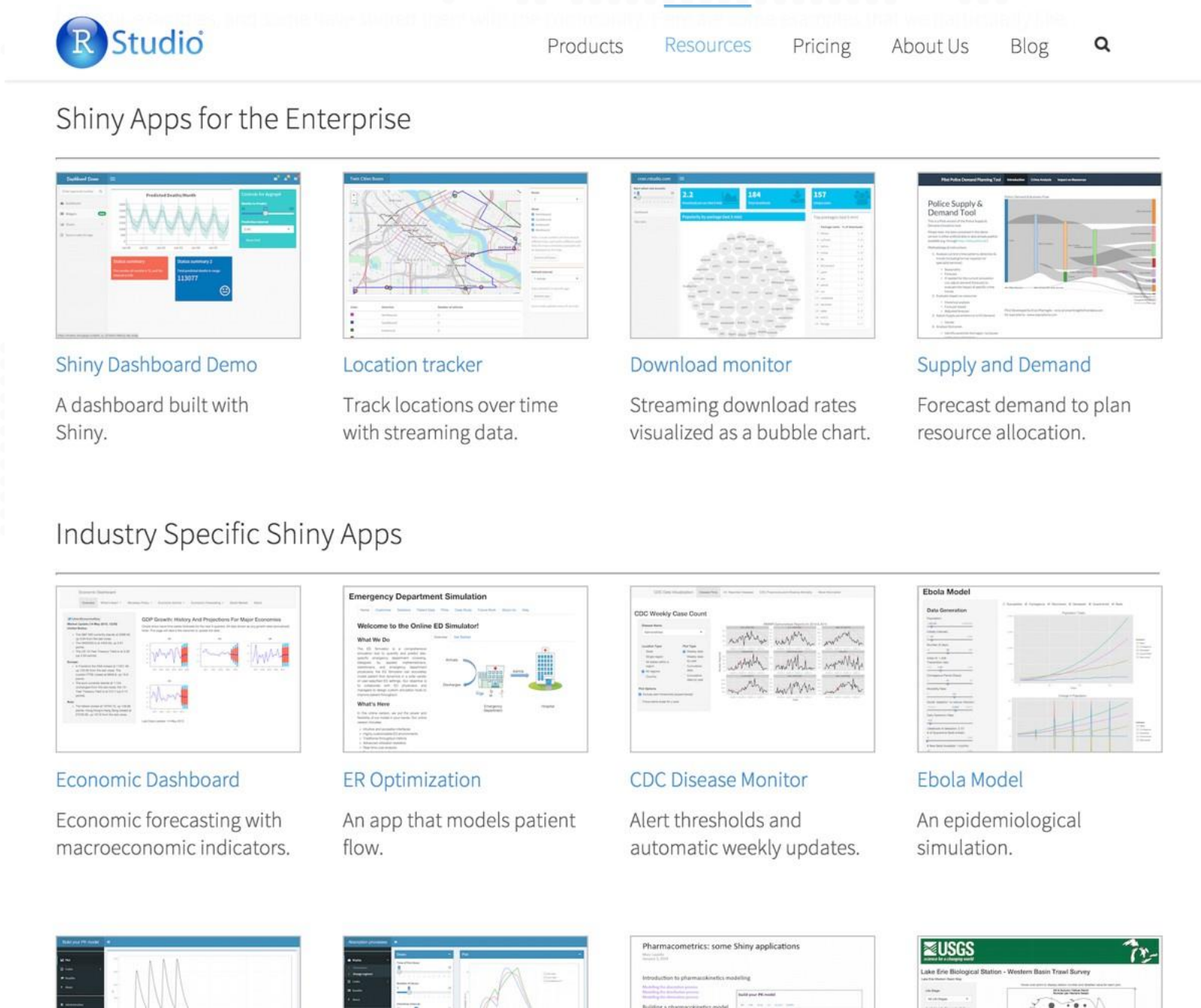
NobleProg

Introduction to Shiny

The World's Local Training Provider

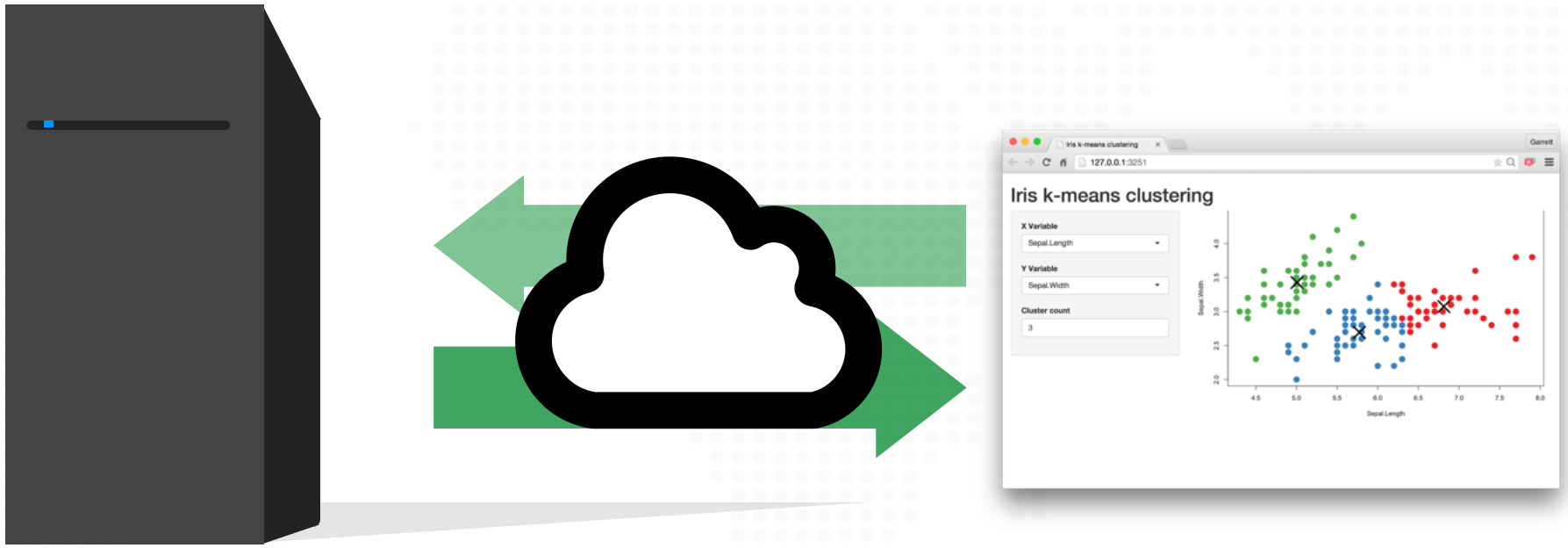
Shiny Showcase

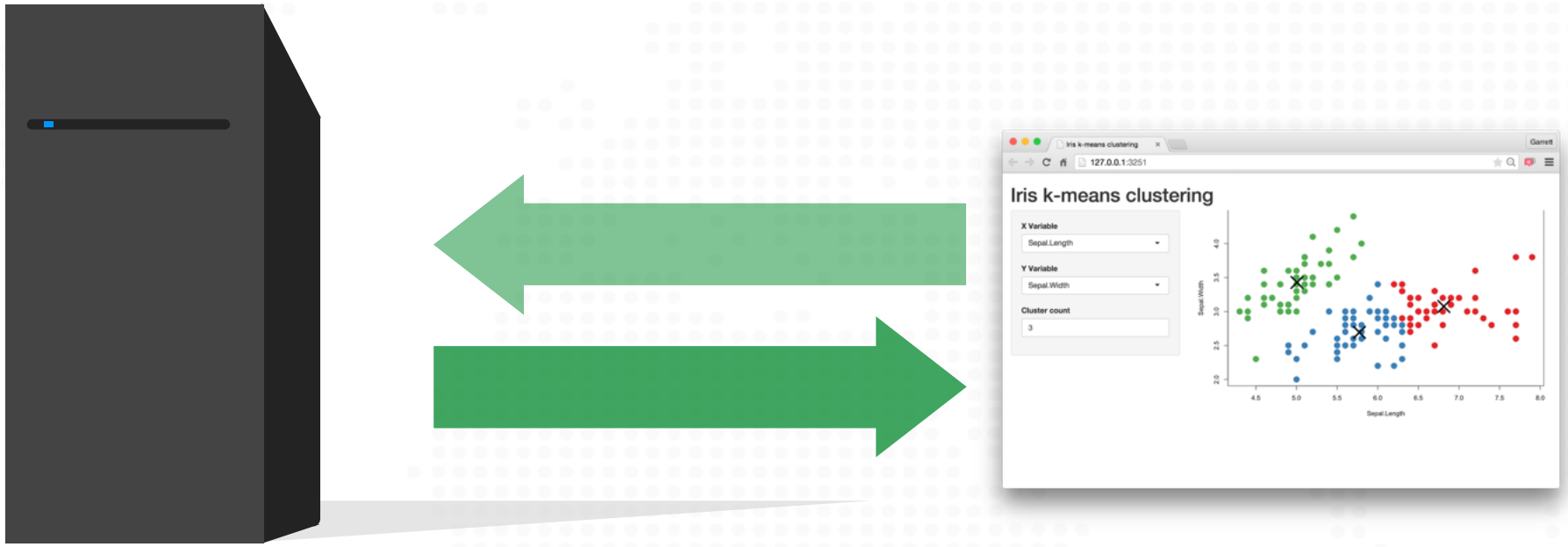
www.rstudio.com/products/shiny/shiny-user-showcase/



Every Shiny app is maintained by a computer running R

Every Shiny app is maintained by a computer running R





Server Instructions

User Interface (UI)

