

Curso: Introducción a R en investigación biomédica



Unidad de Apoyo Metodológico (UAM)

Lore Zumeta Olaskoaga

Jone Renteria Aguirregabiria

Donostia, 24-26 noviembre 2025



Osakidetza



ÍNDICE – Sesión 1 Fundamentos de programación en R

1.1 Introducción a R y RStudio

1.2 Programación básica en R

1.3 Estadística descriptiva en R

1.4 Manipulación y reestructuración de datos



Osakidetza



Introducción a R y RStudio

ÍNDICE

- 1.1 Introducción a R y RStudio
- 1.2 Programación básica en R
- 1.3 Estadística descriptiva en R
- 1.4 Manipulación y reestructuración de datos

SESIÓN 1: FUNDAMENTOS DE PROGRAMACIÓN EN R

1.1 Introducción a R y RStudio

- Entorno de R: ¿Qué es R? ¿Qué es RStudio?
- Administración e instalación de paquetes y uso de librerías de R

1.2 Programación básica en R

1.3 Estadística descriptiva en R

1.4 Manipulación y reestructuración de datos



¿QUÉ ES R?

R es un lenguaje de programación de código abierto para computación estadística.

Principales características:

- *Open source* = gratuito + cualquier usuario puede modificarlo y mejorarlo
- Lenguaje de programación simple y efectivo.
- Manipulación y gestión eficiente de datos.
- Herramientas integradas de análisis estadístico.
- Capacidades gráficas.



7



- R está basado en el lenguaje **S**, creado en 1976 por John Chambers en Bell Labs.
- En **1993**, Robert Gentleman y Ross Ihaka (Universidad de Auckland) desarrollaron su propia versión y la llamaron **R**.
- En **1995**, R se liberó como **software abierto**, permitiendo contribuciones globales.
- Debido a su origen común, **muchos conceptos y funciones son similares en R y S-PLUS**.



Ventajas de R frente a otros paquetes estadísticos

- Código abierto
- Gratis: sin coste de licencia.
- Multiplataforma: funciona en Windows, Linux y Mac.
- Ampliamente usado en nuevos métodos estadísticos, con librerías en constante crecimiento
- Gran comunidad activa y abundantes recursos (manuales, libros, foros).

9



<http://cran.r-project.org/>

1: Install R

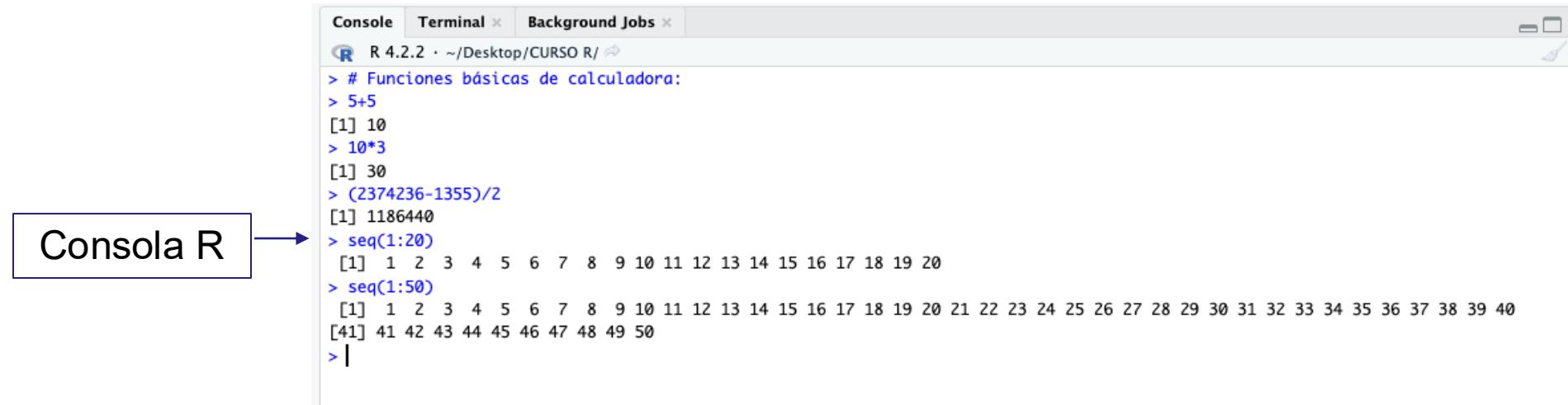
RStudio requires R 3.6.0+. Choose a version of R that matches your computer's operating system.

R is not a Posit product. By clicking on the link below to download and install R, you are leaving the Posit website. Posit disclaims any obligations and all liability with respect to R and the R website.

DOWNLOAD AND INSTALL R



10



Consola R →

```
Console Terminal × Background Jobs ×
R 4.2.2 · ~/Desktop/CURSO R/ ↵
> # Funciones básicas de calculadora:
> 5+5
[1] 10
> 10*3
[1] 30
> (2374236-1355)/2
[1] 1186440
> seq(1:20)
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
> seq(1:50)
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
[41] 41 42 43 44 45 46 47 48 49 50
> |
```

11

- El símbolo `>` (mayor que) es el *prompt symbol* (símbolo de entrada). Cuando aparece, significa que puedes empezar a escribir los comandos.
- `[1]` muestra que es el primer resultado de la operación solicitada. En otros casos, puede devolver múltiples resultados numerados (por ejemplo `[41]` en la imagen).
- Las flechas arriba / abajo del teclado nos permiten utilizar el histórico de comandos.

- **RStudio** es el entorno de desarrollo integrado (IDE, *Integrated Development Environment*) más utilizado para trabajar con **R**.
- Proporciona una **interfaz clara y amigable** para escribir, ejecutar y organizar código.
- En solo lugar incluye editor de código, consola, panel de gráficos...etc.
- Su objetivo es **facilitar y optimizar el trabajo con R**, especialmente para análisis de datos, estadística y visualización.



1

Introducción a R y RStudio

1

INSTALACIÓN DE RSTUDIO

<http://cran.r-project.org/>

2: Install RStudio

[DOWNLOAD RSTUDIO DESKTOP FOR WINDOWS](#)

Size: 296.74 MB | [SHA-256: 439D3200](#) | Version: 2025.09.2+418 |
Released: 2025-10-29



Zip/Tarballs

OS	Download	Size	SHA-256
Windows 10/11	RSTUDIO-2025.09.2-418.ZIP	408.84 MB	20D46F57



DIFERENCIAS ENTRE R Y RSTUDIO

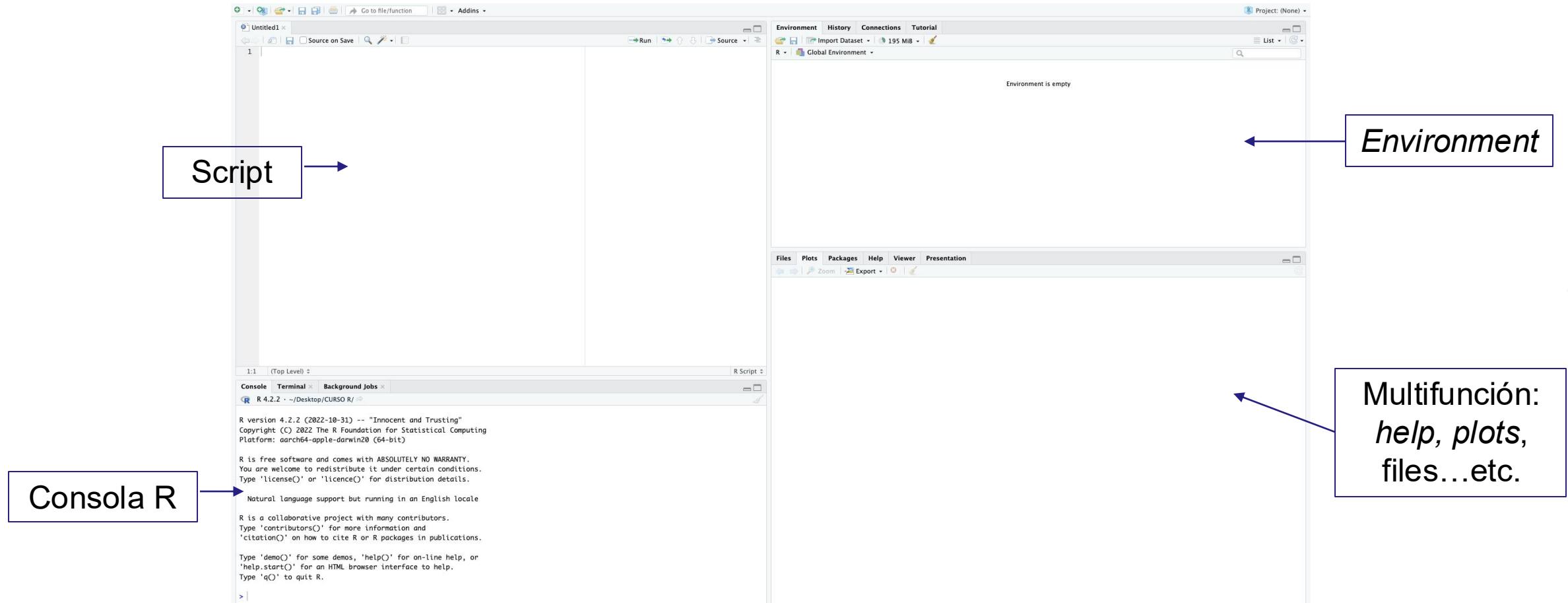
Aspecto	R	RStudio
Naturaleza	Lenguaje de programación	Entorno de desarrollo integrado (IDE)
Función principal	Ejecutar análisis estadístico y visualización de datos	Facilitar el uso de R con herramientas gráficas
Uso	Se escribe código directamente	Ofrece editor, paneles, depuración y gestión de proyectos
Productividad	Básico, requiere más manejo manual	Mejora la organización y productividad
Requisito	Se instala de forma independiente	Necesita tener R instalado para funcionar

14

1

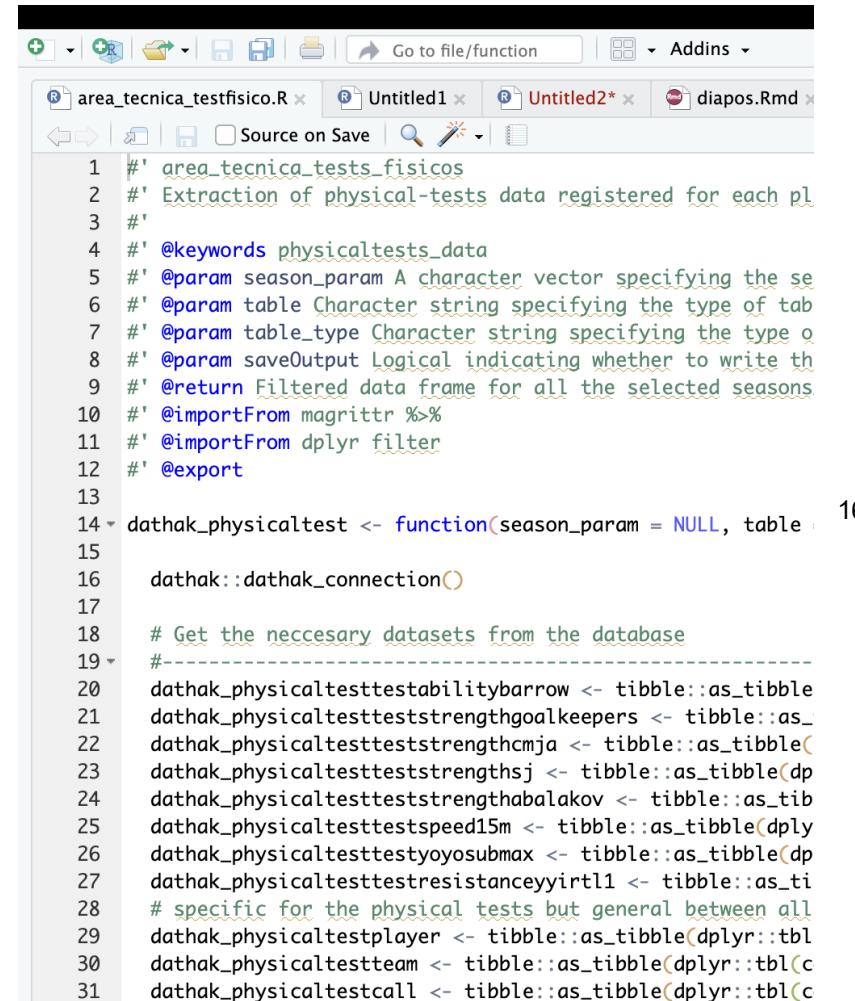
Introducción a R y RStudio

¿CÓMO SE VE?



TRABAJAR MEDIANTE SCRIPTS

- En lugar de guardar el **workspace**, es mejor mantener un **registro de los comandos** que usamos.
- Así podemos **reproducir el entorno** más adelante.
- Para ejecutar: selecciona el código y pulsa **Ctrl + R (Run)**.
- Al finalizar la sesión, guarda el **script** como registro permanente de tu trabajo.

```

1 #' area_tecnica_tests_fisicos
2 #' Extraction of physical-tests data registered for each pl
3 #
4 #' @keywords physicaltests_data
5 #' @param season_param A character vector specifying the se
6 #' @param table Character string specifying the type of tab
7 #' @param table_type Character string specifying the type o
8 #' @param saveOutput Logical indicating whether to write th
9 #' @return Filtered data frame for all the selected seasons
10 #' @importFrom magrittr %>%
11 #' @importFrom dplyr filter
12 #' @export
13
14 dathak_physicaltest <- function(season_param = NULL, table =
15
16   dathak::dathak_connection()
17
18 # Get the neccesary datasets from the database
19 #
20 dathak_physicaltesttestabilitybarrow <- tibble::as_tibble(
21 dathak_physicaltestteststrengthgoalkeepers <- tibble::as_
22 dathak_physicaltestteststrengthcmja <- tibble::as_tibble(
23 dathak_physicaltestteststrengthsj <- tibble::as_tibble(dp
24 dathak_physicaltestteststrengthabalakov <- tibble::as_tib
25 dathak_physicaltesttestspeed15m <- tibble::as_tibble(dply
26 dathak_physicaltesttestyoyosubmax <- tibble::as_tibble(dp
27 dathak_physicaltesttestresistanceyyirtl1 <- tibble::as_t
28 # specific for the physical tests but general between all
29 dathak_physicaltestplayer <- tibble::as_tibble(dplyr::tblc
30 dathak_physicaltestteam <- tibble::as_tibble(dplyr::tblc
31 dathak_physicaltestcall <- tibble::as_tibble(dplyr::tblc

```

- Un **entorno organizado** que guarda scripts, datos y resultados en una carpeta única.
- Facilita mantener todo el trabajo **estructurado y reproducible**.

Beneficio	Descripción
Orden y claridad	Todos los archivos relacionados en un solo lugar
Reproducibilidad	Puedes recrear el análisis en cualquier momento
Colaboración	Compartir el proyecto asegura que otros tengan el mismo entorno
Facilidad de uso	Rutas relativas automáticas, sin necesidad de configurar manualmente
Escalabilidad	Ideal para crecer desde pequeños scripts hasta proyectos grandes

17

- Conjunto de **funciones, datos y documentación** que amplían las capacidades de R.
- Se instalan en **librerías** (carpetas del sistema).
- Permiten reutilizar código y facilitar tareas específicas (ej. leer archivos, hacer gráficos).
- Hay algunos paquetes predeterminados:
 - Al iniciar R se cargan automáticamente los paquetes: datasets, utils, grDevices, graphics, stats, methods (*además del paquete base*)
- Normalmente **necesitamos más paquetes** para trabajar, hay que **instalar nuevos**:

```
install.packages("nombre_paquete")
```



IMPORTANTE! Los paquetes solo hay que instalarlos una única vez

- Una **librería** es un **directorio del sistema** donde se guardan los paquetes instalados.
- R trae por defecto una librería: **R_HOME/library** con los paquetes básicos y recomendados.
- Derivado de los paquetes que instalamos, es posible tener nuestras propias librerías.
- Se cargan al inicio del script:

```
library(nombre_paquete)
```

19

“In 2015, R added 1,357 packages, counting only CRAN, or approximately 27,642 functions. During 2015 alone, R added more functions than SAS Institute has written in its entire history.”

