

# Shiny: Tutorial para principiantes



R Ladies Medellín





Si crees en la revolución de las pequeñas cosas,  
si estás dispuesto a formar parte de un equipo de trabajo,  
si el mundo de la programación te genera interés.

Y si crees en la importancia de que las minorías maximicen su  
potencial...

Entonces R-Ladies está pensado para tí





# R-Ladies y sus capítulos



57439 Miembros

168 Grupos

49 Países

R-Ladies es una organización mundial cuya misión es promover la diversidad de género en la comunidad R

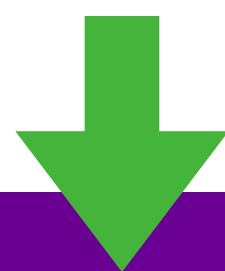
# ¿Dónde encontrar este taller sobre SHINY?

The screenshot shows the GitHub interface for the repository 'RLadiesMedellin / Meetup4-TallerShiny'. The 'Code' tab is selected in the top navigation bar. Below the repository name, there are buttons for 'Go to file', 'Add file', and a green 'Code' button with a download icon. A dropdown menu is open from the 'Code' button, showing options: 'Clone with HTTPS' (with a 'Use SSH' link), 'Open with GitHub Desktop', and 'Download ZIP' (which is highlighted with an orange box). To the left of the dropdown, there is a table of files and folders in the repository.

File/Folder	Commit Message
RLadiesMedellin Create app	
Ejemplo1	Create server.R
Ejemplo2	Create app
PlantillaMF	Create server.R
PlantillaSF	Update and rename app to app.R
README.md	Initial commit

1. Clic en Clone or download

2. Seleccionar casilla Download ZIP



<https://github.com/RLadiesMedellin/Meetup4-shiny>



# ¿Qué es SHINY?

**Shiny** Hoja de referencia  
lee mas en [shiny.rstudio.com](https://shiny.rstudio.com)

Shiny 0.10.0 Actualizado: 6/14



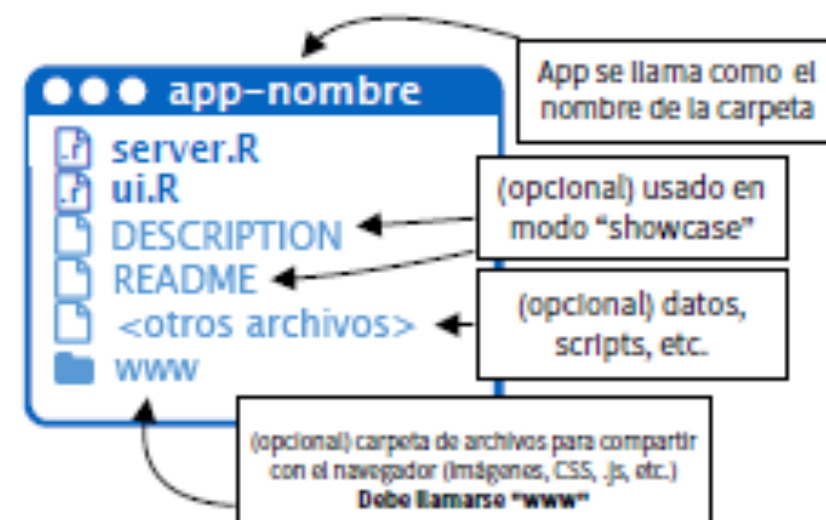
## 2. server.R

Instrucciones que constituyen los componentes R de tu app. Para escribir server.R:

- Provee server.R con el mínimo de código necesario, `shinyServer(function(input, output) {})`.
- Define los componentes en R para tu app entre las llaves `{ }` después de `function(input, output)`.
- Guarda cada componente R destinados para tu interfaz (UI) como `output$<nombre componente>`.
- Crea cada componente de salida con una función `render*`.
- Dale a cada función `render*` el código R que el servidor necesita para construir el componente. El servidor notará valores reactivos que aparecen en el código y reconstruirá el componente cada vez que estos valores cambian.
- Has referencia a valores en "widgets" con `input$<nombre del widget>`.

## 1. Estructura

Cada app es una carpeta que contiene un archivo `server.R` y comúnmente un archivo `ui.R` (opcionalmente contiene archivos extra)



## server.R

```
# carga paquetes, scripts, datos

A shinyServer(function(input, output) {B

  # crea variables especificas para usuario

  output$texto <- renderText({
    input$titulo
  })

  C output$grafica <- D renderPlot({
    x <- mtcars[, input$x]E
    y <- mtcars[, input$y]
    plot(x, y, pch = 16)
  })

  })
```

## funciones render\*

function	espera	crea
<code>renderDataTable</code>	objetos como tablas	tabla DataTables.js
<code>renderImage</code>	lista atributos imágenes	imagen HTML
<code>renderPlot</code>	gráfica	gráfica
<code>renderPrint</code>	salida impresa	texto
<code>renderTable</code>	objetos como tablas	tabla simple
<code>renderText</code>	cadena de caracteres	texto
<code>renderUI</code>	objeto "tag" o HTML	elemento UI (HTML)

valores de entrada (Input) son reactivos.  
Deben estar rodeados por uno de:

**render\*** - crea un componente shiny UI (Interfaz)  
**reactive** - crea una expresión reactiva  
**observe** - crea un observador reactivo  
**isolate** - crea una copia no-reactiva de un objeto reactivo

## 3. Ejecución

Coloca código en el lugar donde correrá la menor cantidad de veces

- Corre una vez** - código puesto fuera de `shinyServer` solo corre una vez cuando inicias tu app. Úsalo para instrucciones generales. Crea una sola copia en memoria.
- Corre una vez por usuario** - código puesto dentro de `shinyServer` corre una vez por cada usuario que visita tu app (o refresca su navegador). Úsalo para instrucciones que necesitas dar por cada usuario del app. Crea una copia por cada usuario.
- Corre a menudo** - código puesto dentro de una función `render*`, `reactive`, o `observe` correrá muchas veces. Úsalo solo para código que el servidor necesita para reconstruir un componente UI después de que un widget cambia.

## 4. Reactividad

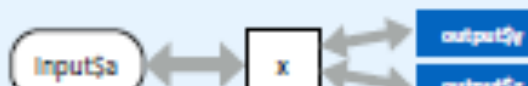
Cuando una entrada (input) cambia, el servidor reconstruye cada salida (output) que depende de ella (también si la dependencia es indirecta). Puedes controlar este comportamiento a través de la cadena de dependencias.

**render\*** - Una salida se actualiza automáticamente cuando una entrada en su función `render*` cambia.



```
output$z <- renderText({
  input$a
})
```

**reactive** - usa `reactive` para crear objetos que se usaran en múltiples salidas.



```
x <- reactive({
  input$a
})
output$y <- renderText({
  x()
})
output$z <- renderText({
  x()
})
```

**isolate** - usa `isolate` para usar una entrada sin dependencia. Shiny no reconstruirá la salida cuando una entrada aislada cambia



```
output$z <- renderText({
  paste(
    isolate(input$a),
    input$b
  )
})
```

**observe** - usa `observe` para crear código que corre cuando una entrada cambia, pero que no crea un objeto de salida.



```
observe({
  input$a
  # código para correr
})
```

RStudio® and Shiny® son marcas registradas de RStudio, Inc.  
© 2017 RStudio <https://rstudio.com>  
844-448-1212 [info@rstudio.com](mailto:info@rstudio.com)  
Traducido por Frans van Gennip - [innovations.nl](https://innovations.nl)

Es un paquete de R para la construcción de cuadro de mando web interactivos.

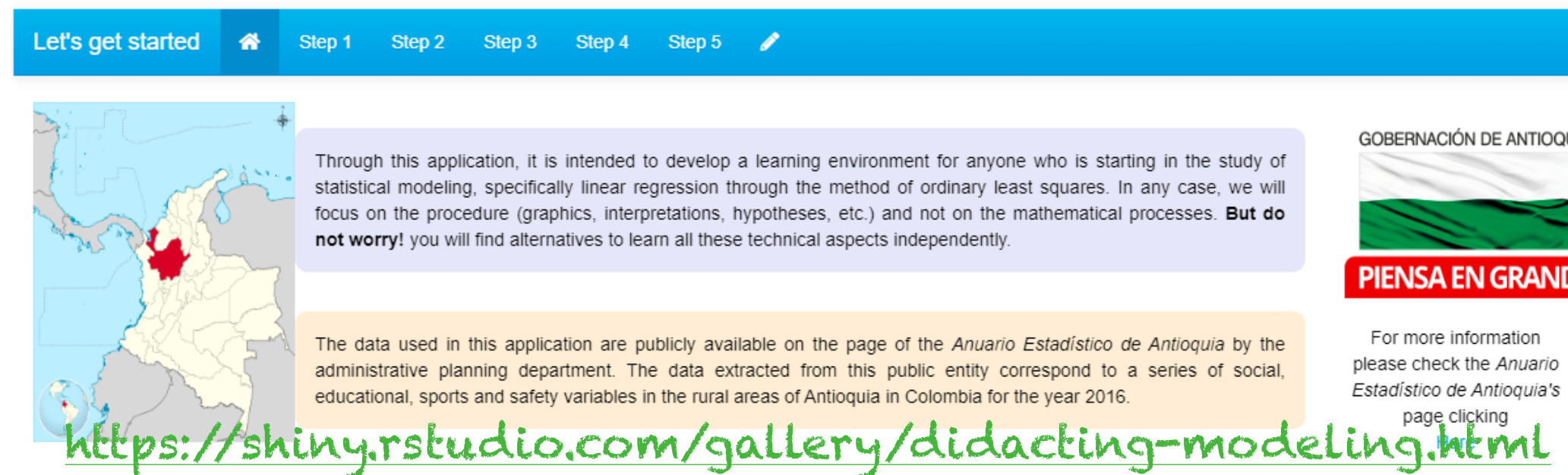
Permite, por ejemplo crear interfaces para algoritmos o acceder y manipular tablas de datos a través de controles de HTML, botones entre otros.



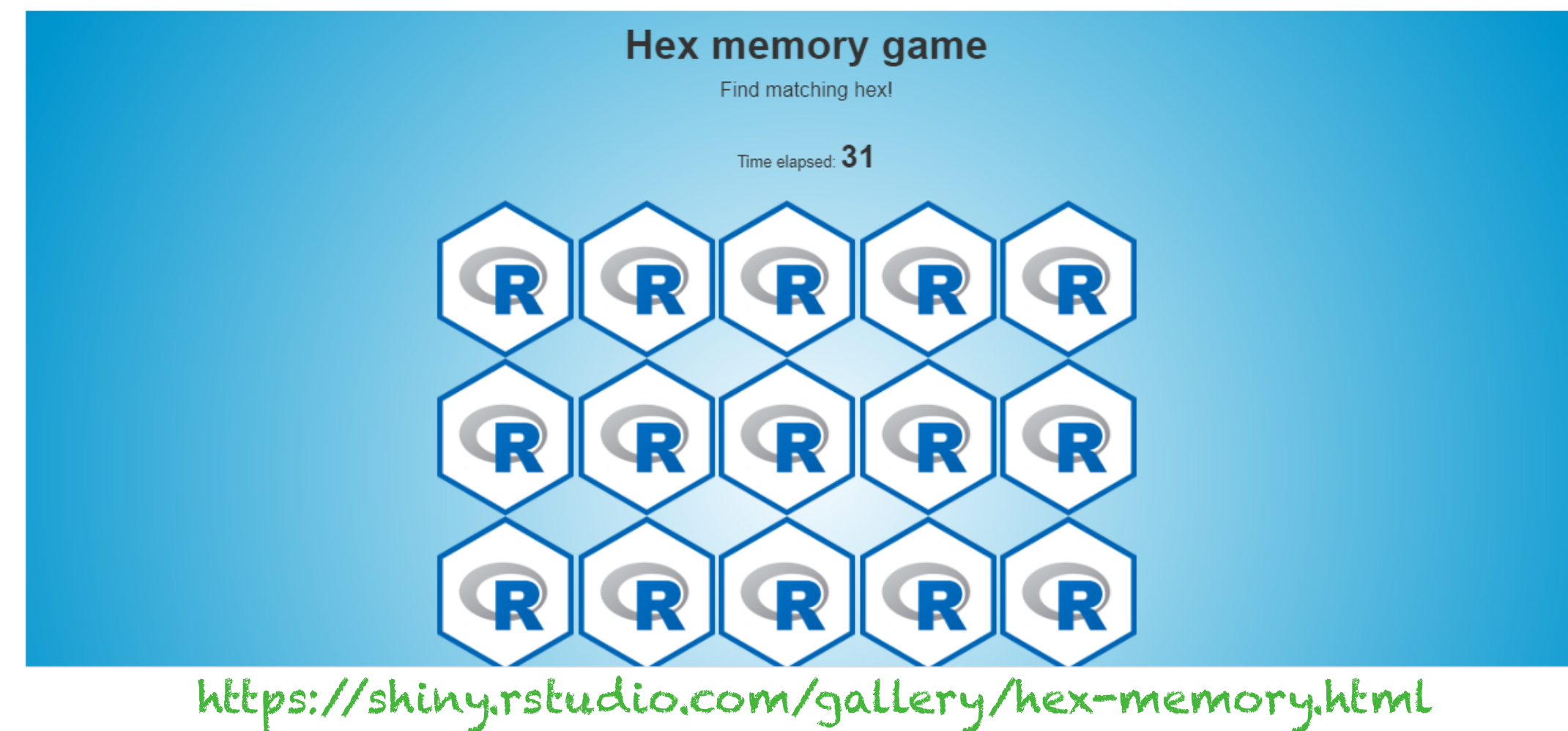
# ¿Qué se puede hacer con SHINY?

## Educación

Didactic modeling process: Linear regression for a safety issue in rural areas of Antioquia - Colombia Oscar Daniel Rivera Baena



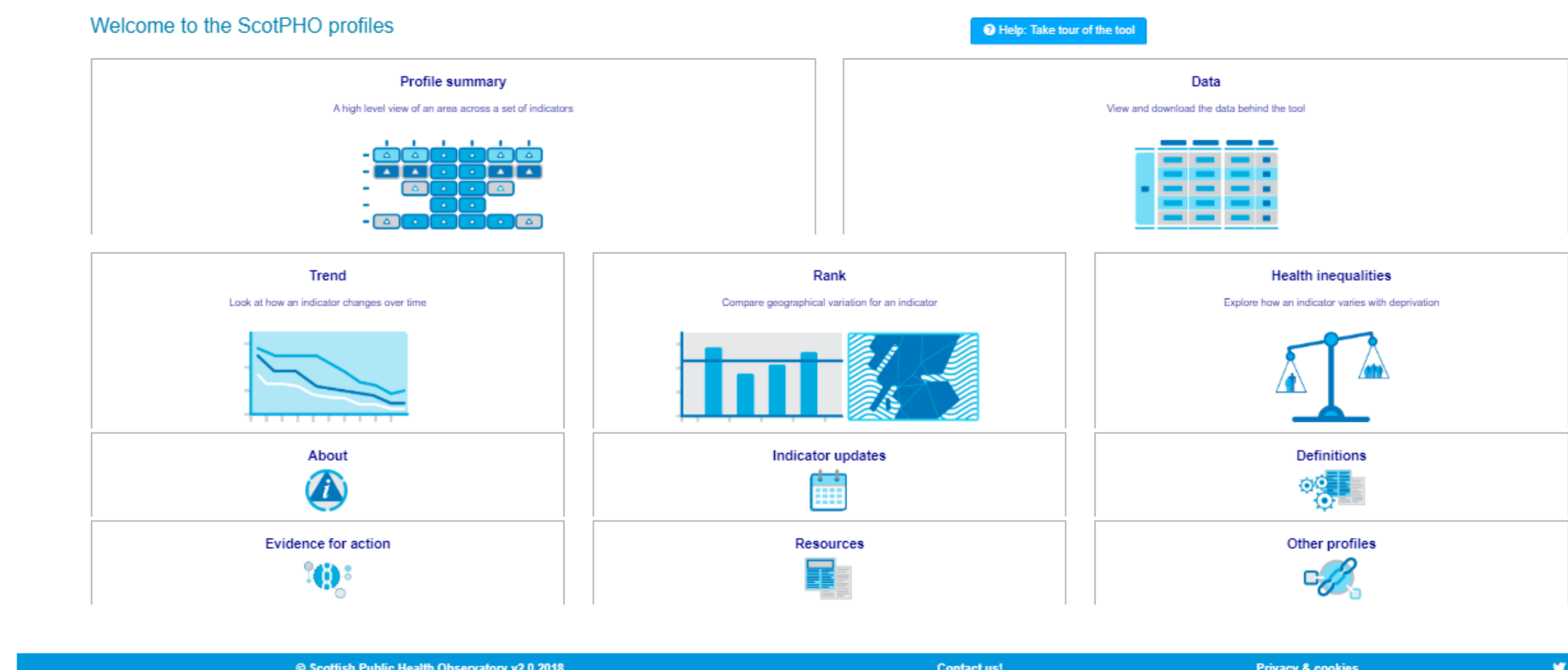
## Juegos



## Problemas Actuales, Mapas



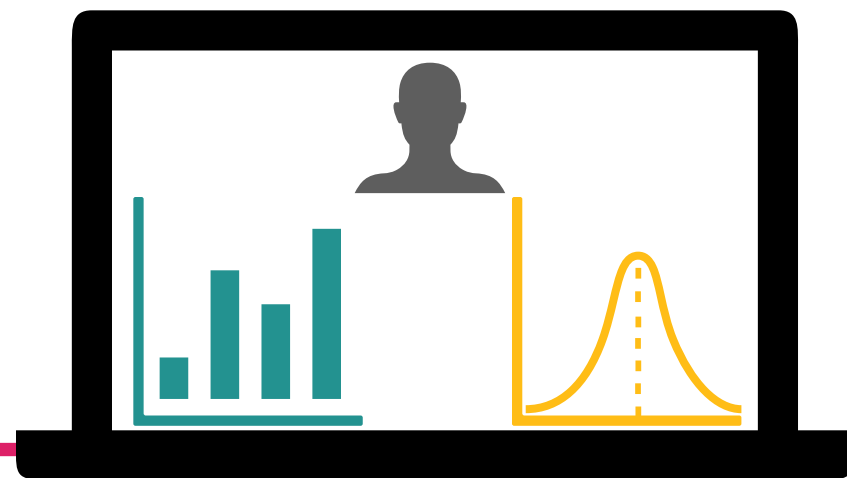
## Interactividad



# ESTRUCTURA BÁSICA

## Interfaz del usuario (UI)

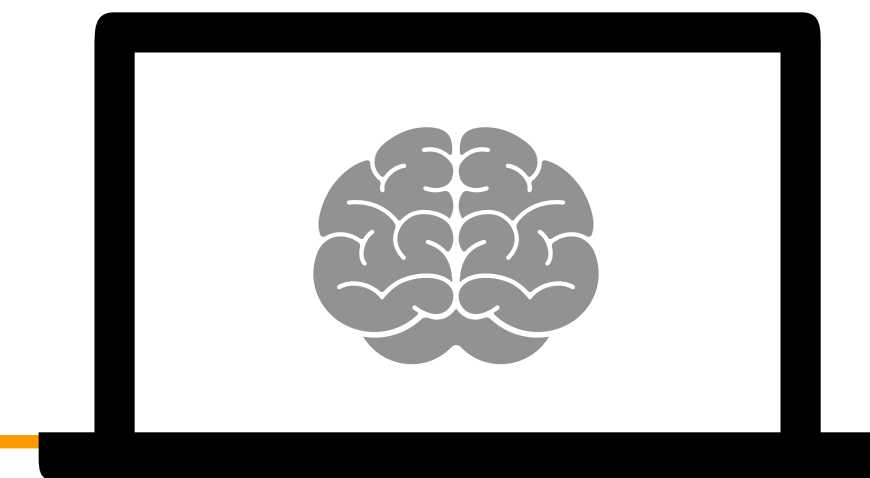
Aquí se crean los objetos que el usuario verá y con los cuales **interactuará**.



+

## Server

Aquí se construyen las salidas que "reaccionan" y se actualizan con base en las entradas que el usuario proporcione.



UI + SERVER = APP



# INTERFAZ DEL USUARIO (UI)

Diseño • Tema • Widgets

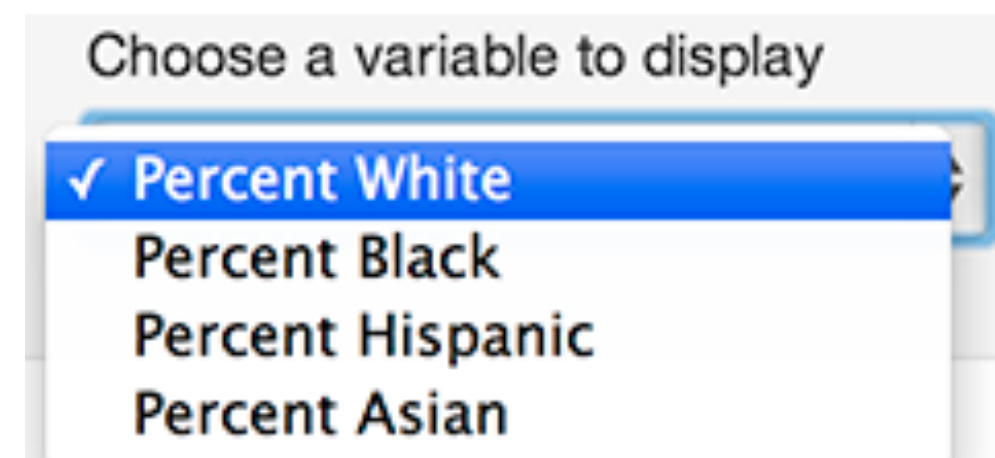
## Widgets:

Recolectan información del usuario (inputs) que se usará para actualizar los outputs creados en el Server.

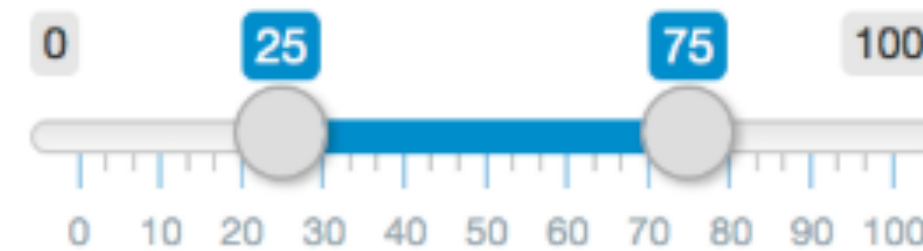
Radio buttons/  
checkboxes

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

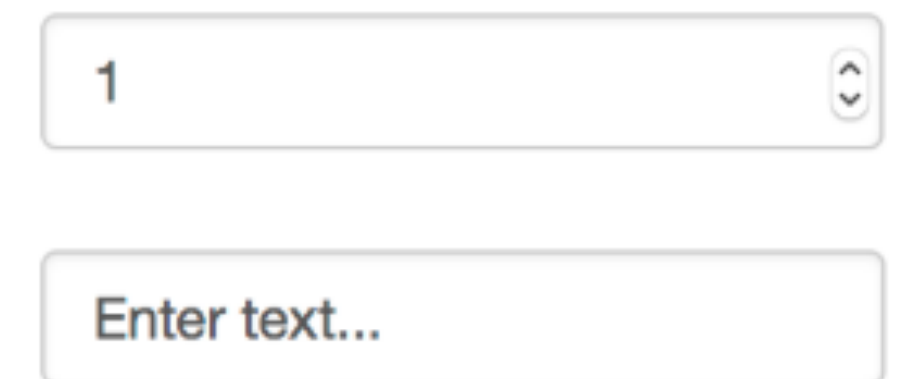
Dropdown  
selection



Sliders



Numeric and  
text inputs

A screenshot showing two input widgets. The top one is a numeric input field containing the number "1". The bottom one is a text input field with the placeholder text "Enter text...".

Y muchos más...

