My First Dashboard With Shiny

2019-11-22

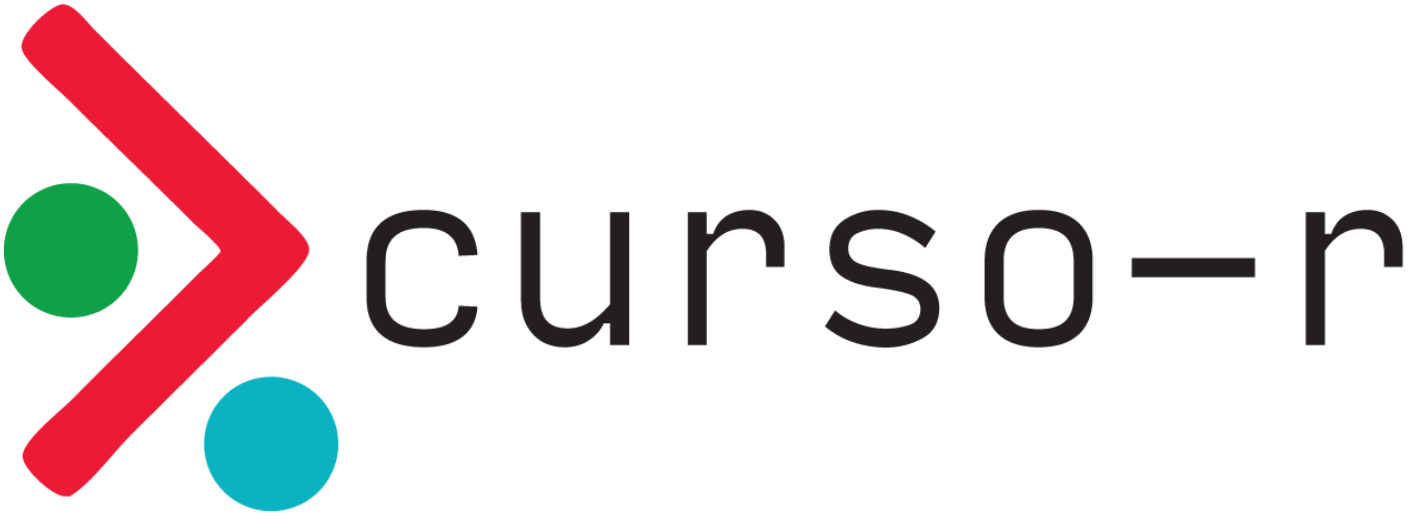
Requirements

- R v3.5.0 or superior

- RStudio v1.2.0 or superior

- Previous programming experience with R

- Files Click Here to Download



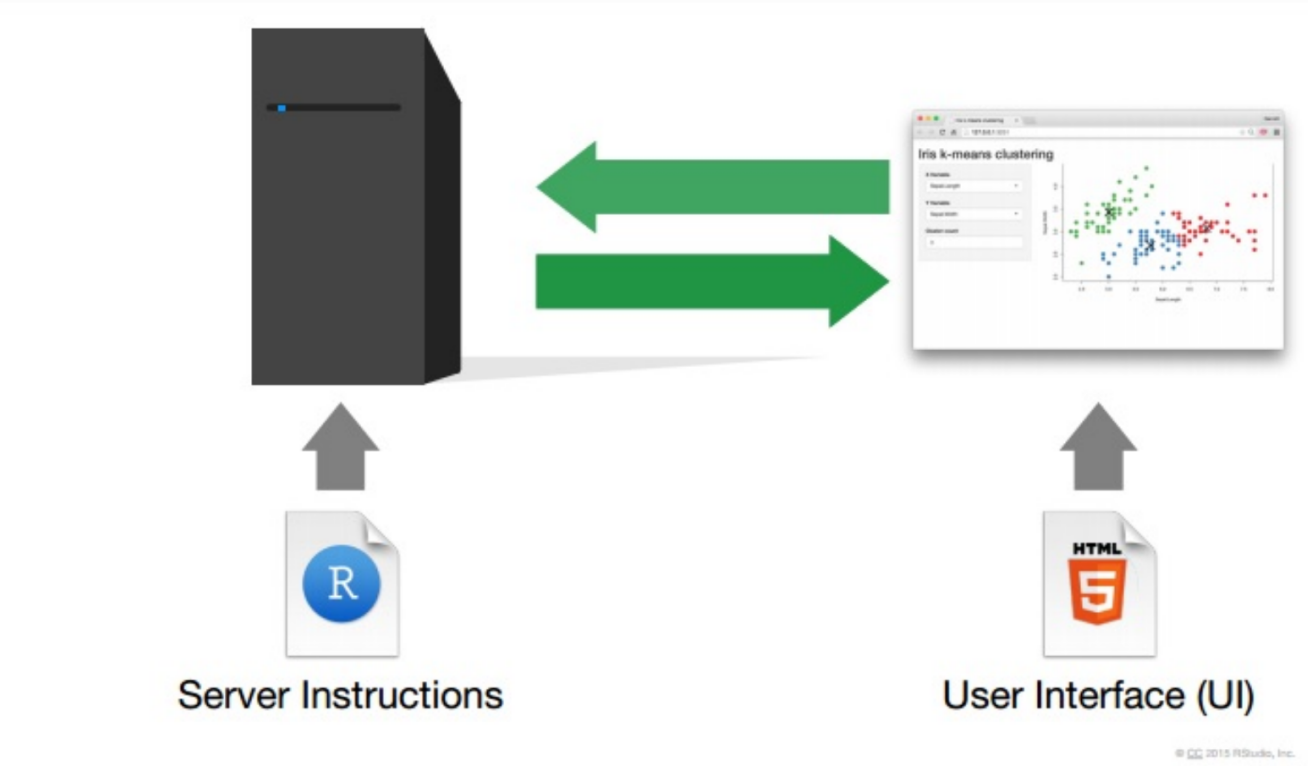
curso-r.com

Athos Petri Damiani

In this course, we are going to learn

1. How to build a Shiny App
2. Reactivity
3. Layouts (shinydashboard)
4. Deploy with Shinyapps.io (if we got time!)

Dynamic Dashboards



source: rstudio.com/shiny/

Motivation:

Shiny Gallery

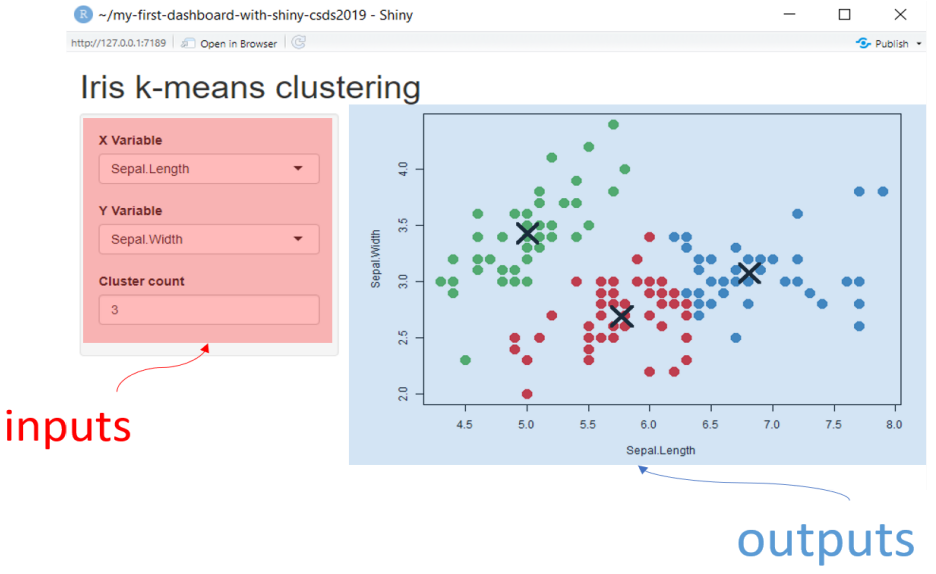
Show me Shiny Website

Hello World

```
library(shiny)

ui <- fluidPage("Hello World!")

server <- function(input, output, session) {

}

shinyApp(ui, server)
```

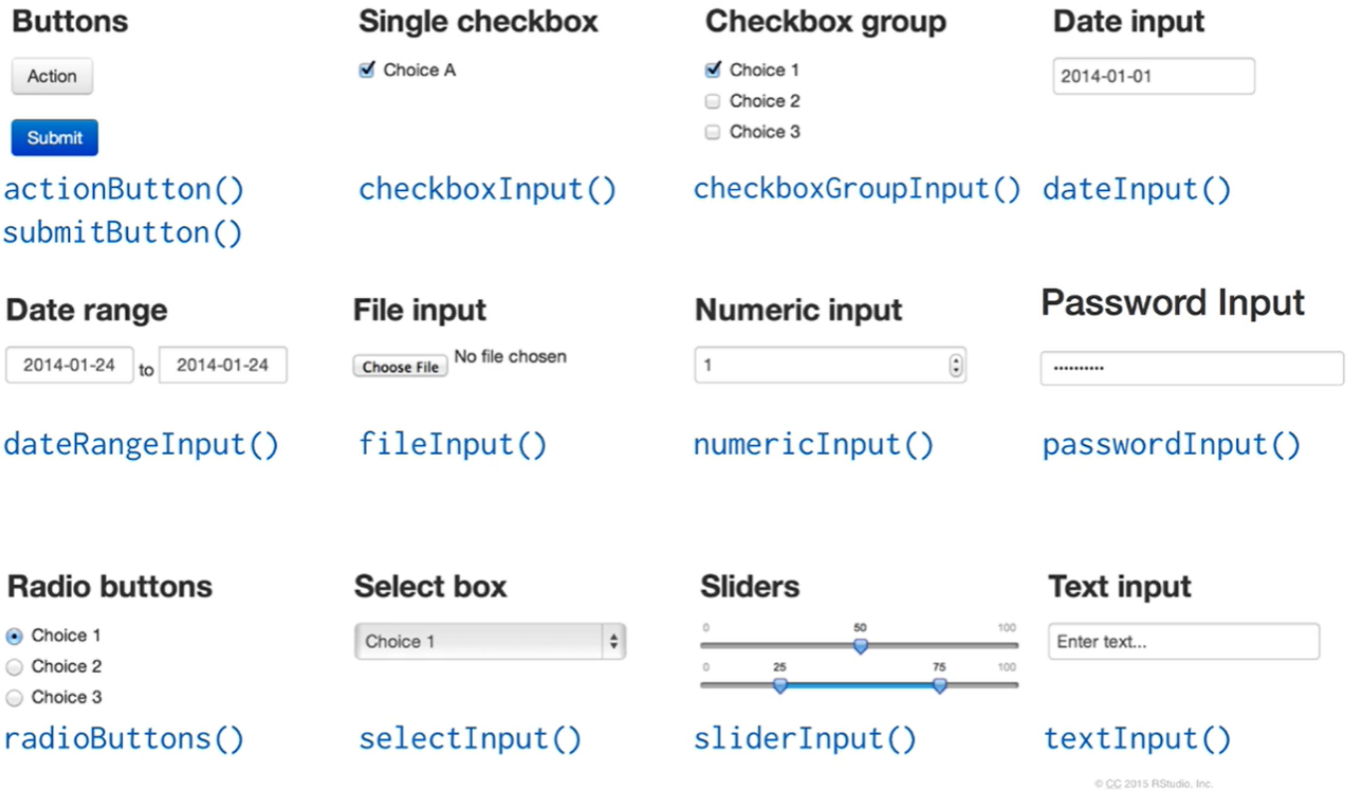
To R!

