

# A FAST INTRODUCTION TO SHINY

The screenshot shows the RStudio interface with several panes:

- Code Editor:** Shows a script named `print.R` with code related to ggvis and Vega. Line 29 highlights `x$data[data_ids]`. Lines 30-41 show comments about collapsing scales and wrapping reactive data objects.
- Console:** Shows the command `> ggvis(diamonds, x = ~price, y = ~color)` being run. The output includes messages about guessing histograms and binwidth, and a stack trace indicating the code was debugged at `/Users/jmcphers/r/ggvis/R/vega.R#29`.
- Environment:** Shows the environment for `as.vega.ggviz()` with variables `data_ids`, `data_props`, and `dynamic`.
- Traceback:** Shows the call stack for `as.vega.ggviz(x, FALSE)`.
- Plots:** A histogram titled "diamonds0/bin1/stack2" showing the distribution of price. The x-axis is labeled "price" and ranges from 0 to 12,000. The y-axis is labeled "count" and ranges from 0 to 120.

Shiny from R Studio

# BACKGROUND

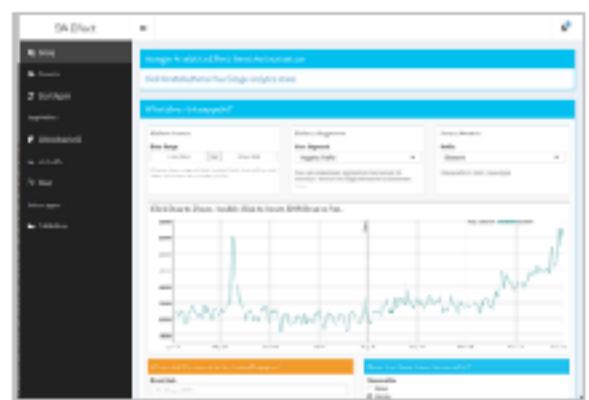
- ▶ You know what a Shiny app is, and may have even made a few of them.
- ▶ You are familiar with R as a programming language.

# SHINY SHOWCASE

## Shiny User Showcase

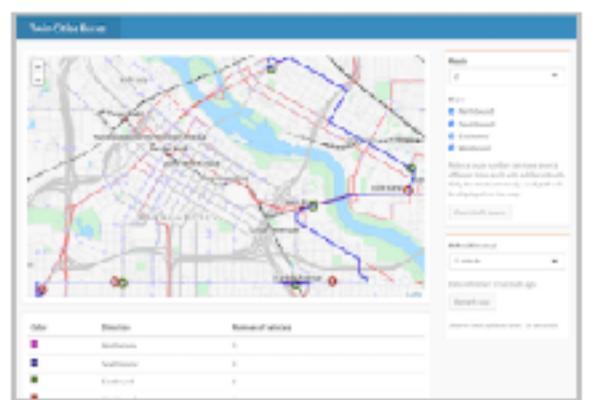
Shiny is designed for fully interactive visualization, using JavaScript libraries like d3, Leaflet, and Google Charts. Our users create fantastic examples, and some have shared them with the community. Here are some examples that we particularly like.

### Shiny Apps for the Enterprise



#### MARKETING EFFECTS

See the effects of your marketing campaigns.



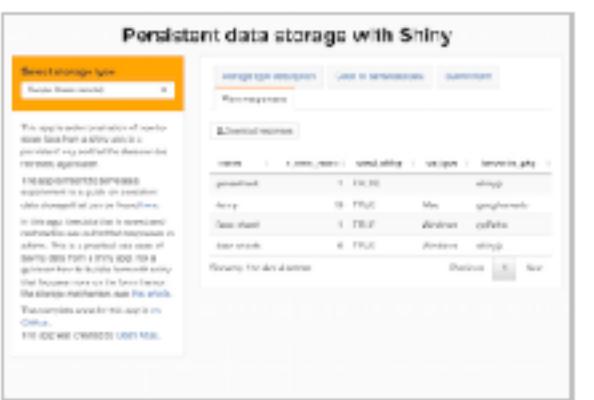
#### LOCATION TRACKER

Track locations over time with streaming data.



#### DOWNLOAD MONITOR

Streaming download rates visualized as a bubble chart.

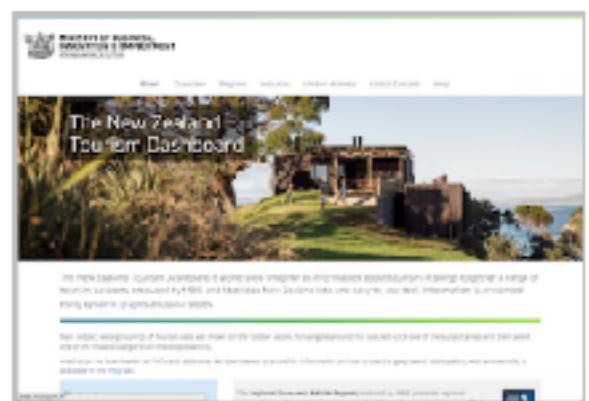


#### PERSISTENT STORAGE

Save data from your apps to local files, servers, databases, and more.

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

### Industry Specific Shiny Apps



#### TOURISM DASHBOARD



#### GENOME BROWSER



#### ER OPTIMIZATION



#### SUPPLY AND DEMAND

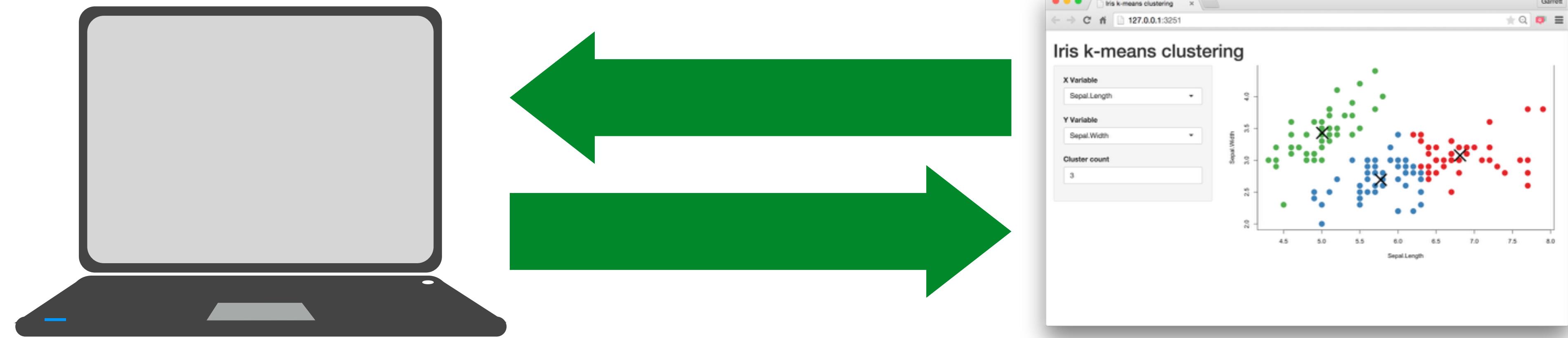
# OUTLINE

- ▶ High level view
- ▶ Anatomy of a Shiny app
  - ▶ User interface
  - ▶ Server function
  - ▶ Running the app
- ▶ File Structure
- ▶ Sharing your app

