Shiny. Entornos web con R

Aurora González, Antonio Maurandi[1]

Caldum.org, 26.Marzo.2014. Universidad de Murcia

[1]Sec. Apoyo Estadístico, Serv Apoyo a la Investigación

¿Qué es?

Shiny es un paquete que te permite convertir tus script de R en aplicaciones web interactivas que *todos* pueden usar.

Se instala escribendo en consola

```
install.packages("shiny")
```

- Ver ejemplos en Showcase, link
- Ver Tutorial: link

Para ejecutar el primer ejemplo se escribe

```
library(shiny)
runExample("01_hello")
```

Las aplicaciones Shiny tienen dos componentes:

1. Una interfaz de usuario `ui.R`
2. Un script de servidor o secuencia de comandos de servidor `server.R`.

Una aplicacion Shiny **es un directorio** que contiene ambos y otros documentos adicionales necesarios (*conjuntos de datos, etc,. . .*)

Código mínimo necesario:

ui.r

```
library(shiny)
shinyUI(pageWithSidebar(

  headerPanel("TITULO"),
  sidebarPanel(),
  mainPanel()
))
```

server.r

```
library(shiny)
shinyServer(function(input, output) {
})
```

lanzador

Cómo lanzar la aplicación

```
library(shiny)
runApp("dir")  # 'dir' es la ruta a la app

# en nuetsro caso,
runApp("~/aurorax/Taller-de-Shiny/App-0/")
```

Sliders: sliderInput (ui.r)

```
sliderInput(inputId, label, min, max, value,
step = NULL, animate = FALSE)
```

```
library(shiny)
shinyUI(pageWithSidebar(
  headerPanel("Elementos del sidebarPanel"),
  sidebarPanel(sliderInput("enteros", "Enteros:",
                           min=0, max=1000, value=500),
               sliderInput("decimales", "Decimales:",
                           min = 0, max = 1, value = 0.5, step= 0.1),
               sliderInput("animacion", "Animación:", 10, 200, 10,
                           step = 10, animate=animationOptions(loop=T)),
    ),
  mainPanel()
))
```

Figure : sliders

Sliders: sliderInput (app)

