

Week 14: *Shiny Apps*

☰ EMSE 4575: Exploratory Data Analysis

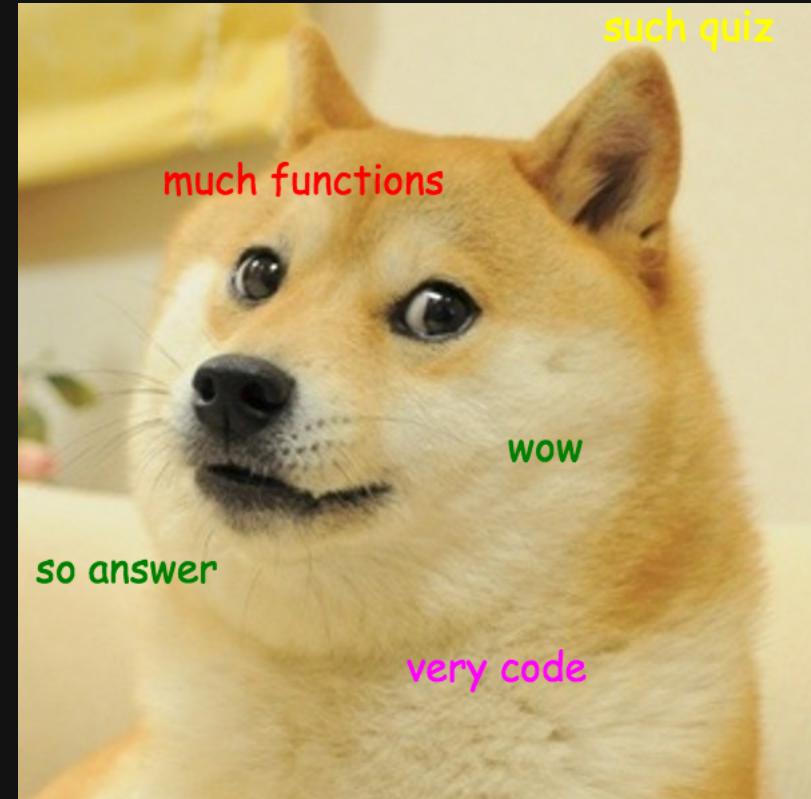
👤 John Paul Helveston

📅 April 14, 2021

Quiz 5

Link is on the [schedule](#)

10 : 00



License

These slides were modified from [Florencia D'Andrea's RLadies Shiny Meetup Slides](#)

This work is licensed under a
[Creative Commons Attribution-ShareAlike 4.0 International License](#)

New libraries to install

```
install.packages('shiny')
install.packages('shinyWidgets')
install.packages('rsconnect')
```



Interactive Webapps in R

Check out the [Shiny Gallery](#)

Week 14: *Shiny Apps*

1. Anatomy of a Shiny App

2. User Interface

3. Server

Intermission

4. Extras

5. Deploying your App

Week 14: *Shiny Apps*

1. Anatomy of a Shiny App

2. User Interface

3. Server

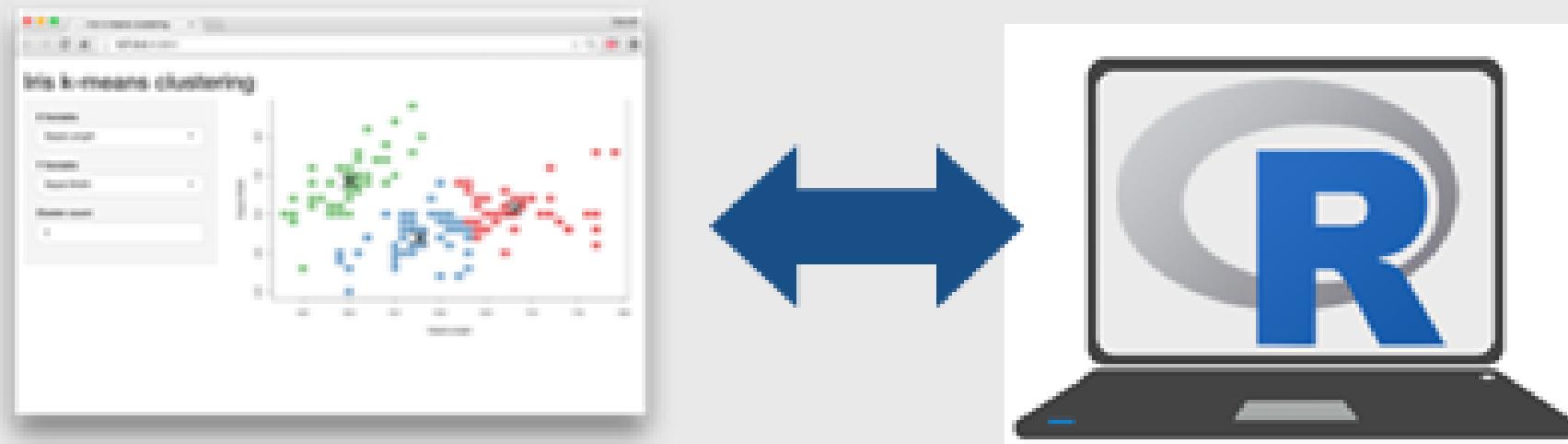
Intermission

4. Extras

5. Deploying your App

Anatomy of a Shiny App

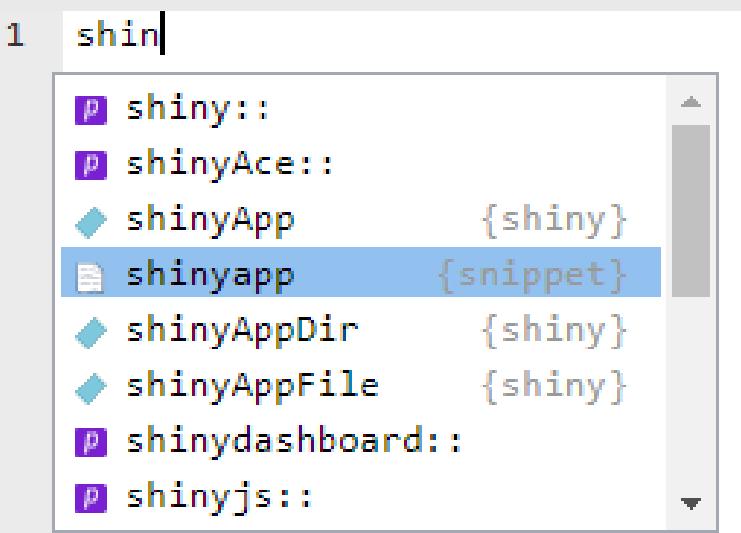
A Shiny app is a web page (UI) connected to a computer running a live R session (Server)



app.R

You can insert all the code at once with the shinyapp snippet!

