Building Shiny App exercises part 4

APPLICATION LAYOUT & REACTIVITY
The fourth part of our series is "separated" into two "sub-parts". In the first one we will start building the skeleton of our application by using tabsetPanel. This is how we will separate the sections of our app and also organize its structure better.
In the second part you will learn hot to load your dataset in RStudio and finally in the third one we will give life to your Shiny App! Specifically, you are going to have your first contact with reactivity and learn how to build reactive output to display in your Shiny app in a form of a data table initially.
Follow the examples below 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.

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.).
TABSET PANEL
In the example below you will see hot to add a tabsetPanel in your shiny app.

```
# ui.R
library(shiny)
fluidPage(
titlePanel("TabPanel"),
sidebarLayout(
sidebarPanel(h3("Menu")),
mainPanel(h3("Main Panel"),tabsetPanel(type = "tabs"))
))
```

```
#server.R
library(shiny)
shinyServer(function(input, output) {
})
```

Exercise 1

Add a tabsetPanel to the mainPanel of your Shiny App.

TAB PANEL
In the example below you will see how to add tabPanel in your tabsetPanel.

```
# ui.R
library(shiny)
fluidPage(
titlePanel("TabPanel"),
sidebarLayout(
sidebarPanel(h3("Menu")),
mainPanel(h3("Main Panel"),tabsetPanel(type = "tabs",
tabPanel("Tab Panel 1")
))))
```

```
#server.R
library(shiny)
shinyServer(function(input, output) {
})
```

Exercise 2

Place a tabPanel in the tabsetPanel you just added to your Shiny App. Name it “Data Table”.

Exercise 3

Put a second tabPanel next to the first one. Name it “Summary”.

LOAD DATASET
Now it is time to give your app a purpose of existence. This can happen with only one way. To add some data into it! As we told in part 1 we will create an application based on the famous (Fisher’s or Anderson’s) iris data set which gives the measurements in centimeters of the variables sepal length and width and petal length and width, respectively, for 50 flowers from each of 3 species of iris. The species are Iris setosa, versicolor, and virginica.
This is a “built in” dataset of RStudio that will help us create our first application.
But normally you want to analyze your own datasets. The first thing you should do in order to perform any kind of analysis on them is to load them properly. We will now see how to load a dataset in txt form from a local file using RStudio. Let’s go!
The process is quite simple. First of all you have to place the txt file that contains your dataset into the same directory that you are working, secondly press the “Import Dataset” button in RStudio and then “From Local File…”. Find the txt file in your computer and click “Open”, then press “Import”. That’s all! Your dataset is properly loaded in your directory and now you can work with it.
Please note that the purpose of this series is not to teach how to form your dataset before you load it nor how to “clean” it. You can gain more information about this subject from here. Our purpose is to teach you build your first Shiny App.

Exercise 4

Load the dataset you want to analyze ("something.txt") from your computer to your directory with RStudio buttons.

INTRODUCTION TO REACTIVITY
From this point you are going to enter in the "Kingdom of Reactivity". Reactive output automatically responds when your user interacts with a widget. You can create reactive output by following two steps. Firstly, add an R object to your ui.R. and then tell Shiny how to build the object in server.R.

1): Add an R object to the UI

Shiny provides a variety of functions that transform R objects into output for your UI as you can see below:

htmlOutput: raw HTML
imageOutput: image
plotOutput: plot
tableOutput: table
textOutput: text
uiOutput: raw HTML
verbatimTextOutput: text

To add output to the UI place the output function inside sidebarPanel or mainPanel in the ui.R script.
For example, the ui.R file below uses tableOutput to add a reactive line of text to "Tab Panel 1". Nothing happens…for the moment!

```
# ui.R
library(shiny)
fluidPage(
titlePanel("TabPanel"),
sidebarLayout(
sidebarPanel(h3("Menu")),
mainPanel(h3("Main Panel"),tabsetPanel(type = "tabs",
tabPanel("Tab Panel 1",dataTableOutput("dt1")),
tabPanel("Tab Panel 2"))
)))
```

```
#server.R
library(shiny)
shinyServer(function(input, output) {
})
```

Notice that datatableOutput takes an argument, the character string “dt1”. Each of the *Output functions require a character string that Shiny will use as the name of your reactive element. Users cannot see it and you will understand its role later.

Exercise 5

Add a datatableOutput to “Data Table”, name its argument “Table”.

2): Provide R code to build the object.

