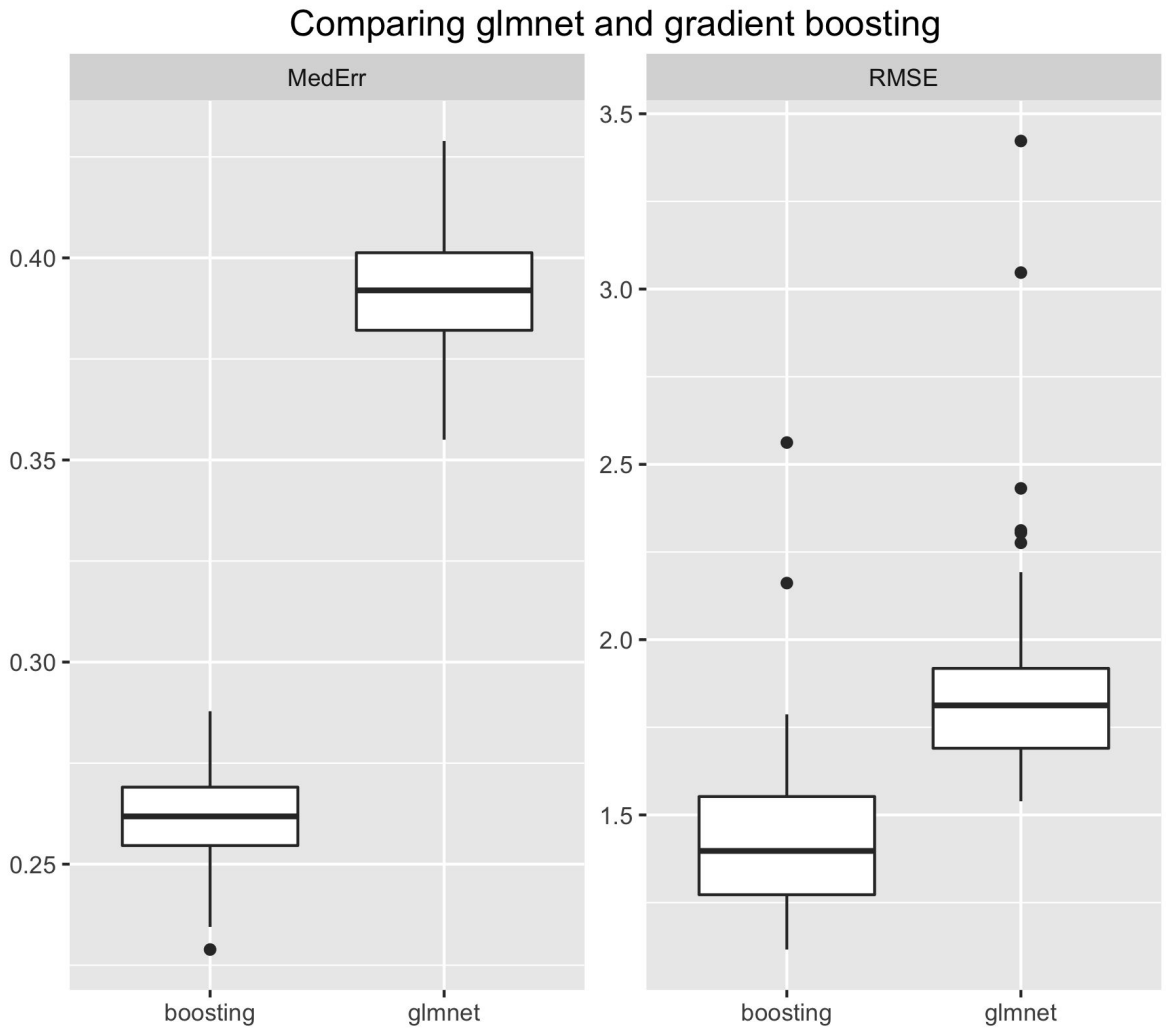
Introduction to Shiny

Yannet Interian -- USF

Agenda

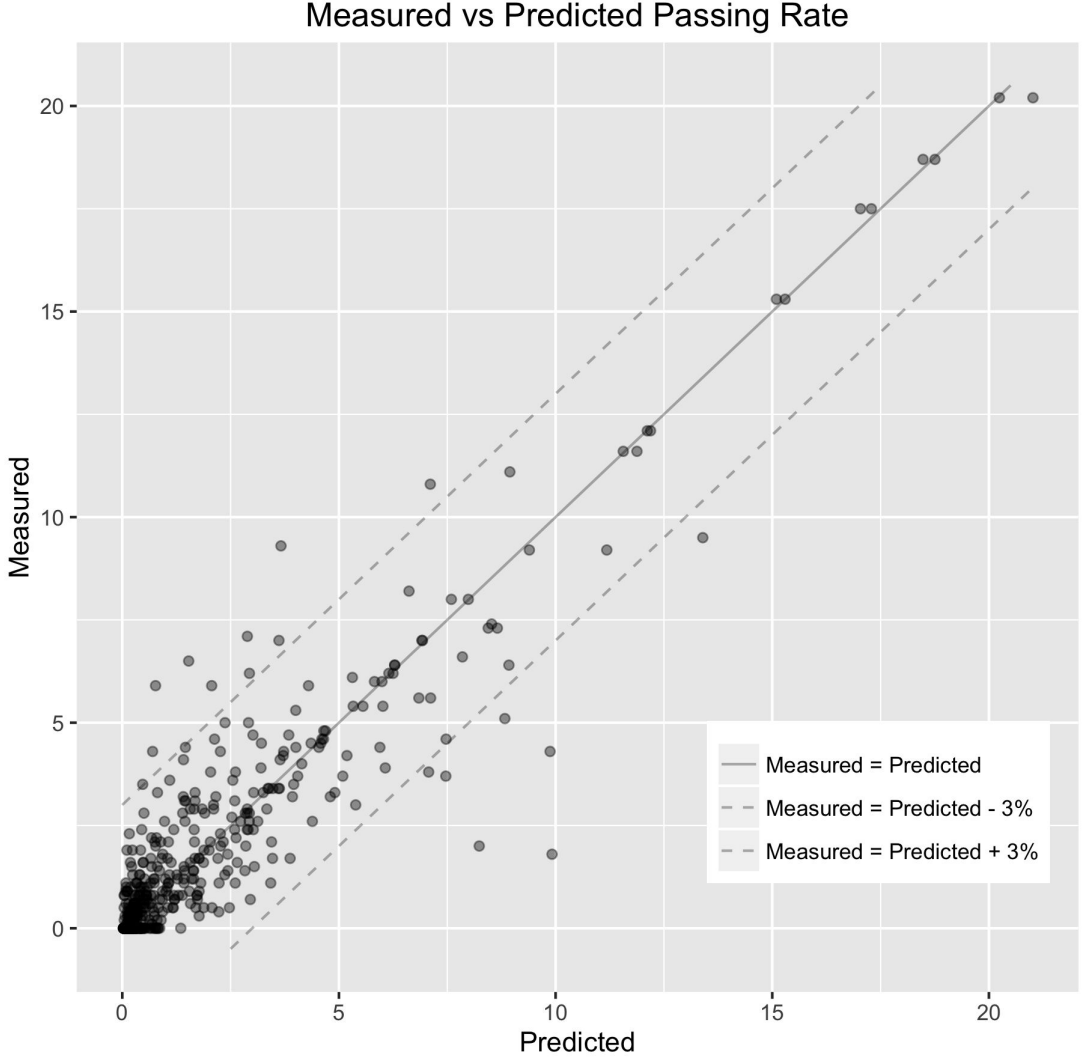
- Homework 1
- Shiny demo
- How to build a basic Shiny app
- Lab-2

Plots for the homework



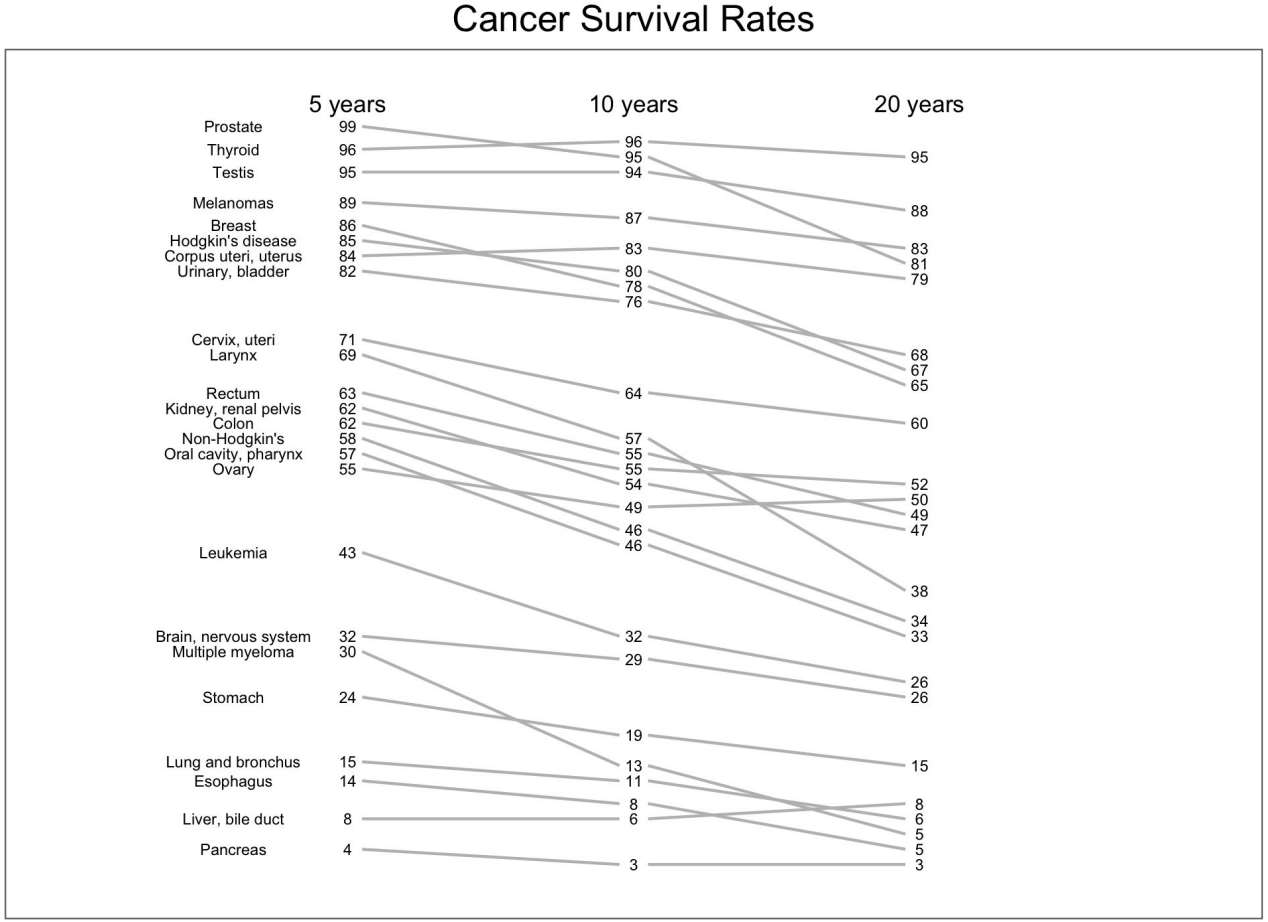
Notes:

- Different scales
- No x or y labels
- You may need to transform the original dataset
- Should be easy



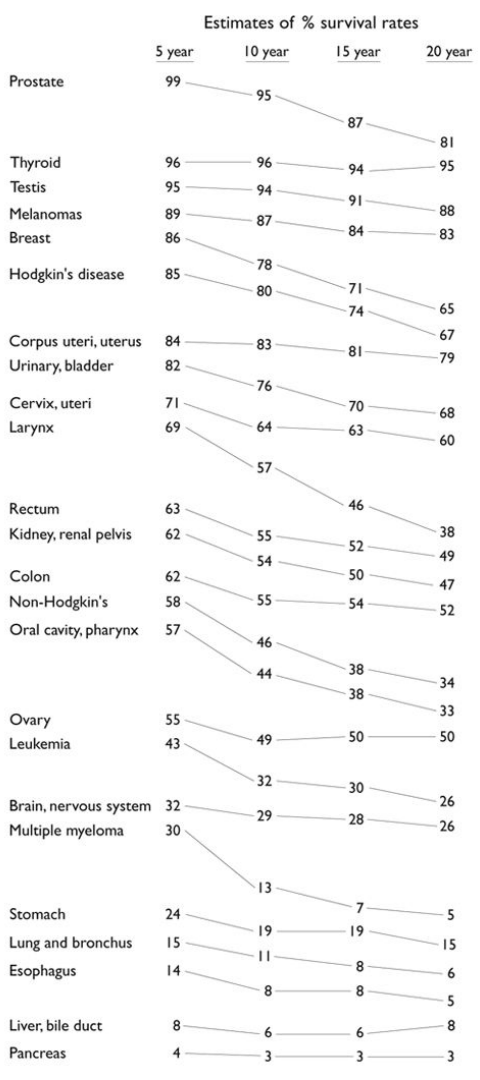
Notes:

- Position and color of the legend
- Transparency of the dots



Notes:

- This plot is difficult
- This version took me 2-3 hours
- This is a slopegraph



This is the original slopegraph from Tufte

Shiny demo

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

https://gallery.shinyapps.io/TSupplyDemand/

https://gallery.shinyapps.io/CDCPlot/

http://datasociety.co/kitamba-the-opportunity-project/

https://gallery.shinyapps.io/LDAelife/

https://vnijs.shinyapps.io/radiant/?SSUID=54b3d4eec6

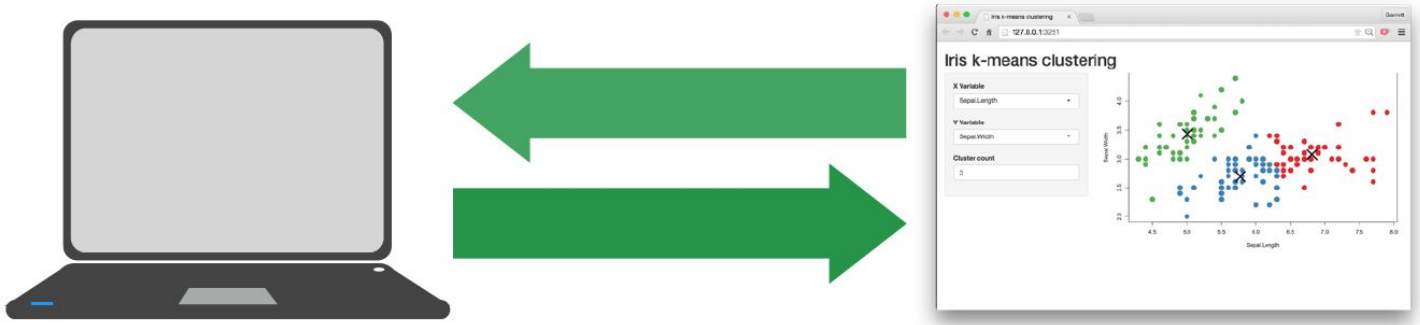
How to build a basic Shiny app

```
git clone git@github.com:usfviz/class-code.git
```

Demo

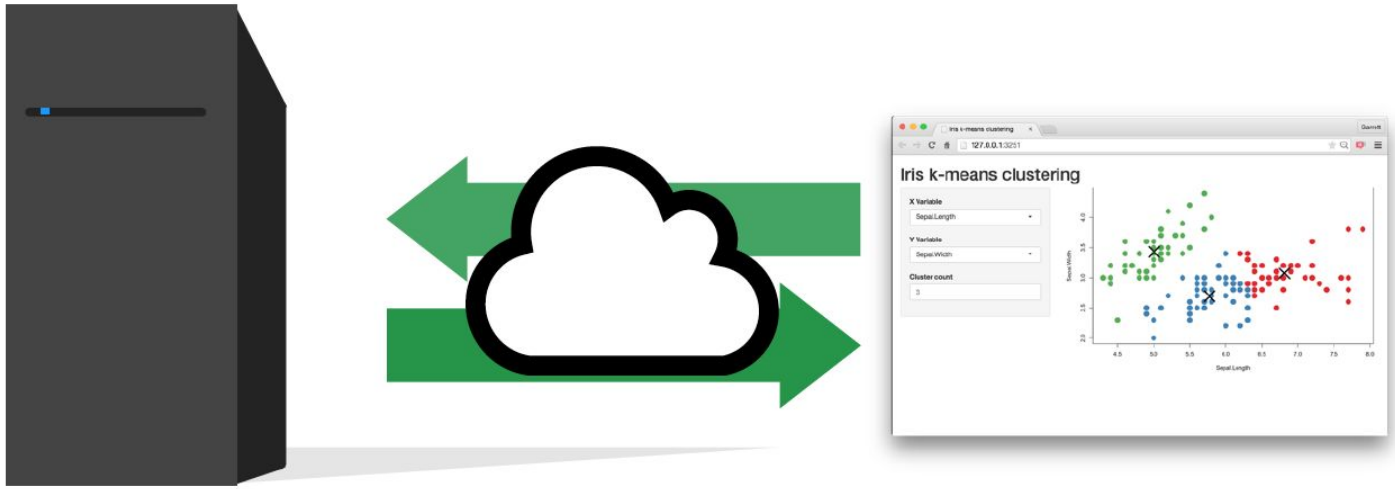

~/teaching/data-vis/github/data-vis/intro-shiny/code - Shiny
http://127.0.0.1:5683
Open in Browser
Publish
Iris k-means clustering
X Variable
Petal.Length
Y Variable
Sepal.Width
Cluster count
3
Sepal.Width
Petal.Length
2.0
2.5
3.0
3.5
4.0
1
2
3
4
5
6
7

Every Shiny app is maintained by a computer running R



Based on bit.ly/shiny-quickstart-1 by RStudio

Every Shiny app is maintained by a computer running R



Based on bit.ly/shiny-quickstart-1 by RStudio

Server Instructions

User Interface (UI)

Based on bit.ly/shiny-quickstart-1 by RStudio

Demo UI

```
library(shiny)
ui <- fluidPage(
  headerPanel('Iris k-means clustering'),
  sidebarPanel(
    selectInput('xcol', 'X Variable', names(iris)),
    selectInput('ycol', 'Y Variable', names(iris),
      selected = names(iris)[[2]]),
    numericInput('clusters', 'Cluster count', 3,
      min = 1, max = 9)
  ),
  mainPanel(
    plotOutput('plot1')
  )
)
```

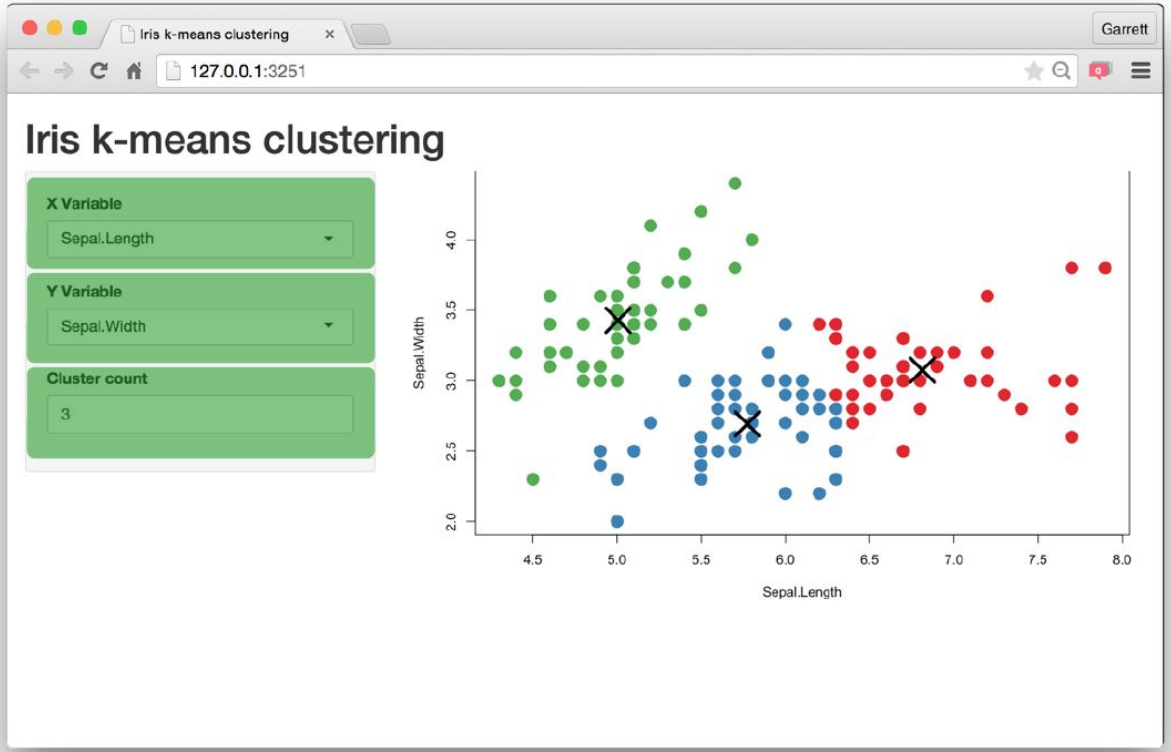
Based on bit.ly/shiny-quickstart-1 by RStudio

