

TALLER SHINY

Diplomado Data Science
Felipe Peña Graf





SHINY



SHINY

- Paquete de R que facilita la construcción de aplicaciones web interactivas escritas directamente en R.
- Se pueden mantener como aplicaciones por si mismas o embebidas en documentos o en dashboards
- Su funcionamiento se puede extender con temas de CSS (shinythemes), HTML (htmlwidgets) y acciones de JavaScript (shinyjs).



USOS

Educación

Finanzas

Sector
público

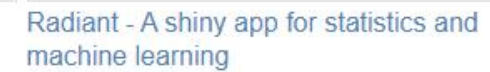
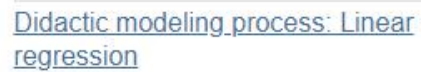
Ciencias
sociales

Deportes

Tecnología

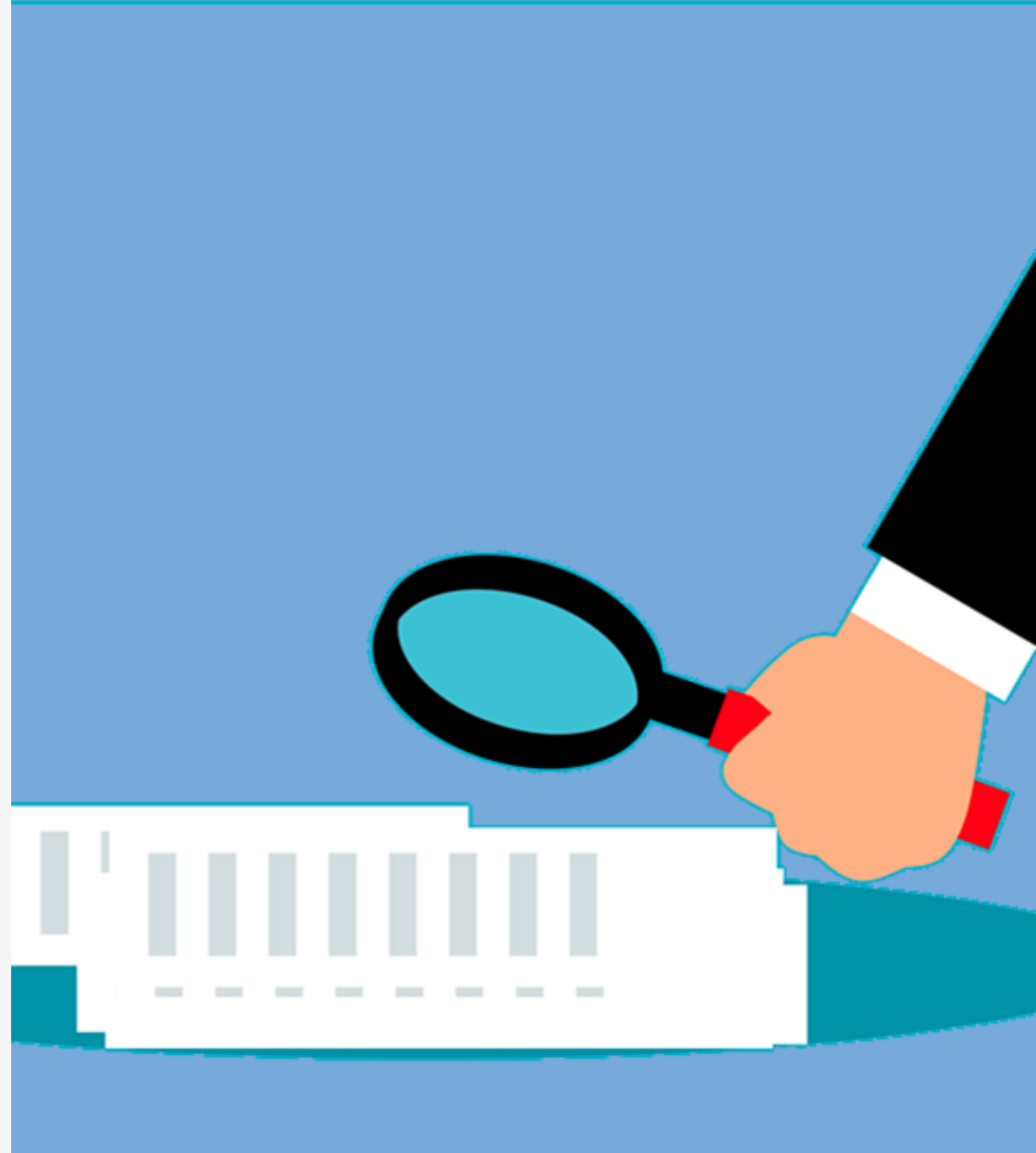
Misceláneos,
juegos.

