Structuring

information delivery and activities

teach-shiny.rbind.io

Mine Çetinkaya-Rundel

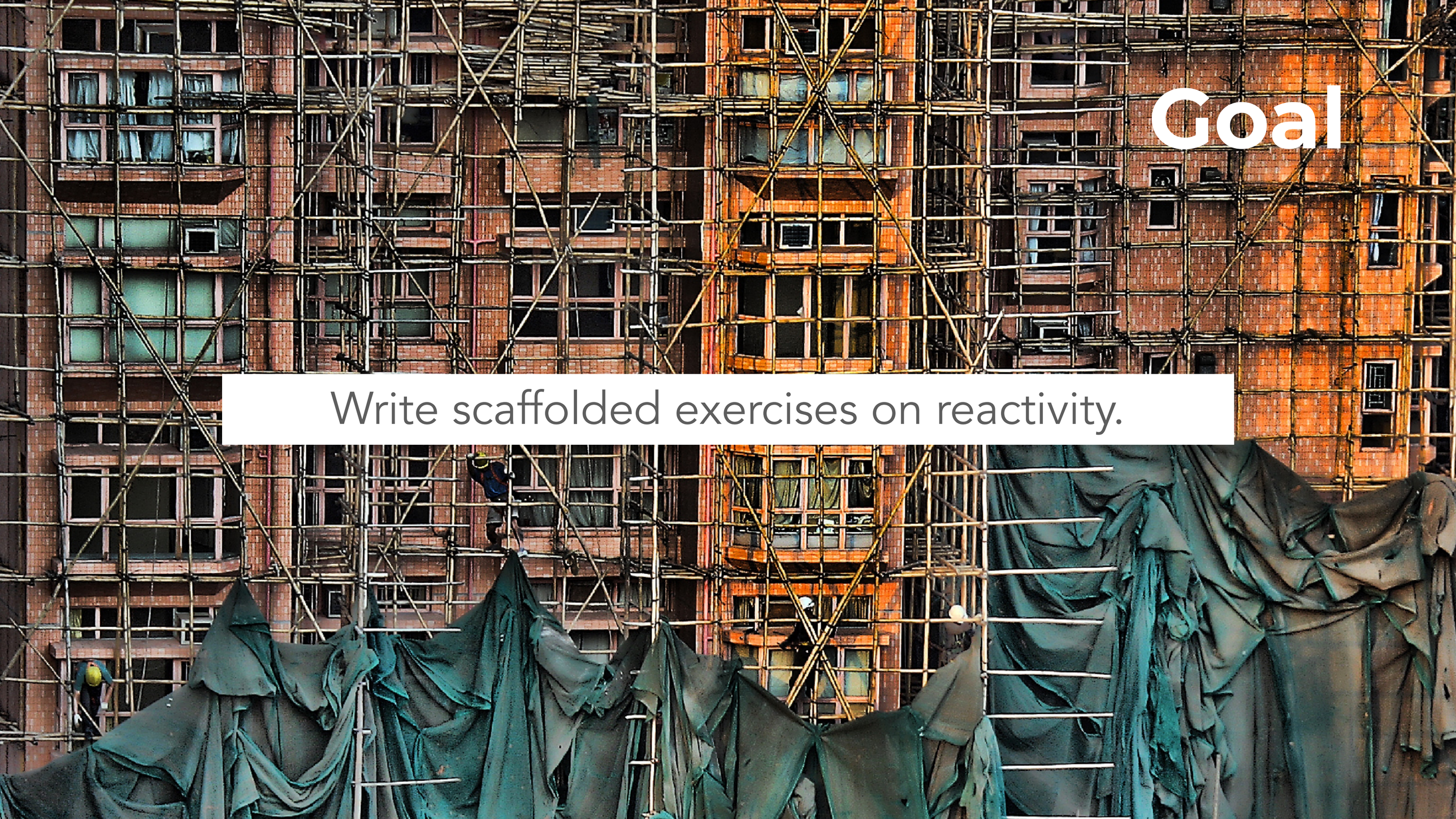
@minebocek

mine-cetinkaya-rundel

mine@rstudio.com

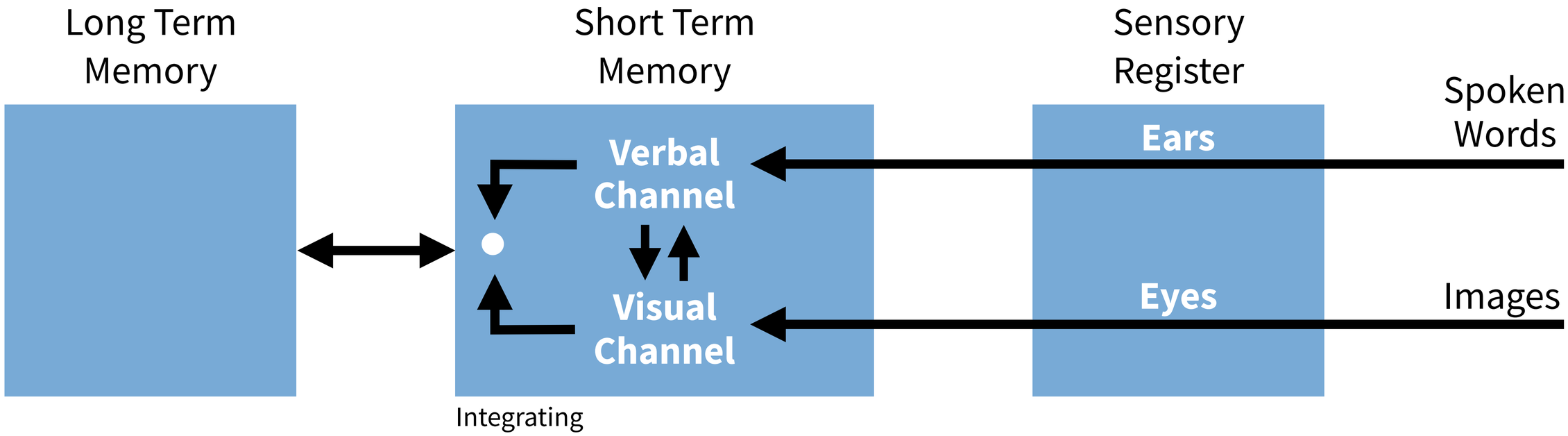
Goal

Write scaffolded exercises on reactivity.



use visual cues

Dual coding theory



Adapted from Mayer, R. E. (2002). Multimedia learning. Psychology of learning and motivation, 41, 85-139. Chicago

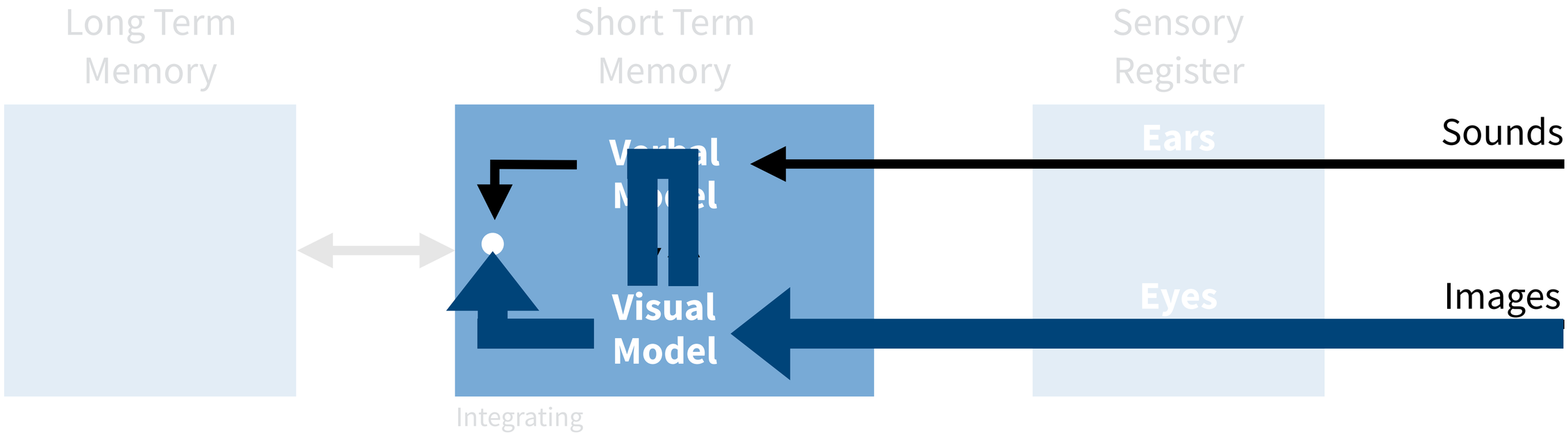
After you get your Shiny app to a state where it works, it's often useful to have an automated system that checks that it continues to work as expected. There are many possible reasons for an app to stop working. These reasons include:

- An ugpraded R package has different behavior. (This could include Shiny itself!
- You make modifications to your app.
- An external data source stops working, or returns data in a changed format

