Introducción a Shiny

Analytics Research Lab

Laura Fuentes, Diego Merlano, Jeffrey Villa

La presente guía te ayudará a conocer cómo crear una aplicación web con el paquete de R shiny, y los tipos de elementos que podrás incluir en tu aplicación o dashboard interactivo. El primer paso es instalar el paquete `{shiny}`, veamos cómo funciona.

¿Qué es shiny?

Shiny es un paquete de R que te brinda la posibilidad de crear una aplicación web o dashboard interactivo de manera muy sencilla. Para que nuestra aplicación pueda ser interactiva, esta cuenta con tres componentes básicos:

1. La interfaz del servidor (ui ó "user interface"): La interfaz define "como se ve" tu app, y es el componente que interactúa con el usuario. Este pide todos los inputs al usuario, y a demás se declaran los outputs que deberían generarse.

2. El servidor ("server"): Define "cómo se comporta" tu app, en este componente de la aplicación se escribe el código para calcular todos los outputs que deben darse dados los inputs que escogió el usuario.

3. ShinyApp: Es una función que combina el servidor y la interfaz, para convertirlos en la aplicación.

Creando mi primera app de shiny

Hay varias formas de crear una aplicación Shiny. La más simple es crear un nuevo directorio para su aplicación y poner un solo archivo llamado `app.R` en él. Este `app.R` archivo se usará para decirle a Shiny cómo debe verse su aplicación y cómo debe comportarse.

Pruébelo creando un nuevo directorio y agregando un app.Rarchivo con este aspecto:

```
library(shiny)

ui <- fluidPage()

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

shinyApp(ui,server)
```

Interfaz (ui)

Existen diferentes elementos que hacen parte de la interfaz, entre ellos tenemos los inputs, los outputs y layouts, veamos un poco sobre cada uno de ellos:

- Inputs:

Es la información que proporciona el usuario, y tienen 3 argumentos:

inputId: es la manera en la que identificamos el input, para conectar el front-end(interfaz) con el back-end(servidor). Si el inputid que seleccionaste es "costo", en el server deberás llamarlo como input$costo.

label: Es una etiqueta para que el input pueda ser leído por el usuario y entienda que información se le está pidiendo.

value: Valor por default del input.

Ejemplo:

```
sliderInput(inputId = "min", label =  "Limit (minimum)", value = 50, min = 0, max = 100)
```

function	widget
actionButton	Action Button
checkboxGroupInput	A group of check boxes
checkboxInput	A single check box
dateInput	A calendar to aid date selection
dateRangeInput	A pair of calendars for selecting a date range
fileInput	A file upload control wizard
helpText	Help text that can be added to an input form
numericInput	A field to enter numbers
radioButtons	A set of radio buttons
selectInput	A box with choices to select from
sliderInput	A slider bar
submitButton	A submit button
textInput	A field to enter text

Figure 1: Widgets

- Outputs

Toda función de output en la interfaz está conectada con una función render en el servidor. Existen 3 tipos de outputs: texto, tablas y gráficos. El primer argumento de un output es el `outputId`, con el cual llamaremos al output en el servidor `output$id`.

Ejemplo:

```
library(shiny)

ui <- fluidPage(
  textOutput(outputId = "text"),
  verbatimTextOutput(outputId = "code")
  )
```

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

  output$text <- renderText({
    "Hello friend!"
  })

  output$code <- renderPrint({
    summary(1:10)
  })
}

shinyApp(ui,server)
```

Output function	Creates
dataTableOutput	DataTable
htmlOutput	raw HTML
imageOutput	image
plotOutput	plot
tableOutput	table
textOutput	text
uiOutput	raw HTML
verbatimTextOutput	text

Figure 2: Outputs

- Layouts

Con los layouts o capas puedes ordenar visualmente tu app. Podemos organizar nuestra app, añadiendo un layout de columna lateral izquierda con las funciones `sidebarLayout()`y `sidebarPanel()` para los inputs, y otro layout para un panel principal con la funcion `mainPanel()` para los outputs.

Ejemplo:

```
fluidPage(
  titlePanel(
    # título app/descripción
  ),
  sidebarLayout(
    sidebarPanel(
      # inputs
    ),
    mainPanel(
      # outputs
    )
  )
)
```

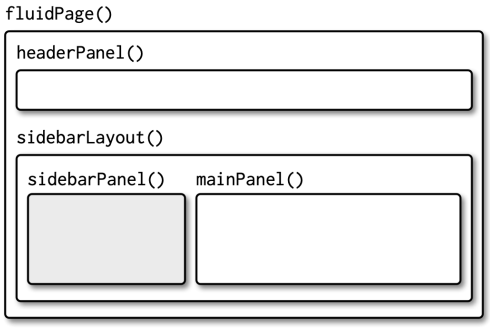

Figure 3: Sidebar layout

Otra opcion basica para organizar tu app es en varias filas, crea filas con la función `fluidRow()`, y columnas con `column()`.

