

# **Tratamiento de Datos en R**

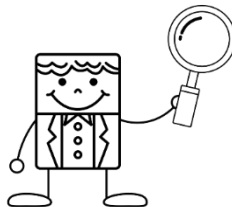




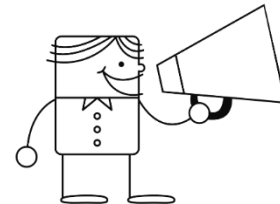
- Dentro del tema encontrarás los epígrafes de contenido, con subepígrafes para una mejor comprensión.
- Todos los temas han sido elaborados siguiendo el criterio de facilidad de exposición.
- Los contenidos incluyen, en este sentido, cinco tipo de llamadas de atención que agilizan y facilitan la comprensión de los mismos:



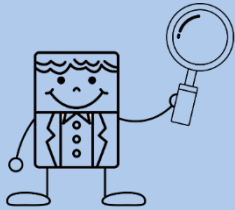
Ejemplo

Tenga en  
cuenta que....

Recuerde



Definición



*Tenga en cuenta que este curso tiene un componente práctico muy fuerte. Es labor del alumno practicar todos los ejemplos y ejercicios planteados.*

***“A programar se aprende programando”***

The background of the slide features a large, light blue 'R' logo, which is the symbol for the R programming language. The logo is semi-transparent and serves as a backdrop for the text.

**1**

# **Importación de Datos**

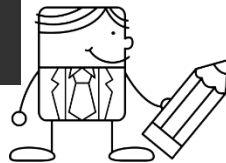
# Importación

- En la mayoría de casos los datos los encontraremos en diferentes formatos y provenientes de diferentes fuentes de información. Las más habituales:
  - Ficheros de texto plano: csv, tsv,...
  - Excel
  - Otras herramientas estadísticas: SAS, STATA, SPSS
  - BD relacionales
  - ...
- Existen numerosas funciones y paquetes en R para cargar información. Algunos ejemplos:
  - Fichero de texto plano: `utils`, `readr`, `data.table`
  - Excel: `readxl`, `gdata`, `XLConnect`
  - SAS, STATA, SPSS: `haven`, `foreign`
  - BD relacionales: `RMySQL`, `RPostgreSQL`, `ROracle`

# Importando desde ficheros de texto plano (I)

- Dentro del paquete *utils* (el cual se carga por defecto al iniciar la sesión de R) existen varias funciones para leer ficheros de texto plano y cargar su contenido en un *data frame*.
- La función más importante del paquete es `read.table(fileName)`. Dicha función tiene una enorme cantidad de parámetros que nos permitirán configurar la carga según el fichero. Los más importantes:
  - `header`: para indicar si el fichero tiene cabeceras.
  - `sep`: para indicar el separador de nuestro fichero.
  - `stringsAsFactors`: para indicar si las cadenas de caracteres se convertirán en factores.
- La ruta al fichero es relativa al directorio de trabajo.

```
mun1_1 <- read.table("dat/municipios1.csv",  
  header = TRUE,  
  sep = ",",  
  stringsAsFactors = FALSE)
```



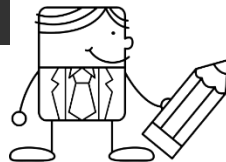
# Importando desde ficheros de texto plano (II)

- Existen otras funciones para leer directamente ficheros csv (coma como separador) o tsv (tabulador como separador):
  - Con punto como separador decimal
    - csv: `read.csv`
    - tsv: `read.delim`

```
# Con read.csv
mun1_2 <- read.csv("dat/municipios1.csv", stringsAsFactors = FALSE)
```

- Con coma como separador decimal
  - csv: `read.csv2`
  - tsv: `read.delim2`

```
# Con read.delim2
mun2_2 <- read.delim2("dat/municipios2.tsv", stringsAsFactors = FALSE)
```



# Importando desde ficheros de texto plano (III)

	<i>utils</i>	<i>readr</i>	<i>data.table</i>
Genérico	<code>read.table()</code>	<code>read_delim()</code>	fread
csv	<code>read.csv()</code>	<code>read_csv()</code>	
tsv	<code>read.delim()</code>	<code>read_tsv()</code>	

- *readr* es un paquete desarrollado por **Hadley Wickham** (creador de varios paquetes famosos de R como *ggplot2*).
- Es más rápido cargando datos que el paquete *utils*.
- Por defecto, no convierte las cadenas de caracteres en factores.
- *data.table* es un paquete para manipulación de datos en R (no está creado exclusivamente para leer ficheros).
- *fread* infiere los tipos de las columnas y los separadores.
- Extremadamente rápido.



