```
}

data_props <- combine_data_props(x$marks)
data_ids <- names(data_props)
data_table <- x$data[data_ids]

# Collapse each list of scale objects into one scale object.
x <- collapse_scales(x)
scale_data_table <- scale_domain_data(x)

# Wrap each of the reactive data objects in another reactive which returns
# only the columns that are actually used, and adds any calculated columns
# that are used in the props.
data_table <- active_props(data_table, data_props)

# From an environment containing data_table objects, get static data for the
# specified ids.
```

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

Traceback
as.vega.ggvis(x, FALSE) at vega.R:29
as.vega(x, FALSE) at vega.R:11
view_static(x, ...) at print.R:67
print.ggvis(c("list()", "list(diamonds0 = function () \nstatic_data)", "li
```

```
> 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]>
```

WELCOME TO SHINY

Shiny from RStudio

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]

(including links to deployed apps used in demos)

Anatomy of a Shiny app

WHAT'S IN AN APP?

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


server <- function(input, output) {}


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

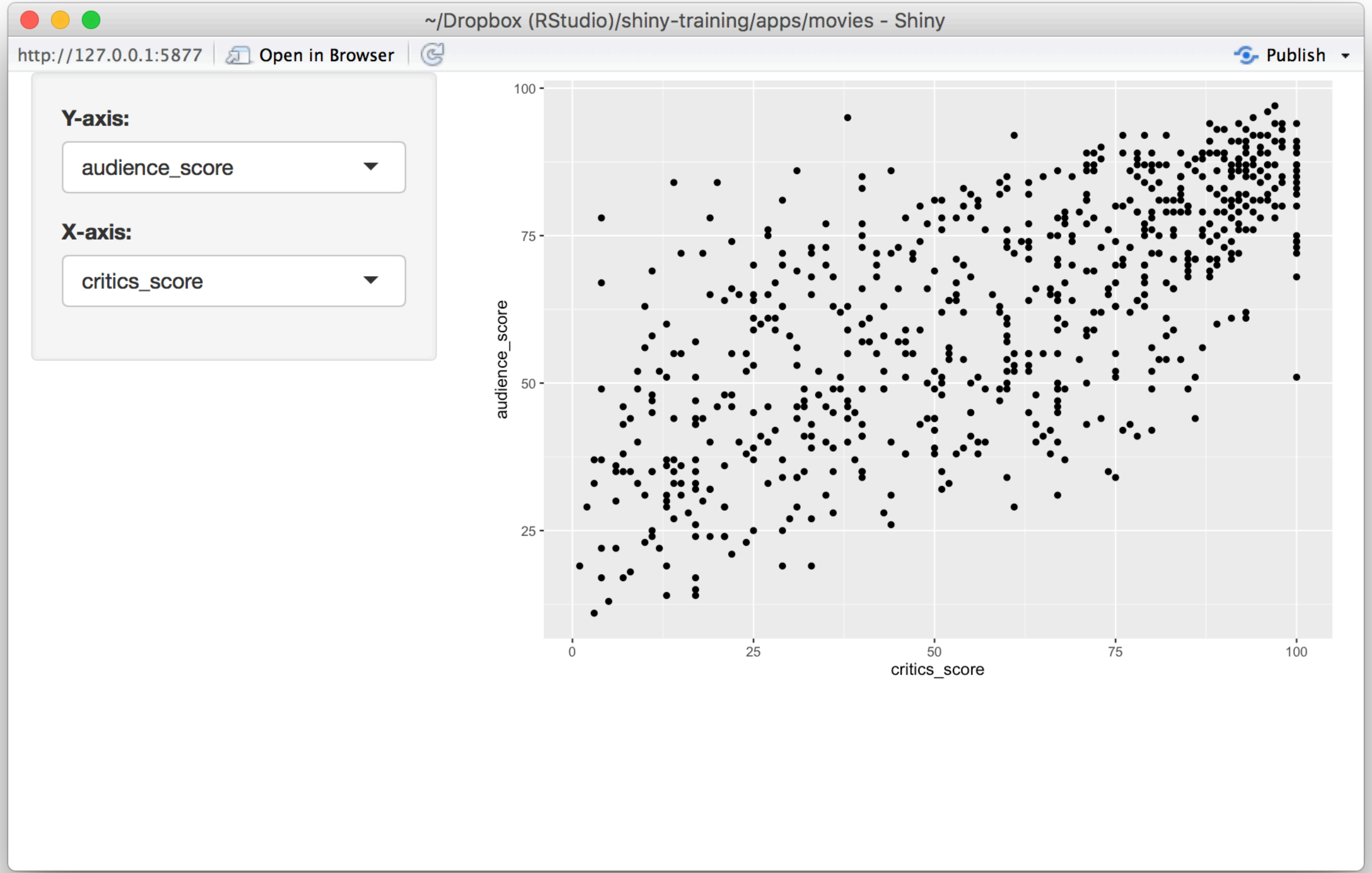
User interface
controls the layout and appearance of app

Server function
contains instructions needed to build app

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


~/Dropbox (RStudio)/shiny-training/apps/movies - Shiny

http://127.0.0.1:5877 Open in Browser Publish

Y-axis: audience_score

X-axis: critics_score

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

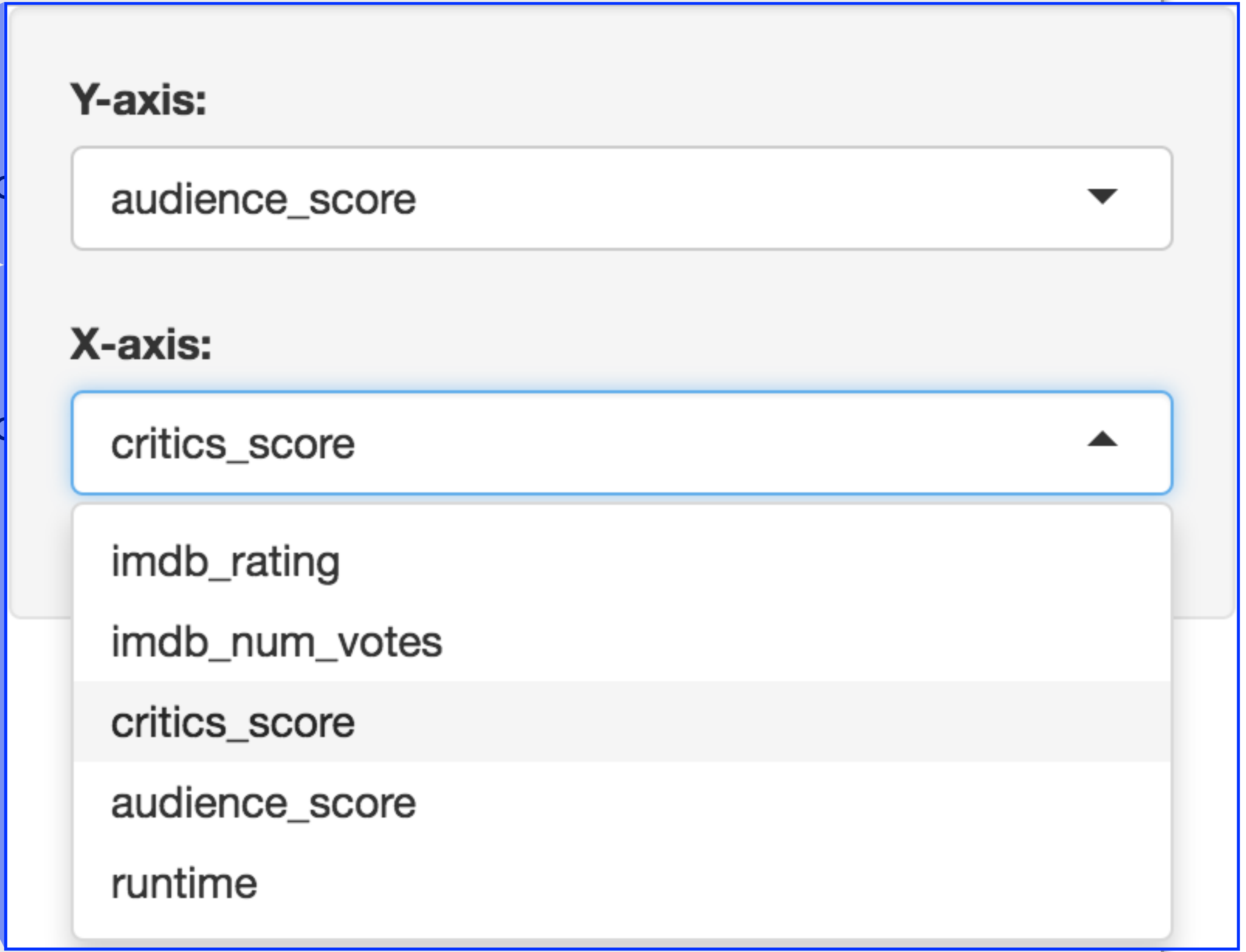
```r
# Define UI for application that plots features of movies
ui <- fluidPage(

  # 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

```r
# 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`

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

    # Output: Show scatterplot
    mainPanel(
      plotOutput(outputId = "scatterplot")
    )
  )
)
```



```r
# 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

