Looking further: Interactive web applications with Shiny

STA 199 - Dr. Çetinkaya-Rundel

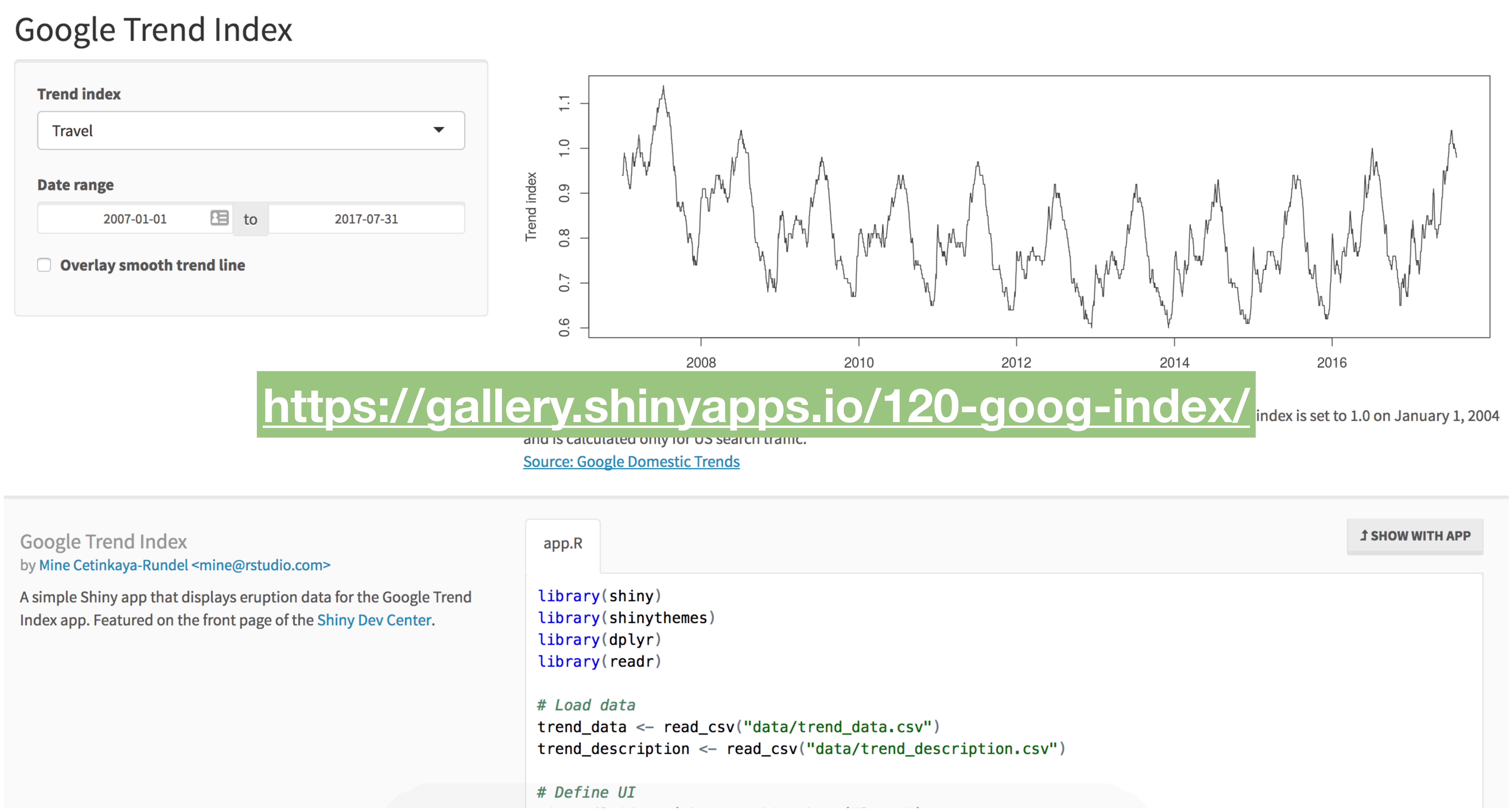
2022-12-08

Announcements

- Projects due **tonight** at 11:59 pm
- HW 6 due tomorrow (Friday) at 11:59 pm
- Team peer evaluations due Sunday at 11:59 pm
- Exam retake (optional) due Thursday, December 15 at 5 pm — no late work will be accepted

Outline

- High level view
- Anatomy of a Shiny app
- Reactivity 101
- File structure



Google Trend Index

Trend index: Travel

Date range: 2007-01-01 to 2017-07-31

Overlay smooth trend line

https://gallery.shinyapps.io/120-goog-index/

index is set to 1.0 on January 1, 2004 and is calculated only for US search traffic.

Source: Google Domestic Trends

Google Trend Index

by Mine Cetinkaya-Rundel <mine@rstudio.com>

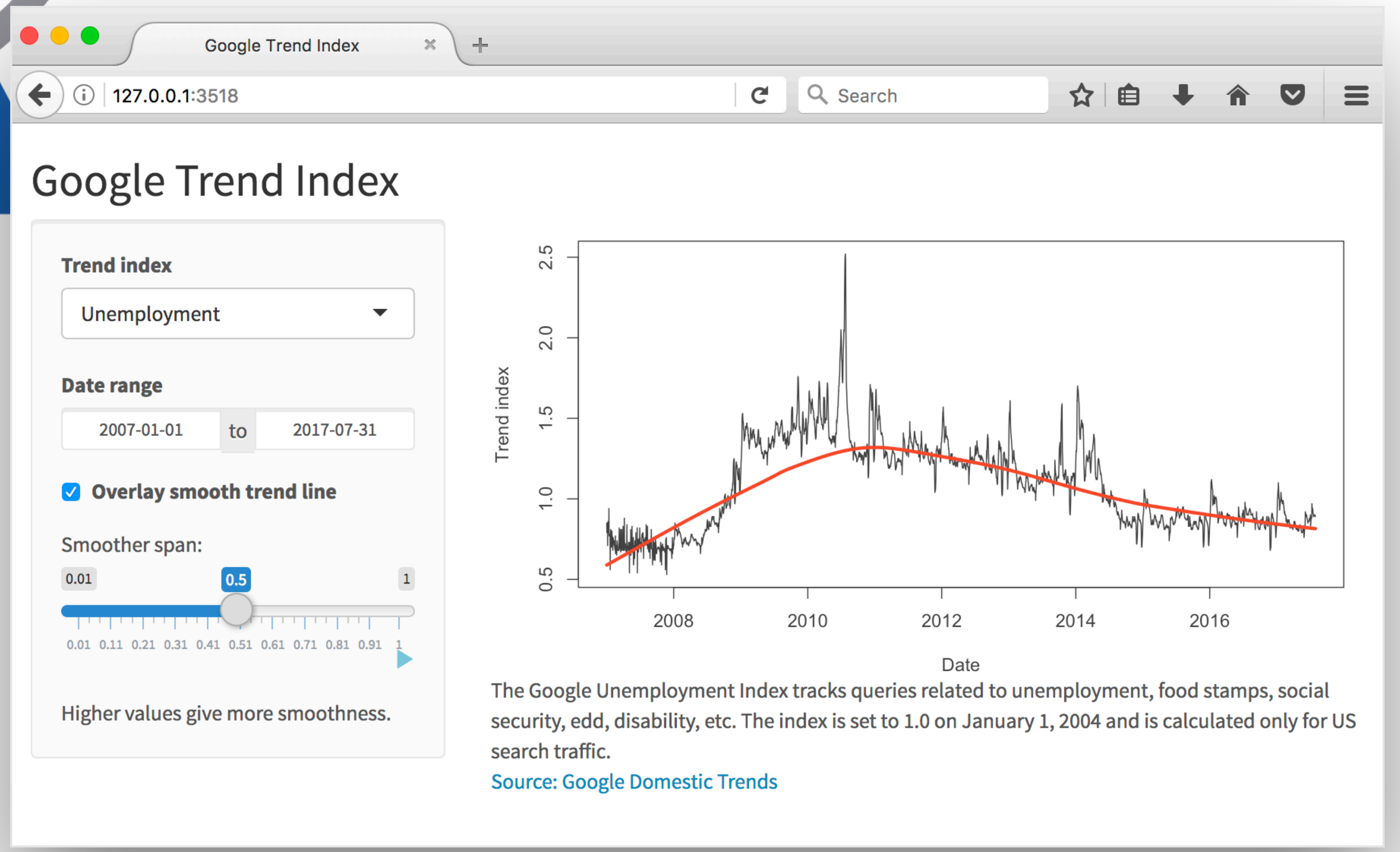
A simple Shiny app that displays eruption data for the Google Trend Index app. Featured on the front page of the Shiny Dev Center.

