RShiny Workshop

Bianca Brusco & Clare Clingain

What is RShiny?

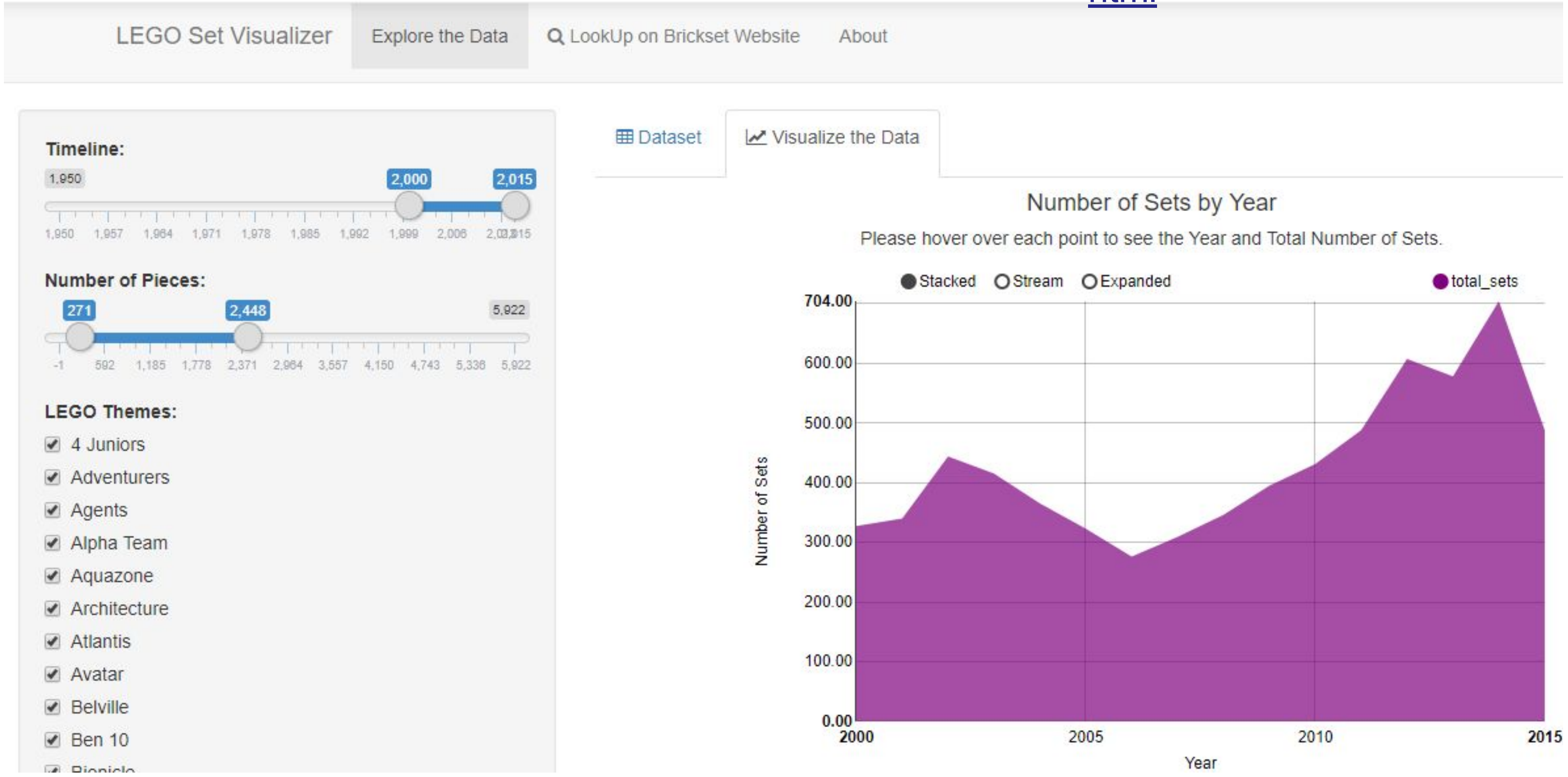
-Build interactive online apps

-Share these apps with the world: Deploy them on the shiny server!

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

What can you do with it?

Link: https://shiny.rstudio.com/gallery/lego-set.html



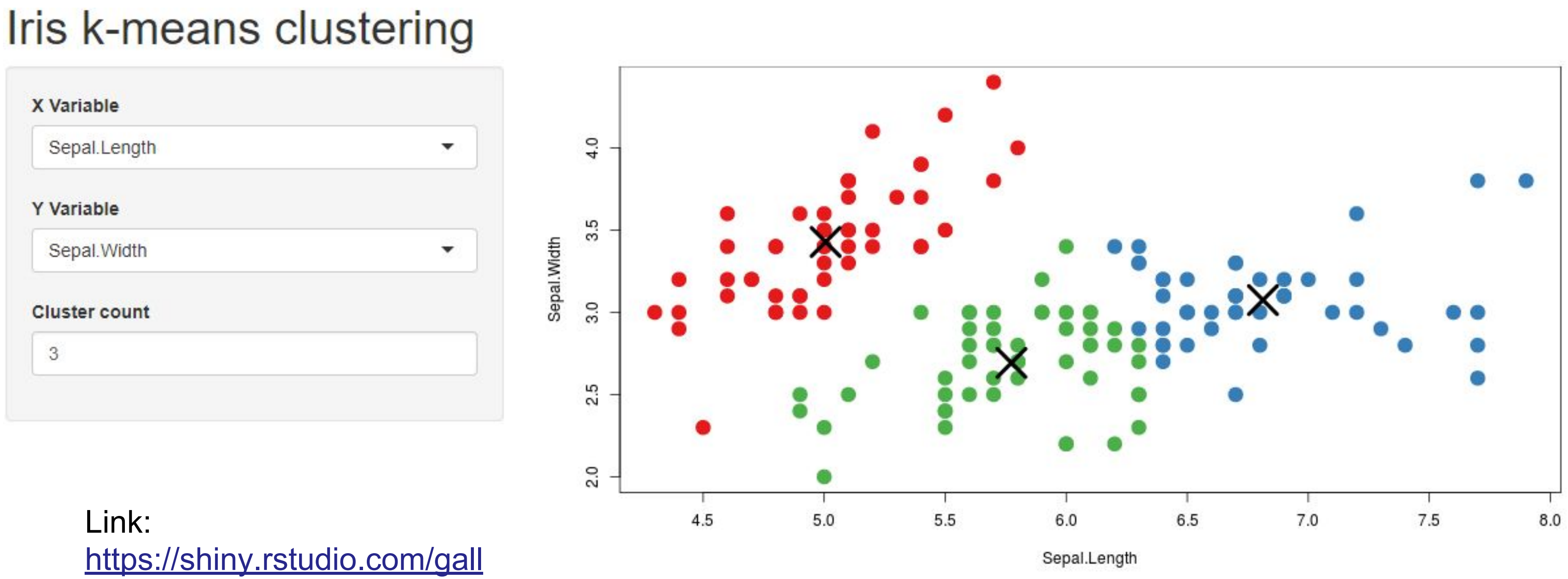
Timeline: 2,000 – 2,015

Number of Pieces: 271 – 2,448

LEGO Themes:
- [✓] 4 Juniors
- [✓] Adventurers
- [✓] Agents
- [✓] Alpha Team
- [✓] Aquazone
- [✓] Architecture
- [✓] Atlantis
- [✓] Avatar
- [✓] Belville
- [✓] Ben 10
- [✓] Bionicle

Options: [● Stacked] ○ Stream ○ Expanded

What can you do with it?