Fuente: <https://r4stats.com/articles/popularity/>

EVOLUCIÓN DE LAS CAPACIDADES DE R

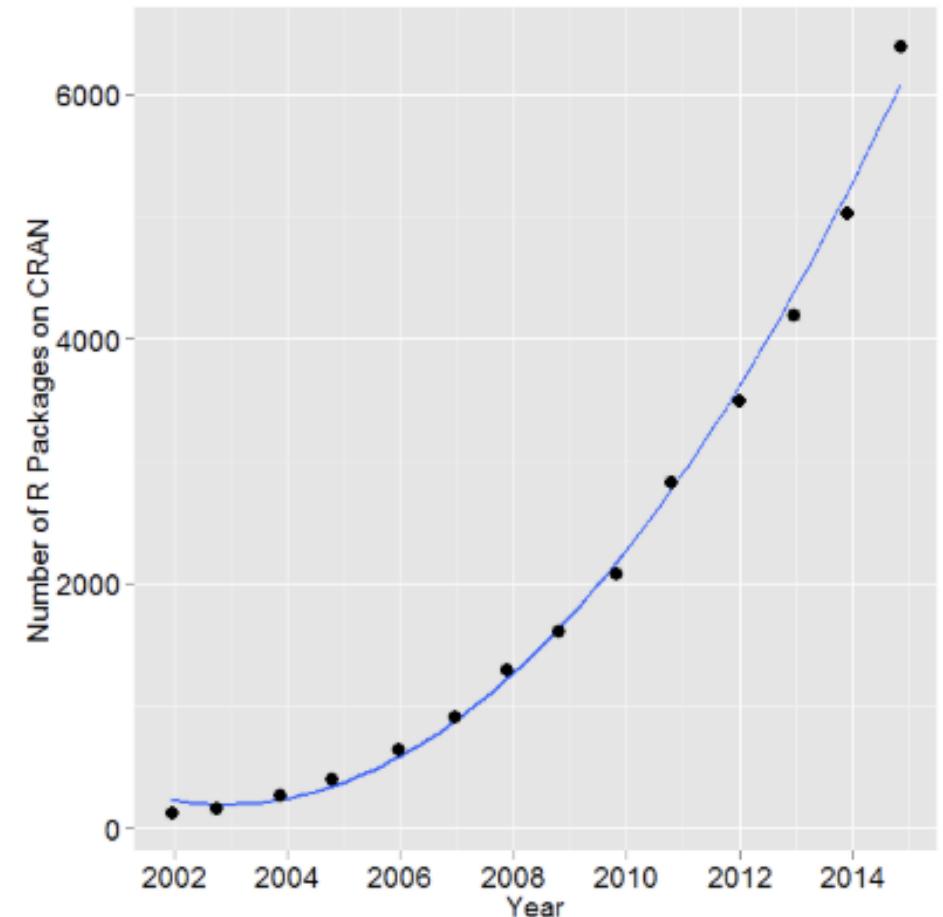


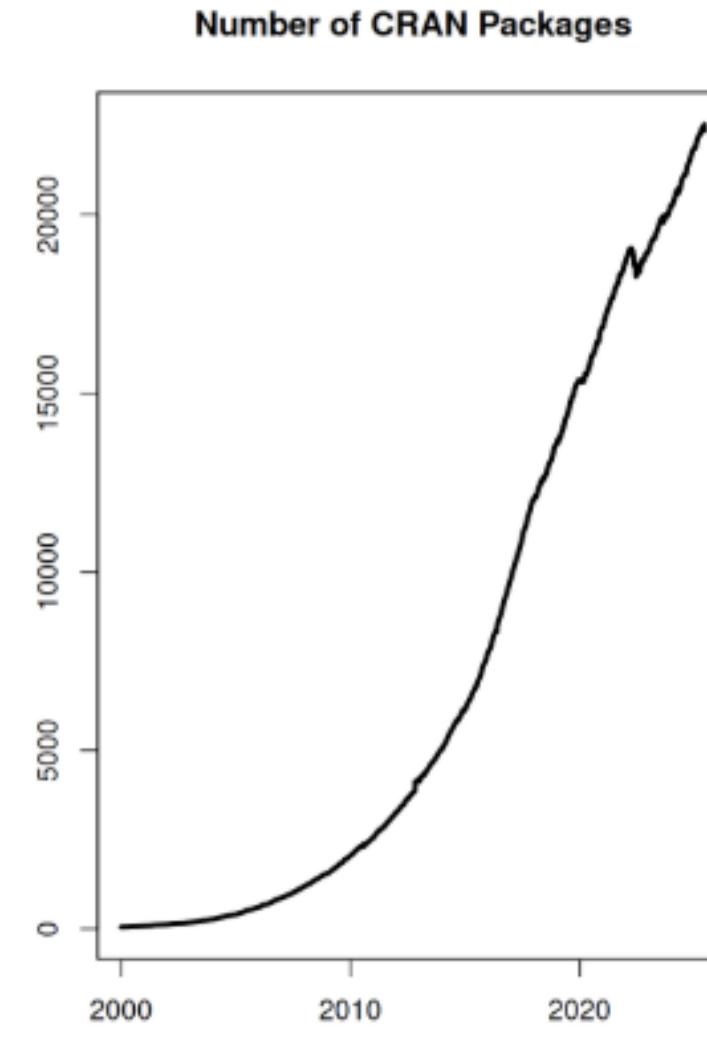
Figure 8. Number of R packages available on its main distribution site for the last version released in each year.



EVOLUCIÓN DE LAS CAPACIDADES DE R

Crecimiento del número de paquetes activos en el repositorio de paquetes CRAN, a principios de 2025.

Fuente:
<https://journal.r-project.org/news/RJ-2025-1-cran/>



- R tiene un **sistema de ayuda**. Se puede invocar de diferentes maneras:
 - `help.start()` – Ayuda en formato HTML
 - **`help('mean')`, `?mean`** – La más utilizada, ayuda sobre la función en concreto
 - `help.search('plot')`, `??plot` – busca en todas las páginas de ayuda la palabra clave (útil cuando sabes el tema pero no la función en concreto)
 - `RSiteSearch('plot')` – Busca en la web de CRAN

The screenshot shows the R Help interface with the following details:

- Menu Bar:** Files, Plots, Packages, Help, Viewer, Presentation.
- Toolbar:** Home, Find in Topic.
- Search Bar:** R: Arithmetic Mean ▾ Find in Topic.
- Content Area:**
 - Function Name:** mean {base}
 - Title:** Arithmetic Mean
 - Description:** Generic function for the (trimmed) arithmetic mean.
 - Usage:**

```
mean(x, ...)

## Default S3 method:
mean(x, trim = 0, na.rm = FALSE, ...)
```
 - Arguments:**
 - x**: An R object. Currently there are methods for numeric/logical vectors and [date](#), [date-time](#) and [time interval](#) objects. Complex vectors are allowed for `trim = 0`, only.
 - trim**: the fraction (0 to 0.5) of observations to be trimmed from each end of **x** before the mean is computed. Values of `trim` outside that range are taken as the nearest endpoint.
 - na.rm**: a logical evaluating to TRUE or FALSE indicating whether NA values should be stripped before the computation proceeds.
 - ...**: further arguments passed to or from other methods.
 - Value:** If `trim` is zero (the default), the arithmetic mean of the values in **x** is computed, as a numeric or complex vector of length one. If **x** is not logical (coerced to numeric), numeric (including integer) or complex, `NA_real_` is returned, with a warning. If `trim` is non-zero, a symmetrically trimmed mean is computed with a fraction of `trim` observations deleted from each end before the mean is computed.
 - References:** Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.
 - See Also:** [weighted.mean](#), [mean.POSIXct](#), [colMeans](#) for row and column means.
 - Examples:**

```
x <- c(0:10, 50)
xm <- mean(x)
c(xm, mean(x, trim = 0.10))
```



1. Instalar paquetes necesarios > `install.packages("NombrePaquete")`
2. Carga de paquetes en un script > `library("NombrePaquete")`
3. Definir el entorno de trabajo (*working environment*)
 - *Es la carpeta donde R busca y guarda archivos por defecto*
 - *Se define con la función `setwd()`* > `setwd("ruta/de/tu/carpeta")`
 - Se consulta con `getwd()`
 - Buena práctica: fijarlo al inicio del script
4. Hacer uso de las funciones de un paquete (por ejemplo, leer un archivo)

24

Programación básica en R

ÍNDICE

- 1.1 Introducción a R y RStudio
- 1.2 Programación básica en R
- 1.3 Estadística descriptiva en R
- 1.4 Manipulación y reestructuración de datos

SESIÓN 1: FUNDAMENTOS DE PROGRAMACIÓN EN R

1.1 Introducción a R y RStudio

1.2 Programación básica en R

- Elementos básicos y comandos básicos de R: operadores, funciones, variables y tipos de datos
- Importación de datos en el entorno R y RStudio
- Tipos de datos en R
- Conjuntos y estructuras de datos (*data.frame*)
- Exportación de datos

1.3 Estadística descriptiva en R

1.4 Manipulación y reestructuración de datos

1.2

Programación básica en R

Elementos y comandos básicos en R

- Operadores: realizan operaciones aritméticas (+, -, *, /), comparaciones (==, <, >), y lógicas (&, |, !).
- Funciones: instrucciones que ejecutan tareas específicas, como `mean()`, `sum()`, `sqrt()`.
- Tipos de datos: numéricos, caracteres, lógicos, además de estructuras como vectores, listas, matrices y *data frames*.

27

- Un **objeto** es una unidad donde se guarda información
- Cada objeto tiene un **nombre (identificador)** y una **estructura** que define cómo se almacena esa información.
- Podemos **manipular objetos** aplicando funciones sobre ellos.
- Se crean normalmente con el operador de asignación < –
- Los objetos permanecen en **memoria** para poder reutilizarlos más adelante.

1.2

- Definimos los objetos →
- Listamos los objetos de la memoria →
- ¿Existe x2? →
- Borrar un objeto →

```
Console Terminal × Background Jobs ×
R 4.2.2 · ~/Desktop/CURSO R/ 🌐
> x1 <- 4
> x2 <- 6
> x3 <- 10
> ls()
[1] "x1" "x2" "x3"
> exists("x2")
[1] TRUE
> rm(x3)
> ls()
[1] "x1" "x2"
>
```

- En R, **todo es un objeto**: datos, resultados y hasta las funciones.
- Cada objeto pertenece a una **clase**, que define cómo se almacena la información y qué operaciones se pueden realizar sobre ella.
- La función **class()** permite identificar el tipo de objeto y entender cómo R lo interpreta.

El tipo character permite almacenar cadenas de texto; se utilizan las comillas (" , ') para delimitar las cadenas de caracteres:

```
x1 <- "Kaixo, Biogipuzkoan gaude"  
class(x1)
```

```
## [1] "character"
```

El tipo numeric permite almacenar números reales:

```
x2 <- 25.5  
class(x2)
```

```
## [1] "numeric"
```

```
x3 <- as.numeric(3)  
class(x3)
```

```
## [1] "numeric"
```

El tipo integer permite almacenar números enteros:

```
x4 <- as.integer(20)  
class(x4)
```

```
## [1] "integer"
```

El tipo factor se utiliza para representar variables categóricas. Almacena niveles (categorías) en lugar de valores numéricos o texto libre. Es útil para variables como: sexo, estado civil, colores, etc.“{r}

```
sexo <- c("mujer", "hombre")
x5 <- as.factor(sexo)
class(x5)

## [1] "factor"
```

El tipo logical permite almacenar valores de verdad lógica (verdadero o TRUE, falso o FALSE):

```
x6 <- FALSE
class(x6)

## [1] "logical"
```

Algunos casos especiales: dato faltante (NA), valor infinito (Inf), valor no numérico (NaN):

```
x7 <- c(1:3,NA,5:8)
x7

## [1] 1 2 3 NA 5 6 7 8
5 / 0

## [1] Inf
```

FECHAS/HORAS

- Las fechas y horas requieren **conversión y presentación en formatos personalizados**.
- Esto hace que trabajar con tiempos sea **más complejo** que con otros tipos de datos (*numeric, character, logical*).

Código	Significado	Ejemplo
%Y	Año con 4 dígitos	2025
%y	Año con 2 dígitos	25
%m	Mes (01–12)	11
%B	Nombre completo del mes	noviembre
%b	Nombre abreviado del mes	nov
%d	Día del mes (01–31)	22

FECHAS/HORAS

- Objetos/variables HORA Y DÍA

Tipo de dato	Función base	Ejemplo	Resultado
Date	as.Date()	as.Date("2025-11-01")	"2025-11-01"
Date (actual)	Sys.Date()	Sys.Date()	"2025-11-01"
Fecha relativa	Sys.Date() - 21	Sys.Date() - 21	"2025-10-11"
POSIXct / POSIXlt	as.POSIXct()	as.POSIXct("2025-11-01 07:49:00")	"2025-11-01 07:49:00 CET"
Fecha y hora actual	Sys.time()	Sys.time()	"2025-11-01 07:49:00 CET"
Formato personalizado	format()	format(Sys.Date() - 21, "%d/%m/%Y")	"11/10/2025"

34

- Los **operadores en R** son símbolos especiales que permiten realizar cálculos, comparaciones, asignaciones y operaciones lógicas sobre objetos y datos.
- Hay diferentes tipos:
 - Aritméticos
 - Relacionales
 - Lógicos
 - De asignación
 - Otros

- Se usan para realizar cálculos matemáticos básicos

Operador	Qué hace	Ejemplo	Resultado
+	Suma	$5 + 2$	7
-	Resta	$5 - 2$	3
*	Multiplicación	$5 * 2$	10
/	División	$5 / 2$	2.5
^	Potencia	$5 ^ 2$	25
%%	Módulo (resto)	$5 \% 2$	1
%/%	División entera	$5 \%/% 2$	2

- Sirven para comparar valores y devuelven TRUE o FALSE

Operador	Qué hace	Ejemplo	Resultado
<	Menor que	5 < 2	FALSE
>	Mayor que	5 > 2	TRUE
<=	Menor o igual	5 <= 2	FALSE
>=	Mayor o igual	5 >= 2	TRUE
==	Igualdad	5 == 2	FALSE
!=	Desigualdad	5 != 2	TRUE
<	Menor que	5 < 2	FALSE

- Permiten combinar condiciones lógicas (las más utilizadas)

Operador	Qué hace	Ejemplo	Resultado
&	AND (elemento a elemento)	(5 > 2) & (3 < 4)	TRUE
	OR (elemento a elemento)	(5 > 2) (3 > 4)	TRUE
!	NOT (negación)	!(5 > 2)	FALSE

- Se utilizan para guardar valores en objetos

Operador	Qué hace	Ejemplo	Resultado
<code><-</code>	Asignar a la izquierda	<code>x <- 5</code>	<code>x</code> vale 5
<code>-></code>	Asignar a la derecha	<code>5 -> x</code>	<code>x</code> vale 5
<code>=</code>	Asignación alternativa	<code>x = 5</code>	<code>x</code> vale 5
<code><<-</code>	Asignación global	<code>x <<- 5</code>	<code>x</code> vale 5 en entorno global

OTROS – OPERADORES MISCELÁNEOS

- Otros operadores útiles para secuencias, pertenencia y funciones

Operador	Qué hace	Ejemplo	Resultado
:	Genera secuencias	1 : 5	1 2 3 4 5
%in%	Comprueba pertenencia	2 %in% c(1, 2, 3)	TRUE
%*%	Multiplicación de matrices	A %*% B	Producto matricial
::	Accede a funciones de un paquete	stats::lm	Usa lm del paquete stats

- Los **vectores** son la estructura de datos más básica en R.
- Permiten almacenar una **colección de elementos del mismo tipo** (numéricos, caracteres, lógicos, etc.).
- Son fundamentales para organizar y analizar datos biomédicos, como mediciones clínicas, resultados de laboratorio o secuencias genéticas.
- Se crean con la función `c()` (*combine*)

ACCESO A ELEMENTOS DENTRO DE UN VECTOR

- Es posible referirse a elementos concretos de un vector mediante índices o condiciones lógicas.

```
glucosa <- c(90, 110, 85, 120, 95)
pacientes <- c("Ane", "Irati", "Jon", "Ander", "Esti")

# Primer valor
glucosa[1]

## [1] 90
# Valor en posición 'a'
a <- 1
glucosa[a]

## [1] 90
# Valores de la posición 2 a 4
glucosa[2:4]

## [1] 110 85 120
```

```
# Posiciones específicas
glucosa[c(1,3,5)]
## [1] 90 85 95

# Todos menos el primero
glucosa[-1]
## [1] 110 85 120 95

# Seleccionar paciente "Jon"
pacientes[pacientes == "Jon"]
## [1] "Jon"

# Todos menos "Jon"
pacientes[pacientes != "Jon"]
## [1] "Ane"   "Irati" "Ander" "Esti"
```

1.2

Programación básica en R

ORDENACIÓN Y RANGO

VECTORES EN R MANIPULACIÓN

- Los vectores permiten aplicar funciones para ordenarlos y obtener posiciones y valores extremos

```
glucosa <- c(90, 110, 85, 120, 95)
rev(glucosa)      # Invertir orden
sort(glucosa)     # Ordenar de menor a mayor
sort(glucosa, decreasing = TRUE)    # Ordenar de mayor a menor
rank(glucosa)     # Posición relativa de cada valor
order(glucosa)    # Índices para ordenar
```
[1] 95 120 85 110 90
[1] 85 90 95 110 120
[1] 120 110 95 90 85
□ [1] 2 4 1 5 3
[1] 3 1 5 2 4
```

# 1.2

Programación básica en R

## VALORES EXTREMOS

## BÚSQUEDA

# VECTORES EN R

## MANIPULACIÓN

```
max(glucosa) # Valor máximo
min(glucosa) # Valor mínimo
range(glucosa) # Rango
...
```

```
□
[1] 120
[1] 85
[1] 85 120
```

```
Búsqueda de valores en el vector
which(glucosa == max(glucosa)) # Posición del máximo

[1] 4

w <- which(glucosa == max(glucosa)); glucosa[w] # Valor máximo

[1] 120
```

### MUESTREO

```
Muestreo
sample(glucosa, 3) # Selección aleatoria de 3 valores

[1] 85 110 95

sample(glucosa, 7, replace=TRUE) # Muestreo con reemplazo

[1] 120 85 85 95 85 120 95
```

### COMBINACIÓN DE VECTORES

```
Combinación
mes_visita <- c("Diciembre", "Enero", "Febrero", "Abril", "Julio")
union(pacientes, mes_visita) # Unión

[1] "Ane" "Irati" "Jon" "Ander" "Esti" "Diciembre"
[7] "Enero" "Febrero" "Abril" "Julio"

intersect(pacientes, mes_visita) # Intersección

character(0)

setdiff(pacientes, mes_visita) # Diferencia

[1] "Ane" "Irati" "Jon" "Ander" "Esti"
```

- Las matrices permiten almacenar datos en tablas de dos dimensiones.
  - Todos los elementos deben ser del mismo tipo.
  - Todas las filas y columnas tienen el mismo tamaño.
  - Se pueden entender como una tabla o como la concatenación de vectores de igual longitud.
  - Podemos construir matrices concatenando vectores por columnas o por filas.

### UNIÓN MEDIANTE COLUMNAS

### VERIFICAR SI ES UNA MATRIZ

