Universidad Juárez Autónoma de Tabasco

División Académica de Informática y Sistemas

Taller:

"Desarrollo de Aplicaciones Interactivas con Shiny R"

Que presenta:

Msc. Santiago Arias García

(Doctorante 5to semestre
en Ciencias de la Computación)

Directores:

Dr. José Hernández Torruco

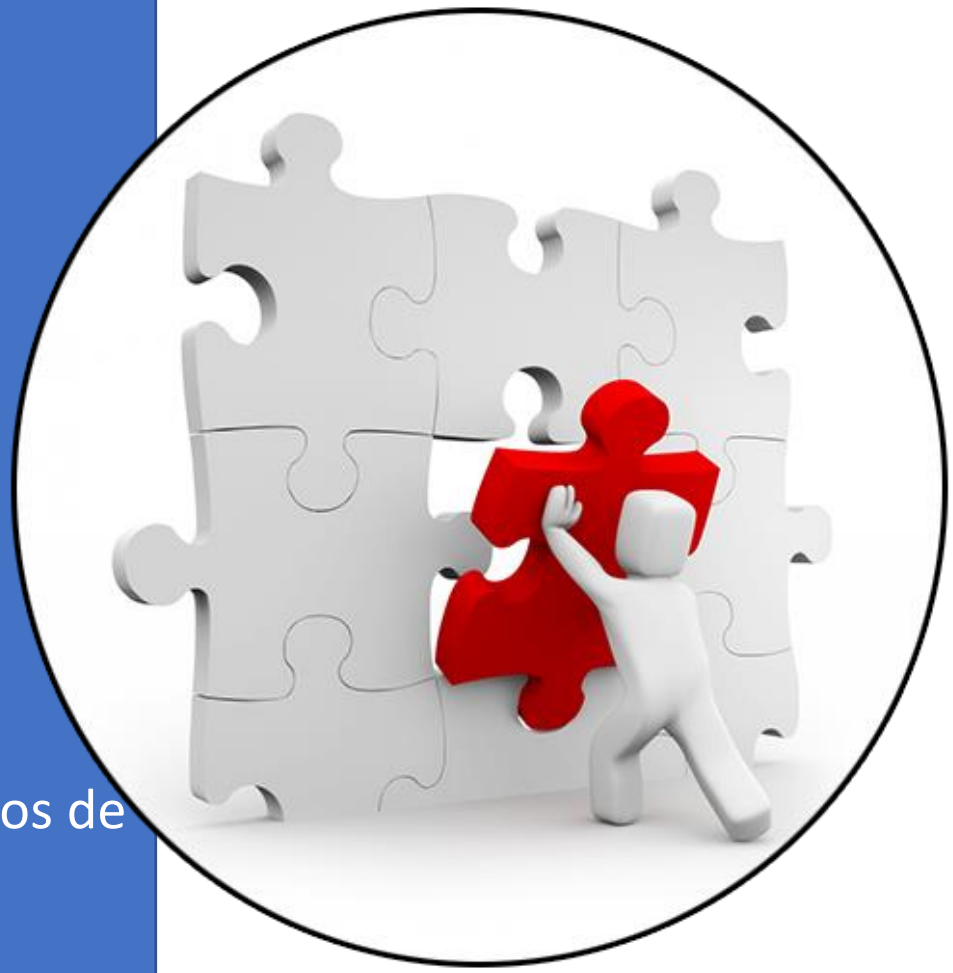
Dra. Betania Hernández Ocaña

Líneas de generación y aplicación del conocimiento:

Ciencias de Datos e Inteligencia Artificial

Abril 2025

Índice



Objetivos del Taller Shiny

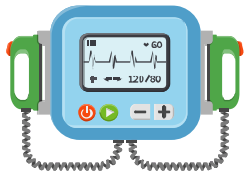
- Taller Shiny
  - Objetivo general
    - Arritmia cardiaca
    - Desarrollar aplicaciones web interactivas con R
    - Manipulación de datos
    - Visualización de datos
    - Modelado de datos
    - Reducción de dimensionalidad
    - Clasificación de datos con IA

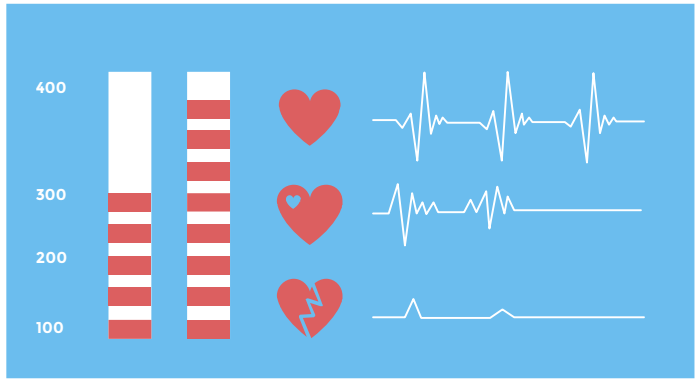
Antecedentes

¿Que es una arritmia cardiaca?

- Arritmia Cardiaca
  - ? — Ritmo cardiaco anormal
    - Cambio en la velocidad
  - Tipos
    - Taquicardia — + Frecuencia
    - Bradicardia — − Frecuencia
  - Muerte

ECG





- Muerte súbita
  - Manera prematura
  - por minuto
- identificar pacientes
- evaluación cardiológica
- Métodos diagnóstico
  - Enfoques clínicos
    - Incómodos
    - Costosos

Antecedentes

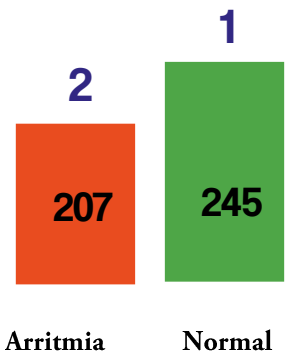
Dataset UCI

UCI Machine Learning Repository
Centro de aprendizaje automático y sistemas inteligentes

Identificación de Arritmia Cardiaca

- Datos
  - Dataset
    - ECG
    - UCI Repository
    - 279 Variables
    - Multiclase
      - Clase (280)
      - Clase Binario
        - 1 - normal — **245** casos
        - 2 - Arritmia — **207** casos
        - Tratamiento
    - **452** casos
  - Preprocesamiento
    - Imputación
      - NA
      - media
      - mediana
      - moda

Código	Clase	No. instancias
01	Normal	245
02	Cambios isquémicos (enfermedad de las arterias coronarias)	44
03	Infarto de miocardio anterior antiguo	15
04	Infarto de miocardio inferior antiguo	15
05	Taquicardia sinusal	13
06	Bradicardia sinusal	25
07	Contracción Ventricular Prematura (PVC)	3
08	Contracción prematura supraventricular	2
09	Bloqueo de rama izquierda	9
10	Bloqueo de rama derecha	50
11	1. grado Atrio Bloqueo ventricular	0
12	2. grado de bloqueo AV	0
13	3. grado de bloqueo AV	0
14	Hipertrofia del ventrículo izquierdo	4
15	Fibrilación o aleteo auricular	5
16	Otros	22

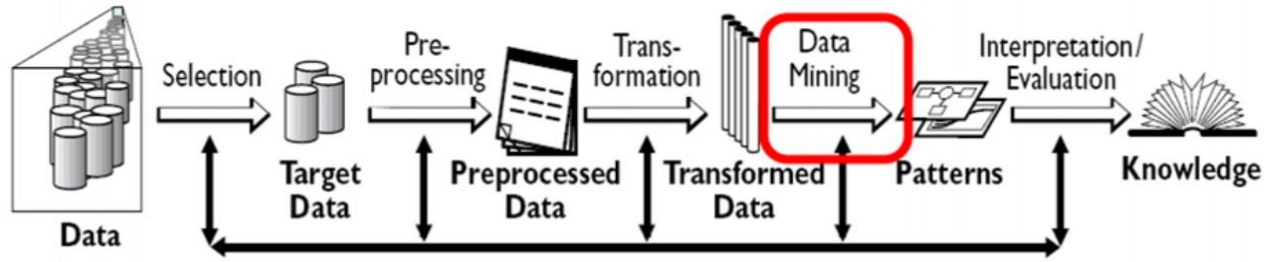


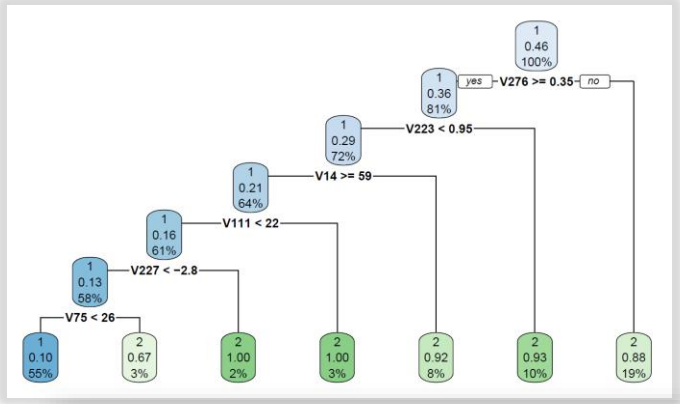
Antecedentes

KDD

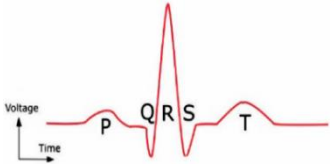
IA

KDD



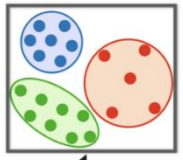



```
(V92 >= 24) and (V5 >= 93) and (V198 >= -37.6) => V279=2 (32.0/1.0)
(V198 >= -20.3) and (V276 <= 0.4) => V279=2 (37.0/8.0)
(V232 >= -4.8) and (V5 >= 87) => V279=2 (17.0/0.0)
(V14 <= 57) => V279=2 (17.0/1.0)
(V242 <= -13.2) and (V8 >= 195) => V279=2 (6.0/0.0)
(V14 >= 89) and (V237 >= -15.6) => V279=2 (12.0/2.0)
(V17 >= 48) and (V5 >= 97) => V279=2 (7.0/1.0)
(V92 >= 40) => V279=2 (5.0/0.0)
 => V279=1 (169.0/18.0)
```



