

# A FAST INTRODUCTION TO SHINY

The screenshot shows the RStudio interface with several panes:

- Code Editor:** Shows R code in a script named `print.R`. The code is part of a function definition, specifically handling the `data_table` object.
- Console:** Shows the execution of `ggvis(diamonds, x = ~price, y = ~color)`. The output includes messages about guessing histograms and binwidth, and a stack trace indicating the code was debugged at line 29 of `vega.R`.
- Environment:** Shows the environment for the `as.vega.ggviz()` function, listing `data_ids` as "diamonds0/bin1/stack2", `data_props` as a "List of 1", and `dynamic` as FALSE.
- Plots:** Displays 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

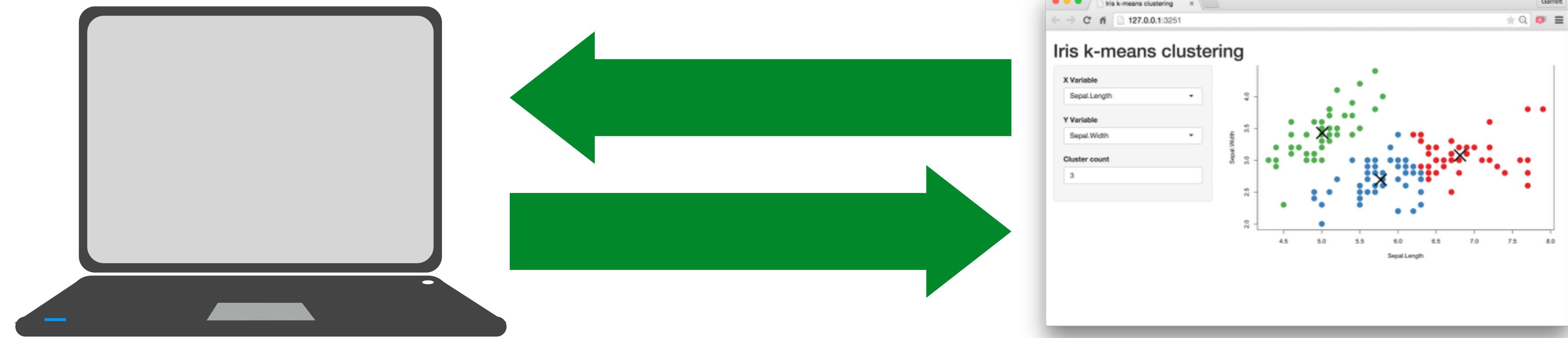
# OUTLINE

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

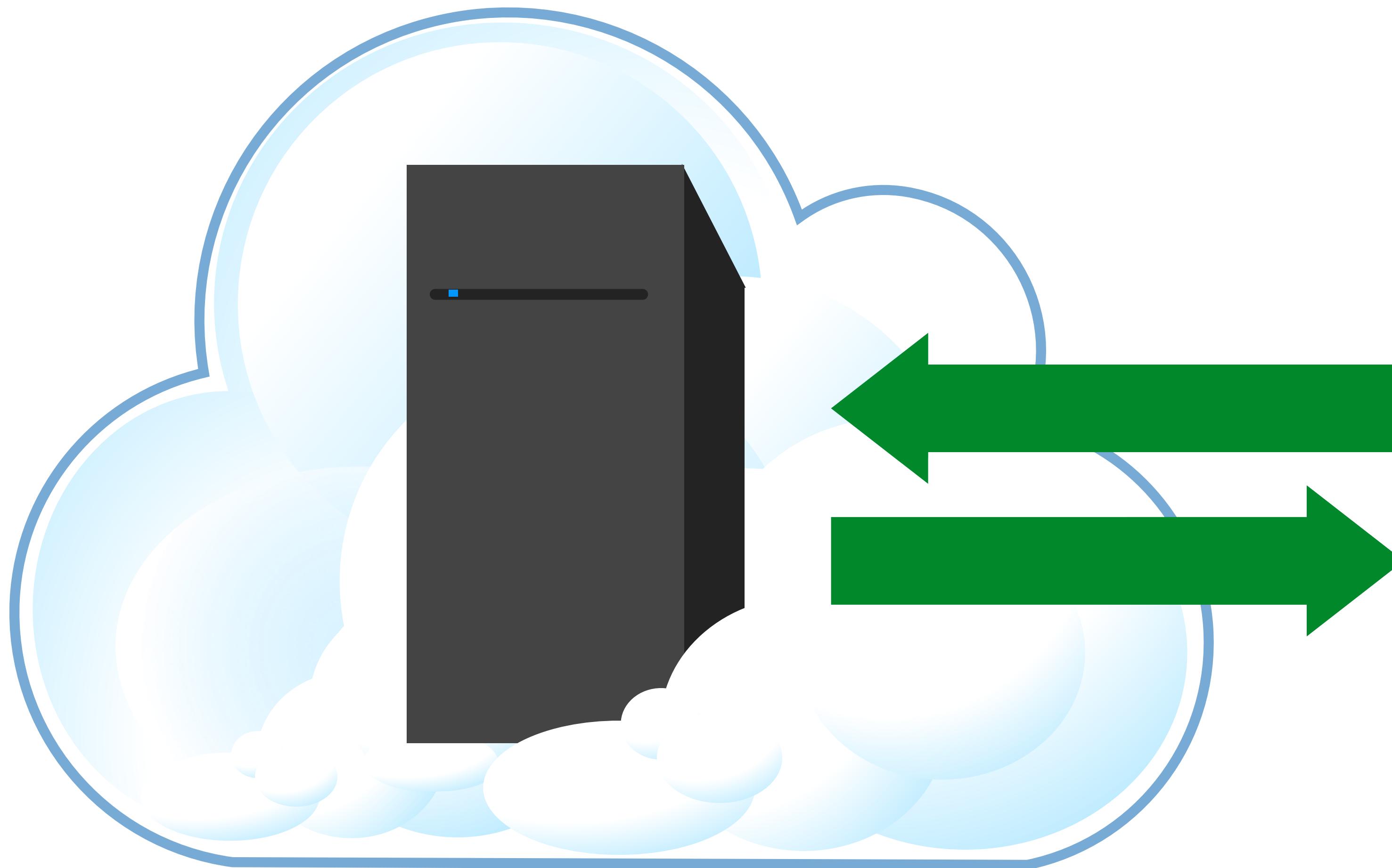
High level

view

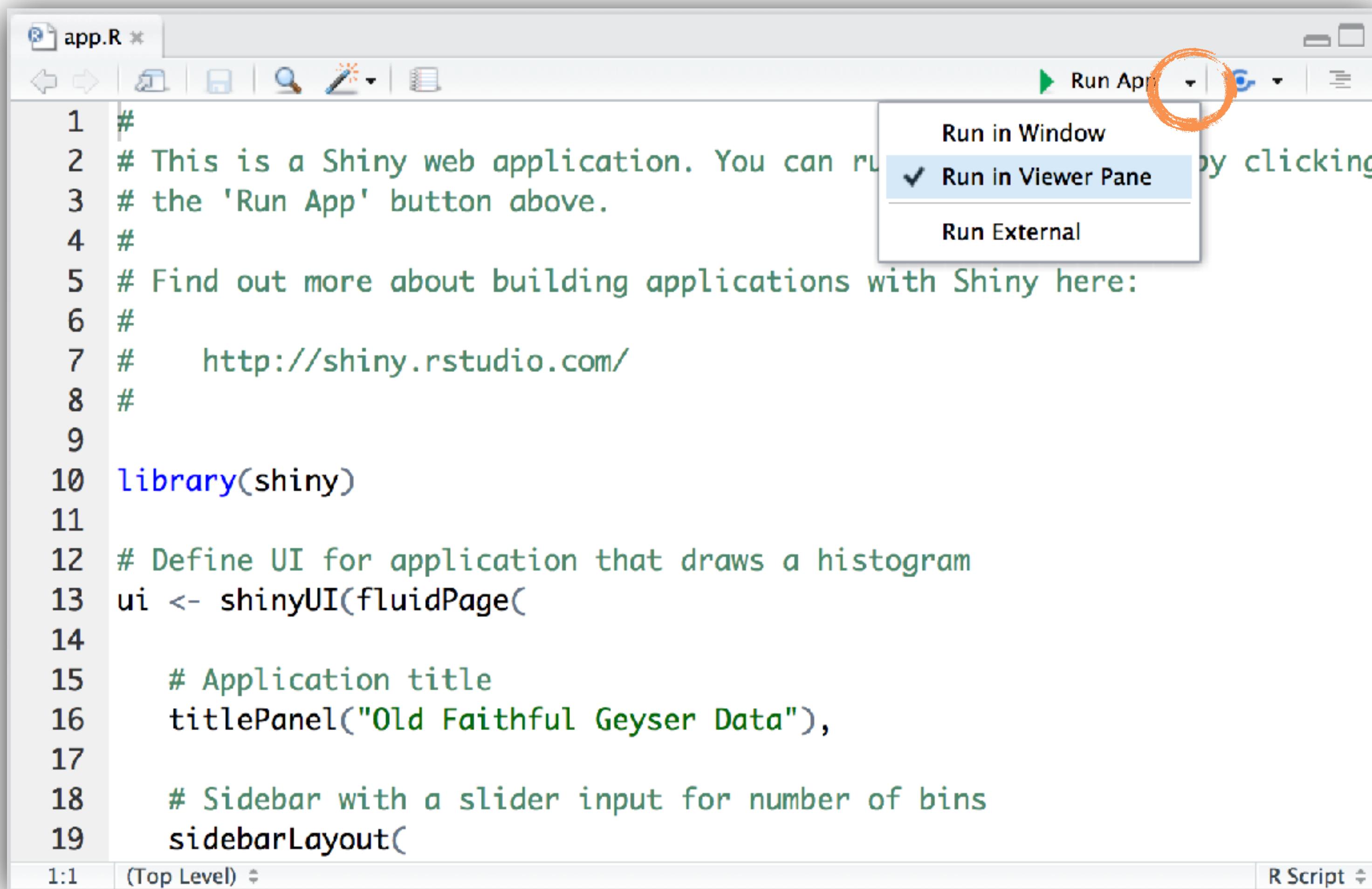
