R para Demografía

Pedro L. Luque

10/2/2022

Índice

1 Introducción a R y RStudio

Este material supone que ya se tienen conocimientos básicos del uso del lenguaje R y del entorno de desarrollo RStudio.

De todas formas, a continuación se recogen algunos enlaces que resultarán útiles:

- Instalación de R, RStudio y LaTeX:
  - Mi web personal: "Instalación de R, RStudio y LaTeX": contiene explicaciones, vídeos y enlaces, para instalar estos programas en los distintos sistemas operativos: Windows, Mac y Linux. Además, incluye algunos consejos adicionales, como paquetes R recomendados, etc.
  - RStudio Cloud: Este servicio de RStudio permite trabajar con R y RStudio en la nube. En febrero de 2020, todavía es posible registrarse de forma gratuita para poder trabajar (desde cualquier navegador web con una conexión a internet).
- Libro bookdown: "R para Ciencia de Datos"
- Libro bookdown: "R Markdown: The Definitive Guide"
- Libro bookdown: "YaRrr! The Pirate's Guide to R"

2 El sistema tidyverse: dplyr, tidyr, etc

En primer lugar, se cargarán los datos almacenados en un fichero "RData" (formato binario de R), que contiene datos de la población de las provincias de España obtenidas en el Censo de 2001.

```
load("datosPobEspCenso2001.RData",verbose = TRUE)
```

```
## Loading objects:
##   datos
```

El fichero "datosPobEspCenso2001.RData" contiene el objeto R del tipo tibble (una mejora del objeto data.frame) con los datos.

```
str(datos)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    52 obs. of  5 variables:
##  $ Provincia: chr  "01-Álava" "02-Albacete" "03-Alicante/Alacant" "04-Almería" ...
##  $ CCAA     : chr  "País Vasco" "Castilla-La Mancha" "Comunidad Valenciana" "Andalucía" ...
##  $ TOTAL    : num  286387 364835 1461925 536731 1062998 ...
##  $ Varon    : num  142036 181461 722162 272023 508995 ...
##  $ Mujer    : num  144351 183374 739763 264708 554003 ...
```

Antes de seguir, se cargará el sistema "tidyverse", que a su vez carga una serie de paquetes asociados a este sistema. En este manual nos centraremos principalmente en: "dplyr" y "ggplot2".

```
library(tidyverse)
#library(tidyverse,warn.conflicts = FALSE)
```

```
── Attaching packages ─────────────────────── tidyverse 1.3.0 ──
✓ ggplot2 3.2.1     ✓ purrr   0.3.3
✓ tibble  2.1.3     ✓ dplyr   0.8.3
✓ tidyr   1.0.0     ✓ stringr 1.4.0
✓ readr   1.3.1     ✓ forcats 0.4.0
── Conflicts ────────────────────────── tidyverse_conflicts() ──
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
```

En R, al escribir el nombre del objeto que contiene los datos, nos mostrará sus valores. Aunque al ser, un objeto de tipo "tibble", mostrará solamente una representación corta de ellos.

Veamos en este caso el contenido del objeto "datos" que contiene información sobre la población de las provincias de España en el Censo de 2001.

```
datos
```

```
## # A tibble: 52 x 5
##    Provincia           CCAA                      TOTAL   Varon   Mujer
##    <chr>               <chr>                     <dbl>   <dbl>   <dbl>
##  1 01-Álava            País Vasco               286387  142036  144351
##  2 02-Albacete         Castilla-La Mancha       364835  181461  183374
##  3 03-Alicante/Alacant Comunidad Valenciana    1461925  722162  739763
##  4 04-Almería          Andalucía                 536731  272023  264708
##  5 33-Asturias         Asturias (Principado de) 1062998  508995  554003
##  6 05-Ávila            Castilla y León           163442   81850   81592
##  7 06-Badajoz          Extremadura               654882  323541  331341
##  8 07-Balears (Illes)  Balears (Illes)           841669  417314  424355
```

```
##  9 08-Barcelona        Cataluña                4805927 2341592 2464335
## 10 09-Burgos           Castilla y León          348934  174576  174358
## # ... with 42 more rows
```

2.1 Los 4 principios del tidyverse

Los 4 principios del sistema tidyverse son los siguientes:

- **Principio 1**: se usarán datos convenientemente organizados.
  - Cada fila o línea de los datos es una observación.
  - Cada columna es una variable.

Fecha	Nombre	Mate	Ingles
1-11-2015	Hernandez, Rodrigo	90	60

mes	año	primer	apellido	materia	puntos
11	2015	Rodrigo	Hernandez	mate	90
11	2015	Rodrigo	Hernandez	ingles	60

- **Principio 2**: en cada paso se usa una función o herramienta.

  Si se quisiera obtener: "la media actual de cada estudiante en la asignatura Matemáticas", los pasos que se necesitan realizar sobre los datos son:

  1. *Filtrado.* Nos quedamos con únicamente las calificaciones de la asignatura Matemáticas.
  2. *Agrupación.* Agrupamos para cada apellidos-nombre (alumno) diferente.
  3. *Cálculo.* Para cada alumno calculamos la media de sus notas de Matemáticas.
  4. *Mostrar resultados.* Se mostrarían para cada alumno sus respectivas medias.

- **Principio 3**: uso del operador tubería **"%>%"** para combinar las herramientas o funciones a utilizar. Facilitará la lectura y la modificación del código.
- **Principio 4**: cada paso es una consulta o un comando.

```
datos %>%
  filter(materia == "mate") %>%
  group_by(apellido,primer) %>%
  summarise(media = mean(puntos))
```

Nota: la información en "datos" no verifica los principios del sistema tidyverse (en una fila no hay un solo valor por provincia), pero como veremos con ejemplos a continuación, aunque no se cumpla lo recomendado, también es posible aplicar las herramientas del paquete tidyverse a datos que no cumplen sus principios.

2.2 Uso de dplyr

La librería o paquete "dplyr" contiene una serie de herramientas que facilitarán la manipulación básica de los datos, además de usar el operador "tubería": **%>%** (del paquete "magrittr") que permite encadenar varias herramientas de forma consecutiva, que como se verá más adelante, simplificará su lectura. La mayoría de estas herramientas tiene sus equivalentes en el sistema base de R o en otros paquetes R, pero se están convirtiendo en un estándar.

Las principales herramientas que contiene dplyr son las siguientes:

- `select()`: para seleccionar determinadas columnas o variables.
- `filter()`: para seleccionar determinadas filas o individuos-observaciones.
- `arrange()`: para ordenar los datos por determinadas columnas.
- `mutate()`: para construir nuevas columnas o variables.
- `summarise()`: para obtener columnas de resumen estadístico.
- `group_by()`: para agrupar las filas o individuos-observaciones por los valores de determinadas columnas.

Existen otras herramientas básicas como: `slice()` (elige filas por posición), `rename()`, `pull()` (convierte columna a un vector), `sample_n()`, `sample_frac()`, `glimpse()` (para presentar en consola), ...

También existen herramientas que permiten realizar operaciones más avanzadas que ayudan a relacionar varios "data.frame": `inner_join(x, y)`, `left_join(x, y)`, `right_join(x, y)`, `semi_join(x, y)`, `anti_join(x, y)`, ...

Aprenderemos el uso básico de "dplyr" con ayuda de ejemplos.

R muestra de forma no muy amigable los datos (como se ha visto anteriormente), por ello se va a usar la función `kable()` del paquete "knitr" para mejorar su presentación.

```
library(knitr)
kable(datos,booktabs = TRUE)
```

Provincia	CCAA	TOTAL	Varon	Mujer
01-Álava	País Vasco	286387	142036	144351
02-Albacete	Castilla-La Mancha	364835	181461	183374
03-Alicante/Alacant	Comunidad Valenciana	1461925	722162	739763
04-Almería	Andalucía	536731	272023	264708
33-Asturias	Asturias (Principado de)	1062998	508995	554003
05-Ávila	Castilla y León	163442	81850	81592
06-Badajoz	Extremadura	654882	323541	331341
07-Balears (Illes)	Balears (Illes)	841669	417314	424355
08-Barcelona	Cataluña	4805927	2341592	2464335
09-Burgos	Castilla y León	348934	174576	174358
10-Cáceres	Extremadura	403621	200820	202801
11-Cádiz	Andalucía	1116491	552463	564028
39-Cantabria	Cantabria	535131	260586	274545
12-Castellón/Castelló	Comunidad Valenciana	484566	240673	243893
51-Ceuta	Ceuta	71505	35949	35556
13-Ciudad Real	Castilla-La Mancha	478957	235189	243768
14-Córdoba	Andalucía	761657	372464	389193
15-Coruña (A)	Galicia	1096027	525388	570639
16-Cuenca	Castilla-La Mancha	200346	99959	100387
17-Girona	Cataluña	565304	280830	284474
18-Granada	Andalucía	821660	401638	420022
19-Guadalajara	Castilla-La Mancha	174999	88535	86464
20-Guipúzcoa	País Vasco	673563	330288	343275
21-Huelva	Andalucía	462579	229013	233566
22-Huesca	Aragón	206502	104089	102413
23-Jaén	Andalucía	643820	317343	326477
24-León	Castilla y León	488751	238139	250612
25-Lleida	Cataluña	362206	180425	181781
27-Lugo	Galicia	357648	173339	184309
28-Madrid	Madrid (Comunidad de)	5423384	2609746	2813638
29-Málaga	Andalucía	1287017	630902	656115

Provincia	CCAA	TOTAL	Varon	Mujer
52-Melilla	Melilla	66411	33134	33277
30-Murcia	Murcia (Región de)	1197646	597265	600381
31-Navarra	Navarra (Comunidad Foral de)	555829	276629	279200
32-Ourense	Galicia	338446	161968	176478
34-Palencia	Castilla y León	174143	85955	88188
35-Palmas (Las)	Canarias	887676	444761	442915
36-Pontevedra	Galicia	903759	433683	470076
26-Rioja (La)	Rioja (La)	276702	137827	138875
37-Salamanca	Castilla y León	345609	167948	177661
38-Santa Cruz de Tenerife	Canarias	806801	398205	408596
40-Segovia	Castilla y León	147694	73973	73721
41-Sevilla	Andalucía	1727603	846220	881383
42-Soria	Castilla y León	90717	45443	45274
43-Tarragona	Cataluña	609673	303684	305989
44-Teruel	Aragón	135858	68724	67134
45-Toledo	Castilla-La Mancha	541379	270406	270973
46-Valencia/València	Comunidad Valenciana	2216285	1084149	1132136
47-Valladolid	Castilla y León	498094	243999	254095
48-Vizcaya	País Vasco	1122637	545557	577080
49-Zamora	Castilla y León	199090	97991	101099
50-Zaragoza	Aragón	861855	422033	439822

Nota: con la función `head(datos)` se mostrarían únicamente las primeras 6 filas de los datos. Pero también se podrían mostrar en número diferente, por ejemplo, para mostrar las 10 primeras filas, se escribiría: `head(datos,10)`.

2.2.1 Ordenación de los datos

La función `arrange()` del paquete "dplyr" nos va a permitir ordenar los datos por una o más columnas. Veamos varios ejemplos.

En el siguiente ejemplo, ordenamos los datos en orden ascendente del valor en la columna o variable "TOTAL" (población total de la provincia) y mostramos únicamente las 10 primeras filas:

```
head(arrange(datos, TOTAL),10)
```

```
## # A tibble: 10 x 5
##    Provincia      CCAA                TOTAL Varon  Mujer
##    <chr>          <chr>               <dbl> <dbl>  <dbl>
##  1 52-Melilla     Melilla             66411 33134  33277
##  2 51-Ceuta       Ceuta               71505 35949  35556
##  3 42-Soria       Castilla y León     90717 45443  45274
##  4 44-Teruel      Aragón             135858 68724  67134
##  5 40-Segovia     Castilla y León    147694 73973  73721
##  6 05-Ávila       Castilla y León    163442 81850  81592
##  7 34-Palencia    Castilla y León    174143 85955  88188
##  8 19-Guadalajara Castilla-La Mancha 174999 88535  86464
##  9 49-Zamora      Castilla y León    199090 97991 101099
## 10 16-Cuenca      Castilla-La Mancha 200346 99959 100387
```

Puede verse que la menos poblada es Melilla con 66.411 habitantes, seguida de Ceuta, Soria, Teruel y Segovia.

Ese mismo objetivo se puede conseguir con ayuda del operador tubería (o pipe) `%>%` para encadenar las mismas operaciones de forma consecutiva, como se puede ver en el siguiente código:

```r
# Se podría escribir así: datos %>% arrange(TOTAL) %>% head(10)
datos %>%
  arrange(TOTAL) %>%
  head(10)
```

```
## # A tibble: 10 x 5
##    Provincia      CCAA                TOTAL Varon  Mujer
##    <chr>          <chr>               <dbl> <dbl>  <dbl>
##  1 52-Melilla     Melilla             66411 33134  33277
##  2 51-Ceuta       Ceuta               71505 35949  35556
##  3 42-Soria       Castilla y León     90717 45443  45274
##  4 44-Teruel      Aragón             135858 68724  67134
##  5 40-Segovia     Castilla y León    147694 73973  73721
##  6 05-Ávila       Castilla y León    163442 81850  81592
##  7 34-Palencia    Castilla y León    174143 85955  88188
##  8 19-Guadalajara Castilla-La Mancha 174999 88535  86464
##  9 49-Zamora      Castilla y León    199090 97991 101099
## 10 16-Cuenca      Castilla-La Mancha 200346 99959 100387
```

Se puede leer del siguiente modo: **"a datos se aplica una ordenación según la columna TOTAL y a continuación al resultado se aplica que se muestren únicamente las 10 primeras filas"**.

En RStudio, para insertar "%>%" en el editor o en la consola, se puede utilizar la **combinación de teclas: Ctrl+May+M (en Mac: Cmd+May+M)**.

Lo que hace realmente el operador tubería es colocar el elemento resultante de lo que está a su izquierda como primer argumento de la función que tiene a su derecha.

```r
x %>% f(y)         es lo mismo que    f(x,y)
y %>% f(x, ., z)   es lo mismo que    f(x,y,z)
```

El ejemplo, anterior sería equivalente a:

```r
elemento01 = arrange(datos,TOTAL)
elemento02 = head(elemento01,10)
elemento02
```

```
## # A tibble: 10 x 5
##    Provincia      CCAA                TOTAL Varon  Mujer
##    <chr>          <chr>               <dbl> <dbl>  <dbl>
##  1 52-Melilla     Melilla             66411 33134  33277
##  2 51-Ceuta       Ceuta               71505 35949  35556
##  3 42-Soria       Castilla y León     90717 45443  45274
##  4 44-Teruel      Aragón             135858 68724  67134
##  5 40-Segovia     Castilla y León    147694 73973  73721
##  6 05-Ávila       Castilla y León    163442 81850  81592
##  7 34-Palencia    Castilla y León    174143 85955  88188
##  8 19-Guadalajara Castilla-La Mancha 174999 88535  86464
##  9 49-Zamora      Castilla y León    199090 97991 101099
## 10 16-Cuenca      Castilla-La Mancha 200346 99959 100387
```

Ejemplo. Se quieren presentar los datos ordenados por **CCAA** (comunidad autónoma) y en caso de empate ordene de forma **descendente** según **TOTAL**.

```r
datos %>%
  arrange(CCAA,desc(TOTAL)) %>%
  head(15) %>%
  kable(booktabs = TRUE)
```

Provincia	CCAA	TOTAL	Varon	Mujer
41-Sevilla	Andalucía	1727603	846220	881383
29-Málaga	Andalucía	1287017	630902	656115
11-Cádiz	Andalucía	1116491	552463	564028
18-Granada	Andalucía	821660	401638	420022
14-Córdoba	Andalucía	761657	372464	389193
23-Jaén	Andalucía	643820	317343	326477
04-Almería	Andalucía	536731	272023	264708
21-Huelva	Andalucía	462579	229013	233566
50-Zaragoza	Aragón	861855	422033	439822
22-Huesca	Aragón	206502	104089	102413
44-Teruel	Aragón	135858	68724	67134
33-Asturias	Asturias (Principado de)	1062998	508995	554003
07-Balears (Illes)	Balears (Illes)	841669	417314	424355
35-Palmas (Las)	Canarias	887676	444761	442915
38-Santa Cruz de Tenerife	Canarias	806801	398205	408596

La función `arrange()` puede ordenar por más de 2 columnas, y el uso de la función `desc()` sobre "TOTAL" (podría usarse en las columnas que se necesite) ha establecido que la ordenación sea de forma descendente según la columna indicada.

Nota importante. El operador "`%>%`" se puede utilizar también con cualquier función del sistema base de R u otra librería.

2.2.2 Selección de variables: select()

Ahora se usará `select()` para quedarse con determinadas columnas.

Ejemplo. Se quiere trabajar únicamente con los datos: nombres de provincia y población total.

```
datos_s01 = datos %>%
  select(Provincia,TOTAL) # se pueden usar posiciones de columna: select(1,3:4)
head(datos_s01)
```

```
## # A tibble: 6 x 2
##   Provincia             TOTAL
##   <chr>                 <dbl>
## 1 01-Álava             286387
## 2 02-Albacete          364835
## 3 03-Alicante/Alacant 1461925
## 4 04-Almería           536731
## 5 33-Asturias         1062998
## 6 05-Ávila             163442
```

Nota: Con `select` también se podrían reordenar las columnas, apareciendo en el nuevo objeto con las columnas colocadas en el orden en el que se han enumerado en la llamada a la función.

Ejemplo. Se quieren obtener las columnas CCAA, Provincia y Población de Mujeres, ordenadas por Población de Mujeres de forma descendente (de mayor a menor).

```
datos %>%
  select(CCAA,Provincia,Mujer) %>%
  arrange(desc(Mujer)) %>%
  head()
```

```
## # A tibble: 6 x 3
##   CCAA                  Provincia              Mujer
##   <chr>                 <chr>                  <dbl>
## 1 Madrid (Comunidad de) 28-Madrid            2813638
## 2 Cataluña              08-Barcelona         2464335
## 3 Comunidad Valenciana  46-Valencia/València 1132136
## 4 Andalucía             41-Sevilla            881383
## 5 Comunidad Valenciana  03-Alicante/Alacant   739763
## 6 Andalucía             29-Málaga             656115
```

2.2.3 Filtrado de observaciones-individuos: filter()

Una operación muy habitual es reducir el conjunto de datos, al quedarse con aquellas filas u observaciones-individuos que cumplen determinadas condiciones lógicas, o dicho de otro modo, poseen ciertas características que nos interesan.

Ejemplo. Se quiere obtener un conjunto de datos que contengan únicamente la información de las provincias de Andalucía que tengan una población total superior a 800.000 habitantes.

```
datos_f01 = datos %>%
  filter(CCAA == "Andalucía", TOTAL>800000)
           # '==' para la igualdad en expresiones lógicas
datos_f01
```

```
## # A tibble: 4 x 5
##   Provincia  CCAA        TOTAL  Varon  Mujer
##   <chr>      <chr>       <dbl>  <dbl>  <dbl>
## 1 11-Cádiz   Andalucía 1116491 552463 564028
## 2 18-Granada Andalucía  821660 401638 420022
## 3 29-Málaga  Andalucía 1287017 630902 656115
## 4 41-Sevilla Andalucía 1727603 846220 881383
```

```
# Equivalente a:
# datos_f02 = datos %>%
#  filter(CCAA == "Andalucía" & TOTAL>800000)
```

Nota: como puede verse en el ejemplo anterior se puede añadir más de una condición. La expresión lógica podría ser tan compleja como se necesite, recordando que se pueden usar paréntesis para facilitar su correcta construcción. Los operadores lógicos más usados son: `==` (igual), `<` (menor), `>` (mayor), `<=` (menor o igual), `>=` (mayor o igual), `!=` (distinto), `&` (y lógico), `|` (o lógico).

Nota: en `filter()` se usa muy a menudo la función `is.na()` para seleccionar las filas que tienen el valor `NA` en una columna, o también `!is.na()` para seleccionar las filas que no tienen el valor `NA` en una columna.

En nuestros datos, no aparecen valores NA, pero se podría haber utilizado una llamada del siguiente tipo:

```
datos %>%
   filter(!is.na(TOTAL))
```

Ejemplo. Se quiere trabajar únicamente con aquellas provincias que tienen más hombres que mujeres, mostrando las variables: Provincia, población de hombres y población de mujeres y ordenando por número de hombres (de forma descendente).

```
datos %>%
  filter(Varon >= Mujer) %>%
  select(Provincia,Varon,Mujer) %>%
  arrange(desc(Varon)) %>%
  kable(booktabs=TRUE)
```

Provincia	Varon	Mujer
35-Palmas (Las)	444761	442915
04-Almería	272023	264708
09-Burgos	174576	174358
22-Huesca	104089	102413
19-Guadalajara	88535	86464
05-Ávila	81850	81592
40-Segovia	73973	73721
44-Teruel	68724	67134
42-Soria	45443	45274
51-Ceuta	35949	35556

Nota. Cualquiera de las herramientas del paquete "dplyr" se podrían emplear varias veces y en distintas posiciones, siempre que sea sintácticamente correcta la expresión.

2.2.4 Añadir nuevas columnas o variables calculadas: mutate()

La función `mutate()` permite añadir nuevas columnas a nuestros datos al efectuar algún tipo de operación más o menos compleja, generalmente a partir de los datos de las otras columnas.

Ejemplo. Se quieren añadir dos columnas:

- una que contenga la diferencia entre el número de hombres y mujeres,
- y otra que contenga el porcentaje de mujeres respecto al total de la población de la provincia.

```
datos_m01 = datos %>%
  mutate(Diferencia = Varon - Mujer,
         PorcMuj = round( 100 * (Mujer/TOTAL),2)
         )
datos_m01 %>%
  head(10) %>%
  kable()
```

Provincia	CCAA	TOTAL	Varon	Mujer	Diferencia	PorcMuj
01-Álava	País Vasco	286387	142036	144351	-2315	50.40
02-Albacete	Castilla-La Mancha	364835	181461	183374	-1913	50.26
03-Alicante/Alacant	Comunidad Valenciana	1461925	722162	739763	-17601	50.60
04-Almería	Andalucía	536731	272023	264708	7315	49.32
33-Asturias	Asturias (Principado de)	1062998	508995	554003	-45008	52.12
05-Ávila	Castilla y León	163442	81850	81592	258	49.92
06-Badajoz	Extremadura	654882	323541	331341	-7800	50.60
07-Balears (Illes)	Balears (Illes)	841669	417314	424355	-7041	50.42
08-Barcelona	Cataluña	4805927	2341592	2464335	-122743	51.28
09-Burgos	Castilla y León	348934	174576	174358	218	49.97

Como puede observarse, se han mantenido las columnas existentes y se han añadido las nuevas al conjunto de datos. **Nota**: la función `transmute()` del paquete dplyr añade columnas pero sin mantener las columnas existentes.

Ejemplo. Ahora construimos los datos con una nueva columna. Trabajamos únicamente con aquellas provincias que tienen mayor número de hombres que de mujeres, y presentamos la nueva variable diferencia (hombres menos mujeres), ordenando por la diferencia (mayor a menor) pero sin mostrar el total.

```
datos %>%
  filter(Mujer <= Varon) %>%
  mutate(Diferencia = Varon - Mujer) %>%
  arrange(desc(Diferencia)) %>%
  select(-TOTAL) %>%
  kable()
```

Provincia	CCAA	Varon	Mujer	Diferencia
04-Almería	Andalucía	272023	264708	7315
19-Guadalajara	Castilla-La Mancha	88535	86464	2071
35-Palmas (Las)	Canarias	444761	442915	1846
22-Huesca	Aragón	104089	102413	1676
44-Teruel	Aragón	68724	67134	1590
51-Ceuta	Ceuta	35949	35556	393
05-Ávila	Castilla y León	81850	81592	258
40-Segovia	Castilla y León	73973	73721	252
09-Burgos	Castilla y León	174576	174358	218
42-Soria	Castilla y León	45443	45274	169

Nota. El uso del "-" delante de la variable "TOTAL" ha significado que se muestren todas las columnas menos la columna "TOTAL".

2.2.5 Obtener resúmenes estadísticos: summarise()

Con la función `summarise()` se obtienen objetos que contienen en sus columnas cualquier tipo de resumen estadístico obtenido sobre alguna columna teniendo en cuenta todas sus filas.

Ejemplo. En el siguiente ejemplo obtenemos de los datos de población de las provincias españolas en el censo del 2001, la suma total de hombres, la suma total de mujeres, la población total (obtenida de 2 formas), la media de población de los hombres por provincia y la cuasidesviación típica de la población de hombres.

```
datos_su01 = datos %>%
  summarise(TotHombres = sum(Varon),
            TotMujeres = sum(Mujer),
            TotalHM = sum(Varon+Mujer),
            TotalHM2 = TotHombres + TotMujeres,
            MediaHombres = mean(Varon),
            SdHombres = sd(Varon))
datos_su01
```

```
## # A tibble: 1 x 6
##   TotHombres TotMujeres  TotalHM TotalHM2 MediaHombres SdHombres
##        <dbl>      <dbl>    <dbl>    <dbl>        <dbl>     <dbl>
## 1   20012882   20834489 40847371 40847371      384863.   474446.
```

Es un objeto con una sola fila, en la que por ejemplo, `TotHombres` se ha obtenido sumando todos los elementos de la columna "Varon", es decir, se ha calculado el total de hombres en España en el censo del 2001.

