



# DASHBOARDS

# OUTLINE

- ▶ What is in a dashboard?
- ▶ Server
  - ▶ reactiveFileReader
  - ▶ reactivePoll
- ▶ UI
  - ▶ Static vs. dynamic dashboards
  - ▶ flexdashboard
  - ▶ Shiny pre-rendered
  - ▶ shinydashboard

**What is in a  
dashboard?**

# DASHBOARDS

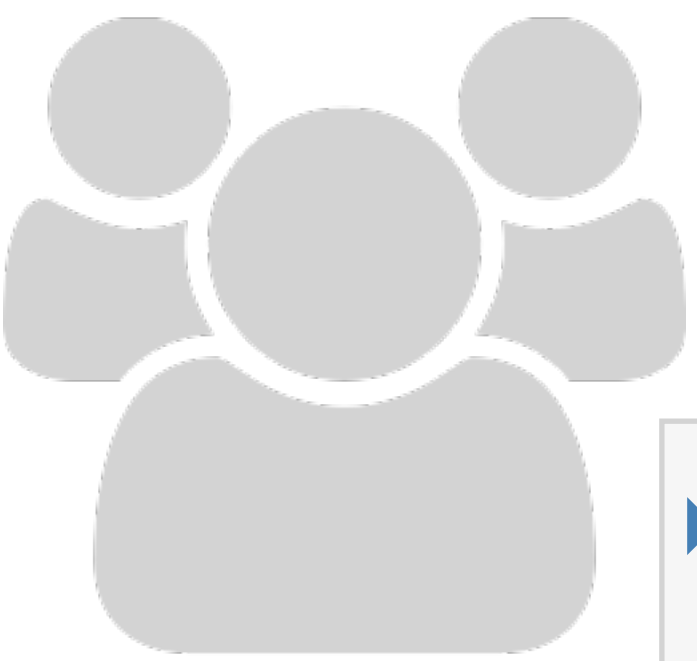
- ▶ Automatically updating
  - ▶ Not just based on user gestures
  - ▶ But also when data source changes
- ▶ Many viewers looking at the same data
- ▶ May or may not be interactive

# Server



# MOTIVATION

- ▶ You have new data coming in — constantly, continuously, or on a schedule
- ▶ When new data comes in, it's automatically received, and transformed, aggregated, summarized, etc.
- ▶ May want to call attention to exceptional results



# EXERCISE

- ▶ Why might this not be a good idea?

```
dataset <- reactive({  
  result <- read.csv("data.csv")  
  invalidateLater(5000)  
  result  
})  
  
output$plot <- renderPlot({  
  plot(dataset()) # or whatever  
})
```



# SOLUTION

Lots of overhead!



**reactiveFileReader**

# REACTIVEFILEREADER

- ▶ Reads the given file ("**data.csv**") using the given function (**read.csv**)
- ▶ Periodically reads the last-modified time of the file
- ▶ If the timestamp changes, then (and only then) re-reads the file

Single file, on disk  
(not database or web API)

```
dataset <- reactiveFileReader(  
  intervalMillis = 1000,  
  session = session,  
  filePath = "data.csv",  
  readFunc = read.csv  
)  
  
output$plot <- renderPlot({  
  plot(dataset()) # or whatever  
})
```

Must have data path as  
first argument

# REACTIVEFILEREADER

```
dataset <- reactiveFileReader(  
  intervalMillis = 1000,  
  session = session,  
  filePath = "data.csv",  
  readFunc = read.csv,  
  stringsAsFactors = FALSE  
)  
  
output$plot <- renderPlot({  
  plot(dataset()) # or whatever  
})
```

Add any named  
arguments

**reactivePoll**

# REACTIVEPOLL

- ▶ **reactiveFileReader** is limited to files on disk. It doesn't work for non-file-based data sources like databases or web APIs
- ▶ **reactivePoll** is a generalization of reactiveFileReader
  - ▶ **checkFunc**: A function that can execute quickly, and merely determine if anything has changed
    - ▶ Should be fast, as it will block the R process while it runs! The slower it is, the greater you should make the polling interval.
    - ▶ Should not return **TRUE** or **FALSE** for changed/unchanged. Instead, just return a value (like the timestamp, or the count); it's **reactivePoll**'s job, not yours, to keep track of whether that value is the same as the previous value or not.
  - ▶ **valueFunc**: A function with the (potentially expensive) logic for actually reading the data

