

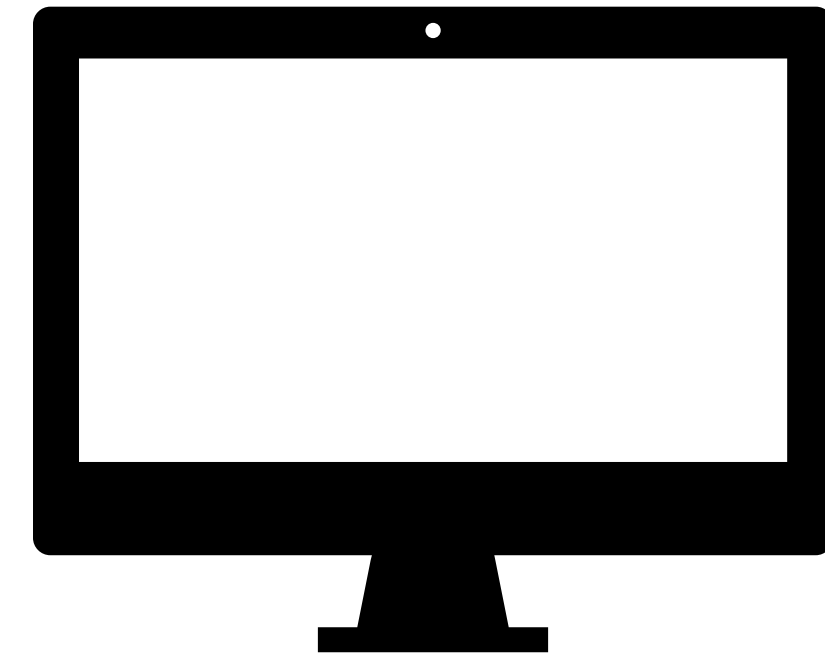
# Getting started with Shiny

Daniel. Kaplan



dtkaplan   
dtkaplan@gmail.com 

apps/goog-index/app.R



**DEMO**

# Your turn

- Open a new Shiny app with File → New File → Shiny Web App...
- Launch the app by opening `app.R` and clicking Run App
- Close the app by clicking the stop icon
- Select view mode in the drop down menu next to Run App

