DATA SOCIETY®

Introduction to Rshiny - Part 1

“One should look for what is and not what he thinks should be.”
-Albert Einstein.

Warm up

Today, we will start talking about RShiny. Before we start, check out this blog to learn about why it is a useful tool: ***https://support.rstudio.com/hc/en-us/articles/218294727-Why-would-I-use-Shiny-instead-of-Tableau-Spotfire-Qlikview-or-similar-BI-tools-***

Welcome back!

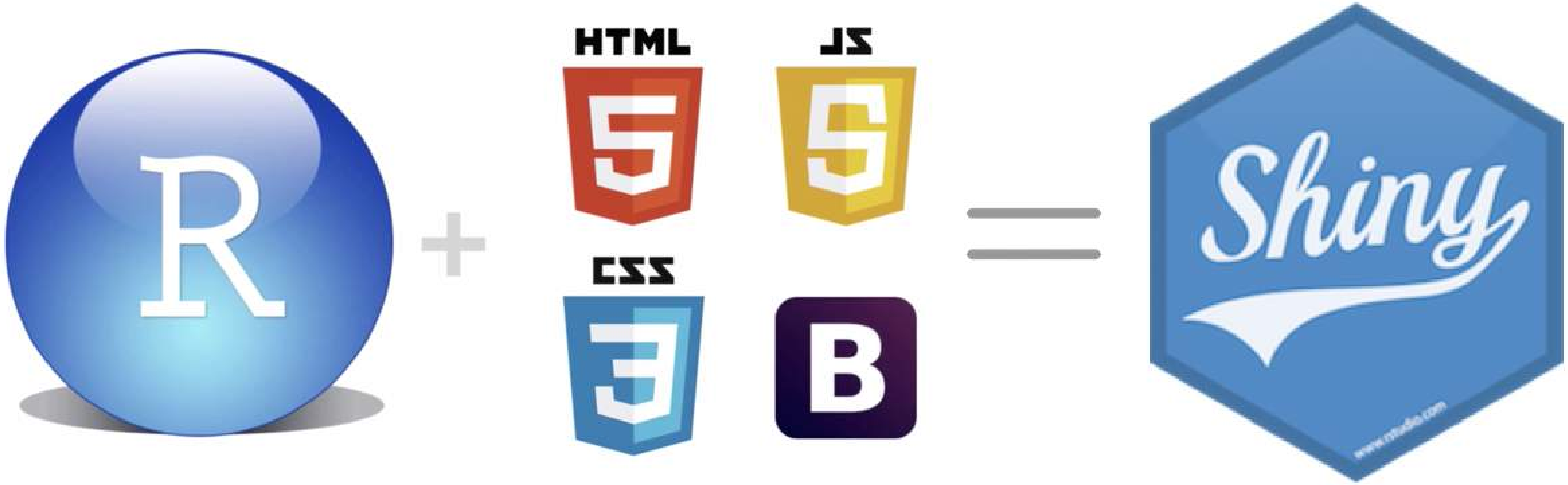
- In the last module we we learned how to create networks using `visNetwork`
- Today, we will learn about the components of RShiny applications and build a base application

Module completion checklist

Objective	Complete
Identify RShiny tools and discuss how they improve user experience	
Set up the layout to implement a simple dashboard	
Describe various output formats and their functionalities	
Integrate output formats into the simple dashboard	
Describe various control widgets and their functionalities	

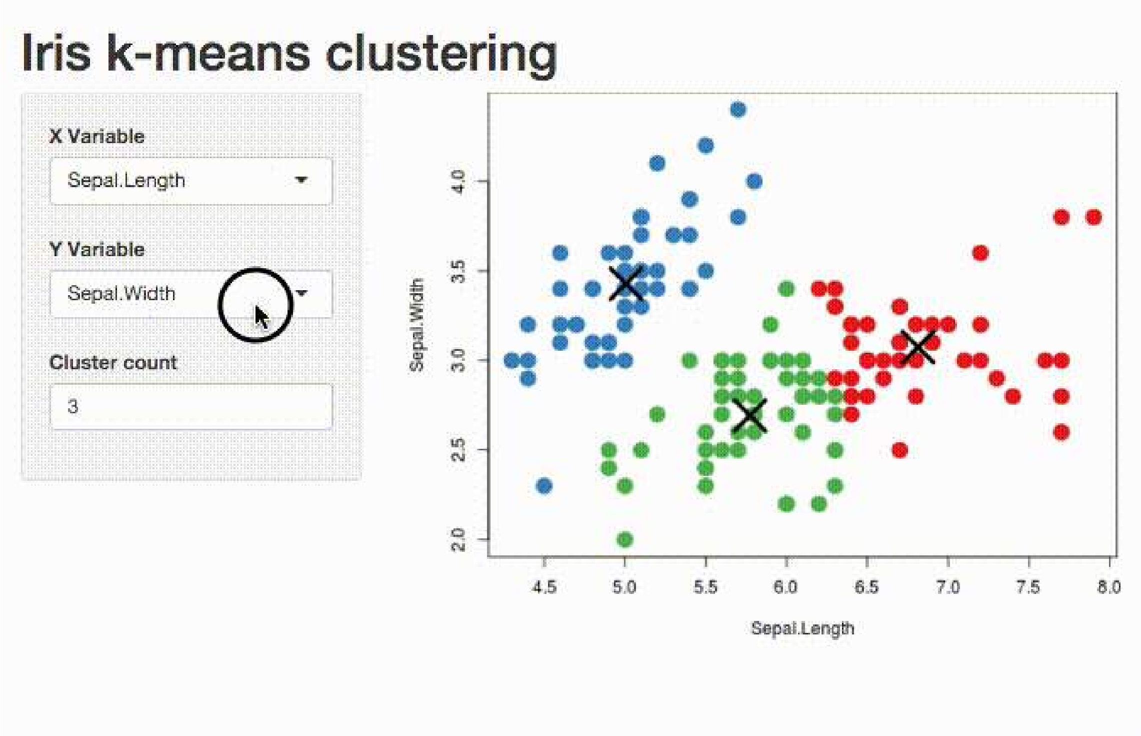
RShiny

- Shiny is an R package used to build interactive web apps which can be:
  - standalone apps on a webpage
  - embedded in R Markdown documents
  - packaged as dashboards
- It combines analytical abilities of R with display abilities of web design software



RShiny: example

- Shiny has an impressive user showcase with examples: ***https://www.rstudio.com/products/shiny/shiny-user-showcase/***



Iris k-means clustering

X Variable: Sepal.Length

Y Variable: Sepal.Width

Cluster count: 3

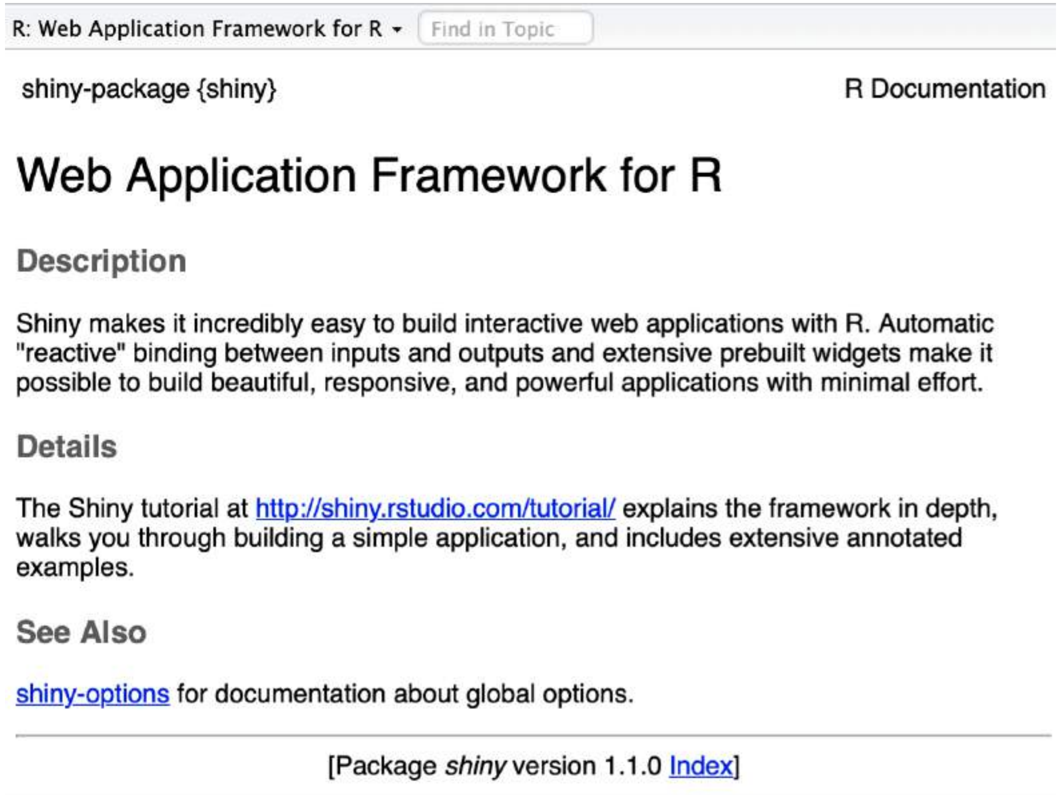
RShiny use cases

RShiny can help with:

- **data exploration**
  - e.g., create graphs that your users can explore interactively
- **user analysis**
  - e.g., create a template that lets the users do their own analysis of the data
- **communicating results**
  - e.g., create a dashboard that neatly showcases your work and insights

The RShiny package

```
library(shiny)

help("shiny-package")
```



shiny-package {shiny} R Documentation

Web Application Framework for R

Description

Shiny makes it incredibly easy to build interactive web applications with R. Automatic "reactive" binding between inputs and outputs and extensive prebuilt widgets make it possible to build beautiful, responsive, and powerful applications with minimal effort.