Se puede observar, que hay más mujeres que hombres, a pesar de que al nacer existe una proporción mayor de niños que de niñas.

Nota. Son muy útiles las funciones definidas en "dplyr" para emplear con `summarise()`: `n()` (para contar filas), `n_distinct(COLUMNA)` (para contar filas de valores distintos), etc.

2.2.6 Agrupar filas u observaciones-individuos: group_by(). Uso con summarise()

La función `group_by()` nos ayudará a obtener resúmenes estadísticos pero para cada uno de los grupos de filas o individuos que se hayan establecido.

Ejemplo. Se quiere obtener la población total para cada CCAA.

```
datos_g01 = datos %>%
  group_by(CCAA) %>%
  summarise(TotalCCAA = sum(TOTAL),
            TotHombresCCAA = sum(Varon),
            TotMujeresCCAA = sum(Mujer,na.rm = TRUE))
datos_g01 %>%
  kable(booktabs=TRUE)
```

CCAA	TotalCCAA	TotHombresCCAA	TotMujeresCCAA
Andalucía	7357558	3622066	3735492
Aragón	1204215	594846	609369
Asturias (Principado de)	1062998	508995	554003
Balears (Illes)	841669	417314	424355
Canarias	1694477	842966	851511
Cantabria	535131	260586	274545
Castilla y León	2456474	1209874	1246600
Castilla-La Mancha	1760516	875550	884966
Cataluña	6343110	3106531	3236579
Ceuta	71505	35949	35556
Comunidad Valenciana	4162776	2046984	2115792
Extremadura	1058503	524361	534142
Galicia	2695880	1294378	1401502
Madrid (Comunidad de)	5423384	2609746	2813638
Melilla	66411	33134	33277
Murcia (Región de)	1197646	597265	600381
Navarra (Comunidad Foral de)	555829	276629	279200
País Vasco	2082587	1017881	1064706
Rioja (La)	276702	137827	138875

La función `group_by` ha preparado los datos para que aparezcan las provincias de cada CCAA agrupadas, de forma que al llamar a la función `summarise()` no devuelve una única fila como en los ejemplos del apartado anterior, sino que obtiene una fila de resultados para cada grupo formado, en este caso para cada CCAA. Por ejemplo, el valor de "TotalCCAA" para Andalucía se ha obtenido al sumar las poblaciones (TOTAL) de las 8 provincias de Andalucía.

Esto mismo se podría haber hecho con ayuda del sistema base de R, con la función `aggregate()` (devuelve un data.frame) o `tapply()` (devuelve un vector), como se muestra a continuación:

```
aggregate(TOTAL ~ CCAA, datos, sum)
```

```
with(datos, tapply(TOTAL,CCAA,sum,na.rm=TRUE))
```

Ejemplo. Se quiere obtener el total de la población por CCAA, pero nos interesa presentar únicamente las que tienen más de 5 millones de habitantes y ordenadas de mayor a menor por población total.

```
datos_ag02 = datos %>%
  group_by(CCAA) %>%
  summarise(TOTALCCAA = sum(TOTAL),
            TOTALVaron = sum(Varon),
```

```
            TOTALMujer = sum(Mujer)) %>%
  filter(TOTALCCAA >= 5000000) %>%
  arrange(desc(TOTALCCAA))

datos_ag02 %>%
  kable()
```

CCAA	TOTALCCAA	TOTALVaron	TOTALMujer
Andalucía	7357558	3622066	3735492
Cataluña	6343110	3106531	3236579
Madrid (Comunidad de)	5423384	2609746	2813638

Nota: Se pueden utilizar: `count()` o `tally()` también para contar observaciones por grupo.

Nota: `slice()`, `slice_head()`, `slice_tail()`, `slice_min()`, `slice_max()`, `slice_sample()`: Permiten seleccionar filas por sus posiciones (valores enteros). Salvo la primera función, `slice()`, todas tienen como argumentos `n` (número filas) o `prop` (fracción de filas). Se pueden usar de forma conjunta con datos agrupados: `group_by()`. Sustituyen a funciones como: `top_n()` o `top_frac()`. Las siguientes instrucciones son correctas:

```
datos %>%
  slice(1:10)

datos %>%
  group_by(CCAA) %>%
  slice_head(prop = 0.5)

datos %>%
  group_by(CCAA) %>%
  slice_tail(prop = 0.5)

datos %>%
  group_by(CCAA) %>%
  slice_min(prop = 0.5,order_by = TOTAL)

datos %>%
  group_by(CCAA) %>%
  slice_max(prop = 0.5,order_by = TOTAL)

set.seed(12345)
datos %>%
  slice_sample(prop = 0.3,replace = FALSE)

datos %>%
  group_by(CCAA) %>%
  slice_sample(n = 5, replace = FALSE)

datos %>%
  group_by(CCAA) %>%
  slice_sample(n = 5, replace = TRUE)
```

Se pueden usar también: `first()`, `last()` y `nth()`, para extraer el primer, último o n-ésimo valor de un grupo (o vector).

Ejemplo. Para contar el número de provincias por cada CCAA

```
datos %>%
  group_by(CCAA) %>%
  summarise(NumProvincias = n()) # Equivale a:
```

```
## # A tibble: 19 x 2
##    CCAA                          NumProvincias
##    <chr>                                 <int>
##  1 Andalucía                                 8
##  2 Aragón                                    3
##  3 Asturias (Principado de)                  1
##  4 Balears (Illes)                           1
##  5 Canarias                                  2
##  6 Cantabria                                 1
##  7 Castilla y León                           9
##  8 Castilla-La Mancha                        5
##  9 Cataluña                                  4
## 10 Ceuta                                     1
## 11 Comunidad Valenciana                      3
## 12 Extremadura                               2
## 13 Galicia                                   4
## 14 Madrid (Comunidad de)                     1
## 15 Melilla                                   1
## 16 Murcia (Región de)                        1
## 17 Navarra (Comunidad Foral de)              1
## 18 País Vasco                                3
## 19 Rioja (La)                                1
```

```
# datos %>%
#   group_by(CCAA) %>%
#   count()  # o tally()
```

2.2.7 Uso de "case_when" para generalizar el "ifelse" en el sistema tidyverse

Para explicar la miniherramienta o función **case_when()**, se utilizan los siguientes datos que contienen personas que pertenecen o no a dos grupos: "Grupo1" y "Grupo2".

```
df = data.frame(
  Nombre = c("Juan","Ana","Marta"),
  Grupo1 = c(F,T,T),
  Grupo2 = c(F,F,T)
)
df
```

```
##   Nombre Grupo1 Grupo2
## 1   Juan  FALSE  FALSE
## 2    Ana   TRUE  FALSE
## 3  Marta   TRUE   TRUE
```

Se utiliza la función **case_when()** para construir una nueva columna (**mutate()**) que asigne valores nuevos en función de los valores de las columnas: "Grupo1" y "Grupo2". Como puede verse, aparecen una serie de expresiones lógicas a la izquierda del símbolo **"~"**, que si es cierta asigna el valor que está a su derecha.

```
df <- df %>%
  mutate(Grupo = case_when(
    Grupo1 & Grupo2 ~ "A",      # Ambos grupos: Grupo  A
```

```
    xor(Grupo1, Grupo2) ~ "B", # A un grupo solamente: Grupo B
    !Grupo1 & !Grupo2 ~ "C",   # Ningún grupo: Grupo C
     TRUE ~ "D"                # En otro caso: Grupo D (no habría otro caso)
))
df
```

```
##   Nombre Grupo1 Grupo2 Grupo
## 1   Juan  FALSE  FALSE     C
## 2    Ana   TRUE  FALSE     B
## 3  Marta   TRUE   TRUE     A
```

Nota: En demografía, cuando se quiere agrupar la variable edad en distintas categorías se puede utilizar `case_when()`.

2.3 Uso del paquete "tidyr"

2.3.1 Formato largo-formato ancho

Esta configuración de miniherramientas que usa el paquete **dplyr** invita a otros paquetes a extenderlo. Uno de tales paquetes es el paquete **tidyr** que forma parte del sistema "tidyverse".

En general, el sistema **tidyverse** supone que cada fila es una observación, y cada columna es una variable. A esta disposición se la conoce como **FORMATO LARGO**.

En el siguiente ejemplo se presenta una situación diferente que aparece habitualmente. Hay una variable, lluvia o cantidad de precipitación, que se extiende sobre tres columnas ("lluvia_estacion01" a "lluvia_estacion03"). Esta disposición de los datos se conoce como **FORMATO ANCHO**.

```
set.seed(24)
dat = data.frame(temper = runif(3, 15, 25),
                 lluvia_estacion01 = runif(3, 1, 3),
                 lluvia_estacion02 = runif(3, 1, 3),
                 lluvia_estacion03 = runif(3, 1, 3))
dat
```

```
##     temper lluvia_estacion01 lluvia_estacion02 lluvia_estacion03
## 1 17.92574          2.037794          1.559471          1.509450
## 2 17.24891          2.325239          2.527641          2.209778
## 3 22.04223          2.840888          2.603261          1.741470
```

Como se ha comentado anteriormente, el sistema tidyverse espera que los datos se encuentren en "formato largo". En algunas situaciones se necesita transformar los datos de "formato ancho" a "formato largo". Es decir, en estos datos se necesita reunir todos los valores de precipitaciones en una única columna, y añadir una columna de identificación adicional que especifique la estación a la que pertenece.

2.3.2 Formato ancho a formato largo o combinación: pivot_longer o gather

La **operación de combinación** que convierte datos **de formato ancho a formato largo** puede realizarse usando la función **`pivot_longer()`** o también con **`gather()`**.

```
pivot_longer(
  data,
  cols,
  names_to = "name",
  values_to = "value",
  # más argumentos
  ...
```

```
)
#gather(data, key, value, ...)
```

donde:

- `data`, es el data.frame de entrada
- `names_to` (`key`), el nombre de la columna con la identificación en el data.frame resultante
- `values_to` (`value`), el nombre de la columna con los valores (precipitacion en nuestro ejemplo) en el data.frame resultante.
- `cols` (`...`), especificación de qué columnas deberían ser reunidas/combinadas. Se usarán nombres de columnas, con un menos delante para excluir la columna de la combinación.

El siguiente código llama a **`pivot_longer()`** para transformar los datos del ejemplo anterior (aparece comentada el modo equivalente de hacerlo con la función: **`gather()`**).

```
library(tidyr)
dat_nuevos = dat %>%
  pivot_longer(cols = -temper,
               names_to = "id_estacion",
               values_to = "precipitacion")
# dat_nuevos = gather(dat, key = id_estacion, value = precipitacion, -temper)
head(dat_nuevos)
```

```
## # A tibble: 6 x 3
##   temper id_estacion       precipitacion
##    <dbl> <chr>                     <dbl>
## 1   17.9 lluvia_estacion01          2.04
## 2   17.9 lluvia_estacion02          1.56
## 3   17.9 lluvia_estacion03          1.51
## 4   17.2 lluvia_estacion01          2.33
## 5   17.2 lluvia_estacion02          2.53
## 6   17.2 lluvia_estacion03          2.21
```

2.3.3 Formato largo a formato ancho o extensión: pivot_wider o spread

La **operación contraria de extensión** se realiza usando la función **`pivot_wider()`** o la función **`spread()`**, las cuales convierten de **formato largo a formato ancho**. En el siguiente ejemplo, se transforman a formato ancho los datos en formato largo creados en el apartado anterior: "dat_nuevos".

```
pivot_wider(dat_nuevos,
            names_from = "id_estacion",
            values_from = "precipitacion")
```

```
## # A tibble: 3 x 4
##   temper lluvia_estacion01 lluvia_estacion02 lluvia_estacion03
##    <dbl>             <dbl>             <dbl>             <dbl>
## 1   17.9              2.04              1.56              1.51
## 2   17.2              2.33              2.53              2.21
## 3   22.0              2.84              2.60              1.74
```

```
#spread(dat_nuevos,key = id_estacion, value = precipitacion)
```

2.3.4 Algunas curiosidades del paquete tidyr: separate() y unite()

La función `separate()` permite obtener nuevas columnas a partir de una columna de partida, al indicar una cadena de texto como separador. Por ejemplo, tenemos una columna fecha y queremos separarla en 3

columnas nuevas: día, mes y año. Su complementario es `unite()`, que construye una nueva columna a partir de unir varias columnas con un separador que indiquemos.

En la columna "Provincia" del conjunto de datos de población del censo de 2001 empleado anteriormente, aparece el código junto al nombre de la provincia separado por un guión.

```
head(datos)
```

```
## # A tibble: 6 x 5
##   Provincia           CCAA                       TOTAL  Varon  Mujer
##   <chr>               <chr>                      <dbl>  <dbl>  <dbl>
## 1 01-Álava            País Vasco                286387 142036 144351
## 2 02-Albacete         Castilla-La Mancha        364835 181461 183374
## 3 03-Alicante/Alacant Comunidad Valenciana     1461925 722162 739763
## 4 04-Almería          Andalucía                 536731 272023 264708
## 5 33-Asturias         Asturias (Principado de) 1062998 508995 554003
## 6 05-Ávila            Castilla y León           163442  81850  81592
```

Para separarlo en dos columnas, se haría del siguiente modo:

```
datos_sep = datos %>%
  separate(col = Provincia,into = c("Codigo","Provincia"),sep="-")
head(datos_sep)
```

```
## # A tibble: 6 x 6
##   Codigo Provincia        CCAA                       TOTAL  Varon  Mujer
##   <chr>  <chr>            <chr>                      <dbl>  <dbl>  <dbl>
## 1 01     Álava            País Vasco                286387 142036 144351
## 2 02     Albacete         Castilla-La Mancha        364835 181461 183374
## 3 03     Alicante/Alacant Comunidad Valenciana     1461925 722162 739763
## 4 04     Almería          Andalucía                 536731 272023 264708
## 5 33     Asturias         Asturias (Principado de) 1062998 508995 554003
## 6 05     Ávila            Castilla y León           163442  81850  81592
```

Por defecto, elimina la columna de partida, pero esto podría cambiarse (consultar la sintaxis de la función).

Ahora podría ordenarse por código o por el nombre de la provincia.

Con `unite()` podríamos volver a unirla, pero por ejemplo, poniendo el código detrás, del siguiente modo:

```
datos_sep %>%
  unite(col="Provincia",sep="-", Provincia,Codigo) %>%
  head()
```

```
## # A tibble: 6 x 5
##   Provincia           CCAA                       TOTAL  Varon  Mujer
##   <chr>               <chr>                      <dbl>  <dbl>  <dbl>
## 1 Álava-01            País Vasco                286387 142036 144351
## 2 Albacete-02         Castilla-La Mancha        364835 181461 183374
## 3 Alicante/Alacant-03 Comunidad Valenciana     1461925 722162 739763
## 4 Almería-04          Andalucía                 536731 272023 264708
## 5 Asturias-33         Asturias (Principado de) 1062998 508995 554003
## 6 Ávila-05            Castilla y León           163442  81850  81592
```

2.4 Recursos adicionales

A continuación se recogen algunos enlaces con material relacionado con "dplyr" para poder profundizar:

- dplyr en CRAN: https://cran.r-project.org/web/packages/dplyr/index.html

- Documentación de dplyr: https://dplyr.tidyverse.org
- Transparencias sobre el uso de dplyr (inglés): http://patilv.com/dplyr-nycflights/
- Youtube: uso de dplyr unos 38 minutos

3 Importar datos desde ficheros excel, csv, px, RData

Habitualmente se necesitará descargar ficheros de las páginas web de instituciones oficiales, que contienen la información necesaria, y para ello, será conveniente recordar la existencia en R de la función:

```
download.file(url, destfile, method, quiet = FALSE, mode = "w",
              cacheOK = TRUE,
              extra = getOption("download.file.extra"), ...)
```

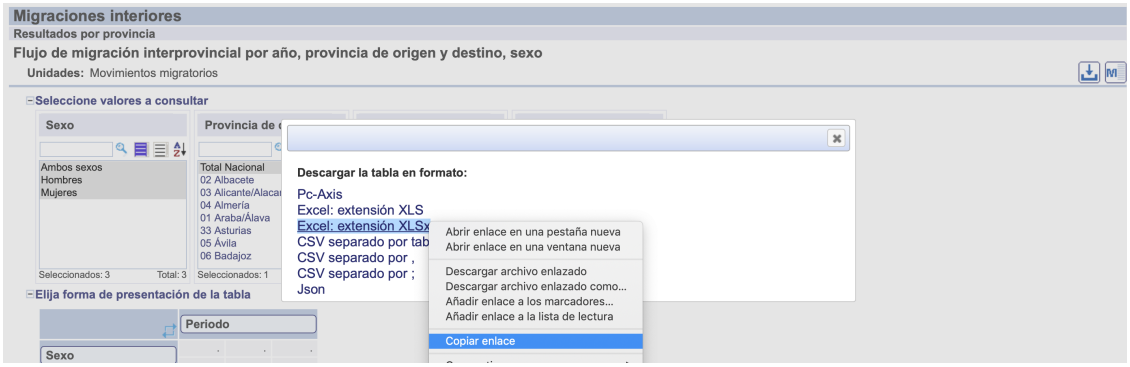
La forma más utilizada de usar esta función es indicando la "url" de descarga y el nombre que tendrá el fichero de destino, "destfile", como puede verse en el siguiente ejemplo:

```
download.file(url="http://destio.us.es/calvo/descargas/datos_ordenadores.csv",
              destfile = "datos_ordenadores.csv")
```

En este caso se podría haber indicado un camino relativo para el fichero de destino, por ejemplo, "datos/datos_ordenadores.csv", copiará el fichero en la subcarpeta "datos" (respecto al directorio de trabajo actual).

El uso de esta función puede ser muy útil para hacer el código reproducible y no será necesario escribir el camino en un navegador para llegar a obtener ese fichero. En el INE, muchas veces el enlace de descarga puede obtenerse utilizando el menú flotante que se activa al pulsar sobre el enlace con el botón derecho del ratón.

Por ejemplo, en la siguiente página web del INE: https://www.ine.es/jaxiT3/Tabla.htm?t=24379&L=0, aparece a la derecha de la página un botón de "descarga" y si pulsamos sobre el se abre una ventana flotante que nos permite elegir entre diferentes formatos de ficheros para descargar toda la información. Si acercamos el ratón sobre cualquiera de ellos y pulsamos el botón derecho, se podrá copiar el enlace de descarga.



A continuación se muestran los enlaces que se han podido copiar:

Formato	url
"px"	https://www.ine.es/jaxiT3/files/t/es/px/24379.px?nocab=1
"excel"	https://www.ine.es/jaxiT3/files/t/es/xlsx/24379.xlsx?nocab=1
"csv, por tabuladores"	https://www.ine.es/jaxiT3/files/t/es/csv/24379.csv?nocab=1
"csv, separado por ',' "	https://www.ine.es/jaxiT3/files/t/es/csv_c/24379.csv?nocab=1
"csv, separado por ';' "	https://www.ine.es/jaxiT3/files/t/es/csv_sc/24379.csv?nocab=1
"json"	https://servicios.ine.es/wstempus/js/es/DATOS_TABLA/24379?tip=AM

Nota. Es interesante observar las ligeras diferencias que existen en las urls de los distintos formatos de ficheros.

3.1 Importar datos desde excel. Paquetes readxl

Para importar datos contenidos en un fichero excel utilizaremos el paquete "readxl", cuya función principal es: `read_excel()` (otras variantes con la misma sintaxis son: `read_xlsx()` y `read_xls()`). Los datos importados son del tipo "tibble" (data.frame mejorados).

Su uso es muy sencillo cuando se quiere leer el contenido completo de una hoja en un fichero Excel:

```
datos = read_excel("ficheroexcel.xlsx") # equivalente a:
datos = read_excel("ficheroexcel.xlsx",sheet = 1)
```

Su sintaxis completa es la siguiente:

```
read_excel(path, sheet = NULL, range = NULL, col_names = TRUE,
  col_types = NULL, na = "", trim_ws = TRUE, skip = 0,
  n_max = Inf, guess_max = min(1000, n_max),
  progress = readxl_progress(), .name_repair = "unique")
```

Ejemplo. En este ejemplo, se copiarán datos en un fichero excel obtenidos de una página del INE, y posteriormente este se importará en R con la función `read_xlsx()`.

Visitamos la página web del INE: http://www.ine.es/censo_accesible/es/seleccion_ambito.jsp

Y seleccionamos:

- Nacional
- Todas las personas
- Filas: Lugares de residencia>Residencia actual>Provincia de residencia
- Columnas: Datos demográficos básicos>Sexo
- Siguiente
- Ver tabla
- Seleccionamos los datos de la tabla y los copiamos al portapapeles con Ctrl+C.
- Abrimos Excel: Pegamos los datos copiados (Ctrl+V)
- Guardamos el fichero Excel como: censo2001act01.xlsx (Hoja1)

	A	B	C	D
1	**Sexo**	**TOTAL**	**Varón**	**Mujer**
2	**Provincia de residencia**			
3	**TOTAL**	40 847 371	20 012 882	20 834 489
4	**01-Álava**	286 387	142 036	144 351
5	**02-Albacete**	364 835	181 461	183 374
6	**03-Alicante/Alacant**	1 461 925	722 162	739 763
7	**04-Almería**	536 731	272 023	264 708
8	**33-Asturias**	1 062 998	508 995	554 003
9	**05-Ávila**	163 442	81 850	81 592
10	**06-Badajoz**	654 882	323 541	331 341
11	**07-Balears (Illes)**	841 669	417 314	424 355
12	**08-Barcelona**	4 805 927	2 341 592	2 464 335
13	**09-Burgos**	348 934	174 576	174 358
14	**10-Cáceres**	403 621	200 820	202 801
15	**11-Cádiz**	1 116 491	552 463	564 028
16	**39-Cantabria**	535 131	260 586	274 545
17	**12-Castellón/Castelló**	484 566	240 673	243 893
18	**51-Ceuta**	71 505	35 949	35 556
19	**13-Ciudad Real**	478 957	235 189	243 768
20	**14-Córdoba**	761 657	372 464	389 193
21	**15-Coruña (A)**	1 096 027	525 388	570 639
22	**16-Cuenca**	200 346	99 959	100 387
23	**17-Girona**	565 304	280 830	284 474
24	**18-Granada**	821 660	401 638	420 022
25	**19-Guadalajara**	174 999	88 535	86 464

Figura 1: Contenido del fichero Excel creado: 'censo2001act01.xlsx' (hoja 1). Fuente: elaboración propia

Ya estamos listos para importar los datos en R, pero como se ve en la captura, los datos no vienen en un formato fácil de manipular.

Con ayuda de la función `read_xlsx()` vamos a leer varios rangos de la hoja excel en diferentes lecturas para posteriormente unir toda la información para conseguir los datos en el formato adecuado.

```
library(readxl)
datosx = read_excel("datos/censo2001act01.xlsx",sheet=1,
                    range="A4:D55",col_names = FALSE)
```

```
## New names:
## * `` -> ...1
## * `` -> ...2
## * `` -> ...3
## * `` -> ...4
```

```
nombres_var = read_excel("datos/censo2001act01.xlsx",sheet=1,
                    range="A1:D1",col_names = FALSE)
```

```
## New names:
## * `` -> ...1
## * `` -> ...2
```

```
## * `` -> ...3
## * `` -> ...4
```

```
datosx_total = read_excel("datos/censo2001act01.xlsx",sheet=1,
                  range="A3:D3",col_names = FALSE)
```

```
## New names:
## * `` -> ...1
## * `` -> ...2
## * `` -> ...3
## * `` -> ...4
```

Arreglamos los nombres de nuestro conjunto de datos, evitando caracteres que pueden cambiar según la codificación (acentos, "ñ", etc.):

```
names(datosx)
```

```
## [1] "...1" "...2" "...3" "...4"
```

```
names(datosx) = c("Provincia",nombres_var[1,2:4])
```

```
## Warning: The `value` argument of `names<-` must be a character vector as of
## tibble 3.0.0.
```

```
names(datosx)[3] = "Varon"
names(datosx)
```

```
## [1] "Provincia" "TOTAL"     "Varon"     "Mujer"
```

```
library(dplyr)
glimpse(datosx)
```

```
## Rows: 52
## Columns: 4
## $ Provincia <chr> "01-Álava", "02-Albacete", "03-Alicante/Alacant", "04-Almerí~
## $ TOTAL     <dbl> 286387, 364835, 1461925, 536731, 1062998, 163442, 654882, 84~
## $ Varon     <dbl> 142036, 181461, 722162, 272023, 508995, 81850, 323541, 41731~
## $ Mujer     <dbl> 144351, 183374, 739763, 264708, 554003, 81592, 331341, 42435~
```

Ejemplo. Igual que el anterior, pero obtenemos los datos de las Comunidades Autónomas.

