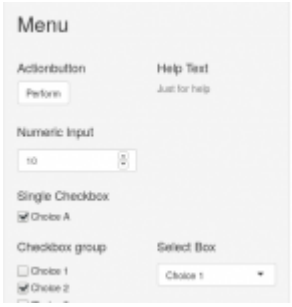
Building Shiny App exercises part 10



SHINY DASHBOARD STRUCTURE & APPEARANCE

Finally we reached in the final part of our series. At this part we will see how to improve the structure and the appearance of our dashboard even more, according to our preferences and of course make it more attractive to the user. Last but not least we will see the simplest and easiest way to deploy it.

Read the examples below to understand the logic of what we are going to do and then test yous skills with the exercise set we prepared for you. Lets begin!

Answers to the exercises are available here.

infoBox

There is a special kind of box that is used for displaying simple numeric or text values, with an icon.The example code below shows how to generate infoBox. The first row of infoBox uses the default setting of fill=FALSE.

Since the content of an infoBox will usually be dynamic, shinydashboard contains the helper functions infoBoxOutput and renderInfoBox for dynamic content.

```
#ui.R
library(shinydashboard)
```

```
dashboardPage(
dashboardHeader(title = "Info boxes"),
dashboardSidebar(),
dashboardBody(
# infoBoxes with fill=FALSE
fluidRow(
# A static infoBox
infoBox("New Orders", 10 * 2, icon = icon("credit-card")),
# Dynamic infoBoxes
infoBoxOutput("progressBox"),
infoBoxOutput("approvalBox")
)))
#server.R
shinyServer(function(input, output) {
output$progressBox <- renderInfoBox({
infoBox(
"Progress", paste0(25 + input$count, "%"), icon =
icon("list"),
color = "purple"
)
})
output$approvalBox <- renderInfoBox({
infoBox(
"Approval", "80%", icon = icon("thumbs-up", lib =
"glyphicon"),
color = "yellow"
)
})
})
```

Exercise 1

Create three infoBox with information icons and color of your choice and put them in the tabItem “dt” under the “DATA-TABLE”.

To fill them with a color follow the example below:
#ui.R

```
library(shinydashboard)

dashboardPage(
dashboardHeader(title = "Info boxes"),
dashboardSidebar(),
dashboardBody(
# infoBoxes with fill=FALSE
fluidRow(
# A static infoBox
infoBox("New Orders", 10 * 2, icon = icon("credit-card"),fill
= TRUE),
# Dynamic infoBoxes
infoBoxOutput("progressBox"),
infoBoxOutput("approvalBox")
)))
#server.R
shinyServer(function(input, output) {
output$progressBox <- renderInfoBox({
infoBox(
"Progress", paste0(25 + input$count, "%"), icon =
icon("list"),
color = "purple",fill = TRUE
)
})
output$approvalBox <- renderInfoBox({
infoBox(
"Approval", "80%", icon = icon("thumbs-up", lib =
"glyphicon"),
color = "yellow",fill = TRUE
)
})
})
```

Exercise 2

Fill the infoBox with the color you selected in Exercise 1. HINT: Use fill.

Exercise 3

Now enhance the appearance of your tabItem named “km” by setting height = 450 in the four box you have there.

Skins

There are a number of color themes, or skins. The default is blue, but there are also black, purple, green, red, and yellow. You can choose which theme to use with dashboardPage(skin = "blue"), dashboardPage(skin = "black"), and so on.

Exercise 4

Change skin from blue to red.

CSS

You can add custom CSS to your app by adding code in the UI of your app like this:

```
#ui.R
dashboardPage(
dashboardHeader(title = "Custom font"),
dashboardSidebar(),
dashboardBody(
tags$head(tags$style(HTML('
.main-header .logo {

font-weight: bold;
font-size: 24px;
}
')))
)
)
```

Exercise 5

Change the font of your dashboard title by adding CSS code.

Long titles

In some cases, the title that you wish to use won’t fit in the default width in the header bar. You can make the space for the title wider with the titleWidth option. In this example, we’ve increased the width for the title to 450 pixels.

```
#ui.R
dashboardPage(
dashboardHeader(
title = "Example of a long title that needs more space",
titleWidth = 450
),
dashboardSidebar(),
dashboardBody(

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

Exercise 6

Set your titlewidth to “400” and then set it to the default value again.

Sidebar width

To change the width of the sidebar, you can use the width option. This example has a wider title and sidebar:

```
#ui.R
library(shinydashboard)
dashboardPage(
dashboardHeader(
title = "Title and sidebar 350 pixels wide",
titleWidth = 350
),
dashboardSidebar(
width = 350,
```