Aplicación de algoritmos

Categoría o clase



Obtener información

DM - Aprendizaje Automático

Aprendizaje Supervisado

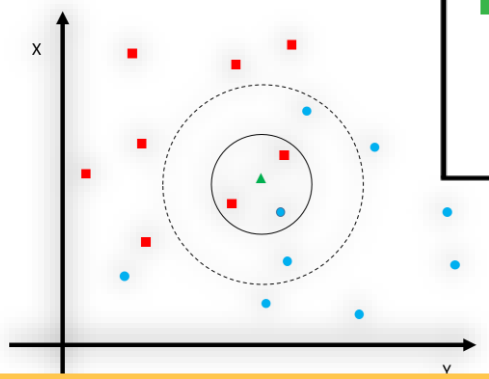
Predice etiquetas de clase

Clasificador

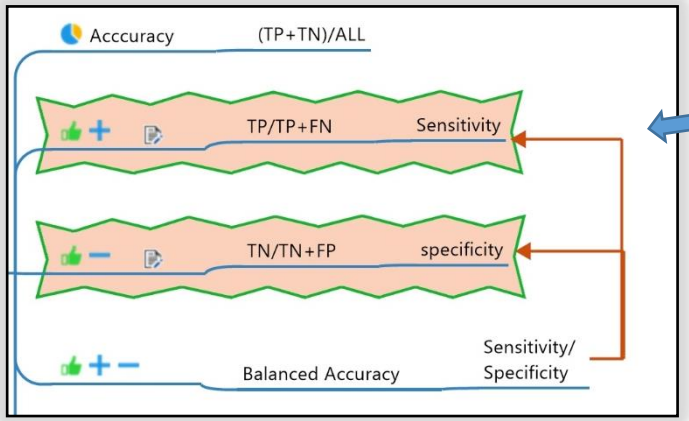
- Árboles
  - C4.5
  - Rpart
  - Random Forest
- Reglas
  - Jrip
  - OneR
- Otros
  - SVMLin
  - KNN

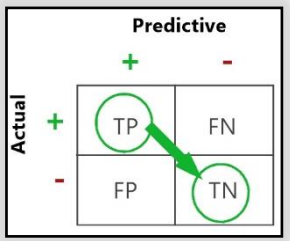
Selección de características

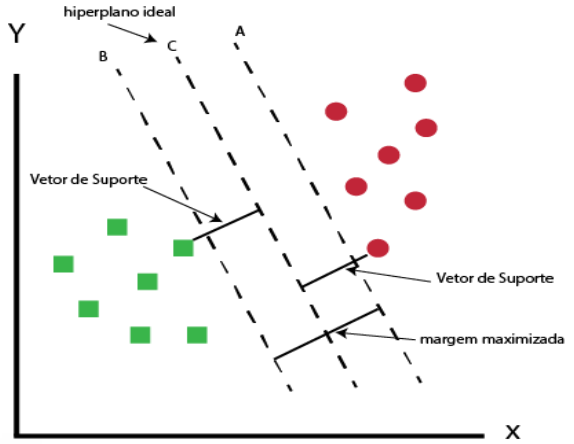
Reduce el tamaño del dataset



Medidas de rendimiento





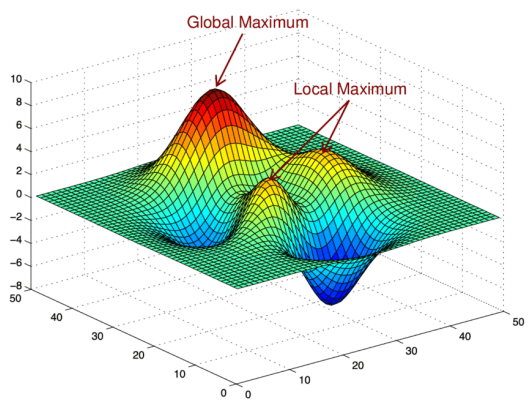


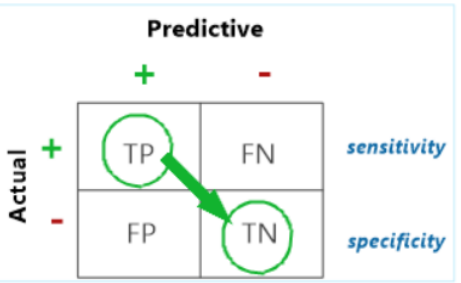
Antecedentes

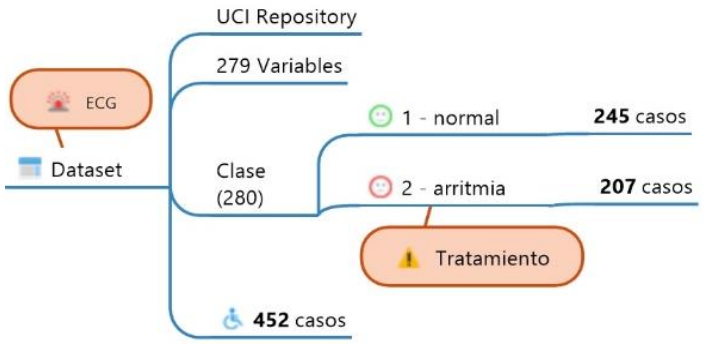
Selección de características

- Variables relevantes Arritmia Cardiaca
- Preprocesamiento
  - Imputación
    - media
    - mediana
    - moda
    - NA
- Selección de características
  - Función objetivo: minimizar
  - 1 Filtros (FSelector)
    - CFS
    - Consistency
    - Chi-squared
    - Information Gain
    - Gain Ratio
    - Symmetrical Uncertainty
    - OneR
  - 2 Metaheurísticas
    - Búqueda Tabú
    - Evolución Diferencial
    - Algoritmo Genético
    - Recocido Simulado
- Clasificadores
  - C4.5
  - Rpart
  - PART
  - Jrip
  - OneR
  - KNN
  - SVMLin
  - Random Forest
- Función objetivo: maximizar
  - Balanced Accuracy
  - Sensitivity
  - Specificity
  - Kappa
  - ROC

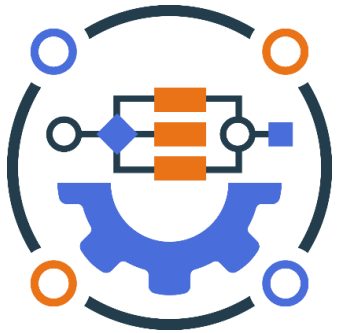
Balanced accuracy = *(Sensitivity + Specificity)/2*



