Doctorado en Demografía
Taller de Introducción a R en entorno RStudio
Notas de clase

Eduardo Bologna
Universidad Nacional de Córdoba
eduardo.leon.bologna@unc.edu.ar

23 marzo, 2020

Contents

Presentación

Este curso es una introducción a R dirigido a investigadores en Ciencias Sociales y busca facilitar la aproximación a un software cuyo uso en investigación tiene ventajas que superan en mucho las dificultades que suelen aparecer en los primeros intentos.

Para realizar análisis de datos existen numerosos programas informáticos, que se ocupan de los procesos computacionales, de manera que el usuario solo deba decidir qué procedimiento aplicar, cuidar que se cumplan las condiciones que hacen válido al procedimiento (los supuestos) y realizar una lectura correcta y completa del resultado que se obtiene, sin involucrarse con las operaciones de cálculo. Estos programas o "paquetes estadísticos" reúnen en un entorno único las operaciones más frecuentemente usadas por investigadores y analistas de datos y las ponen al alcance del usuario no especializado. Algunos de uso muy común son SPSS, SAS, INFOSTAT, STATA, STATISTICAL, etc. Aun así, hay razones para optar por R, manejar sus rudimentos es una inversión de tiempo que se vuelve una puerta de entrada a múltiples posibilidades. Esto se fundamenta en que R es varias cosas al mismo tiempo:

Es un software para análisis de datos: lo usan profesionales de la estadística, analistas de datos e investigadores de diversas disciplinas para extraer significado de información cuantitativa, para hacer descripciones e inferencias, visualización de datos y modelización predictiva.

Es un lenguaje de programación orientado a objetos, diseñado por estadísticos y para el uso en investigación cuantitativa: el análisis se hace escribiendo sentencias en este lenguaje, que provee objetos, operadores y funciones que hacen muy intuitivo el proceso de explorar, modelar y visualizar datos.

Es un ambiente para el análisis estadístico: en R hay funciones para prácticamente todo tipo de transformación de datos, de modelización y de representaciones gráficas que pueda hacer falta.

Es un proyecto de código abierto: esto significa no solo que se lo puede descargar y usar gratis, sino que el código es abierto y cualquiera puede inspeccionar o modificar las rutinas. Como sucede con otros proyectos de código abierto, como Linux, R ha mejorado sus códigos tras varios años de "muchos ojos mirando" y aportando soluciones. También como otros proyectos de código abierto, R tiene interfaces abiertas, por lo que se integra fácilmente a otras aplicaciones y sistemas.

Es una comunidad: R fue inicialmente desarrollado por Robert Gentleman y Ross Ihaka (Ihaka and Gentleman 1996), del Departamento de Estadística de la Universidad de Auckland, en 1993 y desde entonces el grupo que dirige el proyecto ha crecido y se ha difundido por el mundo. Además, miles de otras personas han contribuido con funcionalidades adicionales por medio del aporte de "paquetes" que utilizan los 2 millones de usuarios de todo el mundo. Como resultado, existe en la web una intensa comunidad de usuarios de R, con muchos sitios que ofrecen recursos para principiantes y para expertos. A esa comunidad se puede recurrir para consultas y para salvar dificultades, son muy activas y dispuestas a ayudar.

R integra programas llamados paquetes, que sirven para realizar análisis específicos. Los paquetes son rutinas que realizan conjuntos de operaciones especializadas, y una de las potencialidades de R es que diferentes investigadores pueden desarrollar paquetes para determinados tipos de análisis y ponerlos a disposición de los demás usuarios. En la actualidad hay más de 10000 paquetes y el conjunto crece porque la comunidad R es muy activa y continuamente se hacen aportes.

No solo cuenta con los métodos estándar sino que, debido a que los principales avances en procedimientos estadísticos se realizan en R, las técnicas más actualizadas están usualmente primero disponibles en R, a los desarrolladores de paquetes comerciales les lleva más tiempo poner las actualizaciones al alcance de los usuarios. Y los usuarios a menudo deben pagar por las actualizaciones.

Permite la reproducción de los análisis por parte de cualquiera que conozca el código que se aplicó, por lo que aporta una herramienta necesaria en los proyectos de ciencia abierta, en especial para la reproducibilidad de los resultados.

Por estas razones, R es uno de los lenguajes de programación que más uso tiene y se está convirtiendo en la *lingua franca* del análisis de datos.

Como lenguaje, R tiene varias interfaces gráficas, que es el modo en que las personas puede interactuar con él. R es el motor y del mismo modo en que, para manejar un auto no hace falta saber cómo funciona el motor, también aquí será suficiente contar con un buen conjunto de comandos (volante, pedales...) para hacer uso de la potencia de ese motor, la interface provee esos comandos. De las interfaces existentes, hemos elegido RStudio (RStudio Team 2018) que es un entorno de desarrollo integrado (IDE) de R para facilitar la edición de código que ofrece diversas herramientas, para hacer muy accesible el uso de R por parte de quienes no se dedican a la programación, sino que son usuarios de procedimientos estadísticos. Además, es posible crear una cuenta en RStudio cloud (https://rstudio.cloud/), desarrollar allí un proyecto o bien subir uno que se tenga en curso, y trabajar con todos los archivos disponibles en la nube, sin necesidad de instalar localmente el software.
Una línea de tiempo sobre el desarrollo de R, se puede encontrar en https://blog.revolutionanalytics.com/2017/10/updated-history-of-r.html

Este curso-taller busca proveer una primera aproximación a este entorno para quienes tengan o no experiencia en otros programas de análisis de datos y cuenten con una base conceptual de estadística.
RStudio es una interfaz con bastantes ayudas para el usuario, por ejemplo, tiene autocompletado para comandos y para objetos existentes. Además, durante la sesión de trabajo, se puede pedir ayuda sobre los comandos con un signo de interrogación delante de él, de ese modo se abre un archivo que da detalles sobre el uso del comando pedido. Fuera de eso, en la web hay mucha ayuda disponible.
La dinámica sugerida de trabajo consiste en reproducir los comandos que se indican en este texto e introducir variaciones sobre cada uno, para ver los resultados. Las ayudas, tanto las que están incorporadas en el software como las que se encuentran en la web, sirven como inspiración para explorar variaciones en la sintaxis.

Los datos

Para ilustrar los procedimientos, en este taller se usan datos provenientes de la Encuesta Permanente de Hogares. La EPH es un programa nacional de producción permanente de indicadores sociales cuyo objetivo es conocer las características socioeconómicas de la población. Es realizada en forma conjunta por el Instituto Nacional de Estadística y Censos (INDEC (2011)) y las Direcciones Provinciales de EstadÌstica (DPE)

(INDEC (2003)). Los datos se recogen por medio de dos cuestionarios; uno de ellos que pregunta por características del hogar y la vivienda y el otro por las personas individualmente.

El software

Instalación local de R y RStudio

En http://cran.r-project.org "Download R for [Linux, Mac o Windows]", y luego "install R for the first time". Una vez descargado, se instala siguiendo las instrucciones de las pantallas, aceptando las opciones por defecto que se ofrecen.

Una vez que R está instalado, se debe sumar RStudio. El lugar de descarga es http://www.rstudio.com/products/rstudio/download/. Allí se elige la version gratis (free version) de RStudio Desktop y se baja hasta encontrar el sistema operativo y la version que corresponda a nuestro equipo. Luego se ejecuta el instalador de RStudio y se eligen las opciones por defecto. Cuando esté ya instalado, se accede por medio de RStudio; si al instalar R se creó un acceso directo a R en el escritorio, se lo puede eliminar. Al abrir RStudio, R es detectado automáticamente y desde allí operaremos.

RStudio en la nube

A los fines de este taller, cada cursante creará una cuenta en RStudio cloud (en https://rstudio.cloud/) y una vez abierta, se crea un nuevo proyecto desde un repositorio (repo) existente en GitHub. La dirección es https://github.com/ebologna/R-en-demo.git y allí se encuentra un directorio llamado "materiales taller R" que contiene los archivos que se usarán en adelante. Alternativamente, se puede hacer la instalación local de R y RStudio como se indica arriba y descargar los archivos de ese repositorio.

Los componentes de RStudio

Cuando abrimos RStudio localmente, se observan tres paneles, uno a la izquierda, más grande, y dos a la derecha. En "file" se solicita un nuevo script, que se abre a la izquierda y ahora quedan cuatro paneles:

Cuando se abre en la nube y se descarga un proyecto existente, aparecen directamente estos cuatro paneles.

- Superior izquierdo es el script que se acaba de abrir, un documento editable en el que se escriben los comandos.
- Inferior izquierdo es la consola, donde se encuentra la ejecución de los comandos y, si corresponde, los resultados de operaciones solicitadas.
- Superior derecho es el entorno de trabajo, allí aparece cada uno de los objetos que se crean durante la sesión.
- Inferior derecho, cuatro pestañas con los directorios de trabajo, los paquetes instalados, la ayuda (cuando se pide), los gráficos que se hagan.

Instalación de paquetes

Cuando se descarga R y RStudio se cuenta con el sistema básico del lenguaje R. Las operaciones mencionadas en el apartado anterior y otras, están disponibles en esa base. Sin embargo, una gran cantidad de procedimientos están programados y ofrecidos como "paquetes", que sirven para tareas específicas. Su creación y desarrollo es parte de la potencialidad de R, porque son aportes de la comunidad que los diseña

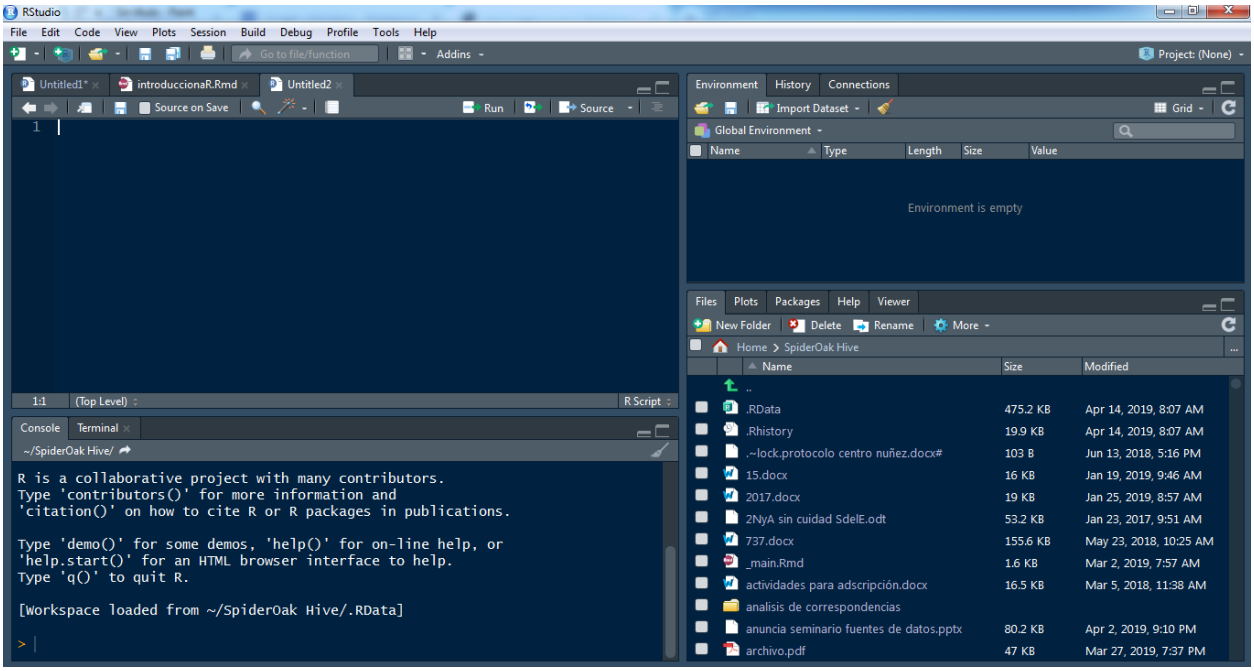


Figure 1: Los cuatro paneles de RStudio

y los ofrece continuamente. En la actualidad hay más de 10000 paquetes en la CRAN (Comprehensive R Archive Network) aplicables a una gran diversidad de procedimientos.

La instalación de paquetes de R puede hacerse desde la línea de comando con la instrucción install.packages("") y el nombre del paquete entre comillas, también puede hacerse más directo, porque la IDE RStudio tiene, en el panel inferior derecho, una pestaña (la tercera) que dice *packages* y en ella, una opción *install* que abre una ventana para escribir (con autocompletado para los existentes) el nombre del paquete que se quiere instalar. Ahora vamos a instalar todos los paquetes que necesitaremos en este recorrido:

```
install.packages("questionr")
install.packages("ggplot2")
install.packages("ggthemes")
install.packages("mblm")
install.packages("RColorBrewer")
install.packages("viridis")
install.packages("demography")
```

Estas instalaciones se realizan por única vez en cada computadora o en cada proyecto creado en RStudio cloud y luego, en cada sesión que vayan a usarse, se deberán cargar con la instrucción `library()`. Trabajaremos primero con los comandos que trae el paquete `base`, que viene por defecto con la instalación de R.

Escribir y ejecutar comandos

Si no se está trabajando en la nubes, antes de empezar a operar es necesario crear un lugar donde se alojarán los archivos que se van a usar. Ese lugar, en R se llama "proyecto". Así, la primera acción será en File → New Project → New Directory → New Project, darle un nombre y definir su ubicación en la computadora en que se trabaje. Si esa ubicación es una carpeta sincronizada (de drive o dropbox u otra) todos los archivos

necesarios para trabajar en el proyecto estarán disponibles. Si se trabaja localmente (con RStudio instalado en la PC), es en esa carpeta donde deben copiarse los archivos bajados de GitHub.

El script es un editor de textos en que se escriben comandos y se ejecutan, ya sea con el botón "run" o con una combinación de teclas que, según la configuración puede ser Ctrl+R o Ctrl+Enter. Una vez escrita la instrucción, se solicita su ejecución y se obtiene el resultado.

```
1+2
```

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

Los elementos que maneja R son objetos: un número, un vector, una base de datos, una tabla y muchos otros. Inicialmente, los que interesan a fin del análisis de datos son: vectores y matrices de datos.

Clases de objeto

Constantes

Un objeto numérico puede tener un valor fijo, si definimos a **x** como el número 3

```
x <- 3
```

En el panel superior derecho aparece este objeto. El signo $<-$ que define el objeto es equivalente a $=$ que da la idea de asignar a **x** el valor **3**. No hay un resultado visible de esta operación, ya que solo consistió en definir a **x** como igual a **3**

Si se lo invoca (se lo llama es decir, se escribe su nombre), muestra su valor.

```
x
```

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

Se puede hacer de una sola vez la asignación y la visualización del resultado, encerrando la expresión entre paréntesis:

```
(x<-3)
```

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

De este modo se asigna a **x** el valor **3** y se lo muestra.

Las salidas de R, es decir los resultados que muestra, están antecedidos por un signo numeral (#) y cada elemento de los resultados lleva su numeración entre corchetes. Aquí el resultado es solo un número, por eso hay un [1] a la izquierda.
Este objeto es un número, lo que puede saberse si se pregunta de qué clase es este objeto:

```
class(x)
```

```
## [1] "numeric"
```

Es numérico.

Si hubiésemos definido el objeto:

```r
t <- "a"
class(t)
```

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

Es carácter, para que lo acepte como valor nuevo, se debe poner entre comillas; de lo contrario, si se escribe:

$t <- a$

Buscará ese objeto, que no ha sido definido antes y dará error. Esta forma de distinguir valores numéricos de textos se puede usar cuando los números codifican categorías, como cuando se usa 1 para varones y 2 para mujeres:

```r
u <- "1"
class(u)
```

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

Allí se entiende al número como un código.

Otros tipos de objeto son lógicos

```r
v <- TRUE
class(v)
```

```
## [1] "logical"
```

El objeto **v** es lógico, esta clase de objeto puede tomar dos valores TRUE y FALSE.

Es posible transformar una clase de objeto en otra. Por ejemplo, si un valor numérico fue cargado como carácter, como el caso de **u** en el ejemplo anterior, se lo vuelve numérico pidiendo:

```r
u <- as.numeric(u)
class(u)
```

```
## [1] "numeric"
```

Pero si intentáramos eso con **t**, el resultado falla, porque no se interpreta un valor numérico.

Los valores textuales corresponden a dos clases diferentes de objetos R: caracteres y factores, que difieren en el modo en que R los almacena internamente. Los factores se guardan como números y una tabla que hace corresponder a cada uno un texto, mientras que los caracteres se guardan como un valor para cada expresión textual, por lo que consumen más memoria. Por defecto, los valores textuales son tomados como de clase `caracter`. Sea x el valor "a":

```r
x<-"a"
class(x)
```

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

Transformado a numérico:

```
x<-as.numeric(x)
```

Produce NA (not available)

```
x
```

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

Si se lo define como factor:

```
x<-as.factor("a")
levels(x)
```

```
## [1] "a"
```

Llevado a numérico:

```
x<-as.numeric(x)
x
```

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

Le asigna el número 1, a la única categoría que tiene **x** la codifica internamente como **1**.

Cuando el objeto es un número, se puede operar simplemente con él

```
x<-8
5 * x
```

```
## [1] 40
```

Aquí, el resultado de la operación no es un objeto nuevo, solo se mostró el resultado. Para crearlo, hace falta ponerle nombre:

```
y <- 5 * x
```

Y no veremos su valor hasta que no lo solicitemos

```
y
```

```
## [1] 40
```

Suma resta, multiplicación y división se hacen con los signos que conocemos:

```
x + y
```

```
## [1] 48
```

```
x - y
```

```
## [1] -32
```

```
x * y
```

```
## [1] 320
```

```
y / x
```

```
## [1] 5
```

```
6 * x + 4 * y
```

```
## [1] 208
```

Para elevar a una potencia se usa ^, por ejemplo, para hacer dos a la tercera potencia, es:

```
2^3
```

```
## [1] 8
```

O **x** (que está guardado con el valor 3) a la quinta potencia:

```
x^5
```

```
## [1] 32768
```

Las raíces son potencias fraccionarias, por lo que puede conseguirse la raíz cuadrada de **x** así:

```
x^(1/2)
```

```
## [1] 2.828427
```

Pero como se usa a menudo, hay una función de biblioteca para eso:

```
sqrt(x)
```

```
## [1] 2.828427
```

Para raíces que no sean cuadradas, se debe usar la potencia fraccionaria. Para la raiz quinta de 24 es:

```
24^(1/5)
```

```
## [1] 1.888175
```

Un comando útil es el de redondeo. Si no queremos expresar la raiz de siete con seis decimales, sino solo con dos, se redondea a dos decimales:

```r
x <- sqrt(7)
x
```

```
## [1] 2.645751
```

```r
round(x, 2)
```

```
## [1] 2.65
```

O todo de una sola vez:

```r
round(sqrt(7), 2)
```

```
## [1] 2.65
```

Vectores

Cuando se trabaja con variables, el conjunto de valores que asume es un objeto que se llama vector. Se lo genera con una letra **c** y paréntesis que indica concatenar valores. Por definición, un vector contiene elementos de la misma clase:

```r
a <- c(1, 5, 8)
b <- c("x", "y", "z")
class(a)
```

```
## [1] "numeric"
```

```r
class(b)
```

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

Si se intenta combinar diferentes clases de objeto, el vector tomará la clase con menos propiedades:

```r
l <- c(1, 3, "a")
class(l)
```

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

Cuando pedimos que muestre los elementos de l:

```r
l
```

```
## [1] "1" "3" "a"
```

Aparecen los números 1 y 3 entre comillas, lo que indica que los está tomando como caracteres, por lo que no podrá operar con ellos. Si lo intentamos por ejemplo, multiplicandolo por cinco, se obtiene un error:

Error in 5 * c : non-numeric argument to binary operator

Esto sucede porque los números fueron tratados como caracteres.

Ejemplo: Los valores de PBI de cinco países son 10000, 3000, 7000, 4000 y 15000, se los puede concatenar así, definiendo el vector que los contiene con el nombre pib5:

```
pib5 <- c(10000, 3000, 7000, 4000, 15000)
```

Y se pueden hacer operaciones con él, por ejemplo, sumar sus valores

```
sum(pib5)
```

```
## [1] 39000
```

O sumarlos y dividir por 5, que va a dar el promedio:

```
sum(pib5) / 5
```

```
## [1] 7800
```

Pero para esto hay una función de biblioteca que calcula la media (promedio) directamente. Veamos su ayuda:

```
?mean
```

Además de la variable que se va a promediar (x), tiene un argumento $trim$, que por defecto viene fijado en cero, sirve para el cómputo de la media recortada, en la misma ayuda está su explicación. Dice además que, por defecto, $na.rm = FALSE$, luego veremos qué significa eso.

```
mean(pib5)
```

```
## [1] 7800
```

O definir un nuevo vector que consista en cada uno de ellos incrementado en un 10%:

```
pib5_10 <- 1.1 * pib5
pib5_10
```

```
## [1] 11000  3300  7700  4400 16500
```

Un vector puede crearse de este modo, concatenando varios valores, o bien como secuencia de números, por ejemplo creamos el vector *diez.pri* como la secuencia de los números que van del 1 al 10, y pedimos que se muestre (encerrando entre paréntesis la expresión):

```
(diez.pri <- 1:10)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

Como puede verse, una vez que los objetos han sido creados, RStudio ofrece el autocompletado, eso vale también para comandos, por lo que no hace falta recordar con precisión el nombre de cada uno, se empieza a escribirlo y RStudio lo sugiere.

Si se quiere que la secuencia tenga saltos de magnitud diferente a 1, el comando es `seq`, cuyos argumentos son: los números inicial y final de la secuencia y la amplitud del salto de cada valor al siguiente. Para ir del 1 al 10 de a 0.50:

```r
seq(1, 10, .5)
```

```
##  [1]  1.0  1.5  2.0  2.5  3.0  3.5  4.0  4.5  5.0  5.5  6.0  6.5  7.0  7.5  8.0
## [16]  8.5  9.0  9.5 10.0
```

En este ejemplo, no creamos ningún objeto, solo solicitamos la secuencia para verla. El [16] que está debajo del [1] indica que el valor 8.5 ocupa el lugar 16 de la secuencia.

El comando `rep`, repite un valor las veces que se solicite, repetir el valor 4, siete veces es:

```r
rep(4, 7)
```

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

O el valor "perro", tres veces:

```r
rep("perro", 3)
```

```
## [1] "perro" "perro" "perro"
```

Estas maneras de generar secuencias pueden combinarse (o concatenarse):

```r
c(1:5, seq(1, 7, .8), rep(65, 4))
```

```
##  [1]  1.0  2.0  3.0  4.0  5.0  1.0  1.8  2.6  3.4  4.2  5.0  5.8  6.6 65.0 65.0
## [16] 65.0 65.0
```

Notemos que hay un decimal (cero) en los enteros, eso es porque los números que componen el vector fueron interpretados como valores reales y no como enteros, como sucedió en el ejemplo anterior. Con que haya un solo número decimal, todos los componentes del vector son tratados como tales, a los enteros les corresponderá parte decimal igual a cero. Un vector siempre contiene elementos de la misma clase.

Una clase de vector frecuente cuando se trata con variables cualitativas es el "factor", que está constituido por números que corresponden a etiquetas de valor. Por ejemplo, se define un vector como los códigos 1 y 2 para personas pertenecientes a los grupos experimental y control respectivamente y pedimos que se muestre:

```r
grupo <- c(1, 2)
class(grupo)
```

```
## [1] "numeric"
```

```r
grupo
```

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

Es un vector numérico con valores 1 y 2. Luego indicamos que lo trate como un factor y volvemos a pedir su visualización:

```r
grupo <- as.factor(grupo)
class(grupo)
```

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

```r
grupo
```

```
## [1] 1 2
## Levels: 1 2
```

Se trata de un factor y, si bien sus valores siguen siendo 1 y 2, ahora son llamados "niveles del factor". Los niveles pueden preguntarse explícitamente:

```r
levels(grupo)
```

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

Y también definirse, como etiquetas:

```r
levels(grupo) <- c("experimental", "control")
```

Ahora éstos son los nuevos niveles:

```r
levels(grupo)
```

```
## [1] "experimental" "control"
```

Observemos la diferencia que esto tiene con haber evitado la codificación numérica y definir:

```r
grupo_2 <- c("experimental", "control")
class(grupo_2)
```

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

```r
grupo_2
```

```
## [1] "experimental" "control"
```

```r
grupo_2 <- as.factor(grupo_2)
levels(grupo_2)
```

```
## [1] "control"      "experimental"
```

```r
grupo_2
```

```
## [1] experimental control
## Levels: control experimental
```

Como el vector fue creado como de caracteres, sus valores se ordenan alfabéticamente. Cuando se muestra el vector, los niveles aparecen en el orden que elegimos, pero cuando se lo vuelve factor, se los ordena alfabéticamente. Eso es un problema que resolvemos evitando los vectores de caracteres. Cuando deben usarse, se realiza una codificación numérica y luego se etiquetan los niveles.

Si 10 personas han sido asignadas al grupo experimental y otras 10 al grupo control, el vector que representa su pertenencia puede ser:

```
pertenencia <- c(rep(1, 10), rep(2, 10))
pertenencia <- as.factor(pertenencia)
levels(pertenencia) <- c("experimental", "control")
```

Así como `class` indica de qué clase es un objeto, existen comandos para preguntar por características específicas de los objetos y obtener respuestas por sí o por no. Por ejemplo, si se pregunta si el valor de x (recién definido) es un factor:

```
is.factor(x)
```

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

O si uno dividido cero es infinito:

```
is.infinite(1 / 0)
```

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

Estos comandos dan resultados de clase lógica, con `FALSE` y `TRUE` como posibilidades.

Los corchetes, [], permiten seleccionar elementos de un vector. La lectura es que, del vector, se retienen los valores que cumplen con la condición que está dentro del corchete. Se define z, como la secuencia de 1 a 6 y luego a h como los elementos de z que sean menores a 5:

```
z <- c(1, 2, 3, 4, 5, 6)
h <- z[z < 5]
z
```

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

```
h
```

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

La longitud de un vector es el número de elementos que contiene, se solicita con el comando `length`:

```
length(z)
```

```
## [1] 6
```

```
length(h)
```

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

En la variable pertenencia, los niveles son:

```
levels(pertenencia)
```

```
## [1] "experimental" "control"
```

Mientras que los valores:

```
pertenencia
```

```
##  [1] experimental experimental experimental experimental experimental
##  [6] experimental experimental experimental experimental experimental
## [11] control      control      control      control      control
## [16] control      control      control      control      control
## Levels: experimental control
```

La longitud del vector es:

```
length(pertenencia)
```

```
## [1] 20
```

Matriz de datos

Cuando se combinan varios vectores, todos de la misma longitud, se construye un "data frame", una matriz de datos, cuyo formato más frecuente es que tenga los casos en las filas y las variables en las columnas; cada columna es un vector que contiene los valores de cada variable.
Por ejemplo, si tenemos 10 observaciones que corresponden a 7 varones y 3 mujeres, que son estudiantes de la universidad, y el vector que representa el sexo de esas personas, con las categorías codificadas como 1 y 2, es:

```
sexo <- c(rep(1, 7), rep(2, 3))
sexo <- as.factor(sexo)
levels(sexo) <- c("varones", "mujeres")
```

Se ha creado el vector *sexo* por medio de la concatenación de dos repeticiones, del 1 siete veces y del 2, tres veces. Luego se trató a ese vector como un factor y se etiquetaron sus niveles. El siguiente vector contiene las edades de las mismas personas:

```
edad <- c(25, 28, 31, 20, 21, 22, 25, 28, 28, 28)
```

Entonces, se crea una matriz de datos con el comando:

```
sexo_edad_estudiantes <- data.frame(sexo, edad)
```

En el panel superior derecho han aparecido los objetos que acaban de crearse. De este último se indica allí la cantidad de casos (observaciones) y de variables. Cuando se lo cliquea, se obtiene una vista en una ventana separada del script.
La misma vista puede lograrse con el comando:

```
View(sexo_edad_estudiantes)
```

Esto que ha sido creado es un nuevo objeto, de clase:

```
class(sexo_edad_estudiantes)
```

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

Y cuyos atributos son:

```
attributes(sexo_edad_estudiantes)
```

```
## $names
## [1] "sexo" "edad"
##
## $class
## [1] "data.frame"
##
## $row.names
##  [1]  1  2  3  4  5  6  7  8  9 10
```

Los nombres (names) son las denominaciones de las columnas (las variables), la clase es lo que solicitamos antes y row.names son los nombres de las filas, que por defecto coloca numerada consecutivamente. Cada uno de esos atributos está precedido por un signo pesos, ese es el modo de acceder a cada uno de ellos. Por ejemplo, para ver el vector que representa el sexo, pedimos:

```
sexo_edad_estudiantes$sexo
```

```
##  [1] varones varones varones varones varones varones varones mujeres mujeres
## [10] mujeres
## Levels: varones mujeres
```

El signo pesos separa el nombre de la matriz de datos del nombre de la variable: df$x quiere decir, la variable **x** perteneciente a la matriz **df**.
Los números entre corchetes (el [1] y [9] en este ejemplo) indican el número del primer elemento de esa fila.
Podemos preguntar de qué clase es este vector:

```
class(sexo_edad_estudiantes$sexo)
```

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

Por defecto lo leyó como factor, con los dos niveles que se indican más arriba. A ellos se puede llegar directamente:

```
levels(sexo_edad_estudiantes$sexo)
```

```
## [1] "varones" "mujeres"
```

Y se los puede redefinir:

```
levels(sexo_edad_estudiantes$sexo) <- c("masculino", "femenino")
```

Ahora la matriz se ve:

```
View(sexo_edad_estudiantes)
```

Esta matriz de datos puede guardarse para usos posteriores, para no tener que volver a correr la sintaxis la próxima vez que la necesitemos:

```
write.table(sexo_edad_estudiantes,
            "sexo_edad_estudiantes.csv",
            sep = ";", row.names = FALSE)
```

Hemos pedido que escriba la tabla con el mismo nombre (no es obligación), como archivo csv y que no ponga nombre a las filas. Esto último es necesario, porque de lo contrario aparece una nueva columna y los nombres de las variables se desplazan una celda a la izquierda. Como no indicamos otra cosa, la matriz será guardada en el directorio de trabajo.

El primer resumen útil de las variables de una matriz de datos es la tabla univariada. Para cada una de las dos variables, tenemos:

```
table(sexo_edad_estudiantes$sexo)
```

```
##
## masculino  femenino
##         7         3
```

```
table(sexo_edad_estudiantes$edad)
```

```
##
## 20 21 22 25 28 31
##  1  1  1  2  4  1
```

Lectura de una matriz de datos

Es poco frecuente la creación de matrices de datos en R, salvo a fines de ejemplificación. Por el contrario, a menudo es necesario leer un base que está guardada con un determinado formato (xls, ods, sav, sas, txt, csv, etc). El comando genérico es `read.table`, que requiere especificación sobre los simbolos que separan los campos y los decimales, si la primera fila lleva el nombre de las variables. Otros comandos más específicos son `read.csv`, `read.csv2`, el primero usa como separador por defecto ",", el segundo usa ";" y no necesitan que se indique si están los nombres de las variables, porque por defecto los toman. A modo de ejemplo, leemos la base de la Encuesta Permanente de Hogares correspondiente al tercer trimestre de 2018. El archivo tiene formato de texto (.txt) y se llama "usu_individual_T318.txt". Vamos a darle el nuevo nombre de eph.3.18:

```
eph.3.18 <- read.table("usu_individual_T318.txt",
  sep = ";", header = TRUE
)
```

Hemos indicado:

- El nombre del archivo a leer junto con la ruta para llegar a él, muy conveniente que esté en el directorio de trabajo.

- El separador de campos: suelen ser comas, punto y comas, tabulaciones, espacio en blanco o pipe | (alt+124 en windows). Si los campos están separados por comas, se indica $sep = ","$ y así con los demás separadores, siempre entrecomillados.

- Que el archivo tiene encabezado: este es el caso casi siempre, porque la primera fila de la matriz de datos tiene los nombres de las variables. Se indica: $header = TRUE$.

- Si a los casos perdidos estuviesen codificados de algún modo particular, por ejemplo como 9999, debe indicarse $na.strings =$ "9999".
  Conviene mirar la base original, desde el archivo .txt, que puede abrirse con bloc de notas, para asegurarse qué sepadores tiene.

En el panel superior derecho aparece la matriz de datos y su tamaño.

Resumen de datos

Distribuciones de frecuencia

Trabajamos a continuación con la base de la EPH, que ya está leída y se llama eph.3.18

```
class(eph.3.18)
```

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

```
names(eph.3.18)
```

```
##   [1] "CODUSU"     "ANO4"       "TRIMESTRE"  "NRO_HOGAR"  "COMPONENTE"
##   [6] "H15"        "REGION"     "MAS_500"    "AGLOMERADO" "PONDERA"
##  [11] "CH03"       "CH04"       "CH05"       "CH06"       "CH07"
##  [16] "CH08"       "CH09"       "CH10"       "CH11"       "CH12"
##  [21] "CH13"       "CH14"       "CH15"       "CH15_COD"   "CH16"
##  [26] "CH16_COD"   "NIVEL_ED"   "ESTADO"     "CAT_OCUP"   "CAT_INAC"
##  [31] "IMPUTA"     "PP02C1"     "PP02C2"     "PP02C3"     "PP02C4"
##  [36] "PP02C5"     "PP02C6"     "PP02C7"     "PP02C8"     "PP02E"
##  [41] "PP02H"      "PP02I"      "PP03C"      "PP03D"      "PP3E_TOT"
##  [46] "PP3F_TOT"   "PP03G"      "PP03H"      "PP03I"      "PP03J"
##  [51] "INTENSI"    "PP04A"      "PP04B_COD"  "PP04B1"     "PP04B2"
##  [56] "PP04B3_MES" "PP04B3_ANO" "PP04B3_DIA" "PP04C"      "PP04C99"
##  [61] "PP04D_COD"  "PP04G"      "PP05B2_MES" "PP05B2_ANO" "PP05B2_DIA"
##  [66] "PP05C_1"    "PP05C_2"    "PP05C_3"    "PP05E"      "PP05F"
##  [71] "PP05H"      "PP06A"      "PP06C"      "PP06D"      "PP06E"
##  [76] "PP06H"      "PP07A"      "PP07C"      "PP07D"      "PP07E"
##  [81] "PP07F1"     "PP07F2"     "PP07F3"     "PP07F4"     "PP07F5"
##  [86] "PP07G1"     "PP07G2"     "PP07G3"     "PP07G4"     "PP07G_59"
##  [91] "PP07H"      "PP07I"      "PP07J"      "PP07K"      "PP08D1"
##  [96] "PP08D4"     "PP08F1"     "PP08F2"     "PP08J1"     "PP08J2"
## [101] "PP08J3"     "PP09A"      "PP09A_ESP"  "PP09B"      "PP09C"
## [106] "PP09C_ESP"  "PP10A"      "PP10C"      "PP10D"      "PP10E"
```

```
## [111] "PP11A"      "PP11B_COD"  "PP11B1"     "PP11B2_MES" "PP11B2_ANO"
## [116] "PP11B2_DIA" "PP11C"      "PP11C99"    "PP11D_COD"  "PP11G_ANO"
## [121] "PP11G_MES"  "PP11G_DIA"  "PP11L"      "PP11L1"     "PP11M"
## [126] "PP11N"      "PP11O"      "PP11P"      "PP11Q"      "PP11R"
## [131] "PP11S"      "PP11T"      "P21"        "DECOCUR"    "IDECOCUR"
## [136] "RDECOCUR"   "GDECOCUR"   "PDECOCUR"   "ADECOCUR"   "PONDIIO"
## [141] "TOT_P12"    "P47T"       "DECINDR"    "IDECINDR"   "RDECINDR"
## [146] "GDECINDR"   "PDECINDR"   "ADECINDR"   "PONDII"     "V2_M"
## [151] "V3_M"       "V4_M"       "V5_M"       "V8_M"       "V9_M"
## [156] "V10_M"      "V11_M"      "V12_M"      "V18_M"      "V19_AM"
## [161] "V21_M"      "T_VI"       "ITF"        "DECIFR"     "IDECIFR"
## [166] "RDECIFR"    "GDECIFR"    "PDECIFR"    "ADECIFR"    "IPCF"
## [171] "DECCFR"     "IDECCFR"    "RDECCFR"    "GDECCFR"    "PDECCFR"
## [176] "ADECCFR"    "PONDIH"
```

Definimos una nueva variable con las etiquetas de *sexo* a partir de CH04 y redefinimos los niveles de *ESTADO* (ver el manual de códigos de la EPH):

```
eph.3.18$sexo<-as.factor(eph.3.18$CH04)
levels(eph.3.18$sexo)<-c("varon", "mujer")
levels(eph.3.18$sexo) # verificamos
```

```
## [1] "varon" "mujer"
```

```
eph.3.18$ESTADO<-as.factor(eph.3.18$ESTADO)
# se descarta el cero y el cuatro
levels(eph.3.18$ESTADO)<-c(NA, "ocupade", "desocupade", "inactive", NA)
levels(eph.3.18$ESTADO)
```

```
## [1] "ocupade"    "desocupade" "inactive"
```

Ahora empezamos a resumir. La tabla univariada se puede pedir solamente:

```
table(eph.3.18$sexo)
```

```
##
## varon mujer
## 27219 29660
```

O definir un objeto que la contenga:

```
tabla.sexo<-table(eph.3.18$sexo)
class(tabla.sexo)
```

```
## [1] "table"
```

```
# se agregan los totales:
addmargins(table(eph.3.18$sexo))
```

```
##
## varon mujer   Sum
## 27219 29660 56879
```

```
addmargins(tabla.sexo)
```

```
##
## varon mujer   Sum
## 27219 29660 56879
```

```
# Y frecuencias relativas:
prop.table(tabla.sexo)
```

```
##
##     varon     mujer
## 0.4785422 0.5214578
```

```
# Frecuencias relativas multiplicadas por 100 y
# redondeadas a dos decimales
round(100*prop.table(table(eph.3.18$sexo)), 2)
```

```
##
## varon mujer
## 47.85 52.15
```

Los corchetes se usan para referenciar elementos de los objetos:

```
tabla.sexo[1]
```

```
## varon
## 27219
```

```
tabla.sexo[2]
```

```
## mujer
## 29660
```

```
tabla.sexo[3] # no existe
```

```
## <NA>
##   NA
```

```
addmargins(tabla.sexo)[3]# es el total
```

```
##   Sum
## 56879
```

La tablas bivariadas se piden con el mismo comando.

```
table(eph.3.18$ESTADO, eph.3.18$sexo)
```

```
##
##              varon mujer
##   ocupade    13202 10196
##   desocupade   940   930
##   inactive    8787 14380
```

O bien si se define y guarda el objeto:

```
sexo_por_estado<-table(eph.3.18$ESTADO, eph.3.18$sexo)
```

Sus elementos están numerados por columnas:

```
sexo_por_estado
```

```
##
##              varon mujer
##   ocupade    13202 10196
##   desocupade   940   930
##   inactive    8787 14380
```

```
sexo_por_estado[1]
```

```
## [1] 13202
```

```
sexo_por_estado[2]
```

```
## [1] 940
```

```
sexo_por_estado[4]
```

```
## [1] 10196
```

El comando *summary* detecta de qué la clase es la variable y ofrece un resumen. En el caso de factores, el resumen es la tabla univariada.

```
summary(eph.3.18$sexo)
```

```
## varon mujer
## 27219 29660
```

Para objetos más complejos, el comando *summary* ofrece más información.

Para usar los ponderadores, usaremos el paquete diseñado para análisis de datos de encuestas: se llama `questionr`, ya lo tenemos instalado, por lo que solo queda cargarlo en esta sesión:

```
library(questionr)
```

El comando de este paquete para tablas ponderadas es *wtd.table* y requiere que se indique el vector de ponderadores. Si no lo recordamos, se pregunta por el paquete:

```
?questionr
```

Y se abre, en el panel inferior derecho la ayuda (en Index), allí está el listado de comandos que tiene el paquete, con una breve descripción. Si se elige *wtd.table* se muestra su sintaxis y los argumentos que acepta; allí puede verse, por ejemplo que *na.rm* es verdadero por deecto, de modo que, a diferencia de los comandos para pedir media, mediana, etc., si hay casos predidos los va a descartar automáticamente. Aquí solo vamos a usar como argumentos la variable y los ponderadores.

