

Aplicaciones interactivas diseñadas con shiny

Julio Mulero

julio.mulero@ua.es

Departamento de Matemáticas
Universidad de Alicante



Outline

- 1 Qué es shiny

- 2 El diseño de la aplicación
 - La estructura de la interfaz
 - El aspecto de la interfaz

- 3 El contenido de la aplicación

- 4 La interactividad de shiny

Outline

- 1 Qué es shiny
- 2 El diseño de la aplicación
 - La estructura de la interfaz
 - El aspecto de la interfaz
- 3 El contenido de la aplicación
- 4 La interactividad de shiny

Qué es shiny

Shiny

Shiny es un paquete de R que permite construir aplicaciones web interactivas a partir de los scripts de R.



Instalación

Si aún no tienes instalado Shiny:

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


Qué es shiny

Interés de shiny en la docencia

- La **interactividad** de estas aplicaciones permite manipular los datos sin tener que manipular el código. De hecho, en la naturaleza de shiny subyace un concepto aún más fuerte: la **reactividad** (habría que comentar estas dos características).
- Las aplicaciones web creadas con shiny pueden estar enfocadas a numerosos ámbitos: investigación, profesional o, por supuesto, la docencia. Estas aplicaciones pueden abrirse desde el propio ordenador, una tablet o incluso el móvil.
- En particular, shiny permite diseñar recursos docentes que ayuden al profesor mostrar los contenidos de las asignaturas en clase.
- Estos recursos pueden facilitar a los alumnos la práctica y la ejercitación de los problemas desde su propia casa con ejercicios de carácter diverso. Serán más o menos complejos dependiendo de las intervenciones del sujeto y del modo en que realice los ejercicios, haciéndole saber a este cual es su nivel de desarrollo o aprendizaje.

▶ Ver correlación

▶ Ver problemas



Creando un proyecto shiny

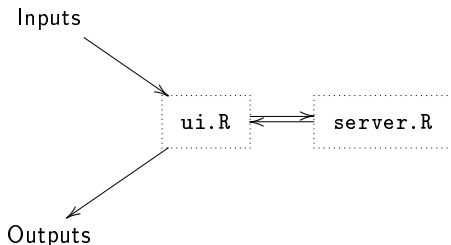
File Create project from: Project Type:
New project ▷ New directory ▷ Shiny Web Application

El resultado es un directorio nuevo con el nombre que hayamos elegido en el que aparecen dos archivos (`ui.R` y `server.R`).

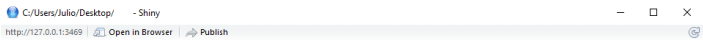
Creando un proyecto shiny

Una app de shiny consta (al menos) de dos archivos:

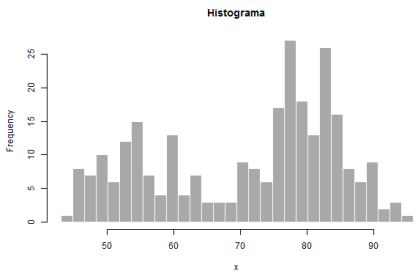
- un script para la interfaz del usuario, (user-interface, `ui.R`), que recibe los inputs y muestra los outputs, y
- un script para los cálculos (`server.R`), que realiza los cálculos necesarios.



Un primer ejemplo



Los histogramas



El archivo ui.R

```
library(shiny)

# UI
shinyUI(fluidPage(

  # Título de la aplicación
  titlePanel("Los histogramas"),

  # Columna lateral con una barra deslizable para el número de
  # intervalos
  sidebarLayout(
    sidebarPanel(
      sliderInput("bins",
                  "Número de intervalos:",
                  min = 1,
                  max = 50,
                  value = 30)
    ),

    # Panel principal con el gráfico generado
    mainPanel(
      plotOutput("distPlot")
    )
  )
)
```