Just start typing **shiny...**



```
library(shiny)  
  
ui <- fluidPage(  
  
)  
  
server <- function(input, output, session) {  
  
}  
  
shinyApp(ui, server)
```

Building a shiny app



ui

1. Pick a layout function
2. Add inputs widgets
3. Add `*Output()` functions

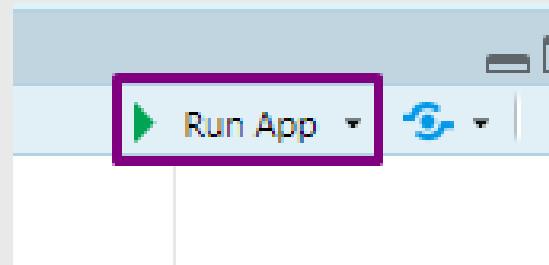
server

1. Use `render*` functions to make outputs
2. Link outputs with `output$<id>`
3. Link inputs with `input$<id>`

Run the app



- **Option 1:** Click the "Run App" button in the toolbar:



- **Option 2:** Use a keyboard shortcut: Cmd/Ctrl + Shift + Enter.
- **Option 3:** `shiny::runApp()` with the path to the **app.R** file.

Your Turn

hello_shiny.app

File → New File → Shiny Web App...

```
library(shiny)

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

  # Application title
  titlePanel("Old Faithful Geyser Data"),

  # Sidebar with a slider input for number of bins
  sidebarLayout(
    sidebarPanel(
      sliderInput("bins",
                  "Number of bins:",
                  min = 1,
                  max = 50,
                  value = 30
    ),
    mainPanel(
      plotOutput("distPlot")
    )
  )
)

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

  output$distPlot <- renderPlot({
    # generate bins based on input$bins from ui.R
    x <- faithful[, 2]
    bins <- seq(min(x), max(x), length.out = input$bins + 1)

    # draw the histogram with the specified number of bins
    hist(x, breaks = bins, col = 'darkgray', border = 'white')
  })
}

# Run the application
shinyApp(ui = ui, server = server)
```

Week 14: *Shiny Apps*

1. Anatomy of a Shiny App

2. User Interface

3. Server

Intermission

4. Extras

5. Deploying your App

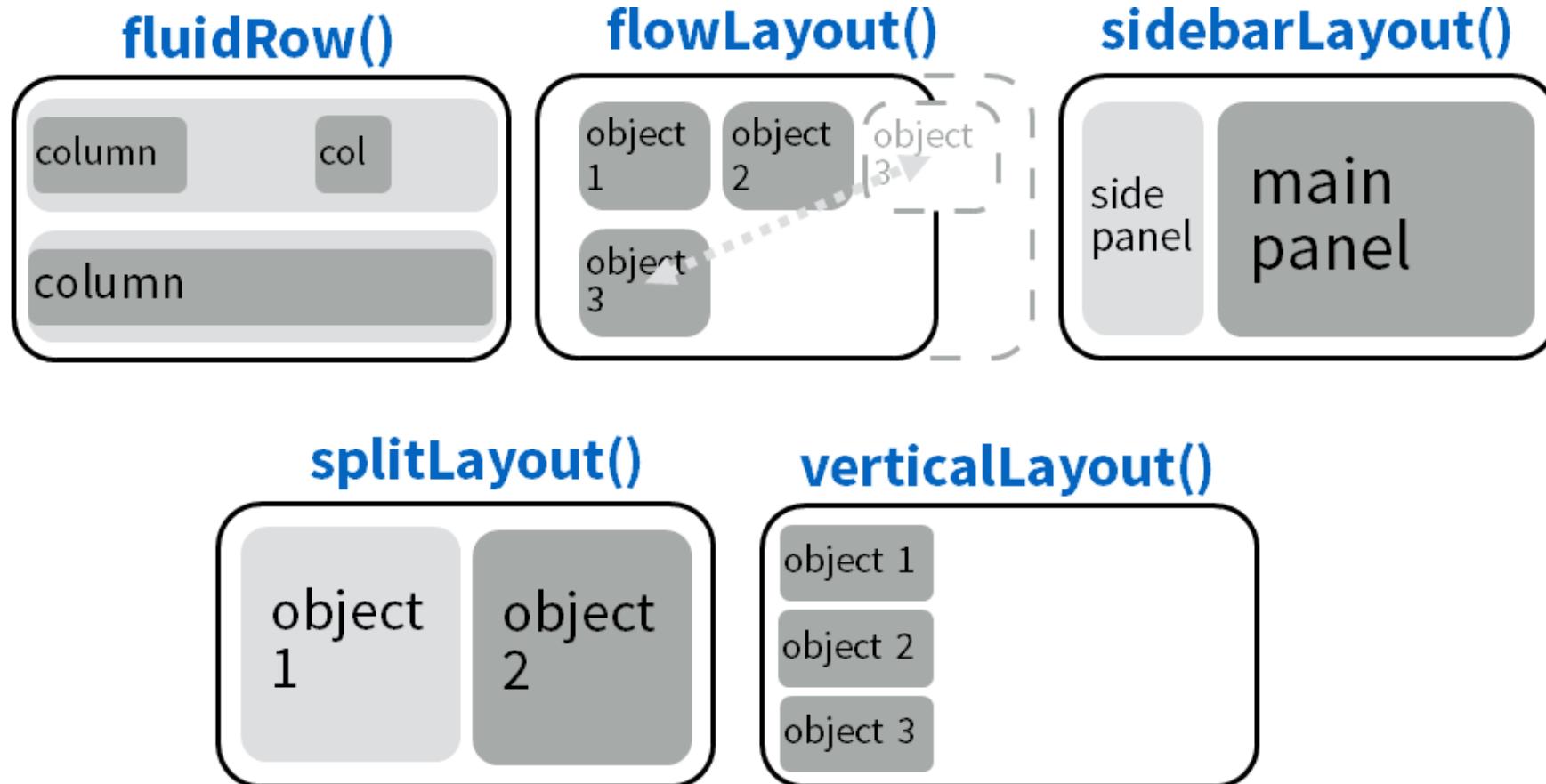
User Interface (UI)

