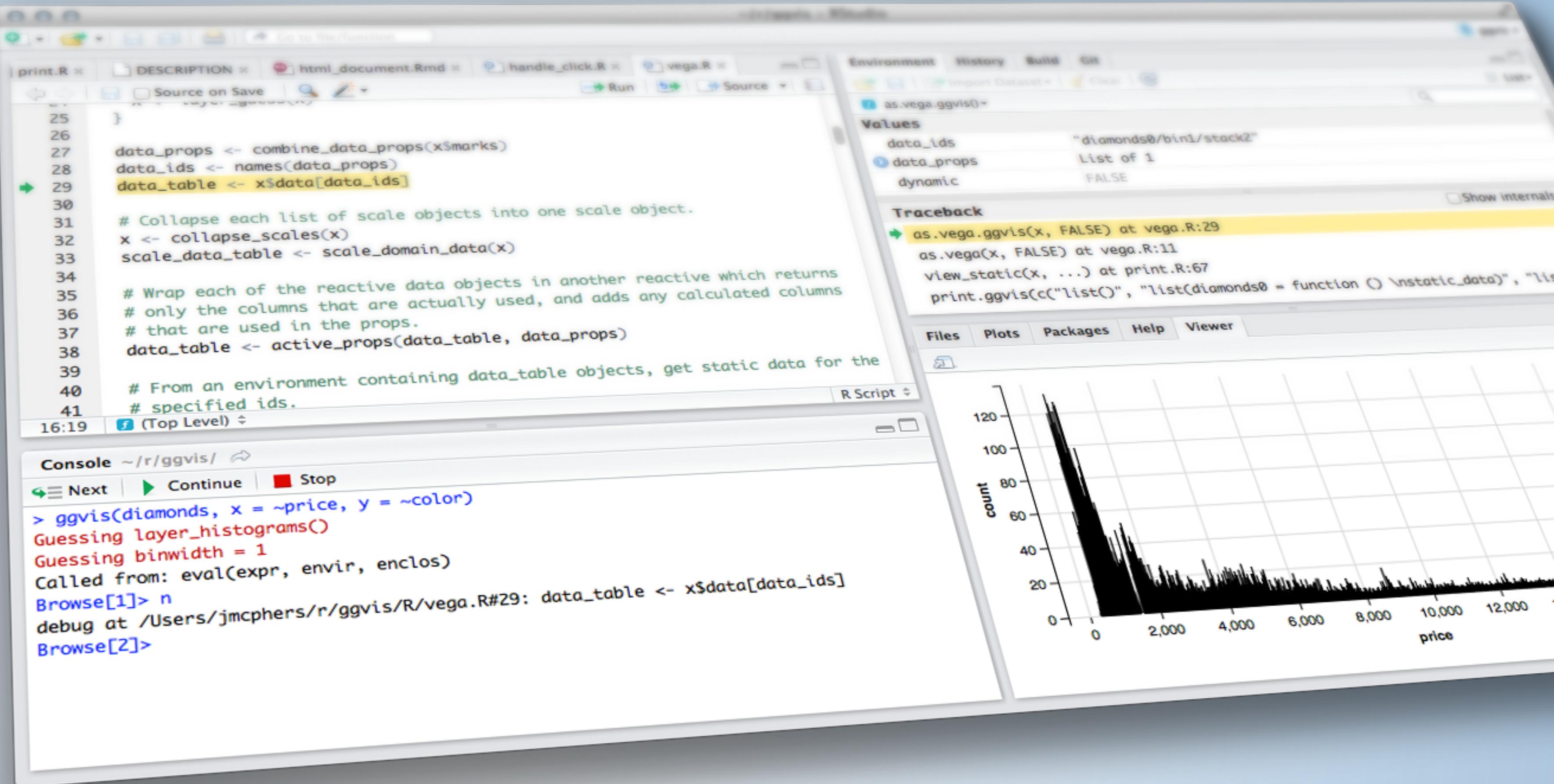


WELCOME TO SHINY



Shiny from  R Studio™

OUTLINE

- ▶ Anatomy of a Shiny app
 - ▶ User interface
 - ▶ Server function
 - ▶ Create the app
- ▶ Sharing your app
- ▶ Dashboards

All materials at
bit.ly/shiny-2017-08-10

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)
```

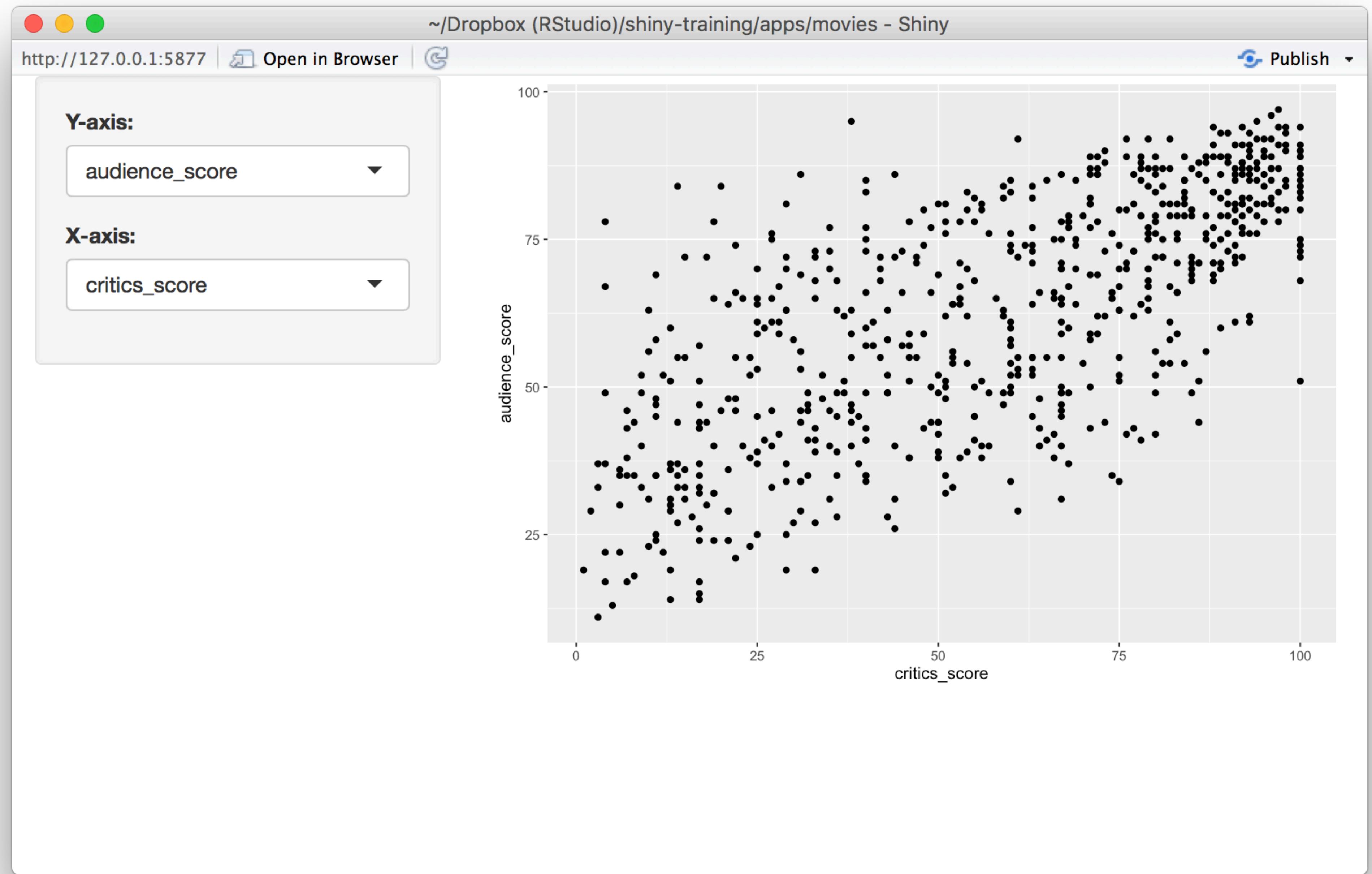


Goal: Build a simple movie browser app



