# Introduction to Shiny Part 1


## Luke McGuinness

**Twitter: @mcguinlu**

**Deptartment of Population Health Sciences,
Bristol Medical School**


**16th March, 2021**

# Overview of this session

- Background to `shiny`
- Getting started
  - Set-up
  - Control widgets & User interface
  - Outputs
- Getting more from `shiny`
  - Execution
  - Customising your app
  - Reactive programming
  - Publishing your app

# Public Service Announcements

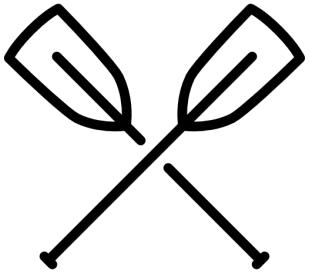
## Workshop plan

- Lecture followed by workshop
- There will be a break for coffee in the middle

## Please ask questions as we go along!

# Public Service Announcements

## I pronounce the letter "R" oddly:



R
("oar")

**means**

R
("arr")

# Introduction: Background to `shiny`

# What is `shiny`?

`shiny` is an R package that allows users to build interactive web applications ("apps") straight from R.

# What does Shiny do?

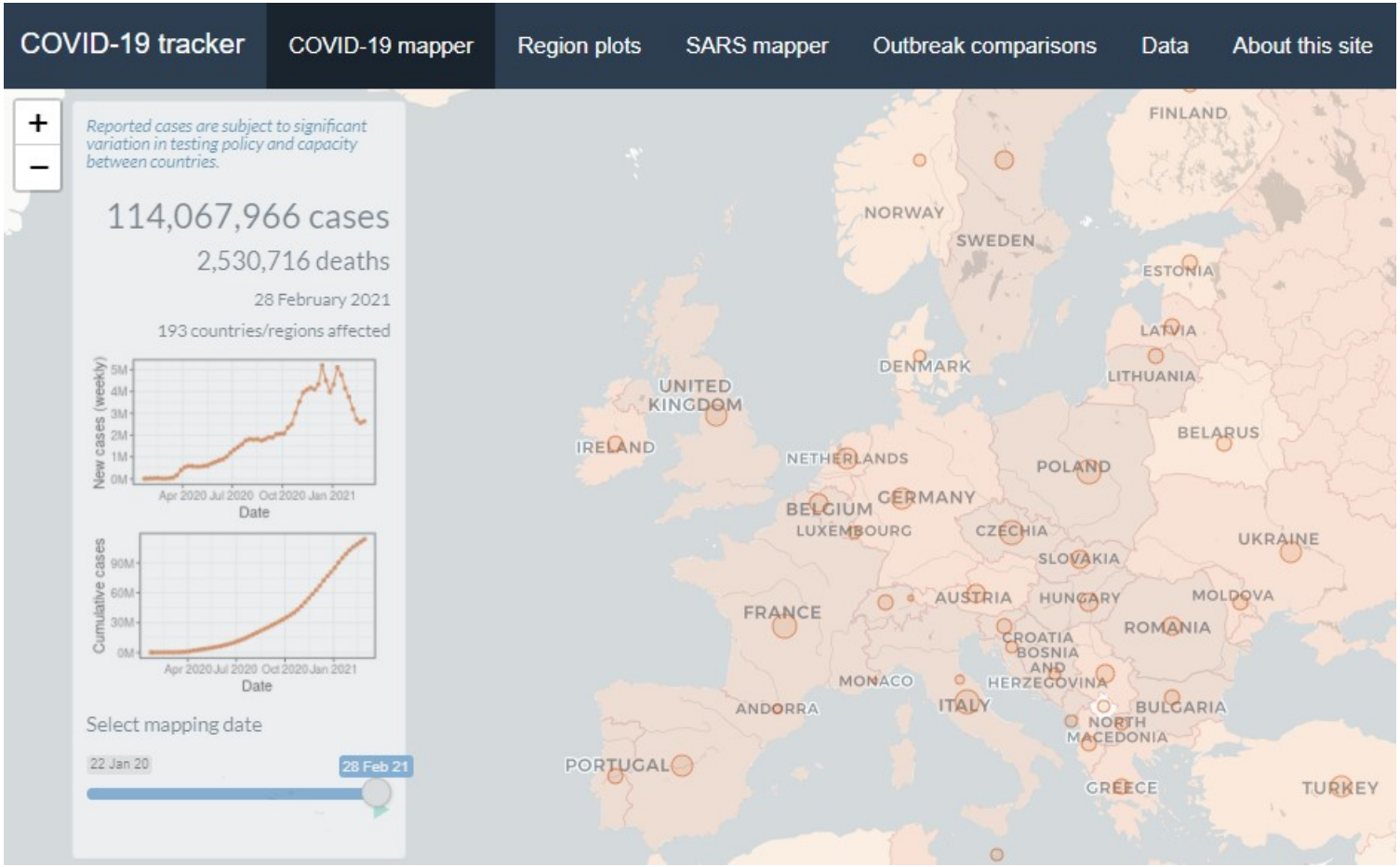
**Variable to fill by:**

Health

**Error:** An error has occurred. Check your logs or contact the app author for clarification.

*Data shown is from the `med` dataset used in this practical*

# What does Shiny do?



# Structure of a `shiny` app

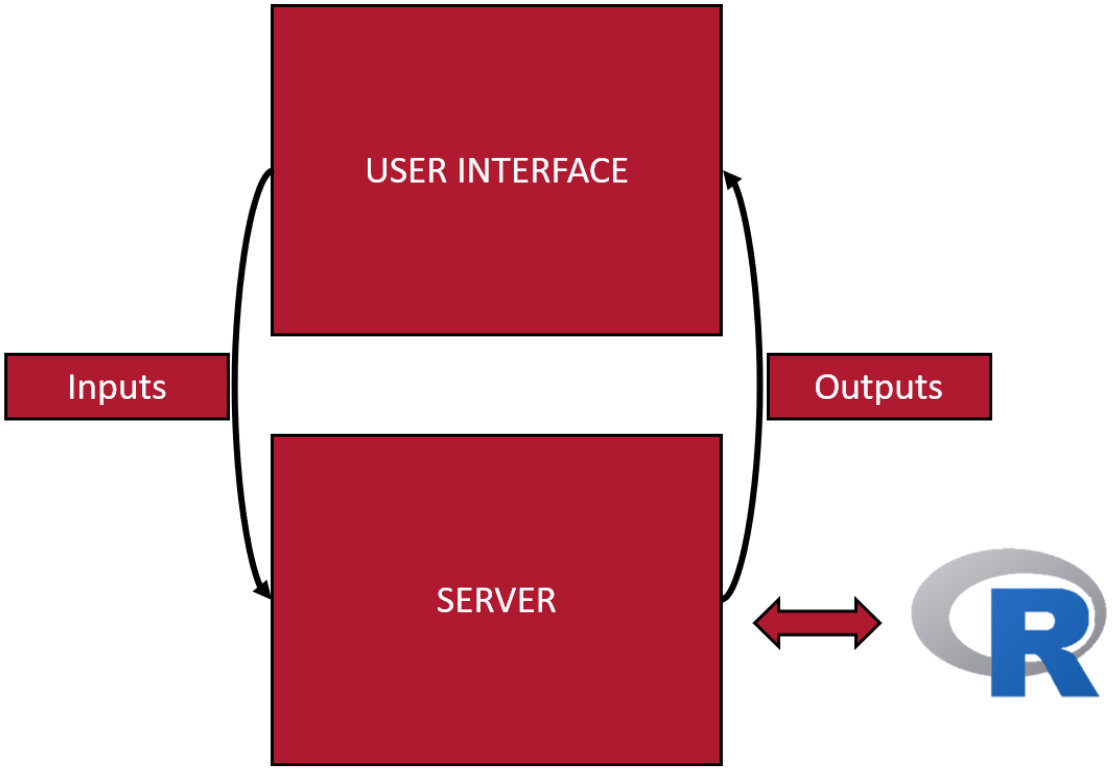
A `shiny` app has three components:

- **User interface**
  - Defines the layout of your app
  - Controls what it looks like (themes/fonts/etc)
- **Server**
  - Defines the logic needed to build the app
  - Performs computational work
- **Call to the `shinyApp` function**
  - Creates the app from an user interface and server pair

# Inputs and outputs

Information moves between the UI and the server via inputs and outputs

NB: No computation takes place in UI - only the server can run R functions

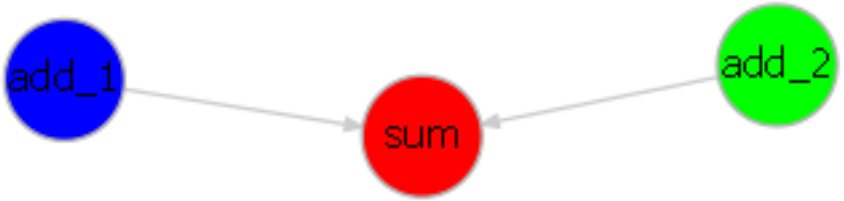


# A simple app: add two numbers

```r
ui <- fluidPage(

  # Allow user to define two numbers
  numericInput(inputId = "add_1", label = "Number:", value = 0),
  numericInput(inputId = "add_2", label = "Number:", value = 0),

  # Display the output
  textOutput(outputId = "sum")
)

server <- function(input, output) {

  output$sum <- renderText({
    input$add_1 + input$add_2
  })
}

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

# Reactive programming

- Outputs (the sum of two numbers) *take a dependency on* inputs (the numbers being added)
- When the inputs change, the outputs change



# A simple app: add two numbers

```
ui <- fluidPage(

  # Allow user to define two numbers
  numericInput(inputId = "add_1", label = "Number:", value = 0),
  numericInput(inputId = "add_2", label = "Number:", value = 0),

  # Display the output
  textOutput(outputId = "sum")
)

server <- function(input, output) {

  output$sum <- renderText({
    input$add_1 + input$add_2
  })
}

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

# A simple app: add two numbers

```
ui <- fluidPage(

  # Allow user to define two numbers
  numericInput(inputId = "add_1", label = "Number:", value = 0),
  numericInput(inputId = "add_2", label = "Number:", value = 0),

  # Display the output
  textOutput(outputId = "sum")
)

server <- function(input, output) {

  output$sum <- renderText({
    input$add_1 + input$add_2
  })
}

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

# A simple app: add two numbers

```
ui <- fluidPage(

  # Allow user to define two numbers
  numericInput(inputId = "add_1", label = "Number:", value = 0),
  numericInput(inputId = "add_2", label = "Number:", value = 0),

  # Display the output
  textOutput(outputId = "sum")
)

server <- function(input, output) {

  output$sum <- renderText({
    input$add_1 + input$add_2
  })
}

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

# Getting started: Control widgets

# A widget for every occasion

Control widgets are used to capture user input and vary based on type of input:

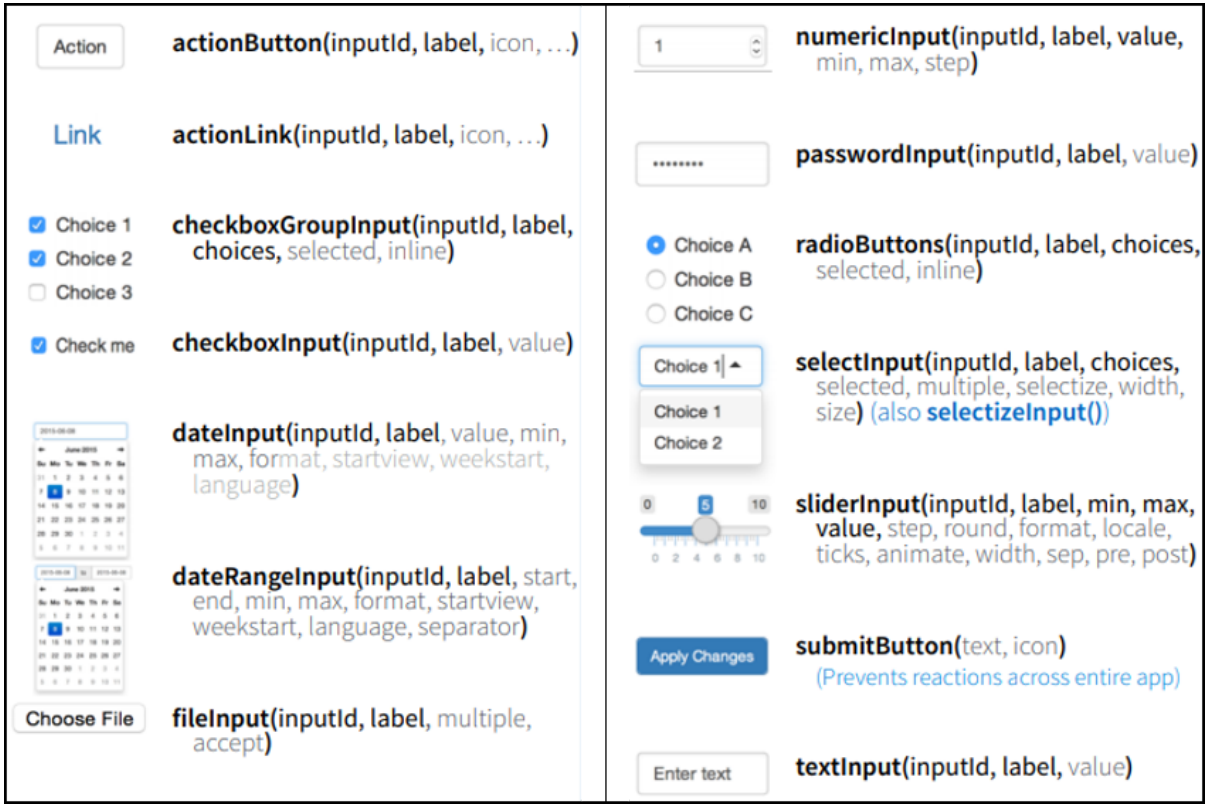



Image sourced from the `shiny` Rstudio cheatsheet

# The anatomy of a widget

We use the `numericInput` widget in our app

```
ui <- fluidPage(

  # Allow user to define two numbers
  numericInput(inputId = "add_1", label = "Number:", value = 0),
  numericInput(inputId = "add_2", label = "Number:", value = 0),

  # Display the output
  textOutput(outputId = "sum")
)

server <- function(input, output) {

  output$sum <- renderText({
    input$add_1 + input$add_2
  })
}

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

# The anatomy of a widget

All control widgets have two elements in common:

- **inputId**: Unique ID for that widget
- **label**: Text to be displayed beside the widget (which can be left blank)

```
textInput(inputId = "textboxinput",
          label = "")
```

Other elements are specific to the widget you are using:

```
selectInput(inputId = "fillby",
            label = "Variable to fill by",
            choices = c("Health","treament")),
```

# The anatomy of a widget

The `inputId` must be unique, so that the value can be used in the server:

```
ui <- fluidPage(

  # Allow user to define two numbers
  numericInput(inputId = "add_1", label = "Number:", value = 0),
  numericInput(inputId = "add_2", label = "Number:", value = 0),