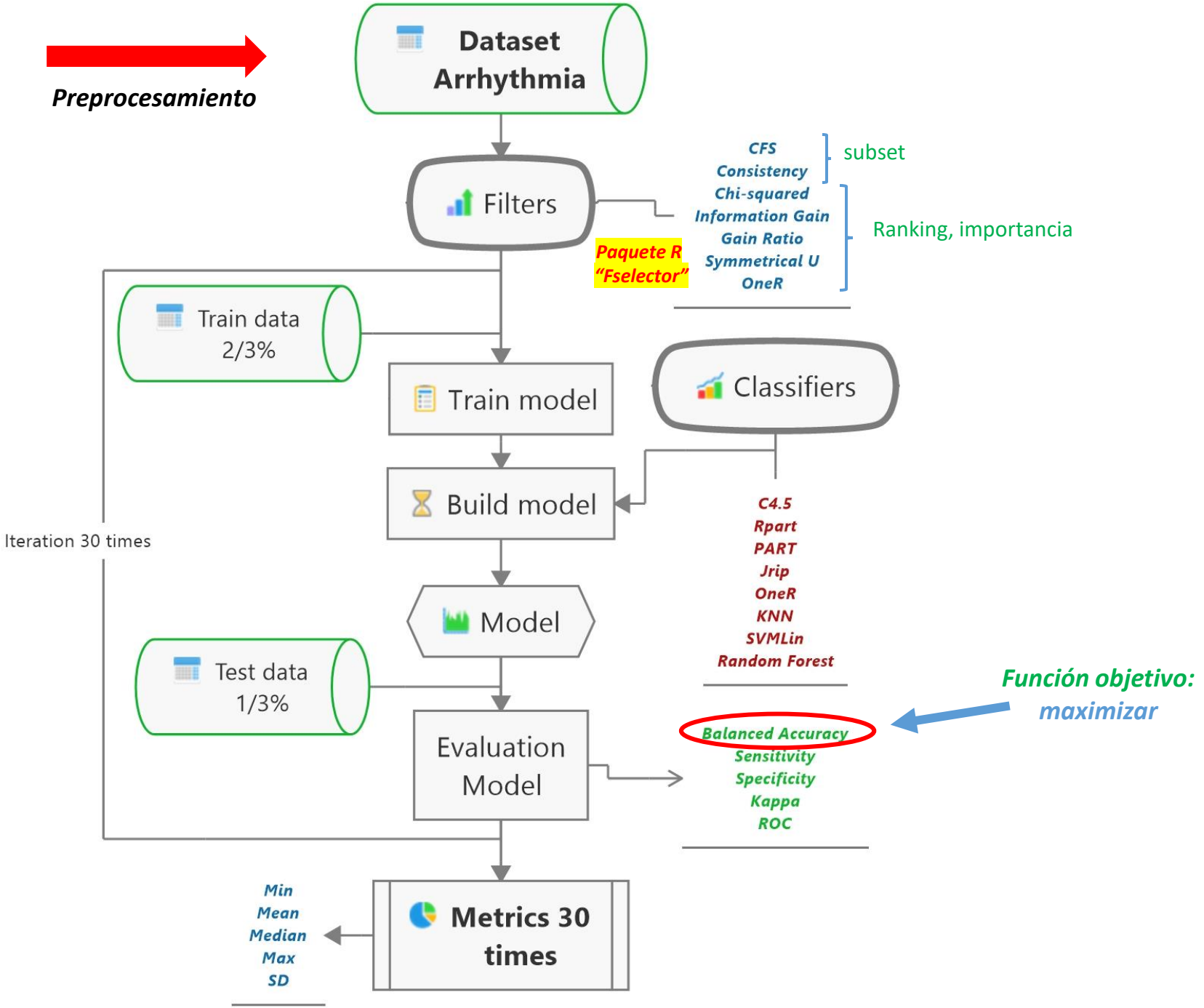




Antecedentes



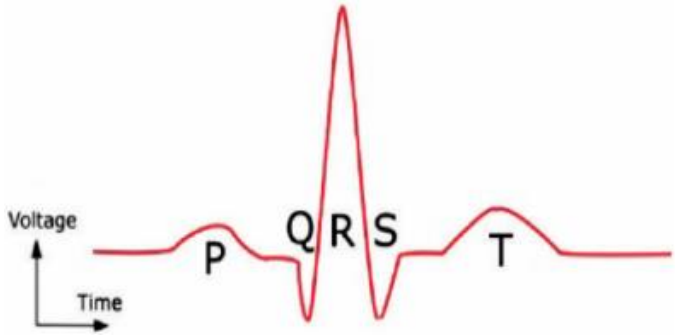
Proceso del Enfoque: Modelos predictivos aplicando métodos Filtro para la selección de variables relevantes



Resultados con los métodos filtros (promedio de accuracy balanceado)

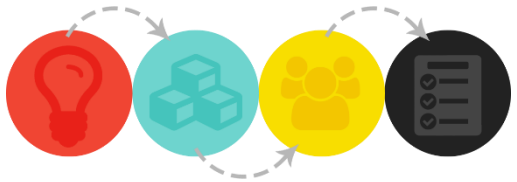
Línea Base
(usando todas variables)

	variables	*Average Balanced Accuracy*
	Completo	
C4.5	278	0.7443
Rpart	278	0.7405
PART	278	0.7488
JRip	278	0.7149
OneR	278	0.5567
KNN	278	0.6185
SVMLin	278	0.7081
Random Forest	**278**	**0.8147**



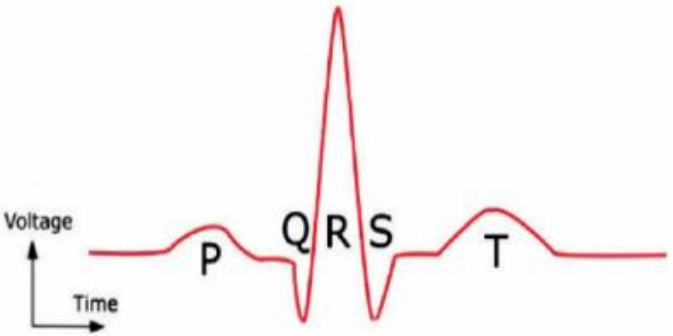
Selección de variables con filtros

	CFS		**Consistency**		**Chi-squared**		**Information Gain**		**Gain Ratio**		**Symmetrical Uncertainty**		**OneR**	
C4.5	37	0.7723	20	0.7565	18	0.7694	15	0.7666	44	0.7612	35	0.7706	53	0.7603
Rpart	37	0.7557	20	0.7684	20	0.7674	21	0.7664	41	0.7779	37	0.7660	42	0.7610
PART	37	0.7718	20	0.7568	35	0.7682	48	0.7700	66	0.7613	34	0.7687	84	0.7536
JRip	37	0.7457	20	0.7558	40	0.7580	13	0.7654	48	0.7653	28	0.7642	32	0.7547
OneR	37	0.5601	20	0.5627	2	0.6195	2	0.6038	2	0.5784	8	0.5886	2	0.6195
KNN	37	0.7358	20	0.6790	17	0.7357	30	0.7415	42	0.7372	7	0.7525	20	0.7156
SVMLin	37	0.7438	20	0.7526	49	0.7595	62	0.7606	48	0.7549	62	0.7537	62	0.7537
Random Forest	**37**	**0.8293**	20	0.8198	78	0.8261	64	0.8295	61	0.8268	59	0.8292	80	0.8275



Resultados de las variables relevantes encontradas actualmente

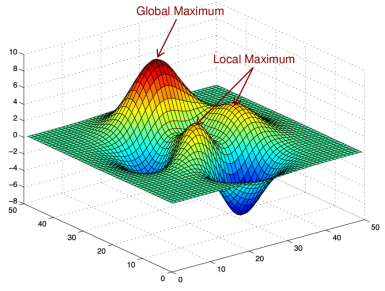
Variables altamente relevantes



Filtro CFS	Búsqueda Tabú	Evolución Diferencial	Algoritmo Genético	Recocido simulado
Random Forest	Random Forest	RPART	Random Forest	Random Forest
82.93%	83.34%	82.77%	82.56%	83.89%
37 variables	14 variables	7 variables	12 variables	8 variables
V5	V15	V15	V15	V15
V8	V30	V76	V76	V76
V13	V69	V112	V112	V112
V15	V76	V224	V169	V224
V18	V91	V257	V179	V234
V30	V112	V69	V197	V277
V69	V197	V234	V211	V69
V76	V211		V224	V169
V81	V217		V234	
V91	V224		V250	
V93	V228		V261	
V100	V234		V277	
V112	V250			
V141	V279			
V169				
V179				
V197				
V199				
V207				
V211				
V217				
V222				
V224				
V228				
V233				
V234				
V237				
V238				
V240				
V243				
V249				
V250				
V257				
V260				
V261				
V277				
V279				

Variables	Descripción de las variables
V5	QRS duración: Promedio de duración QRS en msec.
V8	Duración media de la onda T en msec.
V13	Ángulos vectoriales en grados en el plano frontal de QRS
V15	Frecuencia cardíaca: Número de latidos cardíacos por minuto
V18	Onda S del canal DI, ancho medio, en mseg.
V30	Onda S del canal DII, ancho medio, en mseg.
V69	Número de deflexiones intrínsecas del canal AVL
V76	Onda Q del canal AVF
V81	Número de deflexiones intrínsecas del canal AVF
V91	onda R' , pico pequeño justo después de R del canal V1
V93	Número de deflexiones intrínsecas del canal V1
V100	Onda Q del canal V2
V112	Onda Q del canal V3
V141	Número de deflexiones intrínsecas del canal V5
V169	QRSTA del canal DI
V179	QRSTA del canal DII
V197	Onda T del canal AVR
V199	QRSTA del canal AVR
V207	Onda T del canal AVL
V211	Onda Q del canal AVF
V217	Onda T del canal AVF
V222	Onda R del canal V1
V224	Onda R' del canal V1
V228	QRSA del canal V1
V233	Onda S del canal V2
V234	Onda R' del canal V2
V237	Onda T del canal V2
V238	QRSA del canal V2
V240	Onda JJ del canal V3
V243	Onda S del canal V3
V249	QRSTA del canal V3
V250	Onda JJ del canal V4
V257	Onda T del canal V4
V260	Onda JJ del canal V5
V261	Onda Q del canal V5
V277	Onda T del canal V6
V279	QRSTA del canal V6

Conclusiones



Optimización de modelos

- **Alcances**
  - Crear un modelo que ayude a los especialistas a detectar arritmia cardiaca con prontitud y dar un tratamiento oportuno a los paciente
- **Limitaciones**
  - Enfocado al conjunto de datos del repositorio UCI
- **Contribución**
  - Contribuir a la mejora en la detección de arritmias cardiacas utilizando el conjunto de datos del repositorio UCI.
  - Utilizar los desafíos reales
    - Desequilibrio de datos
    - Dimensionalidad alta
    - Selección de características relevantes.
  - Aportaciones a la comunidad científica y médica
    - evaluación exhaustiva del rendimiento de diferentes modelos
    - estrategia sobre la optimización de modelos de aprendizaje automático
  - Publicación de artículos y Ponencias
- **Resultados**
  - **37 variables** (CFS)
    - Búsqueda Tabú) **14** Variables **83.34%** RF
    - Evolución Diferencial **7** Variables **82.22%** RPART
    - Algoritmo Genético **12** Variables **82.56%** RF
    - Recocido Simulado **8** Variables **83.89%** RF
  - **82.93%** Random Forest ← Promedio accuracy balanceado
  - **Interfaz Shiny**