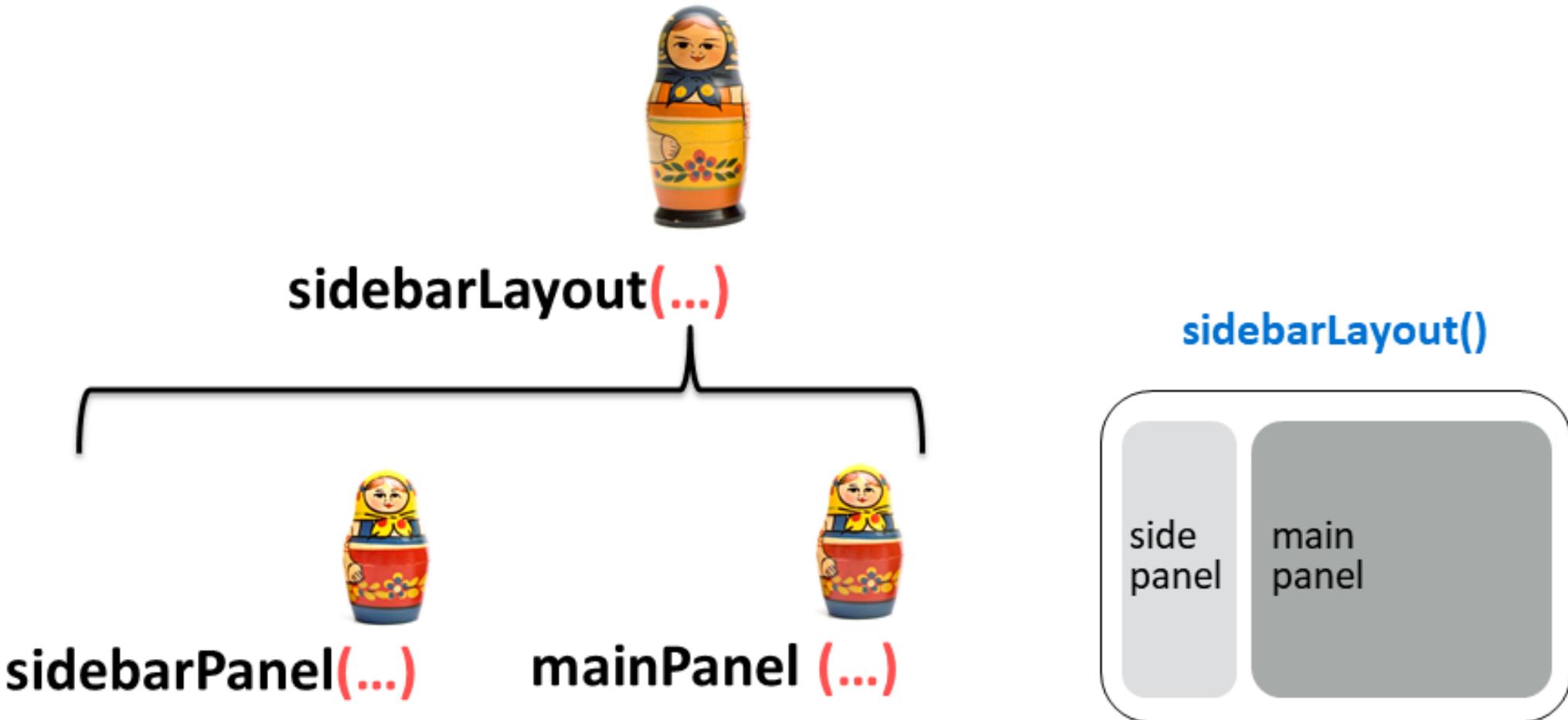


Matryoshka Dolls

Organize panels and elements into a layout with a **layout function**

Top level is usually **fluidPage()**

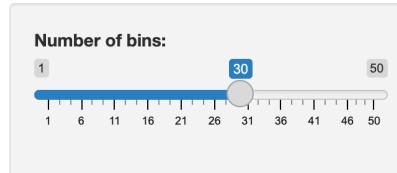




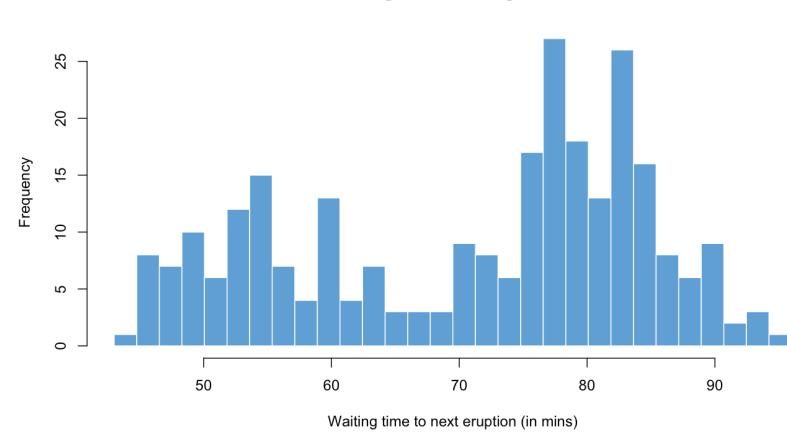
sidebarLayout()

```
ui <- fluidPage(  
  titlePanel("Hello Shiny!"),  
  sidebarLayout(  
    sidebarPanel(  
      sliderInput(  
        "bins", label = "Number of  
        bins", min = 1, value = 30, max =  
        50)  
    ),  
    mainPanel(  
      plotOutput("distPlot")  
    )  
  ))
```

Hello Shiny!



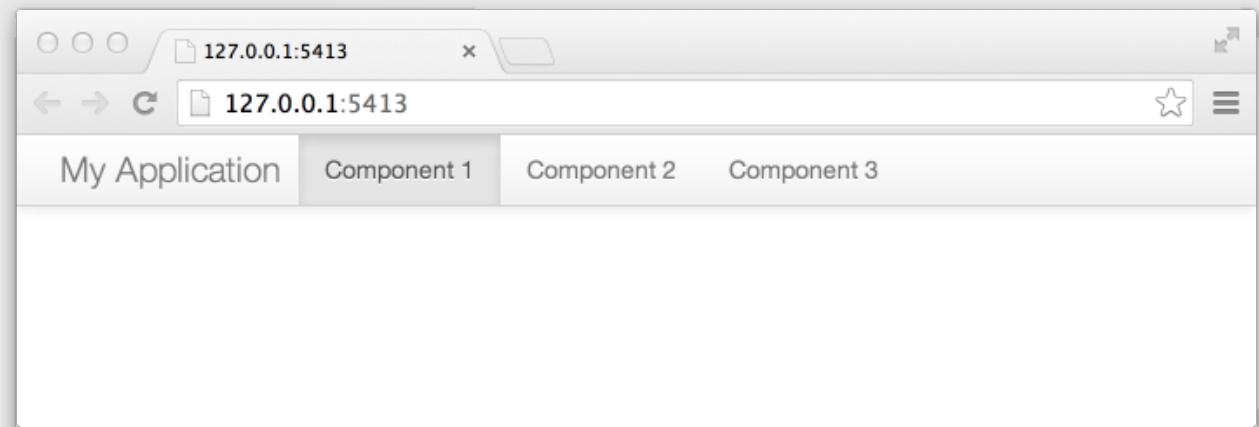
Histogram of waiting times



navbarPage(): An alternative to fluidPage()

