HDAT9800 Visualisation and Communication of Health Data

Chapter 6

Dr James Farrow & Tim Churches

Shiny

Package for building interactive web applications

- standalone apps
- embedded apps in Rmarkdown, *knitr* and *learnr* documents
- dashboards

Extensible with CSS, HTML widgets, and JavaScript

```
install.packages(“shiny")
```

Using shiny

Using *shiny*

- good online *shiny* tutorials: video and written
- *https://shiny.rstudio.com/tutorial/*

Can involve significantly more programming than a ‘simple’ analysis

Large projects benefit from rigorous development methodology

- thoughtful informed specifications, good software design, issue tracking, source control, agile processes, test driven development, proper QA, full release-cycle planning

Shiny architecture

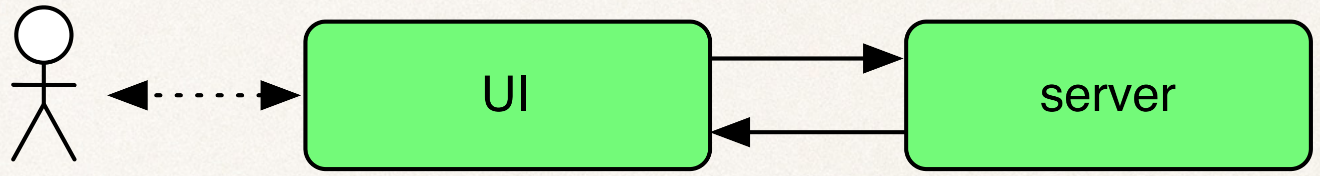
Front end

- text, buttons, controls, graphics, JavaScript, HTML, …

Back end

- R programme responding to inputs and generating outputs

Shiny architecture



User interacts with the front end UI in a browser

Elements of the frontend UI communicate with the backend server

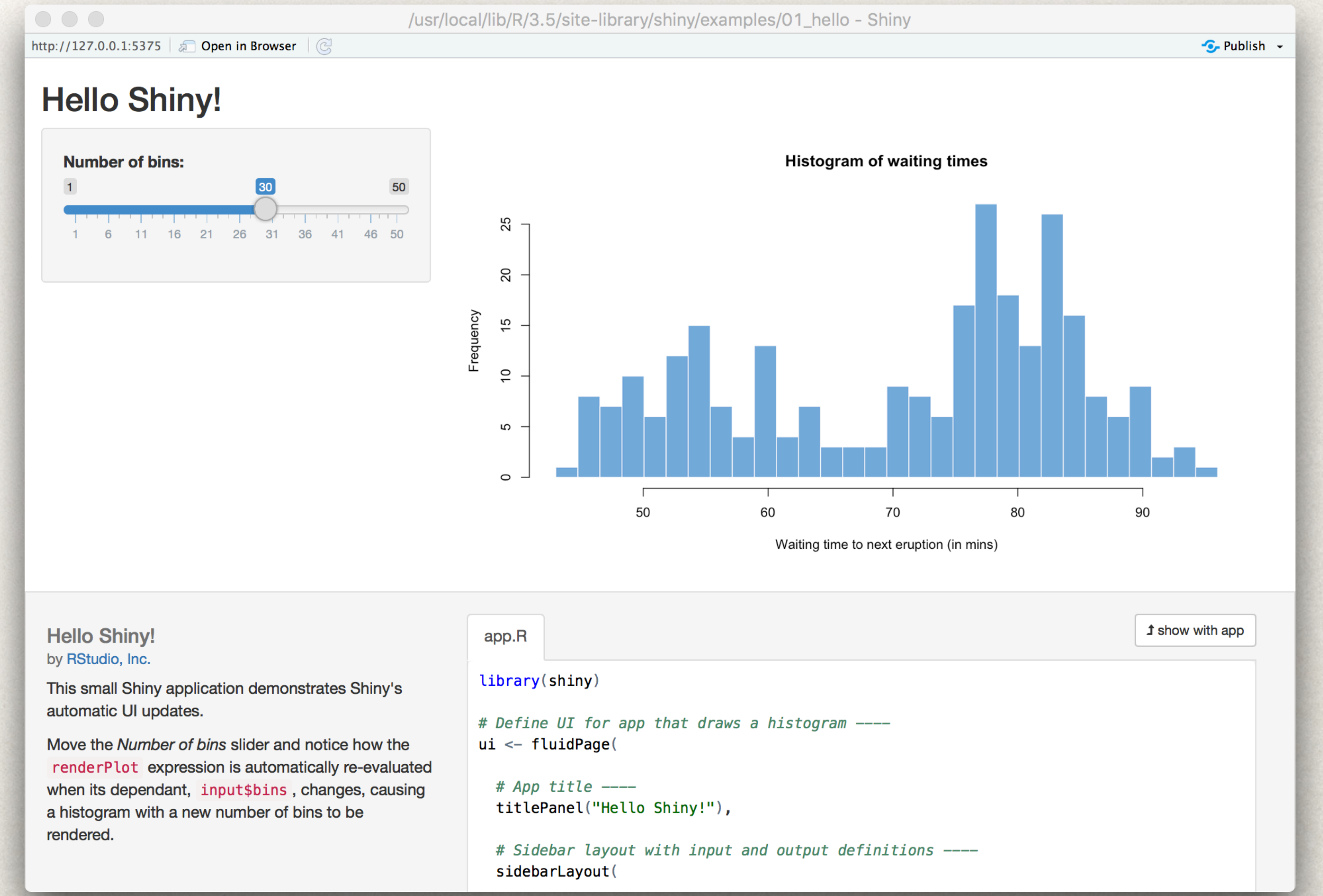
The backend server returns results to the frontend UI