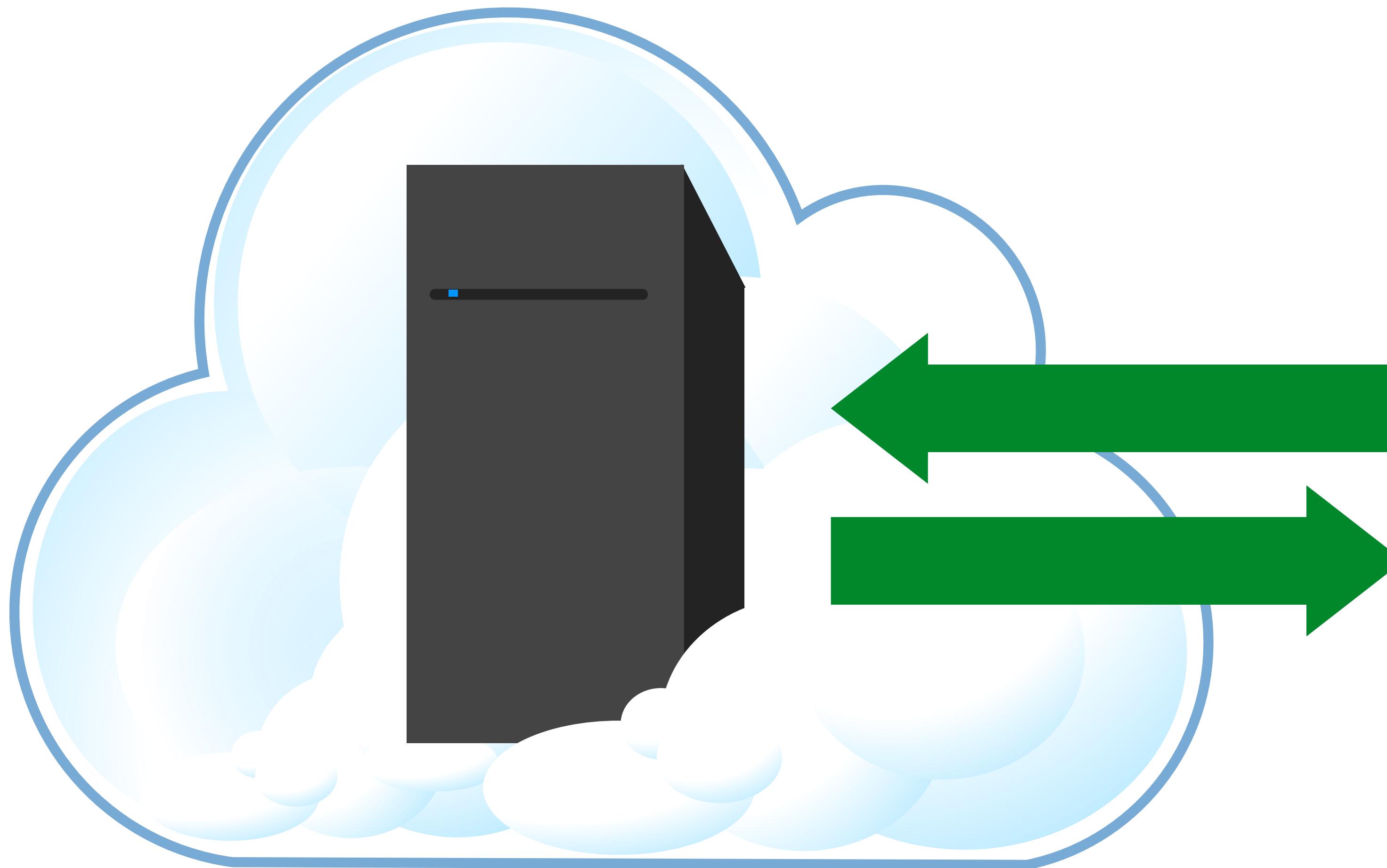
High level

view

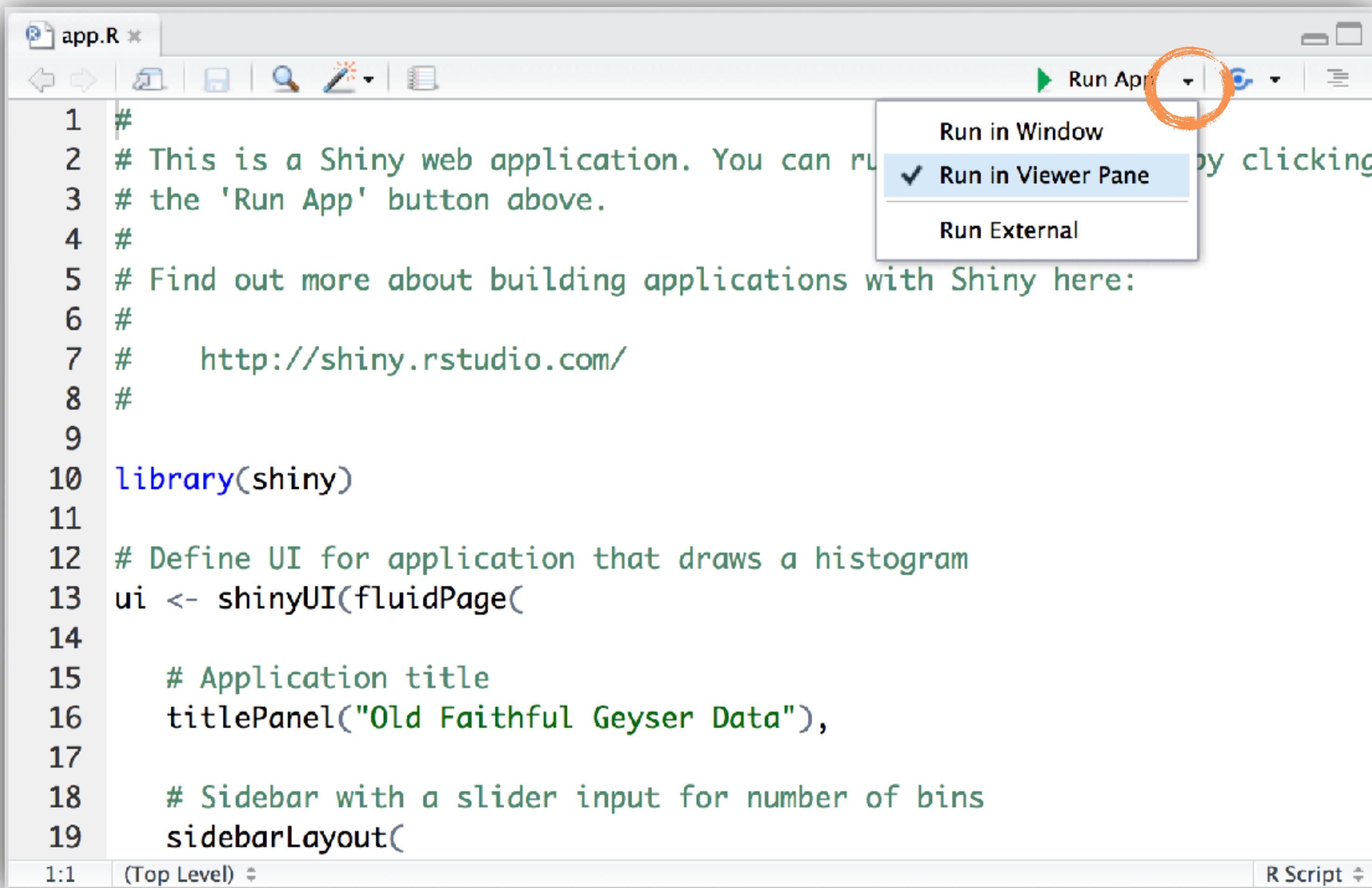
# Every Shiny app is maintained by a computer running R