El archivo server.R

```

library(shiny)

# SERVER
shinyServer(function(input, output) {

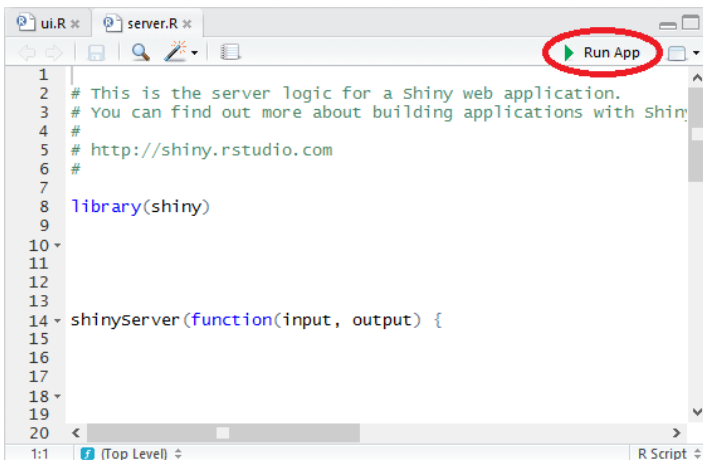
  # La expresión que obtiene el histograma es
  # un renderPlot que quiere decir que:
  #
  # 1) Es "reactiva" y debe re-ejecutarse automáticamente
  # cuando el input cambie.
  # 2) El output es un gráfico.

  output$distPlot <- renderPlot({
    x <- faithful[, 2] # Old Faithful Geyser data
    bins <- seq(min(x), max(x), length.out = input$bins + 1)

    # Dibujar el histograma con dicho número de intervalos
    hist(x, breaks = bins, col = 'darkgray', border = 'white',
         main='Histograma')
  })
})

```

Lanzando la aplicación



The screenshot shows the RStudio interface with two tabs: 'ui.R' and 'server.R'. The 'server.R' tab is active, displaying the following R code:

```
1 |  
2 | # This is the server logic for a Shiny web application.  
3 | # You can find out more about building applications with shiny.  
4 | #  
5 | # http://shiny.rstudio.com  
6 | #  
7 |  
8 | library(shiny)  
9 |  
10 |  
11 |  
12 |  
13 |  
14 | shinyServer(function(input, output) {  
15 |  
16 |  
17 |  
18 |  
19 |  
20 |
```

The 'Run App' button, represented by a green play icon and the text 'Run App', is circled in red in the top right corner of the editor window.

Lanzando la aplicación

El código de la aplicación shiny puede estar también dentro de un solo script y puede ejecutarse de la siguiente manera:

```
library(shiny)

# UI
u<-shinyUI(TipoDePagina(
))

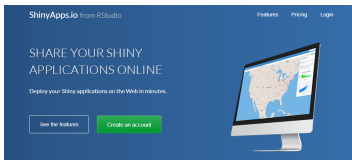
# SERVER
s<-shinyServer(function(input , output) {

})

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

Compartiendo la aplicación

- shinyapps: 440\$ al año una suscripción básica / 1100\$ al año una suscripción estándar.



Looking for an easy way to deploy Shiny?

ShinyApps.io hosts your Shiny applications & documents.

- Shiny server!: “gratiso”.

Outline

- 1 Qué es shiny

- 2 El diseño de la aplicación
 - La estructura de la interfaz
 - El aspecto de la interfaz

- 3 El contenido de la aplicación

- 4 La interactividad de shiny

El diseño de la interfaz

- Todos los aspectos del diseño se indican en el archivo `ui.R`.
- El diseño de la interfaz de usuario está relacionado con:
 - 1 La **estructura** (barras laterales, paneles, pestañas, etc.).
 - 2 El **aspecto** (colores, tipos de fuentes, etc.).

La estructura de la interfaz

La **estructura** de la interfaz de usuario puede ser de tres tipos:

- `pageWithSideBar` (deprecated=obsoleto, `fluidPage` es más flexible).
- `fluidPage`.
- `fixedPage` (igual que el anterior, pero fuerza la anchura de la página web).

En cualquier caso:

- `titlePanel`: título de la aplicación.
- `sidebarLayout`:
 - `sidebarPanel`: barra lateral de contenido.
 - `mainPanel`: panel principal de contenido.

(1) La estructura de la interfaz

La estructura básica es:

```
library(shiny)

# UI
shinyUI(fluidPage(
  #Título de la aplicación
  titlePanel("Título"),

  #Contenido de la aplicación
  sidebarLayout(

    #Panel lateral
    sidebarPanel("Título del panel lateral",
      Contenido
    ),

    #Panel principal
    mainPanel(
      Contenido
    )
  )
))
```

(2) La estructura de la interfaz

Se pueden incluir **filas de contenido** con `fluidRow`, `column`. La función `wellPanel` crea un panel lateral.

```
shinyUI(fluidPage(
  titlePanel("Hello Shiny!"),
  fluidRow(
    column(3,
      wellPanel(
        sliderInput("obs1", "Number of observations:",
                    min = 1, max = 1000, value = 500)
      )),
    column(8,
      plotOutput("distPlot1")
    )),
  fluidRow(
    column(2,
      wellPanel(
        sliderInput("obs2", "Number of observations:",
                    min = 1, max = 1000, value = 500)
      )),
```

(3) La estructura de la interfaz

Se pueden incluir **diferentes pestañas con una misma barra lateral** usando las funciones `tabsetPanel` y `tabPanel`.

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

(4) La estructura de la interfaz

Se pueden incluir **diferentes pestañas con diferentes barras laterales** usando las funciones `conditionalPanel`, `tabsetPanel` y `tabpanel`.

```

sidebarLayout(
  sidebarPanel(
    conditionalPanel(condition="input.conditionedPanels_==_",
      hist'"),
    ...
  ),
  conditionalPanel(condition="input.conditionedPanels_==_",
    datos'"),
  ...
)),
mainPanel(
  tabsetPanel(
    tabPanel("hist",
      ...
    ),
    tabPanel("datos",
      ...
    ),
    id = "conditionedPanels"
  ))))

```

Outline

- 1 Qué es shiny

- 2 El diseño de la aplicación
 - La estructura de la interfaz
 - El aspecto de la interfaz

- 3 El contenido de la aplicación

- 4 La interactividad de shiny

El aspecto de la interfaz

El paquete `shinythemes` dispone de diferentes temas que cambian colores y tipos de fuentes:

- `default.`
- `cerulean.`
- `cosmo.`
- `flatly.`
- `journal.`
- `readable.`
- `spacelab.`
- `united.`

[▶ Ver los temas](#)

El aspecto de la interfaz

Para cambiar el tema de la interfaz, se debe instalar el paquete

```
install.packages("shinythemes")
```

y escribir en el archivo ui.R:

```
library(shiny)
library(shinythemes)

# UI
shinyUI(fluidPage(theme = shinytheme("cerulean"),
  ...
))
```

Outline

- 1 Qué es shiny

- 2 El diseño de la aplicación
 - La estructura de la interfaz
 - El aspecto de la interfaz

- 3 El contenido de la aplicación

- 4 La interactividad de shiny

Incluyendo contenido de tipo HTML

```

#Panel lateral
  sidebarPanel(h5("En esta aplicación podrás practicar gran parte de los contenidos vistos en clase."),
    h4("Las variables cuantitativas discretas son variables cuyo conjunto de posibles valores es finito o infinito numerable. Para organizar estos datos se utilizan las tablas de frecuencia.", style = "color:red")),
  br(),
  p(actionButton("renovar", "Nuevos datos"), align="right"),
  br(),
  p(img(src="ua.png", height = 45, width = 200), align="center"),
  p("Aplicación realizada por", a("Julio Mulero", href="mailto:julio.mulero@ua.es"), ".")
  )
#Panel principal
  mainPanel(

