

Tutorial R Shiny

17 de fevereiro de 2023

Hugo Costeira
A87976

João Silva
A87939

João Gouveia
A87995

Pedro
Martins
A87964

Supervisão:
Cecília Castro, DMAT, Universidade do Minho

Conteúdo

1	Introdução	6
1.1	Resumo	6
1.2	Shiny: definir aplicações web em R	6
1.3	A minha primeira aplicação	7
2	Começar no Rstudio	9
2.1	Uma pasta com um ficheiro	9
2.2	Uma pasta com um ficheiro	10
2.3	Uma pasta com dois ficheiros	10
2.4	Uma pasta com dois ficheiros	10
2.5	Dados/ficheiros adicionais	11
3	Interatividade e comunicação	11
3.1	Exemplo	12
3.2	Introdução processo	13
3.3	Aviso	13
3.4	Parte do UI (definição de input)	13
3.5	Parte do server (construção do output)	14
3.6	Parte do UI (visualização do output)	14
3.7	De volta ao processo	15
3.8	Partilha ui <-> server	15
4	Inputs	16
4.1	Global view	16
4.2	Numéricos	16
4.3	Carateres	17
4.4	Escolha única numa lista	17
4.5	Escolha múltipla numa lista	18
4.6	Checkbox simples	18
4.7	Múltiplas checkboxes	19
4.8	Botões Radio	19
4.9	Data <i>Código</i>	20
4.10	Data <i>Aplicação</i>	20
4.11	Período <i>Código</i>	20
4.12	Período <i>Aplicação</i>	21
4.13	Slider numérico: um valor	21
4.14	Slider numérico: intervalo	21
4.15	Importar um ficheiro	22
4.16	Botão de ação	22
4.17	Levando as coisas mais longe: construir um input	23

5	Outputs	23
5.1	Global view	23
5.2	Regras para definir outputs	23
5.3	Print	24
5.4	Text	24
5.5	Plot <i>Code</i>	24
5.6	Plot <i>App</i>	25
5.7	Table <i>Code</i>	25
5.8	Table <i>App</i>	25
5.9	DataTable <i>Code</i>	25
5.10	DataTable Aplicação	26
5.11	Definir elementos do UI no SERVER Processo	26
5.12	Definir elementos UI no SERVER Um exemplo simples	26
5.13	Levando as coisas mais longe: construir um output	27
6	Organizar a aplicação	27
6.1	sidebarLayout Definição	27
6.2	sidebarLayout Exemplo	28
6.3	navbarPage Definição	28
6.4	navbarPage Aplicação	28
6.5	navbarPage com navbarMenu	28
6.6	navbarPage Aplicação shiny	29
6.7	tabsetPanel Definição	29
6.8	tabsetPanel Exemplo	29
6.9	navlistPanel Definição	30
6.10	navlistPanel Exemplo	30
6.11	Layout da grelha Definição	30
6.12	Layout da grelha Aplicação shiny	30
6.13	wellPanel Definição	31
6.14	wellPanel Exemplo	31
6.15	Combinar estruturas Aplicação shiny	31
6.16	shinydashboard	32
7	Gráficos interativos	32
7.1	Introdução	32
7.2	Integração em shiny	33
7.3	Exemplos para as funções do server e do ui	33
7.4	Gráficos interativos: exemplo	33
7.5	Gráficos interativos: exemplo	34
7.6	Gráficos interativos: exemplo	34
7.7	Gráficos interativos: exemplo	35

8	Isolamento	35
8.1	Definição	35
8.2	Exemplo ui.R	35
8.3	Exemplo server.R	36
8.4	Exemplo Aplicação	36
9	Expressões reativas	37
9.1	Definição	37
9.2	Sem expressões reativas	37
9.3	Com uma expressão reativa	37
9.4	Notas	38
10	Observe & funções para atualizar	38
10.1	Observe & funções para atualizar	38
10.2	Exemplo para um input Aplicação	39
10.3	Exemplo para tabs Aplicação	39
10.4	Exemplo para um input ui.R	40
10.5	Exemplo para um input server.R	40
10.6	Exemplo para um input Aplicação	42
10.7	Exemplo para tabs ui.R	42
10.8	Exemplo para tabs server.R	43
10.9	Exemplo para tabs Aplicação	43
10.10	ObserveEvent	43
11	Painéis condicionais	44
11.1	Definição	44
11.2	Exemplo para um input	44
11.3	Exemplo para um input	45
12	Levando as coisas mais longe: HTML / CSS	45
12.1	Incluir HTML	45
12.2	Algumas tags interessantes	47
12.3	CSS: introdução	47
12.4	HTML / CSS ficheiro css externo	48
12.5	HTML / CSS css no header	48
12.6	HTML / CSS CSS num elemento	49
13	Levando as coisas mais longe: algumas “regras” importantes	50
13.1	Boa abordagem	50
13.2	Boa abordagem	50
14	Levando as coisas mais longe: debugging	51
14.1	Imprimir na consola	51
14.2	Imprimir na consola	51
14.3	Início manual de um browser	51
14.4	Início manual de um browser	52
14.5	Início automático de um browser	52

14.6	Modo “showcase”	52
14.7	Mode “showcase”	53
14.8	Log reativo	53
14.9	Log reativo	54
14.10	Comunicação no servidor	54
14.11	Comunicação no servidor	54
14.12	Tracking the erros	55
14.13	Tracking de erros	55
15	Referências	55
15.1	Tutoriais / Exemplos	55
15.2	Código da nossa aplicação	55

1 Introdução

No âmbito da UC Projeto do 3º ano do Curso de Licenciatura em Ciências da Computação, foi desenvolvida uma aplicação que permite ao utilizador executar diversas operações de análise inicial de dados, tais como gráficos e tabelas. O desenvolvimento desta aplicação *web*, interativa, baseou-se exclusivamente na utilização de ferramentas incluídas no programa R Studio.

Neste documento adicional, apresenta-se um tutorial para o desenvolvimento de aplicações *web*, interativas, a partir da *package* *shiny*.

1.1 Resumo

Os seguintes itens são desenvolvidos ao longo do documento.

- Breve Introdução à *shiny*
- Começar no Rstudio
- Interatividade e comunicação
- Inputs & outputs
- Organização da página *web*
- Gráficos interativos
- HTML / CSS
- Extensões

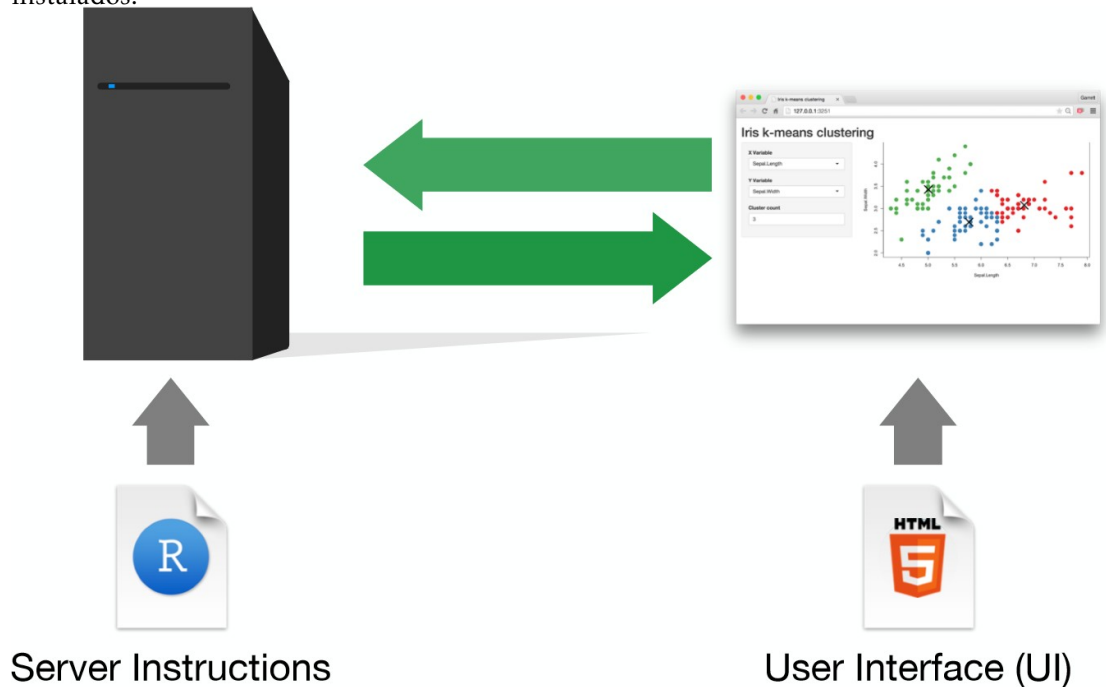
1.2 Shiny: definir aplicações web em R

Shiny é uma *package* do **R** que facilita a construção de aplicações interativas *web* com recurso ao programa **R**.

1. Não requer experiência em linguagens *web*, como HTML, CSS ou Javascript;
2. Combina o poder de data science do **R** com interatividade *web*;
3. Permite desenvolver aplicações locais;
4. Permite distribuir aplicações para outros utilizadores: **shiny-server**, **shinyapps.io**, **shinyproxy**.

<http://shiny.rstudio.com/>
<http://www.shinyapps.io/>
<https://www.shinyproxy.io/>
<https://www.rstudio.com/products/shiny/shiny-server/>.

Uma aplicação shiny requer apenas um computador/server com **R** e **RStudio** instalados.



© CC 2015 RStudio, Inc.

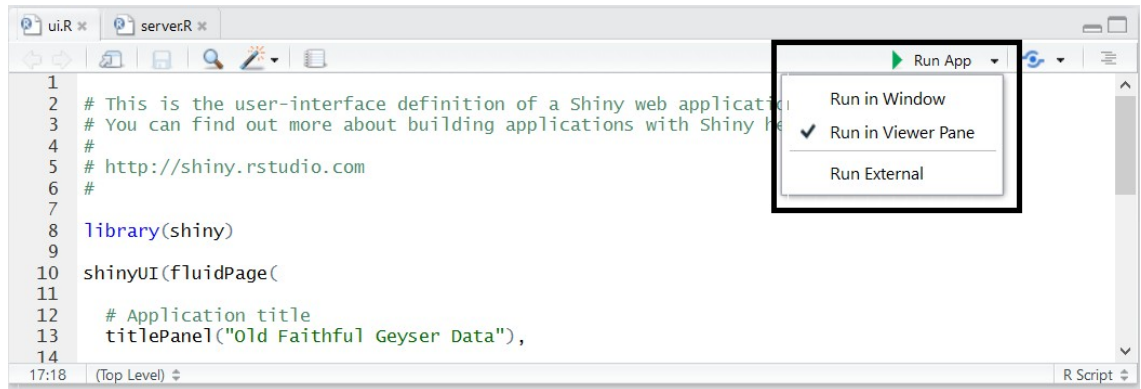
1.3 A minha primeira aplicação

- Para começar o desenvolvimento de uma aplicação com o **RStudio**, deve ser iniciado um **new project** (novo projeto)
 - File -> New Project -> New Directory -> Shiny Web Application
Ou
 - File -> New File -> Shiny Web App -> Multiple File