app.R

```
library(shiny)
library(shinythemes)
library(dplyr)
library(readr)

# Load data
trend_data <- read_csv("data/trend_data.csv")
trend_description <- read_csv("data/trend_description.csv")

# Define UI
```

High level view

Every Shiny app has a webpage that the user visits, and behind this webpage there is a computer that serves this webpage by running R.



Google Trend Index

Trend index

Unemployment

Date range

2007-01-01 to 2017-07-31

☑ **Overlay smooth trend line**

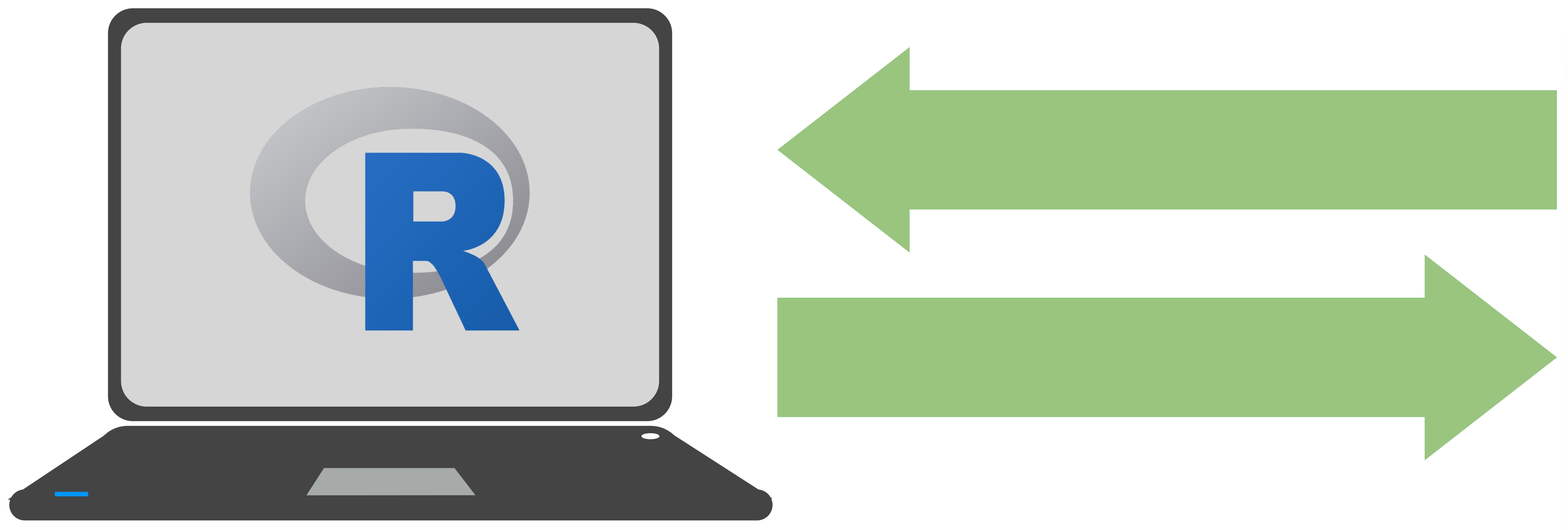
Smoother span:

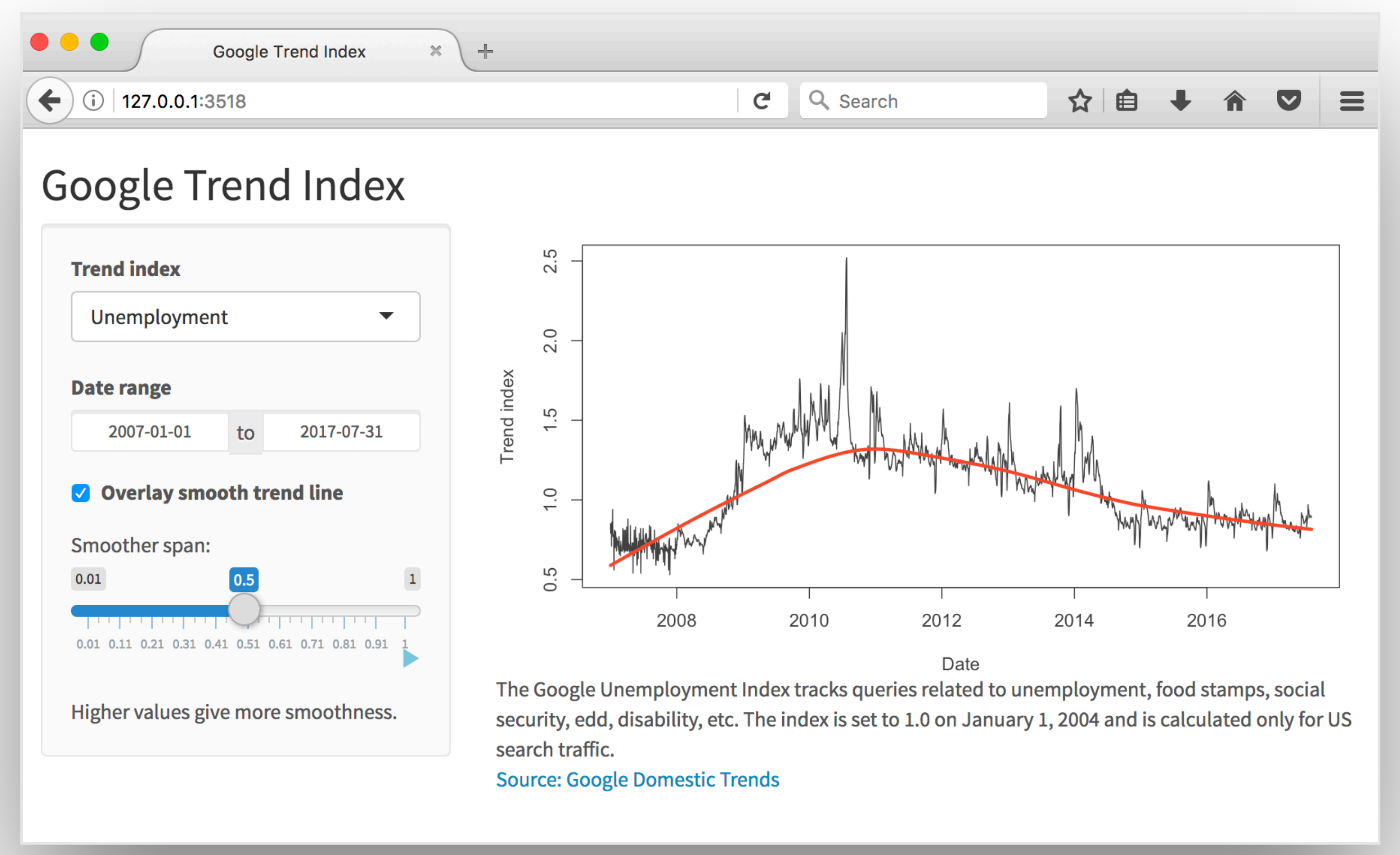
Higher values give more smoothness.

