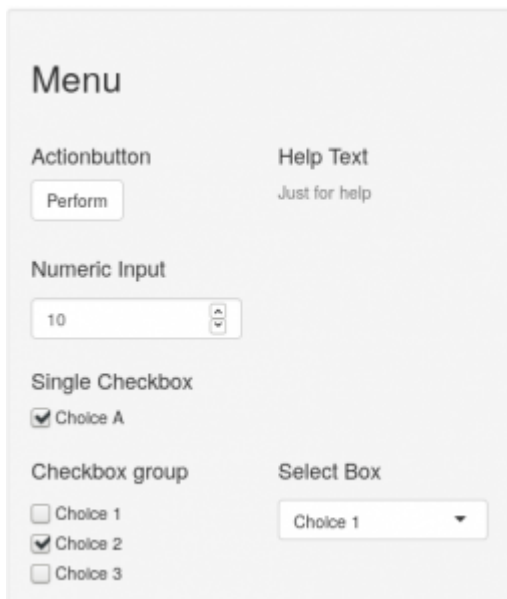


Building Shiny App Exercises (part 5)



RENDER FUNCTIONS

In the [fourth](#) part of our series we just “scratched the surface” of reactivity by analyzing some of the properties of the `renderTable` function.

Now it is time to get deeper and learn how to use the rest of the render functions that shiny provides. As you were told in [part 4](#) these are:

`renderImage`
`renderPlot`
`renderPrint`
`renderText`
`renderUI`

Below you will see the functionality of three of them (`renderImage`, `renderText` and `renderPrint`) and then we will be ready to use those of them that match our needs in the next parts, just like the widgets and give a specific form to our application. As you will probably understand, when reading this part our aim is to perform several statistical analyses on our dataset. We will start by creating a K-Means `tabPanel`.

Follow the examples to understand the logic of the tools you are going to use and then enhance the app you started creating in [part 1](#) by practising with the exercise set we prepared for you. Lets begin!

Answers to the exercises are available [here](#).

If you obtained a different (correct) answer than those listed on the solutions page, please feel free to post your answer as a comment on that page.

renderImage

Sending pre-rendered images with `renderImage`.

These are images saved as a link to a source file. If your Shiny app has pre-rendered images saved in a subdirectory, you can send them using `renderImage`. Suppose the images are in the subdirectory `"www"/`, and are named `"image1.png"`, `"image2.png"`, and so on. The following code would send the appropriate image, depending on the value of `input$n`:

```
# ui.R
library(shiny)
shinyUI(fluidPage(
  titlePanel("RenderImage"),
  sidebarLayout(
    sidebarPanel(
      radioButtons("n", label = h4("Radio Buttons"),
        choices = list("Choice 1" = 1, "Choice 2" = 2),
        selected = 2)
    ),
    mainPanel(
      imageOutput("Image")
    )
  )
))

#server.R

shinyServer(function(input, output, session) {
  # Send a pre-rendered image, and don't delete the image after
  # sending it
  output$Image <- renderImage({
    # When input$n is 3, filename is ./images/image3.jpeg
    filename <- normalizePath(file.path('./www',
      paste('image', input$n, '.png', sep='')))
  })
})
```

```
# Return a list containing the filename
list(src = filename)

}, deleteFile = FALSE)
})
```

Now let's break down what the code above exactly does. First of all as we saw in [part 1](#) you should save your images in a subdirectory called "www" inside the directory that you work. Let's say you save 2 images and you name them "image1" and "image2". As you can see we use `radioButtons` here to select which one of the two we want to be displayed. The filename contains the exact output path of the images while the list contains the filename along with some other values. In this example, `deleteFile` is `FALSE` because we don't want Shiny to delete an image after sending it.



Learn more about Shiny in the online course [R Shiny Interactive Web Apps – Next Level Data Visualization](#). In this course you will learn how to create advanced Shiny web apps; embed video, pdfs and images; add focus and zooming tools; and many other functionalities (30 lectures, 3hrs.).

Exercise 1

Place a `tabPanel` in the `tabsetPanel` of your Shiny App. Name it "K Means".

Exercise 2

Move the `radioButtons` from the `sidebarPanel` inside the `tabPanel` "K Means" you just created and name it "Select Image". Also, move the `submitButton` from the `sidebarPanel` to the `tabPanel` "K Means" without title.

Exercise 3

Place an `imageOutput` inside the `tabPanel` "K Means" with name "Image" (`ui.R`) and the reactive function of it (`server.R`).

Still nothing happens. HINT: Use `renderImage`.

Create a subdirectory inside the directory you work and name it "images". Put there two images with names "pic1" and "pic2" respectively and .png ending.

Exercise 4

Now create the filename. Follow the example above to create the right path. Do not forget to connect it with the `radioButtons`. Two steps left.

Exercise 5

Now it is time to set `deleteFile = "FALSE"`.

Exercise 6

Create the list that contains the filename.

Exercise 7

Set `width = 300` and `height = 200` into the list.

renderText-renderPrint

The example below shows how the `renderText` works.

```
#ui.R
library(shiny)
shinyUI(fluidPage(
  titlePanel("RenderImage"),
  sidebarLayout(
    sidebarPanel(
      sliderInput("slider1", label = h4("Sliders"),
        min = 3 , max = 10, value = 3)
    ),
    mainPanel(
      textOutput("text1")
    )
  )
)
```

```
))  
#server.R  
shinyServer(function(input, output, session) {  
  
  output$text1 <- renderText({  
    paste("You have selected", input$slider1,"clusters")  
  })  
})
```

The code above takes a numeric value from the sliderInput and puts it in the exact place of our sentence in the mainPanel.

Before proceeding to the next exercise move the sliderInput from the sidebarPanel just after the imageOutput in the tabPanel "K Means". Then change its name to "Clusters", its min to 3, its max to 10 and value to 3.

Exercise 8

Put the textOutput named "text1" inside your tabPanel exactly after the sliderInput, then place its reactive function inside server.R using renderText.

Exercise 9

Display the reactive output by putting inside the renderText function the sentence "You have selected",(?),"clusters." HINT : Use paste.

Exercise 10

Follow exactly the same steps but this time instead of renderText use renderPrint and note the difference.