

Introducción a R (II)

FSC

Contents

Antes de empezar...	1
Programando en R	1
if	1
Ejercicios:	3
else	3
Motivational example: Máquina Tragaperras	4
Estrategia general en programación	5
Instrucciones secuenciales	5
Casos paralelos	5
Ejecutando programas en R	8
for and while loops	8
Paquetes en R	9
Instalar paquetes de CRAN	10
Instalar paquetes de Bioconductor	10
Instalar paquetes via <i>devtools</i>	10
Importación de datos en R	11
Importacion de datos desde archivos	11
Funciones standard	12
Cómo abrir grandes archivos	13
<i>tidyverse</i>	14
<i>Tidy</i> data	17
Importando datos de bases de datos	21
Datos estructurados: RMySQL -> RMariaDB	21
Datos estructurados: APIs y paquetes dedicados	24
Datos no estructurados:	25
Datos de un link de internet	25
Web scratching	25

Antes de empezar...

Los materiales de esta clase han sido preparados utilizando dos libros que están disponibles bajo licencia OUP y que os invito a explorar:

Hands-on programming with R <https://rstudio-education.github.io/hopr/>

Data Science with R <https://rafalab.github.io/dsbook/>

Programando en R

if

Utilizamos el comando `if` para seleccionar situaciones que cumplen una determinada condicion. Si se cumple esto, haz aquello:

```
if (this) {
  that
}
```

```
## Error in eval(expr, envir, enclos): object 'this' not found
```

this tiene que ser el resultado de un test lógico y resultar por tanto en TRUE o FALSE.

Si la condición se cumple (TRUE) se ejecutará el código entre corchetes. Si no se cumple (FALSE) no se hará nada.

Por ejemplo, si un número es negativo, multiplica por (-1) para hacerlo positivo:

```
num<-(-1)
num<-2
if (num < 0) {
  num <- num * -1
}
```

```
num <- -2
if (num < 0) {
  num <- num * -1
}
num
```

```
## [1] 2
```

```
## 2
```

```
num <- 4
if (num < 0) {
  num <- num * -1
}
num
```

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

```
## 4
```

El resultado del test lógico tiene que ser un vector de una sola dimension. Si es un vector de varios TRUE/FALSE se evaluará sólo el primer elemento del vector.

*Recuerda: las funciones **any** y **all** condensan las entradas de un vector lógico en un solo valor*

Puedes incluir tantas líneas de código como quieras entre los corchetes:

```
num <- -1
if (num < 0) {
  print("num is negative.")
  print("Don't worry, I'll fix it.")
  num <- num * -1
  print("Now num is positive.")
}
```

```
## [1] "num is negative."
## [1] "Don't worry, I'll fix it."
## [1] "Now num is positive."
```

```
## "num is negative."
## "Don't worry, I'll fix it."
## "Now num is positive."
num
```

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

```
## 1
```

Ejercicios:

Qué devuelven los siguientes códigos? (Intenta razonarlo y no ejecutarlo) Caso 1.

```
x <- 1
if (3 == 3) {
  x <- 2
}
x
```

```
## [1] 2
```

Caso 2.

```
x <- 1
if (TRUE) {
  x <- 2
}
x
```

```
## [1] 2
```

Case 3.

```
x <- 1
if (x == 1) {
  x <- 2
  if (x == 1) {
    x <- 3
  }
}
x
```

```
## [1] 2
```

else

Hasta ahora si no se cumple la condicion impuesta, R no hace nada. Pero puede ser interesante que R haga una cosa distinta si se cumple que si no se cumple la condición. Para eso usamos **else**

```
if (this) {
  Plan A
} else {
  Plan B
}
```

```
## Error: <text>:2:8: unexpected symbol
## 1: if (this) {
## 2:   Plan A
##      ^
```

Vamos a intentar hacer un código que redondee un decimal al entero más cercano:

```
a <- 3.14
```

la función **trunc** nos da el entero más cercano:

```
dec <- a - trunc(a)
dec
```

```
## [1] 0.14
```

```
## 0.14
```

```
if (dec >= 0.5) {
  a <- trunc(a) + 1
} else {
  a <- trunc(a)
}
a
```

```
## [1] 3
```

```
## 3
```

Si hay más de dos casos, podemos usar `else if`:

```
a <- 1
b <- 1
if (a > b) {
  print("A wins!")
} else if (a < b) {
  print("B wins!")
} else {
  print("Tie.")
}
```

```
## [1] "Tie."
```

```
## "Tie."
```

Motivational example: Máquina Tragaperras

Para simular estos datos necesitamos hacer dos cosas:

1. Generar combinaciones de tres elementos de entre los siguientes símbolos: diamonds (DD), sevens (7), triple bars (BBB), double bars (BB), single bars (B), cherries (C), and zeroes (0).

Cada símbolo aparece según su probabilidad en la rueda.

2. Asignar un premio a cada combinación

Las máquinas tragaperras de la marca Manitoba tienen el siguiente esquema de premios:

Combination	Prize(\$)
DD DD DD	100
7 7 7	80
BBB BBB BBB	40
BB BB BB	25
B B B	10
C C C	10
Any combination of bars	5
C C *	5
C * C	5
* C C	5
C * *	2

Combination	Prize(\$)
* C *	2
* * C	2

Cada vez que jugamos a la maquina necesitamos pagar 1 dollar. Queremos crear un programa que nos de un *score* cada vez que jugamos. Podriamos hacer nuestro programa como un script o como una función

Estrategia general en programación

Hay tres recomendaciones que pueden ayudaros a hacer programas por muy complejos que sean:

- Romper el problema en subproblemas mas pequeños
- Escribir en lenguaje natural las instrucciones que se necesitarán para cada parte
- Usar ejemplos concretos

Un programa en R tiene dos tipos de tareas: paralelas y secuenciales

Instrucciones secuenciales

Un ejemplo de instrucciones secuenciales sería la siguiente función:

```
play <- function() {
  # step 1: generate symbols
  symbols <- get_symbols()
  # step 2: display the symbols
  print(symbols)
  # step 3: score the symbols
  score(symbols)
}
```

Genera los simbolos al azar, los muestra y les da una puntuacion de acuerdo con las reglas del juego.

Vamos a escribir una función para la primera tarea:

```
get_symbols <- function() {
  wheel <- c("DD", "7", "BBB", "BB", "B", "C", "0")
  sample(wheel, size = 3, replace = TRUE,
    prob = c(0.03, 0.03, 0.06, 0.1, 0.25, 0.01, 0.52))
}
get_symbols()
```

```
## [1] "B" "BB" "0"
```

```
get_symbols()
```

```
## [1] "0" "BB" "0"
```

```
get_symbols()
```

```
## [1] "0" "B" "0"
```

Casos paralelos

Una vez que se eligen los tres símbolos hay que decidir como se puntua. Aqui tenemos que mirar varias cosas en paralelo como muestra la figura.

Cómo testamos cada una de esas opciones?