The Google Unemployment Index tracks queries related to unemployment, food stamps, social security, edd, disability, etc. The index is set to 1.0 on January 1, 2004 and is calculated only for US search traffic.

Source: Google Domestic Trends

When running your app locally,
the computer serving your app is your computer.

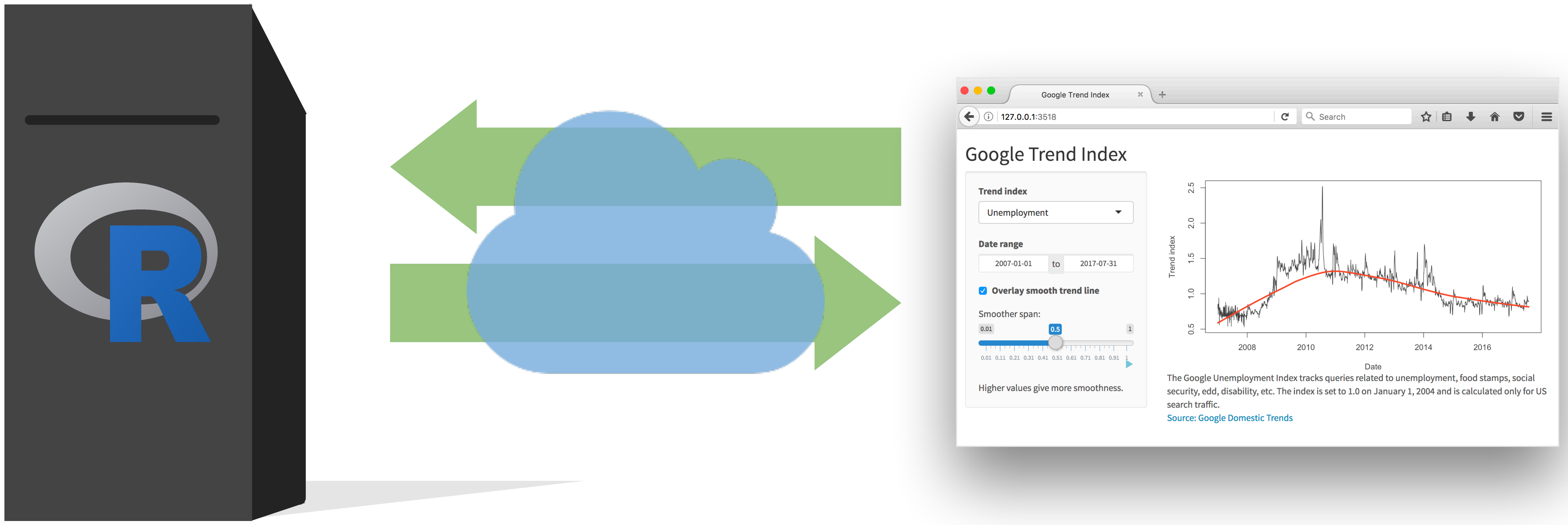




The Google Unemployment Index tracks queries related to unemployment, food stamps, social security, edd, disability, etc. The index is set to 1.0 on January 1, 2004 and is calculated only for US search traffic.
Source: Google Domestic Trends