Apps designed for teaching



Shiny Showcase

ASSESSMENT



PREREQUISITOS

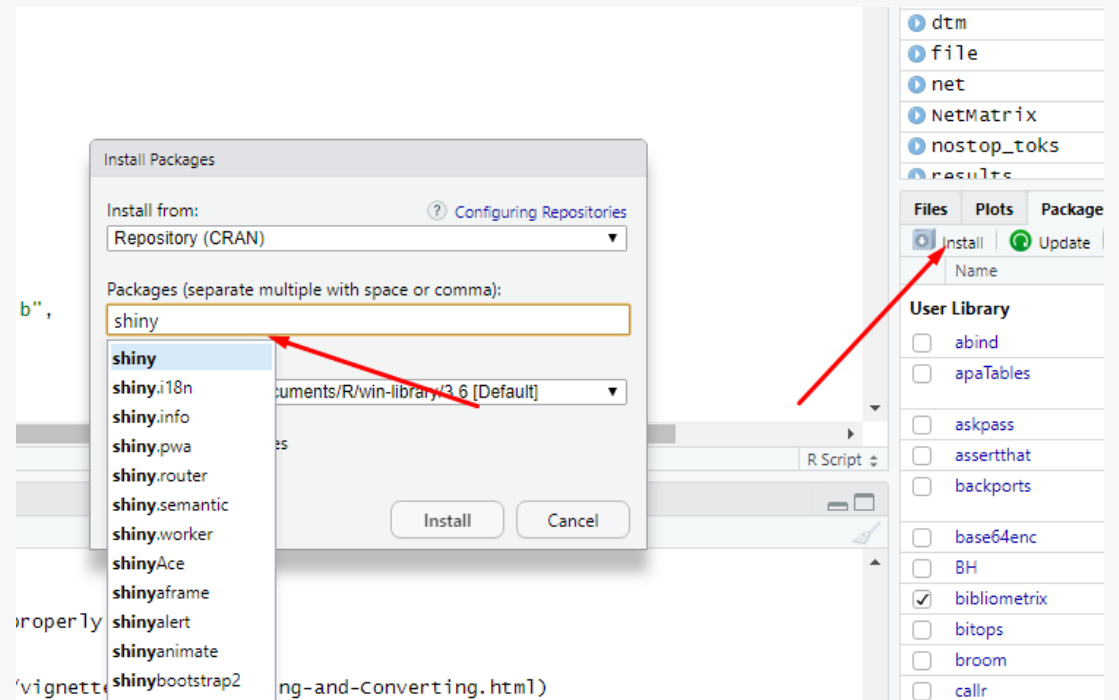
- Saber R
- Saber lo mínimo de cómo funciona internet y las aplicaciones web
- Shiny solamente es una forma de hacer que personas que no saben R puedan interactuar con su sistema

INSTALACIÓN

A través de la consola con:

`install.packages("shiny")`

Desde R studio en paquetes:

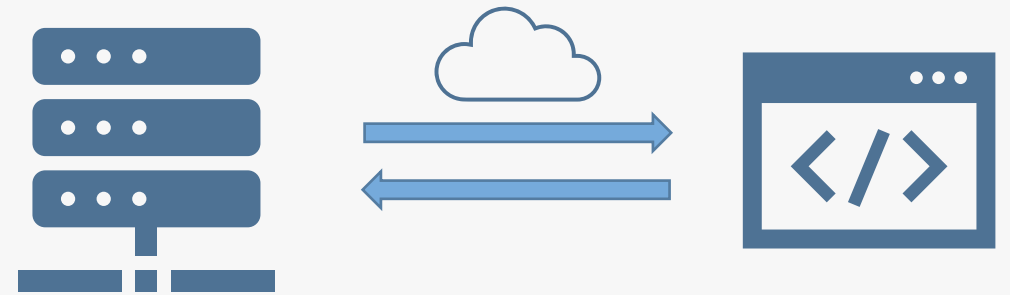


ARQUITECTURA

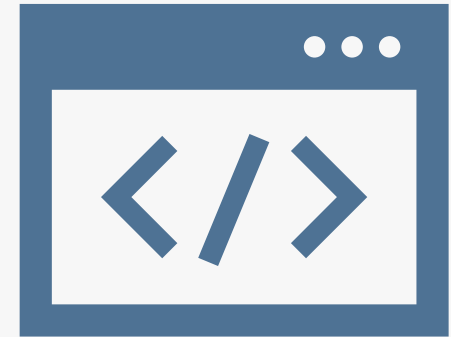
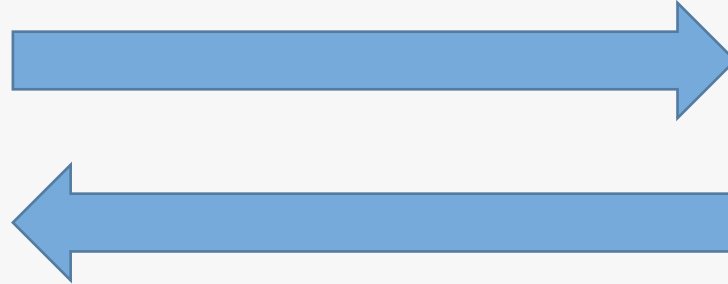
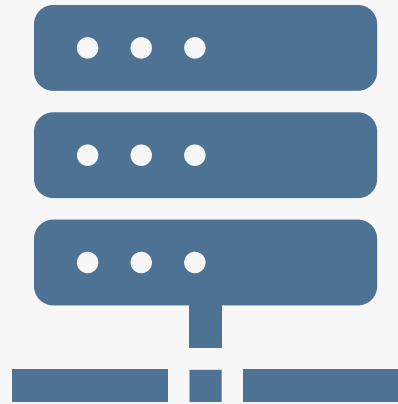
LOCAL



