notas

Eduardo Bologna

6/3/2020

Presentación

Este curso es una introducción a R dirigido a investigadores en Ciencias Sociales y busca ir más allá de la barrera de entrada
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. De la larga lista de opciones disponibles, este manual usa un software que se llama R, elección que 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 [@Ross1996], 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 una intensa comunidad de usuarios de R on-line, 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.

Actualmente, los principales medios de comunicación usan R para expresar datos de manera gráfica.

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 [@RStudioTeam2018] 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 puede encontrar en https://blog.revolutionanalytics.com/2017/10/updated-history-of-r.html

Este curso busca proveer una primera aproximación a este entorno para quienes tienentengan o no experiencia en otros programas de análisis de datos y cuenten con una base conceptual de estadística.

Datos

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) y las Direcciones Provinciales de EstadÌstica (DPE) (@INDEC2003). 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.

Instalación 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.

Los componentes de RStudio

Cuando abrimos RStudio, vamos a encontrar 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:

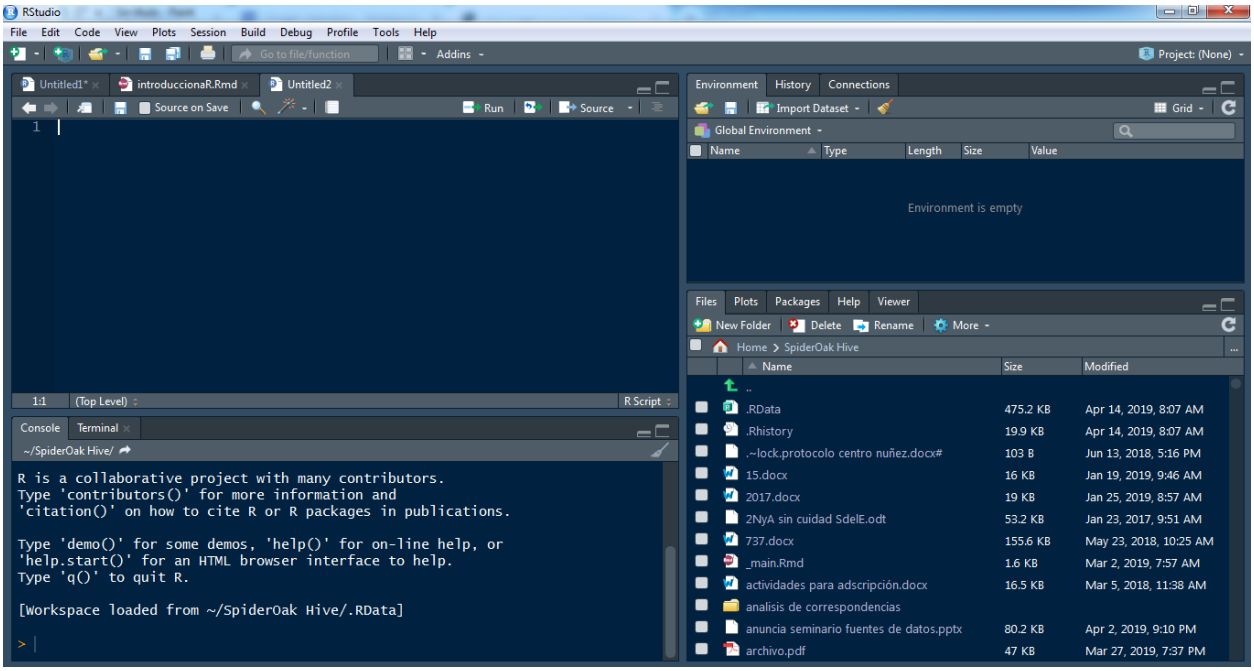


Figure 1: Los cuatro paneles de RStudio

- 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 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")
```

Estas instalaciones se realizan por única vez en cada computadora y, 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.

Escfibir y ejecutar comandos

Antes de empezar a operar es necesario crear un lugar donde se alojarán los archivos que vamos 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.

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

Constante

```
1+2
```

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

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.

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

```
x
```

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

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:

```r
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 cualidad 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 vectores de caracteres se guarda cada texto como un valor, por lo que consume más memoria. Por defecto, los valores textuales son tomados como ed clase caracer. Sea x el valor "a":

```r
x<-"a"
class(x) # carácter
```

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

```r
# Transformado a numérico:
x<-as.numeric(x) # produce NA
```

```
## Warning: NAs introducidos por coerción
```

```r
x
```

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

Si se lo define como factor:

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

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

```r
# llevado a numérico:
x<-as.numeric(x)
x
```

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

Le asigna el número 1.

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

```r
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:

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

Y no veremos su valor hasta que no lo solicitemos

```r
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:

```r
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
```

```r
## o todo de una sola vez:
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:

```
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.

```
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:

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

```
##  [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:

```
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:

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

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

O el valor "perro", tres veces:

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

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

Estas maneras de generar secuencias pueden combinarse:

```
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:

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

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

```
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:

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

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

```
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:

```
levels(grupo)
```

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

Y también definirse, como etiquetas:

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

Ahora éstos son los nuevos niveles:

```
levels(grupo)
```

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

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

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

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

```
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:

```r
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:

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

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

O si uno dividido cero es infinito:

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

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

Estos comandos da un resultado 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:

```r
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 varables. 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:

```r
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:

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

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

Y se los puede redefinir:

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

Ahora la matriz se ve:

```r
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:

```r
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 pultimo es necesario, orque de los 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 guerdada 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:

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

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

```r
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:

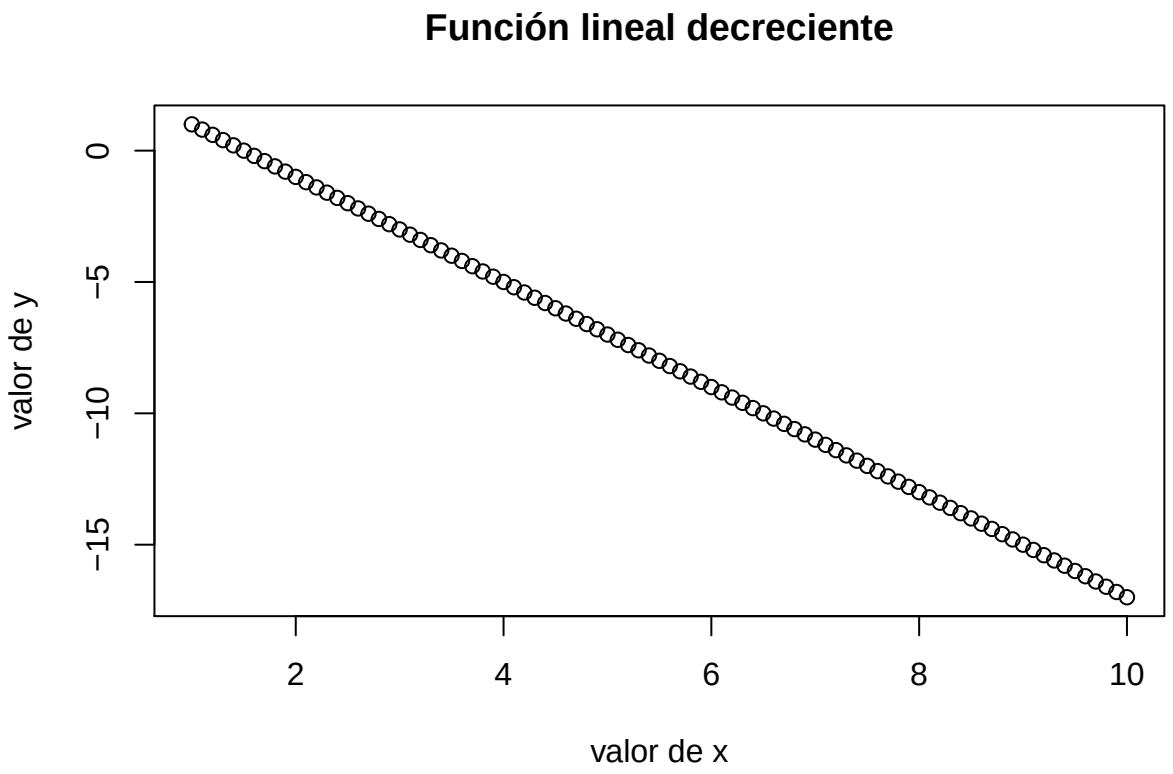
- El nombre del archivo a leer junto con la ruta para llegar a él, muy conveniente que esté en el directorio de trabajo.
- El separador de campos: suelen ser comas, punto y comas, tabulaciones, espacio en blanco o pipe | (alt124 en windows). Si los campos están separados por comas, se indica $sep =$ ", " 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.

Graficar

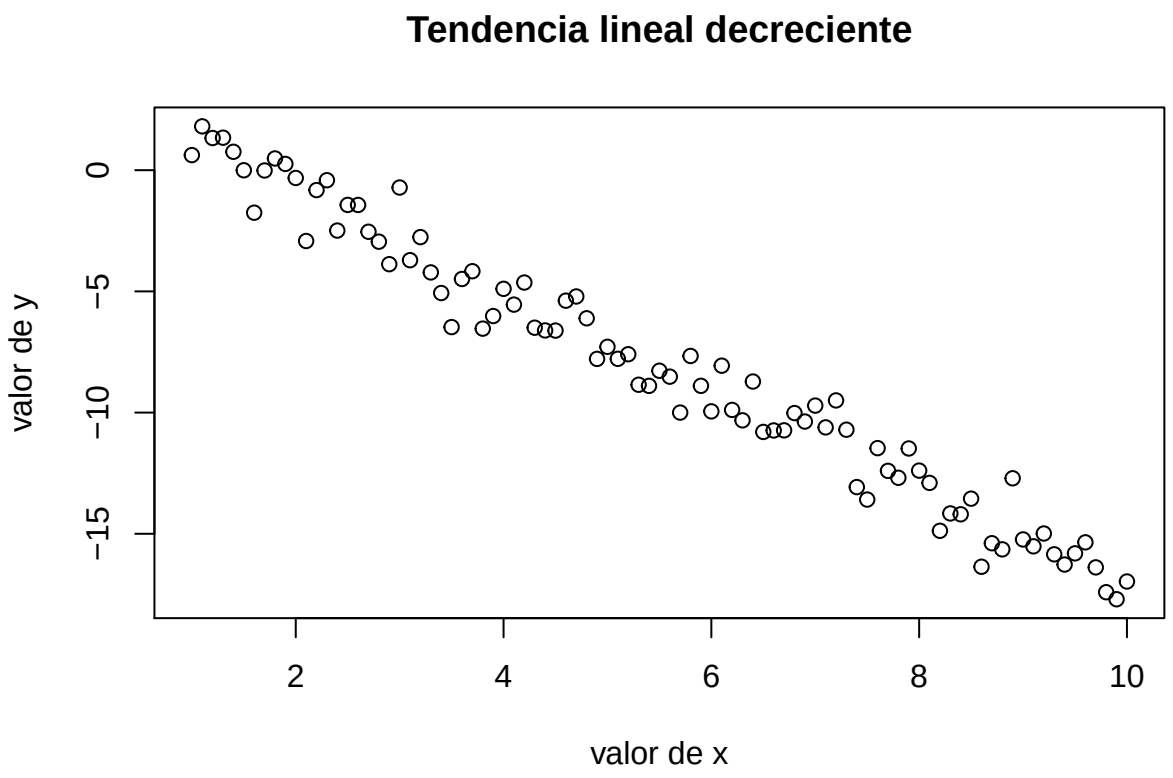
Para ver un ejemplo de gráfico básico, puede copiar y pegar el siguiente código en su script. Allí se define a x como una secuencia de números que va de 1 a 10 en intervalos de 0.1. Luego se define y como una función lineal de x ($y = 3 - 2 * x$). El tercer comando indica que se grafiquen las dos variables y les pone nombre a los ejes y al título del gráfico.

```
x <- seq(1, 10, .1)
y <- 3 - 2 * x
plot(x, y,
  xlab = "valor de x", ylab = "valor de y",
  main = "Función lineal decreciente"
)
```



A la función se puede agregar una perturbación aleatoria, en este caso se suman 91 casos (que la cantidad de valores que tiene x) provenientes de una distribución normal con media uno y desviación estándar cero:

```
y <- 3 - 2 * x + rnorm(91)
plot(x, y,
  xlab = "valor de x", ylab = "valor de y",
  main = "Tendencia lineal decreciente"
)
```



Resumen de datos

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:

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

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

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

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

```r
addmargins(table(eph.3.18$ESTADO))
```

```
##
##    ocupade desocupade   inactive        Sum
##      23398       1870      23167      48435
```

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

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

```r
# Frecuencias relativas multiplicadas por 100 y
# redondeadas a 2 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:

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

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

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

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

```r
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 la clase de variable de que se trata y ofrece un resumen. En el caso de facatores, el resumen es solo 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 tabñas ponderadas es *wtd.table* y requiere que se indique el vector de 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
```

El primer análisis sobre una tabla bivariada es una prueba de independencia:

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

```
##
##  Pearson's Chi-squared test
##
## data:  sexo_por_estado
## X-squared = 1603.9, df = 2, p-value < 2.2e-16
```

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
```

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 ocn 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
```

```
# ahora 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")
```

Si más tarde la queremos usar como numérica (con los números que 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
```

Vamos a usar estas expresiones para corregir la codificación de la edad, que en la EPH se asigna a las personas menores de un año, el valor -1, cuando sería más adecuado 0. Para reasignar, pedimos que a la variable le asigne el valor 0 allí donde ésta 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$CH06[eph.3.18$CH06==-1]<-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, 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)
```

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 es piden por su nombre

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

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

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

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

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

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

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

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

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

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

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

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

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

O construirse una función que lo calcule:

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

Y aplicarla:

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

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

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

```r
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

```r
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.

