Interactive applications

Overview

Interactive web applications using Shiny

Note about today's class

If you get lost, you can download the simple app we are going to create on the class GitHub site:

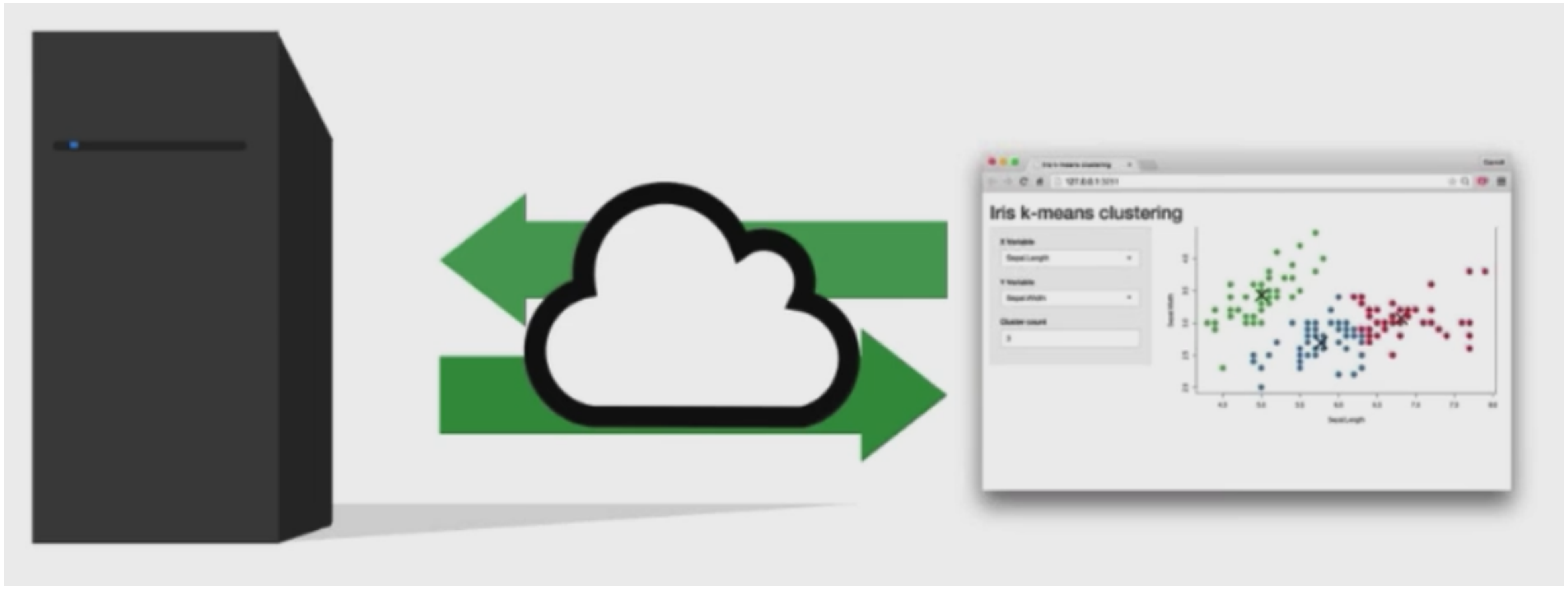
https://github.com/emeyers/SDS230/blob/master/ClassMaterial/class_code/simple_app.R

```
download.file(paste0(SDS230:::get_base_url(), "class_code/simple_app.R"),
"simple_app.R")
```