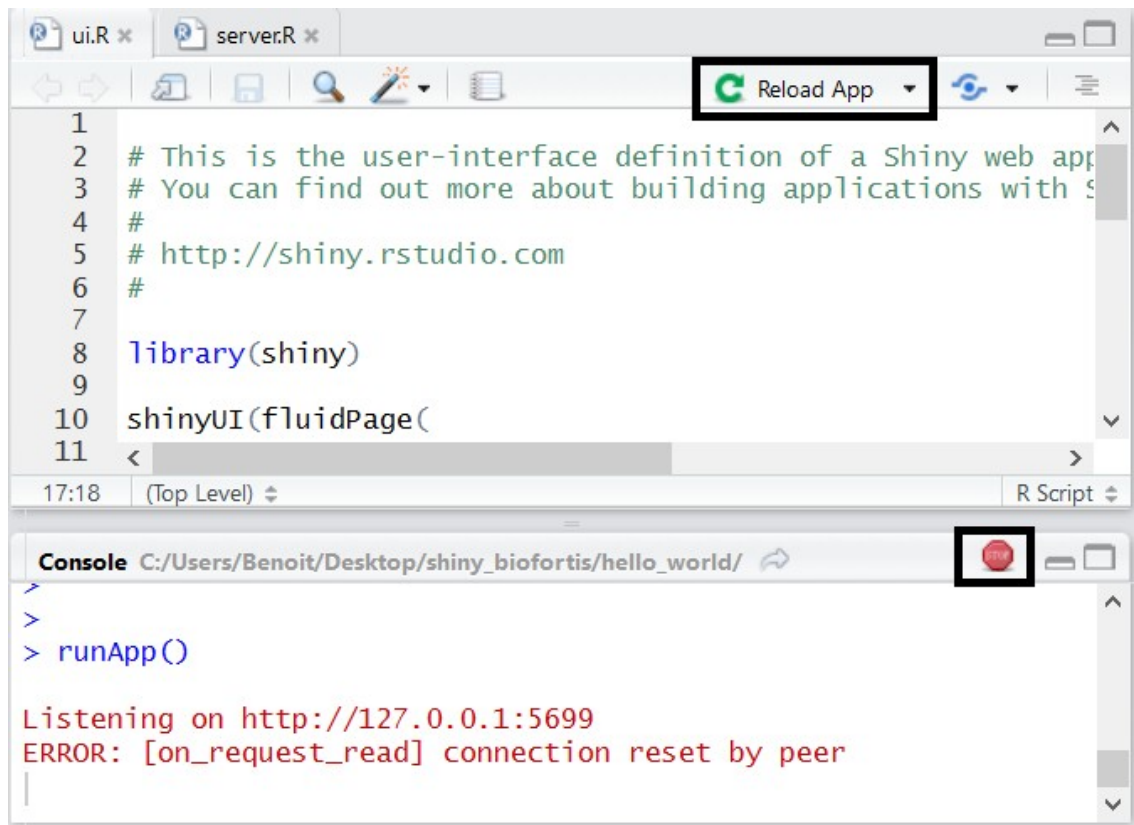
Qualquer aplicação shiny consta de duas funções, `ui()` (que define o layout da aplicação) e `server()` (que faz todos os cálculos e promove a reatividade das funções) que podem estar definidas em dois scripts diferentes, `{ui.R}` and `{server.R}` ou no mesmo script `app.R`. Os nomes dos scripts são convenções não essenciais, mas úteis.

- Comandos facilitadores:
 - Correr a aplicação: botão **Run app**

- Atualizar: botão **Reload app**
- Parar: botão **Stop**



- **Run in window:** nova janela, usando o ambiente **RStudio**
- **Run in Viewer Pane:** *Visualizador* de tabs do **RStudio**
- **Run External:** no browser predefinido

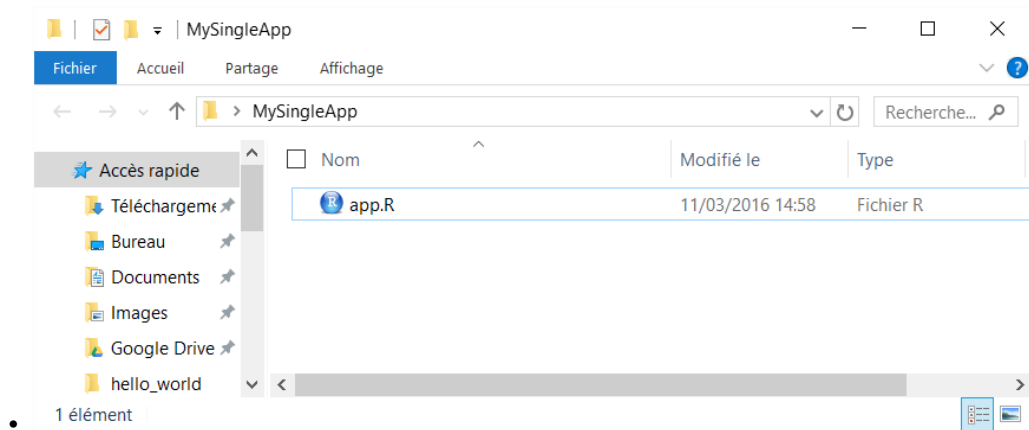


2 Começar no Rstudio

2.1 Uma pasta com um ficheiro

Convenções:

- guardar como **app.R**
- terminar com o comando `shinyApp()`
- for **small applications**



2.2 Uma pasta com um ficheiro

```
library(shiny)
ui <- fluidPage(
  sliderInput(inputId = "num", label = "Choose a number",
    value = 25, min = 1, max = 100),
  plotOutput("hist")
)
server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
}
shinyApp(ui = ui, server = server)
```

2.3 Uma pasta com dois ficheiros

Convenções:

- interface do utilizador (layout e aparência) em **ui.R**
- instruções **R** necessárias para construir a aplicação em **server.R**
- a melhor estrutura para **aplicações complexas**

2.4 Uma pasta com dois ficheiros

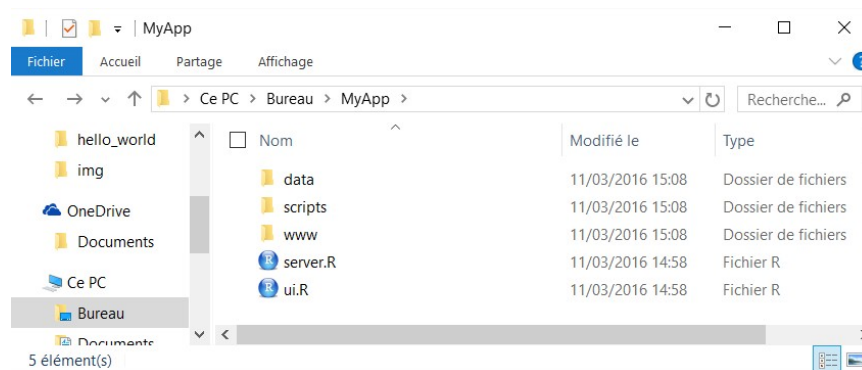
ui.R

```
library(shiny)
fluidPage(
  sliderInput(inputId = "num", label = "Choose a number",
    value = 25, min = 1, max = 100),
  plotOutput("hist")
)
```

server.R

```
library(shiny)
function(input, output) {
  output$hist <- renderPlot({hist(rnorm(input$num))})
}
```

2.5 Dados/ficheiros adicionais



3 Interatividade e comunicação

```
shinyApp(
  ui = fluidPage(
    titlePanel("Hello Shiny!"),
    sidebarLayout(
      sidebarPanel(
        sliderInput("bins",
          "Number of bins:",
          min = 1,
          max = 50,
          value = 30)
      ),
      mainPanel(
        plotOutput("distPlot")
      )
    )
  )
)
```

```

    )
  ),
  server = function(input, output) {

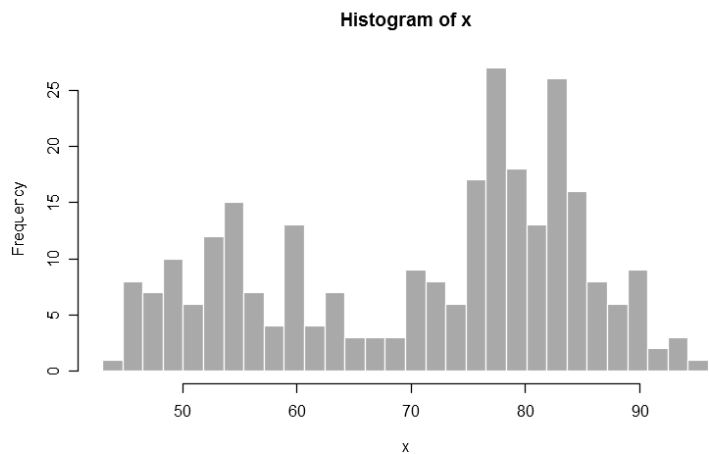
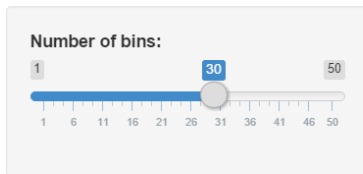
    # A expressao que gera um histograma # e' argumento da funcao
    renderPlot # onde render diz respeito 'a # reatividade,
    "reactive", que permite # que a funcao seja sempre novamente
    # executada quando muda o input
    # o comando que executa o histograma # hist, faz parte da
    expressao

    output$distPlot <- renderPlot({
      x <- faithful[, 2] # dados #Faithful Geysers data
      bins <- seq(min(x), max(x), length.out = input$bins + 1)
      # draw the histogram with the specified number of bins
      hist(x, breaks = bins, col = 'darkgray', border = 'white')
    })
  })
}

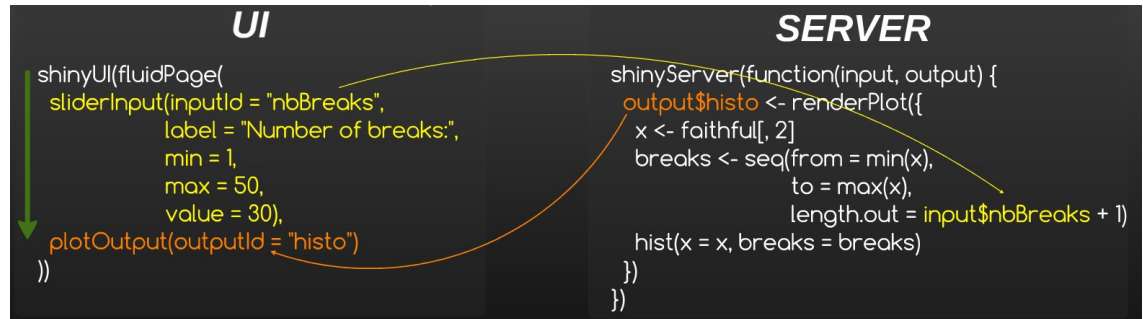
```

3.1 Exemplo

Hello Shiny!



3.2 Introdução | processo



- **ui**: organizar inputs e outputs
- **server**: computar os outputs (de acordo com os inputs)

O server e o ui comunicam através de inputs e outputs

- Por defeito, um output é atualizado quando um input se altera

3.3 Aviso

Definição da interface do utilizador: UI

- definição dos inputs
- arquitetura da página, com localização dos outputs

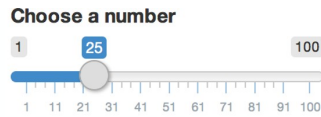
server/parte computacional: SERVER

- definição e computação dos outputs

3.4 Parte do UI (definição de input)

Dois tipos de itens em UI

- `xxInput(inputId = . . . , . . .)`:
 - para um elemento que requer uma ação do utilizador
 - disponível no server através do seu ID `input$inputID`



```
sliderInput(inputId = "num", label = "Choose a number", ...)
```



3.5 Parte do server (construção do output)

- `renderXX({expr})`:
 - computar e retornar um output (que pode depender dos inputs) com comandos R clássicos

```
renderPlot({ hist(rnorm(100)) })
```

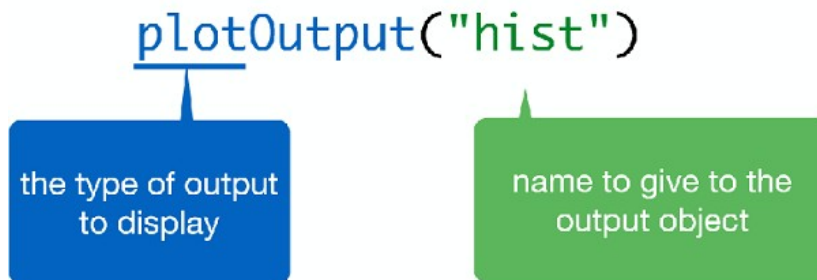


- Exemplo:

```
output$hist <- renderPlot({
  #commands to build the histogram
})
```

3.6 Parte do UI (visualização do output)

- `xxOutput(outputId = . . .)`:
 - referir a um output criado no servidor
 - comumente para gráficos e/ou tabelas



3.7 De volta ao processo

<i>UI</i>	<i>SERVER</i>
<pre>shinyUI(fluidPage(sliderInput(inputId = "nbBreaks", label = "Number of breaks:", min = 1, max = 50, value = 30), plotOutput(outputId = "hist")))</pre>	<pre>shinyServer(function(input, output) { output\$histo <- renderPlot({ x <- faithful[, 2] breaks <- seq(from = min(x), to = max(x), length.out = input\$nbBreaks + 1) hist(x = x, breaks = breaks) }) })</pre>

Está mais claro?

