MÓDULO 1: HERRAMIENTAS BIG DATA

TEMA 4.1. HERRAMIENTAS DE ANALISIS: PROGRAMACIÓN EN R

Ferran Carrascosa Mallafrè

Licenciado en Matemáticas por la Universidad de Barcelona. Data Scientist

**ÍNDICE**

[PROGRAMACIÓN EN R 1](#__RefHeading___Toc73144_40828882)

[INTRODUCCIÓN 1](#__RefHeading___Toc73146_40828882)

[ACTIVIDAD GUIADA 1 2](#__RefHeading___Toc73148_40828882)

[ELEMENTOS BÁSICOS DE R 2](#__RefHeading___Toc73150_40828882)

[GRÁFICOS CON GGPLOT2 7](#__RefHeading___Toc73152_40828882)

[COLECCIONES DE OBJETOS 11](#__RefHeading___Toc73154_40828882)

[DEFINCIÓN DE LOS OBJETOS 11](#__RefHeading___Toc73156_40828882)

[FILTROS 17](#__RefHeading___Toc73158_40828882)

[MODIFICACION DE VALORES 20](#__RefHeading___Toc73160_40828882)

[CONTROL DE FLUJO 23](#__RefHeading___Toc73162_40828882)

[FUNCIONES 23](#__RefHeading___Toc73164_40828882)

[CONDICIONALES 24](#__RefHeading___Toc73166_40828882)

[BUCLES 25](#__RefHeading___Toc73168_40828882)

[GESTIÓN DE DATOS 25](#__RefHeading___Toc73170_40828882)

[IMPORTAR Y EXPORTAR DATOS 26](#__RefHeading___Toc73172_40828882)

[CRUCE ENTRE TABLAS 27](#__RefHeading___Toc73174_40828882)

[RESUMENES DE AGREGADOS 31](#__RefHeading___Toc73176_40828882)

[LUBRIDATE: GESTIÓN DE FECHAS Y HORARIOS 39](#__RefHeading___Toc73178_40828882)

[ANEXO: README DE R 40](#__RefHeading___Toc73180_40828882)

[CLONAR EL PROYETO GITHUB EN LOCAL 40](#__RefHeading___Toc73182_40828882)

[PREPARACIÓN ENTORNO RSTUDIO 41](#__RefHeading___Toc73184_40828882)

# PROGRAMACIÓN EN R

## INTRODUCCIÓN

El *Data science* permite transformar datos en conocimiento. El objetivo es aprender a utilizar las mejores herramientas disponibles en R para el *data science*.

## ACTIVIDAD GUIADA 1

La actividad guiada, que resolveremos a lo largo de este temario consiste en comprender la evolución de la riqueza y la esperanza de vida de las personas a nivel de país y continente.

Para ello, contamos con los datos de [países](https://www.rdocumentation.org/packages/datos/versions/0.3.0/topics/paises) coleccionados por [Gapminder](https://www.gapminder.org/) y traducidos al Castellano en el package [datos](https://cran.r-project.org/web/packages/datos/index.html).

library(datos)  
paises07 <- as.data.frame(paises %>% filter(anio==2007))

paises07 %>% head(10)



Fuente: <http://www.gapminder.org/data/>

Los datos muestran la esperanza de vida media en 142 países, población total y su nivel de riqueza medido con el [Producto Interior Bruto (PIB)](https://es.wikipedia.org/wiki/Producto_interno_bruto) per cápita.

<recuerda>El PIB per cápita se puede interpretar como el Salario medio que recibe cada persona del país.</recuerda>

Para el análisis, el continente se recodificará incorporando Oceanía con Asia.

Para realizar el informe, nos apoyaremos en el package Rmarkdown.

## ELEMENTOS BÁSICOS DE R

Aunque se presuponen unos conocimientos iniciales de R, a continuación se hace un repaso de R y RStudio.

#### INTERFAZ DE USUARIO

Si bien R funciona perfectamente des de consola de texto (R-GUI en windows o bien, ejecutando comando R desde xterm en Linux) en el curso se utilizará la interfaz gráfica de RStudio. Ejecuta la siguiente línea.

6 / (4 - 1)

## [1] 2

<sabías que>Puedes recuperar las líneas ejecutadas previamente apretando la tecla Arriba del cursor.</sabías que>

##### RSTUDIO Y FICHEROS \*.R

El lenguaje R se puede escribir mediante editor de texto en ficheros con extensión \*.R. El entorno RStudio (ver imagen), incorpora, además de un editor y de la consola de R, un conjunto de utilidades que facilitan las tareas de programación y análisis de datos: navegador de ficheros, ayuda, objetos en memoria, visor de gráficos…

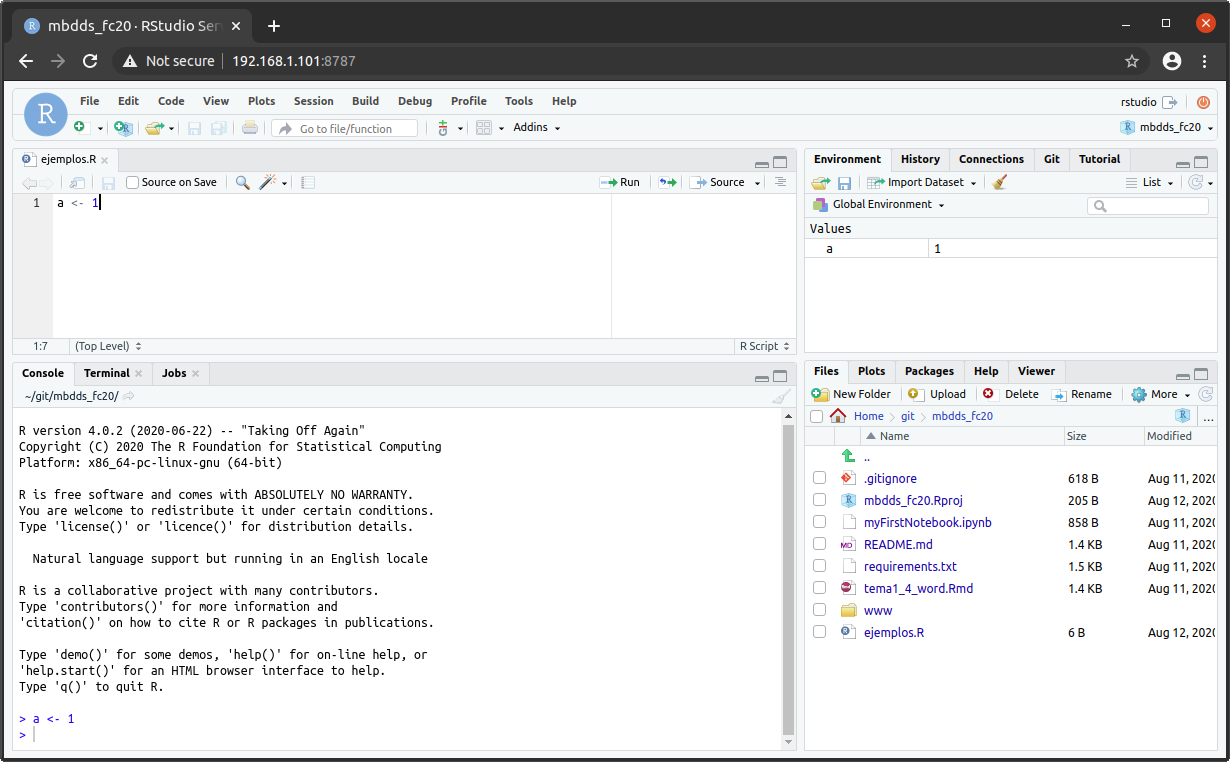


Imagen: Entorno RStudio (Fuente: Elaboración propia).

##### RSTUDIO Y DOCUMENTOS RMARKDOWN (\*.Rmd)

Puedes generar de forma dinámica documentos PDF, Word (como éste que estás leyendo) o HTML mediante ficheros de tipo Rmarkdown (\*.Rmd). Este formato, permite combinar texto con formato, código R y resultados (tablas y gráfcos) ejecutados por R.

<sabías que>Puedes acceder a versión RMarkdown de este documento en [Github.](https://github.com/griu/mbdds_fc20/blob/master/modulo1_tema4_R.Rmd)</sabías que>

##### ACTIVIDAD GUIADA 1.1.

Genera tu primer documento RMarkdown. En RStudio, accede al menú File > New File > R Markdown > Documentos > html y edita Titley Author del formulario de la imagen y el formato de salida HTML > aprieta OK.

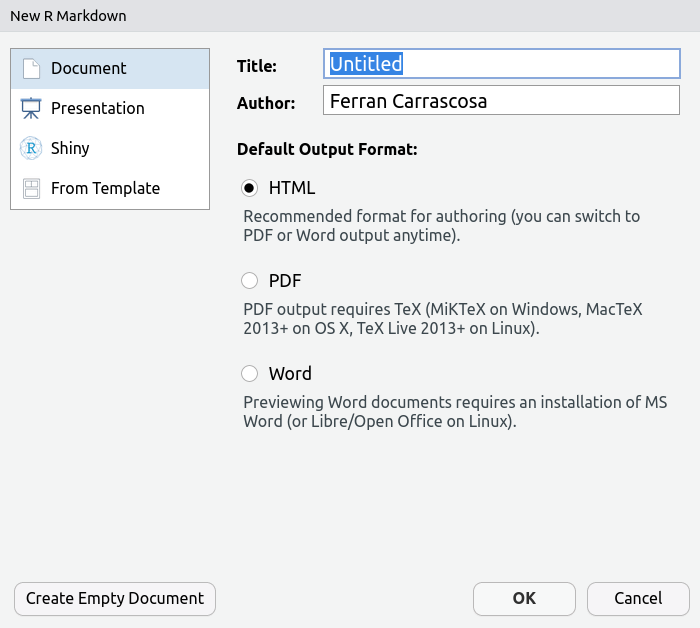


Imagen: Menú de nuevo RMarkdown (Fuente: Elaboración propia).