**UI**



# **Static vs. dynamic dashboards**

# STATIC VS. DYNAMIC

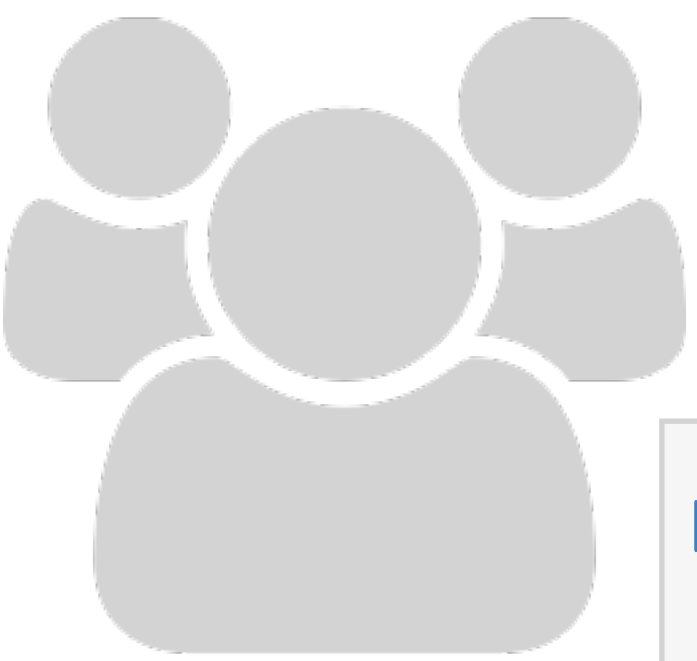
- ▶ Static:
  - ▶ R code runs once and generates an HTML page
  - ▶ Generation of this HTML can be scheduled
- ▶ Dynamic:
  - ▶ Client web browser connects to an R session running on server
  - ▶ User input causes server to do things and send information back to client
  - ▶ Interactivity can be on client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

**flexdashboard**

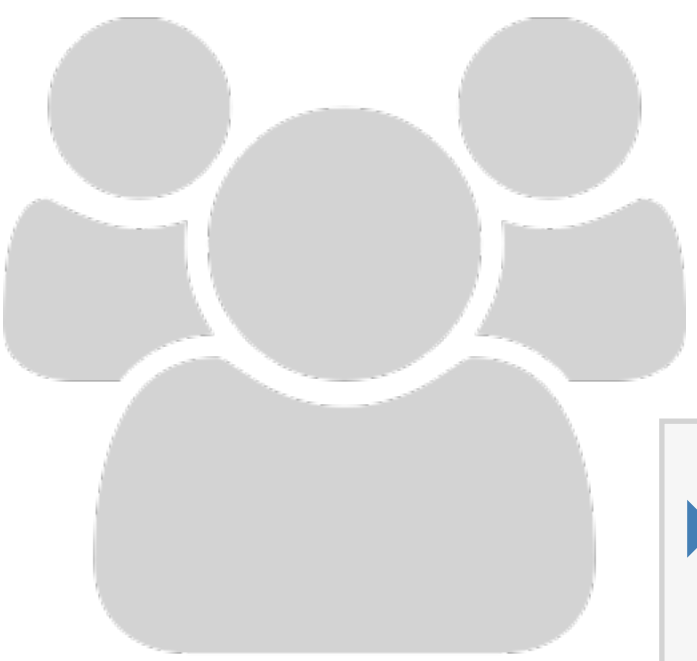
# EXERCISE



- ▶ `library(flexdashboard)`
- ▶ File → New file → R Markdown → From Template
- ▶ Create three plots that go in each of the panes using built-in R datasets or any data we have used in the workshop (or your own data)

