Building dashboard UI for Shiny

Winston Chang and Joe Cheng

RStudio::Conf 2017

RStudio

Overview

- Static vs. dynamic dashboards
- flexdashboard
- shinydashboard

Dynamic dashboards (with Shiny)

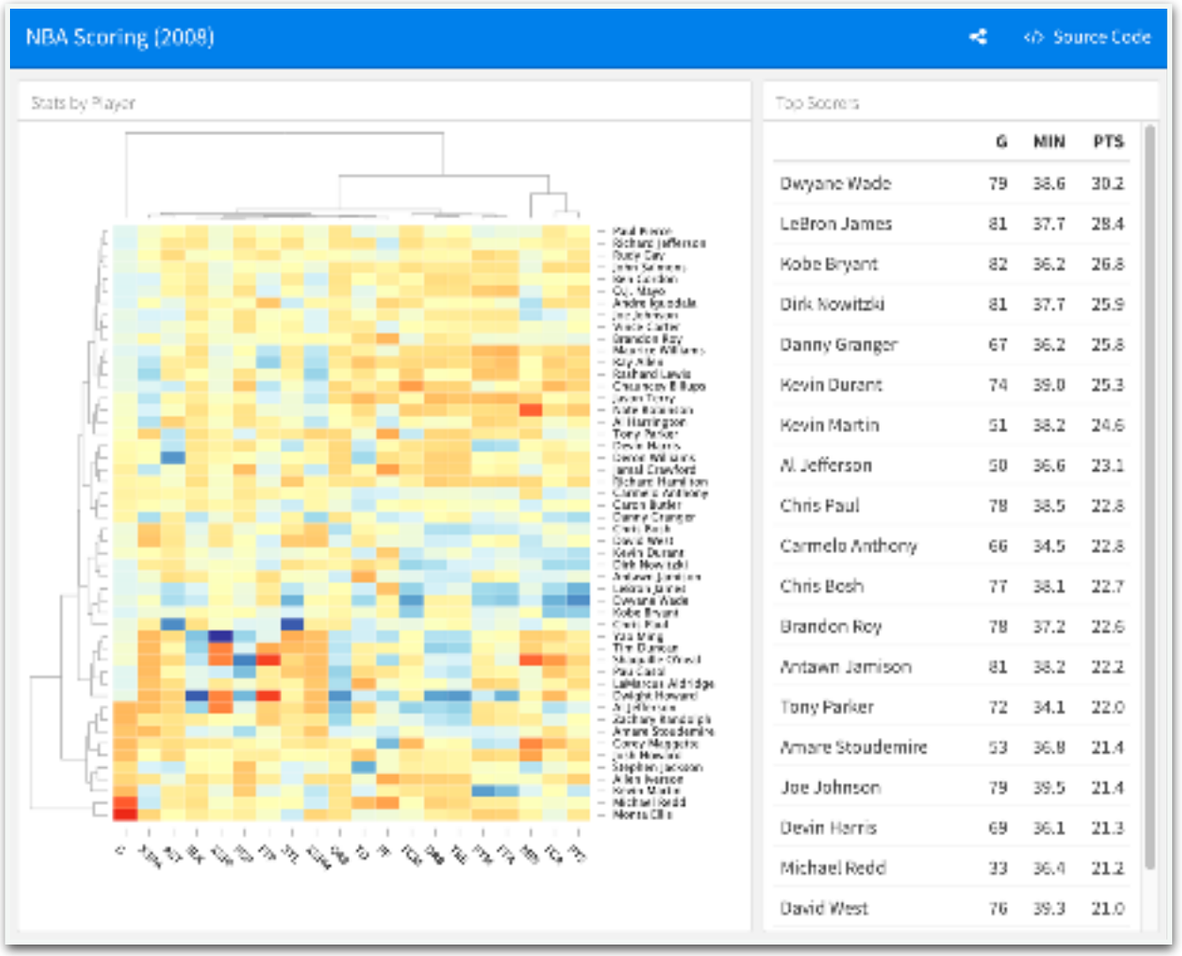
- Client web browser connects to an R session running on server
- User input causes server to do things and send information back to client
- Interactivity can be on client and server
- Can update data in real time
- User potentially can do anything that R can do
- Deploy to Shiny Server, RStudio Connect, or shinyapps.io

Static dashboards

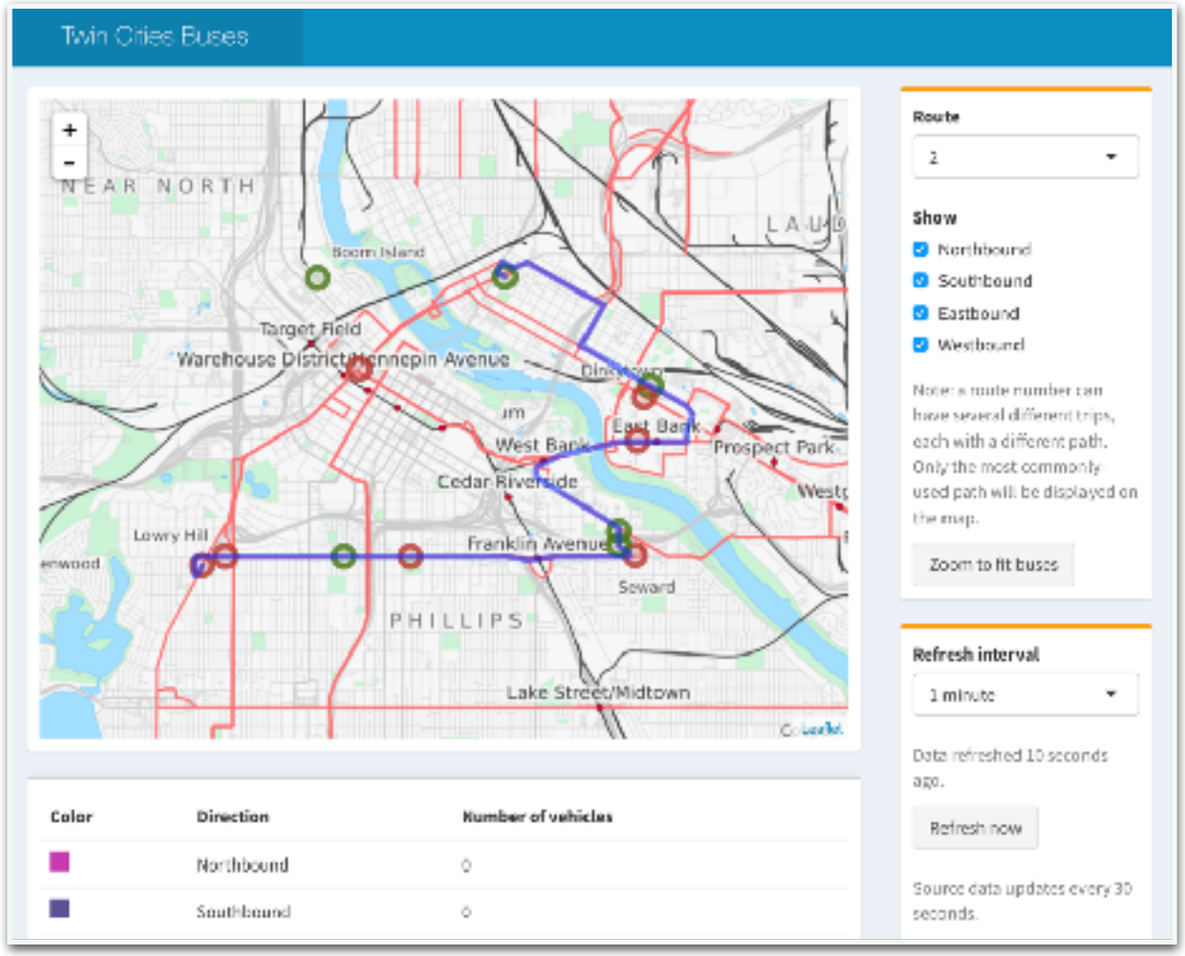
- R code runs once and generates an HTML file
- Generation of HTML file can be scheduled
- Deploy to RStudio Connect, RPubs, or any static web host

- Advantages
  - Simple deployment
  - Easily scalable to many users
- Limitations
  - Only suitable for cases where data does not update frequently
  - Interactivity (if any) is all in the web browser
  - User does not have access to full power of R

Two packages for dashboards



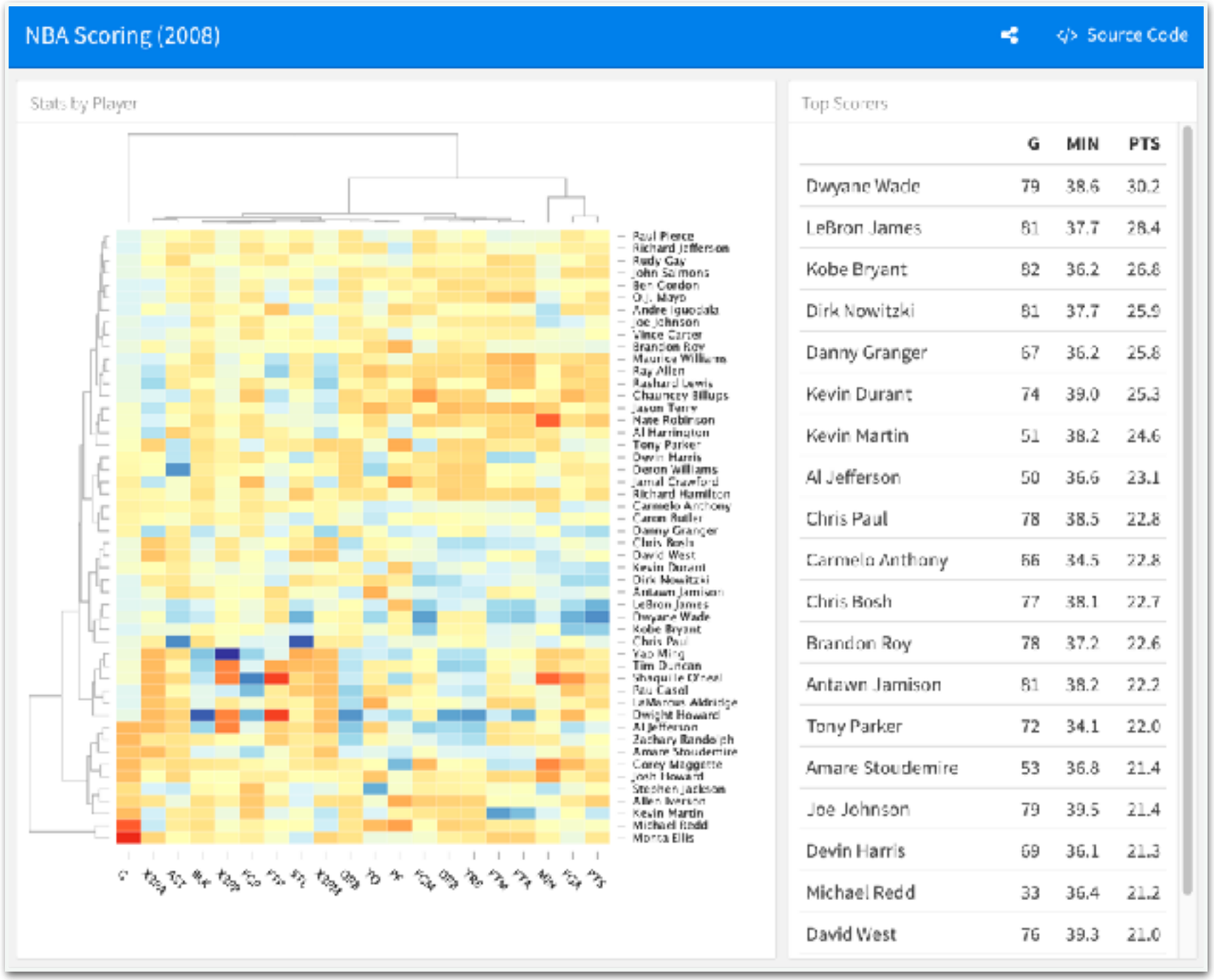
flexdashboard



shinydashboard

flexdashboard	shinydashboard
R Markdown	Shiny UI code
Super easy	Not quite as easy
Static or dynamic	Dynamic
CSS flexbox layout	Bootstrap grid layout