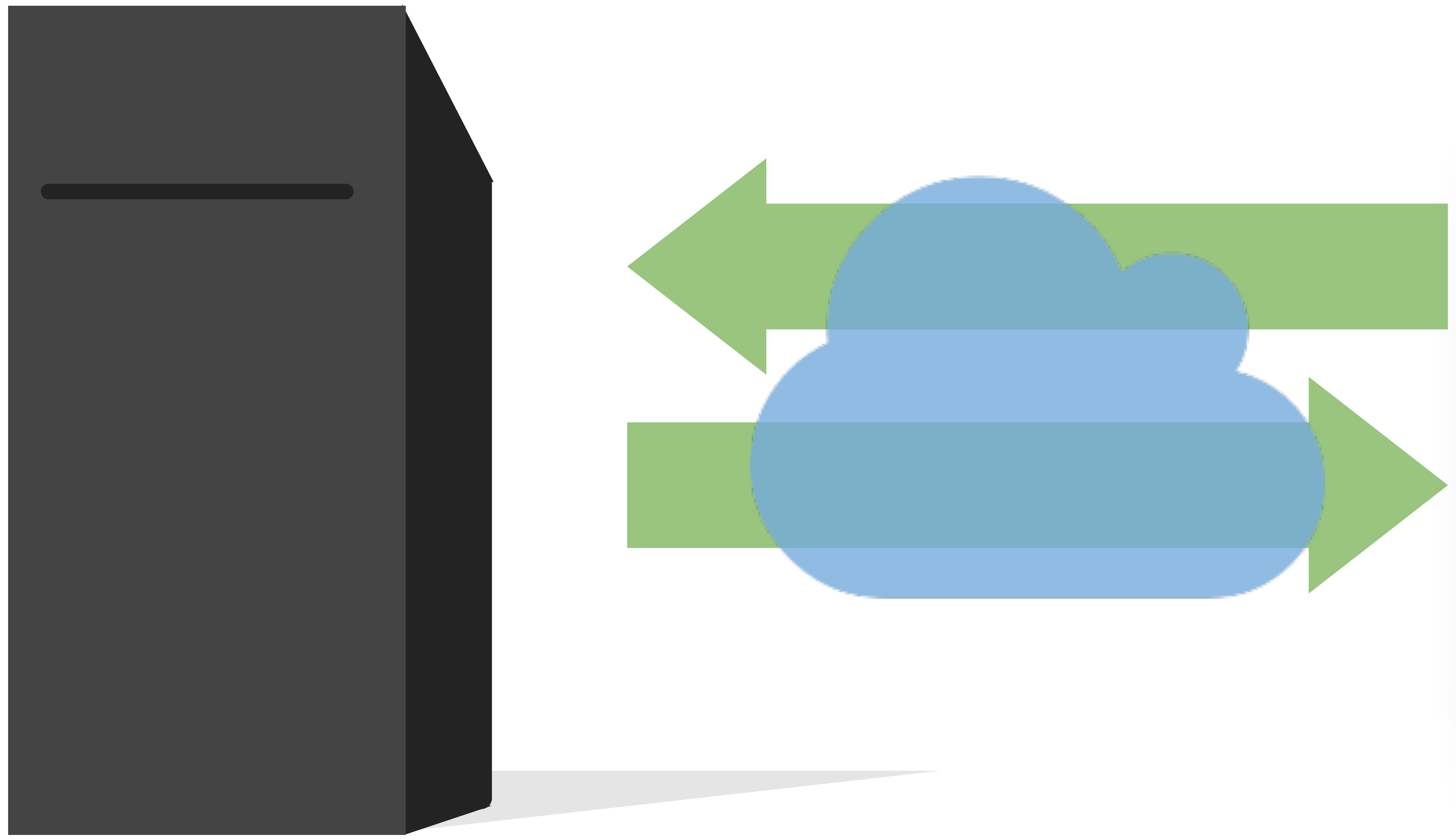
When your app is deployed,

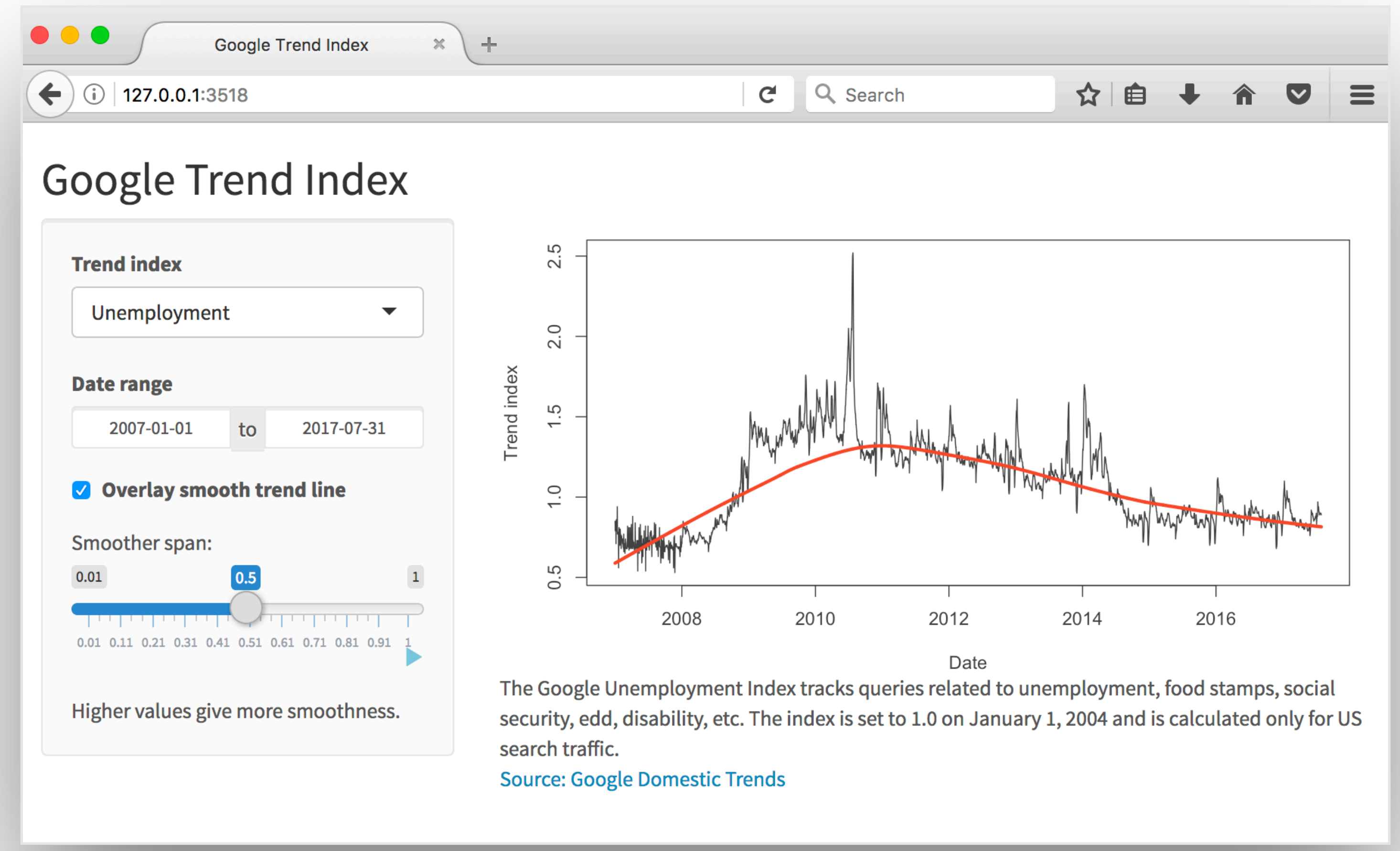
the computer serving your app is a web server.



The Google Unemployment Index tracks queries related to unemployment, food stamps, social security, edd, disability, etc. The index is set to 1.0 on January 1, 2004 and is calculated only for US search traffic.

Source: Google Domestic Trends




Google Trend Index

127.0.0.1:3518

Google Trend Index

Trend index

Unemployment

Date range

2007-01-01 to 2017-07-31

☑ **Overlay smooth trend line**

Smoother span: 0.5 (0.01 – 1)

Higher values give more smoothness.

The Google Unemployment Index tracks queries related to unemployment, food stamps, social security, edd, disability, etc. The index is set to 1.0 on January 1, 2004 and is calculated only for US search traffic.

Source: Google Domestic Trends

Server instructions

User interface

Interactive viz

goog-index/app.R

Anatomy of a Shiny app

What’s in a Shiny app?

```
library(shiny)

ui <- fluidPage()


server <- function(input, output) {}


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

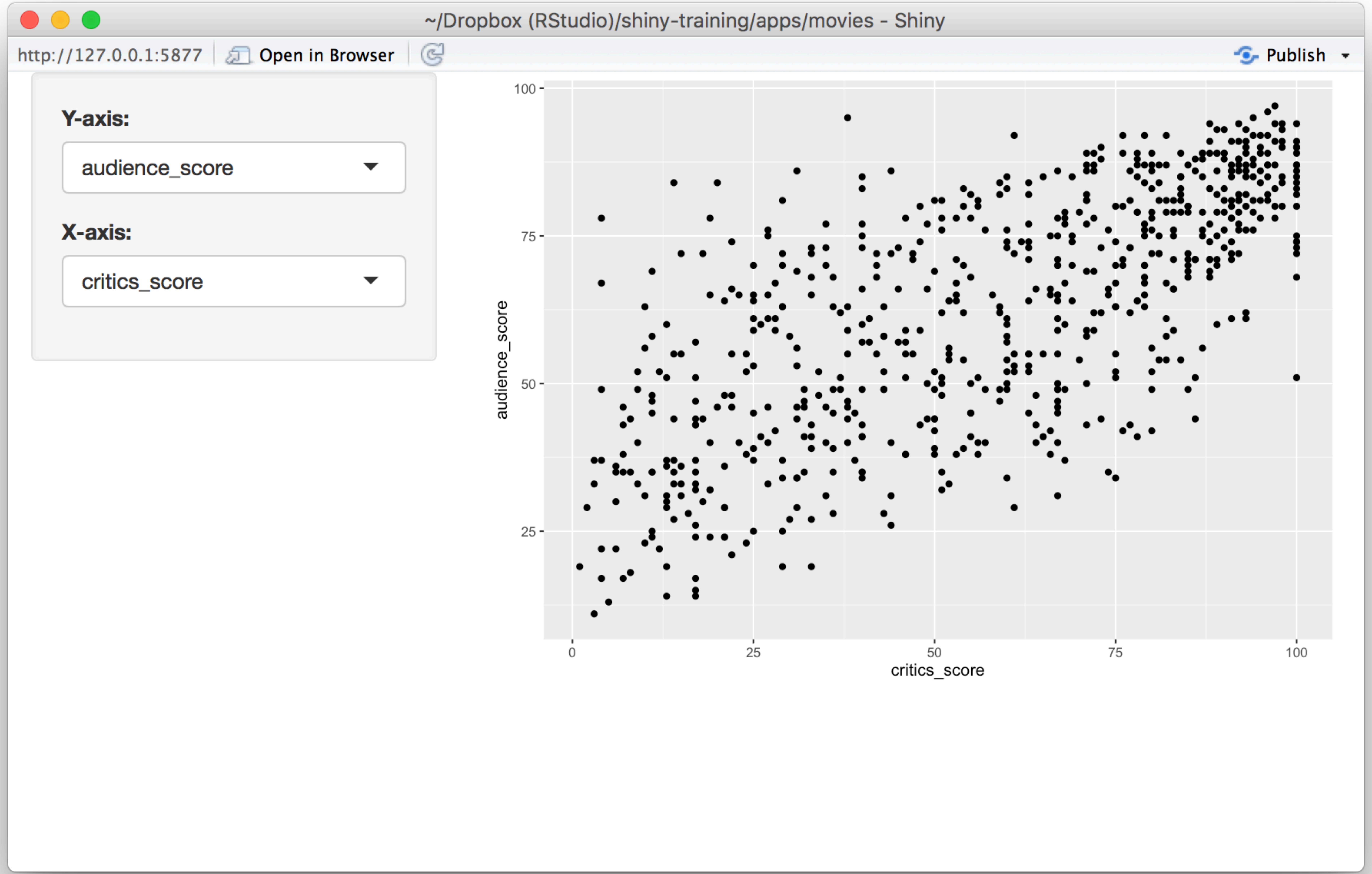
User interface
controls the layout and appearance of app

Server function
contains instructions needed to build app

Let’s build a simple movie browser app!

`data/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
100
75
50
25
audience_score
0
25
50
75
100
critics_score

App template

