



COURSE OVERVIEW

& INTRODUCTION TO SHINY

Shiny from



OUTLINE

- Course Overview
 - Instructor
 - Course Schedule
- R Projects
- Shiny High level view
- Anatomy of a Shiny app
 - User interface
 - Server function
 - Running the app
- File Structure

Course

Overview

HELLO

my name is

**GEOFFREY
ARNOLD**

Geoffrey.Lloyd.Arnold@gmail.com

ABOUT ME

- ▶ Chief Data & Analytics Officer
 - ▶ Allegheny County
- ▶ MSPPM 2015
 - ▶ Heinz College

COURSE SCHEDULE

Class 1 - 7/8 - Course Overview & Introduction to GitHub & Shiny

Class 2 - 7/15 - Reactive Programming & User Interfaces

Class 3 - 7/22 - Reactive Programming Pt. 2 & Dashboards

Class 4 - 7/29 - Interactive Visualizations & Advanced Reactivity

Class 5 - 8/5 – Modules & Bookmarking

Class 6 - 8/12 - Connecting to Databases & API's

Class 7* - 8/19 - Leaflet & LeafletProxy

Projects in RStudio

“R Projects are great.”

—Geoffrey Arnold

R PROJECTS

- ▶ So what is all this?
 - ▶ Avoid messy environment
 - ▶ Keep custom functions in check
 - ▶ Don't lose your work just because you want to do something else
- ▶ Info: <https://support.rstudio.com/hc/en-us/articles/200526207-Using-Projects>

HOW DO PROJECTS WORK?

- ▶ R typically saves your environment information in a default location (typically your Documents folder)
- ▶ When you create a project it gets its own .RData file for the project in the Directory/folder you created
- ▶ This is also the default working directory for your project, so no need to put all of the folders its in
 - ▶ Simply load objects by name if they're in the project folder

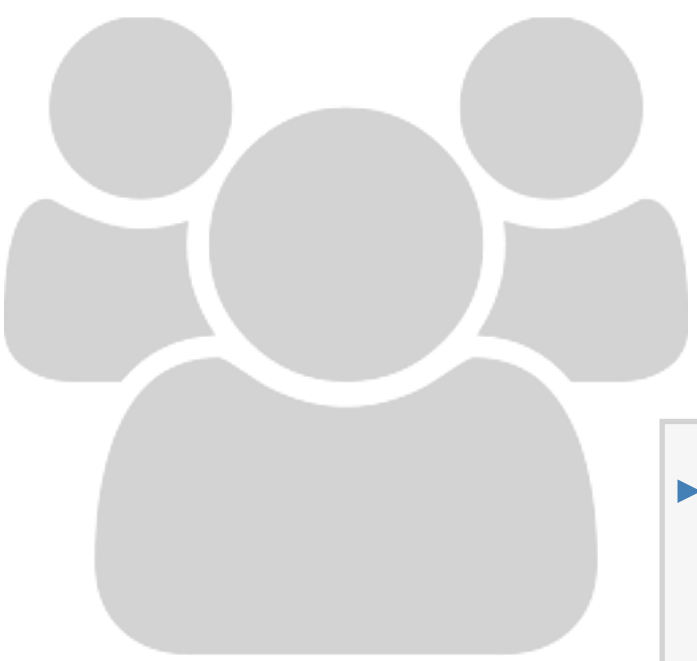
EXERCISE



- ▶ Create a “New Project”
 - ▶ Select “New Directory”
 - ▶ Select “New Project”
 - ▶ Make sure the “Create a git repository” is selected
 - ▶ Give the project any name you want
 - ▶ Click “Create Project”

5_m 00_s

EXERCISE

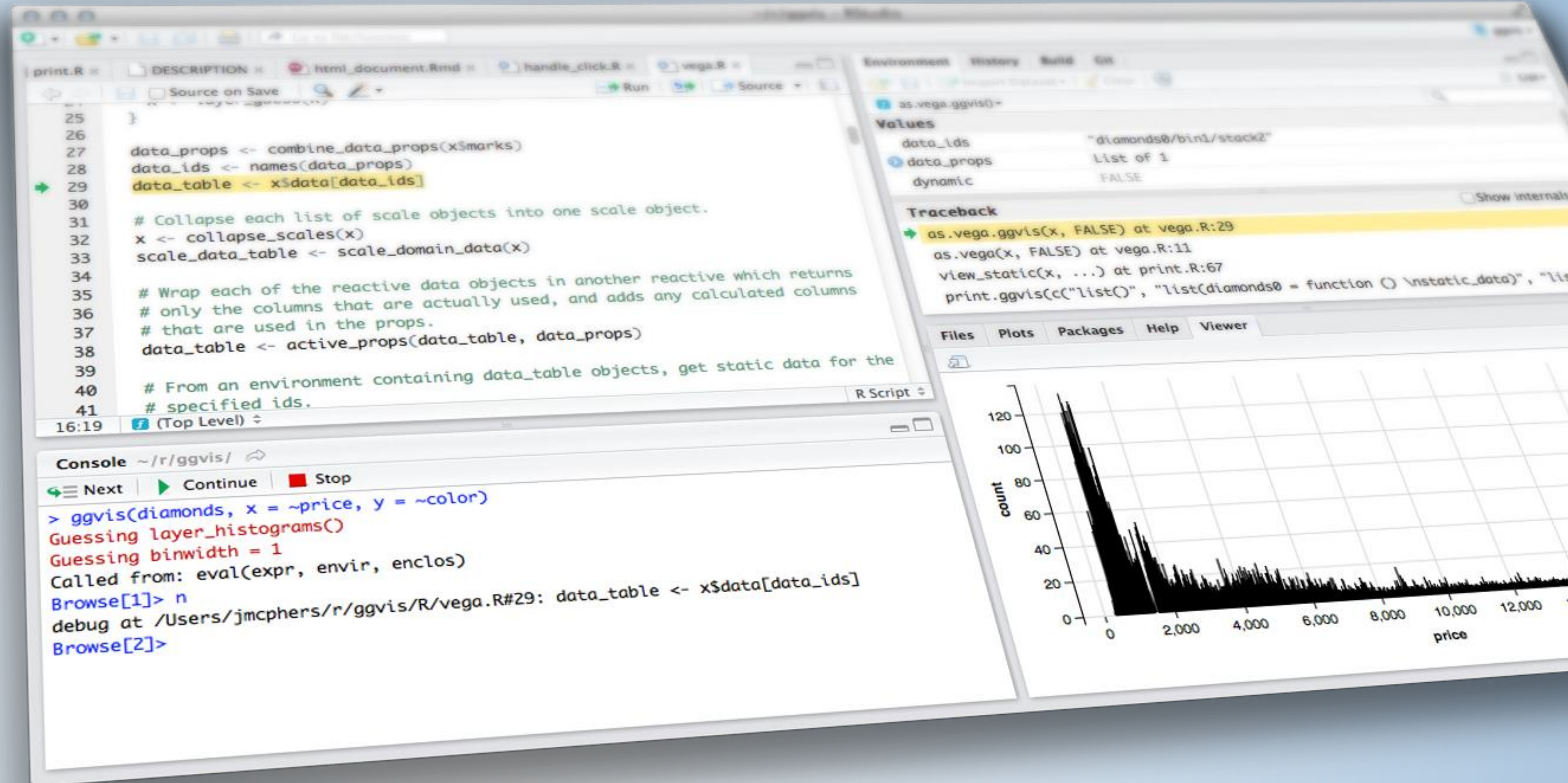


- ▶ Go to “Global Options”
 - ▶ Click “Git/SVN”
 - ▶ Ensure “Enable version control interface for Studio projects” is selected
 - ▶ Click “Create SSH Key...”
 - ▶ Click “Create”
 - ▶ Click “View public key” and copy key
- ▶ Go to <https://github.com/settings/keys>
 - ▶ Click “New SSH key”
 - ▶ Paste key in text box and give your key a name
 - ▶ Click “Add SSH Key”
 - ▶ If you have two factor authorization turned on for GitHub (people with previous GitHub accounts may have this turned on) you will need your Personal Access Token to login later
 - ▶ Everyone else, your GitHub login and password will be important when logging in later.

5_m 00_s

Shiny Examples

- [Port Authority Bus Tracker](#)
- [British Columbia CCISS Tool](#)
- [Commute Explorer](#)
- [Covid-19 Tracker](#)



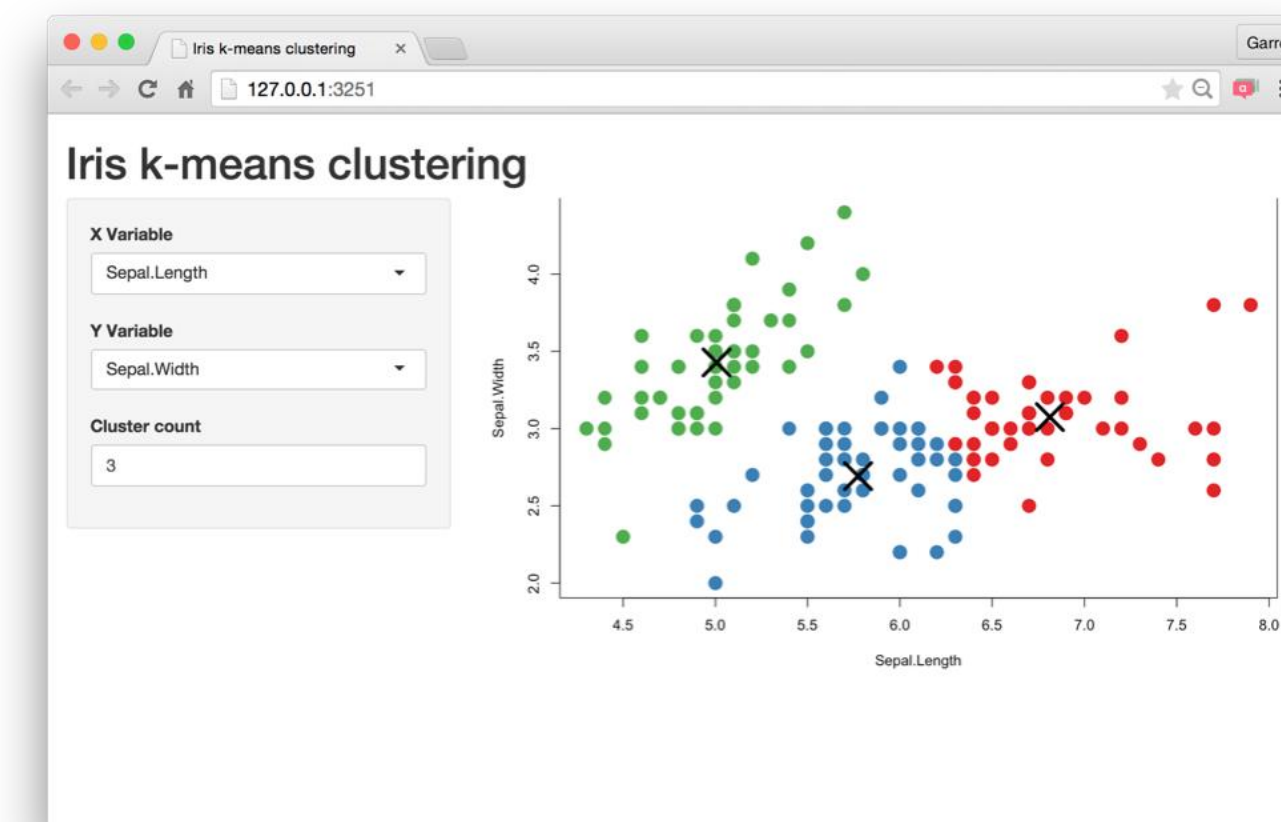
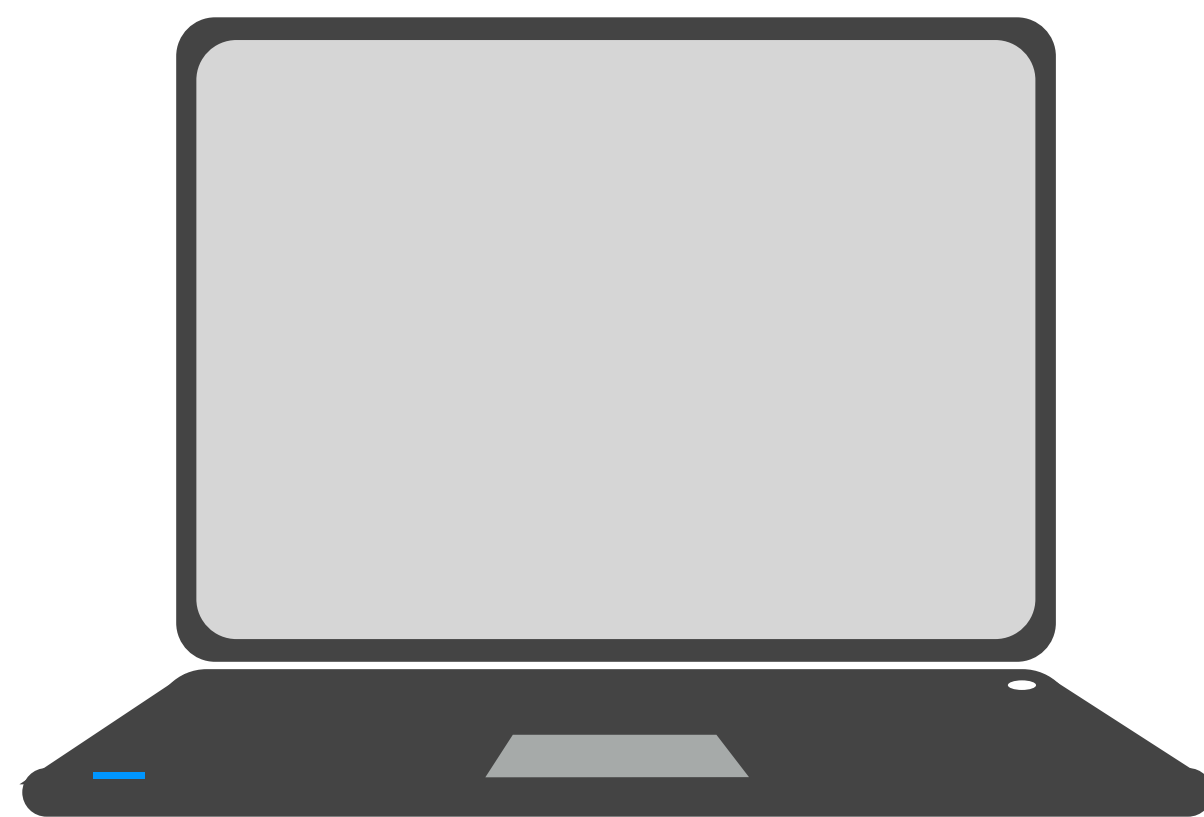
CLASS BREAK