```
fluidPage(
  fluidRow(
    column(4,
      ...
    ),
    column(8,
      ...
    )
  ),
  fluidRow(
    column(6,
      ...
    ),
    column(6,
      ...
    )
  )
)
```

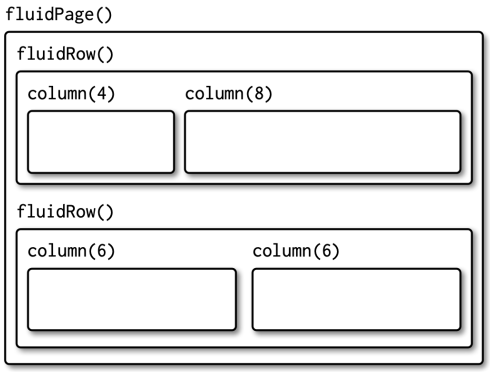

Figure 4: Multirow layout

Servidor (server)

Shiny utiliza un estilo de programación llamado "reactivo" el cual le permite actualizar automáticamente los outputs una vez los inputs cambien. Los outputs declarados en el `ui` siempre son calculados en el `server` con una función reactiva `render`.

Además también existen expresiones reactivas conformadas por la función `reactive()` las cuales te permiten optimizar tu código, haciéndolo mucho más sencillo y menos repetitivo.

Veamos el siguiente ejemplo: Imaginemos que queremos comparar la distribución de dos conjuntos de datos simulados con un histograma trama. En la siguiente imagen, veremos el resultado final deseado.

```
library(shiny)

ui <- fluidPage(
  fluidRow(
    column(4,
           "Distribution 1",
           numericInput("n1", label = "n", value = 1000, min = 1),
           numericInput("mean1", label = "μ", value = 0, step = 0.1),
           numericInput("sd1", label = "Desv", value = 0.5, min = 0.1, step = 0.1)
    ),
    column(4,
           "Distribution 2",
           numericInput("n2", label = "n", value = 1000, min = 1),
           numericInput("mean2", label = "μ", value = 0, step = 0.1),
           numericInput("sd2", label = "Desv", value = 0.5, min = 0.1, step = 0.1)
    )
  ),
  fluidRow(
    column(9, plotOutput("hist"))
  )
)

server <- function(input, output, session) {
  output$hist <- renderPlot({
    set.seed(1)
    x1 <- rnorm(input$n1, input$mean1, input$sd1)
    x2 <- rnorm(input$n2, input$mean2, input$sd2)

    # First distribution
    hist(x1, breaks=30, xlim=c(-5,5), col=rgb(1,0,0,0.5), xlab="height",
         main= "x1 and x2 distributions")

    # Second with add=T to plot on top
    hist(x2, breaks=30, xlim=c(-5,5), col=rgb(0,0,1,0.5), add=T)

    # Add legend
    legend("topright",legend=c("x1","x2"), col=c(rgb(1,0,0,0.5),
                                                 rgb(0,0,1,0.5)), pt.cex=2, pch=15 )

  }, res = 96)

}
```

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

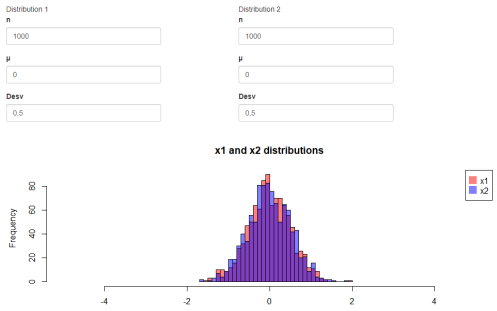

Figure 5: Multirow layout

Las expresiones reactivas son importantes por dos razones:

- Le dan a Shiny más información para que pueda volver a calcular menos cuando las entradas cambian, lo que hace que las aplicaciones sean más eficientes.
- Hacen que sea más fácil para los humanos comprender la aplicación al simplificar el gráfico reactivo.

Las expresiones reactivas tienen una variedad de entradas y salidas:

- Al igual que las entradas, puede utilizar los resultados de una expresión reactiva en una salida.
- Al igual que las salidas, las expresiones reactivas dependen de las entradas y saben - automáticamente cuándo necesitan actualizarse.

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

    text <- reactive({
      paste0("Hello ", input$"insertar inputId", "!")
      })

    output$"insertar outputId" <- renderText(
      text()
      )
}
```

Recursos adicionales

- Mastering Shiny, *Hadley Wickham*. Link: https://mastering-shiny.org/index.html
- https://shiny.rstudio.com/
- Outstanding User Interfaces with Shiny, *David Granjon*. Link: https://divadnojnarg.github.io/outstanding-shiny-ui/