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

# EXERCISE



- ▶ Open **apps/dashboards/flexdashboard\_01.Rmd**
- ▶ How is it different than Shiny apps we have been building so far, how is it similar?
- ▶ Make a change to the layout of the dashboard, see <http://rmarkdown.rstudio.com/flexdashboard/using.html#layout> for help
- ▶ Change the theme of the dashboard, see <http://rmarkdown.rstudio.com/flexdashboard/using.html#appearance> for help

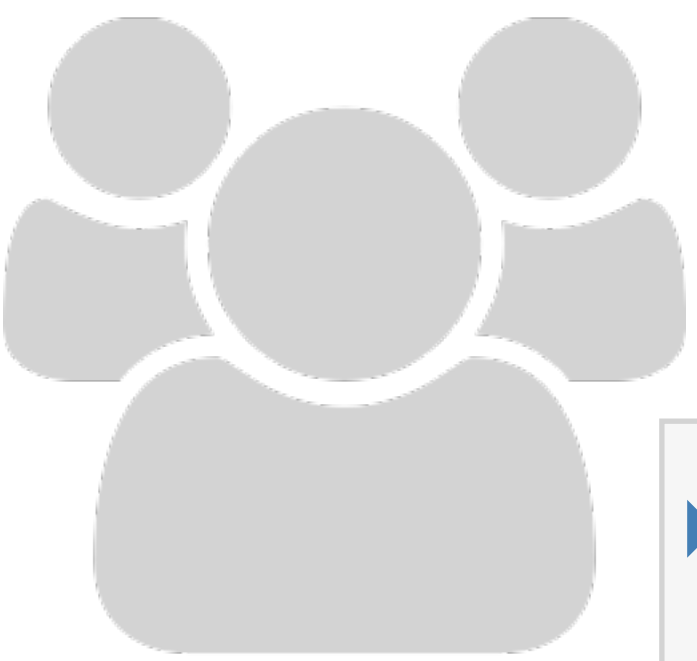
5<sub>m</sub> 00<sub>s</sub>



# SHINY DOCUMENTS

- ▶ Add **runtime: shiny** to header.
- ▶ Add **inputs** in code chunks.
- ▶ Add reactive expressions.
- ▶ Add **renderXyz** functions in code chunks.
  - ▶ No need for **output\$x** `<-` assignment, or for **xyzOutput** functions.

# EXERCISE



- ▶ Continue working on **apps/dashboards/flexdashboard\_01.Rmd**
- ▶ Add another UI widget, a radioButton, that allows the user to select whether the plot used to visualize the distribution of weight should be histogram or a violin plot

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



# SOLUTION

Sample solution at **`apps/dashboards/flexdashboard_02.Rmd`**

# SHINY DOCUMENT DRAWBACKS

- ▶ Start-up time: knits document every time someone visits it
- ▶ Resizing can trigger re-knit
- ▶ Auto-reconnection doesn't work (i.e. client browsers cannot automatically reconnect after being disconnected due to network problems)
- ▶ **The solution:** Pre-rendered Shiny Documents

**Shiny**

**pre-rendered**

# SHINY PRERENDERED

- ▶ **Rendering phase:** UI code (and select other code) is run once, before users connect.
- ▶ **Serving phase:** Server code is run once for each user session.
- ▶ Each phase is run in a separate R sessions and can't access variables from the other phase.
- ▶ Each R code chunk in the document belongs to one phase.