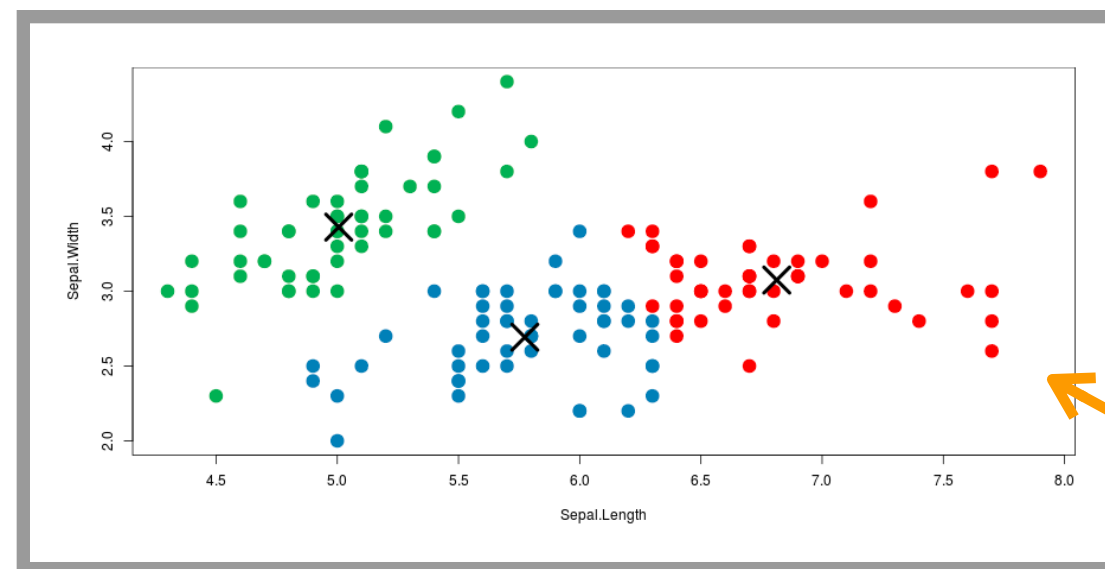


# ¿CÓMO FUNCIONA LA REACTIVIDAD?

En la interfaz del usuario (UI):

**Selector:**

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



Esos outputs "reactivos" se ponen en UI

Los widgets obtienen información del usuario

Actualizan la tabla de datos utilizada para crear outputs basados en las selecciones de los usuarios

Se crean outputs actualizados de la nueva tabla de datos

En el Server

# ¿CÓMO FUNCIONA LA REACTIVIDAD?

1)

EN UI:

Haz un widget que tenga un nombre (inputID)



2)

EN SERVER:

Utiliza ese **inputID** para crear un conjunto de datos "reactivo"



3)

EN SERVER:

Haz un **output** que tenga un **nombre** (ej: tabla2) usando el conjunto de datos "reactivo"



4)

EN UI:

Llama o **invoca** el **output** reactivo para hacerlo aparecer



# ANATOMÍA DE UNA SHINY APP

```
library(shiny)
```

```
ui <- fluidPage()
```

## Interfaz del usuario (UI)

Controla el diseño y la apariencia de la app

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

## Server

Contiene las instrucciones necesarias para construir la app

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

## UI + Server - shinyApp()

Crea el objeto Shiny App

# REGLAS PARA FUNCIONES DEL SERVER

- Guarda los objetos a mostrar como `output$xx`
- Construye objetos a mostrar con `render*()`
- Utiliza valores input con `input$xx`



# REGLAS PARA FUNCIONES DEL SERVER

```
ui <- fluidPage(  
  . . .  
  # Output: Show scatterplot  
  mainPanel(  
    plotOutput(outputId = "scatterplot")  
  . . .  
)
```

```
server <- function(input, output) {  
  
  # Create the scatterplot object the plotOutput function is expecting  
  output$scatterplot <- renderPlot({  
    ggplot(data = movies, aes_string(x = input$x, y = input$y)) +  
      geom_point()  
  })  
}
```

# TIPS

- Corre siempre TODO el script, no solamente la parte en la que estás codificando.
- La indentación permite una mayor organización y entendimiento del código.
- ¡Ten cuidado con las comas y los paréntesis!