Intruducción a R

Laia Egea e Iago Giné

10-12/12/2019

Abstract

El objetivo de este curso es introducir el software R a todas las personas que trabajen con datos y darles las herramientas para el manejo y visualización de datos, así como para algunos análisis estadísticos básicos.

Contents

Presentación del entorno R y Rstudio

- R y RStudio son dos programas diferentes.
  - R es el programa que calcula. Es software libre y gratuito, inicialmente enfocado a la computación estadística.
  - R también es el lenguaje en el que escribimos los comandos, pues es también el lenguaje de programación.
  - Hoy en día está ampliamente extendido y entre los lenguajes de programación más utilizados. Como consecuencia:
    * existen muchos foros en internet donde los usuarios plantean/resuelven sus dudas y/o propuestas.
    * hay muchas librerías para R en continuo desarrollo y que permiten usarlo de una manera mucho más rápida y eficiente.

  - RStudio es la interfaz donde estaremos trabajando, pues nos ofrece algunas comodidades.
    * Nos permite crear scripts de una manera más ágil y en un entorno mucho más agradable que usando R.
    * También hace más sencillo instalar y desinstalar librerías, cargar y visualizar bases de datos, etc.
    * Ofrece otras posibilidades más allá de R como crear páginas web, pdf's o archivos word con R integrado, que va más allá de este curso.

- Entre las consecuencias de lo ampliamente usado de R está que podemos encontrar en internet muchos tutoriales introductorios en cualquier idioma:
  - http://rosuda.org/mitarbeiter/pilhoefer/rkurs2.pdf
  - http://b2slab.upc.edu/software-and-tutorials/r-nutshell/
  - https://www.cyclismo.org/tutorial/R/index.html
  - https://www.uv.es/vcoll/preliminares.html
  - http://people.math.aau.dk/~sorenh/misc/rdocs/Rintro-notes.pdf
  - https://cran.r-project.org/doc/contrib/Verzani-SimpleR.pdf
  - en la pestaña Resources de la web de RStudio
  - otros

- ... y foros:
  - principalmente https://stackoverflow.com/questions/tagged/r
  - para cuestiones relacionadas con RStudio, pero también con las librerías de la familia tidyverse y otros: https://community.rstudio.com/

- ... y blogs:
  - principalmente: https://www.r-bloggers.com/
  - https://statisticsglobe.com/r-programming-language/
  - enfocados a la estadística, pero no sólo (de hecho, diría que los siguientes son los mejores):
    * http://www.sthda.com/english/
    * http://www.flutterbys.com.au/stats/course.html
    * http://www.r-tutor.com/(http://www.r-tutor.com/sitemap)
  - enfocado a la investigación en psicología:
    * http://personality-project.org/r/(http://personality-project.org/r/)
  - centrado en RStudio: https://support.rstudio.com/hc/en-us
  - https://www.statmethods.net/index.html
  - https://www.datacamp.com/community/tags/r-programming

- ... y sobre librerías o materias específicas:
  - sobre librerías en general:
    * https://rdrr.io/
    * https://www.maths.lancs.ac.uk/~rowlings/R/TaskViews/
  - gráficas en general:
    * https://www.r-graph-gallery.com/
  - formas de mostrar descriptivos:
    * https://dabblingwithdata.wordpress.com/2018/01/02/my-favourite-r-package-for-summarising-data/

- ... e incluso multitud de libros que podéis encontrar en https://bookdown.org/

- ... además de diversos cursos online (MOOC's) en plataformas como Coursera, edX, etc.

- En los enlaces citados se encuentran recursos para utilizar las diferentes librerías, y también para usar R enfocado a las más diversas tareas, desde los cálculos estadísticos más habituales, pero, por ejemplo, también para hacer meta-análisis, análisis factorial, estadística bayesiana o machine learning, etc.

- También tenemos las páginas webs principales en torno a R, que también contienen información útil, pero son más técnicas:

  - R, CRAN
  - las publicaciones del projecto: R News (2001 - 2008) y R Journal (2009 - presente)
  - R Forge
  - https://stat.ethz.ch/R-manual/

- Entonces, este curso, para qué?
- Y por qué R?
  - Porque puedes guardar las instrucciones que ejecutas en R scripts, y ejecutarlas todas de una vez, sin tener que memorizar y repetir cada uno de los pasos.
  - Porque en internet podrás encontrar solución a (casi) todos los problemas que tengas.
  - Porque podrás personalizar/modificar cada instrucción con las opciones que desees.
  - Porque es gratuito, como la versión de código abierto de RStudio, y los puedes instalar donde desees.
  - Porque si hay varias respuestas a una misma pregunta, lo que al principio puede hacerte dudar sobre cuál escoger, probablemente hay una que te funcionará y te irá mejor que las otras.

Nociones elementales y funciones básicas

- R es esencialmente una consola en la que el cursor se sitúa tras el símbolo `>`. Ahí se escriben las instrucciones. Se ejecutan con `Enter`.
- Los resultados suelen aparecer debajo. En el caso de gráficas, depende del entorno en que se trabaje (R, RStudio, R Commander, etc.)