```
sidebarMenu(
menuItem("Menu Item")
)
),
dashboardBody()
)
#server.R
function(input, output) { }
```

Exercise 7

Set sidebar width to “400” and then return to the default one.

Icons

Icons are used liberally in shinydashboard. The icons used in shiny and shinydashboard are really just characters from special font sets, and they’re created with Shiny’s icon() function.

To create a calendar icon, you’d call:
`icon("calendar")`

The icons are from Font-Awesome and Glyphicons. You can see lists of all available icons here:

http://fontawesome.io/icons/
http://getbootstrap.com/components/#glyphicons

Exercise 8

Change the icon of the three menuItem of your dashboard. Select whatever you like from the two lists above.

Statuses and colors

Many shinydashboard components have a status or color argument.

The status is a property of some Bootstrap classes. It can have values like status="primary", status="success", and

others.

The color argument is more straightforward. It can have values like color="red", color="black", and others.

The valid statuses and colors are also listed in ?validStatuses and ?validColors.

Exercise 9

Change the status of the three widget box in the tabItem named “km” to “info”, “success” and “danger” respectively.

Shinyapps.io

The easiest way to turn your Shiny app into a web page is to use shinyapps.io, RStudio’s hosting service for Shiny apps.

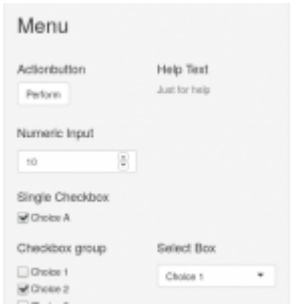
shinyapps.io lets you upload your app straight from your R session to a server hosted by RStudio. You have complete control over your app including server administration tools.

First of all you have to create an account in shinyapps.io.

Exercise 10

Publish your app through Shinyapps.io

Building Shiny App Exercises (part-9)



Shiny Dashboard Overview

In this part we will “dig deeper” to discover the amazing capabilities that a Shiny Dasboard provides.
Read the examples below to understand the logic of what we are going to do and then test yous skills with the exercise set we prepared for you. Lets begin!

Answers to the exercises are available here.

The dashboardPage function expects three components: a header, sidebar, and body:

```
#ui.R
dashboardPage(
dashboardHeader(),
dashboardSidebar(),
dashboardBody()
)
```

For more complicated apps, splitting app into pieces can make it more readable:

```
header <- dashboardHeader()

sidebar <- dashboardSidebar()

body <- dashboardBody()

dashboardPage(header, sidebar, body)
```

Now we’ll look at each of the three main components of a shinydashboard.

HEADER

A header can have a title and dropdown menus. The dropdown menus are generated by the dropdownMenu function. There are three types of menus – messages, notifications, and tasks – and each one must be populated with a corresponding type of item.

Message menus

A messageItem contained in a message menu needs values for from and message. You can also control the icon and a notification time string. By default, the icon is a silhouette of a person. The time string can be any text. For example, it could be a relative date/time like “5 minutes”, “today”, or “12:30pm yesterday”, or an absolute time, like “2014-12-01 13:45”.

```
#ui.R
dropdownMenu(type = "messages",
messageItem(
from = "Sales Dept",
message = "Sales are steady this month."
),
messageItem(
from = "New User",
message = "How do I register?",
icon = icon("question"),
time = "13:45"
),
messageItem(
from = "Support",
message = "The new server is ready.",
icon = icon("life-ring"),
time = "2014-12-01"
)
)
```

Exercise 1

Create a dropdownMenu in your dashboardHeader as the example above. Put date, time and generally text of your choice.

Dynamic content

In most cases, you'll want to make the content dynamic. That means that the HTML content is generated on the server side and sent to the client for rendering. In the UI code, you'd use dropdownMenuOutput like this:

```
dashboardHeader(dropdownMenuOutput("messageMenu"))
```

Exercise 2

Replace dropdownMenu with dropdownMenuOutput and the three messageItem with messageMenu.

The next step is to create some messages for this example.The code below does this work for us.

```
# Example message data in a data frame
messageData <- data.frame(
from = c("Admininstrator", "New User", "Support"),
message = c(
"Sales are steady this month.",
"How do I register?",
"The new server is ready."
),
stringsAsFactors = FALSE
)
```

Exercise 3

Put messageData inside your server.r but outside of the shinyServer function.

And on the server side, you'd generate the entire menu in a renderMenu, like this:

```
output$messageMenu <- renderMenu({
# Code to generate each of the messageItems here, in a list.
messageData
```

```
# is a data frame with two columns, 'from' and 'message'.
# Also add on slider value to the message content, so that
messages update.
msgs <- apply(messageData, 1, function(row) {
messageItem(
from = row[["from"]],
message = paste(row[["message"]], input$slider)
)
})

dropdownMenu(type = "messages", .list = msgs)
})
```

Exercise 4

Put the code above(output$messageMenu) in the shinyServer of server.R.

Hopefully you have understood by now the logic behind the dynamic content of your Menu. Now let’s return to the static one in order to describe it a little bit more. So make the proper changes to your code in order to return exactly to the point we were after exercise 1.

Notification menus

A notificationItem contained in a notification contains a text notification. You can also control the icon and the status color. The code below gives an example.

```
#ui.r
dropdownMenu(type = "notifications",
notificationItem(
text = "20 new users today",
icon("users")
),
notificationItem(
text = "14 items delivered",
icon("truck"),
status = "success"
```

```
),
notificationItem(
text = "Server load at 84%",
icon = icon("exclamation-triangle"),
status = "warning"
)
)
```

Exercise 5

Create a dropdownMenu for your notifications like the example. Use text of your choice. Be careful of the type and the notificationItem.

Task menus

Task items have a progress bar and a text label. You can also specify the color of the bar. Valid colors are listed in ?validColors. Take a look at the example below.

```
#ui.r
dropdownMenu(type = "tasks", badgeStatus = "success",
taskItem(value = 90, color = "green",
"Documentation"
),
taskItem(value = 17, color = "aqua",
"Project X"
),
taskItem(value = 75, color = "yellow",
"Server deployment"
),
taskItem(value = 80, color = "red",
"Overall project"
)
)
```

Exercise 6

Create a dropdownMenu for your tasks like the example above. Use text of your choice and create as many taskItem as you

want. Be carefull of the type and the taskItem.

Disabling the header

If you don’t want to show a header bar, you can disable it with:

```
dashboardHeader(disable = TRUE)
```

Exercise 7

Disable the header.

Now enable it again.

Body

The body of a dashboard page can contain any regular Shiny content. However, if you’re creating a dashboard you’ll likely want to make something that’s more structured. The basic building block of most dashboards is a box. Boxes in turn can contain any content.

Boxes

Boxes are the main building blocks of dashboard pages. A basic box can be created with the box function, and the contents of the box can be (most) any Shiny UI content. We have already created some boxes in part 8 so lets enhance theis appearance a little bit.
Boxes can have titles and header bar colors with the title and status options. Look at the examples below.