# Every Shiny app is maintained by a computer running R



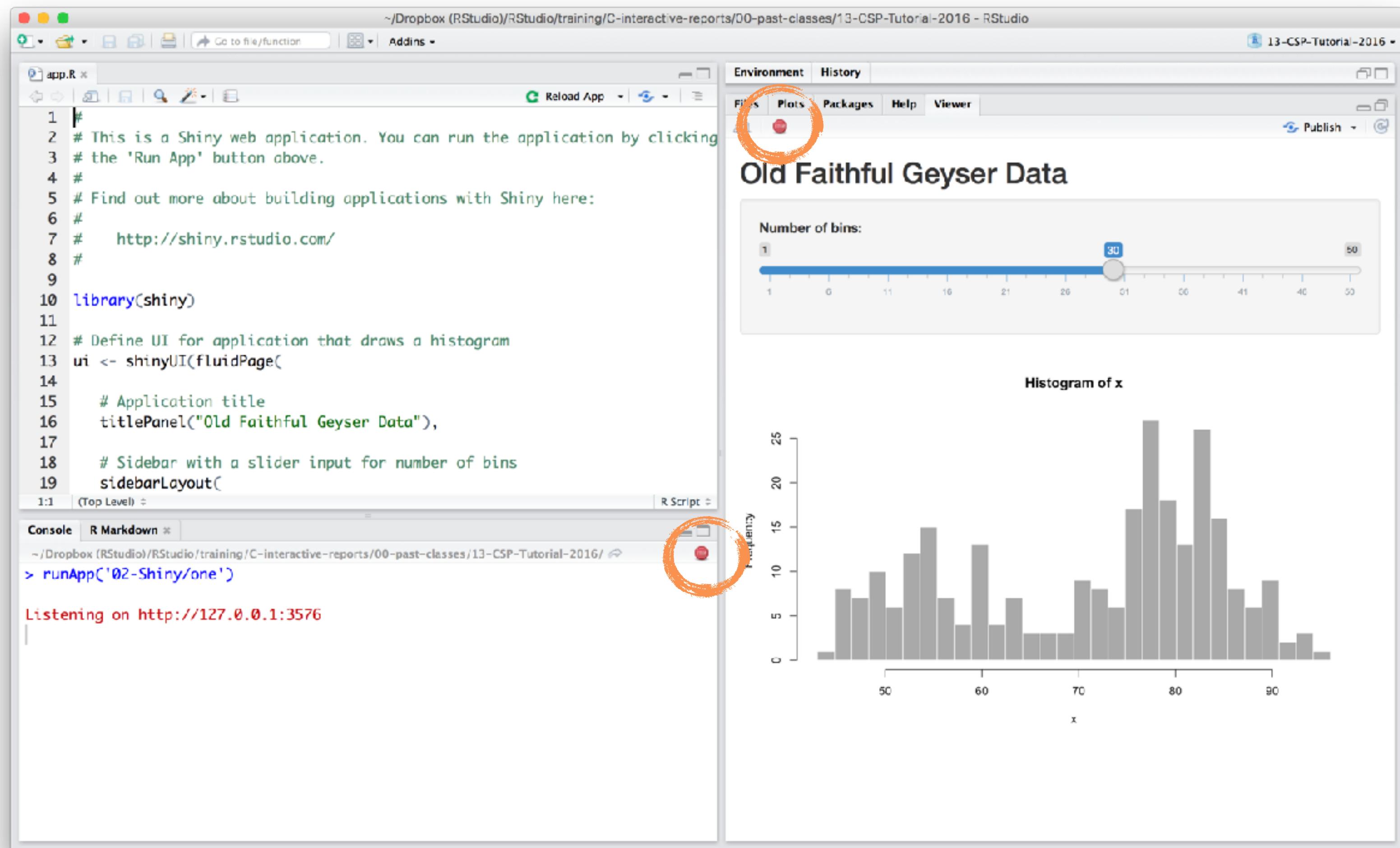
# Every Shiny app is maintained by a computer running R



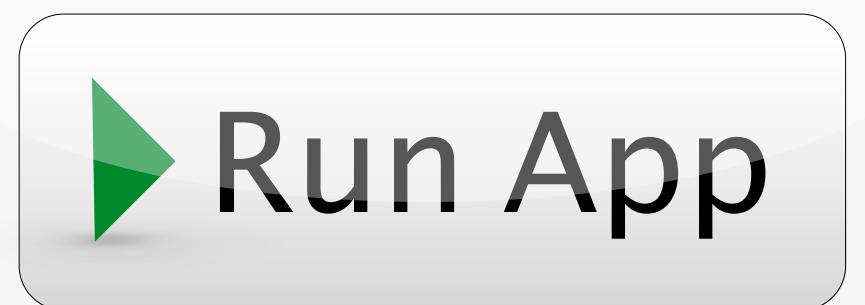
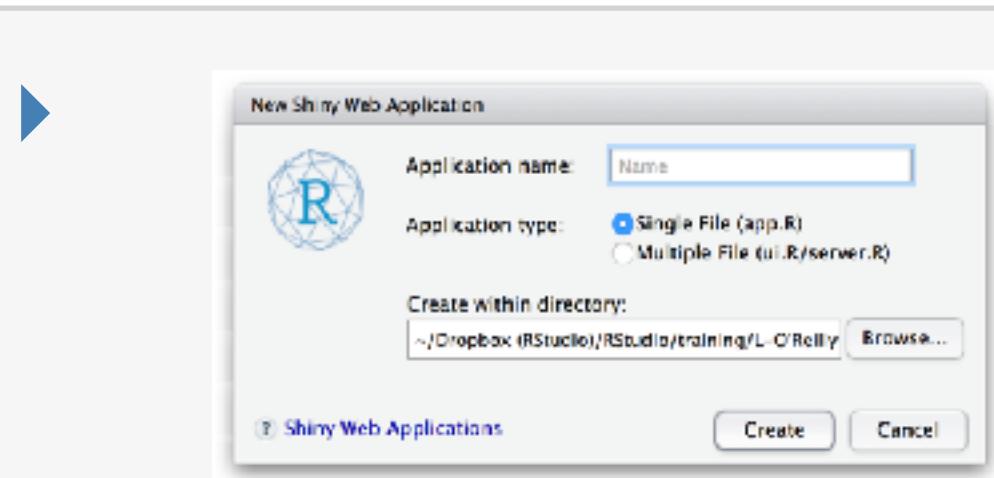
# Change display



# 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) {}
```

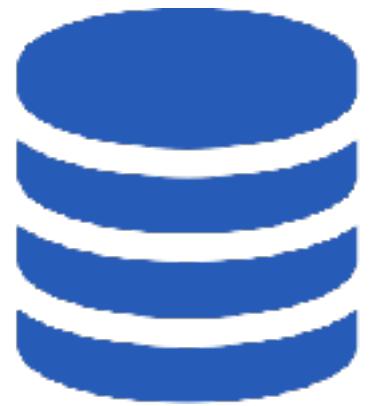
## Server function

contains instructions needed to build app

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

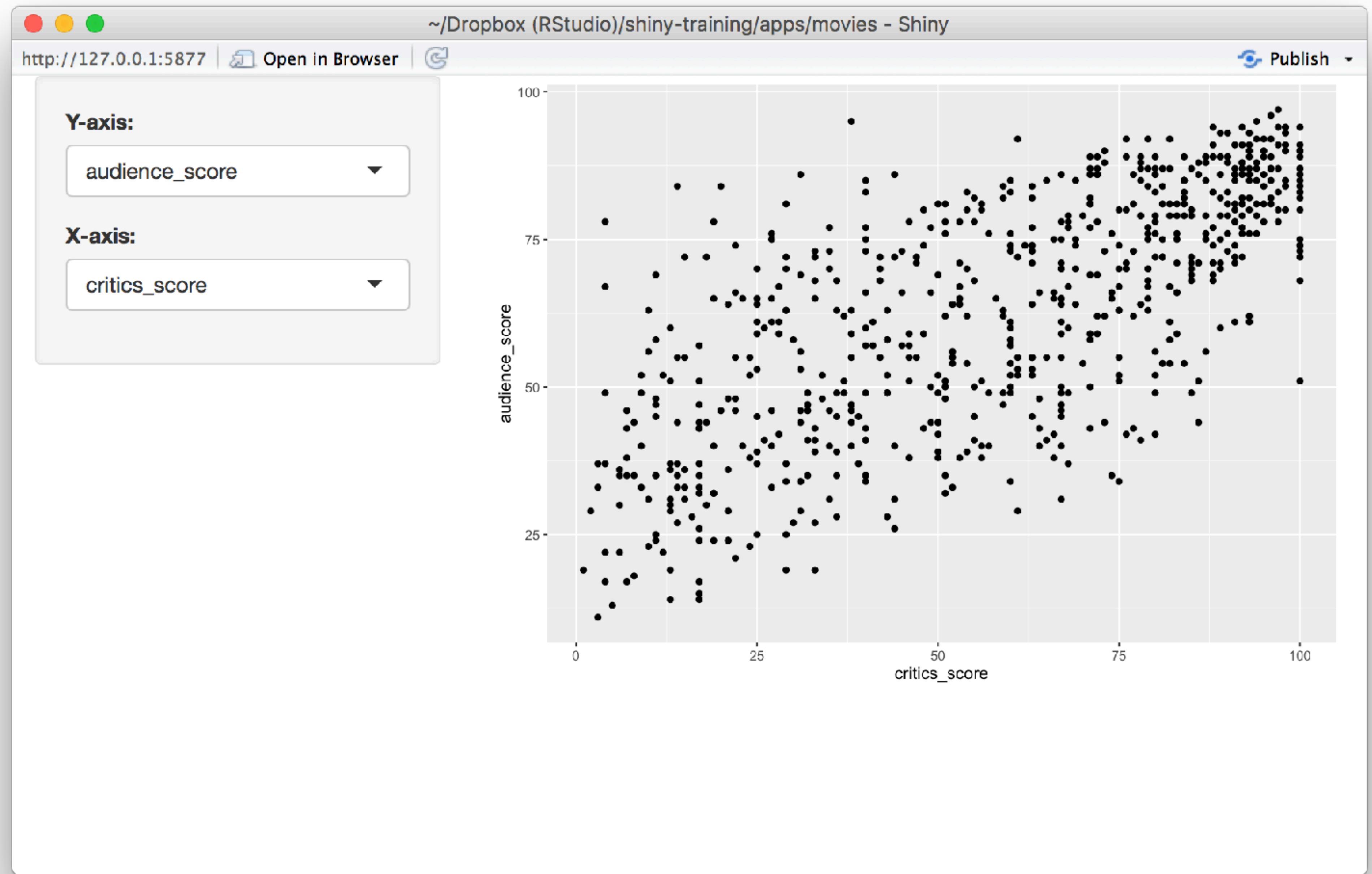