The code should be placed in the function that appears inside shinyServer in your server.R script.
This function is of great importance as it builds a list-like object named output that contains all of the code needed to update the R objects in your app. Be careful, each R object MUST have its own entry in the list.
You can create an entry by defining a new element for output within the function. The element name should match the name of the reactive element that you created in ui.R.
Each entry should contain the output of one of Shiny’s render* functions. Each render* function corresponds to a specific type of reactive object. You can find them below:

renderImage: images (saved as a link to a source file)
renderPlot: plots
renderPrint: any printed output
renderTable: data frame, matrix, other table like structures
renderText: character strings
renderUI: a Shiny tag object or HTML

Each render* function takes a single argument: an R expression

which can either be one simple line of text, or it can involve many lines of code.
In the example below “dt1” is attached to the output expression in server.R and gives us the Data Table of “iris” dataset inside “Tab Panel 1”.

```
# ui.R
library(shiny)
fluidPage(
titlePanel("TabPanel"),
sidebarLayout(
sidebarPanel(h3("Menu")),
mainPanel(h3("Main Panel"),tabsetPanel(type = "tabs",
tabPanel("Tab Panel 1",dataTableOutput("dt1")),
tabPanel("Tab Panel 2"))
)))
```

```
#server.R
library(shiny)
shinyServer(function(input, output) {
output$dt1 <- renderDataTable(
iris)

})
```

Exercise 6

Add the appropriate render* function to server.R in order to create the Data table of the “iris” dataset. Hint: Use the output expression.

The dataTableOutput is your first contact with reactivity, in the next parts of our series you will use the rest of the Output functions that Shiny provides. But for now let’s experiment a little bit on this.
As you can see there is a text filter in your Data Table. You can deactivate it by setting searching to be “FALSE” as the example below.

```
# ui.R
library(shiny)
fluidPage(
titlePanel("TabPanel"),
sidebarLayout(
sidebarPanel(h3("Menu")),
mainPanel(h3("Main Panel"),tabsetPanel(type = "tabs",
tabPanel("Tab Panel 1",dataTableOutput("dt1")),
tabPanel("Tab Panel 2"))
)))
```

```
#server.R
library(shiny)
shinyServer(function(input, output) {
output$dt1 <- renderDataTable(
iris,options = list(searching=FALSE))
})
```

Exercise 7

Disable the Text Filter of your Data Table. Hint: Use options, list and searching.

With the same logic you can disable the pagination that is displayed in your Data Table, as in the example below.

```
# ui.R

library(shiny)
fluidPage(
titlePanel("TabPanel"),
sidebarLayout(
sidebarPanel(h3("Menu")),
mainPanel(h3("Main Panel"),tabsetPanel(type = "tabs",
tabPanel("Tab Panel 1",dataTableOutput("dt1")),
tabPanel("Tab Panel 2"))
)))
```

```
#server.R
```

```
library(shiny)
shinyServer(function(input, output) {
output$dt1 <- renderDataTable(
iris,options = list(searching=FALSE,paging=FALSE))
})
```

Exercise 8

Disable the Pagination of your Data Table. Hint: Use options, list, paging.
Now you can see how to display an exact number of rows (15) and enable filtering again.

```
# ui.R

library(shiny)
fluidPage(
titlePanel("TabPanel"),
sidebarLayout(
sidebarPanel(h3("Menu")),
mainPanel(h3("Main Panel"),tabsetPanel(type = "tabs",
tabPanel("Tab Panel 1",dataTableOutput("dt1")),
tabPanel("Tab Panel 2"))
)))

#server.R
library(shiny)
shinyServer(function(input, output) {
output$dt1 <- renderDataTable(
iris,options = list(pageLength=15))
})
```

Exercise 9

Enable filtering again and set the exact number of rows that are displayed to 10. Hint: Use options, list, pageLength

We can also create a Length Menu in order to control totally the choices of the numbers of rows we want to be displayed. In the example below we assign every number to a menu label. 5 -> ‘5’, 10 -> ’10’, 15 -> ’15’,-1 -> ‘ALL’.

```
# ui.R
library(shiny)
fluidPage(
titlePanel("TabPanel"),
sidebarLayout(
sidebarPanel(h3("Menu")),
mainPanel(h3("Main Panel"),tabsetPanel(type = "tabs",
tabPanel("Tab Panel 1",dataTableOutput("dt1")),
tabPanel("Tab Panel 2"))
)))
```

```
#server.R
library(shiny)
shinyServer(function(input, output) {
output$dt1 <- renderDataTable(
iris,options = list(
lengthMenu = list(c(5, 15, 25,-1),c('5','15','25','ALL')),
pageLength = 15))
})
```

Exercise 10

Create a Length Menu with values (10,20,30,-1) and assign each one ot the values to the appropriate menu label. Hint: Use options, list, lengthMenu,pageLength.