Research Data Workshop Series - Introduction to R Shiny

Lucas Alcantara, Ph.D.
alcantal@uoguelph.ca

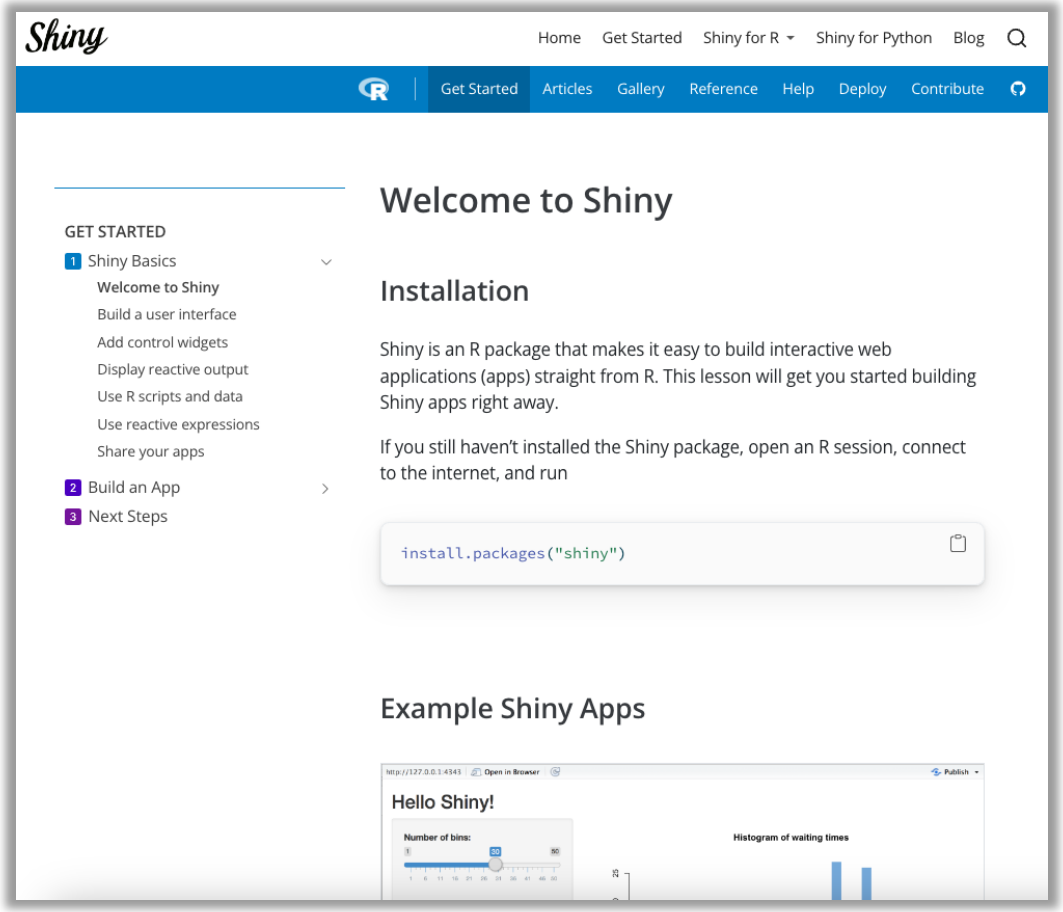
AGRI-FOOD DATA CANADA
AT THE UNIVERSITY *of* GUELPH

Workshop Outline

- Welcome to Shiny
- Build a user interface
- Add control widgets
- Display reactive output
- Use R scripts and data
- Use reactive expressions
- Share your apps