```
box(title = "Histogram", status = "primary",solidHeader = TRUE, plotOutput("plot2", height = 250)),
```

```
box(
title = "Inputs", status = "warning",
"Box content here", br(), "More box content",
sliderInput("slider", "Slider input:", 1, 100, 50),
textInput("text", "Text input:")
```

)

Exercise 8

Give a title of your choice to all the box you have created in your dashboard except of the three widgets' box.

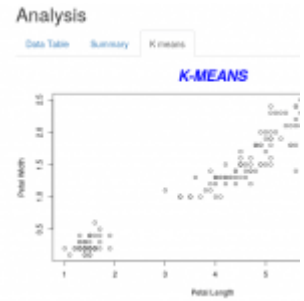
Exercise 9

Change the status of the first three box to "primary" and the last three to "warning".

Exercise 10

Transform the headers of your first three box to solid headers.

Building Shiny App Exercises (part-8)



Transform your App into Dashboard

Now that we covered the basic staff that you need to know in order to build your App it is time to enhance its appearance and its functionality. The interface is very important fot the user as it must not only be friendly but also easy to use.

At this part we will transform your Shiny App into a beautiful

Shiny Dashboard. Firstly we will create the interface and then step by step we will “move” the App you built in the previous parts into this. In part 8 we will move the app step by step into your dashboard and in the last two parts we will enhance its appearance even more and of course deploy it.

Read the examples below to understand the logic of what we are going to do and then test yous skills with the exercise set we prepared for you. Lets begin!

Answers to the exercises are available here.

INSTALLATION

The packages that we are going to use is shinydashboard and shiny . To install, run:

```
install.packages("shinydashboard")
install.packages("shiny")
```

Exercise 1

Install the package shinydashboard and the package shiny in your working directory.

BASICS

A dashboard has three parts: a header, a sidebar, and a body. Here’s the most minimal possible UI for a dashboard page.

```
## ui.R ##
library(shinydashboard)

dashboardPage(
```

```
dashboardHeader(),
dashboardSidebar(),
dashboardBody()
)
```

Exercise 2

Add a dashboardPage and then Header, Sidebar and Body into your UI. HINT: Use dashboardPage, dashboardHeader, dashboardSidebar, dashboardBody.

First of all we should name it with title like below:

```
## ui.R ##
library(shinydashboard)

dashboardPage(
dashboardHeader(title="Dashboard),
dashboardSidebar(),
dashboardBody()
)
```

Exercise 3

Name your dashboard “Shiny App”. HINT: Use title.

Next, we can add content to the sidebar. For this example we’ll add menu items that behave like tabs. These function similarly to Shiny’s tabPanels: when you click on one menu item, it shows a different set of content in the main body.

There are two parts that need to be done. First, you need to add menuItems to the sidebar, with appropriate tabNames.

```
## Sidebar content
dashboardSidebar(
sidebarMenu(
menuItem("Dashboard", tabName = "dashboard", icon = icon("dashboard")),
menuItem("Widgets", tabName = "widgets", icon = icon("th"))
)
```

```
)
```

Exercise 4

Create three menuItem, name them “DATA TABLE”, “SUMMARY” and “K-MEANS” respectively. Make sure to use distict tabName for each one of them. The icon is of your choise. HINT: Use menuItem, tabName and icon.

In the body, add tabItems with corresponding values for tabName:

```
## Body content
dashboardBody(
tabItems(
tabItem(tabName = "dashboard",
h2("Dashboard"),
fluidRow(
box()
)
),
tabItem(tabName = "widgets",
h2("WIDGETS")
),
)
)
```

Exercise 5

Add tabItems in dashboardBody. Be sure to give the same tabName to each one to get them linked with your menuItem. HINT: Use tabItems, tabItem, h2.

Obviously, this dashboard isn’t very useful. We’ll need to add components that actually do something. In the body we can add boxes that have content.

Firstly let’s create a box for our dataTable in the tabItem with tabName “dt”.

```
## Body content
```

```
dashboardBody(
tabItems(
tabItem(tabName = "dashboard",
h2("Dashboard"),
fluidRow(
box()
)
),
tabItem(tabName = "widgets",
h2("WIDGETS")
),
)
)
```

Exercise 6

Specify the fluidrow and create a box inside the “DATA TABLE” tabItem. HINT: Use fluidrow and box.

Exercise 7

Do the same for the other two tabItem. Create one fluidrow and one box in the “SUMMARY” and another fluidrow with four boxin the “K-MEANS”.

Now just copy and paste the code below, which you used in part 7 to move your dataTable inside the “DATA TABLE” tabItem.

```
#ui.R
dataTableOutput("Table"),width = 400
#server.R
output$Table <- renderDataTable(
iris,options = list(
lengthMenu = list(c(10, 20, 30,-1),c('10','20','30','ALL')),
pageLength = 10))
```

Exercise 8

Place the sample code above in the right place in order to add the dataTable “Table” inside the “DATA TABLE” tabItem.

Now just copy and paste the code below, which you used in part 7 to move the dataTable “Table2” inside the “SUMMARY” tabItem.