Think of each tabPanel() as it's own fluidPage()

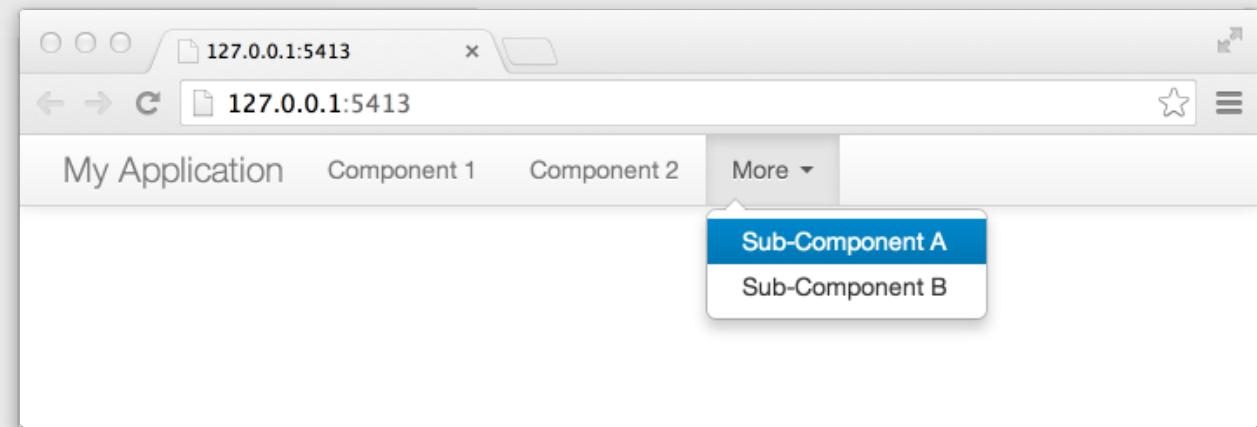
```
ui <- navbarPage("My Application",
  tabPanel("Component 1"),
  tabPanel("Component 2"),
  tabPanel("Component 3")
)
```



navbarPage(): An alternative to fluidPage()

Use navbarMenu() to create a nested menu item

```
ui <- navbarPage("My Application",
  tabPanel("Component 1"),
  tabPanel("Component 2"),
  navbarMenu("More",
    tabPanel("Sub-Component A"),
    tabPanel("Sub-Component B"))
)
```



The UI defines the "what" and "where" for:

1. **Inputs**: collect values from the user
2. **Output**: display something to the user

Inputs: collect values from the user

Buttons

Action

Submit

actionButton()
submitButton()

Date range

2014-01-24 to 2014-01-24

dateRangeInput()

Radio buttons

- Choice 1
- Choice 2
- Choice 3

radioButtons()

Single checkbox

Choice A

checkboxInput()

File input

Choose File No file chosen

fileInput()

Select box

Choice 1

selectInput()

Checkbox group

- Choice 1
- Choice 2
- Choice 3

checkboxGroupInput()

Date input

2014-01-01

Numeric input

1

Password Input

.....

numericInput()

passwordInput()

Sliders



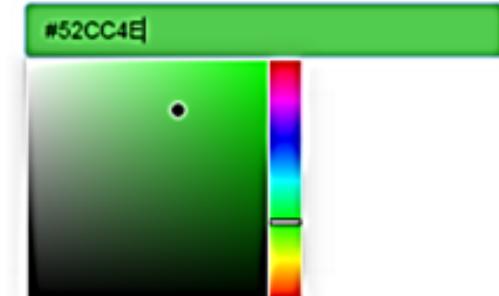
sliderInput()

Text input

Enter text...

textInput()

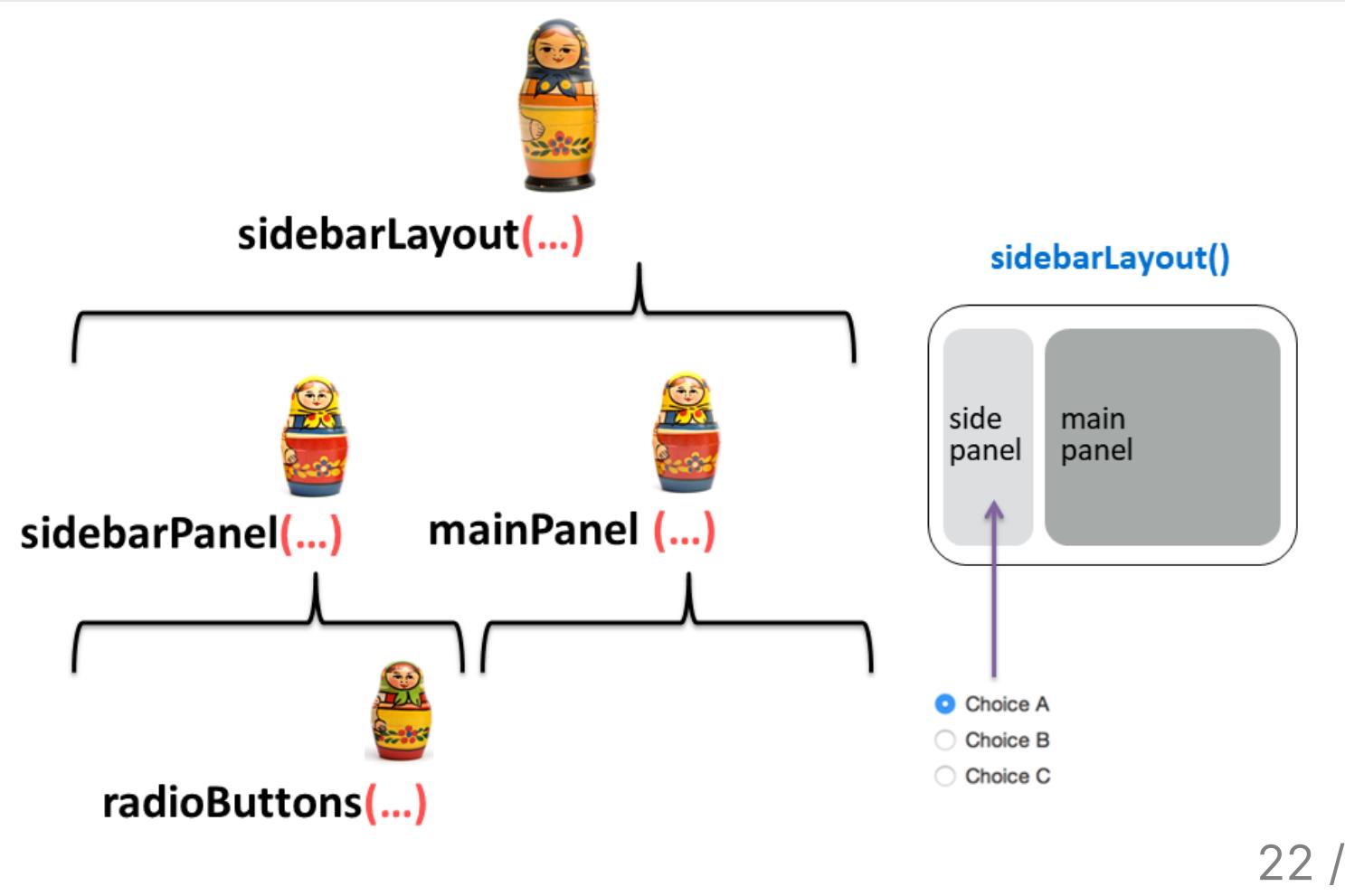
Colour input



shinyjs::colourInput()

