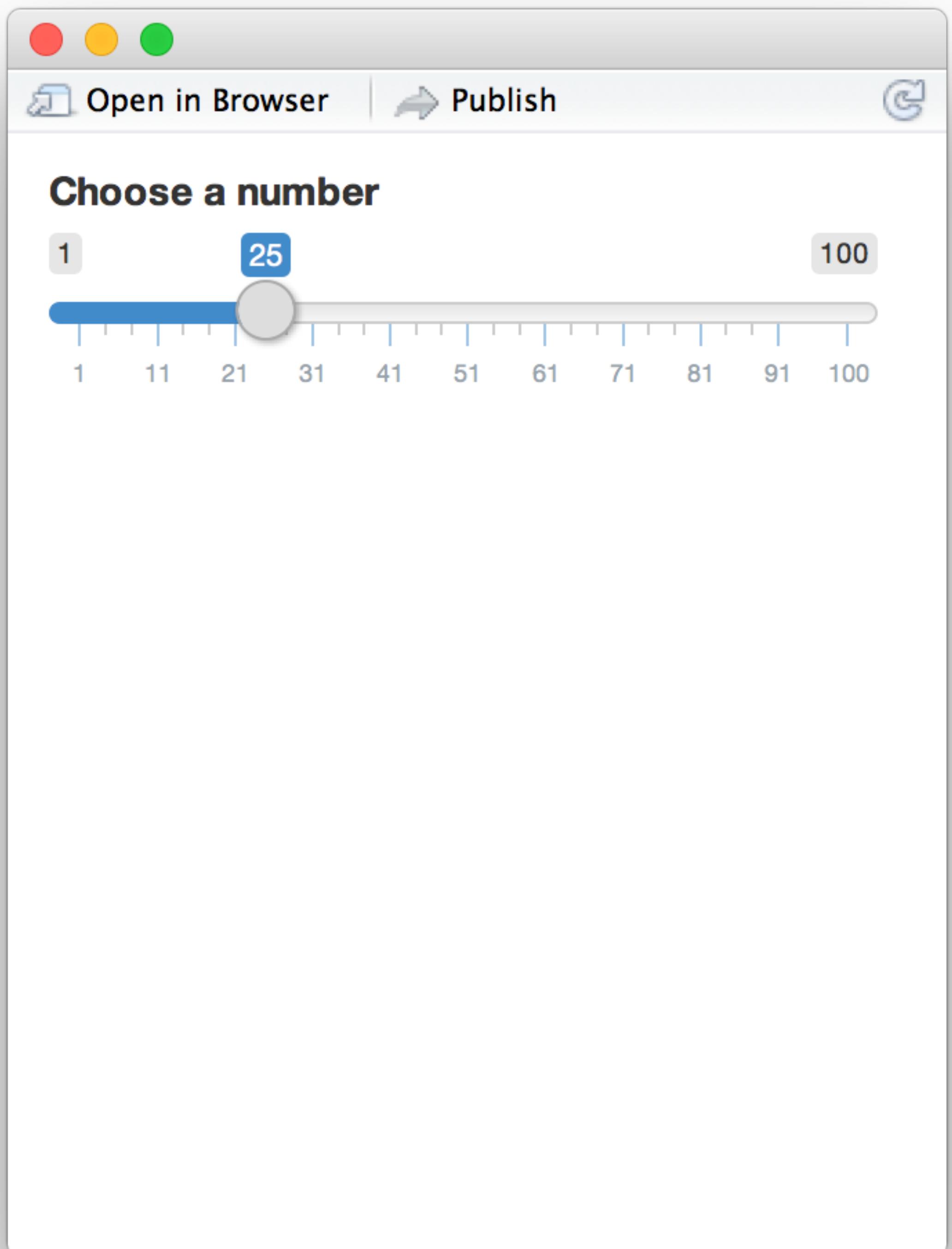
Comma between  
arguments

```
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)
```

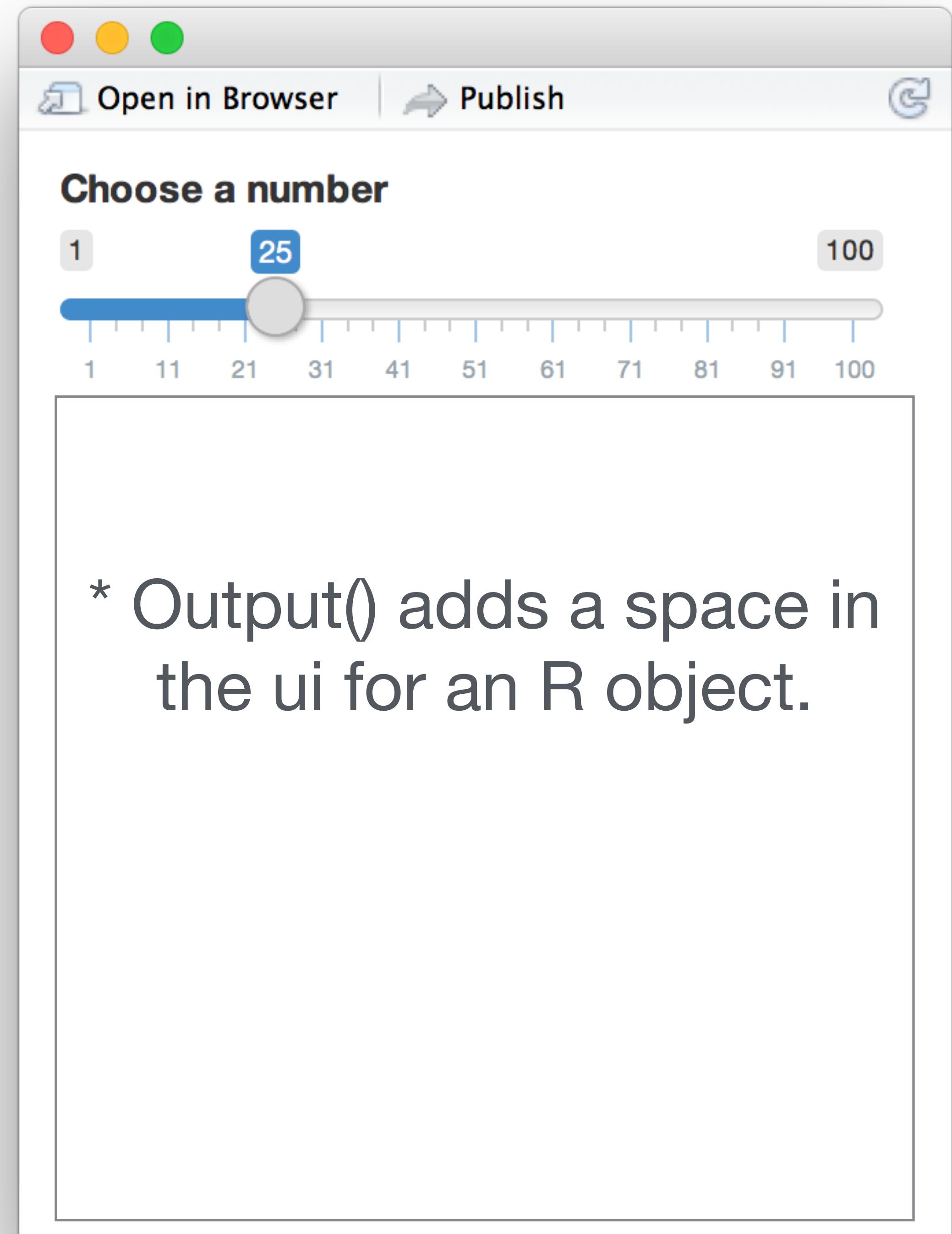


```
library(shiny)

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)
```

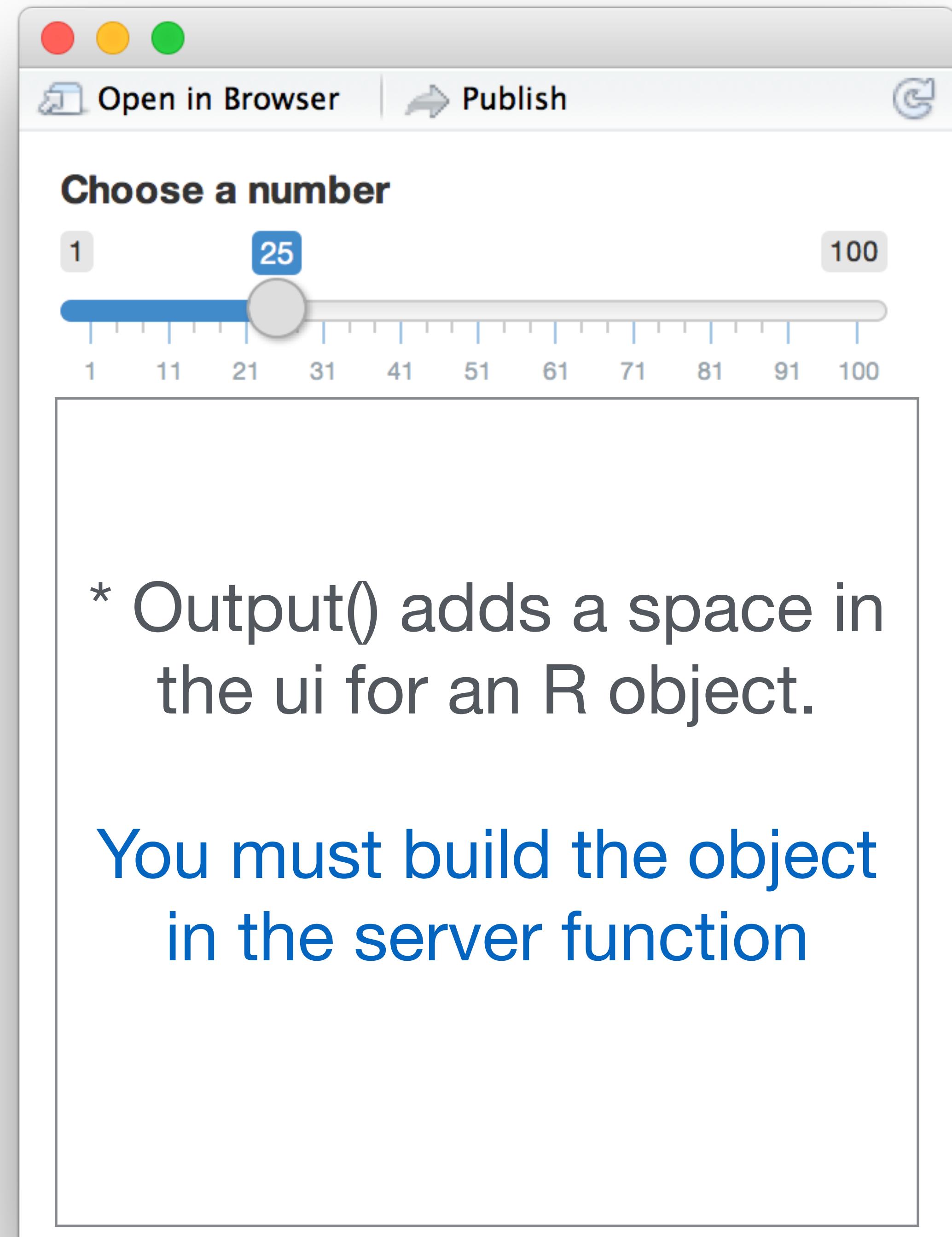


```
library(shiny)

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

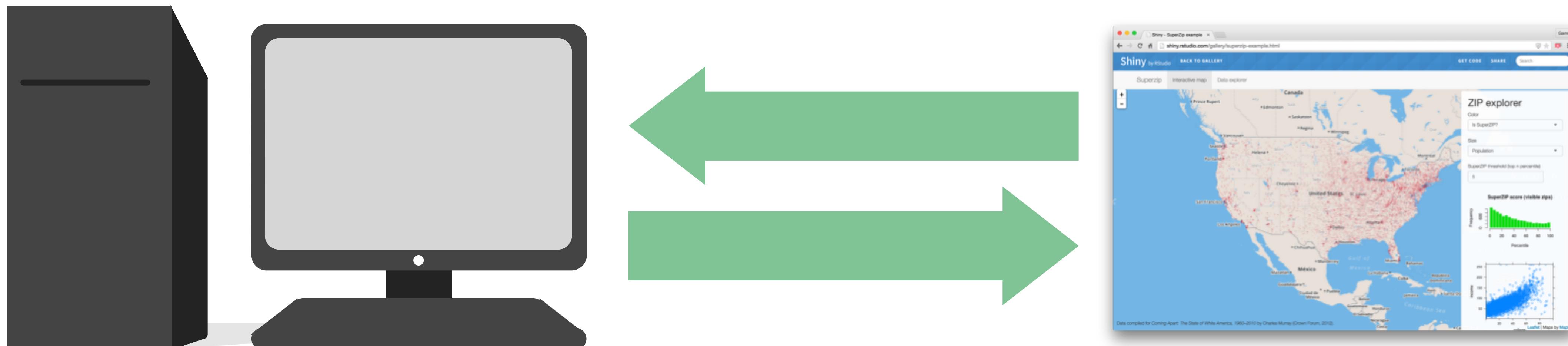
server <- function(input, output) {}

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



server

Every Shiny app is maintained by a computer running R



1

The **server function** is a function that **you** must provide. Shiny calls it anytime a web browser connects to your app (i.e. starts a *session*). You set up a session's outputs, calculations, and actions in this function.

```
function(input, output, session) {  
}  
}
```

# 1

The server function has two required parameters (`input`, `output`) and one optional parameter (`session`). Shiny will provide these objects to you.

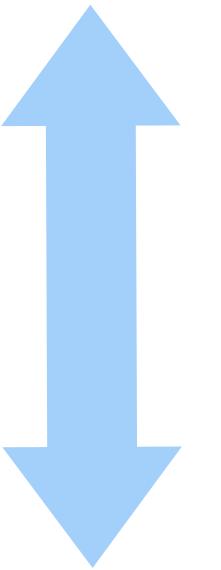
```
function(input, output, session) {  
}  
}
```

# 2

Use the function to build a list of outputs to display

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

output\$hist



plotOutput("hist")

# 3

## Build each object with a render\* function

```
function(input, output) {  
  output$hist <- renderPlot({  
    # code that makes a plot  
  })  
}
```

```
library(shiny)
ui <- fluidPage(
  sliderInput("sl",
    "How loud should the guitar be?",
    min = 0, max = 11, val = 5),
  plotOutput("barPlot")
)

server <- function(input, output) {
  output$barPlot <-

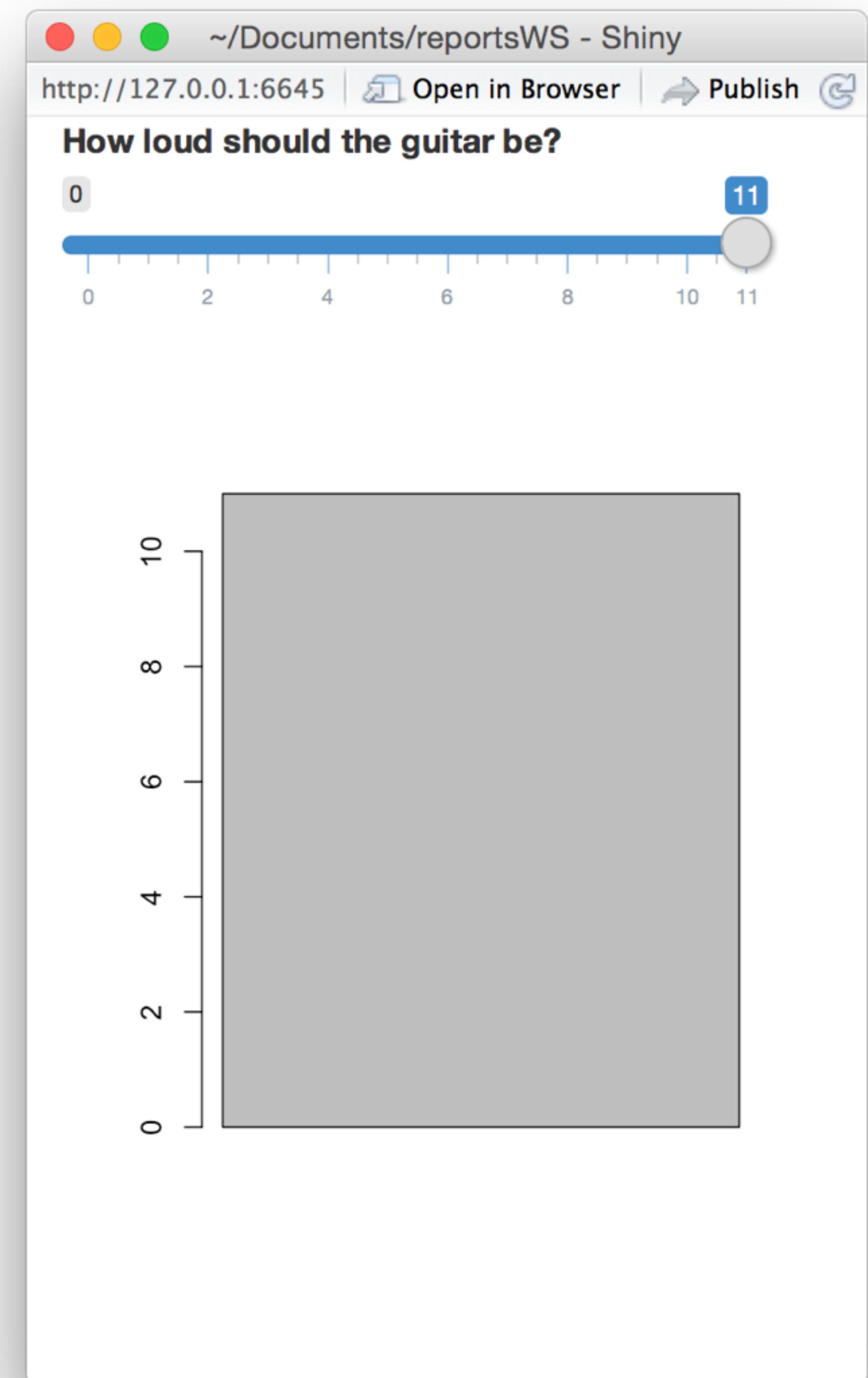
}

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

```
library(shiny)
ui <- fluidPage(
  sliderInput("sl",
    "How loud should the guitar be?",
    min = 0, max = 11, val = 5),
  plotOutput("barPlot")
)

server <- function(input, output) {
  output$barPlot <- renderPlot({
    barplot(input$sl, ylim = c(0, 11))
  })
}

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



# Your turn

Return to the last exercise.

Render a histogram of `input$num` random normal values and save it to `output$hist`, e.g.

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



```
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 <-
}

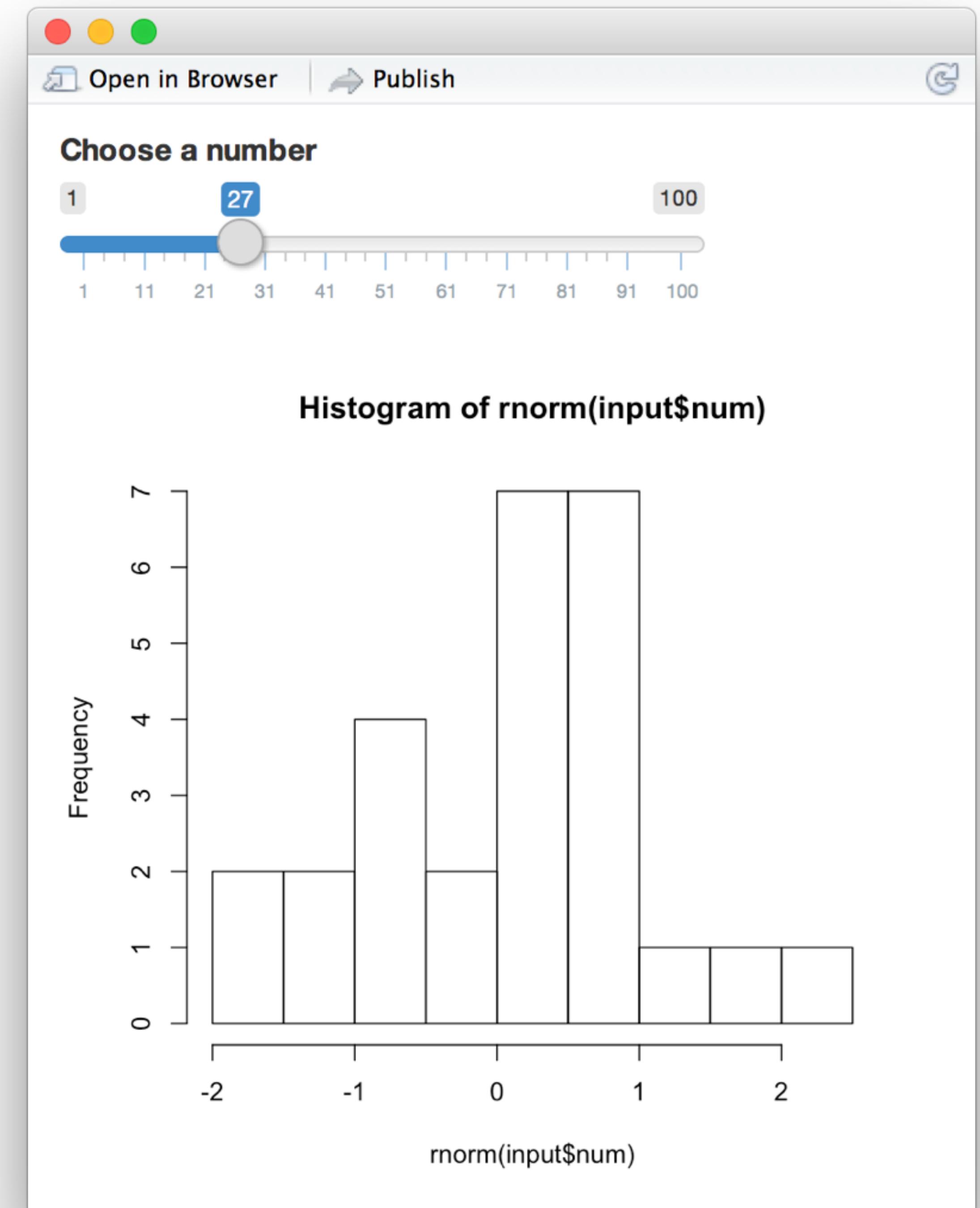
shinyApp(ui = ui, server = server)
```

```
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)
```



# Reactive programming

# Why reactivity?

The goal is to respond to changes in user input,  
with the **minimal amount of computation**  
required to update the outputs: no more, no less.

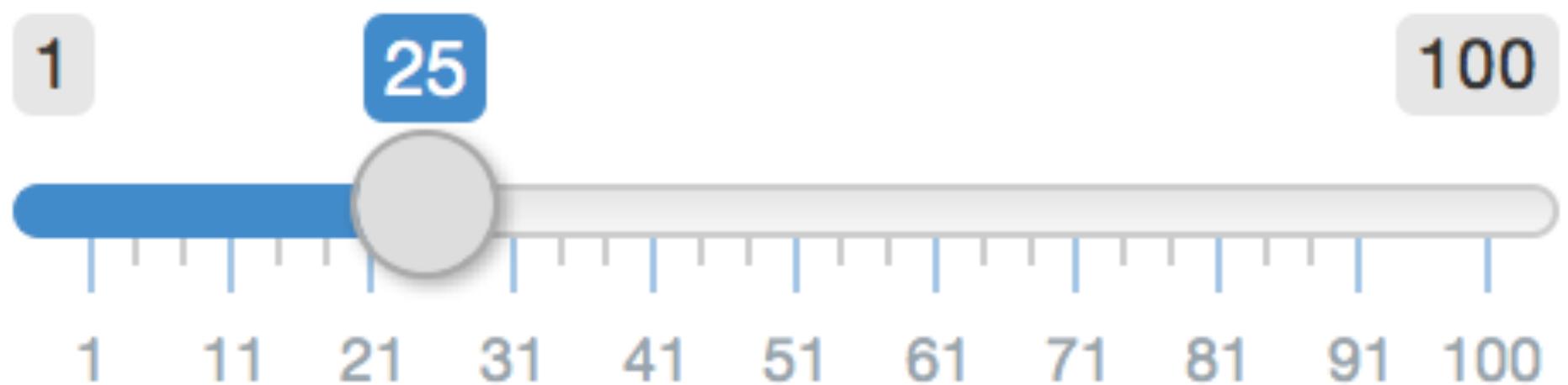
**Too much recomputation is wasteful/slow.**

**Too little recomputation gives us incorrect results.**

Reactivity helps—a lot!

# Syntax

**Choose a number**

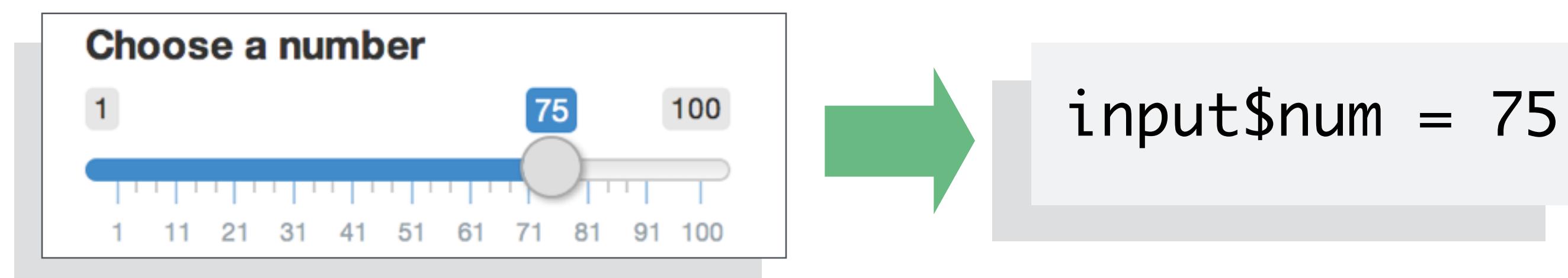
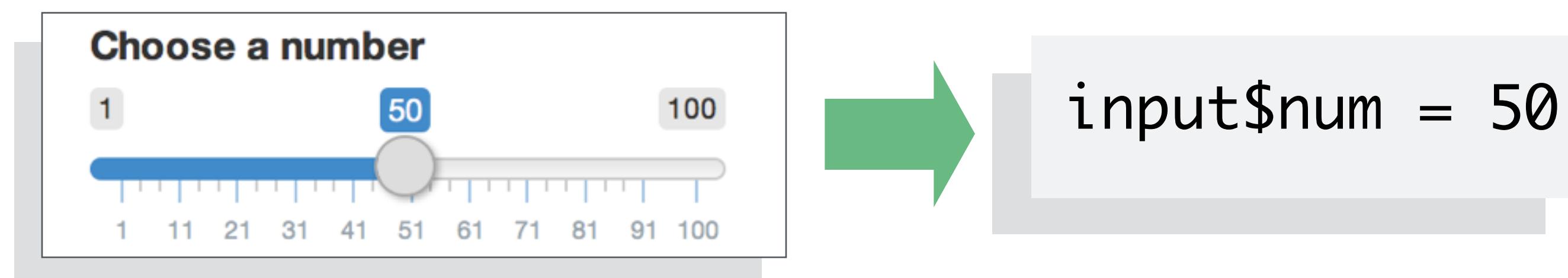
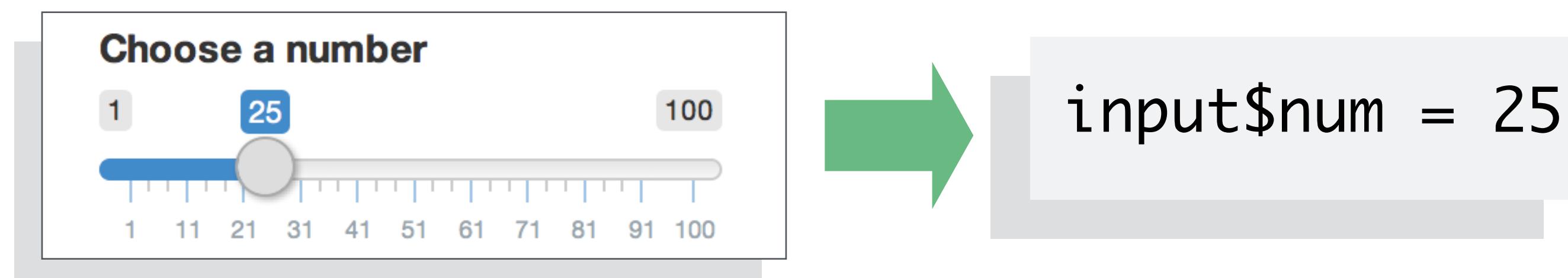


```
sliderInput(inputId = "num", label = "Choose a number", ...)
```

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

# Input values

The input value changes whenever a user changes the input.



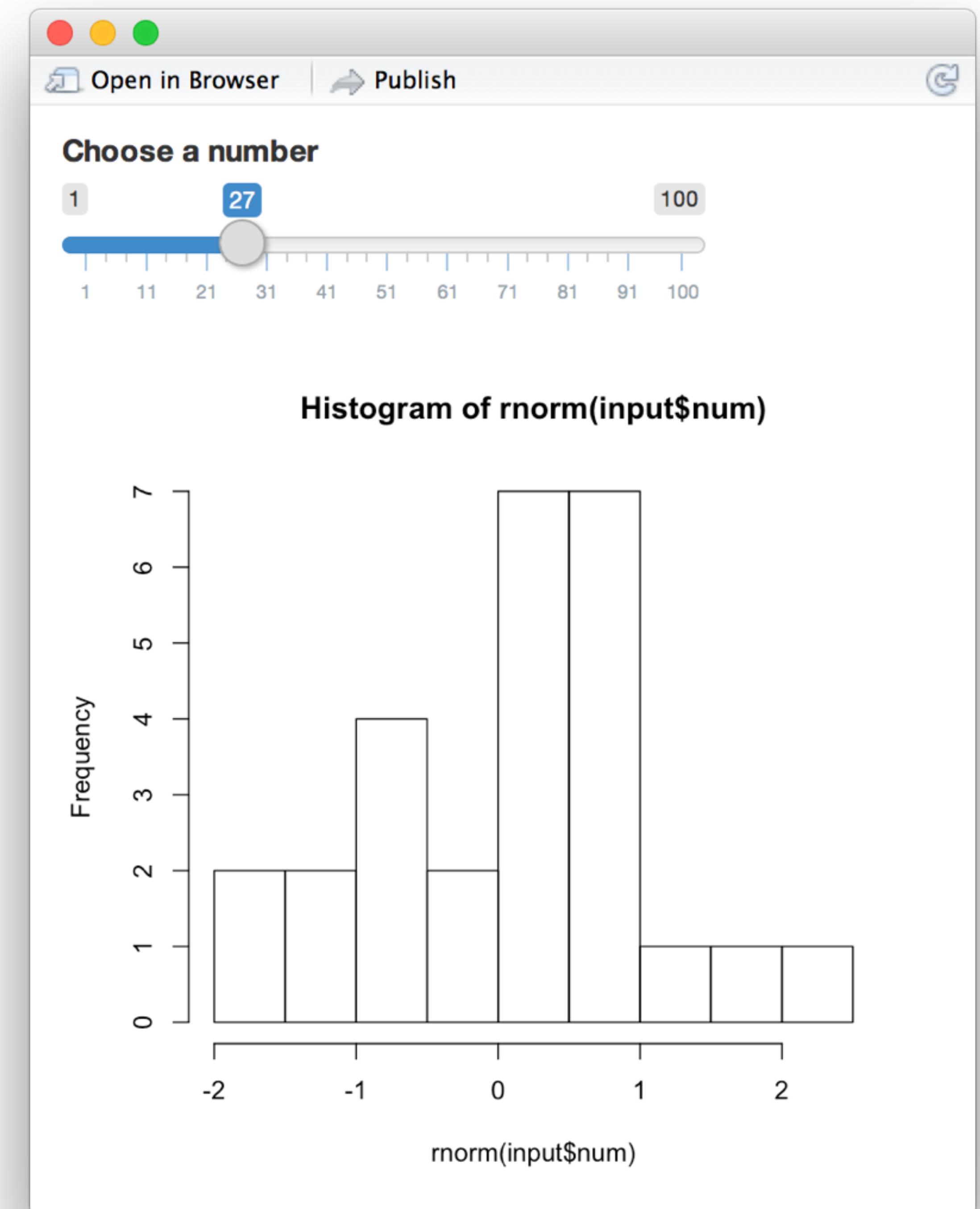
# Reactivity 101

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

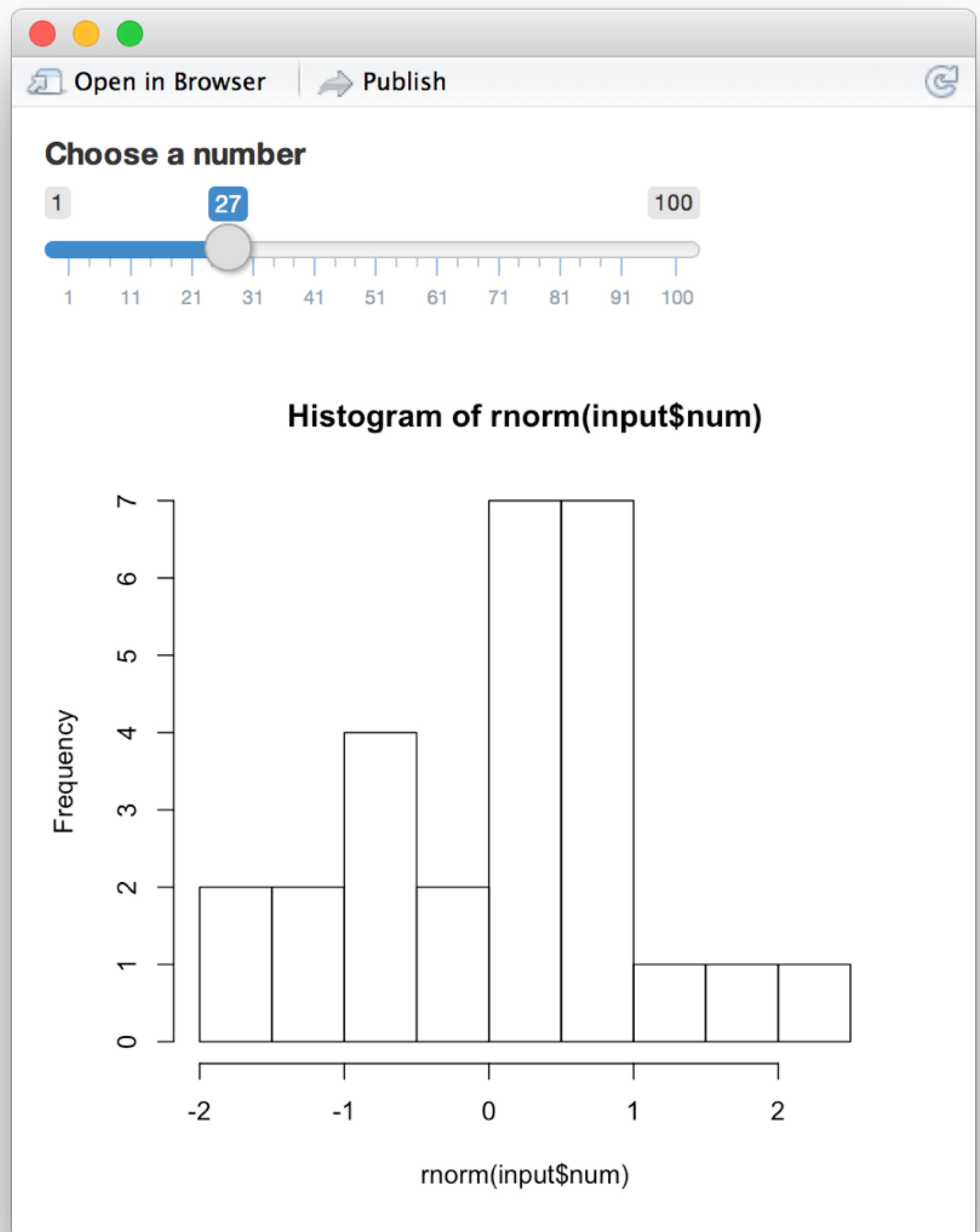
```
function(input, output) {  
  output$hist <- renderPlot({  
    hist(rnorm(input$num))  
  })  
}
```

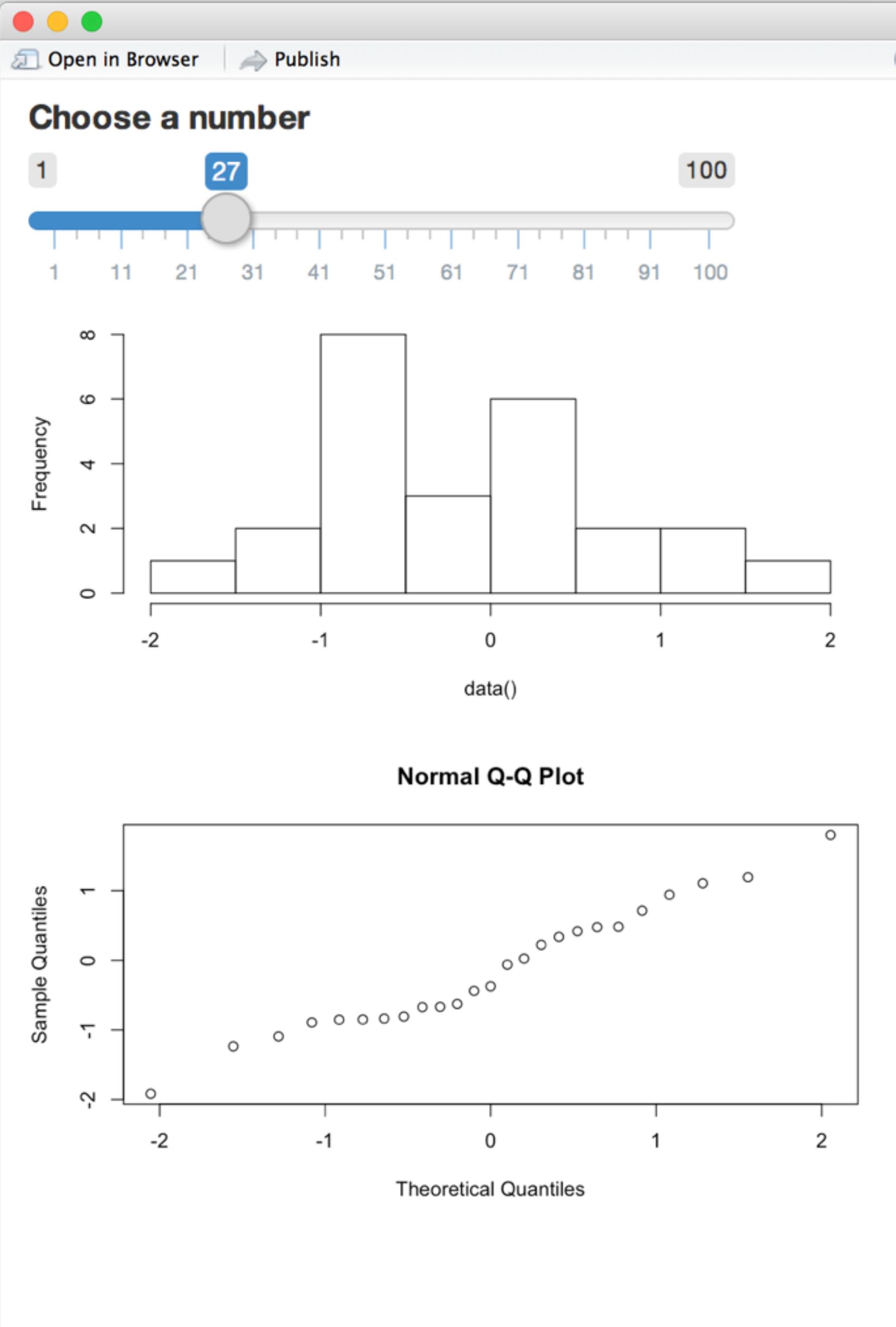
input\$num

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



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





# Your turn

Add a second plot to your app.  
It should contain a qqplot of input  
\$num random normal values, e.g.  
`qqnorm(rnorm(input$num))`

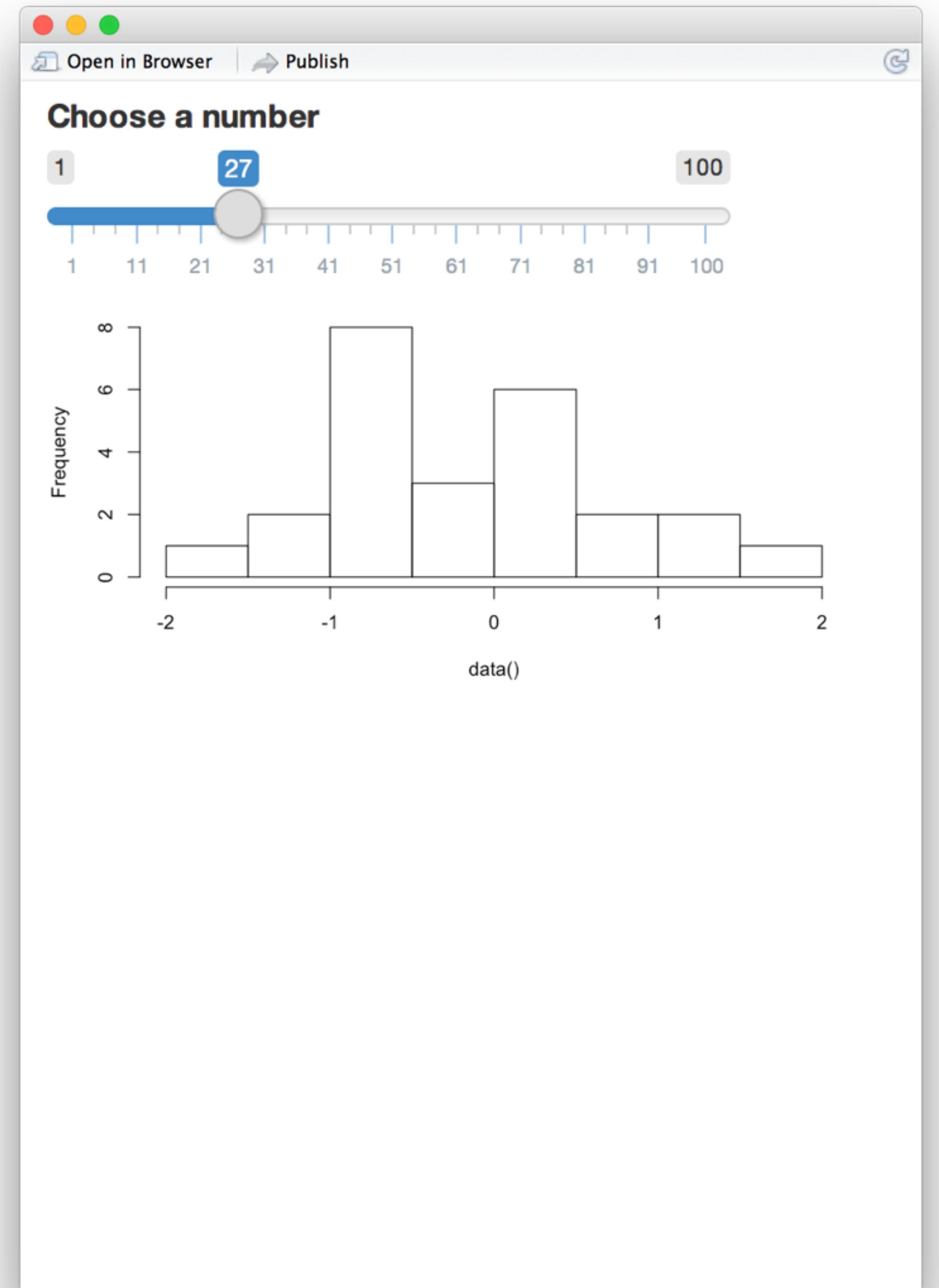