```
library(shiny)
library(tidyverse)
load("data/movies.Rdata")
ui <- fluidPage()

server <- function(input, output) {}

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

Dataset used for this app

Anatomy of a Shiny app

User interface

```
# Define UI
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")
    )
  )
```

```r
# Define UI
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

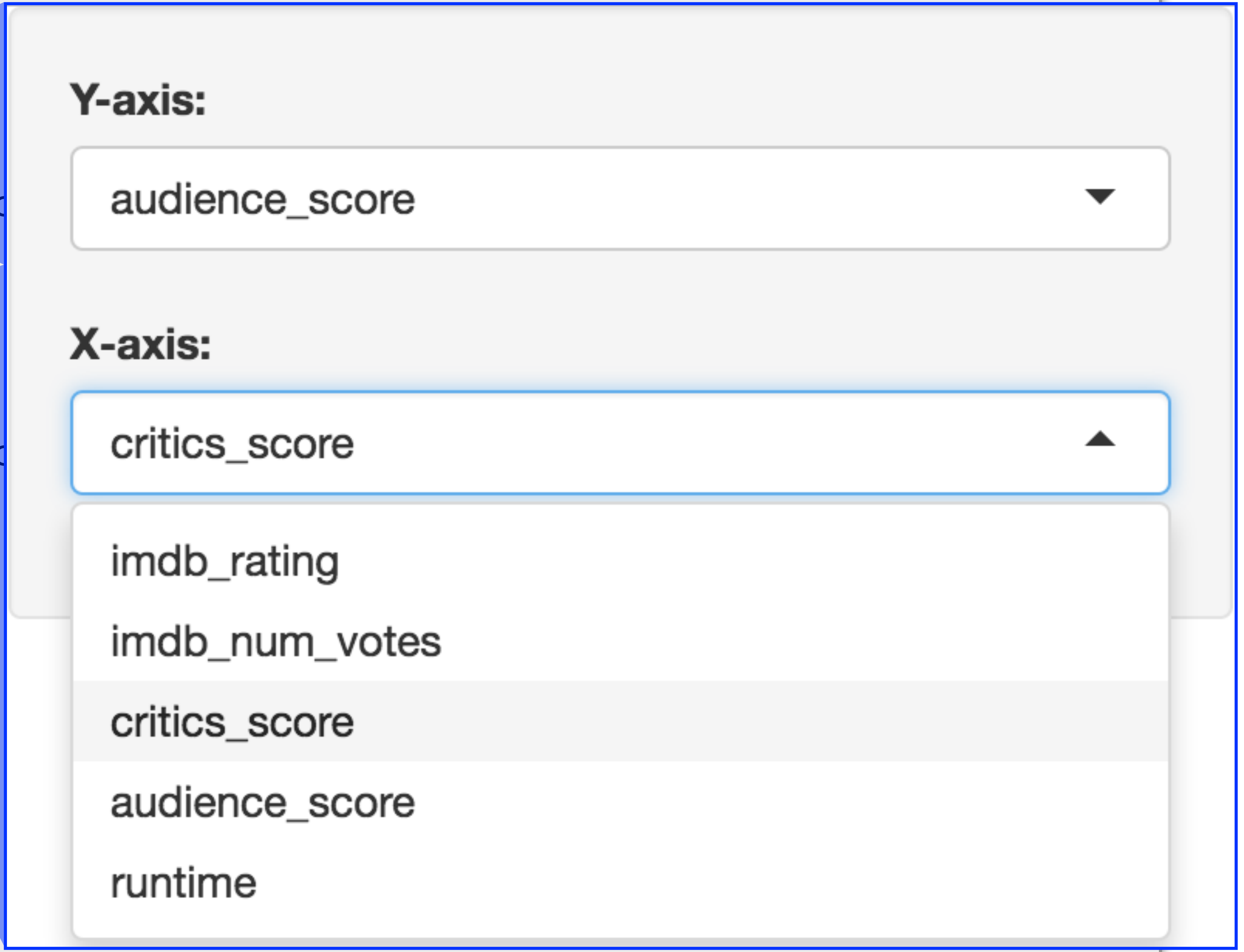
```r
# Define UI
ui <- fluidPage(

  # 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
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

```r
# Define UI
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")
    )
  )
)
```



Y-axis:

audience_score

X-axis:

critics_score

imdb_rating
imdb_num_votes
critics_score
audience_score
runtime

```r
# Define UI
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`

Anatomy of a Shiny app

Server

```r
# Define server function
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()
  })
}
```