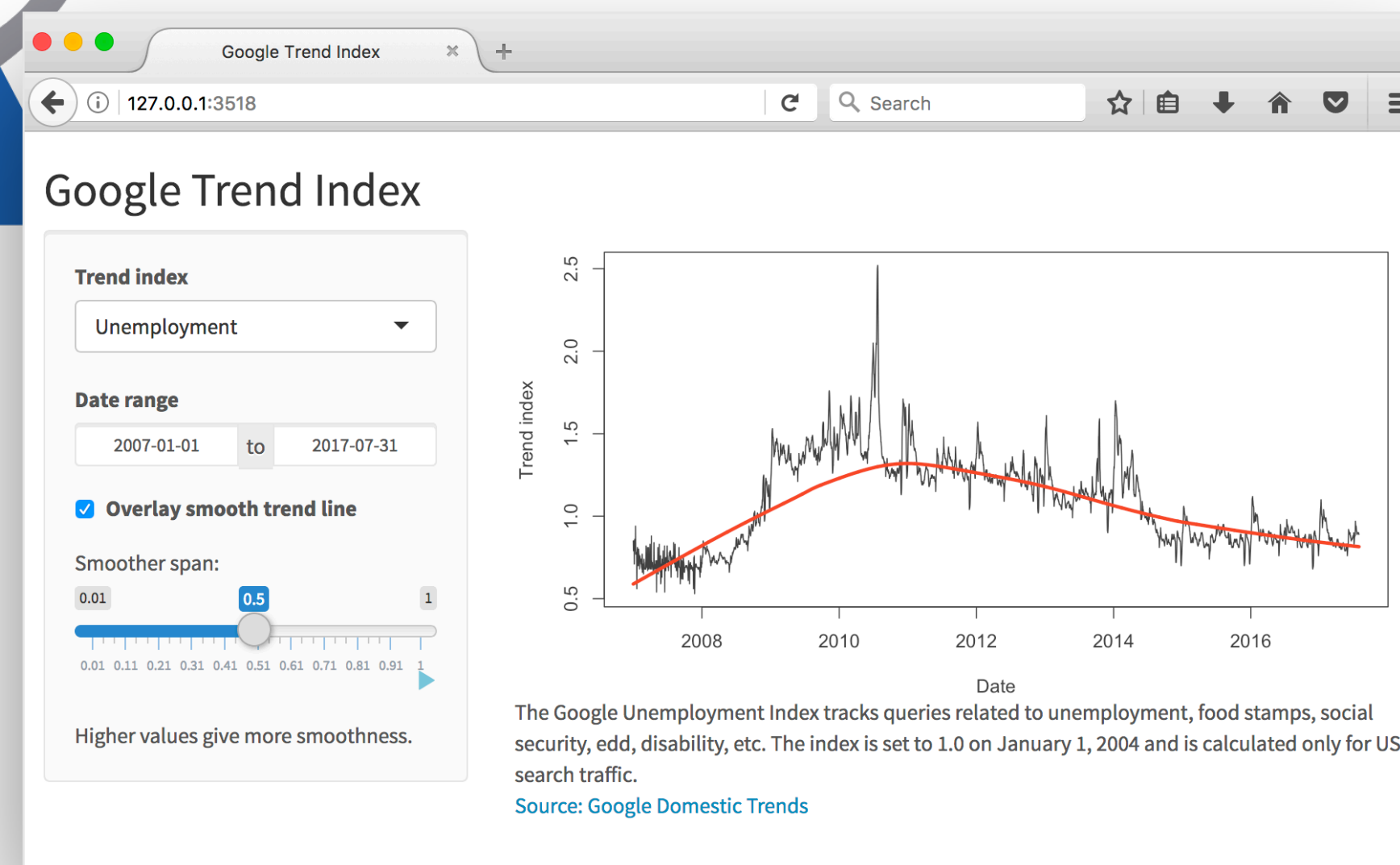


3<sub>m</sub> 00<sub>s</sub>

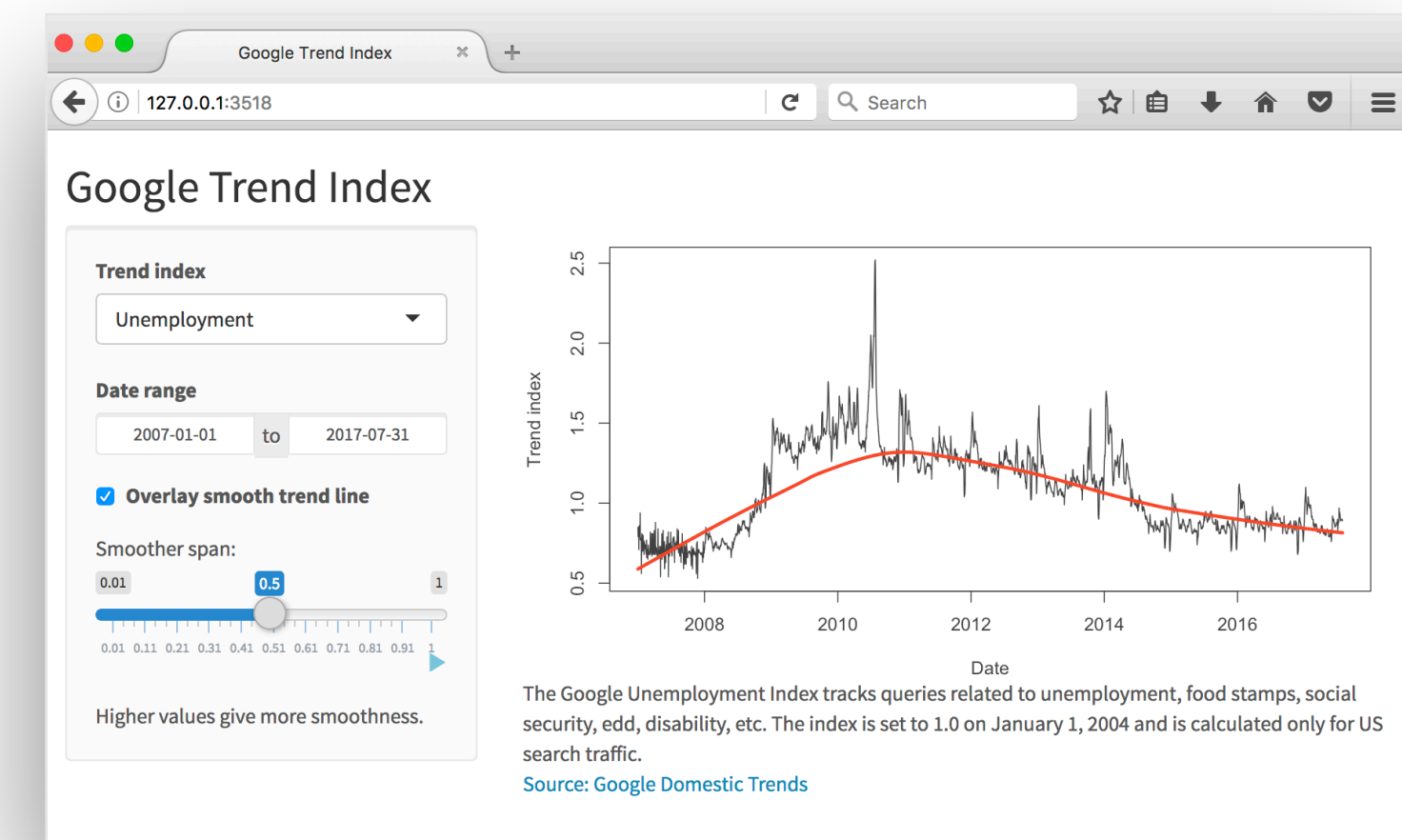


# High level view

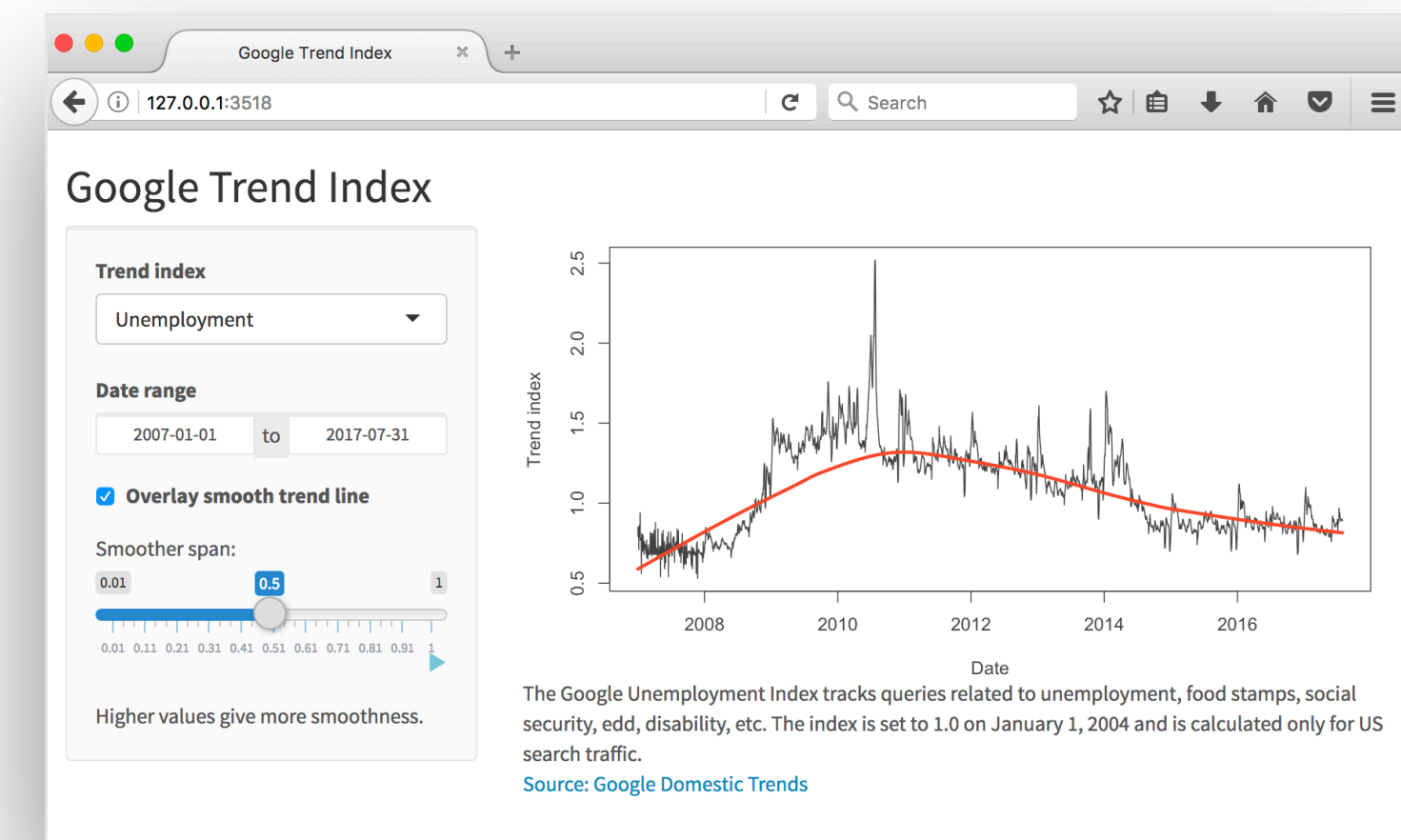
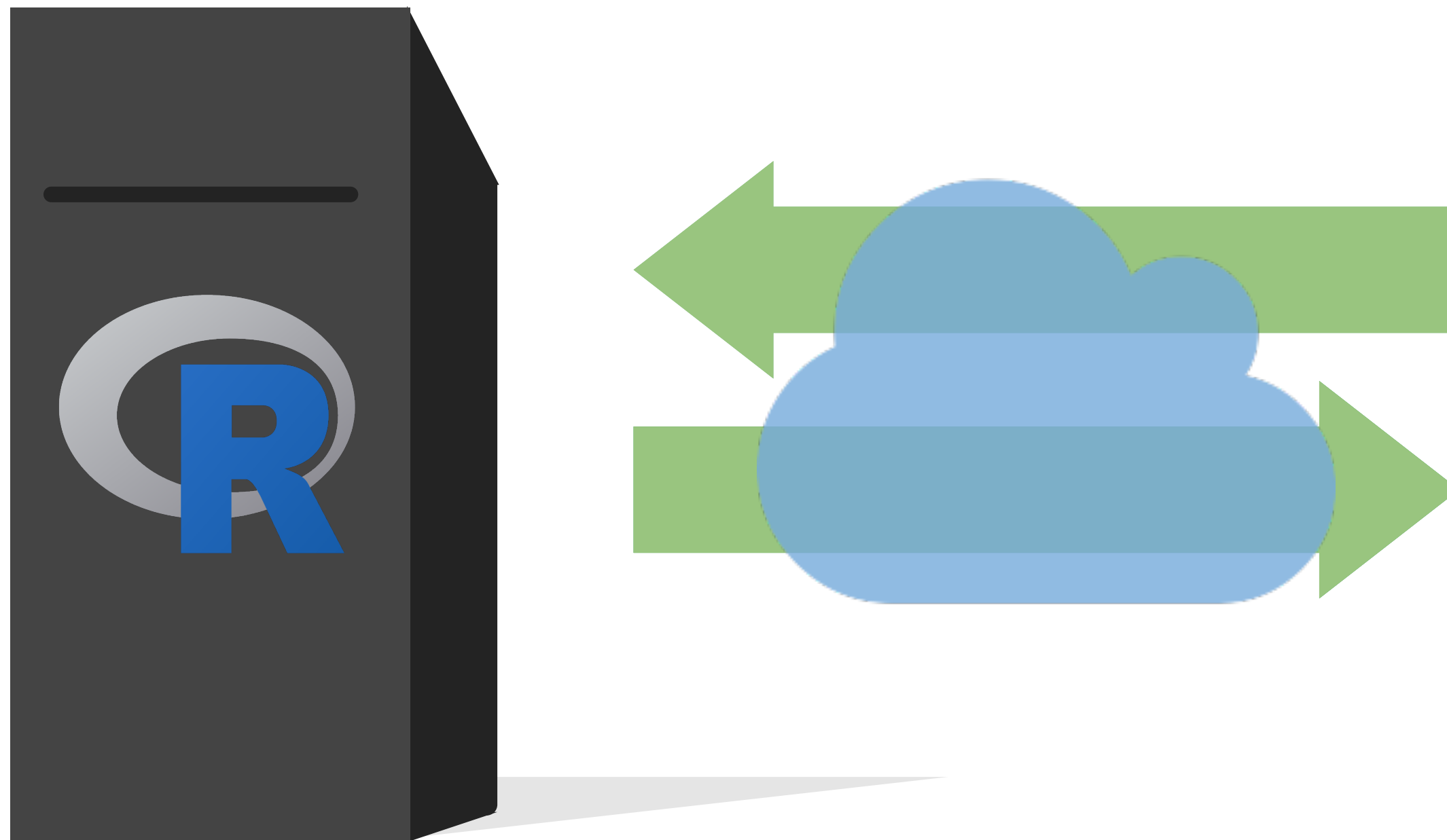
Every Shiny app has a webpage that the user visits,  
and behind this webpage there is a computer  
that serves this webpage by running R.