Guarda el documento generado automáticamente con el nombre 11\_actividad.Rmd. Ahora genera el documento apretando el botón **Knit**.

Comparte con tus compañeros/as, en el foro del curso, tu experiencia, dudas, dificultades con algún otro formato de salida…

#### OBJETOS EN R

##### ASIGNAR OBJETOS

Puedes asignar datos a objetos R. Estos objetos se guardan en memoria.

a <- 0  
a

## [1] 0

b <- a + 1  
b

## [1] 1

<recuerda>Se puede utilizar = , pero, en R se usa <- para diferenciarlo de los valores de los parámetros de las funciones.</recuerda>

Consulta la [guía de estilo de Tydiverse](https://style.tidyverse.org/syntax.html" \l "object-names) para crear nombres.

##### LISTAR OBJETOS

Los objetos aparecen en la pestaña de *environment* (ventana superior derecha).

También puedes acceder al listado con la función ls().

ls()

## [1] "a" "b" "paises07"

<piensa un minuto>¿Por qué cars y pressure no aparecen en el environment? El motivo es porque no están en el environment global, están dentro de la librería (o package) datasets.</ piensa un minuto>

##### GUARDAR OBJETOS

Los objetos R, se pueden guardar en ficheros .RData con save().

save(a, b, file="www/ejemploSave.RData")  
# save.image(file="www/ejemploSave.RData") # guarda todos lo objetos en memoria

##### BORRAR OBJETOS

Se pueden borrar de la memoria con rm():

rm(a,b)  
rm(list=ls()) # borra todo el environment.  
ls()

## character(0)

##### CARGAR OBJETOS

Para cargar en memoria de nuevo los objetos, se utiliza load():

load("www/ejemploSave.RData")  
ls()

## [1] "a" "b"

#### FUNCIONES

Las funciones se pueden separar en los siguientes componentes.

##### NOMBRE DE LA FUNCIÓN

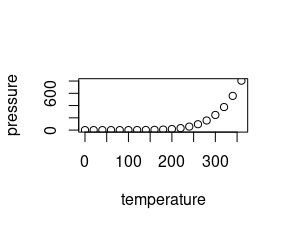
Crea una función llamando a function() seguido de llaves, {}:

describir <- function() {  
 plot(pressure)  
}

<recuerda>El código (o **cuerpo**) de una función se escribe en formato *indented* (o sangría), añadiendo dos espacios a inicio de la fila dentro de las llaves.</recuerda>

Llama la función seguido de paréntesis.

describir()



Resultado función describir

Visualiza su código llamando la función sin paréntesis.

describir

## function() {  
## plot(pressure)  
## }

<importante>Sigue las mismas pautas de estilo que en los nombres de los objetos. Además, se recomienda utilizar verbos:> </importante>

# Bien  
resumir()  
  
# Mal  
resumen()

##### PARÁMETROS DE ENTRADA

Se definen, entre paréntesis, después del nombre de la función.

mi\_funcion <- function(parametro1, paremtro2) {}

Antes, en la función describir, se ha utilizado pressure sin ser un parámetro de entrada. La forma correcta sería:

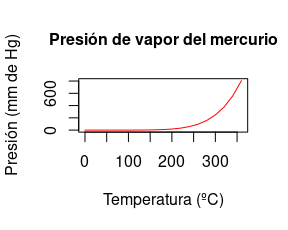
describir <- function(data, type, col = 'red', ...) {  
 plot(data, type = type, col = col, ...)  
}

Se han añadido tres parámetros utilizados por la función plot:

* type: ‘p’ indica puntos, ‘l’ líneas. Ver ?plot.
* col: ‘red’ cambia la línea a rojo. Ver ?colors
* ...: Permite al usuario de la función añadir parámetros nuevos.

Para llamar a la función:

describir(pressure, type='l', main = "Presión de vapor del mercurio"  
 , xlab = "Temperatura (ºC)", ylab = "Presión (mm de Hg)", cex.main=1)



Describir con Parámetros explícitos

Internamente plot toma los nuevos parámetros:

* main: Define el título **principal** del plot.
* xlab: Define el título del **eje x**.
* ylab: Define el título del **eje y**.
* cex.main: modifica el tamaño de letra del título principal.

##### SALIDA DE LA FUNCIÓN

La función devuelve objetos R cuando se llama a return()

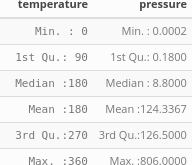
describir <- function(data, type, col = 'red', ...) {  
 plot(data, type = type, col = col, ...)  
 return(summary(data))  
}

En el siguiente ejemplo se asigna la salida de describir() al objeto resumen:

resumen <- describir(pressure, type='l', main = "Presión de vapor del mercurio"  
 , xlab = "Temperatura (ºC)", ylab = "Presión (mm de Hg)", cex.main=1)

Posteriormente se puede visualizar el resumen:

resumen



##### ACTIVIDAD GUIADA 1.2.

Crea un resumen HTML de la esperanza\_de\_vida y el pib\_per\_capita en 2007 con datos de países del package datos.

Para ello abre el fichero `12\_actividad.Rmd y completa los apartados solicitados entre < … > .

Genera el informe HTML con el botón **Knit**.

#### SCRIPTS

Para crear un nuevo script de R puedes ir al menú: File > New File > R Script. Por ejemplo, escribe:

print("Hello World!")

Ejecuta la línea situando el cursor encima y clica el botón **Run** (o Ctrl + Enter).

Guardar tu código: File > Save as > nombre del fichero.R (holaMundo.R).

Ejecuta tu código con source().

source("holaMundo.R")

## [1] "Hello World!"

<recuerda>Si no utilizas print(), al realizar source(), no va a devolver ningún valor por pantalla.</recuerda>

#### PAQUETES

Se publican en [CRAN](https://cran.r-project.org/), un repositorio centralizado. Actualmente contiene más de 16 mil paquetes.

Los más importantes, como stats, graphics, methods, base... están ya disponibles en la consola.

Otros, por ejemplo foreign (leer datos de fuentes externas), vienen instalados, pero es necesario cargarlos con library(foreign) al abrir la sesión.

Otros, la mayoría, es necesario instalarlos antes con install.packages("nombre\_paquete").

## GRÁFICOS CON GGPLOT2

<cita>Un simple gráfico ha brindado más información a la mente del analista de datos que cualquier otro dispositivo. - John Tukey</cita>

#### ESTRUCTURA BÁSICA DE GGPLOT2

ggplot2 es un sistema coherente para hacer gráficos.

library(ggplot2)  
ggplot(data = pressure) +   
 geom\_point(mapping = aes(x = temperature, y = pressure))

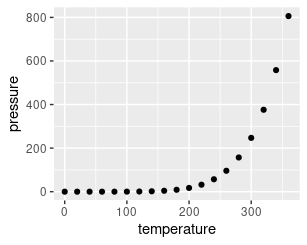


grafico de puntos

Los componentes mínimos son:

* ggplot define el conjunto de datos en el parámetro data que debería contener todas las variables del gráfico.
* geom\_<función>: Define el tipo de gráfico. Por ejemplo, *geom\_point* es un gráfico de puntos.
* mapping: define mapeo de parámetros estéticos (x, y…) a través de la función aes().

A diferencia de plot, ggplot permite asignar el gráfico a un objeto R.

p1 <- ggplot(data = pressure, mapping = aes(x = temperature, y = pressure)) +  
 geom\_point()

Posteriormente, puedes añadirle otros elementos gráficos.

p1 <- p1 + geom\_line()

<importante>Si defines el mapeo de estéticos en ggplot ya no es necesario definirlos en los geoms</importante>

Para añadir títulos a los ejes utiliza labs().

p1 <- p1 + labs(title = "Presión de vapor del mercurio"  
 , x = "Temperatura (ºC)", y = "Presión (mm de Hg)")

Para obtener el plot, ejecuta el objeto.

p1

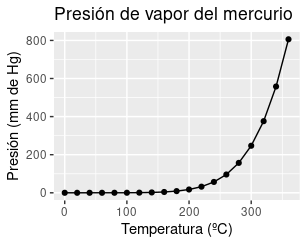
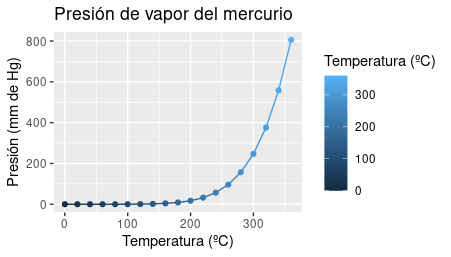


grafico de líneas y puntos

Para añadir graduación continua del color por temperatura.

p1 <- ggplot(data = pressure, mapping = aes(x = temperature, y = pressure  
 , color = temperature)) +  
 geom\_point() +  
 geom\_line() +  
 labs(title = "Presión de vapor del mercurio"  
 , x = "Temperatura (ºC)", y = "Presión (mm de Hg)"  
 , color = "Temperatura (ºC)")  
p1