Shiny from

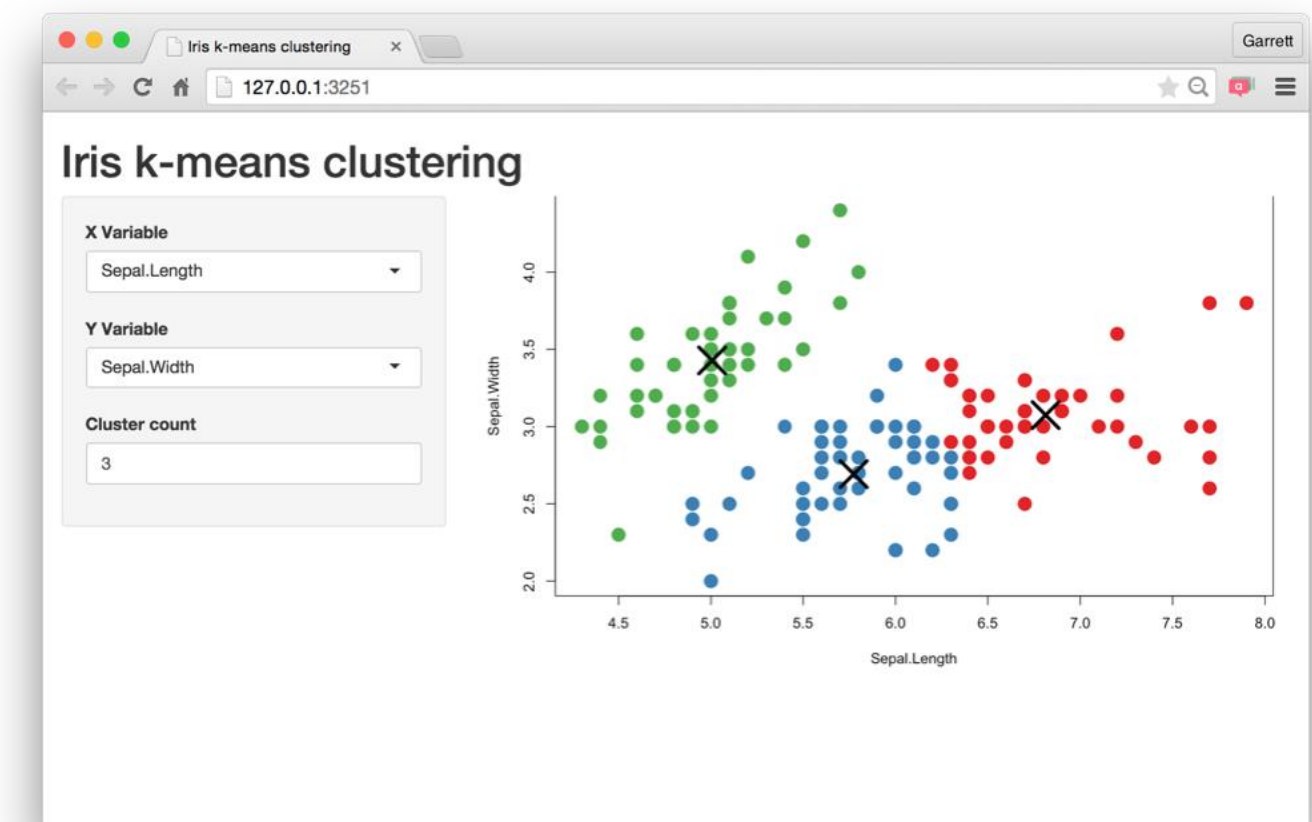
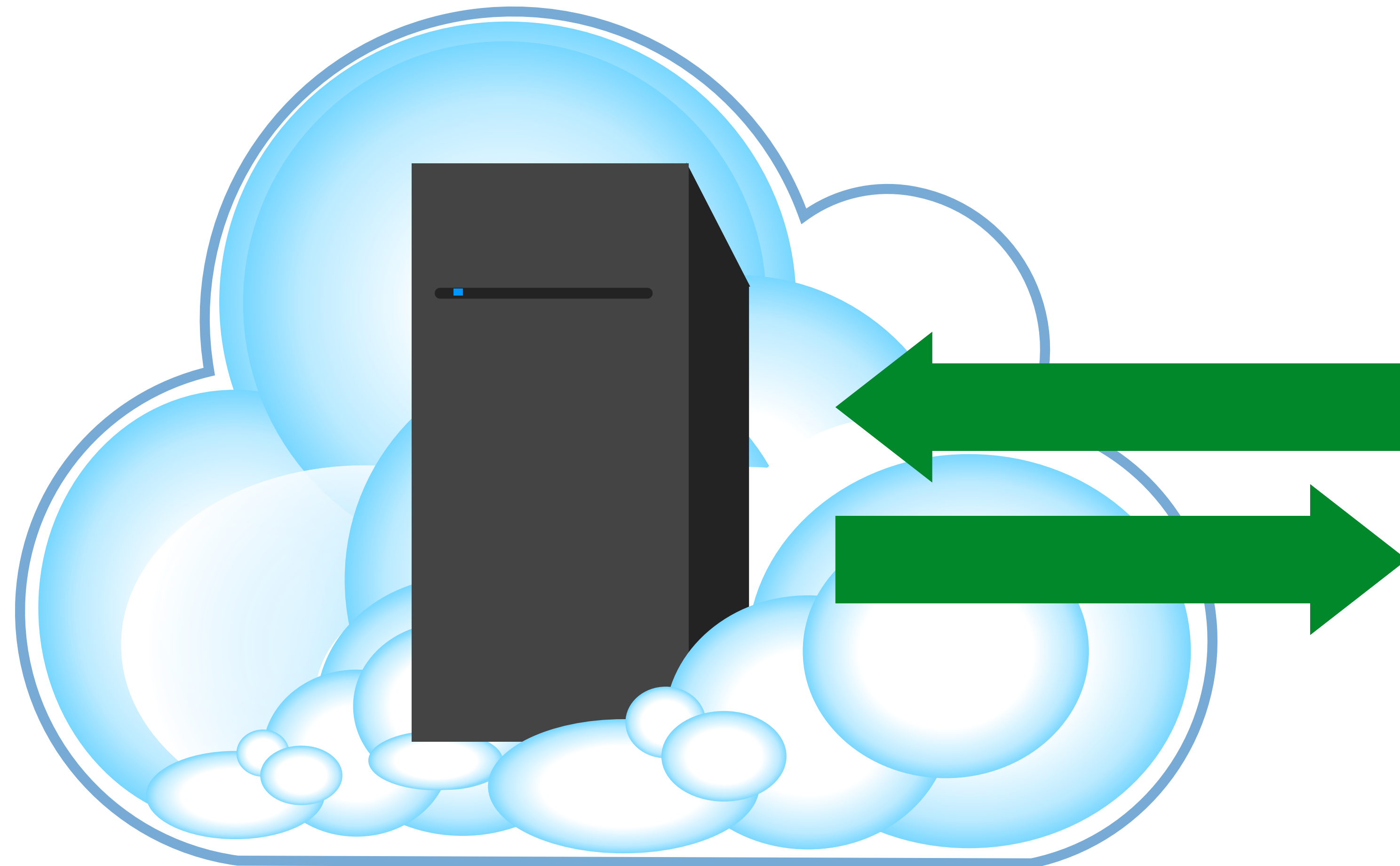


