```
}

data_props <- combine_data_props(x$marks)
data_ids <- names(data_props)
data_table <- x$data[data_ids]

# Collapse each list of scale objects into one scale object.
x <- collapse_scales(x)
scale_data_table <- scale_domain_data(x)

# Wrap each of the reactive data objects in another reactive which returns
# only the columns that are actually used, and adds any calculated columns
# that are used in the props.
data_table <- active_props(data_table, data_props)

# From an environment containing data_table objects, get static data for the
# specified ids.
```

Values

data_ids	"diamonds0/bin1/stack2"
data_props	List of 1
dynamic	FALSE

Traceback

```
as.vega.ggvis(x, FALSE) at vega.R:29
as.vega(x, FALSE) at vega.R:11
view_static(x, ...) at print.R:67
print.ggvis(c("list()", "list(diamonds0 = function () \nstatic_data)", "li
```

```
> ggvis(diamonds, x = ~price, y = ~color)
Guessing layer_histograms()
Guessing binwidth = 1
Called from: eval(expr, envir, enclos)
Browse[1]> n
debug at /Users/jmcphers/r/ggvis/R/vega.R#29: data_table <- x$data[data_ids]
Browse[2]>
```

UNDERSTANDING UI

Shiny from RStudio

- Web application UI is ultimately HTML/CSS/JavaScript
- Let R users write user interfaces using a simple, familiar-looking API…
- …but no limits for advanced users

Ladder of progression

LADDER OF UI PROGRESSION

- Step 1. Shiny built-in inputs/outputs and layouts (sidebarLayout, navbarPage, tabsetPanel)
- Step 2. Use functions from external packages (shinythemes, shinydashboard, shinybs)
- Step 3. Use tag objects, write UI functions **Our focus today**
- Step 4. Author HTML templates
- Step 5. Create custom inputs/outputs, wrap existing CSS/JS libraries and frameworks

High level view

MULTIPLE LEVELS OF ABSTRACTION

High-level funcs

```
fluidRow(...)
```

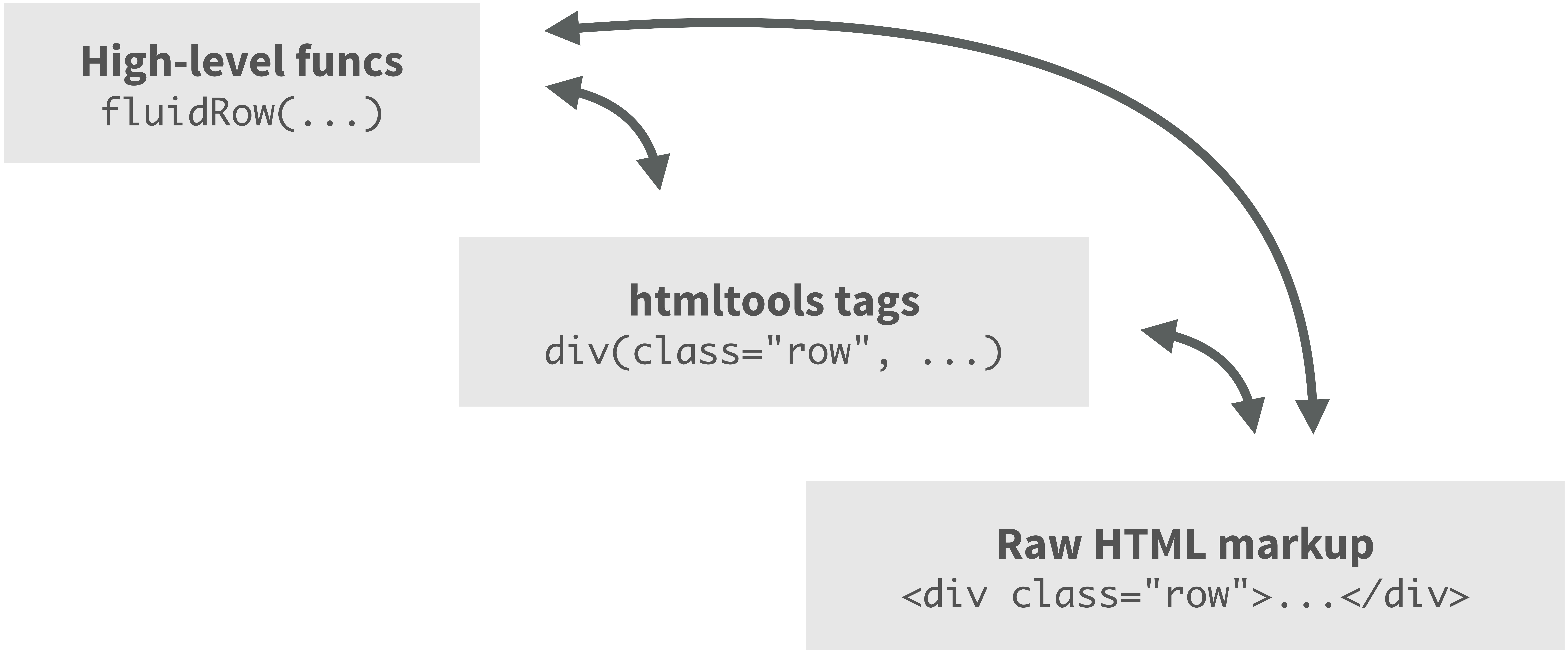
htmltools tags

```
div(class="row", ...)
```