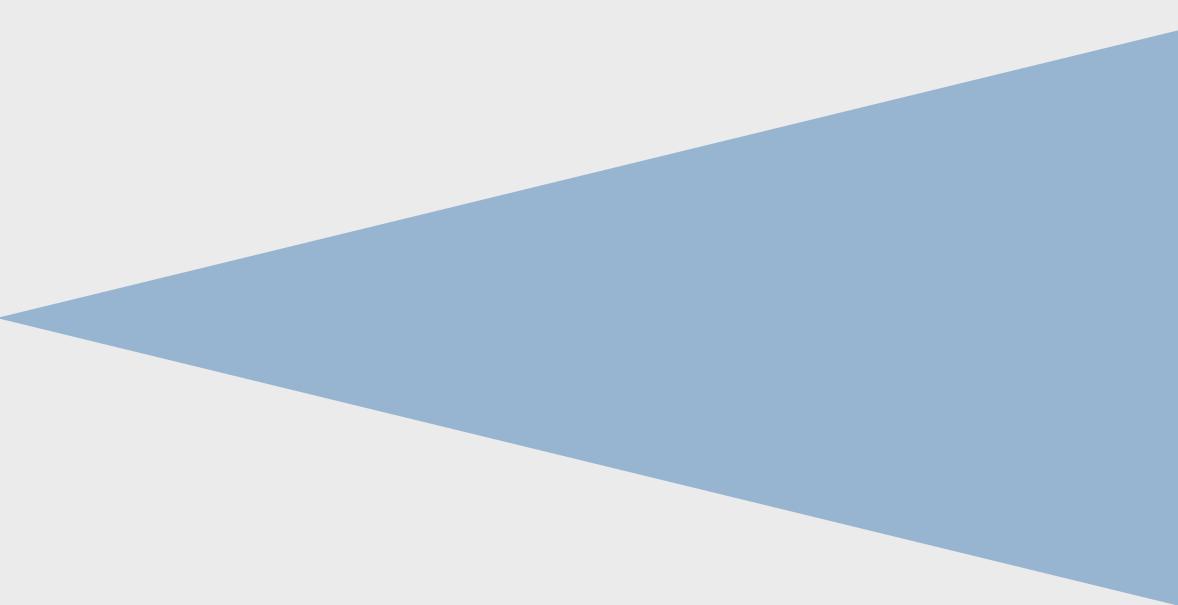
`movies.Rdata`

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(
```



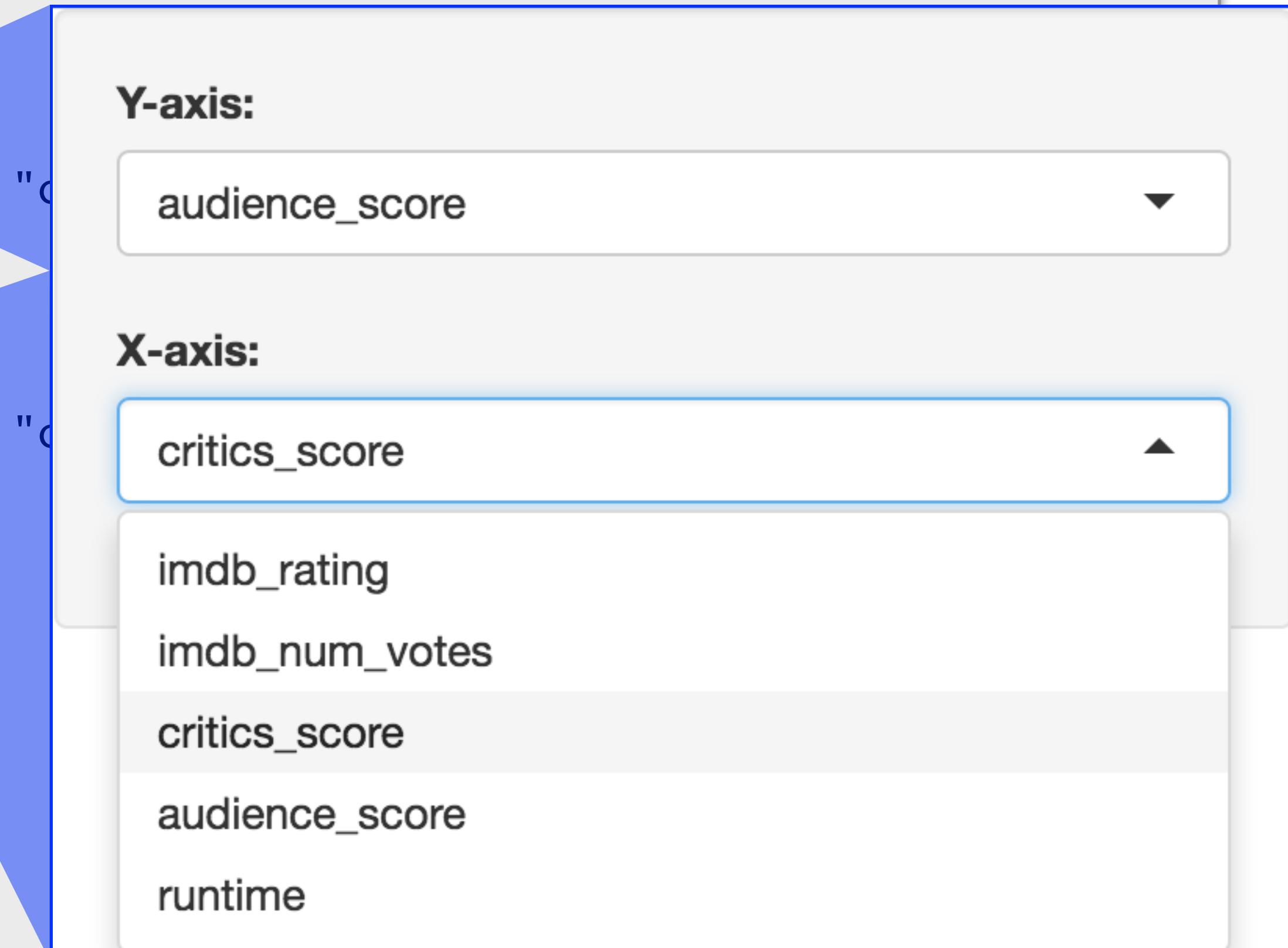
```
      plotOutput(outputId = "scatterplot")
```



```
    )
```



```
  )
```



```
# 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

Create the app

```
# Create Shiny app  
shinyApp(ui = ui, server = server)
```

DEMO



Putting it all together...

`movies_01.R`



- ▶ 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
- ▶ See `movies_02.R`

INPUTS

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

R Studio

Basics

A shiny app is a web page (`UI`) connected to a server running a live R session (`Server`).
Users can manipulate the UI, which will cause the server to update the UI displays (by running R code).