```r
# Define server function
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
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

```r
# Define server function
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

Anatomy of a Shiny app

UI + Server

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

Putting it all together…

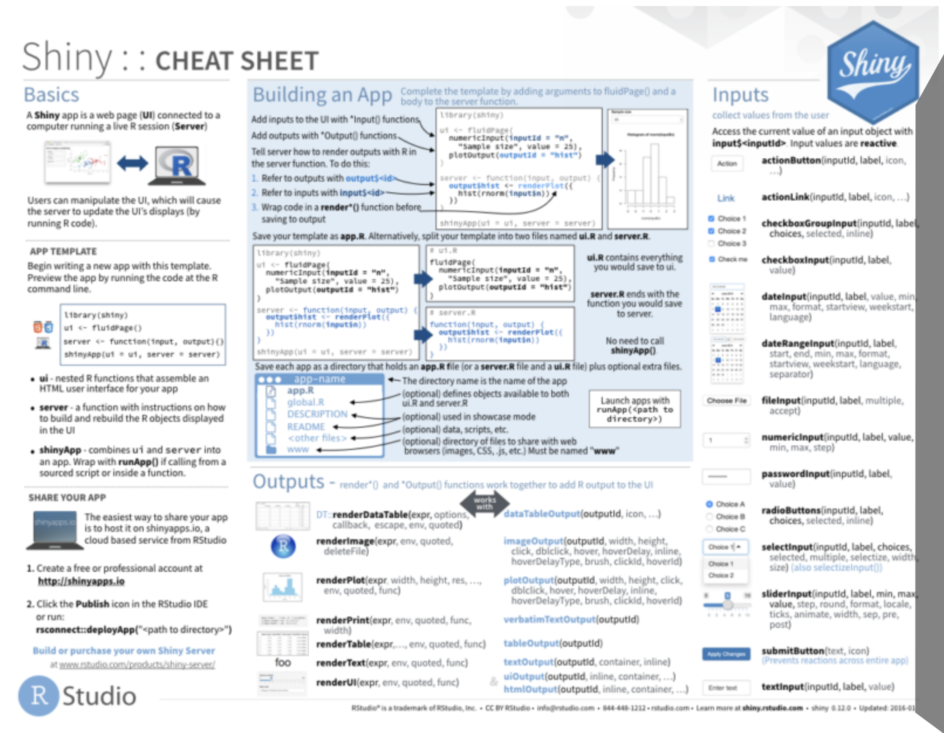
```
movies/movies-01.R
```

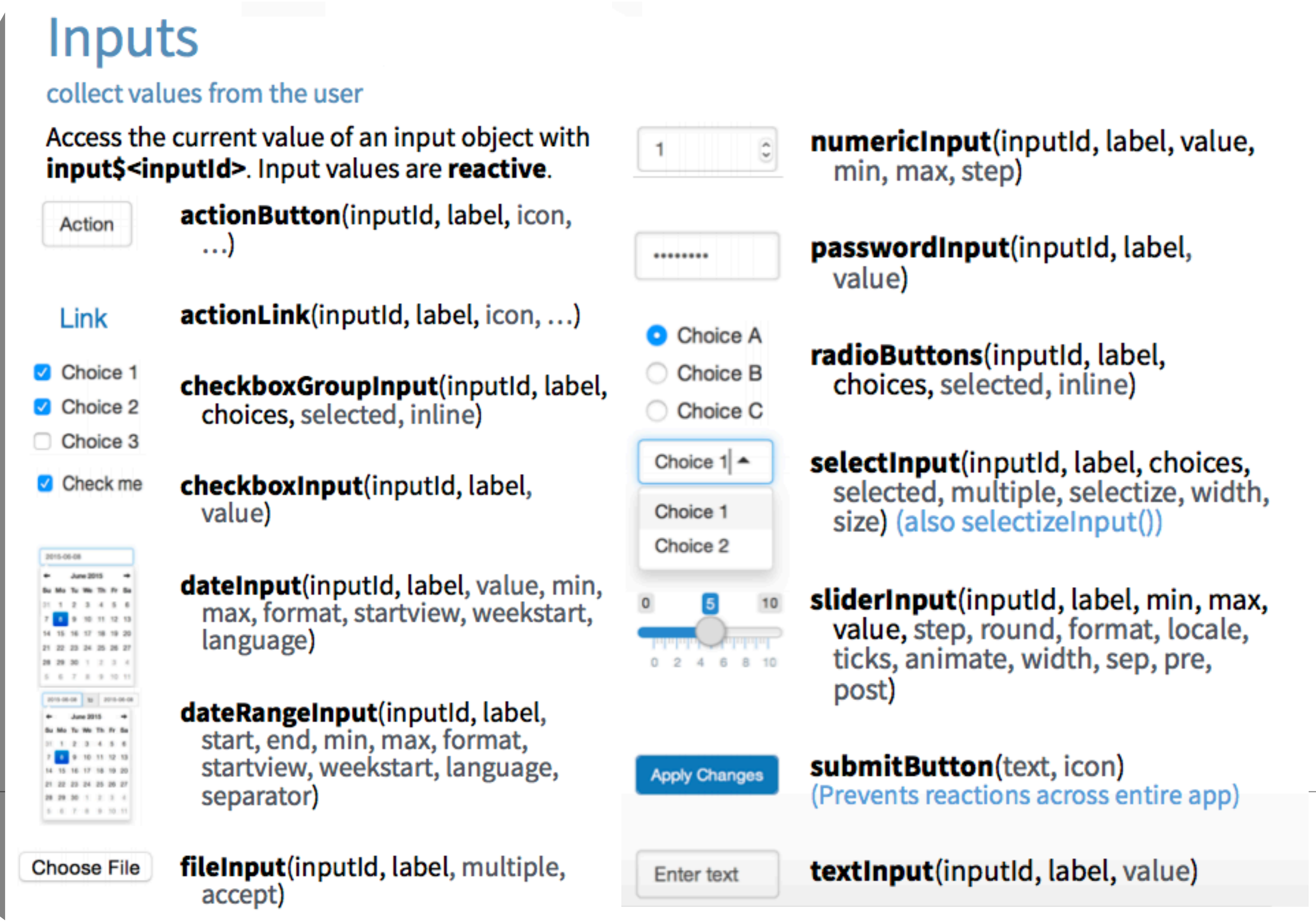
Add a `sliderInput` for alpha level of points on plot

`movies/movies-02.R`

Inputs

www.rstudio.com/resources/cheatsheets/





Inputs

collect values from the user

Access the current value of an input object with **input$<inputId>**. Input values are **reactive**.

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)

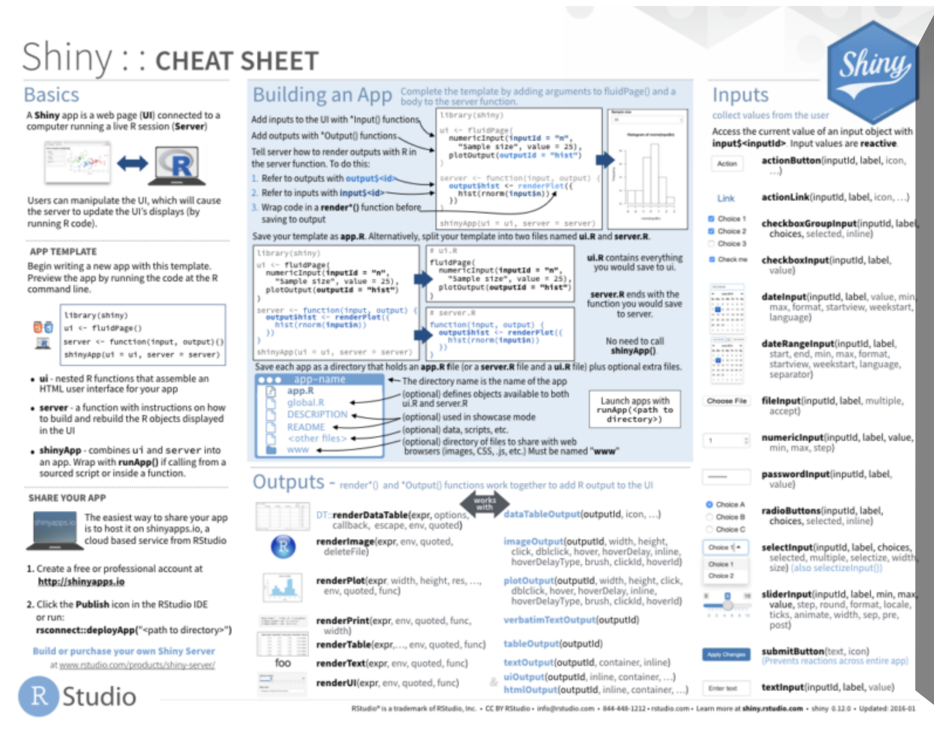
Add a new widget
to color the points by another variable

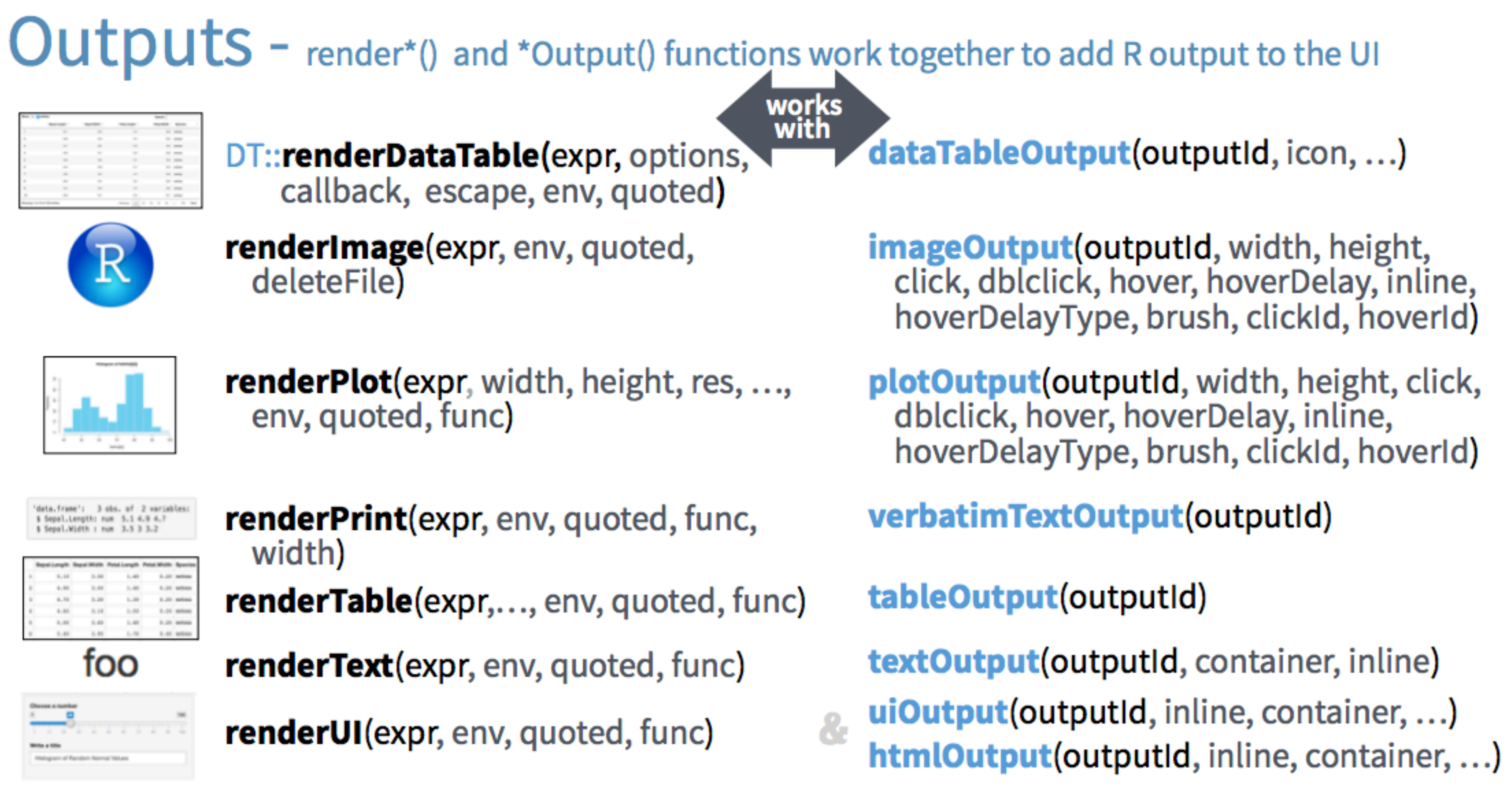
`movies/movies-03.R`

Display data frame *if* box is checked

`movies/movies-04.R`

Outputs





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

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, inline, hoverDelayType, brush, clickId, hoverId)
renderPlot(expr, width, height, res, ..., env, quoted, func)	**plotOutput**(outputId, width, height, click, dblclick, hover, hoverDelay, inline, hoverDelayType, brush, clickId, hoverId)
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

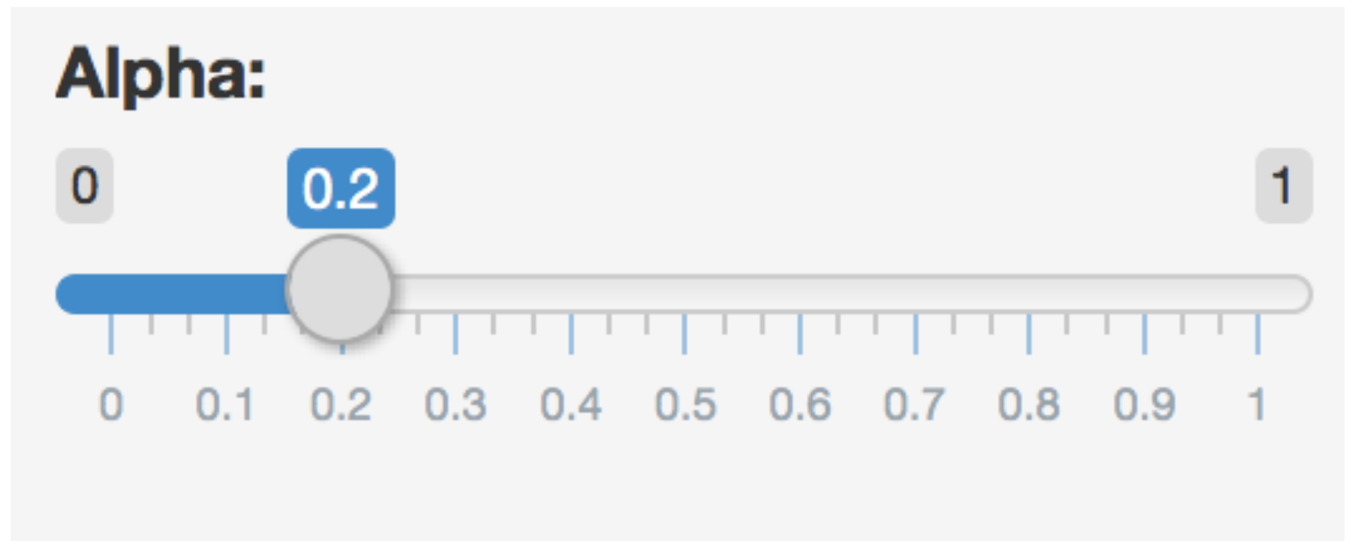
Reactivity 101

Reactions

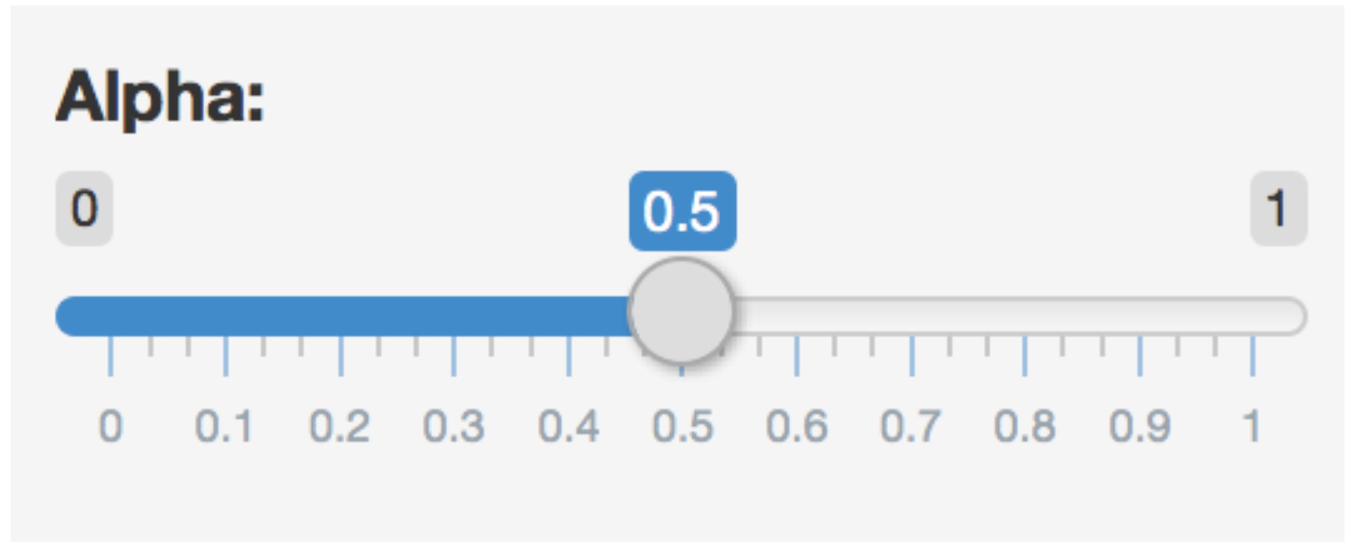
The **input$** list stores the current value of each input object under its name.

