Data Engineering
COMP2031/8031

- Topic Coordinator: Dr Mehwish Nasim
- Office: 3.13 Tonsley

Flinders
UNIVERSITY

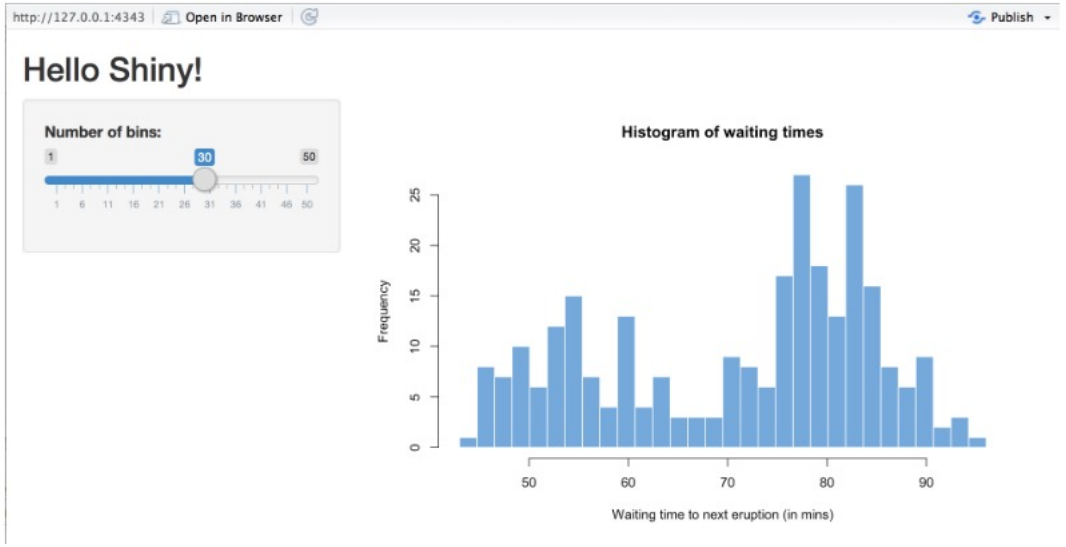
Shiny

- an R package that makes it easy to build interactive web applications (apps) straight from R
- https://shiny.rstudio.com/tutorial/written-tutorial/lesson1/

- install.packages("shiny")

example

- library(shiny)
- runExample("01_hello")



Flinders UNIVERSITY

Structure

- Shiny apps are contained in a single script called app.R. The script app.R lives in a directory (for example, newdir/) and the app can be run with runApp("newdir").
- app.R has three components:
  - a user interface object
  - a server function
  - a call to the shinyApp function
- The user interface (ui) object controls the layout and appearance of your app.
- The server function contains the instructions that your computer needs to build your app.
- Finally the shinyApp function creates Shiny app objects from an explicit UI/server pair.

```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(

    # 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")

    )
  )
)
```

```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")

    })

}
```

```r
library(shiny)

# See above for the definitions of ui and server
ui <- ...

server <- ...

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

Running the App

- library(shiny)
- runApp("my_app")

Practice

- Launch your app by running runApp("App-1"). Then click escape and make some changes to your app:
  - Change the title from “Hello Shiny!” to “Hello World!”.
  - Set the minimum value of the slider bar to 5.
  - Change the histogram border color from "white" to "orange".

Building a UI

- Open app.R file:

```
library(shiny)

# Define UI ----
ui <- fluidPage(

)