# Let's build a simple movie browser app!



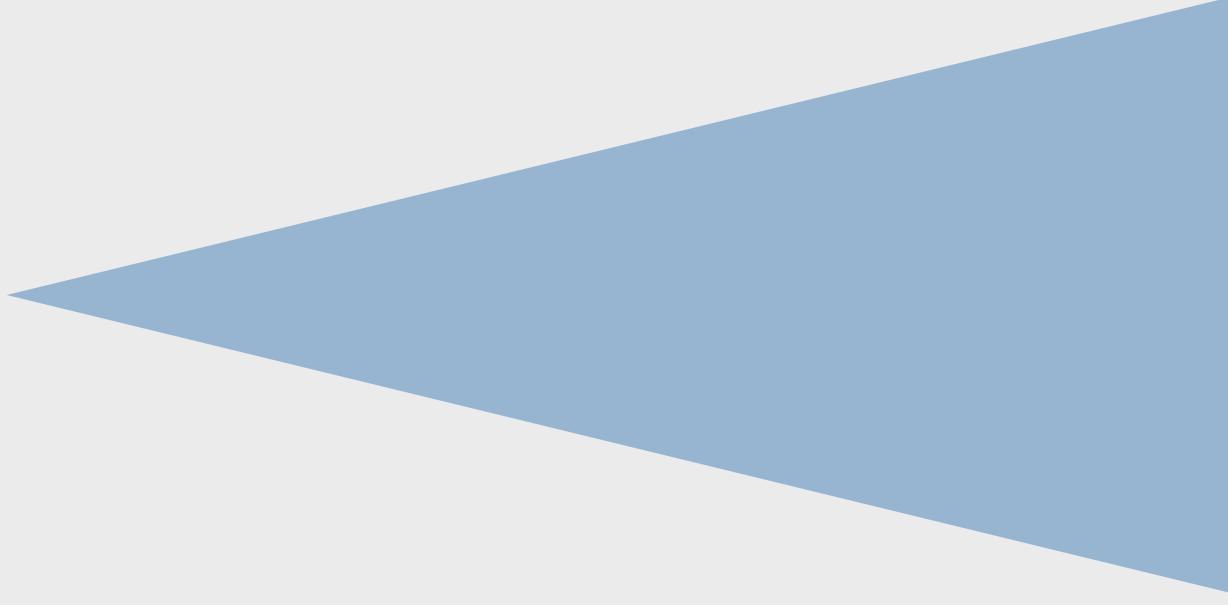
## movies.Rdata

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(
```

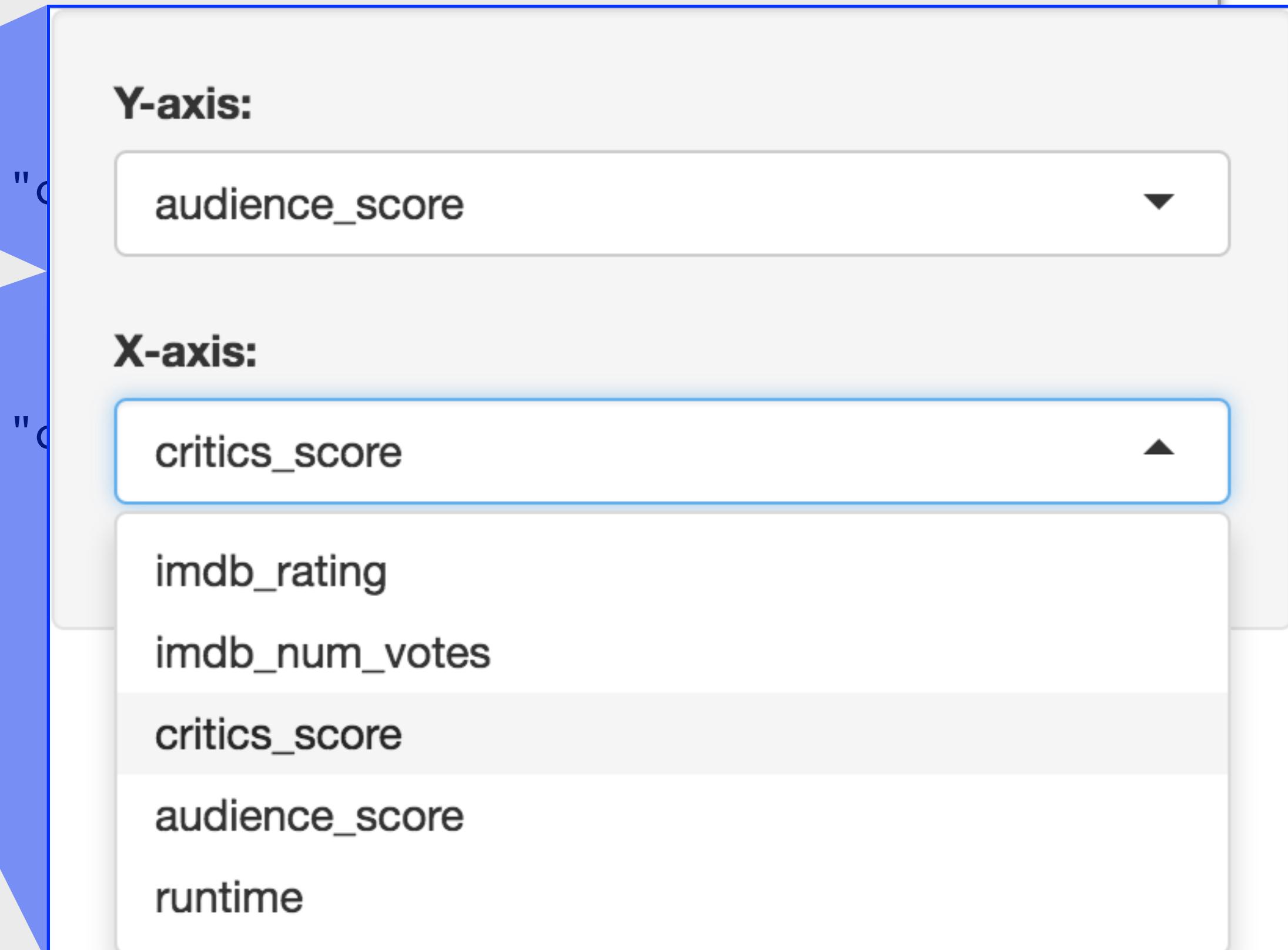
```
      plotOutput(outputId = "scatterplot")
```

```
    )
```

```
  )
```



```
# 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

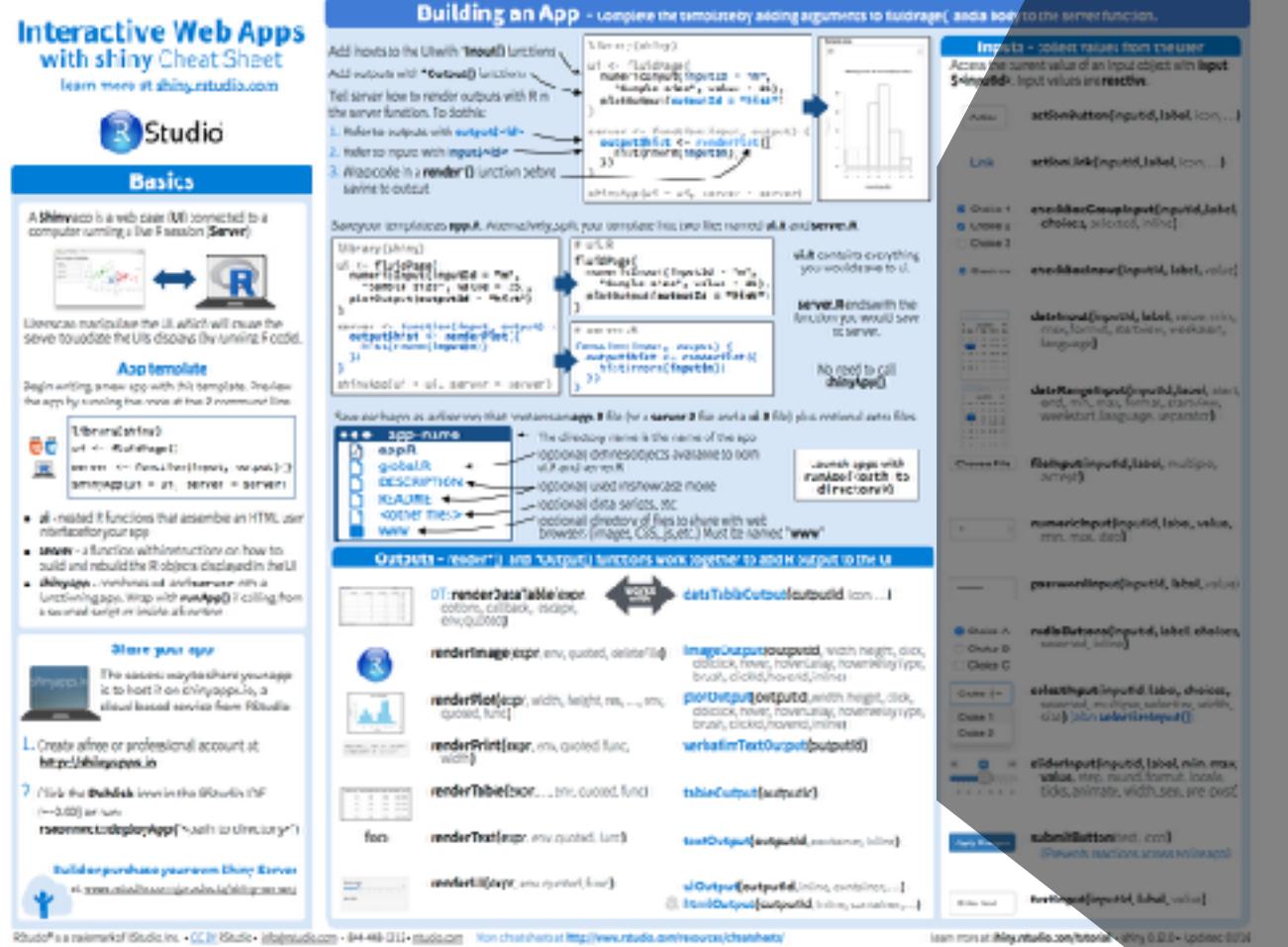


# SOLUTION

Solution to the previous exercise

`movies_02.R`

# INPUTS



**ActionButton(inputId, label, icon, ...)**

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

- Choice 1
- Choice 2
- Choice 3