NUBE



ARQUITECTURA



ESTRUCTURA - TEMPLATE

- Objeto interfaz de usuario
- Objeto de información del servidor
- Llamada al paquete

```
library(shiny)
ui<- fluidPage()

server <- function(input, output) {}

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

HELLO WORLD

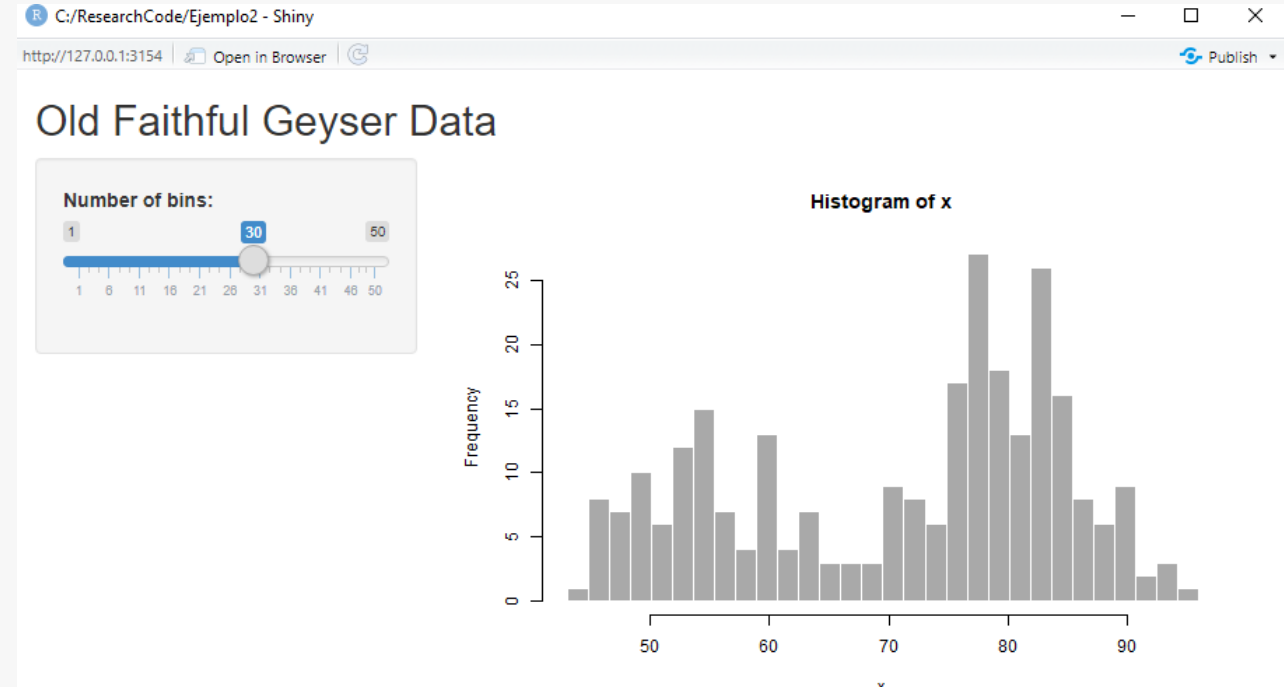
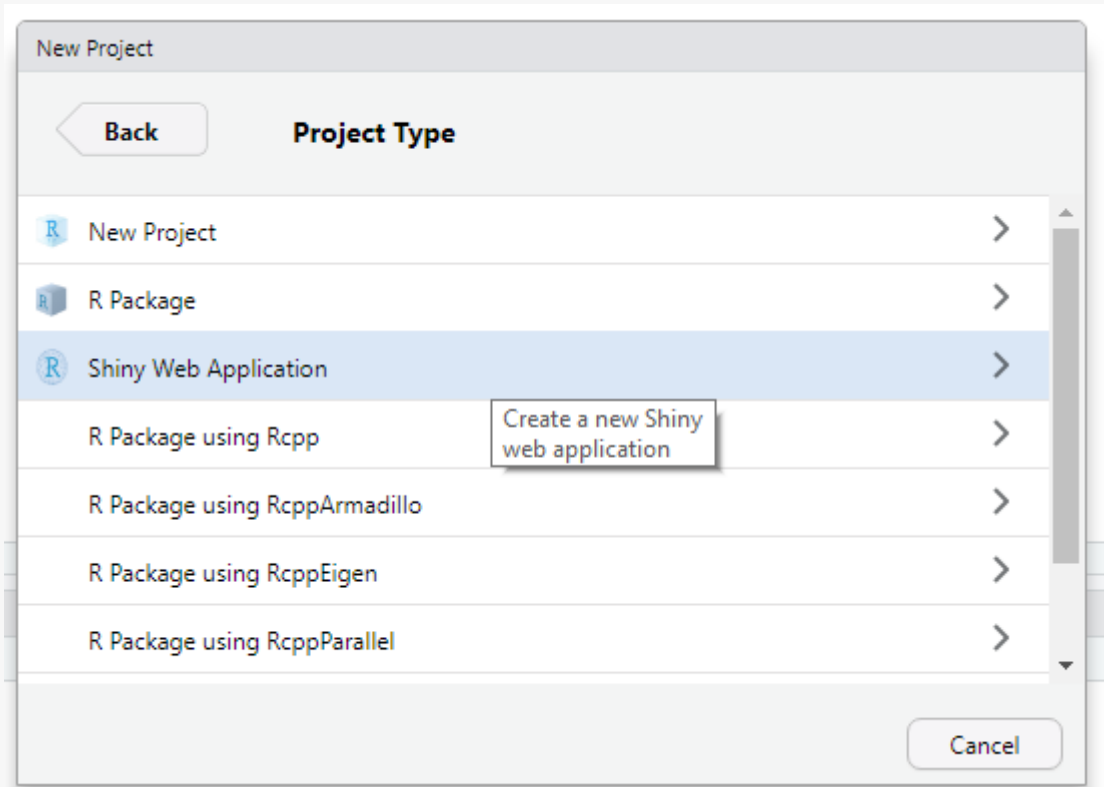
```
library(shiny)
ui<- fluidPage("Hola Mundo")