Use this template

```
library(shiny)

ui <- fluidPage()

server <- function(input, output) {}

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

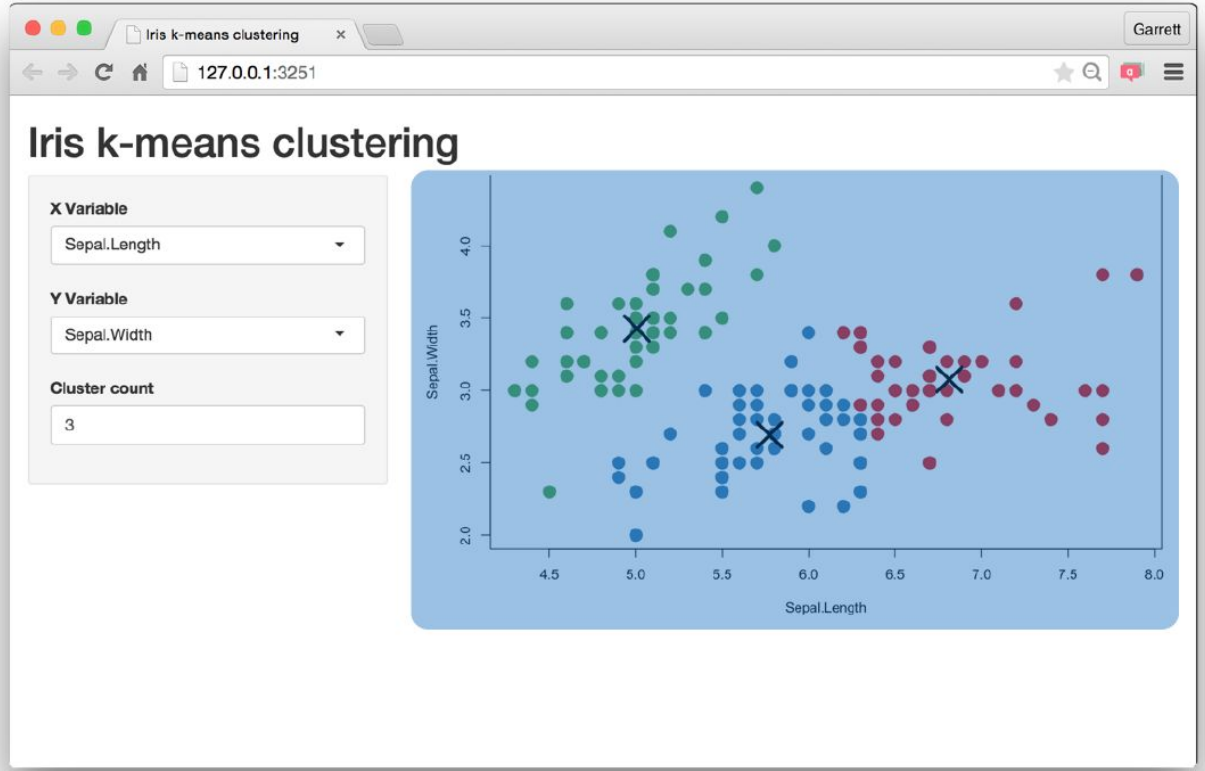
Based on bit.ly/shiny-quickstart-1 by RStudio

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



Based on bit.ly/shiny-quickstart-1 by RStudio

Build your app around inputs and outputs



Based on bit.ly/shiny-quickstart-1 by RStudio

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

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

Based on bit.ly/shiny-quickstart-1 by RStudio

Inputs

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

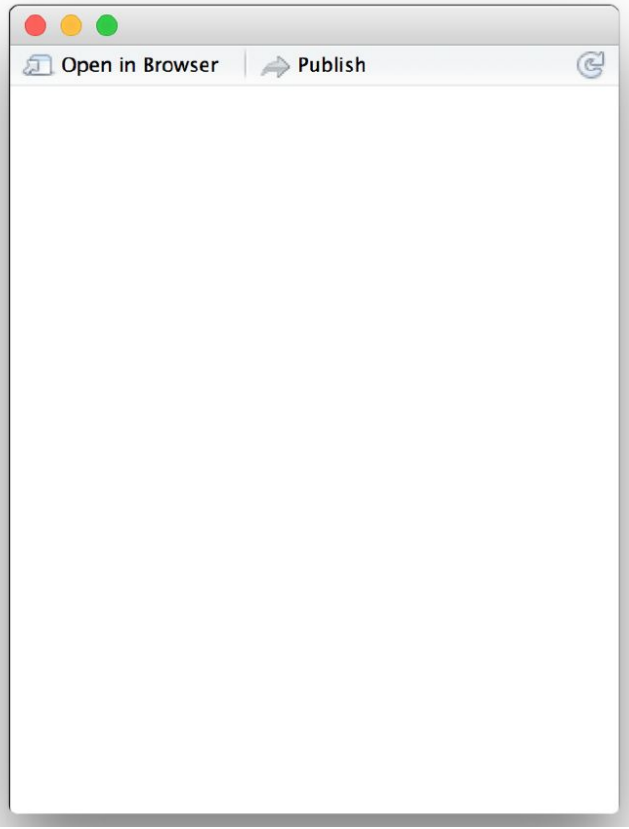
```
sliderInput(inputId = "num",
  label = "Choose a number",
  value = 25, min = 1, max = 100)
```

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

Based on bit.ly/shiny-quickstart-1 by RStudio

Create an input with an input function.

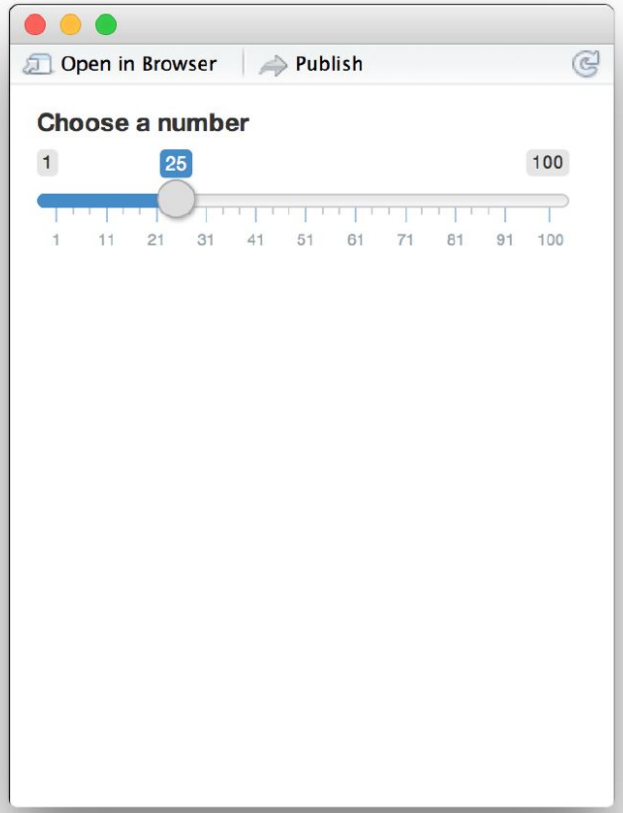
```
library(shiny)
ui <- fluidPage(



)

server <- function(input, output) {}

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



Based on bit.ly/shiny-quickstart-1 by RStudio

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)
```



Based on bit.ly/shiny-quickstart-1 by RStudio

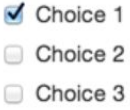
Buttons

`actionButton()`
`submitButton()`

Single checkbox

`checkboxInput()`

Checkbox group



`checkboxGroupInput()`

Date input

`dateInput()`

Date range

`dateRangeInput()`

File input

`fileInput()`

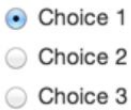
Numeric input

`numericInput()`

Password Input

`passwordInput()`

Radio buttons



`radioButtons()`

Select box

`selectInput()`

Sliders

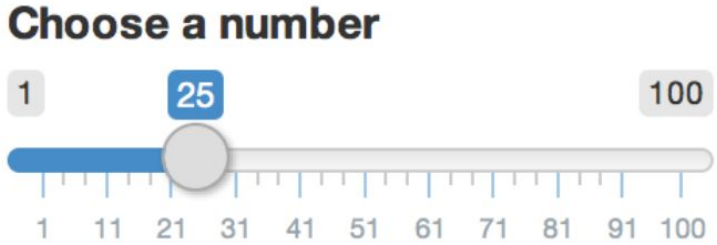
`sliderInput()`

Text input

`textInput()`

Based on bit.ly/shiny-quickstart-1 by RStudio

Syntax




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

input name
(for internal use)

Notice:
Id not ID

label to display

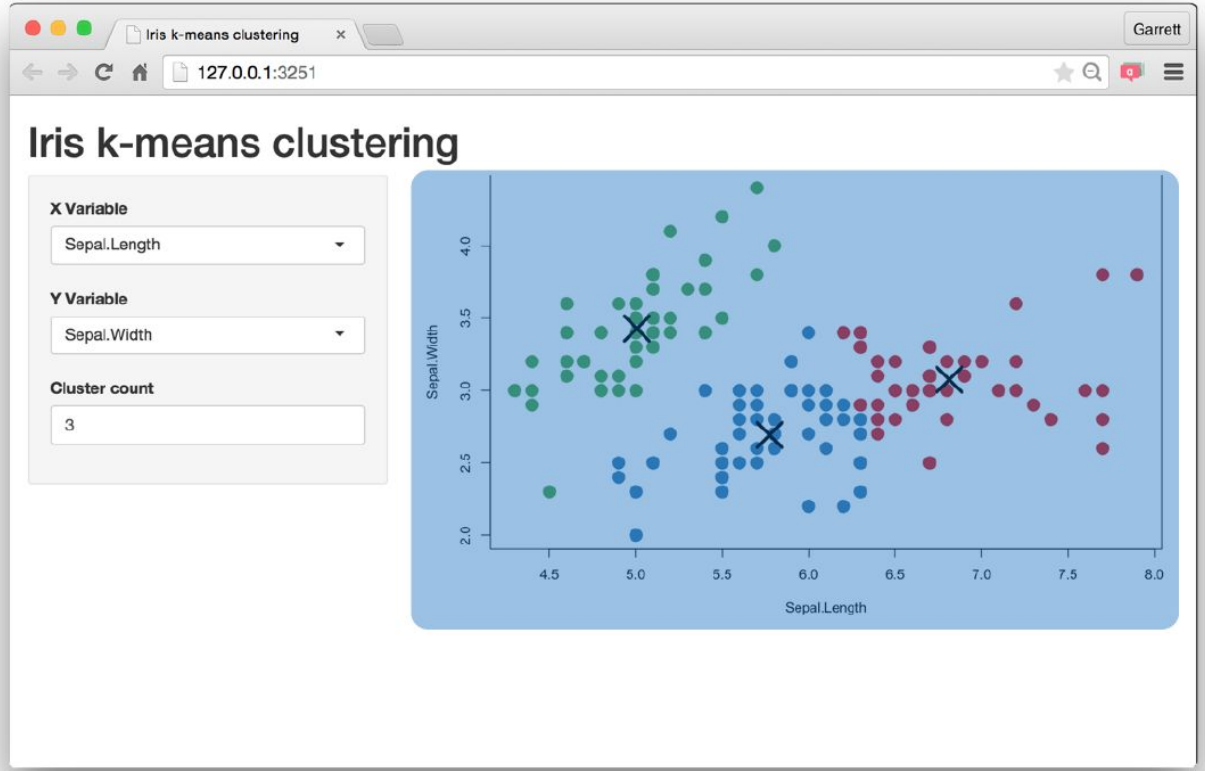
input specific arguments