Intensidad de las asociaciones

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:

```r
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:

```r
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:

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

```
## Warning in cor.test.default(asalariados.con.ingreso.cordoba
## $educacion_numerica, : Cannot compute exact p-value with ties
```

```
##
##  Spearman's rank correlation rho
##
## data:  asalariados.con.ingreso.cordoba$educacion_numerica and asalariados.con.ingreso.cordoba$PP08D1
## S = 44283000, p-value < 2.2e-16
## alternative hypothesis: true rho is not equal to 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$CH06,
    asalariados.con.ingreso.cordoba$PP08D1)
```

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

La prueba se pide igual

```r
cor.test(asalariados.con.ingreso.cordoba$CH06,
    asalariados.con.ingreso.cordoba$PP08D1)
```

```
##
##  Pearson's product-moment correlation
##
## data:  asalariados.con.ingreso.cordoba$CH06 and asalariados.con.ingreso.cordoba$PP08D1
## t = 7.8427, df = 752, p-value = 1.512e-14
## 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.

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. El modelo es:

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

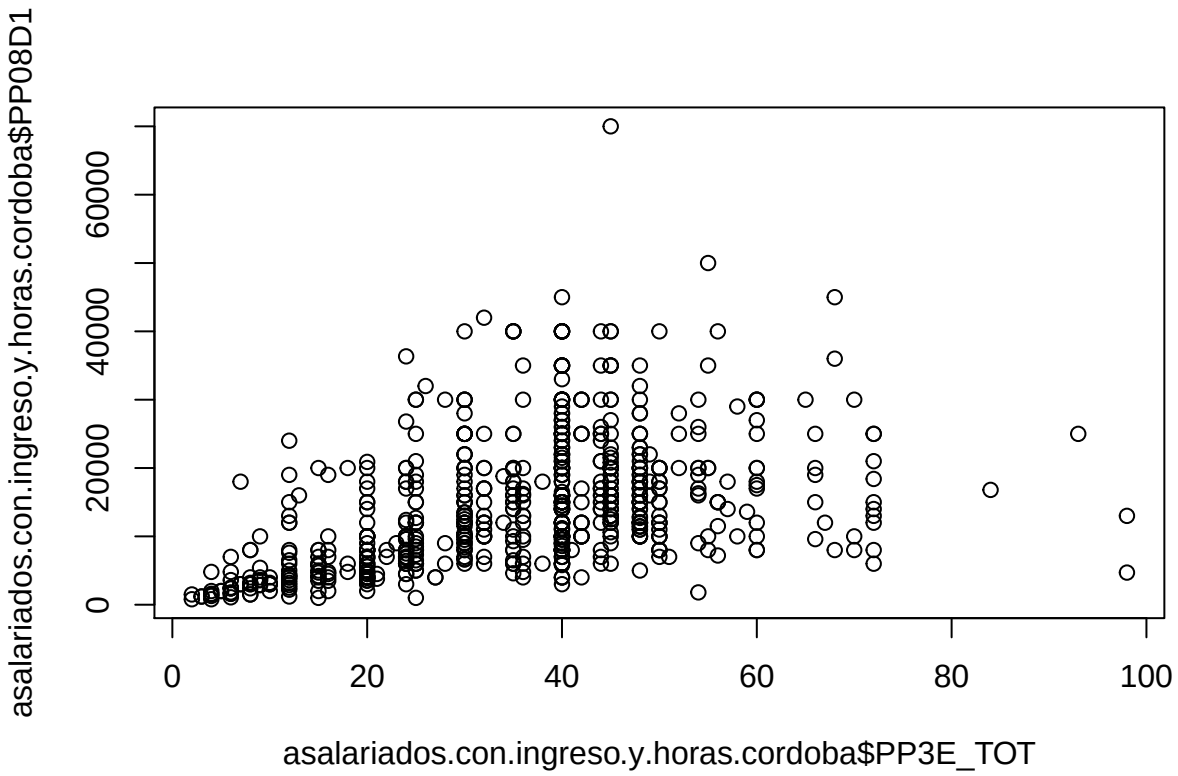
Ahora hay que verlo:

```
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 1.71e-10 ***
## PP3E_TOT      265.91      20.85  12.754  < 2e-16 ***
## ---
## 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: < 2.2e-16
```

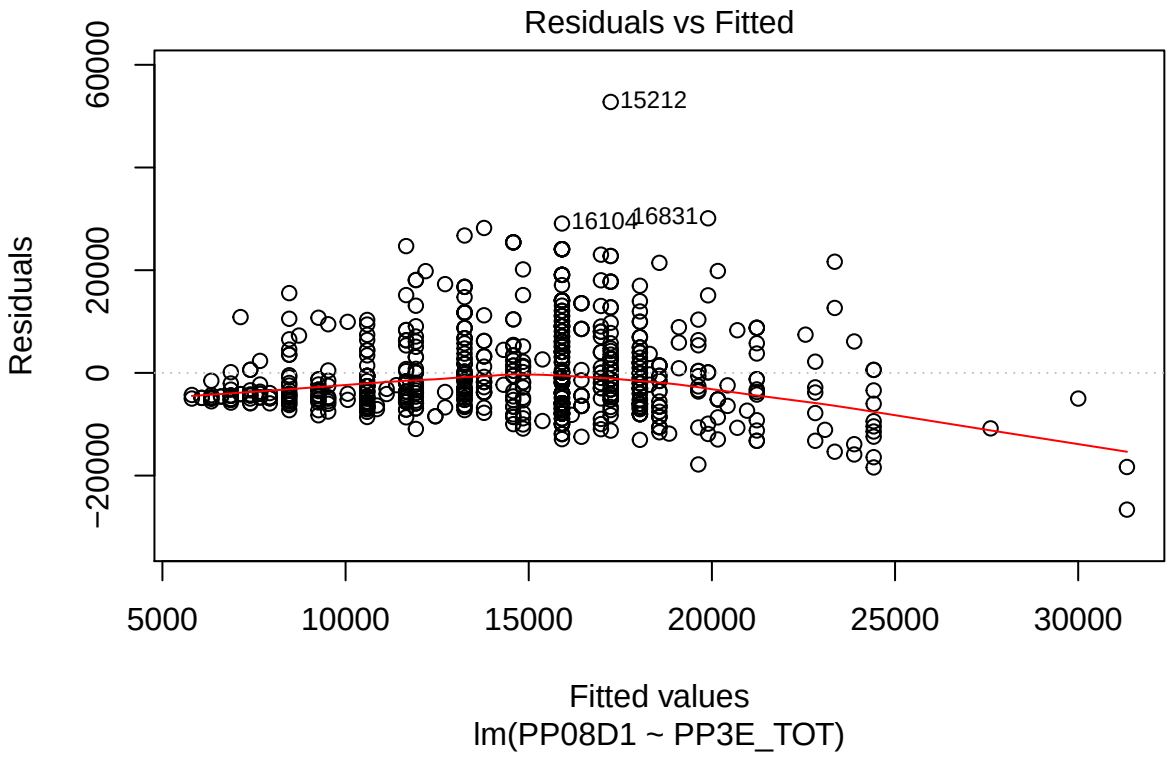
La visualización de la nube de puntos es:

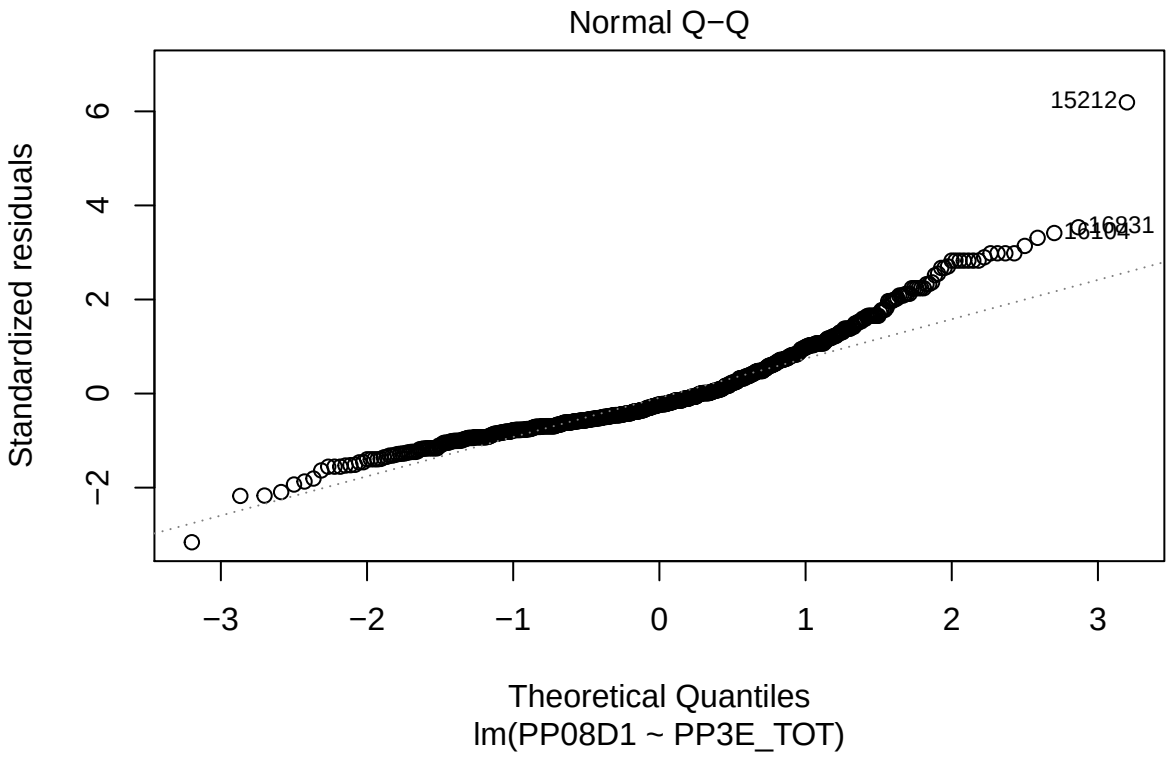
```
plot(asalariados.con.ingreso.y.horas.cordoba$PP3E_TOT,
     asalariados.con.ingreso.y.horas.cordoba$PP08D1)
```

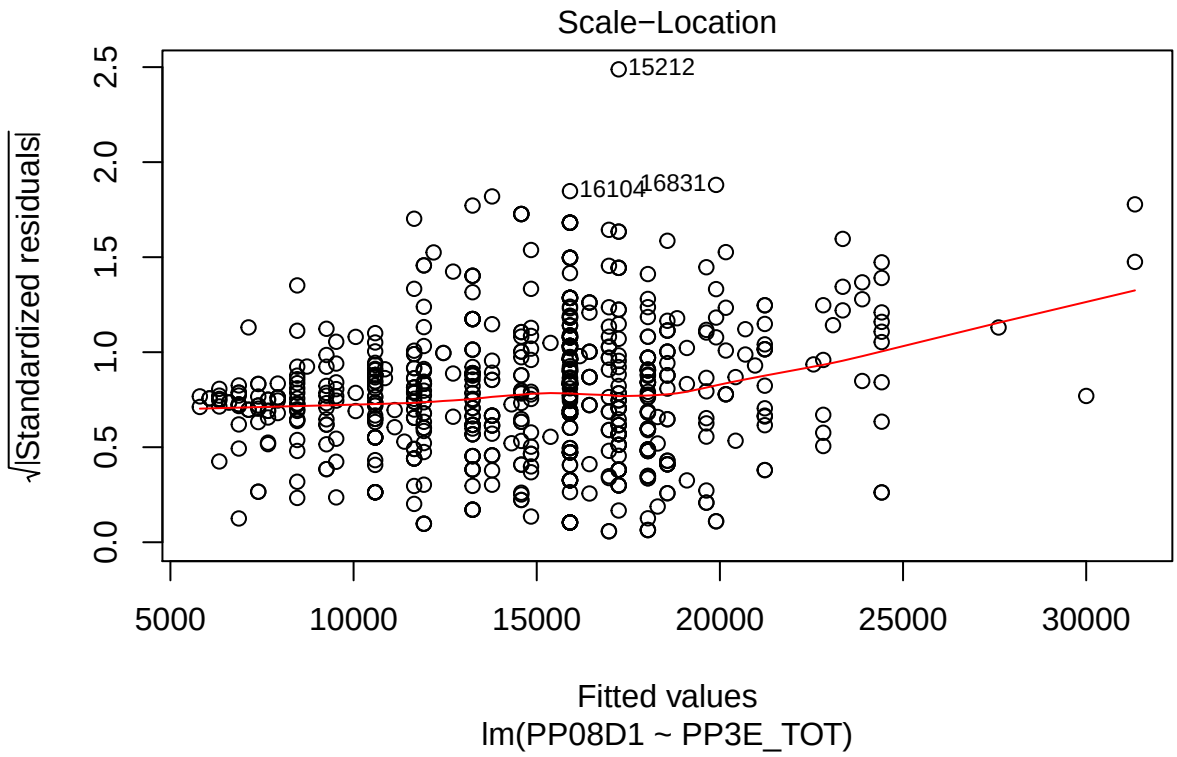


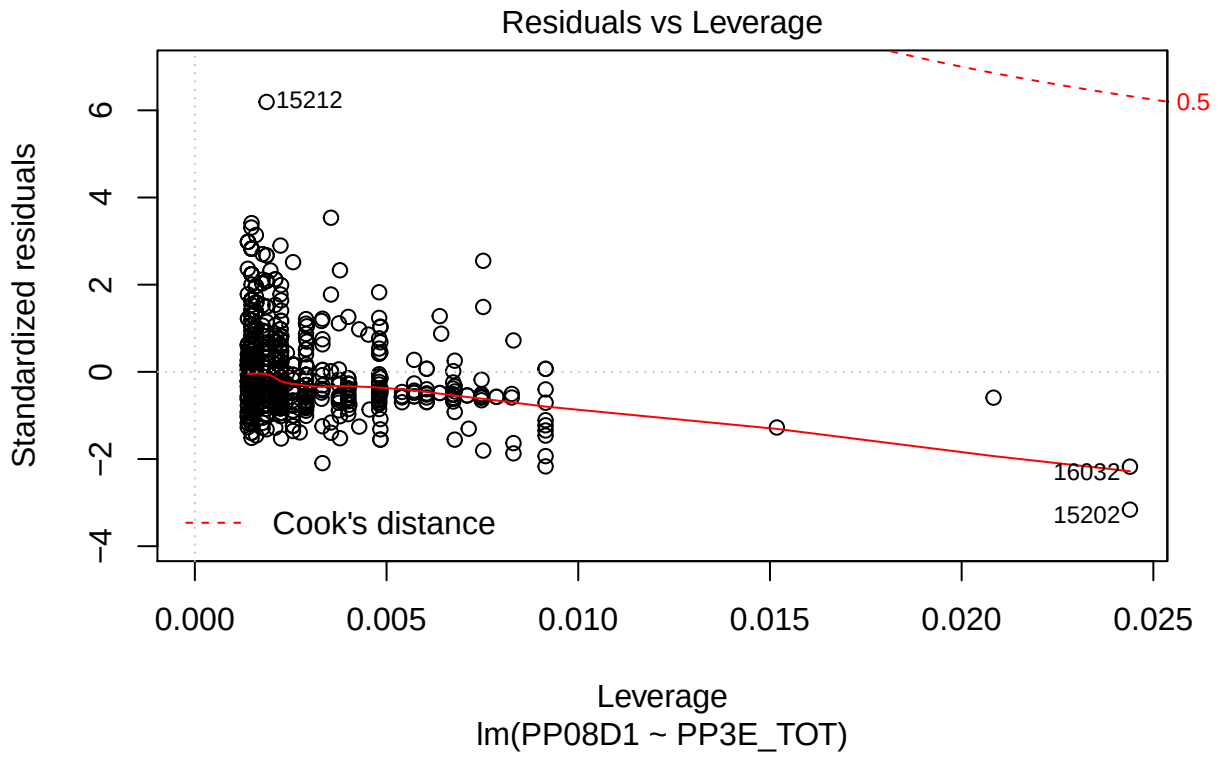
Y los gráficos asociados al modelo:

```
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
16104
16931
−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+CH06,
             data = asalariados.con.ingreso.y.horas.cordoba)
summary(modelo.2)
```

```
##
## Call:
## lm(formula = PP08D1 ~ PP3E_TOT + CH06, 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  < 2e-16 ***
## CH06          177.75      25.10   7.081 3.41e-12 ***
## ---
## 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: < 2.2e-16
```

La comparación por sexos puede hacerse incluyéndola como variable dummy:

```
modelo.3<-lm(PP08D1~PP3E_TOT+CH06+sexo,
             data = asalariados.con.ingreso.y.horas.cordoba)

summary(modelo.3)
```

```
##
## Call:
## lm(formula = PP08D1 ~ PP3E_TOT + CH06 + 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  < 2e-16 ***
## CH06          186.72      25.01   7.465 2.41e-13 ***
## 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: < 2.2e-16
```

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

```
modelo.4<-lm(PP08D1~PP3E_TOT+CH06+sexo+CH06*sexo,
             data = asalariados.con.ingreso.y.horas.cordoba)

summary(modelo.4)
```

```
##
## Call:
## lm(formula = PP08D1 ~ PP3E_TOT + CH06 + sexo + CH06 * 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  < 2e-16 ***
## CH06             243.87      33.73   7.229 1.24e-12 ***
## sexomujer       2310.64    1966.45   1.175   0.2404
## CH06: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: < 2.2e-16
```

Comparación de grupos

Para comparar los ingresos salariales de mujeres y varones, se requiere que los grupos sean independinetes, 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":

```
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 = 6.672e-07
## 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 = 3.336e-07
## 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. Lo cargamos en la sesión actual:

```
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 3.5.3
```

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

Ingresos salariales (PP08D1)

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

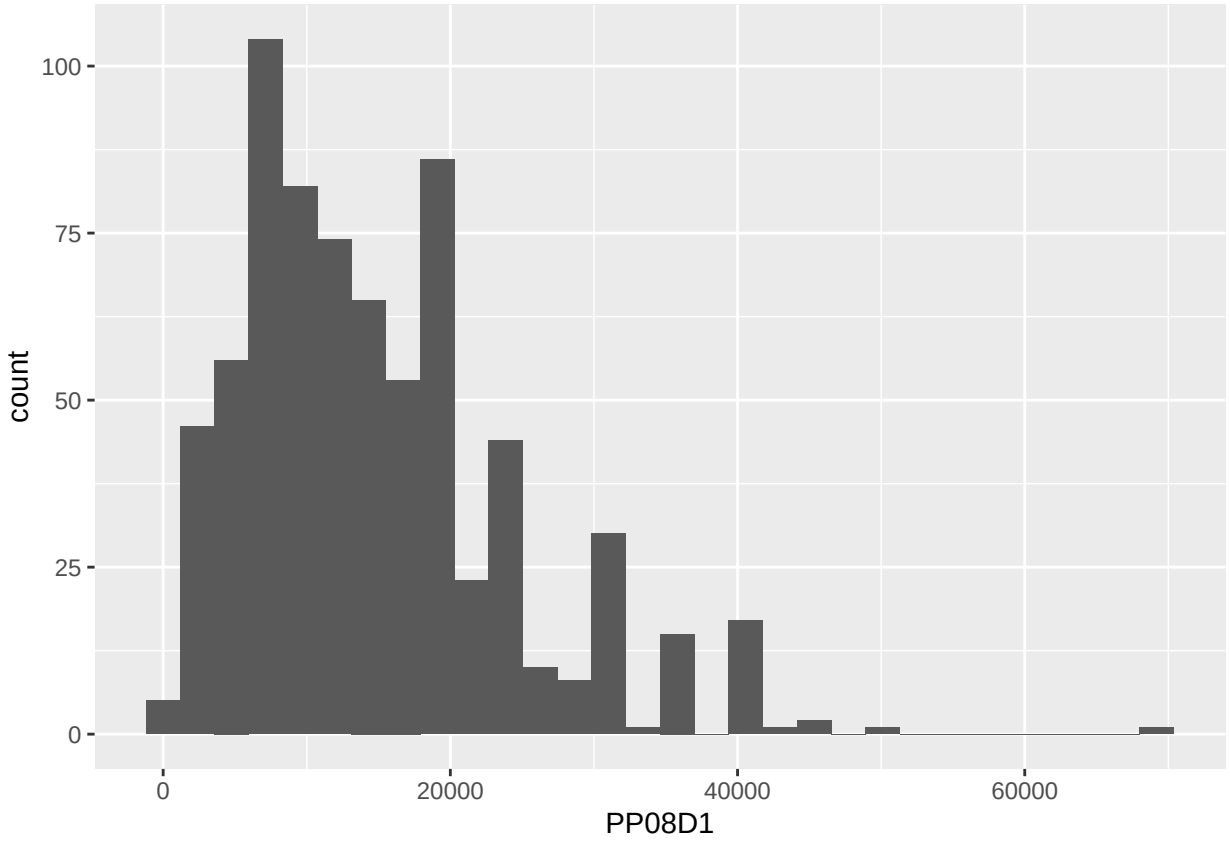
```
ggplot(asalariados.con.ingreso.y.horas.cordoba)
```

Que solo muestra la capa base:

Se agrega una capa con un histograma:

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

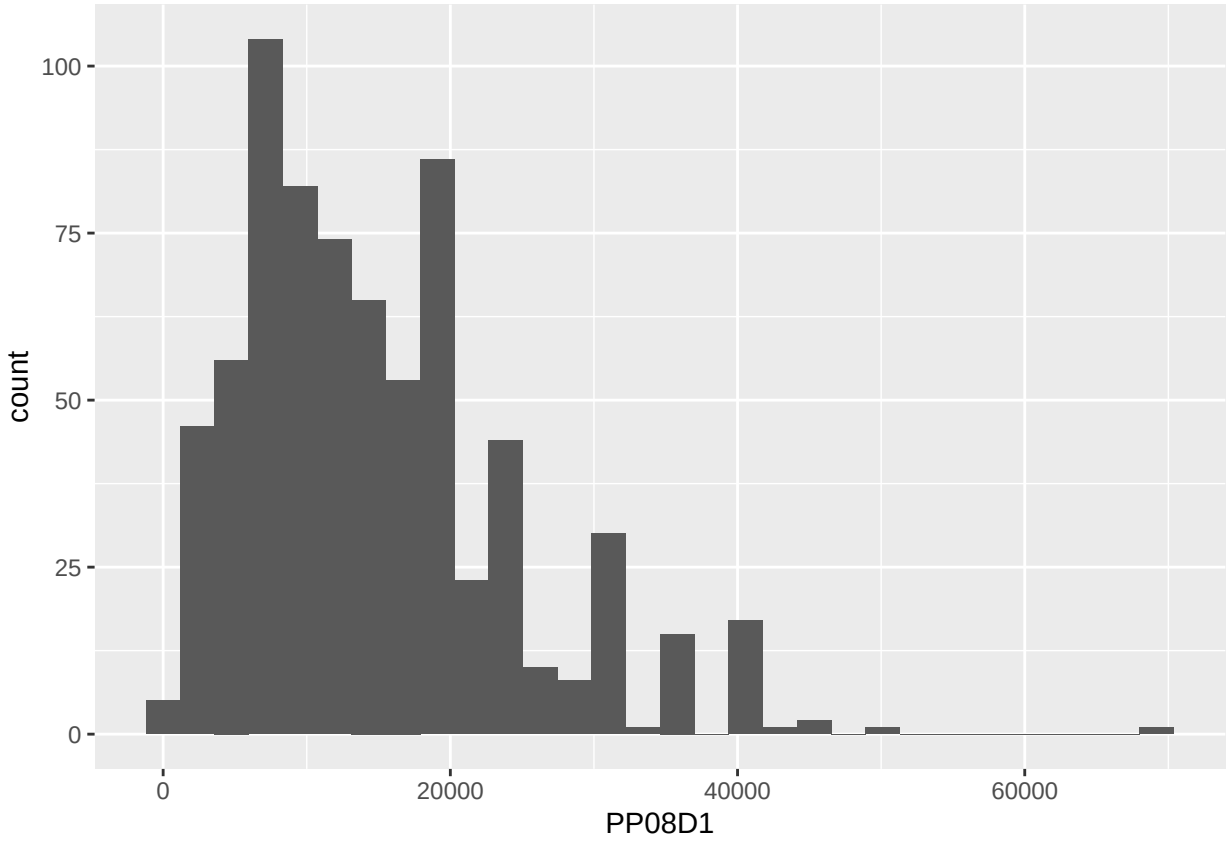
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



El mismo resultado se logra ubicando la estética en la primera instrucción:

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

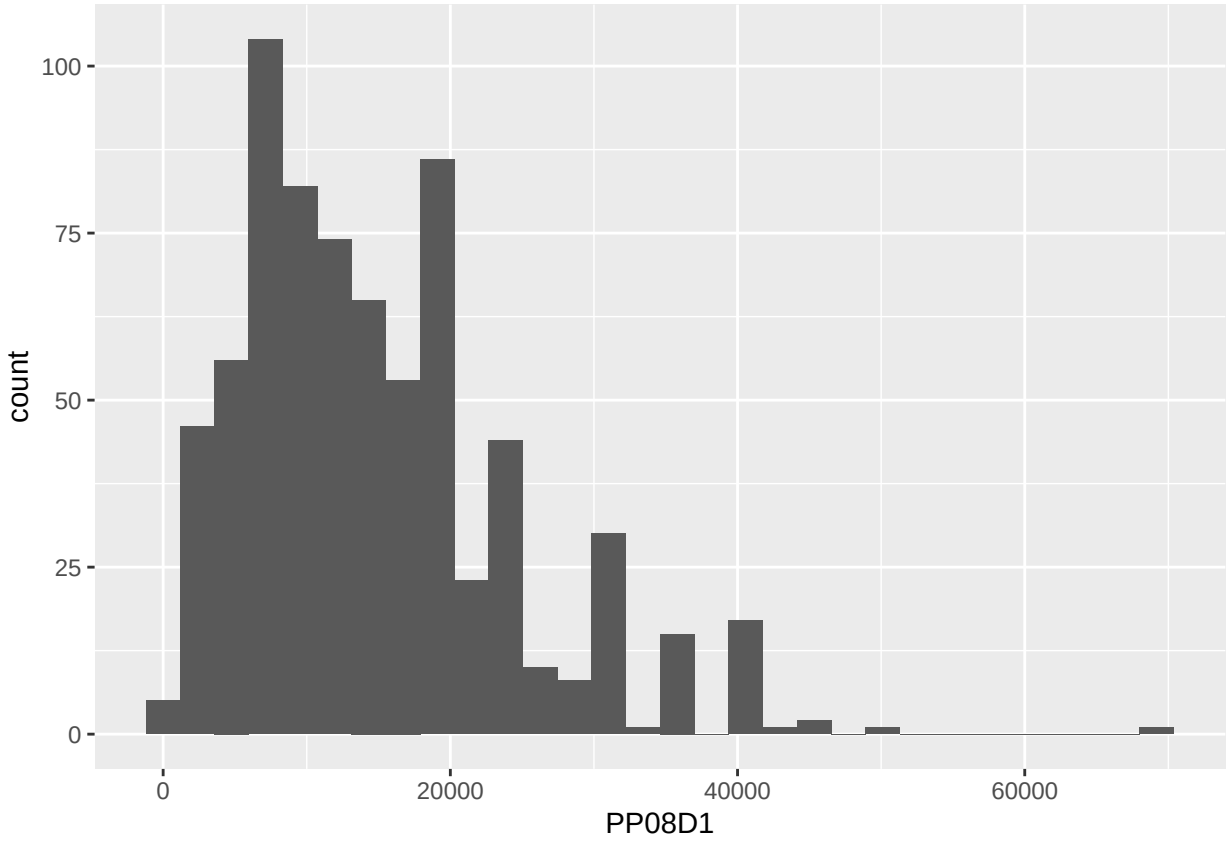
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