server <- function(input, output) {}

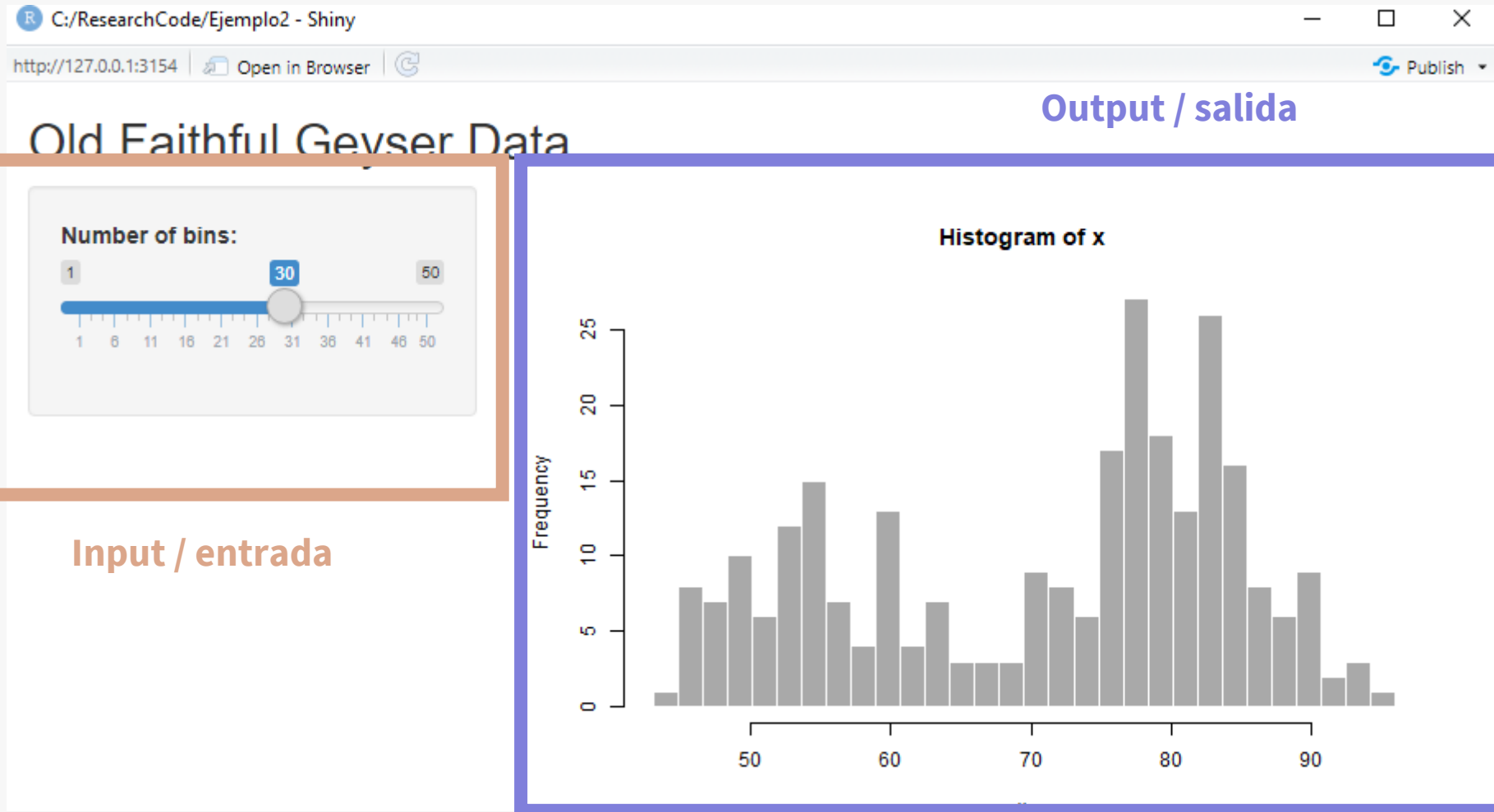
shinyApp(ui=ui, server=server)
```

FORMA MÁS SENCILLA

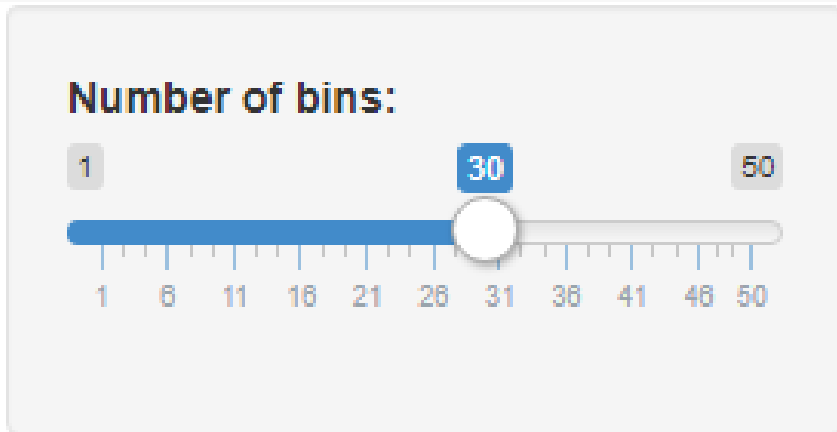
- Crear un nuevo proyecto:



DISEÑANDO LA APLICACIÓN



INPUT