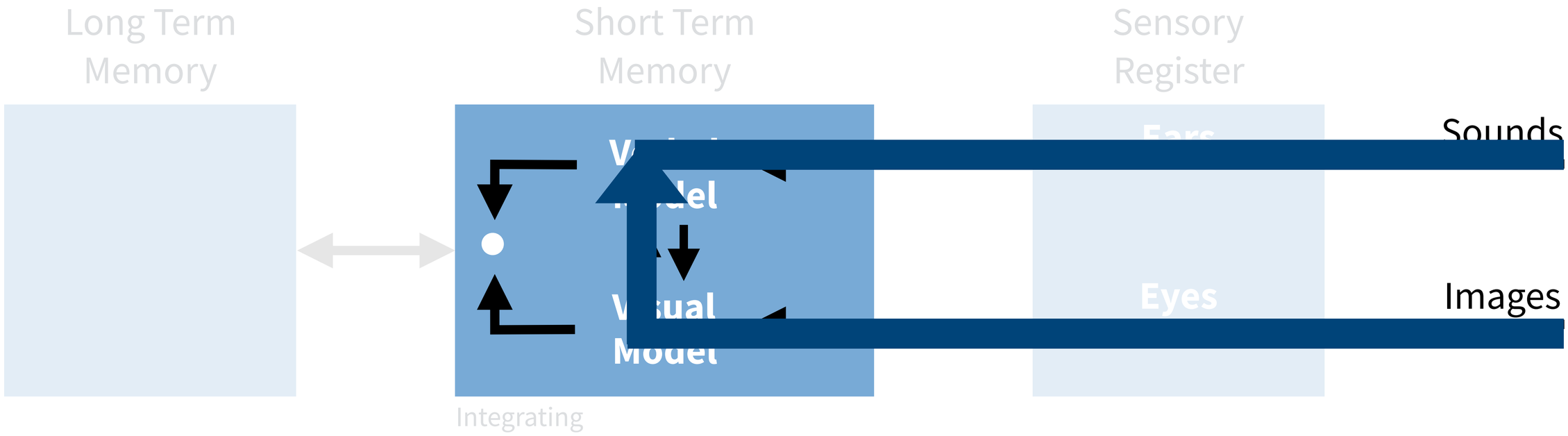
One way to detect these problems is with manual testing – in other words, by having a person interact with the app in a browser – but this can be time-intensive, inconsistent, and imprecise. Having automated tests can alert you to these kinds of problems quickly and with almost zero effort, after the tests have been created.

The **shinytest** package provides tools for creating and running automated tests on Shiny applications.

Dual coding theory

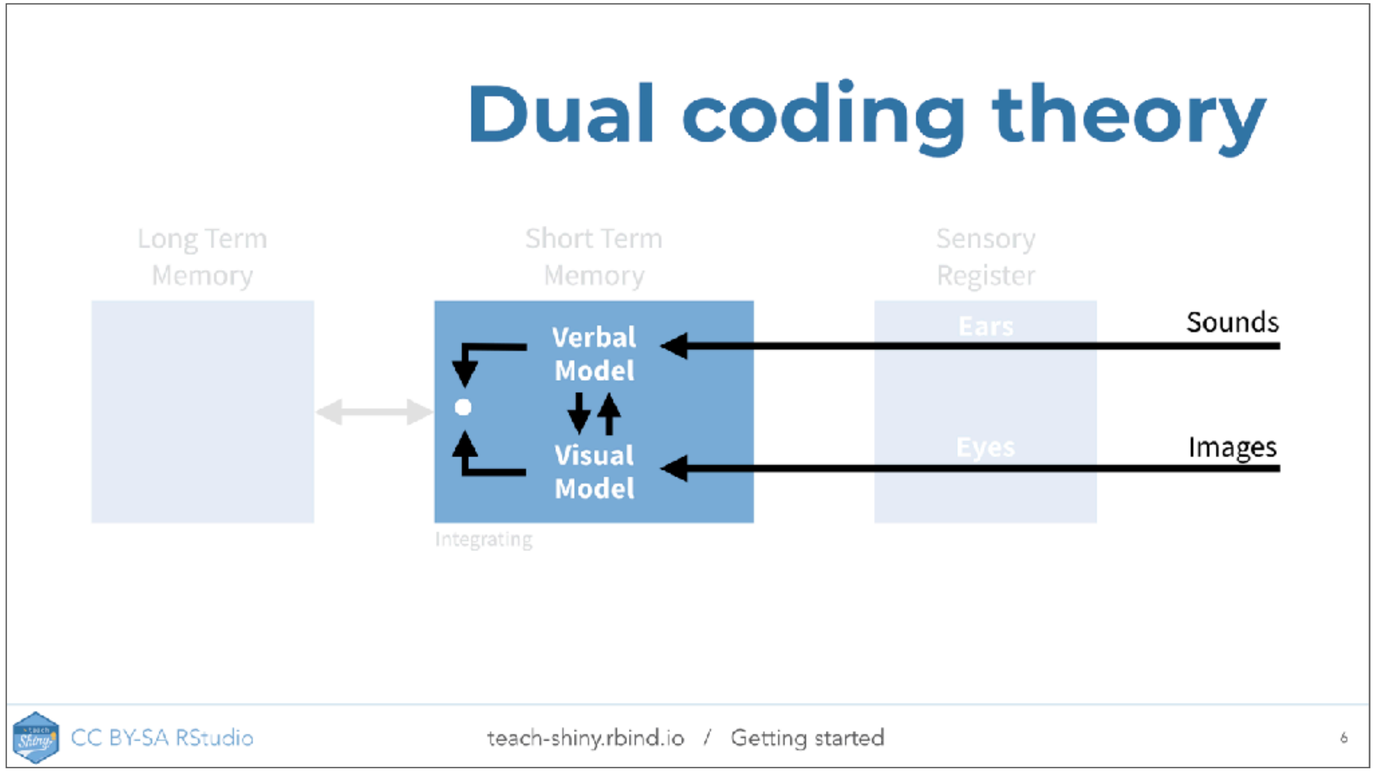


Dual coding theory



Tips

Slides should be for visual information only



visual, with animation

metadata

Tips

Slides should be for visual information only

even when presenting code!

Anatomy of a Shiny app

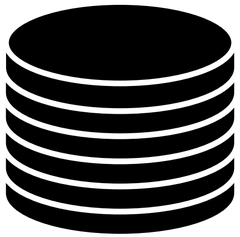
What's in an app?

```
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

National Health and Nutrition Examination Survey



`NHANES::NHANES`

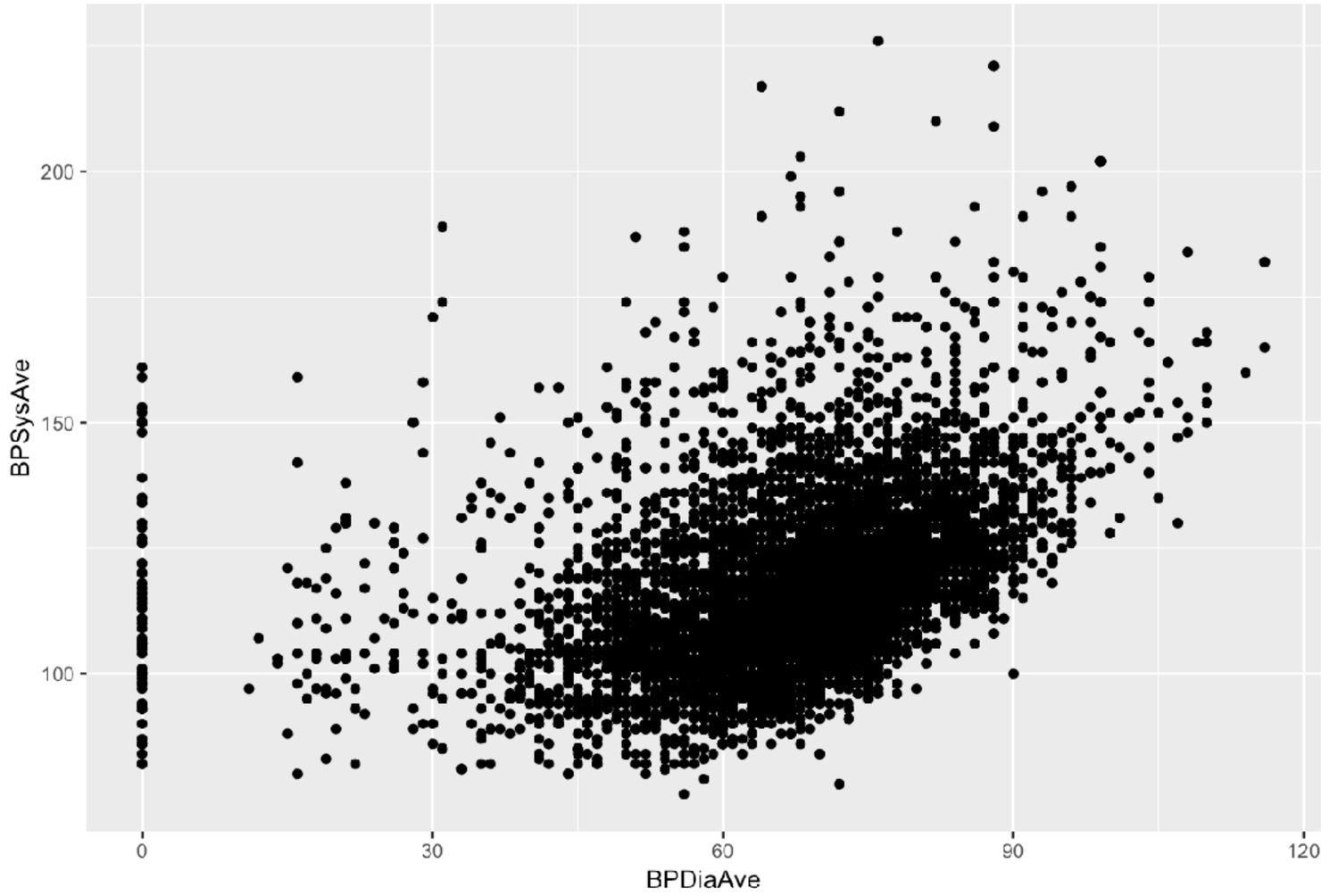
Data from the 2009 - 2010 and 2011 - 2012 surveys on 10,000 participants and 76 variables collected on them

Y-axis:

BPSysAve

X-axis:

BPDiaAve



App template

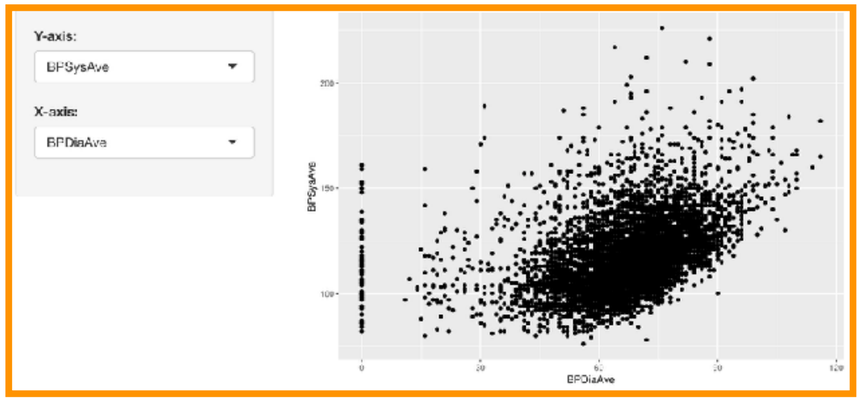
```
library(shiny)
library(tidyverse)
library(NHANES)
ui <- fluidPage()