Inputs and Outputs




```
ui <- fluidPage(
  # *Input() functions,
  # *Output() functions
)
```

Inputs



source: rstudio.com/shiny/

Outputs

Function	Output
imageOutput()	image
plotOutput()	plot
tableOutput()	table
textOutput()	text
verbatimOutput()	text
htmlOutput()	raw HTML
dataTableOutput()	interactive table
uiOutput()	a Shiny UI element

In the "hello world" example, lets...

1) insert a slider input accepting values from 1 to 100.

2) insert a plot output.

3) insert a text input with `inputId = "title"`. (exercise!)

Server

```
server <- function(input, output, session) {




}
```

Server

```
server <- function(input, output, session) {

  output$hist <- renderPlot({
    hist(rnorm(100))
  })

}
```

remember our UI setup...

```
ui <- fluidPage(

  ...

  plotOutput("hist")
)
```

Server

render*() functions

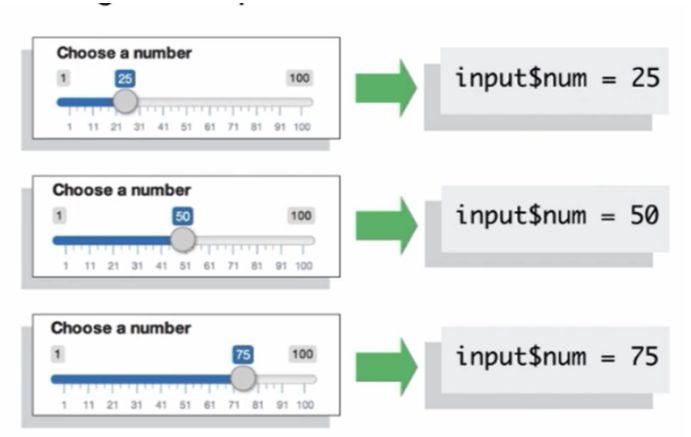
*output()	render*()
imageOutput()	renderImage()
plotOutput()	renderPlot()
tableOutput()	renderTable()
textOutput()	renderText()
verbatimOutput()	renderText()
htmlOutput()	renderUI()
dataTableOutput()	renderDataTable()
uiOutput()	renderUI()

```r
# ui
plotOutput("hist")
```

```r
# server
output$hist <- renderPlot({
  hist(rnorm(100))
})
```

Reactive Values and Reactive Functions

Use input values with **`input$`**




```
sliderInput(inputId = "num",…)
                    ↓
                input$num
```

- `render*()` are **reactive functions**
- `input$*` are **reactive values**

If we put **`input$num`** inside the **`renderPlot()`** in our example, the output would **`react`** if slider were changed by user.

```
# server
hist(rnorm(input$num))
```

source: rstudio.com/shiny/

Server

Recap:

Inside `server` function,

1) save the output using **`output$hist <-`**

2) feed the output **`render*()`** function with code

3) access **input values** with **`input$*`**

4) create reactivity by using **`input$*`** inside **`render*()`** functions

```
# server
output$hist <- renderPrint({
  hist(rnorm(input$num))
})
```

Back to our "hello world" example...

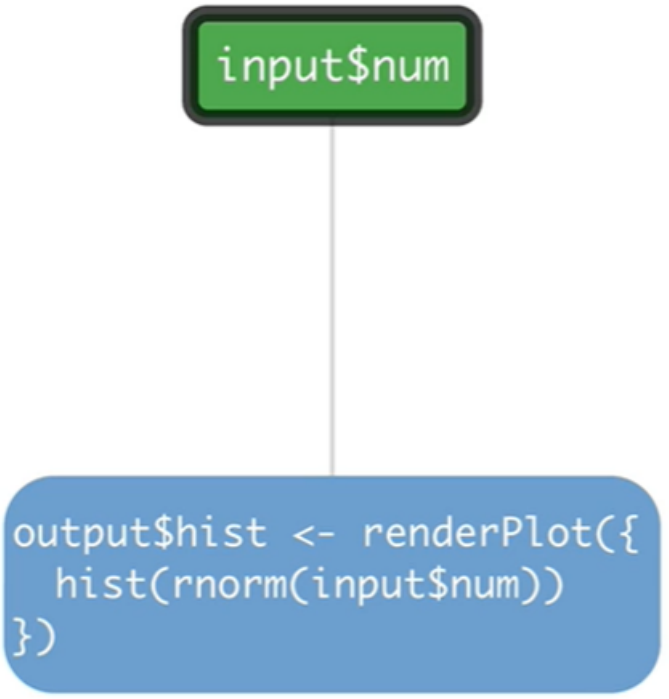
3) then, lets make the title of the histogram depending on the text input that you created before. (exercise!)

