Creating Shiny Apps for biostatisticians and bioinformaticians

ISGlobal

21-22 May 2019

Isaac Subirana

Part III: How Shiny works

Outline

Part III: Logic of Shiny

- How Shiny works
- Isolate
- Reactive objects
- Upload data
- Download files
- Validate inputs
- Rendering elements
- Exercise

How Shiny works

UI

- Where the elements and form structure is specified.
- The commands to create elements must be separated by commas.
- Do not write `input$elem`. You can only refer to input elements values thru `conditionalPanel`.

```
fluidPage(
  textInput(...),
  conditionalPanel("input.help",
        ...
  ),
  tabsetPanel(
    tabPanel(...),
    tabPanel(...)
  ),
  plotOutput(...)
)
```

Server

Where the computations are performed (tables, plots, etc). It can contain:

- UI code using **`renderUI`**:

```
function(input, output){
  output$elementoUI <- renderUI(
    textInput("alias","label")
  )
}
```

- objects that are re-computed when necessary and used by other functions inside server (**`reactive`**):

```
function(input, output){
  dat <- reactive(...)
  output$plot <- renderPlot(hist(dat())
}
```

- nothing.

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

Files structure

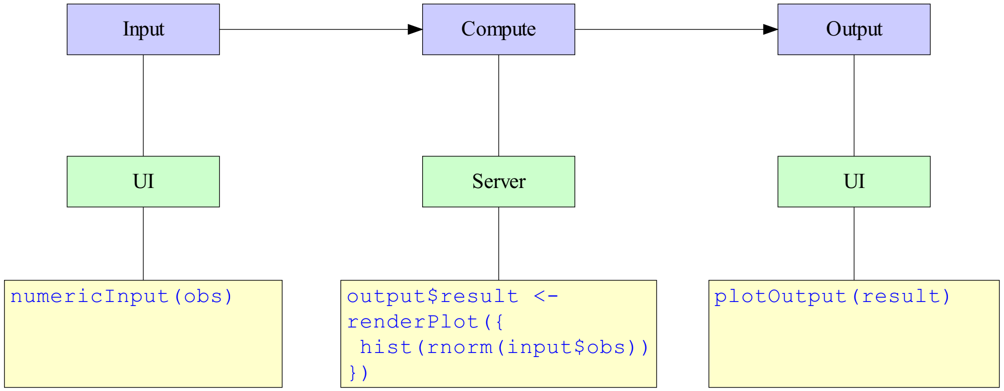
Separated files

- **Mandatory files**
  - `ui.R`: file with UI section code.
  - `server.R`: file with Server section code.
- **Optional files**
  - `global.R`: file where required packages are loaded, data read, objects and functions created, etc. Its code is executed once the app is iniciated.
  - `www`: folder with images, CSS code files,...

Single file

- **Mandatory files**
  - `app.R`: with UI and Server code, and other code that would be placed in "global.R" file.
- **Optional files**
  - `www`: folder with images, CSS code files,...

Pipeline



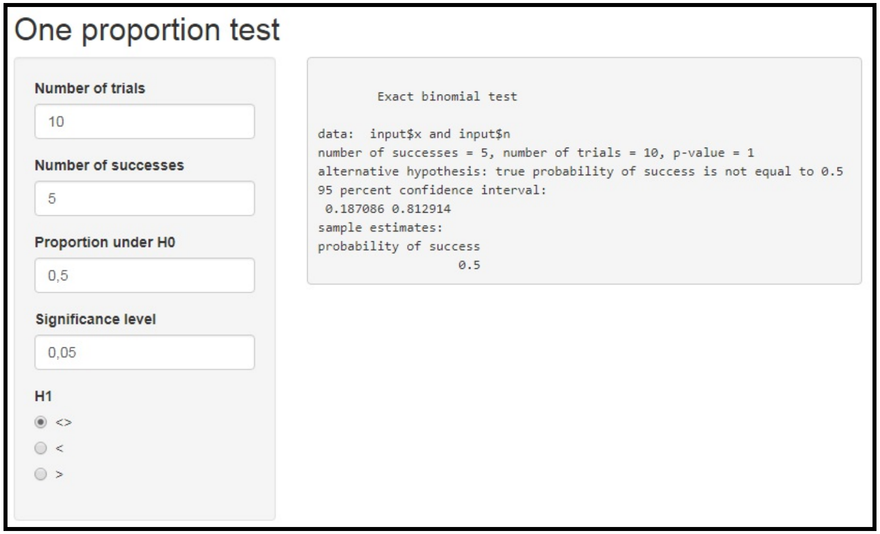
```
numericInput(obs)
```

```
output$result <-
renderPlot({
 hist(rnorm(input$obs))
})
```

```
plotOutput(result)
```

- **Reactivity**: creating and upating elements in **input** and **output** lists.
- **input** and **output** lists are the arguments in the function defined in **Server** section.

Translation of an R function to a Shiny app



Number of trials: 10

Number of successes: 5

Proportion under H0: 0,5

Significance level: 0,05

H1: [✓ <>] < >

```
        Exact binomial test

data:  input$x and input$n
number of successes = 5, number of trials = 10, p-value = 1
alternative hypothesis: true probability of success is not equal to 0.5
95 percent confidence interval:
 0.187086 0.812914
sample estimates:
probability of success
                   0.5
```

Inputs:

- number of trials,
- number of successes,
- proportion under H0,
- significance level,
- H1 direction (one-sided, two-sided).