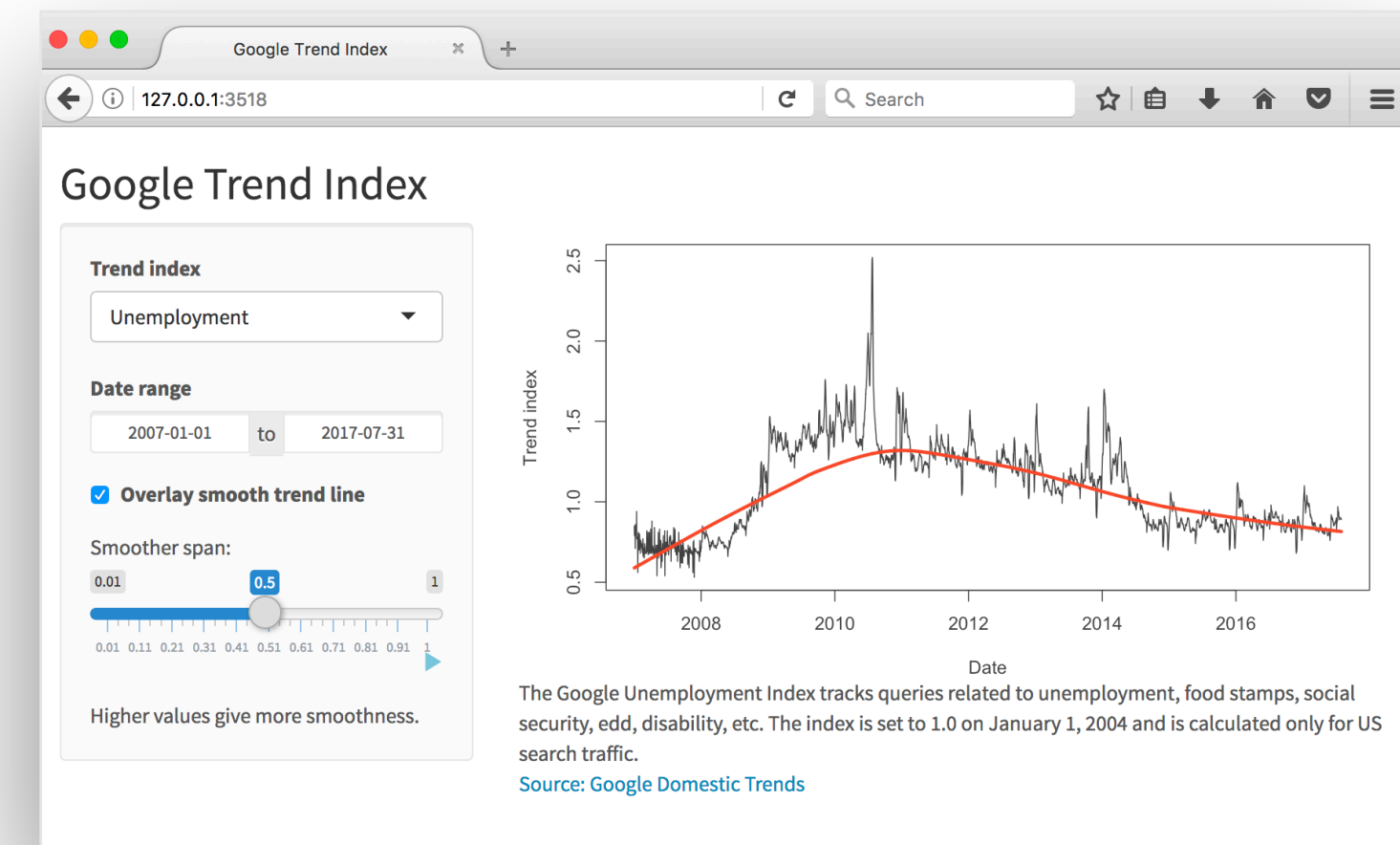
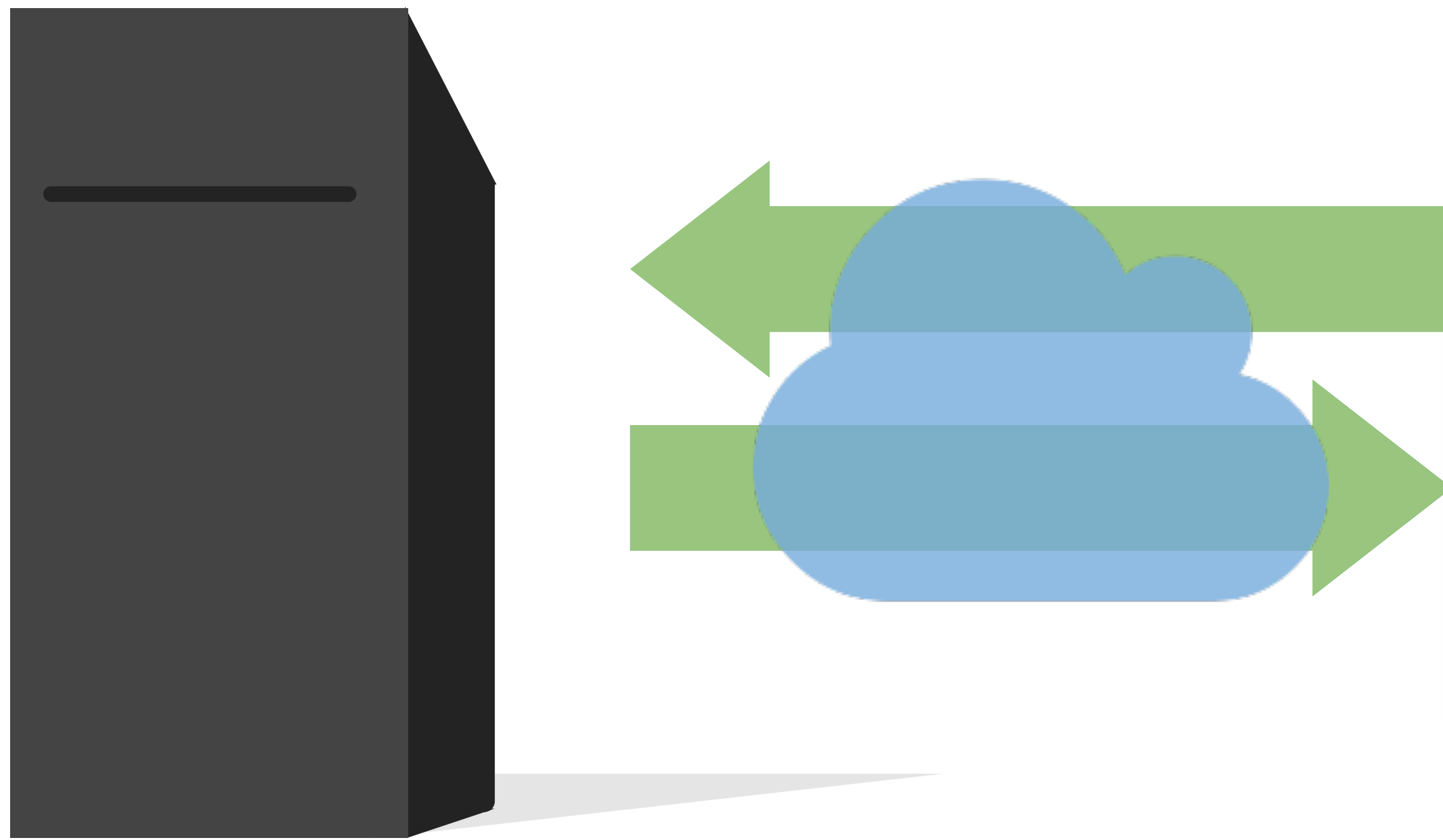