# SHINY PRERENDERED

```
---  
title: "Hello Prerendered Shiny"  
output: html_document  
runtime: shiny_prerendered  
---  
  
```{r, context="render", echo=FALSE}  
sliderInput("bins", "Number of bins:", min = 1, max = 50, value = 30)  
plotOutput("distPlot")  
````  
  
```{r, context="server"}  
output$distPlot <- renderPlot({  
  x <- faithful[, 2] # Old Faithful Geyser data  
  bins <- seq(min(x), max(x), length.out = input$bins + 1)  
  hist(x, breaks = bins, col = 'darkgray', border = 'white')  
})  
```
```

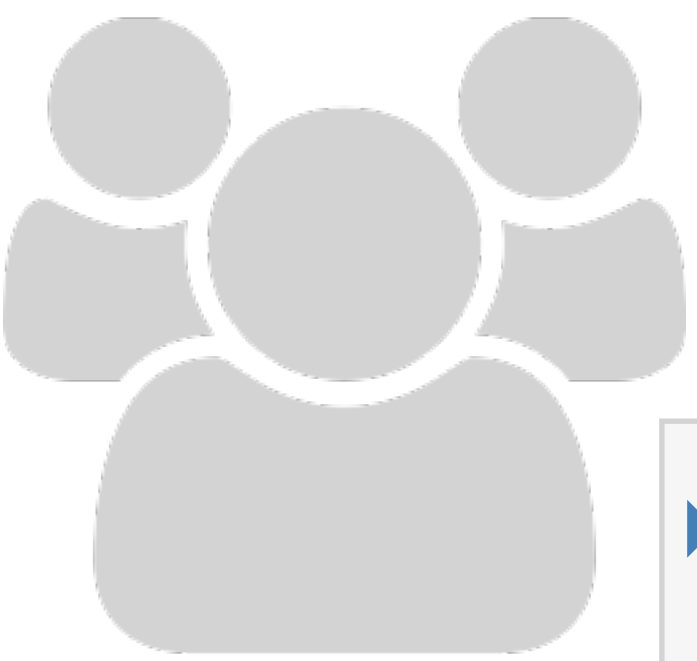
Executed when the  
UI is rendered

Executed whenever  
a client connects  
(server function)

# CONTEXTS FOR SHINY\_PRERENDERED

- ▶ **"render"**: Runs in rendering phase (like **ui**)
- ▶ **"server"**: Runs in serving phase (like **server**)
- ▶ Additional contexts:
  - ▶ **"setup"**: Runs in both phases (like **global.R**)
  - ▶ **"data"**: Runs in rendering phase (any variables are saved to a file, and available to serving phase, useful for data preprocessing)
  - ▶ **"server-start"**: Runs once in serving phase, when the Shiny document is first run and is not re-executed for each new user of the document, appropriate for
    - ▶ establishing shared connections to remote servers (e.g. databases, Spark contexts, etc.)
    - ▶ creating reactive values/expressions to be shared across sessions (e.g. with **reactivePoll**, **reactiveFileReader**)

# EXERCISE



- ▶ Start with **apps/dashboards/flexdashboard\_02.Rmd**
- ▶ Turn your document into **runtime: shiny\_prerendered**
- ▶ *Note:* You will need to use **output\$x** `<-` assignment and **xyzOutput** functions

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



# SOLUTION

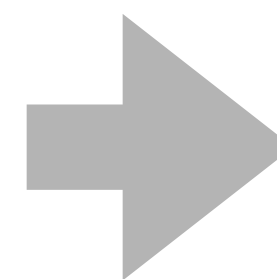
Sample solution at **`apps/dashboards/flexdashboard_03.Rmd`**

**shinydashboard**

# SHINYDASHBOARD

- ▶ Shinydashboard is a theme for Shiny, built on top of AdminLTE
- ▶ Structurally the same as building a regular Shiny app, but with a different “vocabulary” of UI functions