Details

The Shiny tutorial at http://shiny.rstudio.com/tutorial/ explains the framework in depth, walks you through building a simple application, and includes extensive annotated examples.

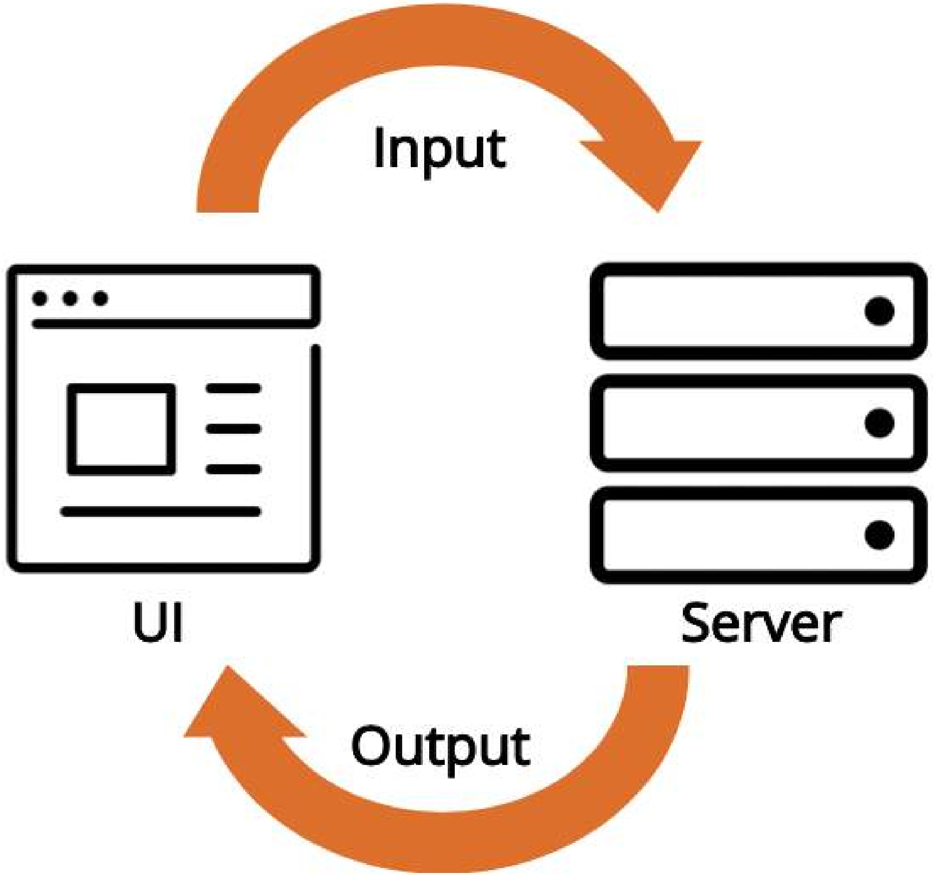
See Also

shiny-options for documentation about global options.

[Package *shiny* version 1.1.0 Index]

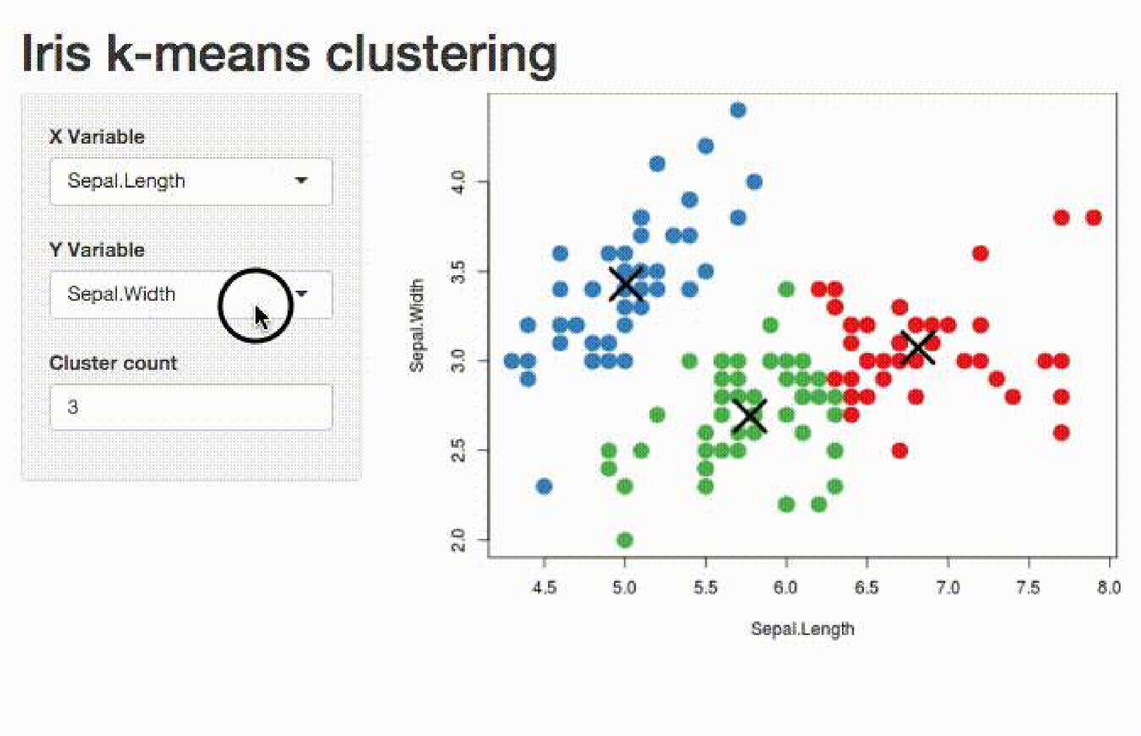
UI and Server

- Shiny has an automatic reactive binding between inputs and outputs for responsive and powerful applications
- A Shiny app usually contains two parts:
    - **UI**: controls the layout and appearance of the app
    - **Server**: contains the logic needed to build the app



UI example

- **UI:** controls the layout and appearance of the app



Iris k-means clustering

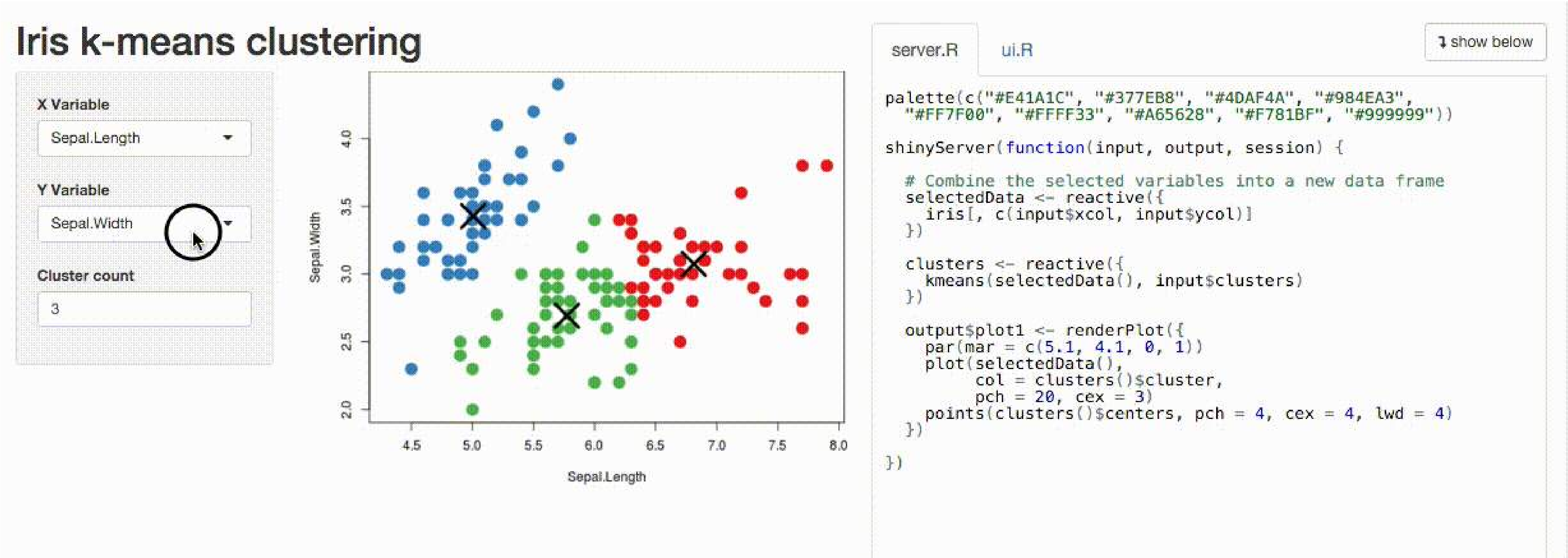
X Variable: Sepal.Length

Y Variable: Sepal.Width

Cluster count: 3

Server example

- **Server:** contains the logic needed to build the app



Iris k-means clustering

X Variable: Sepal.Length

Y Variable: Sepal.Width

Cluster count: 3

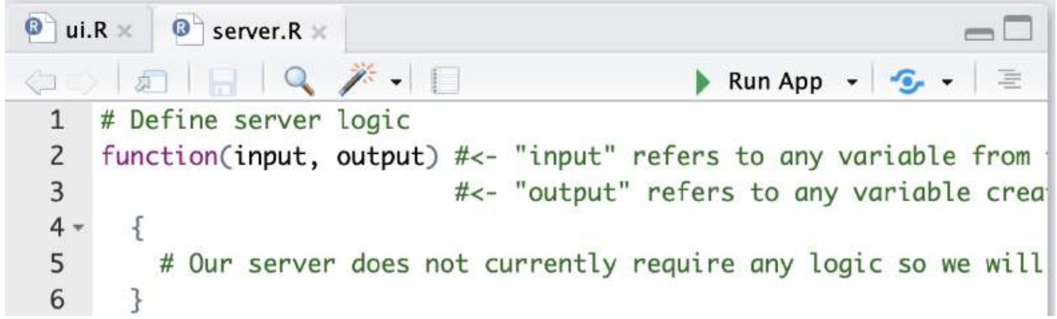
```
palette(c("#E41A1C", "#377EB8", "#4DAF4A", "#984EA3",
  "#FF7F00", "#FFFF33", "#A65628", "#F781BF", "#999999"))

shinyServer(function(input, output, session) {

  # Combine the selected variables into a new data frame
  selectedData <- reactive({
    iris[, c(input$xcol, input$ycol)]
  })

  clusters <- reactive({
    kmeans(selectedData(), input$clusters)
  })

  output$plot1 <- renderPlot({
    par(mar = c(5.1, 4.1, 0, 1))
    plot(selectedData(),
         col = clusters()$cluster,
         pch = 20, cex = 3)
    points(clusters()$centers, pch = 4, cex = 4, lwd = 4)
  })

})
```