  )))
  
```

Outline

- 1 Qué es shiny
- 2 El diseño de la aplicación
 - La estructura de la interfaz
 - El aspecto de la interfaz
- 3 El contenido de la aplicación
- 4 La interactividad de shiny

Esquema simple

El esquema más simple toma como entrada una fuente reactiva y, a partir de ella, da como resultado un punto final reactivo:



Supongamos que queremos mostrar histogramas de conjuntos de datos generados por una distribución normal. Pedimos al usuario que indique un tamaño muestral `input$obs` y se obtendrá el histograma `output$distPlot`:



▶ Hello Shiny!

(5) Esquema simple: ui.R

```

library(shiny)

shinyUI(fluidPage(

  titlePanel("Hello Shiny!"),

  sidebarLayout(
    sidebarPanel(
      sliderInput("obs",
                  "Number of observations:",
                  min = 1,
                  max = 1000,
                  value = 500)
    ),

    mainPanel(
      plotOutput("distPlot")
    )))

```


(5) Esquema simple: server.R

```
library(shiny)

shinyServer(function(input, output) {

  output$distPlot <- renderPlot({
    dist <- rnorm(input$obs)
    hist(dist)
  })

})
```

Esquema simple II

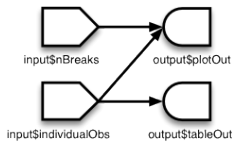
Los inputs pueden usarse tantas veces como queramos y pueden generar tantos outputs como queramos.

Supongamos que deseamos construir/analizar el histograma de cierto conjunto de datos fijado `faithful$eruptions`. Pedimos al usuario que indique:

- un número de intervalos: un parámetro numérico `input$nBreaks`.
- si desea ver las observaciones individuales mediante una marca en el histograma y la tabla de datos: un parámetro lógico `input$individualObs`.

Se mostrará entonces `output$plotOut` (y dependiendo de haber marcado la opción, más cosas).

El esquema sería:



(6) Esquema simple II: ui.R

```
library("shiny")

shinyUI(fluidPage(

  titlePanel("Esquema simple II"),

  sidebarLayout(
    sidebarPanel(
      selectInput(inputId = "nBreaks",
                  label = "Number of bins in histogram (
                    approximate):",
                  choices = c(10, 20, 35, 50),
                  selected = 20),

      checkboxInput(inputId = "individualObs",
                    label = strong("Show individual observations
                    "),
                    value = FALSE)
    ),

    mainPanel(
      plotOutput(outputId = "plotOut", height = "300px"),
      textOutput(outputId = "textOut")
    )))
```

(6) Esquema simple II: server.R

```
shinyServer(function(input, output) {
  output$plotOut <- renderPlot({
    hist(faithful$eruptions, breaks = as.numeric(input$nBreaks))
    if (input$individualObs) rug(faithful$eruptions)
  })

  output$textOut <- renderText({
    if (input$individualObs) faithful$eruptions
    else NULL
  })
})
```

Algo parecido, aunque no igual, puede consultarse en:

[▶ Ver enlace](#)

Los inputs

Como se ha observado en los ejemplos anteriores, los inputs pueden ser números y parámetros lógicos. Más generalmente, shiny dispone de los siguientes inputs que se incorporan mediante lo que se denominan **widgets** (sus nombres acaban generalmente con **Input**):

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

(7) Los inputs: ui.R

```
library(shiny)

# UI
shinyUI(fluidPage(

  # Título de la aplicación
  titlePanel("Los histogramas"),

  # Columna lateral con una barra deslizable para el número de
  # intervalos
  sidebarLayout(
    sidebarPanel(
      sliderInput("bins",
                  "Número de intervalos:",
                  min = 1,
                  max = 50,
                  value = 30)
    ),

    # Panel principal con el gráfico generado
    mainPanel(
      plotOutput("distPlot")
    )
  )
)
```

(7) Los inputs: server.R

```

library(shiny)

# SERVER
shinyServer(function(input, output) {

  # La expresión que obtiene el histograma es
  # un renderPlot que quiere decir que:
  #
  # 1) Es "reactiva" y debe re-ejecutarse automáticamente
  # cuando el input cambie.
  # 2) El output es un gráfico.