Cómo guardar bien la sintaxis

- Las instrucciones se pueden escribir todas en un fichero de texto con la extensión `.r` (el R-script), el cual puede ser cargado y ejecutado desde R. En RStudio, lo podemos ver y ejecutar al mismo tiempo en un panel situado junto al panel de la consola.
  - Desde el R-script en RStudio se ejecutan con `Control+Enter`
  - Sin embargo, RStudio no es imprescindible: para abrir, editar y guardar un R-script, una aplicación básica como el `Bloc de notas` de Windows es suficiente.
- Se pueden escribir comentarios (secciones de código que el programa no ejecuta), situando antes un símbolo `#`
- R es un lenguaje orientado a objetos. Para asignar nombre a los objetos usamos el símbolo `<-`
- Los missings en R se representan con el símbolo NaN (objetos numéricos) NA (cualquier clase de objetos)
- Los objetos elementales básicos pueden ser de las siquientes clases:
  - Lógicos (`TRUE` y `FALSE`)
  - Numéricos
  - Caracteres

```
# línea de código de R que no hace nada

nombre <- "Luis"
nombre # para ver el contenido de un objeto, basta escribir su nombre
varon <- TRUE
# se pueden introducir diferentes instrucciones en una misma línea separadas por ;
edad <- 23; edad  # indiferentemente de los espacios en blanco en medio
estatura <- 1.77
```

- A partir de los anteriores se pueden crear objetos compuestos con diferentes estructuras, como pueden ser:
  - Vectores
  - Factores
  - Matrices

  - Data frames
  - Listas
- Para saber más, Understanding basic data types in R

Objetos

Vectores

- Todos los elementos del vector han de ser del mismo tipo:
- Se crean y se unen con la función `c(...)`
- vector[i] para acceder al i-ésimo elemento del vector
- Para saber la longitud del vector, es decir, cuantos elementos tiene, usamos la función `length()`

```
nombre <- c("Luis","Maria")
edad <- c(23,24)
varon <- c(TRUE,FALSE)
estatura <- c(1.77, 1.64) # entre los objetos y las comas puede haber espacios

length(edad)
estatura[2]
```

Factores

- Los factores pueden ser de dos tipos al igual que las variables categóricas:
  - Nominales: No ordenados
  - Ordinales: Ordenados
- Se crean a partir de un vector numérico con las funciones:
  - Nominales: `as.factor()`
  - Ordinales: `as.ordered`
- Se crean a partir de un vector de caracteres utilizando `factor()`
- Las etiquetas se asignan con `levels()`

```
f <- as.factor(c(1,2,3,1,2,1,1,3,2)) #Factor Nominal
f
levels(f) <- c("Bajo","Medio","Alto")
ford <- as.ordered(f) #Factor Ordinal
ford
```

Matrices

- Las matrices son una ampliación de los vectores con dos dimensiones: filas y columnas
- Todos los elementos deben ser del mismo tipo
- Se crean con la función `matrix()`
- Para seleccionar un elemento de un matriz: Matriz[i,j]
- Para saber las dimensiones: `dim()`

```
ejema <- matrix(c(1,2,3,4,5,6,7,8,9,10,11,12), ncol=3, byrow=TRUE)
ejema

dim(ejema)

ejema[1,1]
```

Listas

- Una lista es un objeto consistente en una colección ordenada de objetos que se suelen llamar componentes.

- No es necesario que los componentes sean del mismo tipo, ni de la misma longitud: una lista puede estar compuesta de, por ejemplo, un vector numérico de tamaño 2, un valor lógico, un vector de tamaño 3, una matriz y una función.
- Se construyen con la función `list()`
- La selección de elementos se hace con doble corchete.
- El corchete simple se utiliza para seleccionar una sublista.

```
ejemplolista <- list(nombre="Pedro",
                     casado=T,
                     esposa="Maria",
                     no.hijos=3,
                     edad.hijos=c(4,7,9))
ejemplolista

ejemplolista[[5]]

ejemplolista[[5]][2]
```

Data frames

¿Cómo deben estar organizadas las filas/columnas en bases de datos longitudinales para leerlas con R? (I)

- Es la clase de objeto que R asigna a bases de datos. Las filas son individuos o casos y las columnas variables.
- Se crean con la función `data.frame()`
- Cada columna tiene que tener los elementos del mismo tipo y todas deben tener la misma longitud.
- Para seleccionar una variable: `NombreDataFrame$NombreVariable`, o lo que es lo mismo `NombreDataFrame[["NombreVariable"]]`
- Para saber la dimensión: `dim()`
- La clase propiamente dicha de un data frame en R es `data.frame`(el resultado de aplicar la función `class`). Sin embargo, la mayoría de librerías que veremos después trabajan con un data frame expandido llamado `tibble` (en este caso el resultado de la función `class` es un vector de 3 componentes, `tbl_df`, `tbl` y `data.frame`).
  - Un `data.frame` podrá ser transformado en `tibble` por medio de la función `as_tibble`, y recíprocamente tenemos la función `as.data.frame`.

```
BD <- data.frame(nombre = c("Luis","Maria","Jesus"),
                 edad = c(23,24,50),
                 varon = c(TRUE,FALSE, TRUE),
                 estatura = c(1.77,1.64,1.50))
BD

dim(BD)
names(BD)
names(BD)[4] <- "altura"
names(BD)

BD$nombre

BD[1,2]
BD[1,]
BD[,2]
```

- Se pueden conocer los objetos que tenemos cargados en R (los objetos de R con los que podemos trabajar en este momento) con la función `ls()`

- También podemos eliminar objetos cargados

```
ls()
rm(f, ford) # para eliminar objetos del entorno
ls()
```

Funciones básicas

- Clase de un objeto

```
class(ejema)
class(BD)
class(ejemplolista)
```

- Atributos de un objeto

```
attributes(ejema)
attributes(BD)
attr(BD, "class")
attributes(ejemplolista)
```

- Estructura de un objeto; sus características y sus datos

```
str(ejema)
str(BD)
str(ejemplolista)
```