```

glucosa <- c(90, 110, 85, 120, 95)
mes_visita <- c("Diciembre", "Enero", "Febrero", "Abril", "Julio")

y <- cbind(glucosa, mes_visita) # Matriz con glucosa y mes de visita
y

glucosa mes_visita
[1,] "90" "Diciembre"
[2,] "110" "Enero"
[3,] "85" "Febrero"
[4,] "120" "Abril"
[5,] "95" "Julio"

is.matrix(y) # Verifica si es matriz
[1] TRUE

```

# 1.2

### DIMENSIONES DE LA MATRIZ

```
dim(y) # Dimensiones de la matriz
```

```
[1] 5 2
```

```
y[2,1] # Elemento fila 2, columna 1
```

```
glucosa
```

```
"110"
```

```
y[,1] # Toda la primera columna (glucosa)
```

```
[1] "90" "110" "85" "120" "95"
```

```
y[1,] # Toda la primera fila
```

```
glucosa mes_visita
```

```
"90" "Diciembre"
```

### ACCEDER A ELEMENTOS CONCRETOS DE LA MATRIZ

## AÑADIR UNA COLUMNA EXTRA A LA MATRIZ “Y”

```
Añadir columna (duplicamos glucosa)
cbind(y, glucosa)
```

```
glucosa mes_visita glucosa
[1,] "90" "Diciembre" "90"
[2,] "110" "Enero" "110"
[3,] "85" "Febrero" "85"
[4,] "120" "Abril" "120"
[5,] "95" "Julio" "95"
```

## AÑADIR UNA FILA EXTRA A LA MATRIZ “Y”

```
Añadir fila (duplicamos la primera fila)
rbind(y, y[1,])
```

```
glucosa mes_visita
[1,] "90" "Diciembre"
[2,] "110" "Enero"
[3,] "85" "Febrero"
[4,] "120" "Abril"
[5,] "95" "Julio"
[6,] "90" "Diciembre"
```

- Un **factor** es un tipo especial de vector que sirve para **clasificar datos en grupos**. Son vectores de variables categóricas.
- Se utilizan para representar **categorías discretas** (por ejemplo: sexo, grupo sanguíneo, estado clínico).
- Existen diferentes tipos de factores:
  - **Factores ordenados** (con un orden lógico, como “leve < moderado < grave”)
  - **Factores no ordenados** (como tipo de sangre “A, B, O, AB” o tratamiento administrado: Placebo, Vacuna, Antibiótico).

- Además de los valores en sí, contienen la información de los diferentes
- Por defecto, **todos los factores** se crean con niveles en orden alfabético.
- Solo cuando defines un **factor ordenado** con ordered=TRUE y especificas los niveles, R respeta el orden que tú indiques.
- Con la función levels() se observan los niveles y nlevels() la cantidad de niveles.

```
gravedad <- factor(c("Leve", "Grave", "Moderado"),
 levels = c("Leve", "Moderado", "Grave"),
 ordered = TRUE)
levels(gravedad)

[1] "Leve" "Moderado" "Grave"
```

- Una **lista** en R es un objeto que consiste en una **colección ordenada de componentes**.
- Los componentes **no necesitan ser del mismo tipo**: una lista puede contener un vector numérico, un valor lógico, una matriz, un vector complejo, una cadena de texto, una función, etc.
- Diferencia con los vectores: las listas son colecciones heterogéneas, pudiendo contener objetos de distinto tipo. Los vectores son colecciones homogéneas.

## Nombres de los componentes (nombre, glucosa, meses, fumador)

## Componente de tipo carácter

## Componente de tipo vector numérico

## Componente de tipo vector de caracteres

## Componente de tipo lógico

```
paciente <- list(
 nombre = "Ane",
 glucosa = c(90, 110, 85),
 → meses = c("Diciembre", "Enero", "Febrero"),
 fumador = FALSE
)
paciente
```

```
$nombre
[1] "Ane"
##
$glucosa
[1] 90 110 85
##
$meses
[1] "Diciembre" "Enero" "Febrero"
##
$fumador
[1] FALSE
```

- Un ***data frame*** es una **lista con clase `data.frame`**.
- Sus componentes deben ser:
  - Vectores (numéricos, de caracteres o lógicos)
  - Factores
  - Matrices numéricas
  - Otras listas o *data frames*

- Restricciones:
  - Los vectores deben tener la **misma longitud**
  - Las matrices deben tener el **mismo número de filas**
- Se puede considerar como una **matriz con columnas de distinto tipo**.
- Sus filas y columnas se extraen con las mismas reglas que en las matrices

- Se construyen con la función `data.frame()`
- Una lista cuyos componentes cumplen las restricciones puede convertirse en *data frame* con: `as.data.frame(nombre_de_la_lista)`
- Se accede igual que en matrices, pero también con el operador `$`.
- Esto permite trabajar fácilmente con **variables clínicas y subconjuntos de datos**.
- En este ejemplo tenemos columna de caracteres (`mes_visita`) y columna numérica (`glucosa`)

```
glucosa <- c(90, 110, 85, 120, 95)
mes_visita <- c("Diciembre", "Enero", "Febrero", "Abril", "Julio")

pacientes <- data.frame(mes_visita, glucosa)
pacientes
```

```
mes_visita glucosa
1 Diciembre 90
2 Enero 110
3 Febrero 85
4 Abril 120
5 Julio 95
```

57

## NOTA:

La función colnames() es de gran ayuda para conocer los nombres de las columnas

```
pacientes$glucosa # Columna glucosa

[1] 90 110 85 120 95

pacientes$mes_visita # Columna meses

[1] "Diciembre" "Enero" "Febrero" "Abril" "Julio"

pacientes[1,] # Primera fila

mes_visita glucosa
1 Diciembre 90

pacientes[,2] # Segunda columna

[1] 90 110 85 120 95

pacientes[3,1] # Elemento fila 3, columna 1

[1] "Febrero"
```

- R contiene varios **conjuntos de datos predefinidos** que se utilizan en ejemplos y prácticas.
- Estos conjuntos permiten aprender y probar funciones sin necesidad de cargar datos externos.
- Uno de los más conocidos es el **conjunto de datos iris**, que es un **data frame**.
- *iris* contiene medidas de las flores de tres especies de iris (*setosa*, *versicolor*, *virginica*).

### PRERREQUISITOS

- Los conjuntos de datos normalmente se leen desde archivos externos. Para ello:
  - Definir el entorno de trabajo (*working environment*)
    - Es la carpeta donde R busca y guarda archivos por defecto
    - Se define con la función `setwd()` > `setwd("ruta/de/tu/carpeta")`
    - Se consulta con `getwd()`
    - Buena práctica: fijarlo al inicio del script
  - En caso de no especificar el Directorio de trabajo, ¡R no sabrá de donde coger los datos que quieras cargar!

60

- **R admite casi todo tipo de datos**
- Las funciones de entrada en R son **simples pero estrictas** en sus requisitos.
- Los datos pueden editarse una vez cargados a R.

No se recomienda hacer modificaciones en el archive original – ¡Las buenas prácticas indican que NO SE DEBE MODIFICAR EL DOCUMENTO FUENTE!

61

- La función `read.table()` permite leer directamente un archivo y convertirlo en un *data.frame*.
- Con la función `data(iris)` es posible acceder a *datasets* incorporados en R.

- Permite trabajar con datos de **múltiples formatos y fuentes** de manera **flexible y rápida**.

| Fuente de datos    | Funciones/Paquetes en R                                                  |
|--------------------|--------------------------------------------------------------------------|
| Archivos CSV / TXT | <code>read.csv()</code> , <code>read.table()</code>                      |
| Archivos Excel     | <code>readxl::read_excel()</code>                                        |
| SPSS / Stata / SAS | <code>foreign</code> , <code>haven</code>                                |
| Bases de datos     | <code>DBI</code> , <code>RMySQL</code> , <code>RPostgres</code>          |
| Web / APIs         | <code>read.csv("url")</code> , <code>jsonlite</code> , <code>httr</code> |

- Permite exportar los datos trabajos en diferentes formados, de manera flexible y rápida.

| Fuente de datos           | Funciones/Paquetes en R                                                                                                                              |
|---------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Archivos CSV / TXT</b> | <code>write.csv()</code> , <code>write.table()</code>                                                                                                |
| <b>Archivos Excel</b>     | <code>openxlsx::write.xlsx()</code> ,<br><code>writexl::write_xlsx()</code>                                                                          |
| <b>SPSS / Stata / SAS</b> | <code>haven::write_sav()</code> (SPSS),<br><code>haven::write_dta()</code> (Stata),<br><code>haven::write_sas()</code> (SAS)                         |
| <b>Bases de datos</b>     | <code>DBI::dbWriteTable()</code> (junto con paquetes como <code>RSQLite</code> , <code>RPostgres</code> , <code>RMariaDB</code> )                    |
| <b>Web / APIs</b>         | Generalmente se exporta con<br><code>jsonlite::write_json()</code> o<br><code>xml2::write_xml()</code> para guardar respuestas en formato JSON o XML |

# *EJERCICIOS PRÁCTICOS – Módulos 1 & 2*



Osakidetza



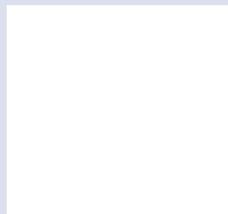
# DESCANSO



Osakidetza



# Estadística descriptiva en R



## ÍNDICE

- 1.1 Introducción a R y RStudio
- 1.2 Programación básica en R
- 1.3 Estadística descriptiva en R**
- 1.4 Manipulación y reestructuración de datos

# SESIÓN 1: FUNDAMENTOS DE PROGRAMACIÓN EN R

## 1.1 Introducción a R y RStudio

## 1.2 Programación básica en R

## 1.3 Estadística descriptiva en R

- Resumen de datos y variables
- Tablas de frecuencias
- Visualización de los datos en R. Gráficas base
- Guardar y exportar gráficos en R

## 1.4 Manipulación y reestructuración de datos

- Es una rama de la estadística que **resume y describe las características principales** de un conjunto de datos.
- Permite obtener una **visión general rápida** antes de aplicar análisis más complejos.

68

La estadística descriptiva en R es el primer paso para **conocer tus datos**, detectar patrones y preparar el terreno para análisis más profundos.

## ESTADÍSTICA DESCRIPTIVA OBJETIVOS PRINCIPALES

**Identificar tendencias centrales** (media, mediana...)

Evaluar la **dispersión** (rango, varianza, desviación estándar)

Explorar la **forma de los datos** (asimetría, curtosis)

Facilitar la **interpretación inicial** de la información

- Algunas funciones son muy útiles para una exploración rápida y entender la forma de los datos:
  - `summary(df)` → resumen estadístico de cada variable (mínimo, máximo, media, mediana, cuartiles).
  - `str(df)` → estructura del objeto: número de filas, columnas y tipo de datos.
  - `head(df)` → primeras filas del *dataset*
  - `tail(df)` → últimas filas del *dataset*

```
```{r}
str(diabetes)
```
```

```
'data.frame': 403 obs. of 19 variables:
 $ id : int 1000 1001 1002 1003 1005 1008 1011 1015 1016 1022 ...
 $ chol : int 203 165 228 78 249 248 195 227 177 263 ...
 $ stab.glu: int 82 97 92 93 90 94 92 75 87 89 ...
 $ hdl : int 56 24 37 12 28 69 41 44 49 40 ...
 $ ratio : num 3.6 6.9 6.2 6.5 8.9 ...
 $ glyhb : num 4.31 4.44 4.64 4.63 7.72 ...
 $ location: chr "Buckingham" "Buckingham" "Buckingham" "Buckingham" ...
 $ age : int 46 29 58 67 64 34 30 37 45 55 ...
 $ gender : chr "female" "female" "female" "male" ...
 $ height : int 62 64 61 67 68 71 69 59 69 63 ...
 $ weight : int 121 218 256 119 183 190 191 170 166 202 ...
 $ frame : chr "medium" "large" "large" "large" ...
 $ bp.1s : int 118 112 190 110 138 132 161 NA 160 108 ...
 $ bp.1d : int 59 68 92 50 80 86 112 NA 80 72 ...
 $ bp.2s : int NA NA 185 NA NA NA 161 NA 128 NA ...
 $ bp.2d : int NA NA 92 NA NA NA 112 NA 86 NA ...
 $ waist : int 29 46 49 33 44 36 46 34 34 45 ...
 $ hip : int 38 48 57 38 41 42 49 39 40 50 ...
 $ time.ppn: int 720 360 180 480 300 195 720 1020 300 240 ...
```

# 1.3

## Estadística descriptiva

# RESUMEN DE DATOS

```
```{r}
summary(diabetes)
```

```

|                  | <b>id</b>     | <b>chol</b>   | <b>stab.glu</b> | <b>hdl</b>       | <b>ratio</b>   | <b>glyhb</b>   | <b>location</b>  | <b>age</b>     |
|------------------|---------------|---------------|-----------------|------------------|----------------|----------------|------------------|----------------|
| Min. :           | 1000          | Min. : 78.0   | Min. : 48.0     | Min. : 12.00     | Min. : 1.500   | Min. : 2.68    | Length:403       | Min. :19.00    |
| 1st Qu.:         | 4792          | 1st Qu.:179.0 | 1st Qu.: 81.0   | 1st Qu.: 38.00   | 1st Qu.: 3.200 | 1st Qu.: 4.38  | Class :character | 1st Qu.:34.00  |
| Median :         | 15766         | Median :204.0 | Median : 89.0   | Median : 46.00   | Median : 4.200 | Median : 4.84  | Mode :character  | Median :45.00  |
| Mean :           | 15978         | Mean :207.8   | Mean :106.7     | Mean : 50.45     | Mean : 4.522   | Mean : 5.59    |                  | Mean :46.85    |
| 3rd Qu.:         | 20336         | 3rd Qu.:230.0 | 3rd Qu.:106.0   | 3rd Qu.: 59.00   | 3rd Qu.: 5.400 | 3rd Qu.: 5.60  |                  | 3rd Qu.:60.00  |
| Max. :           | 41756         | Max. :443.0   | Max. :385.0     | Max. :120.00     | Max. :19.300   | Max. :16.11    |                  | Max. :92.00    |
|                  | NA's :1       |               | NA's :1         | NA's :1          | NA's :1        | NA's :13       |                  |                |
|                  | <b>gender</b> | <b>height</b> | <b>weight</b>   | <b>frame</b>     | <b>bp.1s</b>   | <b>bp.1d</b>   | <b>bp.2s</b>     | <b>bp.2d</b>   |
| Length:403       |               | Min. :52.00   | Min. : 99.0     | Length:403       | Min. : 90.0    | Min. : 48.00   | Min. :110.0      | Min. : 60.00   |
| Class :character |               | 1st Qu.:63.00 | 1st Qu.:151.0   | Class :character | 1st Qu.:121.2  | 1st Qu.: 75.00 | 1st Qu.:138.0    | 1st Qu.: 84.00 |
| Mode :character  |               | Median :66.00 | Median :172.5   | Mode :character  | Median :136.0  | Median : 82.00 | Median :149.0    | Median : 92.00 |
|                  |               | Mean :66.02   | Mean :177.6     |                  | Mean :136.9    | Mean : 83.32   | Mean :152.4      | Mean : 92.52   |
|                  |               | 3rd Qu.:69.00 | 3rd Qu.:200.0   |                  | 3rd Qu.:146.8  | 3rd Qu.: 90.00 | 3rd Qu.:161.0    | 3rd Qu.:100.00 |
|                  |               | Max. :76.00   | Max. :325.0     |                  | Max. :250.0    | Max. :124.00   | Max. :238.0      | Max. :124.00   |
|                  |               | NA's :5       | NA's :1         |                  | NA's :5        | NA's :5        | NA's :262        | NA's :262      |
|                  | <b>waist</b>  | <b>hip</b>    | <b>time.ppn</b> |                  |                |                |                  |                |
| Min. :           | 26.0          | Min. :30.00   | Min. : 5.0      |                  |                |                |                  |                |
| 1st Qu.:         | 33.0          | 1st Qu.:39.00 | 1st Qu.: 90.0   |                  |                |                |                  |                |
| Median :         | 37.0          | Median :42.00 | Median : 240.0  |                  |                |                |                  |                |
| Mean :           | 37.9          | Mean :43.04   | Mean : 341.2    |                  |                |                |                  |                |
| 3rd Qu.:         | 41.0          | 3rd Qu.:46.00 | 3rd Qu.: 517.5  |                  |                |                |                  |                |
| Max. :           | 56.0          | Max. :64.00   | Max. :1560.0    |                  |                |                |                  |                |
| NA's :           | 2             | NA's :2       | NA's :3         |                  |                |                |                  |                |

72

# 1 3

## Estadística descriptiva

# RESUMEN DE DATOS

```{r}  
tail(diabetes)
```

Description: df [6 x 19]

	<b>id</b> <int>	<b>chol</b> <int>	<b>stab.glu</b> <int>	<b>hdl</b> <int>	<b>ratio</b> <dbl>	<b>glyhb</b> <dbl>	<b>location</b> <chr>	<b>age</b> <int>	<b>gender</b> <chr>
398	41503	301	90	118	2.6	4.28	Louisa	89	female
399	41506	296	369	46	6.4	16.11	Louisa	53	male
400	41507	284	89	54	5.3	4.39	Louisa	51	female
401	41510	194	269	38	5.1	13.63	Louisa	29	female
402	41752	199	76	52	3.8	4.49	Louisa	41	female
403	41756	159	88	79	2.0	NA	Louisa	68	female

6 rows | 1-10 of 19 columns