  output$distPlot <- renderPlot({
    x <- faithful[, 2] # Old Faithful Geyser data
    bins <- seq(min(x), max(x), length.out = input$bins + 1)

    # Dibujar el histograma con dicho número de intervalos
    hist(x, breaks = bins, col = 'darkgray', border = 'white',
         main='Histograma')
  })
})

```

Los inputs

Una vez que se ha incluido un widget en ui.R (en cuya sintaxis hay que indicar el nombre que se le da a la variable que representa), se utilizará como

```
input$nombre
```

dentro de `server.R`.

De input a output: server.R

Los inputs anteriores que se introducen en ui.R se envían a server.R y se utilizan para obtener los outputs. Las operaciones que se realizan en server.R con los inputs y que dan como resultado los outputs, son de tipo reactivo (sus nombres empiezan por `render` y acaban dependiendo del tipo de objeto que devuelven):

Tipo de objeto que se obtiene	Uso
<code>renderImage</code>	images (saved as a link to a source file)
<code>renderPlot</code>	plots
<code>renderPrint</code>	any printed output
<code>renderTable</code>	data frame, matrix, other table like structures
<code>renderText</code>	character strings
<code>renderUI</code>	a Shiny tag object or HTML

De input a output: server.R

En las funciones `render`, aparecerán como argumentos los `input$nombre` que hayamos introducido en `ui.R`.

Las funciones `render` se asignan a objetos del tipo

`output$nombre`

De input a output: server.R

```
library(shiny)

# SERVER
shinyServer(function(input, output) {

  # La expresión que obtiene el histograma es
  # un renderPlot que quiere decir que:
  #
  # 1) Es "reactiva" y debe re-ejecutarse automáticamente
  #    cuando el input cambie.
  # 2) El output es un gráfico.

  output$distPlot <- renderPlot({
    x <- faithful[, 2] # Old Faithful Geyser data
    bins <- seq(min(x), max(x), length.out = input$bins + 1)

    # Dibujar el histograma con dicho número de intervalos
    hist(x, breaks = bins, col = 'darkgray', border = 'white')
  })
})
```

Los outputs: ui.R

Los resultados que se han obtenido con el proceso anterior se devuelven a ui.R que deberá mostrarlos (o no). Según el tipo de output, debe indicarse en ui.R utilizando las siguientes opciones:

Tipo de output	Significado
htmlOutput	raw HTML
imageOutput	image
plotOutput	plot
tableOutput	table
textOutput	text
uiOutput	raw HTML
verbatimTextOutput	text

Los outputs: ui.R

Las funciones `Output` necesitan como argumento el "nombre" del `output$nombre`.

```
plotOutput("nombre")
```

Los outputs: ui.R

```
library(shiny)

# UI
shinyUI(fluidPage(

  # Título de la aplicación
  titlePanel("Los histogramas"),

  # Columna lateral con una barra deslizable para el número de
  # intervalos
  sidebarLayout(
    sidebarPanel(
      sliderInput("bins",
                  "Number of bins:",
                  min = 1,
                  max = 50,
                  value = 30)
    ),

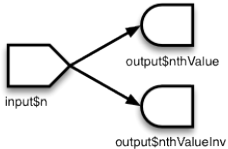
    # Panel principal con el gráfico generado
    mainPanel(
      plotOutput("distPlot")
    )))
```

Los outputs

En server.R	↔	En ui.R	↔	Tipo de objeto
<code>renderImage</code>	↔	<code>imageOutput</code>	↔	Imagen
<code>renderPlot</code>	↔	<code>plotOutput</code>	↔	Gráfico
<code>renderTable</code>	↔	<code>tableOutput</code>	↔	Tabla
<code>renderText</code>	↔	<code>textOutput</code>	↔	Texto
<code>renderText</code>	↔	<code>htmlOutput</code>	↔	HTML
<code>renderText</code>	↔	<code>verbatimTextOutput</code>	↔	Texto

Los conductores reactivos: server.R

Hasta ahora, los inputs (fuentes reactivas) permiten obtener los outputs (puntos finales reactivos) de manera directa. Sin embargo, a veces es necesario transformar los inputs para hacerlos más “tratables” en el server.R de forma que “limpiamos” el código. Esta modificación pasa por convertirlos en conductores reactivos de manera que puedan ser dependientes y, a la vez, tener objetos dependientes. Supongamos que queremos obtener un término de la sucesión de Fibonacci y su inverso. Pedimos al usuario que indique una posición `input$n` y obtendremos `output$nthValue` y `output$nthValueInv`. El esquema sería:



(8) Conductores reactivos: ui.R