Iris k-means clustering

X Variable

Sepal.Length

Y Variable

Sepal.Width

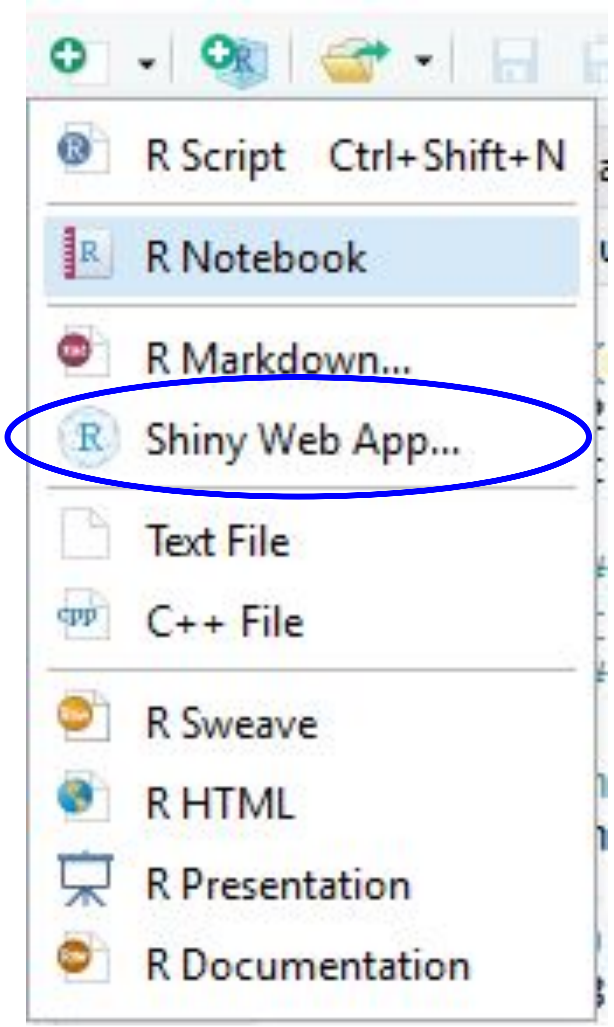
Cluster count

3

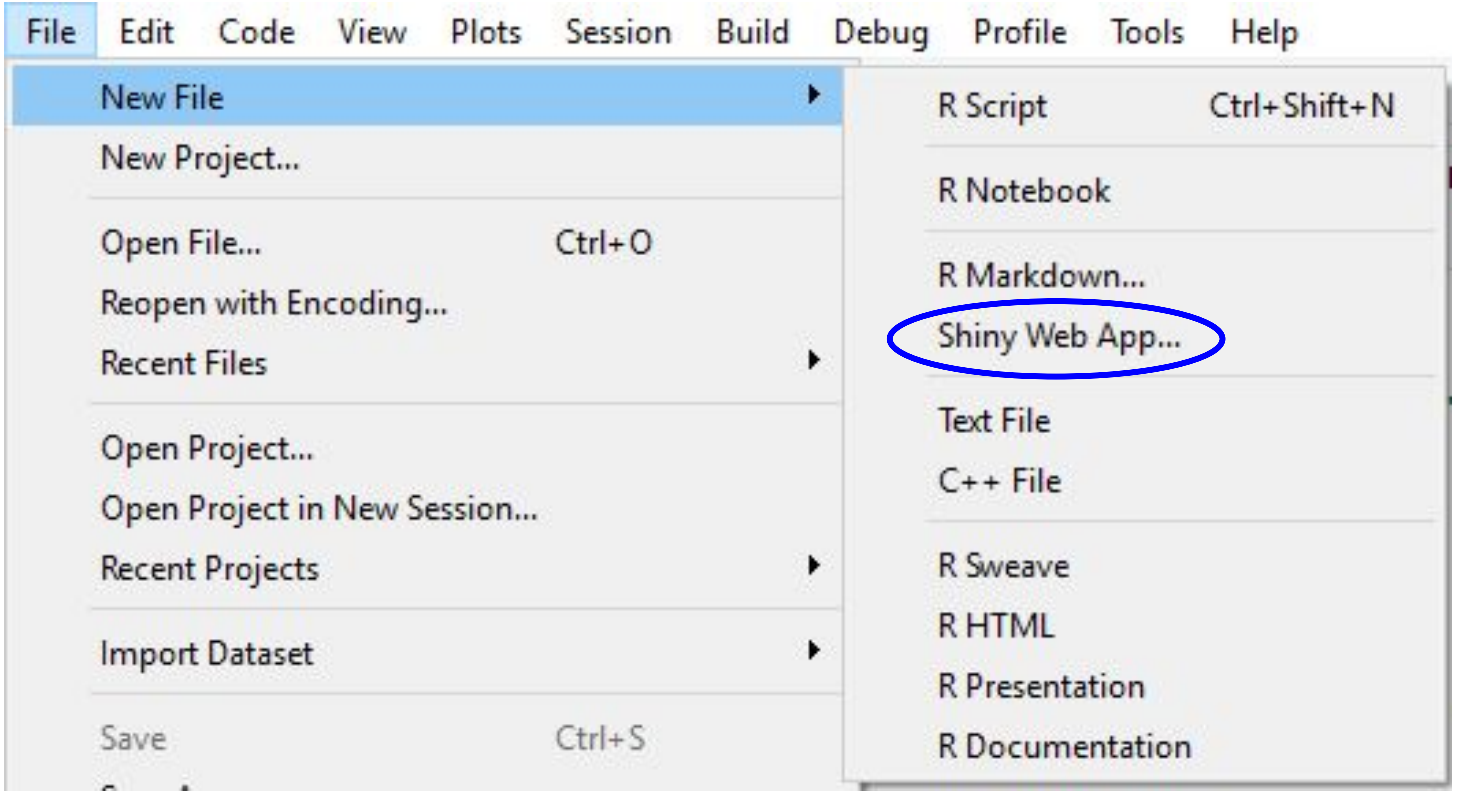
Link: https://shiny.rstudio.com/gallery/kmeans-example.html

Where do I start?