Minimal example

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

server <- function(input, output) {}

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


NobleProg

~/Desktop/my-shiny - RStudio

ui.R | server.R

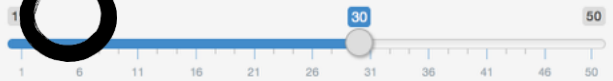
```
# This is the server logic for a Shiny web application.
# You can find out more about building applications with
Shiny here:
#
# http://shiny.rstudio.com
#

library(shiny)

shinyServer(function(input, output) {

  output$distPlot <- renderPlot({

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

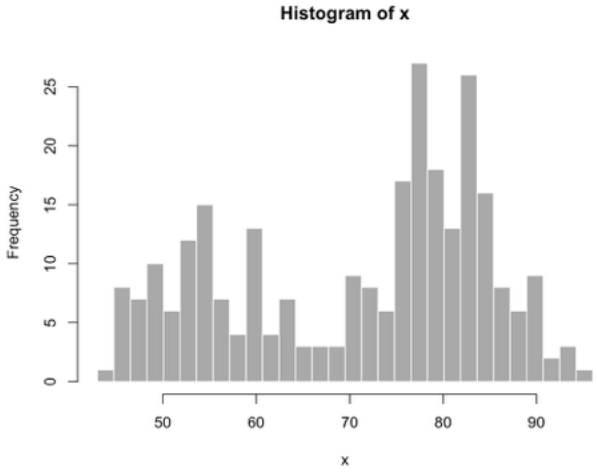
Console ~/Desktop/my-shiny/

```
> shiny::runApp()

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

Old Faithful Geyser Data

Number of bins:



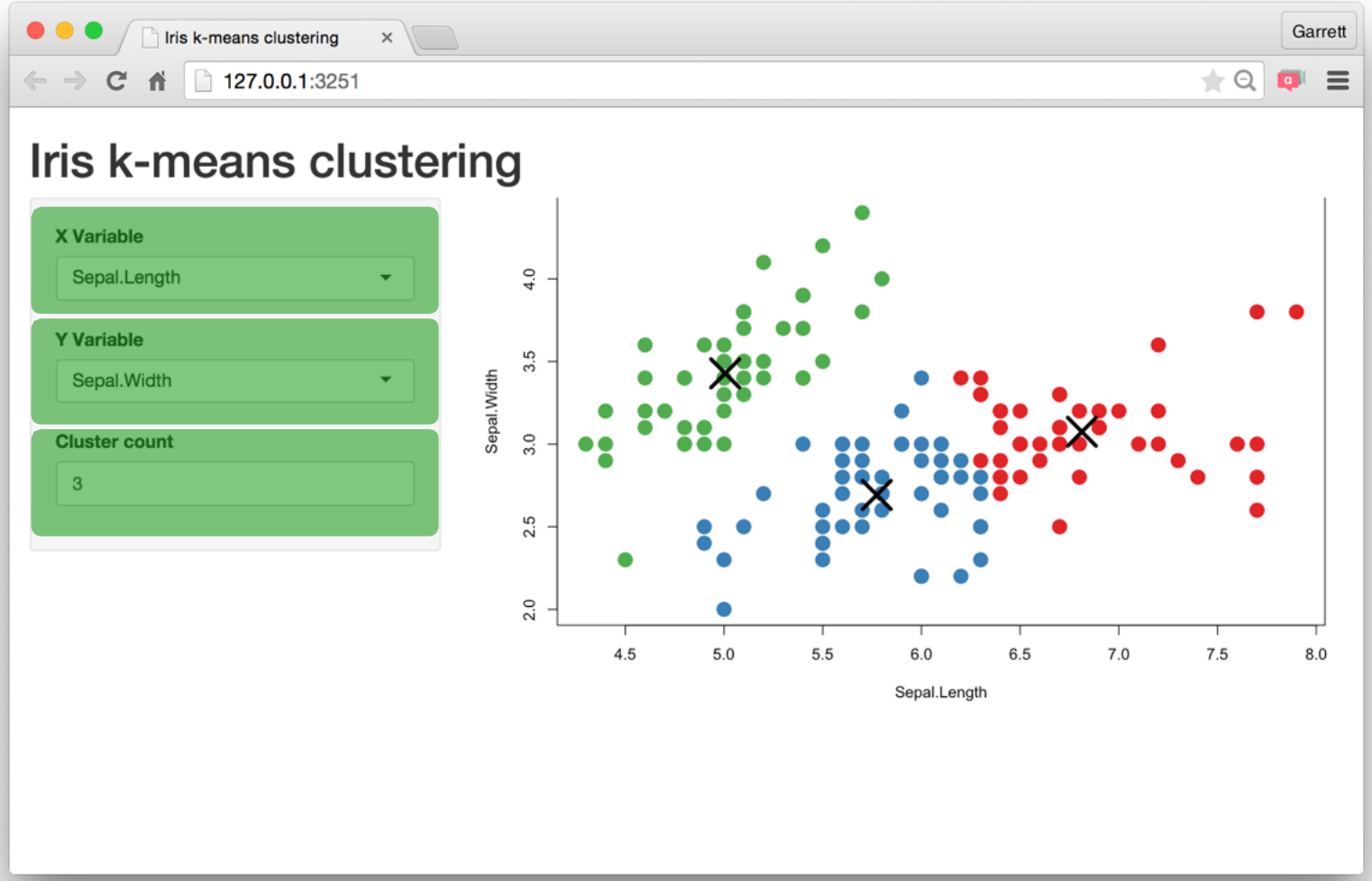


fluidPage()