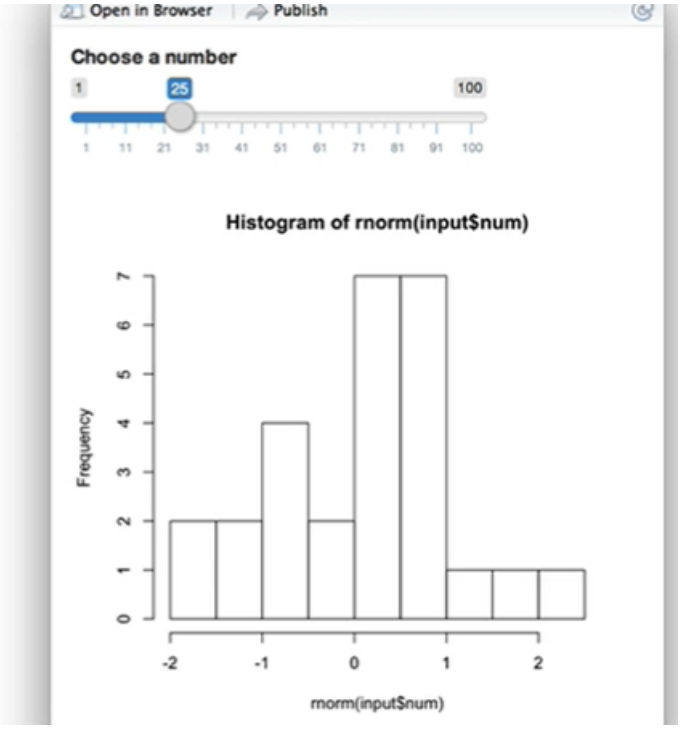
4) finally, create an **`verbatimTextOutput("summary")`** on ui and feed it with **`summary(rnorm(input$num))`** using **`renderPrint({})`** on server. (exercise!)

Reactivity

Reactivity

Reactivity is the relationship between **reactive values** and **reactive functions**: **reactive values** triggers **reactive functions**





Reactive Context

Reactive values must be usend in Reactive Context.

Right

```
# server
  output$hist <- renderPlot({hist(rnorm(input$num))})
```

Wrong

```
# server
output$hist <- hist(rnorm(input$num))
```

Reactive Functions

Main reactive functions in Shiny

- `render*({})`
- `reactive({})`
- `isolate({})`
- `observe({})`
- `eventReactive({})`
- `oberveEvent({})`

Reactive Functions

`reactive({})`

Builds a **reactive expression** (behaves like the **`input$*`**!).

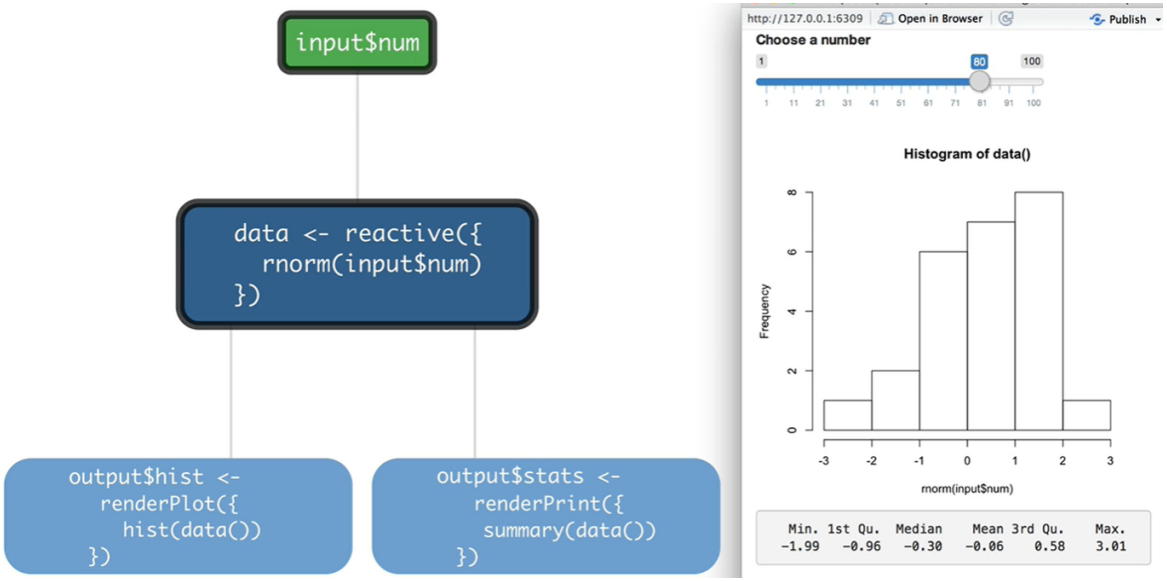
```
# server
data <- reactive({
  rnorm(input$num)
})
```

Is used like an ordinary function:

```
# server
output$summary <- renderPrint({
  summary(data())
})
```

Reactive Functions

`reactive({})`



1) Exercise: create the data() reactive expression in the 'hello world' example.

source: rstudio.com/shiny/

Reactive Functions

`isolate({})`

Turn an **reactive value** into a **non-reactive value**.

```
# server
output$hist <- renderPlot({
  * title <- isolate(input$title)
  hist(data(), main = title)
})
```

Now **`renderPlot({})`** will NOT react if **`input$title`** changes.

Reactive Functions

`observeEvent({})`

Useful to trigger a block of code to run on server whenever a given **`input$*`** changes.

```
# ui
actionButton("write_data", "Write Data as CSV")
```

```
# server
observeEvent(input$write_data, {
  write.csv(data(), "data.csv")
})
```

PS: action button is just another type of input! Useful along with **`observeEvent()`**.
See Using Action Button from Shiny docs.

Exercise: put the 'write data' functionality into 'hello world' example.

Reactive Functions

`observe({})`

Also useful to trigger a block of code to run on server, but any **reactive value** inside the code will trigger it.

```
# server
observe({
  print(data())
  print(as.numeric(input$writ_data))
})
```

Reactive Functions

`eventReactive({})`

Create a **reactive value** whenever a specified **`input$*`** changes (or any other **reactive value**).

```
# ui
actionButton("update", "Update!")
```

```
# server
data <- eventReactive(input$update, {
  write.csv(data(), "data.csv")
})
```

Exercise: put it into 'hello world' example.