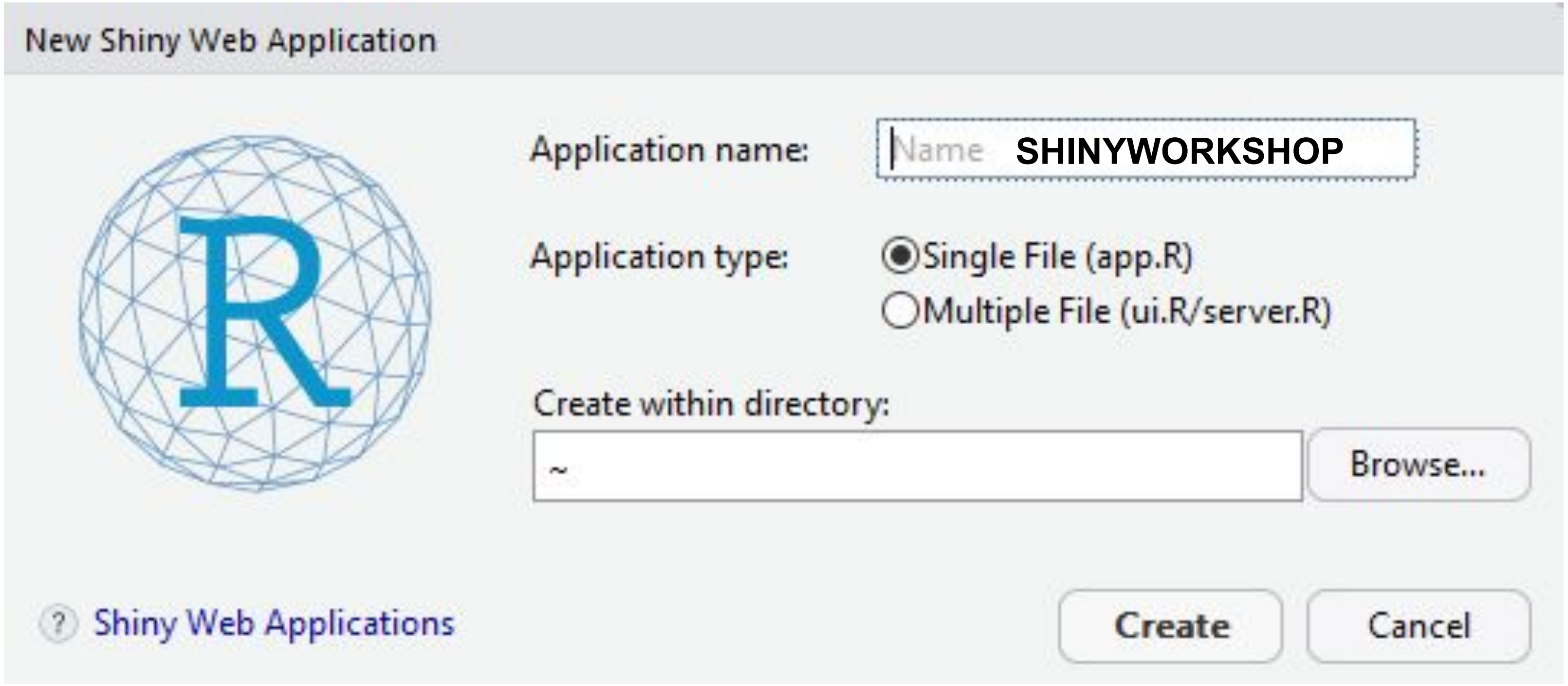
- **EASIEST WAY IS THROUGH R STUDIO!!!**
- Access through file menu or new file shortcut



Where do I start?



Where do I start?


New Shiny Web Application

Application name: Name SHINYWORKSHOP

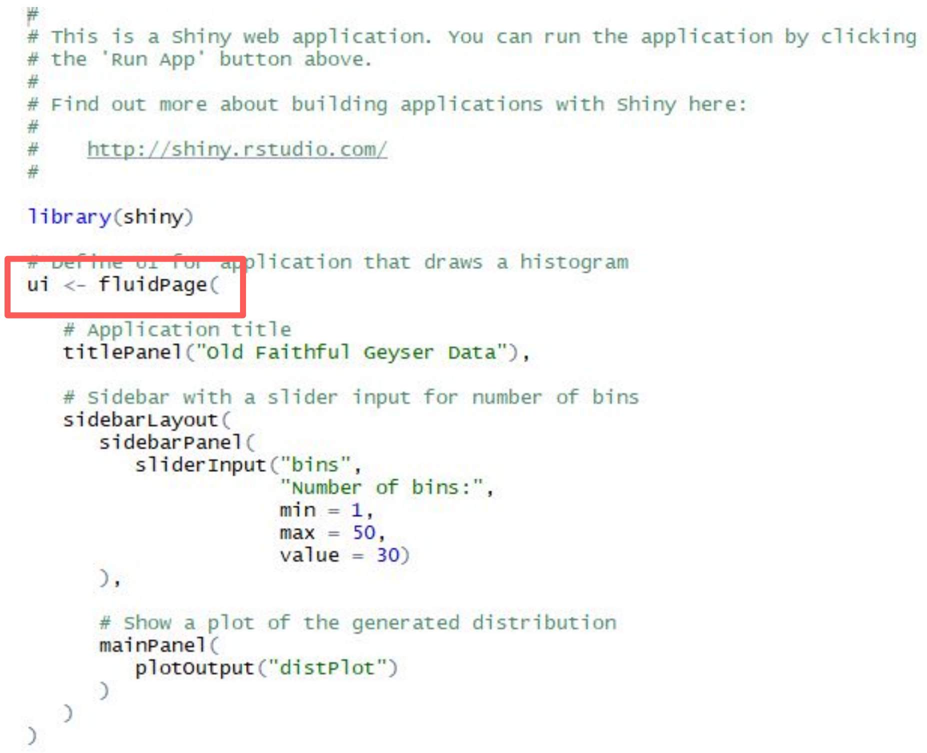
Application type: [✓ Single File (app.R)] Multiple File (ui.R/server.R)

Create within directory: ~

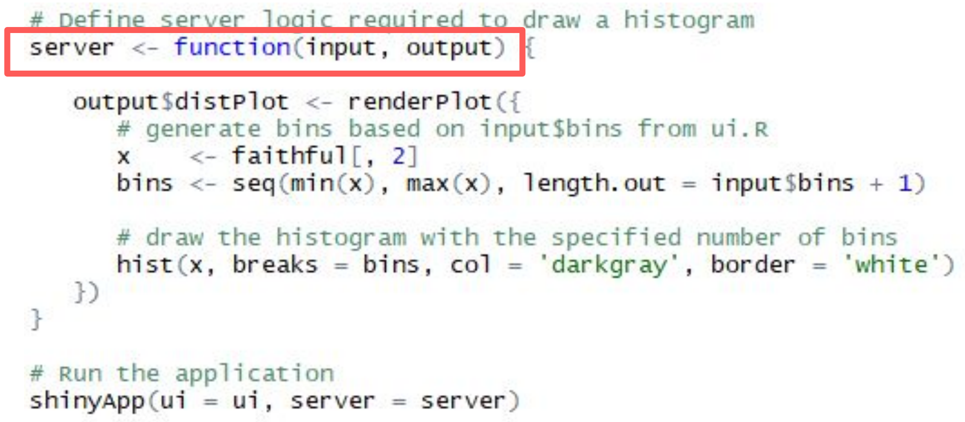
Shiny Library

- Make sure you have Shiny loaded
- If not: install.packages(“Shiny”)
- library(Shiny)

Two Parts: UI and Server



```r
#
# This is a Shiny web application. You can run the application by clicking
# the 'Run App' button above.
#
# Find out more about building applications with Shiny here:
#
#    http://shiny.rstudio.com/
#

library(shiny)

# Define UI for application that draws a histogram
ui <- fluidPage(

   # Application title
   titlePanel("Old Faithful Geyser Data"),

   # Sidebar with a slider input for number of bins
   sidebarLayout(
      sidebarPanel(
         sliderInput("bins",
                     "Number of bins:",
                     min = 1,
                     max = 50,
                     value = 30)
      ),

      # Show a plot of the generated distribution
      mainPanel(
         plotOutput("distPlot")
      )
   )
)
```