```
?sliderInput
```

Based on bit.ly/shiny-quickstart-1 by RStudio

Outputs

Build your app around inputs and outputs



Based on bit.ly/shiny-quickstart-1 by RStudio

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

Based on bit.ly/shiny-quickstart-1 by RStudio

*Output()

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

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

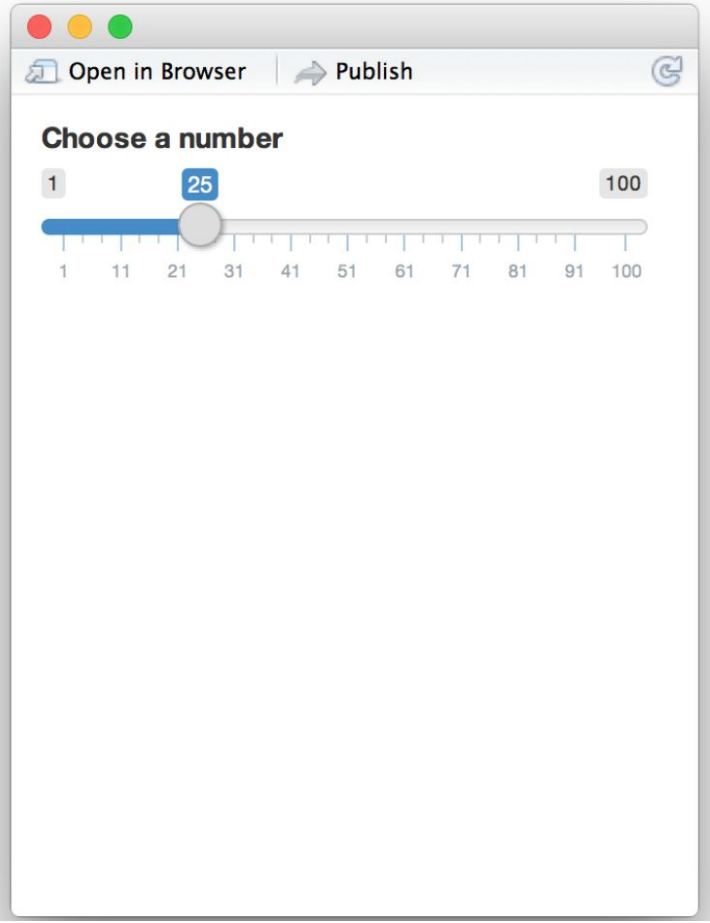
Based on bit.ly/shiny-quickstart-1 by RStudio

```
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

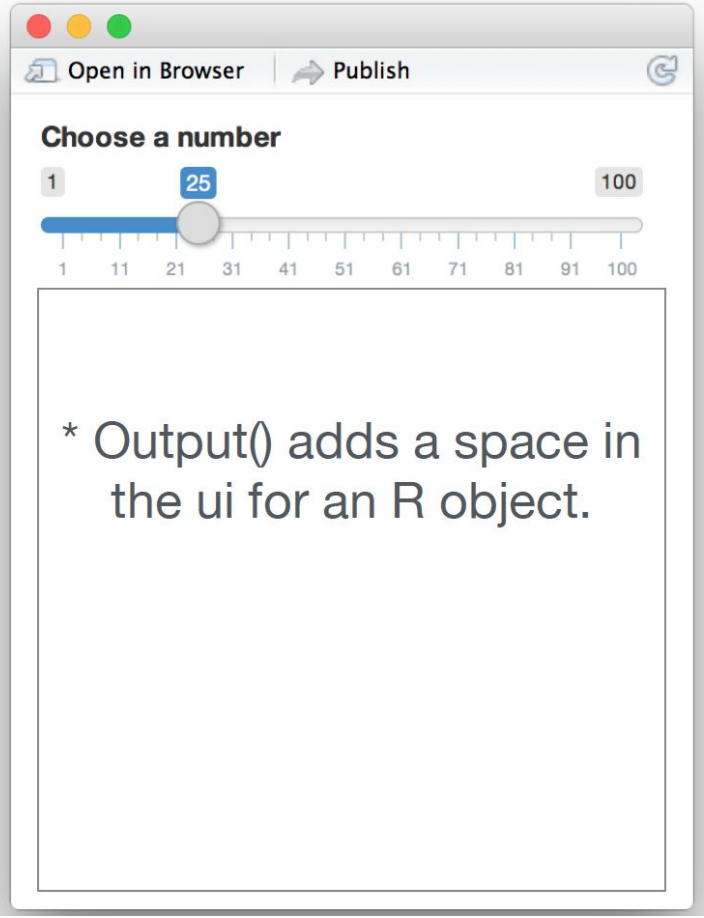
Based on bit.ly/shiny-quickstart-1 by RStudio

```
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)
```



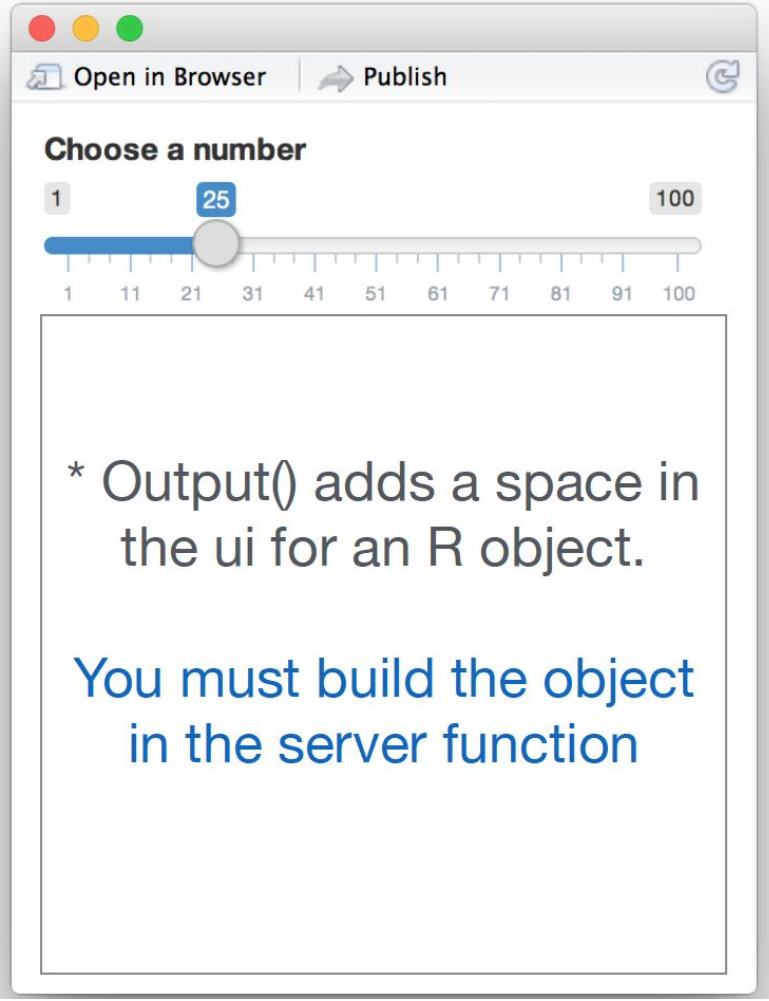
Based on bit.ly/shiny-quickstart-1 by RStudio

```
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)
```



Based on bit.ly/shiny-quickstart-1 by RStudio

```
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)
```



Based on bit.ly/shiny-quickstart-1 by RStudio

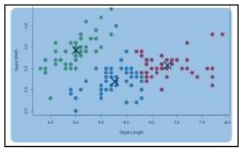
Recap

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

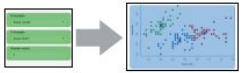
Begin each app with the template

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

Based on bit.ly/shiny-quickstart-1 by RStudio

Tell the
server
how to assemble
inputs into outputs

Use 3 rules to write the server function

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



}
```

Based on bit.ly/shiny-quickstart-1 by RStudio

1 Save objects to display to output$

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



}
```

