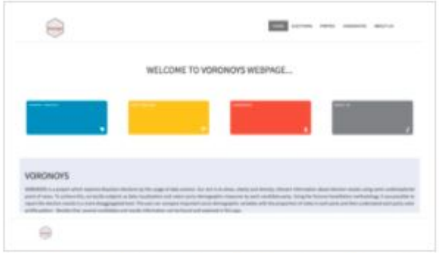
Shiny Apps

Guillermo Solovey

Web apps usando R

Government / Public sector

Mostly open data



Voronoys - Understanding voters' profile in Brazilian elections

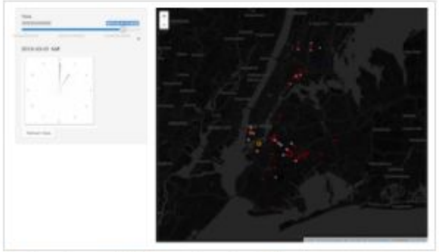
Crime Watch

Pasture Potential Tool for improving dairy farm profitability and environmental impact

Dublin Transport Info

Locating Blood Banks in India

Utah Lake Water Quality Profile Dashboard



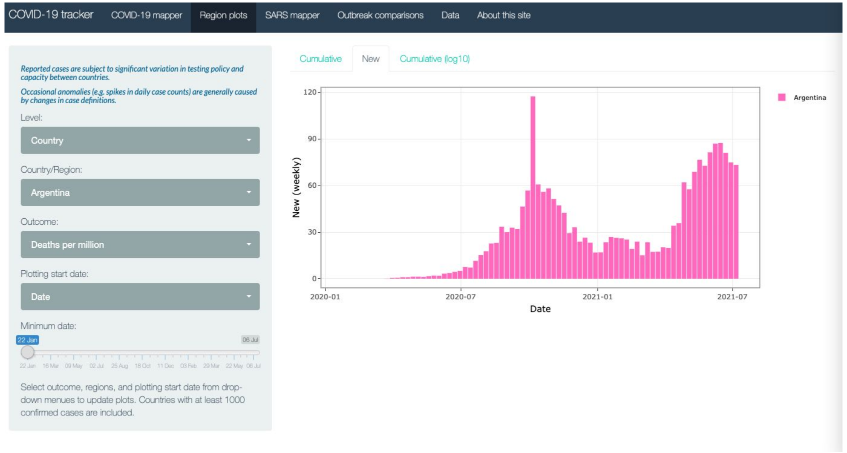
Animated NYC metro traffic

New Zealand trade for the year ended June 2018
Goods
Services
New Zealand trade over the past 20 years

New Zealand Trade Intelligence Dashboard

https://shiny.rstudio.com/gallery/

Web apps usando R



Reported cases are subject to significant variation in testing policy and capacity between countries.

Occasional anomalies (e.g. spikes in daily case counts) are generally caused by changes in case definitions.

Level: Country

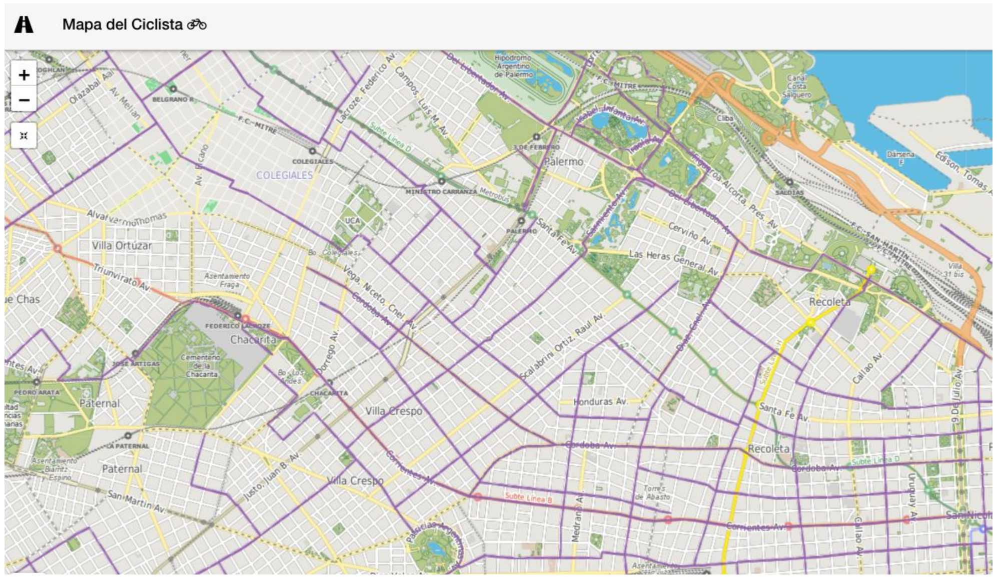
Country/Region: Argentina

Outcome: Deaths per million

Plotting start date: Date

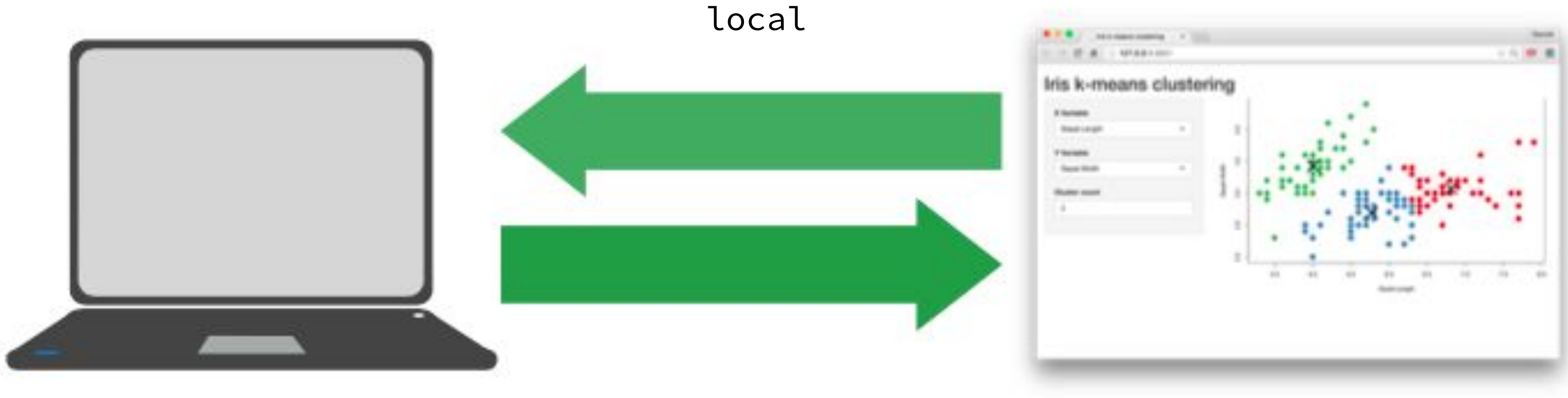
Minimum date: 22 Jan – 06 Jul

Select outcome, regions, and plotting start date from drop-down menues to update plots. Countries with at least 1000 confirmed cases are included.

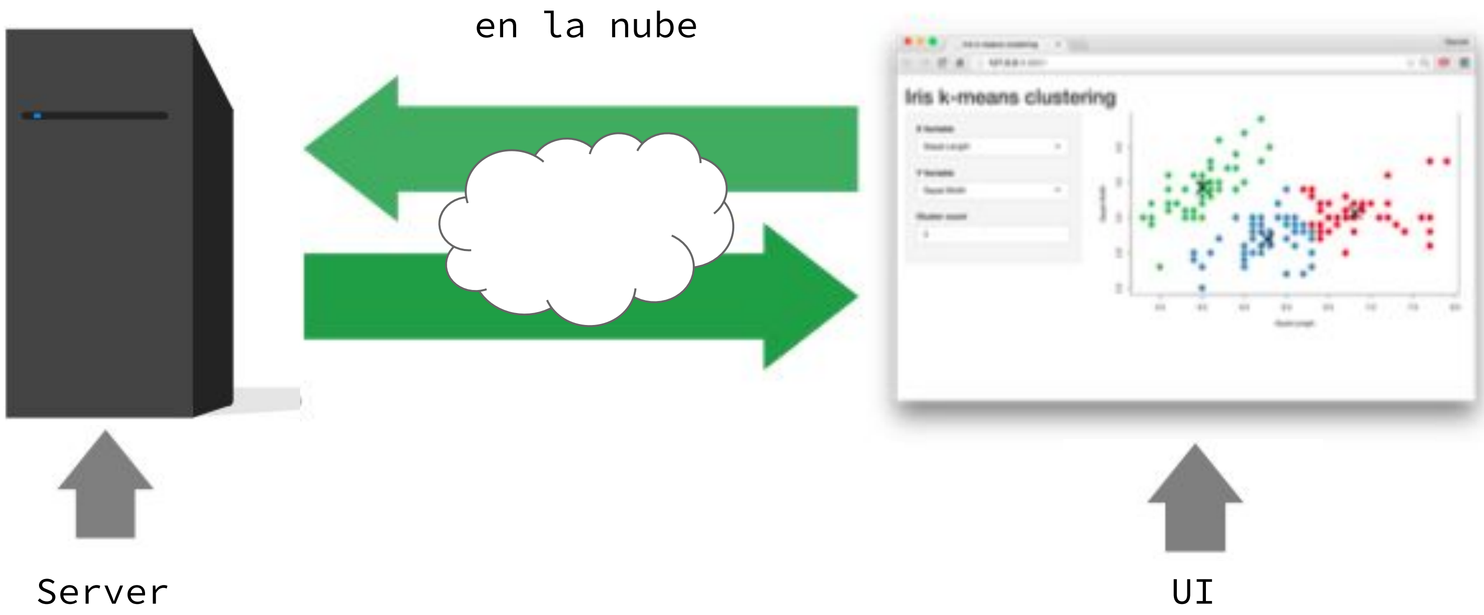


Estructura de una shiny app

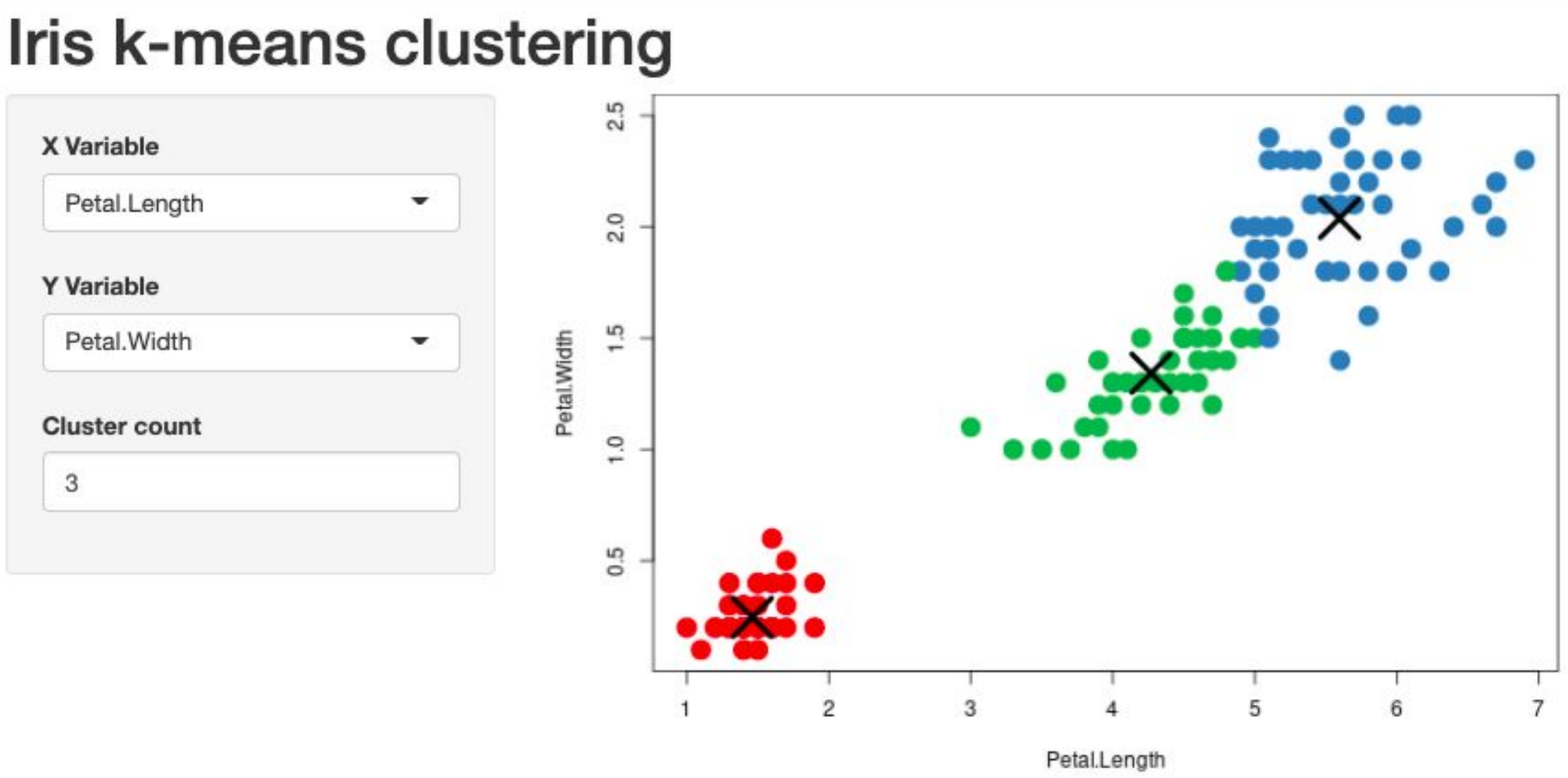
Una shiny app corre en R



Una shiny app corre en R



Shiny app



Iris k-means clustering

X Variable
Petal.Length

Y Variable
Petal.Width

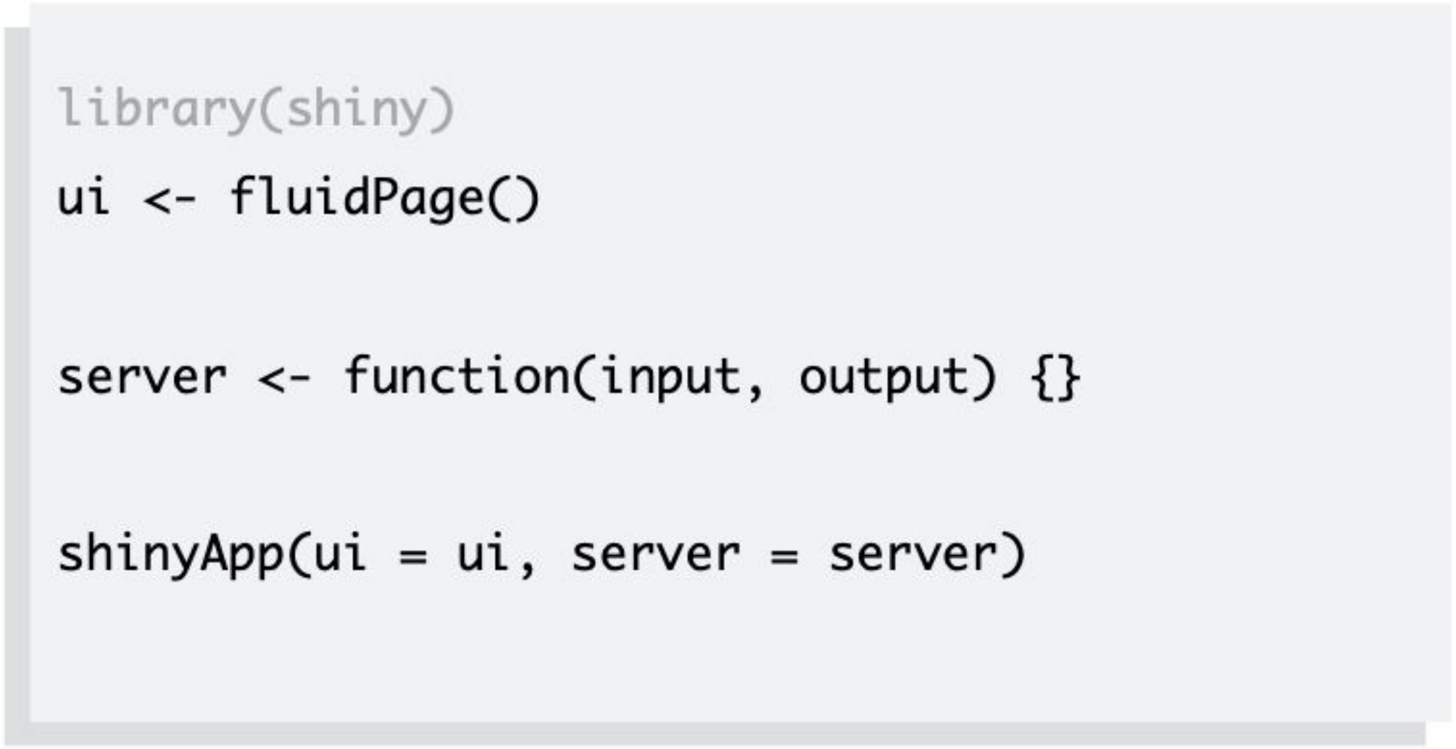
Cluster count
3

Clase intro a shiny

- Estructura de una shiny app: UI y server.
- Tipos de inputs: slider, text, numbers, …
- Tipos de outputs: plots, tablas, ...
- Subir la app a shinyapps.io
- Algunas ideas para cambiarle la apariencia (themes)
- Dónde seguir aprendiendo

Template

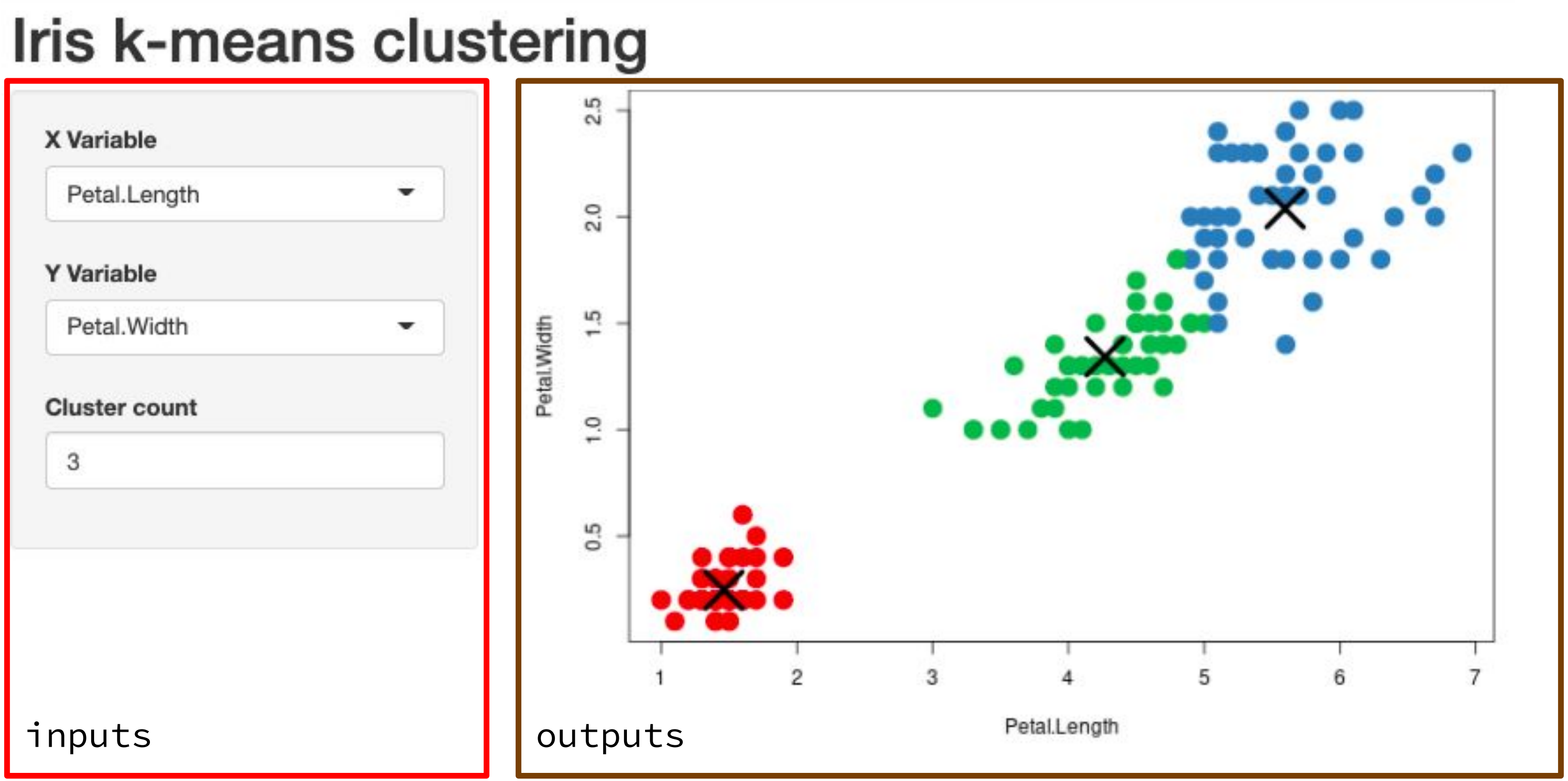
Copiar este template mínimo en el editor de RStudio



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

server <- function(input, output) {}

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

© CC 2015 RStudio, Inc.