```{r}  
head(diabetes)
```

Description: df [6 x 19]

	<b>id</b> <int>	<b>chol</b> <int>	<b>stab.glu</b> <int>	<b>hdl</b> <int>	<b>ratio</b> <dbl>	<b>glyhb</b> <dbl>	<b>location</b> <chr>	<b>age</b> <int>	<b>gender</b> <chr>
1	1000	203	82	56	3.6	4.31	Buckingham	46	female
2	1001	165	97	24	6.9	4.44	Buckingham	29	female
3	1002	228	92	37	6.2	4.64	Buckingham	58	female
4	1003	78	93	12	6.5	4.63	Buckingham	67	male
5	1005	249	90	28	8.9	7.72	Buckingham	64	male
6	1008	248	94	69	3.6	4.81	Buckingham	34	male

6 rows | 1-10 of 19 columns

- Las **tablas de frecuencias** son una forma de organizar datos mostrando cuántas veces aparece cada valor o categoría en un conjunto.
- Las tablas de frecuencias permiten:
  - Resumir la distribución de categorías
  - Comparar valores absolutos y relativos
  - **Facilitar la interpretación** de datos cualitativos
- En R la forma más sencilla es mediante la función `table()`

# 1.3

Estadística descriptiva

## TABLAS DE FRECUENCIAS

### EJEMPLOS



```
```{r}
table(diabetes$location)
```

```

|            |        |
|------------|--------|
| Buckingham | Louisa |
| 200        | 203    |

```
```{r}
table(diabetes$location, diabetes$gender)
```

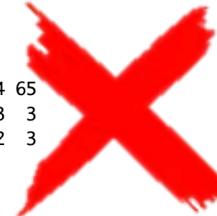
```

|            | female | male |
|------------|--------|------|
| Buckingham | 114    | 86   |
| Louisa     | 120    | 83   |

75

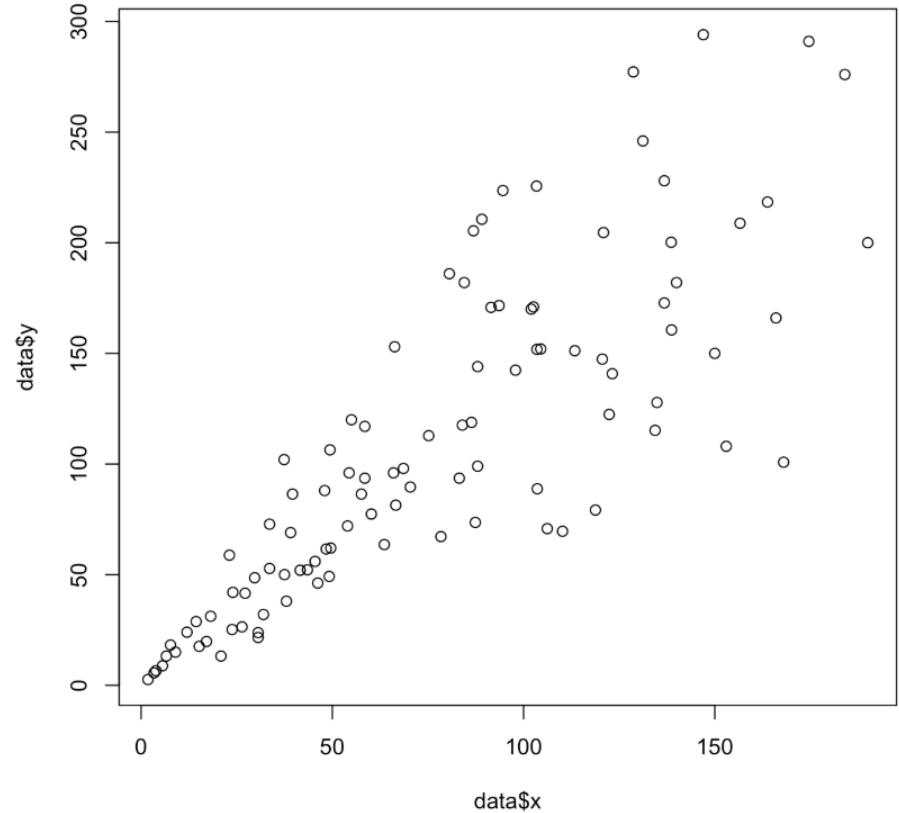
```
> table(diabetes$location, diabetes$age)
```

|            | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 | 64 | 65 |
|------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| Buckingham | 1  | 5  | 2  | 3  | 5  | 1  | 3  | 3  | 4  | 4  | 4  | 4  | 8  | 2  | 2  | 3  | 1  | 7  | 4  | 5  | 0  | 6  | 6  | 3  | 6  | 3  | 5  | 3  | 5  | 5  | 3  | 4  | 1  | 2  | 6  | 2  | 2  | 3  | 4  | 2  | 5  | 5  | 3  | 3  |    |    |    |
| Louisa     | 1  | 5  | 4  | 2  | 2  | 0  | 1  | 3  | 5  | 5  | 3  | 5  | 2  | 2  | 4  | 4  | 4  | 6  | 7  | 6  | 2  | 10 | 7  | 4  | 7  | 4  | 4  | 1  | 2  | 2  | 6  | 5  | 4  | 6  | 4  | 1  | 2  | 2  | 4  | 4  | 8  | 4  | 0  | 6  | 2  | 3  |    |
|            | 66 | 67 | 68 | 69 | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 78 | 79 | 80 | 81 | 82 | 83 | 84 | 89 | 91 | 92 |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| Buckingham | 3  | 2  | 3  | 1  | 4  | 3  | 2  | 1  | 1  | 1  | 5  | 1  | 2  | 0  | 0  | 1  | 0  | 0  | 0  | 1  | 1  |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| Louisa     | 4  | 2  | 3  | 0  | 0  | 1  | 2  | 1  | 2  | 1  | 0  | 3  | 0  | 2  | 1  | 1  | 1  | 1  | 1  | 1  | 0  | 0  |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |



## GRÁFICO DE DISPERSIÓN – SCATTER PLOT

- USO:** analizar relación entre dos variables numéricas
- FUNCTION EN R:** `plot(x, y)`
- CARACTERÍSTICAS CLAVE**
  - Cada punto representa una observación
  - Útil para detectar correlaciones o *outliers*
  - Puede incluir colores o formas para variables adicionales

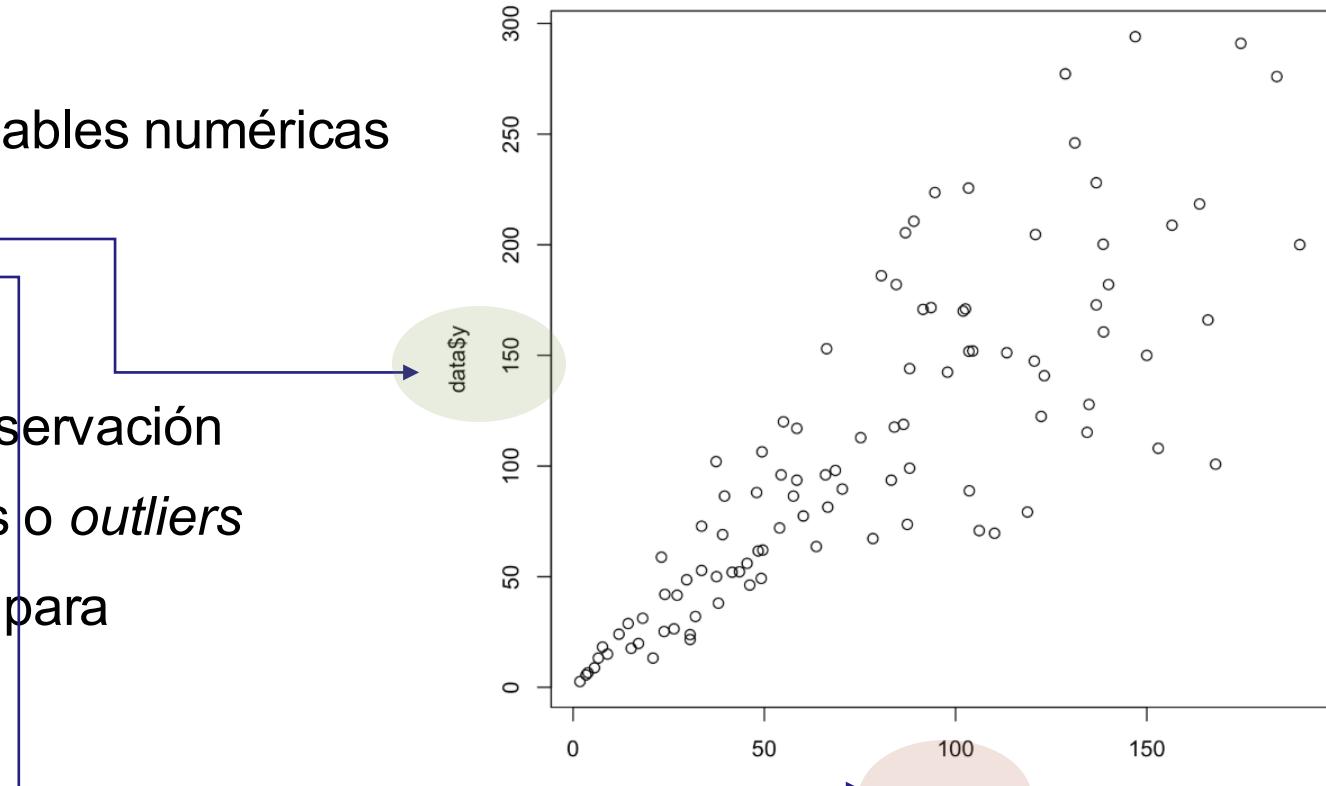


6

Fuente imagen: <https://r-graph-gallery.com/13-scatter-plot.html>

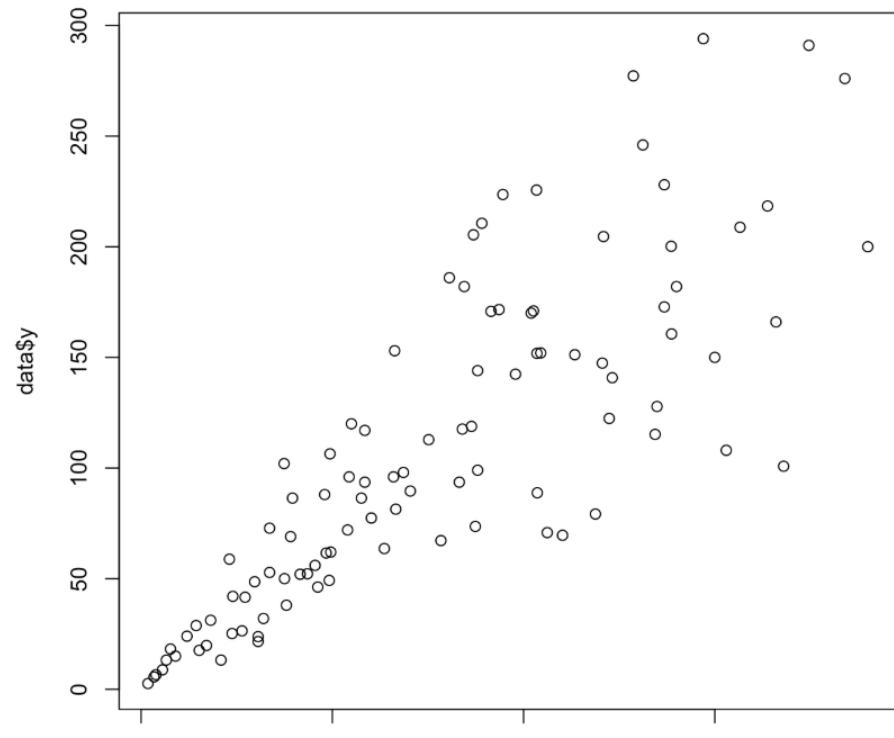
## GRÁFICO DE DISPERSIÓN – SCATTER PLOT

- USO:** analizar relación entre dos variables numéricas
- FUNCTION EN R** `plot(x, y)`
- CARACTERÍSTICAS CLAVE**
  - Cada punto representa una observación
  - Útil para detectar correlaciones o *outliers*
  - Puede incluir colores o formas para variables adicionales

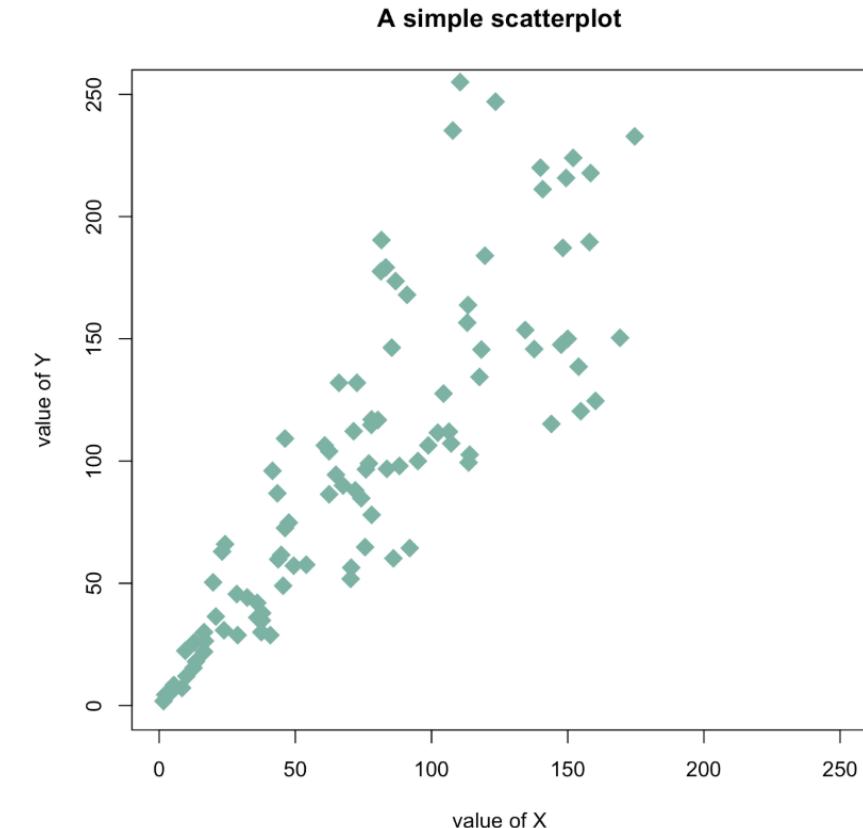
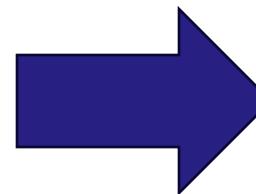


Fuente imagen: <https://r-graph-gallery.com/13-scatter-plot.html>

## GRÁFICO DE DISPERSIÓN – SCATTER PLOT

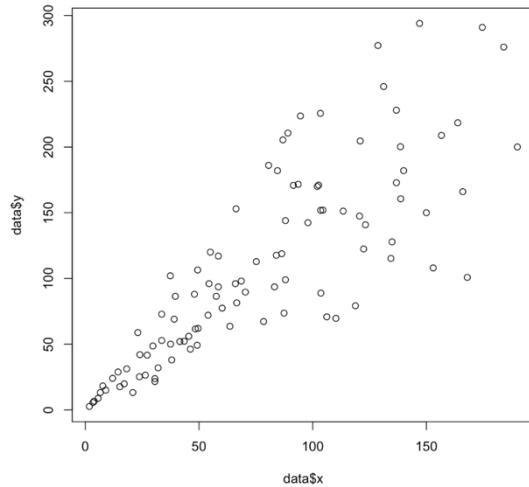


Fuente imagen: <https://r-graph-gallery.com/13-scatter-plot.html>



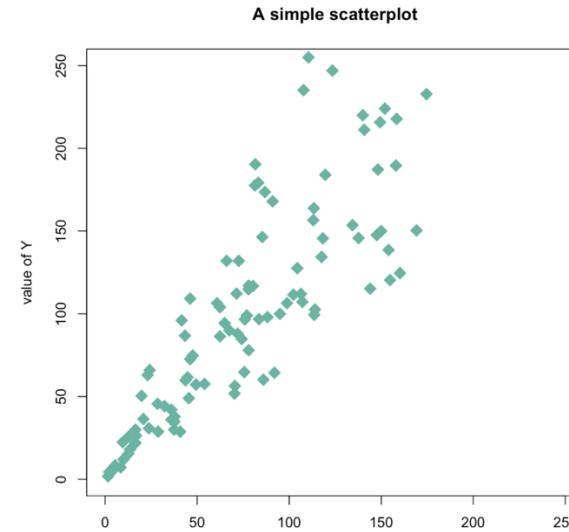
Fuente imagen: <https://r-graph-gallery.com/13-scatter-plot.html>

## GRÁFICO DE DISPERSIÓN – SCATTER PLOT



Fuente imagen: <https://r-graph-gallery.com/13-scatter-plot.html>