```
#ui.R
dataTableOutput("Table2"),width = 400

#server.R
sumiris<-as.data.frame.array(summary(iris))
output$Table2 <- renderDataTable(sumiris)
```

Exercise 9

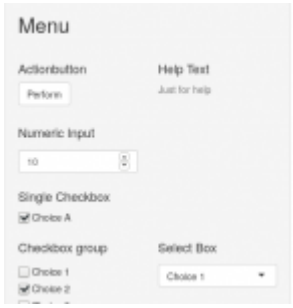
Place the sample code above in the right place in order to add the dataTable “Table2” inside the “SUMMARY” tabItem.

Do the same for the last exercise as you just have to put the code from part 7 inside the “K-MEANS” tabItem.

Exercise 10

Place the K-Means plot and the three widgets from part 7 inside the four box you created before.

Building Shiny App exercises part 7



Connect widgets & plots

In the seventh part of our journey we are ready to connect more of the widgets we created before with our k-means plot in order to totally control its output. Of cousre we will also reform the plot itself properly in order to make it a real k-means plot.
Read the examples below to understand the logic of what we are going to do and then test yous skills 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.

First of all let's move the widgets we are going to use from the sidebarPanel into the mainPanel and specifically under our plot.

Exercise 1

Remove the textInput from your server.R file. Then place the checkboxGroupInput and the selectInput in the same row with the sliderInput. Name them "Variable X" and "Variable Y" respectively. HINT: Use fluidrow and column.

Create a reactive expression

Reactive expressions are expressions that can read reactive values and call other reactive expressions. Whenever a reactive value changes, any reactive expressions that depended on it are marked as "invalidated" and will automatically re-execute if necessary. If a reactive expression is marked as

invalidated, any other reactive expressions that recently called it are also marked as invalidated. In this way, invalidations ripple through the expressions that depend on each other.
The reactive expression is activated like this: example <- reactive({ })

Exercise 2

Place a reactive expression in server.R, at any spot except inside output$All and name it “Data”. HINT: Use reactive

Connect your dataset’s variables with your widgets.

Now let’s connect your selectInput with the variables of your dataset as in the example below.

```
#ui.R
library(shiny)
shinyUI(fluidPage(
titlePanel("Shiny App"),

sidebarLayout(
sidebarPanel(h2(“Menu”),
selectInput(‘ycol’, ‘Y Variable’, names(iris)) ),
mainPanel(h1(“Main”)
)
)
))
#server.R
shinyServer(function(input, output) {
example <- reactive({
iris[, c(input$ycol)]
})
})
```

Exercise 3

Put the variables of the iris dataset as inputs in your

selectInput as “Variable Y” . HINT: Use names.

Exercise 4

Do the same for checkboxGroupInput and “Variable X”. HINT: Use names.

Select the fourth variabale as default like the example below.

```
#ui.R
library(shiny)
shinyUI(fluidPage(
titlePanel("Shiny App"),

sidebarLayout(
sidebarPanel(h2(“Menu”),
checkboxGroupInput(“xcol”, “Variable X”,names(iris),
selected=names(iris)[[4]]),
selectInput(“ycol”, “Y Variable”, names(iris),
selected=names(iris)[[4]])
),
mainPanel(h1(“Main”)
)
)
))
#server.R
shinyServer(function(input, output) {
example <- reactive({
iris[, c(input$xcol,input$ycol)
]
})
})
```

Exercise 5

Make the second variable the default choise for both widgets. HINT: Use selected.

Now follow the example below to create a new function and

place there the automated function for k means calculation.

```
#ui.R
library(shiny)
shinyUI(fluidPage(
titlePanel("Shiny App"),

sidebarLayout(
sidebarPanel(h2(“Menu”),
checkboxGroupInput(“xcol”, “Variable X”,names(iris),
selected=names(iris)[[4]]),
selectInput(“ycol”, “Y Variable”, names(iris),
selected=names(iris)[[4]])
),
mainPanel(h1(“Main”)
)
)
))
#server.R
shinyServer(function(input, output) {
example <- reactive({
iris[, c(input$xcol,input$ycol)
]
})
example2 <- reactive({
kmeans(example())
})
})
```

Exercise 6

Create the reactive function Clusters and put in there the function kmeans which will be applied on the function Data. HINT: Use reactive.

Connect your plot with the widgets.

It is time to connect your plot with the widgets.

Exercise 7

Put Data inside renderPlot as first argument replacing the data that you have chosen to be plotted until now. Moreover delete xlab and ylab.

Improve your k-means visualiztion.

You gan change automatically the colours of your clusters by copying and pasting this part of code as first argument of renderPlot before the plot function:

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

We will choose to have up to nine clusters so we choose nine colours.

Exercise 8

Set min of your sliderInput to 1, max to 9 and value to 4 and use the palette function to give colours.

This is how you can give different colors to your clusters. To activate these colors put this part of code into your plot function.

```
col = Clusters()$cluster,
```

Exercise 9

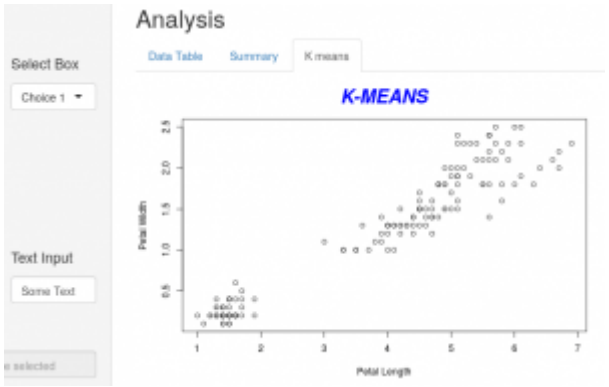
Activate the palette function.

To make your clusters easily foundable you can fully color them by adding into plot function this:
`pch = 20, cex = 3`

Exercise 10

Fully color the points of your plot.

Building Shiny App exercises part 6



RENDER FUNCTIONS

In the sixth part of our series we will talk about the renderPlot and the renderUI function and then we will be ready to create our first visualization. (Find part 1-5 here).
We are going to create a simple interactive scatterplot that will help us see the clusters that are created when we run the k-means algorithm on our dataset. Read the examples below to understand how to activate a renderPlot function and the test yous skills 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.

DESCRIPTIVE STATISTICS

As in every statistical application it is wise to apply descriptive statistics on your dataset and also provide this information to user in an easy-readable way. So, first of all we will place a Data Table inside the “SUMMARY” tabPanel. The

example below can be your guide.