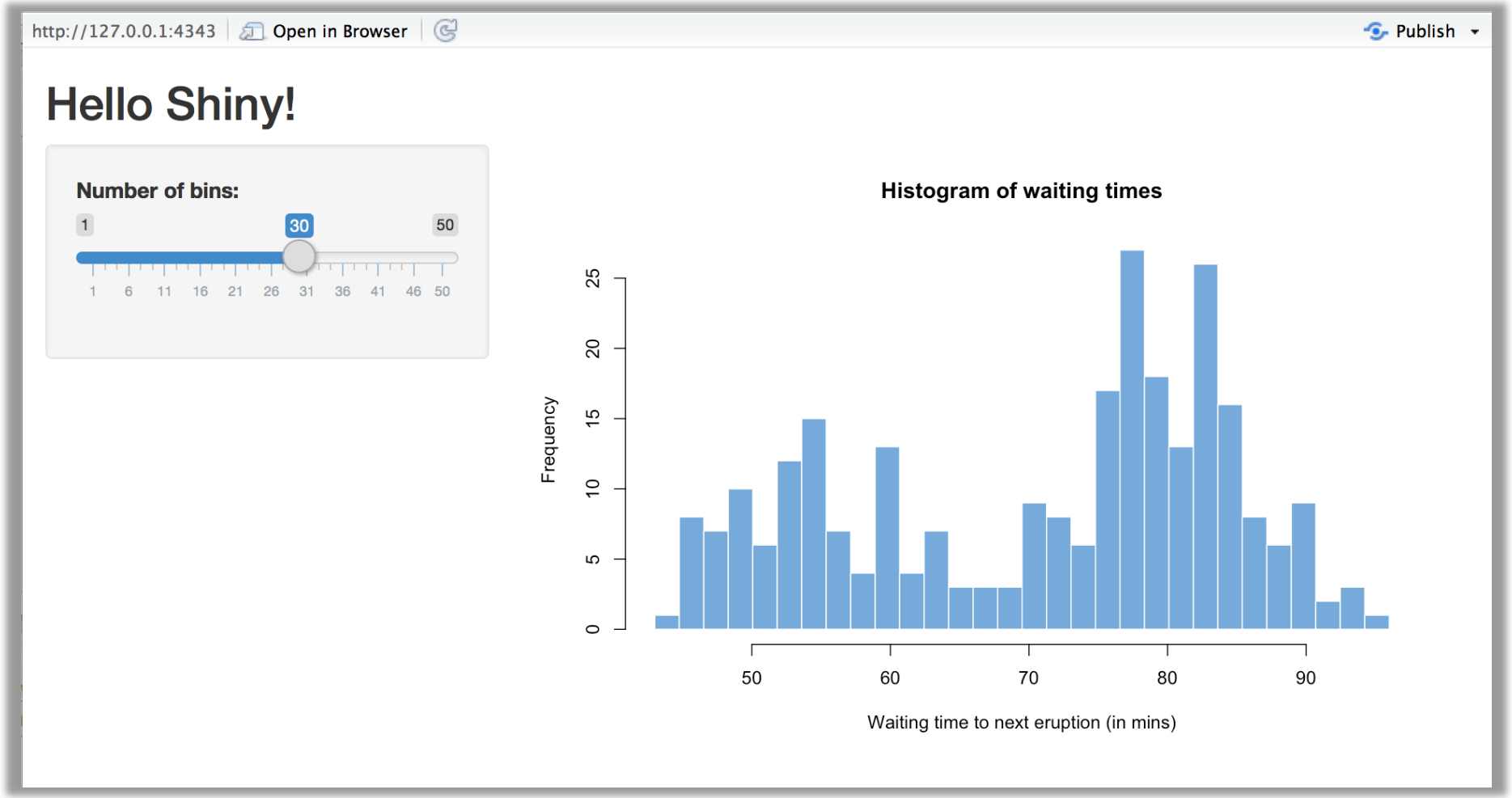
Based on Posit's Shiny Primer

- https://shiny.posit.co/r/getstarted



Welcome to Shiny

Welcome to Shiny



Welcome to Shiny

Installation

- If you still haven't installed the Shiny package, open an R session, connect to the internet, and run

```
> install.packages("shiny")
```

Example Apps

- The Shiny package has eleven built-in examples that each demonstrate how Shiny works. Each example is a self-contained Shiny app

```
> library(shiny)
> runExample("01_hello")
```

Welcome to Shiny

Structure of a Shiny App

- The simplest form of a Shiny app is contained in a single script called app.R, which contains three components:
  - a user interface object
  - a server() function
  - a call to the shinyApp() function

Let’s inspect the Hello Shiny App’s code

Welcome to Shiny

Structure of a Shiny App

- The simplest form of a Shiny app is contained in a single script called app.R, which contains three components:
  - a User Interface (UI) object – layout and appearance of the app
  - a server() function – logics behind the app
  - a call to the shinyApp() function – creates the app from UI-server pair

Let’s inspect the Hello Shiny App’s code

Welcome to Shiny

Running an App

- You can create a Shiny app by making a new directory and saving an app.R file inside it.
  - It is recommended that each app will live in its own unique directory.
- You can run a Shiny app by giving the name of its directory to the function runApp().
- For example, if your Shiny app is in a directory called my_app, run it with the following code:

```
library(shiny)
runApp(”./my_app")
```


9

Welcome to Shiny

Your turn!