Outputs:

- binomial test result.

```r
# Number of trials
x <- 3

# Number of successes
n <- 10

# Proportion under H0
p0 <- 0.5

# Significance level
alpha <- 0.05

# Alternative H1, 1: <>, 2: <, 3: >
side <- 1

## results
binom.test(x = x,
           n = n,
           p = p0,
           alt = switch(side, '1'="two.sided", '2'="less",'3'="greater"),
           conf.level = 1-alpha)
```

```
##
##      Exact binomial test
##
## data:  x and n
## number of successes = 3, number of trials = 10, p-value = 0.3438
## alternative hypothesis: true probability of success is not equal to 0.5
## 95 percent confidence interval:
##  0.06673951 0.65245285
## sample estimates:
## probability of success
##                    0.3
```

```r
ui <- fluidPage(

  titlePanel("One proportion test"),

  sidebarLayout(
    sidebarPanel(
      numericInput("n", "Number of trials", 10),
      numericInput("x", "Number of successes", 5),
      numericInput("p0", "Proportion under H0", 0.5),
      numericInput("alpha", "Significance level", value = 0.05),
      radioButtons("side", "H1", c("<>"=1, "<"=2, ">"=3), 1)
    ),
    mainPanel(
      verbatimTextOutput("result")
    )
  )
)

server <- function(input, output) {

  output$result <- renderPrint({
    binom.test(input$x,
               input$n,
               p = input$p0,
               alt = switch(input$side, '1'="two.sided", '2'="less",'3'="greater"),
               conf.level = 1-input$alpha)
 })

}

shinyApp(ui = ui, server = server)
```

Isolate

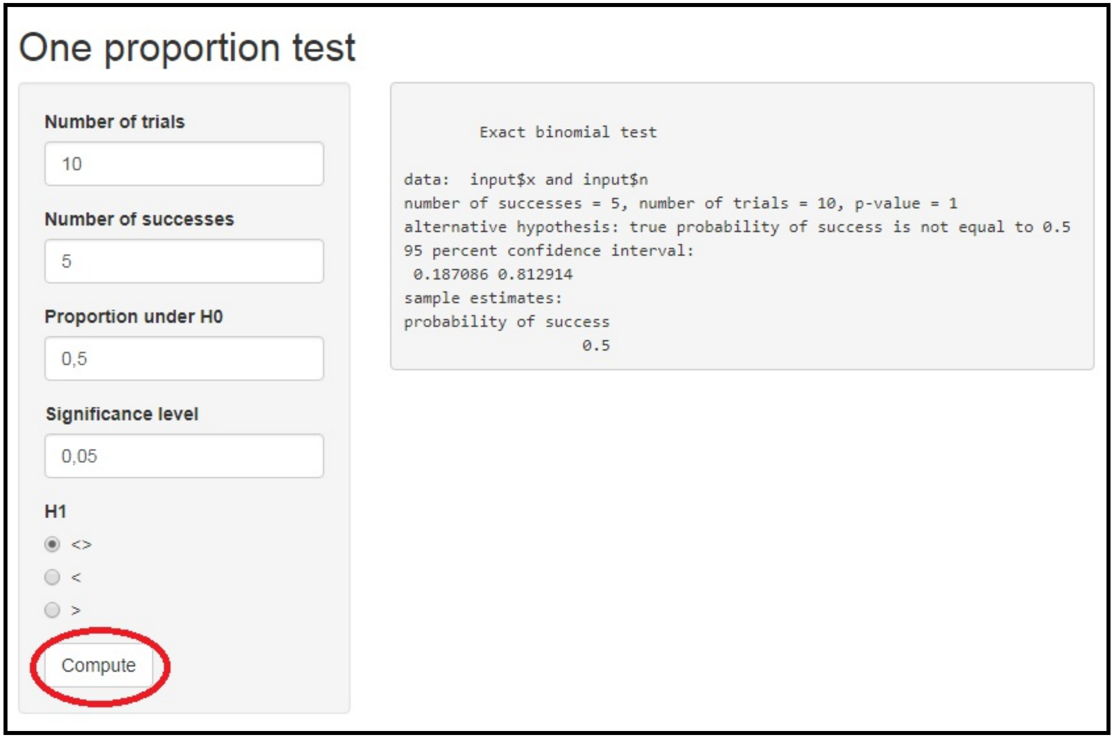
Reactivity control: Isolate

- By default, every time an input element is changed or updated, all instruction inside a "block" or "blocks" in Server section where this or these elements are called (`renderPrint`, `renderTable`, `reactive`, ...) are executed.
- If you don't want all these instructions be executed, or results be updated, untill modify all desired inputs, you must specify it. -> **`isolate`** function.

```
server <- funcion(input, ouput){

  output$foo <- renderXXX({

    input$elem1

     isolate({
       ....
     })

  })
}
```

Example: One proportion test

Let's recover the previous example. Now it is desired that results don't be updated until the **compute** button is pressed.