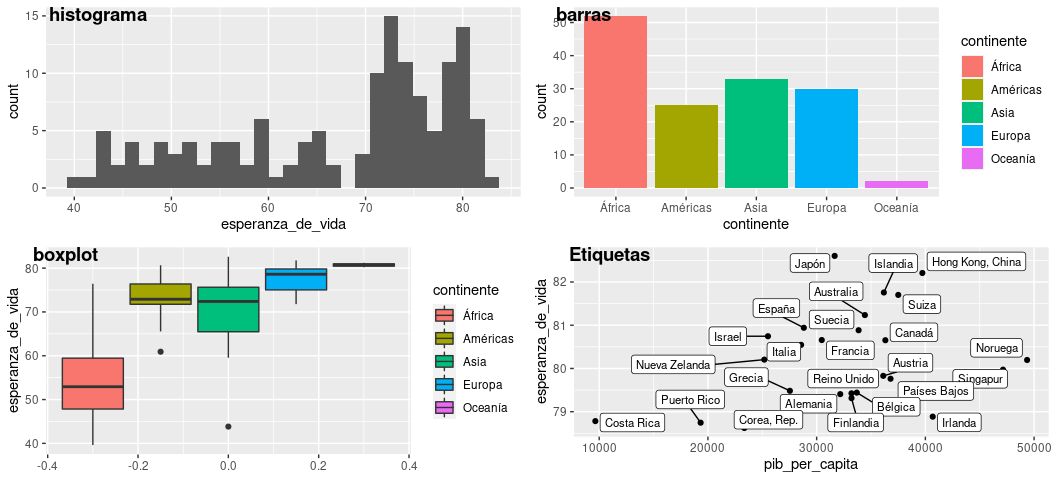


Escala de color contínua

#### OTROS GEOMS

Otros ejemplos de geoms con los datos de países en 2007.

library(ggrepel)  
  
paises07 <- as.data.frame(paises %>% filter(anio==2007)) # paises en 2007  
  
p1 <- ggplot(paises07, aes(x=esperanza\_de\_vida)) +  
 geom\_histogram()  
  
p2 <- ggplot(paises07, aes(x=continente, fill=continente)) +  
 geom\_bar()  
  
p3 <- ggplot(paises07, aes(y=esperanza\_de\_vida, fill=continente)) +  
 geom\_boxplot()  
  
p4 <- ggplot(paises07 %>% arrange(desc(esperanza\_de\_vida)) %>% head(25),  
 aes(x=pib\_per\_capita, y=esperanza\_de\_vida)) +  
 geom\_point() +  
 geom\_label\_repel(aes(label=pais), size=3)



#### OTROS ELMENTOS GRÁFICOS

Otros elementos a tener en cuenta en la presentación:

* Permite crear un panel de gráficos utilizando variables de segmentación: facet\_wrap() y facet\_grid()
* Insertar etiquetas y anotaciones de texto: consulta la función annotate(), geom\_text(), geom\_label(), geom\_label\_repel().
* Escalas: Permite cambiar los valores de los ejes. Por ejemplo, consulta scale\_x\_continuous() y scale\_x\_log10().
* Colores: Para cambiar las paletas de colores consulta scale\_colour\_discrete(), scale\_colour\_brewer(), scale\_colour\_manual(), scale\_fill\_viridis(), scale\_colour\_gradient()…
* Zoom: Importante, para realizar zoom utiliza xlim e ylim en: coord\_cartesian().
* Temas: Personaliza posición de la leyenda y resto de elementos gráficos, tamaños colores, fuente… Consulta: theme() y theme\_bw().
* Interactivos: packages [gganimate](https://gganimate.com/) y [plotly](https://plotly.com/r/).

Para ampliar la lectura consulta:

* [R for Data Science](https://r4ds.had.co.nz/) from Garrett Grolemund and Hadley Wickham.
* [ggplot2: Elegant Graphics for Data Analysis](https://ggplot2-book.org/) de Hadley Wickham.
* [ggplot2 extensions guide](https://exts.ggplot2.tidyverse.org/gallery/) lista los paquetes que extienden ggplot2.
* La [{ggplot2} cheat sheet](https://www.statsandr.com/blog/files/ggplot2-cheatsheet.pdf).

##### ACTIVIDAD GUIADA 1.3.

Mejora el Crea un resumen HTML de la esperanza\_de\_vida y el pib\_per\_capita en 2007 con datos de paises del package datos.

Para ello abre el fichero `13\_actividad.Rmd y completa los apartados solicitados entre < … > .

Genera el informe HTML con el botón **Knit**.

Observa:

* Se han añadido parámetros estéticos: color y size.
* La escala del eje x (pib\_per\_capita) se ha transformado a escala logarítmica con scale\_x\_log10().
* Se ha definido un panel temporal con facet\_wrap()
* Hay que imprimir el gráfico con print()

Adapta los comentarios a la nueva información y regenera el informe.

<sabías que>Hans Rosling de Gapminder utilizó un gráfico muy parecido a éste, añadiendo un quinto eje (el tiempo) en una famosa conferencia [TED Talk](https://www.ted.com/talks/hans_rosling_the_best_stats_you_ve_ever_seen) (14,6M de visualizaciones). Dedícale 1 minuto a ver la simulación dándole al botón de play (abajo izquierda), en el [siguiente enlace](https://www.gapminder.org/tools/" \l "$chart-type=bubbles). La herramienta utilizada Trendalyzer fue posteriormente adquirida por Google.</sabías que>

## COLECCIONES DE OBJETOS

### DEFINCIÓN DE LOS OBJETOS

#### TIPO BÁSICOS

##### NUMERIC

Los números, en R, en general se guardan en coma flotante y doble precisión ("double").

a <- 3.7  
class(a)

## [1] "numeric"

typeof(a)

## [1] "double"

<recuerda>Todo objeto tiene una clase que se puede consultar con la función class(). Adicionalmente se puede saber el tipo de datos interno con typeof().</recuerda>

La precisión doble es exacta hasta los 16 dígitos. La siguiente expresión debería dar 0, pero no es así.

sqrt(2)^2 - 2

## [1] 4.440892e-16

##### INTEGER

Para indicar que es un entero, hay que finalizar el número con L mayúscula. Ahorran memoria pero pierden precisión. No es muy habitual su uso en Data Science.

a <- 4L  
class(a)

## [1] "integer"

typeof(a)

## [1] "integer"

##### LOGICAL

Los tipos lógicos TRUE (o T) y FALSE (o F), son de uso común para filtrar vectores.

a <- 3 > 4  
a

## [1] FALSE

class(a)

## [1] "logical"

typeof(a)

## [1] "logical"

Se pueden convertir a 1 y 0 (respectivamente) con as.numeric().

b <- as.numeric(a)  
b

## [1] 0

##### CHARACTER

Texto entre comas simple o dobles.

a <- "hola" # o con comas simples: 'hola'   
class(a)

## [1] "character"

typeof(a)

## [1] "character"

##### FACTOR

Información categórica, por ejemplo, el color de los ojos. Por su naturaleza de conjunto, se explican dentro de la sección de vectores.

##### FECHAS Y HORARIOS

Las fechas y horarios, son una clase compleja. Internamente son de tipo "double" con dos clases: "POSIXct" "POSIXt".

a <- Sys.time()  
a

## [1] "2020-09-13 12:14:06 CEST"

class(a)

## [1] "POSIXct" "POSIXt"

typeof(a)

## [1] "double"

La clase "POSIXct" internamente almacena el número de **segundos** transcurridos entre la fecha y las 12:00 AM del 1 de enero de 1970 (en la zona del Tiempo Universal Coordinado (UTC)). Se puede obtener este valor con unclass().

unclass(a) # muestra el valor interno de a

## [1] 1599992046

La clase Date, almacena sólo la fecha.

b <- as.Date(a)  
b

## [1] "2020-09-13"

class(a)

## [1] "POSIXct" "POSIXt"

typeof(a)

## [1] "double"

La clase Date contiene el número de **días** transcurridos des de 1 de enero de 1970.

unclass(b)

## [1] 18518

Se ampliará esta clase en el capítulo dedicado al package lubridate.

#### VECTORES

Se define mediante la función c(). Adopta el mismo tipo básico que los datos que lo componen.

class(c(3.1,9.7))

## [1] "numeric"

class(c(4L, 8L))

## [1] "integer"

class(c(TRUE, FALSE))

## [1] "logical"

class(c("es", "un", "vector"))

## [1] "character"

class(c(Sys.Date(), Sys.Date() + 1, Sys.Date()+2))

## [1] "Date"

##### LONGITUD

Con length() se obtiene el número de elementos.

a <- c(2,4,6)  
length(a)

## [1] 3

##### NOMBRES

Los vectores pueden tener atributos. Es de uso común el atributo names.

names(a) <- c("dos","cuatro", "seis")  
a

## dos cuatro seis   
## 2 4 6

##### OPERACIONES ARITMÉTICAS

Operaciones aritméticas con vectores.

c(1,2,3) \* 2 # multiplicación por escalar

## [1] 2 4 6

c(1,2,3) + c(4,5,6) # suma vectores

## [1] 5 7 9

c(1,2,3) \* c(4,5,6) # producto elemento a elemento

## [1] 4 10 18

##### OPERACIONES CON TEXTO

paste(1:3,c("uno","dos", "tres"), sep="-") # junta vectores

## [1] "1-uno" "2-dos" "3-tres"

paste(c("Hola,","Mundo!"), collapse = " ") # junta elementos de un vector

## [1] "Hola, Mundo!"

substr("abcdefgh", 3, 4) # subcadenas

## [1] "cd"

##### SECUENCIAS

3:5

## [1] 3 4 5

seq(from = 2, to = 6, by = 2)