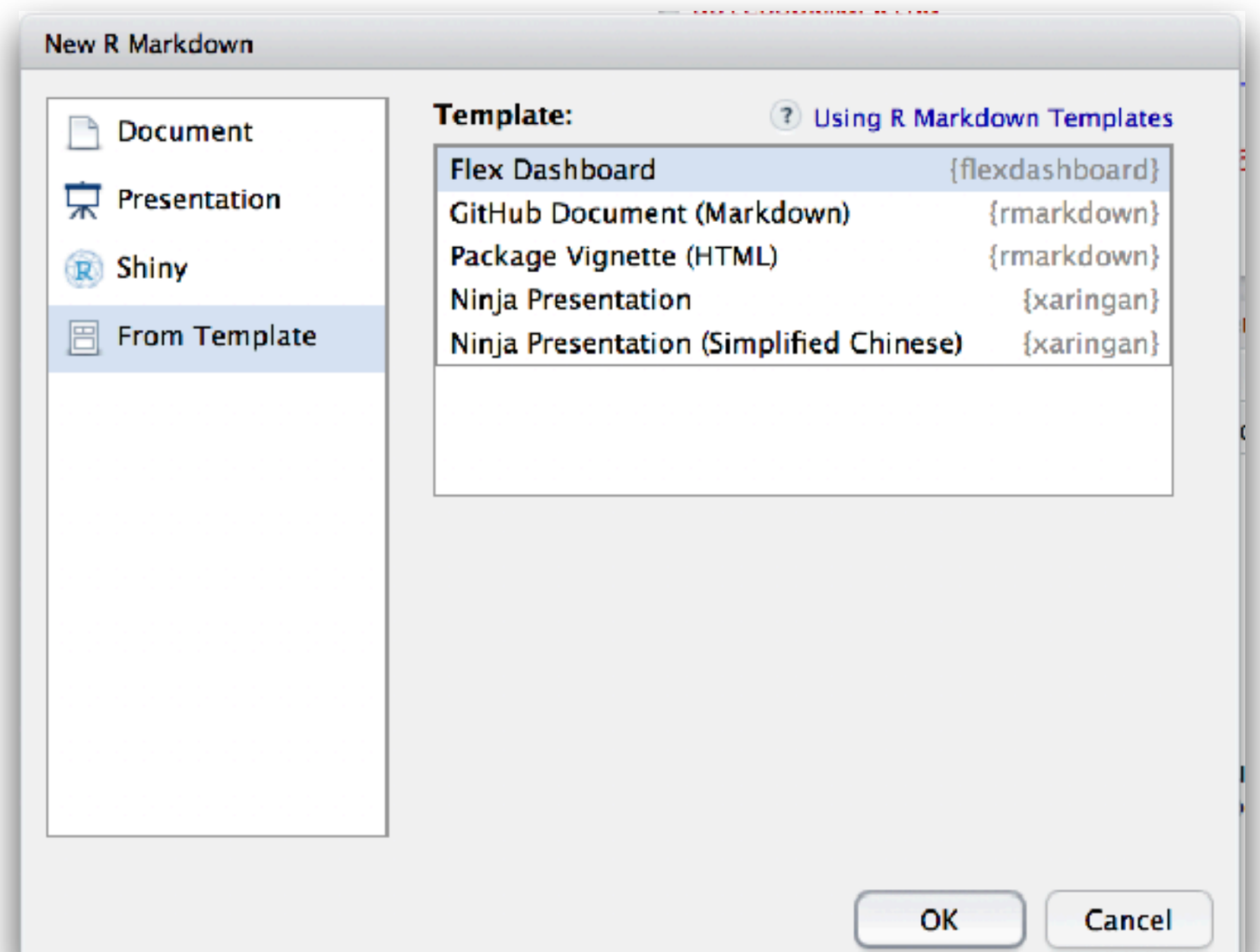
flexdashboard



http://rmarkdown.rstudio.com/flexdashboard/

Getting started

```
install.packages("flexdashboard")
```

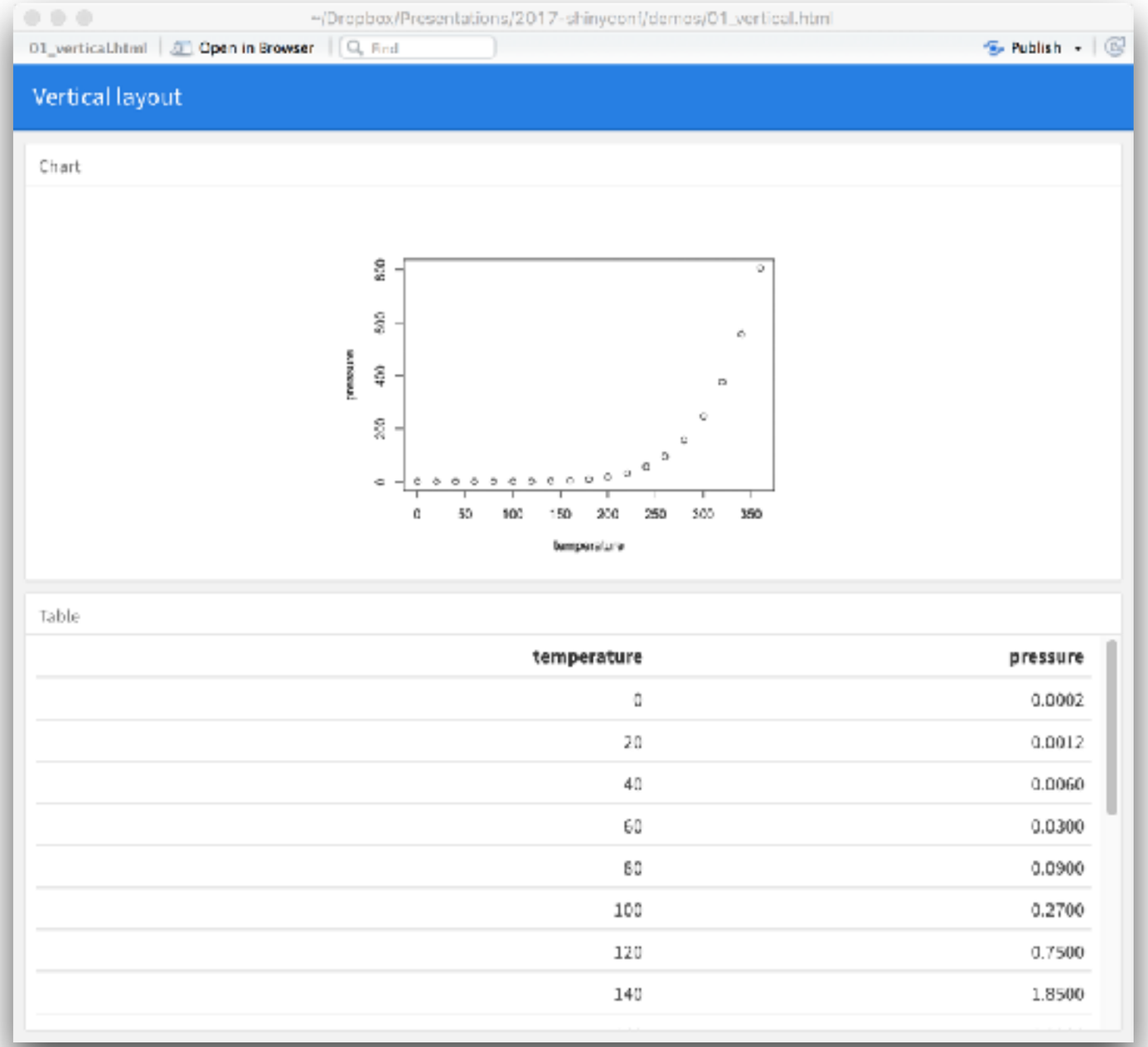


```
---
title: "Vertical layout"
output: flexdashboard::flex_dashboard
---

### Chart

```{r}
plot(pressure)
```

Table

```{r}
knitr::kable(pressure)
```
```



```
---
title: "Horizontal layout"
output: flexdashboard::flex_dashboard
---

Column 1
-----------------

### Chart