Visitamos la página web del INE: http://www.ine.es/censo_accesible/es/seleccion_ambito.jsp

Y seleccionamos:

- Nacional
- Todas las personas
- Filas:
  - Lugares de residencia>Residencia actual>Provincia de residencia
  - Lugares de residencia>Residencia actual>CC.AA. de residencia
- Columnas: Datos demográficos básicos>Sexo
- Siguiente
- Ver tabla
- Seleccionamos los datos de la tabla y los copiamos al portapapeles con Ctrl+C.
- Abrimos Excel: Pegamos los datos copiados (Ctrl+V)
- Guardamos el fichero Excel como: censo2001act01.xlsx (Hoja2)

	A	B	C	D	E	F
1	**Sexo**	**TOTAL**	**Varón**	**Mujer**		
2	**Provincia de residencia**	**CC.AA. de residencia**				
3	**TOTAL**	**TOTAL**	40 847 371	20 012 882	20 834 489	
4	**01-Álava**	**TOTAL**	286 387	142 036	144 351	
5	**País Vasco**	286 387	142 036	144 351		
6	**02-Albacete**	**TOTAL**	364 835	181 461	183 374	
7	**Castilla-La Mancha**	364 835	181 461	183 374		
8	**03-Alicante/Alacant**	**TOTAL**	1 461 925	722 162	739 763	
9	**Comunidad Valenciana**	1 461 925	722 162	739 763		
10	**04-Almería**	**TOTAL**	536 731	272 023	264 708	
11	**Andalucía**	536 731	272 023	264 708		
12	**33-Asturias**	**TOTAL**	1 062 998	508 995	554 003	
13	**Asturias (Principado de)**	1 062 998	508 995	554 003		
14	**05-Ávila**	**TOTAL**	163 442	81 850	81 592	
15	**Castilla y León**	163 442	81 850	81 592		
16	**06-Badajoz**	**TOTAL**	654 882	323 541	331 341	
17	**Extremadura**	654 882	323 541	331 341		
18	**07-Balears (Illes)**	**TOTAL**	841 669	417 314	424 355	
19	**Balears (Illes)**	841 669	417 314	424 355		
20	**08-Barcelona**	**TOTAL**	4 805 927	2 341 592	2 464 335	
21	**Cataluña**	4 805 927	2 341 592	2 464 335		
22	**09-Burgos**	**TOTAL**	348 934	174 576	174 358	
23	**Castilla y León**	348 934	174 576	174 358		
24	**10-Cáceres**	**TOTAL**	403 621	200 820	202 801	
25	**Extremadura**	403 621	200 820	202 801		

Figura 2: Contenido del fichero Excel creado: 'censo2001act01.xlsx' (hoja 2). Fuente: elaboración propia

Ya estamos listos para importar los datos en R.

```
datosx2 = read_xlsx("datos/censo2001act01.xlsx",sheet=2,
                    range="A4:E107",col_names = FALSE)
```

```
## New names:
## * `` -> ...1
## * `` -> ...2
## * `` -> ...3
## * `` -> ...4
## * `` -> ...5
```

```
nombres_var2 = read_xlsx("datos/censo2001act01.xlsx",sheet=2,
                    range="A1:D1",col_names = FALSE)
```

```
## New names:
## * `` -> ...1
## * `` -> ...2
## * `` -> ...3
## * `` -> ...4
```

```
datosx_total2 = read_xlsx("datos/censo2001act01.xlsx",sheet=2,
                    range="A3:E3",col_names = FALSE)
```

```
## New names:
## * `` -> ...1
## * `` -> ...2
## * `` -> ...3
## * `` -> ...4
## * `` -> ...5
```

Arreglamos los nombres de los datos provinciales:

```
names(datosx2)
```

```
## [1] "...1" "...2" "...3" "...4" "...5"
```

```
names(datosx2) = c("Provincia","CCAA",nombres_var2[1,2:4])
```

```
## Warning: The `value` argument of `names<-` must be a character vector as of
## tibble 3.0.0.
```

```
names(datosx2)[4] = "Varon"
names(datosx2)
```

```
## [1] "Provincia" "CCAA"      "TOTAL"     "Varon"     "Mujer"
```

Al mostrar los primeros datos, puede verse que todavía no están de forma correcta:

```
glimpse(datosx2)
```

```
## Rows: 104
## Columns: 5
## $ Provincia <chr> "01-Álava", "País Vasco", "02-Albacete", "Castilla-La Mancha~
## $ CCAA      <chr> "TOTAL", "286387", "TOTAL", "364835", "TOTAL", "1461925", "T~
## $ TOTAL     <dbl> 286387, 142036, 364835, 181461, 1461925, 722162, 536731, 272~
## $ Varon     <dbl> 142036, 144351, 181461, 183374, 722162, 739763, 272023, 2647~
## $ Mujer     <dbl> 144351, NA, 183374, NA, 739763, NA, 264708, NA, 554003, NA, ~
```

Nos quedamos con los datos que tiene en la columna "Mujer" valores no NA:

```
datosx2 %>%
  filter(!is.na(Mujer)) %>%
  head(10) %>%
  kable()
```

Provincia	CCAA	TOTAL	Varon	Mujer
01-Álava	TOTAL	286387	142036	144351
02-Albacete	TOTAL	364835	181461	183374
03-Alicante/Alacant	TOTAL	1461925	722162	739763
04-Almería	TOTAL	536731	272023	264708
33-Asturias	TOTAL	1062998	508995	554003
05-Ávila	TOTAL	163442	81850	81592
06-Badajoz	TOTAL	654882	323541	331341
07-Balears (Illes)	TOTAL	841669	417314	424355
08-Barcelona	TOTAL	4805927	2341592	2464335
09-Burgos	TOTAL	348934	174576	174358

Extraemos los datos de las CCAA de las filas que tienen en la columna "Mujer" valor NA, y nos quedamos con la primera columna ("Provincia") que tienen los nombres de las CCAA:

```
CCAA = datosx2 %>%
  dplyr::filter(is.na(Mujer)) %>%
  dplyr::select(Provincia)
CCAA %>%
  head(10) %>%
  kable()
```

Provincia
País Vasco
Castilla-La Mancha
Comunidad Valenciana
Andalucía
Asturias (Principado de)
Castilla y León
Extremadura
Balears (Illes)
Cataluña
Castilla y León

Finalizamos colocando esos datos en la columna CCAA, de forma que quede la información de forma correcta:

```
datos2_mej = datosx2 %>%
  filter(!is.na(Mujer))
datos2_mej$CCAA = as.character(CCAA$Provincia)
datos2_mej %>%
  kable()
```

Provincia	CCAA	TOTAL	Varon	Mujer
01-Álava	País Vasco	286387	142036	144351
02-Albacete	Castilla-La Mancha	364835	181461	183374
03-Alicante/Alacant	Comunidad Valenciana	1461925	722162	739763
04-Almería	Andalucía	536731	272023	264708
33-Asturias	Asturias (Principado de)	1062998	508995	554003
05-Ávila	Castilla y León	163442	81850	81592
06-Badajoz	Extremadura	654882	323541	331341
07-Balears (Illes)	Balears (Illes)	841669	417314	424355
08-Barcelona	Cataluña	4805927	2341592	2464335
09-Burgos	Castilla y León	348934	174576	174358
10-Cáceres	Extremadura	403621	200820	202801
11-Cádiz	Andalucía	1116491	552463	564028
39-Cantabria	Cantabria	535131	260586	274545
12-Castellón/Castelló	Comunidad Valenciana	484566	240673	243893
51-Ceuta	Ceuta	71505	35949	35556
13-Ciudad Real	Castilla-La Mancha	478957	235189	243768
14-Córdoba	Andalucía	761657	372464	389193
15-Coruña (A)	Galicia	1096027	525388	570639
16-Cuenca	Castilla-La Mancha	200346	99959	100387
17-Girona	Cataluña	565304	280830	284474
18-Granada	Andalucía	821660	401638	420022
19-Guadalajara	Castilla-La Mancha	174999	88535	86464
20-Guipúzcoa	País Vasco	673563	330288	343275
21-Huelva	Andalucía	462579	229013	233566
22-Huesca	Aragón	206502	104089	102413
23-Jaén	Andalucía	643820	317343	326477
24-León	Castilla y León	488751	238139	250612
25-Lleida	Cataluña	362206	180425	181781
27-Lugo	Galicia	357648	173339	184309
28-Madrid	Madrid (Comunidad de)	5423384	2609746	2813638
29-Málaga	Andalucía	1287017	630902	656115
52-Melilla	Melilla	66411	33134	33277

Provincia	CCAA	TOTAL	Varon	Mujer
30-Murcia	Murcia (Región de)	1197646	597265	600381
31-Navarra	Navarra (Comunidad Foral de)	555829	276629	279200
32-Ourense	Galicia	338446	161968	176478
34-Palencia	Castilla y León	174143	85955	88188
35-Palmas (Las)	Canarias	887676	444761	442915
36-Pontevedra	Galicia	903759	433683	470076
26-Rioja (La)	Rioja (La)	276702	137827	138875
37-Salamanca	Castilla y León	345609	167948	177661
38-Santa Cruz de Tenerife	Canarias	806801	398205	408596
40-Segovia	Castilla y León	147694	73973	73721
41-Sevilla	Andalucía	1727603	846220	881383
42-Soria	Castilla y León	90717	45443	45274
43-Tarragona	Cataluña	609673	303684	305989
44-Teruel	Aragón	135858	68724	67134
45-Toledo	Castilla-La Mancha	541379	270406	270973
46-Valencia/València	Comunidad Valenciana	2216285	1084149	1132136
47-Valladolid	Castilla y León	498094	243999	254095
48-Vizcaya	País Vasco	1122637	545557	577080
49-Zamora	Castilla y León	199090	97991	101099
50-Zaragoza	Aragón	861855	422033	439822

3.2 Datos en un fichero RData

Los datos bien definidos que contiene el objeto "datos2_mej" podría guardarse en un fichero con formato "RData" para poder utilizarlo en cualquier estudio posterior sobre ellos. Se podría hacer llamando a la función `save()` del siguiente modo:

```
save(datos2_mej,file="datos2_mej.RData")
```

Podría cargarse con ayuda de la función `load()` del siguiente modo:

```
load("datos2_mej.RData")
```

3.3 Importar datos desde ficheros csv

Estos ficheros se pueden importar con las funciones del sistema base:

- `read.table()`
- `read.csv()`
- `read.csv2()`

La función `read.table()` es la más general y versátil:

```
read.table(file, header = FALSE, sep = "", quote = "\"'",
           dec = ".", numerals = c("allow.loss", "warn.loss", "no.loss"),
           row.names, col.names, as.is = !stringsAsFactors,
           na.strings = "NA", colClasses = NA, nrows = -1,
           skip = 0, check.names = TRUE, fill = !blank.lines.skip,
           strip.white = FALSE, blank.lines.skip = TRUE,
           comment.char = "#",
           allowEscapes = FALSE, flush = FALSE,
           stringsAsFactors = default.stringsAsFactors(),
           fileEncoding = "", encoding = "unknown", text, skipNul = FALSE)
```

Ejemplo 1:

Vamos a descargar datos procedentes del INE que contienen tablas de migración entre las provincias españolas y de diferentes años.

Descargamos un fichero csv (separado por ";") desde el INE e importamos en R:

```
fichero = download.file(url="https://www.ine.es/jaxiT3/files/t/es/csv_c/24379.csv?nocab=1",
                        destfile = "datos/24379.csv")
```

El contenido del fichero descargado es el siguiente:

```
Estadística de Migraciones
Resultados por provincia
Flujo de migración interprovincial por año, provincia de origen y destino, sexo
Unidades:   Movimientos migratorios

,2018,2017,2016,2015,2014,2013,2012,2011,2010,2009,2008,
Ambos sexos,
        Total Nacional,
            Total Nacional,465.536,443.729,448.418,465.514,464.142,467.396,483.793,509.595
                    ,513.592,508.489,524.701,
            02 Albacete,5.110,4.827,4.972,5.268,5.177,5.111,5.178,5.190,5.176,5.010,5.071,
            03 Alicante/Alacant,17.178,16.837,17.438,17.966,18.248,17.629,18.732,19.565,20
                    .842,22.248,22.239,
            04 Almería,8.226,7.789,7.607,8.072,7.639,7.695,8.356,10.706,9.344,9.307,12.326,
            01 Araba/Álava,3.822,3.716,3.614,3.624,3.662,3.785,3.714,3.657,3.670,3.672,3.724,
            33 Asturias,7.394,7.287,7.493,7.868,8.030,7.906,7.986,8.431,8.515,8.009,7.780,
            05 Ávila,3.352,3.389,3.408,3.448,3.472,3.411,3.705,3.781,3.500,3.611,3.710,
            06 Badajoz,6.878,6.868,6.807,6.620,6.768,6.381,6.450,6.405,6.174,6.152,6.486,
            07 Balears, Illes,14.148,13.787,12.261,13.672,13.364,12.556,13.913,16.104,16.576
                    ,16.095,15.223,
            08 Barcelona,43.374,38.356,38.769,39.178,39.827,42.899,45.250,47.425,49.101,50
                    .037,54.264,
            48 Bizkaia,8.864,8.740,8.351,8.651,8.491,8.707,8.539,9.194,9.634,9.173,9.977,
            09 Burgos,4.710,4.506,4.754,4.847,4.909,4.880,5.057,5.141,5.193,5.158,5.195,
```

De forma que para leerlo correctamente tendríamos que utilizar la siguiente llamada a la función `read.table()`:

```
datoscsv = read.table("datos/24379.csv",sep=",",header=TRUE,
                      dec=".",skip = 5,fill = TRUE,
                      strip.white = TRUE,stringsAsFactors=FALSE)
```

```
head(datoscsv,12)
```

```
##                      X   X2018   X2017   X2016   X2015   X2014   X2013   X2012
## 1          Ambos sexos      NA      NA      NA      NA      NA      NA      NA
## 2       Total Nacional      NA      NA      NA      NA      NA      NA      NA
## 3       Total Nacional 465.536 443.729 448.418 465.514 464.142 467.396 483.793
## 4          02 Albacete   5.110   4.827   4.972   5.268   5.177   5.111   5.178
## 5  03 Alicante/Alacant  17.178  16.837  17.438  17.966  18.248  17.629  18.732
## 6           04 Almería   8.226   7.789   7.607   8.072   7.639   7.695   8.356
## 7       01 Araba/Álava   3.822   3.716   3.614   3.624   3.662   3.785   3.714
## 8          33 Asturias   7.394   7.287   7.493   7.868   8.030   7.906   7.986
## 9             05 Ávila   3.352   3.389   3.408   3.448   3.472   3.411   3.705
## 10          06 Badajoz   6.878   6.868   6.807   6.620   6.768   6.381   6.450
## 11          07 Balears   Illes  14.148  13.787  12.261  13.672  13.364  12.556
## 12        08 Barcelona  43.374  38.356  38.769  39.178  39.827  42.899  45.250
##      X2011   X2010   X2009   X2008    X.1
## 1       NA      NA      NA      NA     NA
## 2       NA      NA      NA      NA     NA
## 3  509.595 513.592 508.489 524.701     NA
## 4    5.190   5.176   5.010   5.071     NA
```

```
## 5   19.565  20.842  22.248  22.239     NA
## 6   10.706   9.344   9.307  12.326     NA
## 7    3.657   3.670   3.672   3.724     NA
## 8    8.431   8.515   8.009   7.780     NA
## 9    3.781   3.500   3.611   3.710     NA
## 10   6.405   6.174   6.152   6.486     NA
## 11  13.913  16.104  16.576  16.095 15.223
## 12  47.425  49.101  50.037  54.264     NA
```

Como puede apreciarse existen algunos problemas, por ejemplo, en la fila 11 la provincia "Balears, Illes" ha sido separada al utilizar el csv con separador ",".

Utilizaremos ahora el fichero csv con separador ";"

```
download.file(url="https://www.ine.es/jaxiT3/files/t/es/csv_sc/24379.csv?nocab=1",
              destfile = "datos/24379b.csv")
```

Usamos la función `read.table()` pero con el argumento `sep=";"`:

```
datoscsvb = read.table("datos/24379b.csv",sep=";",header=TRUE,
                       dec=".",skip = 5,fill = TRUE,
                       strip.white = TRUE,stringsAsFactors=FALSE)
```

Puede verse que la importación no tiene los mismos problemas que en la llamada anterior:

```
head(datoscsvb,12)
```

```
##                       X   X2018   X2017   X2016   X2015   X2014   X2013   X2012
## 1           Ambos sexos      NA      NA      NA      NA      NA      NA      NA
## 2        Total Nacional      NA      NA      NA      NA      NA      NA      NA
## 3        Total Nacional 465.536 443.729 448.418 465.514 464.142 467.396 483.793
## 4           02 Albacete   5.110   4.827   4.972   5.268   5.177   5.111   5.178
## 5  03 Alicante/Alacant  17.178  16.837  17.438  17.966  18.248  17.629  18.732
## 6            04 Almería   8.226   7.789   7.607   8.072   7.639   7.695   8.356
## 7        01 Araba/Álava   3.822   3.716   3.614   3.624   3.662   3.785   3.714
## 8           33 Asturias   7.394   7.287   7.493   7.868   8.030   7.906   7.986
## 9              05 Ávila   3.352   3.389   3.408   3.448   3.472   3.411   3.705
## 10           06 Badajoz   6.878   6.868   6.807   6.620   6.768   6.381   6.450
## 11   07 Balears, Illes  14.148  13.787  12.261  13.672  13.364  12.556  13.913
## 12         08 Barcelona  43.374  38.356  38.769  39.178  39.827  42.899  45.250
##      X2011   X2010   X2009   X2008 X.1
## 1       NA      NA      NA      NA  NA
## 2       NA      NA      NA      NA  NA
## 3  509.595 513.592 508.489 524.701  NA
## 4    5.190   5.176   5.010   5.071  NA
## 5   19.565  20.842  22.248  22.239  NA
## 6   10.706   9.344   9.307  12.326  NA
## 7    3.657   3.670   3.672   3.724  NA
## 8    8.431   8.515   8.009   7.780  NA
## 9    3.781   3.500   3.611   3.710  NA
## 10   6.405   6.174   6.152   6.486  NA
## 11  16.104  16.576  16.095  15.223  NA
## 12  47.425  49.101  50.037  54.264  NA
```

Utilizaremos ahora el fichero csv con separador tabulador "`\t`":

```
download.file(url="https://www.ine.es/jaxiT3/files/t/es/csv/24379.csv?nocab=1",
              destfile = "datos/24379c.csv")
```

Ahora usamos la función `read.table()` pero con el argumento `sep="\t"`:

```
datoscsvc = read.table("datos/24379c.csv",sep="\t",header=TRUE,
                       dec=".",skip = 5,fill = TRUE,
                       strip.white = TRUE,stringsAsFactors=FALSE)
```

Puede verse que la importación no tiene los mismos problemas que en la primera llamada (se ha añadido una columna al final que no contiene datos, como en la anterior):

```
head(datoscsvc,12)
```

```
##                       X   X2018   X2017   X2016   X2015   X2014   X2013   X2012
## 1           Ambos sexos      NA      NA              NA      NA      NA      NA
## 2        Total Nacional      NA      NA              NA      NA      NA      NA
## 3        Total Nacional 465.536 443.729 448.418 465.514 464.142 467.396 483.793
## 4           02 Albacete   5.110   4.827   4.972   5.268   5.177   5.111   5.178
## 5  03 Alicante/Alacant  17.178  16.837  17.438  17.966  18.248  17.629  18.732
## 6           04 Almería   8.226   7.789   7.607   8.072   7.639   7.695   8.356
## 7        01 Araba/Álava   3.822   3.716   3.614   3.624   3.662   3.785   3.714
## 8           33 Asturias   7.394   7.287   7.493   7.868   8.030   7.906   7.986
## 9              05 Ávila   3.352   3.389   3.408   3.448   3.472   3.411   3.705
## 10          06 Badajoz   6.878   6.868   6.807   6.620   6.768   6.381   6.450
## 11   07 Balears, Illes  14.148  13.787  12.261  13.672  13.364  12.556  13.913
## 12         08 Barcelona  43.374  38.356  38.769  39.178  39.827  42.899  45.250
##      X2011   X2010   X2009   X2008 X.1
## 1       NA      NA      NA      NA  NA
## 2       NA      NA      NA      NA  NA
## 3  509.595 513.592 508.489 524.701  NA
## 4    5.190   5.176   5.010   5.071  NA
## 5   19.565  20.842  22.248  22.239  NA
## 6   10.706   9.344   9.307  12.326  NA
## 7    3.657   3.670   3.672   3.724  NA
## 8    8.431   8.515   8.009   7.780  NA
## 9    3.781   3.500   3.611   3.710  NA
## 10   6.405   6.174   6.152   6.486  NA
## 11  16.104  16.576  16.095  15.223  NA
## 12  47.425  49.101  50.037  54.264  NA
```

Como hemos comentado anteriormente se trata de datos de tablas de migración entre las provincias españolas y de diferentes años, todavía quedaría el proceso de extraer la tabla de migración que nos pueda interesar. La función `slice()` podría ayudarnos a conseguirlo.

3.4 Importar datos desde ficheros px

El paquete R que nos va a facilitar la lectura de los ficheros con formato "px" es: "pxR". Se debe instalar como cualquier otro paquete.

Cargamos la librería "pxR":

```
library(pxR) # pxR carga el paquete: plyr, que entra en conflictos con tidyverse
library(tidyverse)  # para solucionarlo carga tidyverse después
```

3.4.1 Ejemplo 1

Recurriendo a la web del INE, obtener la siguiente información en formato "px":

- Para las provincias españolas, obtenga la matriz de flujos de migración interprovincial interior para el año 2010 y 2017.
- Para las comunidades autónomas, obtenga la matriz de flujos de mgigración interautonómica interior para el año 2010 y 2017.

Veamos el paso a paso:

- **Paso 0.** Visitamos el enlace: http://www.ine.es/dynt3/inebase/index.htm?padre=3678&capsel=3694 (INEBASE, Demografía y Población, Fenómenos demográficos, Migraciones interiores).
- **Paso 1.** Visitamos el enlace: http://www.ine.es/jaxiT3/Tabla.htm?t=24379&L=0 y descargamos el fichero en formato "PC-axis" o "px" pulsando sobre el icono del lado derecho, y lo guardamos en la subcarpeta "datos" de nuestro proyecto.
- **Paso 2.** Instalamos el paquete R: "pxR".
- **Paso 3.** Cargamos los datos con ayuda de: `as.data.frame(read.px("fichero.px"),stringsAsFactors=FALSE)`.

```
#download.file(url="https://www.ine.es/jaxiT3/files/t/es/px/24379.px?nocab=1",
#               destfile = "datos/24379.px")
dfej01a <- as.data.frame(read.px("datos/24379.px"),stringsAsFactors=FALSE)
head(dfej01a)
```

```
##   Periodo Provincia.de.origen Provincia.de.destino        Sexo    value
## 1    2017      Total Nacional       Total Nacional Ambos sexos 443729.0
## 2    2016      Total Nacional       Total Nacional Ambos sexos 448417.5
## 3    2015      Total Nacional       Total Nacional Ambos sexos 465513.9
## 4    2014      Total Nacional       Total Nacional Ambos sexos 464142.5
## 5    2013      Total Nacional       Total Nacional Ambos sexos 467396.5
## 6    2012      Total Nacional       Total Nacional Ambos sexos 483793.0
```

- **Paso 4.** Descargamos ahora para el ámbito de comunidades autónomas desde: http://www.ine.es/jaxiT3/Tabla.htm?t=24369&L=0

```
dfej01b <- as.data.frame(read.px("datos/24369.px"),stringsAsFactors=FALSE)
head(dfej01b)
```

```
##   Periodo Comunidades.y.ciudades.autónomas.de.origen
## 1    2017                              Total Nacional
## 2    2016                              Total Nacional
## 3    2015                              Total Nacional
## 4    2014                              Total Nacional
## 5    2013                              Total Nacional
## 6    2012                              Total Nacional
##   Comunidades.y.ciudades.autónomas.de.destino        Sexo    value
## 1                              Total Nacional Ambos sexos 342465.0
```

```
## 2                              Total Nacional Ambos sexos 347382.9
## 3                              Total Nacional Ambos sexos 359427.6
## 4                              Total Nacional Ambos sexos 358903.8
## 5                              Total Nacional Ambos sexos 363470.6
## 6                              Total Nacional Ambos sexos 378000.7
```

Nota: estos dos objetos están en formato largo.

