Shiny & reactivity

Hadley Wickham
@hadleywickham
Chief Scientist, RStudio

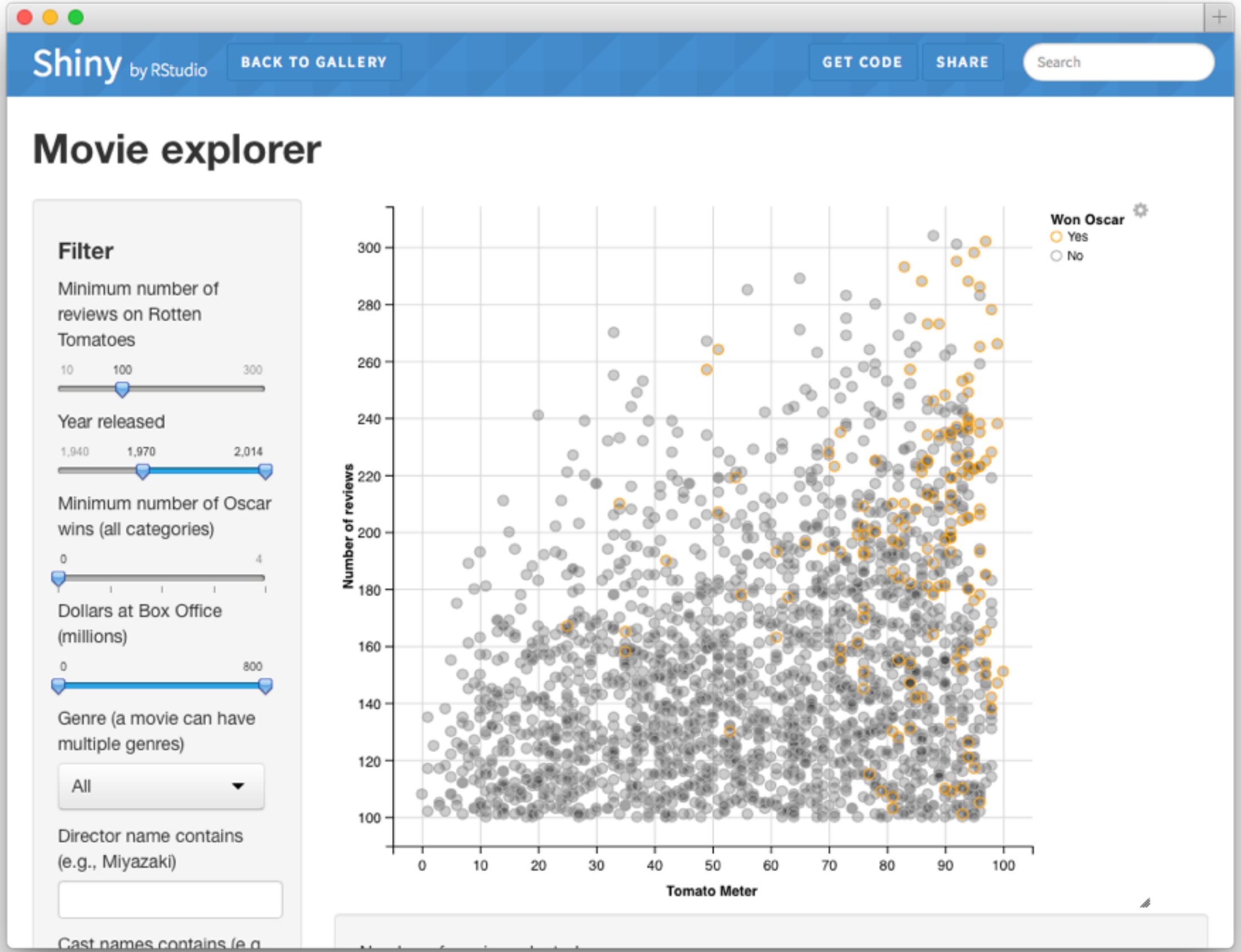
 July 2015

1. Motivation
2. Shiny basics
3. User interface
4. Inputs & outputs
5. Shiny + ggvis
6. Reactivity
7. HTML widgets
8. shinyapps.io

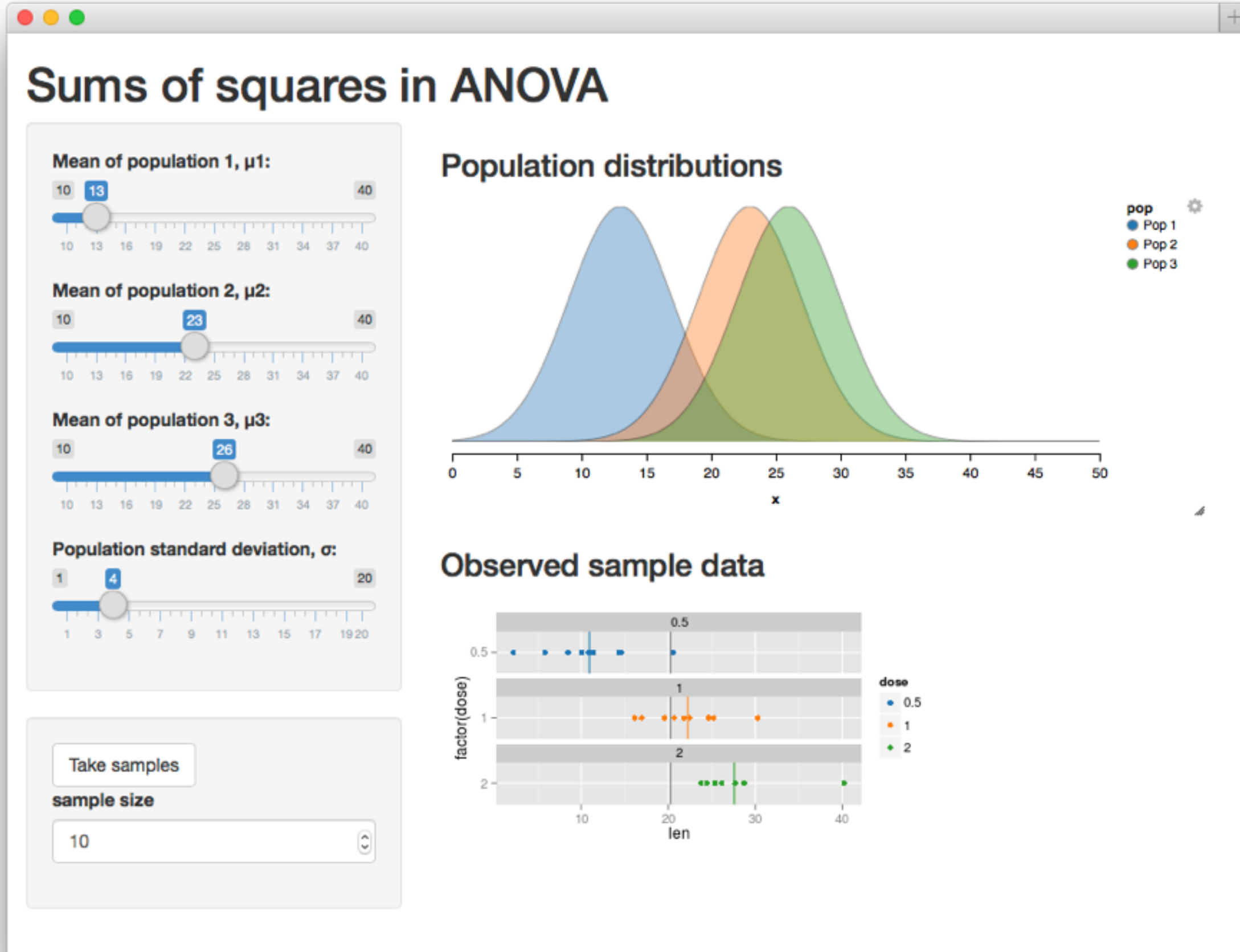
Motivation

Why Shiny?

- In a report, you pose and answer questions. In a shiny app, you allow the reader to pose (constrained) questions
- Tie ggvis into richer set of interactive components and other parts of R
- Makes you look awesome



http://shiny.rstudio.com/gallery/movie-explorer.html



https://gallery.shinyapps.io/anova_shiny_rstudio/

shinyapps.io

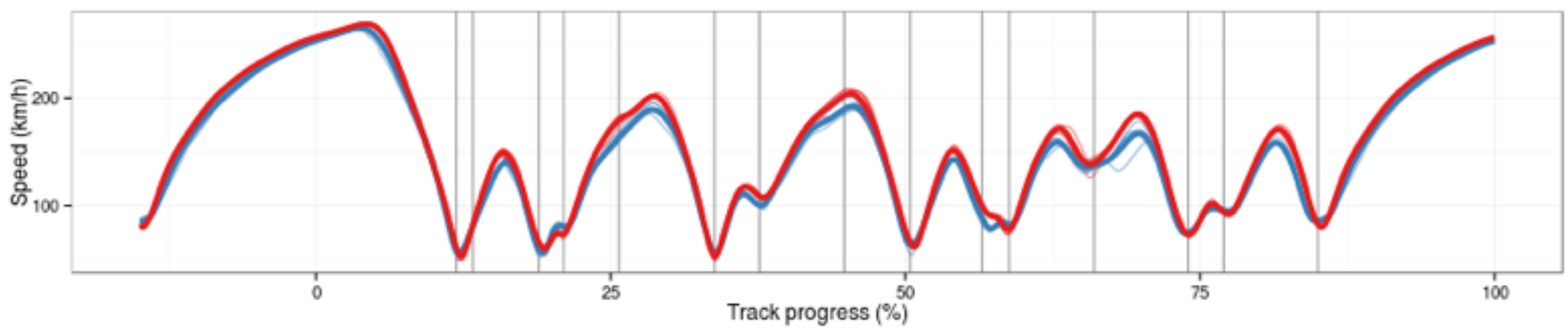
Powered by RStudio

Where are the speeds of the two riders the most different?