```
library(shiny)

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

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

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

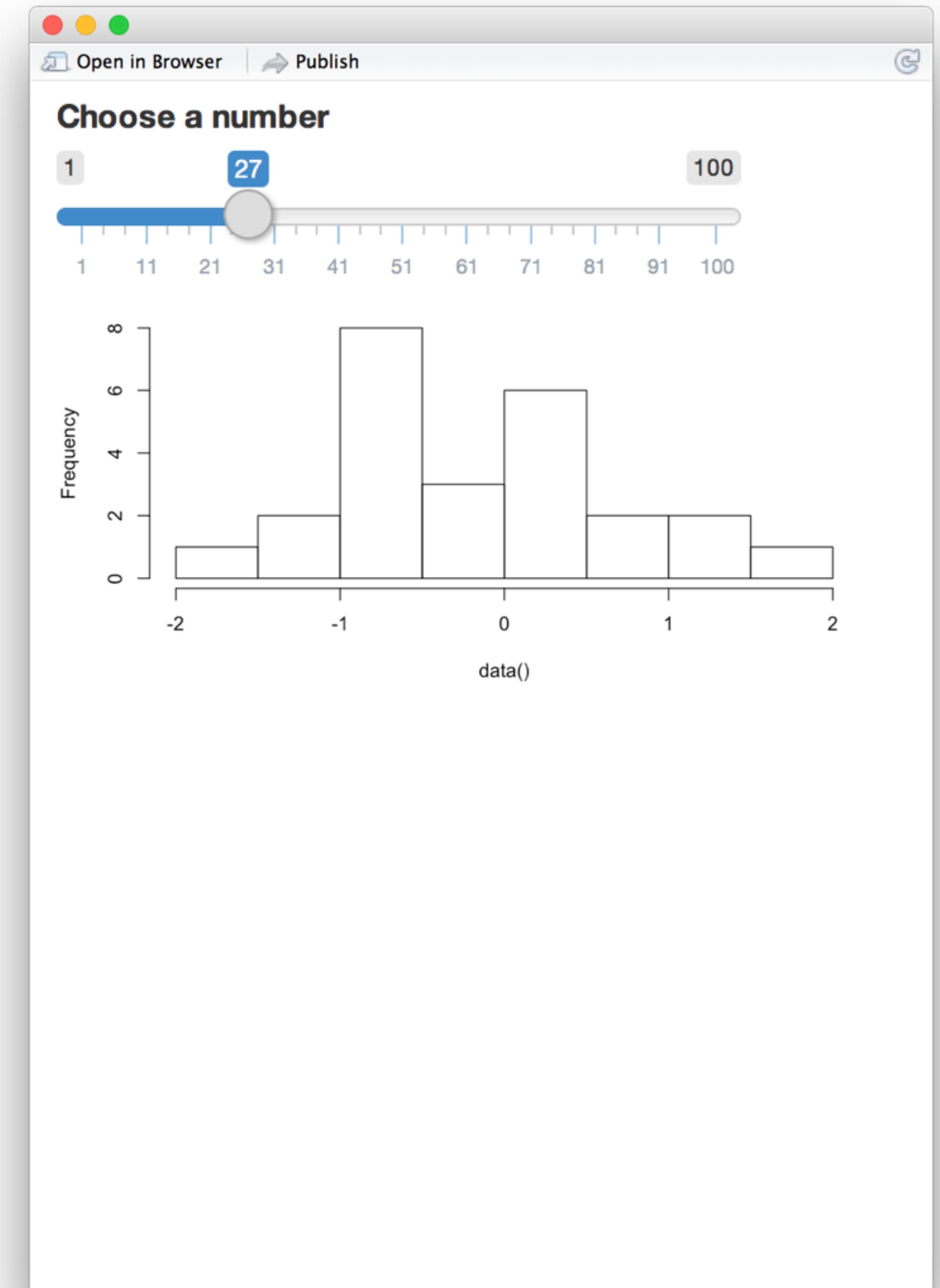


```
library(shiny)

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

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

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



```

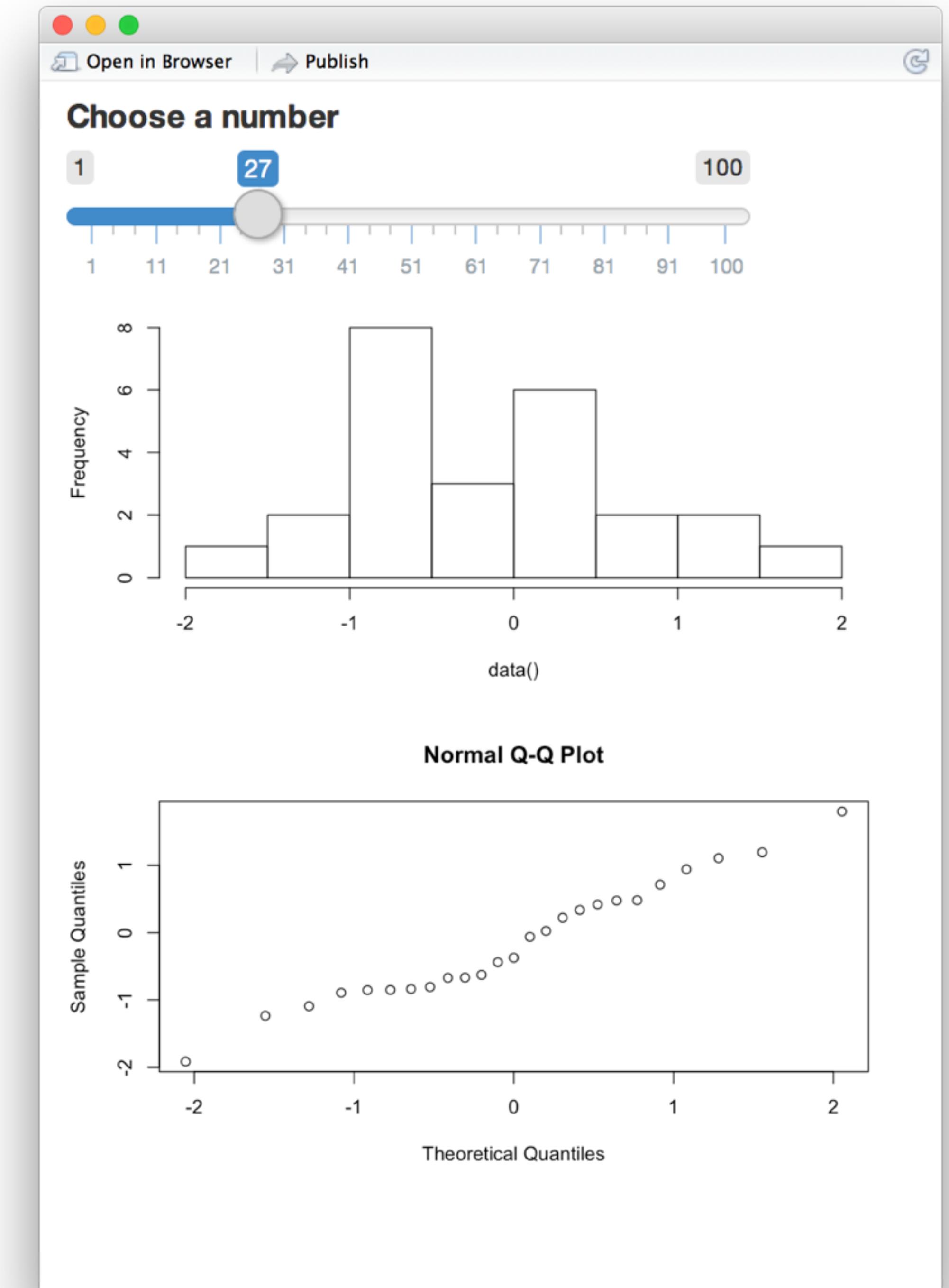
library(shiny)

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

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

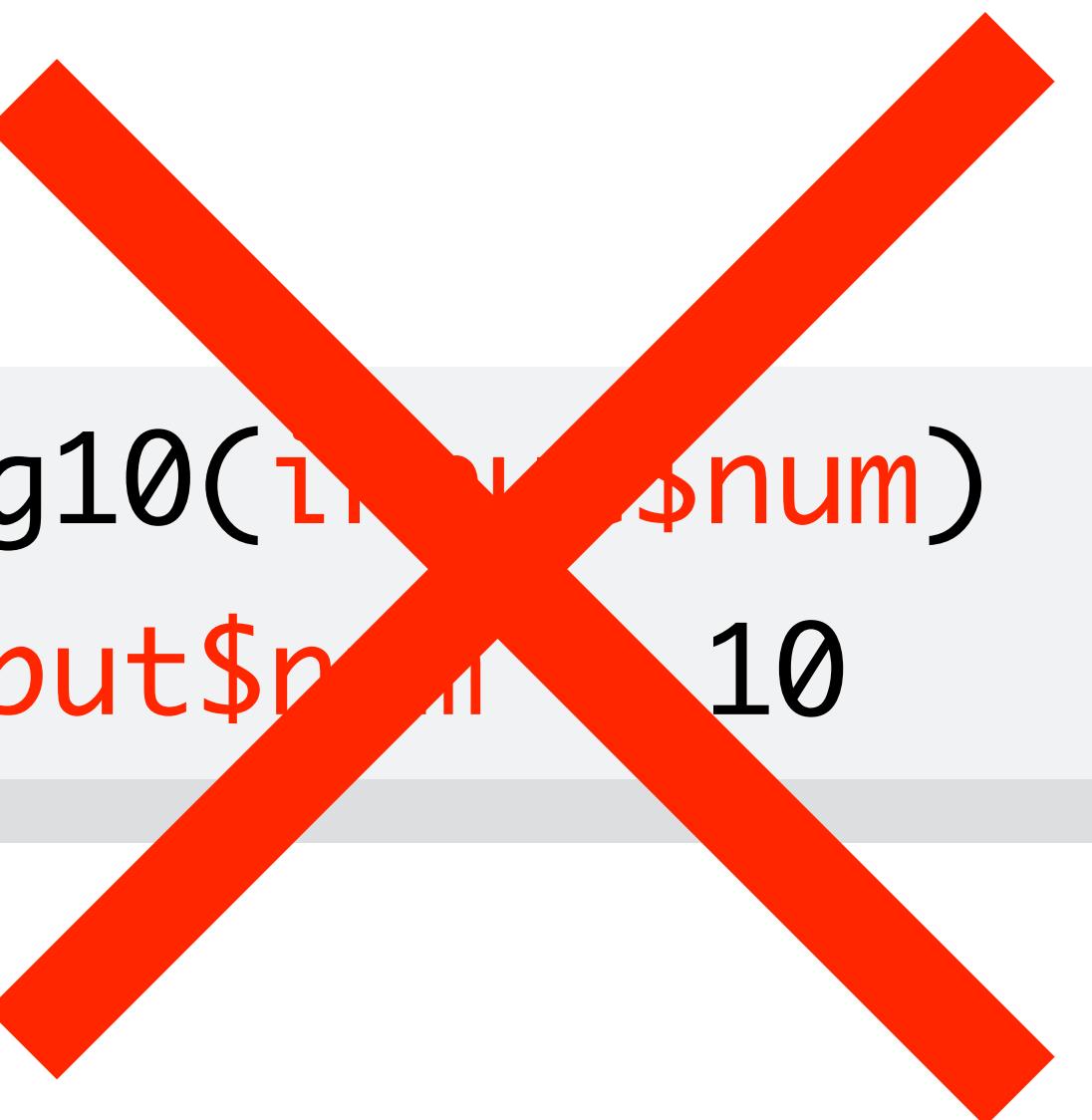
shinyApp(ui = ui, server = server)

```



# Reactivity 102

You can **only** call a reactive value from within a shiny function.

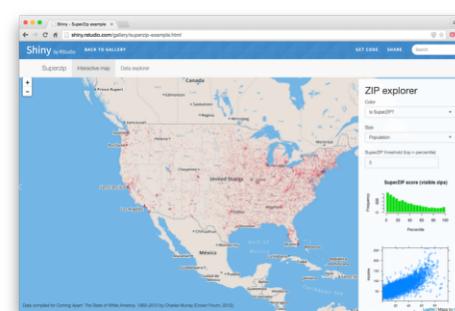


```
log10(input$num)  
input$num / 10
```

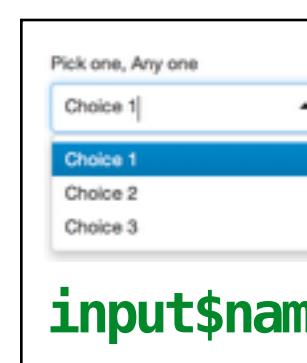
# Recap: Shiny apps

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

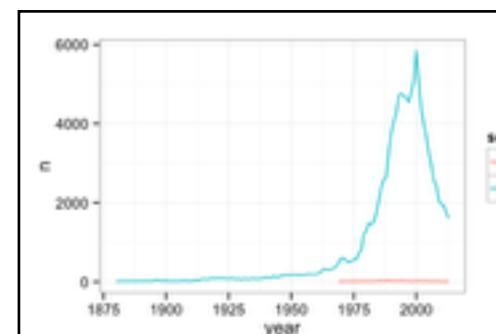
Build an app with a **ui object**, a **server function**, and **shinyApp()**



The ui object lays out what your user will see

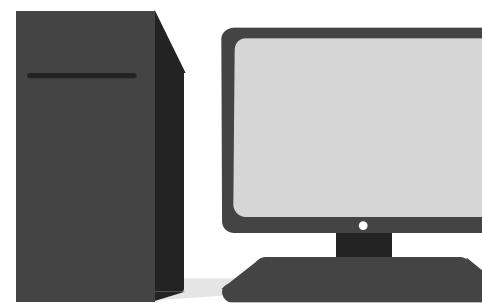


Create reactive inputs with an **\*Input()** function



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

# Recap: Shiny apps



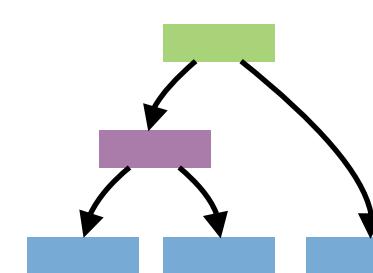
Give the server a set of instructions  
with the `server` function

`output$qq <-`

Save reactive output to `output$`

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

Build reactive output with a `render*`()  
function



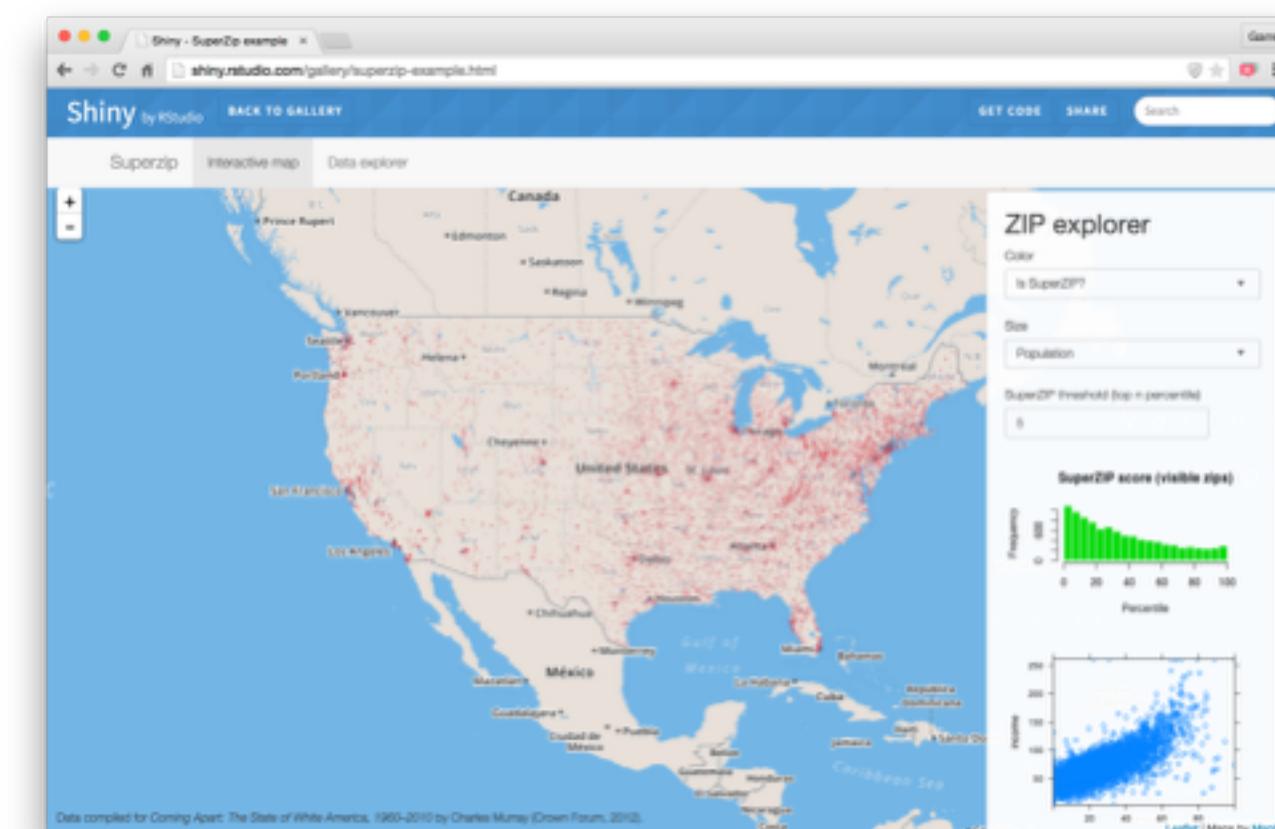
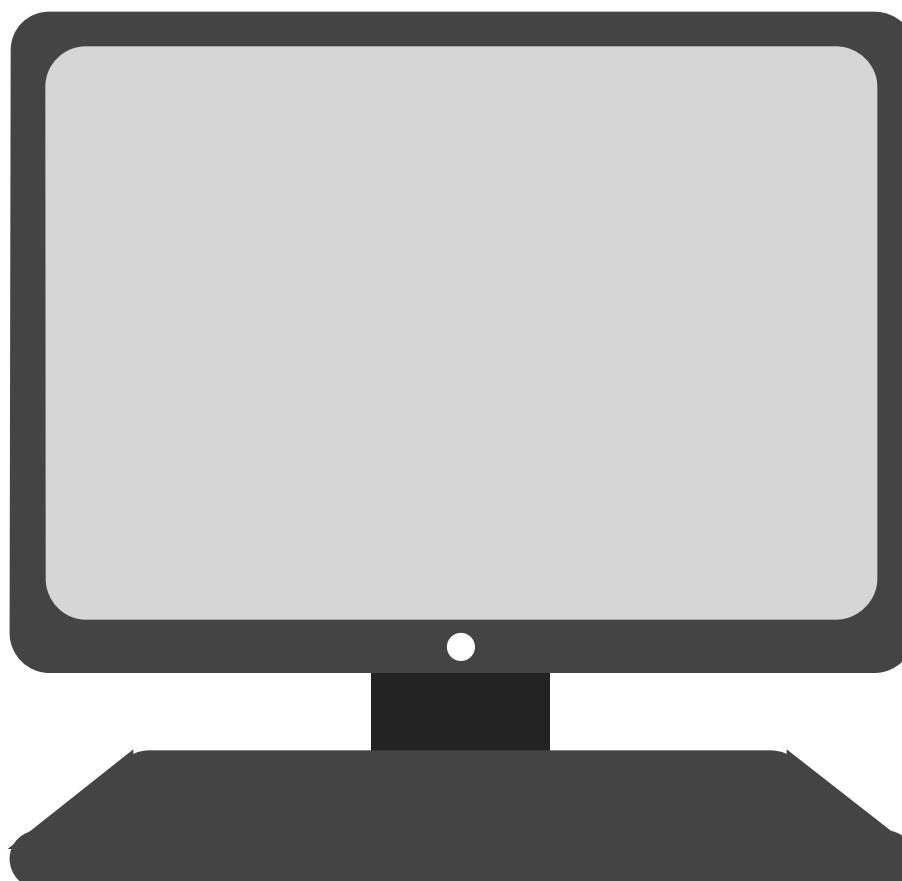
Create reactivity by chaining `Inputs` to  
`(reactive expressions to)` `rendered displays`

# **Reactive toolkit**

**(5 indispensable functions)**

# Tip

These functions give the server instructions that explain **what** code to run and **when** to run it.



# Guiding questions

**1** What code will the server run?

- some code that uses **reactive values**

**2** When will it run it?

- when a **reactive value** changes

**render\*()**

# render\*()

Builds reactive output to display in UI.

The return value must be assigned to `output$outputId`.

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

a shiny function  
(can handle reactive values)

reruns whenever a reactive  
value in the block changes

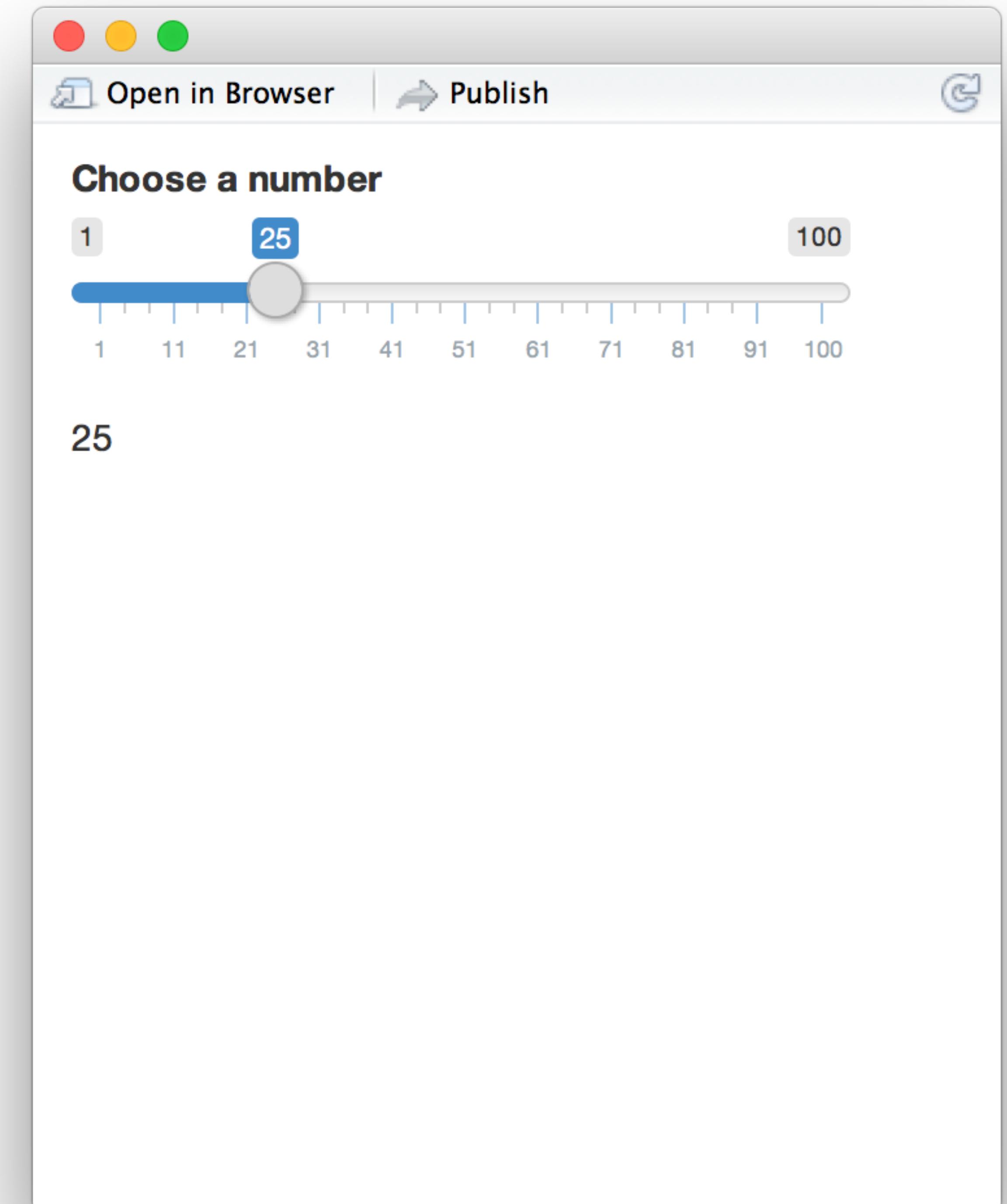
code block to run

Use the 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

```
library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  textOutput("n")
)
server <- function(input, output) {
  output$n <- renderText({
    input$num
  })
}
shinyApp(ui = ui, server = server)
```



reactive()

# reactive()

Builds a new reactive value (reactive expression)

```
data <- reactive( { rnorm(input$num) })
```

A function  
to call

a shiny function  
(can handle reactive values)

Responds to changes  
in any reactive value  
in block

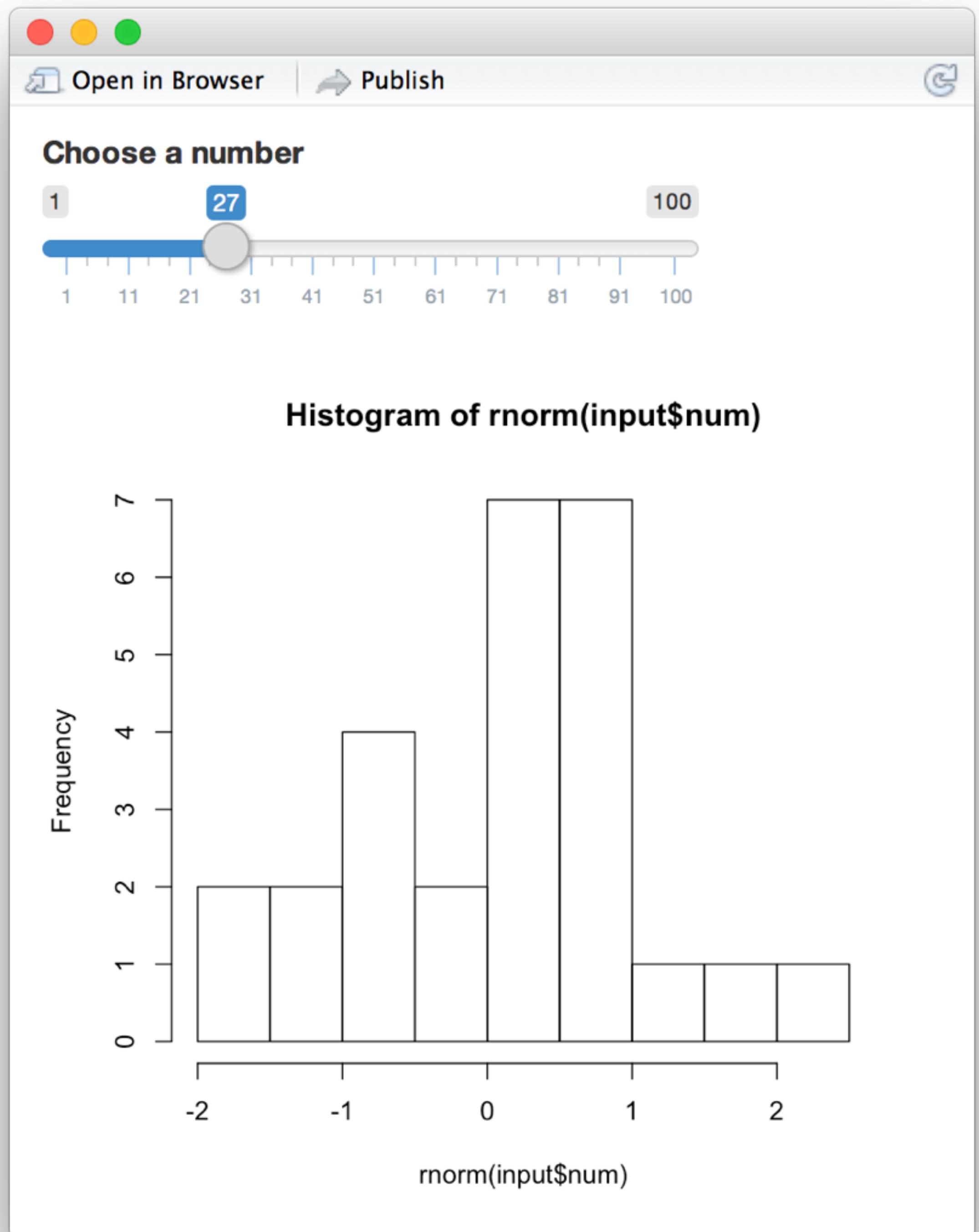
code block to  
run when data()  
is called