# Importando desde ficheros Excel

- El paquete más simple para cargar datos desde Excel es *readxl*. Lee ficheros con extensión xls y xlsx. Muy similar a *readr* (mismo creador **Hadley Wickham**).
- Para leer los archivos de Excel nos apoyaremos en dos funciones:
  - `excel_sheets()`: devuelve la lista de hojas disponibles en el archivo.
  - `read_excel()`: realiza la carga. Algunos parámetros:
    - `sheet`: para indicar el número de hoja a cargar.
    - `col_names`: para indicar si la tabla tiene cabeceras.
    - `col_types`: para indicar el tipo básico de las columnas (también se pueden omitir)
    - `skip`: para omitir las primeras filas del fichero.
- Otros paquetes: *gdata* y *XLConnect*.



*Practica el Ejemplo **08\_import.R***

*¡No pierdas la pista a los ejemplos en la carpeta correspondiente!*

*Es importante practicar y entender todos los ejemplos, antes de lanzarse a realizar ejercicios.*





**2**

# **Tratamiento de Datos**

# Preparación y limpieza

- Muchos de los *datasets* a los que los que nos enfrentaremos estarán sucios de una manera o de otra. Por ejemplo:
  - Valores de una variable/atributo codificados como columnas.
  - Variable codificadas en filas y columnas.
  - Más de una variable contenida en la misma columna. Por ejemplo: trimestre y año.
  - Almacenar medidas en distintas unidades en la misma tabla.
  - Contienen *missing values*: valores desconocidos que habrá que eliminar o imputar. OJO→ Es importante saber por qué esos valores son desconocidos.
  - ...
- Al proceso, por el cual, transformamos un *dataset* en otro más conveniente para nuestro análisis, se le llama proceso de limpieza.
- Se suele decir que el proceso de limpieza y preparación de datos se lleva el **80% del tiempo**, mientras que el resto de tareas conllevan sólo el 20%. Es por este motivo, por el que el proceso de preparación es tan **importante**.

# Preparación y limpieza: Explorando

- El proceso de exploración tiene como objetivo crear una composición general de un *dataset*.
- Contiene tres fases:
  - Comprender y asimilar la estructura de los datos.
  - Ver los datos que lo componen.
  - Visualizar los datos que lo componen.
- Mediante estos tres sencillos pasos podremos identificar los primeros “problemas” en nuestros datos.
- Normalmente nuestro *dataset* estará contenido en un *data frame* de R, con las observaciones en las filas y las variables en las columnas.

# Preparación y limpieza: Explorando la estructura

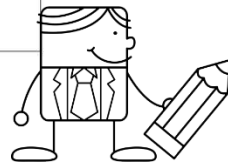
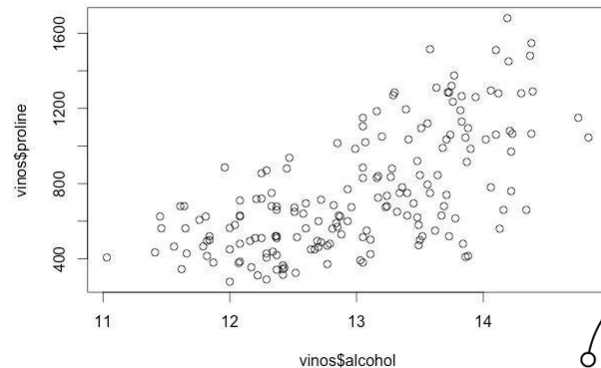
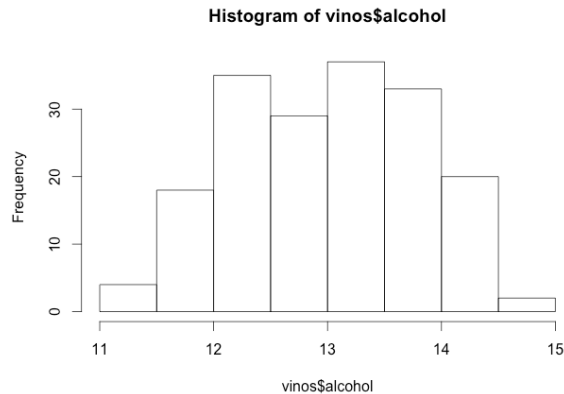
Función	Descripción
<code>class(x)</code>	Normalmente devolverá "data.frame".
<code>dim(x)</code>	Devuelve el número de filas y de columnas del <i>data frame</i> .
<code>names(x)</code>	Devuelve el nombre de las columnas del <i>data frame</i> .
<code>str(x)</code>	Devuelve información sobre la estructura del <i>data frame</i> : Número de filas u observaciones. Número de columnas o variables. Nombre de cada columna o variable. El tipo de cada variable o columna. Primeros valores u observaciones de cada columna.
<code>summary(x)</code>	Devuelve los estadísticos básicos de cada columna o variable: Mínimo. Máximo. Media. Cuartiles (mediana).