server <- function(input, output) {}

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

User interface

```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("Age", "Poverty", "Pulse", "AlcoholYear", "BPSysAve"),
                  selected = "BPSysAve"),
      # Select variable for x-axis
      selectInput(inputId = "x", label = "X-axis:",
                  choices = c("Age", "Poverty", "Pulse", "AlcoholYear", "BPSysAve"),
                  selected = "BPDiaAve")
    ),

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

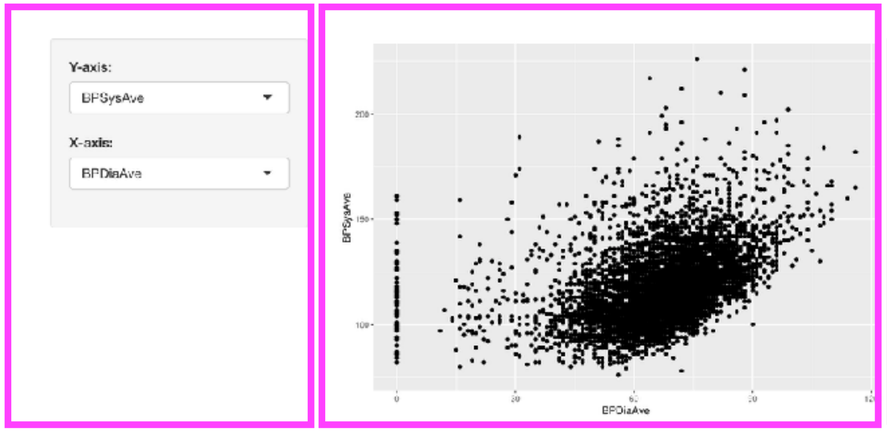
Create fluid page layout

```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("Age", "Poverty", "Pulse", "AlcoholYear", "BPSysAve"),
                  selected = "BPSysAve"),
      # Select variable for x-axis
      selectInput(inputId = "x", label = "X-axis:",
                  choices = c("Age", "Poverty", "Pulse", "AlcoholYear", "BPDiaAve"),
                  selected = "BPDiaAve")
    ),

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



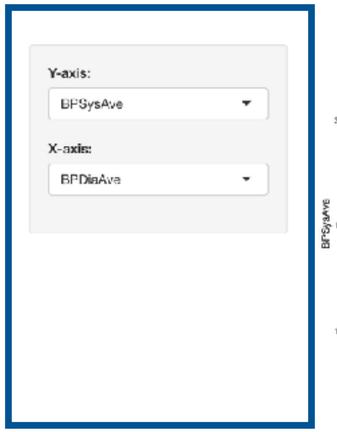
```r
# Define UI
ui <- fluidPage(

  # 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("Age", "Poverty", "Pulse", "AlcoholYear", "BPSysAve"),
                  selected = "BPSysAve"),
      # Select variable for x-axis
      selectInput(inputId = "x", label = "X-axis:",
                  choices = c("Age", "Poverty", "Pulse", "AlcoholYear", "BPDiaAve"),
                  selected = "BPDiaAve")
    ),

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

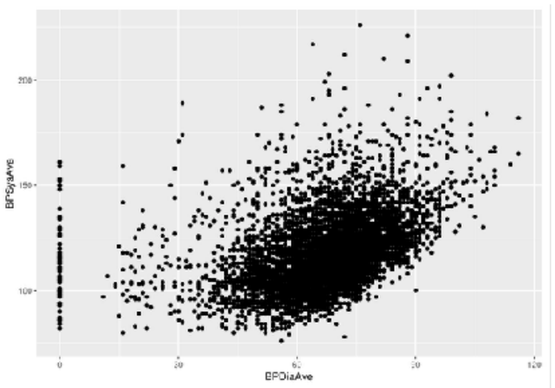
Create a layout with a sidebar and main area



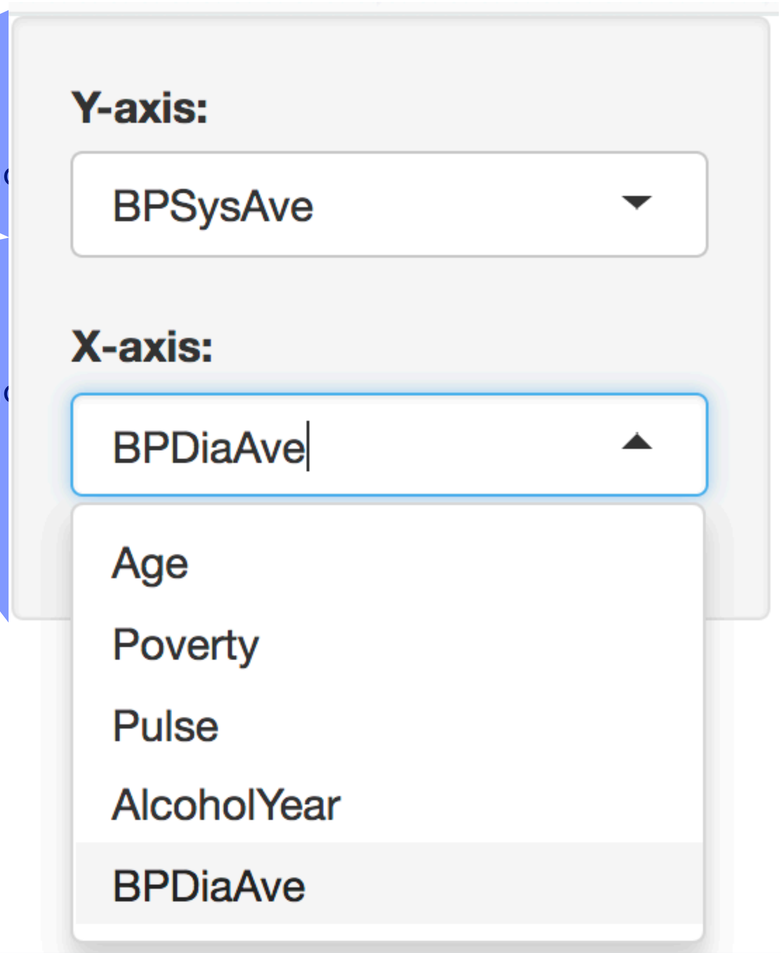
```r
# Define UI
ui <- fluidPage(

  # 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("Age", "Poverty", "Pulse", "AlcoholYear", "BPSysAve"),
                  selected = "BPSysAve"),
      # Select variable for x-axis
      selectInput(inputId = "x", label = "X-axis:",
                  choices = c("Age", "Poverty", "Pulse", "AlcoholYear", "BPDiaAve"),
                  selected = "BPDiaAve")
    ),