```
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)
```

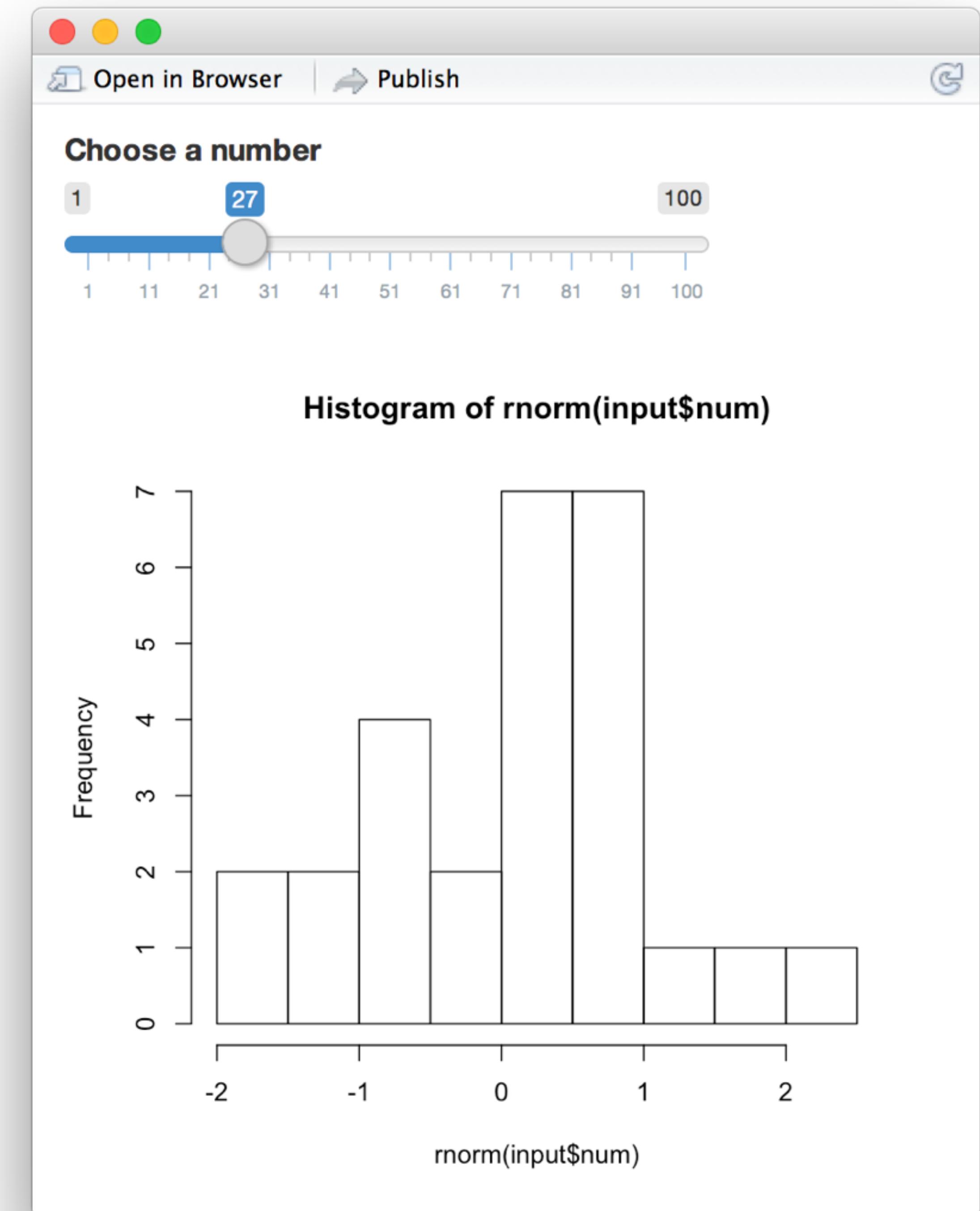


```

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

server <- function(input, output) {
  data <- reactive({rnorm(input$num)})
  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
}
shinyApp(ui = ui, server = server)

```

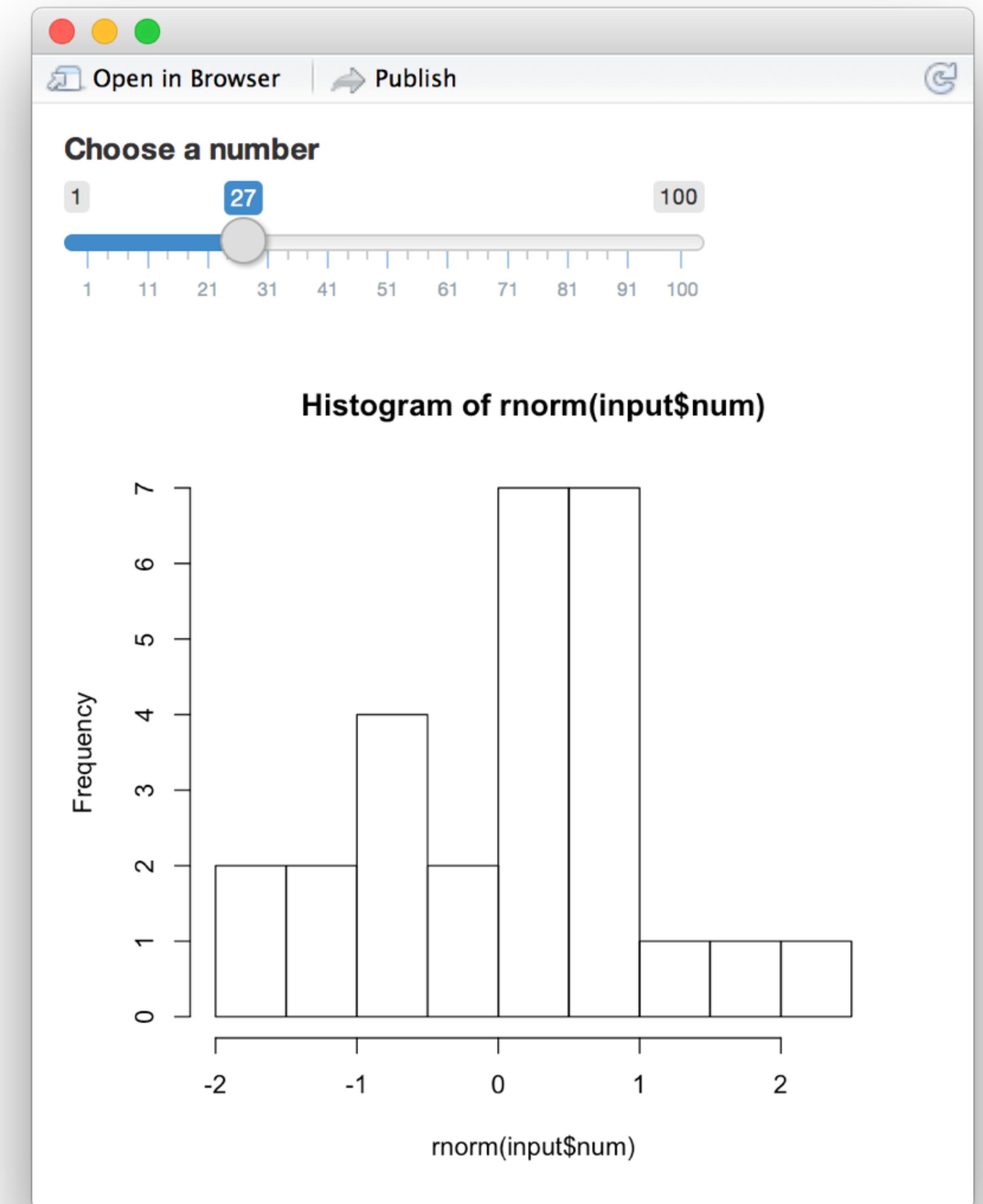


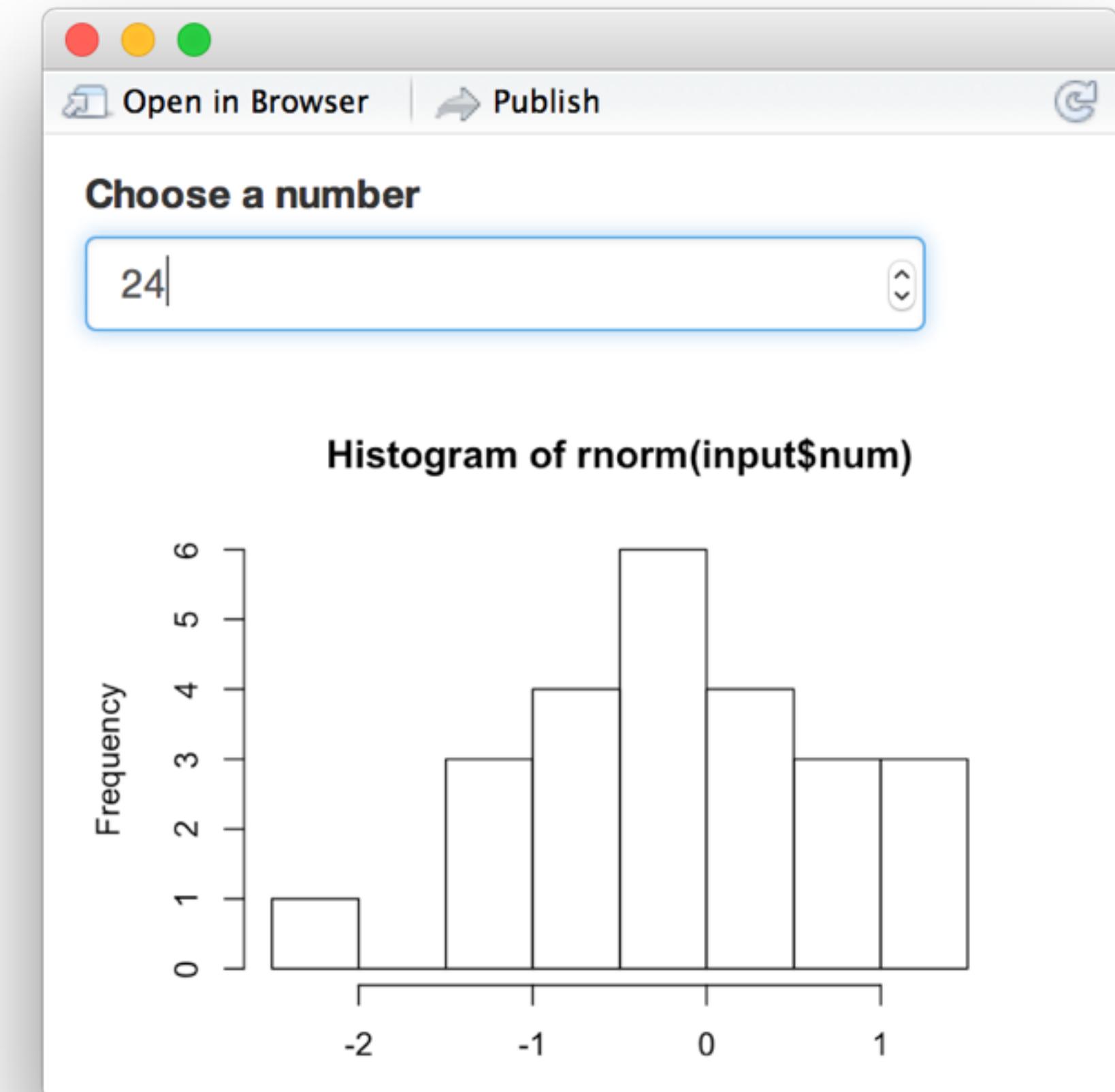
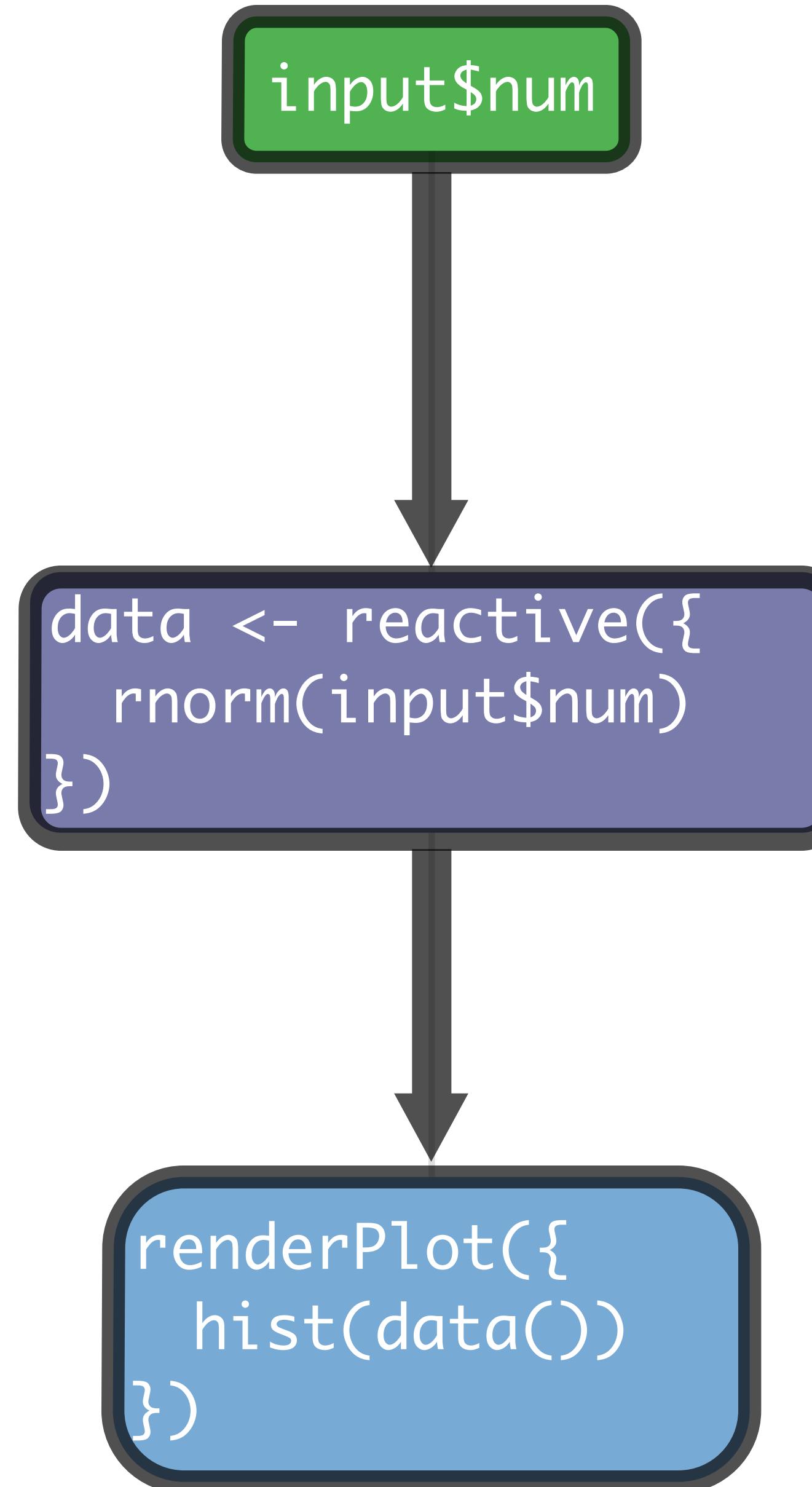
```

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

server <- function(input, output) {
  data <- reactive({rnorm(input$num)})
  output$hist <- renderPlot({
    hist(data())
  })
}
shinyApp(ui = ui, server = server)

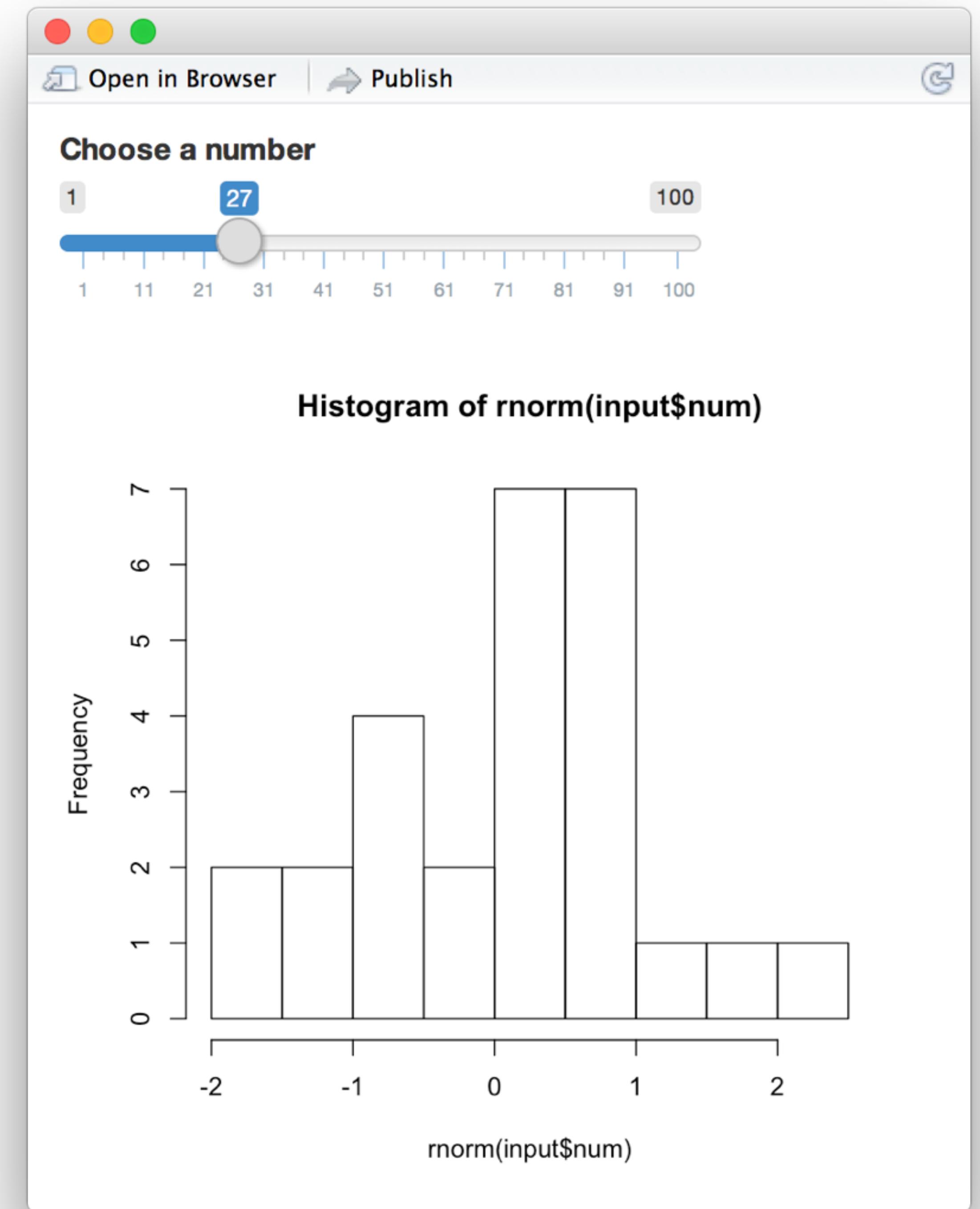
```





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

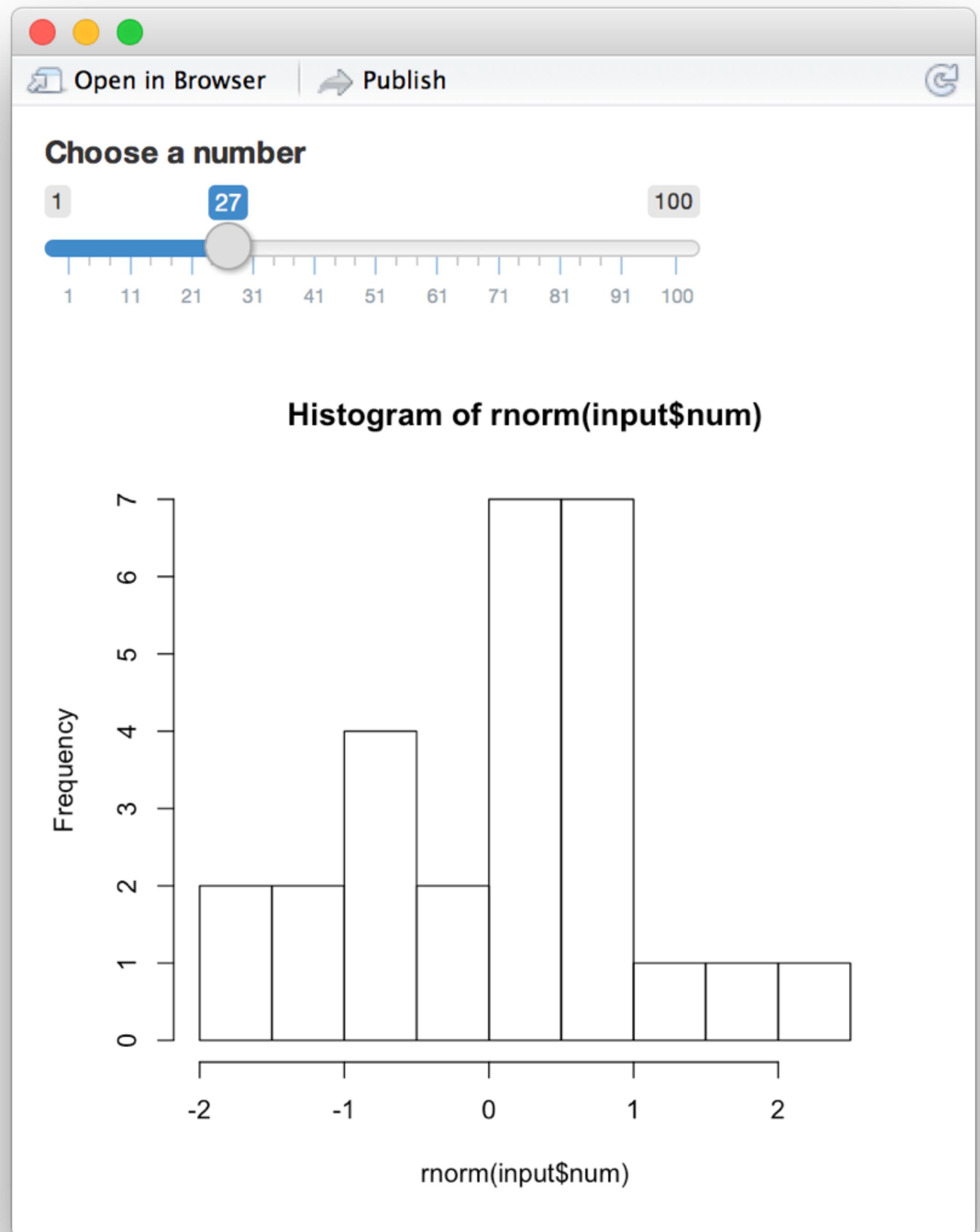
)
server <- function(input, output) {
  data <- reactive(rnorm(input$num))
  output$hist <- renderPlot({
    hist(data())
  })
}
shinyApp(ui = ui, server = server)
```



```

library(shiny)
ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  plotOutput("hist"),
  plotOutput("hist2")
)
server <- function(input, output) {
  data <- reactive(rnorm(input$num))
  output$hist <- renderPlot({
    hist(data())
  })
  output$hist2 <- renderPlot({
  })
}
shinyApp(ui = ui, server = server)

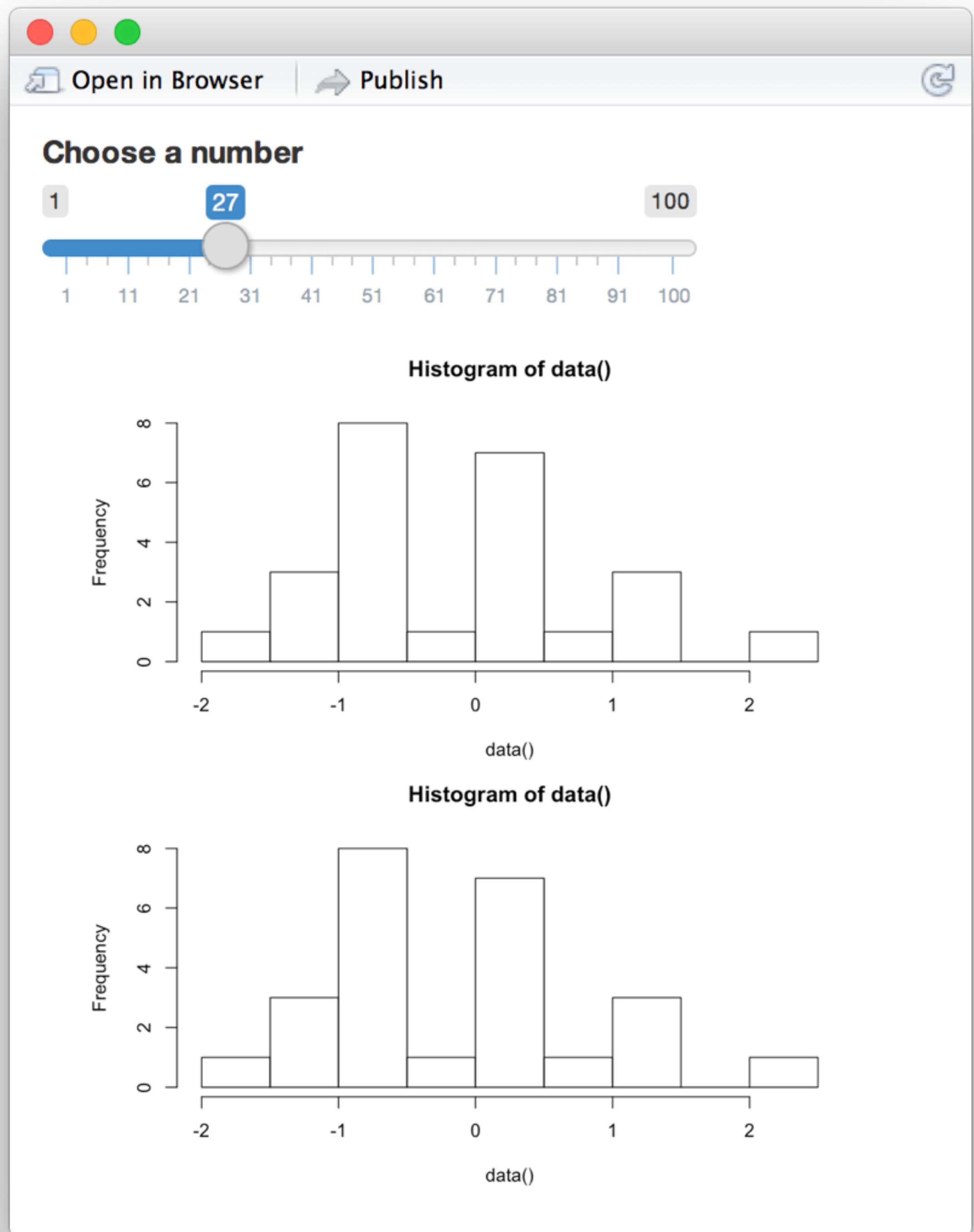
```

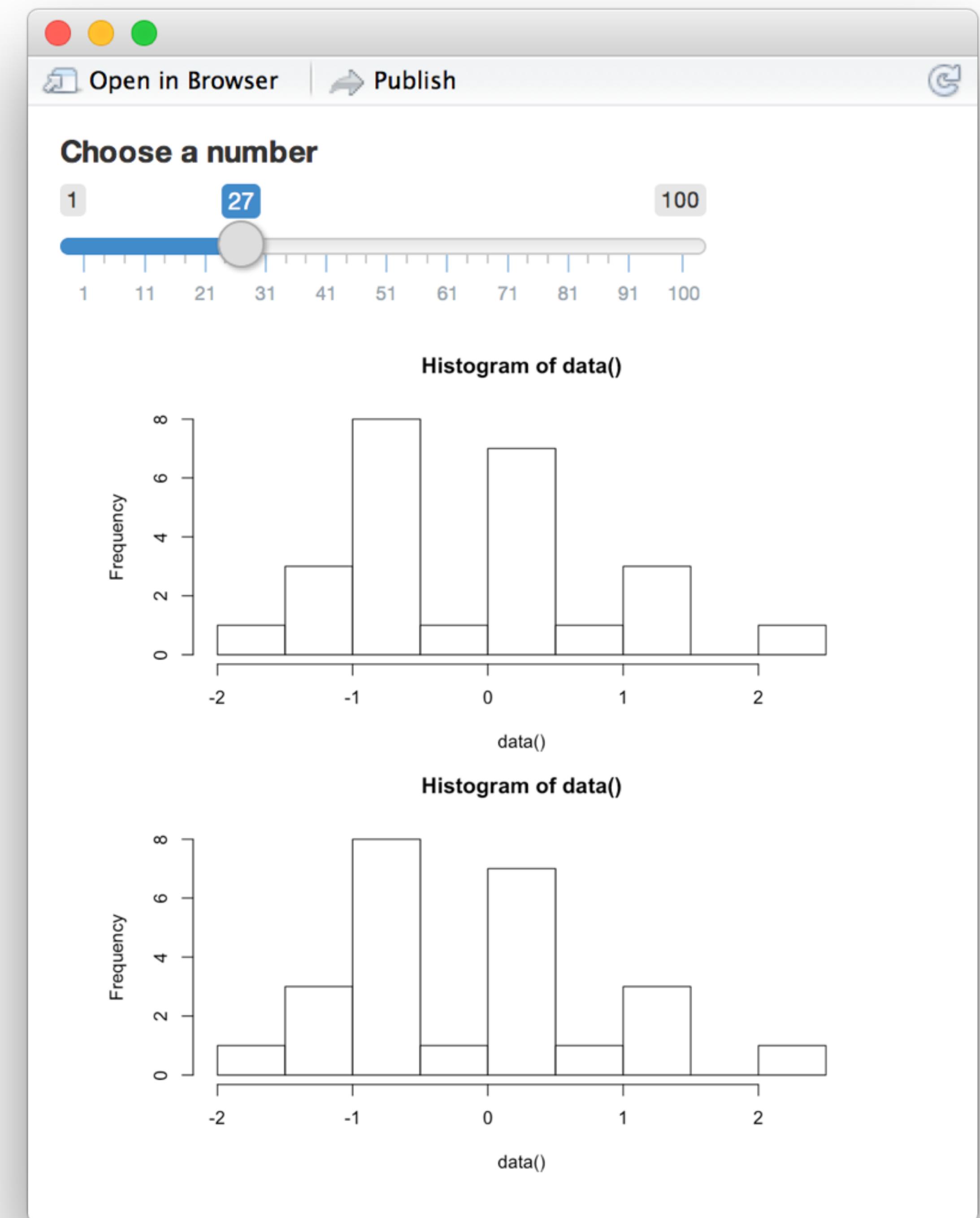
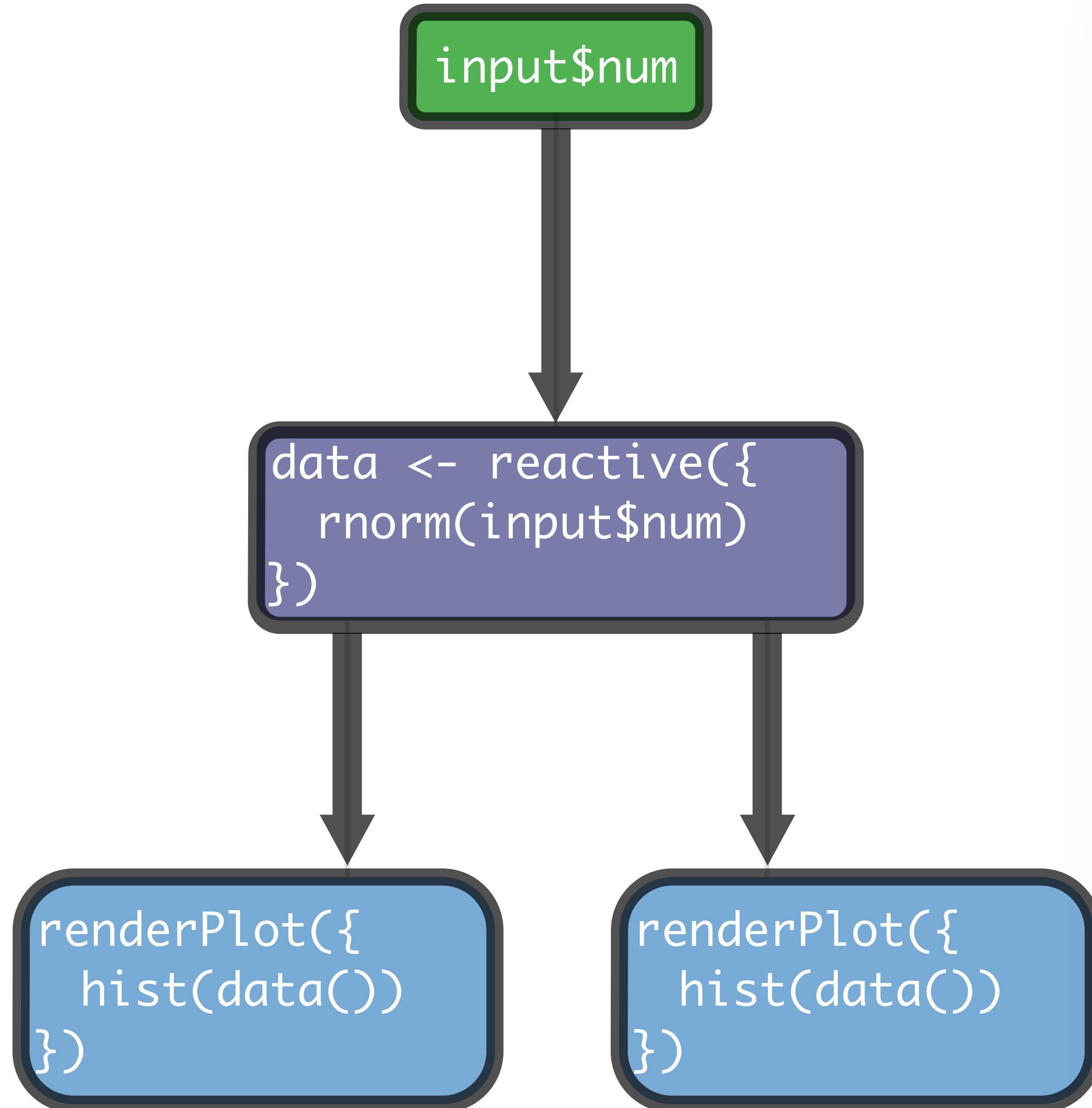


