

Research Data Workshop Series - Introduction to R Shiny

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



**AGRI-FOOD DATA
CANADA**

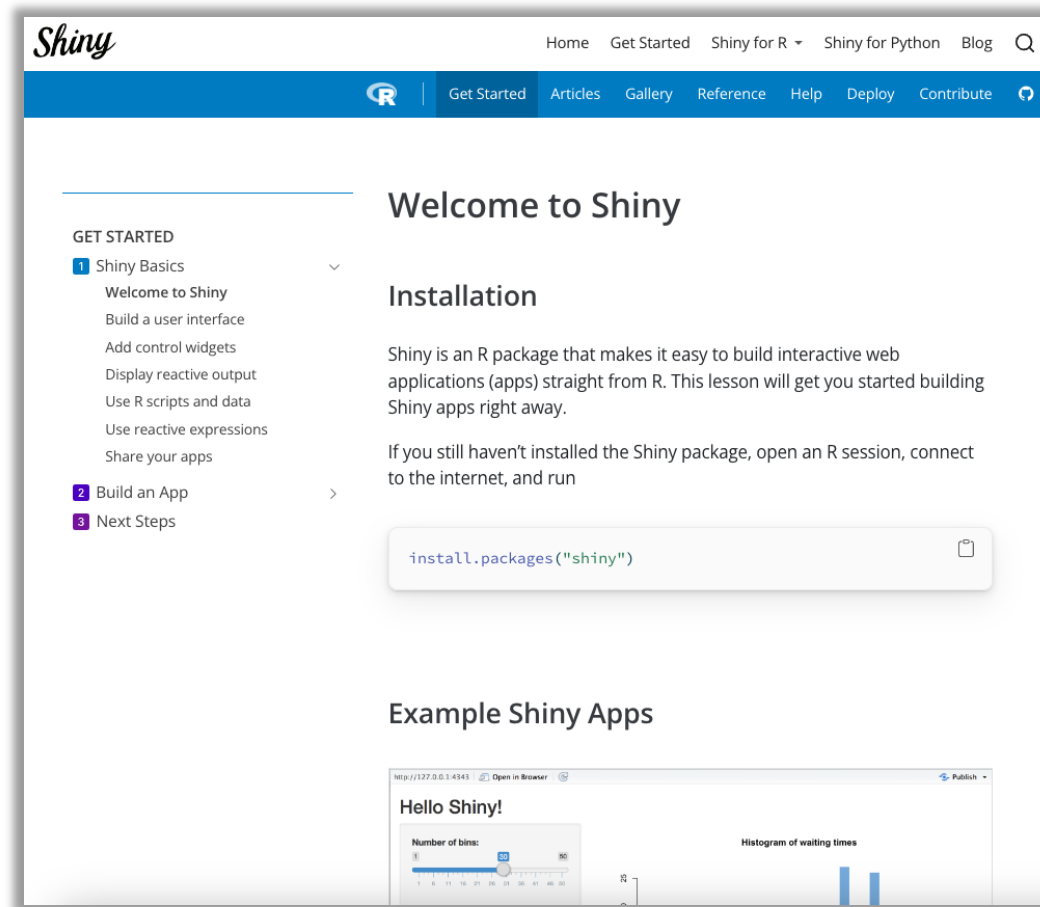
AT THE UNIVERSITY *of* GUELPH

Workshop Outline

- Welcome to Shiny
- Build a user interface
- Add control widgets
- Display reactive output
- Read and manipulate data
- Control the reactive flow

Based on Posit's Shiny Primer

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



The screenshot shows the Shiny website's 'Welcome to Shiny' page. The page has a blue header with the Shiny logo and navigation links: Home, Get Started, Shiny for R, Shiny for Python, and Blog. Below the header is a secondary navigation bar with links: Get Started, Articles, Gallery, Reference, Help, Deploy, and Contribute. The main content area is divided into two columns. The left column is a sidebar titled 'GET STARTED' with a list of links: 1 Shiny Basics (expanded), 2 Build an App, and 3 Next Steps. The 'Shiny Basics' section is further divided into: Welcome to Shiny, Build a user interface, Add control widgets, Display reactive output, Use R scripts and data, Use reactive expressions, and Share your apps. The right column is titled 'Welcome to Shiny' and contains an 'Installation' section. The installation text reads: 'Shiny is an R package that makes it easy to build interactive web applications (apps) straight from R. This lesson will get you started building Shiny apps right away.' Below this text is a code block containing the command `install.packages("shiny")`. Further down is an 'Example Shiny Apps' section, which includes a preview of a Shiny app interface. The app interface shows a 'Hello Shiny!' message, a 'Number of bins' slider set to 50, and a 'Histogram of waiting times' plot with two bars.