Example Input: Radio buttons in the sidebar

```
ui <- fluidPage(  
  sidebarLayout(  
    sidebarPanel(  
      radioButtons(...)  
    ),  
    mainPanel(...)  
  )  
)
```



Quick practice

Open the `apps/widgets.R` file and click the "Run App" button



(((((((((((())))))))))))

```
) #mainPanel  
 ) #sidebarLayout  
 ) #fluidPage
```

Your Turn

10:00

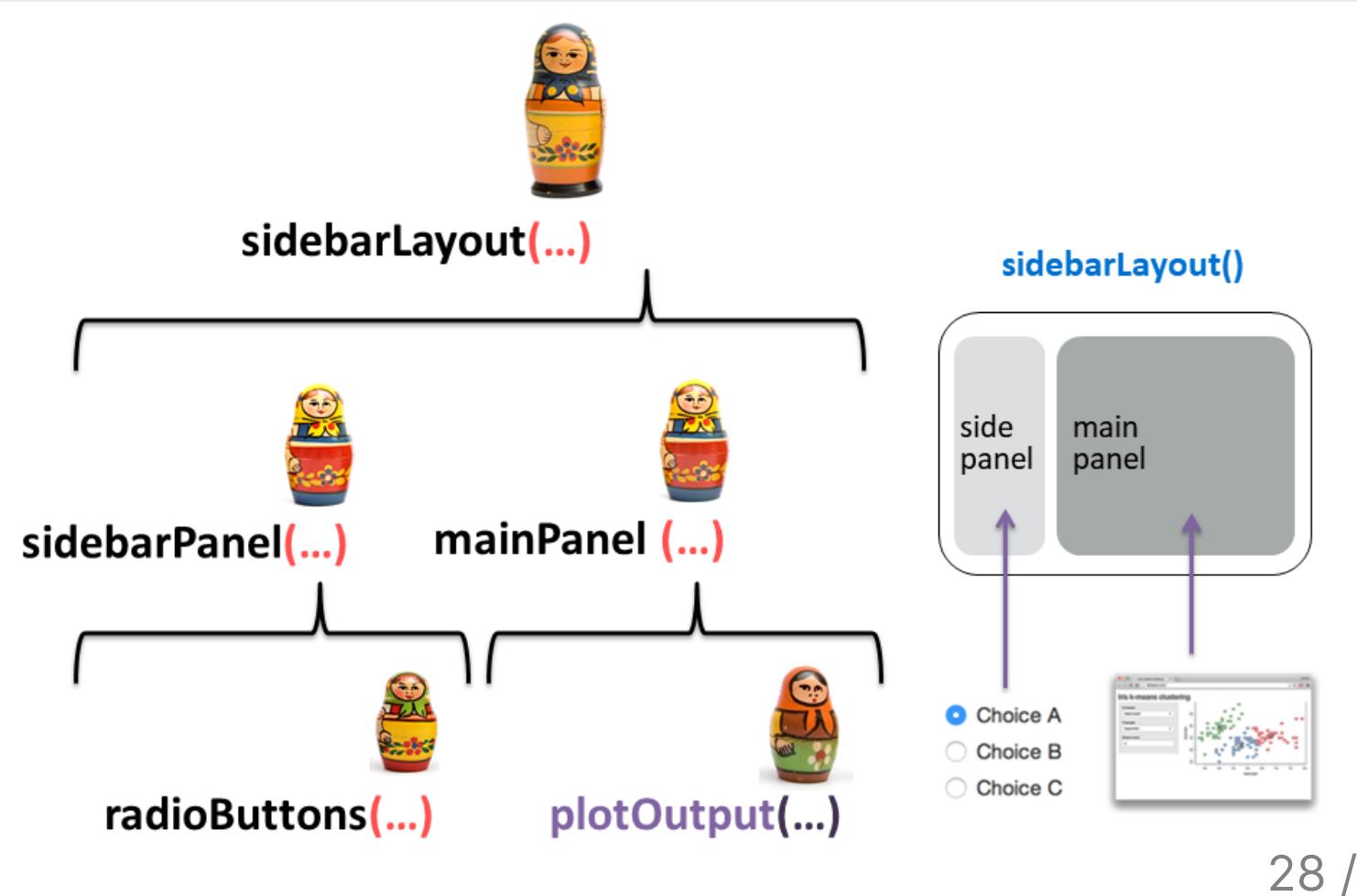
1. Open the `widgets.R` file.
2. Go to the [Shiny Widgets Gallery](#) and pick a new widget.
3. Look at the code for the widget and try to add that widget to the sidebar in the `widgets.R` file below all the other widgets.
4. Debug together! (ask each other questions)

The UI defines the "what" and "where" for:

1. **Inputs**: collect values from the user
2. **Output**: display something to the user

Example Output: Put a plot in the main panel

```
ui <- fluidPage(  
  sidebarLayout(  
    sidebarPanel(  
      radioButtons(...)  
    ),  
    mainPanel(  
      plotOutput(...)  
    )  
  )  
)
```



Output: display something to the user

Output function	Description
<code>plotOutput()</code>	Display a reactive <i>plot</i>
<code>dataTableOutput()</code>	Display a <code>DT::datatable()</code>
<code>textOutput()</code>	Display reactive <i>text</i>
<code>imageOutput()</code>	Display an image

Week 14: *Shiny Apps*

1. Anatomy of a Shiny App

2. User Interface

3. Server

Intermission

4. Extras

5. Deploying your App

Building a shiny app



ui

1. Pick a layout function, e.g.
`sidebarLayout()`
2. Add inputs widgets
3. Add `*Output()` functions

server

1. Use `render*` functions to make outputs
2. Link outputs with `output$<id>`
3. Link inputs with `input$<id>`

Outputs - render*() and *Output() functions work together to add R output to the UI