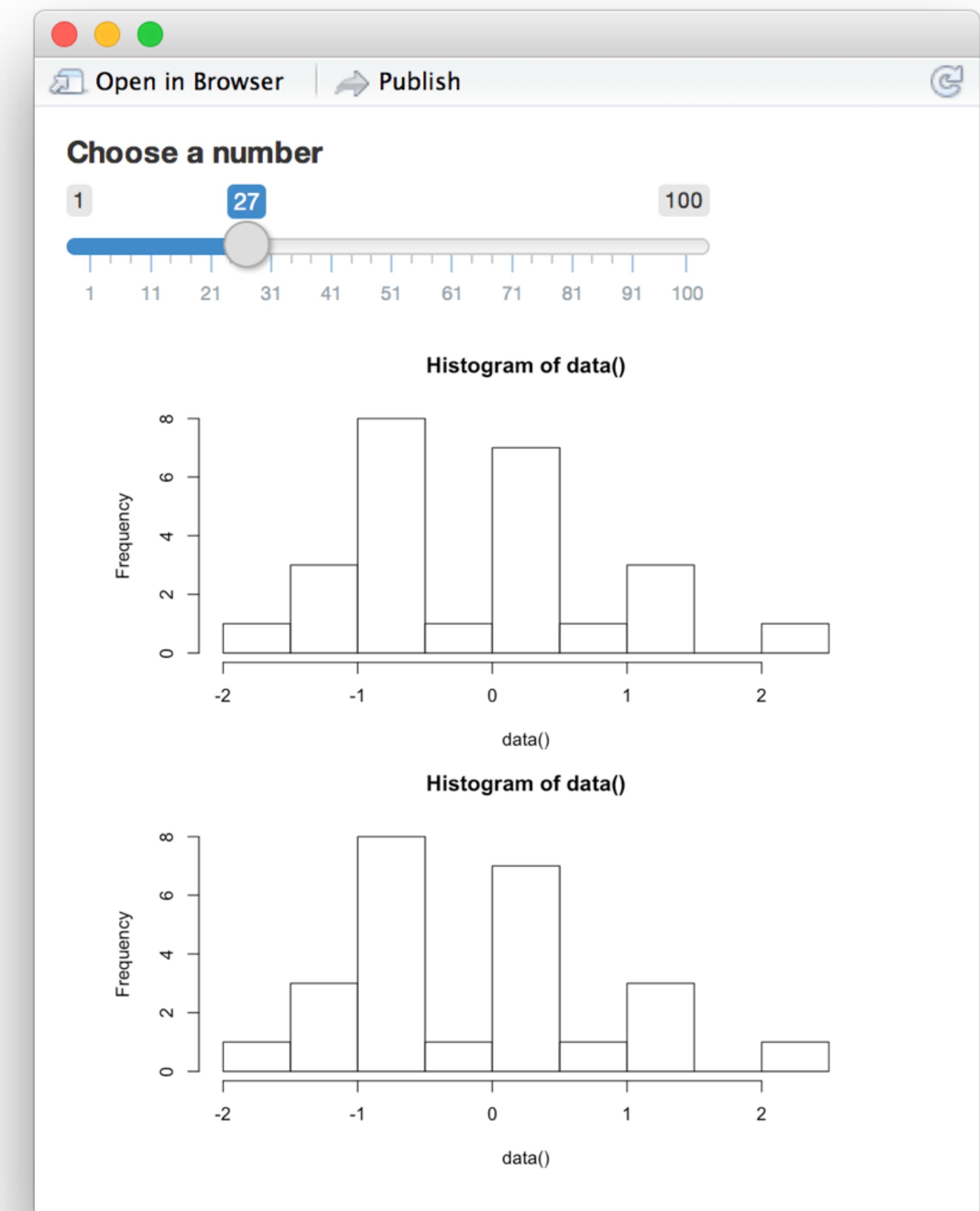
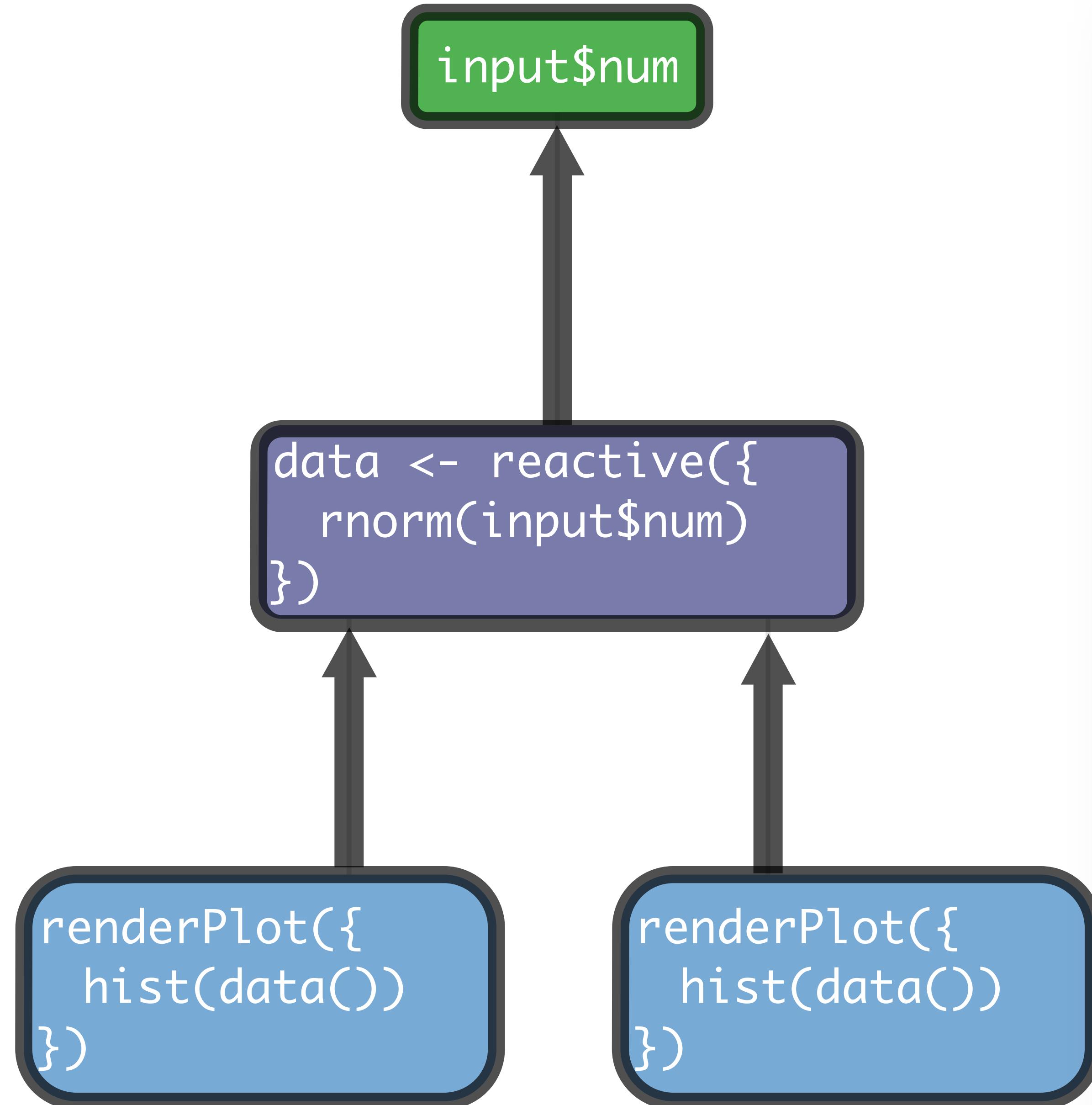
```

library(shiny)
ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  plotOutput("hist"),
  plotOutput("hist2")
)
server <- function(input, output) {
  data <- reactive(rnorm(input$num))
  output$hist <- renderPlot({
    hist(data())
  })
  output$hist2 <- renderPlot({
    hist(data())
  })
}
shinyApp(ui = ui, server = server)

```



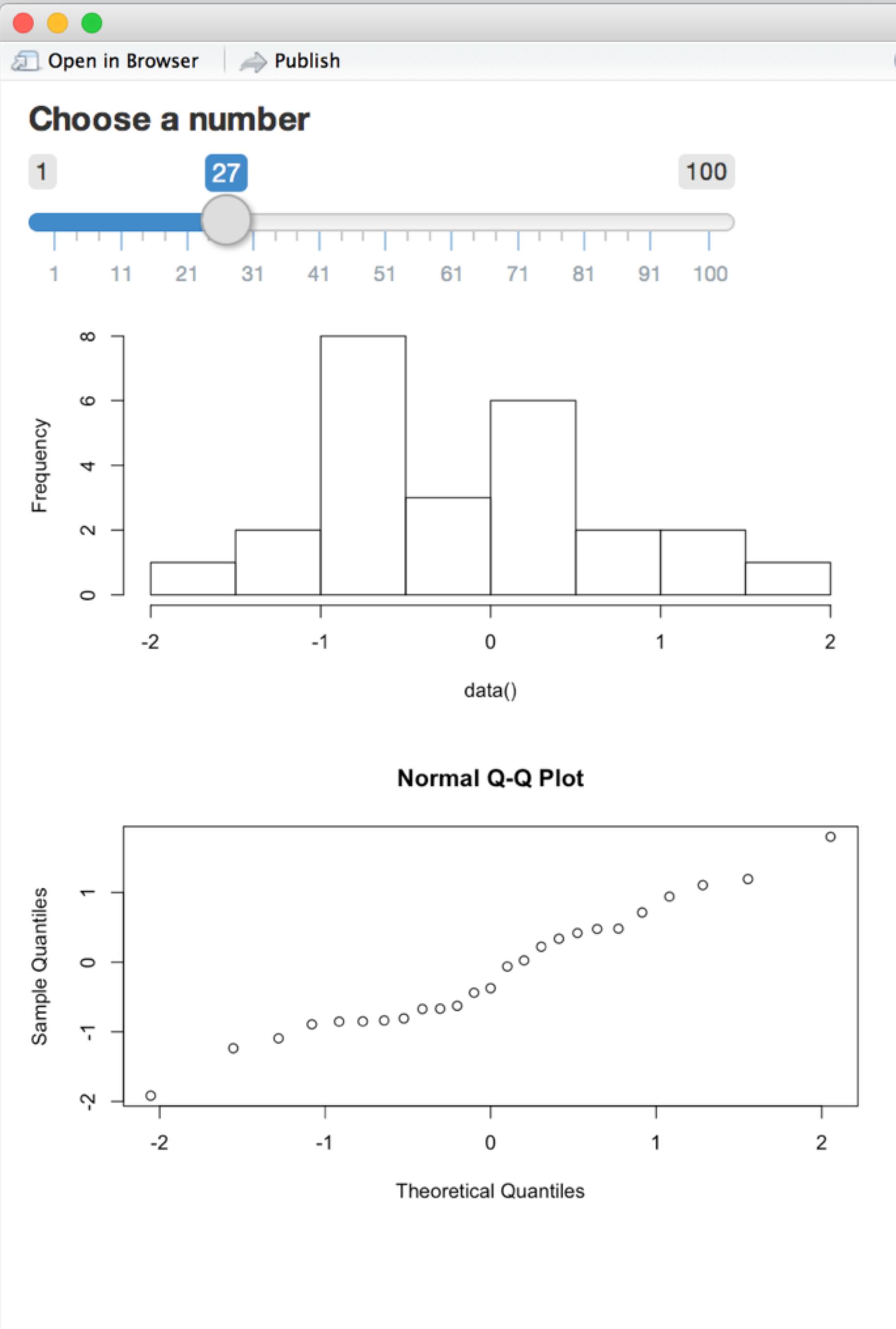




# A reactive expression is special in two ways

```
data()
```

- 1** You call a reactive expression like a function
- 2** Reactive expressions **cache** their values  
(if the expression knows that it is up-to-date, it will return its most recent value, which is saved in memory)



# Your turn

Rewrite your app to use a reactive expression.

Make sure the histogram and qqplot display the same data.

05 : 00

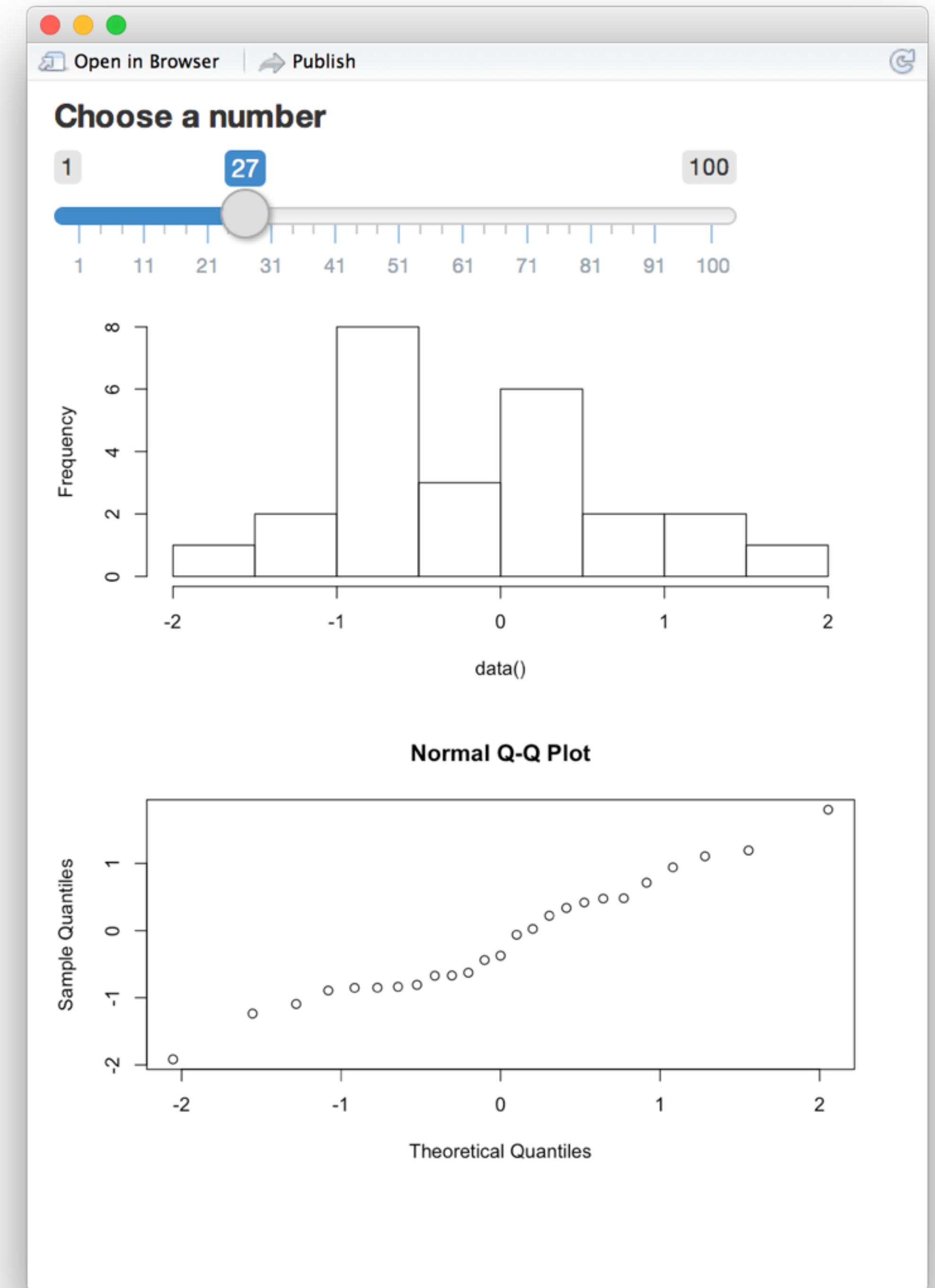
```
library(shiny)

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

server <- function(input, output) {

  output$hist <- renderPlot({hist( )})
  output$qq <- renderPlot({qqnorm( )})
}

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



```

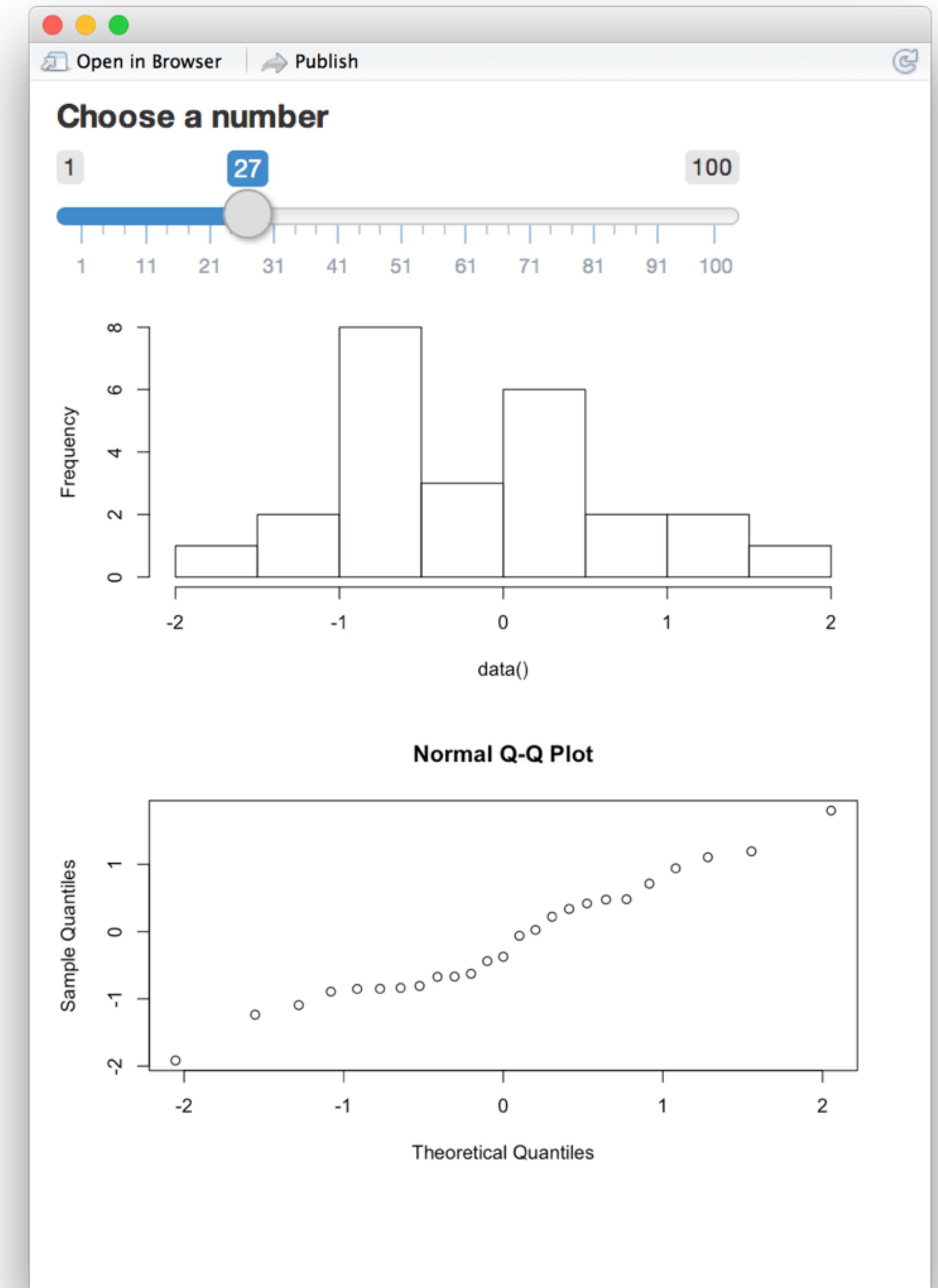
library(shiny)

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

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

shinyApp(ui = ui, server = server)

```



```

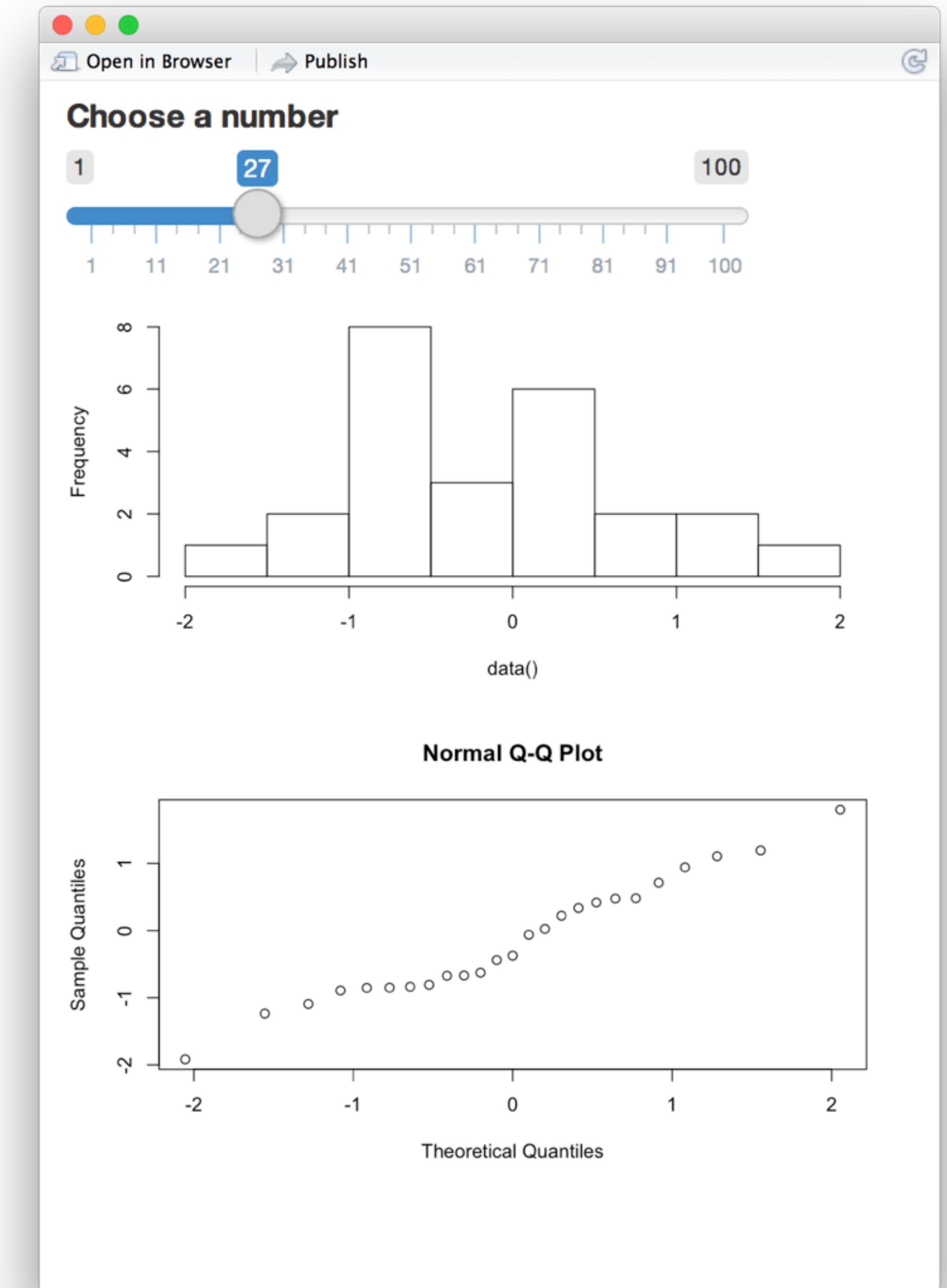
library(shiny)

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

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

shinyApp(ui = ui, server = server)

```



**isolate()**

# isolate()

Returns the result as a non-reactive value

```
isolate({ rnorm(input$num) })
```

a shiny function  
(can handle reactive values)

code block  
to run

code will NOT rerun when a  
reactive value changes

# observeEvent()

# observeEvent()

Triggers code to run on server

```
observeEvent(input$num, { print(input$num) })
```

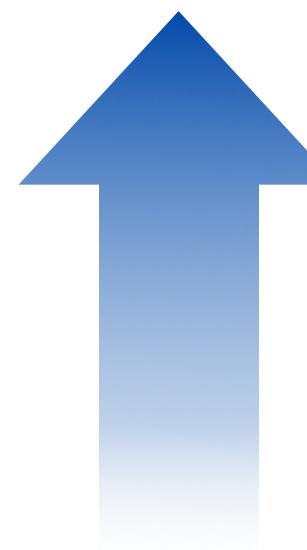
a shiny function  
(can handle reactive values)

a reactive value to  
monitor

code block to run whenever  
the value changes

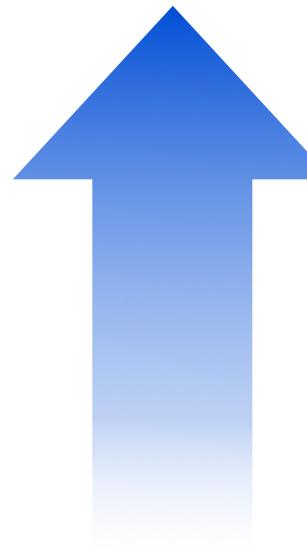
# observeEvent()

```
observeEvent(input$num, { print(input$num) })
```



**2** When will it run it?

Each time this value changes

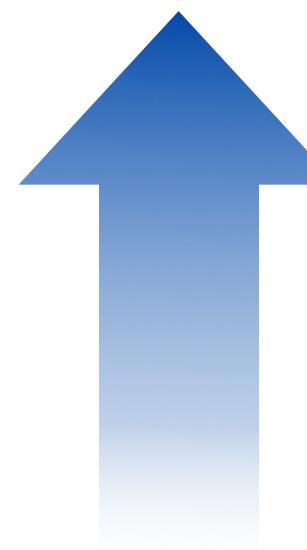


**1** What code will the server run?

All of the code in the code block

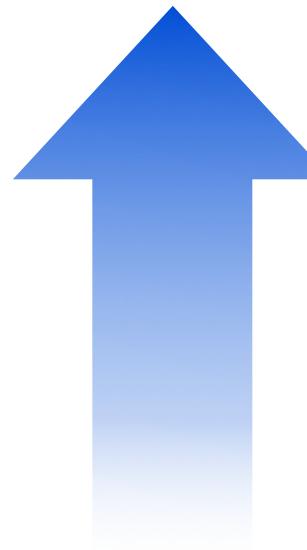
# observeEvent()

```
observeEvent(input$num2, { print(input$num) })
```



**2** When will it run it?

Each time this value changes

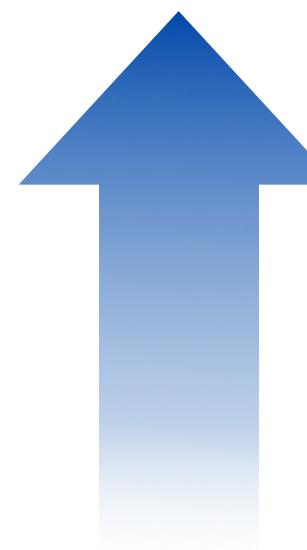


**1** What code will the server run?

All of the code in the code block

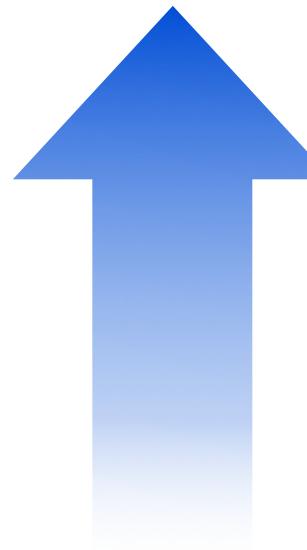
# observeEvent()

```
observeEvent(input$num2, { print("Hello") })
```



**2** When will it run it?

Each time this value changes



**1** What code will the server run?

All of the code in the code block

eventReactive()

# eventReactive()

A reactive expression that remains valid until a specific value changes

```
data <- eventReactive(input$go, { rnorm(input$num) })
```

A function  
to call

a shiny function  
(can handle reactive values)

A reactive value to  
respond to  
(expression invalidates ONLY  
when this value changes)

code block to  
run when data()  
is called

# Recap

# Recap: Reactive toolkit

**render\***() Build reactive output to display in UI

**reactive**() Build reactive expression to use in code

**isolate**() Build non-reactive result to use in code

**observeEvent**() Trigger code on server

**eventReactive**() Build expression that only reacts to set value(s)

# Layout

## 2

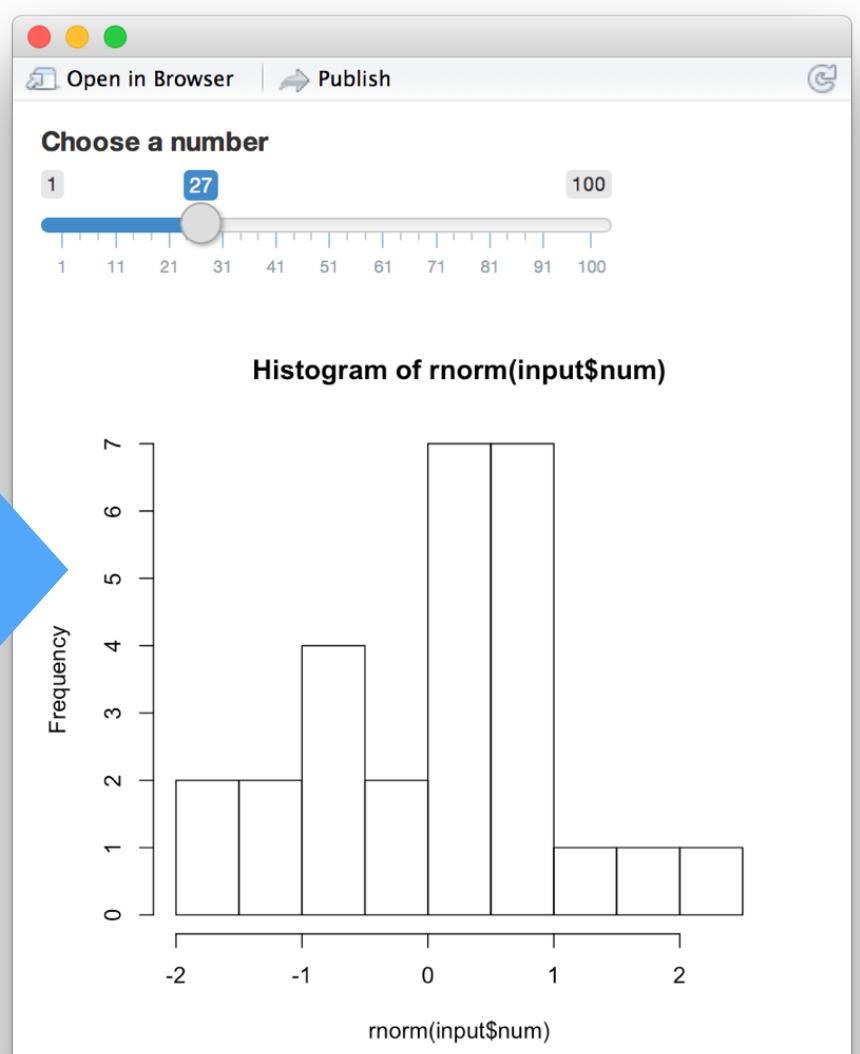
## Add elements to your app as arguments to fluidPage()

```
fluidPage(  
  # input controls,  
  # output controls,  
  # layout objects,  
  # custom HTML/CSS/JavaScript  
)
```

# Shiny's UI functions return HTML

```
fluidPage(  
  sliderInput(inputId = "num",  
    label = "Choose a number",  
    val = 25, min = 1, max = 100),  
  plotOutput("hist")  
)
```

```
<div class="container-fluid">  
  <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"/>  
  </div>  
  <div id="hist" class="shiny-plot-output" style="width: 100% ; height: 400px"></div>  
</div>
```



# Layout functions

Add HTML that divides the UI into a grid

`fluidRow()`

```
<div class="row"></div>
```

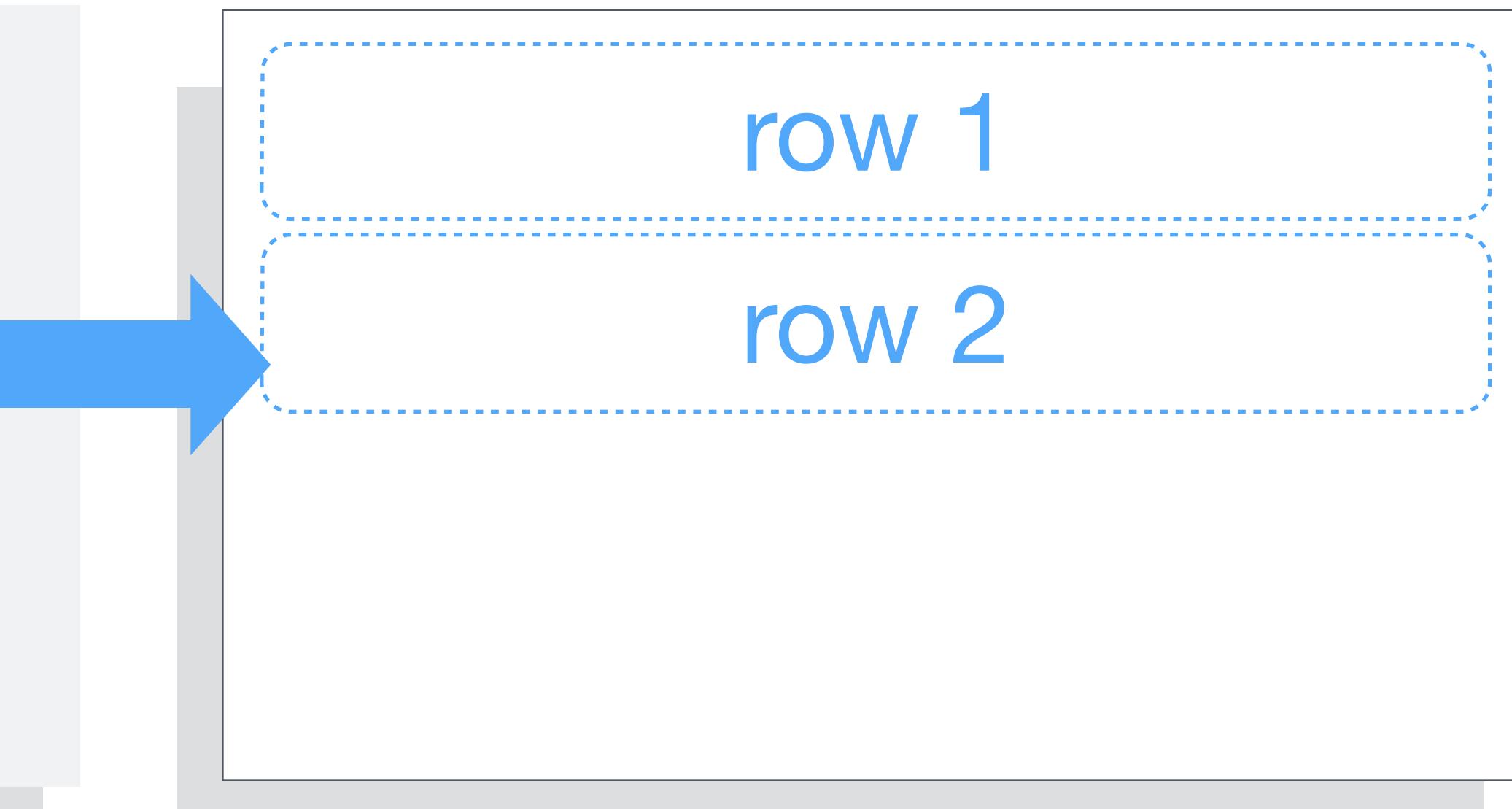
`column(width = 2)`

```
<div class="col-sm-2"></div>
```

# fluidRow()

`fluidRow()` adds rows to the grid. Each new row goes below the previous rows.