Number of trials: 10

Number of successes: 5

Proportion under H0: 0,5

Significance level: 0,05

H1: [✓ <>] < >

```
	Exact binomial test

data:  input$x and input$n
number of successes = 5, number of trials = 10, p-value = 1
alternative hypothesis: true probability of success is not equal to 0.5
95 percent confidence interval:
 0.187086 0.812914
sample estimates:
probability of success 
                   0.5 
```

```r
ui <- fluidPage(
  titlePanel("One proportion test"),
  sidebarLayout(
    sidebarPanel(
      numericInput("n", "Number of trials", 10),
      numericInput("x", "Number of successes", 5),
      numericInput("p0", "Proportion under H0", 0.5),
      numericInput("alpha", "Significance level", value = 0.05),
      radioButtons("side", "H1", c("<>"=1, "<"=2, ">"=3), 1),
      actionButton("compute", "Compute")
    ),
    mainPanel(
      verbatimTextOutput("result")
    )
  )
)

server <- function(input, output) {
  output$result <- renderPrint({
    if (input$compute==0) return(invisible(NULL))
     isolate({
       binom.test(x = input$x, n = input$n, p = input$p0,
                  alt = switch(input$side, '1'="two.sided",
                               '2'="less",'3'="greater"),
                  conf.level = 1-input$alpha)
     })
 })
}

shinyApp(ui = ui, server = server)
```

Reactive objects

Reactive objects

- In Shiny, objects are recomputed when some input values are changed by the user.
- At the same time, these input elements may be involved in the computation of several elements of **output** list.
- Recomputing them every time may be inefficient. For example, when reading a huge data base or solving complex algorithms.
- The reactive object is created by the `reactive` function.

Example: iris data to perform a summary and a plot.

```
function(input, output) {

   dat <- reactive({
     if (input$specie == 'All')
       return(iris)
     else
       return(subset(iris, Species == input$specie))
   })

  output$summary <- renderPrint(summary(dat()))

  output$plot <- renderPlot(pairs(dat()[,-5]))

}
```

Note: The object `dat` is **not** a data.frame but a function, `dat()`. It is only updated when needed.

Exercise: write the UI section with the minimum required input elements

Upload data

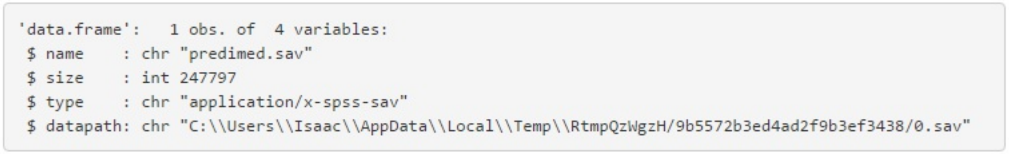
Upload data

- Use `fileInput` function in **UI** section.
- The input element defined with `fileInput` contains the file name, among other file properties.
- The proper **R** function will be used to read the file in **Server** section: `haven::read_sav`, `read.table`, `readxl::read_excel`, etc.
- It is suggested to read the data set as a reactive object with the `reactive` function.

```r
library(haven) # read SPSS
library(readxl) # read Excel

ui <- fluidPage(
  sidebarLayout(
    sidebarPanel(
      fileInput("files", ""),
      radioButtons("datatype", "Type", c("SPSS","EXCEL"))
    ),
    mainPanel(
      tabsetPanel(
        tabPanel("Summary", verbatimTextOutput("sum")),
        tabPanel("Table", tableOutput("tab"))
      )
    )
  )
)

server <- function(input, output) {
  dd<-reactive({
    inFile<-input$files
    if (is.null(inFile)) return(invisible(NULL))
    if (input$datatype=='EXCEL')
      return(read_excel(inFile$datapath,1))
    if (input$datatype=='SPSS')
      return(read_sav(inFile$datapath))
  })
  output$sum <- renderPrint(summary(dd()))
  output$tab <- renderTable(head(dd()))
}

shinyApp(ui = ui, server = server)
```

Let's take a look on `input$files` element:

```r
ui <- fluidPage(
  fileInput("files", ""),
  verbatimTextOutput("files")
)
server <- function(input, output){
  output$files <- renderPrint(str(input$files))
}
shinyApp(ui, server)
```



```
'data.frame':	1 obs. of  4 variables:
 $ name    : chr "predimed.sav"
 $ size    : int 247797
 $ type    : chr "application/x-spss-sav"
 $ datapath: chr "C:\\Users\\Isaac\\AppData\\Local\\Temp\\RtmpQzWgzH/9b5572b3ed4ad2f9b3ef3438/0.sav"
```

Note that `input$files` is itself a list with different components. The one used to read the file is `datapath`.

Exercise: Take the file extension (`fileext`) to choose automatically between SPSS and Excel, without asking the user.