Raw HTML markup

```
<div class="row">...</div>
```

MIX AND MATCH FREELY



High-level funcs

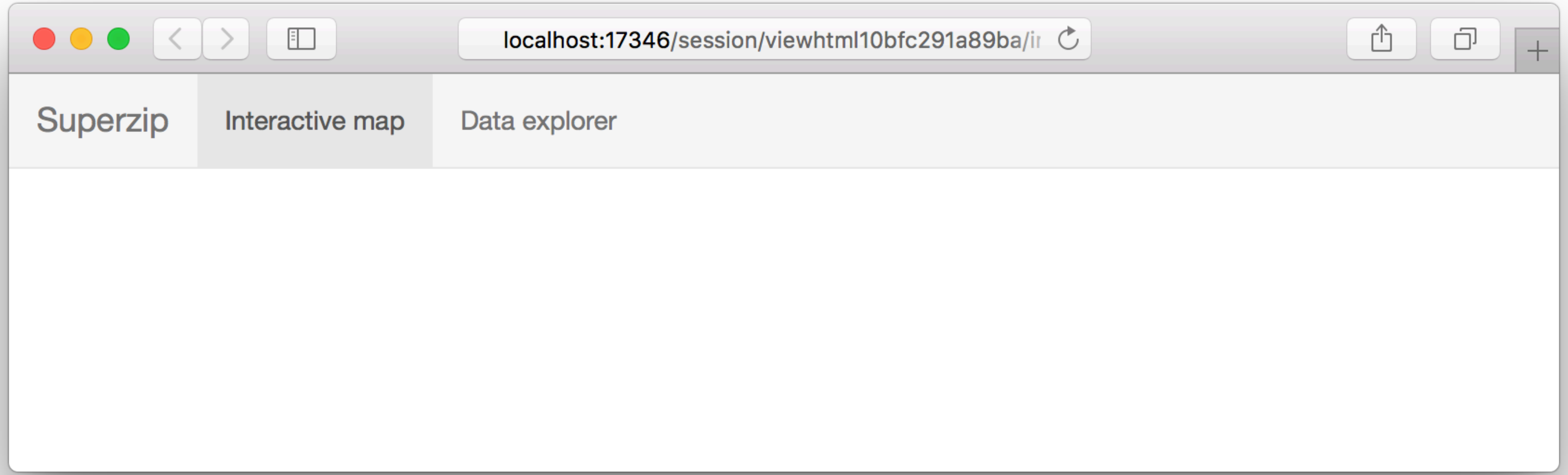
```
fluidRow(...)
```

htmltools tags

```
div(class="row", ...)
```

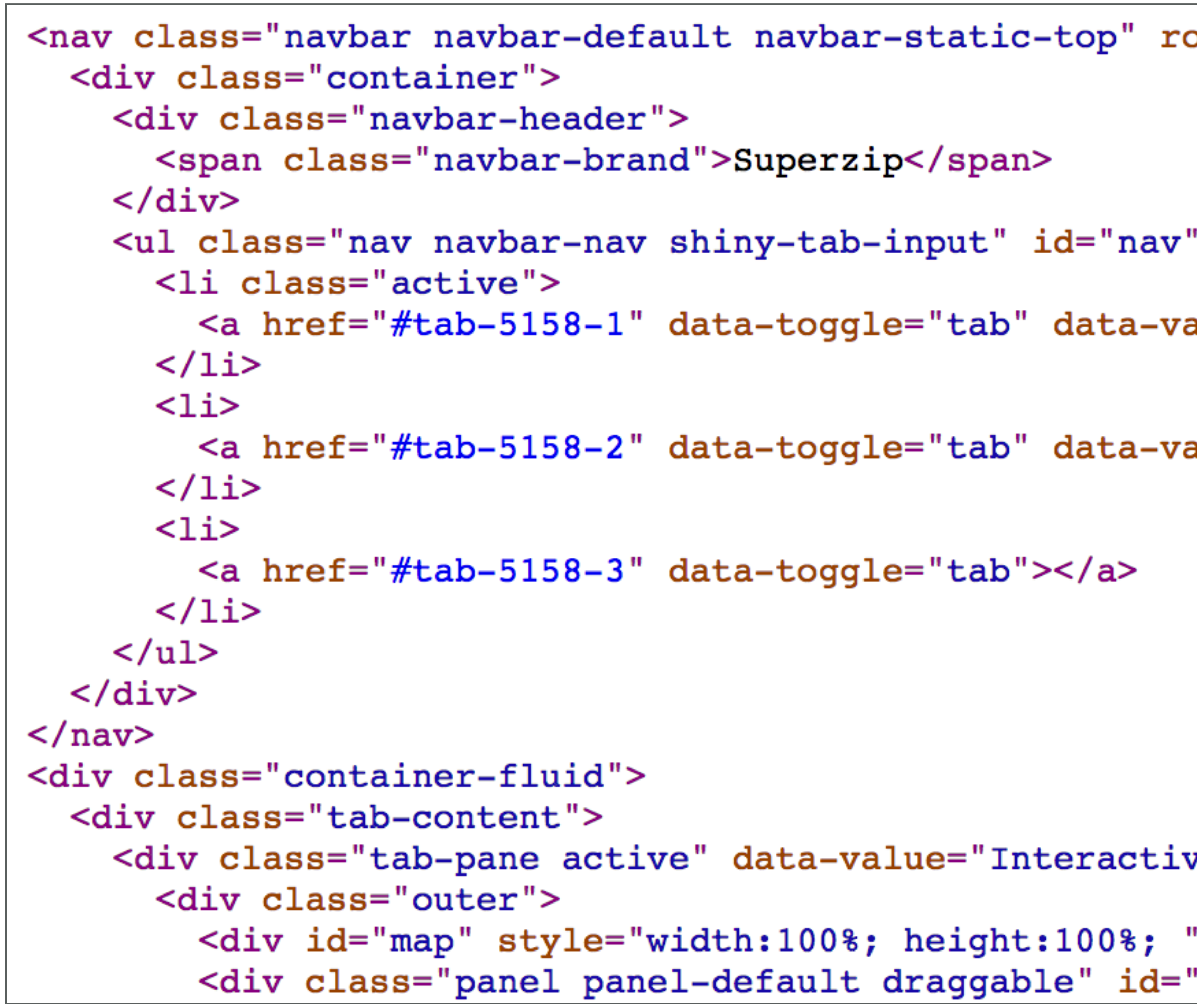
Raw HTML markup

```
<div class="row">...</div>
```


localhost:17346/session/viewhtml10bfc291a89ba/ir
Superzip
Interactive map
Data explorer

RAW HTML

- Pros
  - Can do anything that's possible in a web page
  - Comfortable for designers, web developers
- Cons
  - Unfamiliar for many R users
  - Potentially lots of HTML needed for conceptually simple tasks
  - CSS/JavaScript dependencies must be handled manually