```
ui <- fluidPage(  
  fluidRow(),  
  fluidRow()  
)
```

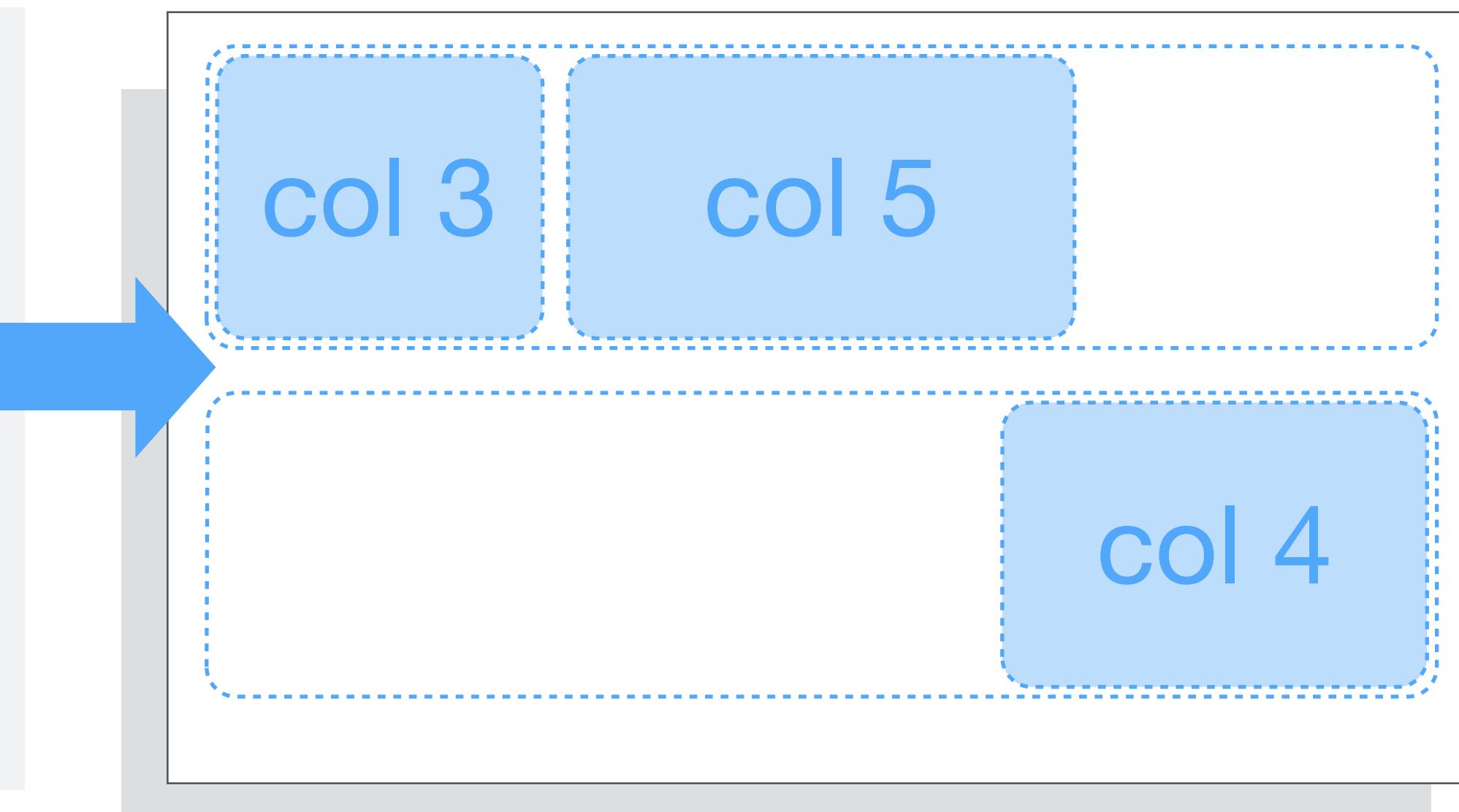


# column()

column() adds columns within a row. Each new column goes to the left of the previous column.

Specify the **width** and **offset** of each column out of 12

```
ui <- fluidPage(  
  fluidRow(  
    column(3),  
    column(5)),  
  fluidRow(  
    column(4, offset = 8))
```



To place an element in the grid, call it as an argument of a layout function

```
fluidRow("In the row")
```

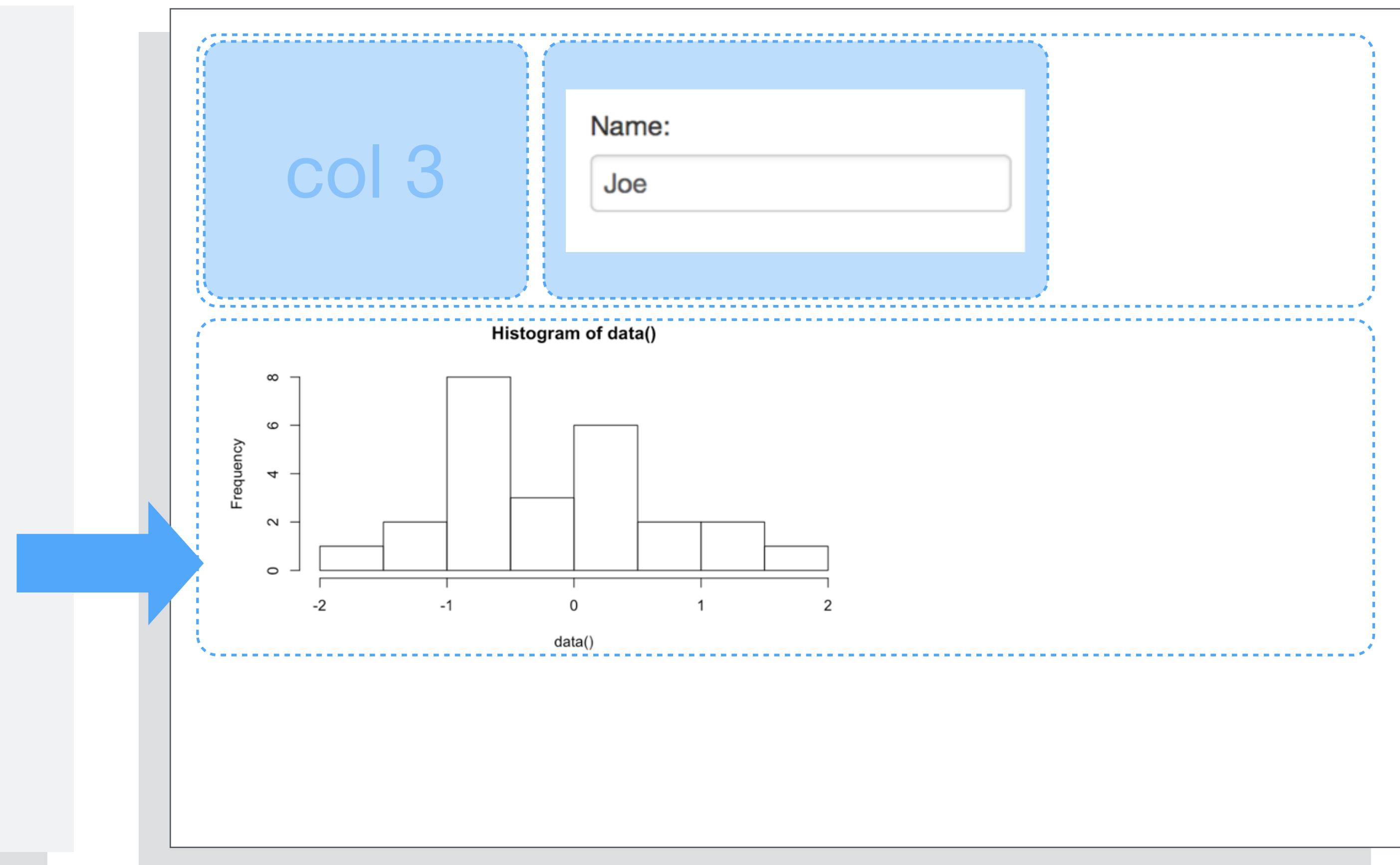
```
<div class="row">In the row</div>
```

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

```
<div class="col-sm-2">  
  <div id="hist" class="shiny-plot-output"  
    style="width: 100% ; height: 400px"></div>  
</div>
```

To place an element in the grid, call it as an argument of a layout function

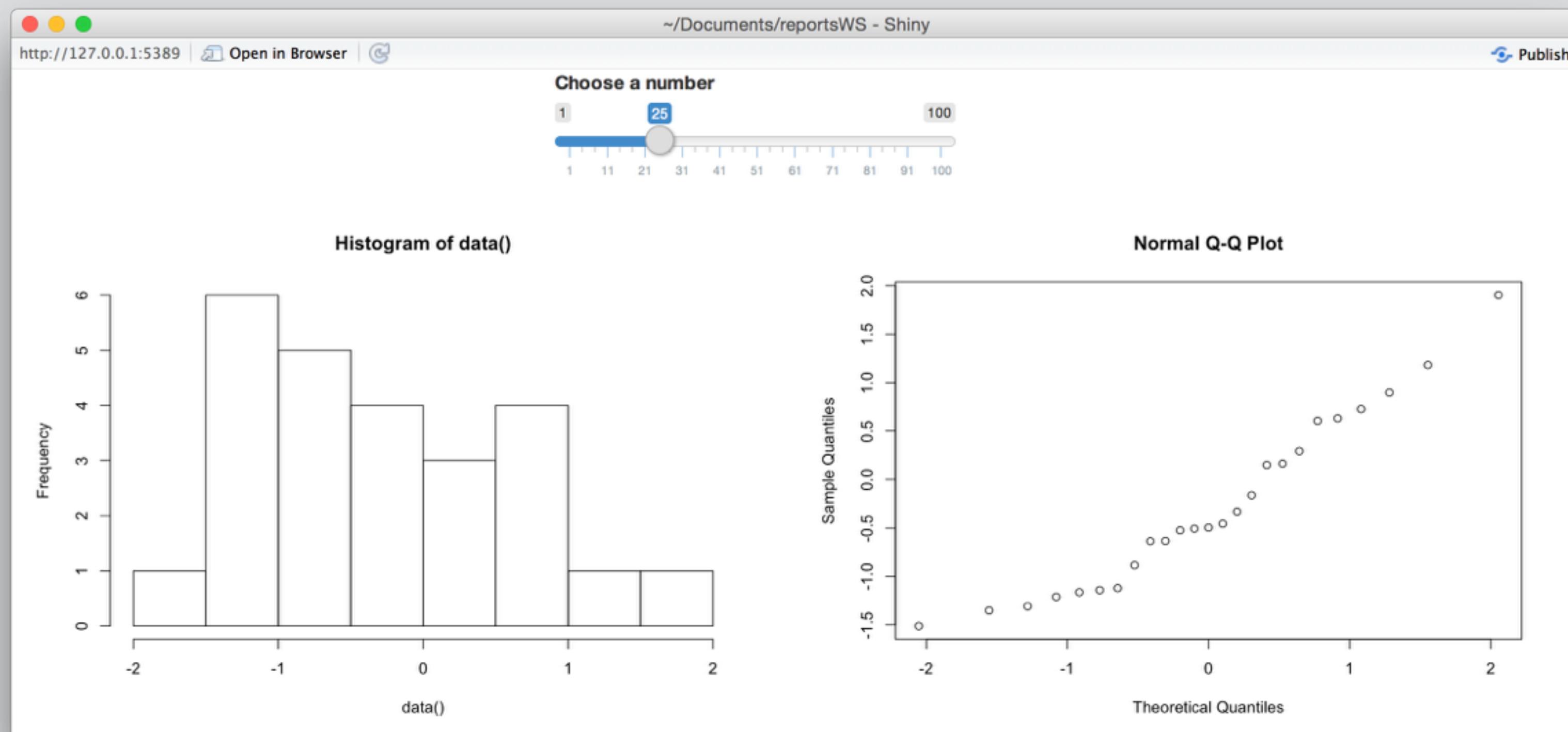
```
fluidPage(  
  fluidRow(  
    column(3),  
    column(5,  
     textInput("n", "Name:")  
    )  
,  
  plotOutput("hist")  
)
```



# Your turn

Return to the last exercise.

Use `fluidRow()` and `column()` to lay your app out like this.

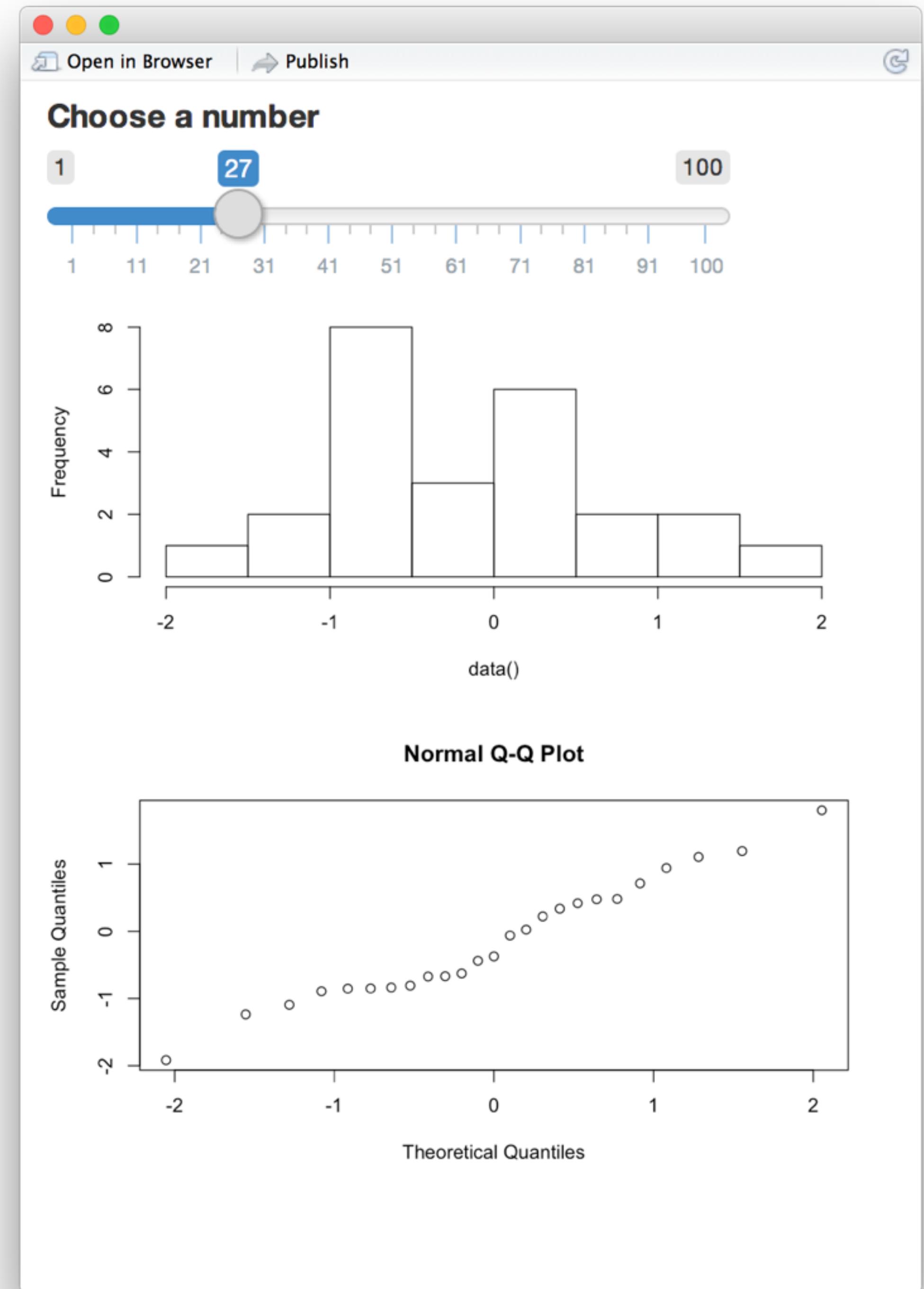


*Hint: You do not need to change anything in server*

03 : 50

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

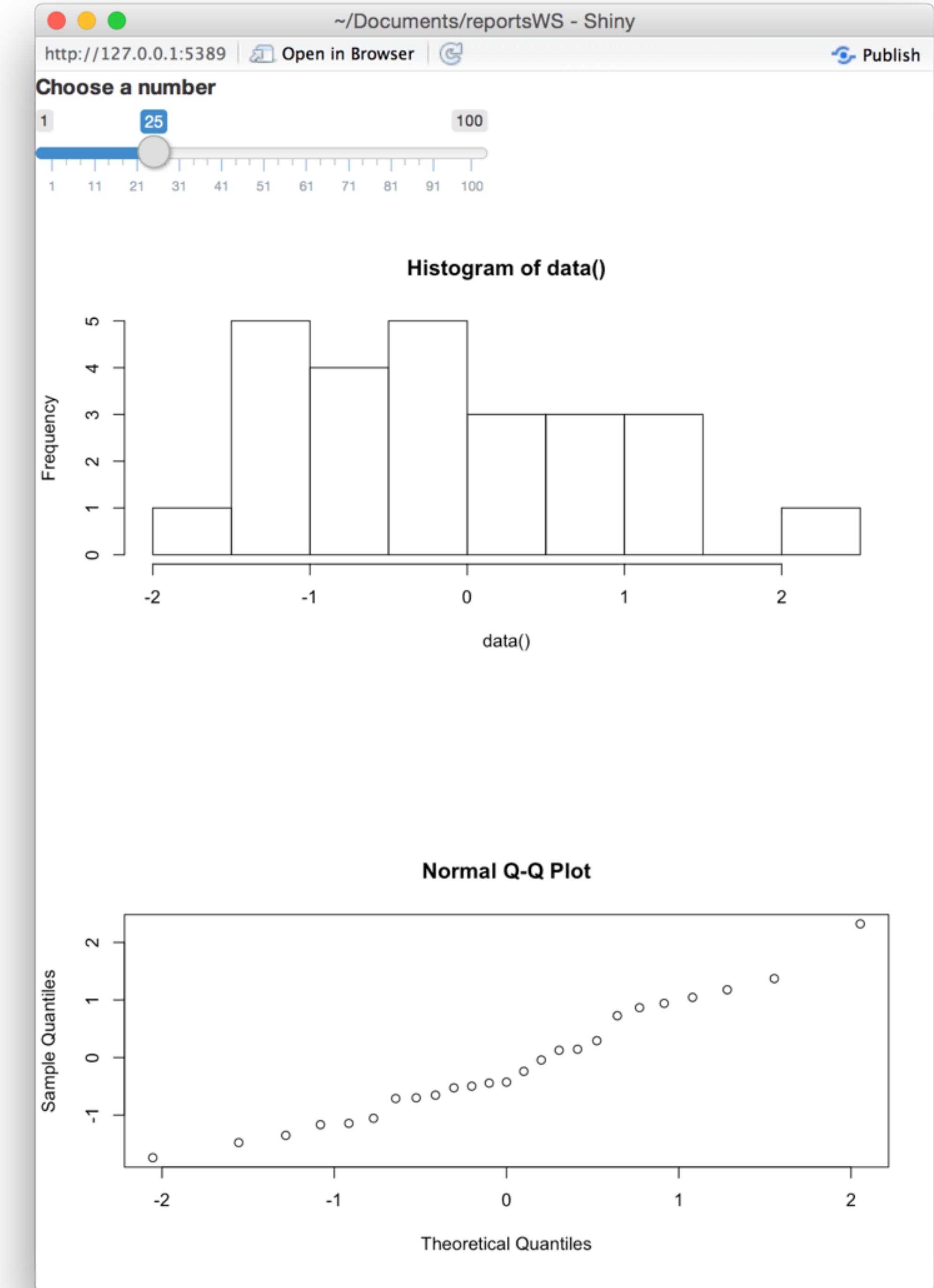
```
)
```



```

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

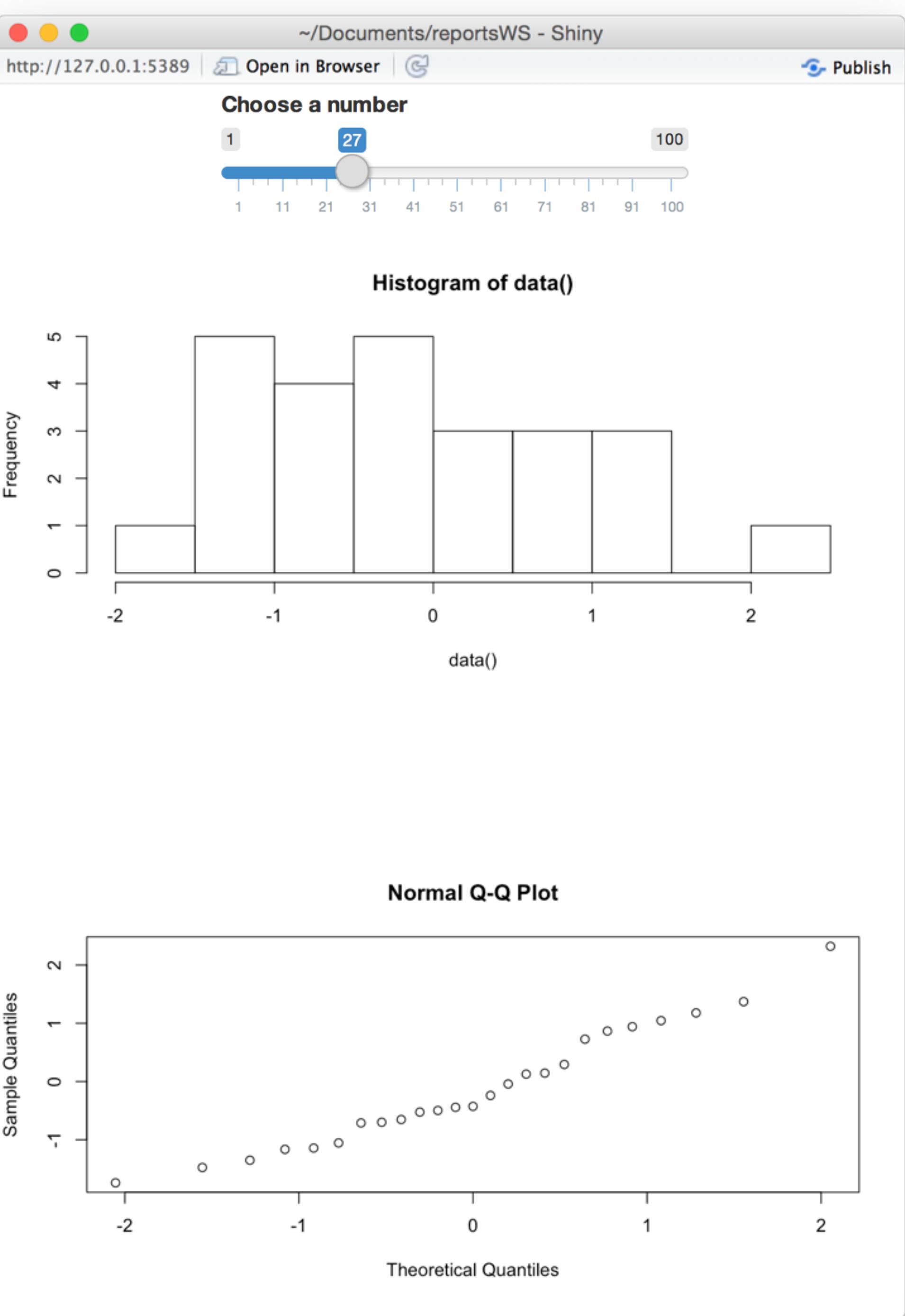
```



```

ui <- fluidPage(
  fluidRow(
    column(width = 4, offset = 4,
      sliderInput(inputId = "num",
                  label = "Choose a number",
                  val = 25, min = 1, max = 100)
    )
  ),
  fluidRow(
    plotOutput("hist"),
    plotOutput("qq")
  )
)

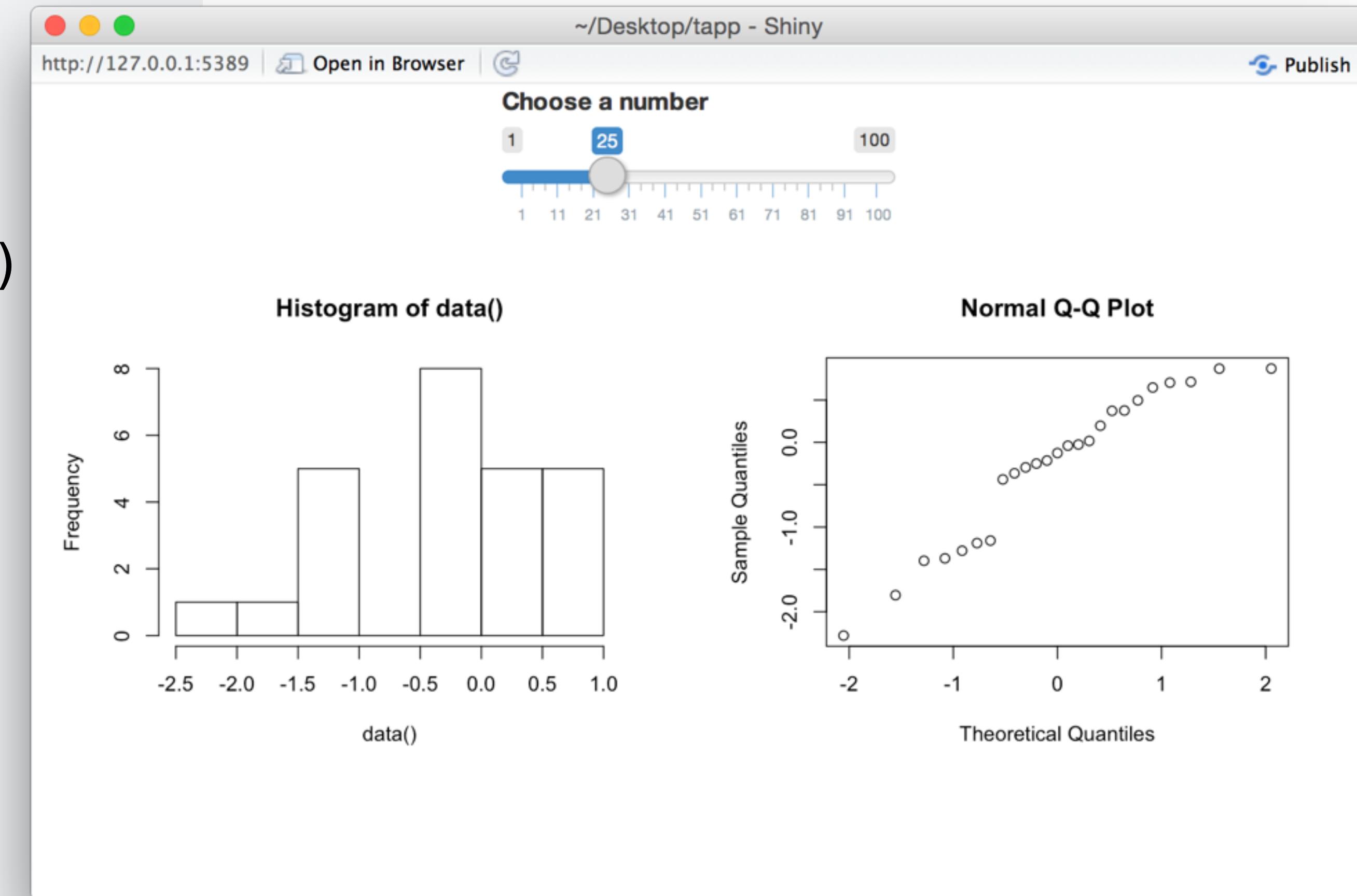
```



```

ui <- fluidPage(
  fluidRow(
    column(width = 4, offset = 4,
      sliderInput(inputId = "num",
        label = "Choose a number",
        val = 25, min = 1, max = 100)
    )
  ),
  fluidRow(
    column(6, plotOutput("hist")),
    column(6, plotOutput("qq"))
  )
)

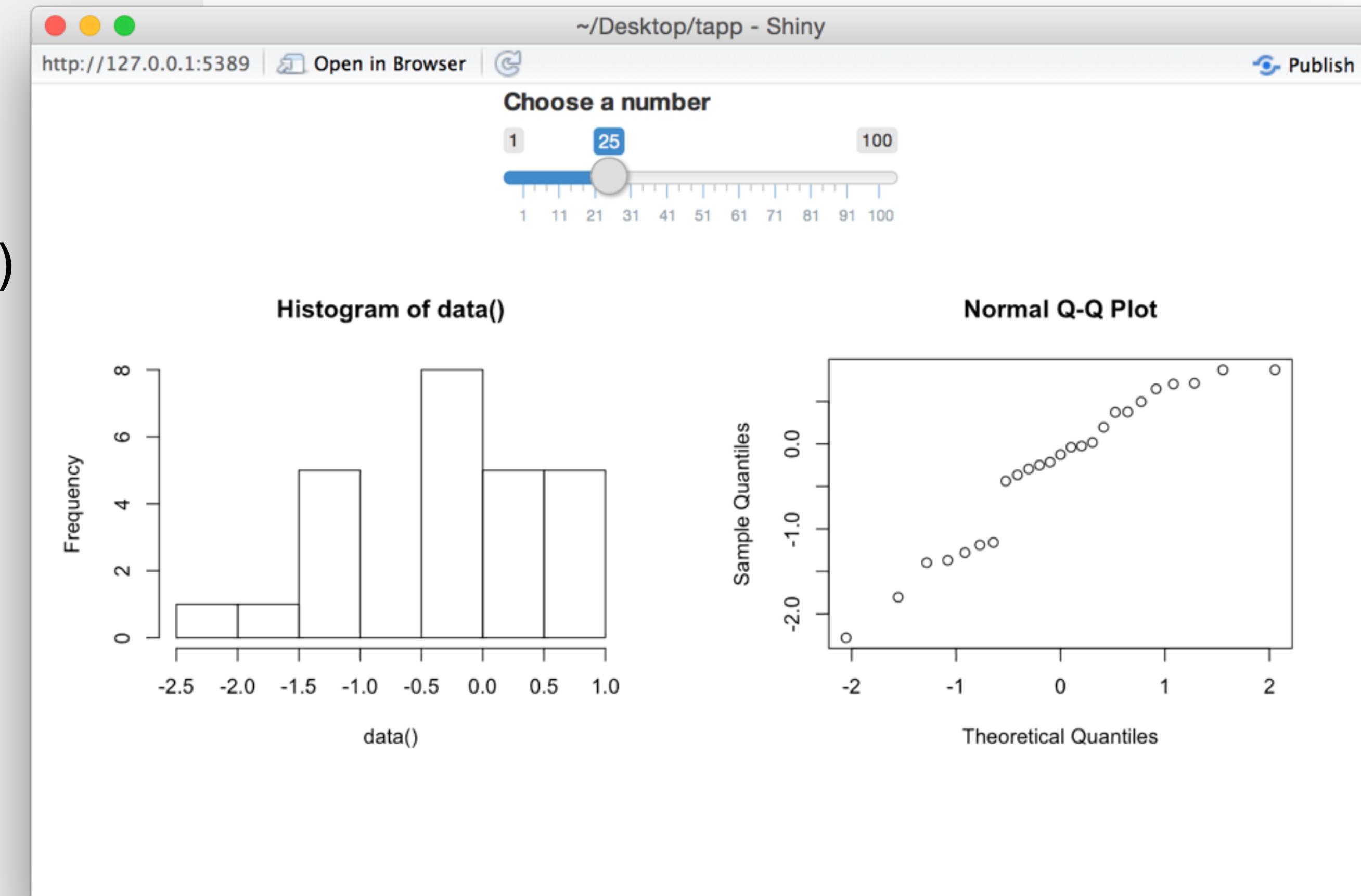
```



```

ui <- fluidPage(
  fluidRow(
    column(width = 4, offset = 4,
      sliderInput(inputId = "num",
                  label = "Choose a number",
                  val = 25, min = 1, max = 100)
    )
  ),
  fluidRow(
    column(6, plotOutput("hist")),
    column(6, plotOutput("qq"))
  )
)

```

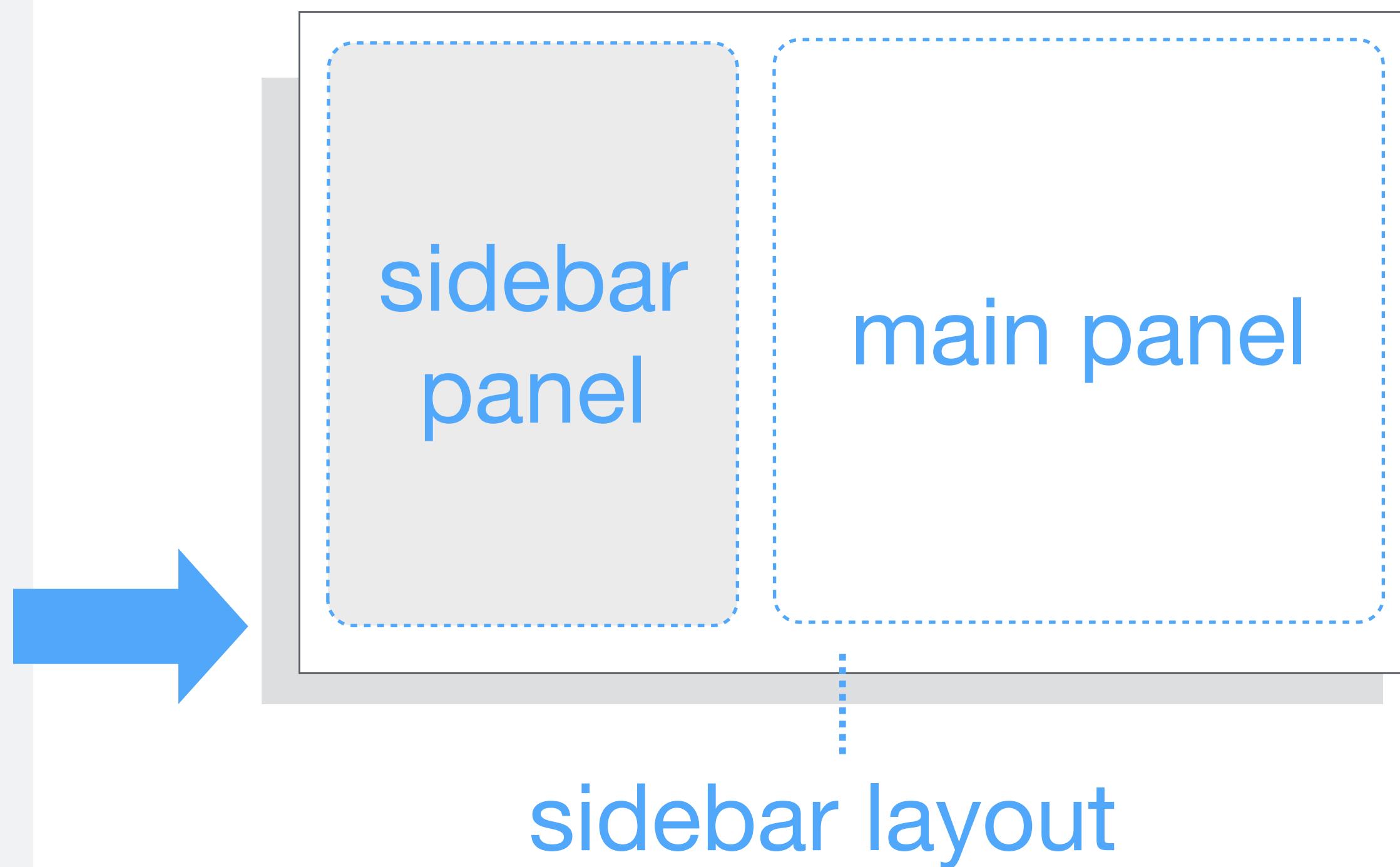


# Prepackaged Layouts

# pageWithSidebar()

The most simple way to layout a Shiny app. Use  
with `titlePanel()`, `sidebarPanel()` and `mainPanel()`

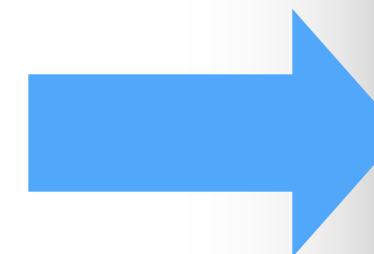
```
ui <- pageWithSidebar(  
  titlePanel(),  
  sidebarPanel(),  
  mainPanel()  
)
```



# navbarPage()

`navbarPage()` combines tabs into a single *page*.  
*navbarPage()* replaces *fluidPage()*.

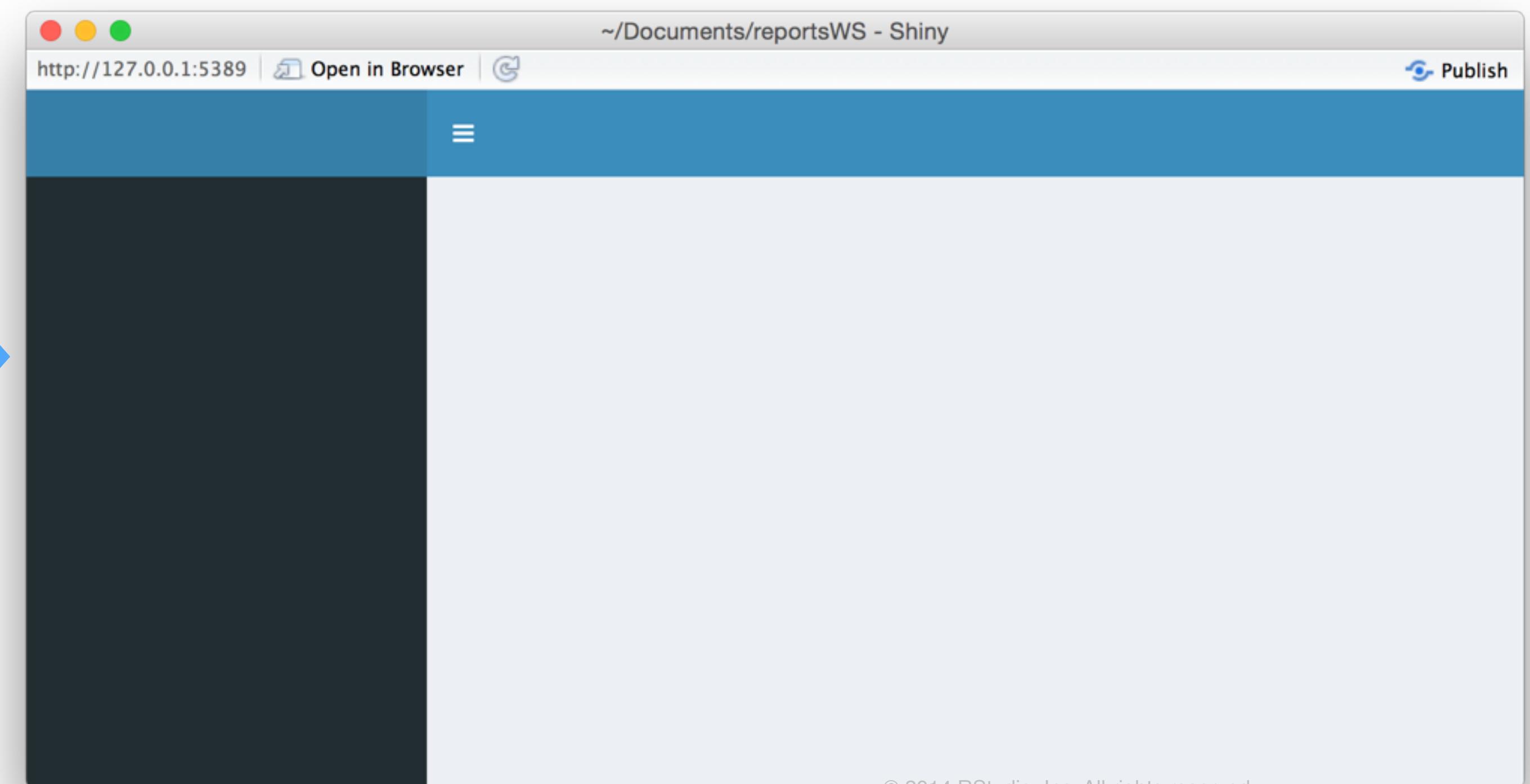
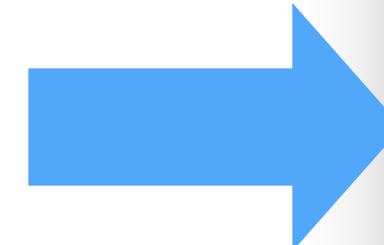
```
ui <- navbarPage("My Tabs",  
  tabPanel("tab 1", "stuff"),  
  tabPanel("tab 2", "stuff"),  
  tabPanel("tab 3", "stuff"))
```



# dashboardPage()

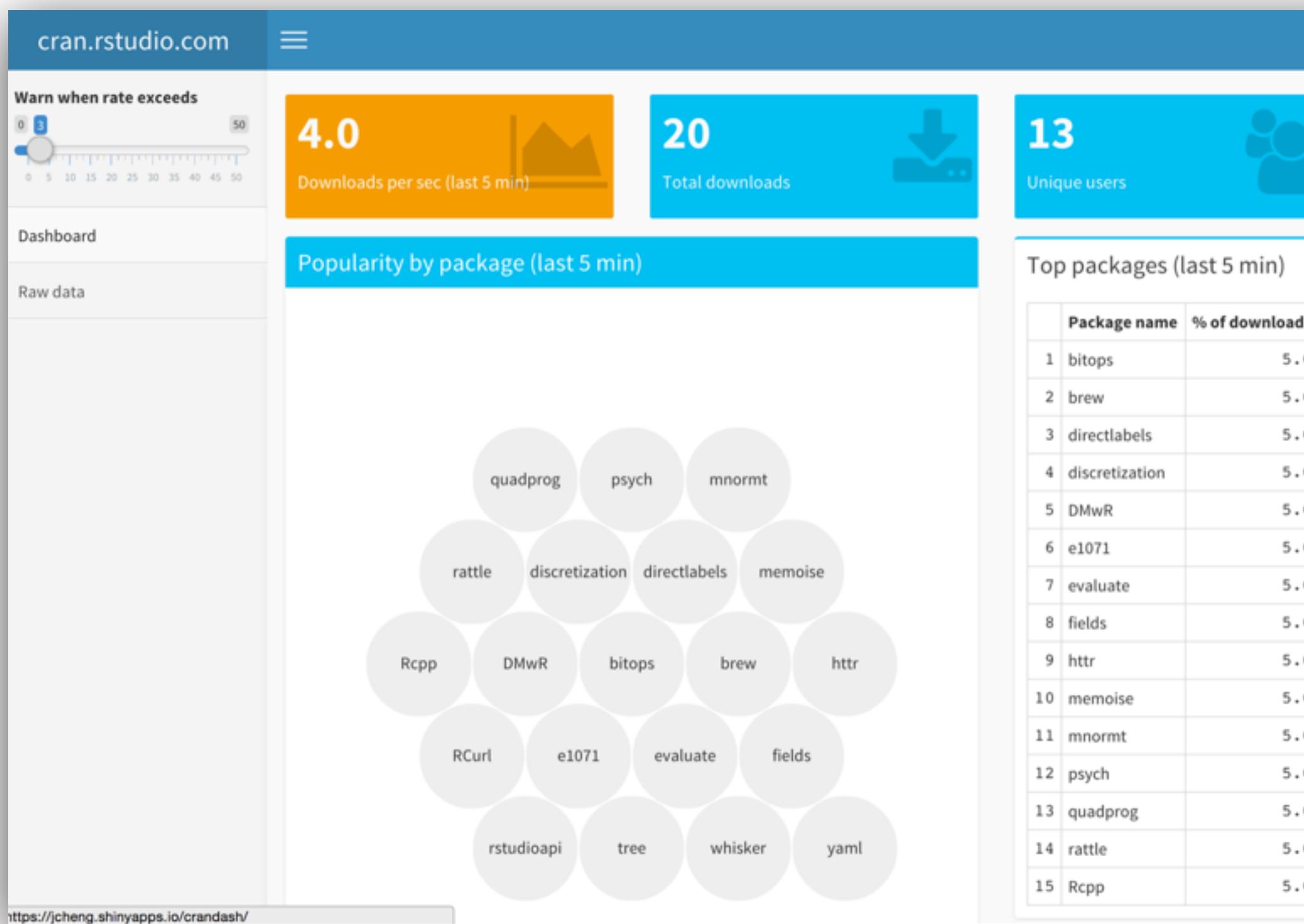
`dashboardPage()` comes in the `shinydashboard` package