```r
# 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

Shiny from RStudio

```r
# 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 **input**s from UI

Shiny from RStudio

Create the app

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

DEMO

Putting it all together…

`apps/movies_01/app.R`

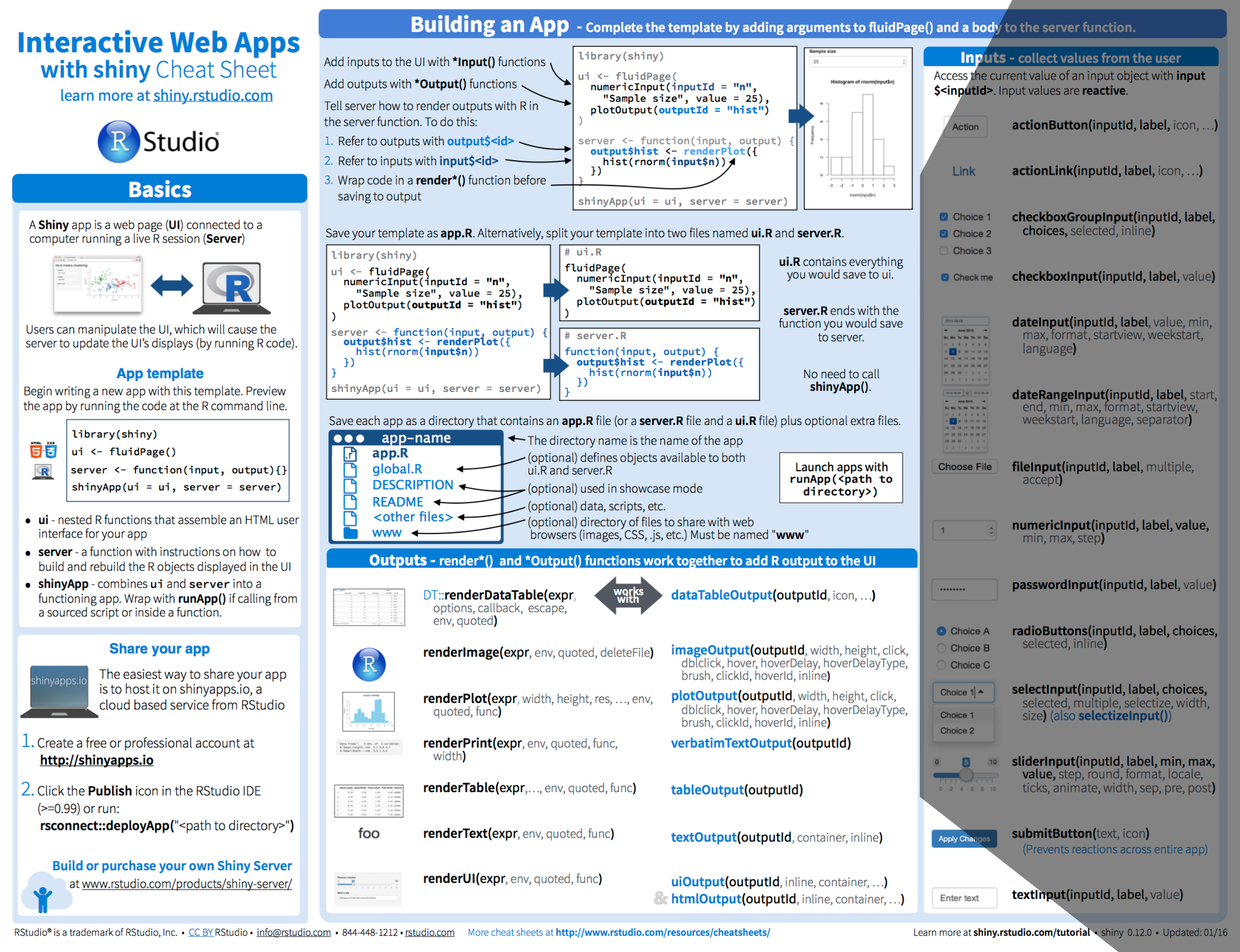
https://minecr.shinyapps.io/movies_01/

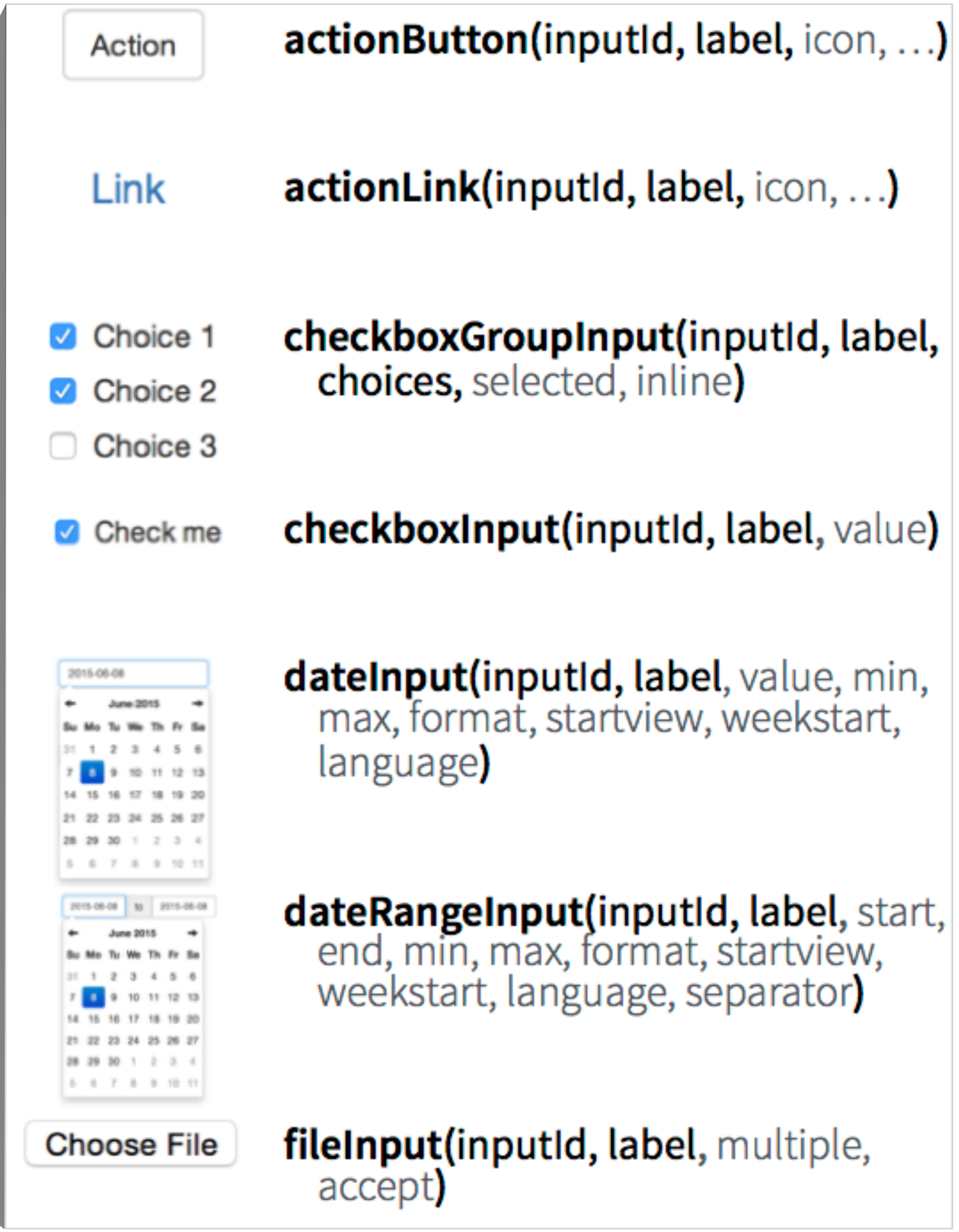
DEMO

- 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 `apps/movies_02/app.R` or

https://minecr.shinyapps.io/movies_02/

INPUTS