## [1] 2 4 6

rep(0, times = 3)

## [1] 0 0 0

rep(1:3, each = 2)

## [1] 1 1 2 2 3 3

##### FACTOR

Son vectores de enteros, donde cada entero tiene asignado una etiqueta.

color\_ojos <- factor(c("marron", "azul", "azul", "verde"))  
color\_ojos

## [1] marron azul azul verde   
## Levels: azul marron verde

El atributo levels contiene las etiquetas. Se puede consultar con levels(). Por defecto, los levels se crean en orden alfanumérico a partir de los valores del vector. También se pueden definir por parámetro.

color\_ojos <- factor(c("marron", "azul", "azul", "verde"),  
 levels = c("marron", "azul", "verde"))  
levels(color\_ojos)

## [1] "marron" "azul" "verde"

Los factores, se codifican internamente como ‘integer’. Cada entero, contiene la posición en el vector de levels. Esta forma de codificar, ahorra memoria y acelera los cálculos. Se puede recuperar los valores integer con unclass().

class(color\_ojos)

## [1] "factor"

typeof(color\_ojos)

## [1] "integer"

unclass(color\_ojos)

## [1] 1 2 2 3  
## attr(,"levels")  
## [1] "marron" "azul" "verde"

También se pueden transformar a character con as.character().

as.character(color\_ojos)

## [1] "marron" "azul" "azul" "verde"

#### MATRICES

Las matrices añaden otra dimensión (fila) a los vectores (columna).

a <- matrix(1:6, ncol = 2)  
a

## [,1] [,2]  
## [1,] 1 4  
## [2,] 2 5  
## [3,] 3 6

Por defecto se rellenan por columna, pero se puede modificar a rellenado por filas.

a <- matrix(1:6, ncol = 2, byrow = TRUE)  
a

## [,1] [,2]  
## [1,] 1 2  
## [2,] 3 4  
## [3,] 5 6

##### DIMENSIÓN

Se puede consultar el número de filas y columnas con dim(), ncol() y nrow().

dim(a)

## [1] 3 2

nrow(a)

## [1] 3

ncol(a)

## [1] 2

##### NOMBRES

Se puede asignar nombres a las filas y a las columnas.

colnames(a) <- paste0("V",1:ncol(a))  
rownames(a) <- LETTERS[1:(nrow(a))]  
a

## V1 V2  
## A 1 2  
## B 3 4  
## C 5 6

##### ARRAYS

Se puede generalizar las matrices a más de 2 dimensiones con array().

a <- array(11:18, dim=c(2,2,2))  
a

## , , 1  
##   
## [,1] [,2]  
## [1,] 11 13  
## [2,] 12 14  
##   
## , , 2  
##   
## [,1] [,2]  
## [1,] 15 17  
## [2,] 16 18

#### LISTAS

Las listas permiten agrupar objetos R con distintos tipos de datos.

a <- list(11:13, "R", matrix(1:4,2,2))  
a

## [[1]]  
## [1] 11 12 13  
##   
## [[2]]  
## [1] "R"  
##   
## [[3]]  
## [,1] [,2]  
## [1,] 1 3  
## [2,] 2 4

##### NOMBRES

Se les puede asignar un nombre a cada elemento.

names(a) <- c("es\_num", "es\_char", "es\_matriz")

#### DATA FRAMES

Formalmente son una lista de vectores de la misma longitud. Tienen forma de matriz.

a <- data.frame(V1=c(1,2,3), V2=c("R","S","T"), V3=c(TRUE, FALSE,TRUE))  
class(a)

## [1] "data.frame"

typeof(a)

## [1] "list"



Para conocer la estructura del objeto es útil la función str().

str(a)

## 'data.frame': 3 obs. of 3 variables:  
## $ V1: num 1 2 3  
## $ V2: chr "R" "S" "T"  
## $ V3: logi TRUE FALSE TRUE

Se utilizan las mismas reglas que en las matrices para saber la dimensión, nombres de filas y columnas.

### FILTROS

Para filtrar matrices y data frames se utiliza corchetes separados por coma [,]. En el caso de vectores, corchetes simples [].

Los datos de paises07 servirán de ejemplo.

class(paises07)

## [1] "data.frame"

dim(paises07)

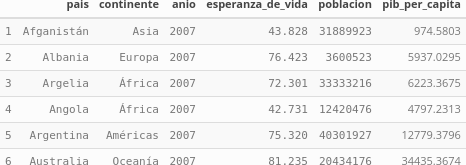
## [1] 142 6

colnames(paises07)

## [1] "pais" "continente" "anio"   
## [4] "esperanza\_de\_vida" "poblacion" "pib\_per\_capita"

Con head() (respec. tail()), se obtiene las primeras (resp. últimas) 6 filas.

head(países)



#### ENTEROS POSITIVOS

Se puede indicar las posiciones fila y columna a seleccionar. Por ejemplo, la esperanza de vida de Angola y Australia.

paises07[c(4,6),4]

## [1] 42.731 81.235

Para evitar que el resultado pase de data frame a vector, se utiliza el parámetro drop.

paises07[c(4,6), 4, drop=FALSE]



Si no se informa la columna (alternativamente fila), se seleccionan todos sus valores.

paises07[c(4,6),]



En el caso de vectores, se utiliza un solo corchete.

a <- c(5,2,4,7,9)  
a[c(2,2,1,5)]

## [1] 2 2 5 9

Observa que se pueden repetir posiciones o índices. Tampoco es necesario mantener un orden monótono en los índices.

<sabías que>R, a diferencia de otros lenguajes (C, Python…), empieza la posiciones en 1. Esto es habitual en lenguajes con un enfoque de programación matemática.</sabías que>

#### ENTEROS NEGATIVOS

Selecciona todas las columnas (alternativamente las filas) excepto las posiciones indicadas con valor negativo.

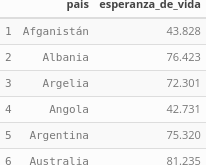
head(paises07[,-c(3,5)])



#### NOMBRES

Se utiliza habitualmente para seleccionar las columnas por su nombre.

head(paises07[, c("pais","esperanza\_de\_vida")])



#### DOLARES Y DOBLES CORCHETES

En listas y data frames se puede obtener un elemento con $nombre , o bien, con corchetes dobles

head(paises07$pib\_per\_capita)

## [1] 974.5803 5937.0295 6223.3675 4797.2313 12779.3796 34435.3674

head(paises07[["esperanza\_de\_vida"]])

## [1] 43.828 76.423 72.301 42.731 75.320 81.235

<sabías que>Una fuente habitual de error es intentar seleccionar las columnas de un objeto matrix con $nombre o dobles corchetes. Esto produce un error.</sabías que>

#### VALORES LÓGICOS

Se seleccionan las posiciones con valores TRUE o T y se descartan las posiciones FALSE o F.

head(paises07[,c(F, F, F, T, F, T)])

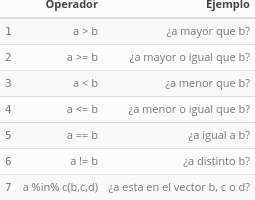


La forma habitual de seleccionar elementos de una matriz es mediante expresiones lógicas.

paises07[paises07$continente=="Oceanía",]



Para construir estas expresiones, puedes utilizar los siguientes operadores.



Operadores lógicos: (fuente: elaboración propia)

Estos operadores resultan extremadamente útiles en vectores.

1 > c(0, 1, 2)

## [1] TRUE FALSE FALSE

c(1, 2, 3) != c(3, 2, 1)

## [1] TRUE FALSE TRUE

El operador %in% tiene un funcionamiento un poco distinto. Evalúa cada elemento del objeto situado a su izquierda respecto a todos los elementos del objeto situado a su derecha.

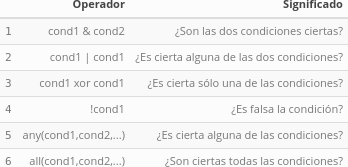
c(1, 2, 3, 4) %in% c(3, 4, 5)

## [1] FALSE FALSE TRUE TRUE

<recuerda>En operadores booleanos utiliza == en vez del operador ‘=’. Este último, hace la misma operación que <- de asignación de valores a objetos. </recuerda>

#### OPERACIONES BOOLEANAS

Puedes construir expresiones lógicas complejas utilizando los operadores booleanos habituales:

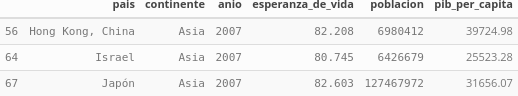


Operaciones booleanas (fuente: elaboración propia)

Para seleccionar los países asiáticos con esperanza de vida media en 2007 superior o igual a 80 años.

consulta <- paises07[paises07$continente=="Asia" & paises07$esperanza\_de\_vida >= 80 ,]

consulta



### MODIFICACION DE VALORES

R ofrece algunas facilidades para modificar los valores de sus objetos.

#### FILTRO Y ASIGNACIÓN “]<-”

Para modificar uno elemento de un vector, R te ofrece la posibilidad de aplicar un filtro con corchetes seguido de una asignación.

a <- 1:6  
a[1] <- 10  
a

## [1] 10 2 3 4 5 6

También puedes modificar varios elementos a la vez asignando un vector de la misma longitud.

a[c(3,5)] <- c(300,500)  
a

## [1] 10 2 300 4 500 6

a[4:6] <- a[4:6] + 1  
a

## [1] 10 2 300 5 501 7

Puedes añadir elementos nuevos a un vector.

a[7] <- 7  
a

## [1] 10 2 300 5 501 7 7

También puedes modificar varios elementos asignando un único valor.