# When running your app locally, the computer serving your app is your computer.



# When your app is deployed, the computer serving your app is a web server.





Server instructions



User interface



# Anatomy of a Shiny app



# What's in an app?

```
library(shiny)
```

```
ui <- fluidPage()
```

## User interface

controls the layout and appearance of app

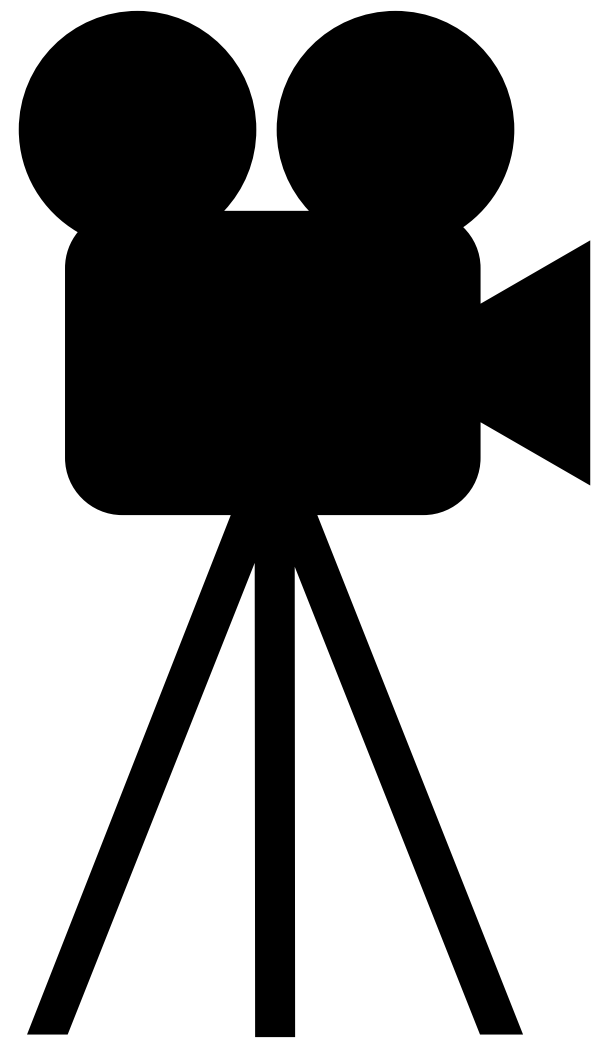
```
server <- function(input, output) {}
```

## Server function

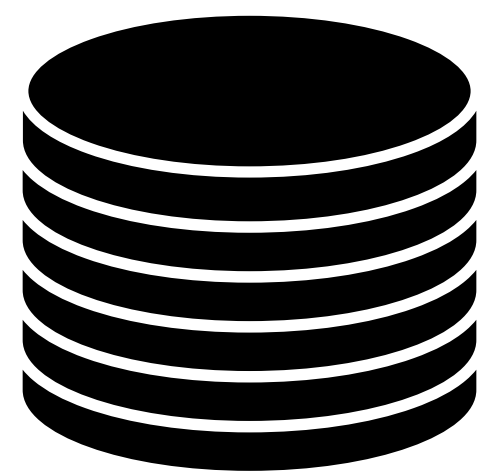
contains instructions needed to build app