```
#ui.R
library(shiny)
shinyUI(fluidPage(
sidebarLayout(
sidebarPanel(
),
mainPanel(
dataTableOutput("Table")
)
)))
#server.R
shinyServer(function(input, output, session) {
sum<-as.data.frame.array(summary(iris))
output$Table <- renderDataTable(sum)
})
```

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

Create a Data Table("Table2") with the descriptive statistics of your dataset. HINT: Use summary, as.data.frame.array and renderDataTable.

renderPlot

The renderPlot function enders a reactive plot that is suitable for assigning to an output slot. The general form of the function that generates the plot is below:

```
renderPlot(expr, width = "auto", height = "auto", res = 72,
...,
env = parent.frame(), quoted = FALSE, execOnResize = FALSE,
```

```
outputArgs = list())
```

The example below shows you how to create a simple scatterplot between two variables of the iris dataset(“Sepal Length” and “Sepal Width”).

```
# ui.R
library(shiny)
shinyUI(fluidPage(
sidebarLayout(
sidebarPanel(
),
mainPanel(
plotOutput("plot1")
)
)))
#server.R
shinyServer(function(input, output, session) {
output$plot1 <- renderPlot({
plot(iris$Sepal.Length,iris$Sepal.Width)
})
})
```

Initially remove renderImage and radioButtons from the tabPanel “K means”.

Exercise 2

Add a scatterplot inside the tabPanel “K Means” between two variables of the iris dataset.

INTERACTIVE PLOTS

Shiny has built-in support for interacting with static plots generated by R’s base graphics functions,this makes it easy to add features like selecting points and regions, as well as zooming in and out of images.
To get the position of the mouse when a plot is clicked, you simply need to use the click option with the plotOutput. For

example, this app will print out the x and y coordinate position of the mouse cursor when a click occurs.

```
#ui.R
library(shiny)
shinyUI(fluidPage(
sidebarLayout(
sidebarPanel(),
mainPanel(
plotOutput("plot1", click = "plot_click"),
verbatimTextOutput("info")
)
)))
#server.R
shinyServer(function(input, output, session) {
output$plot1 <- renderPlot({
plot(iris$Sepal.Length,iris$Sepal.Width)
})
output$info <- renderText({
paste0("x=", input$plot_click$x, "\ny=", input$plot_click$y)
})
})
```

Exercise 3

Add click inside the plotOutput you just created. Name it “mouse”.

Exercise 4

Add a verbatimTextOutput inside the “K Means” tabPanel,under the plotOutput you created before. Name it “coord”.

Exercise 5

Make “x” and “y” coordinates appear in the pre-tag you just created. HINT : Use renderText and paste0 and do not forget to activate it with the submitButton.

Exercise 6

Set height = “auto” and width = “auto”.

PLOT ANNOTATION

This function can be used to add labels to a plot. Its first four principal arguments can also be used as arguments in most high-level plotting functions. They must be of type character or expression. In the latter case, quite a bit of mathematical notation is available such as sub- and superscripts, greek letters, fraction, etc.

```
title(main = NULL, sub = NULL, xlab = NULL, ylab = NULL,
line = NA, outer = FALSE, ...)
```

Look at the example below:

```
# ui.R
library(shiny)
shinyUI(fluidPage(
sidebarLayout(
sidebarPanel(),
mainPanel(
plotOutput("plot1", click = "plot_click"),
verbatimTextOutput("info")
)
)))
#server.R
shinyServer(function(input, output, session) {
output$plot1 <- renderPlot({
plot(iris$Sepal.Length,iris$Sepal.Width,main  =  "SCATTER PLOT",sub  =  "K  Means",xlab="Sepal  Length",ylab  =  "Sepal Width")
})
output$info <- renderText({
paste0("x=", input$plot_click$x, "\ny=", input$plot_click$y)
})
})
```

Exercise 7

Set scatterplot title to “K-Means”, the X-axis label to “Petal Length” and the Y-axis label to “Petal Width”. HINT: Use main,xlab,ylab.

You can also modify and set other graphical parameters related to the title and subtitle like the example below:

```
# ui.R
library(shiny)
shinyUI(fluidPage(
sidebarLayout(
sidebarPanel(),
mainPanel(
plotOutput("plot1", click = "plot_click"),
verbatimTextOutput("info")
)
)))
#server.R
shinyServer(function(input, output, session) {
output$plot1 <- renderPlot({
plot(iris$Sepal.Length,iris$Sepal.Width,main  =  "SCATTER PLOT",sub  =  "K  Means",xlab="Sepal  Length",ylab  =  "Sepal Width",
cex.main = 3, font.main= 5, col.main= "green",
cex.sub = 0.65, font.sub = 4, col.sub = "orange")
})
output$info <- renderText({
paste0("x=", input$plot_click$x, "\ny=", input$plot_click$y)
})
})
```

Exercise 8

Give values to the rest of the graphical parameters of the title like the example above and get used to them. HINT: Use cex.main, font.main and col.main.

renderUI

```
renderUI(expr, env = parent.frame(), quoted = FALSE,
outputArgs = list())
```

Makes a reactive version of a function that generates HTML using the Shiny UI library. As you can see in the example below this expression returns a tag object.