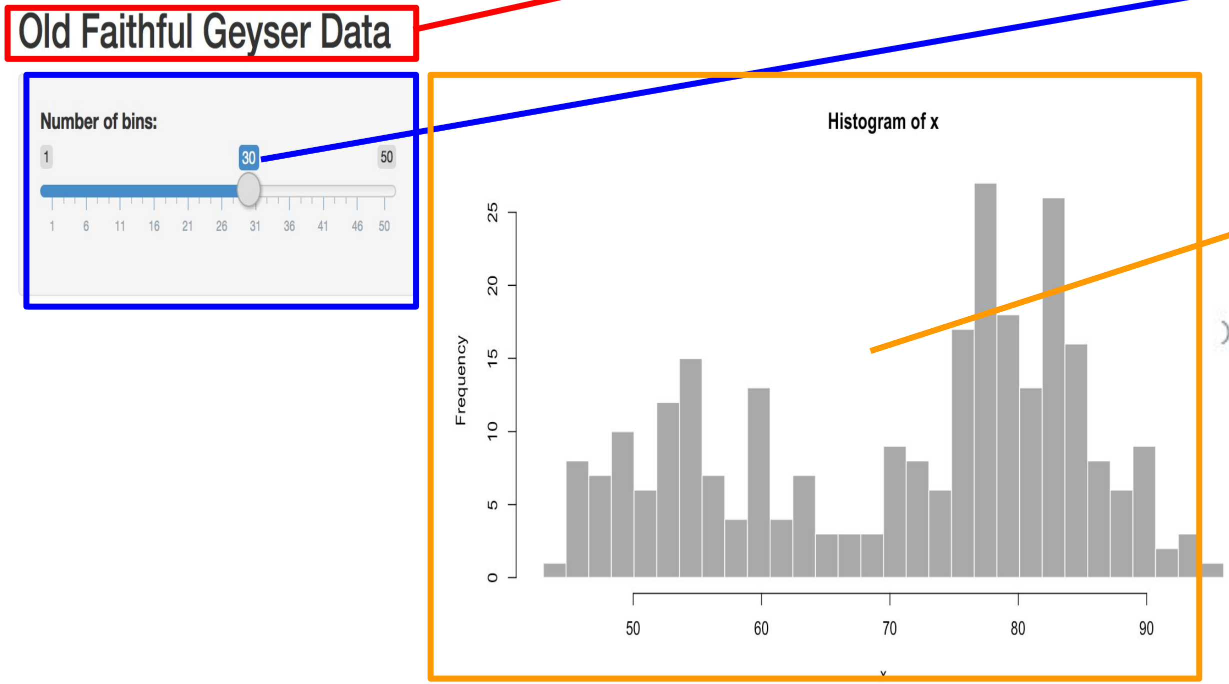
```r
# Define server logic required to draw a histogram
server <- function(input, output) {

   output$distPlot <- renderPlot({
      # generate bins based on input$bins from ui.R
      x    <- faithful[, 2]
      bins <- seq(min(x), max(x), length.out = input$bins + 1)

      # draw the histogram with the specified number of bins
      hist(x, breaks = bins, col = 'darkgray', border = 'white')
   })
}

# Run the application
shinyApp(ui = ui, server = server)
```

The Gist

UI: What people will see

Server: Where the work happens to make the output

Basics: UI



```
library(shiny)

# Define UI for application that draws a histogram
ui <- fluidPage(

  # Application title
  titlePanel("Old Faithful Geyser Data"),

  # Sidebar with a slider input for number of bins
  sidebarLayout(
    sidebarPanel(
      sliderInput("bins",
                  "Number of bins:",
                  min = 1,
                  max = 50,
                  value = 30)
    ),

    # Show a plot of the generated distribution
    mainPanel(
      plotOutput("distPlot")
    )
  )
)
```

Types of Input Functions

Functions

actionbutton

checkboxGroupInput

checkboxInput

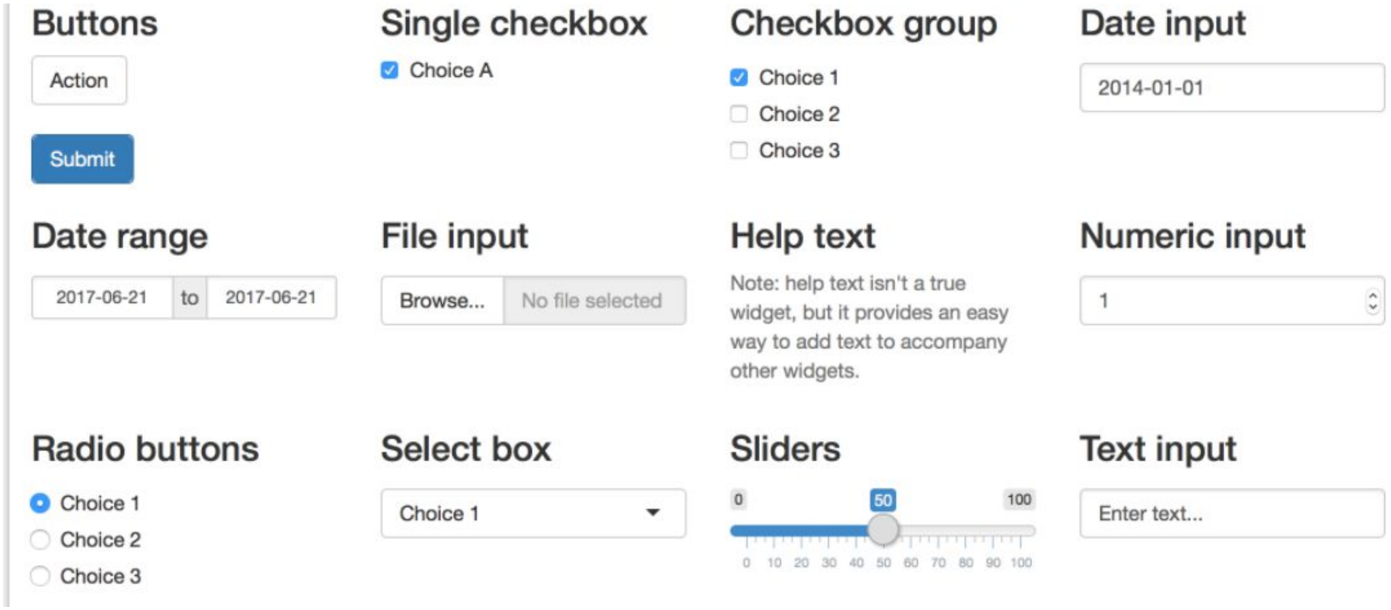
dateInput

fileInput

helpText

numericInput

radioButtons

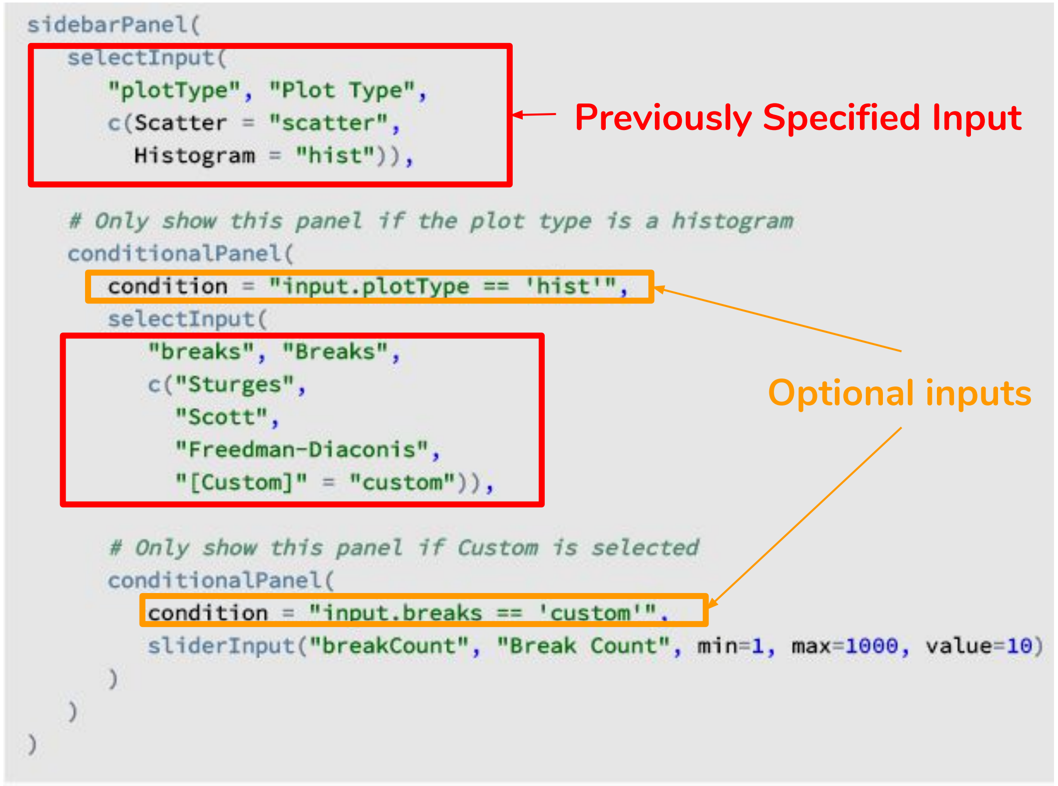


Note: help text isn't a true widget, but it provides an easy way to add text to accompany other widgets.