Pensar en términos de inputs y outputs



Iris k-means clustering

X Variable

Petal.Length

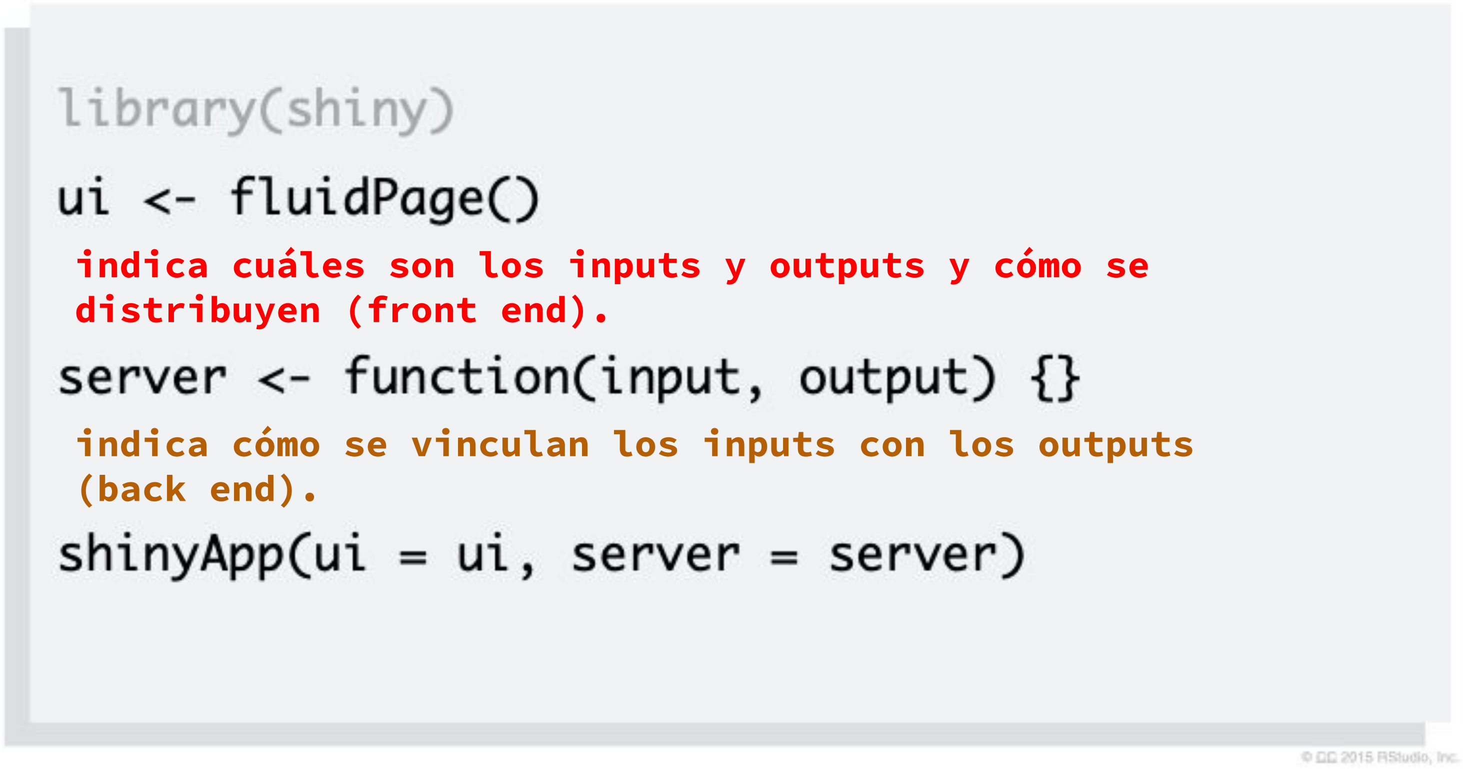
Y Variable

Petal.Width

Cluster count

3

Pensar en términos de inputs y outputs



```
library(shiny)

ui <- fluidPage()
```

indica cuáles son los inputs y outputs y cómo se distribuyen (front end).

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

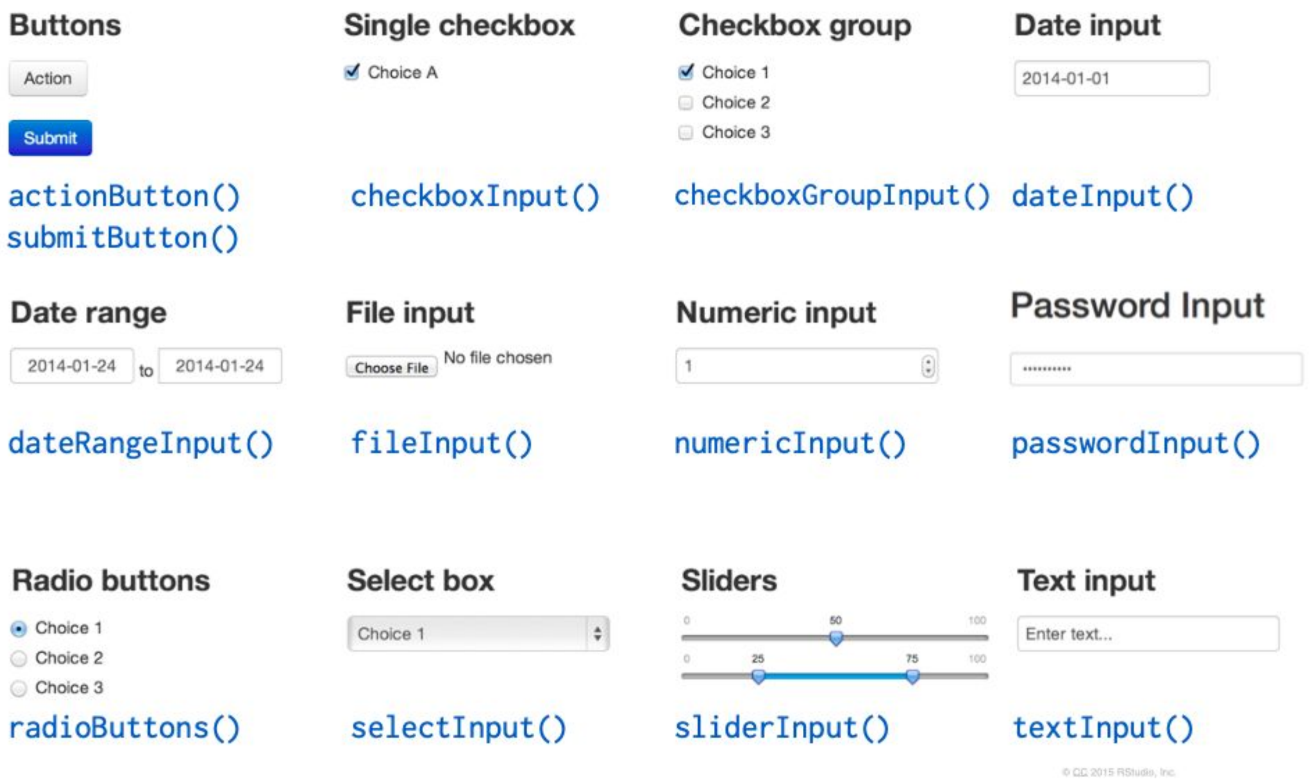
indica cómo se vinculan los inputs con los outputs (back end).

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

© CC 2015 RStudio, Inc.

INPUTS

Tipos de inputs



Buttons

`actionButton()`
`submitButton()`

Single checkbox

`checkboxInput()`

Checkbox group

`checkboxGroupInput()`

Date input

`dateInput()`

Date range

`dateRangeInput()`

File input

`fileInput()`

Numeric input

`numericInput()`

Password Input

`passwordInput()`

Radio buttons

`radioButtons()`

Select box

`selectInput()`

Sliders

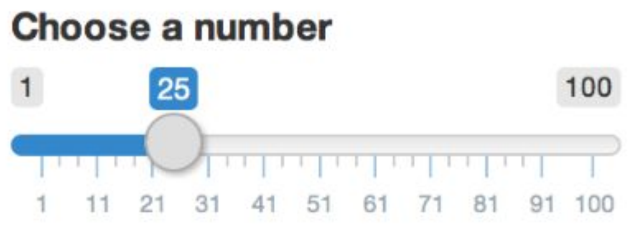
`sliderInput()`

Text input

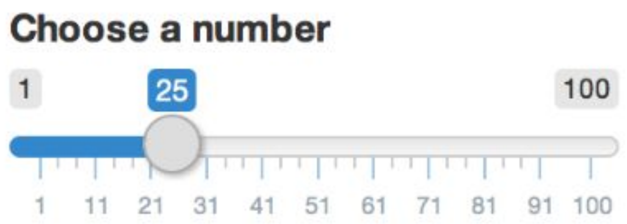
`textInput()`

Crear un input que sea un slider y lo llamamos "num"

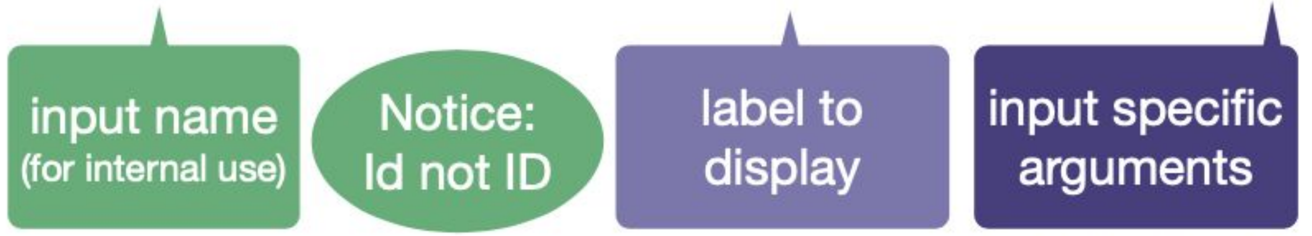
```r
library(shiny)
ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100)
)

server <- function(input, output) {}

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



Sintaxis



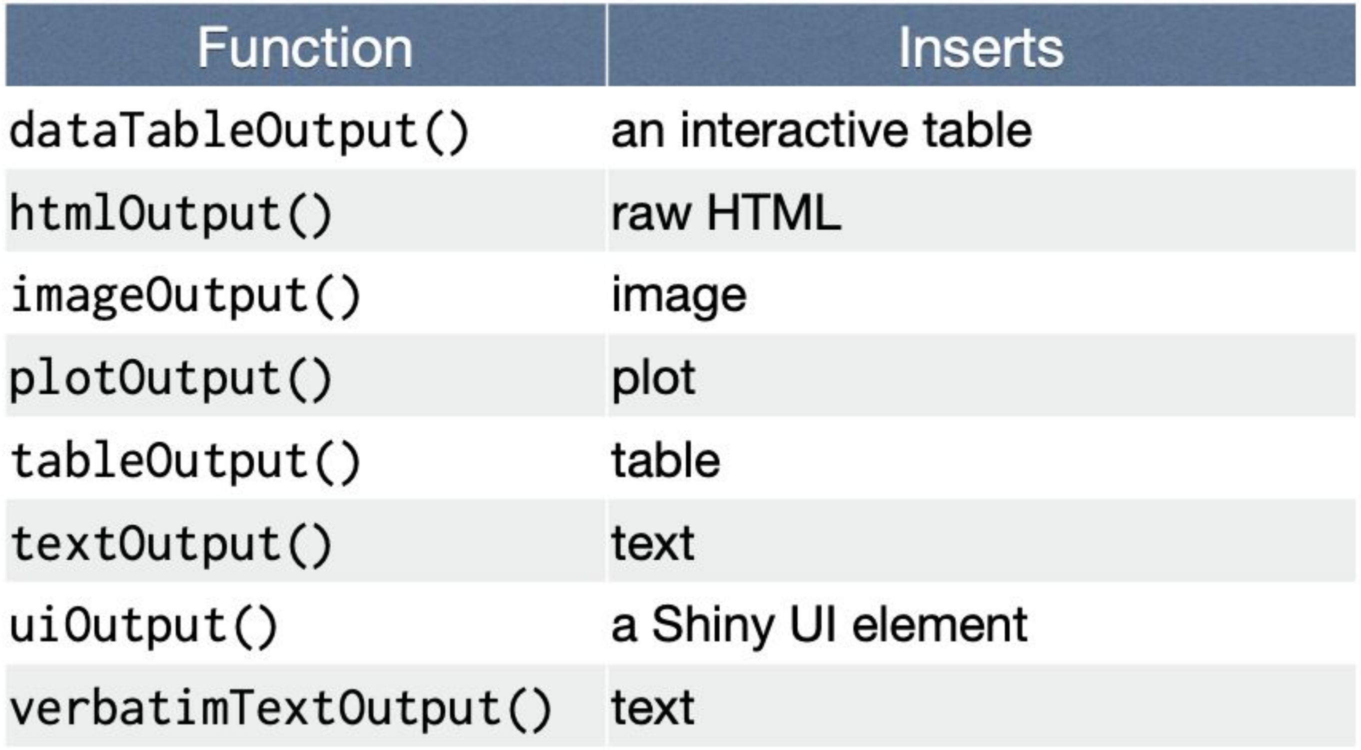
```
sliderInput(inputId = "num", label = "Choose a number", …)
```



```
?sliderInput
```

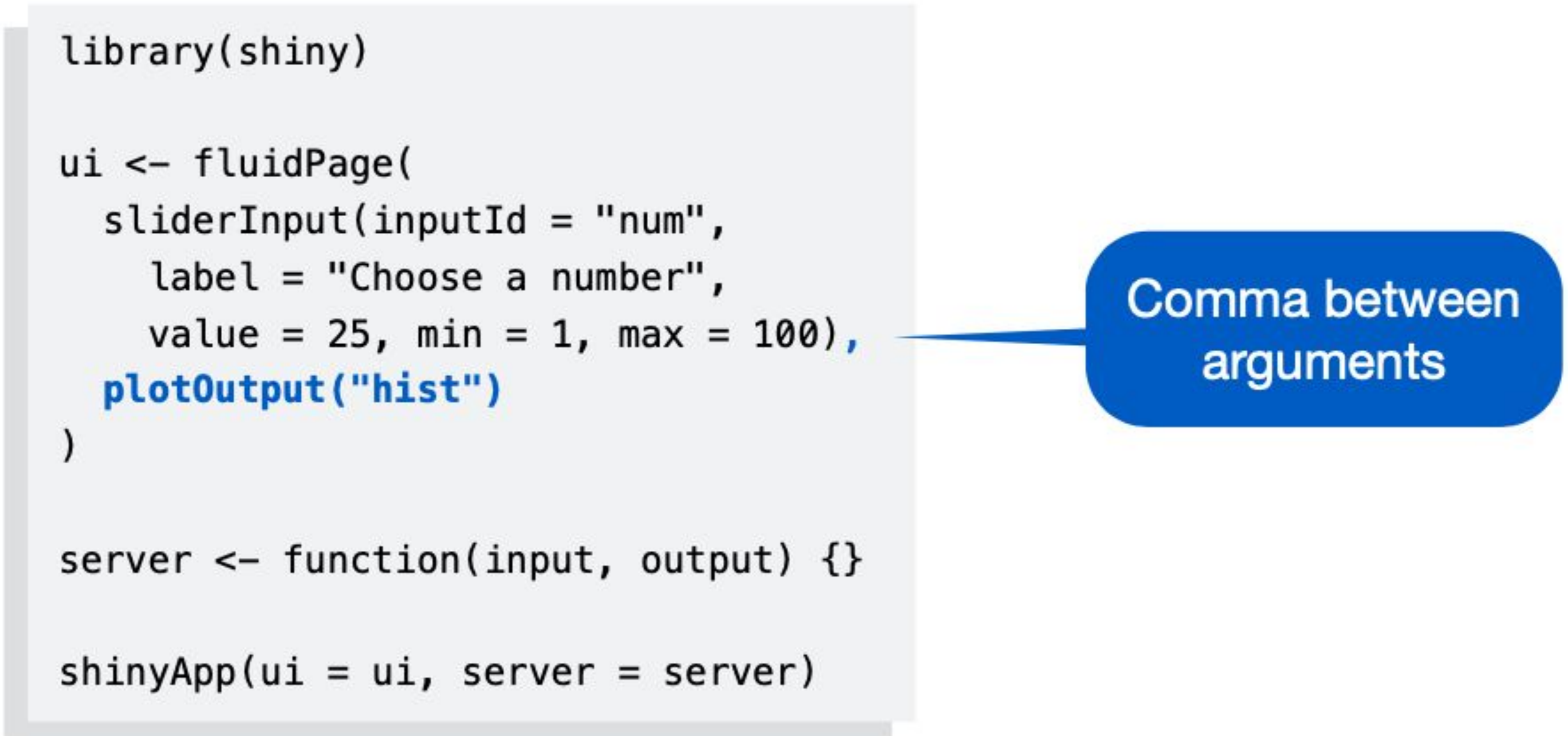
OUTPUTS

Tipos de outputs



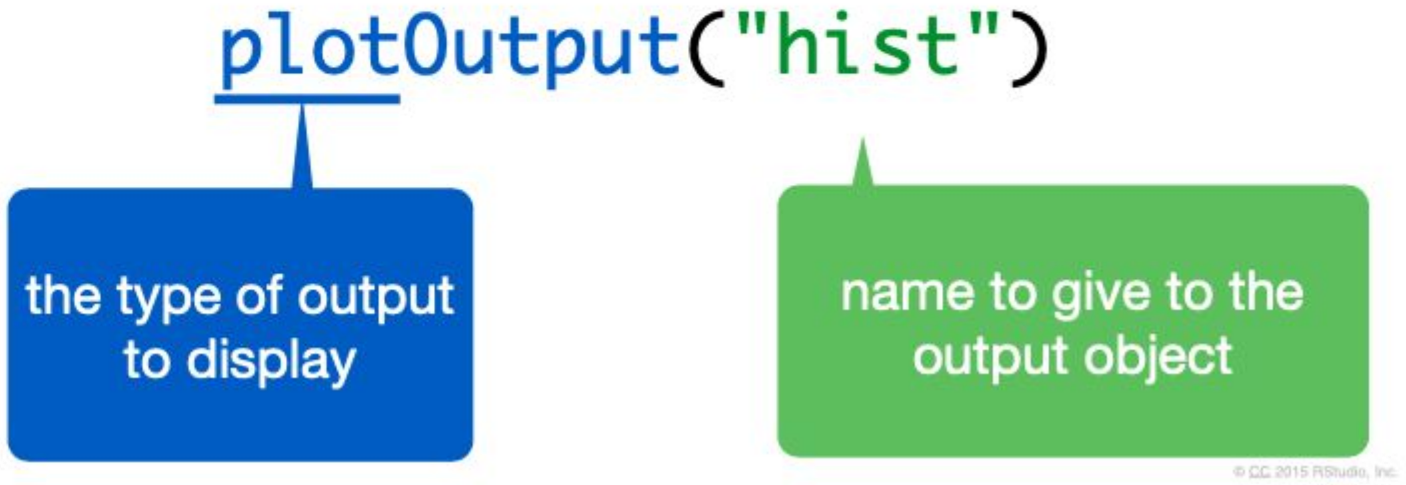
Function	Inserts
`dataTableOutput()`	an interactive table
`htmlOutput()`	raw HTML
`imageOutput()`	image
`plotOutput()`	plot
`tableOutput()`	table
`textOutput()`	text
`uiOutput()`	a Shiny UI element
`verbatimTextOutput()`	text

Crear un output que sea un plot y lo llamamos "hist"



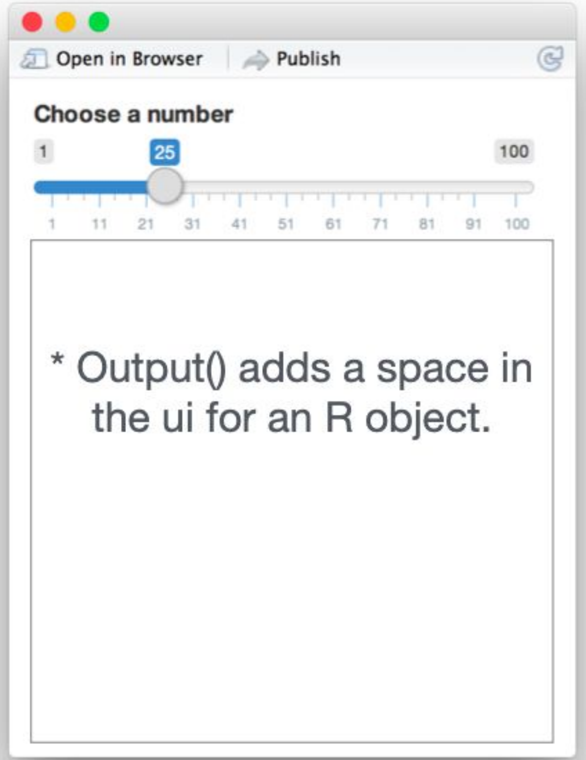
```
library(shiny)

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)
```

Sintaxis


plotOutput("hist")
the type of output to display
name to give to the output object
© CC 2015 RStudio, Inc.

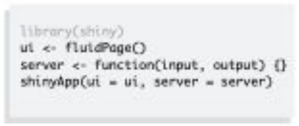
Quedan definidos los inputs y outputs en la UI

```
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.

