DATA SOCIETY®

RShiny - Day 1

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

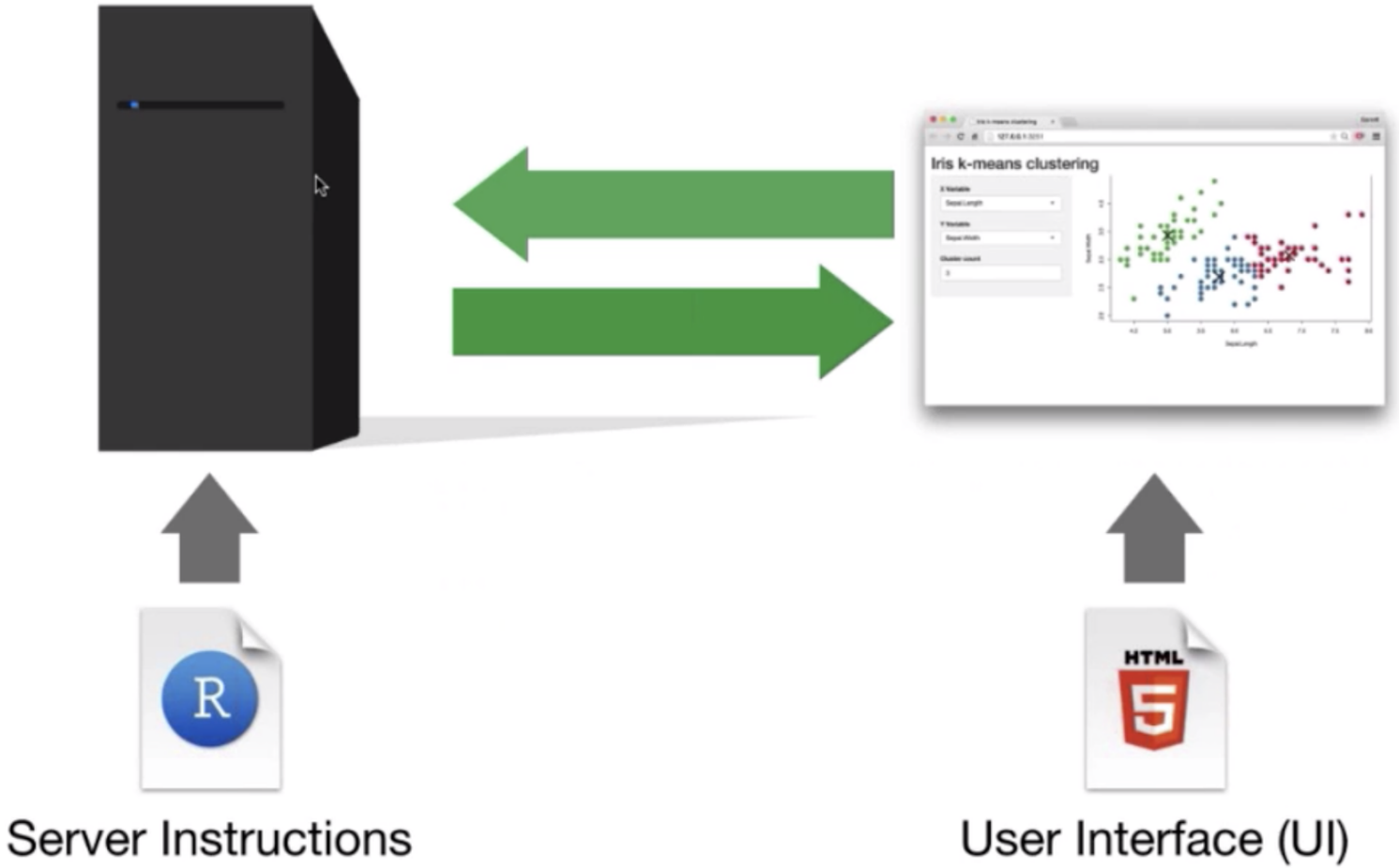
Module completion checklist

Objective	Complete
Explain the basics of RShiny	
Describe RShiny package and best practices for code modularity	
Set up your own application	
Customize your application with input widgets	
Customize your application with output formats	
Incorporate HTML tags to shiny dashboard	
Display images and pdf files on the dashboard	

Intro to 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
- RShiny combines the analytical abilities of R with display abilities of web design software

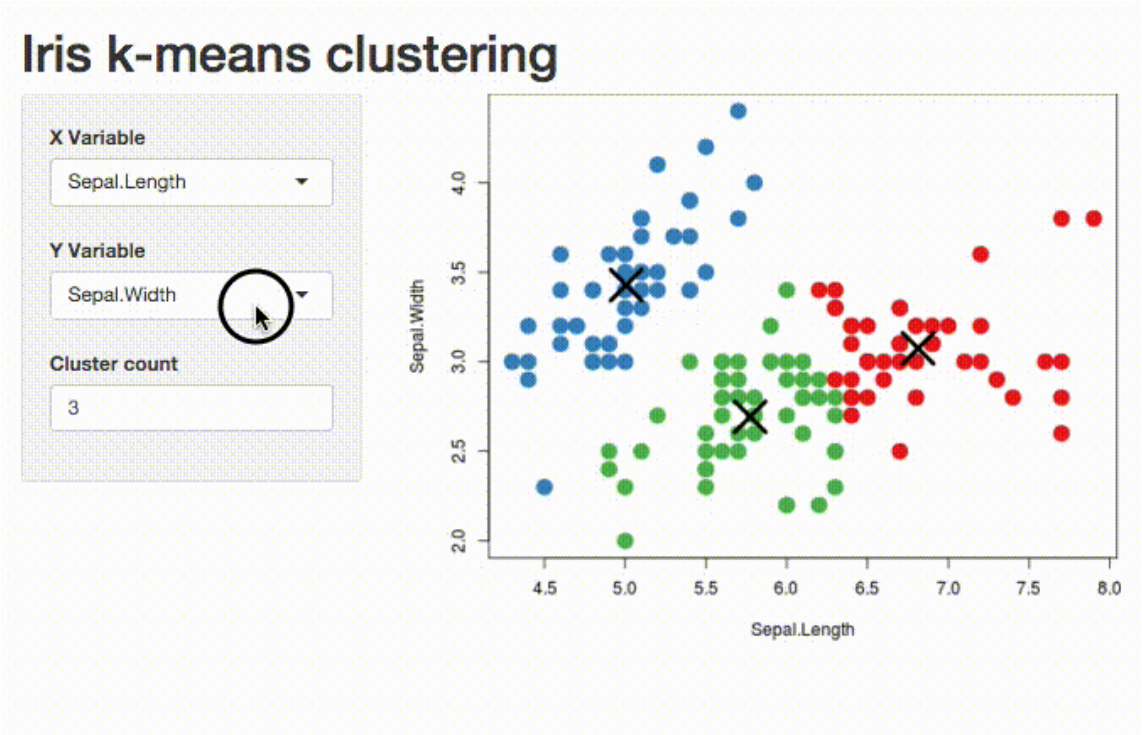
Web page connected to computer running R



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

RShiny example

- Here is a simple RShiny example



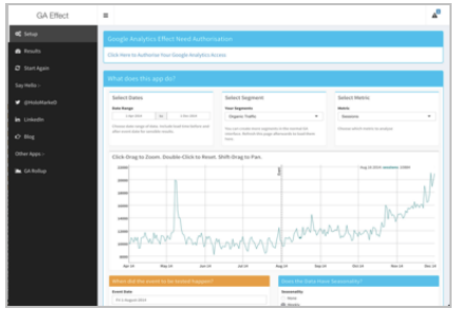
Overview of RShiny use cases

- RShiny can be helpful in the following situations:
  - **Let users explore data**
  - **Example:** With RShiny, you can create graphs that the user can explore interactively
  - **Let users conduct their own analysis**
  - **Example:** With RShiny, you can create a template that lets the users do their own analysis of the data
  - **Communicate results from your analysis with users**
  - **Example:** With RShiny, you can create a dashboard that neatly showcases your work and insights

Shiny user showcase

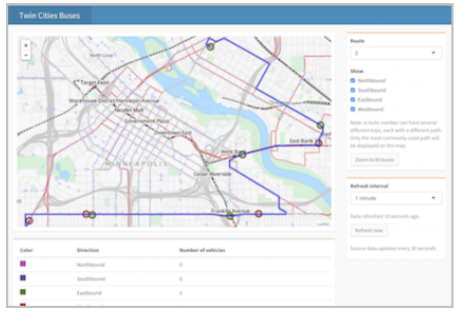
- You can find a wealth of RShiny applications here: ***https://www.rstudio.com/products/shiny/shiny-user-showcase/***
- Check out these applications, they might be helpful to inspire some of your own work

Shiny Apps for the Enterprise



MARKETING EFFECTS

See the effects of your marketing campaigns.

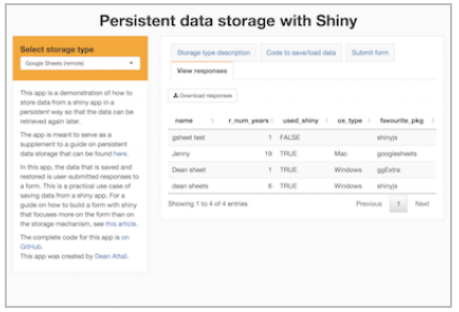


LOCATION TRACKER

Track locations over time with streaming data.