1. Create a new directory named my_app in your ***working directory***
2. Create a new script called app.R
3. Copy the code from the Hello Shiny sample app and paste into app.R
4. Launch your shiny app with the function runApp()

Welcome to Shiny

Let’s try changing some things on app.R

1. Change the title from “Hello Shiny!” to “Hello World!”.
2. Set the minimum value of the slider bar to 5.
3. Change the histogram border color from "white" to "orange"

Welcome to Shiny

Relaunching Apps

- On the R console
  - Run runApp(”./my_app")
- On RStudio
  - Open the app.R script and click the Run App button.
  - Use a keyboard shortcut
    - MacOS Command + Shift + Enter
    - Windows Control + Shift + Enter
    - Posit Cloud Any of the above

Welcome to Shiny

Go Further

The Shiny gallery (https://shiny.posit.co/gallery) provides some good examples. You can use any of the eleven pre-built Shiny examples listed below as a starting point:

```
runExample("01_hello")          # a histogram
runExample("02_text")           # tables and data frames
runExample("03_reactivity")     # a reactive expression
runExample("04_mpg")            # global variables
runExample("05_sliders")        # slider bars
runExample("06_tabsets")        # tabbed panels
runExample("07_widgets").       # help text and submit buttons
runExample("08_html")           # Shiny app built from HTML
runExample("09_upload")         # file upload wizard
runExample("10_download")       # file download wizard
runExample("11_timer")          # an automated timer
```