```
# Set alpha level
sliderInput(inputId = "alpha",
            label = "Alpha:",
            min = 0, max = 1,
            value = 0.5)
```

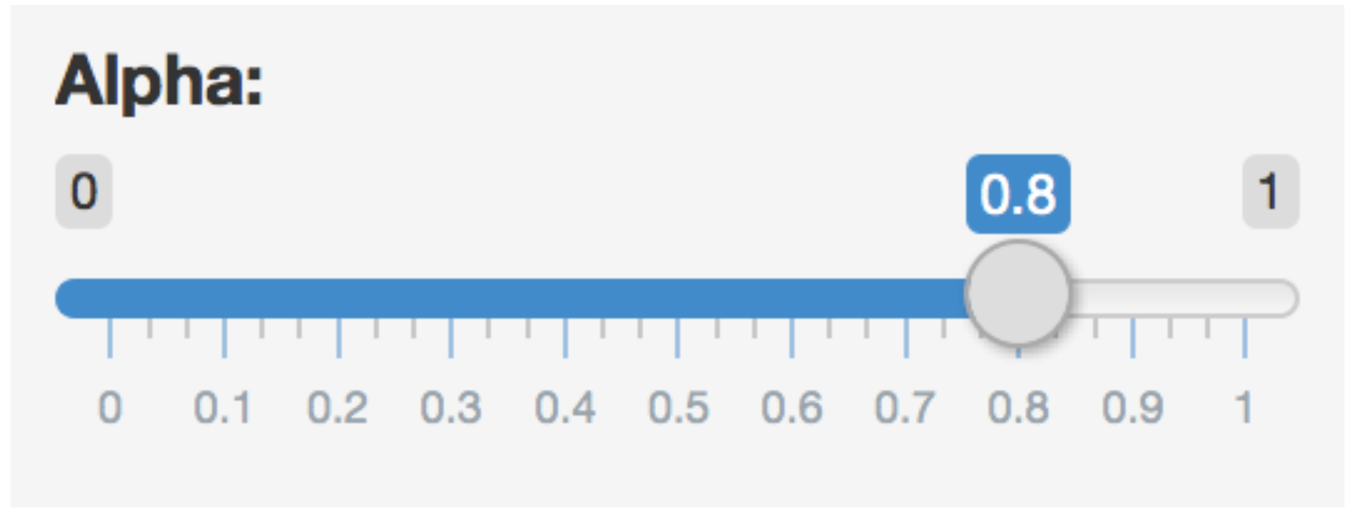
```
input$alpha
```



