Web Apps using Package 'shiny' - II

Contents

Case Study

Background

- A Telecom company wants to analyse the data to provide solution to managerial problems.

Objective

To create a shiny application which will

– Identify usage pattern of the customers across various demographics

– Identify volume of business generated across various demographics

– Develop a dashboard for the in-house analytics team

– Design efficient ways to communicate analysis insights with the marketing team

Case Study

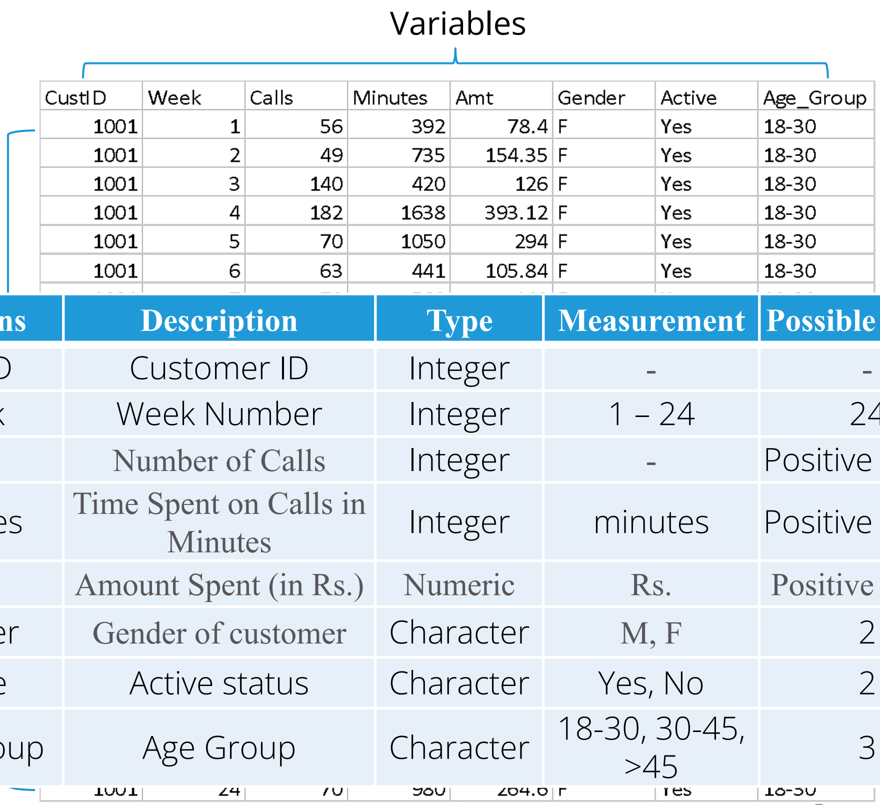
Data Description

Telecom data for 24 weeks was provided by the client at customer level.

Telecom data:

- 21902 rows & 5 columns
- Contains customer level information like Age_Group, Gender, and whether the customer has been active in the past six months.
- Contains Number of Calls, Time Spent on Calls (in Minutes) and the Amount Spent (in Rs.) which are recorded for the past 6 months (24 weeks) for each customer.