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



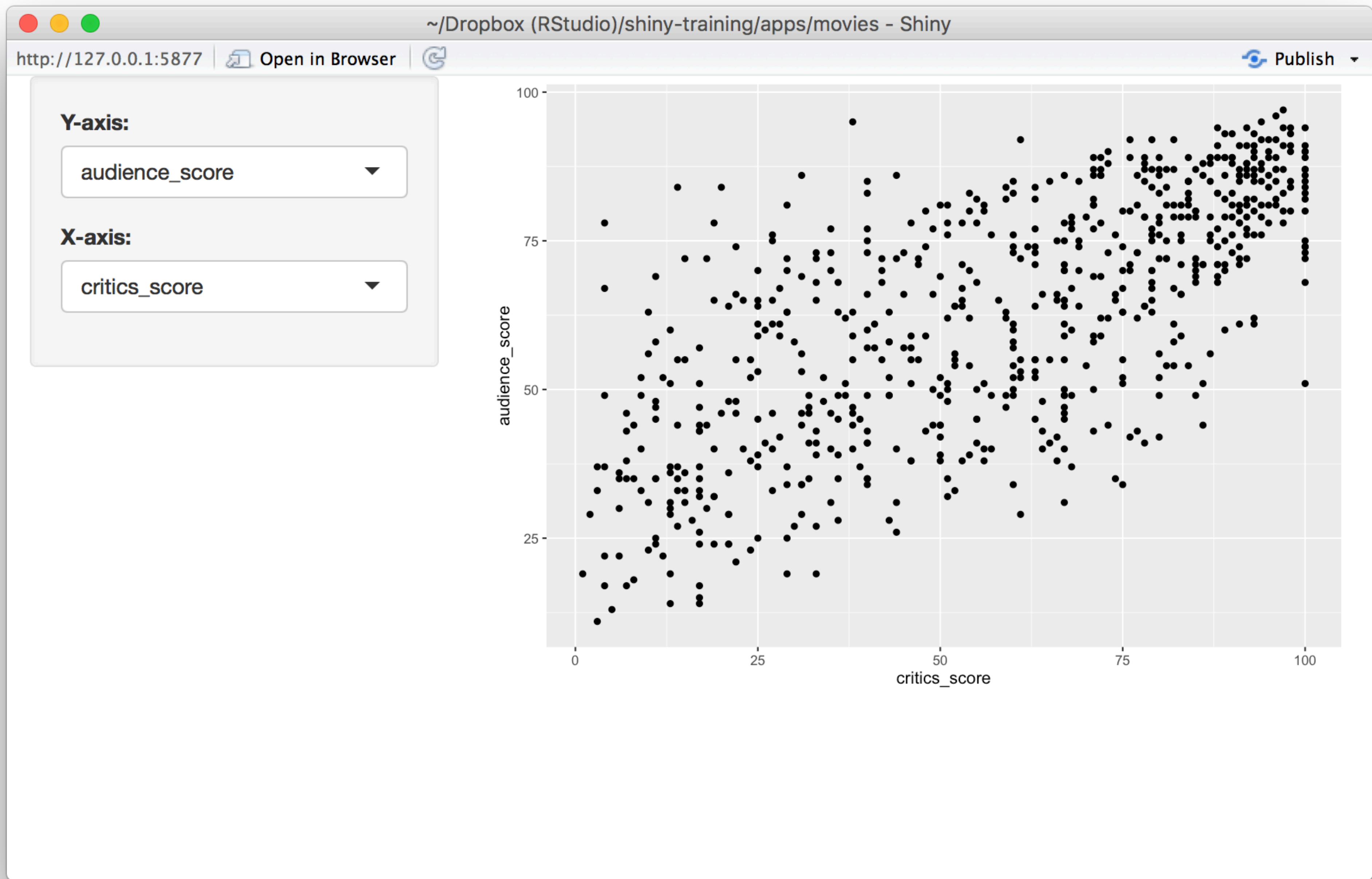


Let's build a simple movie browser app!



`data/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(tidyverse)
```

```
load("data/movies.Rdata")
```

```
ui <- fluidPage()
```

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

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



Dataset used for this 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")
    )
  )
)
```



Create fluid page layout

# 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")
```

```
    )
```

```
  )
```

```
)
```





# Define UI

ui <- fluidPage(  
 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

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
```

```
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")
```

```
  )
```

```
)
```

```
)
```

**Y-axis:**

audience\_score

**X-axis:**

critics\_score

imdb\_rating

imdb\_num\_votes

critics\_score

audience\_score

runtime



# 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

# Server



```
# 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()
  })
}
```



# Define server function

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

# 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, session) {
```

```
# Create the scatterplot object the plotOutput fun
```

```
output$scatterplot <- renderPlot({
```

```
  ggplot(data = movies, aes_string(x = input$x, y
```

```
    geom_point()
```

```
})
```

```
}
```

Renders a **reactive** plot that is suitable for assigning to an output slot





# Define server function

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

# 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



# UI + Server

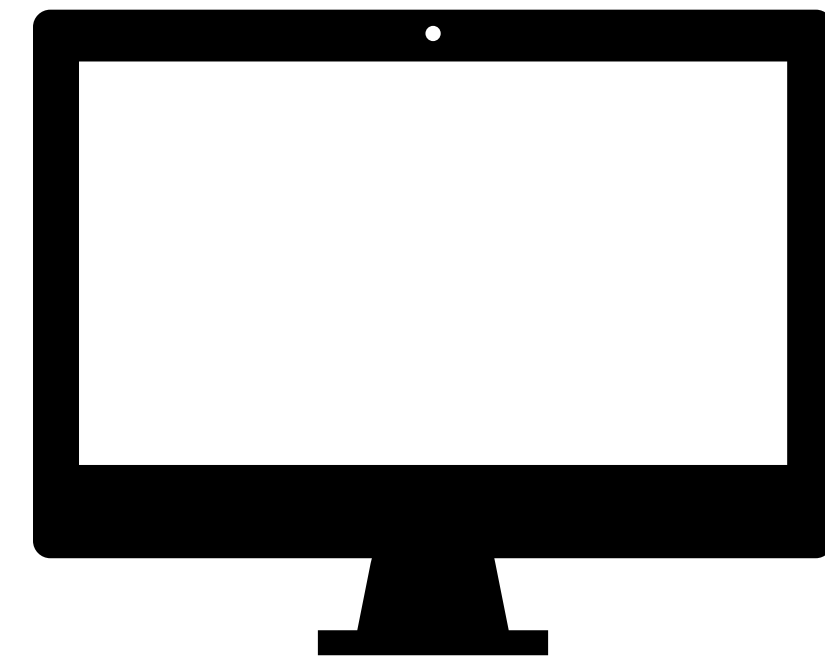


