

# Prácticas de Tecnologías de Gestión y Manipulación de Datos

Guillermo López Taboada (guillermo.lopez.taboada@udc.es), Diego Darriba (diego.darriba@udc.e

2025-10-24



# Índice general

<b>Prólogo</b>	<b>5</b>
<b>1 Introducción</b>	<b>7</b>
1.1 Contenidos . . . . .	7
1.2 Planificación (tentativa) . . . . .	8
1.3 Fuentes de información: . . . . .	9
<b>2 Manipulación de datos con R</b>	<b>11</b>
2.1 Lectura, importación y exportación de datos . . . . .	11
2.2 Manipulación de datos . . . . .	17
2.3 Datos faltantes . . . . .	25
2.4 Ejemplo WoS data . . . . .	33
<b>3 Introducción al lenguaje SQL</b>	<b>39</b>
3.1 Bases de Datos Relacionales . . . . .	39
3.2 Restricciones . . . . .	43
3.3 Sistemas Gestores de Bases de Datos (SGDB) . . . . .	44
3.4 Sintaxis SQL . . . . .	46
3.5 Cláusulas básicas de SQL . . . . .	47
3.6 Gestión de tablas . . . . .	49
3.7 Gestión de datos . . . . .	49
3.8 Gestión de Bases de Datos . . . . .	50
3.9 Ejemplos de consultas SQL . . . . .	50
3.10 Conexión con bases de datos desde R . . . . .	51
3.11 Ejemplo Scopus data . . . . .	54
3.12 Ejercicios SQL con RSQLite . . . . .	54
3.13 Práctica 1: SQL . . . . .	56
<b>4 Manipulación de datos con tidyverse</b>	<b>57</b>
4.1 Introducción al ecosistema tidyverse . . . . .	57
4.2 Manipulación de datos con dplyr y tidyr . . . . .	62
4.3 Herramientas tidyr . . . . .	70
4.4 Operaciones con tablas de datos . . . . .	70

4.5 Bases de datos con dplyr . . . . .	71
<b>5 Introducción a Tecnologías NoSQL</b>	<b>77</b>
5.1 Conceptos y tipos de bases de datos NoSQL (documental, columnar, clave/valor y de grafos) . . . . .	77
5.2 Conexión de R a MongoDB . . . . .	85
5.3 Ejercicios prácticos con MongoDB . . . . .	86
<b>6 Tecnologías para el Tratamiento de Datos Masivos</b>	<b>87</b>
6.1 Introducción al Aprendizaje Estadístico . . . . .	87
6.2 Tecnologías Big Data (Hadoop/Spark y Visualización) . . . . .	88
6.3 Introducción al Análisis de Datos Masivos . . . . .	95
<b>A Enlaces</b>	<b>97</b>
A.1 RStudio . . . . .	99
A.2 Bibliometría . . . . .	100
<b>B Instalación de R</b>	<b>101</b>
B.1 Instalación de R en Windows . . . . .	102
B.2 Instalación en Mac OS X . . . . .	104
B.3 Instalación (opcional) de un entorno o editor de comandos . . . . .	105

# Prólogo

Este libro contiene algunas de las prácticas de la asignatura de Tecnologías de Gestión de Datos del Máster interuniversitario en Técnicas Estadísticas).

Este libro ha sido escrito en R-Markdown empleando el paquete `bookdown` y está disponible en el repositorio Github: [gltaboada/tgdbook](https://gltaboada.github.io/tgdbook). Se puede acceder a la versión en línea a través del siguiente enlace:

<https://gltaboada.github.io/tgdbook>.

donde puede descargarse en formato pdf.

Para ejecutar los ejemplos mostrados en el libro será necesario tener instalados los siguientes paquetes: `dplyr` (colección `tidyverse`), `tidyr`, `stringr`, `readxl`, `openxlsx`, `RODBC`, `sqldf`, `RSQLite`, `foreign`, `magrittr`, `knitr` Por ejemplo mediante los comandos:

```
pkgs <- c('dplyr', 'tidyr', 'stringr', 'readxl', 'openxlsx', 'magrittr',
          'RODBC', 'sqldf', 'RSQLite', 'foreign', 'knitr')
# install.packages(pkgs, dependencies=TRUE)
install.packages(setdiff(pkgs, installed.packages()[, 'Package']), dependencies = TRUE)
```

Para generar el libro (compilar) se recomendaría consultar el libro de “Escritura de libros con bookdown” en castellano.



Este obra está bajo una licencia de Creative Commons Reconocimiento-NoComercial-SinObraDerivada 4.0 Internacional (esperamos poder liberarlo bajo una licencia menos restrictiva más adelante...).



# Capítulo 1

## Introducción

La información relevante de la materia está disponible en la guía docente y la ficha de la asignatura

En particular, los resultados de aprendizaje son:

- Manejar de forma autónoma y solvente el software necesario para acceder a conjuntos de datos en entornos profesionales y/o en la nube.
- Saber gestionar conjuntos de datos masivos en un entorno multidisciplinar que permita la participación en proyectos profesionales complejos que requieran el uso de técnicas estadísticas.
- Saber relacionar el software de diseño y gestión de bases de datos con el específicamente implementado para el análisis de datos.

### 1.1 Contenidos

1. Introducción al lenguaje SQL
  - Bases de datos relacionales
  - Sintaxis SQL
  - Conexión con bases de datos desde R
2. Introducción a tecnologías NoSQL
  - Conceptos y tipos de bases de datos NoSQL (documental, columnar, clave/valor y de grafos)
  - Conexión de R a NoSQL
3. Tecnologías para el tratamiento de datos masivos
  - Introducción al Aprendizaje Estadístico
  - Tecnologías Big Data (Hadoop, Spark, Sparklyr)
  - Ejercicios de análisis de datos masivos.

## 1.2 Planificación (tentativa)

La impartición de los contenidos durante el curso dependerá de los conocimientos de partida y la asimilación de los conceptos. Para completar nuestra visión de los conocimientos previos os requerimos completar este formulario en la primera sesión de clase: <https://forms.gle/D5bhiLLBUFuh6k1n8>

- Semana 1 (9/9) : 10 y 11/9 - Presentación e introducción a Tema 1 & SQL
- Semana 2 (16/9) : 17 y 18/9 - Seminario dplyr.
- Semana 3 (23/9) : 24 y 25/9 - Tema 1 & SQL.
- Semana 4 (30/9) : 1 y 2/10 - Ejercicios SQL.
- Semana 5 (7/10) : 8 y 9/10 - Tema 2 & NoSQL - Seminario texto proc. (CSV, excel, Json) y open data .
- Semana 6 (14/10) : 15/10 - Tutorial sparklyr-SQL. + 16/10 Consultas SQL con sparklyr.
- Semana 7 (21/10) : 22 y 23/10 Tema 3 Big Data.
- Semana 8 (28/10) : 29 y 30/10 - Tutoriales sparklyr-ML.
- Semana 9 (4/11) : 6/11 - Ejercicios ML.
- Semana 10(11/11) : 11 y 13/11 - Intro a AE (día 13 1 hora de 13 a 14h).
- Semana 11(18/11) : 18/11 - Intro a AE Manuel, 20 y 21/11 - Ejercicios ML.
- Semana 12 (25/11): 25/11 o cualquier otro día hasta el 5/12, seguramente el 5/12, dudas práctica ML. Backup.

Examen 21/1/25 4pm.

Recuperación 1/7/25 4pm.

### 1.2.1 Evaluación

- **Examen (60%)**: El examen de la materia evaluará los siguientes aspectos: Conceptos de la materia: Dominio de los conocimientos teóricos y operativos de la materia. Asimilación práctica de materia: Asimilación y comprensión de los conocimientos teóricos y operativos de la materia.
- **Prácticas de laboratorio (40%)**: Evaluación de las prácticas de laboratorio desarrolladas por los estudiantes.

## 1.3 Fuentes de información:

### 1.3.1 Básica

- Daroczi, G. (2015). Mastering Data Analysis with R. Packt Publishing
- Grolemund, G. y Wickham, H. (2016). R for Data Science O'Reilly
- Silberschatz, A., Korth, H. y Sudarshan, S. (2014). Fundamentos de Bases de Datos. Mc Graw Hill
- Rubén Fernández Casal y Julián Costa Bouza. Apuntes de Aprendizaje Estadístico
- Luraschi, J., Kuo, K., Ruiz, K. Mastering Spark with R O'Reilly
- Rubén Fernández Casal (R Machinery):
  - Introducción al Análisis de Datos con R (con Javier Roca y Julián Costa)
  - Ayuda y Recursos para el Aprendizaje de R
  - Escritura de libros con el paquete bookdown (con Tomás Cotos)
  - Apéndice introducción a Rmarkdown
  - Presentación análisis de datos con R

### 1.3.2 Complementaria:

- Wes McKinney (2017). Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython. O'Reilly (2<sup>a</sup> ed.)
- Tom White (2015). Hadoop: The Definitive Guide. O'Reilly (4<sup>a</sup> ed.)
- Alex Holmes (2014). Hadoop in practice. Manning (2<sup>a</sup> ed.)
- Centro de Supercomputación de Galicia (2020). [Servicio de Big Data del CESGA] (<https://bigdata.cesga.es/>)



## Capítulo 2

# Manipulación de datos con R

En el proceso de análisis de datos, al margen de su obtención y organización, una de las primeras etapas es el acceso y la manipulación de los datos (ver Figura 2.1). En este capítulo se repasarán brevemente las principales herramientas disponibles en el paquete base de R para ello. Posteriormente en el Capítulo 4 se mostrará como alternativa el uso del paquete `dplyr`.

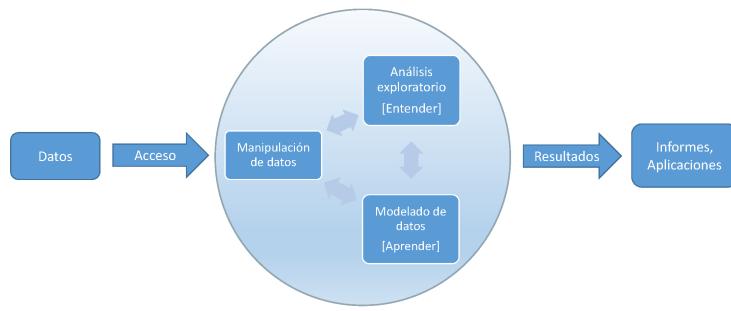


Figura 2.1: Etapas del proceso

### 2.1 Lectura, importación y exportación de datos

Además de la introducción directa, R es capaz de importar datos externos en múltiples formatos:

- bases de datos disponibles en librerías de R
- archivos de texto en formato ASCII

- archivos en otros formatos: Excel, SPSS, Matlab...
- bases de datos relacionales: MySQL, Oracle...
- formatos web: HTML, XML, JSON...
- otros lenguajes de programación: Python, Julia...

### 2.1.1 Formato de datos de R

El formato de archivo en el que habitualmente se almacena objetos (datos) R es binario y está comprimido (en formato "gzip" por defecto). Para cargar un fichero de datos se emplea normalmente `load()`. A continuación se utiliza el fichero `empleados.RData` que contiene datos de empleados de un banco.

```
res <- load("data/empleados.RData")
res

## [1] "empleados"

ls()

##  [1] "cite_cran"  "cite_fig"   "cite_fig2"  "cite_fun"
##  [5] "cite_fun_." "cite_pkg"   "cite_pkg_." "citefig"
##  [9] "citefig2"   "empleados"  "fig.path"   "inline"
## [13] "inline2"    "is_html"   "is_latex"  "latexfig"
## [17] "latexfig2"  "res"

y para guardar save():
# Guardar
save(empleados, file = "data/empleados_new.RData")
```

Aunque, como indica este comando en la ayuda (`?save`):

*For saving single R objects, `saveRDS()` is mostly preferable to `save()`, notably because of the functional nature of `readRDS()`, as opposed to `load()`.*

```
saveRDS(empleados, file = "data/empleados_new.rds")
## restore it under a different name
empleados2 <- readRDS("data/empleados_new.rds")
# identical(empleados, empleados2)
```

Normalmente, el objeto empleado en R para almacenar datos en memoria es el `data.frame`.

### 2.1.2 Acceso a datos en paquetes

R dispone de múltiples conjuntos de datos en distintos paquetes, especialmente en el paquete `datasets` que se carga por defecto al abrir R. Con el comando `data()` podemos obtener un listado de las bases de datos disponibles.

Para cargar una base de datos concreta se utiliza el comando `data(nombre)` (aunque en algunos casos se cargan automáticamente al emplearlos). Por ejemplo, `data(cars)` carga la base de datos llamada `cars` en el entorno de trabajo (`".GlobalEnv"`) y `?cars` muestra la ayuda correspondiente con la descripción de la base de datos.

### 2.1.3 Lectura de archivos de texto

En R, para leer archivos de texto se suele utilizar la función `read.table()`. Suponinedo, por ejemplo, que en el directorio actual está el fichero `empleados.txt`. La lectura de este fichero vendría dada por el código:

```
# Session > Set Working Directory > To Source...?
datos <- read.table(file = "data/empleados.txt", header = TRUE)
# head(datos)
str(datos)

## 'data.frame': 474 obs. of 10 variables:
## $ id      : int 1 2 3 4 5 6 7 8 9 10 ...
## $ sexo    : chr "Hombre" "Hombre" "Mujer" "Mujer" ...
## $ fechnac : chr "2/3/1952" "5/23/1958" "7/26/1929" "4/15/1947" ...
## $ educ    : int 15 16 12 8 15 15 15 12 15 12 ...
## $ catlab  : chr "Directivo" "Administrativo" "Administrativo" "Administrativo" ...
## $ salario : num 57000 40200 21450 21900 45000 ...
## $ salini  : int 27000 18750 12000 13200 21000 13500 18750 9750 12750 13500 ...
## $ tiempemp: int 98 98 98 98 98 98 98 98 98 98 ...
## $ expprev : int 144 36 381 190 138 67 114 0 115 244 ...
## $ minoria : chr "No" "No" "No" "No" ...
class(datos)

## [1] "data.frame"
```

Si el fichero estuviese en el directorio `c:/datos` bastaría con especificar `file = "c:/datos/empleados.txt"`. Nótese también que para la lectura del fichero anterior se ha establecido el argumento `header=TRUE` para indicar que la primera línea del fichero contiene los nombres de las variables.

Los argumentos utilizados habitualmente para esta función son:

- `header`: indica si el fichero tiene cabecera (`header=TRUE`) o no (`header=FALSE`). Por defecto toma el valor `header=FALSE`.
- `sep`: carácter separador de columnas que por defecto es un espacio en blanco (`sep=""`). Otras opciones serían: `sep=","` si el separador es un `,`, `sep="*"` si el separador es un `*`, etc.
- `dec`: carácter utilizado en el fichero para los números decimales. Por defecto se establece `dec = ". "`. Si los decimales vienen dados por `,` se utiliza `dec = ","`.

Resumiendo, los (principales) argumentos por defecto de la función `read.table` son los que se muestran en la siguiente línea:

```
read.table(file, header = FALSE, sep = "", dec = ".")
```

Para más detalles sobre esta función véase `help(read.table)`.

Están disponibles otras funciones con valores por defecto de los parámetros adecuados para otras situaciones. Por ejemplo, para ficheros separados por tabuladores se puede utilizar `read.delim()` o `read.delim2()`:

```
read.delim(file, header = TRUE, sep = "\t", dec = ".")
read.delim2(file, header = TRUE, sep = "\t", dec = ",")
```

#### 2.1.4 Importación desde SPSS

El programa R permite lectura de ficheros de datos en formato SPSS (extensión `.sav`) sin necesidad de tener instalado dicho programa en el ordenador. Para ello se necesita:

- cargar la librería `foreign`
- utilizar la función `read.spss`

Por ejemplo:

```
library(foreign)
datos <- read.spss(file = "data/Employee data.sav",
                    to.data.frame = TRUE)
# head(datos)
str(datos)

## 'data.frame':   474 obs. of  10 variables:
## $ id      : num  1 2 3 4 5 6 7 8 9 10 ...
## $ sexo    : Factor w/ 2 levels "Hombre","Mujer": 1 1 2 2 1 1 1 2 2 2 ...
## $ fechnac : num  1.17e+10 1.19e+10 1.09e+10 1.15e+10 1.17e+10 ...
## $ educ    : Factor w/ 10 levels "8","12","14",...: 4 5 2 1 4 4 4 2 4 2 ...
## $ catlab  : Factor w/ 3 levels "Administrativo",...: 3 1 1 1 1 1 1 1 1 ...
## $ salario : Factor w/ 221 levels "15750","15900",...: 179 137 28 31 150 101 121 31 71 45 ...
## $ salini  : Factor w/ 90 levels "9000","9750",...: 60 42 13 21 48 23 42 2 18 23 ...
## $ tiempemp: Factor w/ 36 levels "63","64","65",...: 36 36 36 36 36 36 36 36 36 36 ...
## $ expprev : Factor w/ 208 levels "Ausente","10",...: 38 131 139 64 34 181 13 1 14 91 ...
## $ minoria : Factor w/ 2 levels "No","Sí": 1 1 1 1 1 1 1 1 1 ...
## - attr(*, "variable.labels")= Named chr [1:10] "Código de empleado" "Sexo" "Fecha de nacimiento" ...
##   ..- attr(*, "names")= chr [1:10] "id" "sexo" "fechnac" "educ" ...
## - attr(*, "codepage")= int 1252
```

**Nota:** Si hay fechas, puede ser recomendable emplear la función `spss.get()` del paquete `Hmisc`.

### 2.1.5 Importación desde Excel

Se pueden leer fichero de Excel (con extensión `.xlsx`) utilizando, por ejemplo, los paquetes:

- `openxlsx`,

```
library(openxlsx)
datos<-read.xlsx("./data/coches.xlsx")
class(datos)
```

```
## [1] "data.frame"
```

- `RODBC` (este paquete se empleará más adelante para acceder a bases de datos), entre otros.

El siguiente código implementa una función que permite leer todos los archivos en formato `.xlsx` en un directorio:

```
library(openxlsx)
read_xlsx <- function(path = '.') {
  files <- dir(path, pattern = '*.xlsx') # list.files
  # file.list <- lapply(files, readWorkbook)
  file.list <- vector(length(files), mode = 'list')
  for (i in seq_along(files))
    file.list[[i]] <- readWorkbook(files[i])
  file.names <- sub('\\\\.xlsx$', '', basename(files))
  names(file.list) <- file.names
  file.list
}
```

Para combinar los archivos, suponiendo que tienen las mismas columnas, podríamos ejecutar una llamada a `rbind()`(R base):

```
df <- do.call('rbind', file.list)
```

o emplear la función `bind_rows()` del paquete `dplyr`, donde las columnas se emparejan por nombre, y cualquier columna que falte se rellenará con `NA`:

```
df <- dplyr::bind_rows(file.list)
```

El Capítulo 4, provee de otras utilidades para la manipulación de datos con `dplyr` (Wickham et al., 2023b).

Los datos cargados en R (usualmente un `data.frame`) se pueden exportar desde Excel fácilmente a un archivo de texto *separado por comas* (extensión `.csv`), evitando utilizar algunos de los paquetes mencionados anteriormente. Por ejemplo, supongamos que queremos leer el fichero `coches.xls`:

- Desde Excel, se selecciona el menú `Archivo -> Guardar como -> Guardar como`, y en `Tipo`, se escoge la opción de archivo CSV. De esta forma se guardarán los datos en el archivo `coches.csv`.

- El fichero *coches.csv* es un fichero de texto plano (se puede editar con el bloc de notas, *Notepad*), con cabecera, las columnas separadas por “;”, y siendo “,” el carácter decimal.
- Por lo tanto, la lectura de este fichero se puede hacer con:

```
datos <- read.table("coches.csv", header = TRUE,
                     sep = ";", dec = ",")
```

Otra posibilidad, es utilizar la función `read.csv2`. Esta función no es más que una adaptación de la función general `read.table` con las siguientes opciones:

```
read.csv2(file, header = TRUE, sep = ";", dec = ",", ...)
```

Por lo tanto, la lectura del fichero *coches.csv* se puede hacer de modo más directo con:

```
datos <- read.csv2("coches.csv")
```

Esta forma de proceder, exportando a formato CSV, se puede emplear con otras hojas de cálculo o fuentes de datos. Hay que tener en cuenta que si estas fuentes emplean el formato anglosajón, el separador de campos será `sep = ","` y el de decimales `dec = "."`, las opciones por defecto en la función `read.csv()`.

### 2.1.6 Exportación de datos

Puede ser de interés la exportación de datos para que puedan ser leídos con otros programas. Para ello, se puede emplear la función `write.table()`. Esta función es similar, pero funcionando en sentido inverso, a `read.table()`, ver Sección 2.1.3.

Veamos un ejemplo:

```
tipo <- c("A", "B", "C")
longitud <- c(120.34, 99.45, 115.67)
datos <- data.frame(tipo, longitud)
datos

##   tipo longitud
## 1     A    120.34
## 2     B     99.45
## 3     C    115.67
```

Para guardar el `data.frame` `datos` en un fichero de texto se puede utilizar:

```
write.table(datos, file = "datos.txt")
```

Otra posibilidad es utilizar la función:

```
write.csv2(datos, file = "datos.csv")
```

que dará lugar al fichero *datos.csv* importable directamente desde Excel. Las opciones anteriores sólo dependen del paquete **utils**, que se instala por defecto con R base.

### 2.1.7 Python, Julia y otros lenguajes de programación

R es un lenguaje de programación libre (derivado del lenguaje S en los Laboratorios Bell) que se caracteriza por su capacidad para interactuar con otros lenguajes de programación, incluyendo Python (Van Rossum and Drake Jr, 1995) y Julia (Bezanson et al., 2017).

En el ámbito de la Estadística (como en la denominada **Ciendica de Datos**), R destaca por su extensa y detallada documentación (en muchos casos como resultado de aportaciones metodológicas y/o avances científicos). Por ejemplo, después de diez años de la primera edición del libro *An Introduction to Statistical Learning con aplicaciones en R (ISLR)* , James et al. (2013), algunos de los mismos autores publicaron la edición en Python (ISLP), James et al. (2023). Por otro lado, en 2015, se lanzó el paquete **reticulate** disponible en <https://rstudio.github.io/reticulate/>, permitiendo la ejecución de código Python desde R (y en 2020 se completó la integración de Python en la interfaz de RStudio).

```
library(reticulate)
os <- import("os")
os$listdir(".")
```

Si queremos trabajar con Python de forma interactiva, podemos usar **repl\_python()**. Los objetos creados en Python se pueden usar en R con **py** de **reticulate**.

Recientemente, *Julia* se presenta también como una alternativa a considerar. El paquete **JuliaConnectoR** disponible en <https://cran.r-project.org/web/packages/JuliaConnectoR/> facilita la importación de funciones y paquetes completos de Julia a R, es decir, permite el uso de funciones de Julia directamente en R.

R también permite el uso/comunicación de otros lenguajes de programación como Java, C, C++, Fortran, entre otros.

## 2.2 Manipulación de datos

Una vez cargada una (o varias) bases de datos hay una series de operaciones que serán de interés para el tratamiento de datos:

- Operaciones con variables:
  - crear
  - recodificar (e.g. categorizar)
  - ...
- Operaciones con casos:

- ordenar
- filtrar
- ...
- Operaciones con tablas de datos:
  - unir
  - combinar
  - consultar
  - ...

A continuación se tratan algunas operaciones *básicas*.