Shiny examples

There are many examples in the *shiny* package

`runExample("01_hello")`

- 01_hello
- 02_text
- 03_reactivity
- 04_mpg
- 05_sliders
- 06_tabsets
- 07_widgets
- 08_html
- 09_upload
- 10_download
- 11_timer



Hello Shiny!
by RStudio, Inc.

This small Shiny application demonstrates Shiny's automatic UI updates.

Move the *Number of bins* slider and notice how the `renderPlot` expression is automatically re-evaluated when its dependant, `input$bins`, changes, causing a histogram with a new number of bins to be rendered.

```r
library(shiny)

# Define UI for app that draws a histogram ----
ui <- fluidPage(

  # App title ----
  titlePanel("Hello Shiny!"),

  # Sidebar layout with input and output definitions ----
  sidebarLayout(
```

Shiny architecture

Formerly *shiny* wanted the UI in one file and the server code in another

The two separate elements went in two different files

- ui.R
- server.R

But these days, the UI and the server objects can be in the same file

- app.R

Whichever approach you use, these filenames must be used *exactly*

Two file app

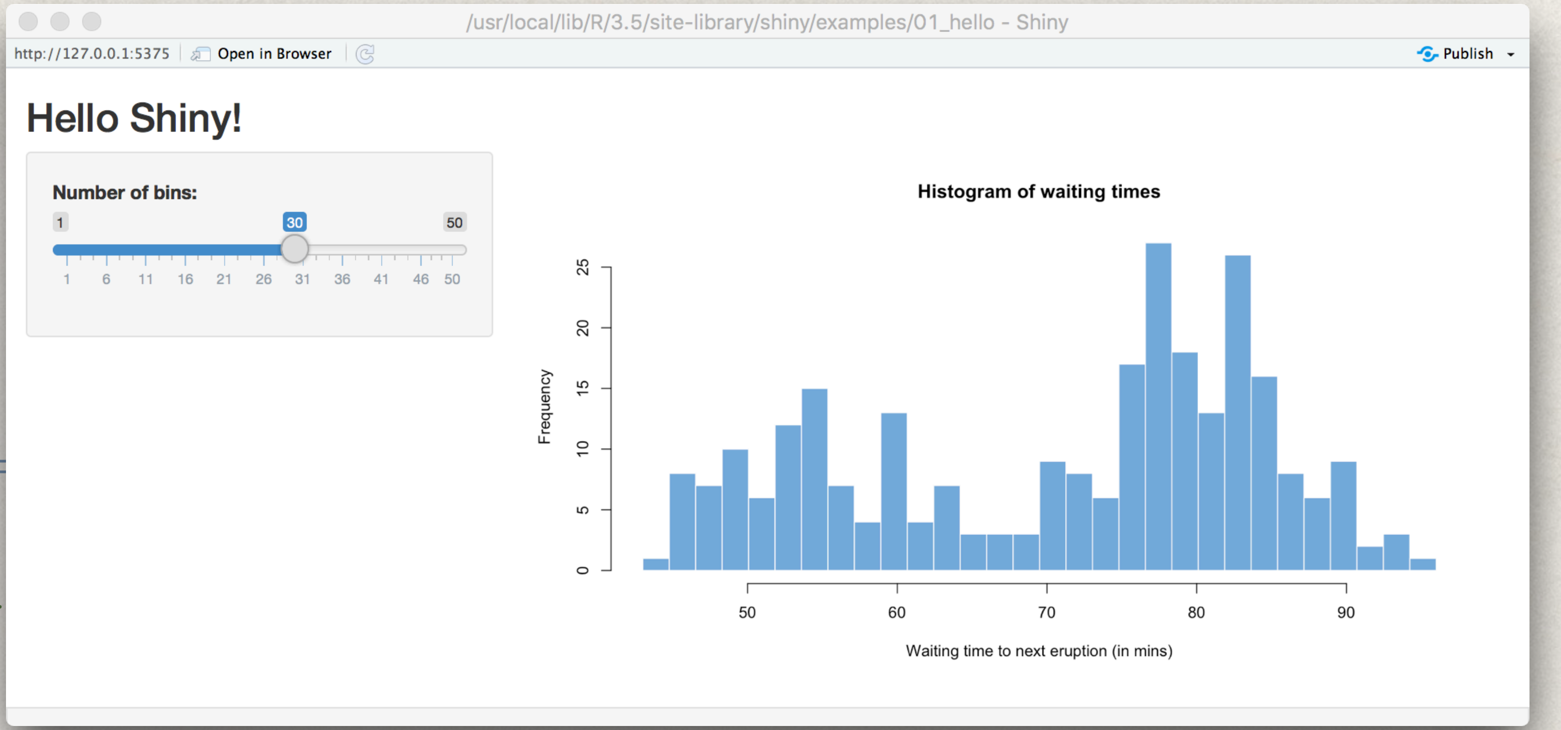
```
# ui.R
library(shiny)

fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25,
    min = 1, max = 100),
  plotOutput("hist")
)
```

```
# server.R
library(shiny)

function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
}
```

Single file app

```
library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25,
    min = 1, max = 100),
  plotOutput("hist")
)

server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