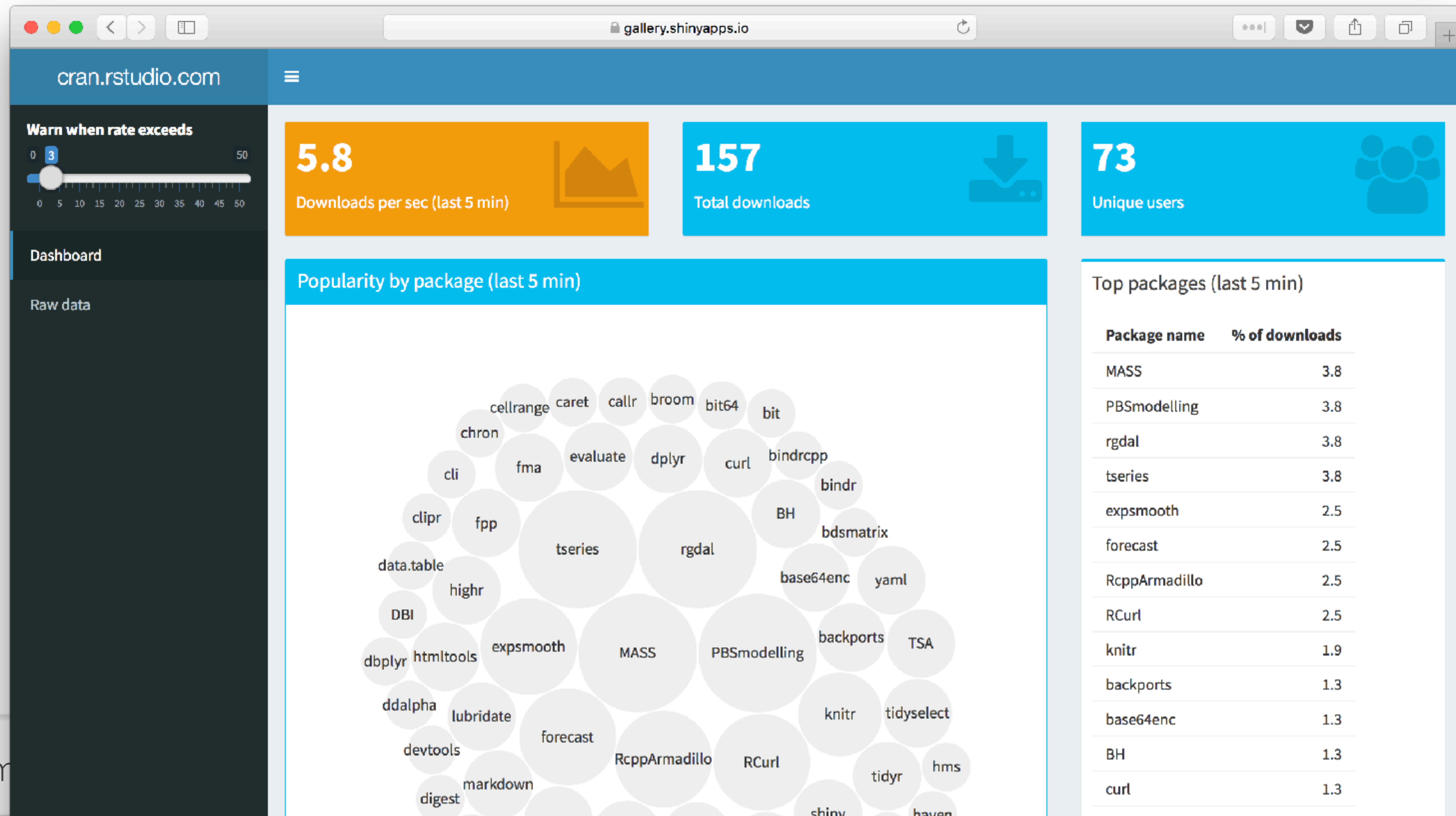
```
ui <- fluidPage(  
  titlePanel("Title"),  
  sidebarLayout(  
    sidebarPanel( ... ),  
    mainPanel( ... )  
  )  
)
```



```
ui <- dashboardPage(  
  dashboardHeader("Title"),  
  dashboardSidebar( ... ),  
  dashboardBody( ... )  
)
```