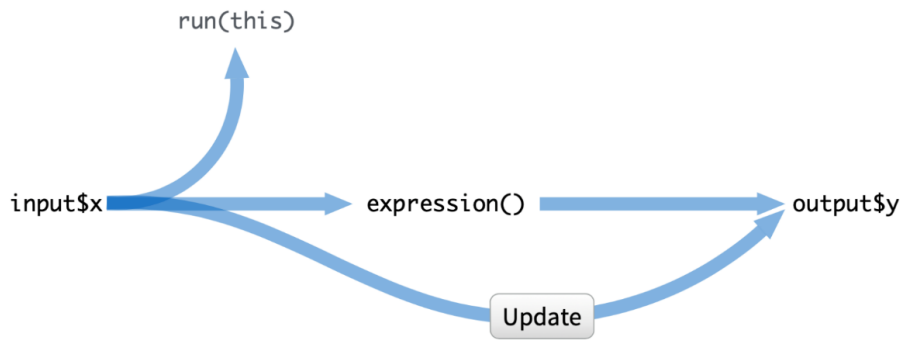
Que es shiny?

"Shiny es un paquete de código abierto que proporciona un marco elegante y potente para crear aplicaciones web. Utilizar R. Shiny ayuda a convertir nuestros análisis en aplicaciones web interactivas sin necesidad de conocimientos de HTML, CSS o JavaScript."

- Shiny
  - ?
    - ✅ Crear aplicaciones Web interactivas
    - ✅ Sin saber HTML, CSS o JavaScript
    - ✅ Programación Reactiva — Si cambia la **entrada** solo actualiza la **salida**
  - Requisitos
    - 1 R
    - 2 RStudio — Entorno de desarrol integrado(IDE)

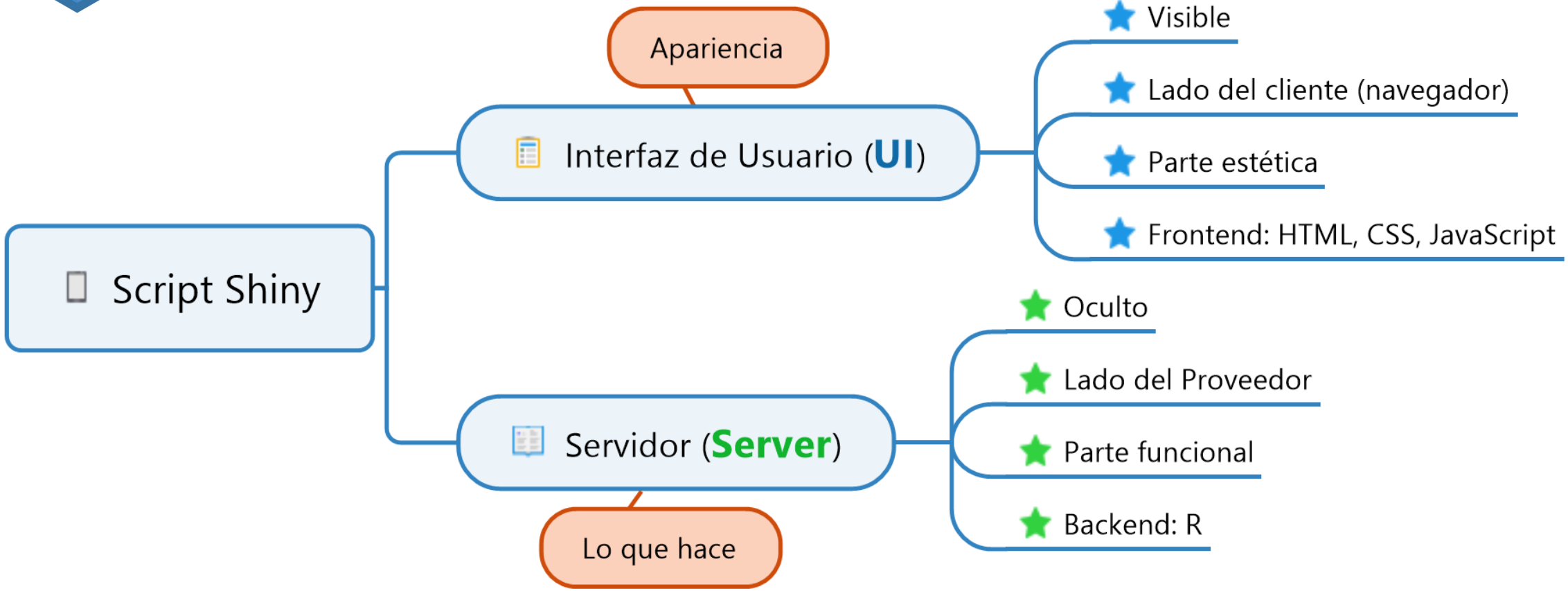
R
Python

Reactividad:
En una aplicación reactiva, el valor de una variable de depende de una entrada cambiante



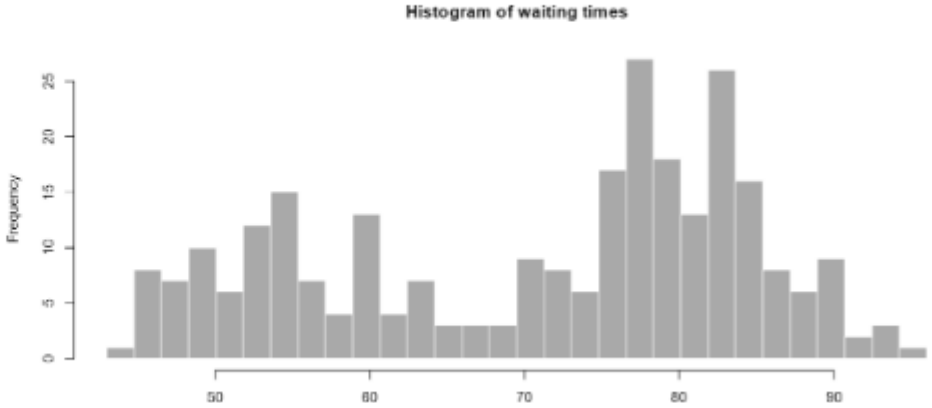
Arquitectura de una aplicación shiny

Arquitectura de una aplicación shiny



UI

Lo que vemos



Lo que escribimos

```
library(shiny)

ui <- fluidPage(

    # Un título
    titlePanel("Old Faithful Geyser Data"),

    # Un slider (input)
    sliderInput("bins", "Number of bins:", min = 1, max = 50, value = 30),

    # Una salida de gráfico (output)
    plotOutput("distPlot")

)
```

- ¿Cuántas secciones va a tener mi app?
- ¿Cómo se va a llamar?
- ¿Qué visualizaciones voy a usar (gráficos, tablas, etc.)? ¿Dónde va a ir cada una?
- ¿Qué elementos van a ser interactivos?
- ¿Cómo va a lucir mi app?
- ¿Va a ser dinámica?