# Preparación y limpieza: Explorando los datos

Función	Descripción
head(x, n)	Muestra las primeras observaciones del <i>data frame</i> . Se puede establecer el número de observaciones devueltas con el parámetro <i>n</i> . Por defecto se muestran las seis primeras.
tail(x, n)	Muestra las últimas observaciones del <i>data frame</i> . Se puede establecer el número de observaciones devueltas con el parámetro <i>n</i> . Por defecto se muestran las seis últimas.
print(x)	Muestra el set de datos completo. OJO → Emplearlo sólo con <i>datasets</i> pequeños.

# Preparación y limpieza: Visualizando los datos

Función	Descripción
hist(x)	Muestra un histograma con la distribución de una determinada columna/variable.
plot(x,y)	Muestra un gráfico de puntos con la relación de dos variables.





# Preparación y limpieza: Visualizando los datos

```
# Cargamos el dataset
vinos <- read.csv("dat/wine.csv",
                  header = T)

# Exploración de la estructura
class(vinos)
dim(vinos)
names(vinos)
str(vinos)
summary(vinos)

# Exploración de los datos
head(vinos)
tail(vinos)
print(vinos)

# Visualizando los datos
hist(vinos$alcohol)
plot(vinos$alcohol, vinos$proline)
```



# Preparación y limpieza: Poniendo orden

- ¿Cuáles son los principios de unos datos “limpios”? [Artículo](#) de **Hadley Wickham** en **Journal of Statistical Software**.
- Estos principios son muy parecidos a los que aplicaríamos al diseñar una base de datos relacional.
- Los principios son:
  - Observaciones en filas.
  - Variables y atributos como columnas.
  - Cada fila de la tabla (*data frame*) contiene únicamente una unidad observacional. O lo que es lo mismo, no mezclar distintas entidades o cosas dentro de la misma tabla.
  - Todas las observaciones de una variable están medidas en la misma unidad.
- Los paquetes **tidyr** y **reshape2** de **Hadley Wickham**, contienen varias funciones que nos permitirán *poner orden* en nuestros datos.


# Preparación y limpieza: datos limpios y ordenados

name	age	eye_color	height
Jake	34	Other	6'1"
Alice	55	Blue	5'9"
Tim	76	Brown	5'7"
Denise	19	Other	5'1"



# Preparación y limpieza: valores como columnas

name	age	brown	blue	other	height
Jake	34	0	0	1	6'1"
Alice	55	0	1	0	5'9"
Tim	76	1	0	0	5'7"
Denise	19	0	0	1	5'1"



name	age	eye_color	height
Jake	34	Other	6'1"
Alice	55	Blue	5'9"
Tim	76	Brown	5'7"
Denise	19	Other	5'1"

## Preparación y limpieza: variables en filas y columnas

name	measurement	value
Jake	n_dogs	1
Jake	n_cats	0
Jake	n_birds	1
Alice	n_dogs	1
Alice	n_cats	2
Alice	n_birds	0