# SHINYDASHBOARD



# Shiny from

# SHINYDASHBOARD

`dashboardHeader(title="cran.rstudio.com")`

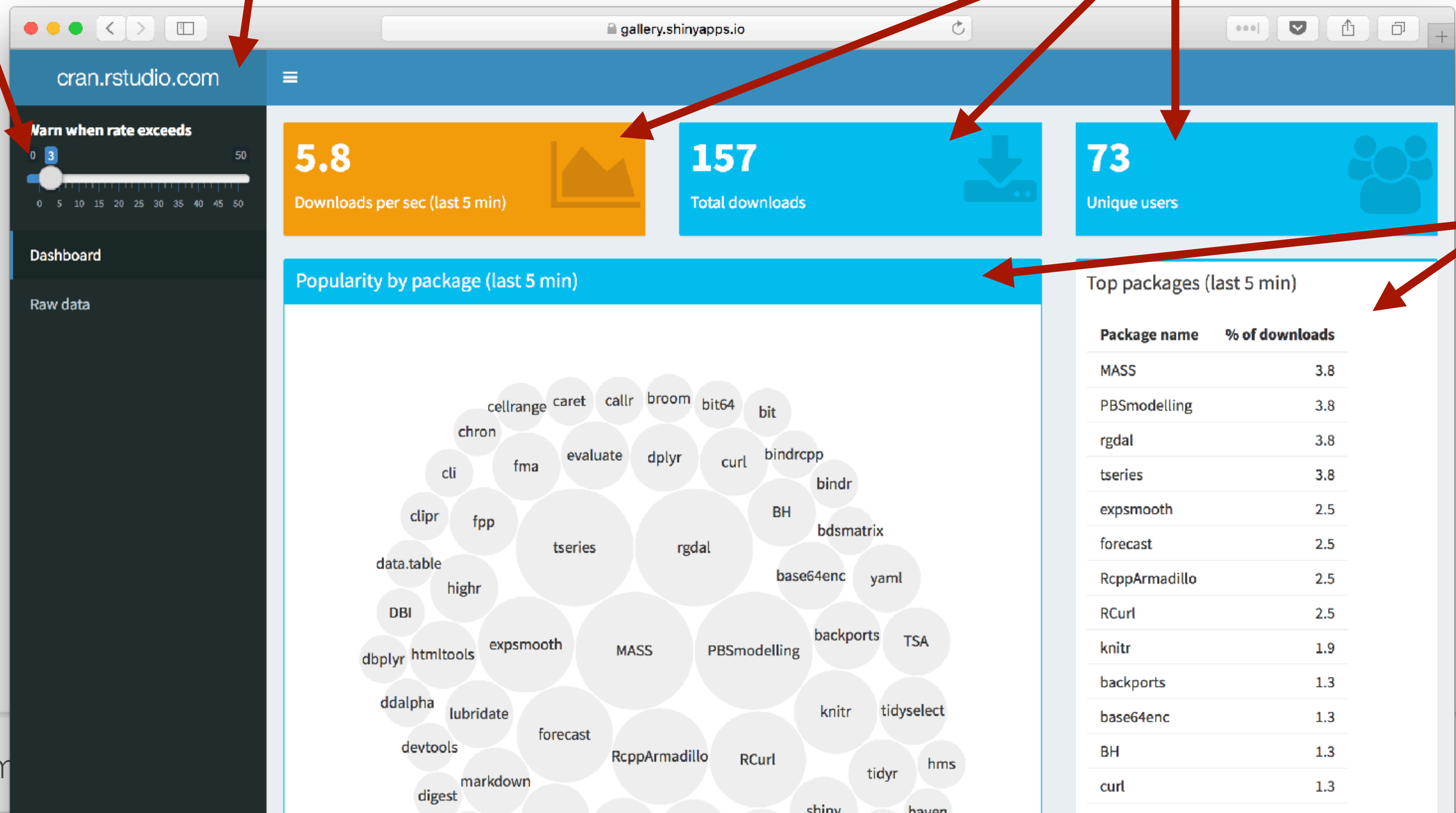
`dashboardSidebar(...)`

`valueBoxOutput/renderValueBox`

`box(...)`

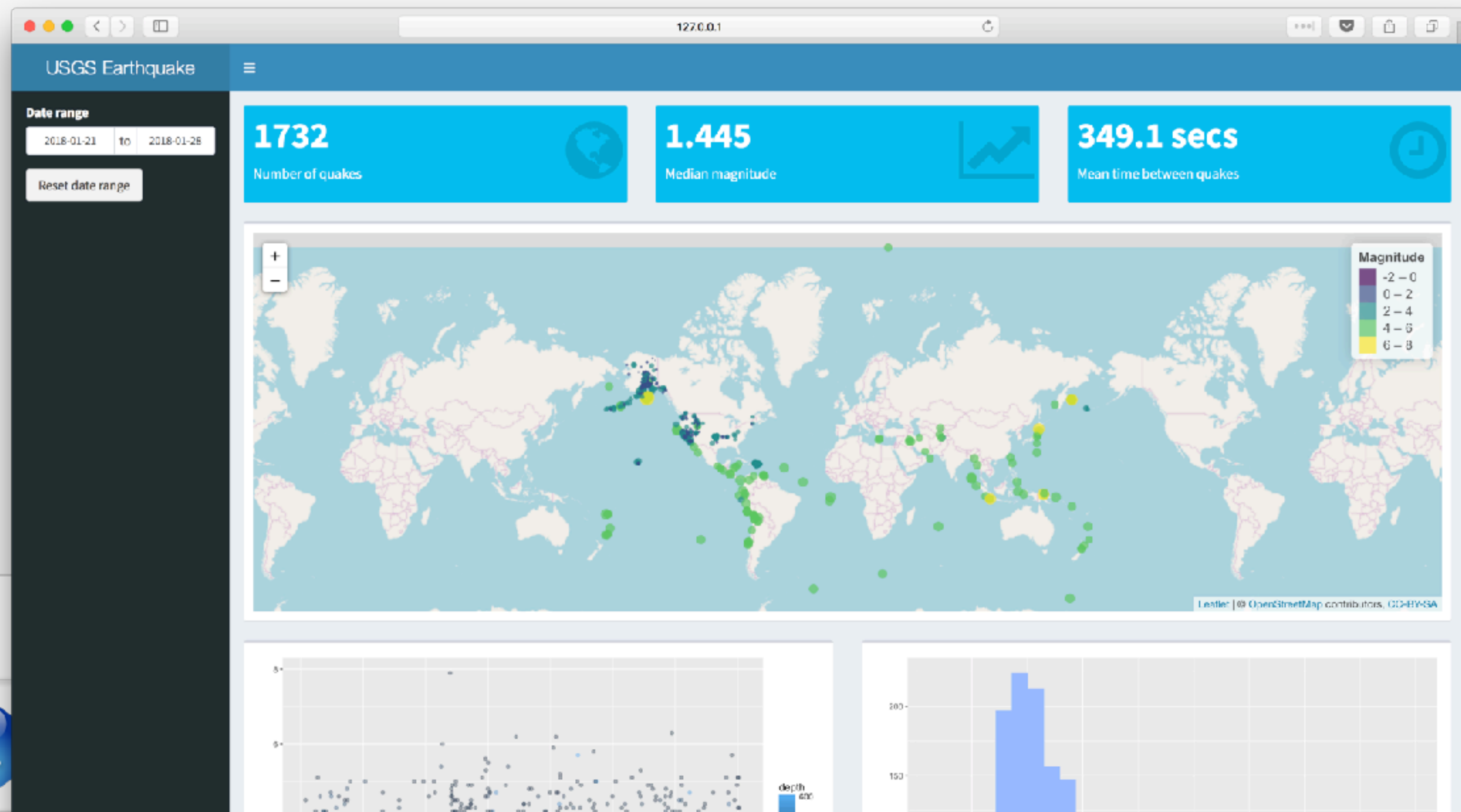
`sidebarMenu,`  
`menuItem`

Shiny from



# EXERCISE

- ▶ Open `apps/dashboards/earthquakes_01.R`
- ▶ Turn this unstyled app into a Shiny Dashboard
- ▶ Refer to <https://rstudio.github.io/shinydashboard/>



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



# SOLUTION

Sample solution at **`apps/dashboards/earthquakes_02.R`**





# DASHBOARDS