



Week 12: *Interactivity*

 EMSE 4572 / 6572: Exploratory Data Analysis

 John Paul Helveston

 November 12, 2025

Quiz 5 - last one!

12:00

Write your name on the quiz!



Rules:

- Work alone; no outside help of any kind is allowed.
- No calculators, no notes, no books, no computers, no phones.

Week 12: *Interactivity*

1. Interactive charts with plotly
2. Interactive tables

Intermission

3. Shiny apps
4. Shiny extras

Week 12: *Interactivity*

1. Interactive charts with plotly

2. Interactive tables

Intermission

3. Shiny apps

4. Shiny extras

Plotly uses JavaScript to create interactive charts

But you don't have to know JavaScript to use it! 🎉

Turn any ggplot into an interactive chart with `ggplotly()`

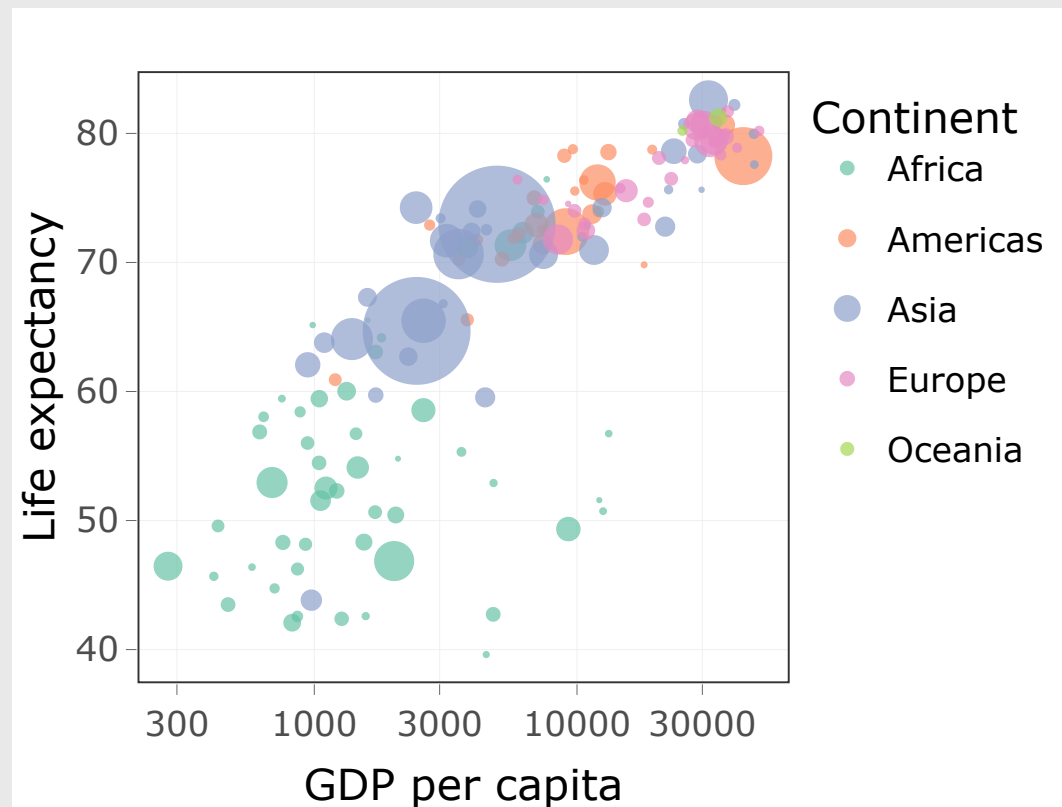
```
plot <- gapminder %>%  
  filter(year == 2007) %>%  
  ggplot(aes(x = gdpPercap, y = lifeExp,  
             size = pop, color = continent,  
             label = country)) +  
  geom_point(alpha = 0.7) +  
  scale_color_brewer(palette = 'Set2') +  
  scale_size_area(  
    guide = FALSE, max_size = 25) +  
  scale_x_log10() +  
  theme_bw(base_size = 16) +  
  labs(x = 'GDP per capita',  
       y = 'Life expectancy',  
       color = 'Continent')
```