O poniendo todo en la capa del histograma

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

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

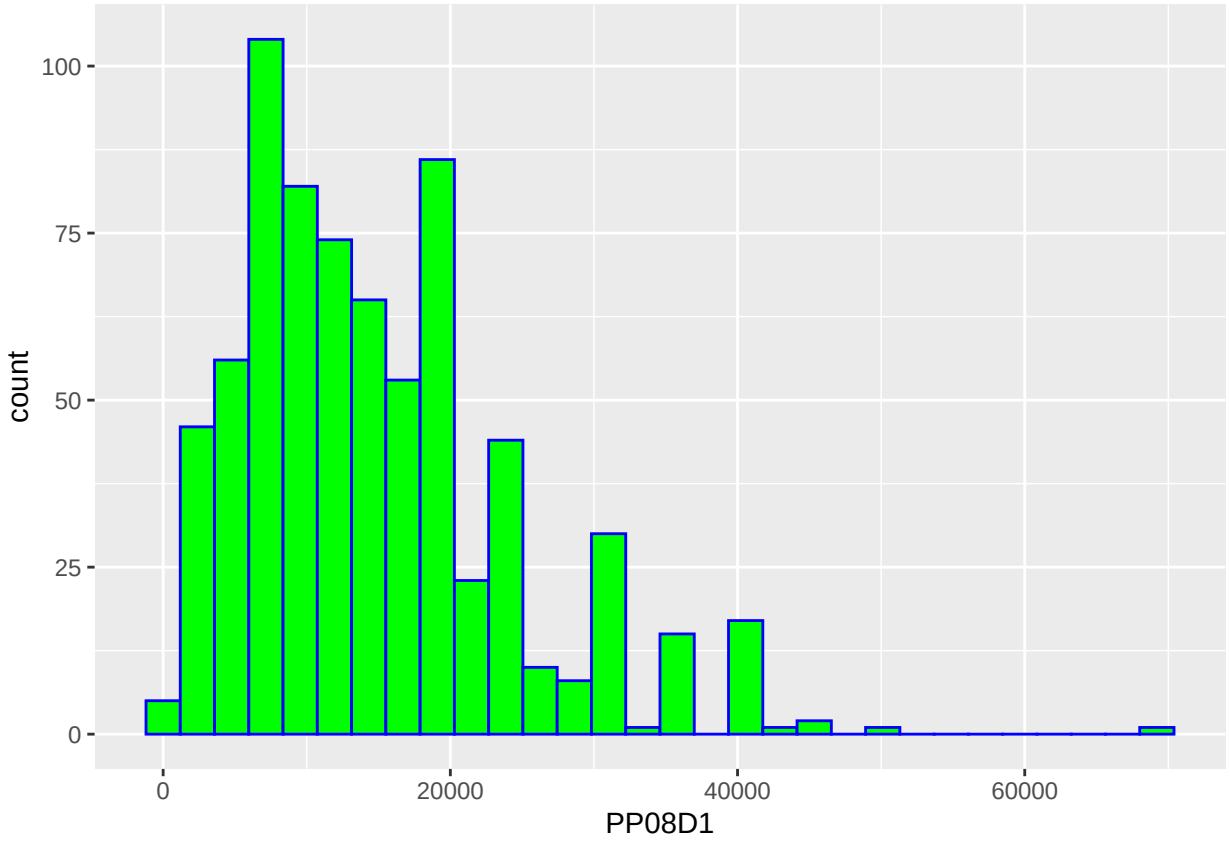


Solo que así debemos indicar que `asalariados.con.ingreso.y.horas.cordoba` son los datos.
La información que vaya en la intrucción ggplot() será válida para todas las capas que se agreguen, la que se incluya en una capa solo se toma para esa capa.

Se lo puede pintar de verde, con contornos azules:

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

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

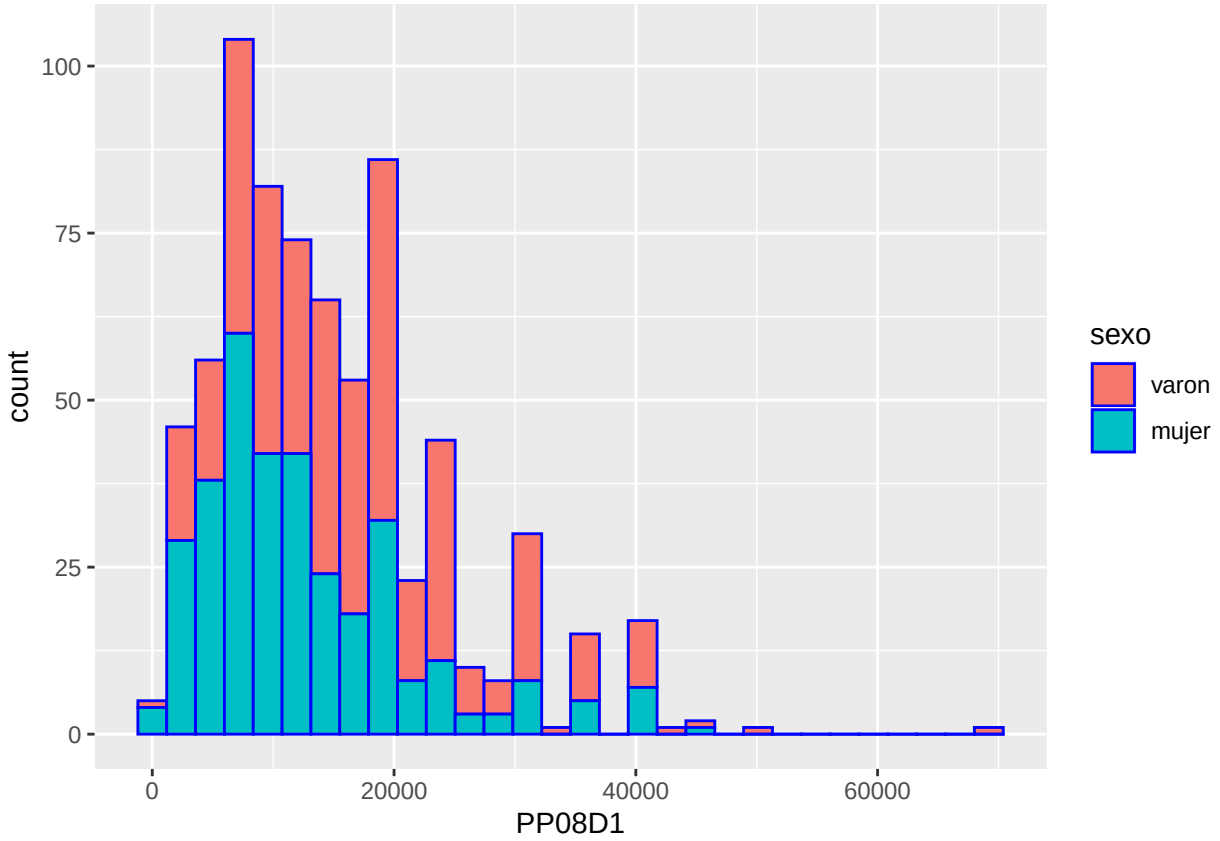


- `fill` es para el relleno de las barras
- `col` es para el contorno

En ese ejemplo, 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")
```

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



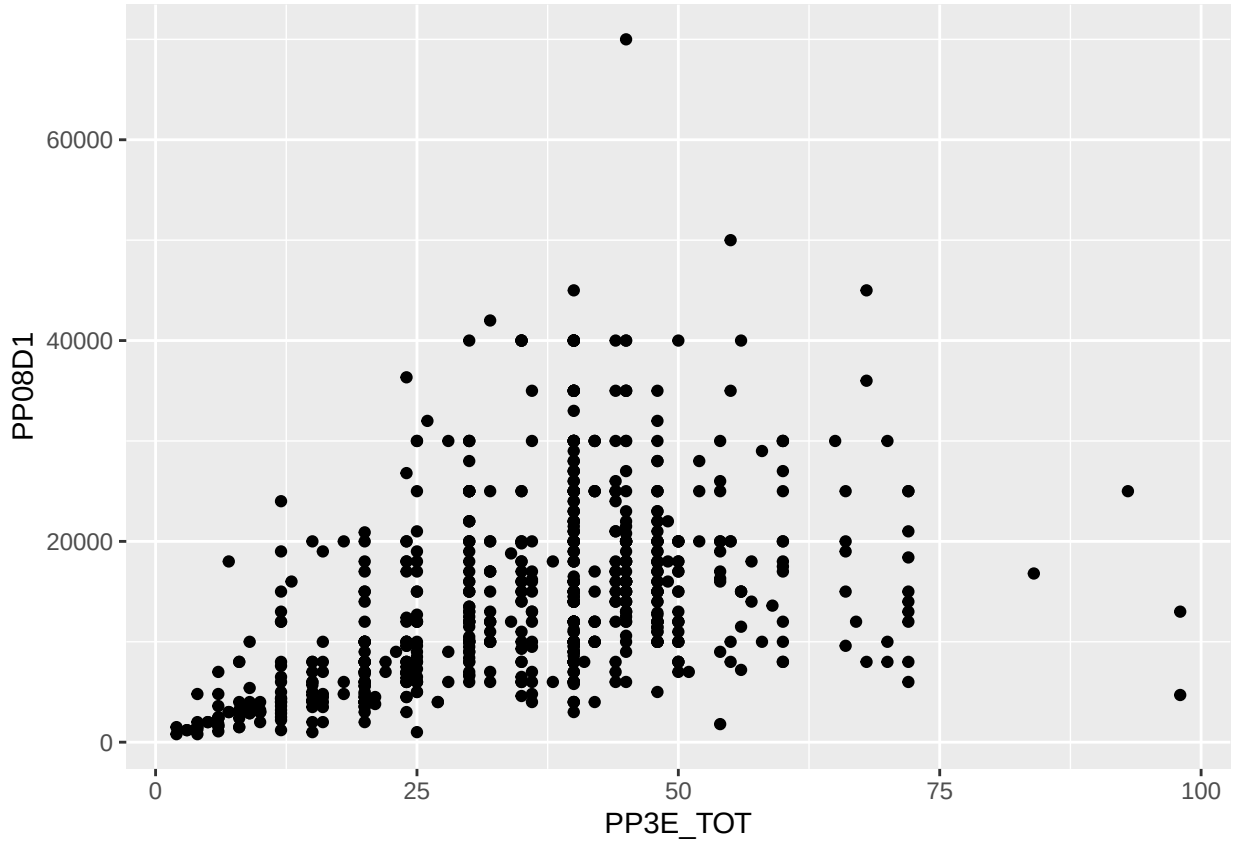
Los contornos están **fijados**, pero el relleno está **mapeado**.

Mapear es vincular valores de una variable a atributos estéticos del gráfico, como el color, la forma, o el tamaño, según qué gráfico sea. **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.

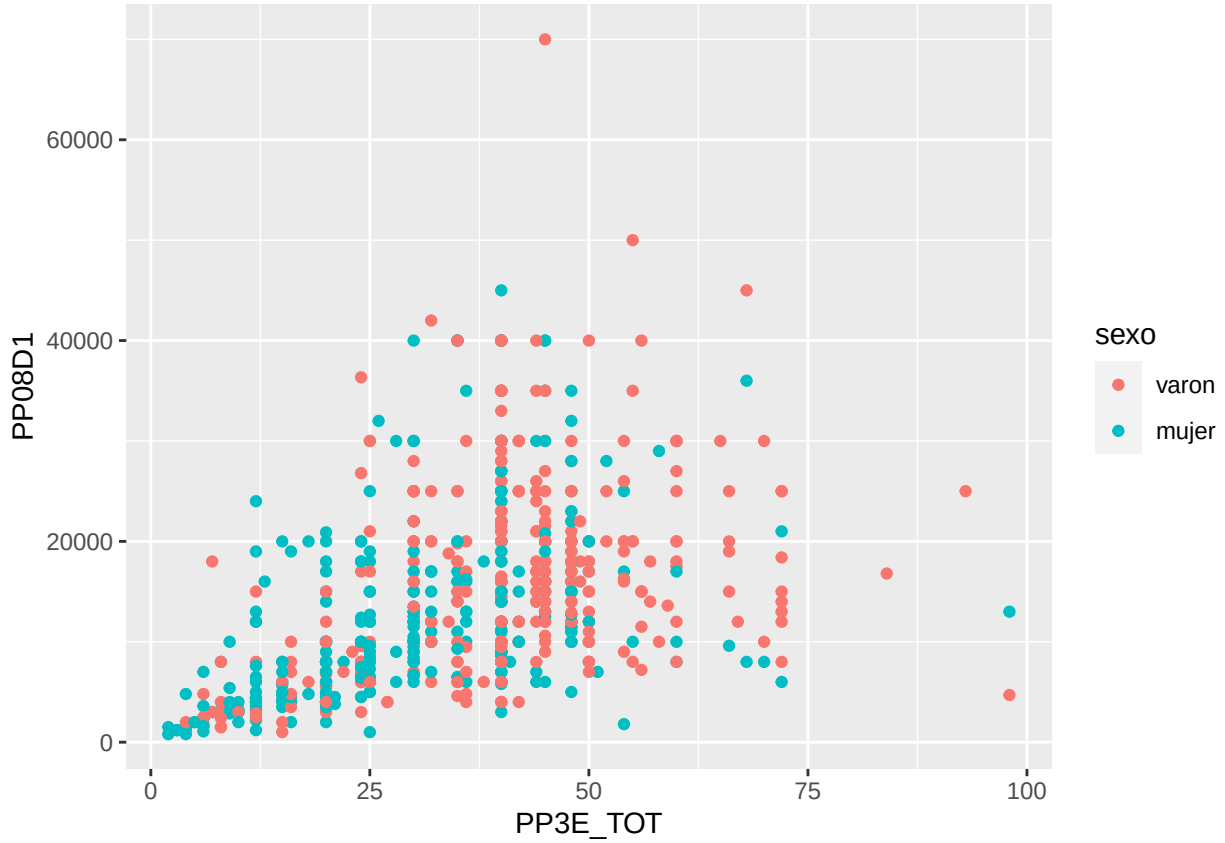
Si se trata de dos variables cuantitativas, como las horas y los ingresos, la capa para el diagrama de dispersión se llama `geom_point` y en la estética deben indicarse las dos variables en el orden x, y:

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



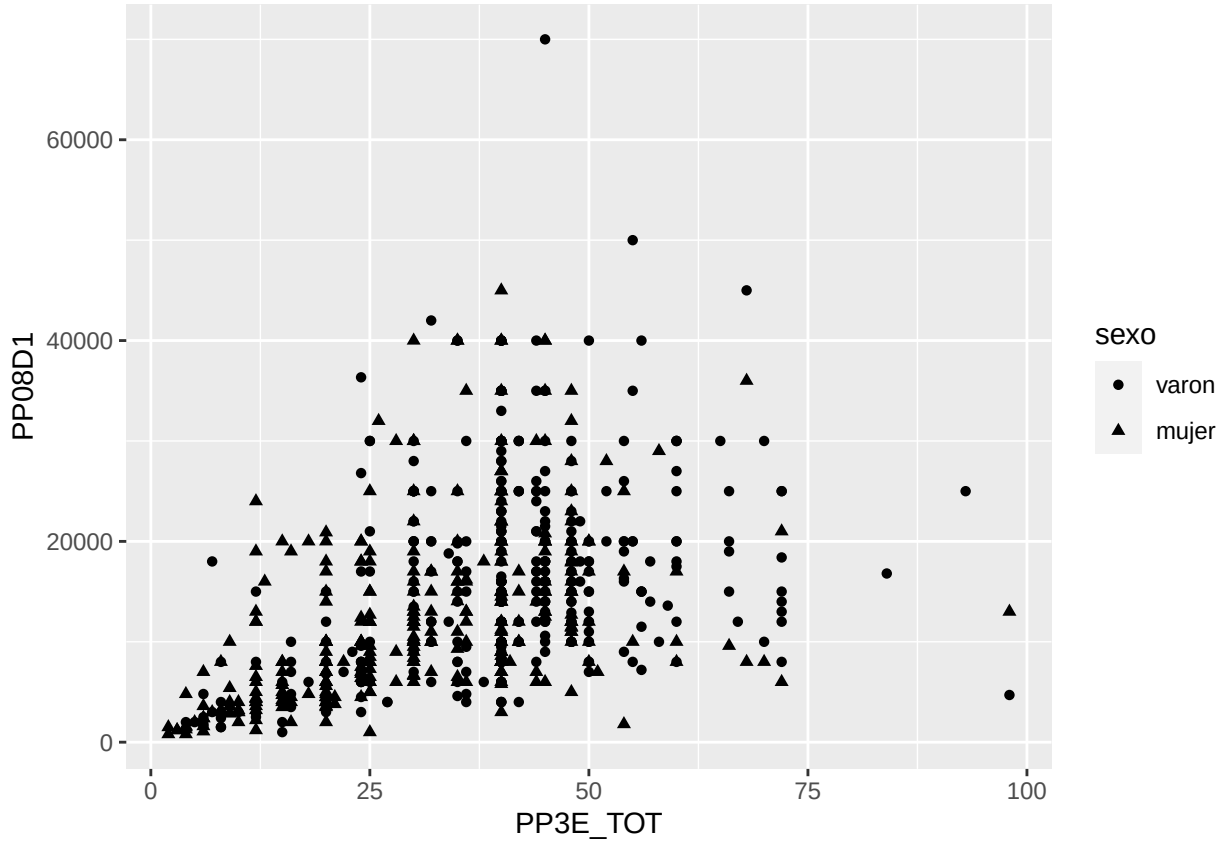
El mapeo de la variable `sexo` al color de los puntos, se pide dentro de la estética:

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



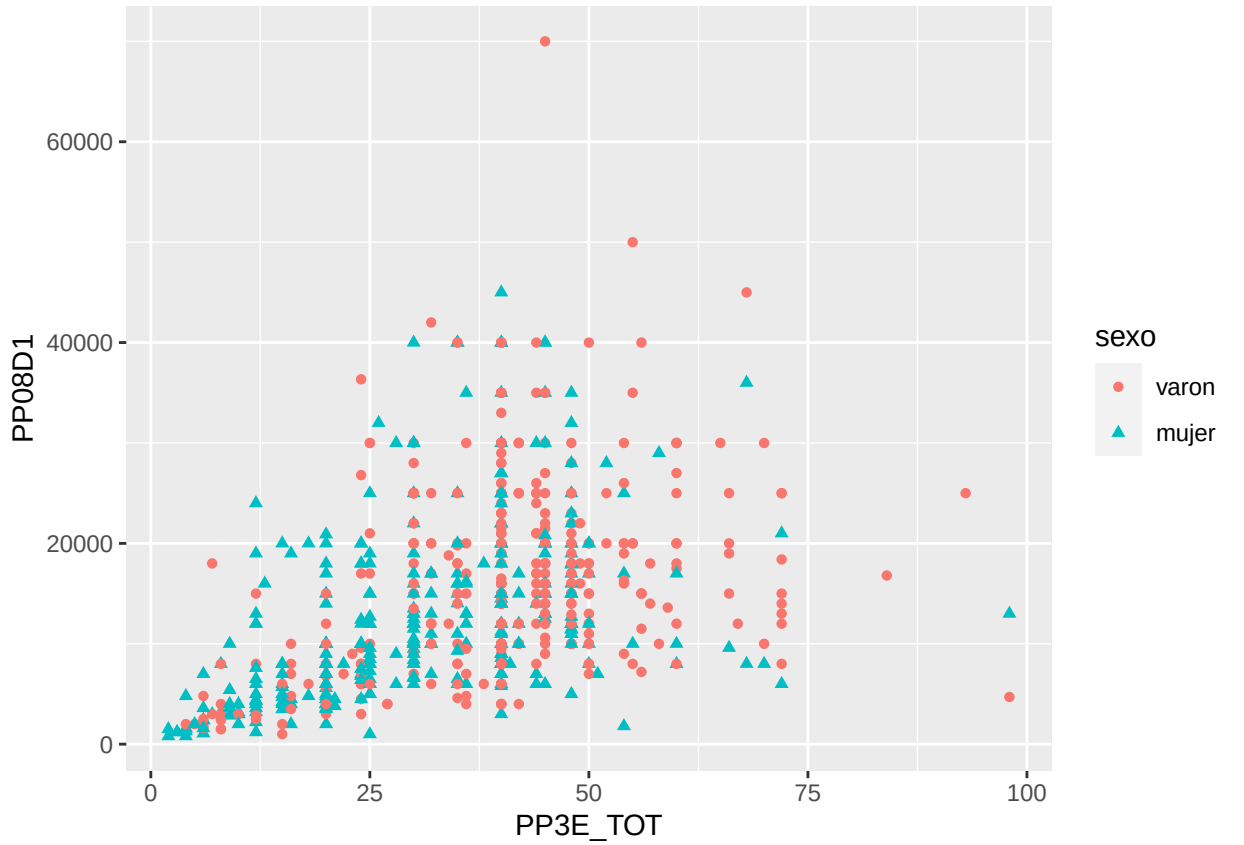
En lugar del color se puede elegir la forma de los puntos:

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



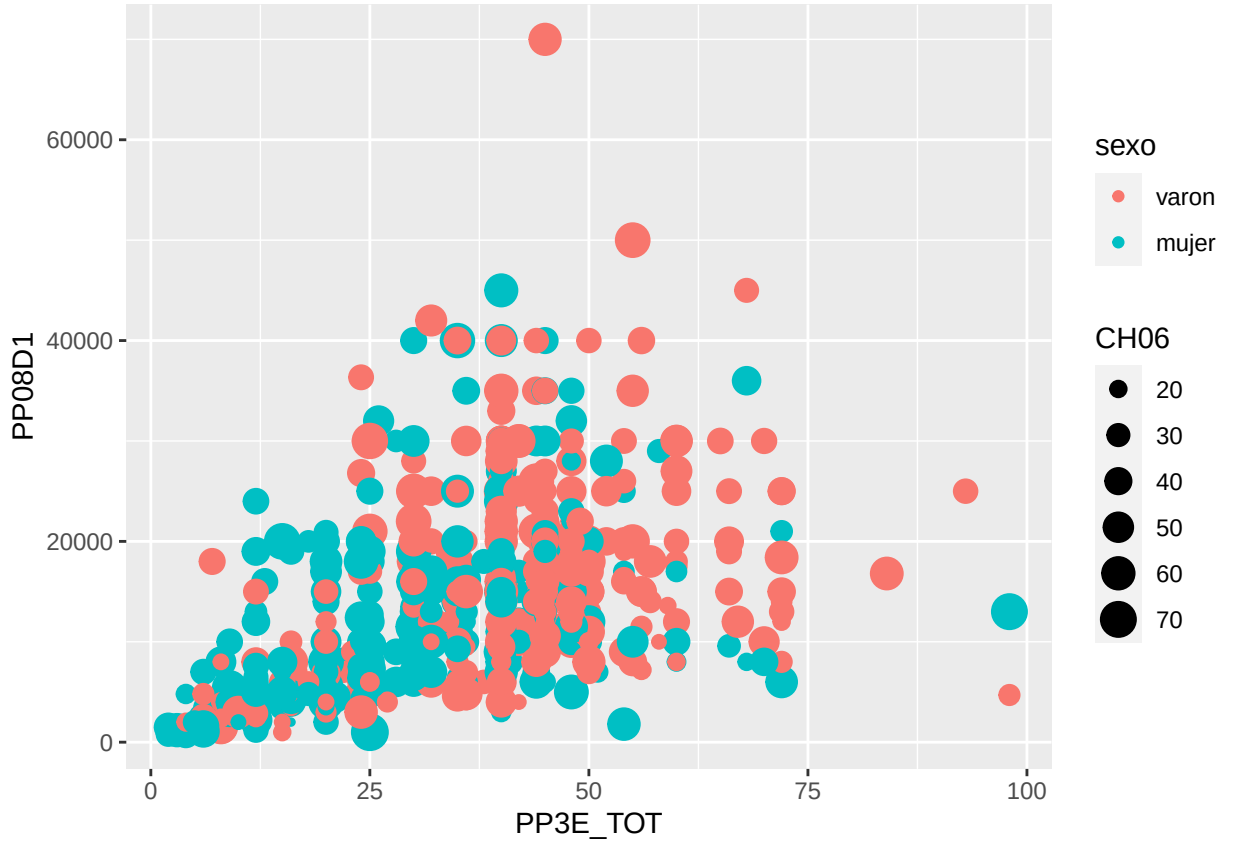
O ambos atributos gráficos a la misma variable

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



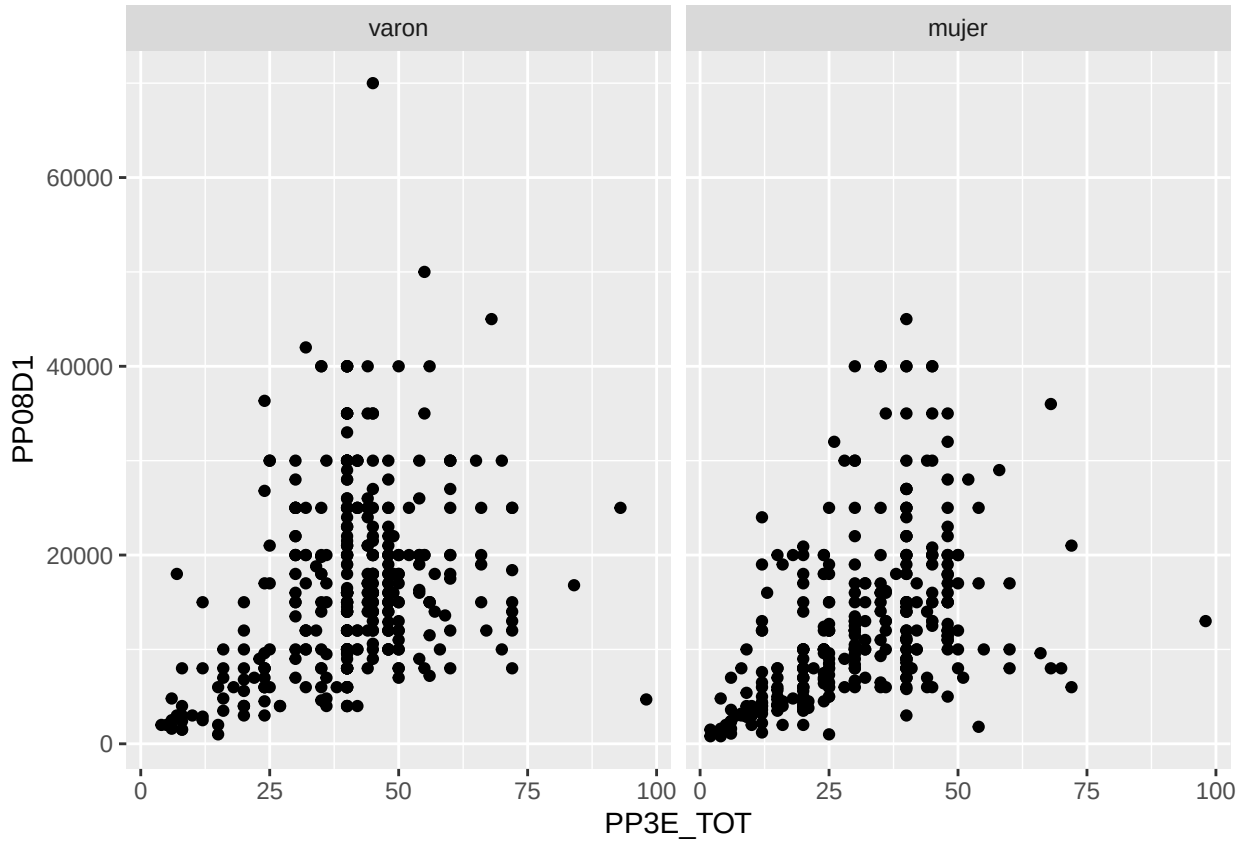
VER COMO SE ILUSTRA ESTO MEJOR El tamaño de los puntos puede mapearse a otra variable cuantitativa así se dibujan puntos cuyo tamaño es proporcional a los valores de la variable edad (CH06).