}

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

01_hello: ui



```r
# Define UI for app that draws a h
ui <- fluidPage(
  # App title ----
  titlePanel("Hello Shiny!"),

  # Sidebar layout with input and output definitions ----
  sidebarLayout(

    # Sidebar panel for inputs ----
    sidebarPanel(
      # Input: Slider for the number of bins ----
      sliderInput(inputId = "bins",
                  label = "Number of bins:",
                  min = 1, max = 50, value = 30)
    ),

    # Main panel for displaying outputs ----
    mainPanel(
      # Output: Histogram ----
      plotOutput(outputId = "distPlot")
    )
  )
)
```

01_hello: server

```r
# Define server logic required to draw a histogram ----
server <- function(input, output) {

  # Histogram of the Old Faithful Geyser Data ----
  # with requested number of bins
  # This expression that generates a histogram is wrapped in a call
  # to renderPlot to indicate that:
  #
  # 1. It is "reactive" and therefore should be automatically
  #    re-executed when inputs (input$bins) change
  # 2. Its output type is a plot

  output$distPlot <- renderPlot({

    x    <- faithful$waiting
    bins <- seq(min(x), max(x), length.out = input$bins + 1)

    hist(x, breaks = bins, col = "#75AADB", border = "white",
         xlab = "Waiting time to next eruption (in mins)",
         main = "Histogram of waiting times")
    })
}
```

Running *shiny* apps

Run *shiny* apps by using *runApp()* on the directory containing the app