Layouts

shiny.rstudio.com/layout-guide

But first... HTML Tags

```
fluidPage(
  tags$h1("My First Shiny App"),
  tags$p(
    "The link to the Shiny website is",
    tags$a(href = "https://www.rstudio.com/shiny/", "rstudio.com/sh
    tags$strong("I strongly recommend that you take a look at it!")
  )
)
```

```
<div class="container-fluid">
  <h1>My First Shiny App</h1>
  <p>
    The link to the Shiny website is
    <a href="https://www.rstudio.com/shiny/">rstudio.com/shiny.</a>
    <strong>I strongly recommend that you take a look at it!</stron
  </p>
</div>
```

But first... HTML Tags

```
names(tags)
```

```
##   [1] "a"          "abbr"       "address"    "area"       "article"
##   [6] "aside"      "audio"      "b"          "base"       "bdi"
##  [11] "bdo"        "blockquote" "body"       "br"         "button"
##  [16] "canvas"     "caption"    "cite"       "code"       "col"
##  [21] "colgroup"   "command"    "data"       "datalist"   "dd"
##  [26] "del"        "details"    "dfn"        "div"        "dl"
##  [31] "dt"         "em"         "embed"      "eventsource" "fieldset"
##  [36] "figcaption" "figure"     "footer"     "form"       "h1"
##  [41] "h2"         "h3"         "h4"         "h5"         "h6"
```

...

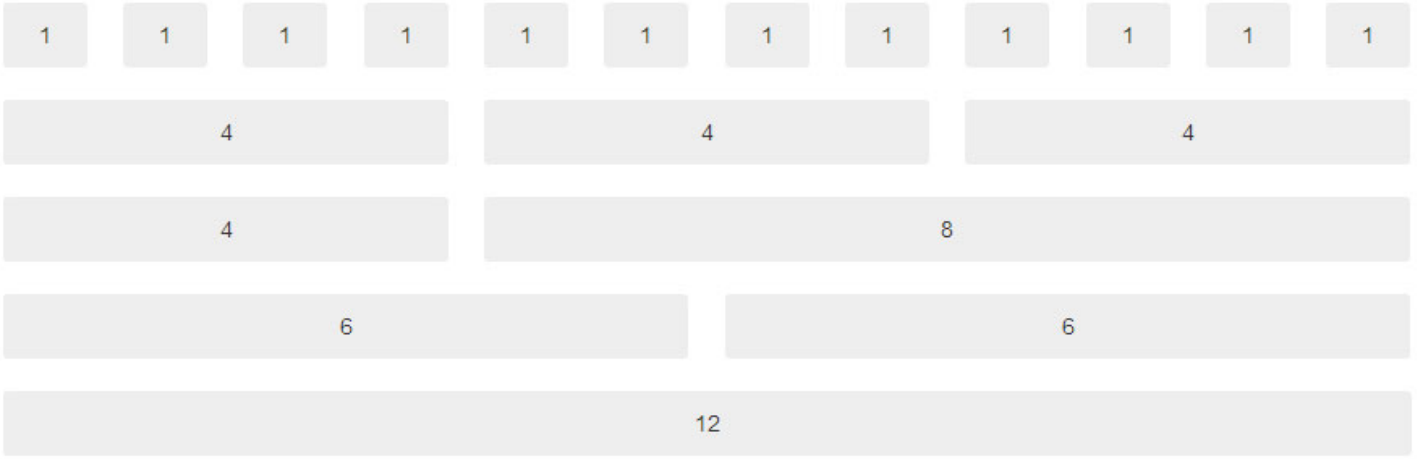
```
tags$h1("My First Shiny App")

## <h1>My First Shiny App</h1>
```

But first... HTML Tags

The most commonly used tags are:

R function	HTML
a()	a hyperlink
hr()	horizontal line
br()	line break
code()	monospaced code styled text
h1(), ..., h6()	headers
img()	image
p()	paragraph
em()	italic text
strong()	bold text

Grid System



- Each row is a **`fluidRow()`**.
- **`columns`** are divided up to 12 parts. The **`width`** determines how many parts wide are the column.

Grid System comes from Twitter's Open-source Bootstrap Framework.

source: dzone.com/articles/working-with-bootstrap-4-grid-system-for-creating

Grid System

Two main layout functions

```
fluidRow()
```

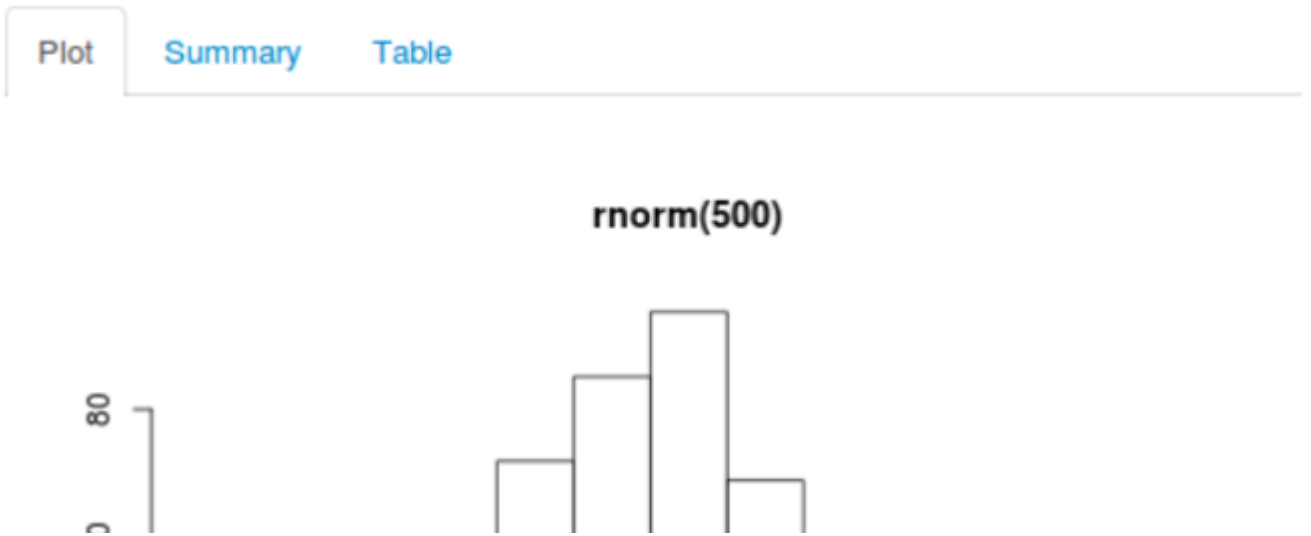
```
## <div class="row"></div>
```

```
column(2)
```

```
## <div class="col-sm-2"></div>
```