A template

```
library(shiny)
ui <- fluidPage(
  numericInput(inputId = "n",
              label = "Sample size",
              value = 25,
              min = 1,
              max = 100)
  server <- function(input, output) {
    output$hist <- renderPlot({
      hist(rnorm(input$n))
    })
  }
  shinyApp(ui = ui, server = server)
```

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

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

```
library(shiny)
ui <- fluidPage(
  numericInput(inputId = "n",
              label = "Sample size",
              value = 25,
              min = 1,
              max = 100)
  server <- function(input, output) {
    output$hist <- renderPlot({
      hist(rnorm(input$n))
    })
  }
  shinyApp(ui = ui, server = server)
```

The directory name of the app (`app`) defines objects available to both `UI` and `server`.
`DESCRIPTION`: optional file describing the app.
`RDMD`: optional data, scripts, or other files to share with web browsers (images, CSS, JS, etc.) Must be named `www`.

Share your app
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 (F12 or run `rsmash`)
Build or purchase your own Shiny Server at www.rstudio.com/products/shiny-server/

RStudio® is a trademark of RStudio, Inc. • CC-BY-RStudio • info@rstudio.com • 844-448-1212 • studio.com

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

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

Choice 1
 Choice 2
 Choice 3

Check me

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)

Choose File

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

.....

Choice A
 Choice B
 Choice C

Choice 1

Choice 1
 Choice 2

0 5 10

Apply Changes

passwordInput(inputId, label, value)

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

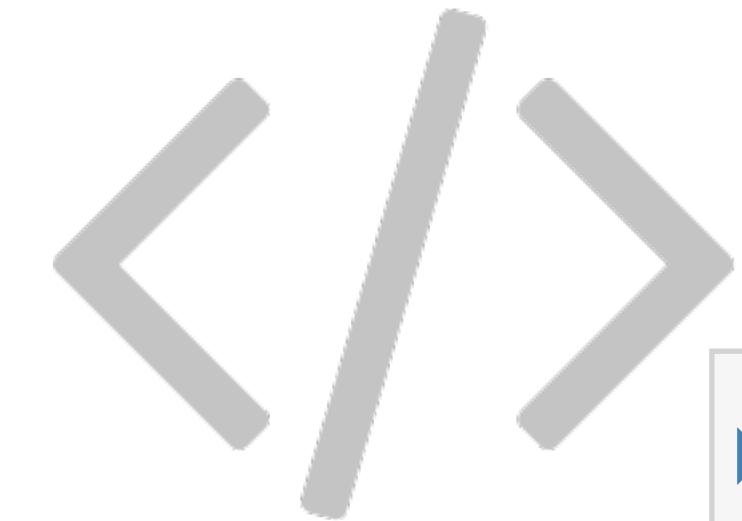
selectInput(inputId, label, choices, selected, multiple, selectize, width, size) (also selectizeInput())

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

Enter text

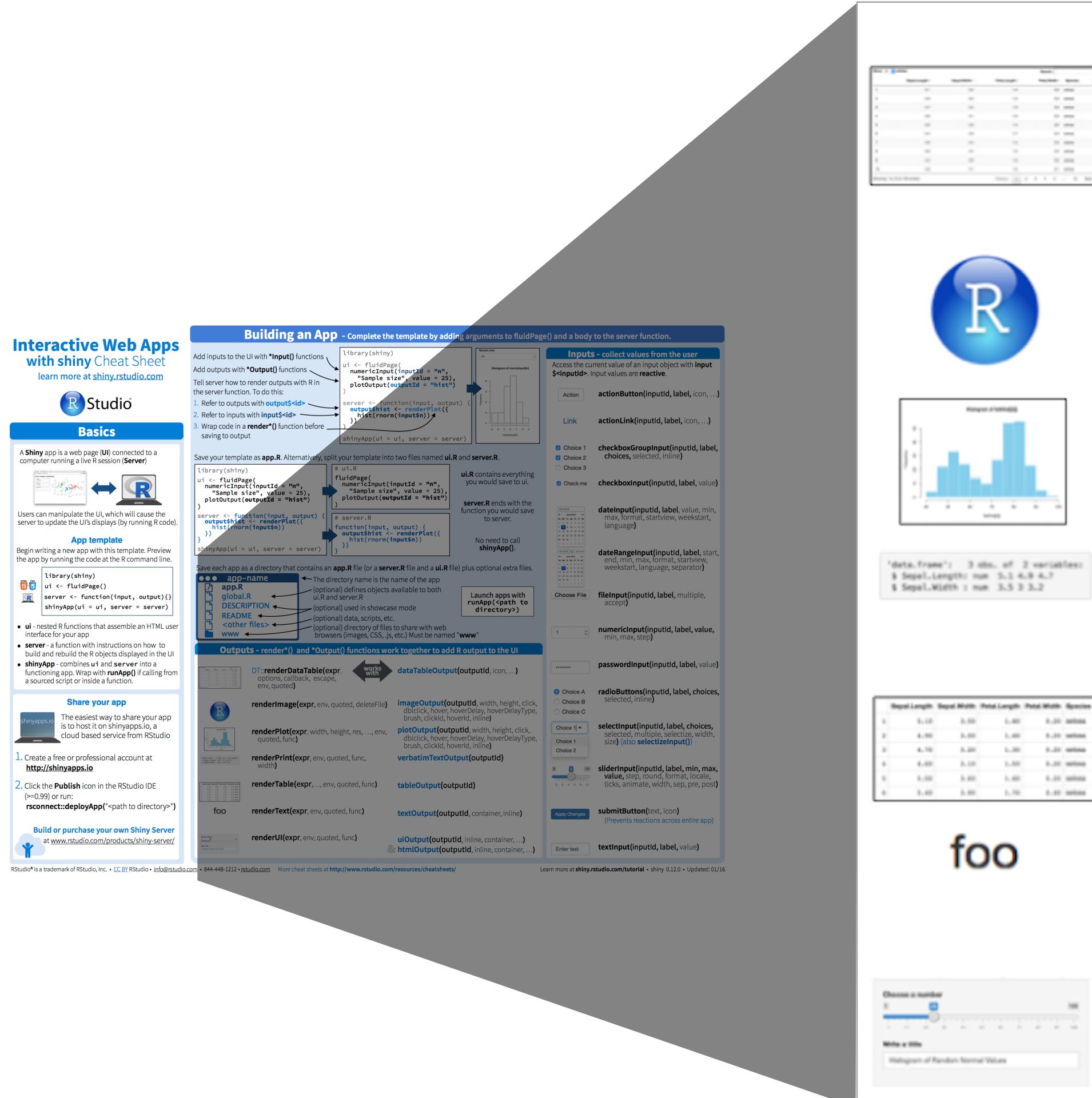
submitButton(text, icon)
(Prevents reactions across entire app)