Download files

Download files

From the Server section

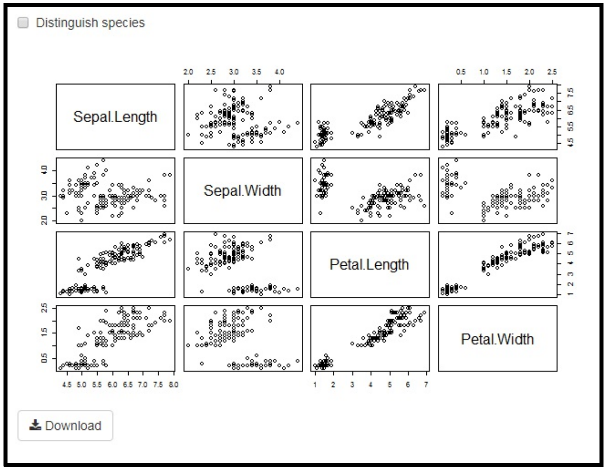
- Use the `downloadHandler` function to create the element of the **output** list.
- `downloadHandler` has two arguments:
  - `filename`: function without any argument that returns the filename.
  - `content`: function that creats and save the plot (or the file in general) as the file name defined in its input.

From the UI section

- In the **UI** section use the `downloadButton` widget.

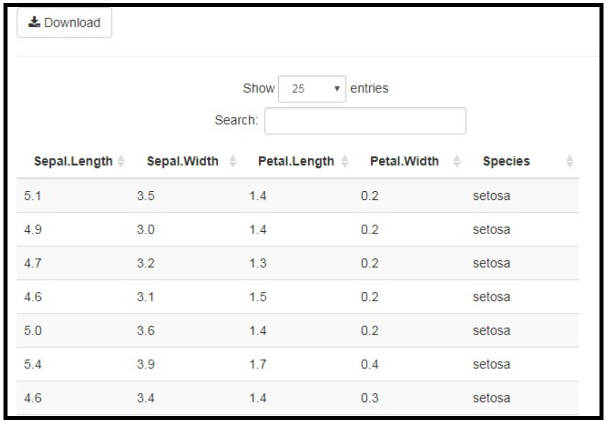
Example 1: download a plot

```r
ui <- fluidPage(
  checkboxInput("group", "Distinguish species"),
  plotOutput("result"),
  downloadButton('down', 'Download')
)

server <- function(input, output) {
  output$result <- renderPlot(
    if (input$group)
      pairs(iris[,-5],col=iris[,5])
    else
      pairs(iris[,-5])
  )
   output$down <- downloadHandler(
     filename = function() "figure.pdf",
     content = function(ff) {
       pdf(ff)
       if (input$group)
         pairs(iris[,-5],col=iris[,5])
       else
         pairs(iris[,-5])
       dev.off()
     }
   )
}

shinyApp(ui = ui, server = server)
```



Example 2: download a data table

```r
ui <- fluidPage(
  downloadButton('down', 'Download'),
  hr(),
  dataTableOutput("result")
)

server <- function(input, output) {
  output$result <- renderDataTable(iris)
   output$down <- downloadHandler(
     filename = function(){
       "iris.csv"
     },
     content = function(ff){
       write.table(iris, file=ff,
                   sep=";", row.names=FALSE)
     }
   )
}

shinyApp(ui = ui, server = server)
```



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
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa

Note: The app must be launched from a navigator (Chome, Firefox, ...). From RStudio viewer only pdf file can be downloaded.

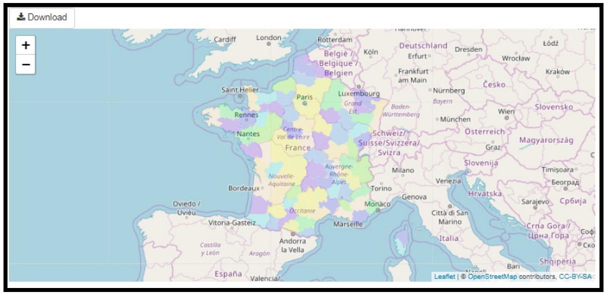
Example 3: download a file

- To download an existing file, use **`file.copy`**.
- The file must be inside the server (or the PC) where the app is stored.
- The file may be created or updated when app is running.

```r
ui <- fluidPage(

  downloadButton('down', 'Download')

)

server <- function(input, output) {

  output$down <- downloadHandler(
    filename = function() "report.pdf",
    content = function(ff){
       file.copy(from="./folder/report.pdf",
                 to=ff)
    }
  )

}

shinyApp(ui = ui, server = server)
```

Example 4: download a dynamic map

```
library(leaflet)
library(maps)
mymap <- map("france", fill=TRUE, plot=FALSE)

ui <- fluidPage(
  downloadButton('down', 'Download'),
  leafletOutput("plot")
)

server <- function(input, output) {
   map <- reactive({
     leaflet(data=mymap)%>%addTiles()%>%
       addPolygons(fillColor=topo.colors(10),
                   stroke=FALSE)
   })
  output$plot <- renderLeaflet(map())
   output$down <- downloadHandler(
     filename = function() "plot.png",
     content = function(ff){
       export(map(), file=ff)
     }
   )
}

shinyApp(ui = ui, server = server)
```



Notice that the map has been created as a reactive object, to be used when showing the plot and when downloading it.