selectInput sliiderInput submitButton textInput

Conditional Panels

- Allow you to have inputs display given a previously specified input choice.
- **conditionalPanel()**
- Refer to previous input as “input.NAME_OF_INPUT”
- Separate consecutive conditional panels by a comma
- **Make sure all are housed under sidebarPanel()!**

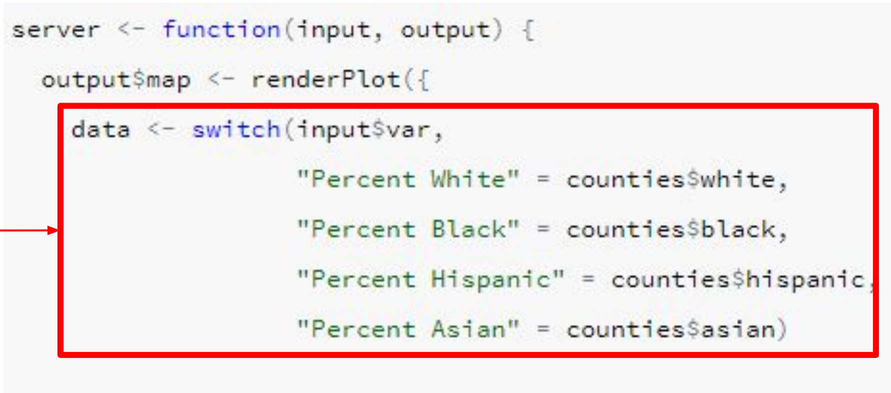


```
sidebarPanel(
  selectInput(
     "plotType", "Plot Type",
     c(Scatter = "scatter",
       Histogram = "hist")),

  # Only show this panel if the plot type is a histogram
  conditionalPanel(
     condition = "input.plotType == 'hist'",
     selectInput(
        "breaks", "Breaks",
        c("Sturges",
          "Scott",
          "Freedman-Diaconis",
          "[Custom]" = "custom")),

     # Only show this panel if Custom is selected
     conditionalPanel(
        condition = "input.breaks == 'custom'",
        sliderInput("breakCount", "Break Count", min=1, max=1000, value=10)
     )
  )
)
```

Source & Reactive Options

- You can source in data or R scripts as long as they are housed in the same folder/directory as your Shiny app
- You can add reactive options that change arguments based on what the user has selected
    - Good way to re-run only certain calculations when re-loading the app takes a long time!

Reactive function to change the data without changing it in the calculation itself (avoid multiple conditional if/else in server)



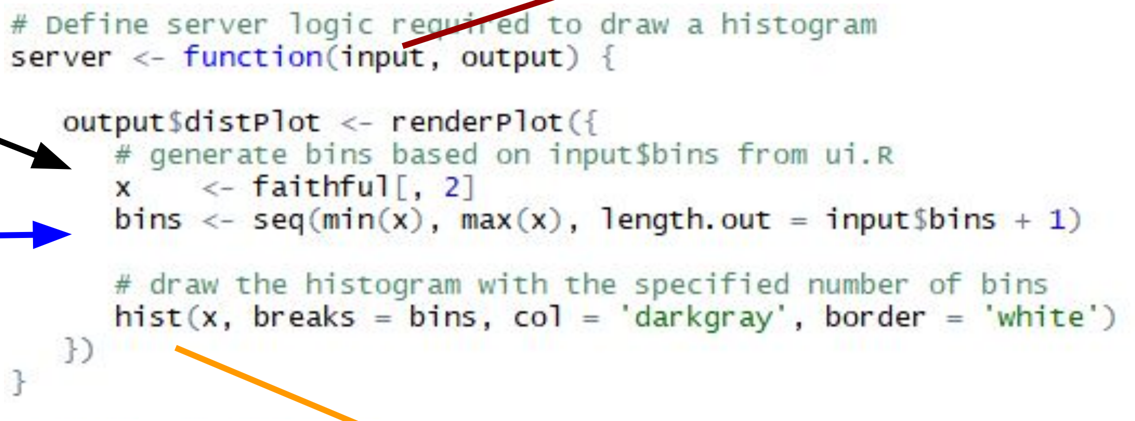
```
server <- function(input, output) {
  output$map <- renderPlot({
    data <- switch(input$var,
                   "Percent White" = counties$white,
                   "Percent Black" = counties$black,
                   "Percent Hispanic" = counties$hispanic,
                   "Percent Asian" = counties$asian)
```

Basics: Server

Back to the Old Faithful Example...

TAKES INPUT FROM WHAT YOU SELECT IN UI PART

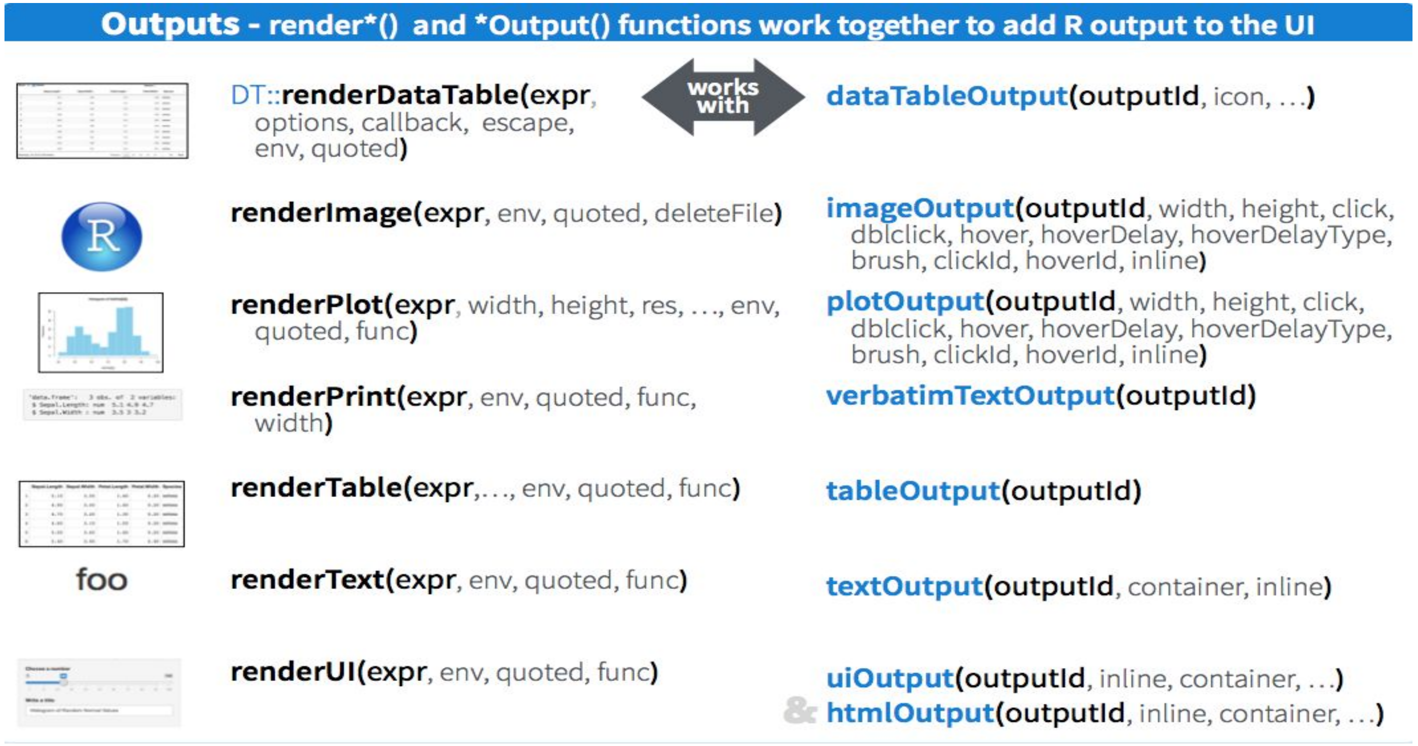
LOAD DATA



```
# Define server logic required to draw a histogram
server <- function(input, output) {

  output$distPlot <- renderPlot({
    # generate bins based on input$bins from ui.R
    x    <- faithful[, 2]
    bins <- seq(min(x), max(x), length.out = input$bins + 1)

    # draw the histogram with the specified number of bins
    hist(x, breaks = bins, col = 'darkgray', border = 'white')
  })
}
```