- Longitud de un objeto

```
length(ejema)
length(BD)
length(ejemplolista)
```

- También se puede verificar o modificar la clase de un objeto

```
is.character(4) # podemos comprobar si un objeto es de una determinada clase
is.character(BD$nombre)

as.character(4) # un objeto numérico se puede coercionar a un carácter

as.numeric("a") # al revés, no

as.numeric(FALSE) # un objeto lógico se puede coercionar a uno numérico
as.numeric(TRUE)

strDates <- c("01/05/1965", "08/16/1975")
# damos formato de fecha a objetos de clase carácter
dates <- as.Date(strDates, "%m/%d/%Y")

strDates2 <- c("01-05-1965", "08-16-1975")
# damos formato de fecha a objetos de clase carácter
dates <- as.Date(strDates2, "%m-%d-%Y")


is.na(BD$edad) # podemos ver si un objeto tiene valores missing
```

- Se pueden crear secuencias de números fácilmente

```r
# Enteros consecutivos
1:10

# Generar secuencias de números
seq(from = 0, to = 1, by = 0.1)
seq(from = 5, to = 20, by = 2)
seq(0,10,2)

# Repetir secuencias
rep(x = 1, times = 5)
rep(x = c(1,3,5), times = 2)
rep(c(1,3,5), each = 5)
```

- Crear cadenas de caracteres a partir de diferentes objetos

```r
x <- 4

paste("a", x)
paste("a", x, sep = "_")

paste0("a", x, "b", 3)
paste0("x", c(1:10))
paste0(c("a","b"), c(1,2))
names(BD) <- paste0(names(BD),"_BD")
names(BD)
names(BD) <- c("nombre","edad","varon","estatura")
names(BD)
```

- Condiciones y loops

```r
x <- "R"

if(x==5){
  x <- x+2
} else if (x=="R"){
  cat("El programa se llama",x)
} else{
  cat("Qué es ",x,"?", sep = "")
}

for(i in 1:10){
  print(2*i)
}

x <- c(20:5)

while (length(x) > 5) {
  print(x)
  x <- x[-1]
}
```

- Para aplicar una función a una matriz/data.frame/lista R dispone de las funciones de la familia apply:
  - lapply: Devuelve una lista de resultados al aplicar una función a un vector o lista.
  - sapply: Devuelve un vector de resultados al aplicar una función a un vector o lista.
  - apply: Devuelve el vector o la lista de valores obtenidos aplicando una función a un vector o matriz por grupos.

  - tapply: Devuelve una lista con los resultados obtenidos al aplicar una función a un vector por grupos.
  - mapply: Devuelve un vector resultado de aplicar una función a tuplas de elementos en paralelo.

```
xlist <- list(a = 1:10, logic = c(TRUE,FALSE,FALSE,TRUE), b = c(1,0,0,1)); xlist

lapply(xlist, quantile, probs = 1:3/4)
sapply(xlist, quantile, probs = 1:3/4)

x <- cbind(x1 = 3, x2 = c(4:1, 2:5)) # columnas enlazadas
x; dimnames(x)
dimnames(x)[[1]] <- letters[1:8]; x

apply(x, 2, sum)
apply(x, 1, sum)
mapply(paste, 1:4, 4:1)
```

- A veces conviene guardar el nombre de un objeto en una variable y trabajar con la misma.

```
x <- 4       # asignamos el nombre x al objeto 4
w <- "x"     # guardamos el nombre x en la variable w
x
w
# con la función get recuperamos el objeto asignado al nombre (x)
# guardado en la variable w
get(w)
assign(w,5)  # asignamos el valor 5 al caracter guardado en w (x)
get(w)
```

Directorio

- El entorno de R tiene como referencia un directorio concreto. Esto nos permite no tener que introducir la dirección de la carpeta cuando queremos acceder a algún elemento de esta (por ejemplo, bases de datos)
- Para saber qué directorio es se utiliza la función `getwd()`
- Para cambiar el directorio se utiliza la función `setwd()`. De maneras alternativa:
  - `Session > Set Working Directory > Choose Directory`
  - `Session > Set Working Directory > To Source File Location`

```
getwd() # el directorio donde R está trabajando
# setwd("C:/...") # para cambiarlo
```

- Se puede saber qué archivos hay en el directorio donde R está trabajando con la función `list.files()`

```
list.files()
```

Cómo guardar bien los resultados

- Los objetos de R se pueden guardar en ficheros con la función `save`.
- Los ficheros se guardan por convención con la extensión `.rda` o `.rdata` (aunque a veces también se escribe la `r` e incluso la `d` en mayúscula, por ejemplo `.RData`)
- Los resultados, si son matrices, data frames o tablas, además, también se pueden guardar en ficheros `.csv` o con formatos de excel (veremos después algún ejemplo).
- En cualquier caso, en general pueden ser guardados en un fichero de texto con la función `sink`.
- A no ser que se especifique la dirección del fichero, serán guardados en el directorio donde R está trabajando (`getwd()`)

```r
save(x, ejema, file = "example.rda")
rm(x, ejema)

test_table <- table(BD$edad,BD$varon)

sink("output.log") # se crea el fichero donde se quieren guardar los resultados
# si el fichero ya existe con resultados anteriores que no se quieren borrar,
# se tiene que poner sink("output.log", append = TRUE)
test_table
cor.test(BD$edad,BD$estatura)
sink()
```

- Recíprocamente, para cargar ficheros de datos de R, tenemos las funciones `load` y `attach`. Debido a la complejidad de esta última, sólo explicamos la primera

```r
load(file = "example.rda")
# carga los objetos guardados con el mismo nombre con el que fueron guardados;
# sobreescribe los objetos cargados con el mismo nombre
```

- Antes explicamos que la sintaxis de R se guarda en un archivo de texto con la extensión `.r`, que se puede ejecutar fácilmente desde RStudio, o que se puede cargar también desde R. Otra manera de ejecutarlo es a través de la función `source`. Por ejemplo `source("fichero.r")`.