- Opción 1: ¿Son los tres iguales? Por ejemplo,

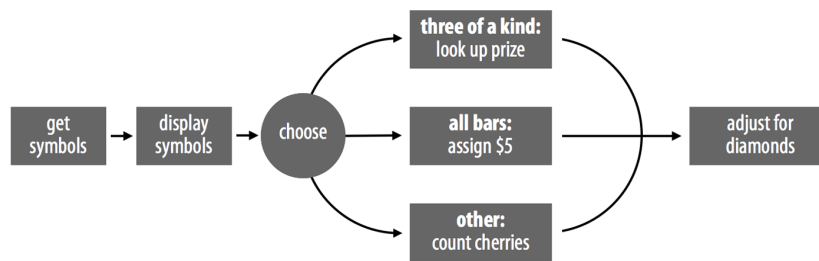


Figure 1: Esquema juego

```
symbols<-c("7","7","7")
symbols[1] == symbols[2] & symbols[2] == symbols[3]
```

```
## [1] TRUE
```

```
## TRUE
```

```
symbols[1] == symbols[2] & symbols[1] == symbols[3]
```

```
## [1] TRUE
```

```
## TRUE
```

```
all(symbols == symbols[1])
```

```
## [1] TRUE
```

```
## TRUE
```

O equivalentemente:

```
length(unique(symbols) == 1)
```

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

Y tenemos la primera opción programada.

- Opción 2: ¿tenemos todos símbolos de tipo B?

Es decir, entre nuestros símbolos tenemos alguna de estas combinaciones:

```
symbols <- c("B", "BBB", "BB")
```

```
symbols[1] == "B" | symbols[1] == "BB" | symbols[1] == "BBB" &
  symbols[2] == "B" | symbols[2] == "BB" | symbols[2] == "BBB" &
  symbols[3] == "B" | symbols[3] == "BB" | symbols[3] == "BBB"
```

```
## [1] TRUE
```

```
## TRUE
```

Pero no parece una solución muy elegante:

```
all(symbols %in% c("B", "BB", "BBB"))
```

```
## [1] TRUE
```

```
## TRUE
```

Hay overlapping entre el caso 1 y el 2: "B-B-B"

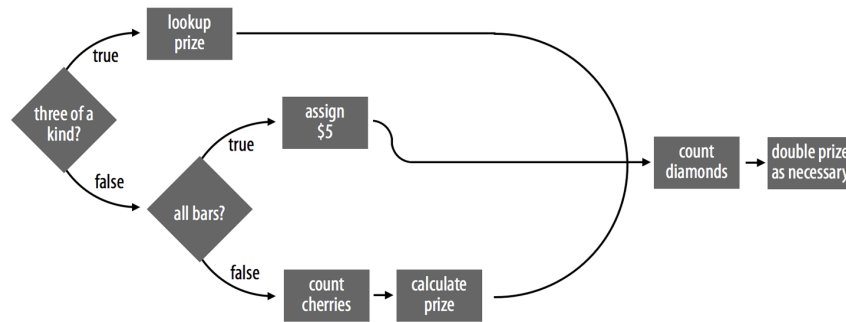


Figure 2: Esquema juego

```
symbols <- c("B", "B", "B")
all(symbols %in% c("B", "BB", "BBB"))
```

```
## [1] TRUE
```

```
## TRUE
```

Nuestro programa ya contendría:

```
same <- symbols[1] == symbols[2] && symbols[2] == symbols[3]
bars <- symbols %in% c("B", "BB", "BBB")
```

- Opción 3: Si no se da ninguna de las anteriores, cuantas cerezas (C) tenemos?

```
count.c <- sum(symbols == "C")
```

Volviendo a la función que calcula el score para cada combinación de la máquina tragaperras, tenemos ocho posibles escenarios excluyentes, que se resumen en la figura 2.

Ya teníamos como testar cada una de las opciones y ahora sabemos como unirlos con if/else. Vamos a rellenar el cuerpo de la función `score()`

```
score <- function(symbols) {
  # calculate a prize
  prize
}
```

Donde calculate de prize va a ser:

```
score <- function(x) {
  # identify case
  same <- x[1] == x[2] && x[2] == x[3]
  bars <- x %in% c("B", "BB", "BBB")

  # get prize
  if (same & x[1] != 0) {
    payouts <- c("DD" = 100, "7" = 80, "BBB" = 40, "BB" = 25,
      "B" = 10, "C" = 10, "0" = 0)
    prize <- unname(payouts[x[1]])
  } else if (all(bars)) {
    prize <- 5
  } else {
```

```

    cherries <- sum(x == "C")
    prize <- c(0, 2, 5)[cherries + 1]
  }

  # adjust for diamonds
  diamonds <- sum(x == "DD")
  prize * 2 ^ diamonds
}
score(symbols)

```

```
## [1] 10
```

Finalmente nuestro programa entero es asi:

```

play <- function() {
  symbols <- get_symbols()
  #print(symbols)
  score(symbols)
}

```

Juguemos:

```
play()
```

```
## [1] 0
```

```
play()
```

```
## [1] 0
```

```
play()
```

```
## [1] 0
```

Ejecutando programas en R

Borra todos los elementos de la sesion. Guarda el script donde has guardado la funcion play como “play.r”. Podriamos ejecutarlo haciendo

```

setwd("C:/Users/fscabo/Desktop/MasterDataScience_KSchool/Intro2R_KaggleSchool/Class2")
source("play.R")
play()

```

```
## [1] "B" "0" "B"
```

```
## [1] "No Premio"
```

```
## [1] 0
```

for and while loops

Queremos generar un vector que contenga 100 resultados de la funcion play:

Vamos a intentar ver si estamos ante una tragaperras fraudulenta. Generamos un data.frame que contenga todas las posibles combinaciones de valores que podemos obtener. Para eso usamos la función *expand.grid*

```

wheel <- c("DD", "7", "BBB", "BB", "B", "C", "0")
combos <- expand.grid(wheel, wheel, wheel, stringsAsFactors = FALSE)
View(combos)

```


Vamos a calcular la probabilidad de obtener cada una de estas combinaciones usando las probabilidades que hemos utilizado en `get_symbols`

```
prob <- c("DD" = 0.03, "7" = 0.03, "BBB" = 0.06,  
         "BB" = 0.1, "B" = 0.25, "C" = 0.01, "0" = 0.52)  
  
combos$prob1 <- prob[combos$Var1]  
combos$prob2 <- prob[combos$Var2]  
combos$prob3 <- prob[combos$Var3]  
View(combos)
```

Si las tiradas son independientes:

```
combos$prob <- combos$prob1 * combos$prob2 * combos$prob3  
sum(combos$prob)
```

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

```
View(combos)
```

Vamos a añadir una columna a nuestro data.frame `combos` que contenga el premio para cada combinación que hemos creado:

```
for (i in 1:nrow(combos)) {  
  symbols <- c(combos[i, 1], combos[i, 2], combos[i, 3])  
  combos$prize[i] <- score(symbols)  
}  
View(combos)
```

Una forma alternativa de hacerlo que se acerca mas a la forma de programar con objetos es usar las funciones de la familia `apply()`

```
score.combo<-function(x){score(x[1:3])}  
combos$prize<-apply(combos,1,score.combo)  
View(combos)
```

Ahora podemos calcular el premio esperado:

```
sum(combos$prize * combos$prob)
```

```
## [1] 0.515451
```

Por lo tanto esta máquina paga mas o menos el 54% de lo que juegas...

Paquetes en R

Ya hemos hablado acerca de una de las principales características de R: las funciones se construyen a partir del propio lenguaje y juntas conforman diferentes paquetes, que no son mas que conjuntos de funciones específicas para un cierto tema. Un paquete contiene código, documentación, ejemplos de uso y datasets.