Click and drag in the plot below to select a portion of the track to examine more closely

Racer Journalist

ggplot2



Racing Line

ggvis

Rider Stats

Averaged over all laps on selected section

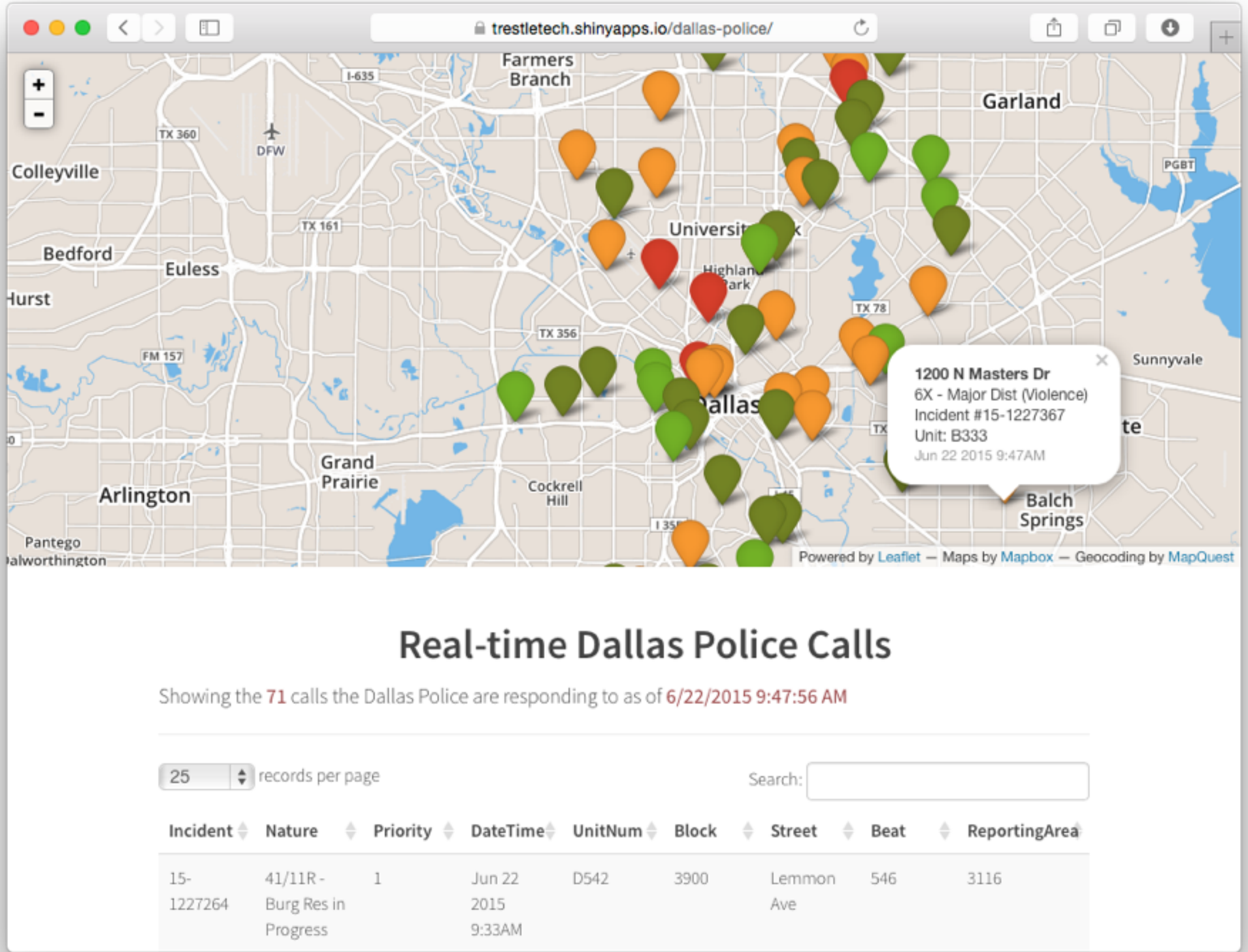
Racer leads by

4.53

seconds

	Racer	Journalist
Time (sec):	104.07	108.6
Avg. speed (km/h):	130	124
Max. speed (km/h):	269	265
Min. speed (km/h):	51	53

https://cwickham.shinyapps.io/motorcycle



https://trestletech.shinyapps.io/dallas-police/

Basics

Open

shiny.Rproj

```r
# install.packages("shiny")
library(shiny)

ui <- fluidPage("Hello World")
server <- function(input, output) {}

runApp(shinyApp(ui, server))
```

```
# install.packages("shiny")
library(shiny)
```

User interface

```
ui <- fluidPage("Hello World")
server <- function(input, output) {}
```

Computation

```
runApp(shinyApp(ui, server))
```

Use stop to quit

Console ~/Dropbox (RStudio)/rstudio-training/15-uzurich/2-shiny/

```
> # install.packages("shiny")
> library(shiny)
>
> ui <- fluidPage("Hello World")
> server <- function(input, output) {}
>
> runApp(shinyApp(ui, server))

Listening on http://127.0.0.1:5045
```

R session now runs shiny app

```r
library(shiny)

ui <- fluidPage(
  sliderInput("number", "Pick a number", 1, 10, value = 5),
  p("You picked: ", textOutput("result", inline = TRUE))
)

server <- function(input, output) {
  message("Initialising")
  output$result <- renderText({
    message("Updating")
    input$number
  })
}

runApp(shinyApp(ui, server))
```

Input

```
library(shiny)

ui <- fluidPage(
  sliderInput("number", "Pick a number", 1, 10, value = 5),
  p("You picked: ", textOutput("result", inline = TRUE))
)

server <- function(input, output) {
  message("Initialising")
  output$result <- renderText({
    message("Updating")
    runif(input$number)
  })
}

runApp(shinyApp(ui, server))
```

Output

```
library(shiny)

ui <- fluidPage(
  sliderInput("number", "Pick a number", 1, 10, value = 5),
  p("You picked: ", textOutput("result", inline = TRUE))
)

server <- function(input, output) {
  message("Initialising")
  output$result <- renderText({
    message("Updating")
    runif(input$number)
  })
}

runApp(shinyApp(ui, server))
```

Reactivity

```
library(shiny)

ui <- fluidPage(
  sliderInput("number", "Pick a number", 1, 10, value = 5),
  p("You picked: ", textOutput("result", inline = TRUE))
)

server <- function(input, output) {
  message("Initialising")
  output$result <- renderText({
    message("Updating")
    runif(input$number)
  })
}

runApp(shinyApp(ui, server))
```

Run once

Run many times

User interface

```
library(shiny)

ui <- fluidPage(
  titlePanel("This is a title"),
  sidebarLayout(
    sidebarPanel("This is a sidebar"),
    mainPanel("This is the main panel")
  )
```

Just HTML!

```
ui
runApp(shinyApp(ui, function(input, output) {}))
```

http://rstudio.github.io/shinydashboard

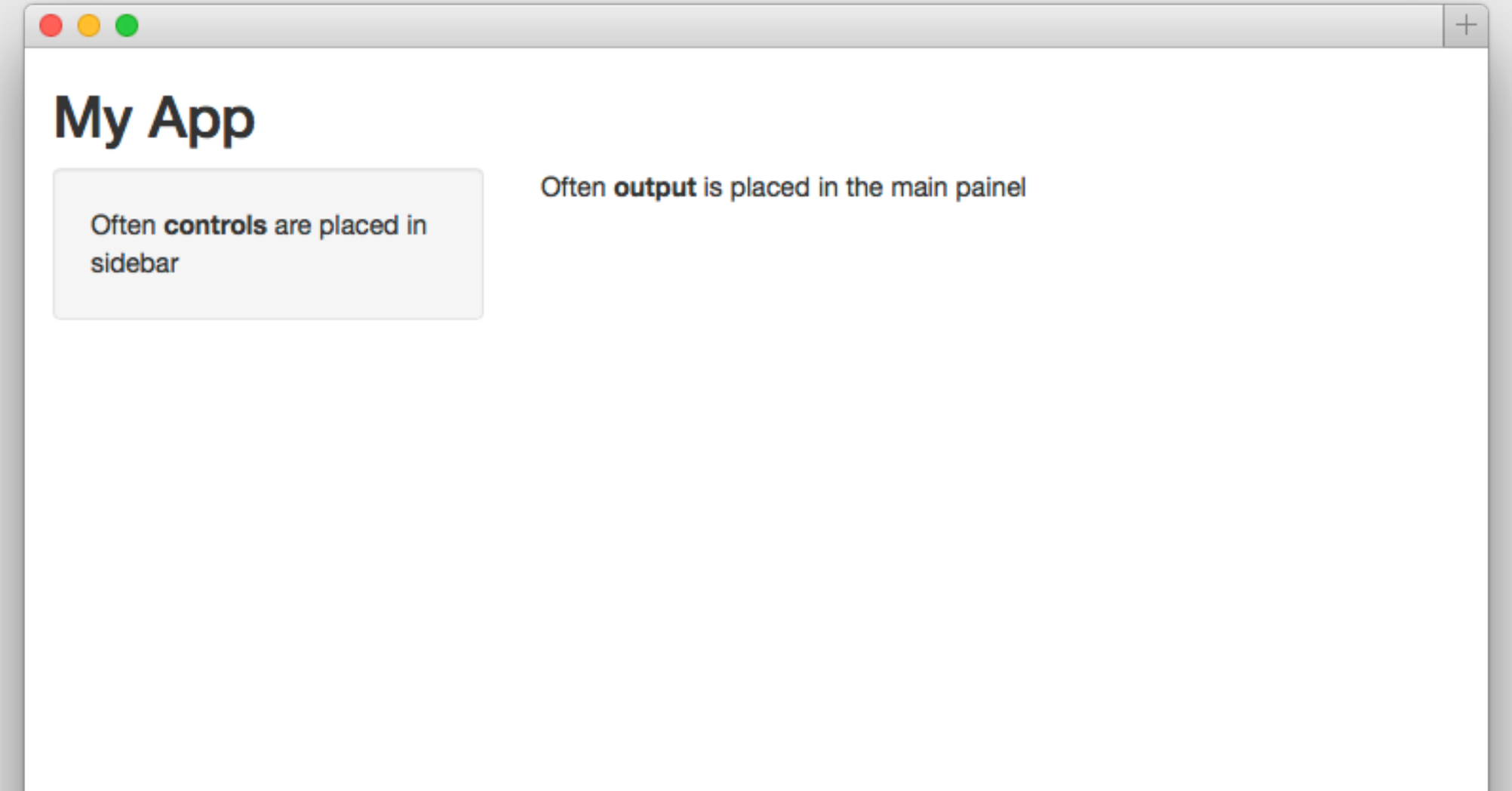
`h1, h2, h3...`	Headings
p	Paragraph
`strong`	Bold text
a	A link