```
ggplot(
  asalariados.con.ingreso.y.horas.cordoba)+
  geom_point(aes(PP3E_TOT, PP08D1, col=sexo, size=CH06))
```



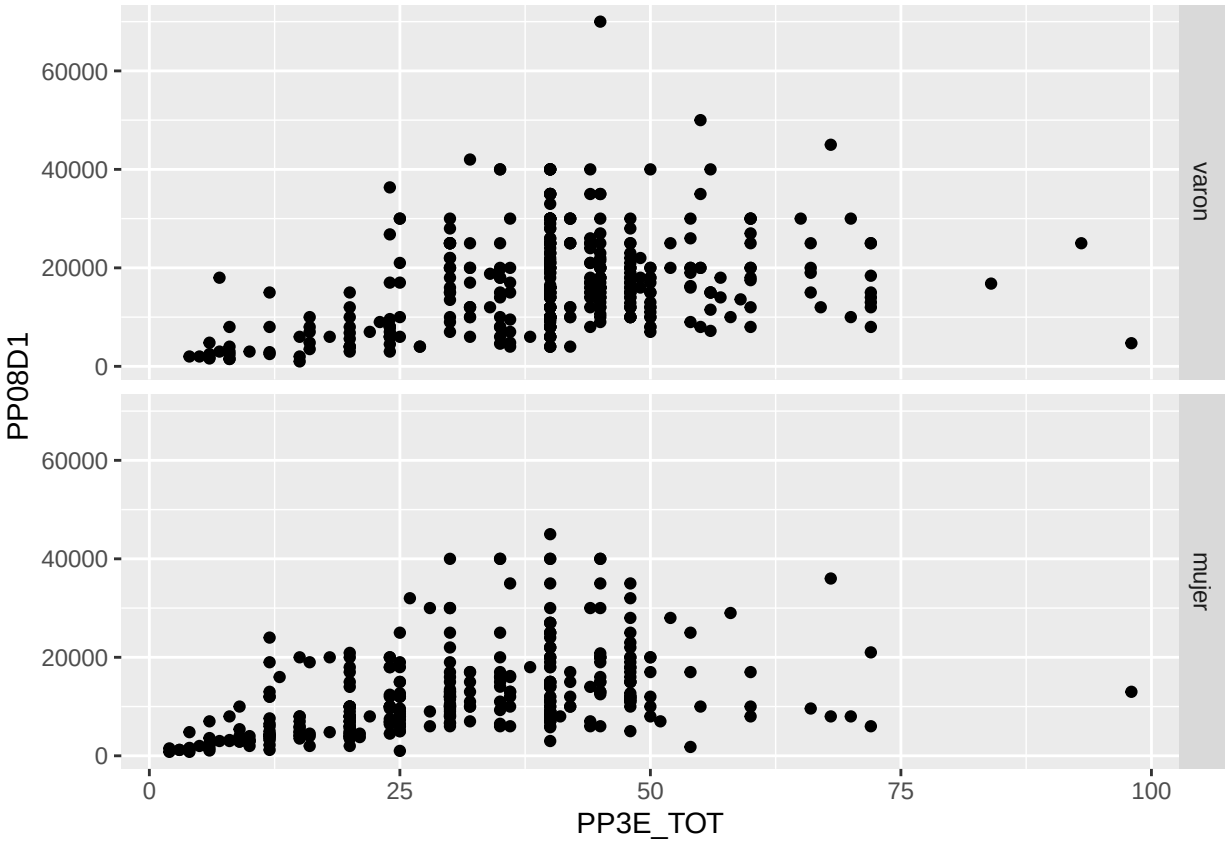
Una opción para comparar grupos es la capa `facet_grid` que puede agregarse a cualquier tipo de gráfico. El comando tiene dos argumentos que corresponden a dos variables de clase `factor`, que se separaran con `~` (alt+126 en windows) que van a separar el gráfico pedido en tantas filas y columnas como 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

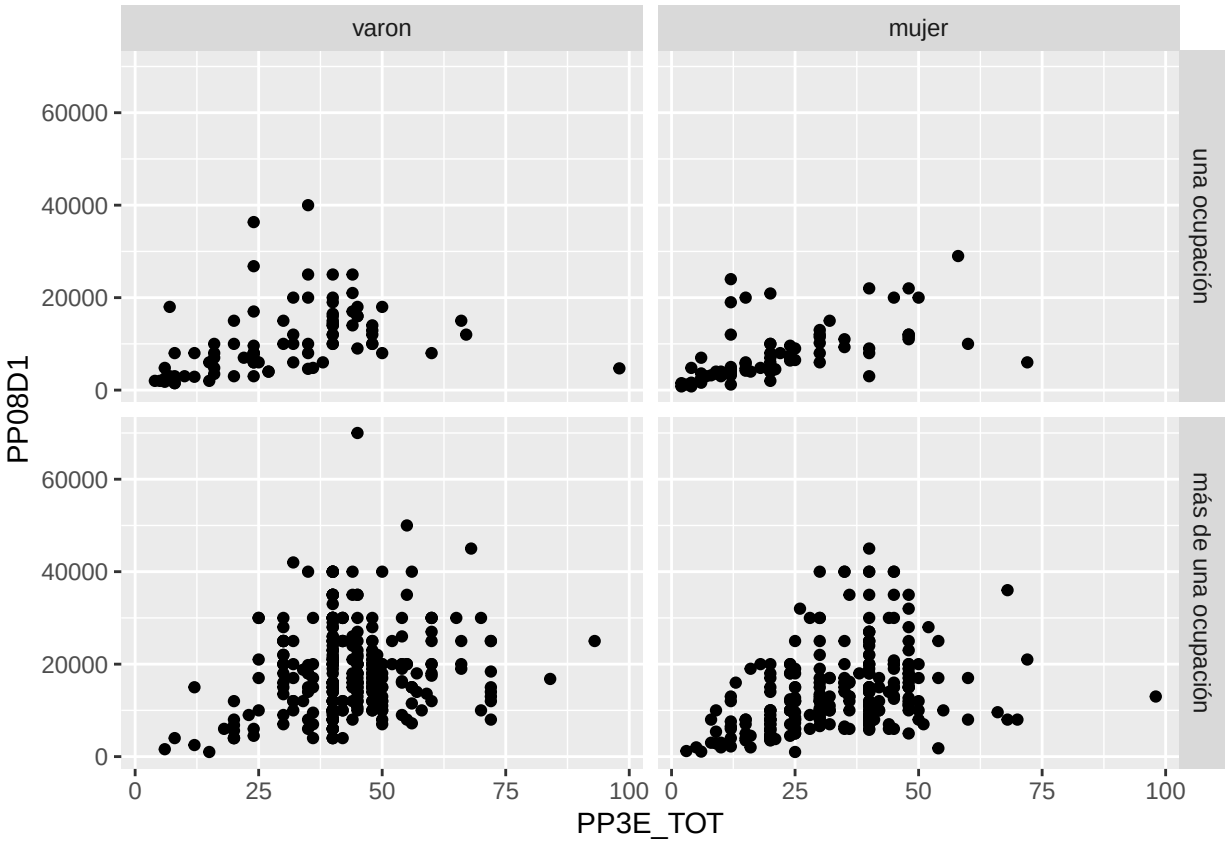
```
ggplot(
  asalariados.con.ingreso.y.horas.cordoba)+
  geom_point(aes(PP3E_TOT, PP08D1))+
  facet_grid(sexo~.)
```



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, la ajustaremos en su nombre y categorías.

```
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")
```

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

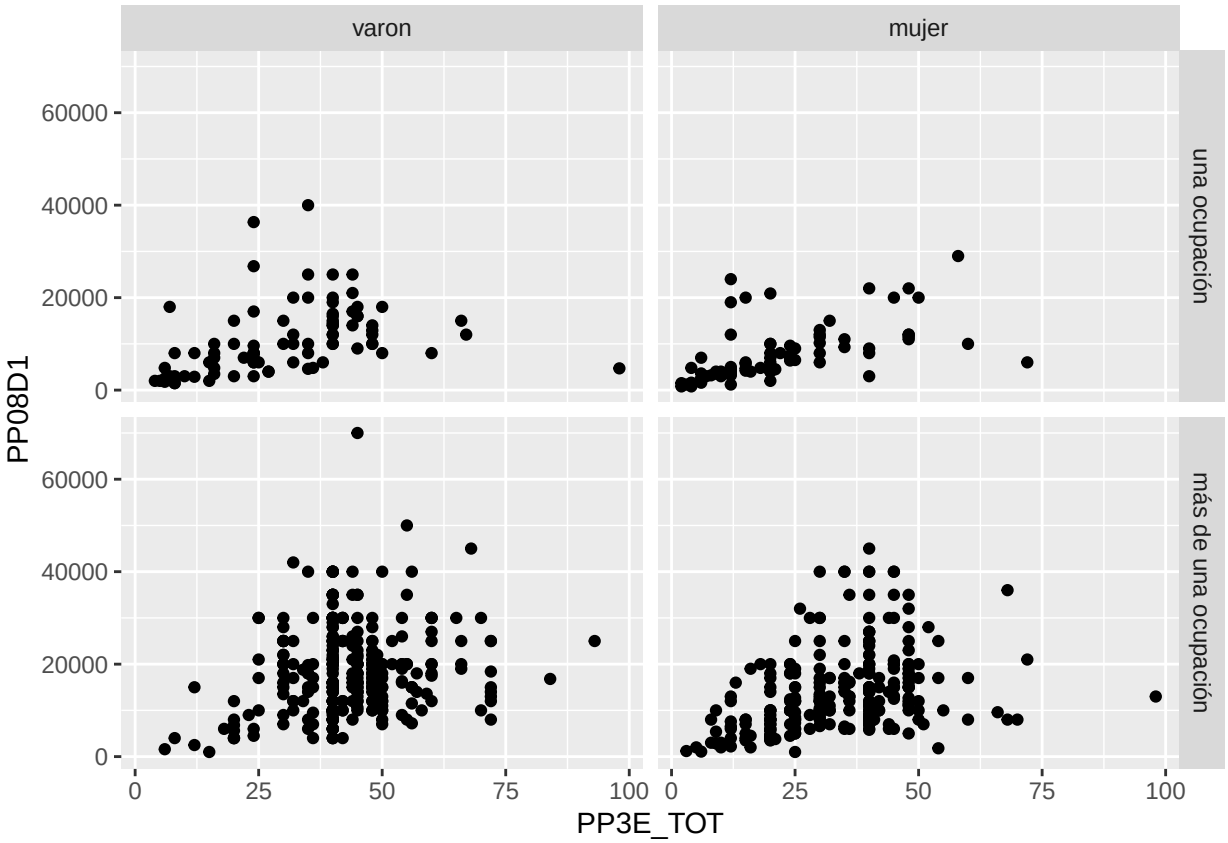


Para no repetir las instrucciones que generan las primera capas del gráfico (aunque se lo haga copiando y pegando), vamos a guardarlo como un objeto:

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

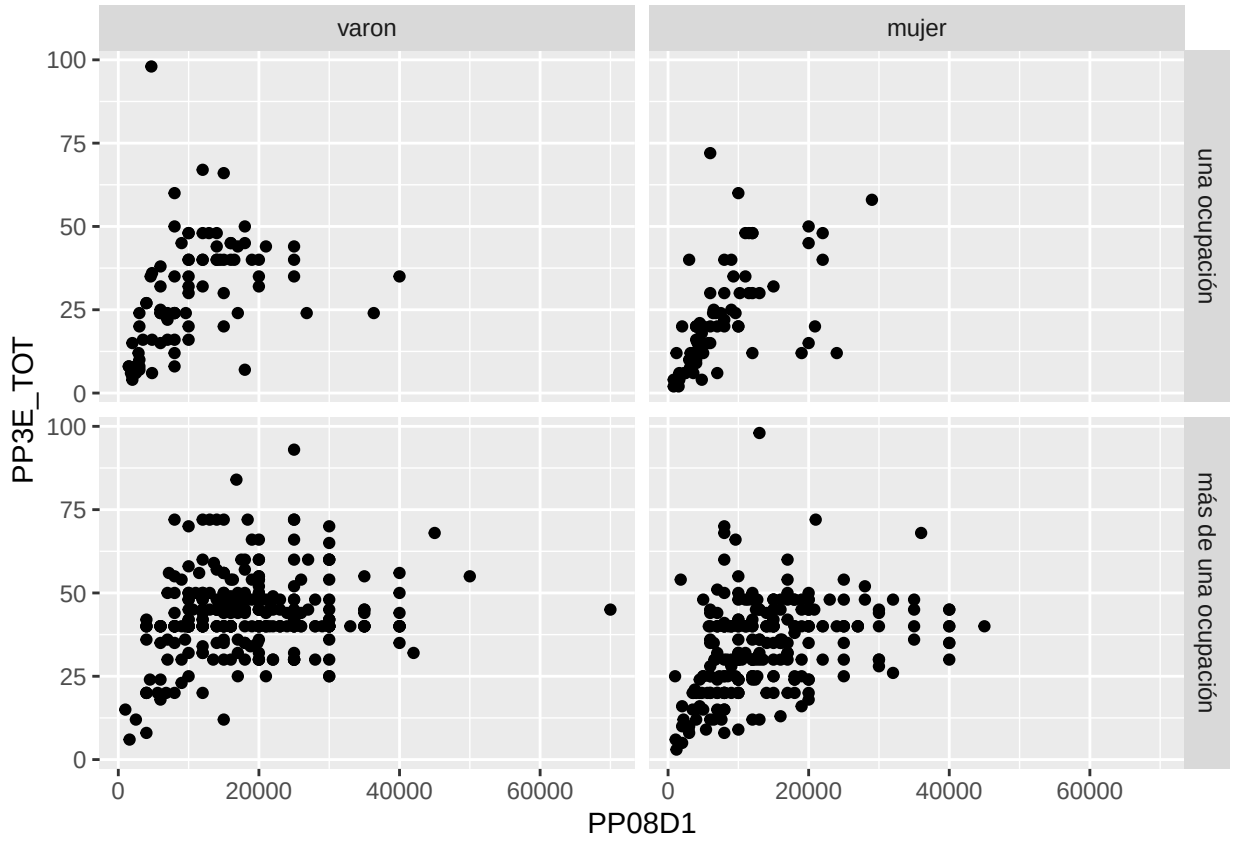
Para que aparezca solo se lo llama:

```
p1
```



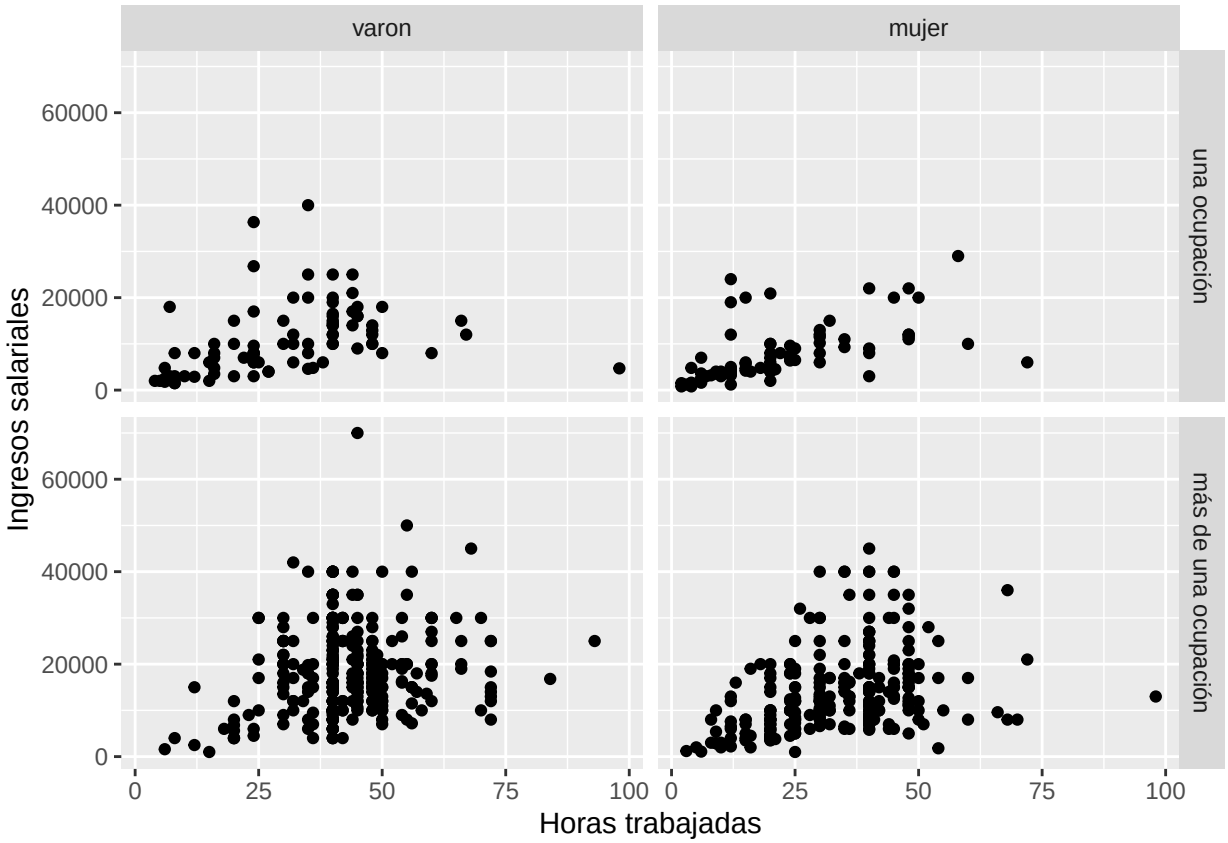
El cambio en la posición de los ejes es `coord_flip`

```
p1+coord_flip()
```



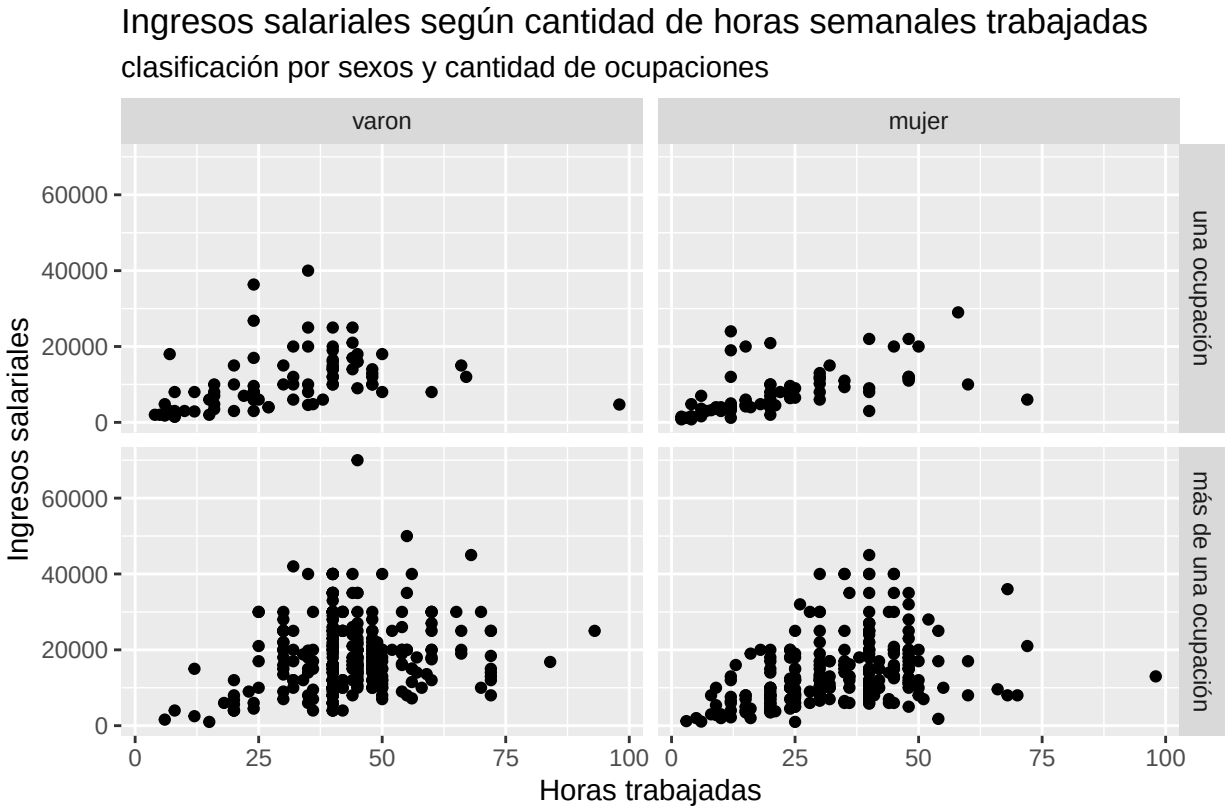
Ahora se pueden seguir agregando capas a p1. Los nombres de los ejes se ajustan con capas de etiquetas:

```
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="clasificació
       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:

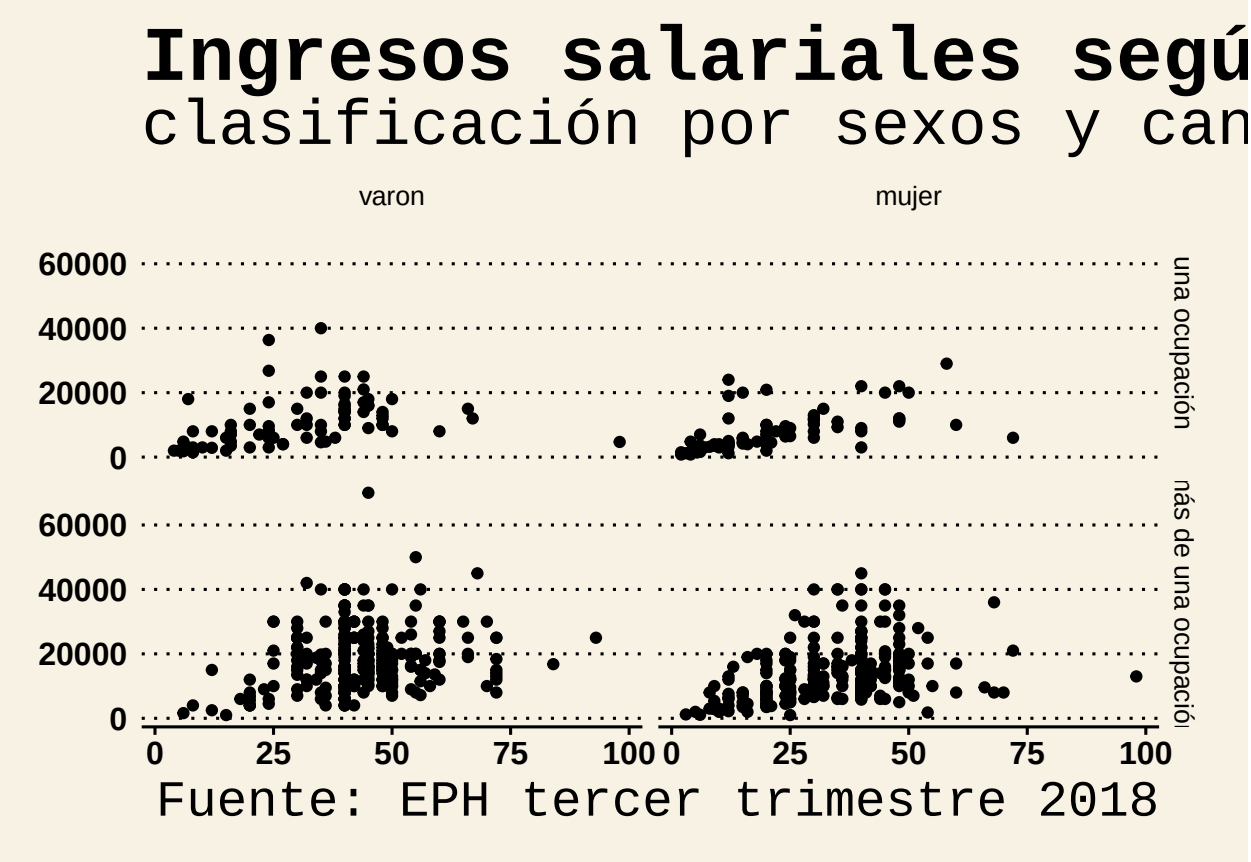
```
p2<-p1+ xlab("Horas trabajadas")+
  ylab("Ingresos salariales")+
  labs(title="Ingresos salariales según cantidad de horas semanales trabajadas", subtitle="clasificació
       caption="Fuente: EPH tercer trimestre 2018")
```