# Define server logic ----
server <- function(input, output) {

}

# Run the app ----
shinyApp(ui = ui, server = server)
```

Layout

- Shiny uses the function fluidPage to create a display that automatically adjusts to the dimensions of your user's browser window. You lay out the user interface of your app by placing elements in the fluidPage function.

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

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

- titlePanel and sidebarLayout are the two most popular elements to add to fluidPage. They create a basic Shiny app with a sidebar.
- sidebarLayout always takes two arguments:
  - sidebarPanel function output
  - mainPanel function output
- These functions place content in either the sidebar or the main panels.
- The sidebar panel will appear on the left side of your app by default. You can move it to the right side by giving sidebarLayout the optional argument position = "right".

HTML elements

- You can add content to your Shiny app by placing it inside a *Panel function

shiny function	HTML5 equivalent	creates
`p`	`<p>`	A paragraph of text
`h1`	`<h1>`	A first level header
`h2`	`<h2>`	A second level header
`h3`	`<h3>`	A third level header
`h4`	`<h4>`	A fourth level header
`h5`	`<h5>`	A fifth level header
`h6`	`<h6>`	A sixth level header
`a`	`<a>`	A hyper link
`br`	` `	A line break (e.g. a blank line)
`div`	`<div>`	A division of text with a uniform style
`span`	`<span>`	An in-line division of text with a uniform style
`pre`	`<pre>`	Text 'as is' in a fixed width font
`code`	`<code>`	A formatted block of code
`img`	`<img>`	An image
`strong`	`<strong>`	Bold text
`em`	`<em>`	Italicized text
`HTML`		Directly passes a character string as HTML code

Flinders UNIVERSITY

Header

- To create a header element:
  - select a header function (e.g., h1 or h5)
  - give it the text you want to see in the header
- h1("My title")
- <h1>My title</h1>