The tree structure of HTML is mimicked by the tree of function calls

```
library(shiny)

ui <- fluidPage(
  h1("This is a title"),
  p("It is ", strong("important"),
    " to understand the basics of HTML."
  ),
  p("A link has an ",
    a(href = "http://shiny.rstudio.com",
    " href attribute"),
    "."
  )
)
ui
runApp(shinyApp(ui, function(input, output) {}))
```

Your turn

Recreate this page.



Inputs & Outputs

Input

```
library(shiny)

ui <- fluidPage(
  sliderInput("number", "Pick a number", 1, 10, value = 5),
  p("You picked: ", textOutput("result", inline = TRUE))
)

server <- function(input, output) {
  message("Initialising")
  output$result <- renderText({
    message("Updating")
    runif(input$number)
  })
}

runApp(shinyApp(ui, server))
```

Define with `input` function

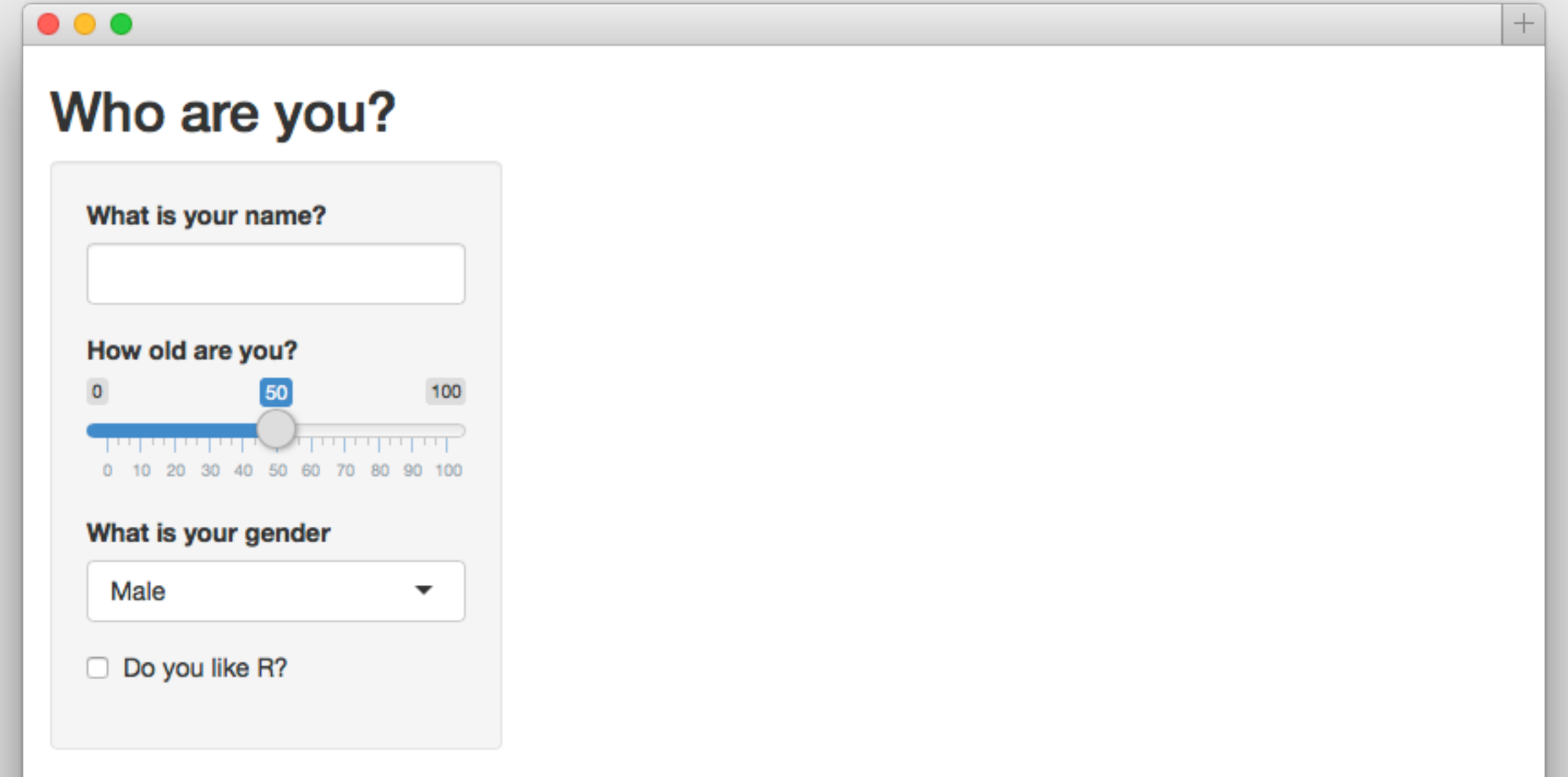
Get value with `input$`

`textInput`	Free form text	`character`
`numericInput`	A number	`numeric`
`sliderInput`	Numeric slider	`numeric`
`selectInput`	Drop down of options	`character`
`checkboxInput`	Check box	`logical`

Hint: type `shiny::input<Tab>`

Your turn

Recreate this page.



```
# As with any complex task, always easiest to solve
# problems in isolation. So before you write the shiny
# code to connect input and output, first do it with
# ordinary variables.
#
# Your turn:
description <- function(name, age, gender, likes_r) {

}

# Example output (e.g.):
description("Hadley", 35, "Male", TRUE)
# Hadley is 35 year old male who likes R.
```

```r
description <- function(name, age, gender, likes_r) {
  gender_desc <- switch(gender,
    Male = "man",
    Female = "female",
    Other = "person"
  )

  paste0(name, " is a ", age, " year old ", gender_desc,
    " who ", if (likes_r) "likes" else "doesn't like", " R")
}
description("Hadley", 35, "Male", TRUE)
```

Process

1. Solve a specific problem.
2. Generalise by making a function.
3. Test your function!
4. Use the function inside `server()`

Advice

- Writing complex code in shiny app is recipe for frustration.
- If it works on the command line, but not in your app, most likely cause is that input is not the type you expect.

Output

```
library(shiny)

ui <- fluidPage(
  sliderInput("number", "Pick a number", 1, 10, value = 5),
  p("You picked: ", textOutput("result", inline = TRUE))
)

server <- function(input, output) {
  message("Initialising")
  output$result <- renderText({
    message("Updating")
    runif(input$number)
  })
}

runApp(shinyApp(ui, server))
```