```

library(shiny)

shinyUI(fluidPage(

  titlePanel("La sucesión de Fibonacci"),

  sidebarLayout(
    sidebarPanel(
      numericInput("n", "n:", 10,
                  min = 1, max = 100)
    ),

    mainPanel(
      h3("El n-ésimo término es:"),
      textOutput("nthValue"),
      h3("El n-ésimo término es:"),
      textOutput("nthValueInv")
    )))

```

(8) Conductores reactivos: server.R

```

fib <- function(n) ifelse(n<3, 1, fib(n-1)+fib(n-2))

shinyServer(function(input, output) {
  output$nthValue <- renderText({ fib(as.numeric(input$n)) })
  output$nthValueInv <- renderText({ 1 / fib(as.numeric(input$n))
    }) })
})

```

Observemos que antes de `shinyServer` podemos cargar todas las funciones necesarias. En este caso, está definida la función `fib` que devuelve el n -ésimo término de la sucesión de Fibonacci. Esta función se utiliza dos veces dentro de `shinyServer`.

(8) Conductores reactivos II: server.R

Si queremos mejorar la eficiencia de nuestra aplicación, podemos calcular en primer lugar el término correspondiente de la sucesión y utilizarlo después en la generación de los outputs. El archivo server.R “mejorado” sería:

```
fib <- function(n) ifelse(n<3, 1, fib(n-1)+fib(n-2))

shinyServer(function(input, output) {
  currentFib      <- reactive({ fib(as.numeric(input$n)) })

  output$nthValue <- renderText({ currentFib() })
  output$nthValueInv <- renderText({ 1 / currentFib() })
})
```

Observemos que:

- `currentFib` depende del valor `input$n`
 - `output$nthValue` y `output$nthValueInv` dependen de `currentFib`.
- y, a su vez, d

(9) Conductores reactivos II: ui.R

Otra situación interesante que pueden resolver los conductores reactivos es: supongamos, por ejemplo, que queremos que el usuario proporcione un conjunto de datos de forma escrita. Desgraciadamente, no existe un widget que permita introducir un vector de datos tal y como los trata R: `c(...)`.

```
library(shiny)

shinyUI(fluidPage(

  titlePanel("Introduciendo datos"),

  sidebarLayout(
    sidebarPanel(

      textInput('vect', 'Introduce los datos (separados por
                comas)', "0,1,2")
    ),

    mainPanel(
      h4('Tus datos son:'),
      verbatimTextOutput("data"),
      h4('La media es:'),
      verbatimTextOutput("mean")
    )))
```

(9) Conductores reactivos II: server.R

```
library(shiny)

numextractall <- function(string){
  unlist(regmatches(string, gregexpr("[[:digit:]]+\\. * [[:digit:]]
    *", string)), use.names=FALSE)
}

shinyServer(function(input, output) {

  vector <- reactive({
    as.numeric(numextractall(input$vect))
  })

  output$data <- renderPrint({
    vector()
  })

  output$mean <- renderPrint({
    mean(vector())
  })

})
```

Recomendaciones

- ¡Mucho cuidado con las llaves, las comas y los paréntesis!
- ¡Mucho cuidado con los nombres de los inputs y outputs!

Aplicaciones interactivas diseñadas con shiny

Julio Mulero

julio.mulero@ua.es

Departamento de Matemáticas
Universidad de Alicante