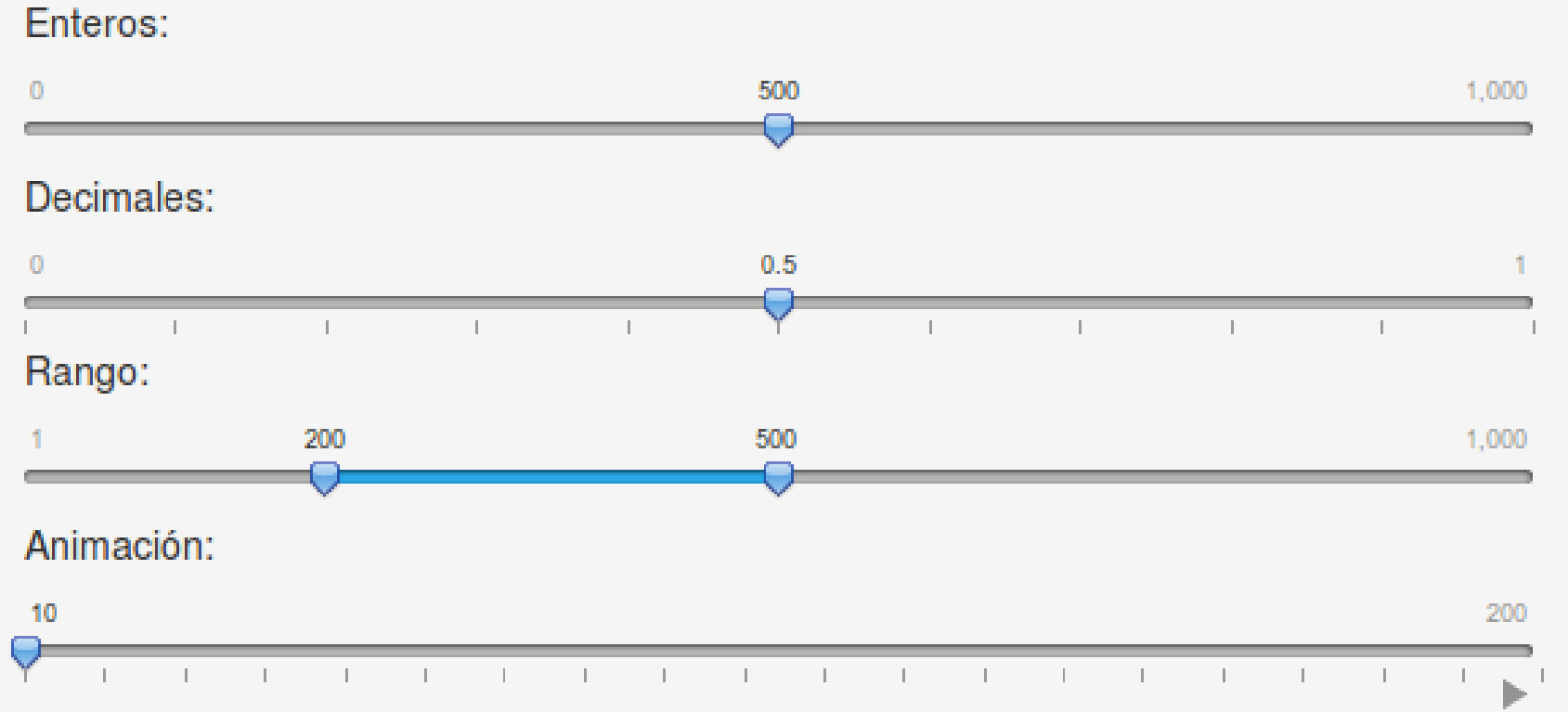

Figure : sliders

Radio buttons

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

```
radioButtons("dist", "Tipo de distribución:",
             c("Normal" = "norm", "Uniforme" = "unif",
               "Log-normal" = "lnorm","Exponencial" = "exp"),
             "Exponencial")  # este último es el valor que por
                             # defecto aparecerá seleccionado
```

Figure : RadioButton

Listas deplegables: selectInput

```
selectInput(inputId, label, choices, selected = NULL,
multiple = FALSE)
```

```r
selectInput("variable", "Variable:",
            c("Seleccion1" = "s1",
              "Seleccion2" = "s2",
              "Seleccion3" = "s3"),multiple=FALSE)
```

Figure : Desplegables

CheckBox: checkboxInput, checkboxGroupInput

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

checkboxGroupInput(inputId, label, choices,
                   selected = NULL)   (varias cajas)
```

```r
checkboxInput("header", "Mostrar titulo", FALSE)

checkboxGroupInput("variable", "Variable:",
                   c("Cilindrada" = "cyl",
                     "Transmision" = "tr",
                     "Engranaje" = "en"))
```

Figure : checkbox

Data input: numericInput

```
numericInput("obs", "Numero de observaciones:", 100)
```

Figure : numericInput

Decorar con textos explicativos

```
helpText("aclaraciones jur jur!")
```

Figure : helpText

¿Qué podemos hacer con todos estos *inputs* ?

Outputs: `ShinyServer` y `mainPanel`

Flujo: En el `ui.r` están los inputs, en el `server.r` se ejecutan scripts y se obtienen resultados (*outputs*) que se mostrarán en el mainpanel (`ui.r`).

En nuestro ejemplo, el primer objeto que encontramos en el `ui.r` es un *sliderInput* y lo hemos llamado `enteros`.

Es en el `server.r` donde se ejecuta código que podemos *sacar/mostrar* como texto, gráficos, etc...

Salida: gráfico

server.r

```
renderPlot(expr, width, height,...)
```

```
library(shiny)
shinyServer(function(input, output) {
  output$gra1 <- renderPlot({
    dist <- rnorm(input$enteros),
    hist(dist)
  })
})
```

Se ha llamado `gra1` al objeto. Creamos una distribución de el número de muestras que se introducen mediante la *barra enteros* y se crea un histograma que es lo que se ve.
Es la `ui.r` la que tiene que mostrar el histograma.

Salida: gráfico (ui.r)

mainPanel

plotOutput(OutpudId,...)

En la `ui.r` pasamos el objeto a `plotOutput` para servirlo en el `mainPanel` del `ui.r`.

