Introducción a R (II)

FSC

Contents

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/

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 especificas 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 traves 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 coleccion de paquetes especificos 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 dia 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 ahi 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.

Entornos en R (environment)

Para seguir avanzando es importante que entendamos cómo R guarda los objetos. En realidad funciona de una forma parecida al sistema de ficheros de un ordenador. Cada objeto se guarda dentro de un environment (o entorno) que a su vez puede estar dentro de otro, con una estructura jerárquica. Vamos a ver la estructura de nuestro entorno en esta sesion:

```
library(pryr)
```

```
## Registered S3 method overwritten by 'pryr':
##   method      from
##   print.bytes Rcpp
```

```
##
## Attaching package: 'pryr'
```

```
## The following objects are masked from 'package:purrr':
##
##     compose, partial
```

```
## The following object is masked from 'package:data.table':
##
##     address
```

```
parenvs(all = TRUE)
```

```
##    label                               name
## 1  <environment: R_GlobalEnv>          ""
## 2  <environment: package:pryr>         "package:pryr"
## 3  <environment: package:vioplot>      "package:vioplot"
## 4  <environment: package:zoo>          "package:zoo"
## 5  <environment: package:sm>           "package:sm"
## 6  <environment: package:rvest>        "package:rvest"
## 7  <environment: package:xml2>         "package:xml2"
## 8  <environment: package:usethis>      "package:usethis"
## 9  <environment: package:devtools>     "package:devtools"
## 10 <environment: package:dslabs>       "package:dslabs"
## 11 <environment: package:forcats>      "package:forcats"
## 12 <environment: package:stringr>      "package:stringr"
## 13 <environment: package:purrr>        "package:purrr"
## 14 <environment: package:tibble>       "package:tibble"
## 15 <environment: package:ggplot2>      "package:ggplot2"
## 16 <environment: package:tidyverse>    "package:tidyverse"
## 17 <environment: package:data.table>   "package:data.table"
## 18 <environment: package:tidyr>        "package:tidyr"
## 19 <environment: package:readr>        "package:readr"
## 20 <environment: package:xlsx>         "package:xlsx"
## 21 <environment: package:dplyr>        "package:dplyr"
## 22 <environment: package:stats>        "package:stats"
## 23 <environment: package:graphics>     "package:graphics"
## 24 <environment: package:grDevices>    "package:grDevices"
## 25 <environment: package:utils>        "package:utils"
## 26 <environment: package:datasets>     "package:datasets"
## 27 <environment: package:methods>      "package:methods"
## 28 <environment: 0x0000000013069598>   "Autoloads"
## 29 <environment: base>                 ""
## 30 <environment: R_EmptyEnv>           ""
```

R_EmptyEnv es el único entorno que no tiene ningún padre. Los entornos de R no se almacenan físicamente en tu ordenador, están en memoria RAM.

Para explorar la jerarquía de nuestro entorno tenemos algunas funciones:

```
as.environment("package:stats")
```

```
## <environment: package:stats>
## attr(,"name")
## [1] "package:stats"
## attr(,"path")
## [1] "C:/Users/fscabo/Documents/R/R-3.6.1/library/stats"
```

```
globalenv()
```

```
## <environment: R_GlobalEnv>
```

```
baseenv()
```

```
## <environment: base>
```

```
emptyenv()
```

```
## <environment: R_EmptyEnv>
```

```
parent.env(globalenv())
```

```
## <environment: package:pryr>
## attr(,"name")
## [1] "package:pryr"
## attr(,"path")
## [1] "C:/Users/fscabo/Documents/R/R-3.6.1/library/pryr"
```

Como habíamos intuído al principio, el entorno global depende del paquete *usethis()*. Y como ya habiamos comentado podemos confirmar que el único entorno sin padres es emptyEnv.

```
#parent.env(emptyenv())
```

Los objetos guardados en un entorno pueden ser vistos con:

```
ls(emptyenv())
```

```
## character(0)
```

```
ls(globalenv())
```

```
## character(0)
```