```
# ui.R
library(shiny)
shinyUI(fluidPage(
sidebarLayout(
sidebarPanel( uiOutput("Controls")),
mainPanel(
plotOutput("plot1", click = "plot_click"),
verbatimTextOutput("info")
)
)))
#server.R
shinyServer(function(input, output, session) {
output$plot1 <- renderPlot({
plot(iris$Sepal.Length,iris$Sepal.Width,main = "SCATTER
PLOT",sub = "K Means",xlab="Sepal Length",ylab = "Sepal
Width",
cex.main = 2, font.main= 4, col.main= "blue",
cex.sub = 0.75, font.sub = 3, col.sub = "red")
})
output$info <- renderText({
paste0("x=", input$plot_click$x, "\ny=", input$plot_click$y)
})
output$Controls <- renderUI({
tagList(
sliderInput("n", "N", 1, 1000, 500),
textInput("label", "Label")
)
})
})
```

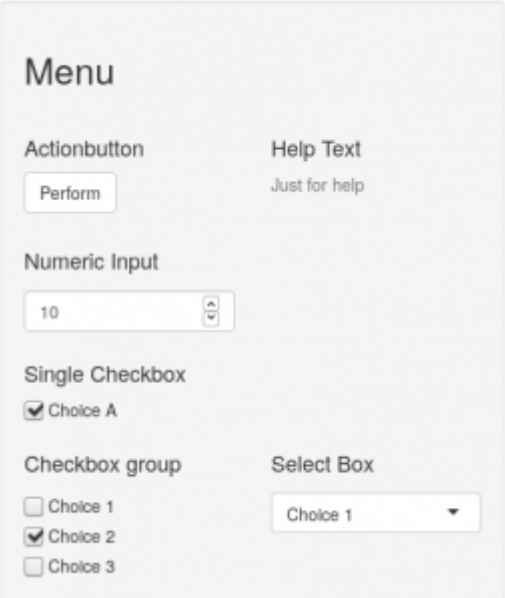
Exercise 9

Put a uiOutput inside tabPanel “K-Means” and name it “All”. Then create its output in server.R with a tagList into it. HINT: Use uiOutput, renderUI and tagList.

Exercise 10

Remove the submitButton and move the sliderInput and the textOutput from the ui.R into the tagList.

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 containining 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 radioButtonshere 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.

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 textOutpout 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.

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.

Building Shiny App exercises part 3

ADD CONTROL WIDGETS

Welcome to the third part of our series. In this part you will learn how to build and place inside your app the rest of the widgets which were mentioned in part 2.

More specifically we will analyze: 1) helptext, 2) numericInput, 3) radioButtons, 4) selectInput, 5) sliderInput and 6) textInput.

As you already know from part 2 reactivity will be added in the upcoming parts of our series so this is something that you do not have to worry about.

Follow the examples below to understand the logic behind the widgets' functions and then enhance the app you created in part 1 by practising with the exercise set we prepared for you.

Firstly, we will add all the kinds of the widgets in our app, for educational reasons and later we will decide which of them is practical to keep.Let's start!

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.

To begin with let's create the space inside our sidebarPanel in order to put in there the rest of our widgets.

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

Use the function fluidrow to make sure that all the elements we are going to use will be in the same line. To do this put fluidrow just under the "Menu" in your sidebarPanel and close its parenthesis just before the submibutton (excluding the two br).

HELP TEXT

In the example below we create a UI with a helpText.

```
# ui.R
shinyUI(fluidPage(
titlePanel("Widgets"),
h3("Help Text"),
helpText("Text that is used to provide some extra details to the user.")))
```

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

Exercise 2

Place a helpText exactly under the actionButton, name it "Help Text" and as text add:"For help". Hint: Use h4.

Exercise 3

Now use column function in order to decide the column width for every row and put the two widgets in the same line. To do this place the column function twice. Firstly place it just before the "Actionbutton" title with width = 6 and close its parenthesis exactly after the label "Perform". Do the same for the helpInput. Both of the column functions must be inside the same fluidrow.

NUMERIC INPUT

In the example below we create a UI with a numericInput.

```
# ui.R
shinyUI(fluidPage(
titlePanel("Widgets"),
numericInput("num",
label = h3("Numeric Input"),
value = 1)
))
```

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

Exercise 4

Put a numericInput under helpText,in the same row with submitButton. Name it “numer”, give it “Numeric Input” as label and value = 10. Hint: Use h4, fluidrow and column.

RADIO BUTTONS

In the example below we create a UI with a radioButtons.

```
#ui.R
shinyUI(fluidPage(
titlePanel("Widgets"),
radioButtons("radio", label = h3("Radio buttons"),
choices = list("Choice 1" = 1, "Choice 2" = 2,
"Choice 3" = 3),selected = 1)
))
```

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

Exercise 5

Add radioButtons under numericInput, in the same row with checkBoxInput. Name it “radiobuttons”, put as label “Radio

Buttons” and give it two choices with no default. Hint: Use h4, fluidrow, column and choices.

Exercise 6

Now put “2” as the default of the choices. Hint: Use selected.

SELECT INPUT

In the example below we create a UI with a selectInput.

```
# ui.R
shinyUI(fluidPage(
titlePanel("Widgets"),
selectInput("select", label = h3("Select Box"),
choices = list("Choice 1" = 1, "Choice 2" = 2,
"Choice 3" = 3), selected = 1)
))

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

Exercise 7

Place under radiobuttons and in the same row with checkBoxGroupInput a selectinput. Its name should be “select”, its label “Select Box” and you should give it two choices with the second one as default. Hint: Use h4, fluidrow, column, choices and selected.