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

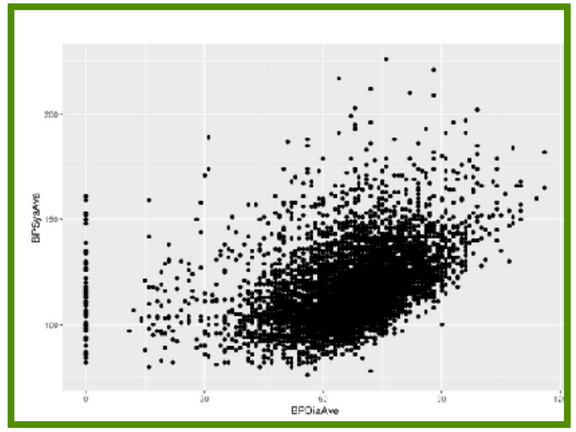
Create a sidebar panel containing **input** controls that can in turn be passed to `sidebarLayout`





```r
# Define UI
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("Age", "Poverty", "Pulse", "Alcoh
                  selected = "BPSysAve"),
      # Select variable for x-axis
      selectInput(inputId = "x", label = "X-axis:",
                  choices = c("Age", "Poverty", "Pulse", "Alcoh
                  selected = "BPDiaAve")
    ),

    # 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("Age", "Poverty", "Pulse", "AlcoholYear", "BPSysAve"),
                  selected = "BPSysAve"),
      # Select variable for x-axis
      selectInput(inputId = "x", label = "X-axis:",
                  choices = c("Age", "Poverty", "Pulse", "AlcoholYear", "BPDiaAve"),
                  selected = "BPDiaAve")
    ),

    # 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$scatterplot <- renderPlot({
    ggplot(data = NHANES, aes_string(x = input$x, y = input$y)) +
      geom_point()
  })
}
```

```
# Define server function
server <- function(input, output) {

  # Create the scatterplot object the plotOutput function is expecting
  output$scatterplot <- renderPlot({
    ggplot(data = NHANES, aes_string(x = input$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 fun
  output$scatterplot <- renderPlot({
    ggplot(data = NHANES, aes_string(x = input$x, y
      geom_point()
  })
}
```

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

```
# Define server function
server <- function(input, output) {

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

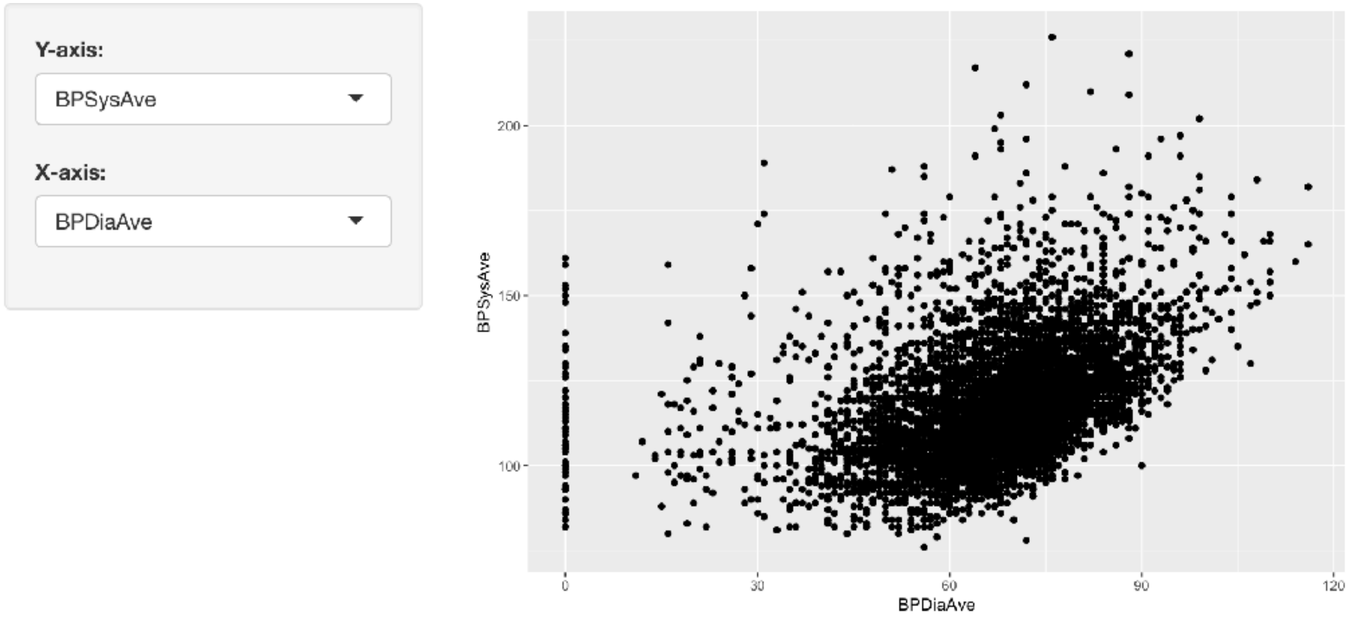
Good ol' ggplot2 code, with **input**s from UI

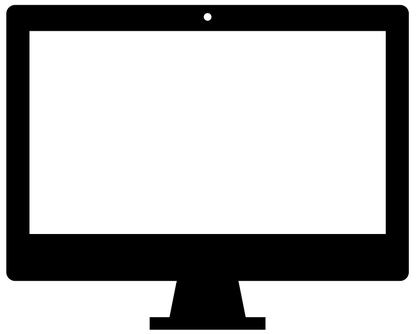
UI + Server

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

Putting it all together...

`nhanes/nhanes-01.R`





DEMO

Annotation vs. reproducibility

etc.

vs.

etc.

Your turn

- Create 1-5 slides that teach some component of the app on the left.
- Feel free to discuss ideas with each other, but create your own unique presentations.
- Then, review each others' presentations, and provide feedback.

Think 10m 00s

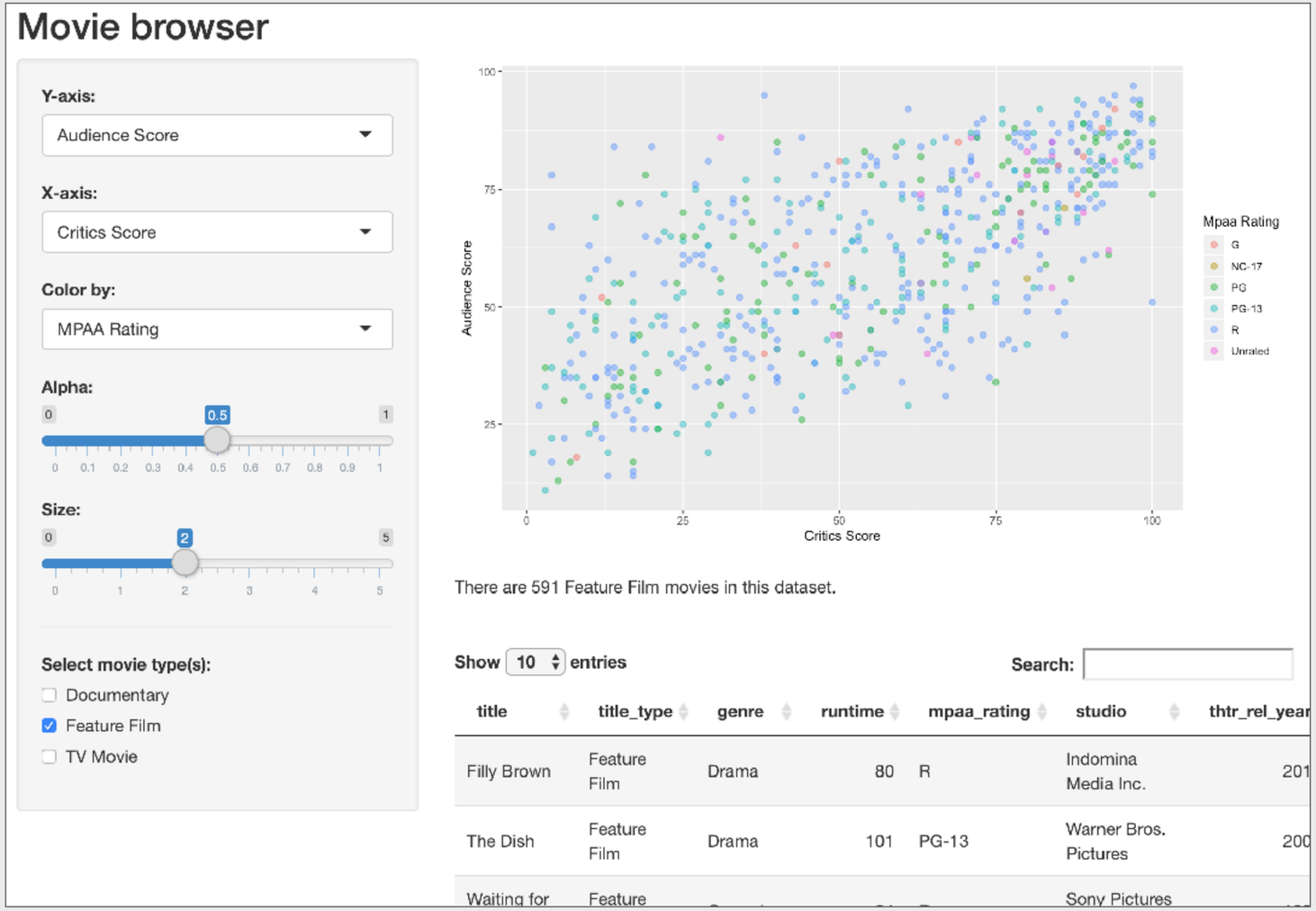
Pair 5m 00s

Your turn

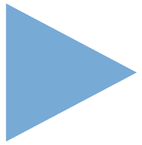
- Create 1-5 slides that teach some component of the app on the left.
- Feel free to discuss ideas with each other, but create your own unique presentations.
- Then, review each others' presentations, and provide feedback.

Think 10m 00s

Pair 5m 00s



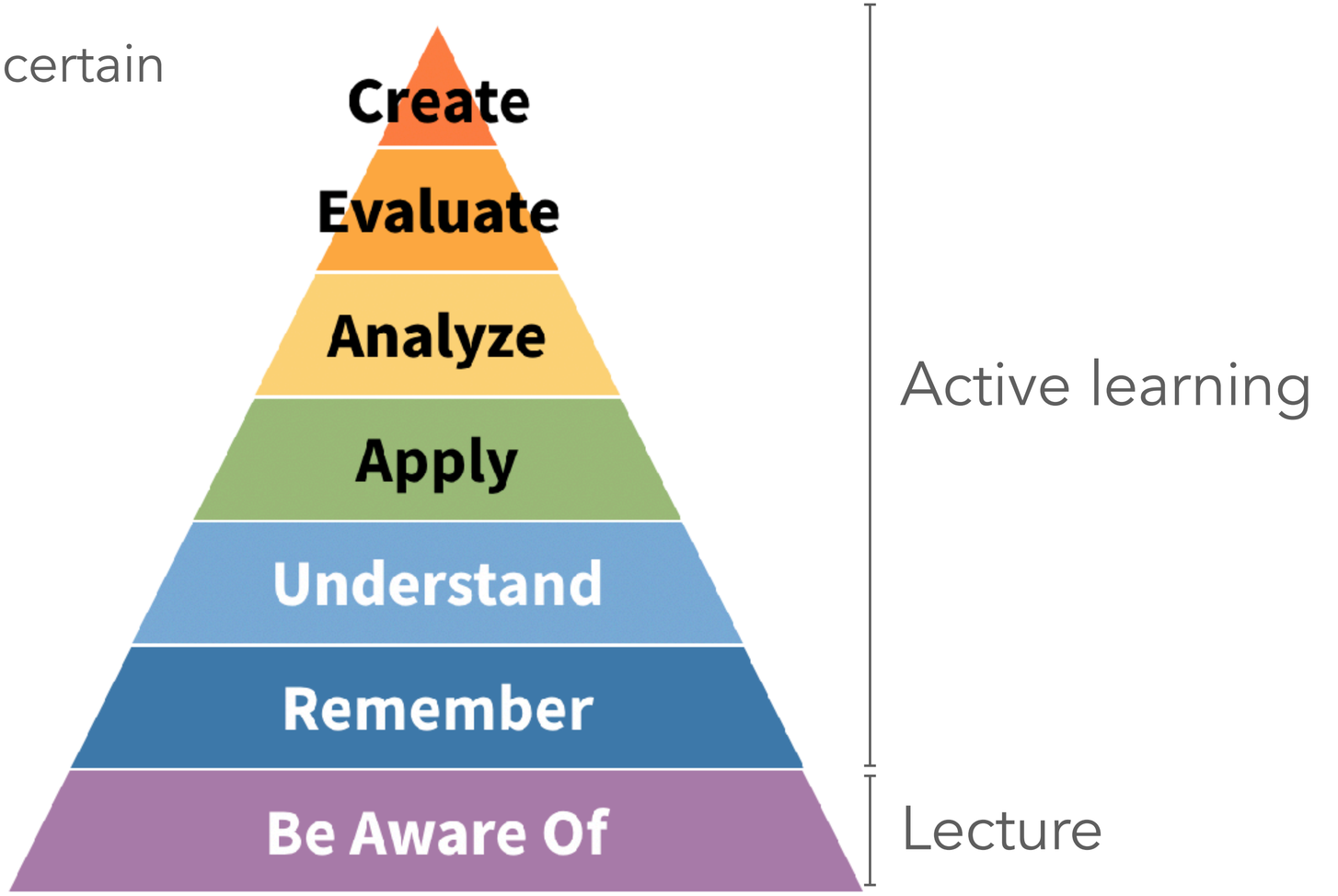
sprinkle inter-activity



Method of delivery

How long it takes to deliver certain material depends on

- Topics covered
- Level of desired mastery



Teaching, fast and slow

- Lecture:
  - Easy to gauge length
  - Useful in a workshop setting to make audience aware of features (and provide additional resources for self study)
- Active learning:
  - Difficult to gauge length, often takes longer than you think
  - Much more likely to hit higher tiers of learning in Bloom's taxonomy
- It's not what you teach, it's what they learn!

Active learning

- Polling questions
- Peer Instruction
- Think-Pair-Share
- One Minute Paper
- Work together in teams
- Assessments

Go to **rstd.io/shiny-poll** to respond

What is wrong with this app?

(a) Line 8: there should be a comma at the end

(b) Line 13: `add_2` should be a reactive expression

(c) Line 14: `current_x` should be a reactive expression

(d) Line 15: should use `renderUI` instead of `renderText`