plot

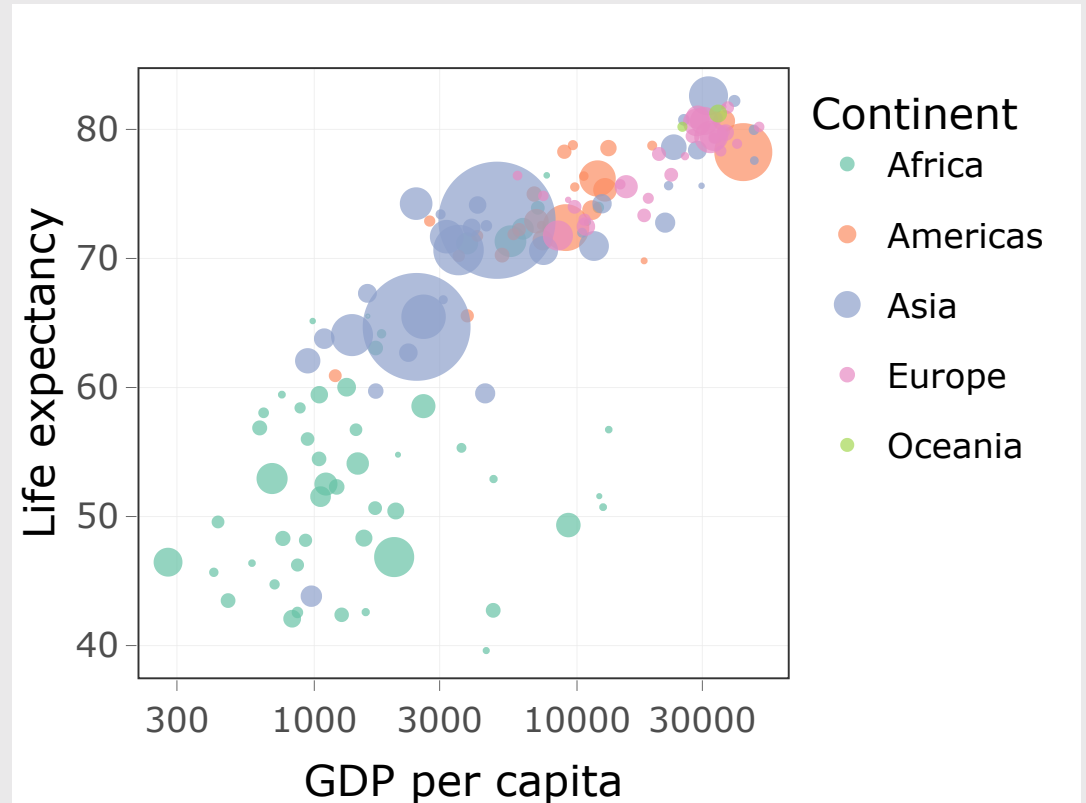
Turn any ggplot into an interactive chart with `ggplotly()`

```
ggplotly(plot)
```



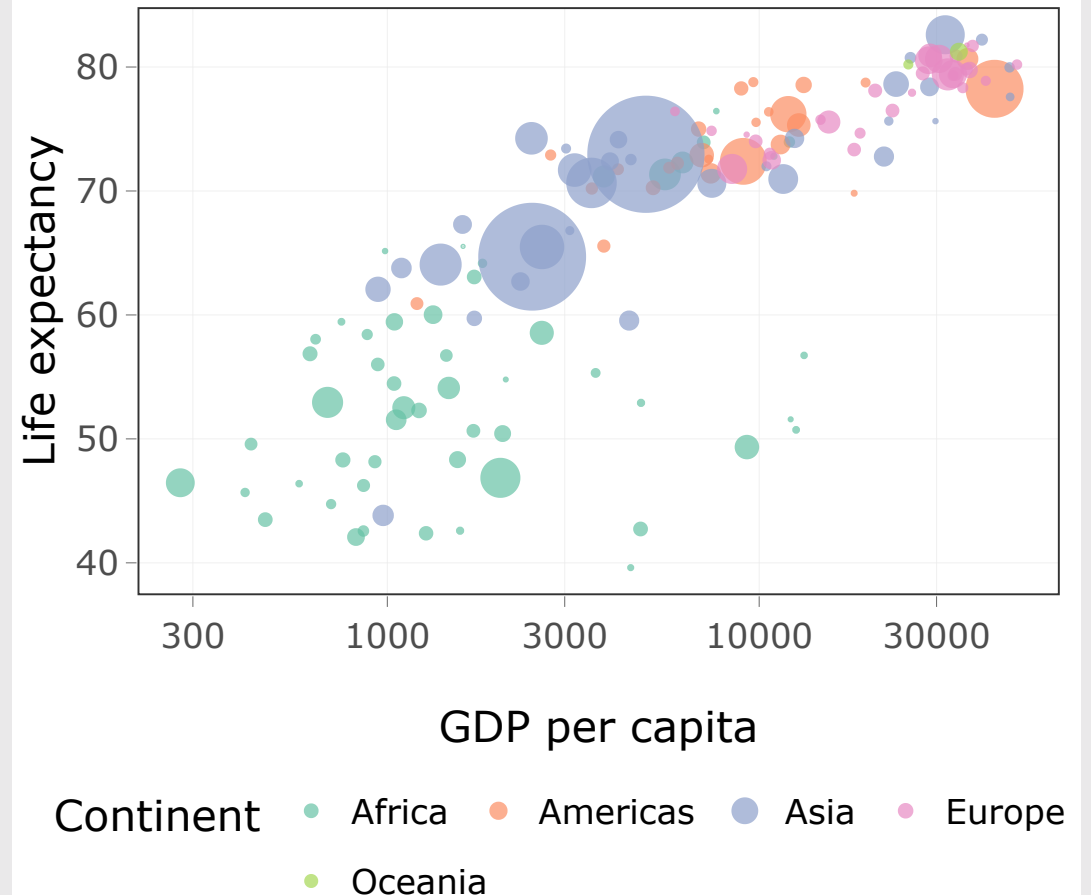
Modify the data shown with `tooltip` argument

```
ggplotly(  
  plot,  
  tooltip = c("country", "pop")  
)
```