```
wtd.table(eph.3.18$sexo, weights = eph.3.18$PONDERA)
```

```
##    varon    mujer
## 13352303 14489213
```

Sea que se le ponga nombre o no, con esa tabla se pueden hacer las mismas operaciones que con una tabla simple:

```
# univariada
addmargins(wtd.table(eph.3.18$sexo,
                     weights = eph.3.18$PONDERA))
```

```
##    varon    mujer      Sum
## 13352303 14489213 27841516
```

```
#bivariada
addmargins(wtd.table(eph.3.18$ESTADO, eph.3.18$sexo,
                     weights = eph.3.18$PONDERA))
```

```
##               varon    mujer      Sum
## ocupade     6677727  5144297 11822024
## desocupade   566368   601409  1167777
## inactive    4062216  6750886 10813102
## Sum        11306311 12496592 23802903
```

Un análisis frecuente sobre tablas bivariadas es una prueba de independencia (χ^2):

```
chisq.test(sexo_por_estado)
```

```
##
##  Pearson's Chi-squared test
##
## data:  sexo_por_estado
## X-squared = 1603.9, df = 2, p-value < 0.00000000000000022
```

Que solo da el puntaje χ^2, los grados de libertad y el valor p. Pero la prueba es un objeto en sí mismo y lo podemos guardar con nombre:

```
prueba_chi<-chisq.test(sexo_por_estado)

# es de una clase especifica
class(prueba_chi)
```

```
## [1] "htest"
```

Su resumen es más informativo:

```
summary(prueba_chi)
```

```
##           Length Class  Mode
## statistic 1      -none- numeric
## parameter 1      -none- numeric
## p.value   1      -none- numeric
## method    1      -none- character
## data.name 1      -none- character
## observed  6      table  numeric
## expected  6      -none- numeric
## residuals 6      table  numeric
## stdres    6      table  numeric
```

Que dice que esta información está disponible, por ejemplo, las frecuencias esperadas:

```
prueba_chi$expected
```

```
##
##                  varon     mujer
##   ocupade    11076.551 12321.449
##   desocupade   885.253   984.747
##   inactive   10967.196 12199.804
```

O el método usado

```
prueba_chi$method
```

```
## [1] "Pearson's Chi-squared test"
```

El puntaje χ^2

```
prueba_chi$statistic
```

```
## X-squared
##   1603.94
```

El nivel de educación (*NIVEL_ED*) está cargada en la EPH de manera particular (ver el manual de códigos) por lo que hace falta, primero tratarla como factor, y, para respetar el orden de las categorías, llevar el valor siete al primer lugar:

```
# definimos una nueva variable como factor a partir de NIVEL_ED
eph.3.18$educacion<-as.factor(eph.3.18$NIVEL_ED)

# ajustamos el orden de sus niveles: primero el 7,
# luego del 1 al 6:
eph.3.18$educacion<-factor(eph.3.18$educacion,
                           levels(eph.3.18$educacion)[c(7, 1:6)])

# la cruzamos con la original para verificar
table(eph.3.18$educacion, eph.3.18$NIVEL_ED)
```

```
##
##         1     2     3     4     5     6     7
##   7     0     0     0     0     0     0  5343
##   1  8322     0     0     0     0     0     0
##   2     0  7276     0     0     0     0     0
##   3     0     0 11787     0     0     0     0
##   4     0     0     0 10967     0     0     0
##   5     0     0     0     0  6416     0     0
##   6     0     0     0     0     0  6768     0
```

```
# y ponemos nombres a los niveles
levels(eph.3.18$educacion)<-c("sin instruccion", "primaria incompleta",
                              "primaria completa", "secundaria incompleta", "secundaria completa",
                              "universitaria incompleta", "universitaria completa")
table(eph.3.18$educacion)
```

```
##
##          sin instruccion      primaria incompleta        primaria completa
##                     5343                     8322                     7276
##    secundaria incompleta      secundaria completa universitaria incompleta
##                    11787                    10967                     6416
##   universitaria completa
##                     6768
```

Si más tarde la queremos usar como numérica (ahora que los números respetan el orden de las categorías), definimos otra variable:

```
eph.3.18$educacion_numerica<-as.numeric(eph.3.18$educacion)
```

Para seleccionar un subconjunto de casos o de variables de una matriz de datos, es necesario establecer condiciones con operadores y conectores lógicos. Los más frecuentes son:
- > mayor
- < menor
- == igual (es doble, porque "=" es equivalente a "<-", indica asignación)
- - no
- & y
- | o
- is.na() ¿es caso perdido?

Los operadores dan resultado verdadero o falso (TRUE, FALSE), por ejemplo:

```
5>4
```

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

```
2+3==6
```

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

```
2>1 & 3<5
```

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

```
2>1 & 3>5
```

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

```
2>1 | 3>5
```

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

Medidas descriptivas

Vamos a usar los operadores lógicos para corregir la codificación de la edad (CH06), que en la EPH se asigna a las personas menores de un año, el valor -1, cuando sería más adecuado 0. Para reasignar, defimos una nueva variable en la base, la llamamos *edad* y es una copia de CH06:

```
eph.3.18$edad<-eph.3.18$CH06
```

Luego asignamos a *edad* el valor 0 allí donde CH06 valga -1. Los corchetes referencian los casos que deben cambiarse, que son aquellos para los cuales la expresión tome el valor "verdadera":

```
eph.3.18$edad[eph.3.18$CH06==-1]<-0
```

La instrucción dice "allí donde la variable $CH06$ de la base *eph*.3.18 tome el valor -1, asignar a la variable *edad* de la base *eph*.3.18, el valor 0".
Las sentencias lógicas sirven para seleccionar casos o variables, si es quiere definir una nueva matriz de datos que solo contenga asalariados del aglomerado Gran Córdoba, que hayan declarado un ingreso no nulo ni perdido, el comando es:

```
asalariados.con.ingreso.cordoba<-subset(
  eph.3.18, eph.3.18$CAT_OCUP==3  &
    eph.3.18$AGLOMERADO==13 &
    eph.3.18$PP08D1>0 &
    is.na(eph.3.18$PP08D1)==FALSE)
```

Así se define la nueva matriz de datos *asalariados.con.ingreso.cordoba* como el subconjunto de eph.3.18 que cumple las cuatro condiciones simultáneamente (&).

Dado que el ingreso salarial (PP08D1) en numérico, el comando *summary* no genera una tabla sino que muestra medidas descriptivas:

```
summary(asalariados.con.ingreso.cordoba$PP08D1)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     800    8000   14000   14979   20000   70000
```

Individualmente, las medidas descriptivas se piden por su nombre

```
mean(asalariados.con.ingreso.cordoba$PP08D1)
```

```
## [1] 14979.02
```

```r
sd(asalariados.con.ingreso.cordoba$PP08D1)
```

```
## [1] 9453.936
```

```r
median(asalariados.con.ingreso.cordoba$PP08D1)
```

```
## [1] 14000
```

```r
min(asalariados.con.ingreso.cordoba$PP08D1)
```

```
## [1] 800
```

```r
max(asalariados.con.ingreso.cordoba$PP08D1)
```

```
## [1] 70000
```

Si la variable tiene casos perdidos, debe indicarse su exclusión, por ejemplo si x es un vector con dos casos perdidos:

```r
x<-c(1,5,8,4,NA,6,4,9,NA)
```

La media falla

```r
mean(x)
```

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

Para excluir los NA se indica que active la función "quitar NA":

```r
mean(x, na.rm = TRUE)
```

```
## [1] 5.285714
```

Esto vale para todas las operaciones resumen, se deben quitar los NA para que devuelva el resultado.

No hay función de biblioteca para el coeficiente de variación, por lo que debe calcularse:

```r
sd(
  asalariados.con.ingreso.cordoba$PP08D1)/mean(
    asalariados.con.ingreso.cordoba$PP08D1)
```

```
## [1] 0.6311452
```

O construirse una función que lo calcule:

```
cv<-function(x){
  sd(x)/mean(x)
}
```

Y aplicarla:

```
cv(asalariados.con.ingreso.cordoba$PP08D1)
```

```
## [1] 0.6311452
```

Se puede incluir el redondeo y la expresión porcentual en la función

```
cv<-function(x){
  100*round(sd(x)/mean(x),2)
}

cv(asalariados.con.ingreso.cordoba$PP08D1)
```

```
## [1] 63
```

Y para que quede mejor presentado, pegar el signo

```
cv<-function(x){
  paste(100*round(sd(x)/mean(x),2),"%")
}

cv(asalariados.con.ingreso.cordoba$PP08D1)
```

```
## [1] "63 %"
```

Si en lugar de "paste" se usa "paste0" se elimina el espacio entre el número y el signo.

Relaciones entre variables

Variables nominales

El paquete `questionr` trae el cálculo del coeficiente V de Cramer, que tiene como argumento una tabla, por ejemplo, para evaluar la intensidad de la relación entre sexo y condición de actividad:

```
cramer.v(sexo_por_estado)
```

```
## [1] 0.1819761
```

Variables ordinales

Para ver la intensidad de la asociación entre el nivel de educación y los ingresos salariales se usa el coeficiente de correlación de Spearman, que se pide:

```
cor(
  asalariados.con.ingreso.cordoba$educacion_numerica,
    asalariados.con.ingreso.cordoba$PP08D1,
    method = "spearman")
```

```
## [1] 0.380168
```

La prueba de hipótesis sobre la significación de este coeficiente es:

```
cor.test(
  asalariados.con.ingreso.cordoba$educacion_numerica,
    asalariados.con.ingreso.cordoba$PP08D1,
    method = "spearman")
```

```
##
##  Spearman's rank correlation rho
##
## data:  asalariados.con.ingreso.cordoba$educacion_numerica and asalariados.con.ingreso.cordoba$PP08D1
## S = 44283000, p-value < 0.00000000000000022
## alternative hypothesis: true rho is not equal to 0
## sample estimates:
##      rho
## 0.380168
```

Para realizar una prueba unilateral derecha, veamos la ayuda:

```
?cor.test
```

Allí se indica que, además de las dos variables a correlacionar, el argumento *alternative* admite tres valores, entonces, se indica así en el comando:

```
cor.test(
  asalariados.con.ingreso.cordoba$educacion_numerica,
    asalariados.con.ingreso.cordoba$PP08D1,
    method = "spearman", alternative = "greater")
```

```
##
##  Spearman's rank correlation rho
##
## data:  asalariados.con.ingreso.cordoba$educacion_numerica and asalariados.con.ingreso.cordoba$PP08D1
## S = 44283000, p-value < 0.00000000000000022
## alternative hypothesis: true rho is greater than 0
## sample estimates:
##      rho
## 0.380168
```

Variables cuantitativas

El coeficiente de Pearson es el que calcula por defecto el comando "cor", por lo que no hace falta indicar el método. Para la correlación entre la edad y el ingreso salarial:

```r
cor(asalariados.con.ingreso.cordoba$edad,
    asalariados.con.ingreso.cordoba$PP08D1)
```

```
## [1] 0.2749696
```

La prueba se pide igual

```r
cor.test(asalariados.con.ingreso.cordoba$edad,
    asalariados.con.ingreso.cordoba$PP08D1)
```

```
##
##  Pearson's product-moment correlation
##
## data:  asalariados.con.ingreso.cordoba$edad and asalariados.con.ingreso.cordoba$PP08D1
## t = 7.8427, df = 752, p-value = 0.00000000000001512
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.2076479 0.3396989
## sample estimates:
##       cor
## 0.2749696
```

Y provee también un intervalo de confianza para el coeficiente.
Si elegimos ponerle nombre al objeto que resulta del cálculo, podemos acceder separadamente a cada uno de los resultados:

```r
prueba.correlacion<-cor.test(asalariados.con.ingreso.cordoba$edad,
    asalariados.con.ingreso.cordoba$PP08D1)
```

Su resumen es:

```r
summary(prueba.correlacion)
```

```
##             Length Class  Mode
## statistic   1      -none- numeric
## parameter   1      -none- numeric
## p.value     1      -none- numeric
## estimate    1      -none- numeric
## null.value  1      -none- numeric
## alternative 1      -none- character
## method      1      -none- character
## data.name   1      -none- character
## conf.int    2      -none- numeric
```

Si solo queremos el intervalo de confianza, lo pedimos así:

```r
prueba.correlacion$conf.int
```

```
## [1] 0.2076479 0.3396989
## attr(,"conf.level")
## [1] 0.95
```

La ayuda en este comando es:

Muestra que, entre los argumentos, está el nivel de confianza para construir el intervalo, que viene por defecto, fijado en 0.95. Si se cambia a 0.99 y se llama ahora prueba.correlacion.2 al nuevo objeto, se obtiene:

```
prueba.correlacion.2<-cor.test(asalariados.con.ingreso.cordoba$edad,
    asalariados.con.ingreso.cordoba$PP08D1, conf.level = 0.99)
prueba.correlacion.2$conf.int
```

```
## [1] 0.1860464 0.3594250
## attr(,"conf.level")
## [1] 0.99
```

Modelo lineal

Para construir un modelo lineal que ajuste los ingresos como función de las horas trabajadas, hay que verificar la calidad de la variable "horas semanales trabajadas" (PP3E_TOT)

```
summary(asalariados.con.ingreso.cordoba$PP3E_TOT)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.00   24.00   40.00   34.51   45.00   98.00
```

Retendremos solo los que declaran horas trabajadas, para lo que volvemos a reducir la matriz:

```
asalariados.con.ingreso.y.horas.cordoba<-subset(
  asalariados.con.ingreso.cordoba,
  asalariados.con.ingreso.cordoba$PP3E_TOT>0)
```

Se perdieron 30 casos que no declaran horas trabajadas en la semana de referencia. Para construir el modelo, recurrimos a la ayuda:

```
?lm
```

Lo indispensable es indicar las variables en la fórmula y el origen de los datos. La fórmula es $variable\ de\ respuesta \sim regresora_1 + regresora_2 + ...$

Para nuestro interés, con una sola regresora, el modelo es:

```
modelo.1<-lm(PP08D1~PP3E_TOT,
             data = asalariados.con.ingreso.y.horas.cordoba)
```

Ahora hay que verlo:

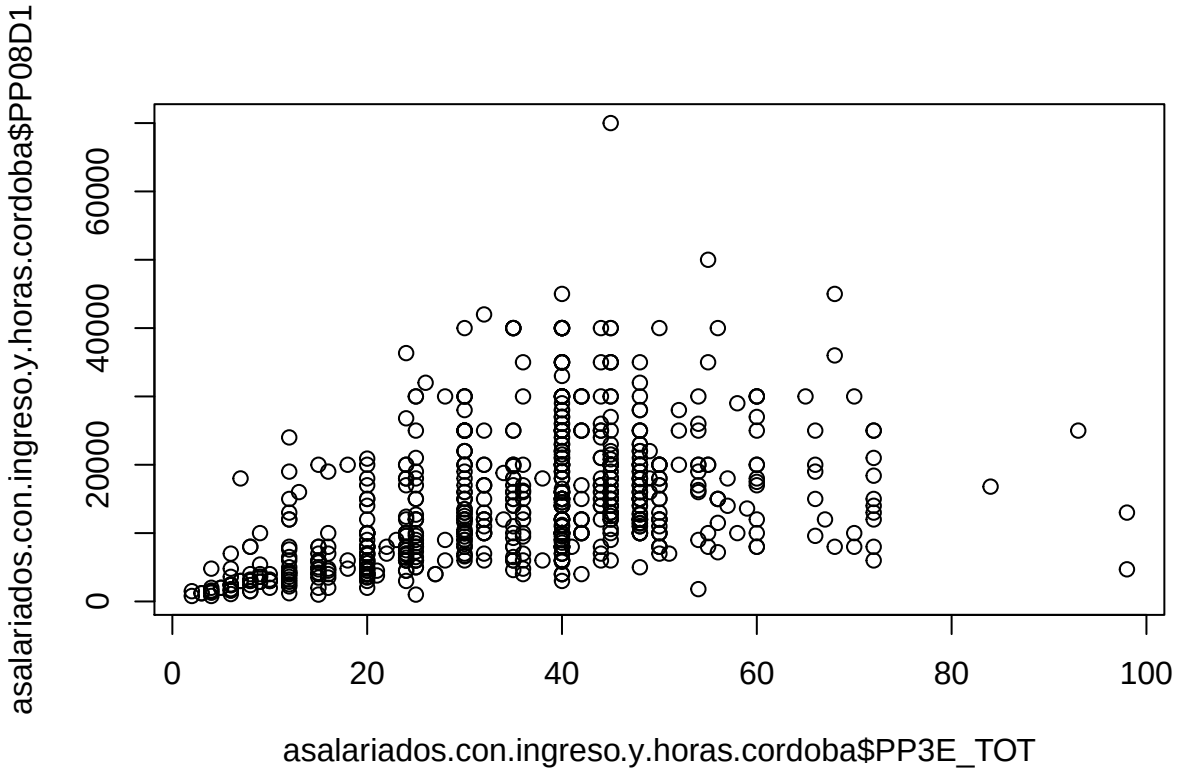
```
summary(modelo.1)
```

```
##
## Call:
## lm(formula = PP08D1 ~ PP3E_TOT, data = asalariados.con.ingreso.y.horas.cordoba)
##
## Residuals:
```

```
##    Min     1Q Median     3Q    Max
## -26631  -5543  -2035   4044  52763
##
## Coefficients:
##             Estimate Std. Error t value             Pr(>|t|)
## (Intercept)  5271.47     813.69   6.478       0.000000000171 ***
## PP3E_TOT      265.91      20.85  12.754 < 0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 8529 on 722 degrees of freedom
## Multiple R-squared:  0.1839, Adjusted R-squared:  0.1827
## F-statistic: 162.7 on 1 and 722 DF,  p-value: < 0.00000000000000022
```

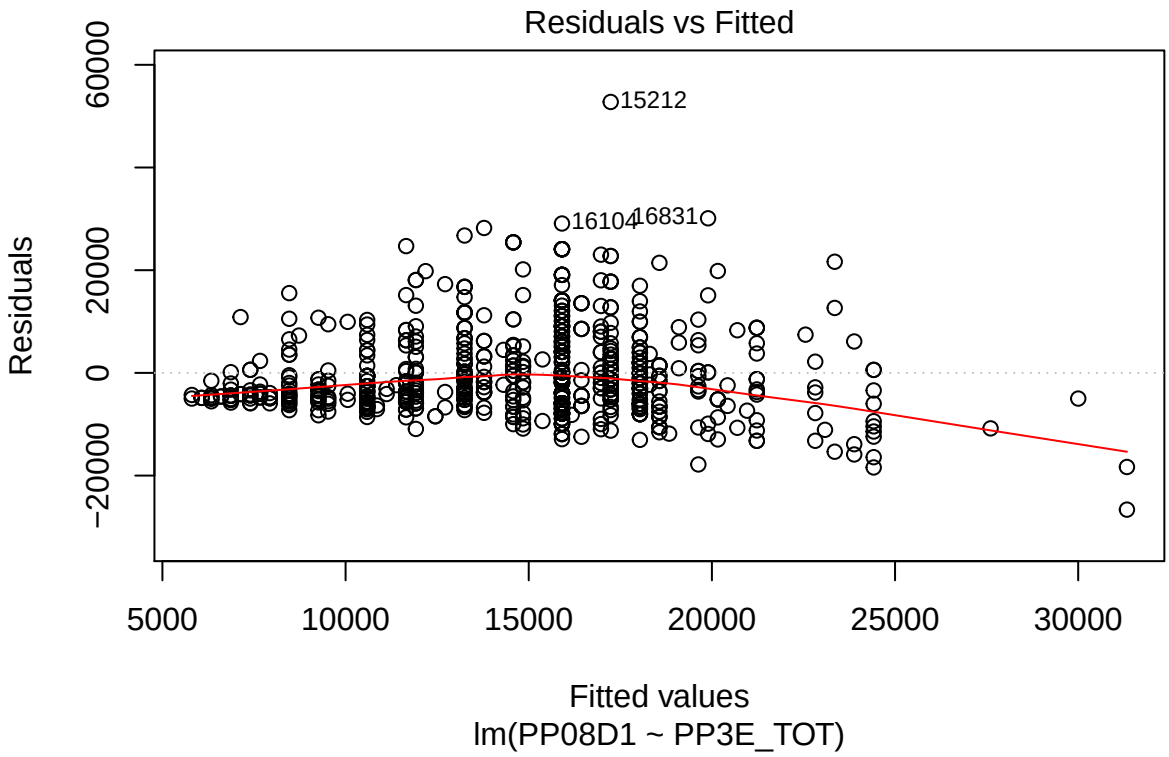
La visualización de la nube de puntos es:

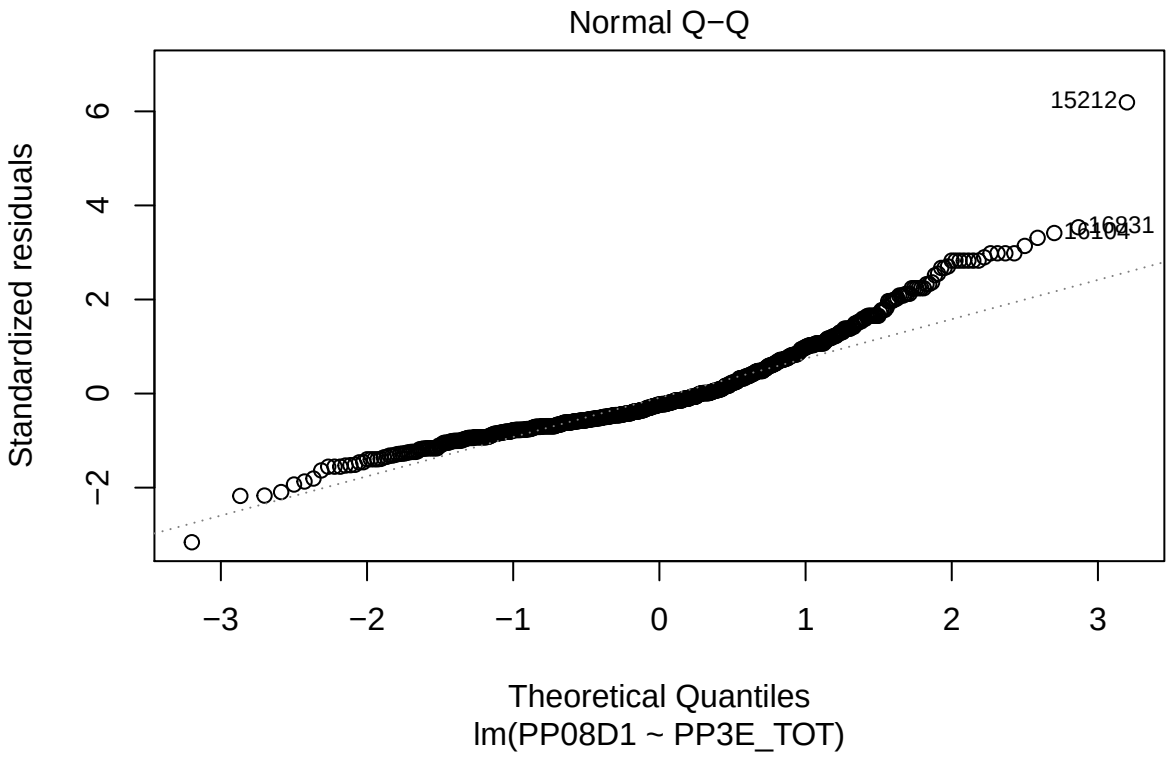
```
plot(asalariados.con.ingreso.y.horas.cordoba$PP3E_TOT,
     asalariados.con.ingreso.y.horas.cordoba$PP08D1)
```

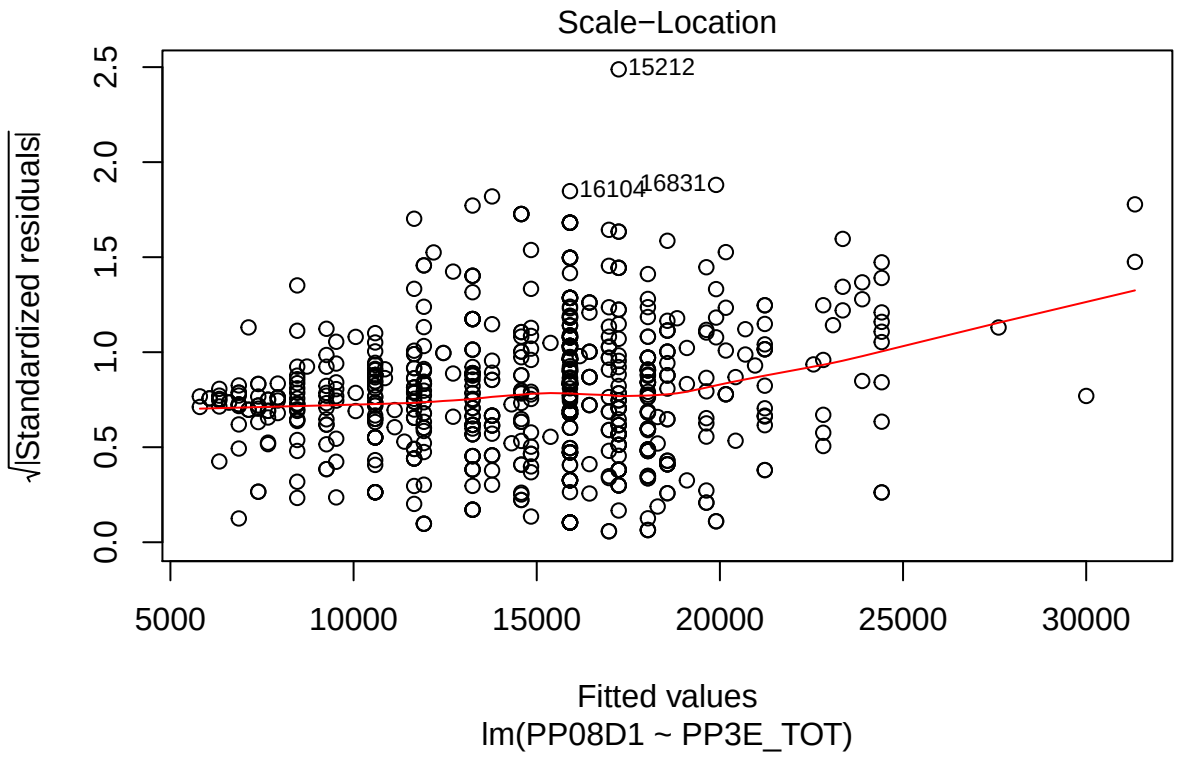


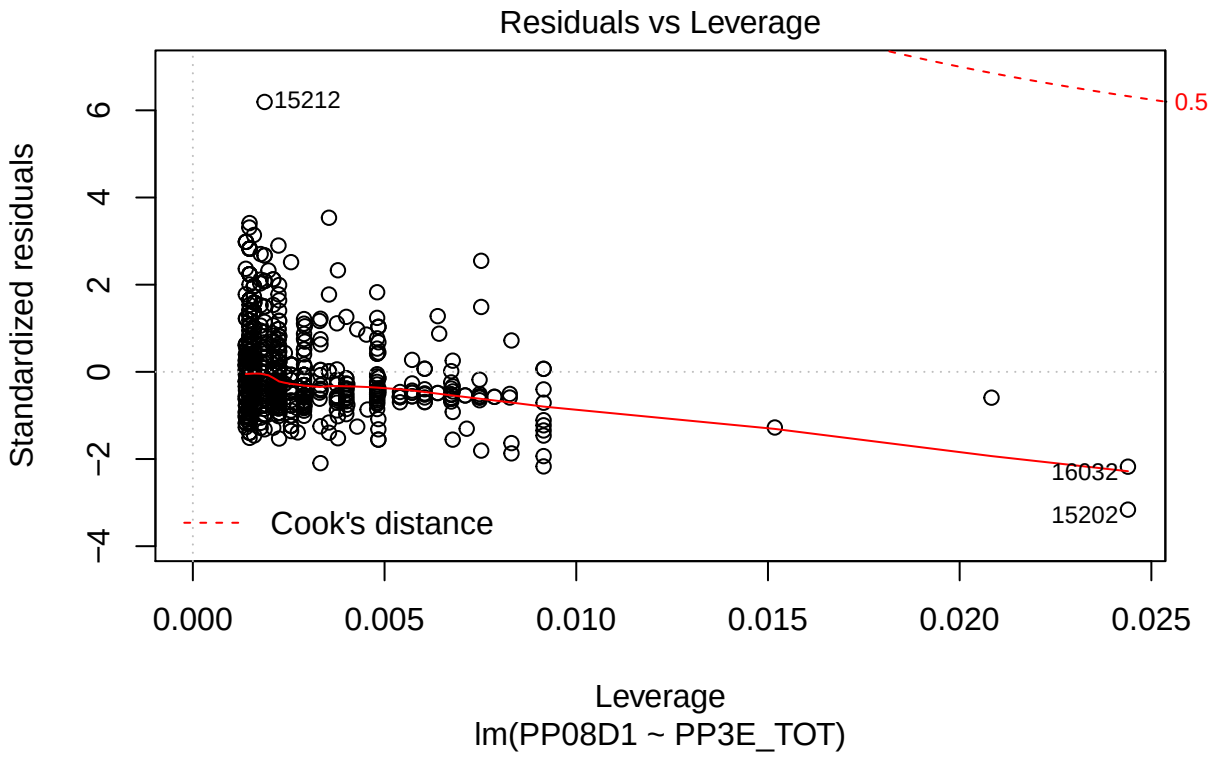
Y los gráficos asociados al modelo:

```
plot(modelo.1)
```


Residuals vs Fitted
15212
16104
16831
Residuals
60000
20000
0
−20000
5000
10000
15000
20000
25000
30000
Fitted values
lm(PP08D1 ~ PP3E_TOT)


Normal Q−Q
Standardized residuals
Theoretical Quantiles
lm(PP08D1 ~ PP3E_TOT)
15212
16931
16104
−3
−2
−1
0
1
2
3
−2
0
2
4
6


Scale−Location
15212
16104
16831
√|Standardized residuals|
0.0
0.5
1.0
1.5
2.0
2.5
5000
10000
15000
20000
25000
30000
Fitted values
lm(PP08D1 ~ PP3E_TOT)



Cuando se incorpora la edad resulta:

```
modelo.2<-lm(PP08D1~PP3E_TOT+edad,
             data = asalariados.con.ingreso.y.horas.cordoba)
summary(modelo.2)
```

```
##
## Call:
## lm(formula = PP08D1 ~ PP3E_TOT + edad, data = asalariados.con.ingreso.y.horas.cordoba)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -24076  -5052  -1431   3785  49511
##
## Coefficients:
##             Estimate Std. Error t value             Pr(>|t|)
## (Intercept) -1029.37    1188.15  -0.866                0.387
## PP3E_TOT      256.98      20.21  12.714 < 0.0000000000000002 ***
## edad          177.75      25.10   7.081     0.00000000000341 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 8253 on 721 degrees of freedom
## Multiple R-squared:  0.2369, Adjusted R-squared:  0.2348
## F-statistic: 111.9 on 2 and 721 DF,  p-value: < 0.00000000000000022
```

La comparación por sexos puede hacerse incluyéndola como variable dummy:

```
modelo.3<-lm(PP08D1~PP3E_TOT+edad+sexo,
             data = asalariados.con.ingreso.y.horas.cordoba)

summary(modelo.3)
```

```
##
## Call:
## lm(formula = PP08D1 ~ PP3E_TOT + edad + sexo, data = asalariados.con.ingreso.y.horas.cordoba)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -23516  -5140  -1426   3824  48478
##
## Coefficients:
##             Estimate Std. Error t value             Pr(>|t|)
## (Intercept)   627.08    1262.47   0.497              0.61955
## PP3E_TOT      231.98      21.18  10.953 < 0.0000000000000002 ***
## edad          186.72      25.01   7.465    0.000000000000241 ***
## sexomujer   -2359.62     646.33  -3.651              0.00028 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 8184 on 720 degrees of freedom
## Multiple R-squared:  0.2508, Adjusted R-squared:  0.2477
## F-statistic: 80.35 on 3 and 720 DF,  p-value: < 0.00000000000000022
```

Para incluir una interacción, por ejemplo entre edad y sexo:

```
modelo.4<-lm(PP08D1~PP3E_TOT+edad+sexo+edad*sexo,
             data = asalariados.con.ingreso.y.horas.cordoba)

summary(modelo.4)
```

```
##
## Call:
## lm(formula = PP08D1 ~ PP3E_TOT + edad + sexo + edad * sexo, data = asalariados.con.ingreso.y.horas.c
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -22810  -4902  -1509   3826  47367
##
## Coefficients:
##                Estimate Std. Error t value             Pr(>|t|)
## (Intercept)    -1377.32    1489.31  -0.925                0.3554
## PP3E_TOT         230.07      21.12  10.896 < 0.0000000000000002 ***
## edad             243.87      33.73   7.229     0.00000000000124 ***
## sexomujer       2310.64    1966.45   1.175                0.2404
## edad:sexomujer  -125.37      49.88  -2.514                0.0122 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 8153 on 719 degrees of freedom
## Multiple R-squared:  0.2573, Adjusted R-squared:  0.2532
## F-statistic: 62.28 on 4 and 719 DF,  p-value: < 0.00000000000000022
```

Comparación de grupos

Para comparar los ingresos salariales de mujeres y varones, se requiere que los grupos sean independientes, para lo cual haremos la comparación entre jefes y jefas de hogar.
La prueba t para comparar medias de grupos independientes se llama "t.test" y su argumento tiene la misma estructura que la del modelo lineal. En este caso los datos de origen están constituídos por un subconjunto de la base que solo contiene jefes y jefas:

```
t.test(PP08D1~sexo, data = subset(
  asalariados.con.ingreso.cordoba,
  asalariados.con.ingreso.cordoba$CH03==1))
```

```
##
##  Welch Two Sample t-test
##
## data:  PP08D1 by sexo
## t = 5.0727, df = 318.67, p-value = 0.0000006672
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  3333.449 7557.453
## sample estimates:
## mean in group varon mean in group mujer
##            19838.15            14392.70
```

En lugar de construir otra matriz solo con los jefes, indicamos que tome como dato solo aquellos casos que tengan, como relación de parentesco, jefe/jefa.
Para una prueba unilateral derecha se pide:

```
t.test(PP08D1~sexo, data = subset(
  asalariados.con.ingreso.cordoba,
  asalariados.con.ingreso.cordoba$CH03==1), alternative="greater")
```

```
##
##  Welch Two Sample t-test
##
## data:  PP08D1 by sexo
## t = 5.0727, df = 318.67, p-value = 0.0000003336
## alternative hypothesis: true difference in means is greater than 0
## 95 percent confidence interval:
##  3674.587      Inf
## sample estimates:
## mean in group varon mean in group mujer
##            19838.15            14392.70
```

El paquete `ggplot`

La lógica

La construcción de los gráficos en ggplot se hace por medio de capas que se van agregando. Las capas tienen cinco componentes:

- Los datos, que es la base de donde provienen la variables que se van a graficar. Si más tarde se grafica lo mismo para otra base, solo se debe cambiar ese origen, lo mismo si la base se modifica.
- Un conjunto de mapeos estéticos (*aes*), que describen el modo en que las variables de la base van a ser representadas en las propiedades estéticas de la capa.
- El *geom*, que describe la figura geométrica que se va a usar para dibujar la capa.
- La transformación estadística (*stat*) que opera sobre los datos originales para sintetizarlos de modo que se los pueda representar.
- Los ajustes de posición

Los gráficos generados con este paquete pueden exportarse con formato gráfico o como pdf. Este paquete está muy bien documentado, cada cosa que quiera hacerse puede buscarse y hay ayudas que suelen resolver casi todos los problemas. Si no se encuentra alguien que haya tenido la misma pregunta, puede formularse en un foro, muy recomendable es https://es.stackoverflow.com/, que está en español o más amplia, la versión en inglés: https://stackoverflow.com/

Cargamos `ggplot2` en la sesión actual:

```
library(ggplot2)
```

Aplicaciones

La primera instrucción para crear un gráfico es ggplot(). Esta instrucción puede tener el origen de los datos y algún mapeo estético; pero también puede quedar en blanco y ubicar esa información en las capas siguientes. Si se ubican los datos en esa primera instrucción, todas las capas usarán esos datos, lo mismo para el mapeo estético, alternativamente, cada capa puede especificarlo.

Una variable cuantitativa

Se define la base del gráfico, que indica de dónde provienen los datos:

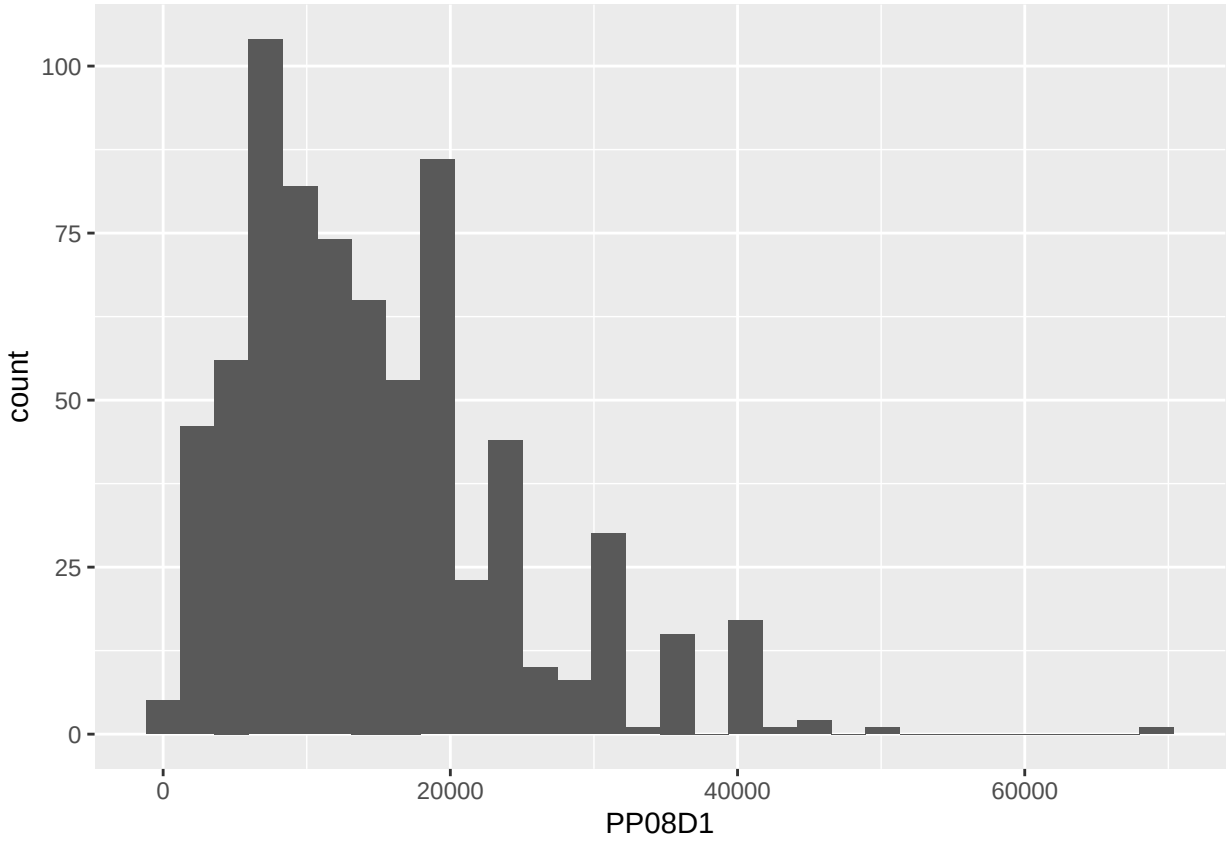
```
ggplot(asalariados.con.ingreso.y.horas.cordoba)
```

Que solo muestra la capa base.

Se agrega una capa con un histograma:

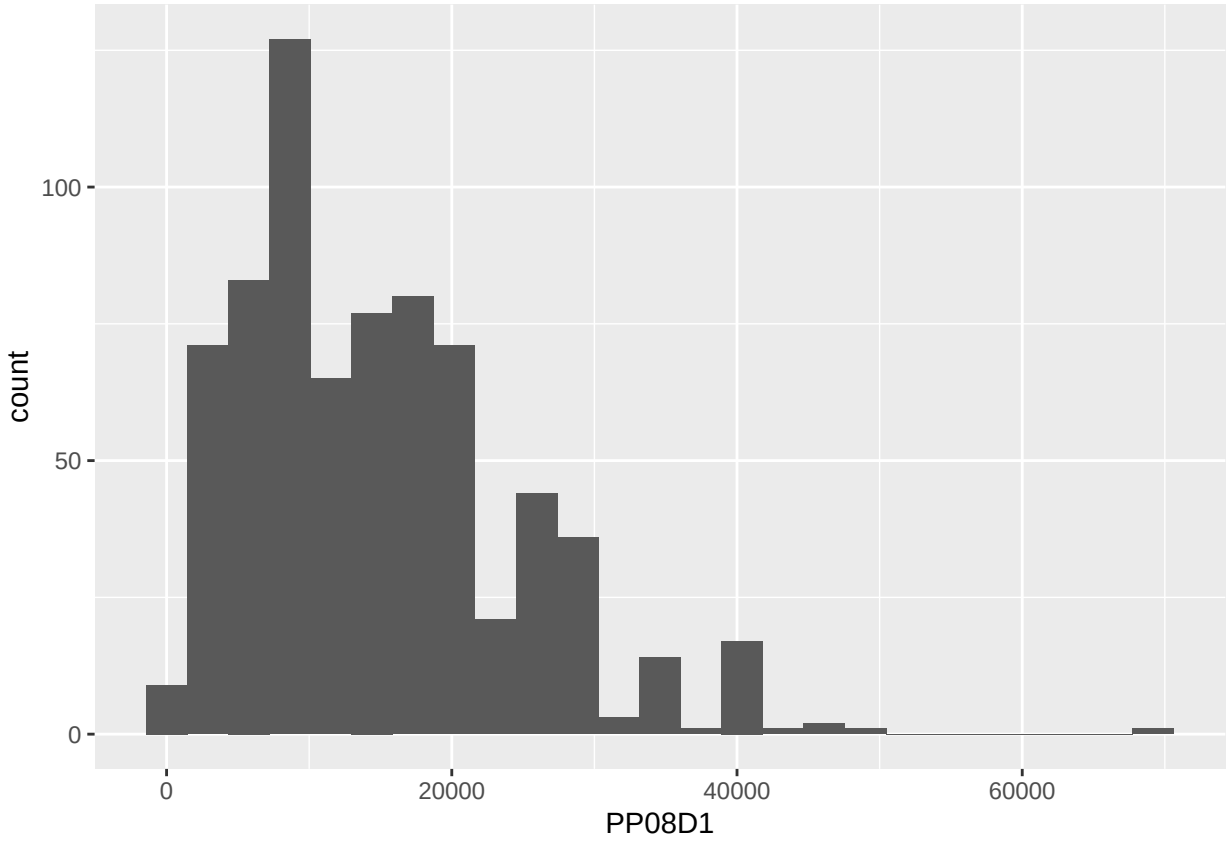
```
ggplot(
  asalariados.con.ingreso.y.horas.cordoba)+
  geom_histogram(aes(PP08D1))
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



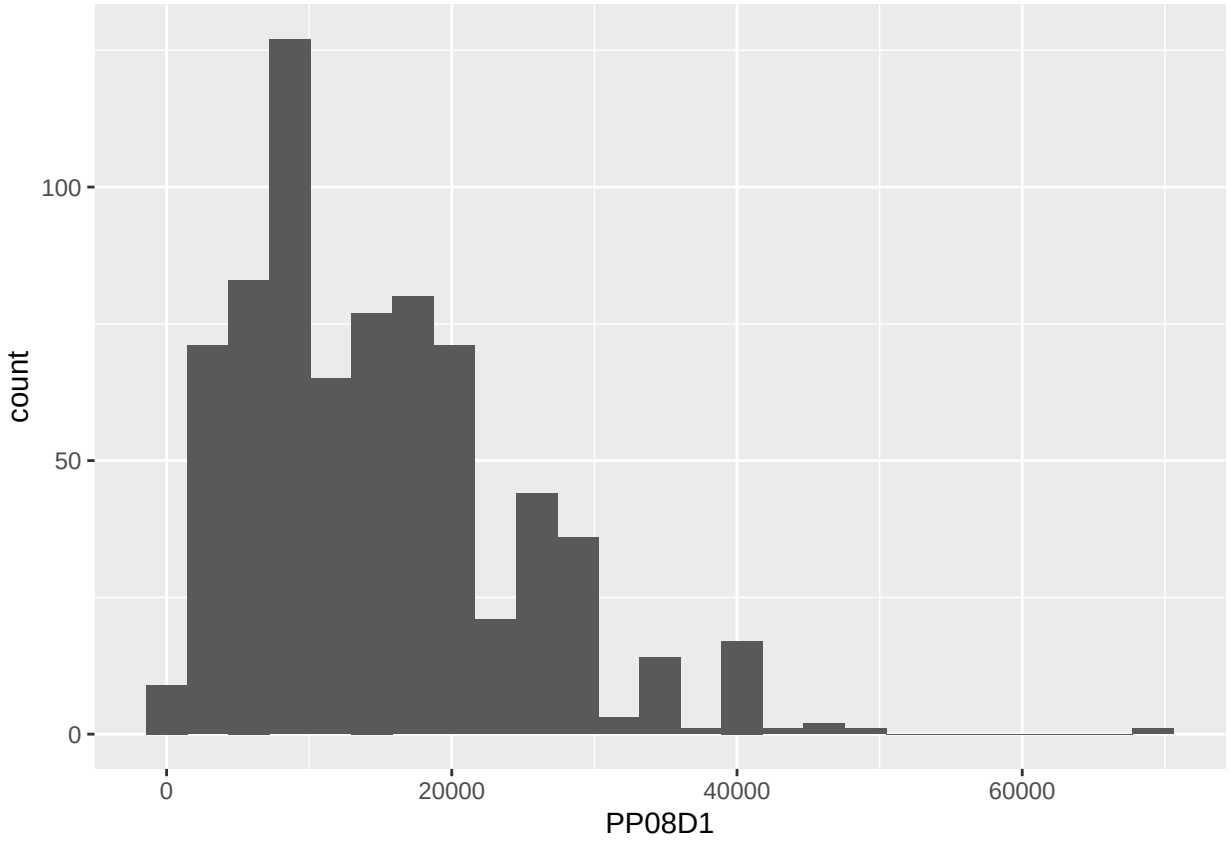
El mensaje indica que se usaron 30 intervalos y que eso puede ajustarse mejor si se lo desea.
El mismo resultado se logra ubicando la estética en la primera instrucción y agregamos, en la capa del histograma, que queremos 25 intervalos:

```
ggplot(
  asalariados.con.ingreso.y.horas.cordoba, aes(PP08D1))+
  geom_histogram(bins=25)
```



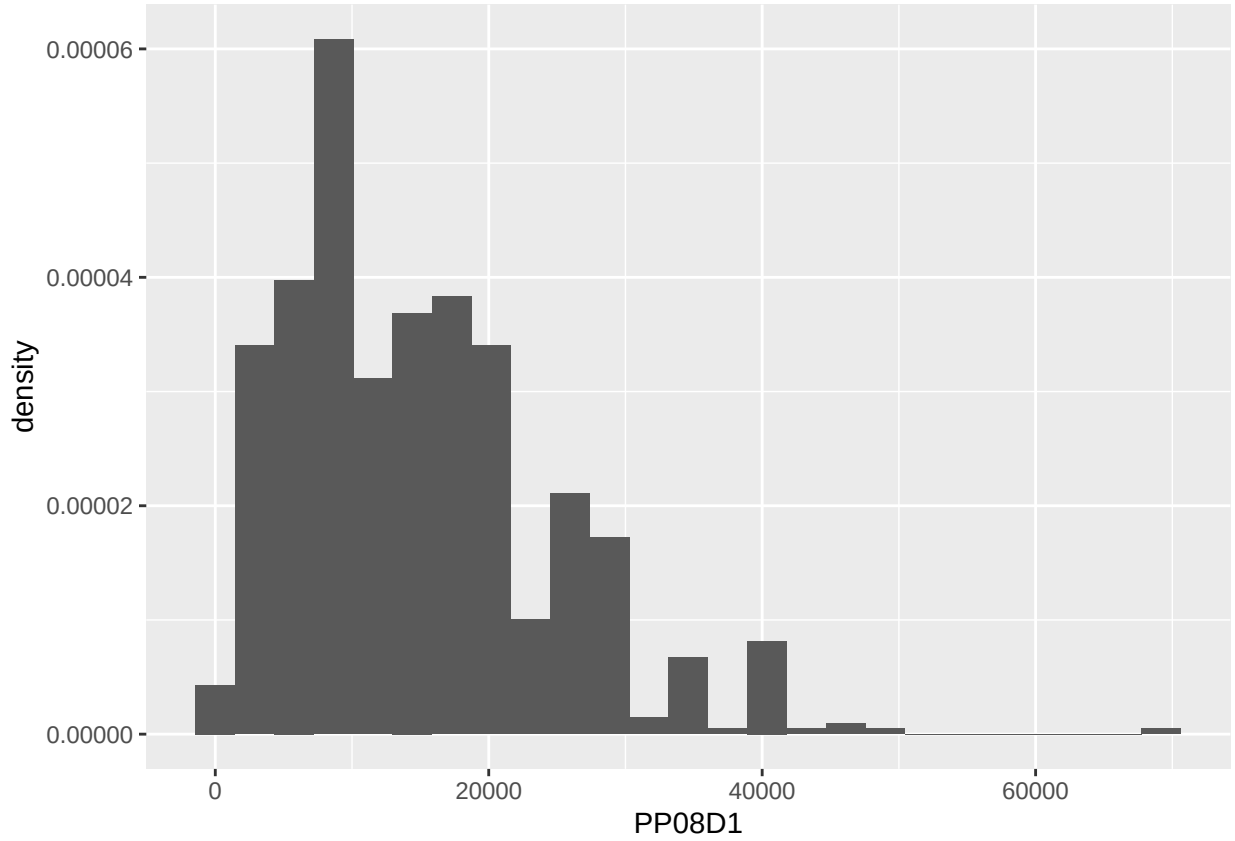
O poniendo todo en la capa del histograma

```
ggplot()+
  geom_histogram(
    data=asalariados.con.ingreso.y.horas.cordoba, aes(PP08D1), bins=25)
```



Solo que así debemos indicar que `asalariados.con.ingreso.y.horas.cordoba` son los datos.
La información que vaya en la intrucción ggplot() será válida para todas las capas que se agreguen, la que se incluya en una capa solo se toma para esa capa.
Para representar frecuencias relativas en el eje de ordenadas, hay que indicar que y mida *densidad*, esto se pone entre dos puntos porque es un resultado que ggplot calcula internamente, las frecuencias relativas son calculadas al hacer las transformaciones que pide el gráfico.
O poniendo todo en la capa del histograma

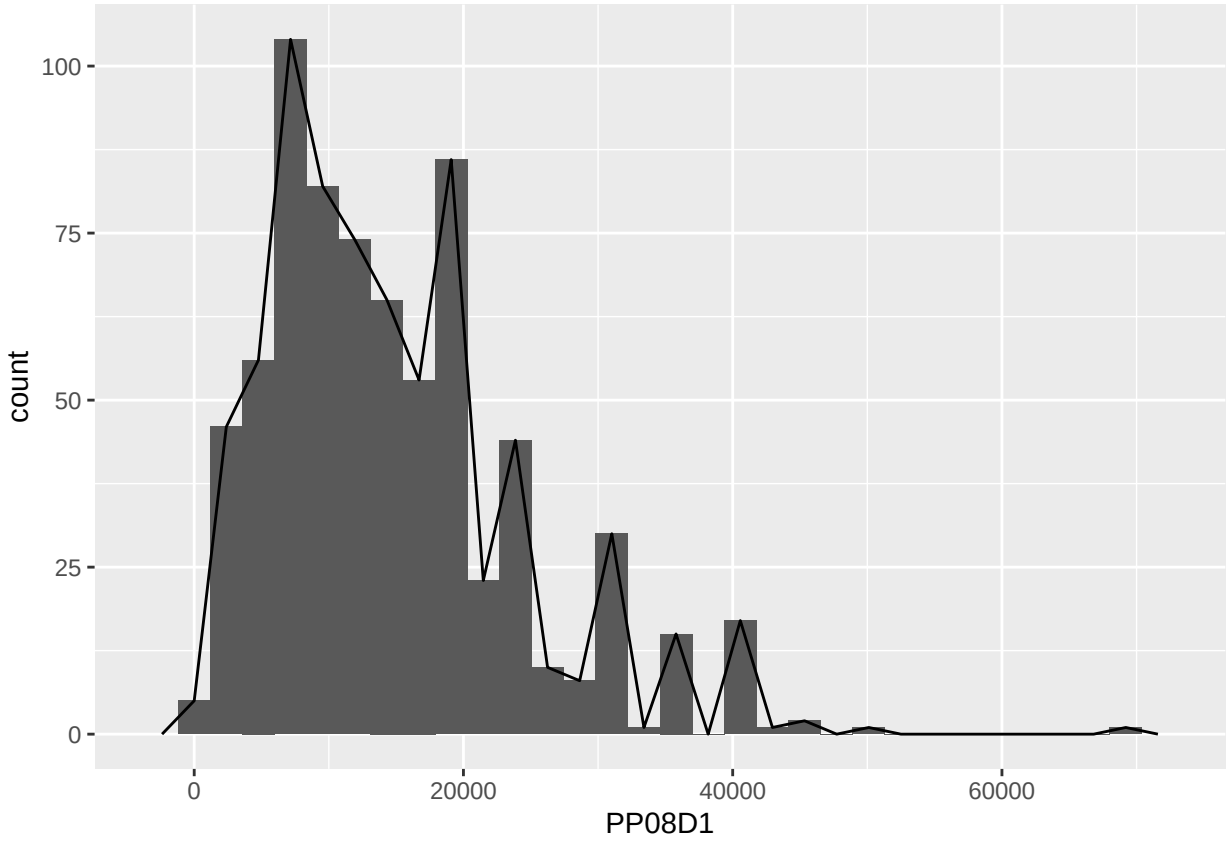
```
ggplot()+
  geom_histogram(
    data=asalariados.con.ingreso.y.horas.cordoba,
    aes(PP08D1, y=..density..), bins=25)
```