actionButton(inputId, label, icon, …)

actionLink(inputId, label, icon, …)

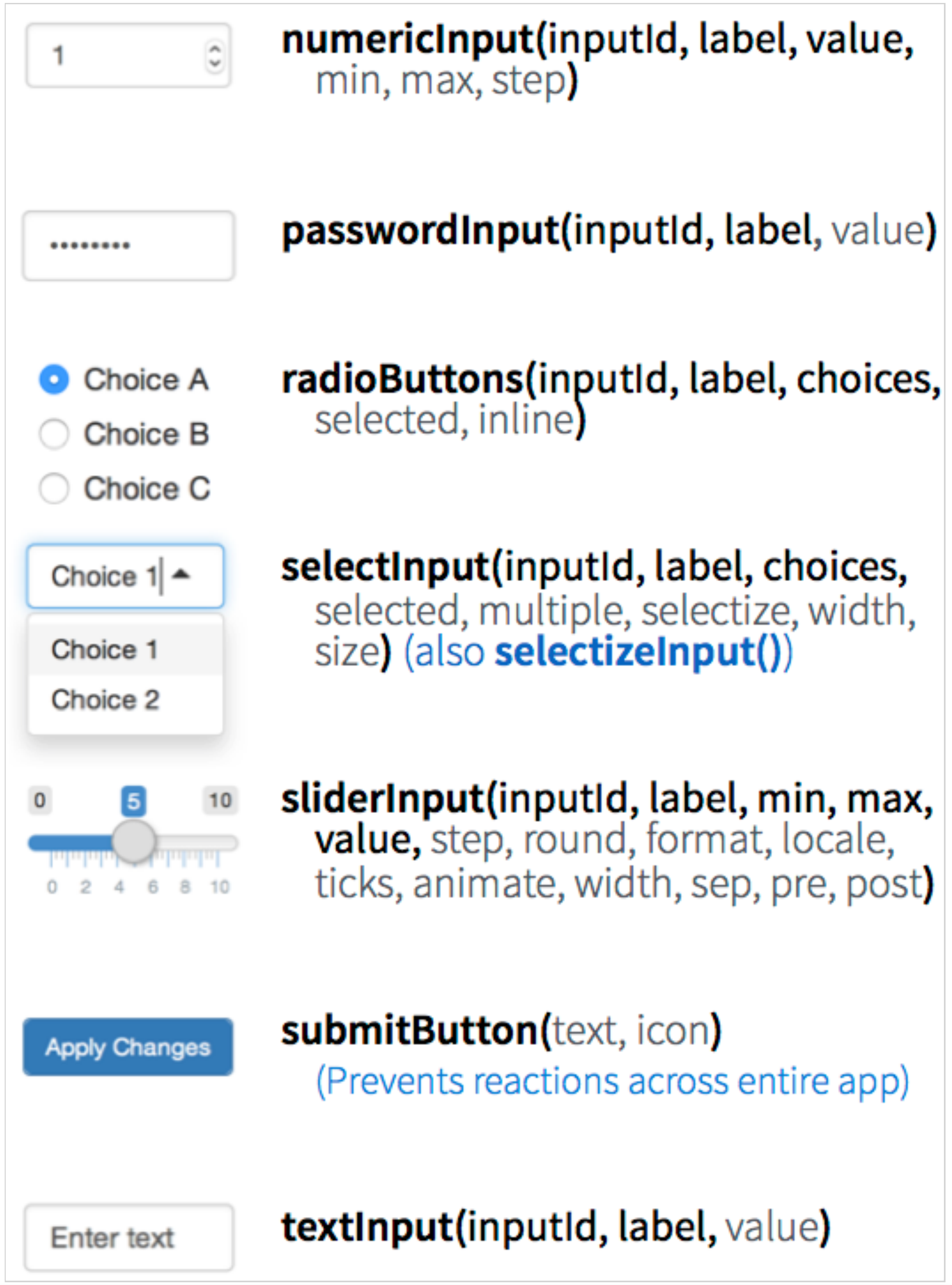
checkboxGroupInput(inputId, label, choices, selected, inline)

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)

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)

Shiny from R Studio™

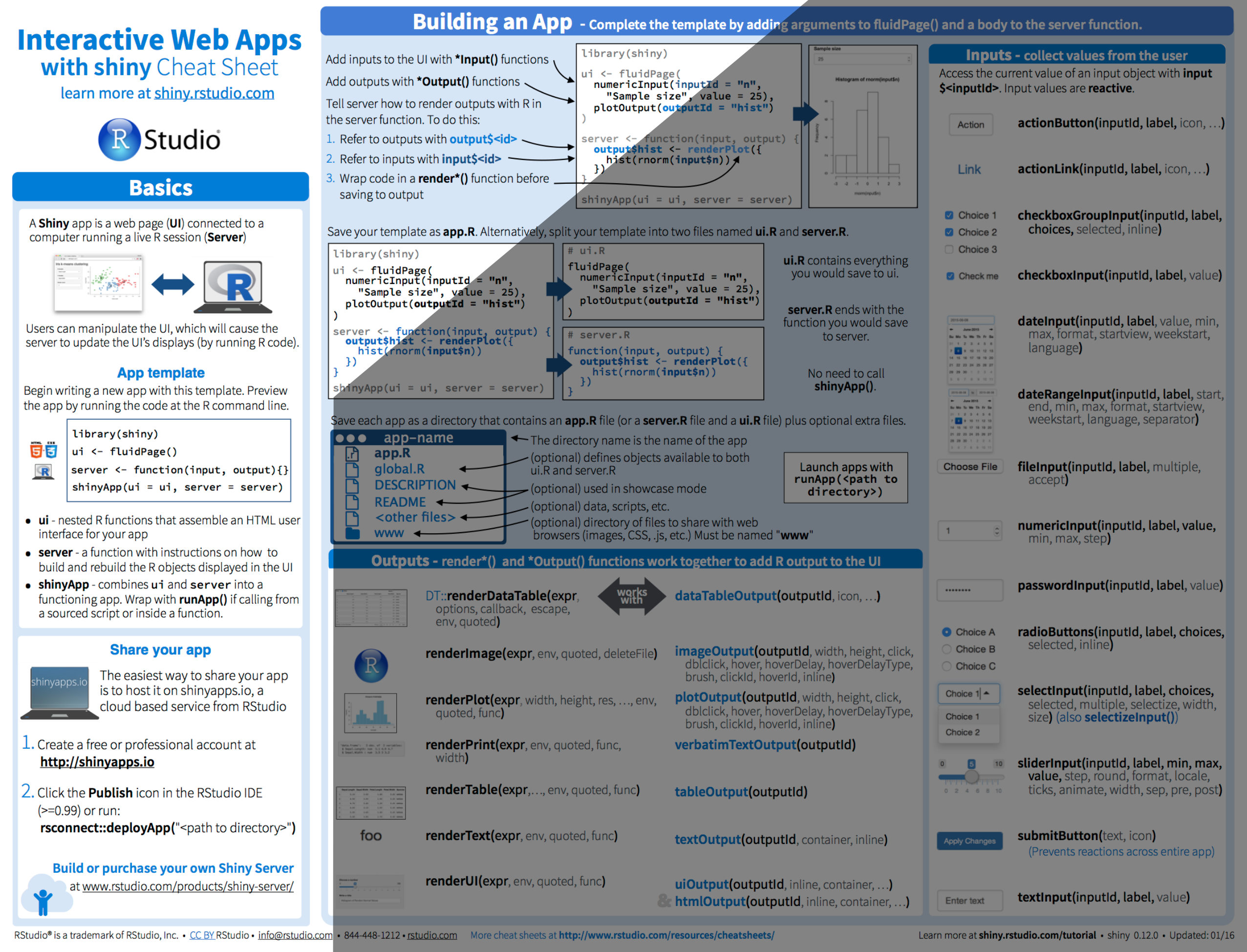
https://www.rstudio.com/resources/cheatsheets/

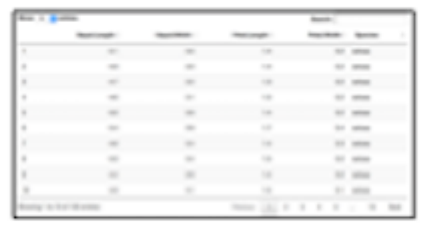
DEMO

- 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 `apps/movies_03/app.R` or

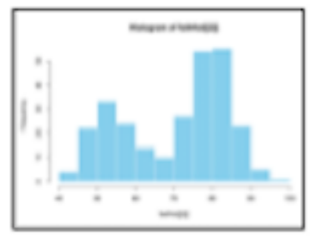
https://minecr.shinyapps.io/movies_03/