3.8 Partilha ui <-> server

O server e o ui só comunicam através de inputs e outputs

- Podemos adicionar outro ficheiro **global.R** se quisermos partilhar elementos (bases de dados, funções. . .) entre o **server** e o **UI**
- Todos os elementos no **global.R** estão disponíveis para o **ui.R** e o **server.R**
- O script **global.R** corre apenas uma vez, no início do processo.

4 Inputs

4.1 Global view

Buttons



`actionButton()`
`submitButton()`

Single checkbox



`checkboxInput()`

Checkbox group



`checkboxGroupInput()` `dateInput()`

Date input

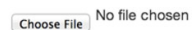


Date range



`dateRangeInput()`

File input



`fileInput()`

accompany other widgets.

`numericInput()`

Password Input



`passwordInput()`

Radio buttons



`radioButtons()`

Select box



`selectInput()`

Sliders



`sliderInput()`

Text input



`textInput()`

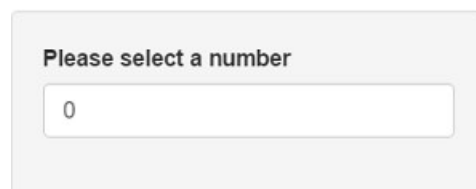
4.2 Numéricos

- Função:

```
numericInput(inputId, label, value, min = NA, max = NA, step = NA)
```

- Example:

```
textInput(inputId = "id_txt", label = "Enter a text", value = "")
```



Value: `[1] 0`

Class: `integer`

4.3 Carateres

- Função:

```
textInput(inputId, label, value = "")
```

- Exemplo:

```
textInput(inputId = "id_txt", label = "Enter a text", value = "")
```



Value: [1] "test"

Class: character

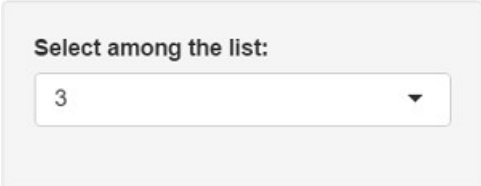
4.4 Escolha única numa lista

- Função:

```
selectizeInput(inputId, label, choices, selected = NULL, multiple  
= FALSE,  
selectize = TRUE, width = NULL, size = NULL)
```

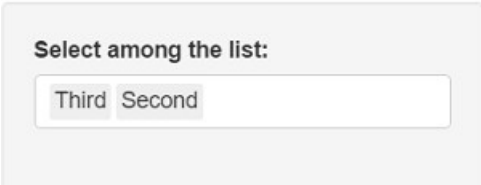
- Exemplo:

```
selectizeInput(inputId = "id_sel1", label = "Select among the  
list: ", selected = 3,  
choices = c(1:3))
```



Value: [1] "3"

Class: character



Value: [1] "3" "2"

Class: character

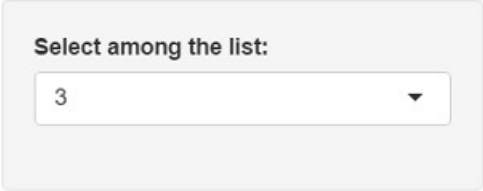
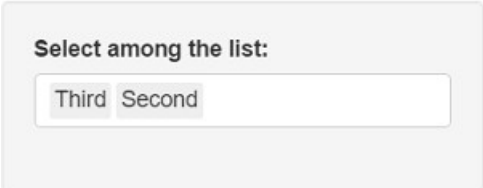
4.5 Escolha múltipla numa lista

- Função:

```
selectInput(inputId, label, choices, selected = NULL, multiple = FALSE,
            selectize = TRUE, width = NULL, size = NULL)
```

- Exemplo:

```
selectInput(inputId = "id_sel2", label = "Select among the list:",
            selected = 3,
            choices = c("First" = 1, "Second" = 2, "Third" = 3),
            multiple = TRUE)
```

	Value:	<input type="text" value='[1] "3"'/>
	Class:	<input type="text" value="character"/>
	Value:	<input type="text" value='[1] "3" "2"'/>
	Class:	<input type="text" value="character"/>

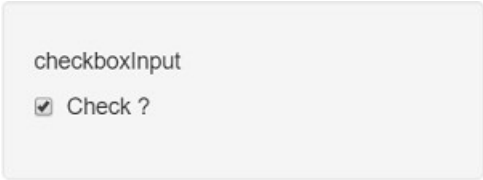
4.6 Checkbox simples

- Função:

```
checkboxInput(inputId, label, value = FALSE)
```

- Exemplo:

```
checkboxInput(inputId = "id_check_1", label = "Check?")
```

	Value:	<input type="text" value="[1] TRUE"/>
	Class:	<input type="text" value="logical"/>

4.7 Múltiplas checkboxes

- Função:

```
checkboxGroupInput(inputId, label, choices, selected = NULL,  
                inline = FALSE)
```

- Exemplo:

```
checkboxGroupInput(inputId = "id_check_2", label = "Please  
select", selected = 3,  
                choices = c("First" = 1, "Second" = 2, "Third" = 3))
```

Please select

First

Second

Third

Value:

Class:

4.8 Botões Radio

- Função:

```
radioButtons(inputId, label, choices, selected = NULL, inline =  
             FALSE)
```

- Exemplo:

```
radioButtons(inputId = "id_radio", label = "Select one",  
             choices = c("First" = 1, "Second" = 2, "Third" = 3),  
             selected = 3)
```

Select one

First

Second

Third

Value:

Class:

4.9 Data | Código

- Função:

```
dateInput(inputId, label, value = NULL, min = NULL, max = NULL,  
  format = "yyyy-mm-dd",  
  startview = "month", weekstart = 0, language = "en")
```

- Exemplo:

```
dateInput(inputId = "id_date", label = "Please enter a date",  
  value = "09/10/2020",  
  format = "dd/mm/yyyy", startview = "month", weekstart = 0,  
  language = "fr")
```

4.10 Data | Aplicação

<div style="border: 1px solid #ccc; padding: 5px; width: fit-content;"><p>Please enter a date</p><input type="text" value="07/12/2015"/></div>	Value:	<div style="border: 1px solid #ccc; padding: 2px; width: fit-content;">[1] "2015-12-07"</div>
	Class:	<div style="border: 1px solid #ccc; padding: 2px; width: fit-content;">Date</div>

4.11 Período | Código

- Função:

```
dateRangeInput(inputId, label, start = NULL, end = NULL, min =  
  NULL, max = NULL,  
  format = "yyyy-mm-dd", startview = "month", weekstart = 0,  
  language = "en", separator = " to ")
```

- Exemplo:

```
dateRangeInput(inputId = "id_daterange", label = "Please Select a  
  date range",  
  start = "2020-10-04", end = "2020-10-18", format =  
  "yyyy-mm-dd",  
  language = "en", separator = " to ")
```

4.12 Período | Aplicação



Value:

[1] "2015-01-01" "2015-08-12"

Class:

Date

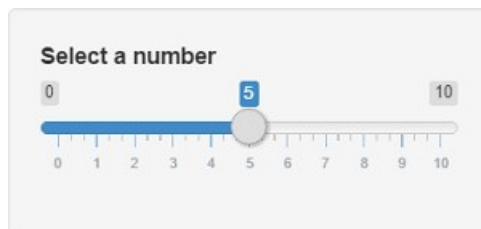
4.13 Slider numérico: um valor

- Função:

```
sliderInput(inputId, label, min, max, value, step = NULL, round = FALSE,
            format = NULL, locale = NULL, ticks = TRUE, animate = FALSE,
            width = NULL, sep = ",", pre = NULL, post = NULL)
```

- Exemplo:

```
sliderInput(inputId = "id_slider", label = "Select a number", min
            = 0, max = 10,
            value = 5, step = 1)
```



Value:

[1] 5

Class:

integer

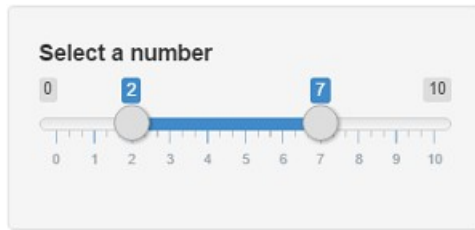
4.14 Slider numérico: intervalo

- Função:

```
sliderInput(inputId, label, min, max, value, step = NULL, round = FALSE,
            format = NULL, locale = NULL, ticks = TRUE, animate = FALSE,
            width = NULL, sep = ",", pre = NULL, post = NULL)
```

- Exemplo:

```
sliderInput(inputId = "id_slider2", label = "Select a number", min
            = 0, max = 10,
            value = c(2,7), step = 1)
```



Value:

Class:

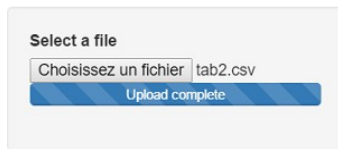
4.15 Importar um ficheiro

- Função:

```
fileInput(inputId, label, multiple = FALSE, accept = NULL)
```

- Exemplo:

```
fileInput(inputId = "id_file", label = "Select a file")
```



Value:

	name	size	type	datapath
1	tab2.csv	40	application/vnd.ms-excel	C:\Users\Benoit\AppData

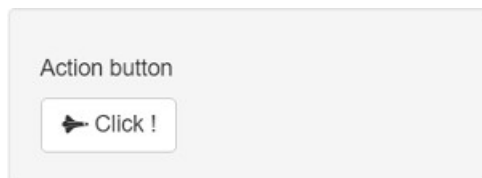
4.16 Botão de ação

- Função:

```
actionButton(inputId, label, icon = NULL, ...)
```

- Exemplo:

```
actionButton(inputId = "id_action", label = "Click !",  
            icon = icon("hand-spock-o"))
```



Value:

Class:

4.17 Levando as coisas mais longe: construir um input

Requer competências em HTML/CSS/JavaScript

Tutorial: <http://shiny.rstudio.com/articles/building-inputs.html>

Two examples:

- <http://shiny.rstudio.com/gallery/custom-input-control.html>
- <http://shiny.rstudio.com/gallery/custom-input-bindings.html>

5 Outputs

5.1 Global view

server fonction	ui fonction	type de sortie
renderDataTable()	dataTableOutput()	une table interactive
renderImage()	imageOutput()	une image sauvegardée
renderPlot()	plotOutput	un graphique R
renderPrint()	verbatimTextOutput()	affichage type console R
renderTable()	tableOutput()	une table statique
renderText()	textOutput()	une chaîne de caractère
renderUI()	uiOutput()	un élément de type UI

5.2 Regras para definir outputs

- atribuir o output na lista **output**, use um bom nome para o identificar no UI
- use uma função **renderXX({expr})**

```
#ui.R
selectInput("lettre", "Lettres:", LETTERS[1:3])
verbatimTextOutput(outputId = "selection")
#server.R
output$selection <- renderPrint({input$lettre})
```

5.3 Print

- **ui.r:**

```
verbatimTextOutput(outputId = "text")
```

- **server.r:**

```
output$texte <- renderPrint({  
  c("Hello shiny !")  
})
```

```
[1] "Hello shiny !"
```

5.4 Text

- **ui.r:**

```
textOutput(outputId = "texte")
```

- **server.r:**

```
output$texte <- renderText({  
  c("Hello shiny !")  
})
```

Hello shiny !