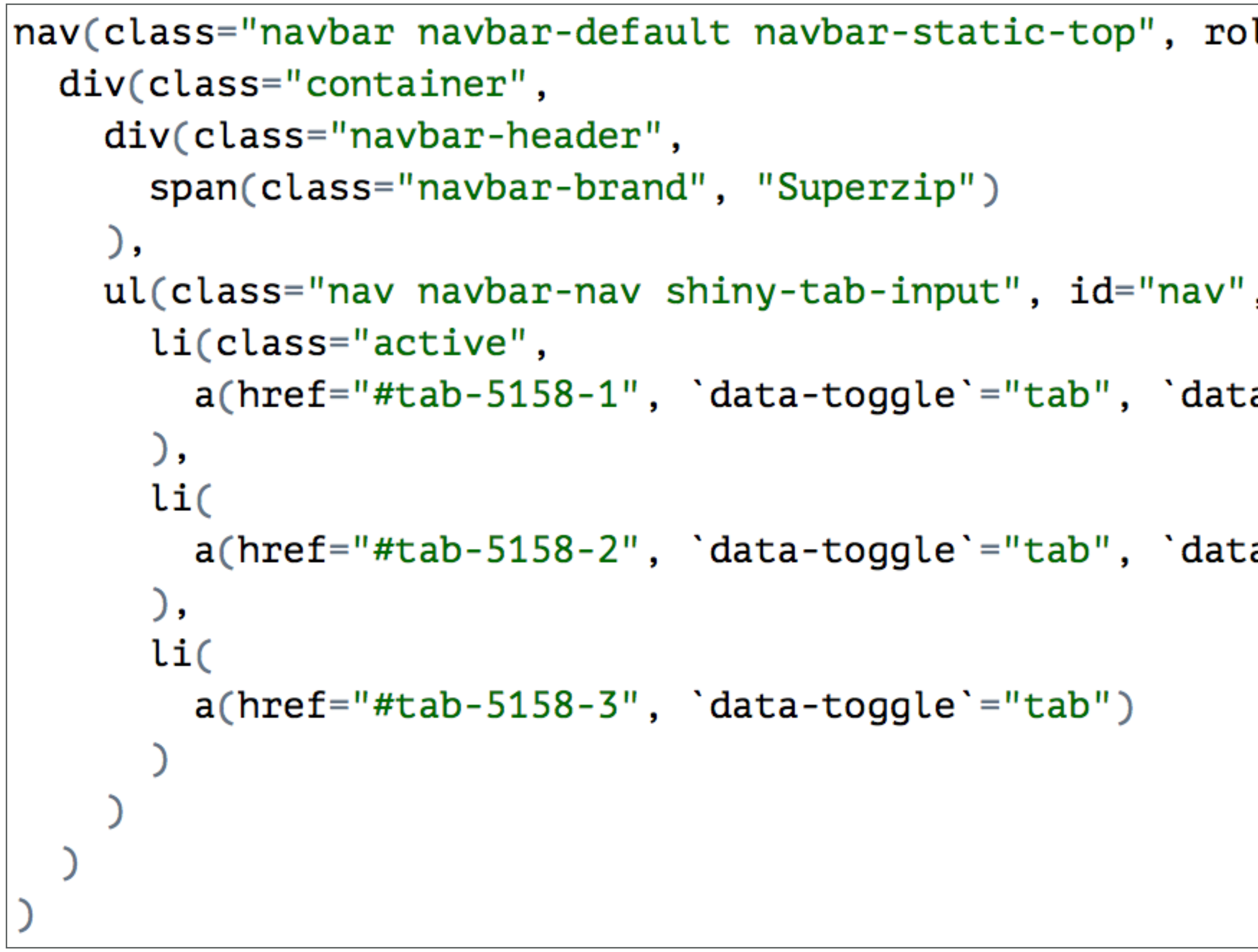
```
<nav class="navbar navbar-default navbar-static-top" ro
  <div class="container">
    <div class="navbar-header">
      <span class="navbar-brand">Superzip</span>
    </div>
    <ul class="nav navbar-nav shiny-tab-input" id="nav"
      <li class="active">
        <a href="#tab-5158-1" data-toggle="tab" data-va
      </li>
      <li>
        <a href="#tab-5158-2" data-toggle="tab" data-va
      </li>
      <li>
        <a href="#tab-5158-3" data-toggle="tab"></a>
      </li>
    </ul>
  </div>
</nav>
<div class="container-fluid">
  <div class="tab-content">
    <div class="tab-pane active" data-value="Interactiv
      <div class="outer">
        <div id="map" style="width:100%; height:100%; "
        <div class="panel panel-default draggable" id="
```

HTMLTOOLS OBJECTS

- HTML-generating R functions

- Pros
  - All the power of HTML, but looks like R
  - Automated CSS/JS dependency handling
  - More composable, programmable than HTML

- Cons
  - Easy to misplace commas
  - Almost as verbose as raw HTML



```
nav(class="navbar navbar-default navbar-static-top", rol
  div(class="container",
    div(class="navbar-header",
      span(class="navbar-brand", "Superzip")
    ),
    ul(class="nav navbar-nav shiny-tab-input", id="nav",
      li(class="active",
        a(href="#tab-5158-1", `data-toggle`="tab", `data
      ),
      li(
        a(href="#tab-5158-2", `data-toggle`="tab", `data
      ),
      li(
        a(href="#tab-5158-3", `data-toggle`="tab")
      )
    )
  )
)
```

HIGH LEVEL FUNCTIONS

- Functions that return htmltools objects
- Pros
  - Less code, clearer intent
  - Anyone can make their own
- Cons
  - Still have to watch out for commas
  - Less flexible

```
navbarPage("Superzip", id = "nav",
  tabPanel("Interactive map", ...),
  tabPanel("Data explorer", ...)
)
```