Resumen hasta acá: UI



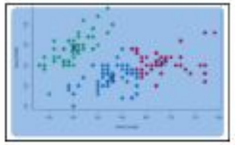
```
library(shiny)
ui <- fluidPage()
server <- function(input, output) {}
shinyApp(ui = ui, server = server)
```

Empezar la app con un template

Hello World

Agregar elementos en fluidPage()

Crear inputs reactivos con una función *Input()



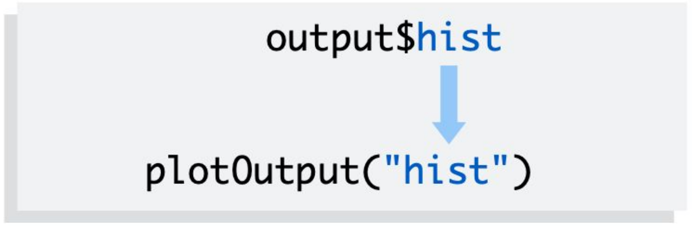
Mostrar resultados reactivos con una función *Output()

Conectar inputs y outputs con código dentro de la función server()

SERVER

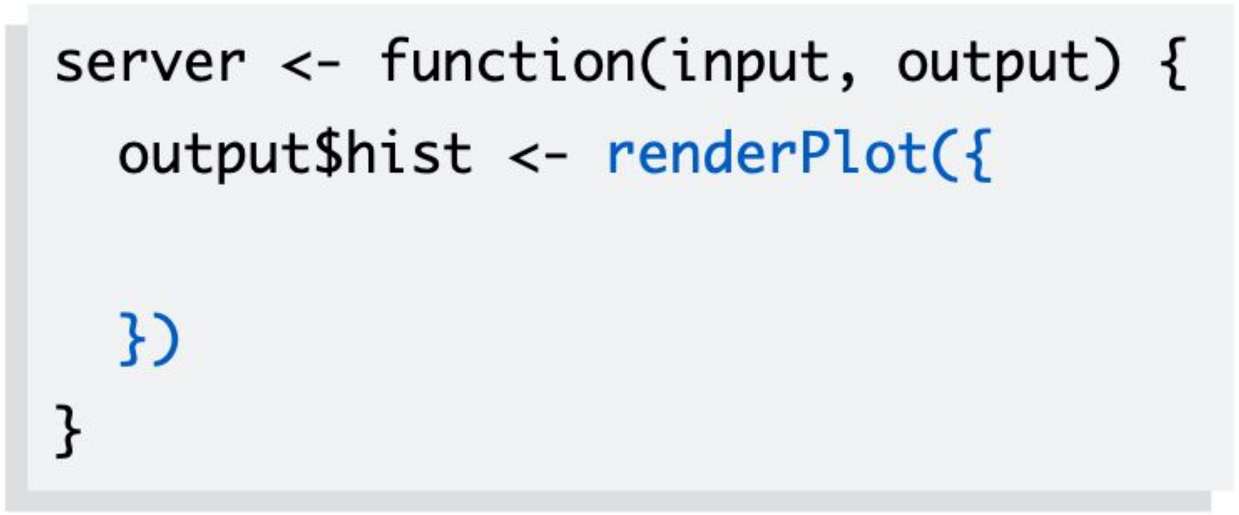
1/3 Crear el objeto que vamos a mostrar como output

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



}
```



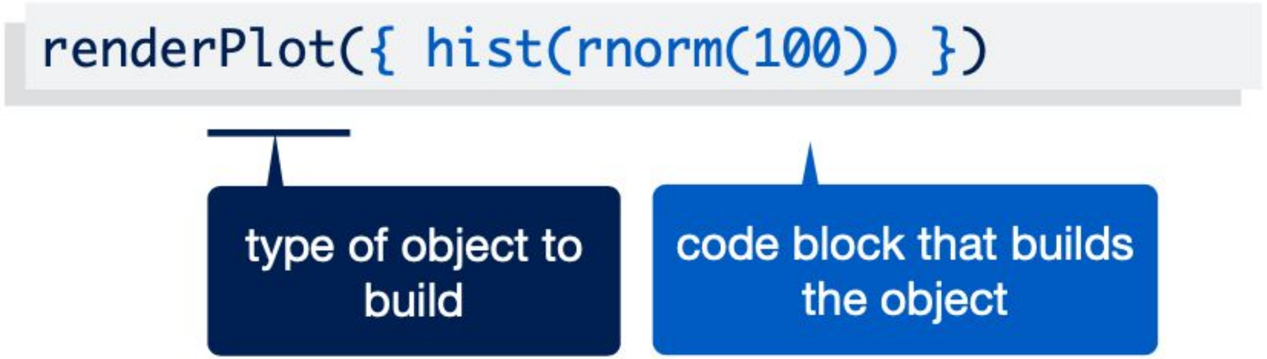
```
output$hist

plotOutput("hist")
```

2/3 Crear objetos para mostrar con render*()



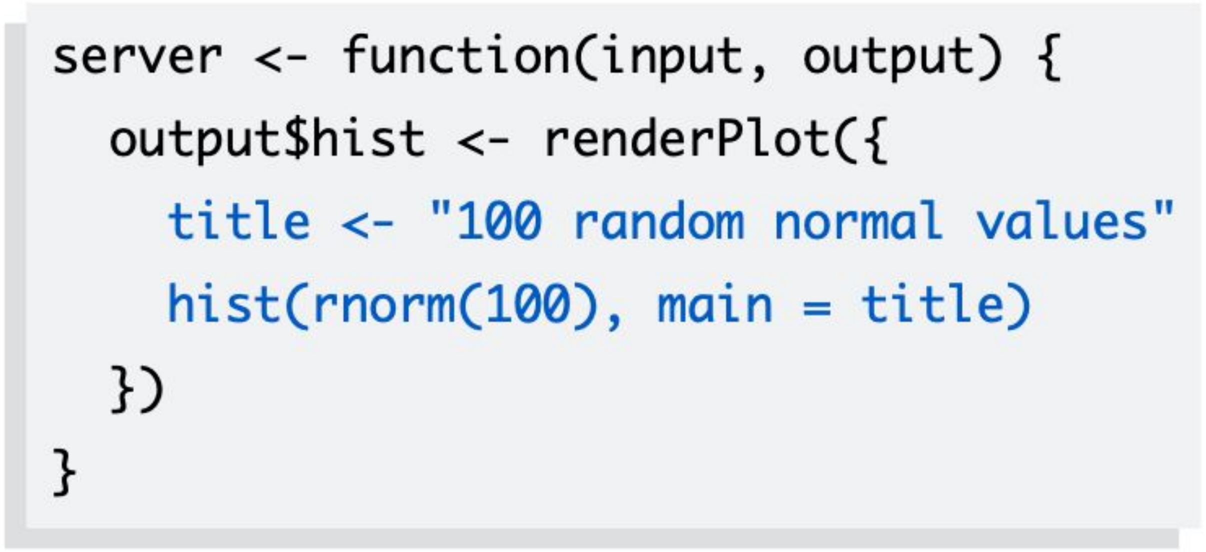
```
server <- function(input, output) {
  output$hist <- renderPlot({

  })
}
```

Código de R que general el objeto a plotear



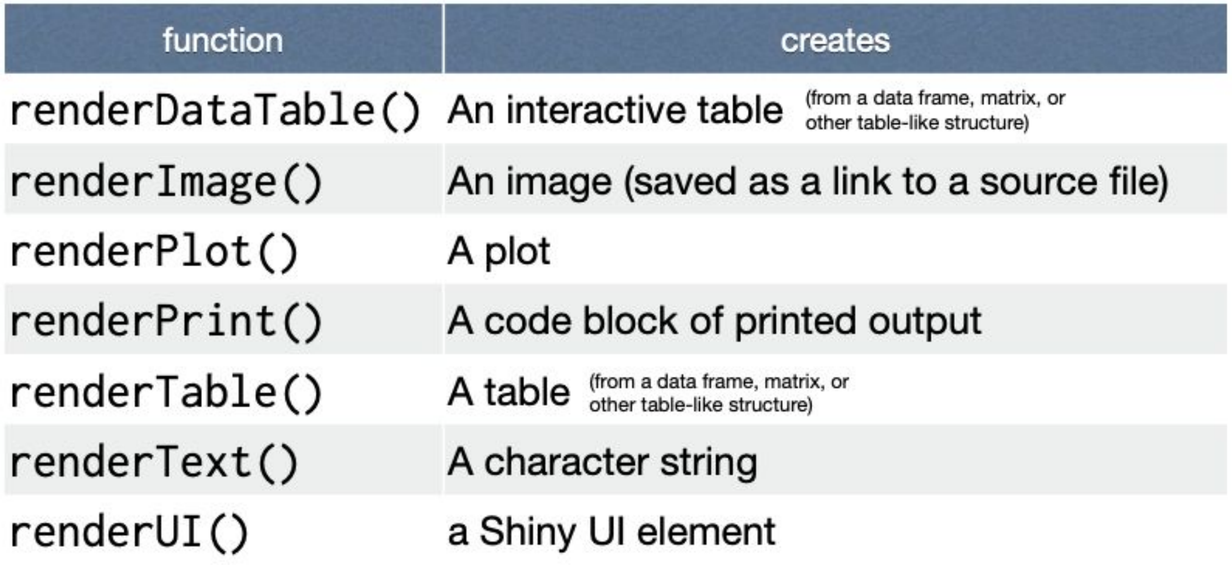
```
renderPlot({ hist(rnorm(100)) })
```

Código de R que general el objeto a plotear



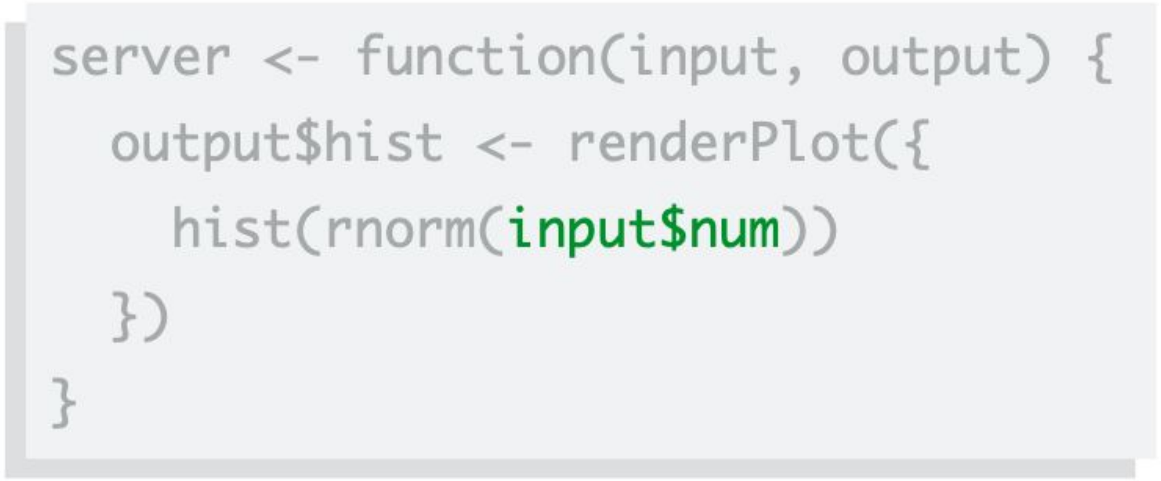
```
server <- function(input, output) {
  output$hist <- renderPlot({
    title <- "100 random normal values"
    hist(rnorm(100), main = title)
  })
}
```

Tipos de render

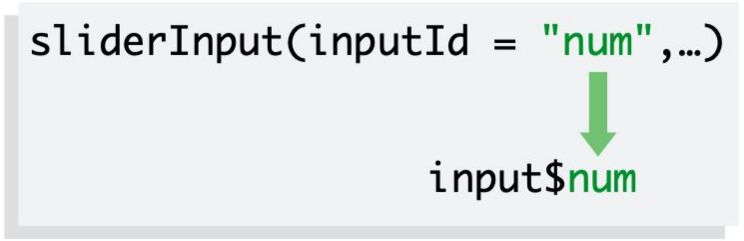


function	creates
`renderDataTable()`	An interactive table (from a data frame, matrix, or other table-like structure)
`renderImage()`	An image (saved as a link to a source file)
`renderPlot()`	A plot
`renderPrint()`	A code block of printed output
`renderTable()`	A table (from a data frame, matrix, or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

3/3 Usar inputs que modifiquen el output

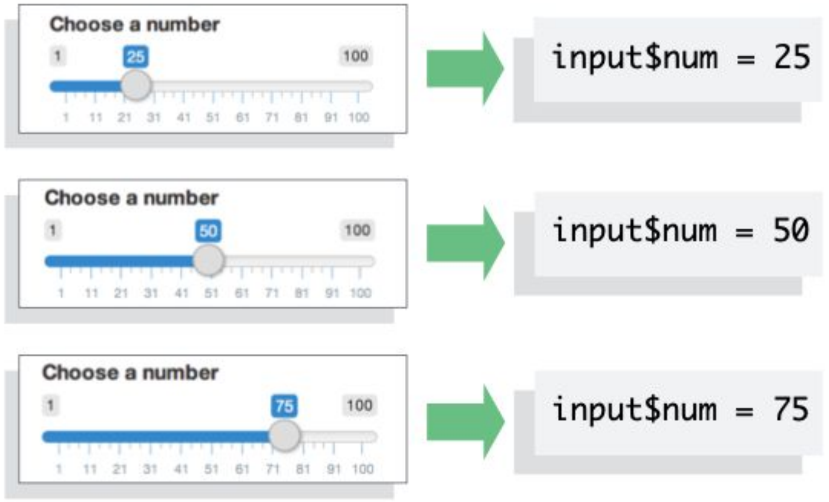


```
server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
}
```

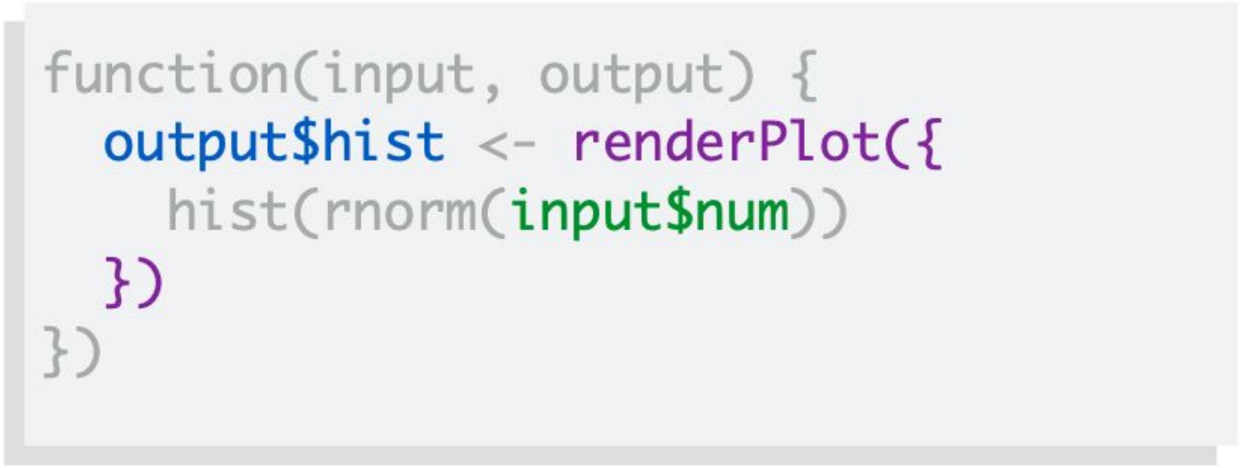


```
sliderInput(inputId = "num",…)

                 input$num
```

3/3 Usar inputs que modifiquen el output



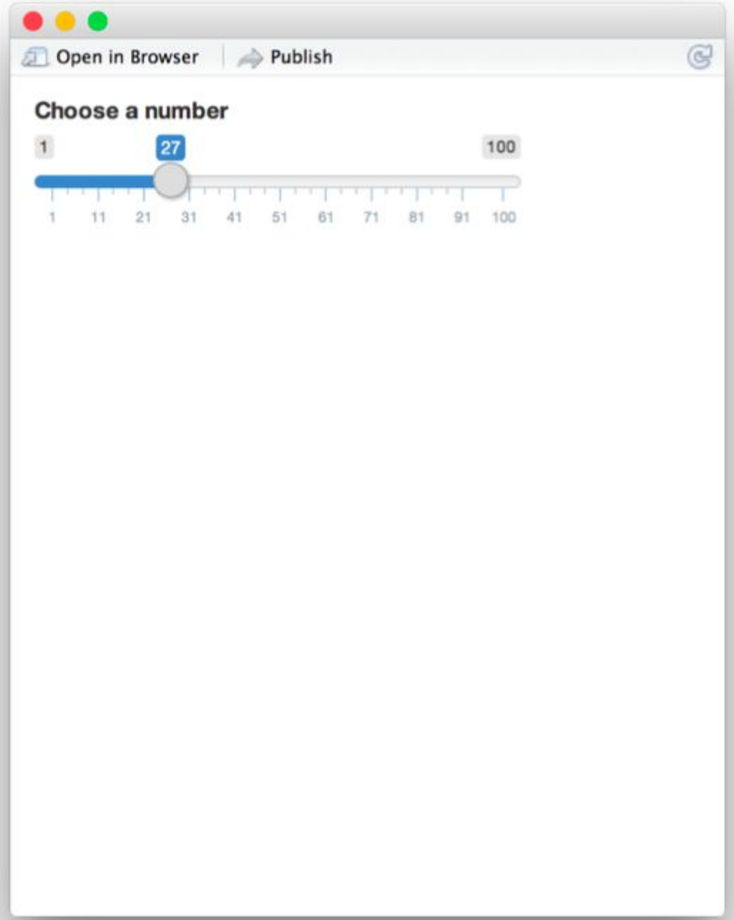
Es un ejemplo de reactividad



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

```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 <-


}

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

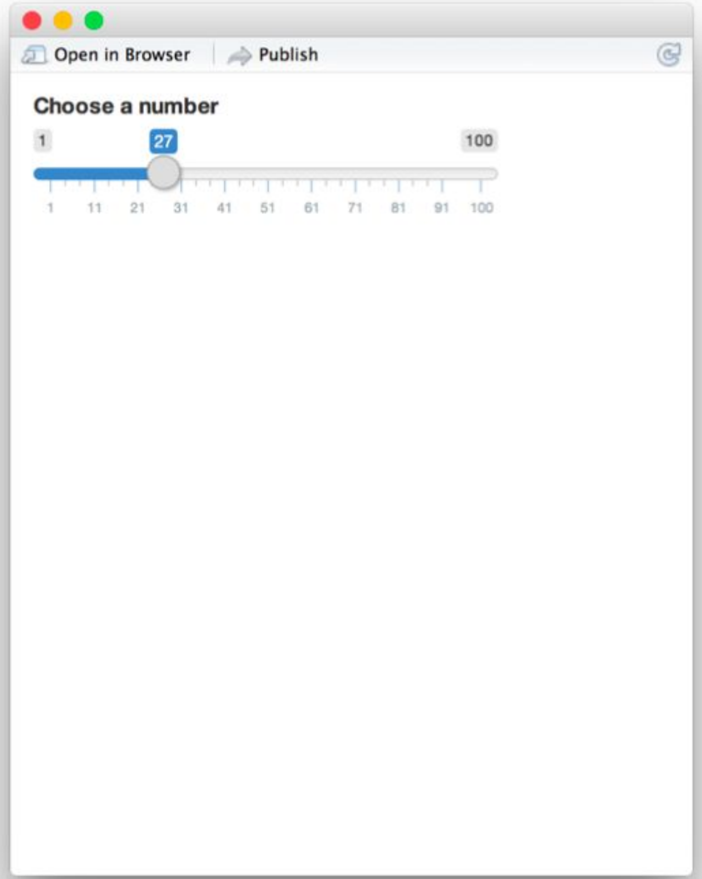
1



© CC 2015 RStudio, Inc.

```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({

  })
}

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

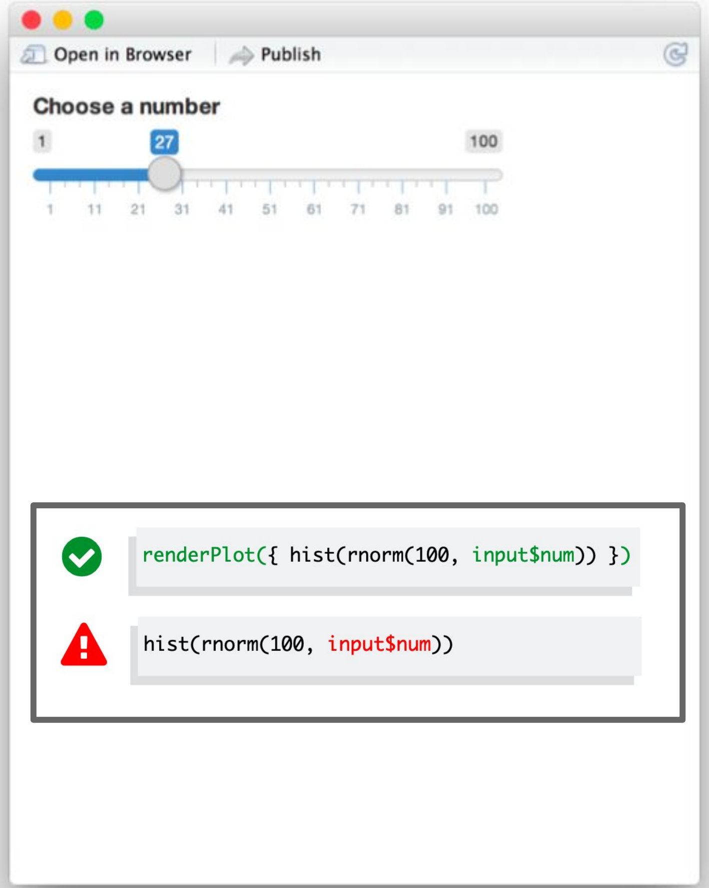
2



© CC 2015 RStudio, Inc.

