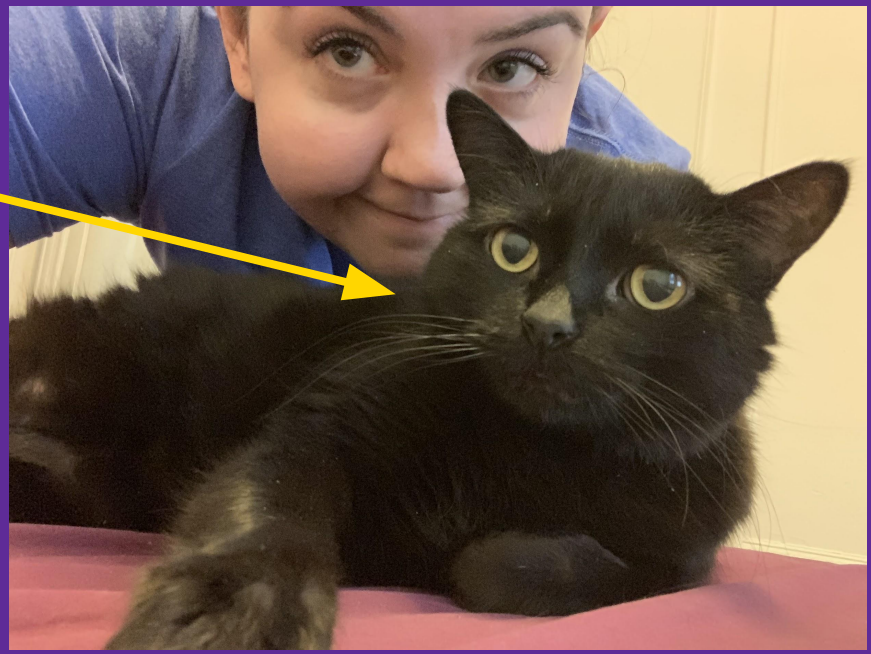
Introduction to Shiny

Kaelen L. Medeiros
@kaelen_medeiros on twitter
Slides (pdf) @
bit.ly/klm-rladiestun-introshiny-july2020

I'm Kaelen!!!!! (That's Scully)

- Lives in Astoria, Queens (NYC, USA)
- Recently became a data scientist on the microservices DS/DE team @ Medidata Solutions
- MS in Biostatistics
- Loves R, data, aliens, and podcasts



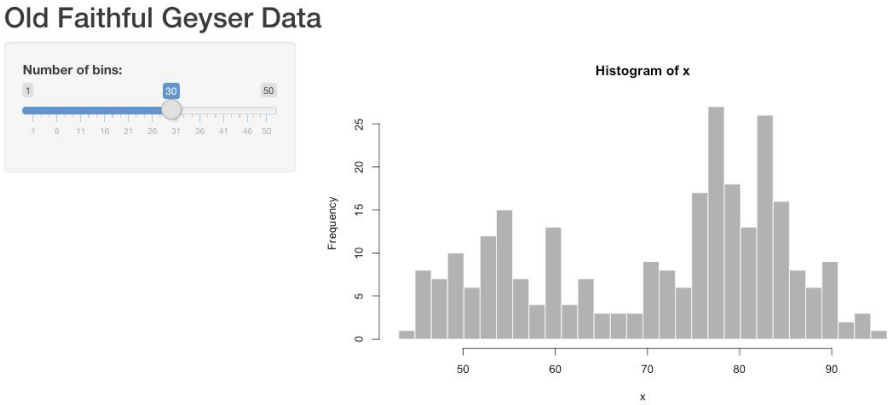
Outline

- Shiny????
- Hello, world! I’m a Shiny app
  - UI
  - Server
  - Basic Reactivity
- Our App
  - We’ll cover:
    - Inputs & Outputs
    - Layouts
    - Interactivity

Shiny???

What is Shiny?

- Shiny is “an R package that makes it easy to build interactive web apps straight from R,” says RStudio (who have so graciously brought Shiny to the world)
- Allows us to use R code to build these web apps without having to know HTML, CSS, or JavaScript (though some knowledge can help!)