4 Ejemplo: agrupar en intervalos de edad a partir de edades simples `case_when()`

Cargamos los datos almacenados en ficheros "RData", con la función R: `load()`:

```
load("datos/DatosINE_PLC.RData")
```

También se pueden cargar ficheros ".px" del INE (y también del IECA) con el paquete "pxR". Pero en este caso, se ha creado una función específica para que las columnas de tipo "factor" sean convertidas a tipo "character".

```
demog_px_ine_import = function(ficheropx) {
  suppressWarnings(my.px.object <- read.px(ficheropx))
  my.px.data2   <-  as.data.frame( my.px.object,stringsAsFactors=FALSE )
  for (i in 1:ncol(my.px.data2)) {
    if (is.factor(my.px.data2[[i]])) {
      my.px.data2[[i]] = as.character(my.px.data2[[i]])
    }
  }
  return(my.px.data2)
}


Nacimientos_CCAA_EdadMadre_px = demog_px_ine_import("datos/02004.px")
```

Se tiene información de todas las CCAA y también de varios años de calendario. Nuestro objetivo es obtener la información exclusivamente para **Cantabria y 2016**.

```
Gano=2016
GCCAA="Cantabria"
nLxAmbos=TRUE
```

Con la función `unique()` obtenemos los valores diferentes (distintos o únicos) que aparecen en la columna `CCAA` y en la columna `Calendario` de `Nacimientos_CCAA_EdadMadre_2010a2016`:

```
CCAAdif = unique(Nacimientos_CCAA_EdadMadre_2010a2016$CCAA)
cualCCAA = which(CCAAdif==GCCAA)
(anodif = unique(Nacimientos_CCAA_EdadMadre_2010a2016$Calendario))
```

```
## [1] 2016 2015 2014 2013 2012 2011 2010
```

```
cualano = which(anodif==Gano)
```

También se ha obtenido el índice correspondiente a Cantabria y 2016: `cualCCAA` y `cualano`.

La siguiente instrucción asegura que cuando usemos `select` se utilice la función del paquete "dplyr":

```
select = dplyr::select
```

Se dispone de datos con edades simples (algunas vienen agrupadas) y además en formato de texto, como puede verse a continuación:

```
unique(Nacimientos_CCAA_EdadMadre_px$Edad.de.la.madre) %>%
  head(20)
```

```
##  [1] "Todas las edades" "Menos de 15 años" "De 15 años"       "De 16 años"
##  [5] "De 17 años"       "De 18 años"       "De 19 años"       "De 20 años"
##  [9] "De 21 años"       "De 22 años"       "De 23 años"       "De 24 años"
## [13] "De 25 años"       "De 26 años"       "De 27 años"       "De 28 años"
## [17] "De 29 años"       "De 30 años"       "De 31 años"       "De 32 años"
```

Queremos agrupar las edades simples, en grupos de edades. Para ello intentaremos construir una nueva variable o columna: "GEdades", con ayuda de `mutate()`, y la función del paquete "dplyr": `case_when()` como veremos en el siguiente ejemplo:

```
Nacimientos_CCAA_EdadMadre_px %>%
    filter(Comunidad.Autónoma==CCAAdif[cualCCAA],
           Estado.civil.de.la.madre=="Todos los nacimientos",
           Sexo.del.nacido=="Ambos sexos") %>%
    mutate(Edades = Edad.de.la.madre) %>%
    mutate(
      GEdades = case_when(
        ((Edades >= "De 15 años") & (Edades <=  "De 19 años")) ~ "15-19",
        ((Edades >= "De 20 años") & (Edades <=  "De 24 años")) ~ "20-24",
        ((Edades >= "De 25 años") & (Edades <=  "De 29 años")) ~ "25-29",
        ((Edades >= "De 30 años") & (Edades <=  "De 34 años")) ~ "30-34",
        ((Edades >= "De 35 años") & (Edades <=  "De 39 años")) ~ "35-39",
        ((Edades >= "De 40 años") & (Edades <=  "De 44 años")) ~ "40-44",
        ((Edades >= "De 45 años") & (Edades <=  "De 49 años")) ~ "45-49"
      )
    ) %>%
  head(20)
```

```
##    Sexo.del.nacido Estado.civil.de.la.madre Edad.de.la.madre Comunidad.Autónoma
## 1      Ambos sexos    Todos los nacimientos Todas las edades          Cantabria
## 2      Ambos sexos    Todos los nacimientos Menos de 15 años          Cantabria
## 3      Ambos sexos    Todos los nacimientos       De 15 años          Cantabria
## 4      Ambos sexos    Todos los nacimientos       De 16 años          Cantabria
## 5      Ambos sexos    Todos los nacimientos       De 17 años          Cantabria
## 6      Ambos sexos    Todos los nacimientos       De 18 años          Cantabria
## 7      Ambos sexos    Todos los nacimientos       De 19 años          Cantabria
## 8      Ambos sexos    Todos los nacimientos       De 20 años          Cantabria
## 9      Ambos sexos    Todos los nacimientos       De 21 años          Cantabria
## 10     Ambos sexos    Todos los nacimientos       De 22 años          Cantabria
## 11     Ambos sexos    Todos los nacimientos       De 23 años          Cantabria
## 12     Ambos sexos    Todos los nacimientos       De 24 años          Cantabria
## 13     Ambos sexos    Todos los nacimientos       De 25 años          Cantabria
## 14     Ambos sexos    Todos los nacimientos       De 26 años          Cantabria
## 15     Ambos sexos    Todos los nacimientos       De 27 años          Cantabria
## 16     Ambos sexos    Todos los nacimientos       De 28 años          Cantabria
## 17     Ambos sexos    Todos los nacimientos       De 29 años          Cantabria
## 18     Ambos sexos    Todos los nacimientos       De 30 años          Cantabria
## 19     Ambos sexos    Todos los nacimientos       De 31 años          Cantabria
## 20     Ambos sexos    Todos los nacimientos       De 32 años          Cantabria
##    value           Edades GEdades
## 1   4118 Todas las edades    <NA>
## 2     NA Menos de 15 años    <NA>
```

```
## 3      2       De 15 años   15-19
## 4      5       De 16 años   15-19
## 5      8       De 17 años   15-19
## 6     20       De 18 años   15-19
## 7     22       De 19 años   15-19
## 8     33       De 20 años   20-24
## 9     28       De 21 años   20-24
## 10    40       De 22 años   20-24
## 11    41       De 23 años   20-24
## 12    53       De 24 años   20-24
## 13    61       De 25 años   25-29
## 14    92       De 26 años   25-29
## 15   120       De 27 años   25-29
## 16   169       De 28 años   25-29
## 17   173       De 29 años   25-29
## 18   227       De 30 años   30-34
## 19   272       De 31 años   30-34
## 20   295       De 32 años   30-34
```

Hay filas que no nos interesan, ya que aparecen en la nueva variable "GEdades" con el valor `NA`.

Por tanto, además de eliminar esas observaciones, necesitamos resumir esos datos al agrupar para los valores distintos de la nueva variable "GEdades", de la siguiente forma:

```
tmp2a = Nacimientos_CCAA_EdadMadre_px %>%
    filter(Comunidad.Autónoma==CCAAdif[cualCCAA],
           Estado.civil.de.la.madre=="Todos los nacimientos",
           Sexo.del.nacido=="Ambos sexos") %>%
    mutate(Edades = Edad.de.la.madre) %>%
    mutate(
      GEdades = case_when(
        ((Edades >= "De 15 años") & (Edades <=  "De 19 años")) ~ "15-19",
        ((Edades >= "De 20 años") & (Edades <=  "De 24 años")) ~ "20-24",
        ((Edades >= "De 25 años") & (Edades <=  "De 29 años")) ~ "25-29",
        ((Edades >= "De 30 años") & (Edades <=  "De 34 años")) ~ "30-34",
        ((Edades >= "De 35 años") & (Edades <=  "De 39 años")) ~ "35-39",
        ((Edades >= "De 40 años") & (Edades <=  "De 44 años")) ~ "40-44",
        ((Edades >= "De 45 años") & (Edades <=  "De 49 años")) ~ "45-49"
      )
    ) %>%
    filter(!is.na(GEdades)) %>%
    select(GEdades,value) %>%
    group_by(GEdades) %>%
    dplyr::summarise(
      CCAA = CCAAdif[cualCCAA],
      Ano = anodif[cualano],
      NacimientosAmbos = sum(value,na.rm = T)
    )
```

De esta forma hemos obtenido el número de nacimientos de **ambos sexos**, que hay en Cantabria durante el 2016, por grupos de Edad de la madre:

```
head(tmp2a)
```

```
## # A tibble: 6 x 4
##   GEdades CCAA        Ano NacimientosAmbos
```

```
##   <chr>   <chr>     <dbl>            <dbl>
## 1 15-19   Cantabria  2016               57
## 2 20-24   Cantabria  2016              195
## 3 25-29   Cantabria  2016              615
## 4 30-34   Cantabria  2016             1486
## 5 35-39   Cantabria  2016             1382
## 6 40-44   Cantabria  2016              361
```

Ahora hacemos operaciones parecidas para obtener el número de nacimientos de **niños**, que hay en Cantabria durante el 2016, por grupos de Edad de la madre:

```
tmp2Hombres = Nacimientos_CCAA_EdadMadre_px %>%
  filter(Comunidad.Autónoma==CCAAdif[cualCCAA],
         Estado.civil.de.la.madre=="Todos los nacimientos",
         Sexo.del.nacido=="Hombres") %>%
  mutate(Edades = Edad.de.la.madre) %>%
  mutate(
    GEdades = case_when(
      ((Edades >= "De 15 años") & (Edades <=    "De 19 años")) ~ "15-19",
      ((Edades >= "De 20 años") & (Edades <=    "De 24 años")) ~ "20-24",
      ((Edades >= "De 25 años") & (Edades <=    "De 29 años")) ~ "25-29",
      ((Edades >= "De 30 años") & (Edades <=    "De 34 años")) ~ "30-34",
      ((Edades >= "De 35 años") & (Edades <=    "De 39 años")) ~ "35-39",
      ((Edades >= "De 40 años") & (Edades <=    "De 44 años")) ~ "40-44",
      ((Edades >= "De 45 años") & (Edades <=    "De 49 años")) ~ "45-49"
    )
  ) %>%
  filter(!is.na(GEdades)) %>%
  select(GEdades,value) %>%
  group_by(GEdades) %>%
  dplyr::summarise(
    CCAA = CCAAdif[cualCCAA],
    Ano = anodif[cualano],
    NacimientosHombres = sum(value,na.rm = T)
  )
```

```
head(tmp2Hombres) %>%
  kable(booktabs=TRUE)
```

GEdades	CCAA	Ano	NacimientosHombres
15-19	Cantabria	2016	27
20-24	Cantabria	2016	89
25-29	Cantabria	2016	332
30-34	Cantabria	2016	750
35-39	Cantabria	2016	723
40-44	Cantabria	2016	186

Y repetimos las mismas operaciones parecidas para obtener el número de nacimientos de **niñas**, que hay en Cantabria durante el 2016, por grupos de Edad de la madre:

```
tmp2Mujeres = Nacimientos_CCAA_EdadMadre_px %>%
  filter(Comunidad.Autónoma==CCAAdif[cualCCAA],
         Estado.civil.de.la.madre=="Todos los nacimientos",
         Sexo.del.nacido=="Mujeres") %>%
```

```
  mutate(Edades = Edad.de.la.madre) %>%
  mutate(
    GEdades = case_when(
      ((Edades >= "De 15 años") & (Edades <=    "De 19 años")) ~ "15-19",
      ((Edades >= "De 20 años") & (Edades <=    "De 24 años")) ~ "20-24",
      ((Edades >= "De 25 años") & (Edades <=    "De 29 años")) ~ "25-29",
      ((Edades >= "De 30 años") & (Edades <=    "De 34 años")) ~ "30-34",
      ((Edades >= "De 35 años") & (Edades <=    "De 39 años")) ~ "35-39",
      ((Edades >= "De 40 años") & (Edades <=    "De 44 años")) ~ "40-44",
      ((Edades >= "De 45 años") & (Edades <=    "De 49 años")) ~ "45-49"
    )
  ) %>%
  filter(!is.na(GEdades)) %>%
  select(GEdades,value) %>%
  group_by(GEdades) %>%
  dplyr::summarise(
    CCAA = CCAAdif[cualCCAA],
    Ano = anodif[cualano],
    NacimientosMujeres = sum(value,na.rm = T)
  )
```

```
head(tmp2Mujeres) %>%
  kable(booktabs=TRUE)
```

GEdades	CCAA	Ano	NacimientosMujeres
15-19	Cantabria	2016	30
20-24	Cantabria	2016	106
25-29	Cantabria	2016	283
30-34	Cantabria	2016	736
35-39	Cantabria	2016	659
40-44	Cantabria	2016	175

Nota: este procedimiento se podría generalizar para cualquier comunidad autónoma y año.

5 Crear informes con R Markdown en RStudio. Uso de Proyectos

5.1 Trabajar con Proyectos en RStudio

5.1.1 Recomendación: no guardar el espacio de trabajo en RStudio

Se suele recomendar que no se guarde nuestro espacio de trabajo "Environment" entre sesiones. En primer lugar, esto hará que RStudio arranque y se cierre más rápidamente (no tiene que cargar ni grabar nada), y en segundo lugar, nos obliga a escribir todo el código para poder reproducir todo el trabajo (no se dependa de recordar exactamente lo que se hizo y en que orden). Esto último es muy adecuado cuando se trabaja en equipo, o cuando se quiera retomar el trabajo después de un largo periodo de tiempo.

Para hacer, en "Options" (de RStudio o del Proyecto) en el apartado primero: "General" se recomienda

- "Restore '.RData' into workspace at startup": dejarlo sin seleccionar.
- "Save workspace to .RData on exit: Never": seleccionar "Never" (nunca) para que al salir no grabe los objetos almacenados en "Environment Global".

5.1.2 Referencias a caminos relativos y no a caminos absolutos

Cuando se utilizan ficheros y directorios en un sistema operativo, existen varias formas de escribir la referencia a ese elemento. En R y RStudio, se permite emplear como separados de directorios "/" (la barra ascendente) para todos los sistemas operativos, de ahí que se recomiende su uso.

También existen diferencias entre Mac/Linux y Windows cuando se utilizan "caminos absolutos" (absolute paths), ya que en Mac/Linux suelen empezar con una barra ascendente, por ejemplo: "/users/jaime", y en Windows suelen empezar con una letra, por ejemplo: "c:/users/jaime".

Una recomendación muy conveniente es no usar nunca "caminos absolutos" en el código, ya que complica que el código sea compartido, ya que para que funcionara sería necesario que en el otro ordenador se tuviese la misma configuración de directorios que la suya (por ejemplo, no funcionaría si el nombre del usuario es distinto a "jaime").

Usar "caminos relativos" evita este tipo de problemas, pero se tiene que garantizar que el "directorio de trabajo" sea correctamente elegido. Por ejemplo, si nuestro fichero de código R: "ejemplo1.R" sde encuentra en un directorio ("directorio de trabajo), y se quiere importar un fichero csv"fic.csv" que está en una subcarpeta del directorio de trabajo llamada "datos", se podría usar en el "camino relativo" en la llamada a la función "read.csv()":

misdatos = read.csv("datos/fic.csv")

Si ejecutamos la función `getwd()`, esta nos informa del directorio de trabajo actual, que también puede verse en RStudio sobre la pestaña de la consola (en la parte superior izquierda).

También es posible definir el directorio de trabajo actual, llamando a la función `setwd(dir="camino")` o utilizando RStudio (menú "Session > Set Working Directory" y eligiendo una de las opciones que se presentan).

Por tanto, definiendo como directorio de trabajo el que contiene nuestro código, la llamada para importar el fichero csv funcionaría correctamente.

5.1.3 Proyectos en RStudio

Tenemos la opción de crear un nuevo **proyecto R**. Un proyecto es simplemente un directorio de trabajo que lleva asociado un fichero de proyecto con extensión ".RProj". Cuando se abre un proyecto (usando "File/Open Project" en RStudio o al hacer doble clic sobre el fichero ".Rproj" fuera de RStudio), el directorio de trabajo automáticamente se definirá al directorio donde se encuentra el fichero del proyecto ".RProj".

Se recomienda crear un nuevo proyecto R cuando se va a comenzar un nuevo proyecto de investigación, un trabajo práctico de cualquier tipo, se va escribir un trabajo fin de estudios, etc. Una vez que se ha creado un

nuevo proyecto, deberían crearse subcarpetas en el directorio (se puede hacer desde RStudio->Files o desde el sistema operativo). Por ejemplo, podrían crearse algunas de las siguientes subcarpetas:

- "codigo": el código R
- "datos": ficheros de datos
- "imagenes": ficheros de imágenes utilizados
- "salidas": ficheros de informes obtenidos

y cualquier otra subcarpeta que pueda ser útil para la realización del trabajo.

5.1.4 Crear un Proyecto en RStudio

Del libro R for Data Science:

Para crear un nuevo proyecto en RStudio, se hará clic en el menú: "File > New Project".

1. Luego aparecerá un panel que nos preguntará si se quiere crear el proyecto en:
    - Nuevo directorio ("New Directory")
    - Directorio existente ("Existing Directory"). Esta opción, nos será útil si ya tenemos parte del material en una carpeta ya existente.
2. Luego nos pregunta que tipo de proyecto queremos, y aquí se seleccionará "Empty Project" (proyecto en un directorio vacío). Existen otras opciones con fines más específicos más avanzadas.
3. Y finalmente, aparece un panel en el que se tendrá que introducir:
    - el nombre que se le dará al directorio ("Directory name"). Elige un nombre que sea fácil de identificar, pero se recomienda no usar acentos, o cualquier otro carácter que sea específico de algunas lenguas ("ñ", epacios en blanco, etc).
    - seleccionar el directorio en el que se creará el proyecto
    - Y otros "checks" que nos permiten añadirles nuevas características a nuestro proyecto. Inicialmente pueden dejarse sin seleccionar.

5.1.5 Trabajando con Proyectos

En RStudio, existe un menú desplegable en la parte superior derecha específica para el trabajo con Proyectos, que nos permite:

- Crear un nuevo proyecto (New Project).
- Abrir un proyecto de nuestro ordenador (Open Project...).
- Cerrar el proyecto actual (Close Project).
- Un listado con los últimos proyectos abiertos, para que al seleccionar cualquiera de ellos, se pueda volver a abrir (a la derecha de cada uno de ellos aparece una flecha, que si la seleccionamos abrirá el proyecto en una ventana y sesión R distinta).
- Se podrá limpiar el listado de últimos proyectos abiertos (Clear Project List).
- Se podrán abrir las "opciones específicas del proyecto" (Project Options...), en las que se podrán concretar algunas características de RStudio que interesen que puedan ser diferentes a las opciones generales de RStudio (Global Options).

Además de las recomendaciones Generales indicadas anteriormente, que podrían indicarse en las opciones específicas del proyecto, también se recomienda activar:

- **"Always save history (even when not saving .RData)"**: que asegurará que el fichero ".Rhistory" siempre sea grabado con los comandos de nuestra sesión aunque hayamos elegido no grabar el fichero ".RData" al salir de Rstudio.

También es conveniente especificar la codificación utilizada en los ficheros de nuestro proyecto, la cual se recomienda que sea **"UTF-8"** (en Mac/Linux se usa por defecto, pero en Windows suele ser "ISO-8859-1" o la específica del país). La opción es: **"Text encoding"**.

Observe que los ficheros de código que no coincidan con la codificación por defecto pueden reabrirse correctamente usando el menú: **"File > Reopen with Encoding"**.

5.1.6 Cómo compartir o entregar un proyecto

Se puede incluir un fichero "README" o "LEEME" que explique brevemente en qué consiste el proyecto y qué aspectos hay que tener en cuenta para realizarlo: paquetes R que hay que instalar, de dónde se han obtenido los datos, etc.

Comprimir la carpeta del proyecto RStudio con todos los documentos utilizados (Rmd, imágenes, datos, etc.) y con las salidas html, pdf y excel generadas. También puede ser útil guardar todos los datos preparados (data.frames utilizados finalmente para presentarlos en las tablas) de la práctica en un único fichero RData.

Esto nos permite ter una copia de seguridad de nuestro proyecto, la cual se podría compartir con otras personas.

5.2 Mostrar o no código R

Los informes que se generan pueden presentar el código utilizado o no. El primer caso suele ser un informe dirigido a una persona que toma decisiones y no le interesa cómo se ha hecho si no las conclusiones obtenidas.

- **Soluciones sin código R**. Se consigue modificando el primer chunk de código R de forma que `echo = FALSE`:

```
knitr::opts_chunk$set(echo = FALSE,cache=F,message = FALSE)
```

- **Soluciones con código R**. Se consigue modificando el primer chunk de código R de forma que `echo = TRUE`:

```
knitr::opts_chunk$set(echo = TRUE,cache=F,message = FALSE)
```

Siempre se puede obligar a mostrar o no el código en un chunk particular usando la opción `echo=TRUE` o `echo=FALSE`.

5.3 Incluir enlaces y capturas de pantalla en un fichero R Markdown

Los enlaces se pueden incluir en nuestro documento R Markdown de dos formas distintas:

- `<url>`
- `[leyenda url](url)`

Y los gráficos pueden incluirse con ayuda de un "chunk" de código R que incluya:

```
knitr::include_graphics("ficherografico")
```

y las opciones de chunk:

- `echo=FALSE`
- `out.width="90%"`
- `fig.align="center"`
- Y para poner leyendas al gráfico: `fig.cap="Leyenda. Fuente: "`.

Ejemplo. El siguiente código markdown en un fichero Rmd produce enlaces e incluye gráficos guardados en nuestra carpeta de proyecto (capturas de pantalla que hemos realizado en nuestro ordenador):

```
- [INEBASE-Demografía y Población-Fenómenos
demográficos](http://www.ine.es/dyngs/INEbase/es/categoria.htm?c=Estadistica_P&
cid=1254735573002)

- [Movimiento Natural de Población: Nacimientos](
```

```
http://www.ine.es/dyngs/INEbase/es/operacion.htm?c=Estadistica_C&cid=1254736177007&
menu=resultados&secc=1254736195442&idp=1254735573002){target="_blank"}

Ver enlace: [Paso final MNP Nacimientos](http://www.ine.es/jaxiT3/Tabla.htm?
t=6506&L=0){target="_blank"}

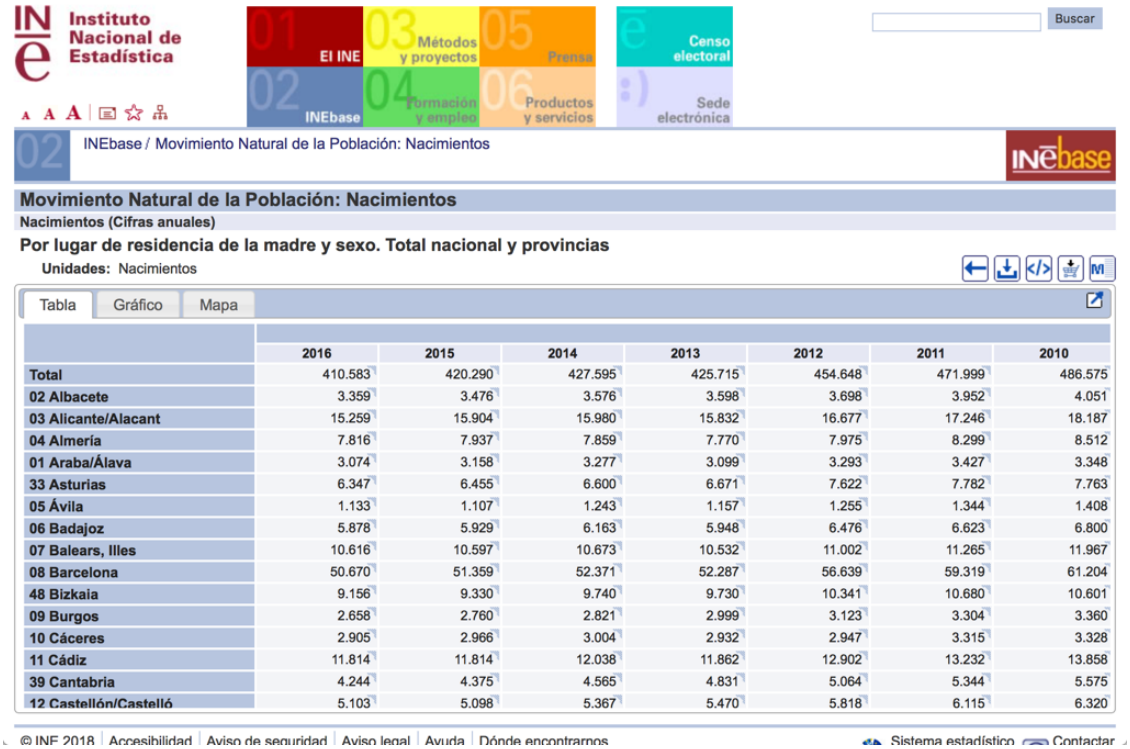

```{r echo=FALSE, out.width="90%",fig.align='center'}
knitr::include_graphics("imagenes/mnp_nac00a.png")
```

```{r echo=FALSE, out.width="90%",fig.align='center'}
knitr::include_graphics("imagenes/mnp_nac03.png")
```
```

Este código produciría los resultados que se muestran a continuación.