`DT::renderDataTable(expr,
options, callback, escape,
env, quoted)`

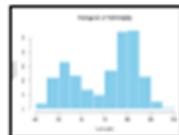


`dataTableOutput(outputId, icon, ...)`



`renderImage(expr, env, quoted, deleteFile)`

`imageOutput(outputId, width, height, click,
dblclick, hover, hoverDelay, hoverDelayType,
brush, clickId, hoverId, inline)`



`renderPlot(expr, width, height, res, ..., env,
quoted, func)`

`plotOutput(outputId, width, height, click,
dblclick, hover, hoverDelay, hoverDelayType,
brush, clickId, hoverId, inline)`

`Inputs: Input A: 1.00, Input B: 2.00, Input C: 3.00
Input A: Length: min: 0.0, max: 4.0
Input B: Mean: 1.5, SD: 0.2`

`renderPrint(expr, env, quoted, func,
width)`

`verbatimTextOutput(outputId)`

Inputs	Input A	Input B	Input C	Input D
1	1.00	1.00	1.00	1.00
2	1.00	1.00	1.00	1.00
3	1.00	1.00	1.00	1.00
4	1.00	1.00	1.00	1.00
5	1.00	1.00	1.00	1.00
6	1.00	1.00	1.00	1.00
7	1.00	1.00	1.00	1.00
8	1.00	1.00	1.00	1.00
9	1.00	1.00	1.00	1.00
10	1.00	1.00	1.00	1.00

`renderTable(expr,..., env, quoted, func)`

`tableOutput(outputId)`

`foo`

`renderText(expr, env, quoted, func)`

`textOutput(outputId, container, inline)`

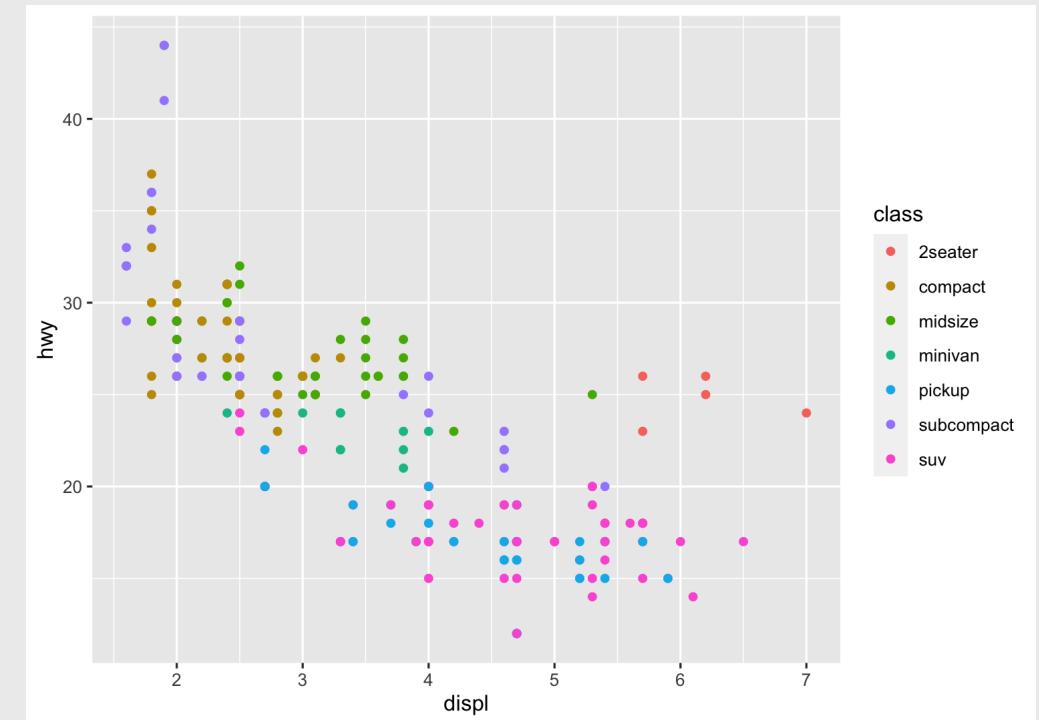
`renderUI(expr, env, quoted, func)`

`uiOutput(outputId, inline, container, ...)`

& `htmlOutput(outputId, inline, container, ...)`

Using `renderPlot()`: make a plot

```
library(ggplot2)  
  
ggplot(mpg) +  
  geom_point(  
    aes(x = displ, y = hwy, color = class))
```



Link plot to output with `output$<id>`

ui

```
ui <- fluidPage(  
  sidebarLayout(  
    sidebarPanel(  
      radioButtons(...)  
    ),  
    mainPanel(  
      plotOutput(  
        outputId = "mpg_plot"  
      )  
    )  
  )
```

server

```
server <- function(input, output, session) {  
  output$mpg_plot <- renderPlot({  
    ggplot(mpg) +  
    geom_point(  
      aes(x = displ, y = hwy, color = class))  
  })  
}
```

Link user inputs to plot with `input$<id>`

ui

```
ui <- fluidPage(  
  sidebarLayout(  
    sidebarPanel(  
      radioButtons(  
        inputId = "xvar",  
        label = "Select the x-axis variable:",  
        selected = "displ",  
        choices = c(  
          "Highway miles per gallon" = "hwy",  
          "City miles per gallon" = "cty",  
          "Engine displacement, in litres" = "displ")  
        ),  
        ),  
      mainPanel(  
        plotOutput(  
          outputId = "mpg_plot"  
        )  
      )  
    )  
)
```

server

```
server <- function(input, output, session) {  
  output$mpg_plot <- renderPlot({  
    ggplot(mpg) +  
    geom_point(  
      aes_string(  
        x = input$xvar,  
        y = "hwy",  
        color = "class"))  
  })  
}
```

Note: I switched the ggplot code from `aes()` to `aes_string()`

Quick practice

Open the `apps/mpg.R` file and click the "Run App" button

10:00

Your Turn

1. Open the `caseConverter.R` file.
2. In the `server`: Write code in the provided `renderText()` to convert the input text to lower case.
3. Run the app and test that it's working.
4. In the `ui` main panel: Add two more `textOutput()` functions for also displaying the input text in "upper" case and "title" case.
5. In the `server`: Define two more outputs to convert the input text to "upper" case and "title" case.