Shiny High level
view

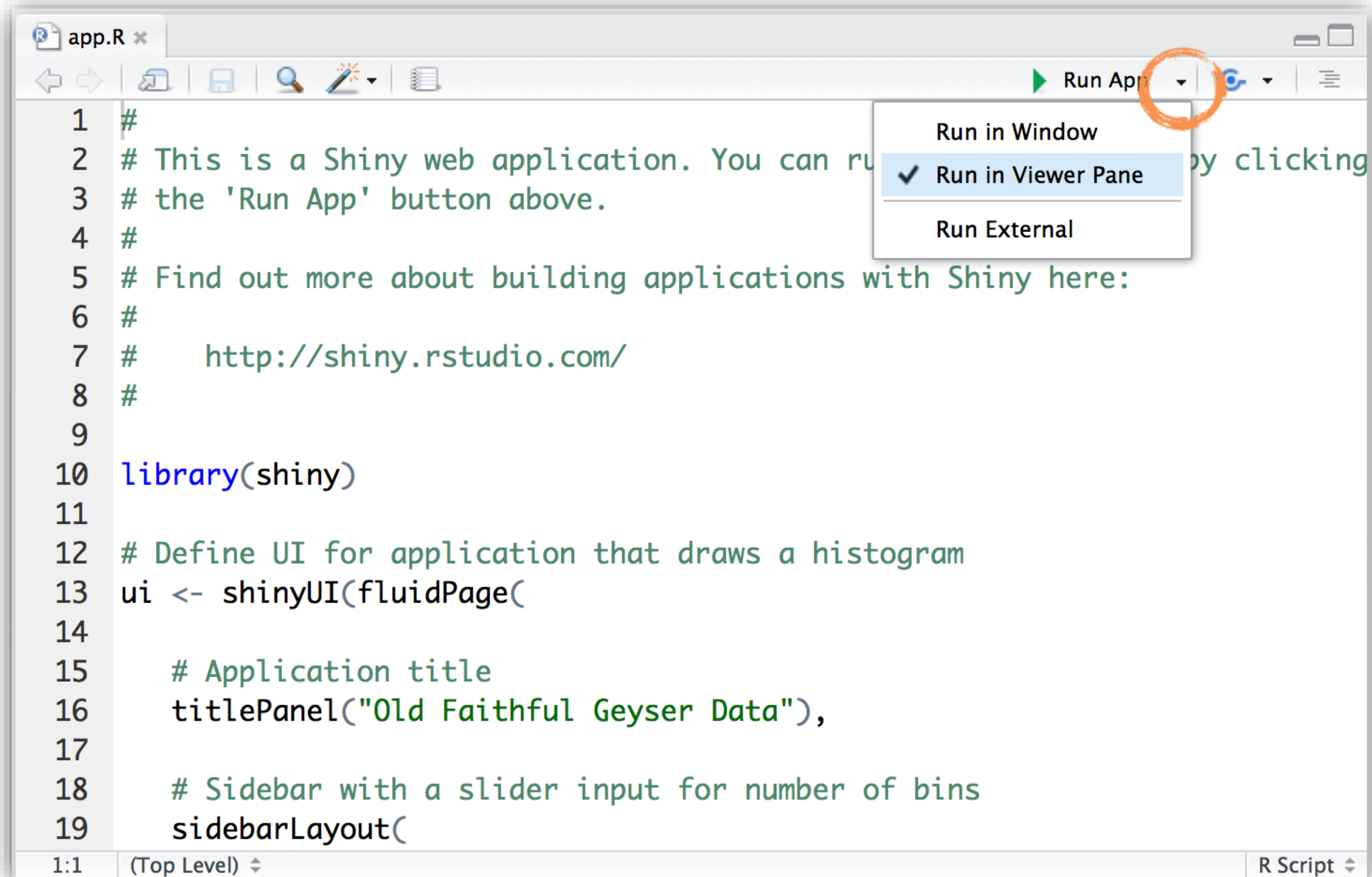
Every Shiny app is maintained by a computer running R



Every Shiny app is maintained by a computer running R



Change display



The screenshot shows the RStudio interface with a file named 'app.R' open. The code in the editor is as follows:

```
1 #  
2 # This is a Shiny web application. You can run this application by clicking  
3 # the 'Run App' button above.  
4 #  
5 # Find out more about building applications with Shiny here:  
6 #  
7 #   http://shiny.rstudio.com/  
8 #  
9  
10 library(shiny)  
11  
12 # Define UI for application that draws a histogram  
13 ui <- shinyUI(fluidPage(  
14  
15   # Application title  
16   titlePanel("Old Faithful Geyser Data"),  
17  
18   # Sidebar with a slider input for number of bins  
19   sidebarLayout(  
20     # Put the title and plot in the main panel  
21     mainPanel(  
22       # Put the plot in the main panel  
23       plotOutput("plot1")  
24     ),  
25     # Put the slider input in the sidebar  
26     sidebar(  
27       # Put the slider input in the sidebar  
28       sliderInput("bins", "Number of bins:", min = 1, max = 50, value = 10)  
29     )  
30   )  
31 )  
32  
33 # Run the application (http://shiny.rstudio.com/)  
34 runApp()
```

The 'Run App' button in the top right toolbar is circled in orange. A dropdown menu is open, showing three options: 'Run in Window', 'Run in Viewer Pane' (which is selected with a checkmark), and 'Run External'. The status bar at the bottom indicates '1:1 (Top Level)' and 'R Script'.

Close an app

The screenshot shows the RStudio environment with a Shiny application running. The R script editor on the left contains the following code:

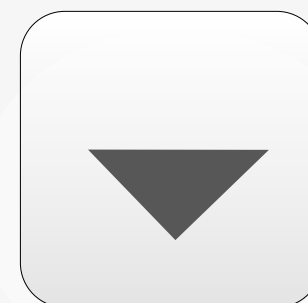
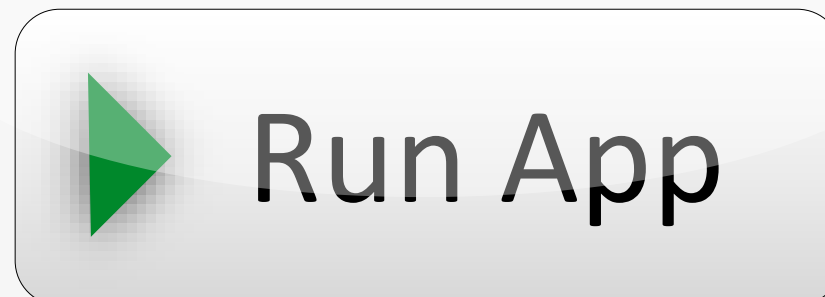
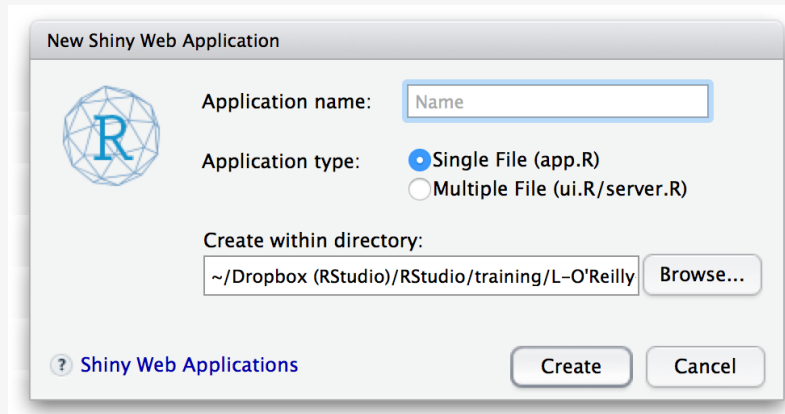
```
1 #  
2 # This is a Shiny web application. You can run the application by clicking  
3 # the 'Run App' button above.  
4 #  
5 # Find out more about building applications with Shiny here:  
6 #  
7 # http://shiny.rstudio.com/  
8 #  
9  
10 library(shiny)  
11  
12 # Define UI for application that draws a histogram  
13 ui <- shinyUI(fluidPage(  
14   # Application title  
15   titlePanel("Old Faithful Geyser Data"),  
16   # Sidebar with a slider input for number of bins  
17   sidebarLayout(  
18     #  
19     #  
20   )  
21 )  
22  
23 server <- function(input, output, session) {  
24   #  
25   #  
26 }  
27  
28 runApp()
```

The console shows the command `> runApp('02-Shiny/one')` and the message `Listening on http://127.0.0.1:3576`.

The Shiny application window on the right displays the title "Old Faithful Geyser Data" and a histogram titled "Histogram of x". The histogram shows the frequency distribution of the variable "x". The x-axis is labeled "x" and ranges from approximately 45 to 95. The y-axis is labeled "Frequency" and ranges from 0 to 25. The histogram has 30 bins, as indicated by the slider input above it.

Two orange circles highlight the "Stop" buttons: one in the top right of the Shiny application window and another in the bottom right of the RStudio interface.

EXERCISE



Open a new Shiny app with
File ▶ **New File** ▶ **Shiny Web App...**

Launch the app by opening app.R and
clicking **Run App**

Close app by clicking the stop sign icon

Select view mode in the drop down
menu next to Run App

3_m 00_s

Anatomy of a Shiny app

WHAT'S IN AN APP?

```
library(shiny)
```

```
ui <- fluidPage()
```

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

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

User interface

controls the layout and appearance of app

Server function

contains instructions needed to build app

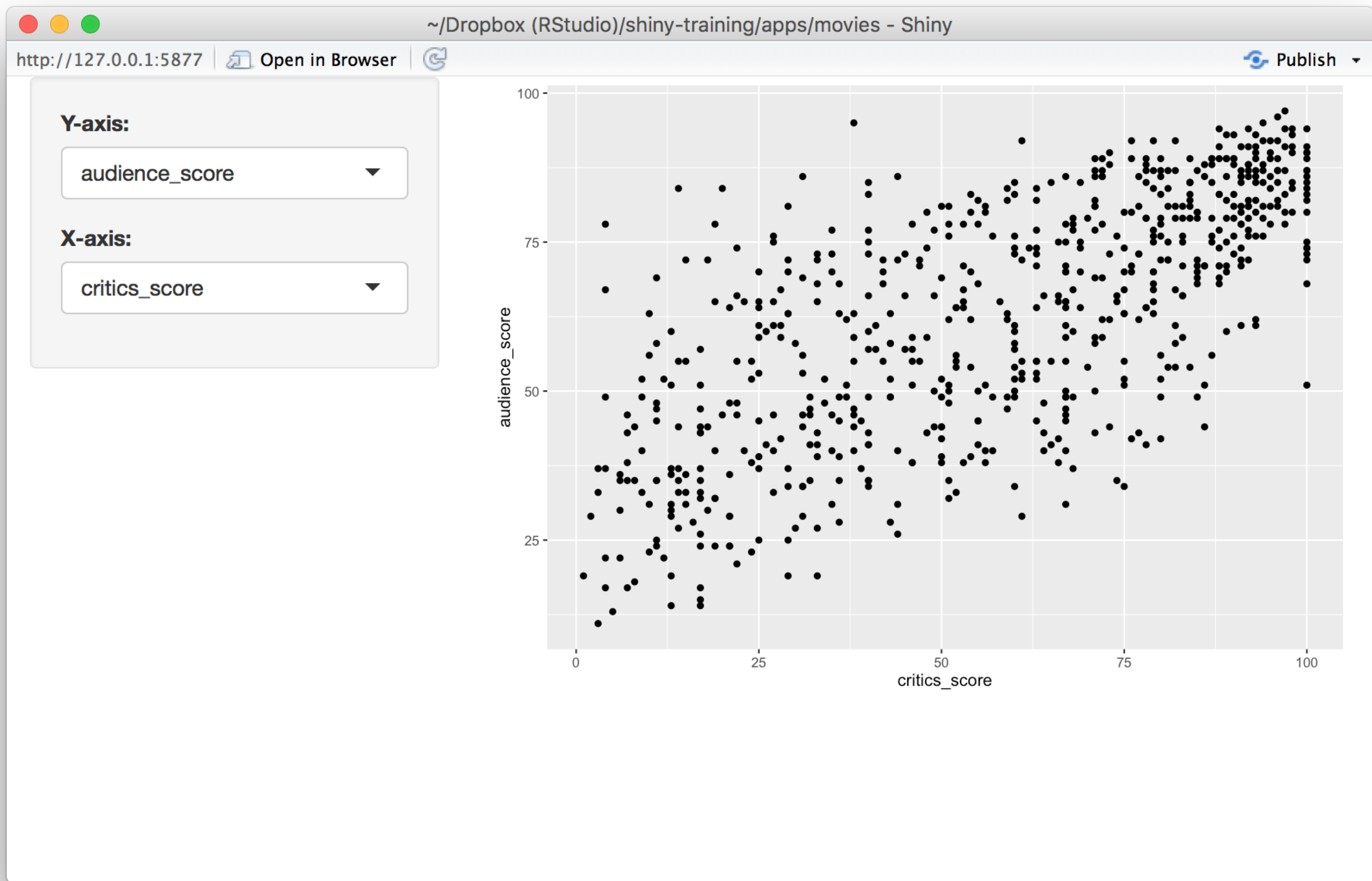


Let's build a simple movie browser app!



movies.Rdata

Data from IMDB and Rotten Tomatoes on random sample of 651 movies released in the US between 1970 and 2014