- INEBASE-Demografía y Población-Fenómenos demográficos
- Movimiento Natural de Población: Nacimientos

Ver enlace: Paso final MNP Nacimientos



Instituto Nacional de Estadística

El INE | Métodos y proyectos | Prensa | Censo electoral | INEbase | Formación y empleo | Productos y servicios | Sede electrónica

Buscar

INEbase / Movimiento Natural de la Población: Nacimientos

Movimiento Natural de la Población: Nacimientos

Nacimientos (Cifras anuales)

Por lugar de residencia de la madre y sexo. Total nacional y provincias

Unidades: Nacimientos

Tabla | Gráfico | Mapa

	2016	2015	2014	2013	2012	2011	2010
Total	410.583	420.290	427.595	425.715	454.648	471.999	486.575
02 Albacete	3.359	3.476	3.576	3.598	3.698	3.952	4.051
03 Alicante/Alacant	15.259	15.904	15.980	15.832	16.677	17.246	18.187
04 Almería	7.816	7.937	7.859	7.770	7.975	8.299	8.512
01 Araba/Álava	3.074	3.158	3.277	3.099	3.293	3.427	3.348
33 Asturias	6.347	6.455	6.600	6.671	7.622	7.782	7.763
05 Ávila	1.133	1.107	1.243	1.157	1.255	1.344	1.408
06 Badajoz	5.878	5.929	6.163	5.948	6.476	6.623	6.800
07 Balears, Illes	10.616	10.597	10.673	10.532	11.002	11.265	11.967
08 Barcelona	50.670	51.359	52.371	52.287	56.639	59.319	61.204
48 Bizkaia	9.156	9.330	9.740	9.730	10.341	10.680	10.601
09 Burgos	2.658	2.760	2.821	2.999	3.123	3.304	3.360
10 Cáceres	2.905	2.966	3.004	2.932	2.947	3.315	3.328
11 Cádiz	11.814	11.814	12.038	11.862	12.902	13.232	13.858
39 Cantabria	4.244	4.375	4.565	4.831	5.064	5.344	5.575
12 Castellón/Castelló	5.103	5.098	5.367	5.470	5.818	6.115	6.320

© INE 2018 | Accesibilidad | Aviso de seguridad | Aviso legal | Ayuda | Dónde encontrarnos | Sistema estadístico | Contactar

Para profundizar más en este apartado se puede consultar: "Escribir un trabajo fin de estudios con R Markdown" y los pdfs asociados: "Escribir un TFE (pdf)" y "Utilidades para documentos R Markdown (pdf)".

5.4 Incluir tablas en R Markdown: kable, kableExtra. Especificar leyendas

Como ya se ha visto una forma sencilla de mejorar la presentación de la información en formato tabla es usando la función `kable()` del paquete "knitr".

Pero existen otros paquetes R: kableExtra, huxtable, etc, que están especializados en la presentación de tablas en informes añadiendo muchas posibilidades, sobre los diferentes tipos de salidas: documentos pdf, páginas web, documentos word, etc.

Este apartado se puede encontrar de forma más desarrollada en el siguiente enlace: "Cómo crear Tablas de Información en R Markdown".

5.4.1 Uso de ficheros con funciones R

Se ha creado un fichero, llamado "funciones_tablas.R", en el que se encuentran definidas varias funciones que nos facilitará el trabajo para presentar información en forma de tablas.

Las funciones son:

- `func_salida_tablas()`
- `func_salida_excel()`
- `func_salida_excel_varias()`

A continuación se muestra su sintaxis:

```
func_salida_tablas(datos1,
                   salida="latex",
                   apaisadalatex=FALSE,
                   variaspaginas=TRUE,
```

```
                   fuentesize=NULL) # 8, 2, 12, ...

func_salida_excel(df01,
                   ficexcel,
                   titulo="Hoja 1",
                   fijaFC = TRUE,
                  anchuraAuto = TRUE)

func_salida_excel_varias(l_df01,
                           ficexcel,
                           vtitulo=paste("Hoja",1:length(l_df01)),
                           fijaFC = TRUE,
                           anchuraAuto = TRUE)
```

Importante: para generar la salida html o pdf, modificar el código de la celda de código R inicial (**setup**) del documento RMarkdown para que el objeto R `VRsalida` tome el valor adecuado, y cargue el fichero R: "funciones_tablas.R" (consultar el fichero "funciones_tablas.pdf" para más detalles), para incluir 3 funciones R predefinidas que facilitarán la presentación de la información en formato tablas y la posibilidad de guardar los datos descargados en ficheros Excel con un formato más personalizado.

- **salida html**: `VRsalida="html"`.

```
```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = FALSE,warning = FALSE,message = FALSE)
source("funciones_tablas.R")
VRsalida = "html"
```
```

- **salida pdf (o latex)**: `VRsalida="latex"`.

```
```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = FALSE,warning = FALSE,message = FALSE)
source("funciones_tablas.R")
VRsalida = "latex"
```
```

5.4.2 Salidas en formato tabla mejoradas con el paquete kableExtra

5.4.3 Creación de una tabla apaisada en pdf

```
datos1 = read_xlsx("datos/2852.xlsx",sheet=1,range="A8:U61")
```

```
## New names:
## * `` -> ...1
```

```
names(datos1)[1] = "Provincias"
```

Cuando se intenta presentar en un fichero pdf una tabla que necesita un ancho que supera los 21cm de un papel en formato a4 (los ficheros html no presentan limitación de ancho porque no está pensado inicialmente para imprimirse en papel), se puede utilizar la función definida `func_salida_tablas()`, indicando los argumentos `apaisadalatex = TRUE,variaspaginas=FALSE`:

```
func_salida_tablas(datos1,VRsalida,apaisadalatex = TRUE,variaspaginas=FALSE)
```

Provincias	2017	2016	2015	2014	2013	2012	2011	2010	2009	2008	2007	2006	2005	2004	2003	2002	2001	2000	1999	1998
Total Nacional	46572132	46557008	46624382	46771341	47129783	47265321	47190493	47021031	46745807	46157822	45200737	44708964	44108530	43197684	42717064	41837894	41116842	40499791	40202160	39852651
02 Albacete	390032	392118	394580	396987	400007	402837	402318	401682	400891	397493	392110	387658	384640	379448	376556	371787	367283	363263	361021	358597
03 Alicante/Alacant	1825332	1836459	1855047	1868438	1945642	1943910	1934127	1926285	1917012	1891477	1825264	1783555	1732389	1657040	1632349	1557968	1490265	1445144	1410946	1388933
04 Almería	706672	704297	701211	701688	699329	704219	702819	695560	684426	667635	646633	635850	612315	580077	565310	546498	533168	518229	512843	505448
01 Araba/Álava	326574	324126	323648	321932	321417	322557	319227	317352	313819	309635	305459	301926	299957	295905	294360	291860	288793	286497	285748	284595
33 Asturias	1034960	1042608	1051229	1061756	1068165	1077360	1081487	1084341	1085289	1080138	1074862	1076896	1076635	1073761	1075381	1073971	1075329	1076567	1084314	1081834
05 Ávila	160700	162514	164925	167015	168825	171265	172704	171896	171680	171815	168638	167818	167032	166108	165480	165138	163885	164991	166259	167132
06 Badajoz	679884	684113	686730	690929	693729	694533	693921	692137	688777	685246	678459	673474	671299	663896	663142	662808	664251	661874	664625	663803
07 Balears, Illes	1115999	1107220	1104479	1103442	1111674	1119439	1113114	1106049	1095426	1072844	1030650	1001062	983131	955045	947361	916968	878627	845630	821820	796483
08 Barcelona	5576037	5542680	5523922	5523784	5540925	5552050	5529099	5511147	5487935	5416447	5332513	5309404	5226354	5117885	5052666	4906117	4804606	4736277	4706325	4666271
48 Bizkaia	1148302	1147576	1148775	1151905	1156447	1158439	1155772	1153724	1152658	1146421	1141457	1139863	1136181	1132861	1133428	1133444	1132616	1132729	1137418	1137594
09 Burgos	358171	360995	364002	366900	371248	374970	375657	374826	375563	373672	365972	363874	361021	356437	355205	352723	349810	347240	347218	346355
10 Cáceres	400036	403665	406267	408703	410275	413597	415446	415083	413633	412498	411531	412899	412580	411390	410762	410242	409130	407546	408949	405616
11 Cádiz	1239435	1239889	1240284	1240175	1238492	1245164	1243519	1236739	1230594	1220467	1207343	1194062	1180817	1164374	1155724	1140793	1131346	1125105	1119802	1107484
39 Cantabria	580295	582206	585179	588656	591888	593861	593121	592250	589235	582138	572824	568091	562309	554784	549690	542275	537606	531159	528478	527137
12 Castellón/Castelló	575470	579245	582327	587508	601699	604564	604344	604274	602301	594915	573282	559761	543432	527345	518239	501237	485173	474385	467895	461712
13 Ciudad Real	502578	506888	513713	519613	524962	530250	530175	529453	527273	522343	510122	506864	500060	492914	487670	484338	478581	476633	479087	479474
14 Córdoba	788219	791610	795611	799402	802422	804498	805857	805108	803998	798822	792182	788287	784376	779870	775944	771131	769625	769237	768676	767175
15 Coruña, A	1120294	1122799	1127196	1132735	1138161	1143911	1147124	1146458	1145488	1139121	1132792	1129141	1126707	1121344	1120814	1111886	1108002	1108419	1108980	1106325
16 Cuenca	198718	201071	203841	207449	211899	218036	219138	217716	217363	215274	211375	208616	207974	204546	202982	201614	201526	201053	200963	199086
20 Gipuzkoa	719282	717832	716834	715148	713818	712097	709607	707263	705698	701056	694944	691895	688708	686513	684416	682977	680069	679370	677275	676439
17 Girona	755716	753576	753054	756156	761632	761627	756810	753046	747782	731864	706185	687331	664506	636198	619692	598112	579650	565599	553348	543191
18 Granada	912938	915392	917297	919455	919319	922928	924550	918072	907428	901220	884099	876184	860898	841687	828107	818959	812637	809004	813061	801177
19 Guadalajara	253310	252882	253686	255426	257723	259537	256461	251563	246151	237787	224076	213505	203737	193913	185474	177761	171532	165347	161669	159331
21 Huelva	518930	519596	520017	519229	520668	522862	521968	518081	513403	507915	497671	492174	483792	476707	472446	464934	461730	458998	457507	453958
22 Huesca	219702	221079	222909	224909	226329	227609	228361	228566	228409	225271	220107	218023	215864	212901	211286	208963	205955	205430	205429	204956
23 Jaén	643484	648250	654170	659033	664916	670242	670600	670761	669782	667438	664742	662751	660284	654458	651565	647387	645781	645711	649662	645792
24 León	468316	473604	479395	484694	489752	494451	497799	499284	500169	500200	497387	498223	495902	492720	495998	496655	499517	502155	506511	506365
25 Lleida	432384	434041	436029	438001	440915	443032	442308	439768	436402	426872	414015	407496	399439	385092	377639	371055	365023	361590	359361	357903
27 Lugo	333634	336527	339386	342748	346005	348902	351530	353504	355195	355549	355176	356595	357625	358452	360512	361782	364125	365619	366934	367751
28 Madrid	6507184	6466996	6436996	6454440	6495551	6498560	6489680	6458684	6386932	6271638	6081689	6008183	5964143	5804829	5718942	5527152	5372433	5205408	5145325	5091336
29 Málaga	1630615	1629298	1628973	1621968	1652999	1641098	1625827	1609557	1593068	1563261	1517523	1491287	1453409	1397925	1374890	1330010	1302240	1278851	1258084	1240580
30 Murcia	1470273	1464847	1467288	1466818	1472049	1474449	1470069	1461979	1446520	1426109	1392117	1370306	1335792	1294694	1269230	1226993	1190378	1149328	1131128	1115068
31 Navarra	643234	640647	640476	640790	644477	644566	642051	636924	630578	620377	605876	601874	593472	584734	578210	569628	556263	543757	538009	530819
32 Ourense	311680	314853	318391	322293	326724	330257	333257	335219	335642	336099	336926	338671	339555	340258	342213	343768	344623	345241	345620	344170
34 Palencia	163390	164644	166035	167609	168955	170713	171668	172510	173306	173454	173281	173153	173471	173990	175047	176125	177345	178316	179465	179623
35 Palmas, Las	1100480	1097800	1098406	1100027	1103850	1100813	1096980	1090605	1083502	1070032	1042131	1024186	1011928	987128	979606	951037	924558	897595	872669	849863
36 Pontevedra	942731	944346	947374	950919	955050	958428	963511	962472	959764	953400	947639	943117	938311	930931	927555	919934	916176	912621	908803	906298
26 Rioja, La	315381	315794	317053	319002	322027	323609	322955	322415	321702	317501	308968	306377	301084	293553	287390	281614	270400	264178	265178	263644
37 Salamanca	333603	335985	339395	342459	345548	350564	352986	353619	354608	353404	351326	353110	352414	350984	348271	347120	350209	349733	351128	349550
38 Santa Cruz de Tenerife	1007641	1004124	1001900	1004788	1014829	1017531	1029789	1027914	1020490	1005936	983820	971647	956352	928412	915262	892718	856808	818681	800020	780152
40 Segovia	154184	155652	157570	159303	161702	163701	164169	164268	164854	163899	159322	156598	155517	152640	150701	149286	147028	146613	146985	146755
41 Sevilla	1939527	1939775	1941480	1941355	1942155	1938974	1928962	1917097	1900224	1875462	1849268	1835077	1813908	1792420	1782862	1758720	1747441	1734917	1725482	1714845
42 Soria	88903	90040	91006	92221	93291	94522	95223	95258	95101	94646	93593	93503	92773	91652	90954	91487	91314	90911	91252	91593
43 Tarragona	791693	792299	795101	800962	810178	814199	811401	808420	803301	788895	757795	730466	704907	674144	654149	631156	612086	598533	588499	580245
44 Teruel	135562	136977	138932	140365	142183	143728	144607	145277	146751	146324	144046	142160	141091	139333	138686	137342	136233	136473	136849	136840
45 Toledo	686841	688672	693371	699136	706407	711228	707242	697959	689635	670203	639621	615618	598256	578060	563099	546538	536131	527965	523459	519664
46 Valencia/València	2540707	2544264	2543315	2548898	2566474	2580792	2578719	2581147	2575362	2543209	2486483	2463592	2416628	2358919	2320297	2267503	2227170	2201200	2187633	2172796
47 Valladolid	521130	523679	526288	529157	532284	534280	534874	533640	532575	529019	521661	519249	514674	510863	506302	501157	497961	495690	494594	492029
49 Zamora	177404	180406	183436	185432	188270	191612	193383	194214	195665	197221	197237	197492	198045	198524	199688	200678	202356	203469	204650	205201
50 Zaragoza	953486	950507	956006	960111	978638	978130	973325	973252	970313	955323	932502	917288	912072	897350	880118	871209	857565	848006	844571	841438
51 Ceuta	84959	84519	84263	84963	84180	84018	82376	80579	78674	77389	76603	75861	75276	74654	74931	76152	75694	75241	73704	72117
52 Melilla	86120	86026	85584	84509	83679	80802	78476	76034	73460	71448	69440	66871	65488	68016	68463	69184	68789	66263	56929	60108

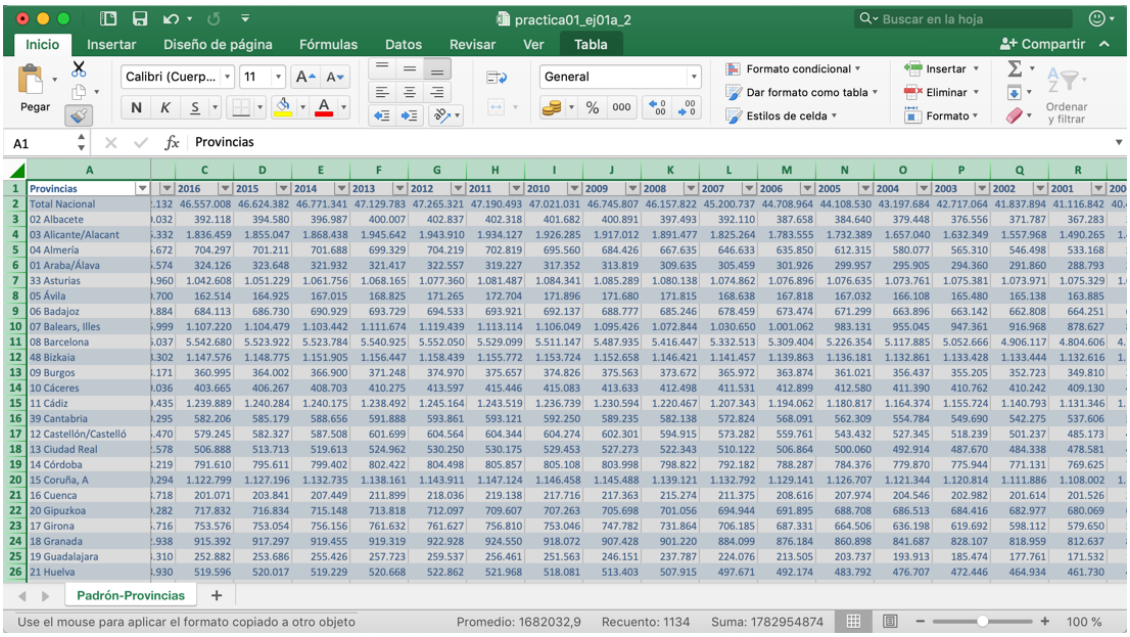
```
# otra alternativa peor:
# func_salida_tablas(datos1,VRsalida,apaisadalatex = TRUE,fuentesize=2)
```

También, nos pueden solicitar que guardemos los datos en un fichero Excel. Con ayuda de la función definida `func_salida_excel()`.

Ejemplo. Con la siguiente orden, se guarda el data.frame R: "datos1" en el fichero excel: "practica01_ej01a_2.xlsx", en la hoja excel llamada: "Padrón-Provincias".

```
func_salida_excel(datos1,"practica01_ej01a_2.xlsx",
                  titulo="Padrón-Provincias")
```

En la siguiente captura puede verse el aspecto que presenta el fichero excel con los datos guardados.



5.4.4 Tabla en varias páginas en pdf

En el siguiente código R se muestra cómo manipular con R los datos obtenidos de una consulta del INE, y finalmente se llama a la función `func_salida_tablas()` con `apaisadalatex = FALSE` para mejorar la presentación de la información en forma de tablas, ocupando en este caso varias páginas en la salida "pdf" (en la salida html no es necesario dividirla).

```
datos2a_alm = read_xlsx("datos/03003_espana.xlsx",skip = 6)
```

```
## New names:
## * `` -> ...1
```

```
indpob = which(datos2a_alm$...1 %in% c("TOTAL"))
nombres = datos2a_alm$...1
nombres2 = nombres[-indpob]
nombres2a = nombres2[3:104]
temp = datos2a_alm[indpob,]
temp$...1 = nombres2a
names(temp)[1] = "España"
temp$Hombres = as.integer(temp$Hombres)
temp$Mujeres = as.integer(temp$Mujeres)
datos2a_alm = temp

func_salida_tablas(datos2a_alm,VRsalida,apaisadalatex = FALSE)
```

España	Hombres	Mujeres
TOTAL EDADES	22832861	23739271
0 años	202547	191949
1 año	215918	204305
2 años	220711	207360
3 años	219851	208302
4 años	234734	221213
5 años	241823	227940
6 años	247625	233931
7 años	253145	237808
8 años	264822	249504
9 años	255163	240836
10 años	253536	239826
11 años	247237	235933
12 años	246671	233209
13 años	241687	230750
14 años	233598	221766
15 años	231605	220776
16 años	232739	218762
17 años	226397	214234
18 años	222734	208042
19 años	226288	214219
20 años	225155	214218
21 años	226711	216216
22 años	230450	221168
23 años	239699	231429
24 años	247245	241046
25 años	248207	242253

(continúa)

España	Hombres	Mujeres
26 años	251787	248410
27 años	257767	255303
28 años	263544	262820
29 años	268432	268735
30 años	276293	277679
31 años	287276	287730
32 años	299977	300066
33 años	312275	310590
34 años	332273	328736
35 años	349519	342680
36 años	368615	359883
37 años	380199	370733
38 años	399614	386427
39 años	407016	390262
40 años	413186	398739
41 años	411065	394642
42 años	407125	393092
43 años	397007	382843
44 años	394388	381553
45 años	386973	376019
46 años	381311	371529
47 años	375359	369110
48 años	371130	367076
49 años	371651	370389
50 años	362019	360897
51 años	356974	359478
52 años	360252	364495
53 años	341982	346508
54 años	328982	336020
55 años	319469	328653
56 años	319808	330045
57 años	309569	320901
58 años	301499	313930
59 años	291946	306519
60 años	270917	285791
61 años	261618	277036
62 años	247044	265149
63 años	246984	264136
64 años	245251	261935
65 años	227535	246500
66 años	220455	240813
67 años	225955	250952
68 años	234367	261668
69 años	211874	238645
70 años	199159	228467
71 años	205418	237758
72 años	192559	225048
73 años	186347	220563

(continúa)

España	Hombres	Mujeres
74 años	157812	188562
75 años	142225	172668
76 años	169296	211279
77 años	107109	140375
78 años	118392	157154
79 años	127686	173693
80 años	131745	187609
81 años	123150	176949
82 años	114633	170368
83 años	109110	167653
84 años	99734	158807
85 años	86454	141954
86 años	76297	134400
87 años	64059	117321
88 años	54613	105728
89 años	43442	88649
90 años	36048	77915
91 años	27819	64008
92 años	21747	53196
93 años	16760	42363
94 años	12171	33889
95 años	8448	24525
96 años	5605	17875
97 años	3509	11814
98 años	2293	8474
99 años	1442	5913
100 años y más	3199	12182

Guardamos el data.frame "datos2a_alm" en el fichero excel "practica01_ej02a_2.xlsx" con la siguiente llamada a la función `func_salida_excel()`:

```
func_salida_excel(datos2a_alm,"practica01_ej02a_2.xlsx",
                  titulo="Padrón-España-Edad-Sexo 2017")
```

Para guardar objetos R en un fichero RData, se usa la función `save()`:

```
save(datos1,datos2a_alm,file="datosfinal.RData")
# se podrían cargar todos esos objetos con la función:
# load("datosfinal.RData)
```

6 Crear gráficos con ggplot2. Especificar leyendas

En este apartado haremos una introducción muy básica a las capacidades gráficas del paquete "ggplot2" que forma parte del sistema "tidyverse" y es posiblemente el paquete R de tratamiento gráfico más usado en la actualidad.

La idea que se usa en este paquete es añadir paso a paso las distintas capas que constituirán el gráfico final:

1. Especificar los datos (tibble o data.frame)
2. Indicar las columnas que se van a utilizar y el papel que desempeñarán en el gráfico (`aes()`)
3. Añadir el tipo o tipos de gráficos que se van a representar
4. Especificar las características particulares del gráfico, en particular las leyendas o texto explicativo que llevará.