DOWNLOAD MONITOR

Streaming download rates visualized as a bubble chart.



PERSISTENT STORAGE

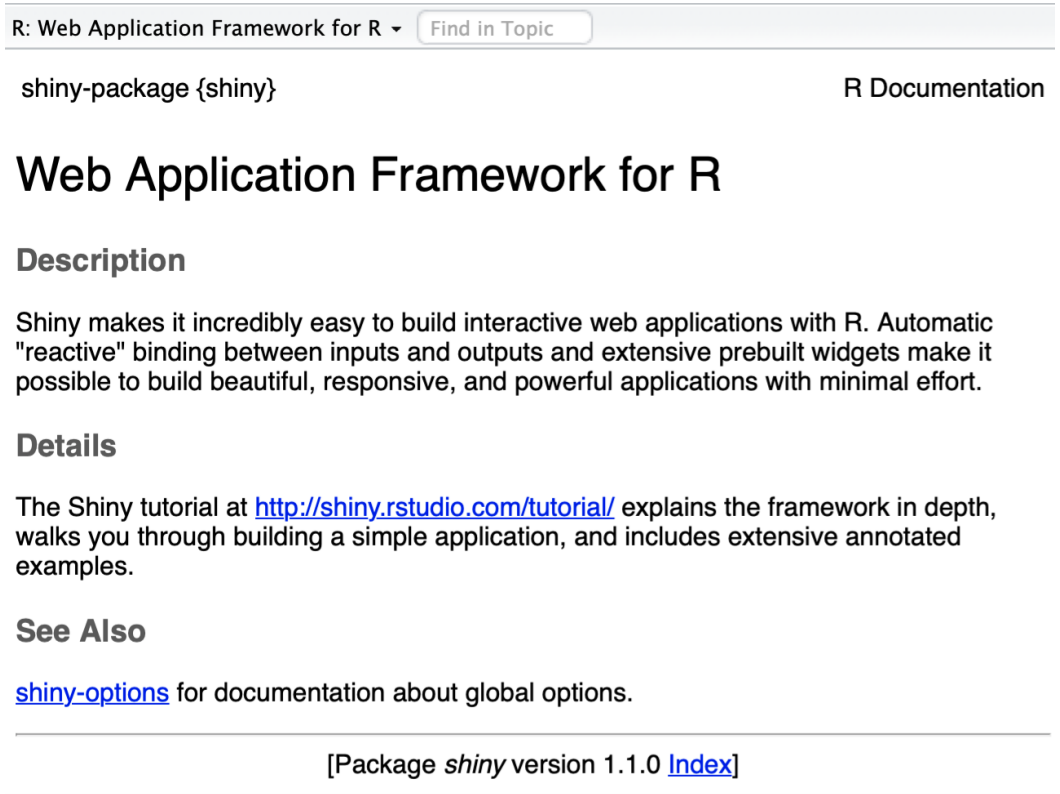
Save data from your apps to local files, servers, databases, and more.

Module completion checklist

Objective	Complete
Explain the basics of RShiny	✔
Describe RShiny package and best practices for code modularity	
Set up your own application	
Customize your application with input widgets	
Customize your application with output formats	
Incorporate HTML tags to shiny dashboard	
Display images and pdf files on the dashboard	

The RShiny package

```
library(shiny)

help("shiny-package")
```



R: Web Application Framework for R ▾ Find in Topic

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.

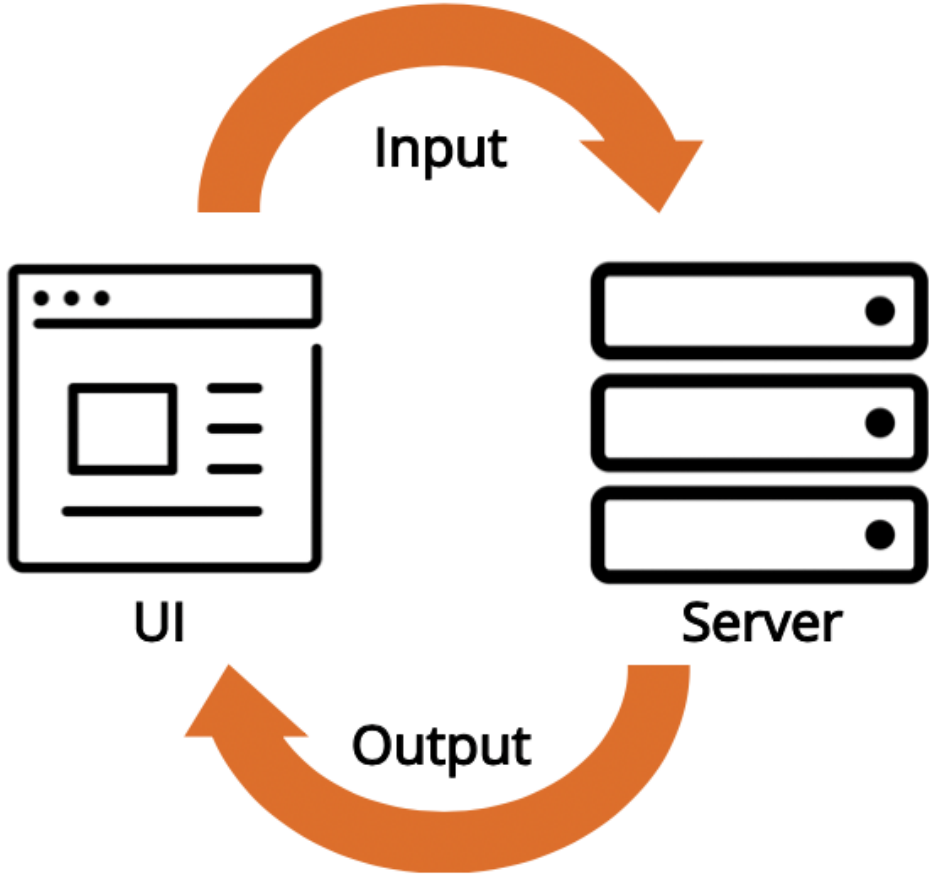
See Also

shiny-options for documentation about global options.

[Package *shiny* version 1.1.0 Index]