### 2.2.1 Operaciones con variables

#### 2.2.1.1 Creación (y eliminación) de variables

Consideremos de nuevo la base de datos `cars` incluida en el paquete `datasets`:

```
data(cars)
# str(cars)
head(cars)
```

```
##   speed dist
## 1     4    2
## 2     4   10
## 3     7    4
## 4     7   22
## 5     8   16
## 6     9   10
```

Utilizando el comando `help(cars)` se obtiene que `cars` es un `data.frame` con 50 observaciones y dos variables:

- `speed`: Velocidad (en millas por hora)
- `dist`: tiempo hasta detenerse (en pies)

Recordemos que, para acceder a la variable `speed` se puede hacer directamente con su nombre o bien utilizando notación “matricial” (se seleccionan las 6 primeras observaciones por comodidad).

```
cars$speed
```

```
## [1]  4  4  7  7  8  9 10 10 10 11 11 12 12 12 12 13 13 13
## [19] 13 14 14 14 14 15 15 15 16 16 17 17 17 18 18 18 18 19
## [37] 19 19 20 20 20 20 22 23 24 24 24 24 24 25
# cars[, 1]      # Equivalente
# cars[, "speed"] # Equivalente
```

Supongamos ahora que queremos transformar la variable original `speed` (millas por hora) en una nueva variable `velocidad` (kilómetros por hora) y añadir esta

nueva variable al data.frame `cars`. La transformación que permite pasar millas a kilómetros es `kilómetros=millas/0.62137` que en R se hace directamente con:

```
(cars$speed/0.62137) [1:10]
```

Finalmente, incluimos la nueva variable que llamaremos `velocidad` en `cars`:

```
cars$velocidad <- cars$speed / 0.62137
head(cars)
```

```
##   speed dist velocidad
## 1     4    2  6.437388
## 2     4   10  6.437388
## 3     7     4 11.265430
## 4     7   22 11.265430
## 5     8   16 12.874777
## 6     9   10 14.484124
```

También transformaremos la variable `dist` (en pies) en una nueva variable `distancia` (en metros), por lo que la transformación deseada es `metros=pies/3.2808`:

```
cars$distancia <- cars$dis / 3.2808
head(cars)
```

```
##   speed dist velocidad distancia
## 1     4    2  6.437388 0.6096074
## 2     4   10  6.437388 3.0480371
## 3     7     4 11.265430 1.2192148
## 4     7   22 11.265430 6.7056815
## 5     8   16 12.874777 4.8768593
## 6     9   10 14.484124 3.0480371
```

Ahora, eliminaremos las variables originales `speed` y `dist`, y guardaremos el data.frame resultante con el nombre `coches`. En primer lugar, veamos varias formas de acceder a las variables de interés:

```
cars[, c(3, 4)]
cars[, c("velocidad", "distancia")]
cars[, -c(1, 2)]
```

Utilizando alguna de las opciones anteriores se obtiene el data.frame deseado:

```
coches <- cars[, c("velocidad", "distancia")]
# head(coches)
str(coches)

## 'data.frame': 50 obs. of 2 variables:
## $ velocidad: num 6.44 6.44 11.27 11.27 12.87 ...
## $ distancia: num 0.61 3.05 1.22 6.71 4.88 ...
```

Finalmente, los datos anteriores podrían ser guardados en un fichero exportable a Excel con el siguiente comando:

```
write.csv2(coches, file = "coches.csv")
```

### 2.2.1.2 Recodificación de variables

Con el comando `cut()` podemos crear variables categóricas a partir de variables numéricas. El parámetro `breaks` permite especificar los intervalos para la discretización, puede ser un vector con los extremos de los intervalos o un entero con el número de intervalos. Por ejemplo, para categorizar la variable `cars$speed` en tres intervalos equidistantes podemos emplear<sup>1</sup>:

```
fspeed <- cut(cars$speed, 3, labels = c("Baja", "Media", "Alta"))
table(fspeed)
```

```
## fspeed
## Baja Media Alta
##    11     24     15
```

Para categorizar esta variable en tres niveles con aproximadamente el mismo número de observaciones podríamos combinar esta función con `quantile()`:

```
breaks <- quantile(cars$speed, probs = 0:3/3)
etiquetas3 <- c("Baja", "Media", "Alta")
fspeed <- cut(cars$speed, breaks, labels = etiquetas3)
table(fspeed)
```

```
## fspeed
## Baja Media Alta
##    17     16     15
```

Para otro tipo de recodificaciones podríamos emplear la función `ifelse()` vectorial:

```
fspeed <- ifelse(cars$speed < 15, "Baja", "Alta")
etiquetas2 <- c("Baja", "Alta")
fspeed <- factor(fspeed, levels = etiquetas2)
table(fspeed)
```

```
## fspeed
## Baja Alta
##    23     27
```

Alternativamente, en el caso de dos niveles podríamos emplear directamente la función `factor()`:

---

<sup>1</sup>Aunque si el objetivo es obtener las frecuencias de cada intervalo puede ser más eficiente emplear `hist()` con `plot = FALSE`.

```

fspeed <- factor(cars$speed >= 15,
                   labels = etiquetas2) # levels = c("FALSE", "TRUE")
table(fspeed)

## fspeed
## Baja Alta
##   23   27

```

En el caso de múltiples niveles, se podría emplear `ifelse()` anidados:

```

fspeed <- ifelse(cars$speed < 10, "Baja",
                  ifelse(cars$speed < 20, "Media", "Alta"))
fspeed <- factor(fspeed, levels = etiquetas3)
table(fspeed)

```

```

## fspeed
## Baja Media Alta
##   6    32    12

```

Otra alternativa, sería emplear la función `recode()` del paquete `car`.

```

library(car)
fspeed <- recode(cars$speed, "0:10 = 'Baja';
                  10:20 = 'Media';
                  else='Alta'
                  ")
fspeed <- factor(fspeed, levels = c("Baja", "Media", "Alta"))

```

NOTA: Para acceder directamente a las variables de un `data.frame` podríamos emplear la función `attach()` para añadirlo a la ruta de búsqueda y `detach()` al finalizar. Sin embargo esta forma de proceder puede causar numerosos inconvenientes, especialmente al modificar la base de datos, por lo que la recomendación sería emplear `with()`. Por ejemplo, podríamos calcular el factor anterior empleando:

```

fspeed <- with(cars, ifelse(speed < 10, "Baja",
                             ifelse(speed < 20, "Media", "Alta")))
fspeed <- factor(fspeed, levels = c("Baja", "Media", "Alta"))
table(fspeed)

## fspeed
## Baja Media Alta
##   6    32    12

```

## 2.2.2 Operaciones con casos

### 2.2.2.1 Ordenación

Continuemos con el data.frame `cars`. Se puede comprobar que los datos disponibles están ordenados por los valores de `speed`. A continuación haremos la ordenación utilizando los valores de `dist`. Para ello, utilizaremos el conocido como vector de índices de ordenación. Este vector establece el orden en que tienen que ser elegidos los elementos para obtener la ordenación deseada. Veamos primero un ejemplo sencillo:

```
x <- c(2.5, 4.3, 1.2, 3.1, 5.0) # valores originales
ii <- order(x)
ii      # vector de ordenación

## [1] 3 1 4 2 5
x[ii] # valores ordenados (por defecto, ascendente)
```

```
## [1] 1.2 2.5 3.1 4.3 5.0
```

En el caso de vectores, el procedimiento anterior se podría hacer directamente con:

```
sort(x)
```

Sin embargo, para ordenar tablas de datos será necesario la utilización del vector de índices de ordenación. A continuación, se muestran los datos de `cars` ordenados por `dist`:

```
ii <- order(cars$dist) # Vector de índices de ordenación
cars2 <- cars[ii, ]    # Datos ordenados por dist
head(cars2)

##   speed dist velocidad distancia
## 1      4    2  6.437388  0.6096074
## 3      7    4 11.265430  1.2192148
## 2      4   10  6.437388  3.0480371
## 6      9   10 14.484124  3.0480371
## 12     12   14 19.312165  4.2672519
## 5      8   16 12.874777  4.8768593
```

### 2.2.2.2 Filtrado

El filtrado de datos consiste en elegir una submuestra que cumpla determinadas condiciones. Para ello, se puede utilizar la función `subset(x, subset, select, drop = FALSE, ...)`, que además permite seleccionar variables con el argumento `select`.

A continuación se muestran un par de ejemplos:

```
# datos con dis>85
subset(cars, dist > 85)

##      speed dist velocidad distancia
## 47      24    92   38.62433  28.04194
## 48      24    93   38.62433  28.34674
## 49      24   120   38.62433  36.57644

# datos con speed en (10,15) y dist > 45
subset(cars, speed > 10 & speed < 15 & dist > 45)

##      speed dist velocidad distancia
## 19      13    46   20.92151  14.02097
## 22      14    60   22.53086  18.28822
## 23      14    80   22.53086  24.38430
```

También se pueden hacer el filtrado empleando directamente los correspondientes vectores de índices:

```
ii <- cars$dist > 85
cars[ii, ] # dis>85

##      speed dist velocidad distancia
## 47      24    92   38.62433  28.04194
## 48      24    93   38.62433  28.34674
## 49      24   120   38.62433  36.57644

ii <- cars$speed > 10 & cars$speed < 15 & cars$dist > 45
cars[ii, ] # speed en (10,15) y dist>45

##      speed dist velocidad distancia
## 19      13    46   20.92151  14.02097
## 22      14    60   22.53086  18.28822
## 23      14    80   22.53086  24.38430
```

En este caso, puede ser de utilidad la función `which()`:

```
it <- which(ii)
str(it)

##  int [1:3] 19 22 23
cars[it, ]

##      speed dist velocidad distancia
## 19      13    46   20.92151  14.02097
## 22      14    60   22.53086  18.28822
## 23      14    80   22.53086  24.38430

# rownames(cars[it, ])
id <- which(!ii)
```

```

str(cars[id, ])

## 'data.frame':   47 obs. of  4 variables:
## $ speed      : num  4 4 7 7 8 9 10 10 10 11 ...
## $ dist       : num  2 10 4 22 16 10 18 26 34 17 ...
## $ velocidad: num  6.44 6.44 11.27 11.27 12.87 ...
## $ distancia: num  0.61 3.05 1.22 6.71 4.88 ...
# Equivalente:
str(cars[-it, ])

## 'data.frame':   47 obs. of  4 variables:
## $ speed      : num  4 4 7 7 8 9 10 10 10 11 ...
## $ dist       : num  2 10 4 22 16 10 18 26 34 17 ...
## $ velocidad: num  6.44 6.44 11.27 11.27 12.87 ...
## $ distancia: num  0.61 3.05 1.22 6.71 4.88 ...
# ?which.min

```

Si se realiza una selección de variables como en:

```

cars[ii, "speed"]

```

```

## [1] 13 14 14

```

es posible que se quiera mantener la estructura original de los datos, para ello, bastaría con:

```

cars[ii, "speed", drop=FALSE]

```

```

##      speed
## 19      13
## 22      14
## 23      14
# subset(cars, ii, "speed") # equivalente

```

A veces puede ser necesario dividir (particionar) el conjunto de datos, uno para cada nivel de un grupo (factor), para ello se puede usar la función `split()`:

```

speed2 <- factor(cars$speed > 20, labels = c("slow","fast"))
table(speed2)

```

```

## speed2
## slow fast
##    43    7
cars2 <- split(cars,speed2)
class(cars2) # lista con 2 data.frames

## [1] "list"

```

```
sapply(cars2,class)

##      slow      fast
## "data.frame" "data.frame"

sapply(cars2,dim)

##      slow fast
## [1,]   43    7
## [2,]    4    4

cars2$fast

##      speed dist velocidad distancia
## 44     22    66  35.40564  20.11704
## 45     23    54  37.01498  16.45940
## 46     24    70  38.62433  21.33626
## 47     24    92  38.62433  28.04194
## 48     24    93  38.62433  28.34674
## 49     24   120  38.62433  36.57644
## 50     25    85  40.23368  25.90832
```

De forma inversa, podríamos recuperar el data.frame original con:

```
unsplit(cars2,speed2)
```

## 2.3 Datos faltantes

La problemática originada por los datos faltantes (*missing data*) en cualquier conjunto de datos subyace cuando se desea realizar un análisis estadístico, para más información en R, se puede consultar CRAN Task View: Missing Data

Vamos a ver un ejemplo, empleando el conjunto de datos `airquality` que contiene datos faltantes en sus dos primeras variables:

```
data("airquality")
datos <- airquality[,1:3]
summary(datos)

##      Ozone          Solar.R          Wind
##  Min.   : 1.00   Min.   : 7.0   Min.   : 1.700
##  1st Qu.:18.00   1st Qu.:115.8  1st Qu.: 7.400
##  Median :31.50   Median :205.0  Median : 9.700
##  Mean   :42.13   Mean   :185.9  Mean   : 9.958
##  3rd Qu.:63.25   3rd Qu.:258.8  3rd Qu.:11.500
##  Max.   :168.00  Max.   :334.0  Max.   :20.700
##  NA's    :37      NA's    :7
```

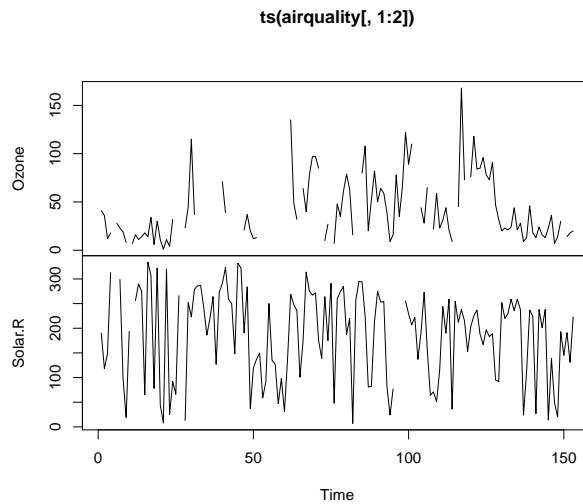
```
nrow(datos)

## [1] 153
# Datos faltantes por variable
sapply(datos, function(x) sum(is.na(x)))

##    Ozone Solar.R      Wind
##      37        7        0
```

A continuación se muestra la distribución de los datos perdidos en el data.frame (a lo largo del tiempo, por mes):

```
plot(ts(airquality[,1:2]))
```



¿Existe un patrón no aleatorio en los datos faltantes del ozono? Esta pregunta puede ser abordada parcialmente utilizando el test de Little (Little, 1988), disponible en la función `mcar_test()` del paquete `naniar`. Este test permite evaluar si los datos faltantes son generados por un mecanismo completamente aleatorio (MCAR). Si la hipótesis de MCAR es rechazada, esto sugiere que los datos faltantes podrían estar siguiendo un mecanismo MAR (*missing at random*) o MNAR (*non missing at random*).

Sin embargo, en muchos estudios, se omite el paso anterior y se procede directamente con alguno de los siguientes métodos:

- Análisis de casos completos (*complete cases*)
- Análisis de casos disponibles (borrado por parejas *pairwise cases*)
- Imputación de datos faltantes (por la media, mediana, último valor observado, vecino más cercano, valores predichos usando los datos observados....)

Siguiendo con el ejemplo, ante la presencia de datos faltantes, en R inicialmente no podemos conocer cómo se relacionan las tres primeras variables:"

```
cor(datos[,1:3])
```

```
##          Ozone Solar.R Wind
## Ozone      1      NA   NA
## Solar.R    NA      1   NA
## Wind       NA      NA    1
```

y requiere indicar cómo tratar los datos perdidos. Por ejemplo, una opción sería realizar un análisis sólo de los casos completos, eliminando todas las observaciones (filas) con algún dato faltante de nuestro conjunto de datos:

```
datosC <- na.omit(datos)
nrow(datosC) # n fija (sólo se utilizan 111 de las 153 de Wind)
```

```
## [1] 111
```

```
cor(datosC[,1:3])
```

```
##          Ozone Solar.R Wind
## Ozone    1.0000000 0.3483417 -0.6124966
## Solar.R  0.3483417 1.0000000 -0.1271835
## Wind     -0.6124966 -0.1271835 1.0000000
```

```
# otra forma de hacerlo sería:
# nrow(complete.cases(datos))
# cor(datos[,1:3], use = "complete.obs")
```

También, se podría usar toda la información disponible. El tamaño muestral  $n$  sería variable en función de los NA's de cada par de variables:

```
cor(datos[,1:3], use = "pairwise.complete.obs")
```

```
##          Ozone Solar.R Wind
## Ozone    1.0000000 0.34834169 -0.60154653
## Solar.R  0.3483417 1.00000000 -0.05679167
## Wind     -0.6015465 -0.05679167 1.00000000
```

Por ejemplo, ahora la correlación usa los 146 pares de observaciones disponibles para (Solar.R,Wind), en lugar de 111 del primer caso.

Por último, también se podría realizar una imputación (Van Buuren, 2018). A modo de ejemplo, en el siguiente código, se utiliza la media:

```
datosI <- datos
datosI$Ozone[is.na(datos$Ozone)] <- mean(datos$Ozone, na.rm = T)
datosI$Solar.R[is.na(datos$Solar.R)] <- mean(datosI$Solar.R, na.rm = T)
cor(datosI[,1:3])
```

```
##          Ozone Solar.R Wind
```

```
## Ozone      1.0000000  0.30296951 -0.53093584
## Solar.R   0.3029695  1.00000000 -0.05524488
## Wind       -0.5309358 -0.05524488  1.00000000
```

Notar que para el caso del ozono, se han sustituido los 37 *NA*'s (24% de las observaciones) por un único valor (de ahí que ahora la varianza sea menor a la observada inicialmente, algo que en principio, no sería deseable).

```
var(datos$Ozone,na.rm = T)
```

```
## [1] 1088.201
```

```
var(datosI$Ozone)
```

```
## [1] 823.3096
```

Los datos faltantes son una realidad común en muchos estudios, aunque nadie los desea. Para tratarlos correctamente, es esencial comprender cómo se obtuvieron los datos observados y por qué algunos datos no fueron registrados antes de iniciar cualquier otro análisis. No abordar adecuadamente los datos faltantes puede tener un efecto perjudicial en nuestro estudio, ya que las conclusiones obtenidas podrían ser no representativas o contener sesgos.

### 2.3.1 Funciones apply

#### 2.3.1.1 La función apply

Una forma de evitar la utilización de bucles es utilizando la sentencia `apply` que permite evaluar una misma función en todas las filas, columnas, etc. de un array de forma simultánea.

La sintaxis de esta función es:

```
apply(X, MARGIN, FUN, ...)
```

- `X`: matriz (o array).
- `MARGIN`: un vector indicando las dimensiones donde se aplicará la función. 1 indica filas, 2 indica columnas, y `c(1,2)` indica filas y columnas.
- `FUN`: función que será aplicada.
- `...`: argumentos opcionales que serán usados por `FUN`.

Veamos la utilización de la función `apply` con un ejemplo:

```
x <- matrix(1:9, nrow = 3)
x
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

```

apply(x, 1, sum)      # Suma por filas
## [1] 12 15 18
apply(x, 2, sum)      # Suma por columnas
## [1] 6 15 24
apply(x, 2, min)      # Mínimo de las columnas
## [1] 1 4 7
apply(x, 2, range)    # Rango (mínimo y máximo) de las columnas
##      [,1] [,2] [,3]
## [1,]     1     4     7
## [2,]     3     6     9

```

Alternativamente, se puede utilizar opciones más eficientes: `colSums()`, `rowSums()`, `colMeans()` y `rowMeans()`, como se muestra en el siguiente código de ejemplo:

```

x <- matrix(1:1e8, ncol = 10, byrow = FALSE)
t1 <- proc.time()
out<-apply(x, 2, mean)
proc.time() - t1

##      user  system elapsed
##  0.496   0.108   0.604

t2 <- proc.time()
out <- colMeans(x)
proc.time() - t2

##      user  system elapsed
##  0.080   0.000   0.079

```

### 2.3.1.2 Variantes de la función `apply`

- La función `lapply(X, FUN, ...)` aplica la función `FUN` a cada elemento de una lista en R y devuelve una lista como resultado (sin necesidad de especificar el argumento `MARGIN`). Notar que todas las estructuras de datos en R pueden convertirse en listas, por lo que `lapply()` puede utilizarse en más casos que `apply()`.

```

# lista con las medianas de las variables
list <- lapply(cars, median)
str(list)

## List of 4
## $ speed    : num 15

```

```
## $ dist      : num 36
## $ velocidad: num 24.1
## $ distancia: num 11
```

- b. La función `sapply(X, FUN, ..., simplify = TRUE, USE.NAMES = TRUE)` permite iterar sobre una lista o vector (alternativa más eficiente a un `for`):

```
# matriz con las medias, medianas y desv. de las variables
res <- sapply(cars,
              function(x) c(mean = mean(x),
                           median = median(x),
                           sd = sd(x)))
# str(res)
res
```

```
##           speed      dist velocidad distancia
## mean    15.400000 42.98000 24.783945 13.100463
## median 15.000000 36.00000 24.140206 10.972933
## sd     5.287644 25.76938  8.509655  7.854602
```

```
cfuns <- function(x, funs = c(mean, median, sd))
       sapply(funs, function(f) f(x))
x <- 1:10
cfuns(x)
```

```
## [1] 5.50000 5.50000 3.02765
```

```
sapply(cars, cfuns)
```

```
##           speed      dist velocidad distancia
## [1,] 15.400000 42.98000 24.783945 13.100463
## [2,] 15.000000 36.00000 24.140206 10.972933
## [3,] 5.287644 25.76938  8.509655  7.854602
```

```
nfuns <- c("mean", "median", "sd")
sapply(nfuns,
       function(f) eval(parse(text = paste0(f, "(x)))))
```

```
##   mean median      sd
## 5.50000 5.50000 3.02765
```

- c. La función `tapply()` es similar a la función `apply()` y permite aplicar una función a los datos desagregados, utilizando como criterio los distintos niveles de una variable factor. Es decir, facilita la creación de tablas resumen por grupos. La sintaxis de esta función es como sigue:

```
tapply(X, INDEX, FUN, ...)
```

- X: matriz (o array).
- INDEX: factor indicando los grupos (niveles).

- FUN: función que será aplicada.
- ....: argumentos opcionales .

Consideremos, por ejemplo, el data.frame `ChickWeight` con datos de un experimento relacionado con la repercusión de varias dietas en el peso de pollos.

```
data(ChickWeight)
# str(ChickWeight)
head(ChickWeight)

##   weight Time Chick Diet
## 1     42    0     1    1
## 2     51    2     1    1
## 3     59    4     1    1
## 4     64    6     1    1
## 5     76    8     1    1
## 6     93   10     1    1

peso <- ChickWeight$weight
dieta <- ChickWeight$Diet
levels(dieta) <- c("Dieta 1", "Dieta 2", "Dieta 3", "Dieta 4")
tapply(peso, dieta, mean) # Peso medio por dieta

## Dieta 1 Dieta 2 Dieta 3 Dieta 4
## 102.6455 122.6167 142.9500 135.2627

tapply(peso, dieta, summary)

## $`Dieta 1`
##   Min. 1st Qu. Median   Mean 3rd Qu.   Max.
## 35.00    57.75   88.00 102.65 136.50 305.00
##
## $`Dieta 2`
##   Min. 1st Qu. Median   Mean 3rd Qu.   Max.
## 39.00    65.50 104.50 122.60 163.00 331.00
##
## $`Dieta 3`
##   Min. 1st Qu. Median   Mean 3rd Qu.   Max.
## 39.00    67.50 125.50 142.90 198.80 373.00
##
## $`Dieta 4`
##   Min. 1st Qu. Median   Mean 3rd Qu.   Max.
## 39.00    71.25 129.50 135.26 184.75 322.00
```

Alternativamente, se podría emplear la función `aggregate()` que tiene las ventajas de admitir fórmulas y disponer de un método para series de tiempo.