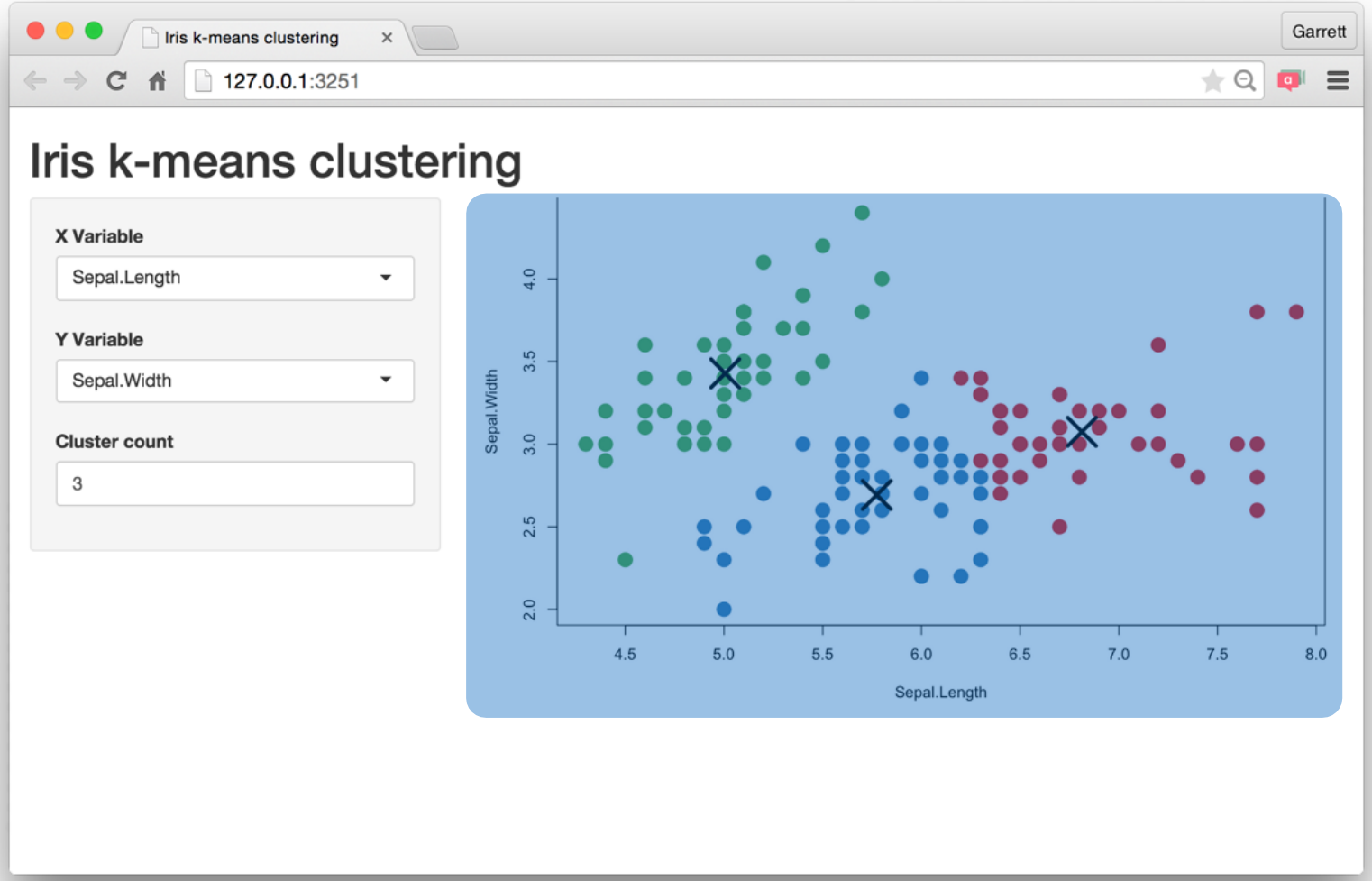
```
library(shiny)
ui <- fluidPage("Hello World")

server <- function(input, output) {}

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


NobleProg

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



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



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

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


NobleProg

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

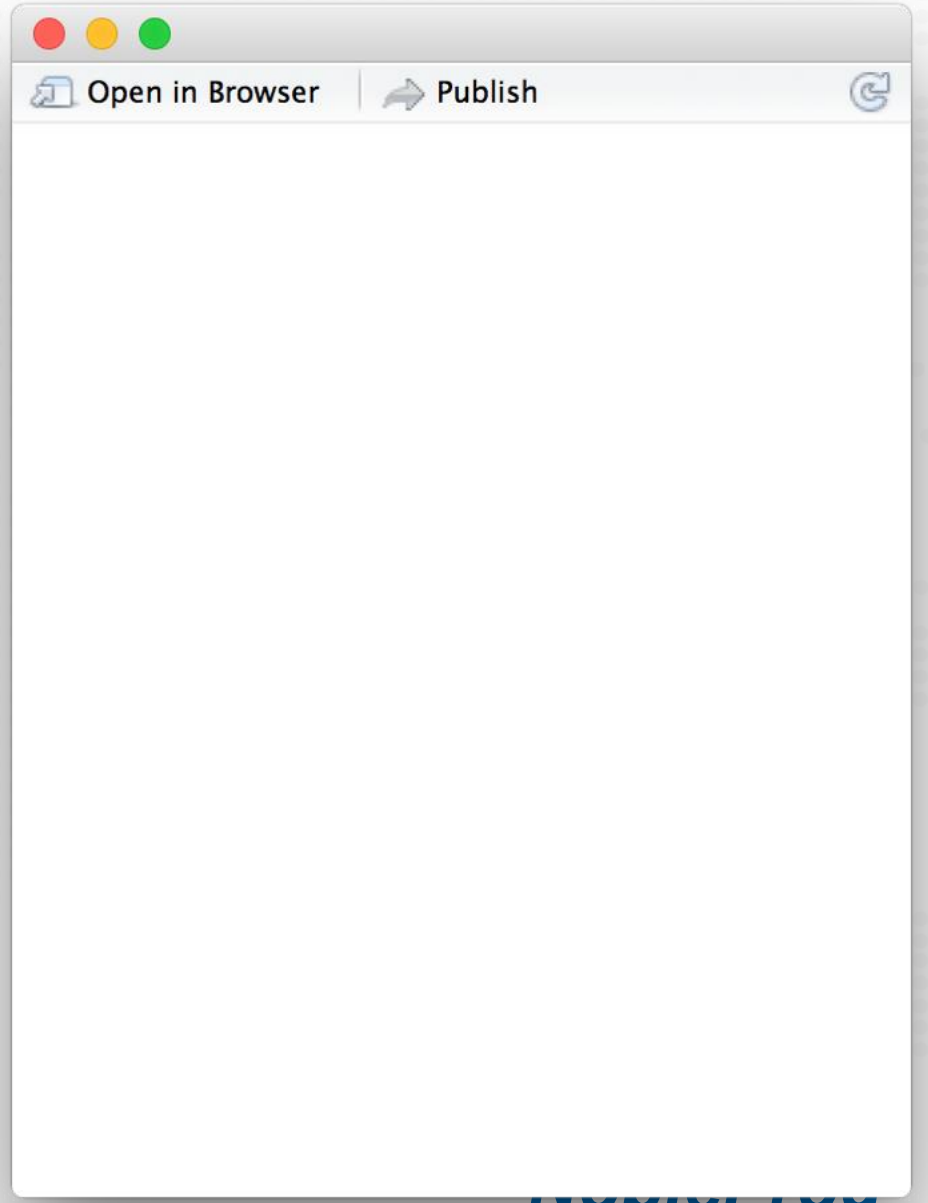
```
sliderInput(inputId = "num",
  label = "Choose a number",
  value = 25, min = 1, max = 100)
```

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


NobleProg

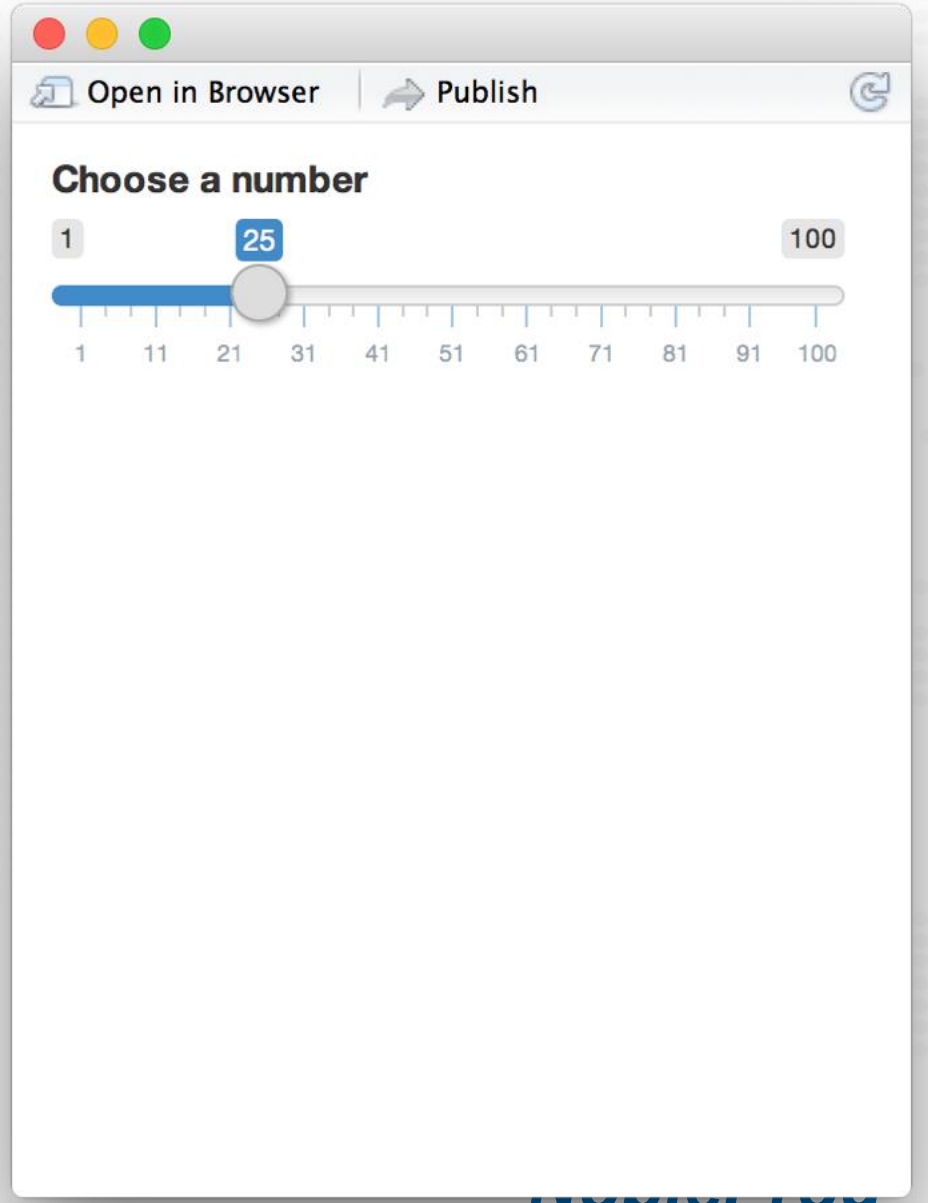
Create an input with an input function.

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




)

server <- function(input, output) {}

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



Create an input with an input function.

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

server <- function(input, output) {}

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



Buttons

Action

Submit

actionButton()

submitButton()

Single checkbox

☑ Choice A

checkboxInput()

Checkbox group

☑ Choice 1

☐ Choice 2

☐ Choice 3

checkboxGroupInput()

Date input

2014-01-01

dateInput()

Date range

2014-01-24 to 2014-01-24

dateRangeInput()

File input

Choose File No file chosen

fileInput()

Numeric input

1

numericInput()

Password Input

••••••••••

passwordInput()

Radio buttons

◉ Choice 1

○ Choice 2

○ Choice 3

radioButtons()

Select box

Choice 1

selectInput()

Sliders

0 50 100

0 25 75 100

sliderInput()

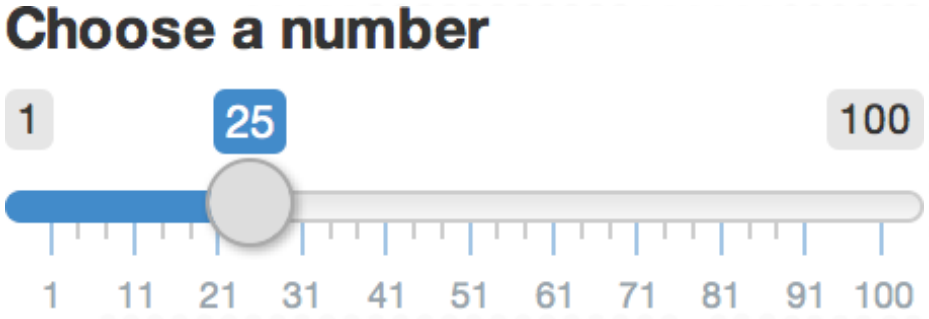
Text input

Enter text...

textInput()

NobleProg

Syntax




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

input name
(for internal use)

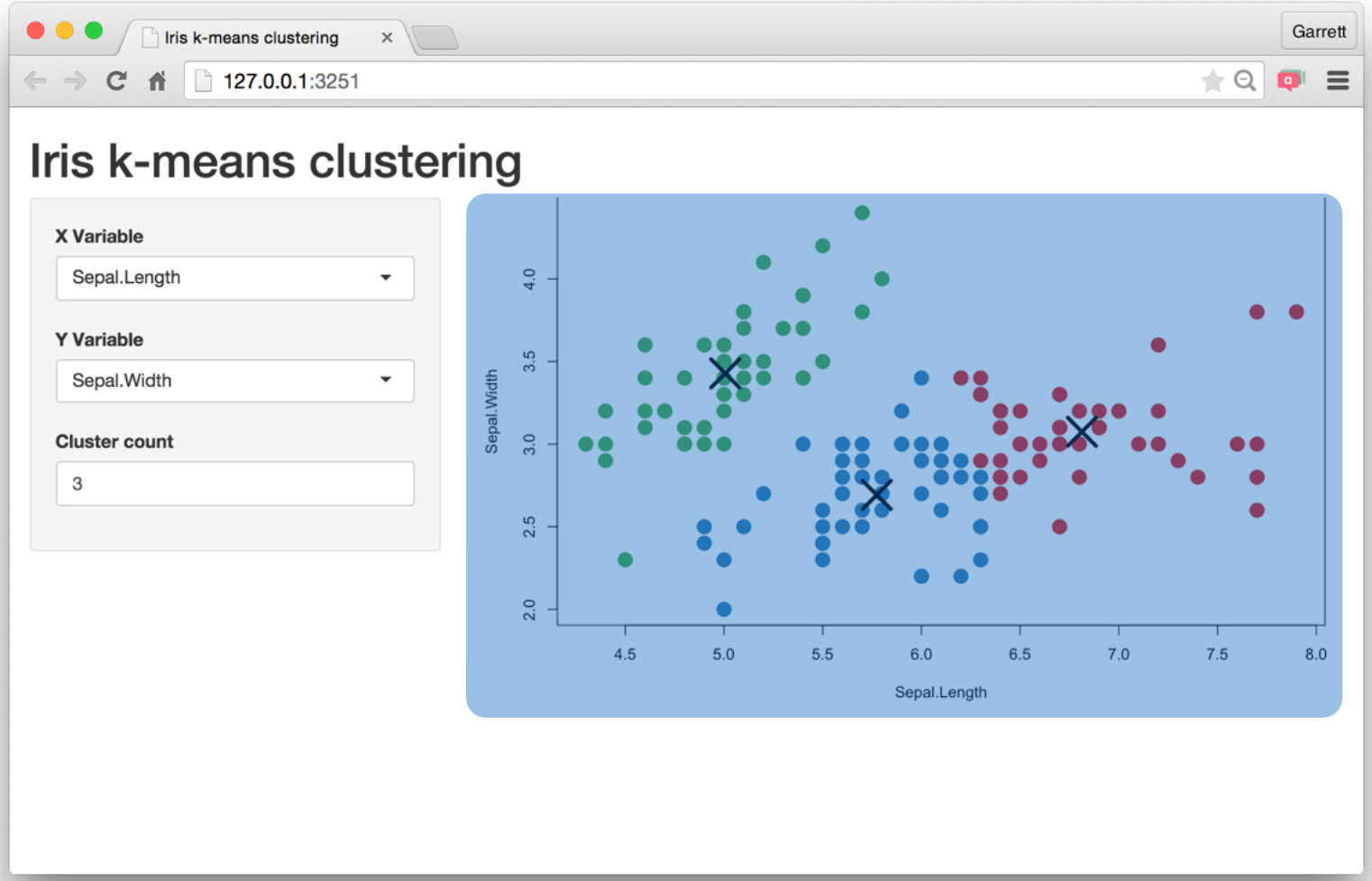
label to display

input specific arguments