Modify other features by piping on `plotly` functions

```
ggplotly(  
  plot,  
  tooltip = c("country", "pop")  
) %>%  
  layout(legend = list(  
    orientation = "h", x = 0, y = -0.3))
```

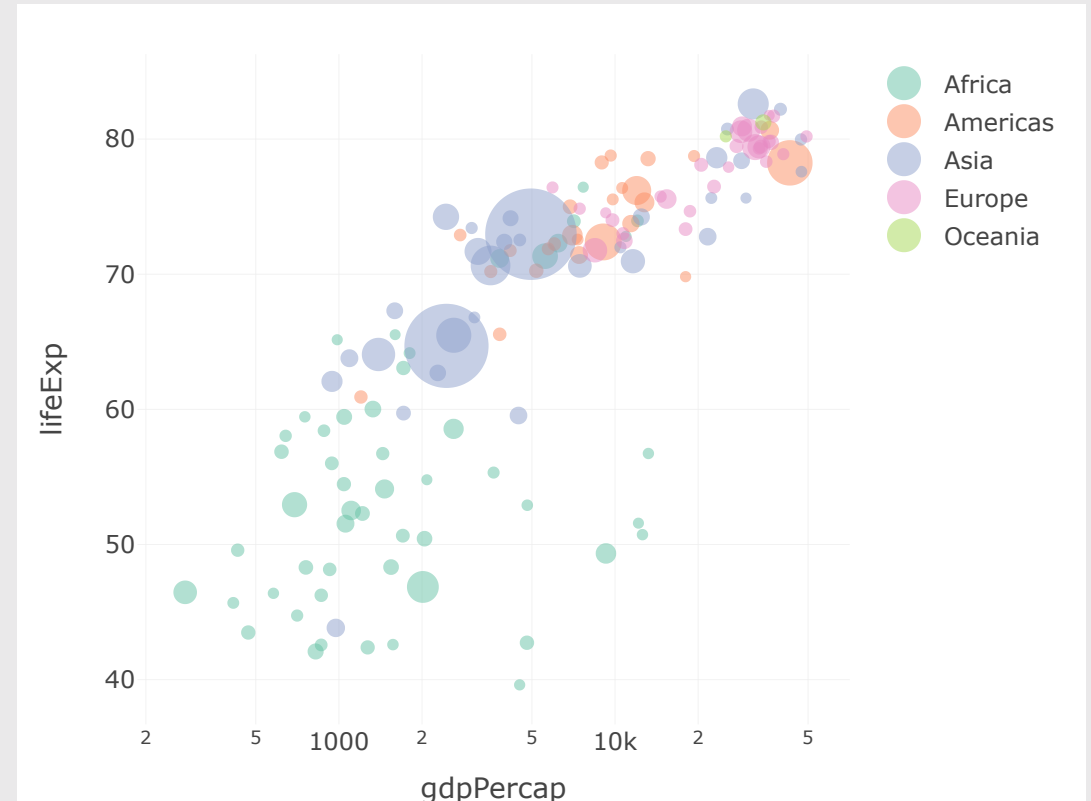


Reference guide: <https://plotly.com/ggplot2/>

Make interactive charts with `plot_ly()`

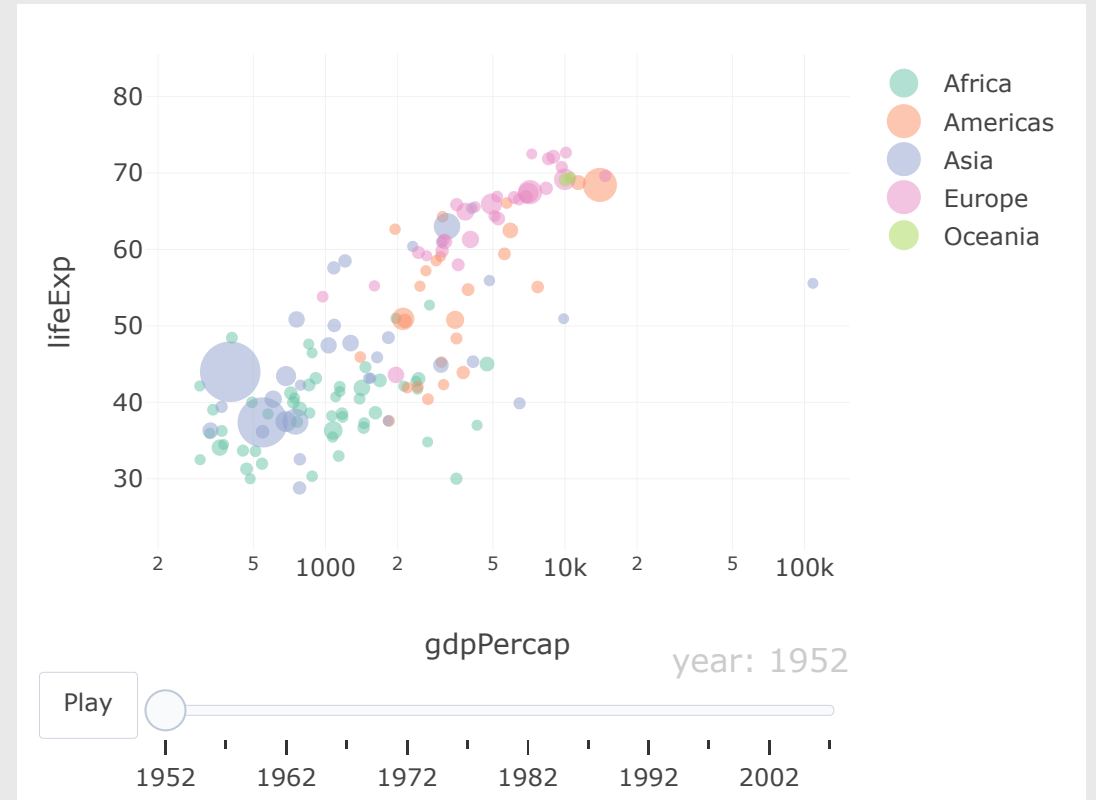
(More examples here: <https://plotly.com/r/>)

```
plot_ly(  
  data = gapminder %>% filter(year == 2007)  
  type = 'scatter',  
  x = ~gdpPerCap,  
  y = ~lifeExp,  
  size = ~pop,  
  color = ~continent,  
  text = ~country,  
  mode = "markers",  
  sizes = c(10, 1000),  
  marker = list(opacity = 0.5),  
  hoverinfo = "text"  
) %>%  
  layout(xaxis = list(type = "log"))
```



Animation is relatively easy with `plot_ly()`

```
plot_ly(  
  data = gapminder,  
  type = 'scatter',  
  x = ~gdpPercap,  
  y = ~lifeExp,  
  size = ~pop,  
  color = ~continent,  
  text = ~country,  
  frame = ~year,  
  mode = "markers",  
  sizes = c(10, 1000),  
  marker = list(opacity = 0.5),  
  hoverinfo = "text"  
) %>%  
  layout(xaxis = list(type = "log"))
```



Save as html page

```
htmlwidgets::saveWidget(  
  ggplotly(plot),  
  file = here::here('figs', 'gapminder.html')  
)
```

Insert using iframe

```
htmltools::tags$iframe(  