```
help(aggregate)
aggregate(peso, by=list(dieta=dieta), FUN = "mean" )
```

```

##      dieta      x
## 1 Dieta 1 102.6455
## 2 Dieta 2 122.6167
## 3 Dieta 3 142.9500
## 4 Dieta 4 135.2627
aggregate(peso~dieta,FUN = "summary" ) # con formula

##      dieta peso.Min. peso.1st Qu. peso.Median peso.Mean
## 1 Dieta 1    35.0000    57.7500    88.0000  102.6455
## 2 Dieta 2    39.0000   65.5000   104.5000  122.6167
## 3 Dieta 3    39.0000   67.5000   125.5000  142.9500
## 4 Dieta 4    39.0000   71.2500   129.5000  135.2627
##      peso.3rd Qu. peso.Max.
## 1       136.5000 305.0000
## 2       163.0000 331.0000
## 3       198.7500 373.0000
## 4       184.7500 322.0000

```

### 2.3.2 Tablas (para informes)

a. Tablas con `kable()`:

A continuación, se muestra un ejemplo, de tabla resumen, con las medias, medianas y desviación típica de las variables:

```

res <- sapply(cars,
               function(x) c(mean = mean(x),
                           median = median(x),
                           sd = sd(x)))
knitr::kable(t(res), digits = 1,
             col.names = c("Media", "Mediana", "Desv. típica"))

```

	Media	Mediana	Desv. típica
speed	15.4	15.0	5.3
dist	43.0	36.0	25.8
velocidad	24.8	24.1	8.5
distancia	13.1	11.0	7.9

Y en este segundo ejemplo, se muestra el resumen de un modelo de regresión lineal simple (distancia de frenado en función de la velocidad del vehículo):

```

modelo <- lm(dist ~ speed, data = cars)
coefs <- coef(summary(modelo))
knitr::kable(coefs, escape = FALSE, digits = 5)

```

	Estimate	Std. Error	t value	Pr(> t )
(Intercept)	-17.57909	6.75844	-2.60106	0.01232
speed	3.93241	0.41551	9.46399	0.00000

b. Tablas interactivas con `datatable()` del paquete DT:

```
library(DT)
datatable(iris,options = list(scrollX = TRUE))
```

Hay muchos otros paquetes de R que se pueden utilizar para generar tablas como: `kableExtra()`, `flextable()`, `reactable()`, `reactablefmtr()`, `formattable()`, `gt()` y `tinytable()`.

### 2.3.3 Operaciones con tablas de datos

*Unir tablas:*

- `rbind()`: combina vectores, matrices, arrays o data.frames por filas.
- `cbind()`: Idem por columnas.
- `merge()`: Fusiona dos data.frame por columnas o nombres de fila comunes. También permite otras operaciones de unión (*join*) de bases de datos, algunas de ellas se verán con más detalle en el Capítulo 4.

*Combinar tablas:*

- `match(x, table)` devuelve un vector (de la misma longitud que `x`) con las (primeras) posiciones de coincidencia de `x` en `table` (o `NA`, por defecto, si no hay coincidencia).

Para realizar consultas combinando tablas puede ser más cómodo el operador `%in%` (?`'%in%'`).

- `pmatch(x, table, ...)`: similar al anterior pero con coincidencias parciales de cadenas de texto.

## 2.4 Ejemplo WoS data

Ejemplo `wosdata.R` en `wosdata.zip`. Ver Apéndice ??.

```
# library(dplyr)
# library(stringr)
# https://rubenfcasal.github.io/scimetr/articles/scimetr.html
# library(scimetr)

db <- readRDS("data/wosdata/db_udc_2015.rds")
str(db, 1)

## List of 11
## $ Docs      :'data.frame': 856 obs. of  26 variables:
## $ Authors   :'data.frame': 4051 obs. of  4 variables:
## $ AutDoc    :'data.frame': 5511 obs. of  2 variables:
## $ Categories:'data.frame': 189 obs. of  2 variables:
```

```

## $ CatDoc    :'data.frame': 1495 obs. of  2 variables:
## $ Areas     :'data.frame': 121 obs. of  2 variables:
## $ AreaDoc   :'data.frame': 1364 obs. of  2 variables:
## $ Addresses :'data.frame': 3655 obs. of  5 variables:
## $ AddAutDoc :'data.frame': 7751 obs. of  3 variables:
## $ Journals  :'data.frame': 520 obs. of  12 variables:
## $ label     : chr ""
## - attr(*, "variable.labels")= Named chr [1:62] "Publication type" "Author" "Book au
## ..- attr(*, "names")= chr [1:62] "PT" "AU" "BA" "BE" ...
## - attr(*, "class")= chr "wos.db"
variable.labels <- attr(db, "variable.labels")
knitr::kable(as.data.frame(variable.labels),
             caption = "Variable labels")

```

Veamos ahora un par de ejemplos, en el primero se buscan los documentos correspondientes a revistas (que contiene `Chem` en el título de la revista *journal*). Para ello utilizamos la función `grepl()` que busca las coincidencias con el patrón `Chem` dentro de cada elemento de un vector de caracteres.

```

# View(db$Journals)
iidj <- with(db$Journals, idj[grepl('Chem', JI)])
db$Journals$JI[iidj]

## [1] "J. Am. Chem. Soc."
## [2] "Inorg. Chem."
## [3] "J. Chem. Phys."
## [4] "J. Chem. Thermodyn."
## [5] "J. Solid State Chem."
## [6] "Chemosphere"
## [7] "Antimicrob. Agents Chemother."
## [8] "Trac-Trends Anal. Chem."
## [9] "Eur. J. Med. Chem."
## [10] "J. Chem. Technol. Biotechnol."
## [11] "J. Antimicrob. Chemother."
## [12] "Food Chem."
## [13] "Cancer Chemother. Pharmacol."
## [14] "Int. J. Chem. Kinet."
## [15] "Chem.-Eur. J."
## [16] "J. Phys. Chem. A"
## [17] "New J. Chem."
## [18] "Chem. Commun."
## [19] "Chem. Eng. J."
## [20] "Comb. Chem. High Throughput Screen"
## [21] "Mini-Rev. Med. Chem."
## [22] "Phys. Chem. Chem. Phys."
## [23] "Org. Biomol. Chem."

```

Tabla 2.1: Variable labels

	variable.labels
PT	Publication type
AU	Author
BA	Book authors
BE	Editor
GP	Group author
AF	Author full
BF	Book authors fullname
CA	Corporate author
TI	Title
SO	Publication name
SE	Series title
BS	Book series
LA	Language
DT	Document type
CT	Conference title
CY	Conference year
CL	Conference place
SP	Conference sponsors
HO	Conference host
DE	Keywords
ID	Keywords Plus
AB	Abstract
C1	Addresses
RP	Reprint author
EM	Author email
RI	Researcher id numbers
OI	Orcid numbers
FU	Funding agency and grant number
FX	Funding text
CR	Cited references
NR	Number of cited references
TC	Times cited
Z9	Total times cited count
U1	Usage Count (Last 180 Days)
U2	Usage Count (Since 2013)
PU	Publisher
PI	Publisher city
PA	Publisher address
SN	ISSN
EI	eISSN
BN	ISBN
J9	Journal.ISI
JI	Journal.ISO
PD	Publication date
PY	Year published
VL	Volume
IS	Issue
PN	Part number
SU	Supplement
SI	Special issue
MA	Meeting abstract

```

## [24] "J. Chem Inf. Model."
## [25] "ACS Chem. Biol."
## [26] "Environ. Chem. Lett."
## [27] "Anal. Bioanal. Chem."
## [28] "J. Cheminformatics"
## [29] "J. Mat. Chem. B"

idd <- with(db$Docs, idj %in% iidj)
which(idd)

## [1] 2 4 16 23 43 69 119 126 138 175 188 190 203 208
## [15] 226 240 272 337 338 341 342 357 382 385 386 387 388 394
## [29] 411 412 428 460 483 518 525 584 600 604 605 616 620 665
## [43] 697 751 753 775 784 796 806 808 847 848

# View(db$Docs[idd, ])
head(db$Docs[idd, -3])

##      idd idj      PT      DT  NR TC Z9 U1 U2      PD      PY VL
## 2     2 37 Journal Article 45 5 5 0 0 DEC 21 2015 54
## 4     4 272 Journal Article 78 2 2 4 21 DEC 14 2015 21
## 16    16 195 Journal Article 34 2 2 0 0      DEC 2015 70
## 23    23 436 Journal Article 48 3 3 0 4      DEC 2015 10
## 43    43 455 Journal Review 214 0 0 0 8      DEC 2015 13
## 69    69 37 Journal Article 86 2 2 8 28 NOV 2 2015 54
##      IS PN SU SI MA      BP      EP AR
## 2     24           11680 11687
## 4     51           18662 18670
## 16    12           3222 3229
## 23    12           2850 2860
## 43    4            413 430
## 69    21           10342 10350
##          DI D2 PG      UT an
## 2 10.1021/acs.inorgchem.5b01652     8 367118100013 9
## 4           10.1002/chem.201502937   9 368280400026 8
## 16          10.1093/jac/dkv262     8 368246800008 10
## 23 10.1021/acschembio.5b00624   11 366875400020 10
## 43 10.1007/s10311-015-0526-2   18 365096700004 2
## 69 10.1021/acs.inorgchem.5b01719   9 364175000028 8

```

En este segundo ejemplo, se buscan los documentos correspondientes a autores (que contiene Abad en su nombre):

```

# View(db$Authors)
iida <- with(db$Authors, ida[grep1('Abad', AF)])
db$Authors$AF[iida]

## [1] "Mato Abad, Virginia" "Abad, Maria-Jose"
## [3] "Abad Vicente, J."    "Abada, Sabah"

```

```
idd <- with(db$AutDoc, idd[ida %in% iida])
idd

## [1] 273 291 518 586
# View(db$Docs[idd, ])
head(db$Docs[idd, -3])

##      idd idj      PT          DT  NR TC Z9 U1 U2      PD
## 273 273 282 Journal Article 107  8  8  0  0     SEP
## 291 291 141 Journal Meeting Abstract  0  0  0  0  1 AUG 1
## 518 518 272 Journal Article 103  4  4  0  0 APR 20
## 586 586 311 Journal Article  32  2  2  2 19     APR
##      PY VL      IS PN SU SI    MA   BP   EP AR
## 273 2015 42 15-16          6205 6214
## 291 2015 36           1 P167   9   9
## 518 2015 21    17          6535 6546
## 586 2015 26    4            369  375
##                  DI D2 PG          UT an
## 273 10.1016/j.eswa.2015.03.011 10 355063700018  7
## 291                   1 361205101026 10
## 518      10.1002/chem.201500155 12 352796100030 10
## 586      10.1002/pat.3462    7 351472700012  6
```



# Capítulo 3

## Introducción al lenguaje SQL

Los sistemas de información gestionan repositorios de información en múltiples formatos, siendo el más popular las bases de datos relacionales a las que se accede mediante SQL (Structured Query Language).

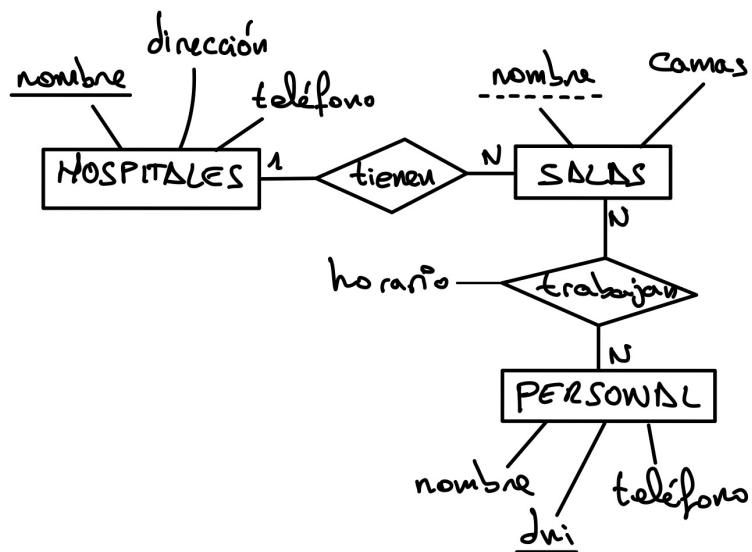
El ejemplo que trabajaremos en este capítulo está disponible en Kaggle:  
[kaggle.com/code/diegodx/txd-2025-tutorialsql](https://kaggle.com/code/diegodx/txd-2025-tutorialsql)

### 3.1 Bases de Datos Relacionales

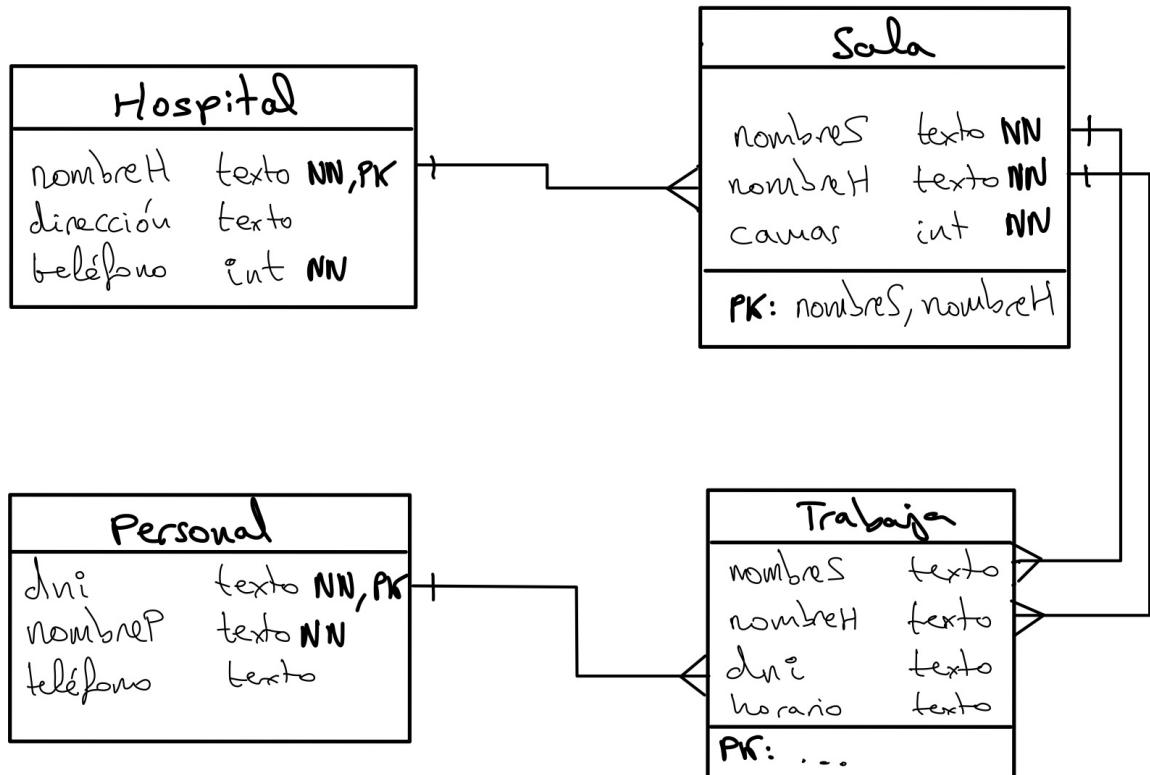
#### 3.1.1 Definiciones

- **Dominio:** contexto (organización, empresa, evento...) objeto de gestión de la información.
- **Dato:** hecho con significado implícito, registable, relevante en un determinado dominio.
- **Base de datos:** colección de datos de un determinado dominio relacionados entre sí, organizados de forma que sea posible manipularlos y recuperarlos de forma eficiente.
- Sistema de Gestión de Bases de Datos (**SGBD**) (en inglés **RDBMS**, Relational Database Management System): software que permite a los usuarios crear y manipular bases de datos mediante operaciones **CRUD**:
  - **Create:** Crear / Insertar datos
  - **Rread:** Consultar / Leer datos
  - **Update:** Actualizar / Modificar datos
  - **Delete:** Eliminar datos

- **Modelo de datos:** abstracción conceptual que propone una manera de organizar y manipular los datos. Definido mediante:
  - Estructura: elementos para organizar datos
  - Integridad: reglas para relaciones los elementos
  - Manipulación: operaciones sobre los datos adaptadas a la estructura y reglas
- Modelo **Entidad Relación** (entidades, relaciones, atributos)

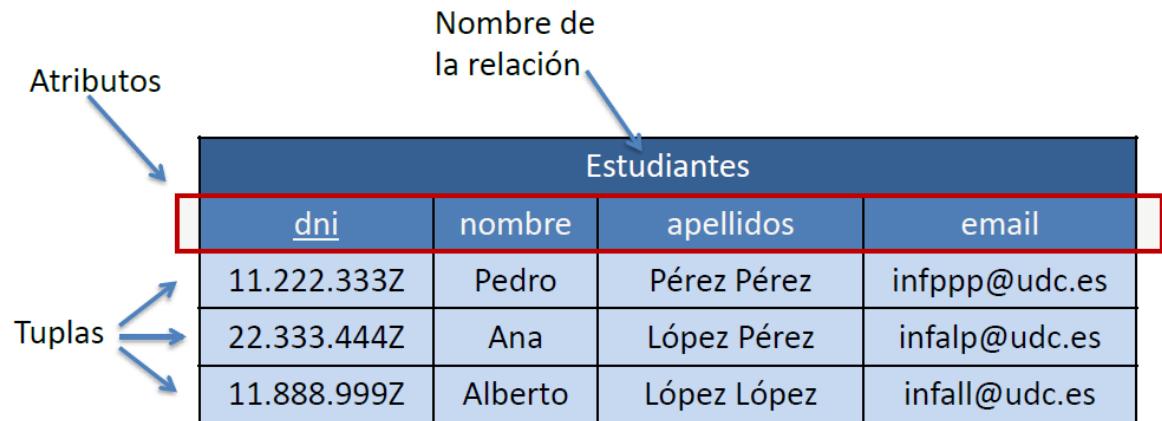


- Modelo de datos lógico o de representación (**modelo relacional** de Codd)
  - Datos en relaciones (tablas)
  - Base matemática formal
  - Flexible

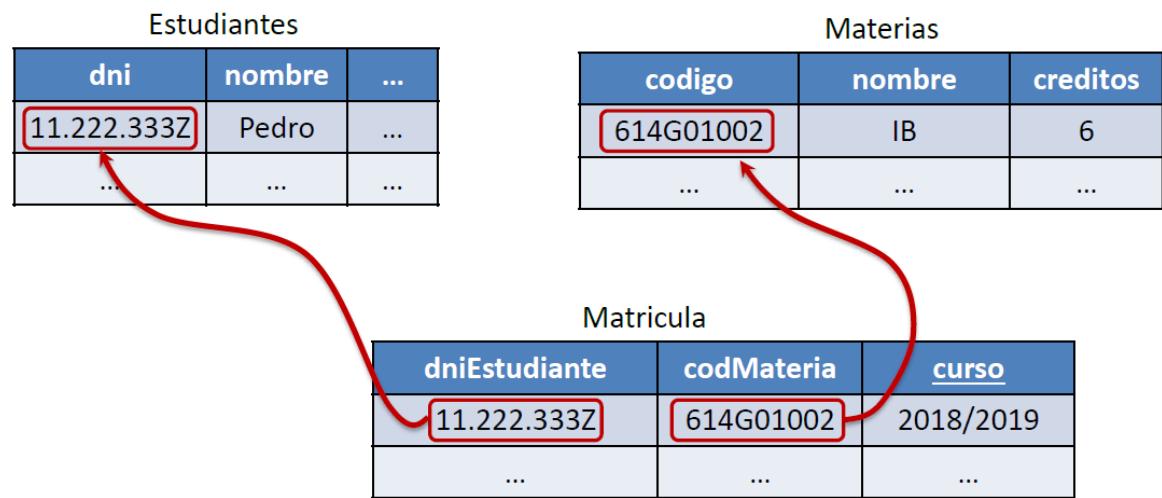


- Modelo de datos físico (tal y como se almacenan los datos)

Una fila de la tabla (relación) es una tupla y una columna un atributo.



Una base de datos es un conjunto de tablas (al menos una).



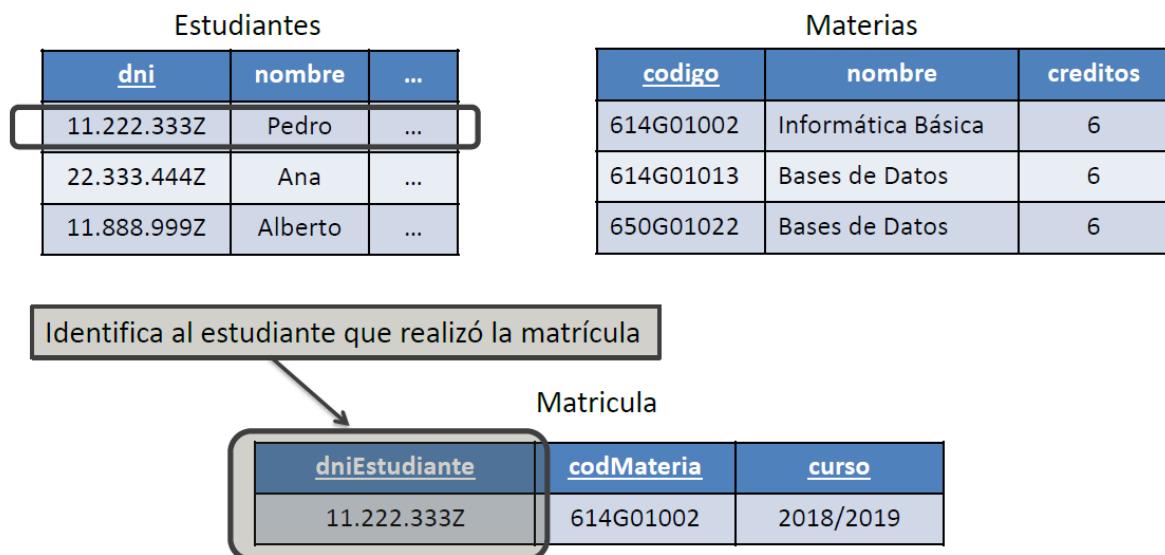
La tabla no es una relación porque la relación es un conjunto sin orden y una tabla puede tener filas repetidas y tiene orden.

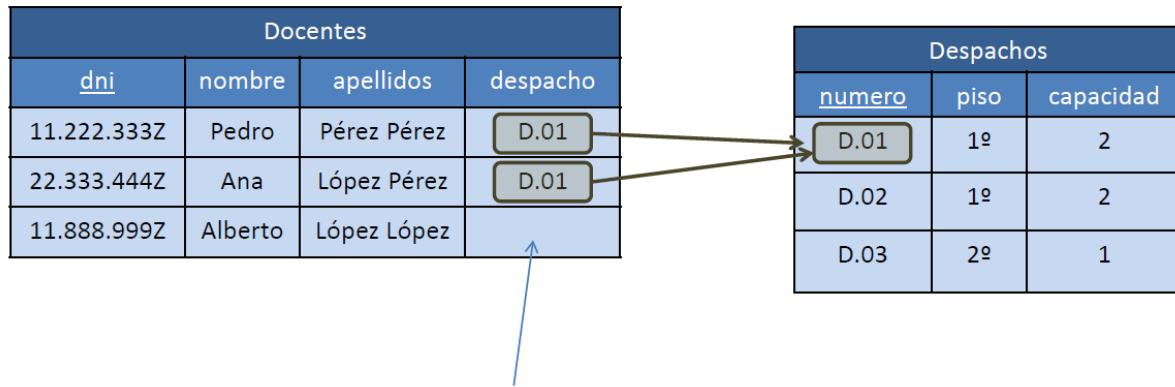
- 
- **Esquema:** estructura de la base de datos
  - **Estado:** contenido de la base de datos
  - Restricción de **integridad:** regla que debe cumplir la información registrada en la base de datos para garantizar la integridad de la información.

## 3.2 Restricciones

Cualquier Base de Datos basada en el modelo relacional debería cumplir como mínimo estas restricciones (además de las propias del dominio):

- **Restricción de dominio:** Cada atributo debe tener un tipo de valores permitido, asegurando que sólo se almacenan datos válidos y consistentes
- **Atributos atómicos:** Cada atributo debe almacenar valores indivisibles (nombre completo descomponible en nombre y apellidos, domicilio en calle, CP, localidad, etc...)
- **Unicidad:** No pueden existir dos tuplas iguales. Para ello se definen claves:
  - Una **superclave** es un subconjunto de atributos tal que no existen dos tuplas con la misma superclave. > Ejercicio. En la relación Empleado(dni, nombre, apellidos, email) ¿cuántas superclaves existen?
  - Una **clave candidata** es una superclave mínima (superclave mínima es la clave a la que no se le puede eliminar un atributo). > ¿Cuántas claves candidatas hay en el ejemplo anterior?
  - La **clave primaria** es la clave candidata que elegimos que identificar de forma única las tuplas de una relación. Restricción de integridad de entidad: Ningún valor de la clave primaria puede ser un valor nulo.
- Una **clave foránea** es un conjunto de atributos de una relación R<sub>1</sub> que, para cada tupla, identifican a otra tupla de una relación R<sub>2</sub> con la que está relacionada. La Restricción de integridad referencial nos dice que la clave foránea ha de corresponderse con la clave primaria de R<sub>2</sub>, y si la clave foránea no es nula ha de referir a una tupla en R<sub>2</sub>.





La foránea *despacho* toma valor nulo en esta tupla  
 (El docente no tiene todavía despacho asignado)  
**ES CORRECTO Y NO VIOLA LA RESTRICCIÓN**

Si borramos/actualizamos un valor de clave foránea podemos: (a) prohibir el cambio, o (b) poner a nulo la clave foránea (borrado) o propagar el cambio (modificación).

### 3.3 Sistemas Gestores de Bases de Datos (SGDB)

Utilizar un SGDB tiene múltiples ventajas:

- Administración centralizada de los datos (por un administrador en un servidor/plataforma central que evita la información en silos -redundante/inconsistente)
- Desacoplamiento del almacenamiento físico de los datos (no es necesario conocerlo)
- Simplicidad de acceso (ODBC + SQL, lenguaje declarativo)
- Control de integridad (restricciones genéricas, integridad de entidad y referencial, de dominio, y las del dominio en software)
- Control de acceso concurrente (evita inconsistencia)
- Seguridad (autenticación, roles de acceso)
- Recuperación ante fallos (backup, logs y transacciones -rollback-)

Existen muchos SGDBs, pero los más populares son:

- **SQLite:** es muy ligero, no necesita un servidor para ejecutarse y es muy rápido cuando el conjunto de datos es pequeño.
  - Su escalabilidad y concurrencia son muy limitadas
  - Ideal para proyectos pequeños, aplicaciones locales o móviles y prácticas de TxD

- Usado en Google Chrome, Firefox, Safari, Dropbox (app de escritorio)
- Es de dominio público
- **MySQL:** requiere un servidor para ejecutarse, cuenta con soporte en la nube
  - Presenta buena escalabilidad y concurrencia
  - Es muy utilizado en aplicaciones web de tamaño medio
  - Usado en Wordpress, y en general un gran porcentaje de páginas y aplicaciones web de tamaño pequeño y mediano. Incluso sitios grandes como Wikipedia o Facebook emplean variantes de MySQL.
  - Licencia GPL (con variantes comerciales)
- **PostgreSQL:** también requiere un servidor y tiene soporte en la nube
  - Su escalabilidad es excelente, óptimo para grandes cantidades de datos
  - Cuenta con muchas extensiones para, por ejemplo, información geográfica (PostGIS), series temporales (TimescaleDB), etc.
  - Ideal para sistemas complejos, analíticos o con alta concurrencia
  - Usado en Reddit, Instagram, Spotify, Netflix, ...
  - Licencia propia tipo MIT
- Otros SGDBs relacionales: MariaDB (derivado de MySQL), Microsoft SQL Server, Oracle

Ranking de popularidad según DB-Engines. La puntuación está calculada en función del número de menciones en páginas web, interés general en los sistemas, frecuencia de discusiones técnicas, cantidad de ofertas de trabajo, relevancia en redes sociales, etc.

2025	2024	SGDB	Modelo	Puntuacion	Var
1.	1.	Oracle	Relacional, Multi-modelo	1212.77	-96.67
2.	2.	MySQL	Relacional, Multi-modelo	879.66	-
					143.09
3.	3.	Microsoft SQL Server	Relacional, Multi-modelo	715.05	-87.04
4.	4.	PostgreSQL	Relacional, Multi-modelo	643.20	-8.96
5.	5.	MongoDB	Documental, Multi-modelo	368.01	-37.20
6.	7.	Snowflake	Relacional	198.65	+58.05
7.	6.	Redis	Clave-valor, Multi-modelo	142.33	-7.30
8.	14.	Databricks	Multi-modelo	128.80	+43.21
9.	9.	IBM Db2	Relacional, Multi-modelo	122.37	-0.40
10.	8.	Elasticsearch	Multi-modelo	116.67	-15.18

2025	2024	SGDB	Modelo	Puntuacion	Var
11.	11.	Apache Casandra	Wide column, Multi-modelo	105.16	+7.56
12.	10.	SQLite	Relacional	104.56	+2.64
13.	15.	MariaDB	Relacional, Multi-modelo	87.77	+2.88
14.	12.	Microsoft Access	Relacional	80.79	-11.36
15.	17.	Amazon Dyna-moDB	Multi-modelo	75.91	+4.06

### 3.4 Sintaxis SQL

SQL (Structured Query Language) es un lenguaje declarativo. Es un lenguaje estándar: tiene un estándar oficial definido por ISO y ANSI. Sin embargo, en la práctica cada SGDB implementa solo una parte de él, y además tiene sus propios dialectos, en los que puede variar por ejemplo lo siguiente:

- Tipos de datos (TEXT, VARCHAR, BLOB, etc.)
- Distinción o no de mayúsculas y minúsculas (cap sensitiveness)
- Cómo se manejan las transacciones
- Funciones (LENGTH(), LEN(), etc.)
- Formato de fechas
- Gestión de triggers
- ...

Sintaxis general:

- Consulta