5.5 Plot | Code

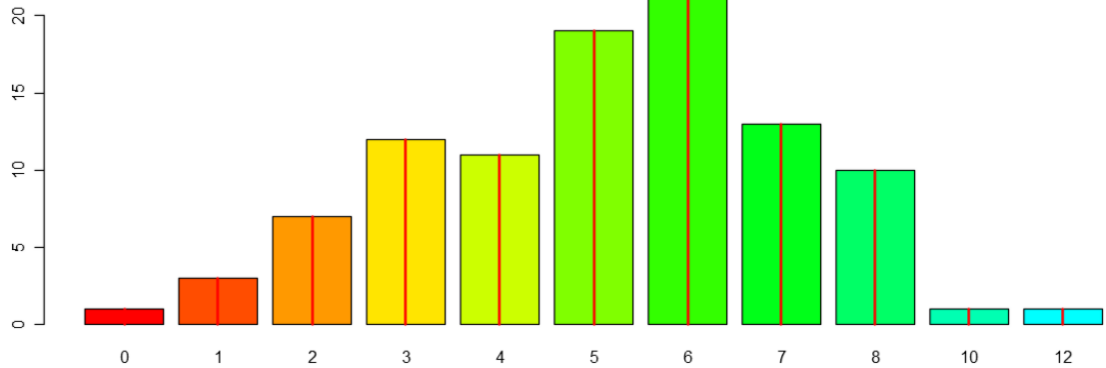
- **ui.r:**

```
plotOutput("myplot")
```

- **server.r:**

```
output$myplot <- renderPlot({  
  require(grDevices) # for colours  
  tN <- table(Ni <- stats::rpois(100, lambda = 5))  
  r <- barplot(tN, col = rainbow(20))  
  lines(r, tN, type = "h", col = "red", lwd = 2)  
})
```

5.6 Plot | *App*



5.7 Table | *Code*

- **ui.r:**

```
tableOutput(outputId = "table")
```

- **server.r:**

```
data("iris")  
output$table <- renderTable({  
  iris[1:5, ]  
})
```

5.8 Table | *App*

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.10	3.50	1.40	0.20	setosa
2	4.90	3.00	1.40	0.20	setosa
3	4.70	3.20	1.30	0.20	setosa
4	4.60	3.10	1.50	0.20	setosa
5	5.00	3.60	1.40	0.20	setosa

5.9 DataTable | *Code*

- **ui.r:**

```
dataTableOutput(outputId = "dataTable")
```

- **server.r:**

```
data("iris")
output$dataTable <- renderDataTable({
  iris
})
```

5.10 DataTable | Aplicação

Show entries Search:

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa

Showing 1 to 5 of 5 entries Previous **1** Next

5.11 Definir elementos do UI no SERVER | Processo

Às vezes pode ser interessante definir inputs no server

Podemos fazer isso com o uiOutput e o renderUI

5.12 Definir elementos UI no SERVER | Um exemplo simples

- **ui.r:**

```
uiOutput(outputId = "columns")
```

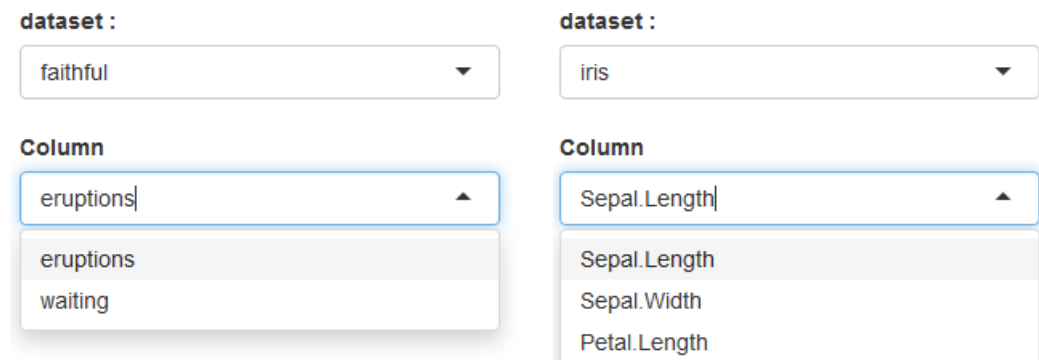
- **server.r:**

```
output$columns <- renderUI({
  selectInput(inputId = "sel_col", label = "Column", choices =
    colNames(data))
})
```

```

shinyApp(
  ui = fluidPage(
    selectInput(inputId = "my_data", label = "dataset : ",
               choices = c("iris", "faithful")),
    uiOutput(outputId = "columns")
  ),
  server = function(input, output) {
    data <- reactive(get(input$my_data, "package:datasets"))
    output$columns <- renderUI({
      selectInput(inputId = "sel_col", label = "Column", choices =
                  colnames(data()))
    })
  })

```



5.13 Levando as coisas mais longe: construir um output

Requer algumas competências em HTML/CSS/JavaScript

Tutorial: <http://shiny.rstudio.com/articles/building-outputs.html>

6 Organizar a aplicação

6.1 sidebarLayout | Definição

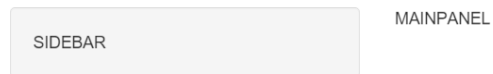
O modelo básico sidebarLayout divide a página em duas colunas e deve conter:

- **sidebarPanel**, parte esquerda, geralmente para os inputs
- **mainPanel**, parte direita, geralmente para os outputs

```
shinyUI(
  fluidPage(
    titlePanel("Old Faithful Geyser Data"), # title
    sidebarLayout(
      sidebarPanel("Elements of sidebar (separated with
commas)"),
      mainPanel("Elements of panel (separated with commas)")
    )
  )
)
```

6.2 sidebarLayout | Exemplo

My first app

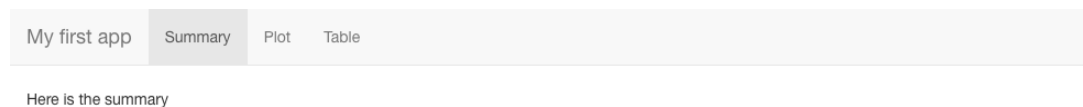


6.3 navbarPage | Definição

Use uma página da barra de navegação com `navbarPage` e `tabPanel`:

```
shinyUI(
  navbarPage(
    title = "My first app",
    tabPanel(title = "Summary",
"Here is the summary"),
    tabPanel(title = "Plot",
"some charts"),
    tabPanel(title = "Table",
"some tables")
  )
)
```

6.4 navbarPage | Aplicação



6.5 navbarPage | com navbarMenu

Podemos adicionar um segundo nível para a navegação com `navbarMenu`:

```
shinyUI(
```

```

navbarPage(
  title = "My first app",
  tabPanel(title = "Summary",
           "Here is the summary"),
  tabPanel(title = "Plot",
           "some charts"),
  navbarMenu("Table",
            tabPanel("a table"),
            tabPanel("another table"))
)
)
)

```

6.6 navbarPage | Aplicação shiny



6.7 tabsetPanel | Definição

Geralmente, podemos criar páginas da barra de navegação em qualquer lado com `tabsetPanel` & `tabPanel`:

```

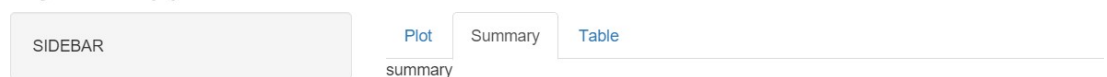
sidebarLayout(
  sidebarPanel("SIDEBAR"),
  mainPanel(
    tabsetPanel(
      tabPanel("Plot", plotOutput("plot")),
      tabPanel("Summary", verbatimTextOutput("summary")),
      tabPanel("Table", tableOutput("table"))
    )
  )
)
)

```

- `navbarPage`: criar tabs na aplicação
- `tabsetPanel`: criar tabs numa estrutura da aplicação

6.8 tabsetPanel | Exemplo

My first app

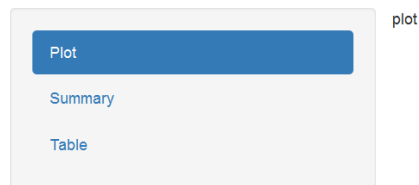


6.9 navlistPanel | Definição

Uma alternativa a `tabsetPanel` para obter a posição vertical em vez de horizontal: `navlistPanel`

```
shinyUI(fluidPage(  
  navlistPanel(  
    tabPanel("Plot", plotOutput("plot")),  
    tabPanel("Summary", verbatimTextOutput("summary")),  
    tabPanel("Table", tableOutput("table"))  
  )  
))
```

6.10 navlistPanel | Exemplo



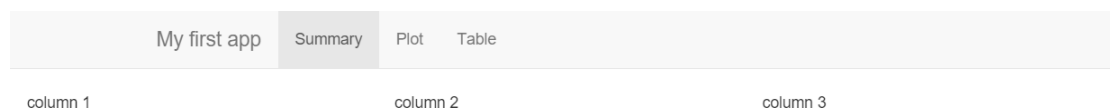
6.11 Layout da grelha | Definição

Defina a sua própria organização com `fluidRow()` e `column()`

- quaisquer linhas podem ser divididas em 12 colunas
- o tamanho da página ajusta-se automaticamente ao número de linhas/-colunas.

```
tabPanel(title = "Summary",  
  # A fluid row can contain from 0 to 12 columns  
  fluidRow(  
    # A column is defined necessarily  
    # with its argument "width"  
    column(width = 4, "column 1"),  
    column(width = 4, "column 2"),  
    column(width = 4, "column 3"),  
  )  
))
```

6.12 Layout da grelha | Aplicação shiny



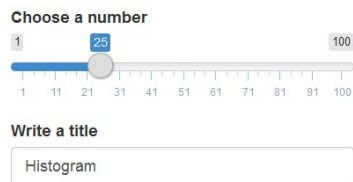
6.13 wellPanel | Definição

Pode-se obter um fundo cinzento com `wellPanel`:

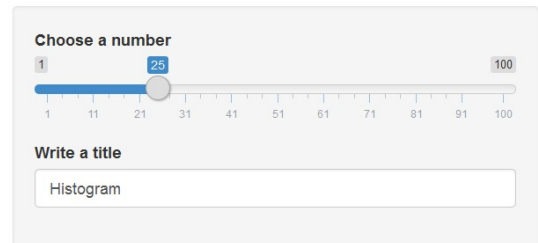
```
fluidRow(  
  column(6,  
    h2("Without wellPanel"), # title  
    sliderInput("num", "Choose a number", value = 25, min = 1, max =  
      100),  
    textInput("title", value = "Histogram", label = "Write a title")  
  ),  
  column(6,  
    h2("With wellPanel"), # title  
    wellPanel(  
      sliderInput("num", "Choose a number", value = 25, min = 1,  
        max = 100),  
      textInput("title", value = "Histogram", label = "Write a  
        title")  
    )  
  )  
)
```