Separate files for UI code and server code

- It's a coding best practice to write UI code and server code in **two separate** files
- However, these separate files need to be in the same folder in order for the RShiny application to work
- Moreover, the files need to be called `ui.R` and `server.R`
- This means that for every RShiny application that we create, we will have a **separate folder**

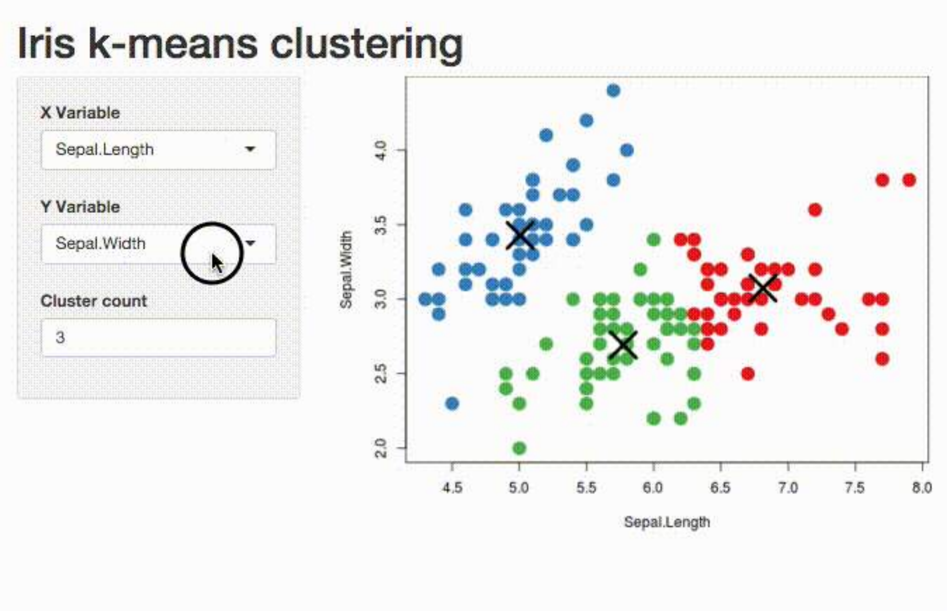
Launching your first RShiny application

- **Step 1:** Navigate to the `introduction-to-Rshiny-code-part-1/1-example` folder
- **Step 2:** Open `ui.R` or `server.R` file
- **Step 3:** Click 'Run App'



```
# Define server logic
function(input, output) #<- "input" refers to any variable from
                        #<- "output" refers to any variable crea
  {
    # Our server does not currently require any logic so we will
  }
```

- What you will see:



Iris k-means clustering

X Variable: Sepal.Length

Y Variable: Sepal.Width

Cluster count: 3

RShiny: integration with JavaScript-based widgets

- We can integrate the JavaScript-based interactive plots, maps, and widgets that we learned in the previous modules into an RShiny web app using `htmlwidgets`
- A Shiny dashboard example, which showcases Hungarian Interbank Lending and was created using `visNetwork` and `ggplot2`, can be found here: ***https://www.showmeshiny.com/hungarian-interbank-lending/***
- We will now learn how to build and customize our own Shiny applications!

Customizing your application

- When building an application, there are four elements that you can customize:
  - **Outputs:** create different output elements, such as plots or tables
  - **Inputs:** use various input widgets the user can utilize in the application
  - **Reactivity:** define how outputs get updated based on the inputs the user chooses
  - **Layout:** decide what your app should look like; use a default one or create your own
- We will first build a base application and then incrementally add each of the above elements

Module completion checklist

Objective	Complete
Identify RShiny tools and discuss how they improve user experience	✔
Set up the layout to implement a simple dashboard	
Describe various output formats and their functionalities	
Integrate output formats into the simple dashboard	
Describe various control widgets and their functionalities	

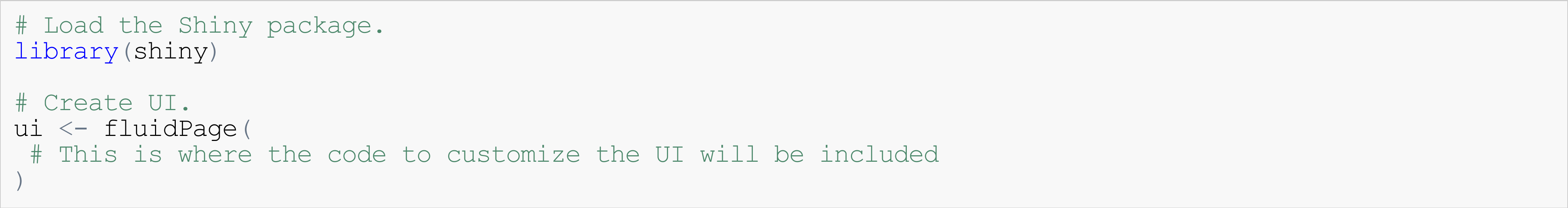
Directory settings

- First, let's make sure to set our directories correctly

```
# Set `main_dir` to the location of your `skillsoft` folder (for Mac/Linux).
main_dir = "~/Desktop/skillsoft"

# Set `main_dir` to the location of your `skillsoft` folder (for Windows).
main_dir = "C:/Users/[username]/Desktop/skillsoft"

# Make `data_dir` from the `main_dir` and
# remainder of the path to data directory.
data_dir = paste0(main_dir, "/data")
```

Set up a base app: UI

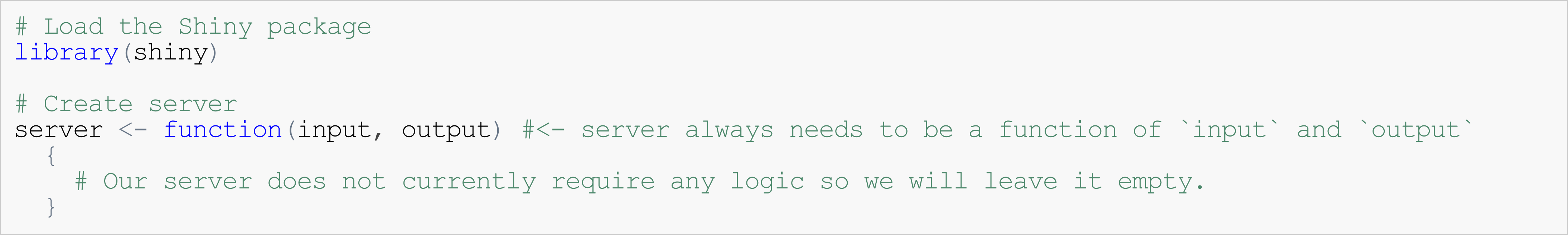
- The shinyUI function creates the UI
- It usually follows the given structure



```
# Load the Shiny package.
library(shiny)

# Create UI.
ui <- fluidPage(
 # This is where the code to customize the UI will be included
)
```

Set up a base app: server.R

- Since our base application does not need any logic to create the output, our server will be empty
- `input` refers to any variable received as user input from the UI
- `output` refers to any variable to be displayed as output to the UI

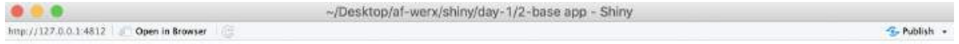


```r
# Load the Shiny package
library(shiny)

# Create server
server <- function(input, output) #<- server always needs to be a function of `input` and `output`
  {
    # Our server does not currently require any logic so we will leave it empty.
  }
```

Set up a base app

- **Step 1:** Navigate to `introduction-to-Rshiny-code-part-1/2-base-app`
- **Step 2:** Open `ui.R` or `server.R` file
- **Step 3:** Click 'Run App'

- What you will see:



- Not very impressive, but we will add on to this structure soon!