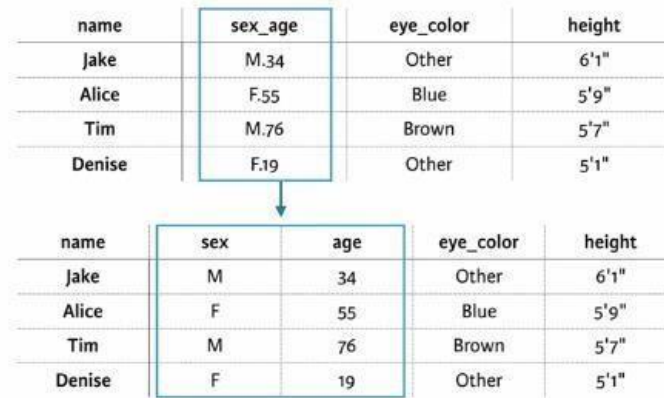
  

name	n_dogs	n_cats	n_birds
Jake	1	0	1
Alice	1	2	0

## *tidyr*: quitando valores como columnas

- Para corregir los problemas anteriores usaremos las funciones **gather** y **spread** del paquete *tidyr*.
- La función **gather** une columnas en pares clave valor. Los parámetros de la función son:
  - **data**: el *data frame* de datos a corregir.
  - **key**: el nombre de la nueva columna que contendrá la clave.
  - **value**: el nombre de la nueva columna que contendrá el valor.
  - **...**: nombres de las columnas a juntar (o no).
- La función **spread** separa pares clave valor en columnas. Los parámetros de la función son:
  - **data**: el *data frame* de datos a corregir.
  - **key**: el nombre de la columna que contiene la clave.
  - **value**: el nombre de la columna que contiene el valor.

# Preparación y limpieza: dos variables en la misma columna



The diagram illustrates the process of splitting a combined 'sex\_age' variable into two separate variables, 'sex' and 'age'. An arrow points from the 'sex\_age' column in the top table to the 'sex' and 'age' columns in the bottom table.

name	sex_age	eye_color	height
Jake	M.34	Other	6'1"
Alice	F.55	Blue	5'9"
Tim	M.76	Brown	5'7"
Denise	F.19	Other	5'1"

name	sex	age	eye_color	height
Jake	M	34	Other	6'1"
Alice	F	55	Blue	5'9"
Tim	M	76	Brown	5'7"
Denise	F	19	Other	5'1"

## *tidyr*: separando columnas

- Para corregir el anterior problema usaremos la función **separate** del paquete *tidyr*. Esta función separa una columna en múltiples columnas. Los parámetros de la función son:
  - **data**: el *data frame* de datos a corregir.
  - **col**: el nombre de la columna a separar.
  - **into**: vector de caracteres con el nombre de las nuevas columnas.
  - **sep**: el separador a emplear. Por defecto se usa como separador cualquier carácter no alfanumérico.
- Existe otra función en el paquete *tidyr* para realizar el proceso inverso, **unite**. Esta función une varias columnas en una única columna. Los parámetros de la función son:
  - **data**: el *data frame* de datos a corregir.
  - **col**: el nombre de la nueva columna que contendrá la unión.
  - **...**: el nombre de las columnas a unir.
  - **sep**: el separador a emplear. Por defecto: "\_".



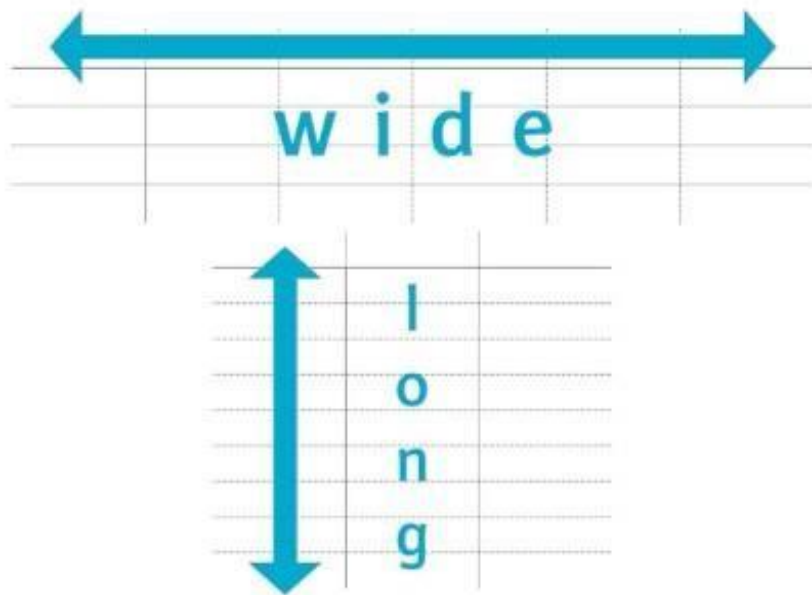
# Preparación y limpieza: más de una medida observacional en la misma tabla

name	age	height	pet_name	pet_type	pet_height
Jake	34	6'1"	Larry	Dog	25"
Jake	34	6'1"	Chirp	Bird	3"
Alice	55	5'9"	Wally	Dog	30"
Alice	55	5'9"	Sugar	Cat	10"
Alice	55	5'9"	Spice	Cat	12"



- Existen personas duplicadas (x3). Esto puede indicar que tenemos que separar en dos *data frames* los datos.
- OJO → A la hora de separar en varios *data frames* debemos crear una PK que nos permita cruzarlos. Similar a las *Primary Keys* de una base de datos relacional.