Server

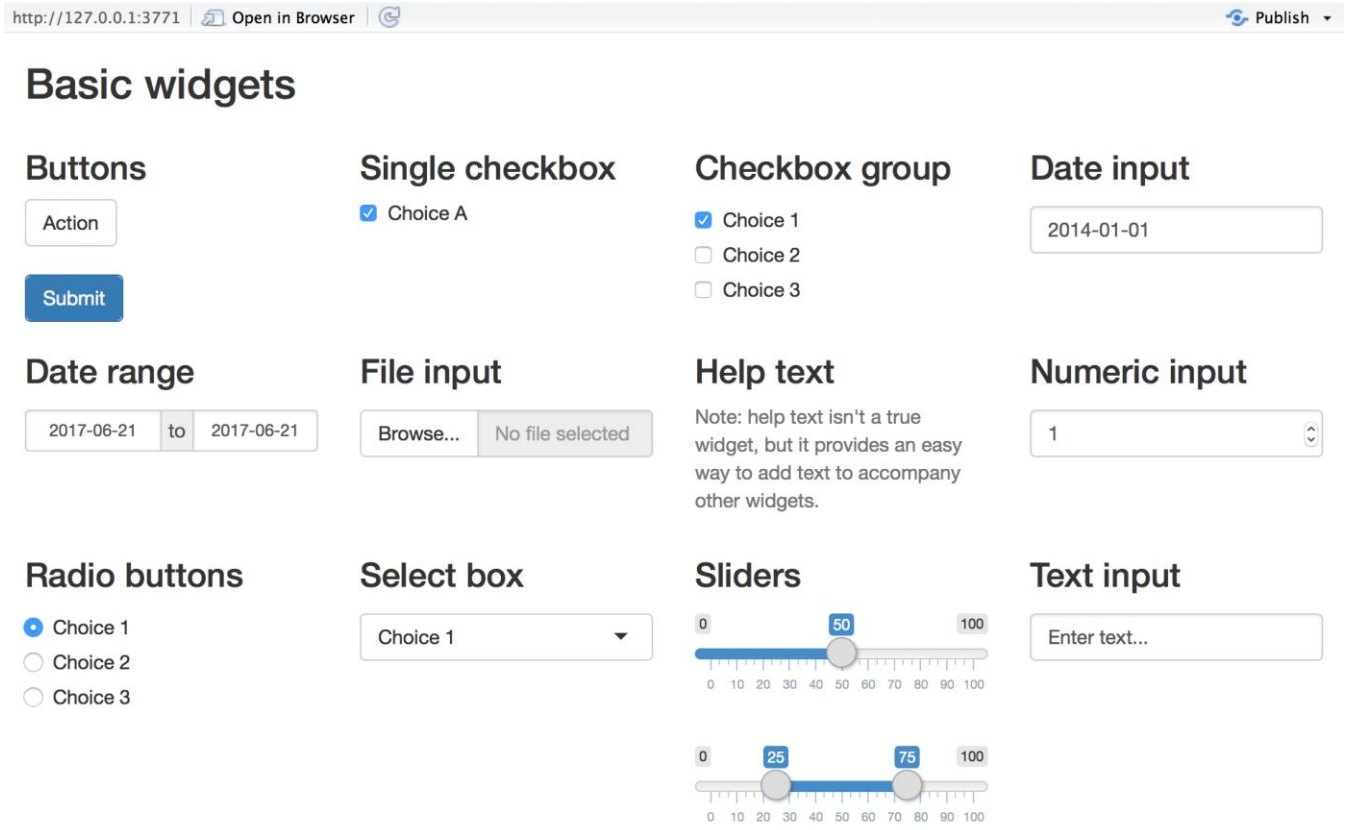
Lo que escribimos

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

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

        hist(x, breaks = bins, col = 'darkgray', border = 'white',
             xlab = 'Waiting time to next eruption (in mins)',
             main = 'Histogram of waiting times')
    })
}
```

En el server definimos la lógica de nuestra aplicación: donde sucede la magia. Escribimos nuestro código de R, ayudándonos de funciones específicas de Shiny para controlar los procesos internos, las entradas y las salidas.

- ¿Qué datos voy a mostrar?
- ¿Qué información le voy a pedir al usuario?
- ¿Cuándo se va a ejecutar determinado proceso?
- ¿Qué tipo de tablas o gráfico voy a mostrar?
- ¿Cómo se relacionan los elementos de mi app?
- ¿Qué debería pasar cuando el usuario haga *click* en un botón?

Shiny (widgets)



function	widget
actionButton	Botón para acciones
checkboxGroupInput	Un grupo de checkboxes
checkboxInput	Una simple casilla de verificación
dateInput	Seleccion de una fecha
dateRangeInput	Seleccion de un par de fechas
fileInput	Subir un archivo
helpText	Texto de ayuda para otro control
numericInput	Campo para insertar un dato numÃ©rico
radioButtons	Un grupo de radio buttons
selectInput	Clasica caja con opciones para seleccionar
sliderInput	Control para seleccionar un dato numérico
submitButton	Un botón de submit
textInput	Campo para ingresar un dato de texto

Los widgets son controles con los que el usuario va a interactuar, es decir que mediante estos controles el usuario va a comunicarse con nuestra aplicación, filtrar, seleccionar, ingresar un dato por parámetro.

Shiny : Guía Rápida

Shiny : : CHEAT SHEET

Basics

A **Shiny** app is a web page (**UI**) connected to a computer running a live R session (**Server**)

Users can manipulate the UI, which will cause the server to update the UI's displays (by running R code).

APP TEMPLATE

Begin writing a new app with this template. Preview the app by running the code at the R command line.

```
library(shiny)
ui <- fluidPage()
server <- function(input, output){}
shinyApp(ui = ui, server = server)
```

- **ui** - nested R functions that assemble an HTML user interface for your app
- **server** - a function with instructions on how to build and rebuild the R objects displayed in the UI
- **shinyApp** - combines `ui` and `server` into an app. Wrap with **runApp()** if calling from a sourced script or inside a function.

SHARE YOUR APP

The easiest way to share your app is to host it on shinyapps.io, a cloud based service from RStudio

1. Create a free or professional account at **http://shinyapps.io**
2. Click the **Publish** icon in the RStudio IDE or run: **rsconnect::deployApp**("<path to directory>")

Build or purchase your own Shiny Server
at www.rstudio.com/products/shiny-server/

Building an App

Complete the template by adding arguments to fluidPage() and a body to the server function.

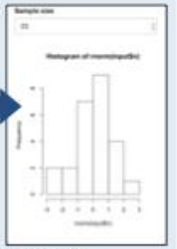
Add inputs to the UI with *Input() functions

Add outputs with *Output() functions

Tell server how to render outputs with R in the server function. To do this:

1. Refer to outputs with **output$<id>**
2. Refer to inputs with **input$<id>**
3. Wrap code in a **render*()** function before saving to output

```
library(shiny)

ui <- fluidPage(
  numericInput(inputId = "n",
    "Sample size", value = 25),
  plotOutput(outputId = "hist")
)

server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$n))
  })
}

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



Save your template as **app.R**. Alternatively, split your template into two files named **ui.R** and **server.R**.

```
library(shiny)
ui <- fluidPage(
  numericInput(inputId = "n",
    "Sample size", value = 25),
  plotOutput(outputId = "hist")
)
server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$n))
  })
}
shinyApp(ui = ui, server = server)
```

```
# ui.R
fluidPage(
  numericInput(inputId = "n",
    "Sample size", value = 25),
  plotOutput(outputId = "hist")
)
```

ui.R contains everything you would save to ui.

```
# server.R
function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$n))
  })
}
```

server.R ends with the function you would save to server.

No need to call **shinyApp()**

Save each app as a directory that holds an **app.R** file (or a **server.R** file and a **ui.R** file) plus optional extra files.

The directory name is the name of the app

(optional) defines objects available to both ui.R and server.R

(optional) used in showcase mode

(optional) data, scripts, etc.

(optional) directory of files to share with web browsers (images, CSS, .js, etc.) Must be named "**www**"

Launch apps with `runApp(<path to directory>)`

Outputs - render*() and *Output() functions work together to add R output to the UI

works with

DT::**renderDataTable**(expr, options, callback, escape, env, quoted)	**dataTableOutput**(outputId, icon, …)
renderImage(expr, env, quoted, deleteFile)	**imageOutput**(outputId, width, height, click, dblclick, hover, hoverDelay, inline, hoverDelayType, brush, clickId, hoverId)
renderPlot(expr, width, height, res, …, env, quoted, func)	**plotOutput**(outputId, width, height, click, dblclick, hover, hoverDelay, inline, hoverDelayType, brush, clickId, hoverId)
renderPrint(expr, env, quoted, func, width)	**verbatimTextOutput**(outputId)
renderTable(expr,…, env, quoted, func)	**tableOutput**(outputId)
renderText(expr, env, quoted, func)	**textOutput**(outputId, container, inline)
renderUI(expr, env, quoted, func)	**uiOutput**(outputId, inline, container, …) & **htmlOutput**(outputId, inline, container, …)