El polígono de frecuencias se superpone como capa:

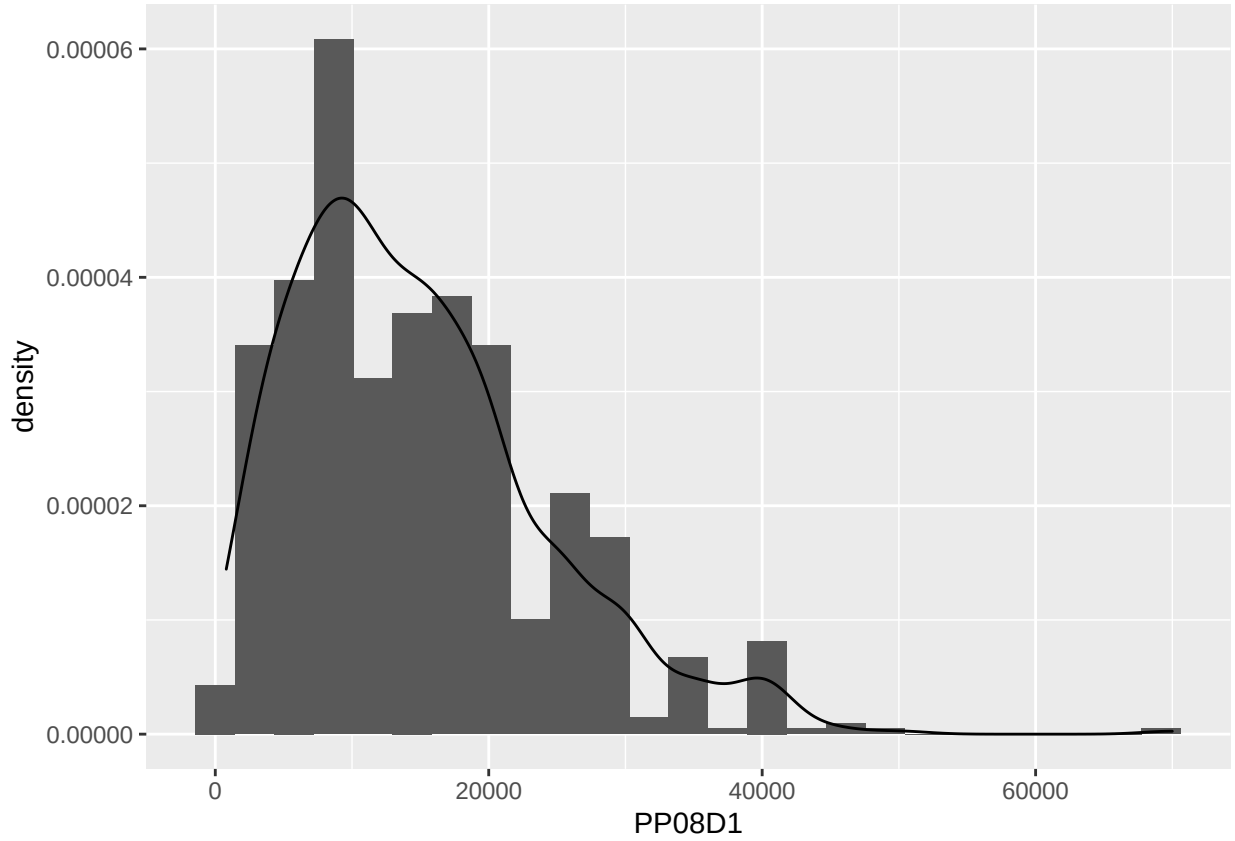
```
ggplot(
  asalariados.con.ingreso.y.horas.cordoba, aes(PP08D1))+
  geom_histogram()+geom_freqpoly()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Al igual que la curva de densidad:

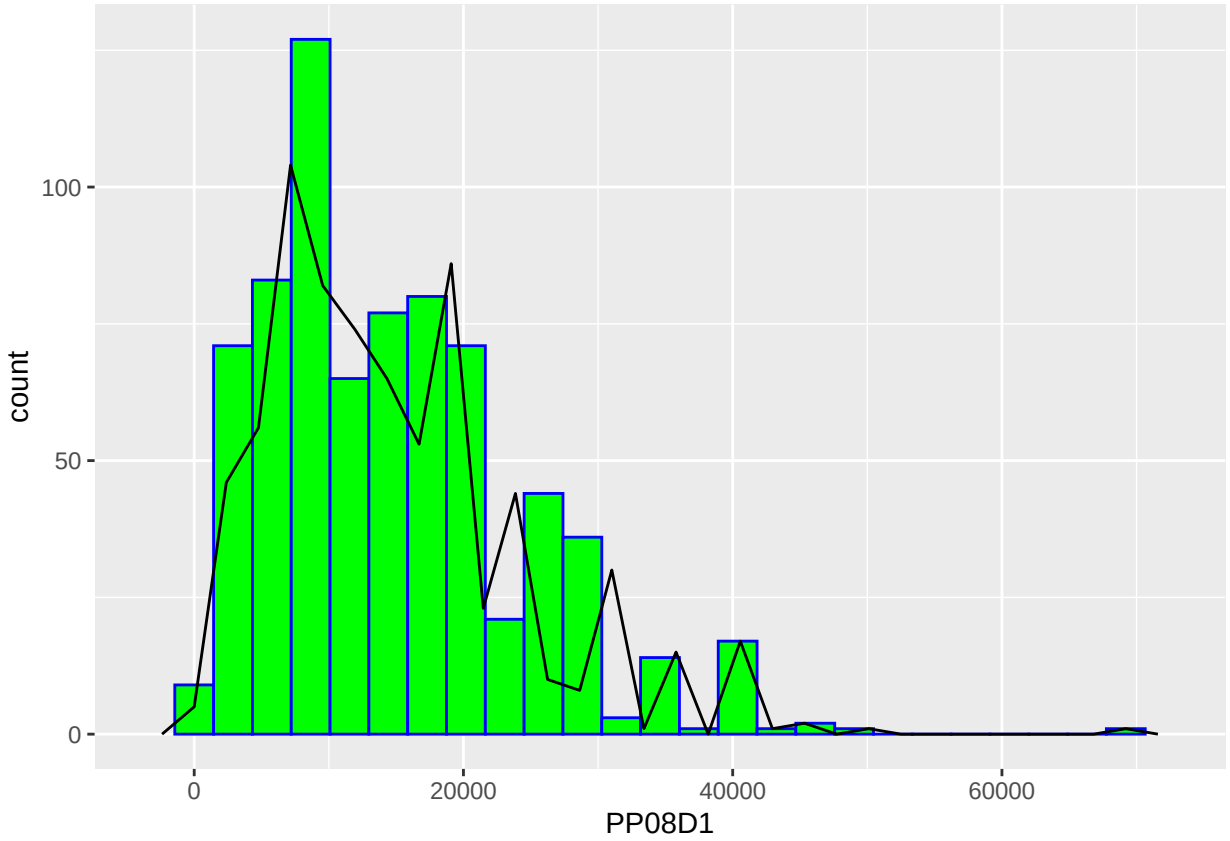
```
ggplot(
  asalariados.con.ingreso.y.horas.cordoba, aes(PP08D1))+
  geom_histogram(aes(y=..density..), bins=25)+
  geom_density()
```



Se lo puede pintar de verde, con contornos azules:

```
ggplot(asalariados.con.ingreso.y.horas.cordoba, aes(PP08D1))+
  geom_histogram(fill="green", col="blue", bins = 25)+
  geom_freqpoly()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

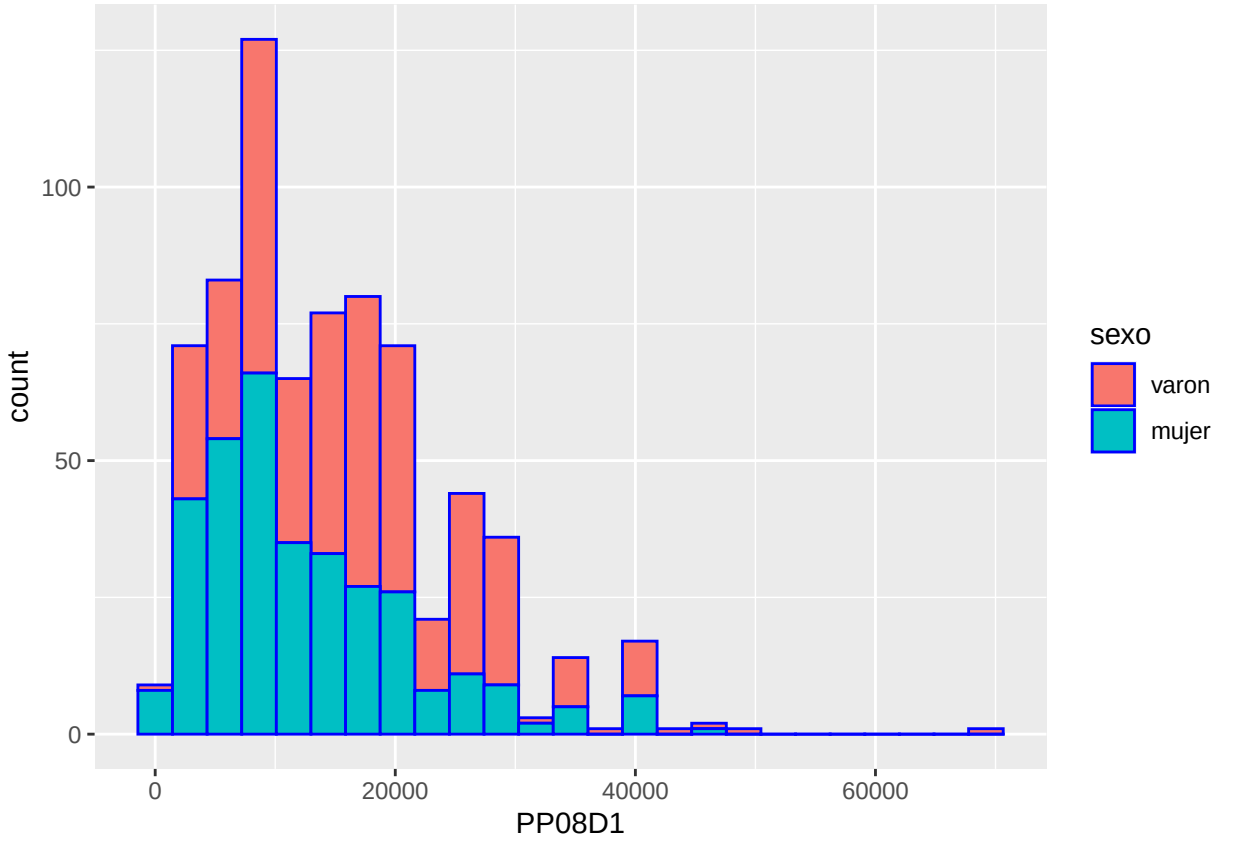


- `fill` es para el relleno de las barras
- `col` es para el contorno

Una cuantitativa comparada entre grupos

En el ejemplo anterior, los colores están **fijados** a los valores constantes "verde" o "azul". Pero se lo puede **mapear** a los valores de una variable, por ejemplo `sexo`:

```
ggplot(asalariados.con.ingreso.y.horas.cordoba)+
  geom_histogram(aes(PP08D1, fill=sexo), col="blue", bins = 25)
```



Los contornos están **fijados**, pero el relleno está **mapeado**.

Mapear es vincular valores de una variable a atributos estéticos del gráfico, como el color, la forma, o el tamaño, según qué gráfico sea. El comando `fill=sexo` indica que rellene según las categorías de esa variable. **Fijar** es establecer una atributo en un valor predeterminado para todo el gráfico. Las expresiones `size=3` o `fill="red"` fijan el tamaño en el valor 3 o el color en rojo, sin tener en cuenta alguna variable.

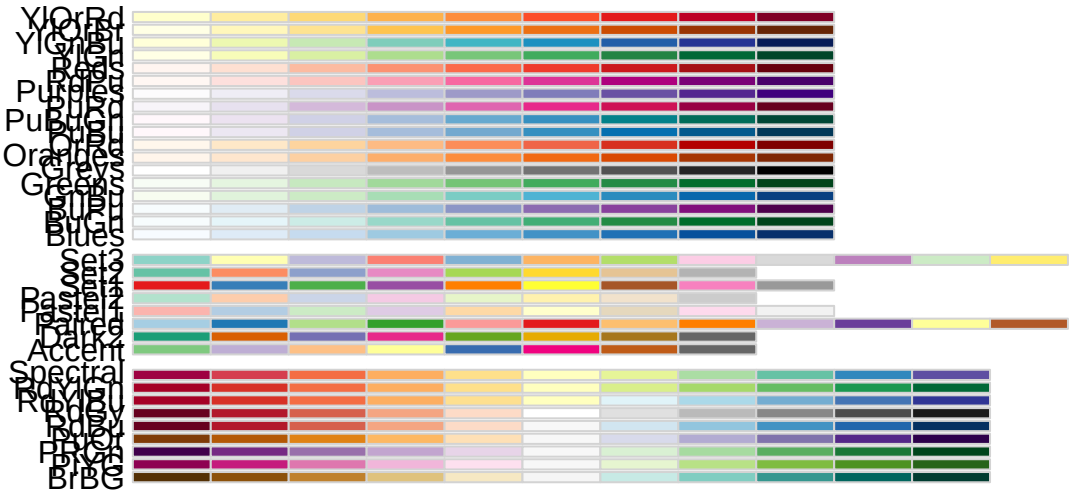
Para mapear, la instrucción debe ir dentro de la estética (aes), mientras que para fijar, va fuera y entre comillas.

Los que se ven son los colores por defecto que `ggplot2` usa para mapear, eso se puede cambiar eligiendo diferentes paletas de colores, que funciona como una capa más.
Para contar con un repertorio amplio de paletas hay que cargar el paquete correspondiente:

```
library("RColorBrewer")
```

Y se las puede ver con:

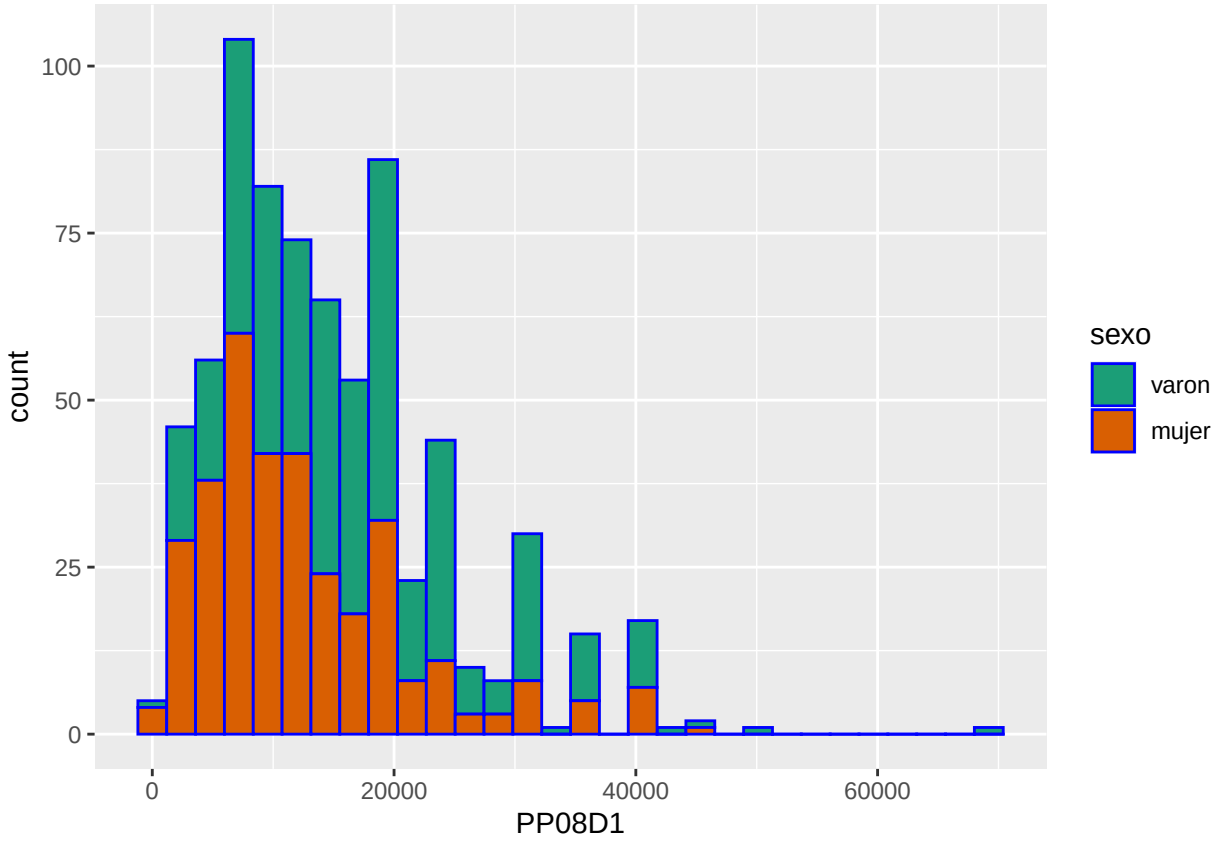
```
display.brewer.all()
```



Si elegimos la paleta *Dark*2

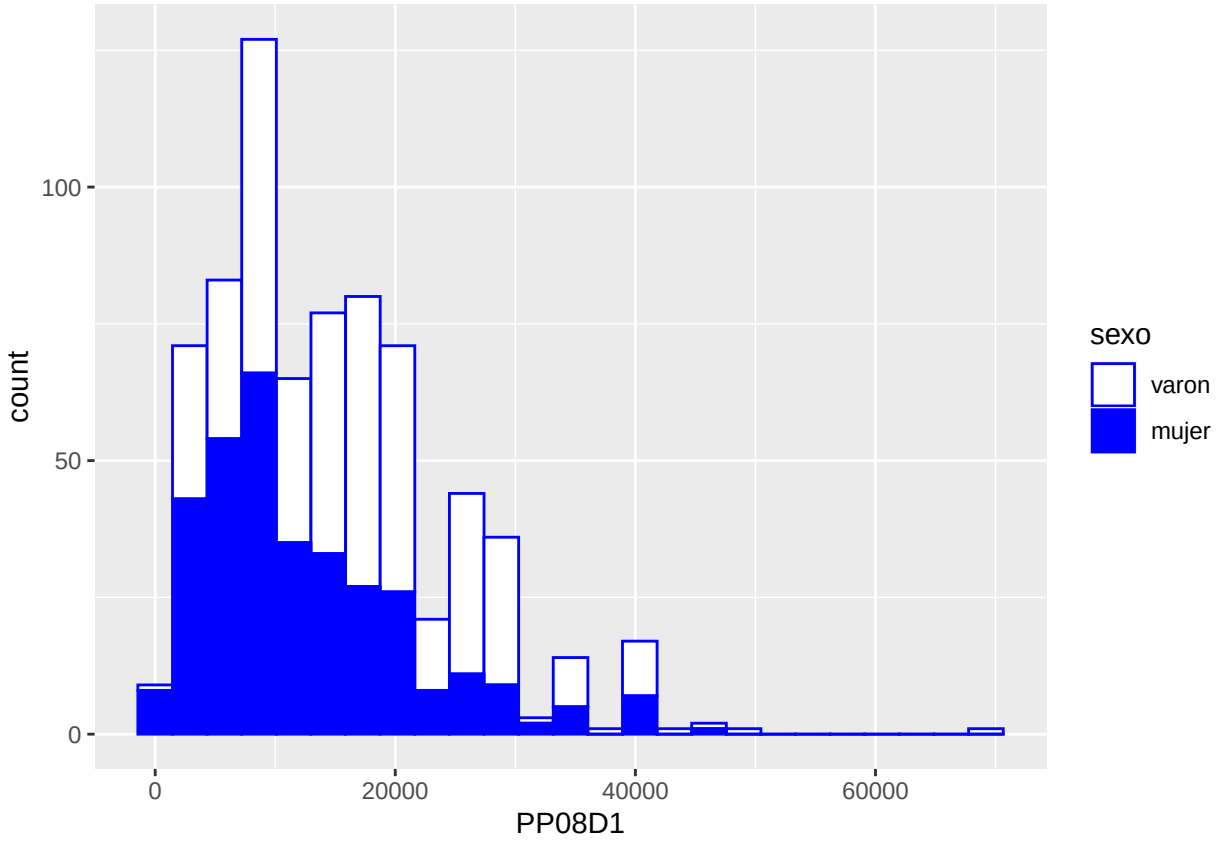
```
ggplot(asalariados.con.ingreso.y.horas.cordoba)+
  geom_histogram(aes(PP08D1, fill=sexo), col="blue")+
  scale_fill_brewer(palette="Dark2")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



O bien se puede elegir manualmente qué color asignar a cada categoría:

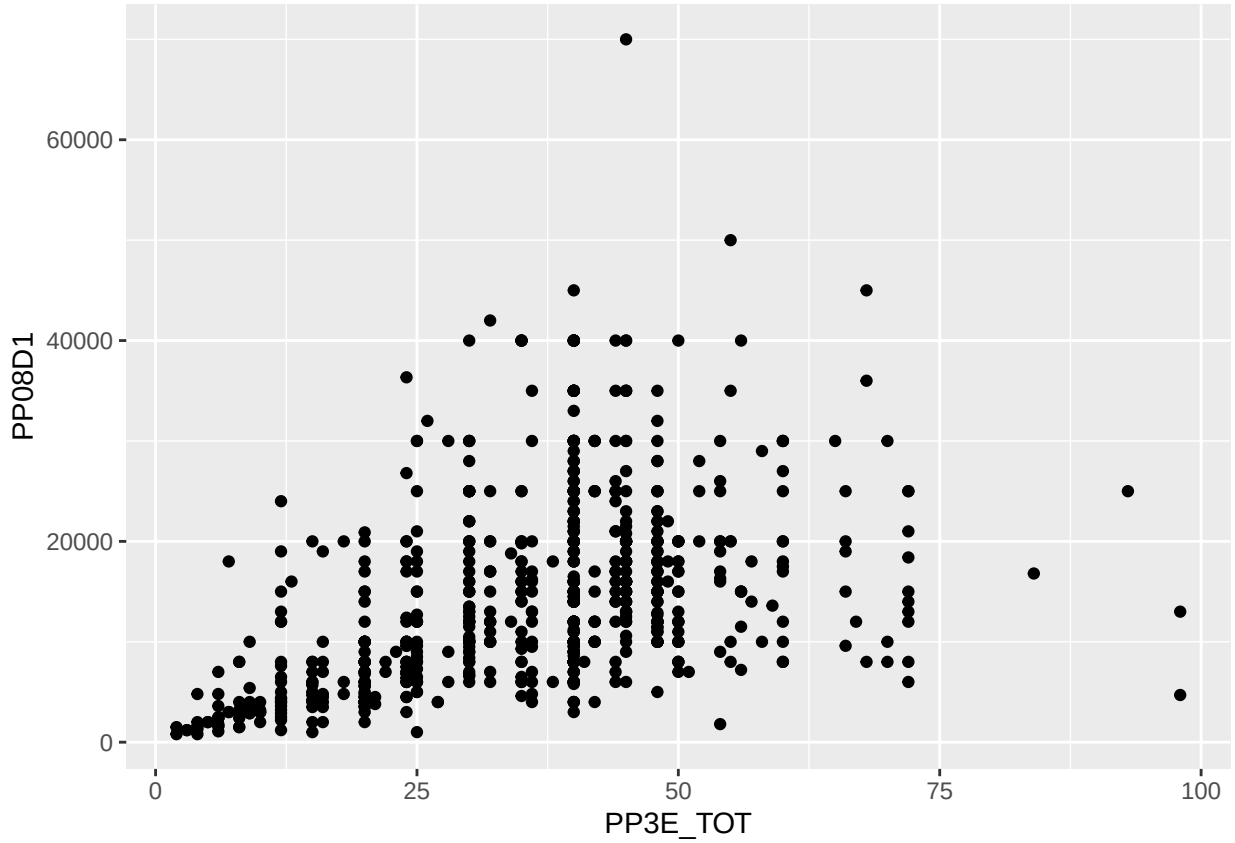
```
ggplot(asalariados.con.ingreso.y.horas.cordoba)+
  geom_histogram(aes(PP08D1, fill=sexo), col="blue", bins = 25)+
  scale_fill_manual(values=c("varon"="white", "mujer"="blue"))
```



Dos variables cuantitativas

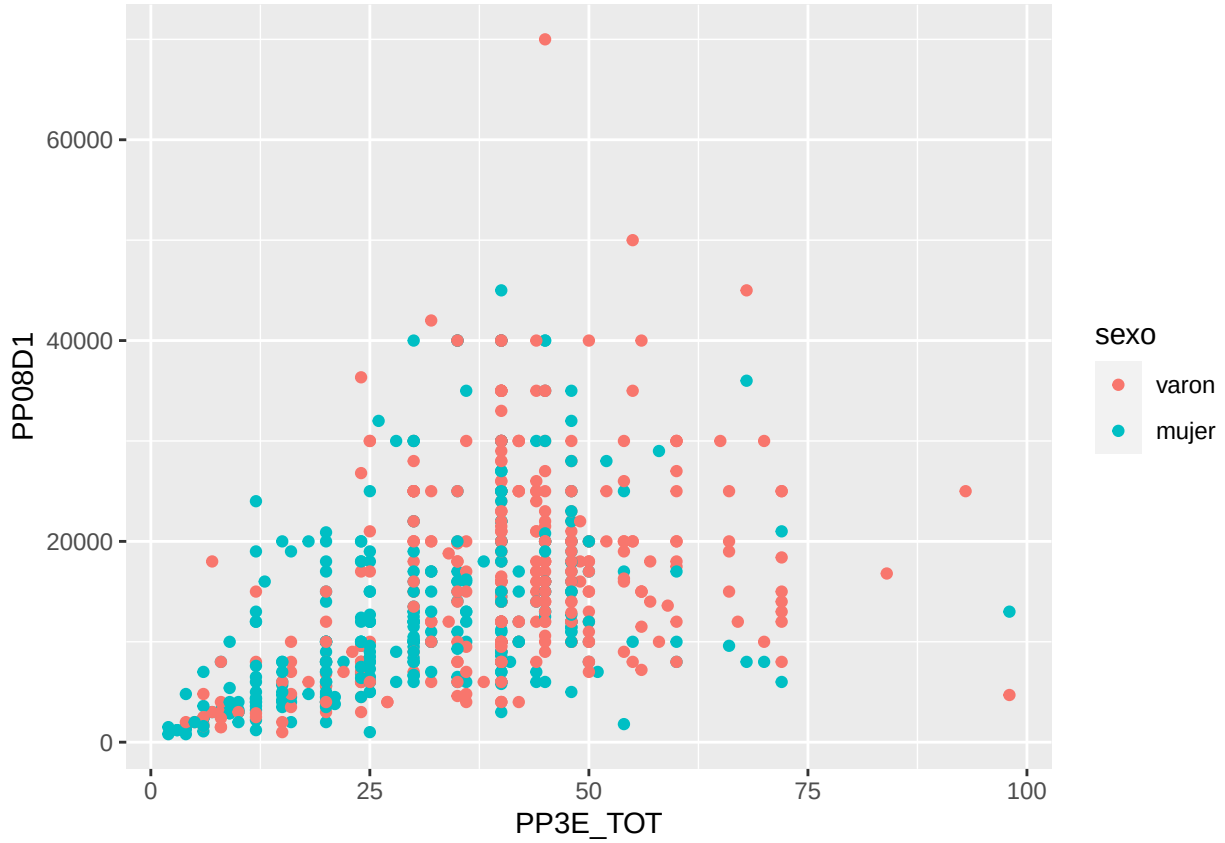
Si se trata de dos variables cuantitativas, como las horas y los ingresos, la capa para el diagrama de dispersión se llama `geom_point` y en la estética deben indicarse las dos variables en el orden x, y:

```
ggplot(
  asalariados.con.ingreso.y.horas.cordoba)+
  geom_point(aes(PP3E_TOT, PP08D1))
```



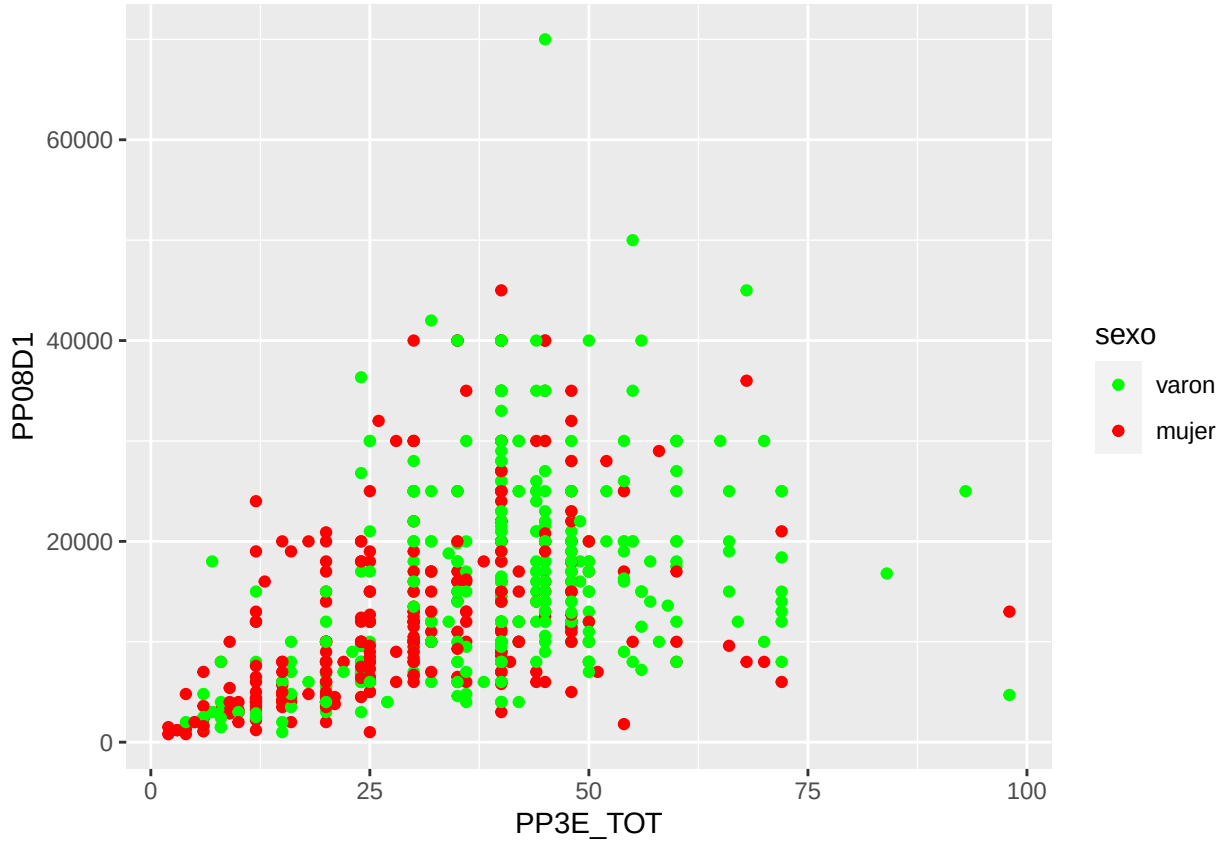
El mapeo de la variable `sexo` al color de los puntos, se pide dentro de la estética:

```
ggplot(
  asalariados.con.ingreso.y.horas.cordoba)+
  geom_point(aes(PP3E_TOT, PP08D1, col=sexo))
```



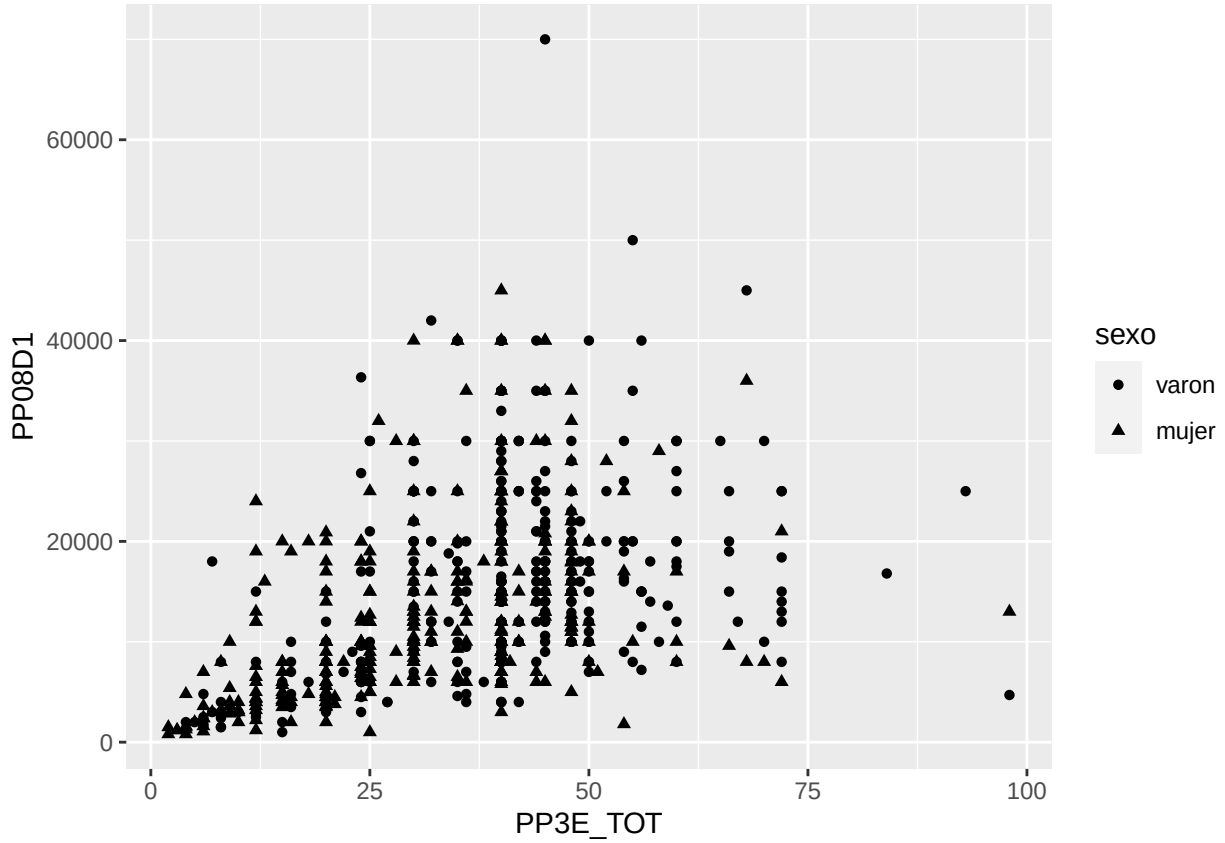
La combinación de colores se elige con la paleta, o bien se establece de manera manual:

```
ggplot(
  asalariados.con.ingreso.y.horas.cordoba)+
  geom_point(aes(PP3E_TOT, PP08D1, col=sexo))+
  scale_colour_manual(values=c("varon"="green", "mujer"="red"))
```



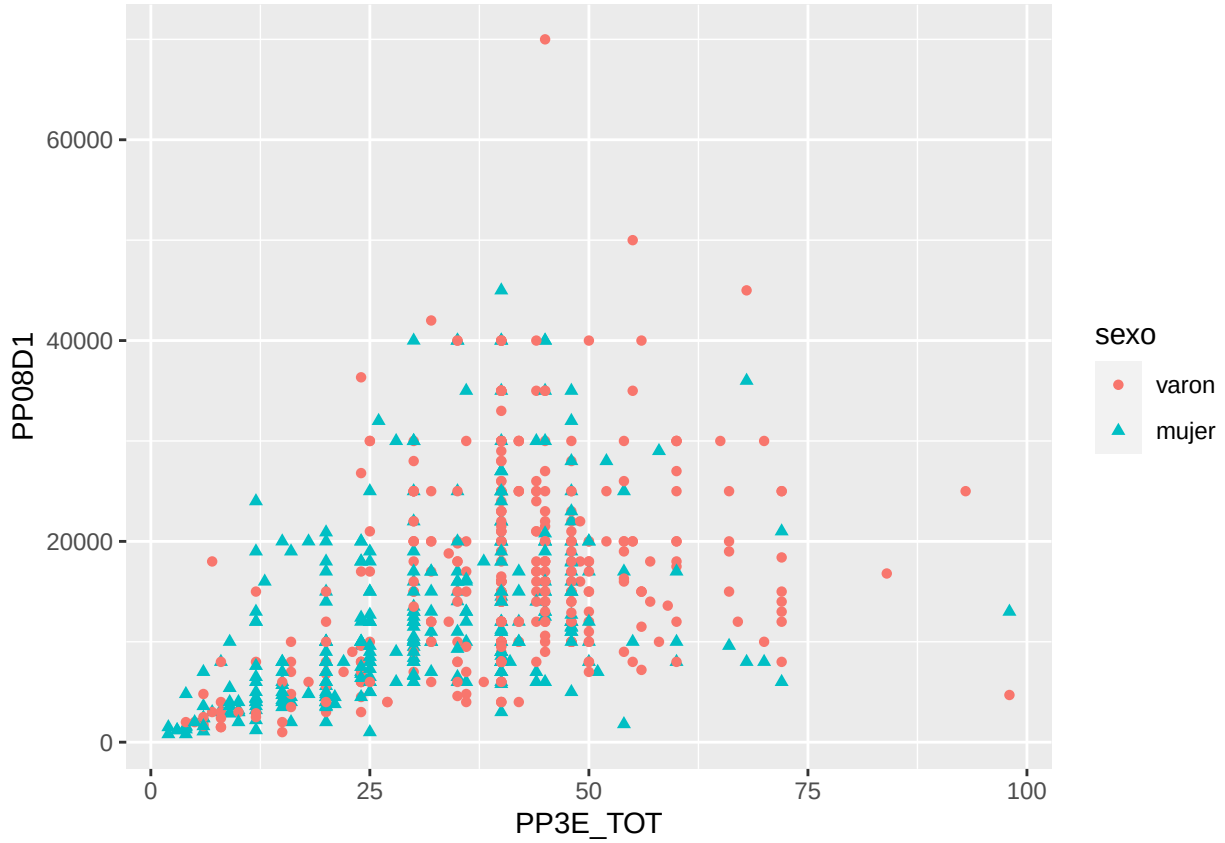
En lugar del color se puede elegir la forma de los puntos:

```
ggplot(
  asalariados.con.ingreso.y.horas.cordoba)+
  geom_point(aes(PP3E_TOT, PP08D1, shape=sexo))
```



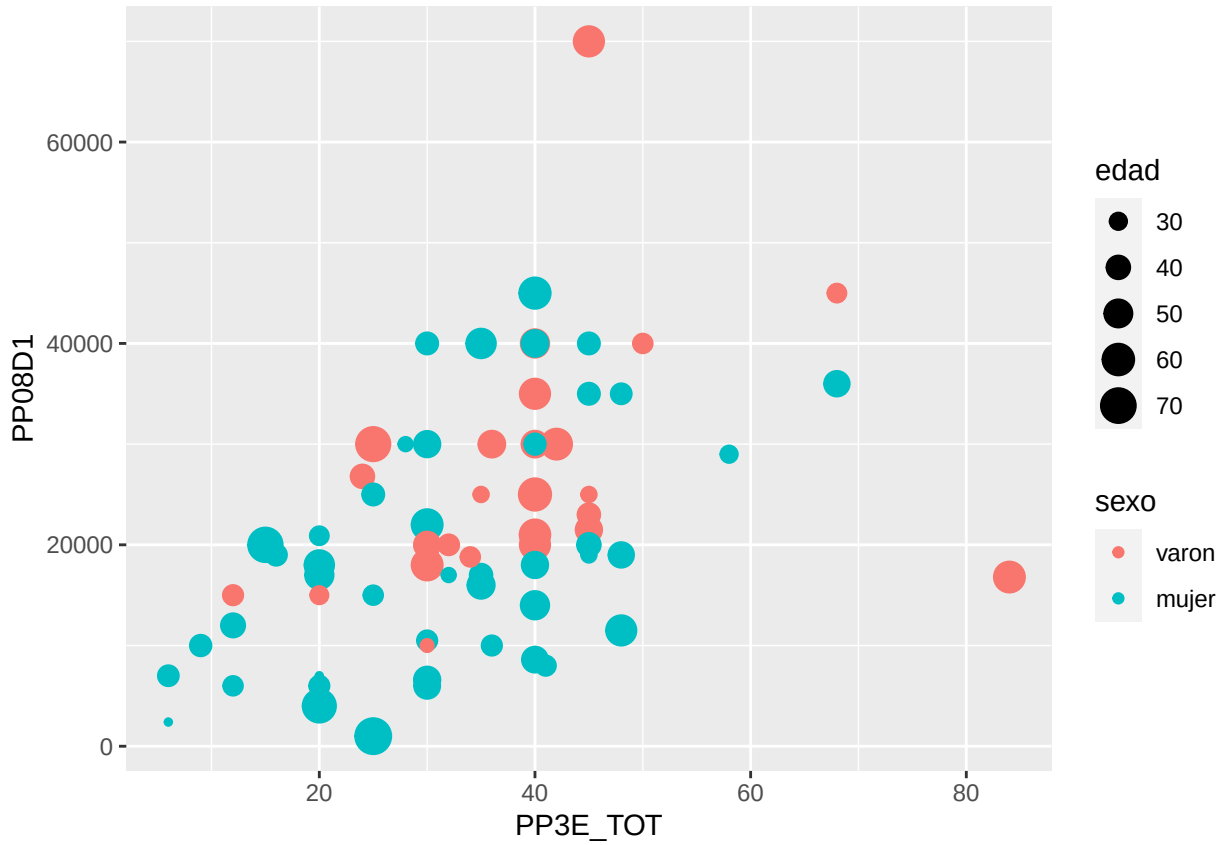
O la misma variable mapeada a ambos atributos gráficos:

```
ggplot(
  asalariados.con.ingreso.y.horas.cordoba)+
  geom_point(aes(PP3E_TOT, PP08D1, col=sexo, shape=sexo))
```



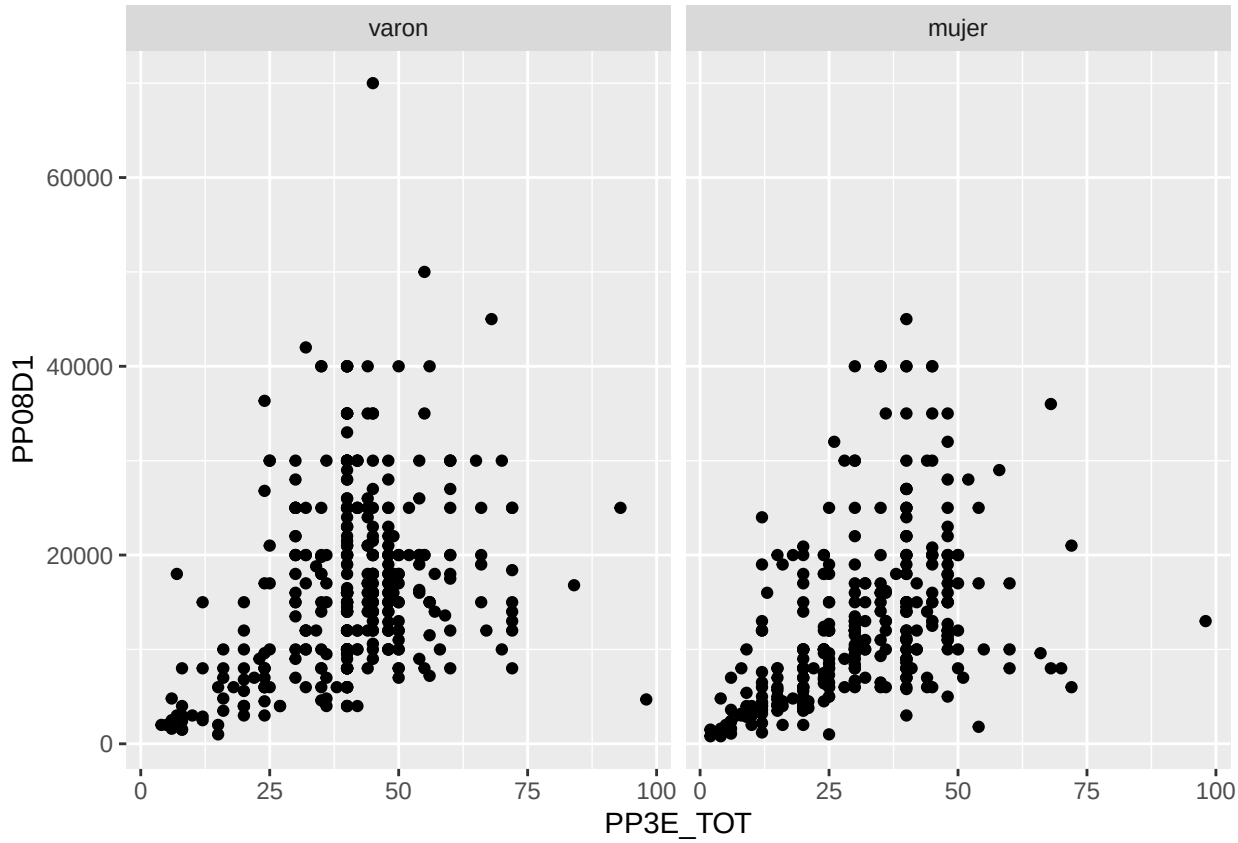
Solo a efectos de ver el funcionamiento de los diferentes mapeos, llevamos el tamaño de los puntos a otra variable cuantitativa (la edad) así se dibujan puntos cuyo tamaño es proporcional a los valores de esa variable. A fin de reducir la cantidad de puntos de la nube, retenemos solo personas con estudios universitarios completos y que sean jefes o jefas de hogar.