```
sliderInput(inputId = "num",  
label="Number of bins:",  
value = 30, min = 1, max = 50)
```



INPUT

http://127.0.0.1:3771

Open in Browser



Publish

Basic widgets

Buttons

Action

Submit

Date range

2017-06-21

to

2017-06-21

Radio buttons

☒ Choice 1

☐ Choice 2

☐ Choice 3

Single checkbox

☒ Choice A

File input

Browse...

No file selected

Select box

Choice 1

Checkbox group

☒ Choice 1

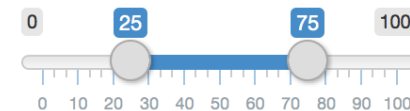
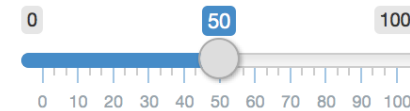
☐ Choice 2

☐ 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

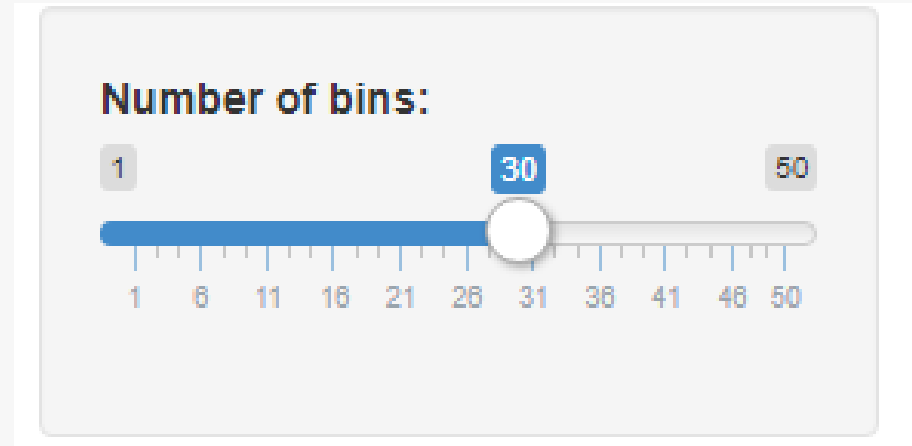
Text input

Enter text...

INPUT

Función	Widget
<code>actionButton</code>	Un botón general de acción
<code>checkboxGroupInput</code>	Un grupo de cajas de opciones para marcar
<code>checkboxInput</code>	Un checkbox simple
<code>dateInput</code>	Campo de fecha con calendario
<code>dateRangeInput</code>	Un par de calendarios para generar rango
<code>fileInput</code>	Activa la ayuda para carga de archivos
<code>helpText</code>	Texto de ayuda para los inputs
<code>numericInput</code>	Un campo de números
<code>radioButtons</code>	Botones de selección radial
<code>selectInput</code>	Una caja de opciones
<code>sliderInput</code>	UN selector de opciones deslizantes
<code>submitButton</code>	Un botón para enviar
<code>textInput</code>	Un campo para introducir texto

INPUT: SINTAXIS