6.14 wellPanel | Exemplo

Without wellPanel

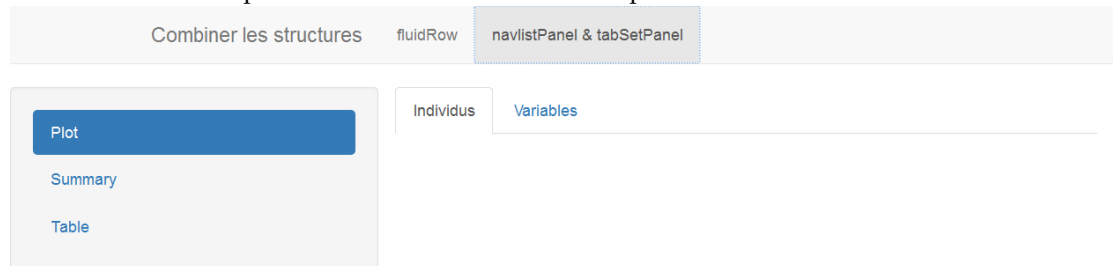


With wellPanel



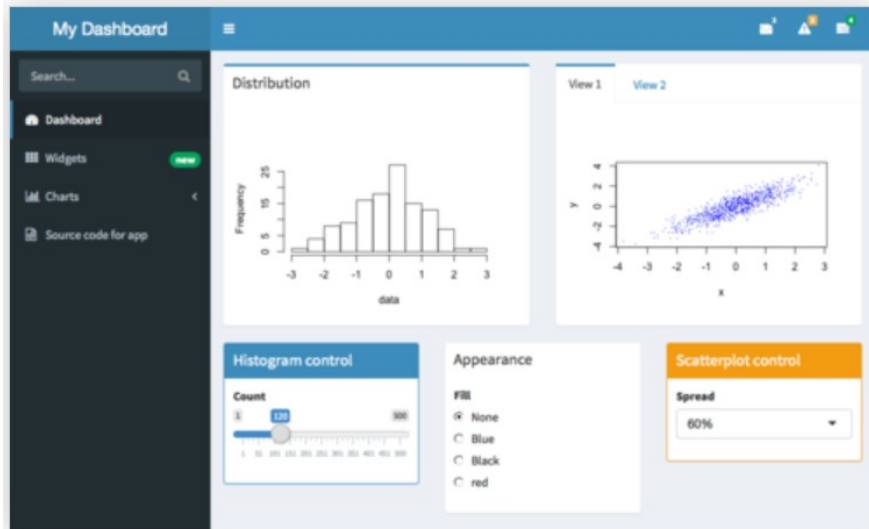
6.15 Combinar estruturas | Aplicação shiny

Todas as estruturas podem ser usadas ao mesmo tempo!



6.16 shinydashboard

O package [shinydashboard](#) tem outras funções para definir dashboards:



<https://rstudio.github.io/shinydashboard/>

7 Gráficos interativos

7.1 Introdução

Desde a criação do package [htmlwidgets](#), mais e mais possibilidades de javascript estão disponíveis com o **R**:

- [dygraphs](#) (séries temporais)
- [DT](#) (tabelas interativas)
- [Leaflet](#) (mapas)
- [d3heatmap](#)
- [rAmCharts](#)
- [visNetwork](#)
- ...

Pode ver [esta galeria](#)

7.2 Integração em shiny

Todas estas packages podem ser usadas no **shiny**. De facto, elas possuem as duas funções requeridas:

- **renderXX**
- **xxOutput**

```
# Server
output$dygraph <- renderDygraph({
  dygraph(predicted(), main = "Predicted Deaths/Month")
})
# Ui
dygraphOutput("dygraph")
```

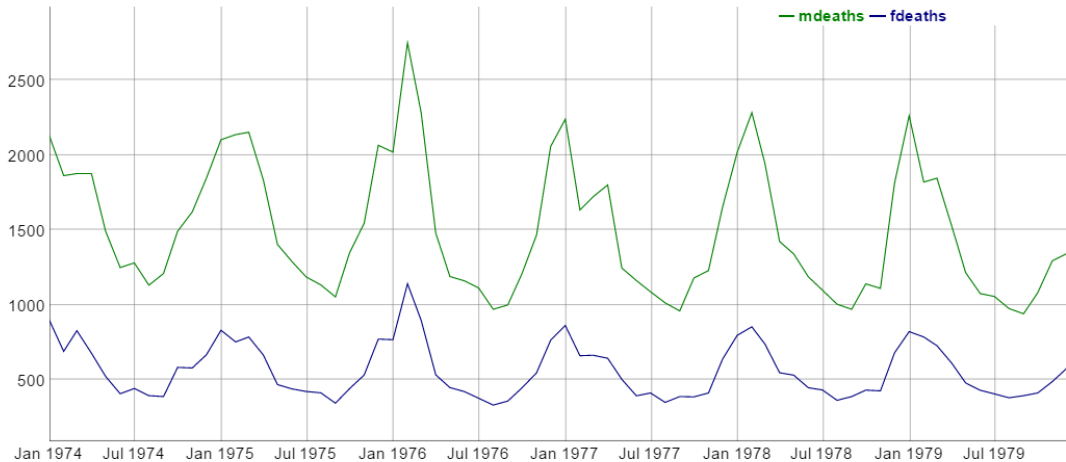
Um exemplo com a package [dygraphs](#) :

7.3 Exemplos para as funções do server e do ui

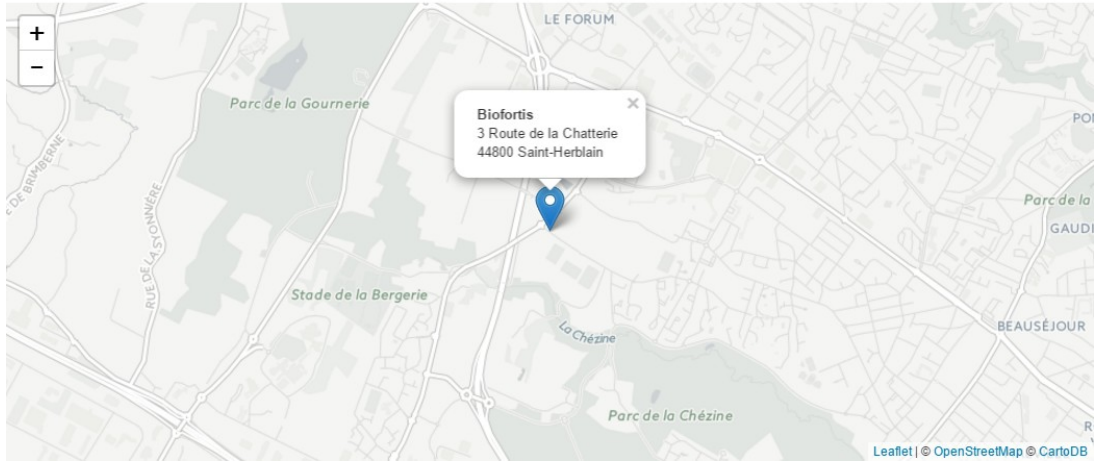
Package	função do server	função do ui
dygraph	renderDygraph	dygraphOutput
rAmcharts	renderAmCharts	amChartsOutput
leaflet	renderLeaflet	leafletOutput
plotly	renderPlotly	plotlyOutput
visNetwork	renderVisNetwork	visNetworkOutput

7.4 Gráficos interativos: exemplo

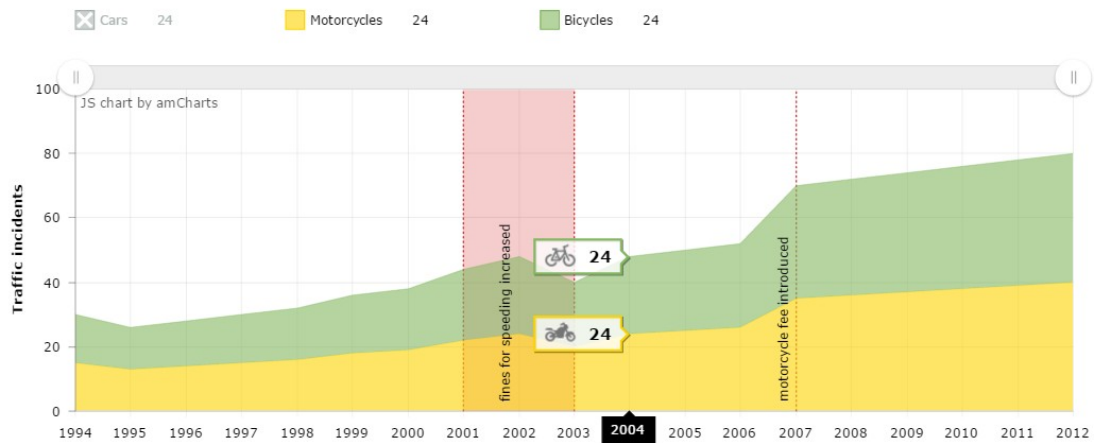
dygraphs



7.5 Gráficos interativos: ejemplo leaflet

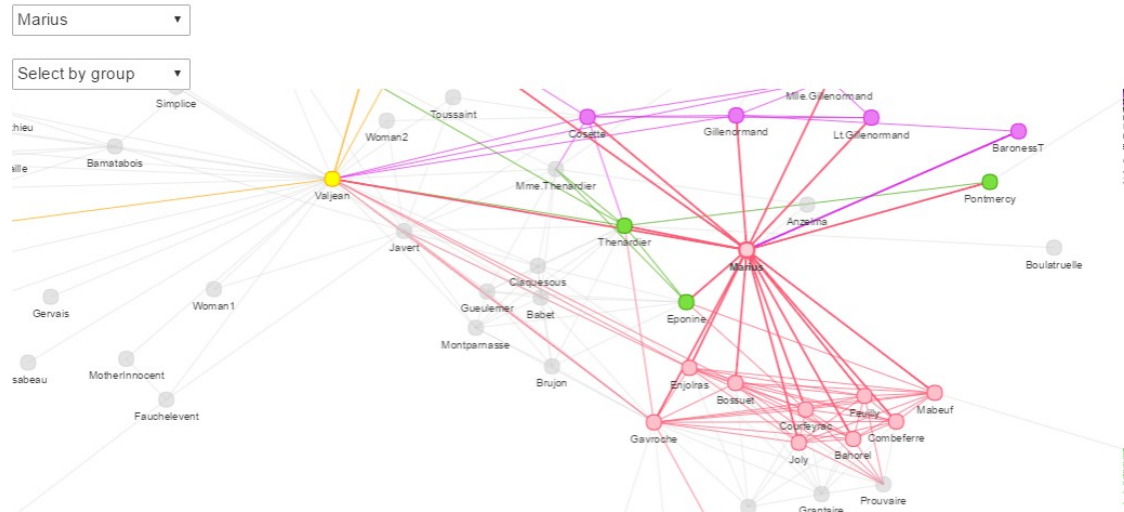


7.6 Gráficos interativos: ejemplo rAmCharts



7.7 Gráficos interativos: exemplo

visNetwork



8 Isolamento

8.1 Definição

- Por defeito, outputs e expressões reativas são atualizadas mal o utilizador altera um input
- Seria interessante controlar esta **atualização do processo**
- Por exemplo, com um botão de verificação (**actionButton**) para iniciar a computação dos outputs
- Um input pode ser isolado com `isolate(input$Id)`
- Para uma expressão usamos `isolate({expr})` (não esquecer {})

8.2 Exemplo | ui.R

Três inputs: **color** e **bins** para o histograma, e um **actionButton**:

```
shinyUI(fluidPage(  
  titlePanel("Isolation"),  
  sidebarLayout(  
    sidebarPanel(  
      radioButtons(inputId = "col", label = "Choose a color",  
        inline = TRUE,  
        choices = c("red", "blue", "darkgrey")),
```

```

        sliderInput("bins", "Number of bins:", min = 1, max = 50,
                    value = 30),
        actionButton("go_graph", "Update!")
    ),
    mainPanel(plotOutput("distPlot"))
)
))

```

8.3 Exemplo | server.R

Isolamento de tudo exceto do `actionButton`:

```

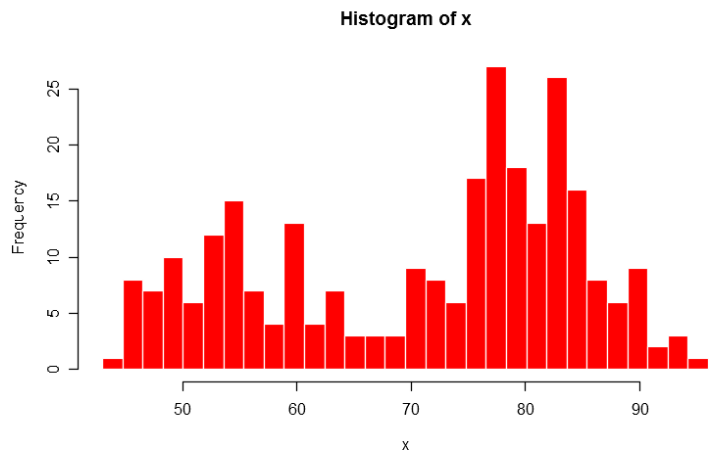
shinyServer(function(input, output) {
  output$distPlot <- renderPlot({
    input$go_graph #action to start the update process
    isolate({
      inputColor <- input$color
      x <- faithful[, 2]
      bins <- seq(min(x), max(x), length.out = input$bins + 1)
      hist(x, breaks = bins, col = inputColor, border = 'white')
    })
  })
})

```

O histograma vai ser atualizado quando o utilizador clicar no botão.