tabPanels and tabsetPanels

```
# ui
tabsetPanel(
  tabPanel("Plot", plotOutput("plot")),
  tabPanel("Summary", verbatimTextOutput("summary")),
  tabPanel("Table", tableOutput("table"))
)
```



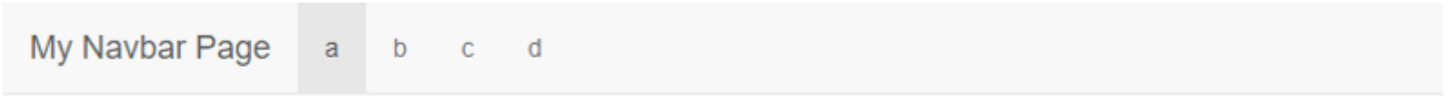
pageWithSidebar Layout

```
pageWithSidebar(
  headerPanel = headerPanel("Hello Shiny!"),

  sidebarPanel(
    sliderInput("n", "Sample size:", min= 0, max=150, value=50)
  ),

  mainPanel(
    plotOutput("iris_plot")
  )
)
```

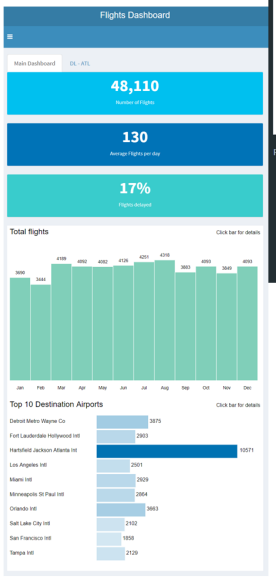
navbarPage Layout

```
navbarPage(title = "My Navbar Page",
           tabPanel("a"),
           tabPanel("b"),
           tabPanel("c"),
           tabPanel("d")
)
```

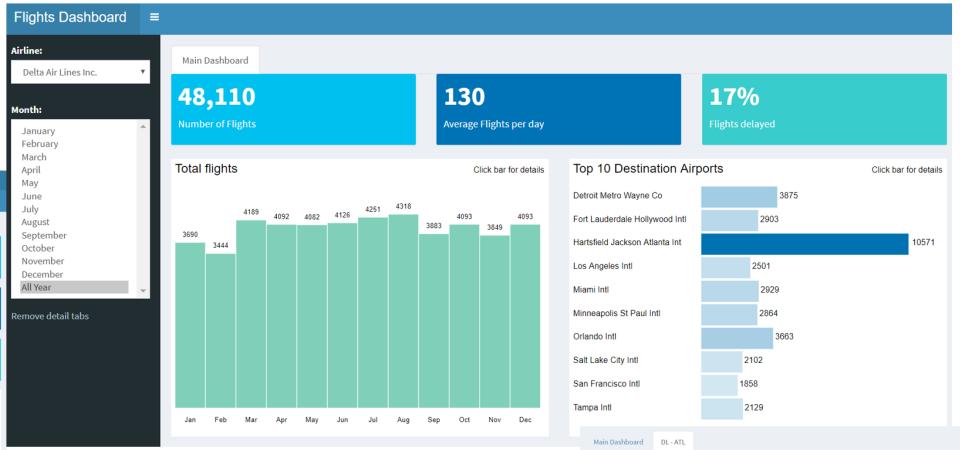
output:



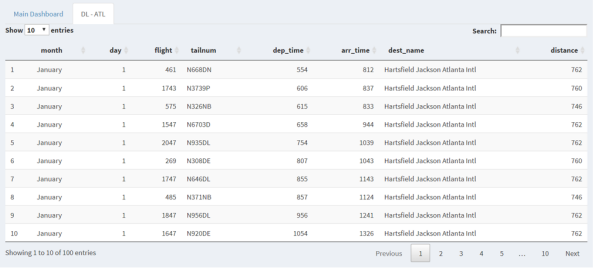
shinydashboard Layout



Mobile ready



Interactive dashboards



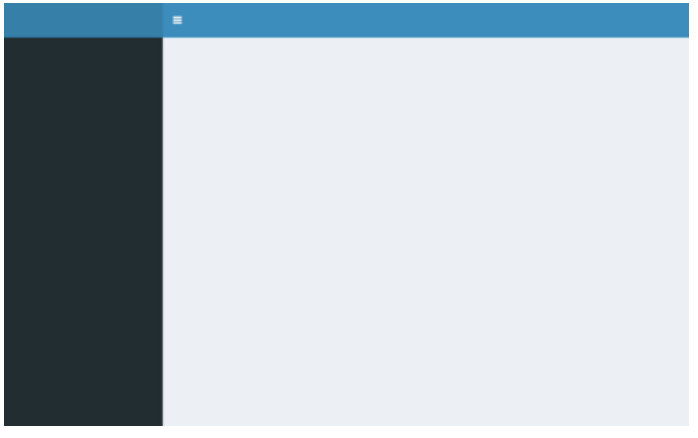
Drill-down reporting

source: db.rstudio.com/dashboards/

shinydashboard Layout

The template

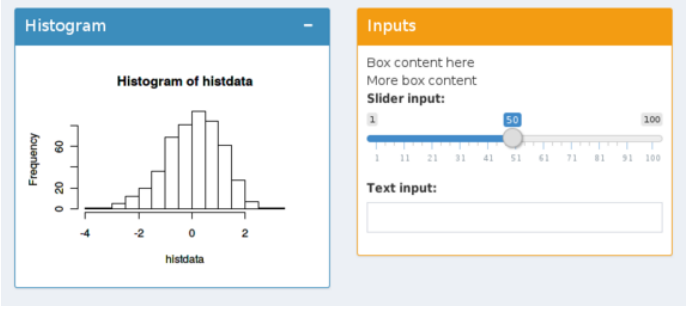
```
ui <- dashboardPage(
  dashboardHeader(),
  dashboardSidebar(),
  dashboardBody()
)

#server...
```



shinydashboard Layout

box

```
box(
  title = "Histogram", ...
),
box(
  title = "Inputs", ...
)
```

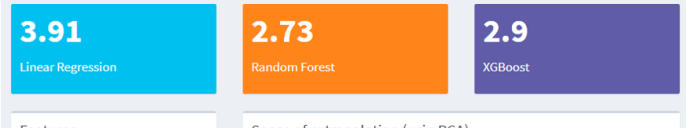


tabBox

```
tabBox(
  title = "Densities",
  tabPanel("Sepal.Length",..),
  tabPanel("Sepal.Width",...),
  tabPanel("Petal.Length",..),
  tabPanel("Petal.Width",...)
)
```

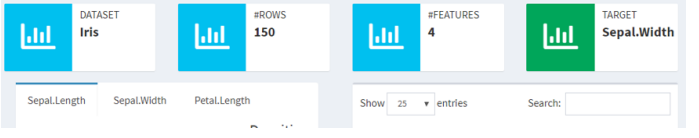
shinydashboard Layout

valueBox




```
renderValueBox({}) +
valueBoxOutput()
```

infoBox

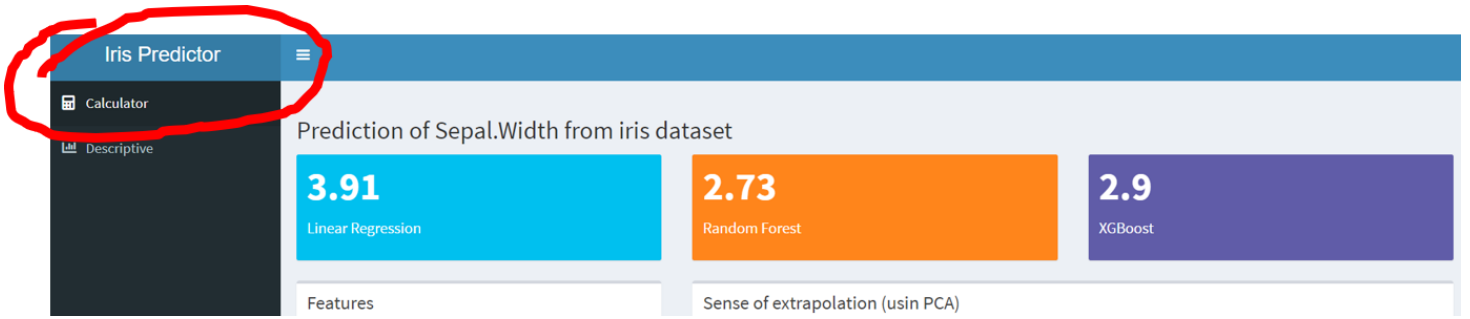



```
renderInfoBox({}) +
infoBoxOutput()
```

shinydashboard Layout

dashboardHeader

```
dashboardPage(
  dashboardHeader(title = "Iris Predictor"),

  dashboardSidebar(...),
  dashboardBody(...)
)

#server...
```



shinydashboard Layout

dashboardSidebar + sidebarMenu + menuItem

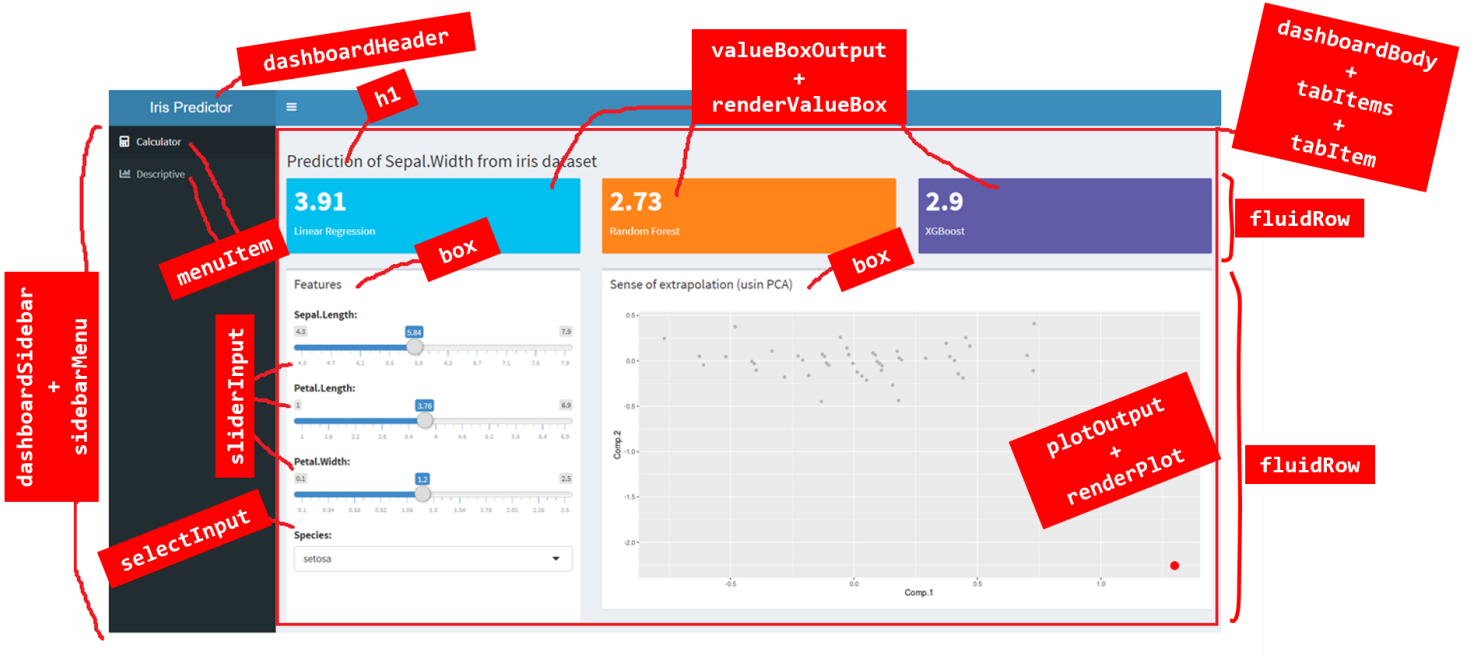
```
dashboardSidebar(
    sidebarMenu(
      menuItem("Calculator", tabName = "calc"),
      menuItem("Descriptive", tabName = "desc")
    )
  )
```