Shiny applications

Server runs R code, creates results

Client uses a web-based GUI to interact with code



Tutorial: https://shiny.rstudio.com/tutorial/

Fisher’s Iris data set

Setosa

Virginica

Vericolor

```
> View(iris)     # Look at the original iris data frame
```

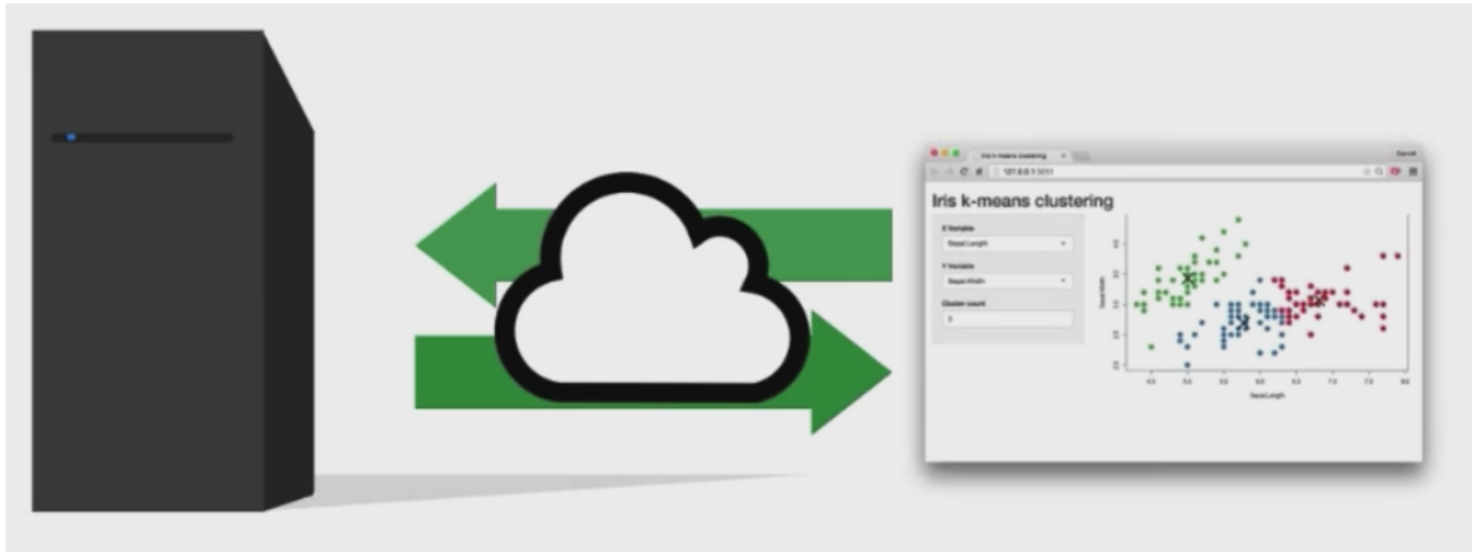
K-means Shiny app on Iris data set

Fisher, 1936

Shiny applications

Server runs R code, creates results

Client uses a web-based GUI to interact with code



You need to write 2 pieces of code to create a Shiny app:

- server: for the code that is run on the server
- ui: for the web interface shown to the user

Shiny application template

```
# include the shiny package
library(shiny)

# the function to create the user interface
ui <- fluidPage()

# the function to create the server
server <- function(input, output) {}

# putting them together to run
shinyApp(ui = ui, server = server)
```