```
library(shinydashboard)  
ui <- dashboardPage(  
  dashboardHeader(),  
  dashboardSidebar(),  
  dashboardBody())
```



# shinydashboard

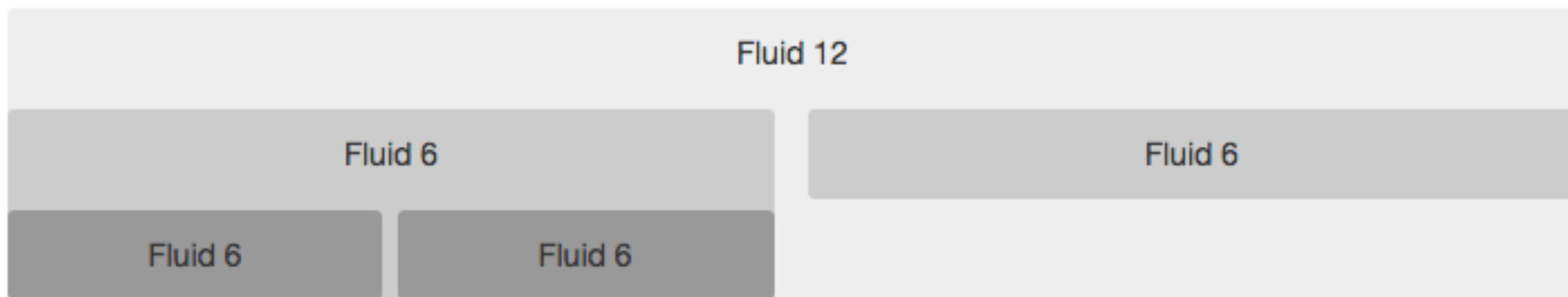
<http://rstudio.github.io/shinydashboard/>



A package of layout functions for building administrative dashboards with Shiny

# The Shiny Layout Guide

<http://shiny.rstudio.com/articles/layout-guide.html>



You can build sophisticated, customized layouts with Shiny's grid system.

**Custom**  
**HTML/CSS/JS**

# Customize your UI with HTML

<http://shiny.rstudio.com/articles/html-tags.html>

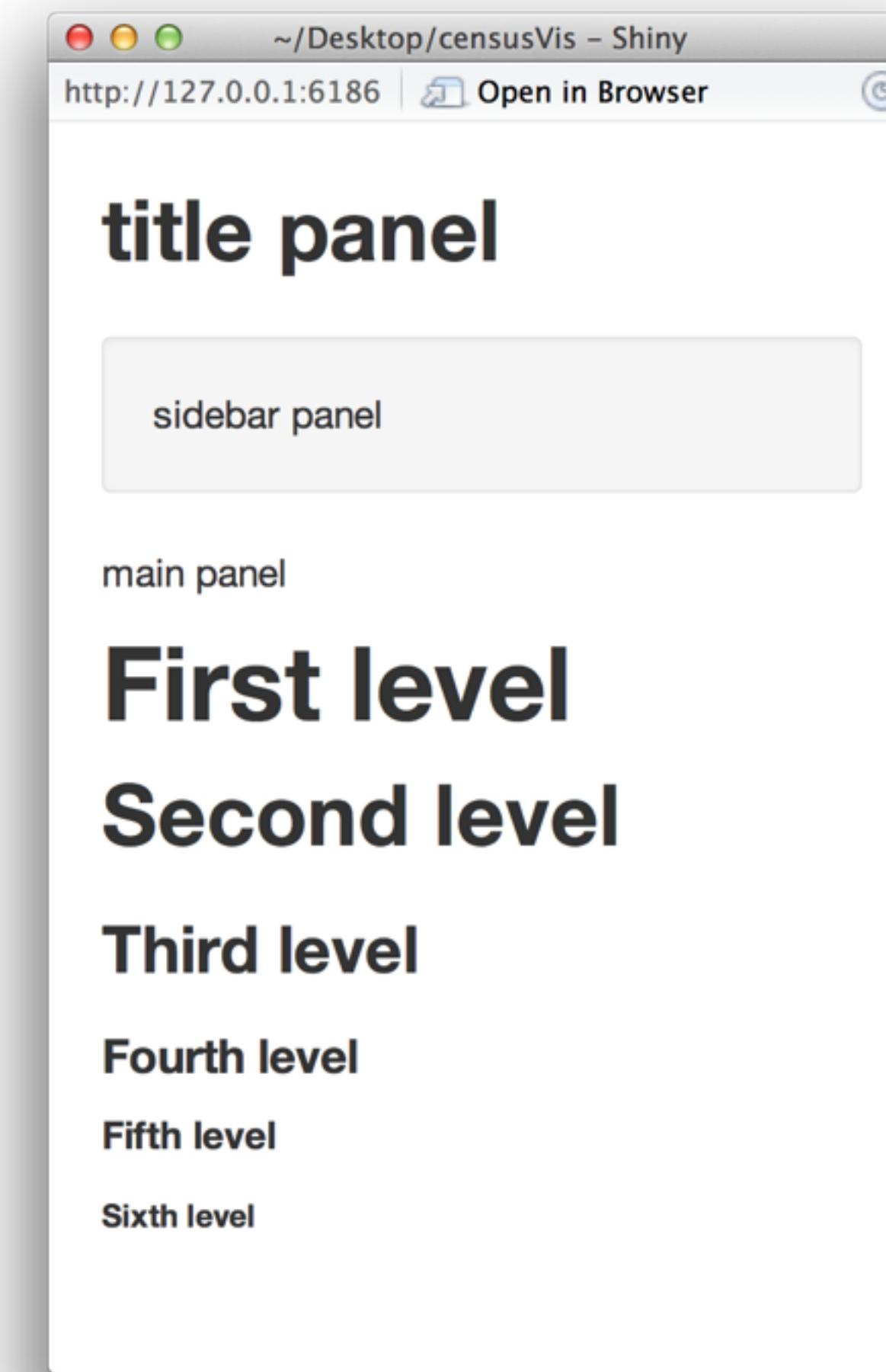
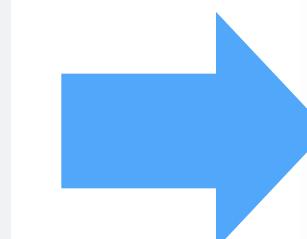
You can add any HTML element to your app with one of Shiny's `tags` function.

```
names(tags)
## [1] "a"                  "abbr"                "address"              "area"                 "article"
## [6] "aside"               "audio"                "b"                   "base"                 "bdi"
## [11] "bdo"                 "blockquote"           "body"                 "br"                   "button"
## [16] "canvas"               "caption"              "cite"                 "code"                 "col"
## [21] "colgroup"             "command"              "data"                 "datalist"              "dd"
## [26] "del"                  "details"              "dfn"                  "div"                  "dl"
## [31] "dt"                   "em"                   "embed"                "eventsouce"            "fieldset"
```

# h1() - h6()

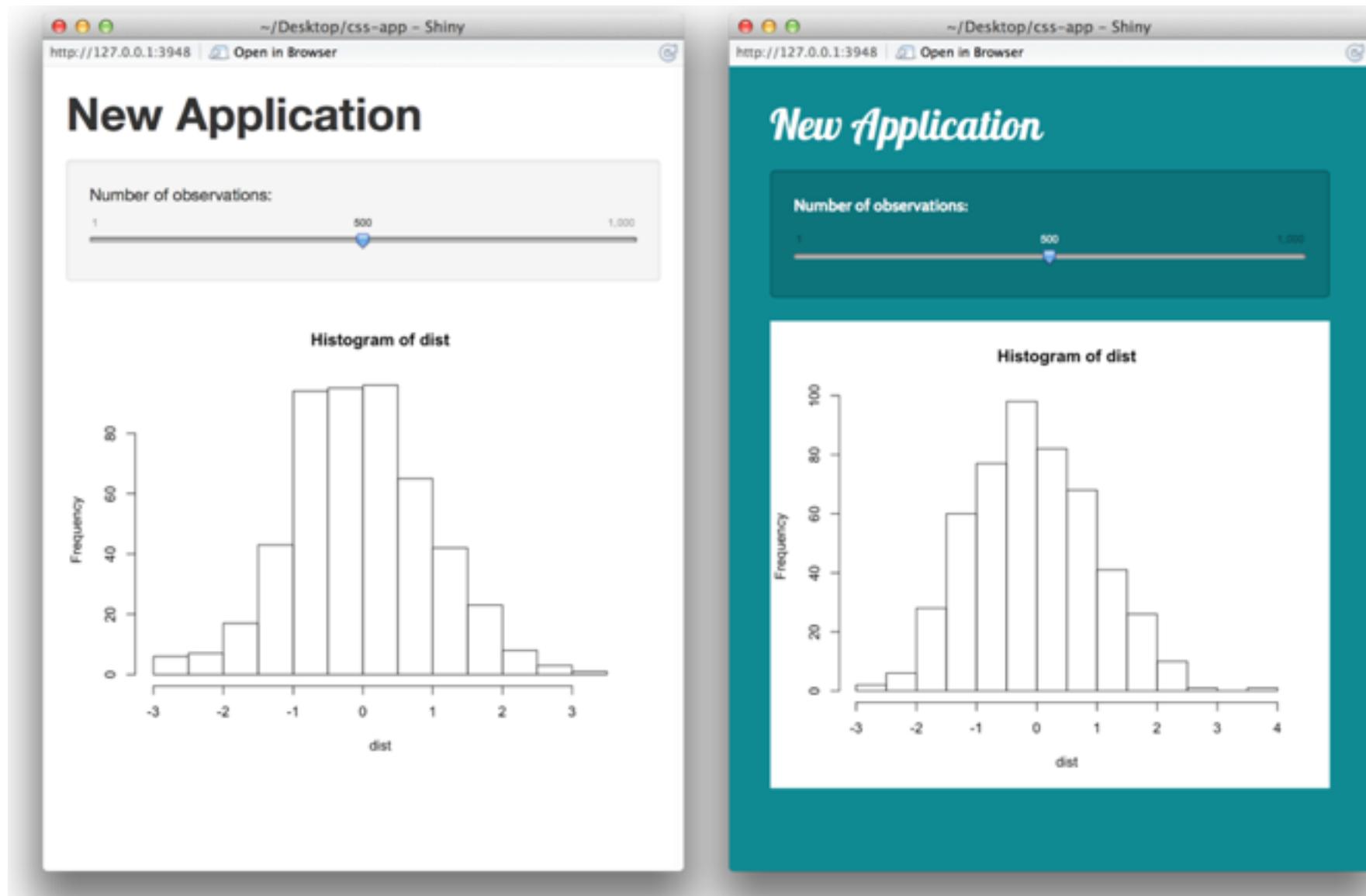
## Headers

```
fluidPage(  
  titlePanel("title panel"),  
  sidebarLayout(  
    sidebarPanel("sidebar panel"),  
    mainPanel("main panel",  
      h1("First level"),  
      h2("Second level"),  
      h3("Third level"),  
      h4("Fourth level"),  
      h5("Fifth level"),  
      h6("Sixth level"))  
)  
)
```



# Customize your apps with HTML, CSS, and Javascript

<http://shiny.rstudio.com/articles/css.html>



You can pair any app with whatever web technologies you wish. The above guide explains how to style your app with CSS.

# HTML UI

[shiny.rstudio.com/articles/html-ui.html](http://shiny.rstudio.com/articles/html-ui.html)

Build your entire UI from HTML

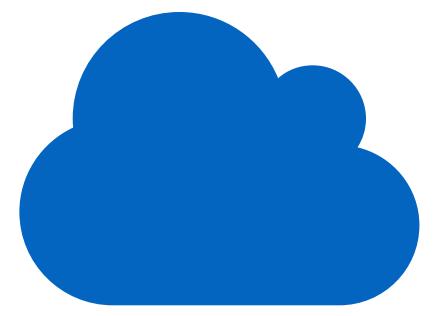
```
<html>

  <head>
    <script src="shared/jquery.js" type="text/javascript"></script>
    <script src="shared/shiny.js" type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="shared/shiny.css"/>
  </head>

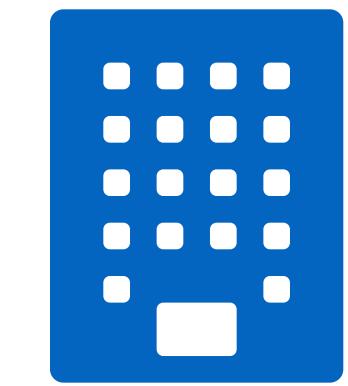
  <body>
    <h1>HTML UI</h1>

    <p>
      <label>Distribution type:</label><br />
```

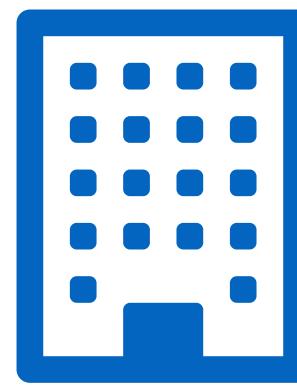
# Sharing Apps



shinyapps.io



Shiny Server



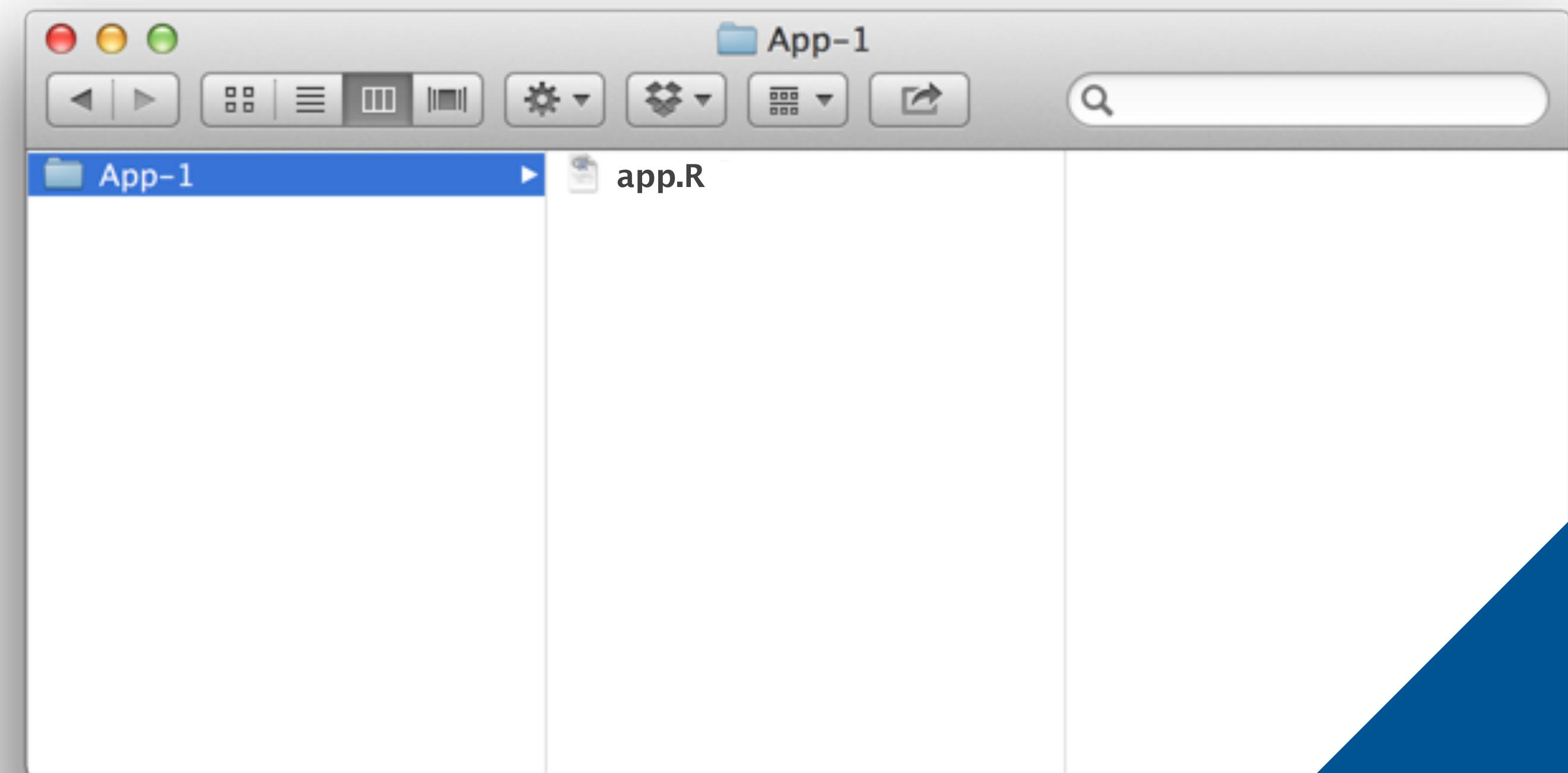
Shiny Server Pro

# App Directories

# Structure of a Shiny web 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)

# runApp

You can launch the app with runApp (your computer will build a local web site that hosts the app).

```
runApp(~Documents/App-1")
```

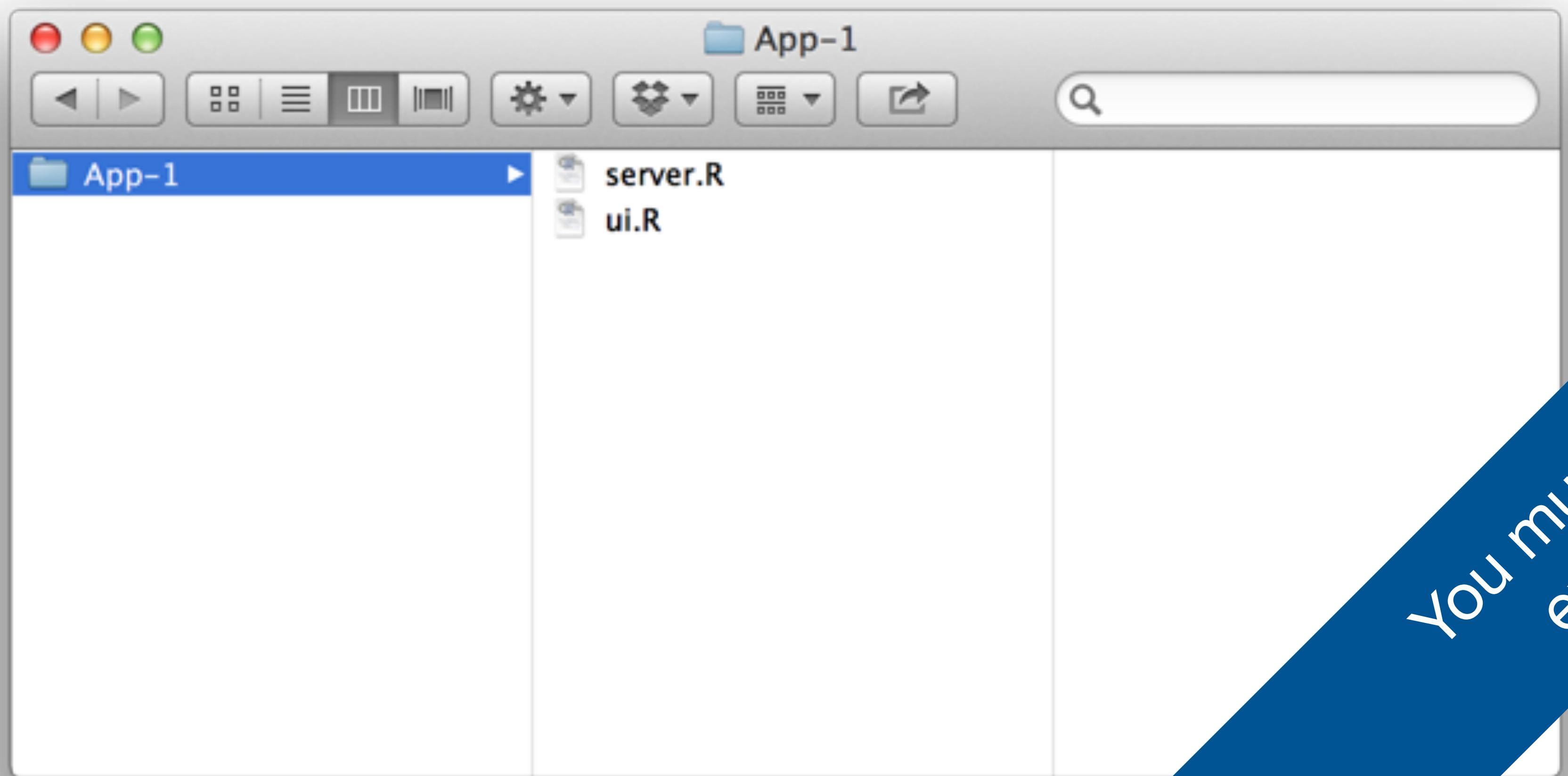
File path to app directory.

R will append the file path to the working directory,  
if path does not begin at the home directory

# Two file apps

One directory with two files:

- `server.R`
- `ui.R`

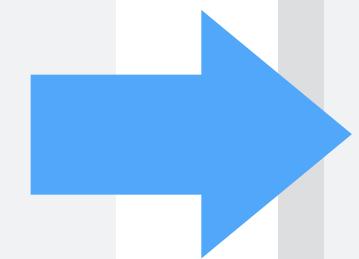


You must use these  
exact names

```
library(shiny)
server <- function(input, output) {
  output$distPlot <- renderPlot({
    x      <- faithful[, 2]
    bins   <- seq(min(x), max(x), len = input$bins + 1)
    hist(x, breaks = bins, col = 'darkgray')
  })
}

ui <- fluidPage(
  titlePanel("Old Faithful Geyser Data"),
  sidebarLayout(
    sidebarPanel(
      sliderInput("bins", "Number of bins:",
                 min = 1, max = 50, value = 30)),
    mainPanel(plotOutput("distPlot"))
  )
)

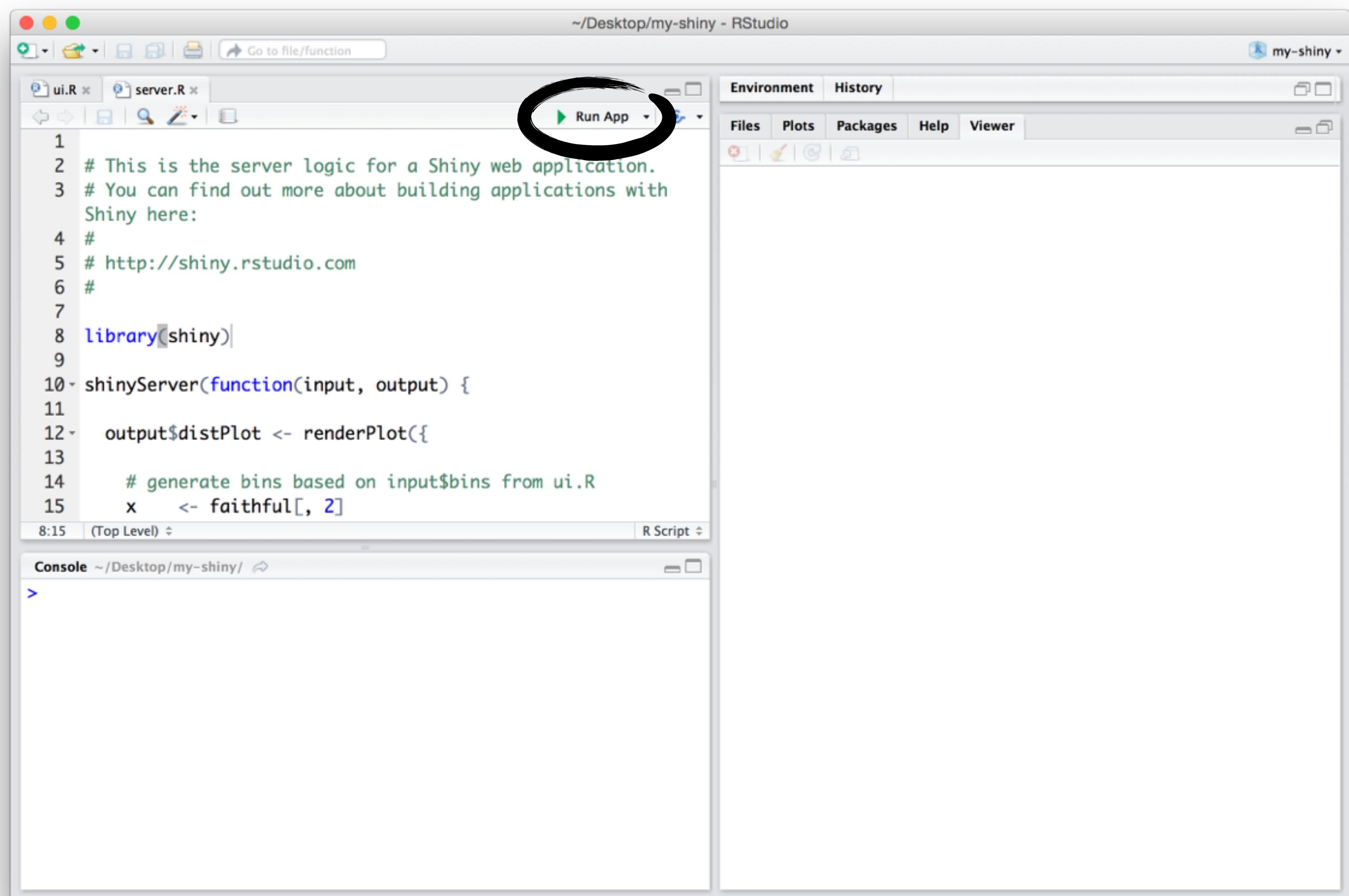
shinyApp(server = server, ui = ui)
```



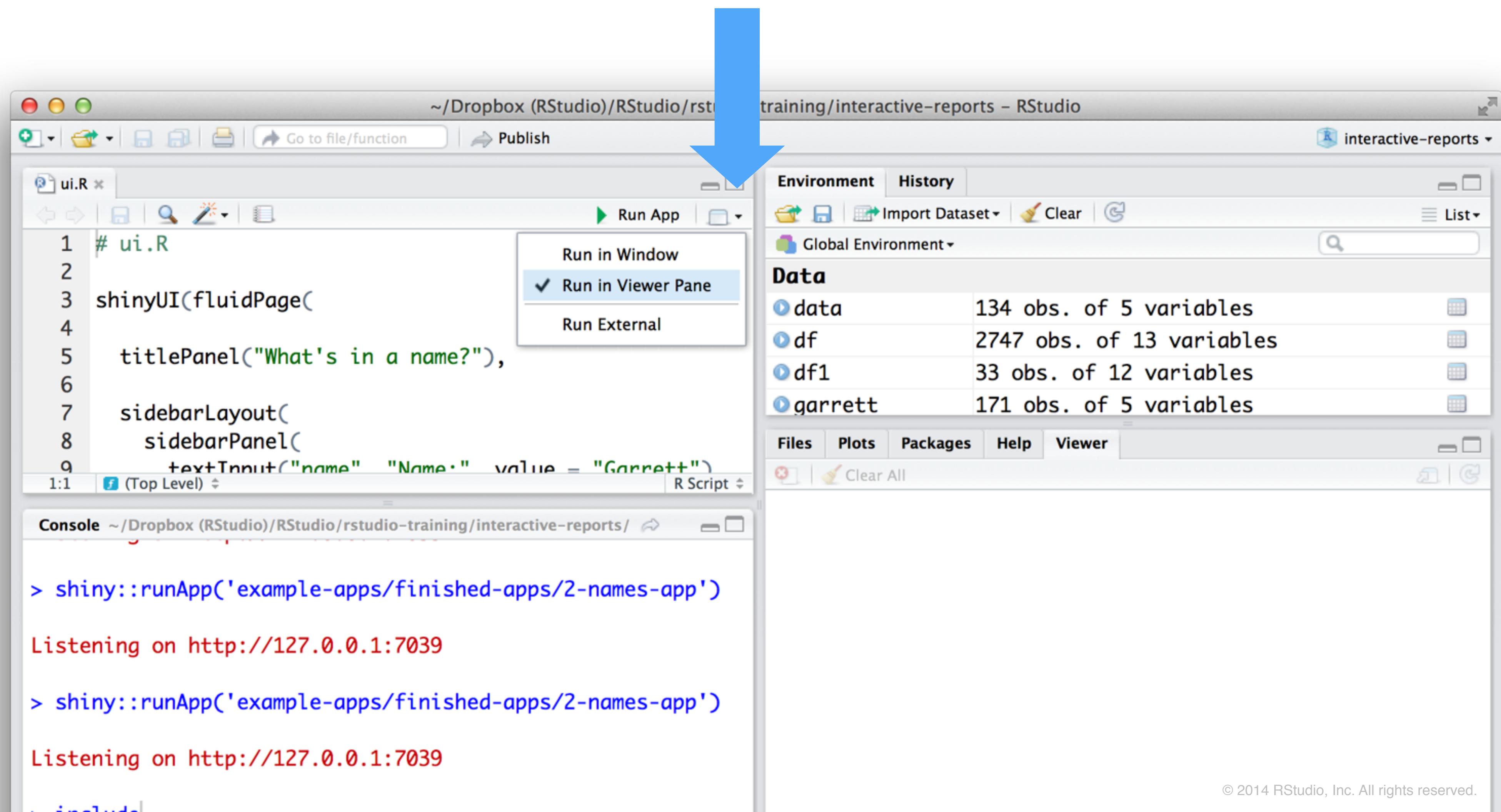
```
# server.R
library(shiny)
function(input, output) {
  output$distPlot <- renderPlot({
    x      <- faithful[, 2]
    bins   <- seq(min(x), max(x), len = input$bins + 1)
    hist(x, breaks = bins, col = 'darkgray')
  })
}