  # Display the output
  textOutput(outputId = "sum")
)

server <- function(input, output) {

  output$sum <- renderText({
    input$add_1 + input$add_2
  })
}

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

# The anatomy of a widget

The value of `label` can be duplicated across widgets:

```
ui <- fluidPage(

  # Allow user to define two numbers
  numericInput(inputId = "add_1", label = "Number:", value = 0),
  numericInput(inputId = "add_2", label = "Number:", value = 0),

  # Display the output
  textOutput(outputId = "sum")
)

server <- function(input, output) {

  output$sum <- renderText({
    input$add_1 + input$add_2
  })
}

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

# The anatomy of a widget

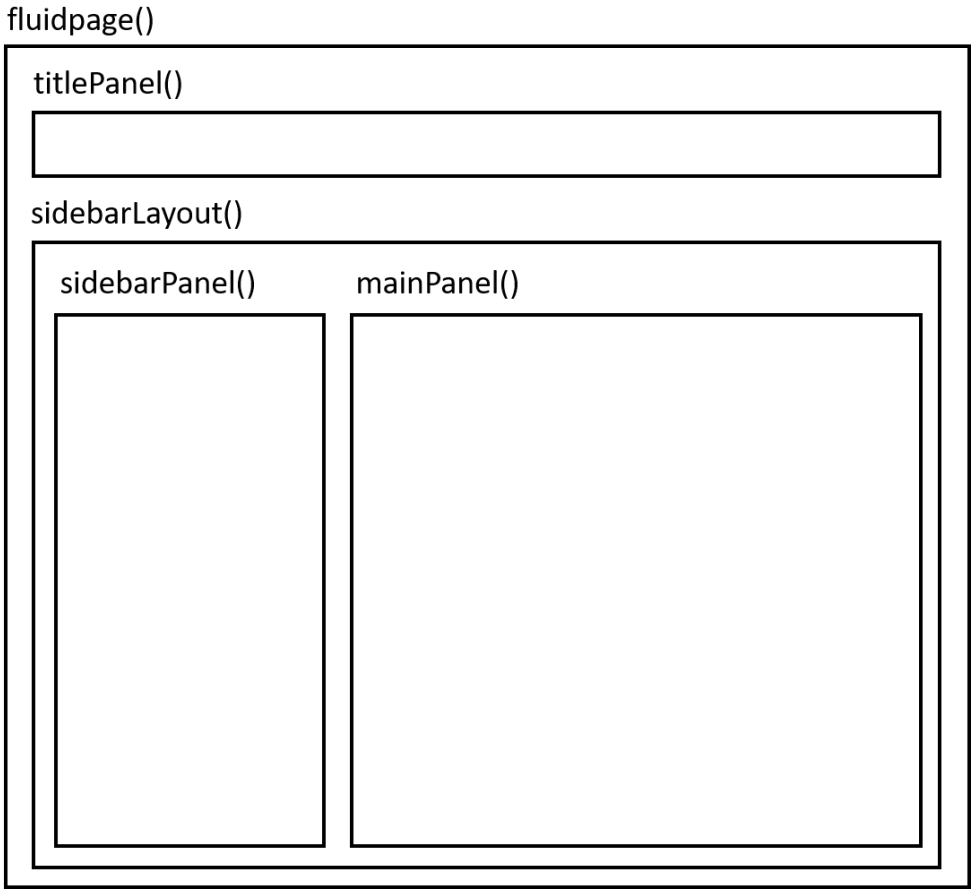
Some widgets require a default value:

```
ui <- fluidPage(

  # Allow user to define two numbers
  numericInput(inputId = "add_1", label = "Number:", value = 0),
  numericInput(inputId = "add_2", label = "Number:", value = 0),

  # Display the output
  textOutput(outputId = "sum")
)

server <- function(input, output) {

  output$sum <- renderText({
    input$add_1 + input$add_2
  })
}

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

# Getting started: User Interface

# `sidebarLayout()`

Common layout for `shiny` apps



# `sidebarLayout()`

```
ui <- fluidPage(
  sidebarLayout(
    sidebarPanel(
      # Control widgets
    ),
    mainPanel(
      # Outputs
    )
  )
)
```

Conventions for use:

- The `sidebarPanel` is usually used to house the control widgets that capture user input.
- The `mainPanel` is usually used to present the output of the app (text/graph/results).