```
SELECT <campo/s>
FROM <tabla>
WHERE <condición>
GROUP BY <campo>
HAVING <condición>
ORDER BY <campo>
LIMIT <m> OFFSET <n>
```

- Modificación

```
UPDATE <tabla>
SET <cambios>
WHERE <condición>
```

- Borrado

```
DELETE FROM <tabla>
WHERE <condición>
```

## 3.5 Cláusulas básicas de SQL

### 3.5.1 Lectura

- Seleccionar todas las columnas de una tabla:

```
SELECT * FROM Track;
```

- Seleccionar columnas específicas:

```
SELECT name, composer FROM Track;
```

- Alias de columna y tabla:

```
SELECT name AS Canción FROM Track;
```

```
SELECT T.name FROM Track AS T;
```

- Funciones de agregación:

```
SELECT COUNT(*), SUM(UnitPrice), MIN(UnitPrice), MAX(UnitPrice) FROM Track;
```

```
SELECT AVG(milliseconds) AS 'Duración Media' FROM Track;
```

- Filtrado de duplicados:

```
SELECT DISTINCT FirstName FROM Customer;
```

```
SELECT COUNT(DISTINCT FirstName) FROM Customer;
```

- Formato:

```
SELECT CONCAT(FirstName, ' ', LastName) AS Nombre FROM Employee;
```

```
SELECT (FirstName || ' ' || LastName) AS Nombre FROM Employee;
```

```
SELECT ROUND(AVG(Total), 2) AS 'Facturación Media' FROM Invoice;
```

#### 3.5.1.1 Filtrado de Resultados

- Seleccionar filas con condiciones:

```
SELECT name FROM Track
WHERE UnitPrice < 2.0;
```

- Múltiples condiciones:

- Operadores: ‘AND’, ‘OR’, ‘LIKE’, ‘NOT’, ‘IS NULL’, ‘IS NOT NULL’ ‘BETWEEN x AND y’, ‘IN (lista)’

```
SELECT name FROM Track
WHERE milliseconds > 120000 AND UnitPrice < 2.0;
```

```
SELECT name FROM Track
WHERE composer LIKE 'Metallica' OR composer LIKE 'Ulrich';
```

- Coincidencias parciales:

```
SELECT * FROM Track
WHERE name LIKE '%Love%';
```

- ‘%’ : Reemplazo por un conjunto de caracteres
- ‘\_’ : Reemplazo por un caracter
- Rangos:

```
SELECT * FROM Track
WHERE UnitPrice BETWEEN 0.5 AND 1.5;
```

- Valores en una lista:

```
SELECT * FROM Track
WHERE composer IN ('Metallica', 'Ulrich');
```

### 3.5.1.2 Ordenar Resultados

- Ordenar por una columna:

```
SELECT * FROM Track
ORDER BY title ASC;
```

```
SELECT * FROM Track
ORDER BY title DESC;
```

- Ordenar por múltiples columnas:

```
SELECT * FROM Track
ORDER BY composer ASC, title DESC;
```

### 3.5.1.3 Número de filas (paginación)

- Obtener las N primeras:

```
SELECT * FROM Track
ORDER BY title ASC
LIMIT 5;
```

- Obtener las N filas siguientes:

```
SELECT * FROM Track
ORDER BY title ASC
OFFSET 5 LIMIT 5;
```

## 3.6 Gestión de tablas

- Creación de tablas

```
CREATE TABLE table_name(
column1 datatype,
column2 datatype,
column3 datatype,
.....
columnN datatype,
PRIMARY KEY( one or more columns )
);
```

- Borrado de tablas

```
DROP TABLE table_name;
```

- Creación de índices

```
CREATE UNIQUE INDEX index_name
ON table_name ( column1, column2,...columnN );
```

- Modificación de tablas

```
ALTER TABLE table_name
DROP INDEX index_name;
```

```
ALTER TABLE table_name
{ADD|DROP|MODIFY} column_name {data_type};
```

```
ALTER TABLE table_name RENAME TO new_table_name;
```

## 3.7 Gestión de datos

- Inserción de tuplas

```
INSERT INTO table_name( column1, column2,...columnN )
VALUES ( value1, value2,...valueN );
```

- Modificación de datos

```
UPDATE table_name
SET column1 = value1, column2 = value2,...columnN=valueN
[ WHERE CONDITION ];
```

- Eliminación de datos

```
DELETE FROM table_name
WHERE {CONDITION};
```

## 3.8 Gestión de Bases de Datos

- Creación de una base de datos

```
CREATE DATABASE database_name;
```

- Eliminación de una base de datos

```
DROP DATABASE database_name;
```

- Selección de base de datos

```
USE database_name;
```

- Gestión de transacciones

```
BEGIN;
```

```
...
```

```
COMMIT;
```

```
ROLLBACK;
```

## 3.9 Ejemplos de consultas SQL

```
SELECT Nombre, Apellido1, Apellido2, Municipio, Provincia
FROM Cliente
WHERE Municipio = 'Lugo'
ORDER BY Apellido1

INSERT Proveedor(Nombre, PersonaContacto, Ciudad, País)
VALUES ('Café Candelas', 'Ivana Candelas', 'Lugo', 'España')

UPDATE Pedidos
SET Cantidad = 2
WHERE IdProducto = 963

DELETE Cliente
WHERE Email = 'alexandregb@gmail.com'
```

## 3.10 Conexión con bases de datos desde R

### 3.10.1 Introducción a SQL en R

SQL se usa para manipular datos dentro de una base de datos. Si la base de datos no es muy grande se puede cargar toda en un data.frame. No obstante, por escalabilidad y offloading de la carga de trabajo al servidor SGBD utilizaremos SQL.

Existen varios SGBD (SQLite, Microsoft SQL Server, MySQL, PostgreSQL, etc) los cuales comparten el soporte de SQL (en concreto ANSI SQL) aunque cada gestor extiende SQL de formas sutiles buscando minar cierta portabilidad de código (*vendor-locking*). En efecto, un código SQL desarrollado para SQLite es probable que falle con MySQL aunque tras aplicar ligeras modificaciones ya funcionará. Asimismo el mecanismo de conexión, configuración, rendimiento y operación suele diferir entre SGBD.

A continuación se lista una serie de paquetes utilizados en el acceso a los datos, lo que suele ser el principal esfuerzo a realizar cuando se trabaja con SGBD:

- DBI
- RODBC
- dbConnect
- RSQLite
- RMySQL
- RPostgreSQL

### 3.10.2 El paquete sqldf

A continuación se presenta una serie de ejercicios con la sintaxis de SQL operando sobre un data.frame con el paquete sqldf. Esto inicialmente no incluye los detalles de conectarse a un SGBD, ni modificar los datos, solamente el uso de SQL para extraer datos con el objetivo de ser analizados en R.

```
library(sqldf)

sqldf('SELECT age, circumference FROM Orange WHERE Tree = 1 ORDER BY circumference ASC')

##      age circumference
## 1    118            30
## 2    484            58
## 3    664            87
## 4   1004           115
## 5   1231           120
## 6   1372           142
## 7   1582           145
```

### 3.10.3 SQL Queries

El comando inicial es SELECT. SQL no es case-sensitive, por lo que esto va a funcionar:

```
sqldf("SELECT * FROM iris")
sqldf("select * from iris")
```

pero lo siguiente no va a funcionar (a menos que tengamos un objeto IRIS):

```
sqldf("SELECT * FROM IRIS")
```

La sintaxis básica de SELECT es:

```
SELECT variable1, variable2 FROM data
```

#### 3.10.3.1 Asterisco/Wildcard

Lo extrae todo

```
bod2 <- sqldf('SELECT * FROM BOD')
```

#### 3.10.3.2 Limit

Limita el número de resultados

```
sqldf('SELECT * FROM iris LIMIT 5')
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
```

#### 3.10.3.3 Order By

Ordena las variables

```
ORDER BY var1 {ASC/DESC}, var2 {ASC/DESC}
```

```
sqldf("SELECT * FROM Orange ORDER BY age ASC, circumference DESC LIMIT 5")
```

```
##   Tree age circumference
## 1    2 118            33
## 2    4 118            32
## 3    1 118            30
## 4    3 118            30
## 5    5 118            30
```

### 3.10.3.4 Where

Sentencias condicionales, donde se puede incorporar operadores lógicos AND y OR, expresando el orden de evaluación con paréntesis en caso de ser necesario.

```
sqldf('SELECT demand FROM BOD WHERE Time < 3')

##   demand
## 1     8.3
## 2    10.3

sqldf('SELECT * FROM rock WHERE (peri > 5000 AND shape < .05) OR perm > 1000')

##   area      peri      shape      perm
## 1 5048  941.543  0.328641  1300
## 2 1016   308.642  0.230081  1300
## 3 5605 1145.690  0.464125  1300
## 4 8793 2280.490  0.420477  1300
```

Y extendiendo su uso con IN o LIKE (es último sólo con %), pudiendo aplicársele el NOT:

```
sqldf('SELECT * FROM BOD WHERE Time IN (1,7)')

##   Time demand
## 1     1     8.3
## 2     7    19.8

sqldf('SELECT * FROM BOD WHERE Time NOT IN (1,7)')

##   Time demand
## 1     2    10.3
## 2     3    19.0
## 3     4    16.0
## 4     5    15.6

sqldf('SELECT * FROM chickwts WHERE feed LIKE "%bean" LIMIT 5')

##   weight      feed
## 1    179 horsebean
## 2    160 horsebean
## 3    136 horsebean
## 4    227 horsebean
## 5    217 horsebean

sqldf('SELECT * FROM chickwts WHERE feed NOT LIKE "%bean" LIMIT 5')

##   weight      feed
## 1    309 linseed
## 2    229 linseed
## 3    181 linseed
```

```
## 4    141 linseed
## 5    260 linseed
```

### 3.11 Ejemplo Scopus data

Ver ejemplo *citan.zip* y Apéndice ??.

“If your data fits in memory there is no advantage to putting it in a database: it will only be slower and more frustrating”

— Hadley Wickham – <https://dbplyr.tidyverse.org/articles/dbplyr.html>

### 3.12 Ejercicios SQL con RSQLite

#### 3.12.1 Setup de RSQLite

Vamos a utilizar RSQLite desde Kaggle. Pero si lo queréis instalar en local La información para su instalación está en el siguiente enlace.

```
library(DBI)

# Create an ephemeral in-memory RSQLite database
con <- dbConnect(RSQLite::SQLite(), ":memory:")
dbListTables(con)

## character(0)

dbWriteTable(con, "mtcars", mtcars)
dbListTables(con)

## [1] "mtcars"

dbListFields(con, "mtcars")

##  [1] "mpg"   "cyl"   "disp"  "hp"    "drat"  "wt"    "qsec" "vs"    "am"    "gear"
## [11] "carb"

dbReadTable(con, "mtcars")

##      mpg cyl  disp  hp drat   wt  qsec vs am gear carb
## 1 21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## 2 21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## 3 22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## 4 21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## 5 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## 6 18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## 7 14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## 8 24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
```

```

## 9 22.8   4 140.8  95 3.92 3.150 22.90  1  0   4   2
## 10 19.2   6 167.6 123 3.92 3.440 18.30  1  0   4   4
## 11 17.8   6 167.6 123 3.92 3.440 18.90  1  0   4   4
## 12 16.4   8 275.8 180 3.07 4.070 17.40  0  0   3   3
## 13 17.3   8 275.8 180 3.07 3.730 17.60  0  0   3   3
## 14 15.2   8 275.8 180 3.07 3.780 18.00  0  0   3   3
## 15 10.4   8 472.0 205 2.93 5.250 17.98  0  0   3   4
## 16 10.4   8 460.0 215 3.00 5.424 17.82  0  0   3   4
## 17 14.7   8 440.0 230 3.23 5.345 17.42  0  0   3   4
## 18 32.4   4  78.7  66 4.08 2.200 19.47  1  1   4   1
## 19 30.4   4  75.7  52 4.93 1.615 18.52  1  1   4   2
## 20 33.9   4  71.1  65 4.22 1.835 19.90  1  1   4   1
## 21 21.5   4 120.1  97 3.70 2.465 20.01  1  0   3   1
## 22 15.5   8 318.0 150 2.76 3.520 16.87  0  0   3   2
## 23 15.2   8 304.0 150 3.15 3.435 17.30  0  0   3   2
## 24 13.3   8 350.0 245 3.73 3.840 15.41  0  0   3   4
## 25 19.2   8 400.0 175 3.08 3.845 17.05  0  0   3   2
## 26 27.3   4  79.0  66 4.08 1.935 18.90  1  1   4   1
## 27 26.0   4 120.3  91 4.43 2.140 16.70  0  1   5   2
## 28 30.4   4  95.1 113 3.77 1.513 16.90  1  1   5   2
## 29 15.8   8 351.0 264 4.22 3.170 14.50  0  1   5   4
## 30 19.7   6 145.0 175 3.62 2.770 15.50  0  1   5   6
## 31 15.0   8 301.0 335 3.54 3.570 14.60  0  1   5   8
## 32 21.4   4 121.0 109 4.11 2.780 18.60  1  1   4   2

# You can fetch all results:
res <- dbSendQuery(con, "SELECT * FROM mtcars WHERE cyl = 4")
dbFetch(res)

##      mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## 1 22.8   4 108.0 93 3.85 2.320 18.61  1  1   4   1
## 2 24.4   4 146.7 62 3.69 3.190 20.00  1  0   4   2
## 3 22.8   4 140.8 95 3.92 3.150 22.90  1  0   4   2
## 4 32.4   4  78.7 66 4.08 2.200 19.47  1  1   4   1
## 5 30.4   4  75.7 52 4.93 1.615 18.52  1  1   4   2
## 6 33.9   4  71.1 65 4.22 1.835 19.90  1  1   4   1
## 7 21.5   4 120.1 97 3.70 2.465 20.01  1  0   3   1
## 8 27.3   4  79.0 66 4.08 1.935 18.90  1  1   4   1
## 9 26.0   4 120.3 91 4.43 2.140 16.70  0  1   5   2
## 10 30.4   4  95.1 113 3.77 1.513 16.90  1  1   5   2
## 11 21.4   4 121.0 109 4.11 2.780 18.60  1  1   4   2

dbClearResult(res)

# Or a chunk at a time
res <- dbSendQuery(con, "SELECT * FROM mtcars WHERE cyl = 4")
while(!dbHasCompleted(res)){

```

```

chunk <- dbFetch(res, n = 5)
print(nrow(chunk))
}

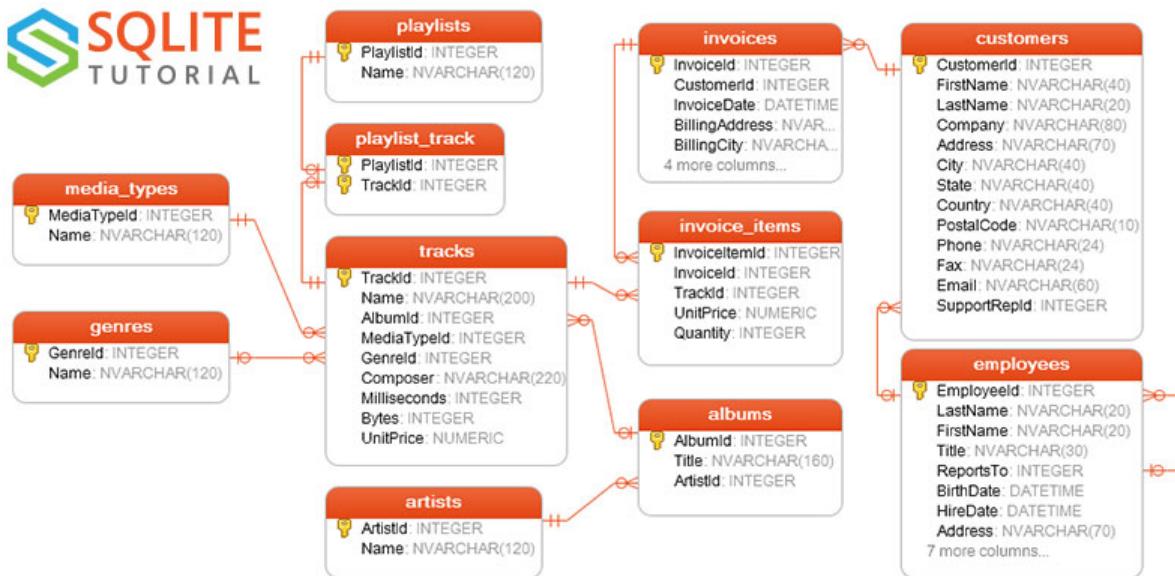
## [1] 5
## [1] 5
## [1] 1
# Clear the result
dbClearResult(res)

# Disconnect from the database
dbDisconnect(con)

```

### 3.13 Práctica 1: SQL

Vamos a utilizar la base de datos Chinook del tutorial de SQLite



Los ejercicios pedidos en Kaggle kaggle.com/gltaboadal/sqlite-tutorial-in-r se entregarán preferentemente antes del **14/10** compartiendo un notebook con las soluciones (¡notebook privado!) con el usuario **gltaboadal**. Antes me tenéis que enviar un email comunicando qué usuario tenéis cada uno. En caso de incidencia me podéis mandar un notebook descargado (.ipynb), o el mecanismo que hayamos acordado previamente.

## Capítulo 4

# Manipulación de datos con tidyverse

Este la primera parte de este capítulo (Sección 4.1), se pretende realizar una breve introducción al *ecosistema Tidyverse*, una colección de paquetes diseñados de forma uniforme (con la misma filosofía y estilo) para trabajar conjuntamente.

La referencia recomendada para usuarios de R que deseen iniciarse en el uso de estos paquetes es:

Wickham, H., y Grolemund, G. (2016). *R for data science: import, tidy, transform, visualize, and model data*, online-castellano, O'Reilly.

En las consecutivas secciones se presentan las alternativas *tidyverse* a la lectura, manipulacióin y escritura de datos tratadas en Capítulo 2.

Más adelante, en la Sección 4.2 se realiza una breve introducción al paquete `dplyr` y en la Sección 4.3 se comentan algunas de las utilidades del paquete `tidyr` que pueden resultar de interés<sup>1</sup>. Finalmente, las secciones 4.4 y 4.5 se muestra las utilizadas para tratar tablas y bases de datos respectivamente.

### 4.1 Introducción al ecosistema tidyverse

El paquete `tidyverse` está diseñado para facilitar la instalación y carga de los paquetes principales de la colección tidyverse con un solo comando. Al instalar este paquete se instalan paquetes que forman el denominado núcleo de tidyverse (se cargan con `library(tidyverse)`):

- `ggplot2`: visualización de datos.

---

<sup>1</sup>Otra alternativa (más rápida) es `data.table` pero en versiones recientes ya se puede emplear desde `dplyr`, como se comenta más adelante.

- **dplyr**: manipulación de datos.
- **tidyverse**: reorganización (limpieza) de datos.
- **readr**: importación de datos.
- **tibble**: tablas de datos (extensión de `data.frame`).
- **purrr**: programación funcional.
- **stringr**: manipulación de cadenas de texto.
- **forcats**: manipulación de factores.
- **lubridate**: manipulación de fechas y horas.

y un conjunto de paquetes recomendados:

- **feather**: almacenamiento eficiente de data frames. - **haven**: lectura y escritura de datos de SPSS, Stata y SAS en R - **modelr**: crear pipelines<sup>2</sup> elegantes al modelar datos en R (obsoleto). **broom...**): resúmenes estadísticos en formato Tidy

Otros paquetes de interés son:

- **readxl**: lectura de archivos Excel.
- **readxl**: exportación a Excel.
- **hms**: manipulación de medidas de tiempo.
- **httr**: interactuar con web APIs.
- **jsonlite**: Lectura y escritura de archivos JSON (*JavaScript Object Notation*).
- **rvest**: extraacción de datos (estructurados) de páginas web *web scraping*.
- **xml2**: lectura y escritura de archivos XML.
- **vroom**: lectura eficiente de archivos delimitados