Ayuda de R

- R dispone de una ayuda muy completa sobre todas las funciones, procedimientos y elementos que configuran el lenguaje
- Se puede acceder a ella con la función `help()` o mediante `?`

```r
help(print)
?cat
?as.POSIXct
```

Librerías

- Multitud de usuarios desarrollan técnicas y las comparten creando librerías de funciones adicionales.
- Para utilizar tales librerías, es necesario descargarlas y cargarlas en cada sesión de R.
- Para descargarlas se utiliza la función `install.packages("NombreLibreria")`.
- Para cargarlas se utiliza la función `library(NombreLibreria)`.

```r
install.packages("magrittr") # para instalar librerías
library(magrittr) # para cargar librerías
(.packages()) # para ver qué librerías tenemos cargadas en este momento
```

- Un ejemplo: recodificar variables con la ayuda de la librería `car`

```r
# cargamos la librería
library(car)

BD$nombre
recode(BD$nombre, "'Maria' = 'Nuria'")

# Para guardar este cambio en nuestra base de datos:
BD$nombre <- recode(BD$nombre, "'Maria' = 'Nuria'")

# También funciona con variables numéricas
```

```r
BD$edad
recode(BD$edad, "50 = 45; 24 = 30")

# se pueden recodificar múltiples valores a uno simultáneamente
recode(c(1:10), "seq(0,10,2) = 0; seq(1,11,2) = 1")
```

Operadores aritméticos y lógicos

```r
x <- 4
2+3
x+5
y <- x-3*7 + 24/2 # los espacios aquí no afectan

y
y^x # potencia

y %% x # residuo

# floor()
# log(), log10(), log2()
# exp()
# sqrt()
# factorial()
round(21.53667674,3)

3 == 4
3 == 3
3 != 5
3 != 3
3 < 3
3 <= 3
3 %in% c(1:5)
!3 %in% c(1:5)
3 %in% c(5:50)
!3 %in% c(5:50)

TRUE & TRUE
TRUE & FALSE
FALSE | FALSE
FALSE | TRUE

nchar(paste(x, 4)) == 3 # el número de caracteres de la cadena "x 4" es 3

length(c(1:3,7,10)) > 6

x <- c(1,3,5,7, NA, 8,5, NA)
sum(is.na(x))
```

Funciones y librerías para interaccionar con otros softwares estadísticos como SPSS o STATA

.csv

```
?read.csv
md_data <- read.csv("MissingData.csv")
str(md_data)
View(md_data)
md_data$Response <- car::recode(md_data$Response,"'Not Available' = NA")
str(md_data)
md_data$Response <- as.numeric(as.character(md_data$Response))
str(md_data)


# Para guardar una matriz o data frame en un fichero csv podemos usar write.csv
write.csv(test_table, file = "foo.csv")
```

Excel

- Hay muchas librerías que nos permiten trabajar con ficheros de excel, entre ellas `readxl` y `openxlsx`

```
library(readxl)
df <- read_excel("example_sheets2.xlsx")
head(df)
str(df)
df <- as.data.frame(df)

library(openxlsx)
df <- read.xlsx("readTest.xlsx",sheet = 1, skipEmptyRows = FALSE); df
df <- read.xlsx("readTest.xlsx",sheet = 3, skipEmptyRows = TRUE); str(df)
df$Date <- convertToDate(df$Date)
str(df)
df2 <- read.xlsx("readTest.xlsx", sheet = 3, skipEmptyRows = TRUE, detectDates = TRUE)
str(df2)

wb <- loadWorkbook("readTest.xlsx")
# útil para trabajar con un fichero de excel y escribir en él
# junto a funciones como writeData y saveWorkbook
df3 <- read.xlsx(wb, sheet = 2, skipEmptyRows = FALSE, colNames = TRUE)
str(df3)
df4 <- read.xlsx("readTest.xlsx", sheet = 2, skipEmptyRows = FALSE, colNames = TRUE)
identical(df3,df4)
```

¿Se puede abrir un fichero en dta (stata) en R y cambiar su versión, por ejemplo de la 15 a la 13?

- La librería haven nos lo permite (entre las versiones 8 y 15).
- Otras librerías que pueden ayudar son: `foreign` y `readstata13`
- Para saber más, Datasets basics
- La librería `haven` también guarda las etiquetas de los objetos como atributos de los mismos, a los que se puede acceder por medio de la función `attributes`. Otras librerías como `labelled`, `sjmisc` o `sjlabelled` también tienen múltiples funciones creadas para trabajar con etiquetas.

```
library(haven)
dta_data <- read_dta("carsdata.dta") # con un archivo de stata 10
dta_data
str(dta_data)

# file from https://digitalcommons.usu.edu/all_datasets/27/
dta_data2 <- foreign::read.dta("Jakus_NCDA.dta") # con un archivo de stata 15
```

```r
dta_data2 <- read_dta("Jakus_NCDA.dta")
str(dta_data2)
write_dta(dta_data2, "Jakus_NCDA2.dta", version = 12)
dta_data2 <- foreign::read.dta("Jakus_NCDA2.dta")

dta_data3 <- read_dta("gss_sample.dta")
dim(dta_data3)
table(is.na(dta_data3$prestg10)) # cuántos valores missing hay?
table(na_tag(dta_data3$prestg10), useNA = "ifany") # cuántos hay de cada categoría?

# Se pueden guardar datos en un fichero dta con la función write_dta
?write_dta
```