Los paquetes de R pueden ser de uso personal o compartidos con la comunidad a través fundamentalmente de 3 tipos de repositorios:

- CRAN: Es el repositorio oficial y consiste en una red de ftp y web servers mantenidos por la comunidad de R en todo el mundo. Para que un paquete pueda ser distribuido a través de CRAN necesita pasar una serie de requerimientos.
- Bioconductor: Es una colección de paquetes específicos de análisis de datos biomédicos y también mantenidos y testados por un grupo específico de expertos.

- Github: Aqui no hay control sobre los paquetes, como en los dos anteriores. Pero hoy en día es una de las formas más comunes de compartir paquetes.

Instalar paquetes de CRAN

Para ello usamos el comando:

```
## Loading required package: sm
## Package 'sm', version 2.2-5.6: type help(sm) for summary information
## Loading required package: zoo
##
## Attaching package: 'zoo'
##
## The following objects are masked from 'package:base':
##
##      as.Date, as.Date.numeric
```

Este es el paquete más antiguo de R. Suele ser útil usar la opción *dependencies=T* porque bajará automáticamente todos los paquetes de los que el paquete de interés depende.

Se pueden instalar varios paquetes a la vez usando la notación vectorial.

Instalar paquetes de Bioconductor

Instalar paquetes via *devtools*

El paquete devtools forma parte de CRAN, por lo tanto podemos instalarlo usando:

A partir de ahí se pueden instalar paquetes de distintas fuentes usando distintas funciones:

```
install_bioc() from Bioconductor,
install_bitbucket() from Bitbucket,
install_cran() from CRAN,
install_git() from a git repository,
install_github() from GitHub,
install_local() from a local file,
install_svn() from a SVN repository,
install_url() from a URL, and
install_version() from a specific version of a CRAN package.
```

Lo usaremos mucho durante el resto del curso porque algunos de los datasets que necesitamos vienen en paquetes contribuidos pero fuera de CRAN.

En esta sesión necesitamos los siguientes paquetes:

```
## -- Attaching packages ----- tidyverse 1.2.1 --
## v ggplot2 3.2.1      v purrr   0.3.2
## v tibble  2.1.3      v dplyr  0.8.3
## v tidyr   1.0.0      v stringr 1.4.0
## v readr   1.3.1      v forcats 0.3.0
##
## -- Conflicts ----- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## Loading required package: xml2
##
## Attaching package: 'rvest'
```

```
## The following object is masked from 'package:purrr':
##
##   pluck
## The following object is masked from 'package:readr':
##
##   guess_encoding
##
## Attaching package: 'data.table'
## The following objects are masked from 'package:dplyr':
##
##   between, first, last
## The following object is masked from 'package:purrr':
##
##   transpose
```

Instalalos desde CRAN.

Importación de datos en R

En la clase anterior hemos leído archivos que estaban en datasets almacenados en librerías de R. Sin embargo, es común tener la necesidad de leer datos a partir de diferentes tipos de archivos (.txt, .csv, .xlsx) o incluso de otras fuentes como páginas web o bases de datos.

Importación de datos desde archivos

Cuando se trata de importar datos a partir de un archivo en una unidad de disco a la que tenemos acceso en local o en remoto lo primero que tenemos que hacer es identificar dicha unidad y decirle a R cuál es el lugar que queremos usar como directorio de trabajo *working_directory*. Es el lugar desde el que se leerán y en el que se escribirá todo.

```
getwd()
```

```
## [1] "C:/Users/fscabo/Desktop/MasterDataScience_KSchool/Intro2R_KaggleSchool/Class2"
```

Para cambiar al *working_directory* que queremos usar podemos utilizar la función *setwd()* o usar la GUI de RStudio en Session -> Set Working directory -> Choose Path

En esta sesión vamos a usar algunos de los datasets almacenados en el paquete dslabs de R. Irizarry en su libro Data Science Book. Vamos a copiar el archivo de datos el ejemplo murders en nuestro *working_directory*

```
dir <- system.file(package="dslabs") #extracts the location of package
dir
```

```
## [1] "C:/Users/fscabo/Documents/R/R-3.6.1/library/dslabs"
```

```
filename <- file.path(dir,"extdata/murders.csv")
file.copy(filename, "murders.csv")
```

```
## [1] FALSE
```

Ahora podemos comprobar que el archivo “murders.csv” está en nuestro *working directory*.

```
list.files()
```

Funciones standard

Se trata de un archivo con extension .csv pequeño y facil de manejar. Hay varias funciones basicas que podemos usar para ello:

```
dat <- read.csv("DataSets/murders.csv")
head(dat)
```

```
##      state abb region population total
## 1  Alabama  AL  South   4779736    135
## 2  Alaska   AK   West    710231     19
## 3  Arizona  AZ   West   6392017    232
## 4  Arkansas AR   South   2915918     93
## 5 California CA   West  37253956   1257
## 6  Colorado CO   West   5029196     65
```

Alternativamente podriamos haber utilizado la funcion *read.delim* tambien del paquete base que es una de las mas flexibles pero también que requiere mas parametrizacion:

```
dat2 <- read.delim("DataSets/murders.csv", sep=",", header=T)
head(dat2)
```

```
##      state abb region population total
## 1  Alabama  AL  South   4779736    135
## 2  Alaska   AK   West    710231     19
## 3  Arizona  AZ   West   6392017    232
## 4  Arkansas AR   South   2915918     93
## 5 California CA   West  37253956   1257
## 6  Colorado CO   West   5029196     65
```

Vamos a ver de que clase es el objeto dat

```
class(dat)
```

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

Es una buena señal, lo ha leído como data.frame. Si recordais podemos acceder cada una de las columnas del data.frame con el símbolo \$

```
head(dat$state)
```

```
## [1] Alabama  Alaska  Arizona  Arkansas  California Colorado
## 51 Levels: Alabama Alaska Arizona Arkansas California ... Wyoming
```

Sin embargo tenemos un problema porque la variable state se he volcado en R como factor y como ya hemos comentado anteriormente está desaconsejado utilizar los factores en R salvo en ocasiones puntuales dado que internamente para R por eficiencia los factores son realmente almacenados como numericos y transformaciones numerico -> string pueden ser complicadas. Por ello seria mucho mejor no leer los strings como factores. R tiene una opcion para ello:

```
dat <- read.csv("DataSets/murders.csv", stringsAsFactors = FALSE)
class(dat$state)
```

```
## [1] "character"
```

Abrid el archivo *murders.csv* y volvedlo a guardar como archivo *.xlsx*. Vamos a leerlo utilizando el paquete *library(xlsx)*

```
dat2 <- read.xlsx(file = "DataSets/murders.xlsx", sheetIndex = 1)
class(dat2)
```

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

```
head(dat2)
```

```
##      state abb region population total
## 1  Alabama AL  South    4779736    135
## 2  Alaska  AK   West     710231     19
## 3  Arizona AZ   West    6392017    232
## 4  Arkansas AR  South    2915918     93
## 5 California CA  West    37253956   1257
## 6  Colorado CO   West    5029196     65
```

```
class(dat2$state)
```

```
## [1] "factor"
```

```
identical(dat$state,dat2$state)
```

```
## [1] FALSE
```

Other function to read tabulated data into R is read.table:

```
dat3=read.table(file = "DataSets/murders.csv",header = T,sep=",")
head(dat3)
```

```
##      state abb region population total
## 1  Alabama AL  South    4779736    135
## 2  Alaska  AK   West     710231     19
## 3  Arizona AZ   West    6392017    232
## 4  Arkansas AR  South    2915918     93
## 5 California CA  West    37253956   1257
## 6  Colorado CO   West    5029196     65
```

NOTA IMPORTANTE: ASCII es un sistema de codificación que convierte los caracteres en números. ASCII usa 7 bits (0000001) por lo que se pueden generar un total de 128 símbolos. Para lenguajes con un gran número de símbolos se puede utilizar otra codificación llamada UTF para la que se pueden escoger combinaciones de 8, 16 o 24 bits. RStudio utiliza UTF-8. **NOTA IMPORTANTE 2:** Todos los archivos que hemos leído hasta ahora son archivos de texto; i.e. se pueden leer con un editor cualquiera. Archivos xls, html, json son binarios y necesitaremos otro tipo de herramientas para leerlos.