# Every Shiny app is maintained by a computer running R



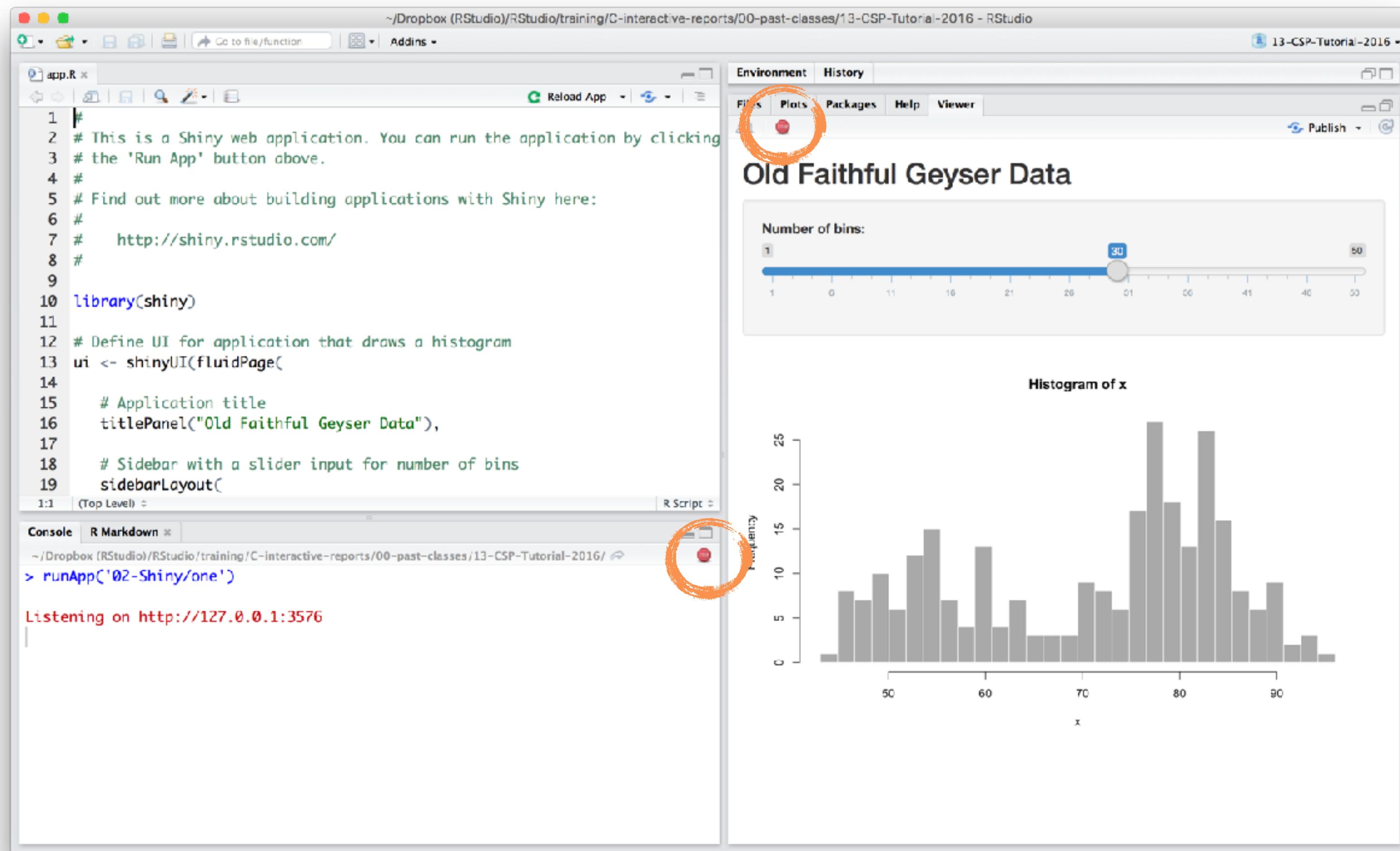
# Change display



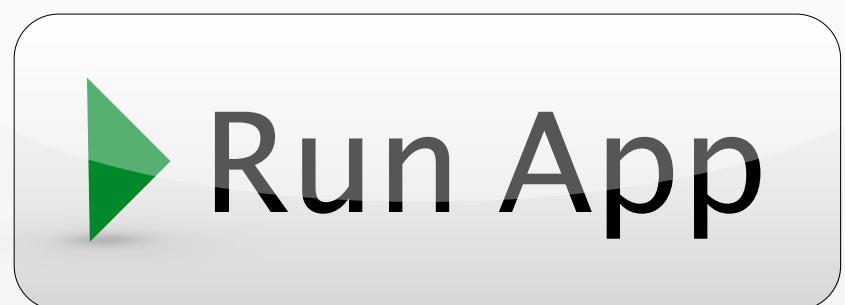
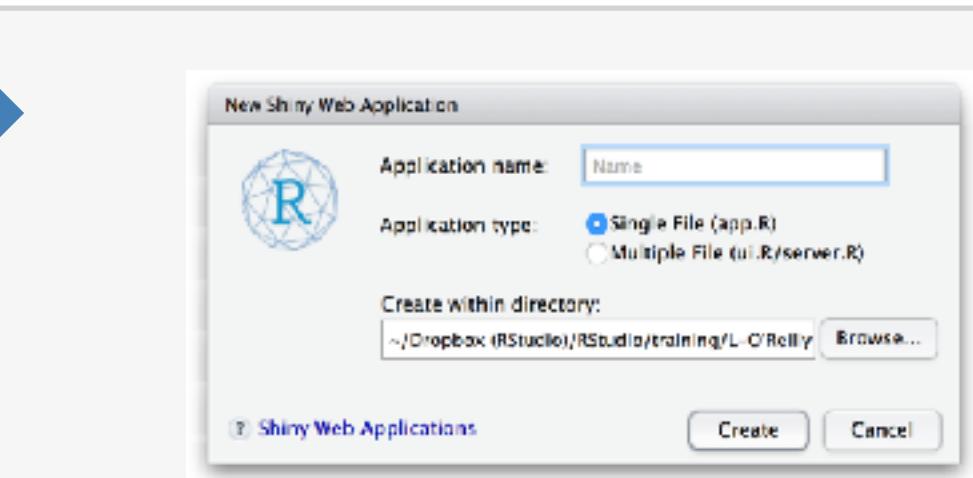
The screenshot shows the RStudio interface with the 'app.R' script open. The 'Run App' button in the toolbar has a red circle drawn around it, indicating it's the focus of the demonstration. A context menu is open from the 'Run App' button, listing three options: 'Run in Window', 'Run in Viewer Pane' (which is checked), and 'Run External'. The code in the script is a basic Shiny application template.

```
1 #  
2 # This is a Shiny web application. You can run  
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(
```