Define with `output` function

Set value with `output$` + `render` function

Create hole

Fill the hole

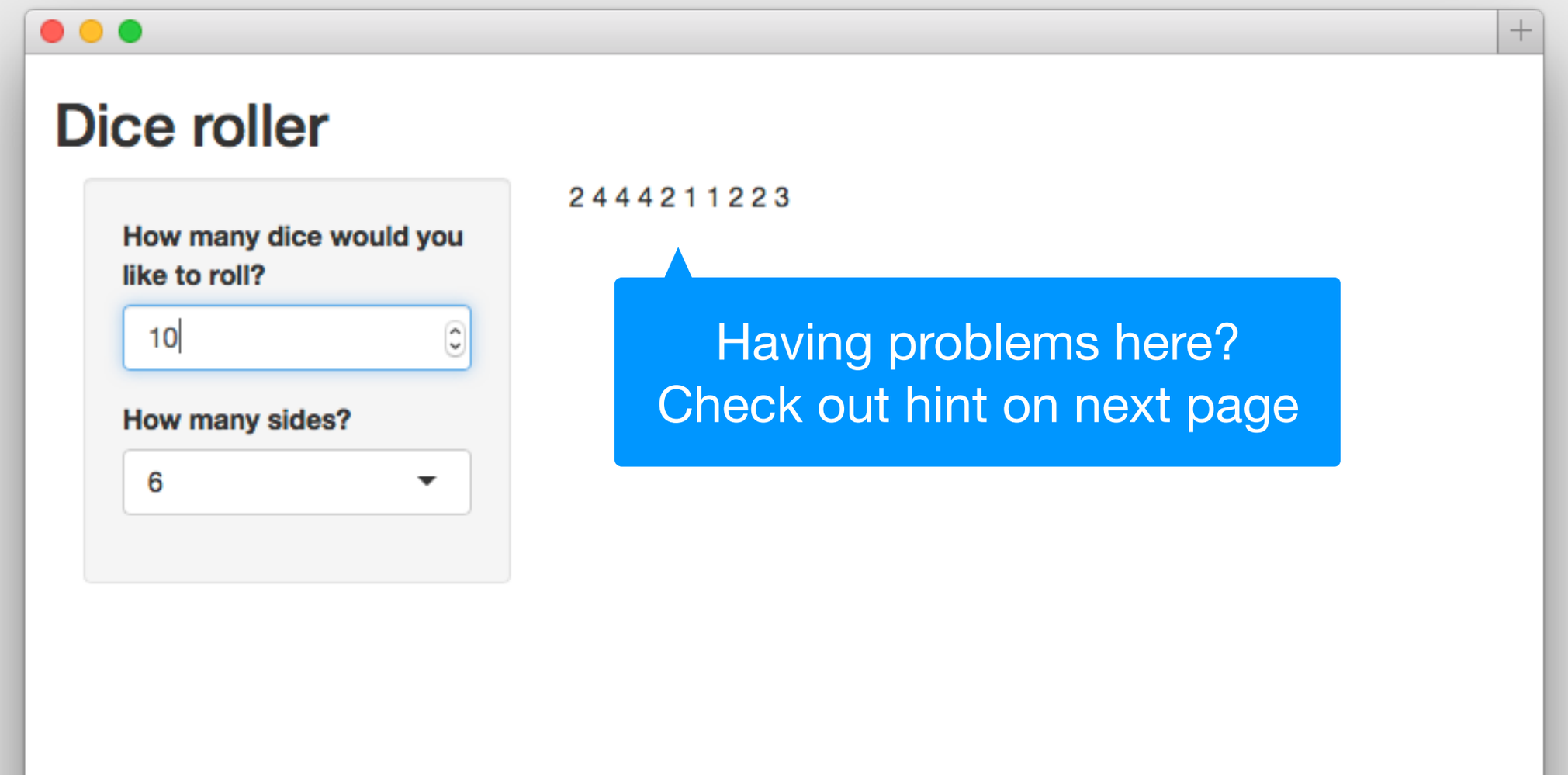
`textOutput`	`renderText`	Text
`plotOutput`	`renderPlot`	Plot
`tableOutput`	`renderTable`	Table

```
ui <- fluidPage(
  titlePanel(
    "Who are you?"
  ),
  sidebarLayout(
    sidebarPanel(
      textInput("name", "What is your name?"),
      sliderInput("age", "How old are you?", 0, 100, 50),
      selectInput("gender", "What is your gender",
         c("Male", "Female", "Other")),
      checkboxInput("like_r", "Do you like R?")
    ),
    mainPanel(
      textOutput("description")
    )
  )
)
```

```
server <- function(input, output) {
  output$description <- renderText(
    description(
      input$name, input$age,
      input$gender, input$like_r
    )
  )
}
```

Your turn

Recreate this simple app.



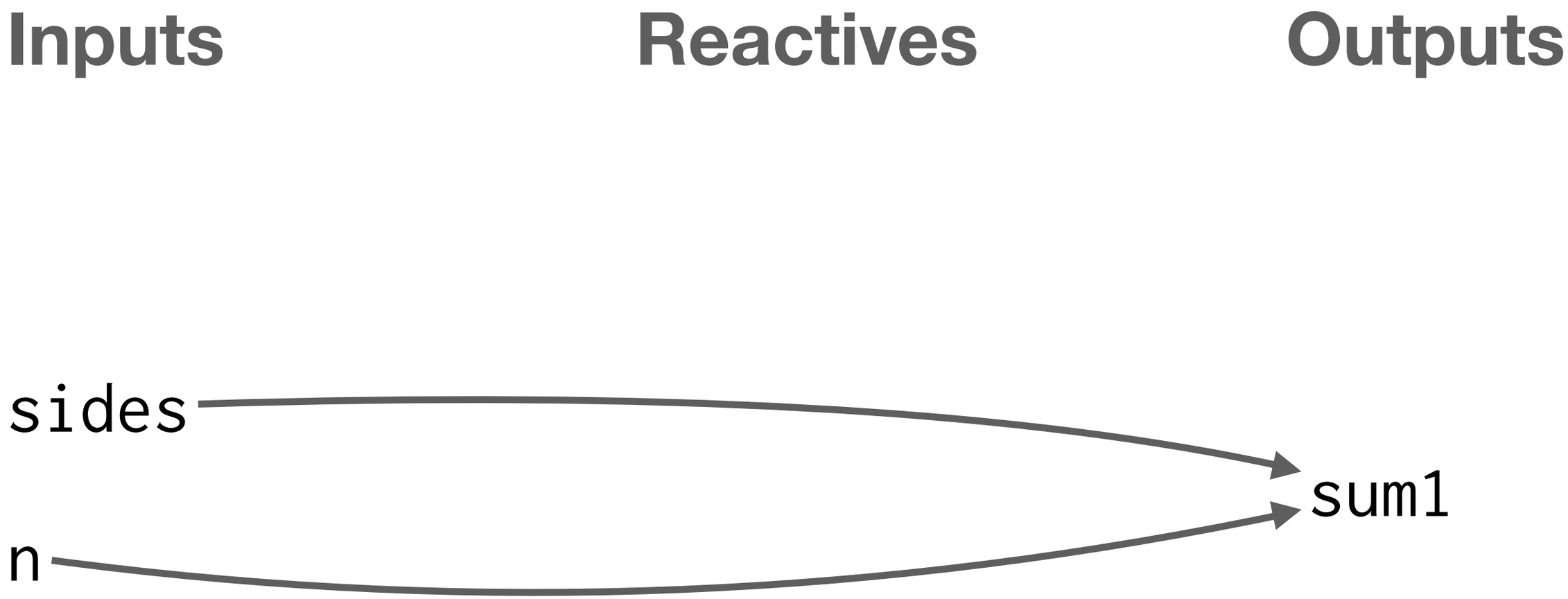
Having problems here?
Check out hint on next page

```
# Hint:
sample("6", 10, replace = TRUE)
sample(6, 10, replace = TRUE)
```

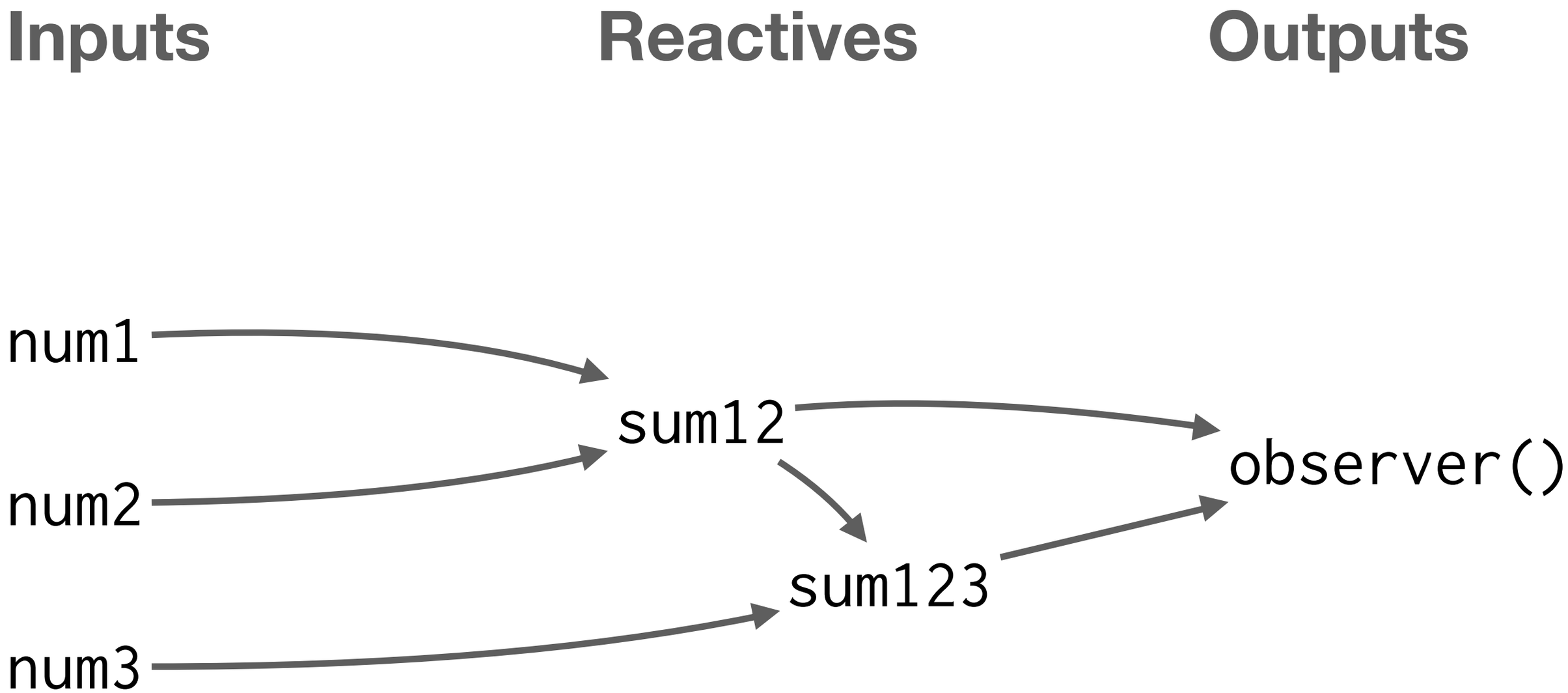
Reactivity

Motivation

- You now have everything you need to make simple apps
- Mastering reactivity gives you the tools to create richer apps
- Key idea is graph/network of calculation. When input changes do minimal recalculation to get output