Cómo abrir grandes archivos

El paquete *data.table* contiene funciones para leer archivos de varios megas de RAM. En particular *fread()* funciona muy bien y se le pueden ajustar los parámetros: *sep*, *colClasses* and *nrows*. *bit64::integer64* types también se detectan y leer automáticamente sin necesidad de transformarlos antes en *character*. También es interesante mostrar el progreso en la lectura del archivo para tener una idea de la duración de la lectura. Otro parámetro que hereda de las funciones básicas de lectura es *stringsAsFactors*.

```
counts.rnaseq=fread("DataSets/Counts.genes.DiffAll.genes.limma.random.txt",sep = "\t",showProgress=T)
#head(counts.rnaseq)
class(counts.rnaseq)
```

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

Esta función devuelve un objeto *data.table* en lugar de un *data.frame* a no ser que se le especifique: *data.table=FALSE*. Ejemplo de tiempos (del manual de *data.table*):

```
n = 1e6
DT = data.table( a=sample(1:1000,n,replace=TRUE),
                 b=sample(1:1000,n,replace=TRUE),
                 c=rnorm(n),
```

```

        d=sample(c("foo","bar","baz","qux","quux"),n,replace=TRUE),
        e=rnorm(n),
        f=sample(1:1000,n,replace=TRUE) )
DT[2,b:=NA_integer_]
DT[4,c:=NA_real_]
DT[3,d:=NA_character_]
DT[5,d:=""]
DT[2,e:=+Inf]
DT[3,e:=-Inf]

write.table(DT,
            "DataSets/test.csv",
            sep=",",
            row.names=FALSE,
            quote=FALSE)

cat("File size (MB):",
    round(file.info("test.csv")$size/1024^2),"\n")

## File size (MB): NA

system.time(DF1 <- read.csv("DataSets/test.csv",stringsAsFactors=FALSE))

##      user  system elapsed
##    4.14    0.13    4.27

system.time(DF2 <- read.table("DataSets/test.csv",header=TRUE,sep=",",quote="",
    stringsAsFactors=FALSE,comment.char="",nrows=n,
    colClasses=c("integer","integer","numeric",
        "character","numeric","integer")))

##      user  system elapsed
##    1.16    0.09    1.25

system.time(DT <- fread("DataSets/test.csv"))

##      user  system elapsed
##    0.32    0.11    0.04

    #require(sqldf)
    #require(ff)

identical(DF1,DF2)

## [1] TRUE

all.equal(as.data.table(DF1), DT)

## [1] TRUE

```

tidyverse

Ya hemos visto como el paquete *xlsx* (y otros) contenian funciones para leer archivos excel en R. El paquete *tidyverse* también puede utilizarse para ello a traves de las funciones:

Function	Separador	Sufijo
read_table	espacio	txt

Function	Separador	Sufijo
read_csv	,	csv
read_csv2	;	csv
read_tsv	tab	tsv
read_delim	general	txt

The readxl package provides functions to read-in Microsoft Excel formats:

Function	Format	Typical suffix
read_excel	auto detect the format	xls, xlsx
read_xls	original format	xls
read_xlsx	new format	xlsx

Si os fijáis todas estas funciones son parecidas a las que teníamos en el paquete base separadas por un punto como *read.csv*. Comparemos ambas funciones:

```
dat=read.csv("DataSets/murders.csv")
class(dat)
```

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

```
head(dat)
```

```
##      state abb region population total
## 1  Alabama AL  South   4779736    135
## 2   Alaska AK   West    710231     19
## 3  Arizona AZ   West   6392017    232
## 4  Arkansas AR  South   2915918     93
## 5 California CA  West   37253956   1257
## 6   Colorado CO  West    5029196     65
```

```
dat2=read_csv("DataSets/murders.csv")
class(dat2)
```

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

```
head(dat2)
```

```
## # A tibble: 6 x 5
##   state     abb region population total
##   <chr>    <chr> <chr>      <dbl> <dbl>
## 1 Alabama  AL    South    4779736    135
## 2 Alaska   AK    West     710231     19
## 3 Arizona  AZ    West    6392017    232
## 4 Arkansas AR    South    2915918     93
## 5 California CA    West    37253956   1257
## 6 Colorado CO    West     5029196     65
```

OMG: What's that? *Tibbles* son nuevos tipos de data.frames que contienen muchas de las buenas prácticas que hemos ido mencionando. Por ejemplo:

- Tibble nunca cambia el tipo de una variable (si es character, es character, no factor...)

```
tibble(x = letters)
```

```
## # A tibble: 26 x 1
```

```
##      x
##      <chr>
## 1 a
## 2 b
## 3 c
## 4 d
## 5 e
## 6 f
## 7 g
## 8 h
## 9 i
## 10 j
## # ... with 16 more rows
tibble(x = 1:3, y = list(1:5, 1:10, 1:20))
```

```
## # A tibble: 3 x 2
##       x y
##   <int> <list>
## 1     1 <int [5]>
## 2     2 <int [10]>
## 3     3 <int [20]>
```

- Tibble nunca cambia el nombre de las variables

```
names(tibble("Old Alphabet" = letters))

## [1] "Old Alphabet"
names(data.frame("Old Alphabet" = letters))

## [1] "Old.Alphabet"
```

- Evalua los argumentos secuencialmente

```
#data.frame(z=1:5,y=z ^ 2)
tibble(x = 1:5, y = x ^ 2)
```

```
## # A tibble: 5 x 2
##       x     y
##   <int> <dbl>
## 1     1     1
## 2     2     4
## 3     3     9
## 4     4    16
## 5     5    25
```

- El objetivo fundamental de tibble es almacenar las variables de una forma consistente, por lo que no hace uso de row.names
- Printing: Por defecto solo se escriben en pantalla las 10 primeras filas y todas las columnas que quepan. Se puede cambiar el tipo de letra, color, etc. `tibble.print_max = Inf` muestra todas las filas

```
tibble(x = 1:5, y = x ^ 2)

## # A tibble: 5 x 2
##       x     y
##   <int> <dbl>
## 1     1     1
## 2     2     4
```



```
## 3      3      9
## 4      4     16
## 5      5     25
```

```
data.frame(x=1:5,y=(1:5)^2)
```

```
##   x  y
## 1 1  1
## 2 2  4
## 3 3  9
## 4 4 16
## 5 5 25
```

- Subsetting siempre devuelve otro objeto tibble, sea de las dimensiones que sea

```
df1 <- data.frame(x = 1:3, y = 3:1)
class(df1[, 1:2])
```

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

```
class(df1[, 1])
```

```
## [1] "integer"
```

```
df2 <- tibble(x = 1:3, y = 3:1)
class(df2[, 1:2])
```

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

```
class(df2[, 1])
```

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

Si una columna no tiene exactamente el nombre por el que se la llama no se hace partial matching:

```
df <- data.frame(abc = 1)
df$a
```

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

```
df2 <- tibble(abc = 1)
df2$a
```

```
## Warning: Unknown or uninitialised column: 'a'.
```

```
## NULL
```

Tidy data

La organización Gapminder trata de desenmascarar falsos mitos acerca del estado del mundo en terminos de pobreza, desigualdad, etc a través del uso de datos. Utilizando el dataset *gapminder* del paquete *dslabs* trataremos de contestar a dos preguntas: * Es cierto que el mundo se divide en paises occidentales ricos y no-occidentales pobres? * Han aumentado las diferencias entre paises en los últimos 40 años?