Using Shiny

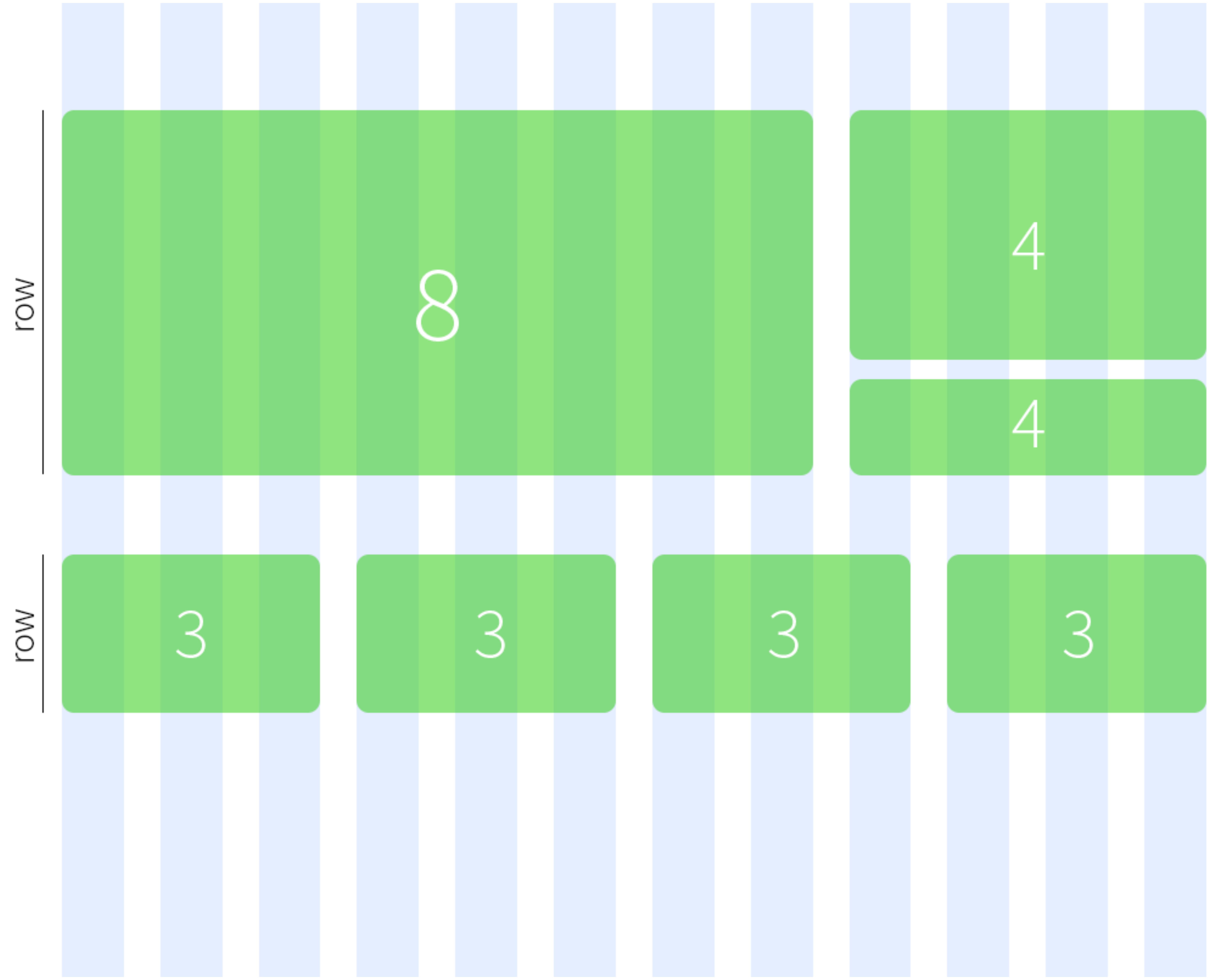
built-ins

SHINY UI BUILT-INS

- **Bootstrap grid framework** – `fluidPage`, `fixedPage`, `fluidRow`, `column`
- **Containers** – `wellPanel`, `absolutePanel`, `fixedPanel`
- **Navigation panels** – `tabsetPanel`, `navlistPanel`, `navbarPage`
- **Fill layouts** (Shiny 0.13+) – `fillPage`, `fillRow`, `fillCol`
- **Modals and notifications** (Shiny 0.14+) – `showModal`, `modalDialog`

BOOTSTRAP GRID FRAMEWORK

- Every page has 12 invisible columns
- Each column of content must span an integral number of columns
- Simple R API for implementing Bootstrap grid
  - `fluidPage(...)` wraps the entire page
  - `fluidRow(...)` wraps each row's column
  - `column(width, ...)` wraps each column's content



Shiny from RStudio

© 2016 RStudio, Inc. All rights reserved.

```r
ui <- fluidPage(
  fluidRow(
    column(8, item1),
    column(4, item2, item3),
  ),
  fluidRow(
    column(3, item4),
    column(3, item5),
    column(3, item6),
    column(3, item7)
  )
)
```

EXERCISE

- Modify `ui_01.R` to display the two outputs next to each other (instead of above and below)
- Assign the left output to be 5 columns wide, and the right output to be 7 columns wide
- See what happens as you change the width of the browser window

3m 00s

© 2016 RStudio, Inc. All rights reserved.

SOLUTION

Solution to the previous exercise

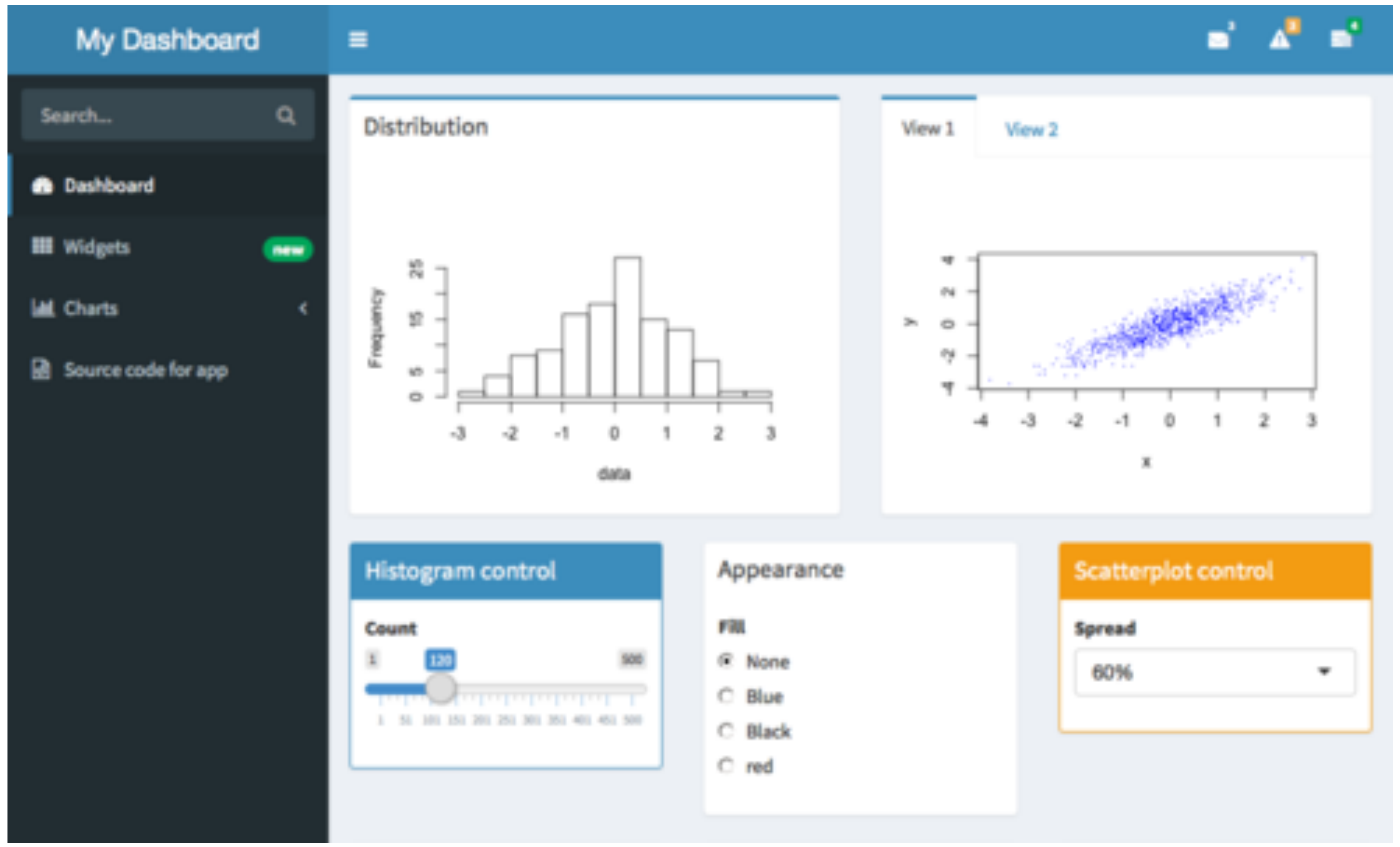
`ui_02.R`

© 2016 RStudio, Inc. All rights reserved.