En cualquier momento podemos saber cual es el entorno con el que está trabajando R:

```
environment()
```

```
## <environment: R_GlobalEnv>
```

```
new<-"Hello Global"
```

Si guardamos algo en ese objeto new R lo sobreescribira:

```
new <- "Hello Active"

new
```

```
## [1] "Hello Active"
```

```
## "Hello Active"
```

Algunas funciones guardan objetos temporales.

```r
roll <- function() {
  die <- 1:6
  dice <- sample(die, size = 2, replace = TRUE)
  sum(dice)
}
```

Para no sobreescribir un posible objeto que se llame "dice", R crea un nuevo entorno cada vez que se evalua una funcion. De ahi las dependencias que hemos visto al principio.Al finalizar la evaluación, R vuelve al entorno global:

```r
show_env <- function(){
  list(ran.in = environment(),
    parent = parent.env(environment()),
    objects = ls.str(environment()))
}
show_env()
```

```
## $ran.in
## <environment: 0x0000000018519db8>
##
## $parent
## <environment: R_GlobalEnv>
##
## $objects
```

R ha creado un nuevo *runtime environment* durante el tiempo que ha estado ejecutando la función.

Programando en R

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
* 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] "BBB" "0"   "B"
```

```
get_symbols()
```

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

```
get_symbols()
```

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

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,

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

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

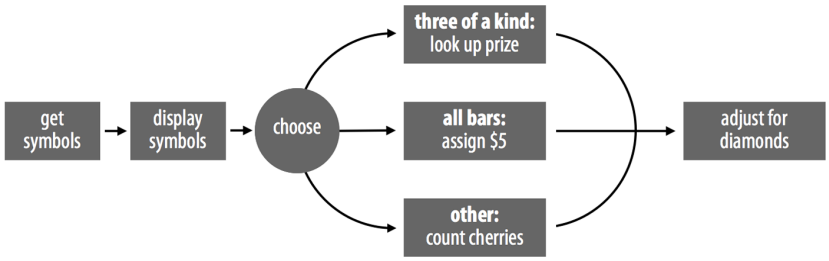


Figure 1: Esquema juego

```
## 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.

Nota: && y || funcionan como sólo uno excepto porque no evaluan la segunda condicion si la primera es clara

- 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: “BBB”

```
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")
```

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 condución 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:

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

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

```
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 lineas 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."
```

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 teniamos como testar cada una de las opciones y ahora sabemos como unirlas con if/else. Vamos a rellenar el cuerpo de la funcion *score()*

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

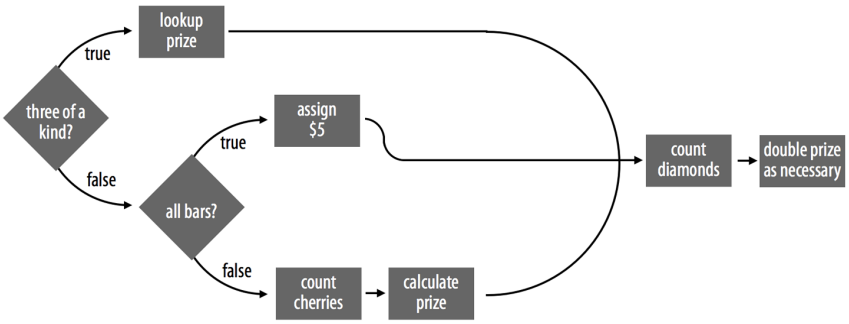


Figure 2: Esquema juego

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" "0" "0"
```

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

```
play()
```

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

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

```
play()
```

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

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

for and while loops

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]
head(combos, 3)
```

```
##   Var1 Var2 Var3 prob1 prob2 prob3
## 1   DD   DD   DD  0.03  0.03  0.03
## 2    7   DD   DD  0.03  0.03  0.03
## 3  BBB   DD   DD  0.06  0.03  0.03
```

Si las tiradas son independientes:

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

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

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)
}
head(combos)
```

```
##   Var1 Var2 Var3 prob1 prob2 prob3     prob prize
## 1   DD   DD   DD  0.03  0.03  0.03 0.000027   800
## 2    7   DD   DD  0.03  0.03  0.03 0.000027     0
## 3  BBB   DD   DD  0.06  0.03  0.03 0.000054     0
## 4   BB   DD   DD  0.10  0.03  0.03 0.000090     0
## 5    B   DD   DD  0.25  0.03  0.03 0.000225     0
## 6    C   DD   DD  0.01  0.03  0.03 0.000009     8
```

Ahora podemos calcular el premio esperado:

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

```
## [1] 0.538014
```

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

Importación de datos en R

En la clase anterior hemos leido archivos que estaban en datasets almacenados en librerias de R. Sin embargo, es comun tener la necesidad de leer datos a partir de diferentes tipos de archivos (.txt, .cvs, .xlsx) o incluso de otras fuentes como paginas web o bases de datos.

Importacion 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 cual es el lugar que queremos usar como directorio de trabajo *working_directory*. Es el lugar desde el que se leeran y en el que se escribira todo.

```
getwd()
```

```
## [1] "C:/Users/fscabo/Desktop/MasterDataScience_KSchool/Session2_ImportExport_v2"
```

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

En esta sesion 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 podeis comprobar que el archivo "murders.csv" esta en vuestro 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 leido como data.frame. Si recordais podemos acceder cada una de las columnas del data.frame con el símbolo $

```
dat$state
```

```
##  [1] Alabama              Alaska               Arizona
##  [4] Arkansas             California           Colorado
##  [7] Connecticut          Delaware             District of Columbia
## [10] Florida              Georgia              Hawaii
## [13] Idaho                Illinois             Indiana
## [16] Iowa                 Kansas               Kentucky
## [19] Louisiana            Maine                Maryland
## [22] Massachusetts        Michigan             Minnesota
## [25] Mississippi          Missouri             Montana
## [28] Nebraska             Nevada               New Hampshire
## [31] New Jersey           New Mexico           New York
## [34] North Carolina       North Dakota         Ohio
## [37] Oklahoma             Oregon               Pennsylvania
## [40] Rhode Island         South Carolina       South Dakota
## [43] Tennessee            Texas                Utah
## [46] Vermont              Virginia             Washington
## [49] West Virginia        Wisconsin            Wyoming
## 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"
```

Utilizamos una de las funciones fundamentales del paquete *dplyr* para volver a poner la variable *state* como un factor por medio de:

```
dat <- mutate(dat, region =  as.factor(region))
```

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
```

```
identical(as.matrix(dat$state),as.matrix(dat2$state))
```

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

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 codificacion que convierte los caracteres en números. ASCII usa 7 bits (0000001) por lo que se pueden generar un total de 128 símbolos. Para lenguages 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 leido 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 caracter.También es interesante mostrar el progreso en la lectura del archivo para tener una idea de la duracion 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
##    4.13    0.20    4.38
```

```
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
##    1.20    0.06    1.28
```

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

```
##    user  system elapsed
##    0.22    0.08    0.07
```

```
  #require(sqldf)
  #require(ff)


identical(DF1,DF2)
```

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

```r
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
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:

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

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

```r
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
```

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

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

```r
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 caracter, es caracter, 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

```
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
```

- Recycling La primera columna de longitud > 1 es la que determina el tamaño del tibble.

```r
tibble(a = 1, b = 1:3)
```

```
## # A tibble: 3 x 2
##       a     b
##   <dbl> <int>
## 1     1     1
## 2     1     2
## 3     1     3
```

```r
tibble(a = 1:3, b = 1)
```

```
## # A tibble: 3 x 2
##       a     b
##   <int> <dbl>
## 1     1     1
## 2     2     1
## 3     3     1
```

```r
#tibble(a = 1:3, c = 1:2)
tibble(a = 1, b = integer())
```

```
## # A tibble: 0 x 2
## # ... with 2 variables: a <dbl>, b <int>
```

```r
tibble(a = integer(), b = 1)
```

```
## # A tibble: 0 x 2
## # ... with 2 variables: a <int>, b <dbl>
```

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:

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

2. Exploramos los datos:

```r
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. De las siguientes parejas, cual dirias que tuvo la mayor mortalidad infantil en 2015? ++ Sri Lanka or Turkey ++ Poland or South Korea ++ Malaysia or Russia ++ Pakistan or Vietnam ++ Thailand or South Africa Ahora, trata de responderlo usando los datos:

```
library(dplyr)
gapminder %>%
  filter(year == 2015 & country %in% c("Sri Lanka","Turkey")) %>%
  select(country, infant_mortality)
```

```
##     country infant_mortality
## 1 Sri Lanka              8.4
## 2    Turkey             11.6
```

```
gapminder %>%
  filter(year == 2015 & country %in% c("Poland","South Korea")) %>%
  select(country, infant_mortality)
```

```
##       country infant_mortality
## 1 South Korea              2.9
## 2      Poland              4.5
```

```
gapminder %>%
  filter(year == 2015 & country %in% c("Malaysia","Russia")) %>%
  select(country, infant_mortality)
```

```
##    country infant_mortality
## 1 Malaysia              6.0
## 2   Russia              8.2
```

```
gapminder %>%
  filter(year == 2015 & country %in% c("Pakistan","Vietnam")) %>%
  select(country, infant_mortality)
```

```
##    country infant_mortality
## 1 Pakistan             65.8
## 2  Vietnam             17.3
```

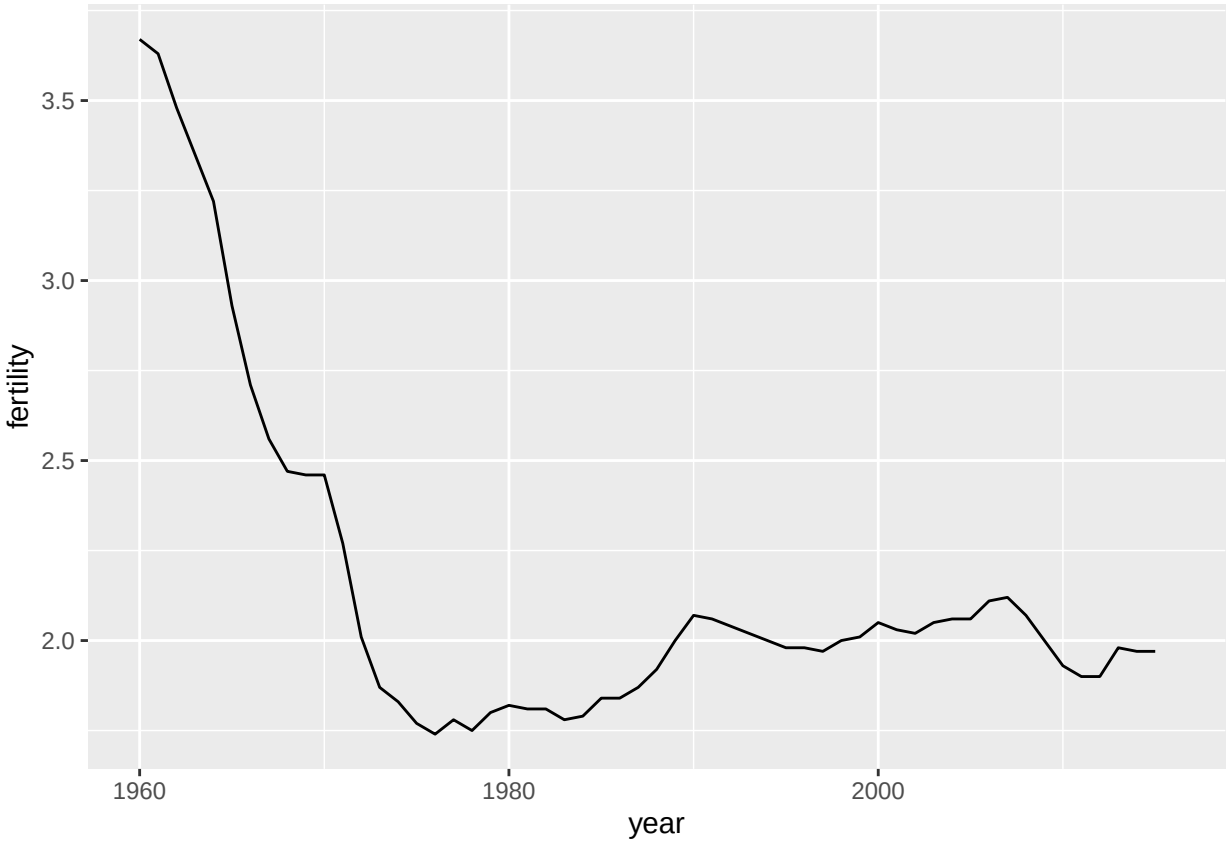
```
gapminder %>%
  filter(year == 2015 & country %in% c("Thailand","South Africa")) %>%
  select(country, infant_mortality)
```

```
##         country infant_mortality
## 1 South Africa              33.6
## 2     Thailand              10.5
```

```
library(ggplot2)
years <- c(1962, 1980, 1990, 2000, 2012)
continents <- c("Europe", "Asia")
gapminder %>%
  filter(year %in% years & continent %in% continents) %>%
  ggplot( aes(fertility, life_expectancy, col = continent)) +
  geom_point() +
  facet_wrap(~year)
```


```
gapminder %>%
  filter(country == "United States") %>%
  ggplot(aes(year,fertility)) +
  geom_line()
```

```
## Warning: Removed 1 rows containing missing values (geom_path).
```