Inputs
Reactives
Outputs
sides
n
sum1

```r
server <- function(input, output, session) {
  sum12 <- reactive({
    message("sum12 = num1 + num2")
    input$num1 + input$num2
  })
  sum123 <- reactive({
    message("sum123 = sum12 + num3")
    sum12() + input$num3
  })

  observe({
    sum12()
    sum123()
    message("----------------------")
  })

  output$sum12 <- renderText(sum12())
  output$sum123 <- renderText(sum123())
}

runApp(shinyApp(ui, server))
```


Inputs
Reactives
Outputs
num1
num2
num3
sum12
sum123
observer()

Challenge

Extend the dice roller app to display a bar chart and a table of results.

```
ui <- fluidPage(
  titlePanel("Dice roller"),
  sidebarPanel(
    numericInput("n", "How many dice would you like to roll?", 10, min = 1),
    selectInput("sides", "How many sides?", c(6, 12, 20))
  ),
  mainPanel(
    textOutput("rolls"),
    plotOutput("dist"),
    tableOutput("summary")
  )
)
```

```
server1 <- function(input, output) {
  rolls <- roll_die(input$n, as.numeric(input$sides))


  output$rolls <- renderText(rolls)
  output$dist <- renderPlot(plot(table(rolls)))
  output$summary <- renderTable(table(rolls))
}
```

What's wrong with this approach?

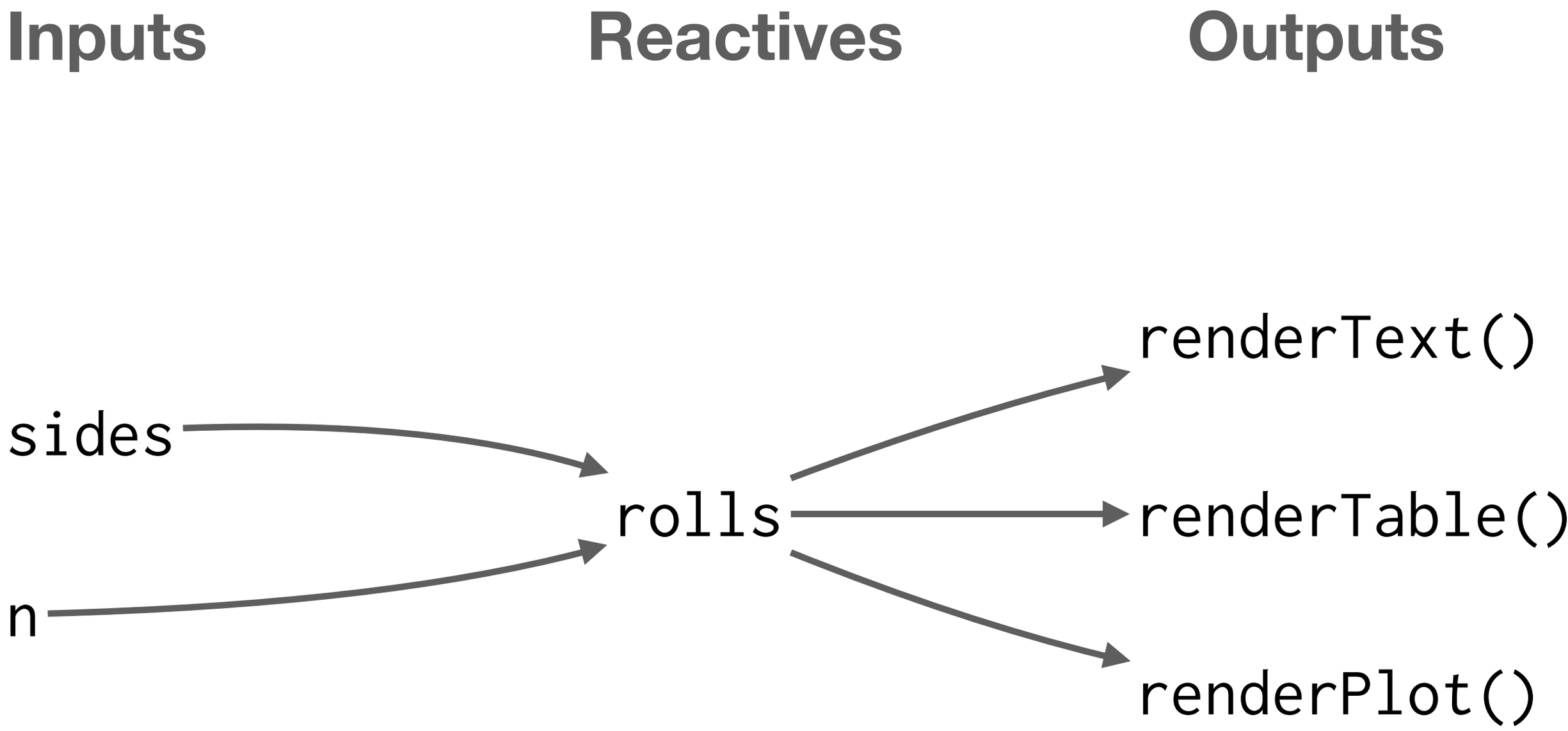
```
server2 <- function(input, output) {
  rolls <- function() {
    roll_die(input$n, as.numeric(input$sides))
  }

  output$rolls <- renderText(rolls())
  output$dist <- renderPlot(plot(table(rolls())))
  output$summary <- renderTable(table(rolls()))
}
```

```r
server3 <- function(input, output) {
  rolls <- reactive({
    roll_die(input$n, as.numeric(input$sides))
  })

  output$rolls <- renderText(rolls())
  output$dist <- renderPlot(plot(table(rolls())))
  output$summary <- renderTable(table(rolls()))
}
```

Creates a function that automatically updates when inputs change

```
server3 <- function(input, output) {
  rolls <- reactive({
    roll_die(input$n, as.numeric(input$sides))
  })

  output$rolls <- renderText(rolls())
  output$dist <- renderPlot(plot(table(rolls())))
  output$summary <- renderTable(table(rolls()))
}
```

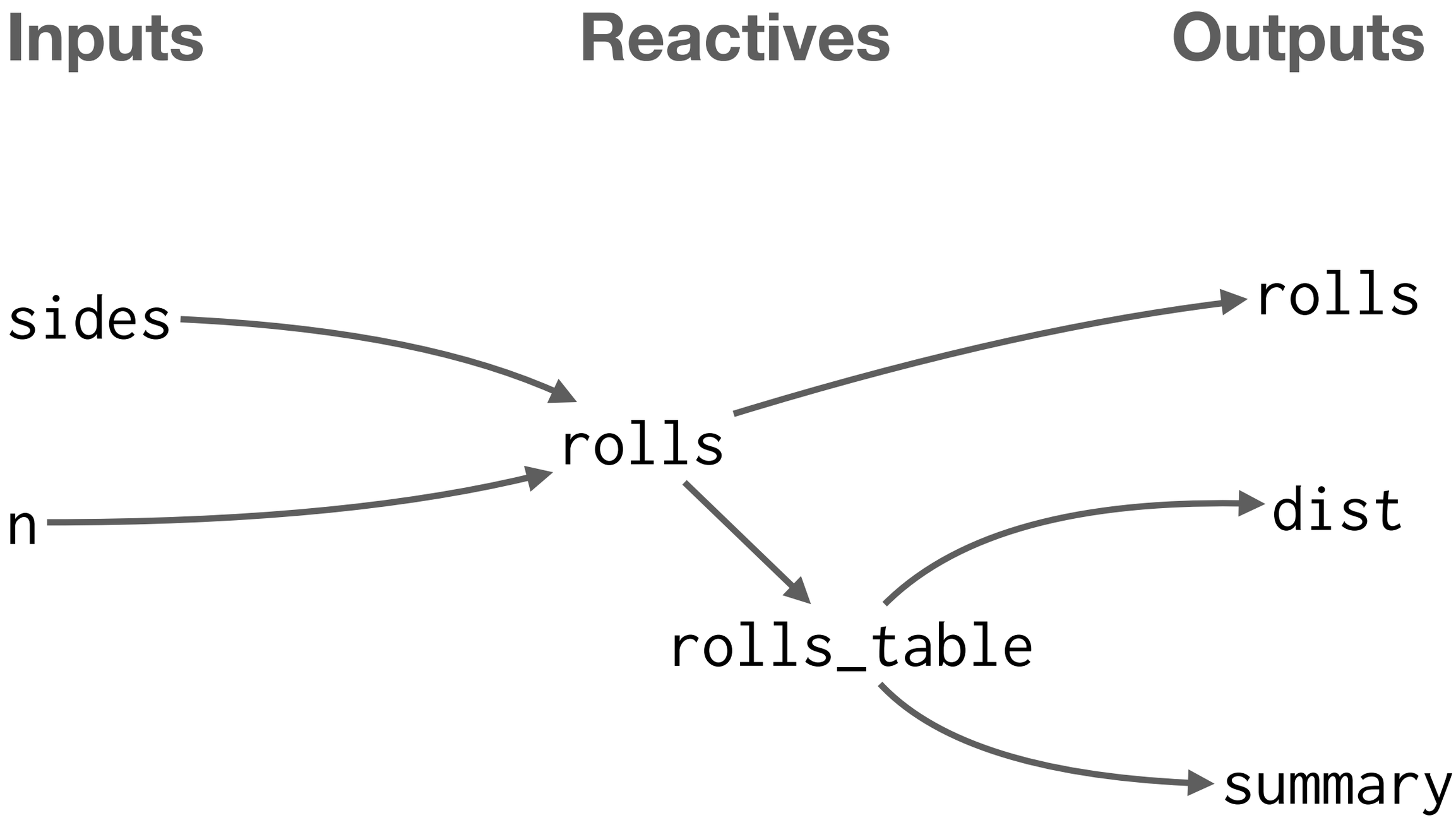
Lazily evaluated: only re-run when inputs change