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")
```

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:

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

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")  
  )  
)
```

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

Build a user interface

HTML Content

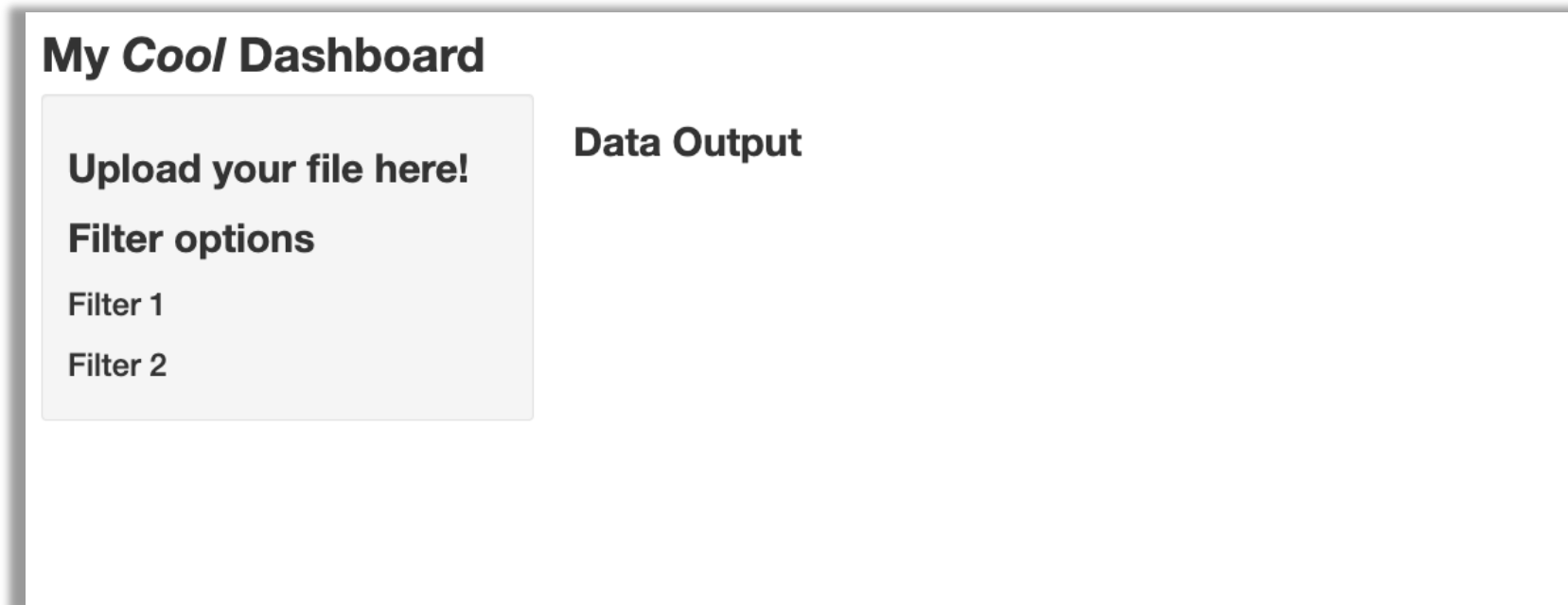
- In general, any HTML tag attribute can be set as an argument in any Shiny tag function. For example, you can center-align your text

```
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")  
    )  
  )  
)
```

Build a user interface

Your turn

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



The screenshot shows a Shiny dashboard with the following layout:

- My Cool Dashboard** (Title)
- Upload your file here!** (Text)
- Filter options** (Section header)
 - Filter 1
 - Filter 2
- Data Output** (Text)