Build a user interface

Build a user interface

Layout

- Shiny uses the fluidPage() function 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")
        )
)
```


More about layouts: https://shiny.posit.co/r/articles/build/layout-guide

Build a user interface

HTML Content

- To add more advanced content, use one of Shiny's 110 HTML tag functions.

Shiny function	HTML5 equivalent	Creates
p()	<p>	A paragraph text
h1(), h2(), … h6()	<h1>, <h2>, … <h6>	A 1st, 2nd, … 6th level header
img()	<img>	An image
br()	 	A line break (i.e., a blank line)
hr()	<hr>	A horizontal line
code()	<code>	A formatted block of code
HTML()		Directly pass a character string as HTML code

More about HTML tags: https://shiny.posit.co/r/articles/build/tag-glossary

16

Build a user interface

HTML Content

- In general, any HTML tag attribute can be set as an argument in any Shiny tag function. For example, you can center-align your text

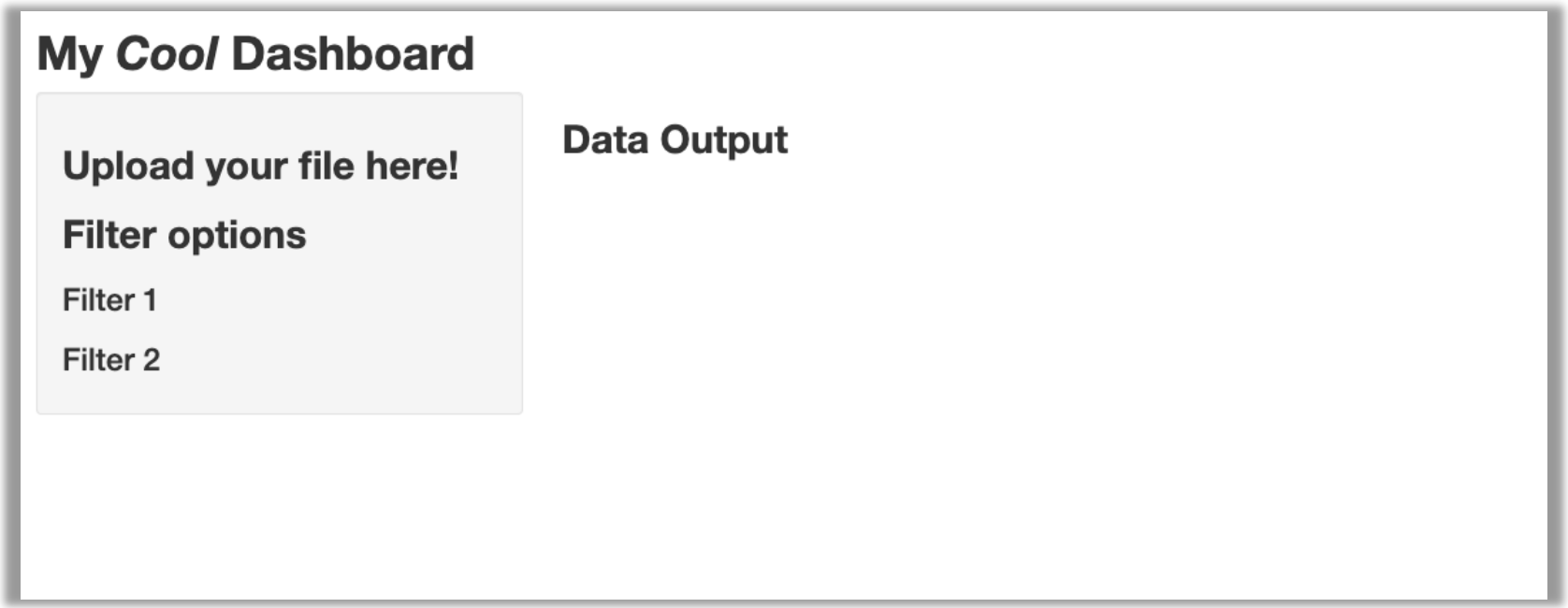
```
ui <- fluidPage(
        titlePanel("My Star Wars App"),
        sidebarLayout(
                sidebarPanel(),
                mainPanel(
                        h6("Episode IV", align = "center"),
                        h6("A NEW HOPE", align = "center"),
                        h5("It is a period of civil war.", align = "center"),