Data Snapshot

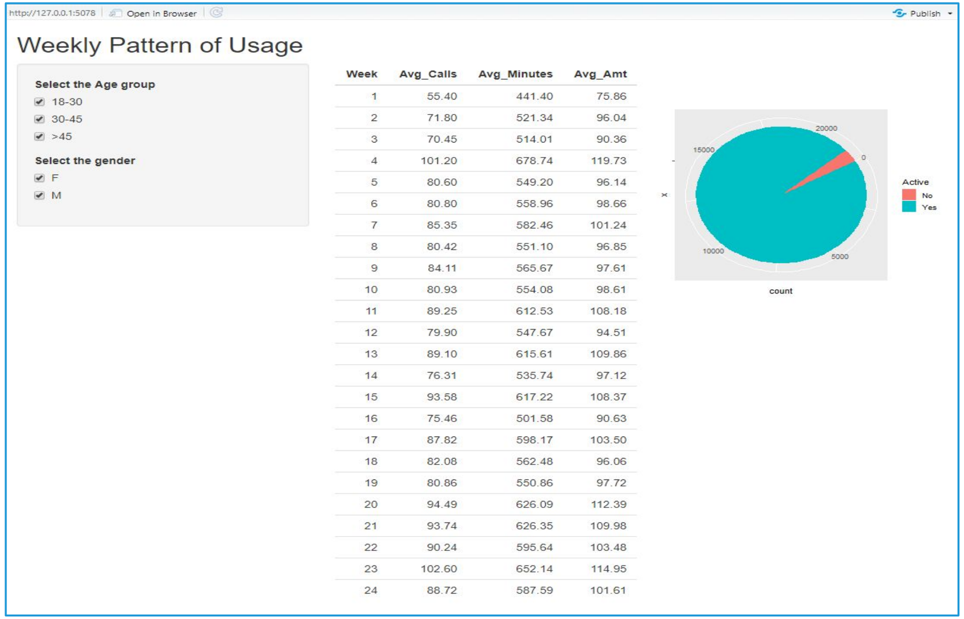


CustID	Week	Calls	Minutes	Amt	Gender	Active	Age_Group
1001	1	56	392	78.4	F	Yes	18-30
1001	2	49	735	154.35	F	Yes	18-30
1001	3	140	420	126	F	Yes	18-30
1001	4	182	1638	393.12	F	Yes	18-30
1001	5	70	1050	294	F	Yes	18-30
1001	6	63	441	105.84	F	Yes	18-30
1001	24	70	980	264.6	F	Yes	18-30

Columns	Description	Type	Measurement	Possible values
CustID	Customer ID	Integer	-	-
Week	Week Number	Integer	1 – 24	24
Calls	Number of Calls	Integer	-	Positive values
Minutes	Time Spent on Calls in Minutes	Integer	minutes	Positive values
Amt	Amount Spent (in Rs.)	Numeric	Rs.	Positive values
Gender	Gender of customer	Character	M, F	2
Active	Active status	Character	Yes, No	2
Age_Group	Age Group	Character	18-30, 30-45, >45	3

App Design

We'll now create the app which will look like this. It will allow you to see a table and pie chart. Table will show week wise average calls, minutes & amount for the chosen **Age Group** and **Gender** and a pie chart will represent active customers information graphically.



http://127.0.0.1:5078 | Open in Browser | Publish

Weekly Pattern of Usage

Select the Age group
- ☑ 18-30
- ☑ 30-45
- ☑ >45

Select the gender
- ☑ F
- ☑ M

Week	Avg_Calls	Avg_Minutes	Avg_Amt
1	55.40	441.40	75.86
2	71.80	521.34	96.04
3	70.45	514.01	90.36
4	101.20	678.74	119.73
5	80.60	549.20	96.14
6	80.80	558.96	98.66
7	85.35	582.46	101.24
8	80.42	551.10	96.85
9	84.11	565.67	97.61
10	80.93	554.08	98.61
11	89.25	612.53	108.18
12	79.90	547.67	94.51
13	89.10	615.61	109.86
14	76.31	535.74	97.12
15	93.58	617.22	108.37
16	75.46	501.58	90.63
17	87.82	598.17	103.50
18	82.08	562.48	96.06
19	80.86	550.86	97.72
20	94.49	626.09	112.39
21	93.74	626.35	109.98
22	90.24	595.64	103.48
23	102.60	652.14	114.95
24	88.72	587.59	101.61

Working of the app:

Whenever the user changes the choice for Age Group and Gender, the output will be rebuilt using the new value.

Create an Empty Shiny App

All Shiny apps follow the same template:

```
# ui.R
```

```
library(shiny)
fluidPage()
```

```
# server.R
```

```
function(input,output){
}
```

- Save the **ui.R** and **server.R** in a folder which is saved in your working directory. Save the folder with the name "**Myshinyapp**".

- After saving the file, RStudio should recognize that this is a Shiny app and you should see the usual '**Run App**' button'.