Module completion checklist

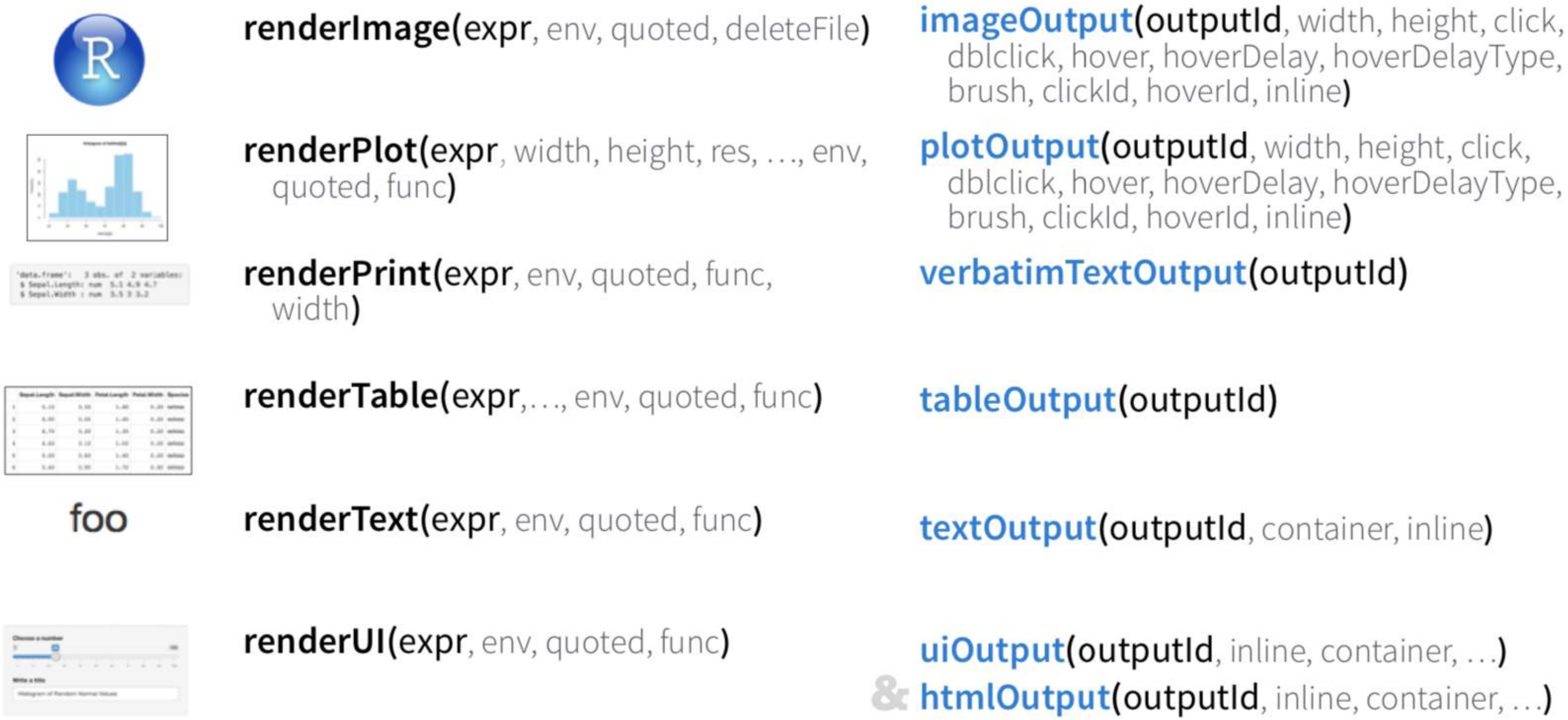
Objective	Complete
Identify RShiny tools and discuss how they improve user experience	✔
Set up the layout to implement a simple dashboard	✔
Describe various output formats and their functionalities	
Integrate output formats into the simple dashboard	
Describe various control widgets and their functionalities	

Customizing your application

- When building an application, there are four elements that you can customize:
  - **Outputs:** create different output elements, such as plots or tables
  - **Inputs:** use various input widgets the user can utilize in the application
  - **Reactivity:** define how outputs get updated based on the inputs the user chooses
  - **Layout:** decide what your app should look like; use a default one or create your own

Outputs

- Shiny can display various output formats
- This ***cheat sheet*** is a useful resource to see what output formats are available and to see the code for each format



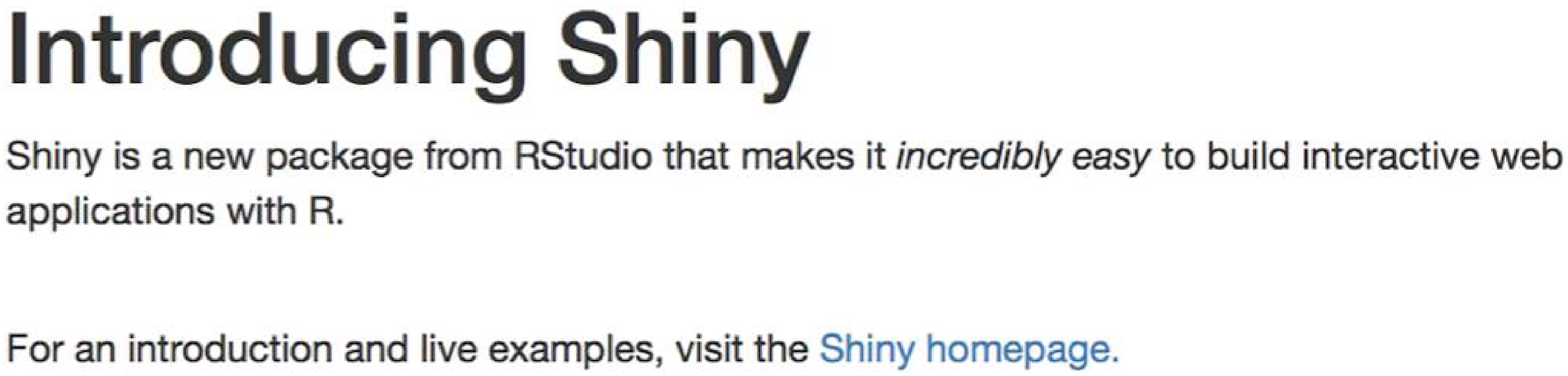
renderImage(expr, env, quoted, deleteFile**)**	**imageOutput(outputId**, width, height, click, dblclick, hover, hoverDelay, hoverDelayType, brush, clickId, hoverId, inline**)**
renderPlot(expr, width, height, res, …, env, quoted, func**)**	**plotOutput(outputId**, width, height, click, dblclick, hover, hoverDelay, hoverDelayType, brush, clickId, hoverId, inline**)**
renderPrint(expr, env, quoted, func, width**)**	**verbatimTextOutput(outputId)**
renderTable(expr,…, env, quoted, func**)**	**tableOutput(outputId)**
renderText(expr, env, quoted, func**)**	**textOutput(outputId**, container, inline**)**
renderUI(expr, env, quoted, func**)**	**uiOutput(outputId**, inline, container, …**)** & **htmlOutput(outputId**, inline, container, …**)**

Outputs: formats we'll explore

Output	Functions used
Text	textOutput, renderText
Table	tableOutput, renderTable
Plot	plotOutput, renderPlot
Print	verbatimTextOutput, renderPrint

Outputs: text

- **What it looks like**



Introducing Shiny

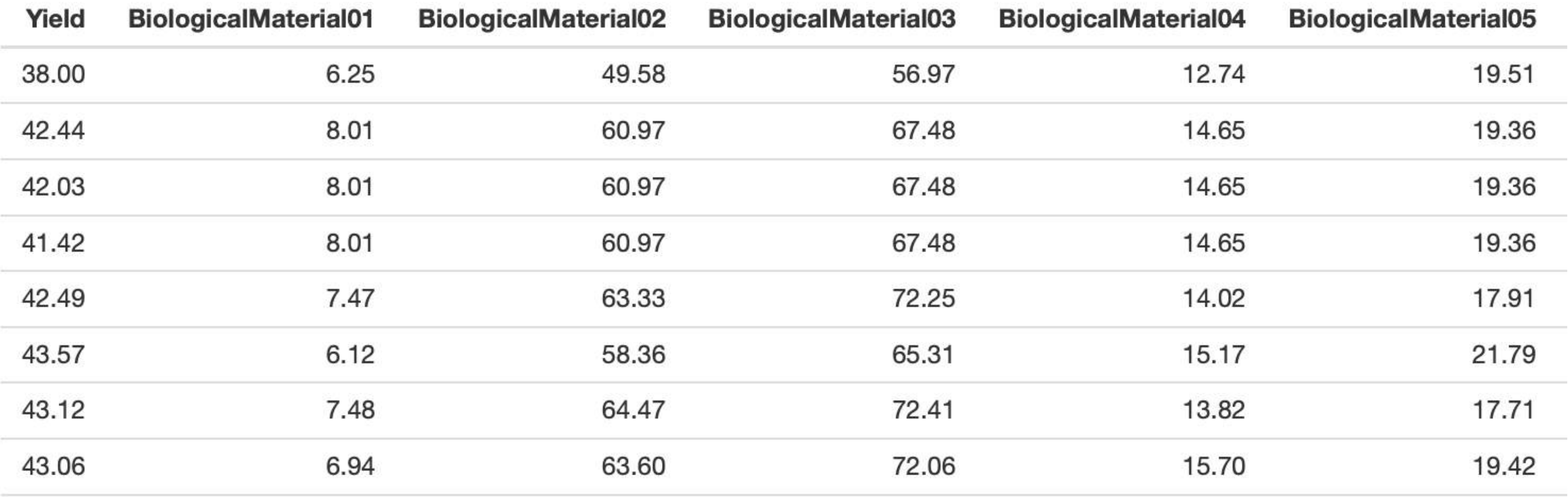
Shiny is a new package from RStudio that makes it *incredibly easy* to build interactive web applications with R.

For an introduction and live examples, visit the Shiny homepage.