                        h4("Rebel spaceships, striking", align = "center"),
                        h3("from a hidden base, have won", align = "center"),
                        h2("their first victory against the", align = "center"),
                        h1("evil Galactic Empire.", align = "center")
                )
        )
)
```


More about HTML tags: https://shiny.posit.co/r/articles/build/tag-glossary

Build a user interface

Your turn

- Modify your app.R to display the app just like below



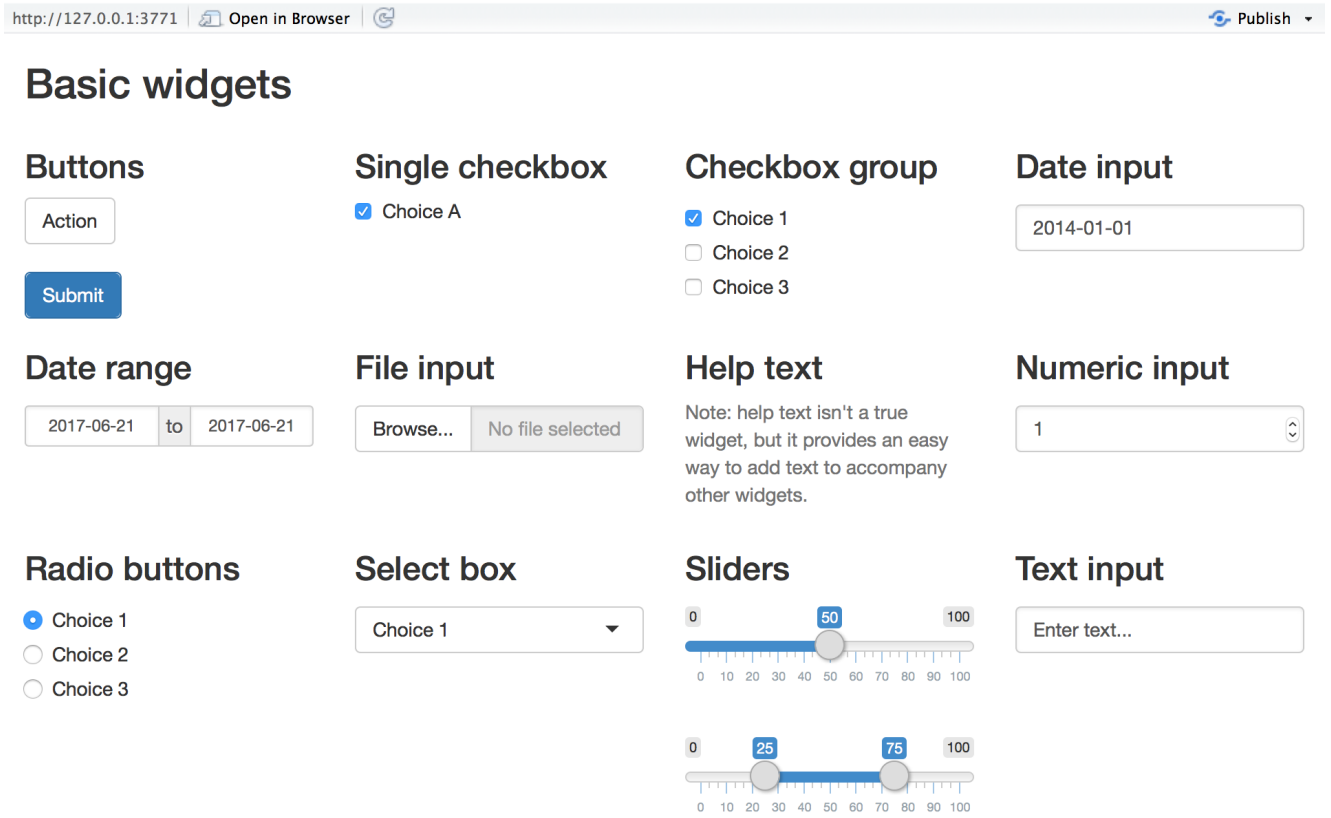
More about HTML tags: https://shiny.posit.co/r/articles/build/tag-glossary

Add Control Widgets

Add control widgets

Control widgets

- Web elements that users can interact with and send messages to the Shiny App
- Widgets collect a value from the user. If the user changes the widget, the value will change as well

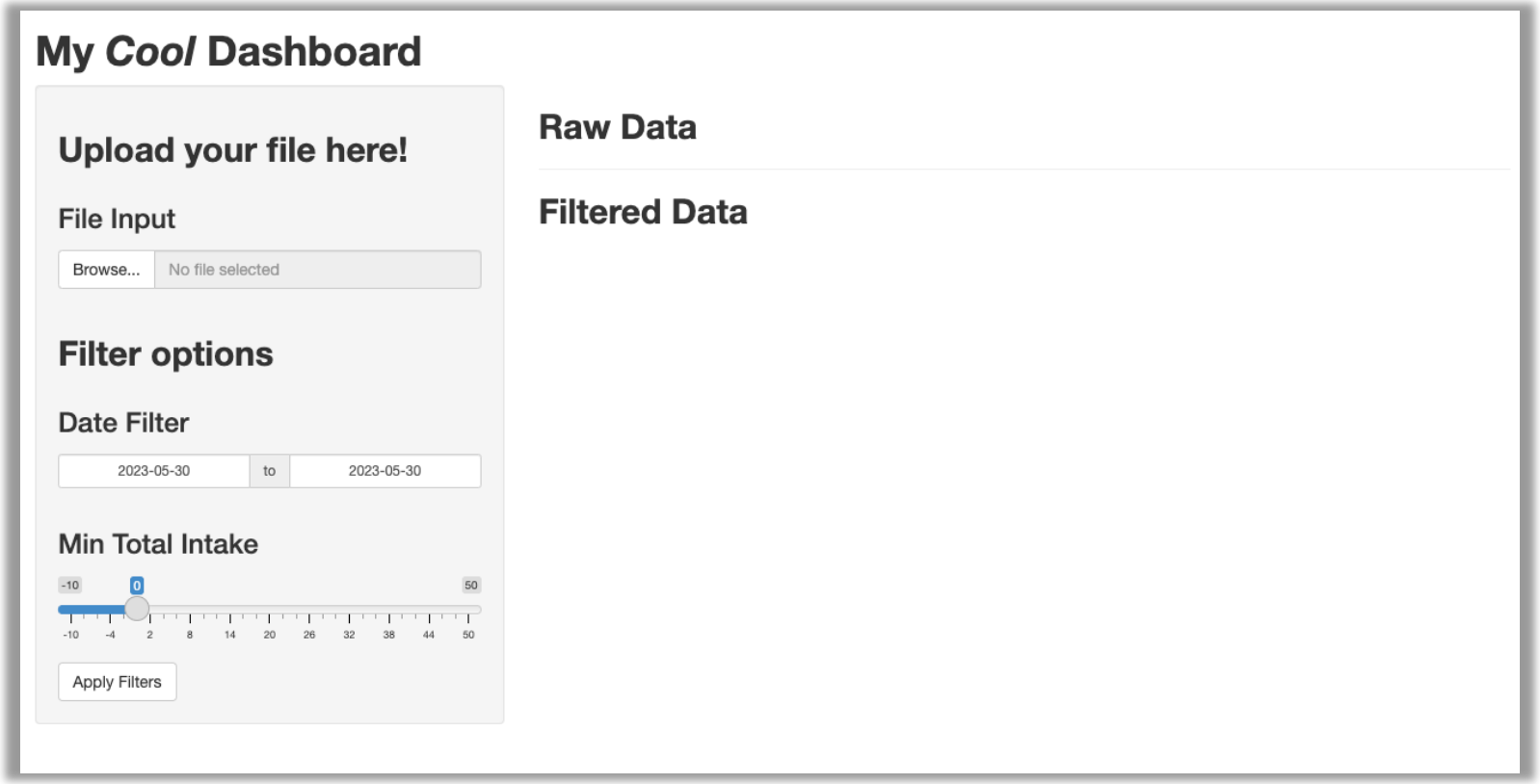


More about widgets: http://shiny.rstudio.com/gallery/widget-gallery.html

Add control widgets

Your turn

- Modify your app.R to display the app just like below



Display Reactive Output

Display reactive output

Reactive output

- It automatically responds when you toggle a widget
- Two steps (where and how):
    1. Add an R object to your user interface