```
library(tidyverse)
```

También hay paquetes “asociados”:

- **rlang**: herramientas para programación funcional.
- **tidyselect**: Sintaxis seleccionar variables (columnas).
- **tune**: hiperparámetros en modelos estadísticos
- **tidymodels**: meta-paquete para todo el proceso de modelado.

Muchos otros paquetes están adaptando este estilo, por ejemplo, el meta paquete **tidyverts**) para el análisis de series temporales (*time series*, TS), que incluye, por ejemplo:

- **tsibble** (infra)estructuras de datos.
- **fable** predicción (*forecasting*).
- **feasts** extracción de características (predictores).

El paquete **fpp3** asociado al libro **Forecasting: Principles and Practice** también sigue una filosofía *tidy*.

---

<sup>2</sup>serie de pasos conectados (tuberías) que procesan datos y los transforman en un formato deseado para su análisis o modelado

Otro ejemplo, en este caso, para el tratamiento de datos espaciales, sería el paquete **sf**, para más detalles ver Sección 2.2 Introducción al paquete sf del libro **Estadística Espacial con R**

Resumiendo, está muy de moda y puede terminar convirtiéndose en un dialecto del lenguaje R, todo lo que resulte de utilidad es bien venido... Aunque se recomienda evitar estos paquetes en las primeras etapas de formación en R.

El estilo de programación tiene como origen la gramática de **ggplot2** para crear gráficos de forma declarativa, basado a su vez en:

Wilkinson, L. (2005). *The Grammar of Graphics*. Springer.

Este paquete se ha convertido en un sustituto de los gráficos **lattice**, de utilidad en algunos informes finales, aplicaciones para empresas, o para gráficos muy especializados. Aunque, en condiciones normales, suele ser más rápido generar o programar gráficos estándar de R.

Para iniciarse en este paquete lo recomendado es consultar los capítulos Data Visualización y Graphics for communication de R for Data Science. También puede resultar de interés la chuleta). La referencia que cubre con mayor profundidad este paquete es:

Wickham, H. (2016). *ggplot2: Elegant graphics for Data Analysis* (3<sup>a</sup> edición, en desarrollo junto a Navarro, D. y Pedersen, T.L.). Springer.

Otra alternativa sería:

Chang, W. (2023). *The R Graphics Cookbook*. O'Reilly.

En **ggplot2** se emplea el operador **+** para añadir componentes de los gráficos (ver , en *Tidyverse* se emplea un operador de redirección para añadir operaciones.

### 4.1.1 Operador *pipe* (redirección)

El operador **%>%** (paquete **magrittr**) permite canalizar la salida de una función a la entrada de otra. Se utiliza para mejorar la legibilidad y la claridad del código al encadenar múltiples operaciones en una secuencia fluida. Por ejemplo, **segundo(primeros(datos))** se traduce en **datos %>% primero %>% segundo**, lo que facilita la lectura de operaciones al escribir las funciones de izquierda a derecha.

Desde la versión 4.1 de R está disponible un operador interno **|>**. Por ejemplo, para el conjunto de datos **empleados.RData** que contiene datos de empleados de un banco. Supongamos, por ejemplo, que estamos interesados en estudiar si hay discriminación por cuestión de sexo o raza.

```
load("data/empleados.RData")
# NOTA: Cuidado con la codificación latin1 (no declarada)
# al abrir archivos creados en versiones anteriores de R < 4.2:
# load("data/empleados.latin1.RData")
```

```
# Listamos las etiquetas
#knitr::kable(attr(empleados, "variable.labels"),
#               col.names = "Etiqueta")

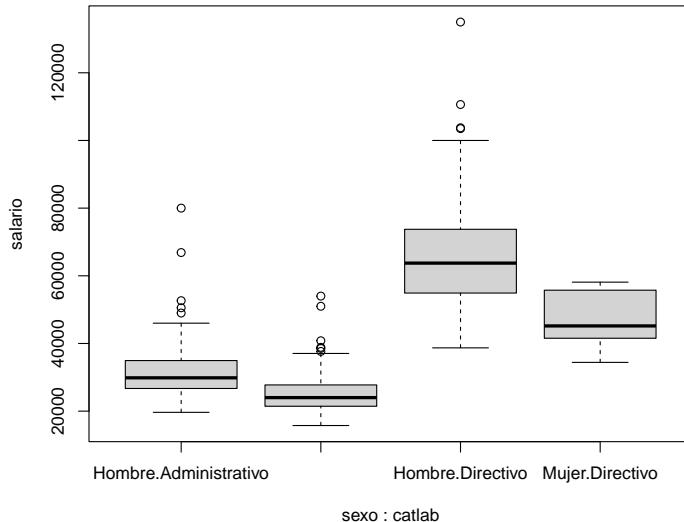
# Eliminamos las etiquetas para que no molesten...
# attr(empleados, "variable.labels") <- NULL

#empleados />
# subset(catlab == "Directivo", catlab:sexoraza) />
# summary()
```

Para que una función sea compatible con este tipo de operadores el primer parámetro debería ser siempre los datos. Sin embargo, el operador `%>%` permite redirigir el resultado de la operación anterior a un parámetro distinto mediante un `..`. Por ejemplo:

```
# ?"/>
# empleados /> subset(catlab != "Seguridad") /> droplevels />
#       boxplot(salario ~ sexo*catlab, data = .) # ERROR

library(magrittr)
empleados %>%
  subset(catlab != "Seguridad") %>%
  droplevels() %>%
  boxplot(salario ~ sexo*catlab, data = .)
```



### 4.1.2 Lectura y escritura de archivos de texto

En esta sección la alternativa *tidyverse*, a la tradicional, vista en las secciones 2.1.3 y 2.1.6 del Capítulo 2.

Para leer archivos de texto en distintos formatos se puede emplear el paquete `readr`, disponible en la colección de paquetes `tidyverse`. Para más información, se recomienda consultar el Capítulo 11 del libro *R for Data Science* (Wickham et al., 2023a) o la versión en español “*R Para Ciencia de Datos*”.

```
library(readr)
# ?readr
datos <- read_csv("./data/coches.csv")
class(datos)

## [1] "spec_tbl_df" "tbl_df"       "tbl"          "data.frame"
```

También se puede importación desde Excel fácilmente:

```
library(readxl)
datos<-read_excel("./data/coches.xlsx")
class(datos)

## [1] "tbl_df"      "tbl"        "data.frame"
excel_sheets("./data/coches.xlsx") # listado de hojas

## [1] "coches"
```

Otra alternativa, sería emplear el paquete `data.table`. La función `fread()` puede considerarse como alternativa a `read_csv()` cuando el proceso de lectura resulta lento, especialmente con datos numéricos pesados. Esta función intenta *adivinar* automáticamente algunos argumentos sin tener que especificarse como, por ejemplo, el delimitador, las filas omitidas y la cabecera. Sin embargo, si requiere especificar el separador del decimal, como a continuación:

```
library(data.table)
# ?fread
datos <- fread(file = "./data/coches.csv", dec = ",")
class(datos)

## [1] "data.table" "data.frame"
```

Para más información, se recomienda ver la viñeta *Introduction to data.table*.

### 4.1.3 Escritura

Con el ecosistema *tidyverse*, también con el paquete `readr` se puede utilizar la función `write_csv2()`:

```
write_csv2(datos, file = "datos.csv")
```

y como opción más rápida, se podría usar `fwrite()` del paquete `data.table`:

```
# datos2 <- data.table(datos)
fwrite(datos2, file = "datos2.csv")
```

Working draft...

En este capítulo se realiza una breve introducción al paquete `dplyr`. Para mas información, ver por ejemplo la ‘vignette’ del paquete `Introduction to dplyr`, o el Capítulo 5 Data transformation del libro R for Data Science<sup>3</sup>.

## 4.2 Manipulación de datos con `dplyr` y `tidyverse`

En esta sección se realiza una breve introducción al paquete `dplyr` y se comentan algunas de las utilidades del paquete `tidyverse` que pueden resultar de interés<sup>4</sup>.

La referencia recomendada para iniciarse en esta herramienta es el Capítulo 5 Data transformation de R for Data Science. También puede resultar de utilidad la viñeta del paquete `Introduction to dplyr` o la chuleta (menú de RStudio *Help* > *Cheat Sheets* > *Data Transformation with dplyr*).

### 4.2.1 El paquete `dplyr`

```
library(dplyr)
```

La principal ventaja de `dplyr` es que permite trabajar (de la misma forma) con datos en distintos formatos:

- `data.frame`, `tibble`.
- `data.table`: extensión (paquete *backend*) `dtplyr`.
- conjuntos de datos más grandes que la memoria disponible: extensiones `duckdb` y `arrow` (incluyendo almacenamiento en la nube, e.g. AWS).
- bases de datos relacionales (lenguaje SQL, locales o remotas); extensión `dbplyr`.
- grandes volúmenes de datos (incluso almacenados en múltiples servidores; ecosistema Hadoop/Spark): extensión `sparklyr` (ver menú de RStudio *Help* > *Cheat Sheets* > *Interfacing Spark with sparklyr*).

El paquete `dplyr` permite sustituir operaciones con funciones base de R (como `subset`, `split`, `apply`, `sapply`, `lapply`, `tapply`, `aggregate`...) por una “gramática” más sencilla para la manipulación de datos. En lugar de operar sobre

<sup>3</sup>Una alternativa (más rápida) es emplear `data.table`.

<sup>4</sup>Otra alternativa (más rápida) es `data.table` pero en versiones recientes ya se puede emplear desde `dplyr`, como se comenta más adelante.

vectores como la mayoría de las funciones base, opera sobre conjuntos de datos (de forma que es compatible con el operador `%>%`). Los principales “verbos” (funciones) son:

- `select()`: seleccionar variables (ver también `rename`, `relocate`, `pull`).
- `mutate()`: crear variables (ver también `transmute()`).
- `filter()`: seleccionar casos/filas (ver también `slice()`).
- `arrange()`: ordenar casos/filas.
- `summarise()`: resumir valores.
- `group_by()`: permite operaciones por grupo empleando el concepto “dividir-aplicar-combinar” (`ungroup()` elimina el agrupamiento).

NOTA: Para entender el funcionamiento de ciertas funciones (como `rowwise()`) y las posibilidades en el manejo de datos, hay que tener en cuenta que un `data.frame` no es más que una lista cuyas componentes (variables) tienen la misma longitud. Realmente las componentes también pueden ser listas de la misma longitud y, por tanto, podemos almacenar casi cualquier estructura de datos en un `data.frame`.

En la primera parte de este capítulo consideraremos solo `data.frame` por comodidad. Emplearemos como ejemplo los datos de empleados de banca almacenados en el fichero `empleados.RData` (y supondremos que estamos interesados en estudiar si hay discriminación por cuestión de sexo o raza).

```
load("data/empleados.RData")
attr(empleados, "variable.labels") <- NULL
```

En la Sección 4.5 final emplearemos una base de datos relacional como ejemplo.

### 4.2.2 Operaciones con variables (columnas)

Podemos **seleccionar variables con `select()`**:

```
emplea2 <- empleados %>% select(id, sexo, minoria, tiempemp,
                                     salini, salario)
head(emplea2)
```

```
##   id   sexo minoria tiempemp salini salario
## 1  1 Hombre     No      98  27000  57000
## 2  2 Hombre     No      98  18750  40200
## 3  3 Mujer      No      98  12000  21450
## 4  4 Mujer      No      98  13200  21900
## 5  5 Hombre     No      98  21000  45000
## 6  6 Hombre     No      98  13500  32100
```

Se puede cambiar el nombre (ver también `rename()`):

```
empleados %>% select(sexo, noblanca = minoria, salario) %>% head()

##      sexo noblanca salario
## 1 Hombre      No    57000
## 2 Hombre      No    40200
## 3 Mujer      No    21450
## 4 Mujer      No    21900
## 5 Hombre      No    45000
## 6 Hombre      No    32100
```

Se pueden emplear los nombres de variables como índices:

```
empleados %>% select(sexo:salario) %>% head()
```

```
##      sexo fechnac educ      catlab salario
## 1 Hombre 1952-02-03 15 Directivo 57000
## 2 Hombre 1958-05-23 16 Administrativo 40200
## 3 Mujer 1929-07-26 12 Administrativo 21450
## 4 Mujer 1947-04-15  8 Administrativo 21900
## 5 Hombre 1955-02-09 15 Administrativo 45000
## 6 Hombre 1958-08-22 15 Administrativo 32100
```

```
# empleados %>% select(-(sexo:salario)) %>% head()
empleados %>% select(!(sexo:salario)) %>% head()
```

```
##   id salini tiempemp expprev minoria      sexoraza
## 1  1  27000          98     144      No Blanca varón
## 2  2  18750          98      36      No Blanca varón
## 3  3  12000          98     381      No Blanca mujer
## 4  4  13200          98     190      No Blanca mujer
## 5  5  21000          98     138      No Blanca varón
## 6  6  13500          98      67      No Blanca varón
```

Se pueden emplear distintas herramientas (*selection helpers*) para seleccionar variables (ver paquete `tidyselect`):

- `starts_with`, `ends_with`, `contains`, `matches`, `num_range`: variables que coincidan con un patrón.
- `all_of`, `any_of`: variables de un vectores de caracteres.
- `everything`, `last_col`: todas las variables o la última variable.
- `where()`: a partir de una función (e.g. `where(is.numeric)`)

Por ejemplo:

```
empleados %>% select(starts_with("s")) %>% head()
```

```
##      sexo salario salini      sexoraza
## 1 Hombre    57000  27000 Blanca varón
```

```
## 2 Hombre 40200 18750 Blanca varón
## 3 Mujer 21450 12000 Blanca mujer
## 4 Mujer 21900 13200 Blanca mujer
## 5 Hombre 45000 21000 Blanca varón
## 6 Hombre 32100 13500 Blanca varón
```

Podemos crear variables con `mutate()`:

```
emplea2 %>%
  mutate(incsal = salario - salini, tsal = incsal/tiempemp) %>%
  head()
```

```
##   id  sexo minoria tiempemp salini salario incsal      tsal
## 1  1  Hombre     No      98 27000  57000 30000 306.12245
## 2  2  Hombre     No      98 18750  40200 21450 218.87755
## 3  3  Mujer      No      98 12000  21450  9450  96.42857
## 4  4  Mujer      No      98 13200  21900  8700  88.77551
## 5  5  Hombre     No      98 21000  45000 24000 244.89796
## 6  6  Hombre     No      98 13500  32100 18600 189.79592
```

### 4.2.3 Operaciones con casos (filas)

Podemos seleccionar casos con `filter()`:

```
emplea2 %>% filter(sexo == "Mujer", minoria == "Sí") %>% head()
```

```
## [1] id      sexo    minoria  tiempemp salini    salario
## <0 rows> (or 0-length row.names)
```

Podemos reordenar casos con `arrange()`:

```
emplea2 %>% arrange(salario) %>% head()
```

```
##   id  sexo minoria tiempemp salini salario
## 1 378 Mujer     No      70 10200  15750
## 2 338 Mujer     No      74 10200  15900
## 3  90 Mujer     No      92  9750  16200
## 4 224 Mujer     No      82 10200  16200
## 5 411 Mujer     No      68 10200  16200
## 6 448 Mujer S\xed     66 10200  16350
```

```
emplea2 %>% arrange(desc(salini), salario) %>% head()
```

```
##   id  sexo minoria tiempemp salini salario
## 1 29 Hombre     No      96 79980 135000
## 2 343 Hombre    No      73 60000 103500
## 3 205 Hombre    No      83 52500  66750
## 4 160 Hombre    No      86 47490  66000
## 5 431 Hombre    No      66 45000  86250
## 6 32 Hombre     No      96 45000 110625
```

Podemos resumir valores con `summarise()`:

```
empleados %>% summarise(sal.med = mean(salario), n = n())
##      sal.med     n
## 1 34419.57 474
```

Para realizar operaciones con múltiples variables podemos emplear `across()` (admite selección de variables `tidyselect`):

```
empleados %>% summarise(across(where(is.numeric), mean), n = n())
##      id     educ salario salini tiempemp expprev    n
## 1 237.5 13.49156 34419.57 17016.09 81.1097 95.86076 474
# empleados %>% summarise(across(where(is.numeric) & !id, mean), n = n())
```

NOTA: Esta función sustituye a las “variantes de ámbito” `_at()`, `_if()` y `_all()` de versiones anteriores de dplyr (como `summarise_at()`, `summarise_if()`, `summarise_all()`, `mutate_at()`, `mutate_if()`...) y también el uso de `vars()`. En el caso de `filter()` se puede emplear `if_any()` e `if_all()`.

Podemos agrupar casos con `group_by()`:

```
empleados %>% group_by(sexo, minoria) %>%
  summarise(sal.med = mean(salario), n = n()) %>%
  ungroup()

## # A tibble: 4 x 4
##   sexo   minoria sal.med     n
##   <fct>  <fct>    <dbl> <int>
## 1 Hombre "No"    44475.   194
## 2 Hombre "S\xed"  32246.   64
## 3 Mujer  "No"    26707.   176
## 4 Mujer  "S\xed"  23062.   40

empleados %>% group_by(sexo, minoria) %>%
  summarise(sal.med = mean(salario), n = n(), .groups = "drop")

## # A tibble: 4 x 4
##   sexo   minoria sal.med     n
##   <fct>  <fct>    <dbl> <int>
## 1 Hombre "No"    44475.   194
## 2 Hombre "S\xed"  32246.   64
## 3 Mujer  "No"    26707.   176
## 4 Mujer  "S\xida" 23062.   40

# dplyr >= 1.1.0 # packageVersion("dplyr")
# empleados %>% summarise(sal.med = mean(salario), n = n(),
#                           .by = c(sexo, minoria))
```

Por defecto la agrupación se mantiene para el resto de operaciones, habría que emplear `ungroup()` (o el argumento `.groups = "drop"`) para eliminarla (se puede emplear `group_vars()` o `str()` para ver la agrupación). Desde dplyr 1.1.0 (2023-01-29) está disponible un parámetro `.by/by` en `mutate()`, `summarise()`, `filter()` y `slice()` como alternativa a agrupar y desagrupar posteriormente. Para más detalles ver Per-operation grouping with `.by/by`.

#### 4.2.4 Datos faltantes

Continuamos con el ejemplo de la Sección @ref{missing}. `tidyverse` dispone de muchas herramientas para el tratamiento de los datos faltantes.

```
data("airquality")
datos <- airquality
library(visdat)
vis_dat(airquality)
# vis_miss(airquality)
```

Visualización (amigable) de la estructura de datos:

```
library(naniar)
bind_shadow(airquality)

## # A tibble: 153 x 12
##   Ozone Solar.R Wind Temp Month Day Ozone_NA Solar.R_NA Wind_NA Temp_NA
##   <int>    <int> <dbl> <int> <int> <fct>    <fct>    <fct>    <fct>
## 1     41      190   7.4    67     5   1 !NA     !NA     !NA     !NA
## 2     36      118    8     72     5   2 !NA     !NA     !NA     !NA
## 3     12      149  12.6    74     5   3 !NA     !NA     !NA     !NA
## 4     18      313  11.5    62     5   4 !NA     !NA     !NA     !NA
## 5     NA       NA  14.3    56     5   5  NA      NA      !NA     !NA
## 6     28       NA  14.9    66     5   6 !NA     NA      !NA     !NA
## 7     23      299   8.6    65     5   7 !NA     !NA     !NA     !NA
## 8     19       99  13.8    59     5   8 !NA     !NA     !NA     !NA
## 9      8       19  20.1    61     5   9 !NA     !NA     !NA     !NA
## 10    NA      194   8.6    69     5  10  NA     !NA     !NA     !NA
## # i 143 more rows
## # i 2 more variables: Month_NA <fct>, Day_NA <fct>
# nabular(airquality)
```

Distribución por variables de los datos faltantes:

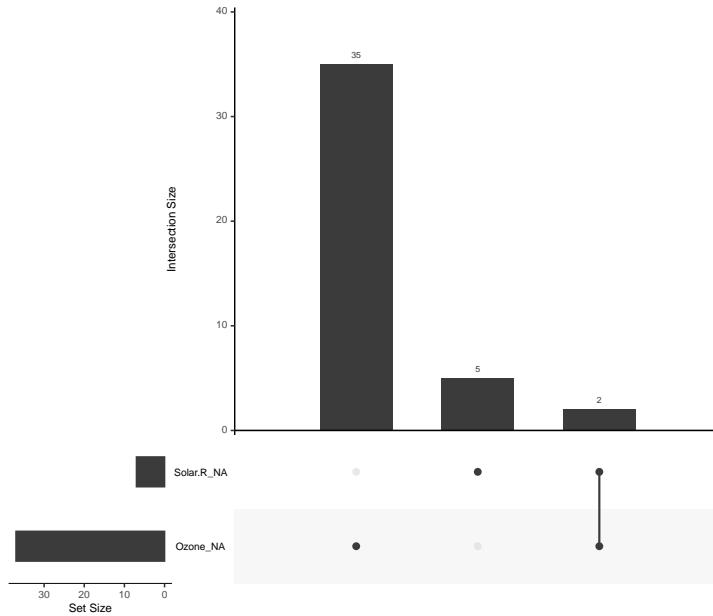
```
miss_var_table(airquality)

## # A tibble: 3 x 3
##   n_miss_in_var n_vars pct_vars
##   <int>    <int>    <dbl>
## 1          0        4     66.7
```

```
## 2      7      1    16.7
## 3     37      1    16.7
```

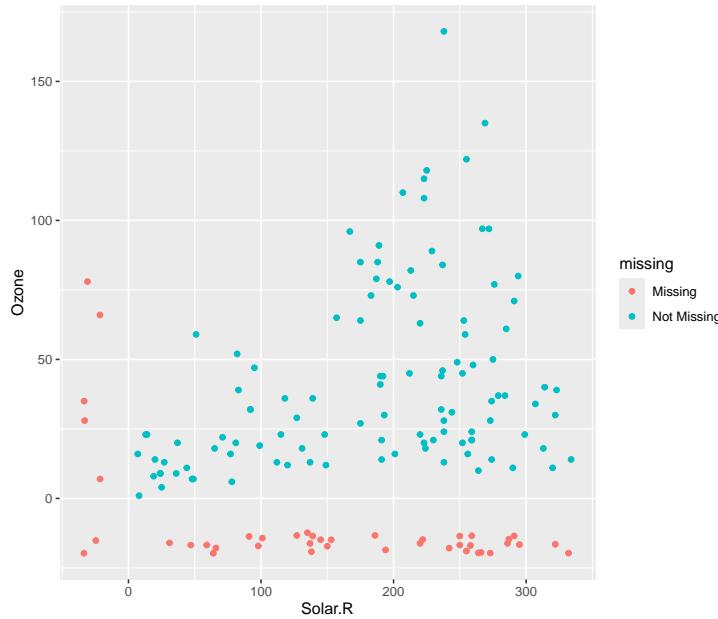
```
prop_miss_case(airquality)
```