## Preparación y limpieza: *wide* vs *long*



- Hablamos de *wide datasets* cuando tienen más variables que observaciones.
- Hablamos de *long datasets* cuando tienen más observaciones que atributos.
- Un *wide dataset* puede indicar que tenemos valores almacenados como columnas. Trataremos de convertirlo en un *long dataset*.

## *reshape2: wide dataset to long dataset*

- Dentro del paquete *reshape2* existe una función, análoga a **gather** de *tidyr*, para pasar de un *wide dataset* a un *long dataset*: **melt**.
- La función **melt** recibe los siguientes parámetros:
  - **data**: el *data frame* de datos a pivotar.
  - **id.vars**: un vector con el nombre de las variables a mantener.
  - **variable.name**: el nombre de la nueva variable que contendrá las variables fundidas.
  - **value.name**: el nombre de la nueva variable que contendrá los valores.

## *reshape2: long dataset to wide dataset*

- Dentro del paquete *reshape2* existe otra función, análoga a **spread** de *tidyr*, para pasar de un *wide dataset* a un *long dataset*: **dcast**.
- La función **dcast** recibe los siguientes parámetros:
  - **data**: el *data frame* de datos a pivotar.
  - **variable1 + variable2 + ... ~ variable**: *variable1*, *variable2*... son las columnas que se mantendrán y *variable* la columna que contiene los valores que generarán nuevas columnas.
  - **value.var**: nombre de la columna donde están los valores.
  - **fun.aggregate**: en el caso de tener múltiples valores por fila especifica la función de agregación. Por ejemplo: *sum*, *mean*, *max*...

# Preparación y limpieza: *tidyr*

```
library(tidyr)

wide_df <- data.frame(col = c('X', 'Y'), A = c(1, 4), B = c(2, 5), C = c(3, 6))

# Gather
long_df <- gather(wide_df, my_key, my_val, -col)

# Spread
new_wide_df <- spread(long_df, my_key, my_val) #Obtenemos el data.frame original

treatments <- data.frame(patient = c('X', 'Y', 'X', 'Y', 'X', 'Y'),
                          treatment = c('A', 'A', 'B', 'B', 'c', 'C'),
                          year_mo = c('2010-10', '2010-10', '2012-08',
                                       '2012-08', '2014-12', '2014-12'),
                          responde = c(1, 4, 2, 5, 3, 6))

# Separate
treatments_sep <- separate(treatments, year_mo, c("year", "month"))

# Unite
new_treatments <- unite(treatments_sep, year_mo, year, month, sep = "-")
```



# Preparación y limpieza: *reshape2*

```
library(reshape2)

wide_df <- data.frame(col = c('X', 'Y'), A = c(1, 4), B = c(2, 5), C = c(3, 6))

# melt
long_df <- melt(wide_df, id.vars = c("col"), variable.name = "my_key", value.name = "my_val")

# dcast
new_wide_df <- dcast(long_df, col ~ my_key, value.var = "my_val")
```



# Preparación y limpieza: preparando los datos para el análisis

- El último paso del proceso de limpieza es la preparación de los datos para el análisis.
- Normalmente, nuestro *dataset* contendrá distintos tipos de variables (cadenas de caracteres, numéricas, lógicas, fechas...).
- Es importante asegurar que cada variable está almacenada en el formato adecuado.
- Otro punto especialmente importante es el tratamiento de *missing values* y *outliers* (valores atípicos).

# Preparación y limpieza: conversión de tipos (I)

Función	Descripción
as.factor(x)	Realiza la coerción de x a factor.
as.character(x)	Realiza la coerción de x a cadena de caracteres.
as.numeric(x)	Realiza la coerción de x a número real.
as.integer(x)	Realiza la coerción de x a entero.
as.logical(x)	Realiza la coerción de x a valor lógico.
as.Date(x, f)	Parsea una fecha según el formato establecido por parámetro.
as.POSIXct(x, f)	Parsea una fecha y hora según el formato establecido por parámetro.



# Preparación y limpieza: *missing values*

- ¿Por qué no están los datos? Es importante descubrir y saber el motivo por el cual faltan datos en un *dataset*.
- Dependiendo del motivo tendremos principalmente dos opciones:
  - Borrarlos
  - Imputarlos
    - Ceros
    - Media/Mediana
    - Empleando un modelo para rellenarlos
- En R los *missing values* se representan con NA.
- Otros valores especiales:
  - Inf: ¿pueden representar *outliers*? Divisiones por cero
  - NaN: ¿pueden representar errores?