Inputs

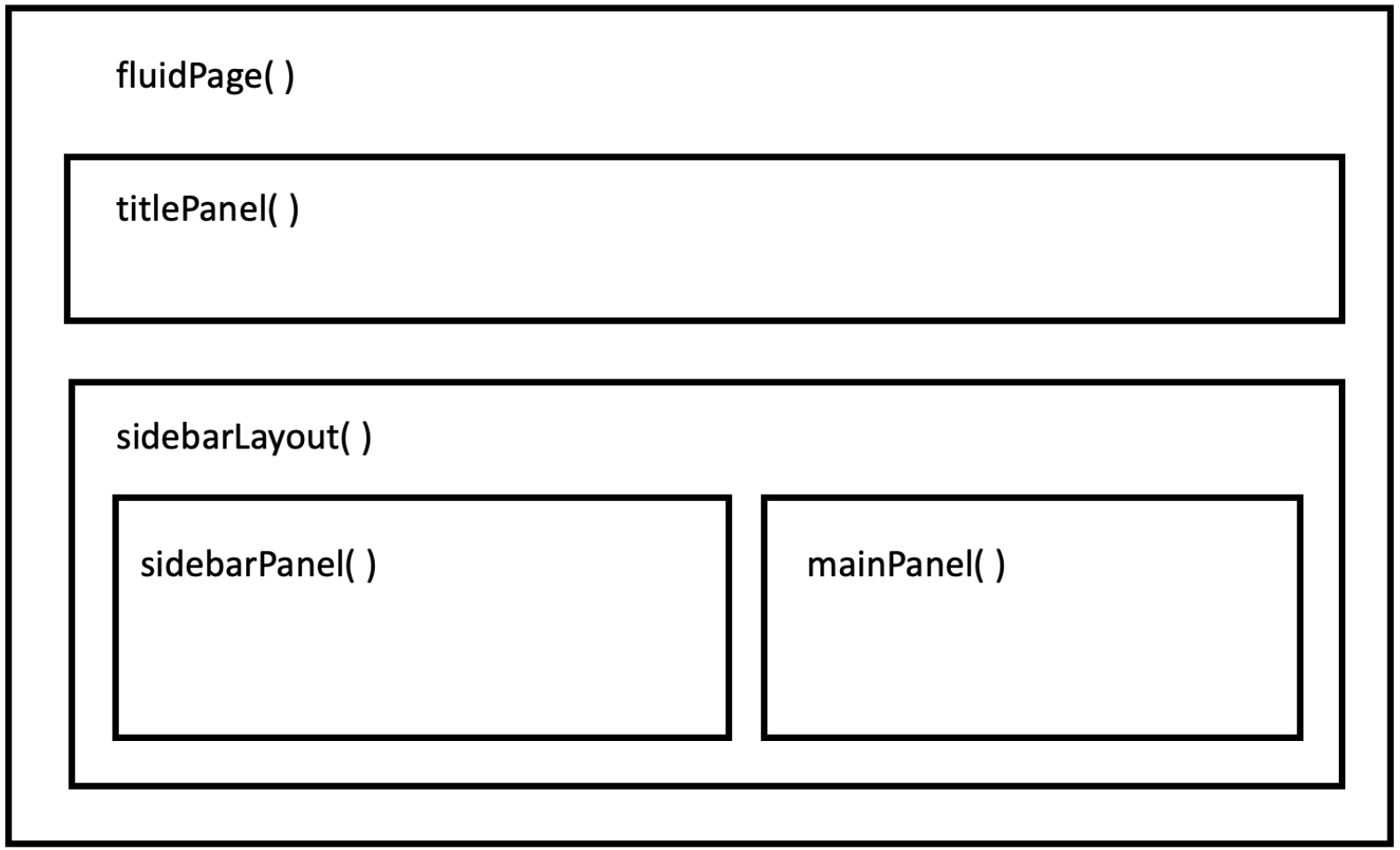
collect values from the user

Access the current value of an input object with **input$<inputId>**. Input values are **reactive**.

Action	**actionButton**(inputId, label, icon, …)
Link	**actionLink**(inputId, label, icon, …)
Choice 1 / Choice 2 / Choice 3	**checkboxGroupInput**(inputId, label, choices, selected, inline)
Check me	**checkboxInput**(inputId, label, value)
	dateInput(inputId, label, value, min, max, format, startview, weekstart, language)
	dateRangeInput(inputId, label, start, end, min, max, format, startview, weekstart, language, separator)
Choose File	**fileInput**(inputId, label, multiple, accept)
1	**numericInput**(inputId, label, value, min, max, step)
•••••••	**passwordInput**(inputId, label, value)
Choice A / Choice B / Choice C	**radioButtons**(inputId, label, choices, selected, inline)
Choice 1 / Choice 1 / Choice 2	**selectInput**(inputId, label, choices, selected, multiple, selectize, width, size) (also selectizeInput())
	sliderInput(inputId, label, min, max, value, step, round, format, locale, ticks, animate, width, sep, pre, post)
Apply Changes	**submitButton**(text, icon) (Prevents reactions across entire app)
Enter text	**textInput**(inputId, label, value)

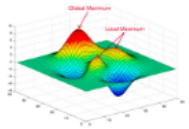
Shiny : Panel

```
ui = fluidPage(

  titlePanel("Newspaper Map"),

  sidebarLayout(

    sidebarPanel = sidebarPanel(),
    mainPanel = mainPanel()

  )
)
```



Ejemplo de Interfaz (Dashboard)

Optimización de Modelos para Identificar Arritmia Cardíaca con Aprendizaje Automático



Menú Principal

Reiniciar Interfaz

Cargar Dataset

Browse... No file selected

Dataset Arritmia Precargado

Leer encabezados

Seleccionar Variable a Predecir:

Clase Negativa (1):

Clases Positivas (2):

Convertir a Clase Binaria

Información | Datos | Variables Relevantes (Filtros) | Variables Relevantes (Metaheurísticas) | Referencias

Bienvenido

Para comenzar, cargue un archivo CSV o seleccione el dataset precargado usando las opciones de la barra lateral.

Una vez cargado el dataset, se habilitarán las pestañas de datos, preprocesamiento y modelado.

Acerca de la arritmia cardiaca y el electrocardiograma (ECG)

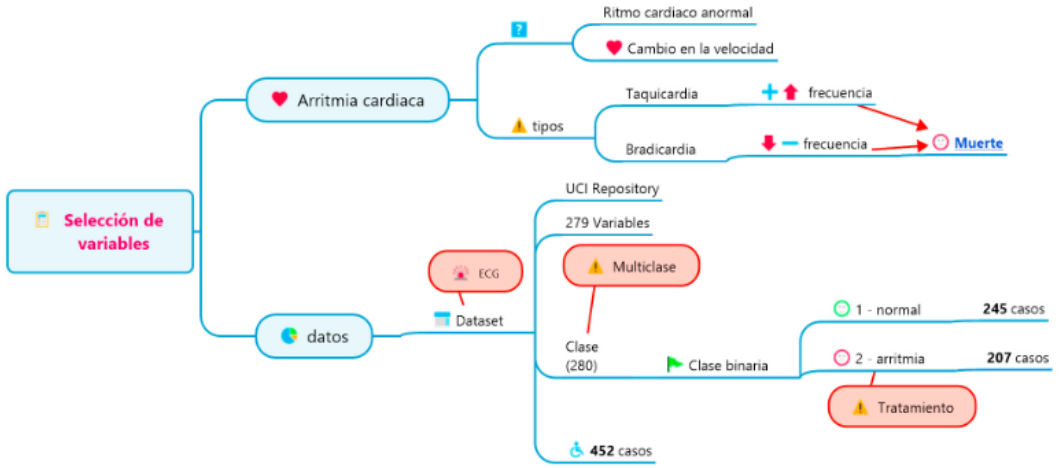
Una arritmia es un latido o ritmo cardíaco irregular. Puede clasificarse en distintas clases.

La electrocardiografía (ECG) es un método para analizar la condición cardíaca de un paciente. Un ECG es una representación eléctrica de la actividad contráctil del corazón y se puede registrar utilizando electrodos de superficie colocados en las extremidades o el tórax del paciente.

Información del conjunto de datos

Esta base de datos contiene 279 atributos, 206 de los cuales son de valor lineal y el resto son nominales.

El objetivo es distinguir entre la presencia y ausencia de arritmia cardiaca y clasificarla en uno de los 16 grupos. La clase 01 se refiere al ECG 'normal', las clases 02 a 15 a diferentes clases de arritmia y la clase 16 al resto de las no clasificadas. Por el momento, existe un programa informático que realiza dicha clasificación.



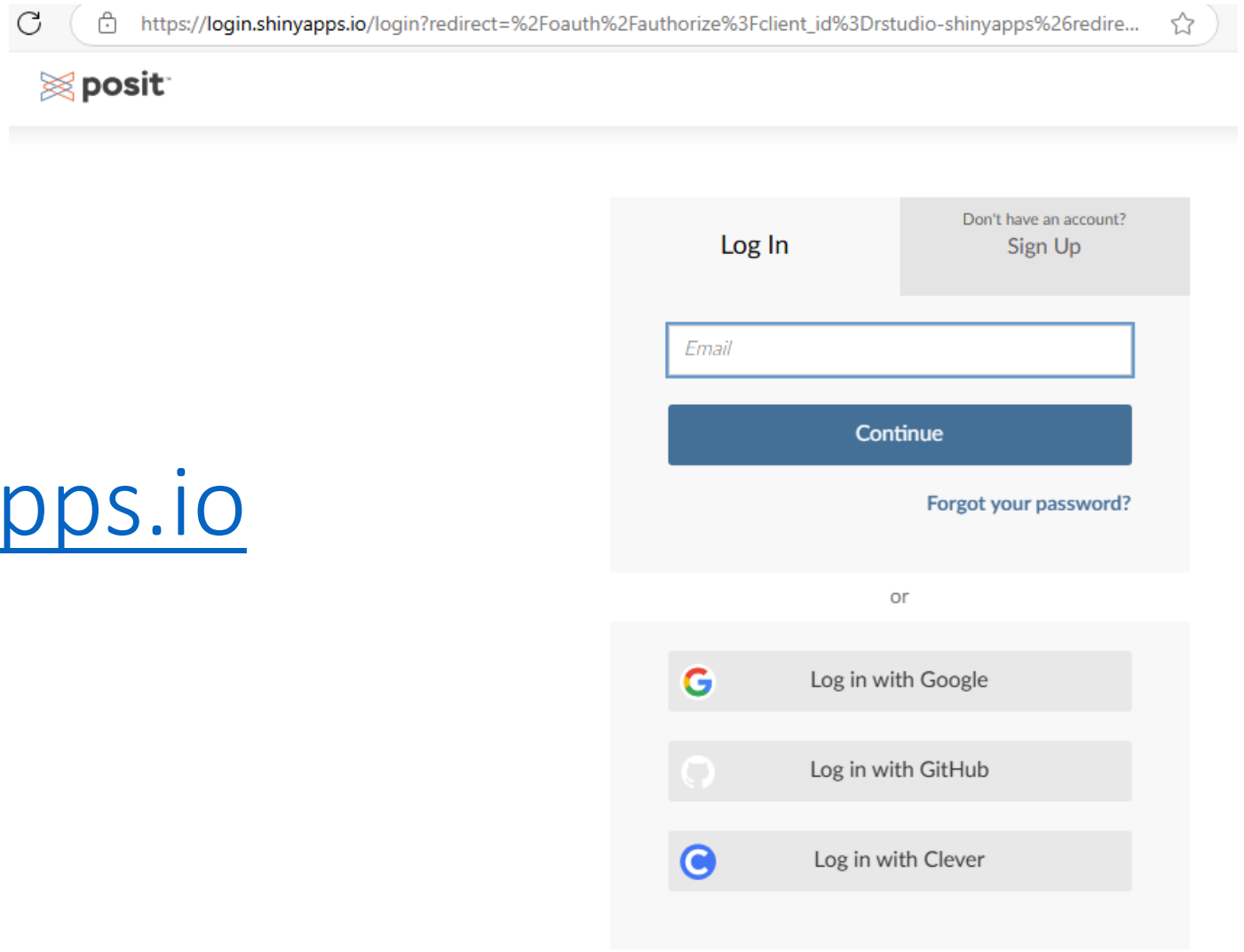
Instalación de R y RStudio

https://posit.co/download/rstudio-desktop/

- **1) Install R**
  - **Download and install R**
  - **Download R-4.5.0 for Windows**
- **2) Install Rstudio**
  - **Download RStudio Desktop for windows**