- **What it does**
  - Displays written text
- **When it is used**
  - To introduce your application
  - To explain your analysis
  - To summarize your results

Outputs: table

- **What it looks like**

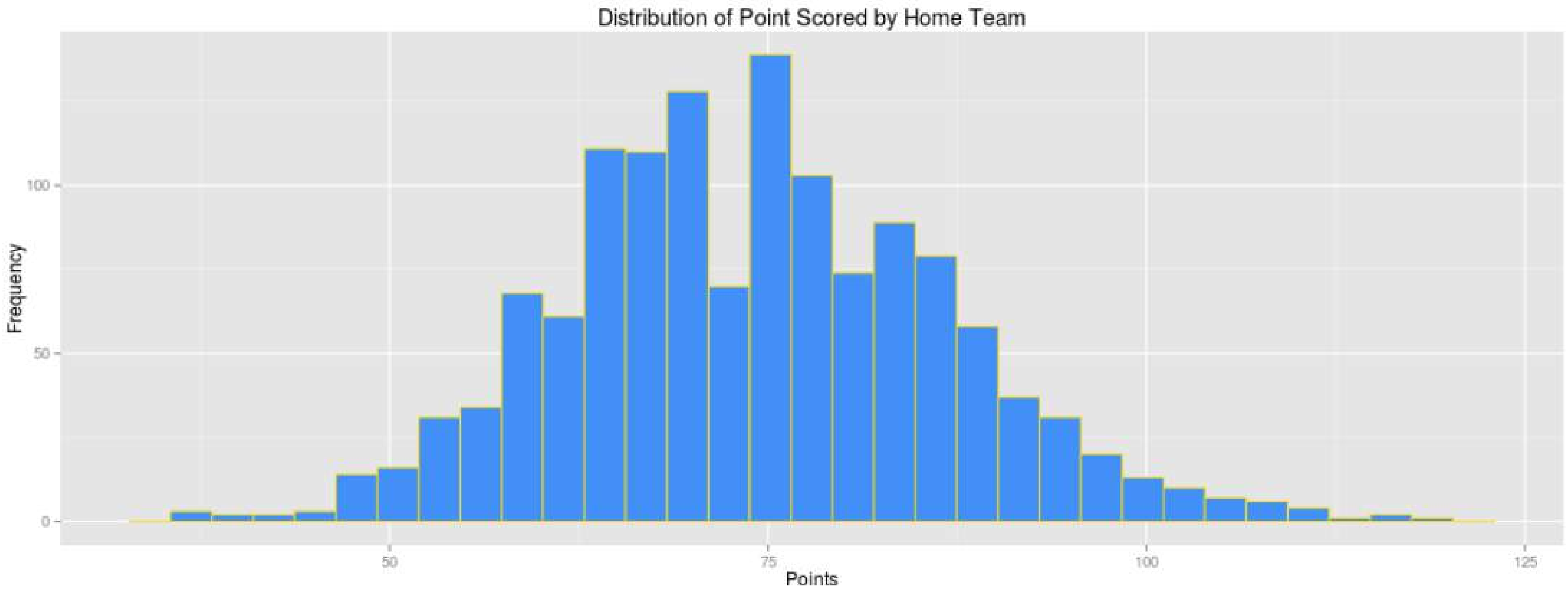


Yield	BiologicalMaterial01	BiologicalMaterial02	BiologicalMaterial03	BiologicalMaterial04	BiologicalMaterial05
38.00	6.25	49.58	56.97	12.74	19.51
42.44	8.01	60.97	67.48	14.65	19.36
42.03	8.01	60.97	67.48	14.65	19.36
41.42	8.01	60.97	67.48	14.65	19.36
42.49	7.47	63.33	72.25	14.02	17.91
43.57	6.12	58.36	65.31	15.17	21.79
43.12	7.48	64.47	72.41	13.82	17.71
43.06	6.94	63.60	72.06	15.70	19.42

- **What it does**
  - Displays data in a nicely formatted table
- **When it is used**
  - When you need to display your data set in a table format

Outputs: plot

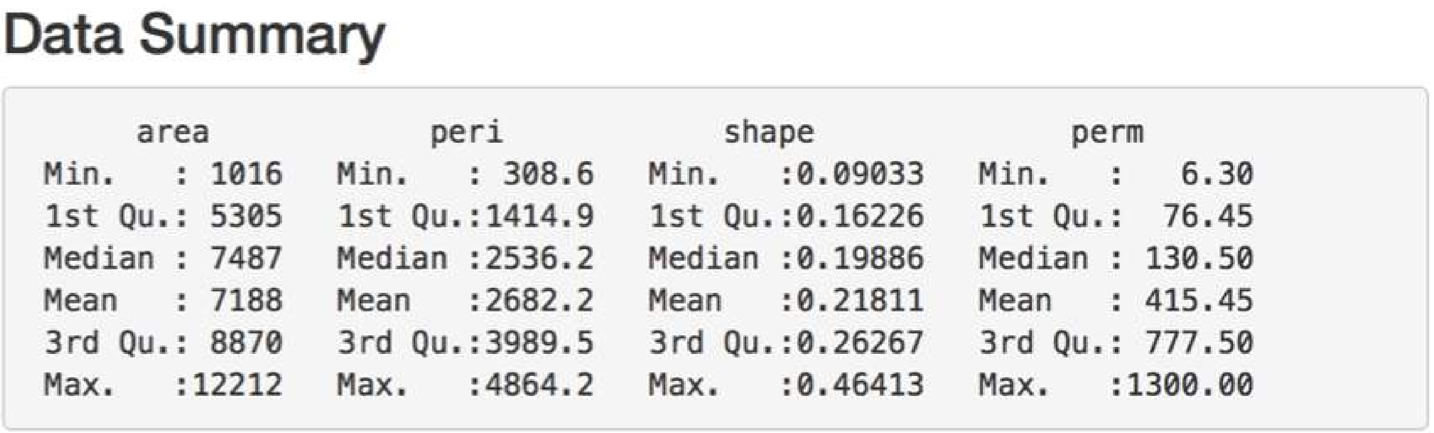
- **What it looks like**



- **What it does**
  - Plots a graph
- **When it is used**
  - To display a graph created with base plot, ggplot2, or another plotting library

Outputs: print

- **What it looks like**



Data Summary

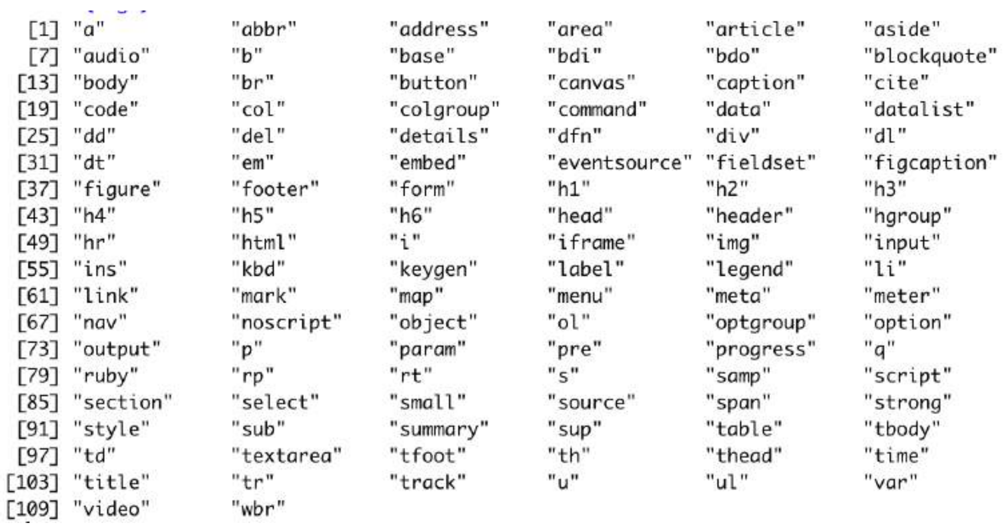
```
     area            peri            shape              perm        
 Min.   : 1016   Min.   : 308.6   Min.   :0.09033   Min.   :   6.30  
 1st Qu.: 5305   1st Qu.:1414.9   1st Qu.:0.16226   1st Qu.:  76.45  
 Median : 7487   Median :2536.2   Median :0.19886   Median : 130.50  
 Mean   : 7188   Mean   :2682.2   Mean   :0.21811   Mean   : 415.45  
 3rd Qu.: 8870   3rd Qu.:3989.5   3rd Qu.:0.26267   3rd Qu.: 777.50  
 Max.   :12212   Max.   :4864.2   Max.   :0.46413   Max.   :1300.00  
```

- **What it does**
  - Prints output that would normally be printed to the console
- **When it is used**
  - To show output from functions that you normally look at in the console, such as`str(costa)`,`summary(costa)`,or `dim(costa)`

HTML tags

- R Shiny can use HTML tags in the UI for a more visually appealing display
- Any HTML tag can be used, including header, footer, reference links, tables, etc.
- You can find all the HTML tags in:

```
shiny::tags

names(tags)
```



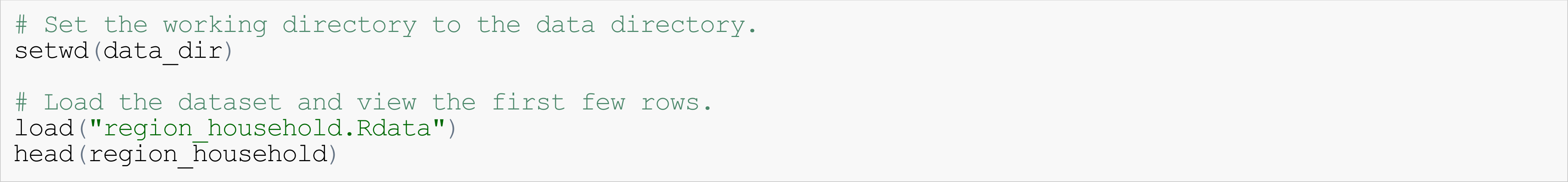
```
  [1] "a"          "abbr"       "address"    "area"        "article"    "aside"
  [7] "audio"      "b"          "base"       "bdi"         "bdo"        "blockquote"
 [13] "body"       "br"         "button"     "canvas"      "caption"    "cite"
 [19] "code"       "col"        "colgroup"   "command"     "data"       "datalist"
 [25] "dd"         "del"        "details"    "dfn"         "div"        "dl"
 [31] "dt"         "em"         "embed"      "eventsource" "fieldset"   "figcaption"
 [37] "figure"     "footer"     "form"       "h1"          "h2"         "h3"
 [43] "h4"         "h5"         "h6"         "head"        "header"     "hgroup"
 [49] "hr"         "html"       "i"          "iframe"      "img"        "input"
 [55] "ins"        "kbd"        "keygen"     "label"       "legend"     "li"
 [61] "link"       "mark"       "map"        "menu"        "meta"       "meter"
 [67] "nav"        "noscript"   "object"     "ol"          "optgroup"   "option"
 [73] "output"     "p"          "param"      "pre"         "progress"   "q"
 [79] "ruby"       "rp"         "rt"         "s"           "samp"       "script"
 [85] "section"    "select"     "small"      "source"      "span"       "strong"
 [91] "style"      "sub"        "summary"    "sup"         "table"      "tbody"
 [97] "td"         "textarea"   "tfoot"      "th"          "thead"      "time"
[103] "title"      "tr"         "track"      "u"           "ul"         "var"
[109] "video"      "wbr"
```