1. Descargamos los datos:

```
library(dslabs)
data(gapminder)
#si no tienes el paquete instalado
#gapminder=read.csv("DataSets/Gapminder.csv")
```

2. Exploramos los datos:

```
View(gapminder)
head(gapminder)
```

```
##           country year infant_mortality life_expectancy fertility
## 1      Albania 1960      115.40           62.87         6.19
## 2      Algeria 1960      148.20           47.50         7.65
## 3       Angola 1960      208.00           35.98         7.32
## 4 Antigua and Barbuda 1960      NA           62.97         4.43
## 5      Argentina 1960      59.87           65.39         3.11
## 6      Armenia 1960      NA           66.86         4.55
##  population      gdp continent      region
## 1    1636054      NA     Europe Southern Europe
## 2   11124892 13828152297     Africa Northern Africa
## 3    5270844      NA     Africa  Middle Africa
## 4     54681      NA  Americas     Caribbean
## 5   20619075 108322326649  Americas  South America
## 6    1867396      NA      Asia   Western Asia
```

```
str(gapminder)
```

```
## 'data.frame':  10545 obs. of  9 variables:
## $ country      : Factor w/ 185 levels "Albania","Algeria",...: 1 2 3 4 5 6 7 8 9 10 ...
## $ year         : int  1960 1960 1960 1960 1960 1960 1960 1960 1960 1960 ...
## $ infant_mortality: num  115.4 148.2 208 NA 59.9 ...
## $ life_expectancy : num  62.9 47.5 36 63 65.4 ...
## $ fertility      : num  6.19 7.65 7.32 4.43 3.11 4.55 4.82 3.45 2.7 5.57 ...
## $ population     : num  1636054 11124892 5270844 54681 20619075 ...
## $ gdp            : num  NA 1.38e+10 NA NA 1.08e+11 ...
## $ continent      : Factor w/ 5 levels "Africa","Americas",...: 4 1 1 2 2 3 2 5 4 3 ...
## $ region         : Factor w/ 22 levels "Australia and New Zealand",...: 19 11 10 2 15 21 2 1 22 21
```

3. Para trabajar con el universo tidyverse necesitamos que los datos esten en formato “tidy”

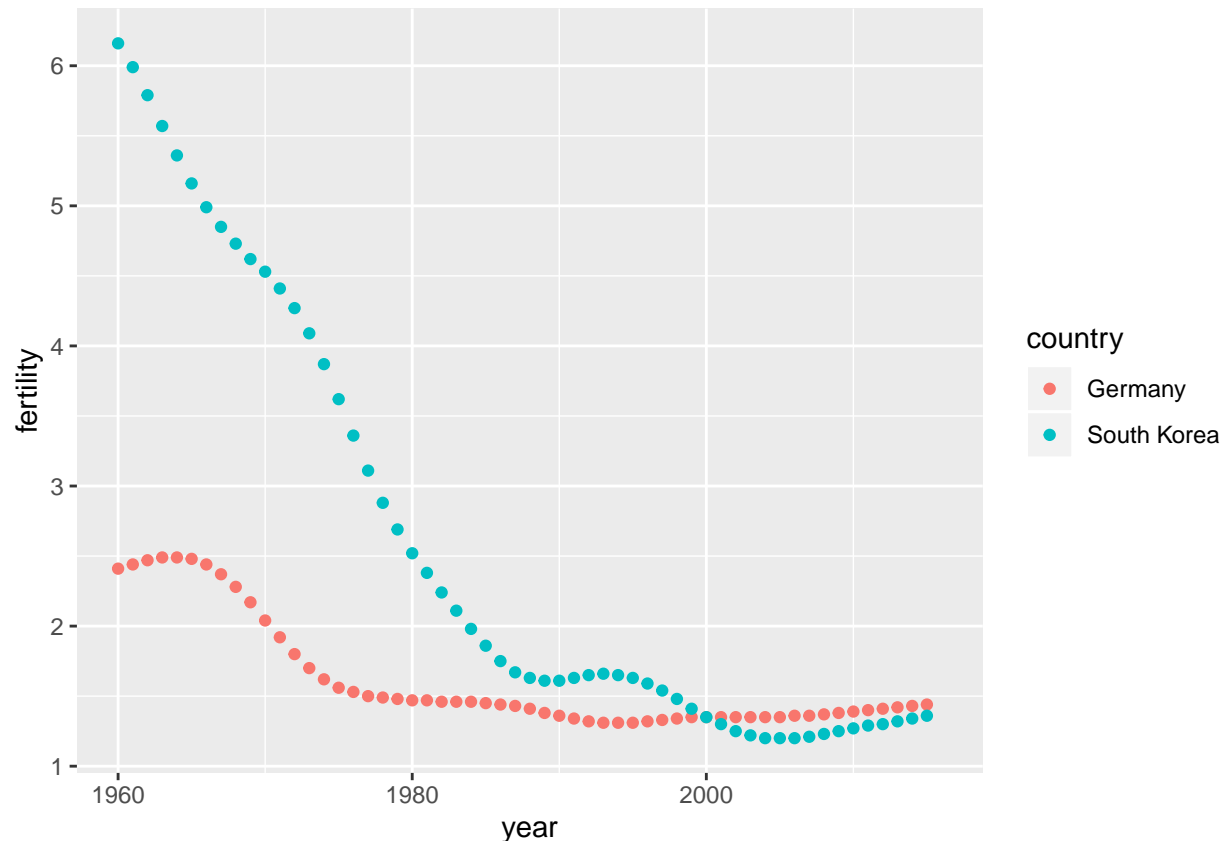
```
tidy_data <- gapminder %>%
  filter(country %in% c("South Korea", "Germany")) %>%
  select(country, year, fertility)
```

```
head(tidy_data)
```

```
##           country year fertility
## 1      Germany 1960      2.41
## 2 South Korea 1960      6.16
## 3      Germany 1961      2.44
## 4 South Korea 1961      5.99
## 5      Germany 1962      2.47
## 6 South Korea 1962      5.79
```

```
tidy_data %>%
  ggplot(aes(year, fertility, color = country)) +
  geom_point()
```

```
## Warning: Removed 2 rows containing missing values (geom_point).
```