Alta en Shiny apps

shinyapps.io



Crear una app Shiny

- Instalar shiny: install.packages("shiny")
- 1) Con script
- 2) Como Shiny
  - Como archivo: New file > Shiny Web App > single file o multiple
  - Como proyecto: New Project > Shiny Application

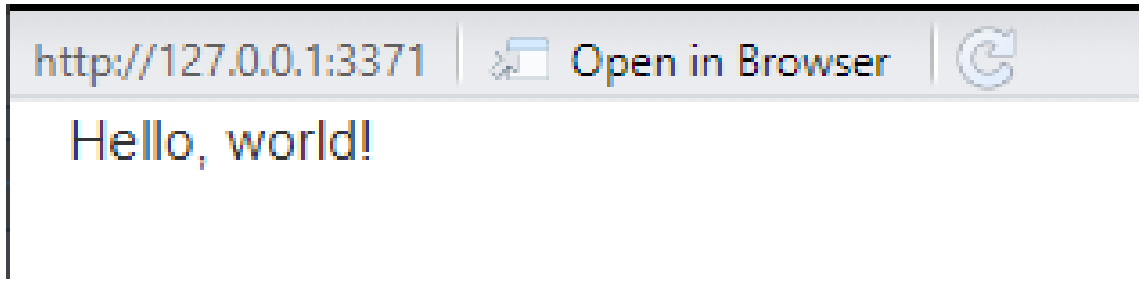
Como recomendación, nuestra Shiny app debería ser un proyecto en sí mismo, que contenga todos, y únicamente, los scripts y archivos necesarios para su funcionamiento.

¡Vamos a Rstudio!

Mi primer shiny app

Desde script:

```
library(shiny)
ui <- fluidPage(
  "Hello, world!"
)
server <- function(input, output, session) {
}
shinyApp(ui, server)
```



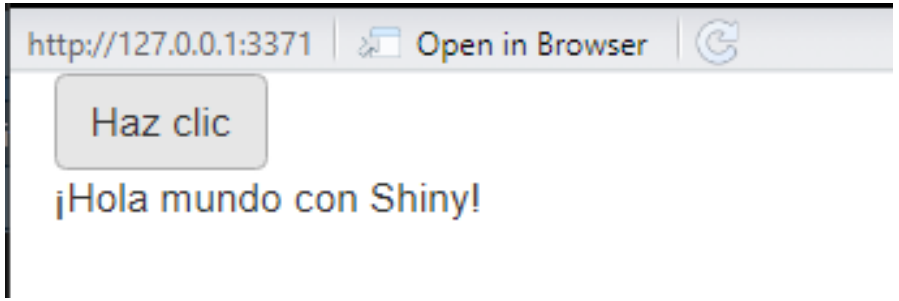
Ejercicio

Desde script:

```
library(shiny)

ui <- fluidPage(
    textInput(“nombre”, “Cuál es tu nombre?”),
    textOutput(“saludo”)
)

server <- function(input, output, session) {
    output$saludo <- renderText({
        paste0(“Hola ", input$nombre)
    })
}

shinyApp(ui, server)
```

2da shiny app

```
library(shiny) #librería para que funcione la app

ui <- fluidPage( #página fluida
  actionButton("boton", "Haz clic"), #botón de acción/ nombre de la variable/ texto
  textOutput("texto") #salida de texto/ nombre de la variable
)

server <- function(input, output) { #función de entrada y salida del servidor
  output$texto <- renderText({ #salida de texto/$nombre de la variable de salida <- función reactiva de texto
    if (input$boton > 0) "¡Hola mundo con Shiny!" else "" #si la entrada del botón es mayor que 0 entonces muestra "¡Hola mundo con Shiny!"
  })
}

shinyApp(ui, server)
```



3ra shiny app

Interfaz de usuario:

Como shiny app:

```
library(shiny) #carga el paquete shiny, necesario para construir la app web
#Interfaz de usuario
ui <- fluidPage(
  titlePanel("Old Faithful Geyser Data"),
  sidebarLayout(
    sidebarPanel(
      sliderInput("bins", "Number of bins:", min = 1, max = 50, value = 30)
    ),
    mainPanel(
      plotOutput("distPlot")
    )
  )
)
```

- `fluidPage()` : crea una página web que se ajusta automáticamente al tamaño de la pantalla.
- `titlePanel()` : muestra el título "Old Faithful Geyser Data" en la parte superior.
- `sidebarLayout()` : divide la pantalla en dos partes:
  - `sidebarPanel()` : contiene un control deslizante (`sliderInput`) llamado `bins` . Permite elegir cuántos *bins* (intervalos) tendrá el histograma (entre 1 y 50).
  - `mainPanel()` : mostrará el gráfico generado (`plotOutput("distPlot")`).

3ra shiny app

Lógica del servidor:

```
server <- function(input, output) {
  output$distPlot <- renderPlot({
    x <- faithful[, 2]
    bins <- seq(min(x), max(x), length.out = input$bins + 1)
    hist(x, breaks = bins, col = 'darkgray', border = 'white')
  })
}
```

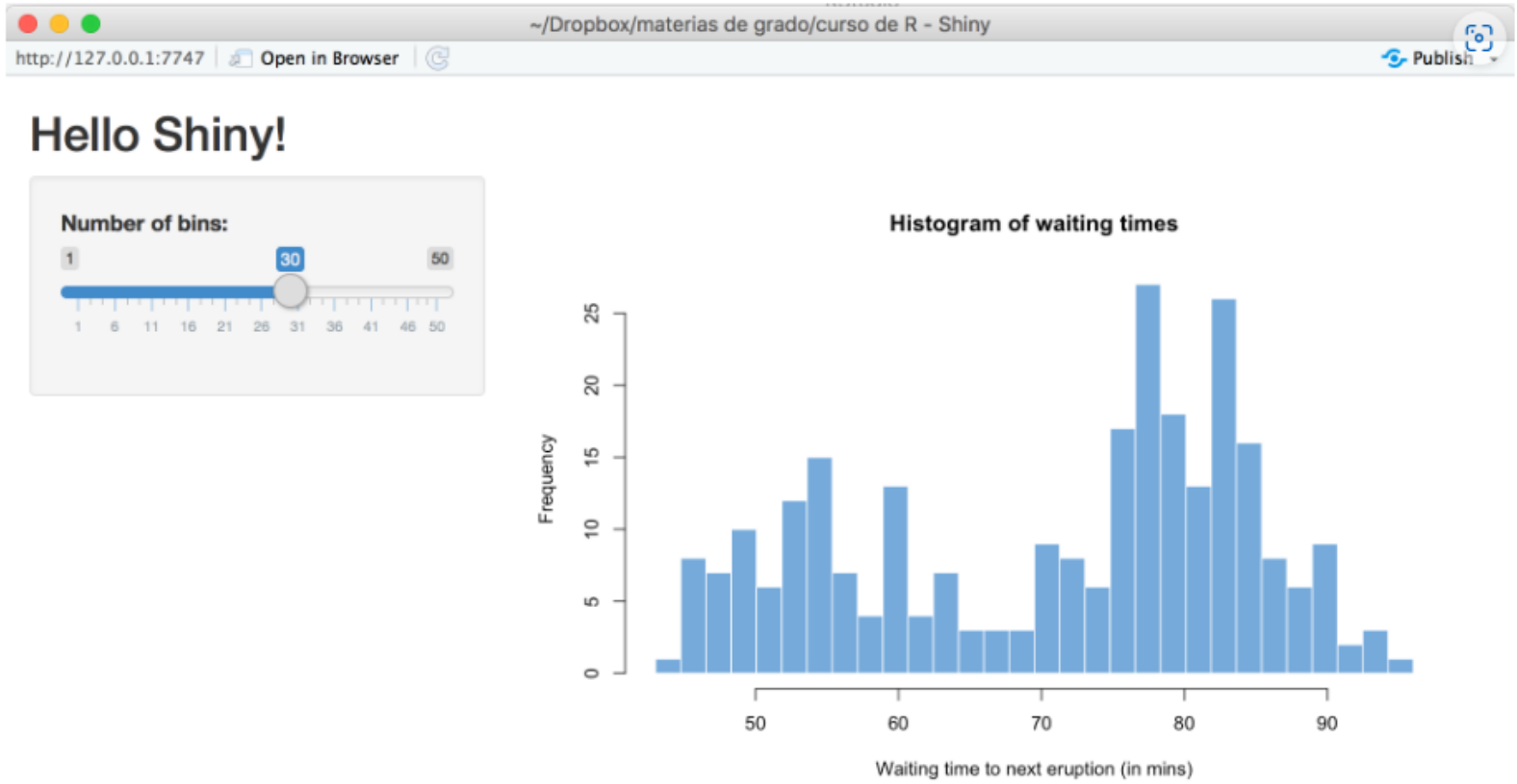
- `server` define **qué pasa** cuando el usuario interactúa.
- `output$distPlot <- renderPlot({...})` : genera un gráfico dinámicamente.
  - `x <- faithful[, 2]` : selecciona la segunda columna del dataset `faithful` (los tiempos de espera entre erupciones del géiser).
  - `bins <- seq(...)` : genera una secuencia de cortes para el histograma. Depende de cuántos bins seleccionó el usuario en el control deslizante.
  - `hist()` : dibuja el histograma usando los cortes generados.