  src      = here::here('figs', 'gapminder.html'),  
  width    = "100%",  
  height   = "400",  
  scrolling = "no",  
  seamless = "seamless",  
  frameBorder = "0"  
)
```

Your Turn: Interactive Charts

10:00

1. Open your reflection from this past week (or a previous week)
2. With a classmate, take turns sharing your interactive chart, or go back to a chart we made in a previous class and make it interactive using either `ggplotly()` or `plot_ly()`
3. If you have an example you want to share, post your code in Slack

Week 12: *Interactivity*

1. Interactive charts with plotly
2. Interactive tables

Intermission

3. Shiny apps
4. Shiny extras

Make pretty static tables with `kable()`

```
library(knitr)
```

```
gapminder %>%
```

```
  kable()
```

country	continent	year	lifeExp	pop	gdpPercap
Afghanistan	Asia	1952	28.80100	8425333	779.4453
Afghanistan	Asia	1957	30.33200	9240934	820.8530
Afghanistan	Asia	1962	31.99700	10267083	853.1007
Afghanistan	Asia	1967	34.02000	11537966	836.1971
Afghanistan	Asia	1972	36.08800	13079460	739.9811
Afghanistan	Asia	1977	38.43800	14880372	786.1134
Afghanistan	Asia	1982	39.85400	12881816	978.0114
Afghanistan	Asia	1987	40.82200	13867957	852.3959
Afghanistan	Asia	1992	41.67400	16317921	649.3414
Afghanistan	Asia	1997	41.76300	22227415	635.3414
Afghanistan	Asia	2002	42.12900	25268405	726.7341