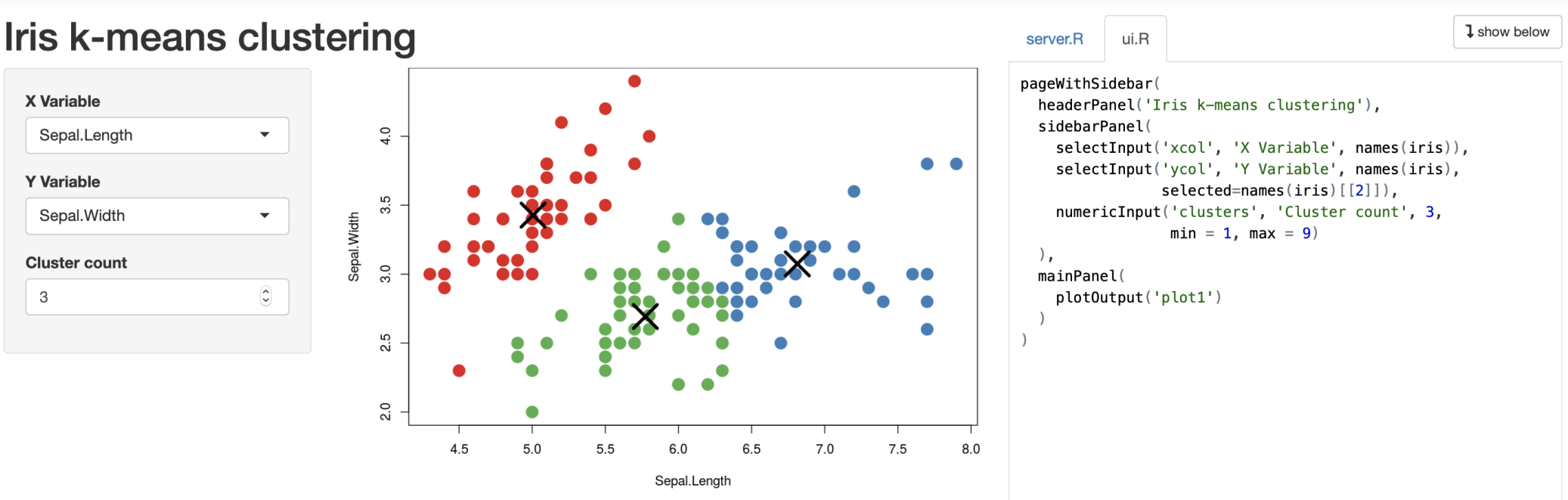
UI and Server

- 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



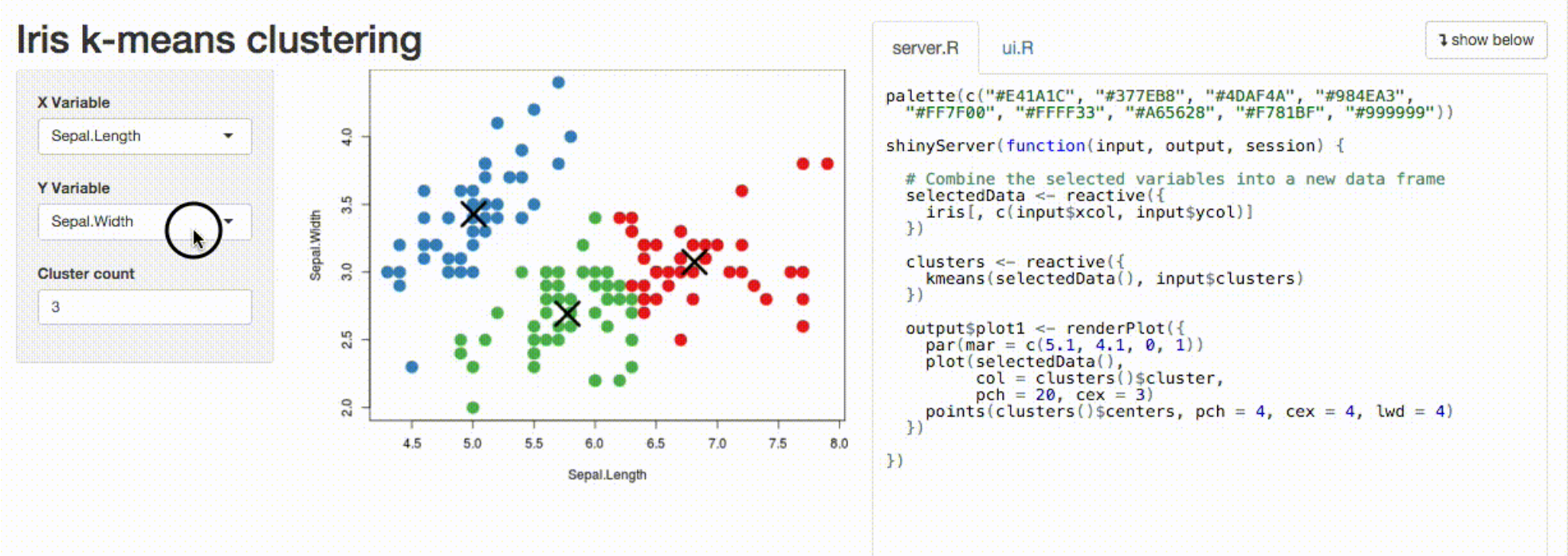
UI example

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



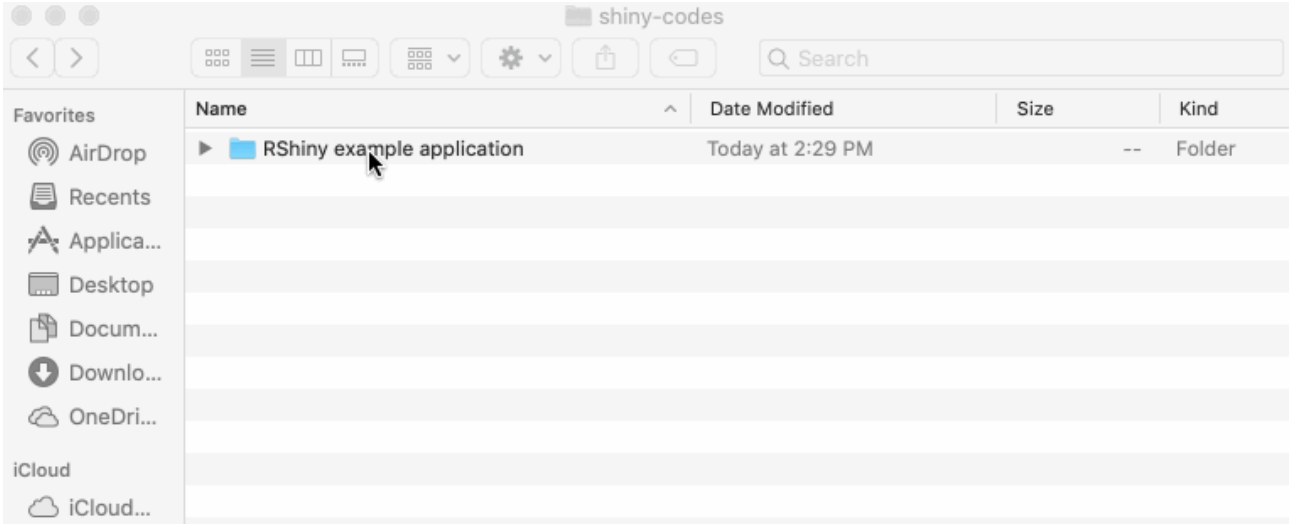
Server example

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



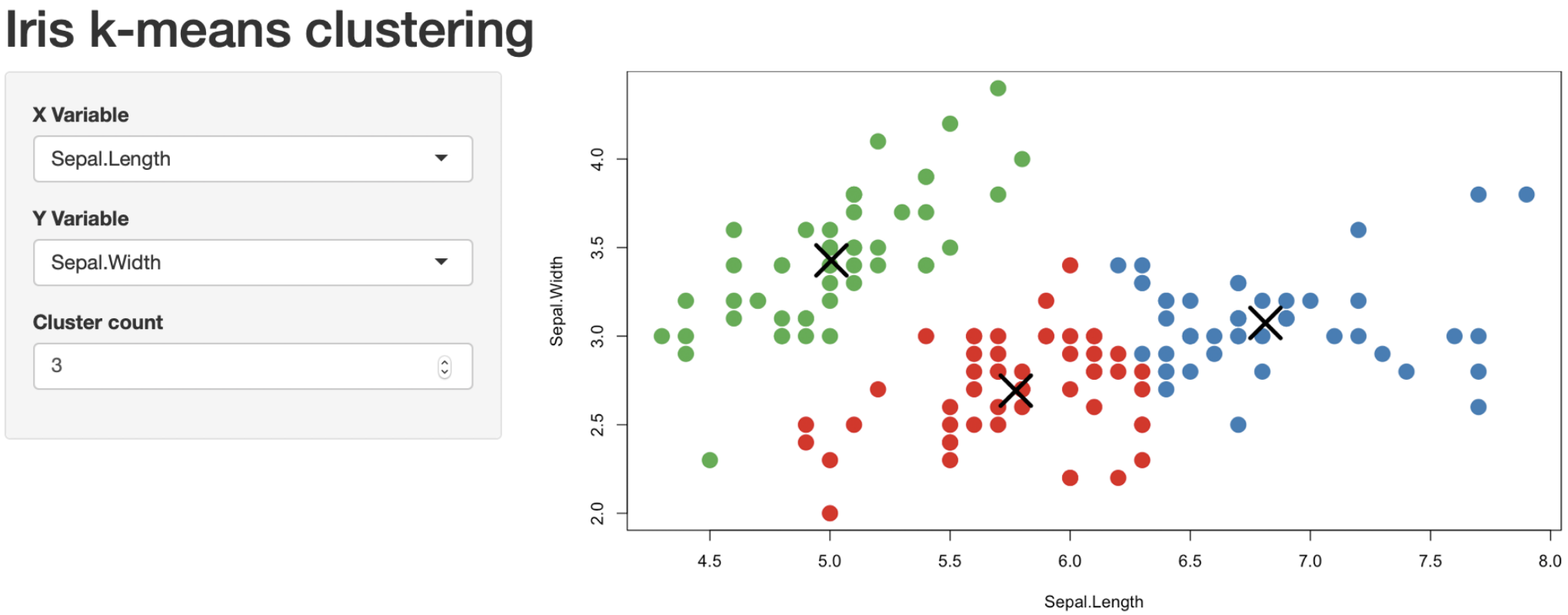
Separate files for UI code and server code

- **Coding best practice**: write UI code and server code in **two separate** files
- In order for the RShiny application to work, these separate files need to be in the same folder
- 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** that looks like this:



Launching your first RShiny application

- **Step 1:** Navigate to `shiny` folder, then `day-1` subfolder, then `1-example` subfolder
- **Step 2:** Open `ui.R` or `server.R` file
- **Step 3:** Click 'Run App'
- What you will see:



Module completion checklist

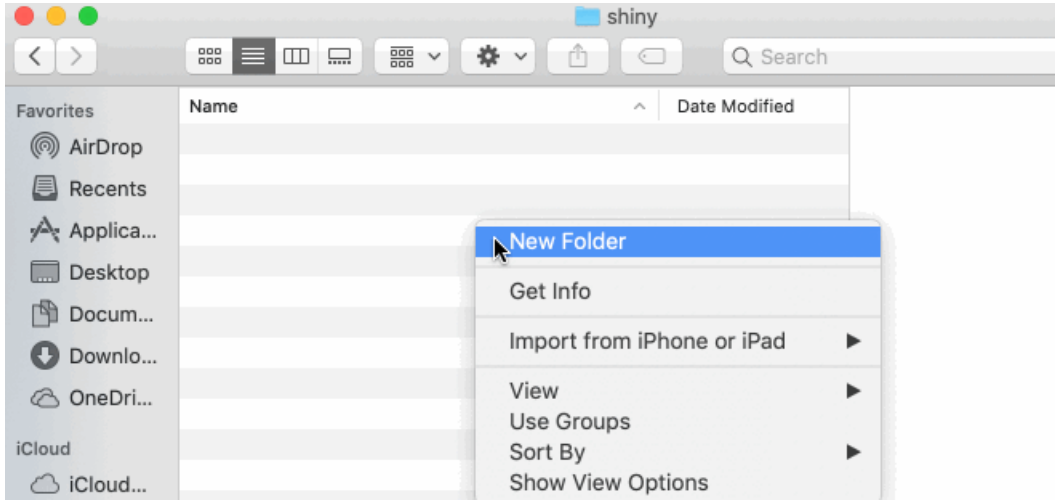
Objective	Complete
Explain the basics of RShiny	✔
Describe RShiny package and best practices for code modularity	✔
Set up your own application	
Customize your application with input widgets	
Customize your application with output formats	
Incorporate HTML tags to shiny dashboard	
Display images and pdf files on the dashboard	