```
sliderInput(inputId = "num", label = "Number of bins:", ...)
```

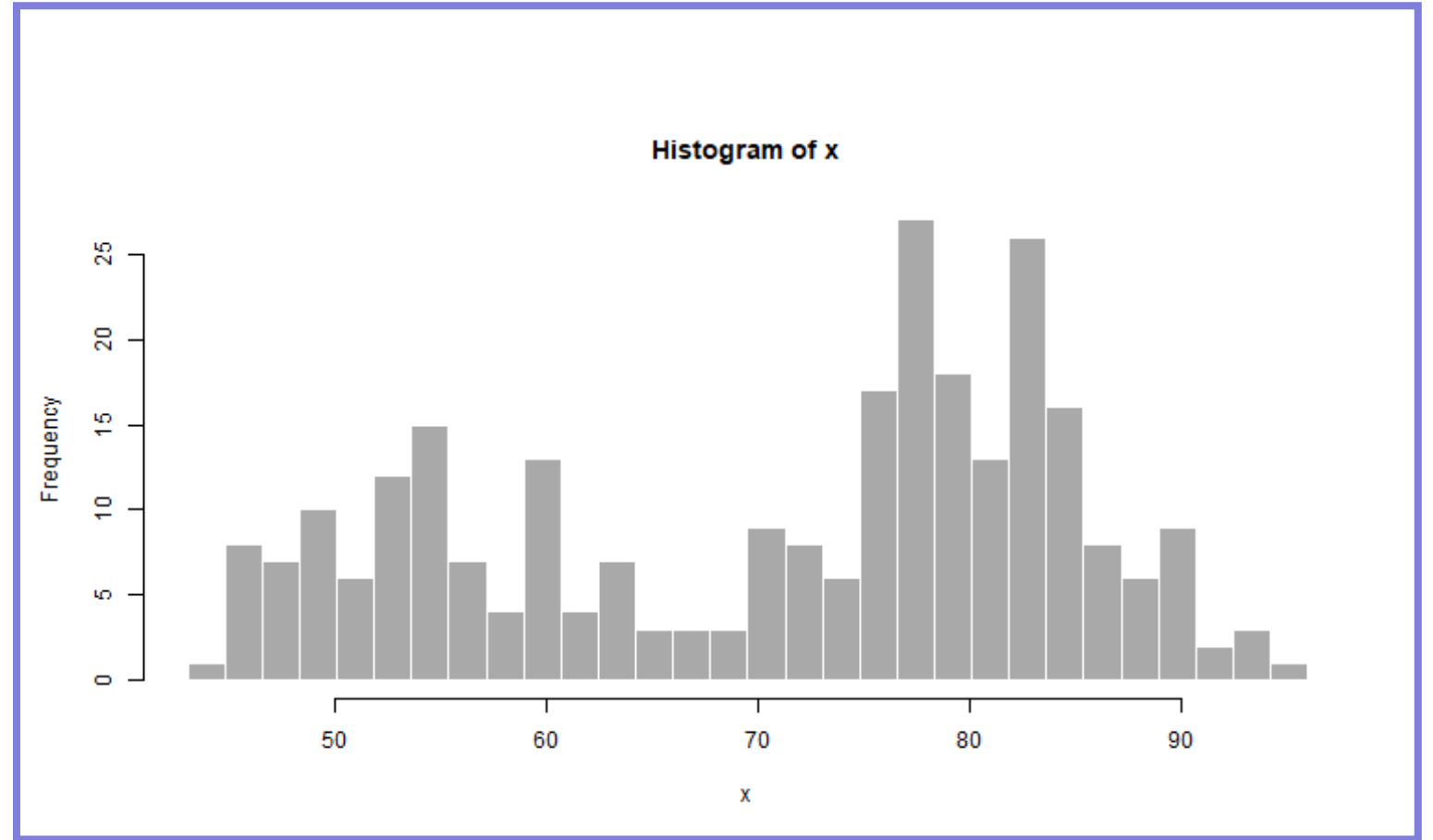
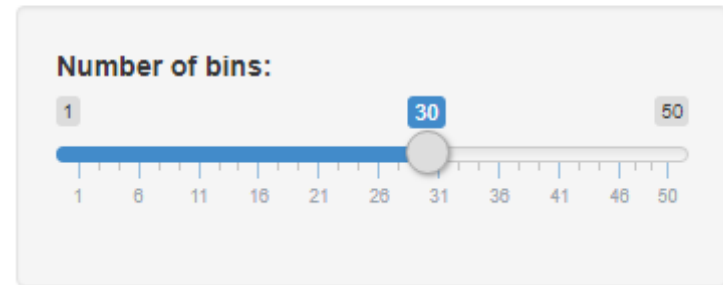
Identificador

Etiqueta

Parámetros
adicionales

OUTPUT

Old Faithful Geyser Data



OUTPUT

Función de salida	Crea
dataTableOutput	DataTable
htmlOutput	HTML puro
imageOutput	Imagen
plotOutput	Gráfico
tableOutput	Tabla
textOutput	Texto
uiOutput	HTML puro
verbatimTextOutput	Texto

El resultado del código/programa. La salida visible de lo realizado por la aplicación

OUTPUT: SINTAXIS

Para mostrar una salida, se debe agregar a la `fluidPage()` una función de output.