```{r}
plot(pressure)
```

Column 2

Table

```{r}
knitr::kable(pressure)
```
```

```
---
title: "Column layout"
output: flexdashboard::flex_dashboard
---

Column {data-width=4}
-------------------------------------

### Table

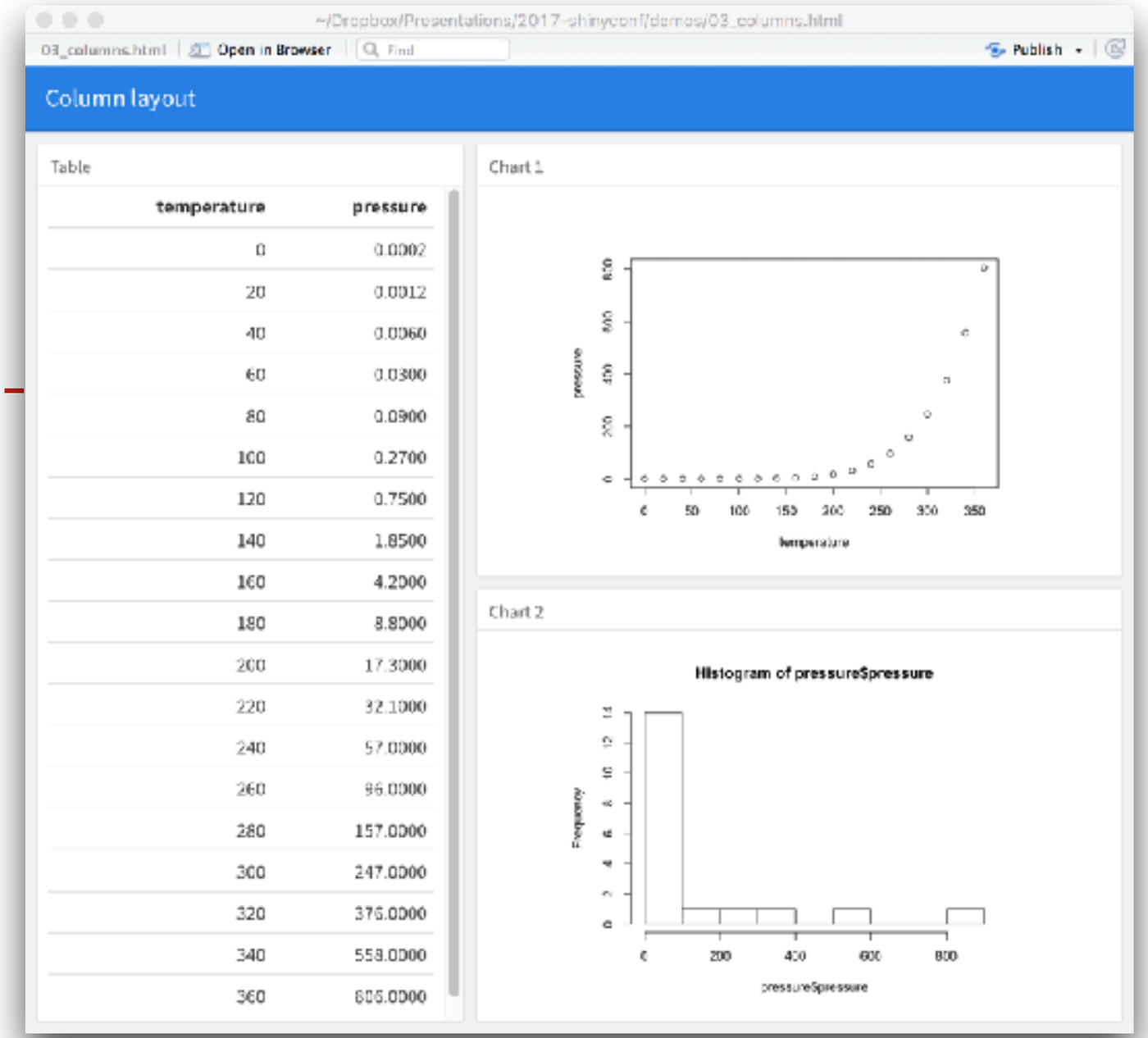
```{r}
knitr::kable(pressure)
```

Column {data-width=6}

Chart 1

```{r}
plot(pressure)

```

Chart 2

```{r}
hist(pressure$pressure)
```
```



```
---
title: "Row layout"
output:
  flexdashboard::flex_dashboard:
    orientation: rows
---

Row
-----------------------------------------

### Table

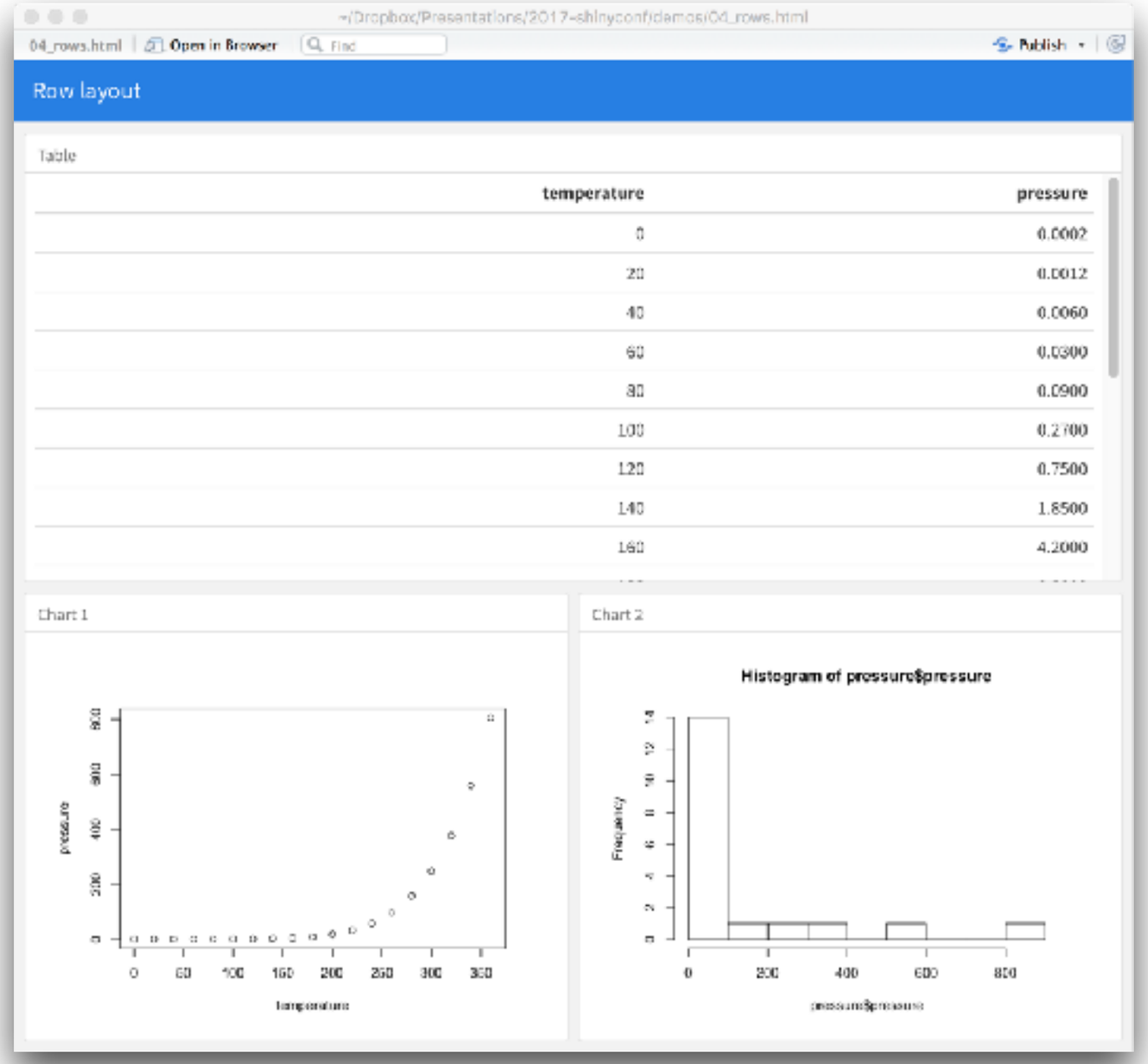
```{r}
knitr::kable(pressure)
```

Row

Chart 1

```{r}
plot(pressure)

```

Chart 2

```{r}
hist(pressure$pressure)
```
```



```
---
title: "Scrolling layout"
output:
  flexdashboard::flex_dashboard:
    orientation: columns
    vertical_layout: scroll
---

Column
-------------------------------------

### Table

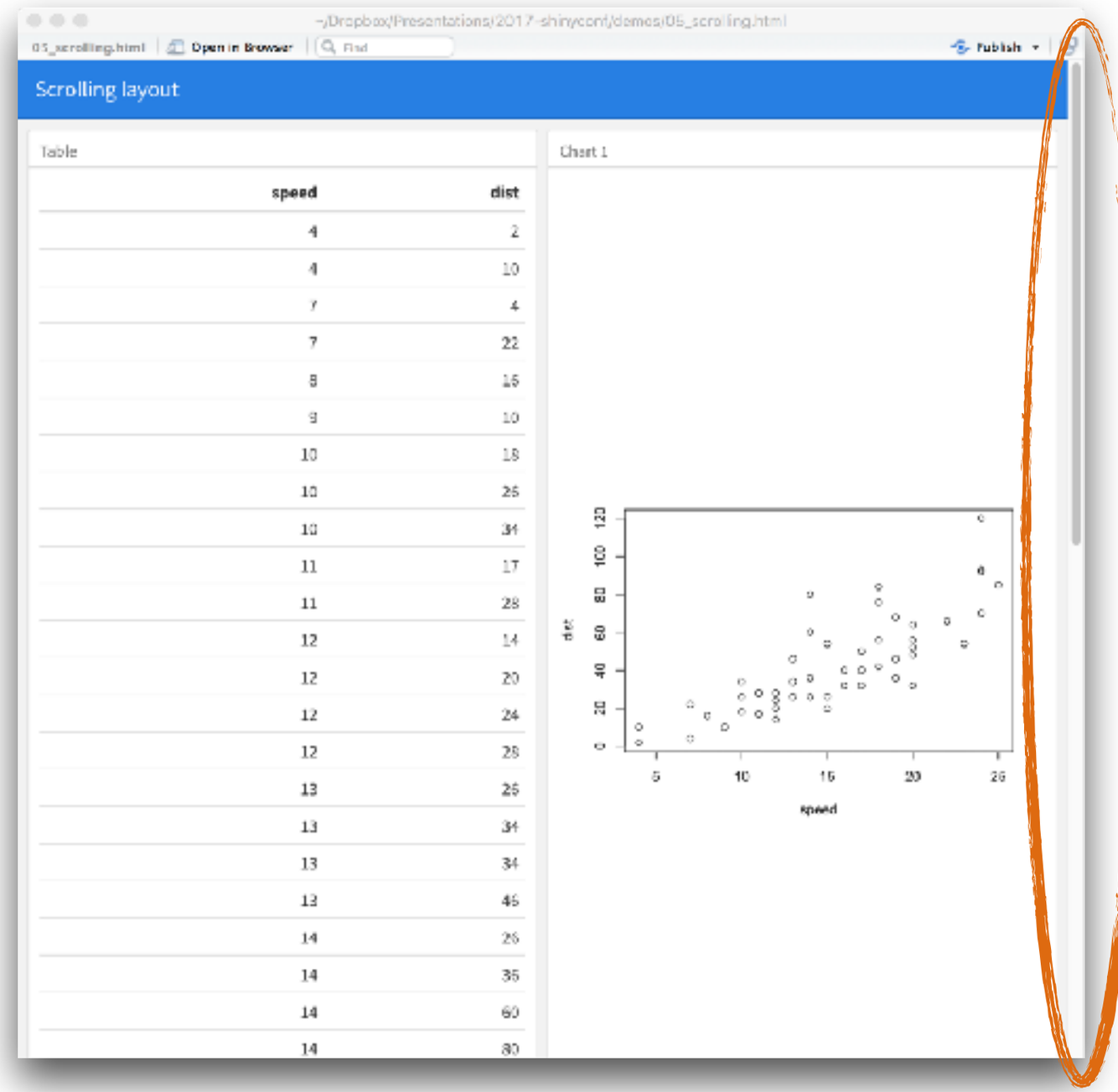
```{r}
knitr::kable(cars)
```


Column

Chart 1