Setting up your own Shiny app

- Always remember to first set up the app through the following steps before you begin coding
  - Set the folder structure for the new app
  - Create `server.R` and `ui.R` scripts with base structure
  - Set the working directory
  - Load the `shiny` package
  - Load the required datasets

Folder structure

- As we just learned, for every application, we need to have the two files `ui.R` and `server.R` in a separate folder. We create a folder structure as follows:
- **Step 1:** In the main `af-werx` folder, keep the folder called `shiny`
- **Step 2:** Within `shiny` folder we have a subfolder `day-1` which contains all the codes for this module
- **Step 3:** Within `day-1`, we have numbered each app in the order we will be using in this module
- **Step 4:** Each app contains a `server.R` and `ui.R` file
- **Step 5:** Each `server` and `ui` file contains a constant structure and then we keep adding any additional code needed for the app



Constant structure of ui and server code

- The constant structure is:
  - Setting the working directory to `data_dir`
  - Loading Shiny package
  - Loading the dataset we will be using for the app

ui.R base structure

- Adding the following base structure in `ui.R`

```
#### Set working directory ####
session_info = sessionInfo()

platform = session_info$platform # capture info on OS being used
directory = "af-werx" # set directory variable to 'af-werx'

if(length(grep("linux|apple", platform)) > 0){ # determine if using Linux-based system
  Sys.getenv("USER")
  dir = paste0("~/Desktop/", directory) # if Linux or Apple, create dir variable here
}else{
  username = Sys.getenv("USERNAME")
  dir=paste0("C:/Users/", username, "/Desktop/", directory) # if Windows, create dir variable here
}

# Create data-dir and set working directory to data directory.
data_dir = paste0(dir, "/data")
setwd(data_dir)

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

#### Load the CMP dataset ####
CMP = read.csv("ChemicalManufacturingProcess.csv", header = TRUE, stringsAsFactors = FALSE)

##### Create UI ####
ui <- fluidPage(

)
```

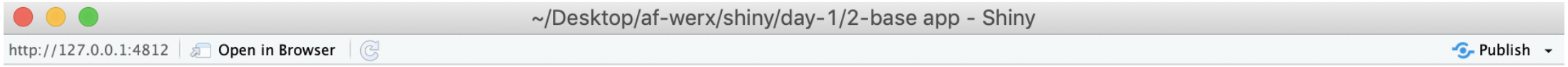
server.R base structure

- Adding the following base structure in `server.R`

```
#### Set working directory ####
session_info = sessionInfo()

platform = session_info$platform # capture info on OS being used
directory = "af-werx" # set directory variable to 'af-werx'

if(length(grep("linux|apple", platform)) > 0){ # determine if using Linux-based system
  Sys.getenv("USER")
  dir = paste0("~/Desktop/", directory) # if Linux, create dir variable here
}else{
  username = Sys.getenv("USERNAME")
  dir=paste0("C:/Users/", username, "/Desktop/", directory) # if Windows, create dir variable here
}

# Create data-dir and set working directory to data directory
data_dir = paste0(dir, "/data")
setwd(data_dir)

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

#### Load the CMP dataset ####
CMP = read.csv("ChemicalManufacturingProcess.csv", header = TRUE, stringsAsFactors = FALSE)

##### Create server ####
server <- function(input, output) {

}
```

Running your empty application

- Open the files from `2-base app` application
- Click on `Run app` in either ui.R or server.R
- You should see the following output (an empty RShiny app):



Knowledge check 1

Exercise 1

Module completion checklist

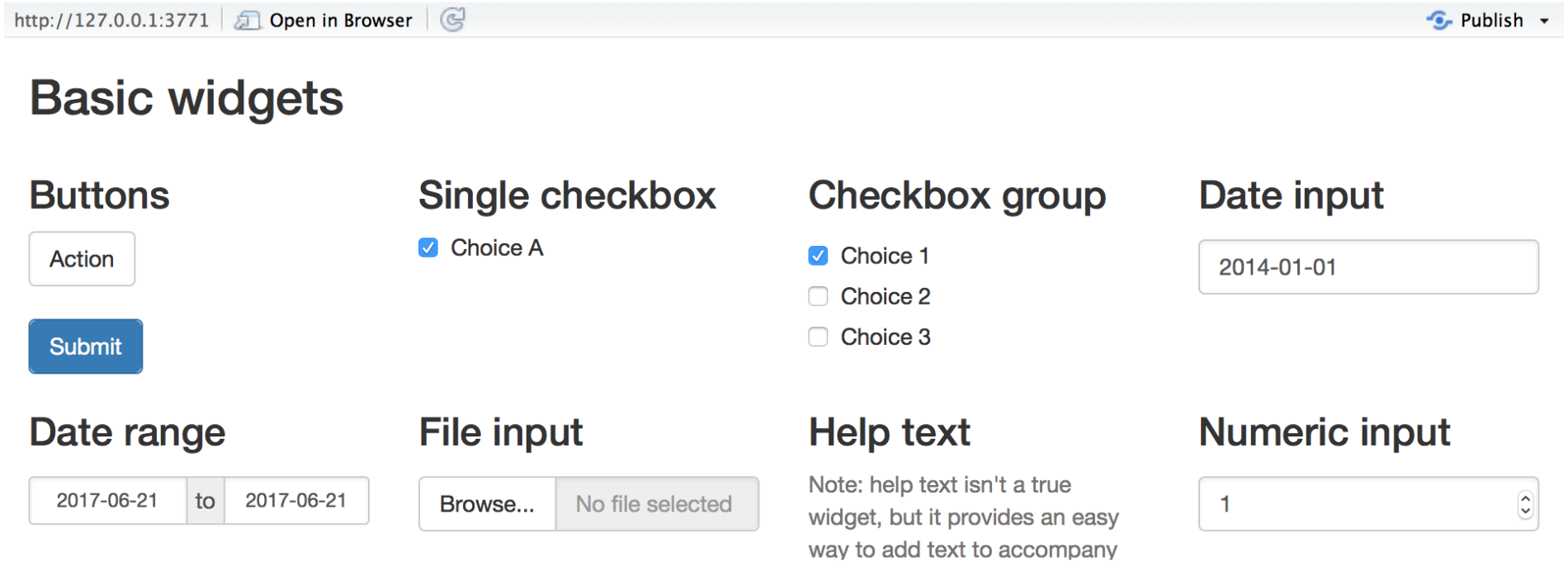
Objective	Complete
Explain the basics of RShiny	✔
Describe RShiny package and best practices for code modularity	✔
Set up your own application	✔
Customize your application with input widgets	
Customize your application with output formats	
Incorporate HTML tags to shiny dashboard	
Display images and pdf files on the dashboard	

Customizing your application

- When building your custom application, you have four elements that you can play with:
  - **Inputs:** use various input widgets the user can utilize in the application
  - **Outputs:** create different output elements, such as plots or tables
  - **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
- Today, we will explore **input widgets** and **output elements**

Input widgets: RShiny has various 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

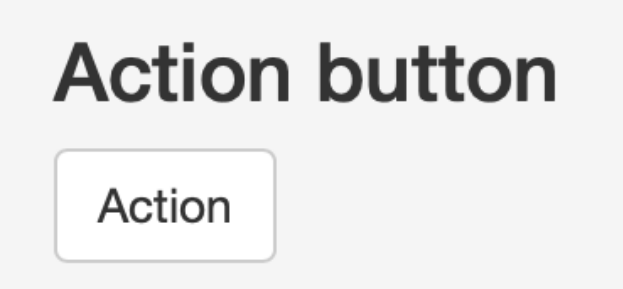


Input widgets: we are exploring 8 widgets today

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

Input widgets: action button

- **What it looks like**

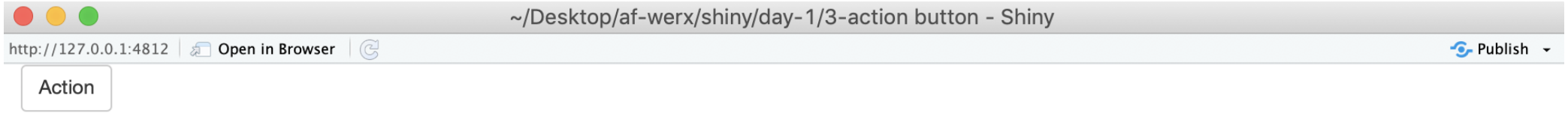


- **What it does**
  - Functions like the 'Enter' key on your key board
- **When it is used**
  - Whenever you want the user to confirm an action, such as update a graph or perform a calculation