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

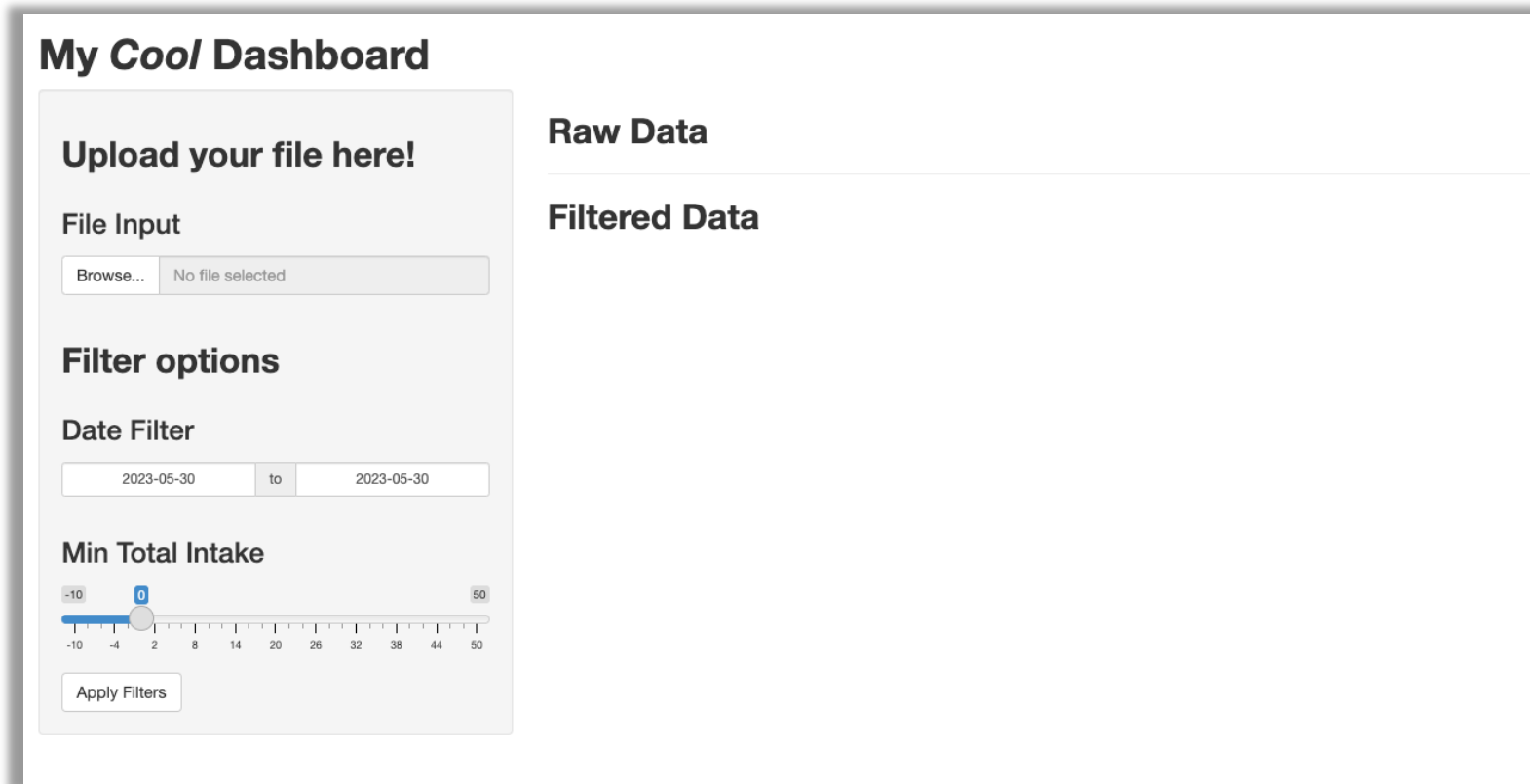
The screenshot displays a web browser interface with a URL bar showing 'http://127.0.0.1:3771' and a 'Publish' button. The main content area is titled 'Basic widgets' and is organized into a grid of 12 categories, each with a corresponding widget example:

- Buttons:** Two buttons labeled 'Action' and 'Submit'.
- Single checkbox:** A checkbox labeled 'Choice A' which is checked.
- Checkbox group:** Three checkboxes labeled 'Choice 1', 'Choice 2', and 'Choice 3', with 'Choice 1' checked.
- Date input:** A text input field containing the date '2014-01-01'.
- Date range:** A date range input field showing '2017-06-21 to 2017-06-21'.
- File input:** A file input field with a 'Browse...' button and the text 'No file selected'.
- Help text:** A text area containing the note: 'Note: help text isn't a true widget, but it provides an easy way to add text to accompany other widgets.'
- Numeric input:** A numeric input field containing the value '1'.
- Radio buttons:** Three radio buttons labeled 'Choice 1', 'Choice 2', and 'Choice 3', with 'Choice 1' selected.
- Select box:** A dropdown menu showing 'Choice 1'.
- Sliders:** Two slider widgets. The top one has a value of 50, and the bottom one has a value of 75. Both range from 0 to 100.
- Text input:** A text input field with the placeholder text 'Enter text...'.

Add control widgets

Your turn

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



The screenshot shows a Shiny dashboard titled "My Cool Dashboard". On the left side, there is a control panel with the following elements:

- Upload your file here!**: A section for file upload.
- File Input**: A text input field with a "Browse..." button and the text "No file selected".
- Filter options**: A section for filtering data.
- Date Filter**: Two date input fields, both containing "2023-05-30", with a "to" label between them.
- Min Total Intake**: A slider widget with a range from -10 to 50. The current value is 0.
- Apply Filters**: A button to apply the selected filters.

On the right side of the dashboard, there are two empty sections:

- Raw Data**: A section for displaying the raw data.
- Filtered Data**: A section for displaying the data after applying filters.

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
<code>dataTableOutput</code>	DataTable
<code>htmlOutput</code>	raw HTML
<code>imageOutput</code>	image
<code>plotOutput</code>	plot
<code>tableOutput</code>	table
<code>textOutput</code>	text
<code>uiOutput</code>	raw HTML
<code>verbatimTextOutput</code>	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"  
  })  
}
```

My Cool Dashboard

Upload your file here!

File Input

Browse... No file selected

Filter options

Date Filter

2023-05-30 to 2023-05-30

Min Total Intake

-10 0 50

-10 -4 2 8 14 20 26 32 38 44 50

Apply Filters

Raw Data

Filtered Data

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)
})
```

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