```{r}
plot(cars)
```

Chart 2

```{r}
hist(cars$dist)
```
```



Layout summary

- Level 2 header: defines row/column.
- Level 3 header: defines a box within a row/column.
- By default, all content scales to fit in the browser window (by using CSS flexbox layout).
- data-width and data-height specify *proportions* of space used by a row, column, or box.

```
Row
-----------

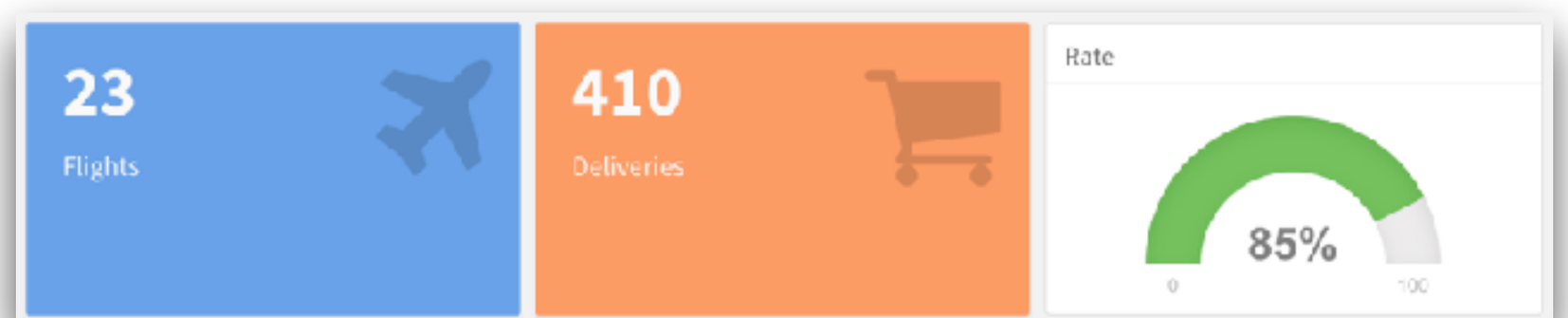
### Flights

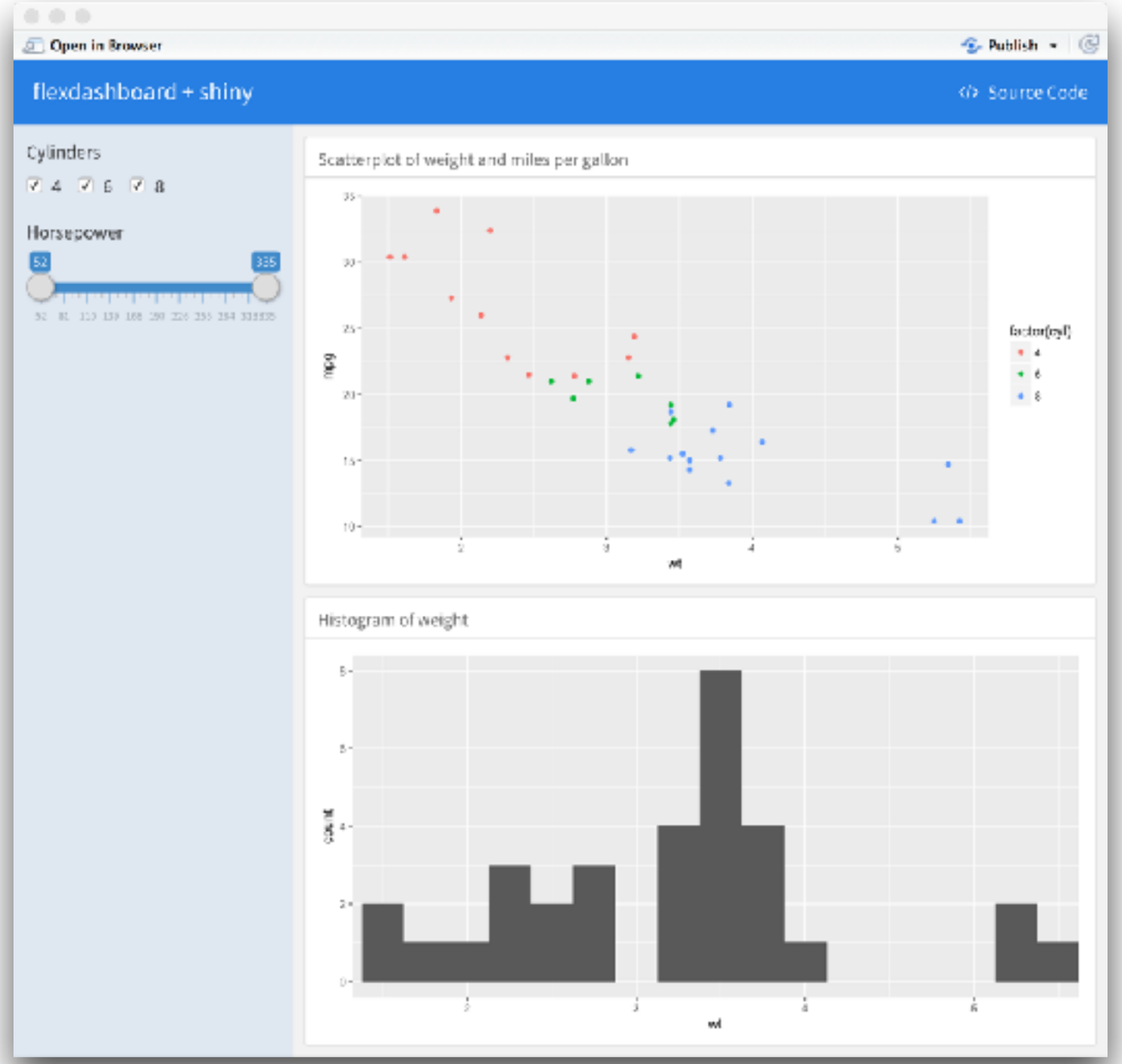
```{r}
flights <- 23
valueBox(flights, icon = "fa-plane")
```

Deliveries

```{r}
deliveries <- 410
valueBox(deliveries, color = "warning", icon = "fa-shopping-cart")
```

Rate

```{r}
rate <- 85
gauge(rate, min = 0, max = 100, symbol = '%',
  gaugeSectors(
    success = c(80, 100),
    warning = c(40, 79),
    danger = c(0, 39)
  )
)
```
```

Icon from Font Awesome



Your turn

- Make a dashboard that uses a built-in data set (or one of your own)
- Show a table of the data (with knitr::kable)
- Show a plot of the data
- Add in some valueBoxes and gauges

Publishing static dashboards

- Static web host
- RPubs
- RStudio Connect (also supports scheduling)

Static dashboard limitations

- Graphics don’t re-render when resized (need to use JS graphics libraries to do live resizing)
- Doesn’t work with live, changing data
- User does not have access to R
- Very limited interactivity

Dynamic dashboards: flexdashboard + shiny

Shiny Document mode

```
---
title: "flexdashboard + shiny"
output: flexdashboard::flex_dashboard
runtime: shiny
---
```

```
Inputs {.sidebar}
-----------------------------------

```{r}
checkboxGroupInput("cyl", "Cylinders",
  choices = c("4", "6", "8"),
  selected = c("4", "6", "8"),
  inline = TRUE
)

sliderInput("hp",
  "Horsepower",
  min = min(mtcars$hp),
  max = max(mtcars$hp),
  value = range(mtcars$hp)
)
```
```



```
Outputs
---------------------

### Scatterplot of weight and miles per gallon

```{r}
mpg_subset <- reactive({
 mtcars[mtcars$hp >= input$hp[1] &
        mtcars$hp <= input$hp[2] &
        mtcars$cyl %in% input$cyl, ]
})

renderPlot({
  ggplot(mpg_subset(),
         aes(x=wt, y=mpg, color=factor(cyl))) +
    geom_point() +
    coord_cartesian(
      xlim = range(mtcars$wt),
      ylim = range(mtcars$mpg))
})
```
```

Shiny Document summary

- Add **`runtime: shiny`** to header.
- Add inputs in code chunks.
- Add **`render`*`Xyz`*** functions in code chunks. No need for **`output$x <-`** assignment, or for ***`xyz`*`Output`** functions.

Your turn

- Turn your static dashboard into a dynamic one
- Add a sidebar and inputs to filter your data

Shiny Document drawbacks

- Start-up time: knits document every time someone visits it.
- Resizing can trigger re-knit.
- Auto-reconnection doesn’t work.

- The solution: Pre-rendered Shiny Documents

Prerendered Shiny Documents

- **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.

Rendering phase

```
Inputs {.sidebar}
-----------------------------

```{r context="render"}
checkboxGroupInput("cyl", "Cylinders",
  choices = c("4", "6", "8"),
  selected = c("4", "6", "8"), inline = TRUE
)

sliderInput("hp", "Horsepower",
  min = min(mtcars$hp),
  max = max(mtcars$hp),
  value = range(mtcars$hp)
)
```
```

Serving phase

```
Outputs
-----------------------

### Scatterplot of weight and miles per gallon

```{r context="server"}
output$scatter <- renderPlot({
  ggplot(mpg_subset(),
         aes(x=wt, y=mpg, color=factor(cyl))) +
    geom_point() +
    coord_cartesian(
      xlim = range(mtcars$wt),
      ylim = range(mtcars$mpg))
})
```

```{r context="render"}
plotOutput("scatter")
```
```

Server code

UI code

plotOutput needed

Contexts for shiny_prerendered

- **`"render"`:** Runs in rendering phase (like ui.R)
- **`"server"`:** Runs in serving phase (like code in server.R, inside server function)
- **`"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, before any sessions start (like code in server.R, outside of server function).

Your turn

- Convert your Shiny Document into a pre-rendered one.