Build the Basic UI

```
# ui.R
library(shiny)
fluidPage(
  titlePanel("Weekly Pattern of Usage")
)
```

titlePanel() adds a title on the top left corner of the app and sets it as “official” title of the page.

- Add a layout:

```
# Add the following code after titlePanel()
sidebarLayout(
  sidebarPanel(),
  mainPanel()
)
```

- ❑ **sidebarLayout()** provides a simple two-column layout with a smaller sidebar and a larger main panel.
- ❑ All the arguments inside **fluidPage()** need to be separated by commas.

The layout of our app will be such that all the inputs (using widgets) that the user can manipulate will be in the sidebar, and the results will be shown in the main panel on the right.

Now we'll define some widgets in our sidebar panel.

* Note: when adding `sidebarLayout()` put ‘,’ after `titlePanel(),`

Add Widgets to Your App

Widget:

- **Control widgets are web elements that users can interact with.** Widgets provide input to your Shiny app. As the input provided to the widget changes, the value also changes.
- Eg: In our **ui.R** script, the buttons where you select the variables are the widgets. They let the user select the inputs for which the table and pie chart is provided in the main panel.
- The widget we have used here is **`checkboxGroupInput()`** which gives user a list of choices to select from, the code for which is :

```
checkboxGroupInput(inputId="age",
                   label="Select the Age group
                   choices=c("18-30","30-45","
                   selected=c("18-30","30-45",
checkboxGroupInput(inputId="gender",
                   label="Select the gender",
                   choices=c("F","M"),
                   selected=c("F","M"))
```

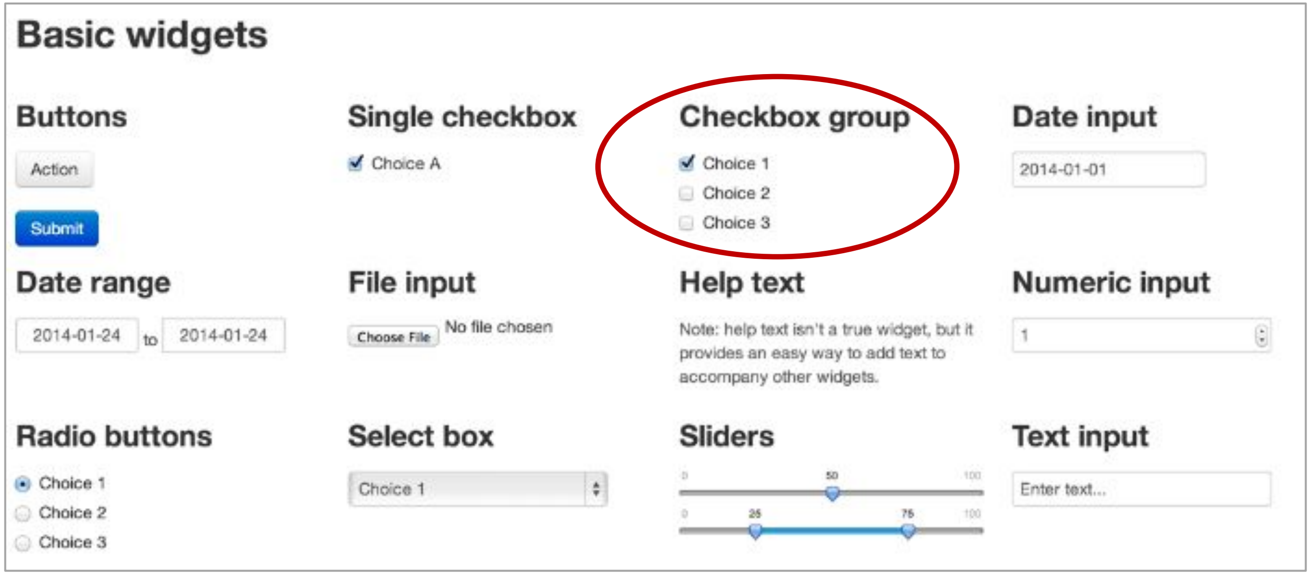
Add the above code in **`sidebarPanel()`**

- ❑ **inputId=** is the name with which the widget is stored. This is for internal computational purposes.
- ❑ **label=** is used just for user-interface, it can also be blank.
- ❑ **choices=** is the list of values to select from.
- ❑ **selected=** is the initially selected value.

