

Shiny

Sesión I: Conociendo Shiny

Hèctor Perpiñán Fabuel - Unitat de Bioestadística, IRBLLEIDA
07 de Junio, 2017

Contenidos

1. Introducción
2. Motivación
3. Empezando a caminar con Shiny
4. Componentes básicas de una aplicación Shiny
 - User interface (ui)
 - Layouts
 - Widgets: Inputs
 - Server
 - Widgets: Outputs
 - Resctividad

1. Introducción

1. Introducción

Hasta el momento hemos hablado de la creación de documentos mediante *R Markdown*, interactivos o no, pensando en un uso local.

Estamos en la sociedad de la información e internet lo domina todo. **¿Podemos utilizar internet para ayudar en la investigación reproducible?**

- Shiny (<http://shiny.rstudio.com/>) es un paquete de R (<https://www.r-project.org/>) que nos permite desarrollar aplicaciones web interactivas.
 - Esta siendo desarrollado por los creadores de RStudio (<https://www.rstudio.com/>), destacan especialmente **Winston Chang**, Joe Cheng, **JJ Allaire**, Yihui Xie, Jonathan McPherson, ...
 - No necesita conocimientos de lenguajes de programación como HTML, CSS o JavaScript.
- **Tutorial y ejemplos:** shiny.rstudio.com (<http://shiny.rstudio.com>)

2. Motivación

Ejemplos

(<https://www.rstudio.com/products/shiny/shiny-user-showcase/>)

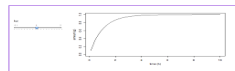
(<http://webpopix.org/shiny/ShinyExamples>)
(<http://webpopix.org/shiny/ShinyExamples>)
Pharmacometrics: some Shiny applications

Marc Lavielle
April 29, 2015

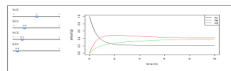
These applications require the [mxR package](#) for the simulation and visualization of longitudinal data.

Introduction to pharmacokinetics modelling

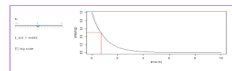
Here, Shiny applications are embedded in html pages to illustrate some basic concepts in pharmacometrics.



Modelling the absorption process



Modelling the distribution process



Modelling the elimination process

Building a pharmacokinetics model

These Shiny applications allow one to interactively modify the PK model and the dosage regimen, and visualize the plasmatic concentration given by this model. Here, a R code is automatically updated on the fly according to the settings of the application.



(<http://webpopix.org/shiny/ShinyExamples>)

(<https://gallery.shinyapps.io/EDsimulation/>)
(<https://gallery.shinyapps.io/EDsimulation/>)

Emergency Department Simulation

Home Customize Statistics Patient Data Plots Case Study Future Work About Us Help

Welcome to the Online ED Simulator!

What We Do

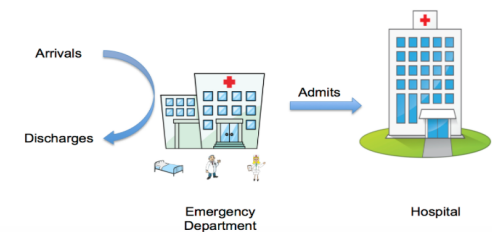
The ED Simulator is a comprehensive simulation tool to quantify and predict site-specific emergency department crowding. Designed by applied mathematicians, statisticians, and emergency department physicians, the ED Simulator can accurately model patient flow dynamics in a wide variety of user-specified ED settings. Our objective is to collaborate with ED physicians and managers to design custom simulation tools to improve patient throughput.

What's Here

In this online version, we put the power and flexibility of our model in your hands. Our online version includes:

- Intuitive and accessible interfaces
- Highly customizable ED environments
- Traditional throughput metrics
- Advanced utilization statistics
- Real-time cost analysis

Overview Get Started



(<https://gallery.shinyapps.io/EDsimulation/>)

(<http://omimexplorer.research.bcm.edu:3838/>)
(<http://omimexplorer.research.bcm.edu:3838/>)

El papel de Shiny en la Investigación Reproducible

Shiny es una herramienta perfecta para compartir nuestros estudios, permitiendo el acceso a la información. Esto genera numerosos beneficios:

- **Mayor visibilidad** de nuestro trabajo (**mayor control**)
- **Transparencia**
- Extensión a **artículos científicos** (apéndices más actuales)
 - Cada vez más revistas piden acceso al código y los datos
- Complemento a **paquetes de R**
- 📖 **Objetivo: Aprender a crear aplicaciones web utilizando Shiny.**

3. Empezando a caminar con Shiny

Instalación de Shiny y creación de la primera app

Paso 1. Abrimos RStudio.

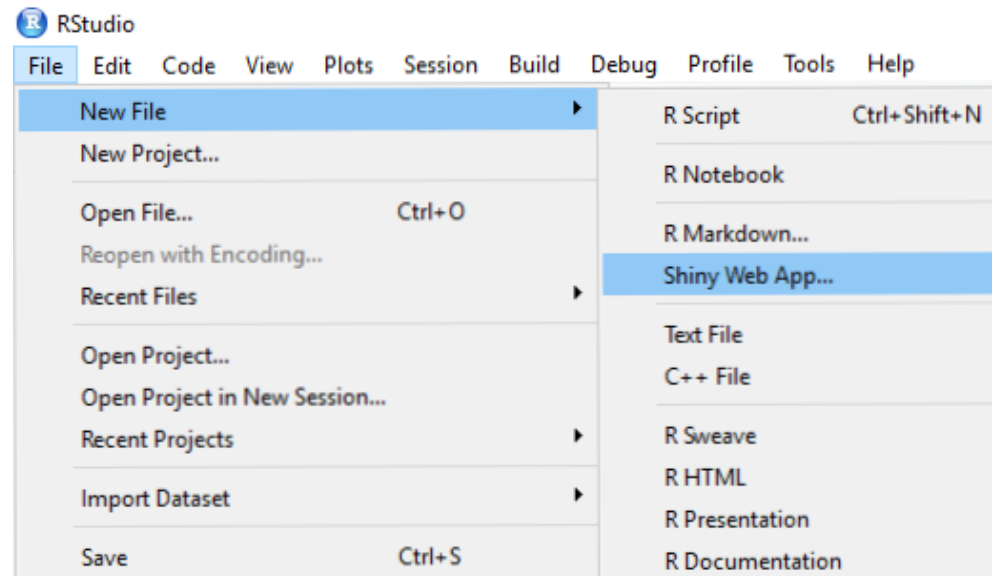
Paso 2. Instalamos el paquete "shiny" desde CRAN (en caso de no haberlo instalado previamente):

```
install.packages("shiny", dependencies = TRUE)
```



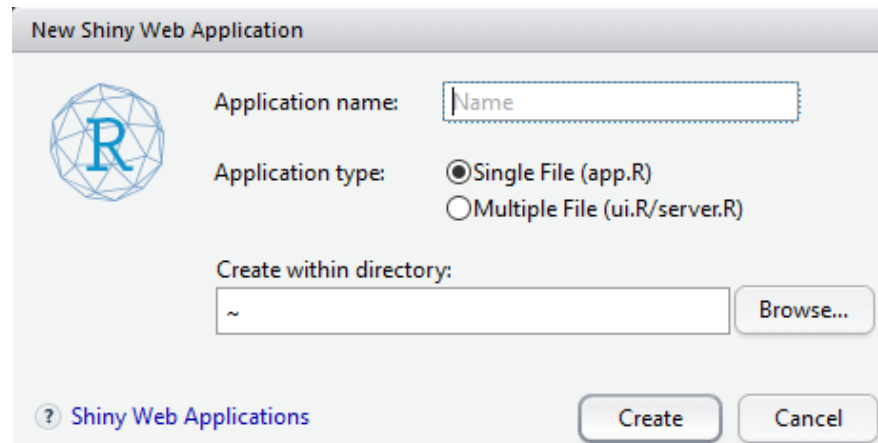
Instalación de Shiny y creación de la primera app

Paso 3. Seleccionamos **File >> New File >> Shiny Web App....**



Instalación de Shiny y creación de la primera app

Paso 4. Indicamos el nombre de la aplicación, el número de archivos que queremos y el directorio de creación.

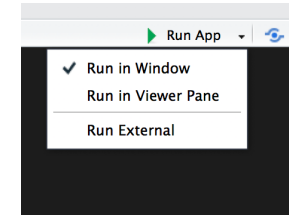


1. **Single File:** Se crea una carpeta con el nombre de la aplicación en el directorio indicado. Esta carpeta contendrá el archivo **app.R**.
2. **Multiple File:** Se crea una carpeta con el nombre de la aplicación en el directorio indicado. Esta carpeta contendrá los archivos **ui.R** y **server.R**.

Ejemplo Shiny >> Single File (app.R)

```
1 #  
2 # This is a Shiny web application. You can run the application by clicking  
3 # the 'Run App' button above.  
4 #  
5 # Find out more about building applications with Shiny here:  
6 #  
7 # http://shiny.rstudio.com/  
8 #  
9  
10 library(shiny)  
11  
12 # Define UI for application that draws a histogram  
13 ui <- fluidPage(  
14   # Application title  
15   titlePanel("Old Faithful Geyser Data"),  
16   # Sidebar with a slider input for number of bins  
17   sidebarLayout(  
18     sidebarPanel(  
19       sliderInput("bins",  
20         "Number of bins:",  
21         min = 1,  
22         max = 50,  
23         value = 30)  
24     ),  
25     # Show a plot of the generated distribution  
26     mainPanel(  
27       plotOutput("distPlot")  
28     )  
29   )  
30 )  
31  
32  
33  
34  
35 # Define server logic required to draw a histogram  
36 server <- function(input, output) {  
37   output$distPlot <- renderPlot({  
38     # generate bins based on input$bins from ui.R  
39     x <- faithful[, 2]  
40     bins <- seq(min(x), max(x), length.out = input$bins + 1)  
41  
42     # draw the histogram with the specified number of bins  
43     hist(x, breaks = bins, col = 'darkgray', border = 'white')  
44   })  
45 }  
46  
47  
48 # Run the application  
49 shinyApp(ui = ui, server = server)
```

Ejemplo Shiny >> Multiple File (ui.R/server.R)

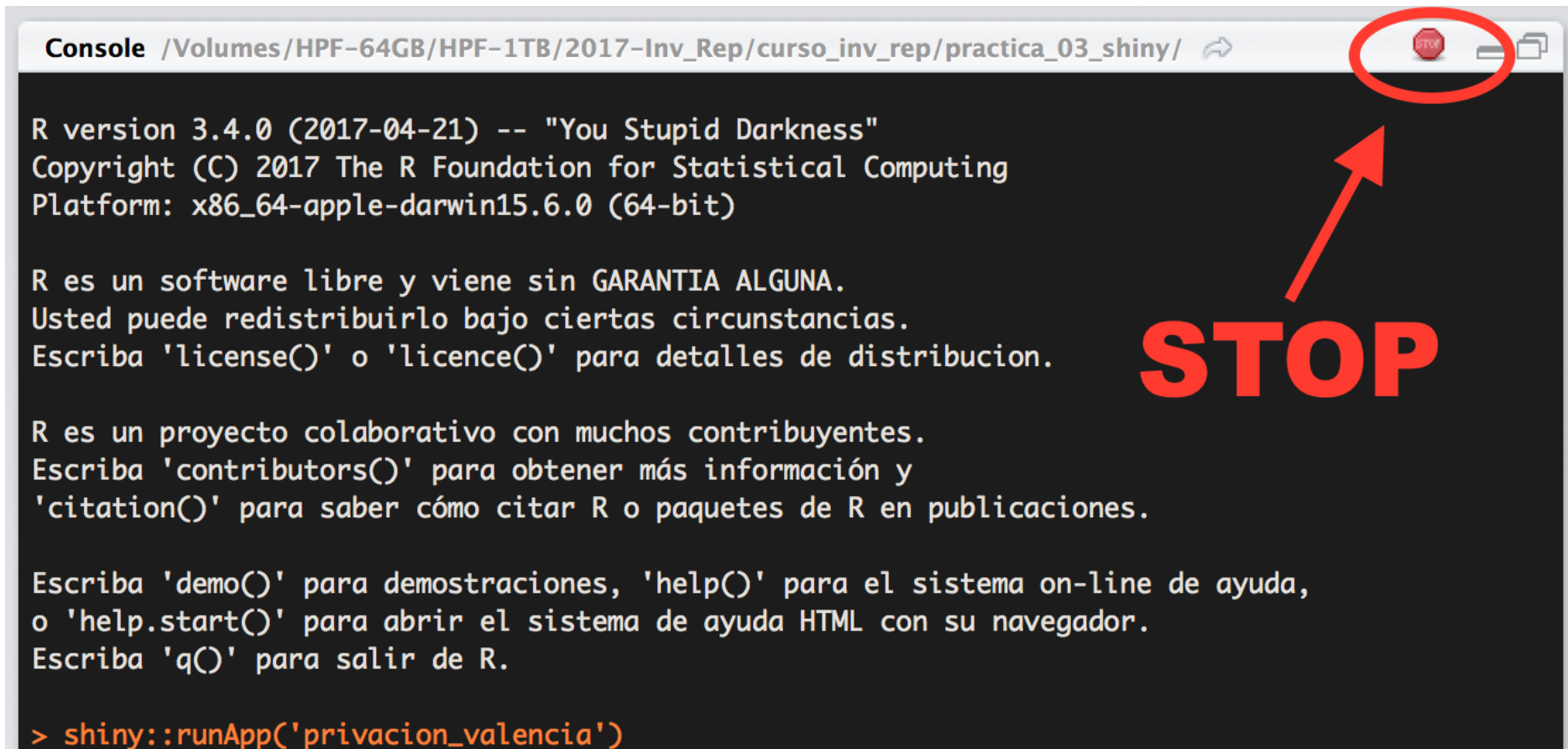


```
1 #  
2 # This is the user-interface definition of a Shiny web application. You can  
3 # run the application by clicking 'Run App' above.  
4 #  
5 # Find out more about building applications with shiny here:  
6 #  
7 # http://shiny.rstudio.com/  
8 #  
9  
10 library(shiny) ui.R  
11  
12 # Define UI for application that draws a histogram  
13 shinyUI(FluidPage(  
14   # Application title  
15   titlePanel("Old Faithful Geyser Data"),  
16   # Sidebar with a slider input for number of bins  
17   sidebarLayout(  
18     sidebarPanel(  
19       sliderInput("bins",  
20         "Number of bins:",  
21         min = 1,  
22         max = 50,  
23         value = 30)  
24     ),  
25     # Show a plot of the generated distribution  
26     mainPanel(  
27       plotOutput("distPlot")  
28     )  
29   )  
30 )  
31 )  
32 )  
33 )  
34 )
```

```
1 #  
2 # This is the server logic of a Shiny web application. You can run the  
3 # application by clicking 'Run App' above.  
4 #  
5 # Find out more about building applications with shiny here:  
6 #  
7 # http://shiny.rstudio.com/  
8 #  
9  
10 library(shiny) server.R  
11  
12 # Define server logic required to draw a histogram  
13 shinyServer(function(input, output) {  
14   output$distPlot <- renderPlot({  
15     # generate bins based on input$bins from ui.R  
16     x <- faithful[, 2]  
17     bins <- seq(min(x), max(x), length.out = input$bins + 1)  
18     # draw the histogram with the specified number of bins  
19     hist(x, breaks = bins, col = 'darkgray', border = 'white')  
20   })  
21 }  
22 )  
23 )  
24 )  
25 )  
26 )  
27 )
```

Ejemplo Shiny >> Cerrar la app

Si no cerramos la app, esta se queda ejecutándose en R y no podremos seguir trabajando.



```
Console /Volumes/HPF-64GB/HPF-1TB/2017-Inv_Rep/curso_inv_rep/practica_03_shiny/

R version 3.4.0 (2017-04-21) -- "You Stupid Darkness"
Copyright (C) 2017 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin15.6.0 (64-bit)

R es un software libre y viene sin GARANTIA ALGUNA.
Usted puede redistribuirlo bajo ciertas circunstancias.
Escriba 'license()' o 'licence()' para detalles de distribucion.

R es un proyecto colaborativo con muchos contribuyentes.
Escriba 'contributors()' para obtener más información y
'citation()' para saber cómo citar R o paquetes de R en publicaciones.

Escriba 'demo()' para demostraciones, 'help()' para el sistema on-line de ayuda,
o 'help.start()' para abrir el sistema de ayuda HTML con su navegador.
Escriba 'q()' para salir de R.

> shiny::runApp('privacion_valencia')
```

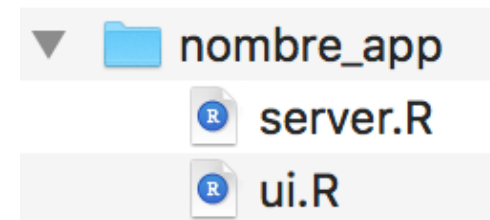
4. Componentes básicas de una aplicación Shiny

4. Componentes básicas de una aplicación Shiny

Para crear una aplicación Shiny es suficiente con tener los scripts **ui.R** y **server.R** (**app.R**) dentro de la misma carpeta. El nombre de estos scripts no se puede cambiar.

La estructura es la siguiente:

- Carpeta **nombre_app**:



- Podemos añadir más archivos (.R, .RData, .jpg, .css, ...) que serán ejecutados desde **ui.R** o **server.R**.

User interface (ui / ui.R)

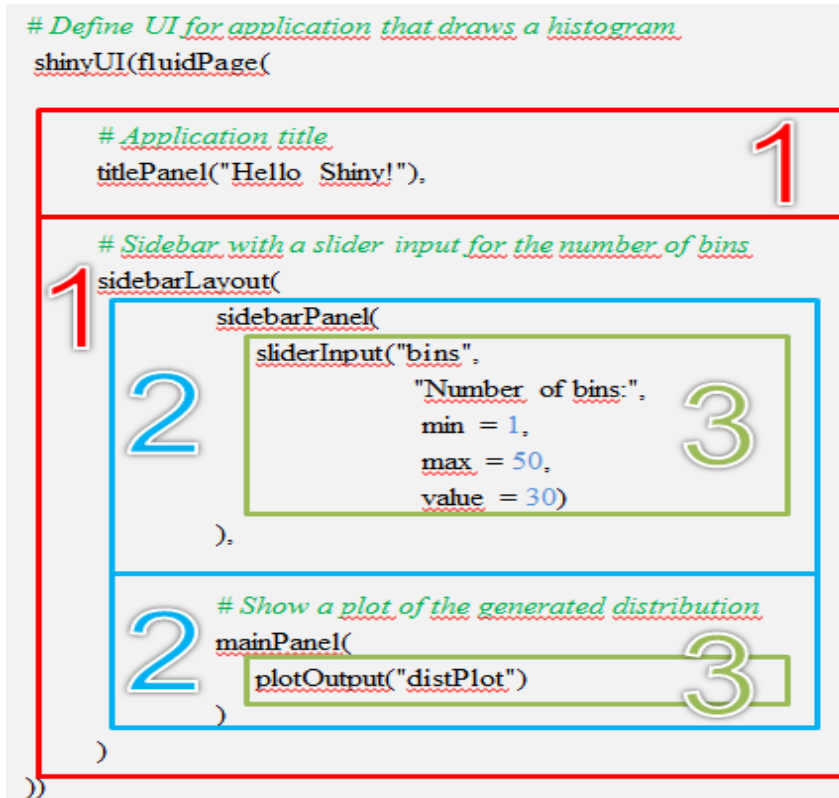
ui / ui.R (User interface)

- La construcción de la interfaz de usuario es como montar un puzzle.

```
# Define UI for application that draws a histogram
shinyUI(fluidPage(

  # Application title
  titlePanel("Hello Shiny!"),

  # Sidebar with a slider input for the number of bins
  sidebarLayout(
    sidebarPanel(
      sliderInput("bins",
        "Number of bins:",
        min = 1,
        max = 50,
        value = 30)
    ),
    # Show a plot of the generated distribution
    mainPanel(
      plotOutput("distPlot")
    )
  )
))
```



1. Estructura general (layouts de página)
2. División de la página (layouts)
3. Contenido de la página (widgets)

User interface

Layouts

Layout para página: fluidPage() / bootstrapPage()

```
ui <- fluidPage(
```

```
  # Título de la aplicación  
  titlePanel("Shiny - fluidPage")
```

```
)
```

```
ui <- bootstrapPage(
```

```
  # Título de la aplicación  
  titlePanel("Shiny - bootstrapPage")
```

```
)
```

```
# Definición del server
```

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

```
# App completa con los componentes ui y server
```

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

Shiny - fluidPage

Shiny - bootstrapPage

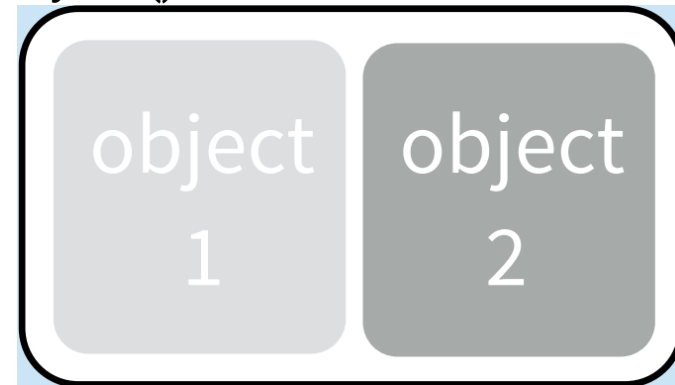
Layouts

(<https://shiny.rstudio.com/reference/shiny/latest/>)

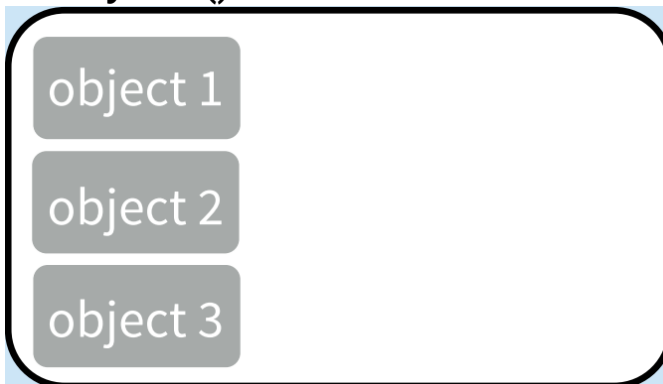
sidebarLayout()



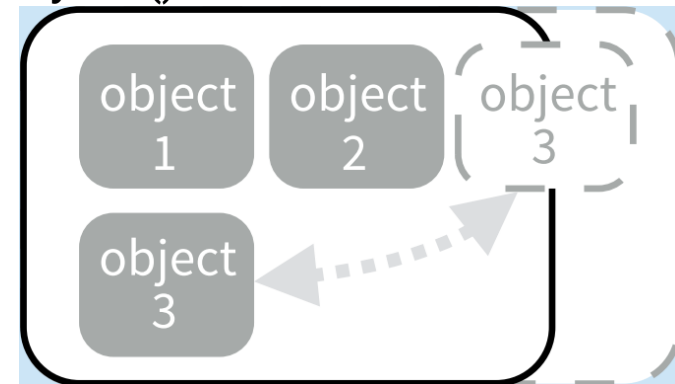
splitLayout()



verticalLayout()



flowLayout()



Layout: Notas

Los elementos de la página se definen **de arriba abajo**.

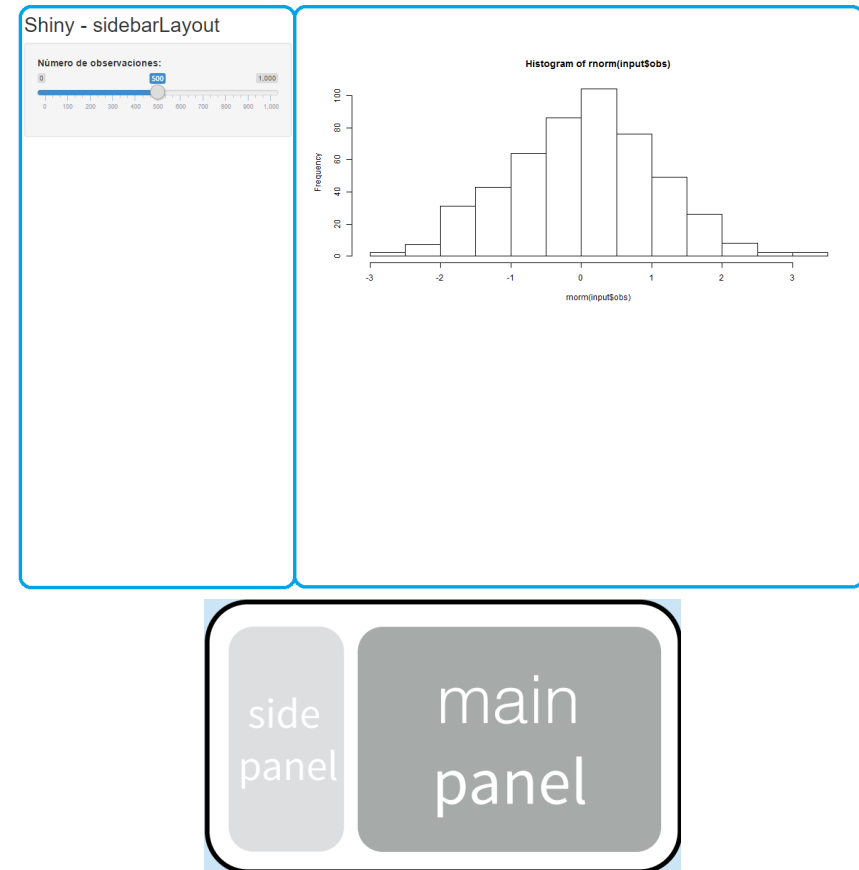
La **separación** entre los elementos es **con comas**.

Gran **flexibilidad** en la configuración del IU.

Los *layouts* se pueden **combinar** unos dentro de otros, **anidados**.

Layout: sidebarLayout()

```
ui <- fluidPage(  
  titlePanel("Shiny - sidebarLayout"),  
  sidebarLayout(  
    # Barra lateral con un selector deslizable  
    sidebarPanel(  
      sliderInput("obs",  
        "Número de observaciones:",  
        min = 0, max = 1000,  
        value = 500)  
    ),  
    mainPanel(  
      plotOutput("distPlot")  
    )  
  )  
)  
  
server <- function(input, output) {  
  output$distPlot <- renderPlot({  
    hist(rnorm(input$obs))  
  })  
}  
  
shinyApp(ui, server)
```



Layout: verticalLayout()

```
ui <- fluidPage(  
  titlePanel("Shiny - verticalLayout"),  
  
  verticalLayout(  
    a(href="http://example.com/link1", "Link Primer  
    a(href="http://example.com/link2", "Link Segunc  
    a(href="http://example.com/link3", "Link Tercer  
  )  
)  
  
server <- function(input, output) {}  
  
shinyApp(ui, server)
```

Shiny - verticalLayout

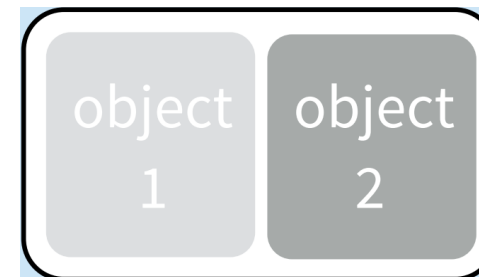
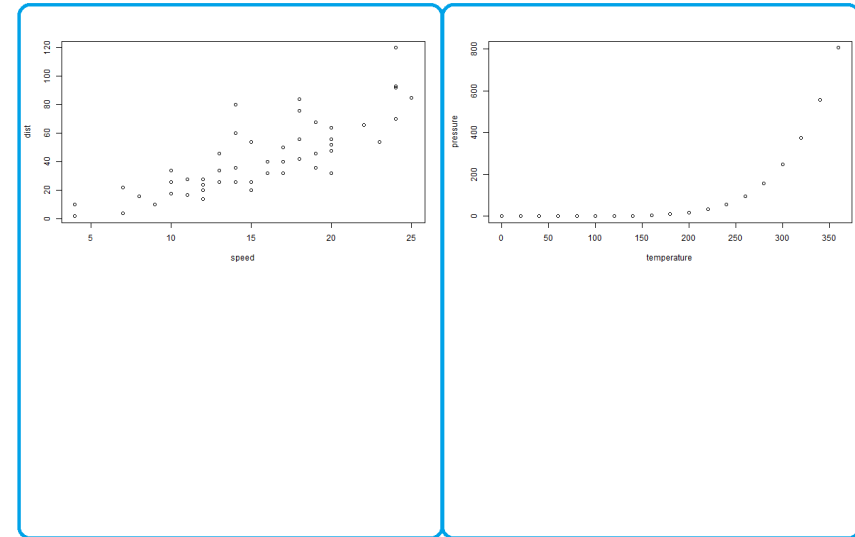
[Link Primero](#)
[Link Segundo](#)
[Link Tercero](#)



Layout: splitLayout()

```
ui <- fluidPage(  
  titlePanel("Shiny - splitLayout"),  
  
  splitLayout(  
    plotOutput("plot1"),  
    plotOutput("plot2")  
  )  
)  
  
server <- function(input, output) {  
  output$plot1 <- renderPlot(plot(cars))  
  output$plot2 <- renderPlot(plot(pressure))  
}  
  
shinyApp(ui, server)
```

Shiny - splitLayout

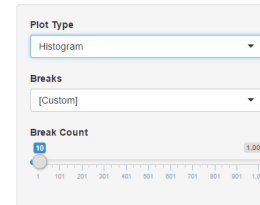


Layout: conditionalPanel()

```
ui <- fluidPage(  
  titlePanel("Shiny - conditionalPanel"),  
  
  sidebarPanel(  
    selectInput("plotType", "Plot Type",  
      c(Scatter = "scatter", Histogram = "hist")),  
  
    conditionalPanel(  
      condition = "input.plotType == 'hist'",  
      selectInput("breaks", "Breaks",  
        c("Sturges", "[Custom]" = "custom")),  
      conditionalPanel(  
        condition = "input.breaks == 'custom'",  
        sliderInput("breakCount", "Break Count", 1,  
          1000)  
      )  
    )  
  )  
)  
  
server <- function(input, output) {}  
  
shinyApp(ui, server)
```

Si condicionamos por una variable definida por el usuario:
- Nos referiremos a la variable como ***input.xxx***, NO usaremos ***input\$xxx***.

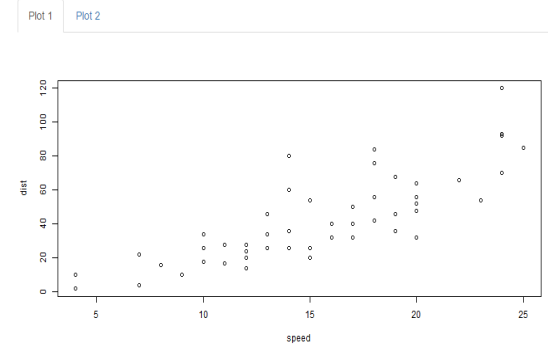
Shiny - conditionalPanel



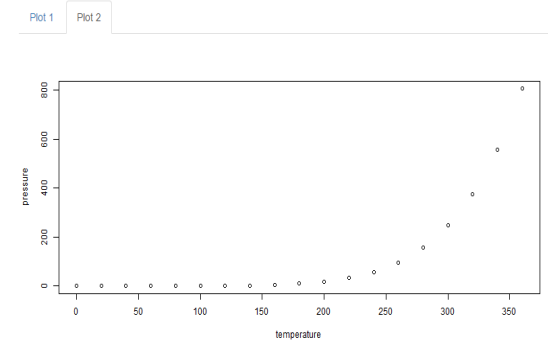
Layer tabPanels: tabsetPanel()

```
ui <- fluidPage(  
  titlePanel("Shiny - tabsetPanel"),  
  
  mainPanel(  
    tabsetPanel(  
      tabPanel("Plot 1", plotOutput("plot1")),  
      tabPanel("Plot 2", plotOutput("plot2"))  
    )  
  )  
)  
  
server <- function(input, output) {  
  output$plot1 <- renderPlot(plot(cars))  
  output$plot2 <- renderPlot(plot(pressure))  
}  
  
shinyApp(ui, server)
```

Shiny - tabsetPanel



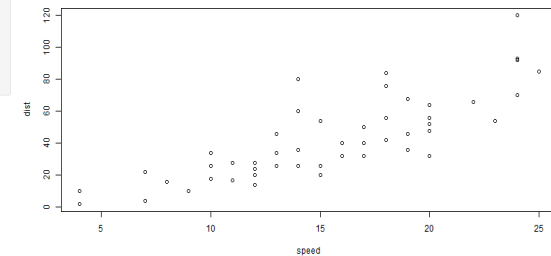
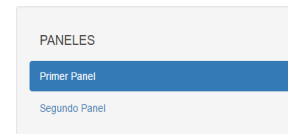
Shiny - tabsetPanel



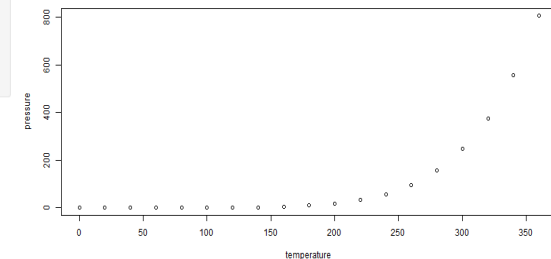
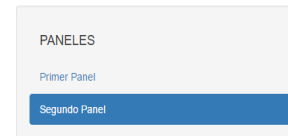
Layer tabPanels: navlistPanel()

```
ui <- fluidPage(  
  titlePanel("Shiny - navlistPanel"),  
  
  navlistPanel("PANELES",  
    tabPanel("Primer Panel",  
      plotOutput("plot1")),  
    tabPanel("Segundo Panel",  
      plotOutput("plot2"))  
  )  
)  
  
server <- function(input, output) {  
  output$plot1 <- renderPlot(plot(cars))  
  output$plot2 <- renderPlot(plot(pressure))  
}  
  
shinyApp(ui, server)
```

Shiny - navlistPanel



Shiny - navlistPanel



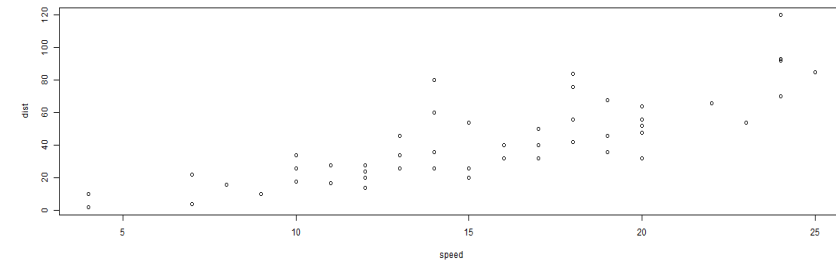
Layer tabPanels: navbarPage()

```
ui <- navbarPage("Shiny - navbarPage",
  tabPanel("Primera Página",
    plotOutput("plot1")),
  tabPanel("Segunda Página",
    plotOutput("plot2"))
)

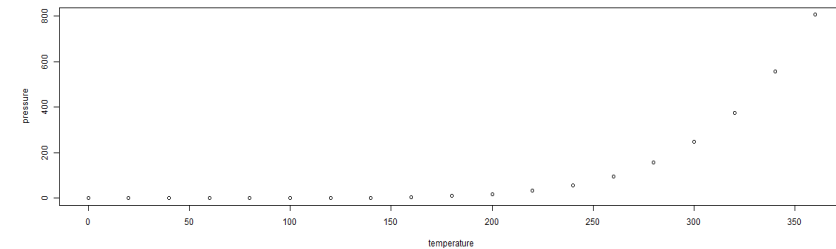
server <- function(input, output) {
  output$plot1 <- renderPlot(plot(cars))
  output$plot2 <- renderPlot(plot(pressure))
}

shinyApp(ui, server)
```

Shiny - navbarPage Primera Página Segunda Página



Shiny - navbarPage Primera Página Segunda Página



User interface

Widgets: Inputs

Widgets: Inputs

Control widgets

Basic widgets

Buttons

Action

Submit

Date range

2014-01-24 to 2014-01-24

Radio buttons

- ☒ Choice 1
- ☐ Choice 2
- ☐ Choice 3

Single checkbox

☒ Choice A

File input

Choose File No file chosen

Select box

Choice 1

Checkbox group

- ☒ Choice 1
- ☐ Choice 2
- ☐ Choice 3

Help text

Note: help text isn't a true widget, but it provides an easy way to add text to accompany other widgets.

Sliders



Date input

2014-01-01

Numeric input

1

Text input

Enter text...

Server

server / server.R

¿Qué hacemos en el server.R?

Al referirnos a variables introducidas por el usuario mediante widgets definidos en el ui, emplearemos ***input\$xxx***, siendo ***xxx*** es el nombre indicado en el argumento ***InputID*** del widget correspondiente.

En el **server** es donde debemos realizaremos los cálculos y gráficos que queramos mostrar en el **ui**.

- Cada acción o grupo de acciones relacionadas, debe ser definida en una función.
- Los elementos del **server** no tienen que seguir un orden de aparición (a diferencia de los del **ui**).

Server

Widgets: Outputs

Widgets: Outputs

Rendered output

To add reactive output to your document, call one of the `render*` functions below in an R code chunk.

`render` function creates

<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

Server

Reactividad

Utilización de expresiones reactivas

- Shiny responde de forma inmediata a los cambios introducidos por el usuario pero, ¿siempre queremos esto?
- En ocasiones nos interesará realizar un cálculo o un gráfico que dependa de más de una variable, ¿que sucedería si permitimos que Shiny nos de una respuesta instantáneamente?

Funciones reactivas

render*() -> Salida por pantalla

- Las funciones **render*()** crean una salida para mostrar por **pantalla**.
- Los **resultados** de estas funciones siempre se guardan en **output\$**.

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

reactive() -> Modularizar código

- Las expresiones **reactive()** generan un objeto para ser utilizado. Este objeto cambiará de valor cada vez que se modifique algún **input\$** de su interior.
- El objeto generado se llama como una función.

```
data <- reactive({  
  rnorm(input$N)  
})
```

```
output$hist <- renderPlot({  
  hist(data())  
})
```

- Las funciones reactive nos permiten una **programación modular**. Esto ayuda a que la ejecución de las apps sea más rápida y fluida

isolate() -> Previene reacciones

- **isolate()** hace que un objeto no sea reactivo.

```
data <- reactive({  
  rnorm(input$N)  
})  
  
output$hist <- renderPlot({  
  hist(data(),  
    main = isolate(input$titulo))  
})
```

observeEvent() -> Activación de código

- Activa el código para que se ejecute en el servidor.

```
ui <- fluidPage(  
  actionButton(inputId = "accion",  
    label = "Acción")  
)  
server <- function(input, output) {  
  observeEvent(input$accion, {  
    print(runif(1, 1, 100))  
  })  
}
```

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

observe()

- Se comporta como el **observeEvent()** pero reaccionando a todos los valores reactivos que contiene.

eventReactive() -> Retrasar reacciones

- Expresión reactiva que solo responden a un valor específico.

```
ui <- fluidPage(  
  numericInput(inputId = "N", label = "Generar número aleatorio entre 0 y:", value = 100, min = 0, max = 300),  
  actionButton(inputId = "actualizar",  
    label = "Actualizar"),  
  textOutput("rnd")  
)  
server <- function(input, output) {  
  data <- eventReactive(input$actualizar, {  
    runif(1, 1, input$N)  
  })  
  
  output$rnd <- renderText({data()})  
}  
  
shinyApp(ui = ui, server = server)
```

- Podemos emplear **eventReactive()** para retrasar la ejecución de determinadas reacciones.

reactiveValues()

- **reactiveValues()** crea una lista de valores reactivos.

-Estos valores reactivos pueden manejarse (usualmente mediante **observeEvent()**)

```
ui <- fluidPage(  
  actionButton(inputId = "norm", label = "Normal"),  
  actionButton(inputId = "unif", label = "Uniforme"),  
  plotOutput("hist")  
)  
server <- function(input, output) {  
  rv <- reactiveValues(data = rnorm(100))  
  observeEvent(input$norm, { rv$data <- rnorm(100) })  
  observeEvent(input$unif, { rv$data <- runif(100) })  
  output$hist <- renderPlot({  
    hist(rv$data)  
  }) }  
shinyApp(ui = ui, server = server)
```

Bibliografía y recursos

Shiny Cheat Sheet (<https://www.rstudio.com/wp-content/uploads/2015/03/shiny-spanish.pdf>)

Tutorial Shiny by RStudio (<https://shiny.rstudio.com/tutorial/lesson1/>)

RStudio Team. 2016. *RStudio: Integrated Development Environment for R*. Boston, MA: RStudio, Inc.
<http://www.rstudio.com/> (<http://www.rstudio.com/>).

Wickham, H. 2015. *Advanced R*. Boca Raton, FL: CRC.