```
plotOutput(outputId = "hist")
```



SERVER

- Gestiona y convierte las entradas en las salidas
- Escribiremos todas las instrucciones para las entradas y salidas
- La documentación oficial define las tres reglas principales para trabajar con el servidor

```
server <- function(input, output) {  
  #El código debe ir aquí  
}
```

SERVER: REGLA 1

GUARDA LOS OBJETOS A MOSTRAR EN **output\$**

```
server <- función(input, output) {  
  output$hist <- #código  
}
```

output\$hist

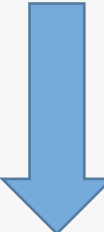


plotOutput("hist")

SERVER: REGLA 2

CONSTRUYE LOS OBJETOS A MOSTRAR CON `render*()`

```
server <- función(input, output) {  
  output$hist <- renderPlot({  
  })  
}
```

output\$hist

plotOutput("hist")

RENDER

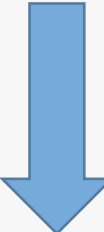
Función de salida	Crea
<code>renderDataTable()</code>	Tabla interactiva (de fuentes diversas)
<code>renderImage()</code>	Una imagen (enlace a archivo)
<code>renderPlot()</code>	Un gráfico
<code>renderPrint()</code>	Un bloque de código que imprime salida
<code>renderTable</code>	Una tabla común
<code>renderText</code>	Texto
<code>renderUI</code>	Un elemento Shiny UI (Html)

Las funciones de **render*()** nos permiten crear el tipo de salida que queremos de nuestro programa

SERVER: REGLA 2

CONSTRUYE LOS OBJETOS A MOSTRAR CON `render*()`

```
server <- función(input, output) {  
  output$hist <- renderPlot({  
    title <- "100 normal"  
    hist(rnorm(100),  
      main = title)  
  })  
}
```

output\$hist

plotOutput("hist")

RENDER: SINTAXIS

Crea una salida reactiva que se puede mostrar en la interfaz de usuario (UI)

```
renderPlot({hist(rnorm(100))})
```



Tipo de Objeto
a construir



Bloque de
Código

SERVER: REGLA 3

USA LOS VALORES DE ENTRADA CON `input$`

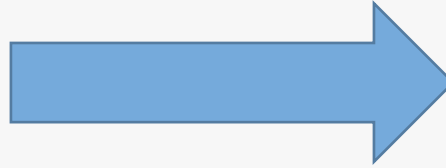
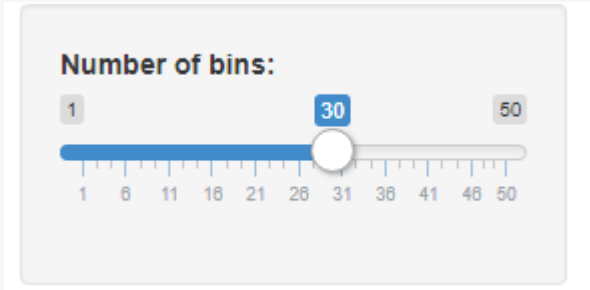
```
server <- función(input, output) {  
  output$hist <- renderPlot({  
    hist(rnorm(input$num))  
  })  
}
```

`input$num`

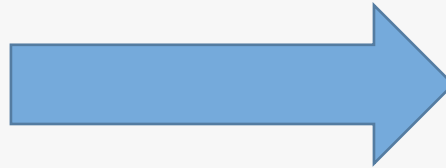
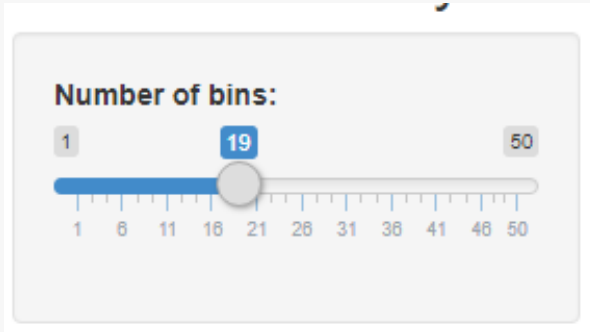