Standard Shiny Widgets

Function	Widget
`actionButton`	Action Button
`checkboxGroupInput`	A group of check boxes
`checkboxInput`	A single check box
`dateInput`	A calendar to aid 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

Standard Shiny Widgets



Add Placeholders for Output in UI

- After creating all the inputs, we should add elements to the UI to display the outputs. Outputs can be any objects that R creates such as a plot, a table, or text.
- We will add a placeholder for the outputs in the UI that will determine where the output will be displayed and what its ID is.
- Each output needs to be built in the server code to be able to respond to the users input.

Add Placeholders for Output in UI

```
# Add placeholder for table and piechart in the mainPanel()
mainPanel(tableOutput("weeks"),plotOutput("pie"))
```

This will add two placeholders in the UI for a table and a pie named **weeks** and **pie** respectively.

- Notice that `tableOutput()` and `plotOutput()` takes an argument, the character string `"weeks"` and `"pie"` respectively.

- Each of the *Output functions require a single argument : a character string that Shiny will use as the name to connect to server.

Output Functions for UI Script

Each function creates a specific type of output	
Output Function	creates
htmlOutput	raw HTML
imageOutput	image
plotOutput	plot
tableOutput	table
textOutput	text
uiOutput	raw HTML
verbatimTextOutput	text

Check Point: What Our App Looks Like After Implementing UI

So far our complete app looks like this

```
# ui.R
```

```
library(shiny)
fluidPage(
  titlePanel("Weekly Pattern of Usage"),
  sidebarLayout(
    sidebarPanel(
      checkboxGroupInput(inputId="age",
                         label="Select the Age group",
                         choices=c("18-30","30-45",">45"),
                         selected=c("18-30","30-45",">45")),
      checkboxGroupInput(inputId="gender",
                         label="Select the gender",
                         choices=c("F","M"),
                         selected=c("F","M"))
  ),
  mainPanel(
    fluidRow(
      column(6,tableOutput("weeks")),
      column(6,plotOutput("pie"))
  ))
))
```

```
# server.R
```

```
function(input,output){}
```

Build the Objects for Output in SERVER

- Recall that we created two output placeholders: weeks(a table) and pie(a plot).
- **server.R** will include the code which will tell Shiny what kind of table or plot to display.
- **There are three rules to build an output in Shiny :**
  - Save the output object into the **output** list (remember that every server function has an **output** argument) which should match the ID defined for your object in your UI script.
  - Build the object with a **render*** function, where * is the type of output
  - Access input values using the **input** list

Build the Objects for Output in SERVER

```
# server.R
```

```
# Import Telecom data.
teledata <- read.csv("Telecom data.csv",header = TRUE)

# Load required libraries
library(ggplot2)
library(dplyr)

function(input,output){

  output$weeks <- renderTable({

    xy <- subset(teledata, (Age_Group %in% input$age) & (Gender %in%
input$gender))

summarise(group_by(xy,Week),Avg_Calls=mean(Calls),Avg_Minutes=mean(Min
utes),Avg_Amt=mean(Amt))

  })
```

Here, we are saving the output list (**output$weeks** & **output$pie**), using a render* function to build the output(**renderTable({})** & **renderPlot({})**) and accessing an input value (**input$age** & **input$gender**)

input$age and **input$gender** stores the value provided to the widget by the user, which is supplied to **Age_Group** and **Gender** variables to create subset **xy.**