Based on bit.ly/shiny-quickstart-1 by RStudio

1 Save objects to display to output$

```
output$hist
```

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

Based on bit.ly/shiny-quickstart-1 by RStudio

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

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

  })
}
```

Based on bit.ly/shiny-quickstart-1 by RStudio

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

Based on bit.ly/shiny-quickstart-1 by RStudio

render*()

Builds reactive output to display in UI

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

Based on bit.ly/shiny-quickstart-1 by RStudio

2

Build objects to display with render*()

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

Based on bit.ly/shiny-quickstart-1 by RStudio

2 Build objects to display with render*()

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

Based on bit.ly/shiny-quickstart-1 by RStudio

3

Access input values with input$

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

Based on bit.ly/shiny-quickstart-1 by RStudio

3 Access **input** values with input$

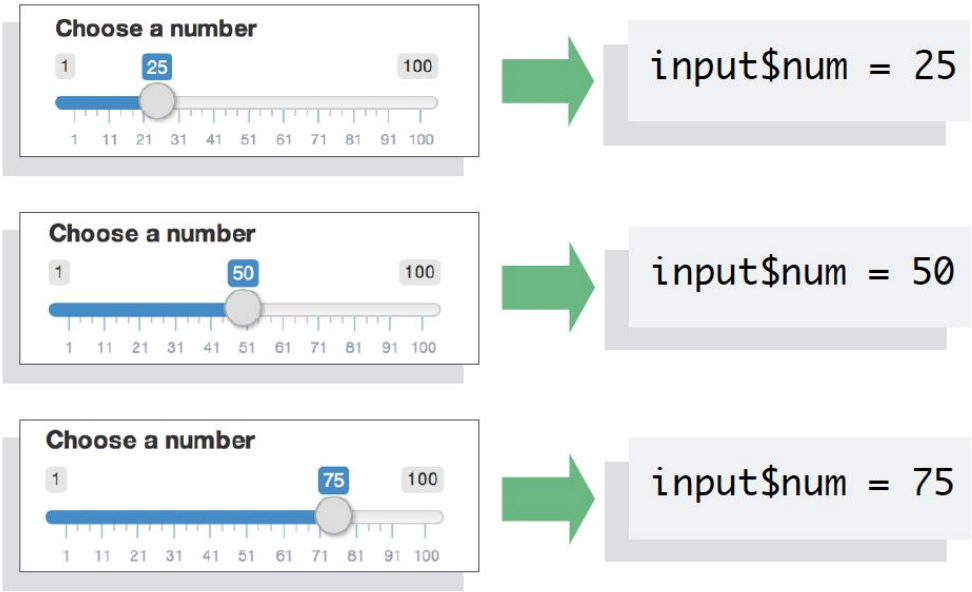
```
sliderInput(inputId = "num",…)
```

```
input$num
```

Based on bit.ly/shiny-quickstart-1 by RStudio

Input values

The input value changes whenever a user changes the input.



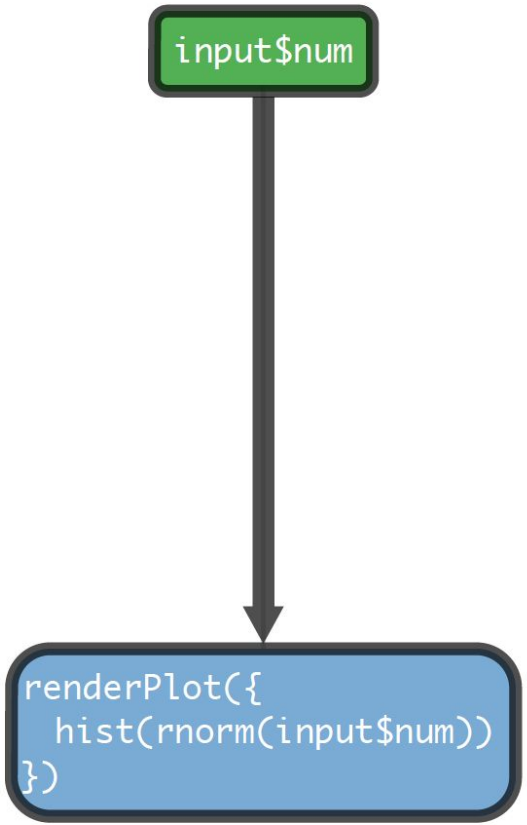
Based on bit.ly/shiny-quickstart-1 by RStudio

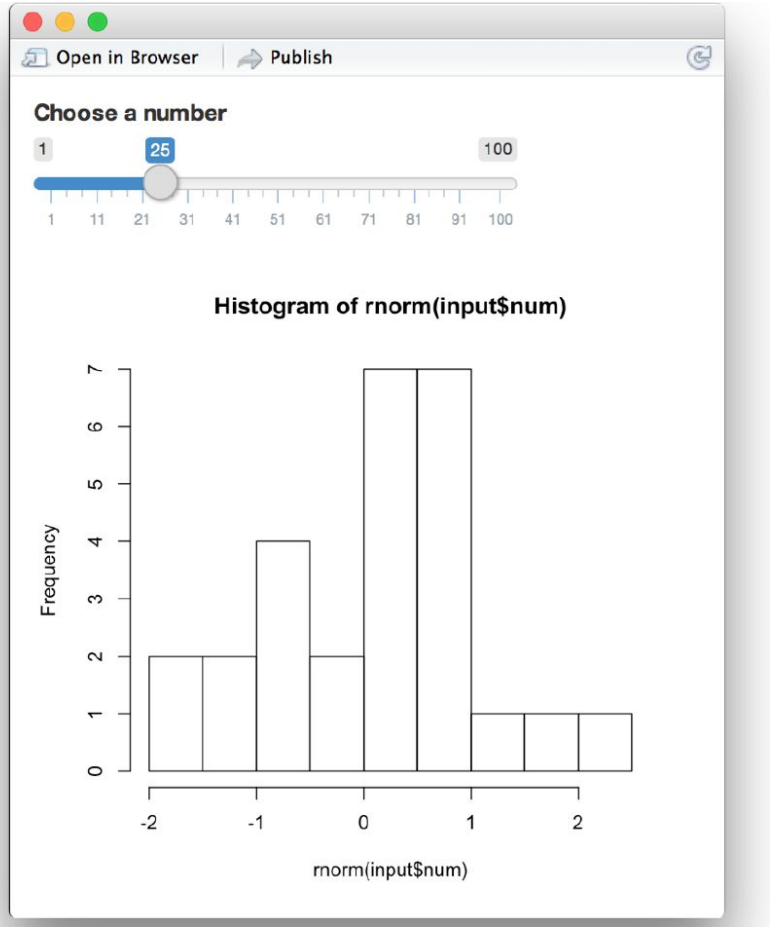
Reactivity 101

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

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

Based on bit.ly/shiny-quickstart-1 by RStudio





Based on bit.ly/shiny-quickstart-1 by RStudio

Recap: Server

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

```
output$hist <-
```

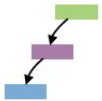
1. Save the output that you build to **output$**

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

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

```
input$num
```

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

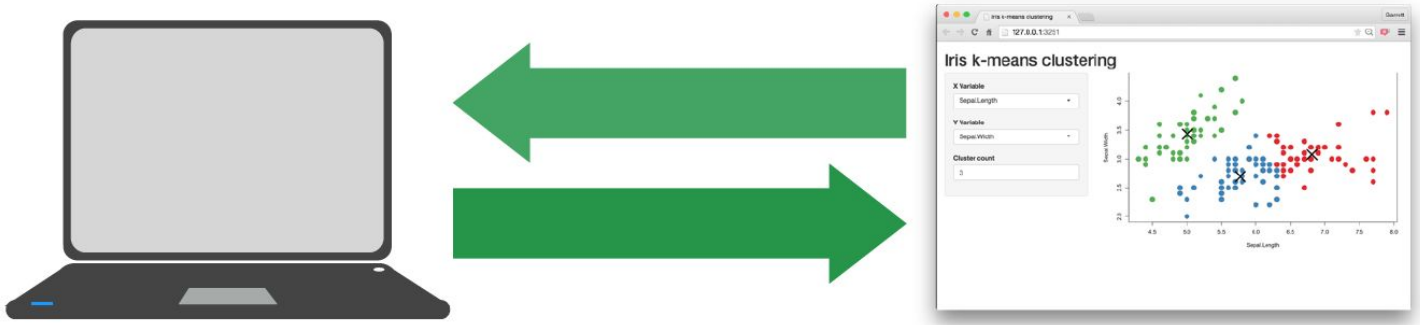


Create reactivity by using **Inputs** to build **rendered Outputs**

Based on bit.ly/shiny-quickstart-1 by RStudio

Share your app

Every Shiny app is maintained by a computer running R



Based on bit.ly/shiny-quickstart-1 by RStudio

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.

Based on bit.ly/shiny-quickstart-1 by RStudio

Two file apps