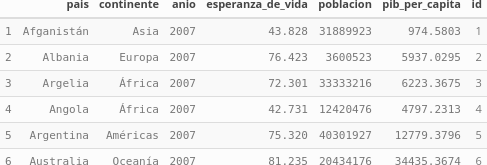
a[c(3,5,7)] <- 0  
a

## [1] 10 2 0 5 0 7 0

En data frames (y listas) se pueden añadir columnas (resp. elementos).

paises07$id <- 1:nrow(paises07)

head(paises07)



Modificar el nombre de una columna.

colnames(paises07)[colnames(paises07)=="id"] <- "identificador"  
colnames(paises07)

## [1] "pais" "continente" "anio"   
## [4] "esperanza\_de\_vida" "poblacion" "pib\_per\_capita"   
## [7] "identificador"

Eliminar las columnas (resp. elmentos) con el objeto NULL.

paises07$identificador <- NULL  
colnames(paises07)

## [1] "pais" "continente" "anio"   
## [4] "esperanza\_de\_vida" "poblacion" "pib\_per\_capita"

#### INFORMACIÓN FALTANTE

Para marcar los valores faltantes en R se utiliza la combinación NA (acrónimo de *Not Available*). Este valor especial tiene su propia aritmética.

a <- 3 + c(NA, 4)  
a

## [1] NA 7

Es decir, cuando un valor es desconocido, la mejor opción es no evaluar la suma con otros valores. Las expresiones lógicas se comportan de un modo similar.

c(8, NA) > 0

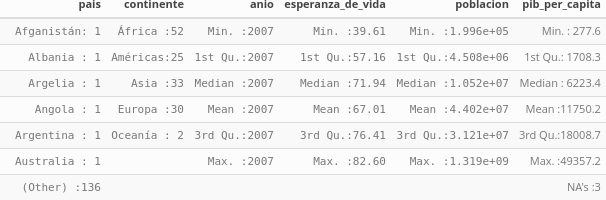
## [1] TRUE NA

##### SELECCION DE FILAS CON VALORES FALTANTES

Para seleccionar las filas con valores faltantes, entre otros usos, puedes utilizar la función is.na().

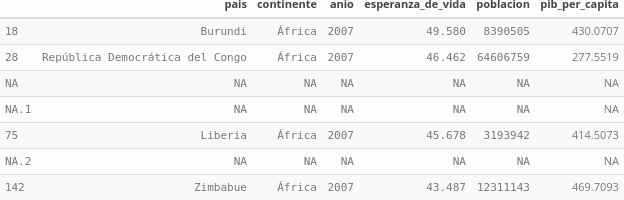
# Introducimos algunos NA para el ejemplo  
paises07NA <- paises07  
paises07NA$pib\_per\_capita[c(34,70,108)] <- NA

summary(paises07NA)



Para seleccionar los registros con missing en una columna.

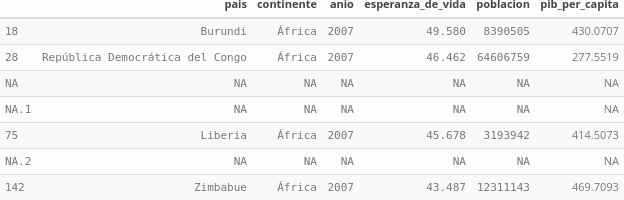
paises07NA[is.na(paises07NA$pib\_per\_capita),]



##### FILTROS EN VARIABLES CON VALORES FALTANTES

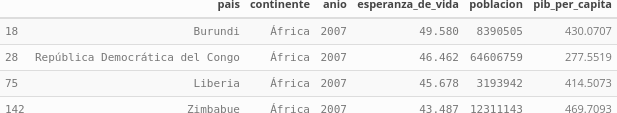
La presencia de NA en una variable dificulta la definición de condiciones lógicas. Por ejemplo, si seleccionamos países con PIB per cápita inferior a 500 dólares se obtiene el siguiente resultado incorrecto:

paises07NA[paises07NA$pib\_per\_capita<500,]



Esta problemática se puede resolver mediante la función which().

paises07NA[which(paises07NA$pib\_per\_capita<500),]



La función which() devuelve la posición de los valores TRUE dentro de un vector.

a <- c(NA, FALSE, NA, TRUE, FALSE, TRUE, NA )  
which(a)

## [1] 4 6

<recuerda>Utiliza which para “protegerte” de la presencia de valores faltantes en la definición de filtros.</recuerda>

##### ACTIVIDAD GUIADA 1.4.

Vamos a mejorar de nuevo el informe de los países introduciendo:

* Haz un panel de boxplot por año del PIB per cápita (escala log10) y la esperanza de vida.
* Mejora el gráfico de dispersión de países:
  + Recodifica Oceanía como Asia y Modifica los colores de los continentes.
  + Utiliza los siguiente paleta de colores:
    - “blue”: “África”
    - “green”: “Américas”
    - “red”: “Asia”
    - “yellow”: “Europa”
  + Inserta Etiquetas para España, China, Etiopía y Canadá.

Adapta los comentarios solicitados en 14\_actividad.R, ejecuta y observa:

* Se ha recodificado una variable factor.
* Se ha parametrizado los colores de los continentes y las etiquetas de los países.
* La función describir() devuelve una lista de resultados (plots y tablas).
* Se han introducido boxplots.
* Se ha construido un data frame con los países con etiqueta.
* Se ha seleccionado las variables a presentar en el summary.

## CONTROL DE FLUJO

Existen distintos mecanismos para controlar el flujo de un programa.

### FUNCIONES

Las funciones ayudan a automatizar piezas de nuestro código.

Para avanzar en el uso de R es importante entender un caso particular de funciones, las **funciones vectorizadas**. Éstas se aplican tantas veces como elementos tengamos en la entrada de la función.

Veamos unos ejemplos simples.

a <- seq(2,8,2)  
a

## [1] 2 4 6 8

a + a

## [1] 4 8 12 16

sqrt(a)

## [1] 1.414214 2.000000 2.449490 2.828427

Estas funciones se aplican a cada uno de los elementos del objeto a.

Otro ejemplo, sería una función para reemplazar los valores faltantes con el valor cero.

a <- c(3, NA, -8, NA)  
remplazar\_na\_cero <- function(x) {  
 x[is.na(x)] <- 0  
 return(x)  
}  
remplazar\_na\_cero(a)

## [1] 3 0 -8 0

En los siguientes puntos se presenta como vectorizar funciones sobre objetos R (vectores, listas, data frames…).

### CONDICIONALES

Una forma de modificar el flujo de un programa, es mediante el uso de expresiones if-then-else. Su forma más general sería la siguiente.

a <- 1  
b <- 1  
  
if (a > b){  
 print("a mayor que b")  
} else if (a < b) {  
 print("a menor que b")  
} else {  
 print("a igual a b")  
}

## [1] "a igual a b"

La expresión if() evalúa un único valor lógico. Es habitual el uso de any() y all() para tratar condiciones sobre vectores.

a <- c(4, NA, 5)   
  
if (all(is.na(a))) {  
 print("todos son NA")  
} else if(any(is.na(a))) {  
 print("hay algun NA")  
} else {  
 print("no hay NA")  
}

## [1] "hay algun NA"

La función ifelse() se puede utilizar sobre vectores y devuelve un vector.

a <- c(2:-2)  
a

## [1] 2 1 0 -1 -2

sqrt(a)

## Warning in sqrt(a): NaNs produced

## [1] 1.414214 1.000000 0.000000 NaN NaN

sqrt(ifelse(a>=0, a, NA))

## [1] 1.414214 1.000000 0.000000 NA NA

sqrt() al aplicarse sobre valores negativos, devuelve un nuevo valor NaN (acrónimo de *Not a Number*). La expresión ifelse() evita el NaN devolviendo un NA.

En R, habitualmente se utiliza la siguiente alternativa basada en filtros.

raiz <- rep(NA, length(a))  
raiz[a>=0] <- sqrt(a[a>=0])  
raiz

## [1] 1.414214 1.000000 0.000000 NA NA

### BUCLES

Si bien existen bucles de tipo while() y repeat, el más habitual es for(). El siguiente ejemplo muestra como calcular el cuadrado de un vector de números con for().

a <- seq(2,8,2)  
a

## [1] 2 4 6 8

cuadrados <- NULL  
  
for (x in a){  
 cuadrados <- c(cuadrados, x^2)   
}  
cuadrados

## [1] 4 16 36 64

La variable x recorre los elementos del objeto a. El resultado se va acumulando en forma de vector en el objeto cuadrados. Esta forma de acumular el resultado, es poco eficiente, ya que obliga a ir ampliando la memoria en cada iteración. La forma correcta es, primero reservar el espacio y posteriormente rellenar el objeto:

cuadrados <- rep(NA,length(a))  
  
for (i in 1:length(a)){  
 cuadrados[i] <- a[i]^2  
}  
cuadrados

## [1] 4 16 36 64

Observa como ahora el bucle recorre los índices de posiciones del objeto a, no directamente sus elementos, esto permite que se pueda utilizar el mismo índice para rellenar el objeto cuadrados.

En muchas ocasiones, se puede reemplazar el uso de un bucle mediante funciones de R vectorizadas. En este caso, la forma más simple de calcular los cuadrados de un vector es la siguiente.

cuadrados <- a^2  
cuadrados

## [1] 4 16 36 64

En el apartado de [RESUMENES DE AGREGADOS](#resumenes-de-agregados), se presenta las funciones de la familia apply(). Éstas representan una alternativa a los bucles for(), para aplicar una misma función de resumen (vectorizar) sobre agrupaciones de objetos de entrada (filas de una matriz, columnas de un data frame…).