```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)
```

3



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

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

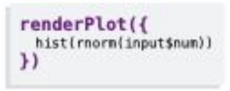
© CC 2015 RStudio, Inc.

Resumen server

Guardar el output que vamos a crear en “output$”

```
output$hist <-
```

Crear el output con una función render*()

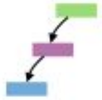


```
renderPlot({
  hist(rnorm(input$num))
})
```

Usar valores ingresados por el usuario con input$

```
input$num
```

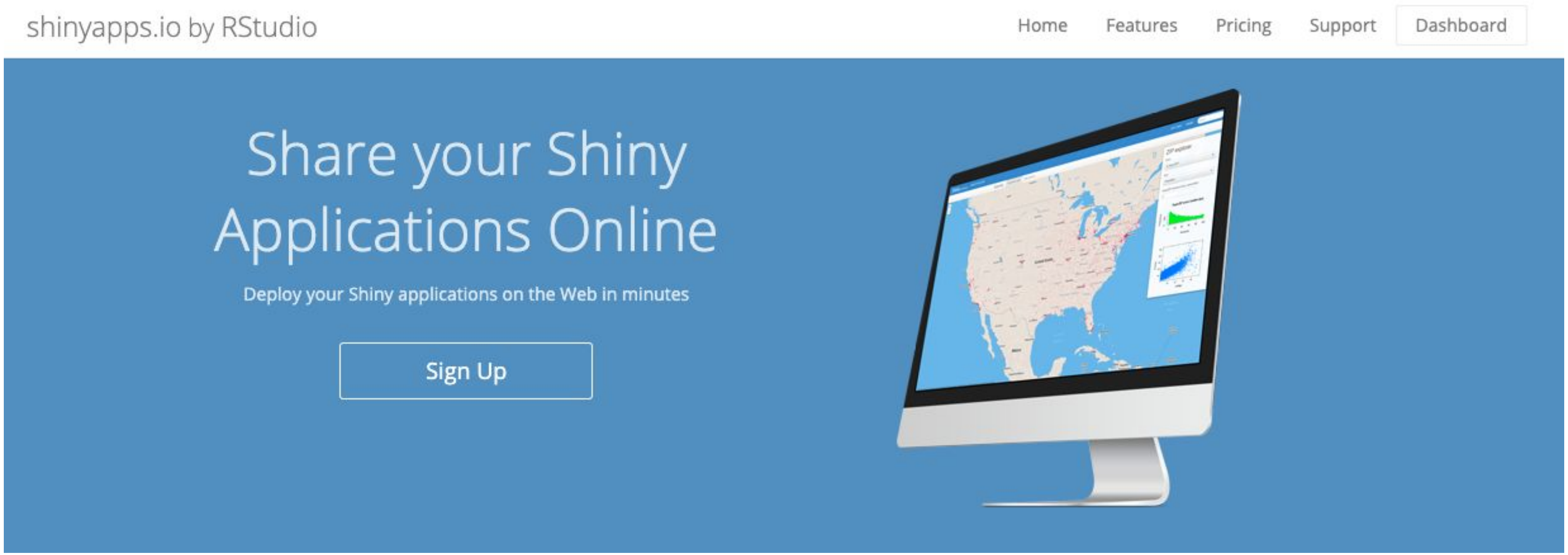
Crear reactividad usando inputs para crear outputs renderizados



Conectar inputs y outputs con código dentro de la función server()

Compartir

SHINYAPPS.IO


shinyapps.io by RStudio

Share your Shiny Applications Online

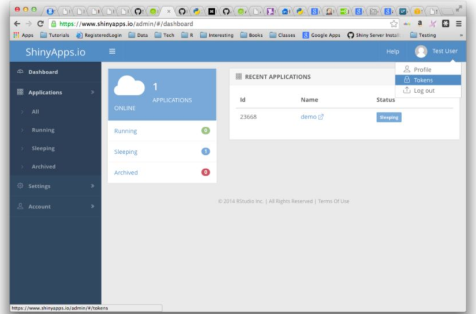
Deploy your Shiny applications on the Web in minutes

Cómo empezar con shinyapps.io

```r
install.packages('rsconnect')
```

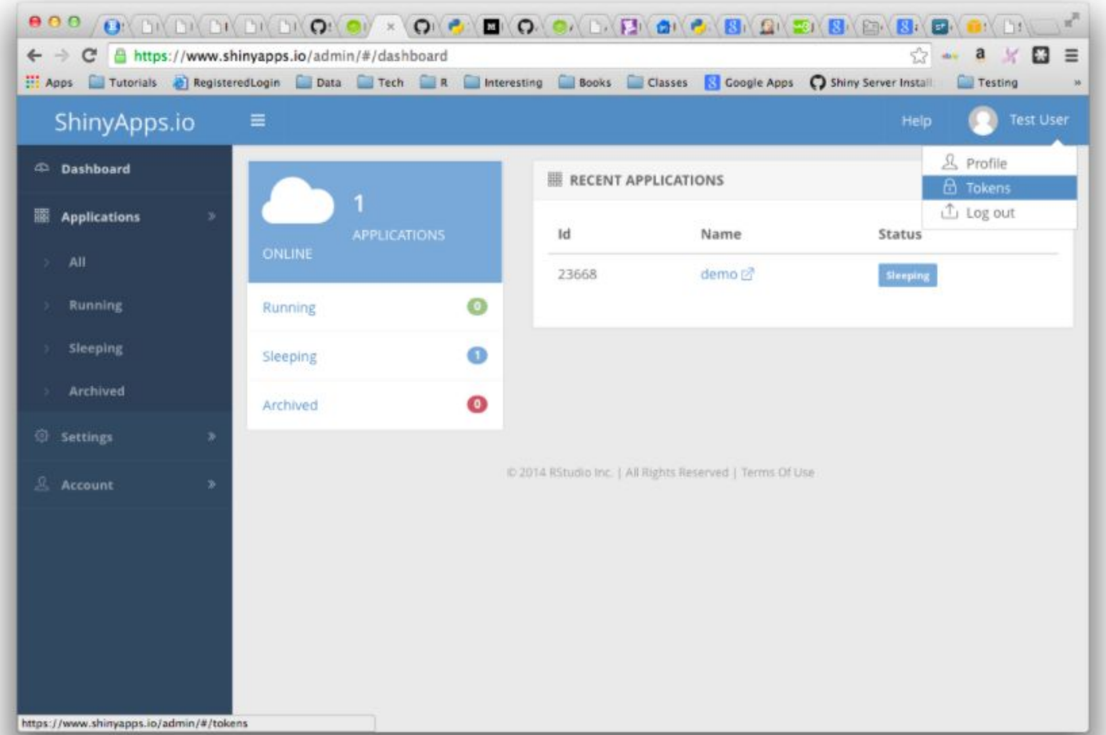
```r
library(rsconnect)
```

Creen una cuenta gratuita en shinyapps.io (pueden entrar con el gmail)



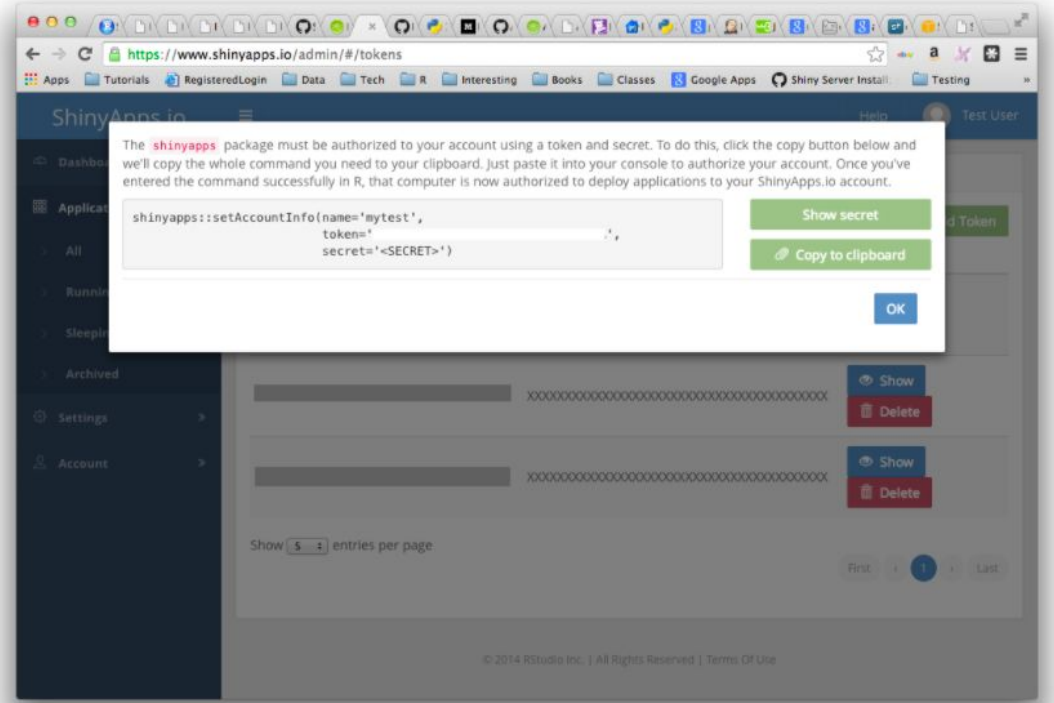
Id	Name	Status
23668	demo	Sleeping

Cómo empezar con shinyapps.io



Id	Name	Status
23668	demo	Sleeping

Cómo empezar con shinyapps.io

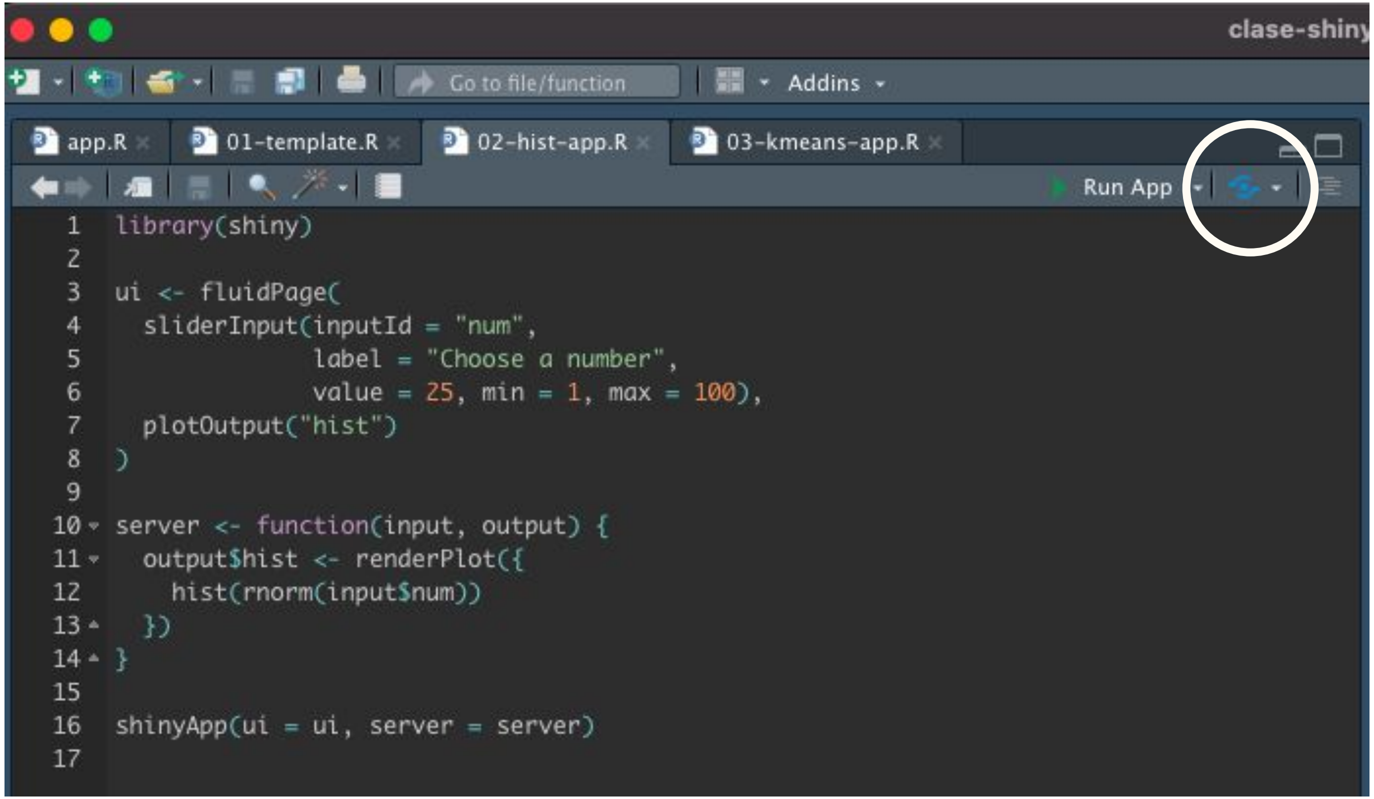


The `shinyapps` package must be authorized to your account using a token and secret. To do this, click the copy button below and we'll copy the whole command you need to your clipboard. Just paste it into your console to authorize your account. Once you've entered the command successfully in R, that computer is now authorized to deploy applications to your ShinyApps.io account.

```
shinyapps::setAccountInfo(name='mytest',
                          token='[redacted]',
                          secret='<SECRET>')
```

© 2014 RStudio Inc. | All Rights Reserved | Terms Of Use

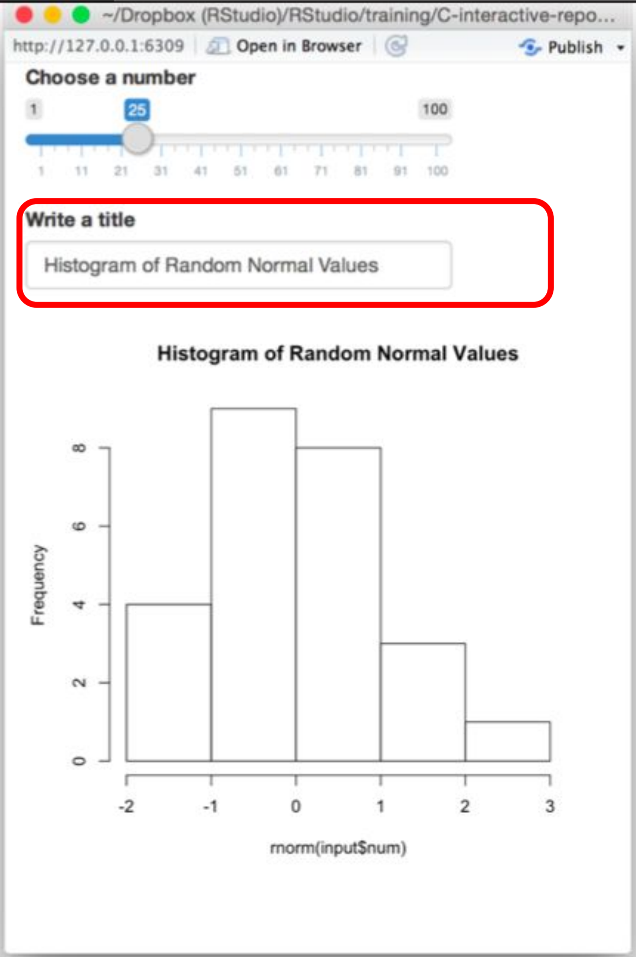
Publicar una app en shinyapps.io



```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)
```

2 INPUTS

```
# 01-two-inputs

library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  textInput(inputId = "title",
    label = "Write a title",
    value = "Histogram of Random Normal Values"),
  plotOutput("hist")
)

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

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



Choose a number

Write a title

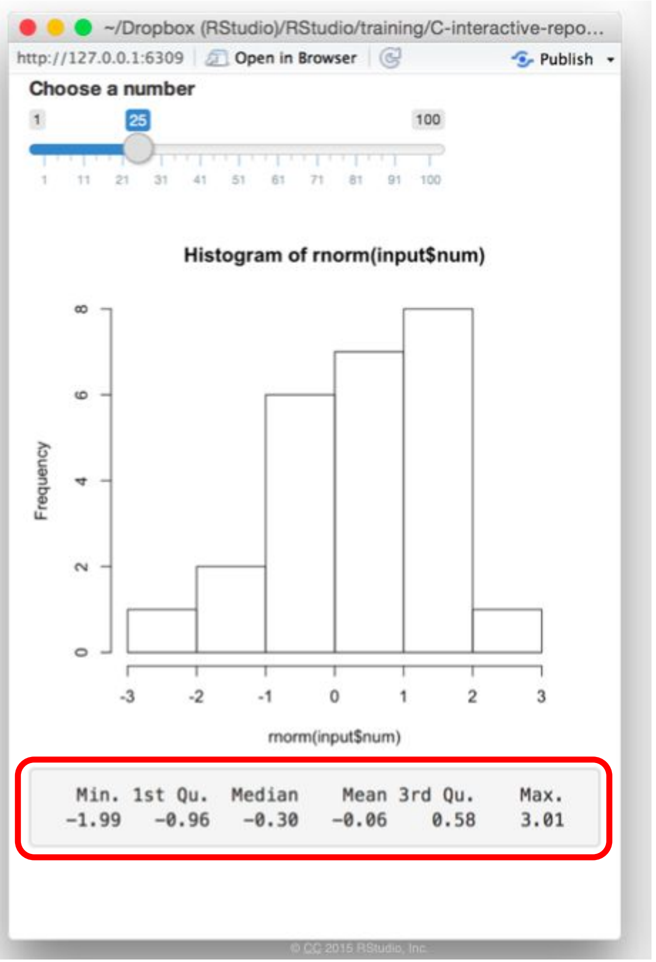
Histogram of Random Normal Values

© CC 2015 RStudio, Inc.

2 OUTPUTS

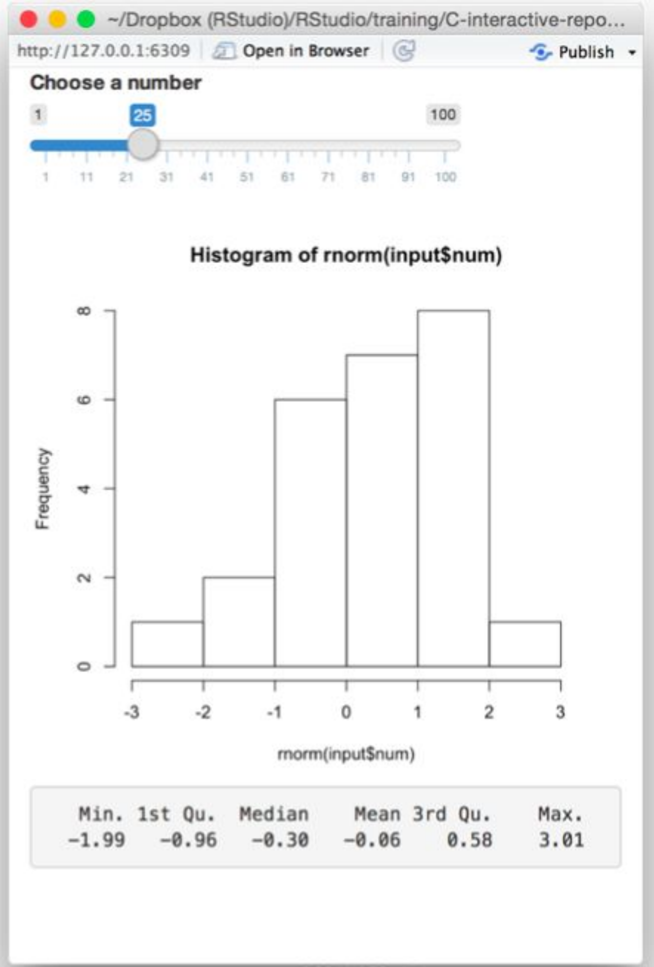
- REACTIVE() -

```r
# 02-two-outputs

library(shiny)

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

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

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



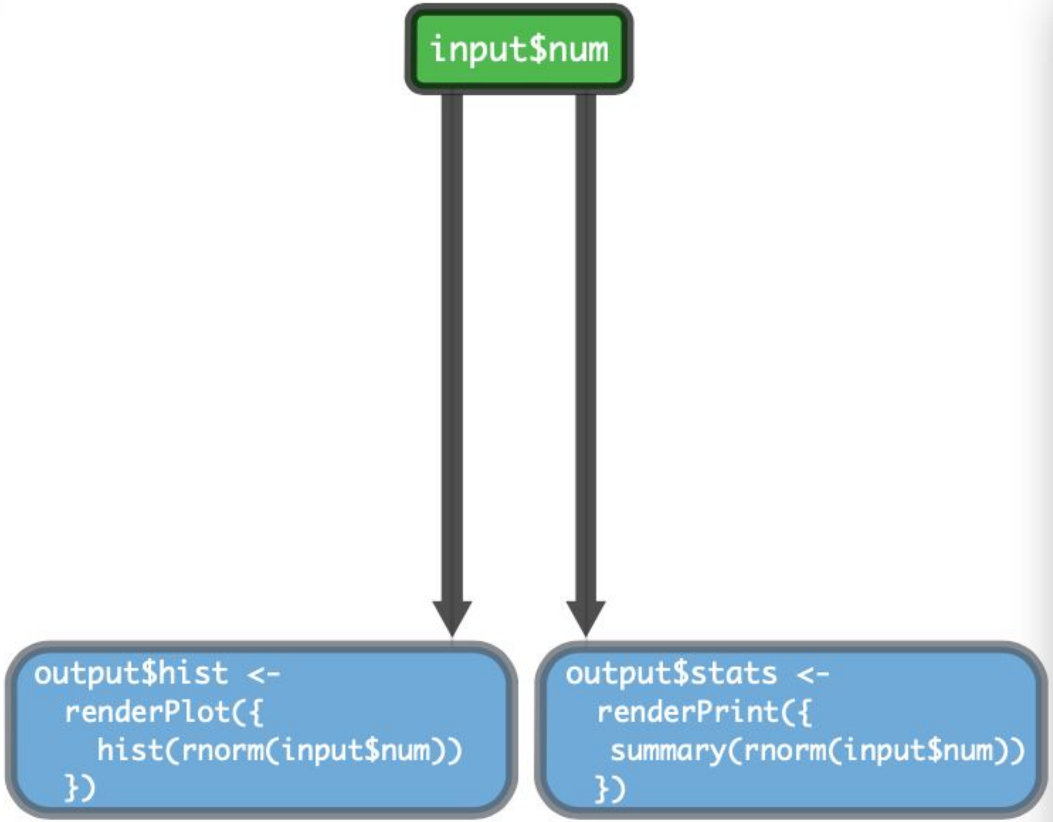
```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  -1.99   -0.96   -0.30   -0.06    0.58    3.01
```