```
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)
```

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

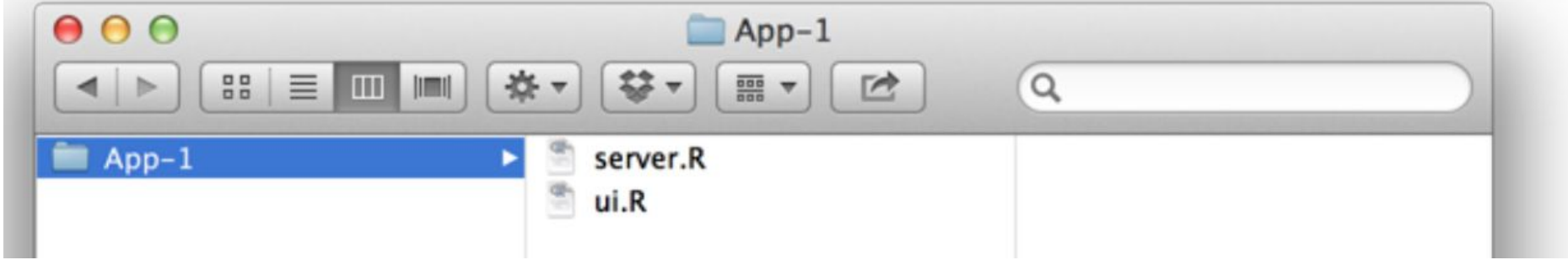
```
# server.R
library(shiny)
function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
}
```

Based on bit.ly/shiny-quickstart-1 by RStudio

Two file apps

One directory with two files:

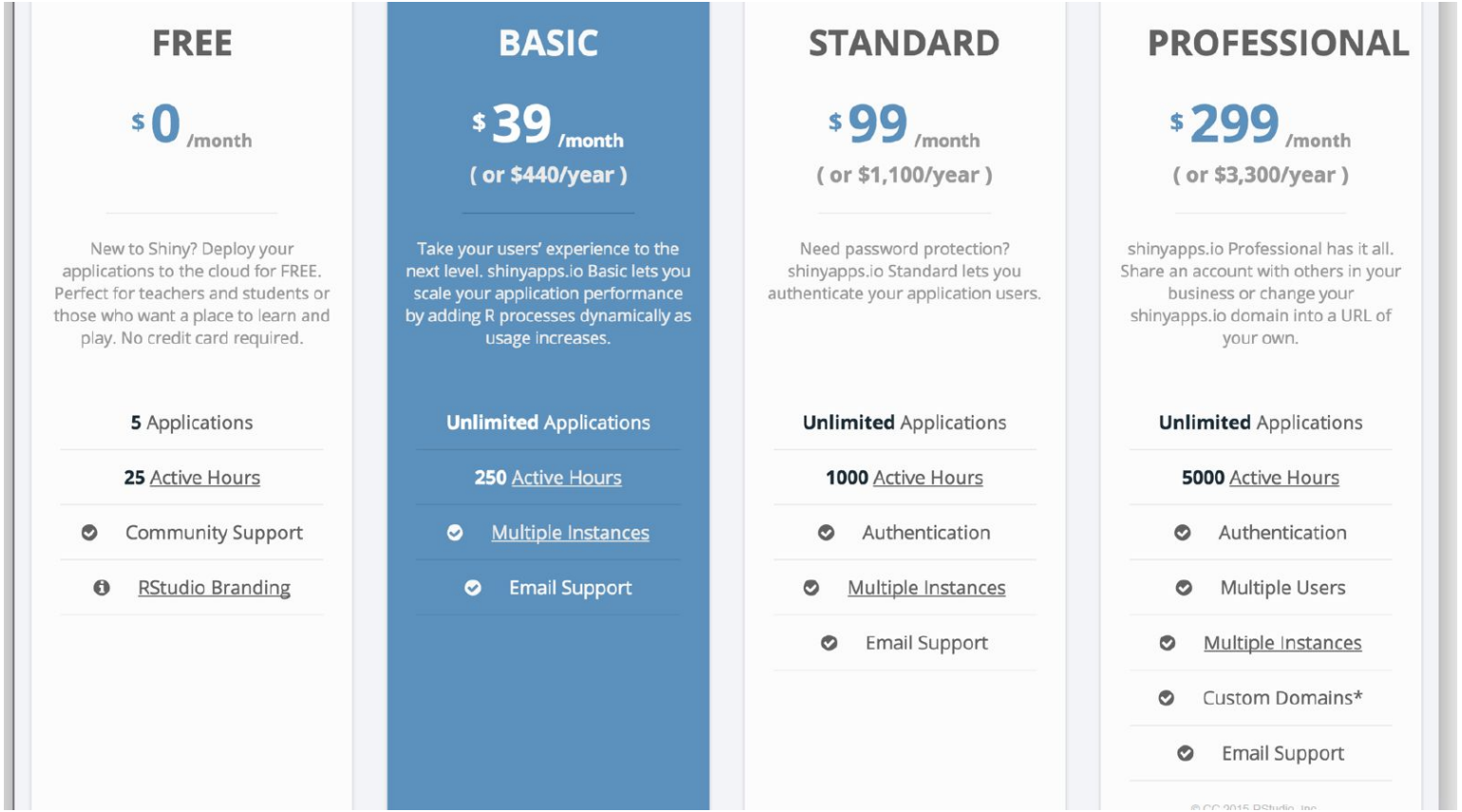
- server.R
- ui.R



Based on bit.ly/shiny-quickstart-1 by RStudio

Share your app with shinyapps.io

Shinyapps.io plans



FREE

$0 /month

New to Shiny? Deploy your applications to the cloud for FREE. Perfect for teachers and students or those who want a place to learn and play. No credit card required.

- **5** Applications
- **25** Active Hours
- Community Support
- RStudio Branding

BASIC

$39 /month

(or $440/year)

Take your users' experience to the next level. shinyapps.io Basic lets you scale your application performance by adding R processes dynamically as usage increases.

- **Unlimited** Applications
- **250** Active Hours
- Multiple Instances
- Email Support

STANDARD

$99 /month

(or $1,100/year)

Need password protection? shinyapps.io Standard lets you authenticate your application users.

- **Unlimited** Applications
- **1000** Active Hours
- Authentication
- Multiple Instances
- Email Support

PROFESSIONAL

$299 /month

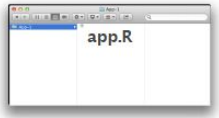
(or $3,300/year)

shinyapps.io Professional has it all. Share an account with others in your business or change your shinyapps.io domain into a URL of your own.

- **Unlimited** Applications
- **5000** Active Hours
- Authentication
- Multiple Users
- Multiple Instances
- Custom Domains*
- Email Support

Based on bit.ly/shiny-quickstart-1 by RStudio

Recap: Sharing



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

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

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

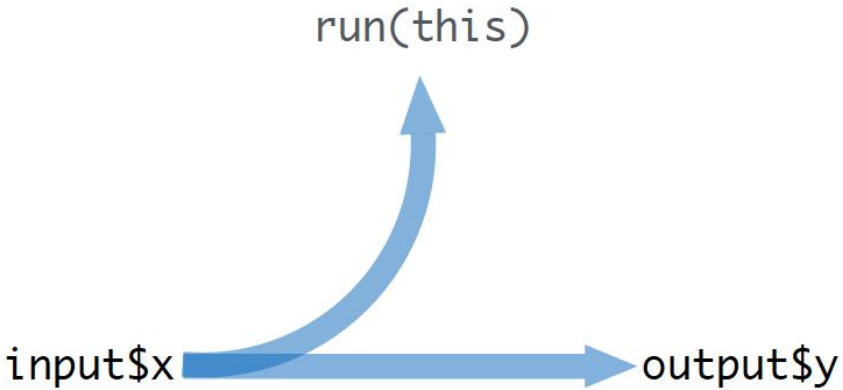
```
library(shinyapps)
```

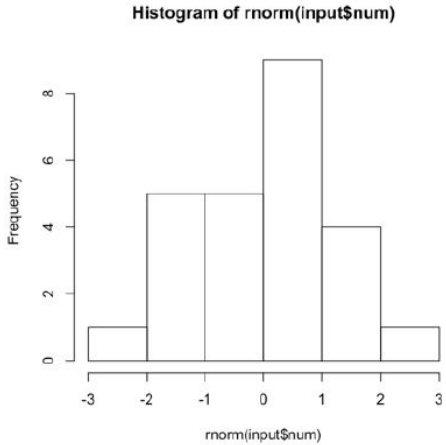
2. Install the **shinyapps** package

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

Based on bit.ly/shiny-quickstart-1 by RStudio

Intro to Reactivity





Based on bit.ly/shiny-quickstart-1 by RStudio

Reactive values work together with reactive functions.
You cannot call a reactive value from outside of one.

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

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

Based on bit.ly/shiny-quickstart-1 by RStudio

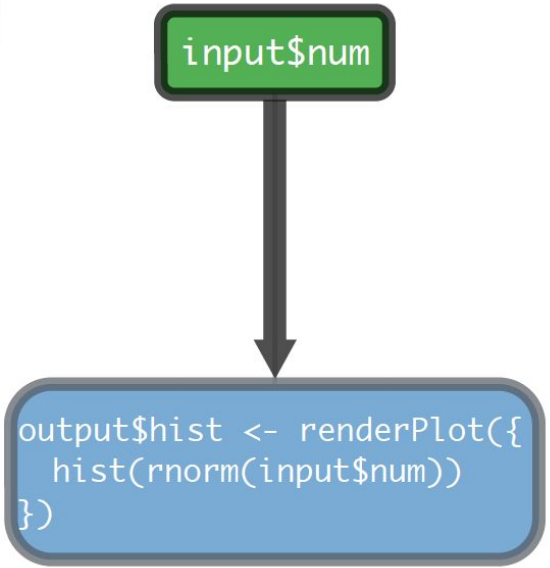
```
library(shiny)

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

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

}

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

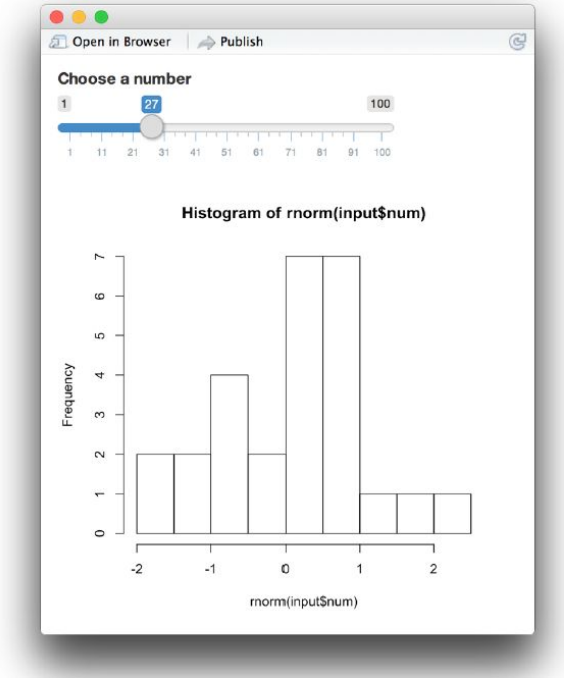
```
 Error in .getReactiveEnvironment()
$currentContext() :
  Operation not allowed without an
active reactive context. (You tried
to do something that can only be done
from inside a reactive expression or
observer.)
```