Creates an “output” object that you call from the UI file - called “distPlot”.
If you need to add extra plots, you can create a new object e.g. output$newplot

What about other outputs?



Outputs - render*() and *Output() functions work together to add R output to the UI

works with

DT::**renderDataTable(expr**, options, callback, escape, env, quoted**)**	**dataTableOutput(outputId**, icon, ...**)**
renderImage(expr, env, quoted, deleteFile**)**	**imageOutput(outputId**, width, height, click, dblclick, hover, hoverDelay, hoverDelayType, brush, clickId, hoverId, inline**)**
renderPlot(expr, width, height, res, ..., env, quoted, func**)**	**plotOutput(outputId**, width, height, click, dblclick, hover, hoverDelay, hoverDelayType, brush, clickId, hoverId, inline**)**
renderPrint(expr, env, quoted, func, width**)**	**verbatimTextOutput(outputId)**
renderTable(expr,..., env, quoted, func**)**	**tableOutput(outputId)**
renderText(expr, env, quoted, func**)**	**textOutput(outputId**, container, inline**)**
renderUI(expr, env, quoted, func**)**	**uiOutput(outputId**, inline, container, ...**)** & **htmlOutput(outputId**, inline, container, ...**)**

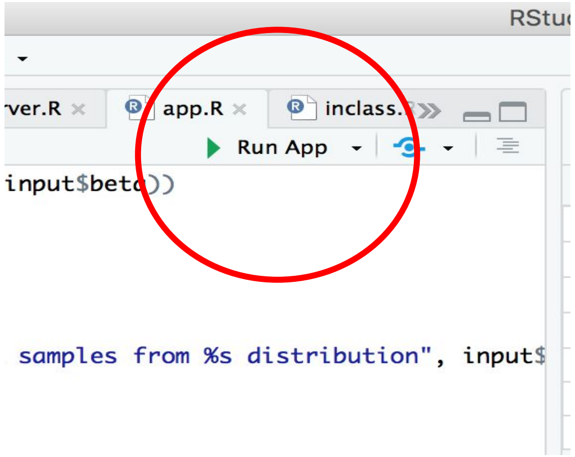
```
'data.frame':   3 obs. of  2 variables:
 $ Sepal.Length: num  5.1 4.9 4.7
 $ Sepal.Width : num  3.5 3 3.2
```

foo

Putting it together:

```
# Run the application
shinyApp(ui = ui, server = server)
```

OR BY CLICKING A BUTTON….



```
input$beta))

 samples from %s distribution", input$
```

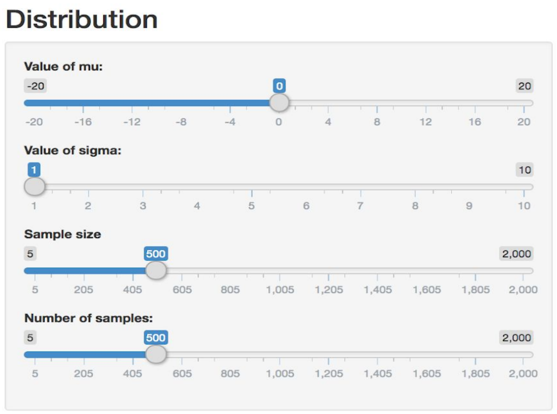
Building our own App!

- Visualizing the Central Limit Theorem:
  - Start with a Normal Distribution
  - Create Sliders for MEAN and SD and each SIZE:
  - Create one plot for Population (using rnorm(sample.size, mu, sd)
  - Create slider to specify NUMBER OF SAMPLES to take
  - Plot the means of each sample

At the end you should have....

- Four sliders
- Two plots! Make sure your titles change too!

The final output should look something like…

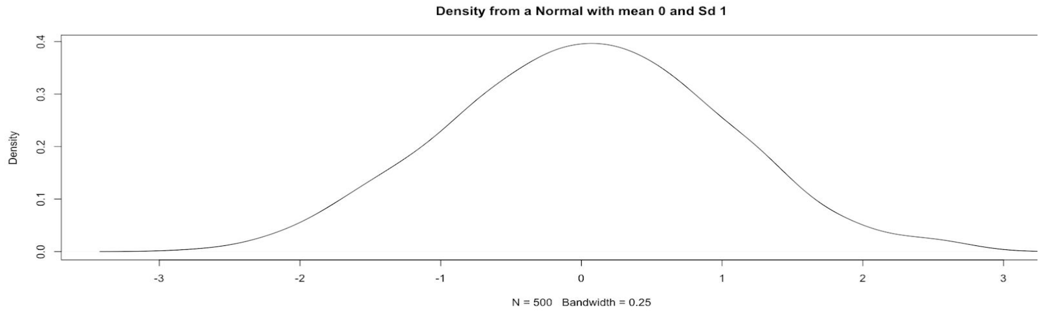

Distribution

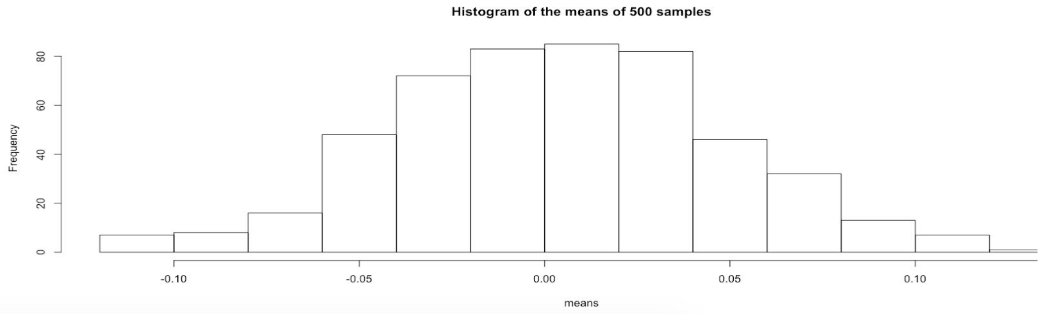
Value of mu: 0 (range -20 to 20)

Value of sigma: 1 (range 1 to 10)

Sample size: 500 (range 5 to 2,000)