Inputs
Reactives
Outputs
sides
n
rolls
renderText()
renderTable()
renderPlot()

Where is some more duplication that we could remove?

```
server3 <- function(input, output) {
  rolls <- reactive({
    roll_die(input$n, as.numeric(input$sides))
  })

  output$rolls <- renderText(rolls())
  output$dist <- renderPlot(plot(table(rolls())))
  output$summary <- renderTable(table(rolls()))
}
```

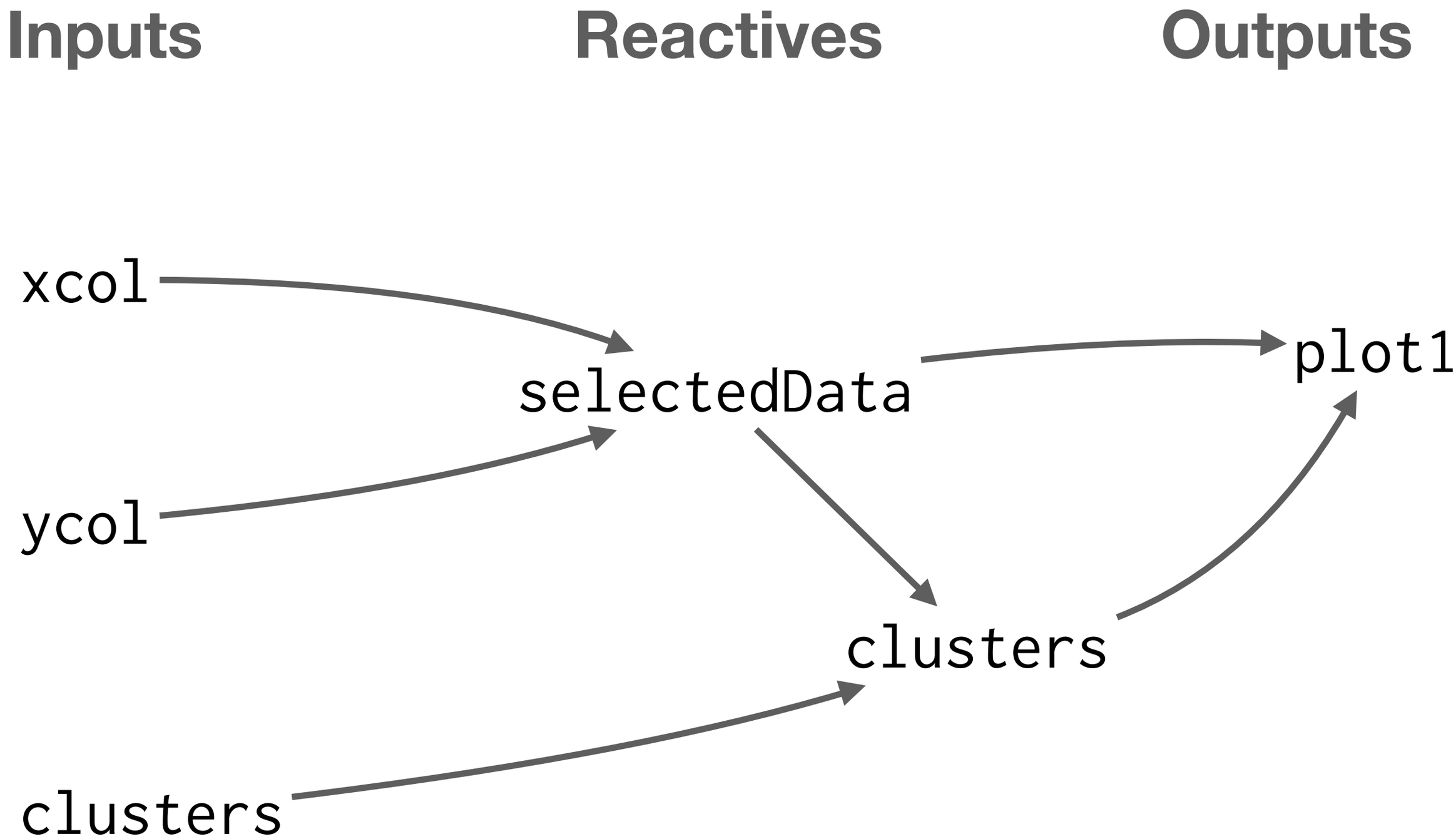
```
server4 <- function(input, output) {
  rolls <- reactive({
    roll_die(input$n, as.numeric(input$sides))
  })
  rolls_table <- reactive(table(rolls()))

  output$rolls <- renderText(rolls())
  output$dist <- renderPlot(plot(rolls_table()))
  output$summary <- renderTable(rolls_table())
}
```


Inputs
Reactives
Outputs
sides
n
rolls
rolls_table
rolls
dist
summary

Your turn

Play around with the shiny app in `kmeans.R`, and then draw the reactivity graph.

(Challenge: what other implicit input causes the plot to be redrawn?)


Inputs
Reactives
Outputs
xcol
ycol
clusters
selectedData
clusters
plot1

ggvis

```
# Step 1: get the plot working for a specific
# set of parameters

mpg %>%
  ggvis(~displ, ~hwy) %>%
  layer_points() %>%
  layer_smooths(span = 0.5)
```

```r
# Step 2: turn it into a function

draw_plot <- function(span) {
  mpg %>%
    ggvis(~displ, ~hwy) %>%
    layer_points() %>%
    layer_smooths(span = span)
}
draw_plot(0.5)
```

```
# Step 3: add it to server()
# Different pattern to other outputs!

server <- function(input, output) {
  draw_plot(reactive(input$span)) %>%
    bind_shiny("p")
}
runApp(shinyApp(ui, server))
```

Arguments must be reactives

Your turn

Adapt `6-ggvis.R` to use `layer_model_predictions()` and allow the user to select between `lm` and `MASS::rlm`.

ggvis reactivity

- Data can be reactive (`tourr.R`)
- Layer parameters can be reactive
- Layer properties can be reactive
- (But not everything might be plumbed up correctly yet)
- Or wrap entire plot in `reactive()`

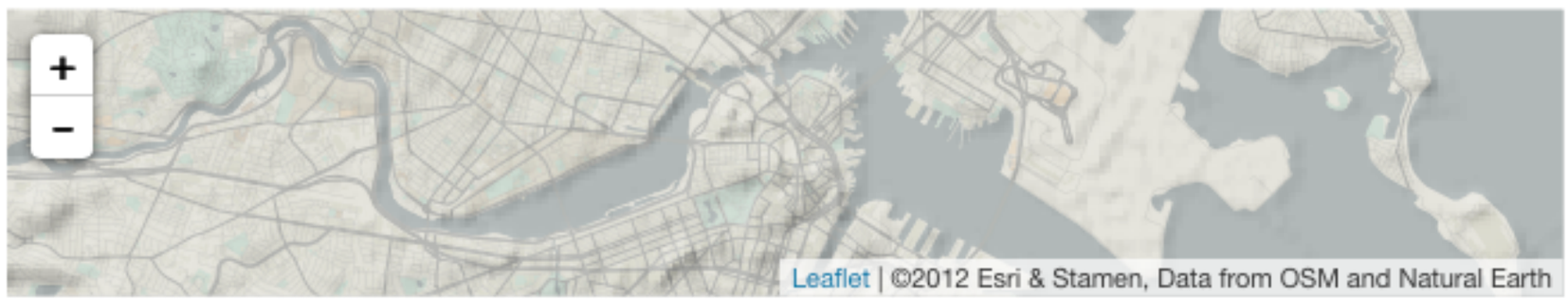
Html widgets

Generalised vs. specialised

- ggvis is a general purpose plotting toolkit
- htmlwidgets packages provide bindings to specialised js libraries
- Work in console, shiny apps & R markdown

```
m %>% addProviderTiles("Stamen.Toner")
```