textInput(inputId, label, value)



- ▶ 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
 - ▶ See **movies_03.R**

OUTPUTS



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

works
with

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)

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)

renderUI(expr, env, quoted, func)



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



- ▶ 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
- ▶ See **movies_04.R**

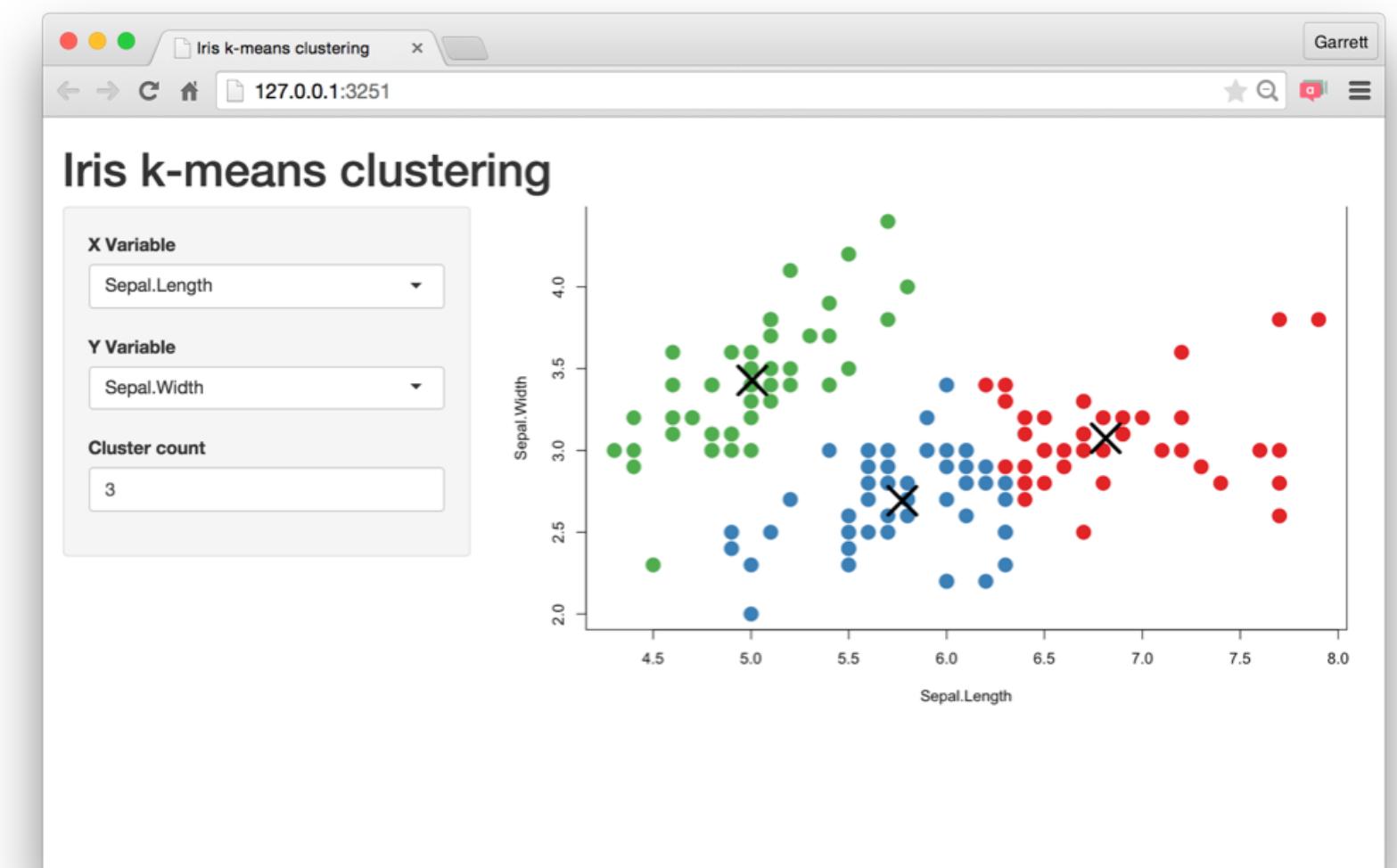
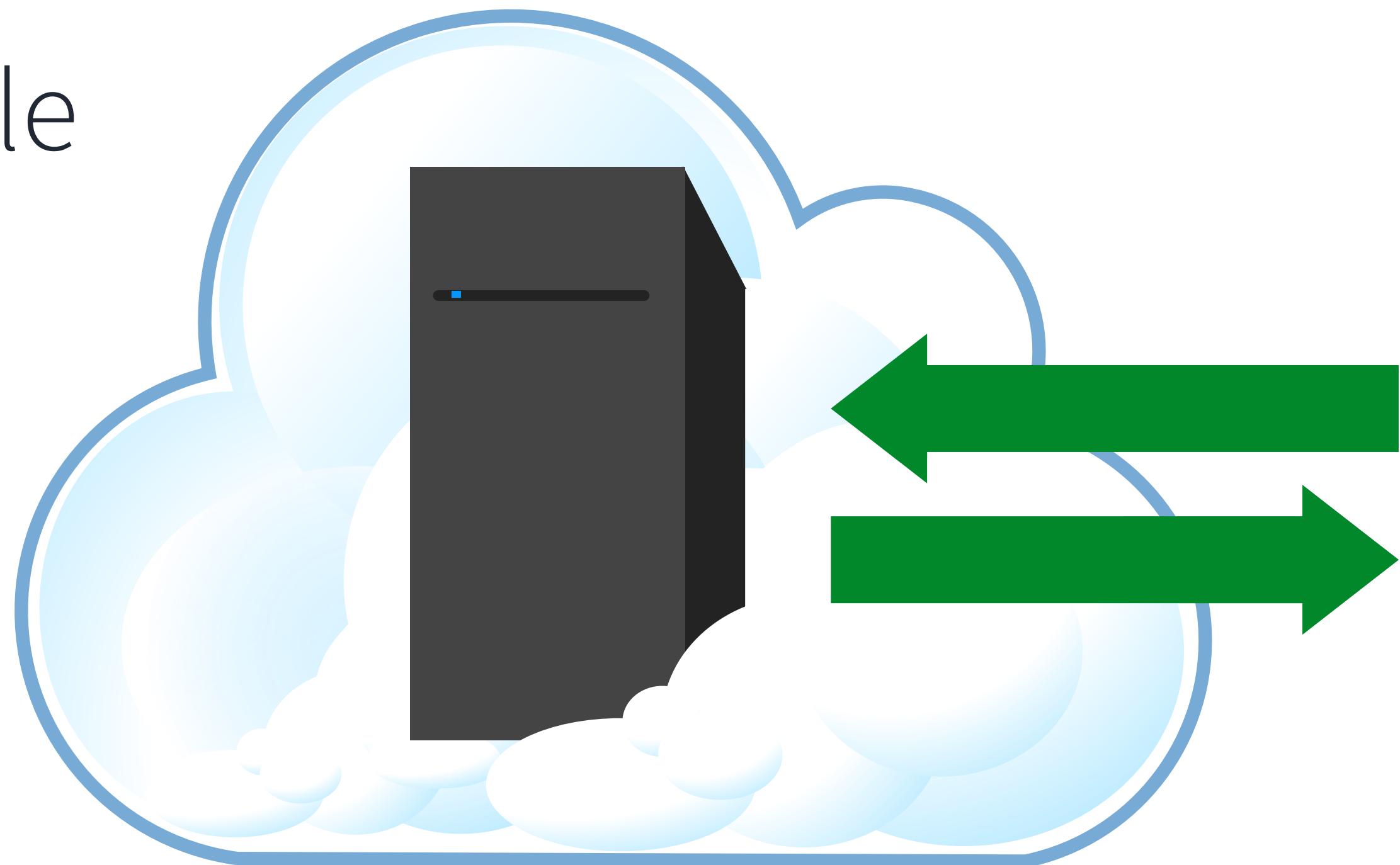
Sharing
your app

shinyapps.io



A server maintained by RStudio

- ▶ easy to use
- ▶ secure
- ▶ scalable



Build your own server



SHINY SERVER PRO

rstudio.com/products/shiny/shiny-server/



✓ **Secure access**

LDAP, GoogleAuth, SSL, and more

✓ **Performance**

fine tune at app and server level

✓ **Management**

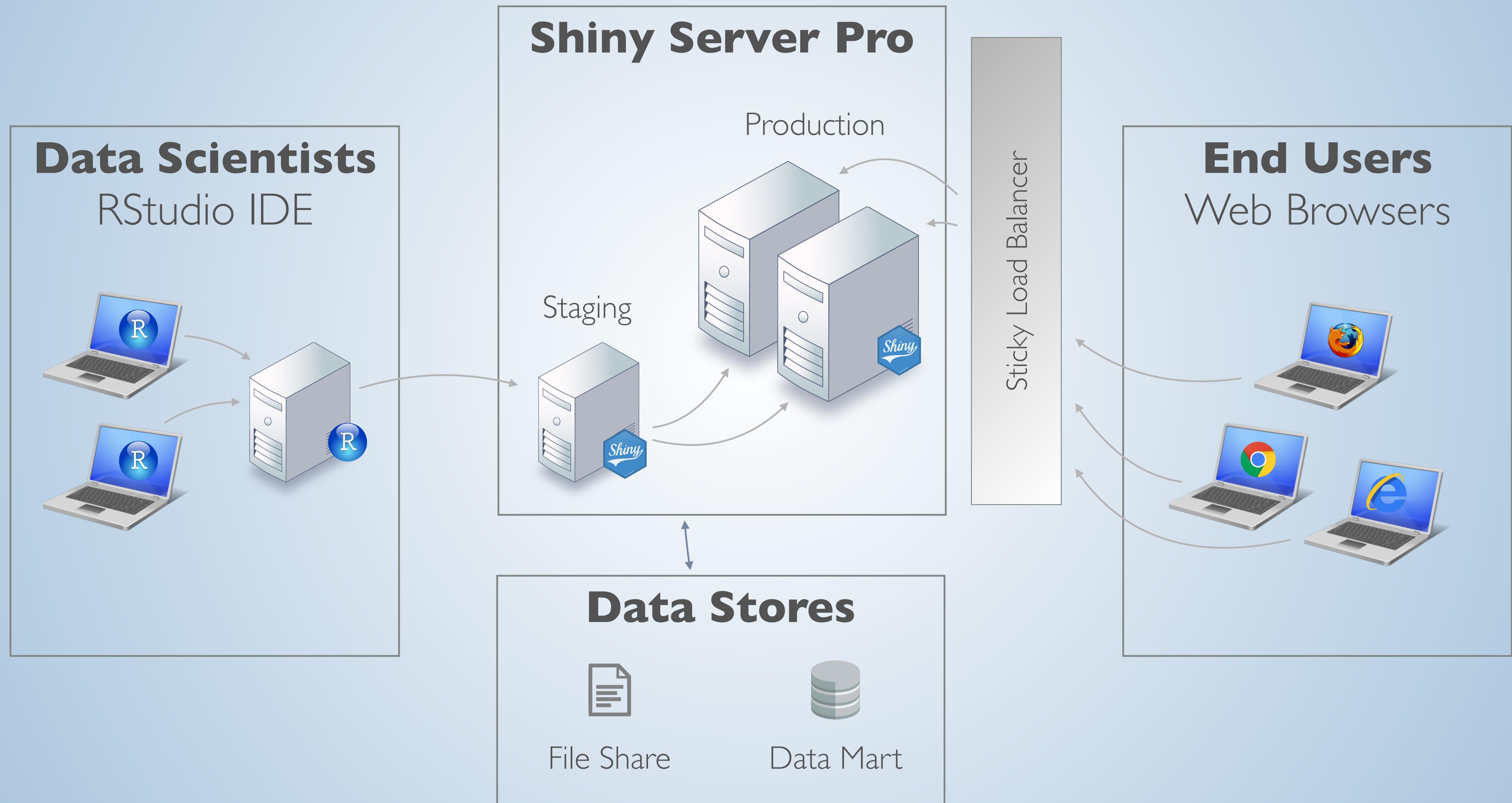
monitor and control resource use

✓ **Support**

direct priority support

45 day
evaluation
free trial

What does a typical setup with SSP look like?



RSTUDIO CONNECT

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



Dashboards

DEMO

movies_05.Rmd

DASHBOARDS

- ▶ Automatically updating
 - ▶ Not just based on user gestures
 - ▶ But also when data source changes
- ▶ Many viewers looking at the same data
- ▶ May or may not be interactive

STATIC VS. DYNAMIC

- ▶ Static:
 - ▶ R code runs once and generates an HTML page
 - ▶ Generation of this HTML can be scheduled
- ▶ Dynamic:
 - ▶ Client web browser connects to an R session running on server
 - ▶ User input causes server to do things and send information back to client
 - ▶ Interactivity can be on client and server
 - ▶ Can update data in real time
 - ▶ User potentially can do anything that R can do

FLEX VS. SHINY DASHBOARD

flexdashboard	shinydashboard
R Markdown	Shiny UI code
Super easy	Not quite as easy
Static or dynamic	Dynamic
CSS flexbox layout	Bootstrap grid layout



DEMO

<https://jjallaire.shinyapps.io/shiny-crardash/>

RStudio interface showing a script editor, console, and viewer.