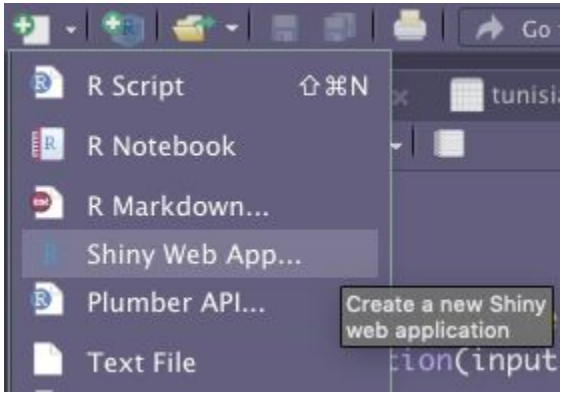
Speaking of that link to the Shiny website...

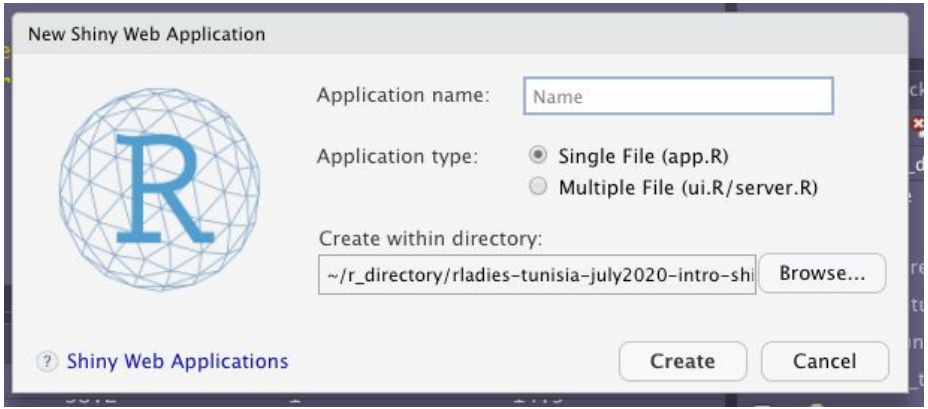
- It’s an amazing resource for getting started with Shiny OR extending your current knowledge!
- Highly recommend using the Gallery, which as both awesome showcase apps and also demos (with code!) to show how certain features work
- But also features articles, reference materials, and help

Hello, world! I'm a Shiny app

Let's dive in!

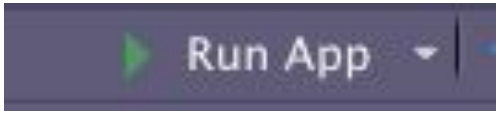
- To create a new app, go to the "New" button in RStudio and click "Shiny Web App"
- Then you can give your app a name and choose if you want to use a single file (called app.R) or separate files (one for the UI, one for the server)
- For now, select single file and call your app something meaningful to you
    - example_app, hello_world





Default app

- The same default app is always created when you make an app this way
- Let’s run it by hitting “Run App” or Cmd + Shift + Enter (Mac) or Window + Shift + Enter (Windows, probably)
    - “Run App” should be in the upper right of your app.R script pane
- Example app uses the Old Faithful geyser data and allows users to slide to select the number of bins in the histogram, between 1-50



Let's look at some of the code

- App has…
  - A UI

```
# 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)
        ),

        # Show a plot of the generated distribution
        mainPanel(
           plotOutput("distPlot")
        )
    )
)
```

Let's look at some of the code

- App has…
  - A UI
  - A server
  - A call to shinyApp() to actually run the application
- Those are the basic building blocks of an app!

```
# 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)
```

UI

- UI = User Interface
- Part of the app where you control the things the user is going to see & interact with
- Here's where you'll put the code to **display**…
  - Inputs
  - Outputs
  - Layout changes
    - Tabs
    - Colors
  - Anything that affects how the app will **LOOK**

Server

- Creates a new environment each time you start the app
- The server() function requires 3 arguments:
  - Input
  - Output
  - Session
- In the server, you'll use the inputs (as needed) to create the things you want to display
  - Plots
  - Tables
  - Maps
  - Value Boxes
  - And more!