**checkboxInput(inputId, label, value)**

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

**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)**

**fileInput(inputId, label, multiple, accept)**

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

.....

- Choice A
- Choice B
- Choice C

Choice 1  
Choice 1  
Choice 2

0 5 10  
0 2 4 6 8 10

Apply Changes

Enter text

**passwordInput(inputId, label, value)**

**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)**

**submitButton(text, icon)**

(Prevents reactions across entire app)

**textInput(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

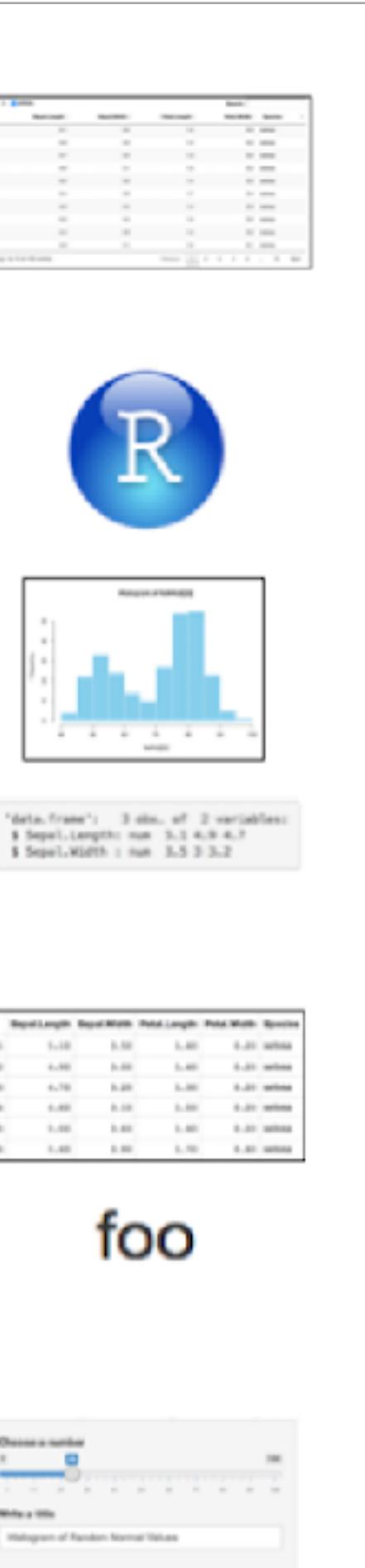