3ra shiny app

Ejecución de la app:

shinyApp(ui = ui, server = server)

- Lanza la aplicación web combinando la interfaz (`ui`) y la lógica del servidor (`server`).



Ejercicio

Ejecución de la app:

```
library(shiny)

ui <- fluidPage(
  actionButton("boton", "Haz clic"),
  textOutput("salida")
)

server <- function(input, output) {
  output$salida <- renderText({
    if (input$boton == 0) {
      "Esperando interacción..."
    } else {
      paste("¡Has hecho clic", input$boton, "veces!")
    }
  })
}

shinyApp(ui, server)
```

Aplicación donde el usuario sube un archivo CSV y se visualiza como tabla interactiva.

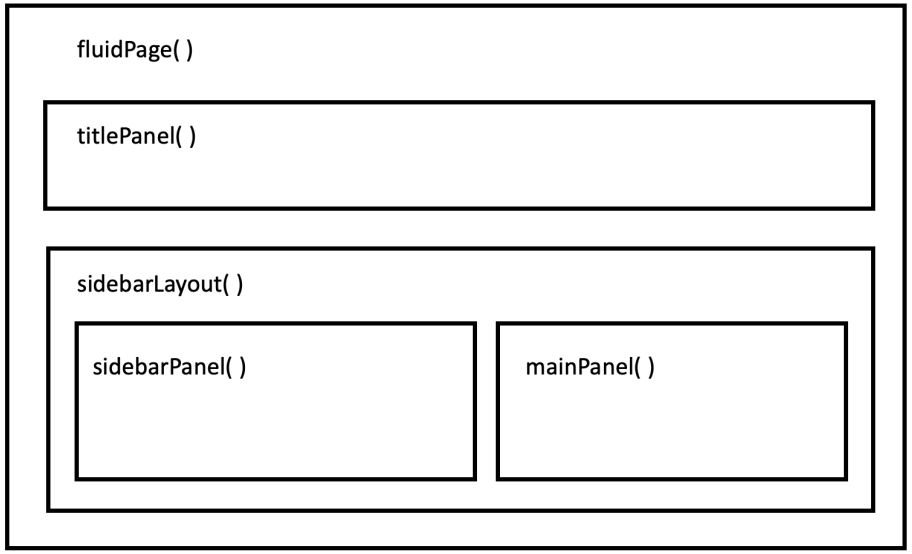
```
ui <- fluidPage(
  fileInput("archivo", "Sube un CSV"),
  DT::dataTableOutput("tabla")
)

server <- function(input, output) {
  datos <- reactive({
    req(input$archivo)
    read.csv(input$archivo$datapath)
  })

  output$tabla <- DT::renderDataTable({
    DT::datatable(datos())
  })
}

shinyApp(ui, server)
```

Creación de Interfaces en Shiny (UI)

```
library(shiny)
library(DT)       # Para tablas interactivas
library(ggplot2)   # Para gráficos

ui <- fluidPage(
 titlePanel("Exploración de Dataset de Arritmia Cardiaca"),   # Título grande

 sidebarLayout(
  sidebarPanel(
   # Aquí van los botones, selectores, etc.
   h3("Controles de usuario"),   # Subtítulo
   selectInput("columna", "Selecciona una variable numérica:", choices =
NULL), # Se llenará dinámicamente
   actionButton("actualizar", "Actualizar gráfica")
  ),
    mainPanel(
   h3("Resultados"),  # Subtítulo
   plotOutput("grafica"),        # Gráfica dinámica
   DT::dataTableOutput("tabla")   # Tabla dinámica
  )
 )
)
```

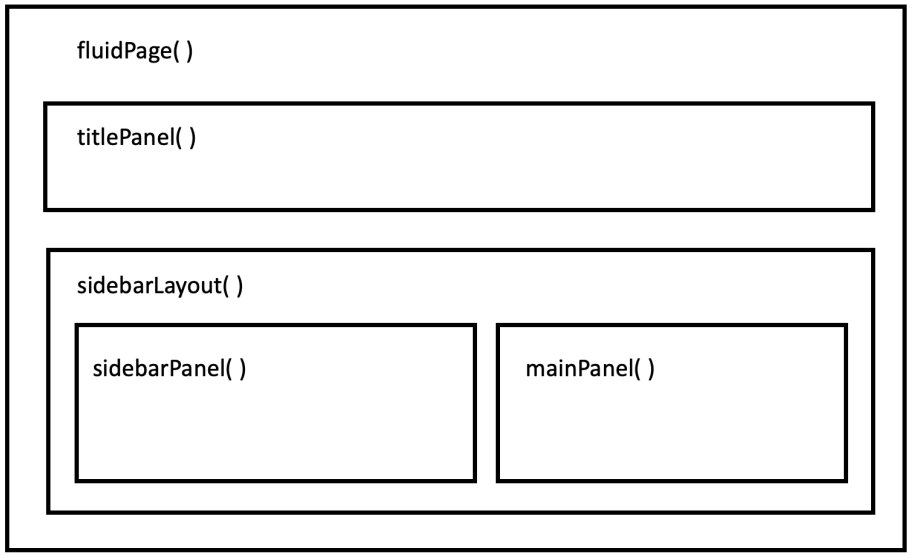
Diseño de interfaces con fluidPage() y sidebarLayout()

- **fluidPage(): es el "contenedor principal" de toda la app.**
- **sidebarLayout(): divide la pantalla en dos partes:**
  - **Una barra lateral (sidebarPanel) donde pones los controles.**
  - **Una parte principal (mainPanel) donde ves los resultados.**



Resumen de elementos dinámicos

Elemento	¿Qué hace?	¿Dónde lo usamos?
`fluidPage()`	Estructura principal	`ui`
`sidebarLayout()`	Divide la pantalla	`ui`
`sidebarPanel()`	Controles de usuario	`ui`
`mainPanel()`	Resultados (gráfica, tabla)	`ui`
`selectInput()`	Selección de variables	`ui`
`actionButton()`	Botón para actualizar	`ui`
`plotOutput()`	Mostrar gráfico	`ui`
`DT::dataTableOutput()`	Mostrar tabla	`ui`
`renderPlot()`	Crear gráfico dinámico	`server`
`DT::renderDataTable()`	Crear tabla dinámica	`server`



Reactividad en Shiny

En Shiny hay cuatro funciones principales de reactividad

Función	¿Para qué sirve?	
`reactive()`	Crear un objeto reactivo (datos procesados)	Crea un objeto que se actualiza automáticamente cuando cambian sus entradas.
`observeEvent()`	Escuchar eventos y hacer acciones cuando ocurren	Escucha un evento (como presionar un botón) y ejecuta un bloque de código.
`eventReactive()`	Crear un objeto reactivo pero sólo cuando ocurre un evento	Crea datos reactivos pero solo cuando ocurre un evento (por ejemplo, al presionar un botón).
`isolate()`	Leer valores **sin activar** reactividad	Sirve para leer un valor sin activar reactividad (sin que se actualice automáticamente).

Gracias!!!

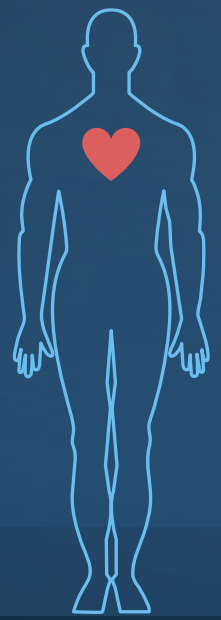
UNIVERSIDAD JUÁREZ AUTÓNOMA DE TABASCO

"ESTUDIO EN LA DUDA. ACCIÓN EN LA FE"

DIVISIÓN ACADÉMICA DE CIENCIAS Y TECNOLOGÍAS DE LA INFORMACIÓN

DACyTI

"CIENCIA Y TECNOLOGÍA AL SERVICIO DE LA HUMANIDAD"



Msc. Santiago Arias-García 231H18005@alumno.ujat.mx

Dr. José Hernández-Torruco jose.hernandezt@ujat.mx

Dra. Betania Hernández-Ocaña betania.hernandez@ujat.mx