```
?sliderInput
```


NobleProg

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



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

*Output()

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

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

the type of output to display

name to give to the output object

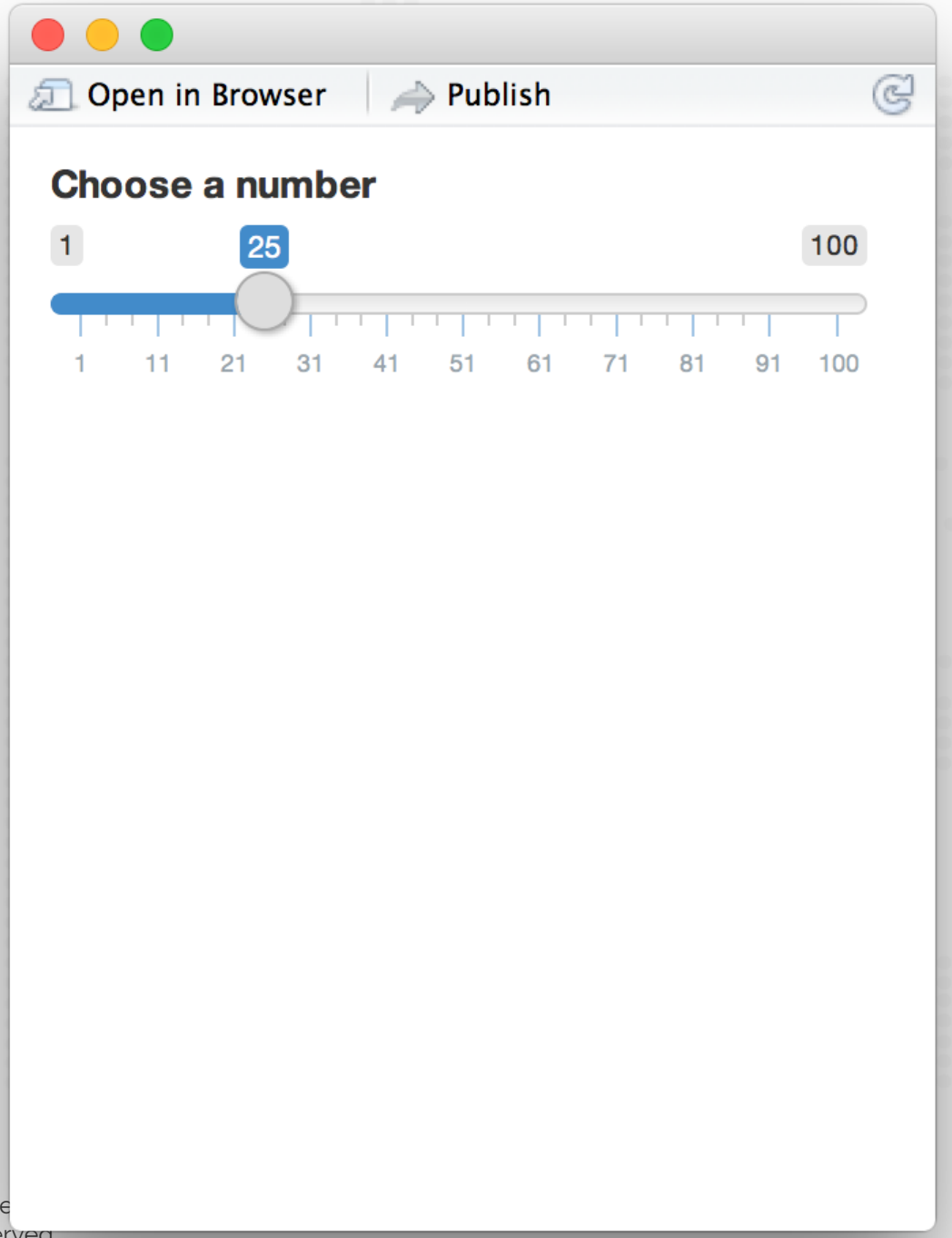
NobleProg

```r
library(shiny)