# Applying sidebarLayout() to our app

```
ui <- fluidPage(
  sidebarLayout(
   sidebarPanel(
      # Allow user to define two numbers
      numericInput(inputId = "add_1", label = "Number:", value = 0),
      numericInput(inputId = "add_2", label = "Number:", value = 0),
    ),
    mainPanel(
      # Display the output
      textOutput(outputId = "sum")
    )
  )
)

server <- function(input, output) {

  output$sum <- renderText({
    input$add_1 * input$add_2
  })
}

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

# Creating your own app: Reactive outputs

# Creating outputs - the basics

Add new functionality to show product of the two numbers?

```
ui <- fluidPage(
  # Allow user to define two numbers
  numericInput(inputId = "add_1", label = "Number:", value = 0),
  numericInput(inputId = "add_2", label = "Number:", value = 0),

  # Display the output
  textOutput(outputId = "sum")

)

server <- function(input, output) {

  output$sum <- renderText({
    input$add_1 + input$add_2
  })




}

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

# Creating outputs - the basics

First, add code needed to create the output to the server using a `render*()` function

```
ui <- fluidPage(
  # Allow user to define two numbers
  numericInput(inputId = "add_1", label = "Number:", value = 0),
  numericInput(inputId = "add_2", label = "Number:", value = 0),

  # Display the output
  textOutput(outputId = "sum")

)

server <- function(input, output) {

  output$sum <- renderText({
    input$add_1 + input$add_2
  })

  output$product <- renderText({
    input$add_1 * input$add_2
  })
}

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

# Creating outputs - the basics

Add the resulting object to the user interface using the corresponding `*Output()` function.

```
ui <- fluidPage(
  # Allow user to define two numbers
  numericInput(inputId = "add_1", label = "Number:", value = 0),
  numericInput(inputId = "add_2", label = "Number:", value = 0),

  # Display the output
  textOutput(outputId = "sum")
  textOutput(outputID = "product")
)

server <- function(input, output) {

  output$sum <- renderText({
    input$add_1 + input$add_2
  })

  output$product <- renderText({
    input$add_1 * input$add_2
  })
}

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

# Rendering the output

Similar to inputs, there are different `render*()`/`*Output()` function pairs for different types of output:

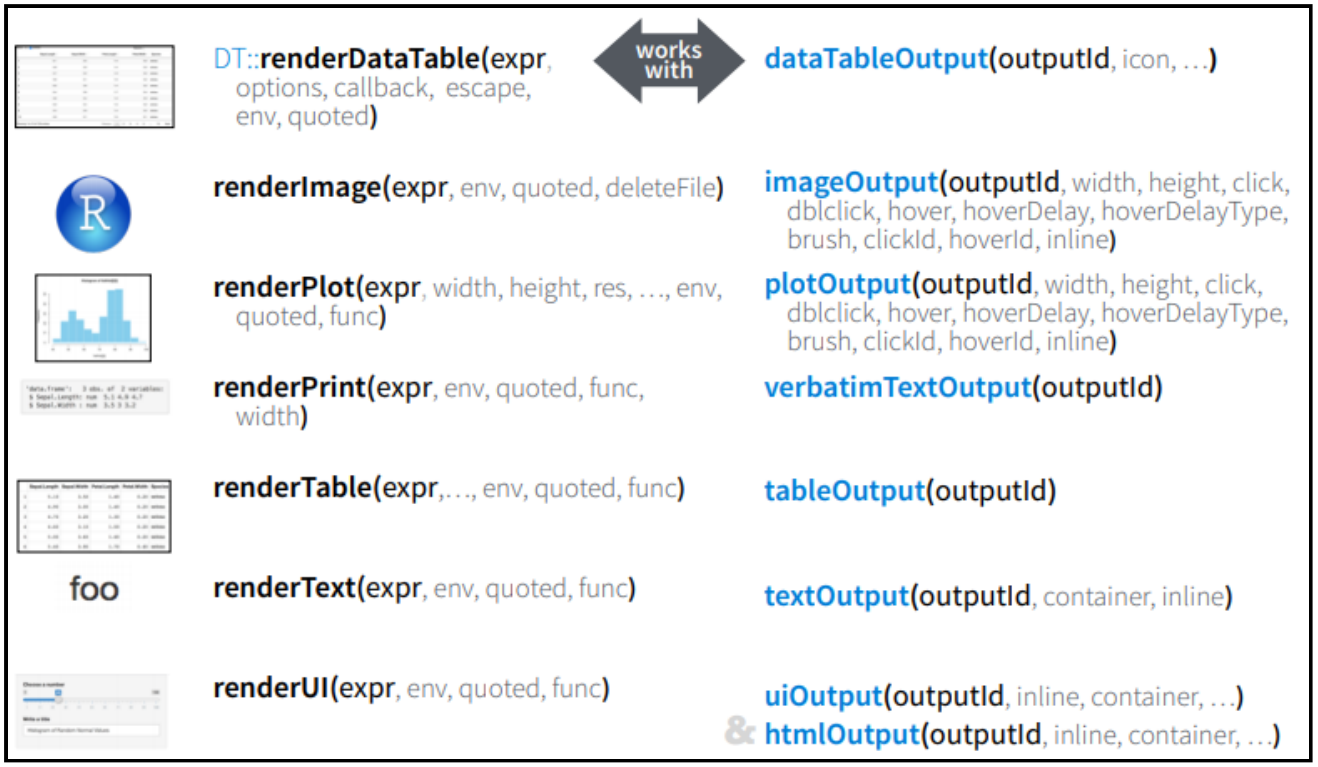



Image sourced from the `shiny` Rstudio cheatsheet

# Accessing widget values

The inputId of the widget is important

---

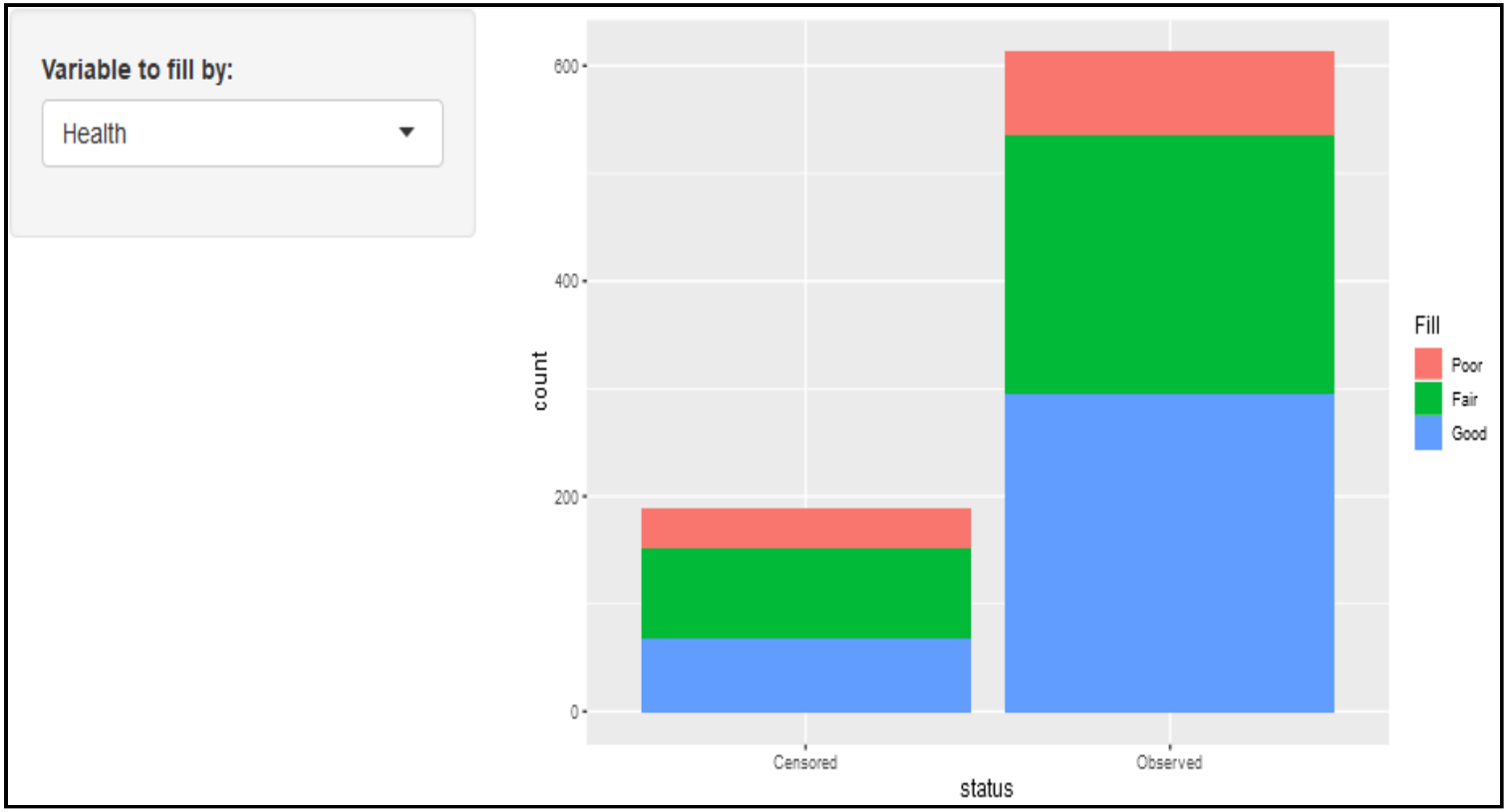
For the following widget:

```
numericInput(inputId = add_1, label = "Number:", value = 0)
```