Input widgets: action button

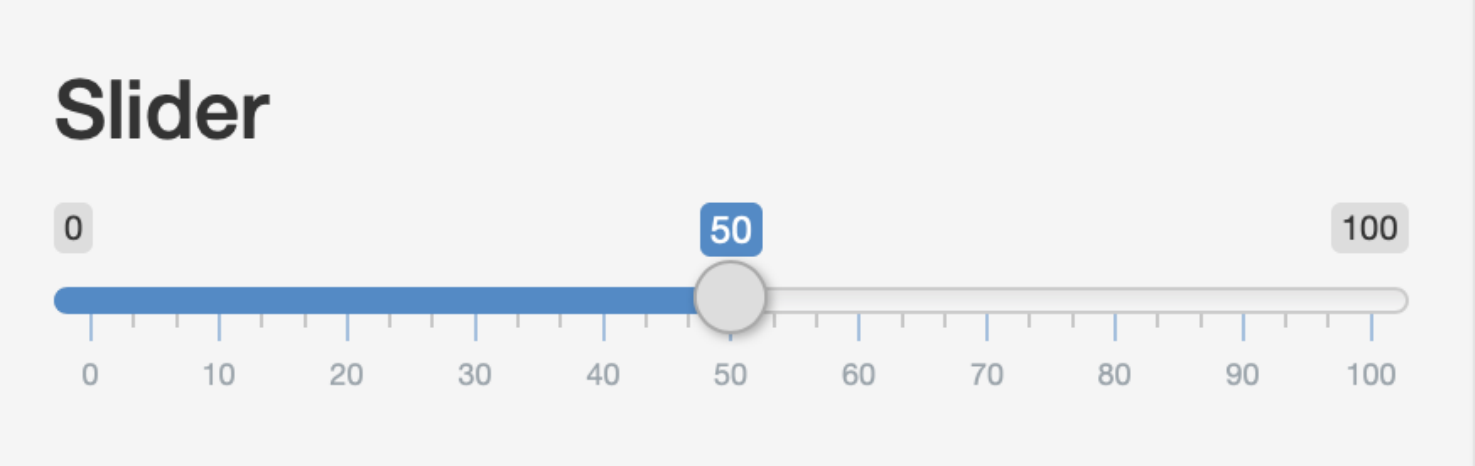
```
fluidPage(

  actionButton(        # create an actionButton element in the US
    "action",          # give a name to the actionButton element
    label = "Action") # specify a label to be displayed on the action button

)
```

- Use the files from `3-action button` app



Input widgets: slider

- **What it looks like**

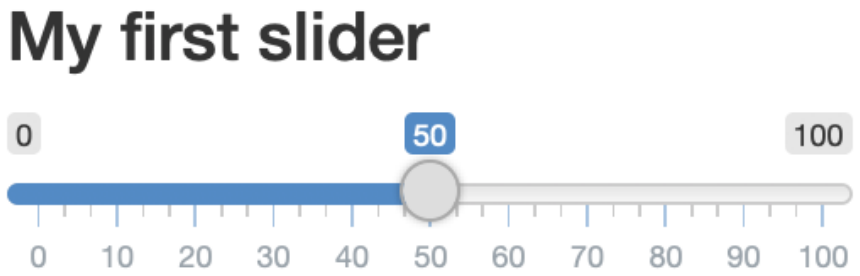


- **What it does**
  - Lets the user select a specific number by moving the slider with the mouse
- **When it is used**
- Whenever you want the user to select a number, such as:
  - Select number of bins in a histogram
  - Select number of rows to be displayed in a table
  - Select a particular year's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

Input widgets: slider

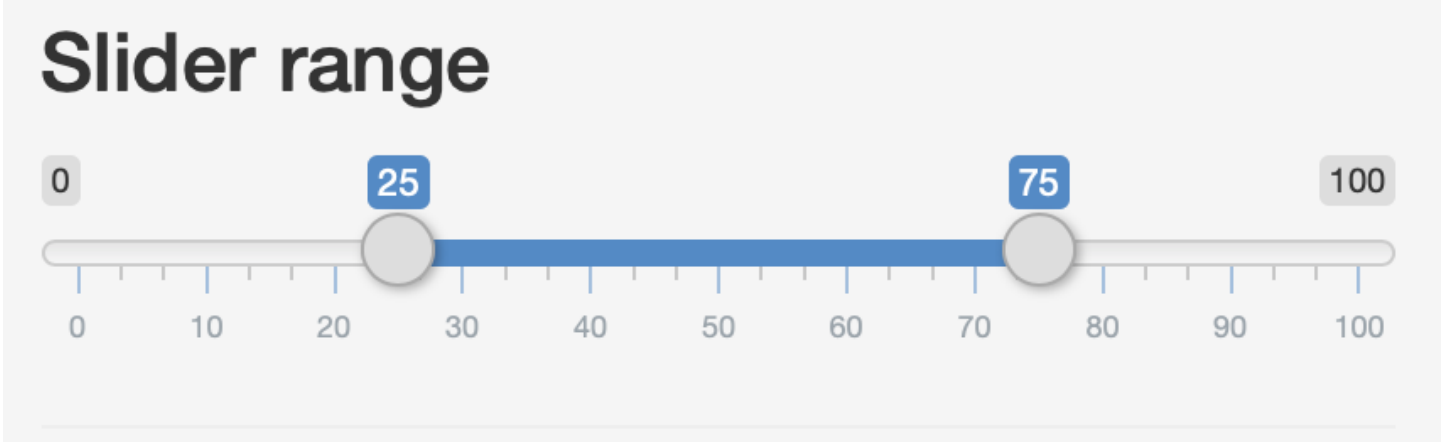
```
fluidPage(

  sliderInput(                    # create sliderInput element
    "slider1",                    # give a name to this element
    label ="My first slider",     # give a header
    min = 0,                      # specify the minimum value that can be selected
    max = 100,                    # specify the maximum value that can be selected
    value = 50                    # specify the default value that is selected
  )
)
```

- Use the files from `4-slider` app



Input widgets: slider range

- **What it looks like**

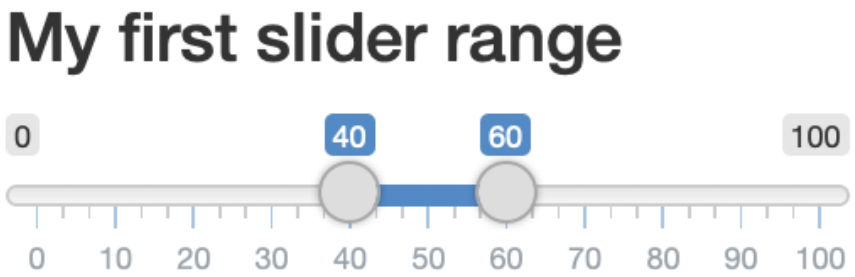


- **What it does**
  - Lets the user select a range of number by moving the slider ends with the mouse
- **When it is used**
  - Whenever you want the user to select a range of numbers, such as select the years for which a map should be displayed

Input widgets: slider range

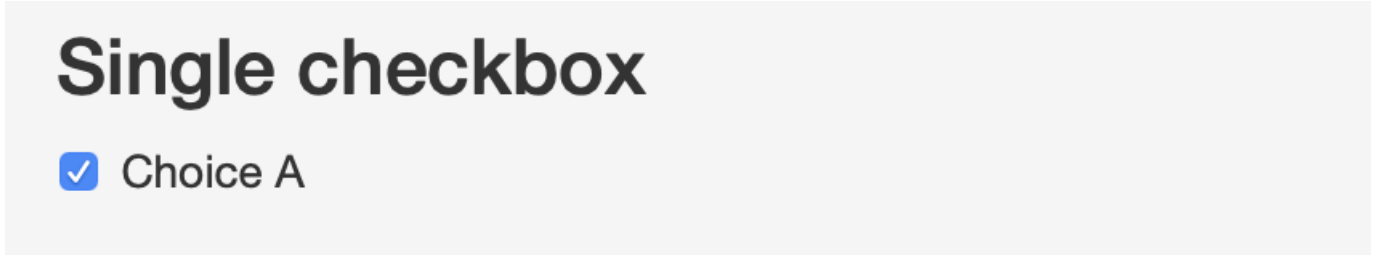
```
fluidPage(

  sliderInput(                          # create sliderInput element
    "slider1",                          # give a name to this element
    label ="My first slider range",     # give a header
    min = 0,                            # specify the minimum value that can be selected
    max = 100,                          # specify the maximum value that can be selected
    value = c(40,60)                    # specify the range of the default values that are selected
  )
)
```