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

server <- function(input, output) {}

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

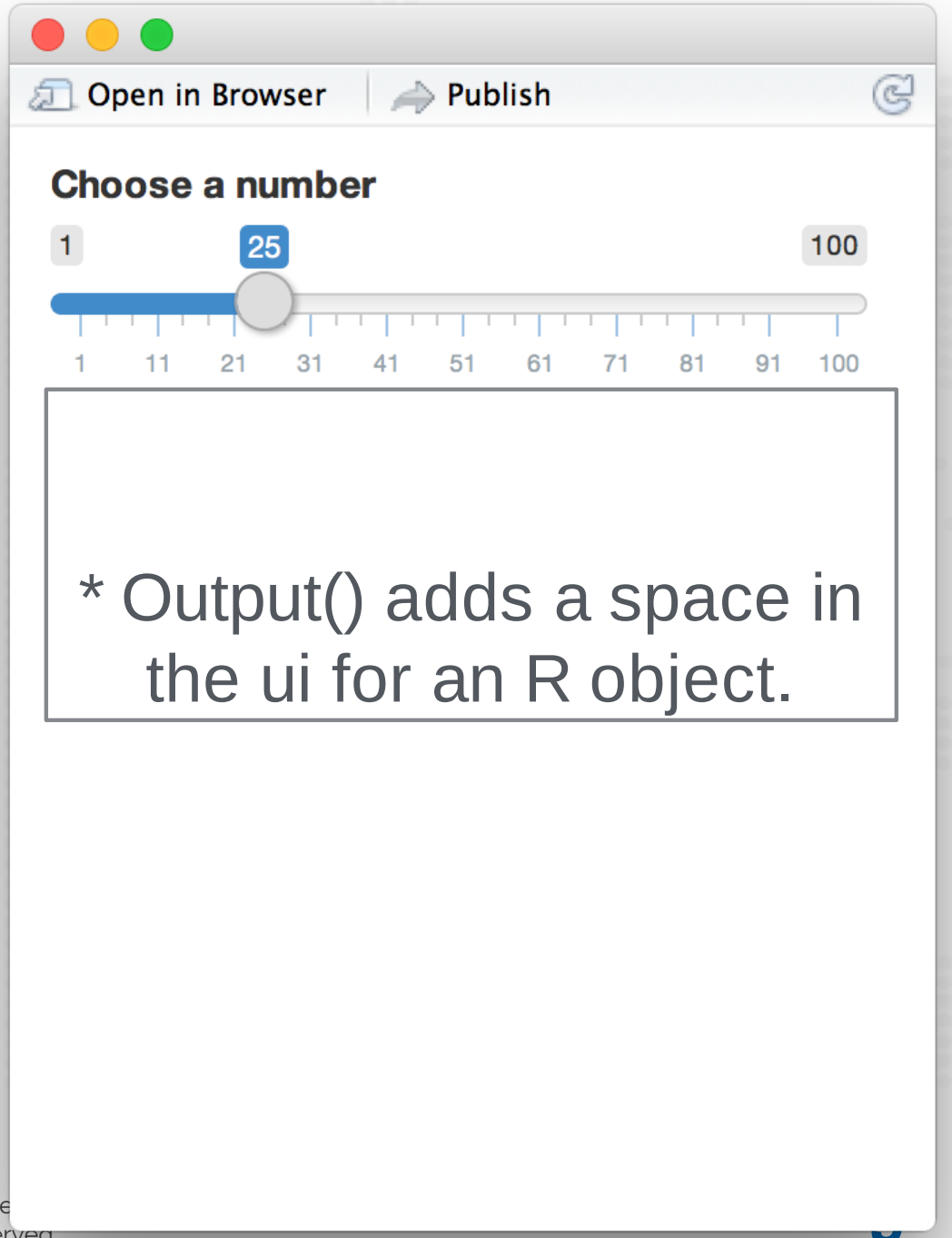
Comma between arguments

NobleProg

```r
library(shiny)

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

server <- function(input, output) {}

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

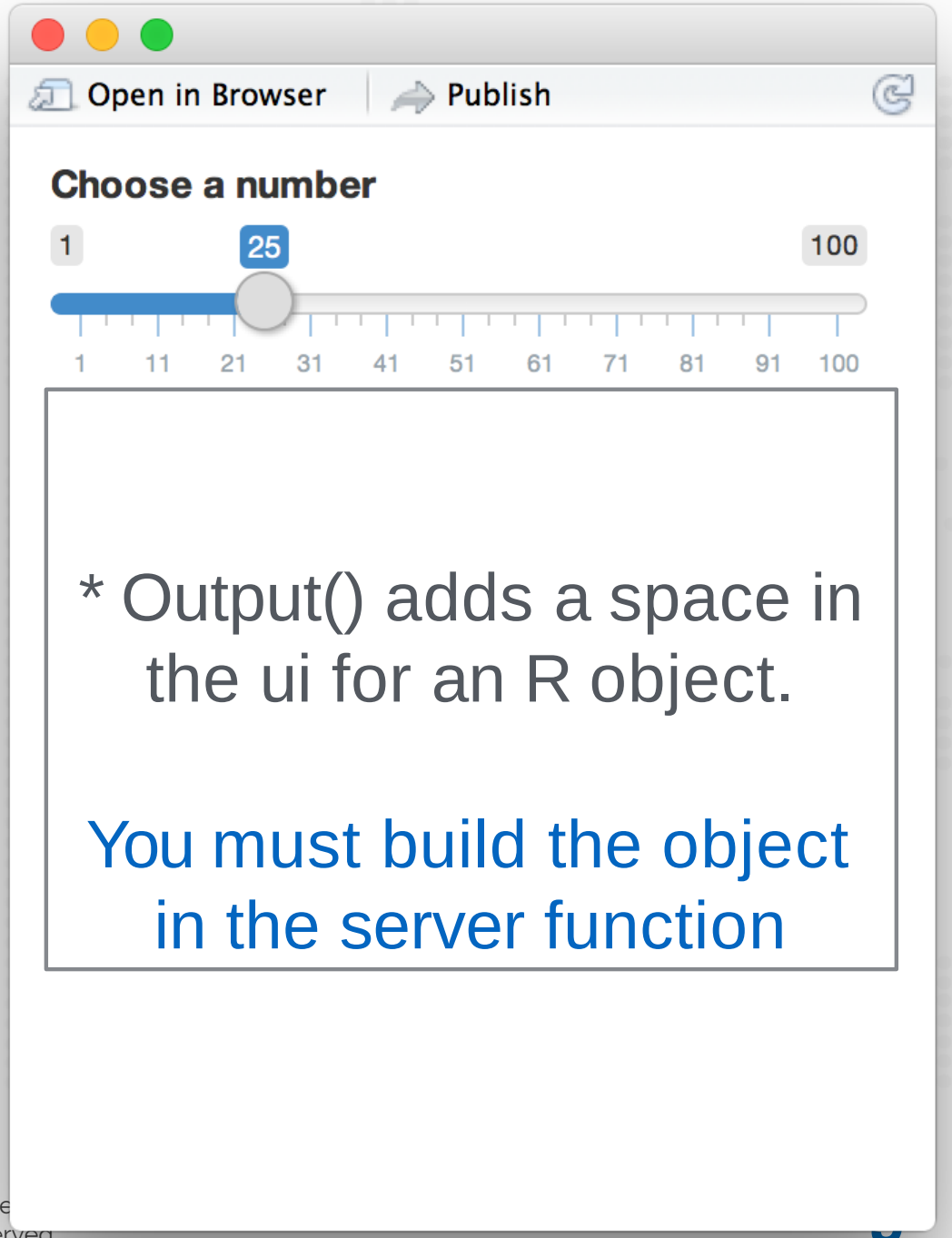


```r
library(shiny)

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

server <- function(input, output) {}

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



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

```r
library(shiny)

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

server <- function(input, output) {}

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



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

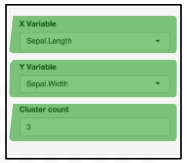
You must build the object in the server function

Recap

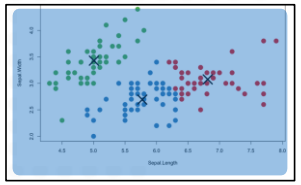
```
library(shiny)
ui <- fluidPage()
server <- function(input, output) {}
shinyApp(ui = ui, server = server)
```

Begin each app with the template

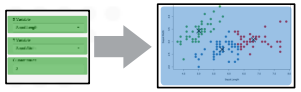
Add elements as arguments to **fluidPage()**



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



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



Assemble outputs from inputs in the server function

NobleProg

Use 3 rules to write the server function

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



}
```


NobleProg

1 Save objects to display to output$

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



}
```


NobleProg

1 Save objects to display to output$

```
output$hist
```

↓

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


NobleProg

2

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

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


})
}
```


NobleProg

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

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

NobleProg

render*()

Builds reactive output to display in UI

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

type of object to build

code block that builds the object

NobleProg

2

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

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


NobleProg

2

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

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


NobleProg

3 Access input values with input$

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


NobleProg

3 Access **input** values with input$

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

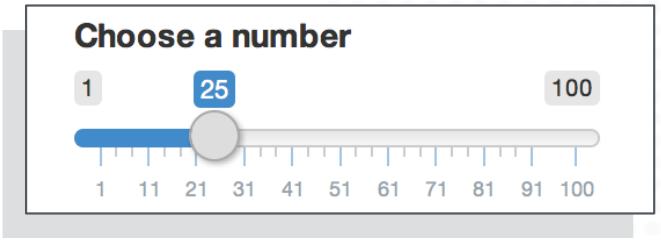
↓

```
input$num
```

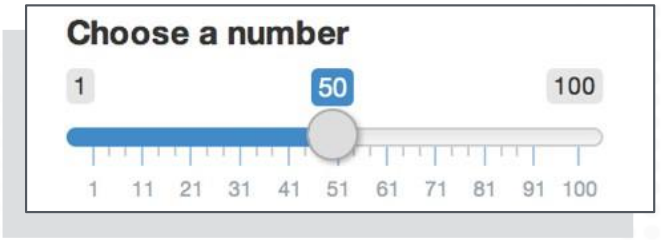

NobleProg

Input values

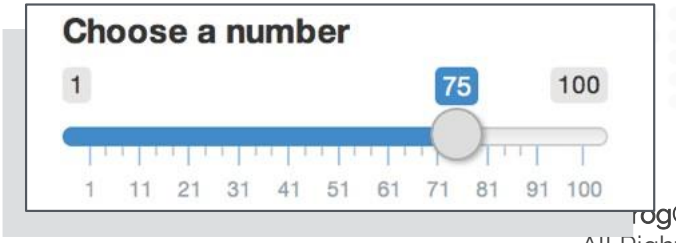
The input value changes whenever a user changes the input.




```
input$num = 25
```




```
input$num = 50
```

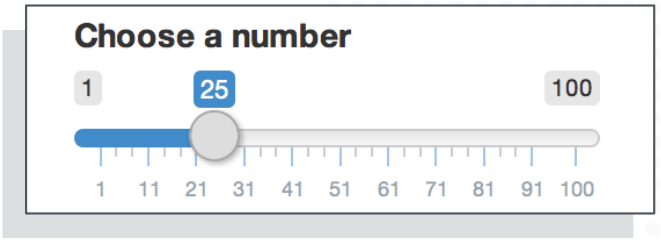



```
input$num = 75
```

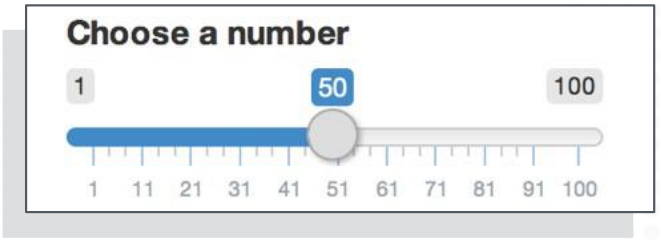

NobleProg

Input values

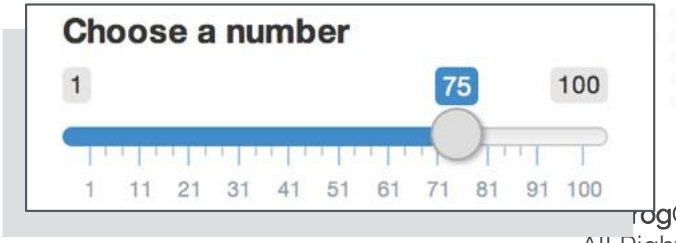
The input value changes whenever a user changes the input.




```
input$num = 25
```




```
input$num = 50
```




```
input$num =
```