```
> plot(data$x, data$y)
```

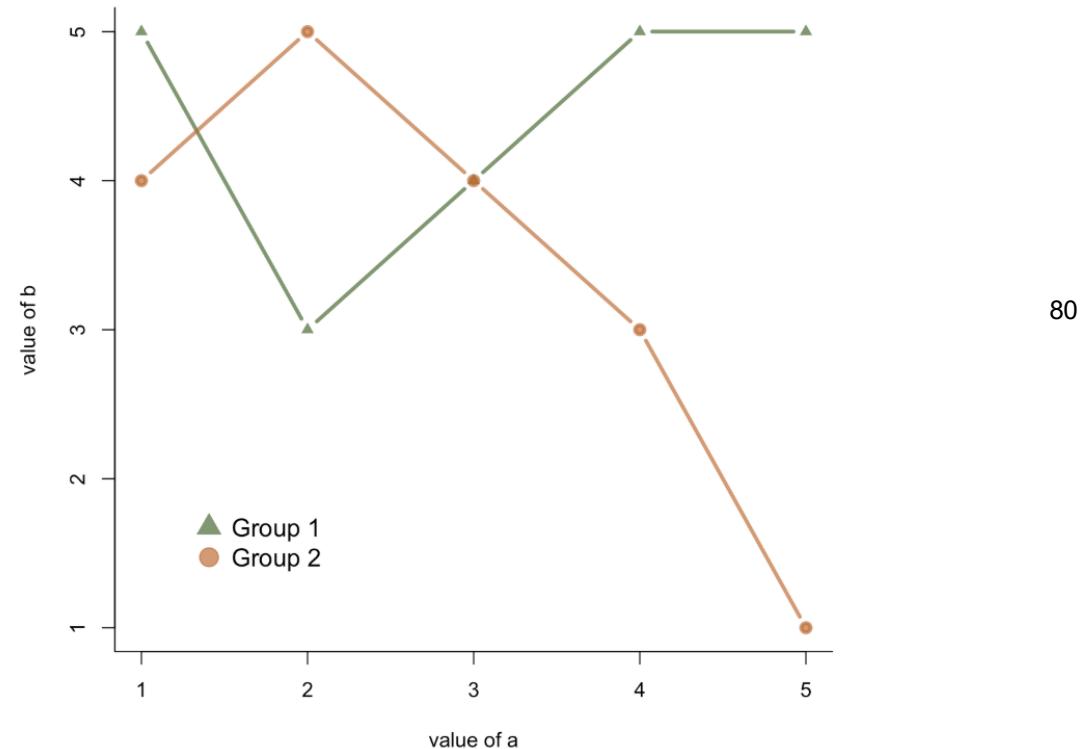


Fuente imagen: <https://r-graph-gallery.com/13-scatter-plot.html>

```
> plot(data$x, data$y,
 xlim = c(0, 250), # límite eje X
 ylim = c(0, 250), # límite eje Y
 pch = 18, # forma de rombos
 main = "Simple Scatterplot") # título
```

## GRÁFICO DE LÍNEAS – LINE PLOT

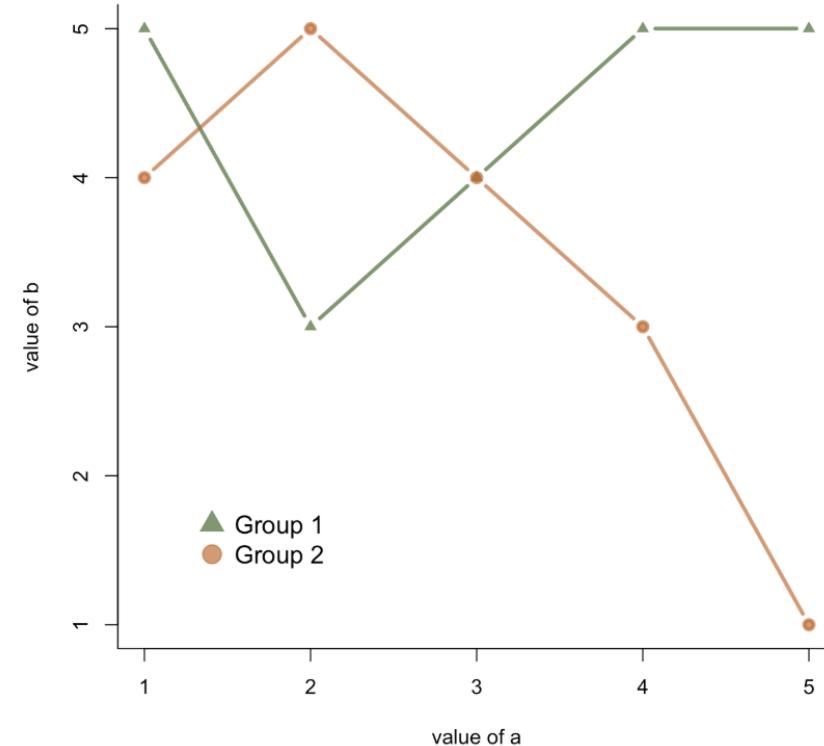
- USO:** mostrar tendencias a lo largo del tiempo
- FUNCTION EN R:** `plot(x, y, type = "l")`
- CARACTERÍSTICAS CLAVE**
  - Ideal para series temporales
  - Conecta puntos con líneas
  - Permite visualizar patrones y evolución



Fuente imagen: <https://r-graph-gallery.com/119-add-a-legend-to-a-plot.html>

## GRÁFICO DE LÍNEAS – LINE PLOT

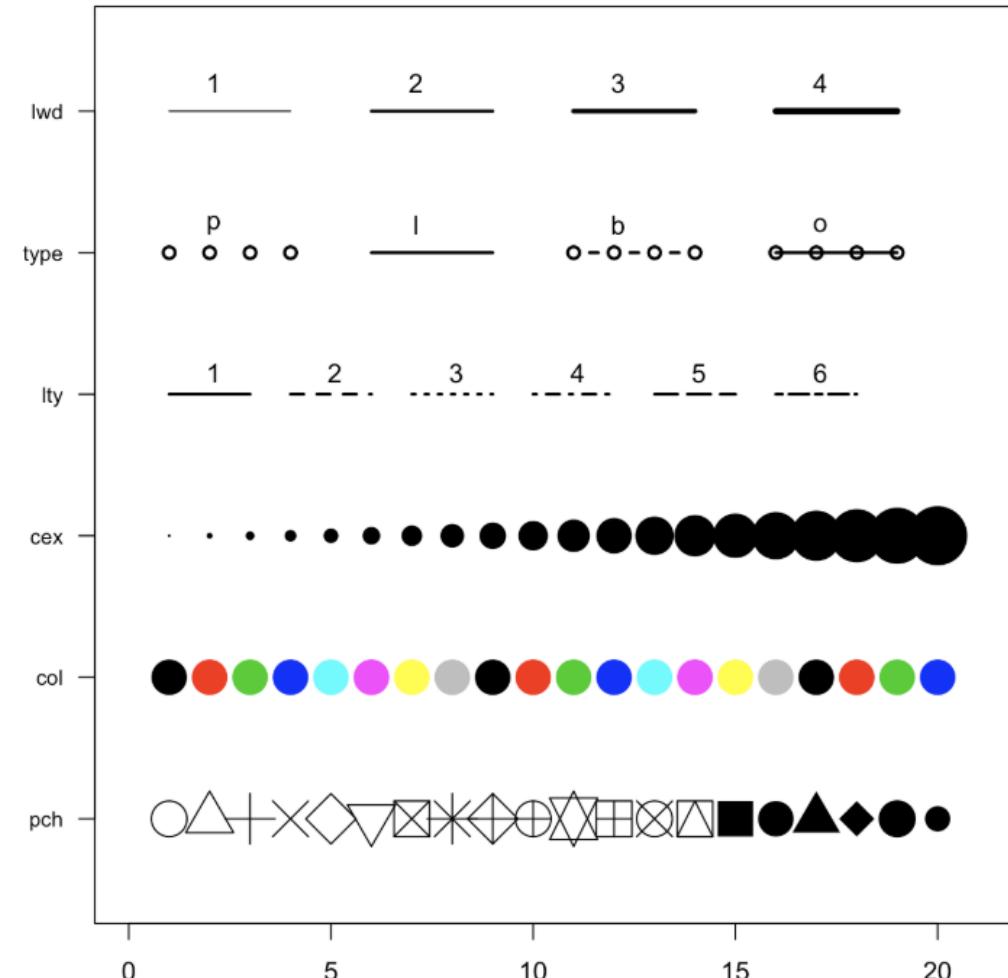
```
> # Create data:
> a=c(1:5)
> b=c(5,3,4,5,5)
> c=c(4,5,4,3,1)
>
> # Make a basic graph
> plot(a,b , type="b" , bty="l" , xlab="value of a" , ylab="value of b" , col=rgb(0.2,0.4,0.1,0.7) , lwd=3 , pch=17 , ylim=c(1,5))
> lines(a,c , col=rgb(0.8,0.4,0.1,0.7) , lwd=3 , pch=19 , type="b")
>
> # Add a legend
> legend("bottomleft",
+ legend = c("Group 1", "Group 2"),
+ col = c(rgb(0.2,0.4,0.1,0.7),
+ rgb(0.8,0.4,0.1,0.7)),
+ pch = c(17,19),
+ bty = "n",
+ pt.cex = 2,
+ cex = 1.2,
+ text.col = "black",
+ horiz = F ,
+ inset = c(0.1, 0.1))
```



Fuente imagen: <https://r-graph-gallery.com/119-add-a-legend-to-a-plot.html>

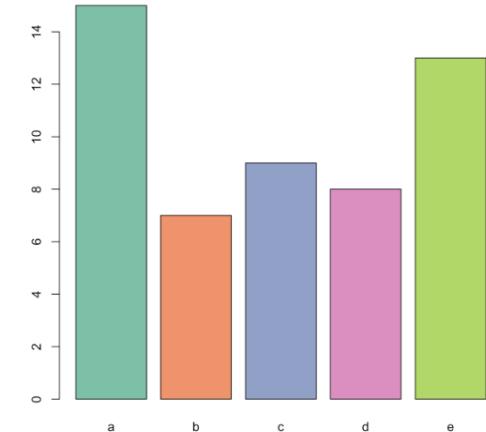
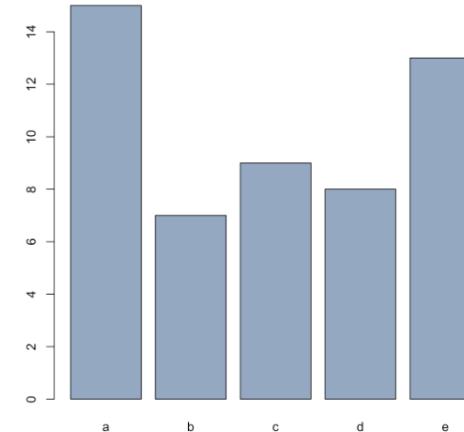
## PERSONALIZACIÓN DE LOS GRÁFICOS DE LÍNEAS

- **Parámetros principales:**
  - `lwd` → grosor de la línea
  - `type` → conexión entre los puntos
  - `lty` → tipo de línea
  - `cex` → tamaño de los puntos o círculos
  - `col` → colores
  - `pch` → forma de los marcadores (símbolos)



Fuente imagen: <https://r-graph-gallery.com/6-graph-parameters-reminder.html>

- **USO:** comparar frecuencias o valores de categorías
- **FUNCIÓN EN R:** barplot()
- **CARACTERÍSTICAS CLAVE**
  - Representa datos categóricos
  - Altura de las barras = valor o la frecuencia
  - Fácil de interpretar y comparar

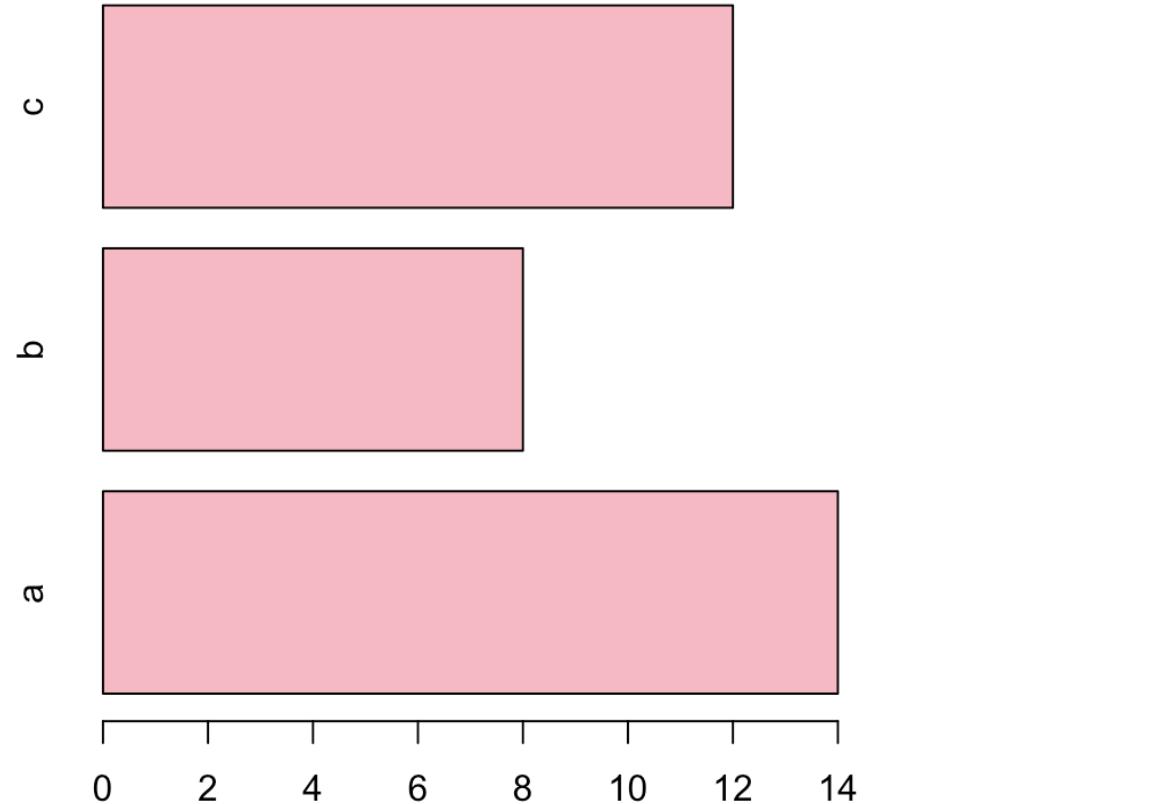


83

Fuente imagen:<https://r-graph-gallery.com/209-the-options-of-barplot.html#color>

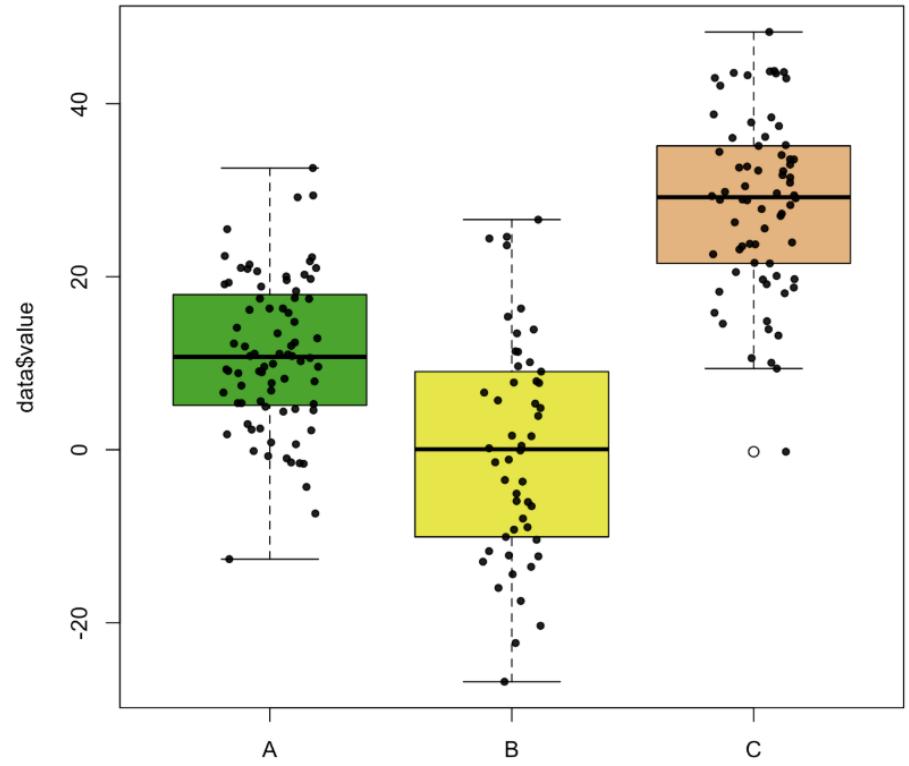
## VISUALIZACIÓN DE DATOS GRÁFICO DE BARRAS