Y probemos alguna capas de temas, para lo que tenemos que cargarlo en la sesión

```
library(ggthemes)
```

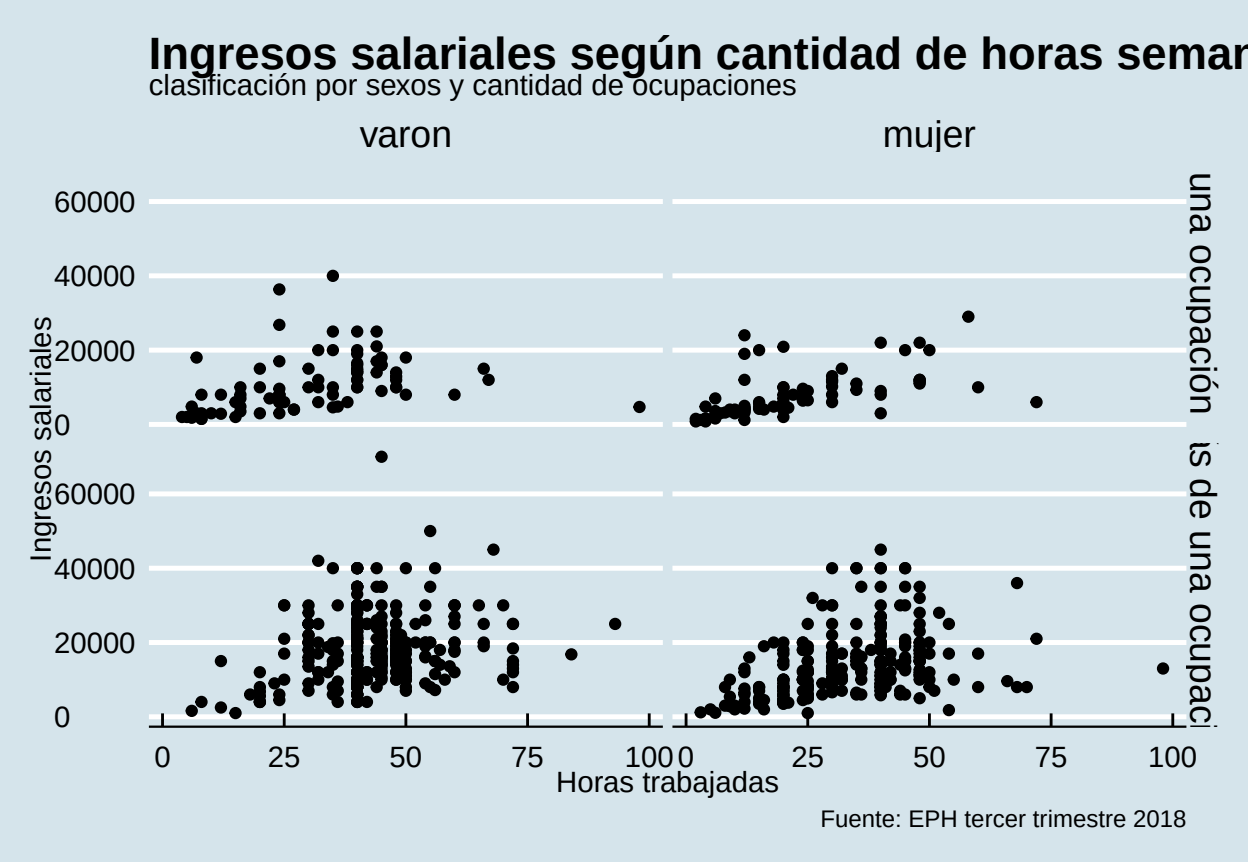
Wall Street Journal

```
p2+theme_wsj()
```



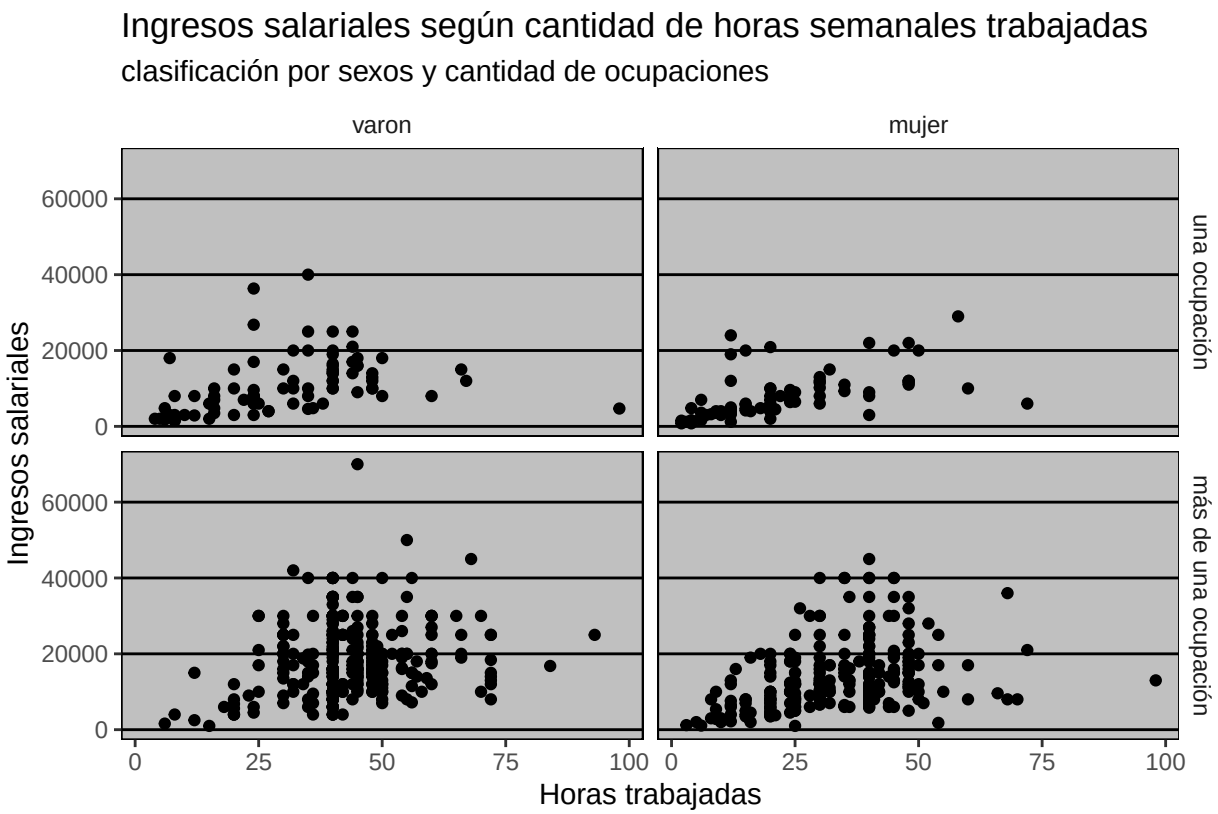
The Economist

```
p2+theme_economist()
```



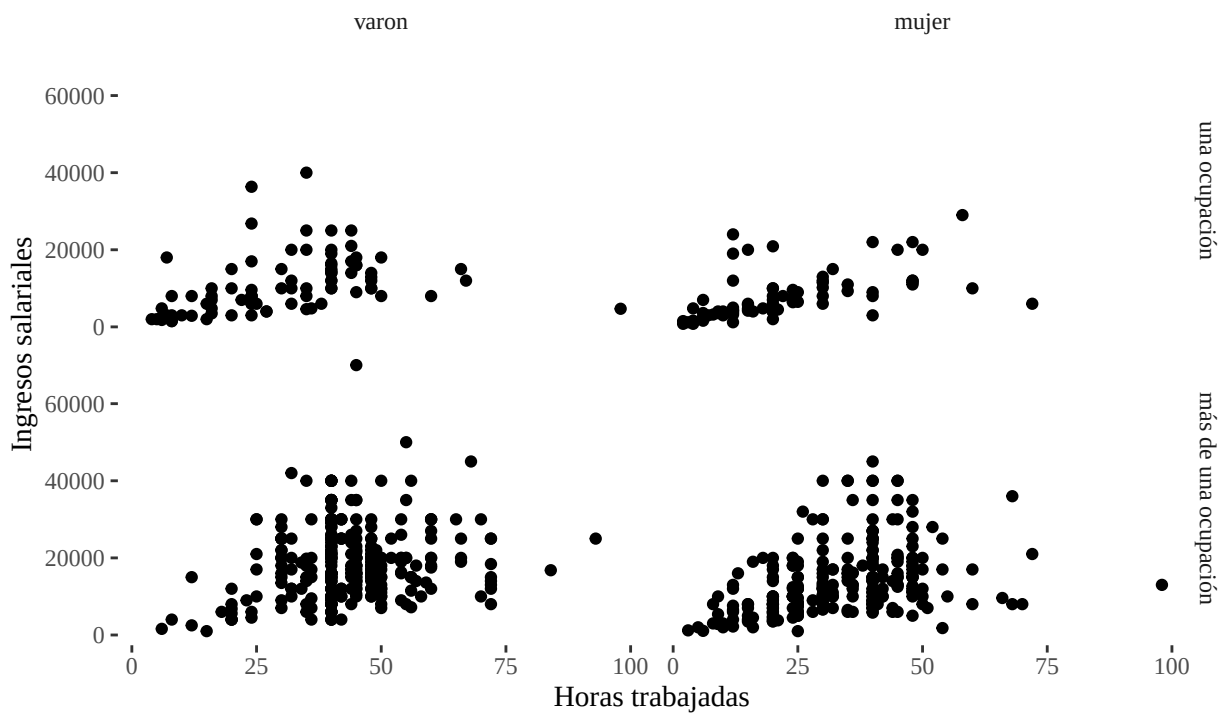
Excel

```
p2+theme_excel()
```



La propuesta de Edward Tufte

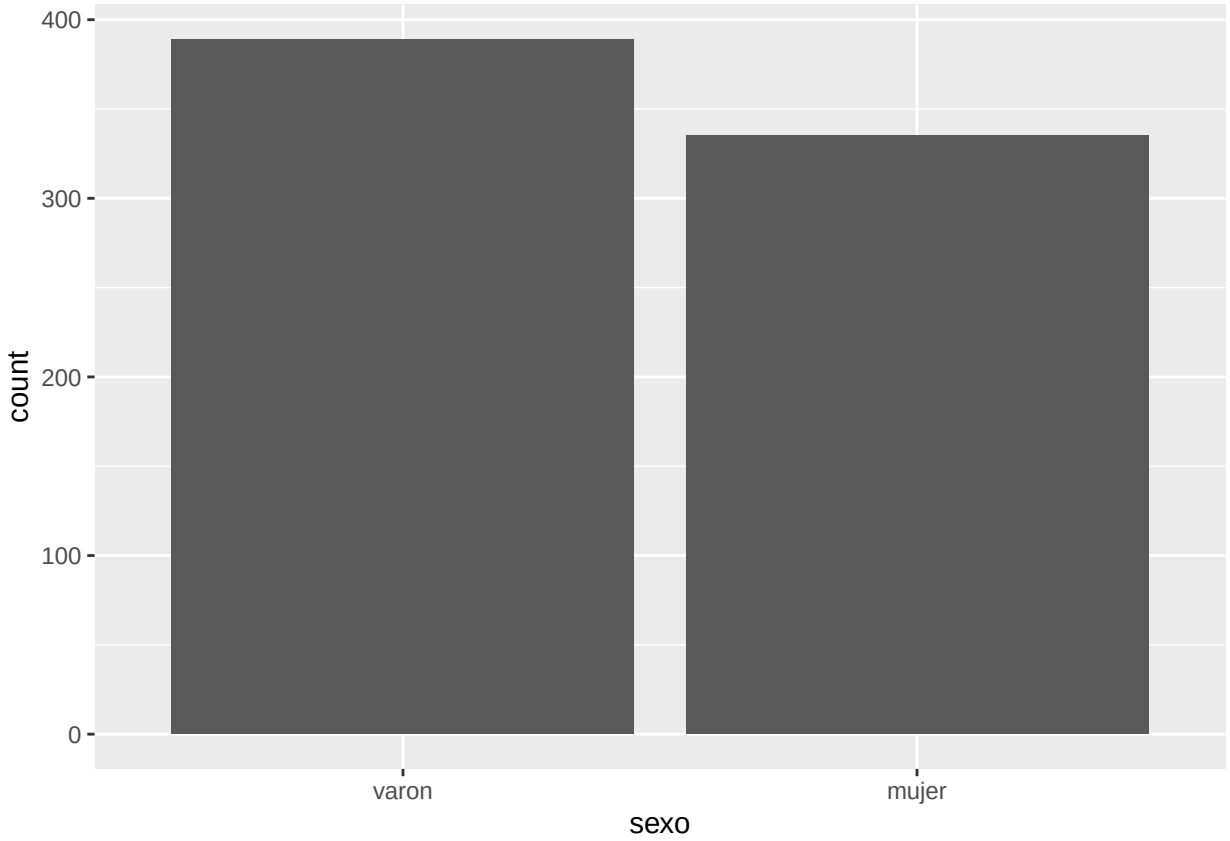
```
p2+theme_tufte()
```



Fuente: EPH tercer trimestre 2018

Para variables categóricas empezamos con un gráfico de barras:

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


400
300
200
100
0
count
varon
mujer
sexo