- Use the files from `5-slider range` app



Input widgets: single checkbox

- **What it looks like**



- **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

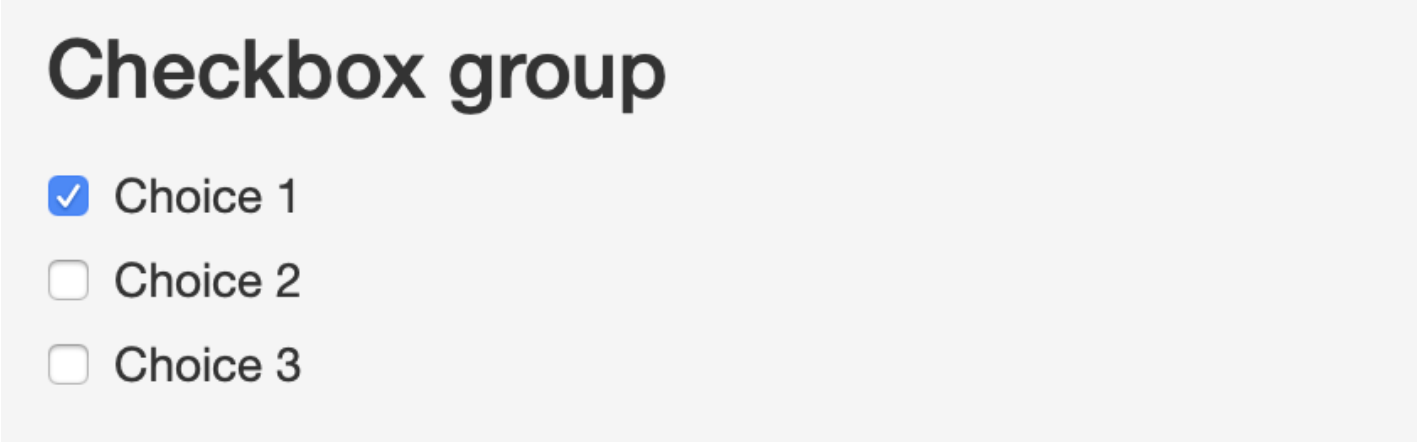
Input widgets: single checkbox

```
fluidPage(

  checkboxInput(         # create checkboxInput element
    "checkbox",          # give a name to this element
    label = "Choice 1", # specify the label of the checkbox
    value = TRUE         # specify the default value - here the checkbox will be selected by default
  )
)
```

- Use the files from `6 - single checkbox` app

Input widgets: checkbox group

- **What it looks like**

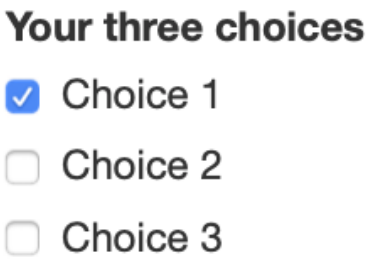


- **What it does**
  - Lets the user select more than one option in a group of choices
- **When it is used**
  - To select which age groups to include in graphs
  - To select which countries and gender to include in the analysis

Input widgets: checkbox group

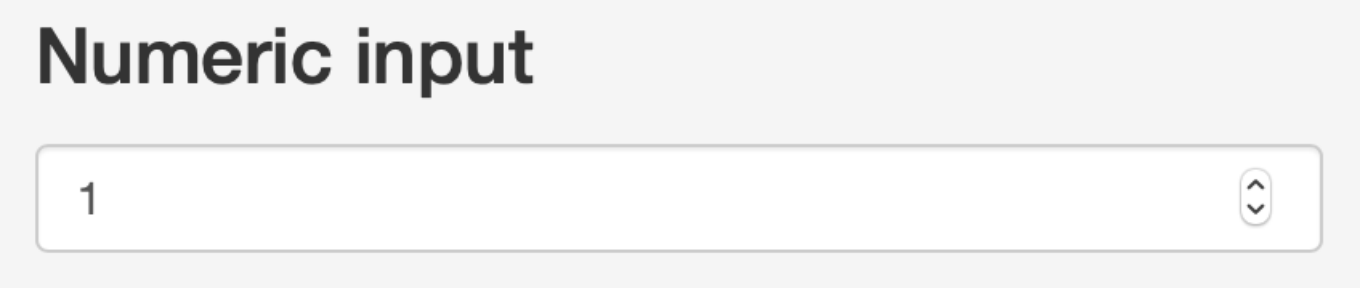
```
fluidPage(

  checkboxGroupInput(              # create CheckboxGroupInput element
    "checkGroup",                  # give a name to this elements
    label = "Your three choices", # define the title of the checkbox group to be displayed
    choices = list("Choice 1" = 1, "Choice 2" = 2, "Choice 3" = 3), # specify the choices in the group
    selected = 1                   # specify the choices to be selected by default
    )

)
```

- Use the files from `7-checkbox group` app



Input widgets: numeric input

- **What it looks like**



- **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

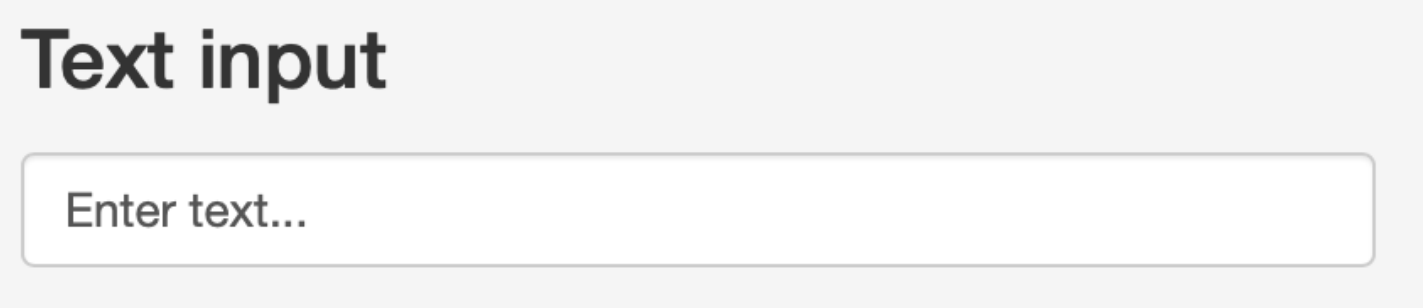
Input widgets: numeric input

```
fluidPage(

  numericInput(               # create numericInput element
    "num",                    # giva a name to this element
    label = "My first numeric input", # define the title of the element to be displayed
    value = 1                 # specify the default value
    )

)
```

- Use the files from `8-numeric input` app

Input widgets: text input

- **What it looks like**



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

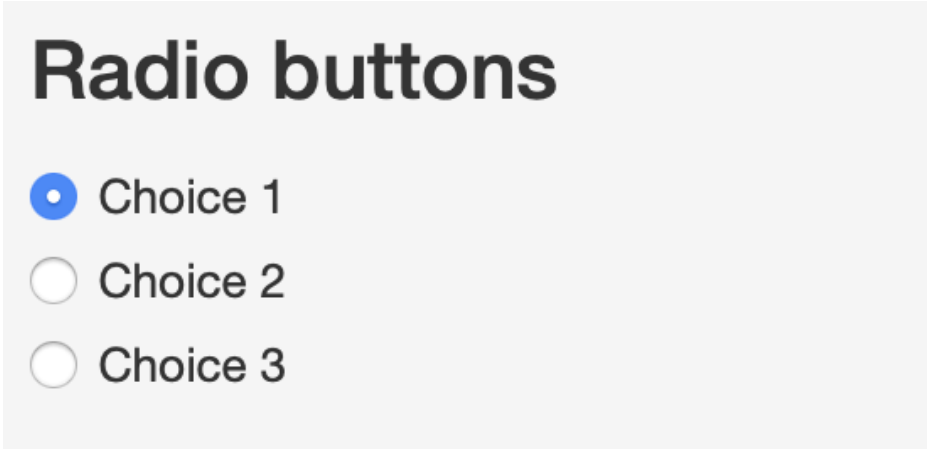
Input widgets: text input

```
fluidPage(

  textInput(                  # create text input element
    "text",                   # give a name to this element
    label = "My first text input",    # define the title of the element to be displayed
    value = "Enter text..."  # specify the default value
    )

)
```

- Use the files from `9-text input` app

Input widgets: radio button

- **What it looks like**

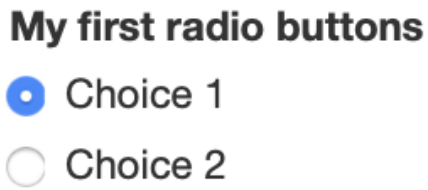


- **What it does**
  - Lets the user select one option out of a group of options
- **When it is used**
  - Let user select the dataset to be displayed in a graph
  - Let user select the type of graph to be displayed
  - Let user select a color in a graph

Input widgets: radio button

```
fluidPage(

  radioButtons(               # create radioButton element
    "radio",                  # give a name to this element
    label = "My first radio buttons", # define the title of the element to be displayed
    choices = list("Choice 1" = 1, "Choice 2" = 2), # specify the list of possible choices
    selected = 1              # specify the default choice to be selected
    )

)
```

- Use the files from `10-radio buttons` app



Knowledge check 2

Exercise 2

Module completion checklist

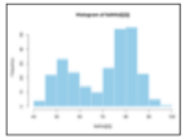
Objective	Complete
Explain the basics of RShiny	✔
Describe RShiny package and best practices for code modularity	✔
Set up your own application	✔
Customize your application with input widgets	✔
Customize your application with output formats	
Incorporate HTML tags to shiny dashboard	
Display images and pdf files on the dashboard	

Output: RShiny can display various 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)

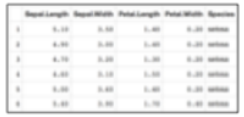


renderPlot(expr, width, height, res, …, env, quoted, func)

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)

foo

renderText(expr, env, quoted, func)

textOutput(outputId, container, inline)

renderUI(expr, env, quoted, func)

uiOutput(outputId, inline, container, …)
& **htmlOutput**(outputId, inline, container, …)