Script Editor:

```
25 }
26
27 data_props <- combine_data_props(x$marks)
28 data_ids <- names(data_props)
29 data_table <- x$data[data_ids]
30
31 # Collapse each list of scale objects into one scale object.
32 x <- collapse_scales(x)
33 scale_data_table <- scale_domain_data(x)
34
35 # Wrap each of the reactive data objects in another reactive which returns
36 # only the columns that are actually used, and adds any calculated columns
37 # that are used in the props.
38 data_table <- active_props(data_table, data_props)
39
40 # From an environment containing data_table objects, get static data for the
41 # specified ids.
```

16:19 (Top Level) R Script

Console:

```
Console ~/r/ggvis/
> ggvis(diamonds, x = ~price, y = ~color)
>   Guessing layer_histograms()
>   Guessing binwidth = 1
Called from: eval(expr, envir, enclos)
Browse[1]> n
debug at /Users/jmcphers/r/ggvis/R/vega.R#29: data_table <- x$data[data_ids]
Browse[2]>
```

Environment:

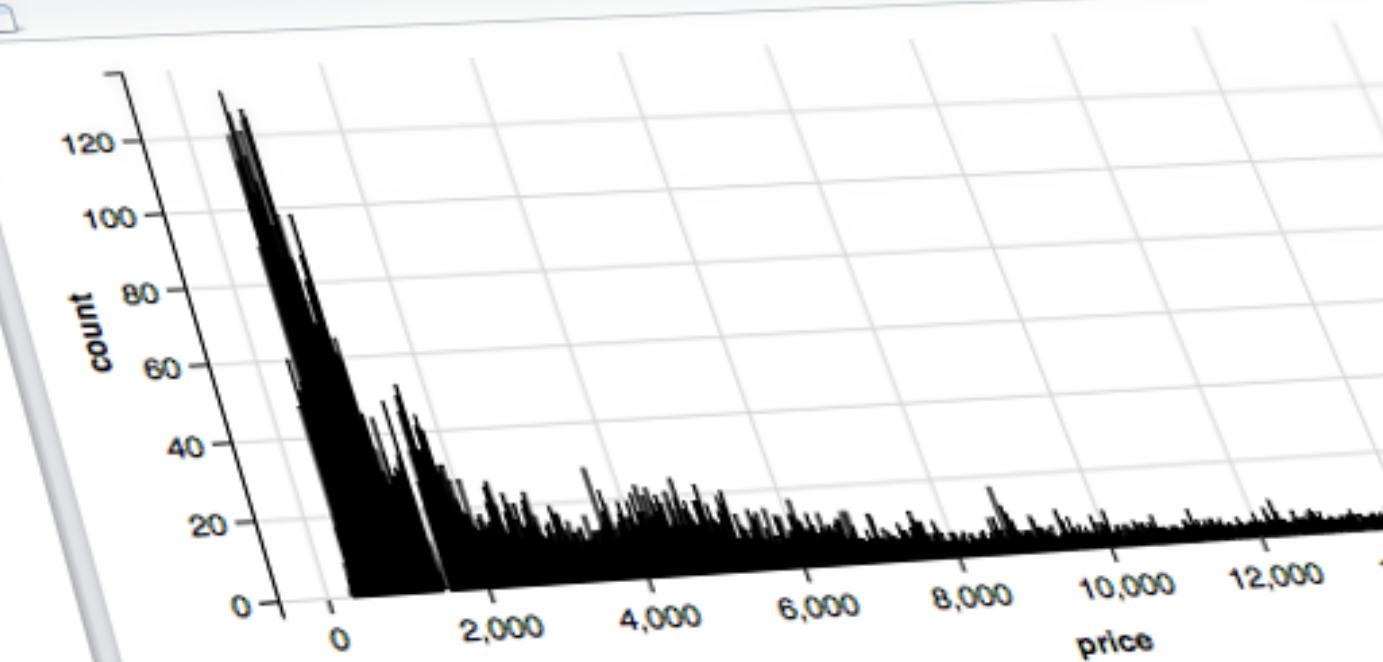
as.vega.ggviz() -

Values	
data_ids	"diamonds0/bin1/stack2"
data_props	List of 1
dynamic	FALSE

Traceback

```
as.vega.ggviz(x, FALSE) at vega.R:29
as.vega(x, FALSE) at vega.R:11
view_static(x, ...) at print.R:67
print.ggviz(c("list()"), "list(diamonds0 = function () \{static_data\}","lis
```

Viewer:



A histogram showing the distribution of diamond prices. The x-axis is labeled 'price' and ranges from 0 to 12,000 with major ticks every 2,000 units. The y-axis is labeled 'count' and ranges from 0 to 120 with major ticks every 20 units. The distribution is highly right-skewed, with the highest frequency occurring at the lowest price points (around 0-1000). The count drops sharply as price increases, with most diamonds costing less than \$10,000.