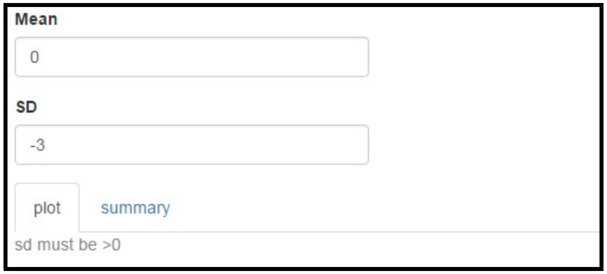
Validate inputs

Validate inputs

- Sometimes the output (plot, table or text) need of some valid input values.
- Under the user point of view it is useful to recieve a message to know what happens if he or she has introduced wrong parameters.
- Use `validate` in conjuntion to `need` function
- This text is printed even when a plot or table is expected.
- First argument of `need` is an expression the input element to be checked.
- If the expression is FALSE, NULL, empty or return an error, a message placed as the second argument is displayed in the output.

Validate inputs

```
ui <- fluidPage(
  numericInput("mu","Mean",0),
  numericInput("sd","SD",1),
  tabsetPanel(
    tabPanel("plot",
      plotOutput("plot")
    ),
    tabPanel("summary",
      verbatimTextOutput("summ")
    )
  )
)

server <- function(input, output){
  dat <- reactive({
     validate(
       need(input$sd>0,"sd must be >0")
     )
    rnorm(1000, input$mu, input$sd)
  })
  output$plot <- renderPlot(hist(dat()))
  output$summ <- renderPrint(summary(dat()))
}

shinyApp(ui, server)
```



sd must be >0

Rendering input elements

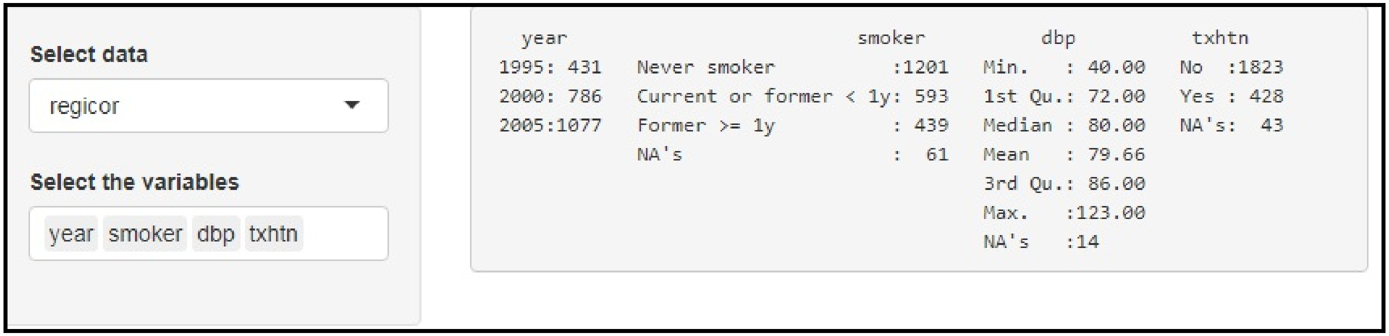
`uiOutput` / `renderUI`

Rendering elements

- Up to this point, we have seen how to hide or show input elements in the form depending on the value of other input elements by using `conditionalPanel` function.
- But often we want to modify some apects or properties of input elements according to other input element value.
- To do so, there exists **`renderUI,uiOutput`** functions to render input elements in a very powerful and flexible way.

Example. Selecting variables from a data base

- Firstly, the user must choose the data base from (regicor, predimed or SNPs) available from `compareGroups` package.
- Once the data set is chosen, a list containing the variables must appaer.
- The user, then choose which variables are summarized.
- Note that the **items** (`choices` argument) of `selectInput` varies according on the chosen data base.

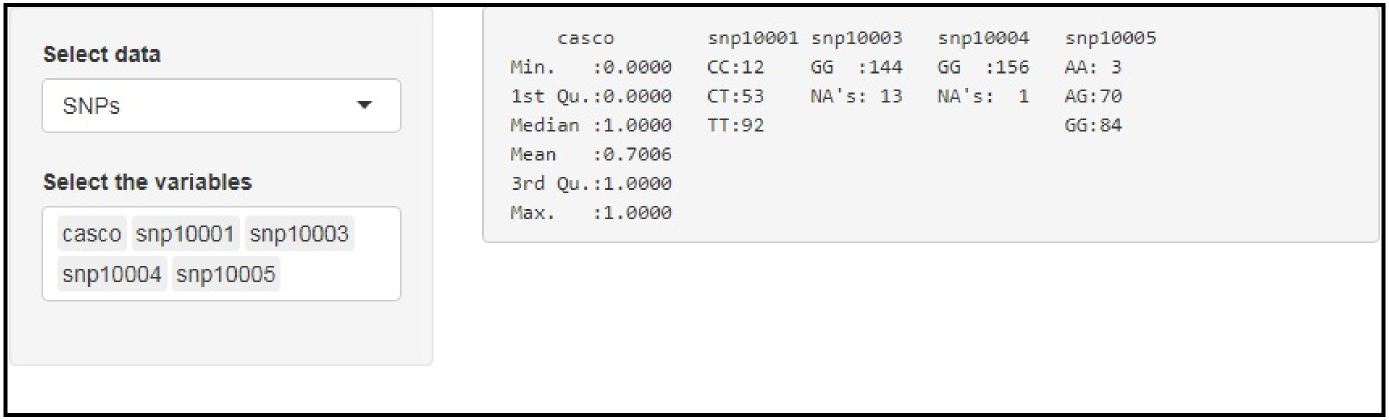

Select data

regicor

Select the variables

year smoker dbp txhtn

```
   year                          smoker          dbp            txhtn
 1995: 431   Never smoker           :1201   Min.   : 40.00   No  :1823
 2000: 786   Current or former < 1y: 593   1st Qu.: 72.00   Yes : 428
 2005:1077   Former >= 1y           : 439   Median : 80.00   NA's:  43
             NA's                   :  61   Mean   : 79.66
                                            3rd Qu.: 86.00
                                            Max.   :123.00
                                            NA's   :14
```