Behind the scenes:

`kable()` generates the code to make a pretty table

```
gapminder %>%  
  kable(format = "pipe")
```

```
|country |continent | year| lifeExp| pop| gdpPercap| |:-----|:-----  
-|-----:|-----:|-----:|-----:| |Afghanistan |Asia | 1952| 28.80100| 8425333|  
779.4453| |Afghanistan |Asia | 1957| 30.33200| 9240934| 820.8530| |Afghanistan  
|Asia | 1962| 31.99700| 10267083| 853.1007| |Afghanistan |Asia | 1967| 34.02000|  
11537966| 836.1971| |Afghanistan |Asia | 1972| 36.08800| 13079460| 739.9811|  
|Afghanistan |Asia | 1977| 38.43800| 14880372| 786.1134| |Afghanistan |Asia | 1982|  
39.85400| 12881816| 978.0114| |Afghanistan |Asia | 1987| 40.82200| 13867957|  
852.3959| |Afghanistan |Asia | 1992| 41.67400| 16317921| 649.3414| |Afghanistan  
|Asia | 1997| 41.76300| 22227415| 635.3414| |Afghanistan |Asia | 2002| 42.12900|  
25268405| 726.7341| |Afghanistan |Asia | 2007| 43.82800| 31889923| 974.5803|  
|Albania |Europe | 1952| 55.23000| 1282697| 1601.0561| |Albania |Europe | 1957|  
59.28000| 1476505| 1942.2842| |Albania |Europe | 1962| 64.82000| 1728137|
```

Behind the scenes:

`kable()` generates the code to make a pretty table

```
gapminder %>%  
  kable(format = "html")
```

```
#> <table>  
#>   <thead>  
#>     <tr>  
#>       <th style="text-align:left;"> country </th>  
#>       <th style="text-align:left;"> continent </th>  
#>       <th style="text-align:right;"> year </th>  
#>       <th style="text-align:right;"> lifeExp </th>  
#>       <th style="text-align:right;"> pop </th>  
#>       <th style="text-align:right;"> gdpPercap </th>  
#>     </tr>  
#>   </thead>  
#> <tbody>  
#>   <tr>  
#>     <td style="text-align:left;"> Afghanistan </td>  
#>     <td style="text-align:left;"> Asia </td>  
#>     <td style="text-align:right;"> 1952 </td>
```

Make *interactive* tables with:

`DT::datatable()`

Make *interactive* tables with `datatable()`

```
library(DT)
```

```
gapminder %>%
```

```
  datatable()
```

Show 10 ▾ entries

Search:

	country	continent	year	lifeExp	pop
1	Afghanistan	Asia	1952	28.801	8425333
2	Afghanistan	Asia	1957	30.332	9240934
3	Afghanistan	Asia	1962	31.997	10267083
4	Afghanistan	Asia	1967	34.02	11537966
5	Afghanistan	Asia	1972	36.088	13079460
6	Afghanistan	Asia	1977	38.438	14880372
7	Afghanistan	Asia	1982	39.854	12881816
8	Afghanistan	Asia	1987	40.822	13867957
9	Afghanistan	Asia	1992	41.674	16317921
10	Afghanistan	Asia	1997	41.763	22227415

Showing 1 to 10 of 1,704 entries

Previous

1

2

3

4

5

...

17

Make *interactive* tables with `datatable()`

```
gapminder %>%  
  datatable(  
    options = list(  
      pageLength = 5,  
      lengthMenu = c(5, 10, 15, 20)  
    )  
  )
```

Show entries Search:

	country	continent	year	lifeExp	pop
1	Afghanistan	Asia	1952	28.801	8425333
2	Afghanistan	Asia	1957	30.332	9240934
3	Afghanistan	Asia	1962	31.997	10267083
4	Afghanistan	Asia	1967	34.02	11537966
5	Afghanistan	Asia	1972	36.088	13079460

Showing 1 to 5 of 1,704 entries Previous 2 3 4 5 ... 34

Modify features by piping on functions

```
gapminder %>%  
  datatable() %>%  
  formatCurrency('gdpPercap') %>%  
  formatStyle(  
    'country',  
    color = 'red',  
    backgroundColor = 'black',  
    fontWeight = 'bold')
```