its value is defined by

```
input$add_1
```

# Using the value of the control widgets



# Using the value of the control widgets

**renderPlot() function in the server**

```
output$barPlot <- renderPlot({
       ggplot(data = med, aes_string(fill = input$fill)) +
          geom_histogram(aes(x = status), stat = "count")
       })
```

---

**Changes based on value selected by user in UI**

When *input$fill = health*:

```
ggplot(data = med, aes_string(fill = "health")) +
   geom_histogram(aes(x = status), stat = "count")
```

When *input$fill = treatment*:

```
ggplot(data = med, aes_string(fill = "treatment")) +
   geom_histogram(aes(x = status), stat = "count")
```

# Running an app

### There are two options when running your app:

Open the app by running `runApp("app-dir")` in the console, where "app-dir" is the name of the directory containing your app.R script:

```
runApp("luke")
```

**OR**

Open the app.R script in Rstudio and then:

- click the "Run App" button:


- Use the keyboard short-cut: *Ctrl/Command+Shift+Enter*

# Introducing the data

The dataset we will use for the practical elements is the `bmi` dataset:

```
bmi <- read.csv("http://bit.ly/bris-data-viz-bmi")
```

| id | age | bmi | sex | diet | status |
|---|---|---|---|---|---|
| 1 | 78.2 | 29.3 | Male | Good | Unhealthy |
| 2 | 48.5 | 33.0 | Female | Good | Unhealthy |
| 3 | 79.5 | 31.5 | Female | Good | Unhealthy |
| 4 | 78.5 | 28.1 | Male | Poor | Healthy |

The data set contains 200 observations across the following six variables:

**Continuous variables:**

- id
- age
- bmi

**Categorical variables:**

- sex: Male / Female
- diet: Good / Moderate / Poor
- status: Healthy / Unhealthy

# Images

Oar: By Florian Pépellin - Own work, CC BY-SA 3.0,
https://commons.wikimedia.org/w/index.php?curid=43042720