CURSO SOBRE BIOHERRAMIENTAS EN BIOESTADISTICA Y BIOINFORMATICA (1ªEdición)

Barcelona, 16, 17 y 18 de Mayo 2017

Cómo crear aplicaciones con Shiny

Parte III: Lógica de Shiny

Funcionamiento de Shiny

Secciones UI y Server

UI

- Se especifican los elementos y la estructura del formulario.
- Las instrucciones (elementos) se separan por comas.
- No puede haber sintaxis del tipo `input$elem`. Sólo se puede hacer referencia a los elementos de input mediante la función `conditionalPanel`.

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

Server

Se hacen los cálculos (tablas, las figuras, etc.) con

- Código propio de la parte UI (**`renderUI`**):

```
function(input, output){
  output$elementoUI <- renderUI(
    sliderInput("alias","etiq", 0, input$n, 0)
  )
}
```

- Objetos que se recalculan cuando es necesario y usados en varias partes:

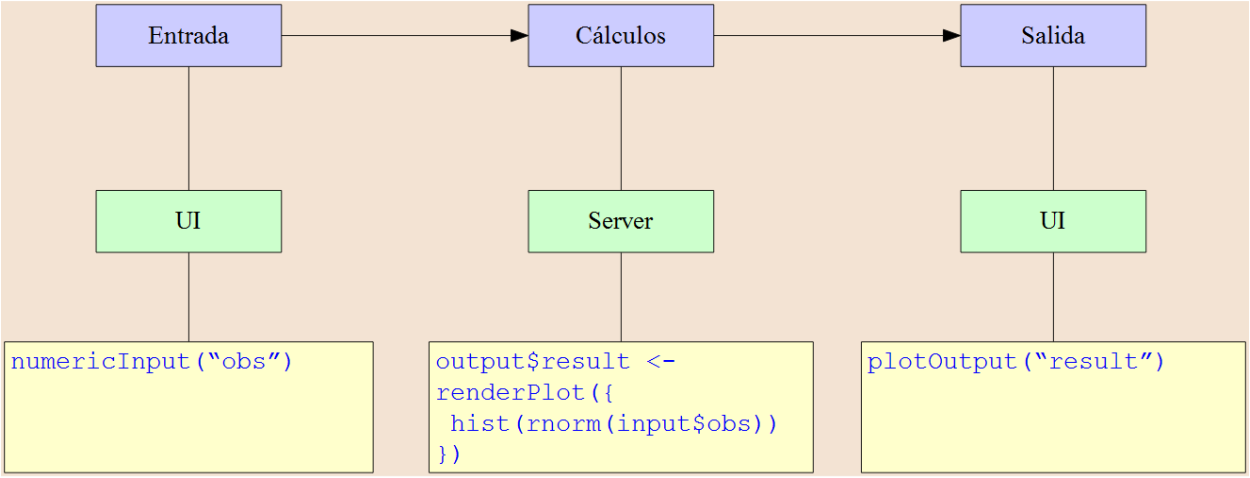
```
function(input, output){
  datos <- reactive(...)
```

```
  output$grafico <- renderPlot(
    hist(datos())
  )
  output$elemento <- renderPrint(
    summary(datos())
  )
}
```

- Estar vacío:

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

Pipeline



- Reactividad: creación y actualización de los elementos de dos listas **input** y **output**.
- Las listas **input** y **output** son los argumentos de la función definida en la sección **Server**.