Build the Objects for Output in SERVER

```
# server.R continued
```

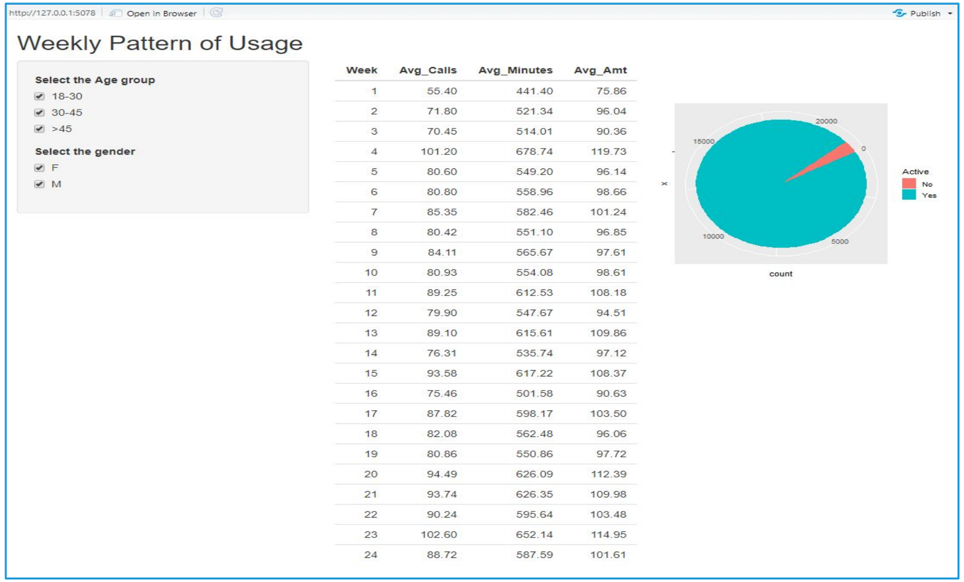
```
 output$pie <- renderPlot({

    xy <- subset(teledata, (Age_Group %in% input$age) & (Gender %in%
input$gender))
    pie <- ggplot(xy, aes(x="",fill=Active))+
           geom_bar(width = 1)+
           coord_polar(theta = "y", start = pi/3)
    pie
  })
}
```

Launch Your App!

You can launch your app by clicking on ‘Run App’ button or by running the below command in your R console

```
runApp("Myshinyapp")
```



Weekly Pattern of Usage

Select the Age group
- ✓ 18-30
- ✓ 30-45
- ✓ >45

Select the gender
- ✓ F
- ✓ M

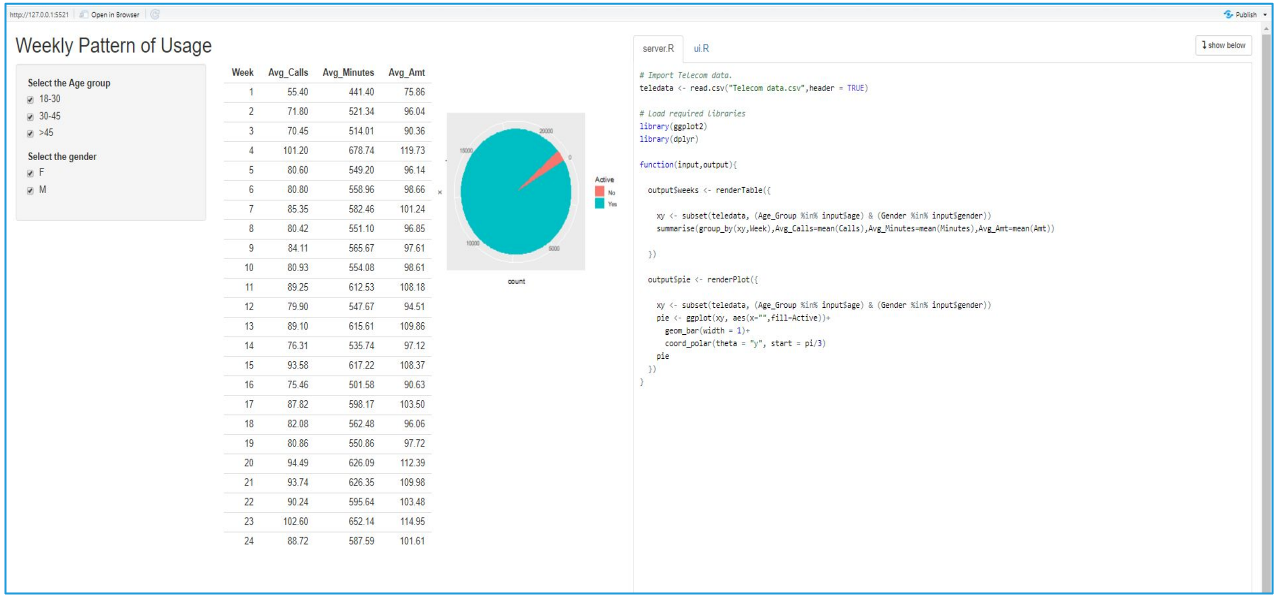
Week	Avg_Calls	Avg_Minutes	Avg_Amt
1	55.40	441.40	75.86
2	71.80	521.34	96.04
3	70.45	514.01	90.36
4	101.20	678.74	119.73
5	80.60	549.20	96.14
6	80.80	558.96	98.66
7	85.35	582.46	101.24
8	80.42	551.10	96.85
9	84.11	565.67	97.61
10	80.93	554.08	98.61
11	89.25	612.53	108.18
12	79.90	547.67	94.51
13	89.10	615.61	109.86
14	76.31	535.74	97.12
15	93.58	617.22	108.37
16	75.46	501.58	90.63
17	87.82	598.17	103.50
18	82.08	562.48	96.06
19	80.86	550.86	97.72
20	94.49	626.09	112.39
21	93.74	626.35	109.98
22	90.24	595.64	103.48
23	102.60	652.14	114.95
24	88.72	587.59	101.61