```
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")
    )
  )
)
```

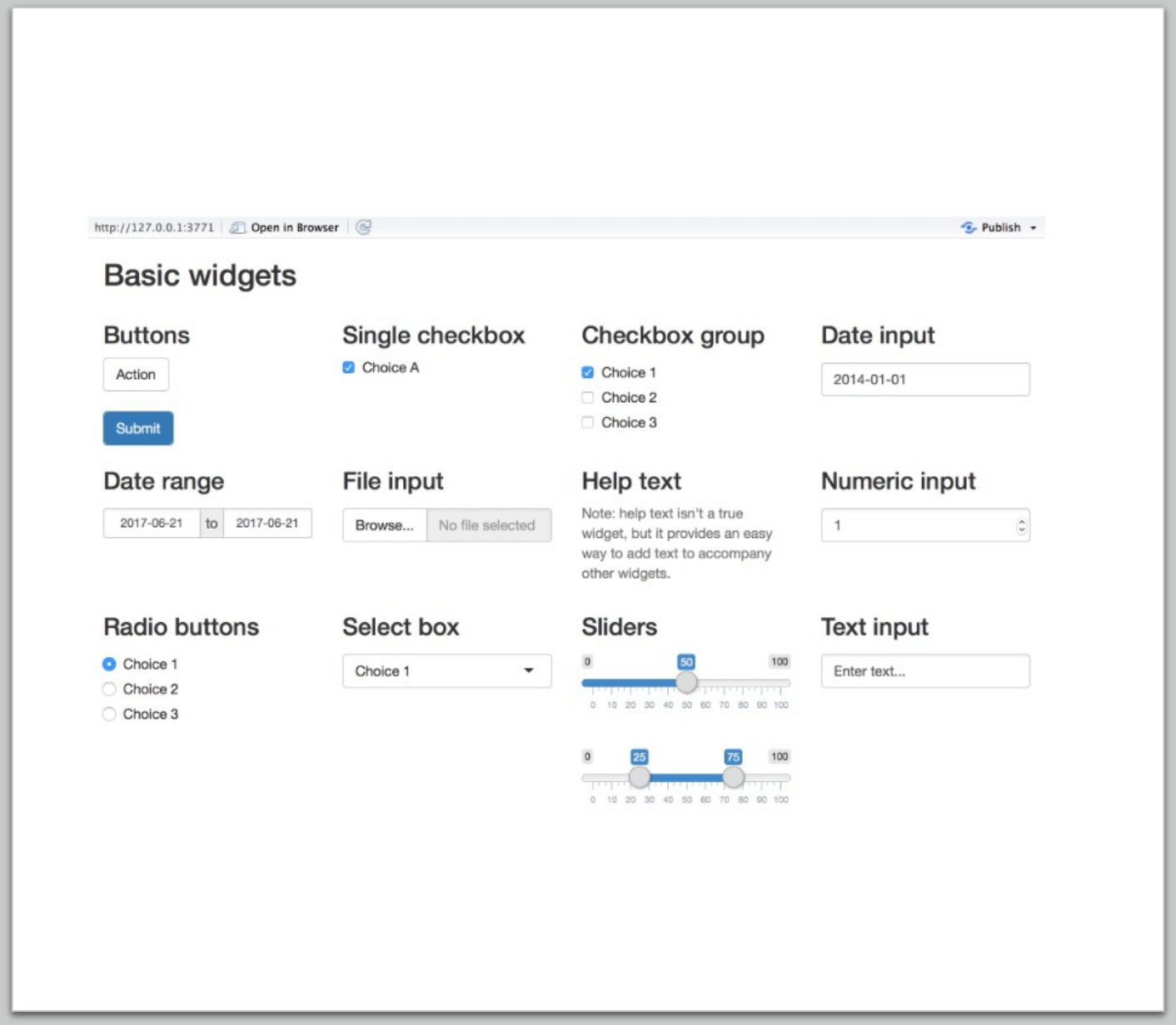
Further reading

- https://shiny.rstudio.com/tutorial/written-tutorial/lesson2/

Add Control Widgets

Flinders UNIVERSITY

- Widgets provide a way for your users to send messages to the Shiny app



Flinders UNIVERSITY

Standard Widgets

function	widget
`actionButton`	Action Button
`checkboxGroupInput`	A group of check boxes
`checkboxInput`	A single check box
`dateInput`	A calendar to aid date selection
`dateRangeInput`	A pair of calendars for selecting a date range
`fileInput`	A file upload control wizard
`helpText`	Help text that can be added to an input form
`numericInput`	A field to enter numbers
`radioButtons`	A set of radio buttons
`selectInput`	A box with choices to select from
`sliderInput`	A slider bar
`submitButton`	A submit button
`textInput`	A field to enter text

- Each widget function requires several arguments. The first two arguments for each widget are
  - a **name for the widget**: The user will not see this name, but you can use it to access the widget's value. The name should be a character string.
  - a **label**: This label will appear with the widget in your app. It should be a character string, but it can be an empty string "".
- Look at the R code

Reactive Output

- You can create reactive output with a two step process.
  - Add an R object to your user interface.
  - Tell Shiny how to build the object in the server function. The object will be reactive if the code that builds it calls a widget value.

Add an object..

- Shiny provides a family of functions that turn R objects into output for your user interface. Each function creates a specific type of output.

Output function	Creates
`dataTableOutput`	DataTable
`htmlOutput`	raw HTML
`imageOutput`	image
`plotOutput`	plot
`tableOutput`	table
`textOutput`	text
`uiOutput`	raw HTML
`verbatimTextOutput`	text

Flinders UNIVERSITY

```
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 = c("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(
      textOutput("selected_var")
    )
  )
)
```

Provide R code to build the object

```
server <- function(input, output) {

  output$selected_var <- renderText({
    "You have selected this"
  })