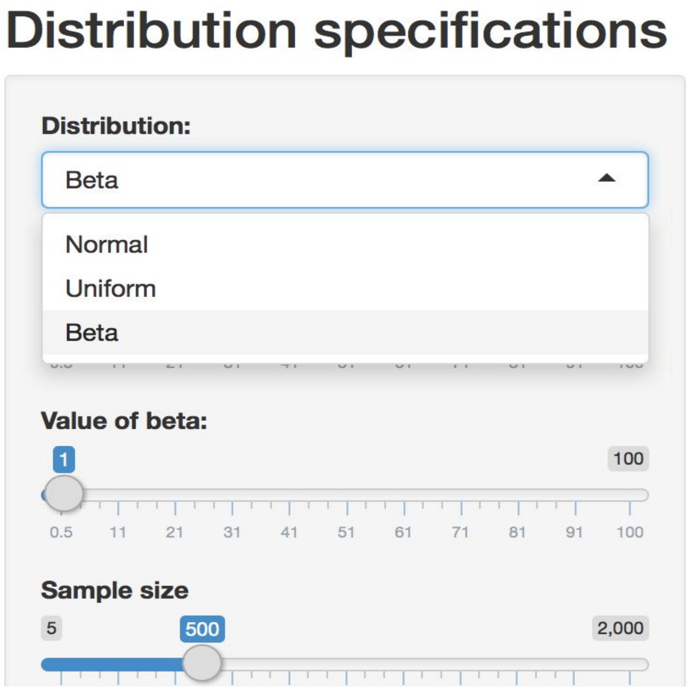
Number of samples: 500 (range 5 to 2,000)





+ **BEAUTIFYING!**

Next: testing different distribution



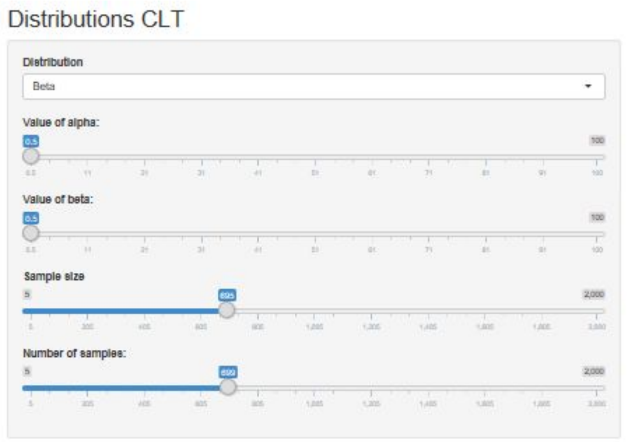
Now try adding a menu drop down to test also

Uniform Distribution, and a Beta Distribution.!

HINT: use “Select Input” and “Conditional Panels”.

Conditional Panels help you sliders or plots that you want to appear only when specific conditions are met

Next: testing different distribution


Distributions CLT

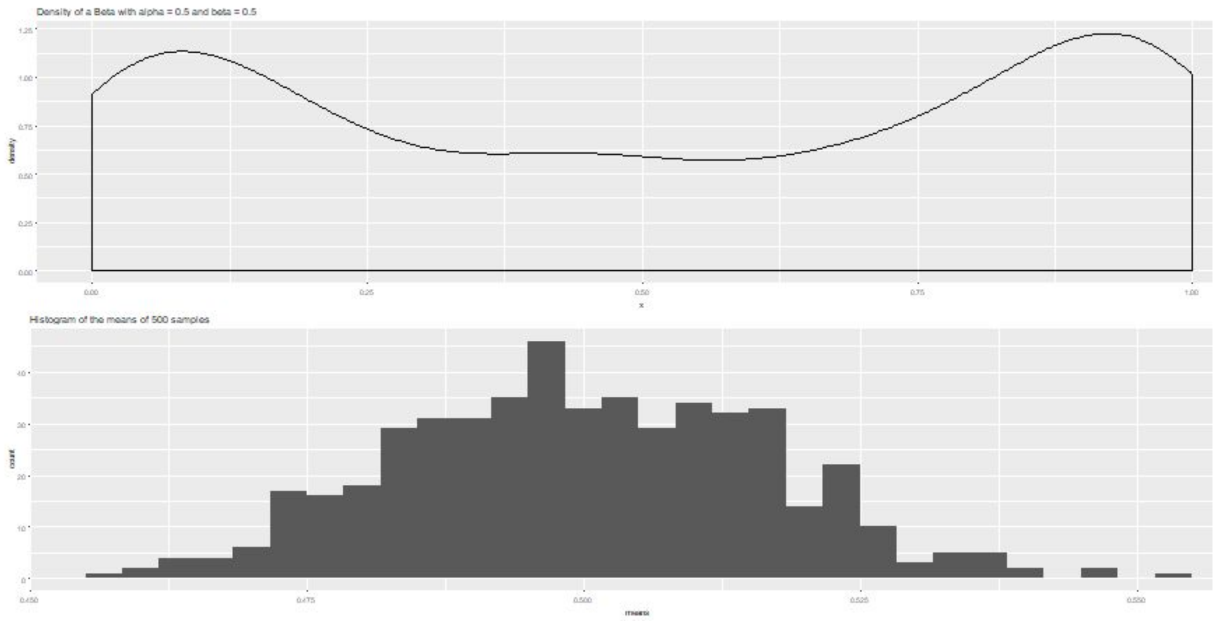
Distribution: Beta

Value of alpha: 0.5 (0.5 – 100)

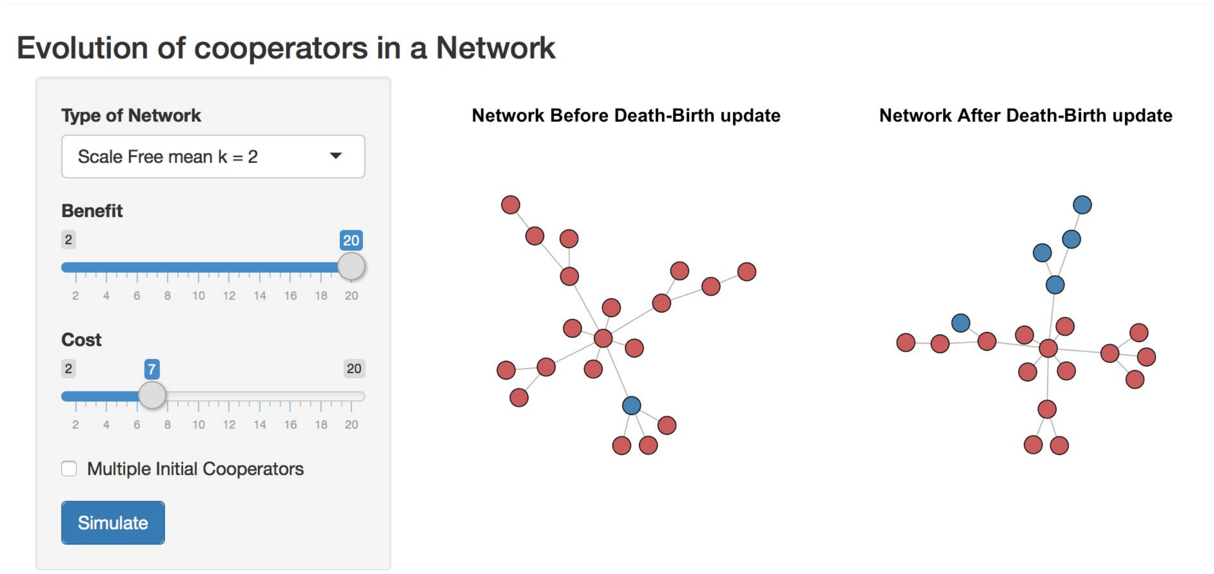
Value of beta: 0.5 (0.5 – 100)

Sample size: 695 (5 – 2,000)

Number of samples: 699 (5 – 2,000)



Example: Bianca



Evolution of cooperators in a Network

Type of Network: Scale Free mean k = 2

Benefit: 20 (range 2–20)