Output: We are exploring 4 output formats today

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

Output: text output

- **What it looks like**

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

Output: text output

- `ui.R`

```
fluidPage(
  # Create text output element that will display
output contents of my_text element.
  textOutput("my_text")

)
```

- `server.R`

```
function(input, output) {

  # Assign text to output contents of my_text
element.
  output$my_text <- renderText({
    # Define text to be assigned to my_text
element.
    "My first text output"
  })

}
```

- Use the files from `11-text output` app

Output: table output

- **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 nicely formatted table of data
- **When it is used**
  - To display your data set in table format

Output: table output

- `ui.R`

```
fluidPage(

  tableOutput("my_table") #<- create text output element that will display output contents of my_text
element

)
```

- `server.R`

```
function(input, output) {

  output$my_table <- renderTable({ #<- assign table to output contents of my_table element
    CMP                            #<- define table to be assigned to output my_table element
  })

}
```

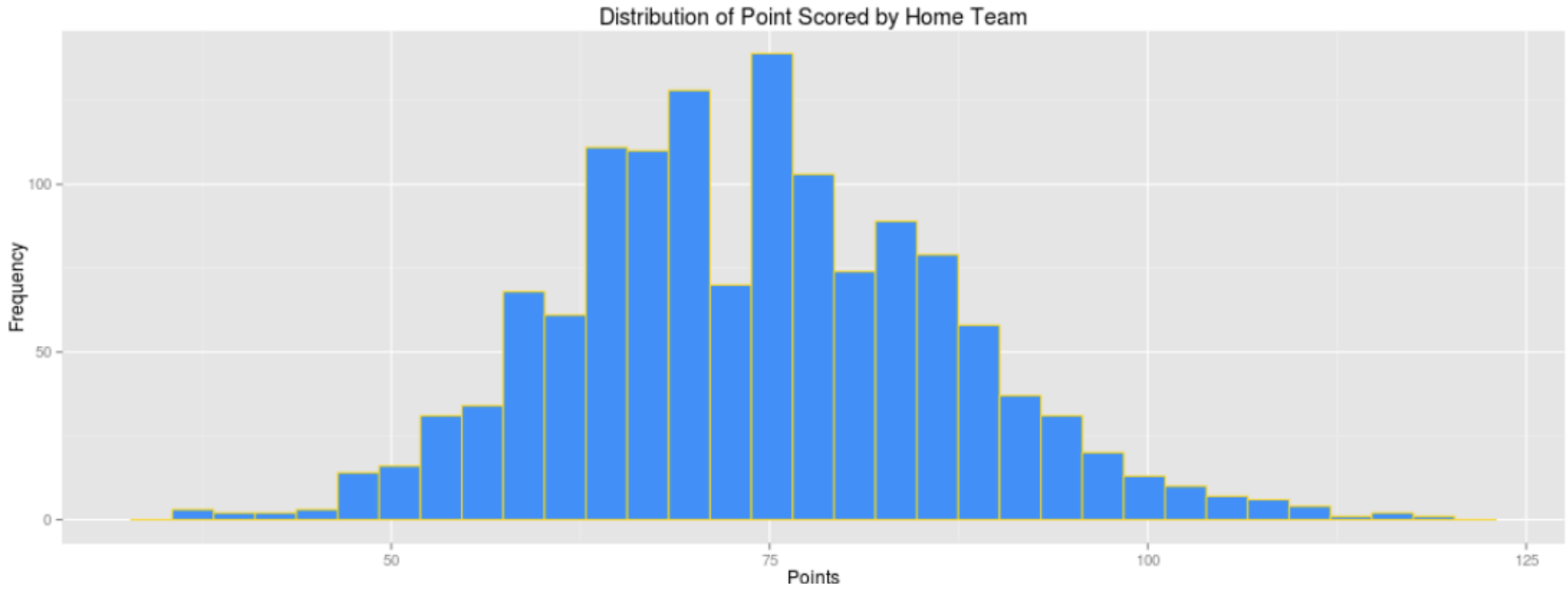
Output: table output

- Use the files from `12- table output` app

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

Output: plot output

- **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

Output: plot output

- `ui.R`

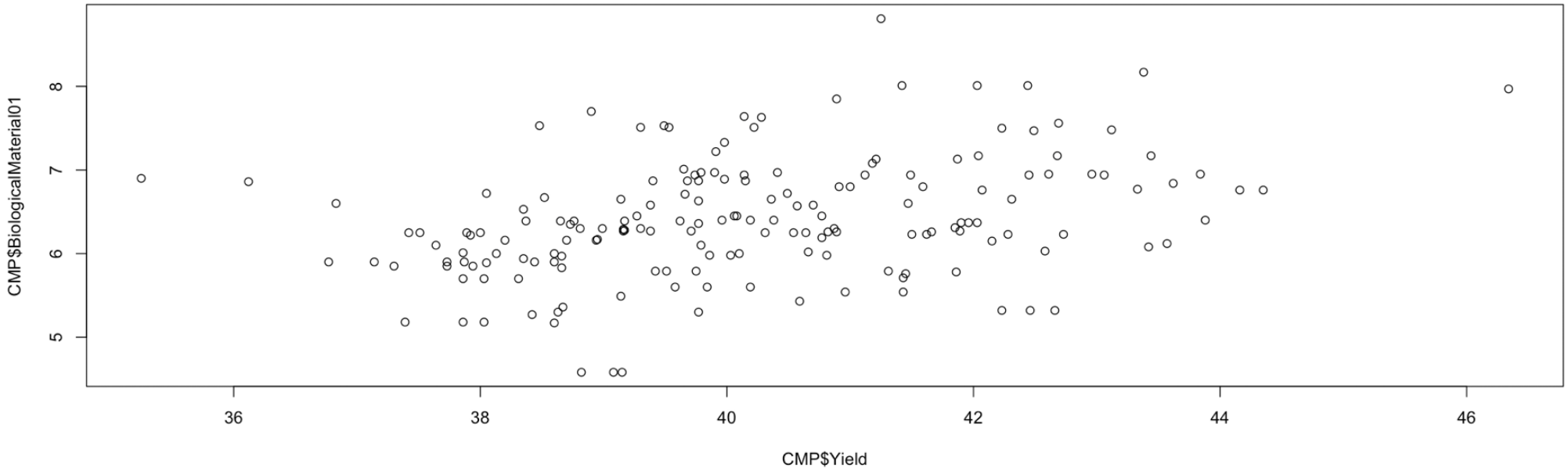
```
fluidPage(

  plotOutput("my_plot") #<- create plot output element that will display output contents of my_plot
element

)
```

- `server.R`

```
function(input, output) {

  output$my_plot <- renderPlot({                #<- assign plot to output contents of my_plot element
      plot(CMP$Yield, CMP$BiologicalMaterial01) #<- define plot to be assigned to output my_table
element
  })

}
```

Output: plot output

- Use the files from `13-plot output` app



Output: print output

- **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(CMP)`
  - `summary(CMP)`
  - `dim(CMP)`

Output: print output

- `ui.R`

```
fluidPage(

  verbatimTextOutput("my_print") #<- create print output element that will display output contents of my_print element

)
```

- `server.R`

```
function(input, output) {

  output$my_print <- renderPrint({   #<- assign print object to output contents of my_print element
      print(str(CMP[,1:20]))         #<- define print object to be assigned to output my_table element
  })

}
```

Output: print output

- Use the files from `14-print output` app

```
'data.frame':   176 obs. of  20 variables:
 $ Yield                 : num  38 42.4 42 41.4 42.5 ...
 $ BiologicalMaterial01  : num  6.25 8.01 8.01 8.01 7.47 6.12 7.48 6.94 6.94 6.94 ...
 $ BiologicalMaterial02  : num  49.6 61 61 61 63.3 ...
 $ BiologicalMaterial03  : num  57 67.5 67.5 67.5 72.2 ...
 $ BiologicalMaterial04  : num  12.7 14.7 14.7 14.7 14 ...
 $ BiologicalMaterial05  : num  19.5 19.4 19.4 19.4 17.9 ...
 $ BiologicalMaterial06  : num  43.7 53.1 53.1 53.1 54.7 ...
 $ BiologicalMaterial07  : num  100 100 100 100 100 100 100 100 100 100 ...
 $ BiologicalMaterial08  : num  16.7 19 19 19 18.2 ...
 $ BiologicalMaterial09  : num  11.4 12.6 12.6 12.6 12.8 ...
 $ BiologicalMaterial10  : num  3.46 3.46 3.46 3.46 3.05 3.78 3.04 3.85 3.85 3.85 ...
 $ BiologicalMaterial11  : num  138 154 154 154 148 ...
 $ BiologicalMaterial12  : num  18.8 21.1 21.1 21.1 21.1 ...
 $ ManufacturingProcess01: num  NA 0 0 0 10.7 12 11.5 12 12 12 ...
 $ ManufacturingProcess02: num  NA 0 0 0 0 0 0 0 0 0 ...
 $ ManufacturingProcess03: num  NA NA NA NA NA NA 1.56 1.55 1.56 1.55 ...
 $ ManufacturingProcess04: int  NA 917 912 911 918 924 933 929 928 938 ...
 $ ManufacturingProcess05: num  NA 1032 1004 1015 1028 ...
 $ ManufacturingProcess06: num  NA 210 207 213 206 ...
 $ ManufacturingProcess07: int  NA 177 178 177 178 178 177 178 177 177 ...
NULL
```