```
runApp(“myApp”)
```

Much as with *knitr* and *learnr* documents, RStudio recognises *shiny* apps

Press the ▶ *Run App* button at the top of the window to run the app

Close the window or press the stop button 🛑 to stop the app

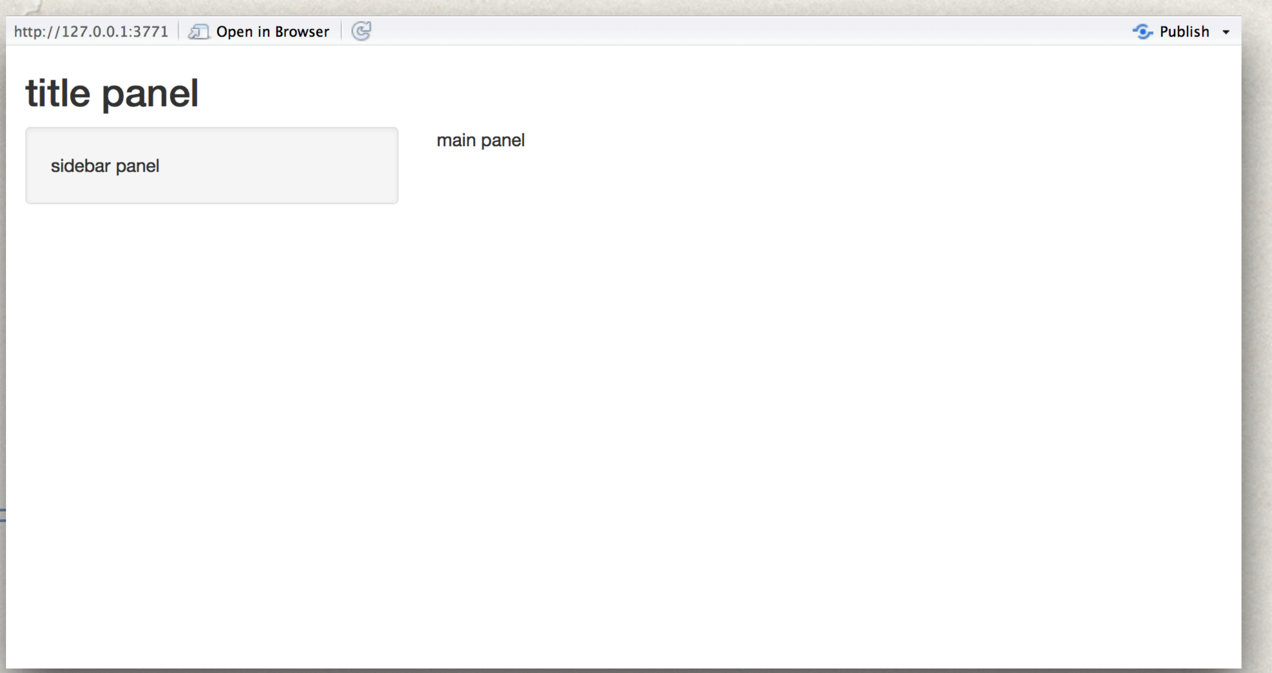
The UI

The shiny UI is basically an HTML page constructed using convenience functions to generate the various HTML page elements