SLIDER INPUT

In the example below we create a UI with two sliderInput.

```
# ui.R
shinyUI(fluidPage(
titlePanel("Widgets"),
sliderInput("slider1", label = h3("Sliders"),
min = 0, max = 10, value = 5),
sliderInput("slider2", "",
min = 0, max = 10, value = c(3, 7))
```

```
))

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

Exercise 8

Under the selectInput and in the same row with the dateInput place a sliderInput with name = slider1, label = “Sliders”, min = 0, max = 100 and value =50. Hint: Use fluidrow, columns and h4.

Exercise 9

Replace the value with a default range “10-90” and see the difference.

TEXT INPUT

In the example below we create a UI with a textInput.

```
# ui.R
shinyUI(fluidPage(
titlePanel("Widgets"),
textInput("text", label = h3("Text Input"),
value = "Text..."))
)

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

Exercise 10

Finally put a textInput under sliderInput and in the same row with the dateRangeInput. Name it “text”, put as label “Text Input” and as value “Some Text”. Hint: Use fluidrow, column and h4.

Building Shiny App exercises part 2

ADD CONTROL WIDGETS

In the second part of our series you will see how to add control widgets in your Shiny app. Widget is a web element that your users can interact with. The widgets provided by Shiny are:

FUNCTIONS

actionButton: Action Button
checkboxGroupInput: A group of check boxes
checkboxInput: A single check box
dateInput: A calendar for date selection
dateRangeInput: A pair of calendars for selecting a date range
fileInput: A file upload control wizard
helpText: Help text that can be added to an input form
numericInput: A field to enter numbers
radioButtons: A set of radio buttons
selectInput: A box with choices to select from
sliderInput: A slider bar
submitButton: A submit button
textInput: A field to enter text

ADDING WIDGETS

You can add widgets to your web page in the same way that you added other types of HTML content in part 1.
To add a widget to your app, place a widget function in sidebarPanel or in mainPanel in your ui.R file.
The first two arguments for each widget are a name for the widget which will be a character string that the user will not see, but you can use it to change the widget's value and a

label which will appear with the widget in your app and it should be a character string too.
The rest of the arguments vary from widget to widget, depending on what the widget needs to be functional. They may be initial values, ranges, and increments.
Follow the examples below to understand the logic behind the widgets' functions and then enhance the app you created in part 1 by practising with the exercise set we prepared for you.
Firstly, we will add all the kinds of the widgets to our app, for educational reasons and later we will decide which of them is practical to keep.
Note that we will just add the buttons in this part. Reactivity will be added to them in a few chapters. 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.

Exercise 1

Open the app you created in part 1 and move the image from sidebarPanel to mainPanel, leave two rows under the title "Main", put the image there and change its dimensions to: height = 150 and width = 200. HINT: Use br.

BUTTONS

In the example below we create a UI with a submitButton and an actionButton. Please note that we use the function fluidrow to

make sure that all the elements we are going to use will be in the same line as we are going to need this in the next parts:

```
# ui.R
shinyUI(fluidPage(
titlePanel(“Widgets”),

fluidRow(h3(“Buttons”),
actionButton(“action”, label = “Action”),
br(),
br(),
submitButton(“Submit”))))

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

Exercise 2

Leave a row and place an actionButton under the title “Menu” in sidebarPanel, give it the title “Actionbutton”, name = “per” and label = “Perform”. HINT: Use br and h4.

Exercise 3

Leave a row from the actionButton you just placed and add a submitButton with title = “Submitbutton” and name = “Submit”. HINT: Use br and h4.

SINGLE CHECKBOX

In the example below we create a UI with a single Checkbox:

```
# ui.R
shinyUI(fluidPage(
titlePanel(“Widgets”),

fluidRow(h3(“Single checkbox”),
checkboxInput(“checkbox”, label = “Choice A”, value = TRUE))))

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

Exercise 4

Add a checkboxInput in the sidebarPanel under the submitButton, put as title “Single Checkbox”, name it “checbox”, name the label “Choice A” and set the value to “TRUE”. HINT: Use h4.

Exercise 5

Change the value to “FALSE” to understand the difference.

CHECKBOX GROUP

In the example below we create a UI with a Checkbox Group:

```
#ui.R
shinyUI(fluidPage(
checkboxGroupInput(“checkGroup”,
label = h3(“Checkbox group”),
choices = list(“Choice 1” = 1,
“Choice 2” = 2, “Choice 3” = 3),
selected = 2)
))

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

Exercise 6

Place a checkboxGroupInput under the checkboxInput, give it title “Checkbox Group”, name it “checkGroup”, name the label “Checkbox Group” and give it 3 choices. HINT: Use h4

Exercise 7

Make the second of the choicesthe default one.

DATE INPUT

In the example below we create a UI with a Date Input:

```
#ui.R
shinyUI(fluidPage(
```

```
dateInput(“date”,
label = h3(“Date input”),
value = “2016-12-07”)
))

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

Exercise 8

Under the checkboxGroupInput add a dateInput with name = “date”, label = “Date Input” and value = “2016-12-01”.

DATE RANGE
In the example below we create a UI with a Date Range Input:

```
#ui.R
shinyUI(fluidPage(
dateRangeInput(“dates”, label = h3(“Date range”))
))

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

Exercise 9

Under the dateInput place a dateRangeInput with name = “dates” and label = “Date Range”. HINT: Use h4.

FILE INPUT
In the example below we create a UI with a File Input.

```
#ui.R
shinyUI(fluidPage(
fileInput(“file”, label = h3(“File input”))
))

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

Exercise 10

Under the dateRangeInput place a fileInput. Name it “file” and give it the label “File Input”.