dashboardBody + tabItems + tabItem

```
dashboardBody(
    tabItems(
      tabItem(tabName = "calc",...),
      tabItem(tabName = "desc",...)
    )
)
```

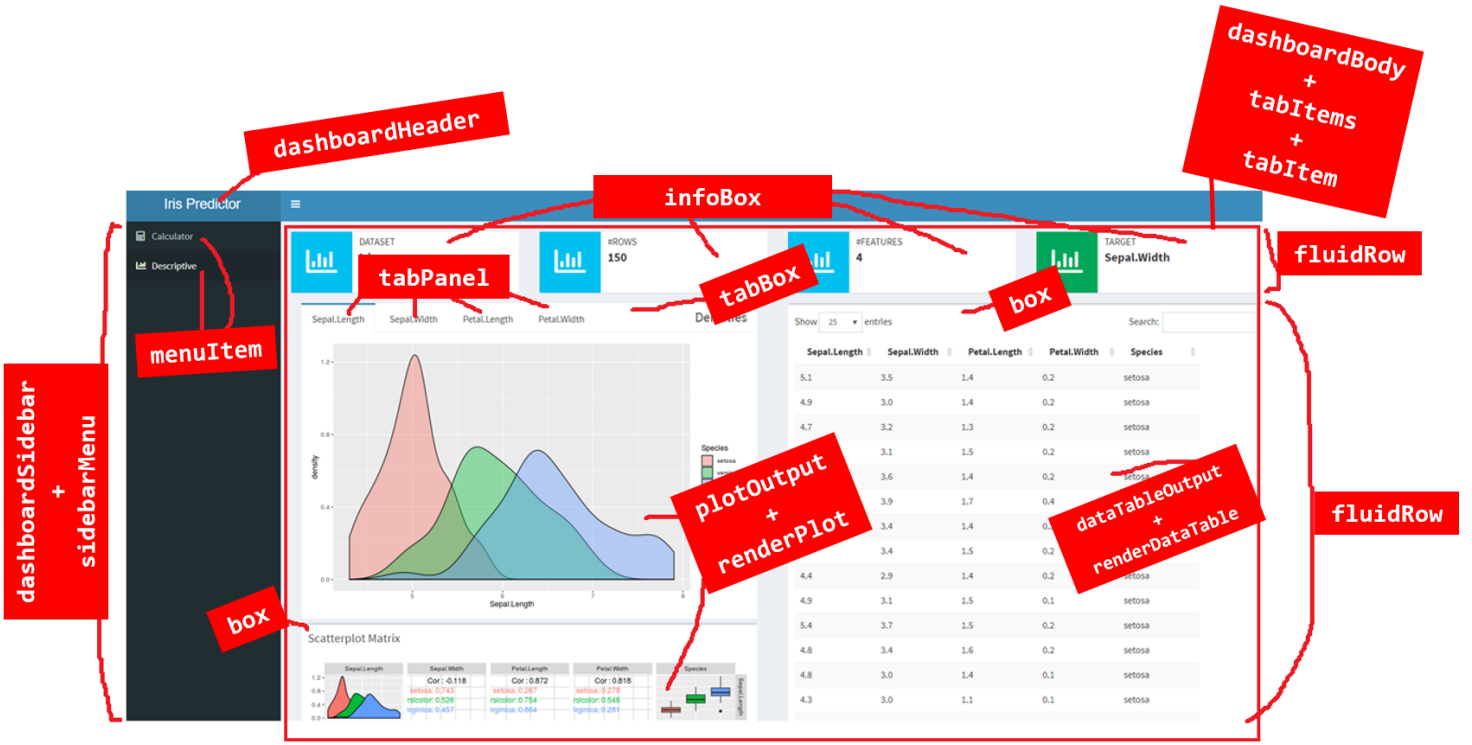
Exercise 1

Replicate the "Calculator" page of this dashboard (click me!).



Exercise 2 (hard!)

Replicate the "Descriptive" page of this dashboard (click me!).



Shinyapps.io

Shinyapps.io

To R!

Reference

This course is based on Garrett Grolemund's Shiny Tutorial.

Thank you!

athos.damiani@curso-r.com