## Preparación y limpieza: ¿cómo encontrar *missing values*?

- Para encontrar *missing values* podemos utilizar algunas de las funciones que hemos visto anteriormente. Por ejemplo: `is.na(x)`, `summary(x)`...

```
#NAs
df <- data.frame(A = c(1, NA, 8, NA),
                 B = c(3, NA, 88, 23),
                 C = c(2, 45, 3, 1))

#Detección
is.na(df) #Util en dataset pequeños.
any(is.na(df))
sum(is.na(df))
summary(df)
```

- Si finalmente decidimos eliminarlos podemos utilizar la función `complete.cases(x)` para indexar y eliminar o directamente `na.omit(x)`

```
#Eliminación
df[complete.cases(df), ]
na.omit(df)
```



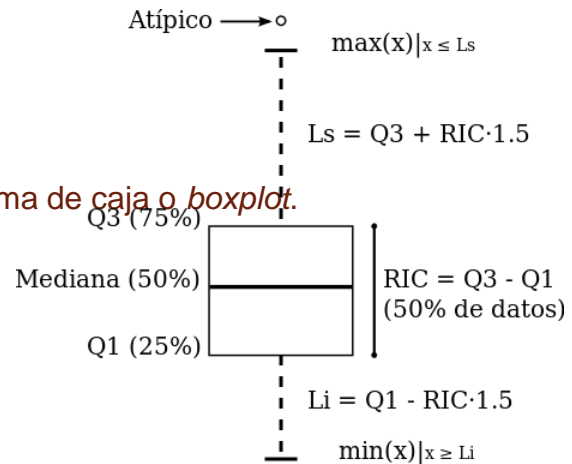
# Preparación y limpieza: *outliers*

○ Los *outliers*, o valores atípicos, son valores extremos, distantes del resto de los valores.

○ Algunas causas por las que pueden existir *outliers*:

- Son errores de medición
- Son errores producidos al transcribir los datos
- Son medidas válidas
- Representar valores por defecto
- ...

○ Una de las herramientas más útiles será el diagrama de caja o *boxplot*.



# Preparación y limpieza: ¿cómo encontrar *outliers*?

- Existen varias herramientas para encontrar *outliers*:

- `boxplot`
- `hist`
- `summary`

```
#Outliers
set.seed(10)
x <- c(rnorm(30, mean = 15, sd = 5), -5, 28, 35)

boxplot(x, horizontal = T)

df2 <- data.frame(A = rnorm(100, 50, 100),
                  B = c(rnorm(99, 50, 10), 500),
                  C = c(rnorm(99, 50, 10), -1))

hist(df2$B, 20) #¿Qué le pasa al 500? Es un error de medición, 50
boxplot(df2)
```

- Dependiendo del contexto debemos decidir que hacer:

- Eliminarlos
  - ¿Puede la edad de una persona ser negativa? Valores sin sentido.
  - ¿Y superior a 200? Valores tan extremos que no tienen sentido.
- Dejarlos ¿Puede la edad de una persona ser 110?



## Ejercicio 6

- Realiza el ejercicio del fichero *Ejercicio06\_tidyr.R*



*Es importante haber practicado y entendido todos los ejemplos anteriormente expuestos (**09\_cleaning.R**) antes de realizar el ejercicio.*

*Intenta realizar los ejercicios y si te atascas tienes las soluciones a tu disposición.*



The background of the slide features a large, light blue 'R' logo, which is the symbol for the R programming language. The logo is semi-transparent and serves as a backdrop for the title text.

**3**

# **Manipulación de Datos**

# Manipulación

- La mayoría de los *datasets* contienen pequeños *insights* que no están accesibles de manera inmediata.
- No hablamos del tipo de conocimiento que nos proporcionan los algoritmos de *machine learning* o de modelado, sino de cosas más simples. Por ejemplo:
  - Nuevas variables.
  - Estadísticas básicas.
  - Diferencias entre grupos.
  - ...
- Existen paquetes que nos ayudarán a extraer esta información:
  - *dplyr*
  - *data.table*

The diagram illustrates data manipulation steps. It starts with a wide table with 4 columns: Group, dose 1, dose 2, and Sum. This table is transformed into a narrow table with 2 columns: Group and Total. Finally, a summary table is shown with 4 columns: n, min, mean, and max.