```
# Create the Shiny app object  
shinyApp(ui = ui, server = server)
```



Putting it all together...

`apps/movies/movies-01.R`



**DEMO**

# Your turn

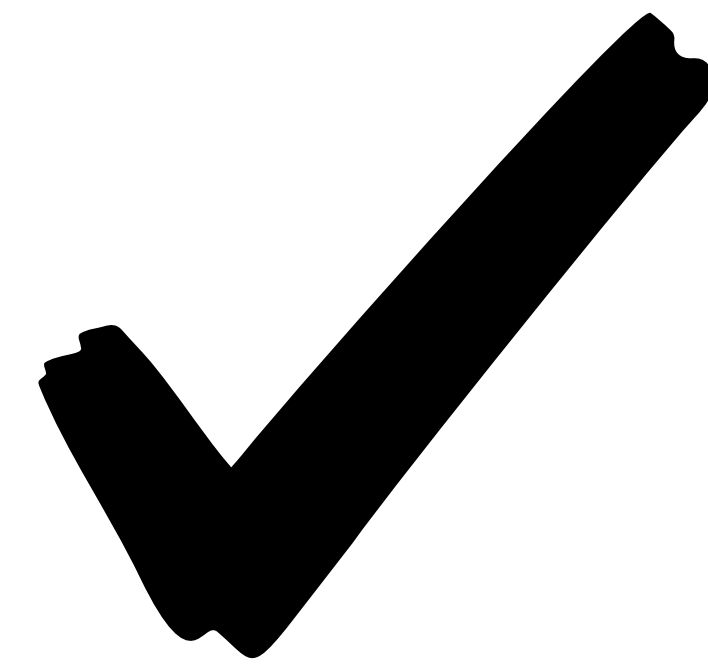
- Add new select menu to color the points by
  - `inputId = "z"`
  - `label = "Color by:"`
  - `choices = c("title_type", "genre", "mpaa_rating", "critics_rating", "audience_rating")`
  - `selected = "mpaa_rating"`
- Use this variable in the aesthetics of the `ggplot` function as the color argument to color the points by
- Run the app in the Viewer Pane
- Compare your code / output with the person sitting next to / nearby you



5<sub>m</sub> 00<sub>s</sub>



Solution to the previous exercise



apps/movies/movies-02.R

**SOLUTION**





# Inputs

## Shiny : : CHEAT SHEET

### Basics

A **Shiny** app is a web page (**UI**) connected to a computer running a live R session (**Server**)

Users can manipulate the UI, which will cause the server to update the UI's displays (by running R code).

### APP TEMPLATE

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

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

• **ui** - nested R functions that assemble an HTML user interface for your app

• **server** - a function with instructions on how to build and rebuild the R objects displayed in the UI

• **shinyApp** - combines **ui** and **server** into an app. Wrap with **runApp()** if calling from a sourced script or inside a function.

### SHARE YOUR APP

The easiest way to share your app is to host it on shinyapps.io, a cloud based service from RStudio

1. Create a free or professional account at <https://shinyapps.io>

2. Click the **Publish** icon in the RStudio IDE or run:

```
rsconnect::deployApp("~/path to directory")
```

**Build or purchase your own Shiny Server** at [www.rstudio.com/products/shiny-server/](https://www.rstudio.com/products/shiny-server/)

R Studio

### Building an App

Complete the template by adding arguments to **fluidPage()** and a body to the **server** function.

Add inputs to the UI with "Input" functions. Add outputs with "Output" functions. Tell server how to render outputs with R in the server function. To do this:

1. Refer to outputs with **output\$** id.
2. Refer to inputs with **input\$id**.
3. Wrap code in a **render()** function before saving to output.

Save your template as **app.R**. Alternatively, split your template into two files named **ui.R** and **server.R**.