Show entries Search:

	country	continent	year	lifeExp	pop
1	Afghanistan	Asia	1952	28.801	8425333
2	Afghanistan	Asia	1957	30.332	9240934
3	Afghanistan	Asia	1962	31.997	10267083
4	Afghanistan	Asia	1967	34.02	11537966
5	Afghanistan	Asia	1972	36.088	13079460
6	Afghanistan	Asia	1977	38.438	14880372
7	Afghanistan	Asia	1982	39.854	12881816
8	Afghanistan	Asia	1987	40.822	13867957
9	Afghanistan	Asia	1992	41.674	16317921
10	Afghanistan	Asia	1997	41.763	22227415

Showing 1 to 10 of 1,704 entries Previous 1 2 3 4 5 ... 17

Modify features by piping on functions

```
gapminder %>%  
  datatable() %>%  
  formatCurrency('gdpPercap') %>%  
  formatStyle(  
    'country',  
    color = 'red',  
    backgroundColor = 'black',  
    fontWeight = 'bold') %>%  
  formatStyle(  
    'lifeExp',  
    background = styleColorBar(  
      gapminder$lifeExp, 'dodgerbl',  
      backgroundSize = '100% 90%',  
      backgroundRepeat = 'no-repeat',  
      backgroundPosition = 'center')
```

Show entries Search:

	country	continent	year	lifeExp	pop
1	Afghanistan	Asia	1952	28.801	8425333
2	Afghanistan	Asia	1957	30.332	9240934
3	Afghanistan	Asia	1962	31.997	10267083
4	Afghanistan	Asia	1967	34.02	11537966
5	Afghanistan	Asia	1972	36.088	13079460
6	Afghanistan	Asia	1977	38.438	14880372
7	Afghanistan	Asia	1982	39.854	12881816
8	Afghanistan	Asia	1987	40.822	13867957
9	Afghanistan	Asia	1992	41.674	16317921
10	Afghanistan	Asia	1997	41.763	22227415

Showing 1 to 10 of 1,704 entries Previous 1 2 3 4 5 ... 17

Make *interactive* tables with:

`reactable::reactable()`

Make *interactive* tables with `reactable()`

```
library(reactable)
```

```
gapminder %>%  
  reactable()
```

country	continent	year	lifeExp	pop	gdpPer					
Afghanistan	Asia	1952	28.801	8425333	779.445					
Afghanistan	Asia	1957	30.332	9240934	820.853					
Afghanistan	Asia	1962	31.997	10267083	853.10					
Afghanistan	Asia	1967	34.02	11537966	836.197					
Afghanistan	Asia	1972	36.088	13079460	739.981					
Afghanistan	Asia	1977	38.438	14880372	786.1					
Afghanistan	Asia	1982	39.854	12881816	978.011					
Afghanistan	Asia	1987	40.822	13867957	852.395					
Afghanistan	Asia	1992	41.674	16317921	649.341					
Afghanistan	Asia	1997	41.763	22227415	635.34					
1–10 of 1704 rows		Previous	1	2	3	4	5	...	171	Next

reactable() has some nice options!

```
library(reactable)
```

```
gapminder %>%
```

```
  reactable(
```

```
    searchable = TRUE,
```

```
    highlight = TRUE,
```

```
    filterable = TRUE,
```

```
    defaultPageSize = 5,
```

```
    showPageSizeOptions = TRUE,
```

```
    pageSizeOptions = c(5, 10, 15)
```

```
  )
```

Search

country	continent	year	lifeExp	pop	gdpPer
Afghanistan	Asia	1952	28.801	8425333	779.445
Afghanistan	Asia	1957	30.332	9240934	820.853
Afghanistan	Asia	1962	31.997	10267083	853.10
Afghanistan	Asia	1967	34.02	11537966	836.197
Afghanistan	Asia	1972	36.088	13079460	739.981

1–5 of 1704 rows Show 5 ▾

Previous 1 2 3 4 5 ... 341 Next

Add more features with `reactablefmtr` library

```
library(reactable)
library(reactablefmtr)

gapminder %>%
  reactable(
    searchable = TRUE,
    highlight = TRUE,
    filterable = TRUE,
    defaultPageSize = 5,
    showPageSizeOptions = TRUE,
    pageSizeOptions = c(5, 10, 15)
    columns = list(
      lifeExp = colDef(cell = data_
        gapminder,
        colors = c("#d7191c", "#ff
        align = "center")) ## align
    )
```

Search

country	continent	year	lifeExp	pop
Afghanistan	Asia	1952	28.801	8425333
Afghanistan	Asia	1957	30.332	9240934
Afghanistan	Asia	1962	31.997	10267083
Afghanistan	Asia	1967	34.02	11537966
Afghanistan	Asia	1972	36.088	13079460