* Note : If the directory is not set, then give the complete path in runApp() function while executing it in R console.

Launch Your App!

You can also choose to launch your app, highlighting the code each time each time it is running by adding the argument **display="showcase"** in the **runApp()** command:

```
runApp("Myshinyapp",display="showcase")
```



Weekly Pattern of Usage

Select the Age group
- [✓] 18-30
- [✓] 30-45
- [✓] >45

Select the gender
- [✓] F
- [✓] M

Week	Avg_Calls	Avg_Minutes	Avg_Amt
1	55.40	441.40	75.86
2	71.80	521.34	96.04
3	70.45	514.01	90.36
4	101.20	678.74	119.73
5	80.60	549.20	96.14
6	80.80	558.96	98.66
7	85.35	582.46	101.24
8	80.42	551.10	96.85
9	84.11	565.67	97.61
10	80.93	554.08	98.61
11	89.25	612.53	108.18
12	79.90	547.67	94.51
13	89.10	615.61	109.86
14	76.31	535.74	97.12
15	93.58	617.22	108.37
16	75.46	501.58	90.63
17	87.82	598.17	103.50
18	82.08	562.48	96.06
19	80.86	550.86	97.72
20	94.49	626.09	112.39
21	93.74	626.35	109.98
22	90.24	595.64	103.48
23	102.60	652.14	114.95
24	88.72	587.59	101.61

server.R | ui.R

```
# Import Telecom data.
teledata <- read.csv("Telecom data.csv",header = TRUE)

# Load required libraries
library(ggplot2)
library(dplyr)

function(input,output){

  output$weeks <- renderTable({

    xy <- subset(teledata, (Age_Group %in% input$age) & (Gender %in% input$gender))
    summarise(group_by(xy,Week),Avg_Calls=mean(Calls),Avg_Minutes=mean(Minutes),Avg_Amt=mean(Amt))

  })

  output$pie <- renderPlot({

    xy <- subset(teledata, (Age_Group %in% input$age) & (Gender %in% input$gender))
    pie <- ggplot(xy, aes(x="",fill=Active))+
      geom_bar(width = 1)+
      coord_polar(theta = "y", start = pi/3)
    pie
  })
}
```

Share Your App!