Automatic reconnections

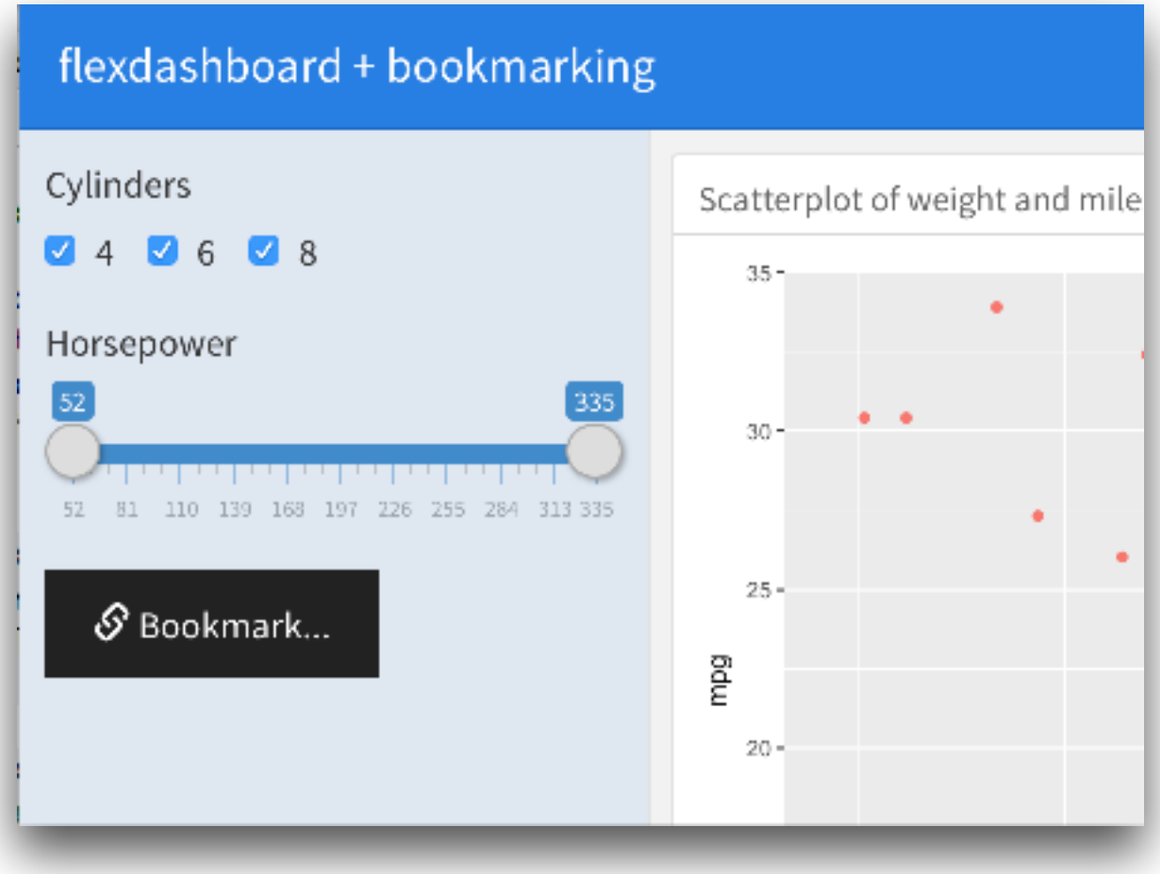
- Client browsers can automatically reconnect after being disconnected due to network problems.
- Requires shiny_prerendered.

```{r context="server"}
session$allowReconnect(TRUE)
```

If you want to test reconnections in a local R session, use: `session$allowReconnect("force")`

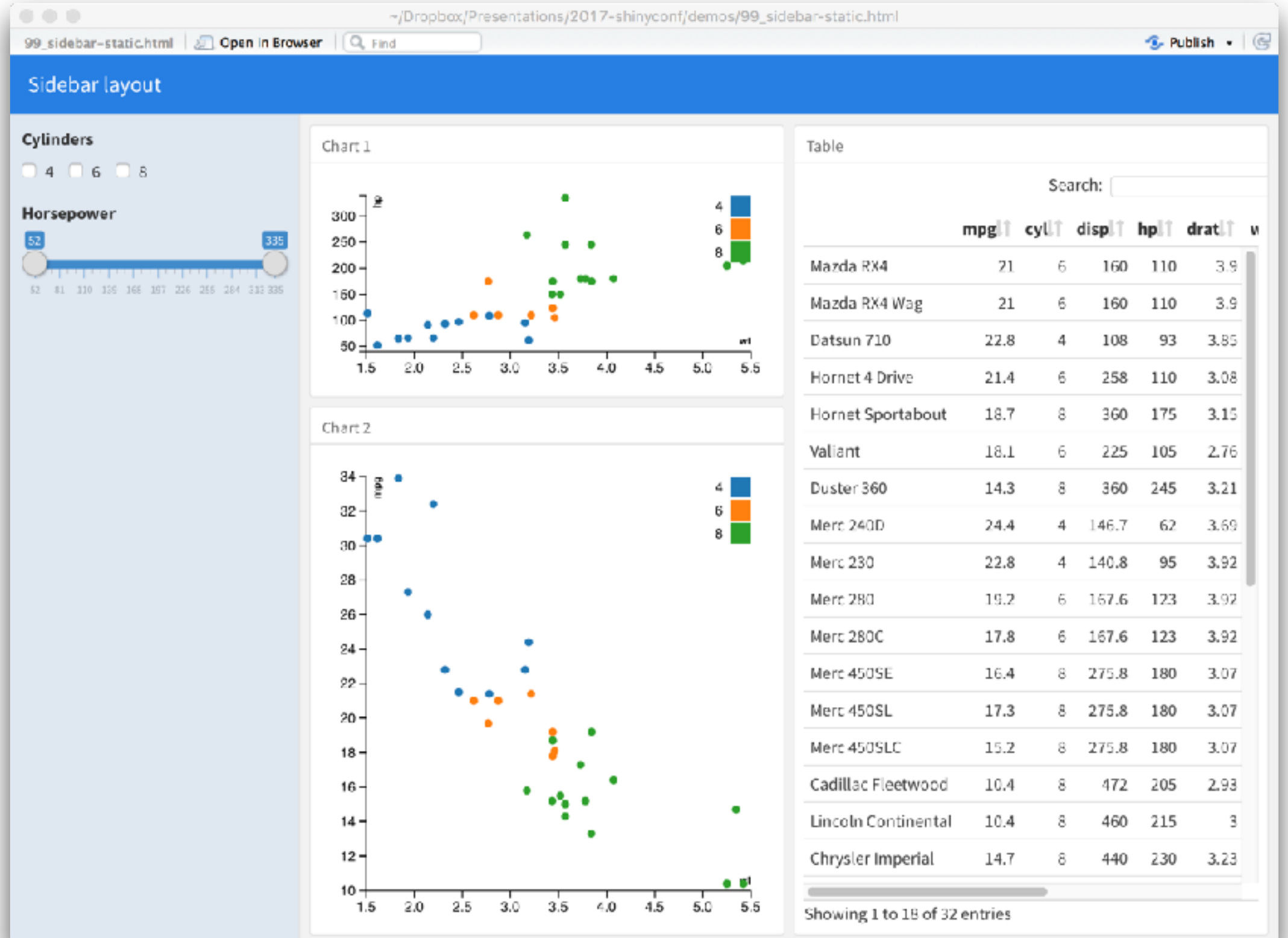
Bookmarking

```{r}
enableBookmarking("url")

bookmarkButton()
```

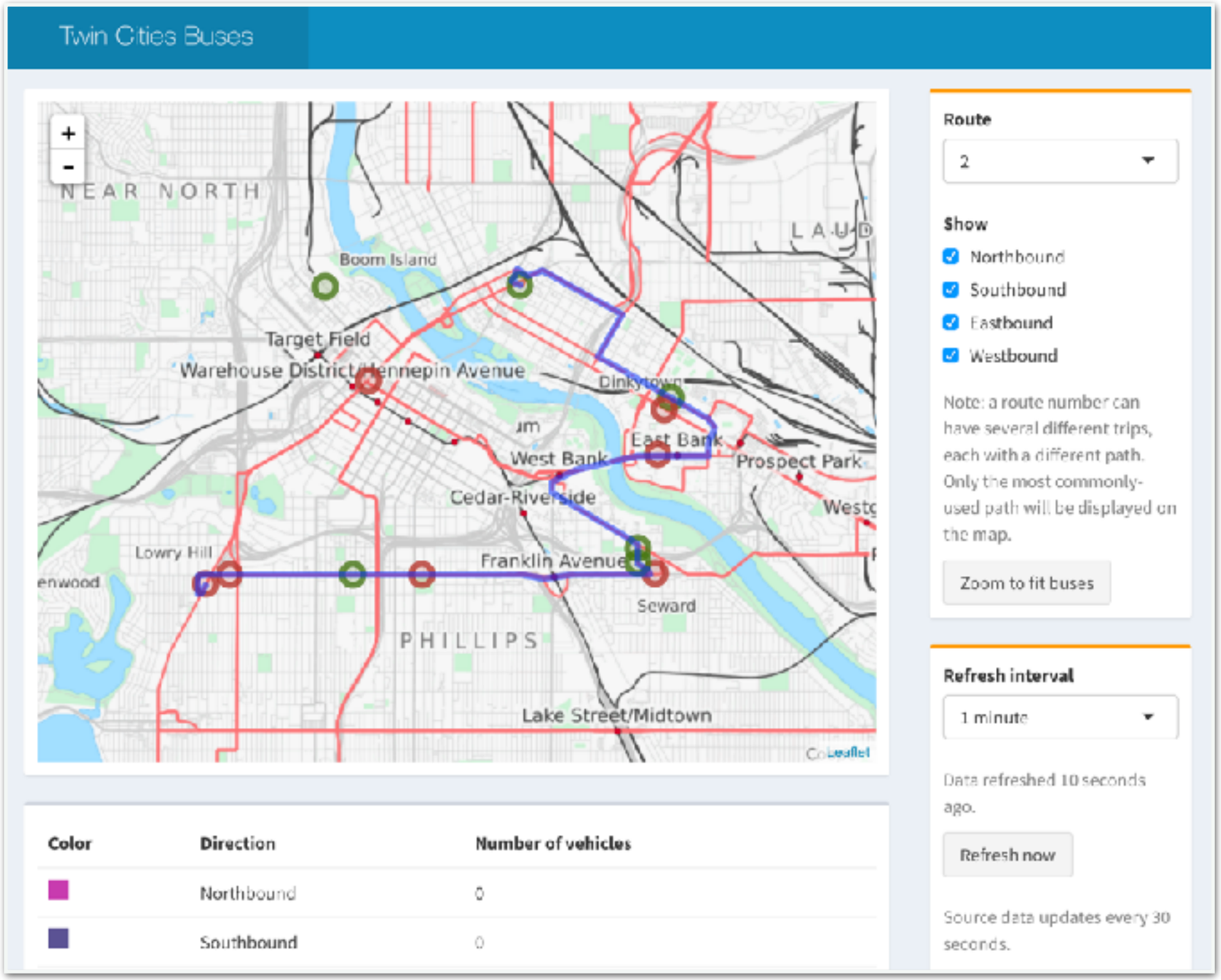


Currently does **not** work with shiny_prerendered.