```
> # create dummy data
> data <- data.frame(
+ name=letters[1:3],
+ value=sample(seq(2,25),3)
+)
>
> # Uniform color
> barplot(height=data$value, names=data$name,
+ col="lightpink",
+ horiz=T)
```



## VISUALIZACIÓN DE DATOS GRÁFICO DE CAJAS

- **USO:** analizar la distribución de una variable numérica
- **FUNCTION EN R:** `boxplot()`
- **CARACTERÍSTICAS CLAVE**
  - Muestra la mediana y los cuartiles.
  - Permite identificar outliers (valores atípicos)
  - Resume la dispersión y tendencia central en un solo gráfico



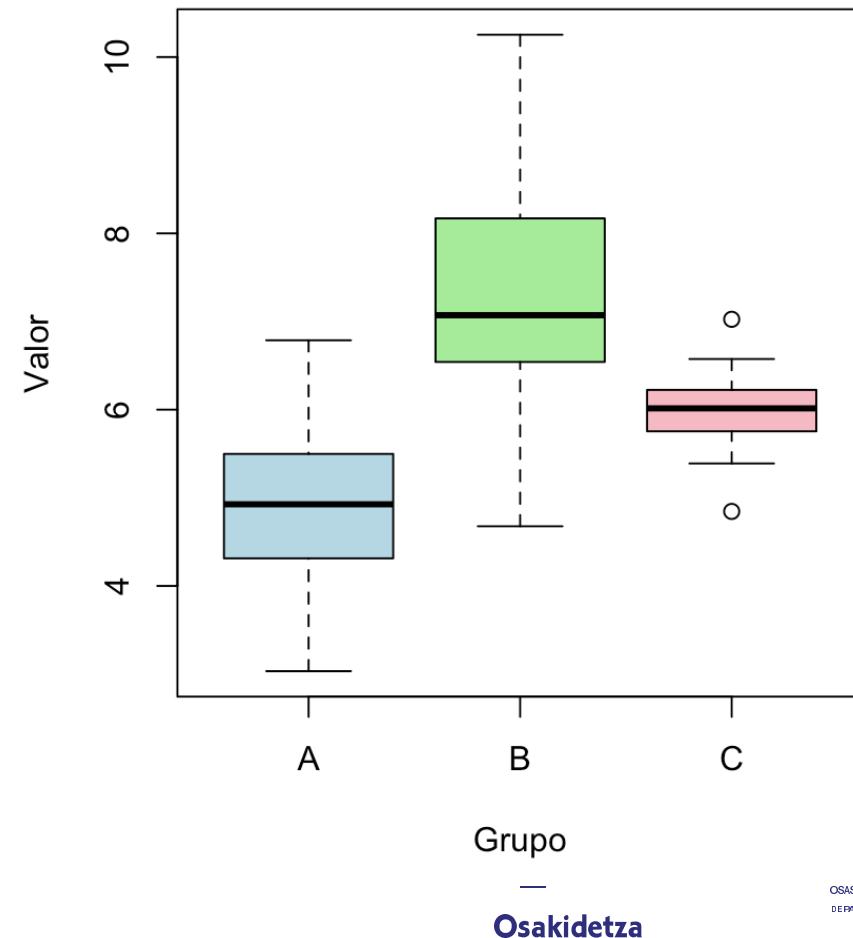
Fuente imagen: <https://r-graph-gallery.com/96-boxplot-with-jitter.html>

# VISUALIZACIÓN DE DATOS

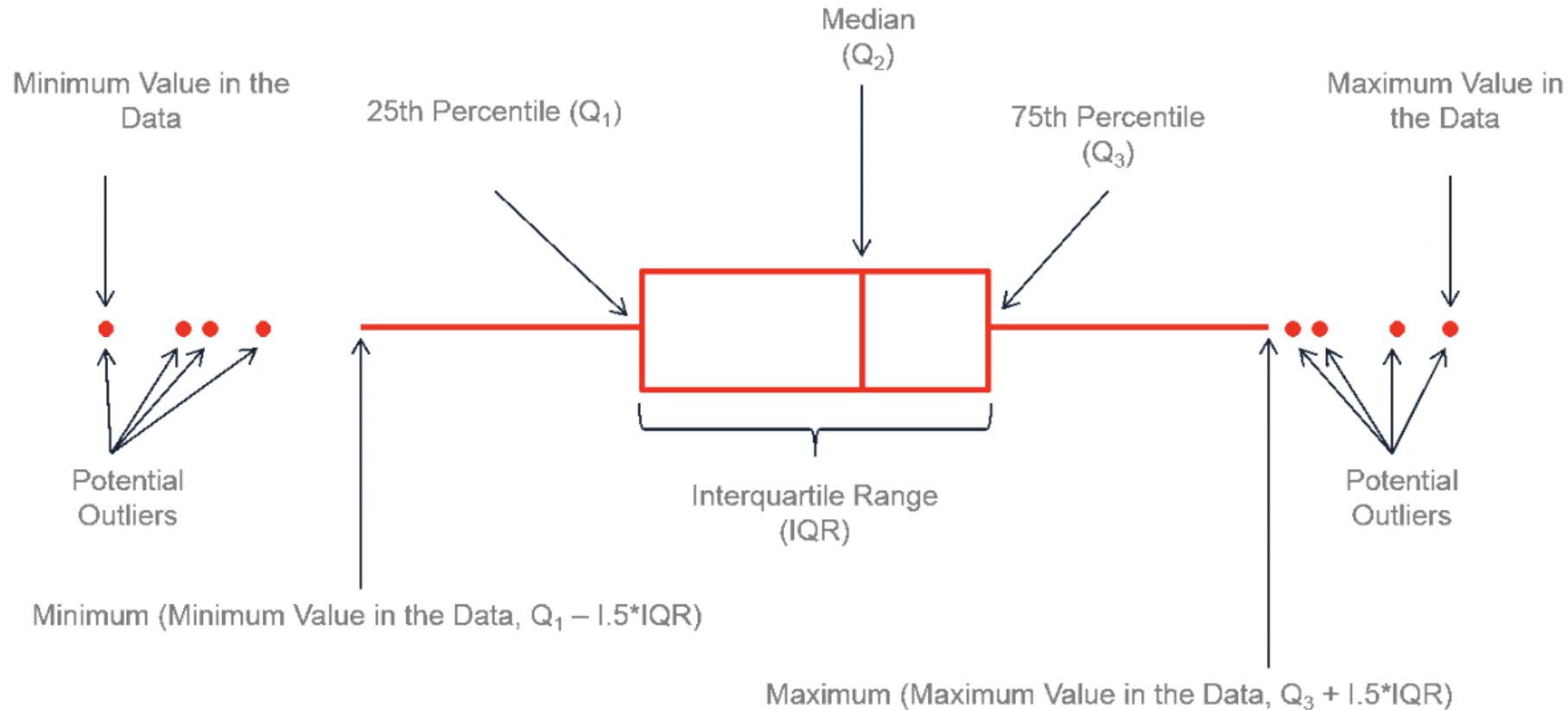
## GRÁFICO DE CAJAS

```
> # Simular datos con varias observaciones por grupo
> set.seed(123)
> data <- data.frame(
+ grupo = rep(c("A","B","C"), each = 30),
+ valor = c(rnorm(30, mean = 5, sd = 1),
+ rnorm(30, mean = 7, sd = 1.5),
+ rnorm(30, mean = 6, sd = 0.5)))
>
> # Boxplot con colores y etiquetas
> boxplot(valor ~ grupo, data = data,
+ col = c("lightblue","lightgreen","lightpink"),
+ main = "Distribución de valores por grupo",
+ xlab = "Grupo",
+ ylab = "Valor")
```

Distribución de valores por grupo



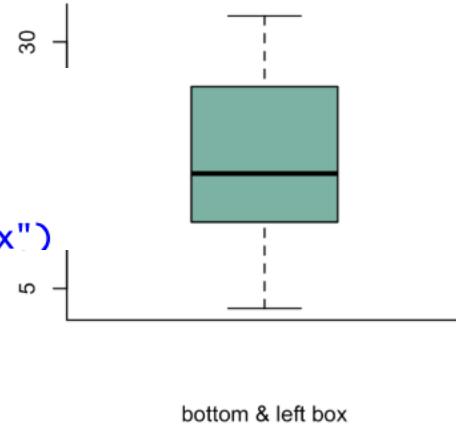
## VISUALIZACIÓN DE DATOS GRÁFICO DE CAJAS



# 1 3

Estadística descriptiva

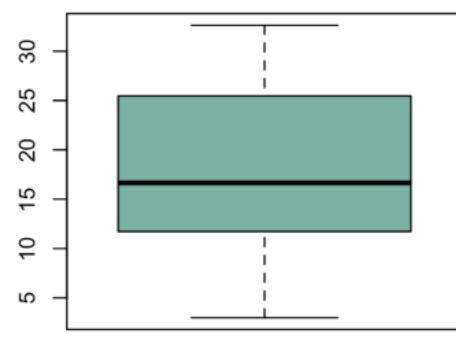
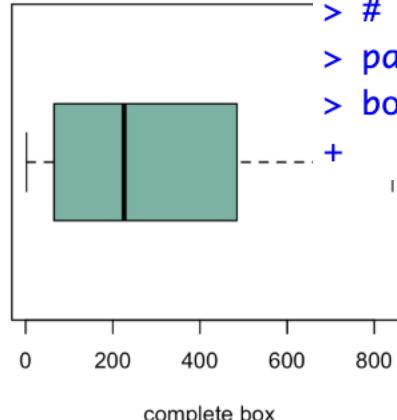
```
> # First graph
> par(bty="l")
> boxplot(a , col="#69b3a2" ,
+ xlab="bottom & left box")
```



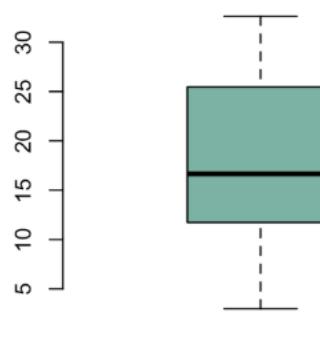
```
> # Cut the screen in 4 parts
> par(mfrow=c(2,2))
```

## VISUALIZACIÓN DE DATOS GRÁFICO DE CAJAS

```
> # Second
> par(bty="o")
> boxplot(b , col="#69b3a2" ,
+ xlab="complete box", horizontal=TRUE)
```



```
> # Third
> par(bty="c")
> boxplot(a , col="#69b3a2" ,
+ xlab="up & bottom & left box", width=0.5)
```



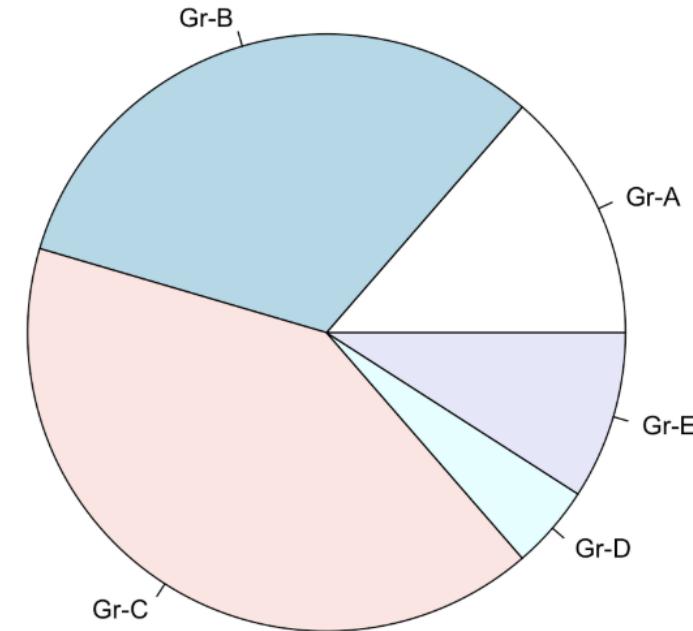
```
> # Fourth
> par(bty="n")
> boxplot(a , col="#69b3a2" ,
+ xlab="no box")
```

[ph-gallery.com/73-box-style-with-the-bty-function.html](http://ph-gallery.com/73-box-style-with-the-bty-function.html)

## VISUALIZACIÓN DE DATOS

### GRÁFICO DE SECTORES – PIE CHART

- **USO:** mostrar proporciones de un conjunto
- **FUNCIÓN EN R:** `pie()`
- **CARACTERÍSTICAS CLAVE**
  - Representa porcentajes de categorías
  - Visualiza la composición total
  - Menos preciso que el *barplot*, pero intuitivo



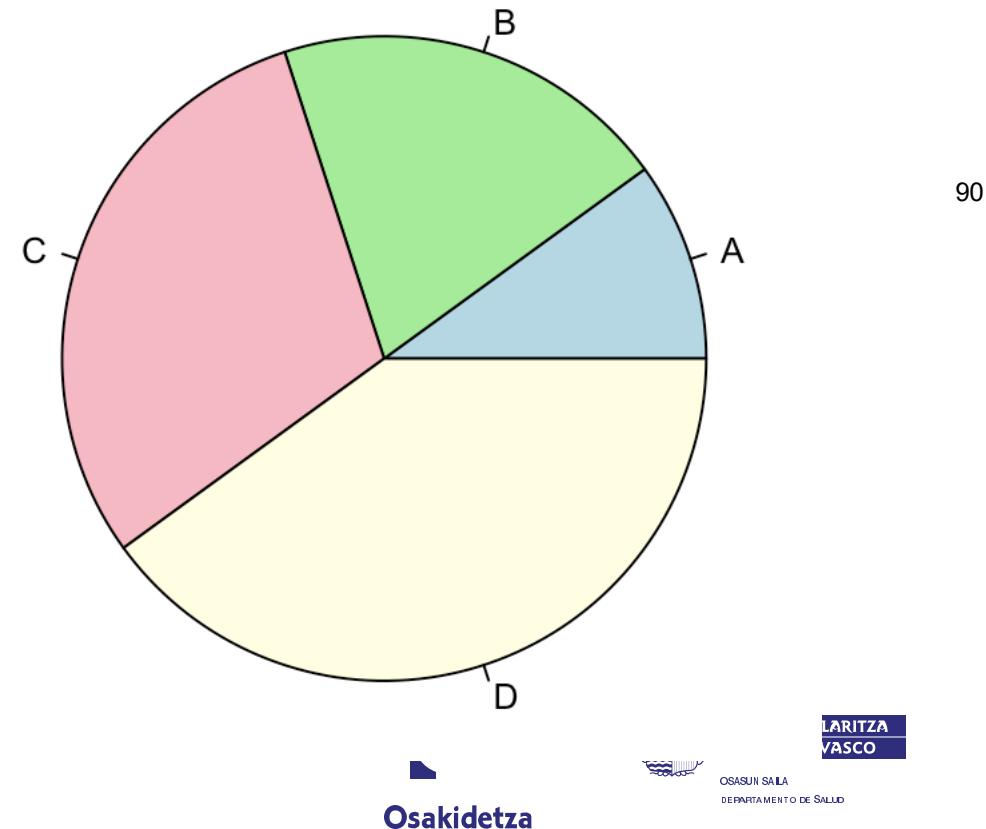
89

Fuente imagen: <https://r-graph-gallery.com/131-pie-plot-with-r.html>

## GRÁFICO DE SECTORES – PIE CHART

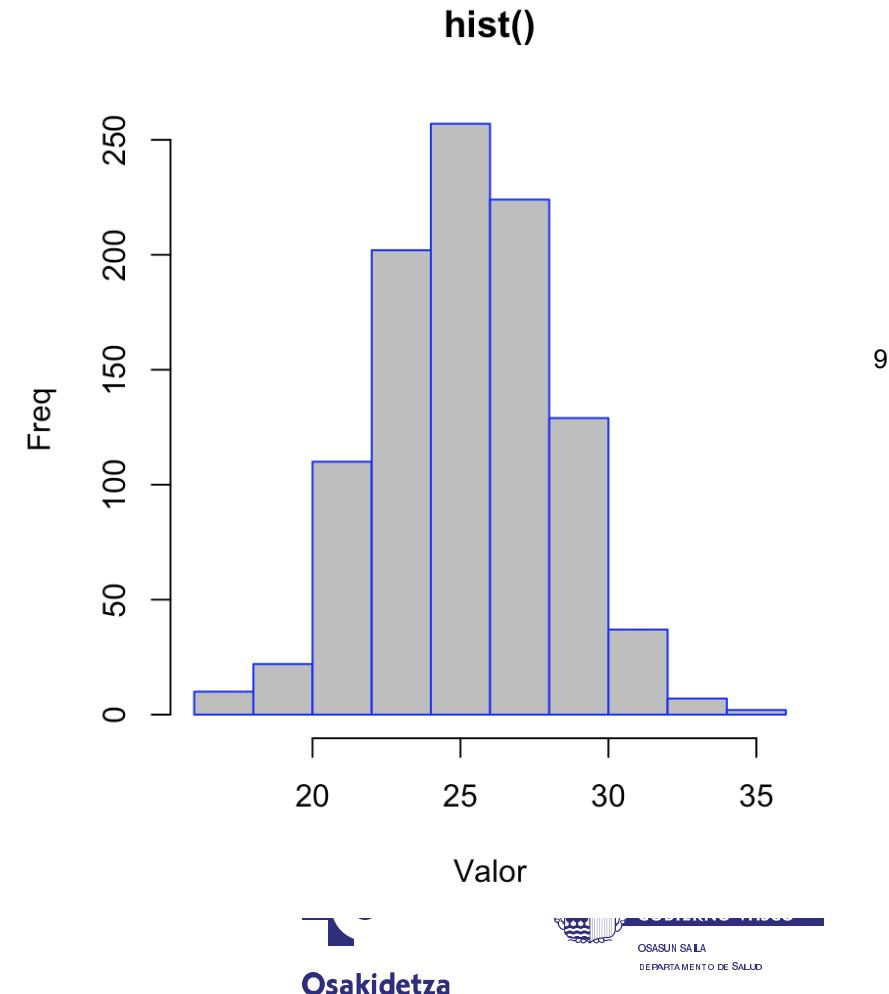
```
> valores <- c(10, 20, 30, 40)
> categorias <- c("A", "B", "C", "D")
>
> # Colores para cada categoría
> colores <- c("lightblue", "lightgreen", "lightpink", "lightyellow")
>
> # Grafico pie
> pie(valores,
+ labels = categorias,
+ col = colores,
+ main = "Distribución por categorías")
```

Distribución por categorías



## VISUALIZACIÓN DE DATOS HISTOGRAMA

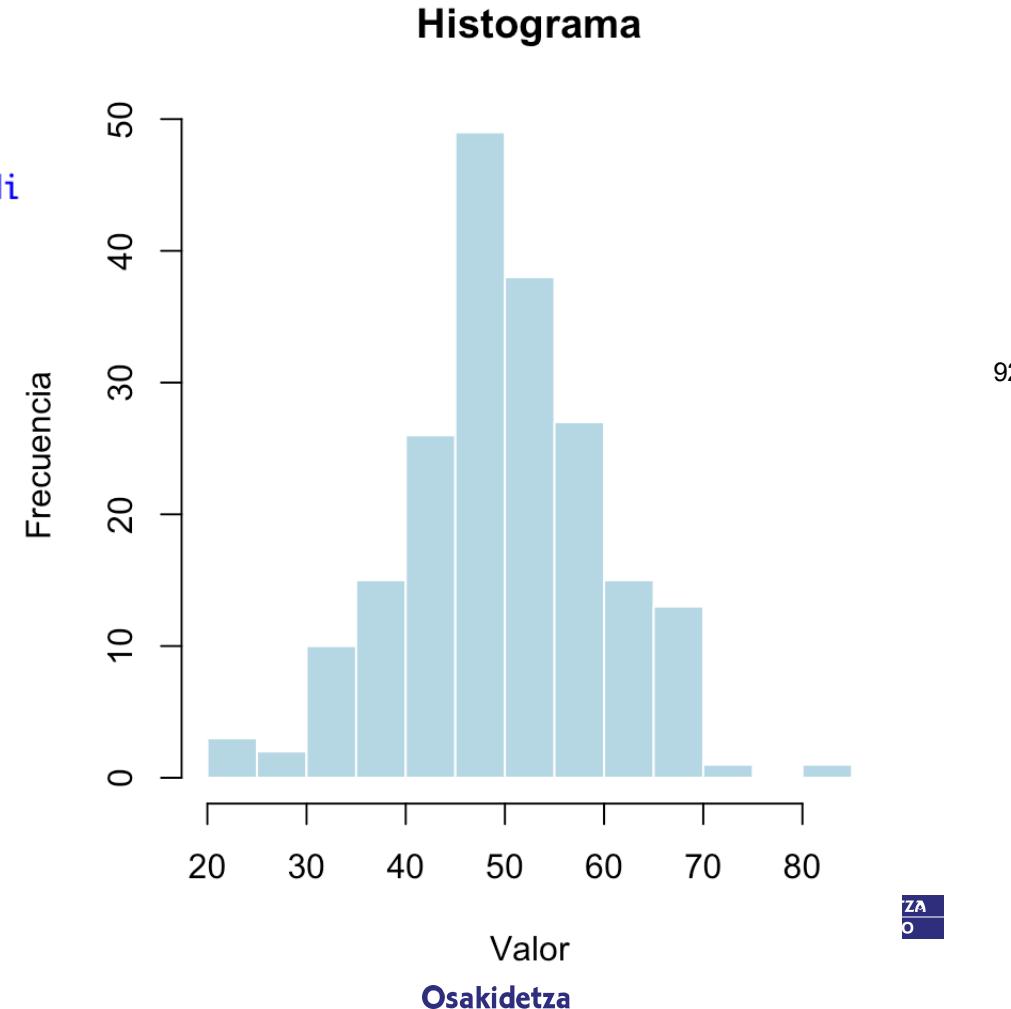
- **USO:** mostrar la distribución de una variable numérica
- **FUNCTION EN R:** `hist()`
- **CARACTERÍSTICAS CLAVE**
  - Divide los datos en intervalos
  - Altura de las barras = frecuencia en cada intervalo
  - Útil para ver la forma de la distribución



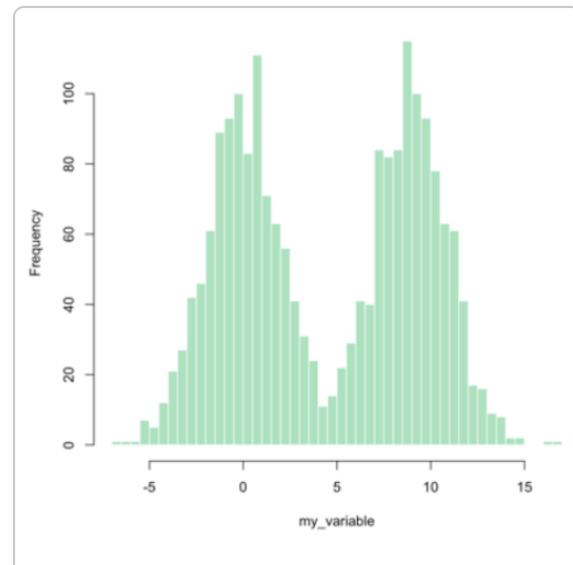
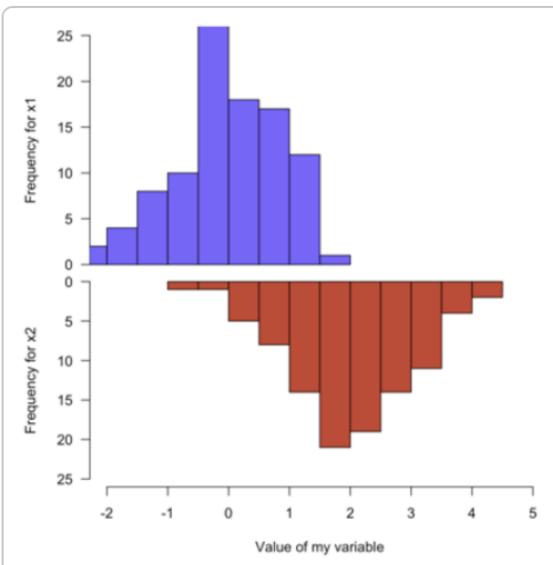
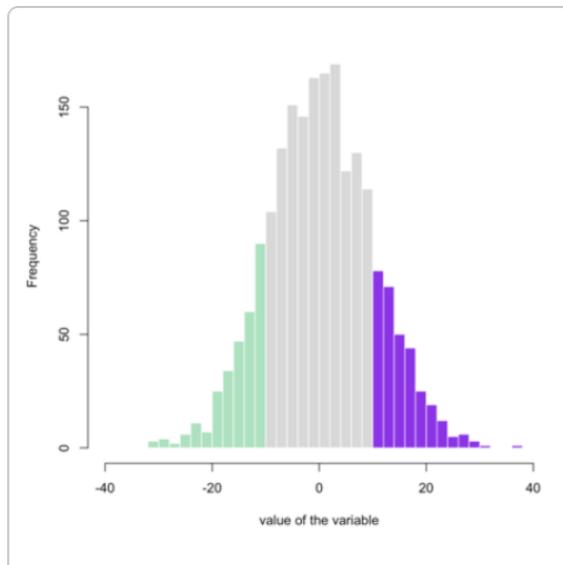
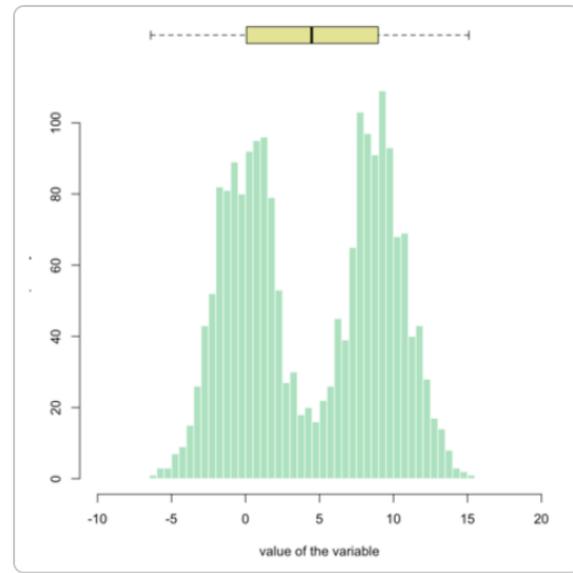
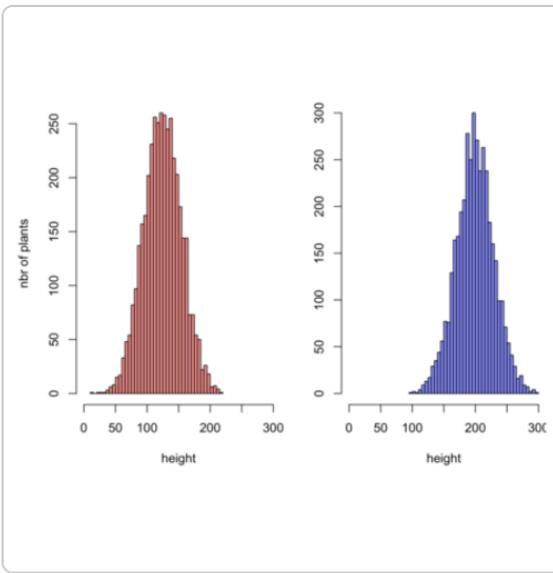
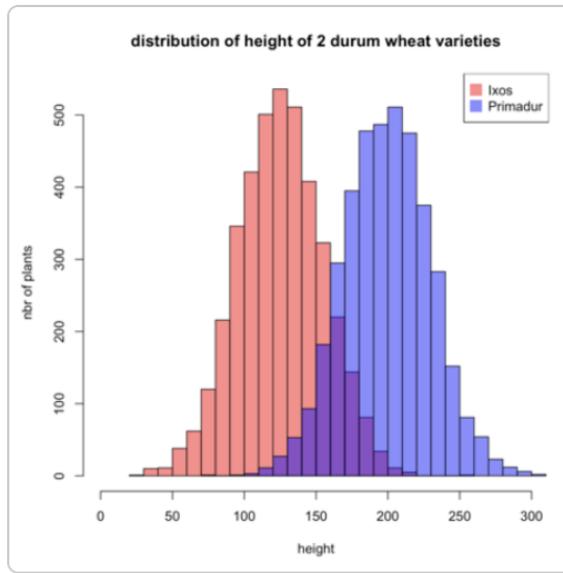
# VISUALIZACIÓN DE DATOS

## HISTOGRAMA