4. Centrémonos en comparar Alemania y South Korea

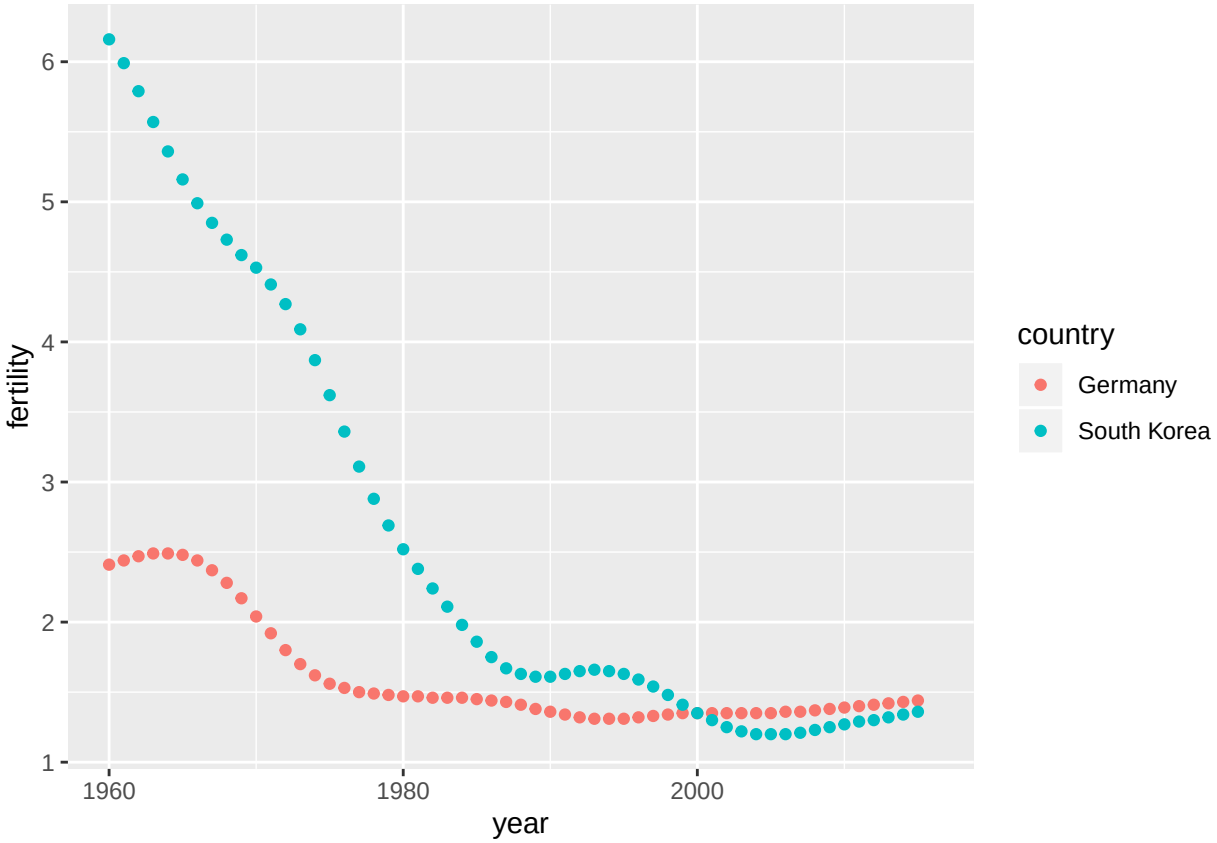
```
data("gapminder")
tidy_data <- gapminder %>%
  filter(country %in% c("South Korea", "Germany")) %>%
  select(country, year, fertility)

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 especifica de pais & fecha ocupa una fila de un *tibble*. Las columnas serian 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 %>%
  gather(year, fertility, `1960`:`2015`)
new_tidy_data
```

```
## # A tibble: 112 x 3
##    country     year  fertility
##    <chr>       <chr>     <dbl>
##  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
##  7 Germany     1963       2.49
##  8 South Korea 1963       5.57
##  9 Germany     1964       2.49
## 10 South Korea 1964       5.36
## # ... with 102 more rows
```

```
new_tidy_data <- wide_data %>%
  gather(year, fertility, -country)
class(tidy_data$year)
```