Este plot ha sido tan fácil de producir porque los datos estaban en formato *tidy*. Que significa esto? Que cada medida específica de país & fecha ocupa una fila de un *tibble* o *data.frame*. Las columnas serían cada una de las observaciones que queramos mirar (fertilidad, esperanza de vida...)

```
path <- system.file("extdata", package="dslabs")
filename <- file.path(path, "fertility-two-countries-example.csv")
wide_data <- read_csv(filename)
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   country = col_character()
## )
## See spec(...) for full column specifications.
```

```
wide_data <- read_csv("DataSets/fertility-two-countries-example.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   country = col_character()
## )
## See spec(...) for full column specifications.
```

```
wide_data
```

```
## # A tibble: 2 x 57
##   country `1960` `1961` `1962` `1963` `1964` `1965` `1966` `1967` `1968`
```

```
##      <chr>      <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 Germany      2.41  2.44  2.47  2.49  2.49  2.48  2.44  2.37  2.28
## 2 South ~      6.16  5.99  5.79  5.57  5.36  5.16  4.99  4.85  4.73
## # ... with 47 more variables: `1969` <dbl>, `1970` <dbl>, `1971` <dbl>,
## #   `1972` <dbl>, `1973` <dbl>, `1974` <dbl>, `1975` <dbl>, `1976` <dbl>,
## #   `1977` <dbl>, `1978` <dbl>, `1979` <dbl>, `1980` <dbl>, `1981` <dbl>,
## #   `1982` <dbl>, `1983` <dbl>, `1984` <dbl>, `1985` <dbl>, `1986` <dbl>,
## #   `1987` <dbl>, `1988` <dbl>, `1989` <dbl>, `1990` <dbl>, `1991` <dbl>,
## #   `1992` <dbl>, `1993` <dbl>, `1994` <dbl>, `1995` <dbl>, `1996` <dbl>,
## #   `1997` <dbl>, `1998` <dbl>, `1999` <dbl>, `2000` <dbl>, `2001` <dbl>,
## #   `2002` <dbl>, `2003` <dbl>, `2004` <dbl>, `2005` <dbl>, `2006` <dbl>,
## #   `2007` <dbl>, `2008` <dbl>, `2009` <dbl>, `2010` <dbl>, `2011` <dbl>,
## #   `2012` <dbl>, `2013` <dbl>, `2014` <dbl>, `2015` <dbl>
```

```
#seleccionamos las primeras 9 columnas
select(wide_data, country, `1960`:`1967`)
```

```
## # A tibble: 2 x 9
##   country      `1960` `1961` `1962` `1963` `1964` `1965` `1966` `1967`
##   <chr>         <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 Germany          2.41  2.44  2.47  2.49  2.49  2.48  2.44  2.37
## 2 South Korea      6.16  5.99  5.79  5.57  5.36  5.16  4.99  4.85
```

5. Convertir wide_data en tidy_data

```
new_tidy_data <- wide_data %>%
  pivot_longer(cols=`1960`:`2015`,names_to="year",values_to = "fertility")
new_tidy_data
```

```
## # A tibble: 112 x 3
##   country year  fertility
##   <chr>   <chr>      <dbl>
## 1 Germany 1960        2.41
## 2 Germany 1961        2.44
## 3 Germany 1962        2.47
## 4 Germany 1963        2.49
## 5 Germany 1964        2.49
## 6 Germany 1965        2.48
## 7 Germany 1966        2.44
## 8 Germany 1967        2.37
## 9 Germany 1968        2.28
## 10 Germany 1969        2.17
## # ... with 102 more rows
```

```
class(new_tidy_data$year)
```

```
## [1] "character"
```

```
new_tidy_data <- wide_data %>%
  pivot_longer(cols=`1960`:`2015`,
    names_to="year",
    values_to = "fertility",
    names_ptypes = list(year=integer()))
class(new_tidy_data$year)
```

```
## [1] "integer"
```

A veces se necesita volver de tidy data a wide data:

```
new_wide_data <- new_tidy_data %>%
  pivot_wider(names_from=year, values_from = fertility)
new_wide_data
```

```
## # A tibble: 2 x 57
##   country `1960` `1961` `1962` `1963` `1964` `1965` `1966` `1967` `1968`
##   <chr>    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 Germany  2.41  2.44  2.47  2.49  2.49  2.48  2.44  2.37  2.28
## 2 South ~  6.16  5.99  5.79  5.57  5.36  5.16  4.99  4.85  4.73
## # ... with 47 more variables: `1969` <dbl>, `1970` <dbl>, `1971` <dbl>,
## # `1972` <dbl>, `1973` <dbl>, `1974` <dbl>, `1975` <dbl>, `1976` <dbl>,
## # `1977` <dbl>, `1978` <dbl>, `1979` <dbl>, `1980` <dbl>, `1981` <dbl>,
## # `1982` <dbl>, `1983` <dbl>, `1984` <dbl>, `1985` <dbl>, `1986` <dbl>,
## # `1987` <dbl>, `1988` <dbl>, `1989` <dbl>, `1990` <dbl>, `1991` <dbl>,
## # `1992` <dbl>, `1993` <dbl>, `1994` <dbl>, `1995` <dbl>, `1996` <dbl>,
## # `1997` <dbl>, `1998` <dbl>, `1999` <dbl>, `2000` <dbl>, `2001` <dbl>,
## # `2002` <dbl>, `2003` <dbl>, `2004` <dbl>, `2005` <dbl>, `2006` <dbl>,
## # `2007` <dbl>, `2008` <dbl>, `2009` <dbl>, `2010` <dbl>, `2011` <dbl>,
## # `2012` <dbl>, `2013` <dbl>, `2014` <dbl>, `2015` <dbl>
```

Importando datos de bases de datos

Datos estructurados: RMySQL -> RMariaDB

MariaDB es un servidor de bases de datos. Podemos interaccionar con el usando un cliente o bien programáticamente usando APIs (Application Programming Interface). Si interaccionamos a través de un cliente como mysql empezariamos conectandonos a la base de datos con el comando:

```
mysql -u root -p -h localhost
```

Existe un paquete de R para hacer lo mismo y traernos datos:

```
library(RMariaDB)
```

```
## Warning: package 'RMariaDB' was built under R version 3.6.2
```

En primer lugar creamos una objeto de conexión a la base de datos y podemos ver todas las bases de datos que hay dentro de ella:

```
ensembl_con <- dbConnect(MariaDB(),
  host = "ensembl.db.ensembl.org",
  user = "anonymous",
  port = 5306,
  password = "")
dbGetQuery(ensembl_con, "SHOW DATABASES")
```

Podemos ahora trabajar para encontrar las tablas que contengan el string “compara”

```
dbGetQuery(ensembl_con, "SHOW DATABASES") %>%
  # filter the table for cases where the the string "compara" is present
  filter(str_detect(Database, "compara")) %>%
  tail()
```

```
##           Database
## 47 ensembl_compara_94
## 48 ensembl_compara_95
## 49 ensembl_compara_96
## 50 ensembl_compara_97
```

```
## 51 ensembl_compara_98
## 52 ensembl_compara_99
```

Nos quedamos con la mas reciente:

```
compara_con <- dbConnect(MariaDB(),
                          dbname = "ensembl_compara_99",
                          host = "ensembldb.ensembl.org",
                          user = "anonymous",
                          port = 5306,
                          password = "")
```