Ejemplo 1: Test de una binomial

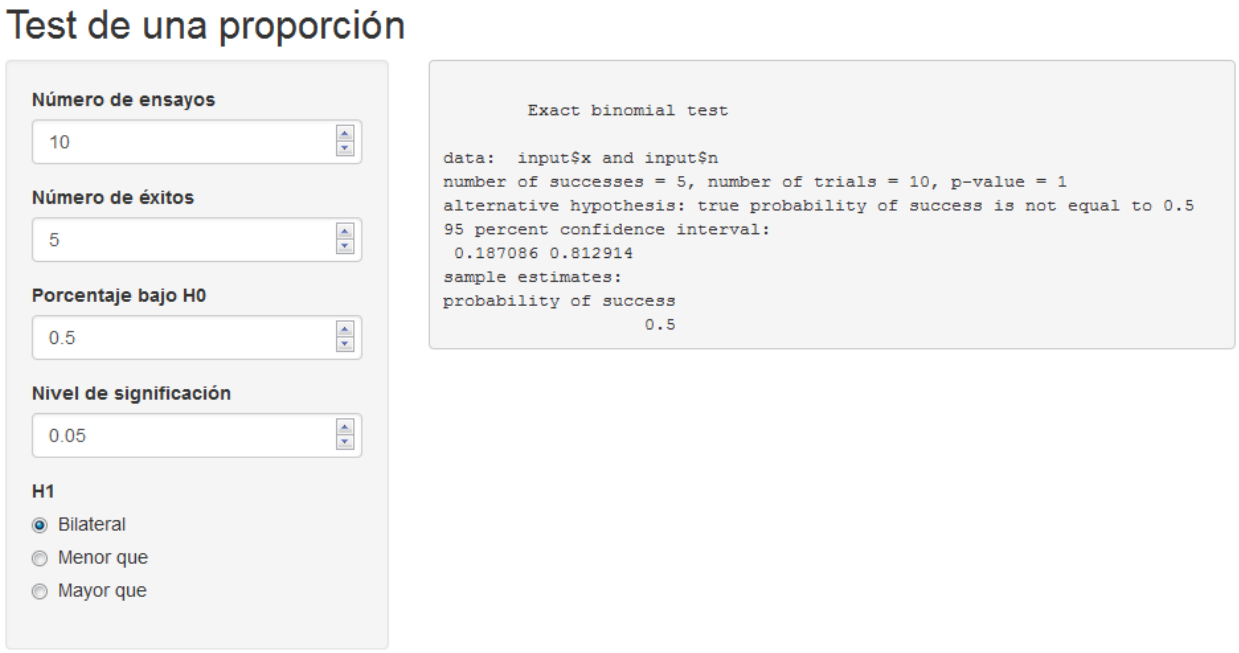


Figure 1:

Inputs:

- número total de ensayos,
- número de éxitos,
- proporción bajo H0.
- nivel de significación,
- sentido de la H1.

Outputs:

- el resultado del test.

==========

```
ui <- fluidPage(

  titlePanel("Test de una proporción"),

  sidebarLayout(
    sidebarPanel(
      numericInput("n", "Número de ensayos", 10),
      numericInput("x", "Número de éxitos", 5),
      numericInput("p0", "Porcentaje bajo H0", 0.5),
      numericInput("alpha", "Nivel de significación", value = 0.05),
      radioButtons("side", "H1", c("Bilateral"=1, "Menor que"=2, "Mayor que"=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

Control de la reactividad: Isolate

- En Shiny, cada vez que se cambia un elemento de entrada, se ejecutan todas las instrucciones del "bloque" ó "bloques" (`renderPrint`, `renderTable`, `reactive`, ...) donde esté el elemento modificado correspondiente.
- Si no se quiere que se ejecuten las instrucciones (ó se actualicen los resultados) hasta modificar todos los inputs deseados, hay que especificarlo. ⇒ función **`isolate`**.

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

  output$foo <- renderXXX({

    input$elem1

    isolate({
      ....
    })

  })
}
```

Ejemplo: Test de una proporción

Recuperamos el ejemplo anterior. Ahora se quiere que **no** se actualicen los resultados hasta que no se apriete el botón **calcular**

Figure 2:

```
ui <- fluidPage(
  titlePanel("Test de una proporción"),
  sidebarLayout(
    sidebarPanel(
      numericInput("n", "Número de ensayos", 10),
      numericInput("x", "Número de éxitos", 5),
      numericInput("p0", "Porcentaje bajo H0", 0.5),
      numericInput("alpha", "Nivel de significación", value = 0.05),
      radioButtons("side", "H1", c("Bilateral"=1, "Menor que"=2, "Mayor que"=3), 1),
      actionButton("calcula", "Calcula")
    ),
    mainPanel(
      verbatimTextOutput("result")
    )
  )
)

server <- function(input, output) {

  output$result <- renderPrint({
    if (input$calcula==0) return(invisible(NULL))
    isolate({
      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)
```

Función reactive

- En Shiny, los objetos se recalculan cuando el usuario varia algunos valores de entrada.
- Éstos pueden estar implicados en el cálculo de varios elementos de la lista **output** a la vez.
- Recalcularlos cada vez puede ser ineficiente. Por ejemplo, en la lectura de una base de datos grande o en general cálculos computacionalmente costosos.
- Se define el objeto reactive mediane la función `reactive`.