1–5 of 1704 rows Show 5 ▾

Previous 1 2 3 4 5 ... 341 Next

Add more features with `sparkline` library (example)

```
library(reactable)
library(sparkline)

gapminder_summary <- gapminder %>%
  group_by(country) %>%
  summarise(lifeExp = list(lifeExp)) %>%
  mutate(leftExpTrend = NA)

gapminder_reactable_sparkline <- gapminder_
  reactable(
    searchable = TRUE,
    highlight = TRUE,
    filterable = TRUE,
    defaultPageSize = 5,
    showPageSizeOptions = TRUE,
    columns = list(
      lifeExp = colDef(
        cell = function(values) {
          sparkline(
            values, type = "bar", chartRang
            chartRangeMax = max(gapminder$l
          )},
      leftExpTrend = colDef(
        cell = function(value, index) {
          sparkline(gapminder_summary$lifeE
        })
      )
    )
  )
```

country	lifeExp	leftExpTrend
Afghanistan		
Albania		
Algeria		
Angola		
Argentina		

1–5 of 142 rows Show ▼

Previous **1** 2 3 4 5 ... 29 Next

References:

- <https://rstudio.github.io/DT/>
- <https://glin.github.io/reactable/>
- <https://kcuilla.github.io/reactablefmtr/>

Your Turn: Interactive Tables

20:00

Use `reactable()` to make the following interactive table

Read [this example](#) on how to embed images in table cells, then use the `gapminder_flags` data frame to make the interactive table.

Search

flag	country	continent	year	lifeExp	
	afghanistan	Asia	2007	43.828	31889
	albania	Europe	2007	76.423	3600
	algeria	Africa	2007	72.301	33333
	angola	Africa	2007	42.731	12420
	argentina	Americas	2007	75.32	40301

1–5 of 142 rows Show 5 ▾

Previous 1 2 3 4 5 ... 29 Next

Intermission

05 : 00

Week 12: *Interactivity*

1. Interactive charts with plotly
2. Interactive tables

Intermission

3. **Shiny apps**
4. Shiny extras

License

These slides were modified from [Florencia D'Andrea's RLadies Shiny Meetup Slides](#)

This work is licensed under a
[Creative Commons Attribution-ShareAlike 4.0](#) International License

New libraries to install

```
install.packages('shiny')  
install.packages('shinyWidgets')  
install.packages('rsconnect')
```



Interactive Webapps in R

Check out the [Shiny Gallery](#)

Anatomy of a Shiny App

A Shiny app is a web page (UI) connected to a computer running a live R session (Server)



app.R

You can insert all the code at once with the shinyapp snippet!

Just start typing **shiny**...



```
library(shiny)
ui <- fluidPage(
)
server <- function(input, output, session) {
}
shinyApp(ui, server)
```

Building a shiny app



ui

1. Pick a layout function
2. Add inputs widgets
3. Add `*Output()` functions

server

1. Use `render*()` functions to make outputs
2. Link outputs with `output$<id>`
3. Link inputs with `input$<id>`

Run the app

- **Option 1:** Click the "Run App" button in the toolbar:



- **Option 2:** Use a keyboard shortcut: Cmd/Ctrl + Shift + Enter.
- **Option 3:** `shiny::runApp()` with the path to the **app.R** file.

Your Turn

hello_shiny.app

File → New File → Shiny Web App...

```
library(shiny)

# 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")
    )
  )
)

# 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)
```

User Interface (UI)



Matryoshka Dolls

Organize panels and elements into a layout with a **layout function**

Top level is usually `fluidPage()`





sidebarLayout()

```
ui <- fluidPage(  
  titlePanel("Hello Shiny!"),  
  sidebarLayout(  
    sidebarPanel(  
      sliderInput(  
        "bins", label = "Number of  
        min = 1, value = 30, max =  
      )  
    ),  
    mainPanel(  
      plotOutput("distPlot")  
    )  
  )  
)
```



navbarPage() : An alternative to fluidPage()

Think of each `tabPanel()` as it's own `fluidPage()`

```
ui <- navbarPage("My Application",  
  tabPanel("Component 1"),  
  tabPanel("Component 2"),  
  tabPanel("Component 3")  
)
```



navbarPage() : An alternative to fluidPage()

Use `navbarMenu()` to create a nested menu item

```
ui <- navbarPage("My Application",  
  tabPanel("Component 1"),  
  tabPanel("Component 2"),  
  navbarMenu("More",  
    tabPanel("Sub-Component A"),  
    tabPanel("Sub-Component B"))  
)
```



The UI defines the "what" and "where" for:

1. **Inputs**: collect values from the user
2. **Output**: display something to the user

Inputs: collect values from the user



Example Input: Radio buttons in the sidebar

```
ui <- fluidPage(  
  sidebarLayout(  
    sidebarPanel(  
      radioButtons(...)  
    ),  
    mainPanel(...)  
  )  
)
```







Quick practice

1. Open the `widgets.R` file.
2. Run the app (Click "run app" button).
3. Go to the [Shiny Widgets Gallery](#) to see other input widgets.