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

```
#> [1] "integer"
class(new_tidy_data$year)
```

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

```
new_tidy_data <- wide_data %>%
  gather(year, fertility, -country, convert = TRUE)
class(new_tidy_data$year)
```

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

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

```
new_wide_data <- new_tidy_data %>% spread(year, fertility)
select(new_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
```

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

raw_dat <- read_csv(filename)
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   country = col_character()
## )
```

```
## See spec(...) for full column specifications.
```

```
select(raw_dat,1:5)
```

```
## # A tibble: 2 x 5
##   country `1960_fertility` `1960_life_expe~ `1961_fertility`
##   <chr>              <dbl>            <dbl>            <dbl>
## 1 Germany             2.41             69.3             2.44
## 2 South ~             6.16             53.0             5.99
## # ... with 1 more variable: `1961_life_expectancy` <dbl>
```

```
# lo transformamos en data tidy
dat <- raw_dat %>% gather(key, value, -country)
head(dat)
```

```
## # A tibble: 6 x 3
##   country     key                  value
##   <chr>       <chr>                <dbl>
## 1 Germany     1960_fertility        2.41
## 2 South Korea 1960_fertility        6.16
## 3 Germany     1960_life_expectancy 69.3
## 4 South Korea 1960_life_expectancy 53.0
## 5 Germany     1961_fertility        2.44
## 6 South Korea 1961_fertility        5.99
```

```
# pero tenemos dos observaciones (año&variable) en cada fila
head(dat %>% separate(key, c("year", "variable_name"), "_"))
```

```
## Warning: Expected 2 pieces. Additional pieces discarded in 112 rows [3, 4,
## 7, 8, 11, 12, 15, 16, 19, 20, 23, 24, 27, 28, 31, 32, 35, 36, 39, 40, ...].
```

```
## # A tibble: 6 x 4
##   country     year  variable_name value
##   <chr>       <chr> <chr>         <dbl>
## 1 Germany     1960  fertility      2.41
```

```
## 2 South Korea 1960  fertility      6.16
## 3 Germany     1960  life          69.3
## 4 South Korea 1960  life          53.0
## 5 Germany     1961  fertility      2.44
## 6 South Korea 1961  fertility      5.99
```

```
#"_" es el separador por defecto y por tanto no hace falta
head(dat %>% separate(key, c("year", "variable_name")))
```

```
## Warning: Expected 2 pieces. Additional pieces discarded in 112 rows [3, 4,
## 7, 8, 11, 12, 15, 16, 19, 20, 23, 24, 27, 28, 31, 32, 35, 36, 39, 40, ...].
```