}
```

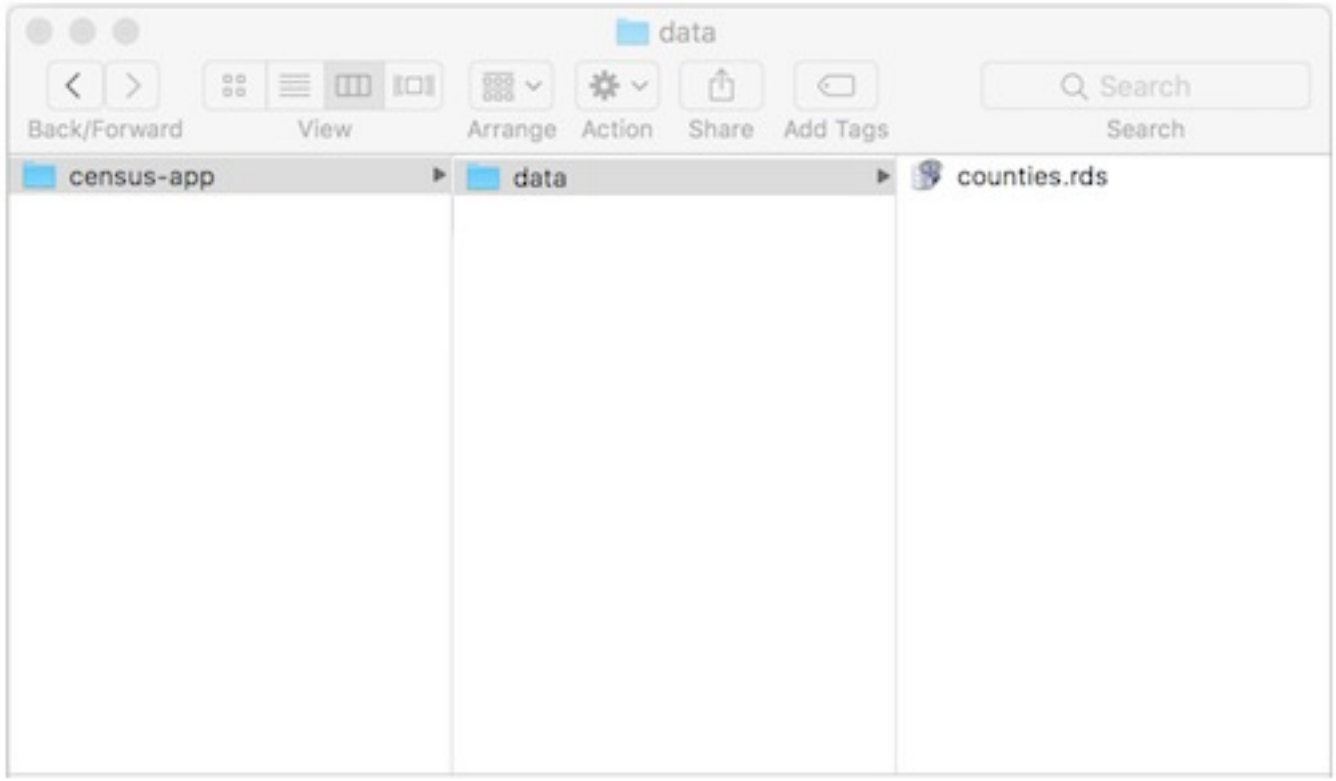
```
server <- function(input, output) {

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

}
```

Use R scripts and data

- counties.rds is a dataset of demographic data for each county in the United States, collected with the UScensus2010 R package. You can download it here

- https://shiny.rstudio.com/tutorial/written-tutorial/lesson5/census-app/data/counties.rds

- Once you have the file,
  - Create a new folder named data in your census-app directory.
  - Move counties.rds into the data folder.
- When you’re done, your census-app folder should look like this.



Flinders UNIVERSITY

- The dataset in counties.rds contains
  - the name of each county in the United States
  - the total population of the county
  - the percent of residents in the county who are White, Black, Hispanic, or Asian

Helpers.R

- helpers.R is an R script that can help you make choropleth maps
- You can download helpers.R here
- Save helpers.R inside your census-app directory
- install.packages(c("maps", "mapproj"))

- The percent_map function in helpers.R takes five arguments:

`var`	a column vector from the `counties.rds` dataset
`color`	any character string you see in the output of `colors()`
`legend.title`	A character string to use as the title of the plot's legend
`max`	A parameter for controlling shade range (defaults to 100)
`min`	A parameter for controlling shade range (defaults to 0)

You can use `percent_map` at the command line to plot the counties data as a choropleth map, like this.