Using external packages

EXTERNAL PACKAGES

- shinydashboard



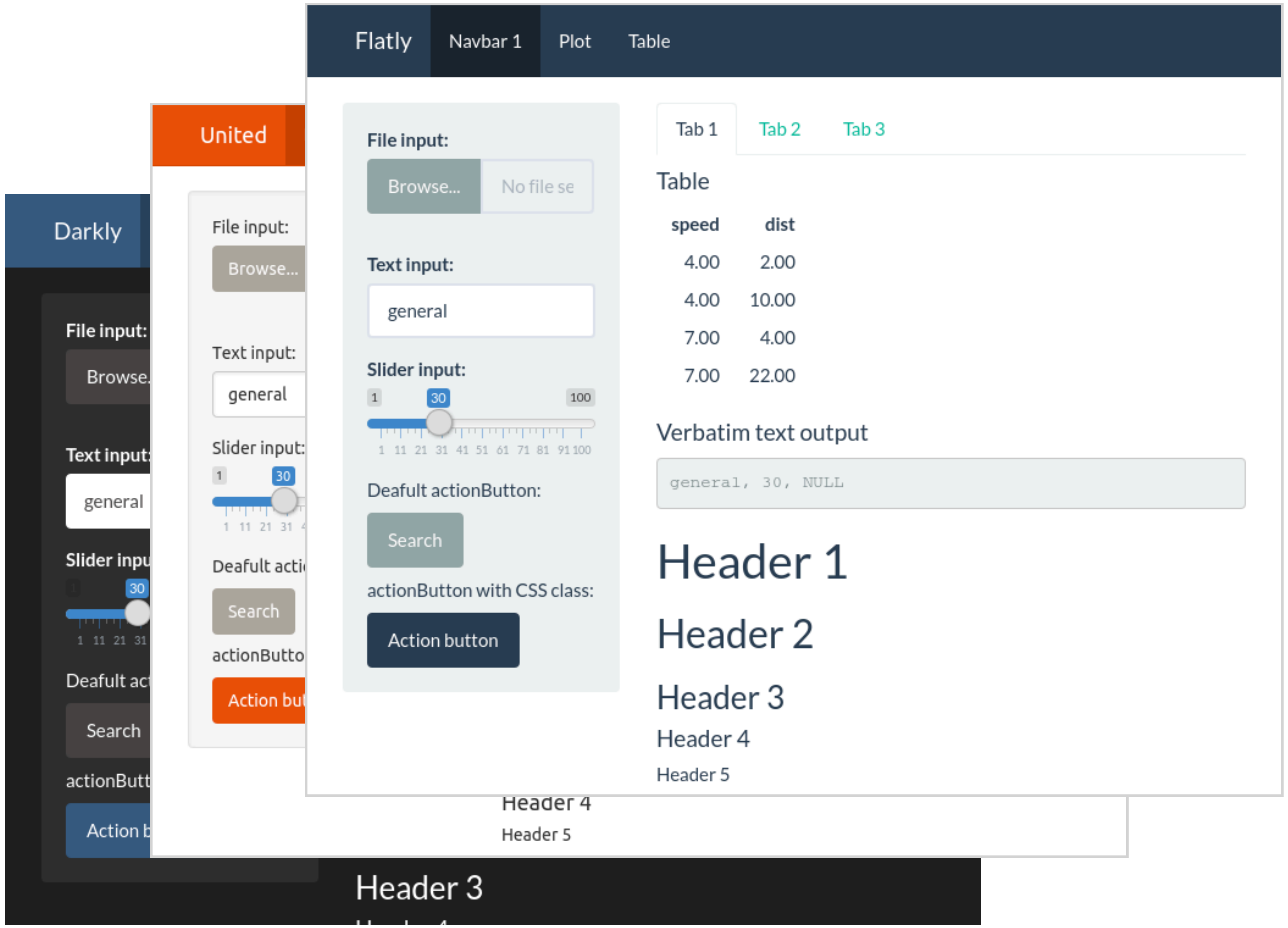
Count: 120 (1–500)

Fill: [✓ None] Blue Black red

Spread: 60%

EXTERNAL PACKAGES

- shinydashboard
- shinythemes

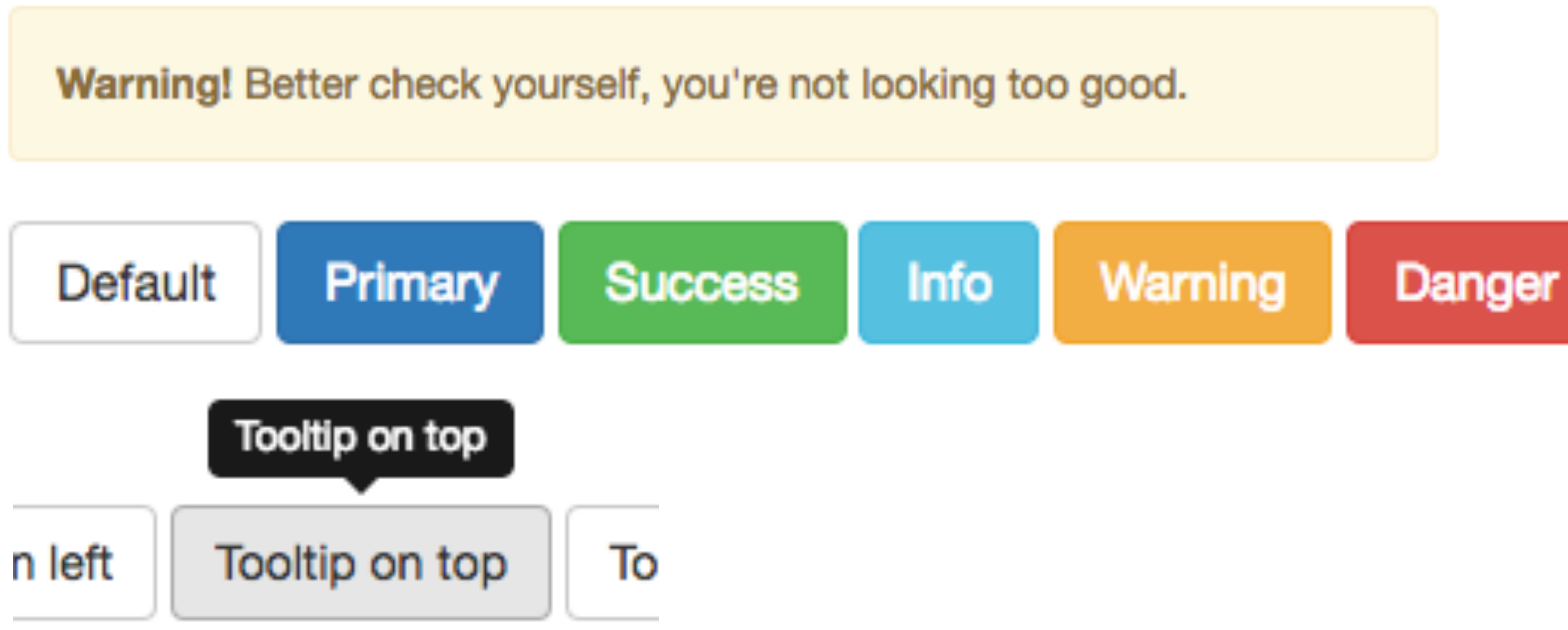


speed	dist
4.00	2.00
4.00	10.00
7.00	4.00
7.00	22.00

```
general, 30, NULL
```

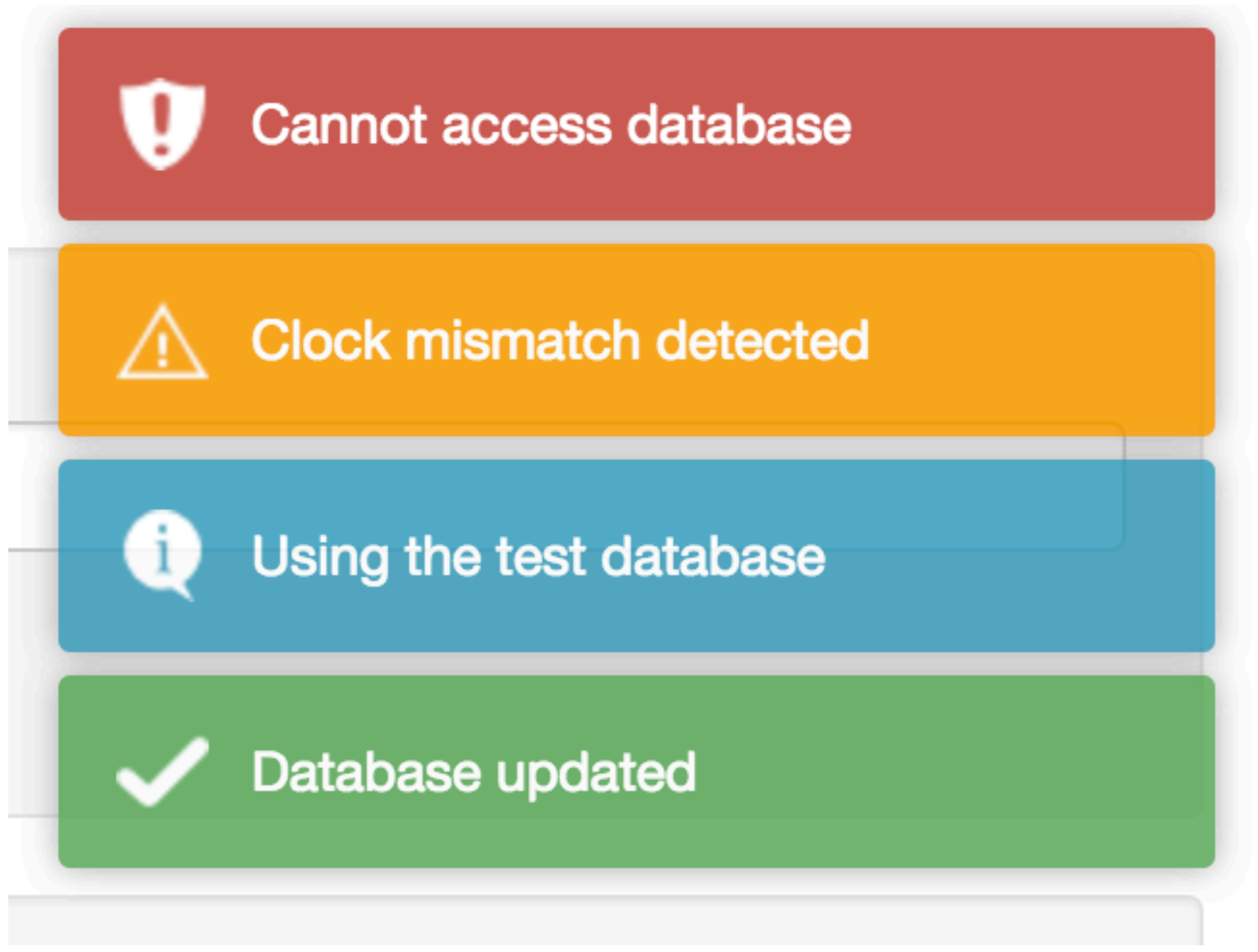
EXTERNAL PACKAGES

- shinydashboard
- shinythemes
- shinyBS (@ebailey78)



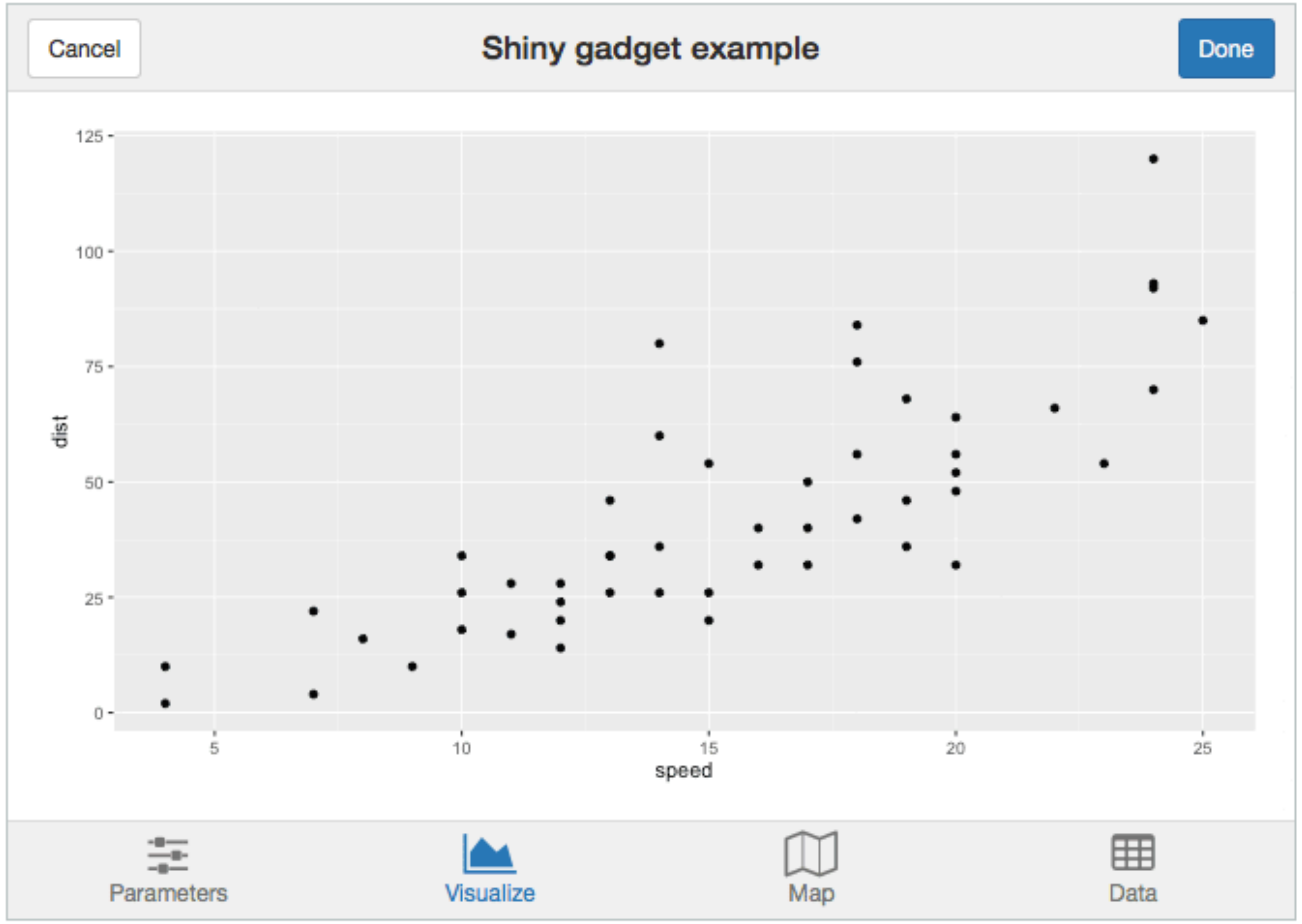
EXTERNAL PACKAGES

- shinydashboard
- shinythemes
- shinyBS (@ebailey78)
- shinytoastr (@gaborcsardi)


Cannot access database
Clock mismatch detected
Using the test database
Database updated

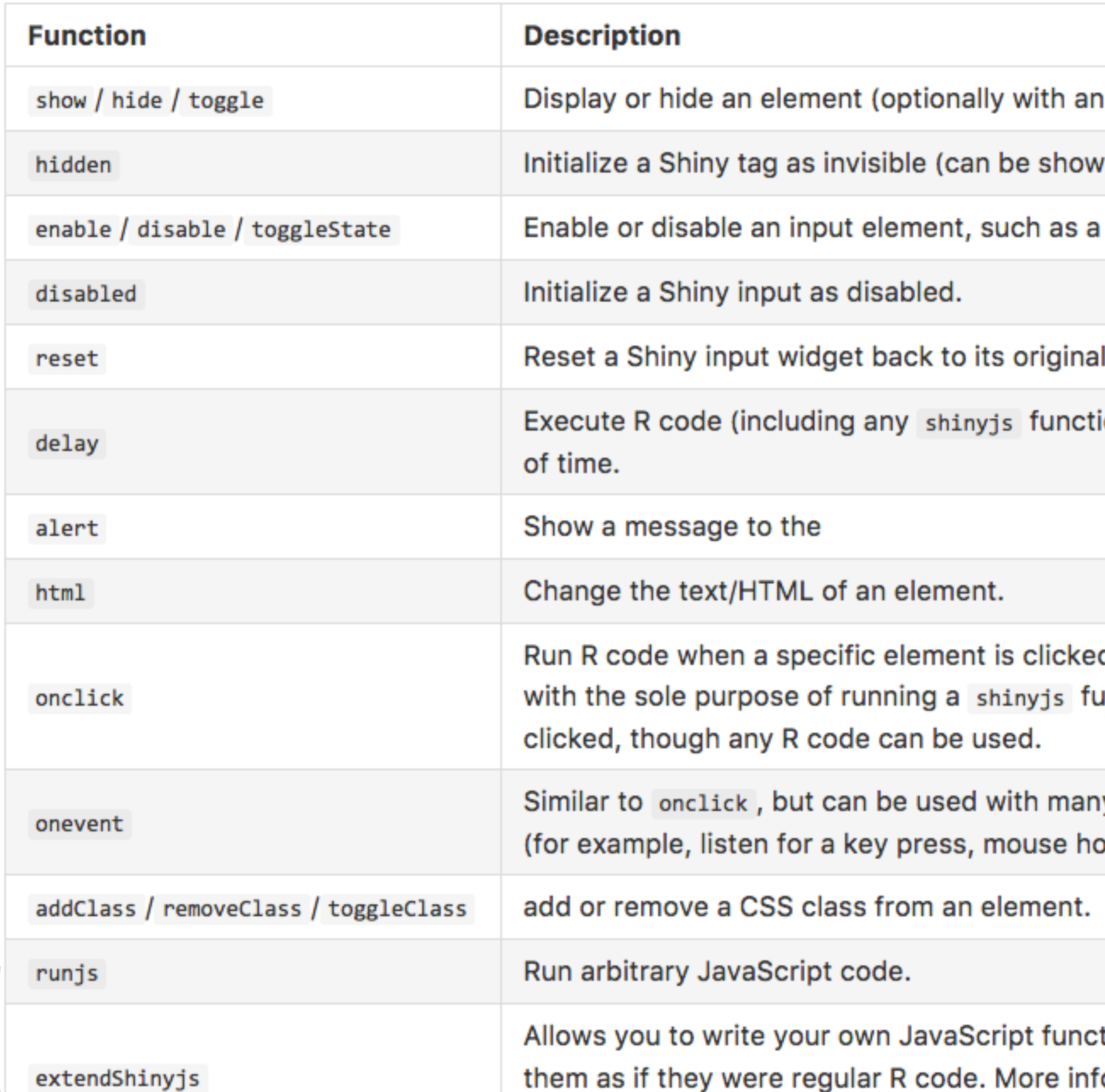
EXTERNAL PACKAGES

- shinydashboard
- shinythemes
- shinyBS (@ebailey78)
- shinytoastr (@gaborcsardi)
- miniUI (for mobile devices or Shiny Gadgets)



EXTERNAL PACKAGES

- shinyjs (@daattali)
  - Perform many UI-related JavaScript operations from R



Function	Description
`show` / `hide` / `toggle`	Display or hide an element (optionally with an
`hidden`	Initialize a Shiny tag as invisible (can be show
`enable` / `disable` / `toggleState`	Enable or disable an input element, such as a
`disabled`	Initialize a Shiny input as disabled.
`reset`	Reset a Shiny input widget back to its original
`delay`	Execute R code (including any `shinyjs` functi of time.
`alert`	Show a message to the
`html`	Change the text/HTML of an element.
`onclick`	Run R code when a specific element is clicked with the sole purpose of running a `shinyjs` fu clicked, though any R code can be used.
`onevent`	Similar to `onclick`, but can be used with many (for example, listen for a key press, mouse ho
`addClass` / `removeClass` / `toggleClass`	add or remove a CSS class from an element.
`runjs`	Run arbitrary JavaScript code.
`extendShinyjs`	Allows you to write your own JavaScript funct them as if they were regular R code. More info

EXERCISE

- Modify `movies_12.R` to use a Bootstrap theme
  - Use the "Live theme selector" feature in shinythemes in your own app
  - Once you've decided on a theme, remove the theme selector and apply your chosen theme permanently
- See shinythemes instructions at: https://rstudio.github.io/shinythemes/

5m 00s

© 2016 RStudio, Inc. All rights reserved.

Solution to the previous exercise

`movies_13.R`

© 2016 RStudio, Inc. All rights reserved.