8.4 Exemplo | Aplicação

Isolation



9 Expressões reativas

9.1 Definição

- Muito úteis quando queremos usar o mesmo resultado/objetos em muitos outputs, através da execução do cálculo apenas uma vez.
- Só é necessário usar a função reativa no **server.R**
- Por exemplo, queremos visualizar dos gráficos de um PCA:
 - projeção de indivíduos
 - projeção de variáveis.

9.2 Sem expressões reativas

- **server.R**: o cálculo é realizado duas vezes. . .
- Por defeito, **apenas** expressões R em funções **renderXX** são atualizadas.

```
require(FactoMineR) ; data("decathlon")

output$graph_pca_ind <- renderPlot({
  res_pca <- PCA(decathlon[,input$variables], graph = FALSE)
  plot.PCA(res_pca, choix = "ind", axes = c(1,2))
})
output$graph_pca_var <- renderPlot({
  res_pca <- PCA(decathlon[,input$variables], graph = FALSE)
  plot.PCA(res_pca, choix = "var", axes = c(1,2))
})
```

9.3 Com uma expressão reativa

- **server.R** : O cálculo é feito apenas uma vez!

```
require(FactoMineR) ; data("decathlon")

res_pca <- reactive({
  PCA(decathlon[,input$variables], graph = FALSE)
})

output$graph_pca_ind <- renderPlot({
  plot.PCA(res_pca(), choix = "ind", axes = c(1,2))
})

output$graph_pca_var <- renderPlot({
  plot.PCA(res_pca(), choix = "var", axes = c(1,2))
})
```

9.4 Notas

- Uma expressão reativa irá poupar tempo e memória.

Use expressões reativas apenas quando elas dependem de inputs

- Expressões reativas atualizam mal o utilizador altera um input
- Obtemos o seu valor com “()”

10 Observe & funções para atualizar

10.1 Observe & funções para atualizar

- Existem muitas funções para atualizar inputs e algumas estruturas
- Elas começam com atualização...
- São geralmente usadas em `observe({expr})`
- **Tenha cuidado:** temos que adicionar “*session*” na definição do **server**

```
shinyServer(function(input, output, session) {...})
```

10.2 Exemplo para um input | Aplicação

Observer

Choose a dataset
 cars iris quakes

Choose a column
speed
speed
dist

Show 10 entries Search:

speed	dist
4	2
4	10
7	4
7	22
8	16

speed dist

Showing 1 to 5 of 50 entries

Previous 1 2 3 4 5 ...

10 Next

Observer

Choose a dataset
 cars iris quakes

Choose a column
Sepal.Length
Sepal.Length
Sepal.Width
Petal.Length
Petal.Width
Species

Show 10 entries Search:

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa

Sepal.Length Sepal.Width Petal.Length Petal.Width Species

Showing 1 to 5 of 150 entries

Previous 1 2 3 4 5 ...

30 Next

10.3 Exemplo para tabs | Aplicação

A NavBar Summary Plot

Go to plot !

A NavBar Summary Plot

Go to Summary !

For inputs:

- `updateCheckboxGroupInput`

- `updateCheckboxInput`
- `updateDateInput` Change
- `updateDateRangeInput`
- `updateNumericInput`
- `updateRadioButtons`
- `updateSelectInput`
- `updateSelectizeInput`
- `updateSliderInput`
- `updateTextInput`

Para alterar a tab selecionada:

- `updateNavbarPage`, `updateNavlistPanel`, `updateTabsetPanel`

10.4 Exemplo para um input | ui.R

```
shinyUI(fluidPage(
  titlePanel("Observe"),
  sidebarLayout(
    sidebarPanel(
      radioButtons(inputId = "id_dataset", label = "Choose a
dataset", inline = TRUE,
        choices = c("cars", "iris", "quakes"), selected =
"cars"),
      selectInput("id_col", "Choose a column", choices =
colnames(cars)),
      textOutput(outputId = "txt_obs")
    ),
    mainPanel(fluidRow(
      dataTableOutput(outputId = "dataset_obs")
    ))
  )
))
```

10.5 Exemplo para um input | server.R

```
shinyServer(function(input, output, session) {
  dataset <- reactive(get(input$id_dataset, "package:datasets"))

  observe({
    updateSelectInput(session, inputId = "id_col", label = "Choose a
      column",
      choices = colnames(dataset()))
  })

  output$txt_obs <- renderText(paste0("Selected column : ",
    input$id_col))

  output$dataset_obs <- renderDataTable(
    dataset(),
    options = list(pageLength = 5)
  )
})
```

10.6 Exemplo para um input | Aplicação

Observer

Choose a dataset

cars iris quakes

Choose a column

speed

speed

dist

Show 10 entries Search:

speed	dist
4	2
4	10
7	4
7	22
8	16

speed dist

Showing 1 to 5 of 50 entries

Previous 1 2 3 4 5 ...

10 Next

Observer

Choose a dataset

cars iris quakes

Choose a column

Sepal.Length

Sepal.Length

Sepal.Width

Petal.Length

Petal.Width

Species

Show 10 entries Search:

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa

Sepal.Length Sepal.Width Petal.Length Petal.Width Species

Showing 1 to 5 of 150 entries

Previous 1 2 3 4 5 ...

30 Next

10.7 Exemplo para tabs | ui.R

Temos que adicionar um ID na estrutura

```
shinyUI(  
  navbarPage(  
    id = "idnavbar", # need an id for observe & update  
    title = "A NavBar",  
    tabPanel(title = "Summary",  
      actionButton("goPlot", "Go to plot !")),  
    tabPanel(title = "Plot",  
      actionButton("goSummary", "Go to Summary !"))  
  )  
)
```

)

10.8 Exemplo para tabs | server.R

```
shinyServer(function(input, output, session) {  
  observe({  
    input$goPlot #action to start the update process  
    updateTabsetPanel(session, "idnavbar", selected = "Plot")  
  })  
  observe({  
    input$goSummary #action to start the update process  
    updateTabsetPanel(session, "idnavbar", selected = "Summary")  
  })  
})
```

10.9 Exemplo para tabs | Aplicação



10.10 ObserveEvent

- Uma alternativa a observe: observeEvent
- Temos que definir a expressão do evento e a expressão que executa quando o evento ocorre

```
# with observe  
observe({  
  input$goPlot  
  updateTabsetPanel(session, "idnavbar", selected = "Plot")  
})  
# same with observeEvent  
observeEvent(input$goSummary, {  
  updateTabsetPanel(session, "idnavbar", selected = "Summary")  
})
```

11 Painéis condicionais

11.1 Definição

- Podemos usar condições para imprimir alguns inputs/outputs

```
conditionalPanel(condition = [...], )
```

- A condição pode depender dos inputs ou outputs
- Tenha cuidado: deve ser escrito em **javascript**. . .

```
conditionalPanel(condition = "input.checkbox == true", [...])
```

11.2 Exemplo para um input

```
shinyApp(  
  ui = fluidPage(  
    fluidRow(  
      column(width = 4, align = "center",  
        checkboxInput("checkbox", "View other inputs", value =  
          FALSE)  
      ),  
      column(width = 8, align = "center",  
        conditionalPanel(  
          condition = "input.checkbox == true",  
          sliderInput("slider", "Select value", min = 1, max =  
            10, value = 5),  
          textInput("txt", "Enter text", value = "")  
        )  
      )  
    )  
  ),  
  server = function(input, output) {}  
)
```

11.3 Exemplo para um input

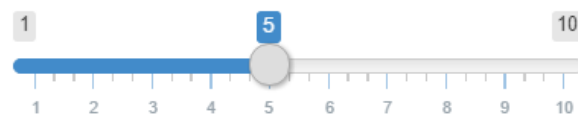
Condition FALSE

View other inputs

Condition TRUE

View other inputs

Select value



Enter text

12 Levando as coisas mais longe: HTML / CSS

12.1 Incluir HTML

Muitas **tags html** estão disponíveis com funções de tags:

##	[1]	"a"	"abbr"	"address"	
##	[4]	"animate"	"animateMotion"	"animateTransform"	
##	[7]	"area"	"article"	"aside"	
##	[10]	"audio"	"b"	"base"	
##	[13]	"bdi"	"bdo"	"blockquote"	
##	[16]	"body"	"br"	"button"	
##	[19]	"canvas"	"caption"	"circle"	
##	[22]	"cite"	"clipPath"	"code"	
##	[25]	"col"	"colgroup"	"color-profile"	
##	[28]	"command"	"data"	"datalist"	
##	[31]	"dd"	"defs"	"del"	
##	[34]	"desc"	"details"	"dfn"	
##	[37]	"dialog"	"discard"	"div"	
##	[40]	"dl"	"dt"	"ellipse"	
##	[43]	"em"	"embed"	"eventsource"	
##	[46]	"feBlend"	"feColorMatrix"	"feComponentTransfer"	
##	[49]	"feComposite"	"feConvolveMatrix"	"feDiffuseLighting"	
##	[52]	"feDisplacementMap"	"feDistantLight"	"feDropShadow"	
##	[55]	"feFlood"	"feFuncA"	"feFuncB"	
##	[58]	"feFuncG"	"feFuncR"	"feGaussianBlur"	
##	[61]	"feImage"	"feMerge"	"feMergeNode"	
##	[64]	"feMorphology"	"feOffset"	"fePointLight"	
##	[67]	"feSpecularLighting"	"feSpotLight"	"feTile"	
##	[70]	"feTurbulence"	"fieldset"	"figcaption"	
##	[73]	"figure"	"filter"	"footer"	
##	[76]	"foreignObject"	"form"	"g"	
##	[79]	"h1"	"h2"	"h3"	
##	[82]	"h4"	"h5"	"h6"	
##	[85]	"hatch"	"hatchpath"	"head"	
##	[88]	"header"	"hgroup"	"hr"	
##	[91]	"html"	"i"	"iframe"	
##	[94]	"image"	"img"	"input"	
##	[97]	"ins"	"kbd"	"keygen"	
##	[100]	"label"	"legend"	"li"	
##	[103]	"line"	"linearGradient"	"link"	
##	[106]	"main"	"map"	"mark"	
##	[109]	"marker"	"mask"	"menu"	
##	[112]	"meta"	"metadata"	"meter"	
##	[115]	"mpath"	"nav"	"noscript"	
##	[118]	"object"	"ol"	"optgroup"	
##	[121]	"option"	"output"	"p"	
##	[124]	"param"	"path"	"pattern"	
##	[127]	"picture"	"polygon"	"polyline"	
##	[130]	"pre"	"progress"	"q"	
##	[133]	"radialGradient"	"rb"	"rect"	
##	[136]	"rp"	"rt"	"rtc"	
##	[139]	"ruby"	"s6"	"samp"	
##	[142]	"script"	"section"	"select"	
##	[145]	"set"	"slot"	"small"	
##	[148]	"solidcolor"	"source"	"span"	
##	[151]	"stop"	"strong"	"style"	
##	[154]	"sub"	"summary"	"sup"	
##	[157]	"svg"	"switch"	"symbol"	
##	[160]	"table"	"tbody"	"td"	

```
tags$a(href = "www.rstudio.com", "RStudio")
```

the list
named tags

the function/tag name
(followed by parentheses)

named arguments
appear as tag attributes
(set boolean attributes to NA)

unnamed arguments
appear inside the tags
(call tags\$...() to create nested tags)

```
<a href="www.rstudio.com">RStudio</a>
```

Podemos também usar código **html** com funções **HTML** :

```
fluidPage(  
  HTML("<h1>My Shiny App</h1>")  
)
```

12.2 Algumas tags interessantes

- `div(..., align = "center")`: elementos centrais
- `br()`: line break
- `hr()`: linha horizontal
- `img(src="img/logo.jpg", title="Popup", width = "80%")`: insira imagem em **www/img**
- `a(href="https://r2018-rennes.sciencesconf.org/", target="_blank", "Rencontres R")`: link para um website
- `a(href = './doc/guide.pdf', target="_blank", class = "btn", icon("download"), "Télécharger le guide utilisateur")`: link para transferir um documento em **www/doc**

12.3 CSS: introdução

Shiny usa **Bootstrap** para a parte de **CSS** .