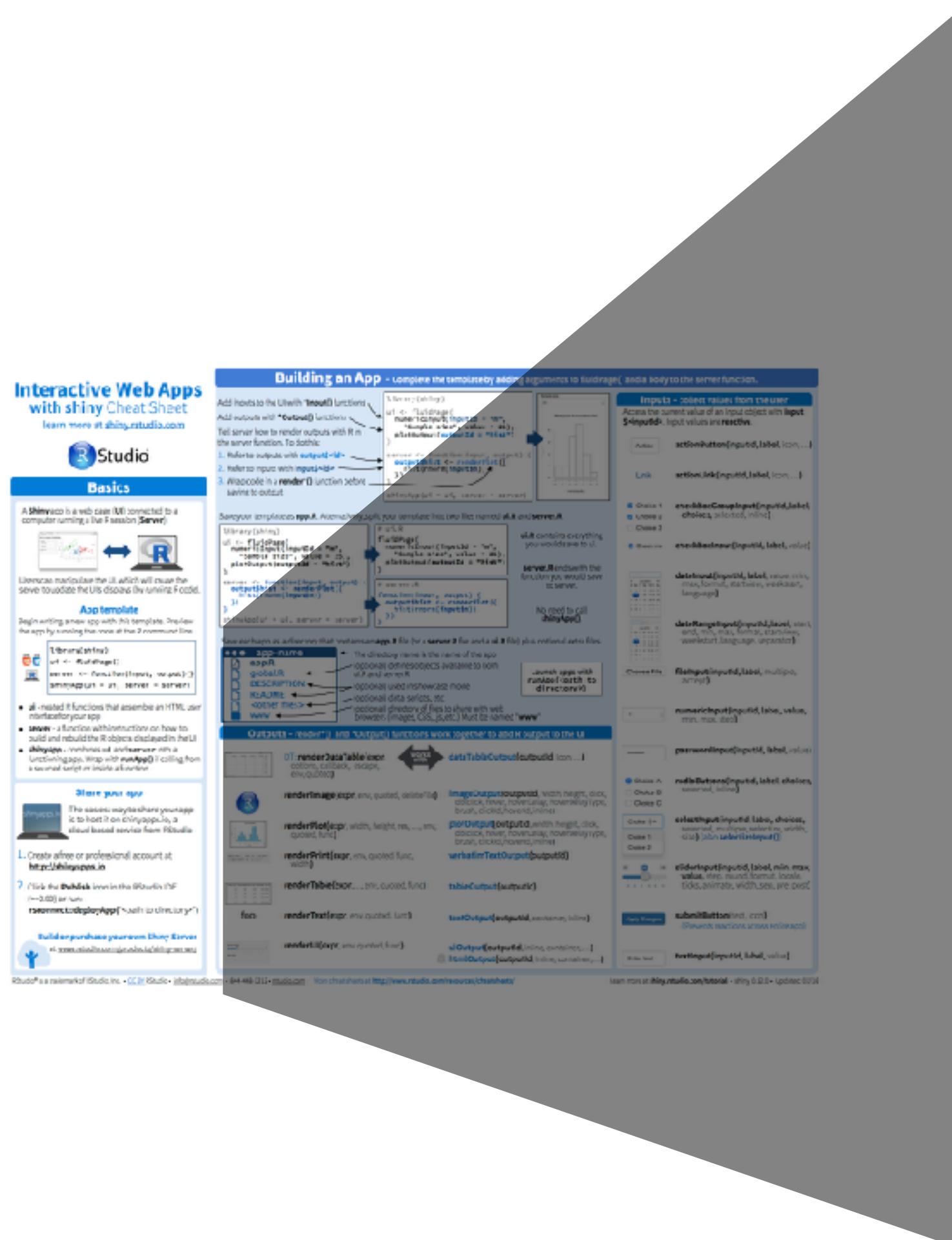


# SOLUTION

Solution to the previous exercise

`movies_03.R`

# OUTPUTS

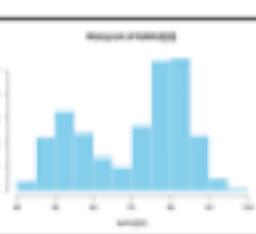


`DT::renderDataTable(expr, options, callback, escape, env, quoted)`



`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)`



foo

`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, ...)`



# 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
- ▶ **Optional:** If you finish early, move on to the next exercise

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

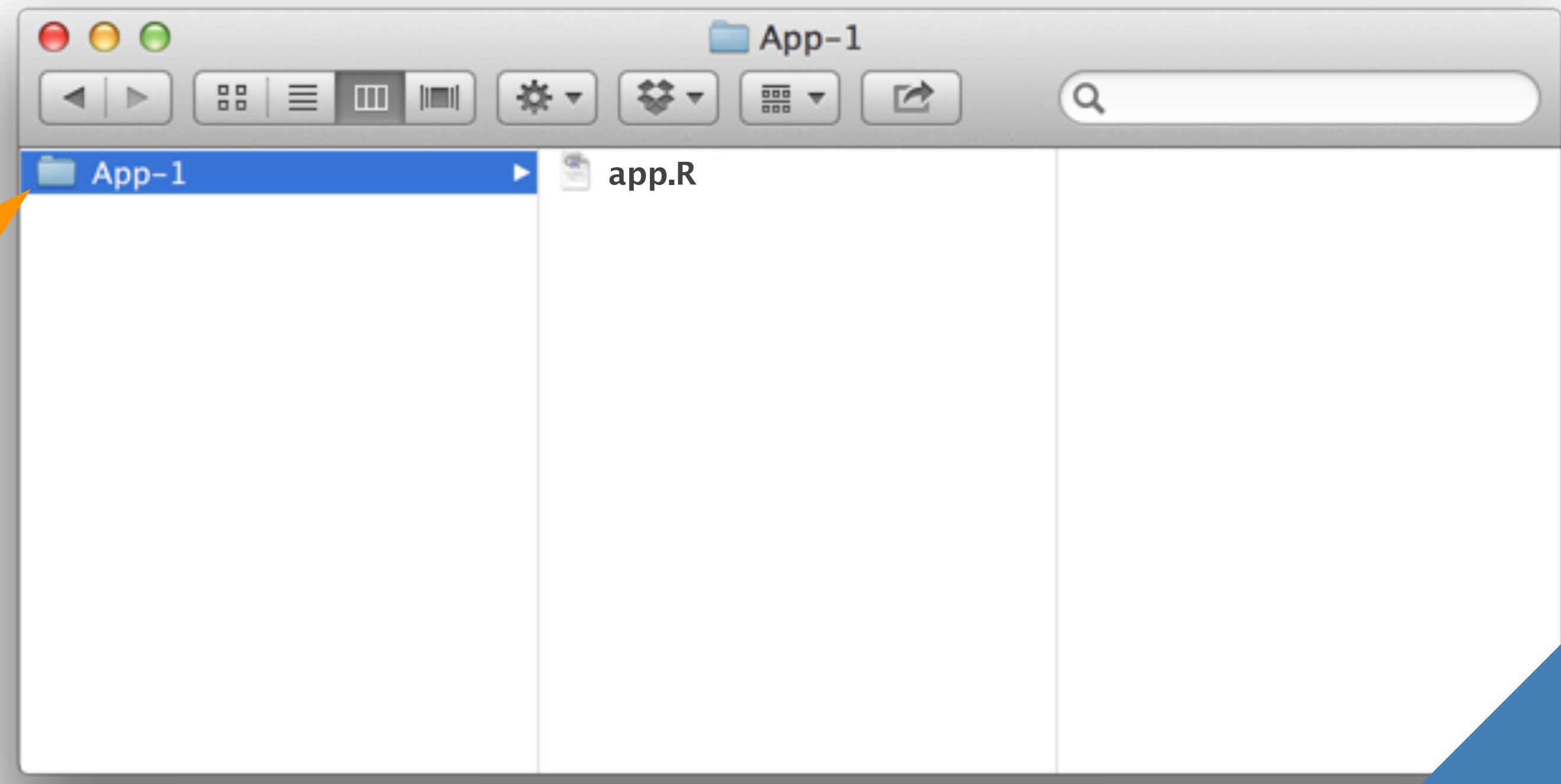
`movies_05.R`