```r
# 02-two-outputs

library(shiny)

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

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

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



```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  -1.99   -0.96   -0.30   -0.06    0.58    3.01
```

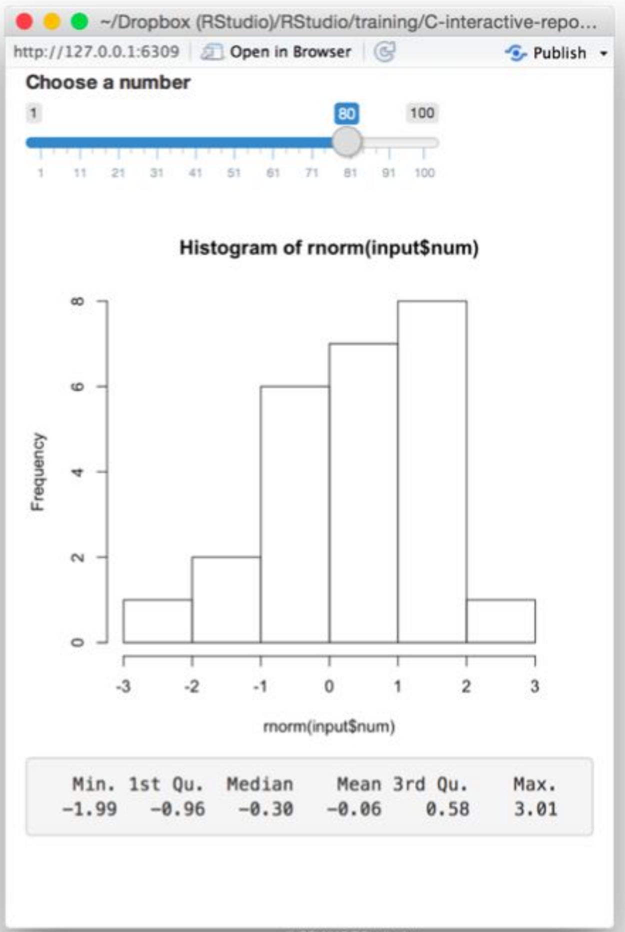
© CC 2015 RStudio, Inc.



```
input$num
```

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

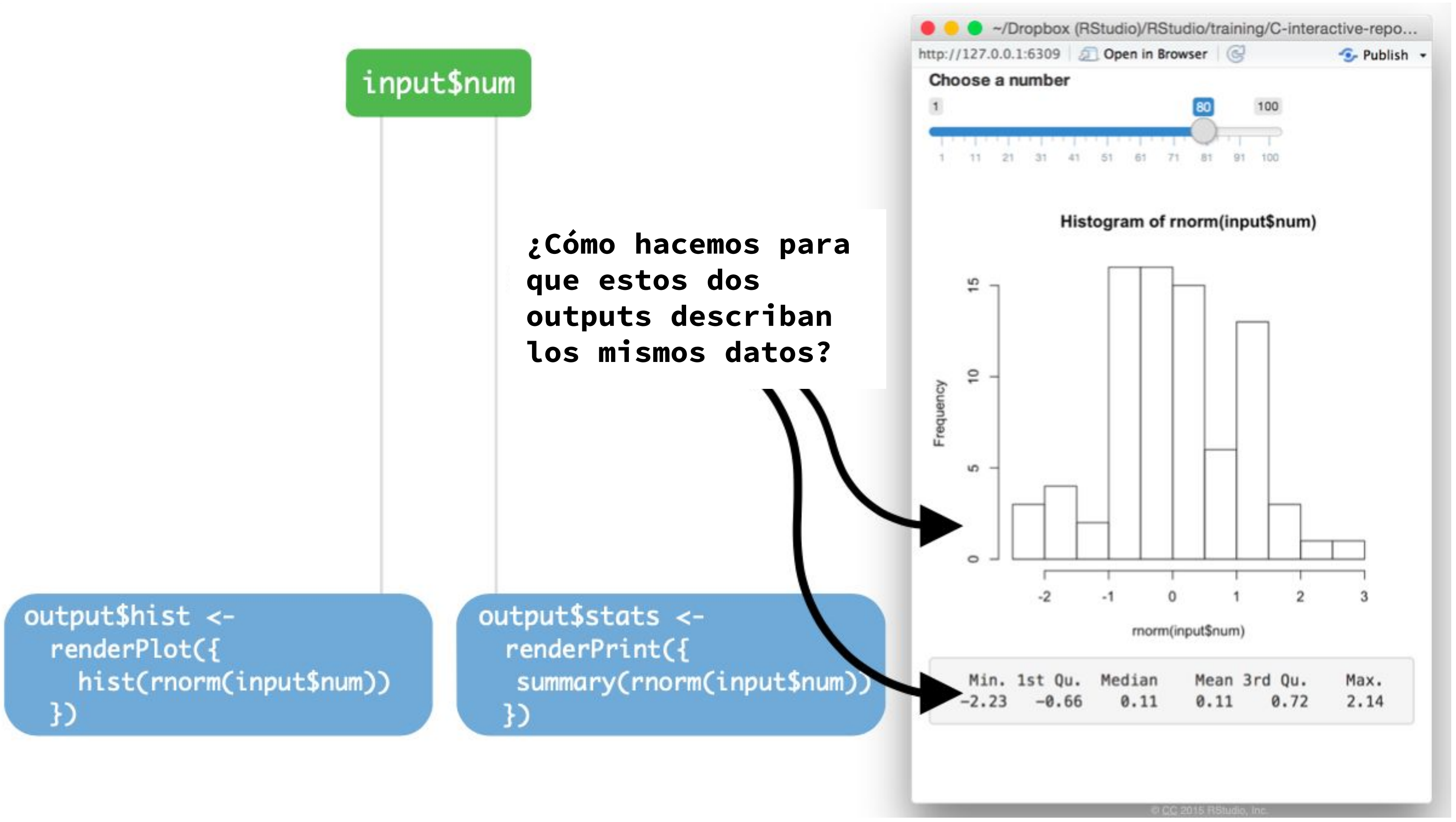
```
output$stats <-
  renderPrint({
   summary(rnorm(input$num))
  })
```



```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  -1.99   -0.96   -0.30   -0.06    0.58    3.01
```

© CC 2015 RStudio, Inc.

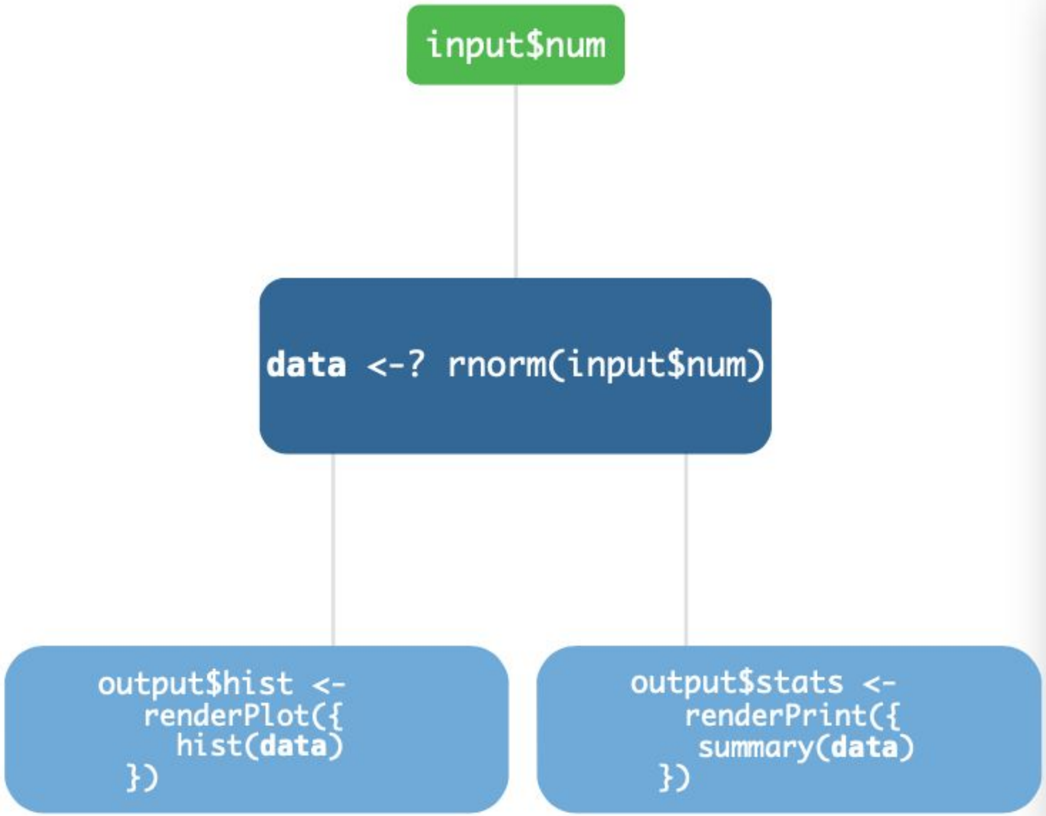
¿Cómo hacemos para que estos dos outputs describan los mismos datos?



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

```r
output$stats <-
  renderPrint({
   summary(rnorm(input$num))
  })
```

```
 Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-2.23   -0.66    0.11    0.11    0.72    2.14
```

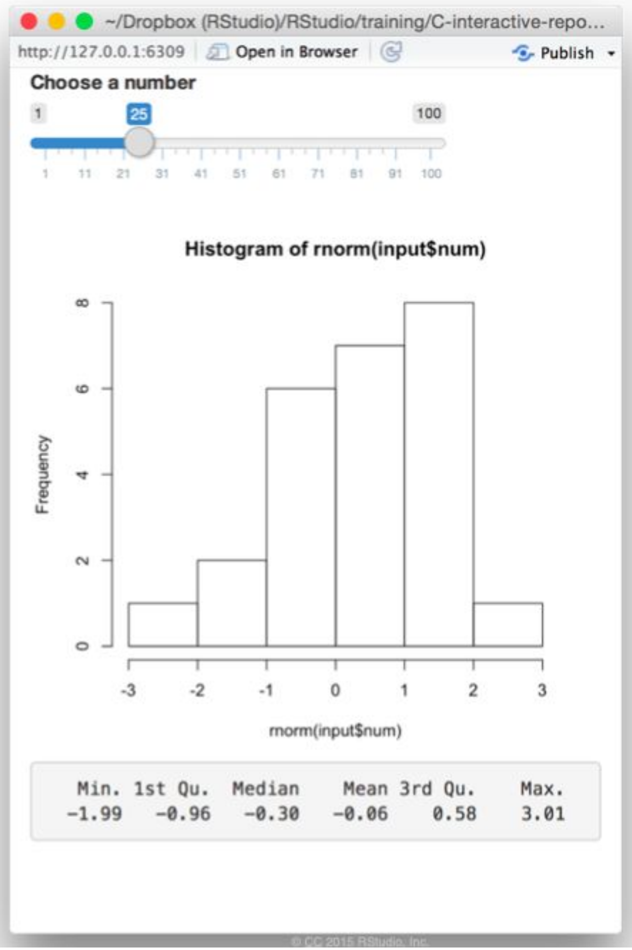


```
input$num
```

```
data <-? rnorm(input$num)
```

```
output$hist <-
  renderPlot({
    hist(data)
 })
```

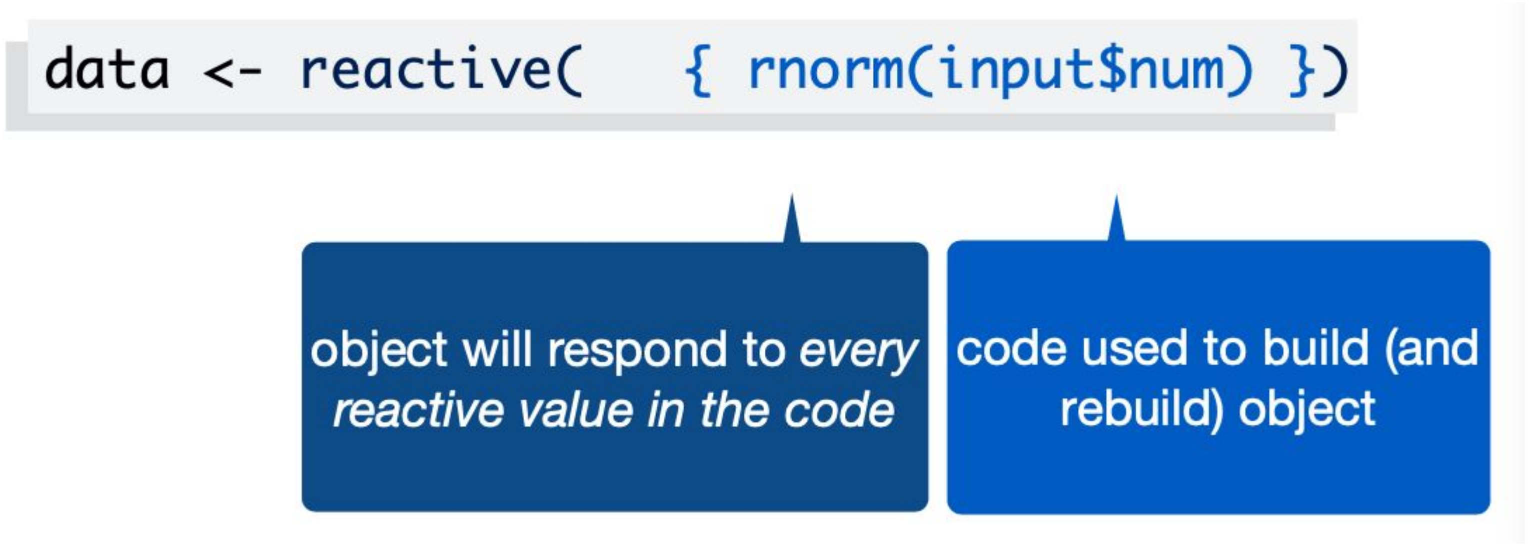
```
output$stats <-
  renderPrint({
   summary(data)
 })
```



```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  -1.99   -0.96   -0.30   -0.06    0.58    3.01
```

© CC 2015 RStudio, Inc.

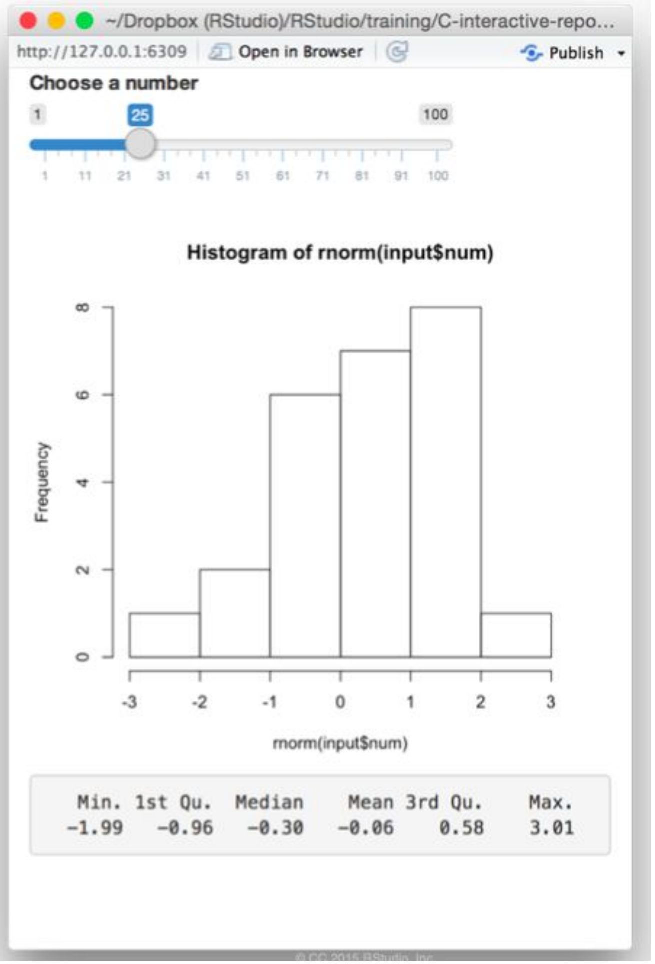
reactive()



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

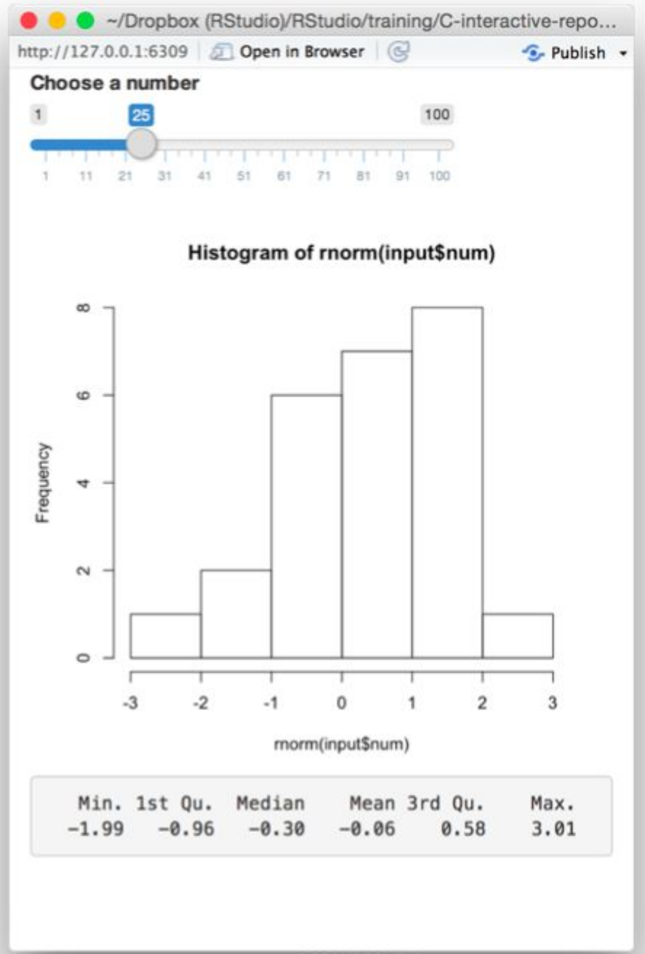
object will respond to *every reactive value in the code*
code used to build (and rebuild) object

```r
# 02-two-outputs

library(shiny)

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

server <- function(input, output) {
  data <- reactive({
    rnorm(input$num)
  })
  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
  output$stats <- renderPrint({
    summary(rnorm(input$num))
  })
}

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



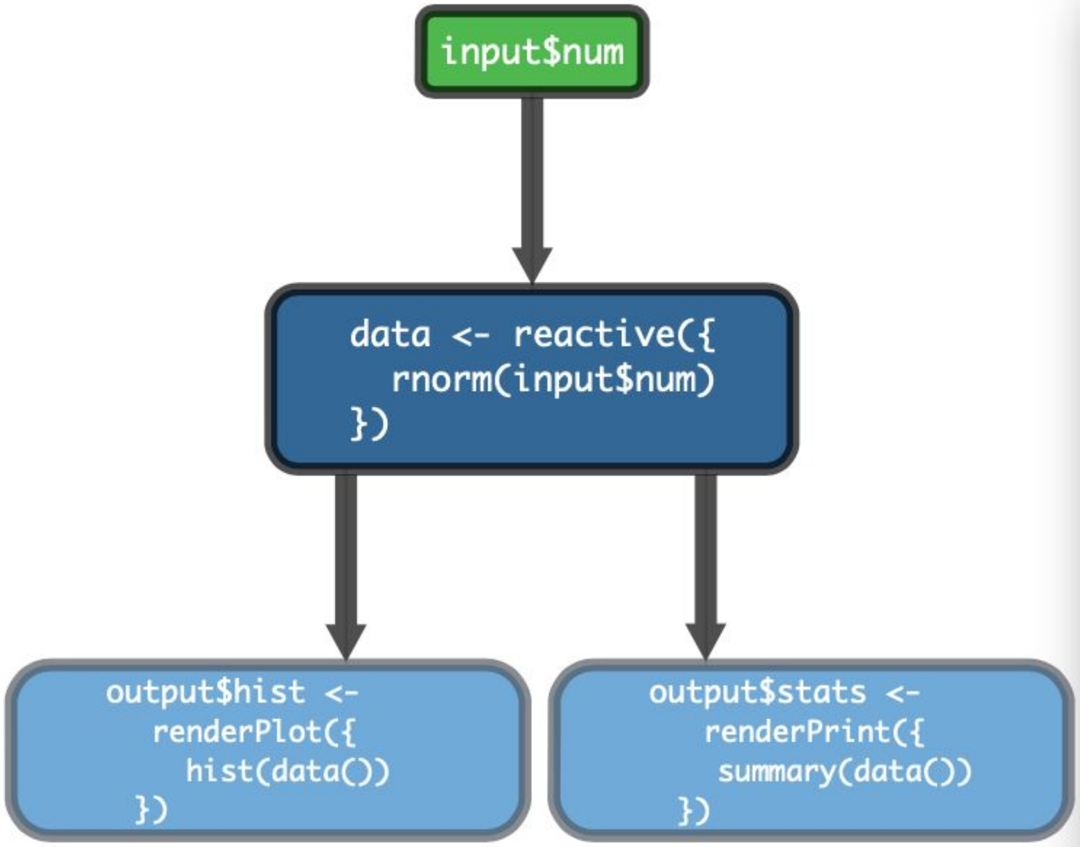
```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  -1.99   -0.96   -0.30   -0.06    0.58    3.01
```

```r
# 03-reactive

library(shiny)

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

server <- function(input, output) {
  data <- reactive({
    rnorm(input$num)
  })
  output$hist <- renderPlot({
    hist(data())
  })
  output$stats <- renderPrint({
    summary(data())
  })
}

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



```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  -1.99   -0.96   -0.30   -0.06    0.58    3.01
```

© CC 2015 RStudio, Inc.