Para ver las tablas que hay en la base de datos que hemos elegido:

```
library(dbplyr)
```

```
## Warning: package 'dbplyr' was built under R version 3.6.2
```

```
##
```

```
## Attaching package: 'dbplyr'
```

```
## The following objects are masked from 'package:dplyr':
```

```
##
```

```
## ident, sql
```

```
src_dbi(compara_con)
```

```
## src: mysql [anonymous@ensembldb.ensembl.org:NA/ensembl_compara_99]
## tbls: CAFE_gene_family, CAFE_species_gene, conservation_score,
## constrained_element, dnafrag, dnafrag_region, exon_boundaries,
## external_db, family, family_member, gene_align, gene_align_member,
## gene_member, gene_member_hom_stats, gene_member_qc, gene_tree_node,
## gene_tree_node_attr, gene_tree_node_tag, gene_tree_object_store,
## gene_tree_root, gene_tree_root_attr, gene_tree_root_tag, genome_db,
## genomic_align, genomic_align_block, genomic_align_tree, hmm_annot,
## hmm_curated_annot, hmm_profile, homology, homology_member,
## mapping_session, member_xref, meta, method_link,
## method_link_species_set, method_link_species_set_attr,
## method_link_species_set_tag, ncbi_taxa_name, ncbi_taxa_node,
## other_member_sequence, peptide_align_feature, seq_member,
## seq_member_projection, seq_member_projection_stable_id, sequence,
## species_set, species_set_header, species_set_tag, species_tree_node,
## species_tree_node_attr, species_tree_node_tag, species_tree_root,
## stable_id_history, syntenic_region
```

Accedemos a la tabla "homology"

```
compara_homology <- tbl(compara_con, "homology")
head(compara_homology)
```

```
## # Source:   lazy query [?? x 15]
## # Database: mysql [anonymous@ensembldb.ensembl.org:NA/ensembl_compara_99]
##   homology_id method_link_spe~ description is_tree_complia~   dn   ds
##           <int>           <int> <chr>           <int> <dbl> <dbl>
## 1             1             132551 other_para~         1 1.02  16.6
## 2             2             132551 other_para~         1 1.07   7.82
## 3             3             132551 other_para~         1 1.07  66.7
## 4             4             132551 other_para~         1 1.20  70.1
## 5             5             132551 other_para~         1 1.04  71.9
```

```
## 6          6          132551 other_para~          1 0.914 77.6
## # ... with 9 more variables: n <dbl>, s <dbl>, lnl <dbl>,
## #   species_tree_node_id <int>, gene_tree_node_id <int>,
## #   gene_tree_root_id <int>, goc_score <int>, wga_coverage <dbl>,
## #   is_high_confidence <int>

compara_linksp <- tbl(compara_con, "method_link_species_set")
head(compara_linksp)

## # Source:   lazy query [?? x 8]
## # Database: mysql [anonymous@ensembl.db.ensembl.org:NA/ensembl_compara_99]
##   method_link_spe~ method_link_id species_set_id name source url
##           <int>           <int>           <int> <chr> <chr> <chr>
## 1             555             16             34144 G.ac~ ensem~ ""
## 2             569             16             34350 C.in~ ensem~ ""
## 3             587             16             34701 M.mu~ ensem~ ""
## 4             592             16             34706 M.mu~ ensem~ ""
## 5             594             16             34708 C.fa~ ensem~ ""
## 6             596             16             34710 A.ca~ ensem~ ""
## # ... with 2 more variables: first_release <int>, last_release <int>

sapiens_ortho <- compara_linksp %>%
  filter(name == "H.sap-P.tro orthologues" | name == "H.sap-M.mus orthologues")

sapiens_ortho

## # Source:   lazy query [?? x 8]
## # Database: mysql [anonymous@ensembl.db.ensembl.org:NA/ensembl_compara_99]
##   method_link_spe~ method_link_id species_set_id name source url
##           <int>           <int>           <int> <chr> <chr> <chr>
## 1             50976             201             35673 H.sa~ ensem~ ""
## 2             103890             201             39100 H.sa~ ensem~ ""
## # ... with 2 more variables: first_release <int>, last_release <int>

compara_result <- sapiens_ortho %>%
  left_join(compara_homology, by = "method_link_species_set_id") %>%
  collect()

head(compara_result)

## # A tibble: 6 x 22
##   method_link_spe~ method_link_id species_set_id name source url
##           <int>           <int>           <int> <chr> <chr> <chr>
## 1             50976             201             35673 H.sa~ ensem~ ""
## 2             50976             201             35673 H.sa~ ensem~ ""
## 3             50976             201             35673 H.sa~ ensem~ ""
## 4             50976             201             35673 H.sa~ ensem~ ""
## 5             50976             201             35673 H.sa~ ensem~ ""
## 6             50976             201             35673 H.sa~ ensem~ ""
## # ... with 16 more variables: first_release <int>, last_release <int>,
## #   homology_id <int>, description <chr>, is_tree_compliant <int>,
## #   dn <dbl>, ds <dbl>, n <dbl>, s <dbl>, lnl <dbl>,
## #   species_tree_node_id <int>, gene_tree_node_id <int>,
## #   gene_tree_root_id <int>, goc_score <int>, wga_coverage <dbl>,
## #   is_high_confidence <int>
```

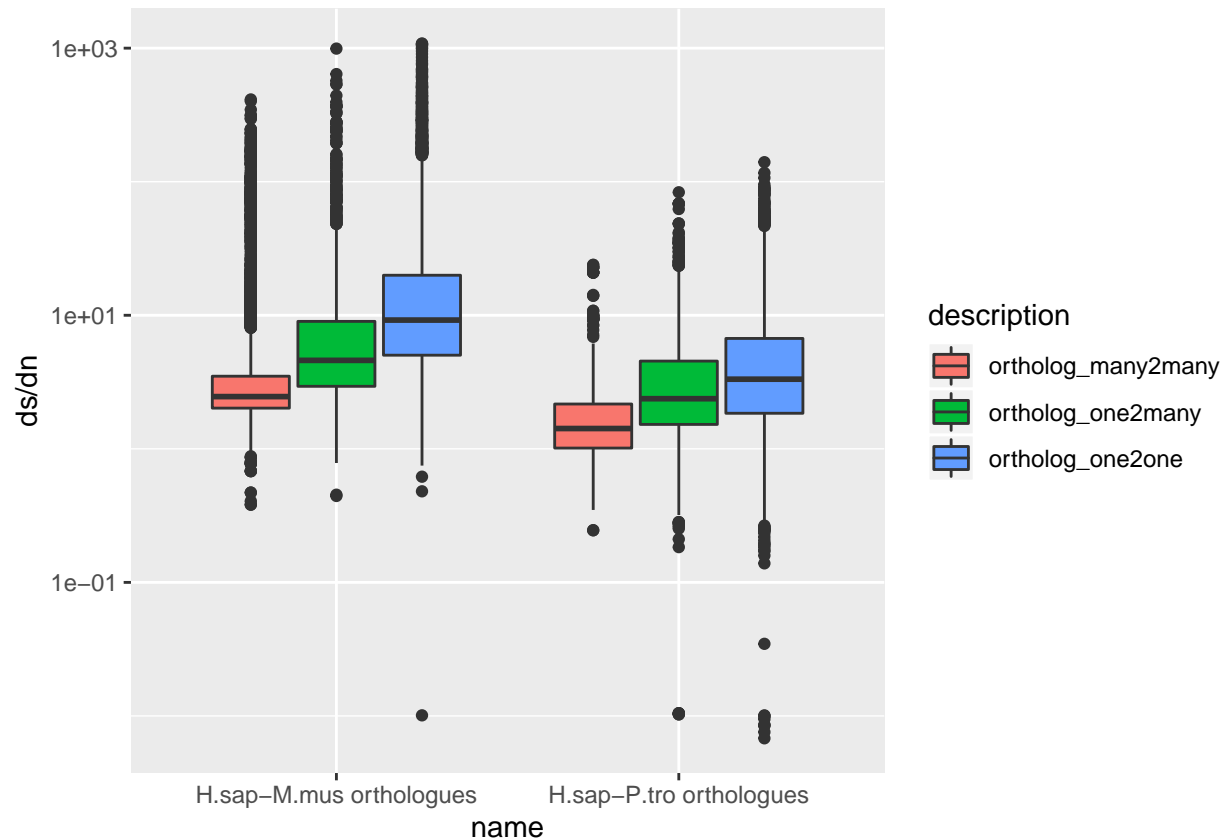
```
compara_result %>%
  count(name, description)
```

```
## # A tibble: 6 x 3
##   name                description      n
##   <chr>              <chr>      <int>
## 1 H.sap-M.mus orthologues ortholog_many2many 5773
## 2 H.sap-M.mus orthologues ortholog_one2many 3450
## 3 H.sap-M.mus orthologues ortholog_one2one 16534
## 4 H.sap-P.tro orthologues ortholog_many2many 556
## 5 H.sap-P.tro orthologues ortholog_one2many 3214
## 6 H.sap-P.tro orthologues ortholog_one2one 19410
```

```
compara_result %>%
  ggplot(aes(name, ds/dn, fill = description)) +
  geom_boxplot() + scale_y_log10()
```

```
## Warning: Transformation introduced infinite values in continuous y-axis
```

```
## Warning: Removed 10524 rows containing non-finite values (stat_boxplot).
```