## GESTIÓN DE DATOS

A continuación, se presentan las funciones para la lectura/escritura de datos, cruce y construcción de tablas resumen.

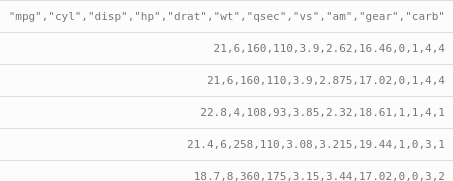
### IMPORTAR Y EXPORTAR DATOS

RStudio aporta facilidades para la importación de datos a través del menú: File -> Import Data set -> From Text...

#### LECTURA TEXTO CON SEPARADOR

Para leer el siguiente fichero de texto:

cat(paste0(readLines("www/mtcars.csv", n = 6), collapse="\n"))



Dado que se trata de un fichero con un carácter separado, la función genérica para teste tipo de ficheros es read.table().

mtcars <- read.table("www/mtcars.csv", dec=".",sep=",", header=TRUE)

Los parámetros especificados son:

* dec: El separador decimal. En Europa se acostumbra a utilizar la coma “,”.
* sep: El separador de columnas. Es habitual usar también “;” y “\t” cuando es un tabulador.
* header: Indica si la primera fila contiene el nombre de las variables o no.

También es común el uso de fileEncoding="Latin1" cuando el fichero se ha creado con Windows y utiliza caracteres especiales de texto (tildes, …).

Dada la estructura del fichero, con read.csv() la carga es más simple.

mtcars <- read.csv("www/mtcars.csv")

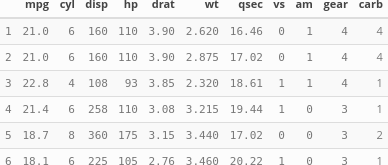
Cuando el csv es europeo, “,” decimal y “;” como separador, se utiliza read.csv2().

Es una buena práctica verificar su carga con dim(), head() y summary().

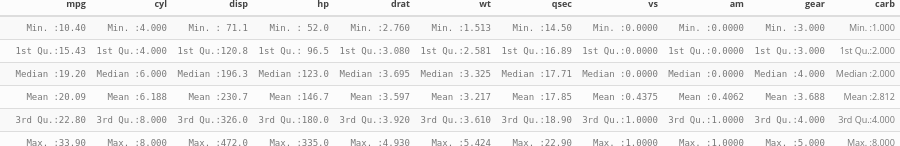
dim(mtcars)

## [1] 32 11

head(mtcars)



summary(mtcars)



#### ESCRITURA TEXTO CON SEPARADOR

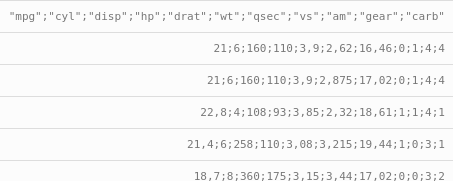
Para la escritura, puedes utilizar sus equivalentes: write.table(), write.csv(), write.csv2(). Cabe destacar algunas pequeñas variaciones.

write.csv2(mtcars, file="www/mtcars2.csv", row.names=FALSE, na="")

* row.names: Si no se especifica, por defecto, se inserta el número de fila.
* na: Si no se especifica, introduce el texto NA.

Observa como el nuevo csv ahora tiene el formato de csv europeo.

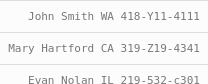
cat(paste0(readLines("www/mtcars2.csv", n = 6), collapse="\n"))



#### TEXTO CON ANCHO FIJO

Veamos un ejemplo de fichero de ancho fijo:

cat(paste0(readLines("www/fwf-sample.txt", n = 6), collapse="\n"))

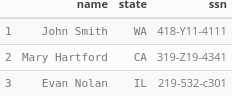


Para leer texto de ancho fijo, se utiliza read.fwf().

fwf\_sample <- read.fwf("www/fwf-sample.txt", widths=c(20,10,12)  
 , col.names=c("name", "state", "ssn"))

* widths: Son los anchos de cada variable.
* col.names: Nombres de las variables.

fwf\_sample



### CRUCE ENTRE TABLAS

Para cruzar 2 o más tablas es necesario que éstas compartan un conjunto de índices.

#### CRUCES POR ORDENACIÓN

Una forma indexar consiste simplemente en compartir una misma ordenación de filas o columnas.

##### ORDENACION DE TABLAS

Para ordenar un vector se utiliza la función sort().

a <- c(5,2,9,4)  
sort(a)

## [1] 2 4 5 9

Lo mismo se puede hacer en dos pasos. Primero, con order() se obtienen los índices de ordenación.

indices\_ordenacion <- order(a)  
indices\_ordenacion

## [1] 2 4 1 3

Segundo se seleccionan los elementos del vector a en ese orden.

a[indices\_ordenacion]

## [1] 2 4 5 9

Observa que el resultado es idéntico a utilizar directamente sort().

Mismo principio ahora, para ordenar la tabla paises07 por continente (decreciente) y esperanza de vida (creciente).

paises07Ord <- paises07[order(paises07$continente,paises07$esperanza\_de\_vida  
 ,decreasing = c(T,F)),]

head(paises07Ord)

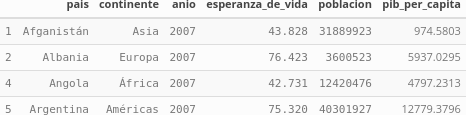


##### UNIÓN POR COLUMNAS

Para unir las columnas de 2 tablas que comparten ordenación de sus filas se utiliza cbind().

a1 <- paises07[,1:2]  
a2 <- paises07[,4:5]  
a <- cbind(a1, a2)

head(a)

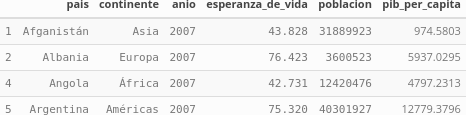


##### UNIÓN POR FILAS

Unir las filas de dos tablas con la misma ordenación de sus columnas rbind().

a1 <- paises07[1:2,]  
a2 <- paises07[4:5,]  
a <- rbind(a1, a2)

head(a)



#### CRUCE POR CAMPOS CLAVE

Un campo clave consiste en una variable que identifica cada registro de forma única.

Por ejemplo, pais y continente son campos clave ya que identifican de forma única a sus respectivas entidades en la tabla de paises.

Antes de realizar cualquier cruce por campos clave, es importante saber si éste tiene o no duplicados.

##### VECTOR DE CLAVES

Para obtener las claves únicas de un vector utiliza unique().

sort(unique(as.character(paises07$continente)))

## [1] "África" "Américas" "Asia" "Europa" "Oceanía"

Para saber qué filas de un data frame están duplicadas, utiliza duplicated().

paisesDup <- paises07[c(1,1,2,3,3,4,5,5,6),]  
any(duplicated(paisesDup)) # ¿Tiene duplicados?

## [1] TRUE

as.character(paisesDup$pais[duplicated(paisesDup)]) # que países son

## [1] "Afganistán" "Argelia" "Argentina"

##### FUSIÓN CON CLAVES

Para cruzar 2 tablas con claves, utiliza merge().

###### INNER JOIN

Analiza el siguiente ejemplo:

a1 <- data.frame(ID = c(1,2,3), X1 = c(1,1,2), X2 = c("b","a","a"))  
a2 <- data.frame(ID = c(2,3,3,4), X1 = c(1,1,2,2), X3 = c(FALSE, TRUE, FALSE, FALSE))  
a\_inner <- merge(a1, a2)

a\_inner



R, ha seleccionado ID y X1 como campos clave porque son las variables con nombres coincidentes en ambas tablas. Veremos cómo modificar esto.

Otra decisión que R ha tomado, es que **sólo** ha recuperado los **registros** con valores de ID y X1 **compartidos** en ambas tablas. Este comportamiento en el lenguaje SQL, es conocido como *INNER JOIN*.

###### FULL JOIN

Para conservar **todos los registros de las tablas originales**, tanto de la tabla izquierda como derecha, coincida o no, se utiliza el parámetro all=TRUE.

a\_full <- merge(a1, a2, all = TRUE)

a\_full



Observa ahora, no ha descartado ningún registro de las tablas origen. Además, ha rellenado con NA los campos que no cruzan. Este comportamiento en SQL es conocido como *FULL JOIN* .

###### LEFT JOIN

Para imponer que conserve **todos los valores origen de la tabla izquierda** (a1) y que descarte los de la taba derecha que no sean compartidos utiliza all.x=TRUE.

a\_left <- merge(a1, a2, all.x = TRUE)

a\_left



Vemos que ID=1 ahora está presente y ID=4 ha quedado descartado. También que ID=3 sólo aparece una vez. Este comportamiento en SQL, es conocido como LEFT JOIN .

###### RIGHT JOIN

Para preservar **todos los valores de la tabla derecha** utiliza all.y=TRUE. Este comportamiento en SQL es el conocido como *RIGHT JOIN*.

a\_right <- merge(a1, a2, all.y = TRUE)

a\_right



Ahora ID=3 aparece 2 veces. También aparece ID=4.