Group	dose 1	dose 2	Sum
A	3	3	6
A	4	5	9
B	3	1	4
B	1	3	4
C	1	3	4
C	2	2	4

Group	Total
A	15
B	8
C	8

n	min	mean	max
6	4	5.2	9

# *dplyr*

○ Paquete de R que define una gramática de manipulación de datos. Creado por **Hadley Wickham**. Las operaciones básicas que realizar son:

- Selección de variables (**select**).
- Creación de nuevas variables (**mutate**).
- Filtrado de observaciones (**filter**).
- Ordenación de observaciones (**arrange**).
- Agrupación de observaciones y cálculo de estadísticos agregados (**group\_by & summarise**).

○ Todas las funciones anteriores devuelven copias del *dataset* original.

○ Funciona muy rápido (está implementado en C++).

○ Su sintaxis puede recordar a SQL.

○ El paquete *dplyr* funciona tanto con objetos *data.frame* como *tbl*.



## dplyr: tbl, tipo especial de data.frame

- Dentro del paquete dplyr existe un nuevo tipo de objeto, la tabla (**tbl**), similar a un *data frame*.
- Tiene una gran ventaja al trabajar con *datasets* grandes, ya que permite visualizar su contenido de una forma más *amigable*. Al mostrar el contenido de una variable del tipo tabla, el contenido se adapta al tamaño de la pantalla (*responsive*).
- Para ver el contenido de una tabla existe el comando `glimpse(x)`, cuyo resultado es similar al `str(x)` sobre un *data frame*.

```
library(hflights)
library(dplyr)

hflights
hflights <- tbl_df(hflights)
hflights
glimpse(hflights)
```



## *dplyr*: selección de variables

- Para hacer selección de variables utilizaremos la función **select** del paquete *dplyr*. Los parámetros de la función son:
  - `df`: el *data frame* o tabla sobre la que actuar.
  - `...`: nombres de las variables a seleccionar. Se puede combinar con el operador `:` para definir rangos de variables o `-` para decidir las que no.
- Existe un conjunto de funciones auxiliares que permiten hacer la selección de las variables de manera más ágil:
  - `starts_with("x")`: selecciona las variables que comienzan con "x".
  - `ends_with("x")`: selecciona las variables que terminan con "x".
  - `contains("x")`: selecciona las variables que contienen "x".
  - `matches("x")`: selecciona las variables que casan con "x". Se puede combinar con expresiones regulares.
  - `num_range("x", 1:5)`: variables de la lista x01, x02, ..., x05.
  - `one_of(x)`: aquellas variables en el vector de caracteres x.

## *dplyr*: creación de variables

- Para añadir nuevas variables utilizaremos la función **mutate** del paquete *dplyr*. Los parámetros de la función son:
  - `df`: el *data frame* o tabla sobre la que actuar.
  - `new_column = expresion`: el nombre de la nueva variable y la expresión que la calcula.
  - `...`: se pueden añadir más de una variable nueva al mismo tiempo.
- **Tip:** puedes emplear las nuevas variables para crear otras en la misma llamada.

## *dplyr*: filtrar observaciones

- Para filtrar las filas de un *dataset* utilizaremos la función **filter** del paquete *dplyr*. Los parámetros de la función son:
  - **df**: el *data frame* o tabla sobre la que filtrar.
  - **logical test**: condición de filtrado sobre aquellas filas que cumplan el test.
- La función **filter** permite utilizar los operadores relacionales y lógicos de R.

## *dplyr*: ordenando observaciones

- Para ordenar las filas de un *dataset* utilizaremos la función **arrange** del paquete *dplyr*. Los parámetros de la función son:
  - **df**: el *data frame* o tabla sobre el que ordenar.
  - **...**: las variables sobre las que se ordenará.
- Por defecto, la función **arrange** ordena de menor a mayor (ascendentemente).
- Para ordenar de mayor a menor (descendentemente), utilizamos la función **desc(x)**.

## *dplyr*: agregar observaciones

- Para agrupar las filas de un *dataset* utilizaremos la función **group\_by** del paquete *dplyr*. Los parámetros de la función son:
  - **df**: el *data frame* o tabla sobre la que se realizará la agrupación.
  - **...**: las variables sobre las que se agrupará.
- La función **group\_by** se suele usar conjuntamente con otra función, **summarise**. Dicha función, permite calcular un resumen de estadísticos básicos que describen el *dataset* o la agrupación. Los parámetros de la función son:
  - **df**: el *data frame* o tabla sobre la que actuar.
  - **new\_column = expresion**: el nombre del nuevo estadístico y la expresión que lo calcula.
  - **...**: se pueden añadir más de una variable nueva al mismo tiempo.
- **Tip**: al igual que con **mutate**, al usar **summarise** puedes emplear las nuevas variables para crear otras en la misma llamada.

## *dplyr*. summarise (I)

- La función **summarise** permite utilizar funciones de agregación de R:
  - `min(x)`: valor mínimo del vector x.
  - `max(x)`: valor máximo del vector x.
  - `mean(x)`: valor medio del vector x.
  - `median(x)`: valor mediano del vector x.
  - `quantile(x, p)`: el cuantil p del vector x.
  - `sd(x)`: la desviación estándar del vector x.
  - `var(x)`: la varianza del vector x.
  - `IQR(x)`: el rango intercuartílico del vector x.

## *dplyr*. summarise (II)

- El paquete dplyr completa la lista de funciones anteriores con algunas propias:
  - `first(x)`: primer elemento del vector `x`.
  - `last(x)`: último elemento del vector `x`.
  - `nth(x, n)`: el enésimo elemento del vector `x`.
  - `n()`: número de filas del *data frame*, tabla o grupo.
  - `n_distinct(x)`: número de valores únicos del vector `x`.



## *dplyr: pipes operator*

- El paquete *dplyr* permite utilizar el **pipe operator** de R para encadenar llamadas a funciones.
  - Envía la salida de una función a la siguiente.
  - Ahorra espacio (no es necesario declarar variables intermedias).
  - Simplifica la lectura de código.

```
summarise(  
  mutate(  
    filter(  
      select(a, X, Y, Z),  
        X > Y),  
    Q = X + Y + Z),  
  all = sum(Q))  
)
```



```
a %>%  
  select(X, Y, Z) %>%  
  filter(X > Y) %>%  
  mutate(Q = X + Y + Z) %>%  
  summarise(all = sum(Q))
```



## *dplyr*: combinando

- El paquete *dplyr* también contiene algunas funciones para realizar operaciones de unión o combinación de *data frames* o tablas.
- Las funciones son **inner\_join**, **left\_join**, **right\_join**, **full\_join**. Los parámetros son comunes para todas ellas:
  - *x*, *y*: *data frames* o tablas a unir.
  - *by*: vector de caracteres con las columnas por las que se realizará la unión. Por defecto, buscará aquellas variables con el mismo nombre.

# Manipulación: *dplyr*

```
# Select
hflights.select <- select(hflights.tbl, ActualElapsedTime, AirTime, ArrDelay, DepDelay)
hflights.select

# Mutate
hflights.mutate <- mutate(hflights.select, loss = ArrDelay - DepDelay)
hflights.mutate

# Filter
hflights.select <- select(hflights.tbl, starts_with("cancel"), DepDelay)
hflights.select
hflights.filter <- filter(hflights.select, Cancelled == 1)
hflights.filter

# Arrange
hflights.select <- select(hflights.tbl, TailNum, contains("Delay"))
hflights.select
hflights.arrange <- arrange(hflights.select, DepDelay)
hflights.arrange
hflights.arrange <- arrange(hflights.select, DepDelay, ArrDelay)
hflights.arrange

# Summarise
hflights.select <- select(hflights.tbl, TailNum, contains("Delay"))
hflights.select <- filter(hflights.select, !is.na(DepDelay))
hflights.summarise <- summarise(hflights.select, min = min(DepDelay), max = max(DepDelay), mean = mean(DepDelay),
                               median = median(DepDelay))
hflights.summarise

# Group by
hflights.group <- group_by(hflights.tbl, UniqueCarrier)
hflights.summarise.group <- summarise(hflights.group,
                                       avgDep = mean(DepDelay, na.rm = T),
                                       avgArr = mean(ArrDelay, na.rm = T))

hflights.summarise.group

# %>%
hflights.tbl %>%
  filter(!is.na(DepDelay)) %>%
  summarise(min = min(DepDelay), max = max(DepDelay), mean = mean(DepDelay), median = median(DepDelay))
```



## Ejercicio 7

- Realiza el ejercicio del fichero *Ejercicio07\_dplyr.R*



*Es importante haber practicado y entendido todos los ejemplos anteriormente expuestos (**10\_dplyr.R**) antes de realizar el ejercicio.*

*Intenta realizar los ejercicios y si te atascas tienes las soluciones a tu disposición.*

# Dónde encontrar esto y más

- DataCamp: <https://www.datacamp.com/>
- R Programming: <https://es.coursera.org/learn/r-programming>

- R in a nutshell
- R cookbook