Using htmltools

tag objects

AN API FOR COMPOSING HTML

- When Shiny was born, it came with a sub-package for composing HTML
- These functions were so useful, we extracted them out into a separate package: htmltools
- Now used by R Markdown and htmlwidgets as well

HTML BASICS

```
<a href="https://www.rstudio.com">RStudio</a>
```

↓

[RStudio](https://www.rstudio.com)

HTML BASICS

```
<a href="https://www.rstudio.com">RStudio</a>
```

End tag

Start tag

Child content

ANATOMY OF A TAG

Attribute name

```
<a href="https://www.rstudio.com">RStudio</a>
```

Tag name

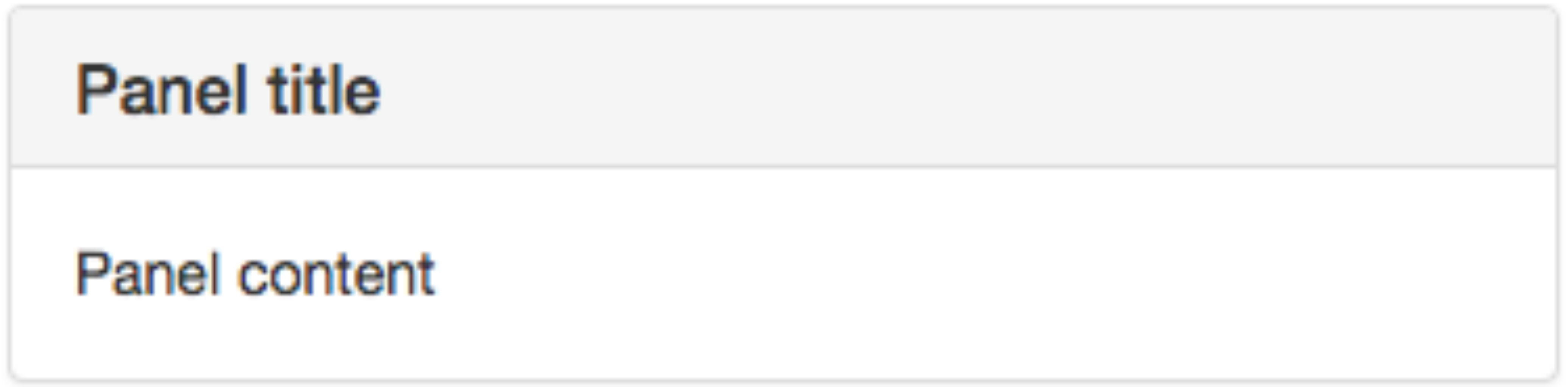
Attribute value

Creates an **a**nchor whose **h**yperlink **ref**erence is the URL https://www.rstudio.com

ANATOMY OF A TAG

- Text can contain tags
- Tags can optionally contain text and/or other tags
- Each start tag can have zero or more attributes

```
<div class="panel panel-default">
  <div class="panel-heading">
    <h3 class="panel-title">Panel title</h3>
  </div>
  <div class="panel-body">
    Panel content
  </div>
</div>
```


Panel title

Panel content

LOOKS LIKE R, MEANS HTML

```
<div class="panel panel-default">
  <div class="panel-heading">
    <h3 class="panel-title">
      Panel title
    </h3>
  </div>
  <div class="panel-body">
    Panel content
  </div>
</div>
```

```
div(class="panel panel-default",
  div(class="panel-heading",
    h3(class="panel-title",
      "Panel title",
    )
  ),
  div(class="panel-body",
    "Panel content"
  )
)
```

USING TAG FUNCTIONS

- Many common tags are exported as functions by htmltools and shiny (p, h1-h6, a, br, div, span, img)

- All other tags can be accessed via the `tags` object. E.g., `<li>Item 1</li>` → `tags$li("Item 1")`

- If you have lots of HTML to write, you can use the `withTags` function—it makes the `tags$` prefix optional.

```
withTags(
  ul(
    li("Item 1"), li("Item 2")
  )
)
```

USING TAG FUNCTIONS

- All tag functions behave the same way
  - Call the function to create a tag object
  - *Named* arguments become attributes
  - *Unnamed* arguments become children

TAG ATTRIBUTES

- Any valid HTML attribute name can be used (use quotes if the name has dashes, e.g. `"data-toggle"="dropdown"`)
- Valid tag attribute values are:
  - `NULL` (omit the attribute)
  - `NA` (the attribute should be included with no value)
  - Single-element character vector (or something to be coerced to character)

```
tags$input(type = "checkbox",
  disabled = if (disabled) NA # else NULL
)
```

TAG CHILDREN

- Valid tag children are:
  - Tag objects
  - Single-element character vectors (treated as text)
  - `NULL` (silently ignored)
  - Raw HTML (see `?htmltools::HTML`)
  - Lists of valid tag children (recursive!)

USING TAGS

- Tags are made using normal R functions that take normal parameters and return normal values! You can do R-like things to them:
  `tags$ul(lapply(1:10, tags$li))`
- Print tag objects at the console to see their HTML source
  - Call `print(x, browse = TRUE)` to see their rendered view instead
  - Use `htmltools::browsable()` to make an object show its rendered view when printed, by default
  - If your top-level object is a list, you'll need to wrap in `tagList(...)` to get the right behavior at the console (or in an R Markdown doc)

EXERCISE

- Open `ui_03.R`.
- Replace `includeHTML("youtube_thumbnail.html")` with the equivalent htmltools tag objects.
  - Hint: Take a look inside `youtube_thumbnail.html`.
- If you get that working, take the next step and define an R function that takes a YouTube URL, a title, and a description, and returns a thumbnail frame like the one you created.

5m 00s

© 2016 RStudio, Inc. All rights reserved.

SOLUTION

Solutions to the previous exercise

`ui_04.R`

`ui_05.R`

© 2016 RStudio, Inc. All rights reserved.

Using raw HTML

USING RAW HTML

- Incorporate tiny amounts of HTML using inline string literals wrapped in `HTML()`
  - `div(HTML("This is <strong>HTML</strong>"))`
- For chunks of (static) HTML, use **`includeHTML`** (or similar **`includeCSS`**, **`includeScript`**)
  - `div(includeHTML("file.html"))`
- Or go the other way, with the HTML Templates feature: start with HTML, and embed R expressions that yield tag objects

```r
}

data_props <- combine_data_props(x$marks)
data_ids <- names(data_props)
data_table <- x$data[data_ids]

# Collapse each list of scale objects into one scale object.
x <- collapse_scales(x)
scale_data_table <- scale_domain_data(x)

# Wrap each of the reactive data objects in another reactive which returns
# only the columns that are actually used, and adds any calculated columns
# that are used in the props.
data_table <- active_props(data_table, data_props)

# From an environment containing data_table objects, get static data for the
# specified ids.
```

as.vega.ggvis()

Values

```
data_ids      "diamonds0/bin1/stack2"
data_props    List of 1
  dynamic     FALSE
```

Traceback

```
as.vega.ggvis(x, FALSE) at vega.R:29
as.vega(x, FALSE) at vega.R:11
view_static(x, ...) at print.R:67
print.ggvis(c("list()", "list(diamonds0 = function () \nstatic_data)", "li
```

Console ~/r/ggvis/

```
> ggvis(diamonds, x = ~price, y = ~color)
Guessing layer_histograms()
Guessing binwidth = 1
Called from: eval(expr, envir, enclos)
Browse[1]> n
debug at /Users/jmcphers/r/ggvis/R/vega.R#29: data_table <- x$data[data_ids]
Browse[2]>
```

UNDERSTANDING UI

Shiny from RStudio