```
ui <- fluidPage(
  # …
)
```

This is a shorthand for creating an HTML `<div>` element with the appropriate class

A basic UI


http://127.0.0.1:3771 Open in Browser Publish

title panel

sidebar panel

main panel

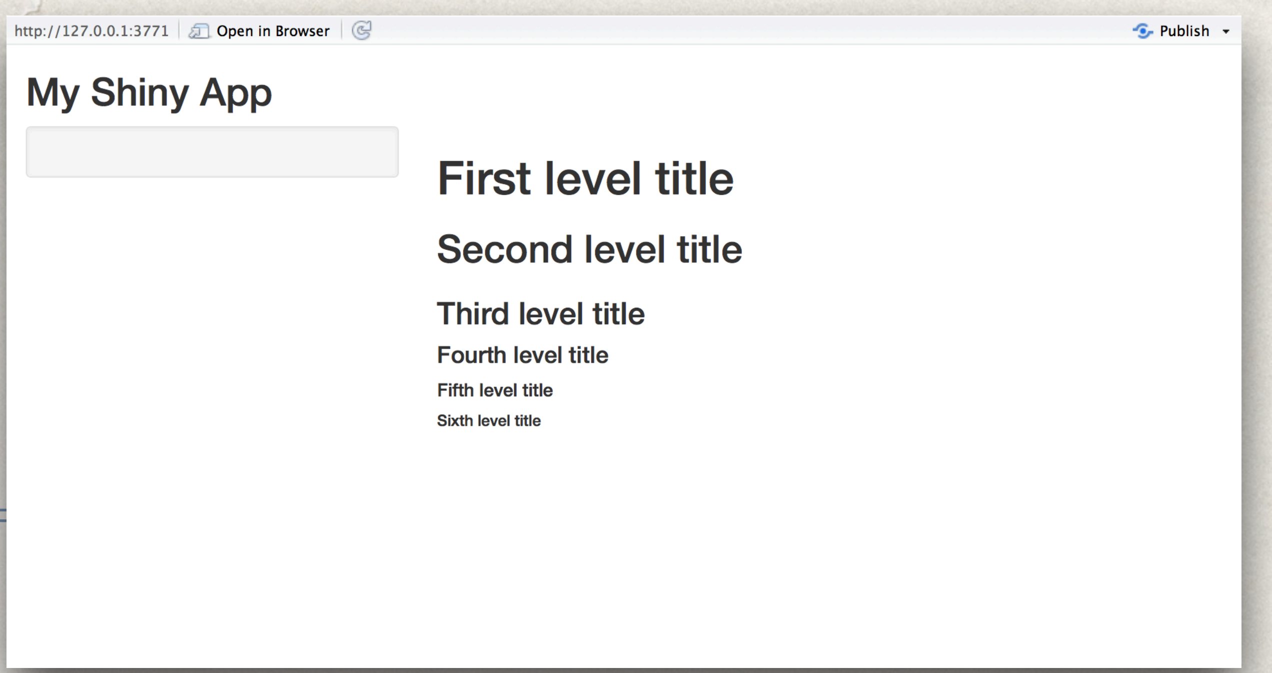
```
ui <- fluidPage(
  titlePanel("title panel"),

  sidebarLayout(
    sidebarPanel("sidebar panel"),
    mainPanel("main panel")
  )
)
```

HTML content

```
p()

h1(), h2(), ..., h6()

a()

br(), div(), span()

pre(), code()

img()

strong(), em()