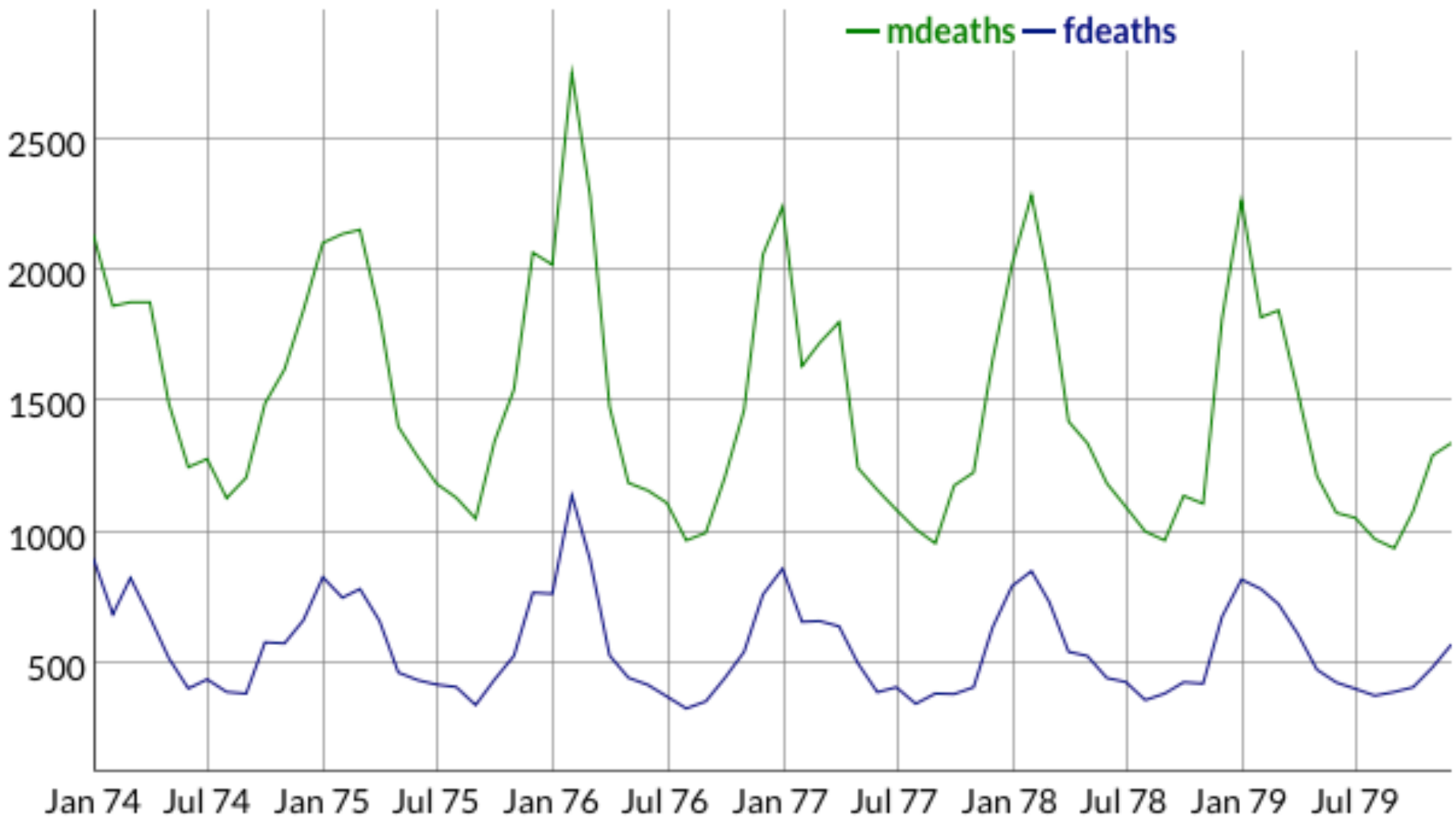

```
m %>% addProviderTiles("Acetate.terrain")
```



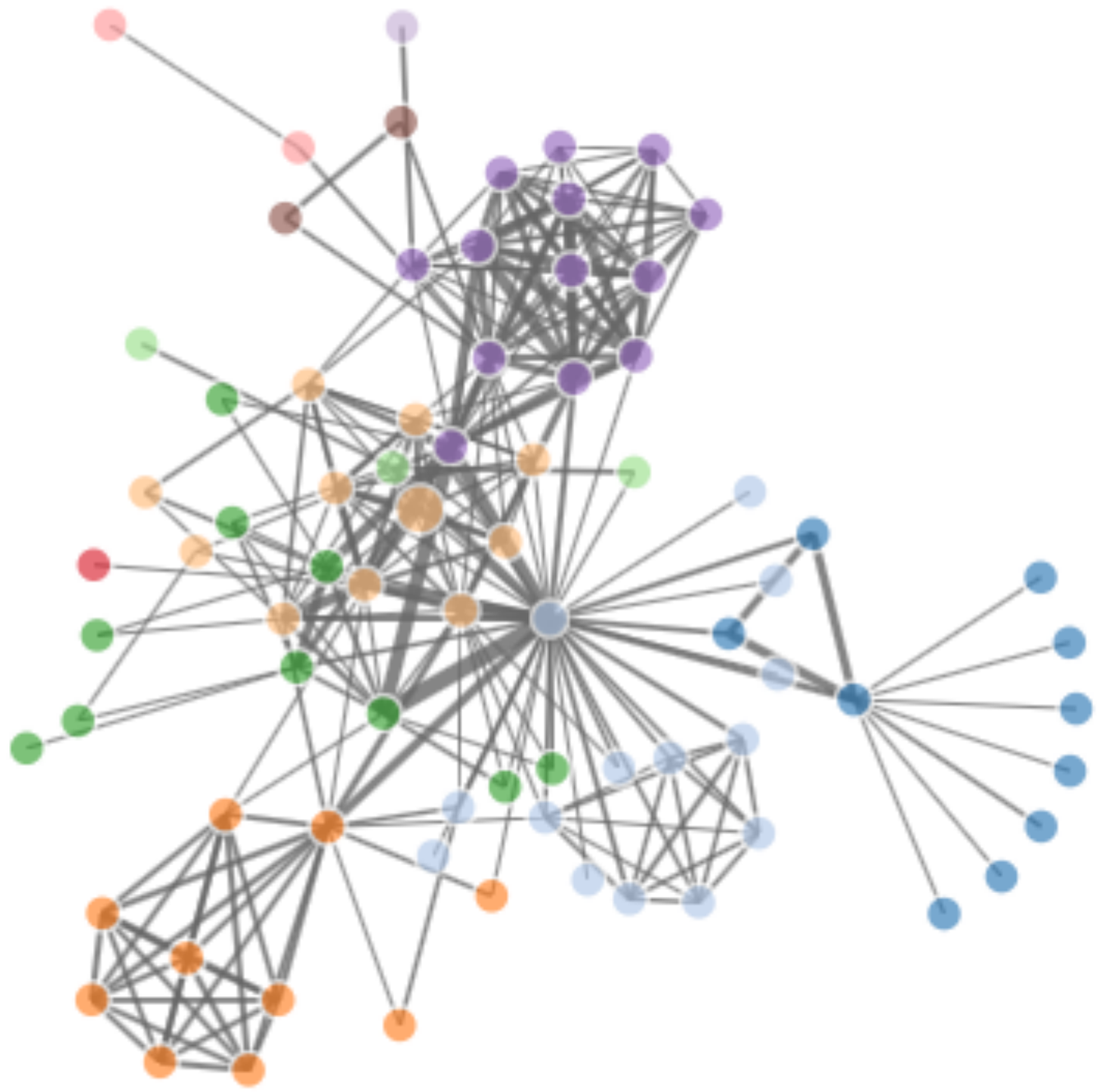

```
m %>% addProviderTiles("CartoDB.Positron")
```

http://rstudio.github.io/leaflet

```r
library(dygraphs)
lungDeaths <- cbind(mdeaths, fdeaths)
dygraph(lungDeaths)
```



http://rstudio.github.io/dygraphs/



http://christophergandrud.github.io/networkD3/

```r
library(DT)
datatable(iris)
```

Show 10 entries

Search:

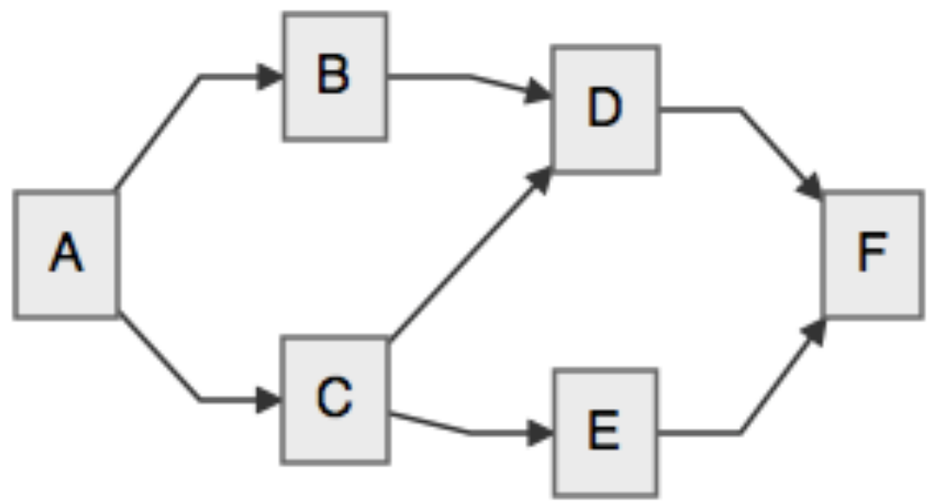
	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa
8	5	3.4	1.5	0.2	setosa
9	4.4	2.9	1.4	0.2	setosa
10	4.9	3.1	1.5	0.1	setosa