También con la librería `haven`:

SPSS - .sav

```r
sav_data <- read_sav("survey.sav")
dim(sav_data)
str(sav_data[,1:10])

# Se pueden guardar datos en un fichero sav con la función write_sav
?write_sav
```

SAS

```r
sas_data <- read_sas(data_file = "nyts2017.sas7bdat", catalog_file = "formats.sas7bcat")
head(sas_data[, 1:5])
names(sas_data)

sapply(sas_data,class)
sapply(sas_data,mode)

# Se pueden guardar datos en un fichero sav con la función write_sas
?write_sas
```

Neuroimagen - .nii

- Para saber más:
  - https://www.alexejgossmann.com/MRI_viz/
  - https://johnmuschelli.com/

```r
library(oro.nifti)
library(neurobase)
t1 = readnii("Template-T1-U8-RALPFH-BR.nii.gz")
# t1 = readnii("Template-T2-U8-RALPFH-BR.nii.gz")
class(t1)
dim(t1)
t1
image(t1, z = 225, plot.type = "single")
# image(t1) # tarda algo de tiempo en ejecutarse
orthographic(t1)
ortho2(t1)
```

Manejo de datos con las librerías dplyr & tidyr

Prólogo: la librería `magrittr` y 3 pipas. Una gramática diferente

- Para saber más, https://www.datacamp.com/community/tutorials/pipe-r-tutorial

```
head(names(sav_data))

sav_data %>% # la pipa más habitual: envía los datos de la izquierda a la función
  names() %>% # que sigue (a la derecha o abajo) y retorna su evaluación
  head() # está incluida en la librería dplyr

sav_data %T>% # como la anterior, pero retorna de nuevo los datos de la izquierda
  View() %>% # sirve para usar los datos en más de una función
  names() %>% # cuando las intermedias no retornan nada, como View, plot, ...
  head()

sav_data %$% # para funciones sin un argumento para data.frame's
  cor.test(age,educ)

sas_data %$%
  table(Q2, Q3)
```

- **Ojo**: en ocasiones no conviene usar la pipa: por ejemplo, es mejor `save(data, file)` que `data %>% save(file)`, ya que la segunda opción da lugar a comportamientos inesperados.
- Merge de data frames

```
library(dplyr)

band_members
band_instruments
band_instruments2

band_members %>% inner_join(band_instruments)
band_members %>% inner_join(band_instruments, by = "name")
band_members %>% left_join(band_instruments)
band_members %>% right_join(band_instruments)
band_members %>% full_join(band_instruments)
band_members %>% full_join(band_instruments2, by = c("name" = "artist"))
?join
```

- Añadir filas/columnas a un data frame

```
BD <- data.frame(nombre = c("Luis","Maria","Jesus"),
                 edad = c(23,24,50),
                 varon = c(TRUE,FALSE, TRUE),
                 estatura = c(1.77,1.64,1.50))

BD2 <- data.frame(nombre = c("Julian", "Laura", "Cristina"),
                  varon = c(TRUE,TRUE, FALSE),
                  edad = c(20, 65, 41),
                  estatura = c(1.6, 1.75, 1.72),
                  peso = c(71,67,65))

bind_rows(BD, BD2)
```

```r
mtcars %>%
  str()

one <- mtcars[1:4, ]; one
two <- mtcars[11:14, ]; two
bind_cols(one, two)
```

Selección y filtro de variables

```r
mtcars %>%
  select(drat:qsec)

mtcars %>%
  select(-drat:-qsec)

mtcars %>%
  select(-c(2,8:9))

iris %>%
  str()

select(iris, starts_with("Petal"))

iris %>% select(ends_with("Width")) %>% head()

iris %>%
  select(-contains("Length")) %>%
  head()

iris %>%
  pull(Species) %>% # pull extrae una variable del data frame como vector
  table()

iris %>%
  pull(Species) %>%
  class()

iris %>%
  filter(Species == "setosa") # el outpup de filter es
# el subdata.frame que cumple la condición dada

# ordenar observaciones por una o más variables
iris %>% arrange(Sepal.Length)
iris %>% arrange(Sepal.Length, Petal.Length)

# filtrar casos únicos
dta_data
dta_data %>% distinct(cars)
dta_data %>% distinct(cars, .keep_all = TRUE)
dta_data %>% distinct(cars, hhsize)
```

Transformación de variables

```
iris %>%
  mutate(
  SL2 = Sepal.Length * 2,
  SL44 = SL2 * 2
  ) %>%
  head()

iris %>%
  mutate(PW05 = Petal.Width / 2) %>%
  head()
iris %>%
  transmute(PW05 = Petal.Width / 2) %>%
  head()


?mutate_all

sas_data %>%
  select(Q1:Q4C) %>%
  mutate_all(zap_empty) %>% # transforma todas las celdas "" en NA's
  mutate(QAN = paste(Q1,Q3)) %>%
  head()

iris %>% head()
iris %>%
  mutate_at(vars(matches("Sepal")), log) %>%
  head()

x <- 1:50
case_when(
  x %% 35 == 0 ~ "fizz buzz",
  x %% 5 == 0 ~ "fizz",
  x %% 7 == 0 ~ "buzz",
  TRUE ~ as.character(x)
)

iris %>%
  mutate(size = case_when( # case_when permite multiples outputs según las condiciones
    Sepal.Length < 5.0 ~ "small", # sin tener que recurrir a múltiples if-else
    TRUE ~ "big"                  # y, a diferencia de una función de recode,
  )) %>%                          # permite condiciones complejas
  head()
```