```
01 library(shiny)
02
03 # UI
04 ui ← fluidPage(
05   titlePanel("Add 2"),
06   sliderInput("x", "Select x", min = 1,
07               max = 50, value = 30),
08   textOutput("x_updated")
09 )
10
11 # Server
12 server ← function(input, output) {
13   add_2             ← function(x) { x + 2 }
14   current_x         ← add_2(input$x)
15   output$x_updated ← renderText({ current
16 }
17
18 # Create Shiny app object
19 shinyApp(ui, server)
```

Polling question

Discuss your response with your partner, then go to **rstd.io/shiny-poll** to respond again

What is wrong with this app?

(a) Line 8: there should be a comma at the end

(b) Line 13: `add_2` should be a reactive expression

(c) Line 14: `current_x` should be a reactive expression

(d) Line 15: should use `renderUI` instead of `renderText`

```
01 library(shiny)
02
03 # UI
04 ui ← fluidPage(
05   titlePanel("Add 2"),
06   sliderInput("x", "Select x", min = 1,
07               max = 50, value = 30),
08   textOutput("x_updated")
09 )
10
11 # Server
12 server ← function(input, output) {
13   add_2             ← function(x) { x + 2 }
14   current_x         ← add_2(input$x)
15   output$x_updated ← renderText({ current
16 }
17
18 # Create Shiny app object
19 shinyApp(ui, server)
```

Peer instruction

How would you correct this app code?

Think about it first for 2 minutes, then pair up and discuss your responses. Note, there is more than one correct answer.

Then, you will be asked to describe your partner's answer to the class.

```
01 library(shiny)
02
03 # UI
04 ui ← fluidPage(
05   titlePanel("Add 2"),
06   sliderInput("x", "Select x", min = 1,
07               max = 50, value = 30),
08   textOutput("x_updated")
09 )
10
11 # Server
12 server ← function(input, output) {
13   add_2             ← function(x) { x + 2 }
14   current_x         ← add_2(input$x)
15   output$x_updated ← renderText({ curren
16 }
17
18 # Create Shiny app object
19 shinyApp(ui, server)
```

Think - pair - share

How would you correct this app code?

Before you leave class, take one minute to write down what was most confusing about this exercise.

```
01 library(shiny)
02
03 # UI
04 ui ← fluidPage(
05   titlePanel("Add 2"),
06   sliderInput("x", "Select x", min = 1,
07               max = 50, value = 30),
08   textOutput("x_updated")
09 )
10
11 # Server
12 server ← function(input, output) {
13   add_2             ← function(x) { x + 2 }
14   current_x         ← add_2(input$x)
15   output$x_updated ← renderText({ curren
16 }
17
18 # Create Shiny app object
19 shinyApp(ui, server)
```

One minute paper

How would you correct this app code?

Get in teams of three and make corrections to the code for this app. Note, there is more than one correct answer.

Then, one member from the team will be asked to present your answer.

```
01 library(shiny)
02
03 # UI
04 ui ← fluidPage(
05   titlePanel("Add 2"),
06   sliderInput("x", "Select x", min = 1,
07               max = 50, value = 30),
08   textOutput("x_updated")
09 )
10
11 # Server
12 server ← function(input, output) {
13   add_2             ← function(x) { x + 2 }
14   current_x         ← add_2(input$x)
15   output$x_updated ← renderText({ curren
16 }
17
18 # Create Shiny app object
19 shinyApp(ui, server)
```

Work in teams

How would you correct this app code?

Pop quiz! Make corrections to the app code, and submit your code and a link to your deployed app.

Note, there is more than one correct answer. For this quiz you are asked to submit **two** working solutions.