APP TEMPLATE

```
library(shiny)
library(ggplot2)
load("movies.Rdata")
ui <- fluidPage()

server <- function(input, output) {}

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



Dataset used for this app

User interface

```
# Define UI for application that plots features of movies
```

```
ui <- fluidPage(
```

```
# Sidebar layout with a input and output definitions
```

```
sidebarLayout(
```

```
# Inputs: Select variables to plot
```

```
sidebarPanel(
```

```
# Select variable for y-axis
```

```
selectInput(inputId = "y", label = "Y-axis:",  
            choices = c("imdb_rating", "imdb_num_votes", "critics_score", "audience_score", "runtime"),  
            selected = "audience_score"),
```

```
# Select variable for x-axis
```

```
selectInput(inputId = "x", label = "X-axis:",  
            choices = c("imdb_rating", "imdb_num_votes", "critics_score", "audience_score", "runtime"),  
            selected = "critics_score")
```

```
),
```

```
# Output: Show scatterplot
```

```
mainPanel(
```

```
plotOutput(outputId = "scatterplot")
```

```
)
```

```
)
```

```
)
```

Define UI for application that plots features of movies

ui <- fluidPage(
 # Sidebar layout with a input and output definitions
 sidebarLayout(
 # Inputs: Select variables to plot
 sidebarPanel(
 # Select variable for y-axis
 selectInput(inputId = "y", label = "Y-axis:",
 choices = c("imdb_rating", "imdb_num_votes", "critics_score", "audience_score", "runtime"),
 selected = "audience_score"),
 # Select variable for x-axis
 selectInput(inputId = "x", label = "X-axis:",
 choices = c("imdb_rating", "imdb_num_votes", "critics_score", "audience_score", "runtime"),
 selected = "critics_score")
),
 # Output: Show scatterplot
 mainPanel(
 plotOutput(outputId = "scatterplot")
)
)
)

Create fluid page layout

Define UI for application that plots features of movies

ui <- fluidPage(
 # Sidebar layout with a input and output definitions
 sidebarLayout(
 # Inputs: Select variables to plot
 sidebarPanel(
 # Select variable for y-axis
 selectInput(inputId = "y", label = "Y-axis:",
 choices = c("imdb_rating", "imdb_num_votes", "critics_score", "audience_score", "runtime"),
 selected = "audience_score"),
 # Select variable for x-axis
 selectInput(inputId = "x", label = "X-axis:",
 choices = c("imdb_rating", "imdb_num_votes", "critics_score", "audience_score", "runtime"),
 selected = "critics_score")
),
 # Output: Show scatterplot
 mainPanel(
 plotOutput(outputId = "scatterplot")
)
)
)
)

Create a layout with a sidebar and main area

Define UI for application that plots features of movies

ui <- fluidPage(
 # Sidebar layout with a input and output definitions
 sidebarLayout(
 # Inputs: Select variables to plot
 sidebarPanel(
 # Select variable for y-axis
 selectInput(inputId = "y", label = "Y-axis:",
 choices = c("imdb_rating", "imdb_num_votes", "critics_score", "audience_score", "runtime",
 selected = "audience_score"),
 # Select variable for x-axis
 selectInput(inputId = "x", label = "X-axis:",
 choices = c("imdb_rating", "imdb_num_votes", "critics_score", "audience_score", "runtime"),
 selected = "critics_score")
),
 # Output: Show scatterplot
 mainPanel(
 plotOutput(outputId = "scatterplot")
)
)
)
)

Create a sidebar panel containing **input** controls that can in turn be passed to **sidebarLayout**

Define UI for application that plots features of movies

ui <- fluidPage(
 # Sidebar layout with a input and output definitions
 sidebarLayout(
 # Inputs: Select variables to plot
 sidebarPanel(
 # Select variable for y-axis
 selectInput(inputId = "y", label = "Y-axis:",
 choices = c("imdb_rating", "imdb_num_votes", "critics_score", "audience_score", "runtime"),
 selected = "audience_score"),
 # Select variable for x-axis
 selectInput(inputId = "x", label = "X-axis:",
 choices = c("imdb_rating", "imdb_num_votes", "critics_score", "audience_score", "runtime"),
 selected = "critics_score"),
),
 # Output: Show scatterplot
 mainPanel(
 plotOutput(outputId = "scatterplot")
)
)
)

Y-axis:

audience_score

X-axis:

critics_score

imdb_rating

imdb_num_votes

critics_score

audience_score

runtime

Define UI for application that plots features of movies

ui <- fluidPage(
 sidebarLayout(
 # Inputs: Select variables to plot
 sidebarPanel(
 # Select variable for y-axis
 selectInput(inputId = "y", label = "Y-axis:",
 choices = c("imdb_rating", "imdb_num_votes", "critics_score", "audience_score", "runtime"),
 selected = "audience_score"),
 # Select variable for x-axis
 selectInput(inputId = "x", label = "X-axis:",
 choices = c("imdb_rating", "imdb_num_votes", "critics_score", "audience_score", "runtime"),
 selected = "critics_score")
),
 # Output: Show scatterplot
 mainPanel(
 plotOutput(outputId = "scatterplot")
)
)
)

Sidebar layout with a input and output definitions

sidebarLayout(
 # Inputs: Select variables to plot
 sidebarPanel(
 # Select variable for y-axis
 selectInput(inputId = "y", label = "Y-axis:",
 choices = c("imdb_rating", "imdb_num_votes", "critics_score", "audience_score", "runtime"),
 selected = "audience_score"),
 # Select variable for x-axis
 selectInput(inputId = "x", label = "X-axis:",
 choices = c("imdb_rating", "imdb_num_votes", "critics_score", "audience_score", "runtime"),
 selected = "critics_score")
),
 # Output: Show scatterplot
 mainPanel(
 plotOutput(outputId = "scatterplot")
)
)

sidebarPanel(
 # Select variable for y-axis
 selectInput(inputId = "y", label = "Y-axis:",
 choices = c("imdb_rating", "imdb_num_votes", "critics_score", "audience_score", "runtime"),
 selected = "audience_score"),
 # Select variable for x-axis
 selectInput(inputId = "x", label = "X-axis:",
 choices = c("imdb_rating", "imdb_num_votes", "critics_score", "audience_score", "runtime"),
 selected = "critics_score")
)

Select variable for y-axis

selectInput(inputId = "y", label = "Y-axis:",
 choices = c("imdb_rating", "imdb_num_votes", "critics_score", "audience_score", "runtime"),
 selected = "audience_score"),
Select variable for x-axis

Select variable for x-axis

selectInput(inputId = "x", label = "X-axis:",
 choices = c("imdb_rating", "imdb_num_votes", "critics_score", "audience_score", "runtime"),
 selected = "critics_score")
)

),
Output: Show scatterplot
mainPanel(
 plotOutput(outputId = "scatterplot")
)
)

Output: Show scatterplot

mainPanel(
 plotOutput(outputId = "scatterplot")
)
)

)

)

)

)

)

)

)

)

)

)

Create a main panel containing **output** elements that get created in the server function can in turn be passed to sidebarLayout

Server function

```
# Define server function required to create the scatterplot
server <- function(input, output) {