UI code converted to HTML for web browsers

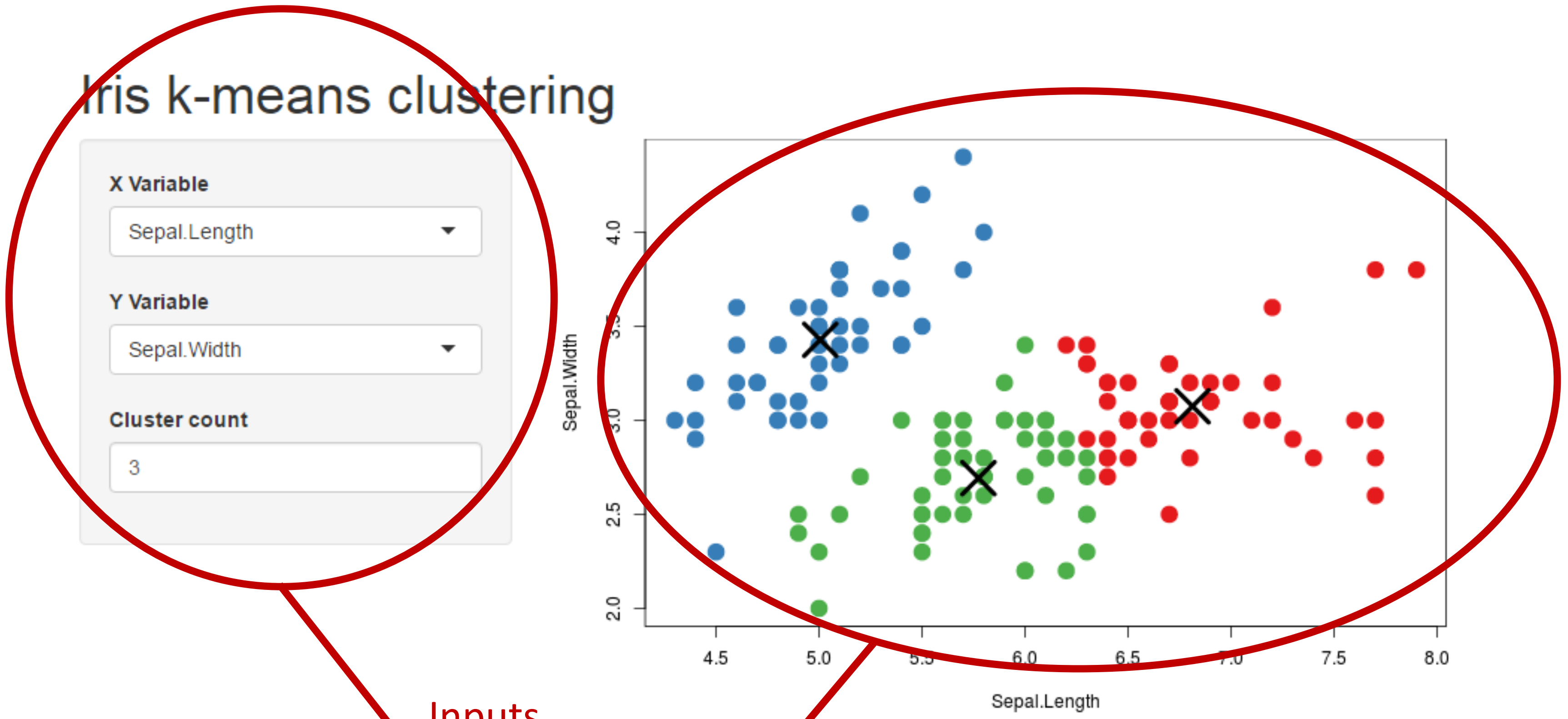
Start by creating a .R script that has this code
Follow along by continually testing the code…

Shiny application template

```
# include the shiny package
library(shiny)

# the function to create the user interface
ui <- fluidPage("Hello world!")

# the function to create the server
server <- function(input, output) {}

# putting them together to run
shinyApp(ui = ui, server = server)
```

Change the UI

Think in terms of inputs and outputs


Iris k-means clustering

X Variable: Sepal.Length

Y Variable: Sepal.Width

Cluster count: 3

Inputs

Outputs

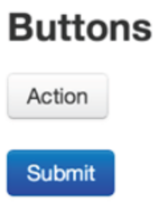
```
ui <- fluidPage(
     # Input() functions
     # Output() functions
)
```

Building a UI

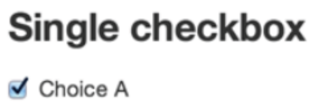
```
# adding a slider…
ui <- fluidPage(

        sliderInput(inputId = "num",
            label = "Choose a number",
            val = 25, min = 1, max = 100)

)   # notice the closing parenthesis!
```

Indentation is important to stay organized!