HTML()
```

HTML content

```
ui <- fluidPage(
  titlePanel("My Shiny App"),
  sidebarLayout(
    sidebarPanel(),
    mainPanel(
      h1("First level title"),
      h2("Second level title"),
      h3("Third level title"),
      h4("Fourth level title"),
      h5("Fifth level title"),
      h6("Sixth level title")
    )
  )
)
```



My Shiny App

First level title

Second level title

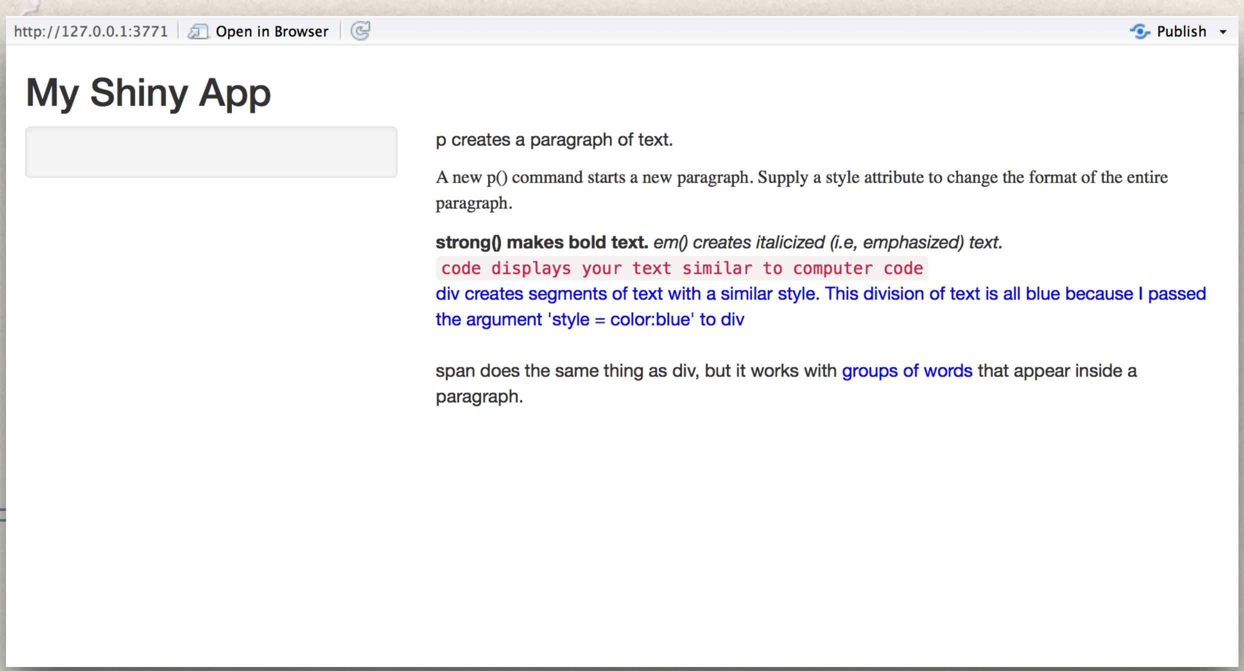
Third level title

Fourth level title

Fifth level title

Sixth level title

HTML content



My Shiny App

p creates a paragraph of text.

A new p() command starts a new paragraph. Supply a style attribute to change the format of the entire paragraph.

strong() makes bold text. *em() creates italicized (i.e, emphasized) text.*
`code displays your text similar to computer code`
div creates segments of text with a similar style. This division of text is all blue because I passed the argument 'style = color:blue' to div

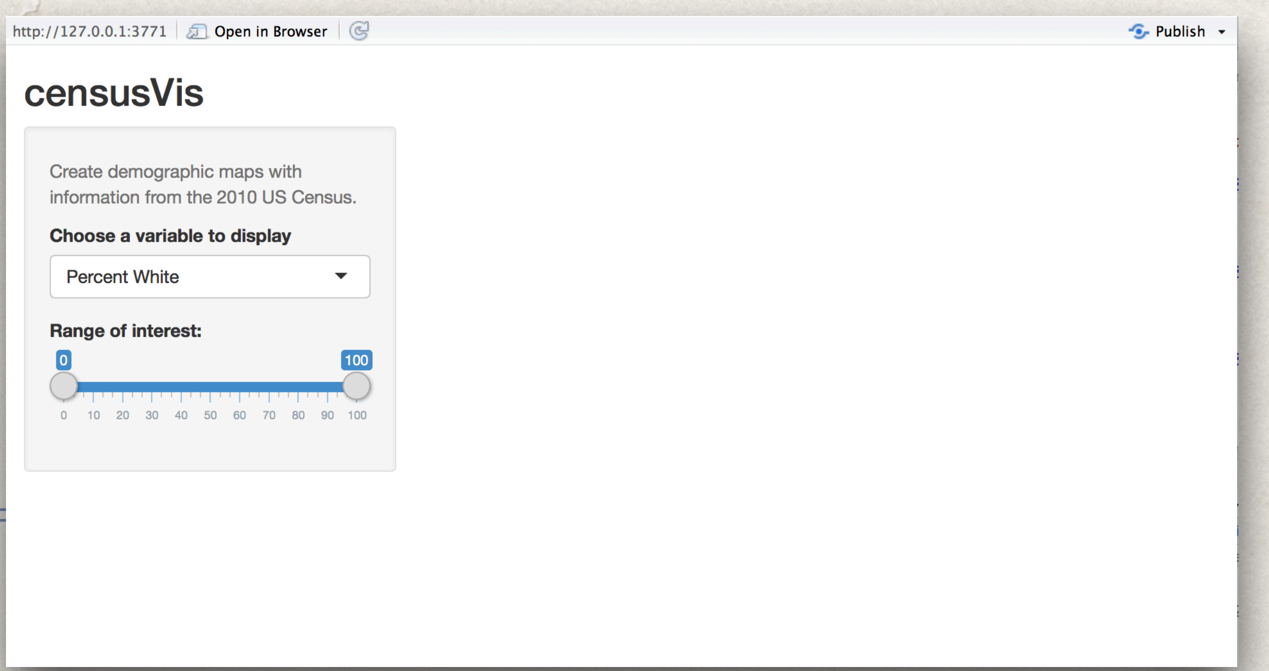
span does the same thing as div, but it works with groups of words that appear inside a paragraph.