Quanto a desenvolvimento web clássico, podemos alterar o **CSS** em três maneiras:

- link para um **.css file** no diretório **www**
- adicionar **CSS** no header **HTML**
- usar códigos **CSS** num elemento.

Ordem de prioridade : 1. códigos CSS num elemento 2. CSS no header HTML

3. ficheiro .css

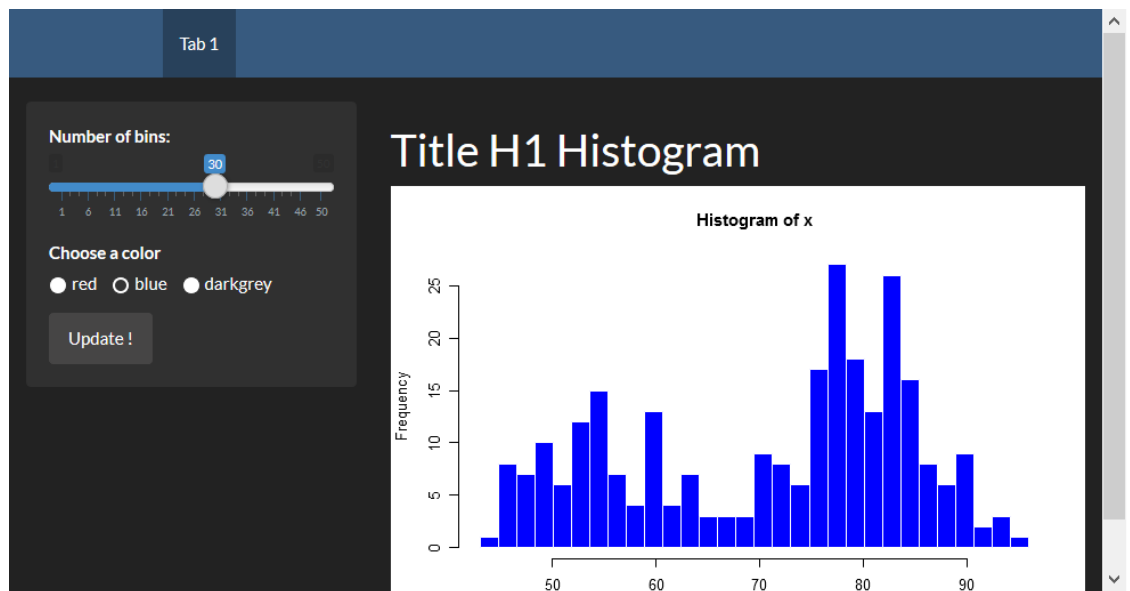
Também podemos usar a package [shinythemes](#).

12.4 HTML / CSS | ficheiro css externo

Pode encontrar alguns temas em bootswatch.

Duas maneiras para especificar o tema: + tema de opção em algumas funções (fluidPage, navbarPage, . . .) + com **tags html** : tags\$head et tags\$link

```
library(shiny)
ui <- fluidPage(theme = "mytheme.css",
  # or with a tags
  tags$head(
tags$link(rel = "stylesheet", type = "text/css", href =
  "mytheme.css")
  ),
  # ...
)
```



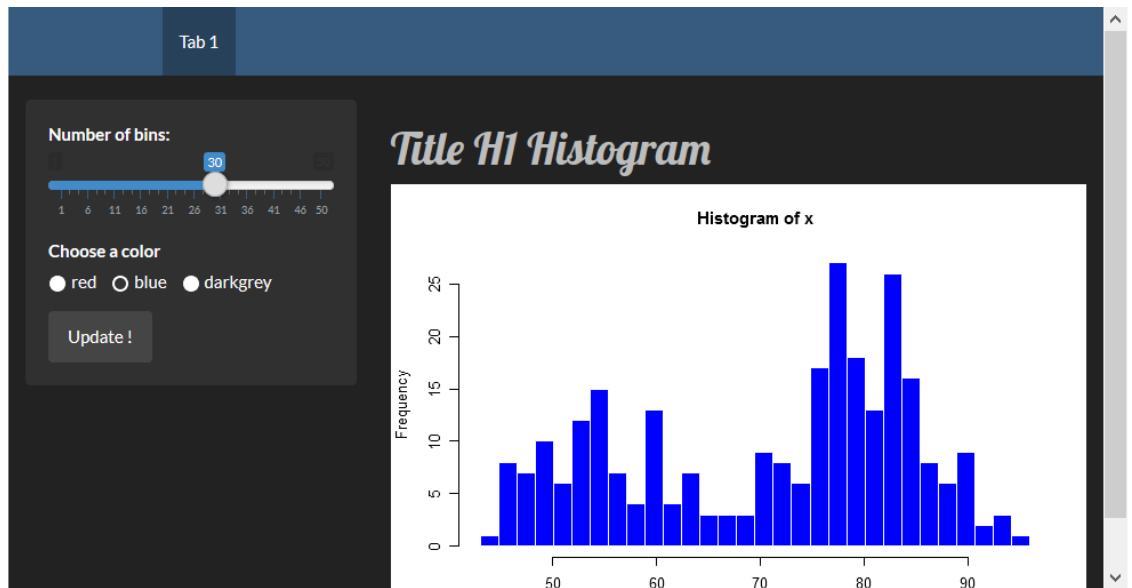
12.5 HTML / CSS | css no header

- With **tags html**: tags\$head e tags\$style

```

library(shiny)
  tags$head(
tags$style(HTML("h1 { color: #48ca3b;}")
)
  ),
  # ...
)

```

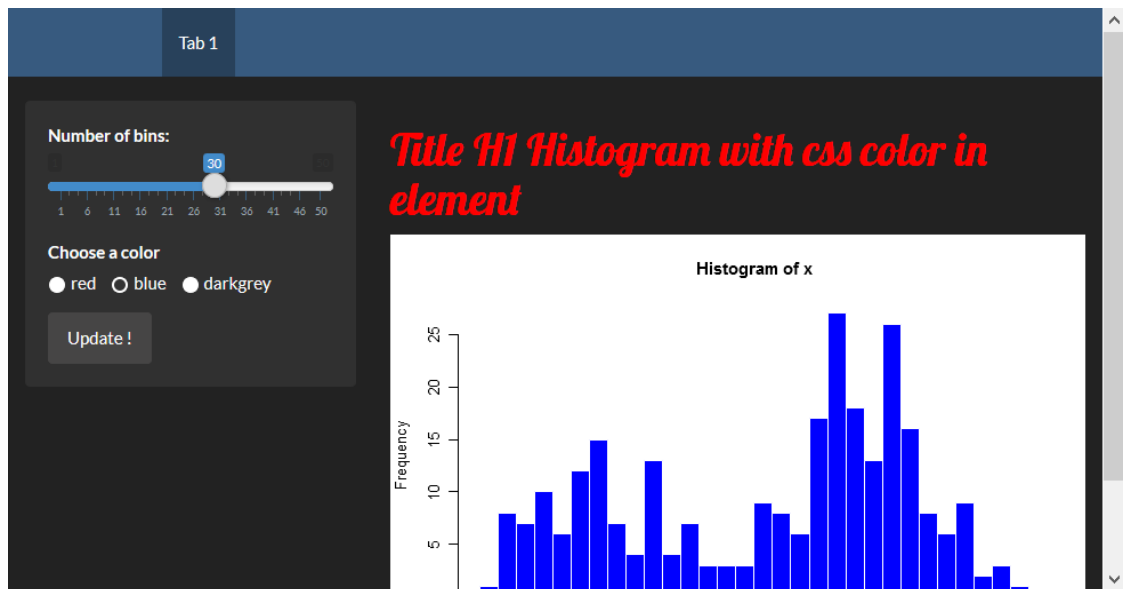


12.6 HTML / CSS | CSS num elemento

```

library(shiny)
  h1("Mon titre", style = "color: #48ca3b;")
  # reste de l'application
)

```



13 Levando as coisas mais longe: algumas “regras” importantes

13.1 Boa abordagem

- Escolha underscore (_) em vez de ponto (.) nos nomes de objetos ou variáveis. De facto, o ponto . pode levar a algumas confusões com outras linguagens, como JavaScript
- Use a package `packrat` para evitar problemas com `version packages`
- Use `R script` para a parte do cálculo e fazer testes (`convém testar`).

13.2 Boa abordagem

- Divida as partes do `ui.R` e do `server.R` em vários scripts, um para cada tab, por exemplo:

```
# ui.R
shinyUI(
  navbarPage("Divide UI & SERVER",
    source("src/ui/01_ui_plot.R", local = TRUE)$value,
    source("src/ui/02_ui_data.R", local = TRUE)$value
  )
)
# server.R
```

```
shinyServer(function(input, output, session) {  
  source("src/server/01_server_plot.R", local = TRUE)  
  source("src/server/02_server_data.R", local = TRUE)  
})
```

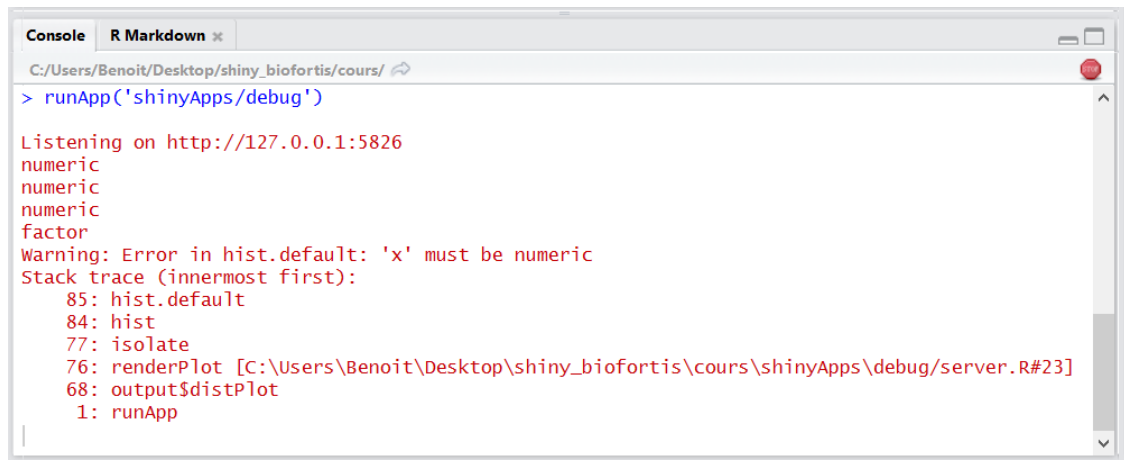
14 Levando as coisas mais longe: debugging