###### DEFINCIÓN DE LAS CLAVES

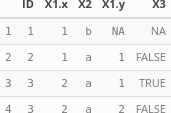
Para definir las claves del cruce, es necesario pasar como parámetro los nombre de las variables:

* Cuando los nombres de los campos clave son compartidos en ambas tablas, utiliza by.
* Cuando hay cambios de nombre, utiliza by.x y by.y

El siguiente ejemplo, realiza un *FULL JOIN* definiendo como clave la variable ID.

a\_comparte <- merge(a1, a2, all.x=TRUE, by = c("ID"))   
# de forma equivalente  
a\_comparte <- merge(a1, a2, all.x=TRUE, by.x = c("ID"), by.y = c("ID"))

a\_comparte



R ha añadido el sufijo .x y .y a X1, porque es una variable compartida que ahora no es clave para el cruce. Observa también que ID=3 aparece 2 veces ya que está duplicado en la tabla a2.

### RESUMENES DE AGREGADOS

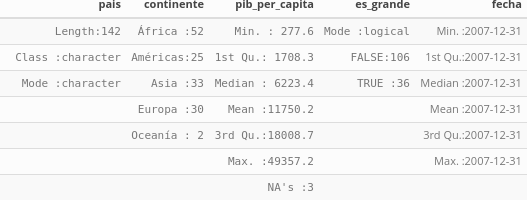
R tiene muchas opciones para construir agregados.

#### SUMMARY

Para tener una primera impresión sobre los datos utiliza summary().

paises07Summ <- paises07  
paises07Summ$pais <- as.character(paises07Summ$pais)  
paises07Summ$fecha <- as.Date(paste(paises07Summ$anio,"12","31",sep="-"),"%Y-%m-%d")  
paises07Summ$es\_grande <- paises07Summ$poblacion > 30\*10^6  
paises07Summ$pib\_per\_capita[c(34,70,108)] <- NA

summary(paises07Summ[,c("pais","continente","pib\_per\_capita","es\_grande","fecha"),])



summary en factor y logic devuelve una tabla de frecuencias, en cambio en character no. En numericy Date devuelve 6 estadísticos básicos de resumen más el conteo de NAs.

#### ESTADÍSTICOS BASICOS

Los 6 estadísticos básicos del summary son:

min(paises07Summ$pib\_per\_capita, na.rm=TRUE)

## [1] 277.5519

quantile(paises07Summ$pib\_per\_capita, 0.25, na.rm=TRUE)

## 25%   
## 1708.268

median(paises07Summ$pib\_per\_capita, na.rm=TRUE)

## [1] 6223.367

mean(paises07Summ$pib\_per\_capita, na.rm=TRUE)

## [1] 11750.22

quantile(paises07Summ$pib\_per\_capita, 0.75, na.rm=TRUE)

## 75%   
## 18008.73

max(paises07Summ$pib\_per\_capita, na.rm=TRUE)

## [1] 49357.19

El parámetro na.rm=TRUE evita que las funciones resumen, devuelvan un NA debido a la presencia de NA s en la variable pib\_por\_capita. Observa como esto sucede en la siguiente función.

mean(paises07Summ$pib\_per\_capita)

## [1] NA

La función quantile(), por sí sola, puede calcular todos los valores, excepto la media.

quantile(paises07Summ$pib\_per\_capita, seq(0,1,1/4), na.rm=TRUE)



#### TABLAS DE FRECUENCIAS

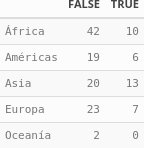
Para obtener una tabla de frecuencias, utiliza table(). Para una variable.

table(paises07Summ$continente) # una variable



Tablas cruzadas de dos o más variables.

table(paises07Summ$continente,paises07Summ$es\_grande)



<sabías que>Asia es el continente con mayor número de países grandes (de más de 30M de hab.)</sabías que>.

En muchas ocasiones, es útil tramar las variables numéricas en intervalos. En R utiliza cut().

Vamos a crear la variable nivel de riqueza a partir del pib\_per\_capita cortando la variable por 1700$ y 18000$.

paises07Summ$nivel\_de\_riqueza <- cut(paises07Summ$pib\_per\_capita,   
 breaks=c(-Inf,1700,18000,Inf))

head(paises07Summ[,c("pais","pib\_per\_capita","nivel\_de\_riqueza")])



<recuerda>Para definir los breaks, R permite utilizar -Inf e Inf (infinito) como si fueran números.</recuerda>

cut() crea un factor ordenado a partir de la variable numeric con los siguientes levels:

levels(paises07Summ$nivel\_de\_riqueza)

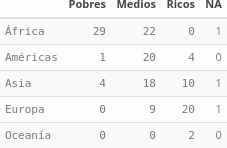
## [1] "(-Inf,1.7e+03]" "(1.7e+03,1.8e+04]" "(1.8e+04, Inf]"

Para modificar las etiquetas utiliza la misma función levels().

levels(paises07Summ$nivel\_de\_riqueza) <- c("Pobres","Medios","Ricos")

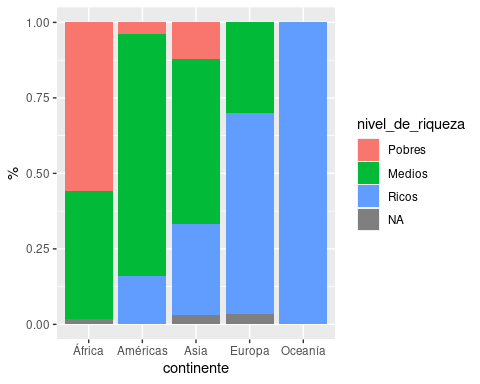
Mostramos a continuación la distribución de países ricos y pobres por continente en 2007.

table(paises07Summ$continente, paises07Summ$nivel\_de\_riqueza, useNA = "ifany")



Gráficamente.

ggplot(paises07Summ) +  
 geom\_bar(aes(x=continente, fill=nivel\_de\_riqueza), position = "fill") +  
 labs(y="%")



<recuerda>Para definir el nivel de riqueza de los países en 2007 se ha realizado la siguiente transformación:</recuerda>

paises07$nivel\_de\_riqueza <- cut(paises07$pib\_per\_capita, breaks=c(-Inf,1700,18000,Inf))  
levels(paises07$nivel\_de\_riqueza) <- c("Pobres","Medios","Ricos")

#### AGREGADOS POR SUBGRUPOS

Además de frecuencias, es necesario saber calcular otros estadísticos básicos para subgrupo de un data frame. Por ejemplo, la media de esperanza de vida y del pib per cápita, así como, el total de población por continente y nivel de riqueza.

agg1 <- aggregate(paises07[,c("esperanza\_de\_vida","pib\_per\_capita")],  
 list(continente = paises07$continente  
 , nivel\_de\_riqueza=paises07$nivel\_de\_riqueza),mean)  
agg2 <- aggregate(paises07[,c("poblacion"),drop=FALSE],  
 list(continente = paises07$continente  
 , nivel\_de\_riqueza=paises07$nivel\_de\_riqueza),sum)

<recuerda>aggregate() permite calcular un mismo estadístico sobre varias variables segmentando por más de un eje.</recuerda>

Se juntan ahora los dos agregados con merge() y se genera la variable combinada continente\_riqueza.

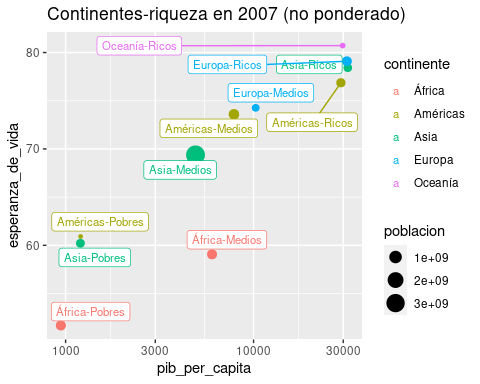
agg3 <- merge(agg1,agg2)  
agg3$continente\_riqueza <- paste(agg3$continente,agg3$nivel\_de\_riqueza,sep="-")

agg3



El mismo gráfico visto para países, ahora, agregado a nivel de continente y riqueza.

ggplot(agg3, aes(x=pib\_per\_capita  
 , y=esperanza\_de\_vida  
 , label=continente\_riqueza  
 , color =continente  
 , size=poblacion)) +  
 geom\_point() +  
 geom\_label\_repel(size=3, force=10) +  
 scale\_x\_log10() +  
 labs(title="Continentes-riqueza en 2007 (no ponderado)")



Contienentes (Fuente: Elaboración propia, datos de Gapminder)

#### LA FAMILIA APPLY

Como se ha comentado, las funciones apply permiten vectorizar las operaciones de forma eficiente, sin utilizar bucles for().

##### TAPPLY

Es una alternativa a aggregate() para aplicar una función de agregación sobre una única variable y segmentando por un único eje.

tapply(paises07$esperanza\_de\_vida,paises07$continente,quantile,probs=0.25)

## África Américas Asia Europa Oceanía   
## 47.83400 71.75200 65.48300 75.02975 80.46175

##### LAPPLY

Permite aplicar una función sobre cada elemento de un vector o lista. Devuelve una lista.

lapply(paises07[,-1], function(x) summary(x))

## $continente  
## África Américas Asia Europa Oceanía   
## 52 25 33 30 2   
##   
## $anio  
## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 2007 2007 2007 2007 2007 2007   
##   
## $esperanza\_de\_vida  
## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 39.61 57.16 71.94 67.01 76.41 82.60   
##   
## $poblacion  
## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 1.996e+05 4.508e+06 1.052e+07 4.402e+07 3.121e+07 1.319e+09   
##   
## $pib\_per\_capita  
## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 277.6 1624.8 6124.4 11680.1 18008.8 49357.2   
##   
## $nivel\_de\_riqueza  
## Pobres Medios Ricos   
## 36 69 37

##### SAPPLY

Misma funcionalidad que lapply, pero simplificando el resultado a vector cuando es posible.

sapply(paises07, function(x) class(x))

## pais continente anio esperanza\_de\_vida   
## "factor" "factor" "integer" "numeric"   
## poblacion pib\_per\_capita nivel\_de\_riqueza   
## "integer" "numeric" "factor"

##### APPLY

Calcula agregados sobre cada una de los vectores marginas fila (MARGIN = 1), alternativamente columna (MARGIN = 2) de una matriz.

a <- matrix(1:10,ncol=2)  
apply(a,MARGIN = 1,sum)

## [1] 7 9 11 13 15

Columnas,

a <- matrix(1:10,ncol=2)  
apply(a,2,sum)

## [1] 15 40

#### LAS ALTERNATIVAS TYDIVERSE

El universo de [Tydiverse](https://www.tidyverse.org/) consiste en una colección de packages de R orientados al Data Science. Se recomienda la consulta del libro online [R para Ciencia de Datos](https://es.r4ds.hadley.nz/index.html).