```
ggplot(
  subset(
    asalariados.con.ingreso.y.horas.cordoba,
    asalariados.con.ingreso.y.horas.cordoba$educacion=="universitaria completa" &
      asalariados.con.ingreso.y.horas.cordoba$CH03==1))+
  geom_point(aes(PP3E_TOT, PP08D1, col=sexo, size=edad))
```



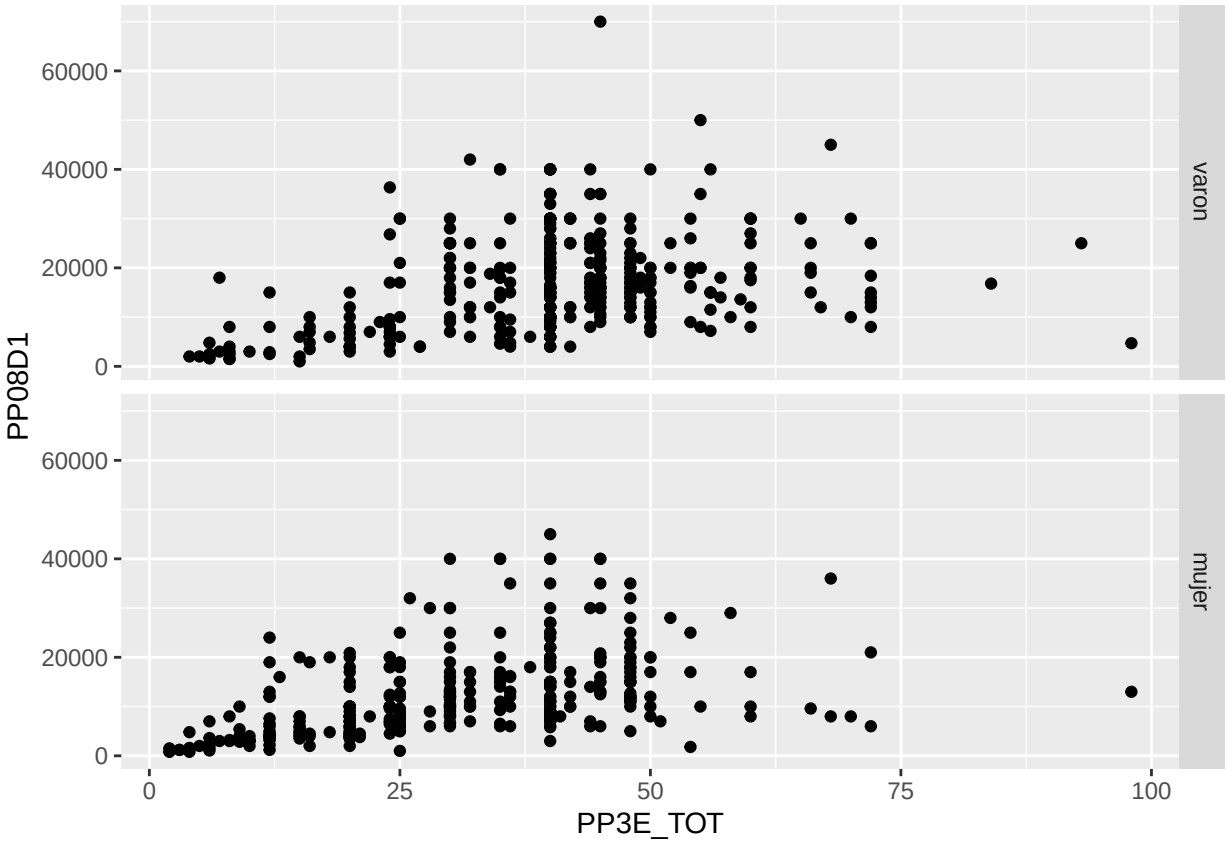
Otra opción para comparar grupos es la capa `facet_grid` que puede agregarse a cualquier tipo de gráfico. El comando tiene dos argumentos que corresponden a dos variables de clase `factor`, que se separan con ~ (alt+126 en windows) que van a "facetear" el gráfico pedido en tantas filas y columnas como categorías tengan esas dos variables. Puede usarse solo una variable, reemplazando con un punto la posición de la otra. Por ejemplo para hacer un diagrama de dispersión de los ingresos salariales según las horas trabajadas para varones y otro para mujeres uno al lado del otro:

```
ggplot(
  asalariados.con.ingreso.y.horas.cordoba)+
  geom_point(aes(PP3E_TOT, PP08D1))+
  facet_grid(.~sexo)
```



Para que estén uno encima del otro, la variable va en el lugar de las filas

```
ggplot(
  asalariados.con.ingreso.y.horas.cordoba)+
  geom_point(aes(PP3E_TOT, PP08D1))+
  facet_grid(sexo~.)
```

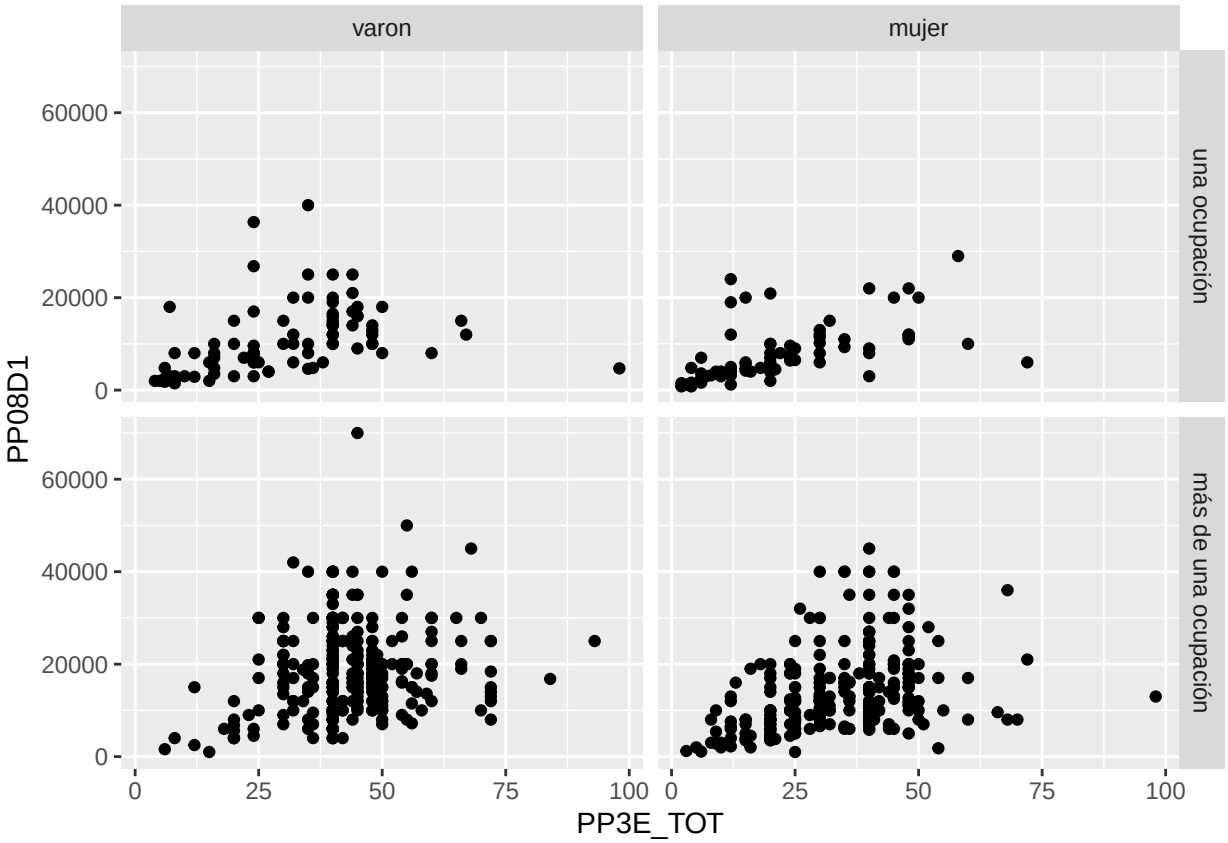


Otra variable a mapear en esta capa puede ser el hecho de tener una o más ocupaciones. Dado que no se ha usado antes esa variable, hay que ajustarla con un nombre y etiquetas.

```
asalariados.con.ingreso.y.horas.cordoba$cantidad.ocupaciones<-
  as.factor(asalariados.con.ingreso.y.horas.cordoba$PP03I)

levels(asalariados.con.ingreso.y.horas.cordoba$cantidad.ocupaciones)<-
  c("una ocupación", "más de una ocupación")
```

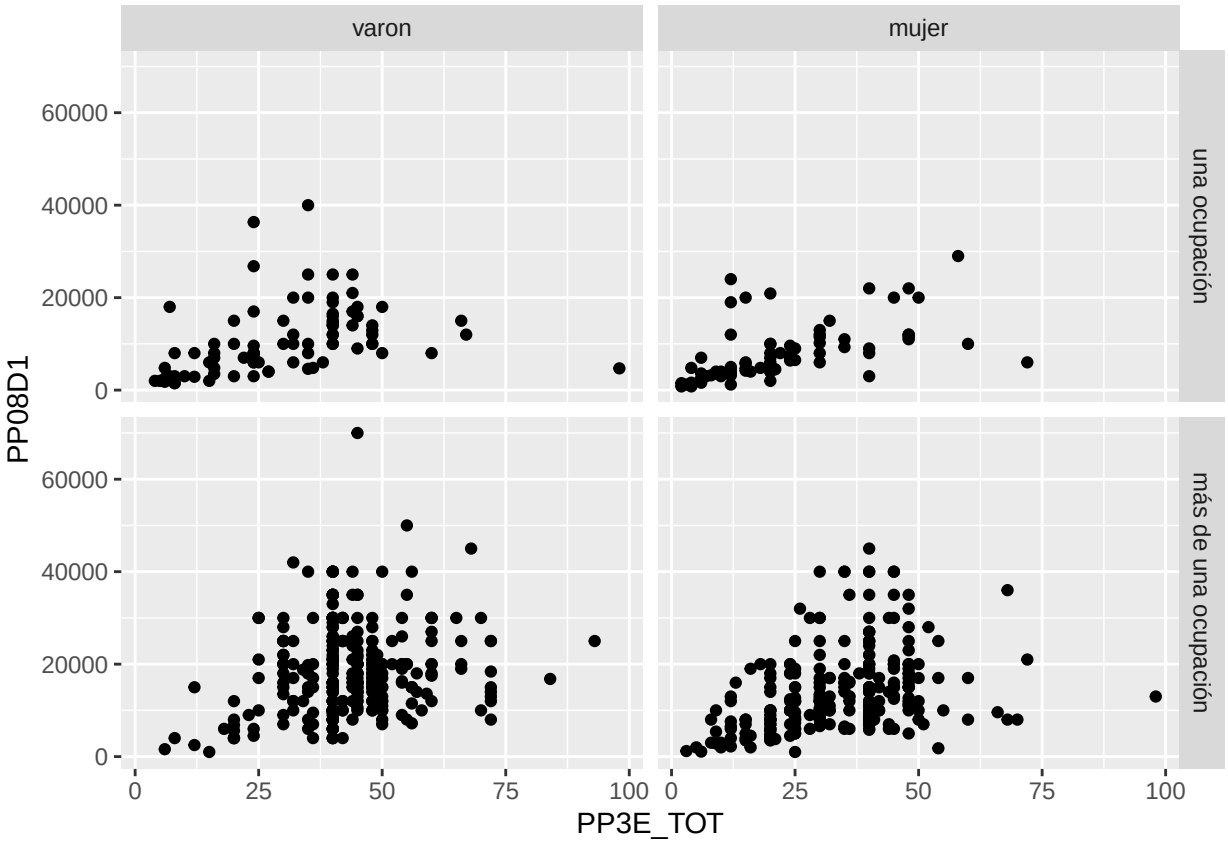
Ahora se puede separar el gráfico según las categorías de las dos variables:

```
ggplot(
  asalariados.con.ingreso.y.horas.cordoba)+
  geom_point(aes(PP3E_TOT, PP08D1))+
  facet_grid(cantidad.ocupaciones~sexo)
```



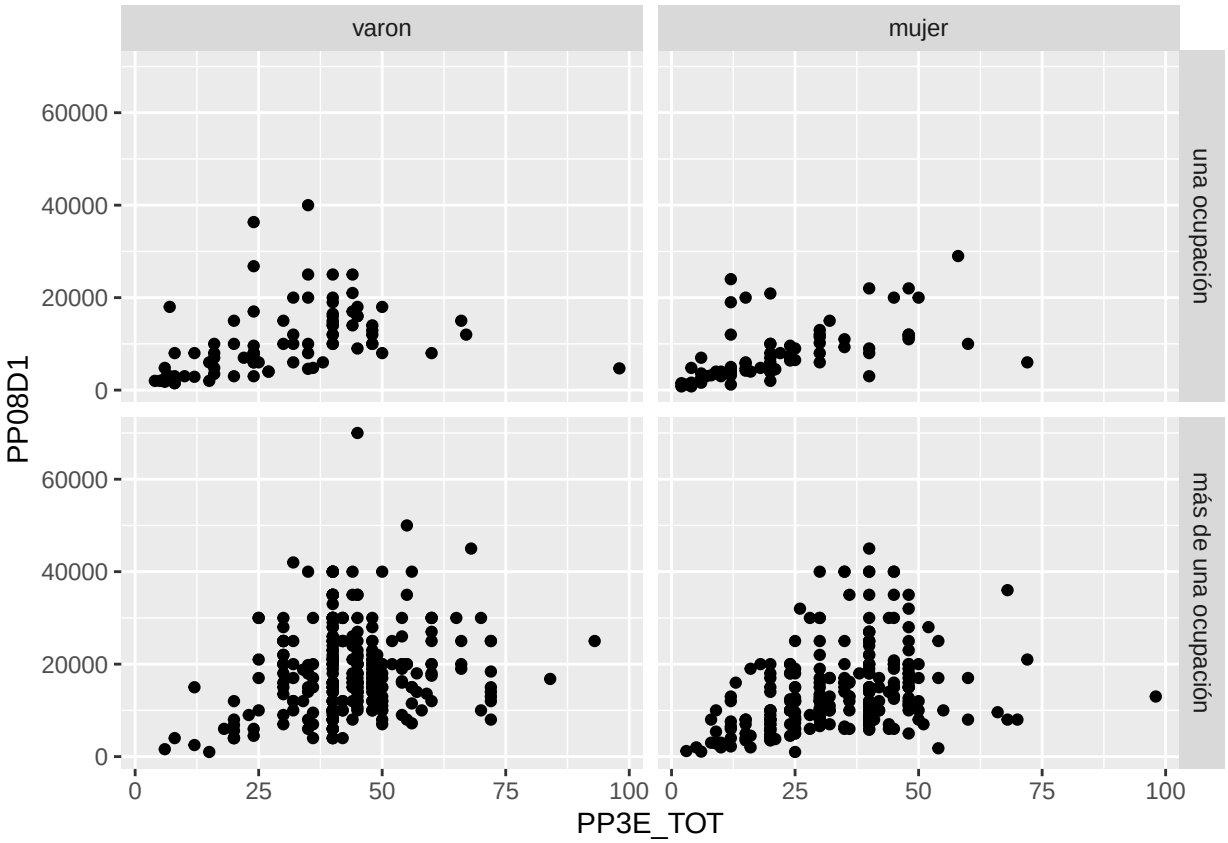
Para no repetir las instrucciones que generan las primeras capas del gráfico (aunque se lo haga copiando y pegando), vamos a guardarlo como un objeto y pedir que lo muestre, encerrando entre paréntesis la expresión:

```
(p1<-ggplot(
  asalariados.con.ingreso.y.horas.cordoba)+
  geom_point(aes(PP3E_TOT, PP08D1))+
  facet_grid(cantidad.ocupaciones~sexo))
```



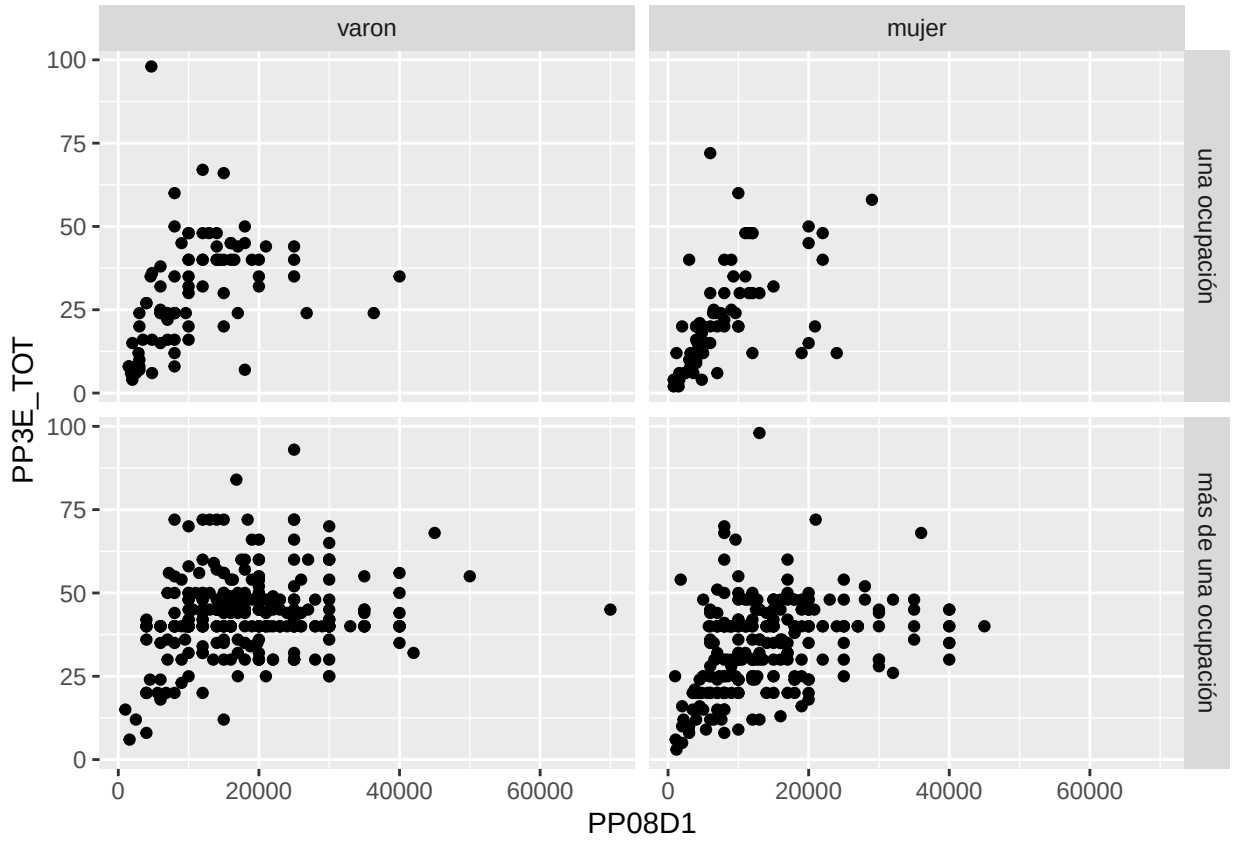
Cuando se lo necesite nuevamente, solo se lo llama:

```
p1
```



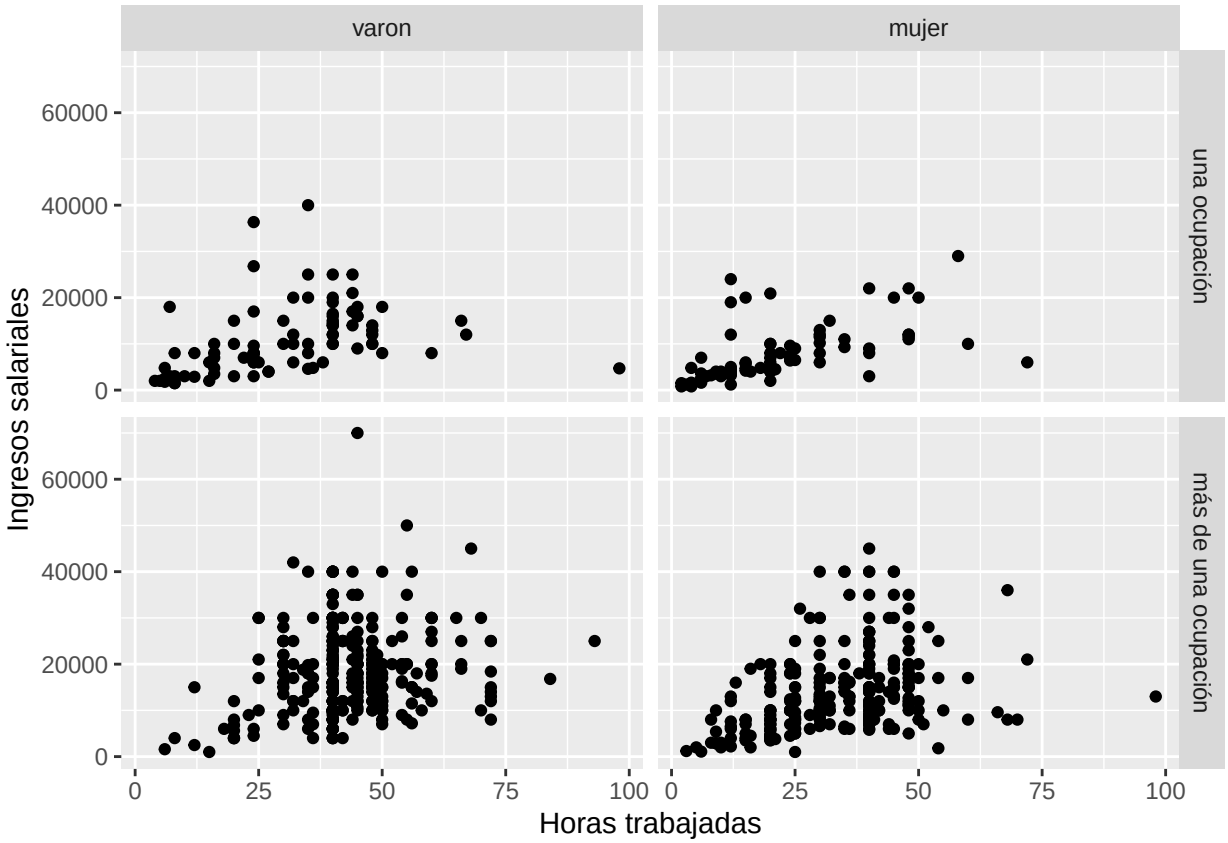
Y se pueden seguir agregando capas a p1. El cambio en la posición de los ejes es `coord_flip`

```
p1+coord_flip()
```



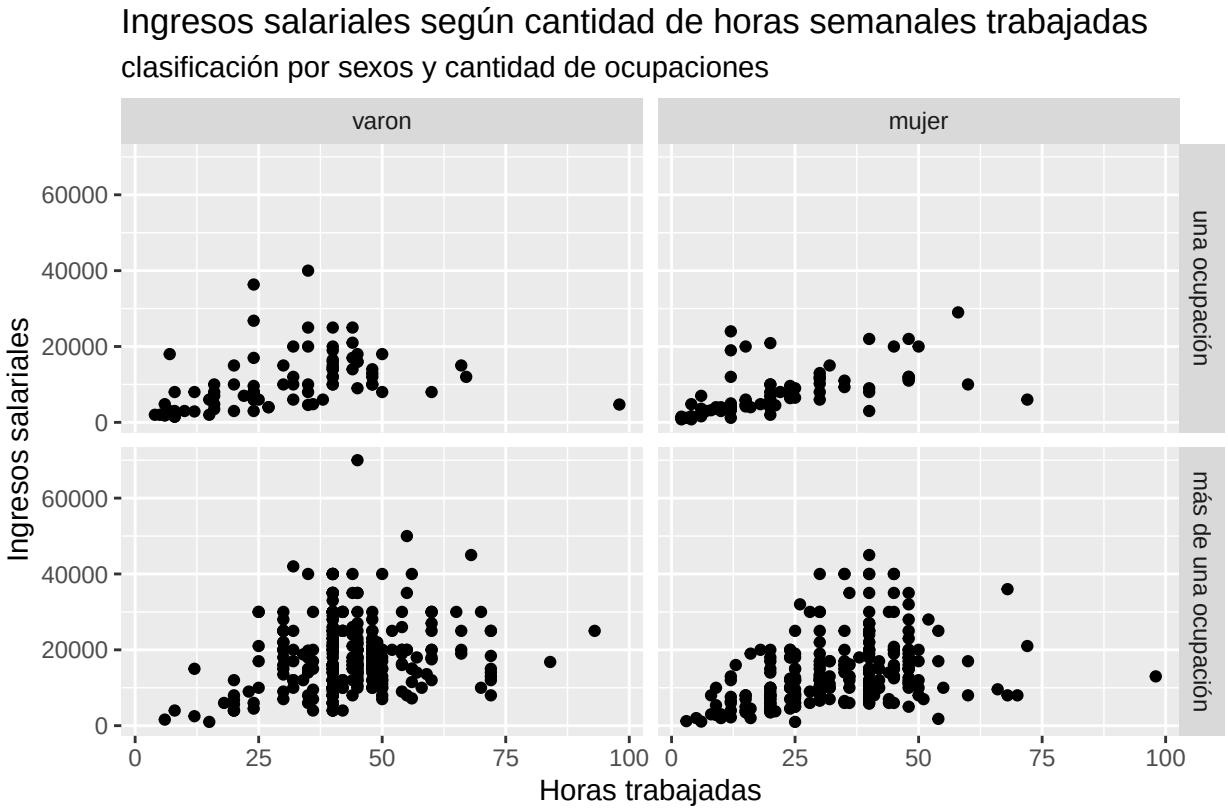
Los nombres de los ejes se ajustan con capas de etiquetas:

```
p1+ xlab("Horas trabajadas")+
  ylab("Ingresos salariales")
```



La capa `labs` agrega un título, subtítulo y epígrafe.

```
p1+ xlab("Horas trabajadas")+
  ylab("Ingresos salariales")+
  labs(title="Ingresos salariales según cantidad de horas semanales trabajadas",
       subtitle="clasificación por sexos y cantidad de ocupaciones",
       caption="Fuente: EPH tercer trimestre 2018")
```



La tipografía y combinación de colores puede elegirse de manera muy precisa con la capa `theme`, sin embargo, el paquete `ggthemes` tiene preformateados algunos como los que usan algunos medios (Wall Street Journal, The economist) o algunos programas muy conocidos (excel, stata). Para simplificar, llamemos p2 al gráfico que tenemos construido hasta ahora, al que vamos a pedir que corte el título en dos renglones, eso se indica con una barra inclinada (alt+92 en windows) y una letra n en el punto donde se quiere bajar de renglón:

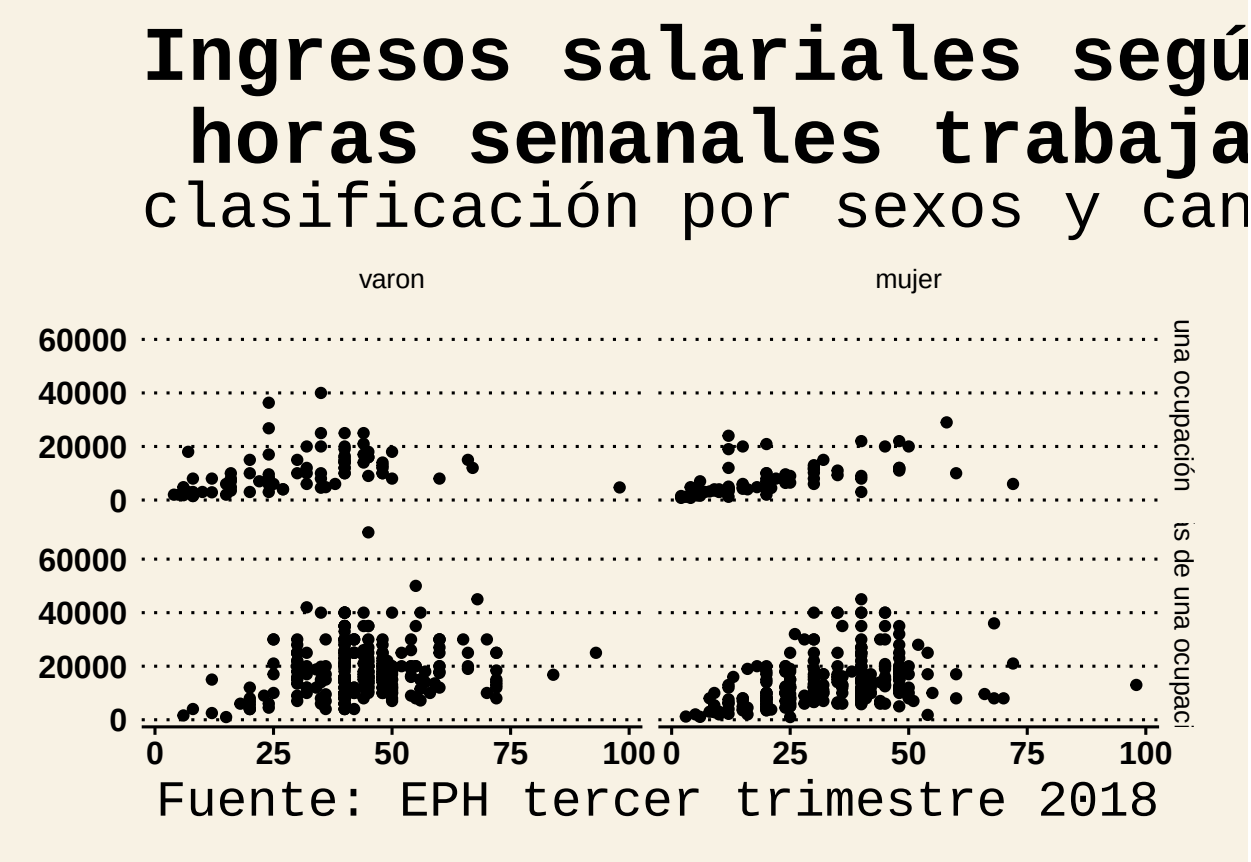
```
p2<-p1+ xlab("Horas trabajadas")+
  ylab("Ingresos salariales")+
  labs(
    title="Ingresos salariales según cantidad de \n horas semanales trabajadas",
    # \n indica que baje al otro renglón
       subtitle="clasificación por sexos y cantidad de ocupaciones",
       caption="Fuente: EPH tercer trimestre 2018")
```

Y probemos alguna capas del paquete `ggthemes`, para lo que tenemos que cargarlo en la sesión

```
library(ggthemes)
```

Wall Street Journal

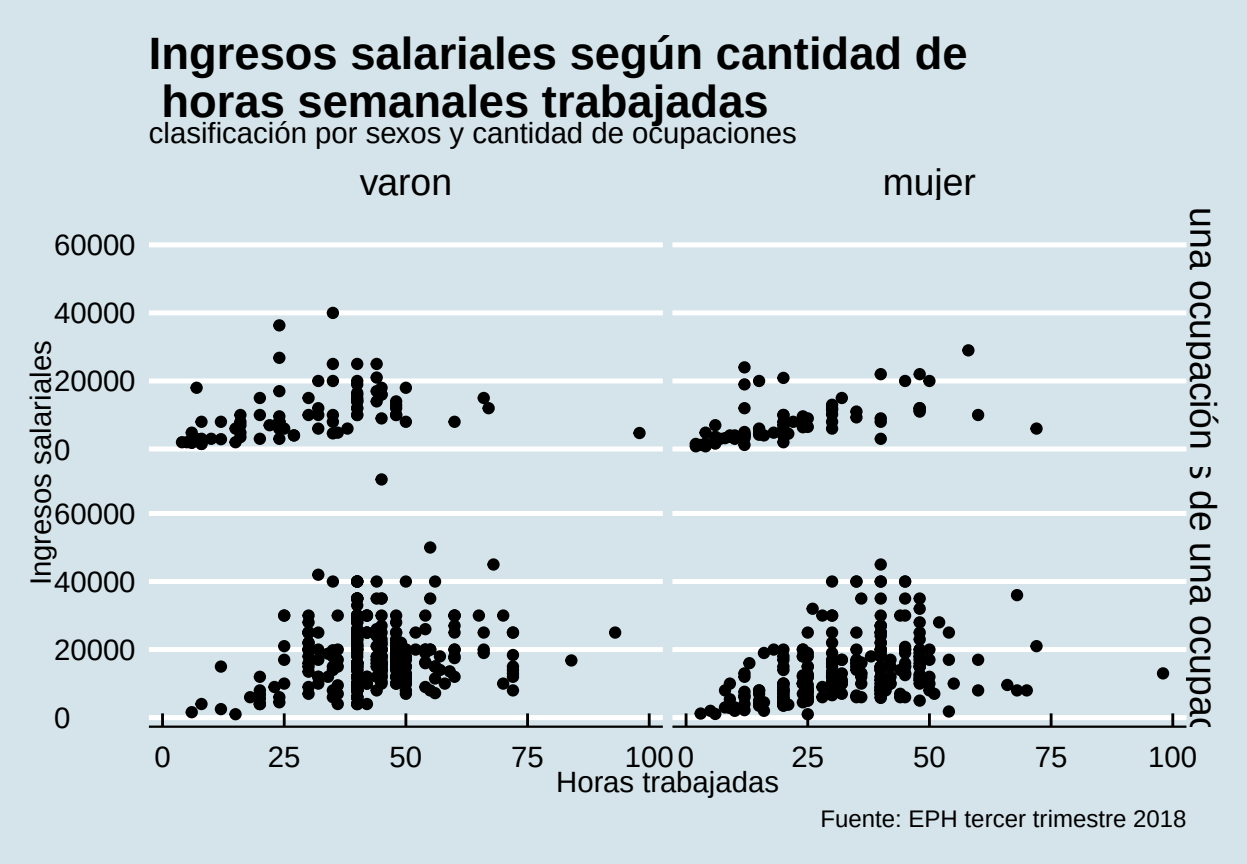
```
p2+theme_wsj()
```



La escala de este tema no es adecuada para nuestro visualizador de gráficos, pero si lo expande, puede verse completo.

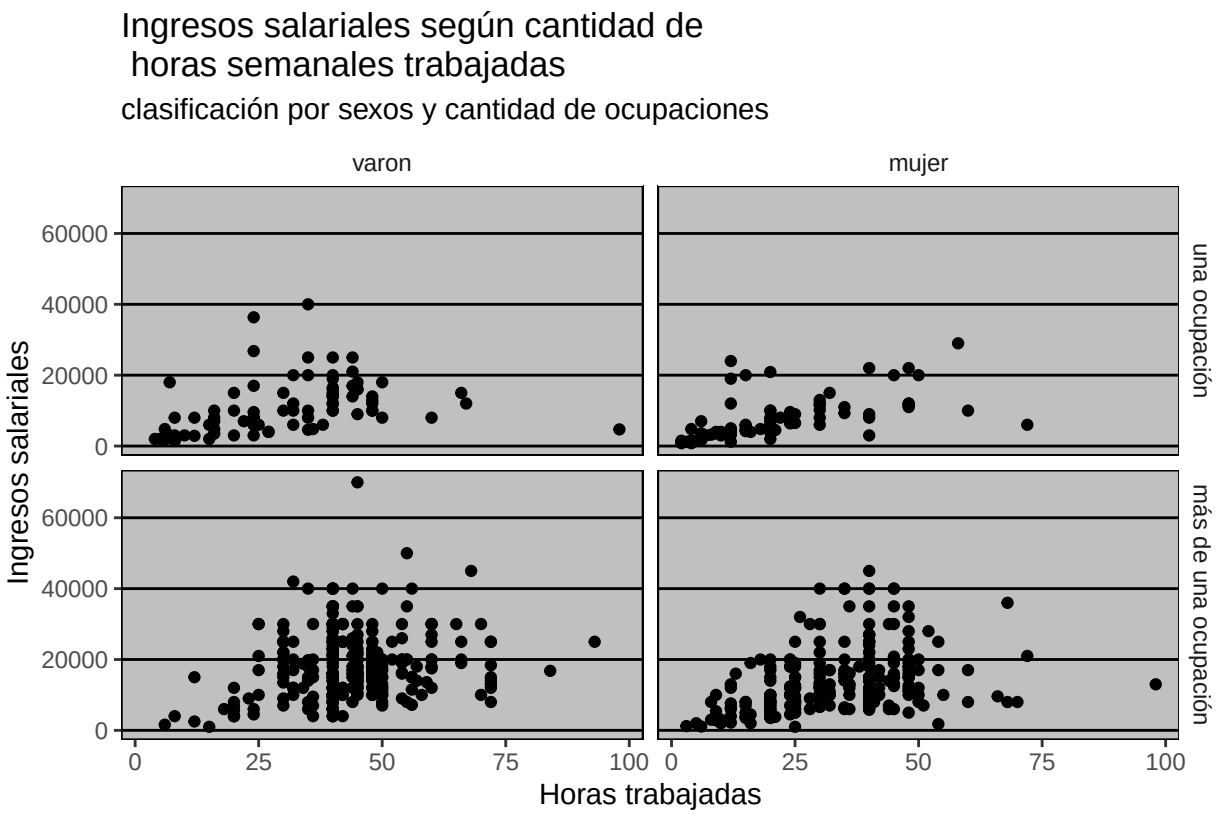
The Economist

```
p2+theme_economist()
```



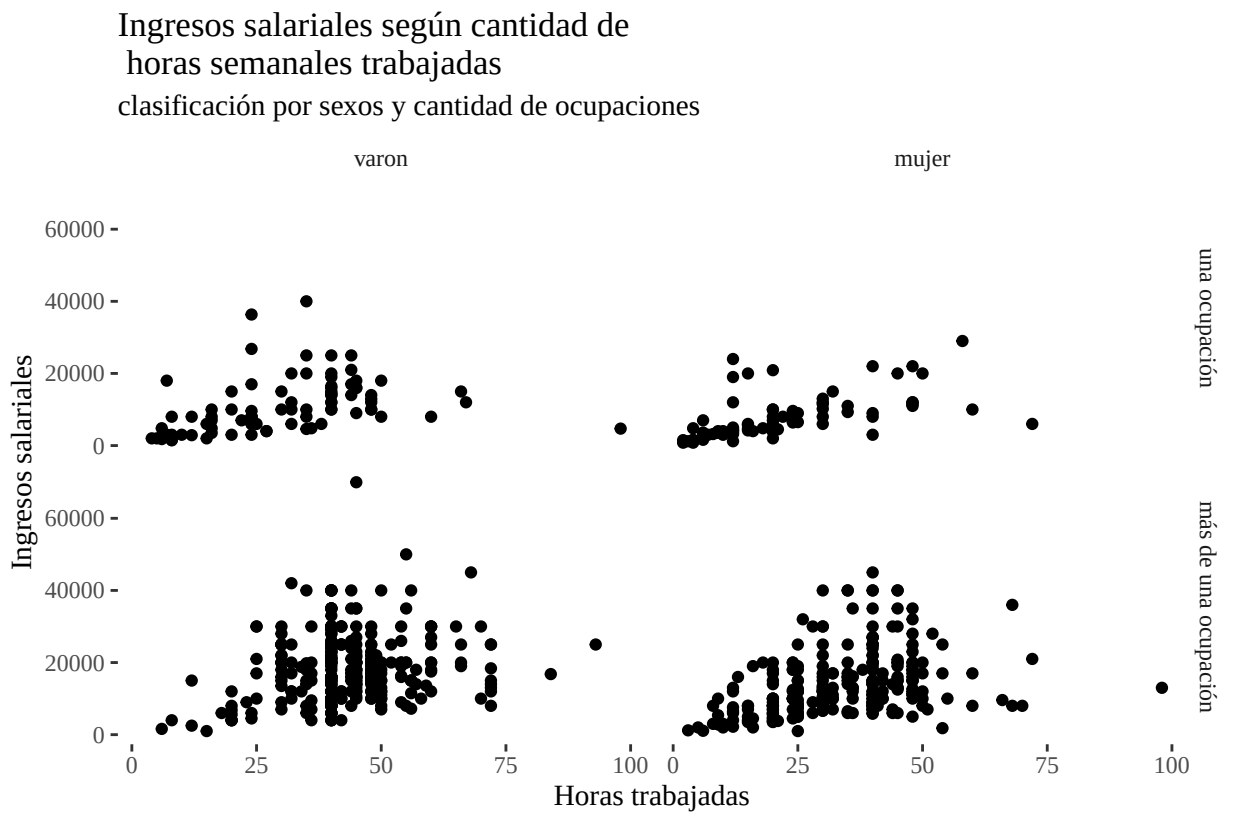
Excel

```
p2+theme_excel()
```



La propuesta de Edward Tufte

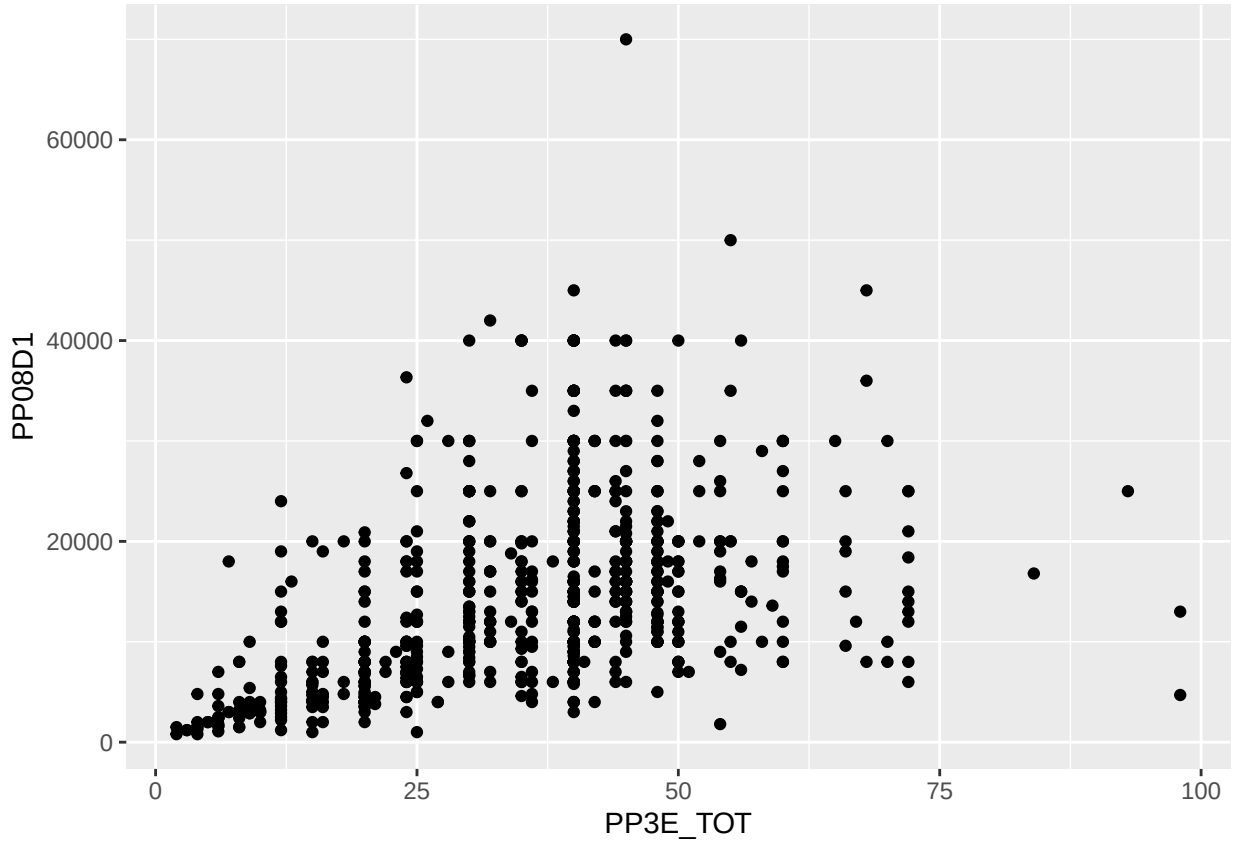
```
p2+theme_tufte()
```



Fuente: EPH tercer trimestre 2018

A los diagramas de dispersión puede agregarse una línea de tendencia, que ayuda a identificar patrones que no se ven en una nube de puntos muy densa: esto se logra con una capa que se llama `geom_smooth`. Volvemos al gráfico original, sin separar por sexos ni cantidad de ocupaciones.

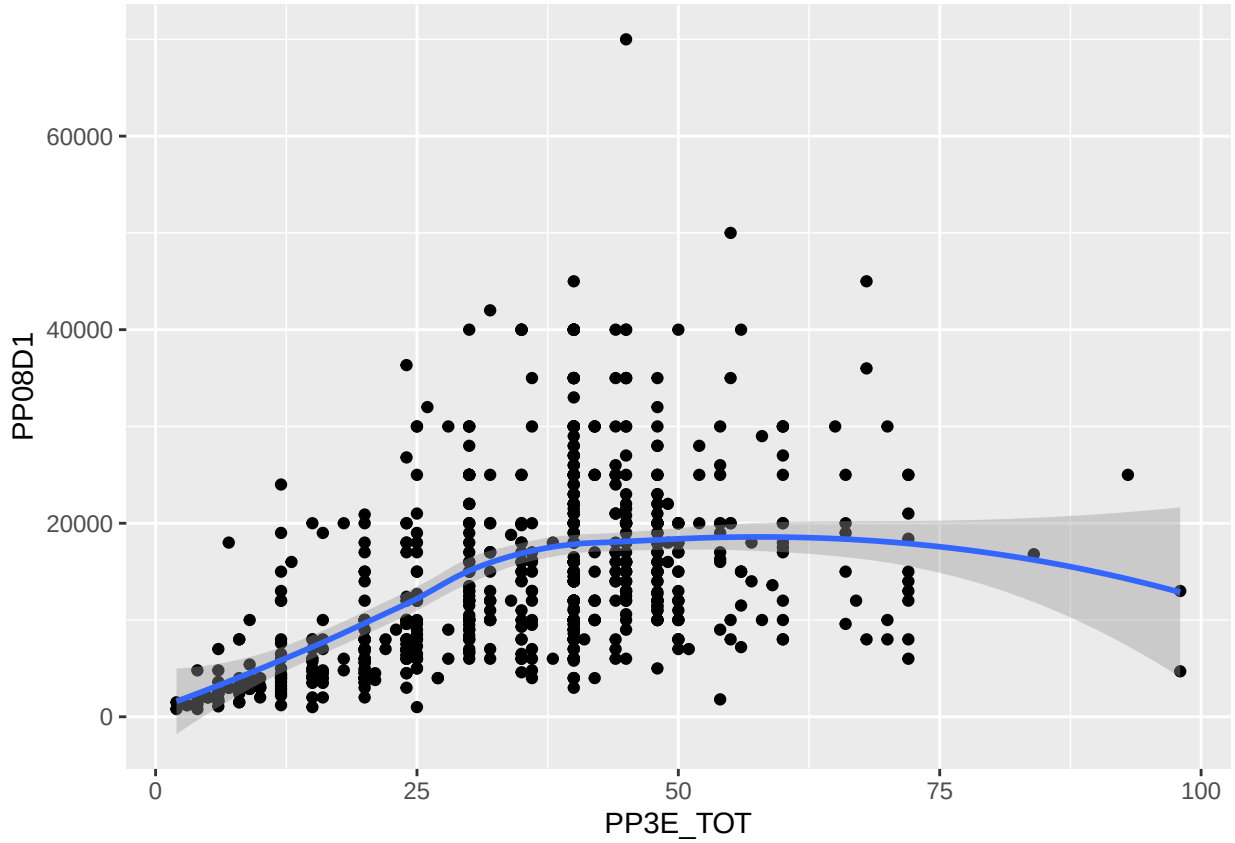
```
ggplot(
  asalariados.con.ingreso.y.horas.cordoba)+
  geom_point(aes(PP3E_TOT, PP08D1))
```



Para poder sumar a este gráfico una capa con la función que ajusta los puntos, necesitamos indicarle a ese coomando cuáles son las variables; esto puede hacerse repitiendo la estética (`aes()`) del `geom_point` o también incorporándola una sola vez en `ggplot()`, hacemos esto último:

```
ggplot(
  asalariados.con.ingreso.y.horas.cordoba, aes(PP3E_TOT, PP08D1))+
  geom_point()+geom_smooth()
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

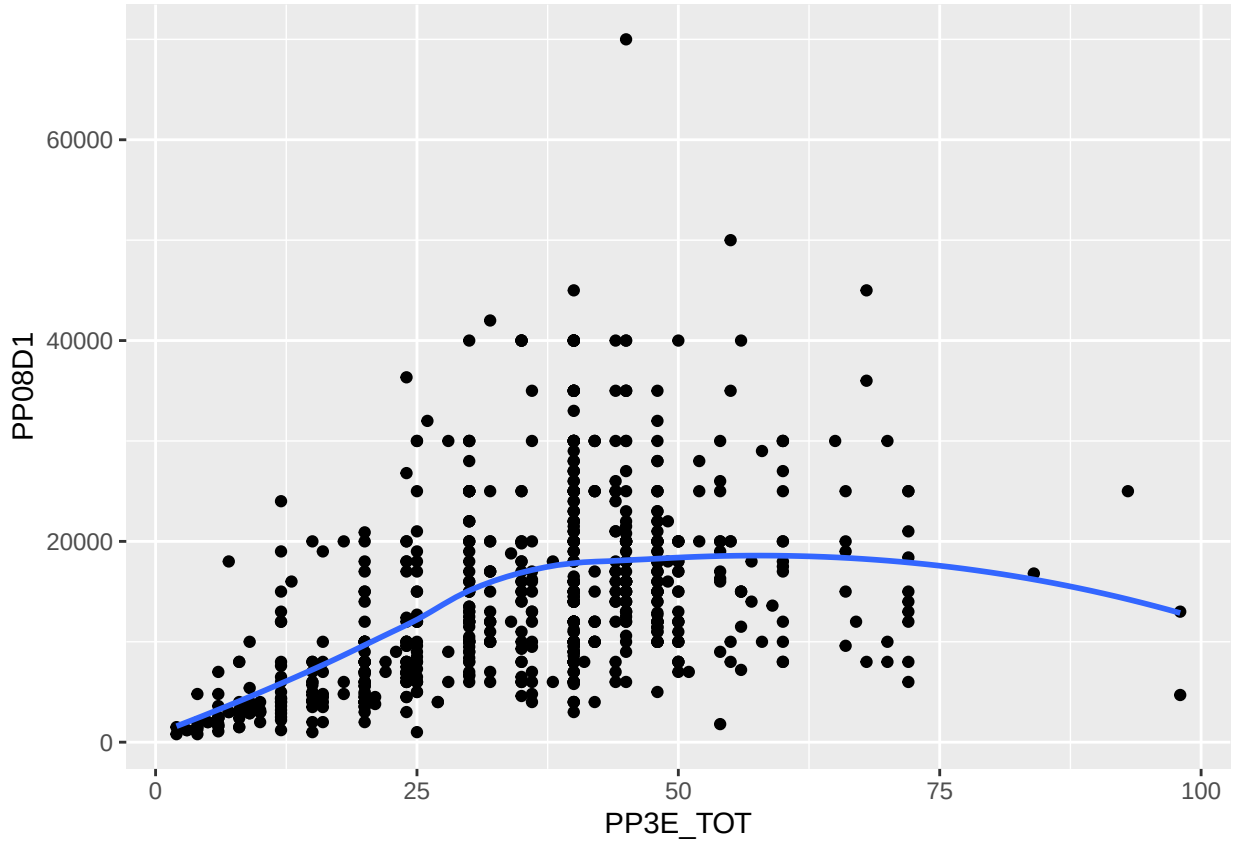


Por defecto, el método que usa `geom_smooth` para buscar la función, se llama *loess*, un método no paramétrico que hace regresiones por mínimos cuadrados en subconjuntos de puntos. Además, también por defecto, genera un intervalo de amplitud igual al error estándar (local) alrededor de la curva. Ambas opciones pueden personalizarse:

- Para quitar el intervalo, simplemente pedimos, en el argumento de `geom_smooth` que no use el error estándar:

```
ggplot(
  asalariados.con.ingreso.y.horas.cordoba, aes(PP3E_TOT, PP08D1))+
  geom_point()+geom_smooth(se=FALSE)
```

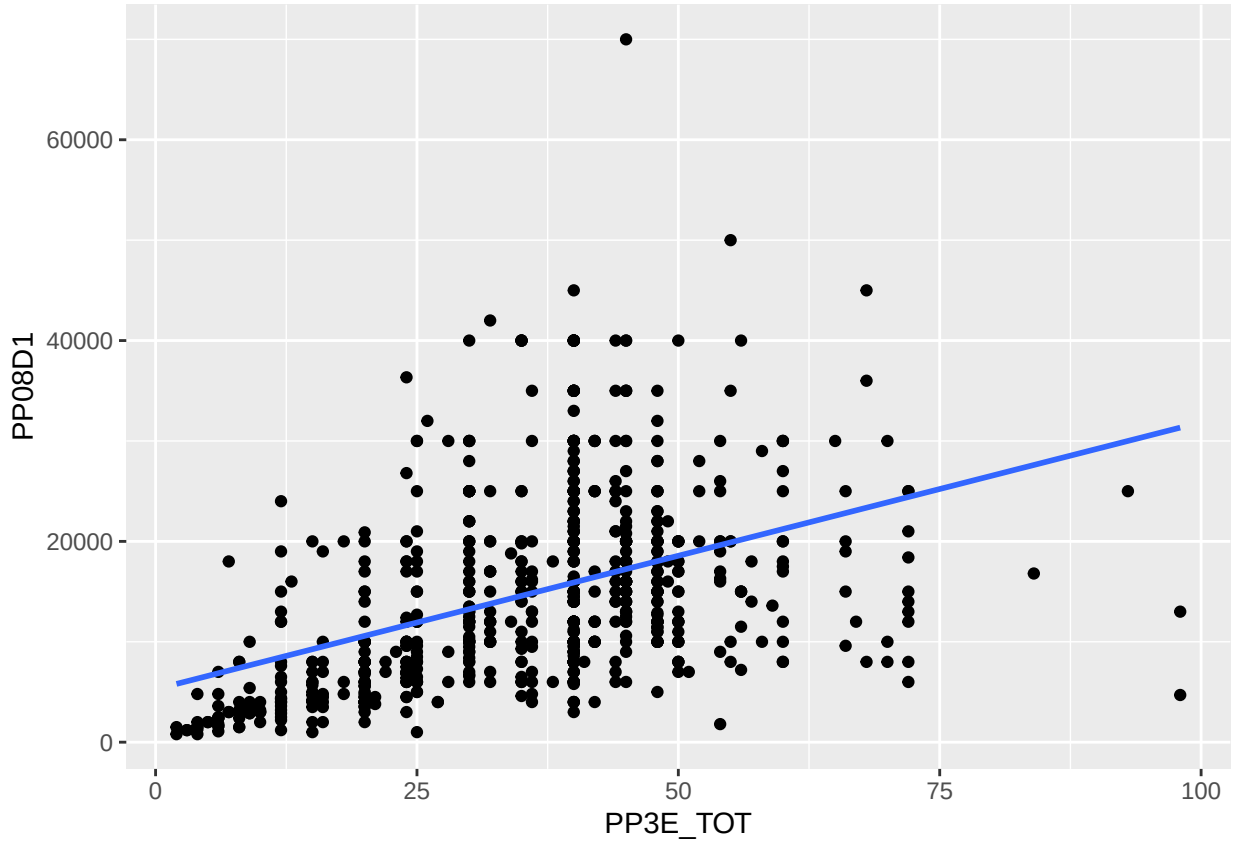
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



- Para cambiar el método de ajuste, hay que especificar uno diferente del que trae por defecto, por ejemplo si se decide ajustar un modelo lineal, su nombre es `lm`:

```
ggplot(
  asalariados.con.ingreso.y.horas.cordoba, aes(PP3E_TOT, PP08D1))+
  geom_point()+geom_smooth(se=FALSE, method = "lm")
```

```
## `geom_smooth()` using formula 'y ~ x'
```



Un método útil cuando hay valores atípicos es el estimador no paramétrico de Theil-Sen, que calcula la mediana de todas las pendientes entre cada par de puntos. Cuando no hay casos atípicos el resultado es muy similar al de míminos cuadrados. El inconveniente es que ese método no está (aun) entre las opciones de `geom_smooth`, por lo que hay que crear una función y luego pedir que la use.
El cálculo del estimador de Theil-Sen está en el paquete `mblm`, pero no puede usarse directamente porque no usa ponderadores (que sí los pide la función que está como argumento del método en `geom_smooth`). Entonces en la función que se crea, se indica un valor nulo para los ponderadores y así (aunque sea nulo) `geom_smooth` lo acepta:

Cargamos el paquete

```
library(mblm)
```

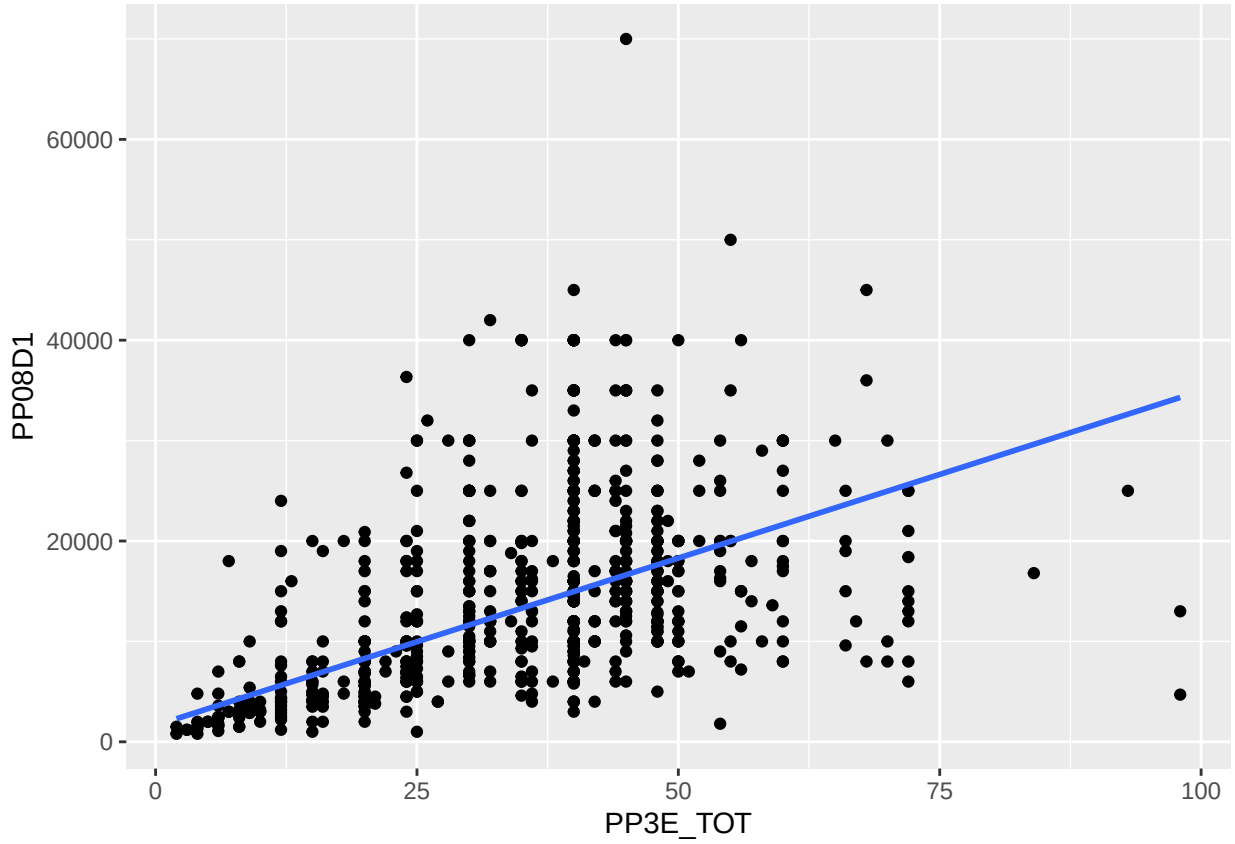
Y construimos la función, indicando que el argumento es el mismo (...) y que en los poderadores ponga "nada":

```
estimador_theil_sen <- function(..., weights = NULL) {
  mblm::mblm(...)
}
```

Ahora la podemos pedir como método para que la use `geom_smooth`:

```
ggplot(
  asalariados.con.ingreso.y.horas.cordoba, aes(PP3E_TOT, PP08D1))+
  geom_point()+geom_smooth(se=FALSE, method = "estimador_theil_sen")
```

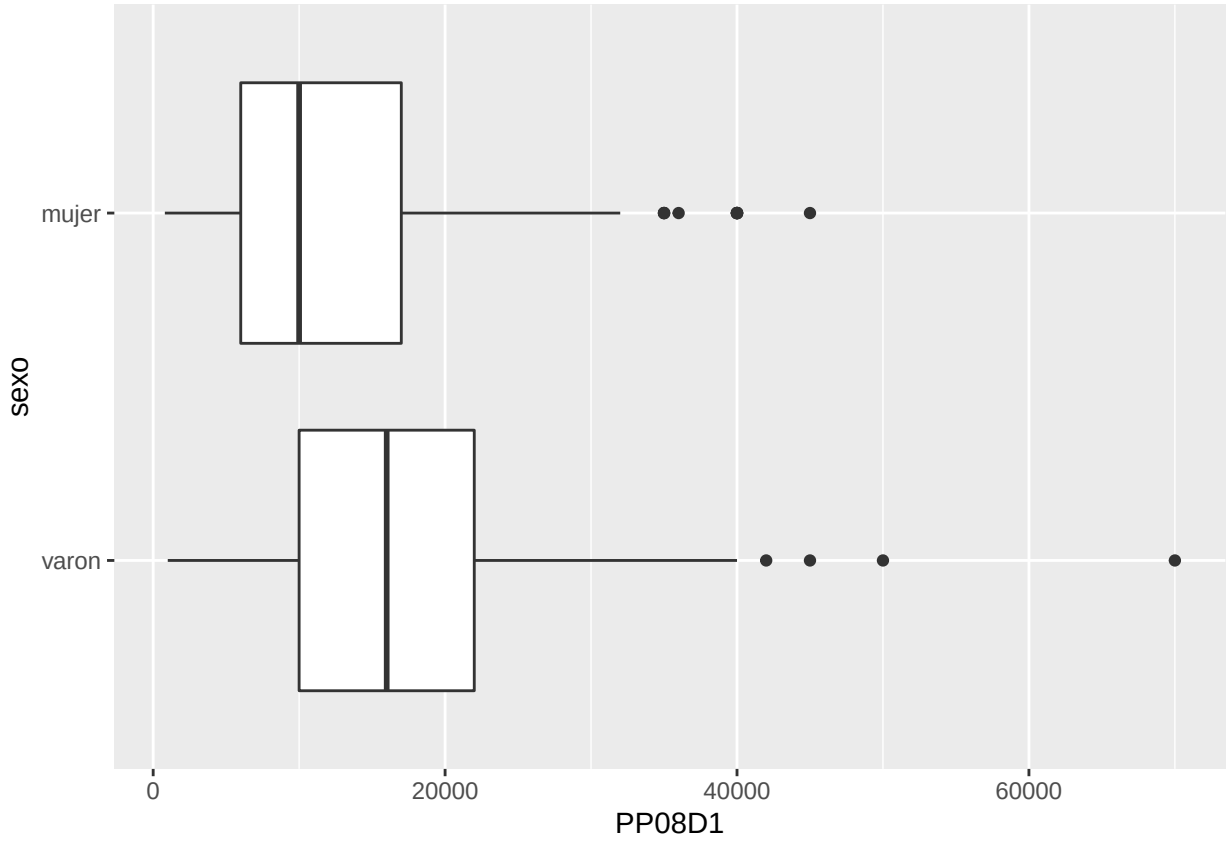
```
## `geom_smooth()` using formula 'y ~ x'
```



La recta es bastante parecida, pero está menos afectada por los casos atípicos.

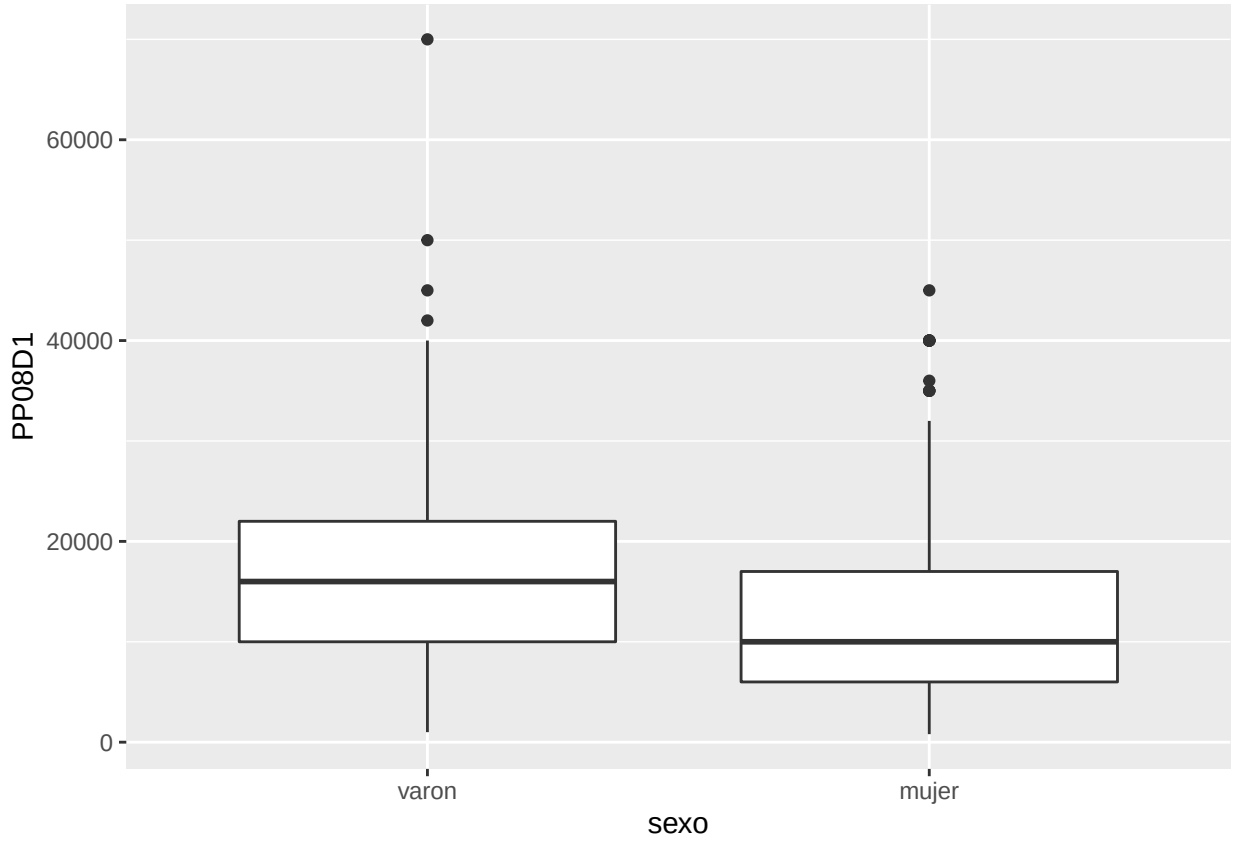
La relación entre una variable cuantitativa y una categórica u ordinal, se ve bien con los box-plots, por ejemplo para observar la distribución de los ingresos salariales entre varones y mujeres:

```
ggplot(
  asalariados.con.ingreso.y.horas.cordoba, aes(
    PP08D1, sexo))+
  geom_boxplot()
```



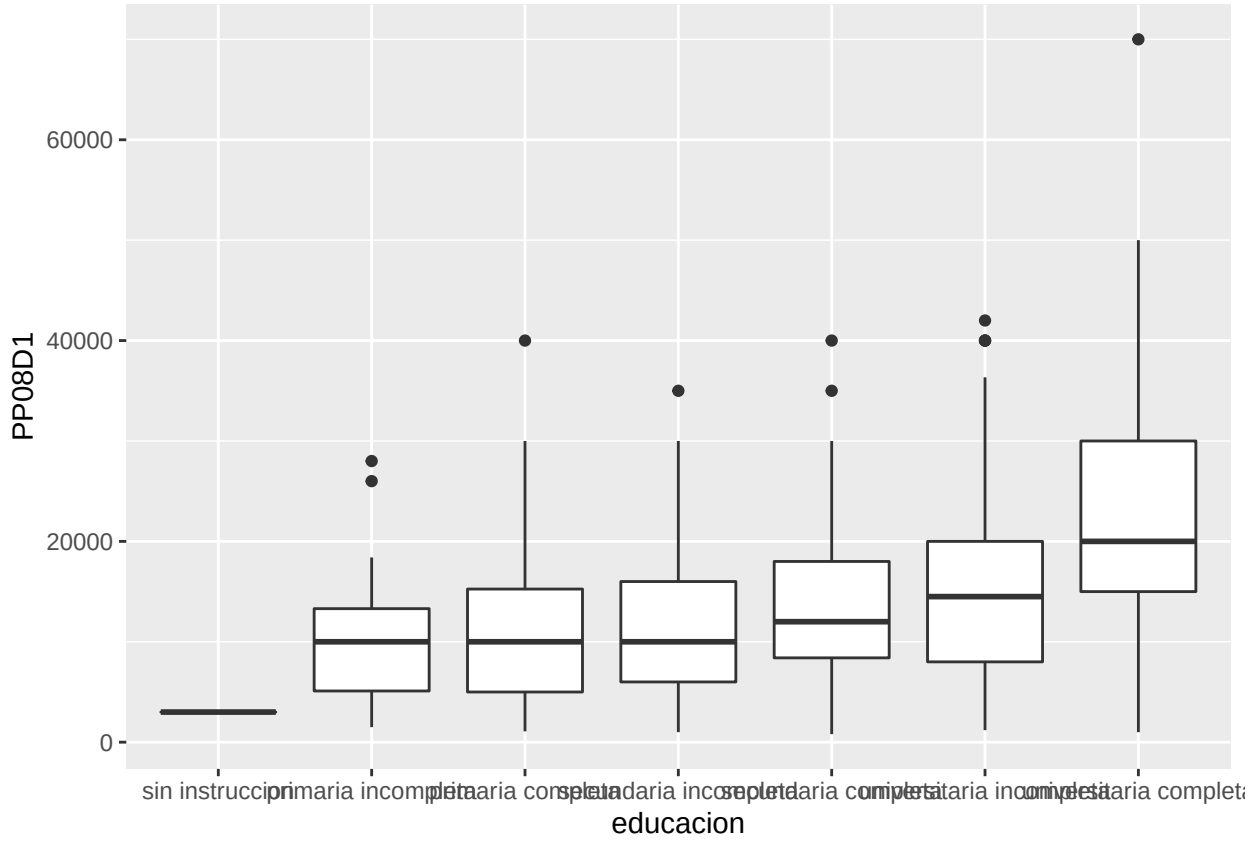
Aquí hemos ubicado las variables en la estética (`aes()`) del comando `ggplot()`, se podría haber puesto dentro de `box_plot()` pero, si se agrega otra capa habría sido necesario repetir las variables en su estética. Este comando detecta automáticamente cuál es la variable numérica. Si se invierte el orden, se obtiene:

```
ggplot(
  asalariados.con.ingreso.y.horas.cordoba, aes(
    sexo, PP08D1))+
  geom_boxplot()
```



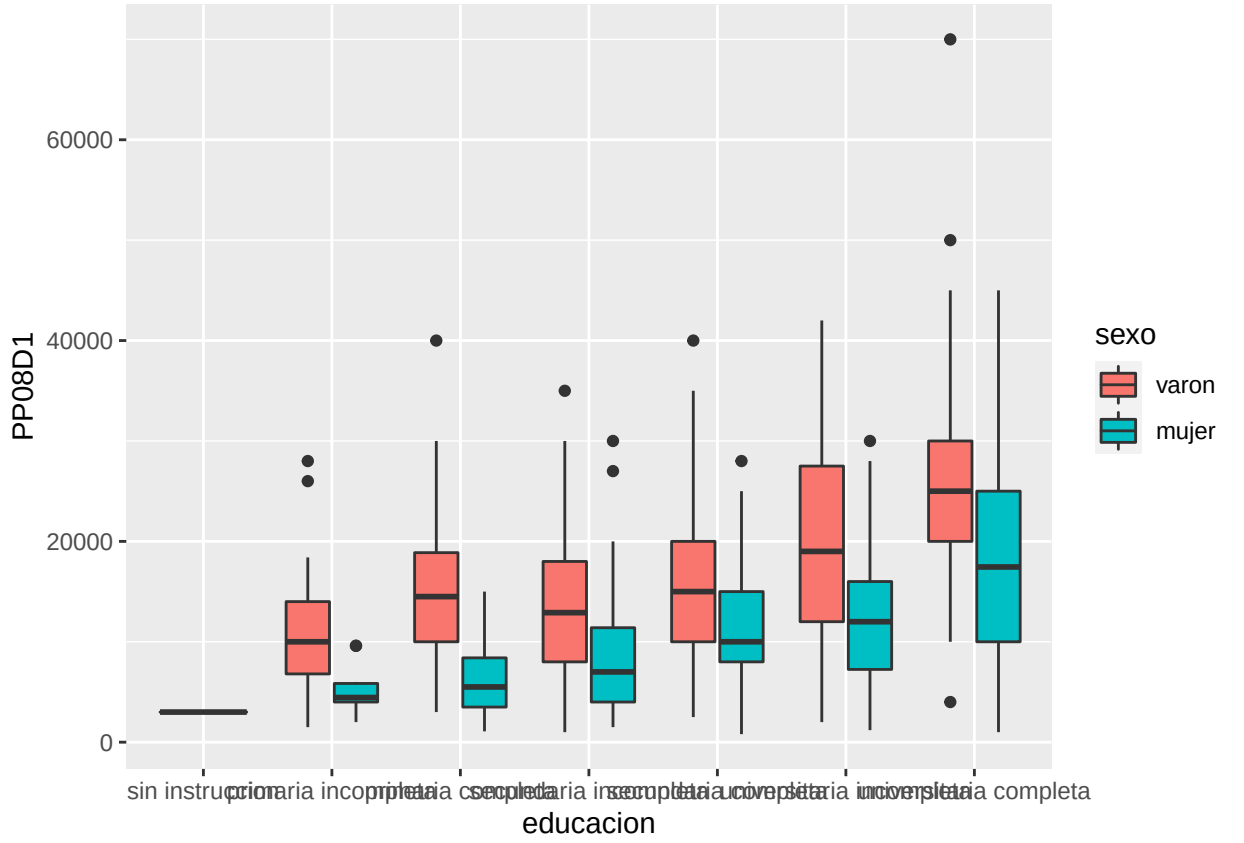
Cuando se usa una variable ordinal, como la educación:

```
ggplot(
  asalariados.con.ingreso.y.horas.cordoba, aes(
    educacion, PP08D1))+
  geom_boxplot()
```



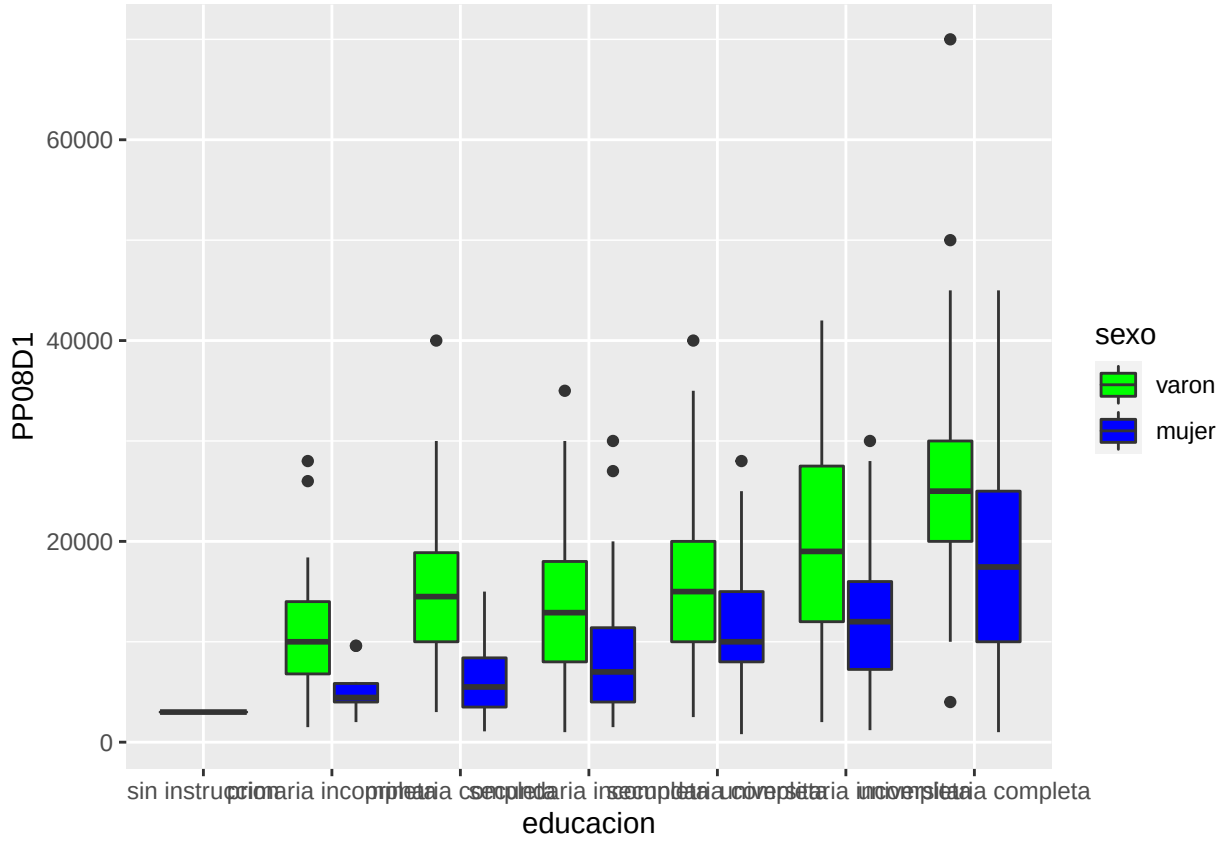
Puede agregarse otra variable, mapeada al color:

```
ggplot(
  asalariados.con.ingreso.y.horas.cordoba, aes(
    educacion, PP08D1))+
  geom_boxplot(aes(fill=sexo))
```



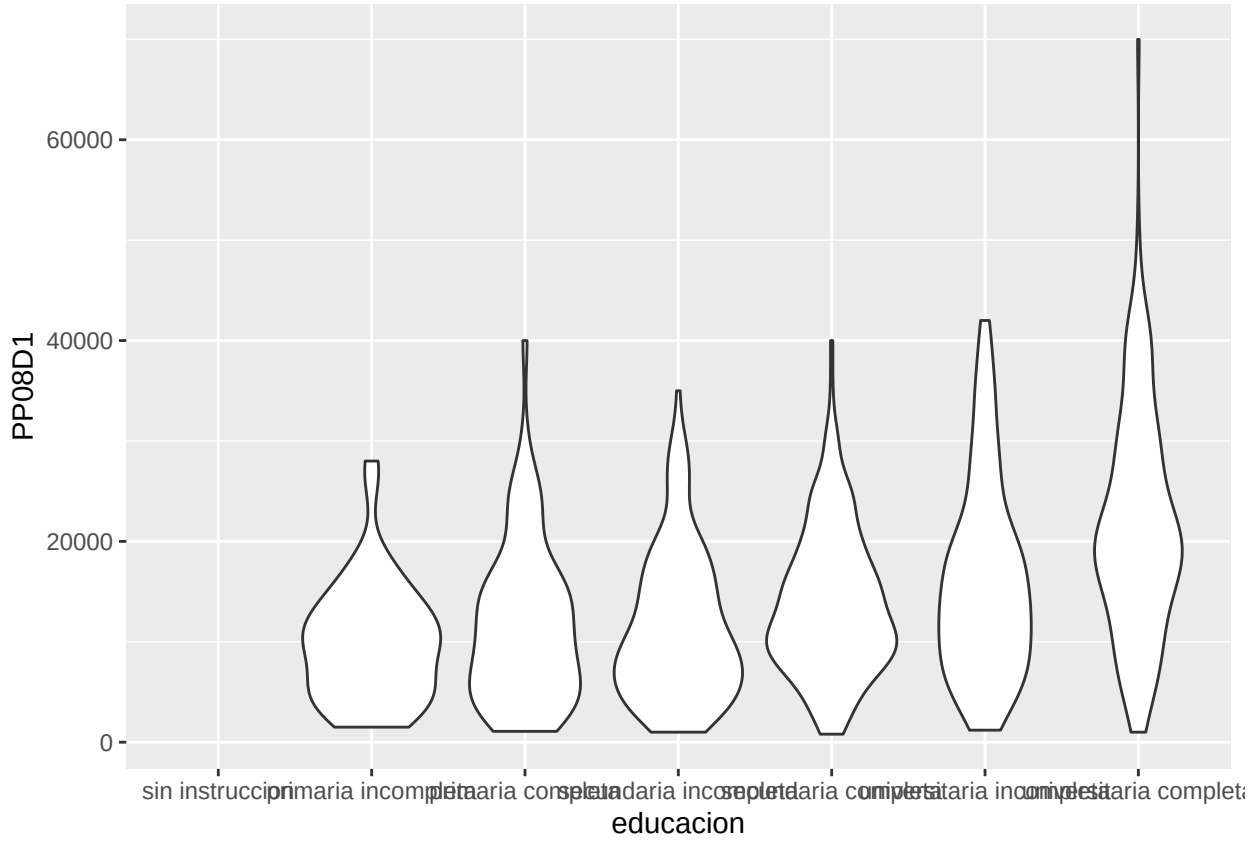
Así, para cada nivel de educación grafica varones y mujeres de diferente color. Como antes, los colores pueden ajustarse manualmente:

```
ggplot(
  asalariados.con.ingreso.y.horas.cordoba, aes(
    educacion, PP08D1))+
  geom_boxplot(aes(fill=sexo))+
  scale_fill_manual(values=c('varon'="green", 'mujer'="blue"))
```



La distribución al interior de cada grupo queda bien expresada visualmente con el gráfico de violin:

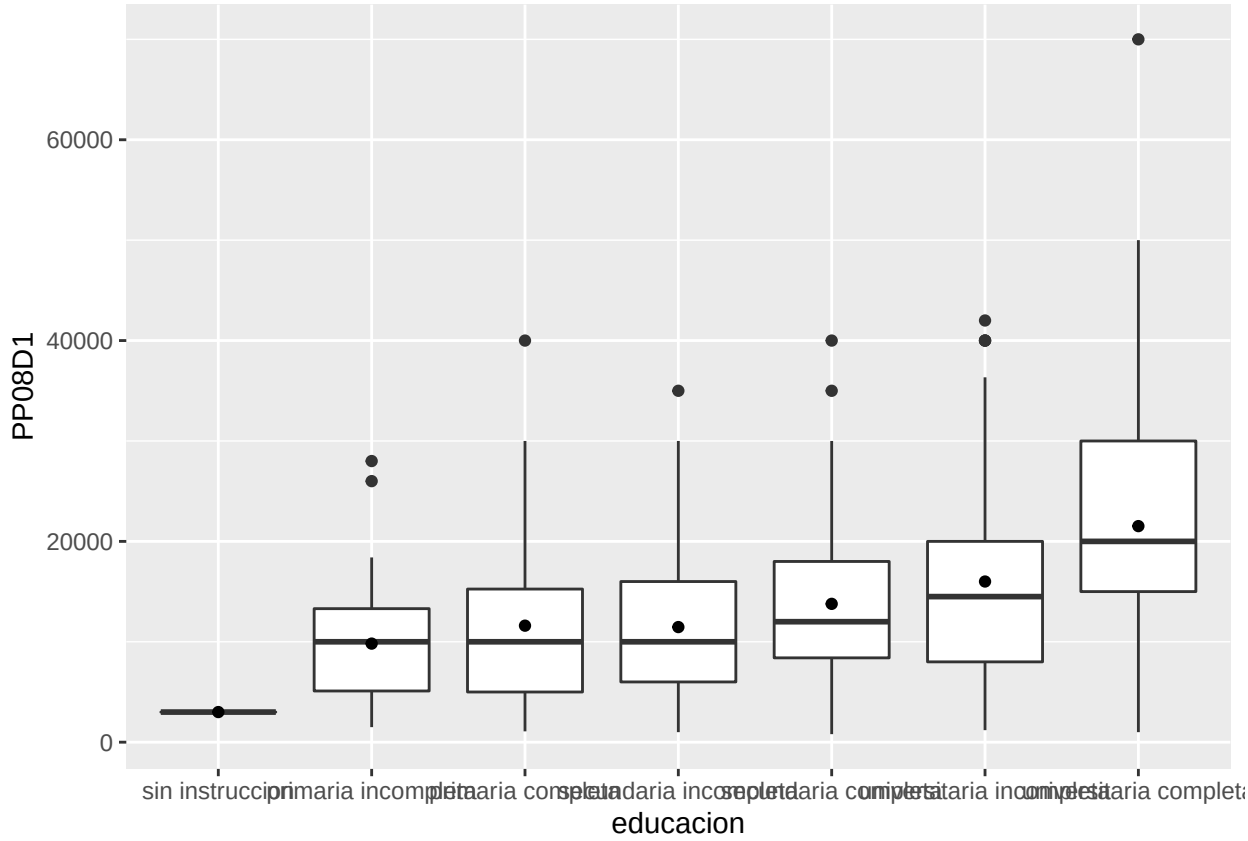
```
ggplot(
  asalariados.con.ingreso.y.horas.cordoba, aes(
    educacion, PP08D1))+geom_violin()
```



Tanto al gráfico de violín como al box-plot se le pueden agregar puntos en la media o la mediana de cada grupo:

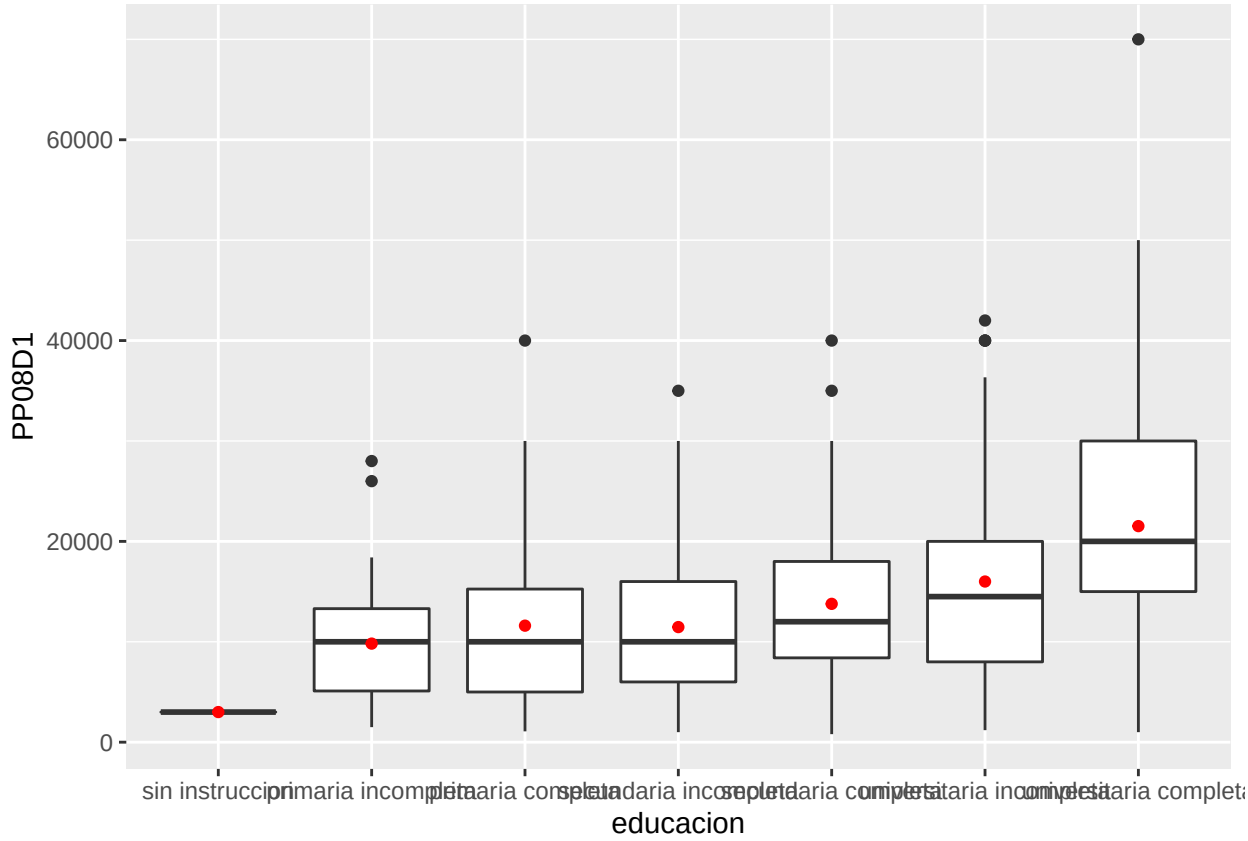
El box-plot con las medias:

```
ggplot(
  asalariados.con.ingreso.y.horas.cordoba, aes(
    educacion, PP08D1))+
  geom_boxplot()+
  stat_summary(fun.y=mean, geom="point")
```



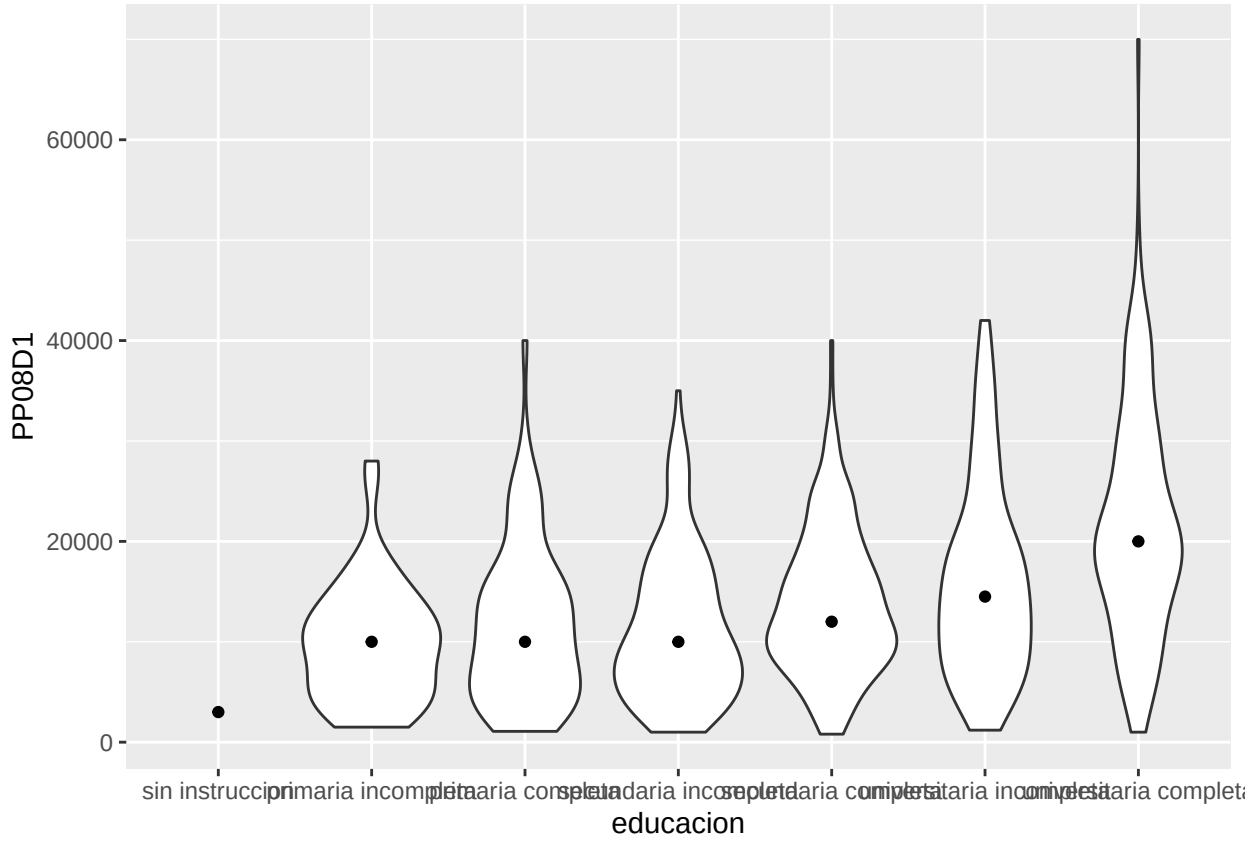
Pintados de rojo

```
ggplot(
  asalariados.con.ingreso.y.horas.cordoba, aes(
    educacion, PP08D1))+
  geom_boxplot()+
  stat_summary(fun.y=mean, geom="point", col="red")
```



El violín con las medianas

```
ggplot(
  asalariados.con.ingreso.y.horas.cordoba, aes(
    educacion, PP08D1))+
  geom_violin()+stat_summary(fun.y=median, geom="point")
```

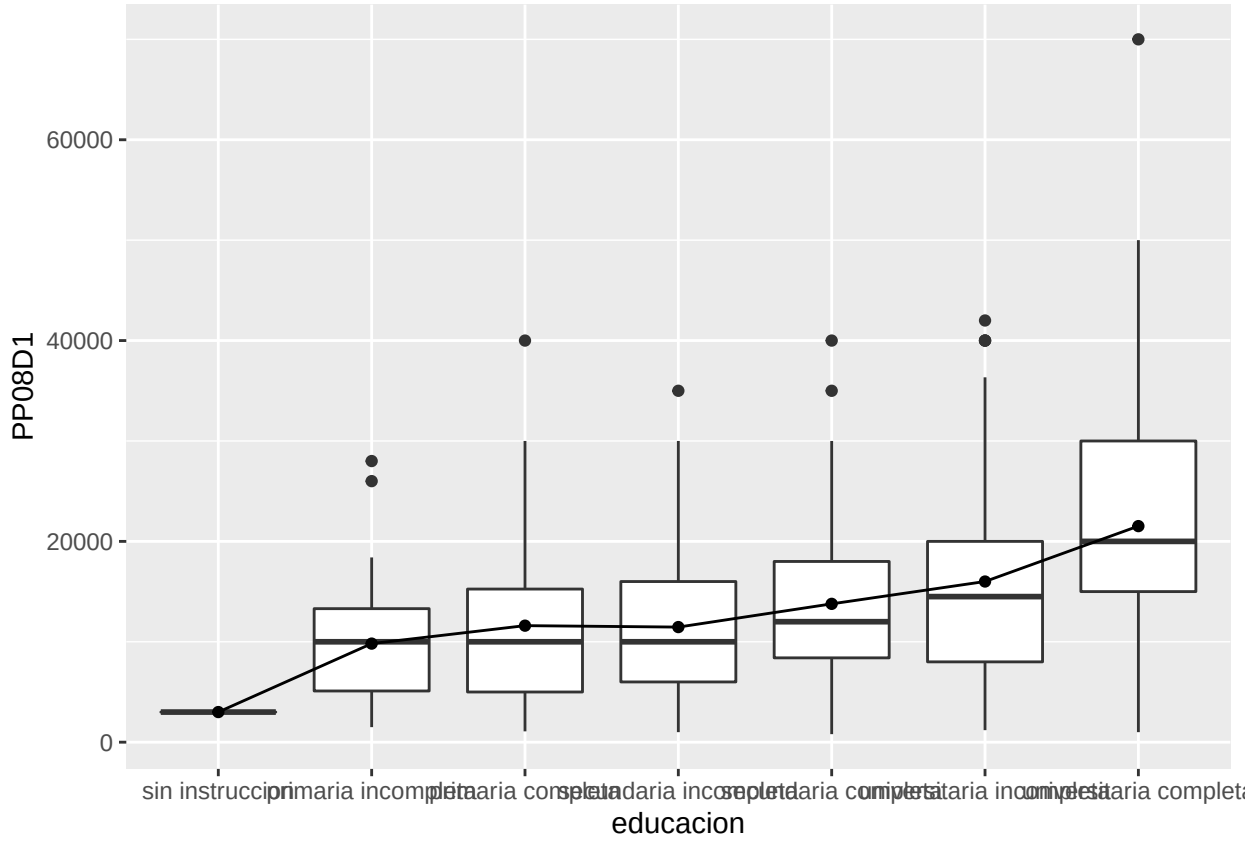


En ambos casos hemos el procedimiento ha sido agregar una capa de resumen estadístico, a la que indicamos qué función queremos para la variable y (media o mediana) y se pide que los elementos geométricos que las expresen sean puntos.