# Close an app



# 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

3m 00s

# Anatomy of a shiny app

# WHAT'S IN AN APP?

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

## User interface

controls the layout and appearance of app

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

## Server function

contains instructions needed to build app

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

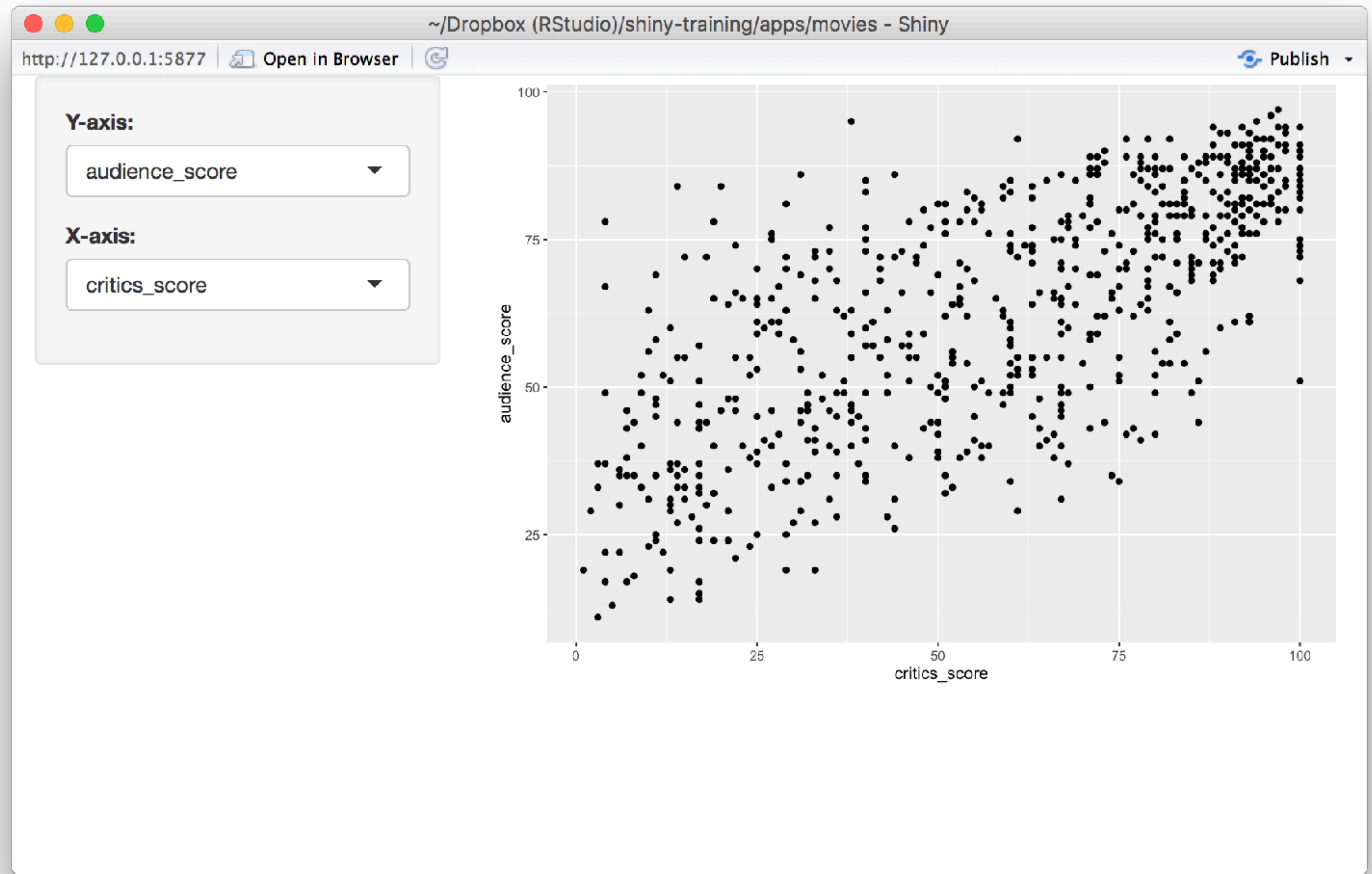


# Let's build a simple movie browser app!



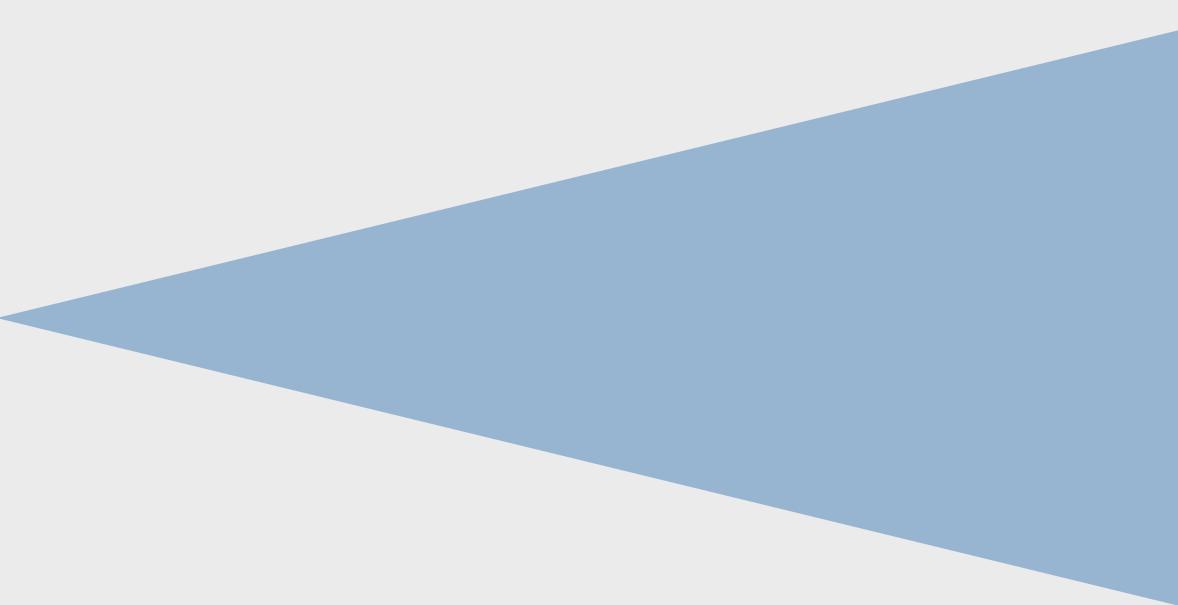
## movies.Rdata

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()
```



Dataset used for this app

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

# 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( Create fluid page layout

  # 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 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"),
                  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(
```