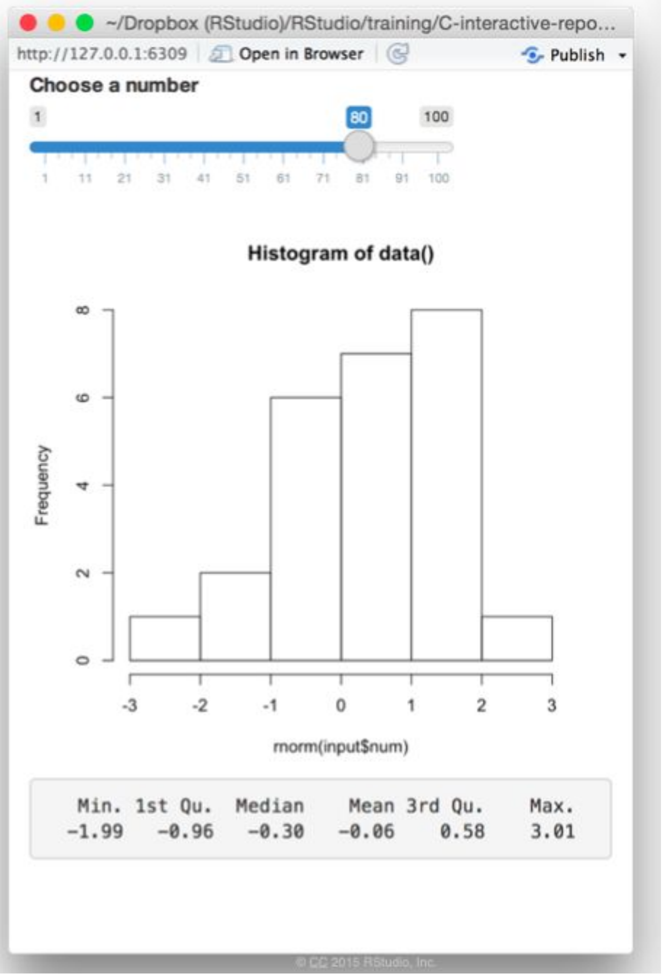


```
input$num
```

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

```
output$hist <-
  renderPlot({
    hist(data())
})
```

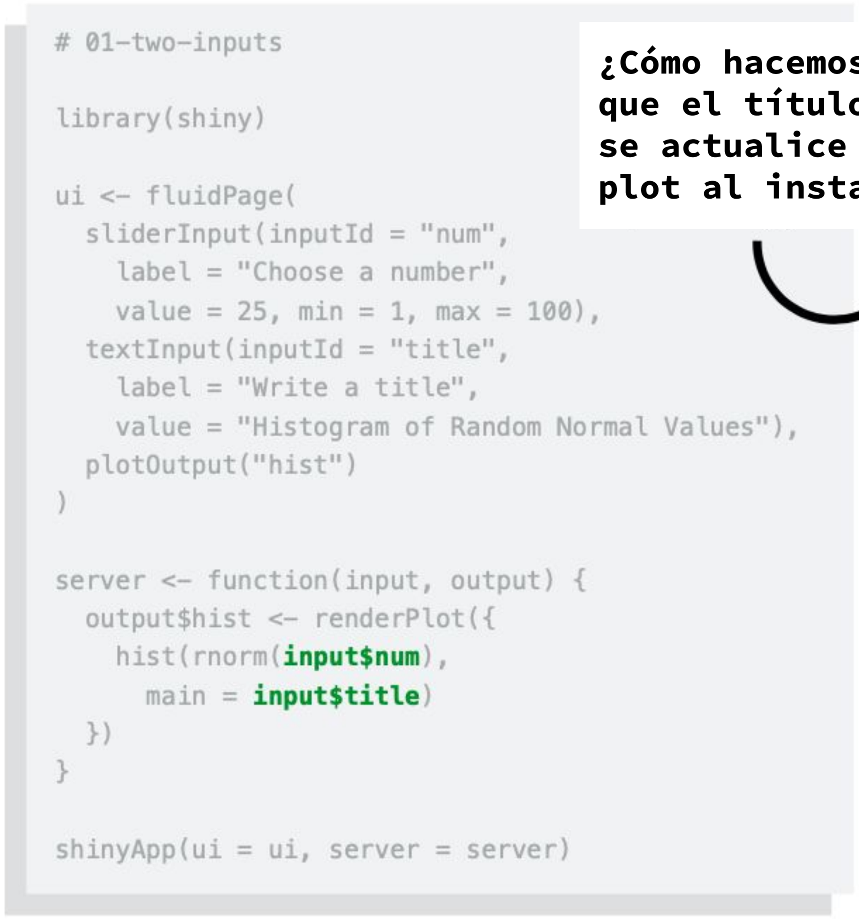
```
output$stats <-
  renderPrint({
   summary(data())
})
```



```
   Min.  1st Qu.  Median    Mean 3rd Qu.    Max.
  -1.99   -0.96   -0.30   -0.06    0.58    3.01
```

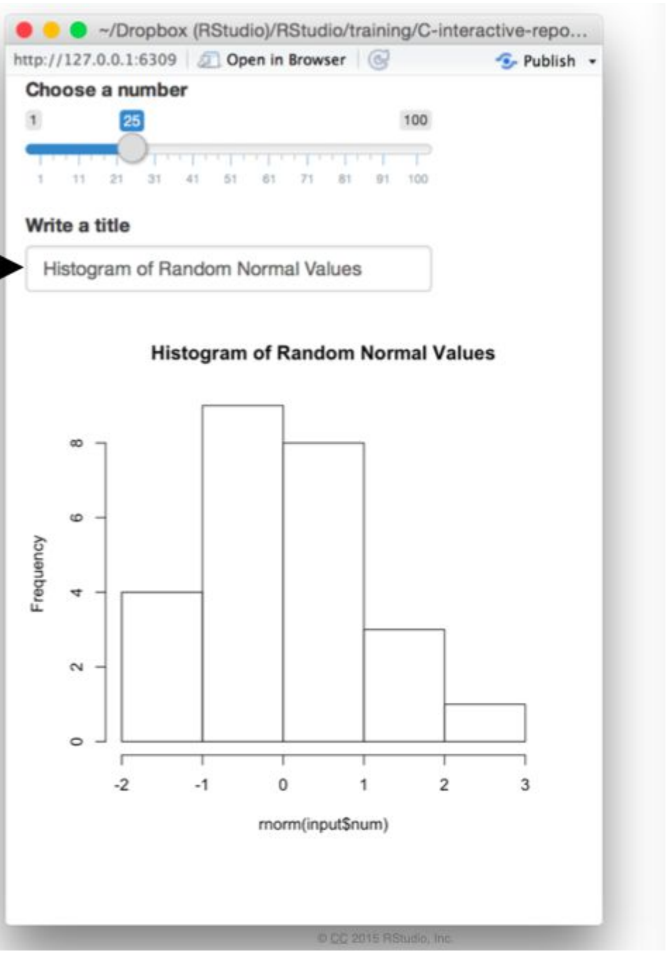
© CC 2015 RStudio, Inc.

Isolate()



```r
# 01-two-inputs

library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  textInput(inputId = "title",
    label = "Write a title",
    value = "Histogram of Random Normal Values"),
  plotOutput("hist")
)

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

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

¿Cómo hacemos para que el título no se actualice en el plot al instante?



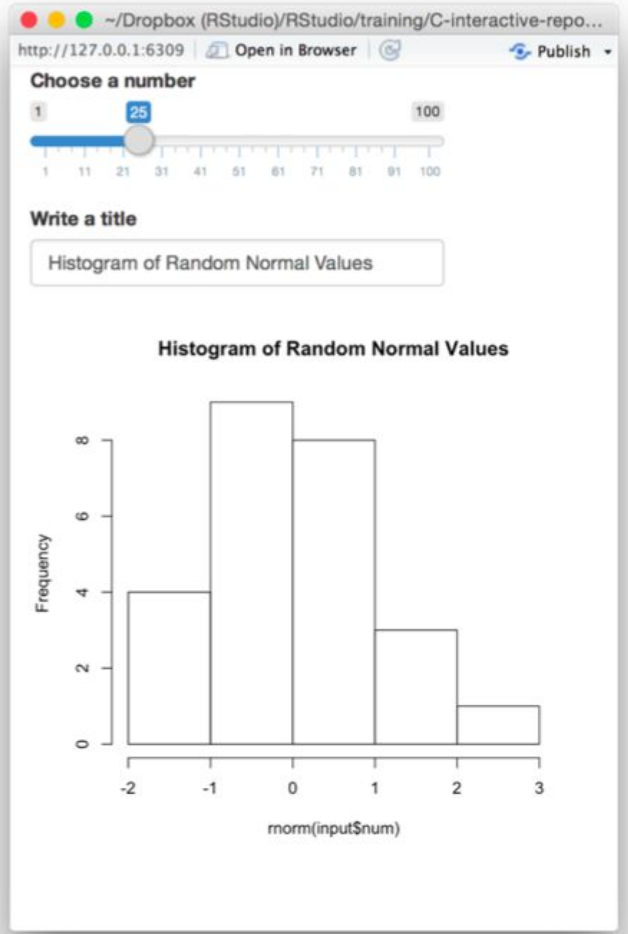
~/Dropbox (RStudio)/RStudio/training/C-interactive-repo...

http://127.0.0.1:6309

Choose a number: 25

Write a title: Histogram of Random Normal Values

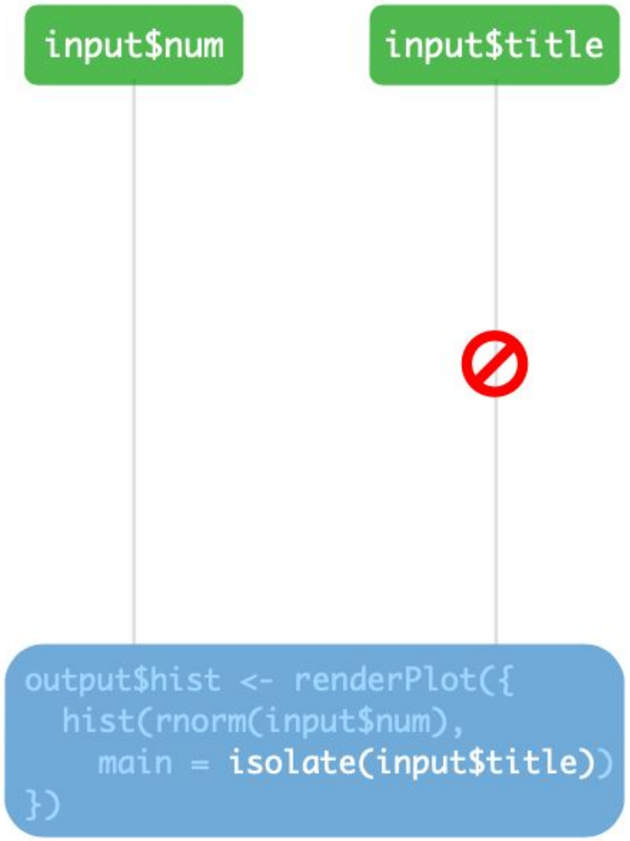
```
# 04-isolate

library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  textInput(inputId = "title",
    label = "Write a title",
    value = "Histogram of Random Normal Values"),
  plotOutput("hist")
)

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

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



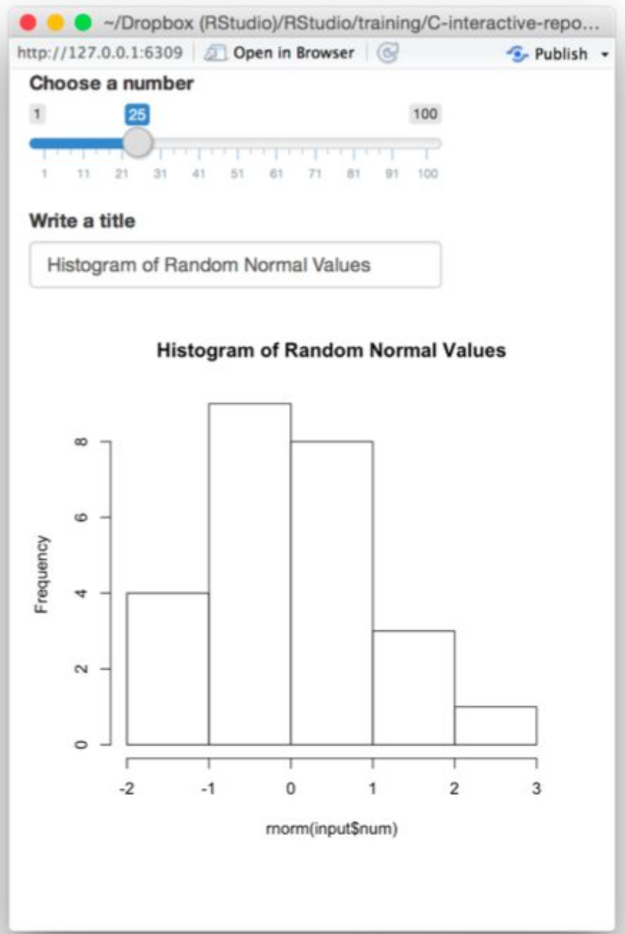
Choose a number: 25

Write a title: Histogram of Random Normal Values

© CC 2015 RStudio, Inc.



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


~/Dropbox (RStudio)/RStudio/training/C-interactive-repo...
http://127.0.0.1:6309
Open in Browser
Publish
Histogram of Random Normal Values
Frequency
rnorm(input$num)

Choose a number: 25 (range 1–100)

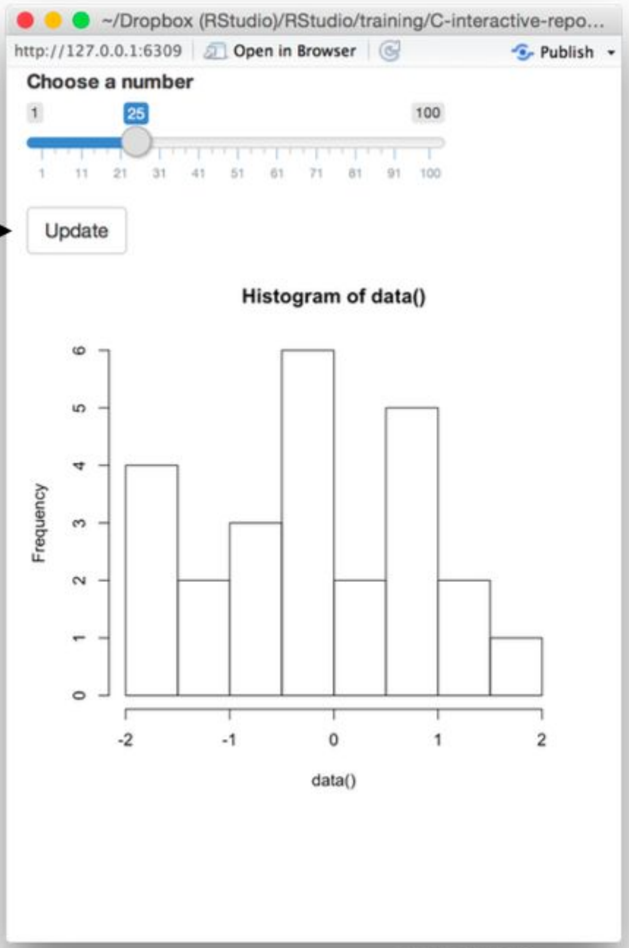
Write a title: Histogram of Random Normal Values

© CC 2015 RStudio, Inc.

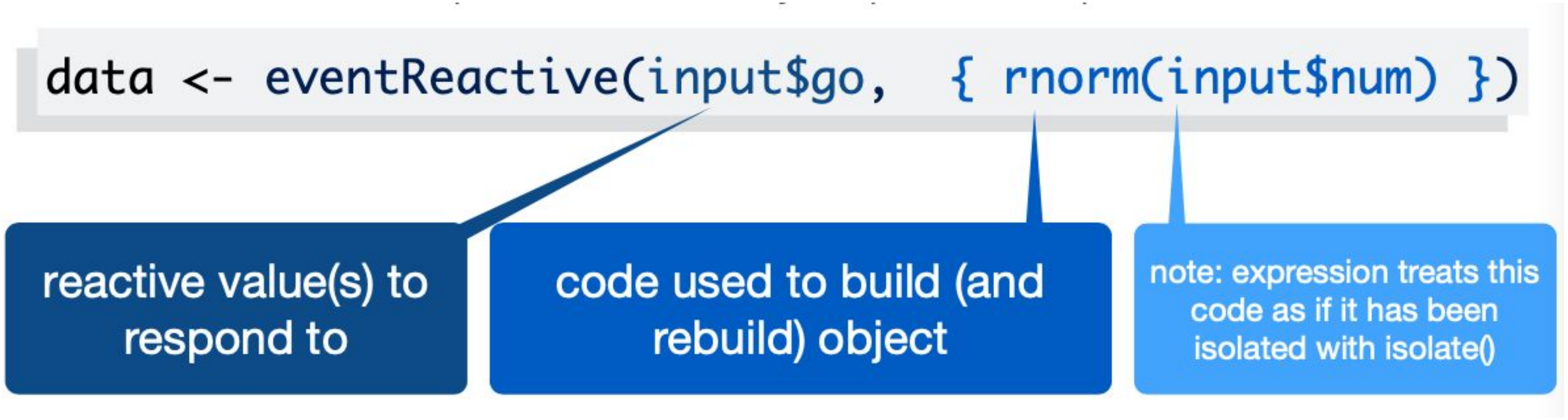
eventReactive()

```r
# 07-eventReactive

library(shiny)

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

server <- function(input, output) {



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

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

¿Cómo hacemos para que el gráfico se actualice sólo cuando apretamos este botón?



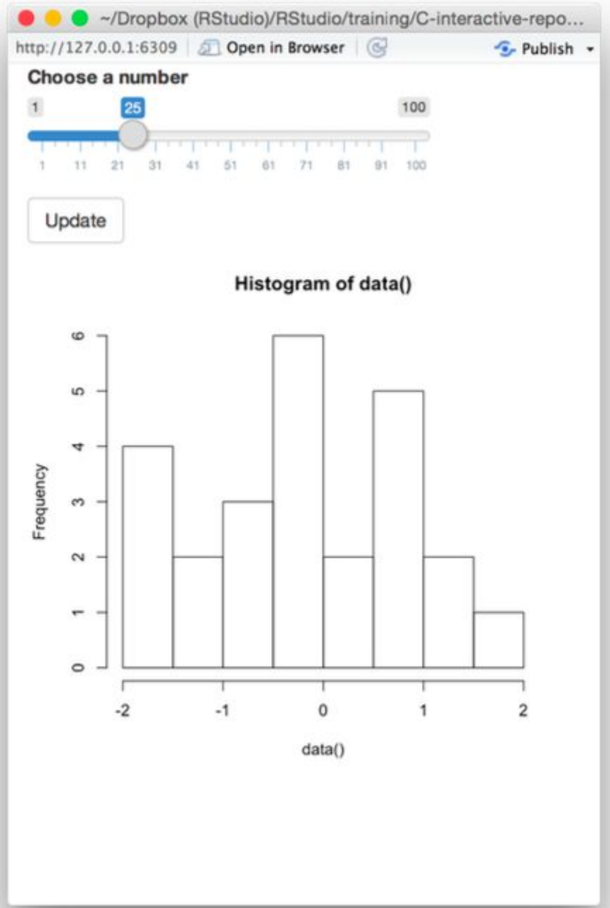
© CC 2015 RStudio, Inc.



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

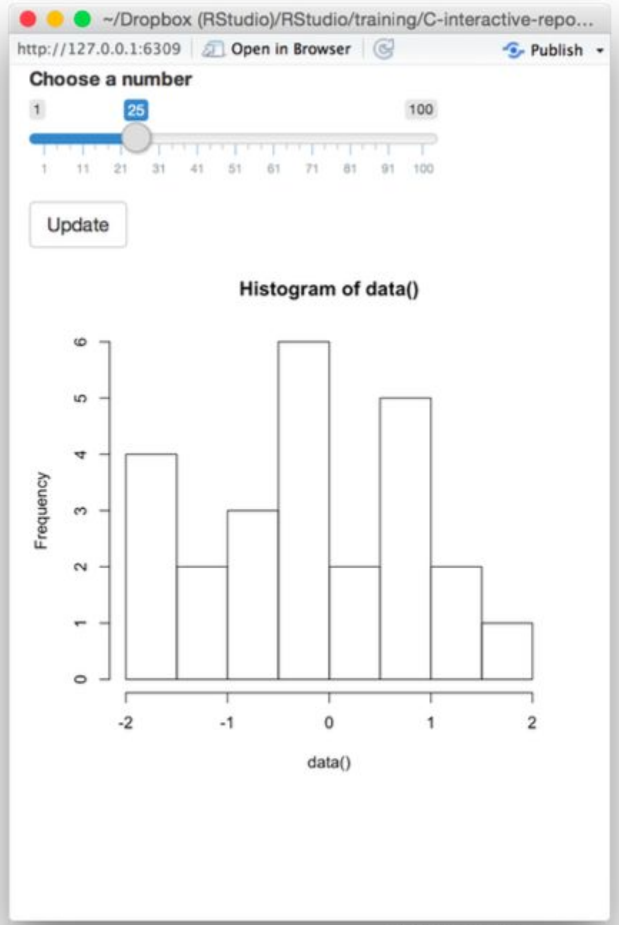
reactive value(s) to respond to
code used to build (and rebuild) object
note: expression treats this code as if it has been isolated with isolate()