Estos gráficos sugieren una tendencia creciente que se podría graficar, hay más de un modo de hacerlo:

- Uniendo las medias de cada grupo

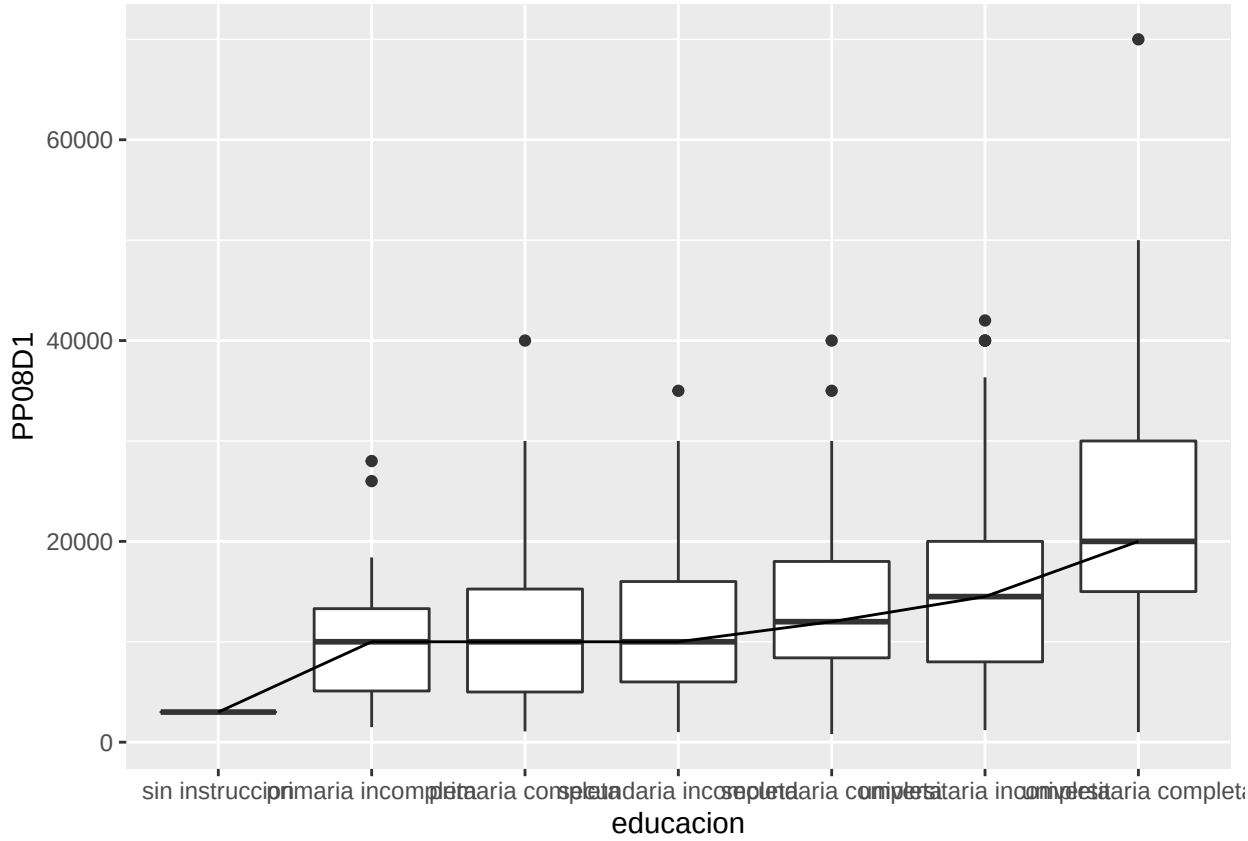
```
ggplot(
  asalariados.con.ingreso.y.horas.cordoba, aes(
    educacion, PP08D1))+
  geom_boxplot()+
  stat_summary(fun.y=mean, geom="point")+
  stat_summary(fun.y=mean, geom="line", aes(group=1))
```



Se agregó otra capa de resumen estadístico, que también calcula las medias grupales, pero ahora las une con una línea. La razón por la que debe ponerse `aes(group=1)` es un tanto críptica, a pesar de la explicacion de Wickham (2009), pero si no se indica así, el gráfico no sale.

- Se pueden unir las medianas de los grupos:

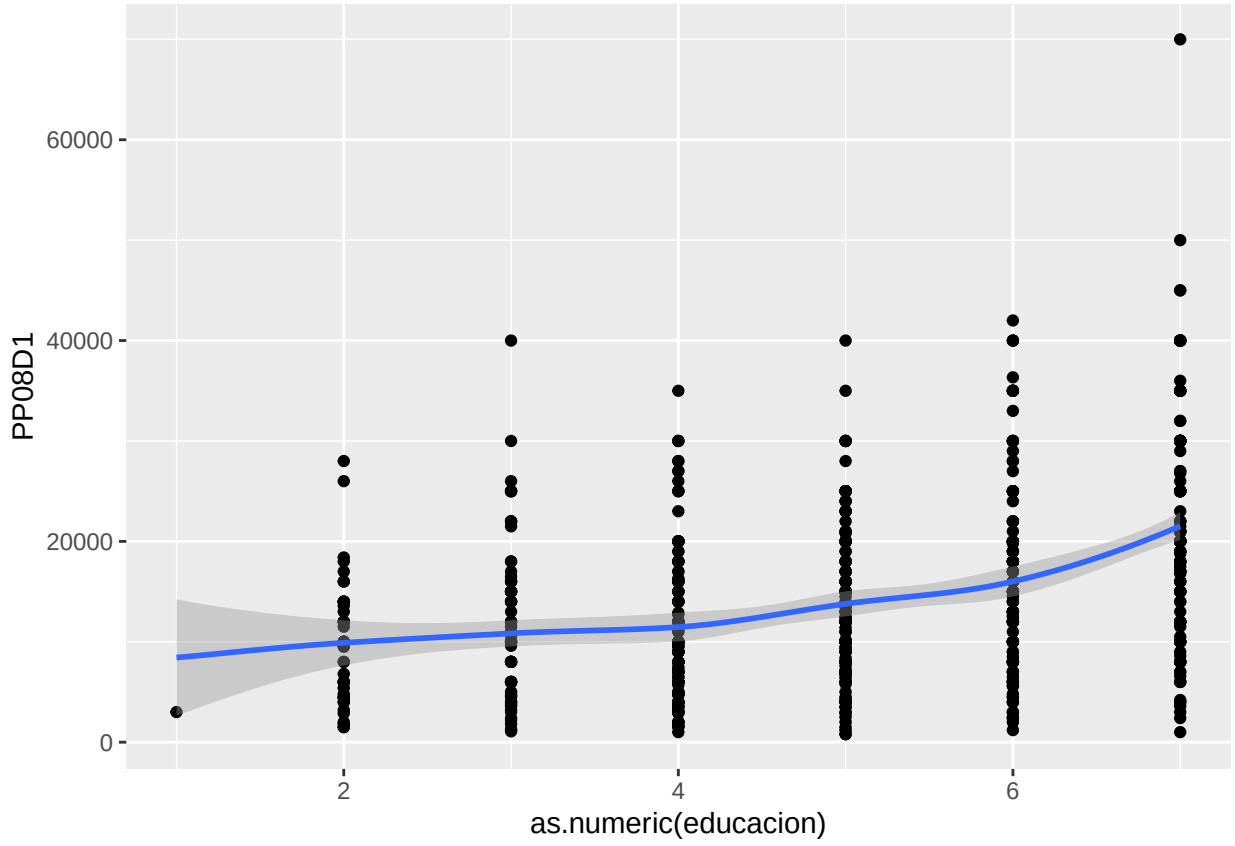
```
ggplot(
  asalariados.con.ingreso.y.horas.cordoba, aes(
    educacion, PP08D1))+
  geom_boxplot()+stat_summary(fun.y=median, geom="line", aes(group=1))
```



- O también tratar a la educación como numérica, descartar los box-plots y hacer un diagrama de dispersión con una función de ajuste:

```
ggplot(
  asalariados.con.ingreso.y.horas.cordoba, aes(
    as.numeric(educacion), PP08D1))+ geom_point()+geom_smooth()
```

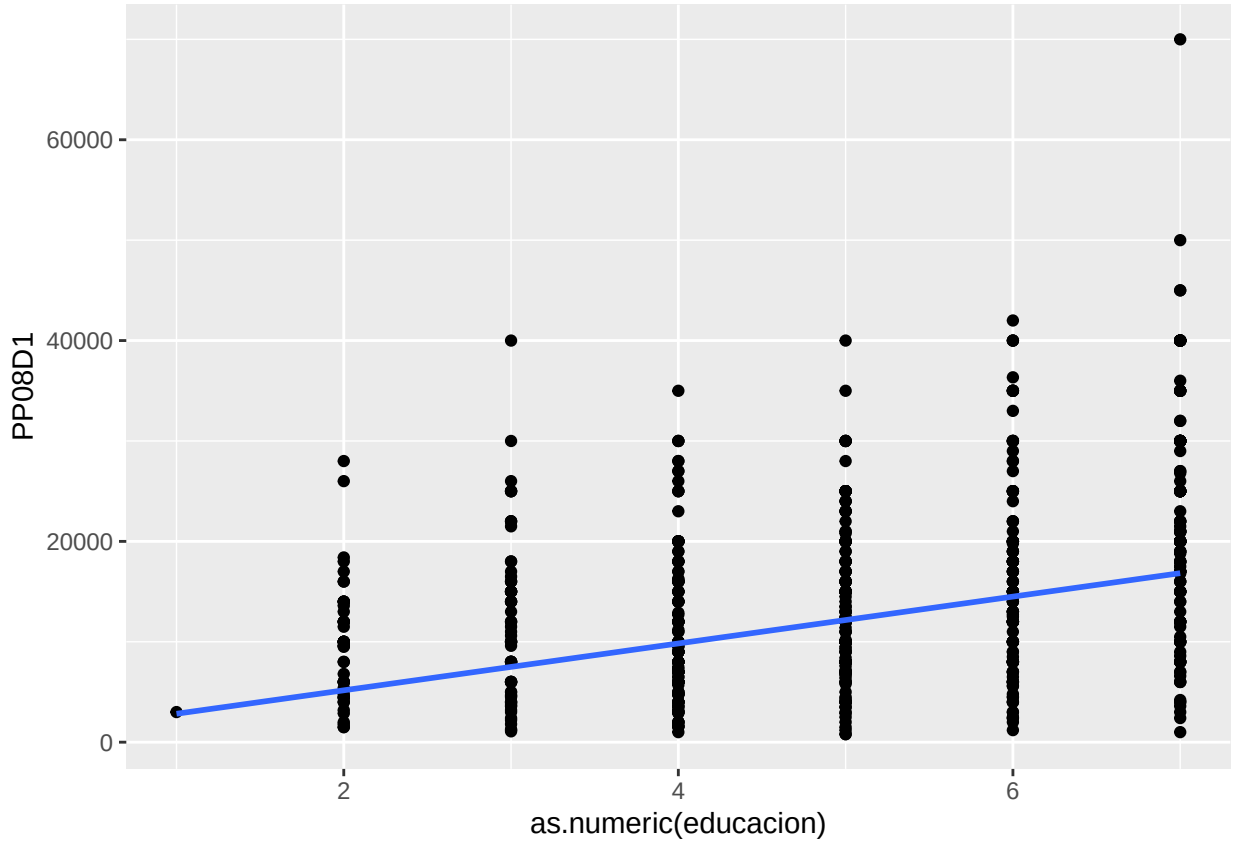
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



- Usando el estimador robusto de la pendiente de una recta y sin la banda de errores estándar:

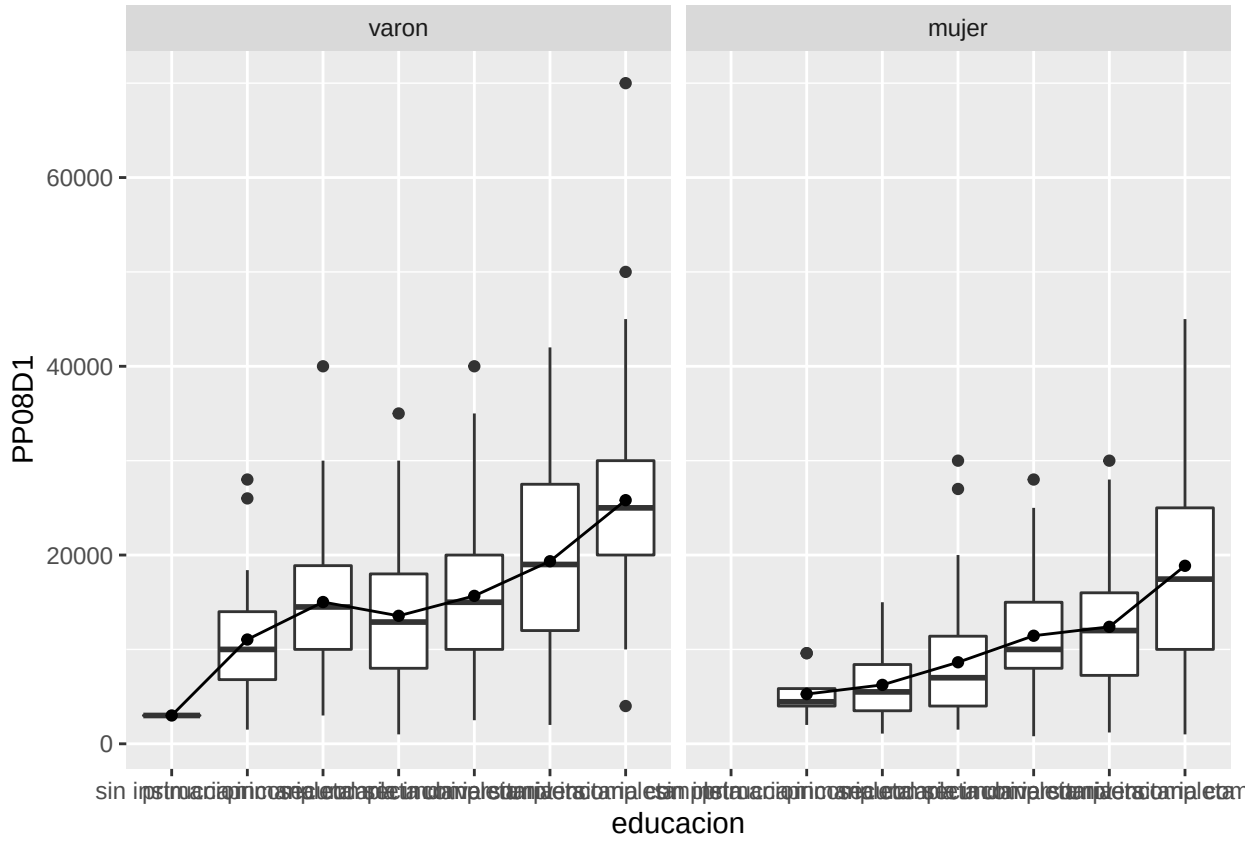
```
ggplot(
  asalariados.con.ingreso.y.horas.cordoba)+ geom_point(aes(
    as.numeric(educacion), PP08D1))+geom_smooth(aes(
    as.numeric(educacion), PP08D1), method = "estimador_theil_sen", se=FALSE)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



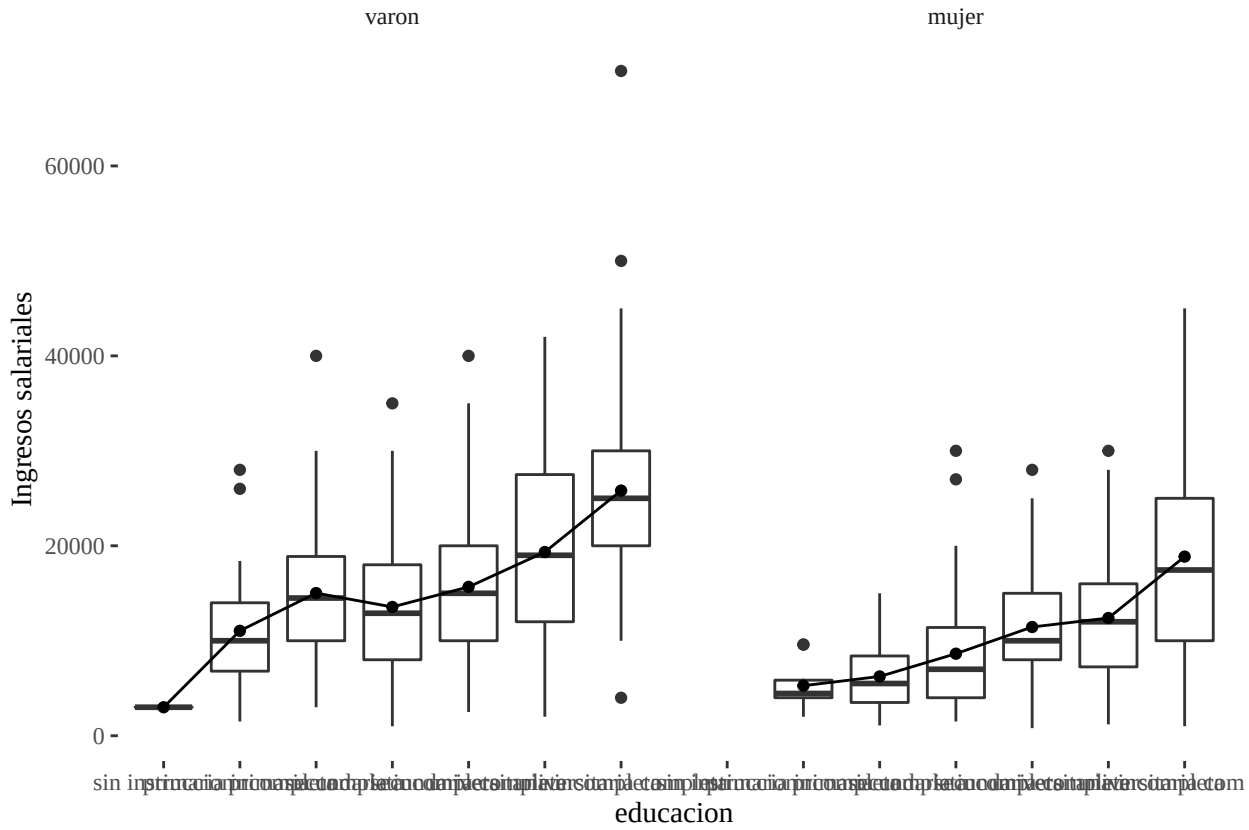
A cada uno de estos gráficos se los puede facetear por otra variable, por ejemplo, el gráfico de box plots con las medias, comparado entre varones y mujeres queda:

```
ggplot(
  asalariados.con.ingreso.y.horas.cordoba, aes(
    educacion, PP08D1))+
  geom_boxplot()+
  stat_summary(fun.y=mean, geom="point")+
  stat_summary(fun.y=mean, geom="line", aes(group=1))+
  facet_grid(.~sexo)
```



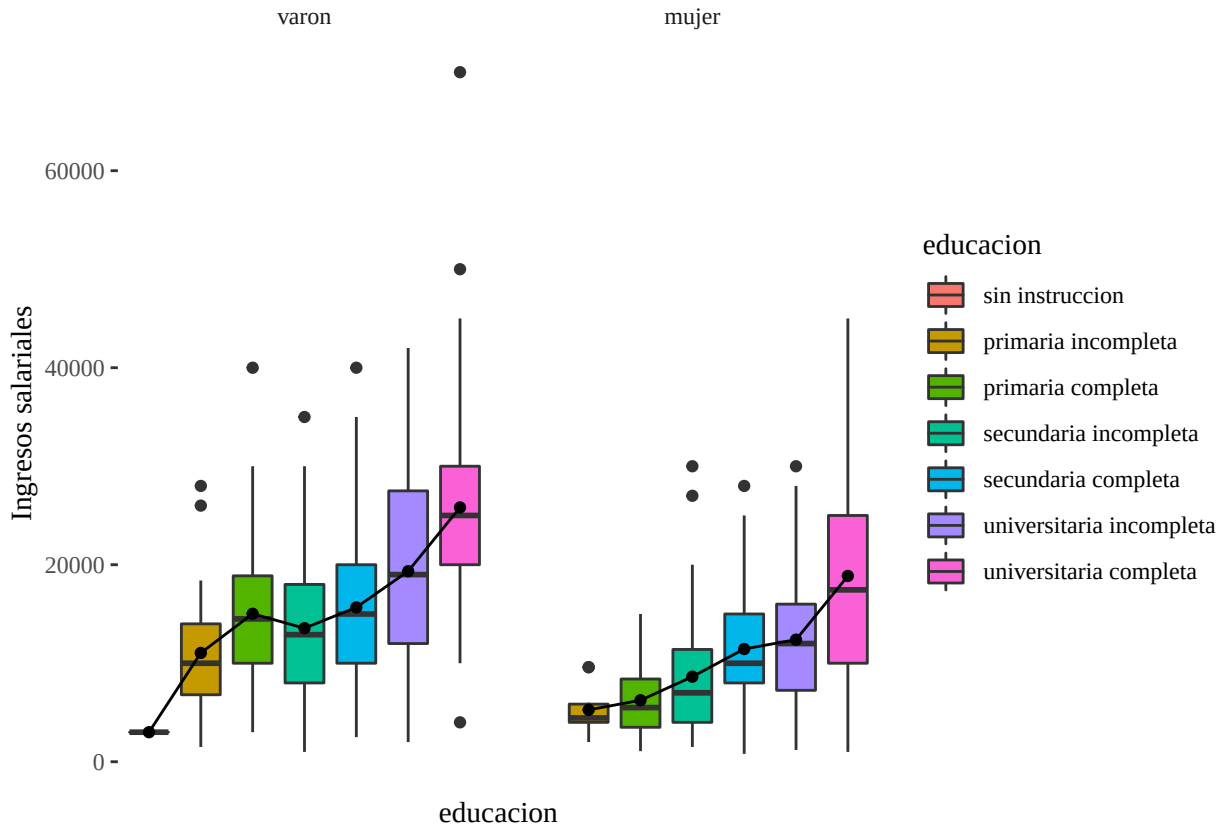
Con el formato de `tufte` y el nombre de la variable en el eje y mejora su aspecto

```
ggplot(
  asalariados.con.ingreso.y.horas.cordoba, aes(
    educacion, PP08D1))+
  geom_boxplot()+
  stat_summary(fun.y=mean, geom="point")+
  stat_summary(fun.y=mean, geom="line", aes(group=1))+
  facet_grid(.~sexo)+ ylab("Ingresos salariales")+
  theme_tufte()
```



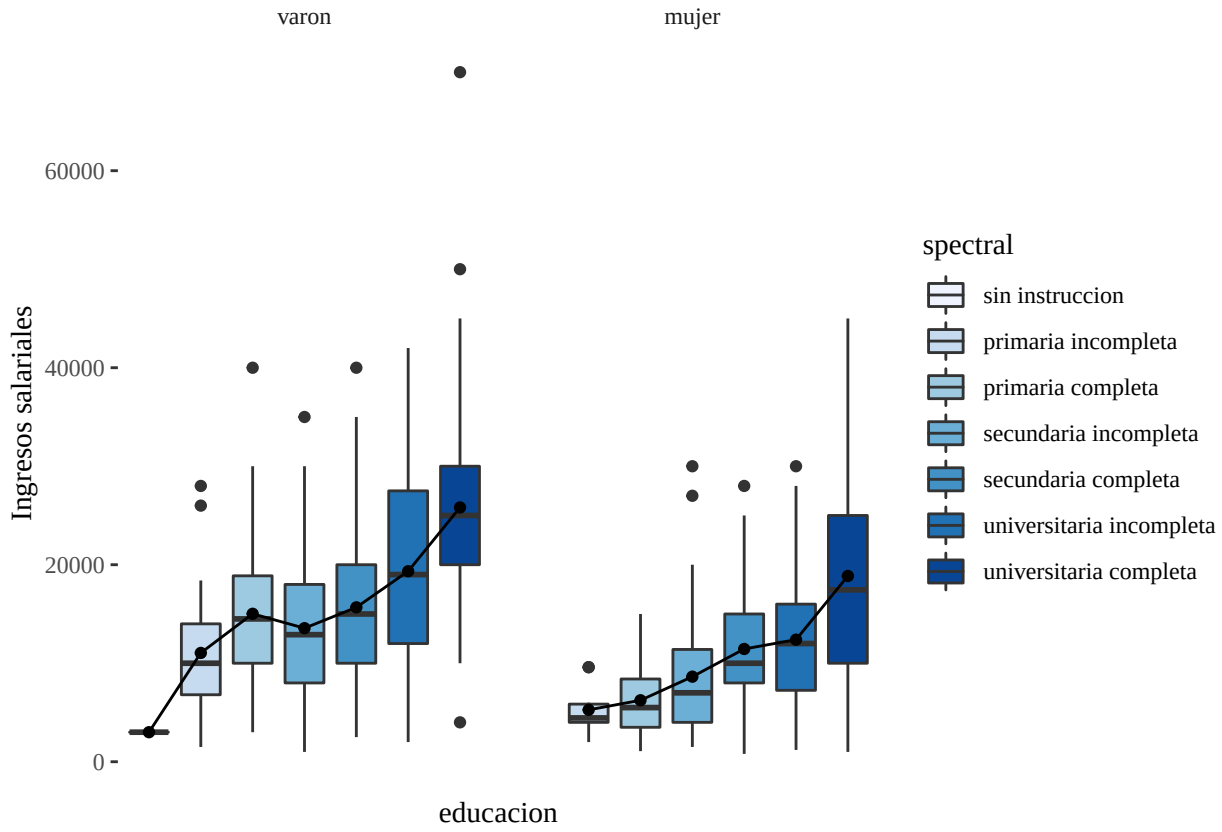
Para evitar los rótulos del eje x que han quedado ilegibles, una opción es colorear cada box plot según el nivel de educación (mapear la educación al color de los box-plots) y eliminar del eje los rótulos y las marcas, con la capa `theme`. Como la secuencia de instrucciones es larga, le ponemos un nombre y, para que en el mismo acto en que lo define como objeto, ejecute los comandos, encerramos todo entre paréntesis:

```
(p4<-ggplot(
  asalariados.con.ingreso.y.horas.cordoba, aes(
    educacion, PP08D1))+
  geom_boxplot(
    aes(
      fill=educacion))+ #acá se mapea la educación al color
  stat_summary(fun.y=mean, geom="point")+
  stat_summary(fun.y=mean, geom="line", aes(group=1))+
  facet_grid(.~sexo)+ ylab("Ingresos salariales")+
  theme_tufte()+
  theme(
    axis.text.x=element_blank(), # esto elimina los rótulos
    axis.ticks.x=element_blank())) # y esto las marcas
```



La combinación de colores se ajusta con la escala:

```
p4+scale_fill_brewer("spectral")
```

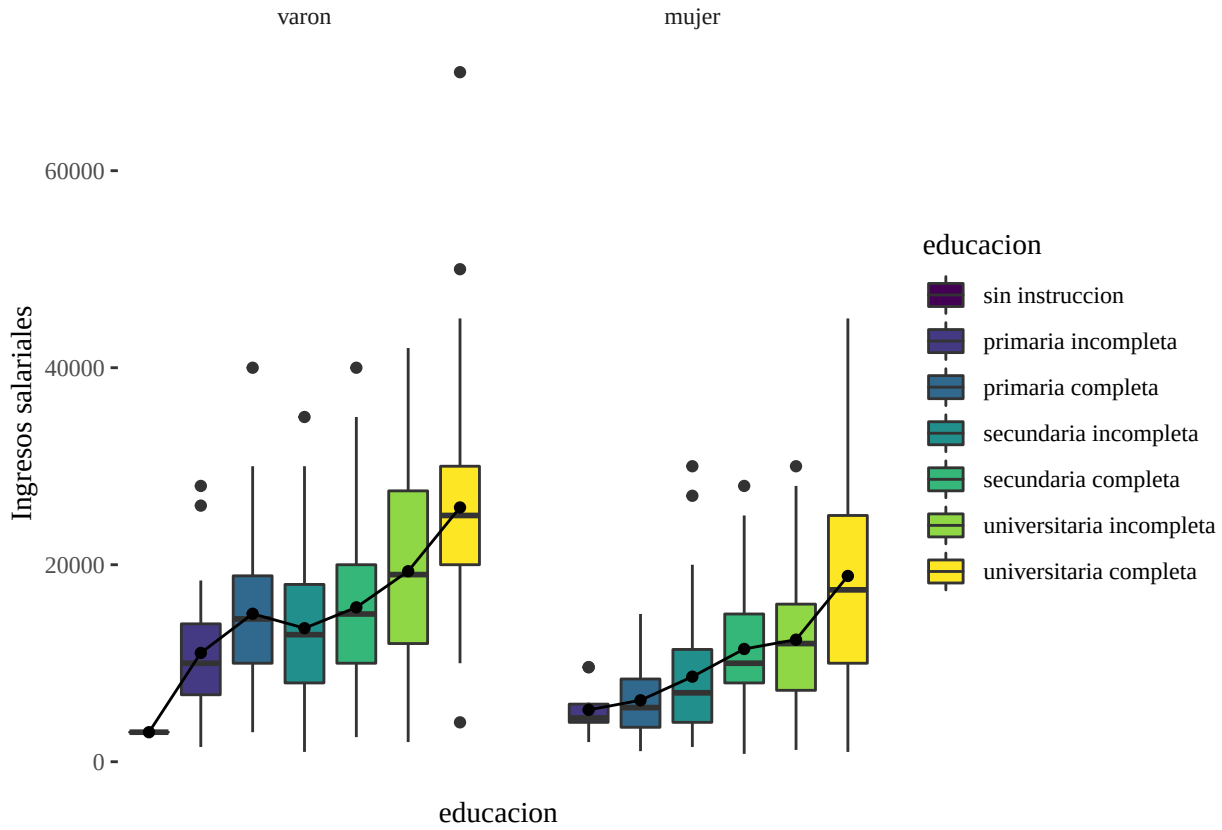


O, si se carga el paquete `viridis`

```
library(viridis)
```

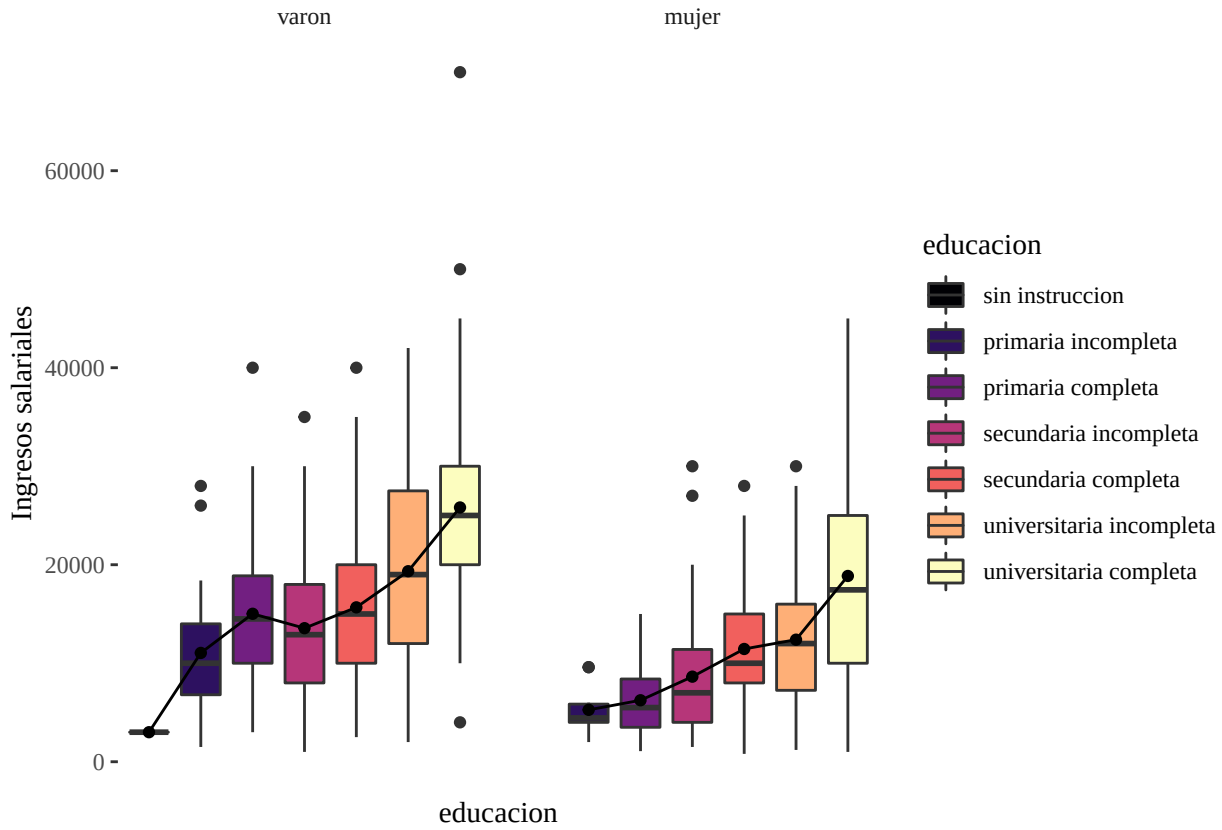
```
## Loading required package: viridisLite
```

```
p4+scale_fill_viridis_d()
```



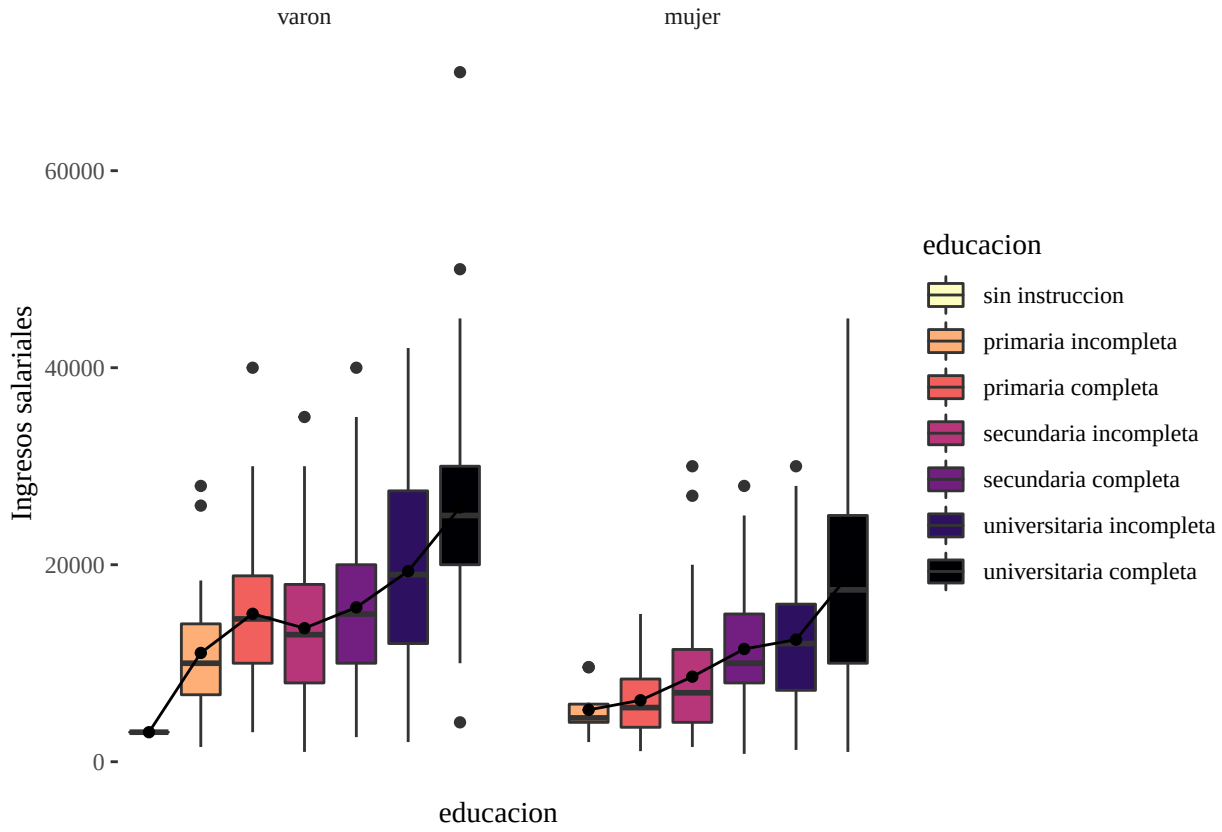
La `_d` en el comando indica que se mapea una variable discreta, además de la combinación que ofrece por defecto, tiene tres opciones: "A", "B" y "C". Por ejemplo, la opción A es:

```
p4+scale_fill_viridis_d(option = "A")
```



Se puede inverrtir el orden en que se asignan los colores, indicando que inicie con uno y termine con cero:

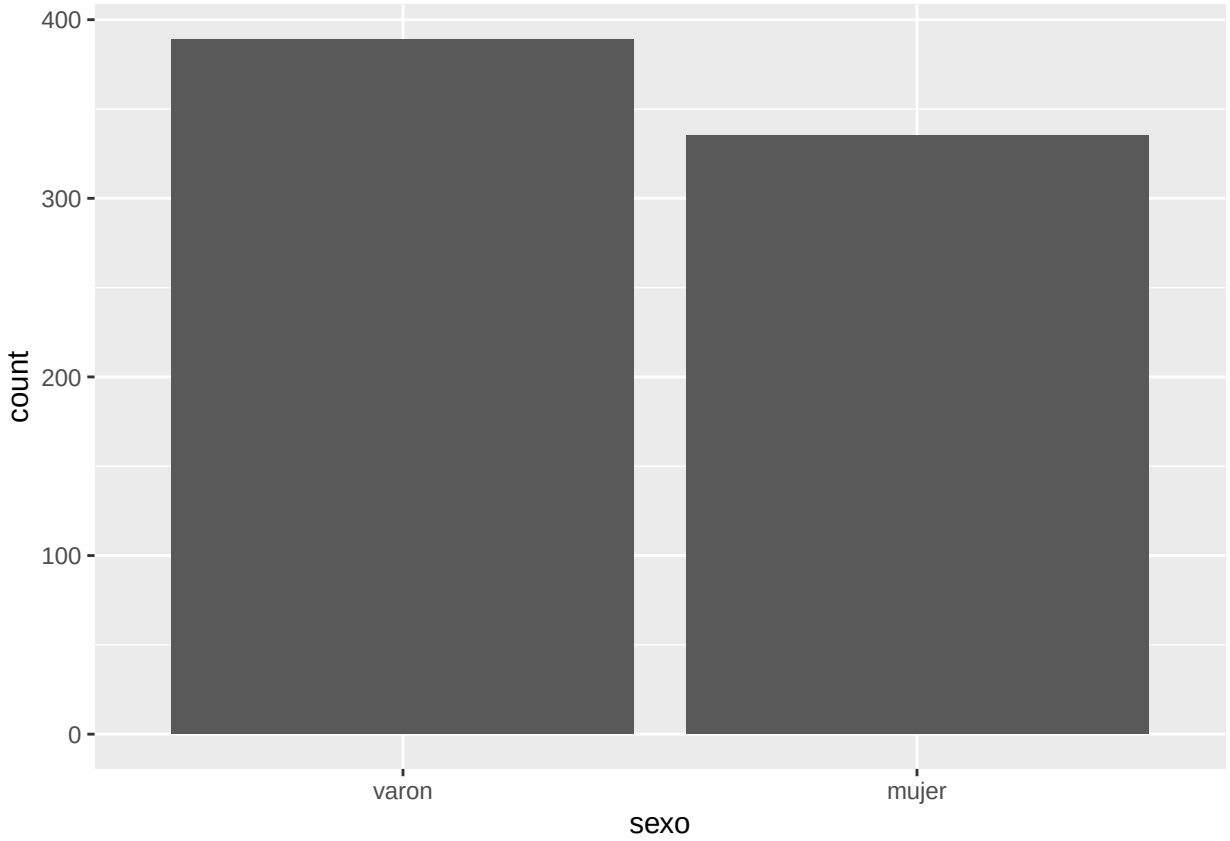
```
p4+scale_fill_viridis_d(option = "A", begin = 1,
  end = 0)
```



Variable categóricas

El más usado es el gráfico de barras:

```
ggplot(asalariados.con.ingreso.y.horas.cordoba)+
  geom_bar(aes(sexo))
```

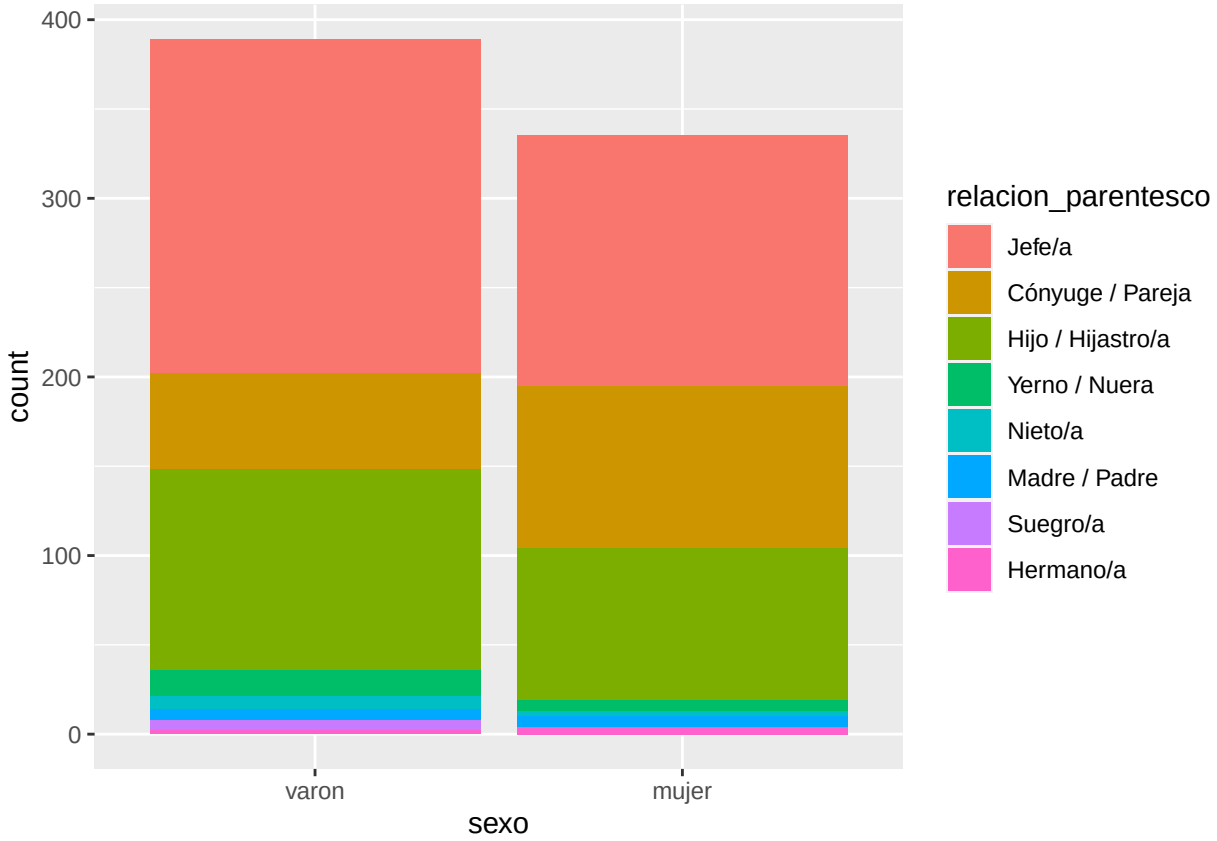


Sobre el cual se puede mapear al color la relación de parentesco. Pero antes de eso hay que tratar a esa variable como factor y asignarle sus etiquetas:

```
asalariados.con.ingreso.y.horas.cordoba$relacion_parentesco<-
  as.factor(asalariados.con.ingreso.y.horas.cordoba$CH03)

levels(asalariados.con.ingreso.y.horas.cordoba$relacion_parentesco)<-
  c("Jefe/a", "Cónyuge / Pareja", "Hijo / Hijastro/a",
    "Yerno / Nuera", "Nieto/a", "Madre / Padre",
    "Suegro/a", "Hermano/a", "Otros Familiares",
    "No Familiares")
```

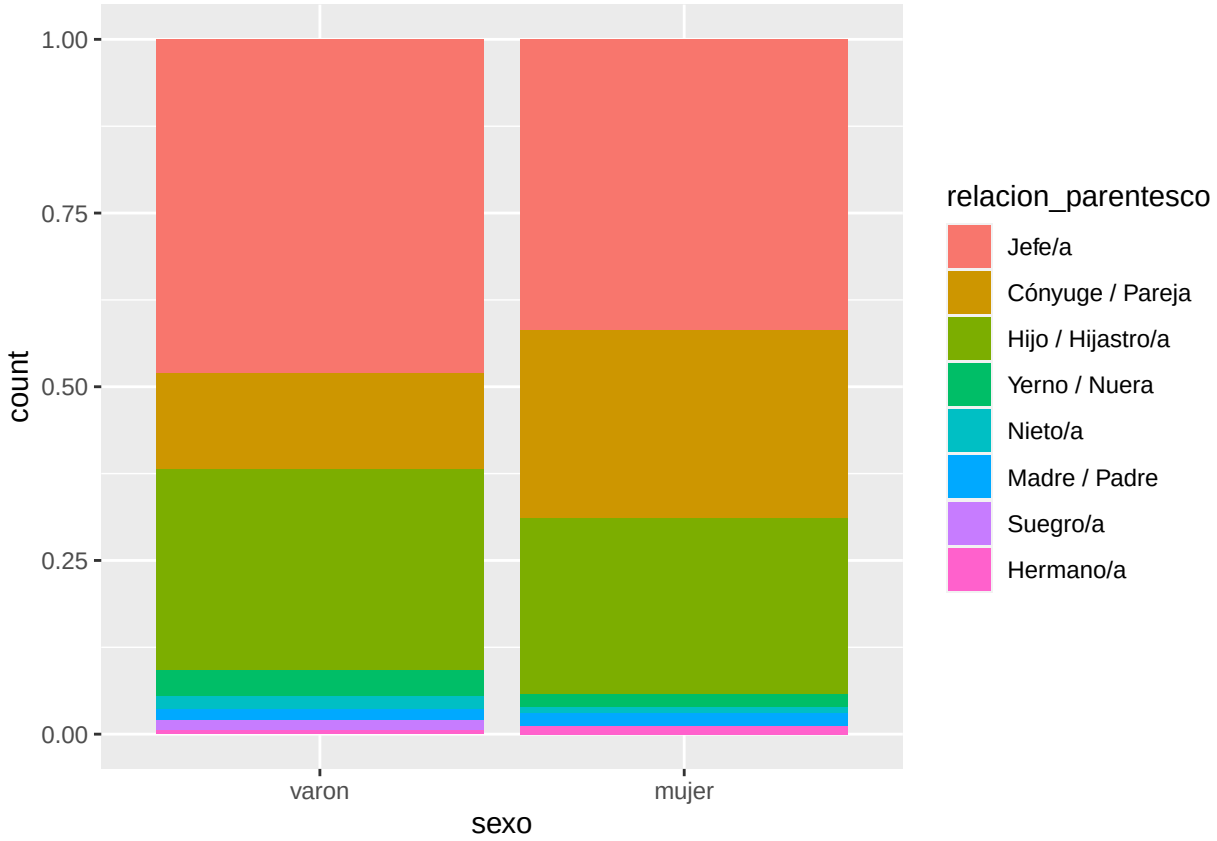
Luego, rellenamos las barras con los colores de las categorías de relación de parentesco.