Ejemplo: datos iris para hacer un resumen y un gráfico.

```
function(input, output) {

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

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

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

}
```

Nota: El objeto `datos` no es un data.frame sinó una función, `datos()`. Sólo se actualiza cuando hace falta.

Ejercicio: crea la sección UI. ¿Qué elementos de entrada necesitas cómo mínimo?

Cargar datos

- Función **`fileInput`** en la sección **UI**.
- El elemento de la lista input contiene el nombre del archivo.
- Según el tipo de datos, se usará la función apropiada de **R** para leerlos: `spss.get`, `read.table`, `read.xls`, `xlsx`, etc.
- Es recomendable leer la base de datos como un objeto definido con la función **`reactive`**.

```
library(Hmisc) # leer SPSS
library(xlsx) # leer Excel

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

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

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

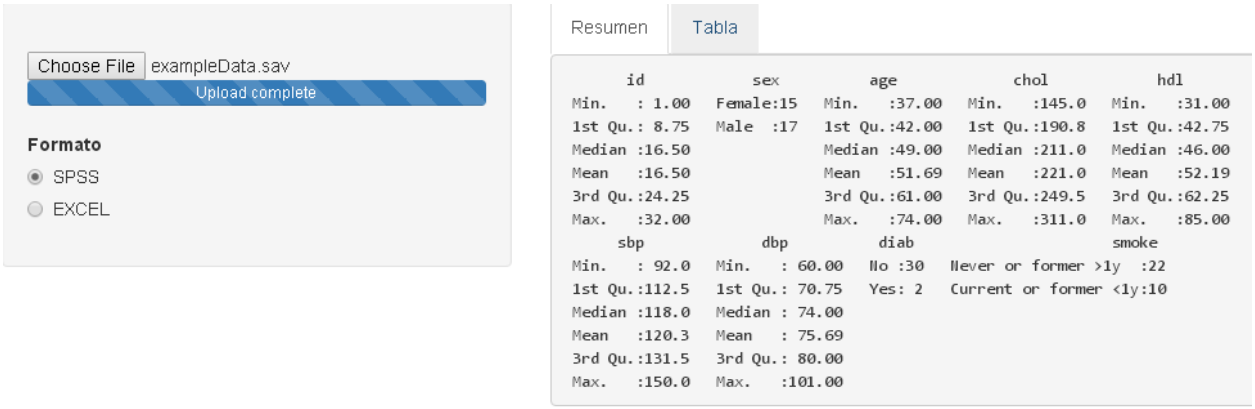


Figure 3:

Descargar archivos

Desde la sección server

- Función **`downloadHandler`** que crea un elemento de la lista **output**.
- Tiene dos argumentos:
- **`filename`**: función sin ningún input y que devuelve el nombre del archivo.
- **`content`**: función que crea y guarda el gráfico (ó el archivo en general) según el input (nombre del archivo).

Desde la sección UI

- En la sección **UI** se usa el *widget* **`downloadButton`**.

Ejemplo 1: descargar un gráfico

```
ui <- fluidPage(
  checkboxInput("group", "Distinguir especies"),
  plotOutput("result"),
  downloadButton('down', 'Descargar')
)