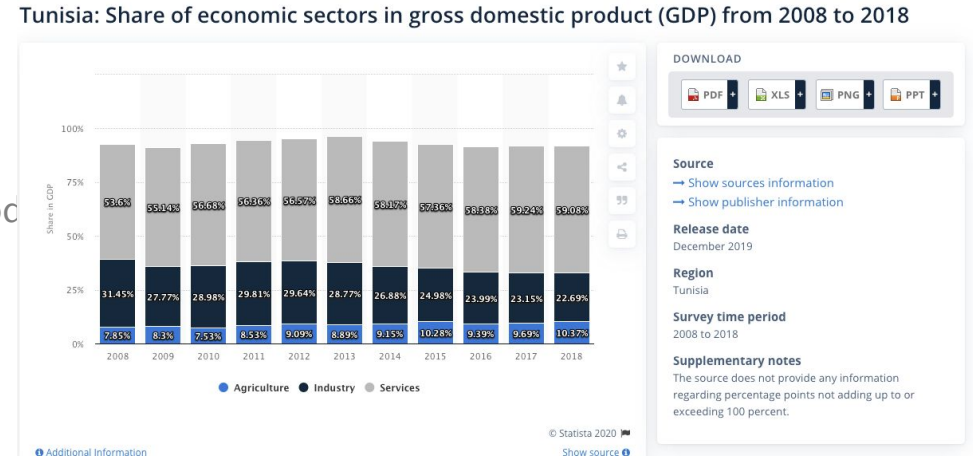
(The Most) Basic Reactivity

- Reactivity fundamentally means that when your inputs are updated, they automatically update any connected outputs
- **Reactive sources** are the inputs themselves
- **Reactive endpoints** are the things that are updated by the reactive sources (inputs) and will appear to your user (updated!)
- You can use these ideas to build reactive expressions, functions that are:
    - Only evaluated when called by a reactive endpoint (inside a render**() function!)
    - Evaluated ONLY when some reactive source changes (an input!)

Our app

The data

- Tunisia-themed, from Statista
- We'll recreate the graph -> in our app
- Columns:
  - Year
  - Growth rate of the real gross domestic proc (GDP)
  - Agriculture
  - Industry
  - Services
  - Population growth rate
  - Infant mortality rate (%)
  - Unemployment rate (%)
  - Inflation rate (%)
  - Fertility rate (%)



Inputs

- Inputs allow your user to input a value that will control the outputs in some way
- Shiny Widgets Gallery displays a bunch of possible inputs…
  - Checkboxes
  - Date/date range selectors
  - Numeric input
  - Select box
  - Sliders
  - Text input
- Created in your UI with a *Input() function
  - textInput(), numericInput(), etc.

Outputs

- Outputs are created in the server:

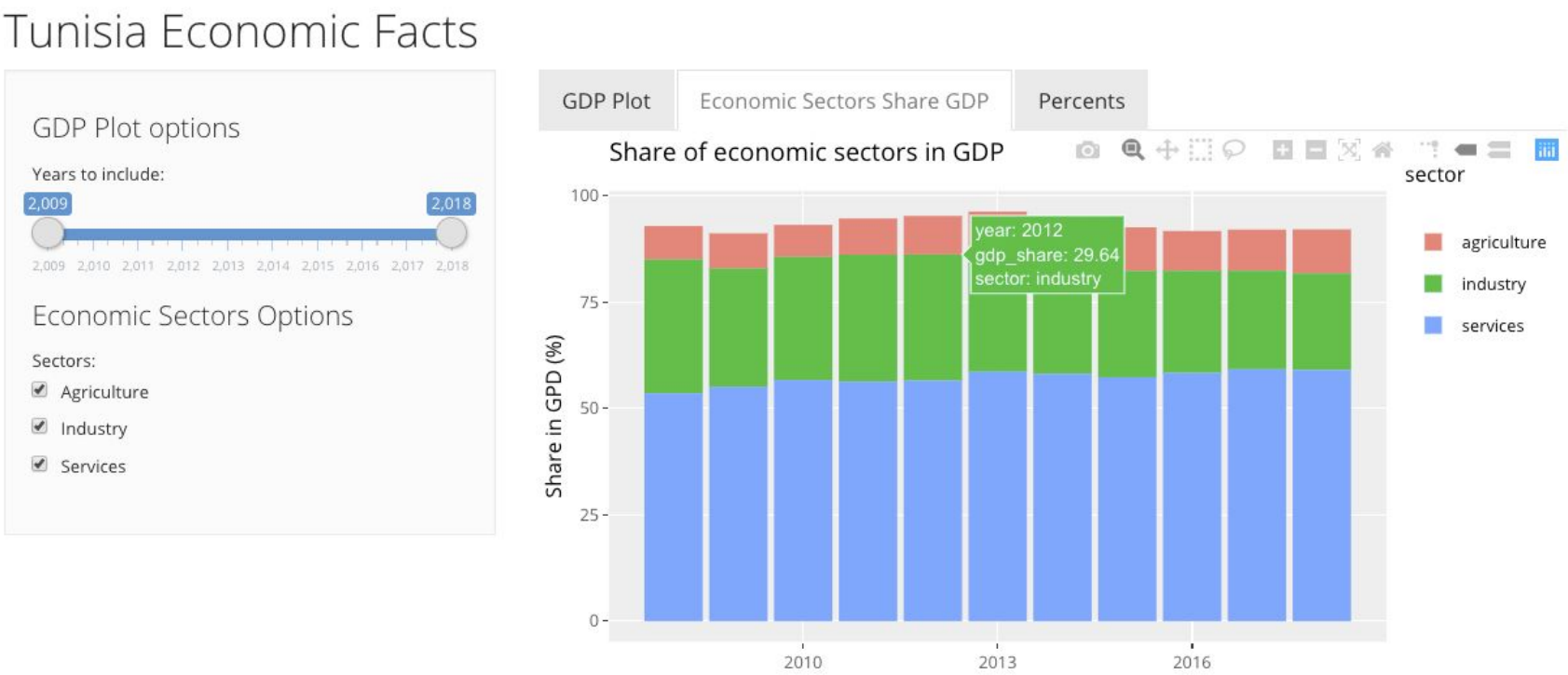
```
output$plot <- renderPlot({

     mtcars %>% ggplot(aes(mpg)) + geom_histogram()

})
```