Intermission



05 : 00

Week 14: *Shiny Apps*

1. Anatomy of a Shiny App

2. User Interface

3. Server

Intermission

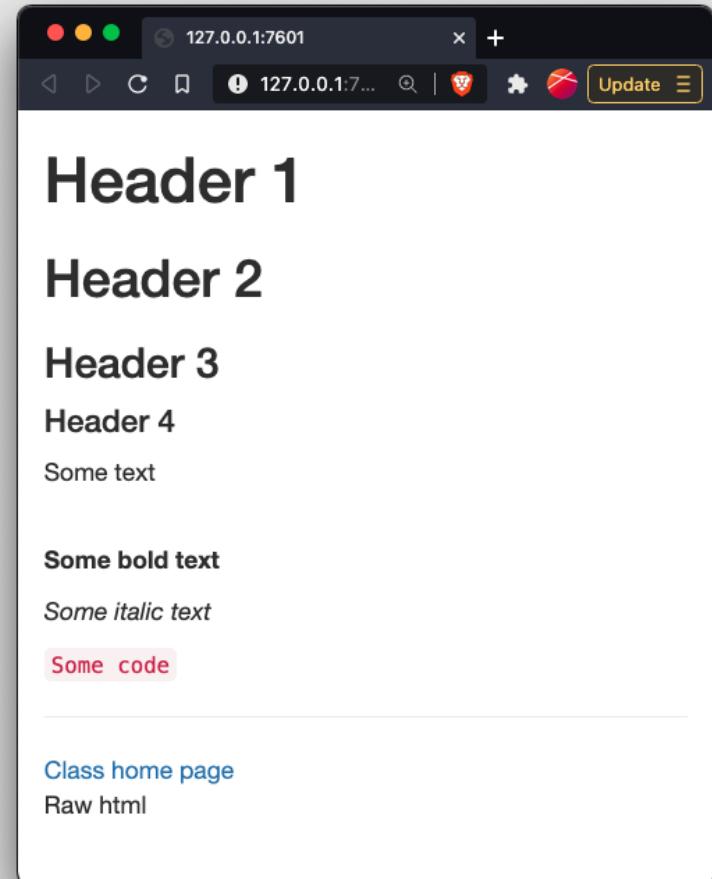
4. Extras

5. Deploying your App

Use html functions to add text

See example in `apps/html.R`

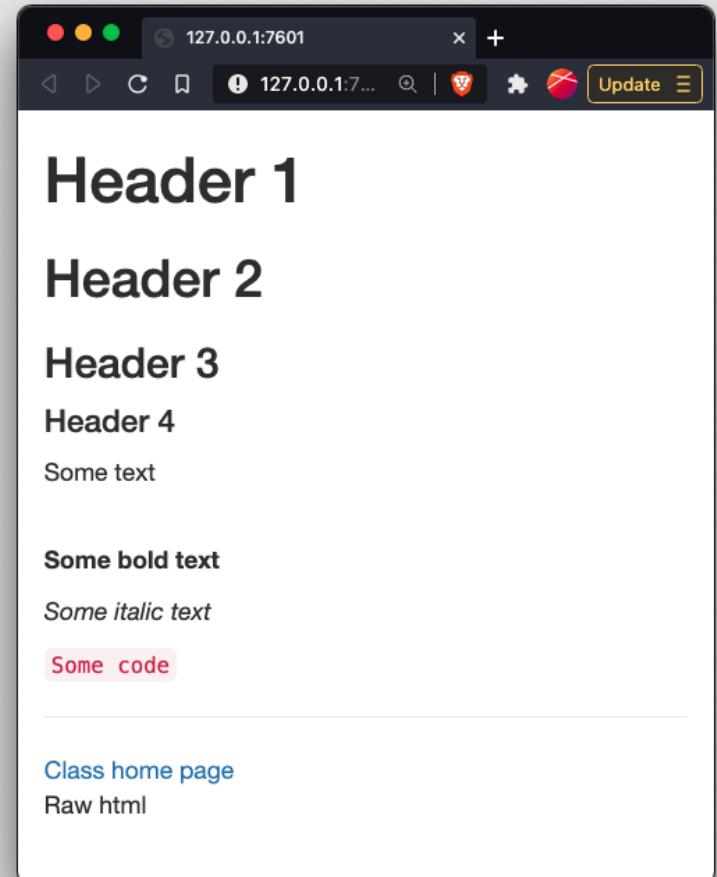
```
ui <- fluidPage(  
  h1("Header 1"),  
  h2("Header 2"),  
  h3("Header 3"),  
  h4("Header 4"),  
  p("Some text"),  
  br(),  
  p(strong("Some bold text")),  
  p(em("Some italic text")),  
  p(code("Some code")),  
  hr(),  
  a(href="https://eda.seas.gwu.edu/2021-Spring/",  
    "Class home page"),  
  HTML("<p>Raw html</p>")  
)
```



Can also just use plain markdown

See example in `apps/markdown.R`

```
ui <- fluidPage(  
  markdown(  
    "# Header 1  
    ## Header 2  
    ### Header 3  
    ##### Header 4  
    Some text  
    **Some bold text**  
    _Some italic text_  
    `Some code`  
    [Class home page](https://eda.seas.gwu.edu/2021-Sp)  
  ")  
)
```