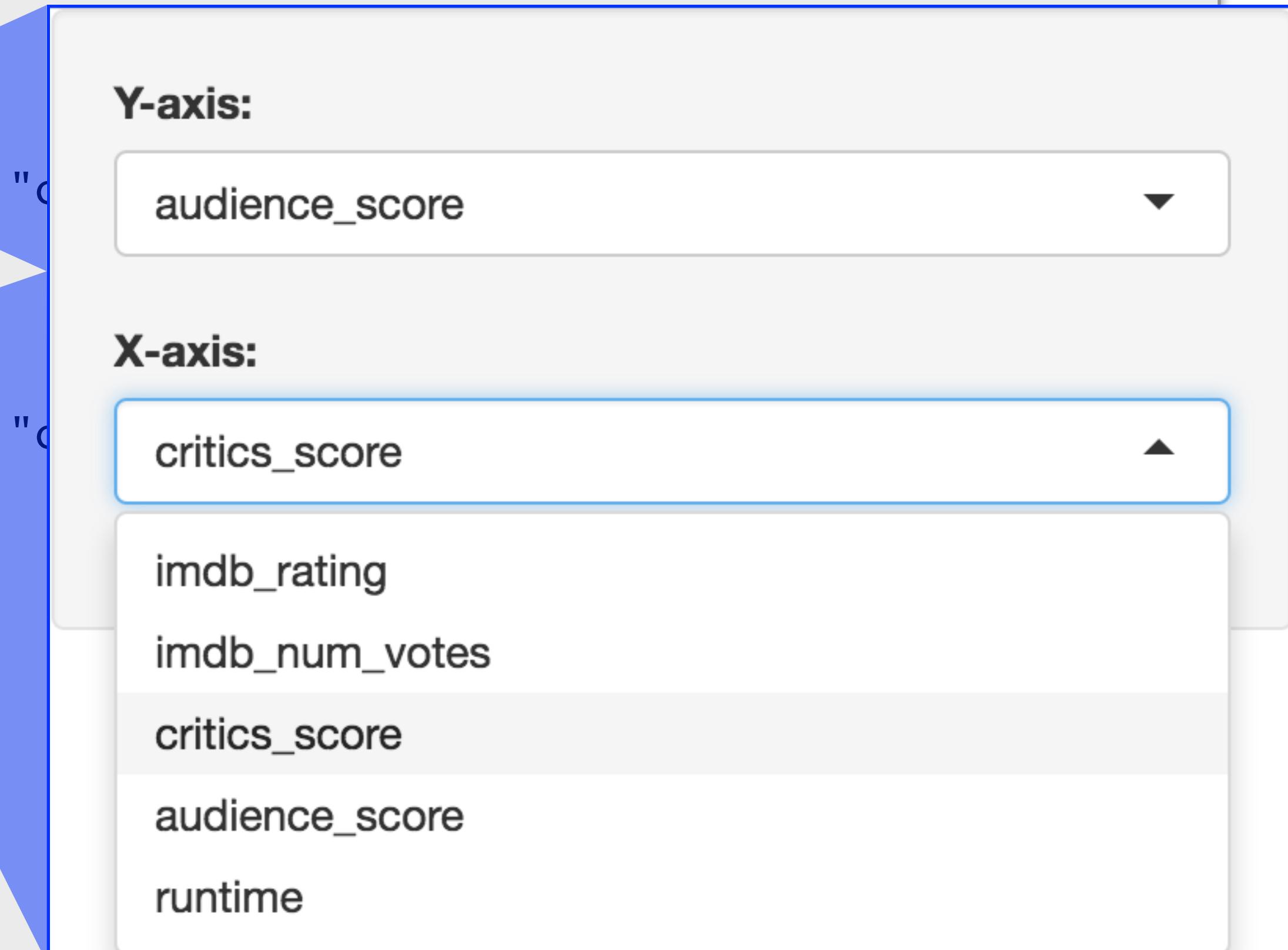
```
      plotOutput(outputId = "scatterplot")
```

```
    )
```

```
  )
```



```
# 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 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 s  
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
  output$scatterplot <- renderPlot({
    ggplot(data = movies, aes_string(x = input$x,
      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

5m 00s

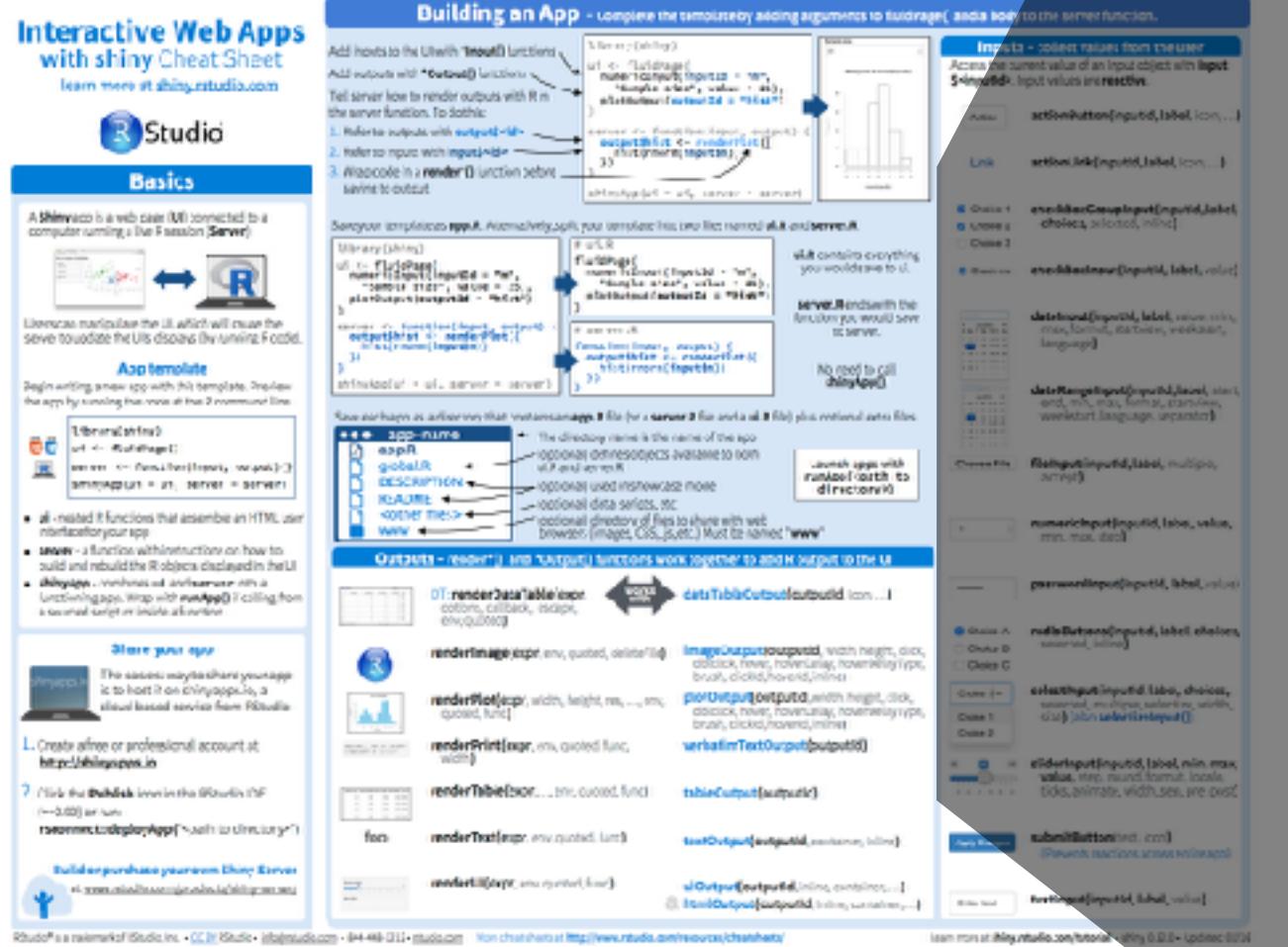


# SOLUTION

Solution to the previous exercise

`movies_02.R`

# INPUTS



|        |   |
|--------|---|
| Action | <b>actionButton</b> (inputId, label, icon, ...)   |
| Link   | <b>actionLink</b> (inputId, label, icon, ...)   |
|        | <b>checkboxGroupInput</b> (inputId, label, choices, selected, inline)   |
|        | <b>checkboxInput</b> (inputId, label, value)  |
|        | <b>dateInput</b> (inputId, label, value, min, max, format, startview, weekstart, language)                      |
|        | <b>dateRangeInput</b> (inputId, label, start, end, min, max, format, startview, weekstart, language, separator) |
|        | <b>fileInput</b> (inputId, label, multiple, accept)   |

|  |
|--|
| <b>numericInput</b> (inputId, label, value, min, max, step)  |
| <b>passwordInput</b> (inputId, label, value)   |
| <b>radioButtons</b> (inputId, label, choices, selected, inline)  |
| <b>selectInput</b> (inputId, label, choices, selected, multiple, selectize, width, size) (also <b>selectizeInput()</b> ) |
| <b>sliderInput</b> (inputId, label, min, max, value, step, round, format, locale, ticks, animate, width, sep, pre, post) |