Building Shiny App exercises part 1

INTRODUCTION TO SHINY

Shiny is a package from RStudio that can be used to build interactive web pages with RStudio which is is an open source set of integrated tools designed to help you be more productive with R and you can download it from here. Use the examples in this tutorial to “take a first bite” and prepare for the exercises that follow and will help you build your first Shiny Application from “zero point”. This is the first part of the series and we will just create the interface, make some HTML formatting and add an image to our application. Specifically we will start creating a Shiny Application that will analyze 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. Lets go!

BUILDING INTERFACE (UI)

Every Shiny application includes two parts: a web page which shows the app to the user (UI), and a computer that activates the app (server). You need to create these two parts. UI is just a web document that the user gets to see and is responsible for creating the layout of the app and telling

Shiny exactly where things go. The server is responsible for the logic of the app.
You can create a Shiny app by making a new directory and saving a ui.R and server.R file inside it. Each app will need its own unique directory.
You can run a Shiny app by giving the name of its directory to the function runApp(). For example if your Shiny app is in a directory called “Shiny App”, run it with the following code:

```
library(shiny)
runApp("Shiny App")
```

Or by just clicking the “Run App” button at the top of the editor which is the safest solution.

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.

Exercise 1

Create a new directory named “Shiny App” in your working directory.

Exercise 2

Create the ui.r and server.r files.

Secondly, we have to install the “Shiny” package with:

```
install.packages("shiny")
```

and then we will call it with:

```
library(shiny)
```

Now that we analyzed the structure of a Shiny app, we will show you how to build a user-interface for this.
To get started, we open the server.R and ui.R files and edit them like this:

```
#ui.R
shinyUI(fluidPage())
#server.R
shinyServer(function(input, output) {})
```

The result is an empty app with a blank user-interface, just to begin with.

Exercise 3

Create an empty app with a blank user-interface.

LAYOUT

Shiny ui.R scripts use the function fluidPage() to create a display that automatically adjusts to the dimensions of your user's browser window. You lay out your app by placing elements in the fluidPage() function.
For example, the ui.R script below creates a user-interface that has a title panel and then a sidebar layout, which includes a sidebar panel and a main panel. Note that these elements are placed within the fluidPage() function.

```
#ui.R

shinyUI(fluidPage(
titlePanel("title panel"),
sidebarLayout(
sidebarPanel( "sidebar panel"),
mainPanel("main panel"))))
```

The TitlePanel() and sidebarLayout() are the two most used elements to add to fluidPage(). They create a basic Shiny app

with a sidebar.
The sidebarLayout() always takes two arguments: sidebarPanel function output and mainPanel function output.

Exercise 4

Create titlePanel(), name it “Shiny App” and sidebarLayout(). Do not forget to add sidebarPanel() and mainPanel() inside this.

HTML Content
You can add content to your Shiny app by placing it inside a *Panel function.
HEADERS
To create a header element: select a header function e.g. (h1) and give it the text you want to see in the header.
For example, you can create a first level header that says “Title” with h1("Title").
To place the element in your app:
Put h1("Title") as an argument to titlePanel(), sidebarPanel(), or mainPanel().
The text will appear in the corresponding panel of your web page. You can place multiple elements in the same panel if you separate them with a comma.
#ui.R

```
shinyUI(fluidPage(
titlePanel("My Shiny App"),
sidebarLayout(
sidebarPanel(),
mainPanel(
h1("First level title"),
h2("Second level title")))))
```

Exercise 5

Create an HTML element to add the title “Menu” in the sidebarPanel() and “Main” in mainPanel() with one of Shiny’s

tag functions. HINT: Use h1,h2.

FORMATTED TEXT
Shiny offers many tag functions for formatting text.Take a look:
p: A paragraph of text
h1: A first level header
h2: A second level header
a: A hyper link
br: A line break (e.g. a blank line)
div: A division of text with a uniform style
span: An in-line division of text with a uniform style
pre: Text ‘as is’ in a fixed width font
code: A formatted block of code
img: An image
strong: Bold text
em: Italicized text

Exercise 6

Add a paragraph in your mainPanel() with a description about the app you are going to make.”This famous (Fisher’s or Anderson’s) iris data set 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.” Use an HTML tag format. HINT:Use p.

Exercise 7

Link the word “iris” in the mainPanel() with this hyperlink “http://stat.ethz.ch/R-manual/R-devel/library/datasets/html/iris.html”. HINT: Use a.

Exercise 8

Add the title “Analysis” under the desciption paragraph of your mainPanel. Do not forget the comma separation. HINT: Use br and h2.

Exercise 9

Use bold text to the words “Iris setosa”,”versicolor” and “virginica”. HINT: Use strong.

IMAGES

Images can improve the appearance of your app and help users understand the content. Shiny uses img() function to put image files in your app. To insert an image, give the img() function the name of your image file as the src argument (e.g., img(src = "my_image.png")). You can also include other HTML parameters such as height and width. For example:

```
img(src = "my_image.png", height = 68, width = 68)
```

.

The img() function looks for your image file in a specific place. Your file must be in a folder named “www” in the same directory as the ui.R script. Shiny will share any file placed here with your user’s web browser, which makes “www” a great place to put images, style sheets, and other things the browser will need to build the web components of your Shiny app. So if you want to use an image named “something.png”, your Shiny App directory should look like this one:

#ui.R

```
shinyUI(fluidPage(
titlePanel("My Shiny App"),
sidebarLayout(
sidebarPanel(),
mainPanel(
img(src="something.png", height = 350, width = 350)))))
```

Exercise 10

Download this image and place it in a folder labeled “www” within your “Shiny App” directory. Name it “petal”, add .jpg extension and then call the img function inside the sidebarPanel(). Use height and width to decide its dimensions.