```
ui <- fluidPage(
  titlePanel("My Shiny App"),
  sidebarLayout(
    sidebarPanel(),
    mainPanel(
      p("p creates a paragraph of text."),
      p("A new p() command starts a new paragraph. Supply a style attribute to 
change the format of the entire paragraph.", style = "font-family: 'times'; font-
si16pt"),
      strong("strong() makes bold text."),
      em("em() creates italicized (i.e, emphasized) text."),
      br(),
      code("code displays your text similar to computer code"),
      div("div creates segments of text with a similar style. This division of text
is all blue because I passed the argument 'style = color:blue' to div", style = 
"color:blue"),
      br(),
      p("span does the same thing as div, but it works with",
        span("groups of words", style = "color:blue"),
        "that appear inside a paragraph.")
    )
  )
)
```

Control widgets

- actionButton
- checkboxGroupInput
- checkboxInput
- dateInput
- dateRangeInput
- fileInput
- helpText
- numericInput
- radioButtons
- selectInput
- sliderInput
- submitButton
- textInput

Control widgets


http://127.0.0.1:3771 Open in Browser Publish

censusVis

Create demographic maps with information from the 2010 US Census.

Choose a variable to display

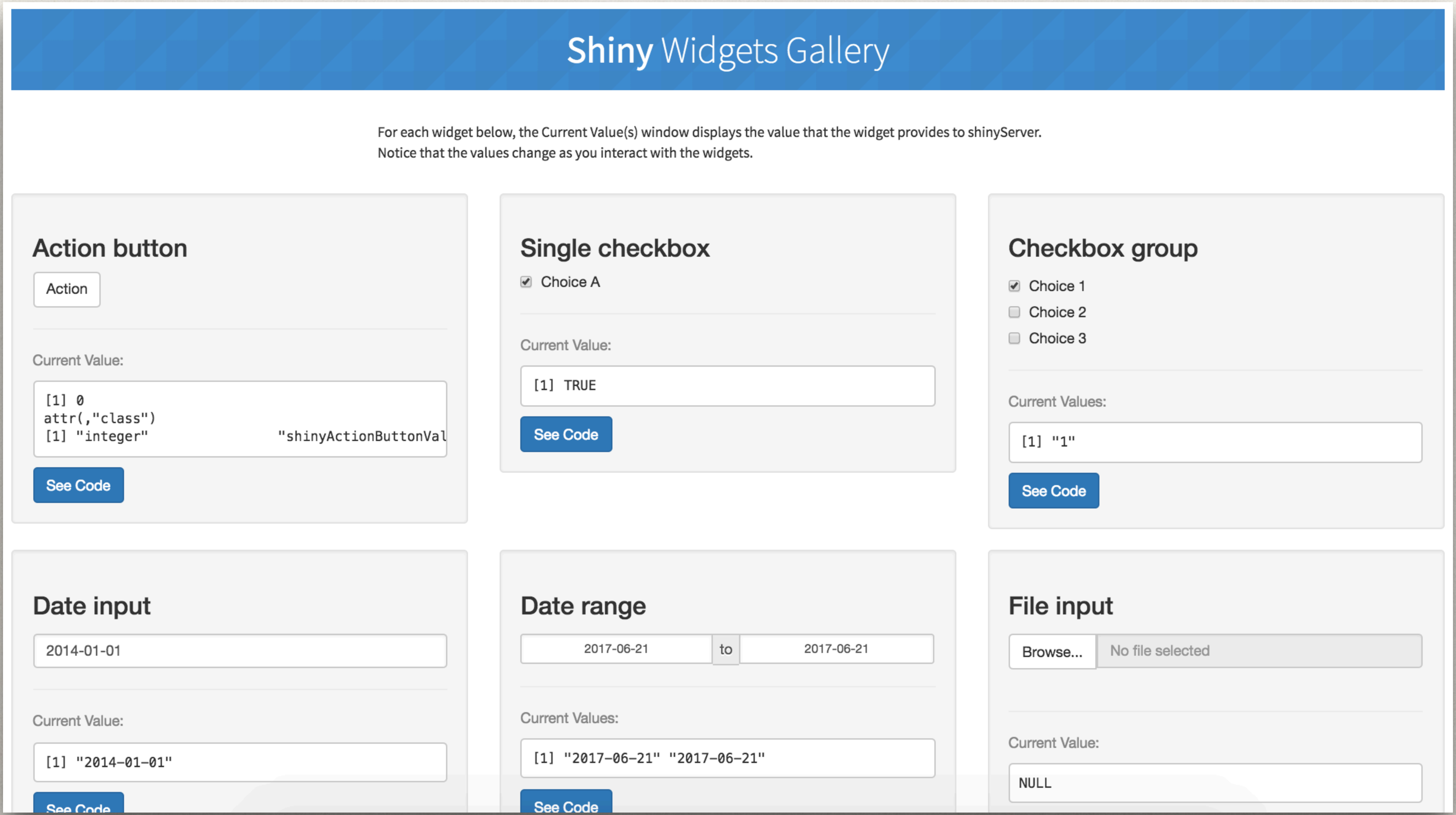
Percent White

Range of interest:

0 100