- Datos agrupados

```
mtcars
mtcars %>%
  group_by(cyl) %>%
  summarise(disp = mean(disp),
            hp = mean(hp), # medias por grupo,
            n = n(),  # casos en cada grupo
            distinct_gear = n_distinct(gear)) # y observaciones diferentes en cada grupo
```

```r
md_data %>%
  group_by(Age) %>%
  summarise(rmaxR = max(Response),
            maxR = max(Response, na.rm = TRUE),
            fR = sum(Response, na.rm = TRUE)) # sumario por grupo

md_data %>%
  group_by(Age) %>%
  mutate(rmaxR = max(Response),
         maxR = max(Response, na.rm = TRUE),
         fR = sum(Response, na.rm = TRUE)) %>%
  ungroup() %>%
  View() #estadísticos por grupo añadidos al data frame original
```

- Pivotaje

¿Cómo deben estar organizadas las filas/columnas en bases de datos longitudinales para leerlas con R? (II)

- Podemos tener bases de datos con una fila para cada individuo y tantas columnas como observaciones por variable;
- O bien, podemos tener bases de datos con tantas filas como observaciones se hayan hecho a todos los individuos y una columna por variable;
- Y podemos cambiar de una a otra pivotando:

```r
library(tidyr)

dta_data
dta_data %>%
  group_by(cars) %>%
  mutate(wave=row_number())

# si no tenemos una variable que indique la ola
# y suponemos que los datos empiezan desde la primera hasta que se acaban, la creamos
dta_data %>%
  group_by(cars) %>%
  mutate(wave=row_number()) %>%    # ungroup() %>%
  pivot_wider(names_from = wave, values_from = "hhsize", names_prefix = "hhsize_wave")
```

```r
data("anscombe")
anscombe
anscombe %>%
  tibble::rownames_to_column('id')

# 1) pasar de forma ancha a larga las x's

anscombe %>%
  pivot_longer(cols = c(starts_with("x")),
               names_to = "x_cases",
               values_to = "x") %T>%
  View() %>%
  str()

# 1.1) además, renombrar la columna resultante
```

```r
anscombe %>%
  pivot_longer(cols = c(starts_with("x")),
               names_to = "x_cases",
               names_prefix = "x",
               values_to = "x") %T>%
  View() %>%
  str()

# 2) pasar de forma ancha a larga las x's y las y's

anscombe2 <- anscombe %>%
  tibble::rownames_to_column('id') %>%
  pivot_longer(cols = c(starts_with("x"),starts_with("y")),
               names_to = c(".value", "id_cases"),
               names_pattern = "([a-z]+)([0-9]+)")
anscombe2 %T>%
  View() %>%
  str()

anscombe2

?pivot_wider

anscombe2 %>%
  pivot_wider(names_from = "id_cases", values_from = c("x","y"), names_sep = "")

anscombe3 <- anscombe2 %>%
  pivot_wider(names_from = "id_cases", values_from = c("x","y"), names_sep = "") %>%
  select(x1:y4) %>%
  as.data.frame()

identical(anscombe,anscombe3)
```

- Un problema:
  - Dataset: anscombe
  - Imaginamos que `x1-x4` son 4 observaciones de la variable `x`, e `y1-y4` son 4 observaciones de la variable `y`.
  - Queremos saber, para cada individuo, cuántas veces la observación de `y` es menor que la observación de `x` y mayor o igual que la última observación de `x` (`x4`).
  - Una solución usando los pivotajes anteriores y una solución usando la librería `purrr`

```r
# solucion 1

anscombe %>%
  tibble::rownames_to_column('id') %>% # asignamos un número a cada individuo
  pivot_longer(cols = c(starts_with("x"),starts_with("y")), # pivotamos creando 2 columnas
               names_to = c(".value", "id_cases"), # con las observaciones de x e y
               names_pattern = "([a-z]+)([0-9]+)") %>%
  group_by(id) %>% # agrupamos por individuo: luego calculamos por grupo (individuo)
  mutate(his_y = sum(y < x & y >= x[id_cases == 4])) %>% #cuántas obs cumplen la condición
  ungroup() %>% # luego se transforma al formato original
  pivot_wider(names_from = "id_cases", values_from = c("x","y"), names_sep = "") %>%
  as.data.frame()
```

```r
# solución 2

library(purrr)
?map_dfc

# map_dfc crea un data.frame de 4 columnas (1:4) y tantas filas como el original, anscombe
# el valor en la columna i es TRUE o FALSE según se cumple o no la condición
# yi < xi & yi >= x4
# con rowSums, sumamos para cada fila del data frame creado
# cuántas veces se cumple la condición

anscombe %>%
  mutate(hist_y = rowSums(map_dfc(
    1:4,
    ~ get(paste0("y",.)) < get(paste0("x",.)) & get(paste0("y",.)) >= x4
  ), na.rm = T))
```

- Sampleo: conservar al azar un determinado porcentaje de la muestra

```r
?sample # tanto la función de R sample
# como las funciones de dplyr sample_n y sample_frac son interesantes


iris %>%
  group_by(Species) %>%
  sample_n(3) # escogemos 3 observaciones al azar para cada especie
```

Estadísticos descriptivos

Datos que usaremos:

```r
head(iris)
?iris

library(dplyr)
?starwars
starwars %>%
  names()
starwars %>%
  View()
```