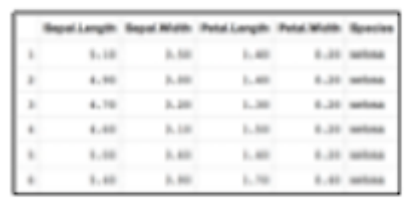
OUTPUTS





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





foo

DEMO

- 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 `apps/movies_04/app.R` or

https://minecr.shinyapps.io/movies_04/

Sharing

your app

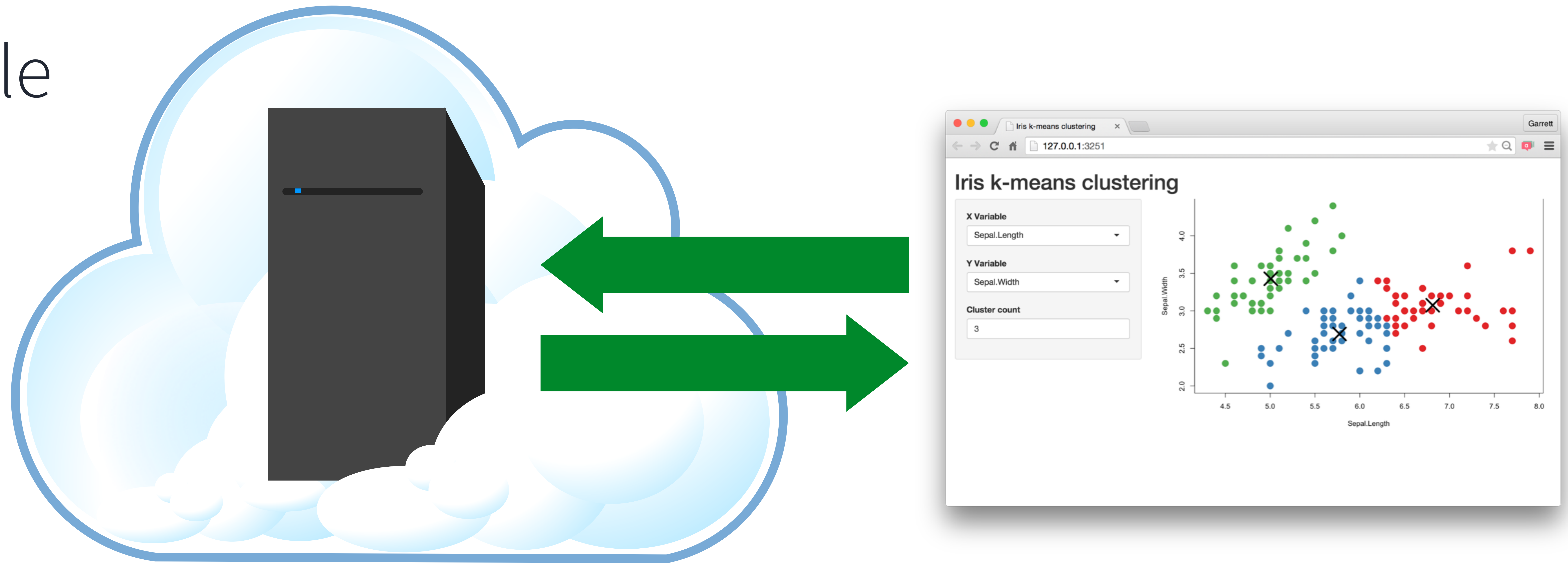
shinyapps.io

by RStudio®

SHINYAPPS.IO

A server maintained by RStudio

- easy to use
- secure
- scalable



Iris k-means clustering

X Variable: Sepal.Length

Y Variable: Sepal.Width

Cluster count: 3

Build your own server

SHINY SERVER

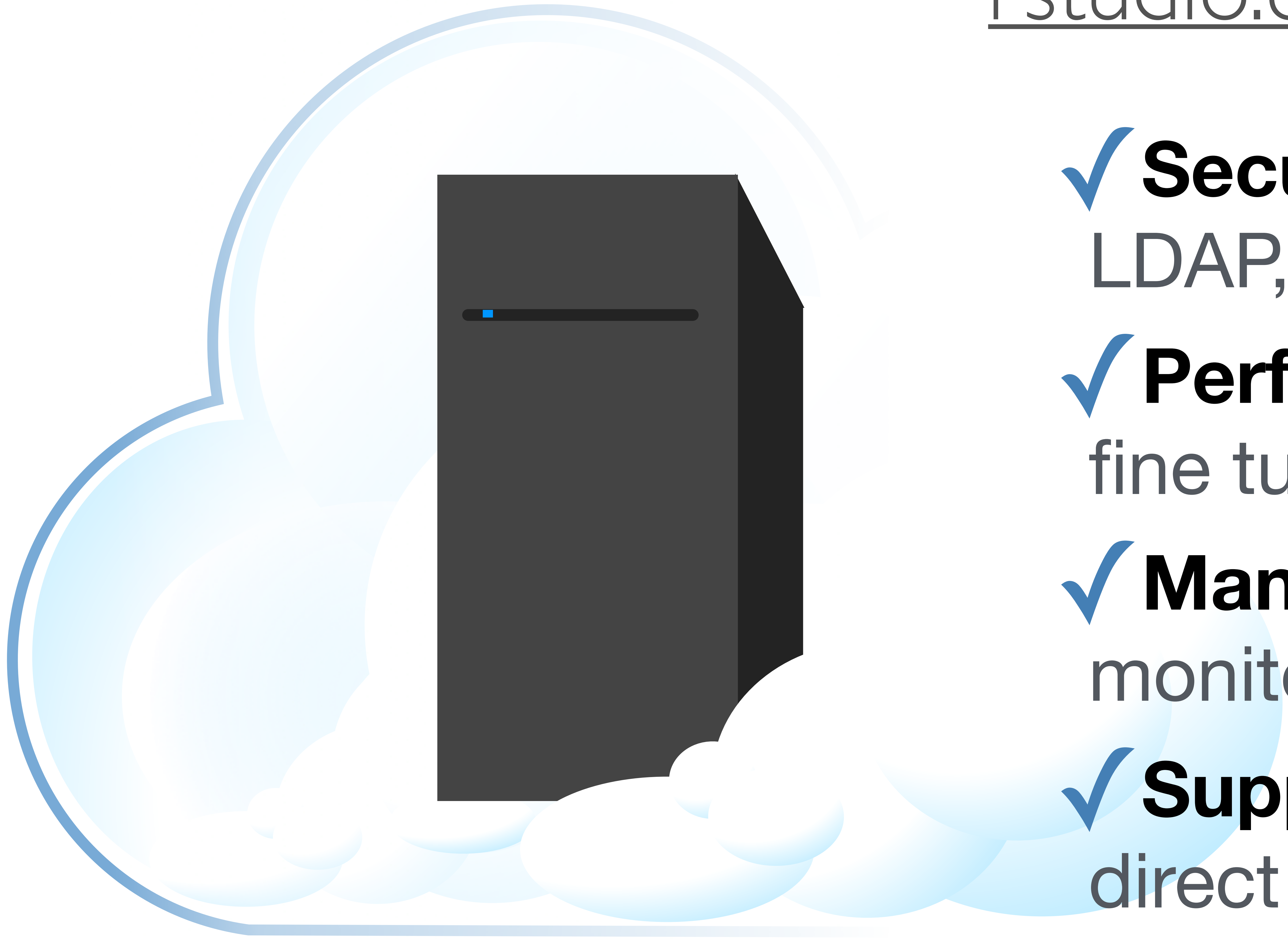