- Create an output object, use the correct render function, and write R code inside to create what you want to display
- Then, back in the UI, they are displayed with the correct *Output() function:
  - plotOutput(), tableOutput(), textOutput(), etc.

Ultimately, we'll have an app that looks like...



Let's begin!

- We'll use the app.R file (or the ui.R AND server.R files) in the tunisia_app_static_1 folder on Github (if you just want the code) or you can code along with me
- A note on the arrangement of the Github repo:
  - I made each step of the app we're going to code a separate app so anyone not here can follow along
  - Each folder is labeled at the end with _#
  - Each is a copy of the previous folder (app) with at least one thing added on
- Packages you'll need installed:
  - shiny
  - tidyverse
  - readxl
  - shinythemes
  - DT
  - plotly

tunisia_app_static_1

- We’ll add:
  - App title
  - Input slider to select range of years
  - Static line graph of growth rate of the GDP in tunisia
    - Import the data into the app

tunisia_app_tabs_2

- We’ll add:
  - A tab layout
  - Checkbox group input
  - Re-creation of graph from the data source showing share of the economic sectors in the GDP

tunisia_app_table_3

- We’ll add:
  - Static table of the rest of the data (economic indicators percentages, e.g. fertility rate, unemployment rate, etc.)

tunisia_app_interactive_4

- We'll:
  - Convert all of our static outputs to interactive outputs
    - plotly for graphs
    - DT for datatables
- We won't load plotly or DT directly in the app, but will rather use explicit calls:
  - plotly::renderPlotly()
  - DT::datatable()
  - etc.

tunisia_app_fancy_5

- We'll:
  - Use shinythemes to browse different default themes and then
  - Select one to use for our app

Depending on time, reactive expressions

- What if we wanted the years slider to affect both graphs? It's probably time to write a reactive expression to filter the tunisia data only once!

```
tunisia_years <- reactive({

   tunisia %>% filter(year >= input$years[1], year <= input$years[2])

})
```

- It's prudent (and less expensive computationally) to only do this filtering once!

Resources

Resources

- RStudio Shiny website
- Mastering Shiny, a forthcoming book by Hadley Wickham he is publishing as he writes on the web
- Google, like me:
  - renderTable
  - shinythemes
  - DT options in a Shiny app (took me to StackOverflow!)

Questions???

Kaelen L. Medeiros

@kaelen_medeiros on twitter
kaelenmedeiros@gmail.com
klmedeiros.com

Code and slides @
bit.ly/klm-rladiestun-introshiny-july2020