    2. Tell Shiny how to build the object in the server function

Display reactive output

1. Add an R object to your user interface

- Shiny provides a family of functions that turn R objects into output for your user interface. Each function creates a specific type of output
- Each of the *Output functions require a single argument: outputId

e.g.: textOutput(outputId = "selected_dates")

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

Display reactive output

2. Tell Shiny how to build the object in the server function

- The server function builds a list-like object named output that contains all codes needed to update the R objects in your app
- Each R object needs to have its own entry in the list
- The new output element name should match the name of the reactive element that you created in the UI

Render function	Creates
renderDataTable	DataTable
renderImage	images (saved as a link to a source file)
renderPlot	plots
renderPrint	any printed output
renderTable	data frame, matrix, other table like structures
renderText	character strings
renderUI	a Shiny tag object or HTML

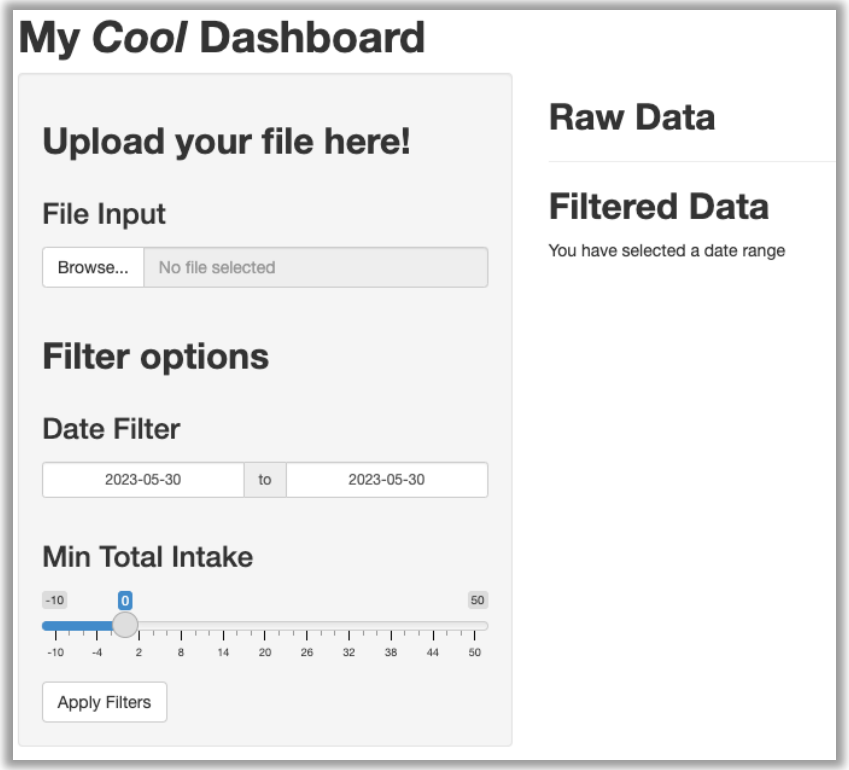
Display reactive output

1. Add an R object to your user interface
2. Tell Shiny how to build the object in the server function