```
ui <- fluidPage(
  titlePanel("censusVis"),

  sidebarLayout(
    sidebarPanel(
      helpText("Create demographic maps with
               information from the 2010 US Census."),

      selectInput("var",
                  label = "Choose a variable to display",
                  choices = list("Percent White",
                                 "Percent Black",
                                 "Percent Hispanic",
                                 "Percent Asian"),
                  selected = "Percent White"),

      sliderInput("range",
                  label = "Range of interest:",
                  min = 0, max = 100, value = c(0, 100))
    ),

    mainPanel()
  )
)
```

Widget gallery



Shiny Widgets Gallery

For each widget below, the Current Value(s) window displays the value that the widget provides to shinyServer.
Notice that the values change as you interact with the widgets.

Action button

Current Value:

```
[1] 0
attr(,"class")
[1] "integer"              "shinyActionButtonVal
```

Single checkbox

☑ Choice A

Current Value:

```
[1] TRUE
```

Checkbox group

☑ Choice 1
☐ Choice 2
☐ Choice 3

Current Values:

```
[1] "1"
```

Date input

2014-01-01

Current Value:

```
[1] "2014-01-01"
```

Date range

2017-06-21 to 2017-06-21

Current Values:

```
[1] "2017-06-21" "2017-06-21"
```

File input

Current Value:

```
NULL
```

Reactive output

Place output objects in the UI

Tell *shiny* how to build the object
in the server function

- dataTableOutput
- htmlOutput
- imageOutput
- plotOutput
- tableOutput
- textOutput
- uiOutput
- verbatimTextOutput

The two parts

```r
ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25,
    min = 1, max = 100),
  plotOutput("hist")
)
```

```r
server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
}
```

Render functions

Output server entries should contain code to render the UI elements

- renderDataTable
- renderImage
- renderPlot
- renderPrint
- renderTable
- renderText
- renderUI

Render functions

Note that render functions take a closure (code surrounded by {})

```
output$selected_var <- renderText({
    paste("You have selected", input$var)
})
```

It's code because it's like an anonymous function (a function without an explicit name)

shiny needs to run this when the input changes to regenerate the output

And much more

We will unpack these aspects of *shiny* over the next few chapters

You will need to steadily work through the online tutorials pointed out at the beginning of this week's workshop

We'll supplement these with targeted materials and suggestions building towards our interactive geospatial analysis scenario

This week's practical

Go over the *shiny* tutorials at *https://shiny.rstudio.com/tutorial/*

There is also a *learnr* tutorial for *shiny*