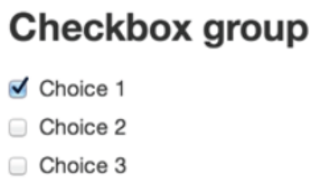
Input functions:



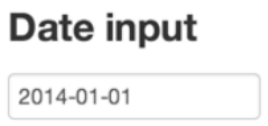
actionButton()
submitButton()



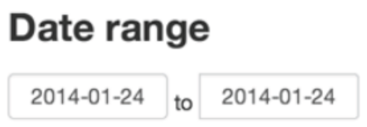
checkboxInput()



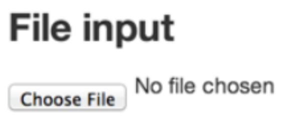
checkboxGroupInput()



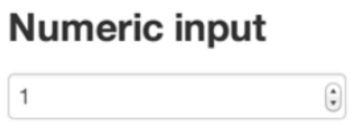
dateInput()



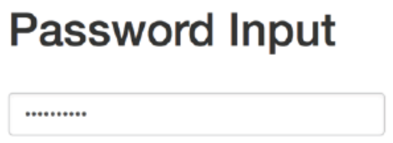
dateRangeInput()



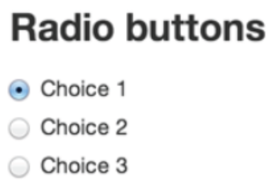
fileInput()



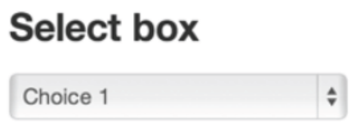
numericInput()



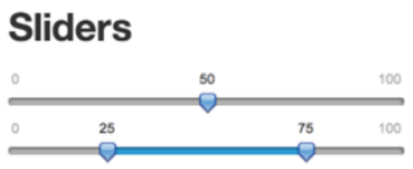
passwordInput()



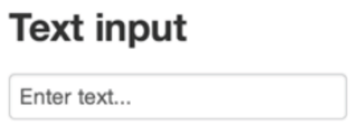
radioButtons()



selectInput()



sliderInput()



textInput()

See Cheat sheet!

Input functions!

Input functions all have a similar form:

```
sliderInput(inputId = "num",                ← Name to refer to returned value
            label = "Choose a number",      ← Label for user to see
            val = 25, min = 1, max = 100)   ← Input type specific arguments
```

Use help page to learn about input arguments:

```
> ? sliderInput
```

Output functions

Function	Inserts
`dataTableOutput()`	an interactive table
`htmlOutput()`	raw HTML
`imageOutput()`	image
`plotOutput()`	plot
`tableOutput()`	table
`textOutput()`	text
`uiOutput()`	a Shiny UI element
`verbatimTextOutput()`	text

Always need to give outputs a name

Example: `plotOutput(outputId = "my_plot")`

Building a UI

```
ui <- fluidPage(
        sliderInput(inputId = "num",
            label = "Choose a number",
            val = 25, min = 1, max = 100),
        plotOutput("my_plot")

)
```

Don’t forget the comma!

Sever function

Sever function connects inputs to outputs

```
server <- function(input, output) {

        output$my_plot <-   # code

}
```

Connecting the ui and the server

```
ui <- fluidPage(
          sliderInput(inputId = "num",
             label = "Choose a number",
             val = 25, min = 1, max = 100),
          plotOutput("my_plot")
)

server <- function(input, output) {
          output$my_plot <-   # code
}
```

Connecting the ui
and the sever

Sever function

Sever function connects inputs to outputs

```
server <- function(input, output) {

        output$my_plot <-   renderPlot({

                # add your plot here!
                # e.g., hist(rnorm(100))   # boring

        })

}
```