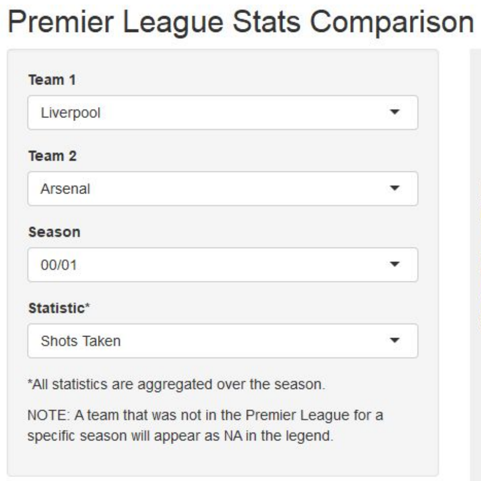
Cost: 7 (range 2–20)

Multiple Initial Cooperators [ ]

Simulate
Network Before Death-Birth update
Network After Death-Birth update

Shiny App useful in this case to visualize a process under different conditions

Example: Clare



Premier League Stats Comparison

Team 1

Liverpool

Team 2

Arsenal

Season

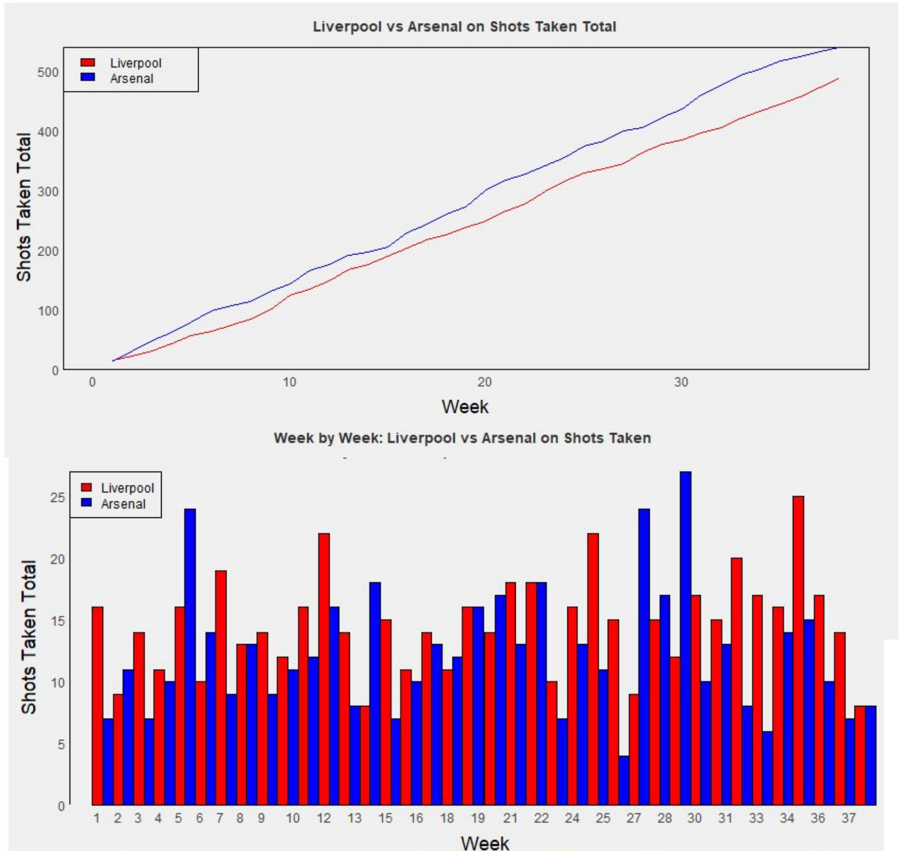
00/01

Statistic*

Shots Taken

*All statistics are aggregated over the season.

NOTE: A team that was not in the Premier League for a specific season will appear as NA in the legend.

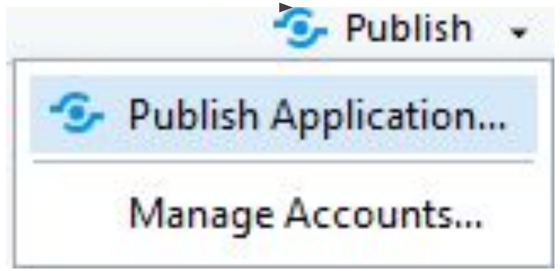


Easily compare any two premier league teams on multiple statistics from 2000 to 2018.

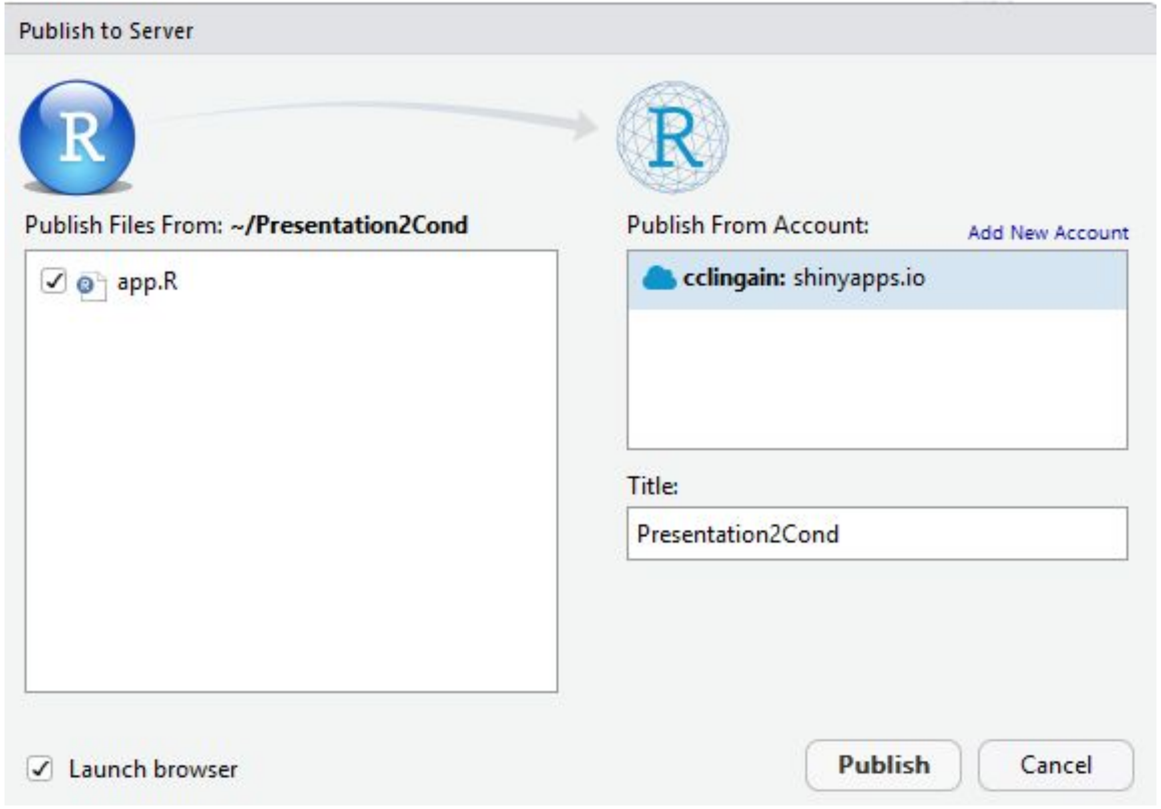
Deployment

- Can create an account with RShiny and "deploy" your apps to the web!

Step 1



Step 2



Publish Files From: ~/Presentation2Cond

✓ app.R

Publish From Account: Add New Account

cclingain: shinyapps.io

Title: Presentation2Cond

✓ Launch browser

THANK YOU!