rstudio.com/products/shiny/shiny-server/

- ✓ **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**

Free & open source

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

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

45 day evaluation free trial

Dashboards

DEMO

apps/movies_05/movies_05.Rmd

https://minecr.shinyapps.io/movies_05/

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

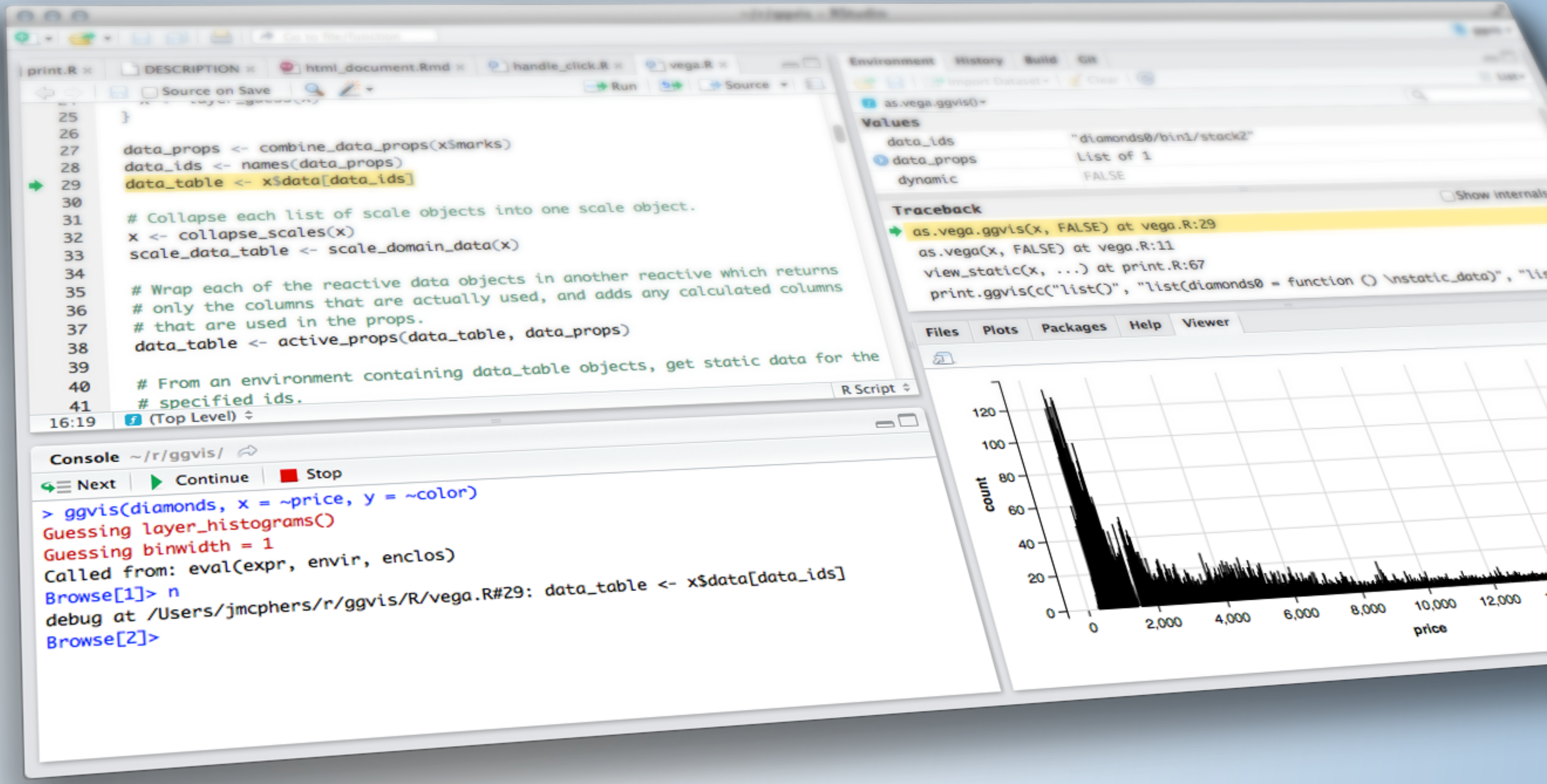
- Static:
  - R code runs once and generates an HTML page
  - Generation of this HTML can be scheduled
- Dynamic:
  - Client web browser connects to an R session running on server
  - User input causes server to do things and send information back to client
  - Interactivity can be on client and server
  - 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-crandash/



```
}

data_props <- combine_data_props(x$marks)
data_ids <- names(data_props)
data_table <- x$data[data_ids]

# Collapse each list of scale objects into one scale object.
x <- collapse_scales(x)
scale_data_table <- scale_domain_data(x)

# Wrap each of the reactive data objects in another reactive which returns
# only the columns that are actually used, and adds any calculated columns
# that are used in the props.
data_table <- active_props(data_table, data_props)

# From an environment containing data_table objects, get static data for the
# specified ids.
```

16:19

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]>
```

Values

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

Traceback

```
as.vega.ggvis(x, FALSE) at vega.R:29
as.vega(x, FALSE) at vega.R:11
view_static(x, ...) at print.R:67
print.ggvis(c("list()", "list(diamonds0 = function () \nstatic_data)", "li
```

Shiny from RStudio™