Module completion checklist

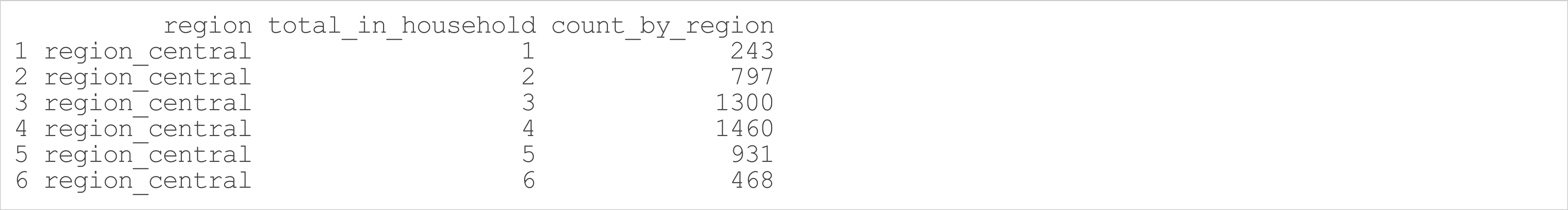
Objective	Complete
Identify RShiny tools and discuss how they improve user experience	✔
Set up the layout to implement a simple dashboard	✔
Describe various output formats and their functionalities	✔
Integrate output formats into the simple dashboard	
Describe various control widgets and their functionalities	

Set up: load the dataset

- Now that we have a base app, let's create an application to visualize our **Costa Rican dataset**
- Remember, the households and individuals in this dataset are characterized by variables that include features about the house they live in, region, gender, age, education, etc.
- We will be using the `region_household` dataset
  - This dataset gave us a summary of the total number of households in each Costa Rican region



```
# Set the working directory to the data directory.
setwd(data_dir)

# Load the dataset and view the first few rows.
load("region_household.Rdata")
head(region_household)
```



```
          region total_in_household count_by_region
1 region_central                  1             243
2 region_central                  2             797
3 region_central                  3            1300
4 region_central                  4            1460
5 region_central                  5             931
6 region_central                  6             468
```

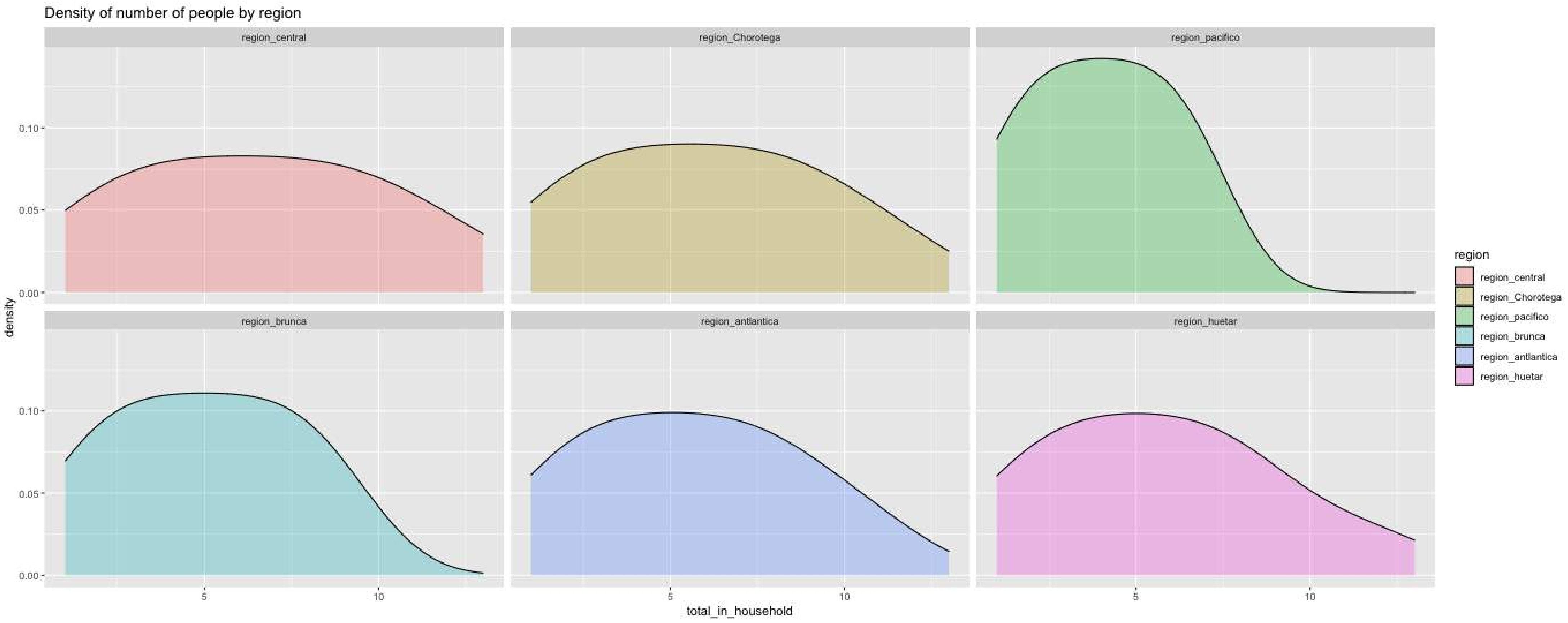
Static density plot based on regions

- Let's create an app to generate the same plot we had defined in a previous module using `ggplot`
- Using RShiny, we will add an interactive feature to the previously static plot

```
# This is how our static `ggplot` density plot was created.
density_plot <-
ggplot(region_household,            #<- set data
       aes(x = total_in_household,  #<- map `x value`
           fill = region )) +       #<- map fill
       geom_density(alpha = 0.3) +  #<- adjust fill transparency
       labs(title =                 #<- add title
            "Density of number of people by region") +
      facet_wrap (~ region,         #<- make facets by 'region'
                  ncol = 3)         #<- set a 3-column grid
```

Static density plot based on regions

```
density_plot
```



Adding density plot to our base app: UI

- We will add the plot object `densityplot` created in the server to our base UI
- We will also update the titles

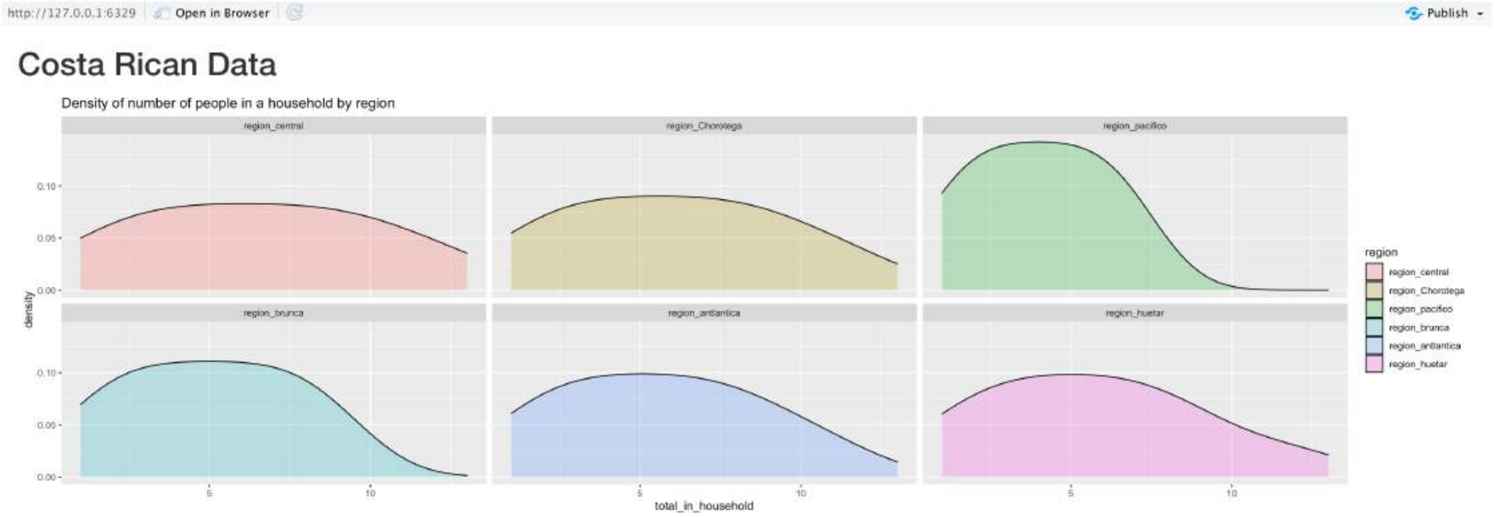
```
library(shiny)

ui <- fluidPage(

  # Title of the app.
  titlePanel("Costa Rican Data"),

  # Render the output as plot.
  plotOutput(outputId = "densityplot")

)
```