One more thing: interactive static dashboards



shinydashboard



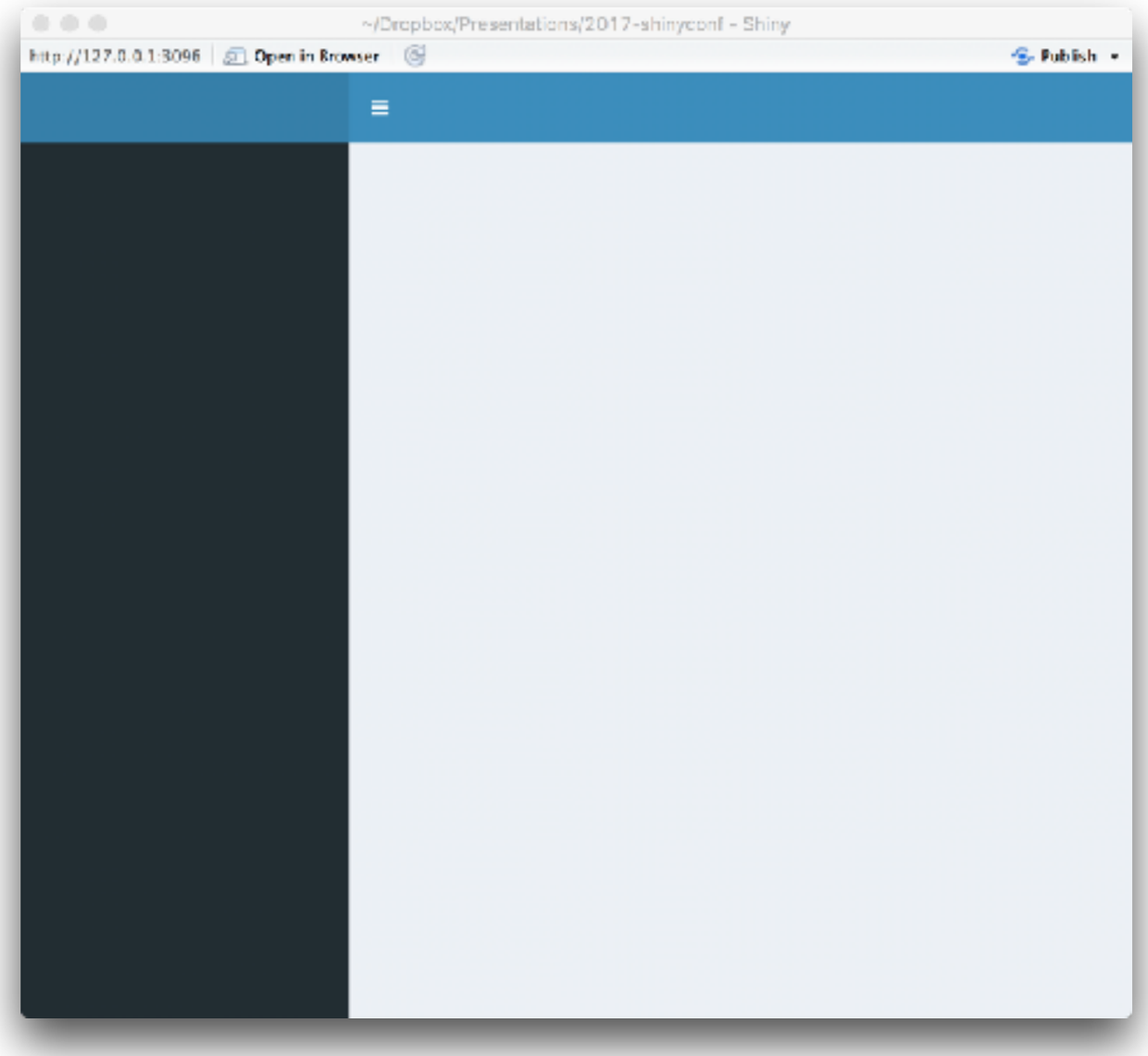
https://rstudio.github.io/shinydashboard/

What is shinydashboard?

- The UI for Shiny is built on the Bootstrap web framework.
- Shinydashboard is a theme for Shiny, built on top of Bootstrap.

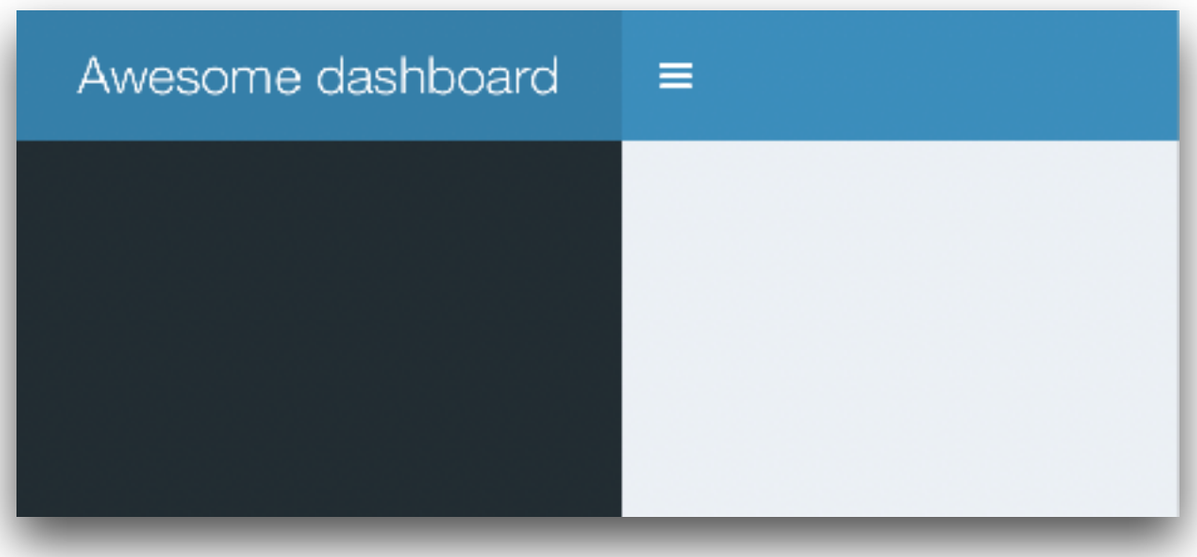
```
install.packages("shinydashboard")
```

```r
## ui.R ##
library(shiny)
library(shinydashboard)

dashboardPage(
  dashboardHeader(),
  dashboardSidebar(),
  dashboardBody()
)
```



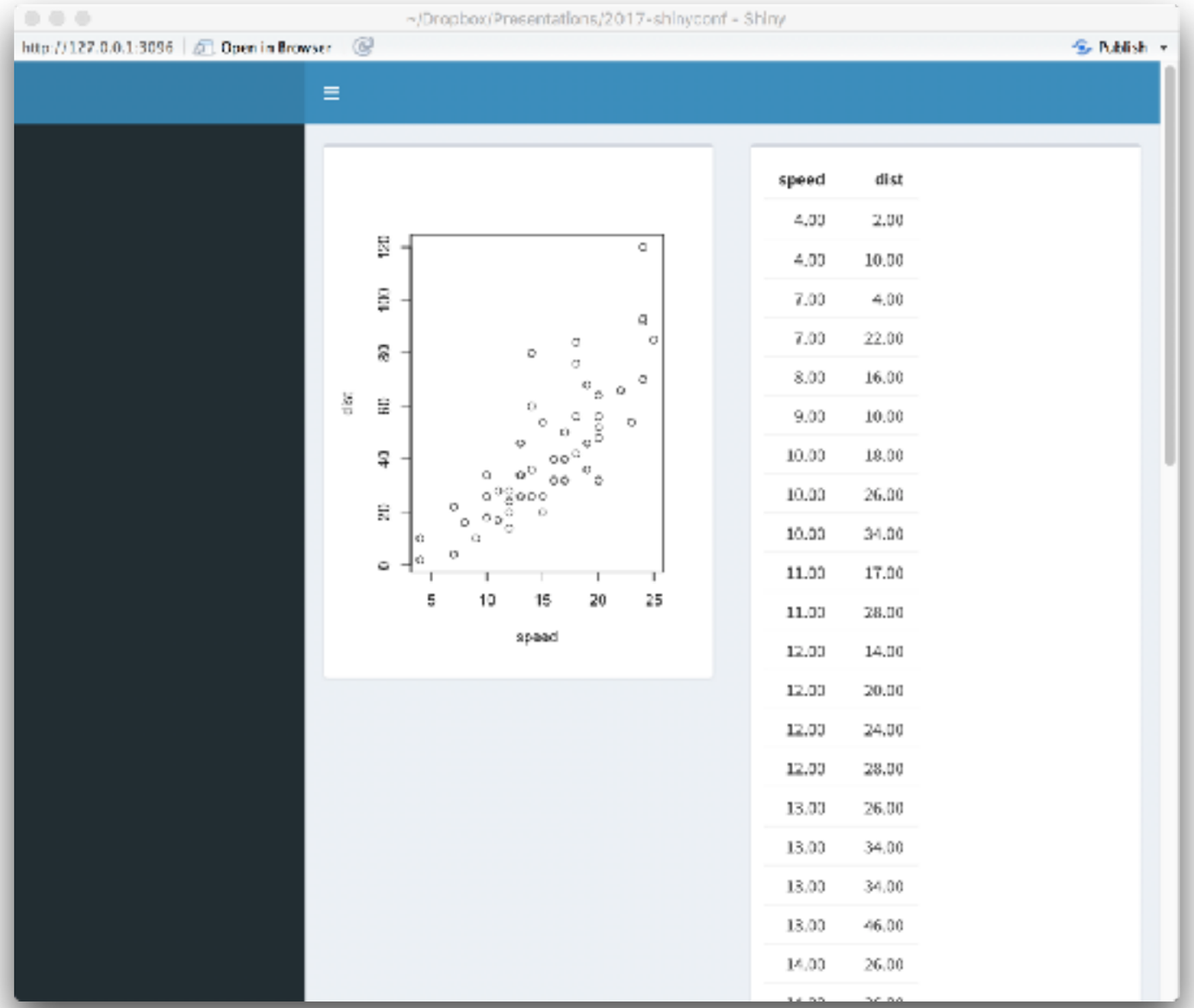

```r
## app.R ##
library(shiny)
library(shinydashboard)

ui <- dashboardPage(
  dashboardHeader(),
  dashboardSidebar(),
  dashboardBody()
)

server <- function(input, output)
{ }

shinyApp(ui, server)
```

```
dashboardPage(
  dashboardHeader(title = "Awesome dashboard"),
  dashboardSidebar(),
  dashboardBody()
)
```