```
library(shiny)
ui <- fluidPage(
  numericInput(inputId = "n",
    "Sample size", value = 25),
  plotOutput(outputId = "m1st")
)
server <- function(input, output) {
  output$m1st <- renderPlot({
    hist(rnorm(input$n))
  })
  shinyApp(ui = ui, server = server)
}
```

Save each app as a directory that holds an **app.R** file (or a **server.R** file and a **ui.R** file) plus optional extra files.

• **app.R** - The directory name is the name of the app

• **global.R** - (optional) defines objects available to both ui.R and server.R

• **DESCRIPTION** - (optional) used in showcase mode

• **other files** - (optional) data, scripts, etc.

• **www** - (optional) directory of files to share with web browsers (images, CSS, JS, etc.). Must be named "www"

• **renderDataTable**(expr, optional, callback, escape, env, quoted, deleteFid)

• **renderImage**(expr, env, quoted, deleteFid)

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

• **renderPrint**(expr, env, quoted, func, width)

• **renderTable**(expr, ..., env, quoted, func)

• **renderText**(expr, env, quoted, func)

• **renderUI**(expr, env, quoted, func)

### Inputs

collect values from the user. Access the current value of an input object with **input\$** - inputs 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)

**textOutput**(outputId, inline, container, ...)

**htmlOutput**(outputId, inline, container, ...)

**tableOutput**(outputId)

**uiOutput**(outputId, inline, container, ...)

**enterText**

Action

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

Link

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

☒ Choice 1

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

☒ Choice 2

☐ Choice 3

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

☒ Check me

**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)

1

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

.....

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

☒ Choice A

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

☐ Choice B

☐ Choice C

Choice 1 | ▲

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

Choice 1

Choice 2

0 5 10

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

0 2 4 6 8 10

Apply Changes

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

Enter text

**textInput**(inputId, label, value)



# Your turn

- Add new input variable to control the alpha level of the points
  - This should be a `sliderInput`
    - See [shiny.rstudio.com/reference/shiny/latest/](https://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

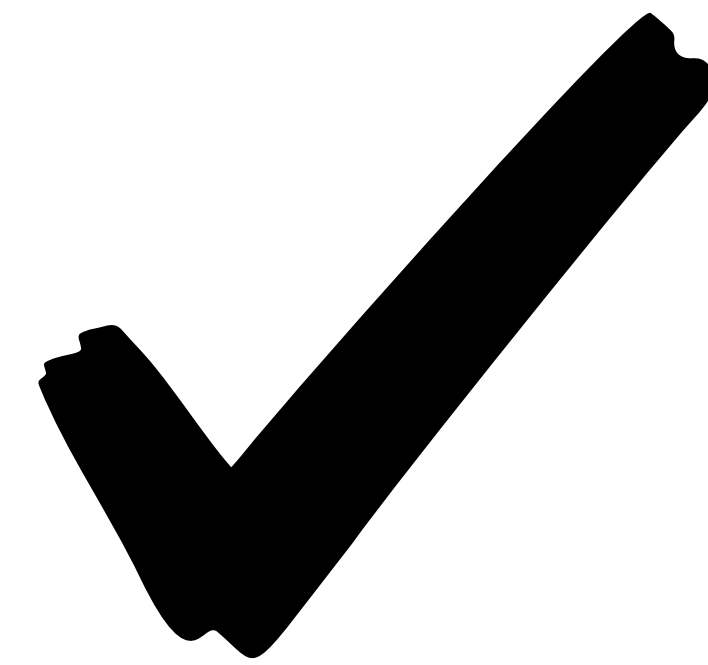


5<sub>m</sub> 00<sub>s</sub>





Solution to the previous exercise



apps/movies/movies-03.R

**SOLUTION**



# Outputs

## Shiny : : CHEAT SHEET

### Basics

A **Shiny** app is a web page (**UI**) connected to a computer running a live R session (**Server**)



Users can manipulate the UI, which will cause the server to update the UI's displays (by running R code).

### APP TEMPLATE

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

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

- **ui** - nested R functions that assemble an HTML user interface for your app

- **server** - a function with instructions on how to build and rebuild the R objects displayed in the UI

- **shinyApp** - combines **ui** and **server** into an app. Wrap with **runApp()** if calling from a sourced script or inside a function.

### SHARE YOUR APP

The easiest way to share your app is to host it on shinyapps.io, a cloud based service from RStudio

1. Create a free or professional account at <https://shinyapps.io>
2. Click the **Publish** icon in the RStudio IDE or run:  
`rsconnect::deployApp()` ("path to directory")

Build or purchase your own Shiny Server at [www.rstudio.com/products/shiny-server/](https://www.rstudio.com/products/shiny-server/)

R Studio



**Building an App** Complete the template by adding arguments to `fluidPage()` and a body to the `server` function

Add inputs to the UI with "Input" functions  