Adding density plot to our base app: server

```
library(shiny)
library(dplyr)
library(ggplot2)

# Define server logic.
server <- function(input, output) {

  # Load the dataset.
  load("region_household.Rdata")

output$densityplot<-
  renderPlot({   #<- function to create plot object to send to UI

  # Create density plot.
    ggplot(region_household,              #<- set data
           aes(x = total_in_household,    #<- map `x value`
               fill = region )) +         #<- map fill
           geom_density(alpha = 0.3)   +  #<- adjust density fill
           labs(title = "Density of number of people in a household by region") +
           facet_wrap (~ region,          #<- make facets by 'region'
                       ncol = 3)          #<- set a 3-column grid

     }) # end of renderPlot
}# end of server
```

Adding density plot to our base app

- Navigate to `introduction-to-Rshiny-code-part-1/3-app-with-plot` folder
- We now have an app which creates a static density plot facet grid



Knowledge check 1

Exercise 1

Module completion checklist

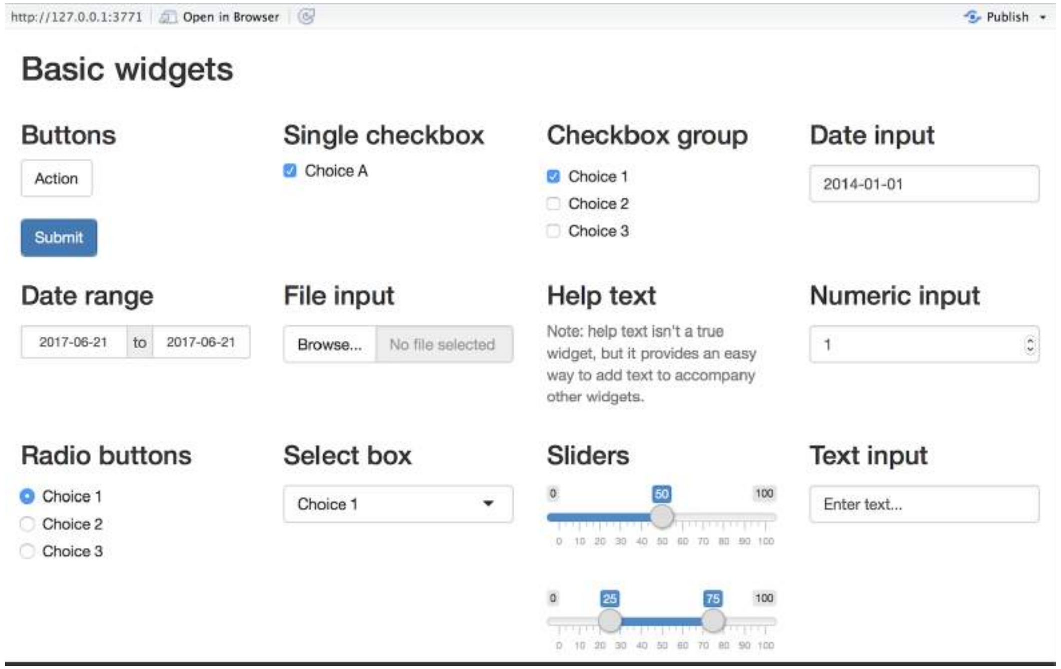
Objective	Complete
Identify RShiny tools and discuss how they improve user experience	✔
Set up the layout to implement a simple dashboard	✔
Describe various output formats and their functionalities	✔
Integrate output formats to the simple dashboard	✔
Describe various control widgets and their functionalities	

Customizing your application

- When building an application, there are four elements that you can customize:
  - **Outputs:** create different output elements, such as plots or tables
  - **Inputs:** use various input widgets the user can utilize in the application
  - **Reactivity:** define how outputs get updated based on the inputs the user chooses
  - **Layout:** decide what your app should look like; use a default one or create your own

Inputs: built-in widgets

- Shiny has built-in **widgets** for user input
- The ***widget gallery*** is a useful resource to see what widgets are available and get the code for each widget



Basic widgets

Buttons

Single checkbox

☑ Choice A

Checkbox group

☑ Choice 1
☐ Choice 2
☐ Choice 3

Date input

2014-01-01

Date range

2017-06-21 to 2017-06-21

File input

Help text

Note: help text isn't a true widget, but it provides an easy way to add text to accompany other widgets.

Numeric input

1

Radio buttons

● Choice 1
○ Choice 2
○ Choice 3

Select box

Choice 1

Sliders

Text input

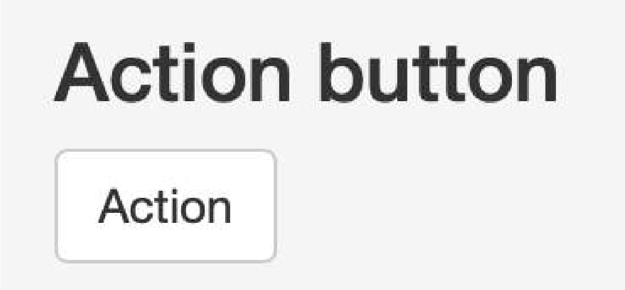
Enter text...

Inputs: widgets we'll explore

Output	Functions used
Action button	actionButton
Slider	sliderInput
Slider range	sliderInput
Single checkbox	checkboxInput
Checkbox group	checkboxGroupInput
Numeric input	numericInput
Text input	textInput
Radio buttons	radioButtons

Inputs: action button

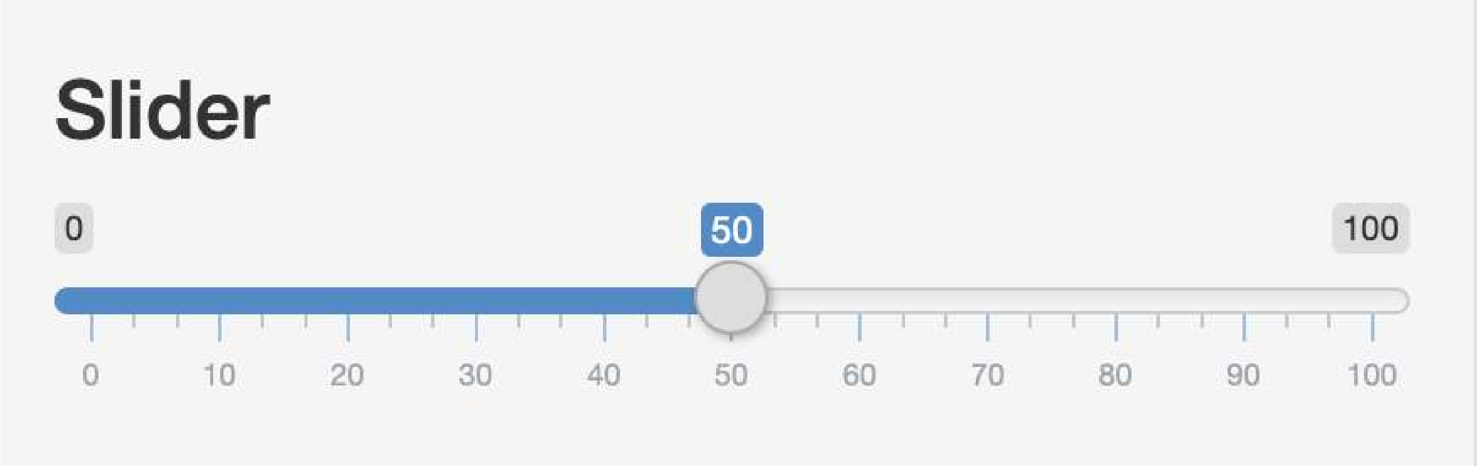
- **What it looks like**



- **What it does**
  - Functions like the 'Enter' key on your key board
- **When it is used**
  - Whenever you want the user to confirm an action, such as update a graph or perform a calculation

Inputs: slider

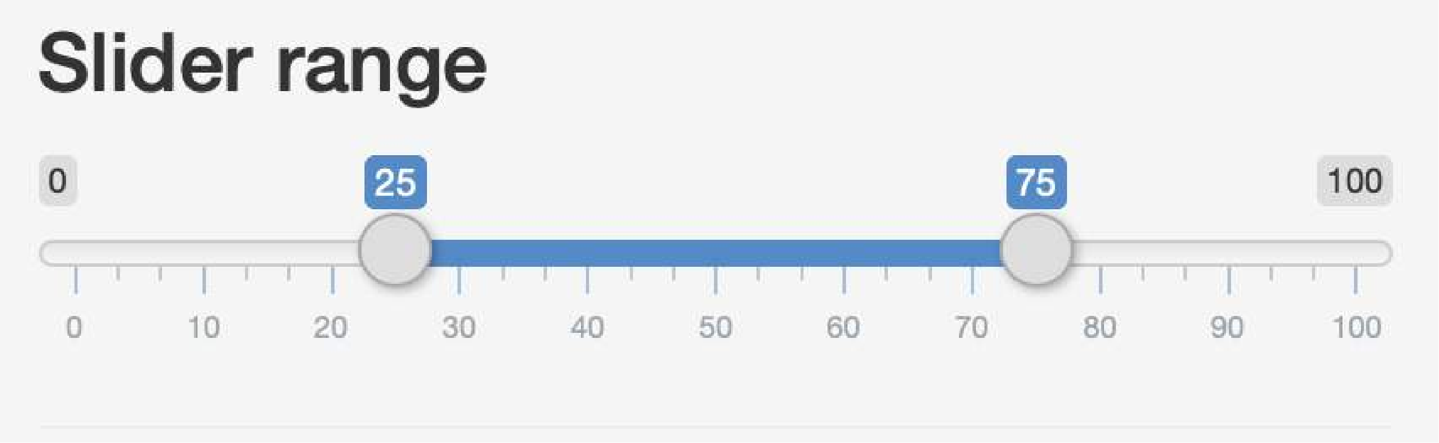
- **What it looks like**



- **What it does**
  - Lets the user select a specific number by moving the slider with the mouse
- **When it is used**
- Whenever you want the user to select a number, such as:
  - Select number of bins in a histogram
  - Select number of rows to be displayed in a table
  - Select a particular year's data to be displayed