```
## [1] 0.2745098
gg_miss_upset(airquality)
```



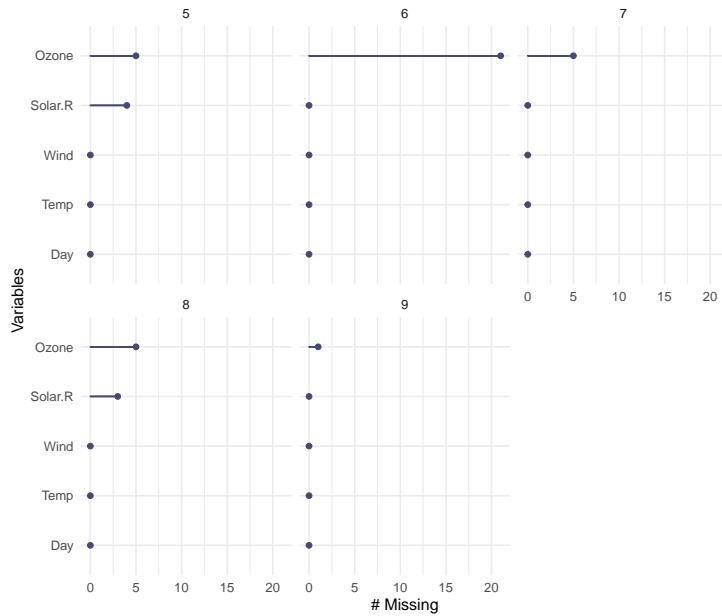
Distribución conjunta de los valores faltantes para la radiación solar y ozono:

```
library(naniar)
library(ggplot2)
ggplot(airquality,
       aes(x = Solar.R,
           y = Ozone)) +
  geom_miss_point()
```



Distribución mensual de los valores faltantes:

```
# gg_miss_var(airquality)
gg_miss_var(airquality, facet = Month)
```



## 4.3 Herramientas `tidyverse`

Algunas funciones del paquete `tidyverse` que pueden resultar de especial interés son:

- `pivot_wider()`: permite transformar valores de grupos de casos a nuevas variables.
- `pivot_longer()`: realiza la transformación inversa, colapsar varias columnas en una.

Ver la viñeta Pivoting para más detalles.

- `separate()`: permite separar una columna de texto en varias (ver también `extract()`).

Ver `mortalidad.R` en ejemplos.

## 4.4 Operaciones con tablas de datos

Se emplean funciones `xxx_join()` (ver la documentación del paquete Join two tbls together, o la vignette Two-table verbs):

- `inner_join()`: devuelve las filas de `x` que tienen valores coincidentes en `y`, y todas las columnas de `x` e `y`. Si hay varias coincidencias entre `x` e `y`, se devuelven todas las combinaciones.
  - `left_join()`: devuelve todas las filas de `x` y todas las columnas de `x` e `y`. Las filas de `x` sin correspondencia en `y` contendrán `NA` en las nuevas columnas. Si hay varias coincidencias entre `x` e `y`, se devuelven todas las combinaciones (duplicando las filas).
- `right_join()` hace lo contrario, devuelve todas las filas de `y`.
- `full_join()` devuelve todas las filas de `x` e `y` (duplicando o asignando `NA` si es necesario).
- `semi_join()`: devuelve las filas de `x` que tienen valores coincidentes en `y`, manteniendo sólo las columnas de `x` (al contrario que `inner_join()` no duplica filas).
- `anti_join()` hace lo contrario, devuelve las filas sin correspondencia.

El parámetro `by` determina las variables clave para las correspondencias. Si no se establece se considerarán todas las que tengan el mismo nombre en ambas tablas. Se puede establecer a un vector de nombres coincidentes y en caso de que los nombres sean distintos a un vector con nombres de la forma `c("clave_x" = "clave_y")`.

Adicionalmente, si las tablas `x` e `y` tienen las mismas variables, se pueden combinar las observaciones con operaciones de conjuntos:

- `intersect(x, y)`: observaciones en `x` y en `y`.

- `union(x, y)`: observaciones en `x` o `y` no duplicadas.
- `setdiff(x, y)`: observaciones en `x` pero no en `y`.

## 4.5 Bases de datos con dplyr

Para poder usar tablas en bases de datos relacionales con `dplyr` hay que emplear el paquete `dbplyr` (convierte automáticamente el código de `dplyr` en consultas SQL).

Algunos enlaces:

- Best Practices in Working with Databases
- Introduction to dbplyr
- Data Carpentry: SQL databases and R,
- R and Data – When Should we Use Relational Databases?

### 4.5.1 Ejemplos

Como ejemplo emplearemos la base de datos de SQLite Sample Database Tutorial, almacenada en el archivo `chinook.db`.

```
# install.packages('dbplyr')
library(dplyr)
library(dbplyr)
```

En primer lugar hay que conectar la base de datos:

```
chinook <- DBI::dbConnect(RSQLite::SQLite(), "data/chinook.db")
```

Podemos listar las tablas:

```
src_dbi(chinook)
```

```
## src:  sqlite 3.47.1 [/home/diego/UDC/Teaching/MTE/TGD/tgdbbook-guillermo/data/chinook.db]
## tbls: albums, artists, customers, employees, genres, invoice_items, invoices,
##       media_types, playlist_track, playlists, sqlite_sequence, sqlite_stat1, tracks
```

Para enlazar una tabla:

```
invoices <- tbl(chinook, "invoices")
invoices

## # Source:  table<`invoices`> [?? x 9]
## # Database: sqlite 3.47.1 [/home/diego/UDC/Teaching/MTE/TGD/tgdbbook-guillermo/data/chinook.db]
##   InvoiceId CustomerId InvoiceDate      BillingAddress BillingCity BillingState
##             <int>        <int> <chr>          <chr>        <chr>        <chr>
##   1           1            2 2009-01-01 00:0~ Theodor-Heuss~ Stuttgart    <NA>
##   2           2            4 2009-01-02 00:0~ Ullevålsveien~ Oslo        <NA>
```

```

## 3      3      8 2009-01-03 00:0~ Grétrystraat ~ Brussels <NA>
## 4      4      14 2009-01-06 00:0~ 8210 111 ST NW Edmonton AB
## 5      5      23 2009-01-11 00:0~ 69 Salem Stre~ Boston MA
## 6      6      37 2009-01-19 00:0~ Berger Straße~ Frankfurt <NA>
## 7      7      38 2009-02-01 00:0~ Barbarossastr~ Berlin <NA>
## 8      8      40 2009-02-01 00:0~ 8, Rue Hanovre Paris <NA>
## 9      9      42 2009-02-02 00:0~ 9, Place Loui~ Bordeaux <NA>
## 10     10     46 2009-02-03 00:0~ 3 Chatham Str~ Dublin Dublin
## # i more rows
## # i 3 more variables: BillingCountry <chr>, BillingPostalCode <chr>,
## #   Total <dbl>

```

Ojo [?? x 9]: de momento no conoce el número de filas.

```
nrow(invoices)
```

```
## [1] NA
```

- Podemos mostrar la consulta SQL correspondiente a una operación:

```
show_query(head(invoices))
```

```

## <SQL>
## SELECT `invoices`.*
## FROM `invoices`
## LIMIT 6
# str(head(invoices))

```

Al trabajar con bases de datos, dplyr intenta ser lo más vago posible:

- No exporta datos a R a menos que se pida explícitamente (`collect()`).
- Retrasa cualquier operación lo máximo posible: agrupa todo lo que se desea hacer y luego hace una única petición a la base de datos.

```
invoices %>% head %>% collect
```

```

## # A tibble: 6 x 9
##   InvoiceId CustomerId InvoiceDate      BillingAddress BillingCity BillingState
##       <int>      <int> <chr>          <chr>          <chr>          <chr>
## 1         1          2 2009-01-01 00:00~ Theodor-Heuss~ Stuttgart <NA>
## 2         2          4 2009-01-02 00:00~ Ullevålsveien~ Oslo    <NA>
## 3         3          8 2009-01-03 00:00~ Grétrystraat ~ Brussels <NA>
## 4         4         14 2009-01-06 00:00~ 8210 111 ST NW Edmonton AB
## 5         5         23 2009-01-11 00:00~ 69 Salem Stre~ Boston MA
## 6         6         37 2009-01-19 00:00~ Berger Straße~ Frankfurt <NA>
## # i 3 more variables: BillingCountry <chr>, BillingPostalCode <chr>,
## #   Total <dbl>

```

```
invoices %>% count # número de filas
```

```
## # Source: SQL [?? x 1]
## # Database: sqlite 3.47.1 [/home/diego/UDC/Teaching/MTE/TGD/tgdbbook-guillermo/data/chinook.db]
##      n
## <int>
## 1  412
```

2. Por ejemplo, para obtener el importe mínimo, máximo y la media de las facturas:

```
res <- invoices %>% summarise(min = min(Total, na.rm = TRUE),
                                max = max(Total, na.rm = TRUE),
                                med = mean(Total, na.rm = TRUE))

# show_query(res)
res %>% collect
```

```
## # A tibble: 1 x 3
##      min     max     med
##   <dbl> <dbl> <dbl>
## 1  0.99  25.9  5.65
```

3. Para obtener el total de las facturas de cada uno de los países:

```
res <- invoices %>% group_by(BillingCountry) %>%
      summarise(n = n(), total = sum(Total, na.rm = TRUE))

# show_query(res)
res %>% collect
```

```
## # A tibble: 24 x 3
##   BillingCountry      n total
##   <chr>          <int> <dbl>
## 1 Argentina        7  37.6
## 2 Australia        7  37.6
## 3 Austria          7  42.6
## 4 Belgium          7  37.6
## 5 Brazil           35 190.
## 6 Canada           56 304.
## 7 Chile            7  46.6
## 8 Czech Republic  14  90.2
## 9 Denmark          7  37.6
## 10 Finland         7  41.6
## # i 14 more rows
```

4. Para obtener un listado con Nombre y Apellidos de cliente y el importe de cada una de sus facturas (Hint: WHERE customer.CustomerID=invoices.CustomerID):

```

customers <- tbl(chinook, "customers")
tbl_vars(customers)

## <dplyr:::vars>
## [1] "CustomerId"    "FirstName"     "LastName"      "Company"       "Address"
## [6] "City"          "State"        "Country"       "PostalCode"    "Phone"
## [11] "Fax"           "Email"         "SupportRepId"

res <- customers %>%
  inner_join(invoices, by = "CustomerId") %>%
  select(FirstName, LastName, Country, Total)
show_query(res)

## <SQL>
## SELECT `FirstName`, `LastName`, `Country`, `Total`
## FROM `customers`
## INNER JOIN `invoices`
##   ON (`customers`.`CustomerId` = `invoices`.`CustomerId`)
res %>% collect

## # A tibble: 412 x 4
##   FirstName LastName Country  Total
##   <chr>     <chr>   <chr>   <dbl>
## 1 Luís      Gonçalves Brazil   3.98
## 2 Luís      Gonçalves Brazil   3.96
## 3 Luís      Gonçalves Brazil   5.94
## 4 Luís      Gonçalves Brazil   0.99
## 5 Luís      Gonçalves Brazil   1.98
## 6 Luís      Gonçalves Brazil  13.9
## 7 Luís      Gonçalves Brazil   8.91
## 8 Leonie    Köhler    Germany  1.98
## 9 Leonie    Köhler    Germany  13.9
## 10 Leonie   Köhler    Germany  8.91
## # i 402 more rows

```

5. Para listar los 10 mejores clientes (aquellos a los que se les ha facturado más cantidad) indicando Nombre, Apellidos, País y el importe total de su facturación:

```

customers %>% inner_join(invoices, by = "CustomerId") %>% group_by(CustomerId) %>%
  summarise(FirstName, LastName, country, total = sum(Total, na.rm = TRUE)) %>%
  arrange(desc(total)) %>% head(10) %>% collect

```

6. Listar los 10 mejores clientes (aquellos a los que se les ha facturado más cantidad) indicando Nombre, Apellidos, País y el importe total de su facturación.

```
customers %>% inner_join(invoices, by = "CustomerId") %>% group_by(CustomerId) %>%
  summarise(FirstName, LastName, Country, total = sum(Total, na.rm = TRUE)) %>%
  arrange(desc(total)) %>% head(10) %>% collect

## # A tibble: 10 x 5
##   CustomerId FirstName LastName Country     total
##       <int>    <chr>   <chr>   <chr>     <dbl>
## 1          6 Helena   Holý    Czech Republic 49.6
## 2         26 Richard Cunningham USA        47.6
## 3         57 Luis      Rojas    Chile        46.6
## 4         45 Ladislav Kovács Hungary      45.6
## 5         46 Hugh     O'Reilly Ireland      45.6
## 6         24 Frank    Ralston   USA        43.6
## 7         28 Julia    Barnett   USA        43.6
## 8         37 Fynn     Zimmermann Germany      43.6
## 9          7 Astrid   Gruber    Austria     42.6
## 10        25 Victor   Stevens   USA        42.6
```

7. Listar los géneros musicales por orden decreciente de popularidad (definida la popularidad como el número de canciones de ese género), indicando el porcentaje de las canciones de ese género.

```
tracks <- tbl(chinook, "tracks")
tracks %>% inner_join(tbl(chinook, "genres"), by = "GenreId") %>% count(Name.y) %>%
  arrange(desc(n)) %>% collect %>% mutate(freq = n / sum(n))

## # A tibble: 25 x 3
##   Name.y             n   freq
##   <chr>           <int> <dbl>
## 1 Rock            1297 0.370
## 2 Latin           579  0.165
## 3 Metal            374 0.107
## 4 Alternative & Punk 332 0.0948
## 5 Jazz              130 0.0371
## 6 TV Shows          93 0.0265
## 7 Blues              81 0.0231
## 8 Classical          74 0.0211
## 9 Drama              64 0.0183
## 10 R&B/Soul          61 0.0174
## # i 15 more rows
```

8. Listar los 10 artistas con mayor número de canciones de forma descendente según el número de canciones.

```
tracks %>% inner_join(tbl(chinook, "albums"), by = "AlbumId") %>%
  inner_join(tbl(chinook, "artists"), by = "ArtistId") %>%
  count(Name.y) %>% arrange(desc(n)) %>% collect
```

```
## # A tibble: 204 x 2
##   Name.y      n
##   <chr>     <int>
## 1 Iron Maiden    213
## 2 U2            135
## 3 Led Zeppelin   114
## 4 Metallica      112
## 5 Lost           92
## 6 Deep Purple     92
## 7 Pearl Jam       67
## 8 Lenny Kravitz    57
## 9 Various Artists   56
## 10 The Office      53
## # i 194 more rows
```

Al finalizar hay que desconectar la base de datos:

```
DBI::dbDisconnect(chinook)
```

## Capítulo 5

# Introducción a Tecnologías NoSQL

Son tecnologías de almacenamiento de datos en servicios web altamente escalables.

## 5.1 Conceptos y tipos de bases de datos NoSQL (documental, columnar, clave/valor y de grafos)

NoSQL - “Not Only SQL” - es una nueva categoría de bases de datos no-relacionales y altamente distribuidas.

Las bases de datos NoSQL nacen de la necesidad de:

- Simplicidad en los diseños
- Escalado horizontal
- Mayor control en la disponibilidad

Pero cuidado, en muchos escenarios las BBDD relacionales siguen siendo la mejor opción.

### 5.1.1 Características de las bases de datos NoSQL

- Libre de esquemas – no se diseñan las tablas y relaciones por adelantado, además de permitir la migración del esquema.
- Proporcionan replicación a través de escalado horizontal.
- Este escalado horizontal se traduce en arquitectura distribuida
- Generalmente ofrecen consistencia débil

- Hacen uso de estructuras de datos sencillas, normalmente pares clave/valor a bajo nivel
- Suelen tener un sistema de consultas propio (o SQL-like)
- Siguen el modelo BASE (*Basic Availability, Soft state, Eventual consistency*) en lugar de ACID (*Atomicity, Consistency, Isolation, Durability*)

El modelo BASE consiste en:

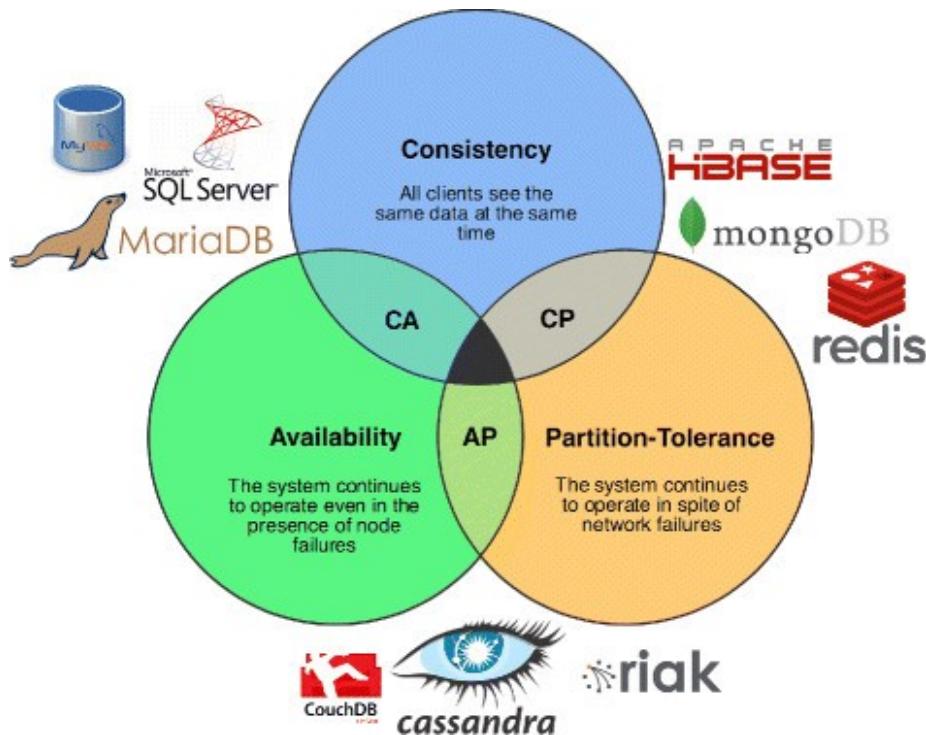
- Basic Availability – el sistema garantiza disponibilidad, en términos del teorema CAP.
- Soft state – el estado del sistema puede cambiar a lo largo del tiempo, incluso sin entrada. Esto es provocado por el modelo de consistencia eventual.
- Eventual consistency – el sistema alcanzará un estado consistente con el tiempo, siempre y cuando no reciba entrada durante ese tiempo.

#### 5.1.1.1 Teorema CAP

Es imposible para un sistema de cómputo distribuido garantizar simultáneamente:

- Consistency – Todos los nodos ven los mismos datos al mismo tiempo
- Availability – Toda petición obtiene una respuesta en caso tanto de éxito como fallo
- Partition Tolerance – El sistema seguirá funcionando ante pérdidas arbitrarias de información o fallos parciales

### 5.1. CONCEPTOS Y TIPOS DE BASES DE DATOS NOSQL (DOCUMENTAL, COLUMNAR, CLAVE/VALOR Y TABLA)



Las razones para escoger NoSQL son:

- Analítica
- Gran cantidad de escrituras, análisis en bloque
- Escalabilidad
- Tan fácil como añadir un nuevo nodo a la red, bajo coste.
- Redundancia
- Están diseñadas teniendo en cuenta la redundancia
- Rápido desarrollo
- Al ser schema-less o schema on-read son más flexibles que schema on-write
- Flexibilidad en el almacenamiento de datos
- Almacenan todo tipo de datos: texto, imágenes, BLOBs
- Gran rendimiento en consultas sobre datos que no implican relaciones jerárquicas
- Gran rendimiento sobre BBDD desnormalizadas
- Tamaño
- El tamaño del esquema de datos es demasiado grande
- Muchos datos temporales fuera de almacén principal

Razones para NO escoger NoSQL:

- \* Consistencia y Disponibilidad de los datos son críticas
- \* Relaciones entre datos son importantes + E.g. joins numerosos y/o importantes
- \* En general, cuando el modelo ACID encaja mejor

### 5.1.2 Tipos de Bases de Datos NoSQL

Columnar:

1	Things	A	foo	B	bar	C	baz
2	Things	C	bam	E	coh	People	A Emmanuel
3	Languages	A	C	B	Java	C	Ceylon

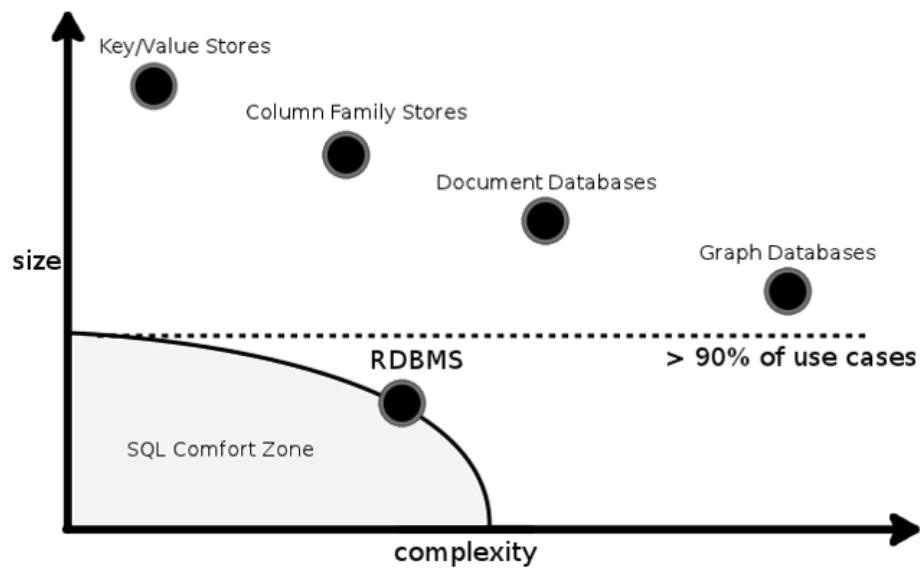
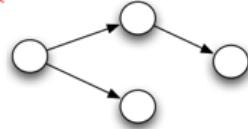
Documental:

```
{"user": {
  "id": "124",
  "name": "Emmanuel",
  "addresses": [
    {"city": "Paris", "country": "France"},
    {"city": "Atlanta", "country": "USA"}
  ]
}}
```

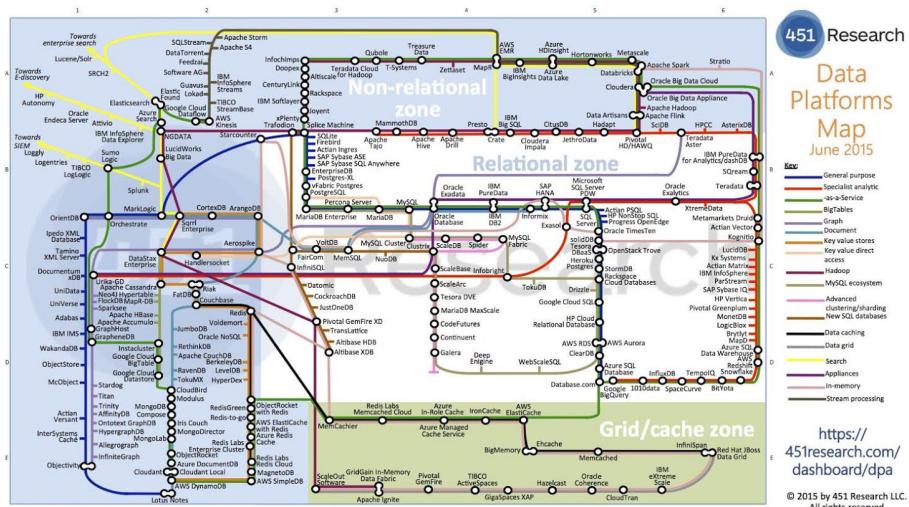
Key-value:

key	value
123	Address@23
126	"Booya"

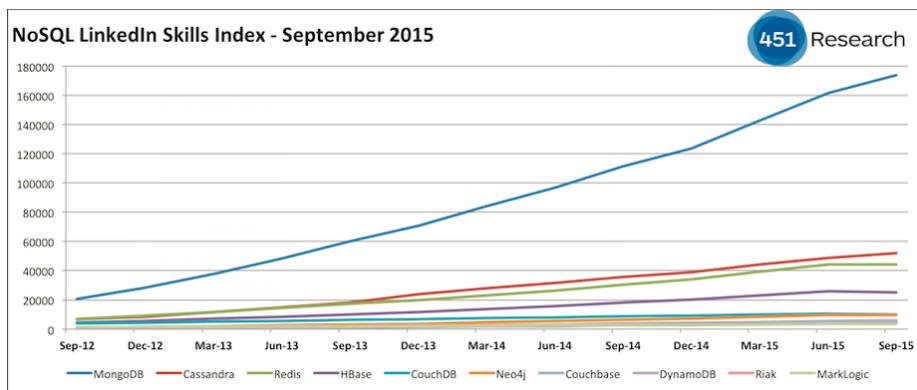
Graph:



## 5.1. CONCEPTOS Y TIPOS DE BASES DE DATOS NOSQL (DOCUMENTAL, COLUMNAR, CLAVE/VALOR Y



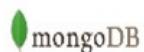
352 systems in ranking, September 2019									
Rank			DBMS	Database Model	Score				
Sep 2019	Aug 2019	Sep 2018			Sep 2019	Aug 2019	Sep 2018	Sep 2019	Sep 2018
1.	1.	1.	Oracle +	Relational, Multi-model ↗	1346.66	+7.18	+37.54		
2.	2.	2.	MySQL +	Relational, Multi-model ↗	1279.07	+25.39	+98.60		
3.	3.	3.	Microsoft SQL Server +	Relational, Multi-model ↗	1085.06	-8.12	+33.78		
4.	4.	4.	PostgreSQL +	Relational, Multi-model ↗	482.25	+0.91	+75.82		
5.	5.	5.	MongoDB +	Document	410.06	+5.50	+51.27		
6.	6.	6.	IBM Db2 +	Relational, Multi-model ↗	171.56	-1.39	-9.50		
7.	7.	7.	Elasticsearch +	Search engine, Multi-model ↗	149.27	+0.19	+6.67		
8.	8.	8.	Redis +	Key-value, Multi-model ↗	141.90	-2.18	+0.96		
9.	9.	9.	Microsoft Access	Relational	132.71	-2.63	-0.69		
10.	10.	10.	Cassandra +	Wide column	123.40	-1.81	+3.85		



Rich Queries	<ul style="list-style-type: none"> <li>Find Paul's cars</li> <li>Find everybody in London with a car built between 1970 and 1980</li> </ul>
Geospatial	<ul style="list-style-type: none"> <li>Find all of the car owners within 5km of Trafalgar Sq.</li> </ul>
Text Search	<ul style="list-style-type: none"> <li>Find all the cars described as having leather seats</li> </ul>
Aggregation	<ul style="list-style-type: none"> <li>Calculate the average value of Paul's car collection</li> </ul>
Map Reduce	<ul style="list-style-type: none"> <li>What is the ownership pattern of colors by geography over time? (is purple trending up in China?)</li> </ul>

## MongoDB

```
{
  first_name: 'Paul',
  surname: 'Miller',
  city: 'London',
  location: [45.123,47.232],
  cars: [
    { model: 'Bentley',
      year: 1973,
      value: 100000, ... },
    { model: 'Rolls Royce',
      year: 1965,
      value: 330000, ... }
  ]
}
```



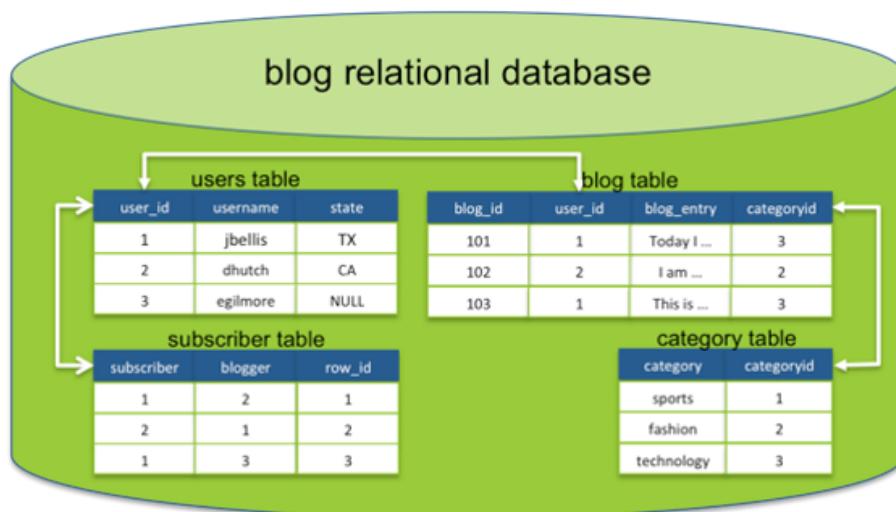
### 5.1.4 Redis: NoSQL key-value

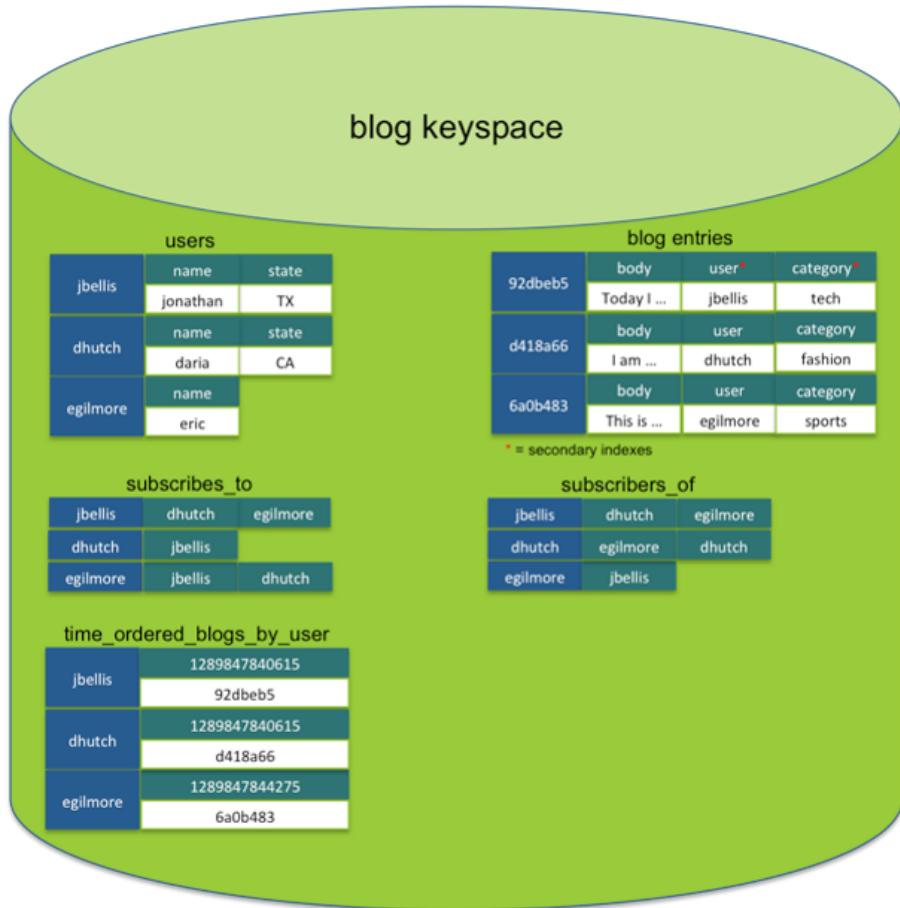
In-memory data structure store, útil para base de datos de login-password, sensor-valor, URL-respuesta, con una sintaxis muy sencilla:

- El comando SET almacena valores
- SET server:name “luna”
- Recuperamos esos valores con GET
- GET server:name
- INCR incrementa atómicamente un valor
- INCR clients
- DEL elimina claves y sus valores asociados
- DEL clients
- TTL (Time To Live) útil para cachés
- EXPIRE promocion 60

## 5.1. CONCEPTOS Y TIPOS DE BASES DE DATOS NOSQL (DOCUMENTAL, COLUMNAR, CLAVE/VALOR Y TABLA)

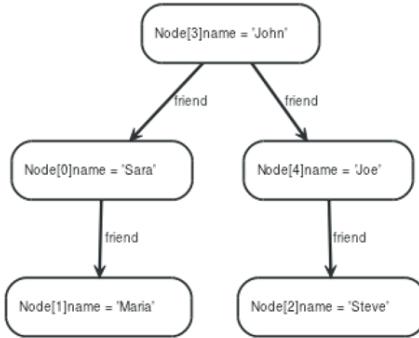
### 5.1.5 Cassandra: NoSQL columnar



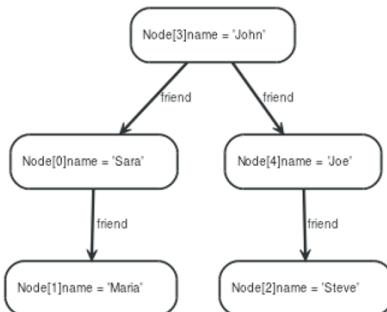


### 5.1.6 Neo4j: NoSQL grafos





`MATCH (john {name: 'John'})-[:friend]->()-[:friend]->(fof)  
RETURN john, fof`



john	fof
<code>Node[3]{name:"John"}</code>	<code>Node[1]{name:"Maria"}</code>
<code>Node[3]{name:"John"}</code>	<code>Node[2]{name:"Steve"}</code>
2 rows	

### 5.1.7 Otros: search engines

Son sistemas especializados en búsquedas, procesamiento de lenguaje natural como ElasticSearch, Solr, Splunk (logs de aplicaciones), etc...

## 5.2 Conexión de R a MongoDB

A través del paquete mongolite, aquí tenéis un Tutorial