- Listado de estadísticos descriptivos más habituales:

```r
mean(iris$Sepal.Length) # Media
sd(iris$Sepal.Length) # Desviación típica
var(iris$Sepal.Length) # Varianza
median(iris$Sepal.Length) # Mediana
min(iris$Sepal.Length) # Mínimo
max(iris$Sepal.Length) # Máximo
range(iris$Sepal.Length) # Rango
quantile(iris$Sepal.Length, probs = c(0.025, 0.975)) # Cuartiles
```

- Con la función `summary()` obtenemos un resumen descriptivo de una variable o bien de todas las variables de un data.frame:

```
summary(iris$Sepal.Length)
summary(iris)
```

- Tablas de frecuencias absolutas y relativas para variables categóricas:

```
table(iris$Species)
prop.table(table(iris$Species))
```

- Cuando la base de datos tiene missings:

```
mean(starwars$height)
sd(starwars$height)

dim(starwars)
table(starwars$gender)
```

- Para arreglarlo hay que añadir `na.rm = TRUE` para funciones de estadística descriptiva o `useNA = 'always'` o `useNA = 'ifany'`.

```
mean(starwars$height, na.rm = TRUE)
sd(starwars$height, na.rm = TRUE)

table(starwars$gender, useNA = 'always')
table(starwars$gender, useNA = 'ifany')
```

```
table(starwars$hair_color, starwars$eye_color)
```

Gráficos con la librería ggplot2

¿Cómo hacer gráficos (los habituales)? y, ¿cómo guardarlos?

- `ggplot2` es una librería que permite hacer gran variedad de gráficos "bonitos" de manera sencilla.
- Todos los gráficos de `ggplot2` se podrían hacer también con R base pero es más complejo. Por otro lado, ggplot2 es un poco más robusto que R base, es decir, no te permite modificar tantas cosas dado que ya viene predeterminado.
- Las funciones que se utilizan:
  - `ggplot()`: Crea un gráfico nuevo
  - `aes()`: Especifica cómo y qué variables intervendrán en todo el gráfico
  - `+`: Añade capas
- Otras librerías amplían las posibilidades de `ggplot`: `GGally`, `ggfortify`, `ggpmisc`, `ggstance`, `ggpubr`, `ggrepel`,...
- Para saber más, http://www.sthda.com/english/wiki/ggplot2-essentials.

Gráfico de puntos

```
library(ggplot2)

SW <- starwars %>%
  select(height:species) %>%
  mutate(hair_color = as.factor(hair_color),
         skin_color = as.factor(skin_color),
         eye_color = as.factor(eye_color),
         gender = as.factor(gender),
         homeworld = as.factor(homeworld),
         species = as.factor(species))
```

```r
ggplot(SW, aes(x = mass, y = height)) +
  geom_point()

ggplot(SW, aes(x = mass, y = height, colour = "red")) +
  geom_point() +
  annotate("text",
           label = paste("cor:", round(cor(SW$height,SW$mass, use = "complete.obs"),3)),
           x = 1000, y = 250)

ggplot(SW, aes(x = mass, y = height, colour = gender)) +
  geom_point()

ggplot(SW, aes(x = mass, y = height, colour = gender, size=birth_year, alpha = 0.5)) +
  geom_point(shape=23)

ggplot(SW,
       aes(x = mass, y = height,
           colour = gender, fill = gender, size = birth_year, alpha = 0.5)) +
  geom_point(shape=23)

ggplot(BD, aes(x=nombre,y=edad)) +
  geom_point() +
  theme(axis.text.x = element_text(angle = 10))
```

Gráfico de histogramas

```r
ggplot(dta_data2, aes(x=Age)) + geom_histogram()

ggplot(dta_data2, aes(x=Age)) + geom_histogram(color = "green", fill = "blue", bins = 50)

ggplot(dta_data2, aes(x=Age)) +
  geom_histogram(color = "green", fill = "blue", bins = 50) +
  ylab("Frecuencia") +
  ggtitle("Histograma de edad")
```

Boxplot

```r
SW %>%
  ggplot(aes(y=mass)) + geom_boxplot()


SW %>%
  ggplot(aes(x = gender, y=mass)) + geom_boxplot()

SW %>%
  ggplot(aes(x = gender, y=mass)) + geom_boxplot() + coord_flip()
```

Guardar gráficos

```r
# Para guardar el último gráfico generado
ggsave(filename = "graf.tiff", dpi = "print")
```

```r
#para más información sobre cómo especificar el tamaño
# o qué otros formatos de imagen admite:
?ggsave

# Si previamente se ha guardado un gráfico mediante
# p <- ggplot(...)
# también se puede especificar
# ggsave(filename = "graf.tiff", plot = p, dpi = "print")
```

Tests estadísticos y modelos de regresión habituales para el análisis estadístico

Tests de hipótesis

Prueba t de Student

- Prueba paramétrica que compara medias de dos variables (supone que son normales).

```r
summary(iris)

?t.test
t.test(iris$Sepal.Length, iris$Sepal.Width, alternative = "two.sided")
t.test(iris$Sepal.Length, iris$Sepal.Width, alternative = "less")
t.test(iris$Sepal.Length, iris$Sepal.Width, alternative = "greater")

t.test(iris$Sepal.Length, iris$Sepal.Width, alternative = "two.sided", mu = 2.8)
```

Prueba Wilcoxon-Mann-Whitney

- Prueba no paramétrica que compara medias de dos variables

```r
wilcox.test(x = iris$Sepal.Length, y = iris$Sepal.Width,
            alternative = "two.sided", mu = 0,
            paired = FALSE, conf.int = 0.95)
```