```
01 library(shiny)
02
03 # UI
04 ui ← fluidPage(
05   titlePanel("Add 2"),
06   sliderInput("x", "Select x", min = 1,
07               max = 50, value = 30),
08   textOutput("x_updated")
09 )
10
11 # Server
12 server ← function(input, output) {
13   add_2             ← function(x) { x + 2 }
14   current_x         ← add_2(input$x)
15   output$x_updated ← renderText({ curren
16 }
17
18 # Create Shiny app object
19 shinyApp(ui, server)
```

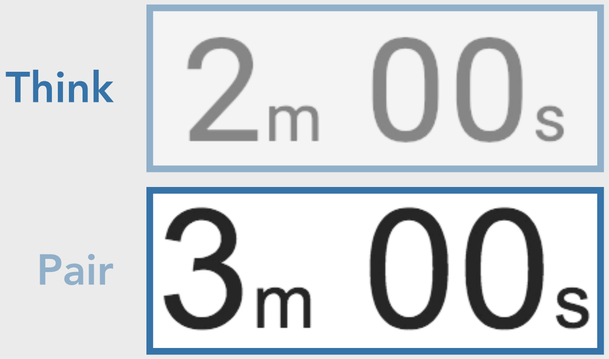
Assessment

Your turn

- Determine the ideal length of time students should be given for the following exercises we reviewed earlier:
  - think-pair-share
  - work in teams
  - quiz
- Compare notes with a partner, discuss any points of disagreement.

Your turn

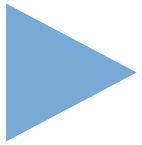
- Determine the ideal length of time students should be given for the following exercises we reviewed earlier:
  - think-pair-share
  - work in teams
  - quiz