Output will automatically update if you follow the 3 rules

NobleProg

Reactivity 101

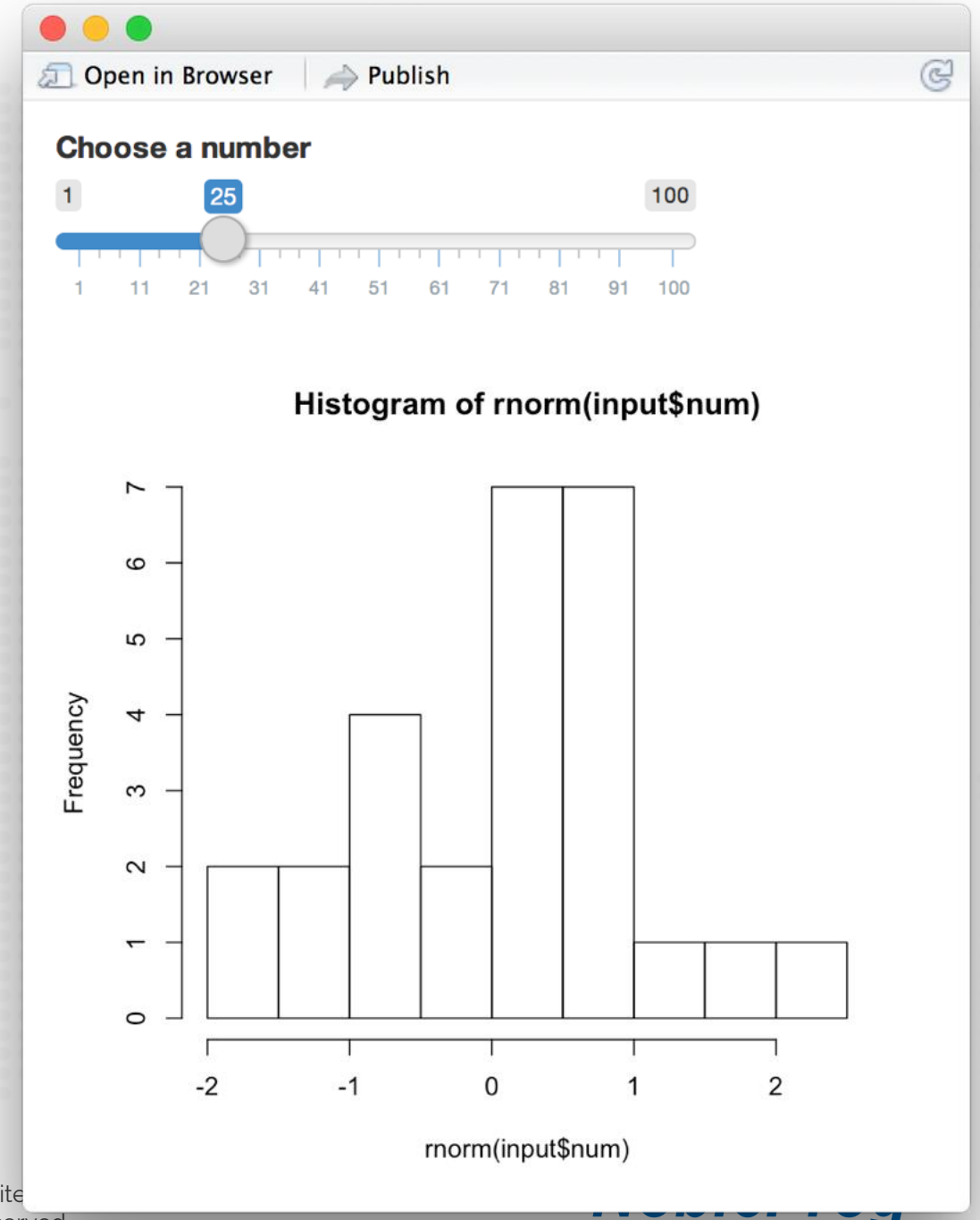
Reactivity automatically occurs whenever you use an input value to render an output object

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


NobleProg

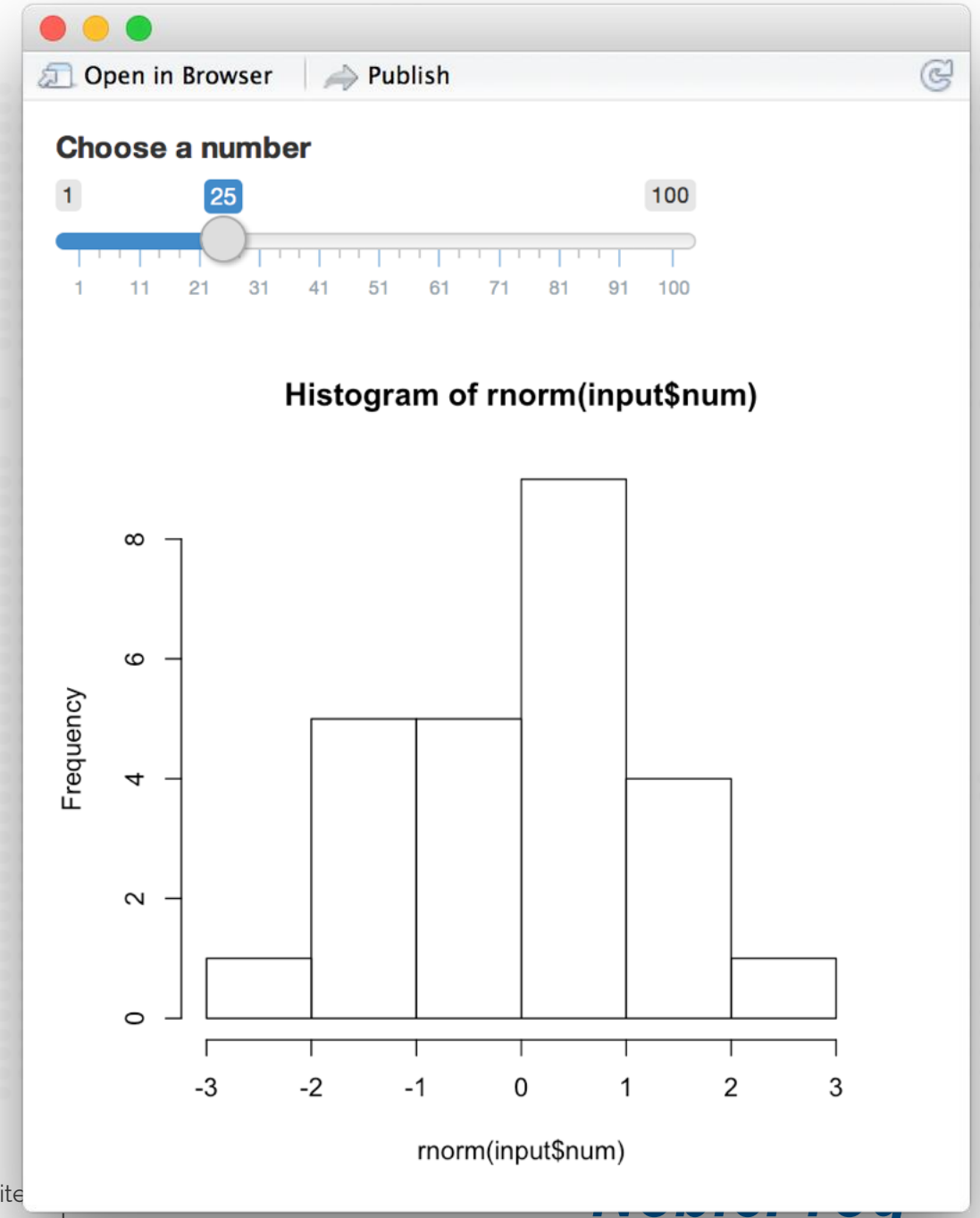
```
input$num
```

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



```
input$num
```

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



Recap: Server

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

```
output$hist <-
```

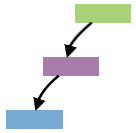
1. Save the output that you build to **output$**

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

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

```
input$num
```

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



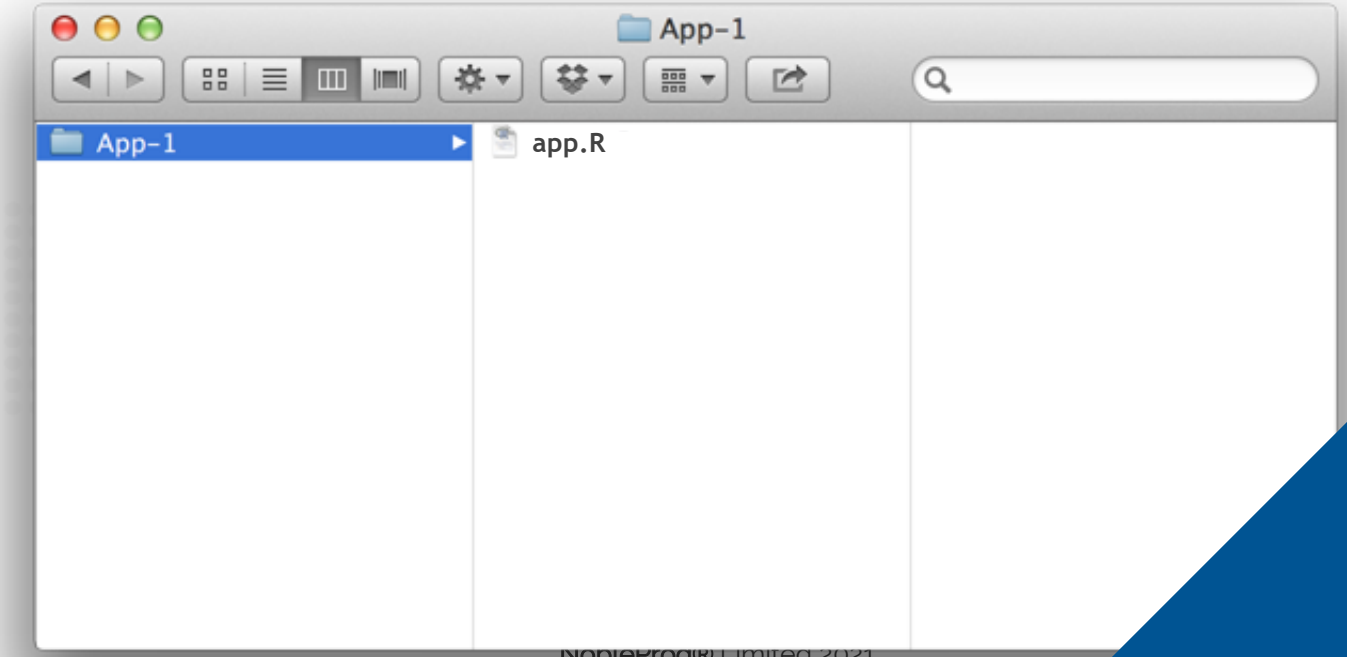
Create reactivity by using **Inputs** to build **rendered Outputs**

NobleProg

How to save your app

One directory with every file the app needs:

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



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

NobleProg

Two file apps

```r
library(shiny)

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

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

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

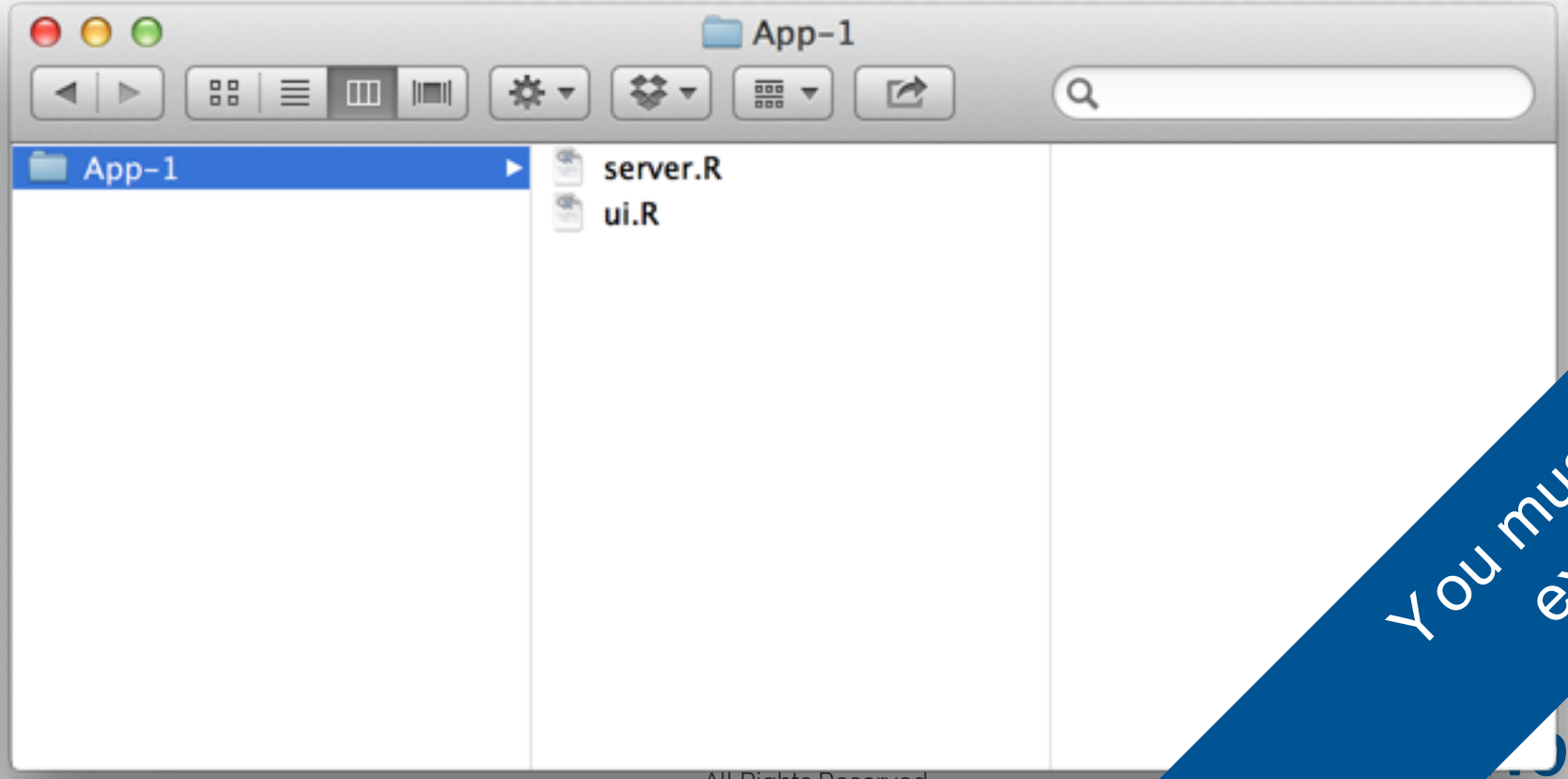
```r
# ui.R
library(shiny)
fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  plotOutput("hist")
)
```

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

Two file apps

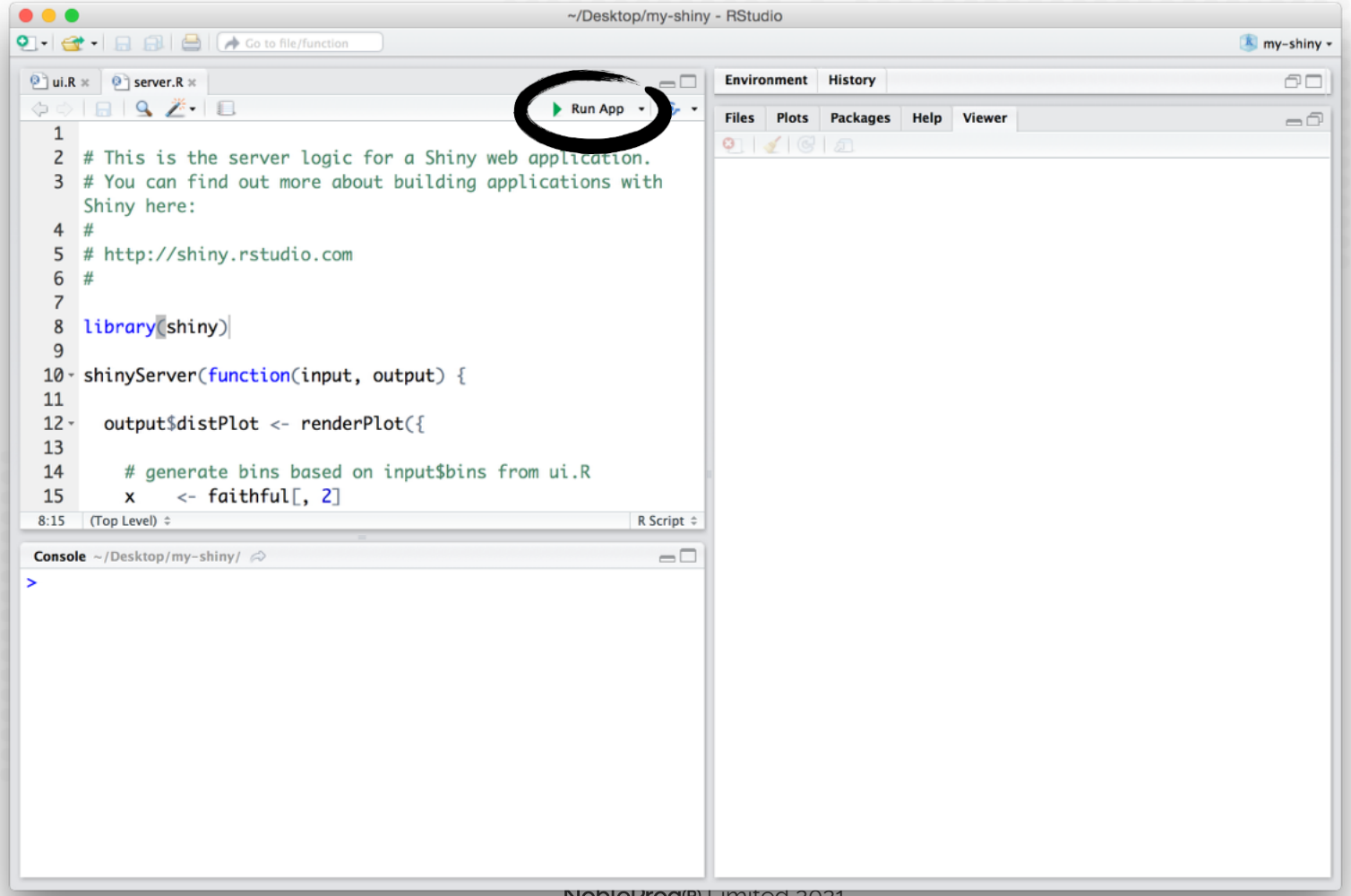
One directory with two files:

- server.R
- ui.R

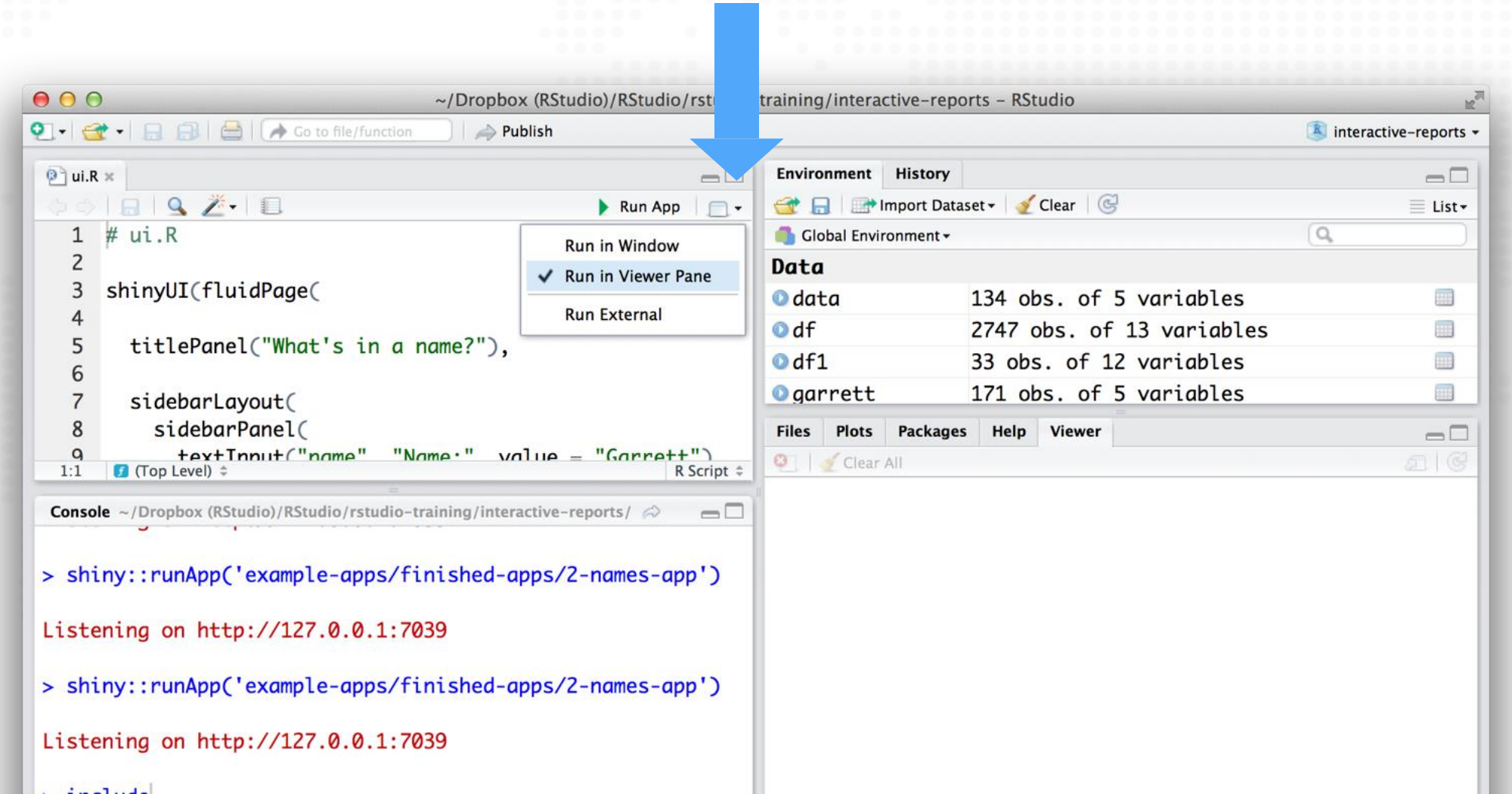


You must use these exact names

Launch an app



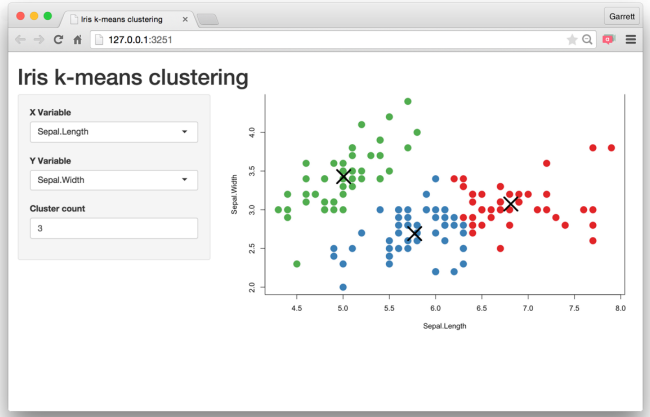
Display options



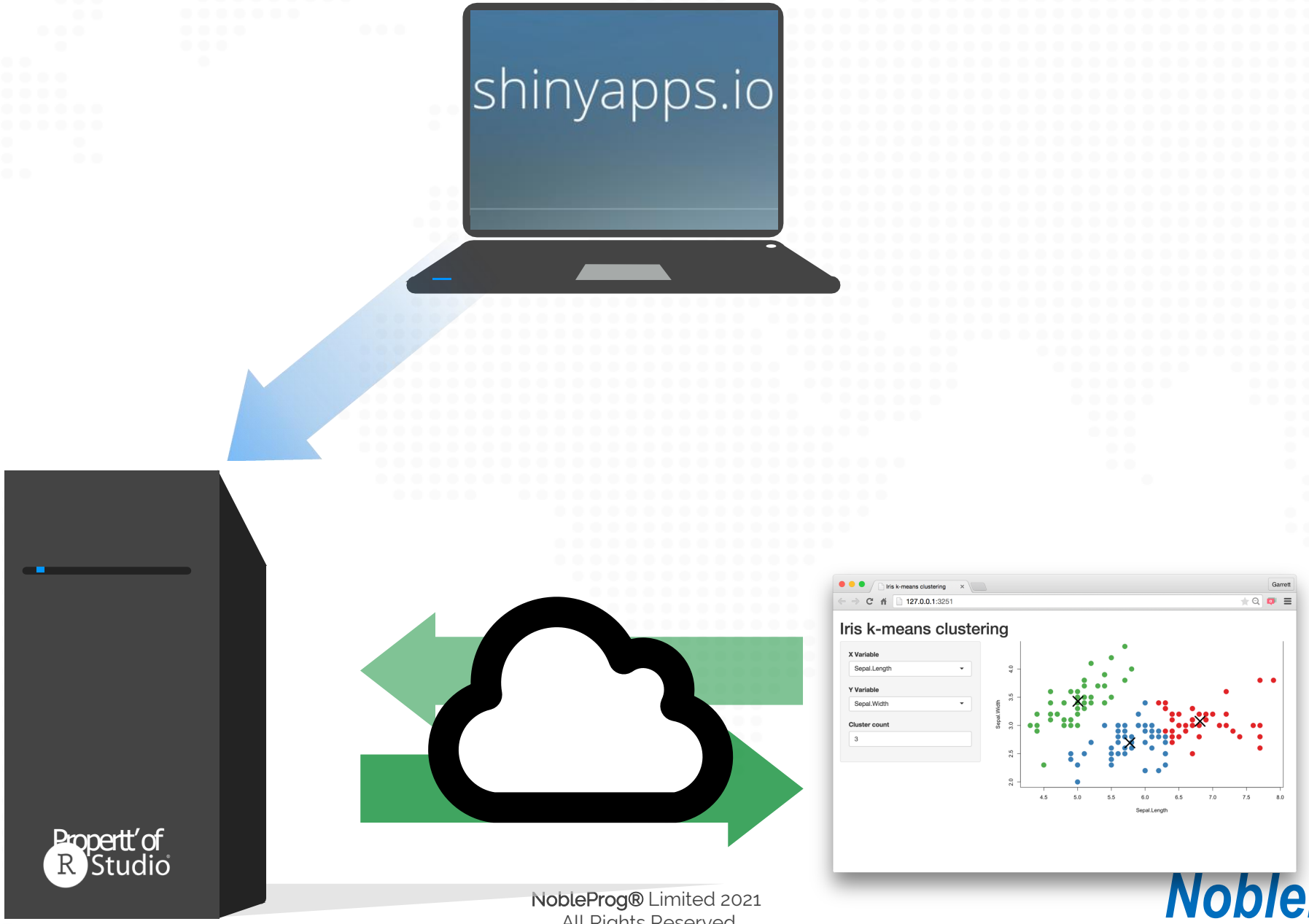
Shinyapps.io

A server maintained by RStudio

- free
- easy to use
- secure
- scalable



NobleProg



NobleProg

Shiny Server

Free!