Once you have made the app, you can share it with others easily. There are two options:

Share your Shiny app as files:

1. Share a copy of server.R and ui.R files, as well as any supplementary materials used in your app (E.g. data files, images, etc.) with anyone, but it will work only if your users have R on their computer. The user can place these files in their working directory and launch the app in R with the same commands you used on your computer.

```
runApp("Myshinyapp")
```

Share Your App!

2. Share as a Web Page:

If you want to share your app with a number of people then the most easy and user friendly way is to share as a web page.

RStudio provides a platform called **shinyapps.io** which lets you upload your app.

To upload the app:

- Go to http://www.shinyapps.io/ and create a free or professional account.
- Make sure all your app files are in an isolated folder saved in your working directory.
- In order to upload the app on shinyapp.io, you need to install package devtools and rsconnect.

Share Your App!

Run following command to upload the app.

```
install.packages("devtools")
install.packages("rsconnect")
library(rsconnect)

rsconnect::setAccountInfo(name=<name>, token=<token>, secret=<token>)

rsconnect::deployApp("Myshinyapp", account=<account name>)
```

- ❑ **rsconnect::setAccountInfo()** configures a ShinyApps account for publishing from this system.
- ❑ **name=** Name of account to save
- ❑ **token=** User token for the account
- ❑ **secret=** User secret for the account

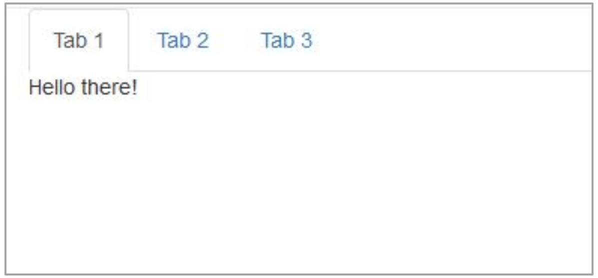
- ❑ The **deployApp()** function automatically syncs up to the Shinyapps.io server, and opens the shinyapps.io website on your browser. Once your app is uploaded, it is on the internet and can be viewed by anybody with access to it.

More Shiny App Features – tabsetPanel()

1. Multiple tabs in UI:

Using **`tabsetPanel()`** in ui.R script you can add multiple tabs and navigate between them.

```
shinyUI(fluidPage(
 tabsetPanel(
    tabPanel("Tab 1", "Hello there!"),
    tabPanel("Tab 2", " "),
    tabPanel("Tab 3", " ")
   )
))
```



Hello there!

More Shiny App Features – Package DT

2. Beautiful and Interactive Tables:

- With `tableOutput()` + `renderTable()`, shiny creates static tables.
- Using package **`DT`**, you can replace the default table with a much sleeker and interactive table.
- To use package **`DT`**, first install and load the package and then replace `tableOutput()` with `dataTableOutput()` & replace `renderTable()` with `renderDataTable()`.

More Shiny App Features – global.R

3. Global Objects:

- If there are objects that you want to have available to both **ui.R** and **server.R**, you can place them in **global.R** and call it with '`source("global.R")`' command at the beginning of ui.R and server.R.

- Defining the objects in **global.R** loads them into global environment of R session.

- **global.R** can also be used to add codes which are repeating in server.R. This is for computational ease and also provides a less messy looking server.R file.

More Shiny App Features – helper.R

3. **Helper Objects:**

- The helpers script file supports the SERVER and UI files to continue working smoothly. They can help you to install a package, calculate intermittent values for analysis and plots.

- Coding that needs to be repeated or have to be taken from a common source is put in helpers.R file. This is for computational ease. It also results in a less messy server.R file.

- The helpers.R file is accessed through the use of **`source("helpers.R")`** command in the server.R file.

More Shiny App Features – reactive()

3. **Reactive function :**

- `reactive()` function in shiny is used when one wants the outputs to be changed as the input selection are changed at user-interface level.

- When a user changes something in the app, the app sends the new input value to the server, which triggers a flush event and invalidates all the input’s dependents. Then all the reactive endpoints (like observers and outputs) are refreshed, which may also trigger a refresh of the reactive conductors (or reactive expressions defined with the reactive function).