  # Create the scatterplot object the plotOutput function is expecting
  output$scatterplot <- renderPlot({
    ggplot(data = movies, aes_string(x = input$x, y = input$y)) +
      geom_point()
  })
}
```


Define server function required to create the scatterplot

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

Create the scatterplot object the plotOutput function is expecting

`output$scatterplot` <- `renderPlot({`

`ggplot(data = movies, aes_string(x = input$x, y = input$y)) +`

`geom_point()`

`})`

`}`

Contains instructions
needed to build app

Define server function required to create the scatterplot

server <- function(input, output) {

Create the scatterplot object the plotOutput function is expecting

output\$scatterplot <- renderPlot({

ggplot(data = movies, aes_string(x = input\$x, y = input\$y)) +

geom_point()

})

}

Renders a **reactive** plot that is suitable for assigning to an output slot

Define server function required to create the scatterplot

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

Create the scatterplot object the plotOutput function is expecting

`output$scatterplot` <- `renderPlot({`

`ggplot(data = movies, aes_string(x = input$x, y = input$y))` +

`geom_point()`

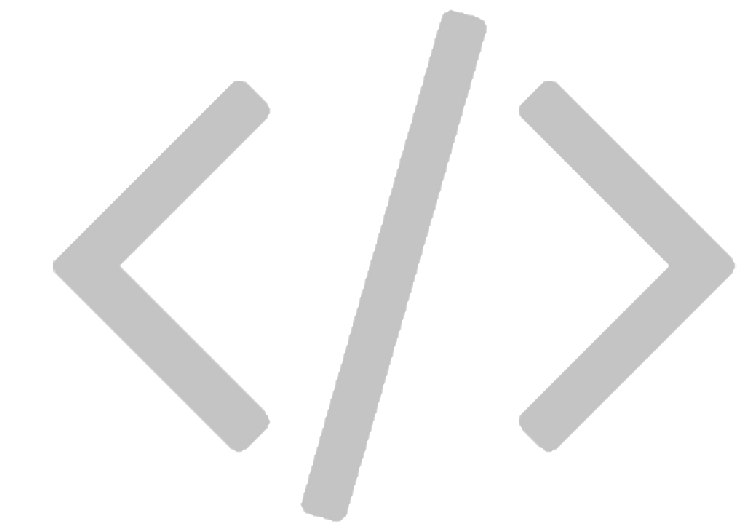
`}`

`}`

Good ol' ggplot2 code,
with **inputs** from UI

Running the app

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



DEMO

Putting it all together...

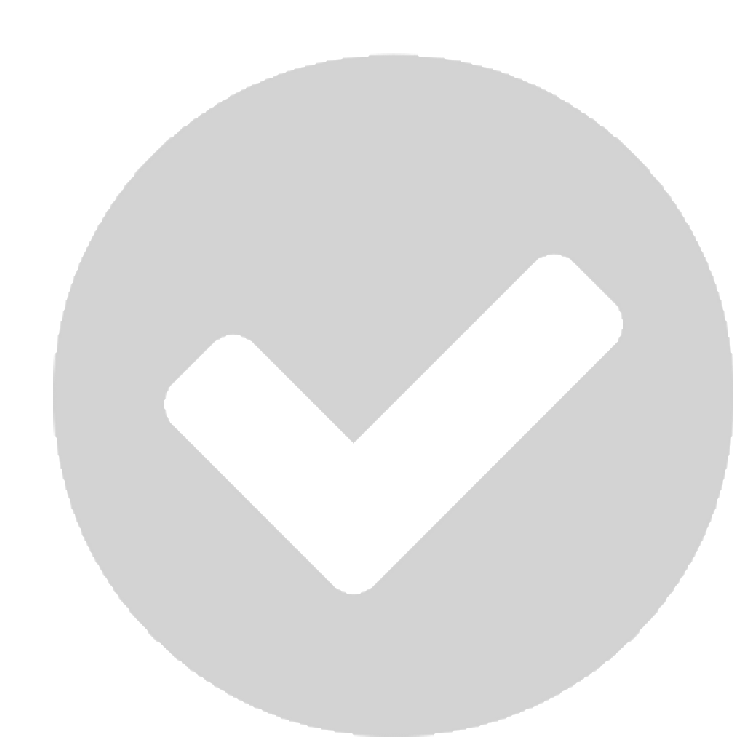
movies_01.R

EXERCISE



- ▶ Add new select menu to color the points by
 - ▶ `inputId = "z"`
 - ▶ `label = "Color by:"`
 - ▶ `choices = c("title_type", "genre", "mpaa_rating", "critics_rating", "audience_rating")`
 - ▶ `selected = "mpaa_rating"`
- ▶ Use this variable in the aesthetics of the `ggplot` function as the color argument to color the points by
- ▶ Run the app in the Viewer Pane
- ▶ Compare your code / output with the person sitting next to / nearby you

5_m 00_s

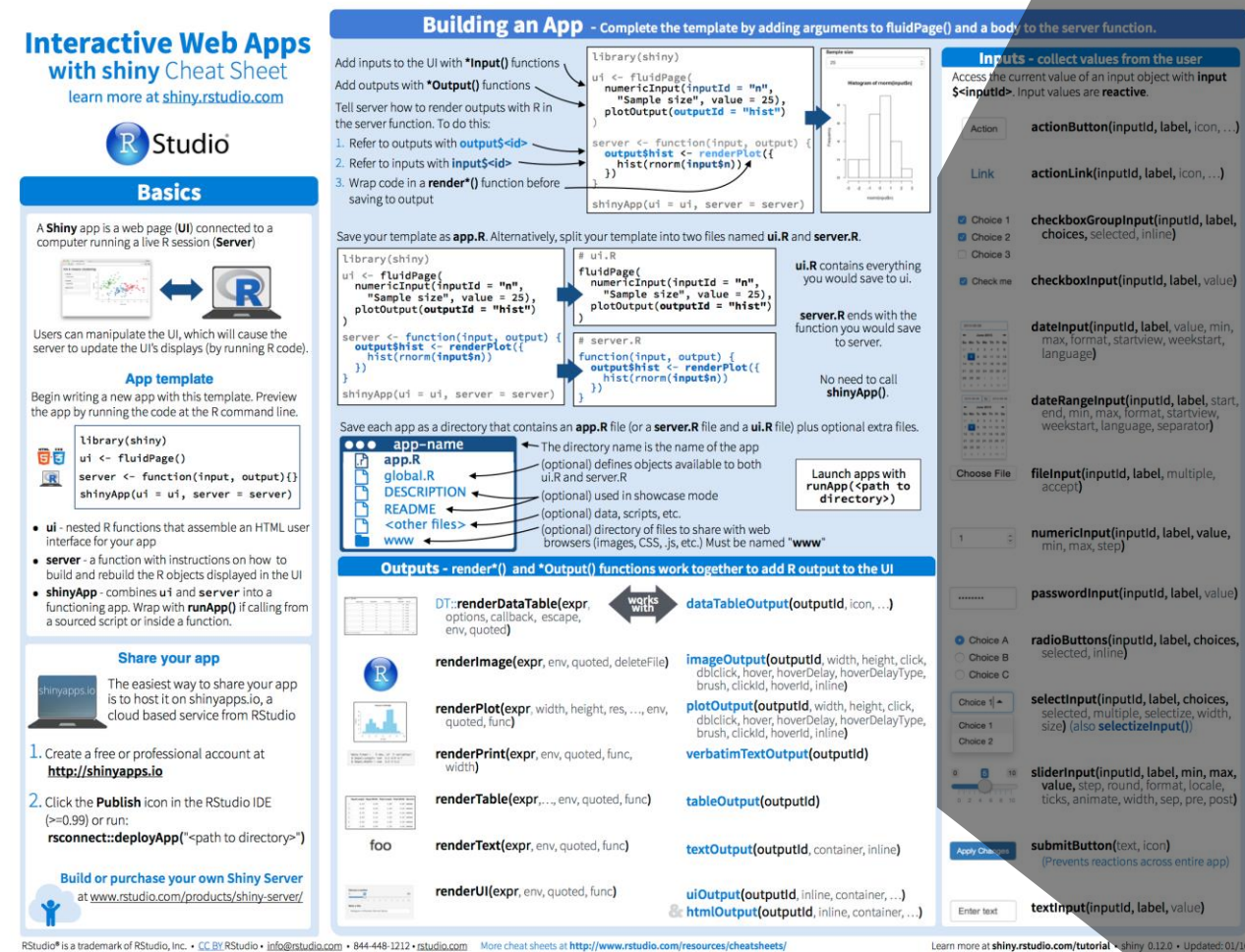


SOLUTION

Solution to the previous exercise

`movies_02.R`

INPUTS



Action

actionButton(inputId, label, icon, ...)

Link

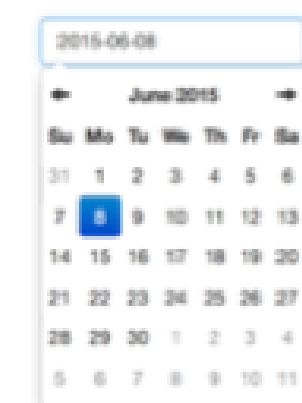
actionLink(inputId, label, icon, ...)

- ☒ Choice 1
- ☒ Choice 2
- ☐ Choice 3

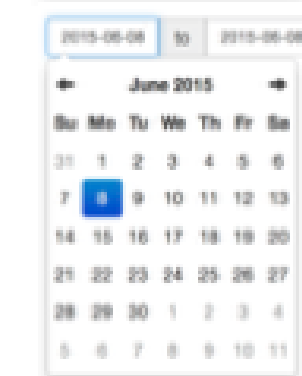
checkboxGroupInput(inputId, label, choices, selected, inline)

- ☒ Check me

checkboxInput(inputId, label, value)



dateInput(inputId, label, value, min, max, format, startview, weekstart, language)



dateRangeInput(inputId, label, start, end, min, max, format, startview, weekstart, language, separator)

Choose File

fileInput(inputId, label, multiple, accept)

numericInput(inputId, label, value, min, max, step)

passwordInput(inputId, label, value)

- ☒ Choice A
- ☐ Choice B
- ☐ Choice C

radioButtons(inputId, label, choices, selected, inline)

selectInput(inputId, label, choices, selected, multiple, selectize, width, size) (also **selectizeInput())**

sliderInput(inputId, label, min, max, value, step, round, format, locale, ticks, animate, width, sep, pre, post)

Apply Changes

submitButton(text, icon)
(Prevents reactions across entire app)

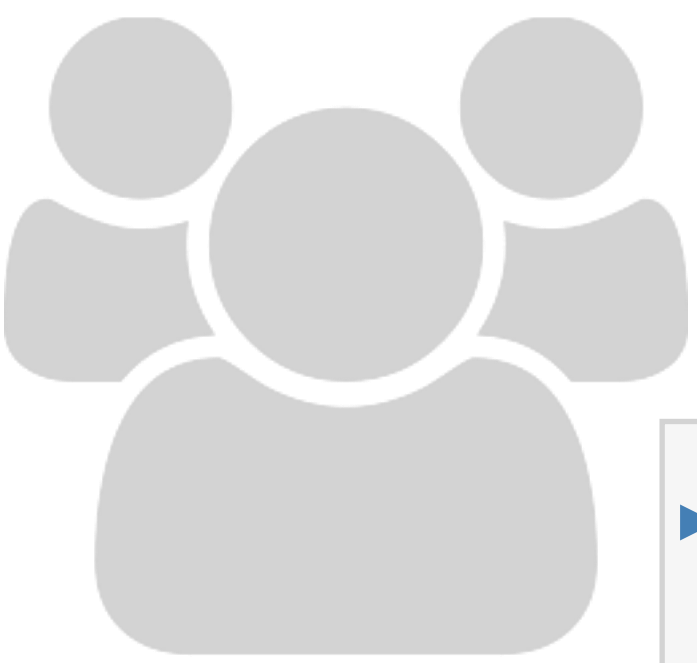