```
> set.seed(2525)
> valores <- rnorm(200, mean = 50, sd = 10) # 200 observaciones con medi
>
> # Crear histograma
> hist(valores,
+ col = "lightblue", # color de las barras
+ border = "white", # color del borde
+ main = "Histograma", # título
+ xlab = "Valor", # etiqueta eje X
+ ylab = "Frecuencia", # etiqueta eje Y
+ breaks = 15) # número de intervalos
```



## HISTOGRAMA



Fuente imagen: <https://r-graph-gallery.com/histogram.html>

- **Parámetros** más comunes para **ajustar la apariencia** de los gráficos:

- `xlim` y `ylim` → límites de los ejes X e Y
- `xlab` y `ylab` → etiquetas de los ejes X e Y
- `main` → título del gráfico
- `sub` → subtítulo del gráfico

- **Al generar un gráfico, éstos se generan en una ventana gráfica temporal.**
- Para conservarlos fuera de R, es necesario **exportarlos a un archivo** en distintos formatos (imagen o documento).
- Se guarda por defecto en el *working environment*, pero se puede especificar otra ruta.
- Las funciones más comunes incluyen → `png("grafico.png"); jpeg("grafico.jpg"); pdf("grafico.pdf")`
- Buenas prácticas incluyen:
  - Nombrar los archivos de forma **clara y descriptiva**.
  - Definir las **dimensiones y resolución** para asegurar la legibilidad según su uso (uso digital, impresión...etc)

- La **ruta del archivo** (o el nombre del archivo) siempre se tiene que indicar **antes de dibujar el gráfico**, porque es justo la llamada a `png()`, `jpeg()` o `pdf()` la que abre el “dispositivo gráfico” donde se va a guardar lo que dibujes.
- Flujo correcto:
  - Llamar a `png("ruta/archivo.png")` → se abre el dispositivo.
  - Hacer el gráfico (`hist(...)`, `plot(...)`, etc.).
  - Cerrar mediante la función `dev.off()` → se guarda el archivo en la ruta indicada.

```
> png("histograma.png", width=800, height=600)
> hist(rnorm(100, mean=50, sd=10),
+ col="lightblue", main="Histograma guardado en PNG")
> dev.off()
```

RStudioGD

2

- R ofrece más alternativas de visualización de datos, según la necesidad:
  - Heatmaps: `heatmap()`
  - Violin plots: `vioplot()`
  - Visualización 3D: con librerías como `rgl`, `plot3D`
  - Gráficos interactivos
  - Árboles jerárquicos
  - Mapas
  - Y un laaaaaargo etcétera...

# **EJERCICIOS PRÁCTICOS – Módulo 3**



Osakidetza



# **Manipulación y reestructuración de datos**

# SESIÓN 1: FUNDAMENTOS DE PROGRAMACIÓN EN R

## ÍNDICE

- 1.1 Introducción a R y RStudio
- 1.2 Programación básica en R
- 1.3 Estadística descriptiva en R
- 1.4 Manipulación y reestructuración de datos**

1.1 Introducción a R y RStudio

1.2 Programación básica en R

1.3 Estadística descriptiva en R

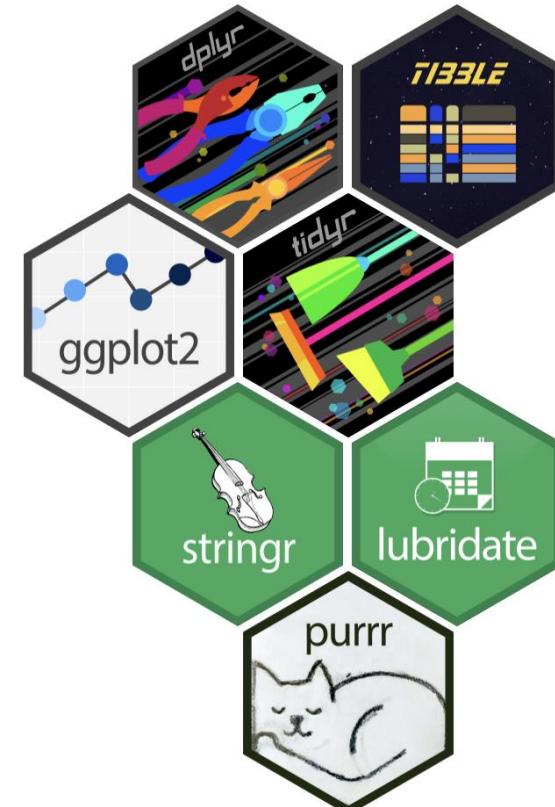
**1.4 Manipulación y reestructuración de datos**

- Introducción al *tidyverse*.
- Cómo crear subconjuntos de datos

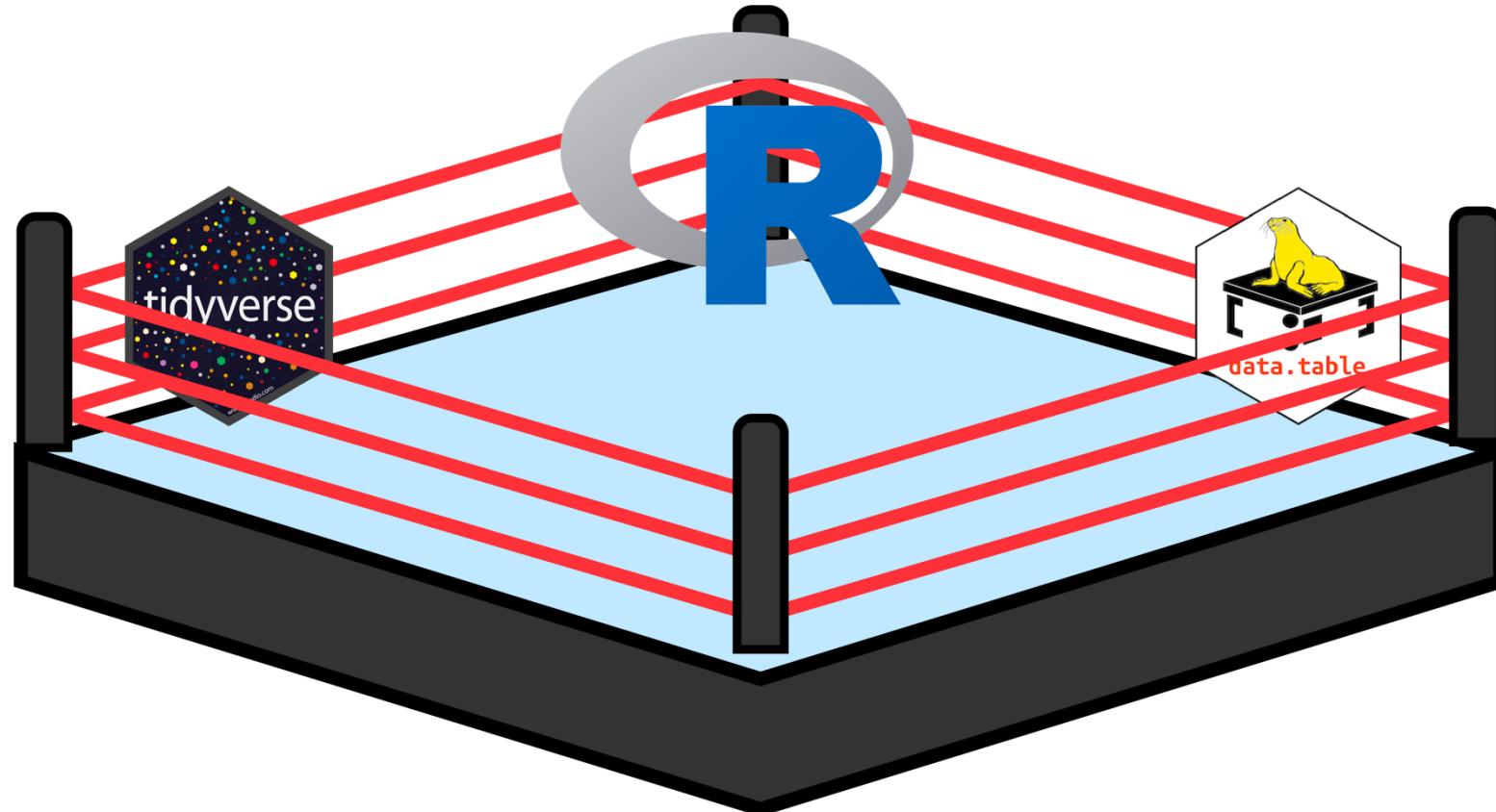
100

## QUÉ ES EL TIDYVERSE

- Conjunto de **paquetes de R** que simplifican el análisis y la visualización de datos.
- Convierte el trabajo con datos en algo **intuitivo y coherente**.
- Promueve una forma **ordenada y eficiente** de trabajar, enfatizando **legibilidad y consistencia**.
- Basado en la idea de **tidy data**:  
datos organizados en filas y columnas.
- Favorece un flujo de trabajo integrado: importar → transformar → visualizar → comunicar.



101



102

Fuente imagen: Fuente: Stack Overflow Discussions (2023)

<https://stackoverflow.com/beta/discussions/77085087/which-r-is-the-best-base-tidyverse-or-data-table>

## QUÉ ES EL TIDYVERSE

### VENTAJAS

**Facilidad** de aprendizaje

**Legibilidad del código** (transmitir claramente el propósito a uno mismo y a otros)

**Rendimiento** (velocidad de cálculo y eficiencia en memoria)

**Concisión** (lograr más con menos líneas de código)

103

| Criterio                        | Tidyverse                                                                                                          | Base R                                                                          |
|---------------------------------|--------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------|
| <b>Facilidad de aprendizaje</b> | Sintaxis intuitiva con verbos claros ( <code>filter()</code> , <code>mutate()</code> , <code>summarise()</code> ). | Sintaxis más técnica, requiere conocer funciones variadas y menos consistentes. |
| <b>Legibilidad del código</b>   | Uso del <i>pipe</i> ( <code>%&gt;%</code> ) que encadena pasos de forma clara y narrativa.                         | Código más fragmentado, puede ser difícil de seguir en procesos largos.         |
| <b>Rendimiento</b>              | Muy eficiente en manipulación de datos grandes, aunque depende de paquetes.                                        | Base R suele ser más rápido en operaciones simples y directas.                  |
| <b>Concisión</b>                | Menos líneas de código para tareas comunes, más expresivo.                                                         | Puede requerir más código y argumentos explícitos para la misma tarea.          |

## QUÉ ES EL TIDYVERSE

### TIDY DATA – DATOS ORDENADOS

- Los *datos ordenados (tidy data)* son aquellos que siguen una estructura clara y estandarizada.
  - **Cada variable → una columna**
  - **Cada observación → una fila**
  - **Cada valor → una celda**
- Importancia:
  - Son el estándar dentro del tidyverse
  - Facilitan el uso de herramientas de análisis y visualización
  - Permiten menos tiempo peleando con el formato y más tiempo analizando resultados.

105

## QUÉ ES EL TIDYVERSE

### TIDY DATA – DATOS ORDENADOS



| Paciente | Fecha      | Tensión | Peso  |
|----------|------------|---------|-------|
| ANE      | 2/5/2025   | 120/80  | 65kg  |
| iker     | 2025-05-12 | 135/90  | 82 kg |



106

| Paciente | Fecha      | Tension.sys | Tension.dia | Peso_kg |
|----------|------------|-------------|-------------|---------|
| Ane      | 2025-05-02 | 120         | 80          | 65      |
| Iker     | 2025-05-12 | 135         | 90          | 82      |

QUÉ ES EL TIDYVERSE  
COMPARACIÓN