Based on bit.ly/shiny-quickstart-1 by RStudio

Think of reactivity in R as a two step process

1 Reactive values notify
the functions that use them
when they become invalid



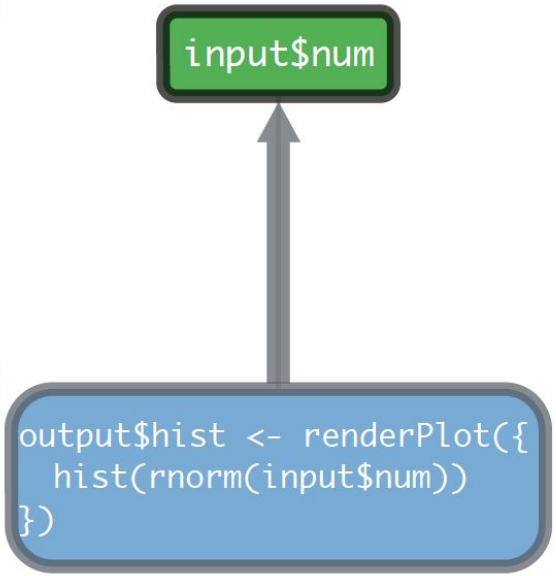


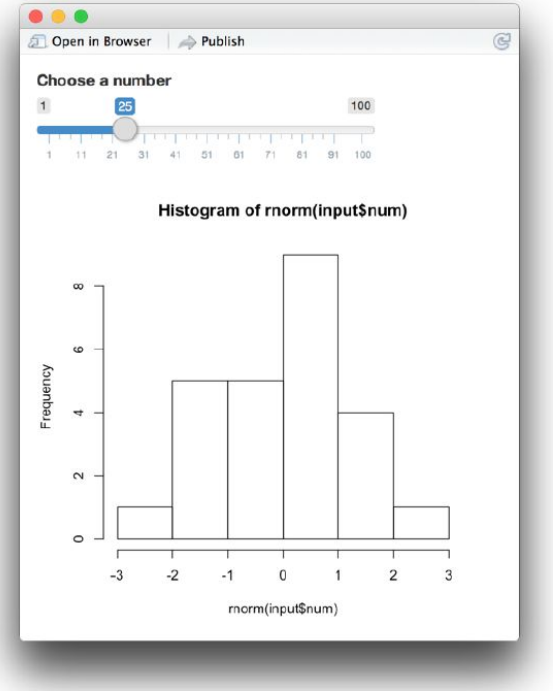
Based on bit.ly/shiny-quickstart-1 by RStudio

Think of reactivity in R as a two step process

1 Reactive values notify
the functions that use them
when they become invalid

2 The objects created by
reactive functions respond
(different objects respond differently)





Based on bit.ly/shiny-quickstart-1 by RStudio

Recap: Reactive values

Reactive values act as the data streams that flow through your app.

The **input** list is a list of reactive values. The values show the current state of the inputs.

You can only call a reactive value from a function that is designed to work with one

Reactive values notify. The objects created by **reactive functions respond.**

Based on bit.ly/shiny-quickstart-1 by RStudio

Reactive functions

1 Use a code chunk to build (and rebuild) an object

- **What code** will the function use?

2 The object will respond to changes in a set of reactive values

- **Which reactive values** will the object respond to?

Based on bit.ly/shiny-quickstart-1 by RStudio

Render functions build output to display in the app

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

Based on bit.ly/shiny-quickstart-1 by RStudio

render*()

Builds reactive output to display in UI

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

object will respond to *every reactive value in the code*

code used to build (and rebuild) object

Based on bit.ly/shiny-quickstart-1 by RStudio

render*()

Builds reactive output to display in UI

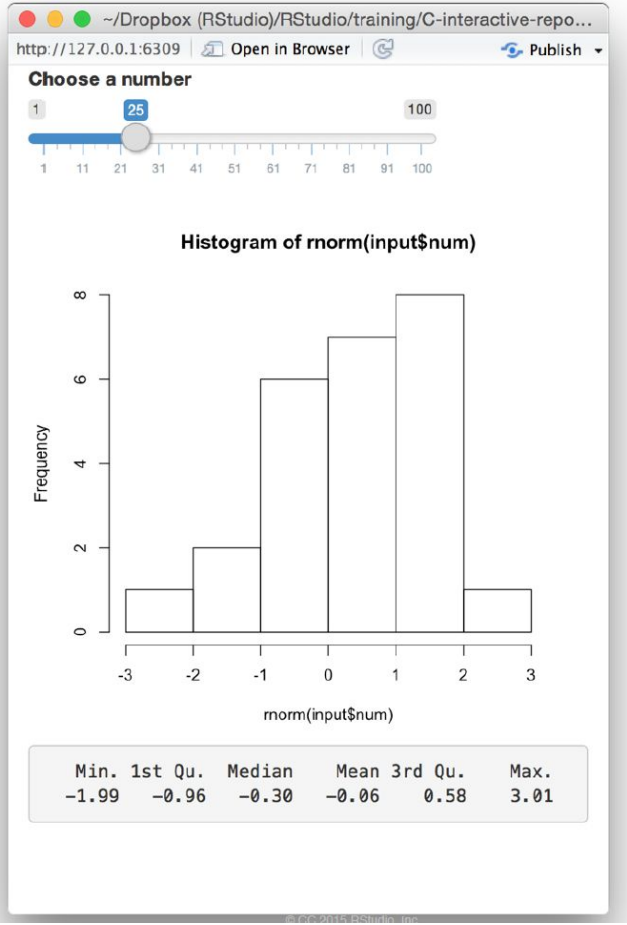
```
renderPlot(  { hist(rnorm(input$num)) })
```

When notified that it is invalid, the object created by a render*() function will rerun the entire block of code associated with it

Based on bit.ly/shiny-quickstart-1 by RStudio

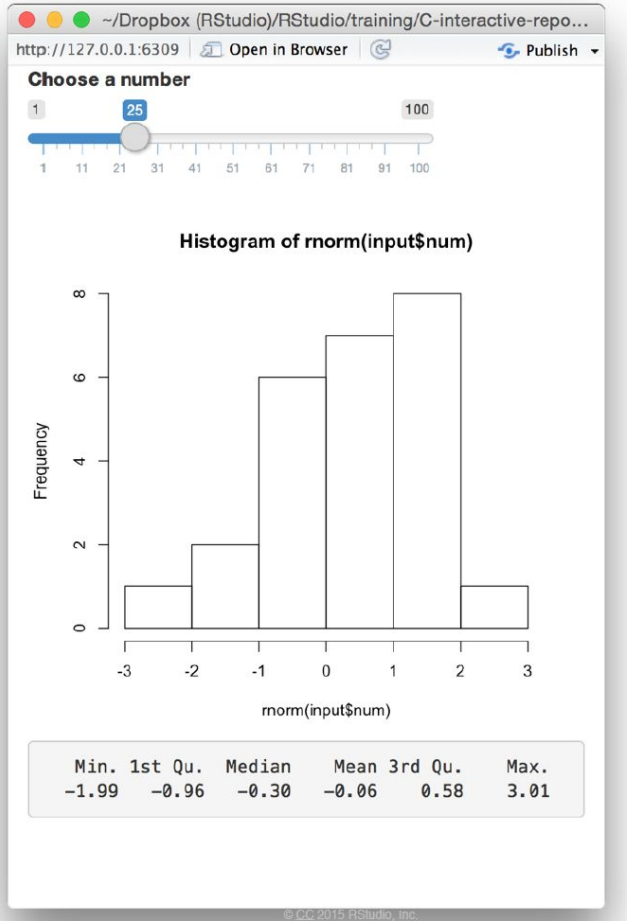
Modularize code with reactive()

```
# 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)
```



Based on bit.ly/shiny-quickstart-1 by RStudio

```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)
```



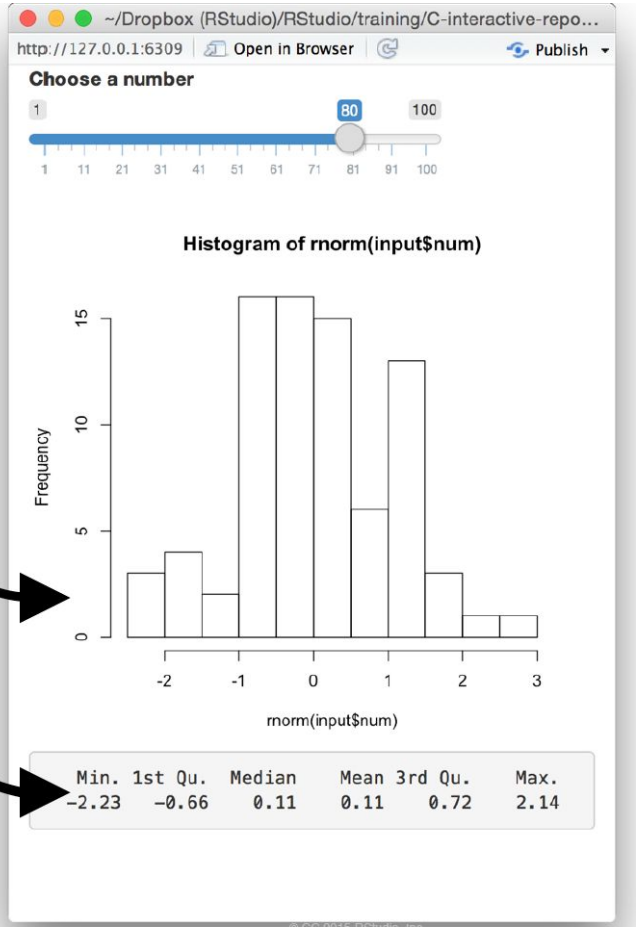
Based on bit.ly/shiny-quickstart-1 by RStudio

```
input$num
```

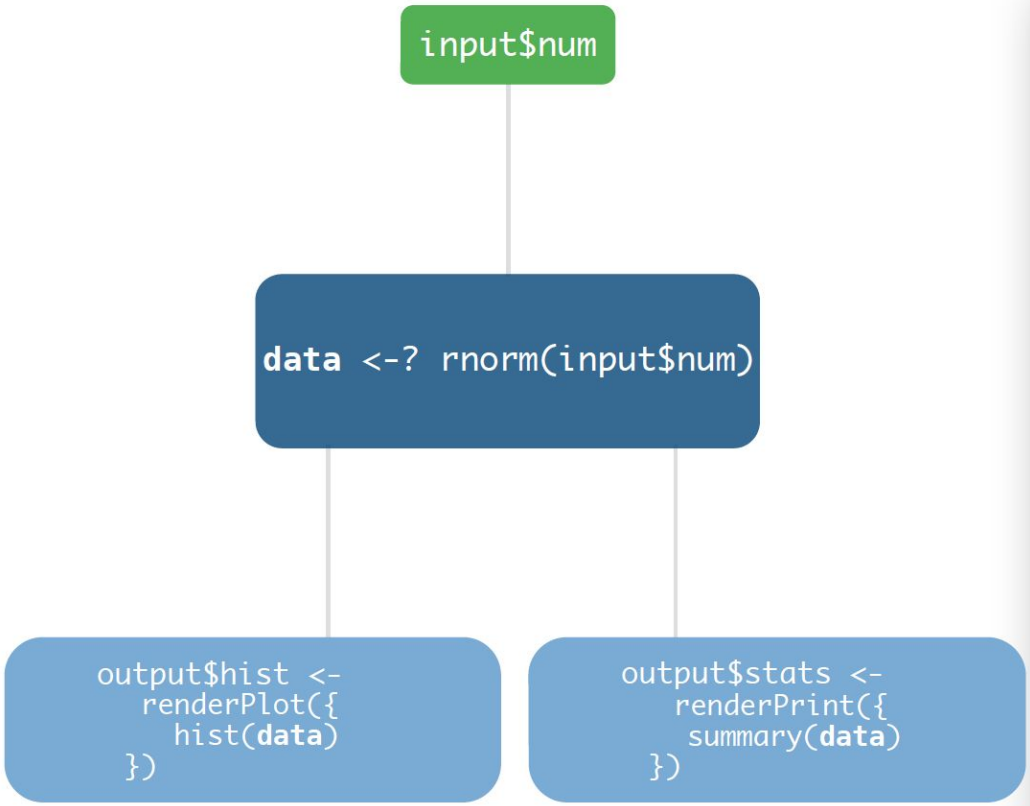
Can these describe the same data?