```
## # A tibble: 6 x 4
##   country     year  variable_name value
##   <chr>       <chr> <chr>         <dbl>
## 1 Germany     1960  fertility      2.41
## 2 South Korea 1960  fertility      6.16
## 3 Germany     1960  life          69.3
## 4 South Korea 1960  life          53.0
## 5 Germany     1961  fertility      2.44
## 6 South Korea 1961  fertility      5.99
```

```
head(dat %>% separate(key, c("year", "first_variable_name", "second_variable_name"),
                fill = "right"))
```

```
## # A tibble: 6 x 5
##   country     year  first_variable_name second_variable_name value
##   <chr>       <chr> <chr>               <chr>                <dbl>
## 1 Germany     1960  fertility           <NA>                  2.41
## 2 South Korea 1960  fertility           <NA>                  6.16
## 3 Germany     1960  life                expectancy           69.3
## 4 South Korea 1960  life                expectancy           53.0
## 5 Germany     1961  fertility           <NA>                  2.44
## 6 South Korea 1961  fertility           <NA>                  5.99
```

```
head(dat %>% separate(key, c("year", "variable_name"), extra = "merge"))
```

```
## # A tibble: 6 x 4
##   country     year  variable_name   value
##   <chr>       <chr> <chr>           <dbl>
## 1 Germany     1960  fertility        2.41
## 2 South Korea 1960  fertility        6.16
## 3 Germany     1960  life_expectancy 69.3
## 4 South Korea 1960  life_expectancy 53.0
## 5 Germany     1961  fertility        2.44
## 6 South Korea 1961  fertility        5.99
```

```
#pero queremos crear una columna para cada variable
dat %>%
  separate(key, c("year", "variable_name"), extra = "merge") %>%
  spread(variable_name, value)
```

```
## # A tibble: 112 x 4
##    country year  fertility life_expectancy
##    <chr>   <chr>     <dbl>           <dbl>
##  1 Germany 1960       2.41            69.3
##  2 Germany 1961       2.44            69.8
```

```
##  3 Germany 1962       2.47           70.0
##  4 Germany 1963       2.49           70.1
##  5 Germany 1964       2.49           70.7
##  6 Germany 1965       2.48           70.6
##  7 Germany 1966       2.44           70.8
##  8 Germany 1967       2.37           71.0
##  9 Germany 1968       2.28           70.6
## 10 Germany 1969       2.17           70.5
## # ... with 102 more rows
```

Importando 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 estadisticas de asesinato en EEUU con las estadisticas de europa, pero el dato de eurostat no esta desglosado y ese número no es comparable. En wikipedia buscamos las estadisticas 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] "2020-01-09 03:56:45 AM"
```

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

```
##    id                  stationName availableDocks totalDocks latitude
## 1 534       Water - Whitehall Plaza              8         31 40.70255
## 2  72              W 52 St & 11 Ave             10         55 40.76727
## 3  79      Franklin St & W Broadway             16         33 40.71912
## 4  82        St James Pl & Pearl St              4         27 40.71117
## 5  83 Atlantic Ave & Fort Greene Pl             32         62 40.68383
## 6 116               W 17 St & 8 Ave             47         50 40.74178
##   longitude statusValue statusKey availableBikes
## 1 -74.01272  In Service         1             22
## 2 -73.99393  In Service         1             44
## 3 -74.00667  In Service         1             16
## 4 -74.00017  In Service         1             23
## 5 -73.97632  In Service         1             30
## 6 -74.00150  In Service         1              3
##                      stAddress1 stAddress2 city postalCode location
```

```
## 1       Water - Whitehall Plaza
## 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 2020-01-09 03:56:08 AM
## 2                FALSE 2020-01-09 03:54:47 AM
## 3                FALSE 2020-01-09 03:54:27 AM
## 4                FALSE 2020-01-09 03:53:15 AM
## 5                FALSE 2020-01-09 03:56:37 AM
## 6                FALSE 2020-01-09 03:53:06 AM
```