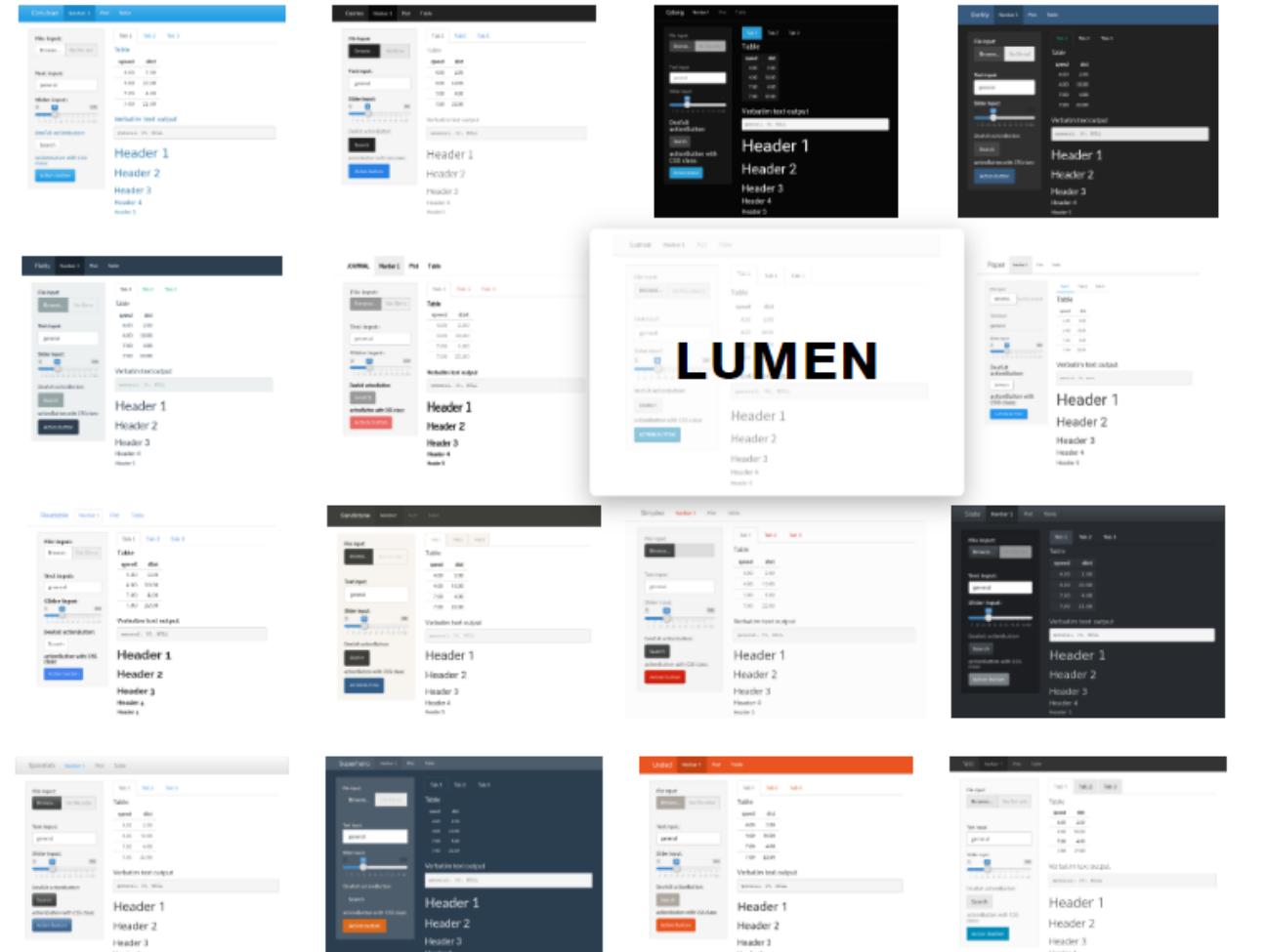
02:00

Quick practice

Open your `caseConverter.R` app and add html functions or markdown to add the following:

- Give the app a title by inserting a level 1 header in the `fluidPage()` function and before the `sidebarLayout()` function.
- In the main `ui` panel, add a short description of what the app does.

Add a theme with "shinythemes" package



Insert theme at top of main ui layout function

```
library(shinythemes)

ui <- fluidPage(
  theme = shinytheme("sandstone"),
  sidebarLayout(
    sidebarPanel(
      <insert widgets>
    ),
    mainPanel(
      <insert outputs>
    )
  )
)
```

The screenshot shows a Shiny application interface with the following components:

- Header:** A dark header bar with tabs: "Sandstone" (selected), "NAVBAR 1", "PLOT", and "TABLE".
- File input:** A file selection input with a "BROWSE..." button and a message "No file sele".
- Text input:** A text input field containing the value "general".
- Slider input:** A slider with a value of 30, ranging from 1 to 100.
- Default actionButton:** A standard "SEARCH" button.
- actionButton with CSS class:** An "ACTION BUTTON" button with a blue background.
- NAVBAR 1:** A tab bar with three tabs: "TAB 1" (selected), "TAB 2", and "TAB 3".
- PLOT:** A section labeled "Table" showing a data frame with columns "speed" and "dist". The data is as follows:

speed	dist
4.00	2.00
4.00	10.00
7.00	4.00
7.00	22.00

- TABLE:** A section labeled "Verbatim text output" containing the text "general, 30, NULL".

Header 1

Header 2

Header 3

Header 4

Header 5

Fancier widgets with "shinyWidgets" package

Open the `apps/shinyWidgets.R` file and click the "Run App" button

Common shiny situations

- Filtering for a single category: `federalSpending.R`
- Filtering for multiple categories: `federalSpendingCompare.R`

Filtering for a single category: `federalSpending.R`

```
ui <- fluidPage(  
  h1("Federal R&D Spending by Department")  
  sidebarLayout(  
    sidebarPanel(  
      selectInput(  
        inputId = "department",  
        label = "Choose a department:",  
        selected = "DOD",  
        choices = c(...))  
    ),  
    mainPanel(  
      plotOutput("spendingPlot"))  
  ))
```

```
server <- function(input, output){  
  output$spendingPlot <- renderPlot({  
    # Filter out the data based on the user input  
    data <- federal_spending %>%  
      filter(department == input$department)  
  
    ggplot(data) +  
      geom_col(  
        aes(x = year, y = rd_budget),  
        fill = "steelblue", width = 0.7, alpha = 0.8)  
    scale_y_continuous(  
      labels = scales::dollar,  
      expand = expansion(mult = c(0, 0.05))) +  
    theme_half_open(font_size = 18) +  
    labs(  
      x = "Year",  
      y = "$USD Millions",  
      title = paste("Federal R&D Spending"))  
  })  
}
```

Filtering for multiple categories: federalSpendingCompare.R

```
ui <- fluidPage(  
  h1("Federal R&D Spending by Department")  
  sidebarLayout(  
    sidebarPanel(  
      selectInput(  
        inputId = "department",  
        label = "Choose a department:",  
        selected = "DOD",  
        multiple = TRUE,  
        choices = c(...))  
    ),  
    mainPanel(  
      plotOutput("spendingPlot"))  
  ))
```

```
server <- function(input, output){  
  output$spendingPlot <- renderPlot({  
    # Filter out the data based on the user input  
    data <- federal_spending %>%  
      filter(department %in% input$department)  
  
    ggplot(data) +  
      geom_col(  
        aes(x = year, y = rd_budget),  
        fill = "steelblue", width = 0.7, alpha = 0.8)  
    facet_wrap(~department) +  
    scale_y_continuous(  
      labels = scales::dollar,  
      expand = expansion(mult = c(0, 0.05))) +  
    theme_half_open(font_size = 18) +  
    labs(  
      x = "Year",  
      y = "$USD Millions",  
      title = paste("Federal R&D Spending"))  
  })  
}
```

If you really want to get good at this:

1. Print out this [Cheatsheet](#)
2. Watch this [2.5 Hour Comprehensive RStudio Tutorial](#)
3. Use this reference manual: [Mastering Shiny](#)

Week 14: *Shiny Apps*

1. Anatomy of a Shiny App

2. User Interface

3. Server

Intermission

4. Extras

5. Deploying your App

You can deploy an app for free on [shinyapps.io](#)

Follow [the RStudio guide](#)

1. Create a shinyapps.io account
2. Open your tokens, click "Show", copy the code
3. Run the code in RStudio
4. Deploy your app:

```
library(rsconnect)  
deployApp()
```

15:00

Your Turn

1. Open the `internetUsers.R` file.
2. Modify the server code so that the inputs control the plot.
3. Deploy your app to shinyapps.io

Fill out course evals: <https://gwu.smartevals.com/>
(please be specific!)



PLEEEEEEASE!