The UI defines the "what" and "where" for:

1. **Inputs**: collect values from the user
2. **Output**: display something to the user

Example Output: Put a plot in the main panel

```
ui <- fluidPage(  
  sidebarLayout(  
    sidebarPanel(  
      radioButtons(...)  
    ),  
    mainPanel(  
      plotOutput(...)  
    )  
  )  
)
```



Output: display something to the user

Output function	Description
<code>plotOutput()</code>	Display a reactive <i>plot</i>
<code>dataTableOutput()</code>	Display a <code>DT::datatable()</code>
<code>textOutput()</code>	Display reactive <i>text</i>
<code>imageOutput()</code>	Display an image

Building a shiny app



ui

1. Pick a layout function, e.g. `sidebarLayout()`
2. Add inputs widgets
3. Add `*Output()` functions

server

1. Use `render*()` functions to make outputs
2. Link outputs with `output$<id>`
3. Link inputs with `input$<id>`



Using `renderPlot()`: make a plot

```
library(ggplot2)

ggplot(mpg) +
  geom_point(
    aes(x = displ, y = hwy, color = class))
```



Link plot to output with `output$<id>`

ui

```
ui <- fluidPage(  
  sidebarLayout(  
    sidebarPanel(  
      radioButtons(...)  
    ),  
    mainPanel(  
      plotOutput(  
        outputId = "mpg_plot"  
      )  
    )  
  )  
)
```

server

```
server <- function(input, output, session) {  
  output$mpg_plot <- renderPlot({  
    ggplot(mpg) +  
      geom_point(  
        aes(x = displ, y = hwy, color = class))  
      })  
}
```

Link user inputs to plot with `input$<id>`

ui

```
ui <- fluidPage(  
  sidebarLayout(  
    sidebarPanel(  
      radioButtons(  
        inputId = "xvar",  
        label = "Select the x-axis variable:",  
        selected = "displ",  
        choices = c(  
          "Highway miles per gallon" = "hwy",  
          "City miles per gallon" = "cty",  
          "Engine displacement, in litres" = "displ")  
        ),  
    ),  
    mainPanel(  
      plotOutput(  
        outputId = "mpg_plot"  
      )  
    )  
  )  
)
```

server

```
server <- function(input, output, session) {  
  output$mpg_plot <- renderPlot({  
    ggplot(mpg) +  
    geom_point(  
      aes_string(  
        x = input$xvar,  
        y = "hwy",  
        color = "class")  
      )  
    })  
}
```

Note: I switched the ggplot code from `aes()` to `aes_string()`

Quick practice

Open the `mpg.R` app and click the "Run App" button

Your Turn



1. Open the `caseConverter.R` file.
2. In the `server`: Write code in the provided `renderText()` to convert the input text to lower case.
3. Run the app and test that it's working.
4. In the `ui` main panel: Add two more `textOutput()` functions for also displaying the input text in "upper" case and "title" case.
5. In the `server`: Define two more outputs to convert the input text to "upper" case and "title" case.

Week 12: *Interactivity*

1. Interactive charts with plotly
2. Interactive tables

Intermission

3. Shiny apps
4. **Shiny extras**

Add a theme with "shinythemes" package



Insert theme at top of main `ui` layout function

```
library(shinythemes)

ui <- fluidPage(
  theme = shinytheme("sandstone"),
  sidebarLayout(
    sidebarPanel(
      <insert widgets>
    ),
    mainPanel(
      <insert outputs>
    )
  )
)
```



Fancier widgets with "shinyWidgets" package

Open the `shinyWidgets.R` app and click the "Run App" button

If you really want to get good at this:

1. Print out this [Cheatsheet](#)
2. Watch this [2.5 Hour Comprehensive RStudio Tutorial](#)
3. Use this reference manual: [Mastering Shiny](#)

You can deploy an app for free on shinyapps.io

Follow [this guide](#)

1. Create a shinyapps.io account
2. Open your tokens, click "Show", copy the code
3. Run the code in RStudio
4. Deploy your app:

```
library(rconnect)  
deployApp()
```

Your Turn

15:00

1. Open the `internetUsers.R` file.
2. Modify the server code so that the inputs control the plot.
3. Deploy your app to shinyapps.io