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

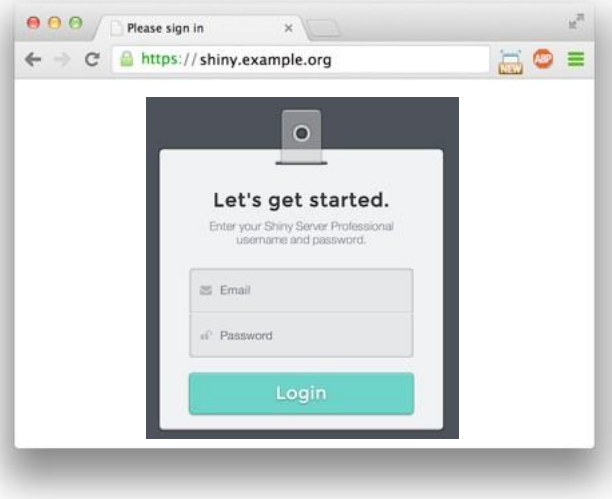
A back end program that builds a linux web server specifically designed to host Shiny apps.

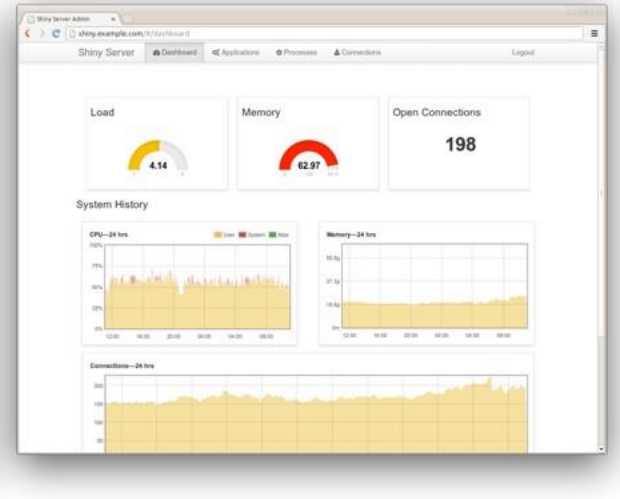
NobleProg

Shiny Server **Pro**

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

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





45 day evaluation free trial

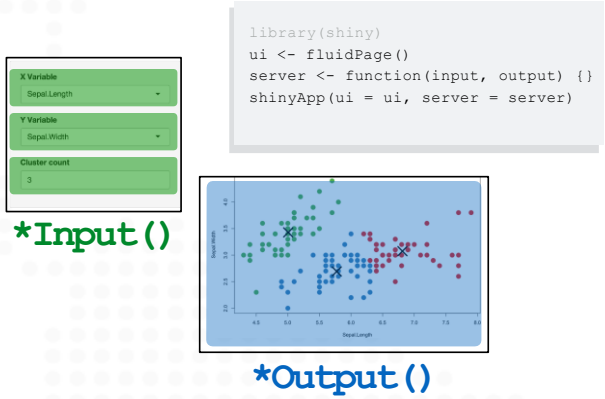
NobleProg

Other options

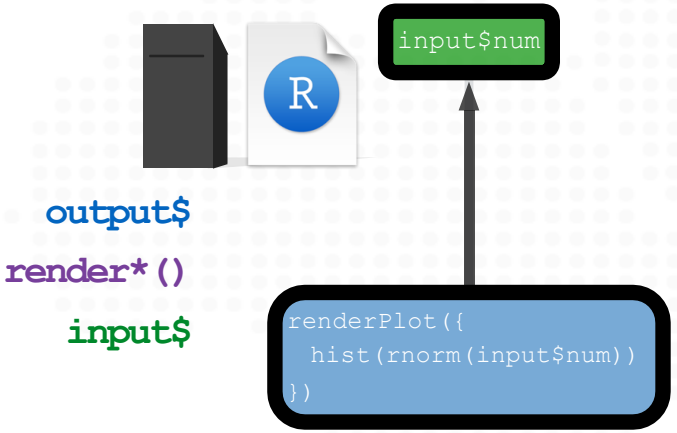
- Shinyapps.io + „hard-coded" credentials.
- Shinyapps.io + Firebase.
- Your own server:
  - My setup: Django auth + nginx + Shiny app in a Docker container.

NobleProg

You now know how to



Build an app



Create interactions

Share your apps

NobleProg

Exercise

- Load the file „population.tsv“ in the NEISS folder.
- Create a select input to filter out data by gender.
- Show the distribution of the population by the selected gender.

NobleProg