El package *dplyr*, al igual que *ggplot2*, está desarrollado por Hadley Wickham. Este paquete, supone un cambio de filosofía en la forma de trabajar con los datos y se está transformando en un nuevo estándar en R.

##### DPLYR: ARRANGE, FILTER, SELECT

* arange: Ordena los registros de un data frame.
* filter: Filtra a partir de un conjunto condiciones lógicas.
* select: Selecciona las variables.

<ejemplo>Paises asiaticos con PIB per cápita inferior a 1000 $ en 2007.</ejemplo>

pais07dplyr <- paises07 %>%   
 arrange(desc(esperanza\_de\_vida)) %>%   
 filter(continente=='Asia' & pib\_per\_capita < 1000) %>%   
 select(pais,esperanza\_de\_vida,pib\_per\_capita)



Las 3 comparten el hecho que no necesitan de comillas para referirse a las variables. También utilizan el operador %>% llamado *pipe*. Éste sustituye el parámetro de entrada de datos. De esta forma, permite escribir las operaciones de izquierda a derecha facilitando así su lectura.

##### DPLYR: MUTATE, GROUP\_BY, SUMMARISE

* mutate: Crea o modifica las variables del data frame.
* group\_by: Especifica una o varias variables de agrupación.
* summarise: define los agregados de las variables originales.

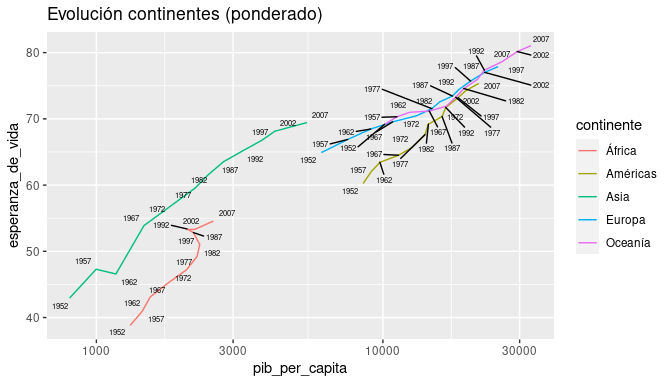
<ejemplo>Esperanza de vida y PIB per cápita medio por continente y año.</ejemplo>

continentes\_anio <- paises %>%   
 arrange(continente,anio) %>%   
 mutate(esperanza\_de\_vida\_total = esperanza\_de\_vida \* poblacion  
 , pib\_total = pib\_per\_capita \* poblacion) %>%   
 group\_by(continente,anio) %>%   
 summarize(n\_paises=n()  
 , esperanza\_de\_vida = sum(esperanza\_de\_vida\_total ) /  
 sum(poblacion )  
 , pib\_per\_capita = sum(pib\_total) /   
 sum(poblacion)  
 , poblacion = sum(poblacion))



Para la evolución de los continentes en ggplot2, es útil el [geom\_path()](https://ggplot2.tidyverse.org/reference/geom_path.html.)

ggplot(continentes\_anio, aes(x=pib\_per\_capita  
 , y=esperanza\_de\_vida  
 , color =continente)) +  
 geom\_path() +  
 scale\_x\_log10() +  
 geom\_text\_repel(aes(label=anio), size=2, color="black") +  
 labs(title="Evolución continentes (ponderado)")



Fuente: Elaboración propia con datos de Gapminder

##### ACTIVIDAD GUIADA 1.5.

Vamos a incorporar los gráficos de evolución de los continentes en el informe teniendo en cuenta los siguientes cambios

* Recodifica Oceanía como Asia y Modifica los colores de los continentes.
* Utiliza la paleta de colores de la actividad guiada 1.4.

Adapta los comentarios solicitados en 15\_actividad.R, ejecuta y observa:

* Se ha utilizado dplyr: arrange,mutate,group\_by y summarize.
* Se ha utilizado geom\_path.

### LUBRIDATE: GESTIÓN DE FECHAS Y HORARIOS

Para obtener el día actual con *lubridate* utiliza today(), día y hora con now().

library(lubridate)

##   
## Attaching package: 'lubridate'

## The following object is masked from 'package:cowplot':  
##   
## stamp

## The following objects are masked from 'package:base':  
##   
## date, intersect, setdiff, union

today()

## [1] "2020-09-13"

now()

## [1] "2020-09-13 12:14:37 CEST"

###### CREAR FECHAS HORAS

Para crear fechas.

ymd("2017-01-31") # Días des de texto

## [1] "2017-01-31"

ymd\_hms("2017-01-31 20:11:59") # Día y hora des de texto

## [1] "2017-01-31 20:11:59 UTC"

make\_datetime(2020, 08, 24, 15, 36) # Des de los componentes

## [1] "2020-08-24 15:36:00 UTC"

as\_datetime(today()) # Conversión

## [1] "2020-09-13 UTC"

###### OBTENER COMPONENTES

Para obtener los meses y días de la semana en Castellano.

Sys.setlocale(locale="es\_ES.UTF-8")

## [1] "LC\_CTYPE=es\_ES.UTF-8;LC\_NUMERIC=C;LC\_TIME=es\_ES.UTF-8;LC\_COLLATE=es\_ES.UTF-8;LC\_MONETARY=es\_ES.UTF-8;LC\_MESSAGES=en\_US.UTF-8;LC\_PAPER=en\_US.UTF-8;LC\_NAME=C;LC\_ADDRESS=C;LC\_TELEPHONE=C;LC\_MEASUREMENT=en\_US.UTF-8;LC\_IDENTIFICATION=C"

fechahora <- ymd\_hms("2019-03-28 15:11:23")  
  
year(fechahora)

## [1] 2019

month(fechahora, label = TRUE) # meses en texto

## [1] mar  
## 12 Levels: ene < feb < mar < abr < may < jun < jul < ago < sep < ... < dic

wday(fechahora, label = TRUE, abbr = FALSE) # días de la semana sin abreviar

## [1] jueves  
## 7 Levels: domingo < lunes < martes < miércoles < jueves < ... < sábado

###### MODIFICAR FECHAS Y HORAS

Editar componentes

year(fechahora) <- 2020  
fechahora

## [1] "2020-03-28 15:11:23 UTC"

hour(fechahora) <- hour(fechahora) + 10  
fechahora

## [1] "2020-03-29 01:11:23 UTC"

También resulta interesante los distintos tipos de redondeo de fechas.

floor\_date(fechahora, "month")

## [1] "2020-03-01 UTC"

Alternativamente round\_date() para aproximar y ceiling\_date() valor por encima.

###### DURACIONES

Se miden en segundos.

edad\_h <- today() - ymd(19760623)  
edad\_h

## Time difference of 16153 days

as.duration(edad\_h)

## [1] "1395619200s (~44.22 years)"

¡Ten cuidado! Están condicionados a cambios de huso horario (hibierno/verano).

fechahora <- ymd\_hms("2020-03-28 15:11:23", tz = "Europe/Madrid")  
fechahora

## [1] "2020-03-28 15:11:23 CET"

fechahora + dhours(24)

## [1] "2020-03-29 16:11:23 CEST"

###### PERIODOS

No se miden en segundos si no directamente en la unidad definida.

fechahora + hours(24)

## [1] "2020-03-29 15:11:23 CEST"

Ahora ya no hay la problemática del cambio de huso horario.

# ANEXO: README DE R

### CLONAR EL PROYETO GITHUB EN LOCAL

Abrimos una línea de comandos (cmd o anaconda en Windows, terminal en Linux).

git clone https://github.com/griu/mbdds\_fc20.git

### PREPARACIÓN ENTORNO RSTUDIO

#### CREAR EL PROYECTO

Crea un nuevo proyecto de RStudio sobre tu directorio local.

File > New Project > Existing Directory > mbdds\_fc20 > OK

El nuevo proyecto ya está conectado a Git.

#### INSTALAR PACKAGES DES DE R

En el entorno de Linux, (no Windows) es necesario haber instalado des del terminal los siguientes paquetes: libxml2-dev, libssl-dev, libcurl4-openssl-dev y libmagick++-dev .

Instala de uno en uno los siguientes paquetes desde la consola de RStudio.

install.packages("pander")  
install.packages("kableExtra")  
install.packages("tidyverse")  
install.packages("reticulate")  
webshot::install\_phantomjs()  
install.packages("magick")  
install.packages("ggrepel")  
install.packages("cowplot")  
install.packages("mosaicData")  
install.packages("datos")  
install.packages("flextable")  
install.packages("gganimate")