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

```
        pairs(iris[,-5],col=iris[,5])
      else
        pairs(iris[,-5])
      dev.off()
    }
  )
}

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

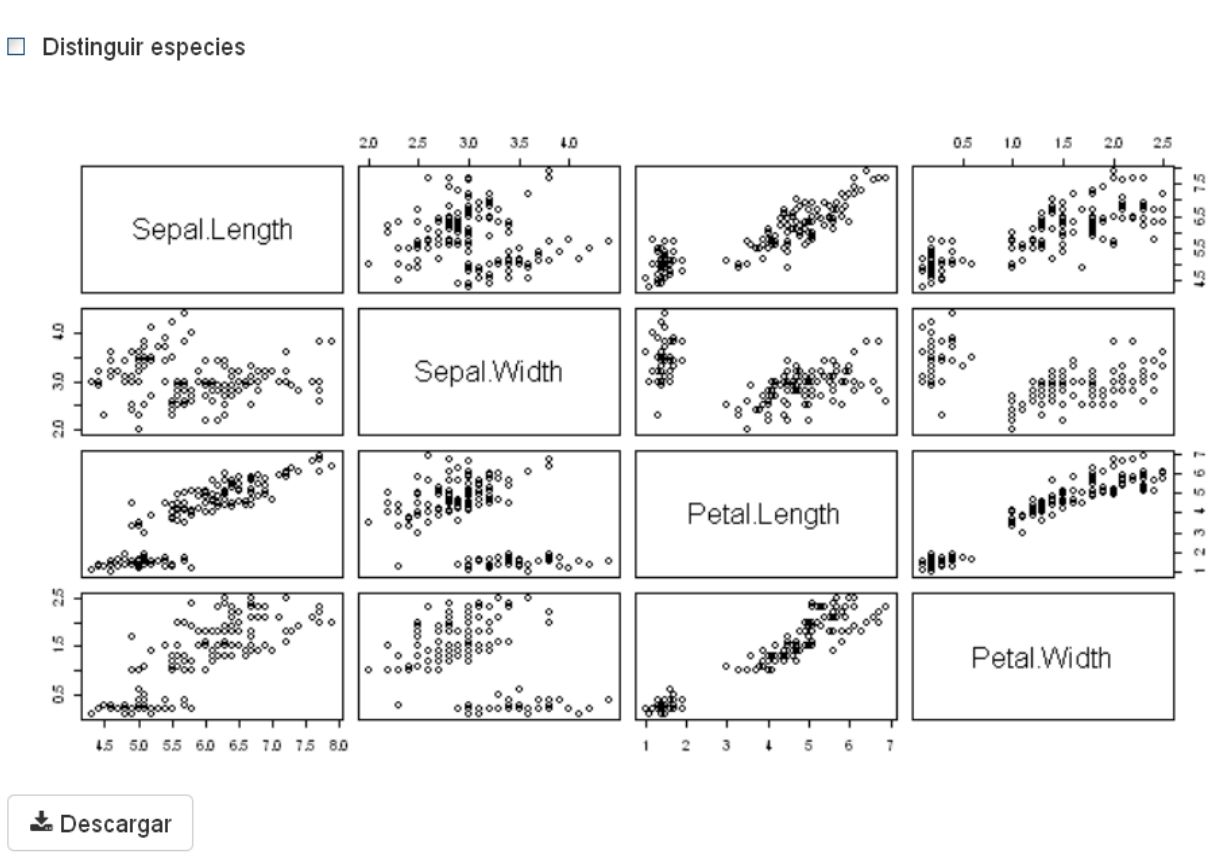


Figure 4:

Ejemplo 2: descargar una tabla de datos