Veremos a través de ejemplos cómo se construyen gráficos con el paquete "ggplot2", en concreto construiremos los siguientes tipos de gráficos:

- Diagramas de barras o columnas
- Diagramas de líneas

6.1 Diagrama de barras o columnas

Utilizaremos los datos creados en el segundo ejemplo de importación de datos de ficheros Excel.

```
datos_CCAA = datos2_mej %>%
  dplyr::group_by(CCAA) %>%
  dplyr::summarise(TOTALCCAA = sum(TOTAL),
            TOTALVarón = sum(Varon),
            TOTALMujer = sum(Mujer)) %>%
  dplyr::arrange(desc(TOTALCCAA))

datos_CCAA %>%
  glimpse()
```

```
## Rows: 19
## Columns: 4
## $ CCAA       <chr> "Andalucía", "Cataluña", "Madrid (Comunidad de)", "Comunida~
## $ TOTALCCAA  <dbl> 7357558, 6343110, 5423384, 4162776, 2695880, 2456474, 20825~
## $ TOTALVarón <dbl> 3622066, 3106531, 2609746, 2046984, 1294378, 1209874, 10178~
## $ TOTALMujer <dbl> 3735492, 3236579, 2813638, 2115792, 1401502, 1246600, 10647~
```

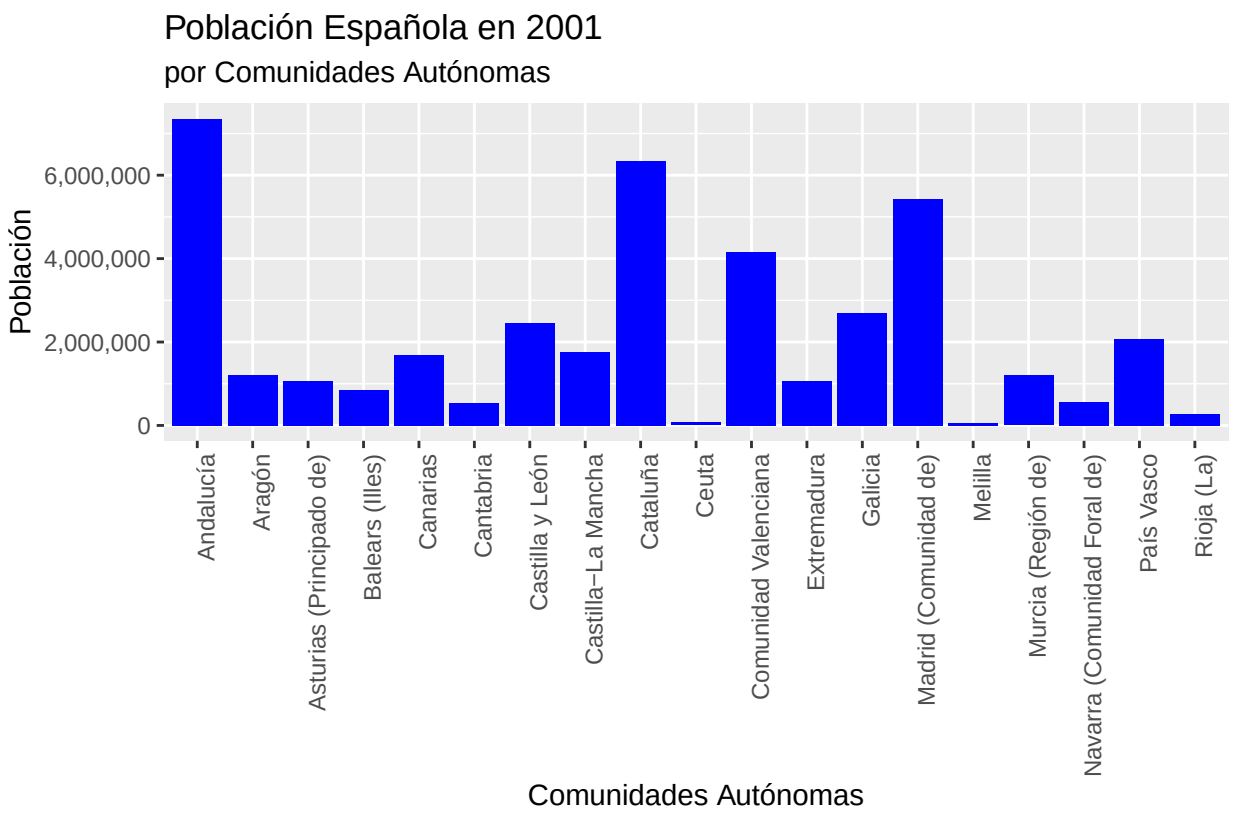
Ejemplo. En el siguiente código veremos como se construye un diagrama de columnas de la variable "Poblacion" de las comunidades autónomas (CCAA) para el censo del 2001.

Los pasos para su construcción son:

1. Usar la llamada a la función: `ggplot()`
   - con los datos que utilizaremos (en el ejemplo: `datos_CCAA`)
   - y la función `aes()`, en la que se especifican quién irá al eje X e Y.
2. Sumamos o añadimos la geometría de representación o el tipo de gráfico. En este caso:
   - 'geom_col()"
   - y especificamos algunas características. Para este tipo de gráfico el color de las barras a través de "`fill`".
3. Y por último, usamos la función: `labs()`, en la que indicaremos las distintas leyendas del gráfico.

Podemos verlo en el siguiente código R:

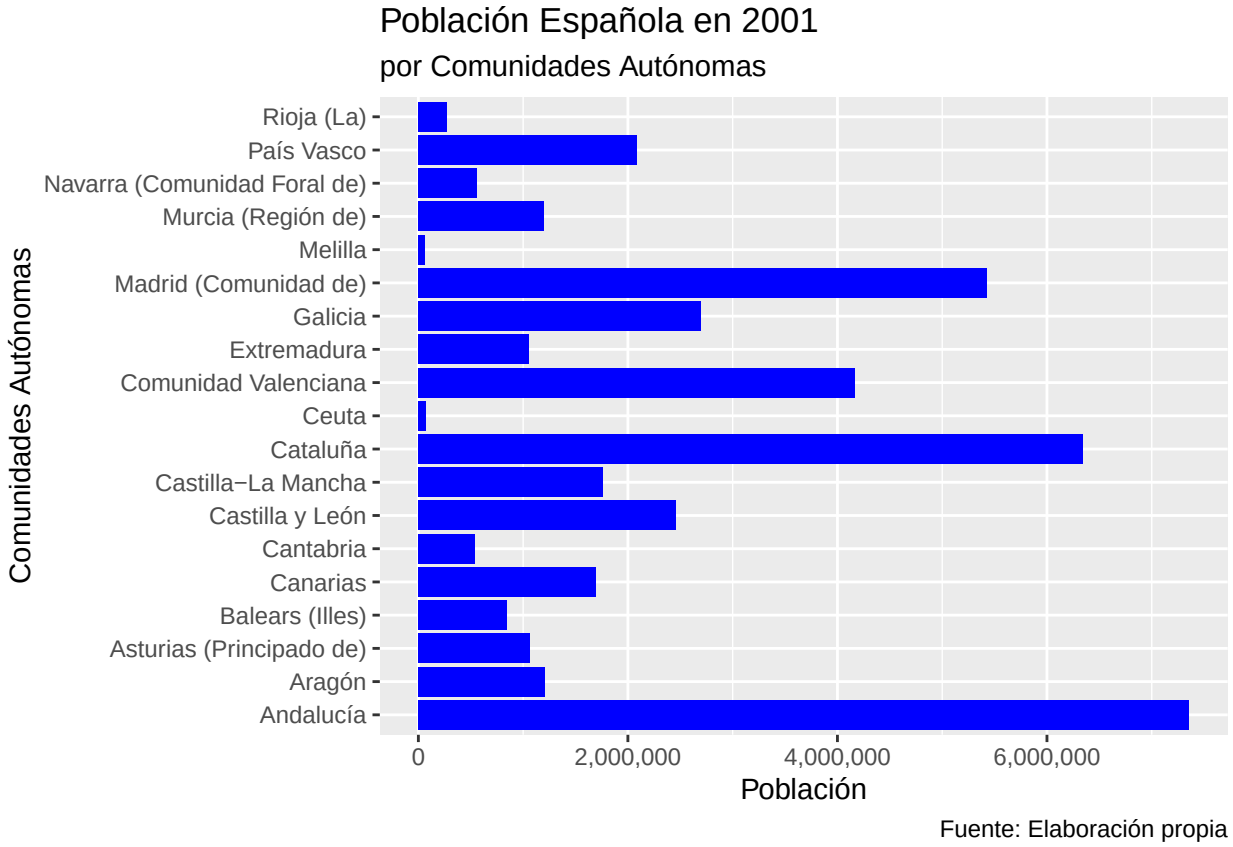
```
#library(ggplot2)
ggplot(datos_CCAA,aes(x=CCAA,y=TOTALCCAA)) +
  geom_col(fill="blue") +
    labs(title="Población Española en 2001",
         subtitle="por Comunidades Autónomas",
        y="Población",x="Comunidades Autónomas",
        caption="Fuente: Elaboración propia") +
  scale_y_continuous(labels = scales::comma) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



Ejemplo. Si queremos hacer la misma representación pero intercambiando los ejes, añadiremos `coord_flip()`.

En este gráfico además hemos modificado la forma de representar los números de las etiquetas del eje en las que aparecen las cifras de población.

```
ggplot(datos_CCAA,aes(x=CCAA,y=TOTALCCAA)) +
  geom_col(fill="blue") +
    labs(title="Población Española en 2001",
         subtitle = "por Comunidades Autónomas",
       y="Población",x="Comunidades Autónomas",
       caption="Fuente: Elaboración propia") +
  scale_y_continuous(labels = scales::comma) +
  theme(axis.text.y = element_text(angle = 0, hjust = 1)) +
  coord_flip()
```

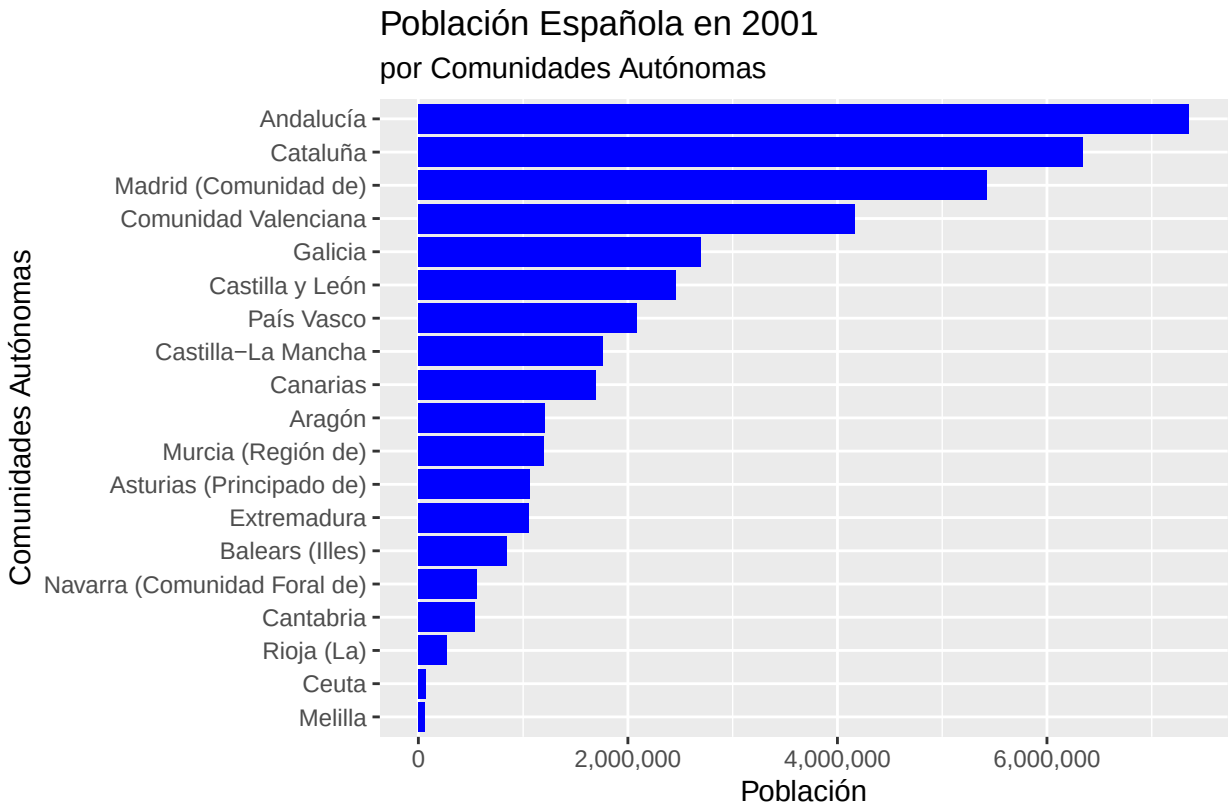


6.1.1 Ordenar barras por orden descendente de valor

Ejemplo. Para presentar las columnas siguiendo algún tipo de orden (por defecto, las ordena según el orden alfabético) se puede utilizar la función `reorder()`. Cuando se llama a `reorder()` el primer argumento indica la columna que se usará para las etiquetas, y la segunda columna será para indicar el orden en el que aparecerán (si se quiere presentar en orden contrario se debe colocar un signo "-" delante del segundo argumento).

```
ggplot(datos_CCAA,aes(x=reorder(CCAA,TOTALCCAA),y=TOTALCCAA)) +
  geom_col(fill="blue") +
    labs(title="Población Española en 2001",
         subtitle = "por Comunidades Autónomas",
       y="Población",x="Comunidades Autónomas",
       caption="Fuente: Elaboración propia") +
  scale_y_continuous(labels = scales::comma) +
```

```
  theme(axis.text.y = element_text(angle = 0, hjust = 1)) +
  coord_flip()
```



Fuente: Elaboración propia

Esta representación nos permite identificar rápidamente la ordenación de las comunidades autónomas según el número de habitantes.

6.2 Diagrama de líneas

6.2.1 Ejemplo 1

Veamos ahora como representar un diagrama de líneas. El procedimiento es prácticamente el mismo que el anterior, pero se cambiará la geometría o tipo de gráfico, que en este caso es: `geom_line()`.

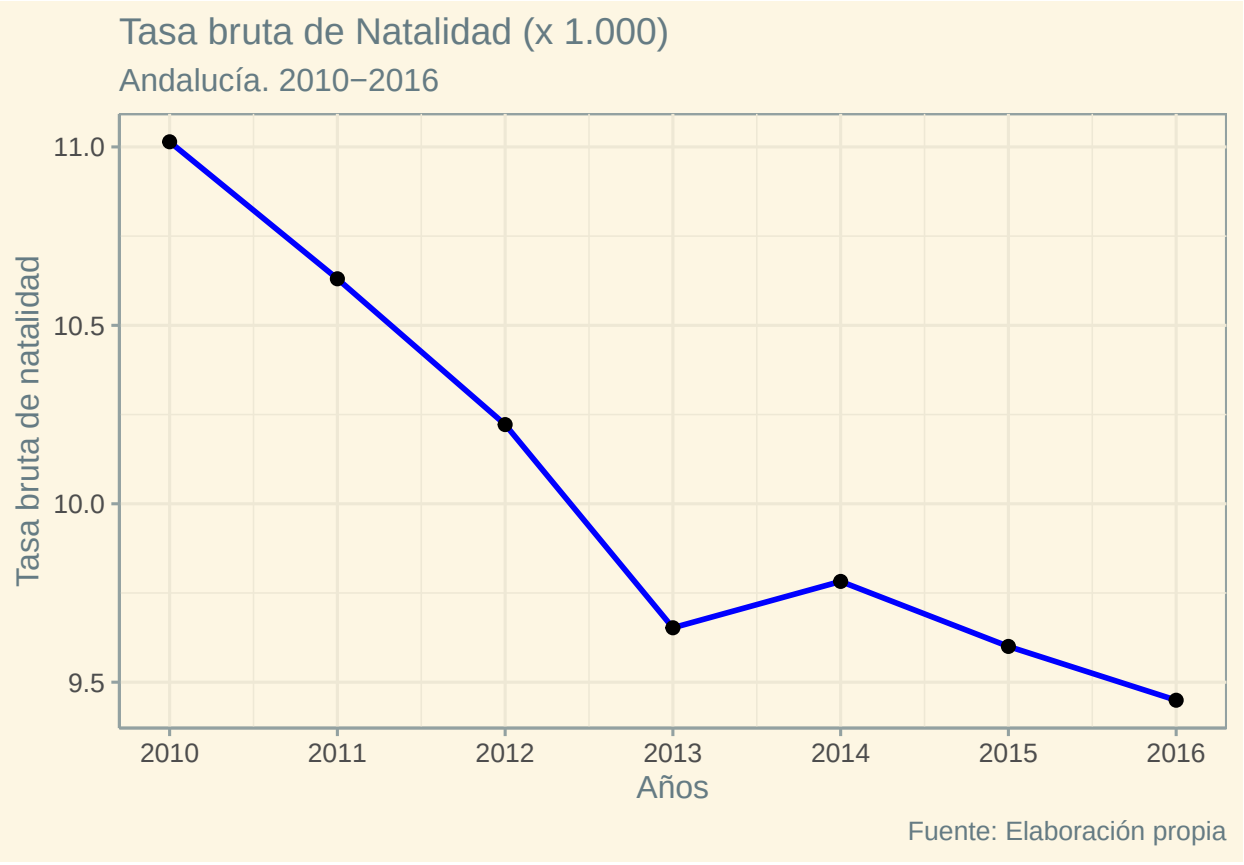
Vamos a construir un diagrama de líneas con el que representaremos la evolución de la tasa bruta de natalidad de 2010 a 2016. Para ello necesitamos previamente obtener esa información en un formato como el siguiente:

```
df.rep = data.frame(Año = 2016:2010,
                    TBN = TBN_2016a2010)
head(df.rep)
```

```
##    Año       TBN
## 1 2016  9.449450
## 2 2015  9.600260
## 3 2014  9.782435
## 4 2013  9.652501
## 5 2012 10.221912
## 6 2011 10.630451
```

El código para el gráfico se recoge a continuación:

```
#library(ggplot2)
#library(ggthemes)
ggplot(df.rep, aes(x = Año, y=TBN)) +
  geom_line(alpha = 1,linetype = "solid", colour="blue",size = 1) +
  geom_point(size = 2) +
  labs(title="Tasa bruta de Natalidad (x 1.000) ",
       subtitle = "Andalucía. 2010-2016",
       y="Tasa bruta de natalidad",
       x="Años",
       caption="Fuente: Elaboración propia") +
  #scale_y_continuous(labels = scales::comma,breaks = seq(0,3.5,by=0.25)) +
  scale_x_continuous(breaks = seq(2010,2016,by=1)) +
  theme(axis.text.y = element_text(angle = 0, hjust = 1)) +
  theme_solarized()
```



En este gráfico también hemos sumado la geometría `geom_point()`, para añadir los puntos sobre el gráfico de líneas para resaltar sus valores.

6.2.2 Ejemplo 2

En este nuevo ejemplo, representaremos las defunciones teóricas (función *dx* de la tabla de vida) en Andalucía en 2015, para edades simples, distinguiendo según el sexo.

Los datos que se utilizarán por sexo son:

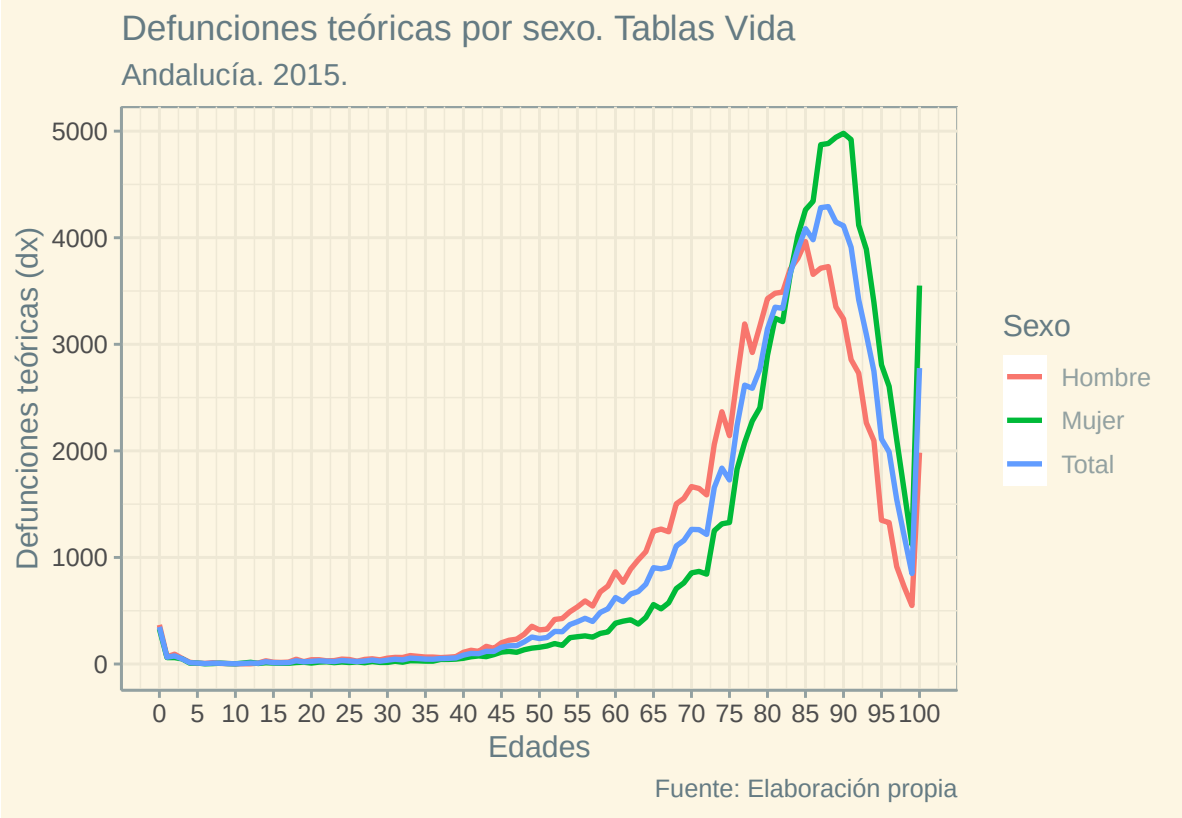
```
df.rep = data.frame(
  Edades = c(tmortalidad2015_AndAmb$x,
```

```
            tmortalidad2015_AndHom$x,
            tmortalidad2015_AndHom$x),
  dx = c(tmortalidad2015_AndAmb$dx,
         tmortalidad2015_AndHom$dx,
         tmortalidad2015_AndMuj$dx),
  Sexo = c(rep("Total",nrow(tmortalidad2015_AndAmb)),
    rep("Hombre",nrow(tmortalidad2015_AndHom)),
           rep("Mujer",nrow(tmortalidad2015_AndMuj)))
)
df.rep %>%
  slice(1:6,102:104,203:205)
```

```
##    Edades  dx   Sexo
## 1       0 348  Total
## 2       1  64  Total
## 3       2  76  Total
## 4       3  48  Total
## 5       4  12  Total
## 6       5   9  Total
## 7       0 367 Hombre
## 8       1  68 Hombre
## 9       2  92 Hombre
## 10      0 329  Mujer
## 11      1  60  Mujer
## 12      2  60  Mujer
```

En el argumento `aes()` se ha utilizado el papel "colour=Sexo", para que se haga el diagrama de líneas con un color distinto para cada modalidad de la variable "Sexo". El código para constuir el gráfico de líneas sería el siguiente:

```
ggplot(df.rep, aes(x = Edades, y=dx, colour = Sexo))  +
  geom_line(alpha = 1,
            linetype = "solid",
            size = 1) +
  labs(title="Defunciones teóricas por sexo. Tablas Vida",
       subtitle = "Andalucía. 2015.",
       y="Defunciones teóricas (dx)",
       x="Edades",
       caption="Fuente: Elaboración propia") +
  #scale_y_continuous(labels = scales::comma,breaks = seq(0,1,by=0.1)) +
  scale_x_continuous(breaks = c(0,seq(5,100,by=5))) +
  theme(axis.text.y = element_text(angle = 0, hjust = 1)) +
  theme_solarized()
```



Se ha añadido la función "`theme_solarized()`" que permite cambiar el aspecto general del gráfico (el tema "solarized" fija un fondo amarillo y otras características). Existen otros muchos tipos de temas predefinidos (consultar la ayuda de "ggplot2" y del paquete "ggthemes").

7 Crear diagramas de Lexis

Para facilitar la elaboración de diagramas de Lexis en R se recomienda el uso del paquete R: "LexisPlotR". Este paquete se basa en el paquete "ggplot2", por tanto, podrían añadirse nuevas capas o características, con todas las posibilidades que admite "ggplot2".

Este paquete se puede instalar como cualquier otro paquete R. Una vez instalado, podría usarse con tan solo cargarlo.

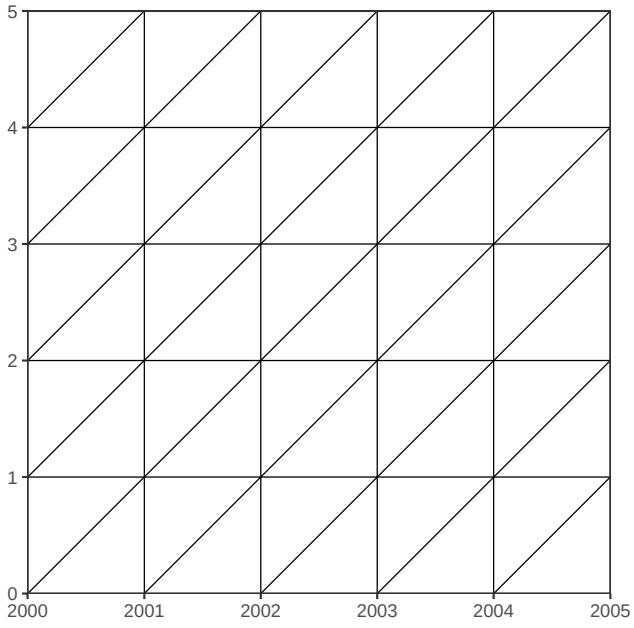
```
library(LexisPlotR)
```

Veamos algunos ejemplos de lo que se puede representar con este paquete extraídas de la página en Github de este paquete.

Nota importante. En este manual se ha usado la versión 0.40 de "LexisPlotR", pero la versión anterior fue la 0.32. Se han producido cambios importantes, que pueden producir que el código usado para la versión 0.32 no funcione para la versión 0.40. **Fundamentalmente** es que en la versión 0.32 se utilizaba el punto "." para separar los elementos y en la versión 0.40 se utiliza el guión bajo "_". En los ejemplos, que se mostrarán a continuación, aparecerá el código de la versión 0.40.

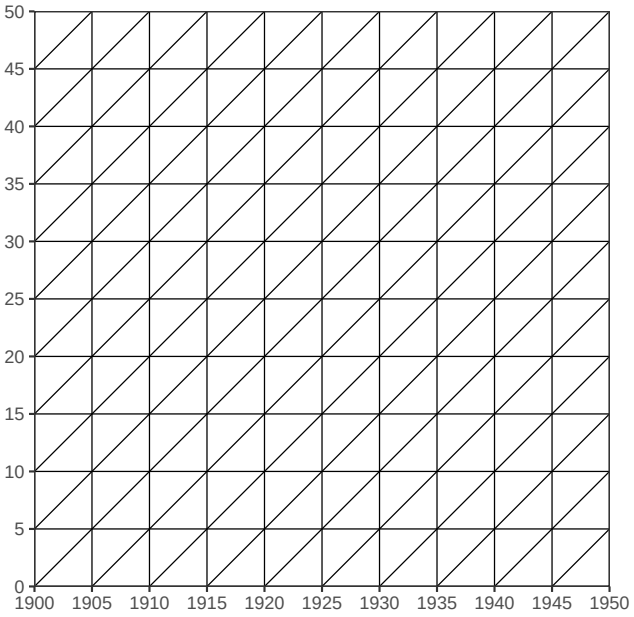
Ejemplo. Con la función `lexis_grid()` se representa un diagrama de Lexis desde el año 2000 a 2005, representando las edades desde 0 a 5.