# ui.R
fluidPage(
  titlePanel("Old Faithful Geyser Data"),
  sidebarLayout(
    sidebarPanel(
      sliderInput("bins", "Number of bins:",
                 min = 1, max = 50, value = 30)),
    mainPanel(plotOutput("distPlot"))
  )
)
```

# Launch an app



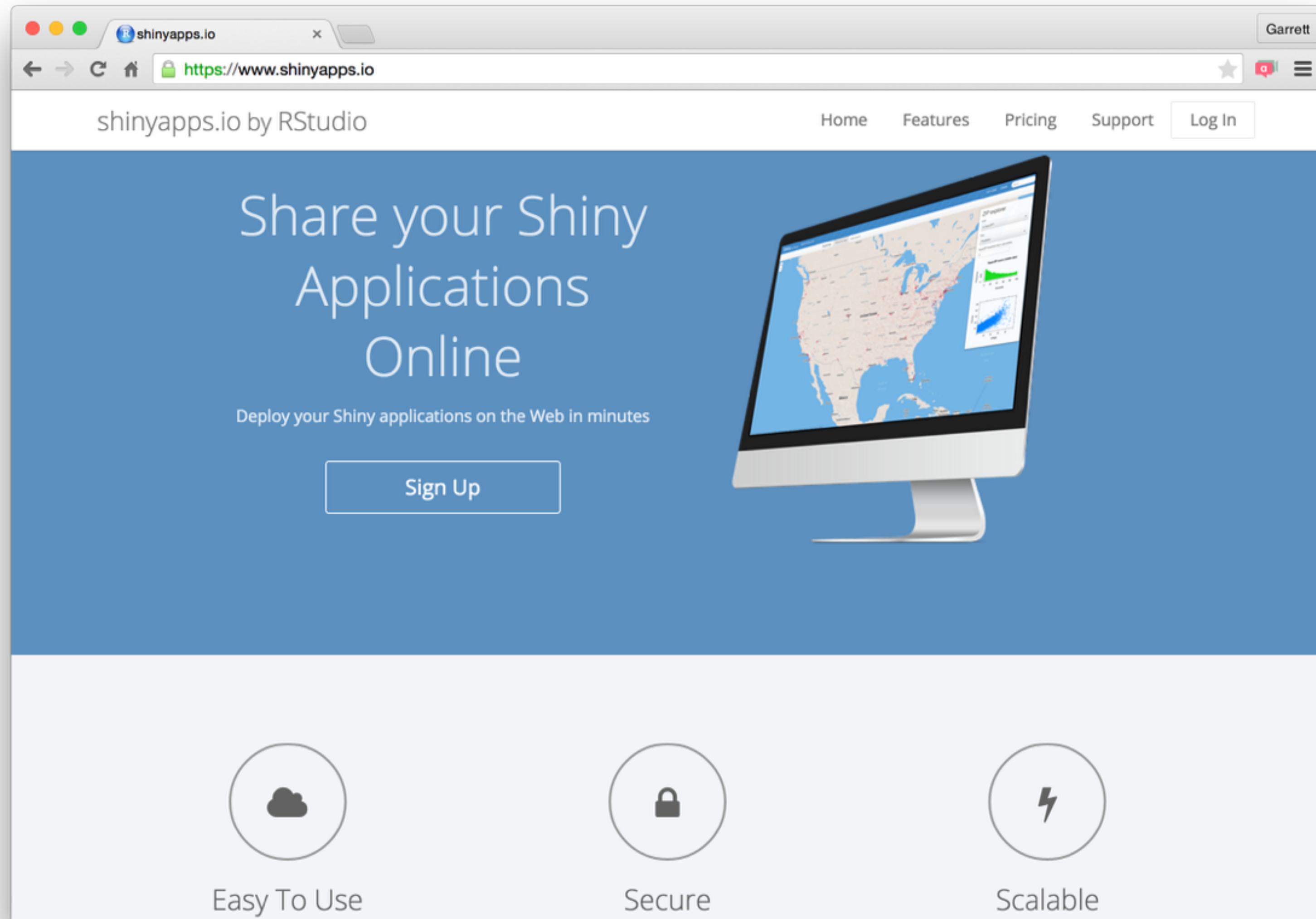
# Display options



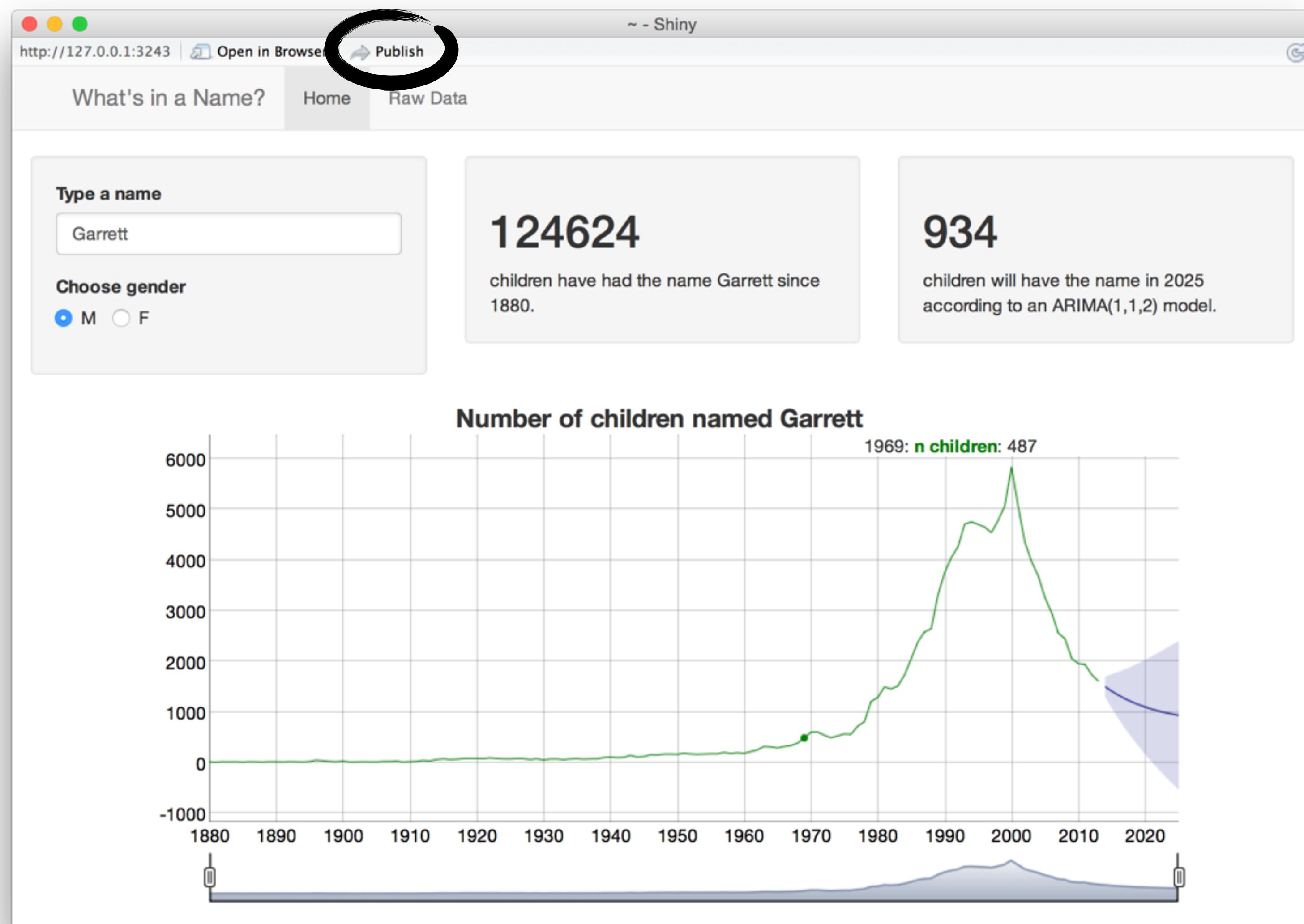
shinyapps.io

# Hassle-free cloud hosting for Shiny

[shinyapps.io](https://shinyapps.io)

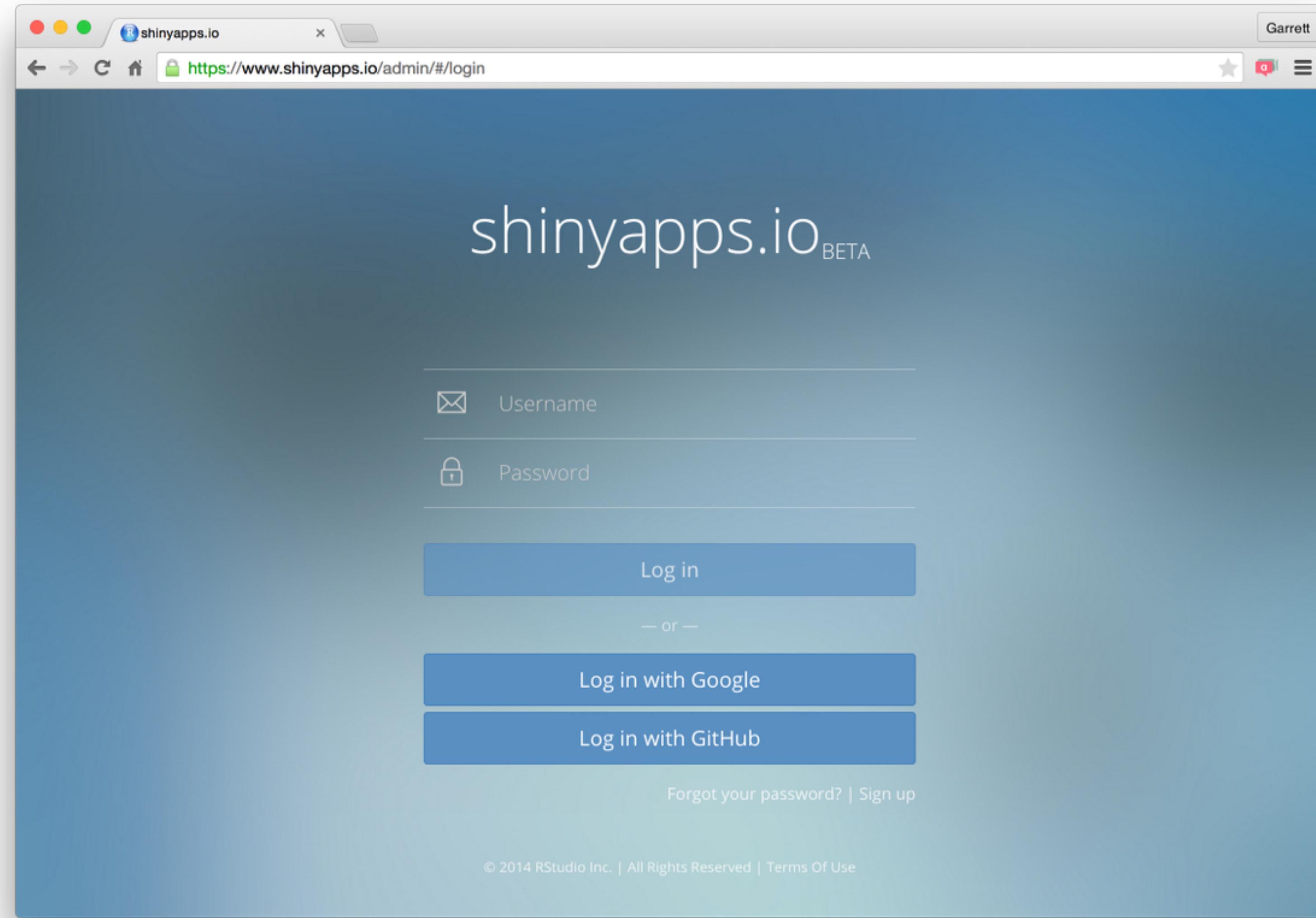


# Deploy to web with one click



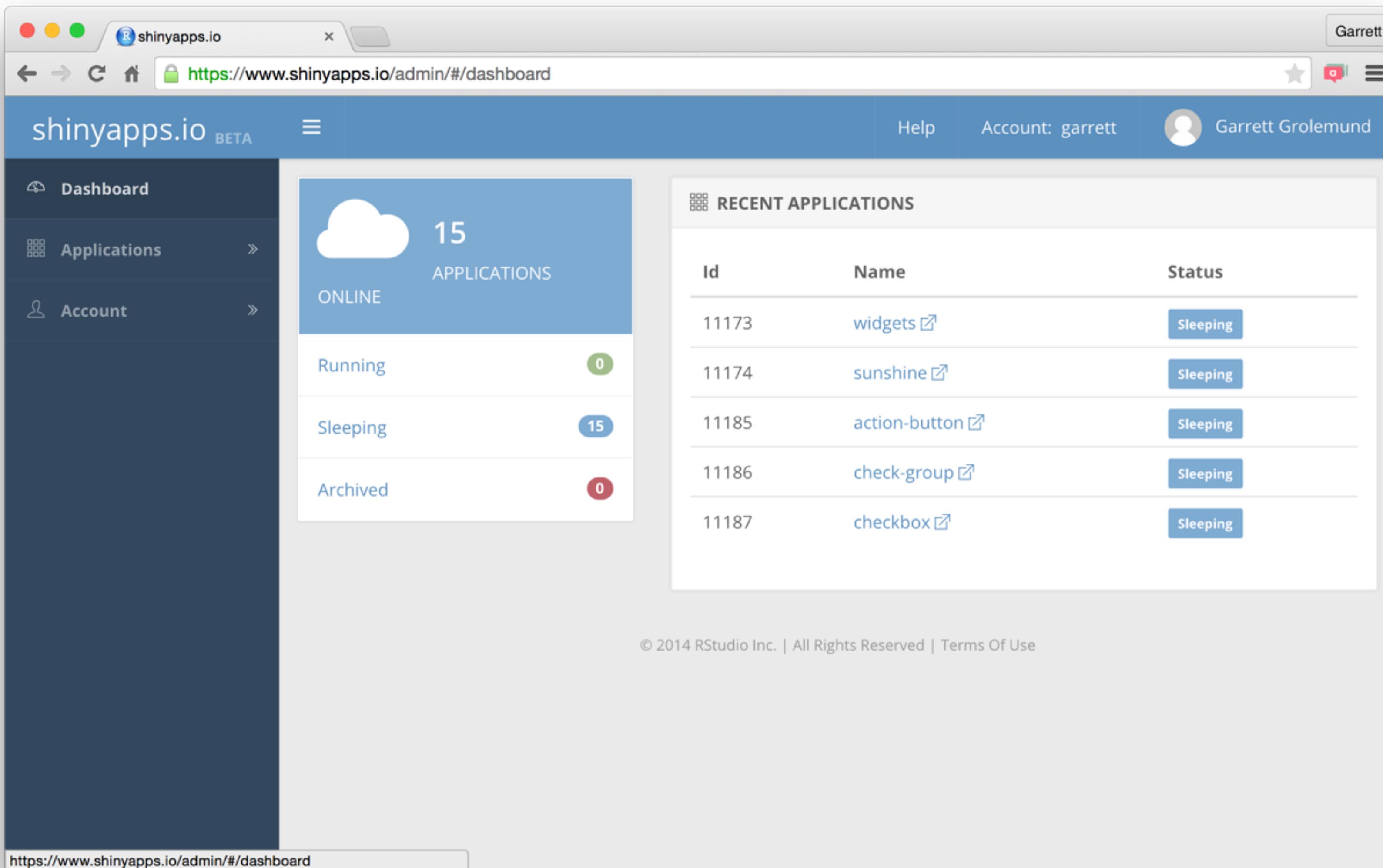
# Hassle-free cloud hosting for Shiny

[shinyapps.io](https://shinyapps.io)



# Hassle-free cloud hosting for Shiny

[shinyapps.io](https://shinyapps.io)



# Hassle-free cloud hosting for Shiny

[shinyapps.io](https://shinyapps.io)

The screenshot shows the shinyapps.io administration interface for application ID 16670, titled "082-WORD-CLOUD".

**OVERVIEW:**

<b>Id</b>	16670
<b>Name</b>	082-word-cloud
<b>URL</b>	<a href="http://gallery.shinyapps.io/082-word-cloud">http://gallery.shinyapps.io/082-word-cloud</a>
<b>Status</b>	Running
<b>Size</b>	medium
<b>Deployed</b>	Dec 4, 2014
<b>Updated</b>	Feb 10, 2015
<b>Created</b>	Jul 28, 2014

**INSTANCES:**

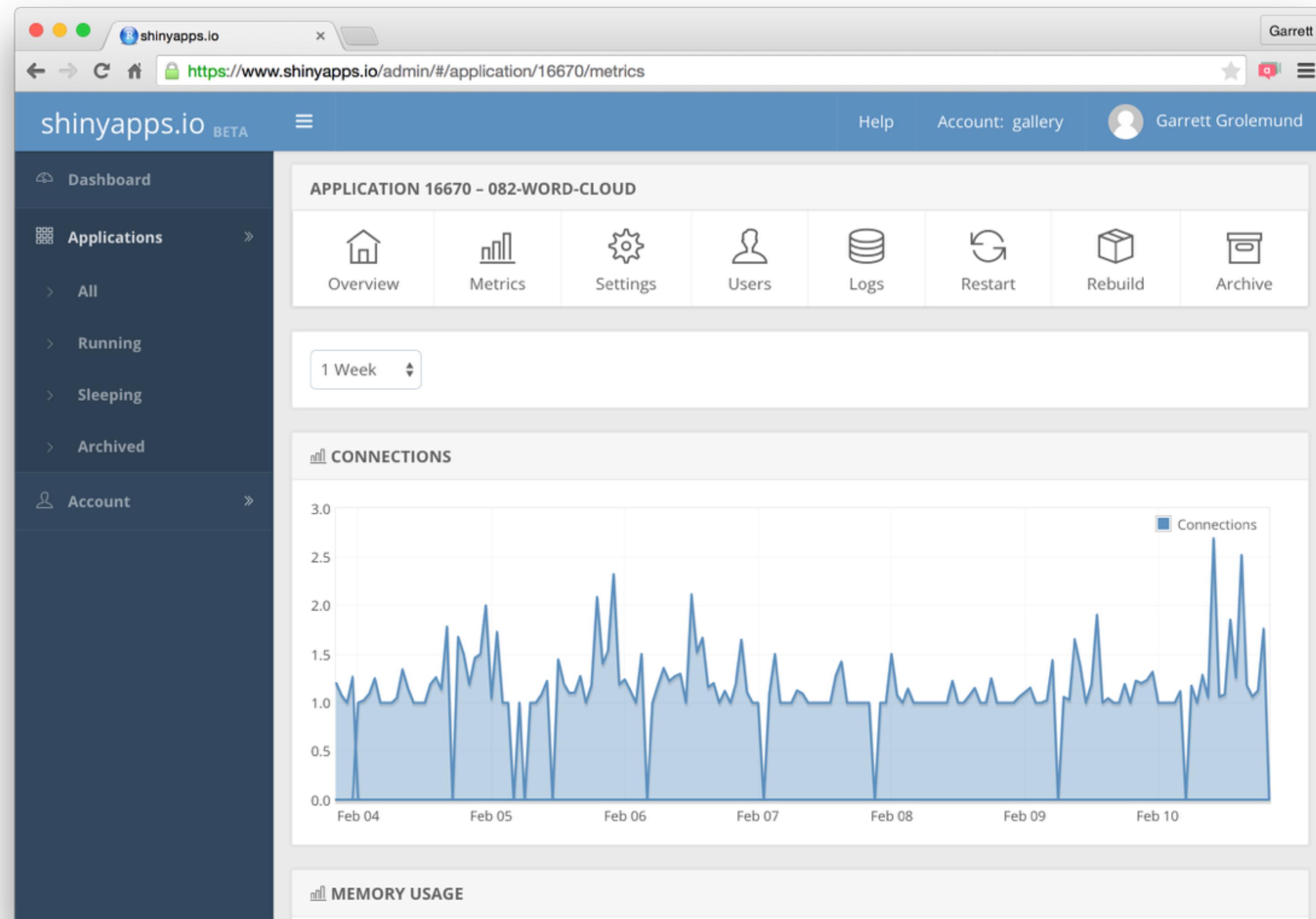
<b>Id:</b> 93138	<input type="checkbox"/>	<input type="button" value="Delete"/>
------------------	--------------------------	---------------------------------------

**APPLICATION USAGE:**

Total: 127.09 hours

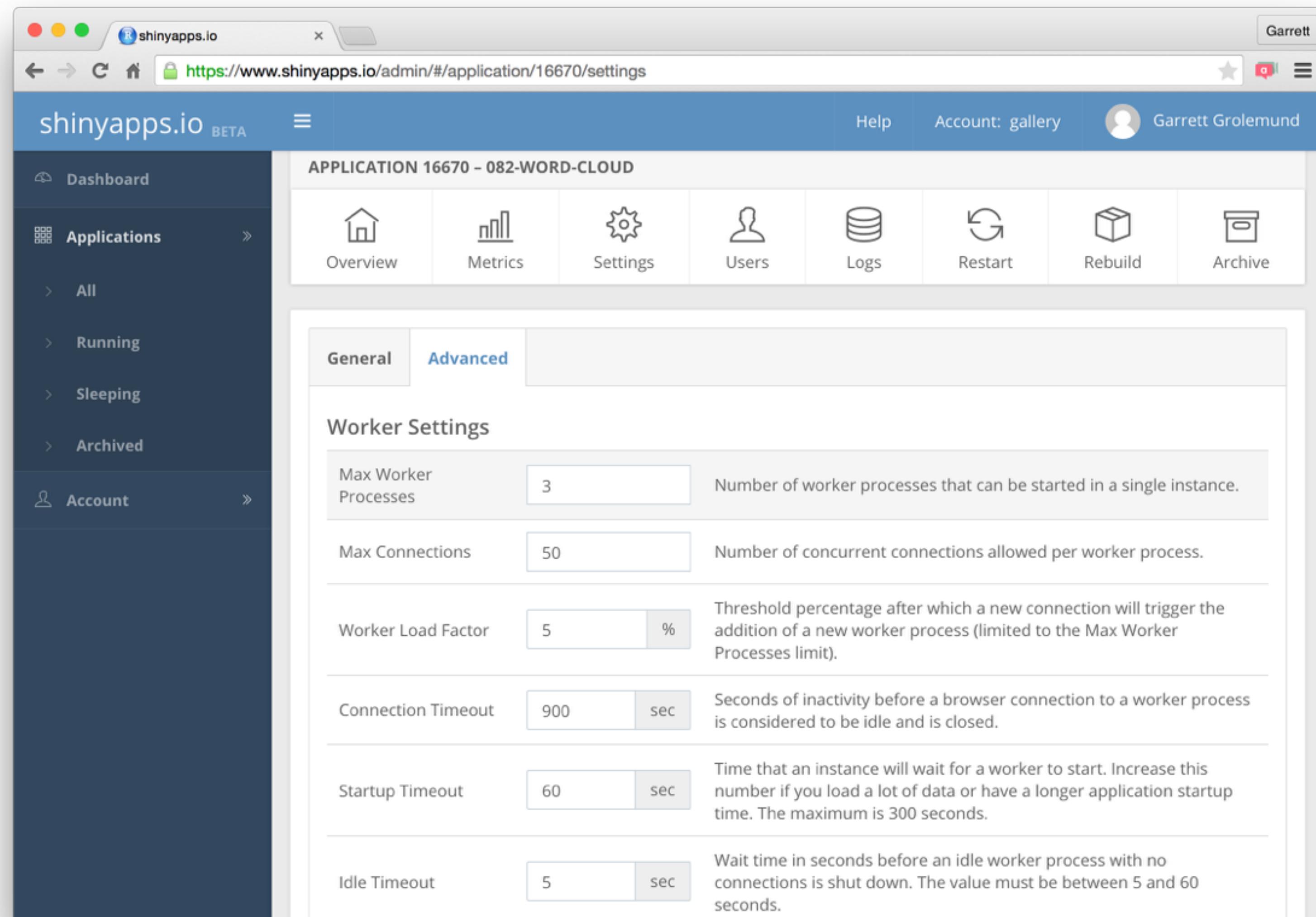
# Hassle-free cloud hosting for Shiny

[shinyapps.io](https://shinyapps.io)



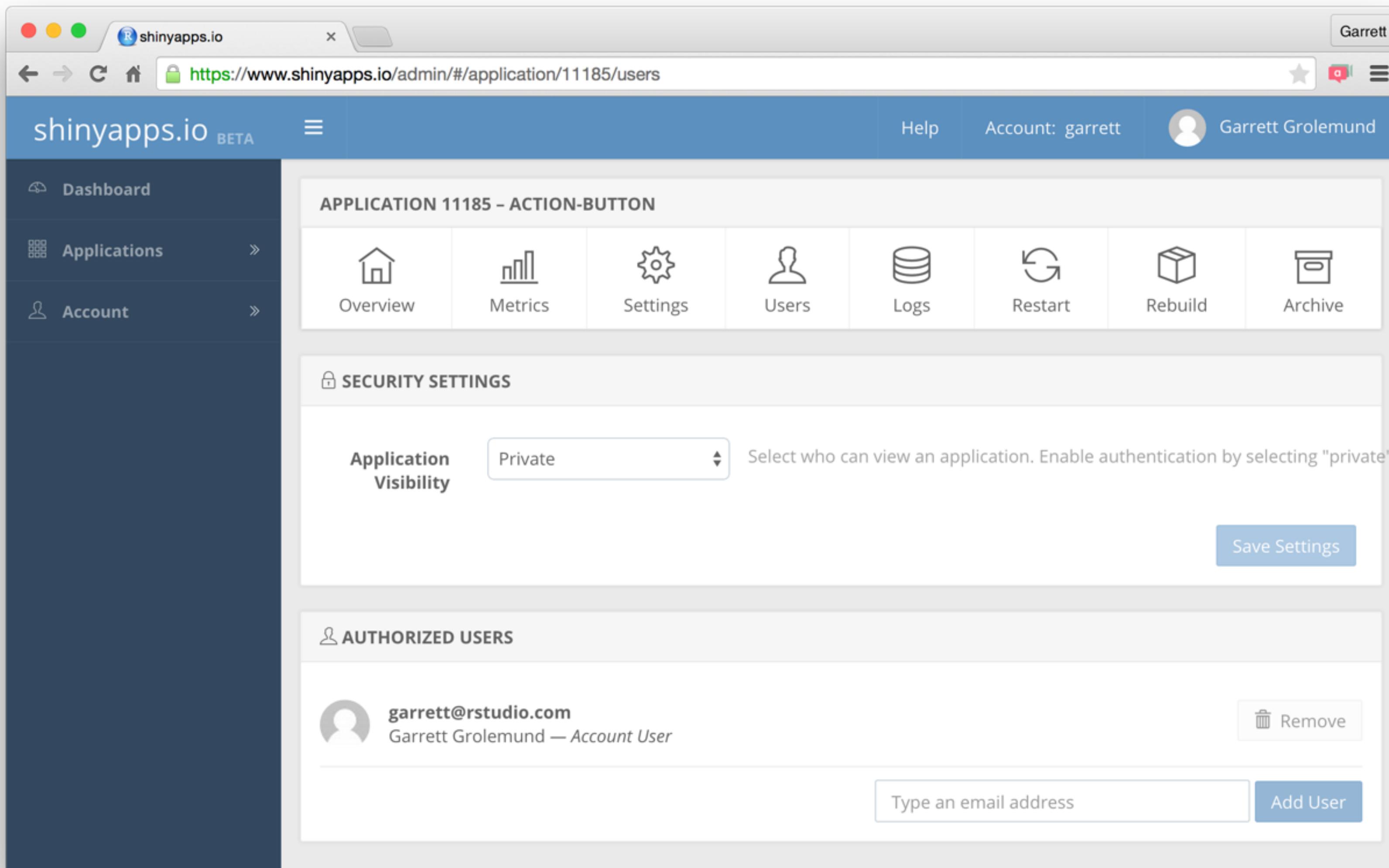
# Hassle-free cloud hosting for Shiny

[shinyapps.io](https://shinyapps.io)



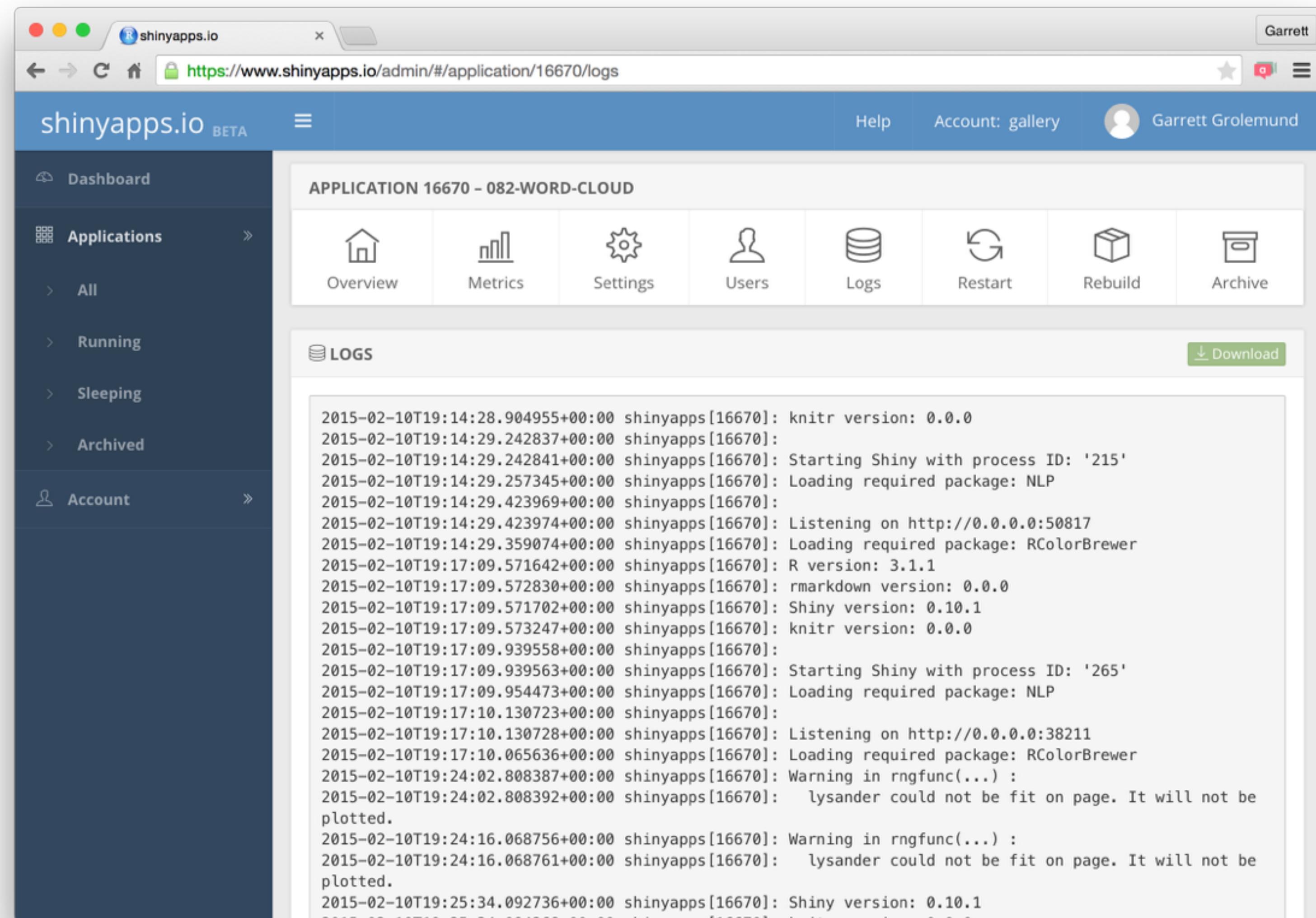
# Hassle-free cloud hosting for Shiny

[shinyapps.io](http://shinyapps.io)



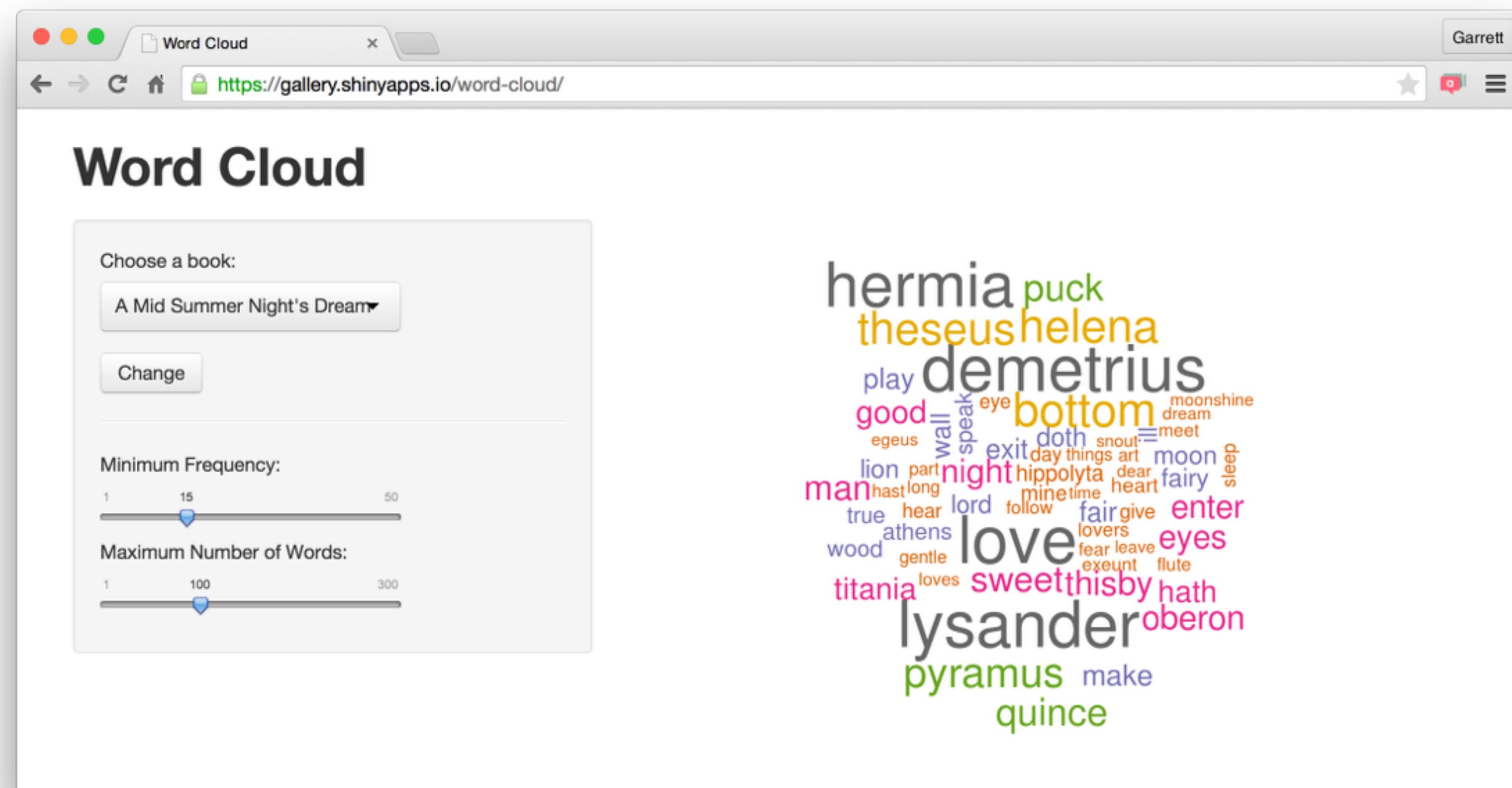
# Hassle-free cloud hosting for Shiny

[shinyapps.io](http://shinyapps.io)



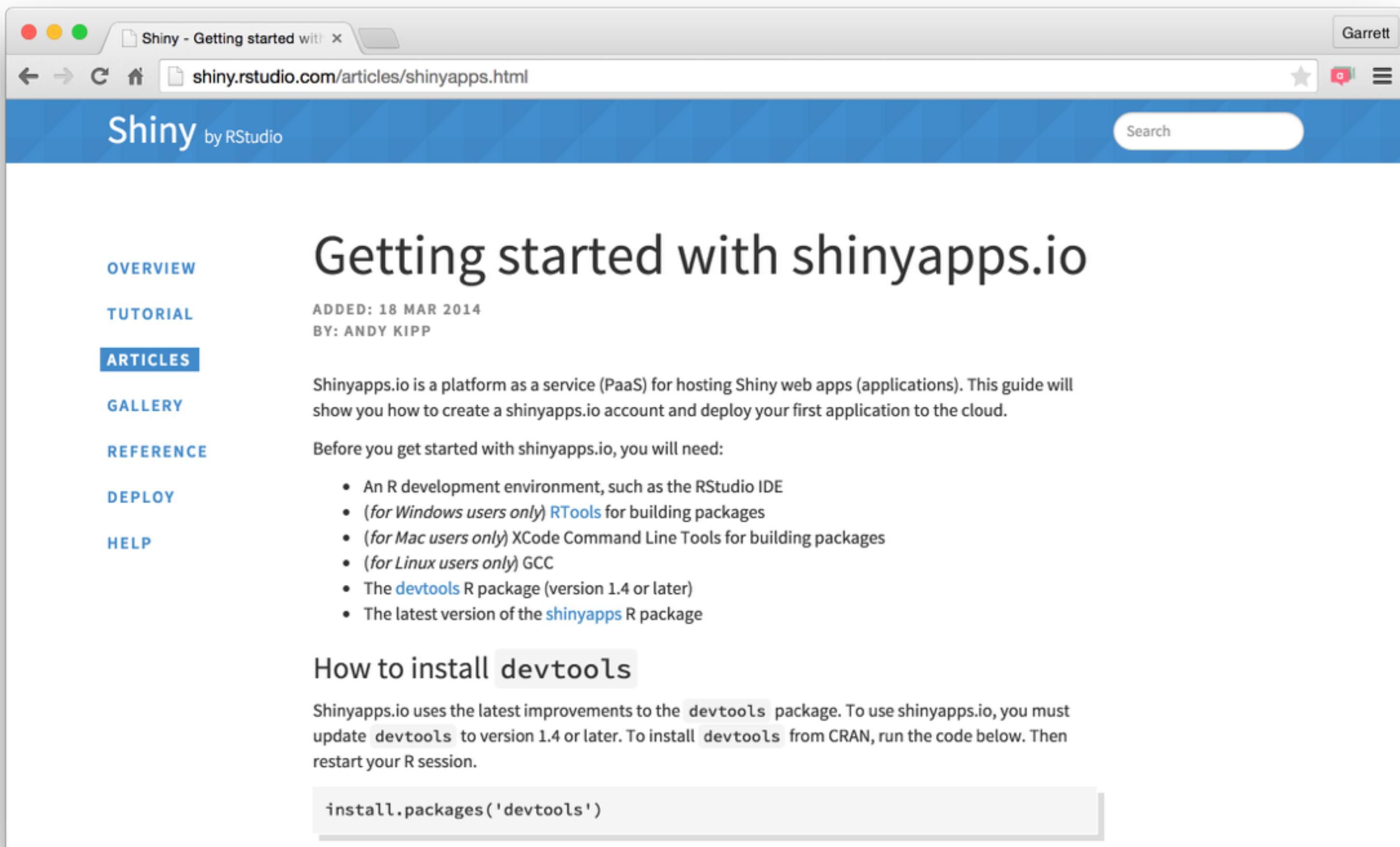
# Each app hosted at its own URL

[gallery.shinyapps.io/word-cloud](https://gallery.shinyapps.io/word-cloud)



# Getting started guide

[shiny.rstudio.com/articles/shinyapps.html](http://shiny.rstudio.com/articles/shinyapps.html)



The screenshot shows a web browser window with the title bar "Shiny - Getting started with" and the address bar containing the URL "shiny.rstudio.com/articles/shinyapps.html". The browser interface includes standard controls like back, forward, and search, along with a user profile "Garrett". The main content area is titled "Getting started with shinyapps.io". On the left, there's a sidebar with navigation links: OVERVIEW, TUTORIAL, ARTICLES (which is selected), GALLERY, REFERENCE, DEPLOY, and HELP. The main content area has a sub-header "ADDED: 18 MAR 2014 BY: ANDY KIPP". It describes Shinyapps.io as a PaaS for hosting Shiny web apps and provides instructions for getting started, listing requirements like RStudio IDE, RTools, XCode Command Line Tools, GCC, devtools package (version 1.4 or later), and the shinyapps package. It also includes a section on how to install devtools and a code snippet for doing so.

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')
```