Connecting the ui and the server

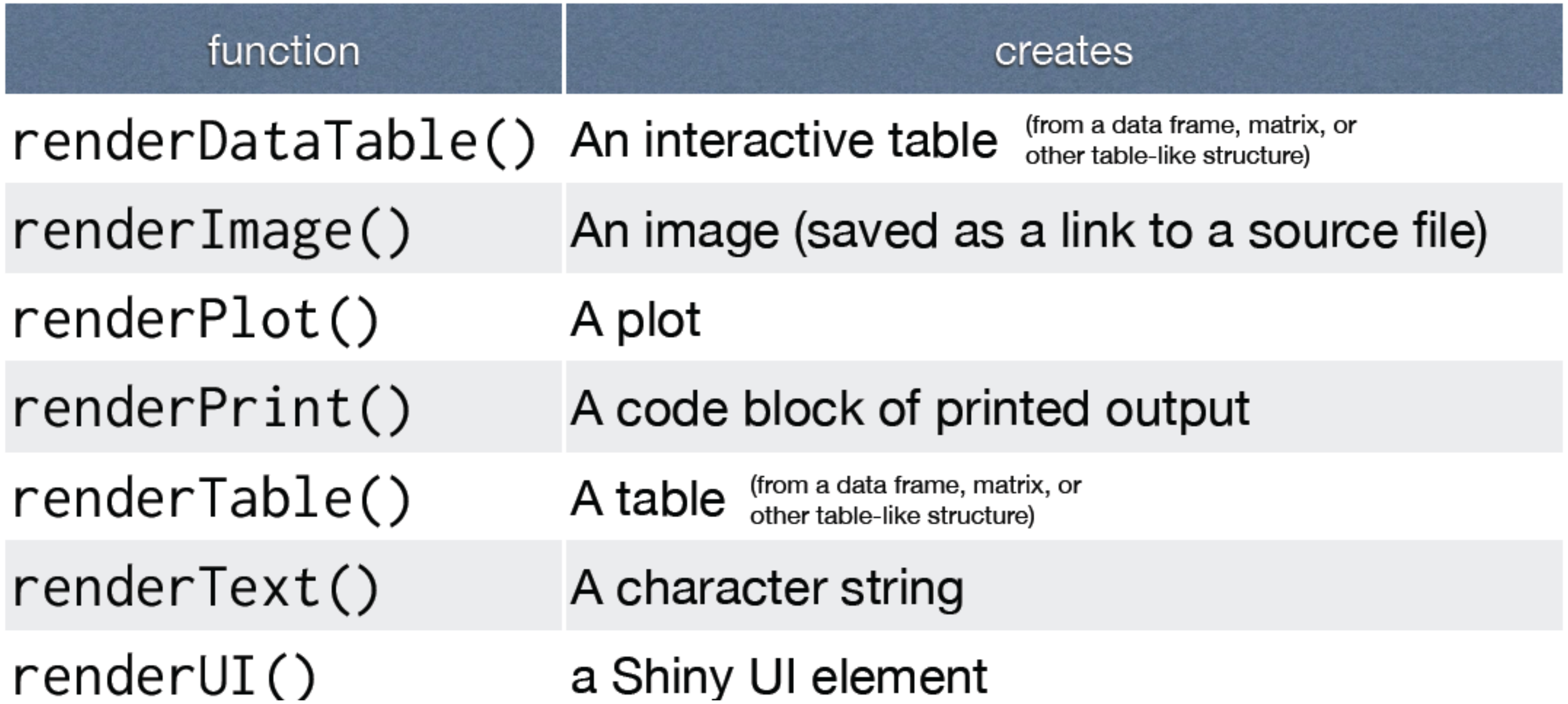
```
ui <- fluidPage(
        sliderInput(inputId = "num",
          label = "Choose a number",
          val = 25, min = 1, max = 100),
        plotOutput("my_plot")
)

server <- function(input, output) {
        output$my_plot <-   renderPlot({

        })
}
```

Usually a pairing of
xOutput and renderX

See Shiny Cheat Sheet for more pairs

Render functions

Render functions take R output and place it in an HTML page in the UI



function	creates
`renderDataTable()`	An interactive table (from a data frame, matrix, or other table-like structure)
`renderImage()`	An image (saved as a link to a source file)
`renderPlot()`	A plot
`renderPrint()`	A code block of printed output
`renderTable()`	A table (from a data frame, matrix, or other table-like structure)
`renderText()`	A character string
`renderUI()`	a Shiny UI element

Connecting the ui and the server

```
ui <- fluidPage(
          sliderInput(inputId = "num",
             label = "Choose a number",
             val = 25, min = 1, max = 100),
          plotOutput("my_plot")
)

server <- function(input, output) {
          output$my_plot <-   renderPlot({
                    hist(rnorm(input$num))
          })
}
```

Connect UI input to render output

Connecting the ui and the server

```
ui <- fluidPage(
        sliderInput(inputId = "num",
           label = "Choose a number",
           val = 25, min = 1, max = 100),
        plotOutput("my_plot")
)

server <- function(input, output) {
        output$my_plot <-   renderPlot({
                hist(rnorm(input$num))
        })
}
```

i.e., it is ‘reactive’.
The plot is redrawn every time slider value changes

This function is called every time the input$num value changes!