| <b>submitButton</b> (text, icon)<br>(Prevents reactions across entire app)   |
| <b>textInput</b> (inputId, label, value)   |



# 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/](http://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

5m 00s

A large, light gray checkmark icon inside a circle, positioned in the top-left corner.

# SOLUTION

Solution to the previous exercise

`movies_03.R`

# OUTPUTS

The Shiny Cheat Sheet is a valuable resource for anyone working with Shiny. It provides a quick reference for all the main output types and their parameters. The 'works with' diagram shows how each output type interacts with its corresponding rendering function.

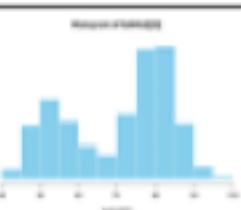
`DT::renderDataTable(expr,  
options, callback, escape,  
env, quoted)`

`dataTableOutput(outputId, icon, ...)`

`renderImage(expr, env, quoted, deleteFile)`



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



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

No need to call `shinyApp()`

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

foo

`renderText(expr, env, quoted, func)`

Choose a number  
Write a title  
Histogram of Random Normal Values

`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, ...)`



# 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/](http://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

5m 00s



# 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

5m 00s



# SOLUTION

Solution to the previous exercise

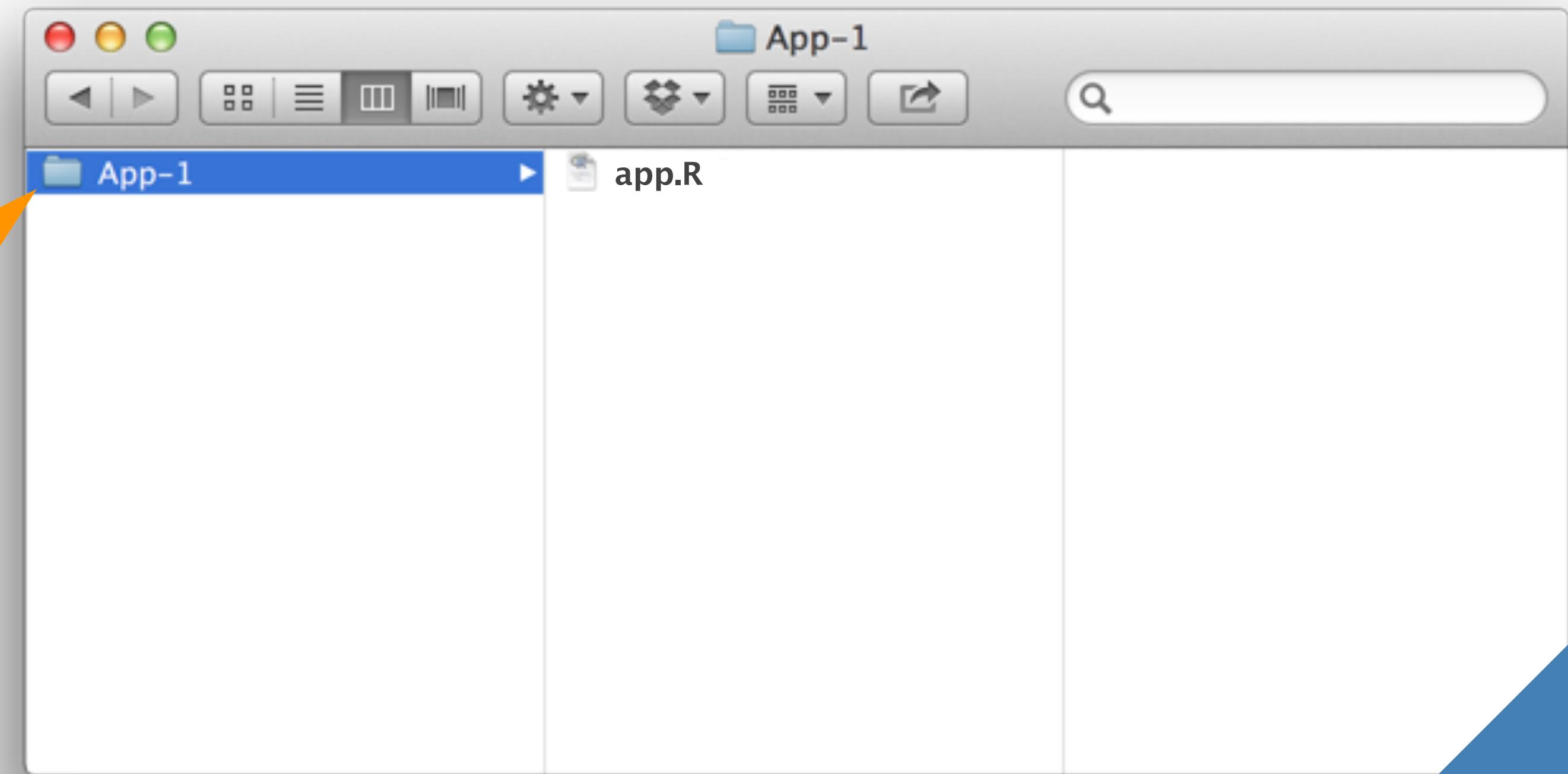
`movies_05.R`

# 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.



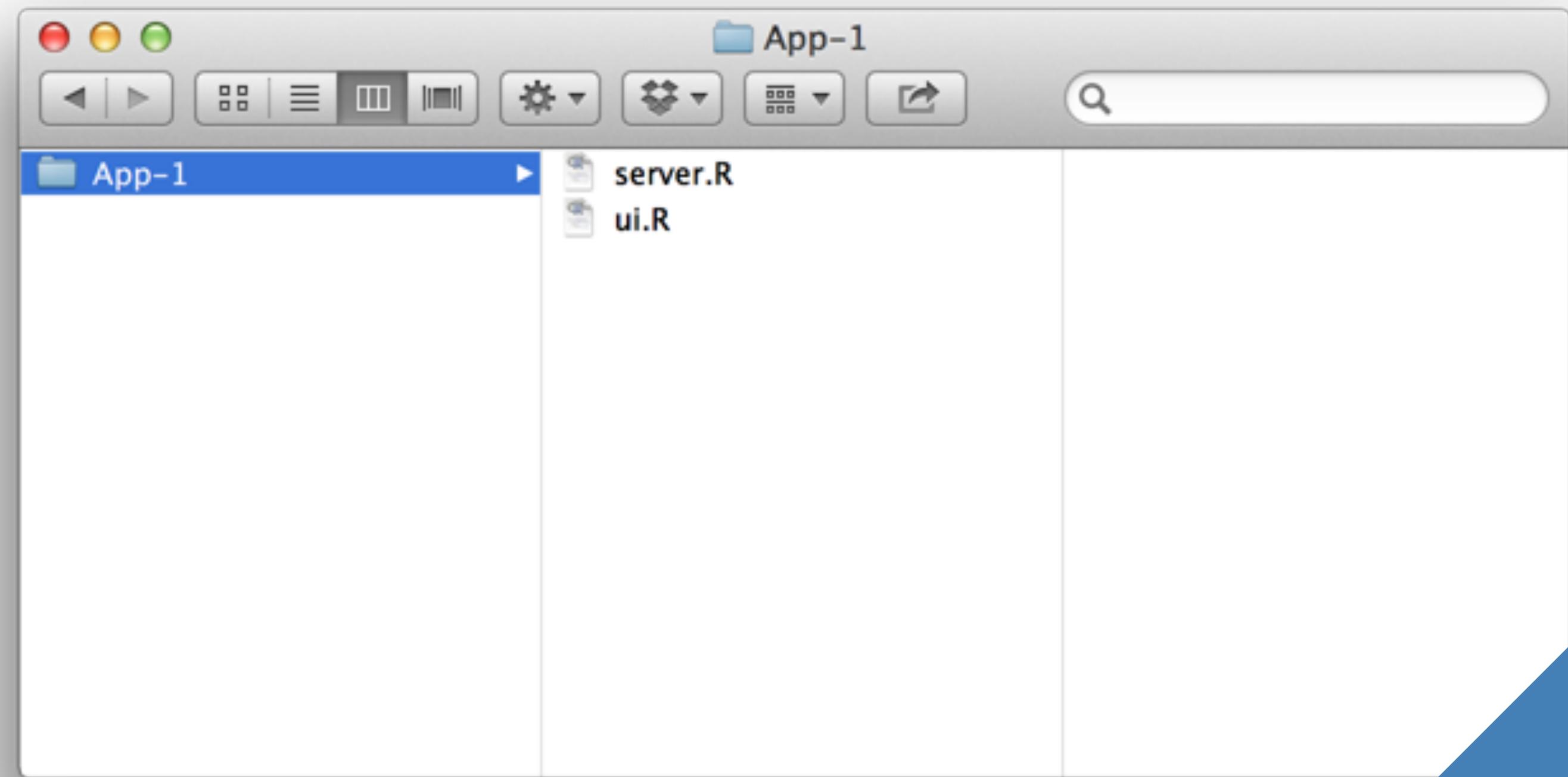