```
#lexis.grid(year.start = 2000, year.end = 2005, age.start = 0, age.end = 5) # 0.32
lexis_grid(year_start = 2000, year_end = 2005, age_start = 0, age_end = 5) # 0.40
```



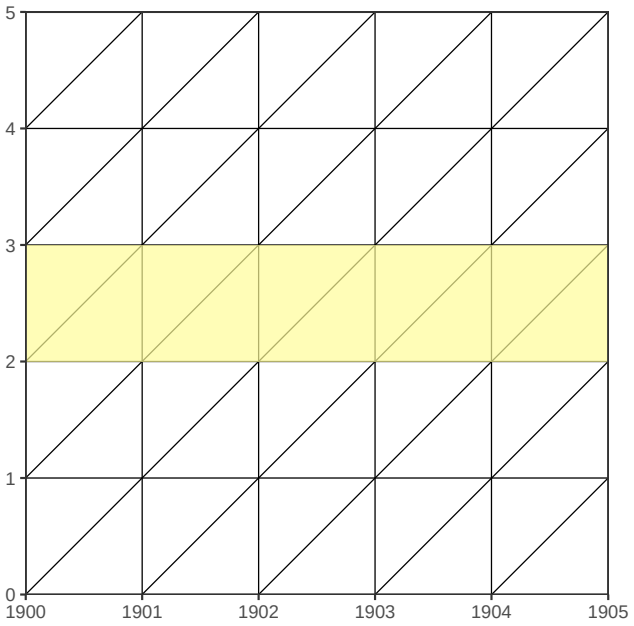
Ejemplo.

```
lexis_grid(year_start = 1900, year_end = 1950, age_start = 0, age_end = 50, delta = 5)
```



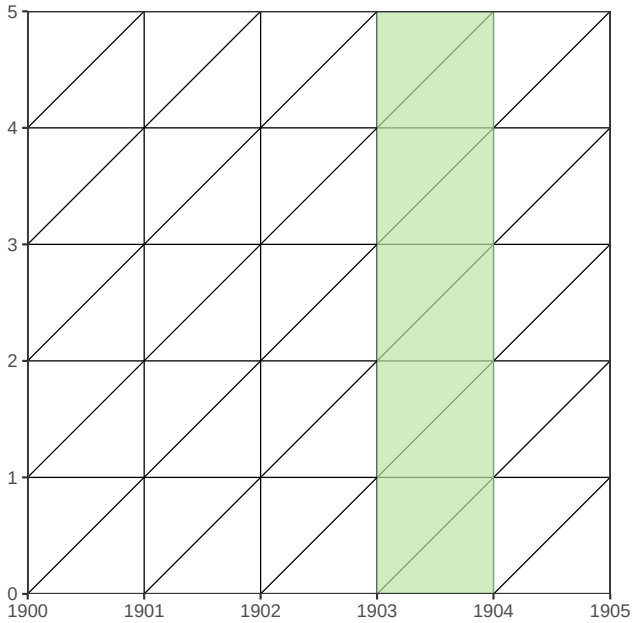
Ejemplo.

```
lexis <- lexis_grid(year_start = 1900, year_end = 1905, age_start = 0, age_end = 5)  # 0.40
lexis_age(lg = lexis, age = 2)  # 0.40
```



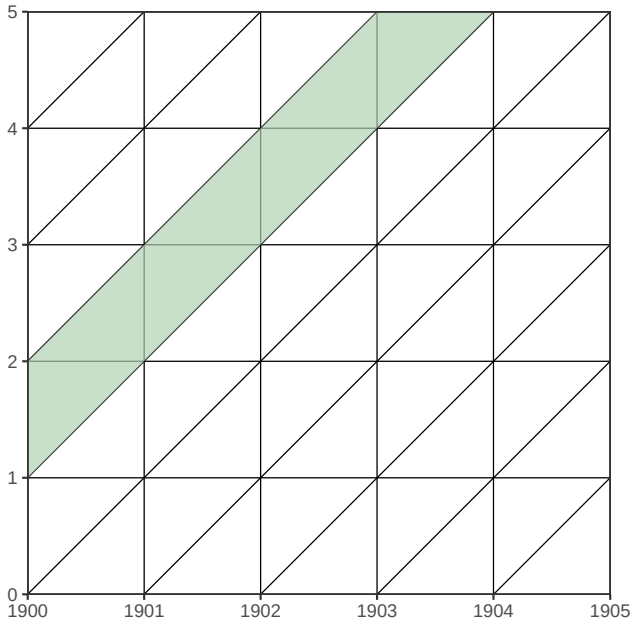
Ejemplo.

```
lexis <- lexis_grid(year_start = 1900, year_end = 1905, age_start = 0, age_end = 5)
lexis_year(lg = lexis, year = 1903)
```



Ejemplo.

```
lexis <- lexis_grid(year_start = 1900, year_end = 1905, age_start = 0, age_end = 5)
lexis_cohort(lg = lexis, cohort = 1898)
```

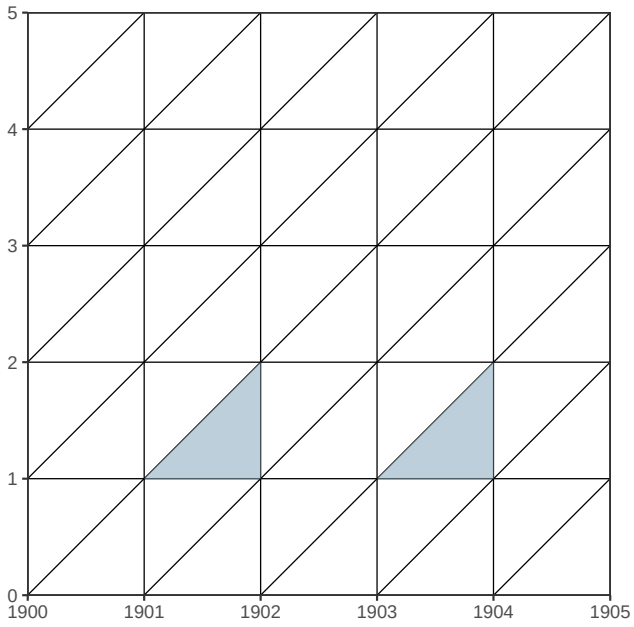


Ejemplo.

```
lexis <- lexis_grid(year_start = 1900, year_end = 1905, age_start = 0, age_end = 5)

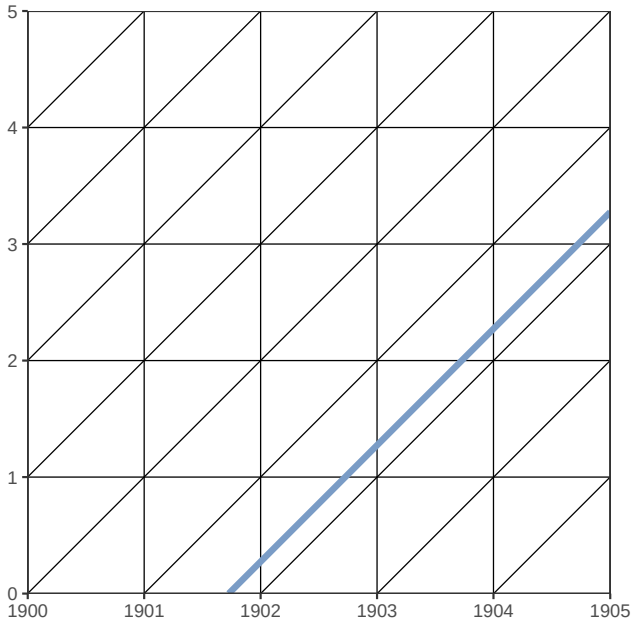
polygons <- data.frame(group = c(1, 1, 1, 2, 2, 2),
                       x = c("1901-01-01", "1902-01-01", "1902-01-01", "1903-01-01",
                             "1904-01-01", "1904-01-01"),
                       y = c(1, 1, 2, 1, 1, 2))

lexis_polygon(lg = lexis, x = polygons$x, y = polygons$y, group = polygons$group)
```



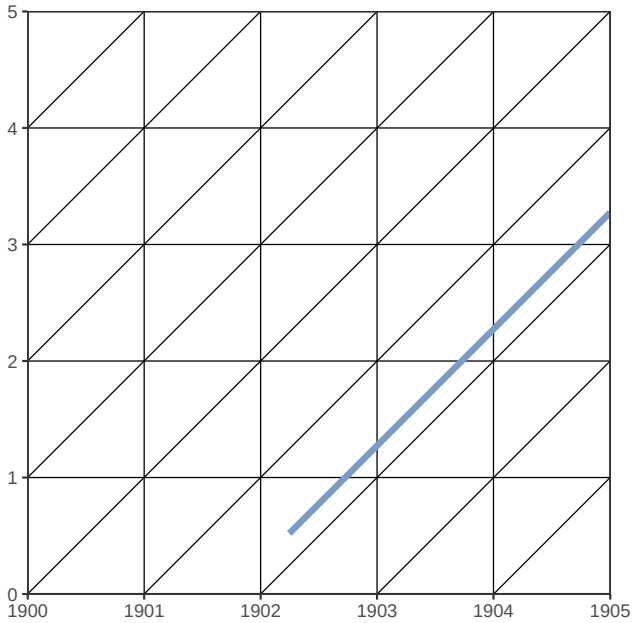
Ejemplo.

```
lg <- lexis_grid(year_start = 1900, year_end = 1905, age_start = 0, age_end = 5)
lexis_lifeline(lg = lg, birth = "1901-09-23", lwd = 1.5)
```



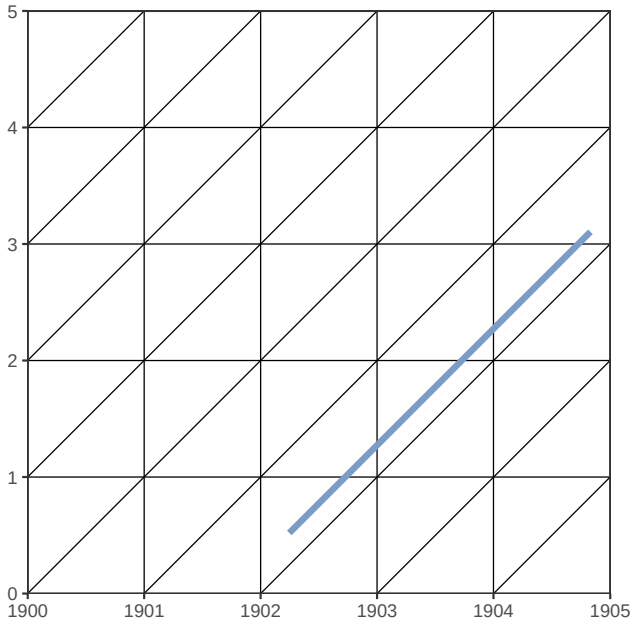
Ejemplo.

```
lexis_lifeline(lg = lg, birth = "1901-09-23", entry = "1902-04-01", lwd = 1.5)
```



Ejemplo.

```
lexis_lifeline(lg = lg, birth = "1901-09-23", entry = "1902-04-01", exit = "1904-10-31", lwd = 1.5)
```

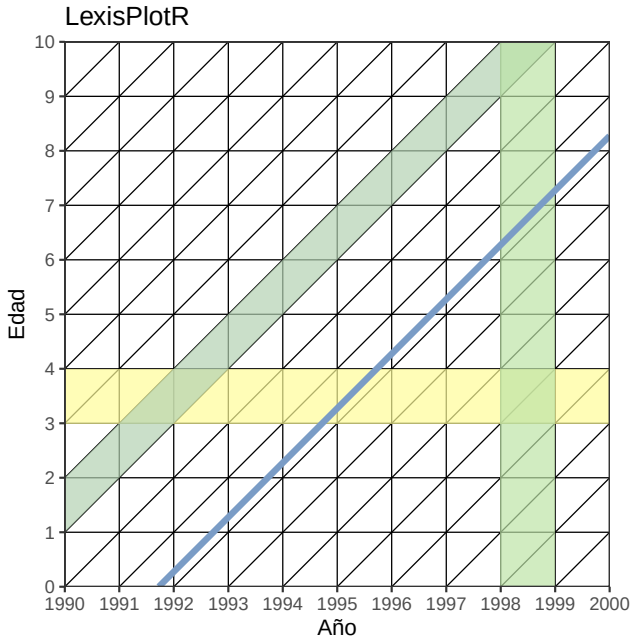


Ejemplo. Se puede utilizar con el operador de "tubería" del paquete "magrittr" (incluido en tidyverse) y "sumar" elementos del paquete ggplot2.

```
library(magrittr)
library(ggplot2)

p <- lexis_grid(year_start = 1990, year_end = 2000, age_start = 0, age_end = 10) %>%
  lexis_age(age = 3) %>%
  lexis_cohort(cohort = 1988) %>%
  lexis_year(year = 1998) %>%
  lexis_lifeline(birth = "1991-09-23", lwd = 1.5)

p <- p + labs(x = "Año", y = "Edad", "title" = "LexisPlotR")
p
```



A continuación se facilitan algunos enlaces útiles sobre este paquete:

- LexisPlotR en CRAN
- LexisPlotR en Github

8 Crear pirámides de población

Para facilitar la elaboración de pirámides de población se han definido una serie de funciones, las cuales se encuentran definidas en el fichero **"funciones_piramides_ggplot2.R"**. Las funciones son las siguientes:

- `func_piramide_ggplot2()`
- `func_piramide_ggplot2_linea()`
- `func_piramides_enfrentadas_ggplot2()`
- `func_piramide_superpuestas_ggplot2()`
- `func_piramide_compuestasCateg_ggplot2()`

Además existen dos funciones auxiliares:

- `func_agrupar_variable()`
- `func_etiquetas_gruposEdad()`

La sintaxis de estas funciones es la siguiente:

```
func_piramide_ggplot2 = function(datosPiramide,
                                 porcentajes=TRUE,
                                 etiquetas=FALSE,etiquetas.size=4,
                                 UsaCaso=FALSE,
                                 etiq.hombre="Hombre",etiq.mujer="Mujer",
                                 colorear="Sexo",colores=NULL)
```

```
func_piramide_ggplot2_linea = function(datosPiramide,
                                       porcentajes=TRUE,
                                       etiquetas=FALSE,etiquetas.size=4,
                                       UsaCaso=FALSE,
                                       etiq.hombre="Hombre",etiq.mujer="Mujer",
                                       colorear="Sexo",colores=NULL)
```

```
func_piramides_enfrentadas_ggplot2 = function(datosPiramide,
                                              porcentajes=TRUE,
                                              etiquetas=FALSE,etiquetas.size=4,
                                              UsaCaso=TRUE,
                                              etiq.hombre="Hombre",etiq.mujer="Mujer",
                                              colorear="Sexo",colores=NULL,
                                              nfilas=NULL,ncols=NULL)
```

```
func_piramide_superpuestas_ggplot2 = function(datosPiramide,
                                 porcentajes=TRUE,
                                 etiquetas=FALSE,etiquetas.size=4,
                                 colores = NULL,
                                 transparente=FALSE,
                                 alfa=0.1,bar.size=1,
                                 etiq.hombre="Hombre",etiq.mujer="Mujer")
```

```
func_piramide_compuestasCateg_ggplot2 = function(datosPiramide,
                                              porcentajes=TRUE,
                                              etiquetas=FALSE,etiquetas.size=4,
                                              colores = NULL,ordeninverso=FALSE,
                                              alfa=1,bar.size=1,
                                              etiq.hombre="Hombre",etiq.mujer="Mujer")
```

Nota: estas funciones construyen gráficos a través del paquete "ggplot2", por lo que es posible "sumarle" alguna característica adicional disponible en este paquete R.

Los **datos deben** de ir en un data.frame o tibble, y deben contener las siguientes columnas:

- **Edad**: pueden ser edades simples o grupos de edad (character o factor).
- **Sexo**: por defecto espera que en esta columna se usen las etiquetas: "Hombre" y "Mujer" (character o factor).
- **Poblacion**: número de habitantes (numeric).
- **Caso**: variable categórica para pirámides "compuestasCateg" (character o factor).

A continuación veremos cómo usarlas mediante ejemplos. En primer lugar tendríamos que cargar el fichero de funciones:

```
source("funciones_piramides_ggplot2.R")
```

8.1 Ejemplo 1

En este primer ejemplo, vamos a construir una pirámide de población de España en 2017. Para ello vamos a usar la función `func_piramide_ggplot2()`.

1. Recurriendo a la web del INE, obtener la siguiente información en formato "px":
    - Población de España residente por fecha, sexo y edad a 1 de enero de 2002 y 2017.

Paso 1. Descargamos el fichero "px" de la página: http://www.ine.es/jaxiT3/Tabla.htm?t=9663&L=0 (INEBASE, Demografía y Población, Cifras de Población, Resultados nacionales)

Paso 2. Importamos los datos en R:

```
dfej02a <- as.data.frame(read.px("datos/9663.px"),stringsAsFactors=FALSE)
head(dfej02a)
```

```
##               Periodo        Sexo Edad.simple    value
## 1  1 de julio de 2018 Ambos sexos       Total 46733038
## 2 1 de enero de  2018 Ambos sexos       Total 46658447
## 3  1 de julio de 2017 Ambos sexos       Total 46532869
## 4 1 de enero de  2017 Ambos sexos       Total 46527039
## 5  1 de julio de 2016 Ambos sexos       Total 46449874
## 6 1 de enero de  2016 Ambos sexos       Total 46440099
```

```
## `summarise()` has grouped output by 'Sexo'. You can override using the `.groups` argument.
## `summarise()` has grouped output by 'Sexo'. You can override using the `.groups` argument.
```

Se han realizado las operaciones necesarias para que el data.frame tenga al menos las 3 columnas necesarias ("Edad", "Sexo", "Población"), como puede verse a continuación:

```
head(dfPir2017)
```

```
##   Edadchar    Sexo Poblacion Edad
## 1   0 años Hombres  210605.9    0
## 2   0 años Mujeres  199294.2    0
## 3    1 año Hombres  218041.9    1
## 4    1 año Mujeres  205462.2    1
## 5   2 años Hombres  223029.6    2
## 6   2 años Mujeres  209112.7    2
```

Construimos la pirámide de población llamando a la función básica:

```
func_piramide_ggplot2(dfPir2017,
                      #etiquetas = T,etiquetas.size = 2,
                      etiq.hombre = "Hombres",etiq.mujer = "Mujeres") +
    labs(title = "Pirámide de Población de España en 2017") +
    scale_x_discrete(# si la variable edad fuera numeric debería usarse scale_x_continuous
```

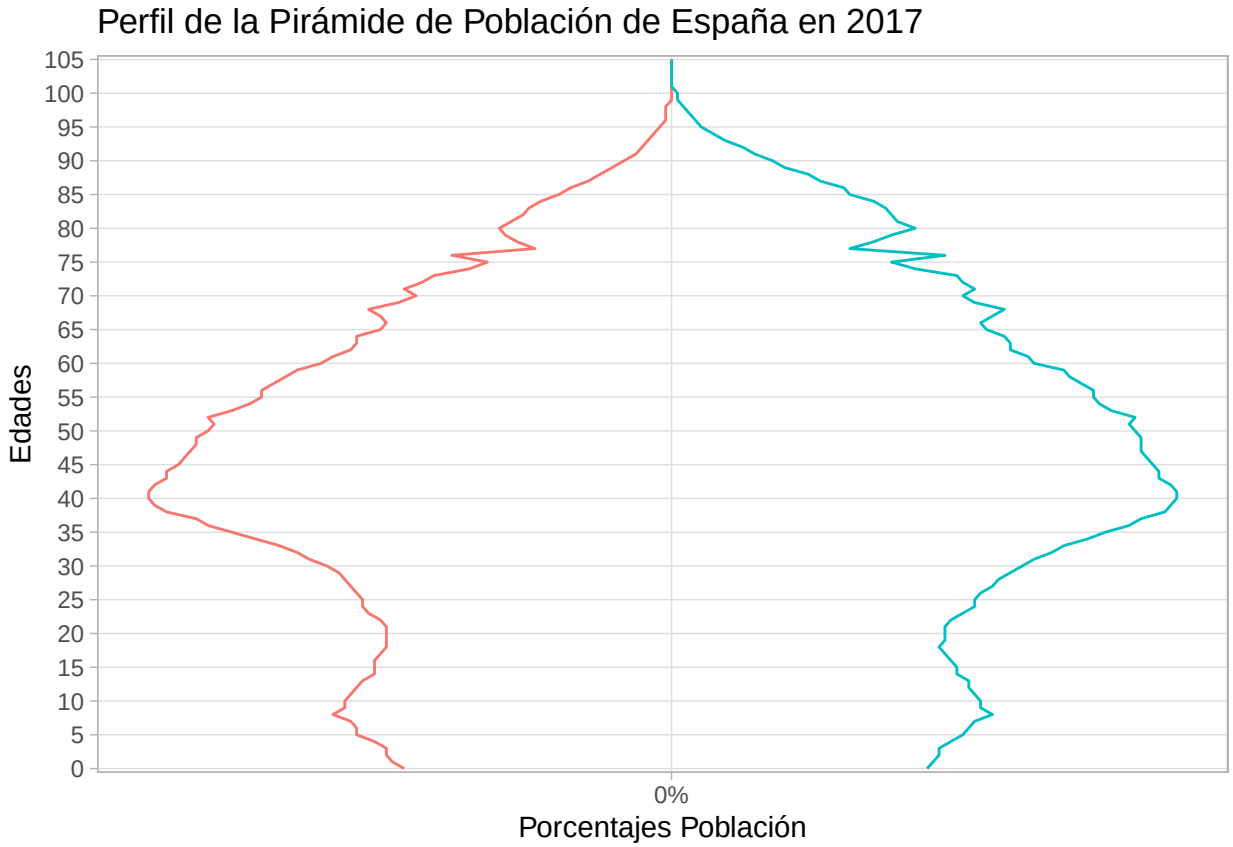
```
          breaks = seq(0,105,by=5),
          labels = paste0(as.character(seq(0,105,by=5)), ""))
```

Por defecto, representa la pirámide de poblaciones utilizando los porcentajes. Si se quisiera mostrar los valores absolutos tendríamos que añadir el argumento: `porcentajes=FALSE`. Se puede personalizar los colores usados (`colores=c("green","blue")`), y también qué variable se usa para colorear, por defecto se usa la variable "Sexo" (`colorear="Edad"`). Se podrían mostrar los valores sobre cada barra activando el argumento `etiquetas=TRUE`.

También podríamos representar solamente el **perfil de la pirámide de población** con ayuda de la función `func_piramide_ggplot2_linea()`:

```
func_piramide_ggplot2_linea(dfPir2017,colorear = "Sexo",
                            etiq.hombre = "Hombres",etiq.mujer = "Mujeres") +
    labs(title = "Perfil de la Pirámide de Población de España en 2017") +
    scale_x_discrete(# si la variable edad fuera numeric debería usarse scale_x_continuous
          breaks = seq(0,105,by=5),
          labels = paste0(as.character(seq(0,105,by=5)), "")) +
  guides(colour=FALSE)
```