```
input$alpha = 0.2
```



```
input$alpha = 0.5
```



```
input$alpha = 0.8
```

Reactions (cont.)

Reactivity automatically occurs
when an **input** value is used to render an **output** object.

```
# 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$y,
                                     color = input$z)) +
      geom_point(alpha = input$alpha)
   )
}
```

Suppose you want the option to plot only certain types of movies as well as report how many such movies are plotted:

1. Add a UI element for the user to select which type(s) of movies they want to plot
2. Filter for chosen title type and save as a new (reactive) expression
3. Use new data frame (which is reactive) for plotting
4. Use new data frame (which is reactive) also for reporting number of observations

1. Add a UI element for the user to select which type(s) of movies they want to plot

```
# Select which types of movies to plot
checkboxGroupInput(inputId = "selected_type",
                   label = "Select movie type(s):",
                   choices = c("Documentary", "Feature Film",
                               "TV Movie"),
                   selected = "Feature Film")
```

2. Filter for chosen title type and save the new data frame as a reactive expression

server:

```
# Create a subset of data filtering fo
movies_subset <- reactive({
  req(input$selected_type)
  filter(movies, title_type %in% input$selected_type)
})
```

Creates a **cached expression** that knows it is out of date when input changes

3. Use new data frame (which is reactive) for plotting

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

Cached - only re-run when inputs change

4. Use new data frame (which is reactive) also for printing number of observations

ui:

```
mainPanel(
  …
  # Print number of obs plotted
  uiOutput(outputId = "n"),
  …
  )
```

server:

```
# Print number of movies plotted
output$n <- renderUI({
  types <- movies_subset()$title_type %>%
    factor(levels = input$selected_type)
  counts <- table(types)

  HTML(paste("There are",
             counts,
             input$selected_type,
             "movies in this dataset.
             <br>"))
})
```

Putting it all together…

`movies/movies-05.R`

5. `req()`
6. App title
7. `selectInput()` choice labels
8. Formatting of x and y axis labels
9. Visual separation with horizontal lines and breaks

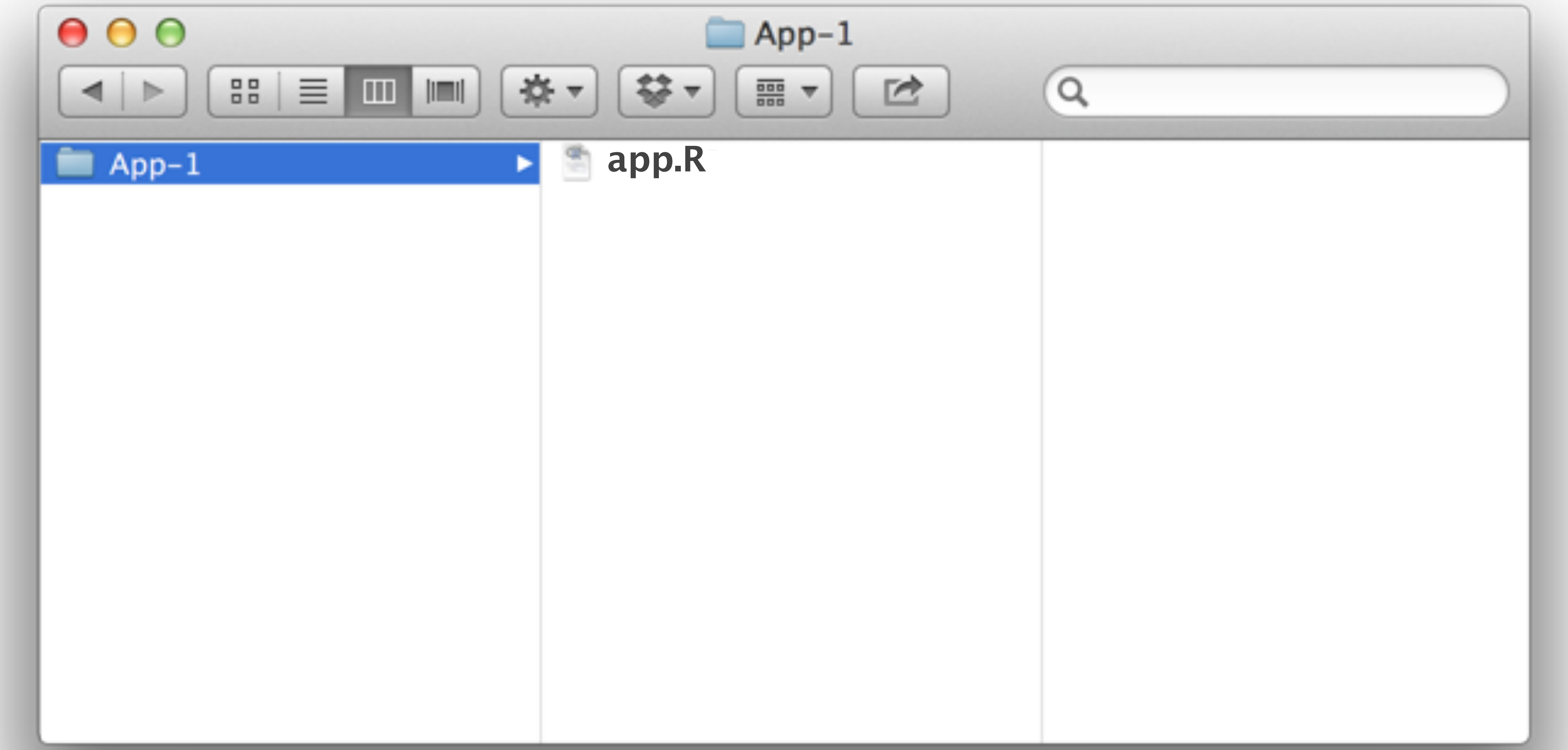
When to use reactive

- By using a reactive expression for the subsetted data frame, we were able to get away with subsetting once and then using the result twice.
- In general, reactive conductors let you
    - not repeat yourself (i.e. avoid copy-and-paste code, which is a maintenance boon), and
    - decompose large, complex (code-wise, not necessarily CPU-wise) calculations into smaller pieces to make them more understandable.
- These benefits are similar to what happens when you decompose a large complex R script into a series of small functions that build on each other.

File structure

File structure

- 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.


App-1
App-1
app.R