```
ui <- fluidPage(
  downloadButton('down', 'Descargar'),
  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)
```

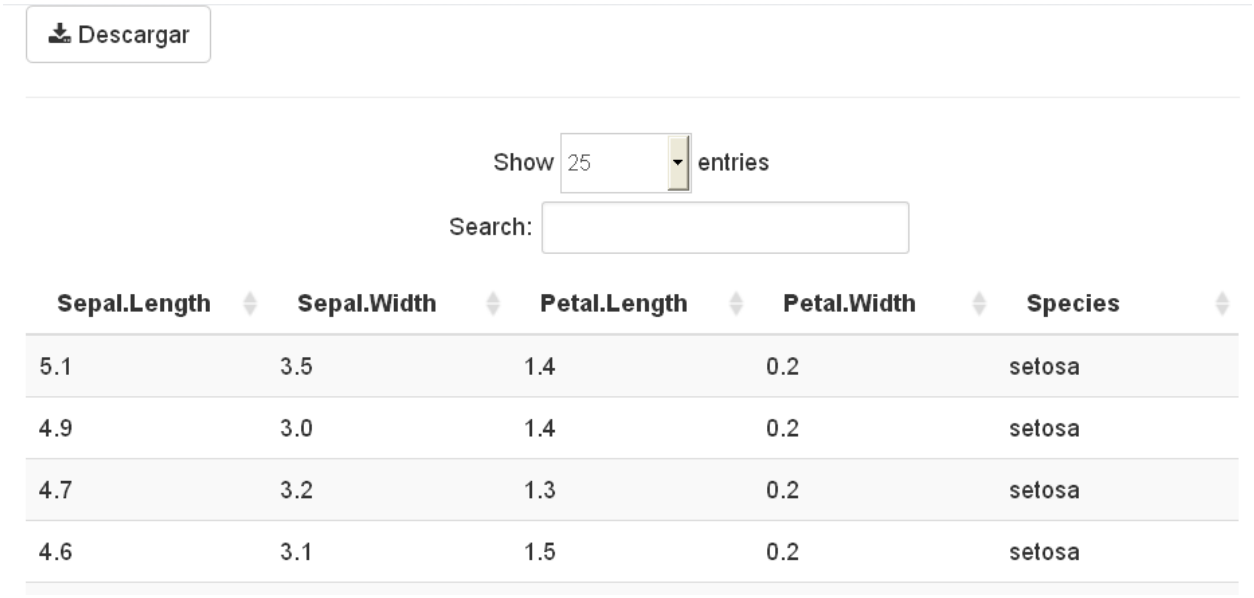


Figure 5:

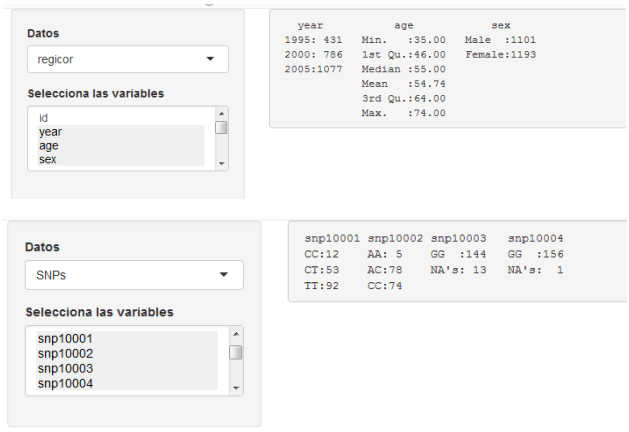
NOTA: Hay que lanzarlo desde un navegador de internet. Desde el visor de RStudio sólo se pueden descargar pdf.

Renderización de los elementos de entrada

- Hasta ahora habíamos visto cómo hacer aparecer uno o más elementos según el valor de otro elemento con la función `conditionalPanel`.
- Pero a menudo queremos modificar alguno de los aspectos o propiedades de los elementos de entrada según el valor de otro elemento.
- Existen las funciones **`renderUI`, `uiOutput`** para renderizar los elementos de forma muy potente y flexible.

Ejemplo. Análisis de variables según la base de datos

- El usuario tiene que elegir primero entre la base de datos disponibles en el package `compareGroups` (regicor, predimed ó SNPs).
- Tiene que aparecer la lista de variables correspondiente a la base de datos elegida.
- Se tiene que poder elegir qué variables son las descritas.
- En este caso, la **opciones de la lista** (argumento `choices`) de la función `selectInput` varia según la base de datos elegida.




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

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

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

```
  dd <- reactive(get(input$datos))

  output$result <- renderPrint(summary(dd()[,input$vars]))

  output$listvars <- renderUI({
    vars <- names(dd())
    selectInput("vars", "Selecciona las variables", vars, multiple=TRUE, selectize=FALSE)
  })

}

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

Ejercicio: Modifica el código usando `conditionalPanel`

Ejemplo 2: Password

La aplicación sólo puede ser visible si la clave introducida es correcta.