textInput(inputId, label, value)

Shiny from



<https://raw.githubusercontent.com/rstudio/cheatsheets/main/shiny.pdf>

EXERCISE



- ▶ Add new input variable to control the alpha level of the points
 - ▶ This should be a sliderInput
 - ▶ See shiny.rstudio.com/reference/shiny/latest/ for help
 - ▶ Values should range from 0 to 1
 - ▶ Set a default value that looks good
- ▶ Use this variable in the geom of the ggplot function as the alpha argument
- ▶ Run the app in a new window
- ▶ Compare your code / output with the person sitting next to / nearby you

5_m 00_s



SOLUTION

Solution to the previous exercise

`movies_03.R`

OUTPUTS

Interactive Web Apps
with shiny Cheat Sheet
learn more at shiny.studio.com

Basics

A Shiny app is a web page (UI) connected to a computer running a live R session (Server).

Users can manipulate the UI, which will cause the server to update the UI's displays (by running R code).

App template

Begin writing a new app with this template. Preview the app by running the code at the R command line.

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

- **ui** - nested R functions that assemble an HTML user interface for your app
- **server** - a function with instructions on how to build and rebuild the R objects displayed in the UI
- **shinyApp** - combines ui and server into a functioning app. Wrap with `runApp()` if calling from a sourced script or inside a function.

Share your app

The easiest way to share your app is to host it on shinyapps.io, a cloud based service from RStudio

1. Create a free or professional account at <http://shinyapps.io>
2. Click the Publish icon in the RStudio IDE (>=v0.99) or run:
`rconnect::deployApp("~/path to directory")`

Build or purchase your own Shiny Server at www.rstudio.com/products/shiny-server/

RStudio® is a trademark of RStudio, Inc. • © 2017 RStudio • info@rstudio.com • 944-448-1212 • xpub.com More cheat sheets at <http://www.rstudio.com/resources/cheatsheets/> Learn more at shiny.studio.com/tutorial • shiny 0.12.0 • Updated: 9/1/16

Building an App - Complete the template by adding arguments to `fluidPage()` and a body to the server function.

Add inputs to the UI with ***input()** functions

Add outputs with ***output()** functions

Tell server how to render outputs with R in the server function. To do this:

1. Refer to outputs with `output$ids`
2. Refer to inputs with `input$ids`
3. Wrap code in a `render*` function before saving to output

Save your template as **app.R**. Alternatively, split your template into two files named **ui.R** and **server.R**.

ui.R contains everything you would save to ui.

server.R ends with the function you would save to server.

No need to call `shinyApp()`.

Save each app as a directory that contains an **app.R** file (or a **server.R** file and a **ui.R** file) plus optional extra files.

The directory name is the name of the app

Optional: defines objects available to both ui.R and server.R

Optional: used in showcase mode

Optional: data, scripts, etc.

Optional: directory of files to share with web browsers (images, CSS, js, etc.) Must be named "www"

Launch apps with `runApp()` (path to directory)