```
ggplot(asalariados.con.ingreso.y.horas.cordoba)+
  geom_bar(aes(sexo, fill=relacion_parentesco))
```



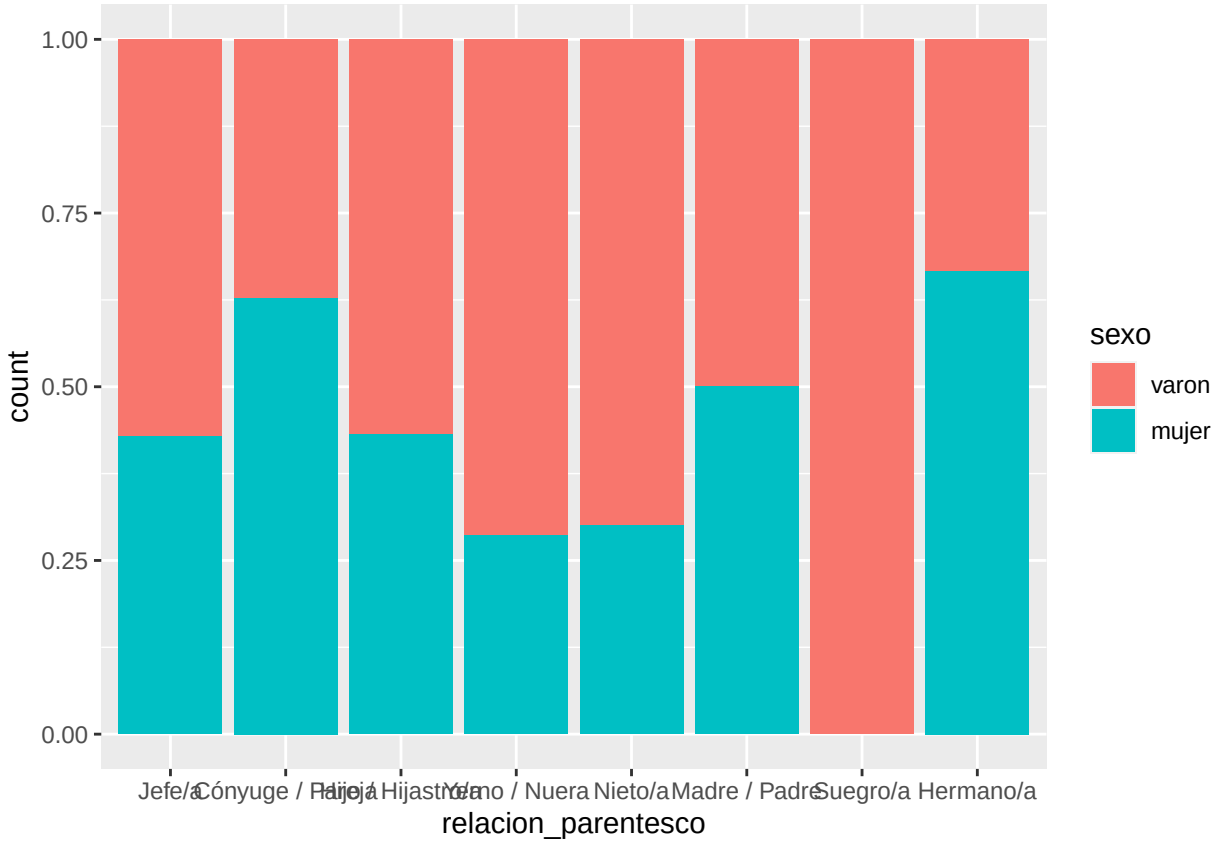
Para totalizar por las columnas del gráfico, `geom_bar` tiene el argumento "position":

```
ggplot(asalariados.con.ingreso.y.horas.cordoba)+
  geom_bar(aes(sexo, fill=relacion_parentesco), position = "fill")
```



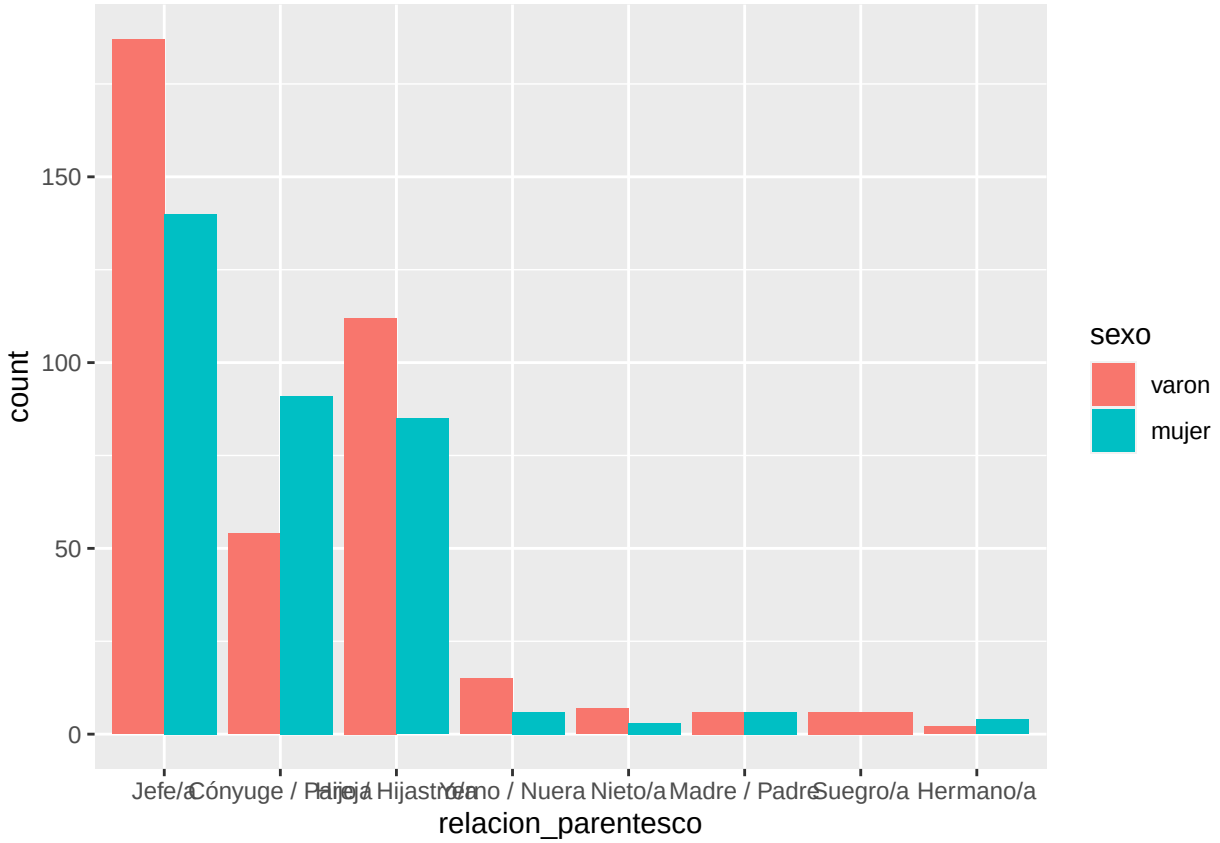
Este gráfico muestra el peso relativo de las diferentes relaciones de parentesco entre varones y mujeres. A la inversa:

```
ggplot(asalariados.con.ingreso.y.horas.cordoba)+
  geom_bar(aes(relacion_parentesco, fill=sexo), position = "fill")
```



Ahora se ve la composición por sexos de cada relación de parentesco.
Para facilitar la comparación, se colocan las barras al lado (*dodge*)

```
ggplot(asalariados.con.ingreso.y.horas.cordoba)+
  geom_bar(aes(relacion_parentesco, fill=sexo), position = "dodge")
```

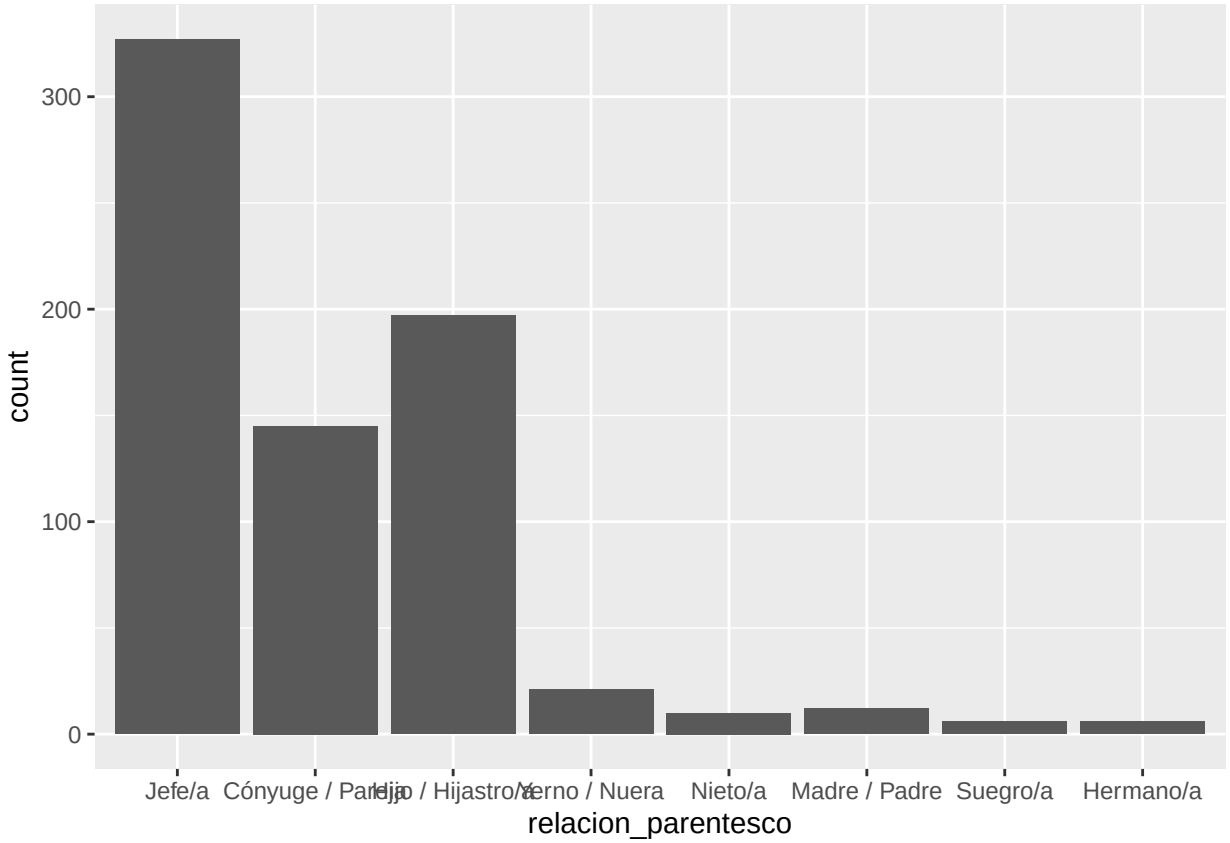


La posibilidad de ubicar el origen de los datos en `ggplot` o en el `geom_` permite que las capas provengan de un misma base de bases diferentes.

Datos provenientes de tablas

Si no se grafica desde la base de microdatos sino desde una tabla resumen, debe especificarse en `ggplot` o en la capa correspondiente. Por ejemplo, el gráfico de barras de la variable "relación de parentesco", se pide directamente desde la base original:

```
ggplot(
  asalariados.con.ingreso.y.horas.cordoba)+
  geom_bar(aes(relacion_parentesco))
```



La estética de este gráfico solo pide una variable, y la tranformación estadística consiste en contar las ocurrencias de cada categoría; eso es lo que hace por defecto `geom_bar`, usa `stat=count`. Pero si se parte de una tabla, como la siguiente:

```
table(
  asalariados.con.ingreso.y.horas.cordoba$relacion_parentesco)
```

```
##
##            Jefe/a  Cónyuge / Pareja Hijo / Hijastro/a     Yerno / Nuera
##               327               145               197                21
##           Nieto/a     Madre / Padre          Suegro/a         Hermano/a
##                10                12                 6                 6
##  Otros Familiares     No Familiares
##                 0                 0
```

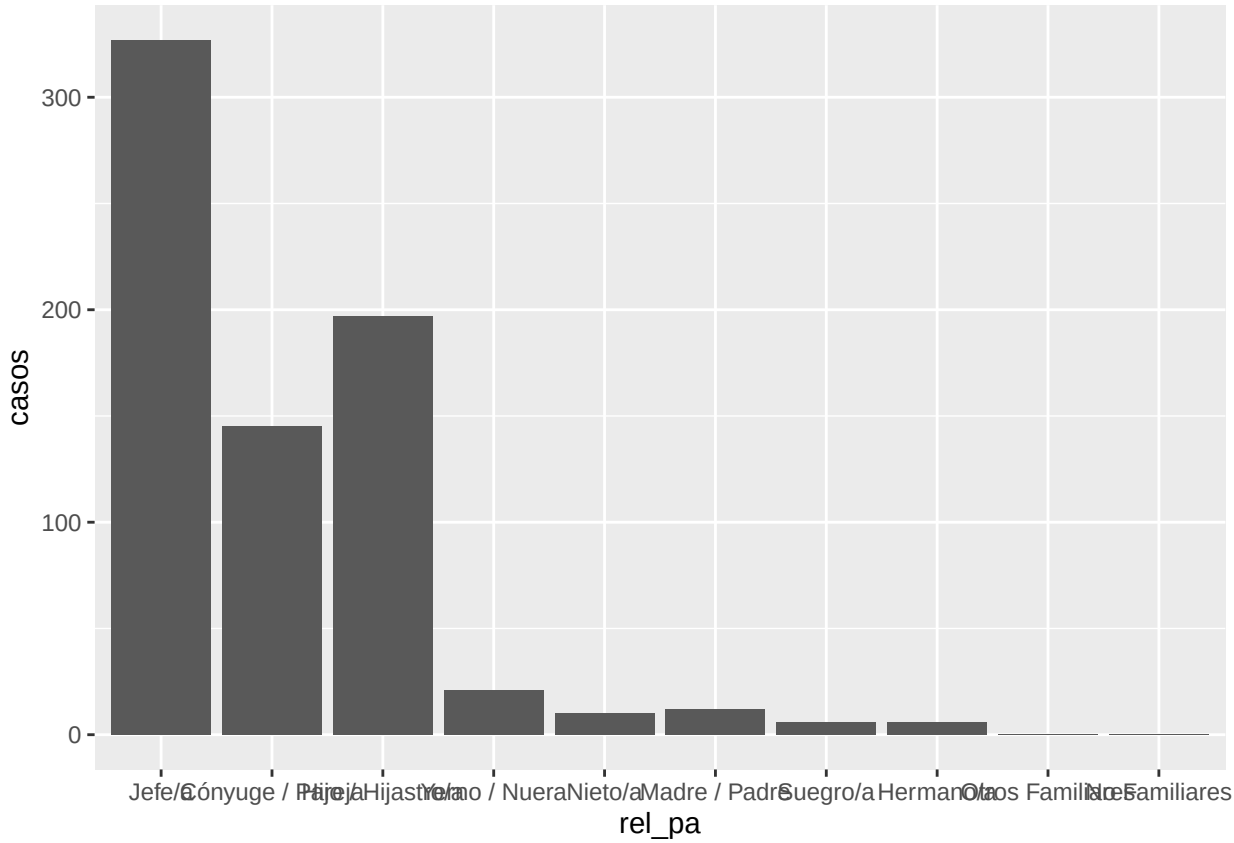
Se la debe denominar, guardar como data frame y, es recomendable, darle nombres a las columnas:

```
tabla_rela_pa<-as.data.frame(
  table(asalariados.con.ingreso.y.horas.cordoba$relacion_parentesco)
)

names(tabla_rela_pa)<-c(
  "rel_pa", "casos"
)
```

Para pedir el gráfico de esta tabla, la estética tiene que informar no solo la variable que se grafica sino las

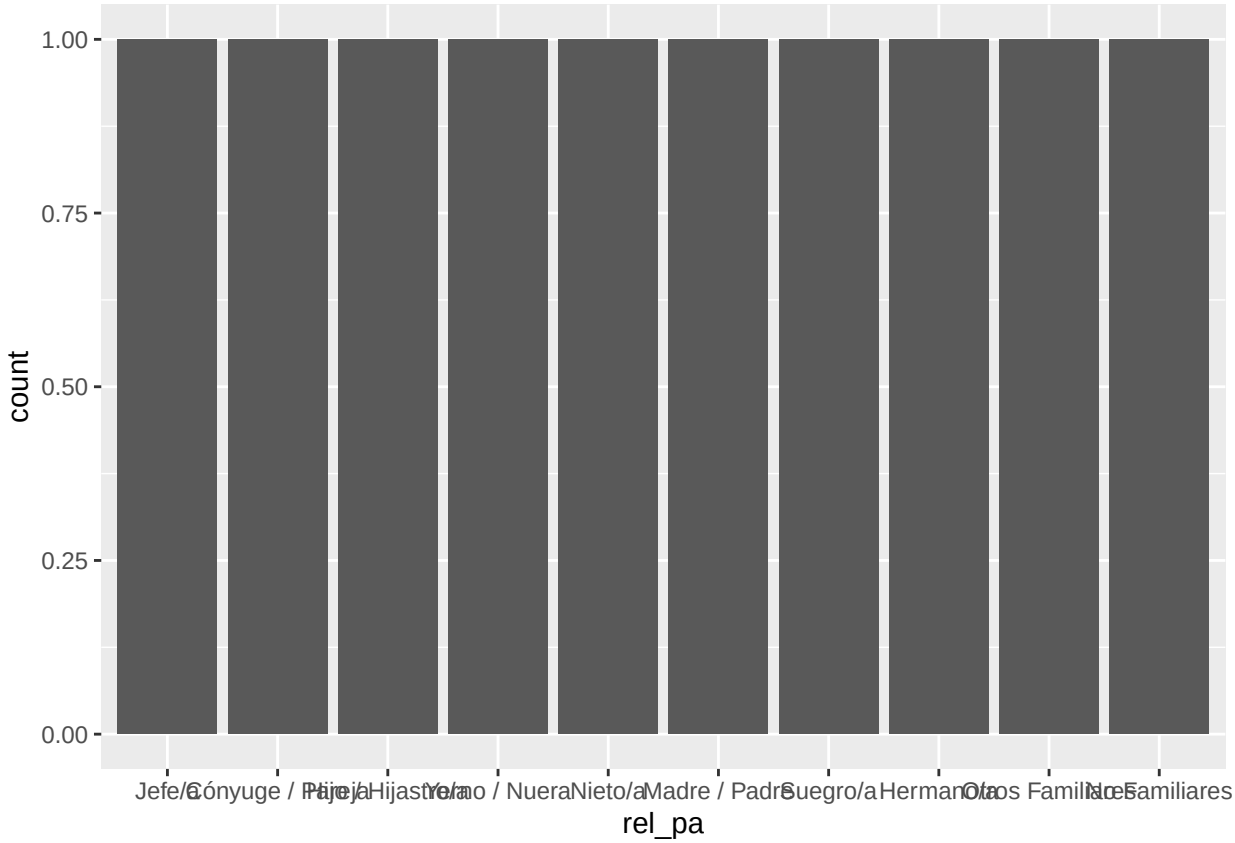
frecuencias, y debe modificarse la transformación estadística a `"identity"`, que indica que use el valor y de la estética (el que está en segundo término) como frecuencias:

```
ggplot(
  tabla_rela_pa)+
  geom_bar(aes(rel_pa,casos), stat = "identity")
```



Si tratáramos a la tabla como al data.frame original y dado que la transformación estadística por defecto de `geom_bar` en el recuento de casos, va a encontrar **una** repetición de cada categoría y el grafico queda:

```
ggplot(
  tabla_rela_pa)+
  geom_bar(aes(rel_pa))
```



El paquete `demography`

Los conjuntos de datos para las aplicaciones de esta sección han sido incluidos en la carpeta que está instalada en el mismo lugar que el proyecto actual, por lo ue deberían estar accesibles.

El paquete R `demography` provee funciones para el análisis demográfico, que incluyen: elaboración de tablas de mortalidad, modelización de Lee-Carter, análisis funcional de datos sobre tasas de mortalidad, fecundidad y migración neta, y proyecciones estocásticas de población.

Elaboración de tablas de mortalidad: una tabla de mortalidad describe la demografía de una población en términos de supervivencia, escencialmente, el número de individuos que se espera que alcancen a la próxima edad o la próxima etapa de su vida.

Modelización de Lee-Carter: es un enfoque para proyectar mortalidad, que no es frecuente en los paquetes estadísticos.

Análisis funcional de datos: permite el análisis de diferentes indicadores de una población: tasas de mortalidad y fecundidad, migración neta y proyecciones estocásticas de población.

Aplicación

Se carga en la sesión el paquete que ya ha sido instalado

```
library(demography)
```

```
## Loading required package: forecast
```

```
## This is demography 1.22
```

Para importar las bases de datos a R se usa el comando `read.demogdata()` del paquete demography. La estructura del comando se ve en la ayuda:

```
?demography
```

Y resulta:

```
read.demogdata(archivo de tasas, archivo de población,
               type, label, max.mx = 10, skip = 2,
               popskip = (por defecto coincide con skip),
               lambda, scale=1)
```

El *archivo de tasas* es el archivo que contiene las tasas que interesan: mortalidad o fecundidad
El *archivo de población* contiene la composición de la población por sexo y edad
Type es el componente de la dinámica demográfica que se analiza, puede valer: "mortality", "fertility" o "migration"
Label es el nombre del área que se analiza
Max.mx establece un límite superior para las tasas bajo análisis; su función es la de tratar a todo valor que sea mayor que ese número, como igual a él
Skip indica cuántas líneas deben saltearse antes de empezar a leer el archivo, para el caso de archivos que tengan títulos antes de los nombres de las columnas
Popskip es lo mismo para el archivo de población
Lambda es un valor que se usa como parámetro en la transformación de Box-Cox Es un modo de convertir datos no normales en normales. Los valores por defecto son 0 para mortalidad, 0.4 para fecundidad y 1 para migración
Scale indica la escala de los datos, si las tasas son por indviduo "scale=1" si son por mil "scale=1000"

Vamos a usar datos del proyecto The Human Mortality Database (Shkolnikov, Barbieri, and Wilmoth 2020). Una vez hecho el registro (sin costo) se pueden descargar los datos para 41 países o áreas y diversos períodos. Para este ejemplo se usarán los datos de USA. Las bases que nos van a interesar por el momento son las de tasas de mortalidad y exposición al riesgo, ambas en la sección "datos periódicos". Para aprovechar las funciones de este paquete, es necesario adecuar otras tablas a este formato.

Una vez que se han descargado las dos bases (tasas y población sometida al riesgo) en la carpeta donde se encuentra el proyecto, conviene observar los archivos .txt, para tener en mente la estructura general de los datos.
Luego las bases son leídas convirtiéndolas en un objeto de clase "demogdata", que será usado por varias funciones del paquete.
Hemos usado los nombres con que vienen por defecto las bases de HMD: Mx_1x1.txt para las tasas de mortalidad y Exposures_1x1.txt para la población expuesta.

```
USA_mortalidad<-read.demogdata(
  file="Mx_1x1.txt",popfile="Exposures_1x1.txt", type="mortality",
  label="USA",skip=2, scale=1 )
```

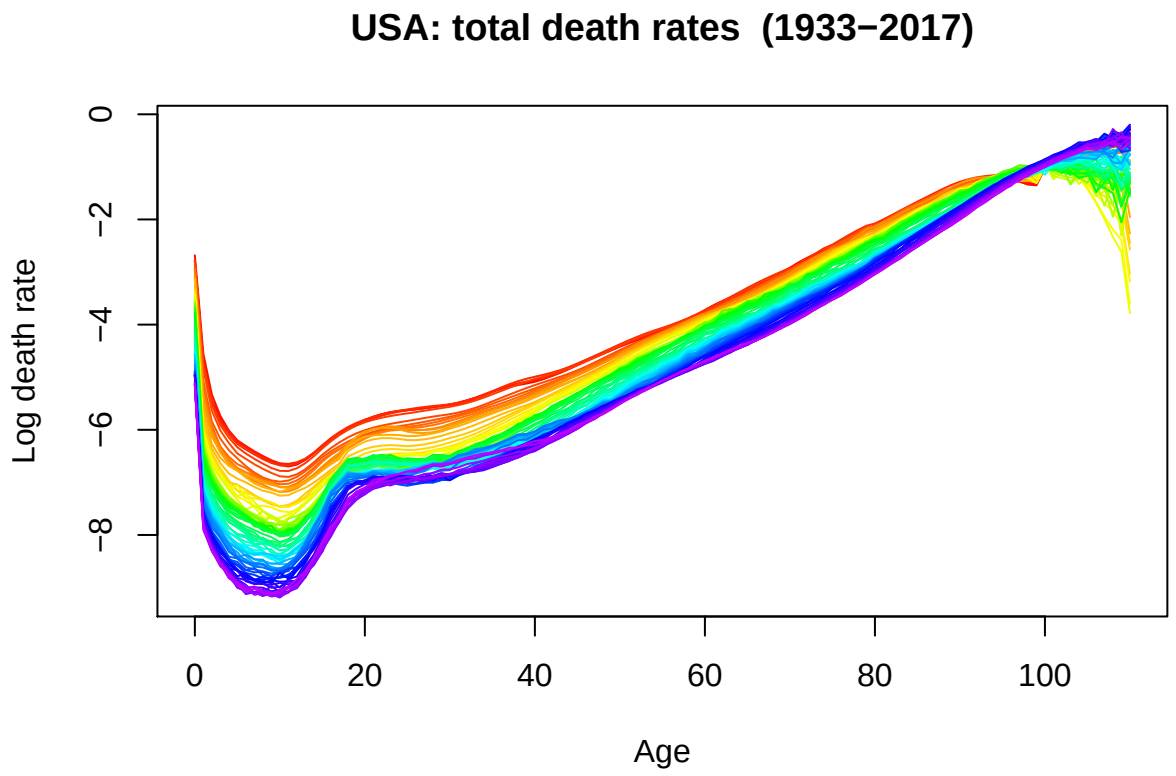
Estos datos se pueden visualizar con el comando plot, con estructura

```
plot(x, series, datatype, years, ages, max.age,
  transform, plot.type, type = "l", main, xlab, ylab,)
```

cuyos argumentos son:
x: el objeto de clase demogdata con que se está trabajando
series: el nombre de la serie que se va a graficar. Si no se especifica, por defecto toma la primera matriz, según datatype
datatype: por defecto es tasas ("rate"), la opción es especificar población datatype="pop"
years: un vector que especifica los años a graficar, por defecto, toma todos los que están en la serie
ages: igual que los años, si no se indica, se toman todas las edades disponibles
max.age: la máxima edad a graficar, por defecto, las disponibles
transform: lógico, por defecto TRUE, que indica que los datos se transforman antes de graficar
plot.type: tipo de gráfico pueden ser funciones o tiempo
main: título del gráfico
xlab e *ylab*: rótulos de los ejes

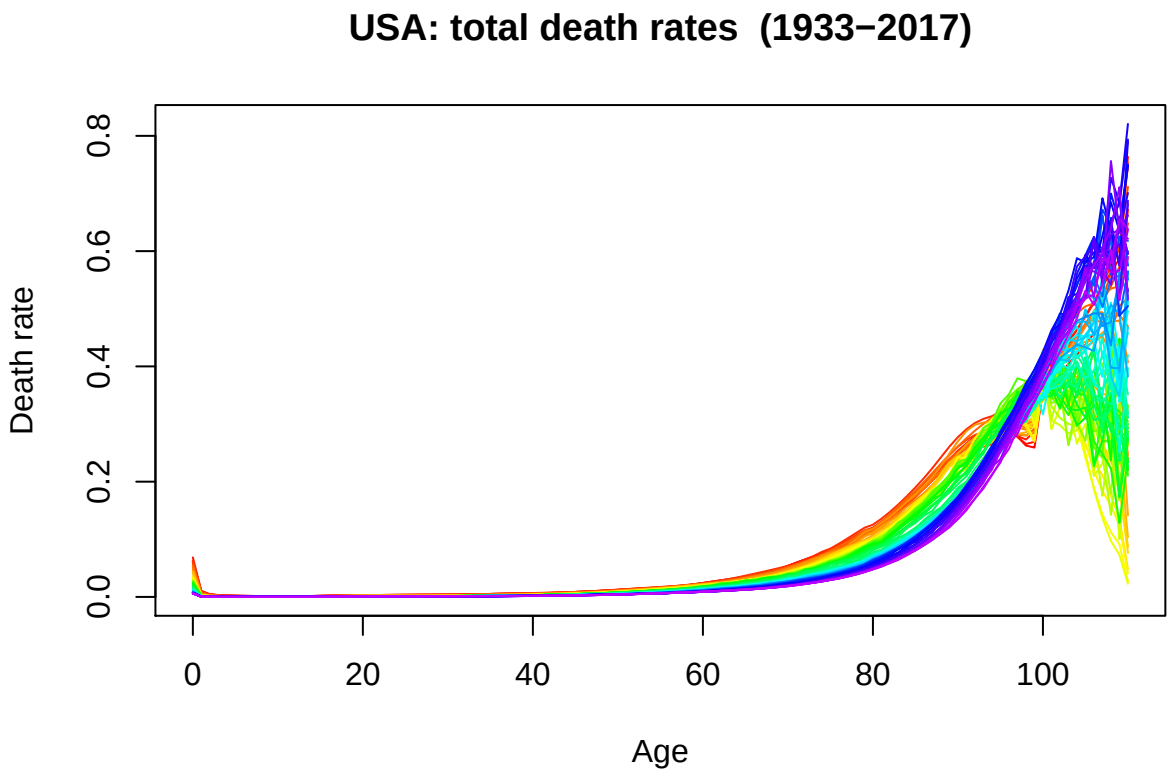
La mayoría de estos argumentos tienen sus valores definidos por defecto y, para un gráfico básico, alcanza con indicar el origen de los datos y la serie elegida. Así, para el logaritmo de tasas de mortalidad (en el eje vertical) por edades (eje horizontal) y año (representado en el color), se pide simplemente:

```
plot(USA_mortalidad, series= "Total")
```



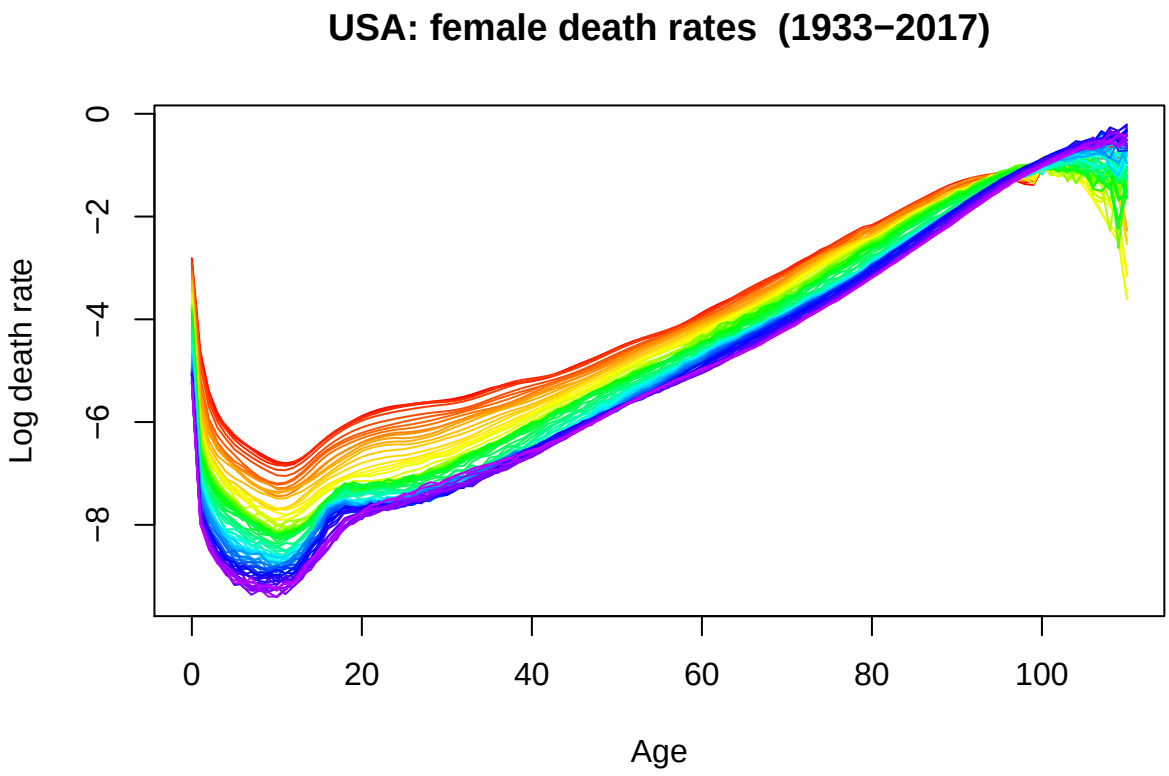
Si no se quiere la transformación logarítmica, se indica:

```
plot(USA_mortalidad, series= "Total", transform = FALSE)
```



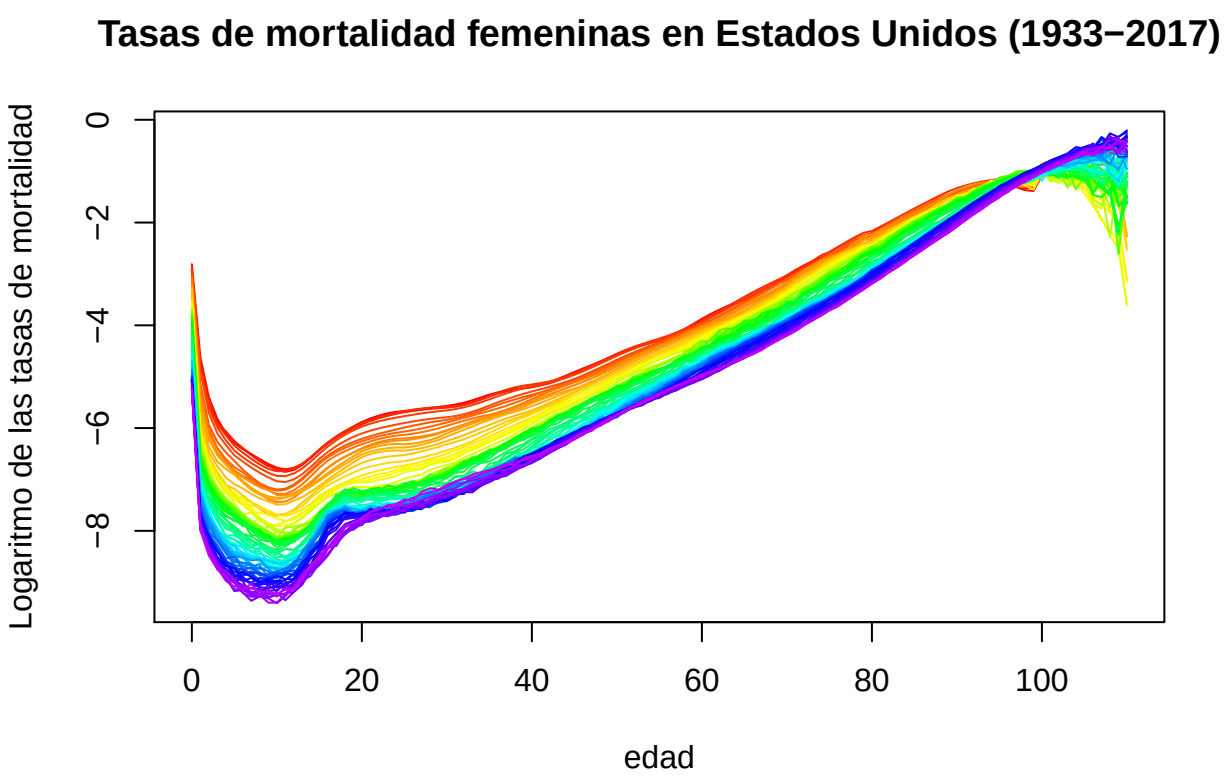
Dado que la transformación logarítmica permite apreciar mejor las diferencias, la conservamos, y solicitamos solo de mujeres:

```
plot(USA_mortalidad, series= "Female")
```



Se pueden ajustar las opciones de formato por defecto

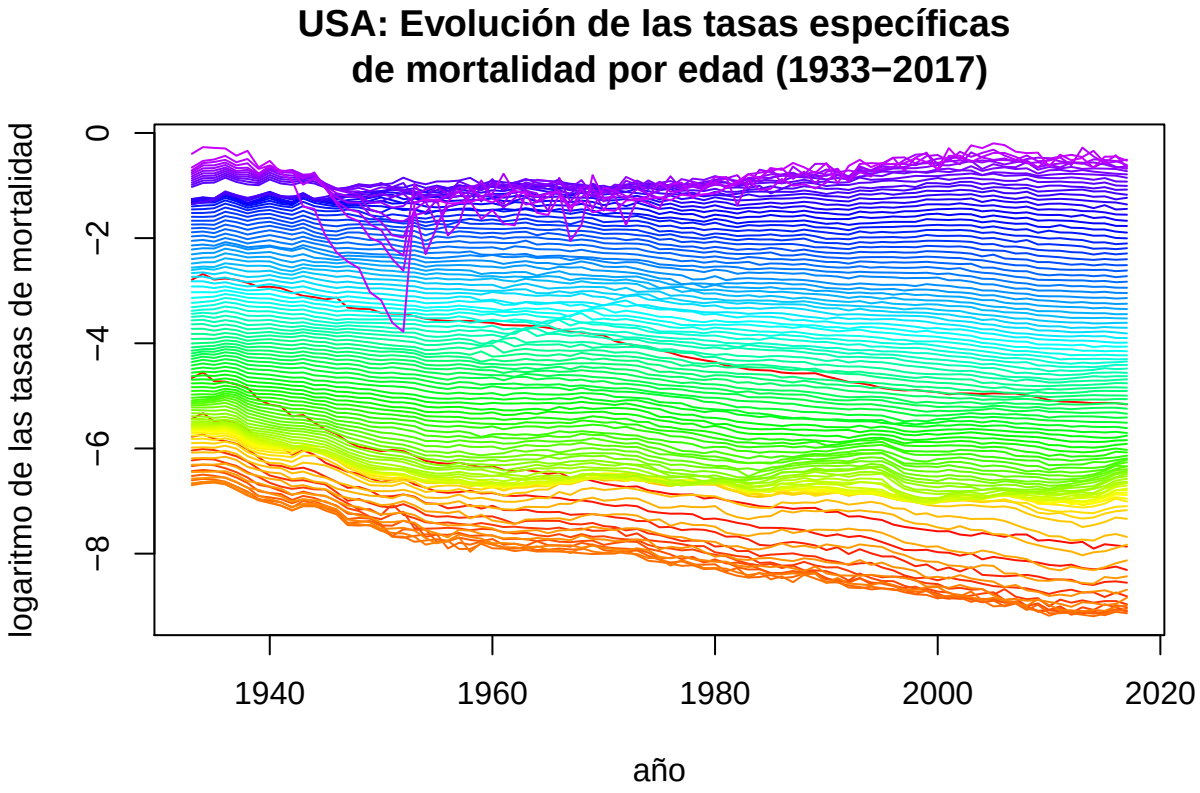
```
plot(USA_mortalidad, series= "Female",
     main = "Tasas de mortalidad femeninas en Estados Unidos (1933-2017)",
     xlab="edad",
     ylab="Logaritmo de las tasas de mortalidad")
```



El todos los gráficos se ve la tendencia general según la cual las tasas tienen un comportamiento similar a lo largo de la vida de las personas y decrecen a lo largo de la historia. El apartamiento del patrón por edades se observa en las edades mayores, donde la caída de la mortalidad aparece más acentuada.

Si se cambia el tipo de gráfico por "tiempo", se visualiza cómo se modifican las tasas esecíficas de mortalidad por edad a lo largo del tiempo. Ahora cada color representa una edad y el eje horizontal es la línea de tiempo. Ajustamos los nombres de los ejes:

```
plot(USA_mortalidad, series= "Total", plot.type = "time",
     main="USA: Evolución de las tasas específicas \n  de mortalidad por edad (1933-2017)",
     xlab = "año",
     ylab = "logaritmo de las tasas de mortalidad")
```



Al que le haría falta un pequeño ajuste de formato, porque el paquete mantiene el nombre del eje. Se puede modificar con xlab.

Luego de haberse registrado en HMD, hay un comando R que permite acceder a los datos directamente desde la página. El comando es:

```
hmd.mx(country,username,password, label=country)
```

En el que:
country es el nombre abreviado del país que se soilicita
username y *password*, son los que corresponden a la cuenta creada
label es la etiqueta, que debe coincidir con el nombre del país que se pidió
(todo va entre comillas)

Tablas de mortalidad

El comando para producir una tabla de mortalidad es `lifetable`, con estructura:

```
lifetable(x, series = names(data$rate)[1], years = data$year,
  ages = data$age, max.age = min(100, max(data$age)),
  type = c("period", "cohort"))
```

data es el archivo `demogdata` que se va a usar
series es el nombre de la serie que se va a usar, por defecto es 1. En este caso 1 es mujeres , 2, varones y 3 el total

years es el vercto ed los años a incluir
ages es el vector de la edades a incluir
max.age la edad para la última fila, las filas siguientes se combinan
type si son datos de príodo o de cohorte

Para construir la tabla de mortalidad para USA en 2016 y guardarla con el nombre "tabla_de_mortalidad", se solicita

```
tabla_de_mortalidad<-
  lifetable(USA_mortalidad, series="Total", years=2016)
```

El objeto resultante es una lista que contiene las columnas usuales de una tabla de mortalidad:

```
names(tabla_de_mortalidad)
```

```
##  [1] "age"    "year"   "mx"     "qx"     "lx"     "dx"     "Lx"     "Tx"
##  [9] "ex"     "rx"     "series" "type"   "label"
```

Y se los puede ver impresos individualmente, por ejemplo, la esperanza de vida para cada edad:

```
tabla_de_mortalidad$ex
```

```
##          2016
## 0   78.875744
## 1   78.338066
## 2   77.368585
## 3   76.388727
## 4   75.403755
## 5   74.415740
## 6   73.425793
## 7   72.434618
## 8   71.443682
## 9   70.451912
## 10  69.459677
## 11  68.467401
## 12  67.475694
## 13  66.484334
## 14  65.494694
## 15  64.508345
## 16  63.526206
## 17  62.549719
## 18  61.579944
## 19  60.620698
## 20  59.665866
## 21  58.714283
## 22  57.769905
## 23  56.826229
## 24  55.884556
## 25  54.943572
## 26  54.004583
## 27  53.064328
## 28  52.127022
```

```
## 29  51.191803
## 30  50.254344
## 31  49.318519
## 32  48.384224
## 33  47.452220
## 34  46.519692
## 35  45.589603
## 36  44.660178
## 37  43.732528
## 38  42.805393
## 39  41.881017
## 40  40.956773
## 41  40.033025
## 42  39.111338
## 43  38.192120
## 44  37.280083
## 45  36.371300
## 46  35.466665
## 47  34.567584
## 48  33.671478
## 49  32.784520
## 50  31.905425
## 51  31.034042
## 52  30.169792
## 53  29.316217
## 54  28.471609
## 55  27.636060
## 56  26.808787
## 57  25.988138
## 58  25.177065
## 59  24.373508
## 60  23.579779
## 61  22.791231
## 62  22.010626
## 63  21.238332
## 64  20.472119
## 65  19.716021
## 66  18.962889
## 67  18.213813
## 68  17.474282
## 69  16.745474
## 70  16.026543
## 71  15.316862
## 72  14.620426
## 73  13.942241
## 74  13.278113
## 75  12.627699
## 76  11.990523
## 77  11.362347
## 78  10.752823
## 79  10.157309
## 80   9.580387
## 81   9.024818
## 82   8.486774
```

```
## 83   7.968045
## 84   7.465084
## 85   6.986932
## 86   6.532340
## 87   6.098584
## 88   5.684033
## 89   5.286357
## 90   4.901044
## 91   4.541950
## 92   4.212051
## 93   3.905425
## 94   3.625232
## 95   3.367656
## 96   3.124947
## 97   2.931744
## 98   2.732745
## 99   2.572675
## 100  2.417898
```

Si hubiésemos pedido dos años, por ejemplo el inicio y fin de la serie disponible:

```
tabla_de_mortalidad_compara<-
  lifetable(USA_mortalidad, series="Total", years=c(1933,2016))
```

Las esperanzas de vida por edades resultan:

```
tabla_de_mortalidad_compara$ex
```

```
##          1933      2016
## 0   60.906901 78.875744
## 1   63.676859 78.338066
## 2   63.277289 77.368585
## 3   62.551028 76.388727
## 4   61.743934 75.403755
## 5   60.890237 74.415740
## 6   60.009625 73.425793
## 7   59.115289 72.434618
## 8   58.209795 71.443682
## 9   57.294633 70.451912
## 10  56.371585 69.459677
## 11  55.442475 68.467401
## 12  54.510316 67.475694
## 13  53.579385 66.484334
## 14  52.653642 65.494694
## 15  51.736319 64.508345
## 16  50.829654 63.526206
## 17  49.933440 62.549719
## 18  49.046822 61.579944
## 19  48.169217 60.620698
## 20  47.300248 59.665866
## 21  46.439358 58.714283
## 22  45.585033 57.769905
## 23  44.735416 56.826229
```

```
## 24  43.888603 55.884556
## 25  43.042601 54.943572
## 26  42.195946 54.004583
## 27  41.348707 53.064328
## 28  40.501316 52.127022
## 29  39.654052 51.191803
## 30  38.807169 50.254344
## 31  37.961013 49.318519
## 32  37.116799 48.384224
## 33  36.275878 47.452220
## 34  35.439676 46.519692
## 35  34.609546 45.589603
## 36  33.785629 44.660178
## 37  32.966059 43.732528
## 38  32.148869 42.805393
## 39  31.332839 41.881017
## 40  30.517524 40.956773
## 41  29.703698 40.033025
## 42  28.893906 39.111338
## 43  28.090645 38.192120
## 44  27.295463 37.280083
## 45  26.509283 36.371300
## 46  25.732538 35.466665
## 47  24.964834 34.567584
## 48  24.205794 33.671478
## 49  23.455707 32.784520
## 50  22.715118 31.905425
## 51  21.984141 31.034042
## 52  21.261649 30.169792
## 53  20.546193 29.316217
## 54  19.836342 28.471609
## 55  19.130836 27.636060
## 56  18.429592 26.808787
## 57  17.735768 25.988138
## 58  17.053270 25.177065
## 59  16.385805 24.373508
## 60  15.736971 23.579779
## 61  15.108708 22.791231
## 62  14.497826 22.010626
## 63  13.900054 21.238332
## 64  13.311713 20.472119
## 65  12.729466 19.716021
## 66  12.151915 18.962889
## 67  11.583483 18.213813
## 68  11.030136 17.474282
## 69  10.497787 16.745474
## 70   9.992694 16.026543
## 71   9.519023 15.316862
## 72   9.072367 14.620426
## 73   8.645844 13.942241
## 74   8.233184 13.278113
## 75   7.826092 12.627699
## 76   7.414556 11.990523
## 77   7.016866 11.362347
```

```
## 78   6.637047 10.752823
## 79   6.279035 10.157309
## 80   5.945614  9.580387
## 81   5.626863  9.024818
## 82   5.331649  8.486774
## 83   5.057955  7.968045
## 84   4.804028  7.465084
## 85   4.568504  6.986932
## 86   4.349673  6.532340
## 87   4.150186  6.098584
## 88   3.974772  5.684033
## 89   3.826648  5.286357
## 90   3.706287  4.901044
## 91   3.610975  4.541950
## 92   3.534401  4.212051
## 93   3.469224  3.905425
## 94   3.407633  3.625232
## 95   3.344999  3.367656
## 96   3.285874  3.124947
## 97   3.212170  2.931744
## 98   3.088239  2.732745
## 99   2.870801  2.572675
## 100  2.576604  2.417898
```

Solo para mujeres:

```
tabla_de_mortalidad_compara_mujeres<-
  lifetable(USA_mortalidad, series="Female", years=c(1933,2016))

tabla_de_mortalidad_compara_mujeres$ex
```

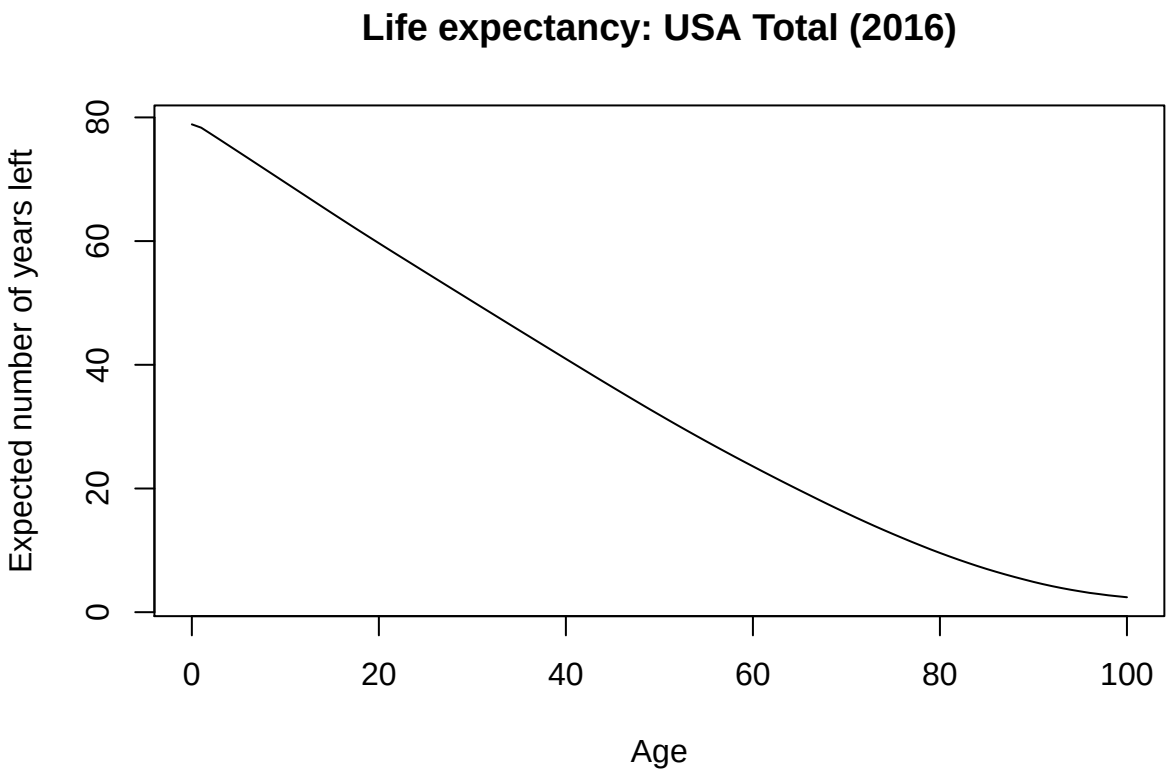
```
##          1933      2016
## 0   62.811055 81.364782
## 1   65.239966 80.798508
## 2   64.816506 79.827100
## 3   64.075902 78.846379
## 4   63.258563 77.859385
## 5   62.398671 76.871222
## 6   61.513414 75.881151
## 7   60.613863 74.889066
## 8   59.702416 73.898068
## 9   58.780615 72.905115
## 10  57.850243 71.912790
## 11  56.913249 70.919432
## 12  55.972965 69.927319
## 13  55.033797 68.934887
## 14  54.099878 67.943647
## 15  53.174541 66.955518
## 16  52.260312 65.968412
## 17  51.357039 64.984454
## 18  50.463900 64.003609
## 19  49.580401 63.027490
## 20  48.706208 62.051693
```

```
## 21  47.840843 61.078228
## 22  46.982699 60.107555
## 23  46.129957 59.138022
## 24  45.280693 58.169695
## 25  44.433027 57.200787
## 26  43.585519 56.234591
## 27  42.737966 55.270105
## 28  41.890467 54.307416
## 29  41.042857 53.345741
## 30  40.194893 52.384491
## 31  39.346618 51.425652
## 32  38.499118 50.470231
## 33  37.653861 49.516375
## 34  36.812362 48.562276
## 35  35.976064 47.610747
## 36  35.145226 46.661230
## 37  34.317841 45.714299
## 38  33.491739 44.767458
## 39  32.665468 43.824821
## 40  31.838241 42.882004
## 41  31.010699 41.941338
## 42  30.185383 41.001389
## 43  29.364957 40.066608
## 44  28.551070 39.136029
## 45  27.744869 38.208888
## 46  26.946845 37.287593
## 47  26.156710 36.370421
## 48  25.374119 35.454456
## 49  24.599450 34.546683
## 50  23.833405 33.647337
## 51  23.076256 32.753280
## 52  22.326996 31.865911
## 53  21.584294 30.985045
## 54  20.846730 30.112862
## 55  20.113029 29.246450
## 56  19.383025 28.386700
## 57  18.659709 27.533182
## 58  17.946759 26.685156
## 59  17.247671 25.843214
## 60  16.565805 25.008227
## 61  15.903125 24.178783
## 62  15.257184 23.351246
## 63  14.624656 22.534191
## 64  14.002724 21.722303
## 65  13.388807 20.918408
## 66  12.781997 20.117988
## 67  12.186119 19.323855
## 68  11.606208 18.542286
## 69  11.047248 17.770713
## 70  10.514513 17.009023
## 71  10.011520 16.258698
## 72   9.534744 15.518973
## 73   9.078685 14.798408
## 74   8.638430 14.093116
```

```
## 75   8.207093 13.400230
## 76   7.775680 12.719677
## 77   7.361473 12.047018
## 78   6.967974 11.396098
## 79   6.598638 10.761938
## 80   6.255613 10.144729
## 81   5.919593  9.551013
## 82   5.606368  8.974904
## 83   5.313547  8.420648
## 84   5.039102  7.881476
## 85   4.781830  7.365294
## 86   4.540413  6.878542
## 87   4.317903  6.409451
## 88   4.119429  5.962340
## 89   3.948101  5.537632
## 90   3.804875  5.127217
## 91   3.689143  4.745027
## 92   3.597437  4.394854
## 93   3.526981  4.067653
## 94   3.476438  3.766602
## 95   3.441207  3.491186
## 96   3.410900  3.231483
## 97   3.359084  3.021537
## 98   3.250376  2.813335
## 99   3.052195  2.638580
## 100  2.781704  2.472343
```

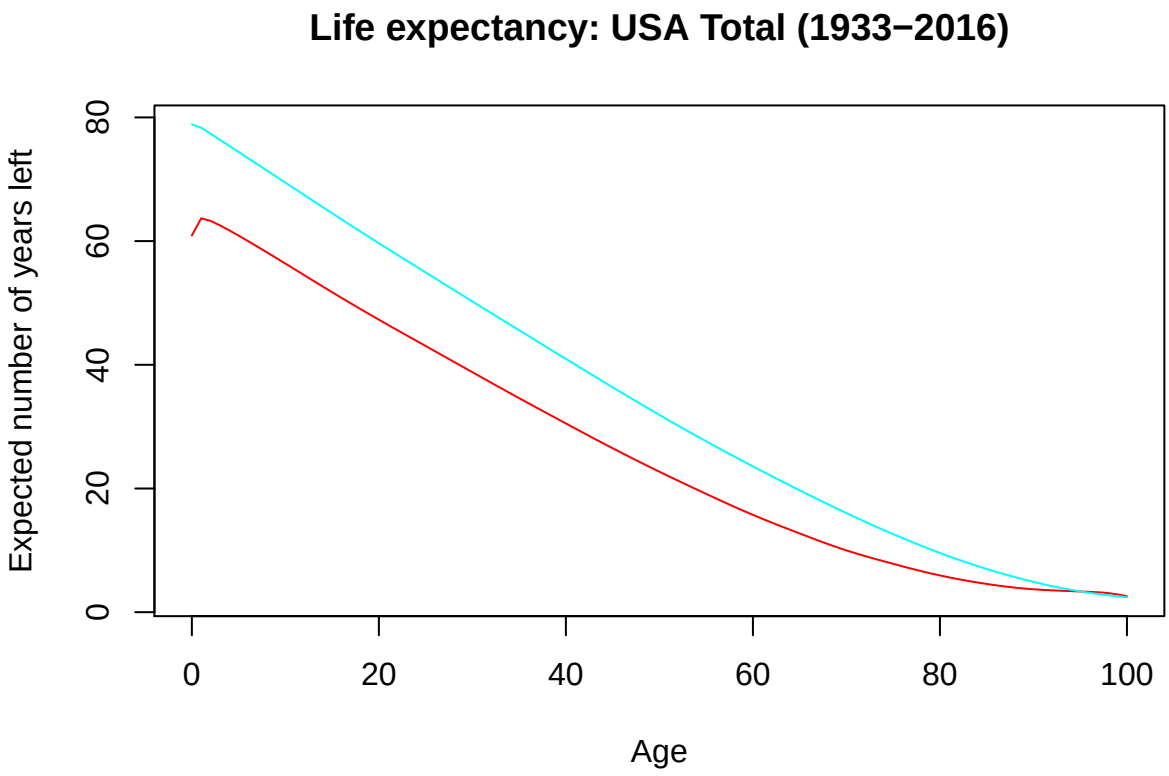
La visualización de estos resultados se logra con el comando `plot`, que, por defecto grafica la esperanza de vida por edad:

```
plot(tabla_de_mortalidad)
```



Al que se pueden ajustar título y nombres de los ejes.
No hay inconveniente en graficar más de un año:

```
plot(tabla_de_mortalidad_compara)
```



El comando `life.expectancy` provee esperanzas de vida para determinadas edades y años; por defecto toma edad cero años y todos los años disponibles. Por ejemplo:

```
ev_nac<-
  life.expectancy(USA_mortalidad, series = "Total")

ev_nac
```

```
## Time Series:
## Start = 1933
## End = 2017
## Frequency = 1
##     1933     1934     1935     1936     1937     1938     1939     1940
## 60.90690 60.19977 60.91639 60.36011 61.04395 62.39958 63.07703 63.22324
##     1941     1942     1943     1944     1945     1946     1947     1948
## 63.79659 64.54374 64.36603 65.06795 65.59747 66.13765 66.81310 67.22941
##     1949     1950     1951     1952     1953     1954     1955     1956
## 67.63678 68.06699 68.17809 68.38618 68.72044 69.50739 69.56082 69.64528
##     1957     1958     1959     1960     1961     1962     1963     1964
## 69.41628 69.66965 69.90069 69.82200 70.24846 70.11187 69.93943 70.19788
##     1965     1966     1967     1968     1969     1970     1971     1972
## 70.24969 70.21302 70.53020 70.21159 70.48781 70.74541 71.10667 71.18100
##     1973     1974     1975     1976     1977     1978     1979     1980
## 71.40062 71.96307 72.54790 72.84992 73.23365 73.41519 73.83803 73.73646
##     1981     1982     1983     1984     1985     1986     1987     1988
## 74.12391 74.47290 74.56344 74.68577 74.66628 74.74541 74.87681 74.85322
```

```
##     1989     1990     1991     1992     1993     1994     1995     1996
## 75.12774 75.39754 75.54871 75.80521 75.59326 75.76505 75.87976 76.20107
##     1997     1998     1999     2000     2001     2002     2003     2004
## 76.52696 76.68826 76.69989 76.83992 76.95717 77.03058 77.18219 77.59121
##     2005     2006     2007     2008     2009     2010     2011     2012
## 77.59102 77.87434 78.13842 78.22962 78.59583 78.80024 78.84706 78.96966
##     2013     2014     2015     2016     2017
## 78.97789 79.05115 78.89756 78.87574 78.85020
```

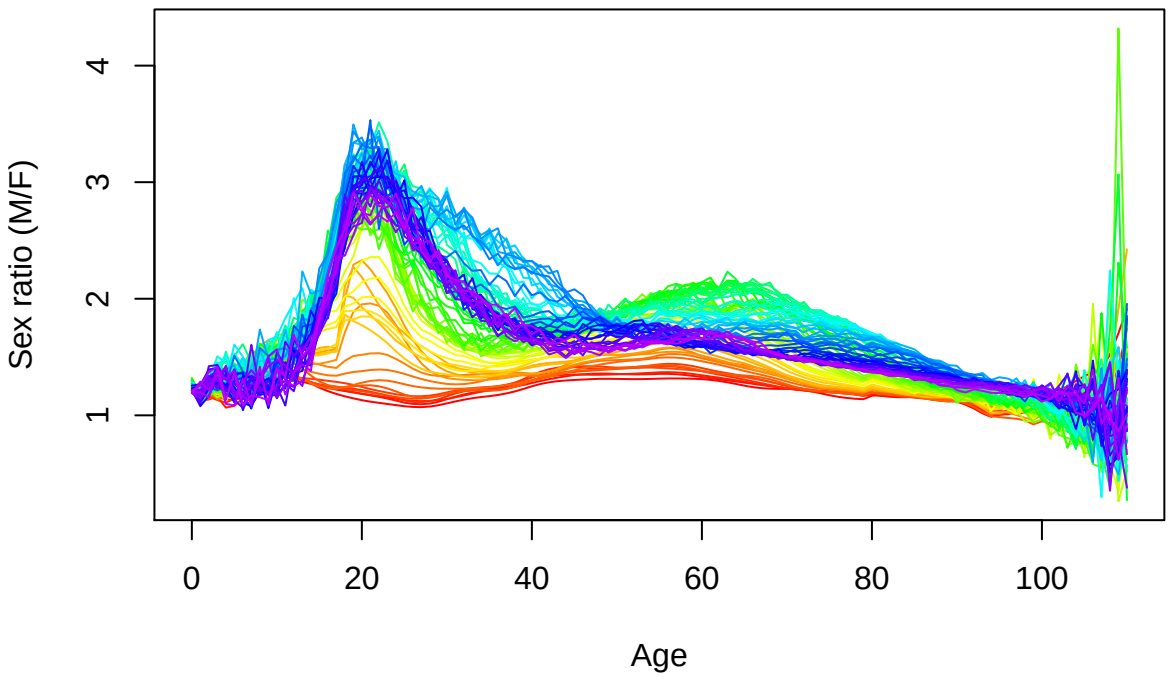
Devuelve la esperanza de vida al nacimiento para cada año que hay en la base. Si se quiere algo más específico, como la esperanza de vida a los 25 años de mujeres en 1945:

```
espe_vida_mujeres_25_1945<-
  life.expectancy(USA_mortalidad, series = "Female",
                  age = 25, years = 1945)

espe_vida_mujeres_25_1945
```

```
## Time Series:
## Start = 1945
## End = 1945
## Frequency = 1
## [1] 47.56695
```

La función `sexratio` muestra las diferencias en la mortalidad a diferentes edades:

```
IM<-
  sex.ratio(USA_mortalidad)

plot(IM)
```



Que muestra el índice de masculinidad de las tasas de mortalidad para diferentes edades, donde cada línea representa una cohorte.

Modelización de Lee Carter

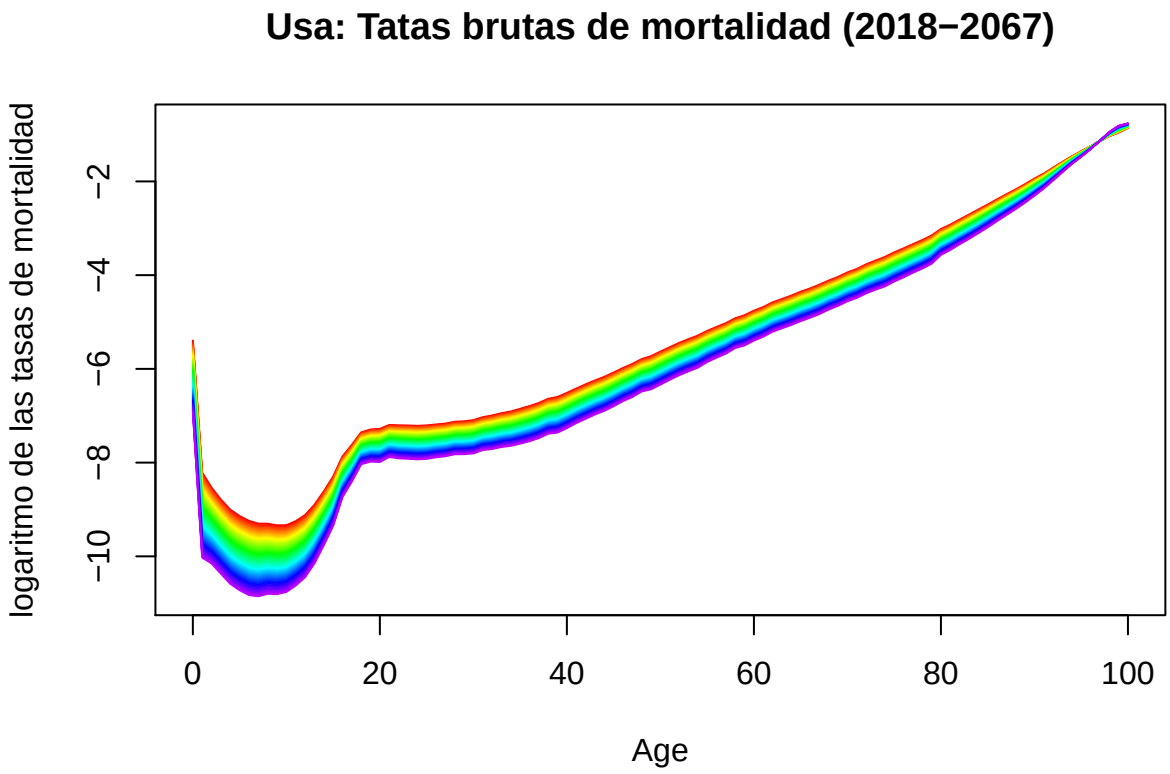
Permite predecir la mortalidad o la esperanza de vida, lo interesante es que este paquete tiene al modelo como función incorporada. Para predecir, primero hay que convertir los datos al modelo.

Vamos a examinar las tasas de mortalidad totales y luego proyectarlas. El comando `lca` requiere de un objeto `demogdata` como input, la mayoría de los otros componentes son similares a los anteriores. El comando `forecast.lca`usa el resultado de `lca` y ofrece una proyección por el número de años indicados en h. Los argumentos `se` y `jumpchoice` se refieren a métodos de computación que pueden verse con más detalle en la página del paquete (https://www.rdocumentation.org/packages/demography/versions/1.22). El nivel de confianza para los intervalos de predicción se indica en el argumento `level`.

```
lca2<-lca(USA_mortalidad,
          series=names(USA_mortalidad$rate)[3],
          years=USA_mortalidad$year,
         ages=USA_mortalidad$age,max.age=100)


proyeccion.1<-forecast(
  lca2, h=50, se=c("innovdrift"),
  jumpchoice=c("fit"), level=90)
```

If we plot our forecasted data we see a similar plotting mechanism as to our initial data plot.

```r
plot(proyeccion.1,
     main="Usa: Tatas brutas de mortalidad (2018-2067)",
     xlab="edad",
     ylab="logaritmo de las tasas de mortalidad")
```




```r
summary(lca2)
```

```
## Lee-Carter analysis
##
## Call: lca(data = USA_mortalidad, series = names(USA_mortalidad$rate)[3],
##
## Call:     years = USA_mortalidad$year, ages = USA_mortalidad$age, max.age = 100)
##
## Adjustment method: dt
## Region: USA
## Years in fit: 1933 - 2017
## Ages in fit: 0 - 100
##
## Percentage variation explained: 96.1%
##
## ERROR MEASURES BASED ON MORTALITY RATES
##
## Averages across ages:
##       ME      MSE      MPE     MAPE
## -0.00001  0.00005  0.00763  0.07028
```

```
##
## Averages across years:
##      IE     ISE     IPE    IAPE
## 0.00013 0.00384 0.76442 6.98597
##
##
## ERROR MEASURES BASED ON LOG MORTALITY RATES
##
## Averages across ages:
##       ME      MSE      MPE     MAPE
##  0.00278  0.00973 -0.00012  0.01668
##
## Averages across years:
##       IE      ISE      IPE     IAPE
##  0.27775  0.96634 -0.01713  1.61878
```

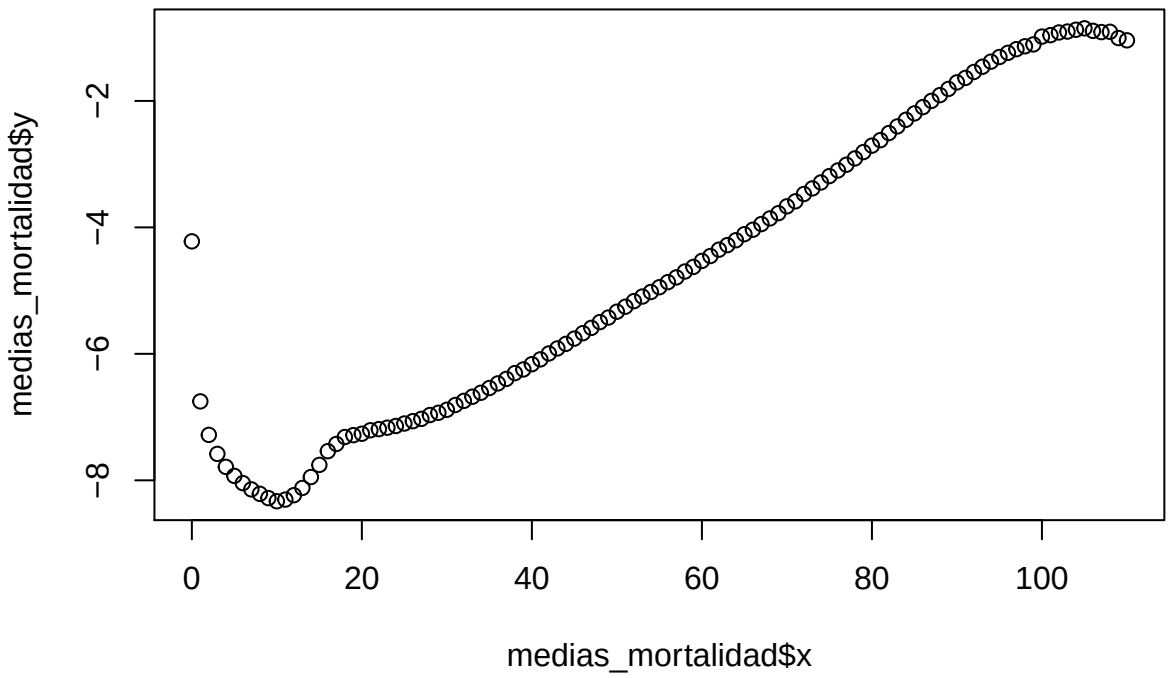
Otros comandos

mean.demogdata calcula la media o mediana de tasas en cada nivel de edad. Los argumentos permiten que se transformen o no los datos antes del cálculo
Para las tasas promedio de mortalidad por edad

```
medias_mortalidad<-mean(USA_mortalidad)
```

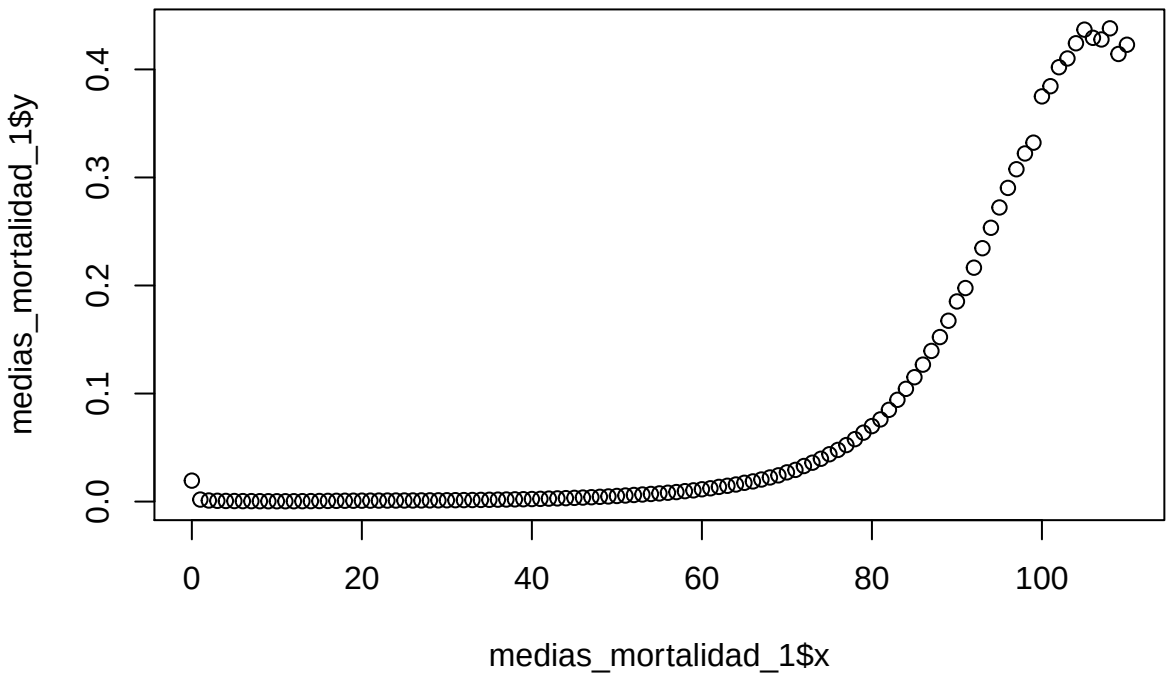
Su gráfico es

```
plot(medias_mortalidad)
```



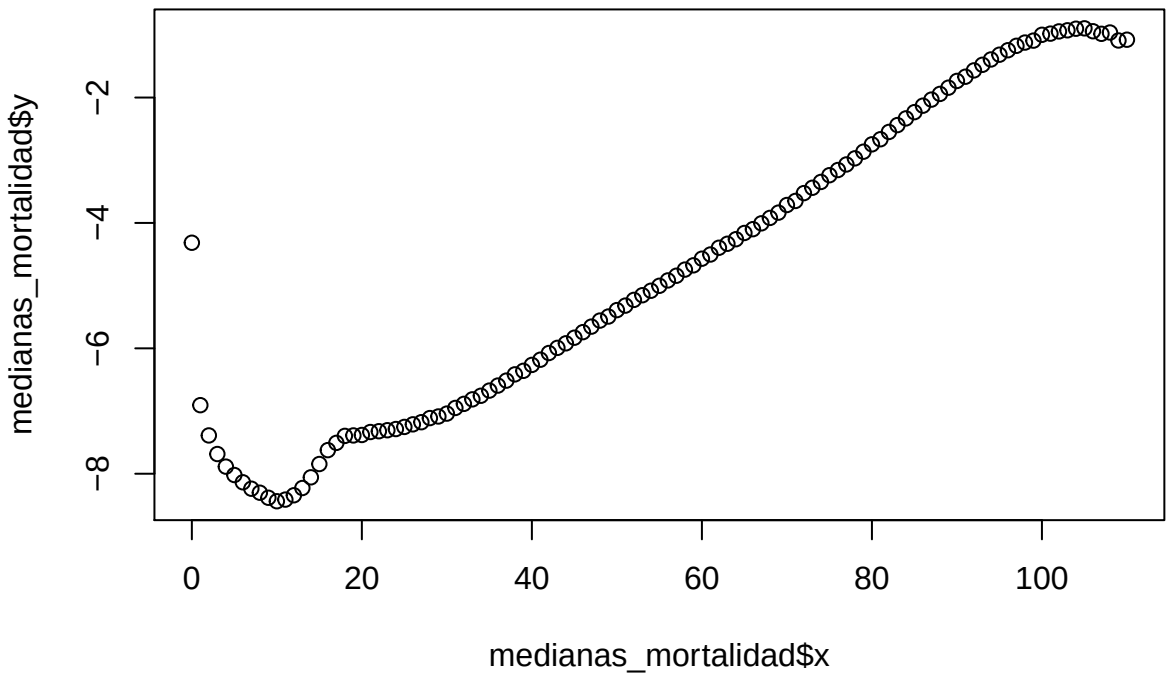
Si se quiere evitar la transformación logarítmica

```
medias_mortalidad_1<-mean(USA_mortalidad, transform = FALSE)
plot(medias_mortalidad_1)
```



Lo mismo para las medianas

```
medianas_mortalidad<-median(USA_mortalidad)
plot(medianas_mortalidad)
```

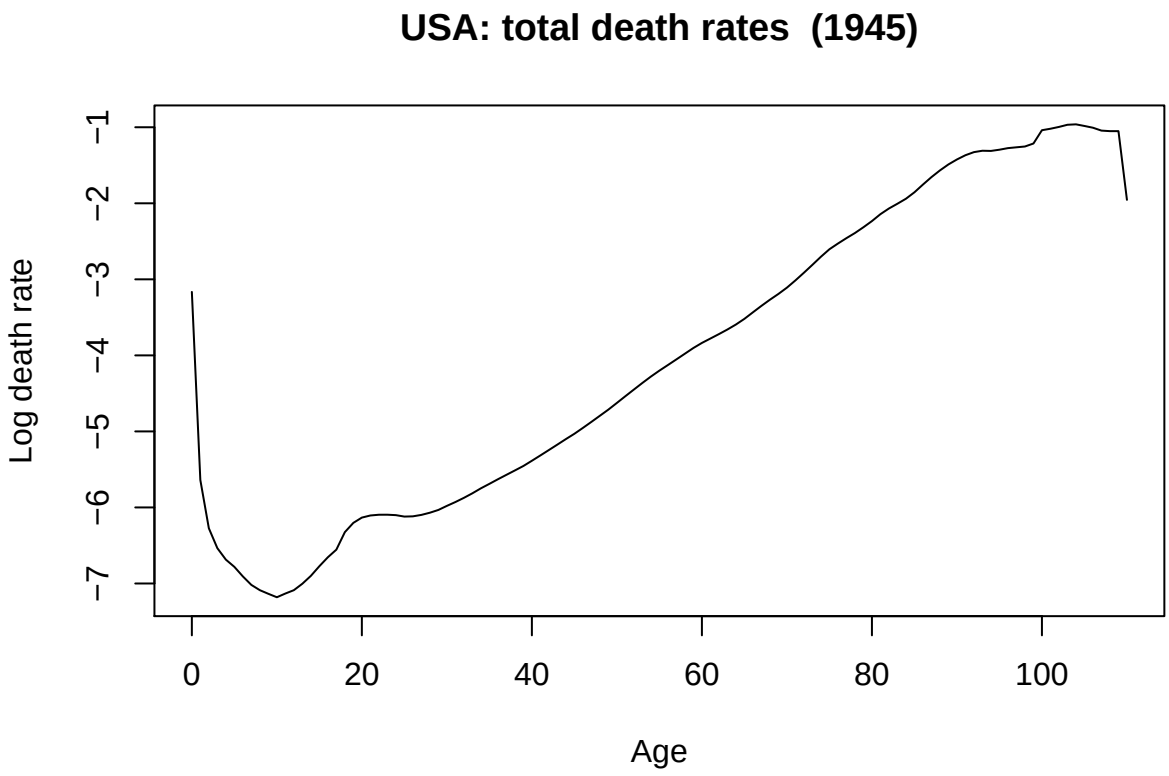


Para verlas como matriz de datos, se lo transforma y se ponen nombres a las columnas:

```
medias_mortalidad_df<-as.data.frame(medias_mortalidad)

names(medias_mortalidad_df)<-
  c("edad", "tasa media de mortalidad")
```

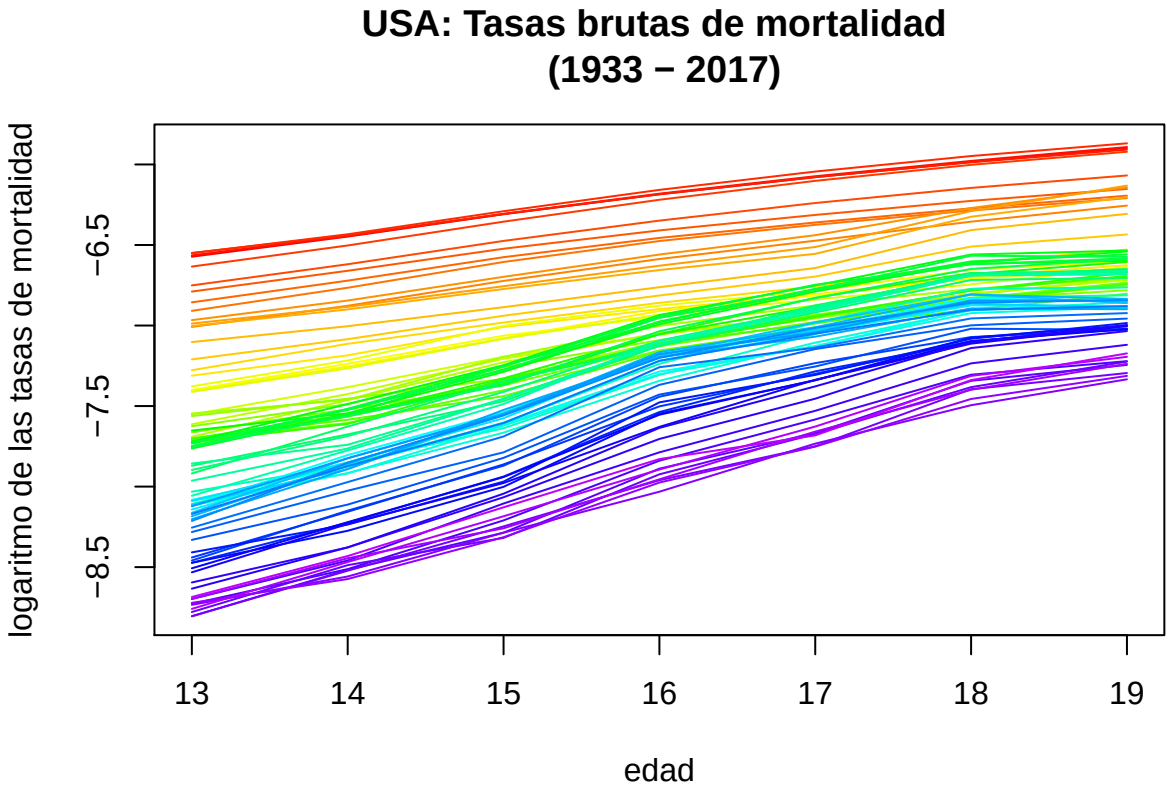
extract.years extrae de un objeto `demogdata`, un vector de valores para un año específico o para una serie de años y crea un nuevo objeto 'demogdata. Crea un subconjunto con algunos años.
Las tasas para 1945:

```
mortalidad_1945<-
  extract.years(USA_mortalidad, 1945)

plot(mortalidad_1945, series = "Total")
```



extract.ages coo el anterior para las edades, crea un subconjunto con algunas edades. Si el último argumento es `TRUE`, todas la edades más allá de la buscada, se combinan en el último grupo de edades
Por ejemplo, para las tasas de personas entre 13 y 19 años:

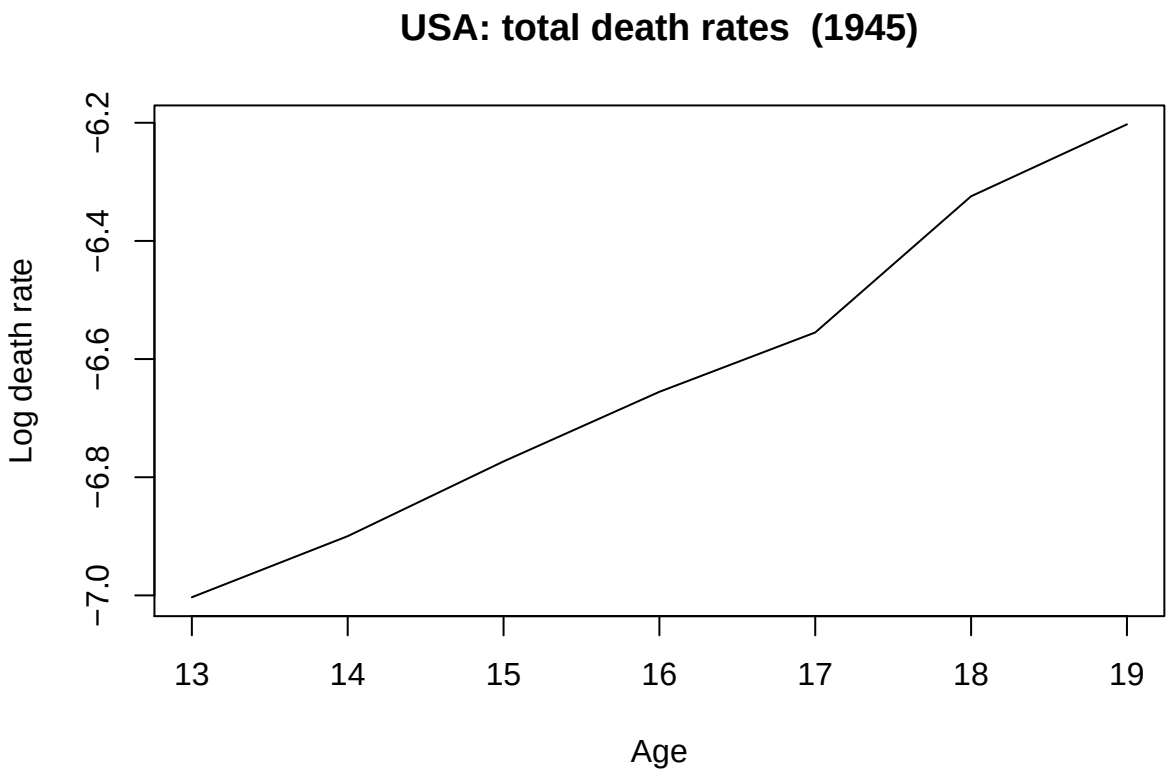
```
mortalidad_jovenes<-
  extract.ages(USA_mortalidad, 13:19,
               combine.upper = FALSE)

plot(mortalidad_jovenes, series = "Total",
     main = "USA: Tasas brutas de mortalidad \n (1933 - 2017)",
     ylab = "logaritmo de las tasas de mortalidad",
     xlab = "edad")
```



Si no se indica que `combine.upper=FALSE`, por defecto, colapsa en 19 años todas las edades superiores a ella.

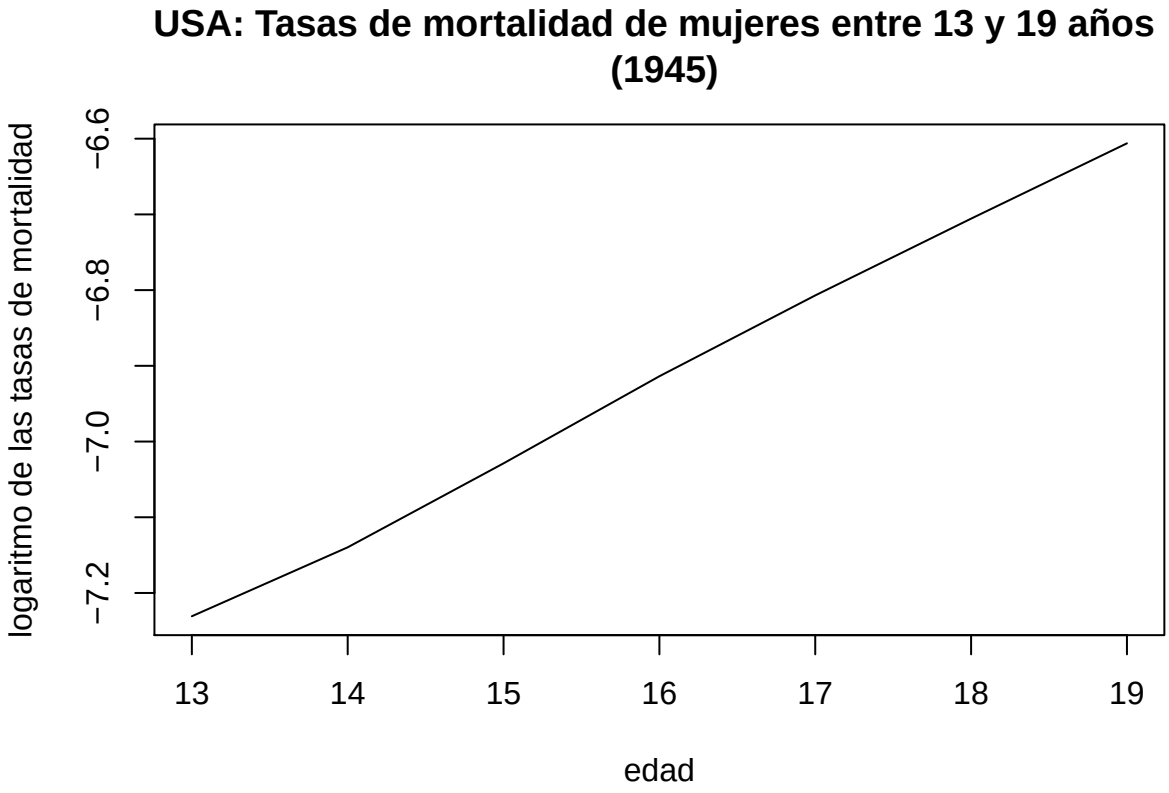
La combinación de los dos comandos permite seleccionar años y edades simultáneamente:

```
mortalidad_jovenes_1945<-
  extract.ages(mortalidad_1945, 13:19,
               combine.upper = FALSE)

plot(mortalidad_jovenes_1945,
     series = "Total")
```



Solo las mujeres

```
plot(mortalidad_jovenes_1945,
     series = "Female",
     main = "USA: Tasas de mortalidad de mujeres entre 13 y 19 años \n (1945)",
     ylab = "logaritmo de las tasas de mortalidad", xlab = "edad")
```



Recursos

Ayuda general

https://support.rstudio.com/hc/en-us/articles/200552336-Getting-Help-with-R
https://r4ds.had.co.nz/
https://es.r4ds.hadley.nz/

Base de datos de mortalidad:

Human Mortality Database. University of California, Berkeley (USA), and Max Planck Institute for Demographic Research (Germany). Available at www.mortality.org or www.humanmortality.de

Paquete `demography`: https://www.rdocumentation.org/packages/demography/versions/1.22/topics/demogdata

https://rpubs.com/Timexpo/487053

Paquete `ggplot2`

https://www.r-graph-gallery.com/ggplot2-package.html
https://www.statmethods.net/advgraphs/ggplot2.html?ref=driverlayer.com/web
https://ggplot2.tidyverse.org/

Paquete `viridis`

https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html https://ggplot2.tidyverse.org/reference/scale_viridis.html

---

Referencias

Ihaka, Ross, and Robert Gentleman. 1996. "R: a language for data analysis and graphics." *Journal of Computational and Graphical Statistics* 5 (3): 299–314.

INDEC. 2003. "EPH-Cambios Metodologicos 2003." Buenos Aires: INDEC.

———. 2011. "Encuesta Permanente de Hogares Conceptos de Condición de Actividad, Subocupación Horaria y Categoría Ocupacional." Buenos Aires: INDEC.

RStudio Team. 2018. *RStudio: Integrated Development Environment for R*. Boston, MA: RStudio, Inc. http://www.rstudio.com/.

Shkolnikov, Vladimir, Magali Barbieri, and John Wilmoth. 2020. "Human Mortality Database." https://www.mortality.org/.

Wickham, Hadley. 2009. *Ggplot2: Elegant Graphics for Data Analysis*. New York: Springer New York. http://had.co.nz/ggplot2/book.