- Compare notes with a partner, discuss any points of disagreement.



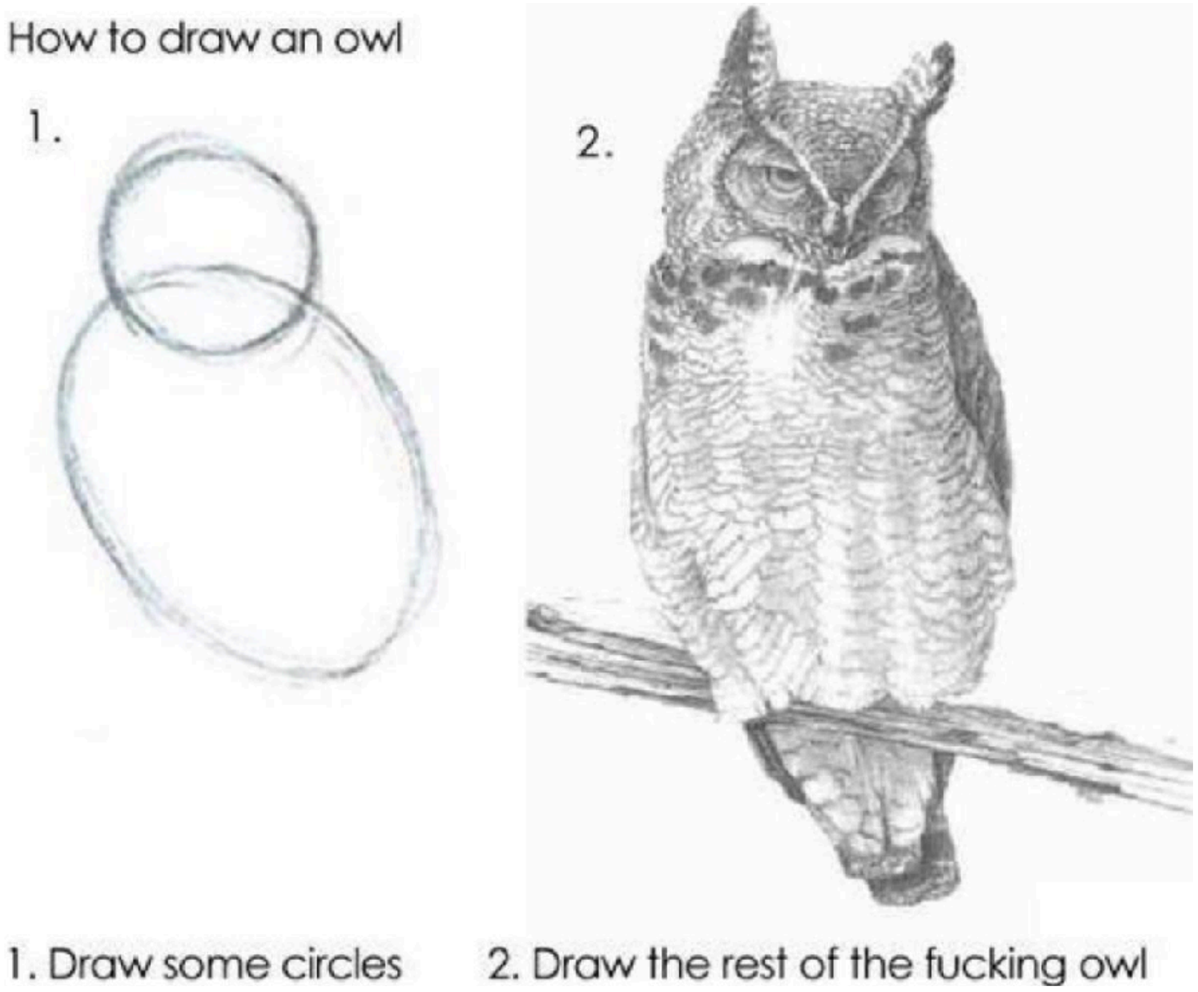
Discussion

What are some tips an instructor can use
for determining how long an exercise
might take students to complete?
Do the experienced instructors in the room
have any tips?

scaffold your exercises



Basically, avoid this!

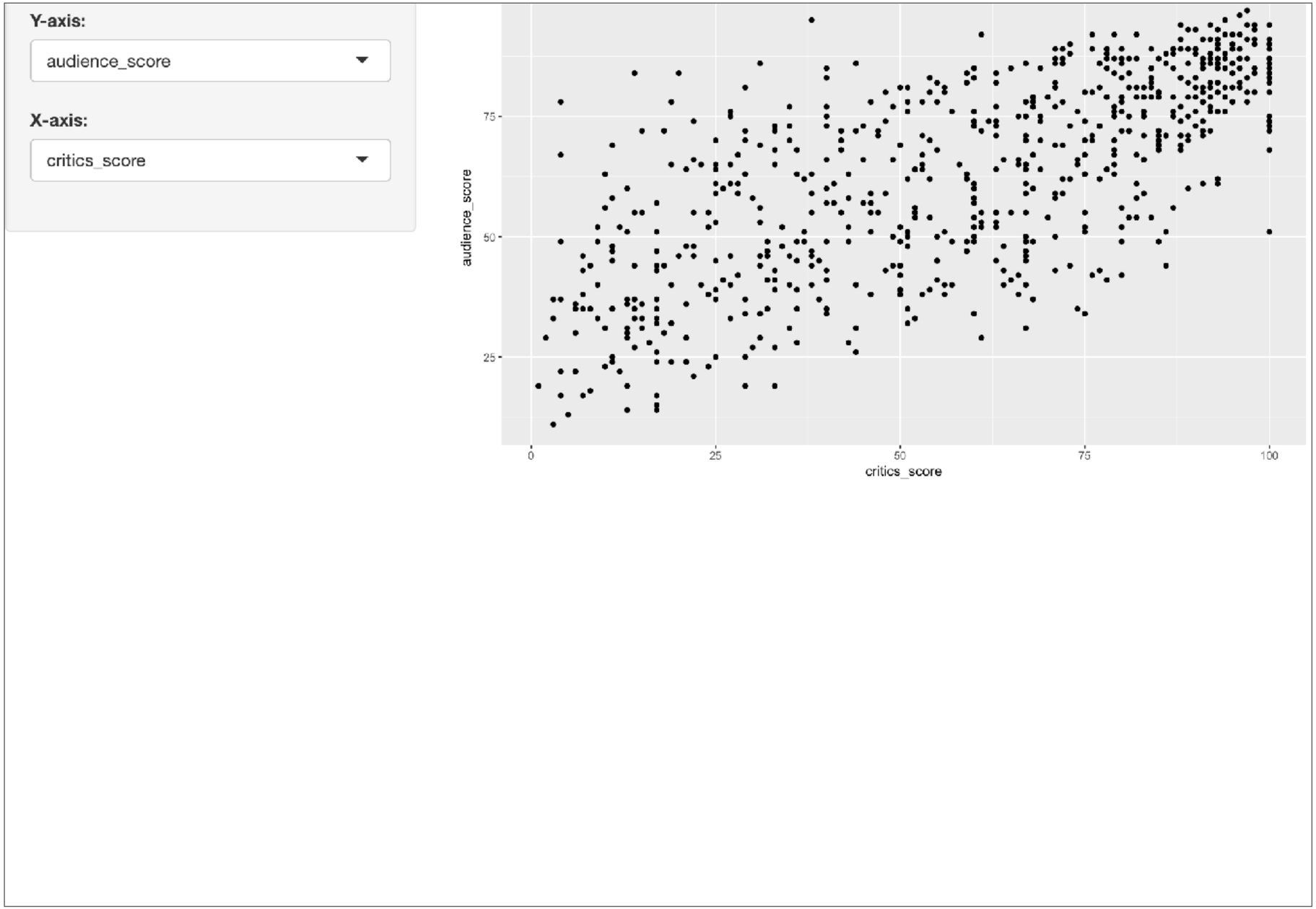


Scaffolding over exercises

- Structure your materials so that you can build up your exercises over time.
- Works especially well when teaching Shiny — start with a simple (borderline boring) app, build up over a series of exercises to a much more complex (interesting) app.

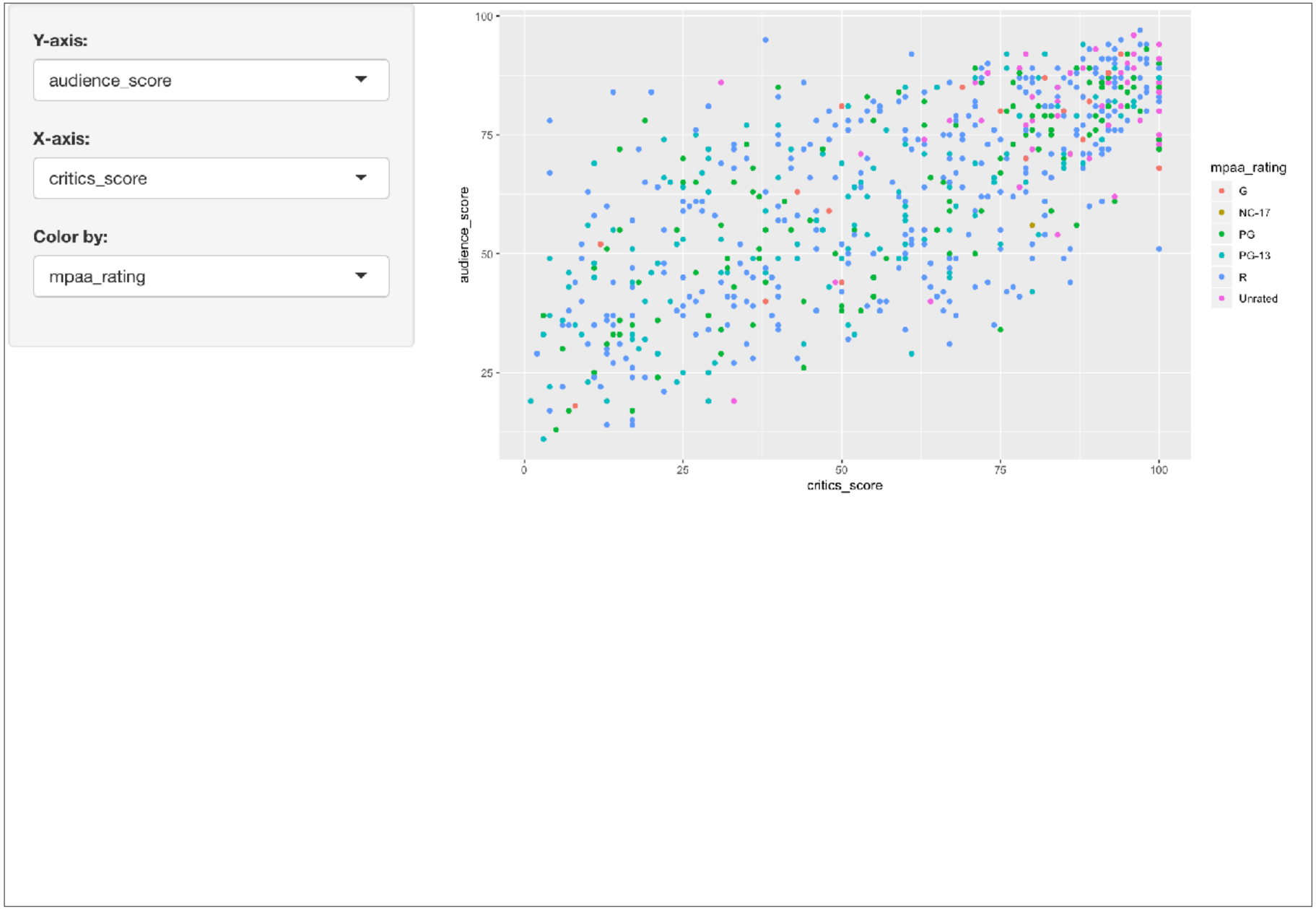
movies_01

Beginning of the day



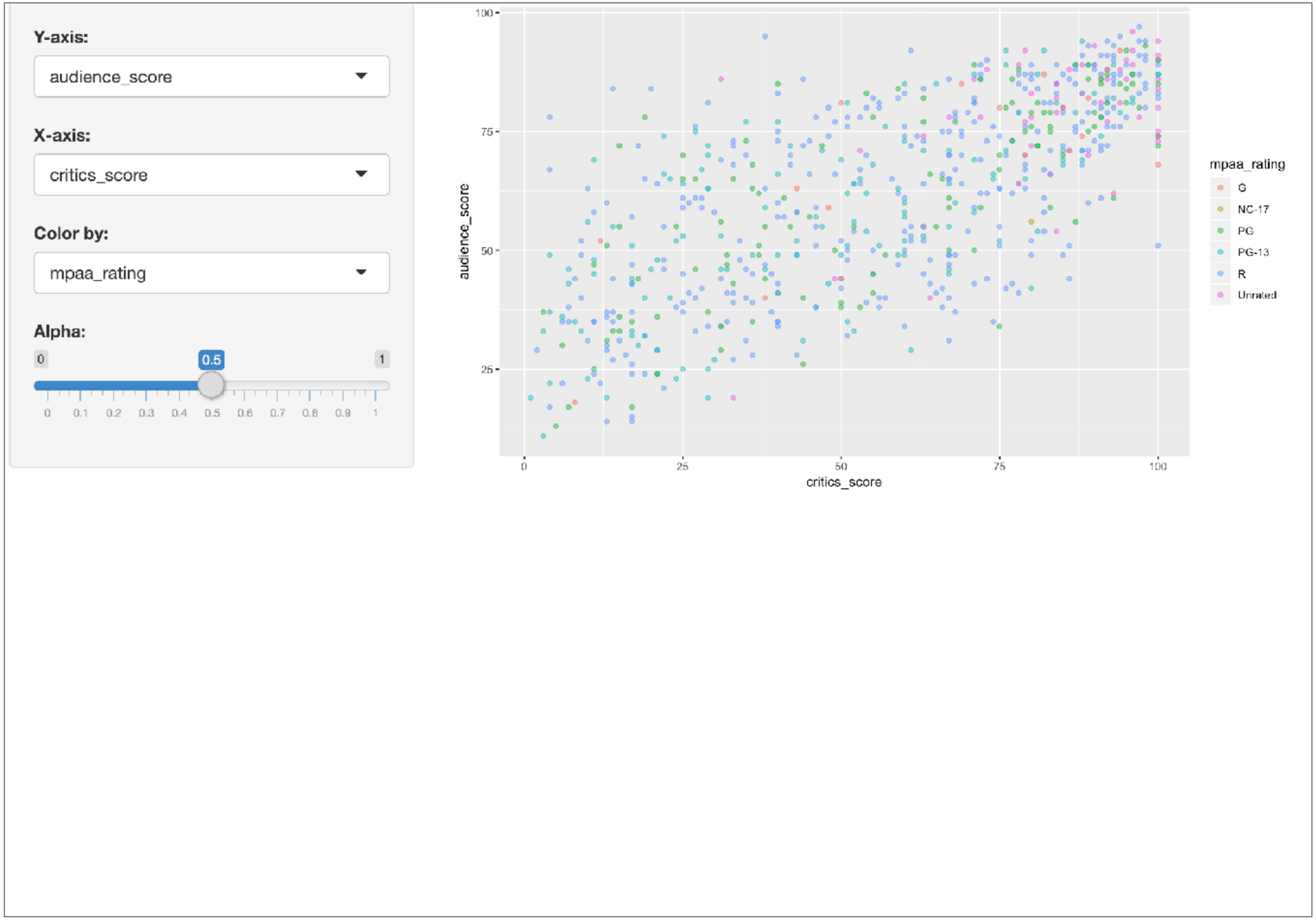
movies_02

A few minutes later



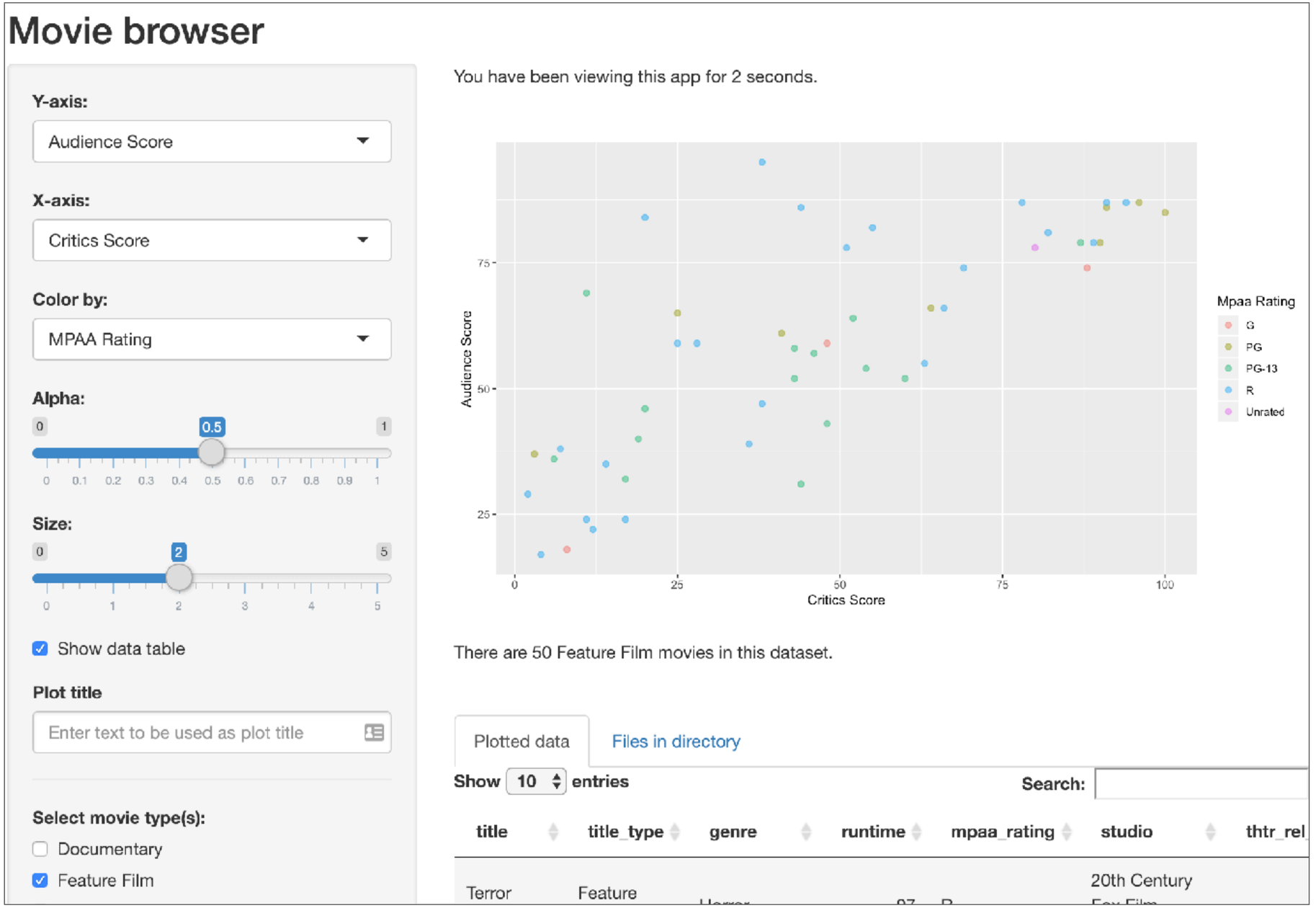
movies_03

After reviewing the cheatsheet for UI widgets



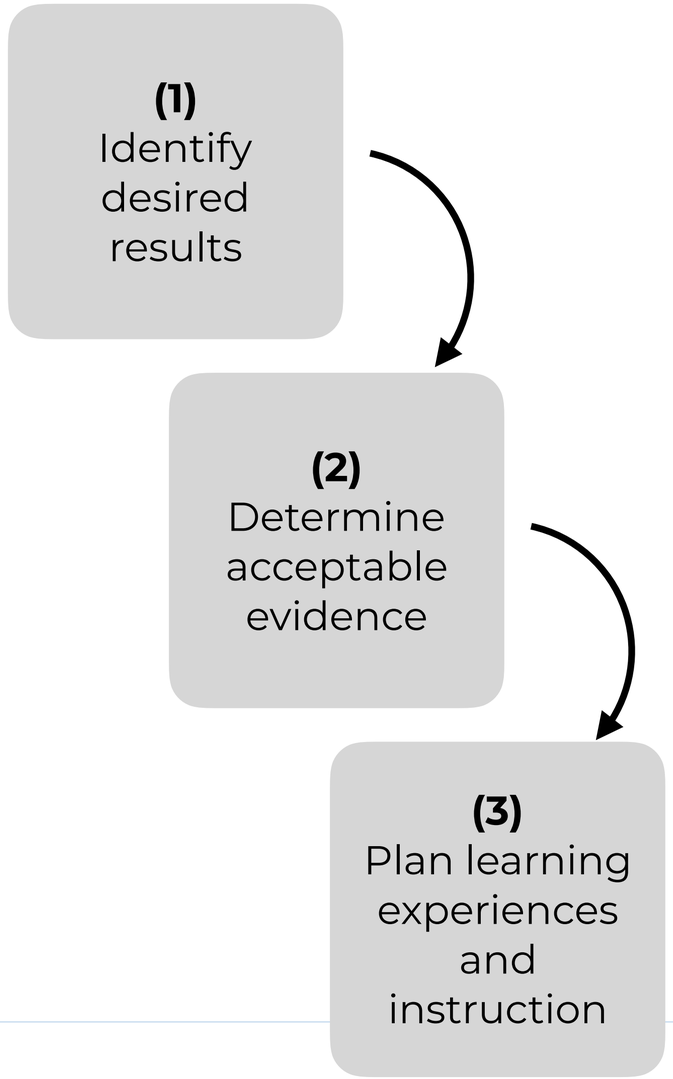