We will focus  
on the single  
file format  
throughout  
the workshop

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

# 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

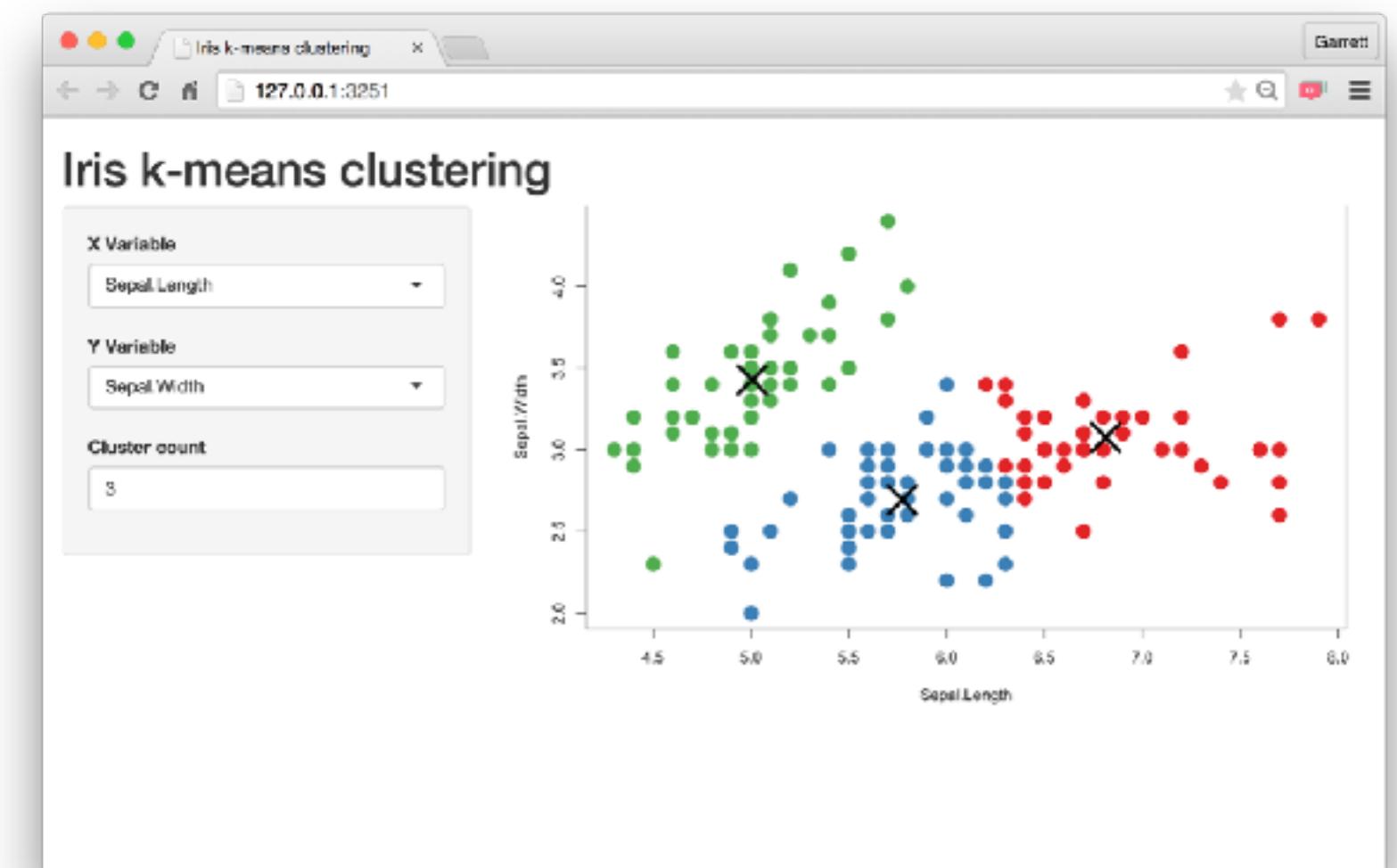
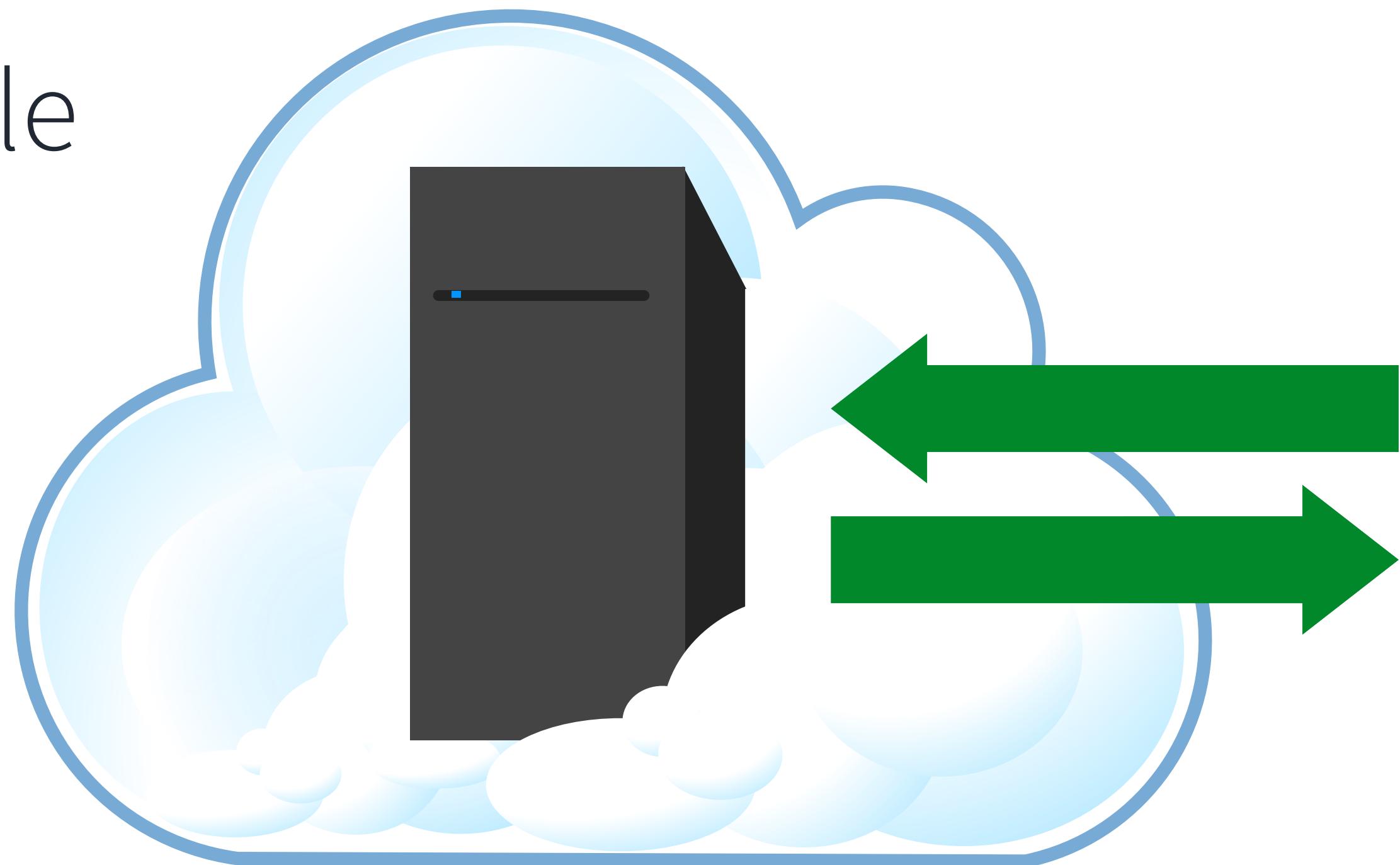
Sharing  
your app

# shinyapps.io



## A server maintained by RStudio

- ▶ easy to use
- ▶ secure
- ▶ scalable



# HASSLE-FREE CLOUD HOSTING FOR SHINY

shinyapps.io by RStudio

Home Features Pricing Support

Log In

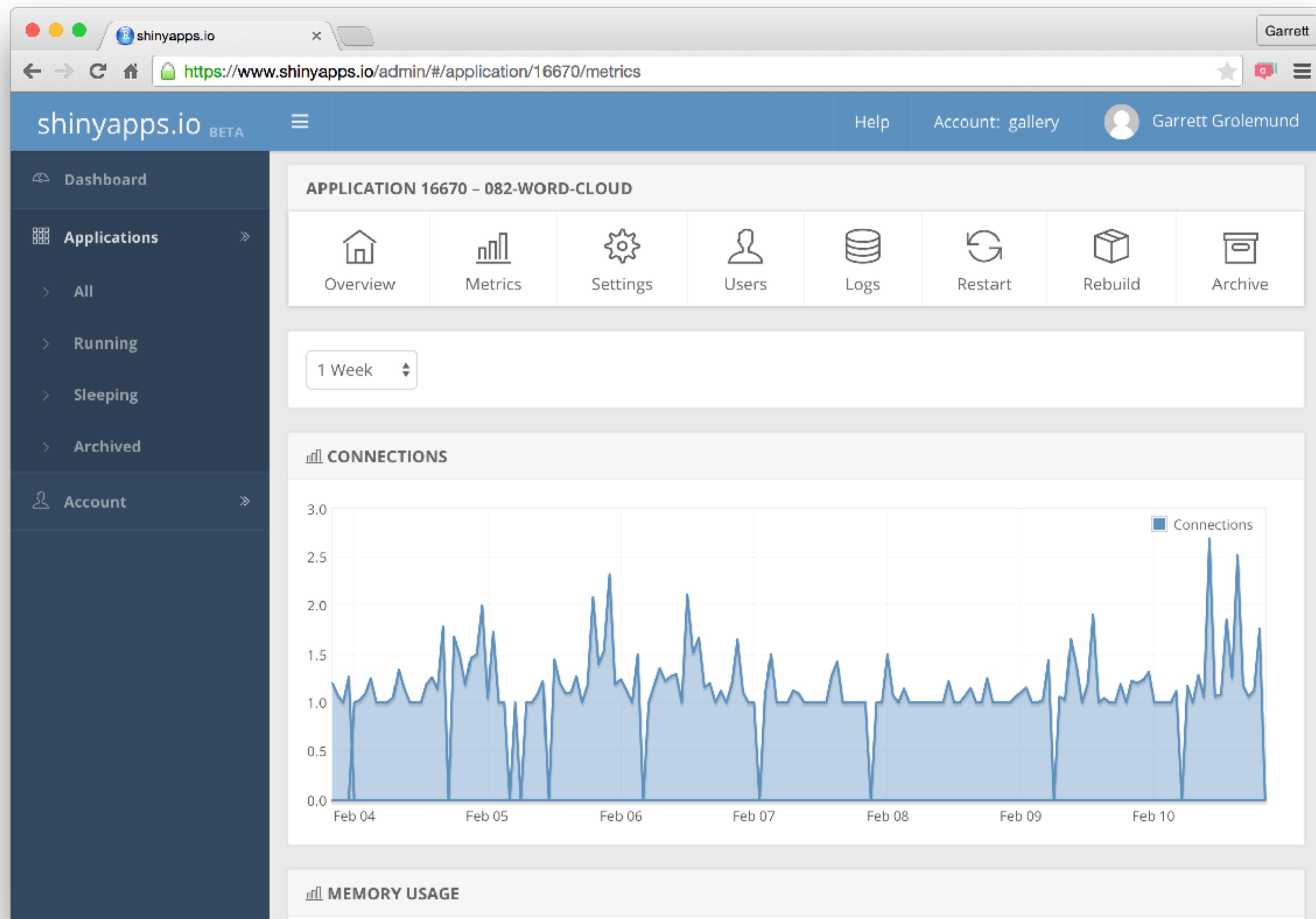
## Share your Shiny Applications Online

Deploy your Shiny applications on the Web in minutes

Sign Up



# WITH BUILT-IN METRICS



# MEMBERSHIP PRICING

FREE

**\$ 0** /month

New to Shiny? Deploy your applications for FREE.

5 Applications

25 Active Hours

Community Support

RStudio Branding

STARTER

**\$ 9** /month

( or \$100/year )

More applications. More active hours!

25 Applications

100 Active Hours

Premium Support

BASIC

**\$ 39** /month

( or \$440/year )

Take your users to the next level!

Unlimited Applications

500 Active Hours

Performance Boost

Premium Support

STANDARD

**\$ 99** /month

( or \$1,100/year )

Password protection? Authenticate your users!

Unlimited Applications

2,000 Active Hours

Authentication

Performance Boost

Premium Support

PROFESSIONAL

**\$ 299** /month

( or \$3,300/year )

Professional has it all! Personalize your domains.

Unlimited Applications

10,000 Active Hours

Authentication

Account Sharing

Performance Boost

Custom Domains

Premium Support

**Build your own  
server**