14.1 Imprimir na consola

- Pode usar alguns “prints” na aplicação
- Permite visualizar informações durante o processo
- No **shiny**, use `cat(file=stderr(), ...)` para ter a certeza que o ecrã opera para todos os tipos de outputs

```
output$distPlot <- renderPlot({  
  x <- iris[, input$variable]  
  cat(file=stderr(), class(x)) # affichage de la classe de x  
  hist(x)  
})
```

14.2 Imprimir na consola



```
Console R Markdown x  
C:/Users/Benoit/Desktop/shiny_biofortis/cours/  
> runApp('shinyApps/debug')  
  
Listening on http://127.0.0.1:5826  
numeric  
numeric  
numeric  
factor  
Warning: Error in hist.default: 'x' must be numeric  
Stack trace (innermost first):  
 85: hist.default  
 84: hist  
 77: isolate  
 76: renderPlot [C:\Users\Benoit\Desktop\shiny_biofortis\cours\shinyApps\debug\server.R#23]  
 68: output$distPlot  
  1: runApp
```

14.3 Início manual de um browser

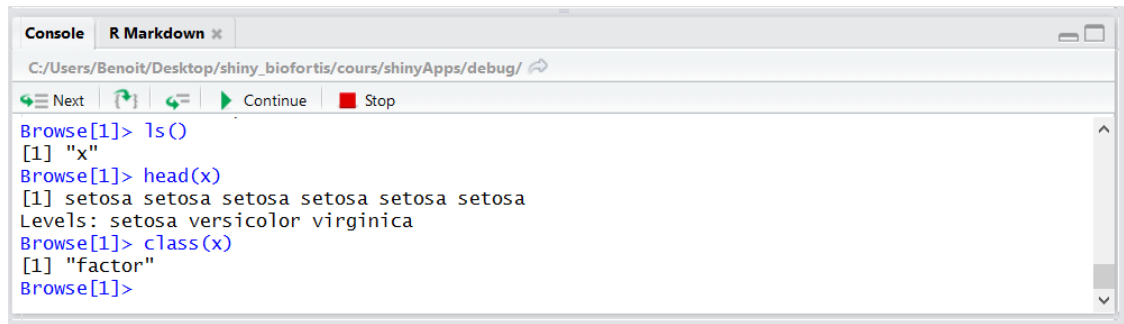
- Podemos iniciar um browser com `browser()` em qualquer lugar

- Permite observar os diferentes objetos

```
output$distPlot <- renderPlot({  
  x <- iris[, input$variable]  
  browser() # inicializao do browser  
  hist(x)  
})
```

- Não esquecer removê-lo!

14.4 Início manual de um browser



14.5 Início automático de um browser

- A opção `options(shiny.error = browser)` permite correr `browser()` mal um erro aparece

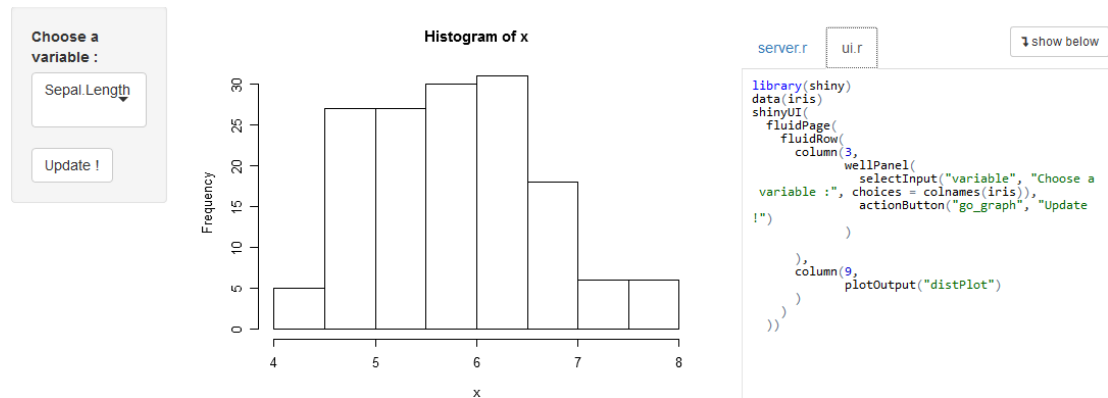
```
options(shiny.error = browser)
```

14.6 Modo "showcase"

- Com o `display.mode="showcase"` em `runApp()`, podemos observar diretamente o código executado:

```
runApp("path/to/myapp", display.mode="showcase")
```

14.7 Mode “showcase”



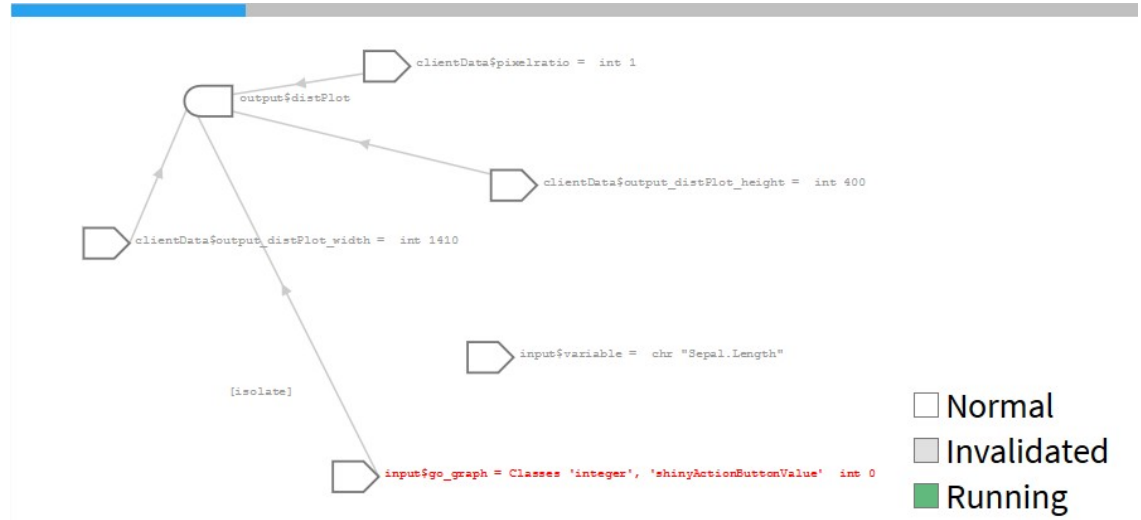
14.8 Log reativo

- Com `shiny.reactlog`, podemos visualizar dependências entre **objetos reativos** e `shiny`
 - use `ctrl+F3` no browser
 - com `showReactLog()` no código shiny

```
options(shiny.reactlog=TRUE)
```

```
output$distPlot <- renderPlot({
  x <- iris[, input$variable]
  showReactLog() # launch shiny.reactlog
  hist(x)
})
```

14.9 Log relativo



14.10 Comunicação no servidor

- Podemos visualizar estas comunicações com a opção `shiny.trace`

```
options(shiny.trace = TRUE)
```

14.11 Comunicação no servidor

```
Console R Markdown x
C:/Users/Benoit/Desktop/shiny_biofortis/cours/
> runApp('shinyApps/debug')

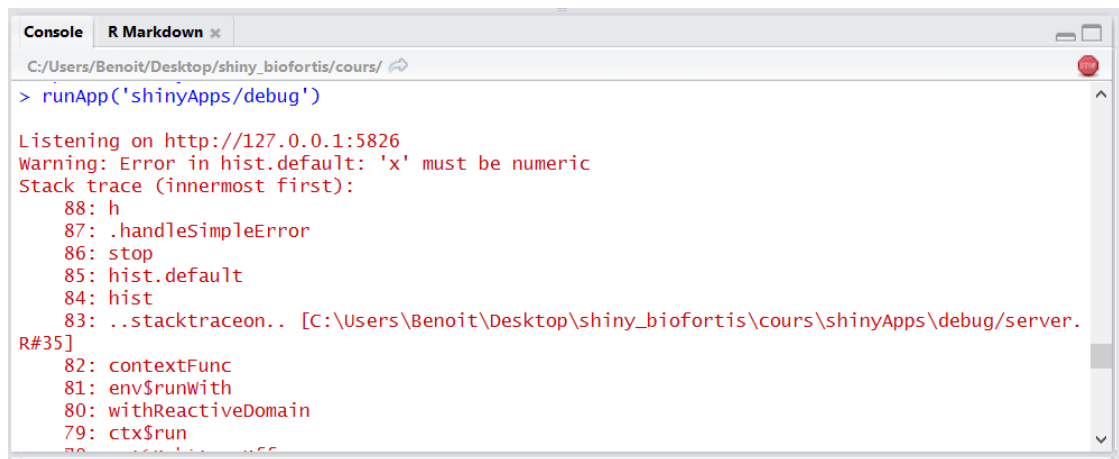
Listening on http://127.0.0.1:5826
SEND {"config":{"workerId":"","sessionId":"d881eec9a56887dd66d5d6bf2f8776ed"}}
RECV {"method":"init","data":{"go_graph:shiny.action":0,"variable":"Sepal.Length",".clientdata_output_distPlot_width":816,".clientdata_output_distPlot_height":400,".clientdata_output_distPlot_hidden":false,".clientdata_pixelratio":1,".clientdata_url_protocol":"http:",".clientdata_url_hostname":"127.0.0.1",".clientdata_url_port":"5826",".clientdata_url_pathname":"/",".clientdata_url_search":"",".clientdata_url_hash_initial":"",".clientdata_singletons":"",".clientdata_allowDataUriScheme":true}}
SEND {"custom":{"busy":"busy"}}
SEND {"custom":{"recalculating":{"name":"distPlot","status":"recalculating"}}}
SEND {"custom":{"recalculating":{"name":"distPlot","status":"recalculated"}}}
SEND {"custom":{"busy":"idle"}}
SEND {"errors":[],"values":{"distPlot":{"src":"data:image/png;base64 data","width":816,"height":400,"coordmap":{"domain":{"left":3.84,"right":8.16,"bottom":-1.24,"top":32.24},"range":{"left":59.04,"right":785.76,"bottom":325.56,"top":58.04},"log":{"x":null,"y":null},"mapping":{}}}}, "inputMessages":[]}}
RECV {"method":"update","data":{"variable":"Petal.Length"}}
```

14.12 Tracking the erros

- Desde o shiny_0.13.1, podemos obter um stack trace quando ocorre um erro
- Podemos obter mais informações com `options(shiny.fullstacktrace = TRUE)`

```
options(shiny.fullstacktrace = TRUE)
```

14.13 Tracking de erros



The screenshot shows an R console window with the following output:

```
> runApp('shinyApps/debug')

Listening on http://127.0.0.1:5826
Warning: Error in hist.default: 'x' must be numeric
Stack trace (innermost first):
 88: h
 87: .handleSimpleError
 86: stop
 85: hist.default
 84: hist
 83: ..stacktraceon.. [C:\Users\Benoit\Desktop\shiny_biofortis\cours\shinyApps\debug\server.R#35]
R#35]
 82: contextFunc
 81: env$runWith
 80: withReactiveDomain
 79: ctx$run
 78: ...
```

15 Referências

15.1 Tutoriais / Exemplos

- <http://shiny.rstudio.com/>
- <http://shiny.rstudio.com/articles/>
- <http://shiny.rstudio.com/tutorial/>
- <http://shiny.rstudio.com/gallery/>
- <https://www.rstudio.com/products/shiny/shiny-user-showcase/>
- <http://www.showmeshiny.com/>

15.2 Código da nossa aplicação

- https://github.com/JDS26/ProjFinal_LCC/