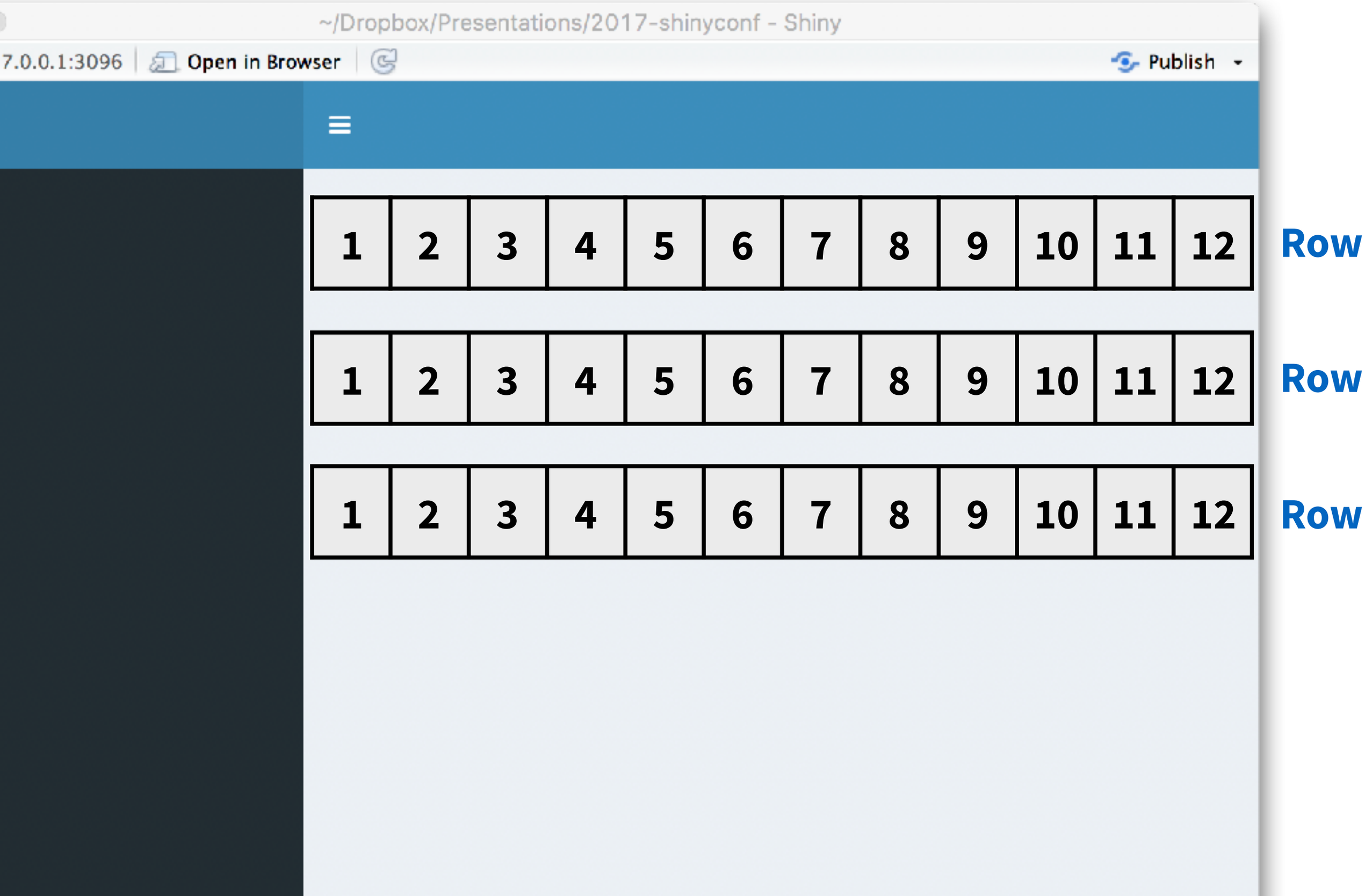


Row-based layout

```
ui <- dashboardPage(
  dashboardHeader(),
  dashboardSidebar(),
  dashboardBody(
    fluidRow(
      box(
        plotOutput("plot1")
      ),
      box(
        tableOutput("table1")
      )
    )
  )
)

server <- function(input, output) {
  output$plot1 <- renderPlot({
    plot(cars)
  })
 output$table1 <- renderTable({
    cars
  })
}

shinyApp(ui, server)
```



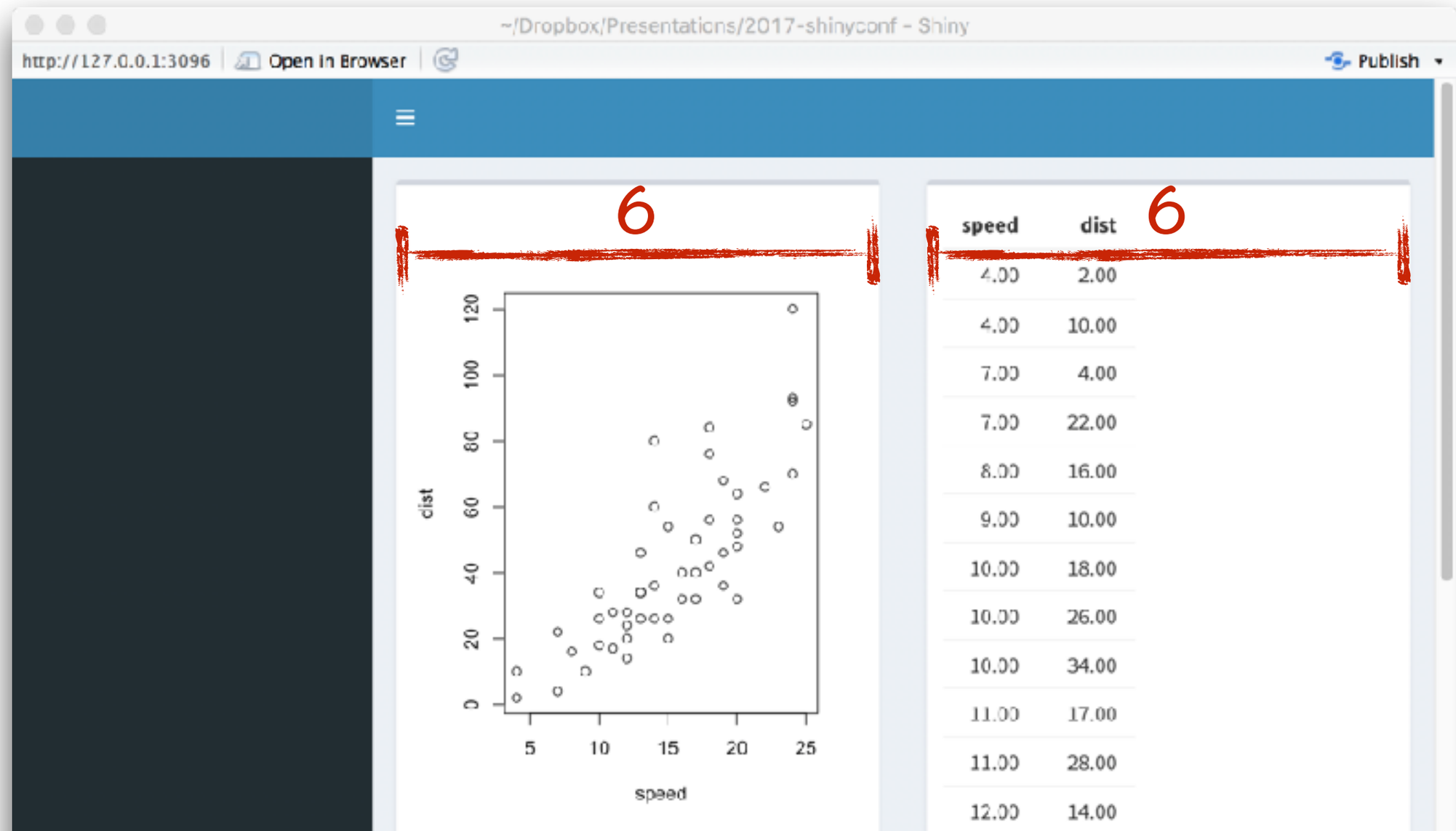
Bootstrap grid



Row-based layout

```
dashboardBody(
   fluidRow(
     box(width = 6,
       plotOutput("plot1")
     ),
     box(width = 6,
       tableOutput("table1")
     )
   )
)
```

Default width is 6

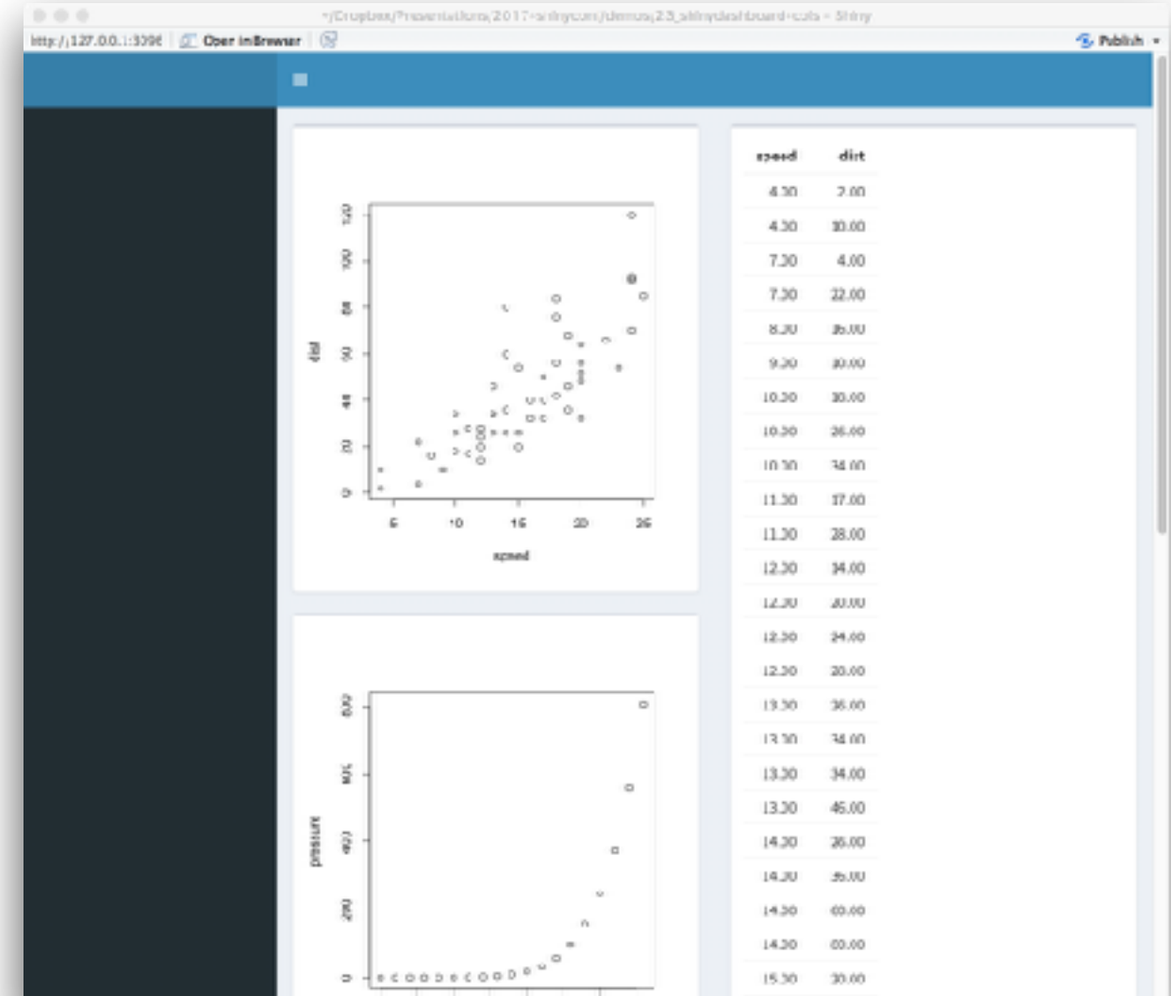


Column-based layout

```
dashboardBody(
  fluidRow(
    column(width = 6,
      box(plotOutput("plot1"), width = NULL),
      box(plotOutput("plot2"), width = NULL)
    ),
    column(width = 6,
      box(tableOutput("table1"), width = NULL)
    )
  )
)
```