```
mainPanel(plotOutput("gra1"))
```

Si nos interesara tener varios gráficos se puede dejar que se dibujen en el mismo panel principal, pero podemos crear otros paneles.

Salida: varios paneles

server.r

Creamos dos gráficos:

```
library(shiny)
shinyServer(function(input, output) {

    output$gra1 <- renderPlot({
        dist <- rnorm(input$animacion)
        hist(dist)
    })
    output$gra2 <- renderPlot({
        dist <- rnorm(input$enteros)
        hist(dist)
    })
})
```

y los llamamos en el mainPanel:

Salida: varios paneles

mainPanel

```
mainPanel(
  tabsetPanel(
    tabPanel('Histograma animacion',
             h3('Subtitulo'),plotOutput("gra1")),
    tabPanel('Histograma enteros',
             plotOutput("gra2"))
    )
  )
```

Figure : paneles

Salida: tablas

Para la creación de tablas se introduce un nuevo concepto que hace de Shiny una herramienta extraordinaria: **la reactividad**

- Como hemos visto, los gráficos cambiaban conforme ibamos cambiando las observaciones en las barras laterales. Es decir, Shiny es una aplicación *interactiva*.
- Si se quiere que suceda lo mismo con operaciones más complejas se usarán: *expresiones reactivas*

Modificamos el server para crear en un nuevo panel una tabla en la que van a figurar los valores que toman las barras.

Salida: tablas

server.r

La función a utilizar es `renderTable` y la expresión a añadir será la siguiente:

```
valorBarras <- reactive({
  data.frame(
    Nombre = c("Enteros",
               "Decimales",
               "Rango",
               "Animacion"),
    Value = as.character(c(input$enteros,
                           input$decimales,
                           input$rango,
                           input$animacion)),
    stringsAsFactors=FALSE)
})
```

Salida: tablas

server.r

junto con

```
output$valores <- renderTable({
 valorBarras()
})
```

Figure : tablas2

Salida: tablas

mainPanel

Se crea un nuevo panel dentro de tabsetPanel escribiendo

```
tabPanel("Tabla valores barras", tableOutput("valores"))
```

Ya hemos visto someramente cómo funciona Shiny.

Nota: Normalmente en el `server.r` hay una función que nos provee de una tabla, de una imagen... que luego el `ui.r` se encargar de sacar por pantalla como Output. Las posibilidades son:

Relación imputs y otputs

Las posibilidades son:
Server –> Ui –> Crea

- renderImage –> imageOutput –> Imagen
- renderPlot –> plotOutput –> Gráfico
- renderTable –> tableOutput –> Tabla
- renderText –> textOutput –> Texto
- renderText – >htmlOutput –> HTML
- renderText –> verbatimTextOutput –> Texto

Reactividad

Modelo de reactividad de shiny:

- *fuente reactiva*
- *conductor reactivo*
- *punto final de la reactividad*.

La estructura más simple:

La *fuente reactiva* suele ser lo que el usuario introduce y el *punto de parada* lo que se muestra por pantalla.

A lo que el usuario introduce se accede con el objeto input y a lo que se muestra por pantalla con el objeto output.

Reactividad

La estructura más simple

```
output$gra1 <- renderPlot({
    dist <- rnorm(input$enteros)
    hist(dist)
})
```

El objeto `output$gra1` es un punto final de la reactividad, y usa la fuente reactiva `input$enteros`. Cuando `input$enteros` cambia, a `output$gra1` se le notifica que necesita ejecutarse de nuevo.

Más cosas en

- Versión extendida de este documento: link aquí
  - Conductores reactivos
  - Control de la reactividad (*actionButtons*, *isolate*)
  - Subir y descargar ficheros (csv..): *fileInput*, *downloadButton*

Muchas Gracias

Figure : Master Yoda