Sharing Shiny apps

Create a directory

Either:

- Save your file as app.R with both sever and ui functions
- Save as two files: server.R and ui.R
- Embed code in an R Markdown file

If you host this directory on a Shiny server you can access this over the web

- Can host on asterius or with RStudio (shinyapps.io)

Layout managers

Layout managers allow you to better position items on the web page (i.e., better ui)

```
ui <- fluidPage(

    sidebarLayout(

        sidebarPanel(  # add controls here),
        mainPanel(   # add plots here )

    ) # end sidebarLayout

)   # end ui
```

Don’t forget the comma!

Layout managers

```
ui <- fluidPage(

     sidebarLayout(
          sidebarPanel(
                    sliderInput(inputId = "num",
                         label = "Choose a number",
                         val = 25, min = 1, max = 100)
          ),

          mainPanel(
                    plotOutput("my_plot")
          )

   ) # end sidebarLayout
)   # end ui
```

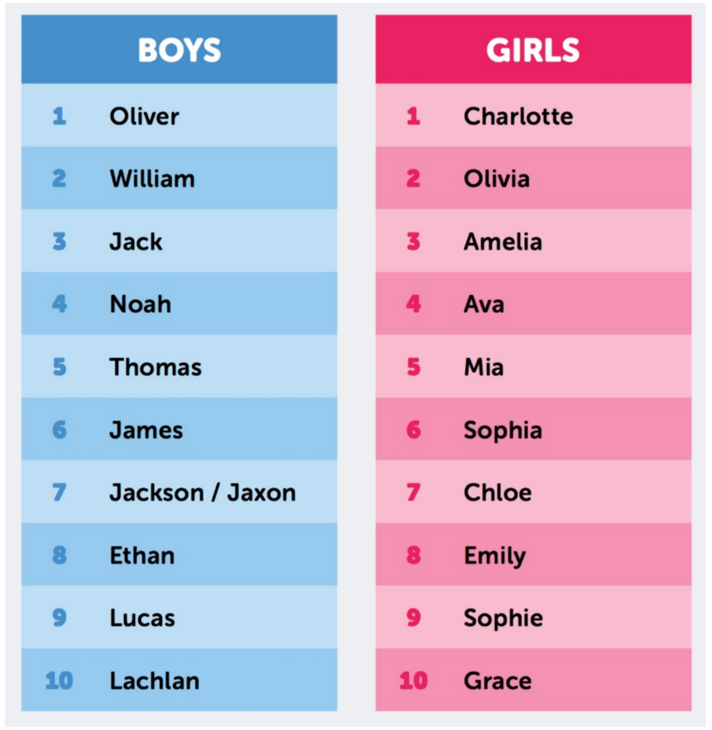
Application title

```
ui <- fluidPage(

    titlePanel("My cool Shiny app!"),

     sidebarLayout(
          sidebarPanel(
                    sliderInput(inputId = "num",
                         label = "Choose a number",
                         val = 25, min = 1, max = 100)
          ),
          mainPanel(
                    plotOutput("my_plot")
          )

    ) # end sidebarLayout
)   # end ui
```

You can add a title to your app

Practice by creating this app!

```
> library('babynames')

> View(babynames)
```



BOYS		GIRLS	
1	Oliver	1	Charlotte
2	William	2	Olivia
3	Jack	3	Amelia
4	Noah	4	Ava
5	Thomas	5	Mia
6	James	6	Sophia
7	Jackson / Jaxon	7	Chloe
8	Ethan	8	Emily
9	Lucas	9	Sophie
10	Lachlan	10	Grace

See if you can create this app:

https://asterius.hampshire.edu:3939/DataScience/babynames/