**Build a  
server**

# Shiny Server

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

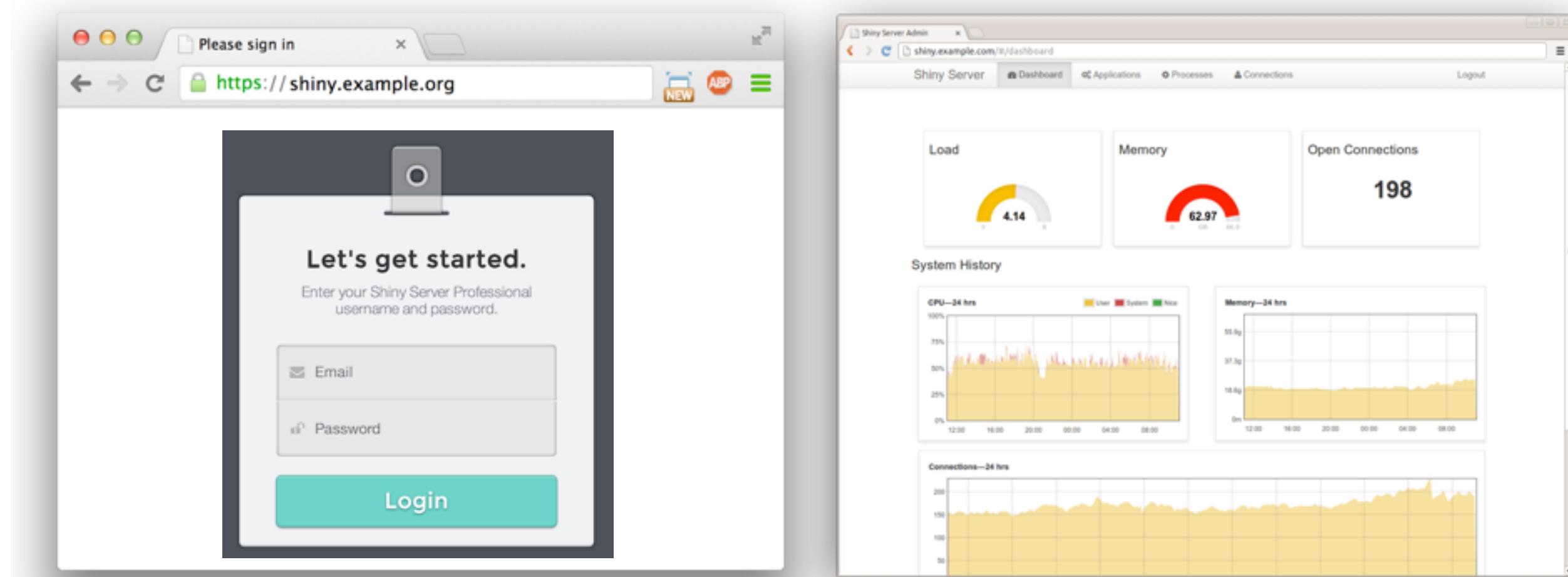
- Each app is hosted at its own URL
- Can deploy to internet, or within a controlled environment
- Starts app when user visits, closes app when user leaves
- Runs on a linux server

<http://shiny.rstudio.com/articles/shiny-server.html>

# Shiny Server Pro

[www.rstudio.com/products/shiny-server-pro/](http://www.rstudio.com/products/shiny-server-pro/)

- **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



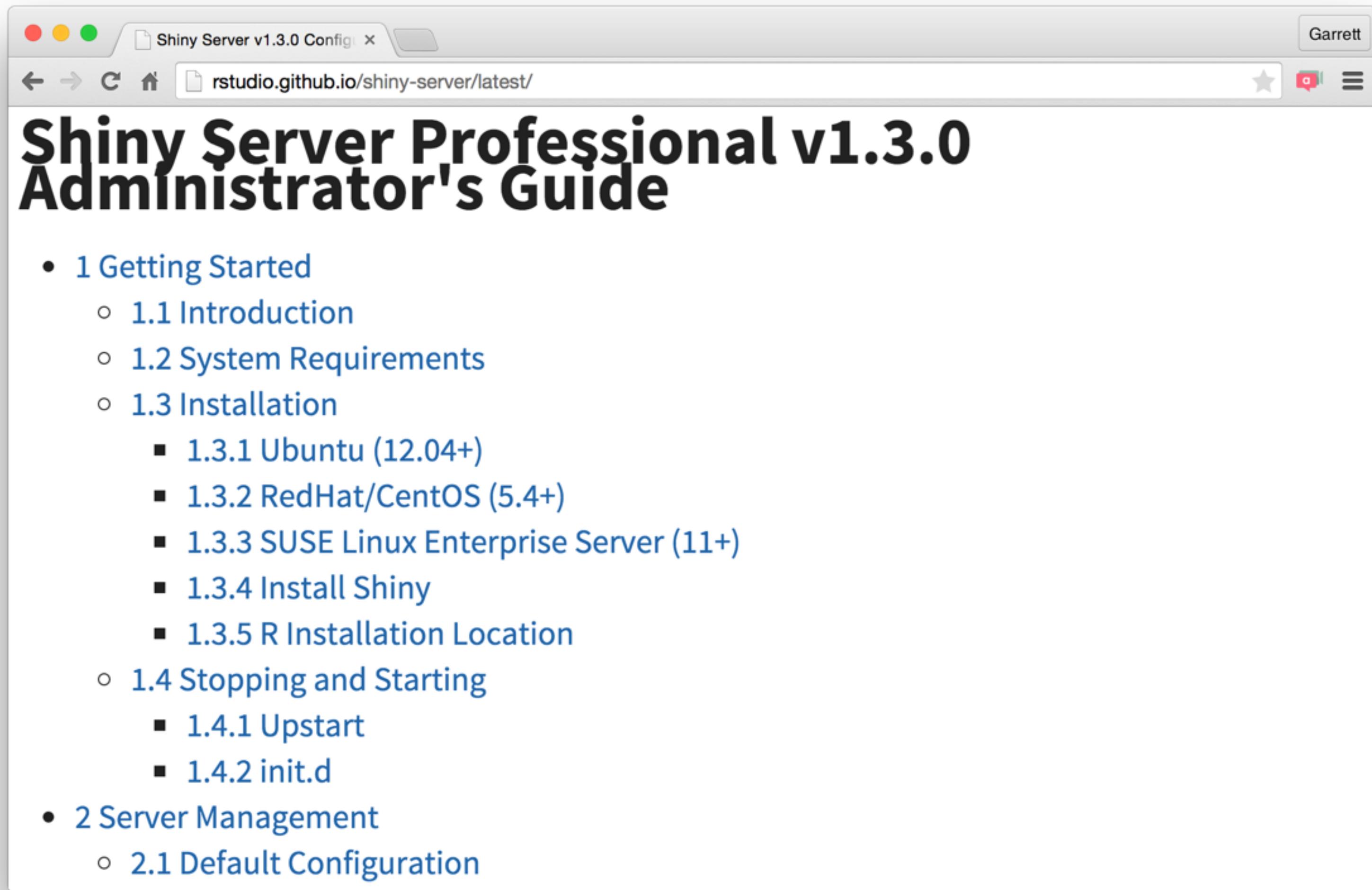
# <http://rstudio.com/shiny/server>

		Open Source Edition	Professional Edition
<b>General</b>	Deploy Shiny applications to the Internet	•	•
	Move computation close to the data	•	•
	Host multiple applications on a single server	•	•
	Deploy Shiny applications behind firewalls	•	•
	Custom page templates	•	•
<b>Security and authentication</b>	Password file authentication	•	
	LDAP and Active Directory authentication	•	
	Google authentication (OAuth2)	•	
	PAM authentication & sessions	•	
	Group based authorization	•	
	SSL support	•	
<b>Tuning and scaling</b>	Scale applications across multiple processes	•	
	View and manage active sessions	•	
	Allocate resources on a per application basis	•	
	Define application concurrency limits	•	
<b>Server monitoring</b>	System performance and resource metrics 	•	
	Per application performance and resource metrics 	•	
	Application usage metrics	•	
	Customizable health check end point	•	

\* For volume discounts, OEMs, or additional capacity for larger audiences please email us at [sales@rstudio.com](mailto:sales@rstudio.com).

# Shiny Server (Pro) Admin Guide

[rstudio.github.io/shiny-server/latest/](https://rstudio.github.io/shiny-server/latest/)



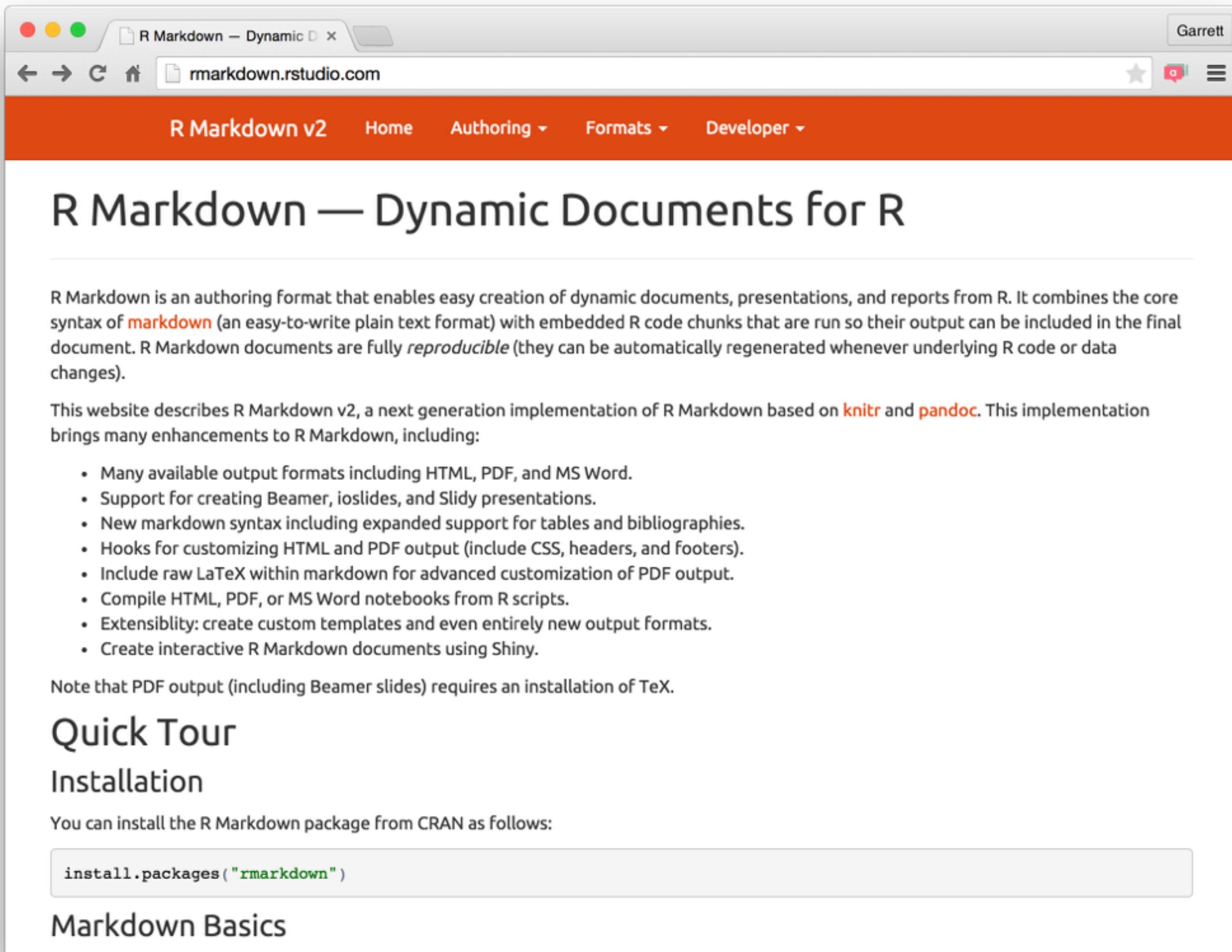
The screenshot shows a web browser window titled "Shiny Server v1.3.0 Config". The address bar contains the URL "rstudio.github.io/shiny-server/latest/". The main content area displays the title "Shiny Server Professional v1.3.0 Administrator's Guide" in large, bold, black font. Below the title is a navigation menu with the following items:

- [1 Getting Started
  - \[1.1 Introduction\]\(#\)
  - \[1.2 System Requirements\]\(#\)
  - \[1.3 Installation
    - \\[1.3.1 Ubuntu \\\(12.04+\\\)\\]\\(#\\)
    - \\[1.3.2 RedHat/CentOS \\\(5.4+\\\)\\]\\(#\\)
    - \\[1.3.3 SUSE Linux Enterprise Server \\\(11+\\\)\\]\\(#\\)
    - \\[1.3.4 Install Shiny\\]\\(#\\)
    - \\[1.3.5 R Installation Location\\]\\(#\\)\]\(#\)
  - \[1.4 Stopping and Starting
    - \\[1.4.1 Upstart\\]\\(#\\)
    - \\[1.4.2 init.d\\]\\(#\\)\]\(#\)](#)
- [2 Server Management
  - \[2.1 Default Configuration\]\(#\)](#)

**How to  
learn more**

# The R Markdown Development Center

[rmarkdown.rstudio.com](http://rmarkdown.rstudio.com)



A screenshot of a web browser displaying the R Markdown v2 website. The browser window has a title bar showing 'R Markdown — Dynamic D' and the address bar showing 'rmarkdown.rstudio.com'. The page itself has a header with tabs for 'R Markdown v2', 'Home', 'Authoring', 'Formats', and 'Developer'. The main content area features a large heading 'R Markdown — Dynamic Documents for R'. Below this, there is a paragraph about what R Markdown is, followed by a section about 'Enhancements to R Markdown' with a bulleted list of features. Further down, there are sections for 'Quick Tour' and 'Installation', and a note about installing the package from CRAN with a code snippet. At the bottom, there is a 'Markdown Basics' section.

R Markdown is an authoring format that enables easy creation of dynamic documents, presentations, and reports from R. It combines the core syntax of [markdown](#) (an easy-to-write plain text format) with embedded R code chunks that are run so their output can be included in the final document. R Markdown documents are fully *reproducible* (they can be automatically regenerated whenever underlying R code or data changes).

This website describes R Markdown v2, a next generation implementation of R Markdown based on [knitr](#) and [pandoc](#). This implementation brings many enhancements to R Markdown, including:

- Many available output formats including HTML, PDF, and MS Word.
- Support for creating Beamer, ioslides, and Slidy presentations.
- New markdown syntax including expanded support for tables and bibliographies.
- Hooks for customizing HTML and PDF output (include CSS, headers, and footers).
- Include raw LaTeX within markdown for advanced customization of PDF output.
- Compile HTML, PDF, or MS Word notebooks from R scripts.
- Extensibility: create custom templates and even entirely new output formats.
- Create interactive R Markdown documents using Shiny.

Note that PDF output (including Beamer slides) requires an installation of TeX.

## Quick Tour

## Installation

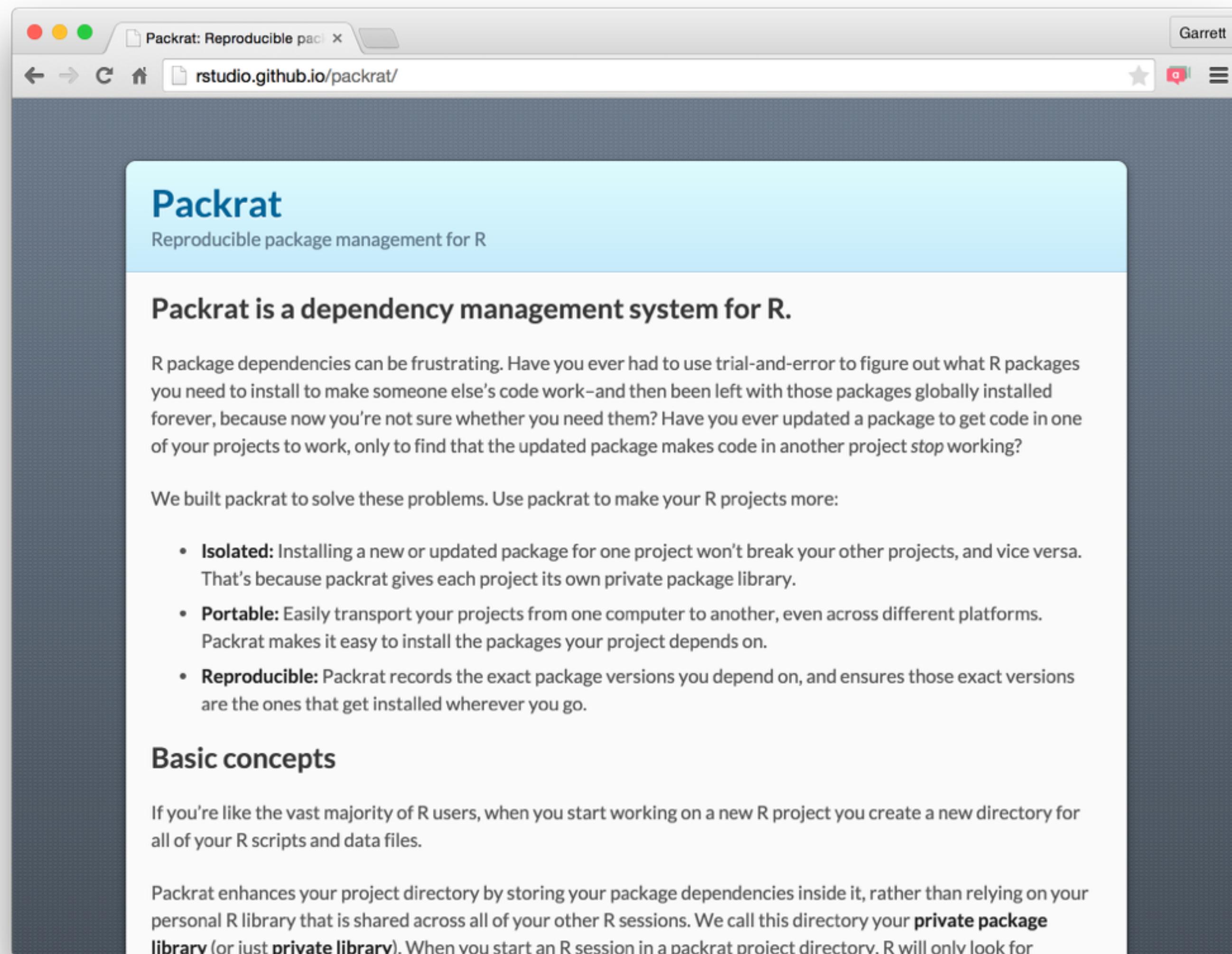
You can install the R Markdown package from CRAN as follows:

```
install.packages("rmarkdown")
```

## Markdown Basics

# The Packrat Development Center

[rstudio.github.io/packrat/](http://rstudio.github.io/packrat/)



A screenshot of a web browser window displaying the Packrat Development Center homepage. The browser's title bar reads "Packrat: Reproducible pac" and the address bar shows "rstudio.github.io/packrat/". The page itself has a light blue header with the word "Packrat" and the subtitle "Reproducible package management for R". Below this, a main heading states "Packrat is a dependency management system for R." followed by a paragraph of text. A bulleted list of features is provided, and a section titled "Basic concepts" with explanatory text follows.

Packrat

Reproducible package management for R

**Packrat is a dependency management system for R.**

R package dependencies can be frustrating. Have you ever had to use trial-and-error to figure out what R packages you need to install to make someone else's code work—and then been left with those packages globally installed forever, because now you're not sure whether you need them? Have you ever updated a package to get code in one of your projects to work, only to find that the updated package makes code in another project stop working?

We built packrat to solve these problems. Use packrat to make your R projects more:

- **Isolated:** Installing a new or updated package for one project won't break your other projects, and vice versa. That's because packrat gives each project its own private package library.
- **Portable:** Easily transport your projects from one computer to another, even across different platforms. Packrat makes it easy to install the packages your project depends on.
- **Reproducible:** Packrat records the exact package versions you depend on, and ensures those exact versions are the ones that get installed wherever you go.

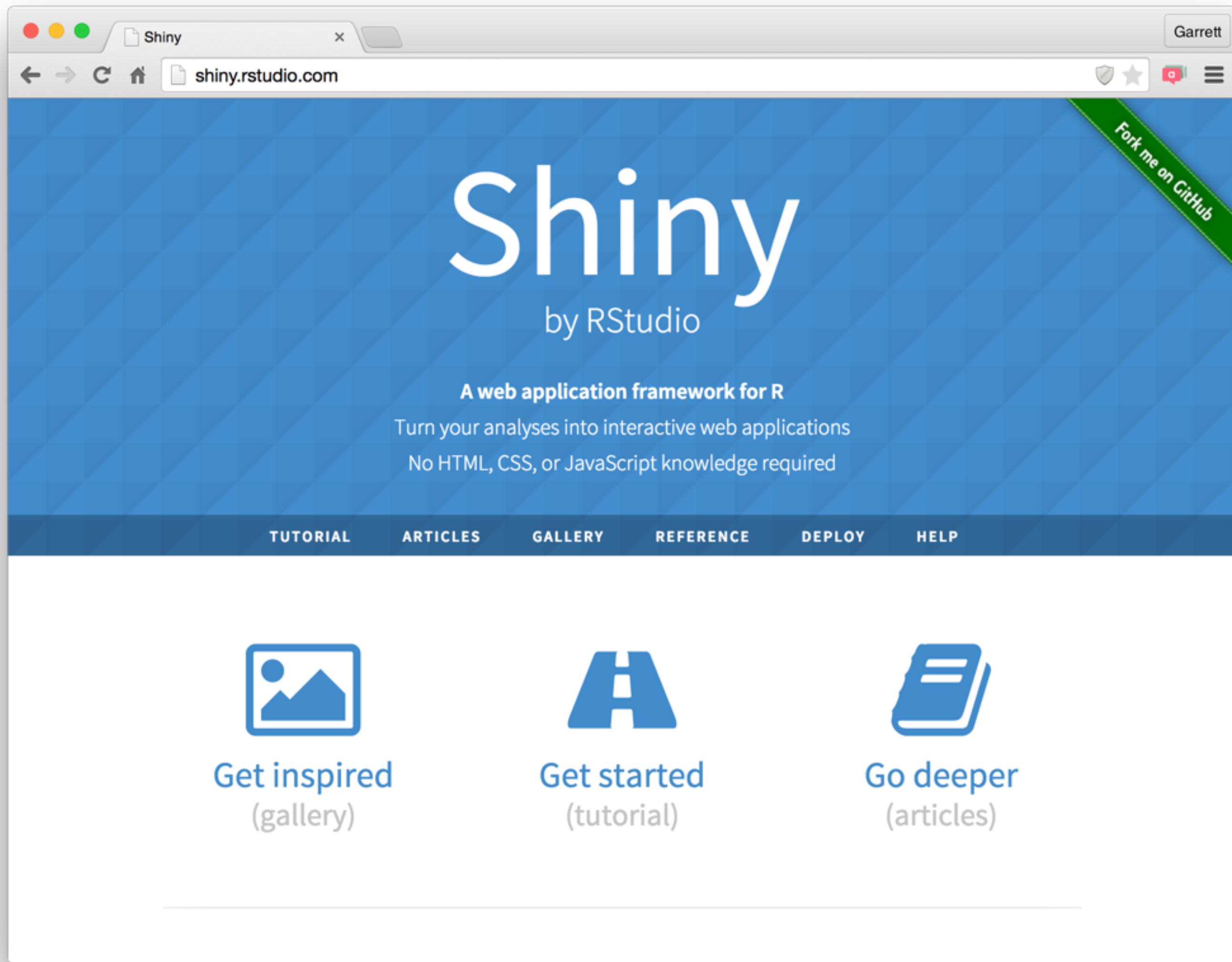
**Basic concepts**

If you're like the vast majority of R users, when you start working on a new R project you create a new directory for all of your R scripts and data files.

Packrat enhances your project directory by storing your package dependencies inside it, rather than relying on your personal R library that is shared across all of your other R sessions. We call this directory your **private package library** (or just **private library**). When you start an R session in a packrat project directory, R will only look for

# The Shiny Development Center

[shiny.rstudio.com](http://shiny.rstudio.com)



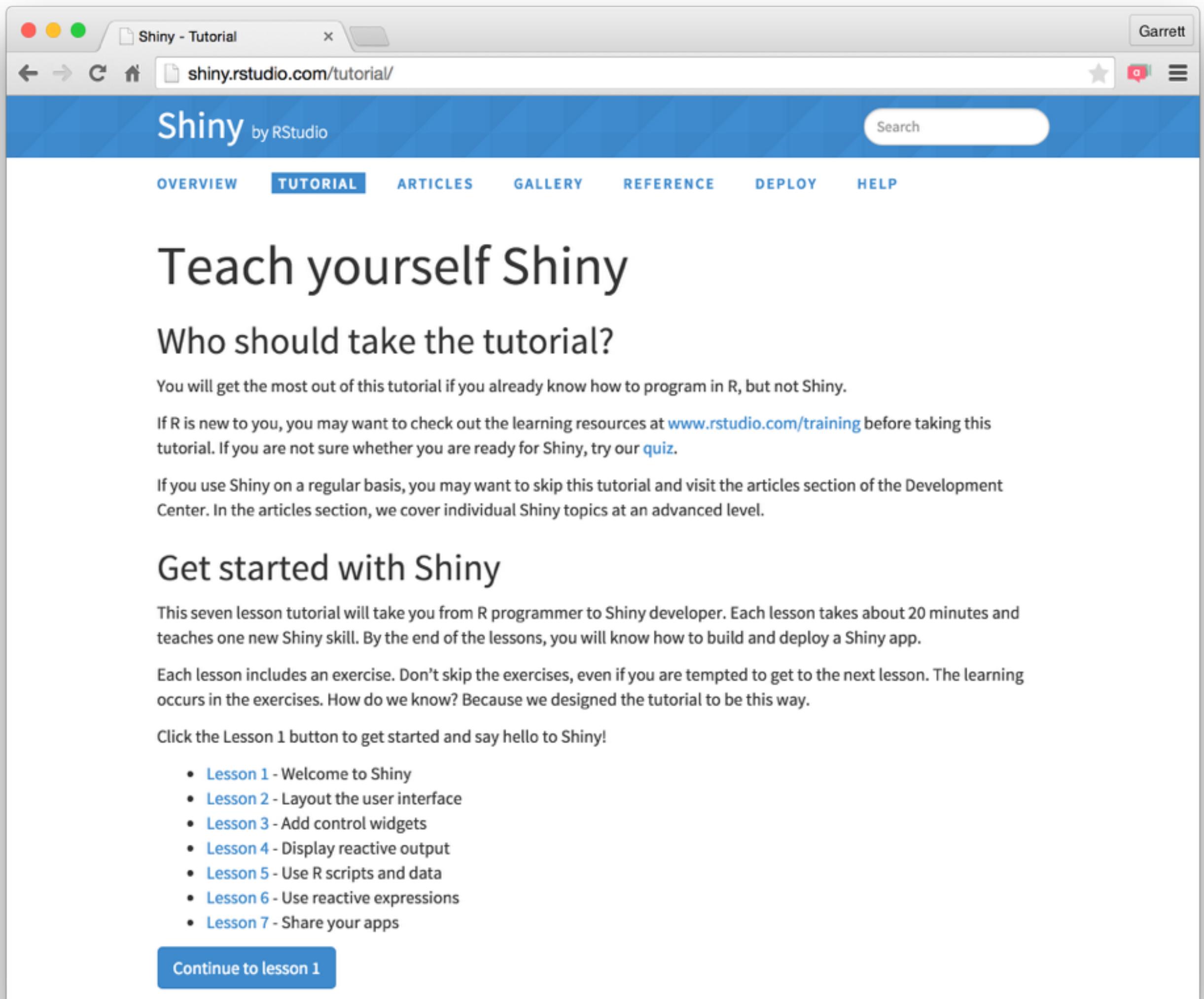
# The Shiny Development Center

[shiny.rstudio.com/gallery](http://shiny.rstudio.com/gallery)

The screenshot shows a web browser displaying the Shiny Gallery. The title bar reads "Shiny - Gallery" and the address bar shows "shiny.rstudio.com/gallery/". The page has a blue header with the "Shiny by RStudio" logo and a search bar. Below the header, there is a navigation menu with tabs: OVERVIEW, TUTORIAL, ARTICLES, GALLERY (which is highlighted in blue), REFERENCE, DEPLOY, and HELP. The main content area is titled "Gallery" and contains a brief introduction: "This gallery contains useful examples to learn from. Visit the [Shiny User Showcase](#) to see an inspiring set of sophisticated apps." Below this, there is a section titled "Interactive visualizations" with a sub-section "Start simple". Under "Interactive visualizations", there are four examples: "SuperZip example" (a map of the US with zip codes), "Bus dashboard" (a map of a city with bus route lines), "Movie explorer" (a scatter plot of movie ratings), and "Google Charts" (a scatter plot of health expenditure vs dependency ratio). Under "Start simple", there are four examples: "Kmeans example" (a scatter plot with three clusters), "Telephones by region" (a bar chart of telephone counts by region), "Faithful" (a histogram of waiting times for eruptions), and "Word cloud" (a word cloud visualization).

# The Shiny Development Center

[shiny.rstudio.com/tutorial](http://shiny.rstudio.com/tutorial)



A screenshot of a web browser window showing the Shiny Tutorial homepage. The title bar says "Shiny - Tutorial". The address bar shows "shiny.rstudio.com/tutorial/". The page has a blue header with the "Shiny by RStudio" logo and a search bar. Below the header, there's a navigation menu with tabs: OVERVIEW, TUTORIAL (which is selected), ARTICLES, GALLERY, REFERENCE, DEPLOY, and HELP. The main content area has a large heading "Teach yourself Shiny" and a sub-section "Who should take the tutorial?". It includes text about the target audience and links to R training and a quiz. Another section, "Get started with Shiny", describes the seven-lesson tutorial and encourages users to click "Lesson 1". A list of lessons is provided, and a "Continue to lesson 1" button is at the bottom.

## Teach yourself Shiny

### Who should take the tutorial?

You will get the most out of this tutorial if you already know how to program in R, but not Shiny.

If R is new to you, you may want to check out the learning resources at [www.rstudio.com/training](http://www.rstudio.com/training) before taking this tutorial. If you are not sure whether you are ready for Shiny, try our [quiz](#).

If you use Shiny on a regular basis, you may want to skip this tutorial and visit the articles section of the Development Center. In the articles section, we cover individual Shiny topics at an advanced level.

## Get started with Shiny

This seven lesson tutorial will take you from R programmer to Shiny developer. Each lesson takes about 20 minutes and teaches one new Shiny skill. By the end of the lessons, you will know how to build and deploy a Shiny app.

Each lesson includes an exercise. Don't skip the exercises, even if you are tempted to get to the next lesson. The learning occurs in the exercises. How do we know? Because we designed the tutorial to be this way.

Click the Lesson 1 button to get started and say hello to Shiny!

- [Lesson 1 - Welcome to Shiny](#)
- [Lesson 2 - Layout the user interface](#)
- [Lesson 3 - Add control widgets](#)
- [Lesson 4 - Display reactive output](#)
- [Lesson 5 - Use R scripts and data](#)
- [Lesson 6 - Use reactive expressions](#)
- [Lesson 7 - Share your apps](#)

[Continue to lesson 1](#)

# The Shiny Development Center

## [shiny.rstudio.com/articles](http://shiny.rstudio.com/articles)

The screenshot shows a web browser window displaying the Shiny Development Center at [shiny.rstudio.com/articles/](http://shiny.rstudio.com/articles/). The page has a blue header with the Shiny logo and navigation links for Overview, Tutorial, Articles (which is selected), Gallery, Reference, Deploy, and Help. A search bar is also present. The main content area is titled "Articles". It features two main sections: "The basics" and "Extend Shiny". The "The basics" section lists articles like "The basic parts of a Shiny app", "How to build a Shiny app", etc. The "Extend Shiny" section lists various R packages: shinythemes, shinydashboard, htmlwidgets, leaflet, dygraphs, MetricsGraphics, networkD3, DataTables, threejs, and rCharts. Below these are "Layouts and UI" and "Deploying apps" sections with their respective articles.

## Articles

### The basics

If you've been through the [tutorial](#) and need a refresher, these articles are a good place to start. They describe the lay of the land.

- [The basic parts of a Shiny app](#)
- [How to build a Shiny app](#)
- [How to launch a Shiny app](#)
- [How to get help](#)
- [The Shiny cheat sheet](#)
- [Single-file Shiny apps](#)
- [App formats and launching methods](#)

### Extend Shiny

These packages provide advanced features that can enhance your Shiny apps.

- [shinythemes](#) - CSS themes ready to use with Shiny
- [shinydashboard](#) - Shiny powered dashboards
- [htmlwidgets](#) - A framework for embedding JavaScript visualizations into R. Ready to use examples include:
  - [leaflet](#) - Geo-spatial mapping
  - [dygraphs](#) - Time series charting
  - [MetricsGraphics](#) - Scatterplots and line charts with D3
  - [networkD3](#) - Graph data visualization with D3
  - [DataTables](#) - Tabular data display
  - [threejs](#) - 3D scatterplots and globes
  - [rCharts](#) - Multiple JavaScript charting libraries

### Layouts and UI

These articles explain how to control the layout, user-interface, and general appearance of your Shiny apps.

- [Application layout guide](#)
- [Display modes](#)
- [Tabs](#)
- [Customize your UI with HTML](#)

### Deploying apps

These articles describe the different ways to share your Shiny apps with users.

- [Getting started with shinyapps.io](#)
- [Shinyapps.io Scaling and Performance Tuning \(beta\)](#)
- [Migrating shinyapps.io authentication \(beta\)](#)
- [Share data across sessions with shinyapps.io](#)

# Thank you

## Reactivity with Shiny

Slides at: [bit.ly/EARL2015-Reports](http://bit.ly/EARL2015-Reports)

Much more at: [shiny.rstudio.com](http://shiny.rstudio.com)

R Markdown at: [rmarkdown.rstudio.com](http://rmarkdown.rstudio.com)

Packrat at: [rstudio.github.io/packrat/](http://rstudio.github.io/packrat/)