Select data

SNPs

Select the variables

casco snp10001 snp10003 snp10004 snp10005

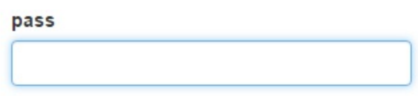
```
     casco        snp10001 snp10003   snp10004   snp10005
 Min.   :0.0000   CC:12    GG  :144   GG  :156   AA: 3
 1st Qu.:0.0000   CT:53    NA's: 13   NA's:  1   AG:70
 Median :1.0000   TT:92                          GG:84
 Mean   :0.7006
 3rd Qu.:1.0000
 Max.   :1.0000
```

```r
library(compareGroups); data(regicor);data(predimed);data(SNPs)

ui <- fluidPage(
  sidebarLayout(
    sidebarPanel(
      selectInput("data", "Select data", c("regicor", "predimed", "SNPs")),
      uiOutput("listvars")
    ),
    mainPanel(verbatimTextOutput("result"))
  )
)

server <- function(input, output) {

  dat <- reactive(get(input$data))

  output$result <- renderPrint({
    summary(dat()[input$vars])
  })

   output$listvars <- renderUI({
     vars <- names(dat())
     selectInput("vars", "Select the variables", names(dat()), multiple=TRUE)
   })

}

shinyApp(ui = ui, server = server)
```

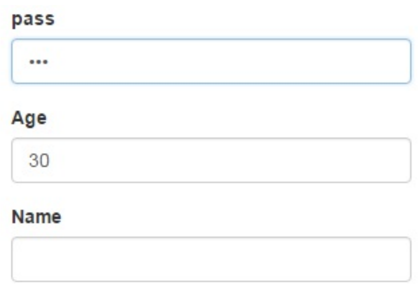
Exercise: Modify the code using `conditionalPanel` function

Example 2: Password

The app must be visible only if the password is correct.



pass



pass

•••

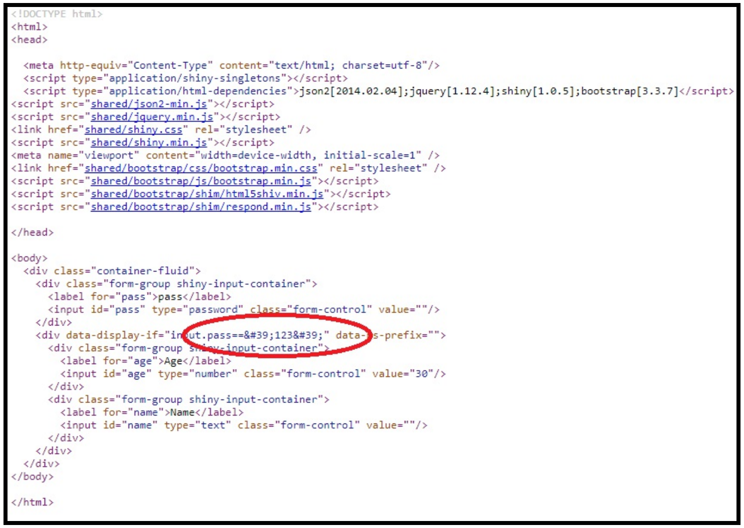
Age

30

Name

1. Using conditionalPanel

```r
ui <- fluidPage(
  passwordInput("pass","pass"),
    conditionalPanel(
      condition = "input.pass=='123'",
      numericInput("age", "Age", 30),
      textInput("name", "Name", "")
    )
)

server <- function(input, output) {}

shinyApp(ui = ui, server = server)
```



```html
<!DOCTYPE html>
<html>
<head>

  <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
  <script type="application/shiny-singletons"></script>
  <script type="application/html-dependencies">json2[2014.02.04];jquery[1.12.4];shiny[1.0.5];bootstrap[3.3.7]</script>
<script src="shared/json2-min.js"></script>
<script src="shared/jquery.min.js"></script>
<link href="shared/shiny.css" rel="stylesheet" />
<script src="shared/shiny.min.js"></script>
<meta name="viewport" content="width=device-width, initial-scale=1" />
<link href="shared/bootstrap/css/bootstrap.min.css" rel="stylesheet" />
<script src="shared/bootstrap/js/bootstrap.min.js"></script>
<script src="shared/bootstrap/shim/html5shiv.min.js"></script>
<script src="shared/bootstrap/shim/respond.min.js"></script>

</head>

<body>
  <div class="container-fluid">
    <div class="form-group shiny-input-container">
      <label for="pass">pass</label>
      <input id="pass" type="password" class="form-control" value=""/>
    </div>
    <div data-display-if="input.pass==&#39;123&#39;" data-ns-prefix="">
      <div class="form-group shiny-input-container">
        <label for="age">Age</label>
        <input id="age" type="number" class="form-control" value="30"/>
      </div>
      <div class="form-group shiny-input-container">
        <label for="name">Name</label>
        <input id="name" type="text" class="form-control" value=""/>
      </div>
    </div>
  </div>
</body>

</html>
```

Problem: Launch the app from a web browser and click the right side mouse button to see the "source code" and you will see the password.