```
## Warning: `guides(<scale> = FALSE)` is deprecated. Please use `guides(<scale> =
## "none")` instead.
```



8.2 Ejemplo 2

En este otro ejemplo se representará la pirámide de población de España en 2002.

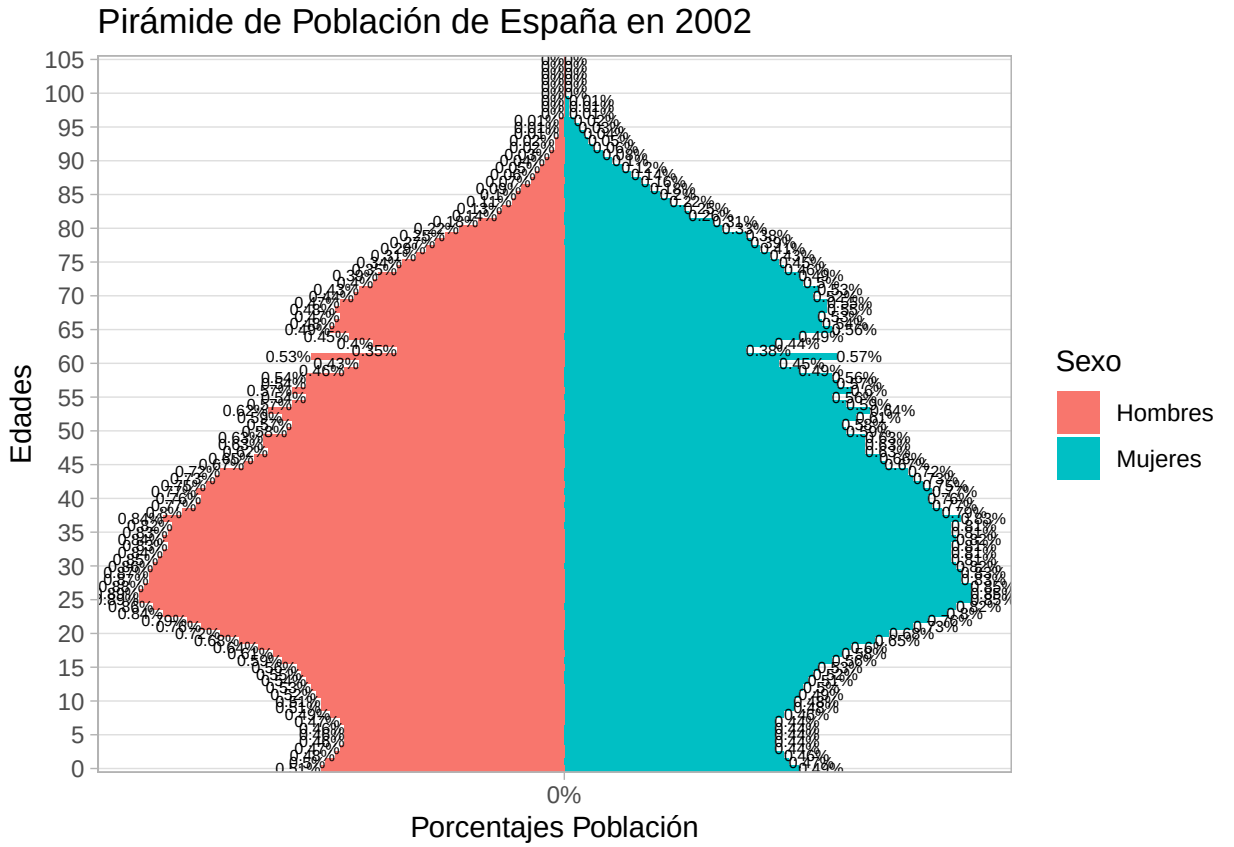
Haciendo las convenientes operaciones, se obtiene el data.frame con los datos necesarios:

```
head(dfPir2002)
```

```
##   Edadchar    Sexo Poblacion Edad
## 1   0 años Hombres  210646.4    0
## 2   0 años Mujeres  201046.8    0
## 3    1 año Hombres  204827.9    1
## 4    1 año Mujeres  193282.1    1
## 5   2 años Hombres  197158.7    2
## 6   2 años Mujeres  187579.5    2
```

En este caso, dibujamos la pirámide de población mostrando los valores sobre las barras:

```
func_piramide_ggplot2(dfPir2002,
          etiquetas = T, etiquetas.size = 2,# etiquetas solamente para edades sin agrupar
          porcentajes = T,
          etiq.hombre = "Hombres",etiq.mujer = "Mujeres") +
  labs(title = "Pirámide de Población de España en 2002") +
    scale_x_discrete( # si la variable edad fuera numeric debería usarse scale_x_continuous
          breaks = seq(0,105,by=5),
          labels = paste0(as.character(seq(0,105,by=5)), ""))
```



Esta forma de representar la pirámide con edades simples no es muy adecuada por la dificultad de presentar tantos valores sobre un gráfico.

8.3 Ejemplo 3 (pirámide superpuesta)

En este otro ejemplo, vamos a comparar las dos piramides de población anteriores utilizando una pirámide superpuesta, en la que se muestre simultáneamente las pirámides de 2002 y 2017.

En primer lugar, preparamos los datos para que se representen correctamente. La columna "Caso" debe contener la información de a qué población se refiere.

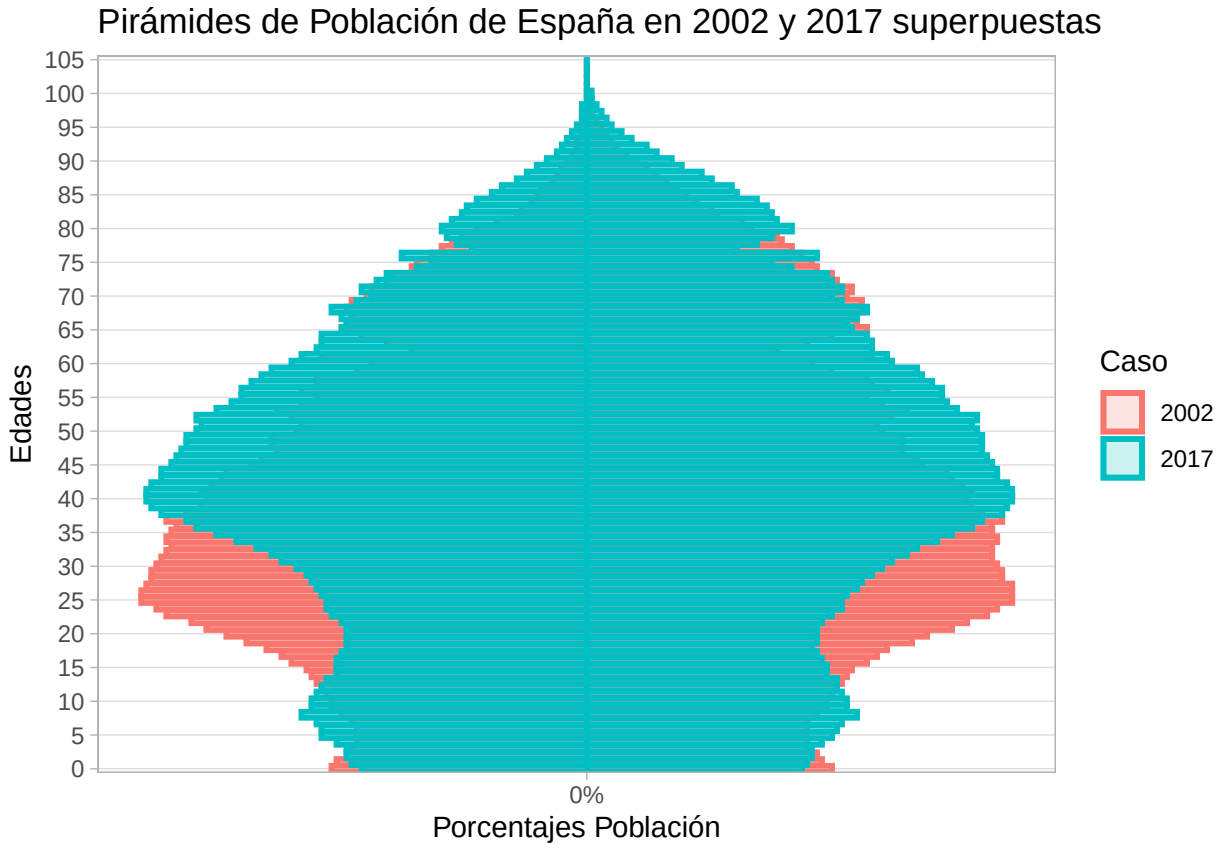
```
dfPir2002y2017 = rbind(dfPir2002,dfPir2017)
dfPir2002y2017$Caso = c(rep(2002,nrow(dfPir2002)),rep(2017,nrow(dfPir2017)))
head(dfPir2002y2017)
```

```
##   Edadchar    Sexo Poblacion Edad Caso
## 1   0 años Hombres  210646.4    0 2002
## 2   0 años Mujeres  201046.8    0 2002
## 3    1 año Hombres  204827.9    1 2002
## 4    1 año Mujeres  193282.1    1 2002
## 5   2 años Hombres  197158.7    2 2002
## 6   2 años Mujeres  187579.5    2 2002
```

Para representar la pirámide de población superpuesta usaremos la función `func_piramide_superpuestas_ggplot2()`:

```
func_piramide_superpuestas_ggplot2(dfPir2002y2017,
                     etiq.hombre = "Hombres",etiq.mujer = "Mujeres",
                     transparente = T) +
 labs(title = "Pirámides de Población de España en 2002 y 2017 superpuestas") +
```

```
scale_x_discrete(
        breaks = seq(0,105,by=5),
        labels = paste0(as.character(seq(0,105,by=5)), ""))
```



9 Crear mapas demográficos

Para facilitar la elaboración de mapas demográficos o cartogramas se han definido una serie de funciones, las cuales se encuentran definidas en el fichero **"funciones_mapas.R"** (este fichero incluye referencias a otros ficheros R de funciones asociadas). Estas funciones dependen principalmente de los paquetes R: "SIANE", "raster" y "maptools". Estas funciones nos van a permitir representar mapas de España (o de una Comunidad Autónoma o provincia), características a un ámbito territorial de: comunidades autónomas, provincias, y municipios. Las funciones son las siguientes:

- `func_mapademografia_prov()`
- `func_mapademografia_ccaa()`
- `func_mapademografia_municipios()`

También incluye algunas funciones auxiliares que facilitarán el trabajo:

- `func_extrae_codigo_provincia(vprovincias)`
- `func_crea_colores_brewer(cuantos,que_paletacolor=3)`
- `func_extrae_codigo_ccaa(vCCAA,ConvierteCodSIANE=TRUE)`
- `func_obtiene_codigo_prov(vProv)`

La sintaxis de las funciones principales se recoge a continuación:

```
func_mapademografia_prov(dtmapa,Titulo,categorias = 7,colores = NULL,
                         LeyendaPos = "bottomleft",Leyenda.size=0.5)
```

```
func_mapademografia_ccaa(dtmapa,Titulo,categorias = 7,colores = NULL,
                          LeyendaPos = "bottomleft",Leyenda.size=0.5)
```

```
func_mapademografia_municipios(dtmapa,Titulo,categorias = 7,colores = NULL,
                           LeyendaPos = "bottomleft",Leyenda.size=0.5)
```

La estrategia seguida en todas las funciones es pasarle un data.frame con la información importante en dos columnas:

- "Codigo": que tiene el código de la CCAA, provincia o municipio.
- "Valor": que tiene el valor a representar (mediante una gama de colores).

Para usar estas funciones en primer lugar tendremos que cargar el fichero "funciones_mapas.R":

```
source("funciones_mapas.R")
```

9.1 Ejemplo 1 (mapa de Andalucía sobre las provincias)

En este primer ejemplo representaremos el "saldo migratorio interno" para las provincias de Andalucía de 2010 a 2017.

En primer lugar, tendremos que realizar las operaciones adecuadas para obtener la información de los códigos de las provincias andaluzas (es importante introducir los códigos correctos) y el valor del saldo migratorio interno correspondiente.

```
load("mapaejemplo01.RData")
df_a_mapa = data.frame(
  Codigo = func_extrae_codigo_provincia(provs_filas),
  Valor = SMInternoAnd
)
df_a_mapa %>%
  head()
```

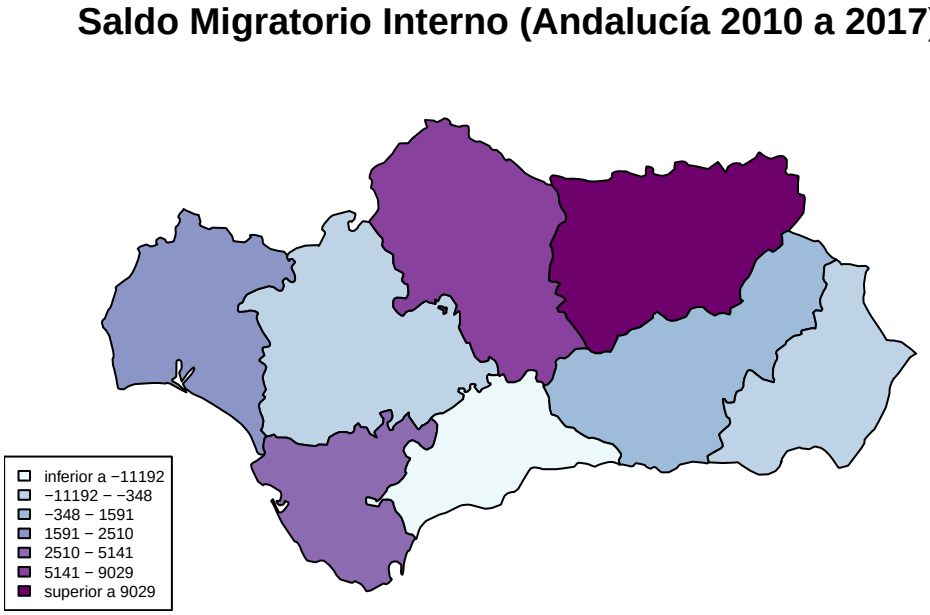
```
##            Codigo Valor
## 04 Almería     04 -1977
```

```
## 11 Cádiz       11  3120
## 14 Córdoba     14  7162
## 18 Granada     18  1282
## 21 Huelva      21  1900
## 23 Jaén        23 10896
```

El cartograma se obtendría con el siguiente código:

```
func_mapademografia_prov(df_a_mapa,
      Titulo = "Saldo Migratorio Interno (Andalucía 2010 a 2017)")
```

```
## Warning in wkt(obj): CRS object has no comment
```



Por defecto, el número de categorías que se utilizan son 7. Por ejemplo, para cambiar a 3 categorías y que utilice unos colores concretos tendríamos que añadir los siguientes argumentos en la función: `categorias = 3,colores=c("blue","red","green")`.

9.2 Ejemplo 2 (mapa de España sobre las provincias)

En el siguiente ejemplo vamos a representar un cartograma con el índice de envejecimiento en las provincias de España en el 2017.

Se ha recurrido a la web del INE, para obtener la siguiente información en formato "px" el índice de envejecimiento para todas las provincias de España en 2017.

Descargamos los datos en formato "px" desde la siguiente url: http://www.ine.es/jaxiT3/Tabla.htm?t=1489 (INEBASE, Fenómenos demográficos, Indicadores demográficos básicos, Crecimiento y Estructura de Población, Indicadores de Estructura de la Población).

```
dfej02b <- as.data.frame(read.px("datos/1489.px"),stringsAsFactors=FALSE)
head(dfej02b)
```

```
##   Periodo     Provincias    value
## 1    2018 Total Nacional 120.4598
## 2    2017 Total Nacional 118.2625
## 3    2016 Total Nacional 116.2798
## 4    2015 Total Nacional 114.7221
## 5    2014 Total Nacional 112.2386
```

```
## 6    2013 Total Nacional 109.5282
```

Realizamos las operaciones necesarias para obtener un data.frame con las columnas "Codigo" y "Valor", adecuadas.

```
tp4 = dfej02b %>%
  dplyr::filter(Periodo=="2017",
                Provincias!="Total Nacional")

df_a_mapa2 = data.frame(
  Codigo = func_extrae_codigo_provincia(tp4$Provincias),
  Valor = tp4$value
)
head(df_a_mapa2)
```

```
##   Codigo     Valor
## 1     02 119.27793
## 2     03 124.76932
## 3     04  80.04999
## 4     01 127.48705
## 5     33 209.95329
## 6     05 191.15470
```

El cartograma sobre el mapa de España a nivel provincial sería el siguiente:

```
func_mapademografia_prov(df_a_mapa2,
      Titulo = "Índice de Envejecimiento provincial (España en 2017)")
```

```
## Warning in wkt(obj): CRS object has no comment
```

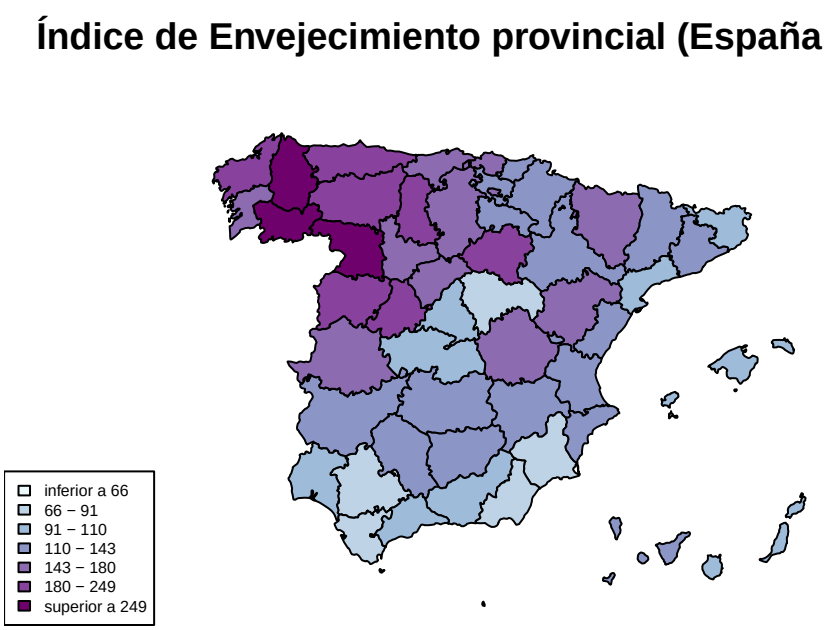


Figura 3: Cartograma con el índice de envejecimiento provincial para España en el 2017. Fuente: elaboración propia

9.3 Ejemplo 3 (mapa de España sobre las comunidades autónomas)

En este último ejemplo, se va a representar de nuevo el índice de envejecimiento en España durante 2017, pero a nivel de comunidades autónomas (CCAA). En este caso, habrá que tener cuidado con el código, ya que debe corresponder al de las CCAA.

Descargamos los datos en formato "px" de la siguiente url: http://www.ine.es/jaxiT3/Tabla.htm?t=1452&L=0

```
dfej02c <- as.data.frame(read.px("datos/1452.px"),stringsAsFactors=FALSE)
head(dfej02c)
```

```
##   Periodo Comunidades.y.Ciudades.Autónomas    value
## 1    2018                   Total Nacional 120.4598
## 2    2017                   Total Nacional 118.2625
## 3    2016                   Total Nacional 116.2798
## 4    2015                   Total Nacional 114.7221
## 5    2014                   Total Nacional 112.2386
## 6    2013                   Total Nacional 109.5282
```

Realizamos las operaciones necesarias para obtener un data.frame con las columnas "Codigo" y "Valor", adecuadas.

```
tp5 = dfej02c %>%
  dplyr::filter(Periodo=="2017",
                Comunidades.y.Ciudades.Autónomas!="Total Nacional")

df_a_mapa3 = data.frame(
  Codigo = func_extrae_codigo_ccaa(tp5$Comunidades.y.Ciudades.Autónomas),
  Valor = tp5$value
)
head(df_a_mapa3)
```

```
##   Codigo     Valor
## 1     61  96.21478
## 2     62 140.25249
## 3     63 209.95329
## 4     64  95.99650
## 5     65 105.73139
## 6     66 146.33787
```

El cartograma sobre el mapa de España a nivel comunidad autónoma sería el siguiente:

```
func_mapademografia_ccaa(df_a_mapa3,
    Titulo = "Índice de Envejecimiento  (Comunidades Autónomas,
    España  2017)")
```

```
## Warning in wkt(obj): CRS object has no comment
```

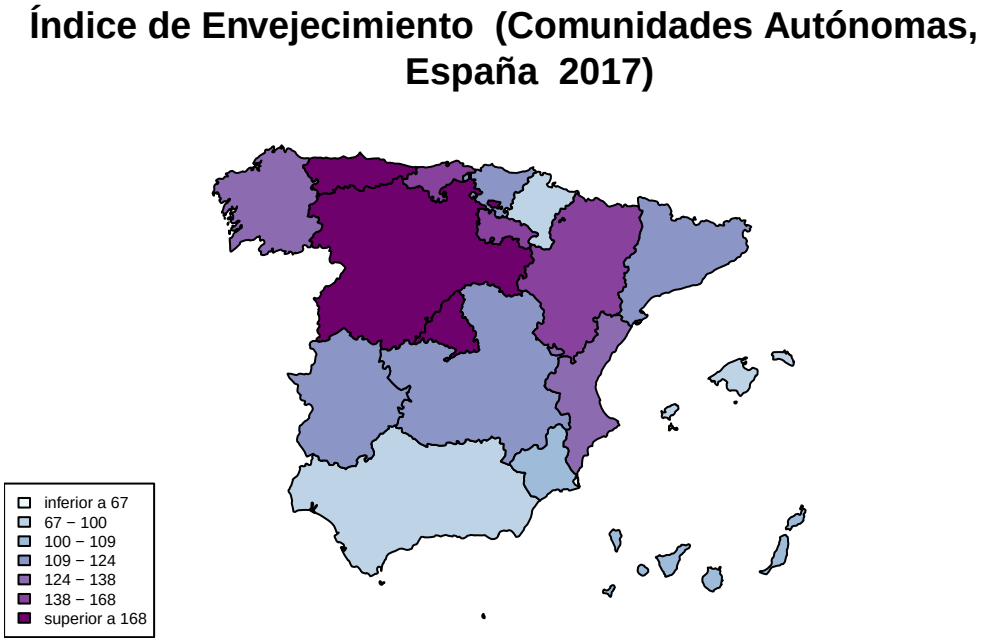


Figura 4: Cartograma con el índice de envejecimiento autonómico para España en el 2017. Fuente: elaboración propia

10 Condiciones en las que se ha creado este documento

```
sessionInfo()
```

```
## R version 4.1.1 (2021-08-10)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS Big Sur 10.16
##
## Matrix products: default
## LAPACK: /Library/Frameworks/R.framework/Versions/4.1/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] classInt_0.4-3     RColorBrewer_1.1-2 maptools_1.1-2     raster_3.5-15
##  [5] sp_1.4-6           Siane_0.1          magrittr_2.0.2     LexisPlotR_0.4.0
##  [9] ggthemes_4.2.4     openxlsx_4.2.5     kableExtra_1.3.4   pxR_0.42.4
## [13] plyr_1.8.6         RJSONIO_1.3-1.6    reshape2_1.4.4     readxl_1.3.1
## [17] knitr_1.37         forcats_0.5.1      stringr_1.4.0      dplyr_1.0.7
## [21] purrr_0.3.4        readr_2.1.1        tidyr_1.1.4        tibble_3.1.6
## [25] ggplot2_3.3.5      tidyverse_1.3.1
##
## loaded via a namespace (and not attached):
##  [1] fs_1.5.2           lubridate_1.8.0    webshot_0.5.2      httr_1.4.2
##  [5] tools_4.1.1        backports_1.4.1    utf8_1.2.2         rgdal_1.5-28
##  [9] R6_2.5.1           KernSmooth_2.23-20 DBI_1.1.2          colorspace_2.0-2
## [13] withr_2.4.3        tidyselect_1.1.1   rematch_1.0.1      compiler_4.1.1
## [17] cli_3.1.1          rvest_1.0.2        xml2_1.3.3         labeling_0.4.2
## [21] scales_1.1.1       proxy_0.4-26       systemfonts_1.0.3  digest_0.6.29
## [25] foreign_0.8-82     rmarkdown_2.11     svglite_2.0.0      pkgconfig_2.0.3
## [29] htmltools_0.5.2    dbplyr_2.1.1       fastmap_1.1.0      highr_0.9
## [33] rlang_1.0.0        rstudioapi_0.13    farver_2.1.0       generics_0.1.2
## [37] jsonlite_1.7.3     zip_2.2.0          Rcpp_1.0.8         munsell_0.5.0
## [41] fansi_1.0.2        lifecycle_1.0.1    terra_1.5-12       stringi_1.7.6
## [45] yaml_2.2.2         grid_4.1.1         crayon_1.4.2       lattice_0.20-45
## [49] haven_2.4.3        hms_1.1.1          pillar_1.7.0       codetools_0.2-18
## [53] reprex_2.0.1       glue_1.6.1         evaluate_0.14      modelr_0.1.8
## [57] vctrs_0.3.8        tzdb_0.2.0         cellranger_1.1.0   gtable_0.3.0
## [61] assertthat_0.2.1   xfun_0.29          broom_0.7.12       e1071_1.7-9
## [65] class_7.3-20       viridisLite_0.4.0  ellipsis_0.3.2
```