```r
# 07-eventReactive

library(shiny)

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

server <- function(input, output) {



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

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



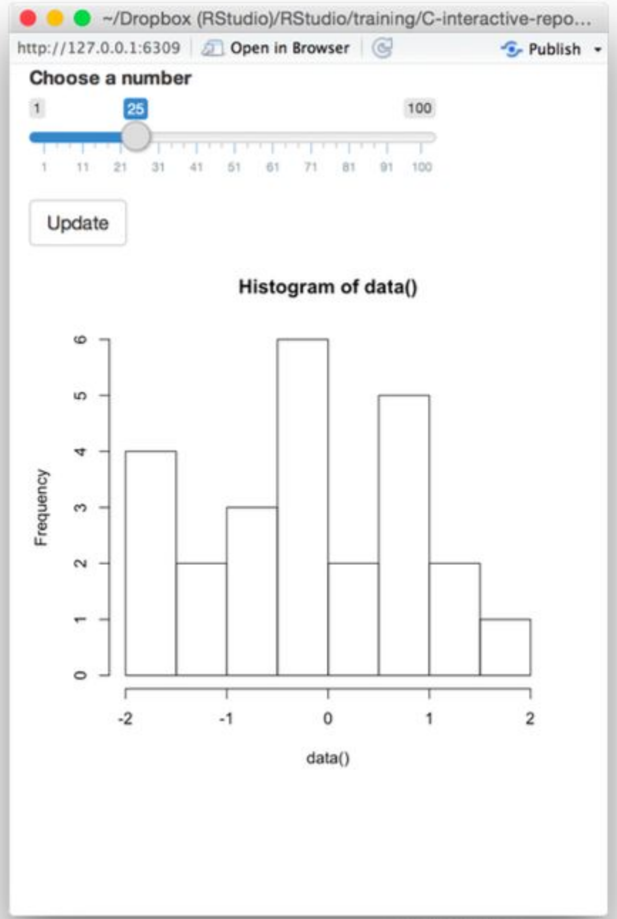
© CC 2015 RStudio, Inc.

```r
# 07-eventReactive

library(shiny)

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

server <- function(input, output) {
  data <- eventReactive(input$go, {

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

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



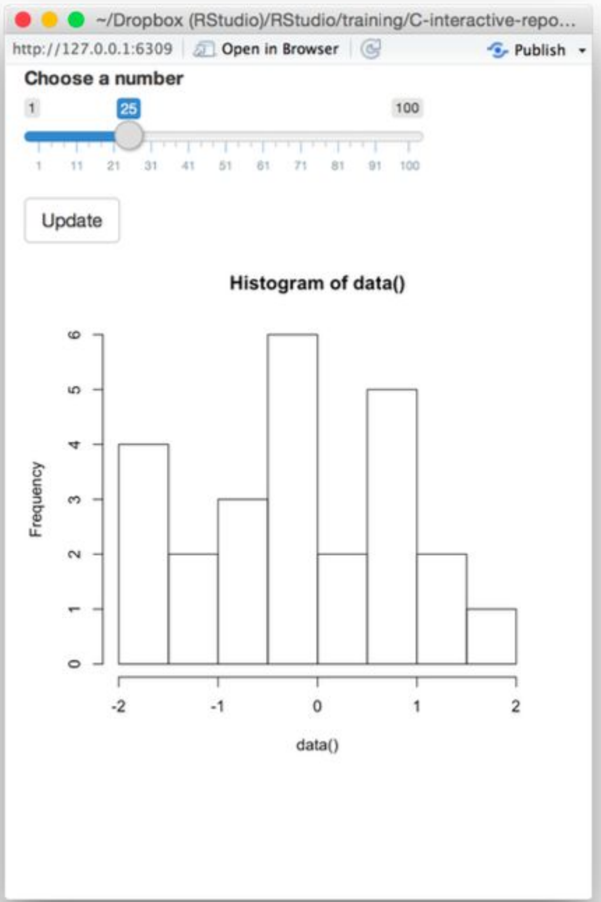
© CC 2015 RStudio, Inc.

```r
# 07-eventReactive

library(shiny)

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

server <- function(input, output) {
  data <- eventReactive(input$go, {

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

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



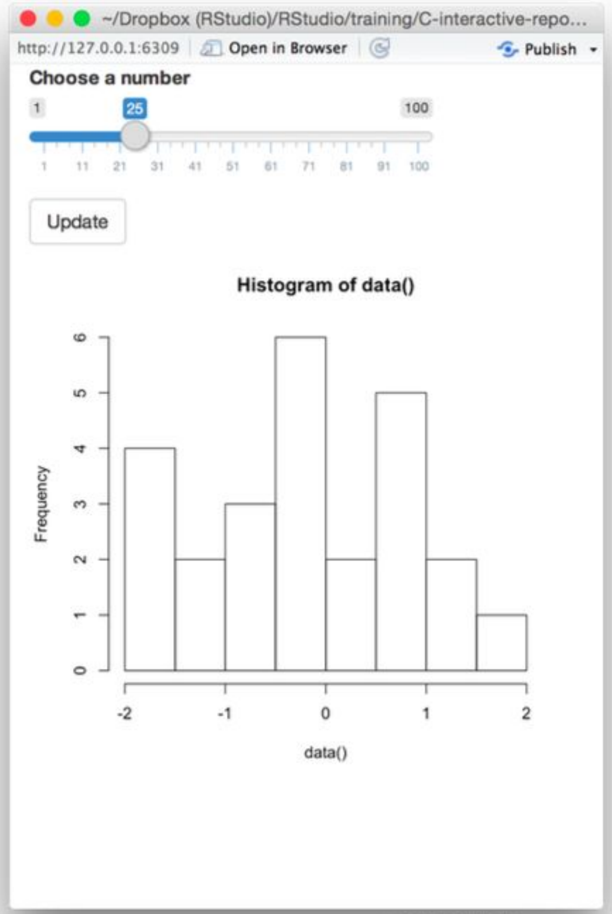
© CC 2015 RStudio, Inc.

```r
# 07-eventReactive

library(shiny)

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

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

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



© CC 2015 RStudio, Inc.

```r
# 07-eventReactive

library(shiny)

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

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

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



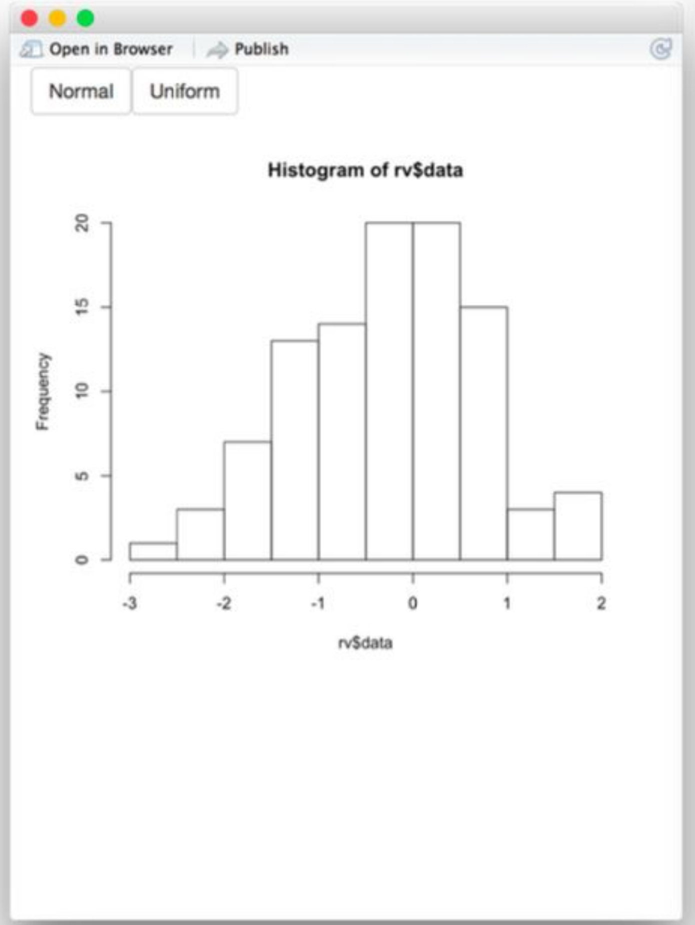
© CC 2015 RStudio, Inc.

reactiveValues()

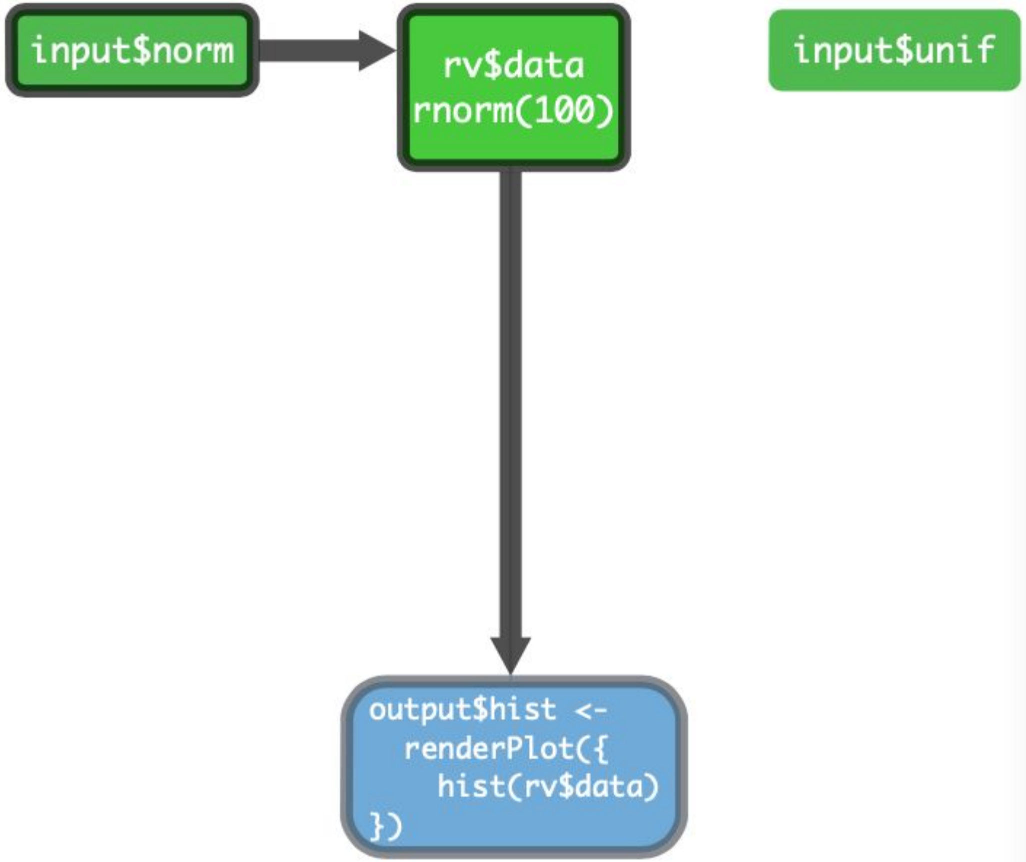
```r
rv <- reactiveValues(data = rnorm(100))
```

(optional) elements to add to the list

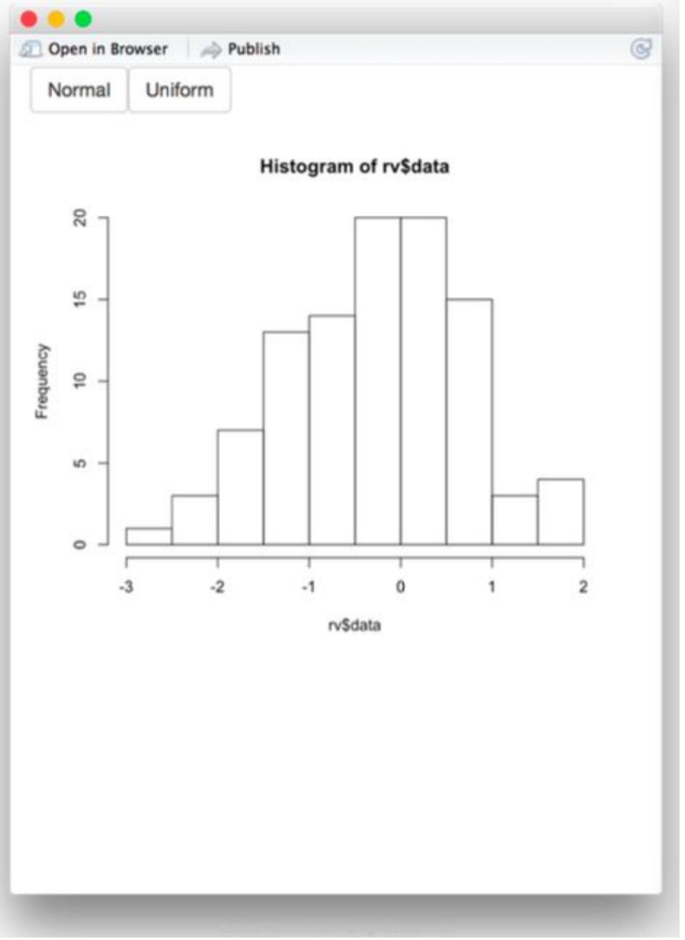
```
# 08-reactiveValues

library(shiny)

ui <- fluidPage(
  actionButton(inputId = "norm", label = "Normal"),
  actionButton(inputId = "unif", label = "Uniform"),
  plotOutput("hist")
)

server <- function(input, output) {

  rv <- reactiveValues(data = rnorm(100))

  observeEvent(input$norm, { rv$data <- rnorm(100) })
  observeEvent(input$unif, { rv$data <- runif(100) })

  output$hist <- renderPlot({
    hist(rv$data)
  })
}

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

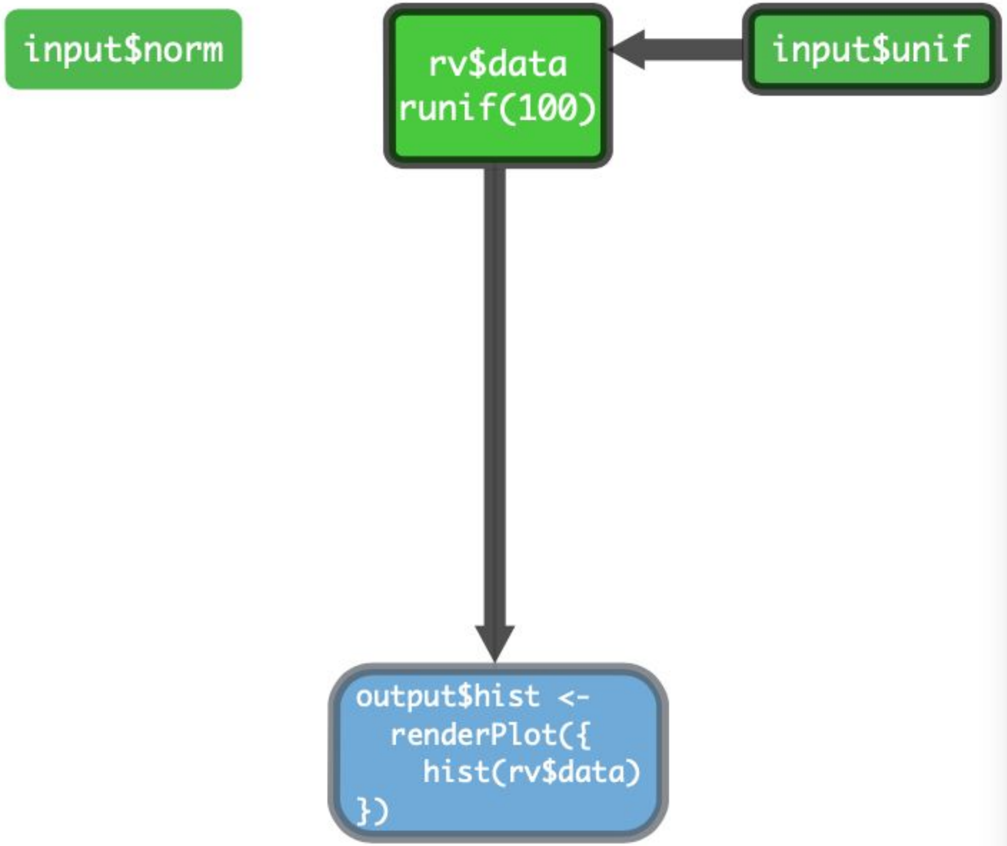


© CC 2015 RStudio, Inc.

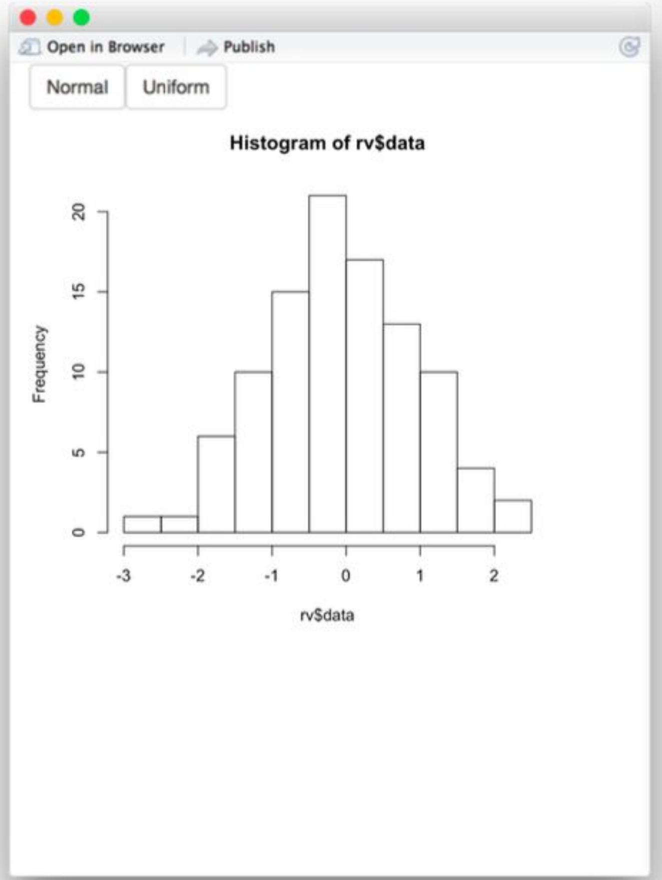


```
output$hist <-
  renderPlot({
    hist(rv$data)
})
```

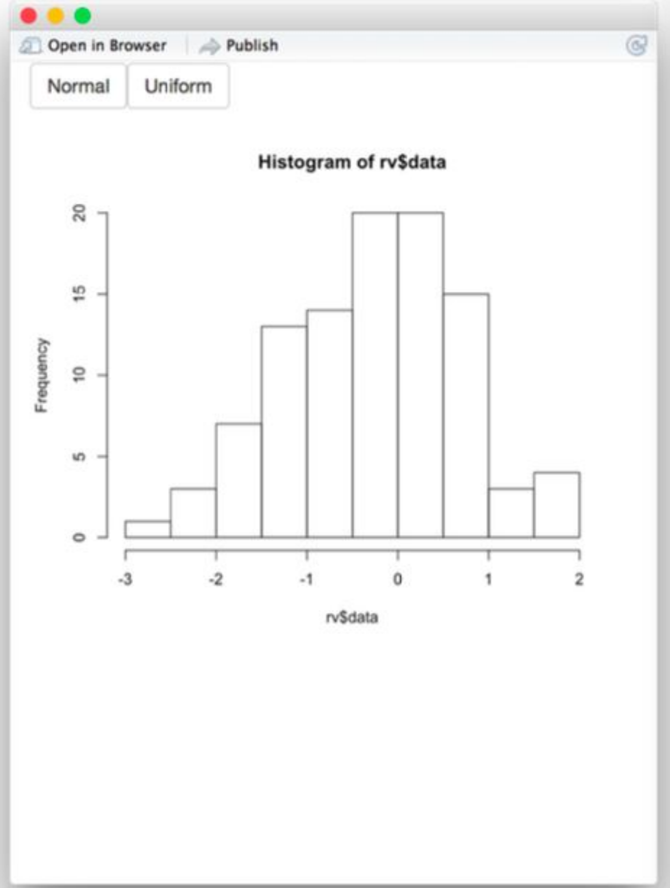




```
output$hist <-
  renderPlot({
    hist(rv$data)
})
```

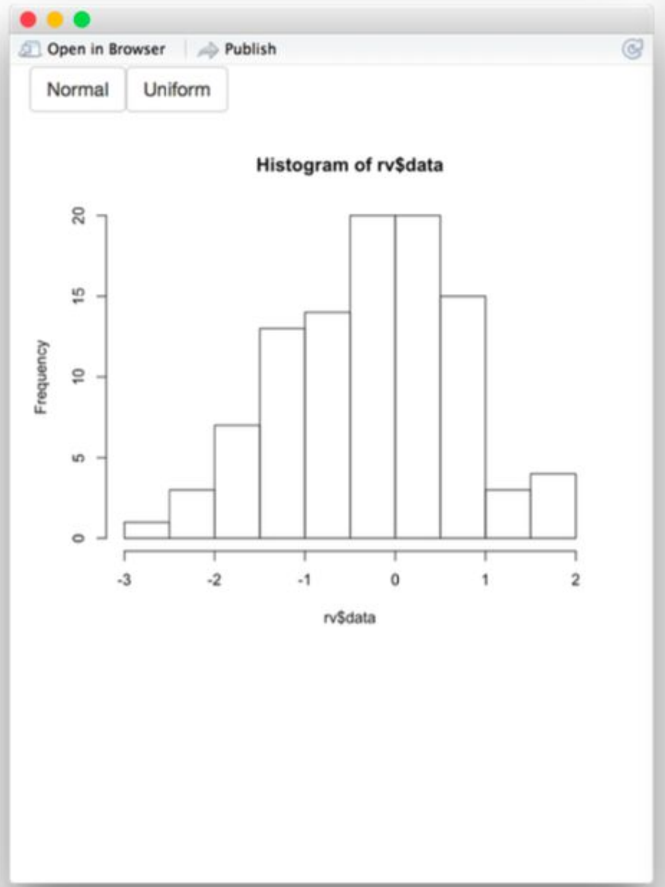


```
# 08-reactiveValues

library(shiny)

ui <- fluidPage(
  actionButton(inputId = "norm", label = "Normal"),
  actionButton(inputId = "unif", label = "Uniform"),
  plotOutput("hist")
)

server <- function(input, output) {

  rv <- reactiveValues(data = rnorm(100))

  observeEvent(input$norm, { rv$data <- rnorm(100) })
  observeEvent(input$unif, { rv$data <- runif(100) })

  output$hist <- renderPlot({
    hist(rv$data)
  })
}

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



© CC 2015 RStudio, Inc.