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

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)

foo

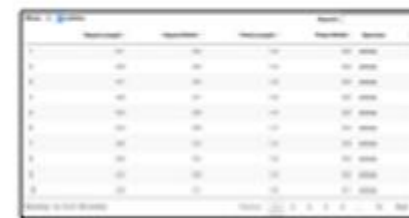
renderText(expr, env, quoted, func)

textOutput(outputId, container, inline)

renderUI(expr, env, quoted, func)

uiOutput(outputId, inline, container, ...)

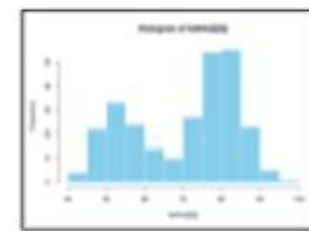
htmlOutput(outputId, inline, container, ...)



DT::renderDataTable(expr, options, callback, escape, env, quoted)



dataTableOutput(outputId, icon, ...)



renderImage(expr, env, quoted, deleteFile)

renderPlot(expr, width, height, res, ..., env, quoted, func)

renderPrint(expr, env, quoted, func, width)

renderTable(expr, ..., env, quoted, func)

renderText(expr, env, quoted, func)

renderUI(expr, env, quoted, func)

imageOutput(outputId, width, height, click, dblclick, hover, hoverDelay, hoverDelayType, brush, clickId, hoverId, inline)

plotOutput(outputId, width, height, click, dblclick, hover, hoverDelay, hoverDelayType, brush, clickId, hoverId, inline)

verbatimTextOutput(outputId)

tableOutput(outputId)

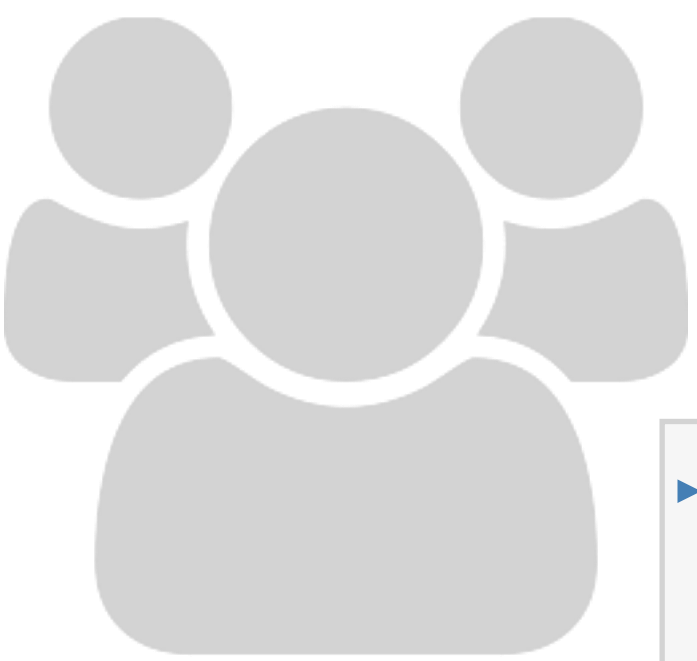
textOutput(outputId, container, inline)

uiOutput(outputId, inline, container, ...)
& **htmlOutput**(outputId, inline, container, ...)

Shiny from



EXERCISE



- ▶ Add a checkbox input to decide whether the data plotted should be shown in a data table
 - ▶ This should be a checkboxInput (see shiny.rstudio.com/reference/shiny/latest/ for help)
- ▶ Create a new output item using DT::renderDataTable, an if statement to check if the box is checked, and DT::datatable
 - ▶ Show first seven columns of movies data, show 10 rows at a time, and hide row names, e.g.
 - ▶ data = movies[, 1:7]
 - ▶ options = list(pageLength = 10)
 - ▶ rownames = FALSE
- ▶ Add a dataTableOutput to the main panel
- ▶ Run the app in a new Window, check and uncheck the box to test functionality
- ▶ Compare your code / output with the person sitting next to / nearby you
- ▶ **Optional:** If you finish early, move on to the next exercise

5_m 00_s

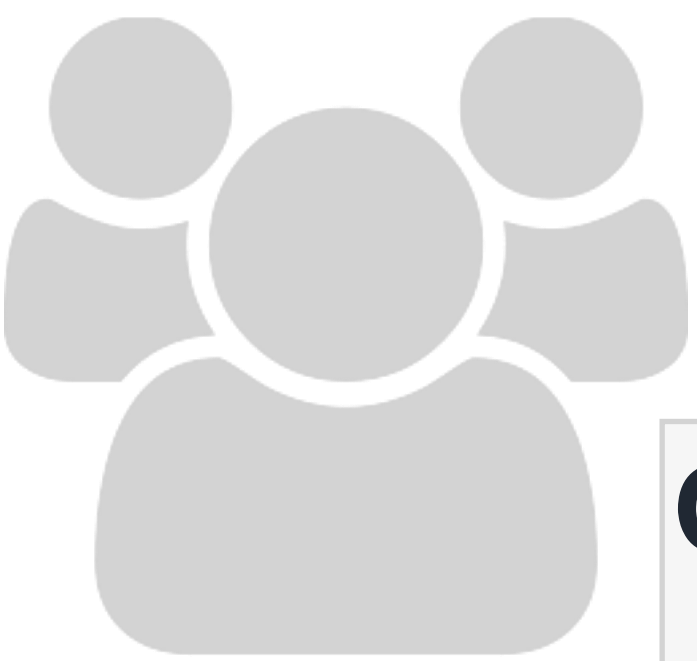


SOLUTION

Solution to the previous exercise

`movies_04.R`

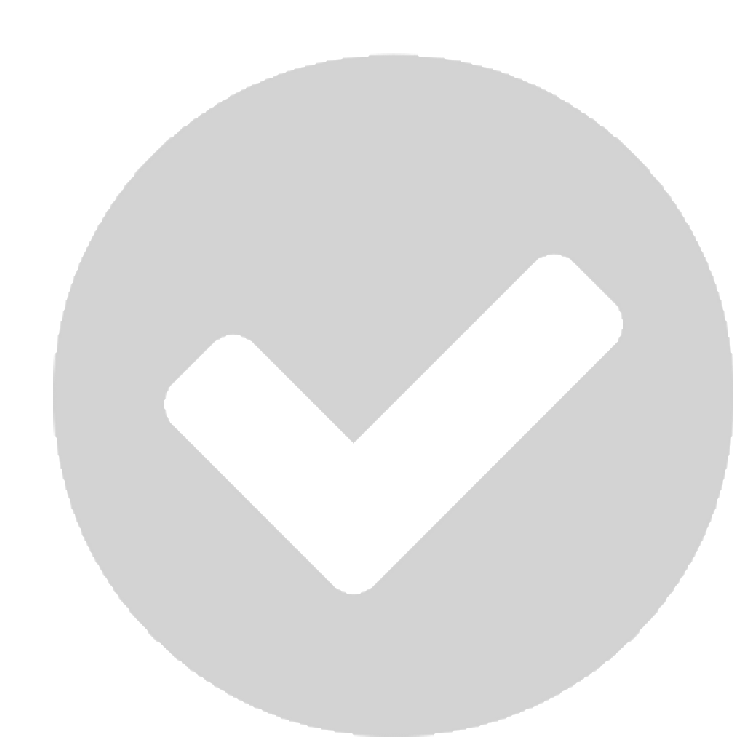
EXERCISE



Optional: If you finish the previous exercise early

- ▶ Add a title to your app with `titlePanel`, which goes before the `sidebarLayout`
- ▶ Prettify the variable names shown as input choices. *Hint:*
 - ▶ `choices = c("IMDB rating" = "imdb_rating", ...)`
- ▶ Prettify the axis and legend labels of your plot. *Hint:* You might use
 - ▶ `str_replace_all` from the `stringr` package
 - ▶ `toTitleCase` from the `tools` package

5_m 00_s



SOLUTION

Solution to the previous exercise

movies_05.R

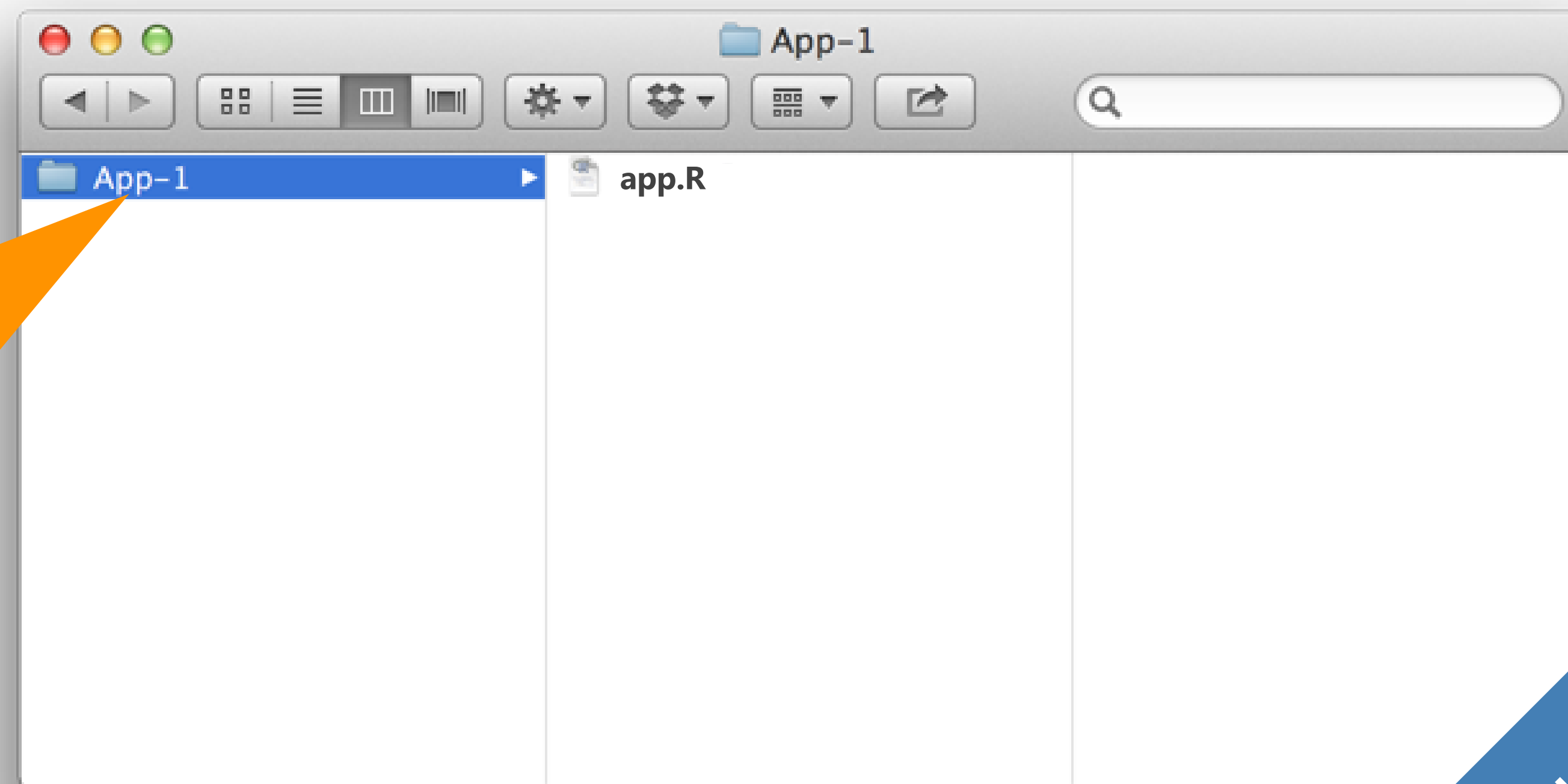
File structure

SAVING YOUR SINGLE FILE 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.

We will focus on the single file format throughout the course

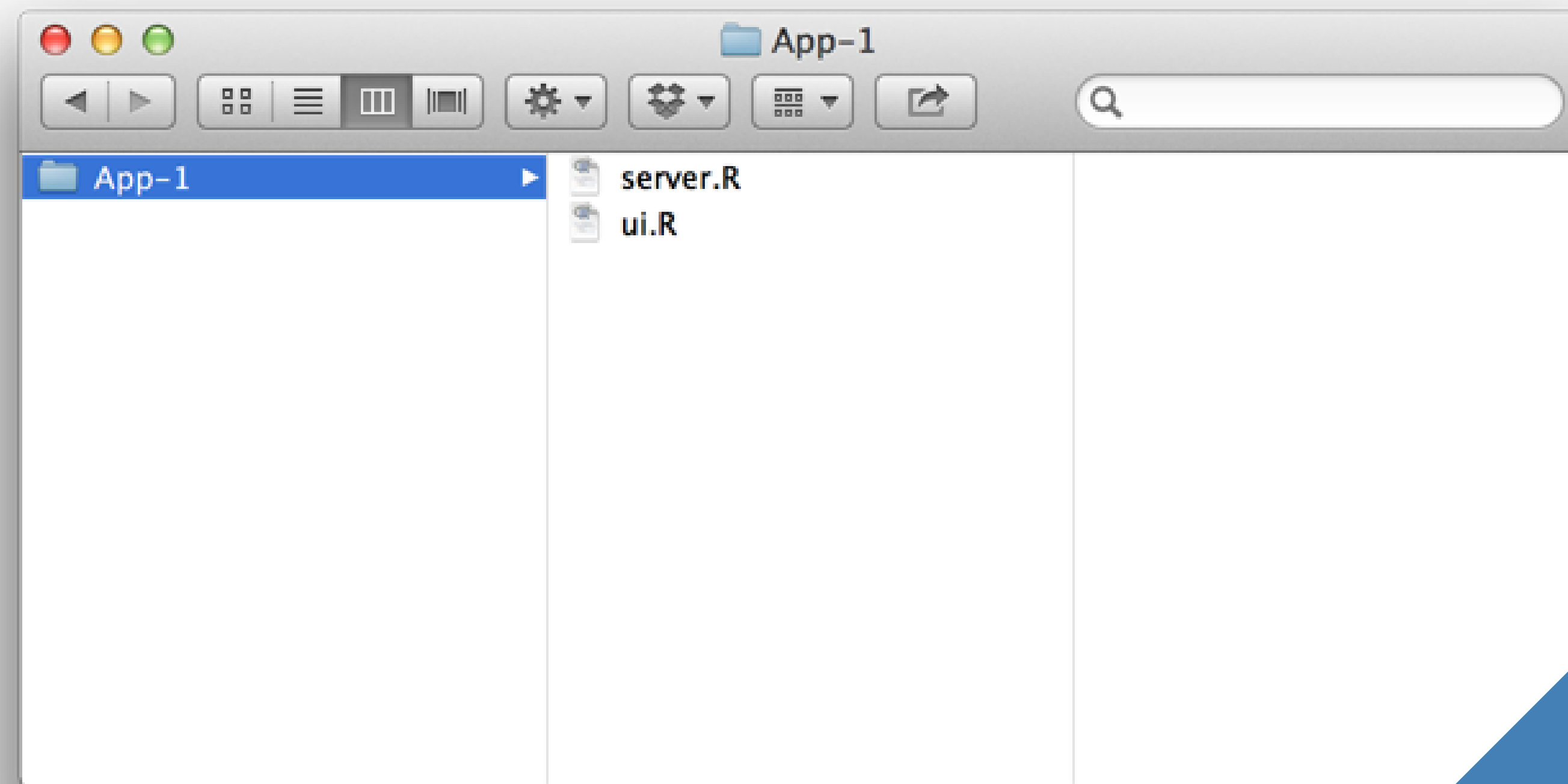


You must use this exact name (app.R) for deploying the app to shinyapps.io

SAVING YOUR MULTIPLE FILE APP

One directory with every file the app needs:

- ▶ ui.R and server.R
- ▶ datasets, images, css, helper scripts, etc.