Add outputs with "Output" functions  
Tell server how to render outputs with R in the server function. To do this:

1. Refer to outputs with `output$`
2. Refer to inputs with `input$`
3. Wrap code in a `render()` function before saving to output

Save your template as **app.R**. Alternatively, split your template into two files named **ui.R** and **server.R**.

**ui.R** contains everything you would save to `ui`.

**server.R** ends with the function you would save to `server`.

No need to call `shinyApp()`

Save each app as a directory that holds an **app.R** file (or a **server.R** file and a **ui.R** file) plus optional extra files.

- **app.R** (optional) defines objects available to both **ui.R** and **server.R**
- **global.R** (optional) used in showcase mode
- **DESCRIPTION** (optional) data, scripts, etc
- **<other files>** (optional) directory of files to share with web browsers (images, CSS, js, etc). Must be named "www"

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

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

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

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

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

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

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

**renderUI**(expr, env, quoted, func)

**dataTableOutput**(outputId, icon, ...)

**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)

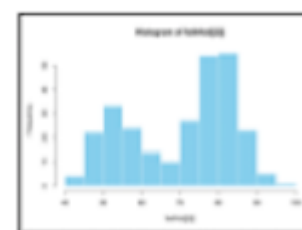
**verbatimTextOutput**(outputId)

**tableOutput**(outputId)

**textOutput**(outputId, container, inline)

**uiOutput**(outputId, inline, container, ...)

**htmlOutput**(outputId, inline, container, ...)



'data.frame': 3 obs. of 2 variables:  
 \$ Sepal.Length: num 5.1 4.9 4.7  
 \$ Sepal.Width : num 3.5 3 3.2

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.6	1.7	0.2	setosa

foo



**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, ...)

# Your turn

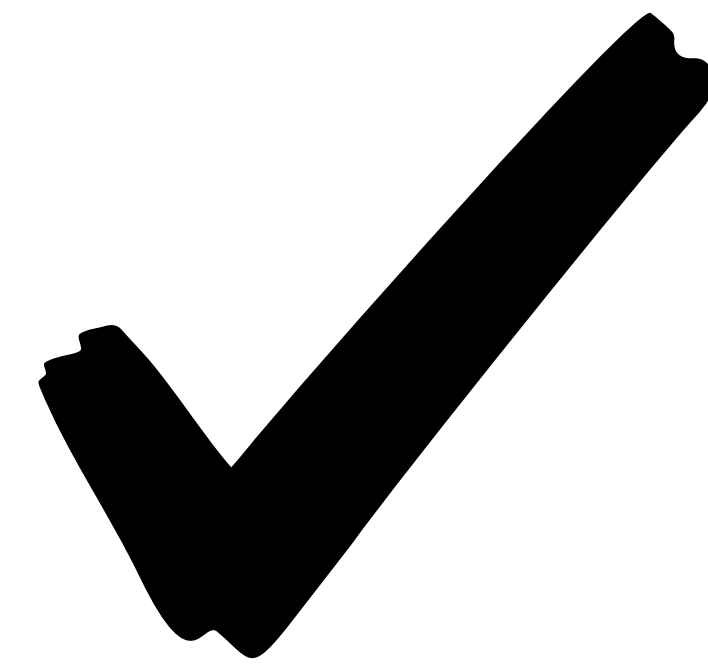
- Create a new output item using `DT::renderDataTable`.
- Show first seven columns of movies data, show 10 rows at a time, and hide row names, e.g.
  - `data = movies[, 1:7]`
  - `options = list(pageLength = 10)`
  - `rownames = FALSE`
- Add a `DT::dataTableOutput` to the main panel
- Run the app in a new Window
- Compare your code / output with the person sitting next to / nearby you
- **Stretch goal:** Make the number of columns visible in the table a user defined input



5<sub>m</sub> 00<sub>s</sub>



Solution to the previous exercise



apps/movies/movies-04.R

**SOLUTION**



# Your turn

- 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

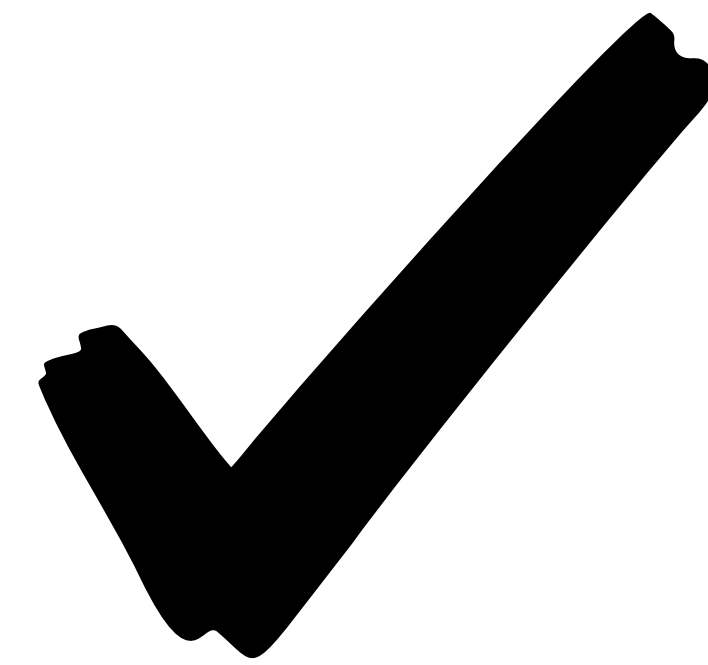


10<sub>m</sub> 00<sub>s</sub>





Solution to the previous exercise



apps/movies/movies-05.R

**SOLUTION**

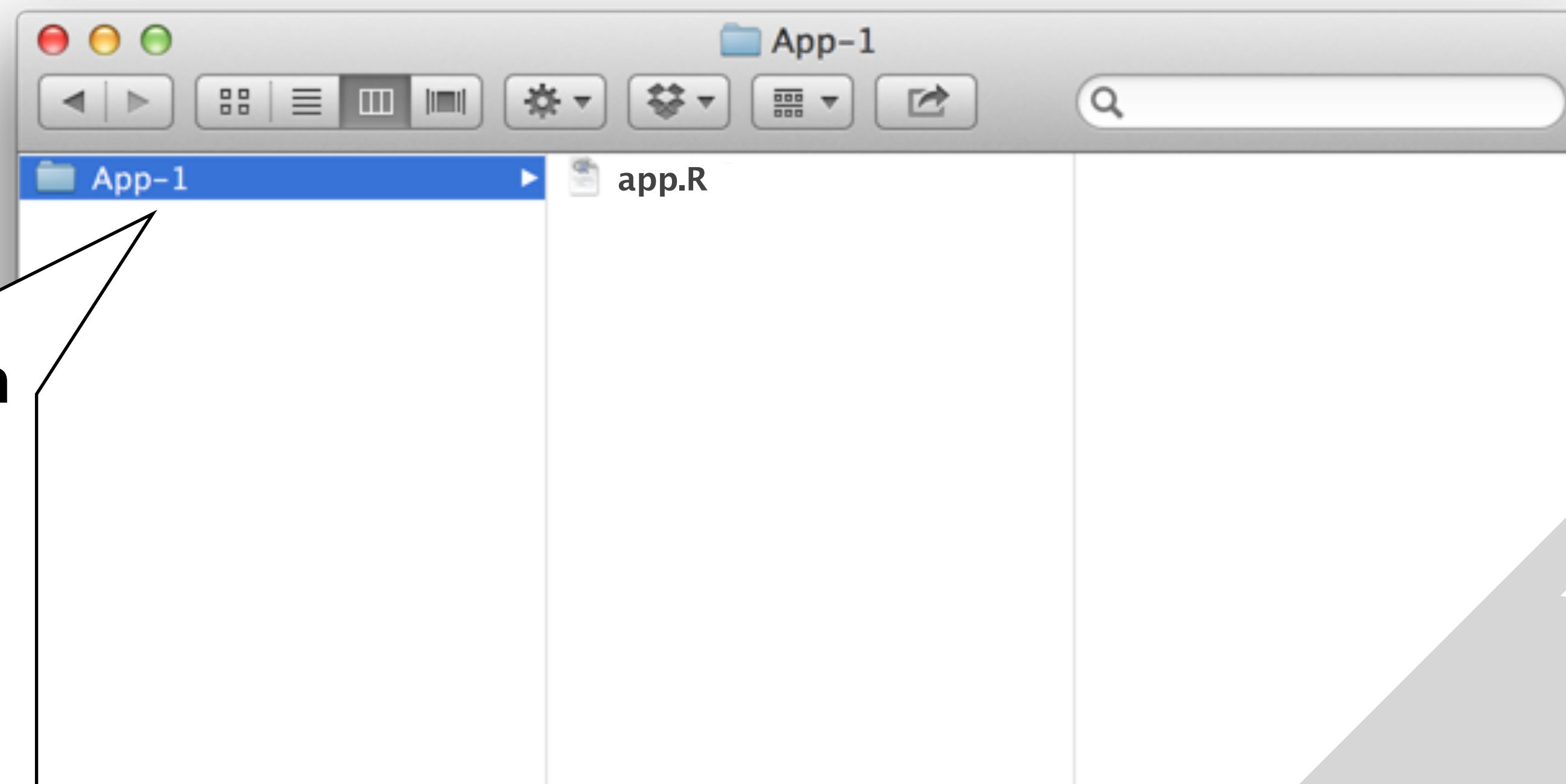


# File structure



# Single file

- 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**

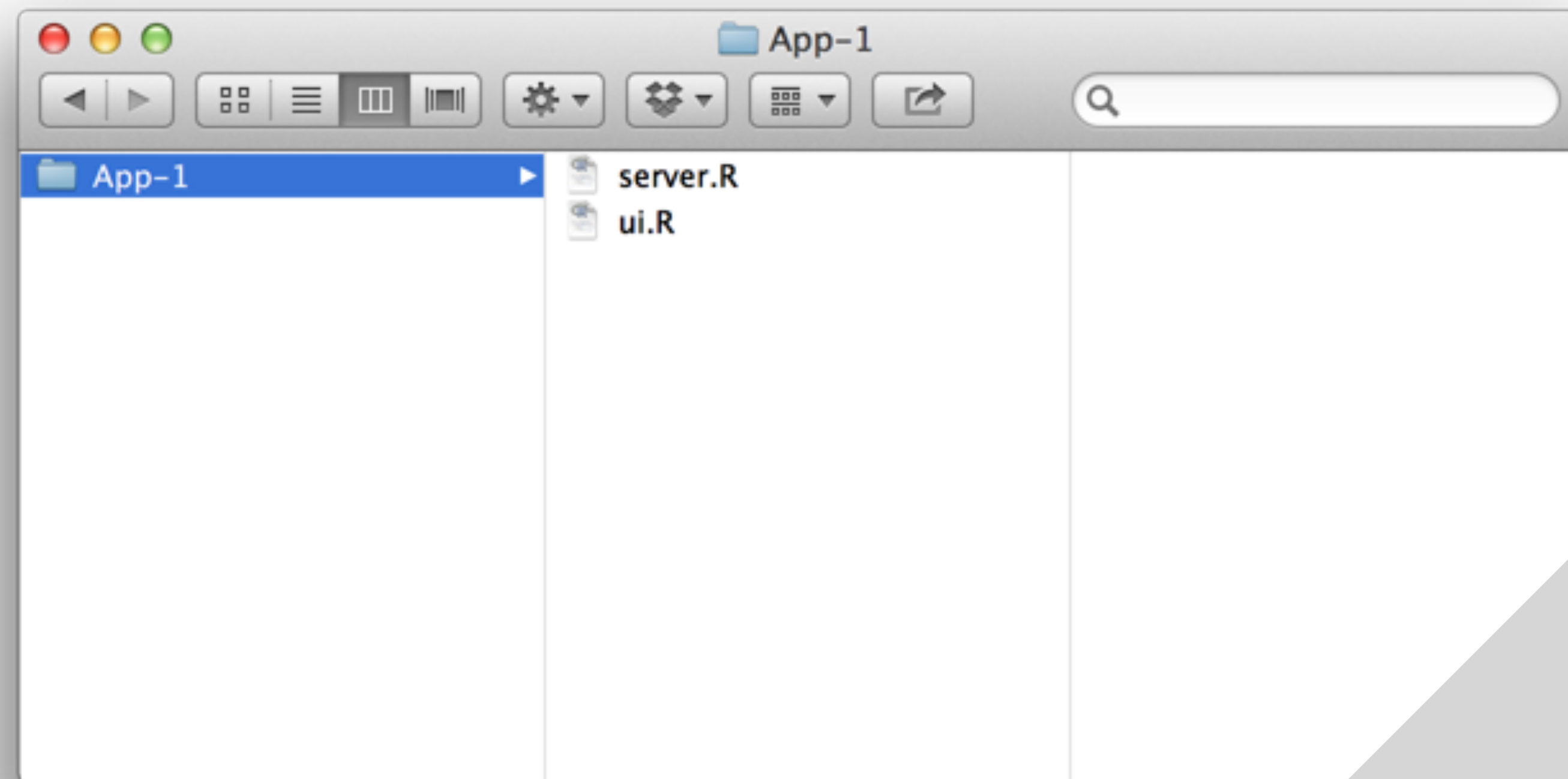


*You must use this  
exact name (**app.R**)  
for deploying the app*



# Multiple files

- 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

# Deploying your app



# shinyapps.io

- A server maintained by RStudio
- Easy to use, secure, and scalable
- Built-in metrics
- Free tier available



# Shiny Server

- Free and open source
- 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



# Shiny Server Pro / RStudio Connect

- Secure access and authentication
- Performance: fine tune at app and server level
- Management: monitor and control resource use
- Direct priority support



# Your turn

- Create a folder called movies
- Move any one of the movies app R scripts you worked on into this folder, and rename it as `app.R`
- Also move the `movies.Rdata` file into this folder
- Run the app
- Go to [shinyapps.io](https://shinyapps.io) and create a free account. Follow the instructions and deploy your first app.



10<sub>m</sub> 00<sub>s</sub>