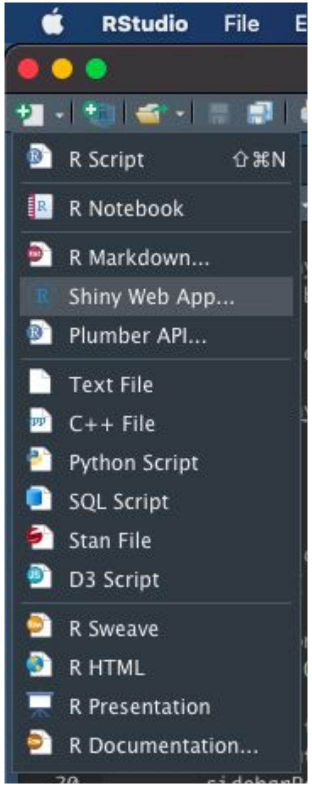
```
# 08-reactiveValues

library(shiny)

ui <- fluidPage(
  actionButton(inputId = "norm", label = "Normal"),
  actionButton(inputId = "unif", label = "Uniform"),
  plotOutput("hist")
)

server <- function(input, output) {

  rv <- reactiveValues(data = rnorm(100))

  observeEvent(input$norm, { rv$data <- rnorm(100) })
  observeEvent(input$unif, { rv$data <- runif(100) })

  output$hist <- renderPlot({
    hist(rv$data)
  })
}

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



© CC 2015 RStudio, Inc.

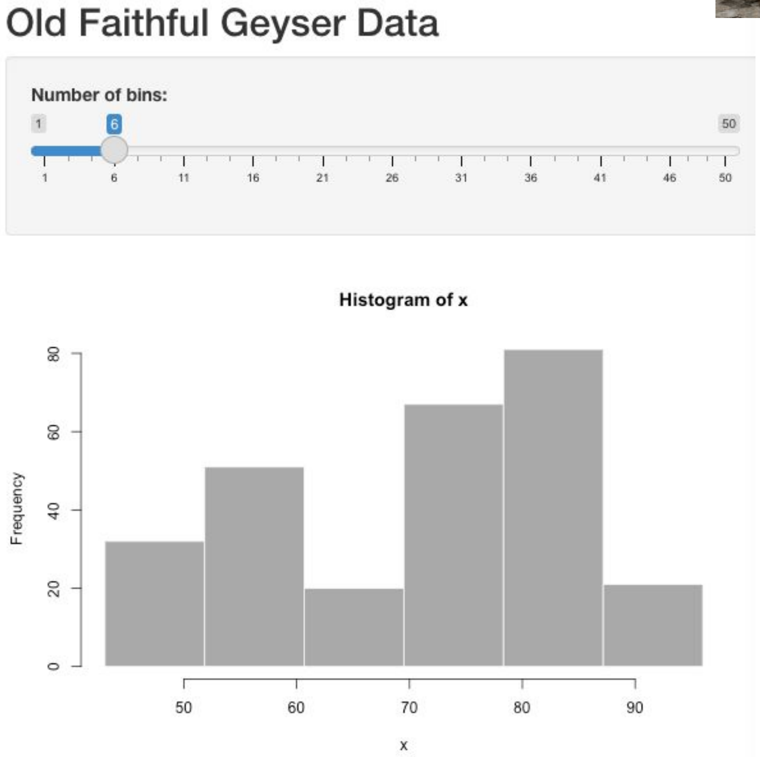
Ejercicio: modificar para que tenga 2 inputs y 2 outputs

Empezar con un template



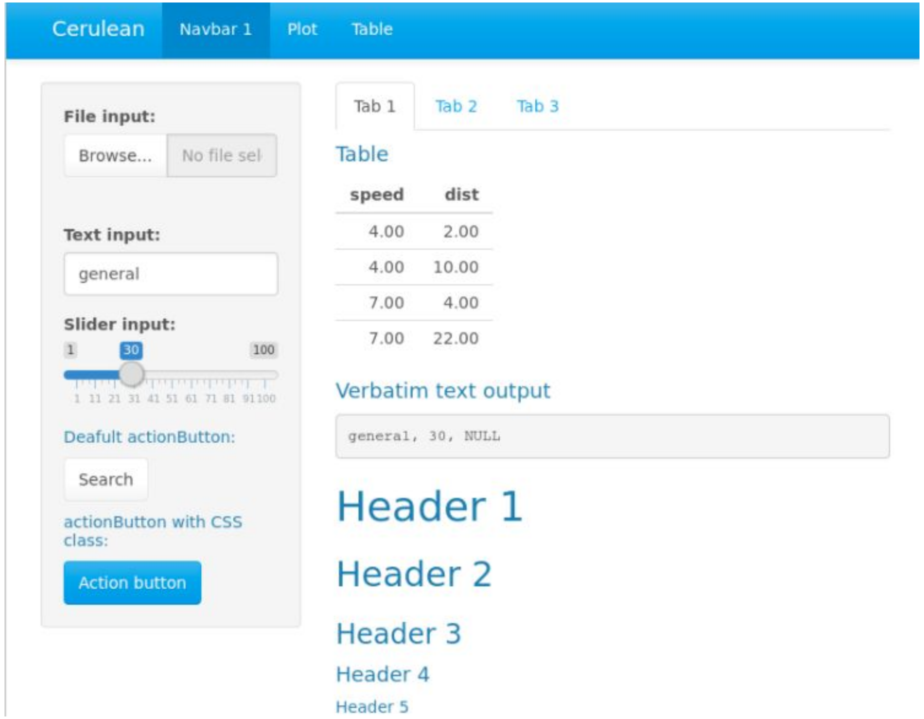
inputs

outputs



¿Qué más? ¿Dónde seguir aprendiendo?

SHINYTHEMES



Table

speed	dist
4.00	2.00
4.00	10.00
7.00	4.00
7.00	22.00

Verbatim text output

```
general, 30, NULL
```

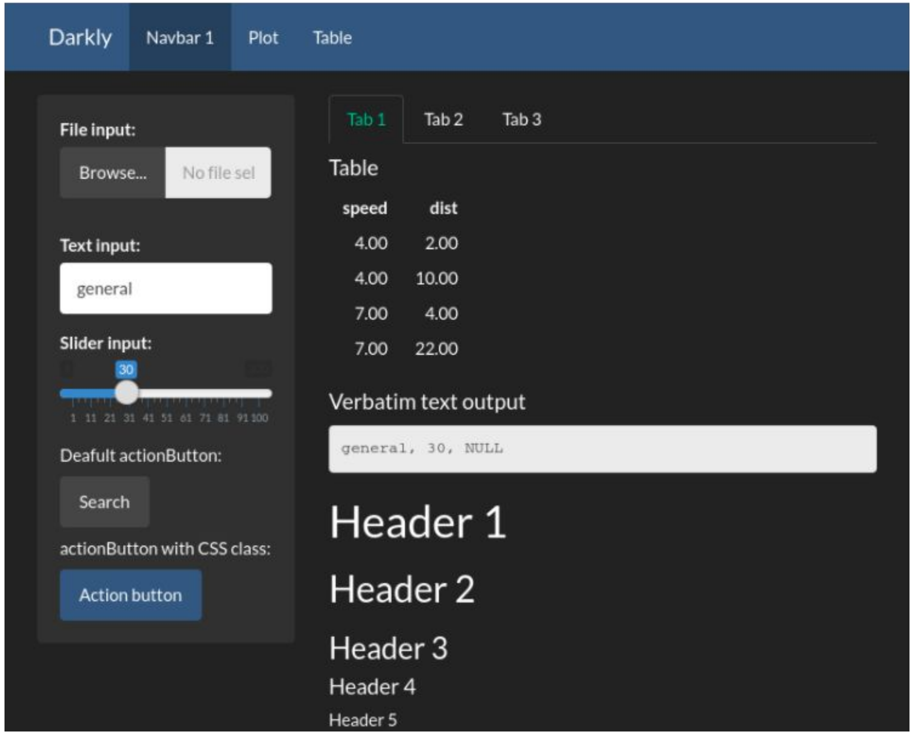
Header 1

Header 2

Header 3

Header 4

Header 5



Table

speed	dist
4.00	2.00
4.00	10.00
7.00	4.00
7.00	22.00

Verbatim text output

```
general, 30, NULL
```

Header 1

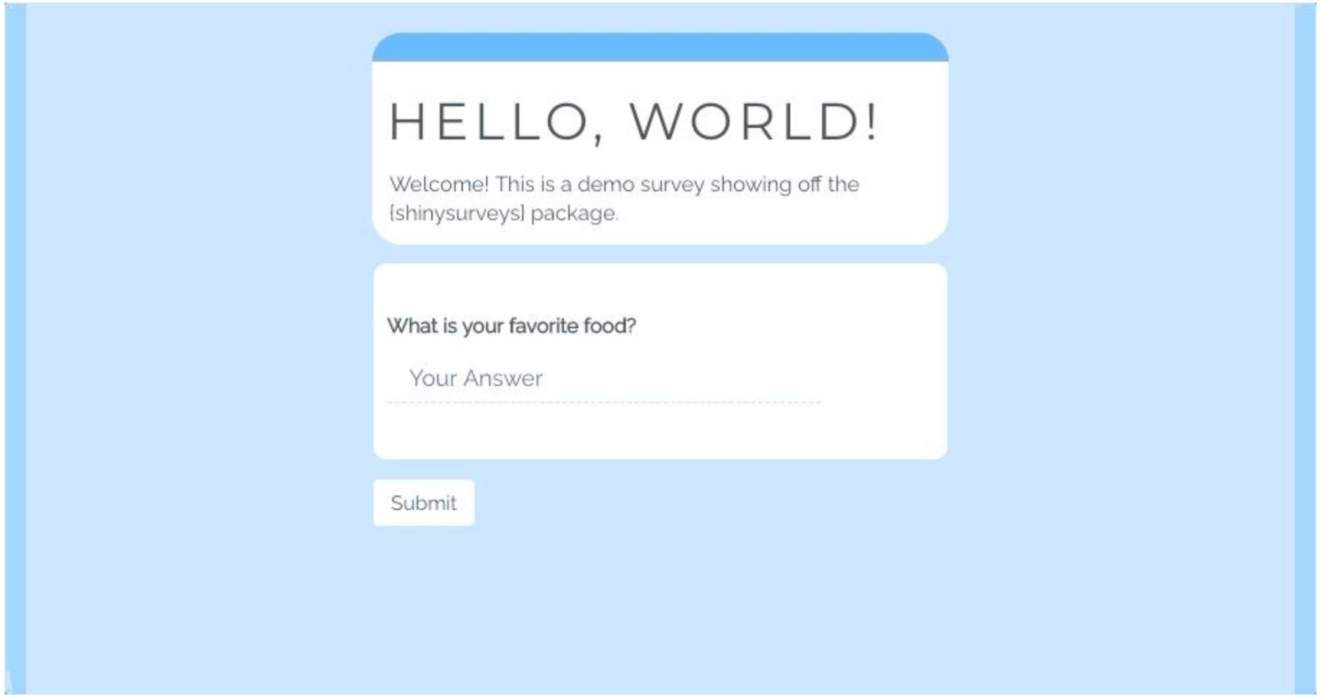
Header 2

Header 3

Header 4

Header 5

SHINYSURVEYS

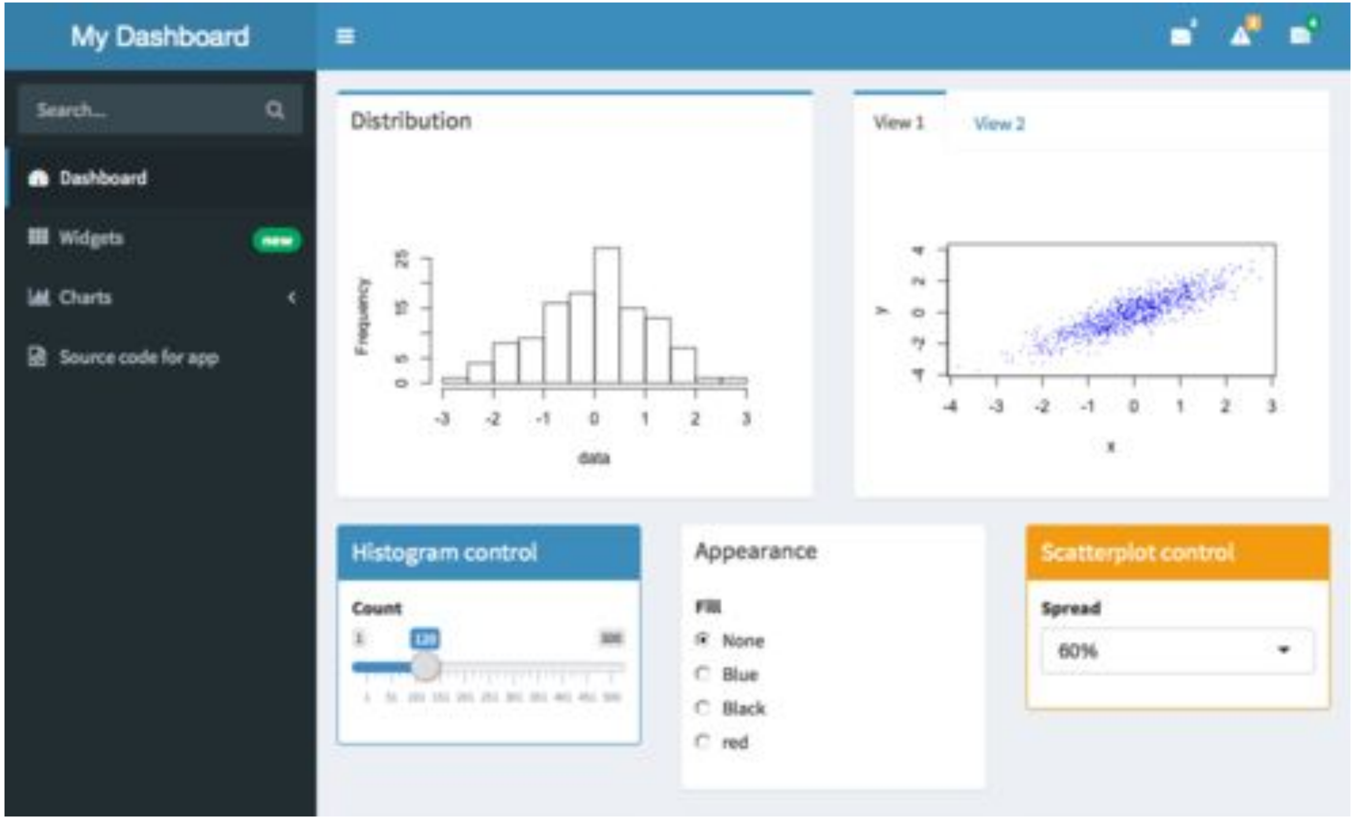


HELLO, WORLD!

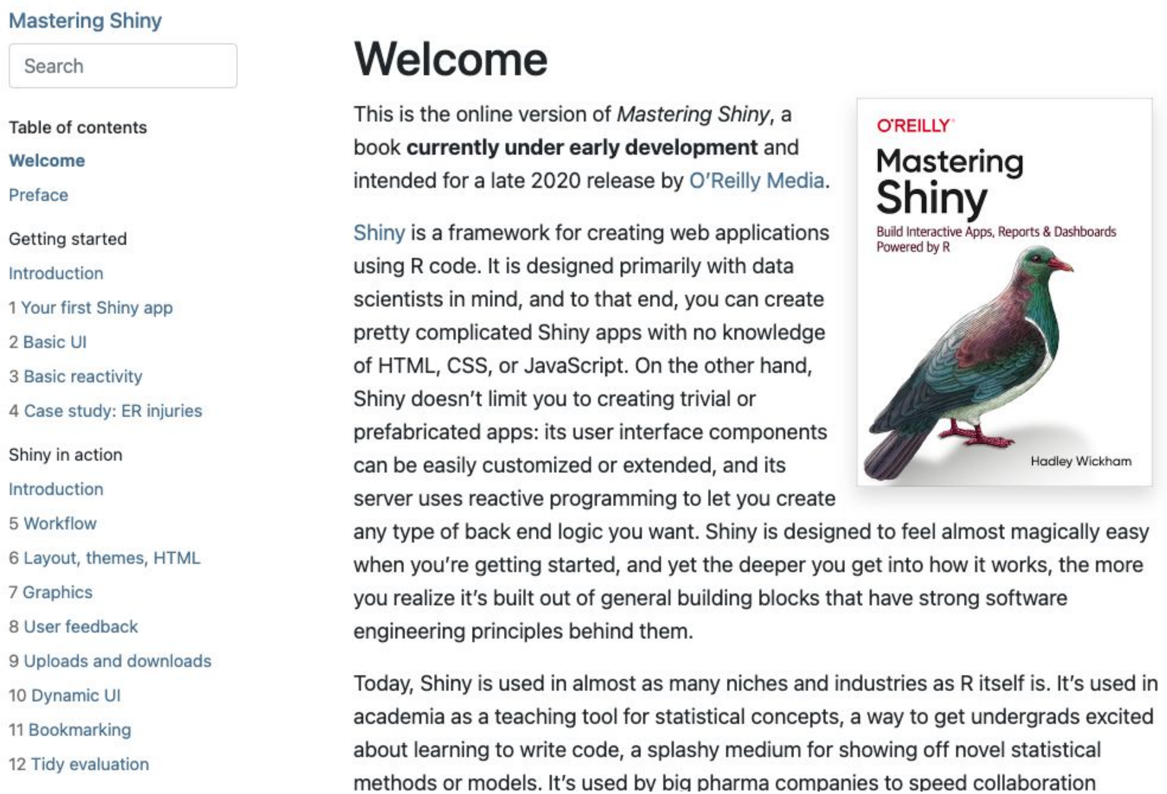
Welcome! This is a demo survey showing off the {shinysurveys} package.

What is your favorite food?

SHINYDASHBOARDS



Mastering Shiny

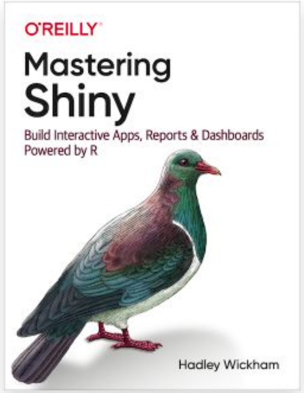


Mastering Shiny

Search

Welcome

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



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

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