# SHINY SERVER

[rstudio.com/products/shiny/shiny-server/](http://rstudio.com/products/shiny/shiny-server/)



- ✓ 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



# SHINY SERVER PRO

[rstudio.com/products/shiny/shiny-server/](http://rstudio.com/products/shiny/shiny-server/)



## ✓ **Secure access**

LDAP, GoogleAuth, SSL, and more

## ✓ **Performance**

fine tune at app and server level

## ✓ **Management**

monitor and control resource use

## ✓ **Support**

direct priority support

45 day  
evaluation  
free trial

# RSTUDIO CONNECT

[rstudio.com/products/connect/](http://rstudio.com/products/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



# A FAST INTRODUCTION TO SHINY

The screenshot shows the RStudio interface with several panes:

- Code Editor:** Shows a script named `print.R` with code related to ggvis and Vega. Line 29 highlights `x$data[data_ids]`. Lines 30-41 show comments about collapsing scales and wrapping reactive data objects.
- Console:** Shows the command `> ggvis(diamonds, x = ~price, y = ~color)` being run. The output includes messages about guessing histograms and binwidth, and a stack trace indicating the code was debugged at `/Users/jmcphers/r/ggvis/R/vega.R#29`.
- Environment:** Shows the environment for `as.vega.ggviz()` with variables `data_ids`, `data_props`, and `dynamic`.
- Traceback:** Shows the call stack for `as.vega.ggviz(x, FALSE)`.
- Plots:** A histogram of diamond prices. The x-axis is labeled "price" and ranges from 0 to 12,000. The y-axis is labeled "count" and ranges from 0 to 120. The distribution is highly right-skewed, with the highest frequency occurring near \$0.

Shiny from R Studio