```
    ),

    # Main panel for displaying outputs ----
    mainPanel(
      h2(tags$b("Raw Data")),
      hr(),
      h2(tags$b("Filtered Data")),
      textOutput("selected_dates")
    )
  )
)

# Define server logic required for the app ----
server <- function(input, output) {
  output$selected_dates <- renderText({
    "You have selected a date range"
  })
}
```



Display reactive output

Your turn!

1. Add another textOutput object to tell display a message about the selected total intake
2. Add a renderText function to render such message to your new output object

Display reactive output

Things to consider about reactive functions

- The function can be one simple line of text, or it can involve many lines of code
- Shiny will run all functions when you first launch your app
- Shiny will re-run them every time it needs to update your objects (in the UI)

Display reactive output

Use widget values

- input is another list-like object, but it stores the current values of all widgets
- These values will be saved under the names that you gave the widgets in your UI

```
   tags$b("Dashboard")
),
# Sidebar layout with input and output definitions ----
sidebarLayout(
  # Side panel with input widgets ----
  sidebarPanel(
    h2(tags$b("Upload your file here!")),
    fileInput("file_input", h3("File Input")),

    h2(tags$b("Filter options")),
    dateRangeInput("date_range", h3("Date Filter")),
    sliderInput("intake", h3("Min Total Intake"),
                min = -10, max = 50, step = 2, value = 0),
    actionButton("submit", "Apply Filters")
  ),

  # Main panel for displaying outputs ----
  mainPanel(
```

```
# Define server logic required for the app ----
server <- function(input, output) {
  output$selected_dates <- renderText({
    paste("You have selected",
          input$date_range[1],
          "and",
          input$date_range[2])
  })
}
```

Display reactive output

Your turn!

1. Add another server function to output the value selected for the total intake

Read and Manipulate Data

Read and Manipulate Data

Read Data

- File is read into memory and can be accessed through the input object
- file_input is the inputId defined in the UI
- datapath is created by the fileInput widget and is the path to a temp file that contains the uploaded data

```
output$raw_data <- renderTable({
  req(input$file_input)

  df <- read.csv(input$file_input$datapath)

  head(df)

})
```

Read and Manipulate Data

Your turn

- Use the renderTable, read.csv and some of the tidyverse functions to

1. Upload data into R
2. Read this data
3. Filter the date column to be between the specified date range
4. Render the resulting first 6 lines

Read and Manipulate Data

Reactive Expressions

- A reactive expression is an R expression that uses widget input and returns a value
- The reactive expression will update this value only when the original widget changes

```
df <- reactive({
  req(input$file_input)
  read.csv(input$file_input$datapath)
})

output$raw_data <- renderTable({
  head(df())
})

output$filtered_data <- renderTable({
  data <- df() %>%
    filter(date >= input$date_range[1],
           date <= input$date_range[2])
  head(data)

})
```


34

Read and Manipulate Data

Your turn

- Create a new reactive expression for the filtered data and use that reactive to

1. Render the resulting first 6 lines, just like before
2. Render a message stating the number of rows on the filtered data

Control the Reactive Flow

Control the Reactive Flow

Event Reactives

- Sometimes you need to perform expensive computations with the parameters given to your widgets
- If a computation is triggered after each time there is a new value for a widget, this is not very efficient
- eventReactive are key to control the reactive flow, i.e.: to tell Shiny when to execute a function

Control the Reactive Flow

Your turn!

- Create an action button to control the rendering of the raw data

THANK YOU!

Questions?

Feel free to reach me later at
alcantal@uoguelph.ca

AGRI-FOOD DATA CANADA
AT THE UNIVERSITY *of* GUELPH