Datos estructurados: APIs y paquetes dedicados

La base de datos del Worl Bank contiene una API desde la que podemos acceder a los World Data Indicators (<http://datatopics.worldbank.org/world-development-indicators/>). El paquete de R “WDI” baja los datos que le pidamos de esa base de datos, parsea el XML correspondiente y formatea el objeto resultante. Además contiene funciones para hacer búsquedas en las bases de datos:


```
library(WDI)

WDIsearch(string = "life.*expectancy", field = "name", cache = NULL)

df.le = WDI(country = "all", indicator = c("SP.DYN.LE00.IN"), start = 1900,
            end = 2012)
head(df.le)
is.data.frame(df.le)
levels(factor(df.le$country))
levels(factor(df.le$year))
```

Datos no estructurados:

Datos de un link de internet

El paquete dslab está en github, por lo que podriamos descargarnos los datos del fichero “murders.csv” directamente de alli.

```
url <- "https://raw.githubusercontent.com/rafalab/dslabs/master/inst/extdata/murders.csv"
```

Y, aún mejor, read_csv puede leerlo directamente

```
dat <- read_csv(url)
```

Podemos no solo leer los datos en R sino bajarnos el fichero a una unidad de disco o de red utilizando R:

```
download.file(url, "murders.csv")
```

Web scratching

Queremos comparar las estadísticas de asesinato en EEUU con las estadísticas de europa, pero el dato de eurostat no esta desglosado y ese número no es comparable. En wikipedia buscamos las estadísticas de muertes no casuales a nivel mundial (CTRL+U para ver el código de la web)

```
url="https://en.wikipedia.org/wiki/List_of_countries_by_intentional_homicide_rate"
h <- read_html(url)
class(h)
```

```
## [1] "xml_document" "xml_node"
```

```
h
```

```
## {html_document}
## <html class="client-nojs" lang="en" dir="ltr">
## [1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset= ...
## [2] <body class="mediawiki ltr sitedir-ltr mw-hide-empty-elt ns-0 ns-sub ...
```

Hay una seccion del codigo html que comienza con

. Ahi estan los datos de la tabla que queremos leer en R.

```
tab <- h %>% html_nodes("table")
tab <- tab[[4]] %>% html_table
head(tab)
```

```
## Country (or dependent territory,subnational area, etc.) Region
## 1 Burundi Africa
## 2 Comoros Africa
## 3 Djibouti Africa
## 4 Eritrea Africa
```

```
## 5 Ethiopia Africa
## 6 Kenya Africa
## Subregion Rate Count Yearlisted Source
## 1 Eastern Africa 6.02 635 2016 CTS/SDG
## 2 Eastern Africa 7.70 60 2015 WHO Estimate
## 3 Eastern Africa 6.48 60 2015 WHO Estimate
## 4 Eastern Africa 8.04 390 2015 WHO Estimate
## 5 Eastern Africa 7.56 7,552 2015 WHO Estimate
## 6 Eastern Africa 5.00 2,466 2017 CTS
```

```
class(tab)
```

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

```
tab <- tab %>%
  select(starts_with("Country"),
         Region, Count, Rate, starts_with("Year")) %>%
  setNames(c("country", "continent", "total", "murder_rate", "year"))
```

```
head(tab)
```

```
## country continent total murder_rate year
## 1 Burundi Africa 635 6.02 2016
## 2 Comoros Africa 60 7.70 2015
## 3 Djibouti Africa 60 6.48 2015
## 4 Eritrea Africa 390 8.04 2015
## 5 Ethiopia Africa 7,552 7.56 2015
## 6 Kenya Africa 2,466 5.00 2017
```

```
### Import from JSON
```

```
library(jsonlite)
```

```
##
```

```
## Attaching package: 'jsonlite'
```

```
## The following object is masked from 'package:purrr':
```

```
##
```

```
## flatten
```

```
citi_bike <- fromJSON("http://citibikenyc.com/stations/json")
citi_bike$executionTime
```

```
## [1] "2020-02-08 02:01:27 AM"
```

```
head(citi_bike$stationBeanList)
```

```
## id stationName availableDocks totalDocks latitude
## 1 359 E 47 St & Park Ave 63 64 40.75510
## 2 72 W 52 St & 11 Ave 14 55 40.76727
## 3 79 Franklin St & W Broadway 28 33 40.71912
## 4 82 St James Pl & Pearl St 4 27 40.71117
## 5 83 Atlantic Ave & Fort Greene Pl 10 62 40.68383
## 6 116 W 17 St & 8 Ave 33 50 40.74178
## longitude statusValue statusKey availableBikes
## 1 -73.97499 In Service 1 0
## 2 -73.99393 In Service 1 40
## 3 -74.00667 In Service 1 4
## 4 -74.00017 In Service 1 23
```

```

## 5 -73.97632 In Service      1          51
## 6 -74.00150 In Service      1          15
##           stAddress1 stAddress2 city postalCode location
## 1           E 47 St & Park Ave
## 2           W 52 St & 11 Ave
## 3      Franklin St & W Broadway
## 4           St James Pl & Pearl St
## 5 Atlantic Ave & Fort Greene Pl
## 6           W 17 St & 8 Ave
## altitude testStation lastCommunicationTime landMark
## 1           FALSE 2020-02-08 02:00:48 AM
## 2           FALSE 2020-02-08 02:00:47 AM
## 3           FALSE 2020-02-08 02:01:07 AM
## 4           FALSE 2020-02-08 01:58:46 AM
## 5           FALSE 2020-02-08 02:01:26 AM
## 6           FALSE 2020-02-08 02:00:44 AM

```