Showing 1 to 10 of 150 entries

Previous 1 2 3 4 5 … 15 Next

http://rstudio.github.io/DT/

EXAMPLE



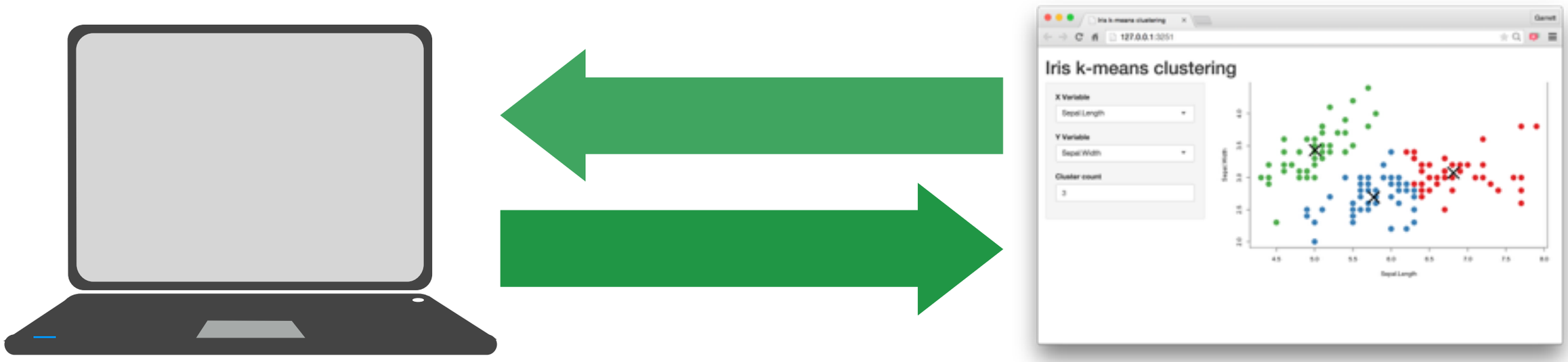
Copy

```
mermaid("
graph LR
  A-->B
  A-->C
  C-->E
  B-->D
  C-->D
  D-->F
  E-->F
")
```

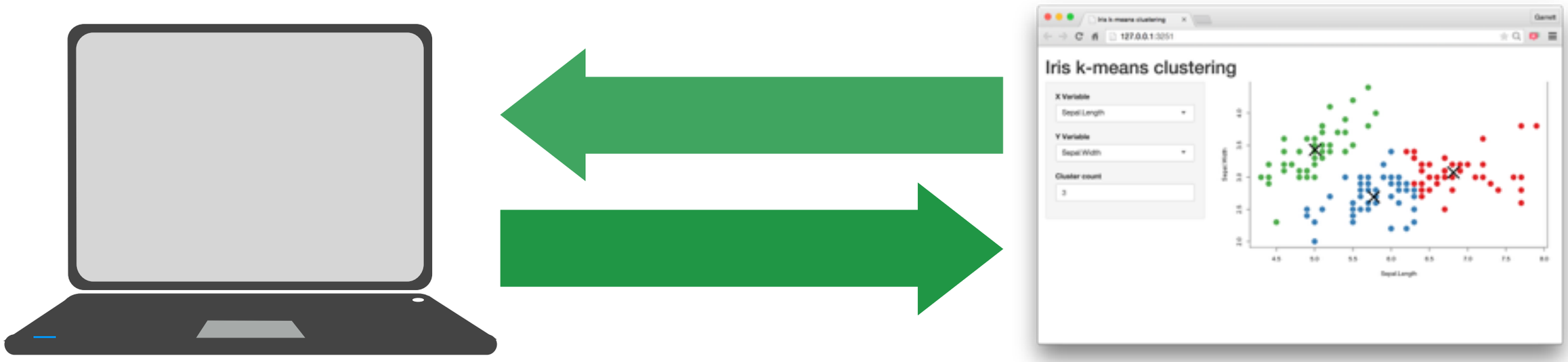
http://rich-iannone.github.io/DiagrammeR

Shiny apps

Every Shiny app needs a computer running R



Every Shiny app needs a computer running R



server

ui

Shinyapps

As your apps get more complex, it's useful to put ui and server code in separate files.

runApp() can also take path to directory containing `ui.R` and `server.R`

```r
library(shiny)

ui <- fluidPage(
  sliderInput("number", "Pick a number", 1, 10, value = 5),
  p("You picked: ", textOutput("result", inline = TRUE))
)

server <- function(input, output) {
  message("Initialising")
  output$result <- renderText({
    message("Updating")
    runif(input$number)
  })
}

runApp(shinyApp(ui, server))
```

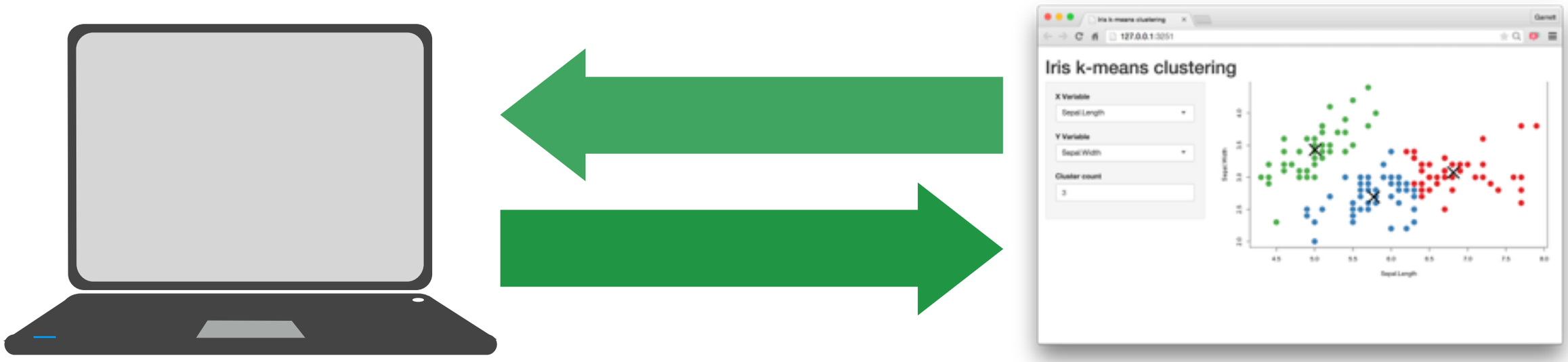
ui.R

```
library(shiny)

fluidPage(
  sliderInput("number",
   "Pick a number", 1, 10,
   value = 5),
  p("You picked: ",
   textOutput(
     "result", inline = TRUE
   )
  )
)
```

server.R

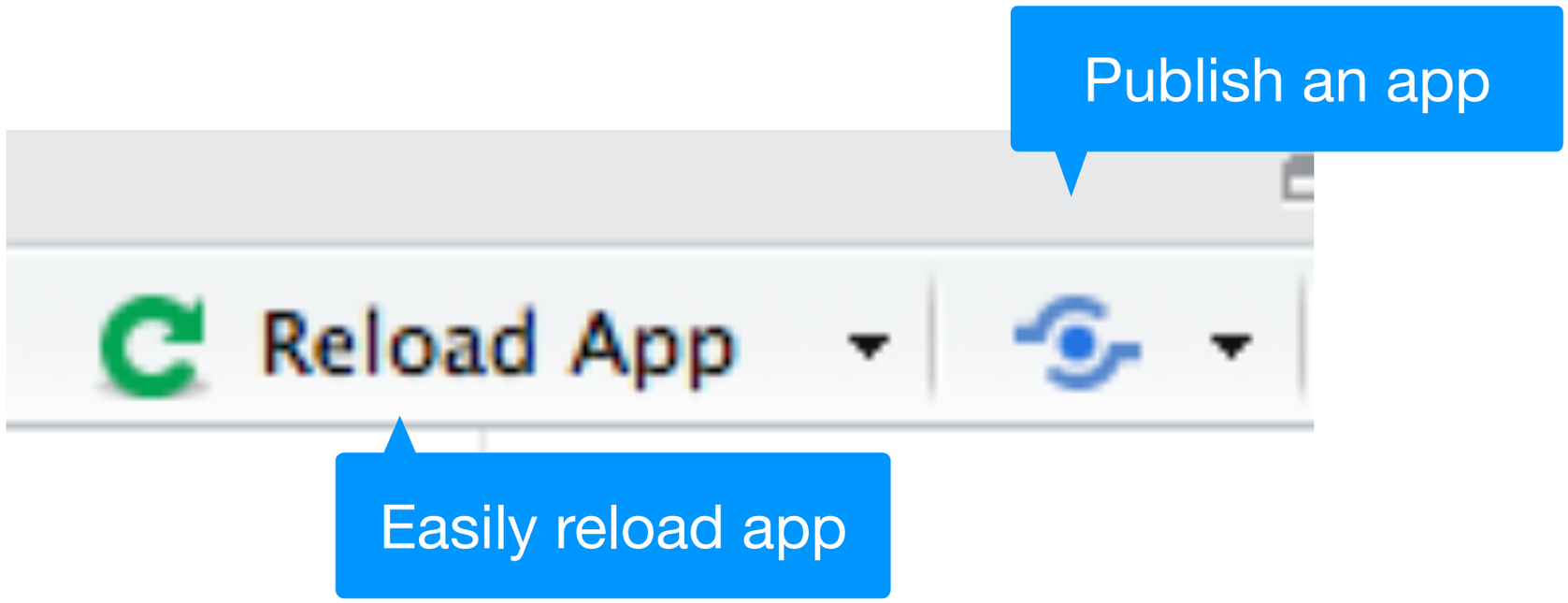
```
library(shiny)

function(input, output) {
 output$result <- renderText({
   runif(input$number)
  })
}
```

Every Shiny app needs a computer running R



server.R

ui.R


Publish an app
Reload App
Easily reload app

To publish you need a server

http://shiny.rstudio.com/deploy/

shinyapps.io	Free (for personal use)	Easy setup. Data in “cloud”
Shiny Server OS	Free	Run on own server
Shiny Server pro	$10,000 / year	“Enterprise”, more control over execution

Your turn

Sign up for shinyapps.io.

Create a two-file app in it’s own directory.

Publish it!