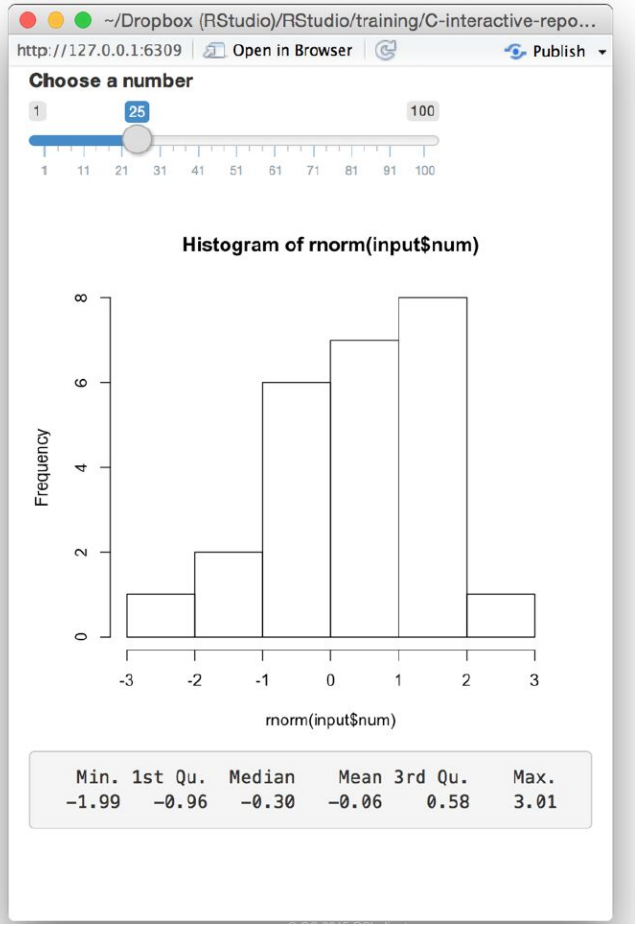
```
output$hist <-
  renderPlot({
    hist(rnorm(input$num))
  })
```

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



Based on bit.ly/shiny-quickstart-1 by RStudio





Based on bit.ly/shiny-quickstart-1 by RStudio

reactive()

Builds a reactive object (reactive expression)

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

object will respond to *every reactive value in the code*

code used to build (and rebuild) object

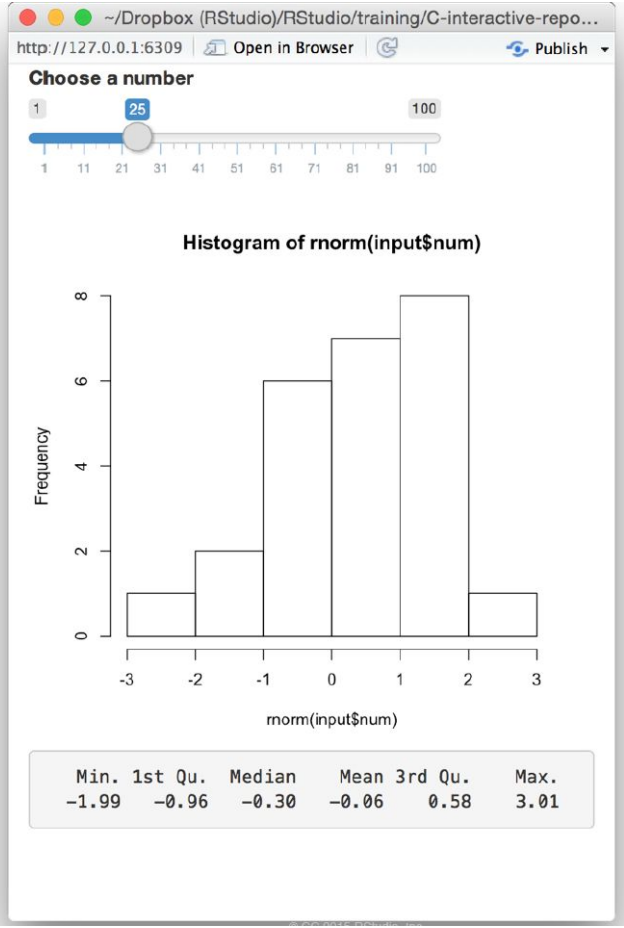
Based on bit.ly/shiny-quickstart-1 by RStudio

A reactive expression is special in two ways

```
data()
```

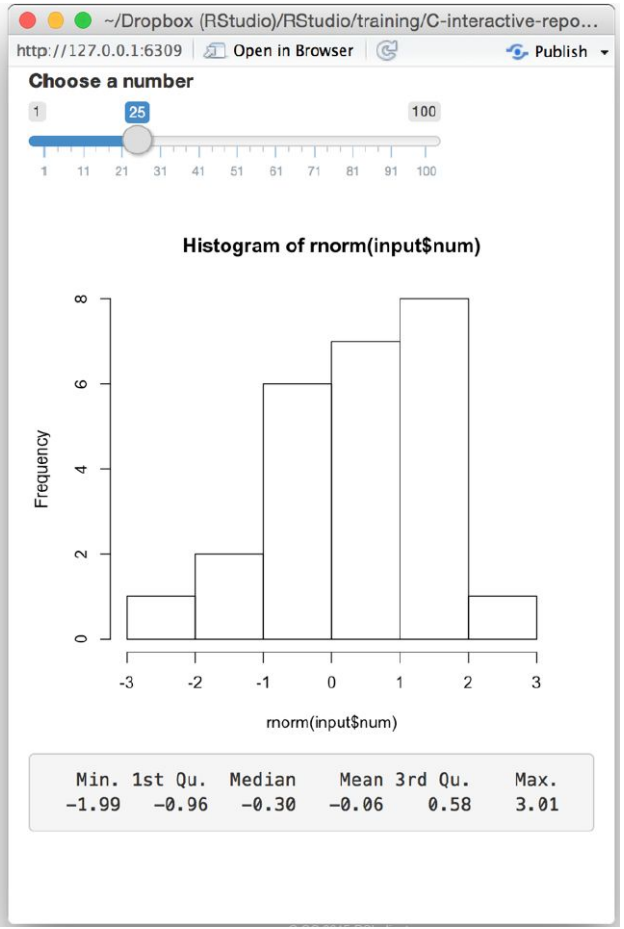
1. You call a reactive expression like a function

```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)
```



Based on bit.ly/shiny-quickstart-1 by RStudio

```
# 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)
```



Based on bit.ly/shiny-quickstart-1 by RStudio

A reactive expression is special in two ways

```
data()
```

1 You call a reactive expression like a function

2 Reactive expressions **cache** their values
(the expression will return its most recent value,
unless it has become invalidated)

Based on bit.ly/shiny-quickstart-1 by RStudio

Recap: reactive()

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

reactive() makes an **object to use** (in downstream code)

Reactive expressions are themselves **reactive**. Use them to modularize your apps.

`data()`

Call a reactive expression like a **function**

2

Reactive expressions **cache** their values to avoid unnecessary computation

Based on bit.ly/shiny-quickstart-1 by RStudio

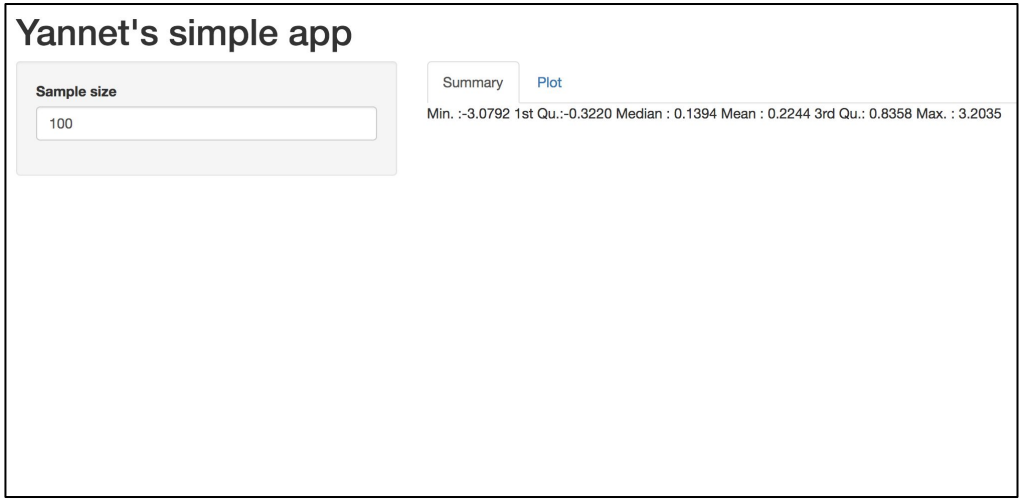
Reference

- https://vimeo.com/rstudioinc/review/131218530/212d8a5a7a/

Your job

- Make an app with two or more panels
  - Search for **mainPanel**, **tabsetPanel, sidebarPanel**
  - Example below
  - https://shiny.rstudio.com/reference/shiny/latest/tabsetPanel.html

```
mainPanel(
   tabsetPanel(
     tabPanel("Summary", textOutput("textDisplay")),
     tabPanel("Trend", plotOutput("trend"))
   )
 )
```



https://shiny.rstudio.com/tutorial/lesson1/