Must specify column width

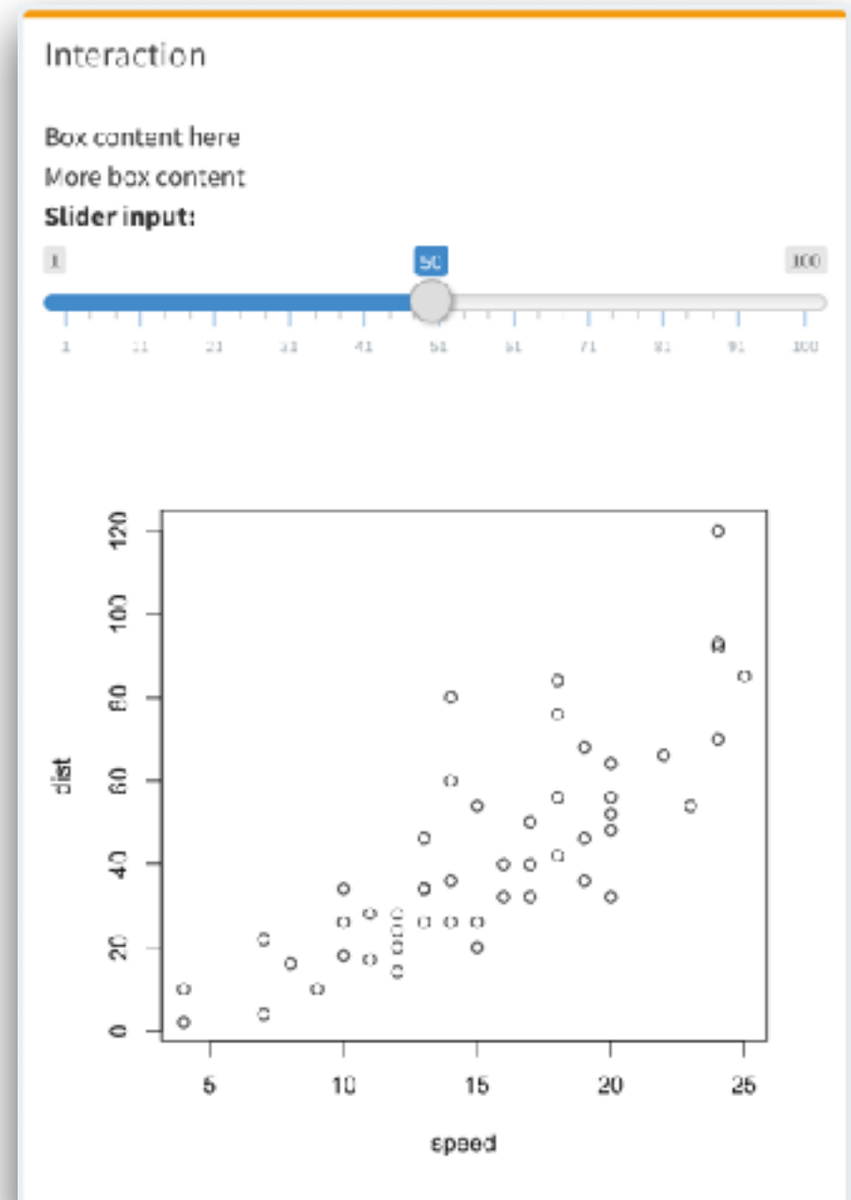
Box width must be NULL



CSS flexbox	Bootstrap grid
Width fits to container	Width fits to container
Height fits to container	Height does not fit to container
Width allocated proportionally for any numeric values	Width allocated in 12 pieces
Height allocated proportionally for any numeric values	Height determined by content

More about boxes

```
box(
  title = "Interaction",
  status = "warning",
  "Box content here", br(), "More box content",
  sliderInput("s", "Slider input:", 1, 100, 50),
  plotOutput("plot1")
)
```



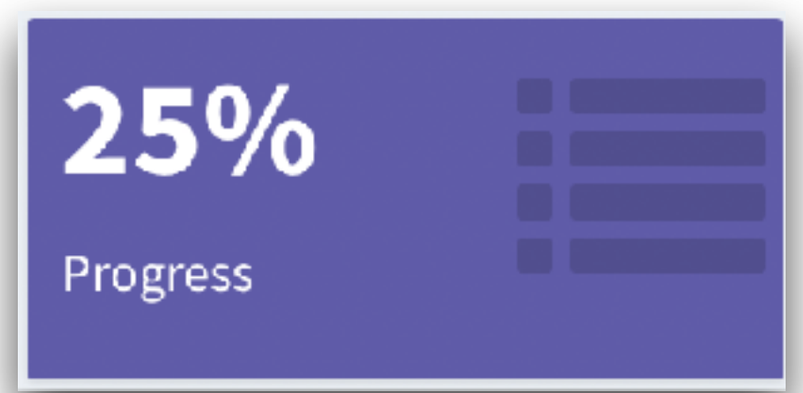
Your turn

- Create a shinydashboard with boxes
- Fill in those boxes with the same content that you used in your flexdashboard

valueBoxes

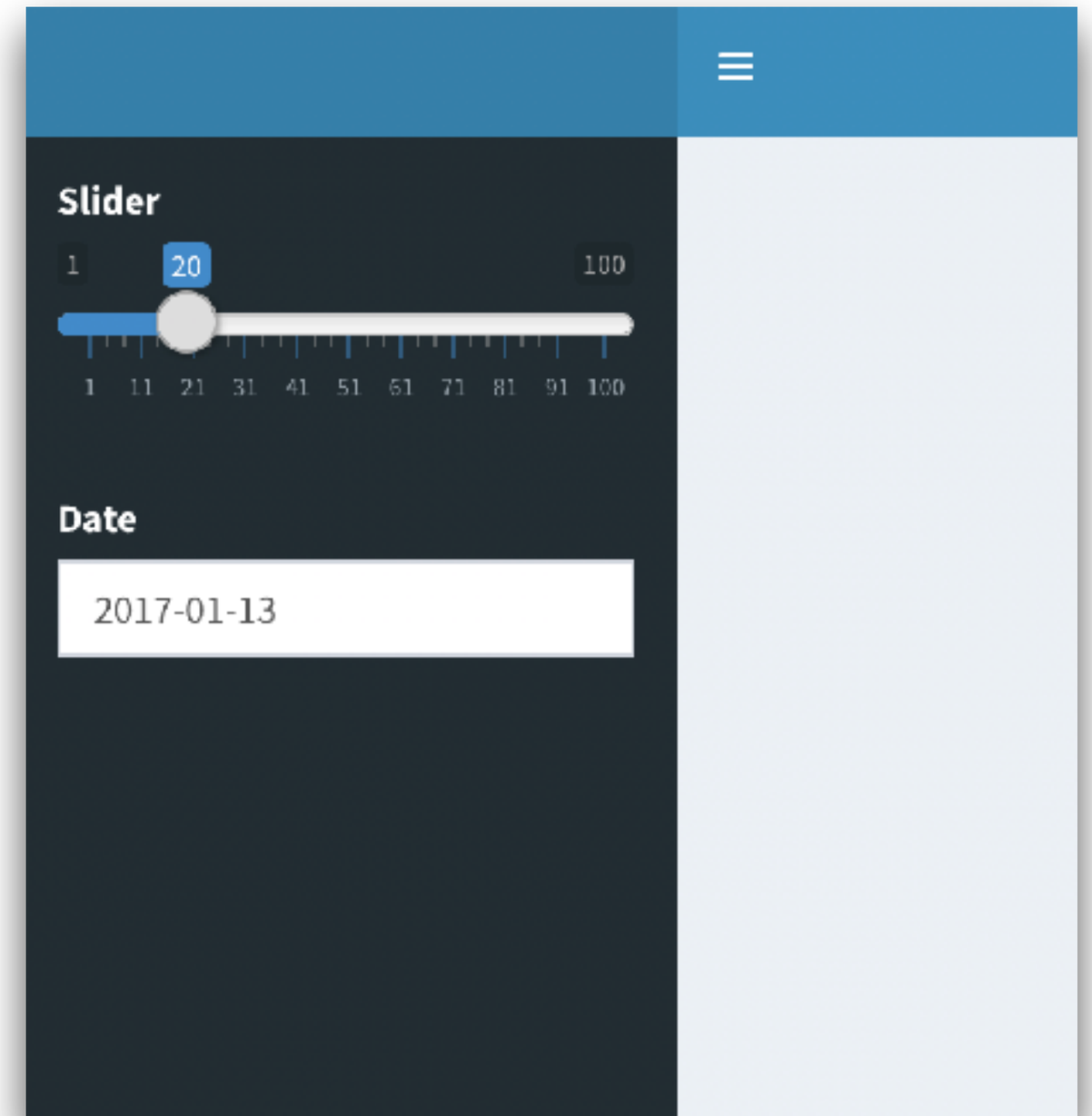
```
## UI ##
fluidRow(
  valueBoxOutput("value1")
)

## Server function ##
output$value1 <- renderValueBox({
  valueBox(
    width = 4,
    paste0(progress(), "%"), "Progress",
    icon = icon("list"),
    color = "purple"
  )
})
```

Default width is 4



Sidebar: inputs

```
dashboardSidebar(
  sliderInput("s", "Slider", 1, 100, 20),
  dateInput("d", "Date")
)
```



Your turn

- Add valueBoxes and inputs to your dashboard so that it works like your flexdashboard.

- **flexdashboard**
  - For static and dynamic dashboards
  - Easy to get started with if you know R Markdown
- **shinydashboard**
  - For dynamic dashboards
  - Easy to use if you know Shiny UI layout
  - Can be extended with arbitrary HTML

http://rmarkdown.rstudio.com/flexdashboard/

https://rstudio.github.io/shinydashboard/