Con `conditionalPanel`

```
ui <- fluidPage(
  passwordInput("pass","pass"),
  conditionalPanel(
    condition = "input.pass=='123'",
    numericInput("edad", "Edad", 30),
    textInput("nombre", "Nombre", "")
  )
)

server <- function(input, output) {}

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

Problema: Si se exporta a un "browser" y con el botón derecho del ratón se puede ver la password en el "código fuente".

Password como objeto

```
pass<-'123'

ui <- fluidPage(
  passwordInput("pass","pass"),
  conditionalPanel(
    condition = "input.pass==pass",
    numericInput("edad", "Edad", 30),
    textInput("nombre", "Nombre", "")
  )
)

server <- function(input, output) {}

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

Problema: No funciona porque **no** se pueden poder variables dentro de la condición del `conditionalPanel`.

Con la función renderUI y uiOutput

```
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("edad", "Edad", 30),
      textInput("nombre", "Nombre", "")
    )
```

```
  })
}

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

Sí funciona

Código fuente (browser)

Con 'conditionalPanel

```
<!DOCTYPE html>
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
  <script type="application/shiny-singletons">
  <script type="application/html-dependencies">json2[2014.02.04];jquery[1.11.0];shiny[0.12.0];bootstrap[3.3.1]
<script src="shared/json2-min.js">
<script src="shared/jquery.min.js">
<link href="shared/shiny.css" rel="stylesheet" />
<script src="shared/shiny.min.js">
<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 src="shared/bootstrap/shim/html5shiv.min.js">
<script src="shared/bootstrap/shim/respond.min.js">

</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;">
      <div>
        <div class="form-group shiny-input-container">
          <label for="edad">Edad</label>
          <input id="edad" type="number" class="form-control" value="30"/>
        </div>
        <div class="form-group shiny-input-container">
          <label for="nombre">Nombre</label>
          <input id="nombre" type="text" class="form-control" value=""/>
        </div>
      </div>
    </div>
  </div>
</body>
</html>
```

Con `renderUI` y `uiOutput`

```
<!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];babel-
polyfill[6.7.2];shiny[1.0.0];bootstrap[3.3.7]</script><script src="shared/json2-min.js">
</script>
<script src="shared/jquery.min.js"></script>
<script src="shared/babel-polyfill.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>
```

Ejercicio

1. Cargue una base de datos en SPSS.
2. El usuario elija las variables (pueden ser más de una).
3. Aparezca un summary de las variables elegidas.
4. La aplicación tiene que ser visible sólo si se introduce el password correcto (p.e. 123), que se verifique cuando se apriete un botón.

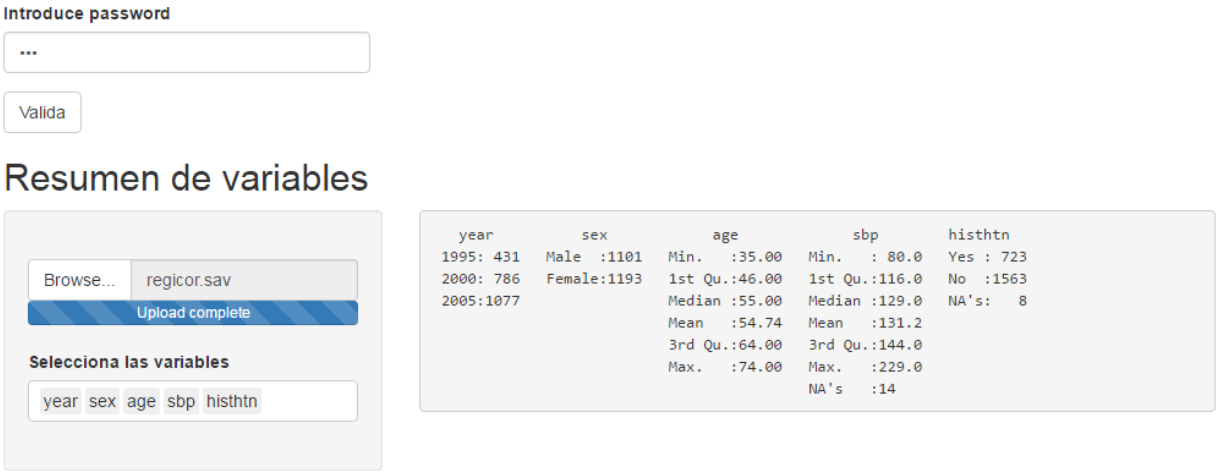


Figure 6: