Introducción a R

Bioestadística

Marta Coronado (marta.coronado@uab.cat)

Grado en Genética | Curso 2025/26

Facultat Biociències UAB

Outline

1. Introducción a R

- R como una potente calculadora
- R para estadística descriptiva básica

2. Manipulación de datos

- Introducción a `Tidyverse`
- Abrir y leer ficheros
- Concepto de "*pipe*"
- Limpiar los datos
- Filtrar, seleccionar, crear y transformar columnas de un `data.frame`
- Maneras de leer y guardar datos

3. Estadística descriptiva con R

- Funciones útiles del paquete `Tidyverse`
- Ejemplos prácticos

01

Introducción a R

R es una potente calculadora

🖩 R permite realizar cálculos interactivos:

```
2+2
```

```
## [1] 4
```

```
sqrt(139)^cos(1.3)-log10(8)
```

```
## [1] 1.031669
```

```
a <- sqrt(139)^cos(1.3)-log10(8)
b <- 4
c <- a^b
c
```

```
## [1] 1.132824
```

Hemos **asignado** un valor a las variables a, b y c con "<-".

R dispone de muchas funciones matemáticas *built-in*.

Funciones matemáticas *built-in*

Comando de R	Función
`abs()`	Valor absoluto | x |
`cos()`, `sin()`, `tan()`	Coseno, seno, tangente
`exp(x)`	Función exponencial, e^x
`log(x)`	Logaritmo natural (base e), $\log x = \ln x$
`log10(x)`	Logaritmo en base 10, $\log_{10} x$
`sum()`	Suma de los valores de un vector
`cumsum()`	Suma acumulada

Funciones matemáticas *built-in*

Ejemplo con `sum()`

```
# Simulate daily cases over a 30-day period
cases_per_day <- c(12, 25, 43, 58, 32, 19, 61, 54, 33, 27, 45, 48, 61, 39, 21, 18, 39, 56,
    44, 52, 35, 27, 42, 51, 36, 29, 49, 47, 52, 37)
total_cases <- sum(cases_per_day)
total_cases
```

```
## [1] 1192
```

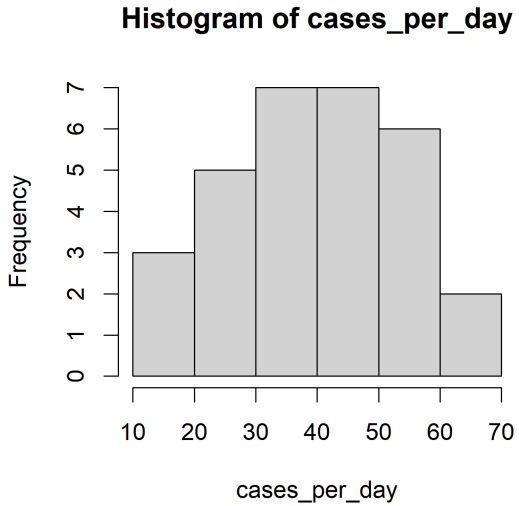
La función `c()` combina los números en un **vector**.

Funciones estadísticas *built-in*

R también ofrece funciones *built-in* para realizar **estadística simple**: gráfica y descriptiva.

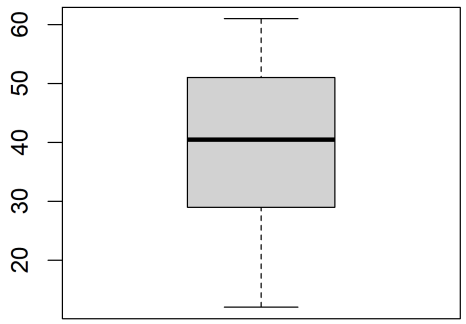
Por ejemplo, para crear un **histograma**:

```
hist(cases_per_day)
```



Para crear un **boxplot**:

```
boxplot(cases_per_day)
```



Nota. Aprenderemos a hacer gráficos más completos durante el curso.

Funciones estadísticas *built-in*

Comando de R	Función
`mean(x)`	Media aritmética de x
`exp(mean(log(x)))`	Media geométrica de x
`1/mean(1/x)`	Media armónica de x
`median(x)`	Mediana de x
`min(x)`, `max(x)`	Mínimo y máximo de x
`range(x)`	Rango de x
`sd(x)`	Desviación estándar de x
`var(x)`	Varianza de x

Ejemplo con `median()`

```
median(cases_per_day)
```

```
## [1] 40.5
```

02

Manipulación de datos

Introducción a Tidyverse

El conjunto de paquetes integrados de Tidyverse está diseñado para hacer que las operaciones comunes en la manipulación de datos sean más fáciles de usar.
Los paquetes incluyen funciones para organizar, ordenar, leer/escribir, analizar y visualizar datos, entre otros.


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

Pipes (%>%)

Para hacer que el código R sea más legible, las herramientas de Tidyverse utilizan la canalización (*pipes*), %>% (desde R 4.10, se puede utilizar |>), que forma parte del paquete dplyr que se instala automáticamente con Tidyverse: permite que la salida de un comando se use como entrada para otro.

```
sqrt(83) %>% round(digits = 2) # Ctrl + Shift + M
```

```
## [1] 9.11
```

Trabajando con datos

Leer datos en R

Independientemente del análisis que estemos realizando, generalmente necesitamos leer datos. La función que usemos en R dependerá del tipo de archivo de datos que estemos importando (por ejemplo, texto, Excel, etc.) y de cómo estén separadas o delimitadas las datos en ese archivo.

Tipo de datos	Extensión	Función	Paquete
Valores separados por comas	`csv`	`read.csv()`	`utils` (por defecto)
		`read_csv()`	`readr`
Valores separados por tabulaciones	`tsv`/`tab`	`read_tsv()`	`readr`
Otros	`txt`	`read.table()`	`utils`
		`read_delim()`	`readr`
Excel	`xlsx`, `xls`	`read_excel()`	`readxl`

Trabajando con datos

Leer datos en R

```
metadata <- read.csv(file="data/virus_exp_design.csv", header = T)

metadata <- read.table(file="data/virus_exp_design.csv", sep = ",", header = T)

metadata <- read_csv(file="data/virus_exp_design.csv", col_names = T)

metadata <- read_delim(file="data/virus_exp_design.csv", delim = ",", col_names = T)
```

```
## # A tibble: 6 × 5
##   sample  virus_type strain treatment    time_point
##   <chr>   <chr>      <chr>  <chr>        <chr>
## 1 sample1 Influenza  A      No Treatment T0
## 2 sample2 Influenza  B      No Treatment T1
## 3 sample3 Influenza  C      No Treatment T2
## 4 sample4 Influenza  A      Vaccine      T0
## 5 sample5 Influenza  B      Vaccine      T1
## 6 sample6 Influenza  C      Vaccine      T2
```

Trabajando con datos

Leeremos los datos que corresponden a casos de una epidemia de ébola simulada (obtenidos de: https://www.epirhandbook.com/). En R, una tabla se llama **data.frame**.

```
surv_raw <- read_tsv("data/ebola_epidemic.tab", col_names = T)
```

	case_id	generation	date_infection	date_onset	date_hospitalisation	date_outcome	outcome	sex	age	age_unit	age_years	age_cat	age_cat5
1	5fe599	4	2014-05-08	2014-05-13	2014-05-15			m	2	years	2	0-4	0-4
2	8689b7	4		2014-05-13	2014-05-14	2014-05-18	Recover	f	3	years	3	0-4	0-4
3	11f8ea	2		2014-05-16	2014-05-18	2014-05-30	Recover	m	56	years	56	50-69	55-59
4	b8812a	3	2014-05-04	2014-05-18	2014-05-20			f	18	years	18	15-19	15-19
5	893f25	3	2014-05-18	2014-05-21	2014-05-22	2014-05-29	Recover	m	3	years	3	0-4	0-4
6	be99c8	3	2014-05-03	2014-05-22	2014-05-23	2014-05-24	Recover	f	16	years	16	15-19	15-19
7	07e3e8	4	2014-05-22	2014-05-27	2014-05-29	2014-06-01	Recover	f	16	years	16	15-19	15-19

Showing 1 to 25 of 25 entries

Limpiar y organizar los datos

Una vez que los datos están importados, lo primero que hacemos es limpiar los datos.

¿Qué significa "limpiar" los datos?

1. Preparación para el análisis y visualización
2. Estandarizar los nombres de las columnas
3. Filtrar filas y columnas
4. Unificar la ortografía
5. Crear variables categóricas y nuevas variables calculadas
6. Unir con otros datos
7. Eliminar duplicados ...

Esto lo haremos con el paquete `dplyr`.

Limpiar y organizar los datos

Funciones principales:

Función	Utilidad
`filter()`	subconjunto de filas
`select()`	subconjunto de columnas
`distinct()`	eliminar duplicados
`clean_names()`	estandarizar nombre de columnas (automático)
`rename()`	cambiar el nombre de las columnas de manera manual
`mutate()`	crear y transformar columnas

Limpiar y organizar los datos

Usaremos el siguiente dataframe más pequeño con el propósito de hacerlo más sencillo:

case_id	age	sex	lab_confirmed	wt (kg)
694928	23	m	FALSE	70
86340d	0	f	TRUE	18
92d002	16	m	TRUE	59
544bd1	10	f	TRUE	39
544bd1	10	f	TRUE	39
544bd1	10	f	FALSE	39

Filtrar filas con `filter()`

```
filter(surv_raw)
```

Primer argumento: el `data.frame`.

case_id	age	sex	lab_confirmed	wt (kg)
694928	23	m	FALSE	70
86340d	0	f	TRUE	18
92d002	16	m	TRUE	59
544bd1	10	f	TRUE	39
544bd1	10	f	TRUE	39
544bd1	10	f	FALSE	39

Filtrar filas con `filter()`

```
filter(surv_raw, age < 18)
```

Segundo (y más) argumento(s): la prueba lógica para filtrar las filas que queremos *mantener*.

case_id	age	sex	lab_confirmed	wt (kg)
86340d	0	f	TRUE	18
92d002	16	m	TRUE	59
544bd1	10	f	TRUE	39
544bd1	10	f	TRUE	39
544bd1	10	f	FALSE	39

Filtrar filas con `filter()`

```
filter(surv_raw, age < 18, sex == "f")
```

Segundo (y más) argumento(s): la prueba lógica para filtrar las filas que queremos *mantener*.

case_id	age	sex	lab_confirmed	wt (kg)
86340d	0	f	TRUE	18
544bd1	10	f	TRUE	39
544bd1	10	f	TRUE	39
544bd1	10	f	FALSE	39

Fíjate que usamos doble igual (==) para comprobar si es igual (equivalente).

Si queremos buscar que no sea igual (diferente), debemos usar la expresión `!=`.

Filtrar filas con `filter()`

```
filter(surv_raw,
  age < 18 &
  (sex == "f" | lab_confirmed == TRUE)
)
```

Los saltos de línea y las indentaciones no tienen impacto (útil para mantener el código más ordenado).

La lógica se puede hacer más compleja utilizando:

- `&` (y)
- `|` (o)
- Paréntesis

case_id	age	sex	lab_confirmed	wt (kg)
86340d	0	f	TRUE	18
92d002	16	m	TRUE	59
544bd1	10	f	TRUE	39
544bd1	10	f	TRUE	39
544bd1	10	f	FALSE	39

Seleccionar columnas con `select()`

```
select(surv_raw, ___)
```

`select()` también espera recibir un `data.frame` en el primer argumento.

case_id	age	sex	lab_confirmed	wt (kg)
694928	23	m	FALSE	70
86340d	0	f	TRUE	18
92d002	16	m	TRUE	59
544bd1	10	f	TRUE	39
544bd1	10	f	TRUE	39
544bd1	10	f	FALSE	39

Seleccionar columnas con `select()`

```
select(surv_raw, case_id, age)
```

Puedes indicar a `select()` los nombres de columna(s) que quieres *mantener*.

case_id	age
694928	23
86340d	0
92d002	16
544bd1	10
544bd1	10
544bd1	10

Seleccionar columnas con `select()`

```
select(surv_raw, case_id, age, sex)
```

Puedes indicar a `select()` los nombres de columna(s) que quieres *mantener*.

case_id	age	sex
694928	23	m
86340d	0	f
92d002	16	m
544bd1	10	f
544bd1	10	f
544bd1	10	f

Seleccionar columnas con `select()`

```
select(surv_raw, -case_id, -lab_confirmed)
```

O qué columnas quieres *eliminar* con el símbolo menos "-".

age	sex	wt (kg)
23	m	70
0	f	18
16	m	59
10	f	39
10	f	39
10	f	39

`filter()` y `select()` simultáneamente

Puedes utilizar el símbolo %>% ("*pipe*") para "pasar" datos de una función a la siguiente.

Una orden típica de limpieza contiene una secuencia de pasos enlazados:

- Renombrar columnas
- Filtrar filas
- Seleccionar columnas
- Eliminar duplicados
- Limpiar valores
- ...

`filter()` y `select()` simultáneamente

Anteriormente, el primer argumento era el dataframe:

```
filter(surv_raw, age < 18)
```

Usando las pipes, podemos escribir lo mismo como:

```
surv_raw %>% filter(age < 18)
```

Puedes pipear los datos a través de *múltiples* funciones:

```
surv_raw
```

case_id	age	sex	lab_confirmed	wt (kg)
694928	23	m	FALSE	70
86340d	0	f	TRUE	18
92d002	16	m	TRUE	59
544bd1	10	f	TRUE	39
544bd1	10	f	TRUE	39
544bd1	10	f	FALSE	39

`filter()` y `select()` simultáneamente

Anteriormente, el primer argumento era el dataframe:

```
filter(surv_raw, age < 18)
```

Usando las pipes, podemos escribir lo mismo como:

```
surv_raw %>% filter(age < 18)
```

Puedes pipear los datos a través de *múltiples* funciones:

```
surv_raw %>% filter(age < 18)
```

case_id	age	sex	lab_confirmed	wt (kg)
86340d	0	f	TRUE	18
92d002	16	m	TRUE	59
544bd1	10	f	TRUE	39
544bd1	10	f	TRUE	39
544bd1	10	f	FALSE	39

`filter()` y `select()` simultáneamente

Anteriormente, el primer argumento era el dataframe:

```
filter(surv_raw, age < 18)
```

Usando las pipes, podemos escribir lo mismo como:

```
surv_raw %>% filter(age < 18)
```

Puedes pipear los datos a través de *múltiples* funciones:

```
surv_raw %>% filter(age < 18) %>% select(case_id, age, sex)
```

case_id	age	sex
86340d	0	f
92d002	16	m
544bd1	10	f
544bd1	10	f
544bd1	10	f

`filter()` y `select()` simultáneamente

Anteriormente, el primer argumento era el dataframe:

```
filter(surv_raw, age < 18)
```

Usando las pipes, podemos escribir lo mismo como:

```
surv_raw %>% filter(age < 18)
```

Puedes pipear los datos a través de *múltiples* funciones:

```
surv_raw %>% filter(age < 18) %>% select(case_id, age, sex) %>% distinct()
```

case_id	age	sex
86340d	0	f
92d002	16	m
544bd1	10	f

Limpiar los nombres de las columnas

Podemos ver los nombres de las columnas del dataframe `surv_raw` con `names()`:

```
surv_raw %>%  # datos
  names()     # mostrar el nombre de las columnas
```

```
## [1] "case_id"       "age"           "sex"           "lab_confirmed"
## [5] "wt (kg)"
```

Aplicar `clean_names()` a `surv_raw` con un *pipe*: esto estandariza automáticamente los nombres de las columnas (minúsculas, sin espacios ni caracteres especiales).

```
library(janitor)

surv_raw %>%             # datos
  clean_names() %>%      # estandarizar nombre de columnas
  names()                # muestra los nuevos nombres
```

```
## [1] "case_id"       "age"           "sex"           "lab_confirmed"
## [5] "wt_kg"
```

Limpiar los nombres de las columnas

Podemos ver los nombres de las columnas del dataframe `surv_raw` con `names()`:

```
surv_raw %>%  # datos
  names()     # mostrar el nombre de las columnas
```

```
## [1] "case_id"       "age"           "sex"           "lab_confirmed"
## [5] "wt (kg)"
```

También podemos aplicar la función `rename()` para cambiar los nombres de las columnas de manera manual.

```
surv_raw %>%                        # datos
  clean_names() %>%                 # estandarizar nombre de columnas
  rename(                           # cambio manual
      age_years  = age)  %>%            # nombre nuevo = nombre antiguo
  names()                           # mostrar el nombre de las columnas
```

```
## [1] "case_id"       "age_years"     "sex"           "lab_confirmed"
## [5] "wt_kg"
```

La función `mutate()` para crear nuevas columnas

La sintaxis es:

```
DATASET %>%
  mutate(NOMBRE_COLUMNA_NUEVA = UNA_FUNCION(argumentos))
```

```
surv_raw %>%
  mutate(age_group = ifelse(
    test = age >= 18,
    yes = "adult",
    no = "minor")
    )
```

case_id	age	sex	lab_confirmed	wt (kg)	age_group
694928	23	m	FALSE	70	adult
86340d	0	f	TRUE	18	minor
92d002	16	m	TRUE	59	minor
544bd1	10	f	TRUE	39	minor
544bd1	10	f	TRUE	39	minor
544bd1	10	f	FALSE	39	minor

`ifelse()` comprueba de manera lógica cada fila y escribe el resultado en la columna `age_group`:

- "adult" si la prueba es TRUE
- "minor" si la prueba es FALSE

La función `mutate()` para editar columnas

La sintaxis es:

```
DATASET %>%
  mutate(MISMO_NOMBRE_COLUMNA = UNA_FUNCION(argumentos))
```

```
surv_raw %>%
  mutate(sex = recode(sex,
    "m" = "male",
    "f" = "female"))
```

La columna `sex` se sobrescribe con los nuevos valores que hemos definido.

- "female" si el valor es `f`
- "male" si el valor es `m`

case_id	age	sex	lab_confirmed	wt (kg)
694928	23	male	FALSE	70
86340d	0	female	TRUE	18
92d002	16	male	TRUE	59
544bd1	10	female	TRUE	39
544bd1	10	female	TRUE	39
544bd1	10	female	FALSE	39

Guardar datos

Para escribir nuestra tabla en un archivo separado por comas (`.csv`), podemos utilizar la función `write_csv()`. Hay dos argumentos obligatorios:

- el nombre de la tabla
- la ruta y el nombre del archivo al que se exportará

Por ejemplo, queremos guardar el dataframe después de haber limpiado los nombres de las columnas:

```
surv_clean <- surv_raw %>%  # guarda en un objeto
  clean_names() %>%
  rename(
      age_years  = age)
```

Por defecto, el delimitador está establecido (ya que es `csv`) y las columnas se separarán con una coma (paquete `utils`, por defecto):

```
write_csv(surv_clean, file="data/surv_clean.csv", col_names = T, quote = "none")
```

Otra función de uso general es `write_delim()`, que permite especificar el delimitador (paquete `tidyr`).

```
write_delim(surv_clean, file="data/surv_clean.tsv", delim = "\t", quote = "none", col_names = T)
```

Guardar datos

Resumen de las funciones generales para guardar archivos:

Tipo de datos	Extensión	Función	Paquete
Valores separados por comas	`.csv`	`write.csv()`	`utils` (por defecto)
		`write_csv()`	`readr`
Valores separados por tabulaciones	`.tsv`, `.tab`	`write.table()`	`utils`
		`write_tsv()`	`readr`
Otros	`.txt`, `.dat`	`write.table()`	`utils`
		`write_delim()`	`readr`
Excel	`.xlsx`, `.xls`	`write_xlsx()`	`writexl`
		`write.xlsx()`	`openxlsx`

03

Estadística descriptiva

Funciones básicas para estadística descriptiva

Continuaremos con `Tidyverse` y otras funciones interesantes que el paquete nos ofrece:

Función	Definición
`count()`	Contar
`group_by()`	Agrupar
`summarize()`	Resumir
`arrange()`	Ordenar

La función `count()`

Pasa los datos con el pipe a `count()` e indica el nombre de la columna.

Esto devuelve el número de filas (observaciones) por cada valor único (por grupo).

```
surv_raw <- read_tsv("data/ebola_epidemic.tab")

surv_raw %>%
  count(hospital)
```

```
## # A tibble: 6 × 2
##   hospital                               n
##   <chr>                              <int>
## 1 Central Hospital                     454
## 2 Military Hospital                    896
## 3 Missing                             1469
## 4 Other                                885
## 5 Port Hospital                       1762
## 6 St. Mark's Maternity Hospital (SMMH)   422
```

La función count()

Puedes incluir múltiples columnas para hacer los grupos:

```
surv_raw %>%
  count(hospital, sex)
```

```
## # A tibble: 18 × 3
##    hospital                sex       n
##    <chr>                   <chr> <int>
##  1 Central Hospital        f       202
##  2 Central Hospital        m       230
##  3 Central Hospital        <NA>     22
##  4 Military Hospital       f       421
##  5 Military Hospital       m       435
##  6 Military Hospital       <NA>     40
##  7 Missing                 f       700
##  8 Missing                 m       696
##  9 Missing                 <NA>     73
## 10 Other                   f       418
## 11 Other                   m       423
## 12 Other                   <NA>     44
## 13 Port Hospital           f       857
## 14 Port Hospital           m       823
```

`group_by()` y `summarize()`

Pero, ¿y si quieres más que solo los recuentos?

- "¿Cuál es la edad media en cada grupo?"
- "¿Cuál es la fecha más reciente de inicio de síntomas en cada grupo?"
- "¿Cuál es el número de defunciones en cada grupo?"

Las funciones `group_by()` y `summarise()` juntas te dan la flexibilidad de crear un nuevo dataframe resumido con estadísticas por grupos.

group_by() y summarize()

La sintaxis es:

```
DATASET %>%
  group_by(COLUMNAS) %>%
  summarise(NUEVA_COLUMNA = UNA_FUNCION())
```

- ¿Cuántos casos hay registrados en cada hospital?

```
surv_raw %>%
  group_by(hospital) %>%
  summarise(n_rows = n())
```

```
## # A tibble: 6 × 2
##   hospital                              n_rows
##   <chr>                                  <int>
## 1 Central Hospital                         454
## 2 Military Hospital                        896
## 3 Missing                                 1469
## 4 Other                                    885
## 5 Port Hospital                           1762
## 6 St. Mark's Maternity Hospital (SMMH)     422
```

group_by() y summarize()

La sintaxis es:

```
DATASET %>%
  group_by(COLUMNAS) %>%
  summarise(NUEVA_COLUMNA = UNA_FUNCION())
```

- ¿Cuál es la edad media en cada grupo?

```
surv_raw %>%
  group_by(hospital) %>%
  summarise(
    n_rows  = n(),
    age_avg = mean(age_years, na.rm = T))
```

```
## # A tibble: 6 × 3
##   hospital                      n_rows age_avg
##   <chr>                          <int>   <dbl>
## 1 Central Hospital                 454    15.7
## 2 Military Hospital                896    15.9
## 3 Missing                         1469    16.0
## 4 Other                            885    16.0
```

group_by() y summarize()

La sintaxis es:

```
DATASET %>%
  group_by(COLUMNAS) %>%
  summarise(NUEVA_COLUMNA = UNA_FUNCION())
```

- ¿Cuál es la fecha más reciente de inicio de síntomas en cada grupo?

```
surv_raw %>%
  group_by(hospital) %>%
  summarise(
    n_rows  = n(),
    age_avg = mean(age_years, na.rm = T),
    max_onset = max(date_onset, na.rm = T))
```

```
## # A tibble: 6 × 4
##   hospital                    n_rows age_avg max_onset
##   <chr>                        <int>   <dbl> <date>
## 1 Central Hospital               454    15.7 2015-04-28
## 2 Military Hospital              896    15.9 2015-04-29
## 3 Missing                       1469    16.0 2015-04-28
```

group_by() y summarize()

La sintaxis es:

```
DATASET %>%
  group_by(COLUMNAS) %>%
  summarise(NUEVA_COLUMNA = UNA_FUNCION())
```

- ¿Cuál es el número de defunciones en cada grupo?

```
surv_raw %>%
  group_by(hospital) %>%
  summarise(
    n_rows  = n(),
    age_avg = mean(age_years, na.rm = T),
    max_onset = max(date_onset, na.rm = T),
    deaths    = sum(outcome == "Death", na.rm = TRUE))
```

```
## # A tibble: 6 × 5
##   hospital                     n_rows age_avg max_onset  deaths
##   <chr>                         <int>   <dbl> <date>      <int>
## 1 Central Hospital                454    15.7 2015-04-28    193
## 2 Military Hospital               896    15.9 2015-04-29    399
```

Ordena con `arrange()`

Ordena la tabla colocando la(s) columna(s) por la(s) que quieres ordenar dentro de `arrange()`:

```
surv_raw %>%
  group_by(hospital) %>%
  summarise(
    n_rows  = n(),
    age_avg = mean(age_years, na.rm = T),
    max_onset = max(date_onset, na.rm=T),
    deaths    = sum(outcome == "Death", na.rm = TRUE)) %>%
  arrange(n_rows)
```

```
## # A tibble: 6 × 5
##   hospital                              n_rows age_avg max_onset  deaths
##   <chr>                                  <int>   <dbl> <date>      <int>
## 1 St. Mark's Maternity Hospital (SMMH)     422    15.6 2015-04-27    199
## 2 Central Hospital                         454    15.7 2015-04-28    193
## 3 Other                                    885    16.0 2015-04-30    395
## 4 Military Hospital                        896    15.9 2015-04-29    399
## 5 Missing                                 1469    16.0 2015-04-28    611
## 6 Port Hospital                           1762    16.3 2015-04-30    785
```

Podemos ordenar en orden descendente con "-" (`arrange(-n_rows)`).

Recursos

- Batra, Neale, et al. The Epidemiologist Rf Handbook. 2021. https://www.epirhandbook.com/en/. Muy recomendable para iniciarse en R con casos aplicados. Algunos ejemplos de este curso se han realizado a partir del material y con ideas de este fabuloso recurso.

- Tutorial general de `Tidyverse`: https://rpubs.com/paraneda/tidyverse

Contacto

Marta Coronado Zamora	David Castellano
marta.coronado@uab.cat	david.castellano@uab.cat
@geneticament.bsky.social	@castellanoed.bsky.social
Universitat Autònoma de Barcelona	University of Arizona