movies_14

At the end of 4 hours



Backwards design

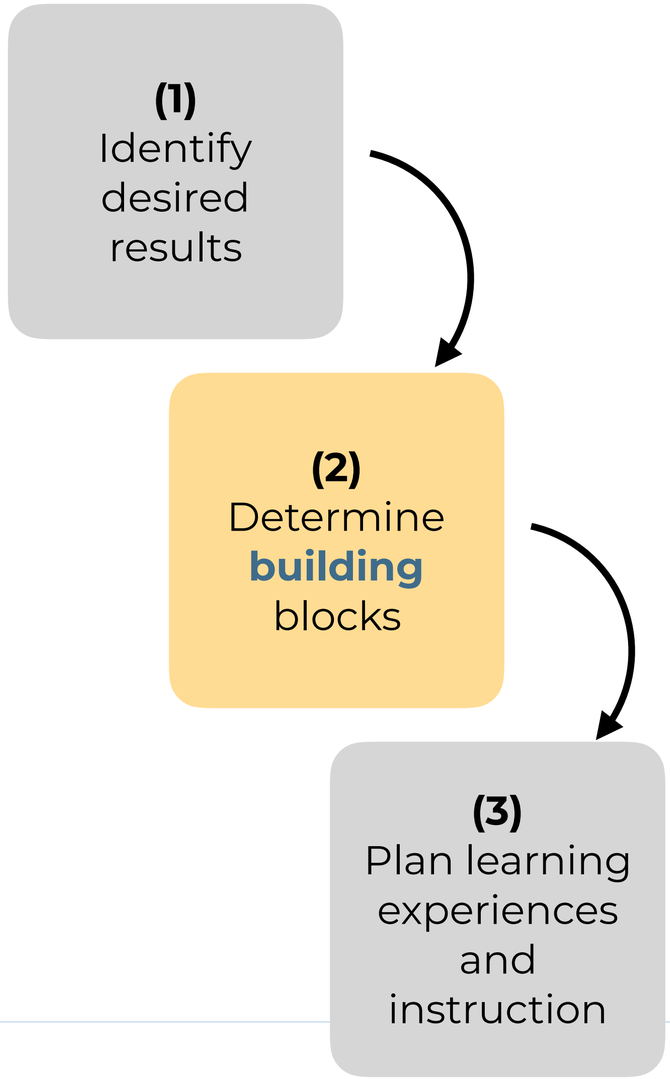
- Set goals for educational curriculum before choosing instructional methods + forms of assessment
- Analogous to travel planning - itinerary deliberately designed to meet cultural goals, not purposeless tour of all major sites in a foreign country



Wiggins, Grant P., Grant Wiggins, and Jay McTighe. Understanding by design. Ascd, 2005.

Designing backwards

- First expose students to the final produce — a complex (not complicated), striking Shiny app
- Then teach the building blocks (concepts, functions, features) used along the way



Your turn

movies / movies_08

- Work in teams to write three exercises that lead up to this app.
- You do not need to start from scratch, instead take a starting point and come up with 3 exercises that end up at this app.

5m 00s

🎨 use visual clues

🏃‍♀️ sprinkle interactivity

🏗️ scaffold your exercises