# 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.



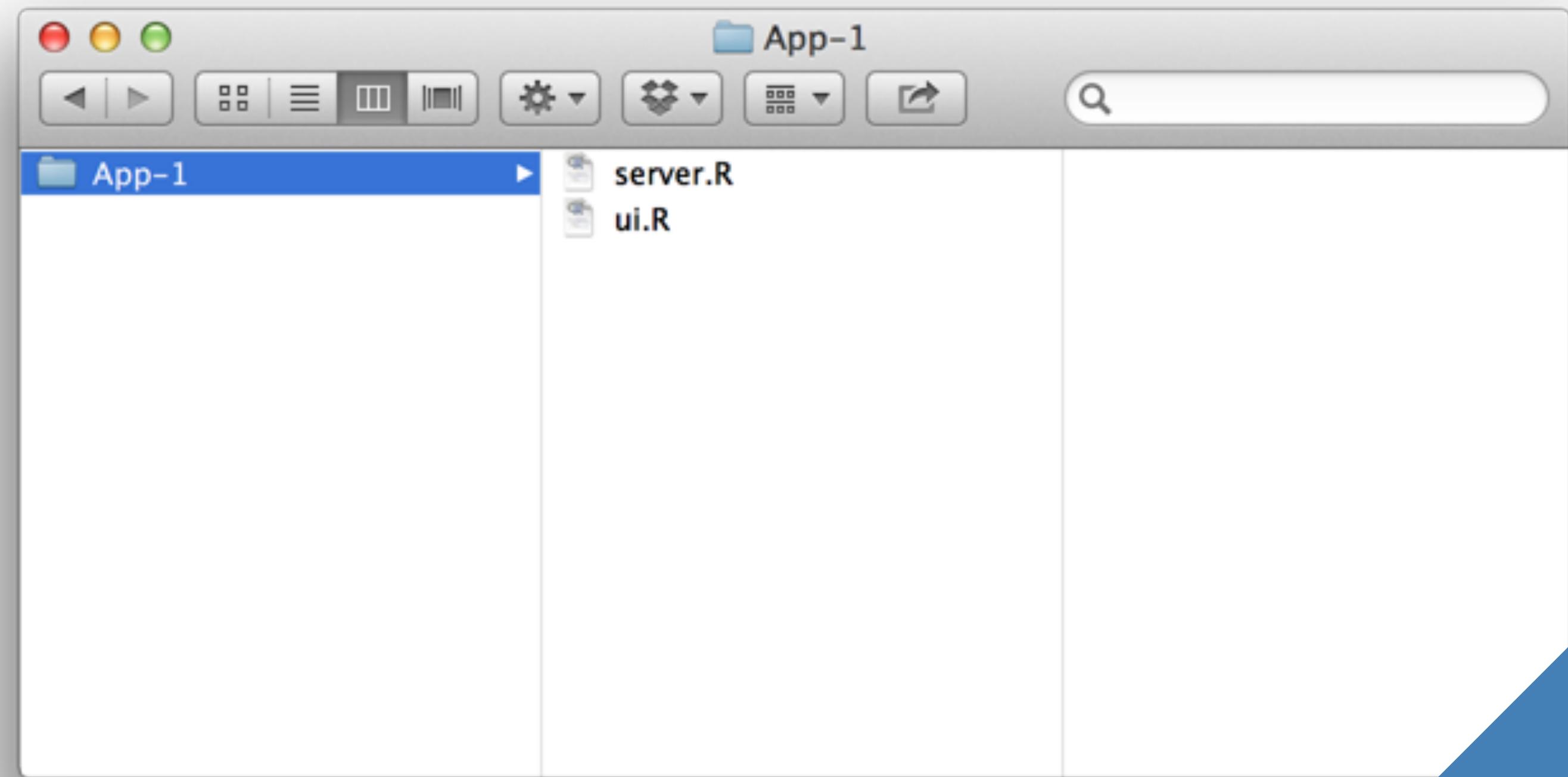
We will focus  
on the single  
file format  
throughout  
the workshop

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

# 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

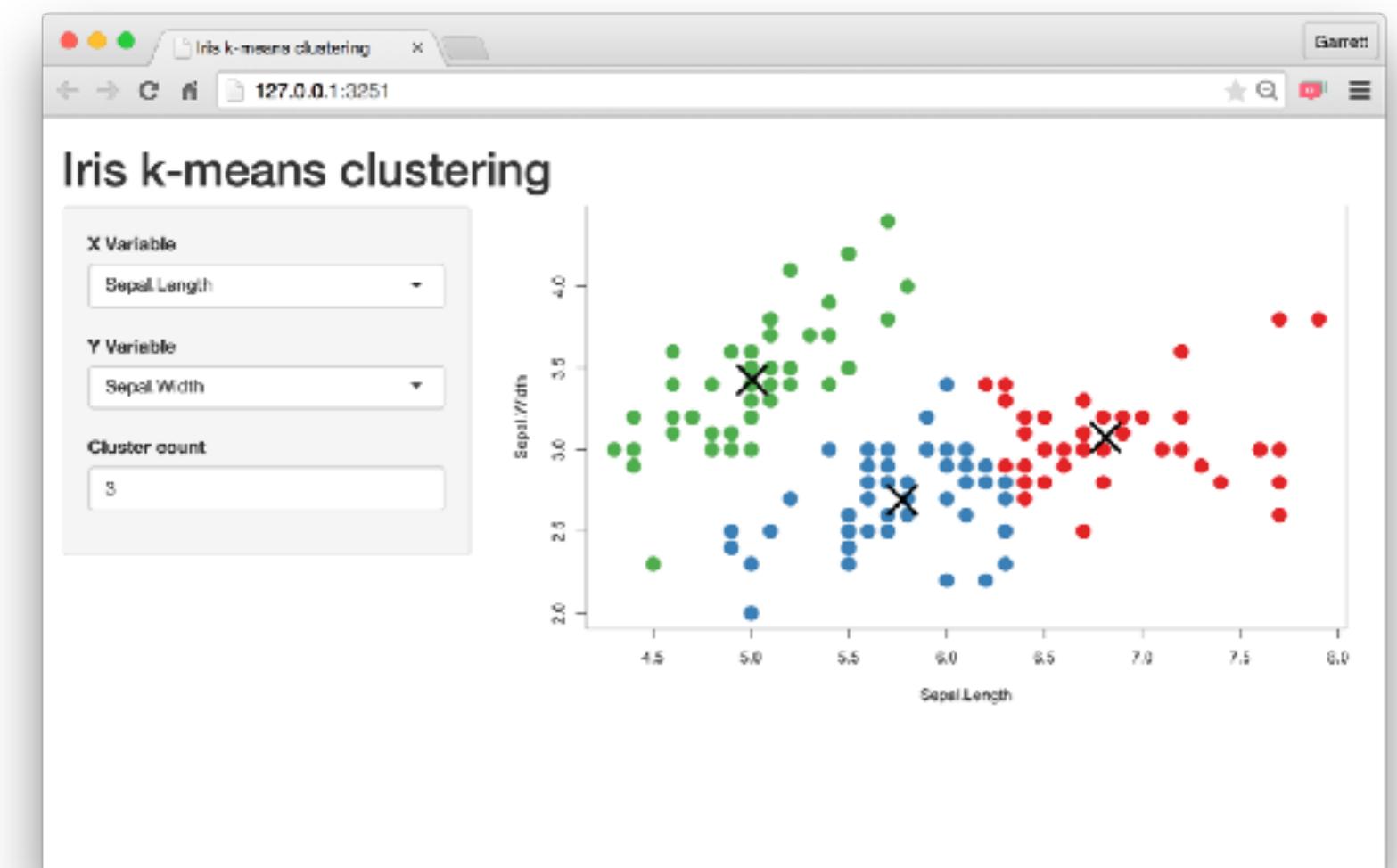
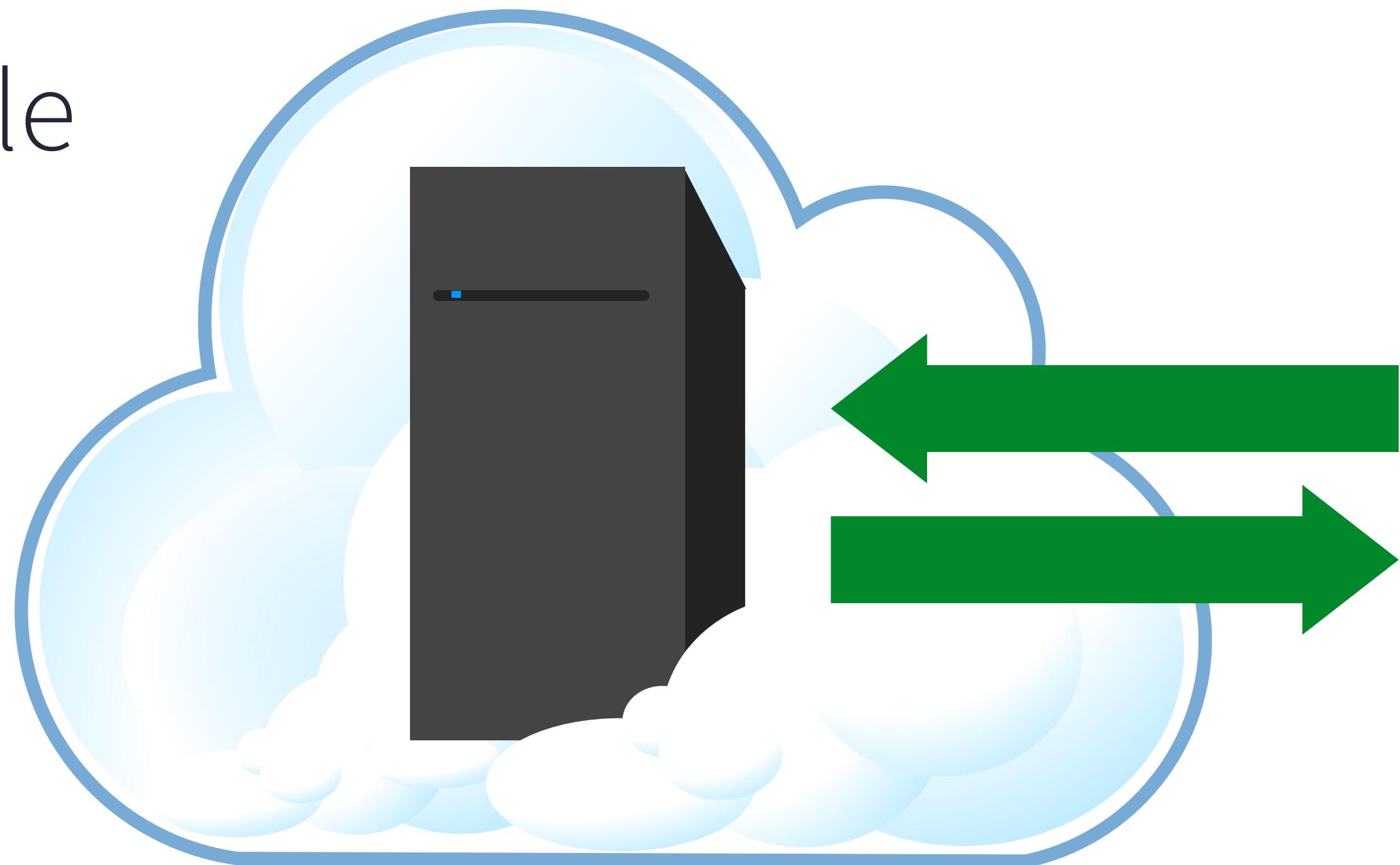
Sharing  
your app

# 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

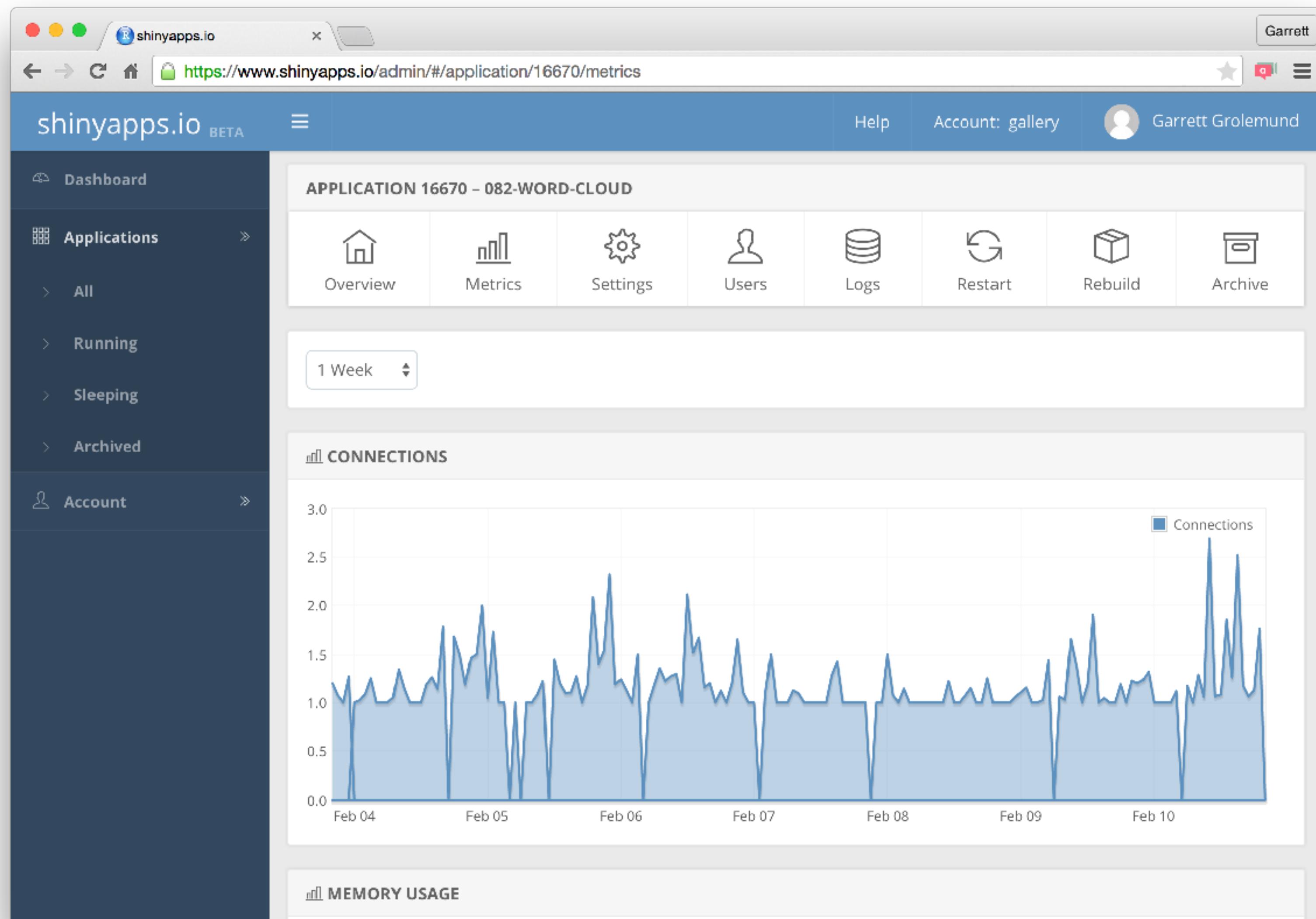
Share your Shiny  
Applications Online

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 R code in a script named `print.R`. The code is part of a function definition, specifically handling the `data_table` object.
- Console:** Shows the execution of `ggvis(diamonds, x = ~price, y = ~color)`. The output includes messages about guessing histograms and binwidth, and a stack trace indicating the code was debugged at line 29 of `vega.R`.
- Environment:** Shows the environment for the `as.vega.ggviz()` function, listing `data_ids` as "diamonds0/bin1/stack2", `data_props` as a "List of 1", and `dynamic` as FALSE.
- Plots:** Displays 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