Inputs: slider range

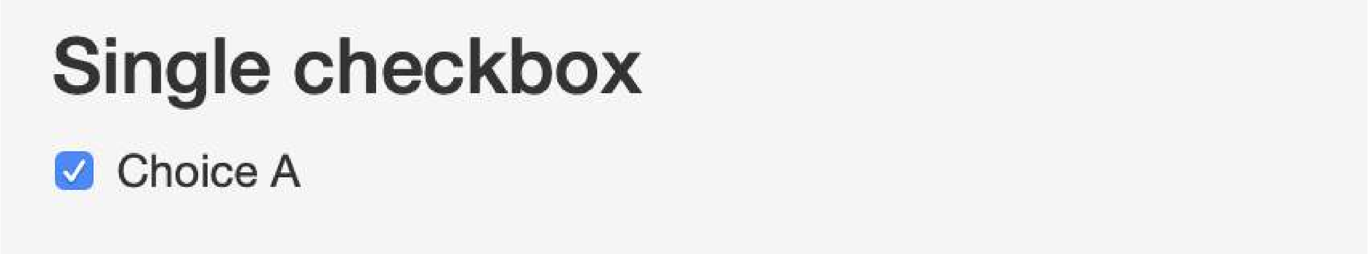
- **What it looks like**


Slider range

- **What it does**
  - Lets the user select a range of number by moving the slider ends with the mouse
- **When it is used**
  - Whenever you want the user to select a range of numbers, such as select the years for which a map should be displayed

Inputs: single checkbox

- **What it looks like**



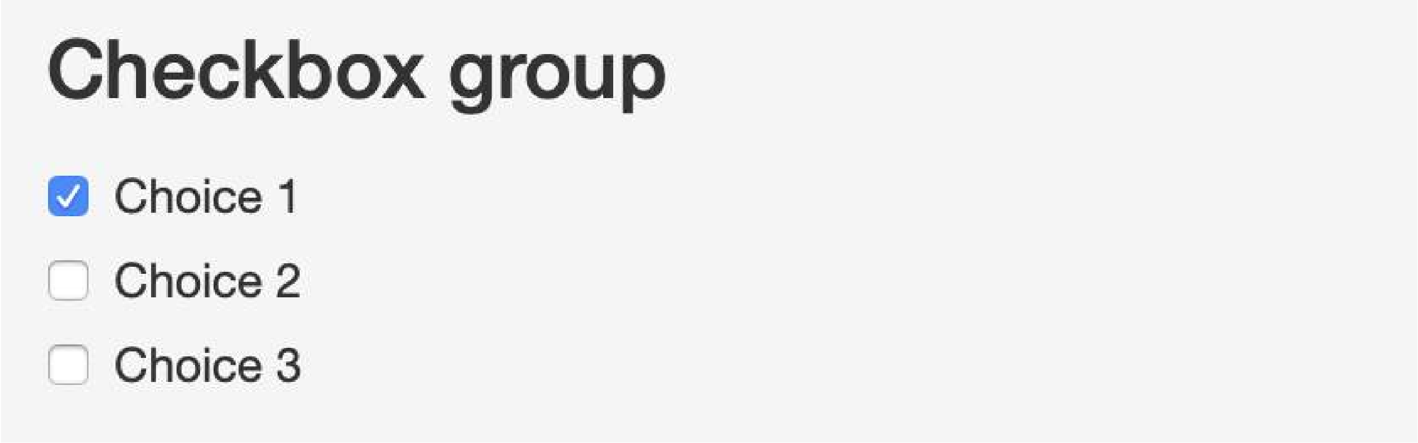
Single checkbox

☑ Choice A

- **What it does**
  - Lets the user select/unselect an option
- **When it is used**
- User should be able to toggle an option “on” and “off”, such as:
  - Select if individual observations should be shown in graph
  - Select if table should show a header
  - Select if data should be updated automatically

Inputs: checkbox group

- **What it looks like**



Checkbox group

☑ Choice 1

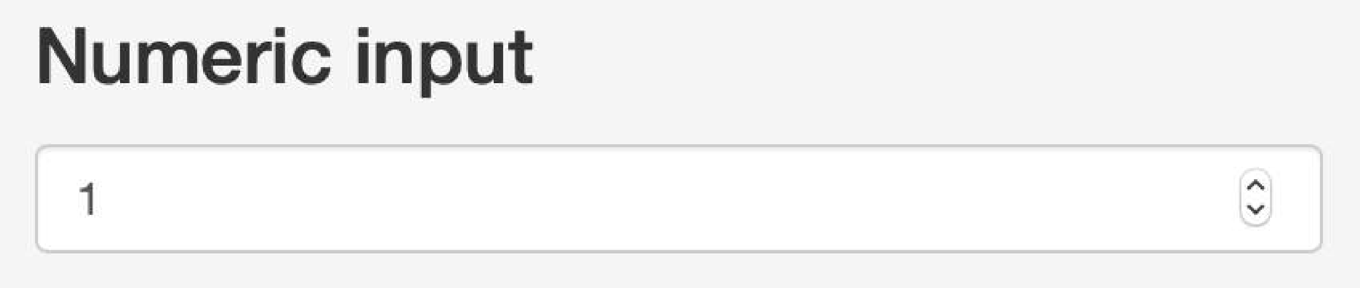
□ Choice 2

□ Choice 3

- **What it does**
  - Lets the user select more than one option in a group of choices
- **When it is used**
  - To select which age groups to include in graphs
  - To select which countries and gender to include in the analysis

Inputs: numeric input

- **What it looks like**



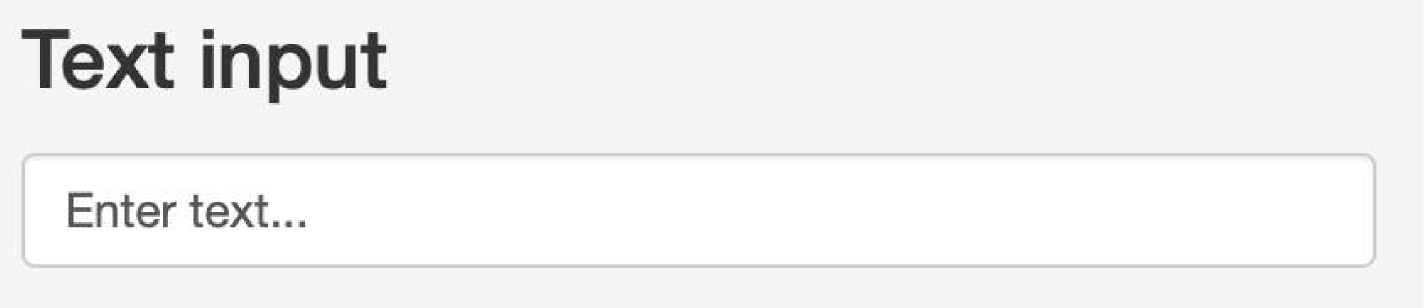
Numeric input

1

- **What it does**
  - Lets the user specify a number
- **When it is used**
  - To let users specify year of birth, income level or zip code

Inputs: text input

- **What it looks like**



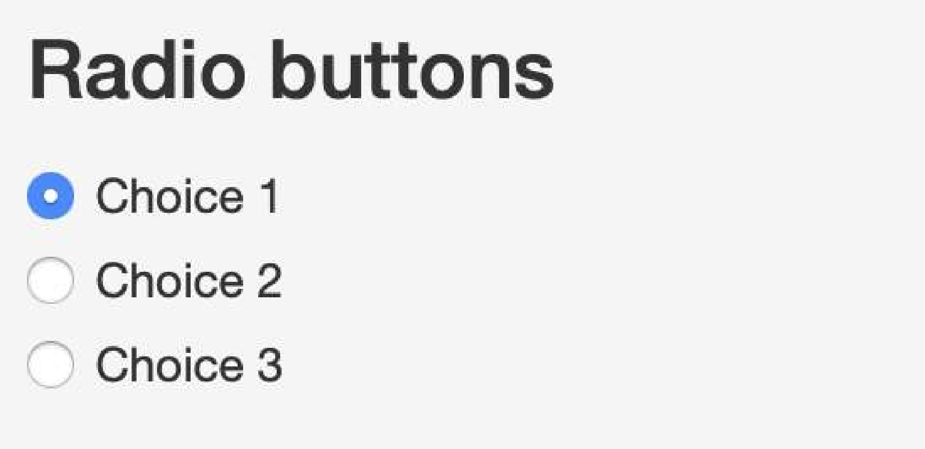
Text input

Enter text...

- **What it does**
  - Lets the user type text
- **When it is used**
  - To allow users to type names, addresses, and email addresses

Inputs: radio button

- **What it looks like**



Radio buttons

● Choice 1

○ Choice 2

○ Choice 3

- **What it does**
  - Lets the user select one option out of a group of options
- **When it is used**
  - Let user select the dataset to be displayed in a graph
  - Let user select the type of graph to be displayed
  - Let user select a color in a graph

Knowledge check 2

Module completion checklist

Objective	Complete
Identify RShiny tools and discuss how they improve user experience	✔
Set up the layout to implement a simple dashboard	✔
Describe various output formats and their functionalities	✔
Integrate output formats into the simple dashboard	✔
Describe various control widgets and their functionalities	✔

Summary

- In this module we talked about the components of the RShiny application, including UI and Server
- Also, we introduced common inputs and outputs and built a base application
- In the next module, we will integrate control widgets into our dashboard
- We will also discuss reactivity and layout elements

This completes our module

Congratulations!