```
install.packages("mongolite")
library(mongolite)
```

```
# Connect to a local MongoDB

my_collection = mongo(collection = "restaurants", db = "Restaurants") # create connection
my_collection$count
```

### 5.3 Ejercicios prácticos con MongoDB

Estos ejercicios se pueden hacer en un notebook Kaggle accediendo a un clúster de MongoDB en el cloud de MongoDB. Se carga la base de datos de ejemplo y se puede hacer con la colección de restaurantes (o alternativamente con otras colecciones).

1. Mostrar todos los documentos de la colección restaurants (u otra)
2. Mostrar nombre de restaurante, barrio y cocina de la colección restaurants (o los campos de otra colección)
3. Mostrar los primeros 5 restaurantes del barrio Bronx.
4. Mostrar los restaurantes con una longitud menor que -75.7541
5. Mostrar los restaurantes con una puntuación superior a 90
6. Mostrar los restaurantes de comida American o Chineese del barrio Queens.
7. Mostrar los restaurantes con un grado “A” y puntuación 9 obtenida en fecha 2014-08-11T00:00:00Z
8. Propón un JSON para descargar (de algún repositorio OpenData o disponible en un API), indícame la URL, si has de hacer algún proceso antes de importarlo en MongoDB, cómo lo importas, dame un pantallazo del análisis exploratorio de ese JSON y una query que harías contra ese JSON (la query en MongoDB, Compass o RmongoDB)

# **Capítulo 6**

## **Tecnologías para el Tratamiento de Datos Masivos**

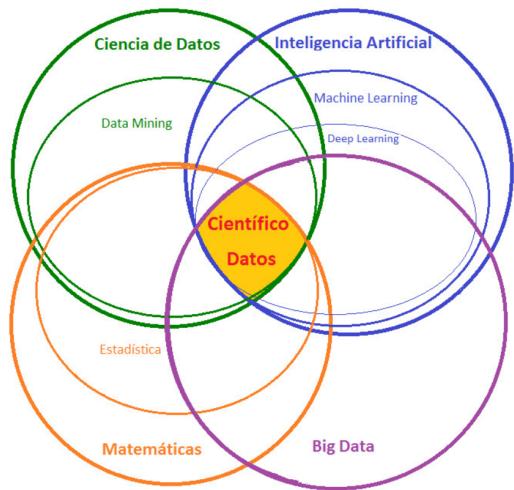
En este apartado trataremos los siguientes epígrafes:

1. Introducción al Aprendizaje Estadístico
2. Tecnologías Big Data (Hadoop, Spark, Sparklyr)
3. Ejercicios de análisis de datos masivos.

### **6.1 Introducción al Aprendizaje Estadístico**

El material para este apartado está disponible en el Capítulo 1 del libro “Aprendizaje Estadístico” de Rubén Fernández Casal.

Para seguir este capítulo es altamente recomendable tener instalado Rattle, para ello consultad el apéndice de instalación de R al final de este libro.



## 6.2 Tecnologías Big Data (Hadoop/Spark y Visualización)

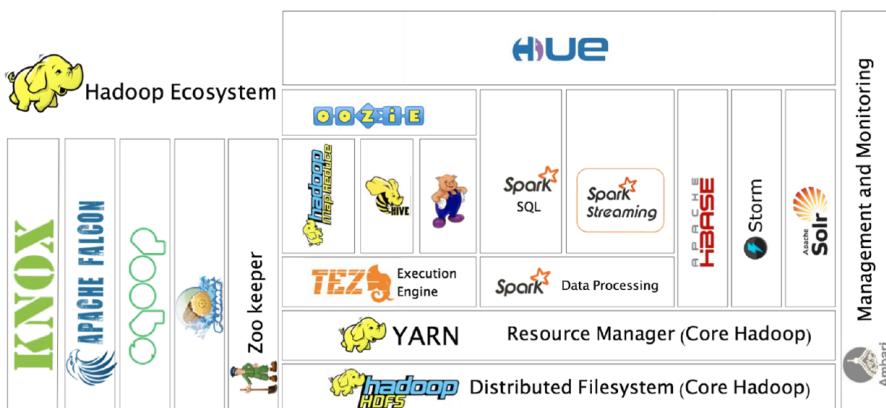
### 6.2.1 Tecnologías Hadoop, Spark, y Sparklyr

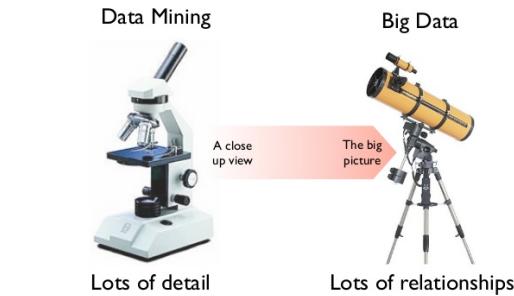
A continuación se introducen los conceptos básicos de las tecnologías Hadoop, Spark y Sparklyr:

- Hadoop: framework open-source desarrollado en Java principalmente que soporta aplicaciones distribuidas sobre miles de nodos y a escala Petabyte. Está inspirado en el diseño de las operaciones de MapReduce de Google y el Google File System (GFS). Entre sus principales componentes destaca HDFS Hadoop Distributed File System, sistema de ficheros distribuido sobre múltiples nodos y accesible a nivel de aplicación. También destaca YARN como gestor de recursos, para ejecutar aplicaciones. Destacar que la versión original, Hadoop 1, estaba basada extensivamente en Map Reduce, Hadoop 2 colocó en su core a YARN y Hadoop 3 está orientado a la provisión de Plataforma como servicio y ejecución simultánea de múltiples cargas de trabajo distribuidas sobre recursos solicitados bajo demanda.
- Hive: es un sistema de almacenamiento y explotación de datos del estilo de un data warehouse open source diseñado para ser ejecutado en entornos Hadoop. Permite agrupar, consultar y analizar datos almacenados en Hadoop File System y en Amazon S3 (almacenamiento de objetos en general) en esquema en estrella. Su lenguaje de consulta de datos, Hive Query Language o (HQL).
- Spark: framework de computación distribuida open-source para el procesamiento de datos masivos sobre Hadoop con un paralelismo implícito sobre

su estructura de datos (Resilient Distributed Dataset o RDD), permite operar en paralelo sobre una colección de datos sin saber en qué servidores están disponibles dichos datos y de una forma tolerante a fallos. Es uno de los principales frameworks de programación de entornos Hadoop al estar optimizado su procesamiento sobre memoria (en lugar de sobre archivos en disco) para obtener velocidad, tanto en sus vertientes Spark streaming y Spark SQL, como Spark Machine Learning MLlib. Dispone de interfaces en Java, Scala, Python y R, siendo las interfaces de R Rspark y Sparklyr.

- **SparkR:** es un paquete, el primero que apareció, para conectar R con Spark. Intenta ser lo más parecida a la interfaz estándar de R de manipulación de datos.
- **Sparklyr:** es una librería para conectar R con Spark posterior a SparkR. Intenta ser lo más parecida a dplyr y embeber SQL en las consultas, soportando una mayor cantidad de paquetes. Por este motivo es el proyecto más activo actualmente, sustituyendo a SparkR.

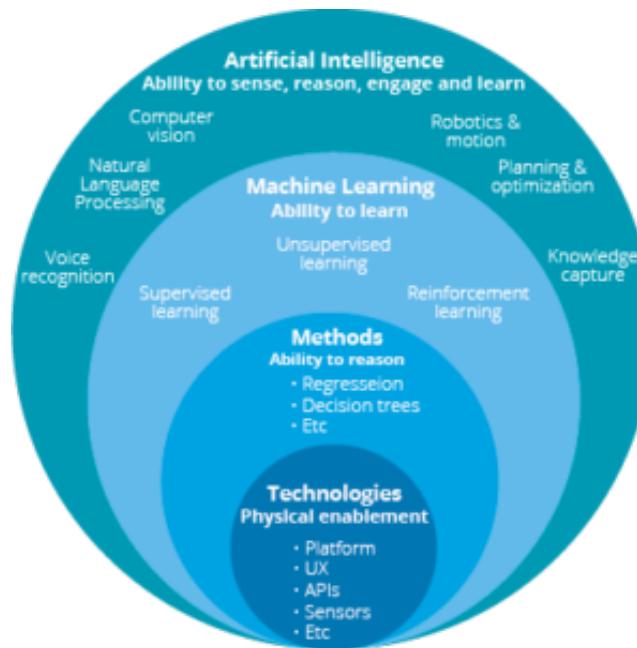


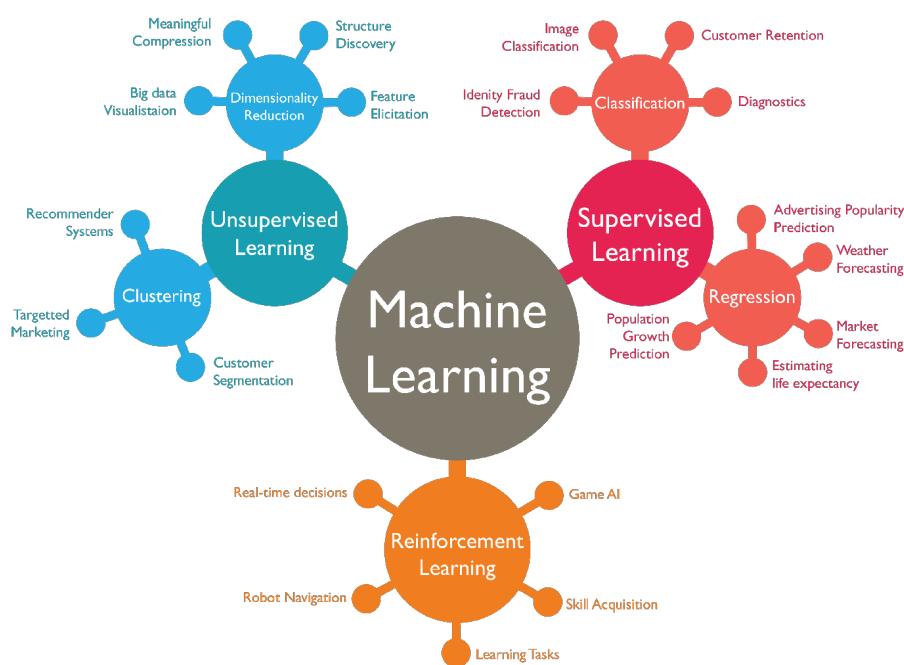
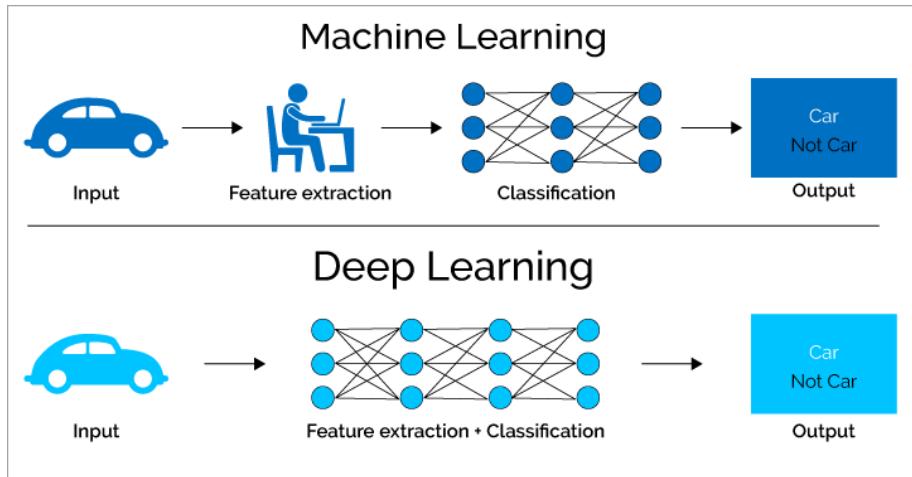


Thursday, 31 January 13

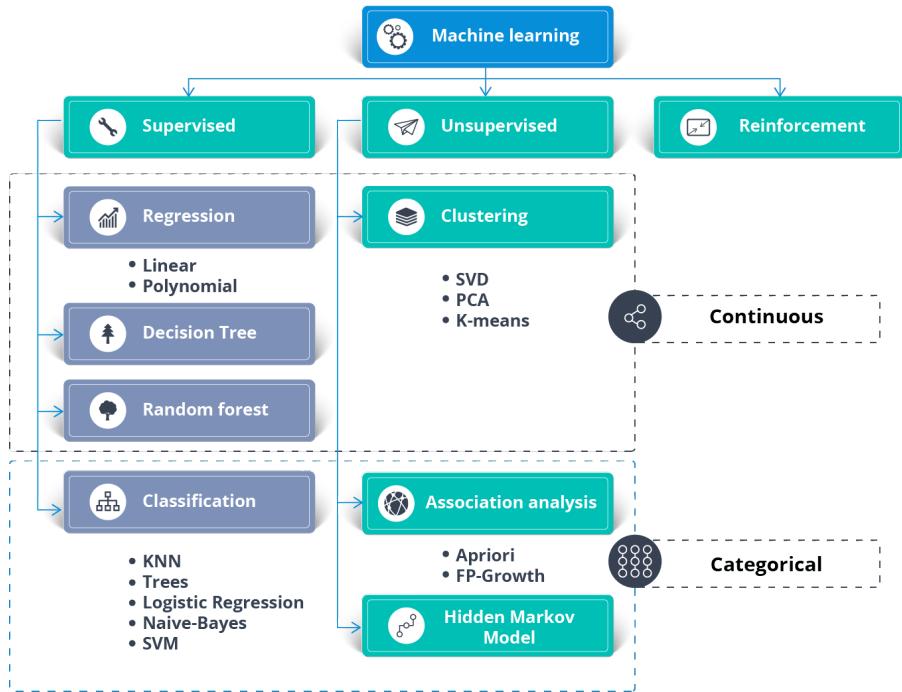
### 6.2.2 Big Data y Machine Learning

El Machine Learning o Aprendizaje Máquina es aquella parte de la inteligencia artificial con capacidad de aprender de los datos.

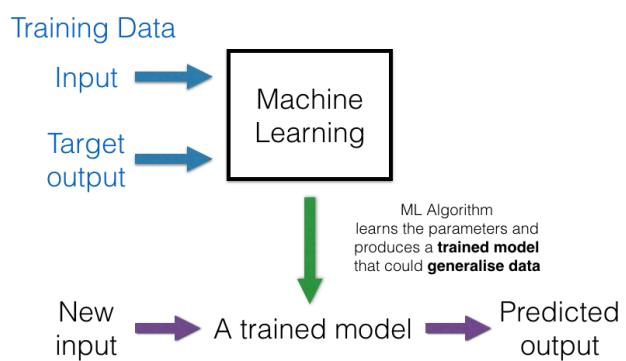




## 92 CAPÍTULO 6. TECNOLOGÍAS PARA EL TRATAMIENTO DE DATOS MASIVOS



Y un ejemplo de cómo se trabaja en machine learning:



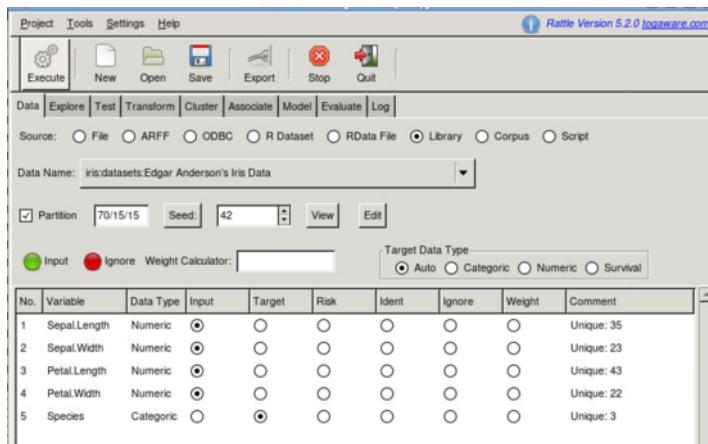
## 6.2. TECNOLOGÍAS BIG DATA (HADOOP/SPARK Y VISUALIZACIÓN)93

		Actual class (ground truth)		
Total (n)=100		Dog (Positive)	Not a Dog (Negative)	
Predicted class	Dog (Positive)	15 (TP)	20 (FP, Type I Error)	Precision =TP/(TP+FP) =0.42
	Not a Dog (Negative)	5 (FN, Type II Error)	60 (TN)	
Accuracy =(TP+TN)/Total =0.75	Sensitivity, Recall, TPR =TP/(TP+FN) = 0.75	FPR = FP/(FP+TN) =0.25	F1 Score =2*(Precision*Recall) /(Precision+Recall) =0.53	
Error Rate =(FP+FN)/Total =0.25	Miss Rate, FNR =FN/(TP+FN) =0.25	Specificity, TNR = TN/(FP+TN) =0.75		

### 6.2.3 Rattle como alternativa a RapidMiner en R

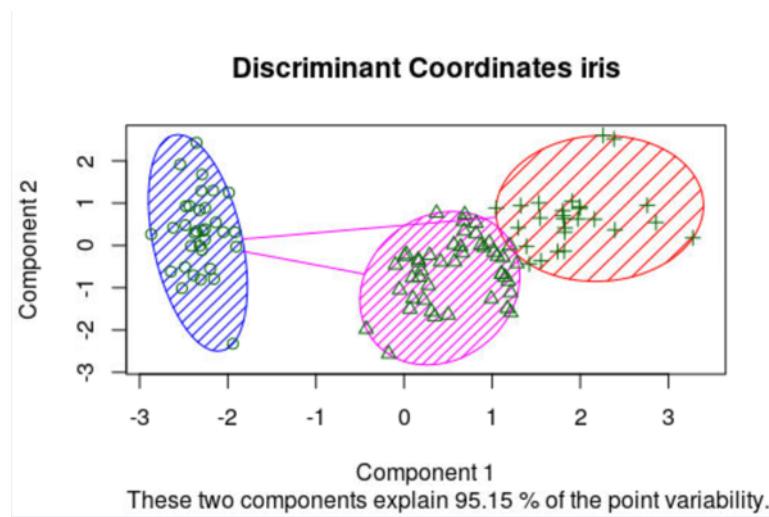
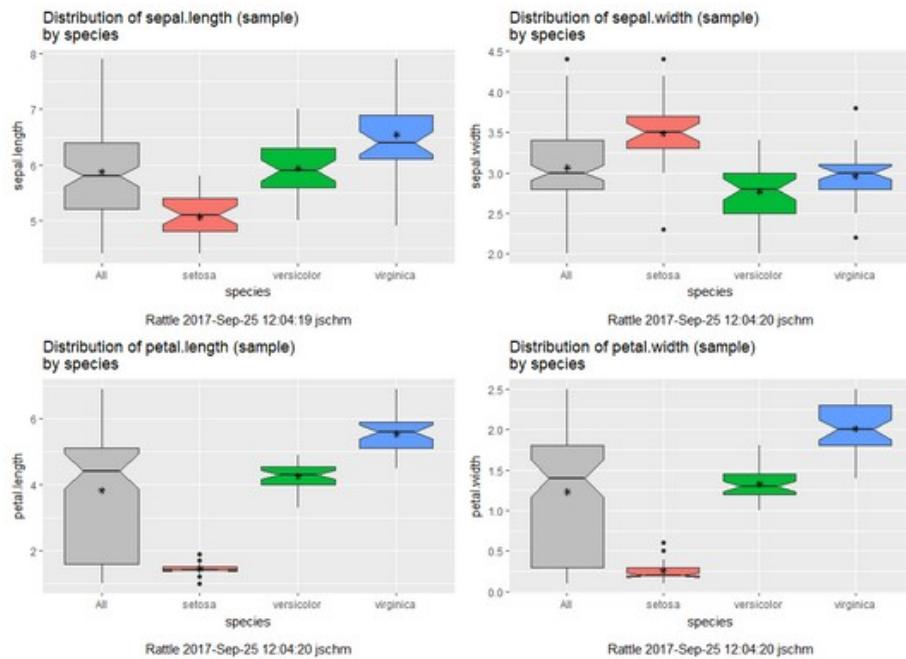
Las instrucciones para instalar R está en el Apéndice 3 de este documento

Un tutorial adecuado para introducirse en Rattle es éste



Con el tutorial se pueden ver las capacidades de rattle de explorar los datos, como se puede apreciar a continuación.

## 94 CAPÍTULO 6. TECNOLOGÍAS PARA EL TRATAMIENTO DE DATOS MASIVOS



### 6.2.4 Visualización y Generación de Cuadros de Mando

Se sigue un tutorial de la herramienta PowerBI, con datos de Excel y OData Feed

Como documentación de soporte se cuenta con la web de PowerBI y un tutorial adicional

### 6.3 Introducción al Análisis de Datos Masivos

En primer lugar se ha de considerar explorar los datos y realizar minería con ellos, y eso es posible hacerlo vía sparklyr.

Este apartado, eminentemente práctico, lo trabajaremos a través de la práctica 3 de TGD.

Working draft...



# Apéndice A

## Enlaces

**Recursos para el aprendizaje de R** (<https://rubenfcasal.github.io/post/ayuda-y-recursos-para-el-aprendizaje-de-r>): A continuación se muestran algunos recursos que pueden ser útiles para el aprendizaje de R y la obtención de ayuda...

*Ayuda online:*

- Ayuda en línea sobre funciones o paquetes: RDocumentation
- Buscador RSeek
- StackOverflow

*Cursos:* algunos cursos gratuitos:

- Coursera:
  - Introducción a Data Science: Programación Estadística con R
  - Mastering Software Development in R
- DataCamp:
  - Introducción a R
- Stanford online:
  - Statistical Learning
- Curso UCA: Introducción a R, R-commander y shiny
- Curso R CODER
- Udacity: Data Analysis with R
- Swirl Courses: se pueden hacer cursos desde el propio R con el paquete swirl.

Para información sobre cursos en castellano se puede recurrir a la web de R-Hispano en el apartado formación. Algunos de los cursos que aparecen en entradas antiguas son gratuitos. Ver: Cursos MOOC relacionados con R.

### *Libros*

- *Iniciación:*

- R for Data Science by Hadley Wickham and Garrett Grolemund (online, O'Reilly). online-castellano, O'Reilly.
- 2011 - The Art of R Programming. A Tour of Statistical Software Design, (No Starch Press)
- Hands-On Programming with R: Write Your Own Functions and Simulations, by Garrett Grolemund (O'Reilly)

- *Avanzados:*

- Advanced R by Hadley Wickham (online: 1<sup>a</sup> ed, 2<sup>a</sup> ed, Chapman & Hall)
- 2008 - Software for Data Analysis: Programming with R - Chambers (Springer)
- R packages by Hadley Wickham (online, O'Reilly)

- *Bookdown:* el paquete `bookdown` de R permite escribir libros empleando R Markdown y compartirlos. En <https://bookdown.org> está disponible una selección de libros escritos con este paquete (un listado más completo está disponible aquí). Algunos libros en este formato en castellano son:

- Introducción al Análisis de Datos con R (disponible en el repositorio de GitHub [rubenfcasal/intror](#)).
- Prácticas de Simulación (disponible en el repositorio de GitHub [rubenfcasal/simbook](#)).
- Escritura de libros con bookdown (disponible en el repositorio de GitHub [rubenfcasal/bookdown\\_intro](#)).
- R para profesionales de los datos: una introducción.
- Estadística Básica Edulcorada.

**Material online:** en la web se puede encontrar mucho material adicional, por ejemplo:

- CRAN: Other R documentation
- Blogs en inglés:
  - <https://www.r-bloggers.com/>
  - <https://www.littlemissdata.com/blog/rstudioconf2019>

- RStudio: <https://blog.rstudio.com>
- Microsoft Revolutions: <https://blog.revolutionanalytics.com>
- Blogs en castellano:
  - <https://www.datanalytics.com>
  - <http://oscarperpinan.github.io/R>
  - <http://rubenfcasal.github.io>
- Listas de correo:
  - Listas de distribución de r-project.org: <https://stat.ethz.ch/mailman/listinfo>
  - Búsqueda en R-help: <http://r.789695.n4.nabble.com/R-help-f789696.html>
  - Búsqueda en R-help-es: <https://r-help-es.r-project.narkive.com>  
<https://grokbase.com/g/r/r-help-es>
  - Archivos de R-help-es: <https://stat.ethz.ch/pipermail/r-help-es>

## A.1 RStudio

*RStudio:*

- Online learning
- Webinars
- sparklyr
- shiny

*tidyverse:*

- dplyr
- tibble
- tidyr
- stringr
- readr
- Databases using R, dplyr as a database interface, Databases using dplyr

*CheatSheets:*

- rmarkdown
- shiny

- dplyr
- tidyverse
- stringr

## A.2 Bibliometría

- CITAN
- scimetr
- bibliometrix
- wosr
- rwos
- rcrossref
- ropensci: Literature
- Diderot
- ...

## Apéndice B

# Instalación de R

En la web del proyecto R ([www.r-project.org](http://www.r-project.org)) está disponible mucha información sobre este entorno estadístico.

---



### The R Project for Statistical Computing

[\[Home\]](#)

**Download**

[CRAN](#)

**R Project**

- [About R](#)
- [Logo](#)
- [Contributors](#)
- [What's New?](#)
- [Reporting](#)
- [Bugs](#)
- [Development](#)
- [Site](#)
- [Conferences](#)
- [Search](#)

**Getting Started**

R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS. To [download R](#), please choose your preferred CRAN mirror.

If you have questions about R like how to download and install the software, or what the license terms are, please read our answers to [frequently asked questions](#) before you send an email.

**News**

- The R Foundation welcomes five new ordinary members: Jennifer Bryan, Dianne Cook, Julie Josse, Tomas Kalibera, and Balasubramanian Narasimhan.

R-project



**CRAN**

[Mirrors](#)

[What's new?](#)

[Task Views](#)

[Search](#)

[About R](#)

[R Homepage](#)

[The R Journal](#)

[Software](#)

[R Sources](#)

[R Binaries](#)

[Packages](#)

[Other](#)

[Documentation](#)

[Manuals](#)

[FAQs](#)

[Contributed](#)

**The Comprehensive R Archive Network**

**Download and Install R**

Precompiled binary distributions of the base system and contributed packages. **Windows and Mac** users most likely want one of these versions of R:

- [Download R for Linux](#)
- [Download R for \(Mac\) OS X](#)
- [Download R for Windows](#)

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

**Source Code for all Platforms**

Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!

- The latest release (Monday 2016-10-31, Sincere Pumpkin Patch) [R-3.3.2.tar.gz](#), read [what's new](#) in the latest version.
- Sources of [R alpha and beta releases](#) (daily snapshots).

CRAN

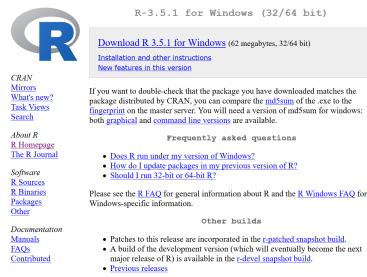
---

Las descargas se realizan a través de la web del CRAN (The Comprehensive R Archive Network), con múltiples mirrors:

- *Oficina de software libre* (CIXUG) <ftp.cixug.es/CRAN>.
- *Spanish National Research Network (Madrid)* (RedIRIS) es <cran.es.r-project.org>.

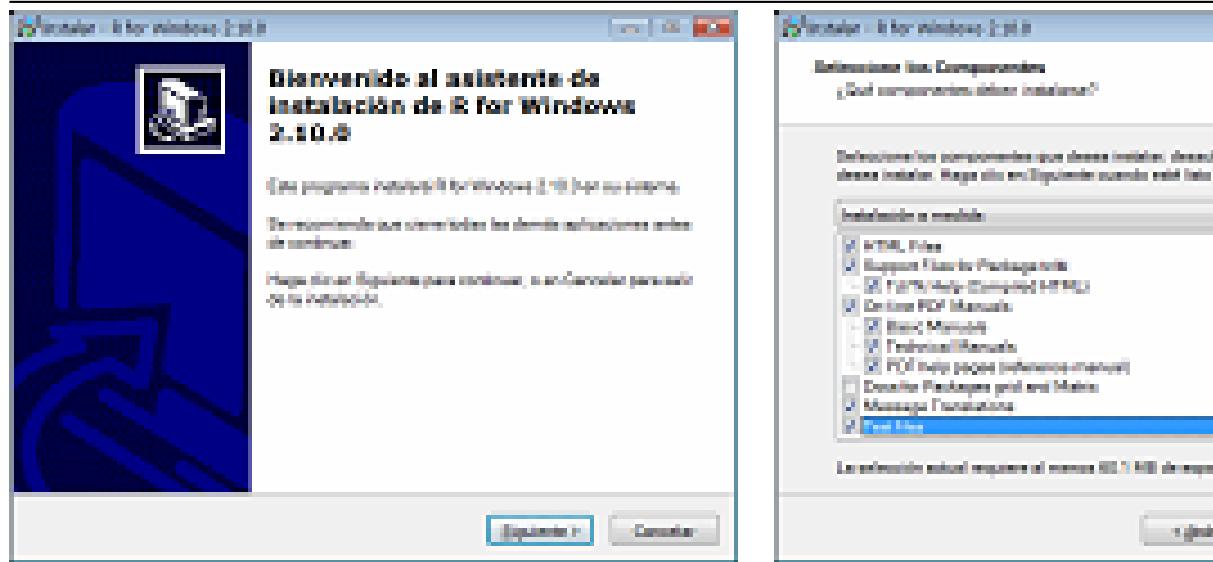
## B.1 Instalación de R en Windows

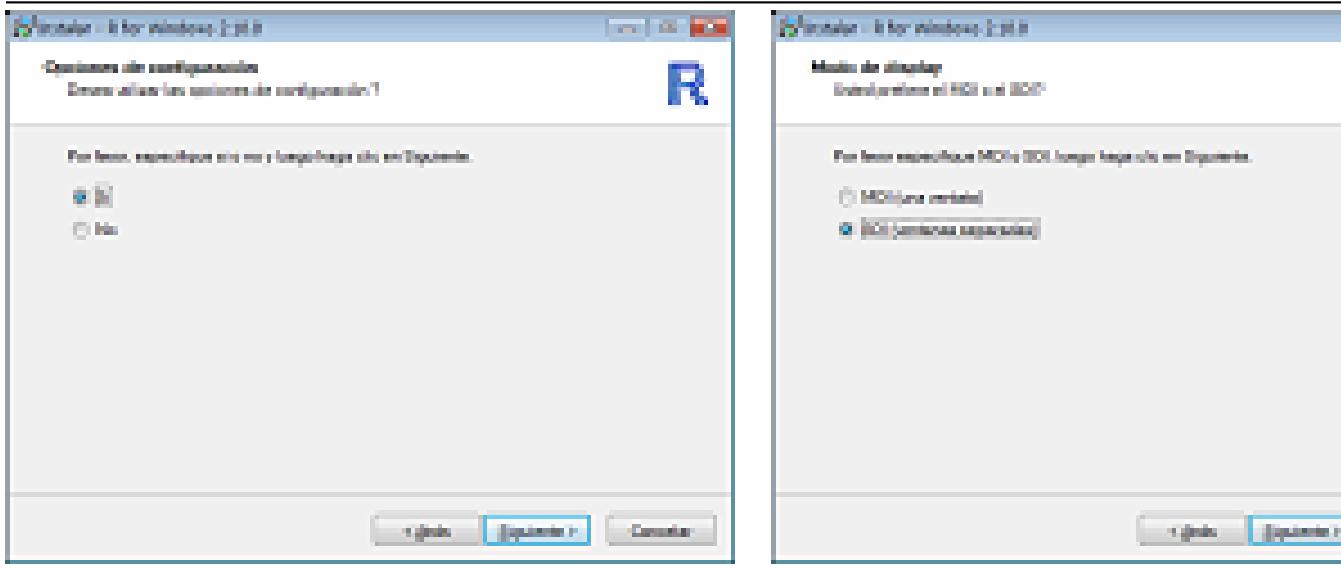
Seleccionando Download R for Windows y posteriormente base accedemos al enlace con el instalador de R para Windows.



### B.1.1 Asistente de instalación

Durante el proceso de instalación la recomendación (para evitar posibles problemas) es seleccionar ventanas simples SDI en lugar de múltiples ventanas MDI (hay que *utilizar opciones de configuración*).





Una vez terminada la instalación, al abrir R aparece la ventana de la consola (simula una ventana de comandos de Unix) que permite ejecutar comandos de R.

### B.1.2 Instalación de paquetes

Después de la instalación de R, puede ser necesario instalar paquetes adicionales (puede ser recomendable ejecutar R como *Administrador* para evitar problemas de permiso de escritura en la carpeta *library*<sup>1</sup>).

Para ejecutar los ejemplos mostrados en el libro será necesario tener instalados los siguientes paquetes: dplyr (colección tidyverse), tidyverse, stringr, readxl , openxlsx, RODBC, sqldf, RSQLite, foreign, magrittr, rattle, knitr. Por ejemplo mediante los comandos:

```
pkgs <- c('dplyr', 'tidyverse', 'stringr', 'readxl', 'openxlsx', 'magrittr',
         'RODBC', 'sqldf', 'RSQLite', 'foreign', 'rattle', 'knitr')
# install.packages(pkgs, dependencies=TRUE)
install.packages(setdiff(pkgs, installed.packages()[, 'Package']), dependencies = TRUE)
```

(puede que haya que seleccionar el repositorio de descarga, e.g. *Spain (Madrid)*).

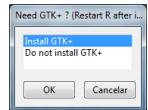
La forma tradicional es esta:

1. Se inicia R y se selecciona *Paquetes -> Instalar paquetes*

<sup>1</sup>Alternativamente se podrían proporcionar a los usuarios del equipo el permiso *control total* en la carpeta de instalación de R.

2. Se selecciona el repositorio.
3. Se selecciona el paquete y automáticamente se instala.

**Rattle** depende de la librería gráfica GTK+, al iniciarla por primera vez con el comando **library(rattle)** nos pregunta si queremos instalarla:



Pulsamos OK y reiniciamos R.

---

## B.2 Instalación en Mac OS X

Instalar R de <http://cran.es.r-project.org/bin/macosx> siguiendo los pasos anteriores.

Para instalar **rattle** seguir estos pasos (<https://zhiyuzo.github.io/installation-rattle>):

1. Instalar Homebrew: <https://brew.sh/>.
2. Ejecutar el siguiente código en la consola:

```
system('brew install gtk+')

local({
  if (Sys.info()[['sysname']] != 'Darwin') return()

  .Platform$pkgType = 'mac.binary.el-capitan'
  unlockBinding('.Platform', baseenv())
  assign('.Platform', .Platform, 'package:base')
  lockBinding('.Platform', baseenv())

  options(
    pkgType = 'both',
    install.packages.compile.from.source = 'always',
    repos = 'https://macos.rbind.org'
  )
})

install.packages(c('RGtk2', 'cairoDevice', 'rattle'))
```

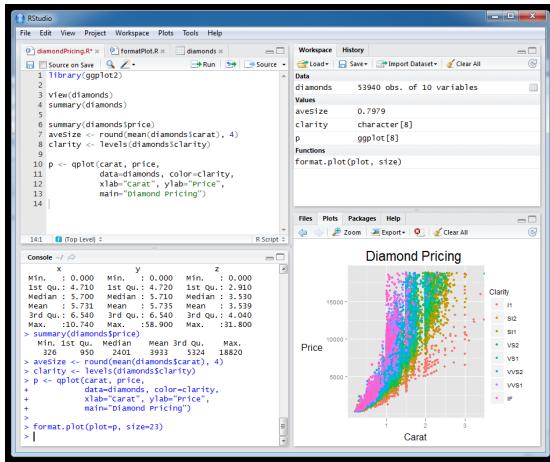
---

## B.3. INSTALACIÓN (OPCIONAL) DE UN ENTORNO O EDITOR DE COMANDOS105

### B.3 Instalación (opcional) de un entorno o editor de comandos

Aunque la consola de R dispone de un editor básico de código (script), puede ser recomendable trabajar con un editor de comandos más cómodo y flexible.

Un entorno de R muy recomendable es el **RStudio**, <http://rstudio.org>:



Para instalarlo descargar el archivo de instalación de <http://rstudio.org/download/desktop>.

#### B.3.1 Opciones adicionales

Nos puede interesar modificar las opciones por defecto en RStudio, por ejemplo que los gráficos se muestren en una ventana de R o que se emplee el navegador por defecto, para ello habría que modificar (con permisos de administrador) los archivos de configuración *Tools.R* y *Options.R* (en Windows se encuentran en la carpeta *C:\Program Files\RStudio\R*).

Para utilizar el dispositivo gráfico de R, modificar *Tools.R*:

```
# set our graphics device as the default and cause it to be created/set
.rs.addFunction( "initGraphicsDevice", function()
{
  # options(device="RStudioGD")
  # grDevices::deviceIsInteractive("RStudioGD")
  grDevices::deviceIsInteractive()
})
```

Para utilizar el navegador del equipo en lugar del visor integrado de R, modificar *Options.R*:

```
# # custom browseURL implementation
# options(browser = function(url)
```

```
# {  
#   .Call("rs_browseURL", url) ;  
# })
```

# Bibliografía

- Bezanson, J., Edelman, A., Karpinski, S., and Shah, V. B. (2017). Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98.
- James, G., Witten, D., Hastie, T., Tibshirani, R., et al. (2013). *An introduction to statistical learning*, volume 112. Springer.
- James, G., Witten, D., Hastie, T., Tibshirani, R., and Taylor, J. (2023). *An introduction to statistical learning: With applications in python*. Springer Nature.
- Little, R. J. (1988). A test of missing completely at random for multivariate data with missing values. *Journal of the American statistical Association*, 83(404):1198–1202.
- Van Buuren, S. (2018). *Flexible imputation of missing data*. CRC press.
- Van Rossum, G. and Drake Jr, F. L. (1995). *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam.
- Wickham, H., Çetinkaya-Rundel, M., and Grolemund, G. (2023a). *R for data science*. ” O'Reilly Media, Inc.”.
- Wickham, H., François, R., Henry, L., Müller, K., and Vaughan, D. (2023b). *dplyr: A Grammar of Data Manipulation*. R package version 1.1.4.