```
> df <- data.frame(
+ paciente = 1:6,
+ sexo = c("M","F","M","F","M","F"),
+ edad = c(45,50,60,55,40,65),
+ mes_visita = c("Enero","Febrero","Marzo","Abril","Mayo","Junio"),
+ glucosa = c(110,95,130,105,140,120)
+)
> View(df)
```

|   | paciente | sexo | edad | mes_visita | glucosa |
|---|----------|------|------|------------|---------|
| 1 | 1        | M    | 45   | Enero      | 110     |
| 2 | 2        | F    | 50   | Febrero    | 95      |
| 3 | 3        | M    | 60   | Marzo      | 130     |
| 4 | 4        | F    | 55   | Abril      | 105     |
| 5 | 5        | M    | 40   | Mayo       | 140     |
| 6 | 6        | F    | 65   | Junio      | 120     |

```
> df <- data.frame(
+ paciente = 1:6,
+ sexo = c("M", "F", "M", "F", "M", "F").
```

### ¿QUÉ QUEREMOS HACER?

- (1) Seleccionar columnas +
- (2) Filtrar glucosa > 100 +
- (3) Transformar mes\_visita a minúsculas +
- (4) Renombrar columnas al inglés +
- (5) Transformar IDs a formato 001, 002...

|   |   |   |    |       |     |
|---|---|---|----|-------|-----|
| 3 | 3 | M | 60 | Marzo | 130 |
| 4 | 4 | F | 55 | Abril | 105 |
| 5 | 5 | M | 40 | Mayo  | 140 |
| 6 | 6 | F | 65 | Junio | 120 |



dreamstime.

**BASE R****QUÉ ES EL TIDYVERSE****COMPARACIÓN EJEMPLO**

```
1 #---
2 # EJEMPLO CON BASE R
3 #---
4
5 # Paso 1: Seleccionar columnas
6 df_sel <- df[, c("paciente", "sexo", "mes_visita", "glucosa")]
7
8 # Paso 2: Filtrar glucosa > 100
9 df_filtrado <- df_sel[df_sel$glucosa > 100,]
10
11 # Paso 3: Transformar mes_visita a minúsculas
12 df_filtrado$mes_visita <- strToLower(df_filtrado$mes_visita)
13
14 # Paso 4: Transformar IDs a formato 001, 002...
15 df_filtrado$paciente <- ifelse(df_filtrado$paciente < 10,
16 paste0("00", df_filtrado$paciente),
17 paste0("0", df_filtrado$paciente))
18
19 # Paso 5: Renombrar columnas al inglés
20 names(df_filtrado)[names(df_filtrado) == "paciente"] <- "patient"
21 names(df_filtrado)[names(df_filtrado) == "sexo"] <- "sex"
22 names(df_filtrado)[names(df_filtrado) == "mes_visita"] <- "visit_month"
23 names(df_filtrado)[names(df_filtrado) == "glucosa"] <- "glucose"
24
25 # Paso 6: Mostrar resultado
26 df_filtrado
```

# QUÉ ES EL TIDYVERSE

## COMPARACIÓN EJEMPLO

### TIDYVERSE

```
1 #-----
2 # EJEMPLO CON TIDYVERSE
3 #-----
4
5 df_filtrado <- df %>%
6 # Paso 1: Seleccionar columnas
7 select(paciente, sexo, mes_visita, glucosa) %>%
8 # Paso 2: Filtrar glucosa > 100
9 filter(glucosa > 100) %>%
10 # Paso 3: Transformar mes_visita a minúsculas
11 mutate(mes_visita = stringr::str_to_lower(mes_visita),
12 # Paso 4: Transformar IDs a formato 001, 002...
13 paciente = stringr::str_pad(paciente, width = 3, pad = "0")) %>%
14 # Paso 5: Renombrar columnas al inglés
15 rename(patient = paciente,
16 sex = sexo,
17 visit_month = mes_visita,
18 glucose = glucosa)
19
20 # Paso 6: Mostrar resultado
21 df_filtrado
```

- Ejecutar en la consola de R `install.packages("tidyverse")`
- **Flexibilidad** a la hora de cargar la librería:
  - Cargar todo el conjunto con `library(tidyverse)`
    - Al cargar, aparecerá la lista de paquetes incluídos.
    - Activa todos los paquetes principales de una vez
  - Se puede cargar uno a uno: `library(dplyr)` o `library(tidyr)`
    - Útil si solo se necesitan funciones específicas.
    - Más rápido y mejor para optimizar memoria y tiempo de ejecución.
    - Aseguras qué función está seleccionando.

- **Dplyr** es un paquete del Tidyverse especializado en **manipulación de datos**.
- Permite realizar operaciones como filtrar, seleccionar, agrupar entre otros, de manera **sencilla y legible**.
- Con funciones como `filter()`, `select()`, y `mutate()`, es posible transformar los conjuntos de datos rápidamente.



112

- **Tidyr es un paquete del Tidyverse especializado en la limpieza y transformación de datos.**
- Permite convertir datos de un formato ancho a uno largo y viceversa, lo que es fundamental para la **preparación de datos antes del análisis**.
- Con funciones como `pivot_longer()` y `pivot_wider()`, es posible reorganizar los datos para que se adapten a las necesidades analíticas de cada caso.



113

- Lubridate es un paquete del Tidyverse especializado en el **manejo de fechas y horas**.
- Permite simplificar operaciones complejas con fechas y horas.
- Con funciones como `ymd("YYYY-MM-DD")`, `month(fecha)` o `interval(f_inicio, f_fin)` es capaz de reorganizar las variables para que se adapten a las necesidades de cada caso.



114

- **Stringr** es un paquete del Tidyverse cuyo objetivo es hacer que **trabajar con texto sea lo más sencillo posible.**
- Permite simplificar operaciones complejas como la búsqueda de patrones, extracción y reemplazo de texto, o la transformación a mayúsculas/minúsculas.
- Ofrece un conjunto de funciones como `str_detect()`, `str_replace()`, `str_to_lower()` que se adaptan a las necesidades de cada caso.



115

- Magrittr es un paquete que **ofrece un conjunto de operadores para hacer el código más legible.**
- Se carga automáticamente dentro del tidyverse.
- Estructura las operaciones de datos de forma que se envía el valor de la izquierda como argumento a la expresión de la derecha.
- **Facilita añadir pasos** en cualquier punto de la secuencia.
- Hace que el **flujo de operaciones sea intuitivo y fácil de seguir.**

`%>%` → *pipe operator*



116

```

test_positivo <- df %>%
 filter(result == "positive")
dim(test_positivo)

[1] 865 17

FILTER CON MÁS DE UNA CONDICIÓN
test_pos_fem <- df %>%
 filter(result == "positive" & gender == "female")

dim(test_pos_fem)

[1] 449 17

summary(as.factor(df$demo_group))

client misc adult other.adult patient unidentified
604 2441 223 12255 1

Filter CON EL OPERADOR OR
test_pos_f_demo <- df %>%
 filter(result == "positive" & gender == "female" & (demo_group == "client" | demo_group == "patient"))

dim(test_pos_f_demo)

[1] 329 17

```

## SELECT()

```
SELECT
test <- df %>%
 select(test_id, result, clinic_name) # El orden de las columnas se mantiene

SELECT TODAS LAS COLUMNAS MENOS UNA
selection <- df %>%
 select(-subject_id)

SELECT TODAS MENOS DOS O MÁS - CON VECTOR
selection <- df %>%
 select(-c(gender, demo_group))

selection <- df %>%
 select(everything(), -result)

SELECT CON UN PATRÓN (start and end)
selection <- df %>%
 select(starts_with("fake_"), gender)

selection <- df %>%
 select(-ends_with("_id")) # not select
```



```
MUTATE - new column
new_col <- df %>%
 mutate(wave = "First")

MUTATE - Crear nueva variable según valores de otra
positive <- df %>%
 select(result) %>%
 mutate(is_positive = if_else(result == "positive", 1, 0))

drive <- df %>%
 select(drive_thru_ind) %>%
 mutate(sample_in_car = if_else(drive_thru_ind == 1, "YES", "NO"))
```

```
RENAME
gender <- df %>%
 rename(sex = gender)

Es posible hacer rename dentro de SELECT
gender2 <- df %>%
 select(subject_id, starts_with("fake_"), sex = gender)
```

120

## GROUP\_BY() + SUMMARISE() + UNGROUP()

```
GROUP_BY and BASIC STATS
gender_pos <- df %>%
```

```
group_by(gender) %>% # Esto solo no hace nada!
summarise(total_positive = sum(result == "positive")) %>%
ungroup() # para seguir haciendo otros cambios
```

```
gender_pos # están agrupados - dimensión ha cambiado
```

```
A tibble: 2 x 2
gender total_positive
<chr> <int>
1 female 449
2 male 416
```

# 1.4

## Estadística descriptiva

# MANIPULACIÓN DE DATOS

## ARRANGE()

```
ARRANGE - ordenar filas
df_by_age <- df %>%
 select(age) %>%
 arrange(age)

head(df_by_age)
```

```
A tibble: 6 x 1
age
<dbl>
1 0
2 0
3 0
4 0
5 0
6 0
```

**DE MENOR A MAYOR**

```
df_by_age_desc <- df %>%
 select(age) %>%
 arrange(desc(age))

head(df_by_age_desc) # Vemos a gente con 138 años! ALERTA!
```

```
A tibble: 6 x 1
age
<dbl>
1 138
2 119
3 119
4 119
5 119
6 99
```

**DE MAYOR A MENOR**

```
library(lubridate)

INCLUDE DATES AND VISUALIZE
fechas <- df %>%
 mutate(f_extraction = sample(seq(as.Date("2020-03-20"),
 as.Date("2020-12-31"), by = "day"),
 size = n(), replace = TRUE),
 mes = month(f_extraction, label = TRUE, abbr = FALSE),
 anno = year(f_extraction)) %>%
 select(f_extraction, mes, anno)

head(fechas)

A tibble: 6 x 3
f_extraction mes anno
<date> <ord> <dbl>
1 2020-12-10 December 2020
2 2020-05-10 May 2020
3 2020-08-15 August 2020
4 2020-05-31 May 2020
5 2020-10-06 October 2020
6 2020-09-02 September 2020
```

**STR\_TO\_UPPER(); STR\_TO\_SENTENCE(); STR\_REPLACE()**

```
library(stringr)

MODIFY THE STRINGS
nombres <- df %>%
 mutate(fake_first_name = str_to_upper(fake_first_name),
 fake_last_name = str_to_sentence(fake_last_name),
 test_id = str_replace(test_id, "covid", "COVID-19")) %>%
 select(starts_with("fake"), test_id)

head(nombres)

A tibble: 6 x 3
fake_first_name fake_last_name test_id
<chr> <chr> <chr>
1 JHEZANE Westerling COVID-19
2 PENNY Targaryen COVID-19
3 GRUNT Rivers COVID-19
4 MELISANDRE Swyft COVID-19
5 ROLLEY Karstark COVID-19
6 MEGGA Karstark COVID-19
```

### VALORES FALTANTES – MISSING VALUES

- Los valores faltantes (NA) pueden sesgar resultados y análisis.
- Revisar si el dataset está completo – esto evita errores en la visualización y en los análisis.

#### LA CALIDAD DE LOS DATOS ES CLAVE PARA OBTENER CONCLUSIONES FIABLES

- Algunas funciones clave para esta revisión:
  - Drop\_na() → elimina filas con valores faltantes
  - Replace\_na() → sustituye valores faltantes por un valor definido
  - Fill() → rellena valores faltantes con el último observado
- Decidir si conviene **eliminar, imputar o mantener** los valores faltantes según el contexto

# **EJERCICIOS PRÁCTICOS – Módulo 4**



Osakidetza



**ESKERRIK ASKO  
MUCHAS GRACIAS  
THANK YOU**



**Osakidetza**



## REESTRUCTURACIÓN DE DATOS AVANZADO

**PIVOT\_LONGER(); PIVOT\_WIDER()**

## Reshape Data

- Pivot data to reorganize values into a new layout.

table4a

| country | 1999 | 2000 |
|---------|------|------|
| A       | 0.7K | 2K   |
| B       | 37K  | 80K  |
| C       | 212K | 213K |



| country | year | cases |
|---------|------|-------|
| A       | 1999 | 0.7K  |
| B       | 1999 | 37K   |
| C       | 1999 | 212K  |
| A       | 2000 | 2K    |
| B       | 2000 | 80K   |
| C       | 2000 | 213K  |

**pivot\_longer(data, cols, names\_to = "name", values\_to = "value", values\_drop\_na = FALSE)**  
 "Lengthen" data by collapsing several columns into two. Column names move to a new names\_to column and values to a new values\_to column.

```
pivot_longer(table4a, cols = 2:3, names_to = "year",
 values_to = "cases")
```

table2

| country | year | type  | count |
|---------|------|-------|-------|
| A       | 1999 | cases | 0.7K  |
| A       | 1999 | pop   | 19M   |
| A       | 2000 | cases | 2K    |
| A       | 2000 | pop   | 20M   |
| B       | 1999 | cases | 37K   |
| B       | 1999 | pop   | 172M  |
| B       | 2000 | cases | 80K   |
| B       | 2000 | pop   | 174M  |
| C       | 1999 | cases | 212K  |
| C       | 1999 | pop   | 1T    |
| C       | 2000 | cases | 213K  |
| C       | 2000 | pop   | 1T    |



| country | year | cases | pop  |
|---------|------|-------|------|
| A       | 1999 | 0.7K  | 19M  |
| A       | 2000 | 2K    | 20M  |
| B       | 1999 | 37K   | 172M |
| B       | 2000 | 80K   | 174M |
| C       | 1999 | 212K  | 1T   |
| C       | 2000 | 213K  | 1T   |

**pivot\_wider(data, names\_from = "name", values\_from = "value")**

The inverse of pivot\_longer(). "Widen" data by expanding two columns into several. One column provides the new column names, the other the values.

```
pivot_wider(table2, names_from = type,
 values_from = count)
```

## REESTRUCTURACIÓN DE DATOS AVANZADO

`LEFT_JOIN(); RIGHT_JOIN(); FULL_JOIN()`

| A | B | C | D  |
|---|---|---|----|
| a | t | 1 | 3  |
| b | u | 2 | 2  |
| c | v | 3 | NA |

**left\_join(x, y, by = NULL, copy = FALSE,  
suffix = c(".x", ".y"), ..., keep = FALSE,  
na\_matches = "na")** Join matching  
values from y to x.

| A | B | C  | D |
|---|---|----|---|
| a | t | 1  | 3 |
| b | u | 2  | 2 |
| d | w | NA | 1 |

**right\_join(x, y, by = NULL, copy = FALSE,  
suffix = c(".x", ".y"), ..., keep = FALSE,  
na\_matches = "na")** Join matching  
values from x to y.

| A | B | C | D  |
|---|---|---|----|
| a | t | 1 | 3  |
| b | u | 2 | 2  |
| c | v | 3 | NA |

**full\_join(x, y, by = NULL, copy = FALSE,  
suffix = c(".x", ".y"), ..., keep = FALSE,  
na\_matches = "na")** Join data. Retain all  
values, all rows.

129

y algunos más: **inner\_join(),  
anti\_join(),**

Etc...