You must use these
exact names

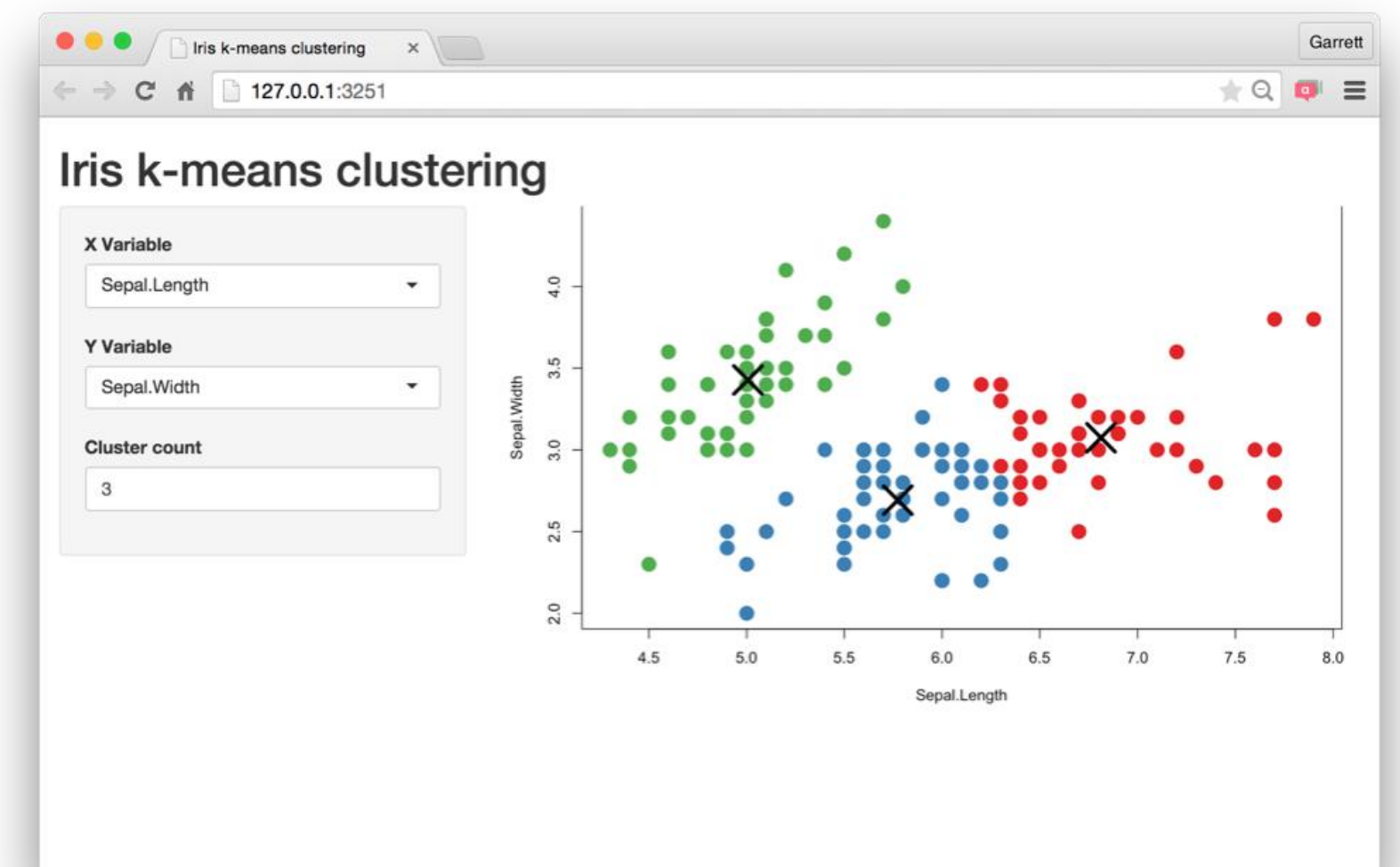
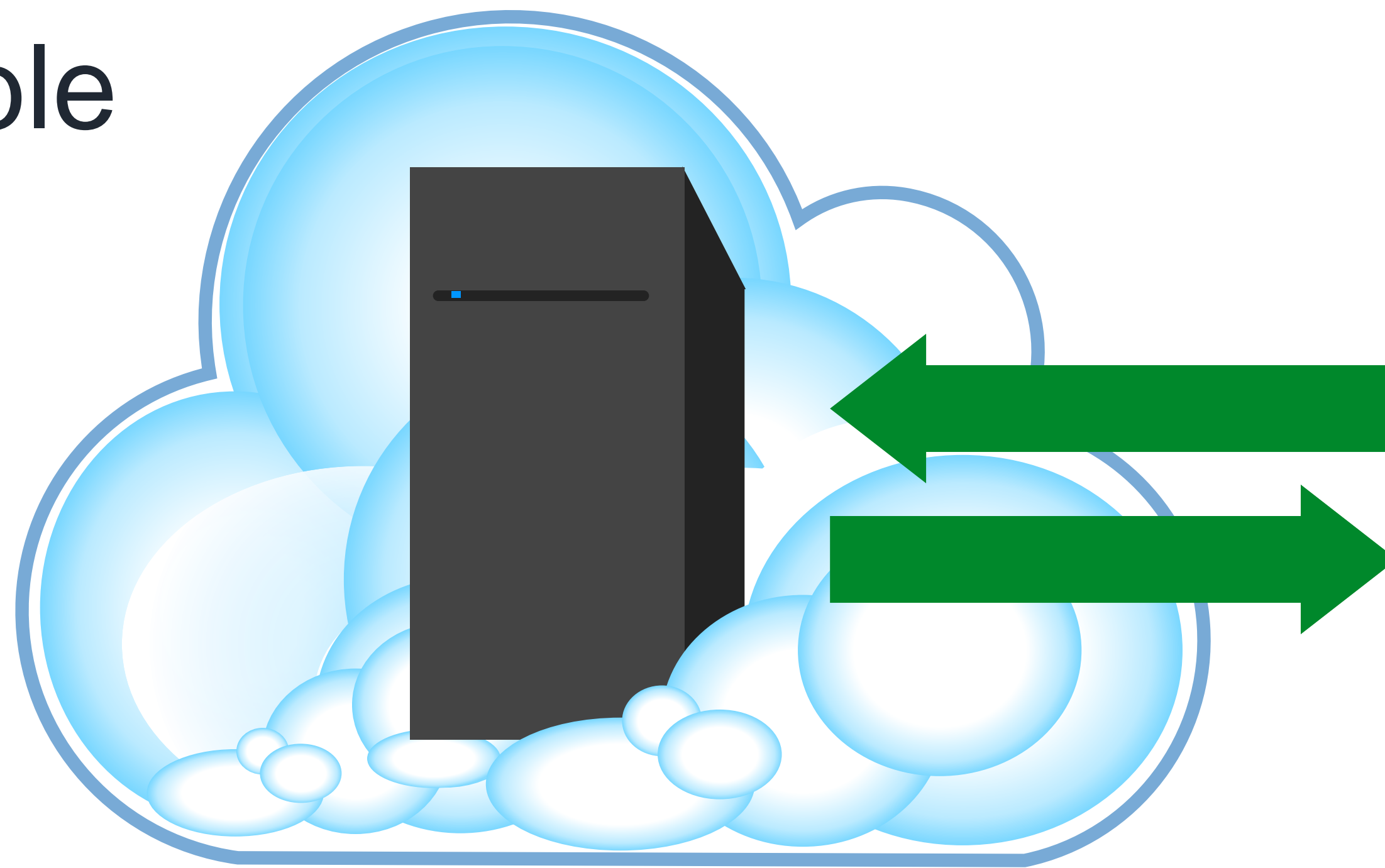
Sharing
your app

shinyapps.io

SHINYAPPS.IO

A server maintained by RStudio

- ▶ easy to use
- ▶ secure
- ▶ scalable



HASSLE-FREE CLOUD HOSTING FOR SHINY

shinyapps.io by Posit

[Home](#)

[Features](#)

[Pricing](#)

[Support](#)

[Log In](#)

Share your Shiny Applications Online

Deploy your Shiny applications on the Web in minutes

[Sign Up](#)



Easy To Use



Secure

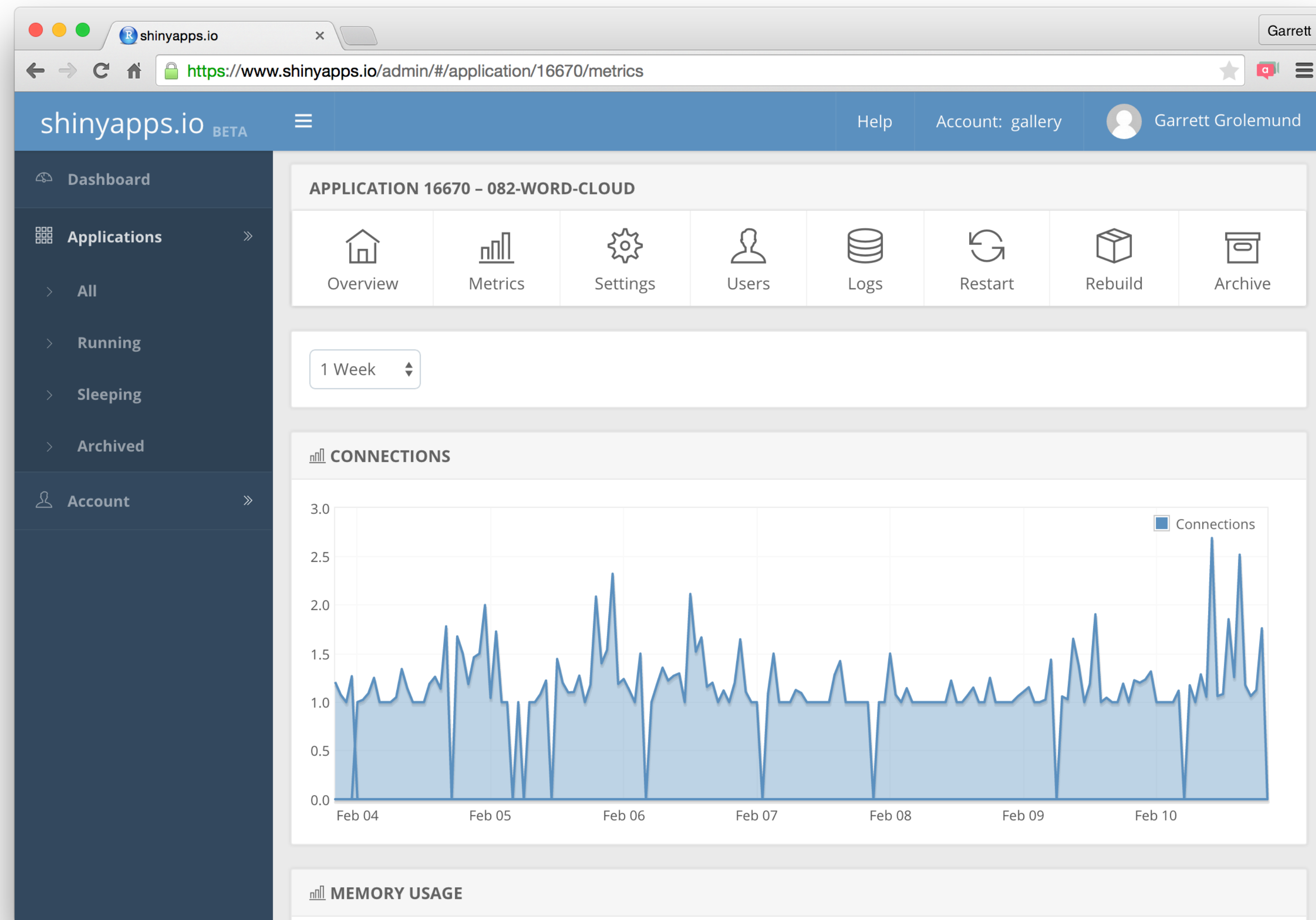


Scalable

Shiny from



WITH BUILT-IN METRICS




MEMBERSHIP PRICING

FREE	STARTER	BASIC	STANDARD	PROFESSIONAL
\$0 /month	\$13 /month (or \$145/year)	\$49 /month (or \$550/year)	\$119 /month (or \$1,330/year)	\$349 /month (or \$3,860/year)
New to Shiny? Deploy your applications for FREE.	More applications. More active hours!	Take your users to the next level!	Password protection? Authenticate your users!	Professional has it all! Personalize your domains.
5 Applications	25 Applications	Unlimited Applications	Unlimited Applications	Unlimited Applications
25 Active Hours	100 Active Hours	500 Active Hours	2,000 Active Hours	10,000 Active Hours
✓ Community Support	✓ Premium Email Support	✓ Performance Boost ✓ Premium Email Support	✓ Authentication ✓ Performance Boost ✓ Premium Email Support	✓ Authentication ✓ Account Sharing ✓ Performance Boost ✓ Custom Domains ✓ Premium Email Support
Sign Up Now	Sign Up Now	Sign Up Now	Sign Up Now	Sign Up Now

POSIT CONNECT

<https://posit.co/products/enterprise/connect/>


- 
- ✓ **Push-button publish from RStudio**
Shiny apps, R Markdown docs, and more
 - ✓ **Self-managed content**
content authors decide permissions
 - ✓ **Scheduled reports**
automatically run and email Rmd
 - ✓ **Support**
direct priority support

evaluation free trial

Build your own
server

SHINY SERVER

<https://posit.co/products/open-source/shinyserver/>

- 
- ✓ **Deploy Shiny apps to the internet**
 - ✓ **Run on-premises**
move computation closer to the data
 - ✓ **Host multiple apps on one server**
 - ✓ **Deploy inside the firewall**
 - ✓ **xcopy deployment**

Free &
open source

EXERCISE

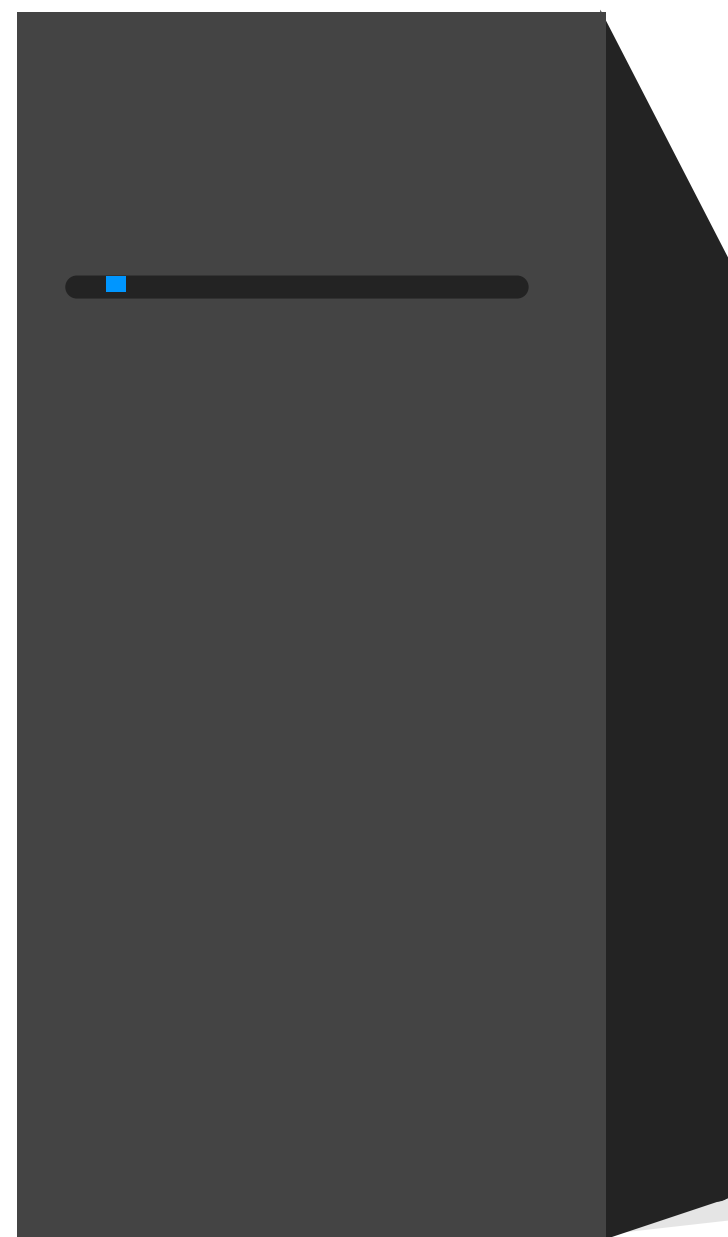


- ▶ Create a folder called movies in the ShinyApps folder
- ▶ Move any one of the movies app R scripts you worked on into this folder, and rename it as app.R
- ▶ Also move the movies.Rdata file into this folder
- ▶ Run the app
- ▶ Rstudio will take you to a browser or local view where you can interact with the deployed app

3_m 00_s

SHINYPROXY

<https://shinyproxy.io/>



- ✓ **Secure access**
LDAP, GoogleAuth, SSL, and more
- ✓ **Management**
View open apps and generate usage statistics
- ✓ **Docker**
uses docker to maintain dedicated app instances
- ✓ **Open Source**
- ✓ **ACL's**
Configurable to restrict app access

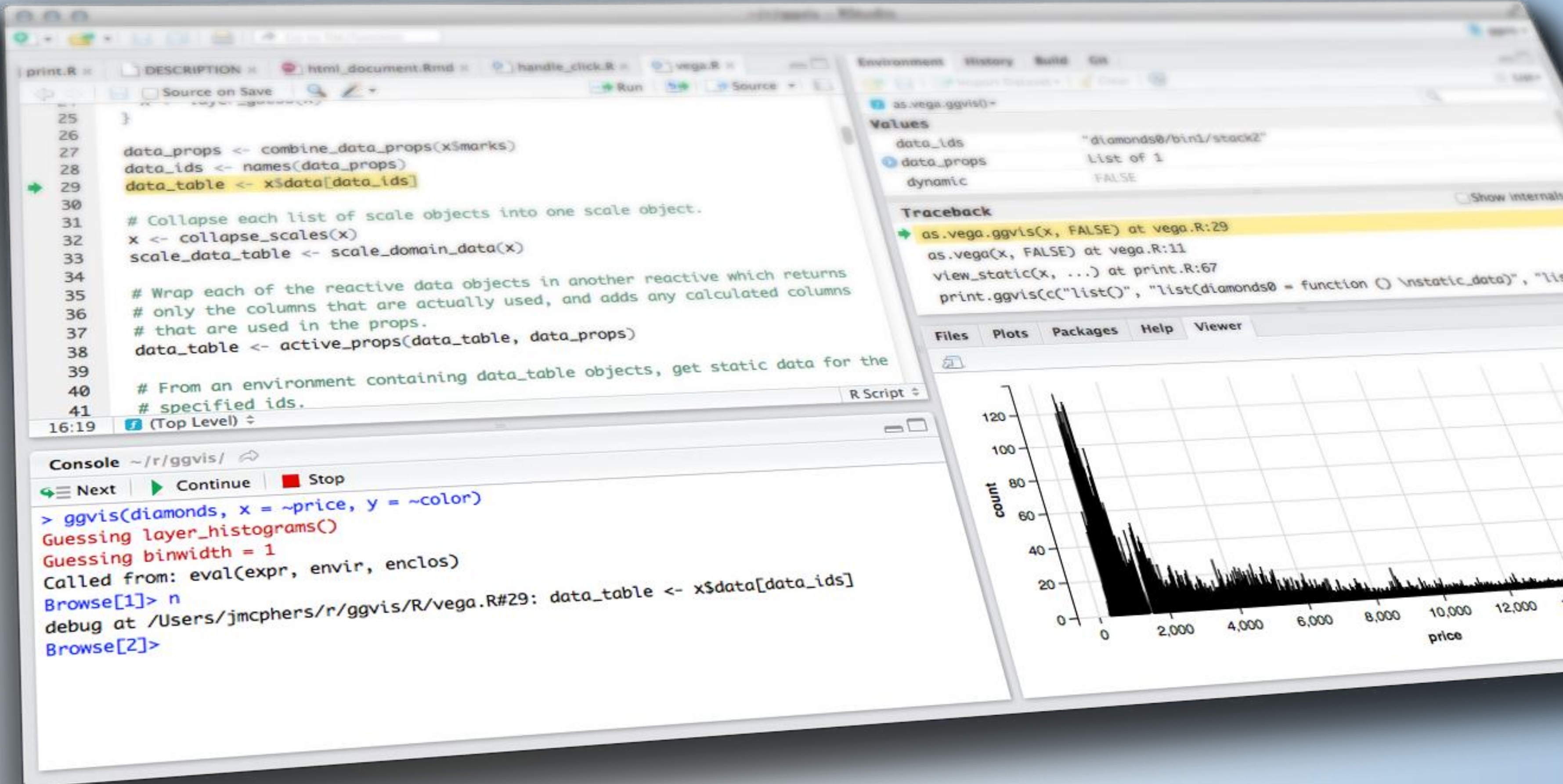
Free & open
source!

More information

- ▶ Shinyproxy documentation:
<https://shinyproxy.io/documentation/deploying-apps/>
- ▶ Shinyproxy github repo:
<https://github.com/openanalytics/shinyproxy>

How do I deploy apps?





COURSE OVERVIEW

& INTRODUCTION TO SHINY

Shiny from