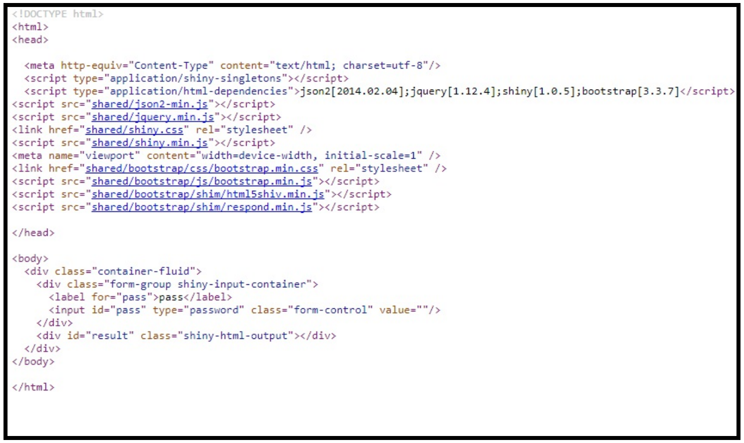
2. Password as an object/variable

```r
pass<-'123'

ui <- fluidPage(
  passwordInput("pass","pass"),
  conditionalPanel(
    condition = "input.pass==pass",
    numericInput("age", "Age", 30),
    textInput("name", "Name", "")
  )
)

server <- function(input, output) {}

shinyApp(ui = ui, server = server)
```

Problem: It doesn't work because no objects or variables can be passed thru `conditionalPanel` condition.

3. Using renderUI & uiOutput

```r
pass<-'123'

ui <- fluidPage(
  passwordInput("pass","pass"),
  uiOutput("result")
)

server <- function(input, output) {
   output$result <- renderUI({
     if (input$pass != pass)
       return(invisible(NULL))
     tagList(
       numericInput("age", "Age", 30),
       textInput("name", "Name", "")
     )
   })
}

shinyApp(ui = ui, server = server)
```



```html
<!DOCTYPE html>
<html>
<head>

  <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
  <script type="application/shiny-singletons"></script>
  <script type="application/html-dependencies">json2[2014.02.04];jquery[1.12.4];shiny[1.0.5];bootstrap[3.3.7]</script>
<script src="shared/json2-min.js"></script>
<script src="shared/jquery.min.js"></script>
<link href="shared/shiny.css" rel="stylesheet" />
<script src="shared/shiny.min.js"></script>
<meta name="viewport" content="width=device-width, initial-scale=1" />
<link href="shared/bootstrap/css/bootstrap.min.css" rel="stylesheet" />
<script src="shared/bootstrap/js/bootstrap.min.js"></script>
<script src="shared/bootstrap/shim/html5shiv.min.js"></script>
<script src="shared/bootstrap/shim/respond.min.js"></script>

</head>

<body>
  <div class="container-fluid">
    <div class="form-group shiny-input-container">
      <label for="pass">pass</label>
      <input id="pass" type="password" class="form-control" value=""/>
    </div>
    <div id="result" class="shiny-html-output"></div>
  </div>
</body>

</html>
```

It works!!!

Exercise

Create an app that:

1. loads a data base in SPSS.

2. the user choose the variables (one or more).

3. displays a summary of chosen variables.

4. download all variables of the data base in semicolon separated variables csv format.

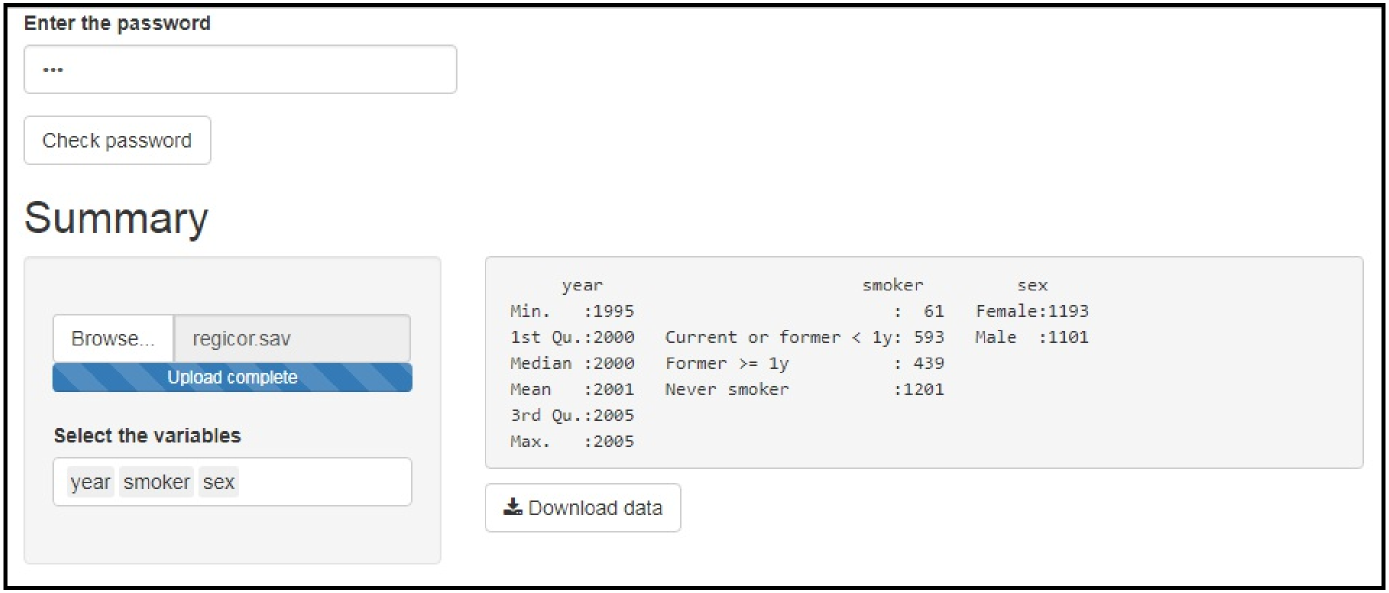
The app must be visible only when the user enter the correct password (e.g. "123"), which is verified after pressing a button.