Prueba de Shapiro-Wilk

- Comprueba la normalidad de una variable

```r
shapiro.test(starwars$height)
```

Prueba chi-cuadrado de Pearson

- Prueba no paramétrica que se utiliza para probar la independencia de dos variables entre sí mediante la presentación de los datos en tablas de contingencia.
- Hipótesis nula = independencia vs Hipotesis alternativa = dependencia

```r
tt <- table(starwars$gender, starwars$hair_color)
chisq.test(tt)
tt
tt[-2,]
chisq.test(tt[-2,])
```

ANOVA

- Constituye la herramienta básica para el estudio del efecto de uno o más factores (cada uno con dos o más niveles) sobre la media de una variable continua.

```
library(ggplot2)
ggplot(data = starwars, aes(x = gender, y = height, color= gender)) +
  geom_boxplot()

anova <- aov(iris$Petal.Length ~ iris$Species)
summary(anova)
```

Regresiones

Regresión Lineal simple

- $y = a + bX$

```
pairs(SW)

# Se puede crear una gráfica similar creada a partir de ggplot
# gracias a la función ggpairs de la librería GGally

ggplot(SW, aes(x = mass, y = height)) +
  geom_point(shape = 1)+
  scale_x_continuous(breaks = seq(0, 1400,200))+theme_bw()

mod0 <- lm(SW$height ~ SW$mass, data = iris)
summary(mod0)

mode(mod0)
mod0$coefficients

# Intervalo de confianza de los coeficientes de la regressión
confint(object = mod0, level = 0.95 )

plot(mod0) # Diagnóstico del modelo:

# La librería ggfortify nos pemite crear gráficos equivalentes de diagnóstico del modelo
# a partir de ggplot, tanto para lm como para glm

library(ggfortify)
autoplot(mod0, which = 1:6, ncol = 2, label.size = 3) # Diagnóstico del modelo:
```

```
starwars$name[starwars$mass == 1358]

SW <- SW[-which(starwars$mass == 1358),]
ggplot(SW, aes(x = mass, y = height)) +
  geom_point(shape = 1)+
  scale_x_continuous(breaks = seq(20, 160,20)) +
  theme_bw()

mod0_1 <- lm(SW$height ~ SW$mass, data = iris)
summary(mod0_1)

mod0_1$coefficients
```

```r
# Intervalo de confianza de los coeficientes de la regressión
confint(object = mod0_1, level = 0.95 )

plot(mod0_1) # Diagnóstico del modelo:
```

- height $= 103.5133 + 0.9327 \cdot$ mass
- Podemos graficarlo:

```r
library(ggpmisc)
ggplot(SW, aes(x = mass, y = height)) +
  geom_point(shape = 1)+
  geom_smooth(method = 'lm')+
  # scale_x_continuous(breaks = seq(20, 160,20))+
  # theme_bw()+
  stat_poly_eq(formula = y~x, aes(label = ..eq.label..), parse = TRUE, coef.digits = 4)
```

- Calcular predicciones:

```r
# Creamos un data.frame
# con los valores de la variable independiente de los que queremos hacer la predicción:
nuevas <- data.frame(mass = c(80, 60, 20))

# Calculamos las predicciones:
predict(mod0_1, nuevas)
```

Regresión Lineal múltiple

```r
mod1 <- lm(height ~ mass + gender, data = SW)
summary(mod1)
confint(object = mod1, level = 0.95 )
plot(mod1)

# Predicciones:
nuevas <- data.frame(mass = c(80, 60, 20), gender = c('male', 'female', 'female'))
predict(mod1, nuevas)
```

Regresión Logística

```r
SW$gender <- car::recode(SW$gender, "'none' = NA")
SW$gender <- factor(SW$gender)

mod2 <- glm(gender ~ height + mass + hair_color + eye_color + birth_year,
            data = SW,
            family="binomial")
summary(mod2)
plot(mod2)

exp(mod2$coefficients)
```

Apéndice

- **Ojo**:
  - Hay una cantidad enorme de librerías, muchas de las cuales comparten funciones (porque unas llaman a otras), pero también muchas de las cuales comparten nombres de funciones que son

distintas (como muestra la advertencia al cargar la librería con funciones que ya están en el entorno).
  - Por ejemplo, las librerías `plyr` -que no veremos- y `dplyr` -la veremos abajo- comparten varias funciones, entre ellas `rename`. Si se tienen ambas librerías cargadas y R entiende que `rename` se refiere a la función de `plyr`, pero la usamos como la función de `dplyr`, es probable que se produzca un error.
  - Es de los más frecuentes y extraños, por ello hay varias maneras de solucionarlo.

```
lapply(
  paste('package:',names(sessionInfo()$otherPkgs),sep=""),
  detach,
  character.only=TRUE, unload=TRUE)

library(dplyr)
library(plyr)
?rename
# con la ayuda podemos ver que esta función está en varias librerías actualmente cargadas

names(mtcars)
names(rename(mtcars, x = gear)) # error
# especificamos que la función que queremos es la de dplyr
names(dplyr::rename(mtcars, x = gear))
# comprobamos antes estaba usando la función de plyr, ya que genera el mismo error
names(plyr::rename(mtcars, x = gear))
# otra solución es asignar al nombre la función de dplyr
rename <- dplyr::rename
names(rename(mtcars, x = gear)) # ya no hay error


detach("package:plyr", unload=TRUE)
detach("package:dplyr", unload=TRUE)
```