Knowledge check 3

Exercise 3

Module completion checklist

Objective	Complete
Explain the basics of RShiny	✔
Describe RShiny package and best practices for code modularity	✔
Set up your own application	✔
Customize your application with input widgets	✔
Customize your application with output formats	✔
Incorporate HTML tags to shiny dashboard	
Display images and pdf files on the dashboard	

HTML tags

- R Shiny can use HTML tags in the UI for prettier display
- Any HTML tag can be used which includes 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"
```

App with html tags

- While understanding the usage of all HTML tags is beyond the scope of this module, we can create a simple app which contains:

Action	HTML tag
header titles	h1 to h6
A color palette whose style can be adjusted using HTML code	Use div and style
A HTML break tag to give space in the UI of the app	br
A hyperlink embedded in R Shiny app using the HTML tag	a and href

- All tags can be accessed by two ways:
- Referring the HTML tags by tags variable

```
tags$tag_name
```

- Directly add the HTML code

App with html tags in UI

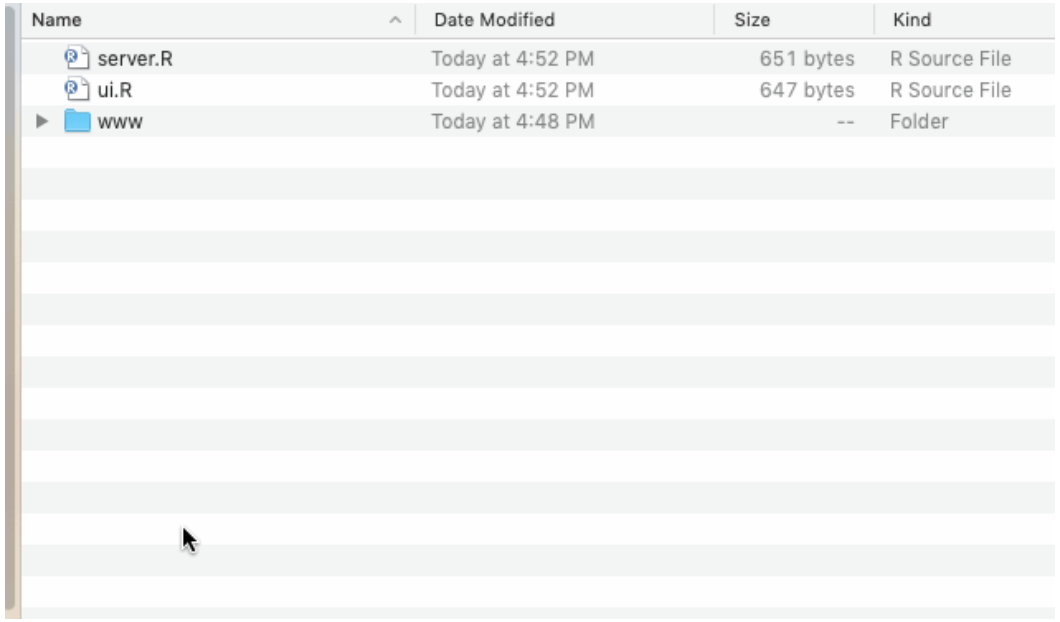
```
fluidPage(
  tags$h1("Display HTML tags"),    #<- header tag

  tags$h3("View action button"),  #<- header tag

  actionButton("action", label = "Action", width = "300px"), #<- action button

  tags$h3("View color input"), #<- header tag

  # HTML tag div specifies style with vertical display aligning at top with width 200 pixels.
  # Color input is another input widget which displays the color palette.
  div(style="display: vertical-align:top; width: 200px;",
      colourInput("box_fillcol", "Select colour to fill", "orangered")),

  tags$br(), #<- break tag to create the space

  h1("View other apps"), #<- header tag

  # Paragraph tag.
  p(style = "font-family:Times new roman", "See other apps in the",

  # a and href to display the hyperlink.
    a('Shiny Showcase', href = "https://www.rstudio.com/products/shiny/shiny-user-showcase/"))
)
```

App with html tags

- Use the code from `15-html_ui` app
- The server does not contain any code considering we do not have any operations there

Embed files in the app

- We can also embed files from our local system to the R Shiny app
- We should create a folder named `www` inside our shiny app folder along with `server` and `ui`
- We will embed a `pdf` and an `image` to our final app of the day
- Pdf is the shiny cheatsheet and image is the RStudio image
- Both the images are present inside `16-embed_files` app's `www` subfolder which is our app folder for this example



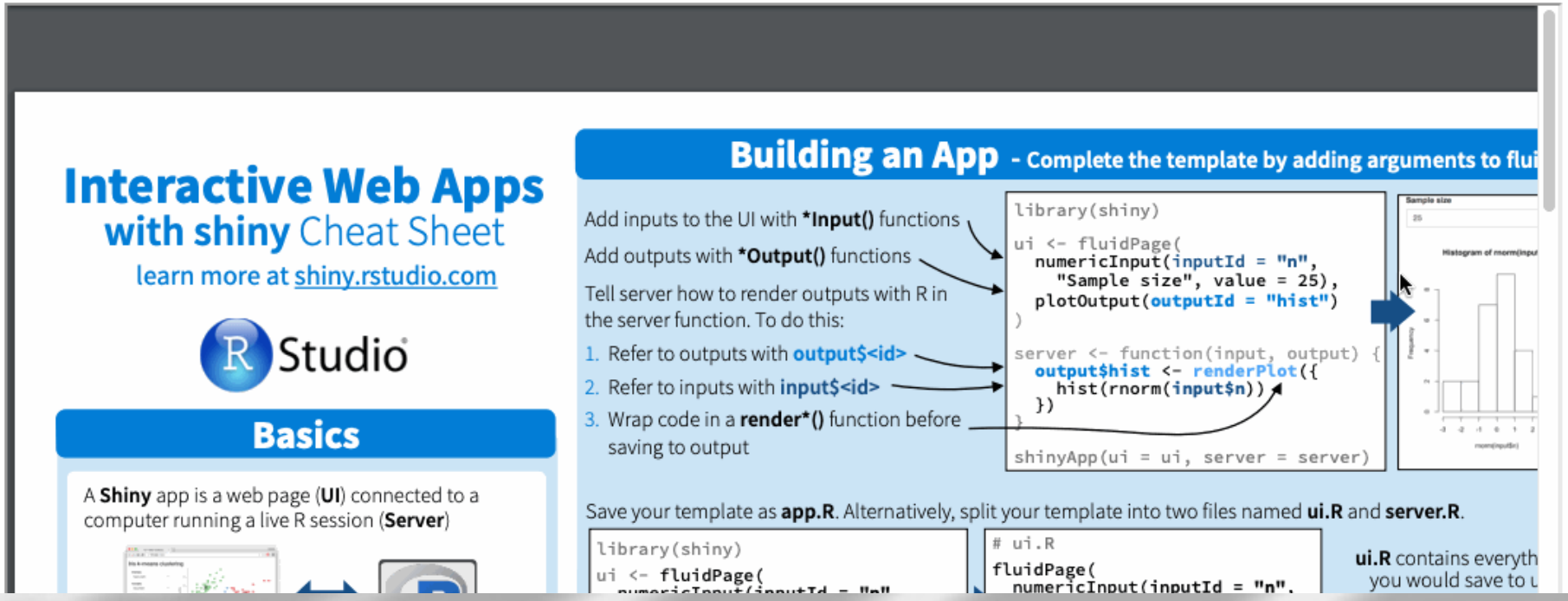
Embed files in the app

```
### Create Shiny application ###
fluidPage(
  tags$h1("Display pdf and images"), #<- html header tag

  img(src='r-studio-image.png'),    #<- embed image

  tags$br(),                        #<- html break tag
  tags$br(),                        #<- html break tag
  tags$br(),                        #<- html break tag

  uiOutput("pdf_view")              #<- embed pdf
)
```

```
function(input, output) {

  output$pdf_view <- renderUI({

    # Embed the pdf as an iframe.
    tags$iframe(style="height:600px; width:100%",  #<- style specifying height and width of the iframe
                src="cheatsheet.pdf") #<- source of the pdf
  })

}
```

Embed files in the app

Display pdf and images



Knowledge check 4

Exercise 4

Module completion checklist

Objective	Complete
Explain the basics of RShiny	✔
Describe RShiny package and best practices for code modularity	✔
Set up your own application	✔
Customize your application with input widgets	✔
Customize your application with output formats	✔
Incorporate HTML tags to shiny dashboard	✔
Display images and pdf files on the dashboard	✔

Workshop!

- **Remember**
  - Workshops are to be completed in the afternoon either with a dataset for a capstone project or with another dataset of your choosing
  - Make sure to annotate your code so that it is easy for others to understand what you are doing
  - This is an exploratory exercise to get you comfortable with the content we discussed today
- Today, you will:
  - come up with at least three use cases on how you can use RShiny in your work
  - use input widgets to customize the UI of your application
  - use different output formats to customize the output displayed by your application

This completes our module

Congratulations!