```
renderPlot({  
  hist(rnorm(input$num))  
})
```

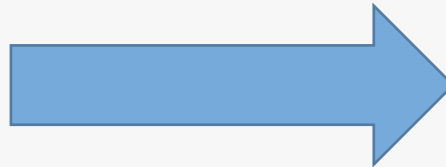
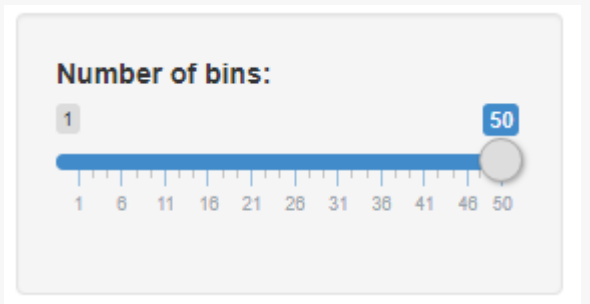
VALORS DE LOS INPUT



input\$bins = 30



input\$bins = 19



input\$bins = 50

REACTIVIDAD

- Inicialmente, la reactividad ocurre automáticamente cuando usamos un input para mostrar un objeto de salida.

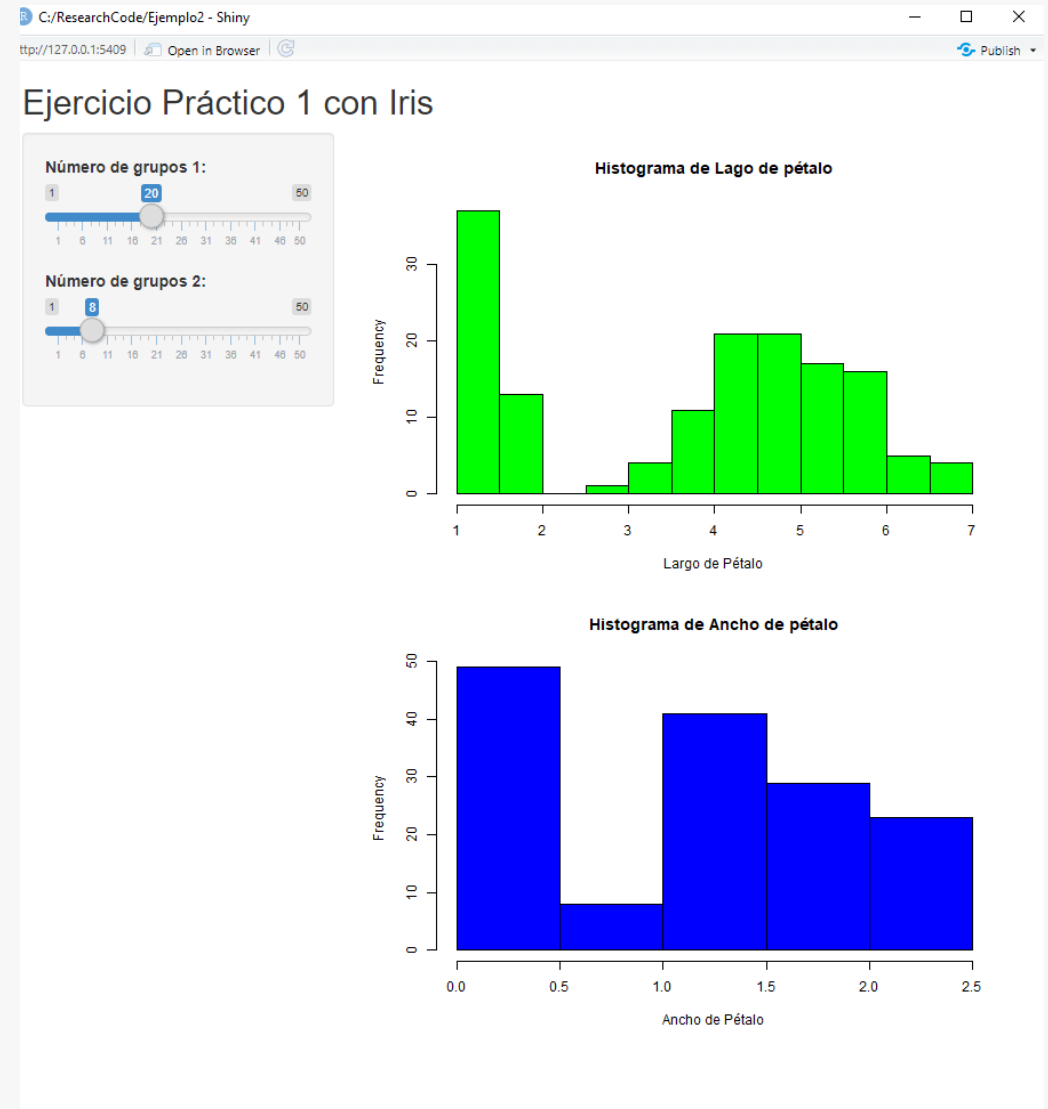
```
function(input, output) {  
  output$hist <- renderPlot({  
    hist(rnorm(input$num))  
  })  
}
```

RESUMIENDO

- Usamos la función `server` para unir las entradas y las salidas con 3 reglas:
- Guardamos la salida de nuestra aplicación a **`output$`**
- Construimos el output con una función **`render*()`**
- Ocupamos los valores de entrada con **`input$`**

EJERCICIO PRÁCTICO 1

- Usando el dataset Iris, cree dos histogramas con la distribución del las variables largo y ancho de pétalo.
- Cree dos SliderInput que controlen la cantidad de separaciones del histograma
- Los histogramas deben ser de diferentes colores y deben estar debidamente etiquetados.
- Use imagen como referencia



EJERCICIO PRÁCTICO 1: TIPS

- `library(datasets)` le permite importar el dataset Iris
- `data(iris)` crea el dataframe Iris que contiene la variable
- Considere la documentación del histograma de R para controlar el diseño, colores, etiquetas y títulos:

<https://www.rdocumentation.org/packages/graphics/versions/3.6.2/topics/hist>

TALLER SHINY

Diplomado Data Science
Felipe Peña Graf