Advanced: Load an excel file

- The user must have the chance of selecting the sheet
- list of available sheets

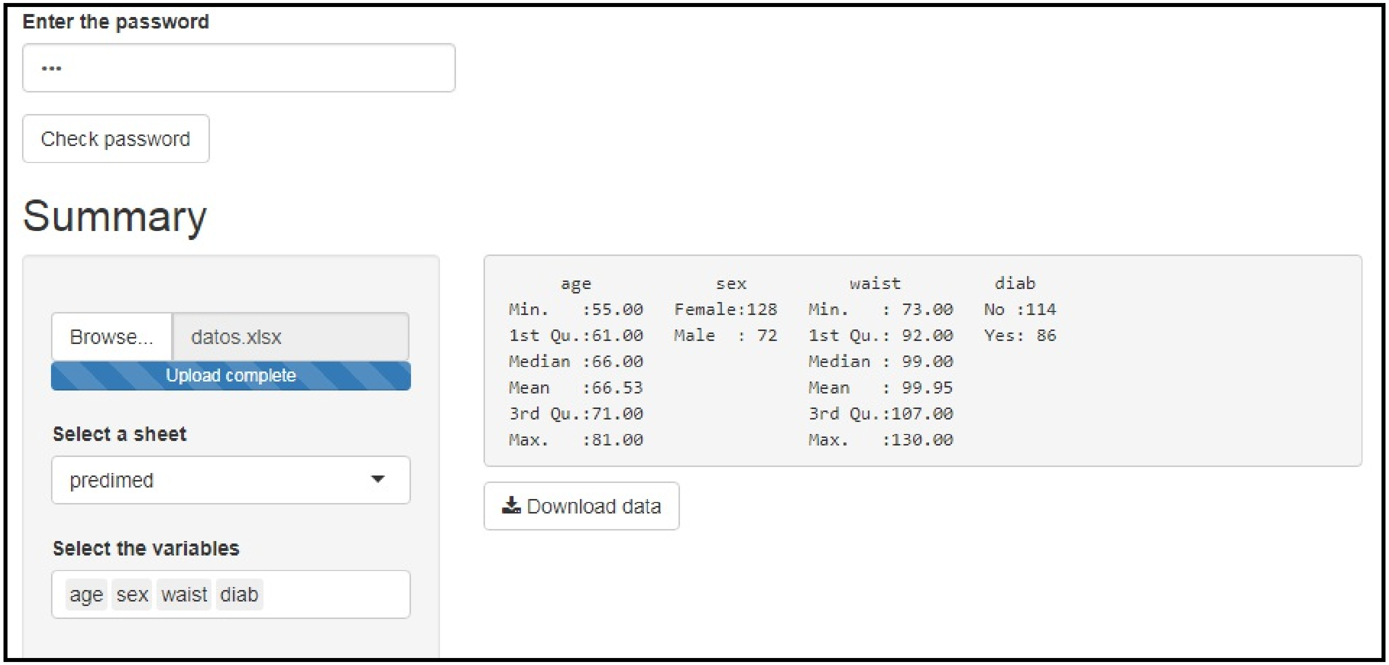
```
readxl::excel_sheets(input$file$datapath)
```

SPSS



```
      year                          smoker         sex      
 Min.   :1995                           :  61   Female:1193  
 1st Qu.:2000   Current or former < 1y: 593   Male  :1101  
 Median :2000   Former >= 1y            : 439                
 Mean   :2001   Never smoker            :1201                
 3rd Qu.:2005                                                
 Max.   :2005                                                
```

Excel



```
      age             sex          waist          diab    
 Min.   :55.00   Female:128   Min.   : 73.00   No :114  
 1st Qu.:61.00   Male  : 72   1st Qu.: 92.00   Yes: 86  
 Median :66.00                Median : 99.00            
 Mean   :66.53                Mean   : 99.95            
 3rd Qu.:71.00                3rd Qu.:107.00            
 Max.   :81.00                Max.   :130.00            
```